Graph theory: trees

Yulia Burkatovskaya

Department of Information Technologies

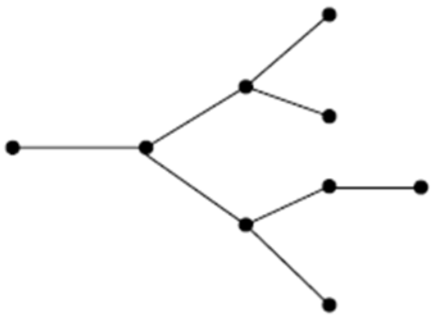
Associate professor

5. Trees

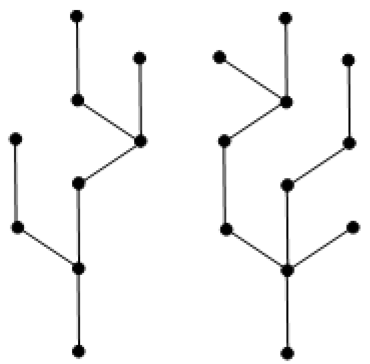
- Trees
- Minimum spanning tree
- Fundamental circuits and fundamental cuts
- Rooted trees
- Maximum branching
- Search trees

5.1. Trees

- A *connected acyclic graph* is called a **tree**. In other words, a connected graph with no cycles is called a tree.



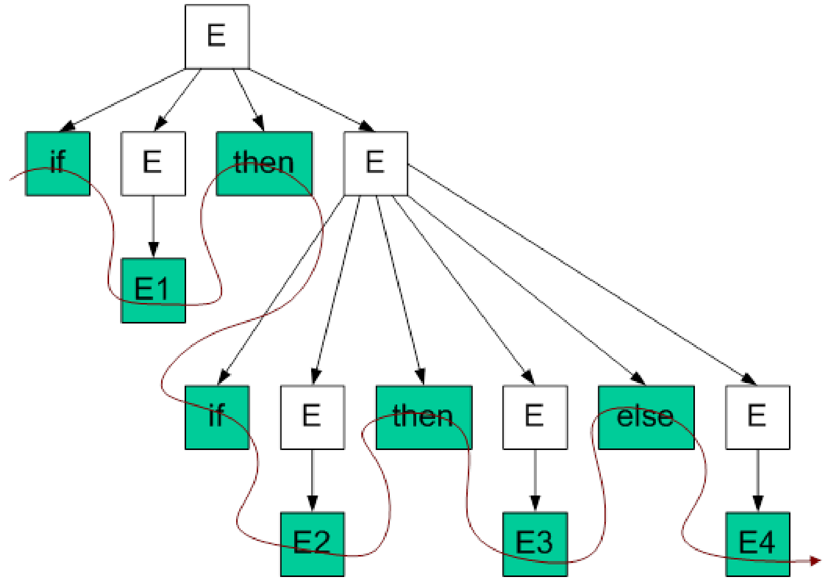
- An *acyclic graph* is called a **forest**.

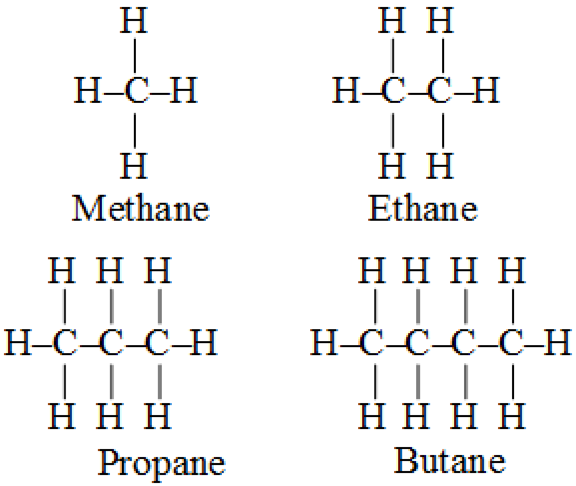


Application of trees

- **Chemistry** (saturated hydrocarbons)
- **Compilers** (parsing)

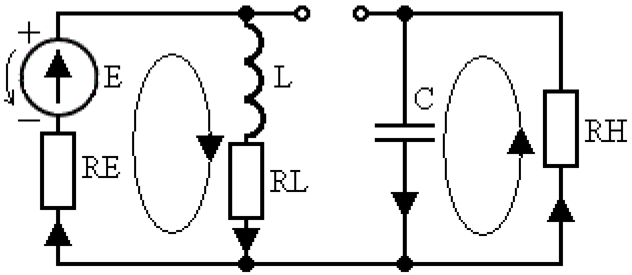
if E1 then if E2 then E3 else E4

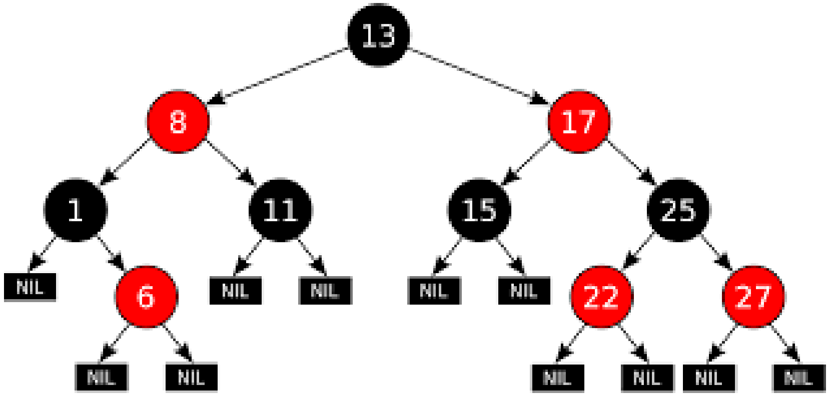


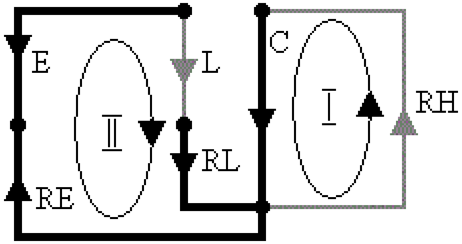


Application of trees

- **Physics** (electrical circuits)
- **Programming** (search trees)







Six different characterizations of a tree

- (1) T is a tree.
- (2) Any two vertices of T are connected by exactly one path.
- (3) T is connected, and every edge is a cut-edge.
- (4) T is connected and has $n - 1$ edges.
- (5) T contains no cycles and has $n - 1$ edges.
- (6) T contains no cycles, and for any new edge e, the graph $T+e$ has exactly one cycle.

Six different characterizations of a tree

(1)→(2)

- (1) *T* is a tree.
- (2) Any two vertices of *T* are connected by exactly one path.

- *T* is connected → any two vertices of *T* are connected.
- If there are two paths <*x*,*y*> then there is a cycle.

Six different characterizations of a tree

(2)→(3)

- (2) Any two vertices of T are connected by exactly one path.
- (3) T is connected, and every edge is a cut-edge.

- Any two vertices of T are connected → T is connected.
- If $e=(x,y)$ is not a cut-edge then $G\backslash(x,y)$ is connected; hence, there is a path $\langle x,y \rangle$ which does not include e. So, there are two different paths $\langle x,y \rangle$ in G.

Six different characterizations of a tree

(3)→(4)

- (3) T is connected, and every edge is a cut-edge.
- (4) T is connected and has $m=n-1$ edges.

Mathematical induction method.

- **Base:** for $n=1$ one has $m=0$.
- **Inductive step:** Let for graphs with $1,\ldots,n-1$ vertices the statement is true. Consider graph T with n vertices. Delete any edge e from T. As e is a cut-edge, $T\backslash e=T_1 \cup T_2$. They are connected, every edge is a cut-edge and the numbers of vertices is less than n; so, $m_1=n_1-1$ and $m_2=n_2-1$. The number of edges in T is the sum of the numbers of edges in T_1 and T_2 plus the deleted edge; consequently

$$m = m_1+m_2-1 = n_1-1+n_2-1+1 = n-1.$$

Six different characterizations of a tree

(4)→(5)

- (4) T is connected and has $n - 1$ edges.
- (5) T contains no cycles and has $n - 1$ edges.

- Let T contain a cycle with s vertices and edges; then, to join other n −s vertices to the cycle one needs at least n −s edges. So, the total number of edges is

$$m \geq s+n-s = n > n-1.$$

Six different characterizations of a tree

(5)→(6)

- (5) T contains no cycles and has $n - 1$ edges.
- (6) T contains no cycles, and for any new edge e, the graph $T+e$ has exactly one cycle.

- As T is acyclic than it is a forest with k trees. For a tree,

$$m=n - 1.$$

The total number of edges in T is

$$m_1+\ldots+m_k = n_1 - 1+\ldots+n_k - 1= n - k.$$

Hence; k=1 and T is a tree. Every two vertices of a tree are joined by the only path; so, adding a new edge produces exactly one cycle.

Six different characterizations of a tree

(6)→(1)

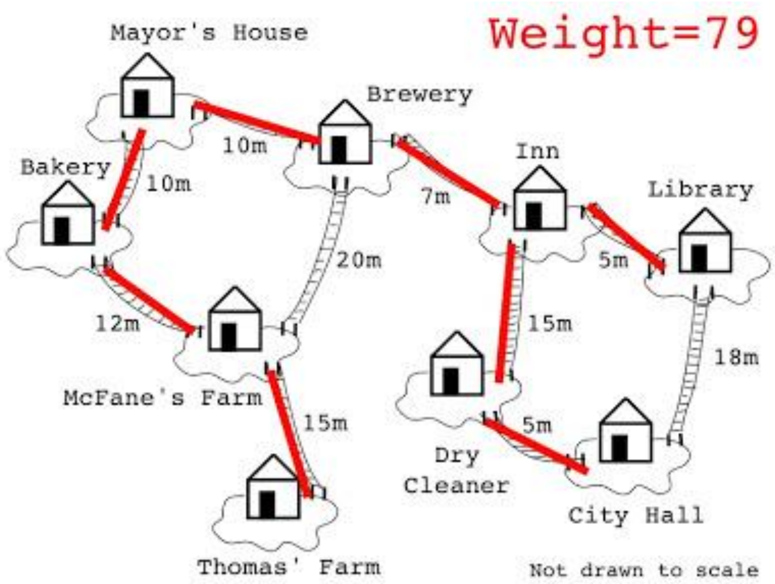
- (6) T contains no cycles, and for any new edge e, the graph $T+e$ has exactly one cycle.
- (1) T is a tree.

- If after adding any new edge a cycle appears then any two vertices are joined by a path; so, T is connected. This together with absence of cycles gives a tree.

5.2. Minimum spanning tree

- How to join all houses and to minimize the length of the communications?

- In a weighted graph, the **minimum spanning tree** is the set of edges with the minimum total weight such that they connect all of the nodes.
- **Applications of MST problem**: https://www.geeksforgeeks.org/?p=11110 .



http://computationaltales.blogspot.ru/2011/08/minimum-spanning-trees-prims-algorithm.html

Greedy algorithms

- A **greedy algorithm** is an algorithmic paradigm that follows the problem solving heuristic of making the locally optimal choice at each stage with the hope of finding a global optimum.
- In many problems, a greedy strategy does not in general produce an optimal solution.
- But for the **minimum spanning tree** problem, greedy algorithms produce a **global optimum**.

Kruskal's algorithm

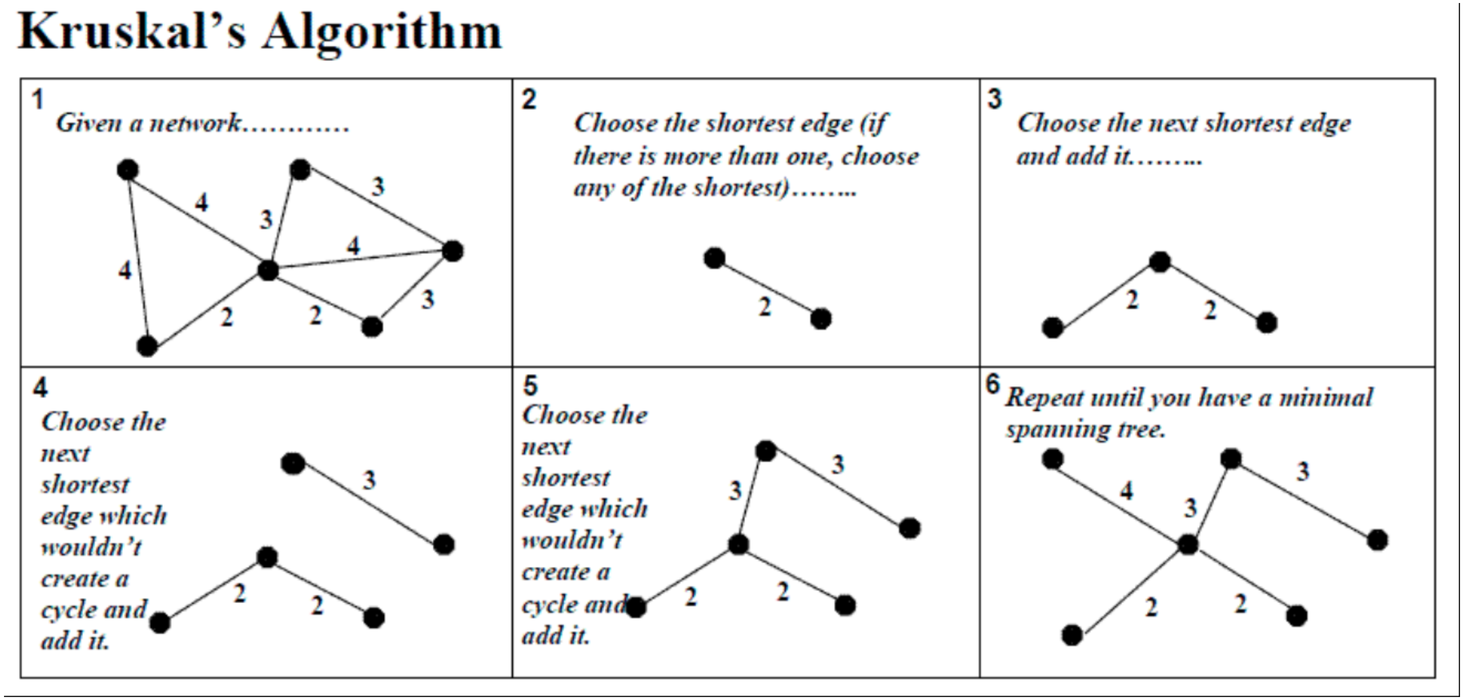
- Sort all the edges from low weight to high
- Take the edge with the lowest weight and add it to the spanning tree. If adding the edge created a cycle, then reject this edge.
- Keep adding edges until we have $p-1$ edges.

- https://www.programiz.com/dsa/kruskal-algorithm
- https://youtu.be/71UQH7Pr9kU

Kruskal’s algorithm

Example.

- http://nadide.github.io/assets/img/algo-image/MST/kruskal.png



Prim’s algorithm

- Initialize the minimum spanning tree with a vertex chosen at random.
- Find all the edges that connect the tree to new vertices, find the minimum and add it to the tree
- Keep adding edges until we have $p-1$ edges.

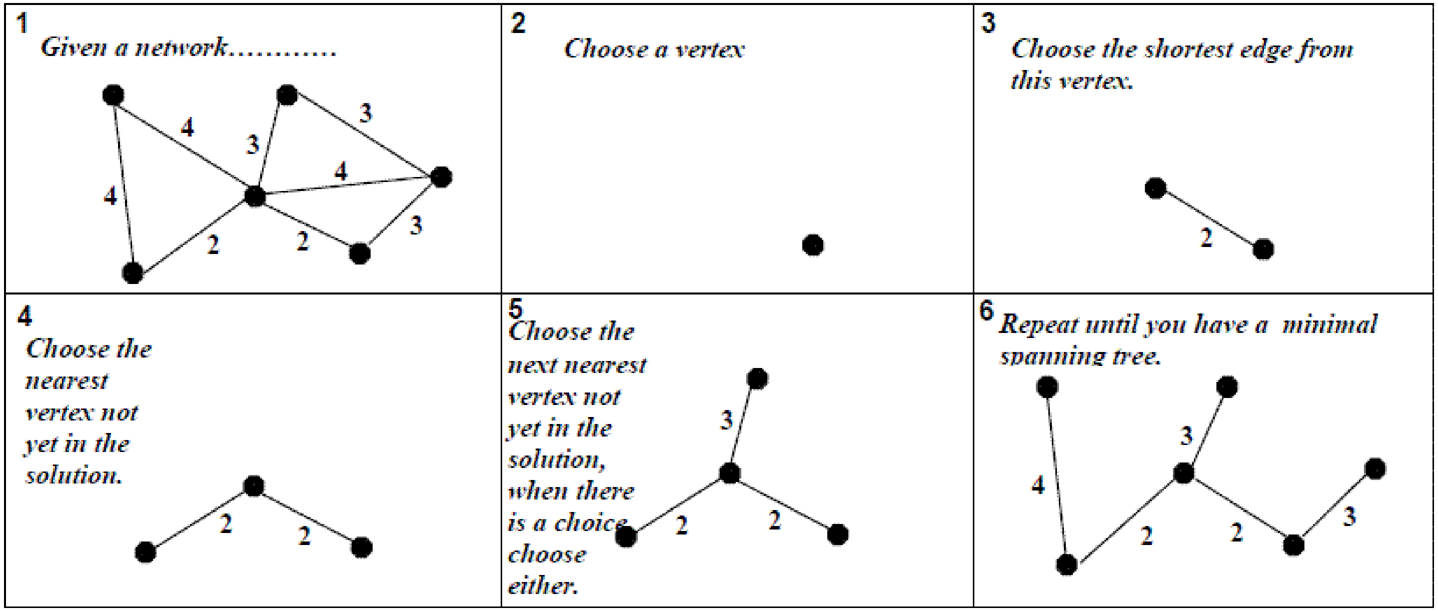
- https://www.programiz.com/dsa/prim-algorithm
- https://youtu.be/cplfcGZmX7I

Prim’s algorithm

Example.

- https://www.thestudentroom.co.uk/attachment.php?attachmentid=23572&stc=1&d=1148396387

Prim’s Algorithm



5.3. Fundamental circuits and fundamental cut sets

Let $G(V,E)$ be a multigraph with n vertices, m edges and k connected components.

- **Cocyclomatic number** of the graph $G(V,E)$ is $\rho(G)=n-k$.

It is the total number of edges in spanning trees of all connected components of the graph.

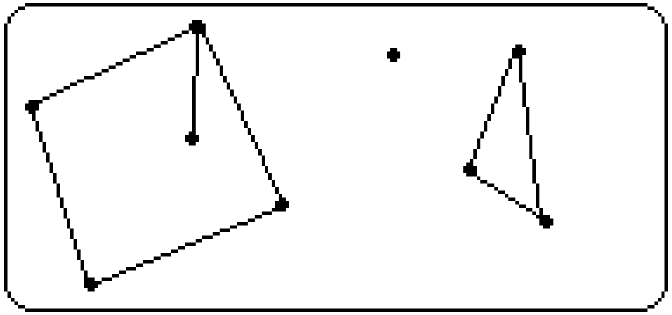
- **Cyclomatic number** of the graph $G(V,E)$ is $\nu(G)=m-n+k$.

It indicates ho many edges need to be removed in order to the graph became a forest with k connected components.

Cyclomatic and cocyclomatic numbers

Example. $\rho(G) = n - k = 9 - 3 = 6$;

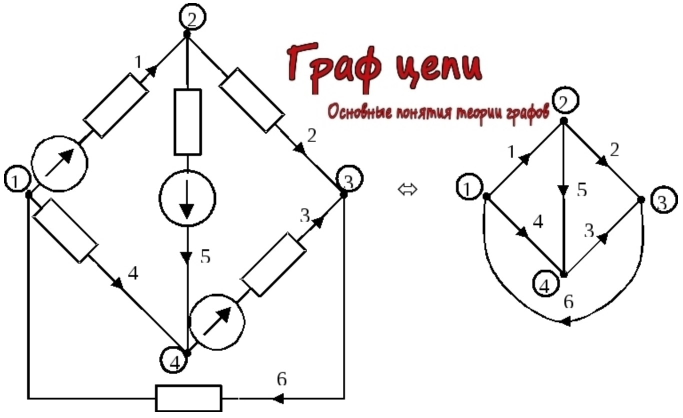
$\nu(G) = m - n + k = 8 - 9 + 3 = 2$.



Cyclomatic and cocyclomatic numbers

In the **theory of electrical circuits**, the numbers have a definite physical meaning.

- The **cyclomatic number** is equal to the largest number of independent circuits in the electric circuit graph, i.e. the largest number of independent circular currents that can flow in the circuit.
- The **cocyclomatic number** is equal to the number of independent potential differences between the nodes of the circuit.



Fundamental circuits

Any circuit or cycle can be represented by the set of its edges.

- **Modulo 2 addition (XOR):**

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

- **Conjunction (AND):**

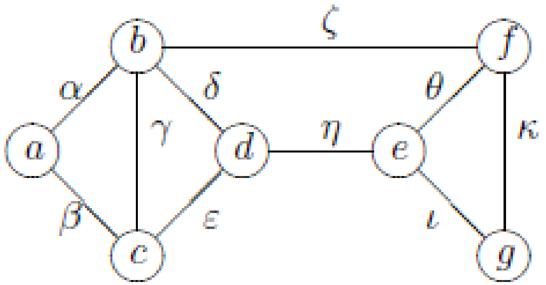
$$0\mu = \emptyset, \quad 1\mu = \mu.$$

- **Linear combination:**

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

Fundamental circuits

Example.

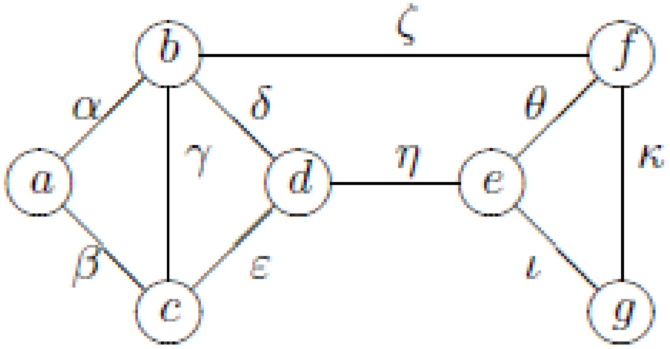


$\mu_1 = \{\delta, \zeta, \eta, \theta\};$
$\mu_2 = \{\gamma, \delta, \varepsilon\};$
$\mu_3 = \{\alpha, \beta, \gamma\};$
$1\mu_1 \oplus 1\mu_2 \oplus 0\mu_3 = \{\delta, \zeta, \eta, \theta\} \oplus \{\gamma, \delta, \varepsilon\} \oplus \emptyset = \{\gamma, \varepsilon, \zeta, \eta, \theta\}.$

Fundamental circuits

A set of circuits is **independent** if any circuit is not a linear combination of others; otherwise, the set is **dependent**.

Example. Set $\{\mu_1, \mu_2, \mu_3, \mu_4\}$ is dependent as $\mu_4 = \mu_2 + \mu_3$; set $\{\mu_1, \mu_2, \mu_4\}$ is independent

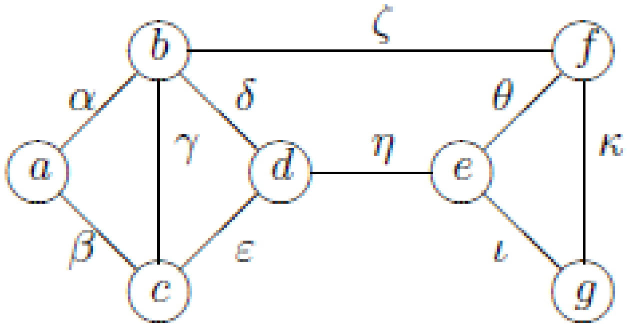


$$\mu_1 = \{\delta, \zeta, \eta, \theta\};$$
$$\mu_2 = \{\gamma, \delta, \varepsilon\};$$
$$\mu_3 = \{\alpha, \beta, \gamma\};$$
$$\mu_4 = \{\alpha, \beta, \delta, \varepsilon\}.$$

Fundamental circuits

An independent set of circuits is a **system of fundamental circuits** if it contains the greatest possible number of circuits; the circuits of this set are **fundamental**.

Example. Set $\{\mu_1, \mu_2, \mu_3, \mu_4\}$ is independent; any circuits is a linear combination of the circuits from the set.



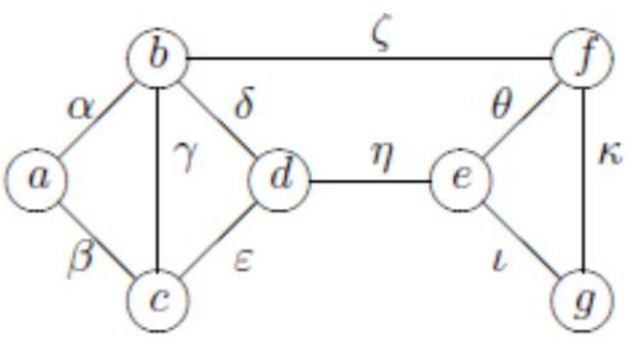
$$\mu_1 = \{\delta, \zeta, \eta, \theta\};$$
$$\mu_2 = \{\gamma, \delta, \varepsilon\};$$
$$\mu_3 = \{\alpha, \beta, \gamma\};$$
$$\mu_4 = \{\theta, \iota, \kappa\}.$$

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

Fundamental circuits theorem

Theorem. For a simple connected graph, the number of fundamental circuits is equal to ν(G)=$m-n+1$.

Example. Here n=7, m=10; there are 4=10−7+1 independent circuits.



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

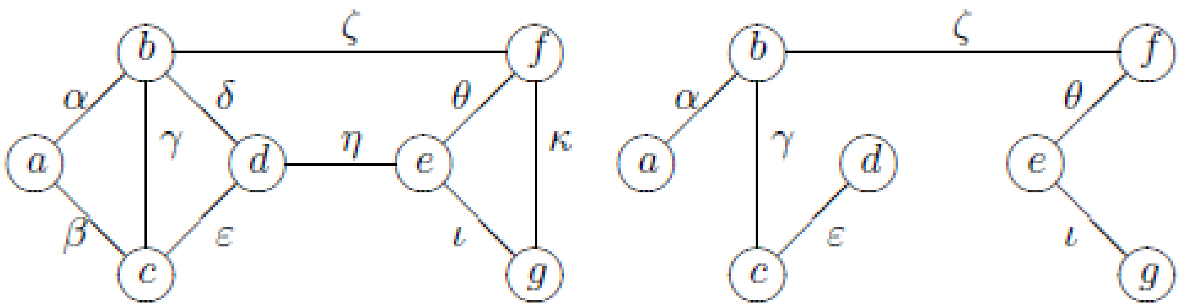
Fundamental cycles construction

Algorithm

- *Start.* There is graph $G(V,E)$.
- *Step 1*. Construct any spanning tree $T(V,E')$. Set j=0.
- *Step 2*. If $j = m-n+1$ then go to End; else set $j=j+1$.
- *Step 3.* Choose the next edge $e_j=(v_j,u_j)$ not included into the spanning tree.
- *Step 4*. Find the path $< v_j,u_j >$ in the spanning tree; together with the edge (v_j,u_j), it gives cycle Z_j. Go to Step 2.
- *End*. $\{Z_j\}$ is a system of fundamental cycles.

Fundamental cycles construction

Example.

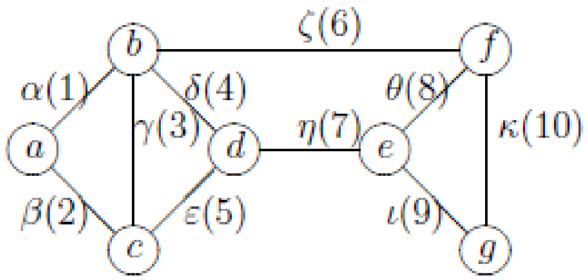


$$
\begin{array}{lll}
e_1 = \beta = (a,c), & < a,c >= abc, & Z_1 = \{\beta, \alpha, \gamma\}; \\
e_2 = \delta = (b,d), & < b,d >= bcd, & Z_2 = \{\delta, \gamma, \varepsilon\}; \\
e_3 = \eta = (d,e), & < d,e >= dcbfe, & Z_3 = \{\eta, \varepsilon, \gamma, \zeta, \theta\}; \\
e_4 = \kappa = (f,g), & < f,g >= feg, & Z_4 = \{\kappa, \theta, \iota\}.
\end{array}
$$

Matrix of fundamental circuits

- Rows correspond to fundamental circuits, columns correspond to edges; an element is equal to 1 iff the edge belongs to the circuit.

Example.



$Z_1 = \{\beta, \alpha, \gamma\};$
$Z_2 = \{\delta, \gamma, \varepsilon\};$
$Z_3 = \{\eta, \varepsilon, \gamma, \zeta, \theta\};$
$Z_4 = \{\kappa, \theta, \iota\}.$

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

Matrix of fundamental circuits

- **Module 2 product** of matrices A:n×k and B:k×m is matrix C:n×m calculated as follows

$$C_{ij} = \bigoplus_{l=1}^{k} A_{ik} B_{kj}.$$

- **Theorem.** If $I(G)$ is the incidence matrix of graph $G(V,E)$, $\Phi(G)$ is its matrix of fundamental circuits, then

$$I \oplus \Phi^T = 0$$

Fundamental cuts

Consider graph G(V,E) and two subsets

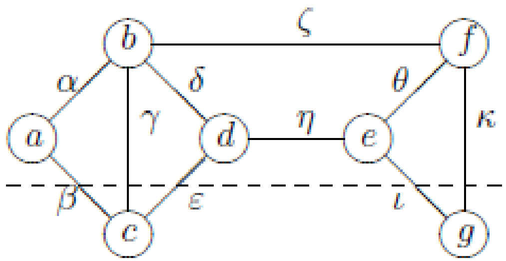
$$V = V_1 \cup V_2,\ V_1 \cap V_2 = \emptyset.$$

Cut (**cocycle**) $P(V_1,V_2)$ is the set of edges joining vertices from V_1 with vertices from V_2, i.e.

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

A cut is **proper** if after removal of any its subset the graph is connected.

Example. Non-proper cut P is union of proper cuts P_1 and P_2.



$$P(V_1, V_2) = \{\beta, \gamma, \varepsilon, \iota, \kappa\}$$

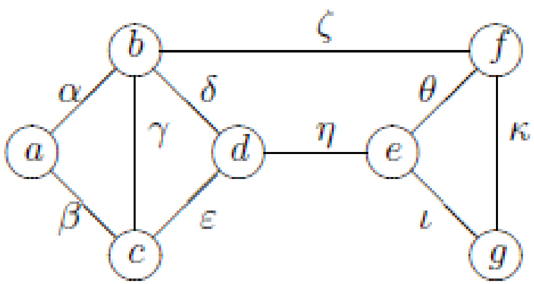
$$P_1 = \{\beta, \gamma, \varepsilon\} \text{ и } P_2 = \{\iota, \kappa\}.$$

Fundamental cuts

Lemma. Any non-proper cut is a union of disjoint proper cuts.

A set of cuts is **independent** if any cut is not a linear combination of others; otherwise, the set is **dependent**.

Example. Set $\{\psi_1, \psi_2, \psi_3, \psi_4\}$ is dependent as $\psi_3 = \psi_1 + \psi_2$; set $\{\psi_1, \psi_2, \psi_4\}$ is independent

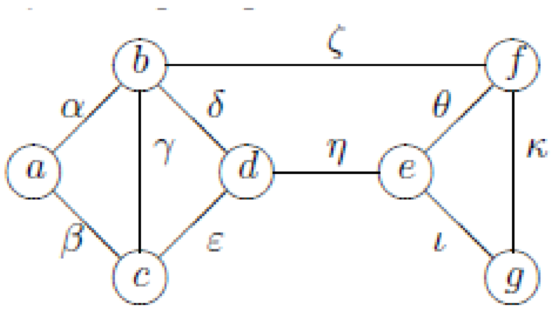


$\psi_1 = \{\beta, \gamma, \varepsilon, \zeta, \eta\}$;
$\psi_2 = \{\zeta, \eta, \iota, \kappa\}$;
$\psi_3 = \{\beta, \gamma, \varepsilon, \iota, \kappa\}$;
$\psi_4 = \{\zeta, \eta, \theta\}$.

Fundamental cuts

An independent set of cuts is a **system of fundamental cuts** if it contains the greatest possible number of cuts; the cuts of this set are **fundamental**.

Example. Set is independent; any cut is a linear combination of the cuts from the set.



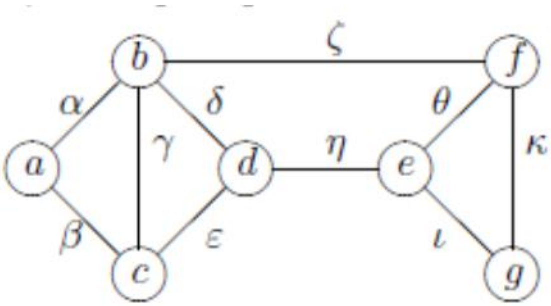
$$\begin{aligned}
\psi_1 &= \{\alpha, \beta\};\\
\psi_2 &= \{\delta, \varepsilon, \zeta\};\\
\psi_3 &= \{\zeta, \eta\};\\
\psi_4 &= \{\zeta, \theta, \iota\};\\
\psi_5 &= \{\iota, \kappa\};\\
\psi_6 &= \{\beta, \gamma, \varepsilon\}.
\end{aligned}$$

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

Fundamental cuts theorem

Theorem. For a simple connected graph, the number of fundamental cuts is equal to $\rho(G)=n-1$.

Example. Here n=7; there are 6=7−1 independent cuts.



$$\psi_1 = \{\alpha, \beta\};$$
$$\psi_2 = \{\delta, \varepsilon, \zeta\};$$
$$\psi_3 = \{\zeta, \eta\};$$
$$\psi_4 = \{\zeta, \theta, \iota\};$$
$$\psi_5 = \{\iota, \kappa\};$$
$$\psi_6 = \{\beta, \gamma, \varepsilon\}.$$

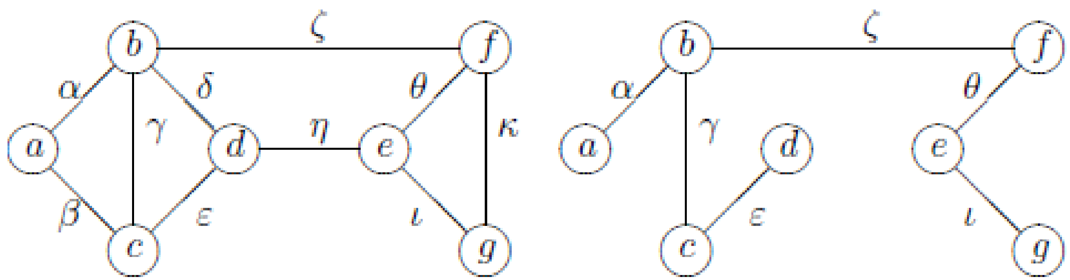
Fundamental cuts construction

Algorithm

- *Start.* There is graph $G(V,E)$.
- *Step 1*. Construct any spanning tree $T(V,E')$. Set $j=0$.
- *Step 2*. If $j=n-1$ then go to End; else set $j=j+1$.
- *Step 3.* Choose the next edge $e_j=(w_j,u_j)$ included into the spanning tree. Remove it from the tree and obtain a forest from two trees with the sets of vertices W_j and U_j.
- *Step 4*. Find the cut $Y_j=\mathrm{P}(W_j, U_j)$. Go to Step 2.
- *End*. $\{Y_j\}$ is a system of fundamental cuts.

Fundamental cuts construction

Example.

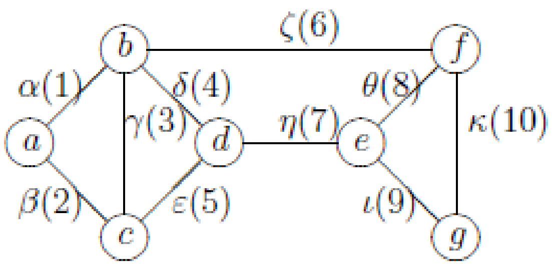


$$
\begin{array}{llll}
e_1 = \alpha = (a, b), & U_1 = \{a\}, & W_1 = \{b, c, d, e, f, g\}, & Y_1 = \{\alpha, \beta\}; \\
e_2 = \gamma = (b, c), & U_2 = \{a, b, f, e, g\}, & W_2 = \{c, d\}, & Y_2 = \{\beta, \gamma, \delta, \eta\}; \\
e_3 = \varepsilon = (c, d), & U_3 = \{a, b, c, e, f, g\}, & W_3 = \{d\},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

Matrix of fundamental cuts

- Rows correspond to fundamental cuts, columns correspond to edges; an element is equal to 1 iff the edge belongs to the cut.

Example.



$Y_1 = \{\alpha, \beta\};$
$Y_2 = \{\beta, \gamma, \delta, \eta\};$
$Y_3 = \{\delta, \varepsilon, \eta\};$
$Y_4 = \{\zeta, \eta\};$
$Y_5 = \{\eta, \theta, \kappa\};$
$Y_6 = \{\iota, \kappa\}.$

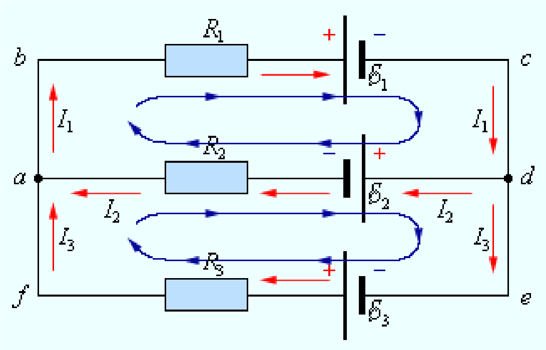
$$\Psi(G) = \begin{array}{c|cccccccccc} & \alpha & \beta & \gamma & \delta & \varepsilon & \zeta & \eta & \theta & \iota & \kappa \\ \hline Y_1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ Y_2 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ Y_3 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ Y_4 & 0 & 0 & 0 & 0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

Matrix of fundamental cuts

- **Theorem.** If *Ψ*(*G*) is the matrix of fundamental cuts of graph *G*(*V*,*E*), *Φ*(*G*) is its matrix of fundamental circuits, then

$$\Phi \oplus \Psi^T = 0$$

- **Kirchhoff's voltage law:** The algebraic sum of the products of the resistances of the conductors and the currents in them in a closed loop is equal to the total emf available in that loop.

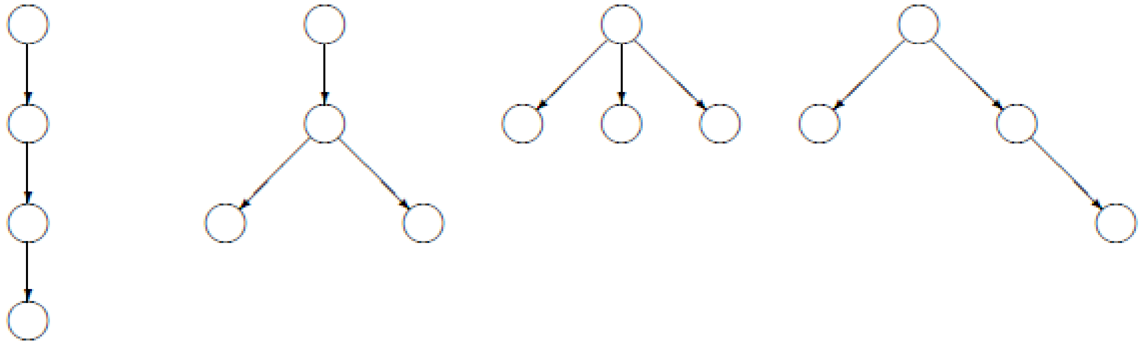


5.4. Rooted trees

Rooted tree is a digraph with the following properties:

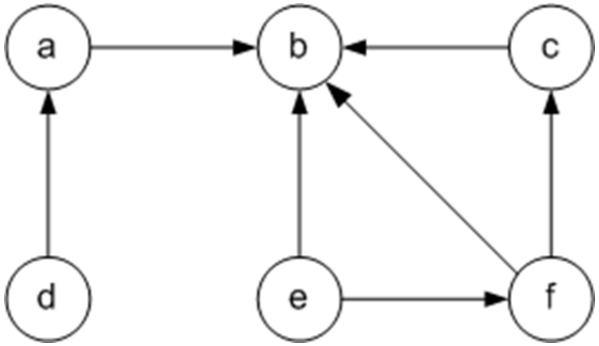
- there is a single node *v* with in-degree equal to 0 (it is called **root**);
- the in-degrees of all other nodes are equal to 1;
- each node is reachable from the root.

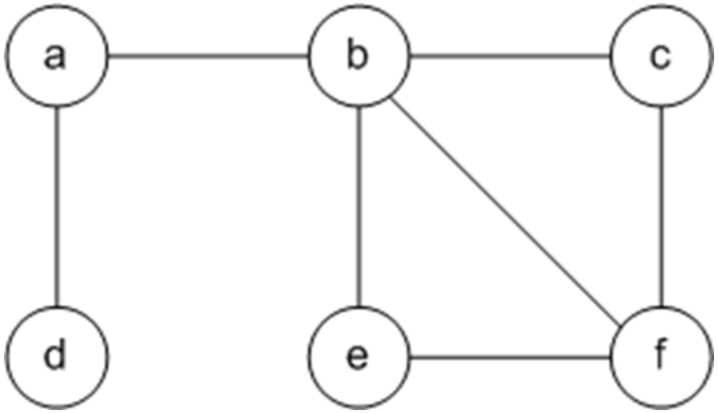
Example. All rooted trees with four vertices.



Properties of rooted trees

Underlying graph of a digraph D(V,E) is the graph obtained after cancelling of edge directions in E.





Properties of rooted trees

Theorem. Any directed tree has the following properties:

- $m=n-1$;
- the underlying graph of a rooted tree is a tree;
- any rooted tree does not have circuits;
- for every vertex v, there is the only path from the root to v;
- a subgraph induced by vertices reachable from vertex v is a rooted tree with the root v (it is called **subtree** of v);
- any undirected tree can be transformed into a rooted tree, and any vertex can be the root.

Properties of rooted trees

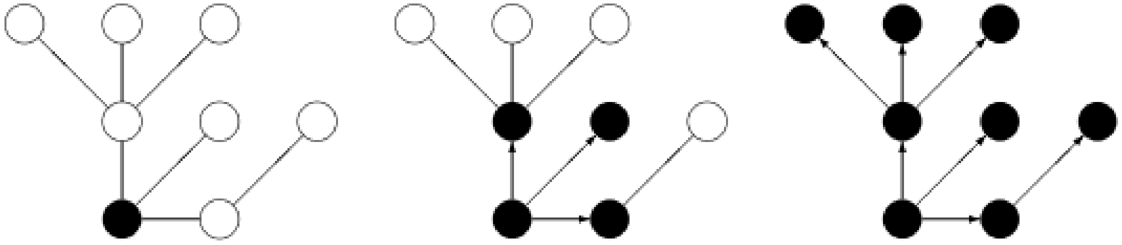
Proof.

- $m = \Sigma d^{-}(v) = n - 1$;
- the underlying graph is connected and m=n−1; so, it is a tree;
- any rooted tree does not have circuits because elsewhere the underlying graph has a circuit; so, is not a tree;
- for vertex v, if there are two paths from the root to v then the underlying graph has a circuit;

Properties of rooted trees

Proof.

- a subgraph induced by vertices reachable from vertex v is a rooted tree with the root v:
  - $d^{-}(v)=0$; elsewhere, there is a circuit;
  - every vertex w is reachable from v; so, $d^{-}(w)=1$;
- any undirected tree can be transformed into a rooted tree, and any vertex can be the root.



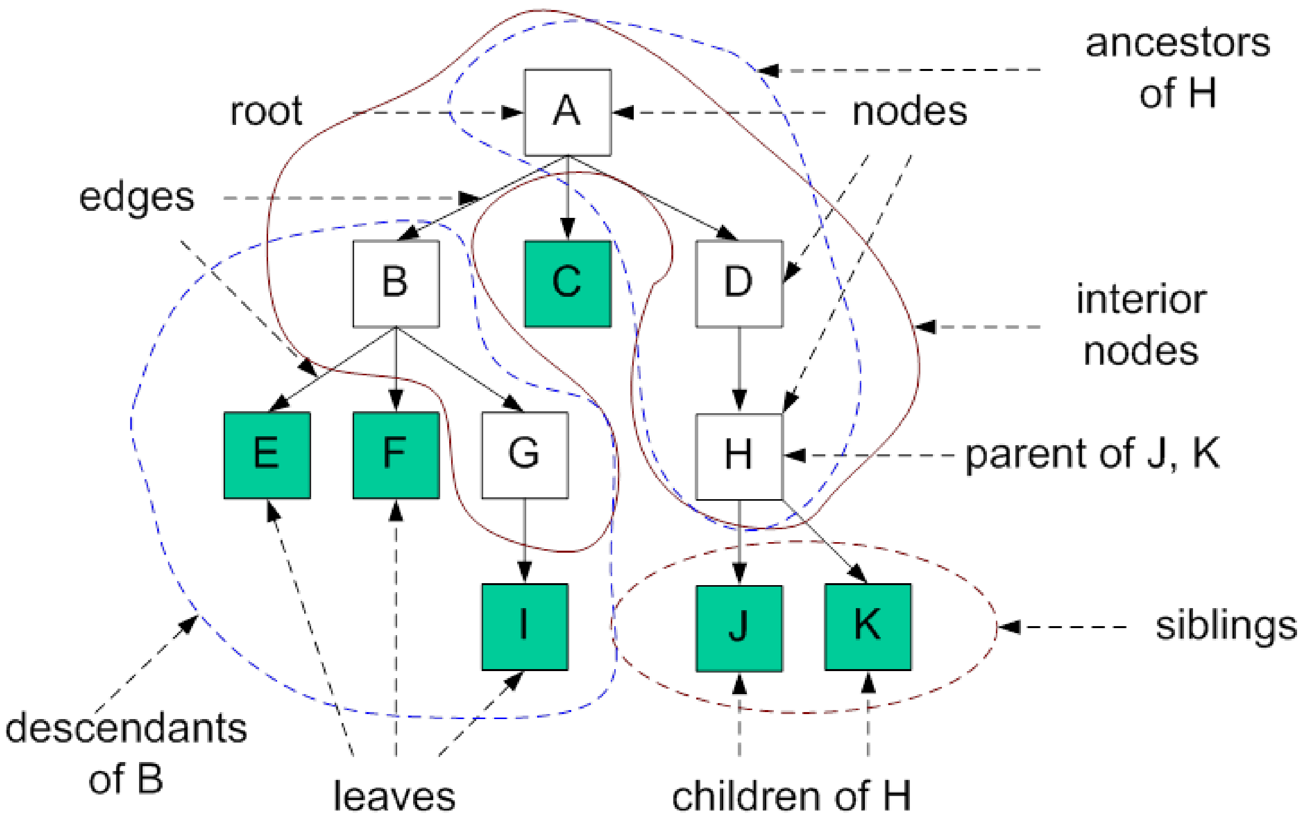
Terminology

- In a rooted tree, the **depth** or **level** of a vertex *v* is its distance from the root, i.e., the length of the unique path from the root to *v*. Thus, the root has depth 0.
- The **height** of a rooted tree is the length of a longest path from the root (or the greatest depth in the tree).
- If vertex v immediately precedes vertex *w* on the path from the root to *w*, then *v* is **parent** of *w* and *w* is **child** of *v*.
- Vertices having the same parent are called **siblings**.

Terminology

- A vertex w is called a **descendant** of a vertex v (and v is called an **ancestor** of w), if v is on the unique path from the root to w. If, in addition, $w \neq v$, then w is a **proper descendant** of v (and v is a **proper ancestor** of w).
- A **leaf** in a rooted tree is any vertex having no children.
- An **internal vertex** in a rooted tree is any vertex that has at least one child. The root is internal, unless the tree is trivial (i.e., a single vertex).

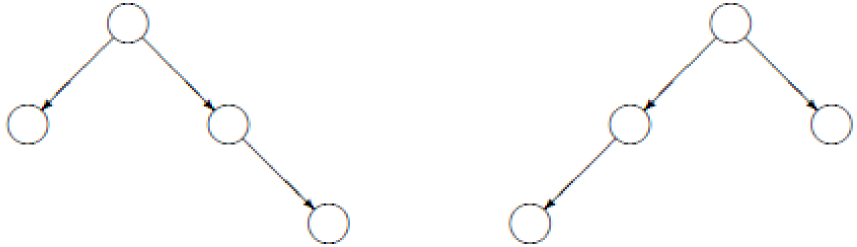
Terminology



Ordered trees

Ordered tree is a rooted tree with the fixed order of subtrees.

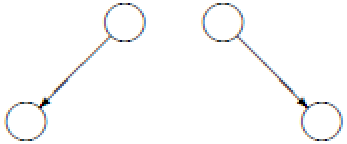
Example. These trees are isomorphic as *rooted* trees but they are not isomorphic as *ordered* trees.



Binary trees

Binary tree is an ordered tree where every vertex is a parent of exactly two siblings: **left** and **right** (can be empty).

Example. These trees are isomorphic as *rooted* trees and as *ordered* trees but they are not isomorphic as *binary* trees.



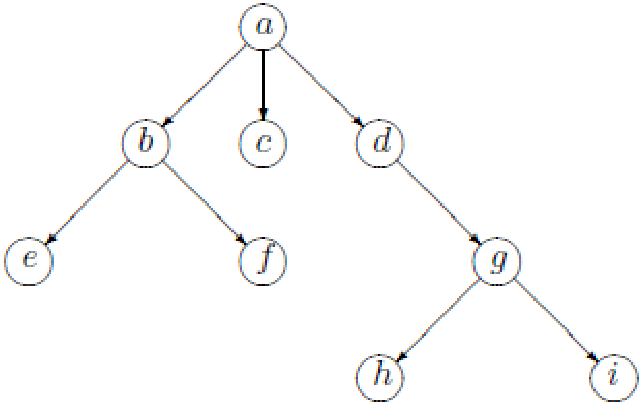
Tree traversal

- The **preorder traversal (root-left-right)**: visit the root; then, visit all subtrees from left to right.
- The **inorder traversal (left-root-right)**: visit the leftmost subtree; then, visit the root; after that, visit all other subtrees from left to right.
- The **postorder traversal (left-right-root)**: visit all subtrees from left to right; then, visit the root.

Tree traversal

Example.

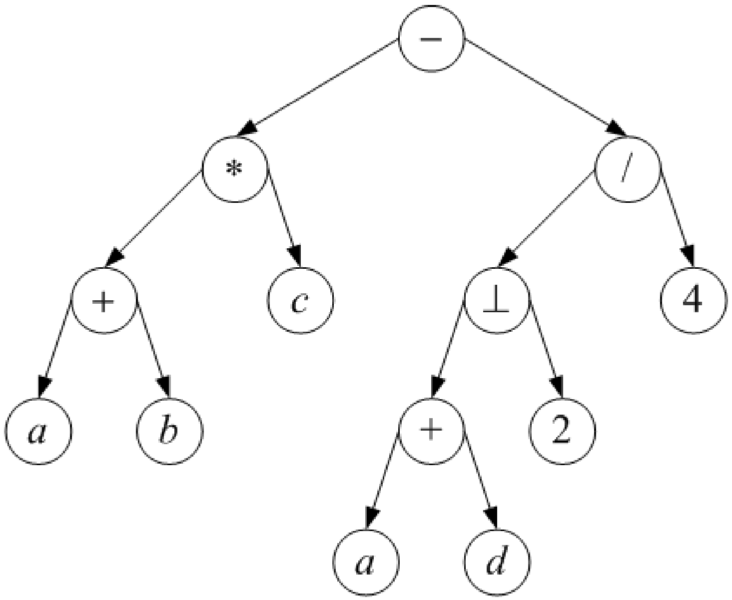
- The **preorder traversal**: *abefcdghi*.
- The **inorder traversal**: *ebfacdhgi*.
- The **postorder traversal**: *efbchigda*.



Tree traversal

Application: arithmetic expressions (in compilers).

Example. $(a+b)*c-(a+d)^2/4$



- The **preorder traversal**:

  − * + a b c / ↑ + a d 2 4 gives the **prefix form** or **Polish notation**.
- The **inorder traversal**: $((a+b)*c)-(((a+d)^2)/4)$ gives the **infix form**.
- The **postorder traversal**:

  a b + c * a d + 2 ↑ 4 / − gives the **postfix form** or **reverse Polish notation**.

5.5. Maximum branching

- **Branching** in a digraph is its subgraph where connected components are rooted trees.
- **Spanning branching** is a branching containing all vertices of the graph.
- **Maximum branching** in a weighted digraph is a branching of the maximum total weight of edges.

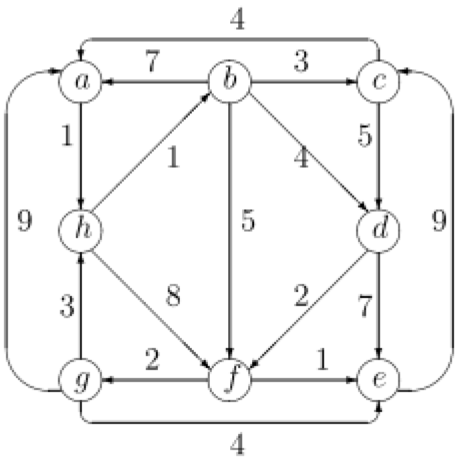
Example. Spanning branching {*ab*, *cd*}.

Edmonds algorithm (1958)

- **Start.** Graph G_0=G(V,E); buckets V0, V1,... and A0, A1,… are empty. Set i=0.
- **Step 1.** If all vertices of G_i, are in bucket V_i, go to step 3. Otherwise, select any vertex v in G_i, that is not in bucket Vi. Place vertex v into bucket V_i. Select an arc y with the greatest positive weight that is directed into v. If no such arc exists, repeat step 1; otherwise, place arc α into bucket A_i. If the arcs in A_i still form a branching repeat step 1; otherwise (if there is a cycle), go to step 2.

Edmonds algorithm

Example. **Step 1.** Cycle *ecde*. Go to Step 2.



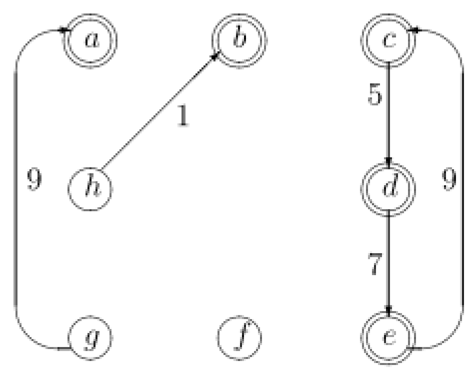
V_0	E_0
a	(g,a)

V_0	E_0
a	(g,a)
b	(h,b)

V_0	E_0
a	(g,a)
b	(h,b)
c	(e,c)

V_0	E_0
a	(g,a)
b	(h,b)
c	(e,c)
d	(c,d)

V_0	E_0
a	(g,a)
b	(h,b)
c	(e,c)
d	(c,d)
e	(d,e)



Edmonds algorithm

- **Step 2.** Arc α forms a cycle with some of the arcs in A_i. Call this cycle C_i.
- Shrink all the arcs and vertices in C_i, into a single vertex called v_i. Call this new graph G_{i+1}. Thus, any arc in G_i, that was incident to exactly one vertex in C_i, will be incident to vertex v_i, in graph G_{i+1}.
- Add all vertices from $V_i \backslash C_\mathrm{i}$ to V_{i+1}. Add all arcs from $A_i \backslash C_i$ to A_{i+1}.

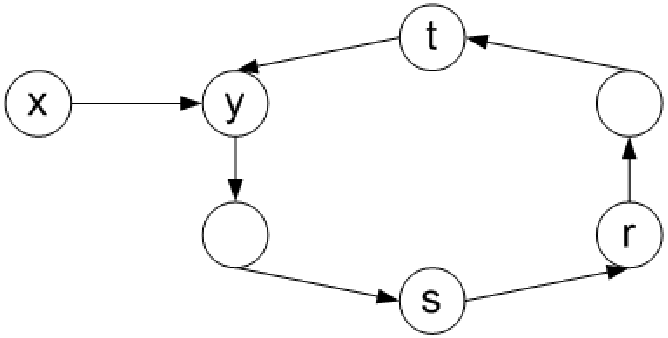
Edmonds algorithm

- Let the weight of each arc in G_{i+1} be the same as its weight in G_i except for the arcs in that are directed into v_i. For each arc (x, y) in G_i that transforms into an arc (x, v_i) in G_{i+1}, let

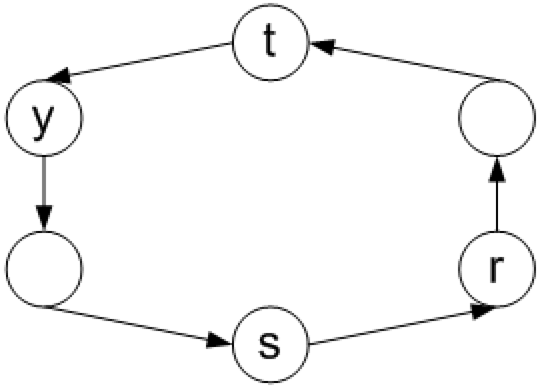
$$W(x,v_i) = W(x,y) - W(t,y) + W(s,r) .$$

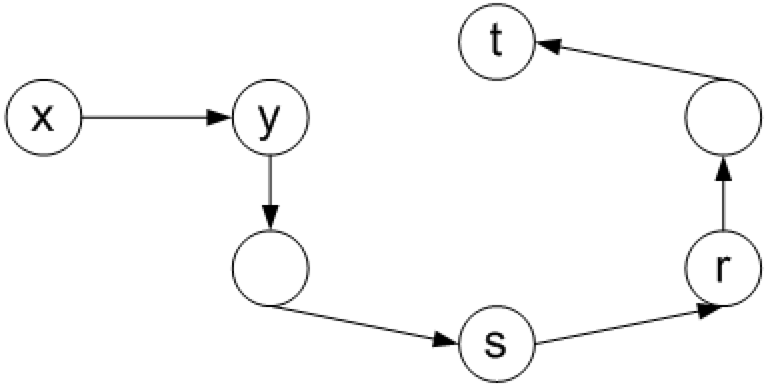
where (s,r) is the minimum weight arc in cycle C_i, and where (t,y) is the unique arc in cycle, whose tail is vertex y. Remove arcs with non-positive weights.

- Increase i by one, and return to step 1.

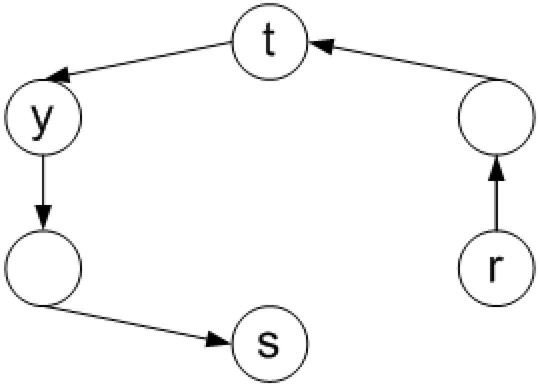


Edmonds algorithm





$\Delta = W(x,y) - W(t,y)$

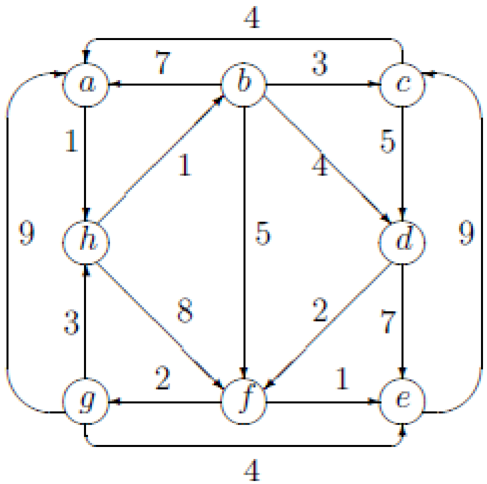


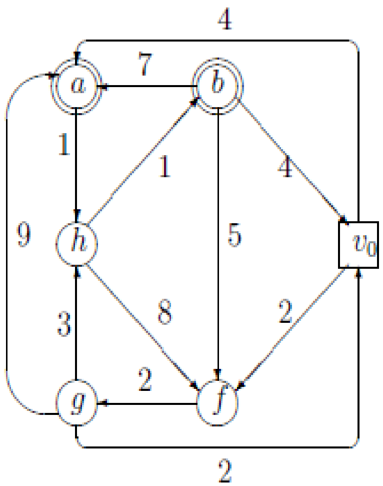
$\Delta = -W(s,r)$

Edmonds algorithm

Example. **Step 2.** Shrink cycle *ecde* and obtain pseudovertex v_0. Update the weights of the arcs going into the cycle. Remove arcs with negative weights. Go to Step 1.

$$C(b,v_0) = C(b,c) - C(e,c) + C(c,d) = 3 - 9 + 5 = -1;$$
$$C(b,v_0) = C(b,d) - C(c,d) + C(c,d) = 4 - 5 + 5 = 4;$$
$$C(f,v_0) = C(f,e) - C(d,e) + C(c,d) = 1 - 7 + 5 = -1;$$
$$C(g,v_0) = C(g,e) - C(d,e) + C(c,d) = 4 - 7 + 5 = 2.$$





V_1	E_1
a	(g,a)
b	(h,b)

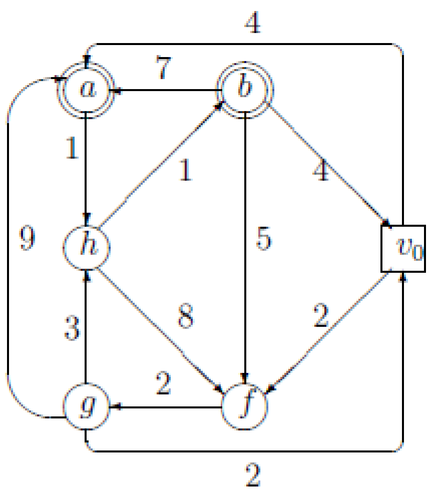
Edmonds algorithm

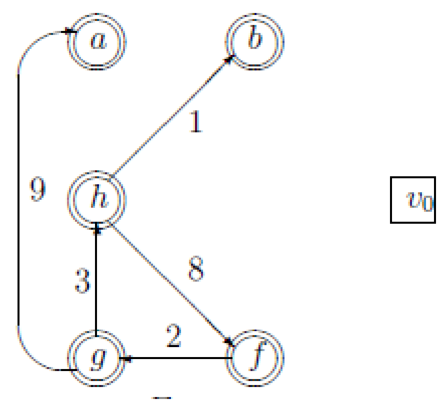
Example. **Step 1.** Sycle *fghf*.
Go to Step 2.

V_1	E_1
a	(g,a)
b	(h,b)
f	(h,f)

V_1	E_1
a	(g,a)
b	(h,b)
f	(h,f)
g	(f,g)

V_1	E_1
a	(g,a)
b	(h,b)
f	(h,f)
g	(f,g)
h	(g,h)

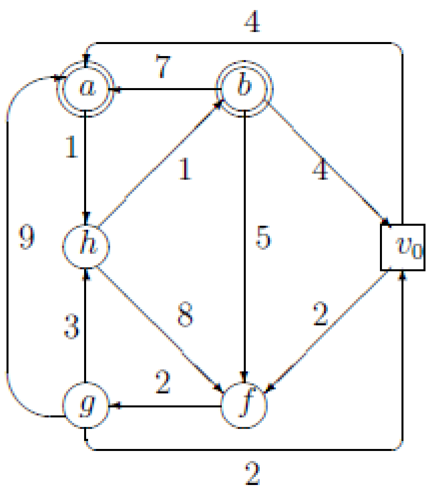


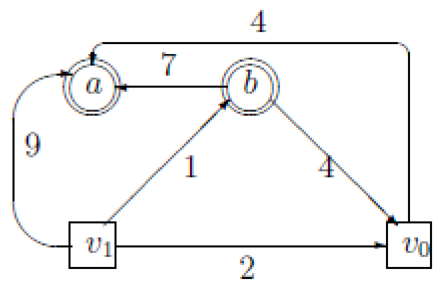


Edmonds algorithm

Example. **Step 2.** Shrink cycle *fghf* and obtain pseudovertex v_1. Update the weights of the arcs going into the cycle. Remove arcs with negative and zero weights. Go to Step 1.

$$C(a, v_1) = C(a, h) - C(g, h) + C(f, g) = 1 - 3 + 2 = 0;$$
$$C(b, v_1) = C(b, f) - C(h, f) + C(f, g) = 5 - 8 + 2 = -1;$$
$$C(v_0, v_1) = C(v_0, f) - C(v_0, f) + C(f, g) = 2 - 8 + 2 = -4.$$





V_2	E_2
a	(g, a)
b	(h, b)

Edmonds algorithm

- **Step 3.** This step is reached only when all vertices of G_i are in V_i, and the arcs in A_i, form a branching for G_i. If $i = 0$, stop because the arcs in A_0 form a maximum branching for G_0. Otherwise, two cases are possible:
- (a) Vertex v_i is the root of some tree in branching A_i, go to step 4.
- (b) Vertex v_i is not the root of some tree in branching A_i, go to step 5.

Edmonds algorithm

- **Step 4.** Restore cycle C_i and remove arc (s,r) with the minimum weight from C_i. Decrease i by 1 and go to step 3.
- **Step 5.** Restore cycle C_i. There is vertex y having two arcs going into y; remove arc (t,y) from C_i. Decrease i by 1 and go to step 3.

Edmonds algorithm

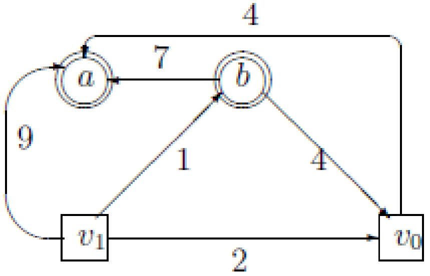
Example.

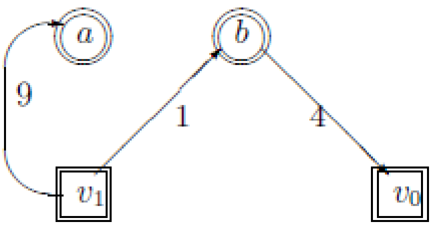
Step 1. All vertices are in the bucket, go to Step 3.

Step 3. As i=2 and v_1 is a root, go to Step 4.

V_2	E_2
a	(g, a)
b	(h, b)
v_0	(b, v_0)

V_1	E_1
a	(g, a)
b	(h, b)
v_0	(b, v_0)
v_1	



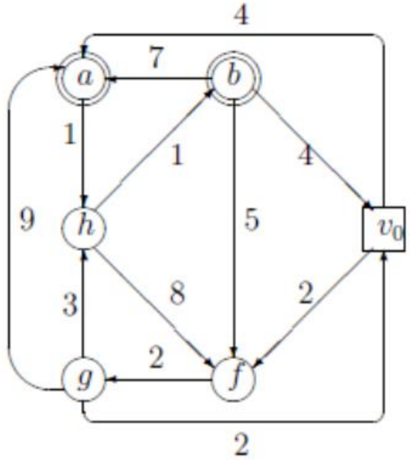


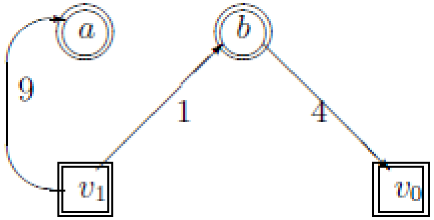
Edmonds algorithm

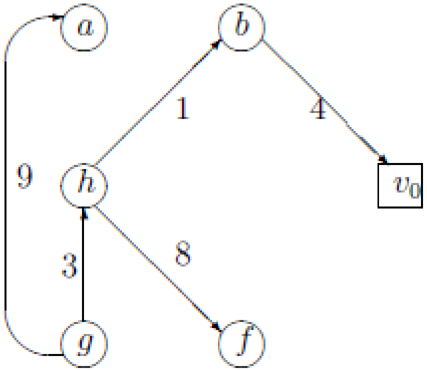
Example.

Step 4. Remove the arc of the minimum weight from *fghf*; it is (f,g). The others arcs from *fghf* together with E_2 give E_1. Set i=1 and go to Step 3.

Step 3. As i=1 and v_0 is not a root, go to Step 5.





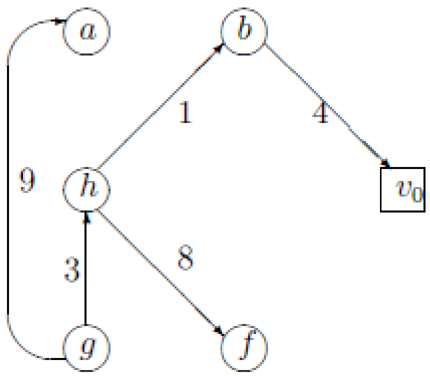


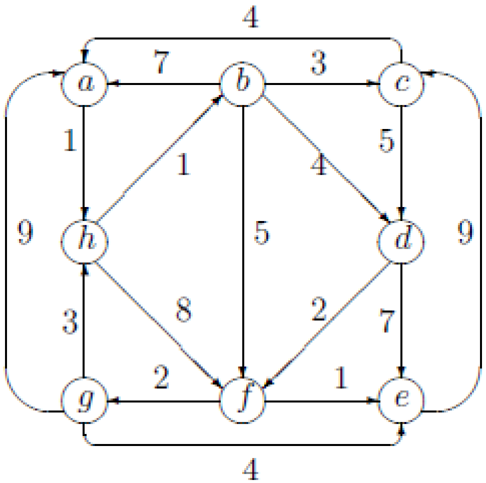
Edmonds algorithm

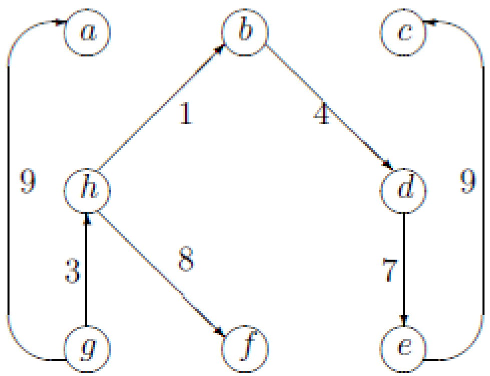
Example.

Step 5. In E_1, there is arc (b,v_0) corresponding to arc (b,d). Remove arc (c,d) from cycle *cdec*. The others arcs from *cdec* together with E_1 give E_0. Set i=0 and go to Step 3.

Step 3. As i=0, the maximum branching is constructed. Go to End.







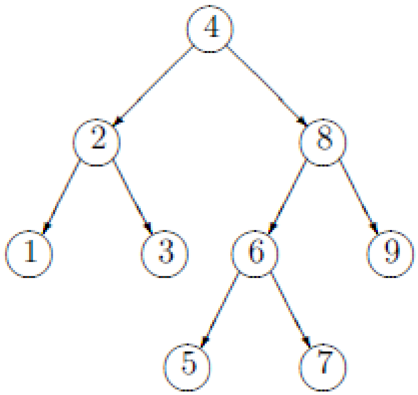
Related problems

- Minimum branching
- Maximum spanning tree
- Minimum spanning tree
- Maximum / minimum forest / spanning tree with the root in a specific vertex.

5.6. Search trees

- **Binary search tree (BST)** (also **sorted binary tree**) is a binary tree whose nodes each store a **key**. The tree additionally satisfies the **binary search property**: the key in each node must be greater than any key stored in the left subtree, and less than any key stored in the right subtree

Example.



BST operations

- Find a key
- Add a key
- Delete a key

Find a key

T is a tree, k is a key to find

TreeSearch(T,k)

- x ← root(T)
- if x = NULL or k = key(x) then return x
- if k < key(x) then return TreeSearch(left(T),k)
- else return TreeSearch(right(T),k)

Insert a key

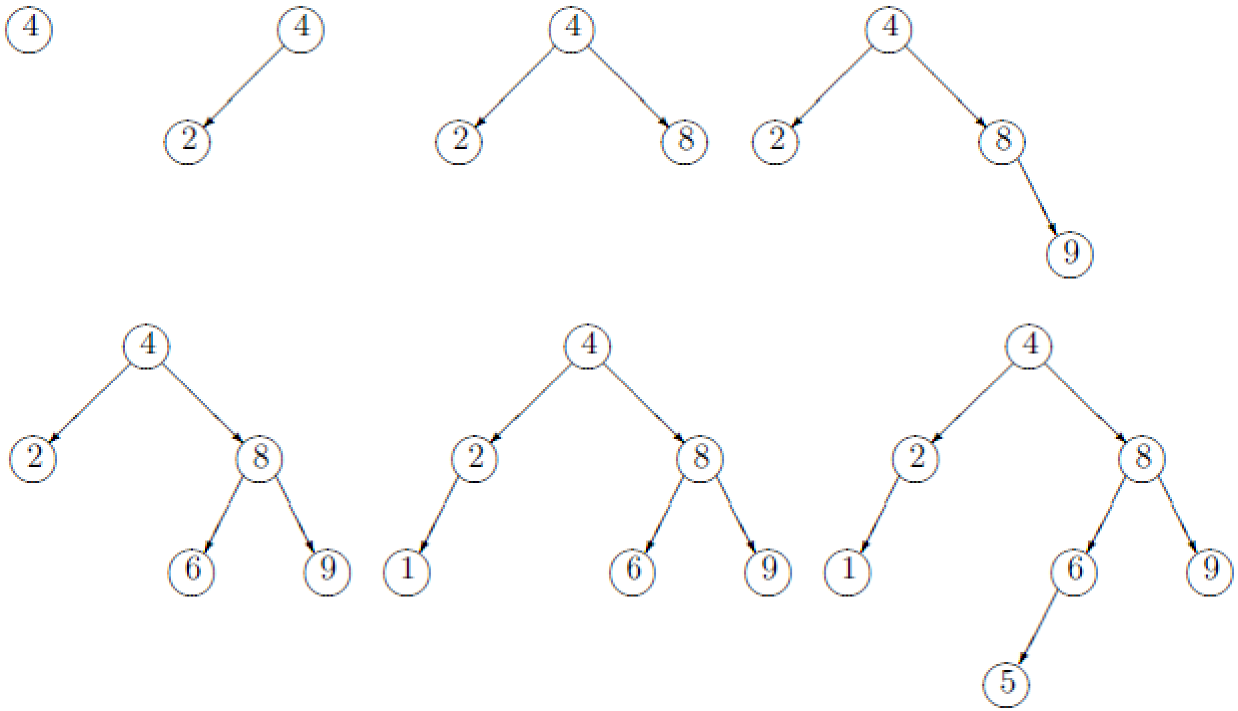
T is a tree, k is a key to insert

TreeInsert(T,k)

- x ← root(T)
- if x = NULL then
  - x ← k
  - return x
- if k = key(x) then return x
- if k < key(x) then return TreeInsert(left(T),k)
- else return TreeSearch(right(T),k)

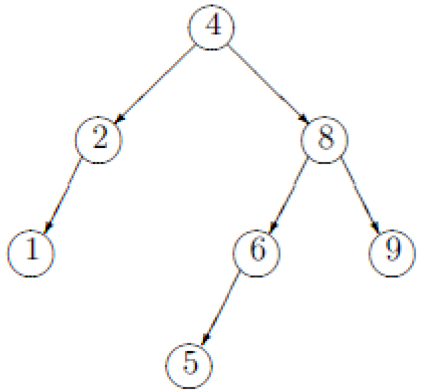
Insert a key

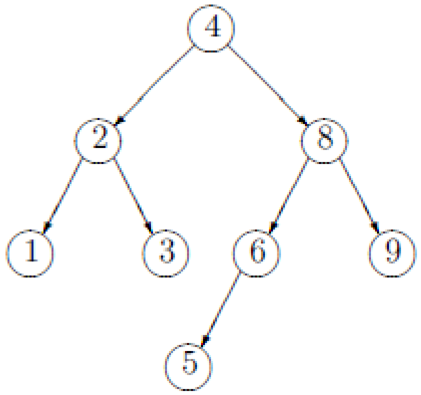
Example. Add 4, 2, 8, 9, 6, 1, 5, 3, 7.

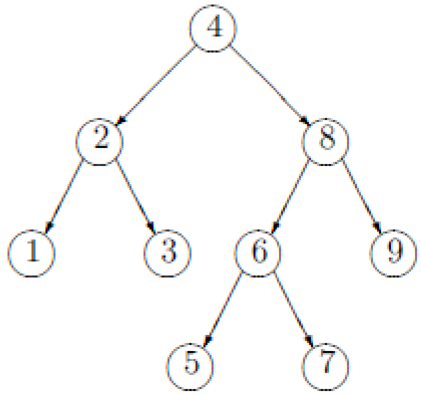


Insert a key

Example. Add 4, 2, 8, 9, 6, 1, 5, 3, 7.





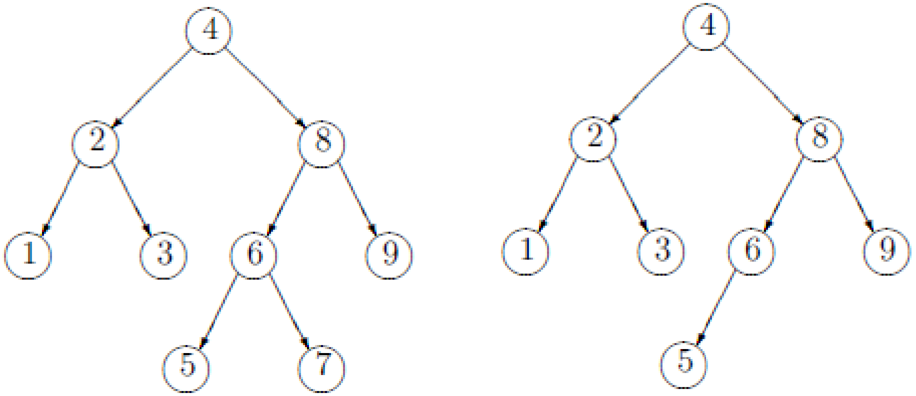


Delete a key

Case 1. The right subtree of the deleting node *a* is empty.

The left subtree of **node *a*** is connected to the parent of **node *a***, instead of **node *a***.

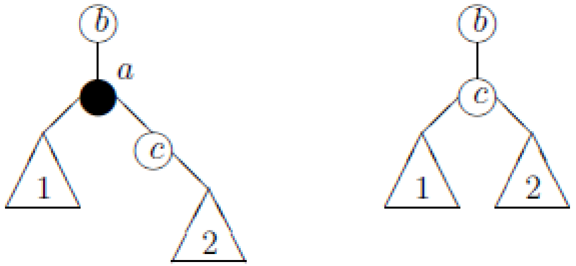
Example. Delete 7.



Delete a key

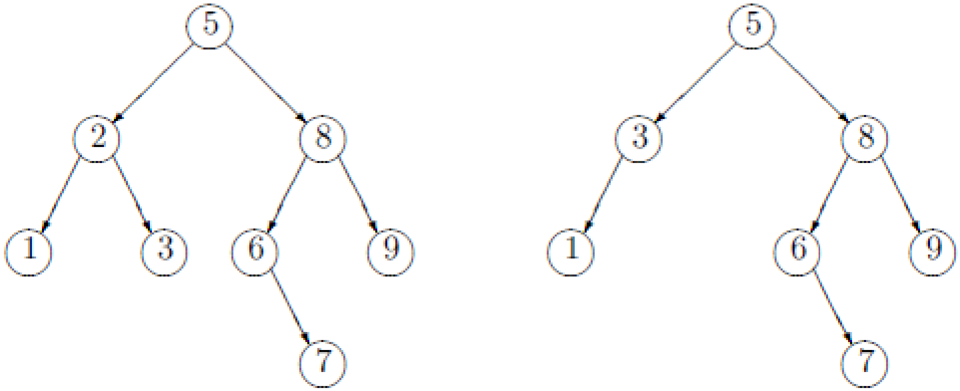
Case 2. The right subtree of the deleting node *a* is not empty. The right child of *a* is *c*; the left subtree of *c* is empty.

The left subtree of **node *a*** becomes the left subtree of **node *c***. Then, **node *c*** is connected to the parent of **node *a*** instead of **node *a***.



Delete a key

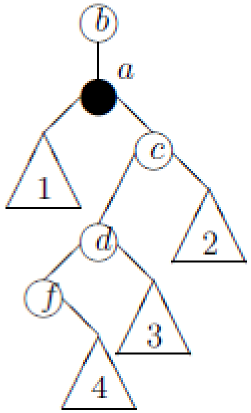
Example. Delete 2.

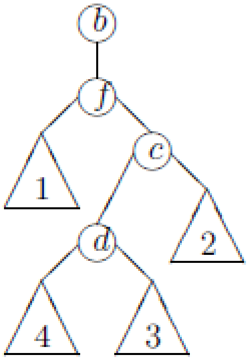


Delete a key

Case 3. The right subtree of the deleting node *a* is not empty. The right child of node *a* is node *c*; the left subtree of node *c* is not empty.

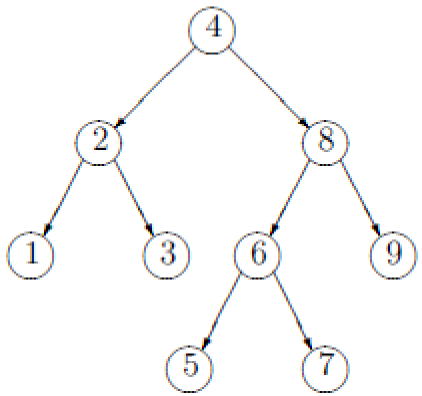
Find the leftmost **node *f*** in the right subtree of **node *a***. Put node f instead of node a. Connect the right subtree of **node *f*** to the previous parent of **node *f*** instead of **node *f***.

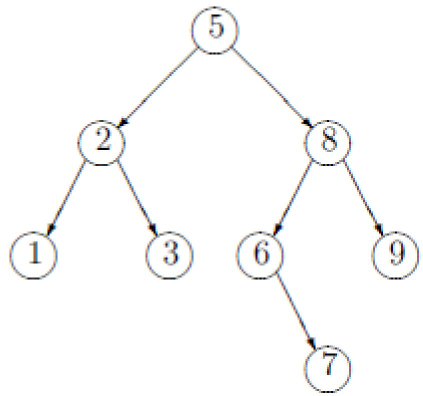




Delete a key

Example. Delete 4.





And three **symmetric cases**.

Computational complexity

h(*n*) − the height of a tree with *n* nodes.

Challenge: to decrease the height of a tree (*h*(*n*)=O(log(*n*))).

	Find	Insert	Delete
Unordered array	n	1	n
Ordered array	log(n)	n	n
Linked list	n	1	1
Tree	h(n)	1	h(n)

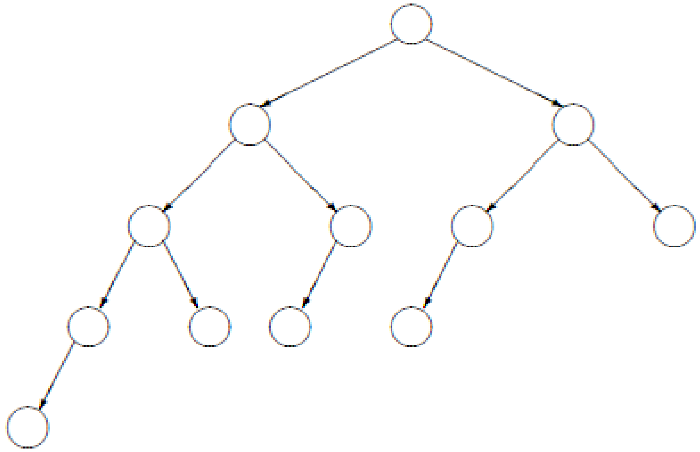
Balanced trees

BST is a **balanced tree** (**AVL-tree**) if:

- The left and right subtrees' heights differ by at most one;
- The left subtree is balanced;
- The right subtree is balanced.

AVL goes from *Adelson-Velskii* and *Landis*.

Example. Maximal asymmetric balanced tree.



Balanced trees

Theorem. For a balanced tree, $h(p) < 2\log_2 p$.

Proof. Let P_h be the number of vertices in the maximal asymmetric balanced tree.

$$p = P_h = P_{h-1} + P_{h-2} + 1.$$

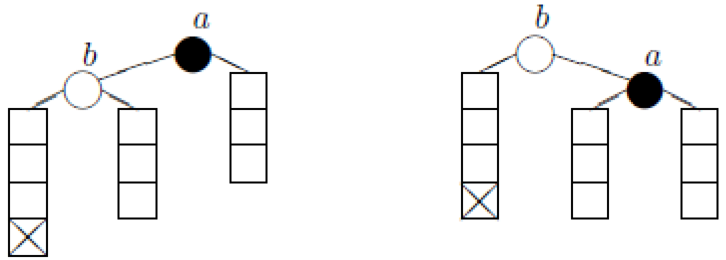
$$P_h \geq (\sqrt{2})^h$$

$$P_0 = 1 = (\sqrt{2})^0,\ P_1 = 2 \geq (\sqrt{2})^1 = \sqrt{2}.$$

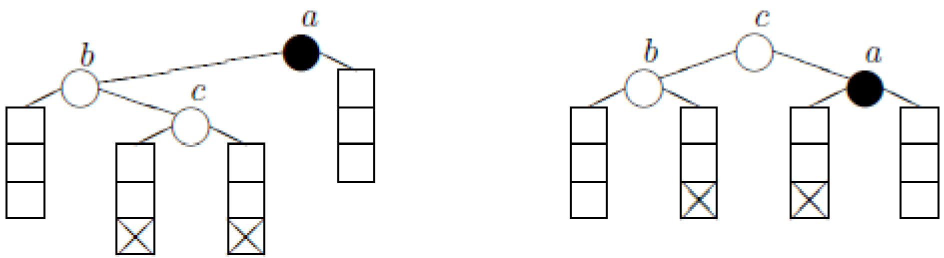
$$P_h = P_{h-1} + P_{h-2} + 1 \geq (\sqrt{2})^h + (\sqrt{2})^{h-1} + 1 = (\sqrt{2})^h \left(1 + \frac{1}{\sqrt{2}} + \frac{1}{(\sqrt{2})^h}\right) >$$
$$> (\sqrt{2})^h \left(1 + \frac{1}{\sqrt{2}}\right) > (\sqrt{2})^h \sqrt{2} = (\sqrt{2})^{h+1}.$$

Balancing

LL-rotation (**RR-rotation** is symmetric).

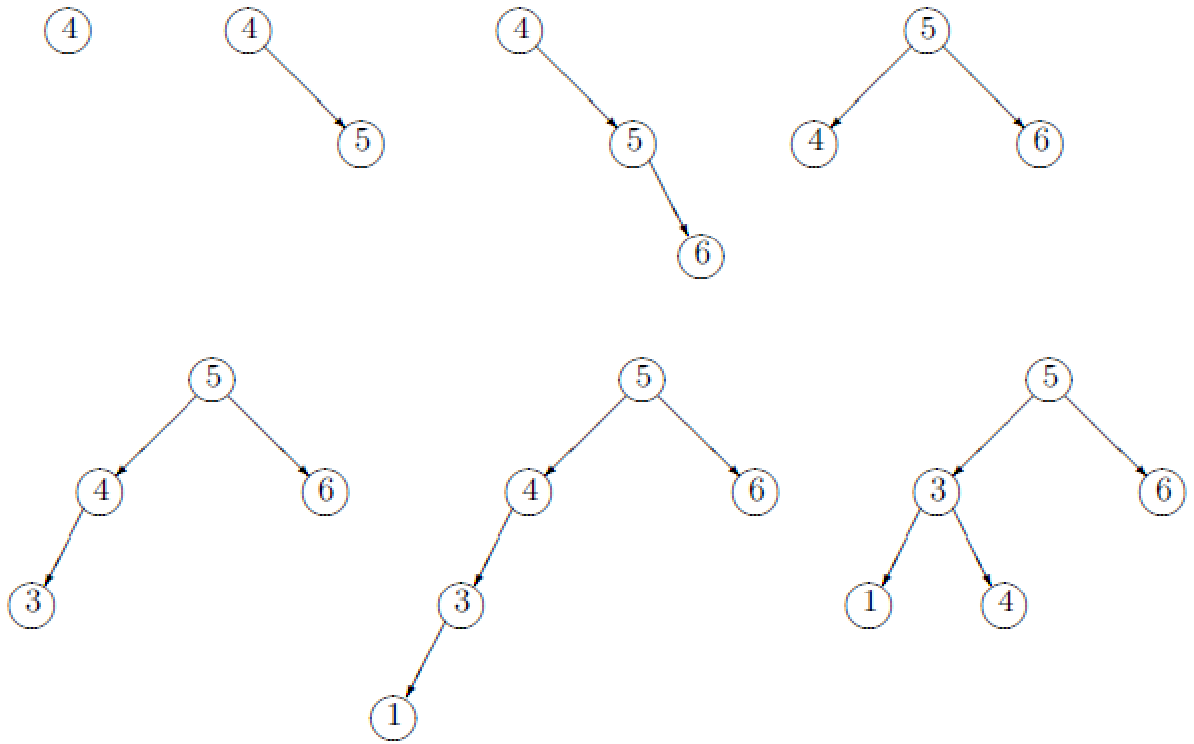


LR-rotation (**RL-rotation** is symmetric).



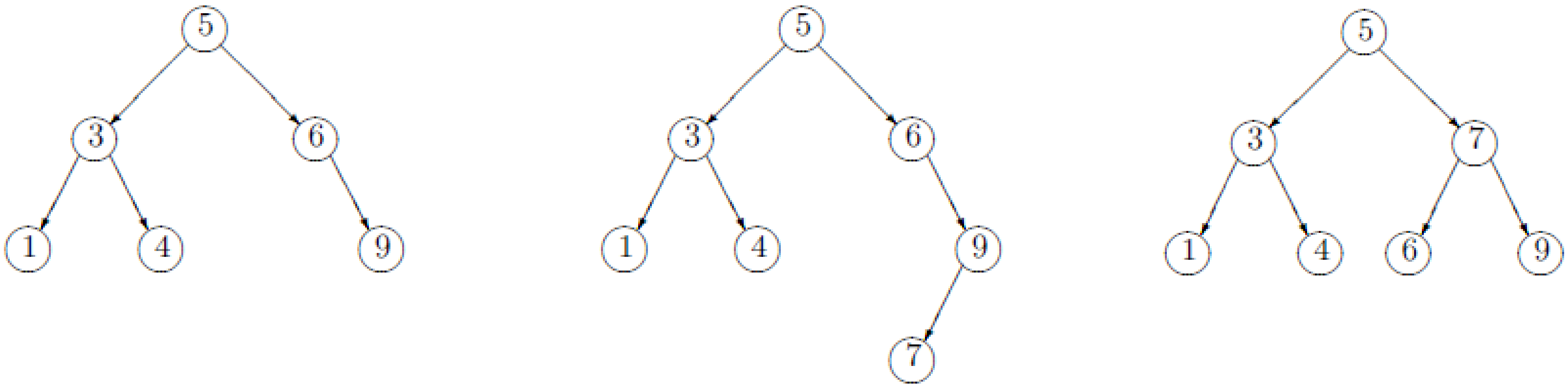
Balancing

Example. Insert 4, 5, 6; after adding 6, do RR-rotation. Insert 3, 1; after that, do LL-rotation.



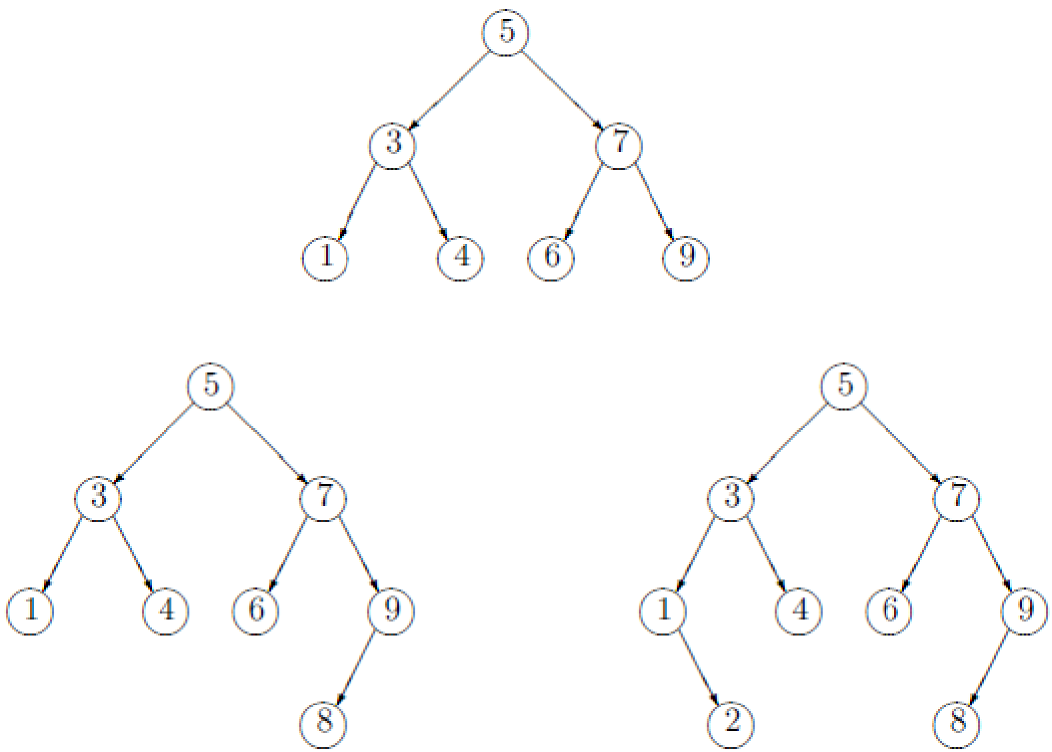
Balancing

Example. Insert 9, 7; node 6 is not balanced. Node 7 lengthens the left right subtree of the left subtree of node 6; hence, do RL-rotation.



Balancing

Example. Insert 8 and 2. The tree is balanced.



Red-black trees

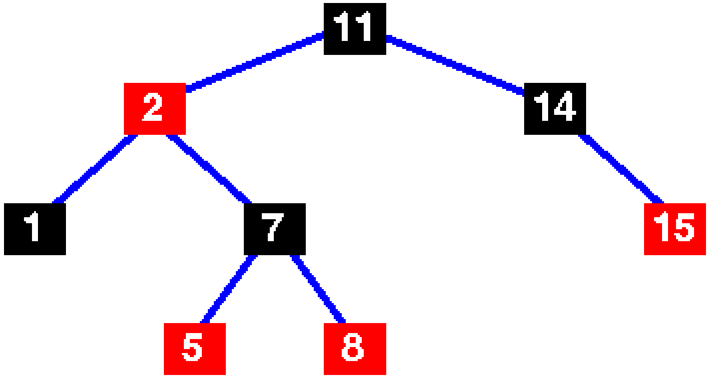
BST is a **red-black tree** if:

- Each node is either red or black.
- The root is black. This rule is sometimes omitted. Since the root can always be changed from red to black, but not necessarily vice versa, this rule has little effect on analysis.
- All leaves (NULL) are black.
- If a node is red, then both its children are black.
- Every path from a given node to any of its descendant NIL nodes contains the same number of black nodes.

Red-black trees

- The number of black nodes from the root to a node is the node's **black depth**.
- The uniform number of black nodes in all paths from root to the leaves is called the **black height** of the red–black tree.

Example. NULL leaves are omitted. The black height is 2. The black depth of 11 and 2 is 1; other nodes have the black height 2.



Red-black trees

Visualization

- https://www.cs.usfca.edu/~galles/visualization/RedBlack.html

Application

- GNU libstdc++ (/usr/include/c++/bits)
  std::map, std::multimap, std::set, std::multiset
- LLVM libc++
  std::map, std::set
- Java
  java.util.TreeMap, java.util.TreeSet
- Microsoft .NET 4.5 Framework Class Library
  SortedDictionary, SortedSet

Red-black trees

Operations

- https://www.youtube.com/watch?v=axa2g5oOzCE
- https://www.youtube.com/watch?v=PhY56LpCtP4
- https://www.youtube.com/watch?v=5IBxA-bZZH8
- https://www.youtube.com/watch?v=95s3ndZRGbk
- https://www.youtube.com/watch?v=7CesCbbVxqc