

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ
ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего
профессионального образования

**«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ТОМСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**

Ю.Б.Буркатовская

ТЕОРИЯ ГРАФОВ. ЧАСТЬ 1

Издательство

Томского политехнического университета

2014

УДК 519.17
ББК 22.176
Б914

Буркатовская Ю.Б.

Б914 Теория графов. Часть 1: учебное пособие / Ю.Б.Буркатовская;
Томский политехнический университет. – Томск: Изд-во Томского
политехнического университета, 2014. – 200 с.

ISBN

Пособие состоит из двух частей. В данной первой части рассмотрены основные понятия теории графов, связность графов. Поставлены оптимизационные задачи теории графов: задачи поиска оптимальных путей и задачи размещения, приведены алгоритмы из решения. Рассмотрен особый вид графа – деревья и связанные с ними задачи: поиск кратчайшего остовного дерева и поиск максимального ориентированного леса, а также применение деревьев для хранения информации. Во второй части будут рассмотрены сети, паросочетания и покрытия, эйлеровы и гамильтоновы графы, планарность графов и раскраска графов, а также задачи оптимизации, связанные с этими разделами теории графов.

Пособие основано на курсе лекций по теории графов, который в течение ряда лет читается автором для студентов элитного технического образования Томского политехнического университета, и предназначено для студентов, изучающих расширенный курс теории графов.

УДК 519.17
ББК 22.176

Рецензенты

Доктор технических наук, профессор кафедры программирования
ТГУ

А.Ю.Матросова

Кандидат технических наук, доцент кафедры автоматизации
обработки информации ТУСУР

Т.О.Перемитина

Кандидат технических наук, доцент кафедры оптимизации систем
управления ТПУ

Е.С.Чердынцев

ISBN ©Томский политехнический университет, 2014
©Буркатовская Ю.Б., 2014
©Оформление. Издательство Томского
политехнического университета, 2014

Содержание

Введение	3
1 Основные понятия теории графов	5
1.1 Определение графов и родственных объектов	5
1.2 Смежность вершин и ребер	6
1.3 Подграфы	8
1.4 Типы графов	9
1.5 Изоморфизм графов	10
1.6 Операции над графами	11
1.7 Способы задания графов	12
1.8 Упражнения	14
2 Связность графов	15
2.1 Маршруты, цепи, циклы	15
2.2 Расстояния в графе	16
2.3 Связность неориентированных графов	17
2.4 Оценка числа ребер в графе	19
2.5 Деревья	20
2.6 Система фундаментальных циклов	21
2.7 Система фундаментальных разрезов	27
2.8 Теорема Менгера	33
2.9 Связность орграфов	36
2.10 Упражнения	42
3 Поиск путей в графе	44
3.1 Стратегии обхода графа	44
3.2 Поиск кратчайших путей	45
3.3 Поиск минимальных путей	48
3.3.1 Поиск минимальных путей от заданной вершины	48
3.3.2 Поиск минимальных путей между всеми парами вершин	64
3.4 Поиск k минимальных путей	71
3.4.1 Пространство R_k	72
3.4.2 Поиск k произвольных минимальных путей от заданной вершины	74
3.4.3 Поиск k произвольных минимальных путей между всеми парами вершин	82
3.4.4 Поиск k простых минимальных путей от заданной вершины	89
3.5 Поиск путей с заданными свойствами	95
3.6 Упражнения	99
4 Задачи размещения	101
4.1 Расстояния во взвешенном графе	101
4.2 Поиск центров графа	112
4.3 Поиск медиан графа	118
4.4 Обобщения задач размещения	122

4.5	Поиск кратных центров	124
4.6	Поиск кратных медиан	130
4.7	Упражнения	150
5	Деревья	150
5.1	Свободные деревья	150
5.2	Задача о соединении городов	152
5.3	Ориентированные, упорядоченные и бинарные деревья	154
5.4	Построение ориентированного дерева и леса	158
5.5	Способы задания деревьев	164
5.6	Деревья сортировки	166
5.6.1	Выровненные деревья	169
5.6.2	Сбалансированные деревья	170
5.6.3	Красно-черные деревья	173
5.7	Упражнения	189
	Приложение. Поиск покрытий булевой матрицы	190
	П1. Поиск всех безызбыточных покрытий	191
	П2. Поиск минимальных и кратчайших покрытий с использованием КНФ функции покрытия	194
	П3. Поиск минимальных и кратчайших покрытий методом ветвей и границ	196
	Список литературы	200

Введение

Теория графов – это область дискретной математики, особенностью которой является геометрический подход к изучению объектов и связей между ними. Объекты называются вершинами графа, связи между парами объектов – ребрами. Всякий раз, когда мы хотим представить себе взаимосвязь пар различных объектов, мы имеем дело с графом. Например, с помощью графа можно представить схему перелетов определенной авиакомпании, где вершинами графа являются города, а ребрами – рейсы, соединяющие пары городов. Дерево каталогов также является графом: диски, папки и файлы являются вершинами, а ребра показывают вложенность файлов и папок в папки и диски. Молекулы также изображаются в виде графов, где вершинами являются атомы, а ребрами – связи между ними.

Первые задачи теории графов были связаны с решением головоломок (задача о Кенигсбергских мостах, задача о расстановке ферзей на шахматной доске, задачи о перевозках, задача о кругосветном путешествии и др.). Одним из первых результатов в теории графов явился критерий существования обхода всех ребер графа без повторений, полученный Л. Эйлером при решении задачи о Кенигсбергских мостах. Вот пересказ отрывка из письма Эйлера от 13 марта 1736 года: «Мне была предложена задача об острове, расположенном в городе Кенигсберге окруженном рекой, через которую перекинуто 7 мостов. Спрашивается, может ли кто-нибудь непрерывно обойти их, проходя только однажды через каждый мост. И тут же мне было сообщено, что никто еще до сих пор не смог это проделать, но никто и не доказал, что это невозможно. Вопрос этот, хотя и банальный, показался мне, однако, достойным внимания тем, что для его решения недостаточны ни геометрия, ни алгебра, ни комбинаторное искусство. После долгих размышлений я нашел легкое правило, основанное на вполне убедительном доказательстве, с помощью которого можно во всех задачах такого рода тотчас же определить, может ли быть совершен такой обход через какое угодно число и как угодно расположенных мостов или не может».

Теория графов получила свое дальнейшее развитие в XIX веке, когда было найдено множество ее практических приложений к теории электрических сетей, кристаллографии, органической химии и другим наукам. Как отдельная математическая дисциплина теория графов была впервые представлена в работе венгерского математика Кенига в 30-е годы XX столетия. Благодаря развитию кибернетики и вычислительной техники, интерес к теории графов возрос, а проблематика теории графов существенным образом обогатилась. Графы стали использоваться для анализа программ, а также как структуры для хранения информации. Кроме того, использование компьютеров позволило решать возникающие на практике конкретные задачи, связанные с большим объемом вычислений, прежде не поддававшиеся решению. Для ряда экстремальных задач теории графов были разработаны методы их решения, например, один из таких методов позволяет решать задачи о построении максимального потока через сеть. Для отдельных классов графов (деревья, плоские графы и т.д.) было показано, что решения некоторых задач для графов из этих классов находятся проще, чем для произвольных графов (нахождение условий существования графов с заданными свойствами, установление изоморфизма графов и др.).

Графы и связанные с ними методы исследований органически пронизывают на разных уровнях современную математику. Теория графов рассматривается как одна из ветвей топологии; непосредственное отношение она имеет также к алгебре и к теории чисел. Графы эффективно используются в теории планирования и управления, теории расписаний, социологии, математической лингвистике, экономике, биологии, медицине, географии, программировании, теории конечных автоматов, электронике.

Большое значение в теории графов имеют алгоритмические вопросы. Для конечных графов, т. е. для графов с конечным множеством вершин и ребер, как правило, решение задачи может быть получено с помощью полного перебора всех допустимых вариантов. Однако таким способом удается решить задачу только для графов с небольшим числом вершин и ребер. Поэтому существенное значение для теории графов имеет построение эффективных алгоритмов, находящих точное решение. Для некоторых задач такие алгоритмы построены, например, для поиска оптимальных путей, определения изоморфизма деревьев, нахождения максимального потока. Точное решение других задач, например, задачи коммивояжера или поиска кратного центра, требует перебора. Для подобных проблем требуются качественные алгоритмы поиска приближенного решения, или методы сокращения перебора.

В первом разделе пособия рассмотрены основные понятия теории графов, введены характеристики графа и его вершин, рассмотрены виды графов и подграфов и изоморфизм графов, а также операции над графами и способы их задания.

Второй раздел посвящен связности графов. В нем изучаются различные виды маршрутов и расстояний в графе. Рассмотрено понятие связности графов и введены ее числовые характеристики. Получена оценка числа ребер в графе через число его вершин и компонент связности. Даны методы построения фундаментальных циклов и разрезов графа. Сформулирована и доказана теорема Менгера о наибольшей мощности разделяющего множества вершин. Отдельно рассмотрена связность ориентированных графов.

Третий раздел связан с классической оптимизационной задачей теории графов – поиском оптимального пути. Рассмотрены алгоритмы обхода графа, поиска кратчайших и минимальных путей во взвешенном графе, поиска минимальных путей между всеми парами вершин, поиска первых k минимальных путей, и поиска путей с заданными свойствами.

В четвертом разделе изучается еще одна оптимизационная задача о размещении пунктов обслуживания, которая сводится к поиску центров и медиан графа, а также его кратных центров и медиан.

Пятый раздел посвящен особому виду графов – деревьям. Изучаются свободные, ориентированные, упорядоченные и бинарные деревья, а также связанные с ними оптимизационные задачи – построение кратчайшего остовного дерева и максимального остовного ориентированного леса. Рассмотрено применение деревьев для хранения информации – различные модификации деревьев сортировки.

Многие переборные задачи теории графов могут быть сведены к поиску покрытий булевой матрицы. Некоторые методы решения этой задачи рассмотрены в приложении. Каждый раздел дополнен упражнениями для закрепления теоретического материала.

1 Основные понятия теории графов

1.1 Определение графов и родственных объектов

Определение. *Простым графом* $G(V, E)$ называется совокупность двух множеств – непустого множества V и множества E неупорядоченных пар различных элементов множества V . Множество V называется множеством *вершин*, множество E называется множеством *ребер*.

$$G(V, E) = \langle V, E \rangle, \quad V \neq \emptyset, \quad E \in V \times V.$$

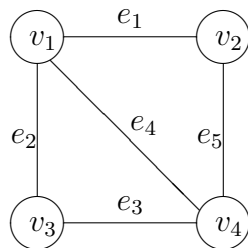
Вместо термина «простой граф» часто употребляется термин «граф». Число вершин графа обозначим через p , число ребер через q .

Вершины графа будем обозначать латинскими буквами (a, b, c, \dots), либо буквой v с указанием номера вершины (v_1, v_2, \dots, v_p). Ребра графа будем обозначать греческими буквами ($\alpha, \beta, \gamma, \dots$), либо буквой e с указанием номера ребра (e_1, e_2, \dots, e_q).

Пример. Пусть $V = \{v_1, v_2, v_3, v_4\}$, $E = \{e_1, e_2, e_3, e_4, e_5\}$, где $e_1 = (v_2, v_1)$, $e_2 = (v_3, v_1)$, $e_3 = (v_4, v_3)$, $e_4 = (v_1, v_4)$, $e_5 = (v_4, v_2)$ тогда $p = 4$, $q = 5$.

Обычно граф изображают *диаграммой*: вершины – точками, ребра – линиями.

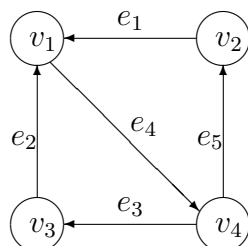
Пример. Рассмотрим граф из предыдущего примера.



Определение. Если элементами множества E являются *упорядоченные* пары (т.е. пары, в которых фиксирован порядок элементов), то граф называется *ориентированным* (или *орграфом*). В этом случае элементы множества V называются *узлами*, а элементы множества E – *дугами*. Первую вершину упорядоченной пары называют *началом дуги*, вторую – *концом*.

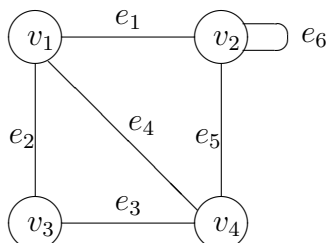
На диаграмме дуги изображаются стрелками, стрелка указывает направление от начала до конца дуги.

Пример. Рассмотрим орграф с теми же множествами вершин и ребер, что и граф из предыдущего примера: $V = \{v_1, v_2, v_3, v_4\}$, $E = \{e_1, e_2, e_3, e_4, e_5\}$, где $e_1 = (v_2, v_1)$, $e_2 = (v_3, v_1)$, $e_3 = (v_4, v_3)$, $e_4 = (v_1, v_4)$, $e_5 = (v_4, v_2)$. Его диаграмма имеет вид



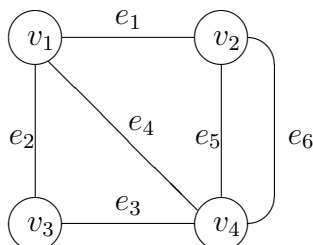
Определение. Пара одинаковых элементов V вида (v, v) называется *петлей*. Если элементами множества E могут быть петли, то граф называется *псевдографом*.

Пример. Добавим к графу из первого примера петлю $e_6 = (v_2, v_2)$, получим псевдограф



Определение. Если E не множество, а *семейство*, то есть если E содержит одинаковые элементы, то такие элементы называются *кратными ребрами*, а граф называется *мультиграфом*.

Пример. Добавим к графу из первого примера ребро $e_6 = (v_4, v_2)$. Теперь ребра e_5 и e_6 – кратные.



Также в различных задачах могут рассматриваться ориентированные мультиграфы, псевдомультиграфы и так далее, то есть, граф может одновременно содержать петли и кратные ребра, и быть при этом ориентированным или неориентированным.

Далее выражение «граф», если не оговорено другое, будет означать неориентированный граф без петель и кратных ребер, то есть простой граф. Как правило, определения, теоремы и алгоритмы, связанные с простыми графами, остаются такими же и для ориентированных графов, мультиграфов и т.д. Случаи, когда это не так, будут рассматриваться отдельно.

1.2 Смежность вершин и ребер

Определение. Пусть v_1, v_2 – вершины, $e = (v_1, v_2)$ – соединяющее их ребро. Тогда вершина v_1 и ребро e *инцидентны*, вершина v_2 и ребро e также *инцидентны*.

Определение. Два ребра, инцидентные одной вершине, называются *смежными*.

Определение. Вершина u *смежна* с вершиной v , если $(v, u) \in E$.

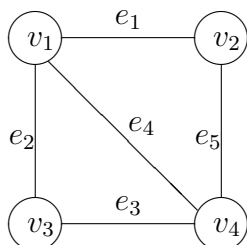
Определение. Множество вершин, смежных с вершиной v , называется *множеством смежности вершины v* и обозначается $\Gamma(v)$:

$$\Gamma(v) = \{u \in V : (v, u) \in E\}.$$

Определение. Если $A \subset V$ – множество вершин, то $\Gamma(A)$ – множество всех вершин, смежных с вершинами из множества A :

$$\Gamma(A) = \bigcup_{v \in A} \Gamma(v).$$

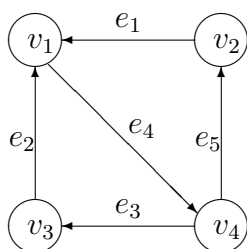
Пример. Рассмотрим граф из предыдущих примеров.



В этом графе вершины v_1 и v_2 , v_1 и v_3 , v_3 и v_4 , v_1 и v_4 , v_2 и v_4 смежны, а вершины v_2 и v_3 не смежны. Ребра e_1 и e_2 , e_2 и e_3 , e_3 и e_4 , e_1 и e_4 , e_1 и e_5 , e_2 и e_4 , e_3 и e_5 , e_4 и e_5 смежны, e_1 и e_3 , e_2 и e_5 не смежны. Здесь $\Gamma(v_1) = \{v_2, v_3, v_4\}$, $\Gamma(v_2) = \{v_1, v_4\}$, $\Gamma(v_3) = \{v_1, v_4\}$, $\Gamma(v_4) = \{v_1, v_2, v_3\}$, $\Gamma(\{v_1, v_2\}) = \{v_1, v_2, v_3, v_4\}$.

Определения, касающиеся смежности вершин, подходят также и для орграфа. Обратите внимание, что если в простом графе вершина u смежна с вершиной v , то верно и обратное: вершина v смежна с вершиной u , поскольку порядок вершин в ребре не важен. Для орграфа же это неверно.

Пример. Рассмотрим орграф из предыдущих примеров.



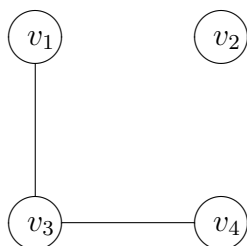
Для данного орграфа $\Gamma(v_1) = \{v_4\}$, $\Gamma(v_2) = \{v_1\}$, $\Gamma(v_3) = \{v_1\}$, $\Gamma(v_4) = \{v_2, v_3\}$, а также $\Gamma(\{v_1, v_2\}) = \{v_1, v_4\}$.

Определение. Количество ребер, инцидентных вершине v , называется *степенью* (или *валентностью*) вершины v и обозначается $d(v)$.

Пример. Для простого графа из предыдущего примера $d(v_1) = d(v_4) = 3$, $d(v_2) = d(v_3) = 2$.

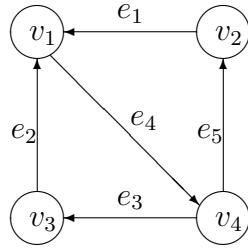
Определение. Если степень вершины равна 0, то вершина называется *изолированной*. Если степень вершины равна 1, то вершина называется *висячей*.

Пример. В данном графе v_1 и v_4 – висячие вершины, v_2 – изолированная.



Определение. Для орграфа число дуг, исходящих из вершины v , называется *полустепенью исхода* $d^+(v)$, а входящих – *полустепенью захода* $d^-(v)$.

Пример. Для орграфа из предыдущего примера $d^+(v_1) = 1$, $d^-(v_1) = 2$.



Лемма (Эйлера). Сумма степеней вершин графа равна удвоенному количеству ребер

$$\sum_{v \in V} d(v) = 2q, \quad \sum_{v \in V} d^-(v) + \sum_{v \in V} d^+(v) = 2q.$$

Доказательство. При подсчете суммы степеней вершин каждое ребро учитывается два раза.

Эта лемма также называется «лемма о рукопожатиях» и имеет следующую интерпретацию: если произошло q рукопожатий, то всего участвующие в этом люди пожали руки $2q$ раз.

Теорема о числе вершин нечетной степени. Число вершин нечетной степени в графе четно.

Доказательство. В противном случае сумма степеней вершин была бы нечетной, что противоречит лемме Эйлера.

1.3 Подграфы

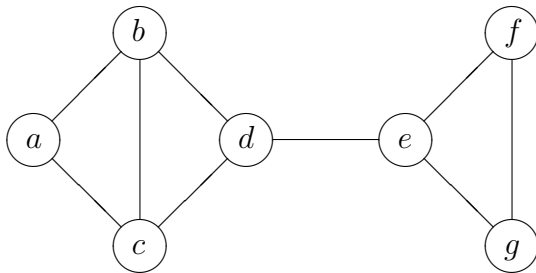
Определение. Граф $G'(V', E')$ называется *подграфом* графа $G(V, E)$, (обозначается $G' \subset G$), если $V' \subseteq V$ и $E' \subseteq E$.

Определение. Если $V' = V$, то подграф G' называется *остовным подграфом* графа G .

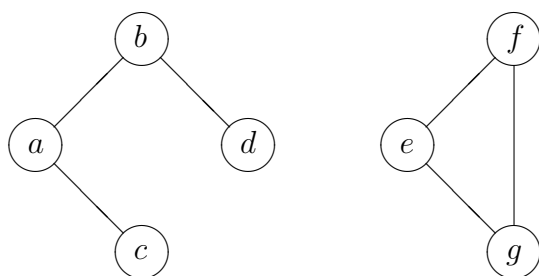
Определение. Если $V' \neq V$ и $E' \neq E$, то подграф G' называется *собственным подграфом* графа G .

Определение. Подграф $G'(V', E')$ называется *правильным подграфом* графа $G(V, E)$, если G' содержит все возможные ребра графа G : $\forall u, v \in V' (u, v) \in E \Rightarrow (u, v) \in E'$.

Примеры. Рассмотрим граф $G(V, E)$.



Его остовный неправильный подграф.



Его собственные подграфы: слева – неправильный (не содержит ребро (a, b) , которое есть в исходном графе), справа – правильный.



Правильный подграф определяется множеством вершин V' . Очевидно, что подграф не может быть одновременно остовным и собственным. Правильный остовный подграф графа $G(V, E)$ есть сам граф $G(V, E)$.

1.4 Типы графов

Рассмотрим некоторые частные виды графов.

Определение. Граф $G(V, E)$ называется *тривиальным*, если $p = 1$, $E = \emptyset$.

Определение. Граф $G(V, E)$ называется *пустым*, если $E = \emptyset$.

Определение. Граф $G(V, E)$ называется *полным*, если в нем любые две вершины смежны, то есть $\forall u, v \in V (u, v) \in E$.

Полный граф определяется числом своих вершин и обозначается K_n , где $n = |V|$. Полный граф имеет максимально возможное число ребер: $q(K_p) = p(p-1)/2$, что следует из леммы Эйлера, поскольку степень любой вершины в полном графе равна $p-1$.

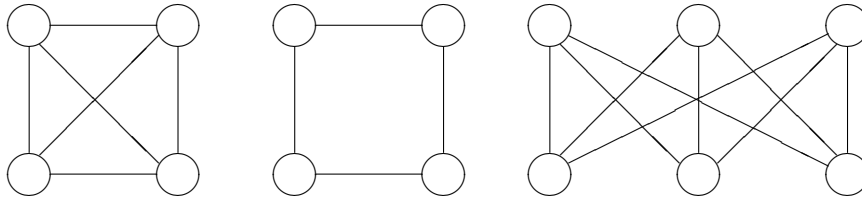
Определение. Если степени всех вершин равны k , то граф называется *регулярным степени k* .

Число ребер регулярного графа есть $kp/2$, что следует из леммы Эйлера.

Определение. Граф $G(V, E)$ называется *двудольным*, если множество V можно разбить на два непересекающихся подмножества V_1 и V_2 : $V_1 \cap V_2 = \emptyset$, $V_1 \cup V_2 = V$ таким образом, что любое ребро из E соединяет вершину из V_1 с вершиной из V_2 . Множества V_1 и V_2 называются *долями* двудольного графа. Если двудольный граф содержит все возможные ребра, то есть $\forall v_1 \in V_1, v_2 \in V_2 (v_1, v_2) \in E$, то он называется *полным двудольным графом* и обозначается $K_{m,n}$, где $m = |V_1|$, $n = |V_2|$.

Число ребер полного двудольного графа есть mn : каждая вершина из V_1 смежна с каждой вершиной из V_2 , и других ребер нет.

Примеры.



Полный граф K_4 Регулярный граф степени 2 Полный двудольный граф $K_{3,3}$

1.5 Изоморфизм графов

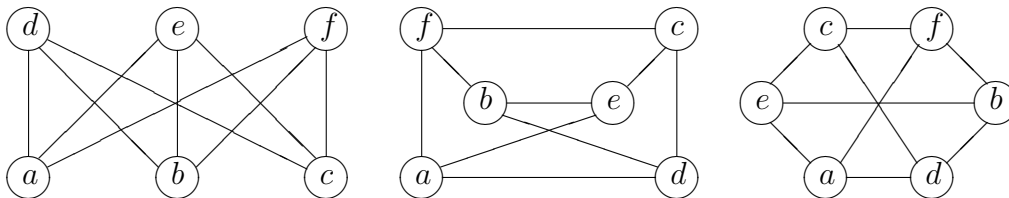
Определение. Графы $G_1(V_1, E_1)$ и $G_2(V_2, E_2)$ *изоморфны*, если существует биективная функция $h : V_1 \rightarrow V_2$, сохраняющая смежность:

$$\begin{aligned} \forall v \in V_1 \exists u = h(v) \in V_2 : \forall v_1 \neq v_2 \quad h(v_1) \neq h(v_2), \\ e_1 = (u, v) \in E_1 \Rightarrow e_2 = (h(u), h(v)) \in E_2, \\ e_2 = (u, v) \in E_2 \Rightarrow e_1 = (h^{-1}(u), h^{-1}(v)) \in E_1. \end{aligned}$$

Изоморфизм графов есть отношение эквивалентности. Графы рассматриваются с точностью до изоморфизма, то есть рассматриваются классы эквивалентности по отношению изоморфизма.

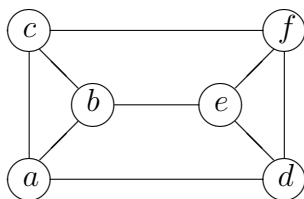
Чтобы доказать, что два графа изоморфны, достаточно найти функцию $h : V_1 \rightarrow V_2$ из определения.

Пример. Все приведенные графы являются изоморфными, это различные диаграммы двудольного графа $K_{3,3}$. Вершины a, b, c во всех трех случаях образуют одну долю, а вершины d, e, f – вторую.



Если графы являются изоморфными, то они имеют одинаковое количество вершин и ребер, а также одинаковое число вершин одной валентности. Обратное неверно.

Пример. Рассмотрим граф, который, как и $K_{3,3}$, имеет 6 вершин валентности 3. Рассмотрим вершины a, b, c . Если вершина a принадлежит первой доле, то вершины b и c должны принадлежать второй, как смежные с вершиной a . Однако, вершины b и c не могут принадлежать одной доле, поскольку они смежны друг с другом. Следовательно, данный граф не является двудольным.



В общем случае для доказательства того, что графы не являются изоморфными, требуется либо перебрать все функции $h : V_1 \rightarrow V_2$ (например, для графа с шестью вершинами одинаковой степени следует перебрать $6! = 720$ перестановок вершин), либо, как в примере выше, найти свойство, которым обладает один граф, но не обладает другой (в рассмотренном случае это двудольность). Так или иначе, проверка графов на изоморфизм является вычислительно сложной задачей.

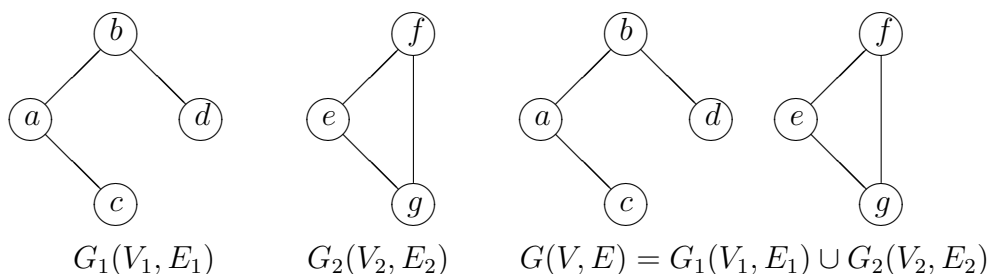
1.6 Операции над графами

Рассмотрим ряд операций над графами.

1. *Объединение* графов $G_1(V_1, E_1)$ и $G_2(V_2, E_2)$ при условии $V_1 \cap V_2 = \emptyset$ (обозначение $G_1(V_1, E_1) \cup G_2(V_2, E_2)$) – граф $G(V, E) = G_1(V_1, E_1) \cup G_2(V_2, E_2)$, где

$$V = V_1 \cup V_2, \quad E = E_1 \cup E_2.$$

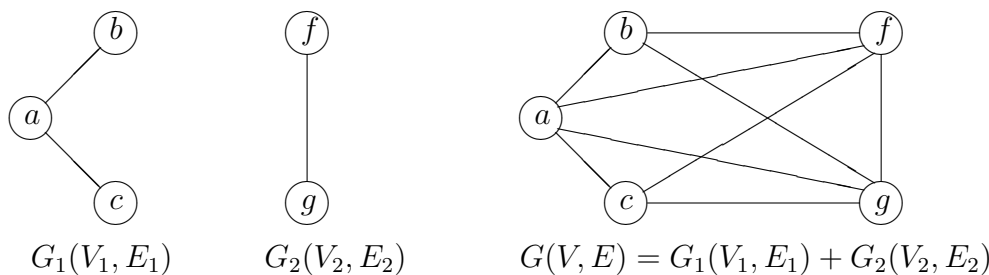
Пример.



2. *Соединение* графов $G_1(V_1, E_1)$ и $G_2(V_2, E_2)$ (обозначение $G_1(V_1, E_1) + G_2(V_2, E_2)$) при условии $V_1 \cap V_2 = \emptyset$ – граф $G(V, E) = G_1(V_1, E_1) \cup G_2(V_2, E_2)$, где

$$V = V_1 \cup V_2, \quad E = E_1 \cup E_2 \cup \{e = (v_1, v_2) : \forall v_1 \in V_1, \forall v_2 \in V_2\}$$

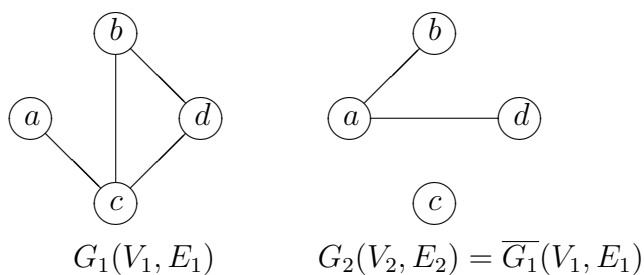
Пример.



3. *Дополнение* графа $G_1(V_1, E_1)$ (обозначение $\overline{G_1}(V_1, E_1)$) – граф $G_2(V_2, E_2)$, где

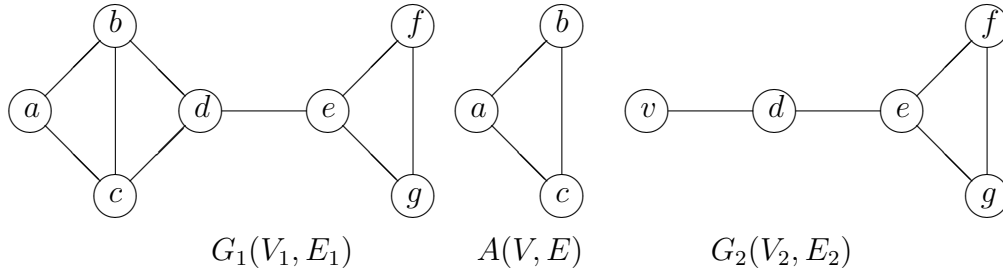
$$V_2 = V_1, \quad E_2 = \{e \in V_1 \times V_1 | e \notin E_1\}.$$

Пример.



4. Стягивание правильного подграфа $A(V, E)$ графа $G_1(V_1, E_1)$ – граф $G_2(V_2, E_2)$
 $V_2 = (V_1 \setminus V) \cup v, \quad E_2 = (E_1 \setminus E) \cup \{e = (v, u) : \exists (w, u) \in E_1, u \in V_1 \setminus V, w \in V\}$

Пример.



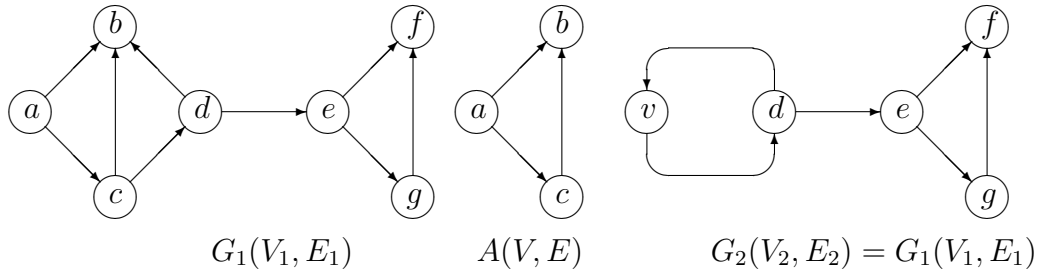
Для ориентированного графа данное определение несколько меняется:

$$V_2 = (V_1 \setminus V) \cup v;$$

$$E_2 = (E_1 \setminus E) \cup \{e = (v, u) : \exists (w, u) \in E_1, u \in V_2, w \in V\} \cup \{e = (u, v) : \exists (u, w) \in E_1, w \in V_1 \setminus V, v \in V\}$$

Это означает, что учитывается также направление ребер.

Пример. Здесь дуга $(d, v) \in E_2$ соответствует дуге $(d, b) \in E_1$, а дуга $(v, d) \in E_2$ – дуге $(c, d) \in E_1$.

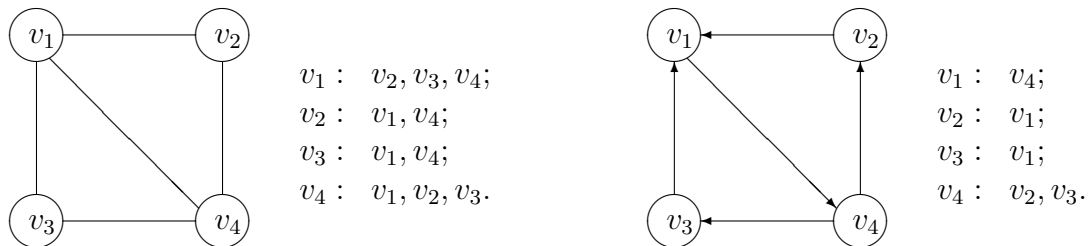


1.7 Способы задания графов

Рассмотрим граф $G(V, E)$ с p вершинами и q ребрами. Он может быть задан несколькими способами.

1. *Списки вершин и ребер* уже рассмотрены выше.
2. *Списки смежности.* Каждой вершине графа сопоставляется список смежных ей вершин.

Примеры. Рассмотрим диаграммы простого и ориентированного графа и их списки смежности.

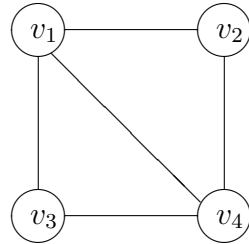


3. *Матрица смежности.* Так называется матрица $M : p \times p$,

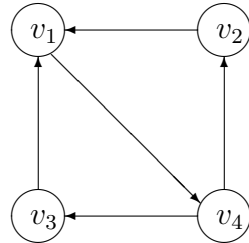
$$M_{i,j} = \begin{cases} 1, & \text{если } (v_i, v_j) \in E, \\ 0, & \text{если } (v_i, v_j) \notin E. \end{cases}$$

Это определение подходит и для орграфа $D(V, E)$.

Пример. Рассмотрим диаграммы простого графа $G(V, E)$ и ориентированного графа $D(V, E)$ и их матрицы смежности.



$$M(G) = \begin{array}{c|cccc} & v_1 & v_2 & v_3 & v_4 \\ \hline v_1 & 0 & 1 & 1 & 1 \\ v_2 & 1 & 0 & 0 & 1 \\ v_3 & 1 & 0 & 0 & 1 \\ v_4 & 1 & 1 & 1 & 0 \end{array}$$



$$M(D) = \begin{array}{c|cccc} & v_1 & v_2 & v_3 & v_4 \\ \hline v_1 & 0 & 0 & 0 & 1 \\ v_2 & 1 & 0 & 0 & 0 \\ v_3 & 1 & 0 & 0 & 0 \\ v_4 & 0 & 1 & 1 & 0 \end{array}$$

Приведем основные свойства матрицы смежности

1) Матрица смежности неориентированного графа симметрична относительно главной диагонали: $M(G) = M(G)^T$.

2) Сумма элементов матрицы смежности неориентированного графа $M(G)$ по i -й строке (по i -му столбцу) равна степени i -й вершины:

$$\sum_{j=1}^p M_{i,j} = \sum_{j=1}^p M_{j,i} = d(v_i).$$

3) Суммы элементов матрицы смежности ориентированного графа $M(D)$ по i -й строке и по i -му столбцу соответственно равны полустепени исхода и полустепени захода i -й вершины:

$$\sum_{j=1}^p M_{i,j} = d^+(v_i), \quad \sum_{j=1}^p M_{j,i} = d^-(v_i).$$

4. *Матрица инцидентий.* Так называется матрица $H : p \times q$,

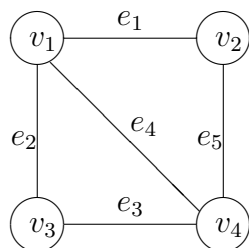
$$H_{i,j} = \begin{cases} 1, & \text{если } e_j = (v_i, v_k), \\ 0, & \text{если } e_j = (v_l, v_k), l \neq i, k \neq i, \end{cases}$$

для неориентированного графа,

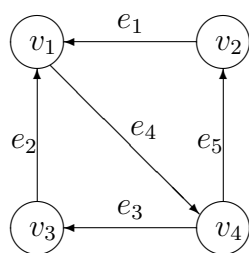
$$H_{i,j} = \begin{cases} 1, & \text{если } e_j = \{v_i, v_k\}, \\ 0, & \text{если } e_j = \{v_l, v_k\}, l \neq i, k \neq i, \\ -1, & \text{если } e_j = \{v_k, v_i\}, \end{cases}$$

для ориентированного графа.

Пример. Рассмотрим диаграммы простого графа $G(V, E)$ и ориентированного графа $D(V, E)$ и их матрицы инцидентности.



$$H(G) = \begin{array}{c|ccccc} & e_1 & e_2 & e_3 & e_4 & e_5 \\ \hline v_1 & 1 & 1 & 0 & 1 & 0 \\ v_2 & 1 & 0 & 0 & 0 & 1 \\ v_3 & 0 & 1 & 1 & 0 & 0 \\ v_4 & 0 & 0 & 1 & 1 & 1 \end{array}$$



$$H(D) = \begin{array}{c|ccccc} & e_1 & e_2 & e_3 & e_4 & e_5 \\ \hline v_1 & -1 & -1 & 0 & 1 & 0 \\ v_2 & 1 & 0 & 0 & 0 & -1 \\ v_3 & 0 & 1 & -1 & 0 & 0 \\ v_4 & 0 & 0 & 1 & -1 & 1 \end{array}$$

Приведем основные свойства матрицы инцидентности

- 1) Сумма строк матрицы $H(D)$ является нулевой строкой.
- 2) Сумма строк матрицы $H(G)$ по модулю 2 является нулевой строкой.

1.8 Упражнения

1. Перечислить все неизоморфные графы с четырьмя вершинами.
2. Доказать, что в нетривиальном графе существуют вершины одинаковой степени.
- 3 (Задача Рамсея). Доказать, что среди любых 6 человек есть 3 либо попарно знакомых, либо попарно незнакомых.
4. Сколько различных правильных подграфов есть у графа с p вершинами?
5. Привести примеры (когда это возможно):
 - двудольного полного регулярного графа;
 - графа, все 9 вершин которого имеют степень 3;
 - связного графа, не являющегося соединением двух графов.
6. Показать, что $\overline{K_{m,n}} = K_m \cup K_n$, $K_{m,n} = \overline{K_m} + \overline{K_n}$.
7. В некоторой стране есть столица и еще 100 городов. Некоторые города (в том числе и столица) соединены дорогами с односторонним движением. Из каждого нестоличного города выходит 20 дорог, а в каждый такой город входит 21 дорога. Доказать, что в столицу нельзя проехать ни из одного города.
8. Привести примеры графов, имеющих одинаковое количество вершин, ребер и для любого k одинаковое количество вершин степени k , но не являющихся изоморфными.
9. Найти все графы с 4 и 5 вершинами, изоморфные своему дополнению.
10. Рассмотрим матрицу смежности ребер $Q : q \times q$

$$H_{i,j} = \begin{cases} 1, & \text{если ребра } i, j \text{ смежны,} \\ 0, & \text{иначе.} \end{cases}$$

Может ли такая матрица служить способом задания графа?

11. Дать алгоритм построения матрицы смежности графа $\overline{G_1 + G_2}$ по матрицам смежности графов G_1 и G_2 .
12. Дать алгоритм построения матрицы смежности графа $\overline{G_1 \cup G_2}$ по матрицам смежности графов G_1 и G_2 .
13. Дать алгоритм построения матрицы смежности графа $\overline{G_1} + \overline{G_2}$ по матрицам смежности графов G_1 и G_2 .
14. Дать алгоритм построения матрицы смежности орграфа G_1 , после стягивания его правильного подграфа G_2 в вершину, по матрицам смежности графов G_1 и G_2 .

2 Связность графов

2.1 Маршруты, цепи, циклы

Определение. *Маршрутом* в графе называется последовательность вершин и ребер вида $v_0 e_1 v_1 e_2 \dots e_k v_k$, в которой $e_i = (v_{i-1}, v_i)$. Вершина v_0 называется *начальной*, а v_k – *конечной* вершиной маршрута.

Это определение подходит также для псевдо-, мульти- и орграфов. Для неориентированного графа достаточно указать только последовательность вершин либо только последовательность ребер.

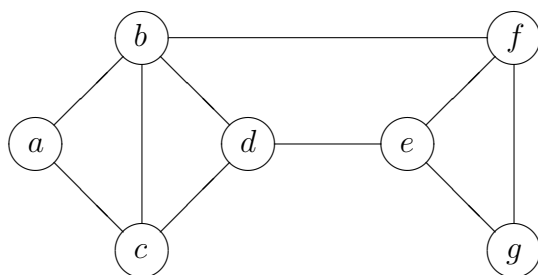
Определение. Если все ребра в маршруте различны, то маршрут называется *цепью*. Если все вершины (а значит и ребра) в маршруте различны, то маршрут называется *простой цепью*. В цепи $v_0 e_1 v_1 e_2 \dots e_k v_k$, вершины v_0, v_k называются *концами цепи*. Говорят, что цепь с концами u, v *соединяет вершины u, v* (обозначается $\langle u, v \rangle$).

Определение. Если $v_0 = v_k$, маршрут называется *замкнутым*.

Определение. Если все ребра в замкнутом маршруте различны, то он называется *циклом*. Если все вершины в замкнутом маршруте, кроме первой и последней, различны, то он называется *простым циклом*.

Для орграфов цепь называется *путем*, а цикл *контуром*.

Примеры. Рассмотрим граф $G(V, E)$.



- $abdbc$ – маршрут, но не цепь;
- $abdc$ – цепь, но не простая цепь;
- $abcde$ – простая цепь;
- $abdbca$ – замкнутый маршрут, но не цикл;
- $abfedbca$ – цикл, но не простой цикл;
- $abca$ – простой цикл.

Лемма. Если есть цепь, соединяющая вершины u, v , то есть и простая цепь, соединяющая вершины u, v .

Доказательство. Индукция по длине цепи. Цепь, состоящая из одного ребра, является простой, следовательно, для нее утверждение верно. Пусть оно верно для цепей длины $1, 2, \dots, k-1$. Рассмотрим цепь длины k . Пусть в цепи $v_0e_1v_1e_2\dots e_kv_k$ есть две одинаковые вершины, то есть цепь выглядит следующим образом $v_0e_1v_1e_2\dots v_ie_{i+1}\dots e_jv_ie_{j+1}\dots e_kv_k$. Выбросив участок $e_{i+1}\dots e_jv_i$, получим новую цепь $v_0e_1v_1e_2\dots v_ie_{j+1}\dots e_kv_k$, длина которой меньше k , и из которой, по индукционному предположению, можно выделить простую цепь.

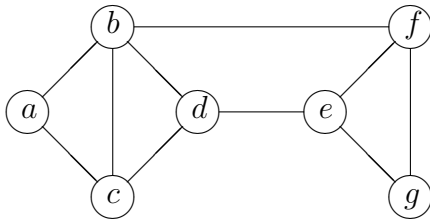
2.2 Расстояния в графе

Определение. *Длиной маршрута* называется количество ребер в нем. Если маршрут $\mu = v_0e_1v_1e_2\dots e_kv_k$, то длина μ равна k (обозначается $|\mu| = k$).

Определение. *Расстоянием между вершинами u, v* (обозначается $s(u, v)$) называется наименьшая длина цепи $\langle u, v \rangle$. Цепь $\mu = \langle u, v \rangle$, для которой $|\mu| = s(u, v)$, называется *кратчайшей цепью*.

Если $\neg \exists \langle u, v \rangle$, то по определению $s(u, v) = \infty$.

Пример. В рассмотренном графе $s(a, b) = 2$, кратчайшая цепь, например, abd .



Определение. *Диаметром графа $G(V, E)$* (обозначается $D(G)$) называется наибольшее расстояние между двумя его вершинами.

$$D(G) = \max_{u, v \in V} s(u, v).$$

Пусть v – произвольная вершина из V .

Определение. Максимальное из расстояний между вершиной v и остальными вершинами из $G(V, E)$ называется *максимальным удалением* в графе $G(V, E)$ от вершины v (обозначается $r(v)$)

$$r(v) = \max_{u \in V} s(u, v).$$

Определение. *Радиусом графа $G(V, E)$* (обозначается $R(G)$) называется минимальное по v из максимальных удалений от вершины v

$$R(G) = \min_{v \in V} r(v) = \min_{v \in V} \max_{u \in V} s(u, v).$$

Определение. Любая вершина $v \in V$, такая что $r(v) = R(G)$ называется *центром графа*.

Пример. В рассмотренном выше графе диаметр равен 3, радиус равен 2, $r(a) = r(c) = r(e) = r(g) = 3$, $r(b) = r(d) = r(f) = 2$, центрами являются вершины b, d, f .

2.3 Связность неориентированных графов

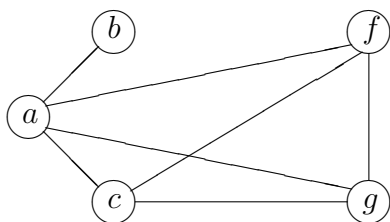
Рассмотрим неориентированный граф $G(V, E)$.

Определение. Две вершины в графе *связны*, если существует соединяющая их цепь.

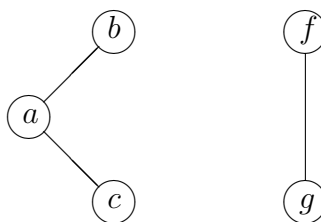
Определение. Граф называется *связным*, если любые две вершины в нем связны.

Определение. *Компонентой связности* графа G называется его правильный связный подграф, не являющийся собственным подграфом никакого другого связного подграфа графа G .

Примеры.



Связный граф



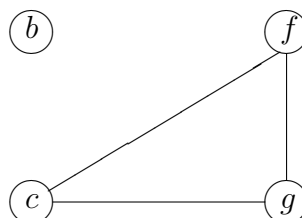
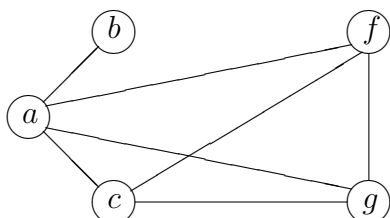
Граф с двумя компонентами связности

Будем обозначать число компонент связности графа G через $k(G)$.

При взгляде на диаграммы связных графов возникает впечатление, что граф может быть «более» и «менее» связным. Рассмотрим некоторые количественные меры связности.

Определение. Вершина графа G называется *точкой сочленения*, если ее удаление (вместе с инцидентными ей ребрами) увеличивает число компонент связности графа.

Примеры. Для графа слева a – точка сочленения. Результат ее удаления (граф с двумя компонентами связности) показан на рисунке справа.



Лемма о точках сочленения. В любом графе G с $p \geq 2$ есть по крайней мере две вершины, не являющиеся точками сочленения.

Доказательство. Если в нетривиальном графе $q = 0$, то он является объединением изолированных вершин, а любая изолированная вершина не является точкой сочленения (ее удаление уменьшает число компонент связности). Пусть $q > 0$. Без ограничения общности предположим, что G – связный граф, иначе рассмотрим любую его компоненту связности, содержащую не менее двух вершин. Пусть вершины u и v таковы, что $d(u, v) = D(G)$. Предположим, что u является точкой сочленения. Удалим ее из графа вместе с инцидентными ей ребрами, получим граф G' , который является несвязным. Это означает, что найдется такая вершина w , которая не связна с вершиной v в графе G' . Следовательно, все

цепи $\langle w, v \rangle$ в графе G проходят через вершину u , и

$$d(w, v) = d(w, u) + d(u, v) = d(w, u) + D(G) > D(G),$$

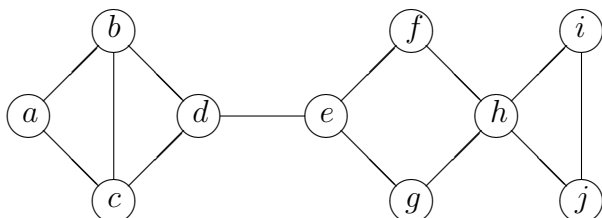
что противоречит определению диаметра графа. Значит, вершина u (как и вершина v) не является точкой сочленения.

Определение. Связный граф, не имеющий точек сочленения, называется *блоком*.

Определение. Ребро графа G называется *мостом*, если его удаление увеличивает число компонент связности графа.

Если в графе, отличном от K_2 , есть мост, то есть и точка сочленения. Концы всякого моста (кроме висячих вершин) являются точками сочленения, но не всякая точка сочленения является концом моста.

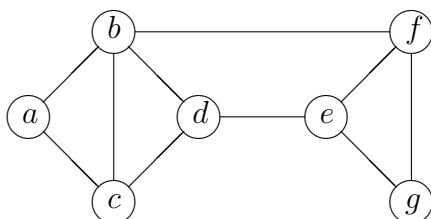
Пример. Здесь блоками являются правильные подграфы, заданные множествами вершин $\{a, b, c\}$, $\{b, c, d\}$, $\{e, f, g, h\}$, $\{h, i, j\}$. Мостом является ребро (d, e) , точками сочленения – вершины d, e и h .



Определение. Числом вершинной связности графа G (обозначается $\kappa(G)$) называется наименьшее число вершин, удаление которых приводит к несвязному или тривиальному графу. Граф G называется m -связным, если $\kappa(G) = m$.

Определение. Числом реберной связности графа G (обозначается $\lambda(G)$) называется наименьшее число ребер, удаление которых приводит к несвязному или тривиальному графу.

Примеры. Здесь $\kappa(G) = 2$ (удаление вершин b и c приведет к несвязному графу), $\lambda(G) = 2$ (например, можно удалить ребра (b, f) и (d, e)).



Обозначим через $\delta(G)$ минимальную из степеней вершин графа G .

Теорема. $\kappa(G) \leq \lambda(G) \leq \delta(G)$.

Доказательство.

• $\kappa(G) \leq \lambda(G)$. Если $\lambda(G) = 0$, то граф несвязен, и $\kappa(G) = 0$. Если $\lambda(G) = 1$, то в графе есть мост, а значит, либо по крайней мере один конец моста является точкой сочленения, либо $G = K_2$, и в обоих этих случаях $\kappa(G) = 1$. Пусть теперь $\lambda(G) \geq 2$. Найдем множество E_λ , содержащее $\lambda = \lambda(G)$ ребер, удаление которых приведет к несвязному или тривиальному графу. Удалим $\lambda - 1$ ребро из множества E_λ , оставшееся ребро (u, v) будет мостом. Для каждого из удаленных ребер удалим инцидентную вершину, отличную от u и v , всего будет удалено не больше,

чем $\lambda - 1$ вершин. Если после этого граф несвязен, то $\kappa \leq \lambda - 1 < \lambda$, если связан, то удалим u или v , граф станет несвязным в результате удаления не более чем λ вершин, а значит, $\kappa \leq \lambda$.

• $\lambda(G) \leq \delta(G)$. Если $\delta(G) = 0$, то граф несвязный и $\lambda(G) = 0$. Иначе рассмотрим вершину v степени $\delta(G)$. Удаление всех инцидентных ей ребер приводит к несвязному графу, а значит, $\lambda(G) \leq \delta(G)$.

2.4 Оценка числа ребер в графе

Теорема. Число вершин p , ребер q и компонент связности k графа $G(V, E)$ удовлетворяет неравенствам

$$p - k \leq q \leq \frac{(p - k)(p - k + 1)}{2}.$$

Доказательство.

• Докажем сначала, что $p - k \leq q$. Используем метод математической индукции по числу ребер q . База индукции: $p = 1, k = 1, q = 0$. Пусть оценка верна для всех графов с числом вершин, меньшим p . Рассмотрим граф $G(V, E)$ с p вершинами. Удалим из G вершину v , которая не является точкой сочленения (по лемме о точках сочленения в графе есть по крайней мере две такие вершины), вместе с инцидентными ей ребрами, получим граф $G'(V', E')$. Тогда если v – изолированная вершина, то в графе G' число вершин $p' = p - 1$, число ребер $q' = q$, число компонент связности $k' = k - 1$. Имеем $p - k = p' - k' \leq q' = q$. Если v – не изолированная вершина, то в графе G' число вершин $p' = p - 1$, число ребер $q' < q$, число компонент связности $k' = k$. Имеем $p - k = p' - k' + 1 \leq q' + 1 \leq q$.

• Докажем теперь, что $q \leq (p - k)(p - k + 1)/2$. Используем метод выделения «критических» графов, т.е. найдем граф с заданным числом вершин и компонент связности, содержащий максимальное число ребер. Число ребер q графа с p вершинами и k компонентами связности, когда число вершин в i -той компоненте связности равно p_i , не превосходит числа ребер в графе p вершинами и k компонентами связности, в котором i -я компонента связности представляет собой полный граф K_{p_i} . Следовательно, достаточно рассматривать графы, в которых все компоненты связности полные. Найдем среди всех таких графов граф с максимальным числом ребер. Пусть есть две компоненты связности $G_1(V_1, E_1) = K_{p_1}$ и $G_2(V_2, E_2) = K_{p_2}$ такие, что $1 < p_1 \leq p_2$. Если перенести одну вершину из G_1 в G_2 , то число ребер изменится на $\Delta q = p_2 - (p_1 - 1) > 0$, то есть число ребер возрастет. Повторяя эту процедуру, пока это возможно, получаем, что наибольшее число ребер имеет граф

$$K_{p-k+1} \cup \left(\bigcup_{i=1}^{k-1} K_1 \right).$$

Число ребер в таком графе

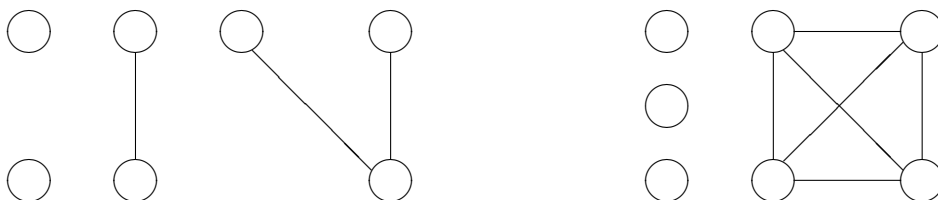
$$q \left(K_{p-k+1} \cup \left(\bigcup_{i=1}^{k-1} K_1 \right) \right) = \frac{(p - k + 1)(p - k + 1 - 1)}{2} + (k - 1)0 = \frac{(p - k)(p - k + 1)}{2}.$$

Что и требовалось доказать.

Следствие. Если $q > (p - 1)(p - 2)/2$, то граф связан.

Доказательство. Из утверждения теоремы следует, что чем больше компонент связности, тем ниже верхняя граница числа ребер. Для графа из двух компонент связности максимальное число ребер равно $(p - 1)(p - 2)/2$. Добавление хотя бы одного ребра приведет к увеличению компонент связности.

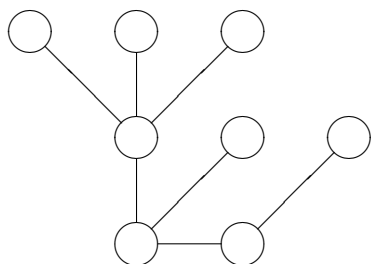
Пример. Пусть число вершин $p = 7$ и число компонент связности $k = 4$. Слева представлен граф с наименьшим числом ребер, справа – с наибольшим.



2.5 Деревья

Определение. Граф $G(V, E)$ называется *деревом*, если он является связным и не имеет циклов. Граф $G(V, E)$, все компоненты связности которого являются деревьями, называется *лесом*.

Пример. Диаграмма дерева.



Деревья являются связными графами с наименьшим числом ребер, поскольку удаление ребра не приведет к увеличению компонент связности тогда и только тогда, когда это ребро входит в цикл. Более строго это и другие свойства деревьев сформулированы в следующей теореме.

Теорема о шести эквивалентных утверждениях о дереве. Следующие утверждения эквивалентны:

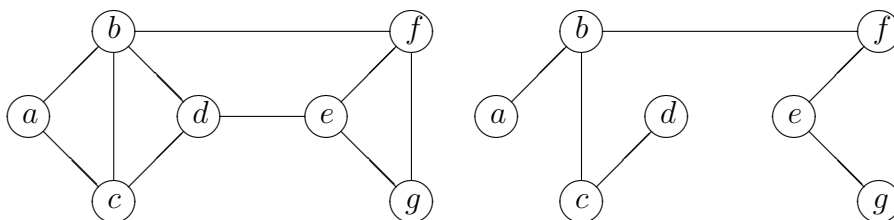
- 1) граф G – дерево, то есть связный граф без циклов;
- 2) любые две вершины графа G соединены единственной простой цепью;
- 3) граф G связный, и любое ребро есть мост;
- 4) граф G связный, и число его ребер на 1 меньше числа вершин ($q = p - 1$);
- 5) граф G не содержит циклов, и число его ребер на 1 меньше числа вершин ($q = p - 1$);
- 6) граф G не содержит циклов, но добавление к нему любого нового ребра приводит к образованию ровно одного простого цикла, проходящего через это ребро.

Доказательство будет рассмотрено позже, в разделе 5.

Следствие. Если граф $G(V, E)$ – лес с p вершинами и k компонентами связности, то $q = p - k$.

Определение. *Остовным деревом (остовом)* связного графа G называется любой его подграф, содержащий все вершины графа G и являющийся деревом.

Пример. Слева – граф, справа – его остовное дерево.



В общем случае граф имеет несколько остовных деревьев.

Алгоритм построения остовного дерева

Начало. Задан связный нетривиальный граф, требуется построить его остовное дерево.

Шаг 1. Выбираем произвольное ребро и включаем его в остовное дерево вместе с концевыми вершинами. Полагаем $n = 1$.

Шаг 2. Если $n < p - 1$, выбираем очередное ребро $e = (u, v)$, где u принадлежит остовному дереву, а v – не принадлежит. Включаем вершину v и ребро u в дерево.

Шаг 3. Полагаем $n = n + 1$ и идем на шаг 2.

Конец. Остовное дерево построено.

Обоснование алгоритма. На первом шаге в дерево включается две вершины и одно ребро. Число n показывает число ребер, включенных в дерево. На втором шаге каждый раз в дерево добавляется одна вершина и одно ребро, поэтому число вершин, включенных в дерево, равно $n + 1$. Отсюда и из связности графа следует, что ребро, которое выбирается на шаге 2, существует при $n < p - 1$. В результате по построению мы получаем связный граф с p вершинами и $p - 1$ ребром, т.е. остовное дерево.

Вычислительная сложность алгоритма. Алгоритм выполняется за $p - 1$ итерацию, на каждой из которых происходит выбор в худшем случае из q ребер. Значит, вычислительная сложность алгоритма равна pq .

Пример. Для графа и его остовного дерева из предыдущего примера порядок включения ребер в дерево может быть таким: (a, b) , (b, c) , (c, d) , (b, f) , (f, e) , (e, g) (возможны и другие варианты).

Остовные деревья часто используются для решения задач теории графов. Две такие задачи, а именно, построение системы фундаментальных циклов и системы фундаментальных разрезов, рассмотрены в следующих подразделах.

2.6 Система фундаментальных циклов

Пусть $G(V, E)$ – мультиграф с p вершинами, q ребрами, и k компонентами связности.

Определение. *Коцикломатическим числом* графа $G(V, E)$ называется число $\rho(G) = p - k$.

Согласно следствию из теоремы о шести эквивалентных утверждениях о дереве, коцикломатическое число представляет собой общее число ребер в остовах всех компонент связности графа.

Определение. Цикломатическим числом графа $G(V, E)$ называется число $\nu(G) = q - p + k$.

Цикломатическое число показывает, сколько ребер нужно удалить, чтобы граф стал лесом с k компонентами связности.

В теории электрических цепей числа $\rho(G)$ и $\nu(G)$ имеют прямой физический смысл. Цикломатическое число равно наибольшему числу независимых контуров в графе электрической цепи, т.е. наибольшему числу независимых круговых токов, которые могут протекать в цепи. Коцикломатическое число равно наибольшему числу независимых разностей потенциалов между узлами цепи.

Цикл, в том числе, простой цикл, может быть представлен множеством ребер, входящих в него. Рассмотрим операцию сложения по модулю 2, или симметрической разности (\oplus) над множествами ребер. Пусть $\mu_1 \subset E$, $\mu_2 \subset E$, тогда

$$\mu_1 \oplus \mu_2 = \{e : e \in \mu_1, e \notin \mu_2\} \cup \{e : e \in \mu_2, e \notin \mu_1\}.$$

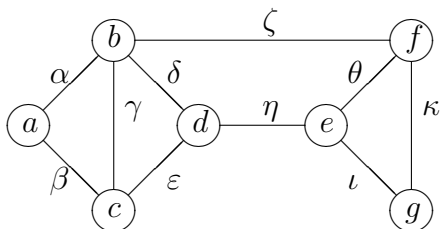
Определим также операцию умножения множества ребер на число из множества $\{0, 1\}$ следующим образом:

$$0\mu = \emptyset, \quad 1\mu = \mu.$$

Определение. Линейной комбинацией множеств μ_1, \dots, μ_n называется множество μ вида

$$\mu = \bigoplus_{i=1}^n a_i \mu_i, \quad a_i \in \{0, 1\}.$$

Пример. Рассмотрим граф из предыдущего примера



$$\mu_1 = \{\delta, \zeta, \eta, \theta\};$$

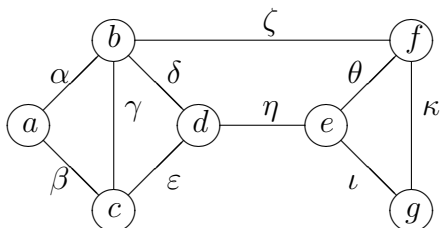
$$\mu_2 = \{\gamma, \delta, \varepsilon\};$$

$$\mu_3 = \{\alpha, \beta, \gamma\};$$

$$1\mu_1 \oplus 1\mu_2 \oplus 0\mu_3 = \{\delta, \zeta, \eta, \theta\} \oplus \{\gamma, \delta, \varepsilon\} \oplus \emptyset = \{\gamma, \delta, \varepsilon, \zeta, \eta, \theta\}.$$

Определение. Множество циклов $\{\mu_i\}_{i=1, \dots, n}$ называется *независимым*, если ни один цикл не является линейной комбинацией остальных, иначе множество называется *зависимым*.

Пример. Рассмотрим граф из предыдущих примеров и множество циклов $\{\mu_1, \mu_2, \mu_3, \mu_4\}$. Это множество циклов зависимо, поскольку $\mu_4 = \mu_2 \oplus \mu_3$. Множество $\{\mu_1, \mu_2, \mu_4\}$ независимо.



$$\mu_1 = \{\delta, \zeta, \eta, \theta\};$$

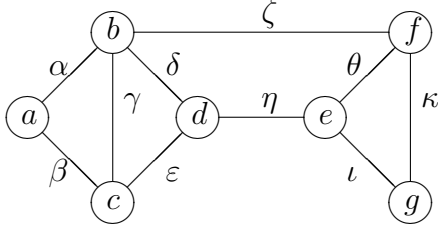
$$\mu_2 = \{\gamma, \delta, \varepsilon\};$$

$$\mu_3 = \{\alpha, \beta, \gamma\};$$

$$\mu_4 = \{\alpha, \beta, \delta, \varepsilon\}.$$

Определение. Независимое множество циклов максимальной мощности называется *системой фундаментальных циклов*, а циклы этой системы – *фундаментальными циклами*.

Пример. Рассмотрим граф из предыдущих примеров и множество циклов $\mu = \{\mu_1, \mu_2, \mu_3, \mu_4\}$



$$\begin{aligned}\mu_1 &= \{\delta, \zeta, \eta, \theta\}; \\ \mu_2 &= \{\gamma, \delta, \varepsilon\}; \\ \mu_3 &= \{\alpha, \beta, \gamma\}; \\ \mu_4 &= \{\theta, \iota, \kappa\}.\end{aligned}$$

Множество μ независимо, поскольку каждый цикл содержит ребро, которое отсутствует в остальных циклах (например, в разрезе μ_1 таким ребром является ζ , в μ_2 – ε , и т.д.). Это означает, что никакой цикл не является линейной комбинацией остальных циклов. Множество μ является системой фундаментальных циклов, поскольку любой цикл в графе может быть представлен как линейная комбинация циклов системы. Например:

$$\begin{aligned}\{\alpha, \beta, \delta, \varepsilon\} &= \mu_2 \oplus \mu_3; \\ \{\delta, \zeta, \eta, \theta, \iota, \kappa\} &= \mu_1 \oplus \mu_4; \\ \{\alpha, \beta, \varepsilon, \zeta, \eta, \theta\} &= \mu_1 \oplus \mu_2 \oplus \mu_3.\end{aligned}$$

Заметим, что не любая линейная комбинация циклов является циклом.

Пример. Для графа из предыдущих примеров линейная комбинация циклов $\{\alpha, \beta, \gamma\} \oplus \{\theta, \iota, \kappa\} = \{\alpha, \beta, \gamma, \theta, \iota, \kappa\}$ циклом не является.

Теорема о фундаментальных циклах. Пусть $G(V, E)$ – простой связный граф. Тогда число его фундаментальных циклов равно $\nu(G) = q - p + 1$.

Доказательство (конструктивное). Рассмотрим граф $G(V, E)$ и некоторое его остовное дерево $T(V, E')$. Пусть множество $E'' = E \setminus E'$ – множество ребер, не вошедших в дерево. По теореме о шести эквивалентных утверждениях о дереве имеем, что число таких ребер равно $\nu(G) = q - p + 1$.

Построим множество циклов $\{Z_1, \dots, Z_{q-p+1}\}$ следующим образом. Выбираем очередное ребро $e_i \in E''$ и строим цикл Z_i как объединение ребра e_i и простой цепи, соединяющей концы этого ребра в дереве $T(V, E')$. Существует ровно один такой цикл, т.к. по теореме о шести эквивалентных утверждениях о дереве такая простая цепь существует и единственна. Очевидно, построенное множество циклов независимо, т.к. в каждом цикле Z_i имеется ребро e_i , которого нет в других циклах.

Покажем теперь, что любой цикл μ может быть представлен в виде линейной комбинации циклов $\{Z_1, \dots, Z_{q-p+1}\}$. Любой цикл μ содержит ребра из множества E'' , т.к. в дереве циклов нет. Пусть μ содержит n таких ребер $\{e_1, \dots, e_n\}$, тогда

$$\mu = Z_1 \oplus \dots \oplus Z_n.$$

Докажем это соотношение с помощью метода математической индукции. Пусть μ содержит одно такое ребро e_1 , тогда $\mu = Z_1$ по построению Z_1 . Пусть формула

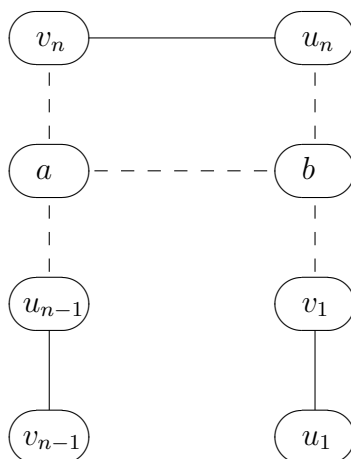
верна для цикла, содержащего $1, \dots, n-1$ ребер. Рассмотрим цикл μ , содержащий n ребер $e_i = (v_i, u_i)$. Такой цикл в общем случае имеет вид

$$(v_1, u_1) < u_1, v_2 > (v_2, u_2) < u_2, v_3 > \dots (v_n, u_n) < u_n, v_1 >$$

где $< u_1, v_2 >, \dots, < u_n, v_1 >$ – простые цепи, состоящие из ребер остовного дерева T . Они не имеют общих ребер, т.к. входят в состав цикла μ . По предположению индукции цикл μ_{n-1} , содержащий ребра $\{e_1, \dots, e_{n-1}\}$, имеет вид

$$\mu_{n-1} = Z_1 \oplus \dots \oplus Z_{n-1}.$$

В цикл μ_{n-1} входит простая цепь $< u_{n-1}, v_1 >$, состоящая из ребер дерева, она не имеет общих ребер с цепями $< u_1, v_2 >, \dots, < u_{n-2}, v_{n-1} >$. Разобьем эту цепь на три участка: $< u_{n-1}, a >$, $< a, b >$ и $< b, v_1 >$, где участок $< u_{n-1}, a >$ является частью цепи $< u_{n-1}, v_n >$, принадлежащей дереву, участок $< b, v_1 >$ – частью цепи $< u_n, v_1 >$, принадлежащей дереву (каждый из этих участков может быть пустым). Схематично это выглядит следующим образом (сплошные линии – ребра, не принадлежащие дереву, пунктирные линии – цепи, принадлежащие дереву):



Цепь $< a, b >$ не имеет общих ребер с цепями $< u_{n-1}, v_n >$ и $< u_n, v_1 >$, поскольку иначе в дереве содержались бы циклы. Поскольку в дереве T любые две вершины соединены единственной простой цепью, таковой цепью для вершин v_n и u_n является цепь, состоящая из участков $< v_n, a >$, $< a, b >$ и $< b, u_n >$, которые вместе с ребром $e_n = (v_n, u_n)$ образуют цикл Z_n . Таким образом, общей частью циклов μ_{n-1} и Z_n является цепь $< a, b >$, откуда

$$\mu_{n-1} \oplus Z_n = (v_1, u_1) < u_1, v_2 > \dots (v_{n-1}, u_{n-1}) < u_{n-1}, v_n > (v_n, u_n) < u_n, v_1 > = \mu.$$

Окончательно имеем

$$Z_1 \oplus \dots \oplus Z_{n-1} \oplus Z_n = \mu,$$

что и требовалось доказать.

На основе доказательства теоремы сформулируем следующий алгоритм.

Алгоритм построения системы фундаментальных циклов (основан на доказательстве теоремы о фундаментальных циклах)

Начало. Задан связный нетривиальный граф $G(V, E)$, требуется построить его систему фундаментальных циклов.

Шаг 1. Строим произвольное остовное дерево $T(V, E')$ графа. Полагаем $i = 0$.

Шаг 2. Если $i = q - p + 1$, идем на конец. Иначе полагаем $i = i + 1$.

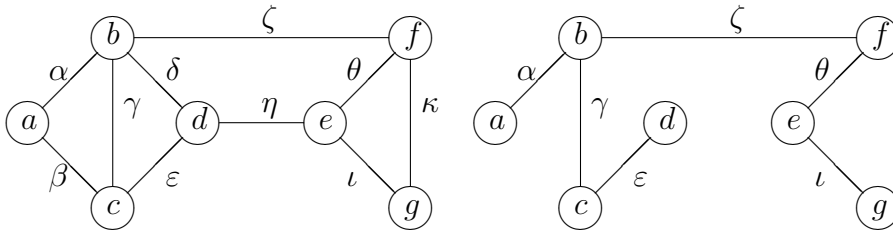
Шаг 3. Выбираем очередное ребро $e_i = (v_i, u_i) \in E \setminus E'$, не входящее в остовное дерево.

Шаг 4. Находим цепь $\langle u, v \rangle$, состоящую из ребер остовного дерева. Объединяем ее с ребром e_i , получаем цикл Z_i . Идем на шаг 2.

Конец. Система фундаментальных циклов построена.

Заметим, что все фундаментальные циклы, полученные с помощью данного алгоритма, являются простыми.

Пример. Слева – граф из предыдущих примеров, справа – его остовное дерево. Построим его систему фундаментальных циклов.



Выбираем очередное ребро $e_i = (v_i, u_i)$, не входящее в остовное дерево, находим цепь $\langle v_i, u_i \rangle$ из ребер остовного дерева, и, объединяя найденную цепь с ребром e_i , получаем цикл Z_i :

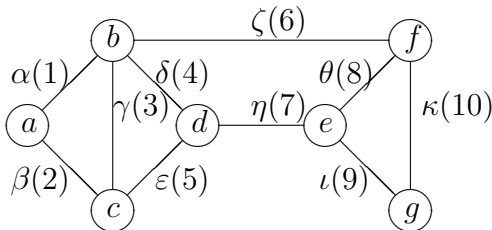
$$\begin{aligned} e_1 = \beta = (a, c), \quad \langle a, c \rangle &= abc, & Z_1 &= \{\beta, \alpha, \gamma\}; \\ e_2 = \delta = (b, d), \quad \langle b, d \rangle &= bcd, & Z_2 &= \{\delta, \gamma, \varepsilon\}; \\ e_3 = \eta = (d, e), \quad \langle d, e \rangle &= dc b f e, & Z_3 &= \{\eta, \varepsilon, \gamma, \zeta, \theta\}; \\ e_4 = \kappa = (f, g), \quad \langle f, g \rangle &= f e g, & Z_4 &= \{\kappa, \theta, \iota\}. \end{aligned}$$

Примечание. Для компактной записи операций над множествами циклов удобно использовать представление подмножеств булевыми векторами. Пусть заданы множество $M = \{m_1, m_2, \dots, m_n\}$ и его подмножество A . Построим булев вектор $\alpha = a_1 a_2 \dots a_n$, представляющий подмножество A , следующим образом: зафиксируем порядок элементов в множестве M и положим

$$a_i = \begin{cases} 1, & \text{если } m_i \in A; \\ 0, & \text{если } m_i \notin A. \end{cases}$$

Операции объединения, пересечения, сложения по модулю 2 над множествами эквивалентны операциям покомпонентной дизъюнкции, конъюнкции и сложения по модулю 2 для соответствующих векторов.

Пример. Рассмотрим граф из предыдущих примеров, перенумеруем его ребра, номер ребра поставим в скобках после его обозначения. Рассмотрим циклы $\mu = \{\mu_1, \mu_2, \mu_3, \mu_4\}$, запишем их в виде подмножеств ребер и в виде булевых векторов.



$$\begin{aligned} \mu_1 &= \{\delta, \zeta, \eta, \theta\} &= 0001011100; \\ \mu_2 &= \{\gamma, \delta, \varepsilon\} &= 0011100000; \\ \mu_3 &= \{\alpha, \beta, \gamma\} &= 1110000000; \\ \mu_4 &= \{\theta, \iota, \kappa\} &= 0000000111. \end{aligned}$$

Рассмотрим объединение подмножеств μ_1 и μ_2 :

$$\begin{array}{r} 0001011100 \\ \vee 0011100000 \\ \hline 0011111100 \end{array}$$

Действительно, $\mu_1 \cup \mu_2 = \{\gamma, \delta, \varepsilon, \zeta, \eta, \theta\}$. Рассмотрим пересечение подмножеств:

$$\begin{array}{r} 0001011100 \\ \wedge 0011100000 \\ \hline 0001000000 \end{array}$$

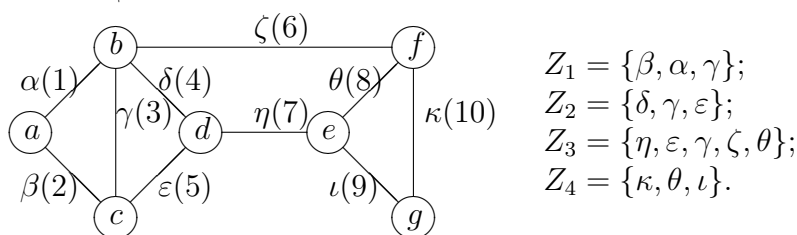
Действительно, $\mu_1 \cap \mu_2 = \{\delta\}$. Рассмотрим сумму по модулю 2 подмножеств:

$$\begin{array}{r} 0001011100 \\ \oplus 0011100000 \\ \hline 0010111100 \end{array}$$

Действительно, $\mu_1 \oplus \mu_2 = \{\gamma, \varepsilon, \zeta, \eta, \theta\}$.

Определение. Матрицей фундаментальных циклов графа $G(V, E)$ называется матрица $\Phi(G)$ размерности $(q-p+1) \times q$, строками которой являются булевы вектора, соответствующие фундаментальным циклам.

Пример. Рассмотрим граф из предыдущих примеров и его систему фундаментальных циклов.



Матрица фундаментальных циклов имеет вид

$$\Phi(G) = \begin{array}{c|cccccccccc} & \alpha & \beta & \gamma & \delta & \varepsilon & \zeta & \eta & \theta & \iota & \kappa \\ \hline Z_1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ Z_2 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ Z_3 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ Z_4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \end{array}$$

Определение. Произведением по модулю 2 (\oplus) для булевых матриц $A : m \times k$ $B : k \times t$ называется матрица $C : m \times t$ (обозначается $C = A \oplus B$), вычисляемая по следующему правилу:

$$C_{ij} = \bigoplus_{l=1}^k A_{il} B_{lj}.$$

Теорема о матрице фундаментальных циклов. Если $I(G)$ – матрица инцидентий графа $G(V, E)$, $\Phi(G)$ – его матрица фундаментальных циклов, то $I \oplus \Phi^T = 0$, 0 – матрица, состоящая из нулей.

Доказательство. Элементы матрицы $C = I \oplus \Phi^T$ имеют вид

$$C_{ij} = \bigoplus_{l=1}^k I_{il} \Phi_{lk}.$$

Здесь $I_{ik}\Phi_{jk} = 1$, если и только если $I_{ik} = 1$ и $\Phi_{jk} = 1$, т.е. если вершина i инцидентна ребру k , и ребро k входит в фундаментальный цикл Z_j . Возможны два варианта: либо вершина i не входит в цикл Z_j , т.е. не инцидентна ни одному ребру этого цикла, и тогда $\forall k : I_{ik}\Phi_{jk} = 0$, либо вершина i входит в цикл Z_j , тогда, поскольку Z_j – простой цикл, она инцидентна ровно двум ребрам этого цикла, и $I_{ik}\Phi_{jk} = 1$ ровно для двух значений k . В любом из этих случаев

$$\bigoplus_{k=1}^q I_{ik}\Phi_{jk} = 0,$$

что и требовалось доказать.

Теорема позволяет получить матрицу инциденций по матрице фундаментальных циклов, и наоборот. Для этого достаточно решить уравнение $I \oplus \Phi^T = 0$. Таким образом, матрица фундаментальных циклов является еще одним способом задания графа. Такое его представление используется в теории электрических цепей и теории релейных схем.

2.7 Система фундаментальных разрезов

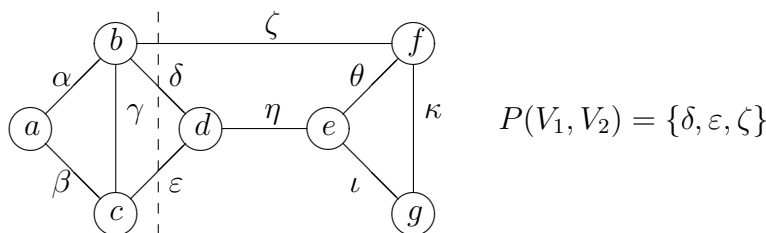
Рассмотрим связный граф $G(V, E)$. Представим его множество вершин в виде объединения двух непересекающихся подмножеств: $V = V_1 \cup V_2$, $V_1 \cap V_2 = \emptyset$.

Определение. Разрезом (коциклом) $P(V_1, V_2)$ называется множество ребер, соединяющих вершины из V_1 с вершинами из V_2 , т.е.

$$P(V_1, V_2) = \{(v_1, v_2) \in E : v_1 \in V_1, v_2 \in V_2\}.$$

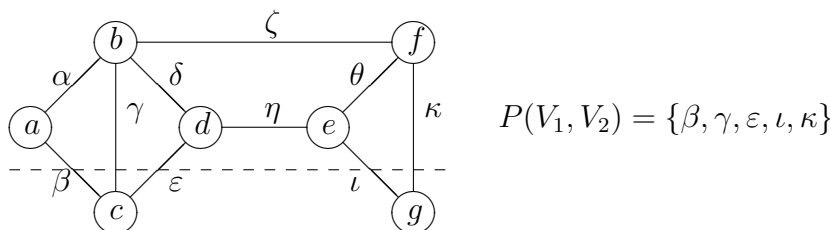
Удаление из графа всех ребер, принадлежащих разрезу, делает граф несвязным. В результате получается граф, являющийся объединением правильных подграфов исходного графа, заданных множествами вершин V_1 и V_2 .

Пример. Рассмотрим граф из предыдущих примеров и его разрез $P(V_1, V_2)$, где $V_1 = \{a, b, c\}$, $V_2 = \{d, e, f, g\}$:



Определение. Разрез называется *правильным*, если удаление любого его подмножества из графа не приводит к нарушению связности графа.

Примеры. Разрез из предыдущего примера является правильным. Рассмотрим другой разрез в том же графе: $V_1 = \{a, b, d, e, f\}$, $V_2 = \{c, g\}$.



Данный разрез не является правильным, он представляет собой объединение правильных разрезов $P_1 = \{\beta, \gamma, \varepsilon\}$ и $P_2 = \{\iota, \kappa\}$.

Из определения следует, что удаление из связного графа правильного разреза разбивает граф ровно на две компоненты связности.

Лемма. *Любой разрез может быть представлен в виде объединения непересекающихся правильных разрезов.*

Доказательство. Пусть $G(V, E)$ – связный граф, $V = U \cup W$, $U \cap W = \emptyset$, и $P = P(U, W)$ – неправильный разрез. В результате его удаления получается несвязный граф вида

$$G'(V, E \setminus P) = G'(U_1) \cup \dots \cup G'(U_n) \cup G'(W),$$

где $n > 1$, $G'(U_1), \dots, G'(U_n)$ – компоненты связности графа $G'(V, E \setminus P)$, причем $U_1 \cup \dots \cup U_n = U$, $G'(W)$ – правильный (возможно, несвязный) подграф графа $G(V, E)$, построенный на вершинах множества W . Поскольку в графе $G(V, E)$ нет ребер, соединяющих вершины множеств U_i и U_j при $i \neq j$, то

$$P(U_i, V \setminus U_i) = \{(u, v) : u \in U_i, v \in W\},$$

и разрез P может быть представлен в виде объединения непересекающихся разрезов

$$P(U, W) = P(U_1, V \setminus U_1) \cup \dots \cup P(U_n, V \setminus U_n).$$

Рассмотрим теперь разрез $P_1 = P(U_1, V \setminus U_1)$. Если он является неправильным, то его удаление разбивает граф $G(V, E)$ на более чем две компоненты связности, в результате получается граф $G_1(V, E \setminus P_1)$ вида

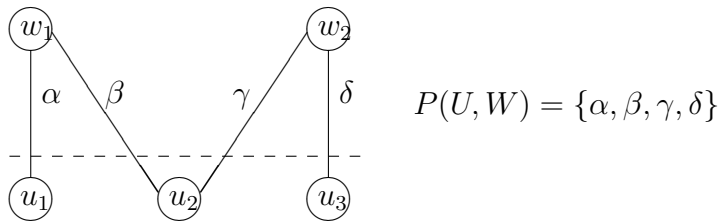
$$G_1(V, E \setminus P_1) = G'(U_1) \cup G'(X_1) \cup \dots \cup G'(X_m),$$

где $G'(U_1), G'(X_1), \dots, G'(X_m)$ – компоненты связности графа $G_1(V, E \setminus P_1)$. Разрез P может быть представлен в виде объединения непересекающихся разрезов

$$P(U_1, V \setminus U_1) = P(U_1, X_1) \cup \dots \cup P(U_n, X_m).$$

Граф $G(V, E)$ является связным, и не содержит ребер, соединяющих вершины множеств X_i и X_j при $i \neq j$. Значит, существует хотя бы одно ребро вида (u_i, x_i) , где $u_i \in U_1$, $x_i \in X_i$ для каждого значения i . Следовательно, после удаления любого разреза $P(U_1, X_i)$ граф $G(V, E \setminus P(U_1, X_i))$ будет содержать ровно две компоненты связности: $G'(X_i)$ и $G'(V \setminus X_i)$, а удаление любого подмножества этого разреза не приведет к нарушению связности, значит, $P(U_1, X_i)$ – правильный разрез. Повторяя данные рассуждения для всех разрезов P_i , получаем представление исходного разреза в виде объединения непересекающихся правильных разрезов.

Пример. Рассмотрим граф $G(V, E)$ и его разрез $P(U, W)$, где $U = \{u_1, u_2, u_3\}$, $W = \{w_1, w_2\}$:

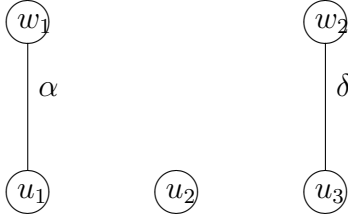


Разрез не является правильным, поскольку после его удаления получается граф с

пятью компонентами связности. Множество вершин U разбивается на три вершины, каждая из которых образует компоненту связности. Согласно доказательству теоремы, разрез $P(U, W)$ можно представить в виде объединения разрезов:

$$\begin{aligned} P(U, W) &= P(\{u_1\}, \{u_2, u_3, w_1, w_2\}) \cup P(\{u_2\}, \{u_1, u_3, w_1, w_2\}) \cup \\ &\cup P(\{u_3\}, \{u_1, u_2, w_1, w_2\}); \\ P(\{u_1\}, \{u_2, u_3, w_1, w_2\}) &= \{\alpha\}; \\ P(\{u_2\}, \{u_1, u_3, w_1, w_2\}) &= \{\beta, \gamma\}; \\ P(\{u_3\}, \{u_1, u_2, w_1, w_2\}) &= \{\delta\}. \end{aligned}$$

Из этих разрезов $P(\{u_2\}, \{u_1, u_3, w_1, w_2\})$ не является правильным, его удаление приводит к графу вида



Согласно доказательству теоремы, разрез $P(\{u_2\}, \{u_1, u_3, w_1, w_2\})$ можно представить в виде объединения разрезов:

$$\begin{aligned} P(\{u_2\}, \{u_1, u_3, w_1, w_2\}) &= P(\{w_1, u_1\}, \{u_2, u_3, w_2\}) \cup P(\{w_2, u_3\}, \{u_1, u_2, w_1\}); \\ P(\{w_1, u_1\}, \{u_2, u_3, w_2\}) &= \{\beta\}; \\ P(\{w_2, u_3\}, \{u_1, u_2, w_1\}) &= \{\gamma\}. \end{aligned}$$

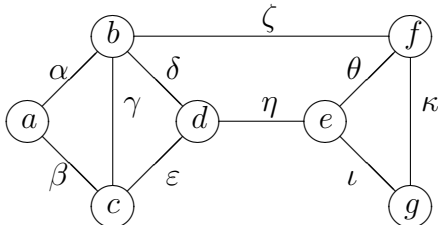
Итак, получаем:

$$\begin{aligned} P(U, W) &= P(\{u_1\}, \{u_2, u_3, w_1, w_2\}) \cup P(\{w_1, u_1\}, \{u_2, u_3, w_2\}) \cup \\ &\cup P(\{w_2, u_3\}, \{u_1, u_2, w_1\}) \cup P(\{u_3\}, \{u_1, u_2, w_1, w_2\}). \end{aligned}$$

Поскольку любой разрез представляет собой множество ребер, для разрезов также используют понятие линейной комбинации, введенное в предыдущем подразделе.

Определение. Множество разрезов $\{\psi_i\}_{i=1, \dots, n}$ называется *независимым*, если ни один разрез не является линейной комбинацией остальных, иначе множество называется *зависимым*.

Пример. Рассмотрим граф из предыдущих примеров и множество разрезов $\{\psi_1, \psi_2, \psi_3, \psi_4\}$

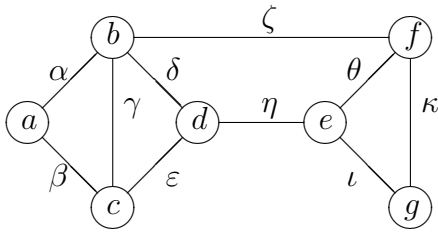


$$\begin{aligned} \psi_1 &= \{\beta, \gamma, \varepsilon, \zeta, \eta\}; \\ \psi_2 &= \{\zeta, \eta, \iota, \kappa\}; \\ \psi_3 &= \{\beta, \gamma, \varepsilon, \iota, \kappa\}; \\ \psi_4 &= \{\zeta, \eta, \theta\}. \end{aligned}$$

Это множество разрезов зависимо, поскольку $\psi_3 = \psi_1 \oplus \psi_2$, множество $\{\psi_1, \psi_2, \psi_4\}$ независимо.

Определение. Независимое множество разрезов максимальной мощности называется *системой фундаментальных разрезов*, а разрезы этой системы – *фундаментальными разрезами*.

Пример. Рассмотрим множество разрезов $\psi = \{\psi_1, \psi_2, \psi_3, \psi_4, \psi_5, \psi_6\}$ графа из предыдущих примеров:



$$\begin{aligned}\psi_1 &= \{\alpha, \beta\}; \\ \psi_2 &= \{\delta, \epsilon, \zeta\}; \\ \psi_3 &= \{\zeta, \eta\}; \\ \psi_4 &= \{\zeta, \theta, \iota\}; \\ \psi_5 &= \{\iota, \kappa\}; \\ \psi_6 &= \{\beta, \gamma, \epsilon\}.\end{aligned}$$

Множество ψ независимо, поскольку каждый разрез содержит ребро, которое отсутствует в остальных разрезах (например, в разрезе ψ_1 таким ребром является α , в ψ_2 – δ , и т.д.). Это означает, что никакой разрез не является линейной комбинацией остальных разрезов. Множество ψ является системой фундаментальных разрезов, поскольку любой разрез в графе может быть представлен как линейная комбинация разрезов системы. Например:

$$\begin{aligned}\{\alpha, \gamma, \delta, \theta, \kappa\} &= \psi_1 \oplus \psi_2 \oplus \psi_4 \oplus \psi_5 \oplus \psi_6; \\ \{\alpha, \gamma, \epsilon\} &= \psi_1 \oplus \psi_6; \\ \{\beta, \gamma, \epsilon, \iota, \kappa\} &= \psi_5 \oplus \psi_6.\end{aligned}$$

Пример. Для графа из предыдущих примеров линейная комбинация циклов $\{\alpha, \beta, \gamma\} \oplus \{\theta, \iota, \kappa\} = \{\alpha, \beta, \gamma, \theta, \iota, \kappa\}$ циклом не является.

Теорема о фундаментальных разрезах. Пусть $G(V, E)$ – простой связный граф. Тогда число его фундаментальных разрезов равно $\rho(G) = p - 1$.

Доказательство (конструктивное). Рассмотрим граф $G(V, E)$ и некоторое его остовное дерево $T(V, E')$. По теореме о шести эквивалентных утверждениях о дереве имеем, что число ребер в дереве равно $\rho(G) = p - 1$.

Построим множество разрезов $\{Y_1, \dots, Y_{p-1}\}$ следующим образом. Выбираем очередное ребро $e_i \in E'$. По теореме о шести эквивалентных утверждениях о дереве, это ребро является мостом, а значит, его удаление разбивает дерево на две компоненты связности. Пусть U_i, W_i – множества вершин в этих компонентах. Построим разрез $Y_i = P(U_i, W_i)$, этот разрез будет правильным, поскольку множества вершин U_i и W_i связны в остовном дереве. Заметим, что в разрезе будет единственное ребро, принадлежащее дереву, а именно, ребро $e_i = (u_i, w_i)$, поскольку в дереве нет другой цепи, соединяющей вершины u_i и w_i . Значит, построенное множество разрезов независимо, т.к. в каждом разрезе Y_i имеется ребро e_i , которого нет в других разрезах.

Покажем теперь, что любой разрез ψ может быть представлен в виде линейной комбинации разрезов $\{Y_1, \dots, Y_{p-1}\}$. Рассмотрим разрез $\psi = P(U, W)$. Он содержит ребра из множества E' , т.к. дерево – связный граф, и между любыми двумя вершинами графа существует цепь из ребер этого дерева. Пусть ψ содержит n таких ребер $\{e_1, \dots, e_n\}$, тогда

$$\psi = Y_1 \oplus \dots \oplus Y_n.$$

Докажем это соотношение с помощью метода математической индукции. Для $n = 1$ разрез $\psi = P(U_1, W_1) = Y_1$. Пусть соотношение верно для разрезов, содержащих $1, 2, \dots, n - 1$ ребер из дерева.

Выберем ребро e_n в разрезе $\psi = P(U, W)$ таким образом, что правильный подграф, образованный множеством вершин U_n , не содержит ребер e_1, \dots, e_n , т.е. множество $U_n \subset U$. Поскольку множество $U = U_n \oplus (U \setminus U_n)$, разрез имеет вид

$$P(U, W) = \{(x, y) : x \in U_n, y \in W\} \oplus \{(x, y) : x \in U \setminus U_n, y \in W\}$$

Поскольку множество $W_n = W \oplus (U \setminus U_n)$, разрез $Y_n = P(U_n, W_n)$ имеет вид

$$P(U_n, W_n) = \{(x, y) : x \in U_n, y \in W\} \oplus \{(x, y) : x \in U_n, y \in U \setminus U_n\}.$$

Рассмотрим разрез $\psi' = P(U', W')$, содержащий ребра e_1, \dots, e_{n-1} . Поскольку ребро e_n не принадлежит разрезу ψ' , то обе его концевые вершины лежат в одном из множеств U' или W' , для определенности, выберем W' . Поскольку ребро e_n выбиралось таким образом, что $U_n \subset U$ и не содержит ребер из разреза $\psi' = P(U', W')$, то $U' = U \setminus U_n$, $W' = W \oplus U_n$, и разрез ψ' можно представить в виде

$$P(U', W') = \{(x, y) : x \in U \setminus U_n, y \in W\} \oplus \{(x, y) : x \in U \setminus U_n, y \in U_n\}.$$

Рассмотрим линейную комбинацию разрезов $\psi' = P(U', W')$ и $Y_n = P(U_n, W_n)$:

$$\begin{aligned} & P(U', W') \oplus P(U_n, W_n) = \\ & = \{(x, y) : x \in U \setminus U_n, y \in W\} \oplus \{(x, y) : x \in U \setminus U_n, y \in U_n\} \oplus \\ & \quad \oplus \{(x, y) : x \in U_n, y \in W\} \oplus \{(x, y) : x \in U_n, y \in U \setminus U_n\} = \\ & = \{(x, y) : x \in U \setminus U_n, y \in W\} \oplus \{(x, y) : x \in U_n, y \in W\} = P(U, W). \end{aligned}$$

По индукционному предположению

$$\psi' = P(U', W') = Y_1 \oplus \dots \oplus Y_{n-1},$$

отсюда и из предыдущего равенства получаем

$$P(U, W) = P(U', W') \oplus P(U_n, W_n) = Y_1 \oplus \dots \oplus Y_{n-1} \oplus Y_n,$$

что и требовалось доказать.

Алгоритм построения системы фундаментальных разрезов (основан на доказательстве теоремы о фундаментальных разрезах)

Начало. Задан связный нетривиальный граф $G(V, E)$, требуется построить его систему фундаментальных разрезов.

Шаг 1. Строим произвольное остовное дерево $T(V, E')$ графа. Полагаем $i = 0$.

Шаг 2. Если $i = p - 1$, идем на конец. Иначе полагаем $i = i + 1$.

Шаг 3. Выбираем очередное ребро $e_i \in E'$ из остовного дерева. Удаляем его из дерева. В результате получаем лес с двумя компонентами связности, одна из которых образована множеством вершин U_i , а другая – множеством вершин W_i .

Шаг 4. Получаем разрез Y_i как множество ребер, соединяющих вершины из U_i с вершинами из W_i , т.е.

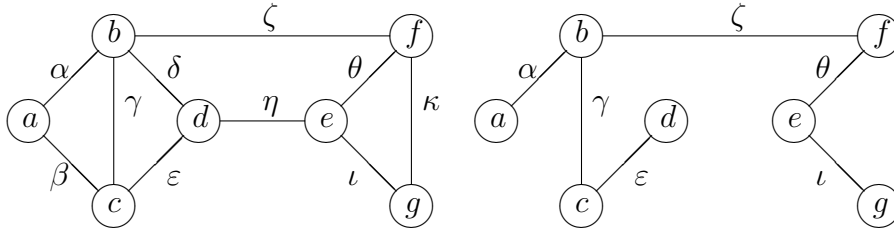
$$Y_i = \{(u, w) \in E : u \in U_i, w \in W_i\}.$$

Идем на шаг 2.

Конец. Система фундаментальных разрезов построена.

Заметим, что все фундаментальные разрезы, полученные с помощью данного алгоритма, являются правильными.

Пример. Слева – граф из предыдущих примеров, справа – его остовное дерево. Построим его систему фундаментальных разрезов.

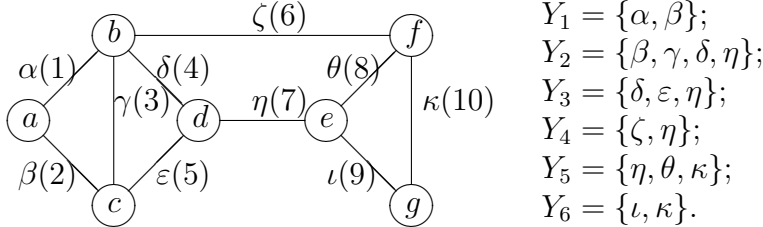


Выбираем очередное ребро e_i , входящее в остовное дерево, выясняем состав множеств U_i, W_i и получаем разрез $Y_i = P(U_i, W_i)$:

$$\begin{aligned}
 e_1 = \alpha = (a, b), & \quad U_1 = \{a\}, & \quad W_1 = \{b, c, d, e, f, g\}, & \quad Y_1 = \{\alpha, \beta\}; \\
 e_2 = \gamma = (b, c), & \quad U_2 = \{a, b, f, e, g\}, & \quad W_2 = \{c, d\}, & \quad Y_2 = \{\beta, \gamma, \delta, \eta\}; \\
 e_3 = \varepsilon = (c, d), & \quad U_3 = \{a, b, c, e, f, g\}, & \quad W_3 = \{d\}, & \quad Y_3 = \{\delta, \varepsilon, \eta\}; \\
 e_4 = \zeta = (b, f), & \quad U_4 = \{a, b, c, d\}, & \quad W_4 = \{e, f, g\}, & \quad Y_4 = \{\zeta, \eta\}; \\
 e_5 = \theta = (e, f), & \quad U_5 = \{e, g\}, & \quad W_5 = \{a, b, c, d, f\}, & \quad Y_5 = \{\eta, \theta, \kappa\}; \\
 e_6 = \iota = (e, g), & \quad U_6 = \{a, b, c, d, e, f\}, & \quad W_6 = \{g\}, & \quad Y_6 = \{\iota, \kappa\}.
 \end{aligned}$$

Определение. Матрицей фундаментальных разрезов графа $G(V, E)$ называется матрица $\Psi(G)$ размерности $(p-1) \times q$, строками которой являются булевы вектора, соответствующие фундаментальным разрезам.

Пример. Рассмотрим граф из предыдущих примеров и его систему фундаментальных разрезов.



Матрица фундаментальных разрезов имеет вид

$$\Psi(G) = \begin{array}{c|cccccccccc} & \alpha & \beta & \gamma & \delta & \varepsilon & \zeta & \eta & \theta & \iota & \kappa \\ \hline Y_1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ Y_2 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ Y_3 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ Y_4 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ Y_5 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ Y_6 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{array}$$

Теорема о матрице фундаментальных разрезов. Если $\Phi(G)$ – матрица фундаментальных циклов графа $G(V, E)$, $\Psi(G)$ – его матрица фундаментальных разрезов, то $\Phi \oplus \Psi^T = 0$, 0 – матрица, состоящая из нулей.

Доказательство. Элементы матрицы $C = \Phi \oplus \Psi^T$ имеют вид

$$C_{ij} = \bigoplus_{k=1}^q \Phi_{ik} \Psi_{jk}.$$

Здесь $I_{ik}\Phi_{jk} = 1$, если и только если $\Phi_{ik} = 1$ и $\Psi_{jk} = 1$, т.е. ребро k входит в фундаментальный цикл Z_i и фундаментальный разрез Y_j . Для доказательства теоремы достаточно показать, что фундаментальный разрез может содержать лишь четное число ребер из фундаментального цикла. Пусть фундаментальный цикл имеет вид $v_1e_1v_2e_2 \dots v_n e_n v_1$, $v_i \in V$, где $e_i = (v_i, v_{i+1})$ для $i = 1, \dots, n-1$, и $e_n = (v_n, v_1)$. Пусть фундаментальному разрезу $P(U, W)$ принадлежат ребра $(e_{i_1}, \dots, e_{i_k})$, для определенности $e_{i_k} = e_n$. Тогда $\{v_1, \dots, v_{i_1}\} \subset U$, \dots , $\{v_{i_{k-1}+1}, \dots, v_{i_k}\} \subset W$, и т.д. Если число ребер k нечетно, то получаем, что $\{v_{i_{k+1}}, \dots, v_n\} \subset U$. Тогда $v_1 \in U$, $v_n \in V$, что противоречит тому, что $e_n = (v_1, v_n)$ принадлежит разрезу. Значит, число общих ребер разреза и цикла либо четное, либо равно нулю, откуда

$$\bigoplus_{k=1}^q \Phi_{ik}\Psi_{jk} = 0,$$

что и требовалось доказать.

Теорема позволяет получить матрицу фундаментальных разрезов по матрице фундаментальных циклов, и наоборот. Для этого достаточно решить уравнение $\Phi \oplus \Psi^T = 0$.

2.8 Теорема Менгера

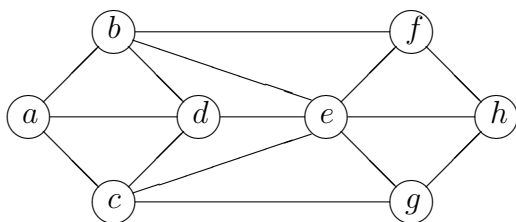
Интуитивно понятно, что граф тем более связан, чем больше существует различных простых цепей, соединяющих одну вершину с другой, и тем менее связан, чем меньше нужно удалить вершин, чтобы отделить одну вершину от другой. Рассмотрим теорему Менгера, в которой эти наблюдения принимают строгую форму.

Пусть $G(V, E)$ – связный граф, u и v – две его не смежные вершины.

Определение. Две цепи $\langle u, v \rangle$ называются *вершинно-непересекающимися*, если у них нет общих вершин, кроме начальной и конечной.

Определение. Две цепи $\langle u, v \rangle$ называются *реберно-непересекающимися*, если у них нет общих ребер.

Пример. Цепи $abefh$, $acgeh$ – реберно-непересекающиеся, но они не являются вершинно-непересекающимися (имеют общую вершину e). Цепи $abfh$, $adeh$ и $acgh$ являются вершинно-непересекающимися.



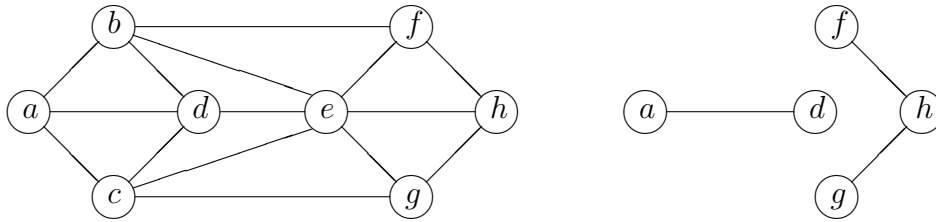
Если две цепи вершинно не пересекаются, то они и реберно не пересекаются.

Будем обозначать через $P(u, v)$ множество вершинно-непересекающихся простых цепей $\langle u, v \rangle$.

Определение. Множество вершин (и/или ребер) графа G называется *разделяющим множеством*, если удаление из графа G элементов множества приводит к тому, что вершины u и v оказываются в разных компонентах связности.

Будем обозначать разделяющее множество вершин для u, v через $S(u, v)$.

Пример. Для графа из предыдущего примера $S(a, h) = \{b, c, e\}$ (можно найти и другие разделяющие множества). Справа показан исходный граф, слева – он же после удаления множества $S(a, h) = \{b, c, e\}$.



Легко видеть, что $|P(u, v)| \leq |S(u, v)|$. Действительно, любая цепь $\langle u, v \rangle$ проходит через одну из вершин множества $S(u, v)$, иначе после удаления этого множества в графе осталась бы цепь $\langle u, v \rangle$, не проходящая через вершины $S(u, v)$, и это множество не являлось бы разделяющим. Поэтому если $|P(u, u)| > |S(u, v)|$, то в $P(u, v)$ есть по крайней мере две цепи, проходящие через одну вершину множества $S(u, v)$, что противоречит выбору множества $P(u, v)$. Следовательно,

$$\max |P(u, v)| \leq \min |S(u, v)|.$$

Теорема Менгера утверждает, что в любом графе существует такое разделяющее множество вершин $S(u, v)$, которое дает точное равенство.

Теорема (Менгера). Пусть $G(V, E)$ – связный граф, u, v – две его не смежные вершины. Наименьшее число вершин в множестве, разделяющем u и v , равно наибольшему числу вершинно-непересекающихся простых цепей, соединяющих u, v , т. е.

$$\max |P(u, v)| = \min |S(u, v)|.$$

Доказательство. Используем метод математической индукции по p и q . База индукции: $V = \{u, v, w\}$, $E = \{(u, w), (w, v)\}$, тогда $\{P(u, v)\} = \{\langle u w v \rangle\}$, $\{S(u, v)\} = \{\{w\}\}$, и $\max |P(u, v)| \leq \min |S(u, v)| = 1$.

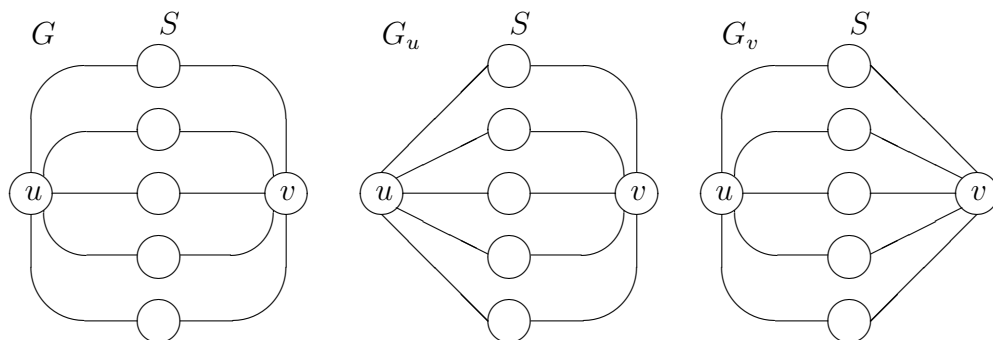


Пусть утверждение верно для всех графов с числом вершин, меньшим p , и числом ребер, меньшим q . Рассмотрим граф $G(V, E)$ с p вершинами и q ребрами. Выберем наименьшее разделяющее множество вершин $S(u, v)$, состоящее из n вершин, и рассмотрим три случая.

Случай 1. Пусть в S есть вершины, не смежные с u , и вершины, не смежные с v . Удалим из $G(V, E)$ множество S , получим несвязный граф $G'(V', E')$, который состоит из двух нетривиальных графов $G_1(V_1, E_1)$ и $G_2(V_2, E_2)$, где $u \in V_1, v \in V_2$. Стянем эти графы к вершинам u и v соответственно, получим графы $G_u(V_u, E_u)$ и $G_v(V_v, E_v)$ (т.е. граф $G_u(V_u, E_u)$ получается из $G(V, E)$ стягиванием подграфа $G_1(V_1, E_1)$ к вершине u , граф $G_v(V_v, E_v)$ получается из $G(V, E)$ стягиванием подграфа $G_2(V_2, E_2)$ к вершине v).

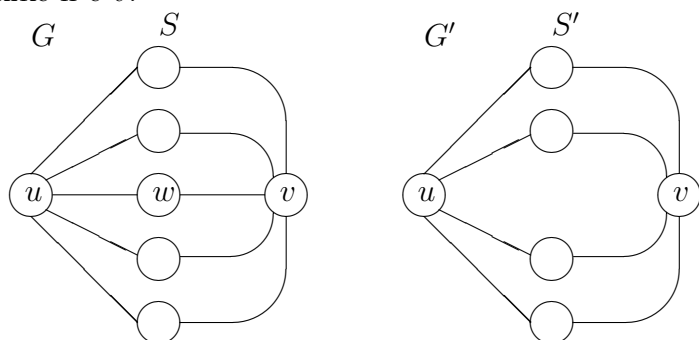
В полученных графах S остается наименьшим разделяющим множеством для u и v . Так как G_1 и G_2 нетривиальны, то G_u и G_v имеют меньше вершин и/или

ребер, чем G , т. е. к ним применимо индукционное предположение – в каждом из этих графов существует множество из ровно n вершинно-непересекающихся простых цепей $\langle u, v \rangle$.



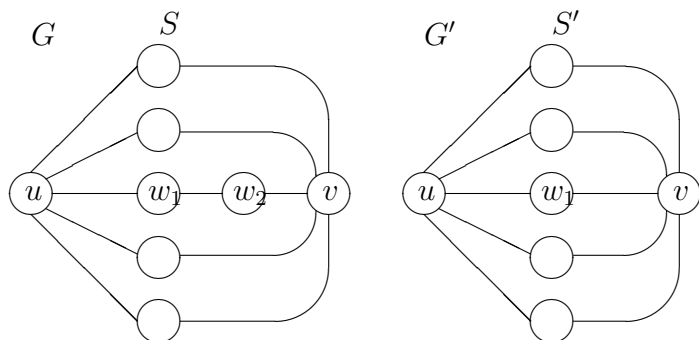
Каждая из этих цепей в каждом графе проходит через свою вершину множества S . Соединив отрезки цепей от u до вершин множества S в G_v с соответствующими отрезками цепей от вершин множества S до v в G_u , получим n вершинно-непересекающихся простых цепей $\langle u, v \rangle$ в исходном графе.

Случай 2. Пусть все вершины множества S смежны с u или v (выберем для определенности u), и среди вершин этого множества есть вершина w , смежная также и с v .



Рассмотрим граф $G'(V', E')$, полученный из $G(V, E)$ удалением вершины w . В нем $S' = S \setminus \{w\}$ – наименьшее разделяющее множество мощности $n - 1$. По индукционному предположению в $G'(V', E')$ имеется $n - 1$ вершинно-непересекающихся простых цепей. Добавим к ним цепь $u w v$, она простая и не пересекается с остальными. Итак, в $G(V, E)$ имеется n вершинно-непересекающихся простых цепей.

Случай 3. Пусть теперь все вершины множества S смежны с u или v (выберем для определенности u), и среди вершин этого множества нет вершин, смежных одновременно с u и v . Рассмотрим кратчайшую цепь из этого множества $u w_1 w_2 \dots v$, $w_1 \in S$, $w_2 \neq v$. Стянем вершины w_1 и w_2 в вершину w_1 , получим граф $G'(V', E')$.



Вершина $w_2 \notin S$, иначе выбранная цепь была бы не кратчайшей. Следовательно, в графе $G'(V', E')$ множество S остается разделяющим множеством вершин наименьшей мощности, и этот граф содержит меньше вершин и ребер (по крайней мере на одно ребро), чем исходный граф. Значит, для $G'(V', E')$ верно индукционное предположение, и в нем существует n вершинно-непересекающихся простых цепей $\langle u, v \rangle$. Эти цепи не пересекаются и в $G(V, E)$, отсюда, добавив в кратчайшую цепь удаленное ранее ребро, получаем n вершинно-непересекающихся простых цепей $\langle u, v \rangle$ в графе G .

Теорема доказана.

Существуют и другие результаты, подобные теореме Менгера.

Теорема. Для любых двух не смежных вершин u и v графа $G(V, E)$ наибольшее число реберно-непересекающихся цепей $\langle u, v \rangle$ равно наименьшему числу ребер в (u, v) разрезе.

Теорема. Чтобы граф $G(V, E)$ был k -связным, необходимо и достаточно, чтобы любые две не смежные вершины были соединены не менее чем k вершинно-непересекающимися простыми цепями, и существовали хотя бы две несмежные вершины, соединенные ровно k вершинно-непересекающимися простыми цепями.

2.9 Связность орграфов

Связность является одним из немногих понятий, которые не распространяются непосредственно с графов на другие родственные объекты и требуют отдельного определения и рассмотрения.

Определение. Говорят, что два узла v_1 и v_2 *сильно связаны* в орграфе $D(V, E)$, если существует путь из v_1 в v_2 и из v_2 в v_1 .

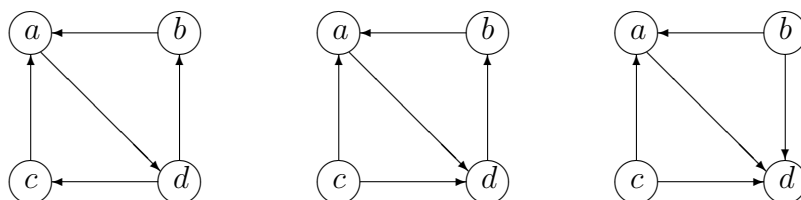
Определение. Говорят, что два узла v_1 и v_2 *односторонне связаны* в орграфе $D(V, E)$, если существует путь либо из v_1 в v_2 , либо из v_2 в v_1 .

Определение. Говорят, что два узла v_1 и v_2 *слабо связаны* в орграфе $D(V, E)$, если они связаны в графе $D'(V', E')$, полученном из $D(V, E)$ отменой ориентации ребер.

Определение. Орграф $D(V, E)$ называется *сильно (односторонне, слабо) связным*, если любые два узла в нем сильно (односторонне, слабо) связаны.

Сильная связность влечет одностороннюю связность, которая влечет слабую связность. Обратное неверно.

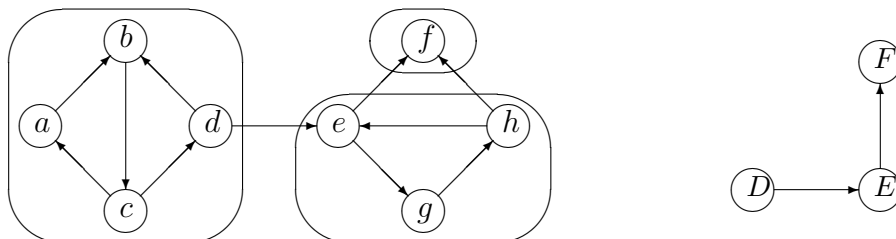
Пример. Слева – сильно связный орграф, в центре – односторонне связный (нет путей до узла c из остальных), справа – слабо связный (нет путей между узлами b и c).



Определение. Компонентой сильной связности орграфа $D(V, E)$ называется его правильный сильно связный подграф, не являющийся собственным подграфом никакого другого сильно связного подграфа орграфа $D(V, E)$.

Определение. *Конденсацией (фактор-графом) орграфа $D(V, E)$* называется орграф, который получается стягиванием в один узел каждой компоненты сильной связности орграфа $D(V, E)$.

Пример. Орграф (слева, на рисунке выделены компоненты сильной связности) и его конденсация (справа). Здесь в узел D стянута компонента сильной связности, содержащая узлы $\{a, b, c, d\}$, в узел E – компонента сильной связности, содержащая узлы $\{e, g, h\}$, в узел F – компонента сильной связности, содержащая узел $\{f\}$.



Определение. *Матрицей достижимости орграфа $D(V, E)$* называется квадратная матрица $T(D)$ размерности $p \times p$, где

$$T_{ij} = \begin{cases} 1, & \text{если } \exists \langle i, j \rangle; \\ 0, & \text{иначе.} \end{cases}$$

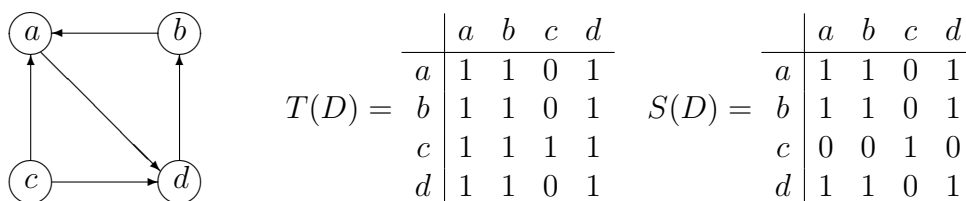
Считается, что путь $\langle i, i \rangle$ существует всегда (это путь длины 0).

Определение. *Матрицей сильной связности орграфа $D(V, E)$* называется квадратная матрица $S(D)$ размерности $p \times p$, где

$$S_{ij} = \begin{cases} 1, & \text{если } \exists \langle i, j \rangle, \langle j, i \rangle; \\ 0, & \text{иначе.} \end{cases}$$

Иначе говоря, $S_{ij} = 1$ тогда и только тогда, когда узлы i и j принадлежат одной компоненте сильной связности орграфа $D(V, E)$.

Пример. Рассмотрим орграф $D(V, E)$, его матрицу достижимости и сильной связности.



Изучим алгоритмы построения матриц достижимости и сильной связности.

Введем операции для булевых матриц. Первые две аналогичны операциям сложения и умножения матриц, но сложение и умножение элементов заменяются соответственно конъюнкцией и дизъюнкцией.

Определение. *Дизъюнкцией булевых матриц $A : n \times m$ и $B : n \times m$* называется булева матрица $C : n \times m$, вычисляемая по формуле:

$$C = A \vee B : C_{ij} = A_{ij} \vee B_{ij}.$$

Определение. Булевым произведением булевых матриц $A : n \times k$ и $B : k \times t$ называется булева матрица $C : n \times t$, вычисляемая по формуле:

$$C = AB : C_{ij} = \bigvee_{l=k}^m A_{il}B_{lj}.$$

Определение. Конъюнкцией булевых матриц $A : n \times t$ и $B : n \times t$ называется булева матрица $C : n \times t$, вычисляемая по формуле:

$$C = A \& B : C_{ij} = A_{ij}B_{ij}.$$

Определение. Степенью n квадратной булевой матрицы $A : t \times t$ называется квадратная булева матрица $A^n : t \times t$

$$A^n = \underbrace{AA \dots A}_n$$

Утверждение 1. Пусть $D(V, E)$ – орграф с матрицей смежности M . Тогда элемент M_{ij}^k равен единице, тогда и только тогда, когда $\exists \langle i, j \rangle$ длины k .

Доказательство (метод математической индукции). При $k = 1$ утверждение верно. Пусть оно верно для $1, 2, \dots, k - 1$. Если $M_{ij}^k = 1$, это означает, что $\exists l : M_{il}^{k-1} = 1, M_{lj} = 1$, т. е. существует путь $\langle i, l \rangle$ длины $k - 1$, и существует дуга $\{l, j\}$. Значит, $\exists \langle i, j \rangle$ длины k , который проходит через вершину l .

Утверждение 2. Пусть $D(V, E)$ – орграф с матрицей смежности M . Тогда

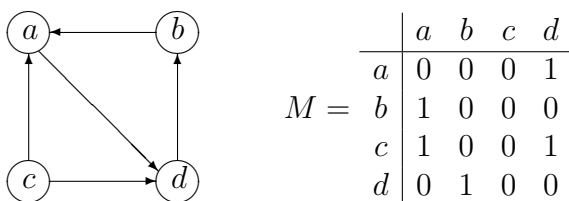
$$\begin{aligned} T(D) &= E \vee M \vee M^2 \vee \dots \vee M^{p-1}; \\ S(D) &= T(D) \& T^T(D) \quad (S_{ij} = T_{ij}T_{ji}), \end{aligned}$$

где $E : p \times p$ – единичная матрица.

Доказательство. Из леммы о простой цепи следует, что если существует путь $\langle i, j \rangle$, то существует и простой путь, вершины в котором не повторяются, а значит, его длина не больше $p - 1$, и тогда $T_{i,j} = 1$. Так как каждая вершина достижима из самой себя, добавляем единицы на главной диагонали. Узлы i и j сильно связны, если $\exists \langle i, j \rangle, \langle j, i \rangle$, т. е. $S_{ij} = T_{ij}T_{ji}$.

Данные соотношения дают алгоритм вычисления матриц достижимости и сильной связности, который состоит в последовательном вычислении матриц M^2, M^3, \dots, M^{p-1} и дальнейшем их булевом сложении. Таким образом, требуется выполнить $p - 2$ булевых умножений матриц ($M^2 = MM, M^3 = M^2M, \dots, M^{p-1} = M^{p-2}M$) и $p - 1$ булевых сложений. Каждое сложение матриц требует порядка p^2 операций, каждое умножение – p^3 . Таким образом, вычислительная сложность алгоритма равна $(p - 2)p^3 + (p - 1)p^2 \approx p^4$.

Пример. Рассмотрим орграф из предыдущего примера и его матрицу смежности.



$$M = \begin{array}{c|cccc} & a & b & c & d \\ \hline a & 0 & 0 & 0 & 1 \\ b & 1 & 0 & 0 & 0 \\ c & 1 & 0 & 0 & 1 \\ d & 0 & 1 & 0 & 0 \end{array}$$

Вычислим последовательно матрицы M^2 и M^3 .

$$M^2 = MM = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

Действительно, в данном графе есть 5 путей длины 2: $\langle a, b \rangle = adb$, $\langle b, d \rangle = bad$, $\langle c, b \rangle = cdb$, $\langle c, d \rangle = cad$, $\langle d, a \rangle = dba$, в чем легко убедиться, используя диаграмму графа.

$$M^3 = M^2M = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Действительно, в данном графе есть 5 путей длины 3: $\langle a, a \rangle = adba$, $\langle b, b \rangle = badb$, $\langle c, a \rangle = cdba$, $\langle c, b \rangle = cadb$, $\langle d, d \rangle = dbad$, в чем легко убедиться, используя диаграмму графа. Подсчитаем матрицу достижимости.

$$T = E \vee M \vee M^2 \vee M^3 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \vee \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \end{bmatrix} \vee \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix} \vee \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 \end{bmatrix}.$$

Подсчитаем матрицу сильной связности.

$$S = T \& T^T = \begin{bmatrix} 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 \end{bmatrix} \& \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \end{bmatrix}$$

Матрицы T и S совпадают с полученными в предыдущем примере визуально по диаграмме графа.

Рассмотрим еще алгоритм поиска матрицы достижимости, обладающий меньшей вычислительной сложностью. Для этого нам понадобится следующее утверждение.

Утверждение 3. Пусть $D(V, E)$ – орграф с матрицей смежности M , а $B^{(0)}, \dots, B^{(p)}$ – последовательность матриц $p \times p$, элементы которых вычисляются по формулам

$$B^{(0)} = E \vee M; \\ B_{ij}^{(l)} = B_{ij}^{(l-1)} \vee B_{il}^{(l-1)} B_{lj}^{(l-1)}.$$

Тогда $T(D) = B^{(p)}$.

Доказательство. Докажем, что элемент матрицы $B_{ij}^{(l)} = 1$ для $i \neq j$ тогда и только тогда, когда существует путь $\langle i, j \rangle$, проходящий через узлы $1, \dots, l$. Здесь подчеркнем, что путь может проходить не через все вершины этого множества, но не может проходить через вершины, не содержащиеся в множестве. Для матрицы $B^{(0)}$ это верно, поскольку она содержит информацию о путях без промежуточных вершин (дугах и путях типа $\langle v, v \rangle$). Предположим, что это верно для матриц $B^{(0)}, \dots, B^{(l-1)}$. Элемент $B_{ij}^{(l)} = 1$, если и только если:

– $B_{ij}^{(l-1)} = 1$, т. е. по индукционному предположению существует путь $\langle i, j \rangle$, проходящий через узлы $1, \dots, l-1$;

– $B_{il}^{(l-1)} B_{lj}^{(l-1)} = 1$, т. е. по индукционному предположению существуют пути $\langle i, l \rangle$ и $\langle l, j \rangle$, проходящие через узлы $1, \dots, l-1$, но тогда объединение этих путей дает путь $\langle i, j \rangle$, проходящий через узлы $1, \dots, l$.

Отсюда следует, что $B_{ij}^{(p)} = 1$ для $i \neq j$ тогда и только тогда, когда существует путь $\langle i, j \rangle$, проходящий через узлы $1, \dots, p$, т. е. когда j достижима из i . Элементы $B_{ii}^{(p)} = 1$ по построению, значит, $B^{(p)}$ – матрица достижимости.

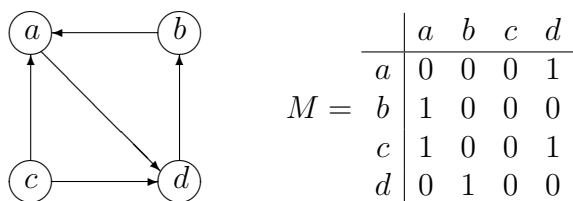
Алгоритм Уоршалла состоит в вычислении матриц $B^{(0)}, \dots, B^{(p)}$. Вычисление каждой матрицы требует порядка p^2 операций, поскольку для вычисления каждого элемента матрицы выполняется по одной операции дизъюнкции и конъюнкции, т.е. вычислительная сложность алгоритма равна $(p+1)p^2 \approx p^3$.

Матрицы $B^{(l)}$ могут вычисляться следующим образом:

$$B^{(l)} = B^{(l-1)} \vee D^{(l)}, \quad D_{ij}^{(l)} = B_{il}^{(l-1)} B_{lj}^{(l-1)},$$

т.е. в матрице $D^{(l)}$ единицы стоят на пересечении строк, содержащих 1 в l -том столбце матрицы $B^{(l-1)}$, и столбцов, содержащих 1 в l -той строке матрицы $B^{(l-1)}$.

Пример. Рассмотрим орграф из предыдущего примера и его матрицу смежности.



Вычислим последовательно матрицы $B^{(0)}, \dots, B^{(d)}$.

$$B^{(0)} = E \vee M = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \vee \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \end{bmatrix}$$

Матрица $B^{(0)}$ содержит единицы в столбце a в строках a, b, c и в строке a – в столбцах a, d . Это означает, что матрица $D^{(a)}$ будет содержать единицы на пересечении строк a, b, c и столбцов a, d .

$$B^{(a)} = B^{(0)} \vee D^{(a)} = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \end{bmatrix} \vee \begin{bmatrix} 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \end{bmatrix}$$

По сравнению с матрицей $B^{(0)}$ в матрице $B^{(a)}$ появляется единица в строке b и столбце d . Действительно, в графе есть путь $\langle b, d \rangle$, проходящий через узел a . Аналогично вычисляем остальные матрицы.

$$B^{(b)} = B^{(a)} \vee D^{(b)} = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \end{bmatrix} \vee \begin{bmatrix} 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \end{bmatrix}$$

Элемент $B_{da}^{(b)} = 1$, поскольку есть путь $\langle d, a \rangle$, проходящий через узел b .

$$B^{(c)} = B^{(b)} \vee D^{(c)} = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \end{bmatrix} \vee \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \end{bmatrix}$$

Матрица не изменилась, поскольку в графе нет путей, содержащих узел c в качестве промежуточного.

$$B^{(d)} = B^{(c)} \vee D^{(d)} = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \end{bmatrix} \vee \begin{bmatrix} 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 \end{bmatrix}$$

Элементы $B_{ab}^{(d)} = 1$ и $B_{cb}^{(d)} = 1$, поскольку пути $\langle a, b \rangle$ и $\langle c, b \rangle$ проходят через узел d . Матрица $B^{(d)}$ совпадает с матрицей достижимости T , найденной в предыдущих примерах.

Алгоритм выделения компонент сильной связности основан на том, что единицы в i -м столбце или i -й строке матрицы связности соответствуют узлам, содержащимся в той же компоненте сильной связности, что и узел i .

Начало. Полагаем $j = 0$. Все строки матрицы сильной связности S считаем неотмеченными.

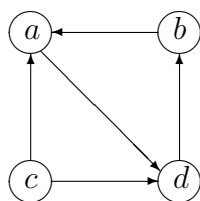
Шаг 1. Если все строки матрицы S отмечены, идем на конец. Иначе полагаем $j = j + 1$ и выбираем очередную неотмеченную строку i матрицы S . Записываем в новую компоненту связности K_j все узлы, соответствующие столбцам, содержащим единицы в строке i .

Шаг 2. Отмечаем строки, содержащие 1 в столбце i . Возвращаемся на шаг 1.

Конец. Найдены компоненты сильной связности графа K_1, \dots, K_j .

В ходе работы алгоритма каждая строка просматривается не более одного раза, и при просмотре строки все ее элементы проверяются на равенство единице. Следовательно, вычислительная сложность алгоритма не превышает число элементов матрицы S и равна p^2 .

Пример. Рассмотрим орграф из предыдущего примера и его матрицу сильной связности.



$$S = \begin{array}{c|cccc} & a & b & c & d \\ \hline a & 1 & 1 & 0 & 1 \\ b & 1 & 1 & 0 & 1 \\ c & 0 & 0 & 1 & 0 \\ d & 1 & 1 & 0 & 1 \end{array}$$

Просматриваем неотмеченную строку a . Заносим в компоненту сильной связности K_1 узлы a, b, d . Отмечаем эти строки.

$$S = \begin{array}{c|cccc} & a & b & c & d \\ \hline a & 1 & 1 & 0 & 1 & * \\ b & 1 & 1 & 0 & 1 & * \\ c & 0 & 0 & 1 & 0 & \\ d & 1 & 1 & 0 & 1 & * \end{array}$$

Просматриваем неотмеченную строку c . Заносим в компоненту сильной связности K_2 узел c и отмечаем строку c .

$$S = \begin{array}{c|cccc} & a & b & c & d \\ \hline a & 1 & 1 & 0 & 1 & * \\ b & 1 & 1 & 0 & 1 & * \\ c & 0 & 0 & 1 & 0 & * \\ d & 1 & 1 & 0 & 1 & * \end{array}$$

Неотмеченных строк нет, конец.

Примечание. Этот же алгоритм может использоваться и для выделения компонент связности простого графа.

2.10 Упражнения

1. Предложить алгоритм поиска простого цикла по матрице смежности неориентированного и ориентированного графа.

2. Предложить алгоритм поиска простого цикла по матрице инцидентий неориентированного и ориентированного графа.

3. Доказать, что любой граф без циклов является двудольным.

4. Доказать, что граф является двудольным тогда и только тогда, когда все его циклы имеют четную длину.

5. Каково наибольшее число ребер в графе с 7 вершинами, не содержащем циклов длины 3? Получить как можно более точные оценки сверху и снизу для этого числа.

6. Доказать, что граф и его дополнение не могут быть несвязными одновременно.

7. Привести примеры, если это возможно:

- графа, у которого точка сочленения не является концом моста;
- графа, у которого конец моста не является точкой сочленения;
- графа, у которого оба конца моста являются точками сочленения;
- графа, у которого нет точек сочленения;
- графа, у которого все вершины являются точками сочленения;
- графа, у которого все ребра являются мостами;
- графа, у которого имеется ровно две вершины, не являющиеся точкам сочленения.

8. Графы заданы матрицами смежности (здесь и далее нули в матрицах смежности опущены для упрощения записи).

	<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	<i>e</i>	<i>f</i>	<i>g</i>	<i>h</i>
<i>a</i>					1		1	1
<i>b</i>			1		1		1	
<i>c</i>		1		1	1			
<i>d</i>			1			1		
<i>e</i>	1	1	1				1	
<i>f</i>				1				1
<i>g</i>	1	1			1			1
<i>h</i>	1					1	1	

	<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	<i>e</i>	<i>f</i>	<i>g</i>	<i>h</i>
<i>a</i>		1	1				1	
<i>b</i>	1				1		1	1
<i>c</i>	1							
<i>d</i>						1	1	
<i>e</i>		1					1	
<i>f</i>				1				1
<i>g</i>	1	1		1	1			1
<i>h</i>		1				1	1	

Для каждого из графов выполнить следующие задания:

- найти диаметр, радиус и центры графа;
- определить числа вершинной и реберной связности графа;
- найти минимальное разделяющее множество вершин для *b, h*;
- найти максимальное множество вершинно-непересекающихся цепей для *b, h*;
- найти минимальное разделяющее множество ребер для *b, h*;
- найти максимальное множество реберно-непересекающихся цепей для *b, h*;
- найти пару вершин, для которых разделяющее множество вершин имеет наименьшую мощность;
- построить матрицы фундаментальных циклов и фундаментальных разрезов.

9. Для каждого из орграфов, заданных матрицами смежности, построить факторграф и определить тип связности.

	<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	<i>e</i>	<i>f</i>	<i>g</i>	<i>h</i>
<i>a</i>					1			1
<i>b</i>			1					
<i>c</i>							1	
<i>d</i>			1			1		
<i>e</i>		1	1					
<i>f</i>								
<i>g</i>		1		1	1			1
<i>h</i>						1		

	<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	<i>e</i>	<i>f</i>	<i>g</i>	<i>h</i>
<i>a</i>								1
<i>b</i>	1							
<i>c</i>								
<i>d</i>			1			1		
<i>e</i>		1	1					
<i>f</i>	1	1						
<i>g</i>	1				1	1		1
<i>h</i>		1		1		1		

10. Пусть орграф задан матрицей смежности:

	<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	<i>e</i>
<i>a</i>			1	1	
<i>b</i>	1				
<i>c</i>	1	1		1	
<i>d</i>					1
<i>e</i>			1		

- с помощью операций над булевыми матрицами выяснить, между какими вершинами графа существуют пути длины 3;
- с помощью алгоритма Уоршалла выяснить, между какими вершинами графа существуют пути, которые могут проходить только через вершины *a, d*.

3 Поиск путей в графе

Задача поиска путей является одной из классических задач теории графов, что обусловлено ее практической значимостью. Действительно, с поиском путей, оптимальных в каком-то смысле, мы сталкиваемся постоянно. Например, в GPS-навигаторах осуществляется поиск минимального по времени пути между двумя перекрестками, поиск кратчайшего пути используется трассировки электрических соединений на кристаллах микросхем и на печатных платах, и т.д.

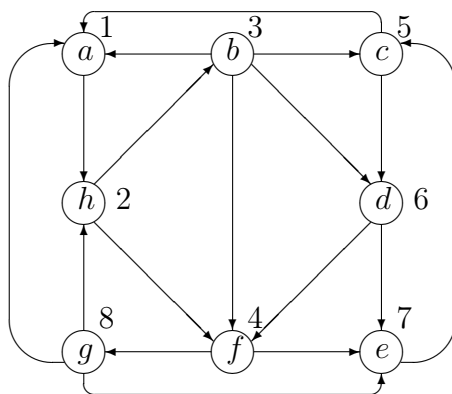
3.1 Стратегии обхода графа

Определение. *Обходом графа* называется перечисление его вершин.

Как правило, обходы используют локальную информацию о структуре графа, т. е. из текущей вершины доступны только смежные с ней вершины. Существуют две основных стратегии обхода – «в ширину» и «в глубину».

Стратегия обхода «в ширину». Начиная со стартовой вершины, обходим все смежные с ней вершины. Затем для каждой из них повторяем эту процедуру, заходя только в непосещенные вершины. Процесс заканчивается, если посещены все вершины, либо нет непосещенных вершин, смежных с уже посещенными.

Пример. Рассмотрим ориентированный граф. Цифры около вершин обозначают порядок обхода.



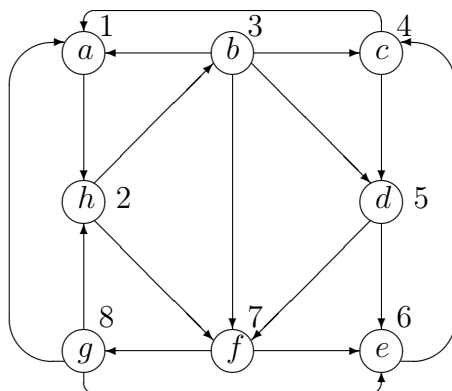
Начинаем обход с узла a . Множество смежности $\Gamma(a) = \{h\}$, значит, посещаем узел h . Множество смежности $\Gamma(h) = \{b, f\}$, значит, посещаем узлы b и f . Далее рассматриваем узел b , множество смежности $\Gamma(b) = \{a, c, d, f\}$. Узлы a и f уже посещались, значит, посещаем c и d . Далее рассматриваем узел f , множество смежности $\Gamma(f) = \{g, e\}$, эти узлы не посещались, значит, посещаем e и g . Далее рассматриваем по очереди узлы c, d, e, g , в том порядке, в котором они посещались. Из них достижимы только посещенные вершины. Обход закончен, порядок обхода имеет вид

$$ahbfcd eg.$$

Стратегия обхода «в глубину». Начиная со стартовой вершины, идем в смежную с ней вершину. Затем из этой вершины идем в смежную с ней непосещенную вершину, и т. д. Если нет непосещенных вершин, смежных с текущей, возвращаемся на шаг назад и идем в смежную непосещенную вершину, если нет

таких вершин, возвращаемся еще на шаг назад, и т. д. Процесс заканчивается, если посещены все вершины, либо мы вернулись в стартовую вершину и не осталось смежных с ней непосещенных вершин.

Пример. Рассмотрим ориентированный граф из предыдущего примера. Цифры около вершин обозначают порядок обхода.



Начинаем обход с узла a . Множество смежности $\Gamma(a) = \{h\}$, значит, посещаем узел h . Множество смежности $\Gamma(h) = \{b, f\}$, значит, посещаем узел b (договоримся в случае, когда имеется выбор, переходить в узел, первый по алфавиту). Далее рассматриваем узел b , множество смежности $\Gamma(b) = \{a, c, d, f\}$. Узел a уже посещался, значит, посещаем c . Множество смежности $\Gamma(c) = \{a, d\}$, узел a уже посещался, значит, переходим в узел d . Множество смежности $\Gamma(d) = \{e, f\}$, переходим в узел e . Множество смежности $\Gamma(e) = \{c\}$, этот узел уже посещался, возвращаемся в предыдущий узел d , из которого переходим в узел f . Множество смежности $\Gamma(f) = \{e, g\}$, узел e уже посещался, значит, переходим в узел g . Множество смежности $\Gamma(g) = \{a, e, h\}$, все эти узлы посещались, возвращаемся в узел f . Узлы e и g , смежные с f , уже посещались, возвращаемся в узел d , из которого пришли в f . Действуя аналогично, возвращаемся в узел c , затем в d , затем в h и в a . Из этих узлов недостижимы непосещенные узлы. Обход закончен, порядок обхода имеет вид

$$ahbcdefg.$$

Стратегии обхода применяются, в частности, для поиска путей между двумя вершинами. При этом алгоритмы поиска кратчайших путей опираются на идеи обхода «в ширину», стратегия обхода «в глубину» используется, когда требуется найти произвольный путь за наименьшее время.

3.2 Поиск кратчайших путей

Определение. *Кратчайшим путем* $\langle u, v \rangle$ в графе называется путь наименьшей длины.

Волновой алгоритм поиска кратчайших путей от вершины s до остальных вершин графа.

Начало. Полагаем номер волны $i = 0$ и помечаем вершину s индексом 0. Все вершины, кроме s , считаем непомяченными. Полагаем множество $V_0 = \{s\}$, множества $V_1 = \dots = V_{p-1} = \emptyset$.

Шаг 1. Если в графе нет вершин, помеченных номером волны i ($V_i = \emptyset$), идем на шаг 3. Иначе увеличиваем номер волны: $i = i + 1$.

Шаг 2. Все непомеченные вершины, смежные с вершинами из множества V_{i-1} , помечаем индексом i и добавляем в множество V_i . Идем на шаг 1.

Шаг 3. Восстанавливаем кратчайшие пути для всех помеченных вершин. Путь от вершины $v_0 = s$ до вершины $v_i \in V_i$ имеет вид

$$v_0 v_1 \dots v_{i-1} v_i.$$

Пути восстанавливаются от конца к началу. Вершина v_k , $k = i - 1, \dots, 0$ определяется из условий

$$v_k \in V_k, \quad (v_k, v_{k+1}) \in E.$$

Для непомеченных вершин путей из s не существует.

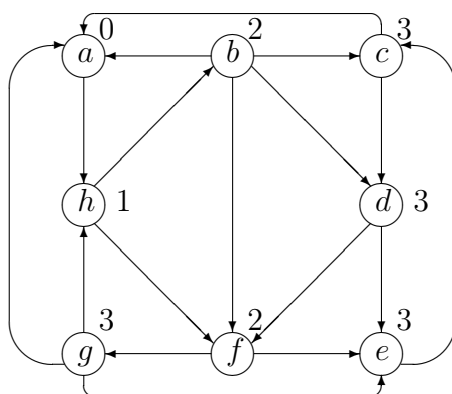
Конец.

Обоснование алгоритма. Обозначим множество вершин, для которых кратчайший путь от вершины s имеет длину k , через S_k . Для обоснования алгоритма достаточно показать, что $S_k = V_k$.

Используем метод математической индукции по номеру волны i . База индукции: $S_0 = \{s\} = V_0$. Пусть теперь $S_i = V_i$ для всех $i < k$. Рассмотрим вершину $v \in S_k$. Пусть $s \dots uv$ – кратчайший путь $\langle s, v \rangle$, тогда $u \in S_{k-1}$, и по индукционному предположению $u \in V_{k-1}$. К тому же $v \notin V_0 \cup V_1 \cup \dots \cup V_{k-1}$, иначе по индукционному предположению до нее существовал бы путь длины, меньшей k . Значит, по алгоритму вершина v будет помечена индексом k , т. е. $v \in V_k$ и $S_k \subseteq V_k$.

Рассмотрим вершину $v \in V_k$. Она не была помечена индексом, меньшим k , т. е. по индукционному предположению до нее существует пути длины, меньшей k . По построению множества V_k найдется вершина $u \in V_{k-1}$ такая, что $(u, v) \in E$. По индукционному предположению $u \in S_{k-1}$, объединяя кратчайший путь $\langle s, u \rangle$ длины $k - 1$ и ребро (u, v) , получаем кратчайший путь $\langle s, v \rangle$ длины k . Значит, $v \in S_k$ и $V_k \subseteq S_k$.

Пример. Рассмотрим ориентированный граф из предыдущего примера. В качестве стартового выберем узел a . Возле каждого узла запишем номер волны.



Начало. Полагаем номер волны $i = 0$ и помечаем узел a индексом 0. Все узлы, кроме a , считаем непомеченными. Полагаем $V_0 = \{a\} \neq \emptyset$, $V_1 = \dots = V_{p-1} = \emptyset$.

Шаг 1. Поскольку $V_0 \neq \emptyset$, полагаем номер волны $i = 1$.

Шаг 2. Множество смежности $\Gamma(V_0) = \{h\}$, помечаем h номером волны 1 и полагаем $V_1 = \{h\}$. Идем на шаг 1.

Шаг 1. Поскольку $V_1 \neq \emptyset$, полагаем номер волны $i = 2$.

Шаг 2. Множество смежности $\Gamma(V_1) = \Gamma(h) = \{b, f\}$, эти узлы еще не отмечены, помечаем их номером волны 2 и полагаем $V_2 = \{b, f\}$. Идем на шаг 1.

Шаг 1. Поскольку $V_2 \neq \emptyset$, полагаем номер волны $i = 3$.

Шаг 2. Множество смежности $\Gamma(V_2) = \Gamma(b) \cup \Gamma(f) = \{a, c, d, e, g\}$, узел a , отмечен, помечаем остальные узлы номером волны 3 и полагаем $V_3 = \{c, d, e, g\}$. Идем на шаг 1.

Шаг 1. Поскольку $V_3 \neq \emptyset$, полагаем номер волны $i = 4$.

Шаг 2. Множество смежности $\Gamma(V_3) = \Gamma(c) \cup \Gamma(d) \cup \Gamma(e) \cup \Gamma(g) = \{a, c, d, e, f, h\}$, все эти узлы отмечены, значит $V_4 = \emptyset$.

Шаг 1. Поскольку $V_4 = \emptyset$, идем на шаг 3.

Шаг 3. Восстанавливаем пути, начиная с узлов с наибольшим номером волны, в данном примере это номер 3. Рассмотрим множество $V_3 = \{c, d, e, g\}$. Выберем узел c . Перед ним может стоять один из узлов из множества $V_2 = \{b, f\}$. Поскольку $(b, c) \in E$, перед узлом c стоит узел b . Рассмотрим теперь узел b с номером волны 2. Перед ним может стоять один из узлов из множества $V_1 = \{h\}$, т.е. это может быть только узел h . Действительно, $(h, b) \in E$. Перед узлом h с номером волны 1 может стоять лишь узел с номером волны 0, а это стартовый узел a . Итак, получили кратчайший путь

$$\langle a, c \rangle_{\text{крат}} = ahbc.$$

Заметим, что в ходе поиска $\langle a, c \rangle_{\text{крат}}$ найдены также кратчайшие пути до узлов, вошедших в искомый путь, а именно

$$\langle a, b \rangle_{\text{крат}} = ahb;$$

$$\langle a, h \rangle_{\text{крат}} = ah.$$

Аналогично можно найти остальные пути (проделайте это самостоятельно).

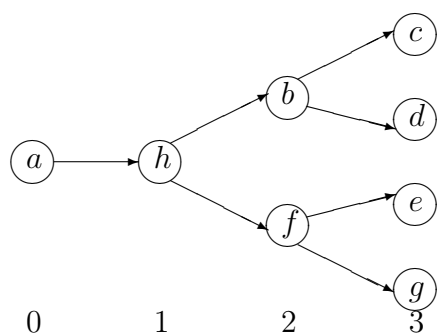
$$\langle a, d \rangle_{\text{крат}} = ahbd;$$

$$\langle a, e \rangle_{\text{крат}} = ahfe;$$

$$\langle a, g \rangle_{\text{крат}} = ahfg;$$

$$\langle a, f \rangle_{\text{крат}} = ahf.$$

Для наглядности ход работы алгоритма изобразим также на диаграмме, узлы с одним номером волны поместим одну под другой, а узлы с большим номером волны – правее, чем узлы с меньшим номером. По данной диаграмме легко также восстановить кратчайшие пути.



3.3 Поиск минимальных путей

Определение. *Нагруженным графом* $G(V, E)$ называется граф, каждому ребру которого сопоставлено некоторое число, называемое *весом*.

Нагруженный граф может быть задан матрицей смежности C , для которой элемент $c_{i,j}$ равен весу ребра (i, j) . Если ребра (i, j) не существует, то $c_{i,j} = \infty$.

Определение. *Минимальным путем* $\langle u, v \rangle_{\min}$ в нагруженном графе называется путь с наименьшим суммарным весом ребер.

Вес минимального пути $\langle u, v \rangle_{\min}$ будем обозначать через $w(u, v)$.

3.3.1 Поиск минимальных путей от заданной вершины

Рассмотрим алгоритм поиска минимального пути, который применим, когда веса ребер неотрицательны. Алгоритм может применяться как для ориентированных, так и для неориентированных графов.

Алгоритм Дейкстры поиска минимальных путей от вершины s до остальных вершин графа.

Начало. Присвоение вершинам начальных меток. Полагаем $\lambda(s) = \lambda_0(s) = 0$, $\lambda_0(v) = \infty \forall v \neq s$, номер итерации $i = 0$. Назначаем текущую вершину $v = s$. Все вершины, кроме s , считаем неокрашенными.

Шаг 1. Увеличиваем номер итерации: $i = i + 1$. Если все вершины окрашены, идем на шаг 3. Иначе для всех неокрашенных вершин u , смежных с вершиной v , пересчитываем метки по формуле

$$\lambda_i(u) = \min\{\lambda_{i-1}(u), \lambda(v) + c_{v,u}\},$$

(т. е. находим минимум из «старого» значения веса пути и веса «нового» пути, в котором текущая вершина лежит непосредственно перед u). Для остальных неокрашенных вершин полагаем $\lambda_i(u) = \lambda_{i-1}(u)$.

Шаг 2. Если для всех неокрашенных вершин $\lambda_i(u) = \infty$, идем на шаг 3. Иначе выбираем новую текущую вершину v из неокрашенных вершин из условия:

$$\lambda_i(v) = \min\{\lambda_i(u), u - \text{неокрашенные}\}.$$

Если $\lambda_i(v) < \infty$, окрашиваем вершину v , полагаем $\lambda(v) = \lambda_i(v)$ и идем на шаг 1.

Шаг 3. Восстанавливаем минимальные пути для всех окрашенных вершин. Пути восстанавливаются от конца к началу. Вершина x , предшествующая вершине v , определяется из соотношения

$$\lambda(x) + c_{x,v} = \lambda(v).$$

Процесс заканчивается, когда $w = s$. Для неокрашенных вершин путей из s не существует.

Конец.

Обоснование алгоритма. Для доказательства корректности алгоритма достаточно показать, что в качестве текущей всегда используется вершина, минимальный путь до которой известен. Другими словами, нужно показать, что, если вершина v окрашена, то $\lambda(v) = w(s, v)$.

Индукция по итерациям алгоритма. База индукции: на нулевой итерации $\lambda(s) = 0 = w(s, s)$ (т. к. длины дуг неотрицательны, никакой путь не может иметь отрицательную стоимость). Пусть $\lambda(u) = w(s, u)$ для всех ранее окрашенных вершин u . Заметим, что кратчайшие пути до этих вершин известны и проходят только через окрашенные вершины. Рассмотрим очередную вершину v , которая будет окрашена, выбранную из условия:

$$\lambda_i(v) = \min\{\lambda_i(u), u - \text{неокрашенные}\}.$$

Допустим, что $\lambda_i(v) > w(s, v)$, т. е. найденный путь не является кратчайшим. Тогда на этом пути должны быть неокрашенные вершины. Пусть первая неокрашенная вершина на этом пути x , до нее найден путь, проходящий через окрашенные вершины, т. е. кратчайший путь. Имеем

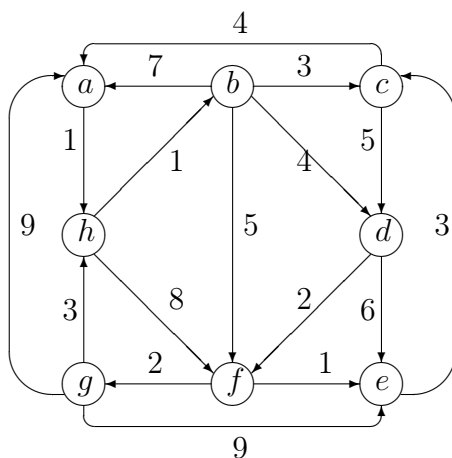
$$\lambda_i(x) = w(s, x) < w(s, v) < \lambda_i(v),$$

а это противоречит выбору вершины v .

Вычислительная сложность алгоритма. На i -й итерации i вершин уже окрашено, соответственно пересчитываются пути для оставшихся $p - i$ вершин. Всего выполняется p итераций алгоритма, значит, его вычислительная сложность равна

$$p + (p - 1) + \dots + 1 = \frac{p(p - 1)}{2} \approx p^2.$$

Пример. Рассмотрим ориентированный нагруженный граф. Около дуг представлены их веса.



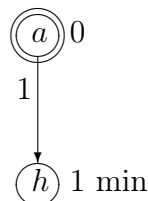
Начало. Будем искать пути от вершины a до остальных вершин. Полагаем $\lambda(a) = \lambda_0(a) = 0$, $\lambda_0(v) = \infty \forall v \neq a$, номер итерации $i = 0$. Назначаем текущую вершину $v = a$. Все вершины, кроме a , считаем неокрашенными.

Шаг 1. Не все вершины окрашены, увеличиваем номер итерации: $i = 1$. Поскольку $\Gamma(a) = \{h\}$, h неокрашена, пересчитываем метку по формуле

$$\lambda_1(h) = \min\{\lambda_0(h), \lambda(a) + c_{a,h}\} = \min\{\infty, 0 + 1\} = 1.$$

Для остальных неокрашенных вершин полагаем $\lambda_2(u) = \lambda_1(u)$. Текущее решение изображено на диаграмме, не показаны вершины, для которых $\lambda_1(u) = \infty$.

Окрашенные вершины обведены двойной линией, рядом с вершинами проставлены веса текущих минимальных путей $\lambda_1(u)$. Минимальный вес для неокрашенной вершины отмечен знаком \min .



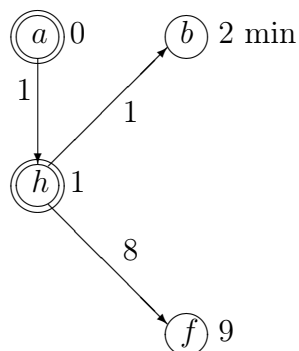
Шаг 2. Выбираем новую текущую вершину h , это единственная из неокрашенных вершин, для которой $\lambda_1(h) < \infty$. Окрашиваем вершину h , полагаем $\lambda(h) = \lambda_1(h)$ и идем на шаг 1.

Шаг 1. Не все вершины окрашены, увеличиваем номер итерации: $i = 2$. Поскольку $\Gamma(h) = \{b, f\}$, и эти вершины неокрашены, пересчитываем их метки

$$\lambda_2(b) = \min\{\lambda_1(b), \lambda(h) + c_{h,b}\} = \min\{\infty, 1 + 1\} = 2;$$

$$\lambda_2(f) = \min\{\lambda_1(f), \lambda(h) + c_{h,f}\} = \min\{\infty, 1 + 8\} = 9.$$

Для остальных неокрашенных вершин полагаем $\lambda_1(u) = \lambda_0(u)$.



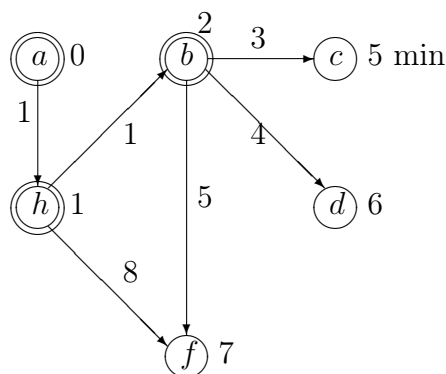
Шаг 2. Выбираем новую текущую вершину b , имеющую минимальный вес текущего минимального пути среди неокрашенных. Окрашиваем вершину b , полагаем $\lambda(b) = \lambda_2(b)$ и идем на шаг 1.

Шаг 1. Не все вершины окрашены, увеличиваем номер итерации: $i = 3$. Поскольку $\Gamma(b) = \{a, c, d, f\}$, и вершины c, d, f неокрашены, пересчитываем их метки

$$\lambda_3(c) = \min\{\lambda_2(c), \lambda(b) + c_{b,c}\} = \min\{\infty, 2 + 3\} = 5;$$

$$\lambda_3(d) = \min\{\lambda_2(d), \lambda(b) + c_{b,d}\} = \min\{\infty, 2 + 4\} = 6;$$

$$\lambda_3(f) = \min\{\lambda_2(f), \lambda(b) + c_{b,f}\} = \min\{9, 2 + 5\} = 7.$$



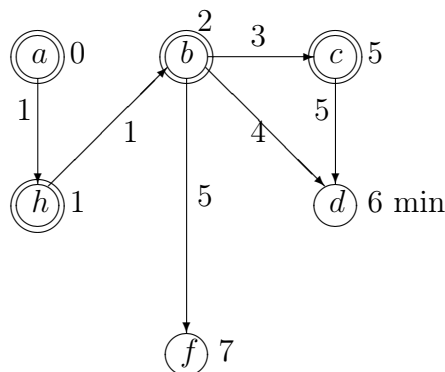
Для остальных неокрашенных вершин полагаем $\lambda_3(u) = \lambda_2(u)$.

Шаг 2. Выбираем новую текущую вершину c . Окрашиваем вершину c , полагаем $\lambda(c) = \lambda_3(c)$ и идем на шаг 1.

Шаг 1. Не все вершины окрашены, увеличиваем номер итерации: $i = 4$. Поскольку $\Gamma(c) = \{a, d\}$, и вершина d неокрашена, пересчитываем ее метку

$$\lambda_4(d) = \min\{\lambda_3(d), \lambda(c) + c_{c,d}\} = \min\{6, 5 + 5\} = 6.$$

Для остальных неокрашенных вершин полагаем $\lambda_4(u) = \lambda_3(u)$.

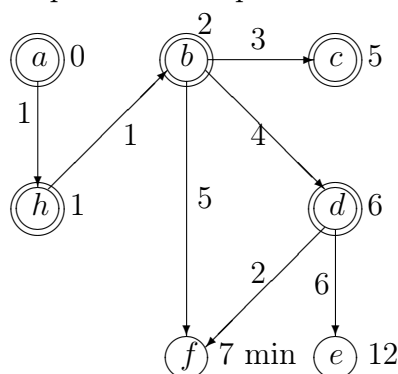


Шаг 2. Выбираем новую текущую вершину d . Окрашиваем вершину d , полагаем $\lambda(d) = \lambda_4(d)$ и идем на шаг 1.

Шаг 1. Не все вершины окрашены, увеличиваем номер итерации: $i = 5$. Поскольку $\Gamma(d) = \{e, f\}$, и эти вершины неокрашены, пересчитываем их метки

$$\begin{aligned} \lambda_5(e) &= \min\{\lambda_4(e), \lambda(d) + c_{d,e}\} = \min\{\infty, 6 + 6\} = 12; \\ \lambda_5(f) &= \min\{\lambda_4(f), \lambda(d) + c_{d,f}\} = \min\{7, 6 + 2\} = 7. \end{aligned}$$

Для остальных неокрашенных вершин полагаем $\lambda_5(u) = \lambda_4(u)$.

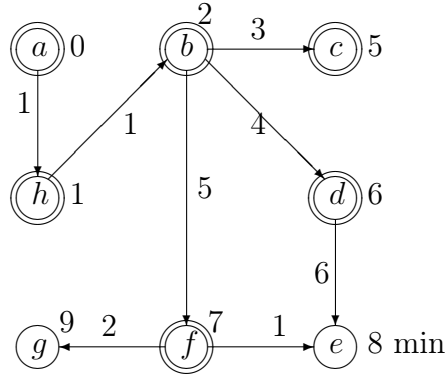


Шаг 2. Выбираем новую текущую вершину f . Окрашиваем вершину f , полагаем $\lambda(f) = \lambda_5(f)$ и идем на шаг 1.

Шаг 1. Не все вершины окрашены, увеличиваем номер итерации: $i = 6$. Поскольку $\Gamma(f) = \{e, g\}$, и эти вершины неокрашены, пересчитываем их метки

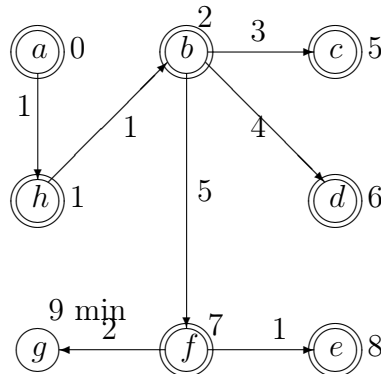
$$\begin{aligned} \lambda_6(e) &= \min\{\lambda_5(e), \lambda(f) + c_{f,e}\} = \min\{12, 7 + 1\} = 8; \\ \lambda_6(g) &= \min\{\lambda_5(g), \lambda(f) + c_{f,g}\} = \min\{\infty, 7 + 2\} = 9. \end{aligned}$$

Для остальных неокрашенных вершин полагаем $\lambda_6(u) = \lambda_5(u)$.



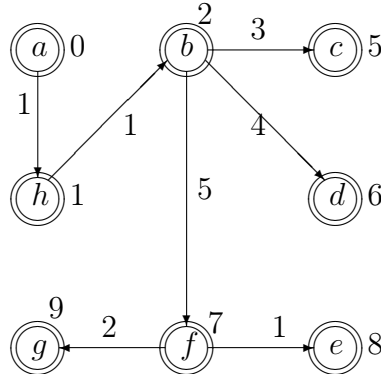
Шаг 2. Выбираем новую текущую вершину e . Окрашиваем вершину e , полагаем $\lambda(e) = \lambda_6(e)$ и идем на шаг 1.

Шаг 1. Не все вершины окрашены, увеличиваем номер итерации: $i = 7$. Поскольку $\Gamma(e) = \{c\}$, и вершина c окрашена, никакие метки не пересчитываются.



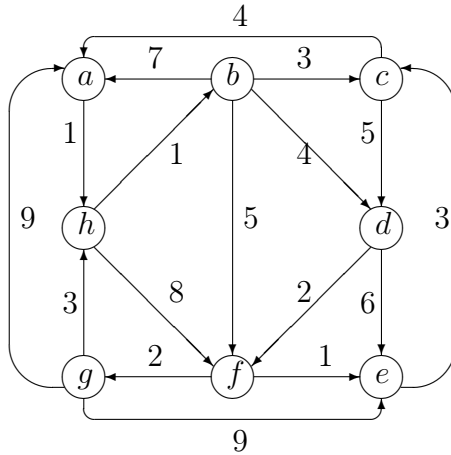
Шаг 2. Выбираем новую текущую вершину g . Окрашиваем вершину g , полагаем $\lambda(g) = \lambda_7(g)$ и идем на шаг 1.

Шаг 1. Все вершины окрашены, увеличиваем номер итерации: $i = 8$. Идем на шаг 3.



Шаг 3. Восстановим минимальные пути. В ходе работы алгоритма найдены веса минимальных путей:

$$\begin{aligned} \lambda(a) &= 0; & \lambda(d) &= 6; \\ \lambda(h) &= 1; & \lambda(f) &= 7; \\ \lambda(b) &= 2; & \lambda(e) &= 8; \\ \lambda(c) &= 5; & \lambda(g) &= 9. \end{aligned}$$



Здесь мы упорядочили вершины по возрастанию весов минимальных путей. Очевидно, путь

$$\langle a, h \rangle_{\text{мин}} = ah,$$

поскольку все веса ребер положительны, то перед вершиной h на минимальном пути может стоять только вершина a .

Найдем путь $\langle a, b \rangle_{\text{мин}} = a \dots xb$, где вершина x определяется из условия

$$\lambda(x) + c_{x,b} = \lambda(b).$$

В качестве x могут быть вершины a и h , проверяем для них условие

$$\begin{aligned} a : \quad & \lambda(a) + c_{a,b} = 0 + \infty \neq 2 = \lambda(b); \\ h : \quad & \lambda(h) + c_{h,b} = 1 + 1 = 2 = \lambda(b). \end{aligned}$$

Условие выполнено при $x = h$, минимальный путь до h был получен ранее, значит,

$$\langle a, b \rangle_{\text{мин}} = ahb.$$

Найдем путь $\langle a, c \rangle_{\text{мин}} = a \dots xc$, где вершина x определяется из условия

$$\lambda(x) + c_{x,c} = \lambda(c).$$

В качестве x могут быть вершины a , h и b , проверяем для них условие

$$\begin{aligned} a : \quad & \lambda(a) + c_{a,c} = 0 + \infty \neq 5 = \lambda(c); \\ h : \quad & \lambda(h) + c_{h,c} = 1 + \infty \neq 5 = \lambda(c); \\ b : \quad & \lambda(b) + c_{b,c} = 2 + 3 = 5 = \lambda(c); \end{aligned}$$

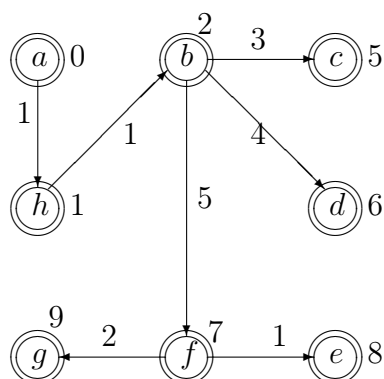
Условие выполнено при $x = b$, минимальный путь до b был получен ранее, значит,

$$\langle a, c \rangle_{\text{мин}} = ahbc.$$

Действуя аналогично, находим остальные минимальные пути (проделайте это самостоятельно)

$$\begin{aligned} \langle a, d \rangle_{\text{мин}} &= ahbd; \\ \langle a, f \rangle_{\text{мин}} &= ahbf; \\ \langle a, e \rangle_{\text{мин}} &= ahbfe; \\ \langle a, g \rangle_{\text{мин}} &= ahbfg. \end{aligned}$$

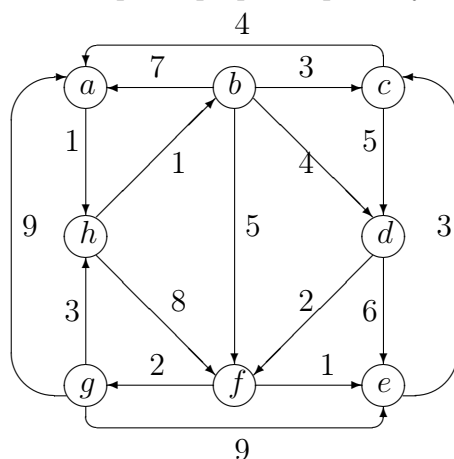
Полученные минимальные пути совпали с путями в графе, построенном в ходе решения задачи.



Замечание. При программировании алгоритма Дейкстры можно запоминать в отдельном массиве предпоследние вершины на минимальных путях. Тогда при восстановлении путей не требуется перебирать вершины, которые могут предшествовать данной, и проверять для них условие.

Ход работы алгоритма Дейкстры удобно оформлять в виде таблицы.

Пример. Рассмотрим граф из предыдущего примера.



Здесь столбец i содержит номер итерации, столбец v – текущий узел, столбец $\lambda(v)$ – вес минимального пути от стартового узла до текущего. В столбцах $a-h$ содержатся метки неокрашенных узлов – наименьшие веса путей, найденные на текущей итерации. Минимальная метка выделяется **жирным** шрифтом. Узел с минимальной меткой назначается текущим на следующей итерации, и далее окрашивается и не рассматривается.

i	v	$\lambda(v)$	a	b	c	d	e	f	g	h
0			0	∞	∞	∞	∞	∞	∞	∞
1	a	0		∞	∞	∞	∞	∞	∞	1
2	h	1		2	∞	∞	∞	9	∞	
3	b	2			5	6	∞	7	∞	
4	c	5				6	∞	7	∞	
5	d	6					12	7	∞	
6	f	7					8		9	
7	e	8							9	
8	g	9								

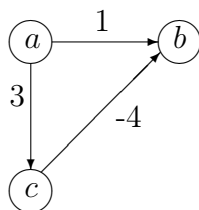
По таблице также удобно восстанавливать минимальные пути. Вершина x , которая стоит непосредственно перед u на пути $\langle s, u \rangle_{\min}$ – это вершина, которая была текущей на той итерации, где было впервые найден вес пути $\langle s, u \rangle_{\min}$. Например, рассмотрим вершину f , вес минимального пути $\lambda(f) = 7$, впервые он был получен, когда текущей вершиной являлась b , значит, перед f на искомом пути стоит b . Вес минимального пути $\lambda(b) = 2$, впервые он был получен, когда текущей вершиной являлась h , значит, перед b на искомом пути стоит h . Вес минимального пути $\lambda(h) = 1$, впервые он был получен, когда текущей вершиной являлась a , значит, перед b на искомом пути стоит a . Вершина a является стартовой, значит, минимальный путь найден и имеет вид

$$\langle a, f \rangle_{\min} = ahbf.$$

Попутно были найдены минимальные пути до всех вершин, входящих в $\langle a, f \rangle_{\min}$:

$$\begin{aligned} \langle a, b \rangle_{\min} &= ahb; \\ \langle a, h \rangle_{\min} &= ah. \end{aligned}$$

Если граф содержит дуги отрицательного веса, то для поиска минимального пути не может быть применен алгоритм Дейкстры. Покажем это на примере. Пусть граф имеет вид



Если применить к данному графу алгоритм Дейкстры для поиска минимальных путей от вершины a до остальных вершин, то вершина a будет являться текущей на первой итерации, а следующей текущей вершиной станет b , причем $\lambda(b) = 1$, и эта вершина окрасится. На второй итерации следующей текущей вершиной станет c с $\lambda(c) = 3$, и эта вершина окрасится. На третьей итерации c будет текущей вершиной, и, т.к. все вершины окрашены, никакие веса путей пересчитываться не будут, в том числе путь до вершины b . В итоге мы получим, что $\lambda(b) = 1$, тогда как $\langle a, b \rangle_{\min} = acb$ и имеет вес $3 - 4 = -1$.

Рассмотрим алгоритм, являющийся модификацией алгоритма Дейкстры. Изменения происходят на первом шаге алгоритма: пересчитываются пути как до окрашенных, так и до неокрашенных вершин, причем, если вес пути уменьшается, то с вершины снимается окраска. В результате она еще раз становится текущей вершиной по крайней мере на одной из следующих итераций.

Алгоритм Форда поиска минимальных путей от вершины s до остальных вершин графа.

Начало. Присвоение вершинам начальных меток. Полагаем $\lambda(s) = \lambda_0(s) = 0$, $\lambda_0(v) = \infty \forall v \neq s$, номер итерации $i = 0$. Назначаем текущую вершину $v = s$. Все вершины, кроме s , считаем неокрашенными.

Шаг 1. Увеличиваем номер итерации: $i = i + 1$. Для всех неокрашенных вершин u , смежных с вершиной v , пересчитываем метки по формуле

$$\lambda_i(u) = \min\{\lambda_{i-1}(u), \lambda_{i-1}(v) + c_{v,u}\},$$

(т. е. находим минимум из «старого» значения веса пути и веса «нового» пути, в котором текущая вершина лежит непосредственно перед u). Если для некоторой окрашенной вершины u происходит уменьшение метки: $\lambda_i(u) < \lambda_{i-1}(u)$, то с этой вершины окраска снимается. Для остальных неокрашенных вершин полагаем $\lambda_i(u) = \lambda_{i-1}(u)$.

Шаг 2. Если все вершины окрашены, или для всех неокрашенных вершин $\lambda_i(u) = \infty$, идем на шаг 3. Иначе выбираем новую текущую вершину v из неокрашенных вершин из условия:

$$\lambda_i(v) = \min\{\lambda_i(u), u - \text{неокрашенные}\}.$$

Если $\lambda_i(v) < \infty$, окрашиваем вершину v , полагаем $\lambda(v) = \lambda_i(v)$ и идем на шаг 1.

Шаг 3. Восстанавливаем минимальные пути для всех окрашенных вершин. Пути восстанавливаются от конца к началу. Вершина x , предшествующая вершине v , определяется из соотношения

$$\lambda(x) + c_{x,v} = \lambda(v).$$

Процесс заканчивается, когда $x = s$. Для неокрашенных вершин путей из s не существует.

Конец.

Обоснование алгоритма. Проведем обоснование методом от противного. При этом будем иметь в виду, что в результате алгоритма Форда для всех пар вершин выполняется соотношение

$$\lambda(v) + c_{v,u} \geq \lambda(u),$$

иначе с вершины u обязательно снялась бы окраска, когда v была текущей вершиной.

Итак, предположим, что для некоторой вершины v вес пути $\lambda(v)$ не является минимальным, т.е. $\lambda(v)$ – вес некоторого не минимального пути. Если таких вершин несколько, то выбирается та из них, найденный путь до которой содержит наименьшее число ребер. Пусть x – вершина, непосредственно предшествующая v на минимальном пути $\langle s, v \rangle_{\text{мин}}$, тогда $\lambda(x)$ – вес минимального пути $\langle s, x \rangle$. Тогда имеем

$$\lambda(x) + c_{x,v} < \lambda(v),$$

что противоречит правилу остановки алгоритма.

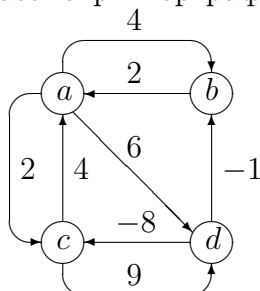
Вычислительная сложность алгоритма. Рассмотрим сначала случай, когда граф не содержит контуров отрицательного веса. Пусть найден минимальный путь до вершины v , т. е. $\lambda_i(v)$ – вес минимального пути, и с вершины v окраска больше сниматься не будет. Тогда если v лежит предпоследней на минимальном пути $\langle s, u \rangle$, то с вершины u снимется окраска, и минимальный путь до нее будет получен на данной итерации, т. е. затем она окрасится окончательно. В

худшем случае каждая из оставшихся вершин окрасится один раз, прежде чем одна из них окрасится окончательно. Значит, каждая вершина окрасится не более чем $p - 1$ раз, и каждый раз путь до нее будет пересчитываться до $p - 1$ вершины. Всего вершин p , следовательно, трудоемкость алгоритма

$$p(p - 1)p \approx p^3.$$

Если же граф содержит циклы отрицательного веса, то, включая этот цикл в путь, можно добиться сколь угодно низкой стоимости пути, т. е. вершина может окрашиваться бесконечное число раз. Поэтому, если заранее неизвестно, содержит ли граф циклы отрицательного веса, можно дополнительно считать, сколько раз окрашивается вершина. Если какая-либо вершина окрасилась p раз, задача не имеет решения.

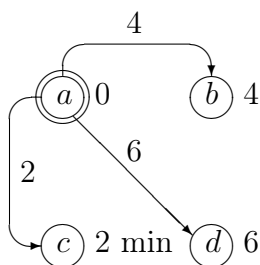
Пример. Рассмотрим орграф, содержащий дуги отрицательного веса.



Начало. Найдем минимальные пути от вершины a до остальных вершин. Полагаем $\lambda(a) = \lambda_0(a) = 0$, $\lambda_0(b) = \lambda_0(c) = \lambda_0(d) = \infty$, номер итерации $i = 0$. Назначаем текущую вершину $v = a$. Все вершины, кроме a , считаем неокрашенными.

Шаг 1. Увеличиваем номер итерации: $i = 1$. Пересчитываем метки для вершин из множества $\Gamma(a) = \{b, c, d\}$

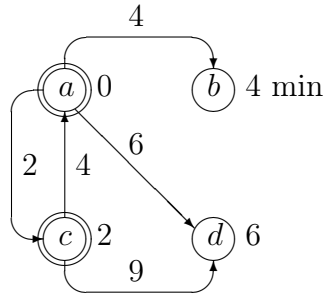
$$\begin{aligned} \lambda_1(b) &= \min\{\lambda_0(b), \lambda_0(a) + c_{a,b}\} = \min\{\infty, 0 + 4\} = 4; \\ \lambda_1(c) &= \min\{\lambda_0(c), \lambda_0(a) + c_{a,c}\} = \min\{\infty, 0 + 2\} = 2; \\ \lambda_1(d) &= \min\{\lambda_0(d), \lambda_0(a) + c_{a,d}\} = \min\{\infty, 0 + 6\} = 6. \end{aligned}$$



Шаг 2. Выбираем новую текущую вершину c из неокрашенных вершин, окрашиваем ее и идем на шаг 1.

Шаг 1. Увеличиваем номер итерации: $i = 2$. Пересчитываем метки для вершин из множества $\Gamma(c) = \{a, d\}$

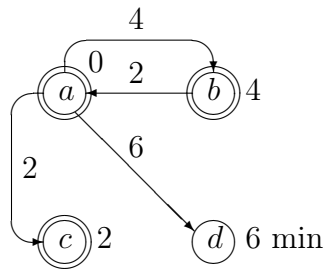
$$\begin{aligned} \lambda_2(a) &= \min\{\lambda_1(a), \lambda_1(c) + c_{c,a}\} = \min\{0, 2 + 4\} = 0; \\ \lambda_2(d) &= \min\{\lambda_1(d), \lambda_1(c) + c_{c,d}\} = \min\{6, 2 + 7\} = 6. \end{aligned}$$



Шаг 2. Выбираем новую текущую вершину b из неокрашенных вершин, окрашиваем ее и идем на шаг 1.

Шаг 1. Увеличиваем номер итерации: $i = 3$. Пересчитываем метки для вершин из множества $\Gamma(b) = \{a\}$

$$\lambda_3(a) = \min\{\lambda_2(a), \lambda_2(b) + c_{b,a}\} = \min\{0, 4 + 2\} = 0.$$



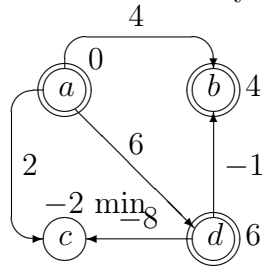
Шаг 2. Выбираем новую текущую вершину d из неокрашенных вершин, окрашиваем ее и идем на шаг 1.

Шаг 1. Увеличиваем номер итерации: $i = 4$. Пересчитываем метки для вершин из множества $\Gamma(d) = \{b, c\}$

$$\lambda_4(b) = \min\{\lambda_3(b), \lambda_3(d) + c_{d,b}\} = \min\{4, 6 - 1\} = 4;$$

$$\lambda_4(c) = \min\{\lambda_3(c), \lambda_3(d) + c_{d,c}\} = \min\{2, 6 - 8\} = -2.$$

Произошло уменьшение метки у вершины c , окраска с нее снимается.

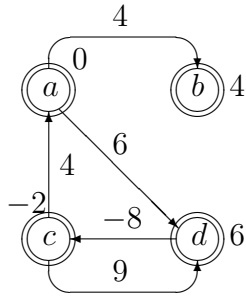


Шаг 2. Выбираем новую текущую вершину c из неокрашенных вершин, окрашиваем ее и идем на шаг 1.

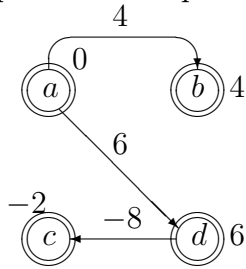
Шаг 1. Увеличиваем номер итерации: $i = 5$. Пересчитываем метки для вершин из множества $\Gamma(c) = \{a, d\}$

$$\lambda_5(a) = \min\{\lambda_4(a), \lambda_4(c) + c_{c,a}\} = \min\{0, -2 + 4\} = 0;$$

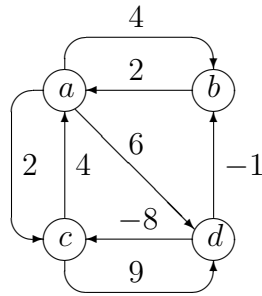
$$\lambda_5(d) = \min\{\lambda_4(d), \lambda_4(c) + c_{c,d}\} = \min\{6, -2 + 9\} = 9.$$



Шаг 2. Неокрашенных вершин нет, идем на шаг 3.



Шаг 3. Восстановим минимальные пути. В отличие от алгоритма Дейкстры, поскольку граф может содержать дуги отрицательного веса, вершина с большей меткой может стоять на минимальном пути перед вершиной с меньшей меткой.



$$\begin{aligned} \lambda(b) &= 4; \\ \lambda(c) &= -2; \\ \lambda(d) &= 6. \end{aligned}$$

Найдем путь $\langle a, b \rangle_{\min} = a \dots xb$, где вершина x определяется из условия

$$\lambda(x) + c_{x,b} = \lambda(b).$$

В качестве x могут быть вершины a , c и d , проверяем для них условие

$$\begin{aligned} a: \quad & \lambda(a) + c_{a,b} = 0 + 4 = 4 = \lambda(b); \\ c: \quad & \lambda(c) + c_{c,b} = -2 + \infty \neq 4 = \lambda(b); \\ d: \quad & \lambda(d) + c_{d,b} = 6 - 4 \neq 4 = \lambda(b). \end{aligned}$$

Условие выполнено при $x = a$, a – стартовая вершина, значит,

$$\langle a, b \rangle_{\min} = ab.$$

Найдем путь $\langle a, \rangle_{\min} = a \dots xc$, где вершина x определяется из условия

$$\lambda(x) + c_{x,c} = \lambda(c).$$

В качестве x могут быть вершины a , b и d , проверяем для них условие

$$\begin{aligned} a: \quad & \lambda(a) + c_{a,c} = 0 + 2 \neq -2 = \lambda(c); \\ b: \quad & \lambda(b) + c_{b,c} = 4 + \infty \neq -2 = \lambda(c); \\ d: \quad & \lambda(d) + c_{d,c} = 6 - 8 = -2 = \lambda(c). \end{aligned}$$

Условие выполнено при $x = d$. Найдем путь $\langle a, d \rangle_{\text{мин}} = a \dots xd$, где вершина x определяется из условия

$$\lambda(x) + c_{x,d} = \lambda(d).$$

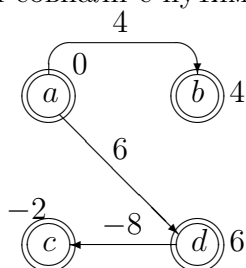
В качестве x могут быть вершины a и b (вершину c не рассматриваем, поскольку минимальные пути являются простыми), проверяем для них условие

$$\begin{aligned} a : \quad \lambda(a) + c_{a,d} &= 0 + 6 = 6 = \lambda(d); \\ b : \quad \lambda(b) + c_{b,d} &= 4 + \infty \neq 6 = \lambda(d). \end{aligned}$$

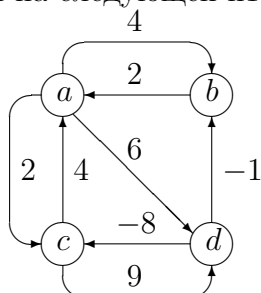
Условие выполнено при $x = a$, a – стартовая вершина, значит,

$$\begin{aligned} \langle a, d \rangle_{\text{мин}} &= ad; \\ \langle a, c \rangle_{\text{мин}} &= adc. \end{aligned}$$

Найденные пути совпали с путями, полученными в ходе работы алгоритма.



Решение также можно оформить в виде таблицы. Здесь столбец i содержит номер итерации, столбец v – текущий узел, столбец $\lambda(v)$ – вес минимального пути от стартового узла до текущего. В столбцах a – d содержатся метки неокрашенных узлов – наименьшие веса путей, найденные на текущей итерации. Минимальная метка для неокрашенных вершин выделяется **жирным** шрифтом, окрашенные на текущий момент вершины – *курсивом*. Узел с минимальной меткой назначается текущим на следующей итерации, и далее окрашивается.



i	v	$\lambda(v)$	a	b	c	d
0			0	∞	∞	∞
1	<i>a</i>	0	0	4	2	6
2	<i>c</i>	2	0	4	2	6
3	<i>b</i>	4	0	4	2	6
4	<i>d</i>	6	0	4	-2	6
5	<i>c</i>	-2	0	4	-2	6

Восстановим теперь минимальные пути. Рассмотрим вершину b , вес минимального пути $\lambda(b) = 4$, впервые он был получен, когда текущей вершиной являлась стартовая вершина a , значит,

$$\langle a, b \rangle_{\text{мин}} = ab.$$

Для вершины c вес минимального пути $\lambda(c) = -2$, впервые он был получен, когда текущей вершиной являлась d , значит, перед c на искомом пути стоит d . Вес

минимального пути $\lambda(d) = 6$, впервые он был получен, когда текущей вершиной являлась a , значит, перед d на искомом пути стоит a . Вершина a является стартовой, значит, минимальные пути до вершин c и d найдены:

$$\begin{aligned} \langle a, c \rangle_{\text{мин}} &= adc; \\ \langle a, d \rangle_{\text{мин}} &= ad. \end{aligned}$$

Рассмотрим еще один алгоритм поиска минимальных путей. Он основан на принципе динамического программирования Беллмана: управление на данном шаге требуется выбирать так, чтобы выигрыш на данном шаге в сумме с оптимальным выигрышем на всех последующих шагах был максимальным. К задаче поиска пути этот принцип применяется следующим образом: на каждой итерации алгоритма выбирается оптимальная вершина, непосредственно предшествующая вершине v на пути из стартовой вершины.

Алгоритм Беллмана–Мура поиска минимальных путей от вершины s до остальных вершин графа.

Начало. Присвоение вершинам начальных меток. Полагаем $\lambda_0(s) = 0$, $\lambda_0(v) = \infty \forall v \neq s$, номер итерации $i = 0$.

Шаг 1. Если $i = p - 1$, идем на шаг 3. Иначе увеличиваем номер итерации: $i = i + 1$. Для всех вершин v пересчитываем метки по формуле

$$\lambda_i(v) = \min_{x \in V} \{ \lambda_{i-1}(x) + c_{x,v} \}.$$

Шаг 2. Если $\exists v : \lambda_i(v) < \lambda_{i-1}(v)$, идем на шаг 1.

Шаг 3. Восстанавливаем минимальные пути. Полагаем для всех вершин $\lambda(v) = \lambda_i(v)$. Рассматриваем вершины v , для которых $\lambda(v) < \infty$. Вершина x , предшествующая вершине v , определяется из соотношения

$$\lambda(x) + c_{x,v} = \lambda(v).$$

Процесс заканчивается, когда $x = s$.

Конец.

Обоснование алгоритма. Достаточно показать, что при отсутствии циклов отрицательного веса $\lambda_i(v)$ – минимальный из весов путей $\langle s, v \rangle$, содержащих не более i дуг. Для $i = 0$ это верно. Пусть это верно для итераций $0, 1, \dots, i - 1$. Согласно алгоритму, $\lambda_i(v)$ вычисляется как минимум весов путей, для которых в качестве вершины, лежащей непосредственно перед v , перебираются все вершины. Заметим, что при $x = v$ имеем, что $c_{x,v} = 0$, и $\lambda_{i-1}(x) + c_{x,v} = \lambda_{i-1}(v)$, т.е. при выборе минимума рассматривается и вес «старого» пути $\lambda_{i-1}(v)$. По индукционному предположению мы берем минимальные пути, состоящие максимум из $i - 1$ дуги, т. е. в итоге мы перебираем пути, состоящие не более чем из i дуг, и выбираем из них минимальный.

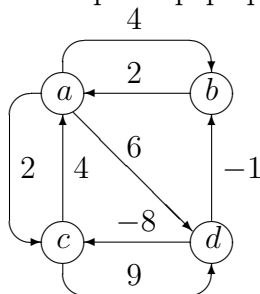
Вычислительная сложность алгоритма. Алгоритм требует выполнения p итераций, в ходе которых для всех p вершин выбирается минимум из p путей, итого требуется p^3 операций. Поэтому в случае, когда веса дуг неотрицательны, следует использовать алгоритм Дейкстры, который имеет сложность p^2 .

Замечание. Если заранее неизвестно, присутствуют ли в графе циклы отрицательного веса, алгоритм можно модифицировать так, чтобы выявить эти циклы. Для этого достаточно проверить соотношение

$$\lambda(x) + c_{x,v} \geq \lambda(v)$$

для всех пар вершин.

Пример. Рассмотрим орграф из предыдущего примера.



Начало. Найдем минимальные пути от вершины a до остальных вершин. Полагаем $\lambda_0(a) = 0$, $\lambda_0(b) = \lambda_0(c) = \lambda_0(d) = \infty$, номер итерации $i = 0$.

Шаг 1. Так как $i < p - 1$, увеличиваем номер итерации: $i = 1$. Для всех вершин v пересчитываем метки по формуле

$$\lambda_1(v) = \min_{x \in V} \{\lambda_0(x) + c_{x,v}\}.$$

Рассмотрим вершину a :

$$\begin{aligned} x = a : \quad & \lambda_0(a) + c_{a,a} = 0 + 0 = 0; \\ x = b : \quad & \lambda_0(b) + c_{b,a} = \infty + 2 = \infty; \\ x = c : \quad & \lambda_0(c) + c_{c,a} = \infty + 4 = \infty; \\ x = d : \quad & \lambda_0(d) + c_{d,a} = \infty + \infty = \infty. \end{aligned}$$

Итак,

$$\lambda_1(a) = \min\{0, \infty, \infty, \infty\} = 0.$$

Рассмотрим вершину b :

$$\begin{aligned} x = a : \quad & \lambda_0(a) + c_{a,b} = 0 + 4 = 4; \\ x = b : \quad & \lambda_0(b) + c_{b,b} = \infty + 0 = \infty; \\ x = c : \quad & \lambda_0(c) + c_{c,b} = \infty + \infty = \infty; \\ x = d : \quad & \lambda_0(d) + c_{d,b} = \infty - 1 = \infty. \end{aligned}$$

Итак,

$$\lambda_1(b) = \min\{4, \infty, \infty, \infty\} = 4.$$

Рассмотрим вершину c :

$$\begin{aligned} x = a : \quad & \lambda_0(a) + c_{a,c} = 0 + 2 = 2; \\ x = b : \quad & \lambda_0(b) + c_{b,c} = \infty + \infty = \infty; \\ x = c : \quad & \lambda_0(c) + c_{c,c} = \infty + 0 = \infty; \\ x = d : \quad & \lambda_0(d) + c_{d,c} = \infty - 8 = \infty. \end{aligned}$$

Итак,

$$\lambda_1(c) = \min\{2, \infty, \infty, \infty\} = 2.$$

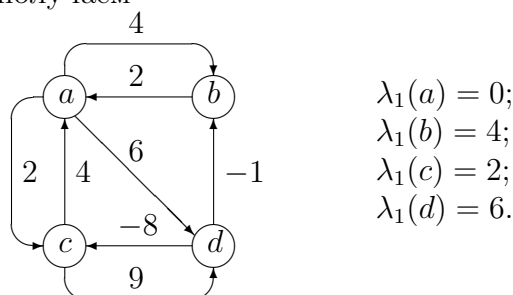
Рассмотрим вершину d :

$$\begin{aligned} x = a : \quad & \lambda_0(a) + c_{a,d} = 0 + 6 = 6; \\ x = b : \quad & \lambda_0(b) + c_{b,d} = \infty + \infty = \infty; \\ x = c : \quad & \lambda_0(c) + c_{c,d} = \infty + 9 = \infty; \\ x = d : \quad & \lambda_0(d) + c_{d,d} = \infty + 0 = \infty. \end{aligned}$$

Итак,

$$\lambda_1(d) = \min\{6, \infty, \infty, \infty\} = 4.$$

Окончательно получаем



Шаг 2. Поскольку веса путей изменились, идем на шаг 1.

Шаг 1. Так как $i < p - 1$, увеличиваем номер итерации: $i = 2$. Для всех вершин v пересчитываем метки по формуле

$$\lambda_2(v) = \min_{x \in V} \{\lambda_1(x) + c_{x,v}\}.$$

Здесь и далее будем расписывать вычисления менее подробно.

$$\begin{aligned} \lambda_2(a) &= \min\{0 + 0, 4 + 2, 2 + 4, 6 + \infty\} = 0; \\ \lambda_2(b) &= \min\{0 + 4, 4 + 0, 2 + \infty, 6 - 1\} = 4; \\ \lambda_2(c) &= \min\{0 + 2, 4 + \infty, 2 + 0, 6 - 8\} = -2; \\ \lambda_2(d) &= \min\{0 + 6, 4 + \infty, 2 + 9, 6 + 0\} = 6. \end{aligned}$$

Шаг 2. Поскольку $\lambda_2(c) < \lambda_1(c)$, идем на шаг 1.

Шаг 1. Так как $i < p - 1$, увеличиваем номер итерации: $i = 3$. Для всех вершин v пересчитываем метки

$$\begin{aligned} \lambda_3(a) &= \min\{0 + 0, 4 + 2, -2 + 4, 6 + \infty\} = 0; \\ \lambda_3(b) &= \min\{0 + 4, 4 + 0, -2 + \infty, 6 - 1\} = 4; \\ \lambda_3(c) &= \min\{0 + 2, 4 + \infty, -2 + 0, 6 - 8\} = -2; \\ \lambda_3(d) &= \min\{0 + 6, 4 + \infty, -2 + 9, 6 + 0\} = 6. \end{aligned}$$

Шаг 2. Веса путей не изменились.

Шаг 3. Восстановление путей производится так же, как и в алгоритме Форда.

Итак, нами рассмотрены три алгоритма поиска минимальных путей. Вычислительная сложность алгоритма Дейкстры равна p^2 , алгоритм работает с графами, веса ребер которых неотрицательны. Вычислительная сложность алгоритмов Форда и Беллмана–Мура равна p^3 , но при этом веса ребер могут быть отрицательными.

3.3.2 Поиск минимальных путей между всеми парами вершин

В предыдущем подразделе рассмотрена задача поиска путей от заданной вершины до остальных вершин. Разумеется, для решения задачи поиска путей между всеми парами вершин можно применить эти алгоритмы, используя все вершины поочередно в качестве стартовой, но более эффективным является использование специальных алгоритмов.

Алгоритм Флойда находит кратчайшие пути между всеми парами вершин в графе (орграфе). На i -й итерации алгоритма находятся минимальные пути, проходящие через вершины $1, \dots, i$. Для хранения информации о путях используется матрица $H : p \times p$, где элемент $h_{j,k}$ содержит первую вершину на пути $\langle j, k \rangle$ или равен нулю, если пути не существует. Для хранения информации о длинах путей используется матрица Λ , где на i -й итерации $\lambda_{j,k}$ – вес пути.

Алгоритм Флойда поиска минимальных путей между всеми парами вершин графа.

Начало. Инициализация матриц. Полагаем для всех $j, k : 1 \leq j, 1 \leq k$

$$\lambda_{j,k} = \begin{cases} c_{j,k}, & \text{если } j \neq k; \\ 0, & \text{если } j = k, \end{cases} \quad h_{j,k} = \begin{cases} j, & \text{если } c_{j,k} < \infty \text{ или } j = k; \\ 0, & \text{иначе.} \end{cases}$$

Номер итерации $i = 0$.

Шаг 1. Если $i = p$, идем на шаг 4, иначе полагаем $i = i + 1$.

Шаг 2. Для всех пар вершин $(j, k) : j \neq i, k \neq i$, для которых $\lambda_{j,i} + \lambda_{i,k} < \lambda_{j,k}$ («новый» путь лучше «старого»), полагаем $h_{j,k} = h_{j,i}$ и $\lambda_{j,k} = \lambda_{j,i} + \lambda_{i,k}$. Если для какого-либо j верно $\lambda_{j,j} < 0$, задача не имеет решения, т. к. вершина j входит в цикл отрицательного веса, идем на конец. Иначе идем на шаг 1.

Шаг 3. Находим минимальные пути между всеми парами вершин. Пусть путь $\langle j, k \rangle_{\min}$ имеет вид

$$\begin{aligned} \langle j, k \rangle_{\min} &= x_0 x_2 \dots x_l; \\ x_0 &= j, \quad x_l = k. \end{aligned}$$

Путь восстанавливаем от начала к концу по матрице H . Сначала полагаем $i = 1$. Очередную вершину x_i находим следующим образом:

$$x_i = h_{x_{i-1}, k}, \quad i > 0.$$

Если $x_i \neq k$, полагаем $i = i + 1$ и опять вычисляем очередную вершину.

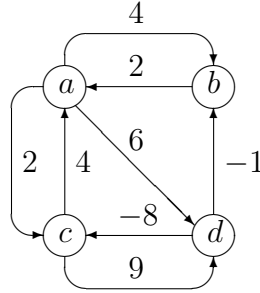
Конец.

Обоснование алгоритма. Алгоритм имеет много общего с рассмотренным ранее алгоритмом Уоршалла. Чтобы обосновать его, достаточно показать, что после выполнения i -й итерации матрицы Λ и H содержат веса минимальных путей, проходящих через вершины $1, \dots, i$, и информацию об этих путях. На нулевой итерации это верно, пусть это верно и для итераций $1, \dots, i - 1$. На i -й итерации вес пути $\langle j, k \rangle$ уменьшается, если существуют пути $\langle j, i \rangle$ и $\langle i, k \rangle$, проходящие через вершины $1, \dots, i - 1$, и суммарный вес этих путей не превосходит веса пути $\langle j, k \rangle$. По индукционному предположению эти пути являются минимальными, проходящими через вершины $1, \dots, i - 1$, а значит, в итоге мы получим минимальный путь $\langle j, k \rangle$, проходящий через вершины $1, \dots, i$.

Вычислительная сложность алгоритма. Шаги 1–3 представляют собой тройной цикл по i, j, k , которые меняются от 1 до p , значит, вычислительная сложность алгоритма p^3 .

Использование алгоритма Флойда для нахождения минимальных путей между всеми парами вершин предпочтительнее, чем использование алгоритмов Форда или Беллмана–Мура для всех вершин в качестве стартовых (в этом случае вычислительная сложность p^4). Однако, если веса ребер положительны, оправданным является применение алгоритма Дейкстры, вычислительная сложность также будет p^3 .

Пример. Рассмотрим орграф из предыдущего примера.



Начало. Полагаем номер итерации $i = 0$. Матрицы Λ и H имеют вид

$$\Lambda = \begin{array}{c|cccc} & a & b & c & d \\ \hline a & 0 & 4 & 2 & 6 \\ b & 2 & 0 & \infty & \infty \\ c & 4 & \infty & 0 & 9 \\ d & \infty & -1 & -8 & 0 \end{array} \quad H = \begin{array}{c|cccc} & a & b & c & d \\ \hline a & a & b & c & d \\ b & a & b & 0 & 0 \\ c & a & 0 & c & d \\ d & 0 & b & c & d \end{array}$$

Шаг 1. Поскольку $i < p$, полагаем $i = 1$.

Шаг 2. Найдем минимальные пути, которые могут проходить только через вершину a . Заметим, что вес пути $\langle j, k \rangle$ может измениться только в случае, если $\lambda_{j,a} + \lambda_{a,k} < \infty$, и при этом $j \neq a$ и $k \neq a$. Поэтому найдем сначала строки, отличные от a , содержащие элементы, отличные от ∞ , в столбце a : это строки b и c . Затем найдем столбцы, отличные от a , содержащие элементы, отличные от ∞ , в строке a : это столбцы b, c и d . Пересчитаем элементы матриц на пересечении этих строк и столбцов. Если $\lambda_{j,a} + \lambda_{a,k} < \lambda_{j,k}$, то положим $\lambda_{j,k} = \lambda_{j,a} + \lambda_{a,k}$, $h_{j,k} = h_{j,a}$, и запишем их новые значения:

$$\begin{aligned} \lambda_{b,a} + \lambda_{a,b} &= 2 + 4 = 6 > 0 = \lambda_{b,b}; \\ \lambda_{b,a} + \lambda_{a,c} &= 2 + 2 = 4 < \infty = \lambda_{b,c}, & \lambda_{b,c} &= 4, & h_{b,c} &= h_{b,a} = a; \\ \lambda_{b,a} + \lambda_{a,d} &= 2 + 6 = 8 < \infty = \lambda_{b,d}, & \lambda_{b,d} &= 8, & h_{b,d} &= h_{b,a} = a; \\ \lambda_{c,a} + \lambda_{a,b} &= 4 + 4 = 8 < \infty = \lambda_{c,b}, & \lambda_{c,b} &= 8, & h_{c,b} &= h_{c,a} = a; \\ \lambda_{c,a} + \lambda_{a,c} &= 4 + 2 = 6 > 0 = \lambda_{c,c}; \\ \lambda_{c,a} + \lambda_{a,d} &= 4 + 6 = 10 > 9 = \lambda_{c,d}; \end{aligned}$$

Матрицы Λ и H имеют вид

$$\Lambda = \begin{array}{c|cccc} & a & b & c & d \\ \hline a & 0 & 4 & 2 & 6 \\ b & 2 & 0 & 4 & 8 \\ c & 4 & 8 & 0 & 9 \\ d & \infty & -1 & -8 & 0 \end{array} \quad H = \begin{array}{c|cccc} & a & b & c & d \\ \hline a & a & b & c & d \\ b & a & b & a & a \\ c & a & a & c & d \\ d & 0 & b & c & d \end{array}$$

Поскольку $\forall j : \lambda_{j,j} = 0$, идем на шаг 1.

Шаг 1. Поскольку $i < p$, полагаем $i = 2$.

Шаг 2. Найдем минимальные пути, которые могут проходить только через вершины $\{a, b\}$. Найдем строки, отличные от b , содержащие элементы, отличные от ∞ , в столбце b : это строки a, c и d . Затем найдем столбцы, отличные от b , содержащие элементы, отличные от ∞ , в строке b : это столбцы a, c и d . Пересчитаем элементы матриц на пересечении этих строк и столбцов. Если $\lambda_{j,b} + \lambda_{b,k} < \lambda_{j,k}$, то положим $\lambda_{j,k} = \lambda_{j,b} + \lambda_{b,k}$, $h_{j,k} = h_{j,b}$, и запишем их новые значения:

$$\begin{aligned}
 \lambda_{a,b} + \lambda_{b,a} &= 4 + 2 = 6 > 0 = \lambda_{a,a}; \\
 \lambda_{a,b} + \lambda_{b,c} &= 4 + 4 = 8 > 2 = \lambda_{a,c}; \\
 \lambda_{a,b} + \lambda_{b,d} &= 4 + 8 = 12 > 6 = \lambda_{a,d}; \\
 \lambda_{c,b} + \lambda_{b,a} &= 8 + 2 = 10 > 4 = \lambda_{c,a}; \\
 \lambda_{c,b} + \lambda_{b,c} &= 8 + 4 = 12 > 0 = \lambda_{c,c}; \\
 \lambda_{c,b} + \lambda_{b,d} &= 8 + 8 = 16 > 9 = \lambda_{c,d}; \\
 \lambda_{d,b} + \lambda_{b,a} &= -1 + 2 = 1 < \infty = \lambda_{d,a}, \quad \lambda_{d,a} = 1, \quad h_{d,a} = h_{d,b} = b; \\
 \lambda_{d,b} + \lambda_{b,c} &= -1 + 4 = 3 > -8 = \lambda_{d,c}; \\
 \lambda_{d,b} + \lambda_{b,d} &= -1 + 8 = 7 > 0 = \lambda_{d,d}.
 \end{aligned}$$

Матрицы Λ и H имеют вид

$$\Lambda = \begin{array}{c|cccc} & a & b & c & d \\ \hline a & 0 & 4 & 2 & 6 \\ b & 2 & 0 & 4 & 8 \\ c & 4 & 8 & 0 & 9 \\ d & 1 & -1 & -8 & 0 \end{array} \quad H = \begin{array}{c|cccc} & a & b & c & d \\ \hline a & a & b & c & d \\ b & a & b & a & a \\ c & a & a & c & d \\ d & b & b & c & d \end{array}$$

Поскольку $\forall j : \lambda_{j,j} = 0$, идем на шаг 1.

Шаг 1. Поскольку $i < p$, полагаем $i = 3$.

Шаг 2. Найдем минимальные пути, которые могут проходить только через вершины $\{a, b, c\}$:

$$\begin{aligned}
 \lambda_{a,c} + \lambda_{c,a} &= 2 + 4 = 6 > 0 = \lambda_{a,a}; \\
 \lambda_{a,c} + \lambda_{c,b} &= 2 + 8 = 10 > 4 = \lambda_{a,b}; \\
 \lambda_{a,c} + \lambda_{c,d} &= 2 + 9 = 11 > 6 = \lambda_{a,d}; \\
 \lambda_{b,c} + \lambda_{c,a} &= 4 + 4 = 8 > 2 = \lambda_{b,a}; \\
 \lambda_{b,c} + \lambda_{c,b} &= 4 + 8 = 12 > 0 = \lambda_{b,b}; \\
 \lambda_{b,c} + \lambda_{c,d} &= 4 + 9 = 13 > 8 = \lambda_{b,d}; \\
 \lambda_{d,c} + \lambda_{c,a} &= -8 + 4 = -4 < 1 = \lambda_{d,a}, \quad \lambda_{d,a} = -4, \quad h_{d,a} = h_{d,c} = c; \\
 \lambda_{d,c} + \lambda_{c,b} &= -8 + 8 = 0 > -1 = \lambda_{d,b}; \\
 \lambda_{d,c} + \lambda_{c,d} &= -8 + 9 = 1 > 0 = \lambda_{d,d}.
 \end{aligned}$$

Матрицы Λ и H имеют вид

$$\Lambda = \begin{array}{c|cccc} & a & b & c & d \\ \hline a & 0 & 4 & 2 & 6 \\ b & 2 & 0 & 4 & 8 \\ c & 4 & 8 & 0 & 9 \\ d & -4 & -1 & -8 & 0 \end{array} \quad H = \begin{array}{c|cccc} & a & b & c & d \\ \hline a & a & b & c & d \\ b & a & b & a & a \\ c & a & a & c & d \\ d & c & b & c & d \end{array}$$

Поскольку $\forall j : \lambda_{j,j} = 0$, идем на шаг 1.

Шаг 1. Поскольку $i < p$, полагаем $i = 4$.

Шаг 2. Найдем минимальные пути, которые могут проходить через вершины $\{a, b, c, d\}$:

$$\begin{aligned}
 \lambda_{a,d} + \lambda_{d,a} &= 6 - 4 = 2 > 0 = \lambda_{a,a}; \\
 \lambda_{a,d} + \lambda_{d,b} &= 6 - 1 = 5 > 4 = \lambda_{a,b}; \\
 \lambda_{a,d} + \lambda_{d,c} &= 6 - 8 = -2 < 2 = \lambda_{a,c}, & \lambda_{a,c} = -2, & h_{a,c} = h_{a,d} = d; \\
 \lambda_{b,d} + \lambda_{d,a} &= 8 - 4 = 4 > 2 = \lambda_{b,a}; \\
 \lambda_{b,d} + \lambda_{d,b} &= 8 - 1 = 7 > 0 = \lambda_{b,b}; \\
 \lambda_{b,d} + \lambda_{d,c} &= 8 - 8 = 0 < 4 = \lambda_{b,c}, & \lambda_{b,c} = 0, & h_{b,c} = h_{b,d} = a; \\
 \lambda_{c,d} + \lambda_{d,a} &= 9 - 4 = 5 > 4 = \lambda_{c,a}; \\
 \lambda_{c,d} + \lambda_{d,b} &= 9 - 1 = 8 = 8 = \lambda_{c,b}; \\
 \lambda_{c,d} + \lambda_{d,c} &= 9 - 8 = 1 > 0 = \lambda_{c,c}.
 \end{aligned}$$

Матрицы Λ и H имеют вид

$$\Lambda = \begin{array}{c|cccc} & a & b & c & d \\ \hline a & 0 & 4 & -2 & 6 \\ b & 2 & 0 & 0 & 8 \\ c & 4 & 8 & 0 & 9 \\ d & -4 & -1 & -8 & 0 \end{array} \quad H = \begin{array}{c|cccc} & a & b & c & d \\ \hline a & a & b & d & d \\ b & a & b & a & a \\ c & a & a & c & d \\ d & c & b & c & d \end{array}$$

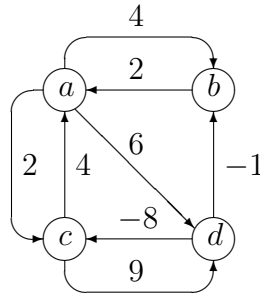
Поскольку $\forall j : \lambda_{j,j} = 0$, идем на шаг 1.

Шаг 1. Поскольку $i = p$, идем на шаг 3.

Шаг 3. Восстановим минимальные пути $\langle j, k \rangle : j \neq k$, используя матрицу H . Рассмотрим подробно путь $\langle a, c \rangle_{\min} = a \dots c$. Найдем вершину, следующую за a , это вершина $h_{a,c} = d$. Поскольку $d \neq c$, то $\langle a, c \rangle_{\min} = ad \dots c$. Найдем вершину, следующую за d , это вершина $h_{d,c} = c$, т.е. мы пришли в конечную вершину пути, который имеет вид $\langle a, c \rangle_{\min} = adc$. Аналогично найдем остальные пути:

$$\begin{aligned}
 \langle a, b \rangle : h_{a,b} &= b, & \langle a, b \rangle_{\min} &= ab; \\
 \langle a, c \rangle : h_{a,c} &= d, h_{d,c} = c, & \langle a, c \rangle_{\min} &= adc; \\
 \langle a, d \rangle : h_{a,d} &= d, & \langle a, d \rangle_{\min} &= ad; \\
 \langle b, a \rangle : h_{b,a} &= a, & \langle b, a \rangle_{\min} &= ba; \\
 \langle b, c \rangle : h_{b,c} &= a, h_{a,c} = d, h_{d,c} = c, & \langle b, c \rangle_{\min} &= badc; \\
 \langle b, d \rangle : h_{b,d} &= a, h_{a,d} = d, & \langle b, d \rangle_{\min} &= bad; \\
 \langle c, a \rangle : h_{c,a} &= a, & \langle c, a \rangle_{\min} &= ca; \\
 \langle c, b \rangle : h_{c,b} &= a, h_{a,b} = b, & \langle c, b \rangle_{\min} &= cab; \\
 \langle c, d \rangle : h_{c,d} &= d, & \langle c, d \rangle_{\min} &= cd; \\
 \langle d, a \rangle : h_{d,a} &= c, h_{c,a} = a, & \langle d, a \rangle_{\min} &= dca; \\
 \langle d, b \rangle : h_{d,b} &= b, & \langle d, b \rangle_{\min} &= db; \\
 \langle d, c \rangle : h_{d,c} &= c, & \langle d, c \rangle_{\min} &= dc.
 \end{aligned}$$

Убедиться, что найдены действительно минимальные пути, можно визуальным образом по диаграмме графа



Рассмотрим еще один алгоритм поиска минимальных путей между всеми парами вершин – *алгоритм Данцига*. Он близок алгоритму Флойда и отличается от него лишь другим порядком выполнения операций. Алгоритм основан на следующих соображениях.

Пусть ищется минимальный путь $\langle i, j \rangle$ или $\langle j, i \rangle$, где $j < i$, проходящий только через i первых вершин. Если в графе нет циклов отрицательного веса, то в качестве промежуточных вершин этих путей могут использоваться только вершины $\{1, \dots, i-1\}$, а сам путь $\langle j, i \rangle$, например, представляет собой объединение минимального пути $\langle j, k \rangle$ из путей, проходящих только через вершины $\{1, \dots, i-1\}$, и ребра (k, i) . Поэтому пути $\langle i, j \rangle$ и $\langle j, i \rangle$ имеет смысл находить только на i -й итерации алгоритма, и матрица минимальных путей на i -й итерации будет иметь размерность $i \times i$, т. е. мы будем строить последовательность матриц $\Lambda^{(1)}, \Lambda^{(2)}, \dots, \Lambda^{(p)}$, размерности которых соответственно равны $1 \times 1, 2 \times 2, \dots, p \times p$. Если граф не содержит циклов отрицательного веса, то вес пути $\langle i, i \rangle$ всегда равен нулю.

Для хранения самих путей, как и в алгоритме Флойда, используется матрица $H : p \times p$.

Алгоритм Данцига поиска минимальных путей между всеми парами вершин графа.

Начало. Инициализация матриц. Полагаем $\Lambda^{(1)} = [0]$, номер итерации $i = 1$. Матрицу H инициализируем следующим образом

$$h_{j,k} = \begin{cases} j, & \text{если } c_{j,k} < \infty \text{ или } j = k; \\ 0, & \text{иначе.} \end{cases}$$

Если $c_{j,k} = \infty$, полагаем $h_{j,k} = 0$, иначе $h_{j,k} = k$.

Шаг 1. Если $i = p$, идем на шаг 5, иначе полагаем $i = i + 1$.

Шаг 2. Находим строку и столбец с номером i матрицы минимальных путей. Полагаем $\lambda_{i,i}^{(i)} = 0$. Для всех вершин $j : 1 \leq j < i$ вычисляем

$$\lambda_{j,i}^{(i)} = \min_{k \in \{1, \dots, i-1\}} \{\lambda_{j,k}^{(i-1)} + c_{k,i}\};$$

$$\lambda_{i,j}^{(i)} = \min_{k \in \{1, \dots, i-1\}} \{c_{i,k} + \lambda_{k,j}^{(i-1)}\}.$$

Если значение $\lambda_{j,i}^{(i)} < c_{j,i}$ ($\lambda_{i,j}^{(i)} < c_{i,j}$), полагаем $h_{j,i} = h_{j,k}$ ($h_{i,j} = k$), где k – вершина, на которой достигнуто значение $\lambda_{j,i}^{(i)}$ ($\lambda_{i,j}^{(i)}$).

Шаг 3. Пересчитываем элементы матрицы, полученные на предыдущих итерациях. Для всех пар вершин $j, k : 1 \leq j < i; 1 \leq k < i$ полагаем

$$\lambda_{j,k}^{(i)} = \min\{\lambda_{j,k}^{(i-1)}, \lambda_{j,i}^{(i)} + \lambda_{i,k}^{(i)}\}.$$

Если значение $\lambda_{j,k}^{(i)} < \lambda_{j,k}^{(i-1)}$ («новый» путь лучше «старого»), полагаем $h_{j,k} = h_{j,i}$ (запоминаем новый путь).

Шаг 4. Если для некоторой вершины j значение $\lambda_{j,j}^{(i)} < 0$, то задача не имеет решения, идем на конец. Иначе идем на шаг 1.

Шаг 5. Находим минимальные пути между всеми парами вершин. Пусть путь $\langle j, k \rangle_{\text{мин}}$ имеет вид

$$\begin{aligned} \langle j, k \rangle_{\text{мин}} &= x_0 x_2 \dots x_l; \\ x_0 &= j, \quad x_l = k. \end{aligned}$$

Путь восстанавливаем от начала к концу по матрице H . Сначала полагаем $i = 1$. Очередную вершину x_i находим следующим образом:

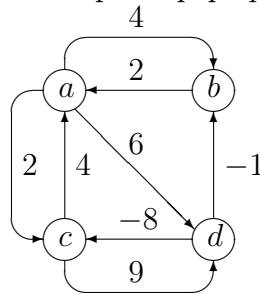
$$x_i = h_{x_{i-1}k}, \quad i > 0.$$

Если $x_i \neq k$, полагаем $i = i + 1$ и опять вычисляем очередную вершину.

Конец.

Вычислительная сложность алгоритма. На шаге 2 вычисляется $2(i - 1)$ элементов матрицы, для вычисления каждого элемента перебирается $i - 1$ вершина: вычисляется сумма двух чисел и осуществляется сравнение. На шаге 3 вычисляется $(i - 1)^2$ элементов матрицы, для каждого вычисления требуется одно сложение и одно сравнение. Итак, на шагах 2–3 выполняется порядка $(i - 1)^2$ операций, где $i \leq p$. Всего эти шаги повторяются $p - 1$ раз, значит, вычислительная сложность алгоритма p^3 .

Пример. Рассмотрим орграф из предыдущего примера.



Начало. Полагаем номер итерации $i = 1$. Матрицы $\Lambda^{(1)}$ и H имеют вид

$$\Lambda^{(1)} = \begin{array}{c|c} & a \\ \hline a & a \\ & 0 \end{array} \quad H = \begin{array}{c|cccc} & a & b & c & d \\ \hline a & a & b & c & d \\ b & a & b & 0 & 0 \\ c & a & 0 & c & d \\ d & 0 & b & c & d \end{array}$$

Шаг 1. Поскольку $i < p$, полагаем $i = 2$.

Шаг 2. Находим строку b и столбец b матрицы минимальных путей. Полагаем $\lambda_{b,b}^{(2)} = 0$. Вычисляем

$$\begin{aligned} \lambda_{b,a}^{(2)} &= c_{b,a} + \lambda_{a,a}^{(1)} = 2 + 0 = 2 = c_{b,a}; \\ \lambda_{a,b}^{(2)} &= \lambda_{a,a}^{(1)} + c_{a,b} = 0 + 4 = 4 = c_{a,b}. \end{aligned}$$

Шаг 3. Пересчитываем элементы матрицы, полученные на первой итерации:

$$\lambda_{a,a}^{(2)} = \min\{\lambda_{a,a}^{(1)}, \lambda_{a,b}^{(2)} + \lambda_{b,a}^{(2)}\} = \min\{0, 4 + 2\} = 0 = \lambda_{a,a}^{(1)}.$$

Матрицы $\Lambda^{(2)}$ и H имеют вид

$$\Lambda^{(2)} = \begin{array}{c|cc} & a & b \\ \hline a & 0 & 4 \\ b & 2 & 0 \end{array} \quad H = \begin{array}{c|cccc} & a & b & c & d \\ \hline a & a & b & c & d \\ b & a & b & 0 & 0 \\ c & a & 0 & c & d \\ d & 0 & b & c & d \end{array}$$

Шаг 4. Поскольку для всех вершин j значение $\lambda_{j,j}^{(2)} = 0$, идем на шаг 1.

Шаг 1. Поскольку $i < p$, полагаем $i = 3$.

Шаг 2. Находим строку c и столбец c матрицы минимальных путей. Полагаем $\lambda_{c,c}^{(3)} = 0$. Вычисляем

$$\begin{aligned} \lambda_{c,a}^{(3)} &= \min\{c_{c,a} + \lambda_{a,a}^{(2)}, c_{c,b} + \lambda_{b,a}^{(2)}\} = \min\{4 + 0, \infty + 2\} = 4 = c_{c,a}; \\ \lambda_{c,b}^{(3)} &= \min\{c_{c,a} + \lambda_{a,b}^{(2)}, c_{c,b} + \lambda_{b,b}^{(2)}\} = \min\{4 + 4, \infty + 0\} = 8 < c_{c,b}, \quad h_{c,b} = a; \\ \lambda_{a,c}^{(3)} &= \min\{\lambda_{a,a}^{(2)} + c_{a,c}, \lambda_{a,b}^{(2)} + c_{b,c}\} = \min\{0 + 2, 4 + \infty\} = 2 = c_{a,c}; \\ \lambda_{b,c}^{(3)} &= \min\{\lambda_{b,a}^{(2)} + c_{a,c}, \lambda_{b,b}^{(2)} + c_{b,c}\} = \min\{2 + 2, 0 + \infty\} = 4 < c_{b,c}, \quad h_{b,c} = h_{b,a} = a. \end{aligned}$$

Шаг 3. Пересчитываем элементы матрицы, полученные на первой итерации:

$$\begin{aligned} \lambda_{a,a}^{(3)} &= \min\{\lambda_{a,a}^{(2)}, \lambda_{a,c}^{(3)} + \lambda_{c,a}^{(3)}\} = \min\{0, 2 + 4\} = 0 = \lambda_{a,a}^{(2)}; \\ \lambda_{a,b}^{(3)} &= \min\{\lambda_{a,b}^{(2)}, \lambda_{a,c}^{(3)} + \lambda_{c,b}^{(3)}\} = \min\{4, 2 + \infty\} = 4 = \lambda_{a,b}^{(2)}; \\ \lambda_{b,a}^{(3)} &= \min\{\lambda_{b,a}^{(2)}, \lambda_{b,c}^{(3)} + \lambda_{c,a}^{(3)}\} = \min\{2, 4 + 4\} = 2 = \lambda_{b,a}^{(2)}; \\ \lambda_{b,b}^{(3)} &= \min\{\lambda_{b,b}^{(2)}, \lambda_{b,c}^{(3)} + \lambda_{c,b}^{(3)}\} = \min\{0, 4 + \infty\} = 0 = \lambda_{b,b}^{(2)}. \end{aligned}$$

Матрицы $\Lambda^{(3)}$ и H имеют вид

$$\Lambda^{(3)} = \begin{array}{c|ccc} & a & b & c \\ \hline a & 0 & 4 & 2 \\ b & 2 & 0 & 4 \\ c & 4 & 8 & 0 \end{array} \quad H = \begin{array}{c|cccc} & a & b & c & d \\ \hline a & a & b & c & d \\ b & a & b & a & 0 \\ c & a & a & c & d \\ d & 0 & b & c & d \end{array}$$

Шаг 4. Поскольку для всех вершин j значение $\lambda_{j,j}^{(3)} = 0$, идем на шаг 1.

Шаг 1. Поскольку $i < p$, полагаем $i = 4$.

Шаг 2. Находим строку d и столбец d матрицы минимальных путей. Полагаем $\lambda_{d,d}^{(4)} = 0$. Вычисляем

$$\begin{aligned} \lambda_{d,a}^{(4)} &= \min\{c_{d,a} + \lambda_{a,a}^{(3)}, c_{d,b} + \lambda_{b,a}^{(3)}, c_{d,c} + \lambda_{c,a}^{(3)}\} = \min\{\infty + 0, -1 + 2, -8 + 4\} = \\ &= -4 < c_{d,a}, \\ & \quad h_{d,a} = c; \\ \lambda_{d,b}^{(4)} &= \min\{c_{d,a} + \lambda_{a,b}^{(3)}, c_{d,b} + \lambda_{b,b}^{(3)}, c_{d,c} + \lambda_{c,b}^{(3)}\} = \min\{\infty + 4, -1 + 0, -8 + 8\} = \\ &= -1 = c_{d,b}; \\ \lambda_{d,c}^{(4)} &= \min\{c_{d,a} + \lambda_{a,c}^{(3)}, c_{d,b} + \lambda_{b,c}^{(3)}, c_{d,c} + \lambda_{c,c}^{(3)}\} = \min\{\infty + 2, -1 + 4, -8 + 0\} = \\ &= -8 = c_{d,c}; \\ \lambda_{a,d}^{(4)} &= \min\{\lambda_{a,a}^{(3)} + c_{a,d}, \lambda_{a,b}^{(3)} + c_{b,d}, \lambda_{a,c}^{(3)} + c_{c,d}\} = \min\{0 + 6, 4 + \infty, 2 + 9\} = 6 = c_{a,d}; \\ \lambda_{b,d}^{(4)} &= \min\{\lambda_{b,a}^{(3)} + c_{a,d}, \lambda_{b,b}^{(3)} + c_{b,d}, \lambda_{b,c}^{(3)} + c_{c,d}\} = \min\{2 + 6, 0 + \infty, 4 + 9\} = 8 < c_{b,d}, \\ & \quad h_{b,d} = h_{b,a} = a; \\ \lambda_{c,d}^{(4)} &= \min\{\lambda_{c,a}^{(3)} + c_{a,d}, \lambda_{c,b}^{(3)} + c_{b,d}, \lambda_{c,c}^{(3)} + c_{c,d}\} = \min\{4 + 6, 8 + \infty, 0 + 9\} = 9 = c_{c,d}. \end{aligned}$$

Шаг 3. Пересчитываем элементы матрицы, полученные на первой итерации:

$$\begin{aligned}
\lambda_{a,a}^{(4)} &= \min\{\lambda_{a,a}^{(3)}, \lambda_{a,d}^{(4)} + \lambda_{d,a}^{(4)}\} = \min\{0, 6 - 4\} = 0 = \lambda_{a,a}^{(3)}; \\
\lambda_{a,b}^{(4)} &= \min\{\lambda_{a,b}^{(3)}, \lambda_{a,d}^{(4)} + \lambda_{d,b}^{(4)}\} = \min\{4, 6 - 1\} = 4 = \lambda_{a,b}^{(3)}; \\
\lambda_{a,c}^{(4)} &= \min\{\lambda_{a,c}^{(3)}, \lambda_{a,d}^{(4)} + \lambda_{d,c}^{(4)}\} = \min\{2, 6 - 8\} = -2 < \lambda_{a,c}^{(3)}, \quad h_{a,c} = h_{a,d} = d; \\
\lambda_{b,a}^{(4)} &= \min\{\lambda_{b,a}^{(3)}, \lambda_{b,d}^{(4)} + \lambda_{d,a}^{(4)}\} = \min\{2, 8 - 4\} = 2 = \lambda_{b,a}^{(3)}; \\
\lambda_{b,b}^{(4)} &= \min\{\lambda_{b,b}^{(3)}, \lambda_{b,d}^{(4)} + \lambda_{d,b}^{(4)}\} = \min\{0, 8 - 1\} = 0 = \lambda_{b,b}^{(3)}; \\
\lambda_{b,c}^{(4)} &= \min\{\lambda_{b,c}^{(3)}, \lambda_{b,d}^{(4)} + \lambda_{d,c}^{(4)}\} = \min\{4, 8 - 8\} = 0 < \lambda_{b,c}^{(3)}, \quad h_{b,c} = h_{b,d} = a; \\
\lambda_{c,a}^{(4)} &= \min\{\lambda_{c,a}^{(3)}, \lambda_{c,d}^{(4)} + \lambda_{d,a}^{(4)}\} = \min\{4, 9 - 4\} = 4 = \lambda_{c,a}^{(3)}; \\
\lambda_{c,b}^{(4)} &= \min\{\lambda_{c,b}^{(3)}, \lambda_{c,d}^{(4)} + \lambda_{d,b}^{(4)}\} = \min\{8, 9 - 1\} = 8 = \lambda_{c,b}^{(3)}; \\
\lambda_{c,c}^{(4)} &= \min\{\lambda_{c,c}^{(3)}, \lambda_{c,d}^{(4)} + \lambda_{d,c}^{(4)}\} = \min\{0, 9 - 8\} = 0 = \lambda_{c,c}^{(3)}.
\end{aligned}$$

Матрицы $\Lambda^{(4)}$ и H имеют вид

$$\Lambda^{(4)} = \begin{array}{c|cccc} & a & b & c & d \\ \hline a & 0 & 4 & -2 & 6 \\ b & 2 & 0 & 0 & 8 \\ c & 4 & 8 & 0 & 9 \\ d & -4 & -1 & -8 & 0 \end{array} \quad H = \begin{array}{c|cccc} & a & b & c & d \\ \hline a & a & b & d & d \\ b & a & b & a & a \\ c & a & a & c & d \\ d & c & b & c & d \end{array}$$

Шаг 4. Поскольку для всех вершин j значение $\lambda_{j,j}^{(4)} = 0$, идем на шаг 1.

Шаг 1. Поскольку $i = p$, идем на шаг 5.

Шаг 5. Минимальные пути восстанавливаются так же, как и в алгоритме Флойда. Обратим внимание, что матрицы $\Lambda^{(4)}$ и H совпадают с матрицами Λ и H , полученными алгоритмом Флойда.

Итак, нами рассмотрены два алгоритма поиска минимальных путей между всеми парами вершин. Вычислительная сложность обоих алгоритмов равна p^3 .

3.4 Поиск k минимальных путей

В предыдущем подразделе рассмотрена задача поиска минимальных путей между вершинами графа, при этом на пути не накладываются никакие дополнительные условия. Но во многих приложениях требуется минимальный путь из обладающих какими-либо дополнительными свойствами (например, проходить через определенное подмножество вершин или ребер, содержать определенное число дуг и т. д.). Можно рассматривать подобную задачу как задачу поиска минимального пути с дополнительными ограничениями, но в ряде случаев это приводит к значительному увеличению вычислительных затрат. Другой подход состоит в том, чтобы найти некоторое множество минимальных путей и выбрать из них те, которые обладают необходимыми свойствами.

Заметим, что k минимальных путей не обязательно, вообще говоря, будут простыми. Задача поиска k минимальных путей без требования их простоты является значительно менее сложной, и для ее решения достаточно модифицировать изученные ранее алгоритмы. Модификация будет состоять в том, что операции сложения и сравнения чисел заменяются на операции сложения и сравнения векторов (т. е. наборов чисел), т. к. теперь мы имеем дело с весами нескольких путей.

В следующем подразделе мы введем необходимые понятия и обозначения.

3.4.1 Пространство R_k

Пусть R_k – пространство k -мерных векторов, координаты которых различны и возрастают с ростом индекса:

$$R_k = \{[x_1, x_2, \dots, x_k] : x_1 < x_2 < \dots < x_k\}.$$

При этом несколько последних компонент вектора могут быть равны бесконечности.

Пусть $A = [a_1, a_2, \dots, a_k]$ и $B = [b_1, b_2, \dots, b_k]$ – два вектора из R_k .

Определение. Результатом *обобщенной операции сравнения* называется вектор $A + B$, составленный из k различных минимальных компонент векторов A и B , упорядоченных в порядке возрастания. Обозначается это следующим образом:

$$A + B = \min_k \{a_1, a_2, \dots, a_k, b_1, b_2, \dots, b_k\}.$$

Определение. Результатом *обобщенной операции сложения* называется вектор $A \times B$, составленный из k различных минимальных сумм $a_i + b_j$ компонент векторов A и B , упорядоченных в порядке возрастания. Обозначается это следующим образом:

$$A \times B = \min_k \{a_1 + b_1, \dots, a_k + b_1, a_1 + b_2, \dots, a_k + b_2, \dots, a_1 + b_k, \dots, a_k + b_k\}.$$

Пример. Рассмотрим вектора из пространства R_3 .

$$\begin{aligned} A &= [1, 2, 5], \quad B = [2, 3, 4]; \\ A + B &= \min_3 \{1, 2, 5, 2, 3, 4\} = [1, 2, 3]; \\ A \times B &= \min_3 \{1 + 2, 2 + 2, 5 + 2, 1 + 3, 2 + 3, 5 + 3, 1 + 4, 2 + 4, 5 + 4\} = [3, 4, 5]. \end{aligned}$$

Заметим, что, т. к. компоненты векторов упорядочены по возрастанию, обобщенная операция сравнения требует k обычных сравнений, а обобщенная операция сложения – k^2 обычных сложений и сравнений.

Установим приоритеты введенных операций следующим образом: обобщенная операция сложения выполняется раньше обобщенной операции сравнения, если другой порядок не определен с помощью скобок.

Пусть имеются два вектора: $A \in R_k^p$ и $B \in R_k^p$ (это вектора длины p , компонентами которых являются вектора пространства R_k). Введем *скалярное произведение* таких векторов по аналогии со скалярным произведением обычных векторов, но операции сложения и умножения чисел заменим соответственно на операции обобщенного сравнения и обобщенного сложения векторов.

Определение. Результатом *скалярного произведения* вектора $A \in R_k^p$ на вектор $B \in R_k^p$ будет вектор из пространства R_k следующего вида

$$(A, B) = A_1 \times B_1 + A_2 \times B_2 + \dots + A_p \times B_p,$$

где A_i, B_i – компоненты векторов A и B соответственно (т.е. вектора из пространства R_k).

Пример. Рассмотрим вектора из пространства R_3^4 .

$$\begin{aligned} A &= ([1, 2, 5], [-1, 3, 4], [2, 3, 6], [-2, -1, 0]), \\ B &= ([2, 3, 4], [-2, -1, 3], [1, 2, 4], [3, 4, 6]); \\ (A, B) &= [1, 2, 5] \times [2, 3, 4] + [-1, 3, 4] \times [-2, -1, 3] + \\ &+ [2, 3, 6] \times [1, 2, 4] + [-2, -1, 0] \times [3, 4, 6] = \\ &= [3, 4, 5] + [-3, -2, 1] + [3, 4, 5] + [1, 2, 3] = [-3, -2, 1]. \end{aligned}$$

Далее определим операцию *умножения матриц*, элементами которых являются вектора из пространства R_k . Рассмотрим две такие матрицы: $A : n \times l$ и $B : l \times m$.

Определение. Результатом *умножения матрицы A на матрицу B* будет матрица $C : n \times m$, элементы которой имеют вид

$$C_{i,j} = (A_{i,\cdot}, B_{\cdot,j}),$$

где $A_{i,\cdot}$ – i -я строка матрицы A , $B_{\cdot,j}$ – j -й столбец матрицы B .

Пример. Рассмотрим матрицы $A : 2 \times 3$ и $B : 3 \times 1$, элементами которых являются вектора из пространства R_3 . Произведение матриц AB будет иметь размерность 2×1 .

$$\begin{aligned} A &= \begin{pmatrix} [1, 2, 5] & [-1, 3, 4] & [2, 3, 6] \\ [0, 1, 3] & [2, 3, 4] & [-2, -1, 3] \end{pmatrix}, \quad B = \begin{pmatrix} [1, 2, 5] \\ [3, 4, 6] \\ [1, 2, 4] \end{pmatrix}; \\ AB &= \begin{pmatrix} [1, 2, 5] \times [1, 2, 5] + [-1, 3, 4] \times [3, 4, 6] + [2, 3, 6] \times [1, 2, 4] \\ [0, 1, 3] \times [1, 2, 5] + [2, 3, 4] \times [3, 4, 6] + [-2, -1, 3] \times [1, 2, 4] \end{pmatrix} = \\ &= \begin{pmatrix} [2, 3, 4] + [2, 3, 5] + [3, 4, 5] \\ [1, 2, 4] + [5, 6, 7] + [-1, 0, 1] \end{pmatrix} = \begin{pmatrix} [2, 3, 4] \\ [-1, 0, 1] \end{pmatrix}. \end{aligned}$$

Определение. *Сверткой* вектора $A \in R_k$ назовем вектор из R_k , вычисляемый по следующему правилу

$$A^C = A \times A \times \dots$$

Ясно, что если среди компонент вектора A есть отрицательные, то при каждой операции обобщенного сложения компоненты результирующего вектора будут уменьшаться, и результатом свертки будет вектор $A^C = [-\infty, -\infty, \dots, -\infty]$. Если все компоненты вектора A положительны, то при каждой операции обобщенного сложения компоненты результирующего вектора будут увеличиваться. В этом случае результатом свертки будет вектор $A^C = [\infty, \infty, \dots, \infty]$. Рассмотрим операцию свертки для вектора вида $[0, a_2, \dots, a_k]$. Если $a_2 < \infty$, то все компоненты свертки вектора меньше бесконечности, и первая компонента равна нулю, а вторая – a_2 . При этом, разумеется, нет необходимости выполнять обобщенную операцию сложения бесконечное число раз: при выполнении очередного обобщенного сложения суммы элементов могут только увеличиваться, а значит, при каждом обобщенном сложении находится как минимумом один из элементов результирующего вектора: a_3, a_4 , и т.д.. Следовательно, свертка требует не более $k - 2$ обобщенных сложений. Если $A = [0, \infty, \dots, \infty]$, то вектор свертки $A^C = [0, \infty, \dots, \infty]$.

Примеры. Рассмотрим вектора из пространства R_3 :

$$A = [-1, 2, 5], \quad B = [0, 3, \infty];$$

Вычислим свертку вектора A

$$\begin{aligned} A \times A &= \min_3\{-1 - 1, 2 - 1, 5 - 1, -1 + 2, 2 + 2, 5 + 2, -1 + 5, 2 + 5, 5 + 5\} = \\ &= [-2, 1, 4]; \\ A \times A \times A &= \min_3\{-2 - 1, 1 - 1, 4 - 1, -2 + 2, 1 + 2, 4 + 2, -2 + 5, 1 + 5, 4 + 5\} = \\ &= [-3, 0, 3]; \\ A \times A \times A \times A &= \\ &= \min_3\{-3 - 1, 0 - 1, 3 - 1, -3 + 2, 0 + 2, 3 + 2, -3 + 5, 0 + 5, 3 + 5\} = \\ &= [-4, -1, 2]; \\ &\dots \end{aligned}$$

Как видно из вычислений, за счет отрицательной компоненты -1 с каждым обобщенным сложением компоненты результирующего вектора уменьшаются, а значит, стремятся к $-\infty$ с увеличением числа обобщенных сложений. Найдем теперь свертку вектора B .

$$\begin{aligned} B \times B &= \min_3\{0 + 0, 3 + 0, \infty + 0, 0 + 3, 3 + 3, \infty + 3, 0 + \infty, 3 + \infty, \infty + \infty\} = \\ &= [0, 3, 6]; \\ B \times B \times B &= \min_3\{0 + 0, 3 + 0, 6 + 0, 0 + 3, 3 + 3, 6 + 3, 0 + \infty, 3 + \infty, 6 + \infty\} = \\ &= [0, 3, 6]. \end{aligned}$$

Поскольку после второго обобщенного сложения результат не изменился, то дальше выполнять вычисления смысла нет, и $B^C = [0, 3, 6]$.

3.4.2 Поиск k произвольных минимальных путей от заданной вершины

Рассмотрим *алгоритм двойного поиска*, результатом которого являются k первых минимальных путей от заданной вершины до остальных вершин графа.

Обобщая обозначения предыдущего подраздела, введем вектор $c_{i,j}$, содержащий длины k ребер (i, j) минимального веса. Если какие-либо два ребра имеют одинаковый вес, в векторе им соответствует одно число. Если число ребер (i, j) различного веса меньше k , то последние компоненты вектора полагаются равными бесконечности. При $i = j$ будем считать, что имеется петля (i, i) нулевого веса, т.е. $c_{i,i} = [0, \infty, \infty]$. Из векторов $c_{i,j}$ составим матрицу C . Будем предполагать, что граф не содержит петель, так что $c_{i,i} = [0, \infty, \dots, \infty]$

Через $\lambda_{i,j}$ обозначим вектор из пространства R_k , состоящий из k минимальных весов путей $\langle i, j \rangle$. Матрицу, составленную из этих векторов, обозначим соответственно через $\Lambda^{(m)}$ и Λ . Вектор $\lambda_{i,j}^{(0)} = c_{i,j}$, т.е. состоит из минимальных весов ребер. Через $\lambda_{i,j,l}^{(m)}$, $\lambda_{i,j,l}$ и $c_{i,j,l}$ будем обозначать l -е компоненты соответствующих векторов.

Пусть нам требуется найти k минимальных путей от вершины 1 до остальных вершин графа, тогда нам надо сформировать строку $\Lambda_{1,\cdot} = [\lambda_{1,1}, \lambda_{1,2}, \dots, \lambda_{1,p}]$

матрицы Λ . Сначала формируется строка $\Lambda_{1,\cdot}^{(0)} = [c_{1,1}, c_{1,2}, \dots, c_{1,p}]$, которая покомпонентно оценивает сверху строку $\Lambda_{1,\cdot}$. (т. е. каждый компонент вектора $\lambda_{1,j}$ не превосходит соответствующего компонента вектора $c_{1,j}$).

В основе алгоритма двойного поиска лежат следующие утверждения.

Утверждение 1. *Если вершина w лежит непосредственно перед вершиной v в t -м минимальном пути $P = \langle u, v \rangle$, где $u \neq v$, то часть этого пути $P' = \langle u, w \rangle$ является l -м минимальным путем $\langle u, w \rangle$, где $l \leq t$.*

Доказательство. От противного. Пусть найдется по крайней мере t путей $\langle u, w \rangle$, вес которых меньше веса P' , тогда, добавляя к ним ребро $(w, v) \in P$, получаем t путей, веса которых меньше веса P , что противоречит тому, что P – t -й минимальный путь.

Утверждение 2. *Если вершина w лежит непосредственно перед вершиной u в $t+1$ -м минимальном пути $P = \langle u, u \rangle$, то часть этого пути $P' = \langle u, w \rangle$ является l -м минимальным путем $\langle u, w \rangle$, где $l \leq t$.*

Доказательство. Заметим, что первый минимальный путь $\langle u, u \rangle$ имеет длину 0, т. е. вообще не содержит дуг. Значит, $t+1$ -й минимальный путь $\langle u, u \rangle$ является t -м минимальным путем $\langle u, u \rangle$ из путей, содержащих дуги. Согласно предыдущей лемме, $t+1$ -й минимальный путь $\langle u, u \rangle$ состоит из l -го минимального пути $\langle u, j \rangle$ и ребра (j, u) , где $l \leq t$.

Таким образом, используя операции обобщенного сравнения и сложения, можно записать условие минимальности весов путей $\langle 1, v \rangle$:

$$\lambda_{1,v} = \lambda_{1,1} \times c_{1,v} + \lambda_{1,2} \times c_{2,v} + \dots + \lambda_{1,p} \times c_{p,v}.$$

Отсюда следует условие, накладываемое на строку минимальных путей $\Lambda_{1,\cdot}$:

$$\Lambda_{1,\cdot} = \Lambda_{1,\cdot} C.$$

Алгоритм двойного поиска является обобщенным вариантом изученного нами алгоритма Беллмана-Мура: на очередной итерации происходит сравнение найденных ранее весов минимальных путей $\langle 1, v \rangle$ с весами путей, найденных следующим образом: для всех вершин x перебираются пути $\langle 1, v \rangle = 1 \dots xv$, при этом пути $1 \dots x$ также должны быть минимальными. Для этого на каждой итерации решается уравнение

$$\Lambda_{1,\cdot}^{(i)} = \Lambda_{1,\cdot}^{(i)} C + \Lambda_{1,\cdot}^{(i-1)}.$$

Для упрощения решения этого матричного уравнения проведем некоторые преобразования. Пусть матрица W образована из C заменой всех элементов векторов $c_{i,j}$ при $i \leq j$ символом ∞ , а матрица U – аналогичной заменой элементов векторов $c_{i,j}$ при $i \geq j$. Таким образом, матрица W является нижней треугольной подматрицей, а матрица U – верхней треугольной подматрицей матрицы C . Тогда уравнение можно переписать в виде

$$\Lambda_{1,\cdot}^{(i)} = \Lambda_{1,\cdot}^{(i)} W + \Lambda_{1,\cdot}^{(i)} U + \Lambda_{1,\cdot}^{(i-1)}.$$

Здесь исключается обобщенное сложение компонент вектора $\Lambda_{1,\cdot}^{(i)}$ с элементами, стоящими на главной диагонали матрицы C , поскольку эти элементы имеют вид $[0, \infty, \dots, \infty]$.

Рассмотрим подробно вычисление вектора $\Lambda_{1,\cdot}^{(i)}W$. Его компоненты – это скалярные произведения $\Lambda_{1,\cdot}^{(i)}$ на соответствующие столбцы матрицы W . Начнем с последней компоненты. Обозначим j -й столбец матрицы W через $W_{\cdot,j}$, имеем

$$\Lambda_{1,\cdot}^{(i)}W_{\cdot,p} = \lambda_{1,1}^{(i)} \times w_{1,p} + \lambda_{2,1}^{(i)} \times w_{2,p} + \dots + \lambda_{1,p}^{(i)} \times w_{p,p}.$$

По построению элементы матрицы $L_{i,j} = [\infty, \infty, \dots, \infty]$ при $i \leq j$, т.е. все компоненты столбца $L_{\cdot,p}$ равны $[\infty, \infty, \dots, \infty]$. Отсюда

$$\Lambda_{1,\cdot}^{(i)}W_{\cdot,p} = [\infty, \infty, \dots, \infty].$$

Найдем теперь предпоследнюю компоненту вектора $\Lambda_{1,\cdot}^{(i)}W$:

$$\Lambda_{1,\cdot}^{(i)}W_{\cdot,p-1} = \lambda_{1,1}^{(i)} \times w_{1,p-1} + \lambda_{2,1}^{(i)} \times w_{2,p-1} + \dots + \lambda_{1,p-1}^{(i)} \times w_{p,p-1}.$$

По построению все компоненты столбца $W_{\cdot,p-1}$, кроме $w_{p,p-1}$, равны $[\infty, \infty, \dots, \infty]$. Отсюда

$$\Lambda_{1,\cdot}^{(i)}L_{\cdot,p-1} = \lambda_{1,p}^{(i)} \times w_{p,p-1}.$$

Рассматривая поочередно все компоненты вектора $\Lambda_{1,\cdot}^{(i)}W$ и учитывая, что при $m \leq j$ элементы матрицы $w_{m,j} = [\infty, \infty, \dots, \infty]$, получаем

$$\Lambda_{1,\cdot}^{(i)}W_{\cdot,j} = \lambda_{1,j+1}^{(i)} \times w_{j+1,j} + \lambda_{1,j+2}^{(i)} \times w_{j+2,j} + \dots + \lambda_{1,p}^{(i)} \times w_{p,j} = \sum_{m=j+1}^p \lambda_{1,m}^{(i)} \times w_{m,j}.$$

Запишем это выражение более компактно, учтем при этом, что при $m > j$ элементы матрицы $w_{m,j} = c_{m,j}$:

$$\Lambda_{1,\cdot}^{(i)}W_{\cdot,j} = \sum_{m=j+1}^p \lambda_{1,m}^{(i)} \times c_{m,j}.$$

Знак \sum здесь обозначает обобщенную операцию сравнения. Таким образом, если вычисление компонент вектора $\Lambda_{1,\cdot}^{(i)}W$ проводить в порядке убывания номеров компонент, то очередная j -я компонента будет вычисляться через ранее вычисленные компоненты с номерами $j+1, \dots, p$.

Рассуждая аналогично и учитывая, что U является верхней треугольной матрицей, т.е. при $m \geq j$ элементы матрицы $u_{m,j} = [\infty, \infty, \dots, \infty]$, а при $m < j$ элементы матрицы $u_{m,j} = c_{m,j}$ получаем

$$\Lambda_{1,\cdot}^{(i)}U_{\cdot,j} = \sum_{m=1}^{j-1} \lambda_{1,m}^{(i)} \times c_{m,j}.$$

Таким образом, если вычисление компонент вектора $\Lambda_{1,\cdot}^{(i)}U$ проводить в порядке возрастания номеров компонент, то очередная j -я компонента будет вычисляться через ранее вычисленные компоненты с номерами $1, \dots, j-1$.

Рассмотрим теперь i -ю итерацию алгоритма двойного поиска. Она состоит из двух этапов. На первом этапе, называемом «обратный поиск», решается уравнение

$$\Lambda_{1,\cdot}^{(2i+1)} = \Lambda_{1,\cdot}^{(2i+1)}W + \Lambda_{1,\cdot}^{(2i)}.$$

Вычисления производятся по убыванию номеров компонент, начиная с последней компоненты, как было описано выше:

$$\begin{aligned}\lambda_{1,p}^{(2i+1)} &= \lambda_{1,p}^{(2i)}; \\ \lambda_{1,p-1}^{(2i+1)} &= \lambda_{1,p}^{(2i+1)} \times c_{p,p-1} + \lambda_{1,p-1}^{(2i)}; \\ &\dots\end{aligned}$$

Для очередной j -й компоненты имеем

$$\lambda_{1,j}^{(2i+1)} = \sum_{m=j+1}^p \lambda_{1,m}^{(i)} \times c_{m,j} + \lambda_{1,j}^{(2i)}.$$

На втором этапе, называемом «прямой поиск», решается уравнение

$$\Lambda_{1,\cdot}^{(2i+2)} = \Lambda_{1,\cdot}^{(2i+2)} U + \Lambda_{1,\cdot}^{(2i+1)}.$$

Вычисления производятся по возрастанию номеров компонент, начиная с первой компоненты, как было описано выше:

$$\begin{aligned}\lambda_{1,1}^{(2i+2)} &= \lambda_{1,1}^{(2i+1)}; \\ \lambda_{1,2}^{(2i+2)} &= \lambda_{1,1}^{(2i+2)} \times c_{1,2} + \lambda_{1,2}^{(2i+1)}; \\ &\dots\end{aligned}$$

Для очередной j -й компоненты имеем

$$\lambda_{1,j}^{(2i+2)} = \sum_{m=1}^{j-1} \lambda_{1,m}^{(2i+2)} \times c_{m,j} + \lambda_{1,j}^{(2i+1)}.$$

Алгоритм двойного поиска

Начало. Полагаем $\Lambda_{1,\cdot}^{(0)} = [c_{1,1}, c_{1,2}, \dots, c_{1,p}]$, номер итерации $i = 0$.

Шаг 1. Обратный поиск. Вычисляем $\Lambda_{1,\cdot}^{(2i+1)}$ по формулам

$$\lambda_{1,j}^{(2i+1)} = \sum_{m=j+1}^p \lambda_{1,m}^{(i)} \times c_{m,j} + \lambda_{1,j}^{(2i)}, \quad j = p, \dots, 1.$$

Шаг 2. Прямой поиск. Вычисляем $\Lambda_{1,\cdot}^{(2i+2)}$ по формулам

$$\lambda_{1,j}^{(2i+2)} = \sum_{m=1}^{j-1} \lambda_{1,m}^{(2i+2)} \times c_{m,j} + \lambda_{1,j}^{(2i+1)}, \quad j = 1, \dots, p.$$

Шаг 3. Если $\Lambda_{1,\cdot}^{(2i+2)} \neq \Lambda_{1,\cdot}^{(2i)}$, полагаем $i = i + 1$ и идем на шаг 1.

Шаг 4. Полагаем $\Lambda_{1,\cdot} = \Lambda_{1,\cdot}^{(2i)}$. Восстанавливаем минимальные пути от конца к началу из следующего соотношения (если оно верно, то вершина v стоит перед вершиной u на m -м минимальном пути $\langle 1, u \rangle$)

$$\lambda_{1,v,l} + C_{v,u,j} = \lambda_{1,u,m}, \quad 1 \leq l \leq m \leq k, \quad 1 \leq j \leq k.$$

Конец.

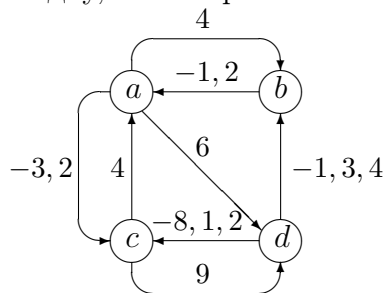
Обоснование алгоритма. Учитывая, что граф не содержит циклов отрицательного веса, имеем, что все первые минимальные пути являются простыми. Следовательно, $\Lambda_{1,}^{(0)}$ уже содержит вес по крайней мере одного первого минимального пути, кроме пути $\langle 1, 1 \rangle$ веса 0. В дальнейшем компоненты вектора $\Lambda_{1,}^{(i)}$ могут лишь уменьшаться, и не могут стать меньше весов минимальных путей, что следует из шагов 1–2 алгоритма.

Покажем, что на каждой итерации двойного поиска получается хотя бы одно новое значение веса одного из k минимальных путей. Предположим, что перед началом очередной итерации определены веса m минимальных путей до всех вершин. Рассмотрим некоторую вершину v , для которой вес $m + 1$ -го минимального пути еще не определен. Рассмотрим этот минимальный путь. Пусть u – вершина, лежащая на этом пути непосредственно перед v . Если уже найден вес $m + 1$ -го минимального пути до u , то, согласно утверждению 1, $m + 1$ -й путь до v определится на текущей итерации, т. к. он складывается из одного из $m + 1$ минимальных путей $\langle 1, u \rangle$ и ребра $\langle u, v \rangle$. Если вес $m + 1$ -го минимального пути до u не найден, и он необходим для построения $m + 1$ -го минимального пути до v , то предыдущие рассуждения могут быть повторены для вершины u в качестве v . Рассмотрев все такие вершины, мы либо найдем $m + 1$ -й минимальный путь до какой-либо вершины, либо, согласно утверждению 2, $m + 1$ -й минимальный путь до вершины 1. Следовательно, алгоритм двойного поиска сходится за конечное число шагов.

Следствие. Если после kr итераций оценочная строка $\Lambda_{1,}^{(i)}$ продолжает меняться, то граф содержит циклы отрицательного веса.

Вычислительная сложность алгоритма. Максимальное число итераций алгоритма – kr , на каждой итерации выполняется r^2 обобщенных сложений и сравнений, т. е. порядка k^2r^2 обычных сложений и сравнений. Итого вычислительная сложность k^3r^3 .

Пример. Рассмотрим мультиорграф, для упрощения диаграммы из кратных дуг оставим одну, на которой выпишем веса всех кратных дуг.



	a	b	c	d
a	$[0, \infty, \infty]$	$[4, \infty, \infty]$	$[-3, 2, \infty]$	$[6, \infty, \infty]$
b	$[-1, 2, \infty]$	$[0, \infty, \infty]$	$[\infty, \infty, \infty]$	$[\infty, \infty, \infty]$
c	$[4, \infty, \infty]$	$[\infty, \infty, \infty]$	$[0, \infty, \infty]$	$[9, \infty, \infty]$
d	$[\infty, \infty, \infty]$	$[-1, 3, 4]$	$[-8, 1, 2]$	$[0, \infty, \infty]$

Начало. Найдем три первых минимальных пути от вершины a до остальных вершин. Полагаем номер итерации $i = 0$,

$$\Lambda_{a,}^{(0)} = \left([0, \infty, \infty] \quad [4, \infty, \infty] \quad [-3, 2, \infty] \quad [6, \infty, \infty] \right).$$

Шаг 1. Обратный поиск. Вычисляем $\Lambda_{a,\cdot}^{(1)}$ по формулам

$$\begin{aligned}\lambda_{a,d}^{(1)} &= \lambda_{a,d}^{(0)} = [6, \infty, \infty]; \\ \lambda_{a,c}^{(1)} &= \lambda_{a,d}^{(1)} \times c_{d,c} + \lambda_{a,c}^{(0)} = [6, \infty, \infty] \times [-8, 1, 2] + [-3, 2, \infty] = [-3, -2, 1]; \\ \lambda_{a,b}^{(1)} &= \lambda_{a,d}^{(1)} \times c_{d,b} + \lambda_{a,c}^{(1)} \times c_{c,b} + \lambda_{a,b}^{(0)} = [6, \infty, \infty] \times [-1, 3, 4] + \\ &+ [-3, -2, 1] \times [\infty, \infty, \infty] + [4, \infty, \infty] = [4, 5, 9]; \\ \lambda_{a,a}^{(1)} &= \lambda_{a,d}^{(1)} \times c_{d,a} + \lambda_{a,c}^{(1)} \times c_{c,a} + \lambda_{a,b}^{(1)} \times c_{b,a} + \lambda_{a,a}^{(0)} = [6, \infty, \infty] \times [\infty, \infty, \infty] + \\ &+ [-3, -2, 1] \times [4, \infty, \infty] + [4, 5, 9] \times [-1, 2, \infty] + [0, \infty, \infty] = [0, 1, 2].\end{aligned}$$

Итак,

$$\Lambda_{a,\cdot}^{(1)} = \left(\begin{array}{cccc} [0, 1, 2] & [4, 5, 9] & [-3, -2, 1] & [6, \infty, \infty] \end{array} \right).$$

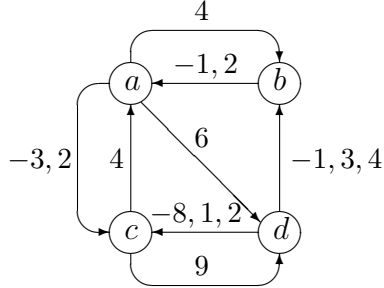
Шаг 2. Прямой поиск. Вычисляем $\Lambda_{a,\cdot}^{(2)}$ по формулам

$$\begin{aligned}\lambda_{a,a}^{(2)} &= \lambda_{a,a}^{(1)} = [0, 1, 2]; \\ \lambda_{a,b}^{(2)} &= \lambda_{a,a}^{(2)} \times c_{a,b} + \lambda_{a,b}^{(1)} = [0, 1, 2] \times [4, \infty, \infty] + [4, 5, 9] = [4, 5, 6]; \\ \lambda_{a,c}^{(2)} &= \lambda_{a,a}^{(2)} \times c_{a,c} + \lambda_{a,b}^{(2)} \times c_{b,c} + \lambda_{a,c}^{(1)} = [0, 1, 2] \times [-3, 2, \infty] + \\ &+ [4, 5, 6] \times [\infty, \infty, \infty] + [-3, -2, 1] = [-3, -2, -1]; \\ \lambda_{a,d}^{(2)} &= \lambda_{a,a}^{(2)} \times c_{a,d} + \lambda_{a,b}^{(2)} \times c_{b,d} + \lambda_{a,c}^{(2)} \times c_{c,d} + \lambda_{a,d}^{(1)} = [0, 1, 2] \times [6, \infty, \infty] + \\ &+ [4, 5, 6] \times [\infty, \infty, \infty] + [-3, -2, -1] \times [9, \infty, \infty] + [6, \infty, \infty] = [6, 7, 8].\end{aligned}$$

Итак,

$$\Lambda_{a,\cdot}^{(2)} = \left(\begin{array}{cccc} [0, 1, 2] & [4, 5, 6] & [-3, -2, -1] & [6, 7, 8] \end{array} \right).$$

Проверим, что действительно существуют пути с весами, найденными на шагах 1–2. Для этого рассмотрим диаграмму графа.



Из диаграммы видно можно визуальнo получить следующие пути (здесь через $w(\langle v, v \rangle_i)$ обозначен вес i -го минимального пути $\langle v, v \rangle$, а между вершинами в скобках указан вес ребра, включаемого в путь):

$$\begin{aligned}\langle a, a \rangle_1 &= a, & w(\langle a, a \rangle_1) &= 0; \\ \langle a, a \rangle_2 &= a_{(-3)}c_{(4)}a, & w(\langle a, a \rangle_2) &= -3 + 4 = 1; \\ \langle a, a \rangle_3 &= a_{(6)}d_{(-8)}c_{(4)}a, & w(\langle a, a \rangle_3) &= 6 - 8 + 4 = 2; \\ \langle a, b \rangle_1 &= a_{(4)}b, & w(\langle a, b \rangle_1) &= 4; \\ \langle a, b \rangle_2 &= a_{(6)}d_{(-1)}b, & w(\langle a, b \rangle_2) &= 6 - 1 = 5; \\ \langle a, b \rangle_3 &= a_{(6)}d_{(-8)}c_{(4)}a_{(4)}b, & w(\langle a, b \rangle_3) &= 6 - 8 + 4 + 4 = 6; \\ \langle a, c \rangle_1 &= a_{(-3)}c, & w(\langle a, c \rangle_1) &= -3; \\ \langle a, c \rangle_2 &= a_{(6)}d_{(-8)}c, & w(\langle a, c \rangle_2) &= 6 - 8 = -2; \\ \langle a, c \rangle_3 &= a_{(-3)}c_{(4)}a_{(6)}d_{(-8)}c, & w(\langle a, c \rangle_3) &= -3 + 4 + 6 - 8 = -1;\end{aligned}$$

$$\begin{aligned}
\langle a, d \rangle_1 &= a_{(6)}d, & w(\langle a, d \rangle_1) &= 6; \\
\langle a, d \rangle_2 &= a_{(-3)}c_{(4)}a_{(6)}d, & w(\langle a, d \rangle_2) &= -3 + 4 + 6 = 7; \\
\langle a, d \rangle_3 &= a_{(6)}d_{(-8)}c_{(4)}a_{(6)}d, & w(\langle a, d \rangle_3) &= 6 - 8 + 4 + 6 = 8.
\end{aligned}$$

Шаг 3. Поскольку $\Lambda_{a,\cdot}^{(2)} \neq \Lambda_{a,\cdot}^{(0)}$, полагаем $i = 1$ и идем на шаг 1.

Шаг 1. Обратный поиск. Вычисляем $\Lambda_{a,\cdot}^{(3)}$ по формулам

$$\begin{aligned}
\lambda_{a,d}^{(3)} &= \lambda_{a,d}^{(2)} = [6, 7, 8]; \\
\lambda_{a,c}^{(3)} &= \lambda_{a,d}^{(3)} \times c_{d,c} + \lambda_{a,c}^{(2)} = [6, 7, 8] \times [-8, 1, 2] + [-3, -2, -1] = [-3, -2, 1]; \\
\lambda_{a,b}^{(3)} &= \lambda_{a,d}^{(3)} \times c_{d,b} + \lambda_{a,c}^{(3)} \times c_{c,b} + \lambda_{a,b}^{(2)} = [6, 7, 8] \times [-1, 3, 4] + \\
&+ [-3, -2, -1] \times [\infty, \infty, \infty] + [4, 5, 6] = [4, 5, 6]; \\
\lambda_{a,a}^{(3)} &= \lambda_{a,d}^{(3)} \times c_{d,a} + \lambda_{a,c}^{(3)} \times c_{c,a} + \lambda_{a,b}^{(3)} \times c_{b,a} + \lambda_{a,a}^{(2)} = [6, 7, 8] \times [\infty, \infty, \infty] + \\
&+ [-3, -2, -1] \times [4, \infty, \infty] + [4, 5, 6] \times [-1, 2, \infty] + [0, 1, 2] = [0, 1, 2].
\end{aligned}$$

Итак,

$$\Lambda_{a,\cdot}^{(3)} = \begin{pmatrix} [0, 1, 2] & [4, 5, 6] & [-3, -2, -1] & [6, 7, 8] \end{pmatrix}.$$

Шаг 2. Прямой поиск. Вычисляем $\Lambda_{a,\cdot}^{(4)}$ по формулам

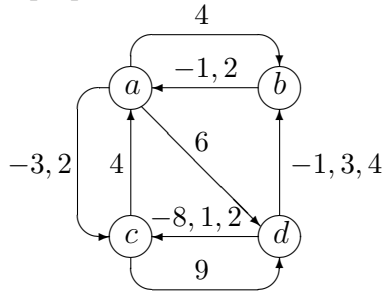
$$\begin{aligned}
\lambda_{a,a}^{(4)} &= \lambda_{a,a}^{(3)} = [0, 1, 2]; \\
\lambda_{a,b}^{(4)} &= \lambda_{a,a}^{(4)} \times c_{a,b} + \lambda_{a,b}^{(3)} = [0, 1, 2] \times [4, \infty, \infty] + [4, 5, 6] = [4, 5, 6]; \\
\lambda_{a,c}^{(4)} &= \lambda_{a,a}^{(4)} \times c_{a,c} + \lambda_{a,b}^{(4)} \times c_{b,c} + \lambda_{a,c}^{(3)} = [0, 1, 2] \times [-3, 2, \infty] + \\
&+ [4, 5, 6] \times [\infty, \infty, \infty] + [-3, -2, -1] = [-3, -2, -1]; \\
\lambda_{a,d}^{(4)} &= \lambda_{a,a}^{(4)} \times c_{a,d} + \lambda_{a,b}^{(4)} \times c_{b,d} + \lambda_{a,c}^{(4)} \times c_{c,d} + \lambda_{a,d}^{(3)} = [0, 1, 2] \times [6, \infty, \infty] + \\
&+ [4, 5, 6] \times [\infty, \infty, \infty] + [-3, -2, -1] \times [9, \infty, \infty] + [6, 7, 8] = [6, 7, 8].
\end{aligned}$$

Итак,

$$\Lambda_{a,\cdot}^{(4)} = \begin{pmatrix} [0, 1, 2] & [4, 5, 6] & [-3, -2, -1] & [6, 7, 8] \end{pmatrix}.$$

Шаг 3. Поскольку $\Lambda_{a,\cdot}^{(4)} = \Lambda_{a,\cdot}^{(2)}$, найдены веса трех первых минимальных путей.

Шаг 4. Положим $\Lambda_{a,\cdot} = \Lambda_{a,\cdot}^{(2)}$. Восстановим минимальные пути. Рассмотрим диаграмму графа вместе с весами минимальных путей.



$$\Lambda_{a,\cdot}^{(4)} = \begin{pmatrix} [0, 1, 2] & [4, 5, 6] & [-3, -2, -1] & [6, 7, 8] \end{pmatrix}$$

Первый минимальный путь $\langle a, a \rangle_1$ имеет вес $\lambda_{a,a,1} = 0$. Некоторые минимальные пути состоят из одного ребра, для этого должно выполняться соотношение

$$c_{a,u,j} = \lambda_{a,u,m}, \quad 1 \leq m \leq k, \quad 1 \leq j \leq k.$$

Проверим это соотношение для вершин b, c, d

$$\begin{aligned} c_{a,b,1} = 4 = \lambda_{a,b,1}, & \quad \langle a, b \rangle_1 = a_{(4)}b; \\ c_{a,c,1} = -3 = \lambda_{a,c,1}, & \quad \langle a, c \rangle_1 = a_{(-3)}c; \\ c_{a,c,2} = 2 > -1 = \lambda_{a,c,3}; \\ c_{a,d,1} = 6 = \lambda_{a,d,1}, & \quad \langle a, d \rangle_1 = a_{(6)}d. \end{aligned}$$

Итак, найдены все первые минимальные пути $\langle a, b \rangle_1, \langle a, c \rangle_1, \langle a, d \rangle_1$. Далее восстановим вторые и третьи минимальные пути из следующего соотношения

$$\lambda_{a,v,l} + c_{v,u,j} = \lambda_{a,u,m}, \quad 1 \leq l \leq m \leq k, \quad 1 \leq j \leq k.$$

Если оно верно, то вершина v стоит перед вершиной u на m -м минимальном пути $\langle a, u \rangle$. Проверим соотношение для всех найденных минимальных путей, при этом ограничимся рассмотрением случаев, когда $c_{v,u,j} < \infty$.

$$\begin{aligned} \lambda_{a,b,1} + c_{b,a,1} = 4 - 1 > 2 = \lambda_{a,a,3}; \\ \lambda_{a,c,1} + c_{c,a,1} = -3 + 4 = 1 = \lambda_{a,a,2}, & \quad \langle a, c \rangle_1 = a_{(-3)}c; \\ & \quad \langle a, a \rangle_2 = a_{(-3)}c_{(4)}a; \\ \lambda_{a,c,1} + c_{c,d,1} = -3 + 9 = 6 = \lambda_{a,d,1}, & \quad \langle a, c \rangle_1 = a_{(-3)}c; \\ & \quad \langle a, d \rangle_1 = a_{(-3)}c_{(9)}d; \\ \lambda_{a,d,1} + c_{d,b,1} = 6 - 1 = 5 = \lambda_{a,b,2}, & \quad \langle a, d \rangle_1 = a_{(6)}d; \\ & \quad \langle a, b \rangle_2 = a_{(6)}d_{(-1)}b; \\ & \quad \langle a, d \rangle_1 = a_{(-3)}c_{(9)}d, \\ & \quad \langle a, b \rangle_2 = a_{(-3)}c_{(9)}d_{(-1)}b; \\ \lambda_{a,d,1} + c_{d,b,2} = 6 + 3 = 9 > 6 = \lambda_{a,b,3}; \\ \lambda_{a,d,1} + c_{d,c,1} = 6 - 8 = -2 = \lambda_{a,c,2}, & \quad \langle a, d \rangle_1 = a_{(6)}d; \\ & \quad \langle a, c \rangle_2 = a_{(6)}d_{(-8)}c; \\ & \quad \langle a, d \rangle_1 = a_{(-3)}c_{(9)}d; \\ & \quad \langle a, c \rangle_2 = a_{(-3)}c_{(9)}d_{(8)}c. \end{aligned}$$

Итак, найден еще один первый минимальный путь $\langle a, d \rangle_1$ (имеются два пути $\langle a, d \rangle$ веса 6), один путь $\langle a, a \rangle_2$, и по два пути $\langle a, b \rangle_2$ и $\langle a, c \rangle_2$. Далее с их использованием будем восстанавливать следующие минимальные пути. Заметим, что, поскольку $\lambda_{a,b,1} + c_{b,a,1} > \lambda_{a,a,3}$, вершина b не может стоять перед a на минимальном пути, а поскольку вершина b смежна только с вершиной a , то путь $\langle a, b \rangle_2$ мы можем исключить из дальнейшего рассмотрения. Также можно не рассматривать ребра $(d, b)_2$ и $(d, b)_3$, поскольку $\lambda_{a,d,1} + c_{d,b,2} > \lambda_{a,b,3}$.

$$\begin{aligned} \lambda_{a,a,2} + c_{a,b,1} = 1 + 4 = 5 = \lambda_{a,b,3}, & \quad \langle a, a \rangle_2 = a_{(-3)}c_{(4)}a, & \quad \langle a, b \rangle_2 = a_{(-3)}c_{(4)}a_{(4)}b; \\ \lambda_{a,a,2} + c_{a,c,1} = 1 - 3 = -2 = \lambda_{a,c,2}, & \quad \langle a, a \rangle_2 = a_{(-3)}c_{(4)}a, & \quad \langle a, c \rangle_2 = a_{(-3)}c_{(4)}a_{(-3)}c; \\ \lambda_{a,a,2} + c_{a,c,2} = 1 + 2 = 3 > \lambda_{a,c,3}; \\ \lambda_{a,c,2} + c_{c,a,1} = -2 + 4 = 2 = \lambda_{a,a,3}, & \quad \langle a, c \rangle_2 = a_{(6)}d_{(-8)}c, & \quad \langle a, a \rangle_3 = a_{(6)}d_{(-8)}c_{(4)}a; \\ & \quad \langle a, c \rangle_2 = a_{(-3)}c_{(9)}d_{(-8)}c, & \quad \langle a, a \rangle_3 = a_{(-3)}c_{(9)}d_{(-8)}c_{(4)}a; \\ & \quad \langle a, c \rangle_2 = a_{(-3)}c_{(4)}a_{(-3)}c, & \quad \langle a, a \rangle_3 = a_{(-3)}c_{(4)}a_{(-3)}c_{(4)}a; \end{aligned}$$

$$\begin{aligned}
\lambda_{a,c,2} + c_{c,d,1} &= -2 + 9 = 7 = \lambda_{a,d,2}, \\
\langle a, c \rangle_2 &= a_{(6)}d_{(-8)}c, & \langle a, d \rangle_2 &= a_{(6)}d_{(-8)}c_{(9)}d; \\
\langle a, c \rangle_2 &= a_{(-3)}c_{(9)}d_{(-8)}c, & \langle a, d \rangle_2 &= a_{(-3)}c_{(9)}d_{(-8)}c_{(9)}d; \\
\langle a, c \rangle_2 &= a_{(-3)}c_{(4)}a_{(-3)}c, & \langle a, d \rangle_2 &= a_{(-3)}c_{(4)}a_{(-3)}c_{(9)}d; \\
\lambda_{a,d,2} + c_{d,b,1} &= 7 - 1 = 6 = \lambda_{a,b,3}, \\
\langle a, d \rangle_2 &= a_{(6)}d_{(-8)}c_{(9)}d, & \langle a, b \rangle_3 &= a_{(6)}d_{(-8)}c_{(9)}d_{(-1)}b; \\
\langle a, d \rangle_2 &= a_{(-3)}c_{(9)}d_{(-8)}c_{(9)}d, & \langle a, b \rangle_3 &= a_{(-3)}c_{(9)}d_{(-8)}c_{(9)}d_{(-1)}b; \\
\langle a, d \rangle_2 &= a_{(-3)}c_{(4)}a_{(-3)}c_{(9)}d, & \langle a, b \rangle_3 &= a_{(-3)}c_{(4)}a_{(-3)}c_{(9)}d_{(-1)}b; \\
\lambda_{a,d,2} + c_{d,c,1} &= 7 - 8 = -1 = \lambda_{a,c,3}, \\
\langle a, d \rangle_2 &= a_{(6)}d_{(-8)}c_{(9)}d, & \langle a, c \rangle_3 &= a_{(6)}d_{(-8)}c_{(9)}d_{(-8)}c; \\
\langle a, d \rangle_2 &= a_{(-3)}c_{(9)}d_{(-8)}c_{(9)}d, & \langle a, c \rangle_3 &= a_{(-3)}c_{(9)}d_{(-8)}c_{(9)}d_{(-8)}c; \\
\langle a, d \rangle_2 &= a_{(-3)}c_{(4)}a_{(-3)}c_{(9)}d, & \langle a, c \rangle_3 &= a_{(-3)}c_{(4)}a_{(-3)}c_{(9)}d_{(-8)}c.
\end{aligned}$$

Найдены минимальные пути $\langle a, a \rangle_3$, $\langle a, d \rangle_2$, $\langle a, b \rangle_3$ и $\langle a, c \rangle_3$. Далее аналогично находим третьи минимальные пути до вершины d .

$$\begin{aligned}
\lambda_{a,a,3} + c_{a,d,1} &= 2 + 6 = 8 = \lambda_{a,d,3}, \\
\langle a, a \rangle_3 &= a_{(6)}d_{(-8)}c_{(4)}a, & \langle a, d \rangle_3 &= a_{(6)}d_{(-8)}c_{(4)}a_{(6)}d; \\
\langle a, a \rangle_3 &= a_{(-3)}c_{(9)}d_{(-8)}c_{(4)}a, & \langle a, d \rangle_3 &= a_{(-3)}c_{(9)}d_{(-8)}c_{(4)}a_{(6)}d; \\
\langle a, a \rangle_3 &= a_{(-3)}c_{(4)}a_{(-3)}c_{(4)}a, & \langle a, d \rangle_3 &= a_{(-3)}c_{(4)}a_{(-3)}c_{(4)}a_{(9)}d; \\
\lambda_{a,c,3} + c_{c,d,1} &= -1 + 9 = 7 = \lambda_{a,d,3}, \\
\langle a, c \rangle_3 &= a_{(6)}d_{(-8)}c_{(9)}d_{(-8)}c, & \langle a, d \rangle_3 &= a_{(6)}d_{(-8)}c_{(9)}d_{(-8)}c_{(9)}d; \\
\langle a, c \rangle_3 &= a_{(-3)}c_{(9)}d_{(-8)}c_{(9)}d_{(-8)}c, & \langle a, d \rangle_3 &= a_{(-3)}c_{(9)}d_{(-8)}c_{(9)}d_{(-8)}c_{(9)}d; \\
\langle a, c \rangle_3 &= a_{(-3)}c_{(4)}a_{(-3)}c_{(9)}d_{(-8)}c, & \langle a, d \rangle_3 &= a_{(-3)}c_{(4)}a_{(-3)}c_{(9)}d_{(-8)}c_{(9)}d.
\end{aligned}$$

3.4.3 Поиск k произвольных минимальных путей между всеми парами вершин

Для поиска k минимальных путей между всеми парами вершин можно использовать обобщенные алгоритмы Флойда и Данцига. Изменение состоит в том, что каждый элемент матрицы $\Lambda^{(i)}$ – это вектор из R_k , состоящий из k весов минимальных путей между соответствующими вершинами, которые могут проходить только через вершины $\{1, 2, \dots, i\}$, а в качестве сравнения и сложения используются их обобщенные векторные эквиваленты. Предполагается, что граф не содержит циклов отрицательного веса, иначе задача поиска минимального пути не имеет решения.

Обобщенный алгоритм Флойда находит не только простые пути, в путях могут присутствовать циклы. Поэтому на втором шаге алгоритма Флойда матрица весов минимальных путей пересчитывается следующим образом: сначала находят веса k минимальных циклов $\langle i, i \rangle$, которые могут проходить только через вершины $\{1, 2, \dots, i\}$. Для этого находится свертка $\lambda_{i,i}^{(i-1)}$, т.к. $\lambda_{i,i}^{(i-1)}$ – вектор, содержащий веса k минимальных циклов $\langle i, i \rangle$, которые могут проходить только через вершины $\{1, 2, \dots, i-1\}$. Заметим, что первый такой путь имеет вес 0, а значит, результат свертки найдется не более чем за $k-2$ шагов. Затем с использованием этих циклов находятся веса проходящих через вершины $\{1, 2, \dots, i\}$ путей $\langle j, i \rangle$ и $\langle j, i \rangle$: к найденным на предыдущей итерации путям $\lambda_{j,i}^{(i-1)}$ и $\lambda_{i,j}^{(i-1)}$ могут быть добавлены найденные на текущей итерации циклы $\lambda_{i,i}^{(i)}$. После этого

с использованием найденных путей $\lambda_{j,i}^{(i)}$ и $\lambda_{i,j}^{(i)}$ вычисляются остальные элементы матрицы: для всех пар вершин $j \neq i, m \neq i$ происходит сравнение весов путей $\lambda_{m,j}^{(i-1)}$, найденных на предыдущей итерации, с весами путей, проходящих через вершину i .

Итак, на i -й итерации матрица весов путей вычисляется следующим образом:

$$\begin{aligned}\lambda_{i,i}^{(i)} &= (\lambda_{i,i}^{(i-1)})^C; \\ \lambda_{j,i}^{(i)} &= \lambda_{j,i}^{(i-1)} \times \lambda_{i,i}^{(i)}, & \forall j \neq i; \\ \lambda_{i,j}^{(i)} &= \lambda_{i,i}^{(i)} \times \lambda_{i,j}^{(i-1)}, & \forall j \neq i; \\ \lambda_{m,j}^{(i)} &= \lambda_{m,i}^{(i)} \times \lambda_{i,j}^{(i)} + \lambda_{m,j}^{(i-1)}, & \forall j \neq i, m \neq i.\end{aligned}$$

Еще раз подчеркнем, что для корректного определения весов путей нельзя менять порядок вычислений: сначала вычисляется элемент $\lambda_{i,i}^{(i)}$, затем $\lambda_{j,i}^{(i)}$ и $\lambda_{i,j}^{(i)}$ для всех вершин $j \neq i$, затем $\lambda_{m,j}^{(i)}$ для всех пар вершин $j \neq i, m \neq i$.

Рассмотрим теперь модификацию алгоритма Данцига. На втором шаге алгоритма сначала вычисляется элемент матрицы $\lambda_{i,i}^{(i)}$. Для этого вектор $\lambda_{i,i}^{(0)} = c_{i,i}$, первая компонента которого по определению равна нулю, а остальные – весам петель, обобщенно сравнивается с векторами, содержащими веса k минимальных циклов $\langle i, i \rangle$, которые могут проходить через вершины $\{1, 2, \dots, i-1\}$. Эти циклы состоят из ребер (i, m) , минимальных путей $\langle m, j \rangle$ и ребер (j, i) , для всех вершин $j < i$ и $m < i$. Минимальные веса таких циклов определяются в результате обобщенной операции сложения $\lambda_{i,m}^{(0)} \times \lambda_{m,j}^{(i-1)} \times \lambda_{j,i}^{(0)}$. Затем, после обобщенного сравнения всех полученных векторов, производится свертка итогового вектора.

Далее вычисляются элементы $\lambda_{j,i}^{(i)}$ и $\lambda_{i,j}^{(i)}$ для всех $j < i$. Пути $\langle j, i \rangle$ состоят из минимальных путей $\langle j, m \rangle$, ребер (m, i) и циклов $\langle i, i \rangle$, т.е. их веса вычисляются в результате обобщенной операции сложения $\lambda_{j,m}^{(i-1)} \times \lambda_{m,i}^{(0)} \times \lambda_{i,i}^{(i)}$. Затем проводится обобщенное сравнение таких векторов для всех $m < i$. Пути $\langle i, j \rangle$ строятся в обратной последовательности: сначала цикл $\langle i, i \rangle$, затем ребро (i, m) , затем путь $\langle m, j \rangle$, и их веса вычисляются аналогично. После этого с использованием найденных путей $\lambda_{j,i}^{(i)}$ и $\lambda_{i,j}^{(i)}$ вычисляются остальные элементы матрицы: для всех пар вершин $j \neq i, m \neq i$ происходит сравнение весов путей $\lambda_{m,j}^{(i-1)}$, найденных на предыдущей итерации, с весами путей, проходящих через вершину i .

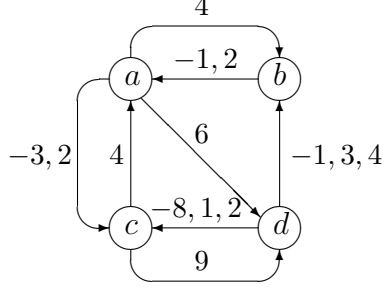
Итак, на втором шаге алгоритма Данцига матрица весов минимальных путей пересчитывается следующим образом.

$$\begin{aligned}\lambda_{i,i}^{(i)} &= \left(\lambda_{i,i}^{(0)} + \sum_{m=1}^{i-1} \sum_{j=1}^{i-1} \lambda_{i,m}^{(0)} \times \lambda_{m,j}^{(i-1)} \times \lambda_{j,i}^{(0)} \right)^C; \\ \lambda_{j,i}^{(i)} &= \sum_{m=1}^{i-1} \lambda_{j,m}^{(i-1)} \times \lambda_{m,i}^{(0)} \times \lambda_{i,i}^{(i)}, & \forall j < i; \\ \lambda_{i,j}^{(i)} &= \sum_{m=1}^{i-1} \lambda_{i,i}^{(i)} \times \lambda_{i,m}^{(0)} \times \lambda_{m,j}^{(i-1)}, & \forall j < i; \\ \lambda_{m,j}^{(i)} &= \lambda_{m,i}^{(i)} \times \lambda_{i,j}^{(i)} + \lambda_{m,j}^{(i-1)}, & \forall j < i, m < i.\end{aligned}$$

Вычислительная сложность алгоритмов. Поскольку обобщенные варианты алгоритмов аналогичны их обычным вариантам, с учетом того, что сравнение и

сложение заменяются на их векторные эквиваленты, то в ходе алгоритма выполняется p^3 обобщенных сравнений и сложений. Каждое обобщенное сравнение требует не более чем k обычных сравнений, а обобщенное сложение – не более чем k^2 обычных сравнений и сложений, следовательно, вычислительная сложность обобщенных вариантов равна $k^2 p^3$.

Пример. Продемонстрируем работу обобщенного алгоритма Флойда. Рассмотрим мультиорграф из предыдущего примера.



	a	b	c	d
$\Lambda^{(0)} = a$	$[0, \infty, \infty]$	$[4, \infty, \infty]$	$[-3, 2, \infty]$	$[6, \infty, \infty]$
b	$[-1, 2, \infty]$	$[0, \infty, \infty]$	$[\infty, \infty, \infty]$	$[\infty, \infty, \infty]$
c	$[4, \infty, \infty]$	$[\infty, \infty, \infty]$	$[0, \infty, \infty]$	$[9, \infty, \infty]$
d	$[\infty, \infty, \infty]$	$[-1, 3, 4]$	$[-8, 1, 2]$	$[0, \infty, \infty]$

На первой итерации рассматриваем пути, которые могут проходить через вершину a . Пересчитаем матрицу Λ . Сначала вычислим элемент $\lambda_{a,a}^{(a)}$:

$$\lambda_{a,a}^{(a)} = (\lambda_{a,a}^{(0)})^C = [0, \infty, \infty]^C = [0, \infty, \infty].$$

Далее пересчитаем остальные элементы строки a и столбца a :

$$\begin{aligned} \lambda_{b,a}^{(a)} &= \lambda_{b,a}^{(0)} \times \lambda_{a,a}^{(a)} = [-1, 2, \infty] \times [0, \infty, \infty] = [-1, 2, \infty]; \\ \lambda_{c,a}^{(a)} &= \lambda_{c,a}^{(0)} \times \lambda_{a,a}^{(a)} = [4, \infty, \infty] \times [0, \infty, \infty] = [4, \infty, \infty]; \\ \lambda_{d,a}^{(a)} &= \lambda_{d,a}^{(0)} \times \lambda_{a,a}^{(a)} = [\infty, \infty, \infty] \times [0, \infty, \infty] = [\infty, \infty, \infty]; \\ \lambda_{a,b}^{(a)} &= \lambda_{a,a}^{(a)} \times \lambda_{a,b}^{(0)} = [0, \infty, \infty] \times [4, \infty, \infty] = [4, \infty, \infty]; \\ \lambda_{a,c}^{(a)} &= \lambda_{a,a}^{(a)} \times \lambda_{a,c}^{(0)} = [0, \infty, \infty] \times [-3, 2, \infty] = [-3, 2, \infty]; \\ \lambda_{a,d}^{(a)} &= \lambda_{a,a}^{(a)} \times \lambda_{a,d}^{(0)} = [0, \infty, \infty] \times [6, \infty, \infty] = [6, \infty, \infty]. \end{aligned}$$

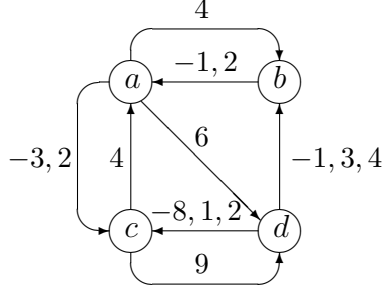
Теперь с использованием этих элементов пересчитаем остальные элементы матрицы:

$$\begin{aligned} \lambda_{b,b}^{(a)} &= \lambda_{b,a}^{(a)} \times \lambda_{a,b}^{(a)} + \lambda_{b,b}^{(0)} = [-1, 2, \infty] \times [4, \infty, \infty] + [0, \infty, \infty] = [0, 3, 6]; \\ \lambda_{b,c}^{(a)} &= \lambda_{b,a}^{(a)} \times \lambda_{a,c}^{(a)} + \lambda_{b,c}^{(0)} = [-1, 2, \infty] \times [-3, 2, \infty] + [\infty, \infty, \infty] = [-4, -1, 1]; \\ \lambda_{b,d}^{(a)} &= \lambda_{b,a}^{(a)} \times \lambda_{a,d}^{(a)} + \lambda_{b,d}^{(0)} = [-1, 2, \infty] \times [6, \infty, \infty] + [\infty, \infty, \infty] = [5, 8, \infty]; \\ \lambda_{c,b}^{(a)} &= \lambda_{c,a}^{(a)} \times \lambda_{a,b}^{(a)} + \lambda_{c,b}^{(0)} = [4, \infty, \infty] \times [4, \infty, \infty] + [\infty, \infty, \infty] = [8, \infty, \infty]; \\ \lambda_{c,c}^{(a)} &= \lambda_{c,a}^{(a)} \times \lambda_{a,c}^{(a)} + \lambda_{c,c}^{(0)} = [4, \infty, \infty] \times [-3, 2, \infty] + [0, \infty, \infty] = [0, 1, 6]; \\ \lambda_{c,d}^{(a)} &= \lambda_{c,a}^{(a)} \times \lambda_{a,d}^{(a)} + \lambda_{c,d}^{(0)} = [4, \infty, \infty] \times [6, \infty, \infty] + [9, \infty, \infty] = [9, 10, \infty]; \\ \lambda_{d,b}^{(a)} &= \lambda_{d,a}^{(a)} \times \lambda_{a,b}^{(a)} + \lambda_{d,b}^{(0)} = [\infty, \infty, \infty] \times [4, \infty, \infty] + [-1, 3, 4] = [-1, 3, 4]; \\ \lambda_{d,c}^{(a)} &= \lambda_{d,a}^{(a)} \times \lambda_{a,c}^{(a)} + \lambda_{d,c}^{(0)} = [\infty, \infty, \infty] \times [-3, 2, \infty] + [-8, 1, 2] = [-8, 1, 2]; \\ \lambda_{d,d}^{(a)} &= \lambda_{d,a}^{(a)} \times \lambda_{a,d}^{(a)} + \lambda_{d,d}^{(0)} = [\infty, \infty, \infty] \times [6, \infty, \infty] + [0, \infty, \infty] = [0, \infty, \infty]. \end{aligned}$$

Итак, получили матрицу $\Lambda^{(a)}$

	a	b	c	d
a	$[0, \infty, \infty]$	$[4, \infty, \infty]$	$[-3, 2, \infty]$	$[6, \infty, \infty]$
b	$[-1, 2, \infty]$	$[0, 3, 6]$	$[-4, -1, 1]$	$[5, 8, \infty]$
c	$[4, \infty, \infty]$	$[8, \infty, \infty]$	$[0, 1, 6]$	$[9, 10, \infty]$
d	$[\infty, \infty, \infty]$	$[-1, 3, 4]$	$[-8, 1, 2]$	$[0, \infty, \infty]$

Убедимся, что пути с такими весами, проходящие через вершину a , действительно существуют. Найдем их визуально по диаграмме графа.



$$\begin{aligned}
\langle b, b \rangle_2 &= b_{(-1)a(4)b}, & w(\langle b, b \rangle_2) &= -1 + 4 = 3; \\
\langle b, b \rangle_3 &= b_{(2)a(4)b}, & w(\langle b, b \rangle_3) &= 2 + 4 = 6; \\
\langle b, c \rangle_1 &= b_{(-1)a(-3)c}, & w(\langle b, c \rangle_1) &= -1 - 3 = -4; \\
\langle b, c \rangle_2 &= b_{(2)a(-3)c}, & w(\langle b, c \rangle_2) &= 2 - 3 = -1; \\
\langle b, c \rangle_3 &= b_{(-1)a(2)c}, & w(\langle b, c \rangle_3) &= -1 + 2 = 1; \\
\langle c, b \rangle_1 &= c_{(4)a(4)b}, & w(\langle c, b \rangle_1) &= 4 + 4 = 8; \\
\langle c, d \rangle_2 &= c_{(4)a(6)d}, & w(\langle c, d \rangle_2) &= 4 + 6 = 10.
\end{aligned}$$

На второй итерации рассматриваем пути, которые могут проходить также и через вершину b . Пересчитаем матрицу Λ . Сначала вычислим элемент $\lambda_{b,b}^{(b)}$:

$$\lambda_{b,b}^{(b)} = (\lambda_{b,b}^{(a)})^C = [0, 3, 6]^C = [0, 3, 6].$$

Теперь пересчитаем остальные элементы строки b и столбца b :

$$\begin{aligned}
\lambda_{a,b}^{(b)} &= \lambda_{a,b}^{(a)} \times \lambda_{b,b}^{(b)} = [4, \infty, \infty] \times [0, 3, 6] = [4, 7, 10]; \\
\lambda_{c,b}^{(b)} &= \lambda_{c,b}^{(a)} \times \lambda_{b,b}^{(b)} = [8, \infty, \infty] \times [0, 3, 6] = [8, 11, 14]; \\
\lambda_{d,b}^{(b)} &= \lambda_{d,b}^{(a)} \times \lambda_{b,b}^{(b)} = [-1, 3, 4] \times [0, 3, 6] = [-1, 2, 3]; \\
\lambda_{b,a}^{(b)} &= \lambda_{b,b}^{(b)} \times \lambda_{b,a}^{(a)} = [0, 3, 6] \times [-1, 2, 8] = [-1, 2, 5]; \\
\lambda_{b,c}^{(b)} &= \lambda_{b,b}^{(b)} \times \lambda_{b,c}^{(a)} = [0, 3, 6] \times [-4, -1, 1] = [-4, -1, 1]; \\
\lambda_{b,d}^{(b)} &= \lambda_{b,b}^{(b)} \times \lambda_{b,d}^{(a)} = [0, 3, 6] \times [5, 8, 11] = [5, 8, 11].
\end{aligned}$$

Далее пересчитаем остальные элементы матрицы:

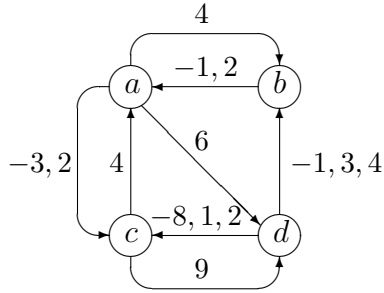
$$\begin{aligned}
\lambda_{a,a}^{(b)} &= \lambda_{a,b}^{(b)} \times \lambda_{b,a}^{(b)} + \lambda_{a,a}^{(a)} = [4, 7, 10] \times [-1, 2, 5] + [0, \infty, \infty] = [0, 3, 6]; \\
\lambda_{a,c}^{(b)} &= \lambda_{a,b}^{(b)} \times \lambda_{b,c}^{(b)} + \lambda_{a,c}^{(a)} = [4, 7, 10] \times [-4, -1, 1] + [-3, 2, \infty] = [-3, 0, 2]; \\
\lambda_{a,d}^{(b)} &= \lambda_{a,b}^{(b)} \times \lambda_{b,d}^{(b)} + \lambda_{a,d}^{(a)} = [4, 7, 10] \times [5, 8, 11] + [6, \infty, \infty] = [6, 9, 12]; \\
\lambda_{c,a}^{(b)} &= \lambda_{c,b}^{(b)} \times \lambda_{b,a}^{(b)} + \lambda_{c,a}^{(a)} = [8, 11, 14] \times [-1, 2, 5] + [4, \infty, \infty] = [4, 7, 10]; \\
\lambda_{c,c}^{(b)} &= \lambda_{c,b}^{(b)} \times \lambda_{b,c}^{(b)} + \lambda_{c,c}^{(a)} = [8, 11, 14] \times [-4, -1, 1] + [0, 1, 6] = [0, 1, 4]; \\
\lambda_{c,d}^{(b)} &= \lambda_{c,b}^{(b)} \times \lambda_{b,d}^{(b)} + \lambda_{c,d}^{(a)} = [8, 11, 14] \times [5, 8, 11] + [9, 10, \infty] = [9, 10, 13];
\end{aligned}$$

$$\begin{aligned}\lambda_{d,a}^{(b)} &= \lambda_{d,b}^{(b)} \times \lambda_{b,a}^{(b)} + \lambda_{c,a}^{(a)} = [-1, 2, 3] \times [-1, 2, 5] + [\infty, \infty, \infty] = [-2, 1, 2]; \\ \lambda_{d,c}^{(a)} &= \lambda_{d,b}^{(b)} \times \lambda_{b,c}^{(b)} + \lambda_{c,c}^{(a)} = [-1, 2, 3] \times [-4, -1, 1] + [-8, 1, 2] = [-8, -5, -2]; \\ \lambda_{d,d}^{(a)} &= \lambda_{d,b}^{(b)} \times \lambda_{b,d}^{(b)} + \lambda_{c,d}^{(a)} = [-1, 2, 3] \times [5, 8, 11] + [0, \infty, \infty] = [0, 4, 7].\end{aligned}$$

Итак, получили матрицу $\Lambda^{(b)}$

	a	b	c	d
a	[0, 3, 6]	[4, 7, 10]	[-3, 0, 2]	[6, 9, 12]
b	[-1, 2, 5]	[0, 3, 6]	[-4, -1, 1]	[5, 8, 11]
c	[4, 7, 10]	[8, 11, 14]	[0, 4, 7]	[9, 10, 13]
d	[-2, 1, 2]	[-1, 2, 3]	[-8, -5, -2]	[0, 4, 7]

Убедимся, что пути с такими весами, проходящие только через вершины a и b , действительно существуют. Найдем их визуально по диаграмме графа (выпишем только те пути, веса которых были найдены на второй итерации).

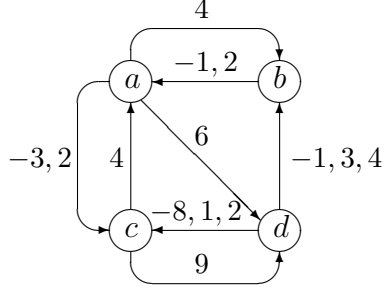


$\langle a, a \rangle_2 = a_{(4)}b_{(-1)}a,$	$w(\langle a, a \rangle_2) = 4 - 1 = 3;$
$\langle a, a \rangle_3 = a_{(4)}b_{(2)}a,$	$w(\langle a, a \rangle_3) = 4 + 2 = 6;$
$\langle a, b \rangle_2 = a_{(4)}b_{(-1)}a_{(4)}b$	$w(\langle a, b \rangle_2) = 4 - 1 + 4 = 7;$
$\langle a, b \rangle_3 = a_{(4)}b_{(2)}a_{(4)}b$	$w(\langle a, b \rangle_3) = 4 + 2 + 4 = 10;$
$\langle a, c \rangle_2 = a_{(4)}b_{(-1)}a_{(-3)}c$	$w(\langle a, c \rangle_2) = 4 - 1 - 3 = 0;$
$\langle a, d \rangle_2 = a_{(4)}b_{(-1)}a_{(6)}d$	$w(\langle a, d \rangle_2) = 4 - 1 + 6 = 9;$
$\langle a, d \rangle_3 = a_{(4)}b_{(2)}a_{(6)}d$	$w(\langle a, d \rangle_3) = 4 + 2 + 6 = 12;$
$\langle b, a \rangle_3 = b_{(-1)}a_{(4)}b_{(2)}a,$	$w(\langle b, a \rangle_3) = -1 + 4 + 2 = 5;$
$\langle b, d \rangle_3 = b_{(-1)}a_{(4)}b_{(2)}a_{(6)}d,$	$w(\langle b, d \rangle_3) = -1 + 4 + 2 + 6 = 11;$
$\langle c, a \rangle_2 = c_{(4)}a_{(4)}b_{(-1)}a,$	$w(\langle c, a \rangle_2) = 4 + 4 - 1 = 7;$
$\langle c, a \rangle_3 = c_{(4)}a_{(4)}b_{(2)}a,$	$w(\langle c, a \rangle_3) = 4 + 4 + 2 = 10;$
$\langle c, b \rangle_2 = c_{(4)}a_{(4)}b_{(-1)}a_{(4)}b,$	$w(\langle c, b \rangle_2) = 4 + 4 - 1 + 4 = 11;$
$\langle c, b \rangle_3 = c_{(4)}a_{(4)}b_{(2)}a_{(4)}b,$	$w(\langle c, b \rangle_3) = 4 + 4 + 2 + 4 = 14;$
$\langle c, c \rangle_3 = c_{(4)}a_{(4)}b_{(-1)}a_{(-3)}c,$	$w(\langle c, c \rangle_3) = 4 + 4 - 1 - 3 = 4;$
$\langle c, d \rangle_3 = c_{(4)}a_{(4)}b_{(-1)}a_{(-3)}c_{(9)}d,$	$w(\langle c, d \rangle_3) = 4 + 4 - 1 - 3 + 9 = 13;$
$\langle d, a \rangle_1 = d_{(-1)}b_{(-1)}a,$	$w(\langle d, a \rangle_1) = -1 - 1 = -2;$
$\langle d, a \rangle_2 = d_{(-1)}b_{(2)}a,$	$w(\langle d, a \rangle_2) = -1 + 2 = 1;$
$\langle d, a \rangle_3 = d_{(3)}b_{(-1)}a,$	$w(\langle d, a \rangle_3) = -1 + 3 = 2;$
$\langle d, b \rangle_2 = d_{(-1)}b_{(-1)}a_{(4)}b,$	$w(\langle d, b \rangle_2) = -1 - 1 + 4 = 2;$
$\langle d, c \rangle_2 = d_{(-1)}b_{(-1)}a_{(-3)}c,$	$w(\langle d, c \rangle_2) = -1 - 1 - 3 = -5;$
$\langle d, c \rangle_3 = d_{(-1)}b_{(2)}a_{(-3)}c,$	$w(\langle d, c \rangle_3) = -1 + 2 - 3 = -2;$
$\langle d, d \rangle_2 = d_{(-1)}b_{(-1)}a_{(6)}d,$	$w(\langle d, d \rangle_2) = -1 - 1 + 6 = 4;$
$\langle d, d \rangle_3 = d_{(-1)}b_{(2)}a_{(6)}d,$	$w(\langle d, d \rangle_3) = -1 + 2 + 6 = 7.$

Заметим, что для некоторых весов можно найти несколько путей, как мы могли убедиться на примере алгоритма двойного поиска. Здесь мы приводим лишь один путь каждого веса.

Поскольку все итерации алгоритма Флойда полностью аналогичны, мы не приводим здесь оставшиеся две итерации. Читателю предоставляется самостоятельно построить матрицы $\Lambda^{(c)}$ и $\Lambda^{(d)}$.

Пример. Продемонстрируем работу обобщенного алгоритма Данцига. Рассмотрим мультиорграф из предыдущего примера.



$$\Lambda^{(0)} = \begin{array}{c|cccc} & a & b & c & d \\ \hline a & [0, \infty, \infty] & [4, \infty, \infty] & [-3, 2, \infty] & [6, \infty, \infty] \\ b & [-1, 2, \infty] & [0, \infty, \infty] & [\infty, \infty, \infty] & [\infty, \infty, \infty] \\ c & [4, \infty, \infty] & [\infty, \infty, \infty] & [0, \infty, \infty] & [9, \infty, \infty] \\ d & [\infty, \infty, \infty] & [-1, 3, 4] & [-8, 1, 2] & [0, \infty, \infty] \end{array}$$

Построим матрицу $\Lambda^{(a)}$ размерности 1×1 .

$$\lambda_{a,a}^{(a)} = \left(\lambda_{a,a}^{(0)}\right)^C = [0, \infty, \infty]^C = [0, \infty, \infty].$$

Итак, получаем

$$\Lambda^{(a)} = \begin{array}{c|c} & a \\ \hline a & [0, \infty, \infty] \end{array}$$

Построим матрицу $\Lambda^{(b)}$ размерности 2×2 . Сначала вычислим элемент $\lambda_{b,b}^{(b)}$:

$$\begin{aligned} \lambda_{b,b}^{(b)} &= \left(\lambda_{b,b}^{(0)} + \lambda_{b,a}^{(0)} \times \lambda_{a,a}^{(a)} \times \lambda_{a,b}^{(0)}\right)^C = \\ &= ([0, \infty, \infty] + [-1, 2, \infty] \times [0, \infty, \infty] \times [4, \infty, \infty])^C = \\ &= ([0, \infty, \infty] + [3, 6, \infty])^C = ([0, 3, 6])^C = [0, 3, 6]. \end{aligned}$$

Затем вычислим остальные элементы строки b и столбца b :

$$\begin{aligned} \lambda_{a,b}^{(b)} &= \lambda_{a,a}^{(a)} \times \lambda_{a,b}^{(0)} \times \lambda_{b,b}^{(b)} = [0, \infty, \infty] \times [4, \infty, \infty] \times [0, 3, 6] = [4, 7, 10]; \\ \lambda_{b,a}^{(b)} &= \lambda_{b,b}^{(b)} \times \lambda_{b,a}^{(0)} \times \lambda_{a,a}^{(a)} = [0, 3, 6] \times [-1, 2, \infty] \times [0, \infty, \infty] = [-1, 2, 5]. \end{aligned}$$

Теперь вычислим элемент $\lambda_{a,a}^{(b)}$:

$$\lambda_{a,a}^{(b)} = \lambda_{a,b}^{(b)} \times \lambda_{b,a}^{(b)} + \lambda_{a,a}^{(a)} = [-1, 2, 5] \times [4, 7, 10] + [0, 3, 6] = [3, 6, 9] + [0, 3, 6] = [0, 3, 6].$$

Итак, получаем

$$\Lambda^{(b)} = \begin{array}{c|cc} & a & b \\ \hline a & [0, 3, 6] & [4, 7, 10] \\ b & [-1, 2, 5] & [0, 3, 6] \end{array}$$

Обратим внимание, что элементы матрицы совпадают с соответствующими элементами, найденными алгоритмом Флойда.

Построим матрицу $\Lambda^{(c)}$ размерности 3×3 . Сначала вычислим элемент $\lambda_{c,c}^{(c)}$:

$$\begin{aligned}\lambda_{c,c}^{(c)} &= \left(\lambda_{c,c}^{(0)} + \lambda_{c,a}^{(0)} \times \lambda_{a,a}^{(b)} \times \lambda_{a,c}^{(0)} + \lambda_{c,a}^{(0)} \times \lambda_{a,b}^{(b)} \times \lambda_{b,c}^{(0)} + \right. \\ &\quad \left. + \lambda_{c,b}^{(0)} \times \lambda_{b,a}^{(b)} \times \lambda_{a,c}^{(0)} + \lambda_{c,b}^{(0)} \times \lambda_{b,b}^{(b)} \times \lambda_{b,c}^{(0)} \right)^C = \\ &= ([0, \infty, \infty] + [4, \infty, \infty] \times [0, 3, 6] \times [-3, 2, \infty] + \\ &\quad + [4, \infty, \infty] \times [4, 7, 10] \times [\infty, \infty, \infty] + [\infty, \infty, \infty] \times [-1, 2, 5] \times [-3, 2, \infty] + \\ &\quad + [\infty, \infty, \infty] \times [0, 3, 6] \times [\infty, \infty, \infty])^C = \\ &= ([0, \infty, \infty] + [1, 4, 6] + [\infty, \infty, \infty] + [\infty, \infty, \infty] + [\infty, \infty, \infty])^C \\ &= ([0, 1, 4])^C = [0, 1, 2].\end{aligned}$$

Теперь вычислим остальные элементы строки c и столбца c :

$$\begin{aligned}\lambda_{a,c}^{(c)} &= \lambda_{a,a}^{(b)} \times \lambda_{a,c}^{(0)} \times \lambda_{c,c}^{(c)} + \lambda_{a,b}^{(b)} \times \lambda_{b,c}^{(0)} \times \lambda_{c,c}^{(c)} = \\ &= [0, 3, 6] \times [-3, 2, \infty] \times [0, 1, 2] + [4, 7, 10] \times [\infty, \infty, \infty] \times [0, 1, 2] = \\ &= [-3, -2, -1] + [\infty, \infty, \infty] = [-3, -2, -1]; \\ \lambda_{b,c}^{(c)} &= \lambda_{b,a}^{(b)} \times \lambda_{a,c}^{(0)} \times \lambda_{c,c}^{(c)} + \lambda_{b,b}^{(b)} \times \lambda_{b,c}^{(0)} \times \lambda_{c,c}^{(c)} = \\ &= [-1, 2, 5] \times [-3, 2, \infty] \times [0, 1, 2] + [0, 3, 6] \times [\infty, \infty, \infty] \times [0, 1, 2] = \\ &= [-4, -3, -2] + [\infty, \infty, \infty] = [-4, -3, -2]; \\ \lambda_{c,a}^{(c)} &= \lambda_{c,c}^{(c)} \times \lambda_{c,a}^{(0)} \times \lambda_{a,a}^{(b)} + \lambda_{c,c}^{(c)} \times \lambda_{c,b}^{(0)} \times \lambda_{b,a}^{(b)} = \\ &= [0, 1, 2] \times [4, \infty, \infty] \times [0, 3, 6] + [0, 1, 2] \times [\infty, \infty, \infty] \times [-1, 2, 5] = \\ &= [4, 5, 6] + [\infty, \infty, \infty] = [4, 5, 6]; \\ \lambda_{c,b}^{(c)} &= \lambda_{c,c}^{(c)} \times \lambda_{c,a}^{(0)} \times \lambda_{a,b}^{(b)} + \lambda_{c,c}^{(c)} \times \lambda_{c,b}^{(0)} \times \lambda_{b,b}^{(b)} = \\ &= [0, 1, 2] \times [4, \infty, \infty] \times [4, 7, 10] + [0, 1, 2] \times [\infty, \infty, \infty] \times [0, 3, 6] = \\ &= [8, 9, 10] + [\infty, \infty, \infty] = [8, 9, 10].\end{aligned}$$

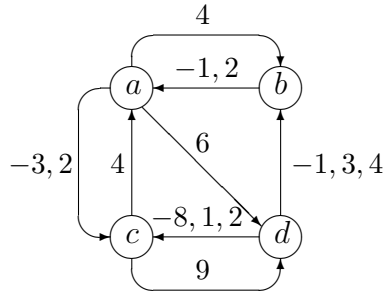
Далее пересчитаем остальные элементы матрицы:

$$\begin{aligned}\lambda_{a,a}^{(c)} &= \lambda_{a,c}^{(c)} \times \lambda_{c,a}^{(c)} + \lambda_{a,a}^{(b)} = [-3, -2, -1] \times [4, 5, 6] + [0, 3, 6] = \\ &= [1, 2, 3] + [0, 3, 6] = [0, 1, 2]; \\ \lambda_{a,b}^{(c)} &= \lambda_{a,c}^{(c)} \times \lambda_{c,b}^{(c)} + \lambda_{a,b}^{(b)} = [-3, -2, -1] \times [8, 9, 10] + [4, 7, 10] = \\ &= [5, 6, 7] + [4, 7, 10] = [4, 5, 6]; \\ \lambda_{b,a}^{(c)} &= \lambda_{b,c}^{(c)} \times \lambda_{c,a}^{(c)} + \lambda_{b,a}^{(b)} = [-4, -3, -2] \times [4, 5, 6] + [-1, 2, 5] = \\ &= [0, 1, 2] + [-1, 2, 5] = [-1, 0, 1]; \\ \lambda_{b,b}^{(c)} &= \lambda_{b,c}^{(c)} \times \lambda_{c,b}^{(c)} + \lambda_{b,b}^{(b)} = [-4, -3, -2] \times [8, 9, 10] + [0, 3, 6] = \\ &= [4, 5, 6] + [0, 3, 6] = [0, 3, 4].\end{aligned}$$

Итак, получаем матрицу $\Lambda^{(c)}$

$$\Lambda^{(c)} = \begin{array}{c|ccc} & a & b & c \\ \hline a & [0, 1, 2] & [4, 5, 6] & [-3, -2, -1] \\ b & [-1, 0, 1] & [0, 3, 4] & [-4, -3, -2] \\ c & [4, 5, 6] & [8, 9, 10] & [0, 1, 2] \end{array}$$

Убедимся, что пути с такими весами, проходящие только через вершины a , b и c , действительно существуют. Найдем их визуально по диаграмме графа. Выпишем только те пути, веса которых отличаются от весов путей, найденных алгоритмом Флойда, т.е. пути, содержащие в качестве промежуточной вершину c .



$\langle a, a \rangle_2 = a_{(-3)(4)}a,$	$w(\langle a, a \rangle_2) = -3 + 4 = 1;$
$\langle a, a \rangle_3 = a_{(-3)(4)}a_{(-3)(4)}a,$	$w(\langle a, a \rangle_3) = -3 + 4 - 3 + 4 = 2;$
$\langle a, b \rangle_2 = a_{(-3)(4)}a_{(4)}b$	$w(\langle a, b \rangle_2) = -3 + 4 + 4 = 5;$
$\langle a, b \rangle_3 = a_{(-3)(4)}a_{(-3)(4)}a_{(4)}b$	$w(\langle a, b \rangle_3) = -3 + 4 - 3 + 4 + 4 = 6;$
$\langle a, c \rangle_1 = a_{(-3)}c$	$w(\langle a, c \rangle_1) = -3;$
$\langle a, c \rangle_2 = a_{(-3)(4)}a_{(-3)}c$	$w(\langle a, c \rangle_2) = -3 + 4 - 3 = -2;$
$\langle a, c \rangle_3 = a_{(-3)(4)}a_{(-3)(4)}a_{(-3)}c$	$w(\langle a, c \rangle_3) = -3 + 4 - 3 + 4 - 3 = -1;$
$\langle b, a \rangle_2 = b_{(-1)}a_{(-3)(4)}a,$	$w(\langle b, a \rangle_2) = -1 - 3 + 4 = 0;$
$\langle b, a \rangle_3 = b_{(-1)}a_{(-3)(4)}a_{(-3)(4)}a,$	$w(\langle b, a \rangle_3) = -1 - 3 + 4 - 3 + 4 = 1;$
$\langle b, b \rangle_3 = b_{(-1)}a_{(-3)(4)}a_{(4)}b,$	$w(\langle b, b \rangle_3) = -1 - 3 + 4 + 4 = 4;$
$\langle b, c \rangle_2 = b_{(-1)}a_{(-3)(4)}a_{(-3)},$	$w(\langle b, c \rangle_2) = -1 - 3 + 4 - 3 = -3;$
$\langle b, c \rangle_3 = b_{(-1)}a_{(-3)(4)}a_{(-3)(4)}a_{(-3)},$	$w(\langle b, c \rangle_3) = -1 - 3 + 4 - 3 + 4 - 3 = -2;$
$\langle c, a \rangle_2 = c_{(4)}a_{(-3)}c_{(4)}a,$	$w(\langle c, a \rangle_2) = 4 - 3 + 4 = 5;$
$\langle c, a \rangle_3 = c_{(4)}a_{(-3)}c_{(4)}a_{(-3)}c_{(4)}a,$	$w(\langle c, a \rangle_3) = 4 - 3 + 4 - 3 + 4 = 6;$
$\langle c, b \rangle_2 = c_{(4)}a_{(-3)}c_{(4)}a_{(4)}b,$	$w(\langle c, b \rangle_2) = 4 - 3 + 4 + 4 = 9;$
$\langle c, b \rangle_3 = c_{(4)}a_{(-3)}c_{(4)}a_{(-3)}c_{(4)}a_{(4)}b,$	$w(\langle c, b \rangle_3) = 4 - 3 + 4 - 3 + 4 + 4 = 10;$
$\langle c, c \rangle_2 = c_{(4)}a_{(-3)}c,$	$w(\langle c, c \rangle_2) = 4 - 3 = 1;$
$\langle c, c \rangle_3 = c_{(4)}a_{(-3)}c_{(4)}a_{(-3)}c,$	$w(\langle c, c \rangle_3) = 4 - 3 + 4 - 3 = 2.$

Выполнить четвертую итерацию алгоритма Данцига, т.е. построить матрицу $\Lambda^{(d)}$ размерности 4×4 , предоставляется читателю.

Восстановление путей выполняется так же, как и в алгоритме двойного поиска.

3.4.4 Поиск k простых минимальных путей от заданной вершины

Предыдущие алгоритмы не гарантируют нахождения пути без циклов, однако в приложениях часто требуются простые пути. Модифицирование предложенных алгоритмов для поиска простых путей осуществить не совсем легко, поэтому рассмотрим другой алгоритм, ориентированный на поиск именно простых путей. Для простоты изложения будем предполагать, что граф не содержит кратных ребер, так как для мультиграфа алгоритм легко модифицировать.

Введем некоторые определения. Пусть P^m – m -й минимальный простой путь $\langle u, v \rangle$, который имеет вид

$$P^m = x_0^m x_1^m \dots x_{l_m}^m.$$

Обозначим через P_i^m отклонение от пути P^{m-1} в вершине i – минимальный из простых путей вида

$$x_0^{m-1} \dots x_i^{m-1} x_{i+1}^m \dots x_{l_m}^m,$$

где x_{i+1}^m – вершина, не совпадающая ни с одной вершиной x_{i+1}^j минимальных путей P^j , $j < m$, которые совпадают с P_i^m до вершины x_i^{m-1} . Подпуть $x_0^m \dots x_i^m = x_0^{m-1} \dots x_i^{m-1}$ пути P_i^m назовем его *корнем* и обозначим через R_i^m , а вторую часть пути $x_{i+1}^m \dots x_m^m$ назовем *ответвлением* и обозначим через S_i^m .

Алгоритм основан на следующих соображениях. Очередной m -й минимальный путь будет отклоняться в какой-либо вершине от одного из предыдущих минимальных путей. Поэтому для его нахождения надо перебрать все такие минимальные отклонения. Будем перебирать отклонения от последнего минимального пути P^{m-1} , для этого поочередно удаляем ребра $(x_i^{m-1}, x_{i+1}^{m-1})$, полагая их веса равными бесконечности. При этом найденный путь должен отличаться также от всех путей P^j , $j < m - 1$, поэтому, если путь P^j совпадает с P^{m-1} до вершины x_i^{m-1} , то вес ребра (x_i^{m-1}, x_{i+1}^j) также полагаем равным бесконечности.

Искомый путь должен быть простым, т.е. ответвление не должно проходить ни через одну вершину корня. Для обеспечения этого условия достаточно удалить ребра, заходящие в вершины корня, т.е. положить их веса равными бесконечности. При этом при рассмотрении очередной вершины она добавляется в корень, поэтому ребра, заходящие в предыдущие вершины корня, были удалены ранее, и достаточно удалить ребра, заходящие в вершину x_i^{m-1} .

Найденные отклонения от последнего минимального пути P^{m-1} записываются в дополнительный список. Путь наименьшего веса из этого списка выбирается в качестве очередного минимального пути и записывается в список минимальных путей. Чтобы избежать повторного нахождения путей из дополнительного списка, список минимальных путей просматривается. Если все минимальные пути совпадают до некоторой вершины, то минимальное отклонение имеет смысл искать, только начиная с этой вершины, поскольку минимальные отклонения от предыдущих вершин были найдены ранее, когда рассматривались предыдущие минимальные пути.

Алгоритм Йена поиска k минимальных простых путей $\langle u, v \rangle$.

Начало. Заводим списки путей $L_0 = \emptyset$ и $L_1 = \emptyset$. Полагаем $i = 0$.

Шаг 1. Находим любым алгоритмом поиска минимального пути P^1 , полагаяем $m = 2$, $i = 0$. Если существует только один минимальный путь P^1 , включаем его в список L_0 и идем на шаг 2. Если таких путей несколько, но меньше, чем k , включаем один из них в список L_0 , а остальные в список L_1 и идем на шаг 2. Если минимальных путей не меньше, чем k , включаем k путей в список L_0 , идем на конец.

Шаг 2. Рассматриваем очередную вершину x_i^{m-1} пути $P^{m-1} = \langle u, v \rangle$. Если $x_i^{m-1} = v$, идем на шаг 7. Иначе полагаяем $c_{x_i^{m-1}, x_{i+1}^{m-1}} = \infty$.

Шаг 3. Если для некоторого пути $P^j : 1 \leq j < m - 1$ верно

$$x_0^j \dots x_i^j = x_0^{m-1} \dots x_i^{m-1},$$

то полагаяем $c_{x_i^{m-1}, x_{i+1}^j} = \infty$.

Шаг 4. Для вершины x_i^{m-1} полагаяем $c_{l, x_i^{m-1}} = \infty : \forall l \leq p$. Находим любым алгоритмом минимальный путь $S_i^m = \langle x_i^{m-1}, v \rangle$.

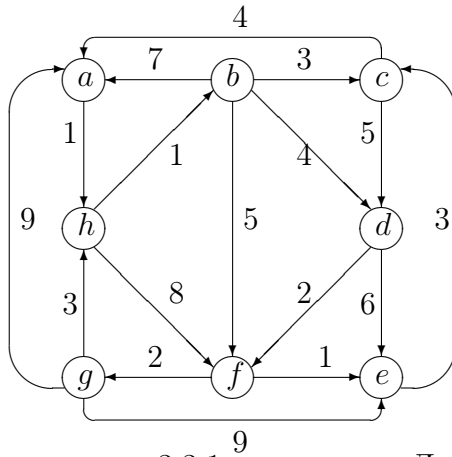
Шаг 5. Находим P_i^m , объединив $R_i^m = x_0^{m-1} \dots x_i^{m-1}$ и S_i^m , и включаем его в список L_1 .

Шаг 6. Полагаем $i = i + 1$ и идем на шаг 2.

Шаг 7. Находим минимальный путь в списке L_1 , обозначаем его через P^m и переносим в список L_0 . Если $m = k$, идем на конец. Иначе полагаем $m = m + 1$. Заменяем элементы матрицы весов, измененные на шагах 2–3, их первоначальными значениями. Если для всех путей $P_j : 1 \leq j < m$ из списка L_0 верно $x_0^j \dots x_l^j = x_0^m \dots x_l^m$, полагаем $i = l$. Для всех вершин $x_j^{m-1} : 0 \leq j < i$ полагаем $c_{l, x_j^{m-1}} = \infty : \forall 1 \leq l \leq p$. Идем на шаг 2.

Конец. Список L_0 содержит первые минимальные простые пути.

Пример. Рассмотрим ориентированный нагруженный граф. Для него в подразделе 3.3.1. были найдены первые минимальные пути алгоритмом Дейкстры. Найдем три первых минимальных пути $\langle a, g \rangle$.



Шаг 1. В подразделе 3.3.1 алгоритмом Дейкстры найден путь $P^1 = ahbfg$ веса 9. Полагаем $m = 2$. Формируем списки путей, в скобках укажем их веса:

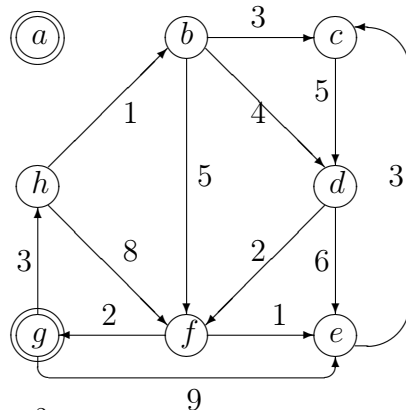
$$L_0 : P^1 = ahbfg(9);$$

$$L_1 : .$$

Шаг 2. Рассматриваем очередную вершину $x_0^1 = a$ пути $P^1 = ahbfg$. Полагаем $c_{a,h} = \infty$ (т.е. удаляем ребро (a, h)).

Шаг 3. Путей $P^j : 1 \leq j < 1$ не существует, матрица весов не меняется.

Шаг 4. Полагаем $\forall x \in V : c_{x,a} = \infty$, т.е. удаляем все ребра, заходящие в вершину a . Итак, граф имеет вид (вершины, между которыми ищется путь, обведены двойной линией). В нем не существует путей $\langle a, g \rangle$.



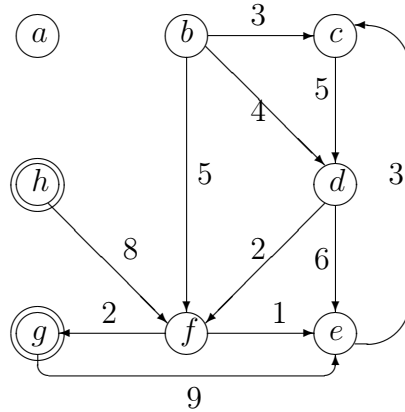
Шаг 5. Путей P_0^2 не существует, список L_1 не меняется.

Шаг 6. Полагаем $i = 1$ и идем на шаг 2.

Шаг 2. Рассматриваем очередную вершину $x_1^1 = h$ пути $P^1 = ahbfg$. Полагаем $c_{h,b} = \infty$ (т.е. удаляем ребро (h, b)).

Шаг 3. Путь $P^j : 1 \leq j < 1$ не существует, матрица весов не меняется.

Шаг 4. Полагаем $\forall x \in V : c_{x,h} = \infty$, т.е. удаляем все ребра, заходящие в вершину h . Итак, граф имеет вид:



Находим минимальный путь $S_2^2 = hfg$.

Шаг 5. Находим $P_2^2 = ahfg$, объединив $R_2^1 = ah$ и $S_2^2 = hfg$, и включаем его в список L_1 , получаем:

$$L_0 : P^1 = ahbfg(9);$$

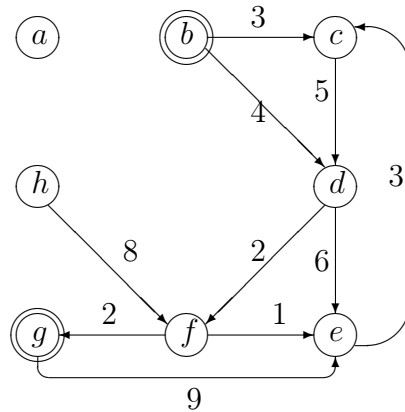
$$L_1 : P_2^2 = ahfg(11).$$

Шаг 6. Полагаем $i = 2$ и идем на шаг 2.

Шаг 2. Рассматриваем очередную вершину $x_2^1 = b$ пути $P^1 = ahbfg$. Полагаем $c_{b,f} = \infty$ (т.е. удаляем ребро (b, f)).

Шаг 3. Путь $P^j : 1 \leq j < 1$ не существует, матрица весов не меняется.

Шаг 4. Полагаем $\forall x \in V : c_{x,b} = \infty$, т.е. удаляем все ребра, заходящие в вершину b . Итак, граф имеет вид:



Находим минимальный путь $S_3^2 = bdfg$.

Шаг 5. Находим $P_3^2 = ahbdfg$, объединив $R_3^1 = ahb$ и $S_3^2 = bdfg$, и включаем его в список L_1 , получаем:

$$L_0 : P^1 = ahbfg(9);$$

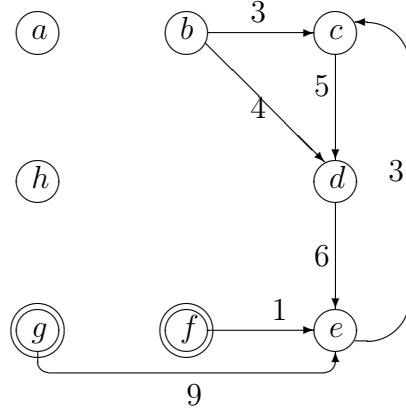
$$L_1 : P_2^2 = ahfg(11), \quad P_3^2 = ahbdfg(10).$$

Шаг 6. Полагаем $i = 3$ и идем на шаг 2.

Шаг 2. Рассматриваем очередную вершину $x_3^1 = f$ пути $P^1 = ahbfg$. Полагаем $c_{f,g} = \infty$ (т.е. удаляем ребро (f, g)).

Шаг 3. Путей $P^j : 1 \leq j < 1$ не существует, матрица весов не меняется.

Шаг 4. Полагаем $\forall x \in V : c_{x,f} = \infty$, т.е. удаляем все ребра, заходящие в вершину f . Итак, граф имеет вид:



Пути $\langle f, g \rangle$ не существует.

Шаг 5. Пути P_4^2 , построить не удастся, список L_1 не меняется.

Шаг 6. Заменяем элементы матрицы весов, измененные на шагах 2–3, их первоначальными значениями, полагаем $i = 4$ и идем на шаг 2.

Шаг 2. Поскольку очередная вершина пути $P^1 = ahbfg$ является конечной, т.е. $x_4^1 = g$, идем на шаг 7.

Шаг 7. Рассматриваем списки путей

$$\begin{aligned} L_0 : P^1 &= ahbfg(9); \\ L_1 : P_2^2 &= ahfg(11), \quad P_3^2 = ahbdfg(10). \end{aligned}$$

Путь $P_3^2 = ahbdfg(10)$ является минимальным в списке L_1 , обозначаем его через P^2 и переносим в список L_0 :

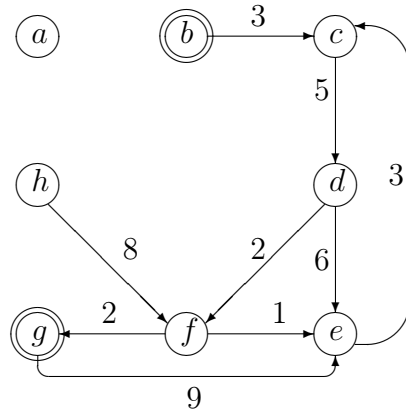
$$\begin{aligned} L_0 : P^1 &= ahbfg(9), \quad P^2 = ahbdfg(10); \\ L_1 : P_2^2 &= ahfg(11). \end{aligned}$$

Номер последнего минимального пути $2 < k$ ($k = 3$), полагаем $m = 3$. Заменяем элементы матрицы весов, измененные на шагах 2–3, их первоначальными значениями. Поскольку $x_0^1 x_1^1 x_2^1 = x_0^2 x_1^2 x_2^2 = abh$, полагаем $i = 2$, $c_{x,a} = c_{x,h} = 0$, и идем на шаг 2.

Шаг 2. Рассматриваем очередную вершину $x_2^2 = b$ пути $P^2 = ahbdfg$. Полагаем $c_{b,d} = \infty$ (т.е. удаляем ребро (b, d)).

Шаг 3. Путь $P^1 = ahbfg$ совпадает с путем $P^2 = ahbdfg$ до вершины b , значит, полагаем $c_{b,f} = \infty$ (т.е. удаляем ребро (b, f)).

Шаг 4. Полагаем $\forall x \in V : c_{x,b} = \infty$, т.е. удаляем все ребра, заходящие в вершину b . Диаграмма графа приведена на следующей странице. Находим минимальный путь $S_2^3 = bcdfg$.



Шаг 5. Находим $P_2^3 = ahbcdfg$, объединив $R_2^2 = ahb$ и $S_2^3 = bcdfg$, и включаем его в список L_1 , получаем:

$$L_0 : P^1 = ahbfg(9), \quad P^2 = ahbdfg(10);$$

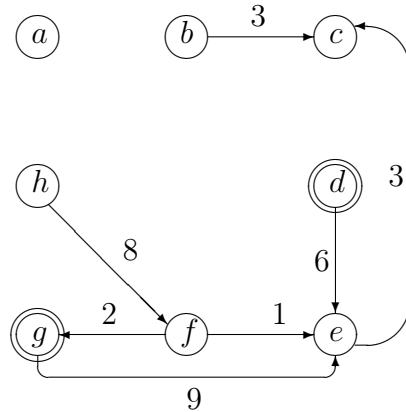
$$L_1 : P_2^2 = ahfg(11), \quad P_2^3 = ahbcdfg(14).$$

Шаг 6. Полагаем $i = 3$ и идем на шаг 2.

Шаг 2. Рассматриваем очередную вершину $x_3^2 = d$ пути $P^2 = ahbdfg$. Полагаем $c_{d,f} = \infty$ (т.е. удаляем ребро (d, f)).

Шаг 3. Путь $P^1 = ahbfg$ не совпадает с путем $P^2 = ahbdfg$ до вершины d , матрица весов не меняется.

Шаг 4. Полагаем $\forall x \in V : c_{x,d} = \infty$, т.е. удаляем все ребра, заходящие в вершину d . Итак, граф имеет вид:



Пути $\langle d, g \rangle$ не существует.

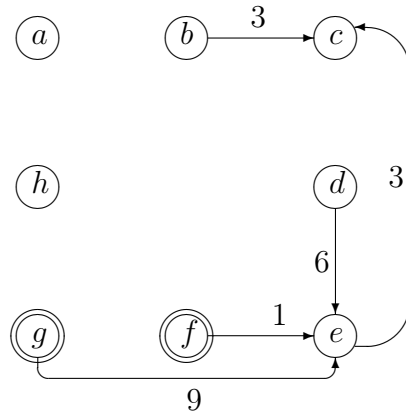
Шаг 5. Путь P_3^3 построить нельзя, список L_1 не меняется.

Шаг 6. Полагаем $i = 4$ и идем на шаг 2.

Шаг 2. Рассматриваем очередную вершину $x_4^2 = f$ пути $P^2 = ahbdfg$. Полагаем $c_{f,g} = \infty$ (т.е. удаляем ребро (f, g)).

Шаг 3. Путь $P^1 = ahbfg$ не совпадает с путем $P^2 = ahbdfg$ до вершины f , матрица весов не меняется.

Шаг 4. Полагаем $\forall x \in V : c_{x,f} = \infty$, т.е. удаляем все ребра, заходящие в вершину f . Диаграмма графа приведена на следующей странице. Пути $\langle f, g \rangle$ не существует.



Шаг 5. Путь P_4^3 построить нельзя, список L_1 не меняется.

Шаг 6. Полагаем $i = 5$ и идем на шаг 2.

Шаг 2. Поскольку очередная вершина пути $P^2 = ahbdfg$ является конечной, т.е. $x_5^2 = g$, идем на шаг 7.

Шаг 7. Рассматриваем списки путей

$$\begin{aligned} L_0 : P^1 &= ahbfg(9), & P^2 &= ahbdfg(10); \\ L_1 : P_2^2 &= ahfg(11), & P_2^3 &= ahbcdfg(14). \end{aligned}$$

Путь $P_2^2 = ahfg(11)$ является минимальным в списке L_1 , обозначаем его через P^3 и переносим в список L_0 :

$$\begin{aligned} L_0 : P^1 &= ahbfg(9), & P^2 &= ahbdfg(10), & P^3 &= ahfg(11); \\ L_1 : P_2^3 &= ahbcdfg(14). \end{aligned}$$

Номер последнего минимального пути $m = 3 = k$, список L_0 содержит первые три минимальных пути.

$$L_0 : P^1 = ahbfg(9), \quad P^2 = ahbdfg(10), \quad P^3 = ahfg(11).$$

3.5 Поиск путей с заданными свойствами

Рассмотрим задачу перечисления путей, обладающих некоторым свойством.

Определение. Композицией путей $P = \langle u, v \rangle = ux_1 \dots x_nv$ и $S = \langle v, w \rangle = vy_1 \dots y_mw$ назовем путь $PS = \langle u, w \rangle = ux_1 \dots x_nv y_1 \dots y_mw$.

Определение. Свойство пути называется *латинским*, если из того, что путь PS обладает этим свойством, следует, что пути P и S также им обладают.

Примеры. Следующие свойства являются латинскими:

- не проходить через заданное множество вершин;
- не проходить через заданное множество ребер;
- быть простой цепью;
- быть цепью;
- не проходить через заданную вершину более k раз.

Примеры. Следующие свойства не являются латинскими:

- проходить через заданное множество вершин;

- проходить через заданное множество ребер;
- быть простым циклом;
- быть циклом;
- проходить через заданную вершину более k раз.

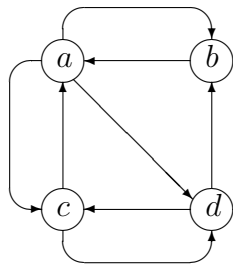
Опишем матричный способ перечисления путей, обладающих латинским свойством, в графе (орграфе), называемый *метод латинской композиции*.

Пусть α – латинское свойство.

Определение. Назовем α -композицией путей путь, определяемый следующим образом:

$$P\alpha S = \begin{cases} PS, & \text{если } PS \text{ является } \alpha\text{-путем;} \\ \emptyset, & \text{иначе.} \end{cases}$$

Пример. Дан орграф.

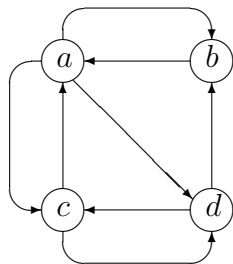


Рассмотрим свойство α «быть простым путем». Пусть $P = adc$, $S = cab$, композиция путей $PS = adcab$. Этот путь не является простым, т.е. не обладает свойством α , следовательно, $P\alpha S = \emptyset$. Пусть $P = dba$, $S = ac$, композиция путей $PS = dbac$. Этот путь является простым, т.е. обладает свойством α , следовательно, $P\alpha S = dbac$.

Определение. Путь, обладающий свойством α , назовем α -путем.

Определение. Латинской матрицей L_α^k назовем матрицу размерности $p \times p$, строкам которой сопоставлены вершины графа, а элементами являются множества α -путей длины k между соответствующими вершинами.

Пример. Рассмотрим орграф из предыдущего примера и его латинскую матрицу простых путей длины 2.



$$L_\alpha^2 = \begin{array}{c|cccc} & a & b & c & d \\ \hline a & \emptyset & \{adb\} & \{adc\} & \{acd\} \\ b & \emptyset & \emptyset & \{bac\} & \{bad\} \\ c & \emptyset & \{cab, cdb\} & \emptyset & \{cad\} \\ d & \{dca, dba\} & \emptyset & \emptyset & \emptyset \end{array}$$

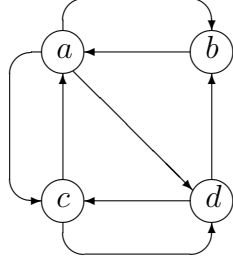
Пусть L , N – латинские матрицы для одного и того же графа.

Определение. Композицией латинских матриц назовем матрицу, элементы которой вычисляются следующим образом

$$\langle L\alpha N \rangle_{i,j} = \bigcup_{m=1}^p \bigcup_{P \in L_{i,m}} \bigcup_{S \in N_{m,j}} P\alpha S.$$

Операция вычисления композиции латинских матриц по своей сути аналогична операции умножения обычных числовых матриц, вместо суммы здесь используется операция объединения, а вместо произведения – операция нахождения α -композиции путей.

Пример. Рассмотрим орграф из предыдущего примера и его латинские матрицы простых путей L_α^1 и L_α^2 .



$$L_\alpha^1 = \begin{array}{c|cccc} & a & b & c & d \\ \hline a & \emptyset & \{ab\} & \{ac\} & \{ad\} \\ b & \{ba\} & \emptyset & \emptyset & \emptyset \\ c & \{ca\} & \emptyset & \emptyset & \{cd\} \\ d & \emptyset & \{db\} & \{dc\} & \emptyset \end{array}$$

$$L_\alpha^2 = \begin{array}{c|cccc} & a & b & c & d \\ \hline a & \emptyset & \{adb\} & \{adc\} & \{acd\} \\ b & \emptyset & \emptyset & \{bac\} & \{bad\} \\ c & \emptyset & \{cab, cdb\} & \emptyset & \{cad\} \\ d & \{dca, dba\} & \emptyset & \emptyset & \emptyset \end{array}$$

Найдем элемент $\langle L_\alpha^1 \alpha L_\alpha^2 \rangle_{c,a}$. Для этого понадобится строка c матрицы L_α^1 и столбец a матрицы L_α^2 . Перебираем все вершины $x \in V$, и для всех путей $P \in \langle L_\alpha^1 \rangle_{c,x} \neq \emptyset$, $S \in \langle L_\alpha^2 \rangle_{x,a} \neq \emptyset$ находим α -композицию $P \alpha S$, где свойство α означает «быть простым путем»:

$$\begin{aligned} \langle L_\alpha^1 \rangle_{c,a} &= \{ca\}, & \langle L_\alpha^2 \rangle_{a,a} &= \emptyset; \\ \langle L_\alpha^1 \rangle_{c,b} &= \emptyset, & \langle L_\alpha^2 \rangle_{b,a} &= \emptyset; \\ \langle L_\alpha^1 \rangle_{c,c} &= \emptyset, & \langle L_\alpha^2 \rangle_{c,a} &= \emptyset; \\ \langle L_\alpha^1 \rangle_{c,d} &= \{cd\}, & \langle L_\alpha^2 \rangle_{d,a} &= \{dca, dba\}, & cd \alpha dca &= \emptyset, & cd \alpha dba &= cdba. \end{aligned}$$

Объединение всех полученных α -композиций дает

$$\langle L_\alpha^1 \alpha L_\alpha^2 \rangle_{c,a} = \{cdba\}.$$

Утверждение. Для любых k, m верно

$$L_\alpha^{k+m} = L_\alpha^k \alpha L_\alpha^m.$$

Доказательство. Пусть путь $P \in \langle L_\alpha^{k+m} \rangle_{i,j}$, тогда P – путь, соединяющий вершины i, j и обладающий латинским свойством α . Тогда для любых S, T , таких что $P = ST$, пути S и T обладают свойством α . Выберем пути из условия $|S| = k$ и $|T| = m$, тогда

$$\exists l : S \in \langle L_\alpha^k \rangle_{i,l}, \quad T \in \langle L_\alpha^m \rangle_{l,j}.$$

Отсюда следует $P = ST \in \langle L_\alpha^k \alpha L_\alpha^m \rangle_{i,j}$.

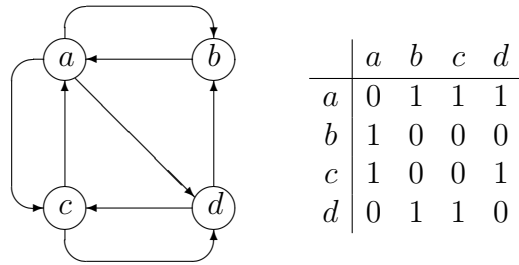
Пусть $P \in \langle L_\alpha^k \alpha L_\alpha^m \rangle_{i,j}$, тогда P обладает латинским свойством и

$$\exists l : S \in \langle L_\alpha^k \rangle_{i,l}, \quad T \in \langle L_\alpha^m \rangle_{l,j}.$$

Отсюда $P \in \langle L_\alpha^{k+m} \rangle_{i,j}$.

Из утверждения следует *метод латинской композиции* перечисления α -путей длины k . Он состоит в построении латинских матриц L_α^k для некоторых k .

Пример. Рассмотрим орграф из предыдущих примеров и его матрицу смежности.



Найдем пути длины 3, не содержащие более одного простого цикла. Очевидно, любой путь, состоящий из одного ребра, обладает таким свойством, поэтому

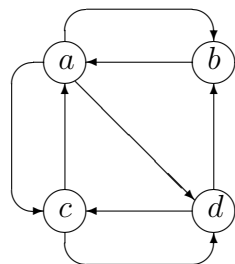
$$L_\alpha^1 = \begin{array}{c|cccc} & a & b & c & d \\ \hline a & \emptyset & \{ab\} & \{ac\} & \{ad\} \\ b & \{ba\} & \emptyset & \emptyset & \emptyset \\ c & \{ca\} & \emptyset & \emptyset & \{cd\} \\ d & \emptyset & \{db\} & \{dc\} & \emptyset \end{array}$$

Воспользовавшись утверждением, подсчитаем $L_\alpha^2 = L_\alpha^1 \alpha L_\alpha^1$:

$$L_\alpha^2 = \begin{array}{c|cccc} & a & b & c & d \\ \hline a & \emptyset & \{ab\} & \{ac\} & \{ad\} \\ b & \{ba\} & \emptyset & \emptyset & \emptyset \\ c & \{ca\} & \emptyset & \emptyset & \{cd\} \\ d & \emptyset & \{db\} & \{dc\} & \emptyset \end{array} \alpha \begin{array}{c|cccc} & a & b & c & d \\ \hline a & \emptyset & \{ab\} & \{ac\} & \{ad\} \\ b & \{ba\} & \emptyset & \emptyset & \emptyset \\ c & \{ca\} & \emptyset & \emptyset & \{cd\} \\ d & \emptyset & \{db\} & \{dc\} & \emptyset \end{array} =$$

$$= \begin{array}{c|cccc} & a & b & c & d \\ \hline a & \{aba, aca\} & \{adb\} & \{adc\} & \{acd\} \\ b & \emptyset & \{bab\} & \{bac\} & \{bad\} \\ c & \emptyset & \{cab, cdb\} & \{cac, cdc\} & \{cad\} \\ d & \{dba, dca\} & \emptyset & \emptyset & \{dcd\} \end{array}$$

Правильность вычислений можно проверить визуально по диаграмме графа.



Действительно, например, существует два пути $\langle a, a \rangle$ длины 2, а именно, aba и aca . Существует один путь $\langle a, b \rangle$ длины 2, а именно, adb . Не существует путей $\langle b, a \rangle$ длины 2. Ясно, что пути длины 2 не могут содержать более одного простого цикла, если в графе нет петель, поэтому в матрицу L_α^2 просто записываются все найденные пути.

Воспользовавшись утверждением, подсчитаем $L_\alpha^3 = L_\alpha^1 \alpha L_\alpha^2$:

$$\begin{array}{c}
 L_\alpha^3 = \begin{array}{c|cccc} & a & b & c & d \\ \hline a & \emptyset & \{ab\} & \{ac\} & \{ad\} \\ b & \{ba\} & \emptyset & \emptyset & \emptyset \\ c & \{ca\} & \emptyset & \emptyset & \{cd\} \\ d & \emptyset & \{db\} & \{dc\} & \emptyset \end{array} \alpha \\
 \\
 \alpha = \begin{array}{c|cccc} & a & b & c & d \\ \hline a & \{aba, aca\} & \{adb\} & \{adc\} & \{acd\} \\ b & \emptyset & \{bab\} & \{bac\} & \{bad\} \\ c & \emptyset & \{cab, cdb\} & \{cac, cdc\} & \{cad\} \\ d & \{dba, dca\} & \emptyset & \emptyset & \{dcd\} \end{array} = \\
 \\
 = \begin{array}{c|cccc} & a & b & c & d \\ \hline a & \{adba, adca\} & \{acab, acdb\} & \{abac, acdc\} & \{abad, acad, adcd\} \\ b & \{baca\} & \{badb\} & \{badc\} & \{bacd\} \\ c & \{caba, cdba, cdca\} & \{cabd\} & \{cadc\} & \{cacd\} \\ d & \emptyset & \{dbab, dcab, dcdb\} & \{dbac, dcac\} & \{dbad, dcad\} \end{array}
 \end{array}$$

Правильность вычислений можно проверить визуально по диаграмме графа. Обратим внимание, что пути $abab = (ab) \alpha (bab)$, $caca = (ca) \alpha (aca)$ и т.д. не были добавлены в матрицу L_α^3 , как содержащие более одного простого цикла, т.е. не обладающие рассматриваемым латинским свойством.

Очевидно, с ростом длины пути может резко возрасти общее количество путей, поэтому данный способ практически можно применить только для матриц относительно небольших размерностей и для путей небольшой длины.

3.6 Упражнения

1. Пусть орграфы заданы матрицами весов ребер (пустые клетки соответствуют отсутствию ребер):

	<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	<i>e</i>	<i>f</i>	<i>g</i>	<i>h</i>
<i>a</i>					3			1
<i>b</i>			4					
<i>c</i>							5	
<i>d</i>			6			1		
<i>e</i>		3	9					
<i>f</i>								
<i>g</i>		2		5	8			1
<i>h</i>						7		

	<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	<i>e</i>	<i>f</i>	<i>g</i>	<i>h</i>
<i>a</i>								3
<i>b</i>	2							
<i>c</i>								
<i>d</i>			1			4		
<i>e</i>		7	8					
<i>f</i>	2	4						
<i>g</i>	5				6	2		1
<i>h</i>		9		2		5		

Решить следующие задачи:

- найти волновым алгоритмом кратчайшие пути от вершины a до остальных вершин графа;
- найти алгоритмом Дейкстры минимальные пути от вершины a до остальных вершин графа.

2. Пусть орграфы заданы матрицами весов ребер (если клетка пустая, соответствующего ребра не существует):

	<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	<i>e</i>
<i>a</i>		-5	6		-3
<i>b</i>			1		4
<i>c</i>	2	4		-1	
<i>d</i>					5
<i>e</i>	5	-3			

	<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	<i>e</i>
<i>a</i>			-4		3
<i>b</i>			1		-6
<i>c</i>		2		-3	6
<i>d</i>					5
<i>e</i>	1	7	-2		

Решить следующие задачи:

– с помощью алгоритма Форда найти минимальные пути от вершины *a* до остальных вершин или показать, что задача не имеет решения;

– с помощью алгоритма Белмана–Мура найти минимальные пути от вершины *e* до остальных вершин или показать, что задача не имеет решения;

– с помощью алгоритма Флойда найти минимальные пути между всеми парами вершин, которые могут проходить только через вершины *a*, *c*, *d*, или показать, что задача не имеет решения;

– с помощью алгоритма Данцига найти минимальные пути между всеми парами вершин, которые не могут проходить через вершину *b*, или показать, что задача не имеет решения.

3. Получить как можно более точные оценки вычислительной сложности алгоритмов Дейкстры, Флойда и Данцига. Оценить, какой из алгоритмов выгоднее использовать для поиска минимальных путей между всеми парами вершин в случае неотрицательных весов ребер.

4. Предложить алгоритм поиска пути, имеющего наибольший минимальный из весов ребер. Обосновать алгоритм и получить оценку его вычислительной сложности.

5. Предложить алгоритм поиска самого длинного простого пути между заданными вершинами. Обосновать алгоритм и получить оценку его вычислительной сложности.

6. Пусть мультиорграф задан матрицей весов ребер:

	<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>
<i>a</i>	[0, ∞, ∞]	[-1, ∞, ∞]	[2, 3, ∞]	[∞, ∞, ∞]
<i>b</i>	[2, ∞, ∞]	[0, ∞, ∞]	[4, 5, ∞]	[∞, ∞, ∞]
<i>c</i>	[∞, ∞, ∞]	[-2, 3, ∞]	[0, ∞, ∞]	[-3, 1, 5]
<i>d</i>	[3, 7, ∞]	[1, 2, ∞]	[6, ∞, ∞]	[0, ∞, ∞]

Решить следующие задачи:

– выполнить одну итерацию алгоритма двойного поиска, применив его для нахождения трех первых минимальных путей от вершины *a* до остальных вершин графа;

– с помощью обобщенного алгоритма Флойда найти три первых минимальных пути между всеми парами вершин, которые могут проходить только через вершины *a*, *c* или показать, что задача не имеет решения;

– с помощью обобщенного алгоритма Данцига найти три первых минимальных пути между всеми парами вершин, которые не могут проходить через вершину *b*, или показать, что задача не имеет решения.

7. Получить как можно более точную оценку вычислительной сложности алгоритма двойного поиска. Модифицировать алгоритм так, чтобы он обнаруживал циклы отрицательного веса.

8. Получить как можно более точные оценки вычислительной сложности обобщенных вариантов алгоритмов Флойда и Данцига.

9. Для орграфов из упражнения 1 найти три первых минимальных пути от вершины a до вершин b, e, f .

10. Предложить алгоритм поиска k минимальных реберно-непересекающихся путей. Обосновать алгоритм и получить оценку его вычислительной сложности.

11. Пусть орграфы заданы матрицами смежности:

	a	b	c	d	e
a		1	1		1
b			1		
c	1	1		1	
d					1
e	1	1			

	a	b	c	d	e
a			1		1
b				1	1
c		1		1	
d					1
e	1		1		

Перечислить пути, обладающие следующими латинскими свойствами:

- пути длины 5, не проходящие через одну вершину более двух раз;
- пути длины 3, не проходящие через вершину d ;
- простые пути длины 3;
- пути длины 5, содержащие не более одного простого цикла.

4 Задачи размещения

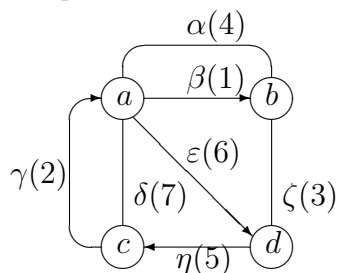
Задачи размещения связаны с решением проблем наилучшего расположения центров обслуживания, например, станций скорой помощи, складов, и т.д. Мы рассмотрим задачи, где область размещения может быть представлена в виде графа, т.е. объекты могут располагаться на ребрах и в вершинах графа.

4.1 Расстояния во взвешенном графе

Определение. Расстоянием вершина-вершина между вершинами i и j (обозначение $d(i, j)$) называется вес минимального пути $\langle i, j \rangle$.

Для вычисления расстояний вершина-вершина могут быть использованы алгоритмы Флойда и Данцига.

Пример. Рассмотрим смешанный граф. Вершины обозначим латинскими буквами, ребра – греческими, в скобках укажем веса ребер.

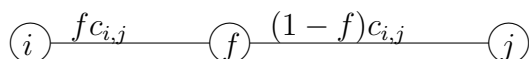


Выпишем последовательность матриц весов путей, построенных с помощью алгоритма Данцига.

$$\Lambda^{(1)} = \begin{array}{c|c} & a \\ \hline a & 0 \end{array}, \quad \Lambda^{(2)} = \begin{array}{c|cc} & a & b \\ \hline a & 0 & 1 \\ b & 4 & 0 \end{array}, \quad \Lambda^{(3)} = \begin{array}{c|ccc} & a & b & c \\ \hline a & 0 & 1 & 7 \\ b & 4 & 0 & 11 \\ c & 2 & 3 & 0 \end{array}, \quad \Lambda^{(4)} = \begin{array}{c|cccc} & a & b & c & d \\ \hline a & 0 & 1 & 7 & 4 \\ b & 4 & 0 & 8 & 3 \\ c & 2 & 3 & 0 & 6 \\ d & 7 & 3 & 5 & 0 \end{array}.$$

Рассмотрим ребро (i, j) , вес которого $c_{i,j} \geq 0$.

Определение. Пусть параметр $f : 0 \leq f \leq 1$. Точку f на ребре, которая делит ребро в пропорции $f : (1 - f)$, назовем f -точкой (обозначение $f_{(i,j)}$).



Вес части ребра (i, f) равен $f c_{i,j}$, вес части ребра (f, j) равен $(1 - f) c_{i,j}$. Очевидно, что 0-точка – это вершина i , 1-точка – вершина j .

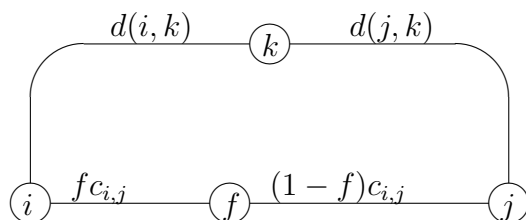
Определение. Точки ребер, не являющиеся вершинами, назовем *внутренними точками*.

Определение. Расстоянием точка-вершина (обозначение $d(f_{(i,j)}, k)$) называется вес минимального пути между f -точкой ребра (i, j) и вершиной k .

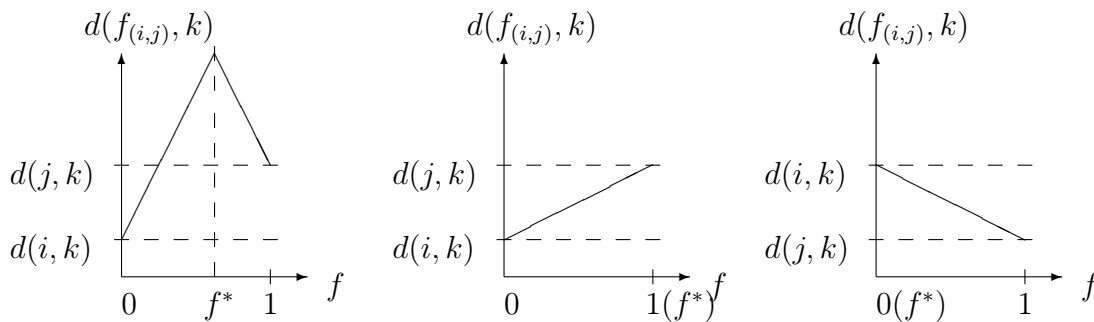
Рассмотрим вычисление расстояний точка-вершина. Если ребро (i, j) неориентированное, то его обход допустим в обоих направлениях, и $d(f_{(i,j)}, k)$ вычисляется как минимум из двух величин

$$(1) \quad d(f_{(i,j)}, k) = \min \{ f c_{i,j} + d(i, k), (1 - f) c_{i,j} + d(j, k) \}$$

В первом случае сначала идем от точки f по ребру (i, j) до вершины i , а затем по пути минимального веса от вершины i до вершины k . Во втором случае сначала идем от точки f по ребру (i, j) до вершины j , а затем по пути минимального веса от вершины j до вершины k .



Зависимость расстояния точка-вершина от параметра f может быть одного из трех типов.



Точка f^* , т.е. точка пересечения прямых $fc_{i,j} + d(i, k)$ и $(1-f)c_{i,j} + d(j, k)$, является точкой максимума функции $d(f_{(i,j)}, k)$. Эта точка имеет вид

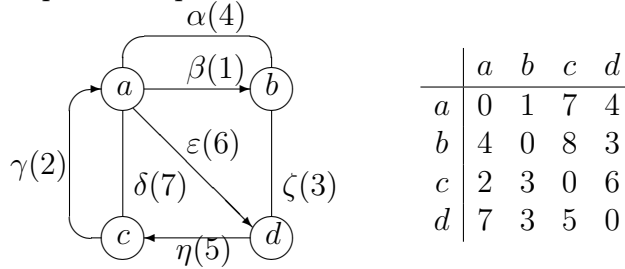
$$(2) \quad f^* = \frac{d(j, k) - d(i, k) + c_{i,j}}{2c_{i,j}}.$$

Поскольку для расстояний в графе верны соотношения

$$\begin{aligned} d(i, k) &\leq c_{i,j} + d(j, k); \\ d(j, k) &\leq c_{i,j} + d(i, k), \end{aligned}$$

то f^* принадлежит интервалу $[0, 1]$.

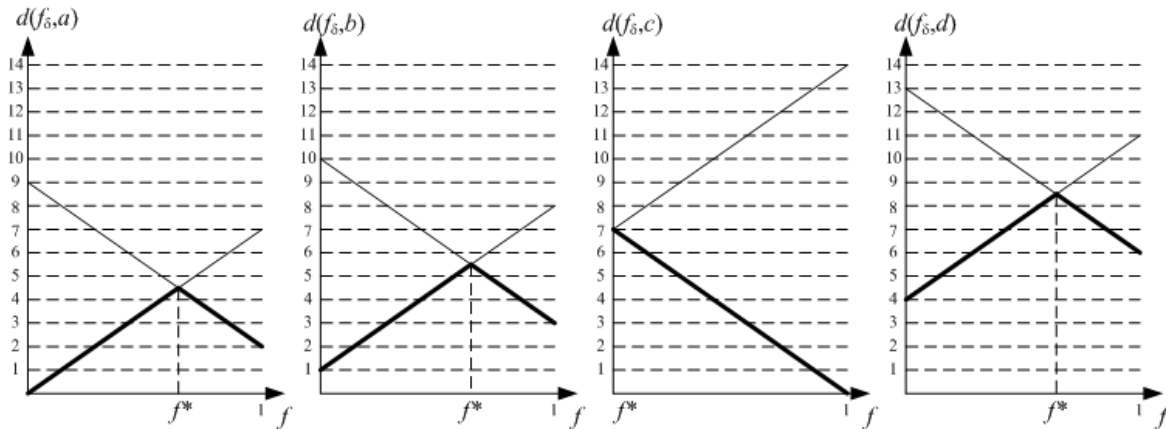
Пример. Рассмотрим смешанный граф из предыдущего примера и его матрицу расстояний вершина-вершина.



Рассмотрим неориентированное ребро $\delta = (a, c)$. Найдем расстояние от f -точки этого ребра до всех вершин графа. Для веса ребра δ будем использовать обозначение c_δ . По формуле (1) имеем

$$\begin{aligned} d(f_\delta, a) &= \min \{fc_\delta + d(a, a), (1-f)c_\delta + d(c, a)\} = \min \{7f + 0, 7(1-f) + 2\}; \\ d(f_\delta, b) &= \min \{fc_\delta + d(a, b), (1-f)c_\delta + d(c, b)\} = \min \{7f + 1, 7(1-f) + 3\}; \\ d(f_\delta, c) &= \min \{fc_\delta + d(a, c), (1-f)c_\delta + d(c, c)\} = \min \{7f + 7, 7(1-f) + 0\}; \\ d(f_\delta, d) &= \min \{fc_\delta + d(a, d), (1-f)c_\delta + d(c, d)\} = \min \{7f + 4, 7(1-f) + 6\}. \end{aligned}$$

Построим графики этих функций. Функция расстояния точка-вершина выделена жирной линией.



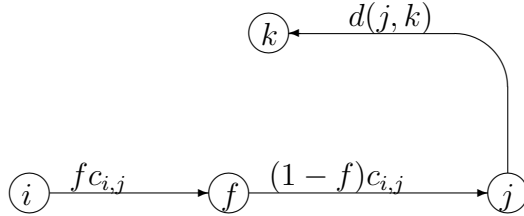
Из условия пересечения прямых найдем точку максимума расстояния f^* :

$$\begin{aligned} a: \quad 7f &= 9 - 7f, & f^* &= 9/14; \\ b: \quad 7f + 1 &= 10 - 7f, & f^* &= 9/14; \\ c: \quad 7f + 7 &= 7 - 7f, & f^* &= 0; \\ d: \quad 7f + 4 &= 13 - 7f, & f^* &= 9/14. \end{aligned}$$

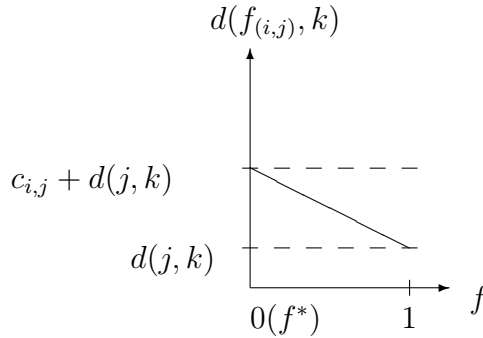
Если ребро (i, j) ориентированное, то его обход допустим только в направлении (i, j) , и $d(f_{(i,j)}, k)$ принимает вид

$$(3) \quad d(f_{(i,j)}, k) = (1 - f)c_{i,j} + d(j, k).$$

Сначала осуществляется перемещение по ребру (i, j) до вершины j , а потом – перемещение в вершину k по кратчайшему пути $\langle j, k \rangle$.

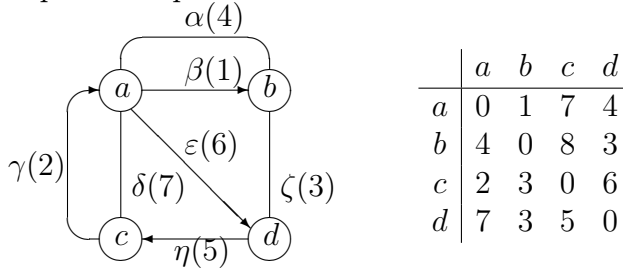


Зависимость расстояния точка-вершина от параметра f в этом случае имеет следующий вид



Точка $f^* = 0$ является точкой максимума функции $d(f_{(i,j)}, k)$.

Пример. Рассмотрим смешанный граф из предыдущего примера и его матрицу расстояний вершина-вершина.

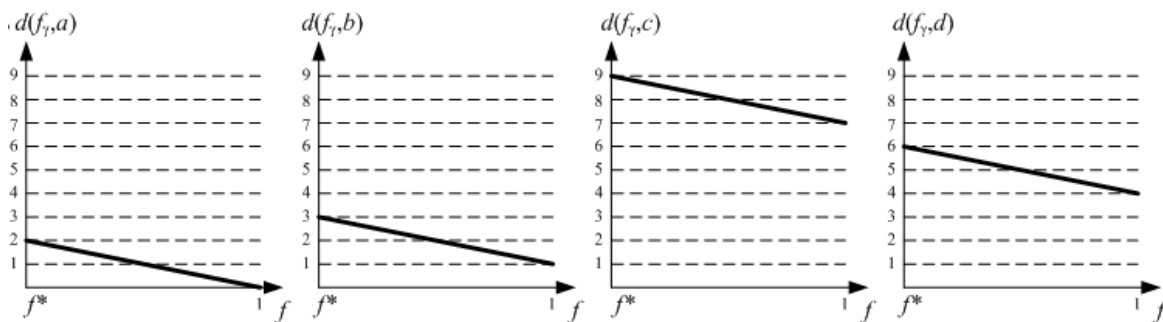


	a	b	c	d
a	0	1	7	4
b	4	0	8	3
c	2	3	0	6
d	7	3	5	0

Рассмотрим ориентированное ребро $\gamma = (c, a)$. Найдем расстояние от f -точки этого ребра до всех вершин графа. По формуле (3) имеем

$$\begin{aligned} d(f_\gamma, a) &= (1 - f)c_\gamma + d(a, a) = 2(1 - f) + 0 = 2 - 2f; \\ d(f_\gamma, b) &= (1 - f)c_\gamma + d(a, b) = 2(1 - f) + 1 = 3 - 2f; \\ d(f_\gamma, c) &= (1 - f)c_\gamma + d(a, c) = 2(1 - f) + 7 = 9 - 2f; \\ d(f_\gamma, d) &= (1 - f)c_\gamma + d(a, d) = 2(1 - f) + 4 = 6 - 2f; \end{aligned}$$

Графики этих функций приведены на следующей странице.



Аналогично можно рассмотреть расстояние от вершины до f -точки ребра.

Определение. Расстоянием вершина-точка (обозначение $d(k, f_{(i,j)})$) называется вес минимального пути между вершиной k и f -точкой ребра (i, j) .

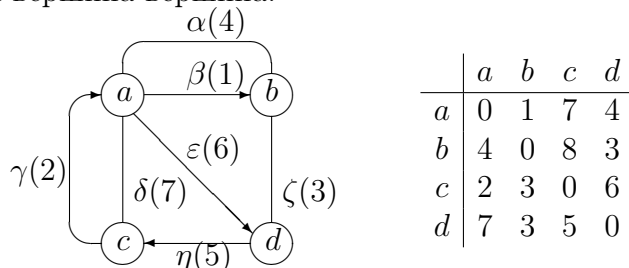
Для неориентированного ребра (i, j) эта величина принимает вид

$$(4) \quad d(k, f_{(i,j)}) = \min \{d(k, i) + fc_{i,j}, d(k, j) + (1-f)c_{i,j}\}$$

Для ориентированного ребра (i, j) имеем

$$(5) \quad d(k, f_{(i,j)}) = d(k, i) + fc_{i,j}.$$

Пример. Рассмотрим смешанный граф из предыдущего примера и его матрицу расстояний вершина-вершина.



Найдем расстояние от вершины a до f -точек всех ребер. Рассмотрим сначала неориентированные ребра.

$$\begin{aligned} \alpha = (a, b) : \quad d(a, f_\alpha) &= \min \{d(a, a) + fc_\alpha, d(a, b) + (1-f)c_\alpha\} = \\ &= \min \{0 + 4f, 1 + 4(1-f)\}; \\ \delta = (a, c) : \quad d(a, f_\delta) &= \min \{d(a, a) + fc_\delta, d(a, c) + (1-f)c_\delta\} = \\ &= \min \{0 + 7f, 7 + 7(1-f)\}; \\ \zeta = (b, d) : \quad d(a, f_\zeta) &= \min \{d(a, b) + fc_\zeta, d(a, d) + (1-f)c_\zeta\} = \\ &= \min \{1 + 3f, 1 + 3(1-f)\}. \end{aligned}$$

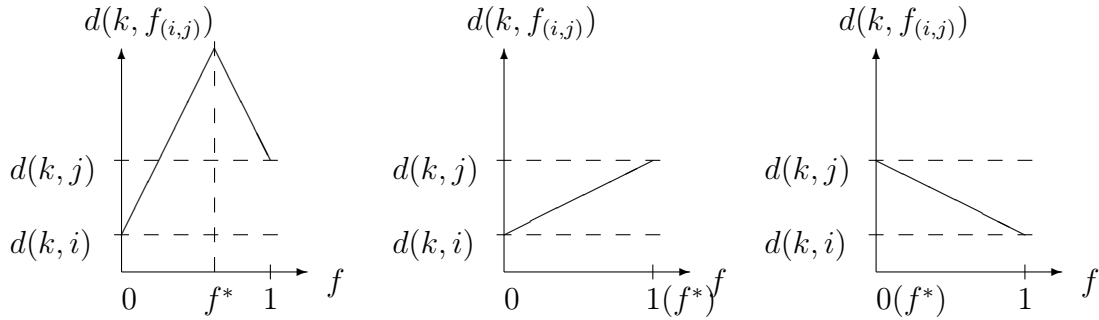
Рассмотрим далее ориентированные ребра.

$$\begin{aligned} \beta = (a, b) : \quad d(a, f_\beta) &= d(a, a) + fc_\beta = 0 + f; \\ \gamma = (c, a) : \quad d(a, f_\gamma) &= d(a, c) + fc_\gamma = 7 + 2f; \\ \epsilon = (a, d) : \quad d(a, f_\epsilon) &= d(a, a) + fc_\epsilon = 0 + 6f; \\ \eta = (d, c) : \quad d(a, f_\eta) &= d(a, d) + fc_\eta = 4 + 5f; \end{aligned}$$

Определение. Расстоянием вершина-ребро (обозначение $d(k, (i, j))$) называется максимальное из расстояний вершина-точка для точек этого ребра, т.е.

$$(6) \quad d(k, (i, j)) = \max_{f \in [0,1]} d(k, f_{(i,j)}).$$

Если ребро (i, j) неориентированное, расстояние вершина-точка представляется в одной из трех форм



Максимум расстояния вершина-точка достигается в точке пересечения прямых $d(k, i) + fc_{i,j}$ и $d(k, j) + (1 - f)c_{i,j}$. Эта точка f^* находится из условия

$$d(k, i) + fc_{i,j} = d(k, j) + (1 - f)c_{i,j},$$

откуда

$$f^* = \frac{d(k, j) - d(k, i) + c_{i,j}}{2c_{i,j}}.$$

Поскольку для расстояний в графе верны соотношения

$$\begin{aligned} d(i, k) &\leq c_{i,j} + d(j, k); \\ d(j, k) &\leq c_{i,j} + d(i, k), \end{aligned}$$

то f^* принадлежит интервалу $[0, 1]$. Подставляя это значение в функцию расстояния, получаем

$$d(k, i) + f^*c_{i,j} = d(k, i) + \frac{d(k, j) - d(k, i) + c_{i,j}}{2} = \frac{d(k, j) + d(k, i) + c_{i,j}}{2}.$$

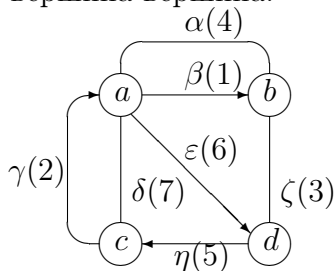
Окончательно имеем формулу для расстояния вершина-ребро

$$(7) \quad d(k, (i, j)) = \frac{d(k, i) + d(k, j) + c_{i,j}}{2}.$$

В случае ориентированного ребра (i, j) максимум, очевидно, достигается в точке $f = 1$, и расстояние вершина-ребро имеет вид

$$(8) \quad d(k, (i, j)) = d(k, i) + c_{i,j}.$$

Пример. Рассмотрим смешанный граф из предыдущего примера и его матрицу расстояний вершина-вершина.



	a	b	c	d
a	0	1	7	4
b	4	0	8	3
c	2	3	0	6
d	7	3	5	0

Найдем для этого графа расстояния вершина-ребро. Подробно рассмотрим поиск расстояний от вершины a до всех ребер графа. Сначала посчитаем расстояния до неориентированных ребер.

$$\begin{aligned}\alpha = (a, b) : d(a, \alpha) &= \frac{d(a, a) + d(a, b) + c_\alpha}{2} = \frac{0 + 1 + 4}{2} = 2,5; \\ \delta = (a, c) : d(a, \delta) &= \frac{d(a, a) + d(a, c) + c_\delta}{2} = \frac{0 + 7 + 7}{2} = 7; \\ \zeta = (b, d) : d(a, \zeta) &= \frac{d(a, b) + d(a, d) + c_\zeta}{2} = \frac{1 + 4 + 3}{2} = 4.\end{aligned}$$

Далее найдем расстояния до ориентированных ребер.

$$\begin{aligned}\beta = (a, b) : d(a, \beta) &= d(a, a) + c_\beta = 0 + 1 = 1; \\ \gamma = (c, a) : d(a, \gamma) &= d(a, c) + c_\gamma = 7 + 2 = 9; \\ \varepsilon = (a, d) : d(a, \varepsilon) &= d(a, a) + c_\varepsilon = 0 + 6 = 6; \\ \eta = (d, c) : d(a, \eta) &= d(a, d) + c_\eta = 4 + 5 = 9.\end{aligned}$$

Поиск расстояний от остальных вершин осуществляется аналогично и предоставляется читателю. Итак, получаем матрицу расстояний вершина-ребро.

	α	β	γ	δ	ε	ζ	η
a	2,5	1	9	7	6	4	9
b	4	5	10	9,5	10	3	8
c	4,5	3	2	4,5	8	6	11
d	7	8	7	9,5	13	3	5

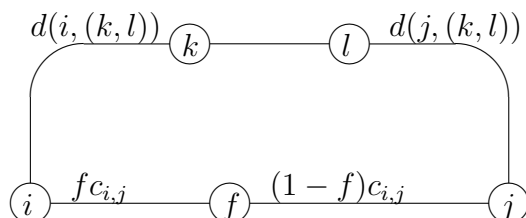
Определение. Расстоянием точка-точка (обозначение $d(f_{(i,j)}, g_{(k,l)})$) называется вес минимального пути от f -точки ребра (i, j) до g -точки ребра (k, l) .

Определение. Расстоянием точка-ребро (обозначение $d(f_{(i,j)}, (k, l))$) называется максимальное из расстояний точка-точка от f -точки ребра (i, j) до g -точки ребра (k, l)

$$(9) \quad d(f_{(i,j)}, (k, l)) = \max_{g \in [0,1]} d(f_{(i,j)}, g_{(k,l)}).$$

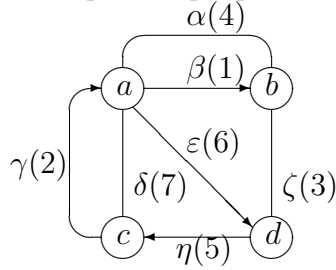
Если ребро (i, j) неориентированное, и $(i, j) \neq (k, l)$, то минимальный путь проходит через вершину i либо j и имеет вид

$$(10) \quad d(f_{(i,j)}, (k, l)) = \min\{f c_{i,j} + d(i, (k, l)), (1 - f) c_{i,j} + d(j, (k, l))\}.$$



Зависимость расстояния точка-ребро от параметра f имеет вид, аналогичный той же зависимости для расстояния точка-вершина.

Пример. Рассмотрим смешанный граф из предыдущих примеров и его матрицу расстояний вершина-ребро.



	α	β	γ	δ	ε	ζ	η
a	2,5	1	9	7	6	4	9
b	4	5	10	9,5	10	3	8
c	4,5	3	2	4,5	8	6	11
d	7	8	7	9,5	13	3	5

Рассмотрим неориентированное ребро $\delta = (a, c)$. Найдем расстояние от f -точки этого ребра до всех ребер графа, кроме δ . По формуле (10) имеем

$$\begin{aligned} d(f_\delta, \alpha) &= \min \{fc_\delta + d(a, \alpha), (1-f)c_\delta + d(c, \alpha)\} = \min \{7f + 2,5, 7(1-f) + 4,5\}; \\ d(f_\delta, \beta) &= \min \{fc_\delta + d(a, \beta), (1-f)c_\delta + d(c, \beta)\} = \min \{7f + 1, 7(1-f) + 3\}; \\ d(f_\delta, \gamma) &= \min \{fc_\delta + d(a, \gamma), (1-f)c_\delta + d(c, \gamma)\} = \min \{7f + 9, 7(1-f) + 2\}; \\ d(f_\delta, \varepsilon) &= \min \{fc_\delta + d(a, \varepsilon), (1-f)c_\delta + d(c, \varepsilon)\} = \min \{7f + 6, 7(1-f) + 8\}; \\ d(f_\delta, \zeta) &= \min \{fc_\delta + d(a, \zeta), (1-f)c_\delta + d(c, \zeta)\} = \min \{7f + 4, 7(1-f) + 6\}; \\ d(f_\delta, \eta) &= \min \{fc_\delta + d(a, \eta), (1-f)c_\delta + d(c, \eta)\} = \min \{7f + 9, 7(1-f) + 11\}. \end{aligned}$$

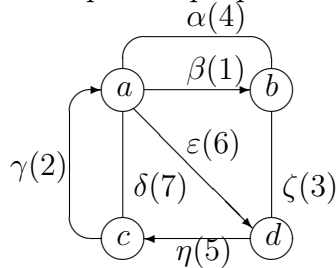
Из условия пересечения прямых найдем точку максимума расстояния f^* :

$$\begin{aligned} \alpha: \quad 7f + 2,5 &= 11,5 - 7f, & f^* &= 9/14; \\ \beta: \quad 7f + 1 &= 10 - 7f, & f^* &= 9/14; \\ \gamma: \quad 7f + 9 &= 9 - 7f, & f^* &= 0; \\ \varepsilon: \quad 7f + 6 &= 15 - 7f, & f^* &= 9/14; \\ \zeta: \quad 7f + 4 &= 13 - 7f, & f^* &= 9/14; \\ \eta: \quad 7f + 9 &= 18 - 7f, & f^* &= 9/14. \end{aligned}$$

Если ребро (i, j) ориентированное, и $(i, j) \neq (k, l)$, то путь может проходить лишь через вершину j , следовательно, формула для расстояния принимает вид

$$(11) \quad d(f_{(i,j)}, (k, l)) = (1-f)c_{i,j} + d(j, (k, l)).$$

Пример. Рассмотрим смешанный граф из предыдущих примеров и его матрицу расстояний вершина-ребро.



	α	β	γ	δ	ε	ζ	η
a	2,5	1	9	7	6	4	9
b	4	5	10	9,5	10	3	8
c	4,5	3	2	4,5	8	6	11
d	7	8	7	9,5	13	3	5

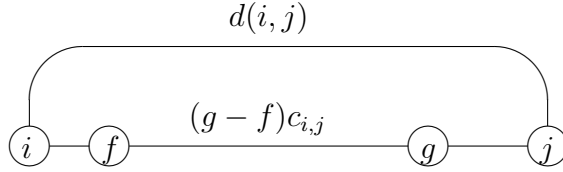
Рассмотрим ориентированное ребро $\gamma = (c, a)$. Найдем расстояние от f -точки этого ребра до всех ребер графа, кроме γ . По формуле (11) имеем

$$\begin{aligned} d(f_\gamma, \alpha) &= (1-f)c_\gamma + d(a, \alpha) = 2(1-f) + 2,5 = 4,5 - 2f; \\ d(f_\gamma, \beta) &= (1-f)c_\gamma + d(a, \beta) = 2(1-f) + 1 = 3 - 2f; \\ d(f_\gamma, \delta) &= (1-f)c_\gamma + d(a, \delta) = 2(1-f) + 7 = 9 - 2f; \\ d(f_\gamma, \varepsilon) &= (1-f)c_\gamma + d(a, \varepsilon) = 2(1-f) + 6 = 8 - 2f; \\ d(f_\gamma, \zeta) &= (1-f)c_\gamma + d(a, \zeta) = 2(1-f) + 4 = 6 - 2f; \\ d(f_\gamma, \eta) &= (1-f)c_\gamma + d(a, \eta) = 2(1-f) + 9 = 11 - 2f. \end{aligned}$$

Пусть ребро (i, j) неориентированное, и $(i, j) = (k, l)$. Рассмотрим расстояние между двумя точками на одном ребре.

Пусть $f < 1/2$. Тогда наиболее удаленные от f точки g лежат ближе к вершине j , т.е. $g > f$. Заметим, что $d(i, j) \leq c_{i,j}$. Если $d(i, j) < c_{i,j}$, может оказаться, что от точки f до точки g выгоднее добраться через вершину i и далее по минимальному пути в вершину j и в точку g по ребру (i, j) . Расстояние точка-точка принимает вид

$$(12) \quad d(f_{(i,j)}, g_{(k,l)}) = \min\{(g - f)c_{i,j}, d(i, j) + (1 - g + f)c_{i,j}\}$$



Максимум этого расстояния достигается в точке g , для которой верно

$$(g - f)c_{i,j} = d(j, i) + (1 - g + f)c_{i,j}.$$

Отсюда получаем

$$(13) \quad \max_g d(f_{(i,j)}, g_{(i,j)}) = \frac{d(i, j) + c_{i,j}}{2}$$

Если $d(i, j) = c_{i,j}$, то минимальный путь от f до g проходит только по ребру (i, j) , и его вес равен

$$(14) \quad d(f_{(i,j)}, g_{(i,j)}) = (g - f)c_{i,j}.$$

Максимум здесь достигается в точке $g = 1$ и равен

$$(15) \quad \max_g d(f_{(i,j)}, g_{(i,j)}) = (1 - f)c_{i,j}.$$

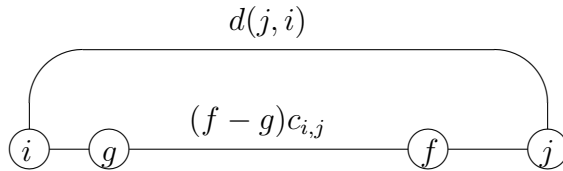
Расстояние точка-ребро для $f < 1/2$ в результате принимает вид

$$(16) \quad d(f_{(i,j)}, (i, j)) = \min\left\{(1 - f)c_{i,j}, \frac{d(i, j) + c_{i,j}}{2}\right\}$$

Это расстояние максимально в точке $f = 0$ и минимально в точке $f = 1/2$, минимум расстояния при этом равен $c_{i,j}/2$.

Пусть $f > 1/2$. Тогда наиболее удаленные от f точки g лежат ближе к вершине i , т.е. $g < f$. Если $d(i, j) < c_{i,j}$, то расстояние точка-точка принимает вид

$$(17) \quad d(f_{(i,j)}, g_{(k,l)}) = \min\{(f - g)c_{i,j}, d(j, i) + (1 - f + g)c_{i,j}\}$$



Максимум этого расстояния достигается в точке g , для которой верно

$$(f - g)c_{i,j} = d(j, i) + (1 - g + f)c_{i,j}.$$

Отсюда получаем

$$(18) \quad \max_g d(f_{(i,j)}, g_{(i,j)}) = \frac{d(j,i) + c_{i,j}}{2}$$

Если $d(i,j) = c_{i,j}$, то минимальный путь от f до g проходит только по ребру (i,j) , и его вес равен

$$(19) \quad d(f_{(i,j)}, g_{(i,j)}) = (f - g)c_{i,j}.$$

Максимум здесь достигается в точке $g = 0$ и равен

$$(20) \quad \max_g d(f_{(i,j)}, g_{(i,j)}) = fc_{i,j}.$$

Расстояние точка-ребро для $f > 1/2$ в результате принимает вид

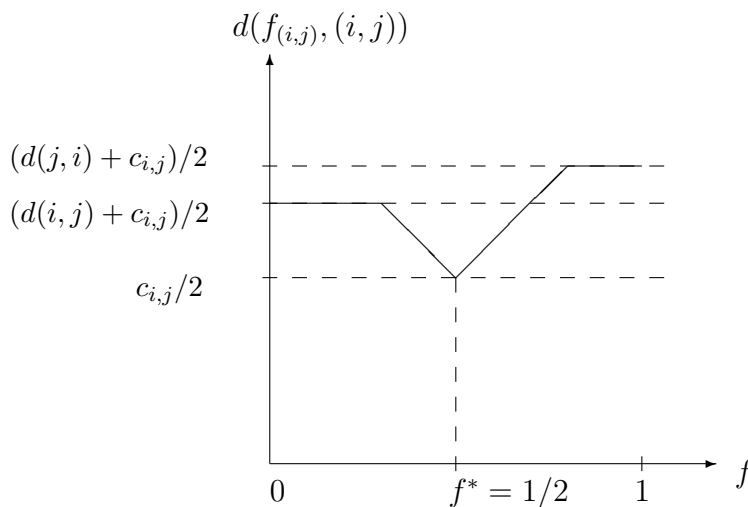
$$(21) \quad d(f_{(i,j)}, (i,j)) = \min \left\{ fc_{i,j}, \frac{d(j,i) + c_{i,j}}{2} \right\}$$

Это расстояние максимально в точке $f = 1$ и минимально в точке $f = 1/2$, минимум расстояния при этом равен $c_{i,j}/2$.

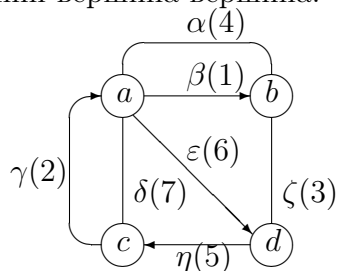
Окончательно имеем

$$(22) \quad d(f_{(i,j)}, (i,j)) = \max \left\{ \min \left\{ (1-f)c_{i,j}, \frac{d(i,j) + c_{i,j}}{2} \right\}, \min \left\{ fc_{i,j}, \frac{d(j,i) + c_{i,j}}{2} \right\} \right\}.$$

Зависимость расстояния точка-ребро от параметра f имеет вид



Пример. Рассмотрим смешанный граф из предыдущих примеров и его матрицу расстояний вершина-вершина.

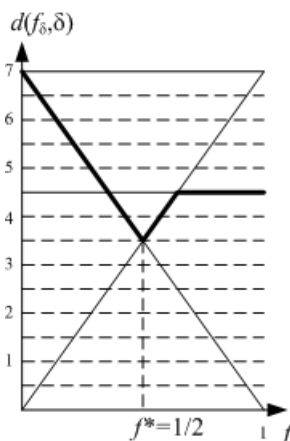


	a	b	c	d
a	0	1	7	4
b	4	0	8	3
c	2	3	0	6
d	7	3	5	0

Рассмотрим неориентированное ребро $\delta = (a, c)$ и найдем расстояние $d(f_\delta, \delta)$. Согласно формуле (22) имеем:

$$\begin{aligned} d(f_\delta, \delta) &= \max \left\{ \min \left\{ (1-f)c_\delta, \frac{d(a,c) + c_\delta}{2} \right\}, \min \left\{ fc_\delta, \frac{d(c,a) + c_\delta}{2} \right\} \right\} = \\ &= \max \left\{ \min \left\{ 7(1-f), \frac{7+7}{2} \right\}, \min \left\{ 7f, \frac{2+7}{2} \right\} \right\} = \\ &= \max \left\{ \min \{7-7f, 7\}, \min \{7f, 4, 5\} \right\}. \end{aligned}$$

Нарисуем графики этих прямых. Функция расстояния точка-ребро выделена жирной линией.



Пусть ребро (i, j) ориентированное, и $(i, j) = (k, l)$. Тогда максимальное расстояние от точки f до точки g будет в том случае, когда точка g стремится к точке f со стороны вершины i . В этом случае

$$(23) \quad d(f_{(i,j)}, (i, j)) = d(j, i) + c_{i,j}.$$

Пример. В графе из предыдущих примеров рассмотрим ребро $\gamma = (c, a)$. Для него по формуле (23)

$$d(f_\gamma, \gamma) = d(a, c) + c_\gamma = 7 + 2 = 9.$$

Определим теперь максимальные и средние расстояния в графе, необходимые для формализации задач размещения.

Максимальное расстояние от вершины i до вершин графа:

$$MBV(i) = \max_j \{d(i, j)\}.$$

Суммарное расстояние от вершины i до вершин графа:

$$CBV(i) = \sum_j \{d(i, j)\}.$$

Максимальное расстояние от f -точки ребра (i, j) до вершин графа:

$$MTB(f_{(i,j)}) = \max_k \{d(f_{(i,j)}, k)\}.$$

Суммарное расстояние от f -точки ребра (i, j) до вершин графа:

$$\text{CTB}(f_{(i,j)}) = \sum_k \{d(f_{(i,j)}, k)\}.$$

Максимальное расстояние от вершины i до ребер графа:

$$\text{MBP}(i) = \max_{(k,l)} \{d(i, (k, l))\}.$$

Суммарное расстояние от вершины i до ребер графа:

$$\text{CBP}(i) = \sum_{(k,l)} \{d(i, (k, l))\}.$$

Максимальное расстояние от f -точки ребра (i, j) до ребер графа:

$$\text{MTP}(f_{(i,j)}) = \max_{(k,l)} \{d(f_{(i,j)}, (k, l))\}.$$

Суммарное расстояние от f -точки ребра (i, j) до ребер графа:

$$\text{CTP}(f_{(i,j)}) = \sum_{(k,l)} \{d(f_{(i,j)}, (k, l))\}.$$

4.2 Поиск центров графа

Определение. *Центром графа* называется любая его вершина v , для которой

$$\text{MBV}(v) = \min_j \text{MBV}(j).$$

Определение. *Главным центром графа* называется любая его вершина v , для которой

$$\text{MBP}(v) = \min_j \text{MBP}(j).$$

Определение. *Абсолютным центром графа* называется любая его точка $g_{(v,u)}$, для которой

$$\text{MTB}(g_{(v,u)}) = \min_{f_{(i,j)}} \text{MTB}(f_{(i,j)}).$$

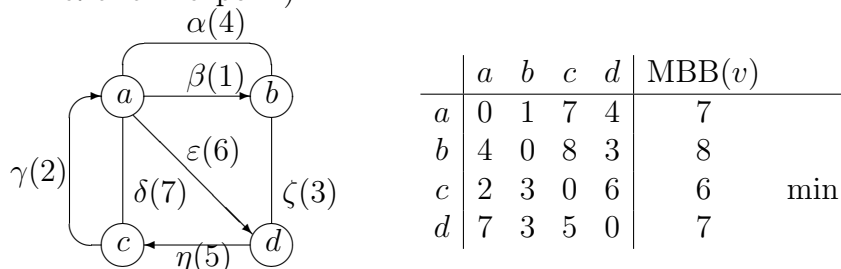
Определение. *Главным абсолютным центром графа* называется любая его точка $g_{(v,u)}$, для которой

$$\text{MTP}(g_{(v,u)}) = \min_{f_{(i,j)}} \text{MTP}(f_{(i,j)}).$$

Таким образом, центры графа – это вершины или точки, максимальное расстояние от которых до вершин или ребер минимально. При этом абсолютные центры – это точки, главные центры – вершины или точки, от которых ищется расстояние до ребер.

Поиск центра. Чтобы найти центр графа, требуется найти все расстояния вершина-вершина. Затем для каждой вершины v вычислить $\text{MBV}(v)$. Центром будет любая вершина, для которой эта величина минимальна.

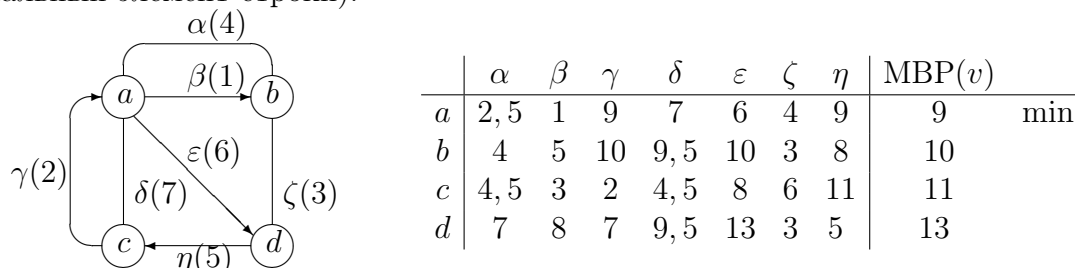
Пример. Рассмотрим смешанный граф из предыдущих примеров и его матрицу расстояний вершина-вершина. Добавим справа в матрицу столбец, в котором для каждой вершины укажем максимальное расстояние вершина-вершина (это максимальный элемент строки).



Минимум максимального расстояния вершина-вершина достигается в вершине c , значит, вершина c является центром графа.

Поиск главного центра. Чтобы найти главный центр графа, требуется найти все расстояния вершина-ребро. Затем для каждой вершины v вычислить $\text{МВР}(v)$. Главным центром будет любая вершина, для которой эта величина минимальна.

Пример. Рассмотрим смешанный граф из предыдущих примеров и его матрицу расстояний вершина-ребро. Добавим справа в матрицу столбец, в котором для каждой вершины укажем максимальное расстояние вершина-ребро (это максимальный элемент строки).



Минимум максимального расстояния вершина-ребро достигается в вершине a , значит, вершина a является главным центром графа.

Поиск абсолютного центра. Абсолютным центром является любая точка графа, расстояние от которой до наиболее удаленной вершины графа минимально. Очевидно, эта задача сложнее, чем предыдущие, поскольку нужно кроме вершин рассмотреть и внутренние точки ребер. Перебор можно сократить, используя следующее утверждение.

Утверждение 1. Ни одна внутренняя точка ориентированного ребра не может быть абсолютным центром.

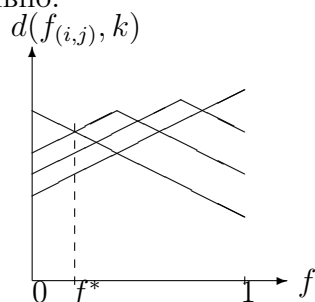
Доказательство. Согласно формуле (3)

$$d(f_{(i,j)}, k) = (1 - f)c_{i,j} + d(j, k).$$

Минимум этого расстояния достигается в точке $f = 1$, т.е. в вершине j .

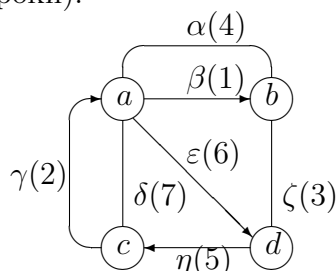
Из утверждения следует, что при поиске абсолютного центра достаточно рассмотреть только вершины графа и внутренние точки его неориентированных ребер. Поиск внутренних точек – претендентов на абсолютный центр можно выполнить графически. Как уже было показано, расстояние точка-вершина представляет собой отрезок прямой либо ломаную из двух отрезков. Если изобразить на

одном графике все расстояния точка-вершина, то на графике получится замкнутая область, ограниченная сверху ломаной линией. Эта верхняя граница представляет собой максимальное расстояние точка-вершина в зависимости от точки. Претендентом на абсолютный центр будет точка, в которой значение этой границы минимально.



Затем среди найденных точек-претендентов и вершин графа выбирается точка, максимальное расстояние от которой до вершин графа минимально.

Пример. Рассмотрим граф из предыдущих примеров и его матрицу расстояний вершина-вершина. Добавим справа в матрицу столбец, в котором для каждой вершины укажем максимальное расстояние вершина-вершина (это максимальный элемент строки).



	a	b	c	d	$\text{MBV}(v)$
a	0	1	7	4	7
b	4	0	8	3	8
c	2	3	0	6	6
d	7	3	5	0	7

Для точек неориентированных ребер построим на одном графике все расстояния точка-вершина. Для ребра $\delta = (a, c)$ графики были построены в одном из предыдущих примеров. По формуле (1) имеем

$$\begin{aligned} d(f_\delta, a) &= \min \{fc_\delta + d(a, a), (1-f)c_\delta + d(c, a)\} = \min \{7f + 0, 7(1-f) + 2\}; \\ d(f_\delta, b) &= \min \{fc_\delta + d(a, b), (1-f)c_\delta + d(c, b)\} = \min \{7f + 1, 7(1-f) + 3\}; \\ d(f_\delta, c) &= \min \{fc_\delta + d(a, c), (1-f)c_\delta + d(c, c)\} = \min \{7f + 7, 7(1-f) + 0\}; \\ d(f_\delta, d) &= \min \{fc_\delta + d(a, d), (1-f)c_\delta + d(c, d)\} = \min \{7f + 4, 7(1-f) + 6\}. \end{aligned}$$

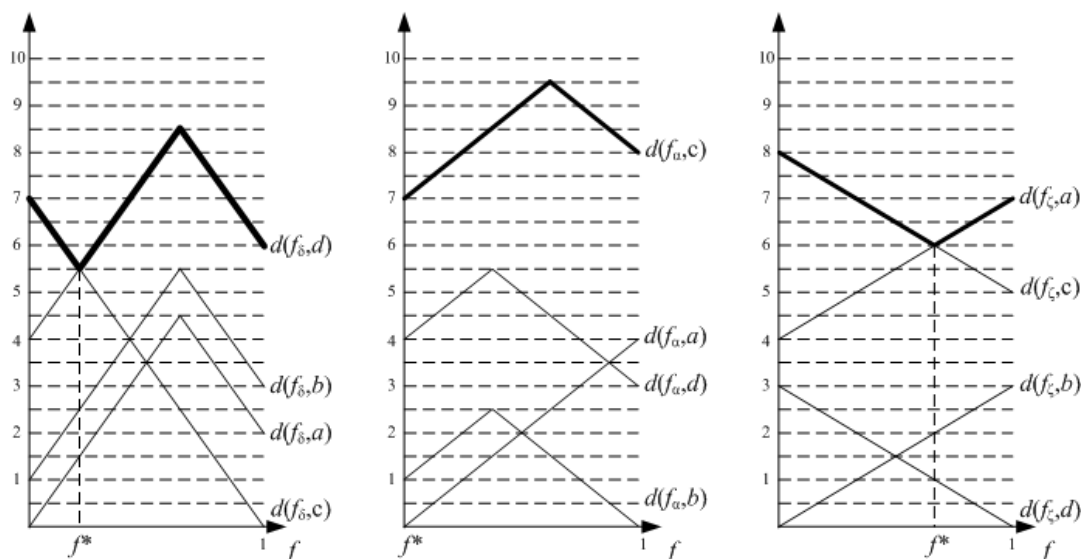
Найдем расстояния точка-вершина для точек ребра $\alpha = (a, b)$. По формуле (1) имеем

$$\begin{aligned} d(f_\alpha, a) &= \min \{fc_\alpha + d(a, a), (1-f)c_\alpha + d(b, a)\} = \min \{4f + 0, 4(1-f) + 4\}; \\ d(f_\alpha, b) &= \min \{fc_\alpha + d(a, b), (1-f)c_\alpha + d(b, b)\} = \min \{4f + 1, 4(1-f) + 0\}; \\ d(f_\alpha, c) &= \min \{fc_\alpha + d(a, c), (1-f)c_\alpha + d(b, c)\} = \min \{4f + 7, 4(1-f) + 8\}; \\ d(f_\alpha, d) &= \min \{fc_\alpha + d(a, d), (1-f)c_\alpha + d(b, d)\} = \min \{4f + 4, 4(1-f) + 3\}. \end{aligned}$$

Найдем расстояния точка-вершина для точек ребра $\zeta = (b, d)$. По формуле (1) имеем

$$\begin{aligned} d(f_\zeta, a) &= \min \{fc_\zeta + d(b, a), (1-f)c_\zeta + d(d, a)\} = \min \{3f + 4, 3(1-f) + 7\}; \\ d(f_\zeta, b) &= \min \{fc_\zeta + d(b, b), (1-f)c_\zeta + d(d, b)\} = \min \{3f + 0, 3(1-f) + 3\}; \\ d(f_\zeta, c) &= \min \{fc_\zeta + d(b, c), (1-f)c_\zeta + d(d, c)\} = \min \{3f + 8, 3(1-f) + 5\}; \\ d(f_\zeta, d) &= \min \{fc_\zeta + d(b, d), (1-f)c_\zeta + d(d, d)\} = \min \{3f + 3, 3(1-f) + 0\}. \end{aligned}$$

Построим графики этих функций. Функция максимального расстояния точка-вершина выделена жирной линией.



На графике слева изображена функция максимального расстояния точка-вершина от точки f_δ . Эта функция принимает минимальное значение в точке f^* , которая находится из условия пересечения прямых

$$7 - 7f = 4 + 7f,$$

откуда получаем

$$f^* = 3/14, \quad \text{МТВ}(f_\delta^*) = 7 - 7 \times 3/14 = 5,5.$$

На графике посередине изображена функция максимального расстояния точка-вершина от точки f_α . Эта функция принимает минимальное значение в точке $f^* = 0$, т.е. в вершине a . На графике справа изображена функция максимального расстояния точка-вершина от точки f_ζ . Эта функция принимает минимальное значение в точке f^* , которая находится из условия пересечения прямых

$$8 - 3f = 4 + 3f,$$

откуда получаем

$$f^* = 2/3, \quad \text{МТВ}(f_\zeta^*) = 8 - 3 \times 2/3 = 6.$$

Сравнивая полученные расстояния точка-вершина с вычисленными ранее расстояниями вершина-вершина, получаем, что минимальное расстояние точка-вершина достигается в точке $(3/14)_\delta$ и равно 5,5. Значит, абсолютным центром является точка, лежащая на ребре $\delta = (a, c)$ и делящая его в пропорции $(3/14) : (11/14)$.

Поиск главного абсолютного центра. Главным абсолютным центром является любая точка графа, расстояние от которой до наиболее удаленного ребра графа минимально. Решение этой задачи аналогично решению предыдущей, только в качестве расстояний используются расстояния точка-ребро.

Утверждение 2. Если внутренняя точка ориентированного ребра является главным абсолютным центром, то конец этого ребра также является главным абсолютным центром.

Доказательство. Согласно формуле (11), если $(i, j) \neq (k, l)$

$$d(f_{(i,j)}, (k, l)) = (1 - f)c_{i,j} + d(j, (k, l)).$$

Минимум этого расстояния достигается в точке $f = 1$, т.е. в вершине j . Для $(i, j) = (k, l)$, согласно формуле (22),

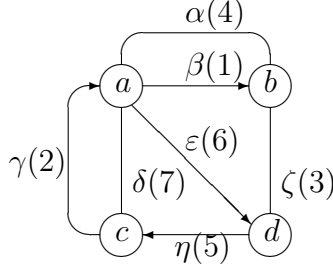
$$d(f_{(i,j)}, (i, j)) = d(j, i) + c_{i,j},$$

т.е. расстояние от точки до ребра равно константе. Следовательно, для любой внутренней точки f

$$\max_{(k,l)} d(f_{(i,j)}, (k, l)) \geq \max_{(k,l)} d(j, (k, l)).$$

Таким образом, как и при поиске абсолютного центра, в качестве претендентов рассматриваются вершины графа и внутренние точки неориентированных ребер. Из них выбирается точка, расстояние от которой до наиболее удаленного ребра минимально.

Пример. Рассмотрим смешанный граф из предыдущих примеров и его матрицу расстояний вершина-ребро. Добавим справа в матрицу столбец, в котором для каждой вершины укажем максимальное расстояние вершина-ребро (это максимальный элемент строки).



	α	β	γ	δ	ε	ζ	η	MBP(v)
a	2,5	1	9	7	6	4	9	9
b	4	5	10	9,5	10	3	8	10
c	4,5	3	2	4,5	8	6	11	11
d	7	8	7	9,5	13	3	5	13

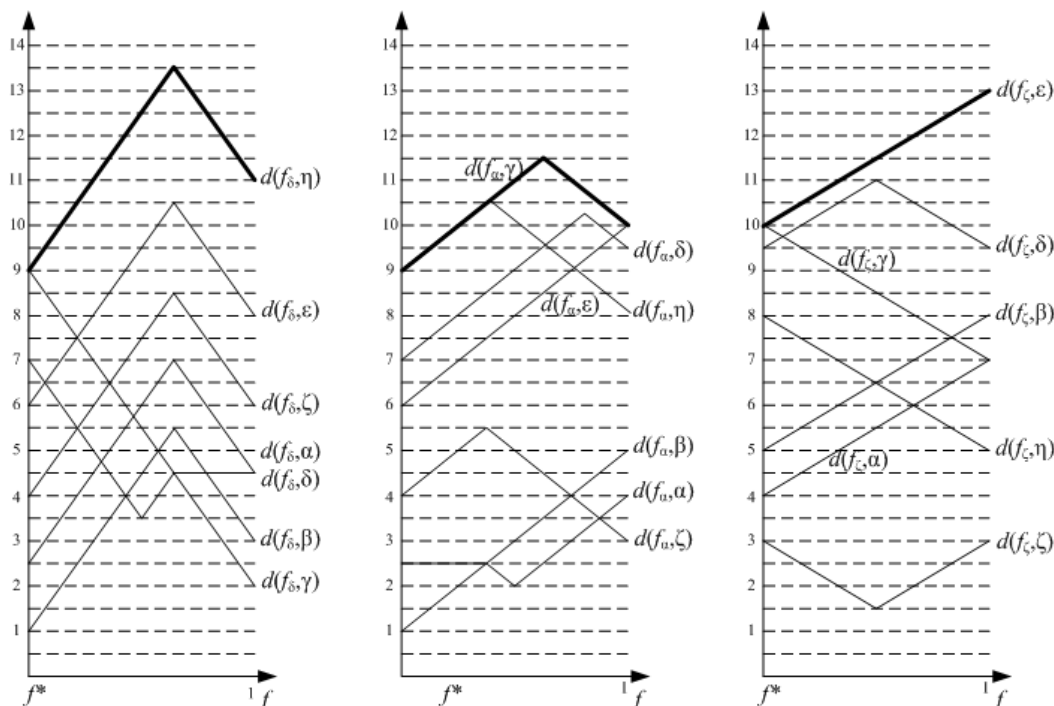
Найдем функции расстояний точка-ребро для внутренних точек неориентированных ребер. Для ребра δ эти расстояния были найдены ранее. По формуле (10) имеем

$$\begin{aligned} d(f_\delta, \alpha) &= \min \{fc_\delta + d(a, \alpha), (1 - f)c_\delta + d(c, \alpha)\} = \min \{7f + 2, 5, 7(1 - f) + 4, 5\}; \\ d(f_\delta, \beta) &= \min \{fc_\delta + d(a, \beta), (1 - f)c_\delta + d(c, \beta)\} = \min \{7f + 1, 7(1 - f) + 3\}; \\ d(f_\delta, \gamma) &= \min \{fc_\delta + d(a, \gamma), (1 - f)c_\delta + d(c, \gamma)\} = \min \{7f + 9, 7(1 - f) + 2\}; \\ d(f_\delta, \varepsilon) &= \min \{fc_\delta + d(a, \varepsilon), (1 - f)c_\delta + d(c, \varepsilon)\} = \min \{7f + 6, 7(1 - f) + 8\}; \\ d(f_\delta, \zeta) &= \min \{fc_\delta + d(a, \zeta), (1 - f)c_\delta + d(c, \zeta)\} = \min \{7f + 4, 7(1 - f) + 6\}; \\ d(f_\delta, \eta) &= \min \{fc_\delta + d(a, \eta), (1 - f)c_\delta + d(c, \eta)\} = \min \{7f + 9, 7(1 - f) + 11\}. \end{aligned}$$

Согласно формуле (22) имеем:

$$\begin{aligned} d(f_\delta, \delta) &= \max \left\{ \min \left\{ (1 - f)c_\delta, \frac{d(a, c) + c_\delta}{2} \right\}, \min \left\{ fc_\delta, \frac{d(c, a) + c_\delta}{2} \right\} \right\} = \\ &= \max \left\{ \min \left\{ 7(1 - f), \frac{7 + 7}{2} \right\}, \min \left\{ 7f, \frac{2 + 7}{2} \right\} \right\} = \\ &= \max \{ \min \{ 7 - 7f, 7 \}, \min \{ 7f, 4, 5 \} \}. \end{aligned}$$

График функции расстояния точка-ребро для точки f_δ , а также графики для внутренних точек остальных ребер, приведены ниже. Функция максимального расстояния точка-ребро выделена жирной линией.



Получим расстояния точка-ребро для остальных неориентированных ребер. Рассмотрим ребро $\alpha = (a, b)$. По формуле (10) имеем

$$\begin{aligned} d(f_\alpha, \beta) &= \min \{fc_\alpha + d(a, \beta), (1-f)c_\alpha + d(b, \beta)\} = \min \{4f + 1, 4(1-f) + 5\}; \\ d(f_\alpha, \gamma) &= \min \{fc_\alpha + d(a, \gamma), (1-f)c_\alpha + d(b, \gamma)\} = \min \{4f + 9, 4(1-f) + 10\}; \\ d(f_\alpha, \delta) &= \min \{fc_\alpha + d(a, \delta), (1-f)c_\alpha + d(b, \delta)\} = \min \{4f + 7, 4(1-f) + 9, 5\}; \\ d(f_\alpha, \varepsilon) &= \min \{fc_\alpha + d(a, \varepsilon), (1-f)c_\alpha + d(b, \varepsilon)\} = \min \{4f + 6, 4(1-f) + 10\}; \\ d(f_\alpha, \zeta) &= \min \{fc_\alpha + d(a, \zeta), (1-f)c_\alpha + d(b, \zeta)\} = \min \{4f + 4, 4(1-f) + 3\}; \\ d(f_\alpha, \eta) &= \min \{fc_\alpha + d(a, \eta), (1-f)c_\alpha + d(b, \eta)\} = \min \{4f + 9, 4(1-f) + 8\}. \end{aligned}$$

Согласно формуле (22) имеем:

$$\begin{aligned} d(f_\alpha, \alpha) &= \max \left\{ \min \left\{ (1-f)c_\alpha, \frac{d(a, b) + c_\alpha}{2} \right\}, \min \left\{ fc_\alpha, \frac{d(b, a) + c_\alpha}{2} \right\} \right\} = \\ &= \max \left\{ \min \left\{ 4(1-f), \frac{1+4}{2} \right\}, \min \left\{ 4f, \frac{4+4}{2} \right\} \right\} = \\ &= \max \{ \min \{ 4-4f, 2, 5 \}, \min \{ 4f, 4 \} \}. \end{aligned}$$

Рассмотрим ребро $\zeta = (b, d)$. По формуле (10) имеем

$$\begin{aligned} d(f_\zeta, \alpha) &= \min \{fc_\zeta + d(b, \alpha), (1-f)c_\zeta + d(d, \alpha)\} = \min \{3f + 4, 3(1-f) + 7\}; \\ d(f_\zeta, \beta) &= \min \{fc_\zeta + d(b, \beta), (1-f)c_\zeta + d(d, \beta)\} = \min \{3f + 5, 3(1-f) + 8\}; \\ d(f_\zeta, \gamma) &= \min \{fc_\zeta + d(b, \gamma), (1-f)c_\zeta + d(d, \gamma)\} = \min \{3f + 10, 3(1-f) + 7\}; \\ d(f_\zeta, \delta) &= \min \{fc_\zeta + d(b, \delta), (1-f)c_\zeta + d(d, \delta)\} = \min \{3f + 9, 5, 3(1-f) + 9, 5\}; \\ d(f_\zeta, \varepsilon) &= \min \{fc_\zeta + d(b, \varepsilon), (1-f)c_\zeta + d(d, \varepsilon)\} = \min \{3f + 10, 3(1-f) + 13\}; \\ d(f_\zeta, \eta) &= \min \{fc_\zeta + d(b, \eta), (1-f)c_\zeta + d(d, \eta)\} = \min \{3f + 8, 3(1-f) + 5\}. \end{aligned}$$

Согласно формуле (22) имеем:

$$\begin{aligned} d(f_\zeta, \zeta) &= \max \left\{ \min \left\{ (1-f)c_\zeta, \frac{d(b,d) + c_\zeta}{2} \right\}, \min \left\{ fc_\zeta, \frac{d(d,b) + c_\zeta}{2} \right\} \right\} = \\ &= \max \left\{ \min \left\{ 3(1-f), \frac{3+3}{2} \right\}, \min \left\{ 3f, \frac{3+3}{2} \right\} \right\} = \\ &= \max \{ \min \{ 3-3f, 3 \}, \min \{ 3f, 3 \} \}. \end{aligned}$$

На графике слева на предыдущей странице изображена функция максимального расстояния точка-ребро от точки f_δ . Эта функция принимает минимальное значение в точке $f^* = 0$, т.е. в вершине a . На графике посередине изображена функция максимального расстояния точка-ребро от точки f_α . Эта функция принимает минимальное значение в точке $f^* = 0$, т.е. в вершине a . На графике справа изображена функция максимального расстояния точка-ребро от точки f_ζ . Эта функция принимает минимальное значение в точке $f^* = 0$, т.е. в вершине b . Таким образом, никакие внутренние точки неориентированных ребер не могут быть главными абсолютными центрами, и главный абсолютный центр совпадает с главным центром и находится в вершине a .

4.3 Поиск медиан графа

Определение. *Медианой графа* называется любая его вершина v , для которой

$$СВВ(v) = \min_j СВВ(j).$$

Определение. *Главной медианой графа* называется любая его вершина v , для которой

$$СВР(v) = \min_j СВР(j).$$

Определение. *Абсолютной медианой графа* называется любая его точка $g_{(v,u)}$, для которой

$$СТВ(g_{(v,u)}) = \min_{f_{(i,j)}} СТВ(f_{(i,j)}).$$

Определение. *Главной абсолютной медианой графа* называется любая его точка $g_{(v,u)}$, для которой

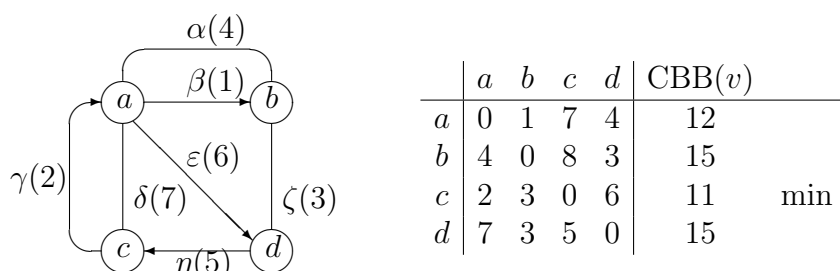
$$СТР(g_{(v,u)}) = \min_{f_{(i,j)}} СТР(f_{(i,j)}).$$

Таким образом, медианы графа – это вершины или точки, суммарное расстояние от которых до вершин или ребер минимально. При этом абсолютные медианы – это точки, главные медианы – вершины или точки, от которых ищется расстояние до ребер.

Поиск медианы. Чтобы найти медиану графа, требуется найти все расстояния вершина-вершина. Затем для каждой вершины v вычислить $СВВ(v)$. Центром будет любая вершина, для которой эта величина минимальна.

Пример. Рассмотрим смешанный граф из предыдущих примеров и его матрицу расстояний вершина-вершина. Добавим справа в матрицу столбец, в котором

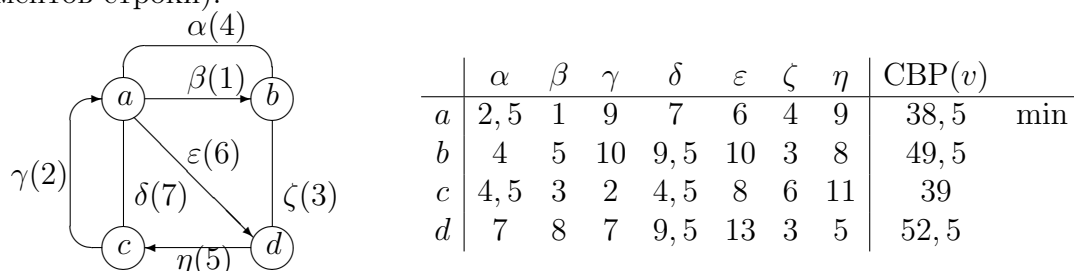
для каждой вершины укажем суммарное вершина-вершина (это сумма элементов строки).



Минимум суммарного расстояния вершина-вершина достигается в вершине c, значит, вершина c является медианой графа.

Поиск главной медианы. Чтобы найти главную медиану графа, требуется найти все расстояния вершина-ребро. Затем для каждой вершины v вычислить СВР(v). Центром будет любая вершина, для которой эта величина минимальна.

Пример. Рассмотрим смешанный граф из предыдущих примеров и его матрицу расстояний вершина-ребро. Добавим справа в матрицу столбец, в котором для каждой вершины укажем суммарное расстояние вершина-ребро (это сумма элементов строки).



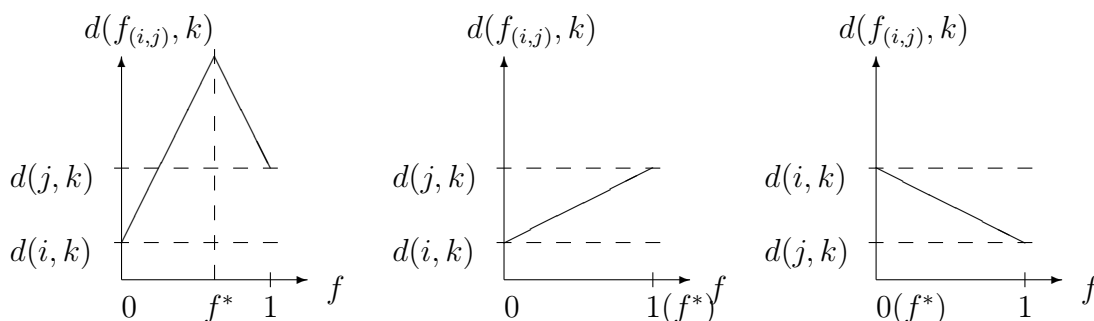
Минимум суммарного расстояния вершина-ребро достигается в вершине a, значит, вершина a является главной медианой графа.

Поиск абсолютной медианы. Абсолютная медиана – это точка, суммарное расстояние от которой до вершин графа минимально.

Поиск абсолютной медианы можно свести к поиску медианы, используя следующий результат.

Теорема 1. В графе всегда существует вершина, являющаяся абсолютной медианой.

Доказательство. Расстояние точка-вершина всегда является вогнутой функцией от f.



Это означает, что для любых $f, g : 0 \leq \alpha \leq 1$

$$d((\alpha f + (1 - \alpha)g)_{(i,j)}, k) \leq \alpha d(f_{(i,j)}, k) + (1 - \alpha)d(g_{(i,j)}, k).$$

Свое минимальное значение такая функция принимает на одной из границ области определения, т.е. минимальное расстояние точка-вершина достигается в одном из концов ребра. Сумма вогнутых функций также является вогнутой, что может быть проверено непосредственно. Значит, минимальная сумма расстояний точка-вершина также достигается в одном из концов ребра (i, j) .

Из теоремы следует, что любая медиана является также абсолютной медианой.

Пример. Для смешаного графа из предыдущих примеров главной медианой является вершина c , поскольку она является медианой графа.

Поиск главной абсолютной медианы. Абсолютная медиана – это точка, суммарное расстояние от которой до вершин графа минимально.

Утверждение 4. Ни одна внутренняя точка ориентированного ребра не может быть главной абсолютной медианой.

Доказательство. Согласно формуле (11), если $(i, j) \neq (k, l)$

$$d(f_{(i,j)}, (k, l)) = (1 - f)c_{i,j} + d(j, (k, l)).$$

Минимум этого расстояния достигается в точке $f = 1$, т.е. в вершине j , причем для любой внутренней точки f

$$d(f_{(i,j)}, (k, l)) < d(j, (k, l)).$$

Для $(i, j) = (k, l)$, согласно формуле (22),

$$d(f_{(i,j)}, (i, j)) = d(j, i) + c_{i,j},$$

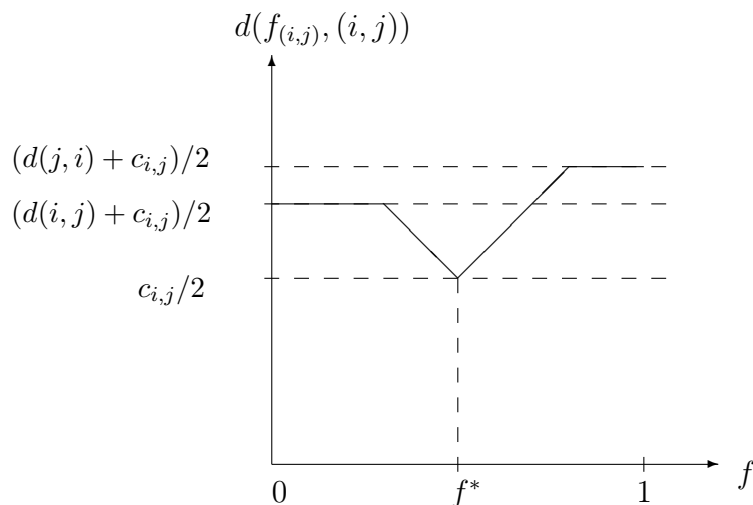
т.е. расстояние от точки до ребра равно константе. Следовательно, для любой внутренней точки f

$$\sum_{(k,l)} d(f_{(i,j)}, (k, l)) < \sum_{(k,l)} d(j, (k, l)).$$

Таким образом, претендентами на главную абсолютную медиану могут внутренние точки только неориентированных ребер. Следующая теорема показывает, что достаточно рассмотреть концы и середину неориентированного ребра.

Теорема 2. В графе всегда существует либо вершина, либо середина неориентированного ребра, являющаяся абсолютной медианой.

Доказательство. В случае $(i, j) \neq (k, l)$ расстояние от точки $f_{(i,j)}$ является вогнутой функцией от f . В случае $(i, j) = (k, l)$ это расстояние имеет вид



Эта функция является вогнутой на участках $[0, 1/2]$ и $[1/2, 1]$. Следовательно, суммарное расстояние точка-ребро также является вогнутой функцией на участках $[0, 1/2]$ и $[1/2, 1]$, и может принять минимальное значение только на концах этих интервалов, т.е. в точках $0, 1, 1/2$. Таким образом, в качестве претендентов на главную абсолютную медиану должны быть рассмотрены все вершины графа и середины неориентированных ребер, причем расстояние точка-ребро для точки – середины ребра для $(i, j) \neq (k, l)$ принимает вид

$$d((1/2)_{(i,j)}, (k, l)) = \frac{1}{2}c_{i,j} + \min\{d(i, (k, l)), d(j, (k, l))\}.$$

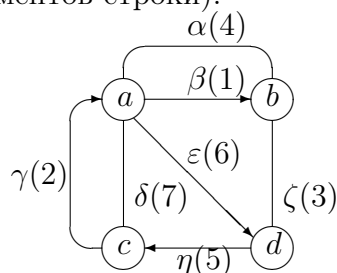
В случае $(i, j) = (k, l)$

$$d((1/2)_{(i,j)}, (i, j)) = \frac{1}{2}c_{i,j}.$$

Суммарное расстояние от середины неориентированного ребра до всех ребер графа принимает вид

$$(24) \quad \text{СТР} \left(\frac{1}{2} \right)_{(i,j)} = \frac{q}{2}c_{i,j} + \sum_{(k,l) \neq (i,j)} \min\{d(i, (k, l)), d(j, (k, l))\}.$$

Пример. Рассмотрим смешанный граф из предыдущих примеров и его матрицу расстояний вершина-ребро. Добавим справа в матрицу столбец, в котором для каждой вершины укажем суммарное расстояние вершина-ребро (это сумма элементов строки).



	α	β	γ	δ	ε	ζ	η	CBP(v)
a	2,5	1	9	7	6	4	9	38,5 min
b	4	5	10	9,5	10	3	8	49,5
c	4,5	3	2	4,5	8	6	11	39
d	7	8	7	9,5	13	3	5	52,5

Для этого графа претендентами на главную абсолютную медиану являются середины неориентированных ребер $\alpha = (a, b)$, $\delta = (a, c)$ и $\zeta = (b, d)$. Используя формулу (24), получаем:

$$\begin{aligned} \text{СТР} \left(\frac{1}{2} \right)_{\alpha} &= \frac{7}{2}c_{\alpha} + \sum_{e \neq \alpha} \min\{d(a, e), d(b, e)\} = \frac{7}{2}4 + \min\{1, 5\} + \min\{9, 10\} + \\ &\quad + \min\{7, 9, 5\} + \min\{6, 10\} + \min\{4, 3\} + \min\{9, 8\} = \\ &\quad = 14 + 1 + 9 + 7 + 6 + 3 + 9 = 49; \\ \text{СТР} \left(\frac{1}{2} \right)_{\delta} &= \frac{7}{2}c_{\delta} + \sum_{e \neq \delta} \min\{d(a, e), d(c, e)\} = \frac{7}{2}7 + \min\{2, 5, 4, 5\} + \min\{1, 3\} + \\ &\quad + \min\{9, 2\} + \min\{6, 8\} + \min\{4, 6\} + \min\{9, 11\} = \\ &\quad = 24,5 + 2,5 + 1 + 2 + 6 + 4 + 9 = 49; \\ \text{СТР} \left(\frac{1}{2} \right)_{\zeta} &= \frac{7}{2}c_{\zeta} + \sum_{e \neq \zeta} \min\{d(b, e), d(d, e)\} = \frac{7}{2}3 + \min\{4, 7\} + \min\{5, 8\} + \\ &\quad + \min\{10, 7\} + \min\{9, 5, 9, 5\} + \min\{10, 13\} + \min\{8, 5\} = \\ &\quad = 10,5 + 4 + 5 + 7 + 9,5 + 10 + 5 = 51. \end{aligned}$$

Все суммарные расстояния точка-ребро для середин неориентированных ребер превышают суммарное расстояние вершина-ребро от вершины a , которая является главной медианой графа. Следовательно, она также является главной абсолютной медианой.

4.4 Обобщения задач размещения

Взвешенные задачи размещения. Предположим, что вершинам и/или ребрам приписаны некоторые числа $W(i)$ и $W(i, j)$, которые имеют смысл вероятностей необходимости посещения вершины/ребра (или частоты посещения). Ясно, что чем больше величина $W(i)$ ($W(i, j)$), тем ближе к данной вершине (ребру) необходимо разместить пункт обслуживания.

В применении к данному графу можно рассмотреть задачи, аналогичные задачам размещения, т.е. задачи минимизации максимального либо суммарного расстояния до вершин (ребер) графа с учетом вероятностей посещения этих вершин.

Подобные взвешенные задачи размещения могут быть сведены к обычным задачам поиска центров и медиан, если рассматривать вместо расстояния вершина-вершина величину

$$d^*(i, j) = W(j)d(i, j),$$

а вместо расстояния вершина/ребро величину

$$d^*(i, (k, l)) = W(k, l)d(i, (k, l)).$$

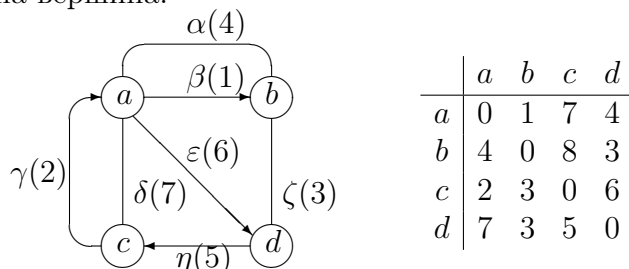
Поиск кратных центров и медиан. Предположим, требуется оптимально разместить несколько пунктов обслуживания, чтобы минимизировать максимальное расстояние до вершины или ребра графа. Пусть r – число таких пунктов. Поставим задачу на языке теории графов.

Пусть X_r – множество точек мощности r . В частном случае можно рассматривать 0-точки, т.е. вершины графа.

Определение. Расстоянием множество-вершина между множеством точек X_r и вершиной j назовем минимальное из расстояний от точек множества X_r до вершины j , т.е.

$$d(X_r, j) = \min_{i \in X_r} d(i, j).$$

Пример. Рассмотрим граф из предыдущих примеров и его матрицу расстояний вершина-вершина.



Рассмотрим множество из трех точек: вершина c , точка $(2/7)_\delta$, где $delta = (a, c)$, и $(1/2)_\alpha$, где $\alpha = (a, b)$. Таким образом, $X_3 = \{b, (2/7)_\delta, (1/2)_\alpha\}$. Найдем расстояние

$d(X_3, d)$. Расстояние $d(c, d) = 6$. По формуле (10) имеем:

$$d\left(\frac{2}{7}\delta, d\right) = \min \left\{ \frac{2}{7}c_\delta + d(a, d), \left(1 - \frac{2}{7}\right)c_\delta + d(c, d) \right\} = \min \left\{ \frac{2}{7}7 + 4, \frac{5}{7}7 + 6 \right\} = 6;$$

$$d\left(\frac{1}{2}\alpha, d\right) = \min \left\{ \frac{1}{2}c_\alpha + d(a, d), \left(1 - \frac{1}{2}\right)c_\alpha + d(b, d) \right\} = \min \left\{ \frac{1}{2}4 + 4, \frac{1}{2}4 + 3 \right\} = 5.$$

Таким образом

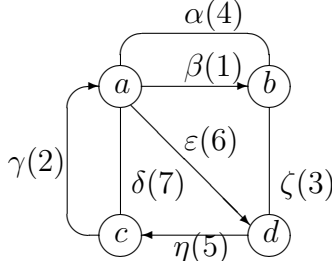
$$d(X_3, d) = \min \left\{ d(c, d), d\left(\frac{2}{7}\delta, d\right), d\left(\frac{1}{2}\alpha, d\right) \right\} = \min \{6, 6, 5\} = 5.$$

Минимум достигается в точке $(1/2)_\alpha$, минимальный путь проходит из этой точки через вершину b .

Определение. Расстоянием множество–ребро между множеством точек X_r и ребром (k, l) назовем минимальное из расстояний от точек множества X_r до ребра (k, l) , т.е.

$$d(X_r, (k, l)) = \min_{i \in X_r} d(i, (k, l)).$$

Рассмотрим смешанный граф из предыдущих примеров и его матрицу расстояний вершина-ребро.



	α	β	γ	δ	ε	ζ	η
a	2,5	1	9	7	6	4	9
b	4	5	10	9,5	10	3	8
c	4,5	3	2	4,5	8	6	11
d	7	8	7	9,5	13	3	5

Рассмотрим множество из трех точек из предыдущего примера: $X_3 = \{b, (2/7)_\delta, (1/2)_\alpha\}$. Найдем расстояние $d(X_3, \eta)$. Расстояние $d(c, \eta) = 11$. По формуле (10) имеем:

$$d\left(\frac{2}{7}\delta, \eta\right) = \min \left\{ \frac{2}{7}c_\delta + d(a, d\eta), \left(1 - \frac{2}{7}\right)c_\delta + d(c, \eta) \right\} = \min \left\{ \frac{2}{7}7 + 9, \frac{5}{7}7 + 11 \right\} = 11;$$

$$d\left(\frac{1}{2}\alpha, \eta\right) = \min \left\{ \frac{1}{2}c_\alpha + d(a, \eta), \left(1 - \frac{1}{2}\right)c_\alpha + d(b, \eta) \right\} = \min \left\{ \frac{1}{2}4 + 9, \frac{1}{2}4 + 8 \right\} = 10.$$

Таким образом

$$d(X_3, \eta) = \min \left\{ d(c, \eta), d\left(\frac{2}{7}\delta, \eta\right), d\left(\frac{1}{2}\alpha, \eta\right) \right\} = \min \{11, 11, 10\} = 10.$$

Минимум достигается в точке $(1/2)_\alpha$, минимальный путь проходит из этой точки через вершину b .

Поставим задачу поиска *кратных центров* (r -центров) и *кратных медиан* (r -медиан) как задачу поиска множества X_r , для которого максимальное/суммарное расстояние до вершин/ребер графа минимально.

Можно сформулировать и другую задачу: найти множество X_r наименьшей мощности, для которого максимальное/суммарное расстояние до вершин/ребер графа не превосходит заданной величины δ .

Аналогично можно сформулировать задачи и для абсолютных кратных центров и медиан.

4.5 Поиск кратных центров

Рассмотрим задачу поиска кратных абсолютных центров графа. Методы ее решения можно легко модифицировать для поиска кратных центров, кратных главных центров и главных абсолютных центров.

Изучим точный метод решения, основанный на поиске кратчайшего покрытия булевой матрицы. Введем некоторые определения.

Определение. Будем говорить, что вершина k достижима из точки $f_{(i,j)}$ в пределах расстояния Δ , если

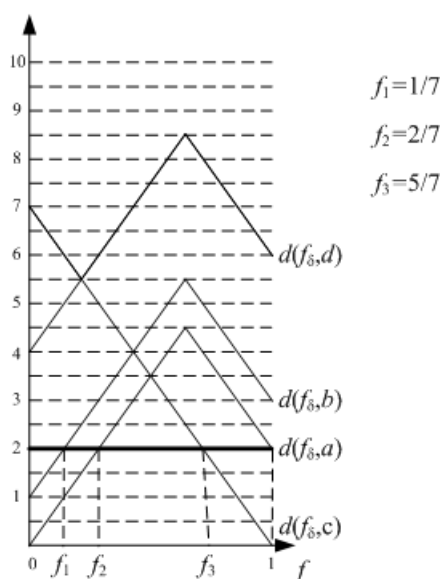
$$d(f_{(i,j)}, k) \leq \Delta.$$

Определение. Непустое множество точек графа назовем *областью*.

Определение. Будем говорить, что вершина k достижима из области D в пределах расстояния Δ , если вершина k достижима из любой точки этой области в пределах расстояния Δ .

Определить достижимость вершины из точки можно графически, с помощью функций расстояния точка-вершина.

Пример. Рассмотрим граф из предыдущих примеров. Для него для точек ребра $\delta = (a, c)$ были найдены расстояния точка-вершина до всех вершин графа.



На графике жирной линией обозначен уровень расстояния $\Delta = 2$. Через f_1, f_2, f_3 обозначены точки пересечения этой прямой с графиками функций $d(f_\delta, k)$ для различных вершин k . Точка $f_1 = 1/7$ является точкой пересечения прямой с функцией $d(f_\delta, b)$. Из графика видно, что на участке $[0, f_1]$ графики функций $d(f_\delta, a)$ и $d(f_\delta, b)$ лежат не выше прямой, а это означает, что вершины a, b достижимы из области $[0_\delta, (1/7)_\delta] = [a, (1/7)_\delta]$ в пределах расстояния 2 .

Далее можно выделить область $(f_1, f_2]$, где $f_2 = 2/7$ является точкой пересечения прямой с функцией $d(f_\delta, a)$. Из графика видно, что на участке $(f_1, f_2]$ график функции $d(f_\delta, a)$ лежит не выше прямой, а это означает, что вершина a достижима из области $((1/7)_\delta, (2/7)_\delta]$ в пределах расстояния 2 .

Из области (f_2, f_3) , где $f_3 = 5/7$ является точкой пересечения прямой с функцией $d(f_\delta, c)$ недостижима в пределах расстояния 2 ни одна вершина. Из области $[f_3, 1) = [(5/7)_\delta, 1_\delta) = [(5/7)_\delta, c)$ достижима в пределах расстояния 2 вершина c . Из точки 1_δ , т.е. из вершины c , достижимы в пределах расстояния 2 вершины a и d .

Далее для краткости мы будем говорить, что вершина достижима из точки, подразумевая, что она достижима в пределах заданного расстояния.

Существует две классические постановки задачи поиска кратного абсолютного центра графа.

Задача 1. Задано расстояние Δ . Найти множество точек графа X_r наименьшей мощности r , для которого

$$\max_{k \in V} d(X_r, k) \leq \Delta.$$

Задача 2. Задана мощность множества r . Найти множество точек графа X_r мощности, не большей r , для которого минимальна величина

$$\max_{k \in V} d(X_r, k).$$

Для описания отношения достижимости вершин из точек построим булеву матрицу следующего вида. Строкам сопоставим области точек, столбцам – вершины графа. На пересечении строки и столбца ставится 1, если вершина, сопоставленная столбцу, достижима из области, сопоставленной строке, в пределах расстояния Δ . Будем называть эту матрицу Δ -матрицей.

Процесс выделения областей можно ускорить, используя следующее утверждение.

Утверждение 4. Если вершина k достижима в пределах расстояния Δ из какой-либо внутренней точки ориентированного ребра (i, j) , то она достижима в пределах этого расстояния и из вершины j .

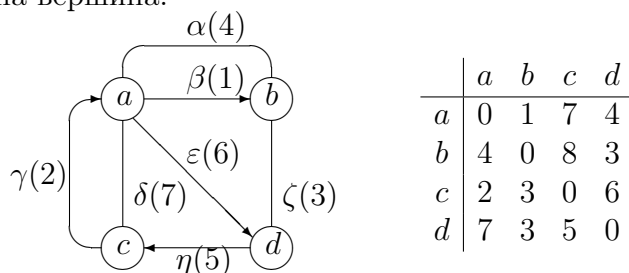
Доказательство. Согласно формуле (3), для точек ориентированного ребра (i, j) расстояние точка-вершина имеет вид

$$d(f_{(i,j)}, k) = (1 - f)c_{i,j} + d(j, k).$$

Отсюда следует, что если $d(f_{(i,j)}, k) \leq \Delta$, то и $d(j, k) \leq \Delta$.

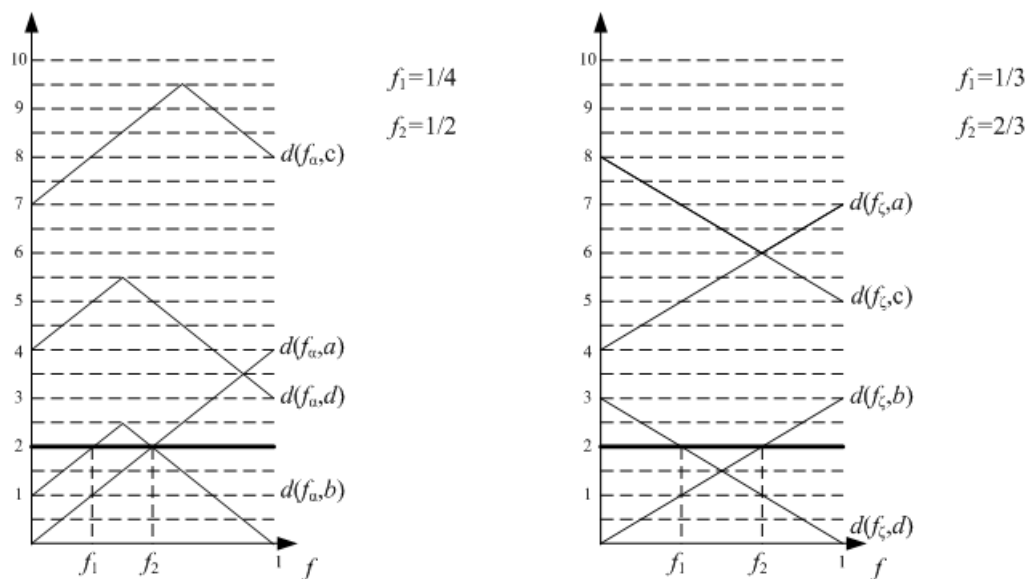
Таким образом, выделение областей можно проводить только для неориентированных ребер, а также рассматривать в качестве областей концы ориентированных ребер. Достижимость вершин из вершин определяется по матрице расстояний вершина-вершина сравнением величины $d(j, k)$ с расстоянием Δ .

Пример. Рассмотрим граф из предыдущих примеров и его матрицу расстояний вершина-вершина.



Из матрицы видно, что из вершины a в пределах расстояния $\Delta = 2$ достижимы

a и c , из b – вершина b , из c – вершины a и c , из d – вершина d . Ранее были выделены области достижимости в пределах расстояния $\Delta = 2$ для ребра $\delta = (a, c)$. Рассмотрим остальные неориентированные ребра $\alpha = (a, b)$ и $\zeta = (b, d)$.



Как видно из графика, из области $[0_\alpha, (1/4)_\alpha] = [a, (1/4)_\alpha]$ достижимы вершины a и b , из области $[(1/4)_\alpha, (1/2)_\alpha]$ – вершина a , из области $[(1/2)_\alpha, 1_\alpha] = [(1/2)_\alpha, b]$ – вершина b , из области $[0_\zeta, (1/3)_\zeta] = [b, (1/3)_\zeta]$ – вершина b , из области $[(1/3)_\zeta, (2/3)_\zeta]$ – вершины b и d , из области $[(2/3)_\zeta, 1_\zeta] = [(2/3)_\zeta, d]$ – вершина d . Сведем полученные результаты в Δ -матрицу. Для удобства нули в матрице опущены. Строки, соответствующие вершинам a, b и d не выписаны, поскольку эти вершины входят в состав областей.

	a	b	c	d
c	1		1	
$[a, (1/7)_\delta]$	1	1		
$[(1/7)_\delta, (2/7)_\delta]$	1			
$[(5/7)_\delta, c]$			1	
$[a, (1/4)_\alpha]$	1	1		
$[(1/4)_\alpha, (1/2)_\alpha]$	1			
$[(1/2)_\alpha, b]$		1		
$[b, (1/3)_\zeta]$		1		
$[(1/3)_\zeta, (2/3)_\zeta]$		1		1
$[(2/3)_\zeta, d]$				1

Задача о поиска абсолютного кратного центра минимальной мощности может быть сведена к задаче поиска кратчайшего покрытия Δ -матрицы. Постановка задачи и некоторые методы ее решения рассмотрены в Приложении.

Кратчайшее покрытие Δ -матрицы дает минимальное число областей, из которых достижимы в совокупности все вершины графа в пределах заданного расстояния. Выбрав по точке в каждой такой области, мы получим решение задачи о кратном абсолютном центре в первой постановке.

Пример. Рассмотрим матрицу из предыдущего примера. Из одинаковых строк оставим одну, а также применим, пока это возможно, правило строки-предшественницы. Сокращенная матрица имеет вид

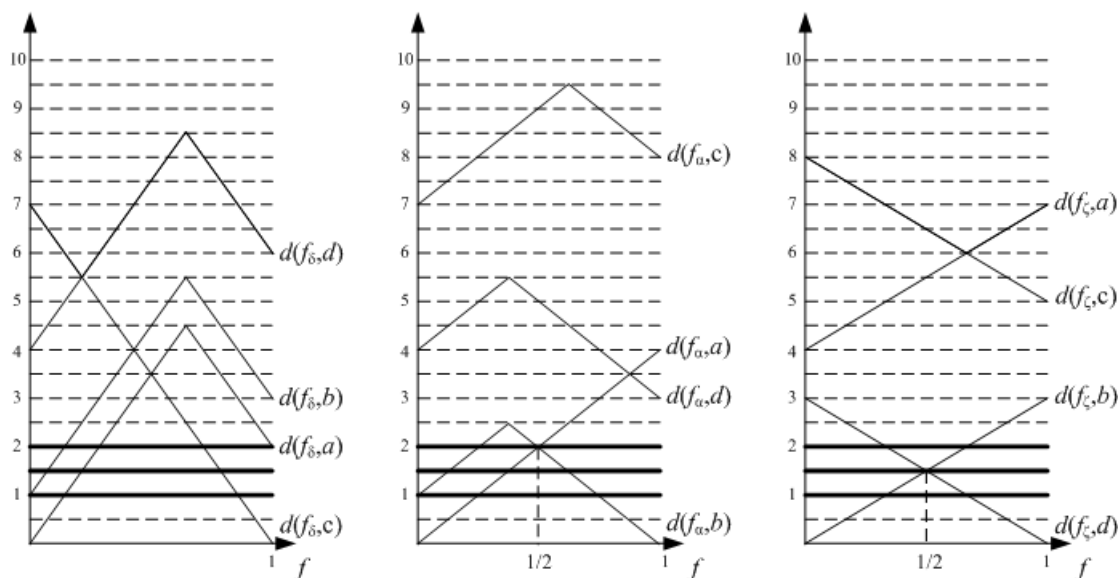
	a	b	c	d
c	1		1	
$[a, (1/7)_\delta]$	1	1		
$[(1/3)_\zeta, (2/3)_\zeta]$		1		1

Ядерные строки (первая и третья) образуют кратчайшее покрытие. Значит, кратным абсолютным центром является множество из двух точек

$$X_2 = \{c, f_\zeta\}, \quad f_\zeta \in [(1/3)_\zeta, (2/3)_\zeta].$$

Задачу поиска кратного абсолютного центра во второй постановке также можно свести к задаче поиска покрытия булевой матрицы. Основная идея такова: строится Δ -матрица для некоторого расстояния Δ и ищется ее кратчайшее покрытие. Если мощность данного покрытия больше, чем заданное число r , то увеличивается величина Δ и снова строится Δ -матрица и ищется ее кратчайшее покрытие. Решение найдено, когда мощность полученного покрытия становится меньше или равной заданного числа r . Очередную величину Δ можно находить как ординату точки пересечения функций расстояния точка-вершина для точек одного неориентированного ребра между собой, а также с прямыми $f = 0$ и $f = 1$.

Пример. Рассмотрим граф из предыдущих примеров и найдем для него кратный абсолютный центр мощности $r = 2$. Для этого графа ранее были найдены расстояния точка-вершина для точек всех неориентированных ребер.



Выбираем сначала $\Delta = 1$, поскольку функции $d(f_\delta, b)$ и $d(f_\alpha, b)$ пересекаются с

прямой $f = 0$ в точке $(0, 1)$. Составим для этого значения Δ -матрицу

	a	b	c	d
a	1	1		
b		1		
c			1	
d				1

Применим правило строки-предшественницы и удалим строку b .

	a	b	c	d
a	1	1		
c			1	
d				1

Кратчайшим покрытием этой матрицы является множество $\{a, c, d\}$ мощности 3. Увеличиваем Δ . Функции $d(f_\zeta, b)$ и $d(f_\zeta, c)$ пересекаются в точке $(1/2, 3/2)$, поэтому полагаем $\Delta = 3/2$. Составим для этого значения Δ -матрицу, добавив в предыдущую матрицу строку $(1/2)_\zeta$

	a	b	c	d
a	1	1		
b		1		
c			1	
d				1
$(1/2)_\zeta$	1			1

Применим правило строки-предшественницы и удалим строки b и d .

	a	b	c	d
a	1	1		
c			1	
$(1/2)_\zeta$	1			1

Кратчайшим покрытием этой матрицы является множество $\{a, c, (1/2)_\zeta\}$ мощности 3. Увеличиваем Δ . Функция $d(f_\delta, c)$ пересекается с прямой $f = 1$ в точке $(1, 2)$, а $d(f_\alpha, a)$ и $d(f_\alpha, b)$ пересекаются в точке $(1/2, 2)$, поэтому полагаем $\Delta = 2$. Составим для этого значения Δ -матрицу, изменив в предыдущей матрице строку c , поскольку $c = 1\delta$, и добавив строку $(1/2)_\alpha$

	a	b	c	d
a	1	1		
b		1		
c	1		1	
d				1
$(1/2)_\zeta$		1		1
$(1/2)_\alpha$	1	1		

Применим правило строки-предшественницы и удалим строки a , b и d .

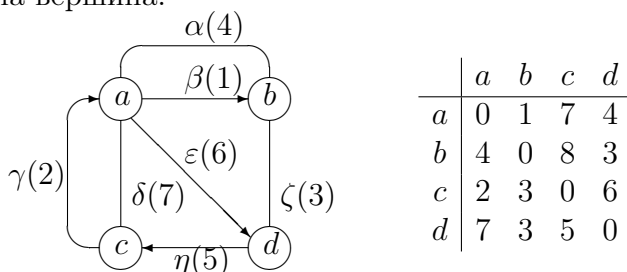
	a	b	c	d
c	1	1		
$(1/2)_\zeta$		1	1	
$(1/2)_\alpha$	1	1		

Ядерные строки c и $(1/2)_\zeta$ образуют кратчайшее покрытие этой матрицы мощности 2. Следовательно, найден абсолютный кратный центр

$$X_2 = \{c, (1/2)_\zeta\}.$$

Рассмотрим теперь модификации алгоритмов поиска кратного абсолютного центра. Если требуется найти кратный центр, то задача существенно упрощается: вместо областей рассматриваются только вершины графа.

Пример. Рассмотрим граф из предыдущих примеров и его матрицу расстояний вершина-вершина.



Пусть требуется найти кратный центр для расстояния $\Delta = 2$. В Δ -матрице элемент $\Delta_{i,j}$ полагаем равным единице, если и только если $d(i, j) \leq \Delta$.

	a	b	c	d
a	1	1		
b		1		
c	1		1	
d				1

Применим правило строки-предшественницы и удалим строку b .

	a	b	c	d
a	1	1		
c	1		1	
d				1

Кратчайшее покрытие образуют ядерные строки a , c и d , значит, кратный центр имеет вид

$$X_r = \{a, c, d\}.$$

Обратим внимание, что мощность кратного абсолютного центра получилась равной двум, т.е. меньше мощности кратного центра.

Найдем теперь кратный центр мощности $r = 2$. Из предыдущего примера ясно, что расстояние Δ должно быть больше двух. Увеличиваем расстояние Δ

следующим образом: находим в матрице расстояний вершина-вершина минимальный элемент $d(i, j) > 2$ и полагаем Δ равным этому элементу. В данном случае $\Delta = d(b, d) = 3$. Затем строим Δ -матрицу для значения 3.

	a	b	c	d
a	1	1		
b		1		1
c	1	1	1	
d		1		1

Применим правило строки-предшественницы и удалим строки a и b .

	a	b	c	d
c	1	1	1	
d		1		1

Ядерные строки c и d образуют кратчайшее покрытие матрицы, значит, найден кратный центр мощности 2

$$X_2 = \{c, d\}.$$

Обратим внимание, что от абсолютного кратного центра вершины достижимы в пределах расстояния, меньшего 3, а именно, в пределах расстояния 2.

Поиск кратного главного центра и кратного главного абсолютного центра осуществляется аналогично, но вместо расстояний вершина-вершина и точка-вершина используются расстояния вершина-ребро и точка-ребро.

С ростом r растет число возможных множеств X_r , для которых требуется вычисление максимальных расстояний, поэтому метод перебора становится слишком трудоемким. Данные задачи для больших размерностей графа требуют приближенных алгоритмов решения с относительно небольшими вычислительными затратами.

4.6 Поиск кратных медиан

Рассмотрим задачу поиска кратных медиан. Как было показано в подразделе 4.3, в качестве абсолютных медиан графа можно рассматривать только его вершины. Этот результат распространяется и на кратные медианы. Доказательство практически идентично доказательству в случае абсолютной медианы, и поэтому здесь не приводится. Таким образом, задача поиска кратной абсолютной медианы сводится к задаче поиска кратной медианы.

Можно сформулировать две постановки задачи поиска кратной медианы.

Задача 1. Задана мощность множества r . Найти множество вершин графа X_r мощности r , для которого минимальна величина

$$\sum_{k \in V} d(X_r, k).$$

Задача 2. Задано расстояние Δ . Найти множество вершин графа X_r наименьшей мощности r , для которого

$$\sum_{k \in V} d(X_r, k) \leq \Delta.$$

Рассмотрим сначала задачу 1. Она может быть поставлена как задача дискретного линейного программирования.

Будем называть вершины, принадлежащие кратной медиане X_r , *медианными вершинами*, а остальные вершины – *немедианными*. Будем говорить, что вершина j *прикреплена к вершине i* , если $i \in X_r$, и $d(X_r, j) = d(i, j)$. Другими словами, вершина i является медианной вершиной, и расстояние от нее до вершины j минимально среди всех расстояний от медианных вершин до вершины j . Очевидно, что если i является медианной вершиной, то она прикрепляется сама к себе.

Введем переменные $\xi_{i,j} \in \{0, 1\}$, описывающие прикрепление вершин

$$\xi_{i,j} = \begin{cases} 1, & \text{если вершина } j \text{ прикреплена к вершине } i; \\ 0, & \text{иначе.} \end{cases}$$

Задача поиска кратной медианы может быть поставлена следующим образом

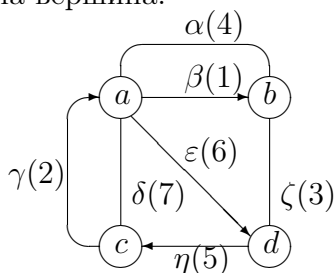
$$\begin{aligned} \sum_{i \in V} \sum_{j \in V} d(i, j) \xi_{i,j} &\rightarrow \min; \\ \sum_{i \in V} \xi_{i,j} &= 1, \quad \forall j \in V; \\ \sum_{i \in V} \xi_{i,i} &= r; \\ \xi_{i,j} &\leq \xi_{i,i}, \quad \forall i \in V, j \in V; \\ \xi_{i,j} &\in \{0, 1\}, \quad \forall i \in V, j \in V. \end{aligned}$$

Для решение подобных задач успешно применяется метод ветвей и границ. Идея метода состоит в следующем: вводится дополнительное ограничение на одну из переменных, и задача делится на две подзадачи. В одной из них переменная удовлетворяет условию, в другой – нет. Затем подзадачи решаются аналогично.

С каждой задачей, кроме того, связывается оценочная функция \underline{C} , которая оценивает снизу стоимость решения. Если оценочная функция превышает минимальную стоимость решения C^* , то задача отбрасывается. В качестве C^* можно сначала взять стоимость произвольно найденного решения, или даже $+\infty$. Если получено решение, стоимость которого меньше минимальной стоимости ($\underline{C} < C^*$), то минимальная стоимость заменяется стоимостью полученного решения.

Рассмотрим метод ветвей и границ в применении к задаче поиска кратной медианы. Построим матрицу M размерности $p \times p$, в которой столбец j соответствует вершине j , и в столбце расположены вершины по неубыванию расстояния до вершины j , т.е. по предпочтению прикрепления к ним вершины j . Иными словами, если $m_{i,j} = k$, то найдется $i - 1$ вершина, расстояние от которых до j не больше $d(k, j)$, и $p - i$ вершин, расстояние от которых до j не меньше $d(k, j)$. Очевидно, что $m_{1,j} = j$.

Пример. Рассмотрим граф из предыдущих примеров и его матрицу расстояний вершина-вершина.



	a	b	c	d
a	0	1	7	4
b	4	0	8	3
c	2	3	0	6
d	7	3	5	0

Матрица M имеет вид

$$M = \begin{array}{c|cccc} & a & b & c & d \\ \hline 1 & a & b & c & d \\ 2 & c & a & d & b \\ 3 & b & c & a & a \\ 4 & d & d & b & c \end{array}$$

Для каждой вершины j определим индекс k_j , который равен номеру строки матрицы M . На пересечении этой строки и столбца j находится вершина $x_j = m_{k_j, j}$, которая в данной задаче является наилучшим вариантом прикрепления для вершины j . Для стартовой задачи для всех вершин $k_j = 1$, $x_j = j$, и расстояние $d(x_j, j) = 0$.

Определим теперь оценки качества решения. В качестве начальной оценки минимальной стоимости решения C^* логично взять суммарное расстояние от медианы графа до всех его вершин, т.е. минимальную из сумм строк матрицы расстояний вершина-вершина. Оценочная функция текущего решения \underline{C} – это сумма расстояний $d(x_j, j)$, поскольку вершина x_j является на данный момент лучшим вариантом прикрепления для вершины j :

$$\underline{C} = \sum_{j \in V} d(x_j, j).$$

Деление на две подзадачи происходит по одной из переменных $\xi_{x_j, j}$: в одной из подзадач она полагается равной 1, в другой – 0. Если $\xi_{x_j, j} = 1$, то x_j является медианной вершиной. Если при этом $x_j \neq j$, то j – немедианная вершина. Если $\xi_{x_j, j} = 0$, то, поскольку вершина x_j является в текущей задаче наилучшим вариантом прикрепления для вершины j , то x_j является немедианной вершиной. При этом возможны два варианта: $x_j = j$ (в этом случае, очевидно, j – немедианная вершина), и $x_j \neq j$. Заметим, что в случае, когда $x_j \neq j$, деление по переменной $\xi_{j, j}$ было проведено ранее, и в текущей задаче $\xi_{j, j} = 0$, т.е. и вершина j является немедианной.

Если некоторая вершина i классифицирована как медианная, и существует вершина t , для которой i является текущим наилучшим вариантом прикрепления, то вершина t окончательно прикрепляется к вершине i , т.е. переменная $\xi_{i, t} = 1$. Если некоторая вершина i классифицирована как немедианная, и для некоторой вершины t в текущей задаче вершина i является наилучшим вариантом прикрепления, то для вершины t ищется следующий вариант прикрепления, и оценочная функция решения \underline{C} пересчитывается.

Мощность множества медианных вершин в любой задаче не должна превышать r , а мощность множества немедианных вершин – $p - r$. Для текущей задачи введем следующие обозначения:

- S^+ – множество медианных вершин;
- S^- – множество немедианных вершин;
- F – множество неприкрепленных вершин.

Все медианные вершины являются прикрепленными сами к себе, т.е. $F \cap S^+ = \emptyset$. Немедианные вершины могут быть как прикрепленными, так и неприкрепленными, т.е. в общем случае $F \cap S^- \neq \emptyset$.

Алгоритм поиска кратной медианы

Начало. Рассмотрим стартовую постановку задачи. Положим $S^+ = \emptyset$, $S^- = \emptyset$, $F = \emptyset$. Положим для всех вершин $k_j = 1$, т.е. $x_j = j$. Запишем стартовую задачу в список задач. В качестве оценки сверху C^* возьмем суммарное расстояние от медианы графа до всех его вершин, т.е. минимальную из сумм элементов строк матрицы расстояний вершина-вершина. Оценка качества решения в стартовой задаче $\underline{C} = 0$. Занесем стартовую задачу в список задач. Требуется найти медиану мощности $r > 1$.

Шаг 1. Если список задач пуст, идем на конец. Иначе выберем из списка задачу A . Выполним процедуру ветвления. Выберем из множества $F(A)$ очередную вершину j , с которой связан индекс k_j и вариант прикрепления $x_j = m_{k_j, j}$. Произведем ветвление по переменной $\xi_{x_j, j}$. В задаче A' положим $\xi_{x_j, j} = 1$, в задаче A'' положим $\xi_{x_j, j} = 0$. Оценки качества решения $\underline{C}(A') = \underline{C}(A'') = \underline{C}(A)$.

Шаг 2. Рассмотрим задачу A' . Сначала положим $S^+(A') = S^+(A)$, $S^-(A') = S^-(A)$, $F(A') = F(A)$.

Шаг 3. Добавим вершину x_j в множество медианных вершин и удалим из множества неприкрепленных, т.е. положим $S^+(A') = S^+(A) \cup \{x_j\}$, $F(A') = F(A) \setminus \{x_j\}$. Удалим вершину j из множества неприкрепленных вершин, т.е. положим $F(A') = F(A) \setminus \{j\}$. Если $|S^+(A')| < r$, идем на шаг 6.

Шаг 4. Поскольку $|S^+(A')| = r$, то очередное медианное множество построено. Положим $S^-(A') = S^-(A) \cup F(A')$. Рассмотрим очередную еще не прикрепленную вершину $t \in F(A')$. Если такой вершины нет, перейдем на шаг 5. Найдем для вершины t наилучший вариант прикрепления из множества $S^+(A')$: будем увеличивать индекс k_t до тех пор, пока $x_t = m_{k_t, t}$ принадлежит множеству $S^-(A')$. Положим $\xi_{x_t, t} = 1$, удалим вершину t из множества неприкрепленных вершин, т.е. положим $F(A') = F(A) \setminus \{t\}$, и повторим шаг 4.

Шаг 5. Вычислим оценку качества решения: $\underline{C}(A') = \sum_{t \in S^-(A')} d(x_t, t)$. Сравним оценку качества решения с оценкой C^* : если $\underline{C}(A') < C^*$, то запомним решение $S^+(A')$, положим $C^* = \underline{C}(A')$ и удалим из списка задач все задачи B , для которых $\underline{C}(B) < C^*$. Перейдем на шаг 10.

Шаг 6. Рассмотрим очередную еще не прикрепленную вершину $t \in F(A) \cap S^-(A')$. Если такой вершины нет, идем на шаг 7. Если $x_t = x_j$, прикрепим вершину t к вершине x_j , т.е. положим $\xi_{x_j, t} = 1$ и удалим вершину t из множества $F(A')$, т.е. положим $F(A') = F(A) \setminus \{t\}$. Повторим шаг 6.

Шаг 7. Если $j \neq x_j$ и $j \notin S^-(A')$, добавляем вершину j в множество немедианных вершин, т.е. положим $S^-(A') = S^-(A) \cup \{j\}$ и перейдем на шаг 8. Иначе занесем задачу A' в список задач и перейдем на шаг 10.

Шаг 8. Рассмотрим очередную еще не прикрепленную вершину $t \in F(A) \cap S^-(A')$. Если такой вершины нет, идем на шаг 9. Если $x_t = j$, найдем для вершины t очередной вариант прикрепления из множества $S^+(A) \cup F(A')$: будем увеличивать индекс k_t до тех пор, пока $x_t = m_{k_t, t}$ принадлежит множеству $S^-(A')$. В результате возможны два варианта. Если $x_t \in S^+(A')$, то окончательно прикрепим вершину t к вершине x_t , т.е. положим $\xi_{x_t, t} = 1$ и удалим вершину t из множества неприкрепленных вершин, т.е. положим $F(A') = F(A) \setminus \{t\}$. Если $x_t \in F(A')$, то будем считать x_t текущим наилучшим вариантом прикрепления для вершины t .

Пересчитаем оценку качества решения: $\underline{C}(A') = \underline{C}(A) - d(j, t) + d(x_t, t)$. Повторим шаг 8.

Шаг 9. Сравним оценку качества решения с оценкой C^* : если $\underline{C} < C^*$, то задачу A' занесем в список задач.

Шаг 10. Рассмотрим задачу A'' . Сначала положим $S^+(A'') = S^+(A)$, $S^-(A'') = S^-(A)$, $F(A'') = F(A)$.

Шаг 11. Добавим вершину x_j в множество немедианных вершин, т.е. положим $S^-(A'') = S^-(A'') \cup \{x_j\}$.

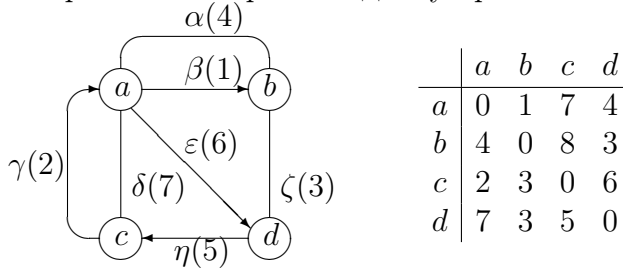
Шаг 12. Рассмотрим очередную еще не прикрепленную вершину $t \in F(A'') \cap S^-(A'')$, для которой $x_t = x_j$. Если такой вершины нет, перейдем на шаг 13. Найдем для вершины t очередной вариант прикрепления из множества $S^+(A'') \cup F(A'')$: будем увеличивать индекс k_t до тех пор, пока $x_t = m_{k_t, t}$ принадлежит множеству $S^-(A'')$. В результате возможны два варианта. Если $x_t \in S^+(A'')$, то окончательно прикрепим вершину t к вершине x_t , т.е. положим $\xi_{x_t, t} = 1$ и удалим вершину t из множества неприкрепленных вершин, т.е. положим $F(A'') = F(A'') \setminus \{t\}$. Если $x_t \in F(A'')$, то будем считать x_t текущим наилучшим вариантом прикрепления для вершины t . Пересчитаем оценку качества решения: $\underline{C}(A'') = \underline{C}(A'') - d(x_j, t) + d(x_t, t)$. Повторим шаг 12.

Шаг 13. Если $\underline{C}(A'') \geq C^*$, не будем дальше рассматривать задачу A'' и перейдем на шаг 1. Иначе если $|S^-(A'')| < p - r$, добавим задачу A'' в список задач и перейдем на шаг 1.

Шаг 14. Поскольку $|S^-(A'')| = p - r$, и $\underline{C}(A'') < C^*$ то очередное медианное множество построено. Положим $S^+(A'') = V \setminus S^-(A'')$. Для каждой вершины t из множества $F(A'')$ положим $\xi_{x_t, t} = 1$ и удалим вершину t из множества неприкрепленных вершин, т.е. положим $F(A') = F(A') \setminus \{t\}$. Запомним решение $S^+(A'')$ и положим $C^* = \underline{C}(A'')$. Удалим из списка задач все задачи B , для которых $\underline{C}(B) < C^*$. Перейдем на шаг 1.

Конец. Кратная медиана построена.

Пример. Рассмотрим граф из предыдущих примеров и его матрицу расстояний вершина-вершина. Построим медиану кратности $r = 2$.



Матрица M имеет вид (жирным шрифтом выделим наилучшие варианты прикрепления вершин)

$$M(A) = \begin{array}{c|cccc} & a & b & c & d \\ \hline 1 & \mathbf{a} & \mathbf{b} & \mathbf{c} & \mathbf{d} \\ 2 & c & a & d & b \\ 3 & b & c & a & a \\ 4 & d & d & b & c \end{array}$$

Для стартовой задачи A положим $S^+(A) = \emptyset$, $S^-(A) = \emptyset$, $F(A) = \{a, b, c, d\}$.

Оценка качества решения $C^* = 11$, оценка качества стартовой задачи $\underline{C}(A) = 0$. Список состоит из задачи A .

Шаг 1. Выбираем задачу A из списка. Выполняем процедуру ветвления. Выбираем из множества $F(A)$ вершину a , с которой связан индекс $k_a = 1$ и вариант прикрепления $x_a = a$ (т.е. в обозначениях алгоритма $j = a$, $x_j = a$). Производим ветвление по переменной $\xi_{a,a}$. В задаче B полагаем $\xi_{a,a} = 1$, в задаче C полагаем $\xi_{a,a} = 0$. Записываем оценки качества решения $\underline{C}(B) = \underline{C}(C) = \underline{C}(A) = 0$.

Шаг 2. Рассматриваем задачу B . Сначала полагаем $S^+(B) = S^+(A) = \emptyset$, $S^-(B) = S^-(A) = \emptyset$, $F(B) = F(A) = \{a, b, c, d\}$.

Шаг 3. Добавляем вершину a в множество медианных вершин и удаляем из множества неприкрепленных, т.е. полагаем $S^+(B) = \{a\}$, $F(B) = \{b, c, d\}$. Поскольку $|S^+(B)| < r$, идем на шаг 6.

Шаг 6. Поскольку $S^-(B) = \emptyset$, идем на шаг 7.

Шаг 7. Поскольку $a = x_a$, заносим задачу B в список задач. Список состоит из задачи B . Задача B имеет вид:

$$\begin{array}{l}
 S^+(B) = \{a\}; \\
 S^-(B) = \emptyset; \\
 F(B) = \{b, c, d\}; \\
 \xi_{a,a} = 1.
 \end{array}
 \quad
 M(B) =
 \begin{array}{c|cccc}
 & a & b & c & d \\
 \hline
 1 & \mathbf{a} & \mathbf{b} & \mathbf{c} & \mathbf{d} \\
 2 & c & a & d & b \\
 3 & b & c & a & a \\
 4 & d & d & b & c
 \end{array}
 \quad
 \underline{C}(B) = 0.$$

Идем на шаг 10.

Шаг 10. Рассматриваем задачу C . Сначала полагаем $S^+(C) = S^+(A) = \emptyset$, $S^-(C) = S^-(A) = \emptyset$, $F(C) = F(A) = \{a, b, c, d\}$.

Шаг 11. Добавляем вершину $x_a = a$ в множество немедианных вершин, т.е. полагаем $S^-(C) = S^-(C) \cup \{a\} = \{a\}$.

Шаг 12. Рассматриваем еще не прикрепленную вершину $a \in S^-(C) \cap F(C)$. Находим для вершины a наилучший вариант прикрепления к вершине из множества $S^+(C) \cup F(C)$, это вершина c . Матрица M имеет вид

$$M(C) =
 \begin{array}{c|cccc}
 & a & b & c & d \\
 \hline
 1 & a & \mathbf{b} & \mathbf{c} & \mathbf{d} \\
 2 & \mathbf{c} & a & d & b \\
 3 & b & c & a & a \\
 4 & d & d & b & c
 \end{array}$$

Поскольку $c \notin S^+(C)$, вершина a остается неприкрепленной. Пересчитываем оценку качества решения $\underline{C}(C) = \underline{C}(C) - d(a, a) + d(c, a) = 0 - 0 + 2 = 2$. Поскольку множество $S^-(C) \cap F(C)$ просмотрено, идем на шаг 13.

Шаг 13. Поскольку $\underline{C}(C) < C^*$, и $|S^-(C)| < p - r$, заносим задачу C в список задач. Список состоит из задач B, C . Задача C имеет вид:

$$\begin{array}{l}
 S^+(C) = \emptyset; \\
 S^-(C) = \{a\}; \\
 F(C) = \{a, b, c, d\}; \\
 \xi_{a,a} = 0.
 \end{array}
 \quad
 M(C) =
 \begin{array}{c|cccc}
 & a & b & c & d \\
 \hline
 1 & a & \mathbf{b} & \mathbf{c} & \mathbf{d} \\
 2 & \mathbf{c} & a & d & b \\
 3 & b & c & a & a \\
 4 & d & d & b & c
 \end{array}
 \quad
 \underline{C}(C) = 2.$$

Шаг 1. Выбираем задачу B из списка. Задача B имеет вид:

$$\begin{array}{l}
 S^+(B) = \{a\}; \\
 S^-(B) = \emptyset; \\
 F(B) = \{b, c, d\}; \\
 \xi_{a,a} = 1.
 \end{array}
 \quad
 M(B) =
 \begin{array}{c|cccc}
 & a & b & c & d \\
 \hline
 1 & \mathbf{a} & \mathbf{b} & \mathbf{c} & \mathbf{d} \\
 2 & c & a & d & b \\
 3 & b & c & a & a \\
 4 & d & d & b & c
 \end{array}
 \quad
 \underline{C}(B) = 0.$$

Выбираем из множества $F(B)$ вершину b , с которой связан индекс $k_b = 1$ и вариант прикрепления $x_b = b$. Производим ветвление по переменной $\xi_{b,b}$. В задаче D полагаем $\xi_{b,b} = 1$, в задаче E полагаем $\xi_{b,b} = 0$. Оценки качества решения $\underline{C}(D) = \underline{C}(E) = \underline{C}(B) = 0$.

Шаг 2. Рассматриваем задачу D . Сначала полагаем $S^+(D) = S^+(B) = \{a\}$, $S^-(D) = S^-(B) = \emptyset$, $F(D) = F(B) = \{b, c, d\}$.

Шаг 3. Добавляем вершину b в множество медианных вершин и удаляем из множества неприкрепленных, т.е. полагаем $S^+(D) = \{a, b\}$, $F(D) = \{c, d\}$. Поскольку $|S^+(D)| = r$, идем на шаг 4.

Шаг 4. Поскольку $\underline{C}(D) < C^*$ и $|S^+(D)| = r$, то очередное медианное множество построено. Полагаем $S^-(D) = S^-(D) \cup F(D) = \{c, d\}$ и найдем для еще не прикрепленных вершин наилучшие варианты прикрепления к вершинам из множества $S^+(A')$: $x_c = a$, $x_d = b$. Матрица M имеет вид

$$M(D) =
 \begin{array}{c|cccc}
 & a & b & c & d \\
 \hline
 1 & \mathbf{a} & \mathbf{b} & c & d \\
 2 & c & a & d & \mathbf{b} \\
 3 & b & c & \mathbf{a} & a \\
 4 & d & d & b & c
 \end{array}$$

Полагаем $\xi_{a,c} = 1$, $\xi_{b,d} = 1$.

Шаг 5. Пересчитываем оценку качества решения: $\underline{C}(D) = d(a, c) + d(b, d) = 7 + 3 = 10$. Поскольку $\underline{C}(D) < C^*$, запоминаем текущее множество медианных вершин $S^+(D) = \{a, b\}$ и новую оценку качества решения $C^* = 10$. Список состоит из задачи C , для нее $\underline{C}(C) < C^*$, оставляем ее в списке. Идем на шаг 10.

Шаг 10. Рассматриваем задачу E . Сначала полагаем $S^+(E) = S^+(B) = \{a\}$, $S^-(E) = S^-(B) = \emptyset$, $F(E) = F(B) = \{b, c, d\}$.

Шаг 11. Добавляем вершину b в множество немедианных вершин, т.е. полагаем $S^-(E) = \{b\}$.

Шаг 12. Рассматриваем очередную еще не прикрепленную вершину $b \in F(E) \cap S^-(E)$. Поскольку $x_b = b$, находим для вершины b очередной вариант прикрепления из столбца b матрицы M : $x_b = a$. Матрица M имеет вид

$$M(E) =
 \begin{array}{c|cccc}
 & a & b & c & d \\
 \hline
 1 & \mathbf{a} & b & c & \mathbf{d} \\
 2 & c & \mathbf{a} & d & b \\
 3 & b & c & a & a \\
 4 & d & d & b & c
 \end{array}$$

Поскольку $a \in S^+(E)$, то окончательно прикрепляем вершину b к вершине a , т.е. полагаем $\xi_{a,b} = 1$. Удаляем вершину b из множества $F(E)$, теперь $F(E) = \{c, d\}$.

Пересчитываем оценку качества решения: $\underline{C}(E) = \underline{C}(E) - d(b, b) + d(a, b) = 0 - 0 + 1 = 1$. Поскольку множество $F(E) \cap S^-(E)$ просмотрено, идем на шаг 13.

Шаг 13. Поскольку $\underline{C}(E) < C^*$, и $|S^-(E)| < p - r$, заносим задачу E в список задач. Список состоит из задач C, E . Задача E имеет вид:

$$\begin{array}{l} S^+(E) = \{a\}; \\ S^-(E) = \{b\}; \\ F(E) = \{c, d\}; \\ \xi_{a,a} = 1; \\ \xi_{b,b} = 0; \\ \xi_{a,b} = 1; \end{array} \quad M(E) = \begin{array}{c|cccc} & a & b & c & d \\ \hline 1 & \mathbf{a} & \mathbf{b} & \mathbf{c} & \mathbf{d} \\ 2 & c & \mathbf{a} & d & b \\ 3 & b & c & a & a \\ 4 & d & d & b & c \end{array}; \quad \underline{C}(E) = 1.$$

Идем на шаг 1.

Шаг 1. Выбираем задачу C из списка. Задача C имеет вид:

$$\begin{array}{l} S^+(C) = \emptyset; \\ S^-(C) = \{a\}; \\ F(C) = \{a, b, c, d\}; \\ \xi_{a,a} = 0. \end{array} \quad M(C) = \begin{array}{c|cccc} & a & b & c & d \\ \hline 1 & a & \mathbf{b} & \mathbf{c} & \mathbf{d} \\ 2 & \mathbf{c} & a & d & b \\ 3 & b & c & a & a \\ 4 & d & d & b & c \end{array}; \quad \underline{C}(C) = 2.$$

Выбираем из множества $F(C)$ вершину a , с которой связан индекс $k_a = 2$ и вариант прикрепления $x_a = c$. Производим ветвление по переменной $\xi_{c,a}$. В задаче F полагаем $\xi_{c,a} = 1$, в задаче G полагаем $\xi_{c,a} = 0$. Оценки качества решения $\underline{C}(F) = \underline{C}(G) = \underline{C}(C) = 2$.

Шаг 2. Рассматриваем задачу F . Сначала полагаем $S^+(F) = S^+(C) = \emptyset$, $S^-(F) = S^-(C) = \{a\}$, $F(F) = F(C) = \{a, b, c, d\}$.

Шаг 3. Добавляем вершину c в множество медианных вершин и удаляем из множества неприкрепленных, т.е. полагаем $S^+(F) = \{c\}$, $F(F) = \{a, b, d\}$. Удаляем вершину a из множества неприкрепленных вершин, т.е. полагаем $F(F) = \{b, d\}$. Поскольку $|S^+(F)| < r$, идем на шаг 6.

Шаг 6. Поскольку $S^-(F) \cap F(F) = \emptyset$, идем на шаг 7.

Шаг 7. Поскольку $j \notin S^-(F)$, заносим задачу F в список задач. Список состоит из задач E, F . Задача F имеет вид:

$$\begin{array}{l} S^+(F) = \{c\}; \\ S^-(F) = \{a\}; \\ F(F) = \{b, d\}; \\ \xi_{a,a} = 0; \\ \xi_{c,a} = 1; \end{array} \quad M(F) = \begin{array}{c|cccc} & a & b & c & d \\ \hline 1 & a & \mathbf{b} & \mathbf{c} & \mathbf{d} \\ 2 & \mathbf{c} & a & d & b \\ 3 & b & c & a & a \\ 4 & d & d & b & c \end{array}; \quad \underline{C}(F) = 2.$$

Идем на шаг 10.

Шаг 10. Рассматриваем задачу G . Сначала полагаем $S^+(G) = S^+(C) = \emptyset$, $S^-(G) = S^-(C) = \{a\}$, $F(G) = F(C) = \{a, b, c, d\}$.

Шаг 11. Добавляем вершину c в множество немедианных вершин, т.е. полагаем $S^-(G) = \{a, c\}$.

Шаг 12. Рассматриваем еще не прикрепленные вершины $a \in F(G) \cap S^-(G)$, $c \in F(G) \cap S^-(G)$. Поскольку $x_a = c$, $x_c = c$ и $c \in S^-(G)$, находим для этих вершин вариант прикрепления: $x_a = b$, $x_c = d$. Матрица M имеет вид

$$M(G) = \begin{array}{c|cccc} & a & b & c & d \\ \hline 1 & a & \mathbf{b} & c & \mathbf{d} \\ 2 & c & a & \mathbf{d} & b \\ 3 & \mathbf{b} & c & a & a \\ 4 & d & d & b & c \end{array}$$

Пересчитываем оценку качества решения: $\underline{C}(G) = \underline{C}(G) - d(c, a) + d(b, a) - d(c, c) + d(d, c) = 2 - 2 + 4 - 0 + 5 = 9$. Поскольку множество $F(G) \cap S^-(G)$ просмотрено, идем на шаг 13.

Шаг 13. Поскольку $\underline{C}(G) < C^*$, и $|S^-(G)| = p - r$, идем на шаг 14.

Шаг 14. Полагаем $S^+(G) = V \setminus S^-(G) = \{b, d\}$. Для вершин $\{a, b, c, d\} \in F(G)$ полагаем $\xi_{b,a} = 1$, $\xi_{b,b} = 1$, $\xi_{c,d} = 1$, $\xi_{d,d} = 1$. Запоминаем решение $S^+(G) = \{b, d\}$ и положим $C^* = \underline{C}(G) = 9$. Из списка задач ничего не удаляем. Идем на шаг 1.

Шаг 1. Выбираем задачу E из списка. Задача E имеет вид:

$$\begin{array}{l} S^+(E) = \{a\}; \\ S^-(E) = \{b\}; \\ F(E) = \{c, d\}; \\ \xi_{a,a} = 1; \\ \xi_{b,b} = 0; \\ \xi_{a,b} = 1; \end{array} \quad M(E) = \begin{array}{c|cccc} & a & b & c & d \\ \hline 1 & \mathbf{a} & b & c & \mathbf{d} \\ 2 & c & \mathbf{a} & d & b \\ 3 & b & c & a & a \\ 4 & d & d & b & c \end{array}; \quad \underline{C}(E) = 1.$$

Выбираем из множества $F(E)$ вершину c , с которой связан индекс $k_c = 1$ и вариант прикрепления $x_c = c$. Производим ветвление по переменной $\xi_{c,c}$. В задаче H полагаем $\xi_{c,c} = 1$, в задаче I полагаем $\xi_{c,c} = 0$. Оценки качества решения $\underline{C}(H) = \underline{C}(I) = \underline{C}(E) = 1$.

Шаг 2. Рассматриваем задачу H . Сначала полагаем $S^+(H) = S^+(E) = \{a\}$, $S^-(H) = S^-(E) = \{b\}$, $F(H) = F(E) = \{c, d\}$.

Шаг 3. Добавляем вершину c в множество медианных вершин и удаляем из множества неприкрепленных, т.е. полагаем $S^+(H) = \{a, c\}$, $F(H) = \{d\}$. Поскольку $|S^+(H)| = r$, идем на шаг 4.

Шаг 4. Поскольку $\underline{C}(H) < C^*$ и $|S^+(H)| = r$, то очередное медианное множество построено. Полагаем $S^-(H) = S^-(H) \cup F(H) = \{b, d\}$ и найдем для еще не прикрепленной вершины d наилучший вариант прикрепления к вершине из множества $S^+(H)$: $x_d = a$. Матрица M имеет вид

$$M(H) = \begin{array}{c|cccc} & a & b & c & d \\ \hline 1 & \mathbf{a} & b & c & \mathbf{d} \\ 2 & c & \mathbf{a} & d & b \\ 3 & b & c & a & \mathbf{a} \\ 4 & d & d & b & c \end{array}$$

Полагаем $\xi_{a,d} = 1$.

Шаг 5. Пересчитываем оценку качества решения: $\underline{C}(H) = d(a, b) + d(a, d) = 1 + 4 = 5$. Поскольку $\underline{C}(H) < C^*$, запоминаем текущее множество медианных вершин $S^+(H) = \{a, c\}$ и новую оценку качества решения $C^* = 5$. Список состоит из задачи F , для нее $\underline{C}(F) < C^*$, оставляем ее в списке. Идем на шаг 10.

Шаг 10. Рассматриваем задачу I . Сначала полагаем $S^+(I) = S^+(E) = \{a\}$, $S^-(I) = S^-(E) = \{b\}$, $F(I) = F(E) = \{c, d\}$.

Шаг 11. Добавляем вершину c в множество немедианных вершин, т.е. полагаем $S^-(E) = \{b, c\}$.

Шаг 12. Рассматриваем еще не прикрепленную вершину $c \in F(I) \cap S^-(I)$. Поскольку $x_c = c$ и $c \in S^-(I)$, находим для этой вершины вариант прикрепления: $x_c = d$. Матрица M имеет вид

$$M(I) = \begin{array}{c|cccc} & a & b & c & d \\ \hline 1 & \mathbf{a} & b & c & \mathbf{d} \\ 2 & c & \mathbf{a} & \mathbf{d} & b \\ 3 & b & c & a & a \\ 4 & d & d & b & c \end{array}$$

Пересчитываем оценку качества решения: $\underline{C}(I) = \underline{C}(I) - d(c, c) + d(d, c) = 1 - 0 + 5 = 6$. Поскольку множество $F(I) \cap S^-(I)$ просмотрено, идем на шаг 13.

Шаг 13. Поскольку $\underline{C}(I) > C^*$, не рассматриваем далее задачу I и идем на шаг 1.

Шаг 1. Выбираем задачу F из списка. Задача F имеет вид:

$$\begin{array}{l} S^+(F) = \{c\}; \\ S^-(F) = \{a\}; \\ F(F) = \{b, d\}; \\ \xi_{a,a} = 0; \\ \xi_{c,a} = 1; \end{array} \quad M(F) = \begin{array}{c|cccc} & a & b & c & d \\ \hline 1 & a & \mathbf{b} & \mathbf{c} & \mathbf{d} \\ 2 & \mathbf{c} & a & d & b \\ 3 & b & c & a & a \\ 4 & d & d & b & c \end{array} \quad \underline{C}(F) = 2.$$

Выбираем из множества $F(F)$ вершину b , с которой связан индекс $k_b = 1$ и вариант прикрепления $x_b = b$. Производим ветвление по переменной $\xi_{b,b}$. В задаче J полагаем $\xi_{b,b} = 1$, в задаче K полагаем $\xi_{b,b} = 0$. Оценки качества решения $\underline{C}(J) = \underline{C}(K) = \underline{C}(F) = 2$.

Шаг 2. Рассматриваем задачу J . Сначала полагаем $S^+(J) = S^+(F) = \{c\}$, $S^-(J) = S^-(F) = \{a\}$, $F(J) = F(F) = \{b, d\}$.

Шаг 3. Добавляем вершину b в множество медианных вершин и удаляем из множества неприкрепленных, т.е. полагаем $S^+(J) = \{b, c\}$, $F(J) = \{d\}$. Поскольку $|S^+(J)| = r$, идем на шаг 4.

Шаг 4. Поскольку $\underline{C}(J) < C^*$ и $|S^+(J)| = r$, то очередное медианное множество построено. Полагаем $S^-(J) = S^-(J) \cup F(J) = \{a, d\}$ и найдем для еще не прикрепленной вершины d наилучший вариант прикрепления к вершине из множества $S^+(J)$: $x_d = b$. Матрица M имеет вид

$$M(J) = \begin{array}{c|cccc} & a & b & c & d \\ \hline 1 & a & \mathbf{b} & \mathbf{c} & d \\ 2 & \mathbf{c} & a & d & \mathbf{b} \\ 3 & b & c & a & a \\ 4 & d & d & b & c \end{array}$$

Полагаем $\xi_{b,d} = 1$.

Шаг 5. Пересчитываем оценку качества решения: $\underline{C}(J) = d(c, a) + d(b, d) = 2 + 3 = 5$. Поскольку $\underline{C}(H) = C^*$, не рассматриваем далее задачу J . Идем на шаг 10.

Шаг 10. Рассматриваем задачу K . Сначала полагаем $S^+(K) = S^+(F) = \{c\}$, $S^-(K) = S^-(F) = \{a\}$, $F(K) = F(F) = \{b, d\}$.

Шаг 11. Добавляем вершину b в множество немедианных вершин, т.е. полагаем $S^-(K) = \{a, b\}$.

Шаг 12. Рассматриваем еще не прикрепленную вершину $b \in F(K) \cap S^-(K)$. Поскольку $x_b = b$ и $b \in S^-(K)$, находим для этой вершины вариант прикрепления: $x_b = c$. Матрица M имеет вид

$$M(K) = \begin{array}{c|cccc} & a & b & c & d \\ \hline 1 & a & b & \mathbf{c} & \mathbf{d} \\ 2 & \mathbf{c} & a & d & b \\ 3 & b & \mathbf{c} & a & a \\ 4 & d & d & b & c \end{array}$$

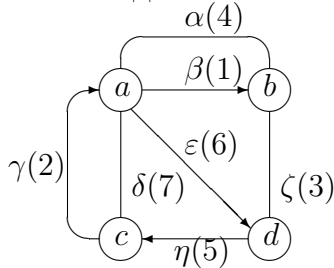
Пересчитываем оценку качества решения: $\underline{C}(K) = \underline{C}(K) - d(b, b) + d(c, b) = 2 - 0 + 3 = 5$. Поскольку множество $F(K) \cap S^-(K)$ просмотрено, идем на шаг 13.

Шаг 13. Поскольку $\underline{C}(K) = C^*$, не рассматриваем далее задачу K и идем на шаг 1.

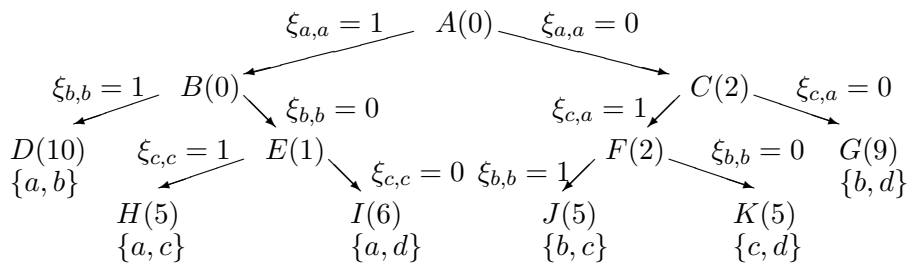
Шаг 1. Список задач пуст, идем на конец.

Конец. Получена кратная медиана $\{a, c\}$.

Для наглядности выпишем дерево решения и распишем ниже все задачи. В скобках после обозначения задачи выписана ее оценка качества решения. Для задач, в которых построено медианное множество мощности $r = 2$, оно выписано под обозначением задачи.



	a	b	c	d
a	0	1	7	4
b	4	0	8	3
c	2	3	0	6
d	7	3	5	0



Задача A:

$$\begin{array}{l}
 S^+(A) = \emptyset; \\
 S^-(A) = \emptyset; \\
 F(A) = \{a, b, c, d\};
 \end{array}
 \quad
 M(A) =
 \begin{array}{c|cccc}
 & a & b & c & d \\
 \hline
 1 & \mathbf{a} & \mathbf{b} & \mathbf{c} & \mathbf{d} \\
 2 & c & a & d & b \\
 3 & b & c & a & a \\
 4 & d & d & b & c
 \end{array}
 ; \quad \underline{C}(A) = 0.$$

Задача B:

$$\begin{array}{l}
 S^+(B) = \{a\}; \\
 S^-(B) = \emptyset; \\
 F(B) = \{b, c, d\};
 \end{array}
 \quad
 M(B) =
 \begin{array}{c|cccc}
 & a & b & c & d \\
 \hline
 1 & \mathbf{a} & \mathbf{b} & \mathbf{c} & \mathbf{d} \\
 2 & c & a & d & b \\
 3 & b & c & a & a \\
 4 & d & d & b & c
 \end{array}
 ; \quad \underline{C}(B) = 0.$$

Задача C:

$$\begin{array}{l}
 S^+(C) = \emptyset; \\
 S^-(C) = \{a\}; \\
 F(C) = \{a, b, c, d\};
 \end{array}
 \quad
 M(C) =
 \begin{array}{c|cccc}
 & a & b & c & d \\
 \hline
 1 & a & \mathbf{b} & \mathbf{c} & \mathbf{d} \\
 2 & \mathbf{c} & a & d & b \\
 3 & b & c & a & a \\
 4 & d & d & b & c
 \end{array}
 ; \quad \underline{C}(C) = 2.$$

Задача D:

$$\begin{array}{l}
 S^+(D) = \{a, b\}; \\
 S^-(D) = \{c, d\}; \\
 F(D) = \emptyset;
 \end{array}
 \quad
 M(D) =
 \begin{array}{c|cccc}
 & a & b & c & d \\
 \hline
 1 & \mathbf{a} & \mathbf{b} & c & d \\
 2 & c & a & d & \mathbf{b} \\
 3 & b & c & \mathbf{a} & a \\
 4 & d & d & b & c
 \end{array}
 ; \quad \underline{C}(D) = 10.$$

Задача E:

$$\begin{array}{l}
 S^+(E) = \{a\}; \\
 S^-(E) = \{b\}; \\
 F(E) = \{c, d\};
 \end{array}
 \quad
 M(E) =
 \begin{array}{c|cccc}
 & a & b & c & d \\
 \hline
 1 & \mathbf{a} & \mathbf{b} & \mathbf{c} & \mathbf{d} \\
 2 & c & \mathbf{a} & d & b \\
 3 & b & c & a & a \\
 4 & d & d & b & c
 \end{array}
 ; \quad \underline{C}(E) = 1.$$

Задача F:

$$\begin{array}{l}
 S^+(F) = \{c\}; \\
 S^-(F) = \{a\}; \\
 F(F) = \{b, d\};
 \end{array}
 \quad
 M(F) =
 \begin{array}{c|cccc}
 & a & b & c & d \\
 \hline
 1 & a & \mathbf{b} & \mathbf{c} & \mathbf{d} \\
 2 & \mathbf{c} & a & d & b \\
 3 & b & c & a & a \\
 4 & d & d & b & c
 \end{array}
 ; \quad \underline{C}(F) = 2.$$

Задача G :

$$\begin{array}{l}
 S^+(G) = \{b, d\}; \\
 S^-(G) = \{a, c\}; \\
 F(G) = \emptyset;
 \end{array}
 \quad
 M(G) =
 \begin{array}{c|cccc}
 & a & b & c & d \\
 \hline
 1 & a & \mathbf{b} & c & \mathbf{d} \\
 2 & c & a & \mathbf{d} & b \\
 3 & \mathbf{b} & c & a & a \\
 4 & d & d & b & c
 \end{array}
 ; \quad \underline{C}(G) = 2.$$

Задача H :

$$\begin{array}{l}
 S^+(H) = \{a, c\}; \\
 S^-(H) = \{b, d\}; \\
 F(H) = \emptyset;
 \end{array}
 \quad
 M(H) =
 \begin{array}{c|cccc}
 & a & b & c & d \\
 \hline
 1 & \mathbf{a} & b & \mathbf{c} & d \\
 2 & c & \mathbf{a} & d & b \\
 3 & b & c & a & \mathbf{a} \\
 4 & d & d & b & c
 \end{array}
 ; \quad \underline{C}(H) = 5.$$

Задача I :

$$\begin{array}{l}
 S^+(I) = \{a, d\}; \\
 S^-(I) = \{b, c\}; \\
 F(I) = \emptyset;
 \end{array}
 \quad
 M(I) =
 \begin{array}{c|cccc}
 & a & b & c & d \\
 \hline
 1 & \mathbf{a} & b & c & \mathbf{d} \\
 2 & c & \mathbf{a} & \mathbf{d} & b \\
 3 & b & c & a & a \\
 4 & d & d & b & c
 \end{array}
 ; \quad \underline{C}(I) = 6.$$

Задача J :

$$\begin{array}{l}
 S^+(J) = \{b, c\}; \\
 S^-(J) = \{a, d\}; \\
 F(J) = \emptyset;
 \end{array}
 \quad
 M(J) =
 \begin{array}{c|cccc}
 & a & b & c & d \\
 \hline
 1 & a & \mathbf{b} & \mathbf{c} & d \\
 2 & \mathbf{c} & a & d & \mathbf{b} \\
 3 & b & c & a & a \\
 4 & d & d & b & c
 \end{array}
 ; \quad \underline{C}(J) = 5.$$

Задача K :

$$\begin{array}{l}
 S^+(K) = \{c, d\}; \\
 S^-(K) = \{a, b\}; \\
 F(K) = \emptyset;
 \end{array}
 \quad
 M(K) =
 \begin{array}{c|cccc}
 & a & b & c & d \\
 \hline
 1 & a & b & \mathbf{c} & \mathbf{d} \\
 2 & \mathbf{c} & a & d & b \\
 3 & b & \mathbf{c} & a & a \\
 4 & d & d & b & c
 \end{array}
 ; \quad \underline{C}(K) = 5.$$

В методе ветвей и границ число перебираемых вариантов зависит не только от данных, но и от порядка выбора переменных, по которым происходит ветвление. В данном примере мы фактически произвели полный перебор всех возможных множеств из двух вершин, что является долго вычислимым для задач большой размерности. Попробуем сократить перебор за счет выбора переменной ветвления. Заметим, что увеличение оценки качества решения происходит в задаче A'' , когда переменная $\xi_{x_j, j} = 0$, т.к. при этом вершина x_j классифицируется как немедианная. В связи с этим для каждой вершины $t \in F(A'')$, для которых наилучшим вариантом прикрепления является вершина x_j (по крайней мере, для вершины j)

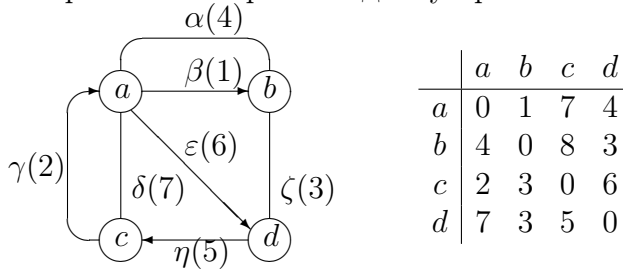
ищется другой вариант прикрепления x'_t и оценка качества решения увеличивается на величину $-d(x_j, t) + d(x'_t, t)$.

Будем теперь выбирать переменную ветвления $\xi_{x_j, j}$ как аргумент максимума величины

$$\sum_{t \in F(A'') : x_t = x_j} (-d(x_j, t) + d(x'_t, t)).$$

Чем больше эта величина, тем больше вероятность, что задача A'' будет отброшена. Кроме того, будем всегда выбирать из списка задачу с наименьшей оценкой стоимости решения.

Пример. Рассмотрим граф из предыдущих примеров и его матрицу расстояний вершина-вершина. Построим медиану кратности $r = 2$.



Записываем стартовую задачу.

Задача A :

$$\begin{array}{l}
 S^+(A) = \emptyset; \\
 S^-(A) = \emptyset; \\
 F(A) = \{a, b, c, d\};
 \end{array}
 \quad
 M(A) =
 \begin{array}{c|cccc}
 & a & b & c & d \\
 \hline
 1 & \mathbf{a} & \mathbf{b} & \mathbf{c} & \mathbf{d} \\
 2 & c & a & d & b \\
 3 & b & c & a & a \\
 4 & d & d & b & c
 \end{array}
 ; \quad \underline{C}(A) = 0.$$

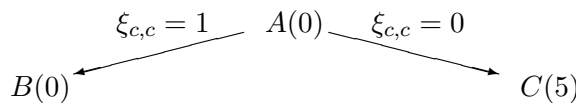
В задаче A для всех вершин $k_j = 1$, и $x_j = j$. Допустим, мы будем производить деление по переменной $\xi_{a, a}$, т.е. $j = a$, $x_j = a$. Тогда $\{t \in F(A'') : x_t = x_j\} = \{a\}$, новый вариант прикрепления для вершины a – это $x'_a = c$ и оценка качества решения задачи A'' по сравнению с задачей A увеличится на величину

$$-d(a, a) + d(c, a) = -0 + 2 = 2.$$

Рассуждая аналогично, получаем для остальных переменных:

$$\begin{array}{l}
 \xi_{b, b} : x'_b = a, \quad -d(b, b) + d(a, b) = -0 + 1 = 1; \\
 \xi_{c, c} : x'_c = d, \quad -d(c, c) + d(d, c) = -0 + 5 = 5; \\
 \xi_{d, d} : x'_d = b, \quad -d(d, d) + d(b, d) = -0 + 3 = 3.
 \end{array}$$

Значит, наибольшее увеличение качества решения задачи A'' произойдет при выборе переменной ветвления $\xi_{c, c}$. Записываем дерево решений:



Задача B :

$$\begin{array}{l}
 S^+(B) = \{c\}; \\
 S^-(B) = \emptyset; \\
 F(B) = \{a, b, d\};
 \end{array}
 \quad
 M(B) =
 \begin{array}{c|cccc}
 & a & b & c & d \\
 \hline
 1 & \mathbf{a} & \mathbf{b} & \mathbf{c} & \mathbf{d} \\
 2 & c & a & d & b \\
 3 & b & c & a & a \\
 4 & d & d & b & c
 \end{array}
 ; \quad \underline{C}(B) = 0.$$

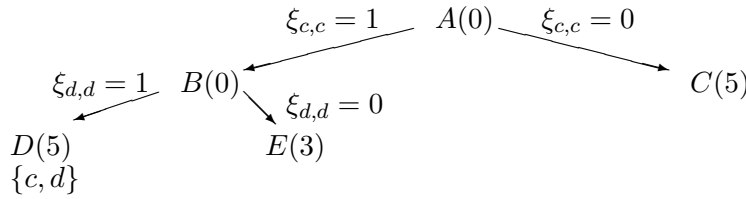
Задача C :

$$\begin{array}{l}
 S^+(C) = \emptyset; \\
 S^-(C) = \{c\}; \\
 F(C) = \{a, b, c, d\};
 \end{array}
 \quad
 M(C) =
 \begin{array}{c|cccc}
 & a & b & c & d \\
 \hline
 1 & \mathbf{a} & \mathbf{b} & \mathbf{c} & \mathbf{d} \\
 2 & c & a & \mathbf{d} & b \\
 3 & b & c & a & a \\
 4 & d & d & b & c
 \end{array}
 ; \quad \underline{C}(C) = 5.$$

Выбираем из списка задачу B . Рассматриваем все возможные переменные ветвления:

$$\begin{array}{l}
 \xi_{a,a}: \quad x'_a = c, \quad -d(a, a) + d(c, a) = -0 + 2 = 2; \\
 \xi_{b,b}: \quad x'_b = a, \quad -d(b, b) + d(a, b) = -0 + 1 = 1; \\
 \xi_{d,d}: \quad x'_d = b, \quad -d(d, d) + d(b, d) = -0 + 3 = 3.
 \end{array}$$

Производим ветвление по переменной $\xi_{d,d}$, получим дерево решений:



Задача D :

$$\begin{array}{l}
 S^+(D) = \{c, d\}; \\
 S^-(D) = \{a, b\}; \\
 F(D) = \emptyset;
 \end{array}
 \quad
 M(D) =
 \begin{array}{c|cccc}
 & a & b & c & d \\
 \hline
 1 & a & b & \mathbf{c} & \mathbf{d} \\
 2 & \mathbf{c} & a & d & b \\
 3 & b & \mathbf{c} & a & a \\
 4 & d & d & b & c
 \end{array}
 ; \quad \underline{C}(D) = 5.$$

Задача E :

$$\begin{array}{l}
 S^+(E) = \{c\}; \\
 S^-(E) = \{d\}; \\
 F(E) = \{a, b, d\};
 \end{array}
 \quad
 M(E) =
 \begin{array}{c|cccc}
 & a & b & c & d \\
 \hline
 1 & \mathbf{a} & \mathbf{b} & \mathbf{c} & \mathbf{d} \\
 2 & c & a & d & \mathbf{b} \\
 3 & b & c & a & a \\
 4 & d & d & b & c
 \end{array}
 ; \quad \underline{C}(E) = E.$$

В задаче D получено медианное множество $\{c, d\}$, теперь $C^* = \underline{C}(D) = 5$. Удаляем из списка задачу C , поскольку $\underline{C}(C) = C^*$.

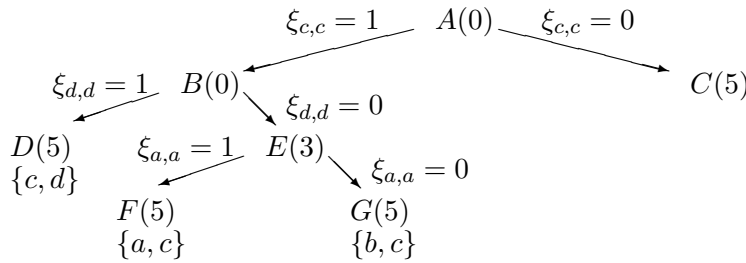
Выбираем из списка задачу E :

$$\begin{array}{l}
 S^+(E) = \{c\}; \\
 S^-(E) = \{d\}; \\
 F(E) = \{a, b, d\};
 \end{array}
 \quad
 M(E) =
 \begin{array}{c|cccc}
 & a & b & c & d \\
 \hline
 1 & \mathbf{a} & \mathbf{b} & \mathbf{c} & d \\
 2 & c & a & d & \mathbf{b} \\
 3 & b & c & a & a \\
 4 & d & d & b & c
 \end{array}
 ; \quad \underline{C}(E) = E.$$

Рассматриваем все возможные переменные ветвления:

$$\begin{array}{l}
 \xi_{a,a} : x'_a = c, \quad -d(a, a) + d(c, a) = -0 + 2 = 2; \\
 \xi_{b,b} : x'_b = a, x'_d = a, \quad -d(b, b) + d(a, b) - d(b, d) + d(a, d) = -0 + 1 - 3 + 4 = 2; \\
 \xi_{b,d} : x'_b = a, x'_d = a, \quad -d(b, b) + d(a, b) - d(b, d) + d(a, d) = -0 + 1 - 3 + 4 = 2.
 \end{array}$$

Можно выбрать любую переменную ветвления, выбираем $\xi_{a,a}$. Записываем дерево решений:



Задача F :

$$\begin{array}{l}
 S^+(F) = \{a, c\}; \\
 S^-(F) = \{b, d\}; \\
 F(F) = \emptyset;
 \end{array}
 \quad
 M(F) =
 \begin{array}{c|cccc}
 & a & b & c & d \\
 \hline
 1 & \mathbf{a} & b & \mathbf{c} & d \\
 2 & c & \mathbf{a} & d & b \\
 3 & b & c & a & \mathbf{a} \\
 4 & d & d & b & c
 \end{array}
 ; \quad \underline{C}(F) = 5.$$

Задача G :

$$\begin{array}{l}
 S^+(G) = \{b, c\}; \\
 S^-(G) = \{a, d\}; \\
 F(G) = \emptyset;
 \end{array}
 \quad
 M(G) =
 \begin{array}{c|cccc}
 & a & b & c & d \\
 \hline
 1 & a & \mathbf{b} & \mathbf{c} & d \\
 2 & \mathbf{c} & a & d & \mathbf{b} \\
 3 & b & c & a & a \\
 4 & d & d & b & c
 \end{array}
 ; \quad \underline{C}(G) = 5.$$

Для задач F и G получаем, что $\underline{C}(F) = C^*$, $\underline{C}(G) = C^*$, значит, далее их не рассматриваем. Итак, получено медианное множество $\{c, d\}$. Оно отличается от полученного в предыдущем примере, но обладает той же оценкой качества решения. Обратим внимание, что на этот раз было просмотрено три множества вместо шести.

Итак, нами рассмотрена задача о кратной медиане в первой постановке. Рассмотрим вторую постановку задачи.

Задача 2. Задано расстояние Δ . Найти множество вершин графа X_r наименьшей мощности r , для которого

$$\sum_{k \in V} d(X_r, k) \leq \Delta.$$

Для ее решения также можно использовать метод ветвей и границ с некоторыми модификациями.

Во-первых, задачи, для которых $\underline{C} > \Delta$, не добавляются в список задач и далее не рассматриваются. Также не рассматриваются задачи, для которых число медианных вершин $|S^+| \geq r^* - 1$, где r^* – число вершин в наилучшем найденном решении, поскольку медианные множества всех задач, полученных из текущей после процедур ветвления, будут содержать по крайней мере на одну вершину больше. Сначала $r^* = p$, т.е. все вершины считаются медианными.

Во-вторых, для каждой задачи, наряду с нижней границей качества решения вычисляется верхняя граница \overline{C} , которая представляет собой сумму расстояний до всех вершин от ближайших к ним вершин текущего медианного множества, т.е.

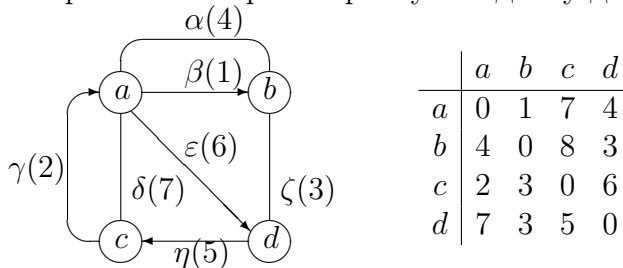
$$\overline{C} = \sum_{j \in V \setminus S^+} \min_{i \in S^+} d(i, j).$$

Для нахождения такой ближайшей вершины может использоваться матрица M : в столбце j находится элемент с наименьшим номером строки i , для которого $m_{i,j} \in S^+$. Если медианное множество пусто, то \overline{C} можно рассчитать в предположении, что медианное множество будет содержать одну вершину, т.е.

$$\overline{C} = \min_{i \in F} \sum_{j \in V} d(i, j).$$

В третьих, решение считается полученным, если $\overline{C} \leq \Delta$. Тогда число вершин r в медианном множестве S^+ сравнивается с числом вершин наилучшем найденном решении r^* . Если $r < r^*$, то текущее решение запоминается как наилучшее, и полагается $r^* = r$. Из списка при этом удаля

Пример. Рассмотрим граф из предыдущих примеров и его матрицу расстояний вершина-вершина. Построим кратную медиану для $\Delta = 4$.

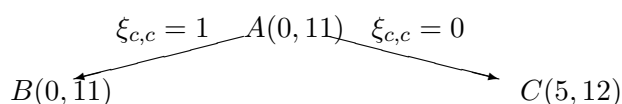


Записываем стартовую задачу.

Задача A :

$$\begin{aligned}
 S^+(A) &= \emptyset; \\
 S^-(A) &= \emptyset; \\
 F(A) &= \{a, b, c, d\};
 \end{aligned}
 \quad
 M(A) =
 \begin{array}{c|cccc}
 & a & b & c & d \\
 \hline
 1 & \mathbf{a} & \mathbf{b} & \mathbf{c} & \mathbf{d} \\
 2 & c & a & d & b \\
 3 & b & c & a & a \\
 4 & d & d & b & c
 \end{array}
 ; \quad
 \begin{aligned}
 \underline{C}(A) &= 0; \\
 \overline{C}(A) &= 11.
 \end{aligned}$$

Выбираем переменную ветвления $\xi_{c,c}$. Записываем дерево решений (в скобках указываем нижнюю и верхнюю оценки качества решения):



Задача B :

$$\begin{array}{l} S^+(B) = \{c\}; \\ S^-(B) = \emptyset; \\ F(B) = \{a, b, d\}; \end{array} \quad M(B) = \begin{array}{c|cccc} & a & b & c & d \\ \hline 1 & \mathbf{a} & \mathbf{b} & \mathbf{c} & \mathbf{d} \\ 2 & c & a & d & b \\ 3 & b & c & a & a \\ 4 & d & d & b & c \end{array}; \quad \begin{array}{l} \underline{C}(B) = 0; \\ \overline{C}(B) = 11. \end{array}$$

Оценка сверху качества решения вычислена следующим образом:

$$\overline{C}(B) = d(c, a) + d(c, b) + d(c, d) = 2 + 3 + 6 = 11.$$

Задача C :

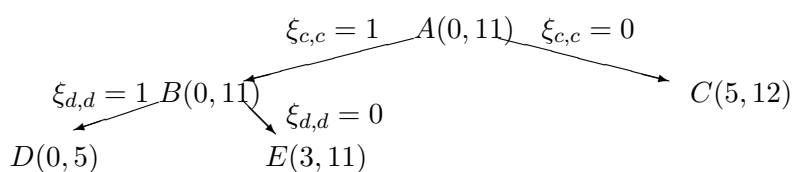
$$\begin{array}{l} S^+(C) = \emptyset; \\ S^-(C) = \{c\}; \\ F(C) = \{a, b, c, d\}; \end{array} \quad M(C) = \begin{array}{c|cccc} & a & b & c & d \\ \hline 1 & \mathbf{a} & \mathbf{b} & \mathbf{c} & \mathbf{d} \\ 2 & c & a & \mathbf{d} & b \\ 3 & b & c & a & a \\ 4 & d & d & b & c \end{array}; \quad \begin{array}{l} \underline{C}(C) = 5; \\ \overline{C}(C) = 12. \end{array}$$

Оценка сверху качества решения вычислена следующим образом:

$$\overline{C}(C) = d(a, b) + d(a, c) + d(a, d) = 1 + 7 + 4 = 12.$$

Поскольку $\overline{C}(C) > \Delta$, задачу C в список не заносим и далее не рассматриваем.

Выбираем из списка задачу B . Производим ветвление по переменной $\xi_{d,d}$, получим дерево решений:



Задача D :

$$\begin{array}{l} S^+(D) = \{c, d\}; \\ S^-(D) = \emptyset; \\ F(D) = \{a, b\}; \end{array} \quad M(D) = \begin{array}{c|cccc} & a & b & c & d \\ \hline 1 & \mathbf{a} & \mathbf{b} & \mathbf{c} & \mathbf{d} \\ 2 & c & a & d & b \\ 3 & b & c & a & a \\ 4 & d & d & b & c \end{array}; \quad \begin{array}{l} \underline{C}(D) = 0; \\ \overline{C}(D) = 5. \end{array}$$

Оценка сверху качества решения вычислена следующим образом:

$$\overline{C}(D) = d(c, a) + d(c, b) = 2 + 3 = 5.$$

Задача E :

$$\begin{array}{l}
 S^+(E) = \{c\}; \\
 S^-(E) = \{d\}; \\
 F(E) = \{a, b, d\};
 \end{array}
 \quad
 M(E) =
 \begin{array}{c|cccc}
 & a & b & c & d \\
 \hline
 1 & \mathbf{a} & \mathbf{b} & \mathbf{c} & d \\
 2 & c & a & d & \mathbf{b} \\
 3 & b & c & a & a \\
 4 & d & d & b & c
 \end{array}
 \quad
 \begin{array}{l}
 \underline{C}(E) = 3; \\
 \overline{C}(E) = 11.
 \end{array}$$

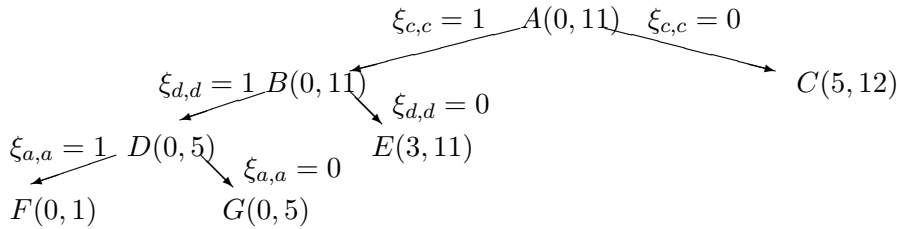
Оценка сверху качества решения вычислена следующим образом:

$$\overline{C}(E) = d(c, a) + d(c, b) + d(c, d) = 2 + 3 + 6 = 11.$$

Выбираем из списка задачу D :

$$\begin{array}{l}
 S^+(D) = \{c, d\}; \\
 S^-(D) = \emptyset; \\
 F(D) = \{a, b\};
 \end{array}
 \quad
 M(D) =
 \begin{array}{c|cccc}
 & a & b & c & d \\
 \hline
 1 & \mathbf{a} & \mathbf{b} & \mathbf{c} & \mathbf{d} \\
 2 & c & a & d & b \\
 3 & b & c & a & a \\
 4 & d & d & b & c
 \end{array}
 \quad
 \begin{array}{l}
 \underline{C}(D) = 0; \\
 \overline{C}(D) = 5.
 \end{array}$$

Производим ветвление по переменной $\xi_{a,a}$, получим дерево решений:



Задача F :

$$\begin{array}{l}
 S^+(F) = \{a, c, d\}; \\
 S^-(F) = \emptyset; \\
 F(F) = \{b\};
 \end{array}
 \quad
 M(F) =
 \begin{array}{c|cccc}
 & a & b & c & d \\
 \hline
 1 & \mathbf{a} & \mathbf{b} & \mathbf{c} & \mathbf{d} \\
 2 & c & a & d & b \\
 3 & b & c & a & a \\
 4 & d & d & b & c
 \end{array}
 \quad
 \begin{array}{l}
 \underline{C}(F) = 0; \\
 \overline{C}(F) = 1.
 \end{array}$$

Оценка сверху качества решения вычислена следующим образом:

$$\overline{C}(F) = d(a, b) = 1.$$

Поскольку $\overline{C}(F) < \Delta$ и $|S^+(F)| < r^*$, запоминаем новую кратную медиану $\{a, c, d\}$ и число вершин в ней $r^* = 3$.

Задача G :

$$\begin{array}{l}
 S^+(G) = \{c, d\}; \\
 S^-(G) = \{a\}; \\
 F(G) = \{a, b\};
 \end{array}
 \quad
 M(G) =
 \begin{array}{c|cccc}
 & a & b & c & d \\
 \hline
 1 & a & \mathbf{b} & \mathbf{c} & \mathbf{d} \\
 2 & \mathbf{c} & a & d & b \\
 3 & b & c & a & a \\
 4 & d & d & b & c
 \end{array}
 \quad
 \begin{array}{l}
 \underline{C}(G) = 2; \\
 \overline{C}(G) = 5.
 \end{array}$$

Оценка сверху качества решения вычислена следующим образом:

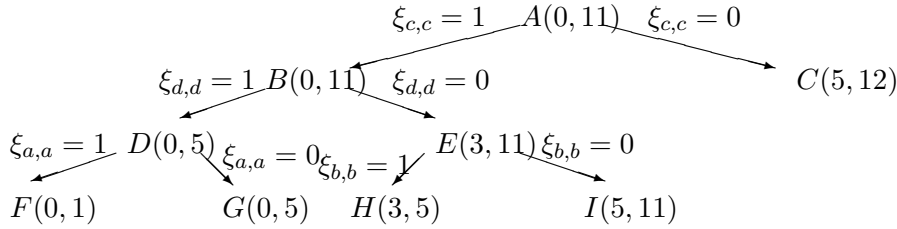
$$\overline{C}(G) = d(c, a) + d(c, b) = 2 + 3 = 5.$$

Задачу G не включаем в список и далее не рассматриваем, т.к. $|S^+(G)| = 2 = r^* - 1$, и медианные множества задач, полученных из G , будут включать по крайней мере три вершины, а медианное множество из трех вершин уже найдено.

Выбираем из списка задачу E :

$$\begin{array}{l} S^+(E) = \{c\}; \\ S^-(E) = \{d\}; \\ F(E) = \{a, b, d\}; \end{array} \quad M(E) = \begin{array}{c|cccc} & a & b & c & d \\ \hline 1 & \mathbf{a} & \mathbf{b} & \mathbf{c} & d \\ 2 & c & a & d & \mathbf{b} \\ 3 & b & c & a & a \\ 4 & d & d & b & c \end{array}; \quad \begin{array}{l} \underline{C}(E) = 3; \\ \overline{C}(E) = 11. \end{array}$$

Производим ветвление по переменной $\xi_{b,b}$, получим дерево решений:



Задача H :

$$\begin{array}{l} S^+(H) = \{c, b\}; \\ S^-(H) = \{d\}; \\ F(H) = \{a, d\}; \end{array} \quad M(H) = \begin{array}{c|cccc} & a & b & c & d \\ \hline 1 & \mathbf{a} & \mathbf{b} & \mathbf{c} & d \\ 2 & c & a & d & \mathbf{b} \\ 3 & b & c & a & a \\ 4 & d & d & b & c \end{array}; \quad \begin{array}{l} \underline{C}(H) = 3; \\ \overline{C}(H) = 5. \end{array}$$

Оценка сверху качества решения вычислена следующим образом:

$$\overline{C}(H) = d(c, a) + d(b, d) = 2 + 3 = 5.$$

Поскольку $\overline{C}(H) > \Delta$ и $|S^+(H)| = r^* - 1$, задачу H далее не рассматриваем.

Задача I :

$$\begin{array}{l} S^+(I) = \{c\}; \\ S^-(I) = \{b, d\}; \\ F(I) = \{a, b, d\}; \end{array} \quad M(I) = \begin{array}{c|cccc} & a & b & c & d \\ \hline 1 & \mathbf{a} & b & \mathbf{c} & d \\ 2 & c & \mathbf{a} & d & b \\ 3 & b & c & a & \mathbf{a} \\ 4 & d & d & b & c \end{array}; \quad \begin{array}{l} \underline{C}(I) = 5; \\ \overline{C}(I) = 11. \end{array}$$

Поскольку $\underline{C}(I) > \Delta$, задачу I далее не рассматриваем.

Все задачи просмотрены, найдено медианное множество $\{a, c, d\}$.

Поиск кратной главной медианы можно выполнять аналогично, с двумя отличиями: использовать расстояние вершина-ребро вместо расстояния вершина-вершина, и рассматривать, кроме вершин, еще середины неориентированных ребер графа.

4.7 Упражнения

1. Найти все возможные центры и медианы графа, заданного матрицей инцидентий, где $C(e)$ – вес ребра.

	α	β	γ	δ	ε	λ
a		1	1	-1		
b	1	1			-1	-1
c				1	1	
d	1		-1			1
$C(e)$	7	1	3	2	4	5

2. Найти абсолютные центры кратности 2 и 3 для графа из упражнения 1.

3. Найти кратные абсолютные центры минимальной мощности в пределах расстояний 2 и 4 для графа из упражнения 1.

4. Найти медианы кратности 2 и 3 для графа из упражнения 1.

5. Найти абсолютные медианы минимальной мощности в пределах расстояний 4 и 6 для графа из упражнения 1.

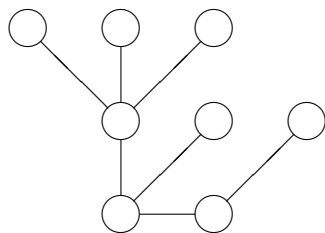
5 Деревья

Деревья занимают в теории графов особое положение. С одной стороны, как уже отмечалось, это связные графы с наименьшим возможным числом ребер. Они имеют ряд свойств, которые позволяют достаточно легко решить задачи, сложные для графов в общем случае, т.е. являются полезными с теоретической точки зрения, поскольку любую гипотезу теории графов можно сначала проверить на примере дерева. С другой стороны, они встречаются в качестве математической модели в различных областях. Например, еще в XIX веке Густав Кирхгоф использовал деревья для описания электрических цепей. Независимо от Кирхгофа деревья использовались Артуром Кэли для описания изомеров насыщенных углеводородов, и именно он впервые исследовал их свойства. В настоящее время деревья широко используются в программировании для хранения данных, для описания структуры программы, и т.д.

5.1 Свободные деревья

Определение. Граф $G(V, E)$ называется *деревом*, если он является связным и не имеет циклов.

Пример. Диаграмма дерева.



Определение. Граф $G(V, E)$, все компоненты связности которого являются деревьями, называется *лесом*.

Дерево обладает рядом свойств, которые сформулированы и доказаны в следующей теореме.

Теорема о шести эквивалентных утверждениях о дереве. Следующие утверждения эквивалентны:

- 1) граф $G(V, E)$ – дерево, то есть связный граф без циклов;
- 2) любые две вершины графа $G(V, E)$ соединены единственной простой цепью;
- 3) граф $G(V, E)$ связный, и любое ребро есть мост;
- 4) граф $G(V, E)$ связный, и число его ребер на 1 меньше числа вершин;
- 5) граф $G(V, E)$ не содержит циклов, и число его ребер на 1 меньше числа вершин;
- 6) граф $G(V, E)$ не содержит циклов, но добавление к нему любого нового ребра приводит к образованию ровно одного простого цикла, проходящего через это ребро.

Доказательство.

$1 \Rightarrow 2$: от противного. Пусть вершины v, w соединены двумя простыми цепями $vx_1x_2\dots x_lw$ $vy_1y_2\dots y_mv$. Тогда в графе существует цикл $vx_1x_2\dots x_lwy_my_{m-1}\dots y_1v$. Пусть не существует цепи $\langle v, w \rangle$, тогда граф несвязный.

$2 \Rightarrow 3$: от противного. Пусть ребро x , соединяющее вершины v и w – не мост. Тогда его удаление не приведет к увеличению числа компонент связности, то есть в графе $G - x$, а значит и в G , существует простая цепь, связывающая v и w , а ребро x – вторая цепь.

$3 \Rightarrow 4$: индукция по p . База индукции: при $p = 1$ $q = 0$, утверждение верно. Пусть $q(G) = p(G) - 1$ для всех графов $G(V, E)$ с числом вершин $p(G) < p$. Рассмотрим граф G с p вершинами и q ребрами. Удалим из него произвольное ребро x , которое является мостом. Получим граф, состоящий из двух компонент связности $G'(V', E')$, $G''(V'', E'')$, удовлетворяющих индукционному предположению. Имеем $q' = p' - 1$, $q'' = p'' - 1$, $q = q' + q'' + 1 = p' - 1 + p'' - 1 + 1 = p' + p'' - 1 = p - 1$.

$4 \Rightarrow 5$: от противного. Пусть граф содержит цикл $v_1v_2\dots v_lv_1$. Так как граф связный, остальные $p - l$ вершин имеют инцидентные им ребра, связывающие их с вершинами, входящими в цикл. Следовательно, число ребер в графе $q \geq l + p - l = p$, что противоречит условию $q = p - 1$.

$5 \Rightarrow 6$: от противного. Пусть при добавлении ребра (u, v) не образуется цикла, тогда в графе нет цепи $\langle u, v \rangle$, следовательно, граф не связный. Так как он не содержит циклов, его компоненты связности являются деревьями. Пусть в графе k компонент связности $G_i(V_i, E_i)$ с числом ребер q_i и числом вершин p_i , по доказанному для них $1 \Rightarrow 2 \Rightarrow 3 \Rightarrow 4$

$$q = \sum_{i=1}^k q_i = \sum_{i=1}^k (p_i - 1) = p - k.$$

Но так как $q = p - 1$, получаем $k = 1$, граф связный, без циклов, значит, является деревом и для каждой пары вершин u, v существует единственная цепь $\langle u, v \rangle$ ($1 \Rightarrow 2$). Соединив эти вершины ребром (u, v) , получим единственный простой цикл.

$6 \Rightarrow 1$: от противного. Пусть $G(V, E)$ – не связный, тогда есть две несвязные вершины u, v . Тогда добавление ребра (u, v) не приведет к образованию цикла, так как другой цепи $\langle u, v \rangle$ в графе нет.

5.2 Задача о соединении городов

Определение. *Остовным деревом (остовом)* связного графа G называется любой его подграф, содержащий все вершины графа G и являющийся деревом.

Несвязный граф не имеет остовных деревьев. Связный граф в общем случае имеет множество остовных деревьев. Число остовных деревьев полного графа K_p равно p^{p-2} .

Рассмотрим следующую задачу. Есть несколько городов. Требуется найти множество авиарейсов с минимальной суммарной длиной, соединяющих эти города, таким образом, чтобы из любого города можно было добраться до всех остальных городов.

Задачу о соединении городов можно формализовать следующим образом. Построим граф, каждой вершине которого соответствует город, а весом ребра является расстояние между соответствующими городами. Ставится задача нахождения *кратчайшего остова* – остова с минимальной суммой весов ребер.

Алгоритм Краскала. Пусть $G(V, E)$ – связный нагруженный граф с p вершинами.

Шаг 1. Выберем в E ребро e_1 наименьшего веса. Пусть оно образует дерево U_1 . Положим $n = 1$.

Шаг 2. Если $n = p - 1$, то исходным минимальным остовом является дерево U_n , конец. Иначе идем на шаг 3.

Шаг 3. Строим дерево U_{n+1} , добавляя к уже построенному дереву U_n ребро e_{n+1} минимального веса из не включенных в U_n ребер графа $G(V, E)$ и не образующее цикла с ребрами из U_n . n полагаем равным $n + 1$.

Обоснование алгоритма. Предположим, алгоритм строит остовное дерево T , а кратчайшим остовным деревом является S . Так как оба дерева содержат ровно $p - 1$ ребро, найдется по крайней мере одно ребро $e_1 \in T$, $e_1 \notin S$. Выберем из всех таких ребер ребро, которое в ходе выполнения алгоритма было первое включено в T (т.е. ребро минимального веса). Поскольку S также является деревом, в нем есть единственная простая цепь, соединяющая концы ребра e_1 . Вместе с e_1 эта цепь образует цикл, а т. к. T не содержит циклов, то в этой цепи найдется ребро $e_2 \notin T$.

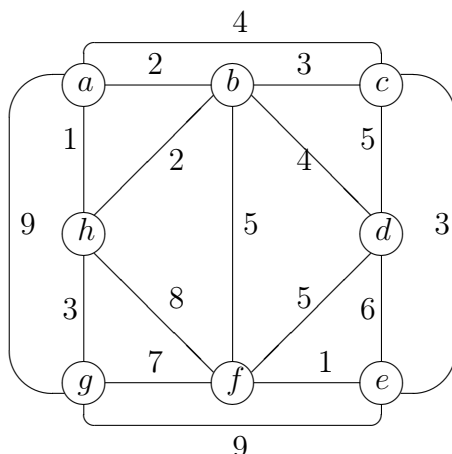
Удалим из дерева S ребро e_2 и добавим ребро e_1 , получим дерево S' . Поскольку S – кратчайший остов, вес дерева S' не меньше веса S , а значит, $C(e_2) \leq C(e_1)$. Если $C(e_2) < C(e_1)$, то ребро e_2 было просмотрено в алгоритме раньше, чем e_1 , и не было включено в дерево T , т. к. образовывало цикл с ребрами, включенными в него ранее. Но мы выбирали ребро e_1 как ребро минимального веса из включенных в T и не включенных в S , а значит, ребра, с которыми e_2 образует цикл, включены также и в S , что противоречит тому, что S является деревом. Следовательно, $C(e_2) = C(e_1)$ и веса деревьев S и S' совпадают.

Итак, мы построили дерево S' минимального веса, в котором ребер, включенных в дерево T , на одно больше, чем в S . Повторяя эти рассуждения для всех

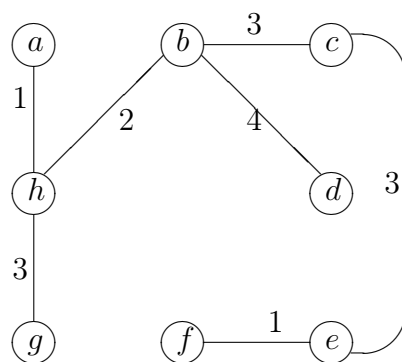
ребер из T , не включенных в S , мы получим, что остовное дерево T имеет минимальный вес.

Вычислительная сложность алгоритма. На каждой итерации алгоритма выбирается ребро с минимальным весом, это требует максимум q операций. Всего выполняется $p - 1$ итерация, а это означает, что вычислительная сложность алгоритма равна pq .

Пример. Рассмотрим нагруженный граф.



Выбираем ребро минимального веса 1 $e_1 = (a, h)$. Затем выбираем следующее ребро $e_2 = (e, f)$ веса 1, оно имеет минимальный вес среди ребер, не образующих цикл с e_1 . Следующее ребро $e_3 = (a, b)$ веса 2. Ребро (b, h) веса 2 не включаем в дерево, т.к. оно образует цикл с ребрами $e_1 = (a, h)$ и $e_3 = (a, b)$. Далее включаем в дерево ребра $e_4 = (b, c)$, $e_5 = (c, e)$ и $e_6 = (g, h)$ веса 3, затем $e_7 = (b, d)$ веса 4. Кратчайший остов построен.



- $e_1 = (a, h)$;
- $e_2 = (e, f)$;
- $e_3 = (a, b)$;
- $e_4 = (b, c)$;
- $e_5 = (c, e)$;
- $e_6 = (g, h)$;
- $e_7 = (b, d)$.

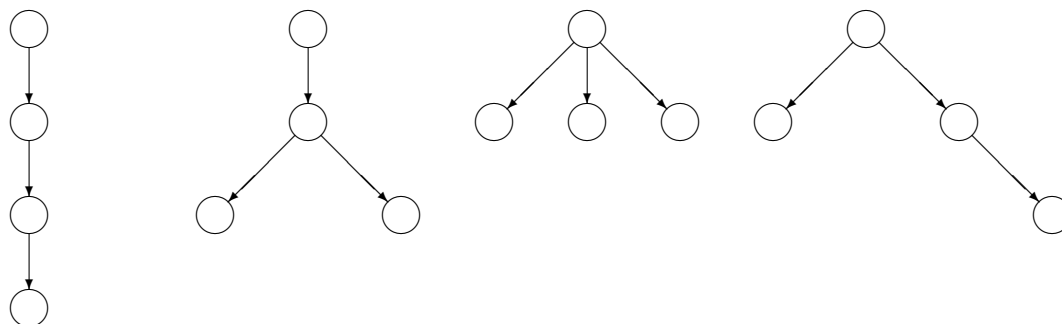
Задачу о соединении городов можно считать полностью решенной. Однако, если немного изменить условия, то алгоритм перестает решать задачу. Например, рассмотрим *задачу Штейнера* – предположим, что города соединяются железной дорогой, и нужно проложить рельсы так, чтобы все города были соединены и суммарная длина пути была минимальна. На языке теории графов это означает, что в граф могут вводиться дополнительные вершины. Эффективного алгоритма решения этой задачи не существует. алгоритм Краскала может использоваться как первое приближение.

5.3 Ориентированные, упорядоченные и бинарные деревья

Определение. *Ориентированным деревом* называется оргграф со следующими свойствами:

- 1) существует единственный узел, полустепень захода которого равна 0 (он называется *корнем* ордерова);
- 2) полустепень захода всех остальных узлов равна 1;
- 3) каждый узел достижим из корня.

Пример. Все различные ориентированные деревья с четырьмя узлами.



Замечание. Часто при изображении ориентированных деревьев корень изображается наверху, и, если существует дуга (u, v) , то узел u изображается выше узла v . При этом ориентация дуг опускается, поскольку предполагается, что дуги ориентированы «сверху вниз».

Теорема. *Ордерова обладает следующими свойствами:*

- 1) $q = p - 1$;
- 2) если в ордерове отменить ориентацию ребер, то получится свободное дерево;
- 3) в ордерове нет контуров;
- 4) для каждого узла существует единственный путь, ведущий в этот узел из корня;
- 5) правильный подграф, определяемый множеством узлов, достижимых из узла v , является ордером с корнем v (это дерево называется *поддеревом* узла v);
- 6) если в свободном дереве любую вершину назначить корнем, то можно ориентировать ребра таким образом, что получится ордерова.

Доказательство.

1) Число дуг равно сумме полустепеней захода всех узлов. Поскольку для всех узлов, кроме одного (корня), полустепень захода равна 1, а полустепень захода для корня равна 0, то число дуг $q = p - 1$.

2) Поскольку в ордерове каждый узел достижим из корня, то после отмены ориентации ребер в графе будет существовать путь между корнем и любой вершиной. Значит, между любыми двумя вершинами будет существовать путь, проходящий через корень. Итак, граф после отмены ориентации ребер будет связным, и в 1) для него доказано, что $q = p - 1$. Согласно теореме о шести эквивалентных утверждениях о дереве, такой граф является свободным деревом.

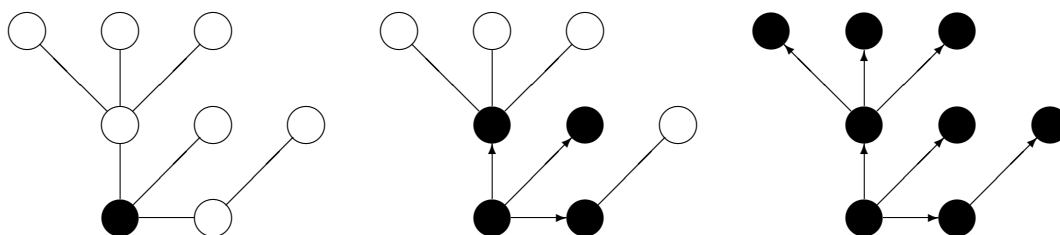
3) Предположим, в ордерове есть контур. Тогда после отмены ориентации ребер он преобразуется в цикл, что противоречит тому, что после отмены ориента-

ции ребер ордерено преобразуется в свободное дерево.

4) Пусть для узла v существуют два различных пути $\langle u, v \rangle$ из корня u . Тогда эти пути имеют хотя бы один общий узел (v), кроме u . Пусть s – первый такой узел после корня, тогда в него заходят две различные дуги, принадлежащие двум различным путям $\langle u, v \rangle$, что противоречит тому, что полустепень захода всех узлов, кроме корня, равна 1.

5) Проверим для данного подграфа свойства ордеревьев. По построению, каждый его узел, кроме v , достигим из v . Это означает, что во все такие узлы подграфа заходит по крайней мере одна дуга. Поскольку в исходном графе полустепень захода всех узлов равна 1, то это же свойство сохранится и для подграфа. Поскольку в ордереве нет контуров, то в построенном подграфе нет пути $\langle v, v \rangle$, содержащего хотя бы одну дугу, т.е. полустепень захода узла v равна 0. Все свойства ордерова выполнены.

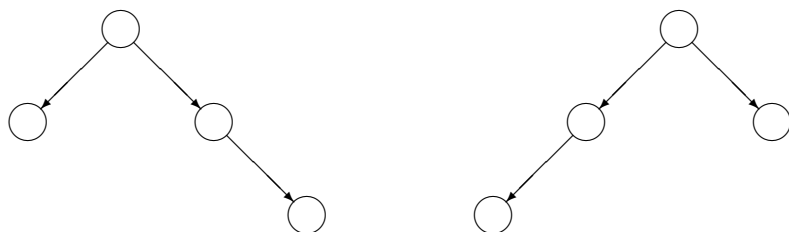
6) Выберем в свободном дереве любую вершину, отметим ее и назначим корнем. Ориентируем ребра графа в направлении «от корня», используя принцип обхода «в ширину». Рассматриваем все ребра, инцидентные корню, и ориентируем их так, чтобы они исходили из корня. Отмечаем вершины, в которые заходят данные ребра. Затем повторяем ту же процедуру для каждой отмеченной вершины: ориентируем еще не ориентированные ребра так, чтобы они исходили из вершины, и т.д. Проиллюстрируем этот процесс на примере (отмеченные вершины выделены черным).



Поскольку любые две вершины дерева соединены единственной простой цепью, в каждую вершину, кроме корня, будет заходить ровно одно ребро, и из корня будет существовать путь до каждой вершины. Поскольку в дереве нет циклов, в корень не будет заходить ни одной дуги.

Определение. Если порядок поддеревьев в ордереве фиксирован, то дерево называется *упорядоченным*.

Пример. Данные поддеревья изоморфны как ориентированные деревья, но неизоморфны как упорядоченные.



Ориентированные деревья широко используются в программировании.

1. Различные выражения языков программирования задаются упорядоченными деревьями.

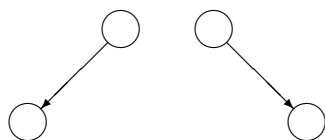
2. Ориентированные деревья используются для представления блочной структуры программы и связанной с ней структуры областей определения идентификаторов.

3. Для представления различных иерархических структур.

4. Для уравновешенных скобочных структур.

Определение. Если поддеревьев в ордереве ровно два – левое и правое (могут быть пустыми), то дерево называется *бинарным*.

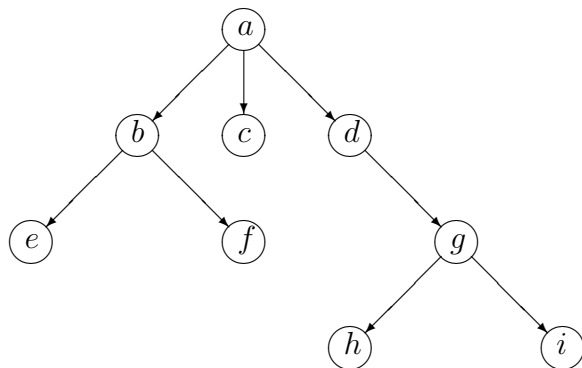
Пример. Деревья изоморфны как ориентированные и упорядоченные, но неизоморфны как бинарные.



Терминология. Узлы с нулевой полустепенью исхода называются *листьями*. Путь из корня в лист называется *ветвью*. Длина наибольшей ветви ордерова называется *высотой*. *Уровень* узла ордерова – это расстояние от корня до узла. Сам корень имеет уровень 0. Узлы одного уровня образуют *ярус* дерева. Узлы, не являющиеся листьями, называются *внутренними*.

Наряду с «растительной» применяется еще «генеалогическая» терминология. Узлы, достижимые из узла v , называются *потомками* узла v (потомки образуют поддерево). Если в дереве существует дуга $\{u, v\}$, то узел u называется *отцом* (или *родителем*) узла v , а узел v – *сыном* узла u . Сыновья одного отца называются *братьями*. В упорядоченном дереве чем левее расположен брат, тем он младше. Брат отца называется *дядей*.

Пример. Рассмотрим ордереве



В данном дереве:

- a – корень дерева;
- c, e, f, h, i – листья;
- $ac, abe, abf, adgh, adgi$ – ветви;
- высота дерева равна трем;
- узел a имеет уровень 0, узлы b, c, d образуют ярус уровня 1, узлы e, f, g – ярус уровня 2, узлы h, i – ярус уровня 3;
- узлы h, i являются потомками узла g , узлы g, h, i – потомками узла d , узлы e, f – потомками узла b , все узлы, кроме a – потомками узла a ;
- правильные подграфы, заданные множествами узлов $\{g, h, i\}, \{d, g, h, i\}, \{c\}, \{b, e, f\}$ образуют поддеревья;

- узлы b, c, d являются сыновьями узла a (узел a – отец узлов b, c, d), узлы e, f – сыновьями b (узел b – отец узлов e, f), узел g – сыном d (узел d – отец g), узлы h, i – сыновьями g (узел g – отец узлов h, i);
- узлы b, c, d – братья (b – младший, c старше b и младше d , d – старший), e, f – братья (e – младший, f – старший), и h, i – братья (h – младший, i – старший);
- узлы c, d являются дядями для e, f , узлы b, c – дяди для g .

Замечание. При изображении деревьев предполагается, что корень находится вверху, а все стрелки направлены сверху вниз, поэтому стрелки не изображаются.

Лемма о числе узлов бинарного дерева. Число узлов i -го яруса бинарного дерева не превосходит 2^i . Число узлов p в дереве высоты h не превосходит $2^{h+1} - 1$.

Доказательство. Докажем первое утверждение с помощью метода математической индукции. В ярус 0 входит только корень дерева, т.е. один узел, $1 = 2^0$, значит, утверждение верно. Предположим, оно верно для ярусов $0, 1, \dots, i-1$. Рассмотрим ярус i , на нем располагаются сыновья узлов яруса $i-1$, число которых не превосходит 2^{i-1} . Поскольку каждый узел имеет не более двух сыновей, число узлов на i -м ярусе не превосходит $2 \times 2^{i-1} = 2^i$.

Докажем второе утверждение леммы. Число узлов в дереве равно сумме узлов на всех уровнях. Используя первое утверждение леммы и формулу для суммы членов геометрической прогрессии, получаем

$$p \leq \sum_{i=0}^h 2^i = \frac{2^{h+1} - 1}{2 - 1} = 2^{h+1} - 1.$$

Лемма о числе листьев бинарного дерева. Число листьев бинарного дерева не превосходит $n + 1$, где n – число внутренних узлов. Если все узлы дерева имеют полустепень исхода 0 или 2, то число его листьев в точности равно $n + 1$.

Доказательство. Докажем первое утверждение, используя метод математической индукции. Пусть в бинарном дереве $n = 1$, т.е. имеется один внутренний узел. У этого узла будет 1 или 2 сына, которые являются листьями. Пусть утверждение верно для числа внутренних узлов $1, 2, \dots, n-1$. Рассмотрим дерево с n внутренними узлами. Выберем внутренний узел, оба сына которого являются листьями, либо, если таких узлов нет, узел, имеющий одного сына, который является листом. Удалим этот узел из дерева, заменив его листом. В полученном дереве $n-1$ внутренний узел, и, согласно индукционному предположению, не более чем n листьев. Значит, в исходном дереве число листьев не превосходит $n-1 + 2 = n+1$. Докажем второе утверждение. Если все узлы дерева имеют полустепень исхода 0 или 2, то при $n = 1$ в дереве имеется ровно два листа. Пусть утверждение верно для числа внутренних узлов $1, 2, \dots, n-1$. Рассмотрим дерево с n внутренними узлами. Выберем внутренний узел, оба сына которого являются листьями. Удалим этот узел из дерева, заменив его листом. В полученном дереве $n-1$ внутренний узел, и, согласно индукционному предположению, ровно n листьев. Значит, в исходном дереве число листьев равно $n-1 + 2 = n+1$.

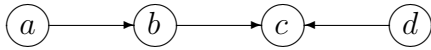
5.4 Построение ориентированного дерева и леса

В ориентированном графе существует несколько видов подграфов вида «лес» и «дерево». Рассмотрим орграф $D(V, E)$.

Определение. *Остовным ориентированным деревом (лесом)* орграфа $D(V, E)$ называется его остовный подграф, являющийся ордеревом (лесом).

В орграфе, в отличие от простого графа, может не существовать остовного ордерова.

Пример. Здесь $\{(a, b), (d, c)\}$ – остовный ориентированный лес, остовного ордерова не существует.



Рассмотрим нагруженный орграф $D(V, E)$.

Определение. *Максимальным (минимальным) ориентированным остовным деревом (лесом)* называется остовное ориентированное дерево (лес) с максимальной (минимальной) суммой весов дуг.

Рассмотрим алгоритм построения максимального ориентированного леса, который, как будет показано далее, может быть модифицирован для других задач.

В алгоритме используются два букета – *букет узлов*, содержащий просмотренные узлы, и *букет дуг*, содержащий дуги, условно включенные в максимальный ориентированный лес (некоторые из этих дуг на последнем этапе алгоритма могут быть исключены). Изначально оба букета пусты.

В ходе алгоритма выявленные контуры будут стягиваться в узел. Обозначим через G_0 начальный граф $D(V, E)$, а через G_1, G_2 и т. д. – графы, последовательно получающиеся после стягивания контуров. Букеты узлов и букеты дуг для этих графов соответственно обозначим через V_0, V_1 , и т. д. и E_0, E_1 и т. д.

Алгоритм Эдмондса построения максимального ориентированного леса

Начало. Задан нагруженный орграф $D(V, E)$. Положим $G_0(V, E) = D(V, E)$, $i = 0$, $V_0 = E_0 = \emptyset$.

Шаг 1. Если все узлы графа G_i включены в букет V_i , идем на шаг 3. Иначе рассматриваем очередной узел $v \notin V_i$ и включаем его в V_i . Среди дуг, заходящих в эту вершину, выбираем дугу e с максимальным положительным весом. Если таких дуг нет, повторяем шаг 1. Иначе включаем e в E_i . Если после этого E_i остается лесом, повторяем шаг 1.

Шаг 2. Находим в графе G_i контур μ_i , включающий дугу e . Стягиваем контур в узел v_i . Полученный граф обозначаем через G_{i+1} . Для всех дуг G_{i+1} , входящих в узел v_i , полагаем

$$C(x, v_i) = C(x, y) + C(r, s) - C(t, y),$$

где (r, s) – дуга минимального веса из контура μ_i , а (t, y) – дуга контура μ_i , заходящая в узел y (заметим, что $C(r, s) \geq 0$, $C(t, y) \geq C(r, s)$, а $C(t, y) \geq C(x, y)$, поскольку дуга (t, y) была включена в букет на первом шаге алгоритма. Формируем букет узлов V_{i+1} , включив в него узлы графа G_{i+1} , содержащиеся в V_i ($v_i \notin V_{i+1}$). Формируем букет дуг E_{i+1} , включив в него все дуги из E_i , не входящие в контур μ_i . Полагаем $i = i + 1$ и идем на шаг 1.

Шаг 3. Если $i = 0$, то дуги букета E_i образуют максимальный ориентированный лес в исходном графе, идем на конец. Если $i > 0$ и узел v_{i-1} является корнем некоторого ориентированного дерева в лесе, образованном дугами из E_i , идем на шаг 4, иначе на шаг 5.

Шаг 4. Рассматриваем дуги из букета E_i в совокупности с дугами контура μ_{i-1} . Это множество содержит ровно один контур μ_{i-1} . Удаляем из множества дугу наименьшего веса из контура μ_{i-1} , получаем новый букет дуг E_{i-1} , который образует в графе G_{i-1} ориентированный лес. Полагаем $i = i - 1$ и идем на шаг 3.

Шаг 5. В этом случае имеется единственная дуга (x, v_{i-1}) , заходящая в узел v_{i-1} . Эта дуга соответствует некоторой дуге (x, y) в графе G_{i-1} , где узел y принадлежит контуру μ_{i-1} . Рассматриваем дуги из букета E_i в совокупности с дугами контура μ_{i-1} . Это множество содержит ровно один контур μ_{i-1} . Значит, в узел y заходит ровно две дуги: (x, y) и (z, y) – дуга из контура μ_{i-1} . Удаляем (z, y) из множества дуг, получаем новый букет дуг E_{i-1} , который образует в графе G_{i-1} ориентированный лес. Полагаем $i = i - 1$ и идем на шаг 3.

Обоснование алгоритма. Рассмотрим букет дуг E_i в графе G_i для наибольшего полученного в алгоритме значения i . Лес, образованный дугами из E_i , в этом случае включает дуги, имеющие наибольший положительный вес из дуг, заходящих в данную вершину. Значит, этот лес является максимальным в графе G_i .

Покажем, что если ориентированный лес, определяемый букетом дуг E_i , является максимальным ориентированным лесом в графе G_i , то то же самое верно и для леса E_{i-1} графа G_{i-1} . Введем некоторые определения.

Пусть G' – подграф графа G_{i-1} , включающий все дуги G_{i-1} , кроме дуг, заходящих в вершины контура μ_{i-1} , а G'' – подграф графа G_{i-1} , включающий все дуги, заходящие в вершины контура μ_{i-1} . Через E' и E'' обозначим E_{i-1} подмножества дуг из букета E_{i-1} , принадлежащих G' и G'' соответственно. Очевидно, что E' и E'' – ориентированные леса в G' и G'' .

Если ориентированный лес E_{i-1} не является максимальным в G_{i-1} , то в нем найдется ориентированный лес B с большим суммарным весом дуг. Через B' и B'' обозначим подмножества дуг этого леса, принадлежащих G' и G'' соответственно. Понятно, что либо $C(B') \geq C(E')$, либо $C(B'') \geq C(E'')$.

Утверждение 1. *Ориентированный лес, определяемый букетом E'' , является максимальным в графе G'' .*

Доказательство. Пусть контур μ_{i-1} включает n узлов. В графе G'' для каждого узла найдется хотя бы одна заходящая дуга с положительным весом, в противном случае контур бы не образовался. Так как в графе G'' ровно n узлов, а именно вершины контура, имеют заходящие дуги, то ориентированный лес не может состоять более чем из n дуг. Вес ориентированного леса не превосходит суммарного веса дуг контура, т.к. контур строился из дуг максимального веса, входящих в его узлы. Но никакой лес не может содержать контур целиком, поэтому по крайней мере один узел y из контура либо не имеет заходящей дуги, либо дуга заходит в него из узла, не принадлежащего контуру.

Для всех узлов y из контура построим леса B_y по следующему правилу: удалим из контура дугу (t, y) , заходящую в y , и добавим дугу (x, y) максимального положительного веса из дуг, исходящих не из узлов контура и заходящих в y , если

таковые дуги имеются. В итоге вес дуг контура изменится на $C(x, y) - C(t, y) \leq 0$, и максимальный вес будет иметь лес B_y^* , для которого эта величина максимальна, т.е. величина $|C(x, y) - C(t, y)|$ минимальна. Следует учесть, что если $|C(x, y) - C(t, y)| \geq C(r, s)$, где (r, s) – дуга контура с наименьшим весом, то оказывается более выгодным просто удалить дугу (r, s) . Поэтому достаточно рассмотреть дуги, начинающиеся не в узлах контура, для которых $C(x, y) - C(t, y) + C(r, s) \geq 0$, и максимальный вес будет иметь лес B_y^* , для которого величина $C(x, y) - C(t, y) + C(r, s)$ максимальна. Из алгоритма следует, что лес B_y^* совпадает с E'' .

Любое изменение леса B_y^* будет состоять в том, что по крайней мере одна дуга (u, v) контура удаляется из B_y^* , и к нему возможно добавляется еще одна дуга (r, v) , исходящая из узла, не принадлежащего контуру. Такая модификация может лишь уменьшить вес леса B_y^* , поскольку изначально дуги контура выбирались как дуги максимального веса, заходящие в данные узлы. Следовательно, B_y^* , а значит и E'' – максимальный лес в графе G'' , что и требовалось доказать.

Утверждение 2. *Весы ориентированных лесов E' и B' равны.*

Доказательство. Рассмотрим два возможных случая:

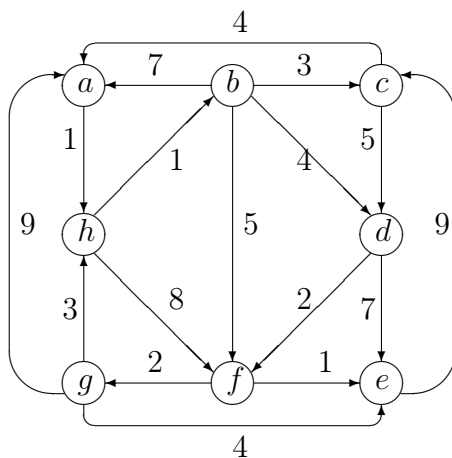
- а) лес E_i содержит дугу (x, v_{i-1}) ;
- а) лес E_i не содержит дуг (x, v_{i-1}) .

В случае а) $(x, v_{i-1}) \in E_i$. Значит, E'' и, в силу Утверждения 1, B'' содержит дугу (x, y) , где узел y принадлежит контуру μ_{i-1} , а x не принадлежит. Поскольку B является ориентированным лесом в графе G_{t-1} , то его подграф B' не может включать путь из какого-либо узла контура μ_{i-1} до узла x . Это же верно для леса E' . Так как лес B является к тому же максимальным, то B' имеет к тому же максимальный вес среди всех таких лесов.

Каждая дуга графа G' соответствует дуге графа G_i с таким же весом. Более того, каждому ориентированному лесу в G' соответствует ориентированный лес в G_i такого же веса. Поскольку лес E_i содержит дугу (x, v_{i-1}) , лес E' не содержит пути из какого-либо узла контура μ_{i-1} до узла x . Значит, $C(B') \geq C(E')$, и лес E_i не является максимальным в G_i .

В случае б) лесу E_i соответствует лес E' , и если E' – не максимальный лес в G' , то E_i – не максимальный лес в G_i .

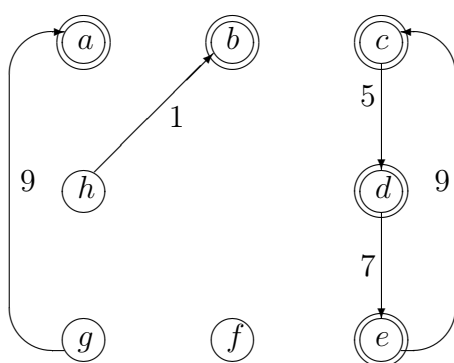
Пример. Рассмотрим ориентированный нагруженный граф. Около дуг представлены их веса.



Шаг 1. Выбираем узлы в алфавитном порядке и включаем их в букет узлов V_0 в букет дуг E_0 включаем дуги максимального положительного веса, заходящими в эти узлы. Последовательно получаем:

V_0	E_0	V_0	E_0	V_0	E_0	V_0	E_0	V_0	E_0
a	(g, a)	a	(g, a)	a	(g, a)	a	(g, a)	a	(g, a)
		b	(h, b)	b	(h, b)	b	(h, b)	b	(h, b)
				c	(e, c)	c	(e, c)	c	(e, c)
						d	(c, d)	d	(c, d)
								e	(d, e)

Результат представлен на диаграмме, узлы, включенные в букет узлов, выделены двойным контуром.

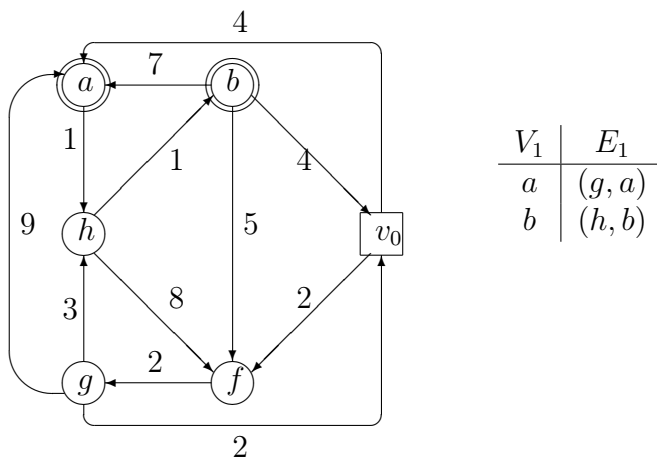


Поскольку множество E_0 содержит контур $ecde$, идем на шаг 2.

Шаг 2. Стягиваем контур $\mu_0 = ecde$ в узел v_0 . Минимальный вес дуги в этом контуре равен 5 и достигается на дуге (c, d) . Пересчитываем веса дуг, заходящих в v_0 :

$$\begin{aligned}
 C(b, v_0) &= C(b, c) - C(e, c) + C(c, d) = 3 - 9 + 5 = -1; \\
 C(b, v_0) &= C(b, d) - C(c, d) + C(c, d) = 4 - 5 + 5 = 4; \\
 C(f, v_0) &= C(f, e) - C(d, e) + C(c, d) = 1 - 7 + 5 = -1; \\
 C(g, v_0) &= C(g, e) - C(d, e) + C(c, d) = 4 - 7 + 5 = 2.
 \end{aligned}$$

В итоге получаем граф G_1 (дуги отрицательного веса удаляем из графа, поскольку они не включаются в остовный лес).

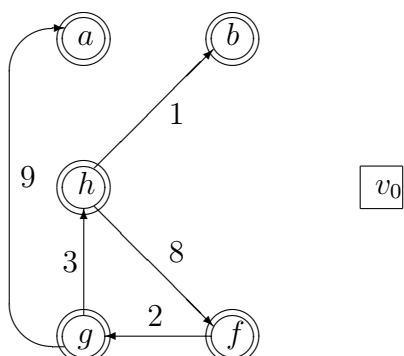


V_1	E_1
a	(g, a)
b	(h, b)

Шаг 1. Выбираем узлы в алфавитном порядке и включаем их в букет узлов V_1 в букет дуг E_1 включаем дуги максимального положительного веса, заходящими в эти узлы. Последовательно получаем:

V_1	E_1	V_1	E_1	V_1	E_1
a	(g, a)	a	(g, a)	a	(g, a)
b	(h, b)	b	(h, b)	b	(h, b)
f	(h, f)	f	(h, f)	f	(h, f)
		g	(f, g)	g	(f, g)
				h	(g, h)

Результат представлен на диаграмме.



Поскольку множество E_1 содержит контур $fgfh$, идем на шаг 2.

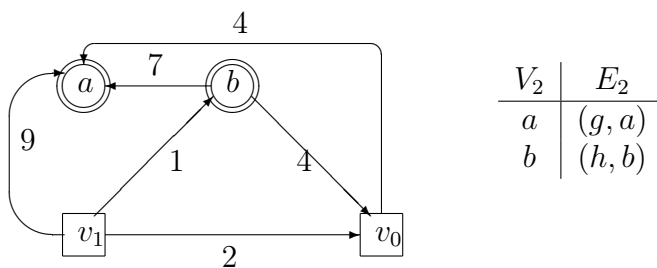
Шаг 2. Стягиваем контур $\mu_1 = fghf$ в узел v_1 . Минимальный вес дуги в этом контуре равен 2 и достигается на дуге (f, g) . Пересчитываем веса дуг, заходящих в v_1 :

$$C(a, v_1) = C(a, h) - C(g, h) + C(f, g) = 1 - 3 + 2 = 0;$$

$$C(b, v_1) = C(b, f) - C(h, f) + C(f, g) = 5 - 8 + 2 = -1;$$

$$C(v_0, v_1) = C(v_0, f) - C(v_0, f) + C(f, g) = 2 - 8 + 2 = -4.$$

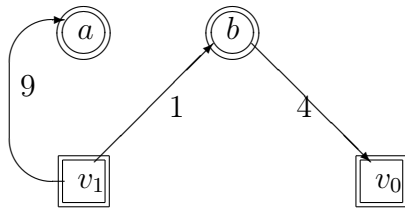
В итоге получаем граф G_2 (дуги отрицательного и нулевого веса удаляем из графа, поскольку они не включаются в остовный лес).



Шаг 1. Выбираем узлы в алфавитном порядке и включаем их в букет узлов V_1 в букет дуг E_1 включаем дуги максимального положительного веса, заходящими в эти узлы. Последовательно получаем:

V_2	E_2	V_1	E_1
a	(g, a)	a	(g, a)
b	(h, b)	b	(h, b)
v_0	(b, v_0)	v_0	(b, v_0)
		v_1	

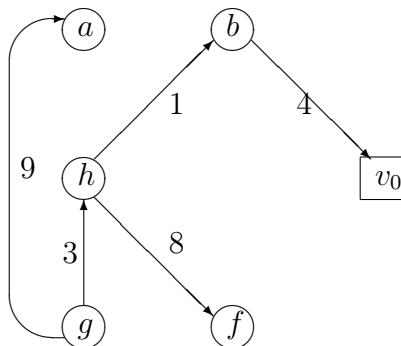
Результат представлен на диаграмме.



Все узлы включены в букет, идем на шаг 3.

Шаг 3. Поскольку $i = 2$ и узел v_1 является корнем ориентированного дерева в лесе, идем на шаг 4.

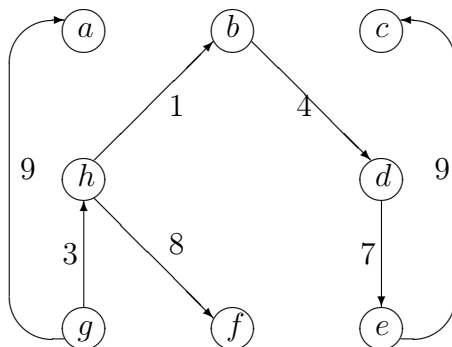
Шаг 4. Рассматриваем дуги из букета E_2 в совокупности с дугами контура μ_1 . Удаляем из множества дугу наименьшего веса (f, g) из контура μ_1 , получаем новый букет дуг E_1 , который образует в графе G_1 ориентированный лес.



Полагаем $i = 1$ и идем на шаг 3.

Шаг 3. Поскольку $i = 1$ и узел v_0 не является корнем ориентированного дерева в лесе, идем на шаг 5.

Шаг 5. В этом случае имеется дуга (b, v_0) , заходящая в узел v_0 . Эта дуга соответствует дуге (b, d) в графе G_0 , где узел d принадлежит контуру μ_0 . Рассматриваем дуги из букета E_1 в совокупности с дугами контура μ_0 . Это множество содержит ровно один контур μ_0 . В узел d заходит ровно две дуги: (b, d) и (c, d) – дуга из контура μ_0 . Удаляем (c, d) из множества дуг, получаем новый букет дуг E_0 , который образует в графе G_0 ориентированный лес.



Полагаем $i = 0$ и идем на шаг 3.

Шаг 3. Поскольку $i = 0$, то дуги букета E_0 образуют максимальный ориентированный лес в исходном графе, идем на конец.

Модификации алгоритма для решения других задач.

1. *Минимальный ориентированный лес* находится после изменения знака веса каждой дуги.

2. *Максимальное остовное дерево* находится после увеличения веса каждой дуги на достаточно большую величину.

3. *Минимальное остовное дерево* находится после изменения знака веса каждой дуги и затем увеличения веса каждой дуги на достаточно большую величину.

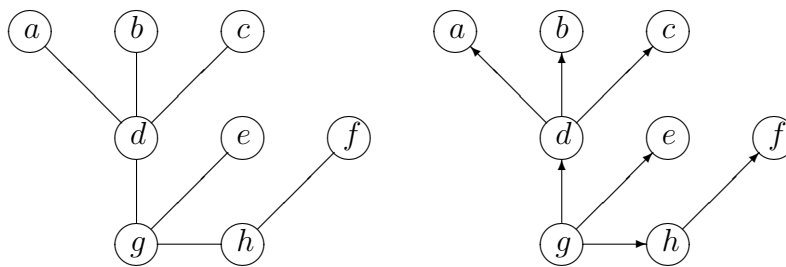
4, 5. *Максимальное (минимальное) остовное дерево с корнем в заданном узле a* находится после добавления узла a' и дуги (a, a') . Затем в полученном графе находится максимальное (минимальное) остовное дерево.

5.5 Способы задания деревьев

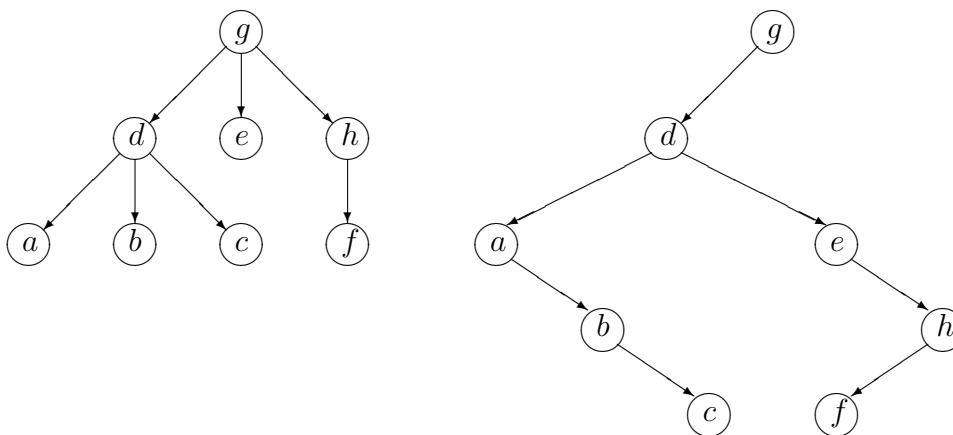
Деревья, как и все графы, могут быть заданы множествами вершин и ребер, списками смежности, матрицами смежности и инциденций. Кроме этого, существуют специфические способы представления деревьев.

Всякое свободное дерево можно ориентировать, назначив один из узлов корнем, как это показано в теореме о свойствах ордерезьев. Всякое ордерезье можно произвольно упорядочить. Всякое упорядоченное дерево можно представить бинарным деревом, проведя правую связь к старшему брату, а левую – к младшему сыну.

Пример. Слева представлена диаграмма свободного дерева, справа – диаграмма полученного из него ориентированного дерева.



На следующем рисунке слева – упорядоченное дерево (порядок поддерезьев – слева направо), справа – бинарное дерево.



Структура бинарного дерева, конечно, совершенно иная, чем у упорядоченного. Тем не менее, все связи, существующие в упорядоченном дереве, могут быть восстановлены из бинарного. Для того, чтобы найти всех сыновей узла x , необходимо рассмотреть левую связь этого узла (пусть она ведет в узел y), по ней находится младший сын. Далее, рассматривая последовательно правых потомков узла y , можно перечислить всех братьев узла y , а значит, и сыновей x .

Пример. В предыдущем примере сыновьями узла d в упорядоченном дереве являются узлы a (в него ведет левая связь из d), b (в него ведет правая связь из a) и c (в него ведет правая связь из b).

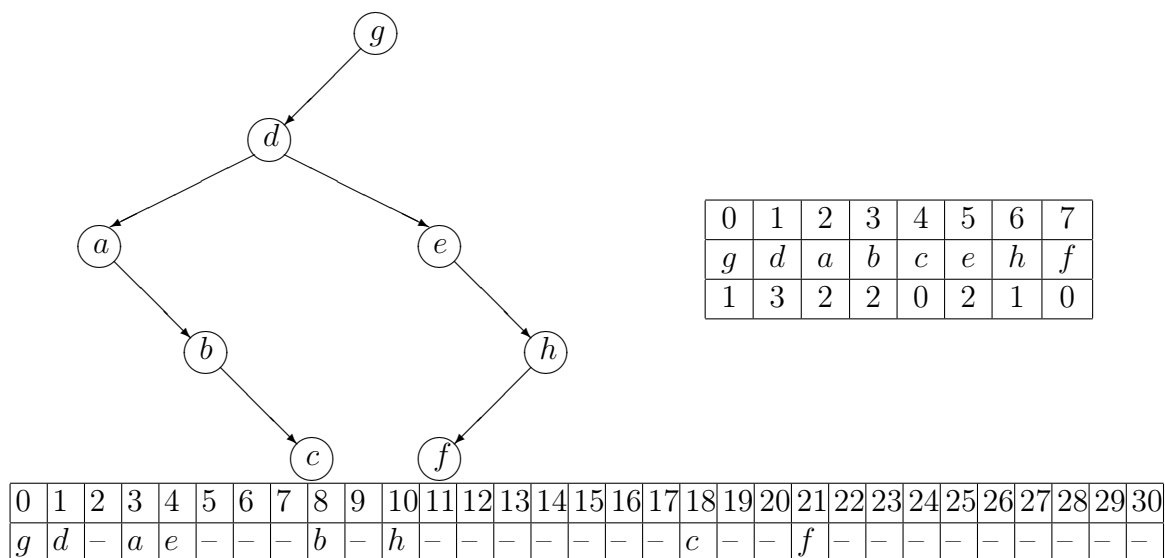
Таким образом, достаточно рассмотреть представление бинарных деревьев. Основные способы представления бинарных деревьев таковы.

1. *Списочные структуры.* Каждый узел представляется записью, содержащей ссылки на левое и правое поддеревья и информацию об узле.

2. *Массивы.* Все узлы располагаются в массиве так, что если узел имеет номер i , то корень левого поддерева имеет номер $2i + 1$, а корень правого $- 2i + 2$ (при этом предполагается, что нумерация в массиве начинается с нуля). Если какой-либо узел отсутствует в дереве, в массиве он помечается специальным значением. Согласно лемме о числе узлов в бинарном дереве, число элементов массива равно $2^{h+1} - 1$, где h – высота дерева.

3. *Польская запись.* Все узлы располагаются в массиве так, что все узлы поддерева расположены сразу после его корня. Вместе с информацией об узле хранится «размеченная степень» каждого узла (например, 0 – лист, 1 – есть левая связь, но нет правой, 2 – есть правая связь, но нет левой, 3 – есть обе связи). При этом число элементов массива равно числу узлов.

Пример. Рассмотрим бинарное дерево из предыдущего примера. Слева графически представлена списочная структура, справа – польская запись, снизу – массив. Отсутствие узла обозначается специальным символом –.



В данном случае массив содержит большое число пустых ячеек. Таким образом, использовать массивы выгодно, если число узлов яруса близко к максимальному значению.

5.6 Деревья сортировки

Определение. *Дерево сортировки* – бинарное дерево, каждый узел которого содержит ключ, причем значения ключей во всех узлах левого поддерева меньше (меньше или равно), а значения ключей во всех узлах правого поддерева больше (больше или равно), чем значение ключа в корне поддерева.

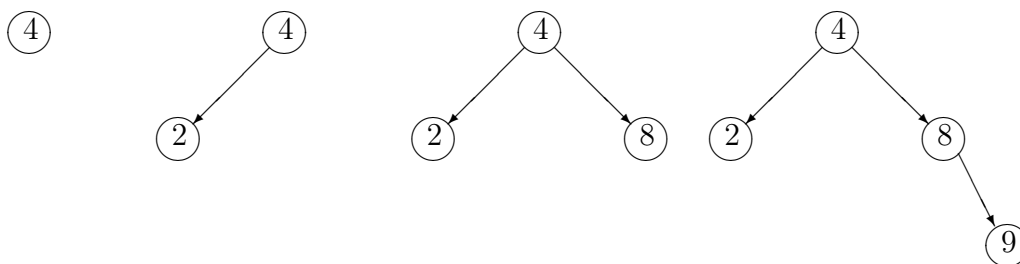
Деревья сортировки предназначены для хранения упорядоченной информации (так же как массивы, файлы, списки). Они позволяют организовать три основные операции по работе с информацией – поиск, вставка и удаление – в среднем более эффективно, чем остальные структуры. Далее мы покажем, что вычислительная сложность любой операции не превышает высоты дерева, которая при применении определенных алгоритмов работы с деревом не превышает порядка $\log_2 p$, где p – количество ключей).

Поиск. В соответствии с определением дерева сортировки, поиск осуществляется следующей рекурсивной процедурой:

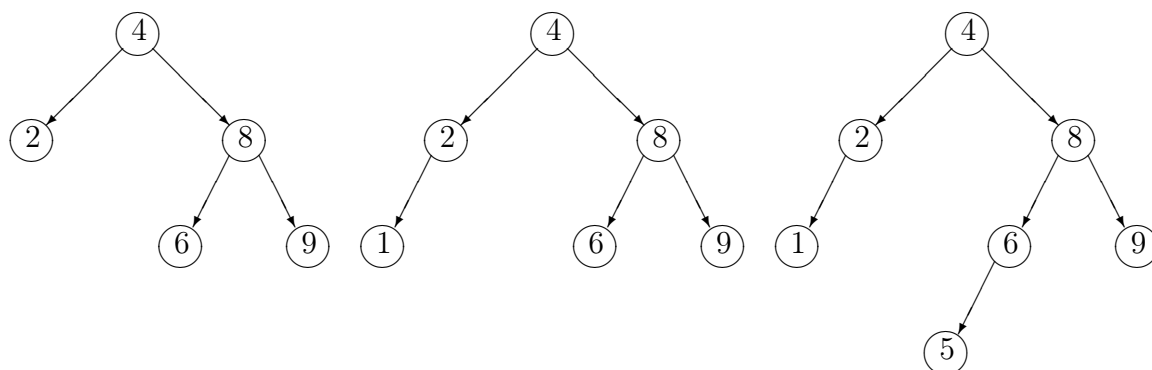
- если искомое значение равно ключу в рассматриваемом узле, значение найдено, конец;
- если искомое значение меньше ключа в рассматриваемом узле, и левое поддерево пусто, искомого значения нет в дереве, если левое поддерево не пусто, продолжаем поиск в нем;
- если искомое значение больше ключа в рассматриваемом узле, и правое поддерево пусто, искомого значения нет в дереве, если правое поддерево не пусто, продолжаем поиск в нем.

Вставка. Алгоритм аналогичен алгоритму поиска: в дереве ищется такой узел, имеющий свободную связь для вставки нового узла, чтобы не нарушалось условие дерева сортировки. А именно, если новое значение меньше текущего ключа, то новый узел можно подцепить слева, если левая связь свободна, либо найти в левом поддереве подходящее место. Если новый ключ больше текущего, поступаем аналогично, но рассматриваем правое поддерево.

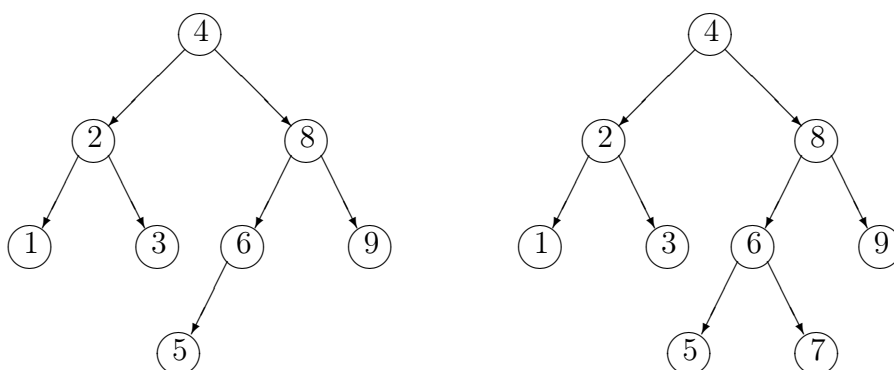
Пример. Построим дерево поиска, последовательно добавляя в него узлы с ключами 4, 2, 8, 9, 6, 1, 5, 3, 7. Добавляем 4 в корень дерева. Далее добавляем 2, сравниваем 2 и 4. Поскольку $2 < 4$, рассматриваем левую связь узла 4, эта связь пуста, значит, добавляем 2 слева от 4. Далее добавляем 8, сравниваем 8 и 4. Поскольку $8 > 4$, рассматриваем правую связь узла 4, эта связь пуста, значит, добавляем 8 справа от 4. Далее добавляем 9, сравниваем 9 и 4. Поскольку $9 > 4$, рассматриваем правую связь узла 4, эта связь не пуста, значение ключа в узле равно 8. Сравниваем 9 и 8. Поскольку $9 > 8$, рассматриваем правую связь узла 8, эта связь пуста, значит, добавляем 9 справа от 8.



Далее добавляем 6, сравниваем 6 и 4. Поскольку $6 > 4$, рассматриваем правую связь узла 4, эта связь не пуста, значение ключа в узле равно 8. Сравниваем 6 и 8. Поскольку $6 < 8$, и левая связь узла 8 пуста, добавляем 6 слева от 8. Далее добавляем 1, сравниваем 1 и 4. Поскольку $1 < 4$, рассматриваем левую связь узла 4, эта связь не пуста, значение ключа в узле равно 2. Сравниваем 1 и 2. Поскольку $1 < 2$, и левая связь узла 2 пуста, добавляем 1 слева от 2. Далее добавляем 5, сравниваем 5 и 4. Поскольку $5 > 4$, рассматриваем правую связь узла 4, эта связь не пуста, значение ключа в узле равно 8. Сравниваем 5 и 8. Поскольку $5 < 8$, рассматриваем левую связь узла 8, эта связь не пуста, значение ключа в узле равно 6. Сравниваем 5 и 6. Поскольку $5 < 6$, и левую связь узла 6 пуста, добавляем 5 слева от 6.



Действуя аналогично, добавляем 3 и 7. Дерево построено.

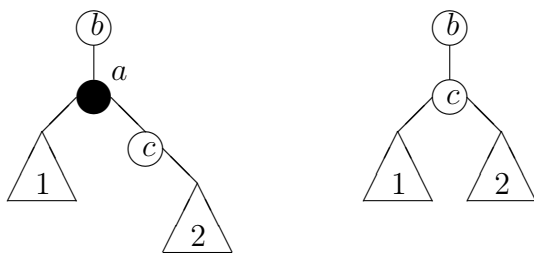


Удаление. Удаление узла требует перестройки дерева сортировки. При этом возможны три случая.

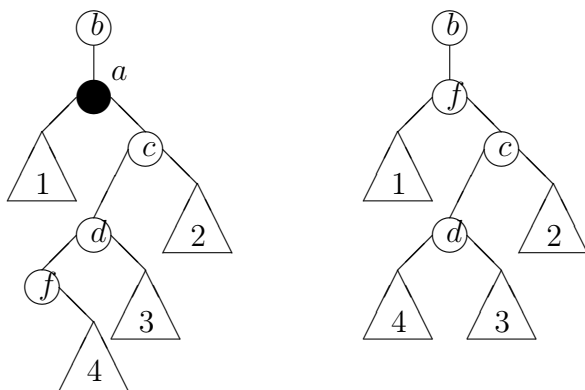
Случай 1. Правая связь удаляемого узла a пуста. В этом случае левое поддерево 1 узла a подцепляется к родительскому узлу b с той же стороны, с которой был подцеплен узел a . В частности, если a – лист, левая связь родительского узла b становится пустой.



Случай 2. Правая связь удаляемого узла a не пуста и ведет в узел c , левая связь которого пуста. В этом случае левое поддерево 1 узла a подцепляется к узлу c слева, а сам узел c подцепляется к родительскому узлу b с той же стороны, с которой был подцеплен узел a .

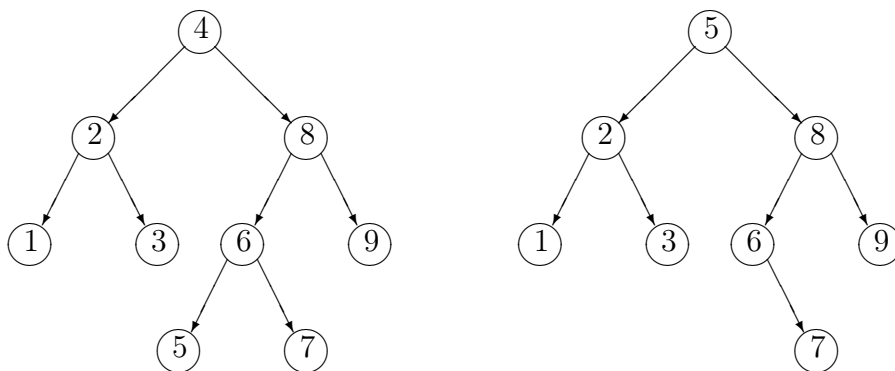


Случай 3. Правая связь удаляемого узла a не пуста и ведет в узел c , левая связь которого не пуста. В этом случае находим самый левый узел d правого поддерева узла a , левая связь у f пуста. Заменяем ключ узла a на ключ узла f , а правое поддерево 4 узла f подцепляем на место f (слева от d).

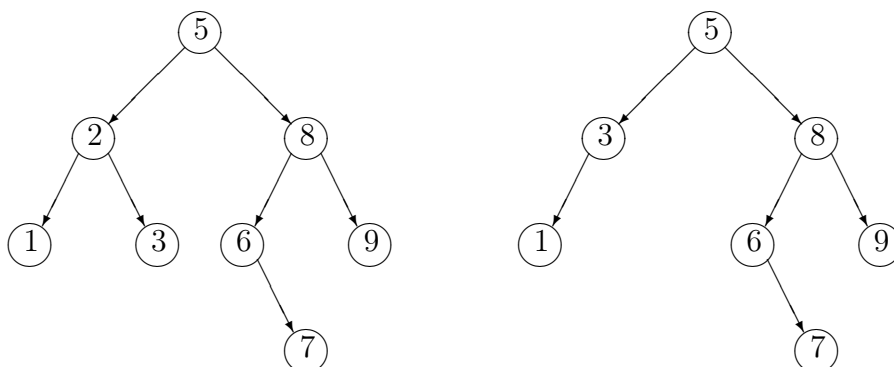


Существуют также три симметричных случая, которые получаются из рассмотренных заменой правой связи на левую и левой на правую. С учетом этого действия в таких случаях аналогичны рассмотренным.

Пример. Рассмотрим дерево из предыдущего примера. Удалим корень дерева – узел 4. У этого узла обе связи не пусты, при этом левый потомок узла 4 имеет непустую правую связь, а правый – непустую левую связь. Значит, имеет место случай 3. Узел 4 заменяем на самый левый узел в его правом поддереве, это узел 5. Затем удаляем узел 5, который является листом, так что имеет место случай 1. При удалении узла 5 левая связь родительского узла 6 становится пустой.



Далее удалим узел 2. У этого узла обе связи не пусты, при этом правый потомок узла 2, а именно, узел 3, имеет пустую левую связь. Значит, имеет место случай 2. Левое поддерево узла 2 (поддерево с корнем 1) становится левым поддеревом узла 3, узел 3 становится на место узла 2, а узел 2 удаляется.



Обратим внимание, что, как при вставке, так и при удалении узлов свойства дерева сортировки не нарушались: все ключи левого поддерева меньше ключа корня, а все ключи правого поддерева больше ключа корня.

Вычислительная сложность алгоритмов. При поиске, вставке и удалении в худшем случае просматриваются все узлы одной ветви дерева, поэтому вычислительная сложность всех этих операций пропорциональна высоте дерева.

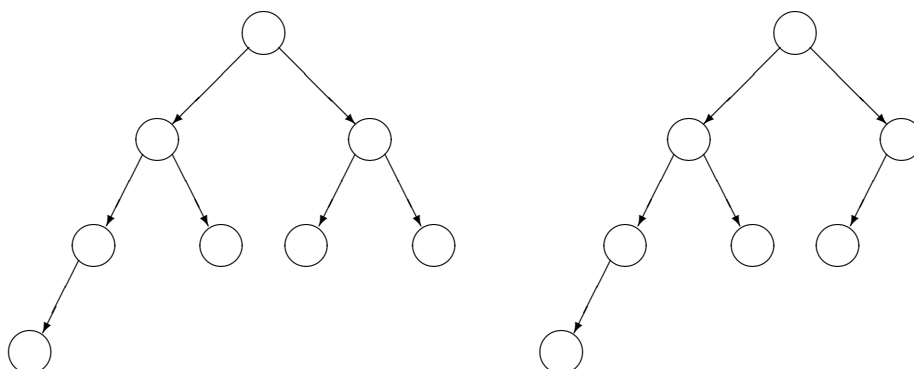
Понятно, что высота дерева сортировки, а значит, и вычислительная сложность работы с информацией, зависит от порядка добавления новых данных. Если эти данные уже упорядочены, дерево превращается в связный список, и его высота становится равной числу узлов дерева p . Поэтому рассмотрим некоторые модификации деревьев сортировки, высота которых имеет порядок $\log_2 p$.

5.6.1 Выровненные деревья

Определение. Ордерево называется *выровненным*, если все листья располагаются на одном или двух последних уровнях.

Выровненное дерево имеет наименьшую возможную для данного p высоту h .

Пример. Слева – выровненное дерево, справа – не выровненное.



Теорема. Для выровненного бинарного дерева $\log_2(p+1) - 1 \leq h < \log_2(p+1)$.

Доказательство. На i -м уровне находится самое большое 2^i вершин, следовательно,

$$\sum_{i=0}^{h-1} 2^i < p \leq \sum_{i=0}^{h-1} 2^i.$$

Используя формулу для суммы геометрической прогрессии, получаем $2^h - 1 < p \leq 2^{h+1} - 1$, значит, $2^h < p + 1 \leq 2^{h+1}$. Логарифмируя, имеем $h < \log_2(p+1)$ и $h+1 \geq \log_2(p+1)$, отсюда получаем утверждение теоремы.

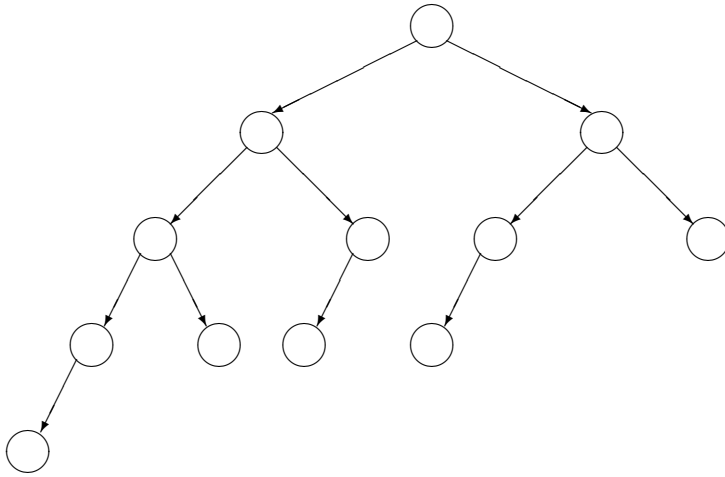
Так как выровненные деревья оптимальны по высоте, они дают наибольший эффект при поиске данных. Однако известно, что вставка или удаление вершины может потребовать перестройки всего дерева, и таким образом в худшем случае трудоемкость этих операций будет равна p . Поэтому рассмотрим другие варианты организации деревьев поиска, высота которых не более чем в два раза больше высоты выровненного дерева, но при этом поиск, добавление и удаление узла требуют просмотра не более чем одной ветви дерева, т.е. вычислительная сложность операций имеет порядок $\log_2 p$.

5.6.2 Сбалансированные деревья

Определение. Бинарное дерево называется *сбалансированным*, если для любого узла высота левого и правого поддеревьев отличается не больше чем на 1.

Сбалансированные деревья называются также *АВЛ-деревьями* (Адельсон-Вельский и Ландис).

Пример. Максимально несимметричное сбалансированное дерево.



Теорема. Для сбалансированного бинарного дерева $h < 2 \log_2 p$.

Доказательство. Рассмотрим наиболее несимметричные сбалансированные деревья, например, такие, у которых всякое левое поддерево на 1 выше правого (как показано на рисунке выше). Такие деревья имеют максимально возможную высоту среди всех сбалансированных деревьев с данным числом вершин. Пусть P_h – число вершин в наиболее несимметричном сбалансированном дереве высоты h . По построению дерева имеем $p = P_h = P_{h-1} + P_{h-2} + 1$. Покажем методом математической индукции, что $P_h \geq (\sqrt{2})^h$. База: $P_0 = 1 = (\sqrt{2})^0$, $P_1 = 2 \geq (\sqrt{2})^1 = \sqrt{2}$.

$P_2 = 4 \geq (\sqrt{2})^2 = 2$. Пусть утверждение верно для $0, 1, \dots, h$, тогда

$$P_h = P_{h-1} + P_{h-2} + 1 \geq (\sqrt{2})^h + (\sqrt{2})^{h-1} + 1 = (\sqrt{2})^h \left(1 + \frac{1}{\sqrt{2}} + \frac{1}{(\sqrt{2})^h} \right) >$$

$$> (\sqrt{2})^h \left(1 + \frac{1}{\sqrt{2}} \right) > (\sqrt{2})^h \sqrt{2} = (\sqrt{2})^{h+1}.$$

Отсюда имеем $p = P_h \geq (\sqrt{2})^h = 2^{h/2}$. Логарифмируя, получаем утверждение теоремы.

Поиск в сбалансированном дереве производится так же, как и в обычном дереве поиска.

Вставка выполняется так же, как и для дерева поиска. После вставки узла проверяется, осталось ли дерево сбалансированным. Если нет, производится балансировка узла, для которого нарушена сбалансированность.

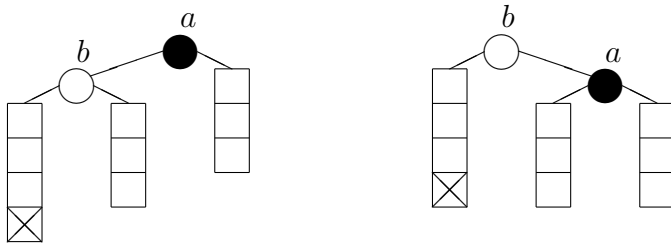
Удаление выполняется так же, как и для дерева поиска. После удаления узла проверяется, сбалансировано ли дерево. Если нет, производится балансировка в худшем случае всех узлов, лежащих на пути от корня до удаляемого узла.

Балансировка узла выполняется, если высота его левого и правого поддеревьев отличается больше, чем на 1 (т. к. балансировка производится после вставки или удаления одного узла, высота поддеревьев отличается на 2). Например, при вставке возможны следующие 4 варианта: новая вершина удлиняет:

- левое поддерево левого поддерева узла (*LL*);
- левое поддерево правого поддерева узла (*RL*);
- правое поддерево левого поддерева узла (*LR*);
- правое поддерево правого поддерева узла (*RR*).

Ситуации *LL/RR* и *RL/LR* аналогичны с точностью до симметрии, поэтому рассмотрим по одной из них.

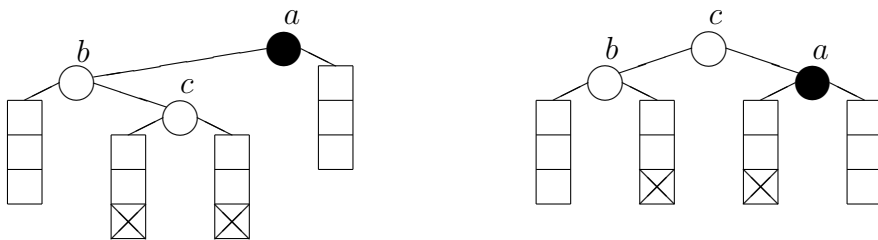
- *LL*-поворот



Пусть необходимо сбалансировать узел a , а левая связь этого узла указывает на узел b . В результате:

- узел b подцепляется на место узла a ;
- правое поддерево узла b становится левым поддеревом узла a ;
- узел a подцепляется к узлу b справа.

- *LR*-поворот



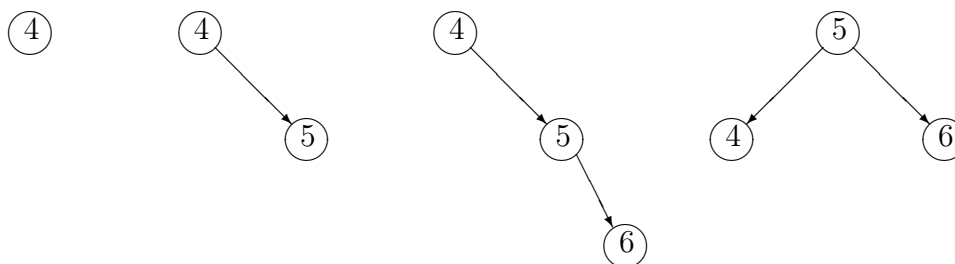
Пусть необходимо сбалансировать узел a , а левая связь этого узла указывает на узел b , правая связь которого указывает на узел c . В результате:

- узел c подцепляется на место узла a ;
- правое поддерево узла c становится левым поддеревом узла a ;
- левое поддерево узла c становится правым поддеревом узла b ;
- узел a подцепляется к узлу c справа;
- узел b подцепляется к узлу c слева.

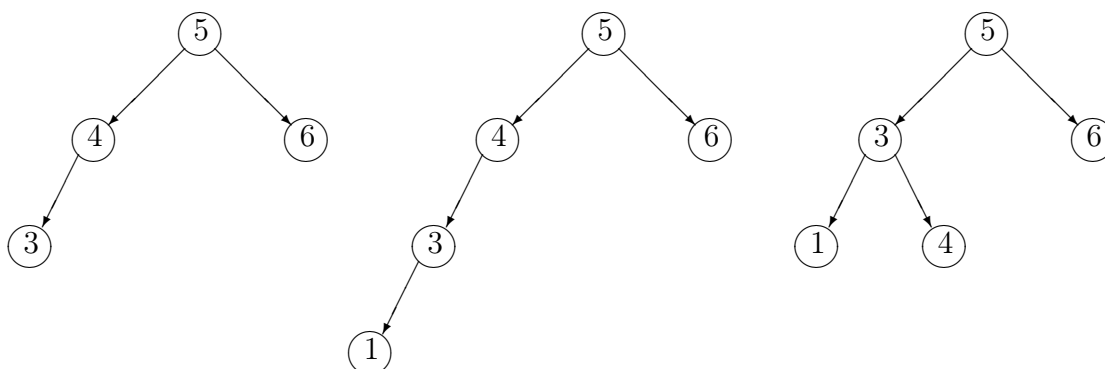
В обоих случаях движение узлов происходит только «по вертикали», т. е. условие дерева поиска не нарушается. Отметим, что поворот выполняется за конечное число операций, и балансировка требуется в худшем случае для всех вершин одной ветви дерева, т.е. вычислительная сложность алгоритма остается пропорциональной высоте дерева, т.е. $\log_2 p$.

Пример. Последовательно добавим в сбалансированное дерево ключи 4, 5, 6, 3, 1, 9, 7, 8, 2.

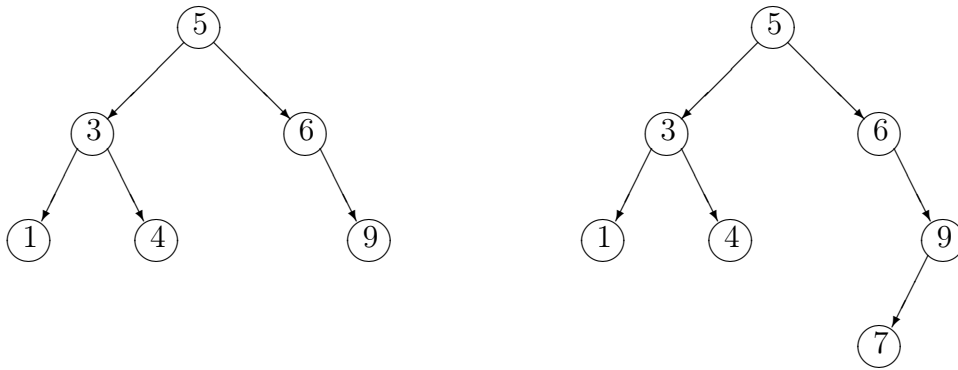
Добавляем 4 в корень дерева. Далее добавляем 5 справа от 4 и 6 справа от 5. Узел 4 перестал быть сбалансированным, добавленный узел удлинил правое поддерево правого поддерева узла 4, значит, выполняем RR -поворот. Узел 5 становится корнем, узлы 4 и 6 – его сыновьями. Процесс построения дерева показан ниже.



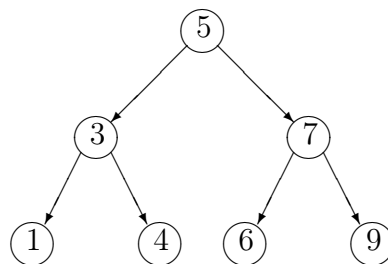
Добавляем 3 слева от 4 и 1 слева от 3. Узел 4 оказывается несбалансированным, последний добавленный узел 1 удлиняет левое поддерево левого поддерева узла 4, значит, выполняем LL -поворот. Узел 3 становится на место узла 4, его сыновья теперь – узлы 1 и 4.



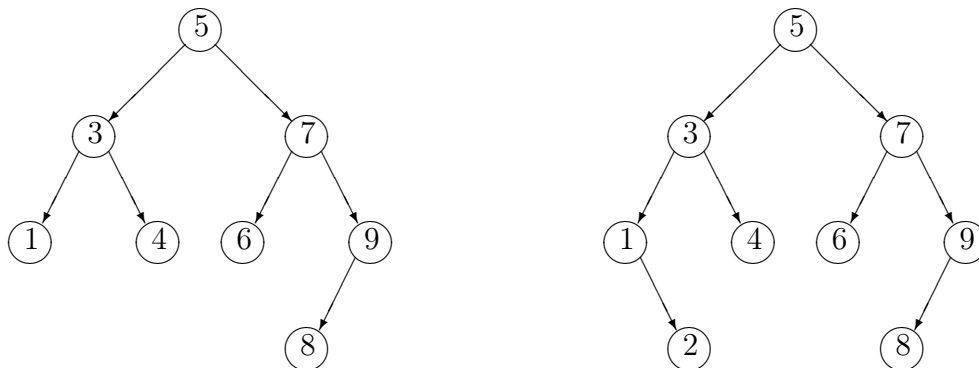
Добавляем 9 справа от 6 и 7 слева от 9. Результат представлен на следующей странице.



Узел 6 оказывается несбалансированным, последний добавленный узел 7 удлиняет правое поддерево левого поддерева узла 6, значит, выполняем *LR*-поворот. Узел 7 становится на место узла 6, его сыновья теперь – узлы 6 и 9.



Добавляем 8 слева от 9 и 2 справа от 1. При добавлении этих узлов балансировка не нарушается, дерево построено.

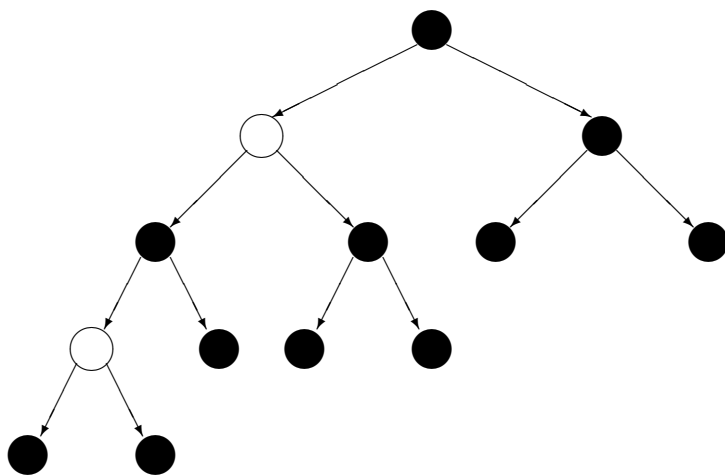


5.6.3 Красно-черные деревья

Красно-черное дерево – двоичное дерево поиска, в котором каждый узел имеет атрибут «цвет», принимающий значения красный или черный. Листья красно-черных деревьев не содержат данных. Такие листья не нуждаются в явном выделении памяти – нулевой указатель на потомка может фактически означать, что этот потомок – лист, но в некоторых случаях работы с красно-черными деревьями использование явных листовых узлов может послужить упрощением алгоритма. Данные содержатся во внутренних узлах дерева, т.е. в узлах, не являющихся листьями. В дополнение к обычным требованиям, налагаемым на двоичные деревья поиска, к красно-черным деревьям применяются следующие требования:

1. Узел либо красный, либо черный.
2. Корень – черный. (В других определениях это правило иногда опускается. Это правило слабо влияет на анализ, так как корень всегда может быть изменен с красного на черный, но не обязательно наоборот).
3. Все листья – черные.
4. Оба сына каждого красного узла – черные.
5. Всякий простой путь от данного узла до любого листа, являющегося его потомком, содержит одинаковое число черных узлов.

Пример. Максимально несимметричное сбалансированное дерево (здесь и далее красные узлы обозначаются белыми кружками, черные узлы – черными, узлы, которые могут быть любого цвета – штрихованными).



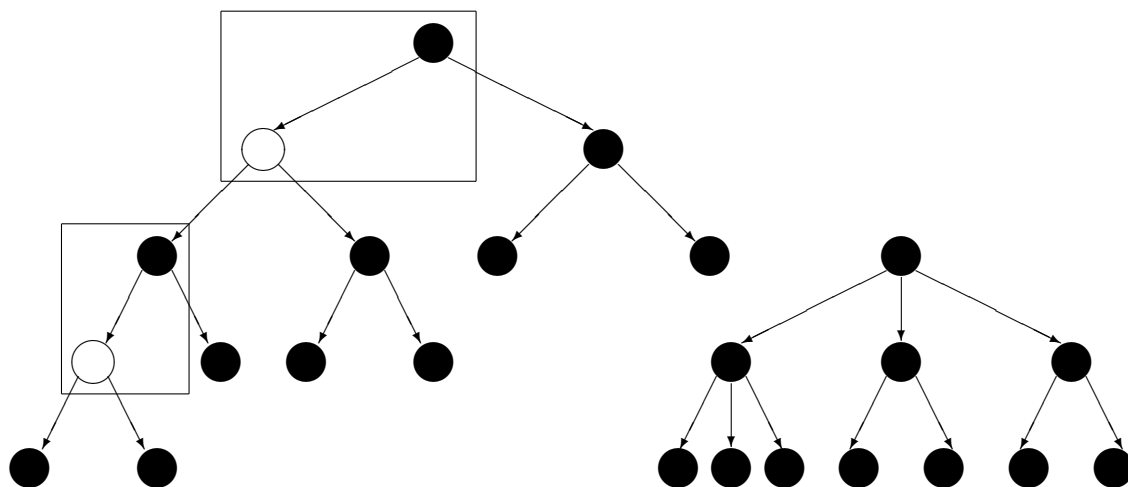
Эти ограничения реализуют критическое свойство красно-черных деревьев: путь от корня до самого дальнего листа не более чем в два раза длиннее пути от корня до ближайшего листа. Результатом является то, что дерево примерно сбалансировано. Так как такие операции как вставка, удаление и поиск значений требуют в худшем случае времени, пропорционального длине дерева, эта теоретическая верхняя граница высоты позволяет красно-черным деревьям быть более эффективными в худшем случае, чем обычные двоичные деревья поиска.

Чтобы понять, почему это гарантируется, достаточно рассмотреть эффект свойства 4 и 5 вместе. Пусть для красно-черного дерева T число черных узлов в свойстве 5 равно m . Тогда кратчайший возможный путь от корня дерева T до любого листа содержит m черных узлов. Более длинный возможный путь может быть построен путем включения красных узлов. Однако, свойство 4 не позволяет вставить несколько красных узлов подряд. Поэтому самый длинный возможный путь состоит из $2m$ узлов, попеременно красных и черных. Любой максимальный путь имеет одинаковое число черных узлов (по свойству 5), следовательно, не существует пути, более чем вдвое длиннее, чем любой другой путь.

Теорема. Для красно-черного бинарного дерева $h < 2 \log_2 p$, где p – число внутренних узлов.

Доказательство. Рассмотрим произвольное красно-черное дерево. Объединим каждый красный узел с его отцом. В результате получим упорядоченное дерево, полустепень исхода каждого нелистового узла которого равна 2, 3 или 4. Про-

иллюстрируем это на дереве из предыдущего примера. Здесь слева – исходное дерево, справа – дерево после преобразования.



По Свойству 5 все ветви преобразованного дерева имеют одинаковую высоту k . Число листьев в преобразованном дереве не меньше, чем число узлов на уровне k в бинарном дереве высоты k , по лемме о числе узлов в бинарном дереве, оно равно 2^{k+1} . С другой стороны, число листьев в преобразованном дереве равно числу листьев в красно-черном дереве, а поскольку полустепень исхода каждого узла красно-черного дерева равна 0 или 2, то, по лемме о числе листьев в бинарном дереве, оно равно $p + 1$. Отсюда

$$p + 1 \geq 2^{k+1}, \quad k \leq \log_2(p + 1) - 1 < \log_2 p.$$

Поскольку высота h исходного красно-черного дерева не более чем в два раза больше k , то $h < 2 \log_2 p$.

Поиск для красно-черного дерева ничем не отличается от поиска для бинарного дерева поиска.

Вставка начинается с добавления узла, точно так же, как и в обычном бинарном дереве поиска, и окрашивания его в красный цвет. Но если в бинарном дереве поиска мы всегда добавляем лист, в красно-черном дереве листья не содержат данных, поэтому мы добавляем красный внутренний узел с двумя черными фиктивными листьями на место черного листа. Что происходит дальше – зависит от цвета близлежащих узлов. Термин «дядя» будем использовать для обозначения брата отца, как и в фамильном дереве. Заметим, что:

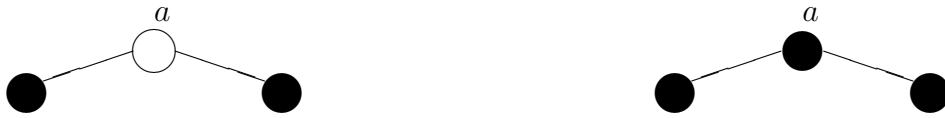
Свойство 3 (все листья черные) выполняется всегда.

Свойство 4 (оба сына любого красного узла – черные) может нарушиться только при добавлении красного узла, при перекрашивании черного узла в красный или при повороте.

Свойство 5 (все пути от любого узла до листовых узлов содержит одинаковое число черных узлов) может нарушиться только при добавлении черного узла, перекрашивании красного узла в черный (или наоборот), или при повороте.

Случай 1. Текущий узел a в корне дерева. В этом случае, он перекрашивается в черный цвет, чтобы оставить верным Свойство 2 (корень – черный). Так как это действие добавляет один черный узел в каждый путь, Свойство 5 (все пути

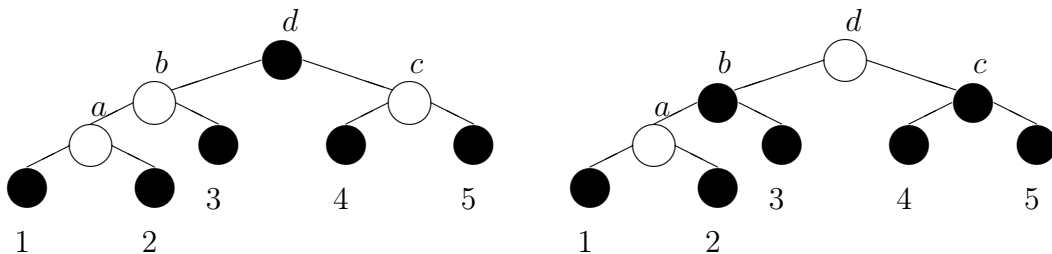
от любого данного узла до листовых узлов содержат одинаковое число черных узлов) не нарушается.



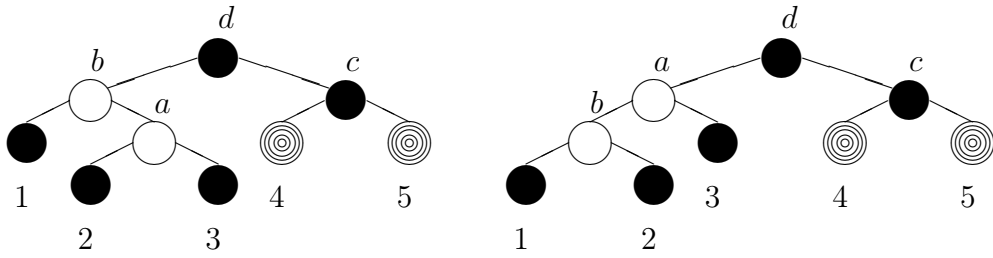
Случай 2. Отец b текущего узла a черный, то есть Свойство 4 (оба сына каждого красного узла – черные) не нарушается. Свойство 5 (все пути от любого данного узла до листовых узлов содержат одинаковое число черных узлов) не нарушается, потому что текущий узел a имеет два черных сына, которые являются листьями, но так как a является красным, пути до каждого из этих листьев содержат такое же число черных узлов, что и путь до черного листа d , который был заменен узлом a , так что свойство остается верным.



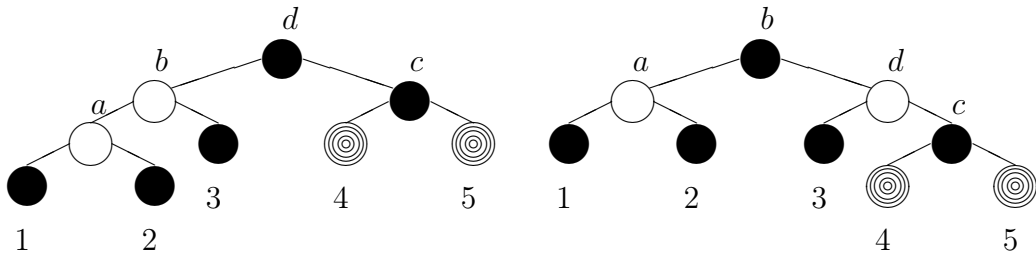
Случай 3. Если и для текущего узла a и отец b и дядя c – красные, то они оба могут быть перекрашены в черный, и дедушка d станет красным для сохранения Свойства 5 (все пути от любого данного узла до листовых узлов содержат одинаковое число черных узлов). Теперь у текущего красного узла a черный родитель. Так как любой путь через родителя или дядю должен проходить через дедушку, число черных узлов в этих путях не изменится. Однако, дедушка узла a – узел d теперь может нарушить свойства 2 (корень – черный) или 4 (оба сына каждого красного узла – черные). Свойство 4 может быть нарушено, так как родитель узла d может быть красным. Чтобы это исправить, вся процедура рекурсивно выполняется для узла d .



Случай 4. Отец b текущего узла a является красным, но дядя c – черный. Также, текущий узел a – правый сын b , а b в свою очередь – левый сын своего отца d . В этом случае может быть произведен поворот дерева, который меняет роли текущего узла a и его отца b . Тогда бывший родительский узел b рассматривается, используя случай 5, потому что Свойство 4 (оба сына любого красного узла – черные) все еще нарушено. Вращение приводит к тому, что некоторые пути (в поддереве, обозначенном «1» на схеме) проходят через узел a , чего не было до этого. Это также приводит к тому, что некоторые пути (в поддереве, обозначенном «3») не проходят через узел b . Однако, оба этих узла являются красными, так что Свойство 5 (все пути от любого данного узла до листовых узлов содержат одинаковое число черных узлов) не нарушается при вращении.

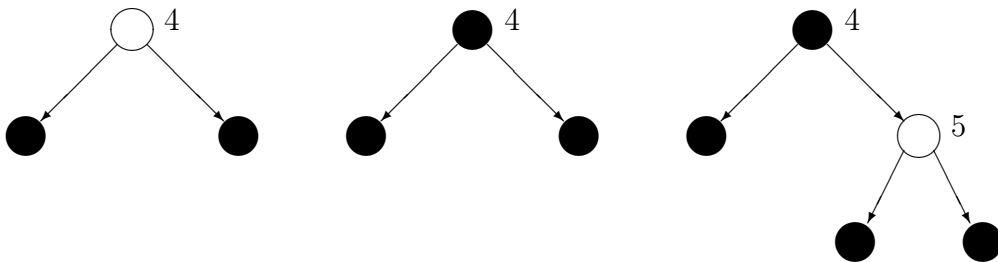


Случай 5. Отец b текущего узла a является красным, но дядя c – черный, текущий узел a – левый сын b и b – левый сын d . В этом случае выполняется поворот дерева. В результате получается дерево, в котором бывший отец b теперь является отцом и текущего узла a и бывшего дедушки d . Известно, что d – черный, так как его бывший сын b не мог бы в противном случае быть красным (без нарушения Свойства 4). Тогда цвета b и d меняются и в результате дерево удовлетворяет Свойству 4 (оба сына любого красного узла – черные). Свойство 5 (все пути от любого данного узла до листовых узлов содержат одинаковое число черных узлов) также остается верным, так как все пути, которые проходят через любой из этих трех узлов, ранее проходили через d , поэтому теперь они все проходят через b . В каждом случае из этих трех узлов только один окрашен в черный.

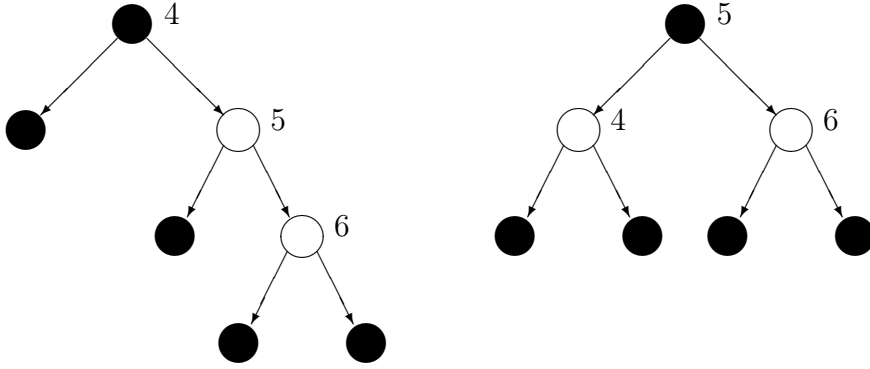


Пример. Последовательно добавим в красно-черное дерево ключи 4, 5, 6, 3, 1, 9, 7, 8, 2. Напомним, что добавляемый узел всегда красный и имеет двух черных сыновей, которые являются листьями и не содержат данных.

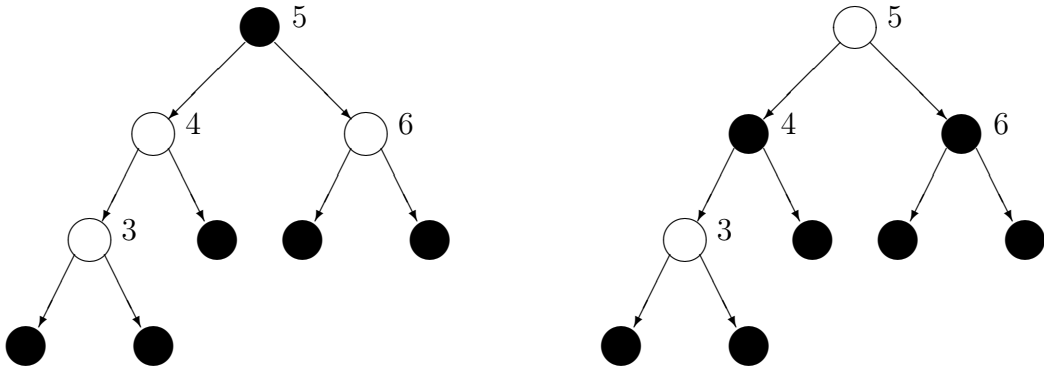
Добавляем 4 в корень дерева, это случай 1. Перекрашиваем узел 4 в черный цвет. Далее добавляем 5 справа от 4, у узла 5 черный отец, значит, имеет место случай 2, и дерево не требует перестройки и перекрашивания узлов.



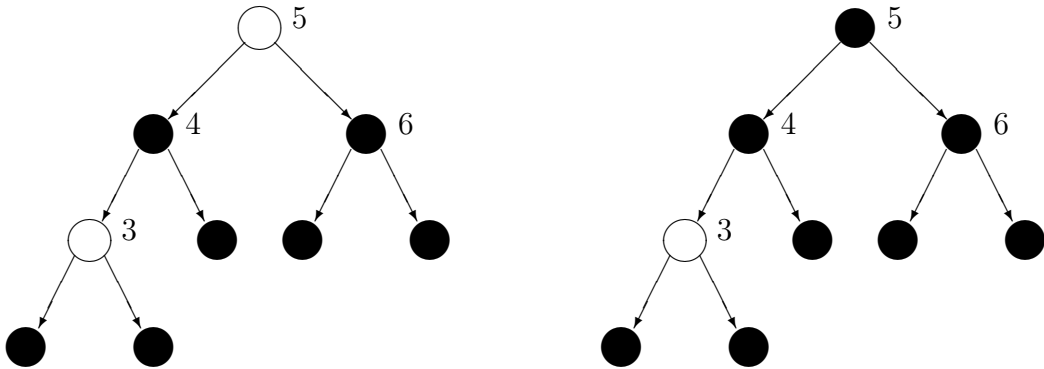
Далее добавляем 6 справа от 5, у узла 6 красный отец 5, черный дядя (лист), 6 – правый сын 5, и 5 – правый сын 4, значит, данный случай симметричен случаю 5. Выполняем поворот дерева, узел 5 становится отцом узлов 4 и 6, а левое поддереву узла 5 – правым поддеревом узла 4. Узлы 4 и 6 окрашиваем в красный, а узел 5 – в черный цвет. Процесс изменения дерева показан на следующей странице.



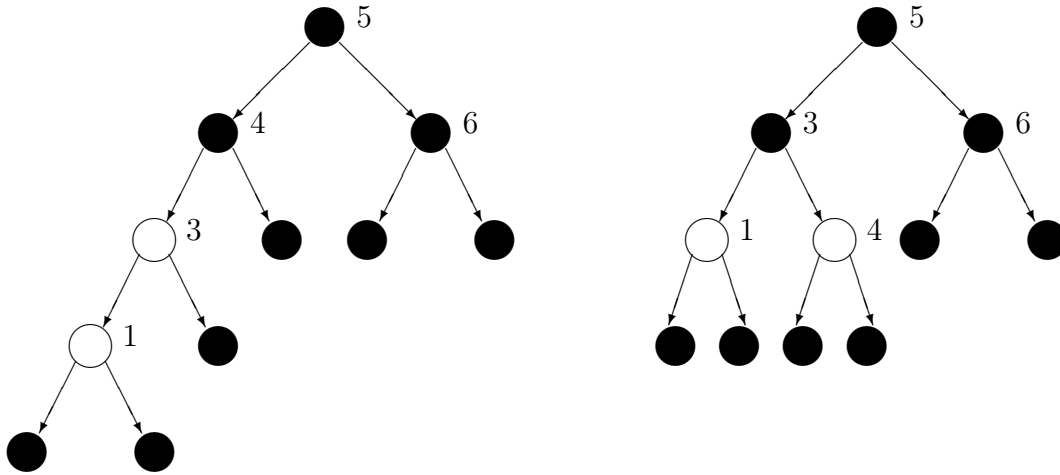
Далее добавляем 3 слева от 4, у узла 3 красный отец 4 и красный дядя 6, значит, имеет место случай 3. Перекрашиваем узлы 4 и 6 в черный цвет, а их отца 5 – в красный.



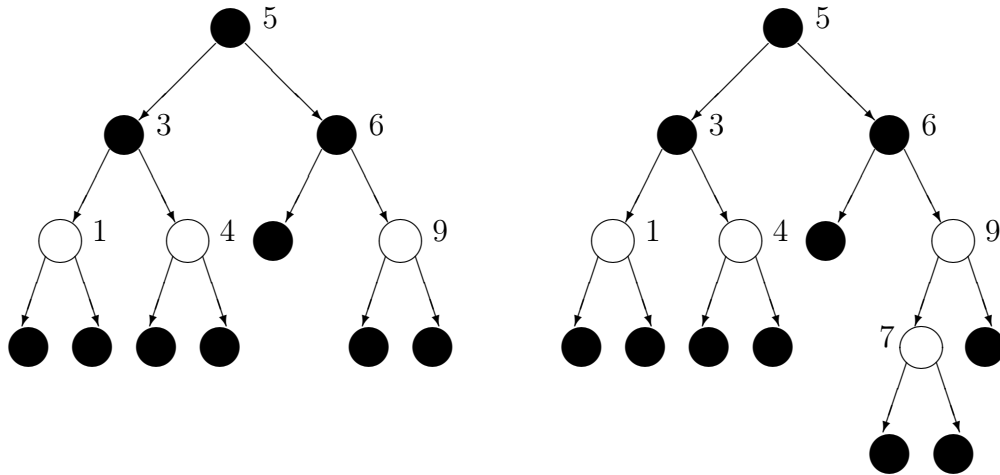
В результате корень становится красным, это случай 1, поэтому перекрашиваем узел 5 в черный цвет.



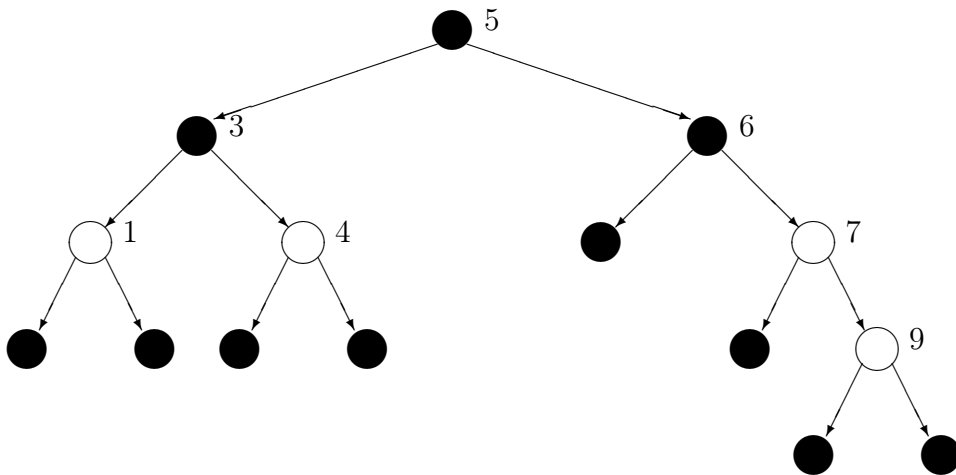
Добавляем узел 1 слева от 3. У узла 1 красный отец 3 и черный дядя (лист), при этом 1 – левый сын 3, и 3 – левый сын 4, это случай 5. Выполняем поворот дерева, узел 3 становится отцом для 1 и 4 и перекрашивается в черный цвет, а узлы 1 и 4 перекрашиваются в красный. Правое поддерево узла 3 становится левым поддеревом узла 4. Процесс изменения дерева показан на следующей странице.



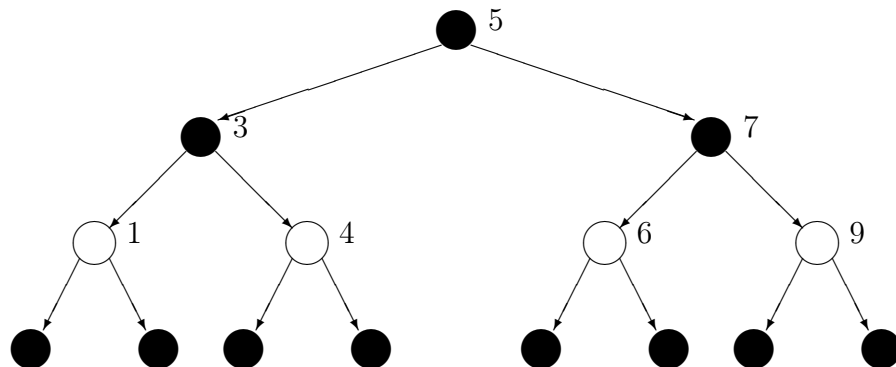
Добавляем узел 9 справа от 6. У узла 9 черный отец 6, это случай 2, поворотов и перекрашиваний не требуется. Затем добавляем узел 7 слева от узла 9. У узла 7 красный отец 9 и черный дядя (лист), причем узел 7 – левый сын узла 9, а узел 9 – правый сын узла 6. Этот случай симметричен случаю 4.



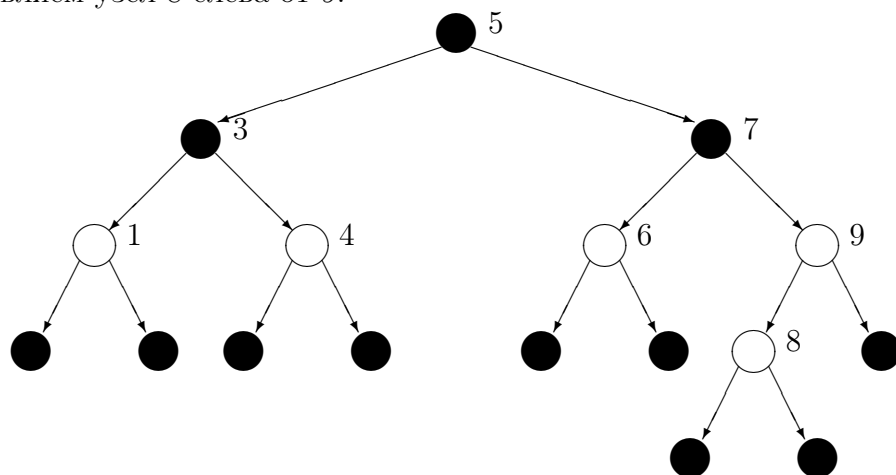
Выполняем поворот дерева, при котором узлы 7 и 9 меняются ролями: 7 становится отцом 9, а правое поддереву узла 7 – левым поддеревом узла 9.



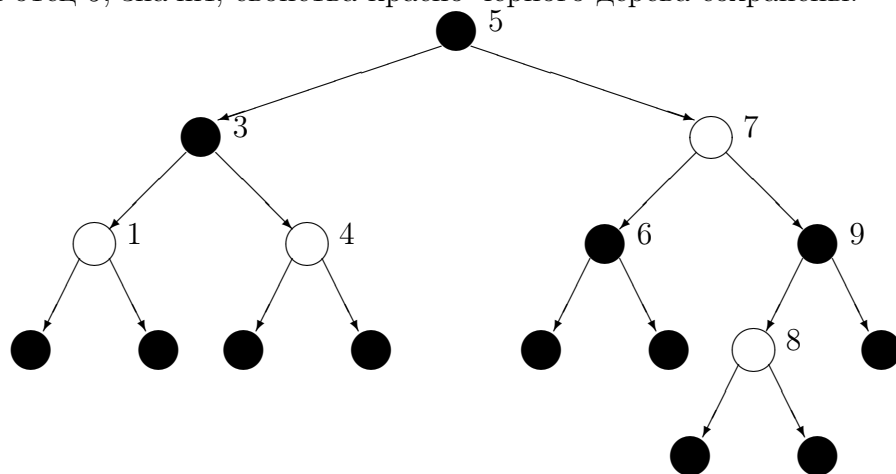
Теперь у красного узла 7 – красный отец 9 и черный дядя (лист), при этом узел 9 – правый потомок узла 7, а узел 7 – правый потомок узла 6. Этот случай симметричен случаю 5. Выполняем поворот дерева, при котором узел 7 становится отцом узлов 6 и 9, а левое поддерево узла 7 – правым поддеревом узла 6. Затем перекрашиваем узел 7 в черный цвет, а узлы 6 и 9 – в красный.



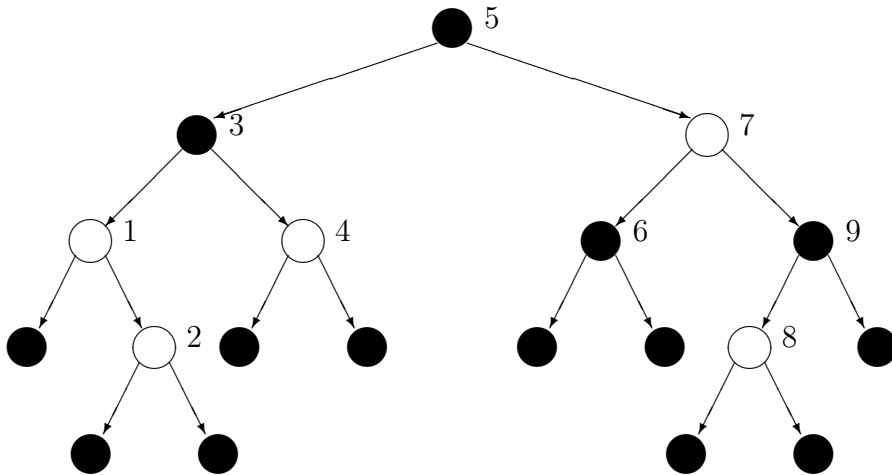
Добавляем узел 8 слева от 9.



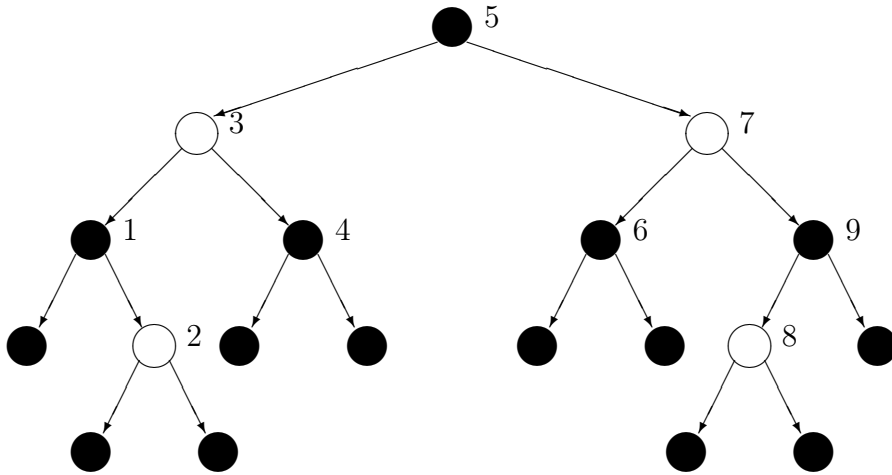
У красного узла 8 красный отец 9 и красный дядя 6, значит, имеет место случай 3. Перекрашиваем узлы 6 и 9 в черный цвет, а их отца 7 – в красный. У узла 7 красный отец 5, значит, свойства красно-черного дерева сохранены.



Добавляем узел 2 справа от 1.



У красного узла 2 красный отец 1 и красный дядя 4, значит, имеет место случай 3. Перекрашиваем узлы 1 и 4 в черный цвет, а их отца 3 – в красный. У узла 3 красный отец 5, значит, свойства красно-черного дерева сохранены.



Красно-черное дерево построено.

Удаление. При удалении узла с двумя нелистовыми сыновьями в обычном двоичном дереве поиска мы ищем либо наибольший (т.е. самый правый) элемент в его левом поддереве, либо наименьший (т.е. самый левый) элемент в его правом поддереве и перемещаем его значение в удаляемый узел. Затем мы удаляем узел, из которого копировали значение. Удаление узла в красно-черном дереве предполагает восстановление свойств красно-черного дерева, если в ходе удаления произошло их нарушение.

Будем использовать обозначение a для узла, ключ которого надо удалить из дерева; через b обозначим сына a . Если a имеет нелистового сына, возьмем этого сына за b . В противном случае за b возьмем любой из листовых сыновей. Узел, которым мы замещаем удаляемый, обозначим через d . В ходе выполнения удаления узел d , в свою очередь, извлекается из дерева и заменяется своим сыном, которым будем обозначать через x . Далее будем предполагать, что d – наименьший элемент правого поддерева, симметричные случаи рассматриваются аналогично.

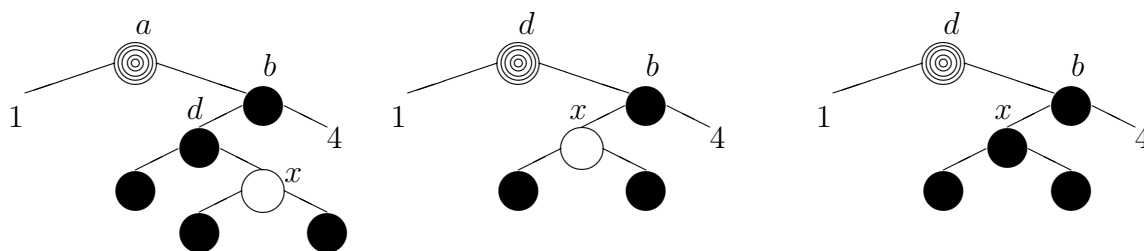
Случай 1. Узел d – красный, тогда для сохранения свойств красно-черного дерева он имеет двух черных сыновей. Поскольку этот узел является самым левым нелистовым элементом правого поддерева узла a , оба его потомка являются черными листьями, иначе нарушится либо Свойство 4 (оба сына каждого красного узла – черные), либо Свойство 5 (всякий простой путь от данного узла до любого листа, являющегося его потомком, содержит одинаковое число черных узлов). При удалении узла d он заменяется на черный лист, при этом сохраняются свойства дерева: свойство 2 (корень – черный), поскольку d не является корнем, и свойство 5 (всякий простой путь от данного узла до любого листа, являющегося его потомком, содержит одинаковое число черных узлов). Ниже продемонстрирована замена узла a на d и удаление d . Заметим, что d не обязательно является сыном b , это самый левый нелистовой узел правого поддерева a . Цифрами обозначены поддерева.



Аналогично рассматривается симметричный случай, когда d – самый правый нелистовой узел левого поддерева a .

Если извлекаемый из дерева узел d – черный, тогда его удаление может нарушить свойства красно-черного дерева.

Случай 2. Узел d – черный, его нелистовой сын x – красный (аналогично рассматривается случай, когда у узла a один сын – лист, а второй – красный узел, тогда в качестве d выступает a). Узел x имеет двух черных листовых сыновей. Узел d заменяется узлом x с двумя черными листовыми сыновьями, при этом может нарушиться Свойство 4 (оба сына каждого красного узла – черные). Кроме того, в любом случае извлечение из дерева черного узла d приводит к тому, что все пути, проходившие через d , будут содержать на один черный узел меньше, и для всех предков d нарушается Свойство 5 (всякий простой путь от данного узла до любого листа, являющегося его потомком, содержит одинаковое число черных узлов). Узел x перекрашивается в черный цвет, тогда Свойства 4 и 5 восстанавливаются.



Аналогично рассматривается симметричный случай, когда d – самый правый нелистовой узел левого поддерева a .

Случай 3. Узел d – черный, оба его сына – черные листья (аналогично рассматривается случай, когда у узла a оба сына являются листьями, тогда в качестве d выступает a). Обозначим лист, которым будем заменять узел d , через x . Извлече-

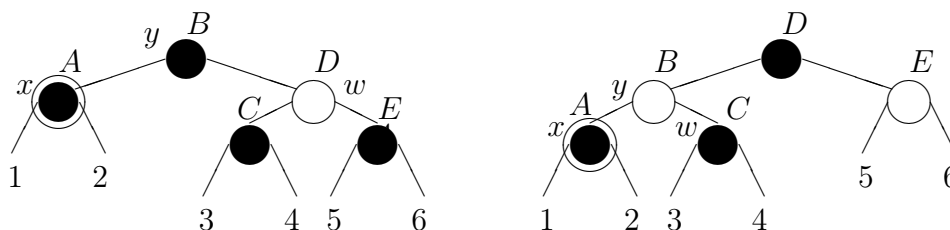
ние из дерева черного узла d приводит к тому, что все пути, проходившие через d , будут содержать на один черный узел меньше, и для всех предков d нарушается Свойство 5 (всякий простой путь от данного узла до любого листа, являющегося его потомком, содержит одинаковое число черных узлов). В этом случае узлу x приписывается «дополнительная чернота», т.е., если путь проходит через узел x , то число черных узлов дополнительно увеличивается на 1. Тогда начинает выполняться Свойство 5, но перестает выполняться Свойство 1 (узел либо красный, либо черный), вместо этого узел x становится «дважды черным» (при прохождении через него число черных узлов увеличивается на 2). На рисунке эта «двойная чернота» показана дополнительным контуром вокруг узла x .



Затем дополнительная чернота перемещается вверх по дереву, т.е. в качестве x рассматривается его отец. В ходе этого перемещения x может стать «красно-черным» (если ранее был красным), либо остаться «дважды черным» (если ранее был черным). Если узел x – «красно-черный», то он перекрашивается в черный цвет. Если узел x – «дважды черным», то можно выполнить повороты и перекрашивание дерева, в ходе которых двойная чернота будет устранена. В частности, если x становится корнем, то он просто окрашивается в черный цвет.

Рассмотрим подробно эти процедуры. Здесь x – узел с дополнительной чернотой, w – брат узла x . Далее показаны четыре возможных варианта, в которых x – левый, а w – правый сын, y – их отец (как и ранее, для каждого из них существует аналогичный симметричный случай). Большими буквами обозначены узлы, цифрами – поддеревья.

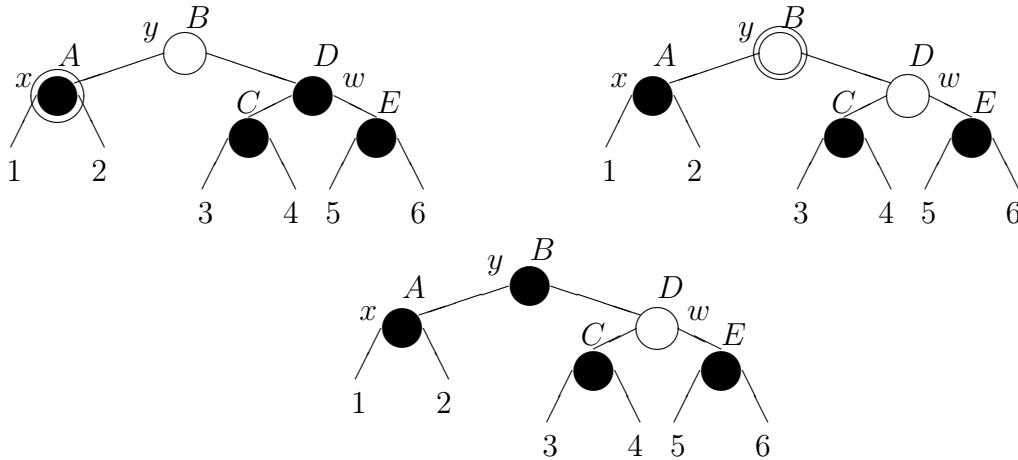
Случай 3a. Узел w – красный. В этом случае выполняется поворот дерева, при котором узел $w(D)$ становится отцом узла $y(B)$. При этом левый сын узла w (узел C) становится правым сыном узла $y(B)$ и братом узла $x(A)$, а значит, новым w . Узел B окрашивается в красный цвет, а D – в черный.



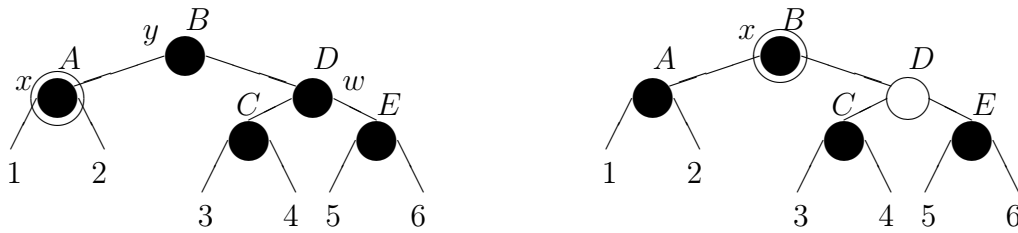
Заметим, что порядок узлов и поддеревьев не меняется, дерево так и остается деревом поиска.

Случай 3b. Узел w – черный, оба его сына – черные. Отец узлов w и x , узел y , может иметь любой цвет. В этом случае с узла x убирается дополнительная чернота, которая передается узлу y , а узел w перекрашивается в красный цвет. Это позволяет сохранить Свойство 5 (всякий простой путь от данного узла до любого листа, являющегося его потомком, содержит одинаковое число черных узлов), по-

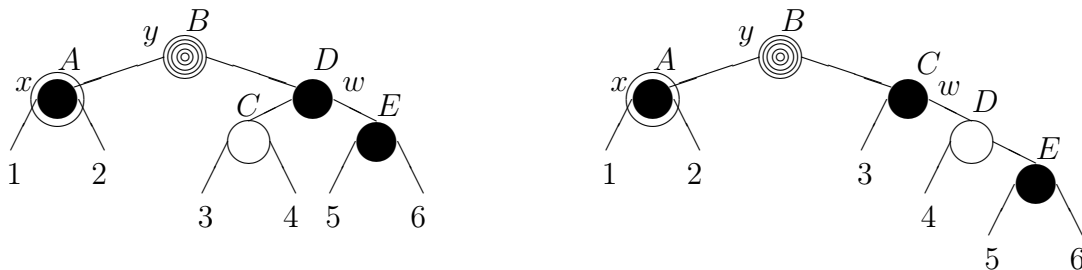
сколько все пути, которые ранее шли к поддеревьям 3–6 через черный узел $w(D)$, и к поддеревьям 1–2 через «дважды черный» узел $x(A)$, проходят через черный узел $y(B)$, которому теперь приписана дополнительная чернота. Если узел y был красным, то он становится «красно-черным», и тут же перекрашивается в черный.



Если узел y был черным, то он становится «дважды черным», и рассматривается в качестве нового x . Поворот дерева не производится.

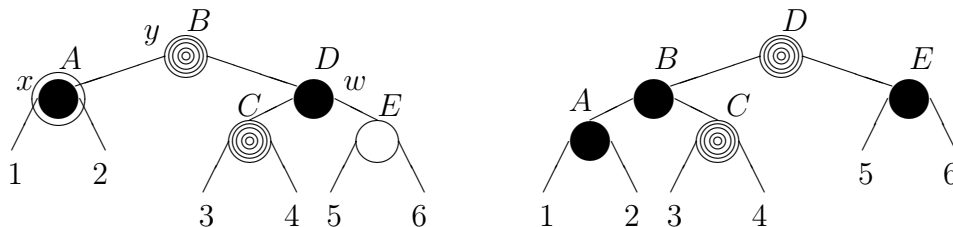


Случай 3с. Узел w – черный, его левый сын – красный, а правый – черный. Отец узлов w и x , узел y , может иметь любой цвет. В этом случае выполняется поворот дерева, при котором левый сын узла $w(D)$, узел C , переходит на место узла $w(D)$, узел $w(D)$ становится правым сыном узла C , а правое поддерево узла C , обозначенное номером 4, становится левым поддеревом узла D . Теперь узел C является братом узла $x(A)$, и рассматривается как новый w . Происходит также перекрашивание узлов: узел C окрашивается в черный цвет, а узел D – в красный. Это позволяет сохранить Свойство 5 (всякий простой путь от данного узла до любого листа, являющегося его потомком, содержит одинаковое число черных узлов), поскольку все пути, которые ранее шли к поддеревьям 3–6 через черный узел D , теперь идут через черный узел C .



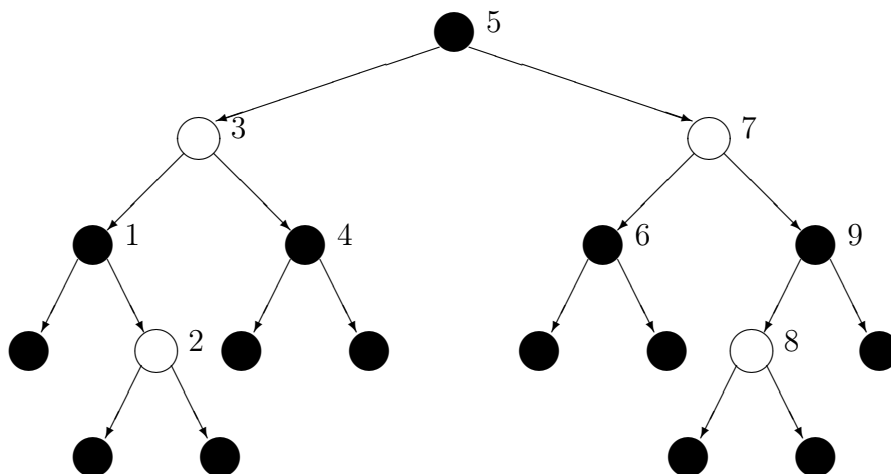
Заметим, что порядок узлов и поддеревьев не меняется, дерево остается деревом поиска.

Случай 3d. Узел w – черный, его правый сын – красный. Отец узлов w и x , узел y , может иметь любой цвет, так же, как и левый сын узла w . В этом случае выполняется поворот дерева, при котором узел $w(D)$ переходит на место узла $y(B)$, т.е. $y(B)$ становится левым сыном узла $w(D)$, а левое поддерево узла $w(D)$ с корнем в узле C – правым поддеревом узла $y(B)$. Происходит также переокрашивание узлов: узлы B и E , которые теперь являются сыновьями узла D , окрашиваются в черный цвет, и с узла x снимается двойная чернота, он становится просто черным узлом. Также узел D принимает первоначальный цвет узла B , а узел C сохраняет свой цвет. Это позволяет сохранить Свойство 5 (всякий простой путь от данного узла до любого листа, являющегося его потомком, содержит одинаковое число черных узлов), поскольку все пути, которые ранее шли через узел B , теперь идут через черный узел D . Также пути к поддеревьям 1 и 2, которые ранее проходили через двойной черный узел $x(A)$, теперь проходят через два черных узла: A и B . Пути, которые проходили к поддеревьям 3 и 4 через черный узел D и узел C , теперь проходят через черный узел B и узел C , который не менял окраски. Пути, которые проходили к поддеревьям 5 и 6 через черный узел D и красный узел E , теперь проходят через черный узел E .

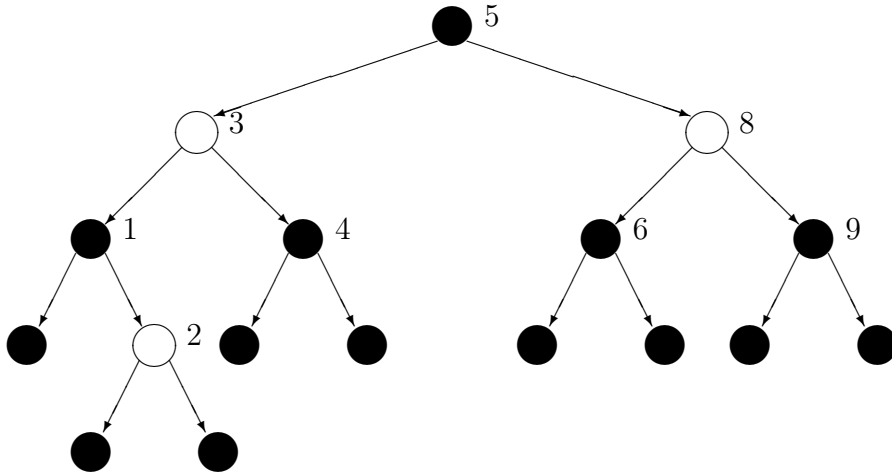


Заметим, что порядок узлов и поддеревьев не меняется, дерево так и остается деревом поиска. Также никакой узел не становится «двойным черным», а это означает, что свойства черно-красного дерева, нарушенные при удалении, восстановлены.

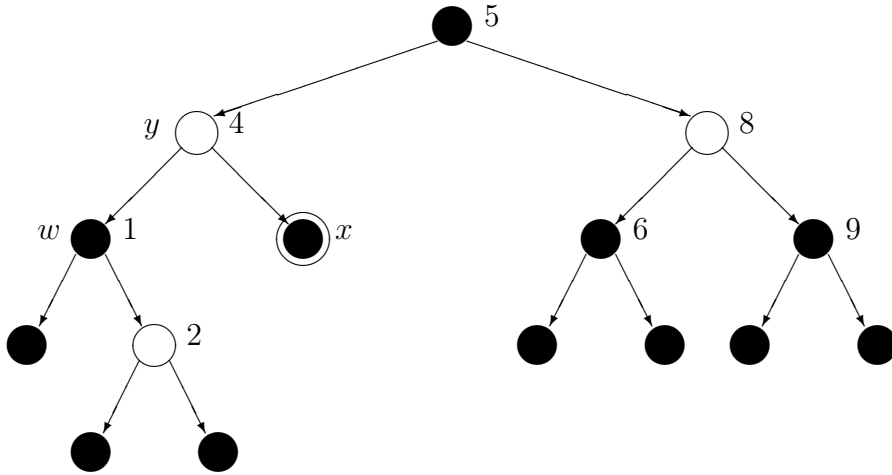
Пример. Рассмотрим красно-черное дерево из предыдущего примера.



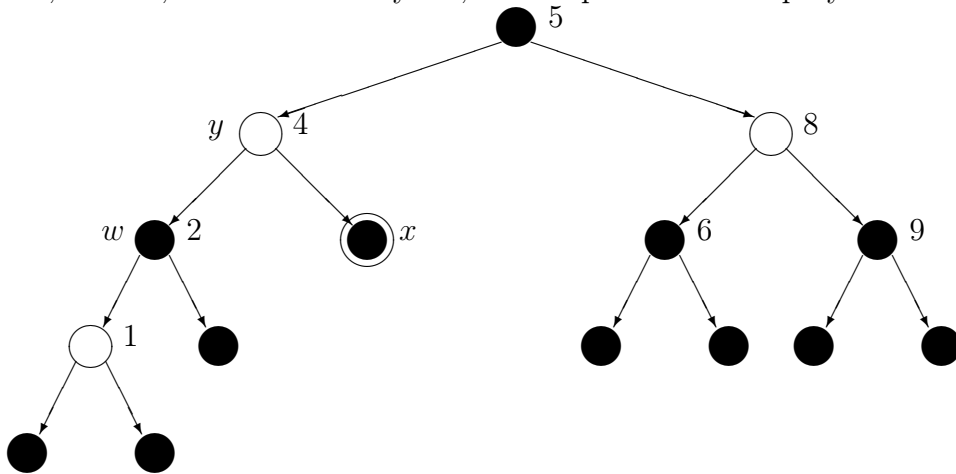
Удалим узел 7. Заменим его на самый левый нелистовой узел в правом поддереве, это узел 8, и извлечем узел 8 из дерева. Узел 8 – красный, значит, имеет место случай 1. Заменим узел 8 вместе с его листовыми сыновьями на лист без данных. Нарушения свойств красно-черного дерева не происходит.



Удалим узел 3, заменим его самым левым нелистовым элементом правого поддерева – узлом 4. Узел 4 извлечем из дерева. У него имеются два черных листовых сына, это случай 3. Заменяем узел 4 листом, приписываем ему двойную черную окраску и обозначаем через x . Братом узла x является узел 1, обозначаем его через w . Отцом x и w является узел 4, обозначаем его через y .

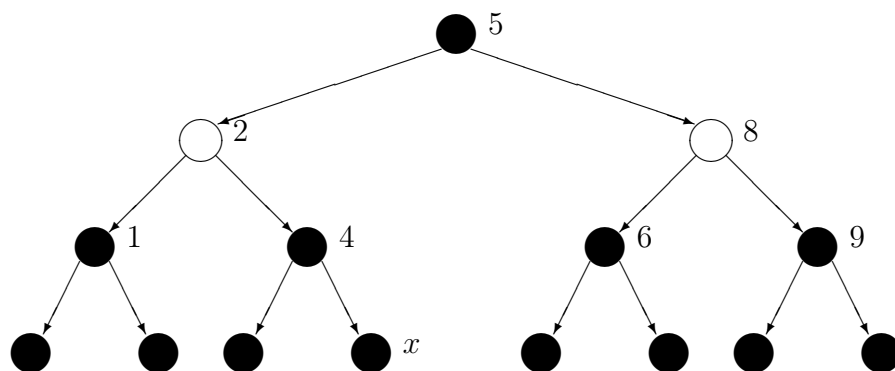


Узел w является левым сыном узла y , правый сын узла w – красный, левый – черный, значит, имеет место случай, симметричный 3с. В результате имеем



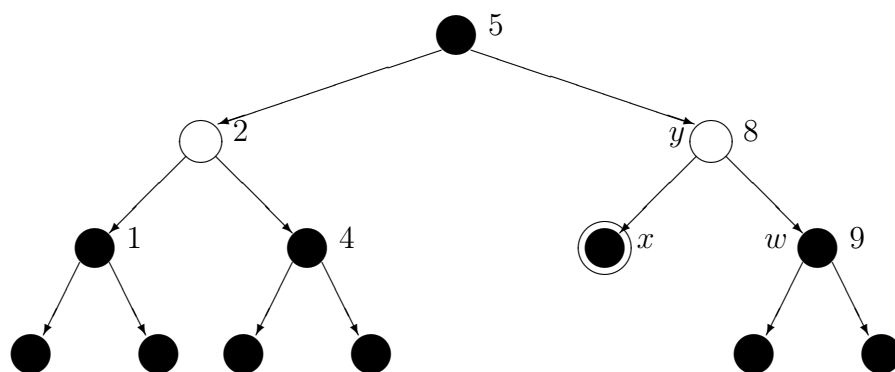
Здесь потребовался поворот дерева. Правый сын узла w , узел 2, переносится на место w , а узел $w(1)$ становится левым сыном узла 2, при этом левое поддерево узла 2 становится правым поддеревом узла 1. Также узлы перекрашиваются: узел 1 – в красный цвет, узел 2 – в черный. Узел 2 становится братом узла x , а значит, новым узлом w .

Узел w по прежнему является левым сыном узла $y(4)$, но теперь левый сын узла $w(2)$, узел 1 – красный, значит, имеет место случай, симметричный случаю 3d. Выполняем поворот дерева: узел $w(2)$ переносится на место узла $y(4)$, узел $y(4)$ становится правым сыном узла $w(2)$, а правое поддерево узла $w(2)$ – левым поддеревом узла $y(4)$. Узел $w(2)$ окрашивается в цвет узла $y(4)$, т.е. становится красным, а узел $y(4)$ окрашивается в черный цвет. Также в черный цвет перекрашивается красный сын узла $w(2)$ – узел 1.

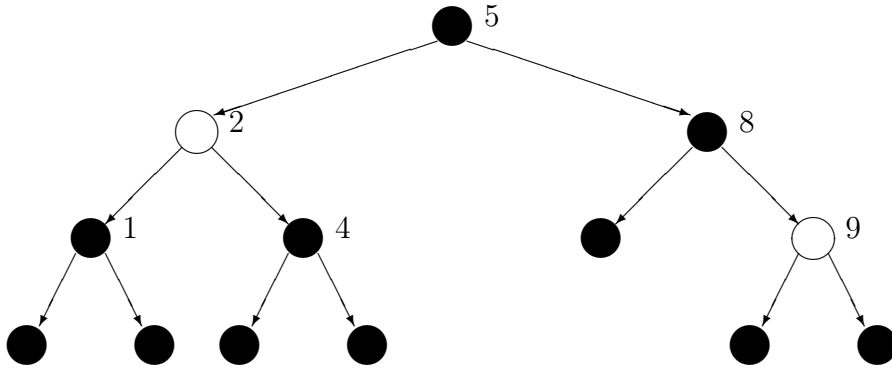


Свойства красно-черного дерева восстановлены.

Далее удалим узел 6. Поскольку оба его сына являются листьями, имеет место случай 3. Заменяем узел 6 листом без данных, которому приписываем дополнительную черноту. Обозначим этот лист через x , его брата 9 – через w , их отца 8 – через y .

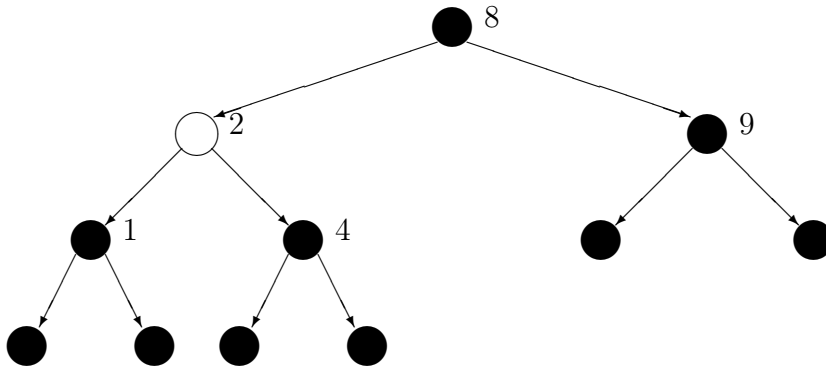


Узел $w(9)$ – черный, оба его сына – черные, значит, имеет место случай 3b. Узел $w(9)$ перекрашивается в красный цвет, а дополнительная чернота с узла x переносится на его отца $y(8)$, который в результате становится «черно-красным» и сразу перекрашивается в черный цвет. Результат показан на следующей странице.



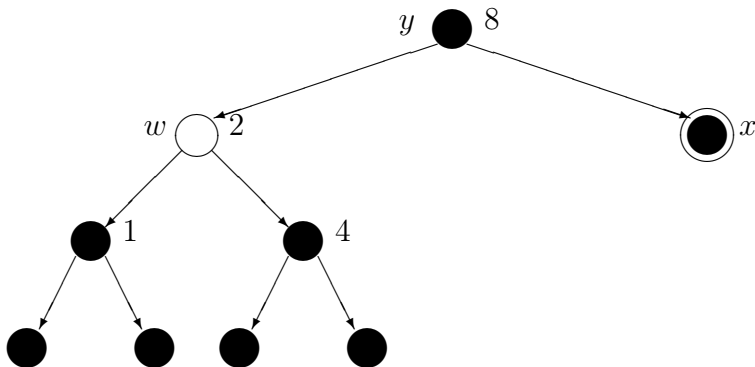
Свойства красно-черного дерева восстановлены.

Удалим узел 5, заменив его на самый левый нелистовой узел правого поддерева – узел 8. Извлечем 8 из дерева, он является черным, и имеет правого нелистового сына – красный узел 9, значит, имеет место случай 2. Заменяем узел 8 на узел 2. Заменяем узел 8 на узел 2 с двумя листовыми сыновьями и перекрашиваем узел 9 в черный цвет.



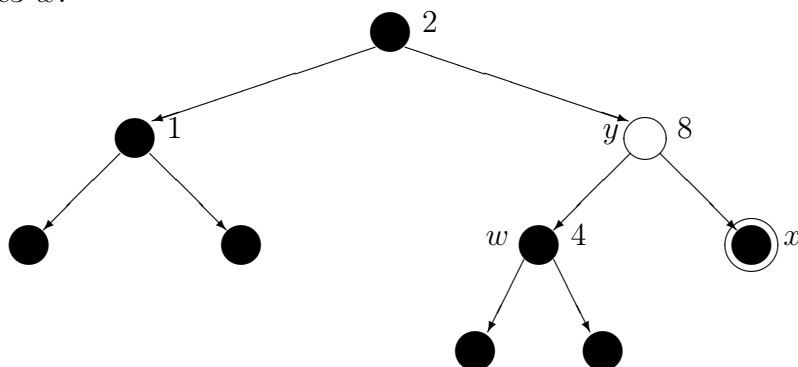
Свойства красно-черного дерева восстановлены.

Удалим из дерева узел 9. Он имеет двух листовых сыновей и является черным, значит, имеет место случай 3. Заменяем узел 9 листом, обозначаем его через x и приписываем ему дополнительную черную окраску. Брат узла x – это узел 2, обозначаем его через w , а их отца 8 – через y .

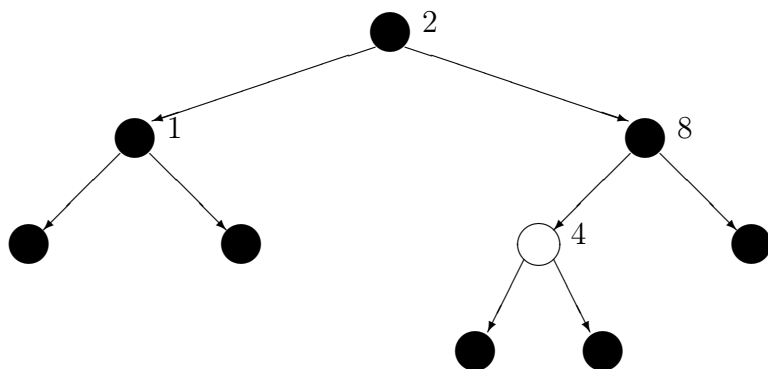


Узел $w(2)$ является левым сыном $y(8)$ и окрашен в красный цвет, значит, имеет место случай, симметричный случаю 3а. Выполняем поворот дерева, узел $w(2)$ переносится на место узла $y(8)$, $y(8)$ становится правым сыном узла $w(2)$, а правое

поддереву узла $w(2)$ – левым поддеревом узла $y(8)$. Узел $y(8)$ перекрашивается в красный цвет, а $w(2)$ – в черный. Братом узла x теперь является узел 4, обозначаем его через w .



Теперь узел w и оба его сына – черные, значит, имеет место случай 3b. Узел $w(4)$ перекрашивается в красный цвет. С узла x убирается дополнительная чернота, которая приписывается его отцу $y(8)$, который был красного цвета, и в результате становится черно-красным. Узел $y(8)$ перекрашивается в красный цвет.



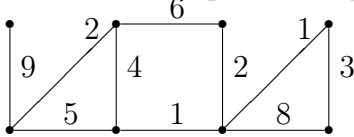
Свойства красно-черного дерева восстановлены.

Вычислительная сложность алгоритмов. Поиск в красно-черном дереве не отличается от поиска в обычном бинарном дереве поиска. Результат вставки или удаления может привести к нарушению свойств красно-черных деревьев. Восстановление свойств требует конечного числа операций смены цветов или поворотов дерева, которые в худшем случае выполняются для всех узлов одной ветви дерева. Поэтому вычислительная сложность алгоритмов остается пропорциональной высоте дерева, т.е. $O(\log_2 p)$.

5.7 Упражнения

1. Нарисовать диаграммы всех неизоморфных деревьев с шестью вершинами.
2. Нарисовать диаграммы всех неизоморфных ордеревьев с пятью вершинами.
3. Нарисовать диаграммы всех неизоморфных упорядоченных деревьев с четырьмя листьями.
4. Нарисовать диаграммы всех неизоморфных бинарных деревьев с четырьмя листьями.

5. Найти алгоритмом Краскала кратчайший остов в графе.



6. Граф задан матрицей весов ребер. Найти алгоритмом Эдмондса максимальный остовный лес.

	<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	<i>e</i>	<i>f</i>	<i>g</i>	<i>h</i>
<i>a</i>			4		1	7		6
<i>b</i>	2			2				
<i>c</i>	5				9			2
<i>d</i>			5				6	7
<i>e</i>		3	1					
<i>f</i>			8					
<i>g</i>	4				7			1
<i>h</i>		8				5		

7. Последовательно построить дерево сортировки из букв своей фамилии, имени и отчества (всего 10 различных букв). Удалить букву, добавленную четвертой. Удалить корень левого поддерева.

8. Последовательно построить сбалансированное дерево из букв своей фамилии, имени и отчества (всего 10 различных букв). Удалить букву, добавленную четвертой. Удалить корень левого поддерева.

9. Последовательно построить черно-красное дерево из букв своей фамилии, имени и отчества (всего 10 различных букв). Удалить букву, добавленную четвертой. Удалить корень левого поддерева.

Приложение. Поиск покрытий булевой матрицы

Определение. Будем говорить, что строка булевой матрицы *покрывает* столбец, если она содержит единицу в этом столбце.

Пример. Рассмотрим матрицу *Q*, и далее будем использовать это обозначение в последующих примерах.

1	1									<i>A</i>
		1	1							<i>B</i>
				1	1					<i>C</i>
							1	1		<i>D</i>
						1			1	<i>E</i>
				1		1				<i>F</i>
								1	1	<i>G</i>
	1	1			1		1			<i>H</i>
1	2	3	4	5	6	7	8	9	10	

Строка *H* покрывает второй, третий, шестой и восьмой столбцы матрицы *Q*.

Определение. *Покрытием* матрицы назовем подмножество таких ее строк, которые в совокупности покрывают все столбцы таблицы.

Примеры. Покрытиями матрицы Q являются: все восемь ее строк; семь строк: A, B, C, D, F, G, H , первые шесть строк; первые пять строк; однако не удается найти ни одного четырехстрочного покрытия.

Определение. *Длиной покрытия* назовем количество строк, образующих покрытие.

Примеры. Длины покрытий из предыдущего примера равны соответственно 8, 7, 6 и 5.

Определение. Покрытие матрицы называется *безызбыточным*, если при удалении из него хотя бы одной строки оно перестает быть покрытием.

Примеры. Множество из первых шести строк матрицы Q не является безызбыточным покрытием, поскольку при удалении шестой строки оно остается покрытием. Первые пять строк образуют безызбыточное покрытие, и оно не единственное – шесть строк A, B, C, D, F, G тоже образуют безызбыточное покрытие.

Определение. *Кратчайшим* покрытием матрицы назовем покрытие минимальной длины.

Примеры. Первые пять строк матрицы Q , а также пять строк A, B, F, G, H образуют ее кратчайшие покрытия.

Пусть строкам таблицы приписаны некоторые числа, называемые *рангами строк*.

Определение. *Минимальным* покрытием матрицы назовем покрытие с минимальной суммой рангов строк.

П1. Поиск всех безызбыточных покрытий

Очевидно, что все кратчайшие и минимальные покрытия матрицы безызбыточны. Поэтому поиск всех таких покрытий можно свести к построению всех безызбыточных покрытий и выделению из них кратчайших и минимальных. Тривиальный метод поиска всех безызбыточных покрытий состоит в переборе и анализе всех подмножеств строк матрицы. Однако, число перебираемых подмножеств велико (2^m , где m – число строк матрицы), поэтому остановимся на другом, более эффективном методе.

Будем предполагать, что матрица имеет покрытие, то есть не содержит столбцов, целиком состоящих из нулей, и прежде всего попытаемся упростить матрицу.

Определение. Столбец булевой матрицы назовем *ядерным*, если он содержит ровно одну единицу. Строку булевой матрицы назовем *ядерной*, если она покрывает ядерный столбец.

Пример. В матрице Q ядерными являются столбцы 1, 4 и строки A, B .

Очевидно, что ядерные строки входят в любое покрытие, что дает возможность сформулировать следующее правило упрощения матрицы.

Правило ядерной строки. Если в матрице есть ядерная строка, то она включается в любое искомое покрытие и удаляется из матрицы вместе со всеми столбцами, которые ядерная строка покрывает. Удаляются также пустые строки (состоящие из нулей), которые при этом могут появиться.

Пример. Применим правило ядерной строки дважды к знакомой нам матрице Q . В результате строки A и B будут включены в покрытие, а матрица упростится.

$Q' =$	1	1					C	
				1	1		D	
			1			1		E
	1		1					F
					1	1		G
		1		1				H
	5	6	7	8	9	10		

В частности, если все столбцы матрицы покрываются ядерными строками, то эти строки образуют единственное ее безызбыточное покрытие.

Упростив булеву матрицу по правилу ядерной строки, и оставив в ней по одному из одинаковых столбцов (что, очевидно, не повлияет на решение), продолжим поиск всех безызбыточных покрытий упрощенной матрицы.

Припишем ее строкам булевы переменные s_1, \dots, s_m , обозначим через S произвольное подмножество строк и положим $s_i = 1$, если и только если i -я строка принадлежит множеству S , то есть будем задавать подмножество S строк матрицы набором σ значений переменных s_1, \dots, s_m .

Определение. Булеву функцию $p(s_1, \dots, s_m)$, принимающую значение единицы на тех и только тех наборах σ , которые задают покрытия матрицы, назовем *функцией покрытия матрицы*.

Пример. Зададим формулой функцию покрытия предыдущей матрицы Q' , исходя из следующих рассуждений.

Первый столбец покрывается подмножеством строк S , если и только если в S входят:

- либо строка C (переменная $C = 1$),
- либо строка F (переменная $F = 1$),
- либо обе строки одновременно ($C = 1$ и $F = 1$),

то есть если и только если $C \vee F = 1$.

Аналогично второй столбец покрывается подмножеством S , если и только если $C \vee H = 1$, и так далее для всех столбцов.

Все столбцы покрываются подмножеством строк S , то есть $p(\sigma) = 1$, если и только если одновременно выполняются все перечисленные условия, то есть $(C \vee F)(C \vee H) \dots = 1$. Это означает, что функцию покрытия матрицы Q' можно задать формулой:

$$p(C, \dots, H) = (C \vee F)(C \vee H)(E \vee F)(D \vee H)(D \vee G)(E \vee G).$$

В общем случае для матрицы с k столбцами функция покрытия задается конъюнкцией вида $D_1 \cdot \dots \cdot D_k$, где D_j – дизъюнкция всех переменных, приписанных строкам с единицей в j -м столбце. Будем называть полученную формулу *конъюнктивной нормальной формой (КНФ)* функции покрытия:

$$p(s_1, \dots, s_m) = D_1 \cdot \dots \cdot D_k = \text{КНФ}_p.$$

Так как каждое слагаемое дизъюнкции D_j задает безызбыточное покрытие j -го столбца, то, перемножив дизъюнкции D_i и D_j , мы получим все безызбыточные покрытия столбцов i и j (но не только безызбыточные).

Пример. Перемножим первые две дизъюнкции предыдущего примера:

$$(C \vee F)(C \vee H) = C \vee CH \vee CF \vee FH.$$

Получены не только все безызбыточные покрытия первых двух столбцов: $\{C\}$ и $\{F, H\}$, но и покрытия $\{C, H\}$ и $\{C, F\}$, которые содержат строку C , поэтому не являются безызбыточными (это эквивалентно тому, что конъюнкции CH и CF поглощаются конъюнкцией C).

Обобщая приведенные рассуждения на все столбцы булевой матрицы, приходим к выводу, что для построения всех безызбыточных покрытий матрицы надо перемножать все дизъюнкции КНФ функции покрытия, выполняя при этом все возможные поглощения. В результате будет получена ДНФ, конъюнкции которой зададут все безызбыточные покрытия.

Алгоритм поиска всех безызбыточных покрытий булевой матрицы (основан на построении КНФ функции покрытия и преобразовании ее в ДНФ с выполнением поглощений).

Начало. Задана булева матрица.

Шаг 1. Применим к матрице правило ядерной строки многократно, пока это возможно. Если матрица стала пустой, то идем на конец. Иначе из равных столбцов оставим по одному, пронумеруем столбцы упрощенной матрицы и припишем строкам булевы переменные s_1, \dots, s_m .

Шаг 2. Выберем очередной (j -й) столбец матрицы. Если столбцы исчерпаны, идем на шаг 3. Иначе построим дизъюнцию D_j переменных, приписанных строкам, покрывающим j -й столбец, и идем на шаг 2.

Шаг 3. Из полученных дизъюнкций построим конъюнктивную нормальную форму $D_1 \cdot \dots \cdot D_k$.

Шаг 4. Раскроем скобки в КНФ, выполняя поглощения конъюнкций. Получим ДНФ $= K_1 \vee \dots \vee K_t$.

Конец. Конъюнкции K_1, \dots, K_t задают все безызбыточные покрытия упрощенной матрицы. Добавив к каждому из них ядерные строки (выделенные на первом шаге), получим все безызбыточные покрытия исходной матрицы.

Пример. Продемонстрируем алгоритм на знакомой нам булевой матрице Q . Для нее уже выполнен шаг 1: строки A и B выделены как ядерные и удалены вместе с первыми четырьмя столбцами. В результате получена упрощенная матрица и построена КНФ функции ее покрытия (шаг 2 повторен 6 раз и выполнен шаг 3):

1	1					C
			1	1		D
		1			1	E
1		1				F
				1	1	G
	1		1			H
5	6	7	8	9	10	

$$\text{КНФ} = (C \vee F)(C \vee H)(E \vee F)(D \vee H)(D \vee G)(E \vee G).$$

Шаг 4. Перемножив скобки в КНФ (удобнее перемножать первую на вторую, третью на шестую и четвертую на пятую скобки) и выполнив поглощения, получим ДНФ.

$$\begin{aligned}(C \vee FH)(E \vee FG)(D \vee GH) &= (CE \vee CFG \vee EFH \vee FGH)(D \vee GH) = \\ &= CDE \vee CEGH \vee CDFG \vee CFGH \vee \\ &\vee DEFH \vee EFGH \vee DFGH \vee FGH = \\ &= CDE \vee CEGH \vee CDFG \vee DEFH \vee FGH = \text{ДНФ}.\end{aligned}$$

Конец. Конъюнкции ДНФ задают все безызбыточные покрытия упрощенной матрицы:

$$\{C, D, E\}, \{C, E, G, H\}, \{C, D, F, G\}, \{D, E, F, H\}, \text{ и } \{F, G, H\}.$$

Добавив к каждому из них ядерные строки A и B , выделенные на первом шаге, получим все безызбыточные покрытия исходной матрицы (два покрытия длины 5 и три покрытия длины 6):

$$\begin{aligned}\{A, B, C, D, E\}, \{A, B, F, G, H\} \\ \{A, B, C, E, G, H\}, \{A, B, C, D, F, G\}, \{A, B, D, E, F, H\}.\end{aligned}$$

П2. Поиск минимальных и кратчайших покрытий с использованием КНФ функции покрытия

Зная все безызбыточные покрытия матрицы, выбрать из них кратчайшие или минимальные не составляет труда.

Пример. Из пяти найденных в предыдущем примере безызбыточных покрытий матрицы Q только два являются кратчайшими: это покрытия $\{A, B, C, D, E\}$ и $\{A, B, F, G, H\}$.

Однако поиск всех безызбыточных покрытий булевой матрицы становится излишне трудоемким, если требуется найти одно кратчайшее покрытие. В этом случае можно ограничиться поиском только некоторых безызбыточных покрытий, лишь бы среди них содержалось хотя бы одно кратчайшее. Тогда применимы следующие два правила упрощения матрицы.

Правило строки-предшественницы. Если в булевой матрице есть такие строки α и β , что $\alpha \preceq \beta$, то строка-предшественница α удаляется из матрицы (при этом кратчайшее покрытие не теряется, так как в любом покрытии строку α можно заменить строкой β , и оно останется покрытием).

Правило столбца-последователя. Если в булевой матрице есть такие столбцы γ и δ , что $\gamma \preceq \delta$, то столбец-последователь δ удаляется из матрицы (так как любое покрытие столбца γ одновременно покрывает столбец δ).

Заметим, что удаление столбцов-последователей может привести к появлению строк-предшественниц, и наоборот, поэтому эти два правила стоит выполнять «циклически» друг за другом.

Определение. Булеву матрицу, не содержащую ни ядерных строк, ни строк-предшественниц, ни столбцов-последователей, назовем *циклическим остатком*.

Примеры к правилам и определению циклического остатка будут приведены чуть позже.

Алгоритм поиска одного кратчайшего покрытия булевой матрицы (основан на построении циклического остатка и поиске всех его безызбыточных покрытий).

Начало. Задана булева матрица.

Шаг 1. Если в матрице есть однострочное покрытие, то включим его в искомое покрытие, идем на конец.

Шаг 2. Применим правило ядерной строки. Если ядерная строка обнаружилась, идем на шаг 1.

Шаг 3. Применим правило строки-предшественницы. Если такая строка обнаружилась, идем на шаг 2.

Шаг 4. Применим правило столбца-последователя. Если такой столбец обнаружился, идем на шаг 3.

Шаг 5. Построим КНФ функции покрытия циклического остатка и преобразуем КНФ в ДНФ, раскрывая скобки и выполняя поглощения.

Шаг 6. Конъюнкция минимального ранга полученной ДНФ задает кратчайшее покрытие циклического остатка. Добавим к этому покрытию ядерные строки, выделенные на шаге 2.

Конец. Получено кратчайшее покрытие исходной матрицы.

Примеры. Все та же матрица Q сократилась до циклического остатка сразу после удаления ядерных строк, что не дает возможности продемонстрировать шаги 3 и 4 алгоритма. Рассмотрим в связи с этим другую матрицу – Q'' .

$$Q'' = \begin{array}{|c|c|c|c|c|c|c|c|} \hline & 1 & & 1 & & 1 & & \\ \hline & 1 & 1 & & & & & \\ \hline 1 & & & 1 & & 1 & & \\ \hline 1 & & & & & & 1 & 1 \\ \hline 1 & & & 1 & & & 1 & \\ \hline & & & & 1 & 1 & & 1 \\ \hline & & & & 1 & & 1 & 1 \\ \hline \end{array} \begin{array}{l} A \\ B \\ C \\ D \\ E \\ F \\ G \end{array}$$

1 2 3 4 5 6 7 8

Шаги 1, 2. Матрица Q'' не имеет однострочного покрытия, но в ней есть ядерная строка B . Включаем ее в покрытие и удаляем из матрицы вместе со вторым и третьим столбцами, которые она покрывает. Получаем матрицу Q''_1 и идем на шаг 1.

$$Q''_1 = \begin{array}{|c|c|c|c|c|} \hline & 1 & & 1 & & \\ \hline 1 & 1 & & 1 & & \\ \hline 1 & & & & 1 & 1 \\ \hline 1 & 1 & & & 1 & \\ \hline & & 1 & 1 & & 1 \\ \hline & & 1 & & 1 & 1 \\ \hline \end{array} \begin{array}{l} A \\ C \\ D \\ E \\ F \\ G \end{array}$$

1 4 5 6 7 8

Шаги 1–3. В матрице Q''_1 нет однострочного покрытия и ядерной строки, но есть строка-предшественница: $A \preceq C$. Удаляем строку A , получаем матрицу Q''_2 . Идем на шаг 2.

Шаги 2–4. В матрице Q''_2 нет ядерной строки и строки-предшественницы, но

первый столбец следует за четвертым. Удаляем первый столбец, получаем матрицу Q_3'' . Идем на шаг 3.

$$Q_2'' = \begin{array}{|c|c|c|c|c|c|} \hline 1 & 1 & & 1 & & \\ \hline 1 & & & & 1 & 1 \\ \hline 1 & 1 & & & 1 & \\ \hline & & 1 & 1 & & 1 \\ \hline & & 1 & & 1 & 1 \\ \hline \end{array} \begin{array}{l} C \\ D \\ E \\ F \\ G \end{array} \quad Q_3'' = \begin{array}{|c|c|c|c|c|} \hline 1 & & 1 & & \\ \hline & & & 1 & 1 \\ \hline 1 & & & 1 & \\ \hline & 1 & 1 & & 1 \\ \hline & 1 & & 1 & 1 \\ \hline \end{array} \begin{array}{l} C \\ D \\ E \\ F \\ G \end{array}$$

1 4 5 6 7 8 4 5 6 7 8

Шаги 3–4. В матрице Q_3'' появилась строка-предшественница: $D \preceq G$. Удалив D , получаем матрицу Q_4'' и идем на шаг 2.

Шаг 3. В матрице Q_4'' нет ядерных строк и строк-предшественниц, но восьмой столбец совпадает с пятым (то есть следует за ним). Удалив восьмой столбец, получаем матрицу Q_5'' .

$$Q_4'' = \begin{array}{|c|c|c|c|c|} \hline 1 & & 1 & & \\ \hline 1 & & & 1 & \\ \hline & 1 & 1 & & 1 \\ \hline & 1 & & 1 & 1 \\ \hline \end{array} \begin{array}{l} C \\ E \\ F \\ G \end{array} \quad Q_5'' = \begin{array}{|c|c|c|c|} \hline 1 & & 1 & \\ \hline 1 & & & 1 \\ \hline & 1 & 1 & \\ \hline & 1 & & 1 \\ \hline \end{array} \begin{array}{l} C \\ E \\ F \\ G \end{array}$$

4 5 6 7 8 4 5 6 7

Шаги 2–5. В матрице Q_5'' нет ни ядерных строк, ни строк-предшественниц, ни столбцов-последователей, значит, получен циклический остаток. Строим КНФ функции его покрытия:

$$\text{КНФ} = (C \vee E) (F \vee G) (C \vee F) (E \vee G) =$$

[преобразуем КНФ в ДНФ: для удобства умножаем первую скобку на третью, вторую на четвертую и выполняем поглощения]

$$= (C \vee EF) (G \vee EF) = CG \vee EF = \text{ДНФ}.$$

Шаг 6. Обе конъюнкции имеют минимальный ранг, поэтому выбираем любую из них, CG , она задает кратчайшее покрытие циклического остатка. Добавив ядерную строку B , получаем кратчайшее покрытие исходной булевой матрицы: $\{B, C, G\}$.

Замечание. При поиске минимального покрытия можно пользоваться тем же алгоритмом, за исключением применения правила строки-предшественницы.

ПЗ. Поиск минимальных и кратчайших покрытий методом ветвей и границ

Метод ветвей и границ используется для задач оптимизации с дискретными значениями переменных. Идея метода состоит в следующем: вводится дополнительное ограничение на одну из переменных, и задача делится на две подзадачи. В одной из них переменная удовлетворяет условию, в другой – нет. Затем каждая из подзадач решается аналогично.

С каждой задачей, кроме того, связывается оценочная функция, которая оценивает снизу стоимость решения. Если оценочная функция превышает минимальную стоимость решения, то задача отбрасывается. В качестве минимальной стоимости решения можно сначала взять стоимость произвольно найденного решения, или даже $+\infty$. Если получено решение, стоимость которого меньше минимальной стоимости, то минимальная стоимость становится равной стоимости полученного решения.

В применении к поиску покрытий булевой матрицы задача поиска покрытий делится на следующие задачи: покрытия, включающие определенную строку, и покрытия, не включающие эту строку. Если покрытие включает строку, то она удаляется из матрицы вместе со столбцами, которые она покрывает. Если покрытие не включает строку, то строка удаляется из матрицы. Для быстреего получения решения, как правило, выгоднее выбирать строку наибольшего веса, т.е. с наибольшим числом единиц.

Кроме того, при использовании метода ветвей и границ применяются все правила упрощения матрицы, рассмотренные в предыдущих подразделах.

Алгоритм поиска кратчайшего покрытия булевой матрицы

Начало. Задана булева матрица M , имеющая покрытие. Полагаем $C^* = m$, где m – число строк матрицы. Включаем матрицу M с оценочной функцией $C(M) = 0$ в список задач.

Шаг 1. Если список задач пуст, то кратчайшее покрытие найдено, идем на конец. Иначе выбираем очередную матрицу M из списка задач.

Шаг 2. Если в матрице M нет однострочного покрытия, идем на шаг 3. Иначе включаем однострочное покрытие в искомое покрытие матрицы M . Полагаем $C(M) = C(M) + 1$. Если $C(M) < C^*$, полагаем $C^* = C(M)$ и запоминаем найденное покрытие матрицы M как кратчайшее покрытие исходной матрицы. Идем на шаг 1.

Шаг 3. Применим правило ядерной строки. Если ядерных строк не обнаружилось, идем на шаг 5.

Шаг 4. Полагаем $C(M) = C(M) + 1$. Если $C(M) \geq C^*$, отбрасываем задачу M и идем на шаг 1, иначе идем на шаг 2.

Шаг 5. Применим правило строки-предшественницы. Если такая строка обнаружилась, идем на шаг 3.

Шаг 6. Применим правило столбца-последователя. Если такой столбец обнаружился, идем на шаг 5.

Шаг 7. Выберем строку α наибольшего веса. Запишем в список задач две матрицы.

1) Матрица M' получается из матрицы M удалением строки α и всех покрытых этой строкой столбцов. Покрытие матрицы M' включает покрытие матрицы M и строку α . Оценочная функция $C(M') = C(M) + 1$. Если $C(M') < C^*$, добавляем матрицу M' в список задач.

2) Матрица M'' получается из матрицы M удалением строки α . Покрытие матрицы M'' включает покрытие матрицы M . Оценочная функция $C(M'') = C(M)$. Добавляем матрицу M'' в список задач. Идем на шаг 1.

Конец. Получено кратчайшее покрытие исходной матрицы.

Пример. Рассмотрим матрицу Q из предыдущих примеров.

1	1									<i>A</i>
		1	1							<i>B</i>
				1	1					<i>C</i>
							1	1		<i>D</i>
						1			1	<i>E</i>
				1		1				<i>F</i>
								1	1	<i>G</i>
	1	1			1		1			<i>H</i>
1	2	3	4	5	6	7	8	9	10	

Начало. Полагаем $C^* = 8$. Включаем матрицу Q в список задач, $C(Q) = 0$.

Шаг 1. Список задач не пуст, выбираем матрицу Q из списка задач.

Шаги 2–4. Матрица Q не имеет однострочного покрытия, но в ней есть ядерные строки A и B . Включаем их в покрытие и удаляем из матрицы вместе со столбцами, которые они покрывают. Покрытие матрицы $\pi(Q) = \{A, B\}$.

	1	1								<i>C</i>
				1	1					<i>D</i>
			1						1	<i>E</i>
$Q =$	1		1							<i>F</i>
						1	1			<i>G</i>
		1		1						<i>H</i>
	5	6	7	8	9	10				

Полагаем $C(Q) = C(Q) + 2 = 2$, т.к. $C(Q) \geq C^*$, идем на шаг 2.

Шаги 2–5. В матрице Q нет однострочного покрытия, ядерных строк, строк-предшественниц и столбцов-последователей.

Шаг 6. Выберем строку C . Запишем в список задач две матрицы.

1) Матрица N получается из матрицы M удалением строки C и покрытых этой строкой столбцов 5 и 6.

		1	1			<i>D</i>
	1				1	<i>E</i>
	1					<i>F</i>
			1	1		<i>G</i>
		1				<i>H</i>
	7	8	9	10		

Покрытие матрицы $\pi(N) = \{A, B, C\}$. Оценочная функция $C(N) = C(Q) + 1 = 3$. Поскольку $C(N) < C^*$, добавляем матрицу N в список задач.

2) Матрица P получается из матрицы M удалением строки C . Покрытие матрицы $\pi(P) = \{A, B\}$.

				1	1		<i>D</i>
			1				<i>E</i>
	1		1				<i>F</i>
					1	1	<i>G</i>
		1		1			<i>H</i>
	5	6	7	8	9	10	

Оценочная функция $C(P) = C(Q) = 2$. Добавляем матрицу P в список задач. Идем на шаг 1.

Шаг 1. Список задач не пуст, выбираем матрицу N из списка задач.

Шаги 2–5. Матрица N не имеет однострочного покрытия и ядерных строк, но имеет строки-предшественницы: $F \preceq E$, $H \preceq D$. Удаляем их из матрицы.

$$N = \begin{array}{|c|c|c|c|} \hline & 1 & 1 & \\ \hline 1 & & & 1 \\ \hline & & 1 & 1 \\ \hline \end{array} \begin{array}{l} D \\ E \\ G \end{array}$$

7 8 9 10

Идем на шаг 3.

Шаги 3–4. Матрица N имеет ядерную строку E . Включаем ее в покрытие и удаляем из матрицы вместе со столбцами, которая она покрывает.

$$N = \begin{array}{|c|c|} \hline 1 & 1 \\ \hline & 1 \\ \hline \end{array} \begin{array}{l} D \\ G \end{array}$$

8 9

Покрытие $\Pi(N) = \{A, B, C, E\}$. Полагаем $C(N) = C(N) + 1 = 4$. Идем на шаг 2.

Шаг 2. Матрица N имеет однострочное покрытие D . Включаем эту строку в покрытие, получаем $\Pi(N) = \{A, B, C, E, D\}$. Полагаем $C(N) = C(N) + 1 = 5$. Поскольку $C(N) < C^*$, полагаем $C^* = C(N)$ и запоминаем найденное покрытие как кратчайшее покрытие исходной матрицы. Идем на шаг 1.

Шаг 1. Список задач не пуст, выбираем матрицу P из списка задач.

$$P = \begin{array}{|c|c|c|c|c|c|} \hline & & & 1 & 1 & \\ \hline & & 1 & & & 1 \\ \hline 1 & & 1 & & & \\ \hline & & & & 1 & 1 \\ \hline & 1 & & 1 & & \\ \hline \end{array} \begin{array}{l} D \\ E \\ F \\ G \\ H \end{array}$$

5 6 7 8 9 10

Шаги 2–4. Матрица P не имеет однострочного покрытия, но в ней есть ядерные строки F и H . Включаем их в покрытие и удаляем из матрицы вместе со столбцами, которые они покрывают. Покрытие матрицы $\pi(P) = \{A, B, F, H\}$.

$$P = \begin{array}{|c|c|} \hline 1 & \\ \hline & 1 \\ \hline 1 & 1 \\ \hline \end{array} \begin{array}{l} D \\ E \\ G \end{array}$$

9 10

Оценочная функция $C(P) = C(P) + 2 = 4 < C^*$, идем на шаг 2.

Шаг 2. Матрица P имеет однострочное покрытие G . Включаем эту строку в покрытие, получаем $\Pi(P) = \{A, B, F, H, G\}$. Полагаем $C(P) = C(P) + 1 = 5 = C^*$. Идем на шаг 1

Шаг 1. Список задач пуст, идем на конец.

Конец. Кратчайшее покрытие матрицы $\Pi(Q) = \{A, B, C, E, D\}$.

Список литературы

- [1] К. Берж. Теория графов и ее применения. – М.:Изд. иностр. лит., 1962.
- [2] А.В. Белоусов, С.В. Ткачев. Дискретная математика. – М.: Изд-во МГТУ им. Баумана, 2002.
- [3] Т.Х. Кормен, Ч.И. Лейзерсон, Р.Л. Ривест, К.Штайн. Алгоритмы. Построение и анализ (3-е издание). – М.: Издательский дом «Вильямс», 2013.
- [4] Н.Кристофидес. Теория графов. Алгоритмический подход. – М.: Мир, 1977.
- [5] Э. Майника. Алгоритмы оптимизации на сетях и графах. – М.: Мир, 1981.
- [6] В.Н. Нефедов, В.А. Осипова. Курс дискретной математики. – М.: МАИ, 1992.
- [7] А.А.Новиков. Дискретная математика для программистов. – СПб.: Питер, 2001.
- [8] Р.Уилсон. Введение в теорию графов. – М.: Мир, 1977.
- [9] Ф.Харари. Теория графов. – М.: Мир, 1973.

Учебное издание

БУРКАТОВСКАЯ Юлия Борисовна
ТЕОРИЯ ГРАФОВ. ЧАСТЬ 1


Учебное пособие

Выпускающий редактор И.О. Фамилия
Редактор И.О. Фамилия
Компьютерная верстка И.О. Фамилия
Дизайн обложки И.О. Фамилия



Национальный исследовательский Томский политехнический университет
Система менеджмента качества
Издательства Томского политехнического университета сертифицирована
NATIONAL QUALITY ASSURANCE по стандарту BS EN ISO 9001:2008



ИЗДАТЕЛЬСТВО  ТПУ. 634050, г. Томск, пр. Ленина, 30
Тел./факс: 8(3822)56-35-35, www.tpu.ru