Университетские субботы-2021. Информатика.

Юлия Борисовна Буркатовская, ОИТ

Динамическое программирование

Ричард Эрнест Бе́ллман
(26 августа 1920— 19 марта 1984)

- *Беллман Р.* Динамическое программирование. — М.: Издательство иностранной литературы, 1960.
- *Кормен, Т., Лейзерсон, Ч., Ривест, Р., Штайн, К.* Глава 15. Динамическое программирование // Алгоритмы: построение и анализ = Introduction to Algorithms / Под ред. И. В. Красикова. — 2-е изд. — М.: Вильямс, 2005.

Динамическое программирование *— это когда у нас есть задача, которую непонятно как решать, и мы разбиваем ее на меньшие задачи, которые тоже непонятно как решать.* (с)
А. Кумок.

Динамическое программирование — способ решения сложных задач путём разбиения их на более простые подзадачи. Он применим к задачам с оптимальной подструктурой, выглядящим как набор перекрывающихся подзадач, сложность которых чуть меньше исходной. В этом случае время вычислений, по сравнению с «наивными» методами, можно значительно сократить.

Слово «программирование» в словосочетании «динамическое программирование» в действительности к «традиционному» программированию (написанию кода) почти никакого отношения не имеет и имеет смысл как в словосочетании «математическое программирование», которое является синонимом слова «оптимизация». Поэтому слово «программа» в данном контексте скорее означает оптимальную последовательность действий для получения решения задачи. (Википедия)

Для того, чтобы задача могла быть решена методом динамического программирования, должны быть выполнены следующие условия:

- в задаче можно выделить подзадачи аналогичной структуры, но меньшей размерности;
- среди выделенных подзадач есть тривиальные, то есть имеющие «малую размерность» и очевидное решение;
- оптимальное решение подзадачи большой размерности может быть построено из оптимальных решений подзадач;
- решения подзадач запоминаются в массивы, имеющие разумные размеры (обычно это одномерные или двумерные массивы, размер зависит от параметров исходной задачи).

Как правило, с помощью динамического программирования можно посчитать максимум, минимум, количество способов…

Пример 1. Классика – числа Фибоначчи.

$$F(n) = F(n-1) + F(n-2)$$
$$F(0) = 0, F(1) = 1$$

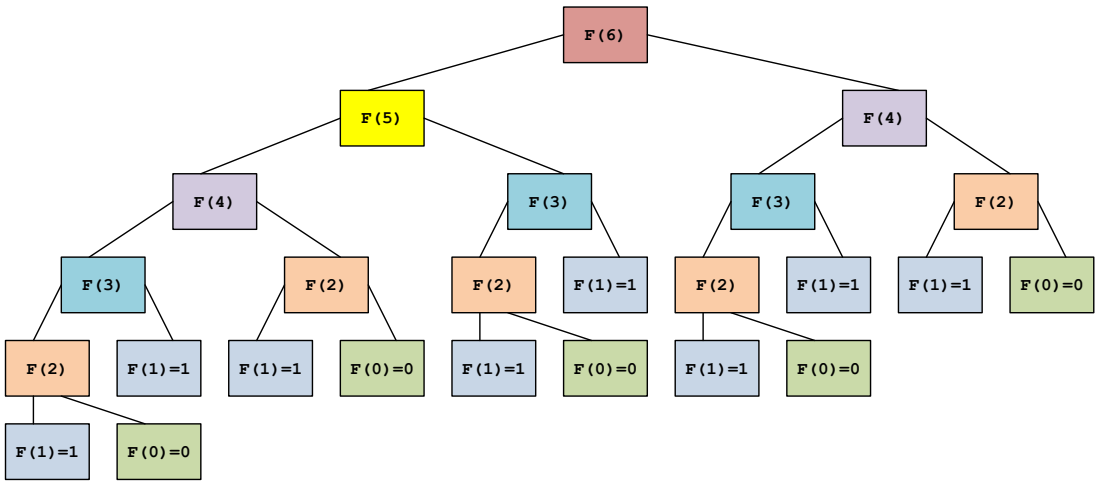
Рекурсивная реализация, непосредственно по формуле. В функцию добавлен вывод, для какого параметра n вызывается функция, для наглядности.

Pascal	C++	Python

function F(n:integer):integer;
begin
  writeln('Call F(',n,')');
  if (n<=1) then
  F:=n
  else
  F:=F(n-1)+F(n-2);
end;
 	
var n:integer;
begin
  readln(n);
  write(F(n));
end.
	#include <iostream>
	using namespace std;
	
	int F(int n) {
	  cout<<"Call F("<<n<<")";
	  cout<<endl;
	  if(n<=1) return n;
	  return (F(n-1)+F(n-2));
	}
	
	int main() {
	  int n;
	  cin >> n;
	  cout << F(n);
	  return 0;
	}
		def F(n):
		  print("Call F(",n,")")
		  if n<=1:
		  return n
		  return F(n-1)+F(n-2)
		
		n=int(input())
		print(F(n))

Результат работы программы для $n = 6$: Слева – выдача, справа – дерево, иллюстрирующее порядок вызовов функции для различных параметров. Можно видеть, что, чем меньше параметр, тем большее число раз вызывается функция. В целом число вызовов функции, а значит и сложность алгоритма, $O(2^n)$.

```
Call F(6)
Call F(5)
Call F(4)
Call F(3)
Call F(2)
Call F(1)
Call F(0)
Call F(1)
Call F(2)
Call F(1)
Call F(0)
Call F(3)
Call F(2)
Call F(1)
Call F(0)
Call F(1)
Call F(4)
Call F(3)
Call F(2)
Call F(1)
Call F(0)
Call F(1)
Call F(2)
Call F(1)
Call F(0)
8
```



Динамическое программирование:

1. Разбиваем задачу на подзадачи: для поиска $F(n)$ необходимо найти $F(n-1)$ и $F(n-2)$.
2. Запоминаем решения промежуточных задач, не решаем их каждый раз, когда они нужны.
3. База – $F(0)$ и $F(1)$ известны, вычисляем $F(2), F(3)$, ... до тех пор, пока не дойдем до $F(n)$.

Pascal	C++	Python

var n,i:integer;
  F: array[0..20] of integer;
begin
  readln(n);
  F[0]:=0;
  F[1]:=1;
  for i:=2 to n do
  F[i]:=F[i-1]+F[i-2];
  writeln(F[n]);
end
	#include <iostream>
	
	using namespace std;
	
	int main() {
	  int n,i;
	  cin >> n;
	  int F[n+1];
	  F[0]=0; F[1]=1;
	  for(i=2;i<=n;i++)
	  F[i]=F[i-1]+F[i-2];
	  cout << F[n];
	  return 0;
	}
		n=int(input())
		F=[0,1]
		for i in range(2,n+1):
		  F.append(F[i-1]+F[i-2])
		print(F[n])

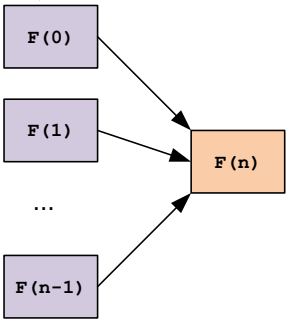
Время линейно (один цикл, в котором выполняется одно сложение).
Память линейна (завели массив). Важно – даже по памяти мы не выигрываем по сравнению с рекурсией, поскольку там память используется для сохранения в стек параметра функции, тоже хранится n параметров.
Можно обойтись без массива, если учесть, что мы используем лишь два последних вычисленных значения. Достаточно помнить только их.

Pascal	C++	Python

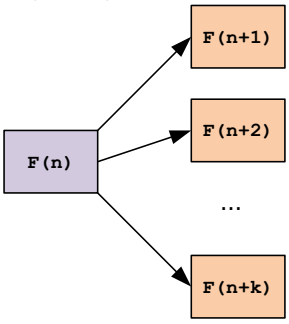
var n,i,x,y,f:integer;
begin
  readln(n);
  x:=0;
  y:=1;
  f:=n;
  for i:=2 to n do
  begin
  f:=x+y;
  x:=y;
  y:=f;
  end;
  writeln(f);
end.
	#include <iostream>
	
	using namespace std;
	
	int main() {
	  int n,i,x=0,y=1,f;
	  cin >> n;
	  f=n;
	  for(i=2;i<=n;i++)
	  {
	  f=x+y;
	  x=y;
	  y=f;
	  }
	  cout << f;
	  return 0;
	}
		n=int(input())
		x=0
		y=1
		f=n
		for i in range(2,n+1):
		  f=x+y
		  x=y
		  y=f
		print(f)

Одномерная динамика – задача зависит от одного параметра (вычисляем $F(n)$).

Прямая динамика: вычисляем $F(n)$ через $F(0), \dots, F(n-1)$.



Обратная динамика: имеем $F(n)$, вычисляем $F(n+1), \dots, F(n+k)$.



Термины не устоявшиеся, в разных источниках может быть по-разному.

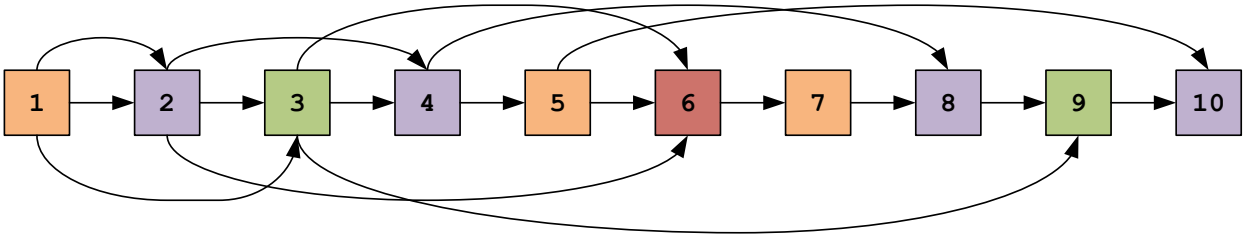
Задания с сайта https://informatics.msk.ru/

Пример 2. Задача №2963. Калькулятор

Имеется калькулятор, который выполняет три операции:

1. Прибавить к числу X единицу.
2. Умножить число X на 2.
3. Умножить число X на 3.

Определите, какое наименьшее число операций необходимо для того, чтобы получить из числа 1 заданное число N.



Прямая динамика. Число $k > 1$ можно получить из:

- $k-1$, прибавив 1;
- если k кратно 2, то из $k/2$, умножив на два;
- если k кратно 3, то из $k/3$, умножив на три.

Введем массив $a[k], k = 1, \dots, n$, где $a[k]$ – наименьшее количество команд, позволяющее получить число k из 1 (первоначально $a[1] = 0$). Просматриваем по возрастанию числа $k = 2, \dots, n$; сначала полагаем $a[k] = a[k-1] + 1$ (получаем k прибавлением единицы к $k-1$, общее число команд на 1 больше, чем для получения $k-1$). Затем проверяем делимость k на 2 и 3; если k кратно одному из этих чисел, то, возможно, выгоднее получить k либо из $k/2$ либо из $k/3$, количество команд в этом случае равно $a[k/2]+1$ или $a[k/3]+1$. Выбираем минимум.

Pascal	C++	Python

var n,k:longint;
a:array[0..1000000] of longint;
begin
readln(n);
a[1]:=0;
for k:=2 to n do
begin
 a[k]:=a[k-1]+1;
 if(k mod 2=0)then
  if(a[k]>a[k div 2]+1) then
  a[k]:=a[k div 2]+1;
 if(k mod 3=0)then
  if(a[k]>a[k div 3]+1) then
  a[k]:=a[k div 3]+1;
end;
writeln(a[n]);
end.
	#include<iostream>
	using namespace std;
	int main() {
	long n,k;
	cin >> n;
	long a[n+1];
	a[1]=0;
	for(k=2;k<=n;k++)
	{
	 a[k]=a[k-1]+1;
	 if((k%2==0)&&(a[k]>a[k/2]+1))
	  a[k]=a[i/2]+1;
	 if((k%3==0)&&(a[k]>a[k/3]+1))
	  a[k]=a[k/3]+1;
	}
	cout << a[n];
	return 0;
	}
		n=int(input())
		a=[0]*(n+1)
		a[1]=0
		for k in range(2,n+1):
		  a[k]=a[k-1]+1
		  if k%2==0 and a[k]>a[k//2]+1:
		  a[k]=a[k//2]+1
		  if k%3==0 and a[k]>a[k//3]+1:
		  a[k]=a[k//3]+1
		print(a[n])

Обратная динамика. Из числа k могут за один шаг быть получены числа $k+1$, $2k$ и $3k$. При этом число необходимых операций будет для всех этих чисел на 1 больше, чем для получения k. Изначально полагаем $a[k]=n, k>1$ (минимальное число операций всегда меньше n). Поэтому просматриваем все числа $k=2,\dots,n-1$, и, если $a[k+1]/a[2k]/a[3k]$ больше, чем $a[k]+1$, полагаем его равным $a[k]+1$. К тому же следим, чтобы не выйти за границы массива, последний его элемент имеет индекс n.

Pascal	C++	Python

var n,k:longint;
a:array[0..1000000] of longint;
begin
readln(n);
for k:=2 to n do
  a[k]:=n;
a[1]:=0;
for k:=1 to n-1 do
begin
if (a[k+1]>a[k]+1) then
  a[k+1]:=a[k]+1;
if(2*k<=n)then
  if(a[2*k]>a[k]+1) then
  a[2*k]:=a[k]+1;
if(3*k<=n)then
  if(a[3*k]>a[k]+1) then
  a[3*k]:=a[k]+1;
end;
writeln(a[n]);
end.
	#include<iostream>
	using namespace std;
	int main() {
	long n,k;
	cin >> n;
	long a[n+1];
	for(k=0;k<=n;k++)
	 a[k]=n;
	a[1]=0;
	for(k=1;k<n;k++)
	{
	 if(a[k+1]>a[k]+1)
	  a[k+1]=a[k]+1;
	 if((2*k<=n)&&(a[2*k]>a[k]+1))
	  a[2*k]=a[k]+1;
	 if((3*k<=n)&&(a[3*k]>a[k]+1))
	  a[3*k]=a[k]+1;
	}
	cout << a[n];
	return 0;
	}
		n=int(input())
		a=[n]*(n+1)
		a[1]=0
		for k in range(1,n):
		  if a[k+1]>a[k]+1:
		  a[k+1]=a[k]+1
		  if 2*k<=n and a[2*k]>a[k]+1:
		  a[2*k]=a[k]+1
		  if 3*k<=n and a[3*k]>a[k]+1:
		  a[3*k]=a[k]+1
		print(a[n])

Вспомним об эффективности. Так как самое большое число, которое нас интересует, это n, то на 3 имеет смысл умножать только числа, не большие $n/3$, а на 2 – числа, не большие $n/2$. Поэтому организуем три цикла: в первом рассмотрим числа от 2 до $n/3$, во втором – от $n/3+1$ до $n/2$ (эти числа на 3 уже не умножаем), в третьем – остальные (к ним только прибавляем 1). Программа выглядит более длинной, но мы избавились от $2n$ сравнений.

Pascal	C++	Python

var n,i,n2,n3:longint;
a:array[0..1000000] of longint;
begin
readln(n);
for i:=2 to n do
  a[i]:=n;
a[1]:=0;
n2:=n div 2;
n3:=n div 3;
for i:=1 to n3 do
begin
if (a[i+1]>a[i]+1) then
  a[i+1]:=a[i]+1;
if(a[2*i]>a[i]+1) then
  a[2*i]:=a[i]+1;
if(a[3*i]>a[i]+1) then
  a[3*i]:=a[i]+1;
end;
for i:=n3+1 to n2 do
begin
if (a[i+1]>a[i]+1) then
  a[i+1]:=a[i]+1;
if(a[2*i]>a[i]+1) then
  a[2*i]:=a[i]+1;
end;
for i:=n2+1 to n-1 do
if (a[i+1]>a[i]+1) then
  a[i+1]:=a[i]+1;
writeln(a[n]);
end.
	#include<iostream>
	using namespace std;
	int main() {
	long n,i,n2,n3;
	cin >> n;
	long a[n+1];
	for(i=0;i<=n;i++)
	a[i]=n;
	a[1]=0;
	n2=n/2;
	n3=n/3;
	for(i=1;i<=n3;i++)
	{
	 if(a[i+1]>a[i]+1)
	  a[i+1]=a[i]+1;
	 if(a[2*i]>a[i]+1)
	  a[2*i]=a[i]+1;
	 if(a[3*i]>a[i]+1)
	  a[3*i]=a[i]+1;
	}
	for(i=n3+1;i<=n2;i++)
	{
	 if(a[i+1]>a[i]+1)
	  a[i+1]=a[i]+1;
	 if(a[2*i]>a[i]+1)
	  a[2*i]=a[i]+1;
	}
	for(i=n2+1;i<n;i++)
	 if(a[i+1]>a[i]+1)
	  a[i+1]=a[i]+1;
	cout << a[n];
	return 0;
	}
		n=int(input())
		a=[n]*(n+1)
		a[1]=0
		n2=n//2
		n3=n//3
		for i in range(1,n3+1):
		  if a[i+1]>a[i]+1:
		  a[i+1]=a[i]+1
		  if a[2*i]>a[i]+1:
		  a[2*i]=a[i]+1
		  if a[3*i]>a[i]+1:
		  a[3*i]=a[i]+1
		for i in range(n3+1,n2+1):
		  if a[i+1]>a[i]+1:
		  a[i+1]=a[i]+1
		  if a[2*i]>a[i]+1:
		  a[2*i]=a[i]+1
		for i in range(n2+1,n):
		  if a[i+1]>a[i]+1:
		  a[i+1]=a[i]+1
		print(a[n])

Заметим, что жадные алгоритмы тут не работают.

Жадная идея 1. Давайте увеличивать данное число как можно сильнее, то есть, умножать на 3, а потом, когда это невозможно, умножать на 2, в последнюю очередь только прибавлять 1.

Контрпример. Жадный алгоритм:

$$1 \to 3 \to 9 \to 27 \to 81 \to 82 \to 83 \to 84 \to 85 \to 86 \to 87 \to 88 \to 89$$

Лучшее решение:

$$1 \to 3 \to 9 \to 10 \to 11 \to 22 \to 44 \to 88 \to 89$$

Жадная идея 2. Давайте уменьшать итоговое число как можно сильнее, то есть, делить на 3 или на 2, в последнюю очередь только вычитать 1.

Контрпример. Жадный алгоритм:

$$10 \leftarrow 5 \leftarrow 4 \leftarrow 2 \leftarrow 1$$

Лучшее решение:

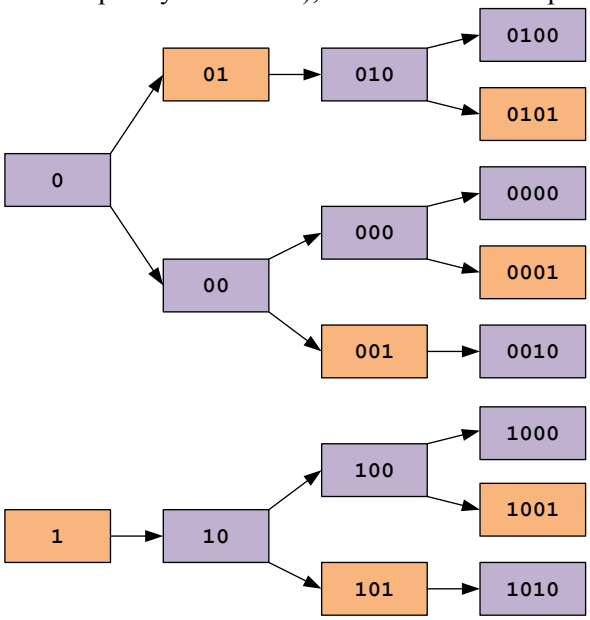
$$10 \leftarrow 9 \leftarrow 3 \leftarrow 1$$

Жадная идея 3?

Пример 3. Задача №207. Последовательность из 0 и 1

Требуется подсчитать количество последовательностей длины N, состоящих из 0 и 1, в которых никакие две единицы не стоят рядом.

Решение. Рассмотрим, как получаются последовательности большей длины из последовательностей меньшей длины. Если последовательности оканчиваются на 0 (в фиолетовых прямоугольниках), то к ним можно приписать как 0, так и 1. Если последовательность оканчивается на 1 (в оранжевых прямоугольниках), то к ним можно приписывать только 0.



Таким образом, последовательности длины $k+1$, оканчивающиеся на 1, получаются из последовательностей длины k, оканчивающихся на 0. Последовательности длины $k+1$, оканчивающиеся на 0, получаются из любых последовательностей длины k. Итак, рекуррентная формула (здесь $N0(k)$ и $N1(k)$ – количество последовательностей длины k, оканчивающихся на 0 и 1, соответственно):

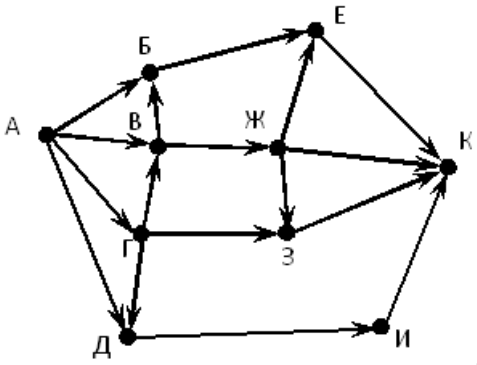
$$N1(k+1) = N0(k)$$
$$N0(k+1) = N0(k) + N1(k)$$
$$N0(1) = 1, N1(1) = 1$$

Осталось это запрограммировать.

Можно также заметить, что число всех последовательностей есть число Фиббоначчи, обоснуйте это самостоятельно при желании.

Задания с сайта https://inf-ege.sdamgia.ru/
ЕГЭ, задание 13. Поиск путей в графе.

Пример 4. Задание 13 № 3285
На рисунке — схема дорог, связывающих города А, Б, В, Г, Д, Е, Ж, З, И, К. По каждой дороге можно двигаться только в одном направлении, указанном стрелкой. Сколько существует различных путей из города А в город К?



Решение. Здесь подойдет идеология прямой динамики. Число путей в вершину x есть сумма числа путей в вершины $z_1, \ldots, z_k$, из которых есть ребро в вершину x. Обозначая число путей в вершину через имя самой вершины, получаем уравнения:

К=Е+Ж+З+И
И=Д
З=Ж+Г
Е=Б+Ж
Ж=В
Д=А+Г
Б=В+А
В=А+Г
Г=А

Уравнения в процессе составления упорядочиваем таким образом, чтобы величины, входящие в правую часть, всегда вычислялись ниже. Если это не так, можно переупорядочить уравнения позже.
Затем полагаем А=1, решаем уравнения «снизу вверх», подставляя уже известные величины в уравнения, пока не придем к результату.

К=Е+Ж+З+И=12
И=Д=2
З=Ж+Г=3
Е=Б+Ж=5
Ж=В=2
Д=А+Г=2
Б=В+А=3
В=А+Г=2
Г=А=1

Ответ: 12.

Аналогично решаются задачи с избегаемой вершиной (просто удаляем ее из графа), с обязательной вершиной (сначала ищем количество путей в обязательную, затем – из обязательной в финальную.

ЕГЭ, задание 23. Оператор присваивания и ветвления. Перебор вариантов, построение дерева

Динамическое программирование здесь работает лучше, чем дерево. Дерево часто получается слишком объемным, легко запутаться и что-то пропустить.

Пример 5. Задание 23 № 15932

Исполнитель РазДваТри преобразует число на экране.

У исполнителя есть три команды, которым присвоены номера:

1. Прибавить 1
2. Умножить на 2
3. Умножить на 3

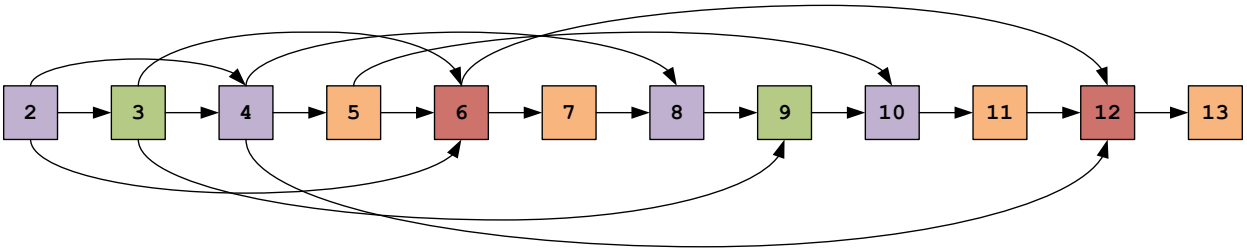
Первая команда увеличивает число на экране на 1, вторая умножает его на 2, третья умножает его на 3.

Программа для исполнителя РазДваТри — это последовательность команд.

Сколько существует программ, которые преобразуют исходное число 2 в число 44 и при этом траектория вычислений содержит число 13 и не содержит числа 29?

Траектория вычислений — это последовательность результатов выполнения всех команд программы. Например, для программы 312 при исходном числе 6 траектория будет состоять из чисел 18, 19, 38.

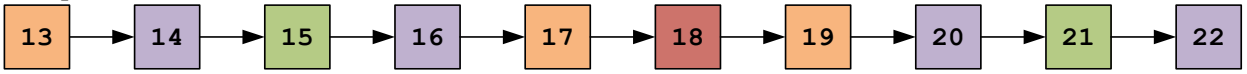
Решение. Сначала нам надо дойти от числа 2 до числа 13. Графически работу исполнителя можно представить следующим образом.



На самом деле можно даже этого не делать, просто держим в уме, что, если число k не кратно 2 или 3, оно получается только из $k-1$; числа, кратные 2 или 3, получаются двумя способами (еще и из $k/2$ или $k/3$); числа, кратные 6 – из $k-1$, $k/2$, $k/3$. Составляем систему уравнений (тут можно идти от меньшего к большему).

$$N_2 = 1$$
$$N_3 = N_2 = 1$$
$$N_4 = N_2 + N_3 = 2$$
$$N_5 = N_4 = 2$$
$$N_6 = N_2 + N_3 + N_5 = 4$$
$$N_7 = N_6 = 4$$
$$N_8 = N_4 + N_7 = 6$$
$$N_9 = N_3 + N_8 = 7$$
$$N_{10} = N_5 + N_9 = 9$$
$$N_{11} = N_{10} = 9$$
$$N_{12} = N_4 + N_6 + N_{11} = 15$$
$$N_{13} = N_{12} = 15$$

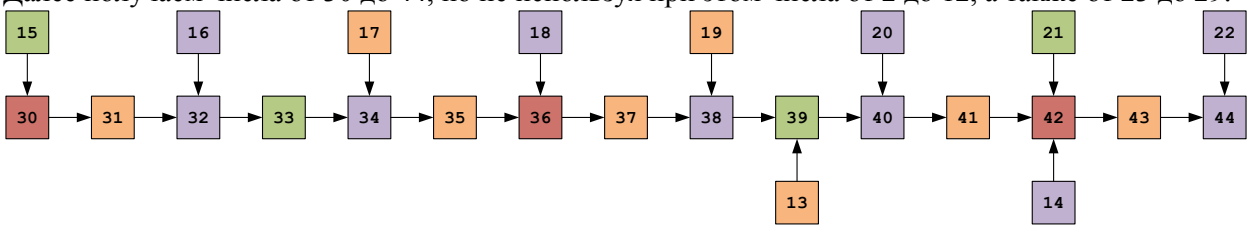
Далее посчитаем число траекторий от 13 до 44, не включающих 29. Это означает, что 30 нельзя получить из 29, прибавлением 1. К тому же нам не нужны числа от 23 до 28, если мы будем прибавлять к ним 1, то придем к числу 29, если умножать, то получим число, большее 44. Поэтому нас будут интересовать числа от 14 до 22, и далее от 30 до 44. При этом все числа мы получаем из 13, то есть, например, нельзя прийти в 22 из 11. По сути, мы стартуем с 13. При этом из 13 до 22 мы приходим только добавлением единицы.



Отсюда

$$N_{22} = \cdots = N_{14} = N_{13} = 15$$

Далее получаем числа от 30 до 44, но не используя при этом числа от 2 до 12, а также от 23 до 29.



Для простоты расчетов можно положить

$$N_{22} = \cdots = N_{14} = N_{13} = 1,$$

а потом умножить полученное число на 15 (но это не обязательно).

$$N_{30} = N_{15} = 1$$
$$N_{31} = N_{30} = 1$$
$$N_{32} = N_{16} + N_{31} = 2$$
$$N_{33} = N_{32} = 2$$
$$N_{34} = N_{17} + N_{33} = 3$$
$$N_{35} = N_{34} = 3$$
$$N_{36} = N_{18} + N_{35} = 4$$
$$N_{37} = N_{36} = 4$$
$$N_{38} = N_{19} + N_{37} = 5$$
$$N_{39} = N_{13} + N_{38} = 6$$
$$N_{40} = N_{20} + N_{39} = 7$$
$$N_{41} = N_{40} = 7$$
$$N_{42} = N_{14} + N_{21} + N_{41} = 9$$
$$N_{43} = N_{42} = 9$$
$$N_{44} = N_{22} + N_{43} = 10$$

Умножаем 15 на 10, получаем 150 программ.

Ответ: 150.

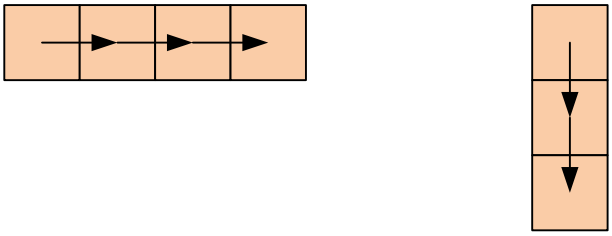
Все это можно не рассчитывать вручную, а вывести формулы и запрограммировать. Программа будет похожа на ту, что мы писали для калькулятора в примере 2.

Двумерная динамика – задача зависит от двух параметров (вычисляем $F(i,j)$). Промежуточные решения часто хранятся в двумерном массиве.

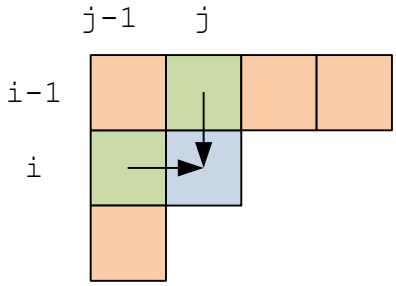
Задания с сайта https://informatics.msk.ru/

Пример 6. Задача №206. Количество маршрутов в прямоугольной таблице
В прямоугольной таблице $N \times M$ в начале игрок находится в левой верхней клетке. За один ход ему разрешается перемещаться в соседнюю клетку либо вправо, либо вниз (влево и вверх перемещаться запрещено). Посчитайте, сколько есть способов у игрока попасть в правую нижнюю клетку.

Решение. Разобьем задачу на небольшие подзадачи, которые все же понятно, как решать. Например, если таблица состоит из одной строки, то игрок движется только вправо, и способ будет один. Аналогично, если таблица состоит из одного столбца, игрок движется только вниз, и способ один.



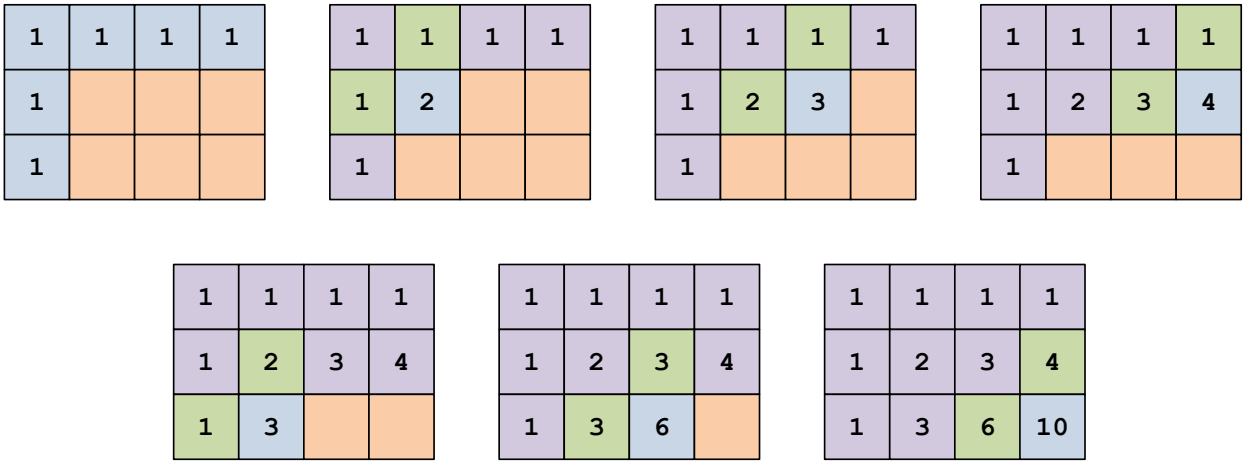
Рассмотрим более сложную таблицу. В клетку, выделенную голубым цветом, можно попасть из клеток слева и сверху (выделены зеленым). Если мы знаем, сколько путей ведут в зеленые клетки, мы получим число путей, ведущих в голубую клетку, как сумму этих чисел.



Таким образом, мы получили алгоритм. Заводим массив $a: N \times M$. Заполняем его первую строку и первый столбец единицами. Затем просматриваем строки слева направо, начиная со второй строки и со второй позиции в этой строке, вычисляем количество путей по формуле:

$$a_{i,j} = a_{i-1,j} + a_{i,j-1}.$$

Ответом будет число в последней строке и последнем столбце.



Pascal	C++	Python

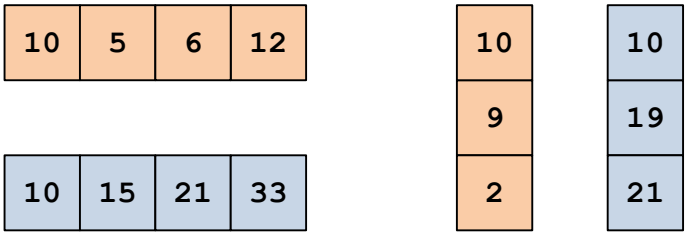
var n,m,i,j:integer;
a:array[0..9,0..9] of longint;
begin
readln(n,m);
for i:=0 to n-1 do
  a[i,0]:=1;
for j:=1 to m-1 do
  a[0,j]:=1;
for i:=1 to n-1 do
 for j:=1 to m-1 do
  a[i,j]:=a[i-1,j]+a[i,j-1];
writeln(a[n-1,m-1]);
end.
	#include<iostream>
	using namespace std;
	int main() {
	int n,m,i,j;
	cin >> n >> m;
	long a[n][m];
	for(i=0;i<n;i++)
	  a[i][0]=1;
	for(j=1;j<m;j++)
	  a[0][j]=1;
	for(i=1;i<n;i++)
	 for(j=1;j<m;j++)
	  a[i][j]=a[i-1][j]+a[i][j-1];
	cout << a[n-1][m-1];
	return 0;
	}
		a=[]
		n,m=map(int,input().split())
		for i in range(n):
		  a.append([1] * m)
		for i in range(1,n):
		  for j in range(1,m):
		  a[i][j]=a[i-1][j]+a[i][j-1]
		print(a[n-1][m-1])

Пример 7. Задача №2965. Максимальная стоимость маршрута

В левом верхнем углу прямоугольной таблицы размером N×M находится черепашка. В каждой клетке таблицы записано некоторое число. Черепашка может перемещаться вправо или вниз, при этом маршрут черепашки заканчивается в правом нижнем углу таблицы.

Подсчитаем сумму чисел, записанных в клетках, через которую проползла черепашка (включая начальную и конечную клетку). Найдите наибольшее возможное значение этой суммы.

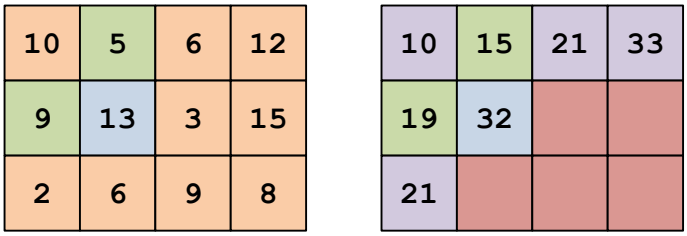
Решение. Разобьем задачу на небольшие подзадачи, которые все же понятно, как решать. Тут прекрасно сработает идея из предыдущей задачи. Например, если таблица состоит из одной строки, то черепашка движется только вправо, и при этом она собирает все, находящееся в строке. Каждый раз, двигаясь в очередную клетку, она прибавляет к набранной сумме число из данной клетки. Аналогично, если таблица состоит из одного столбца, черепашка движется только вниз, и собирает все числа из столбца.



Здесь в оранжевых таблицах выписаны числа, которые находились в таблице изначально, а в голубых – соответствующий выигрыш черепашки. Заведем два массива: в массиве $a: m \times n$ сохраним исходные данные, в $b: m \times n$ – итоговый выигрыш черепашки в данной клетке. Формулы имеют вид:

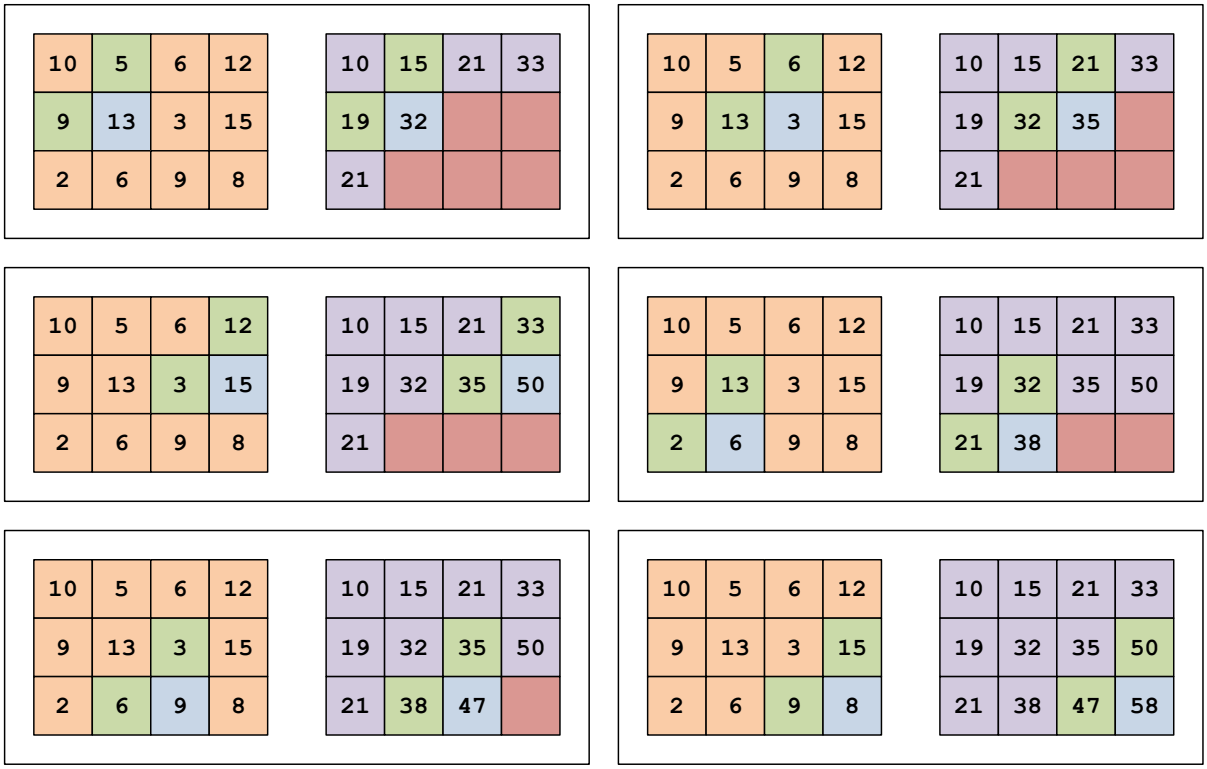
$$b_{0,0} = a_{0,0}$$
$$b_{0,j} = b_{0,j-1} + a_{0,j}, j = 1, \dots, m-1$$
$$b_{i,0} = b_{i-1,0} + a_{i,0}, i = 1, \dots, n-1$$

Рассмотрим более сложную таблицу. В клетку, выделенную голубым цветом, можно попасть из клеток слева и сверху (выделены зеленым). Если мы знаем выигрыш в зеленых клетках, мы получим выигрыш в голубой клетке как максимум выигрышей в зеленых клетках плюс число в данной клетке. На картинке слева показана исходная таблица, справа – таблица выигрыша черепашки.



Таким образом, мы можем заполнять правую таблицу слева направо и сверху вниз по формулам

$$b_{i,j} = \max\{b_{i-1,j}, b_{i,j-1}\} + a_{i,j}, \qquad i = 1, \dots, n-1, j = 1, \dots, m-1.$$



Можно также восстановить путь черепашки, от конца к началу. Путь восстанавливается по принципу – мы пришли в данную клетку из соседней клетки (слева или сверху) с максимальным значением. В клетку 58 мы пришли из 50, в 50 − из 35, в 35 – из 32, и т.д.

10	15	21	33
19	32	35	50
21	38	47	58

Замечание. Жадные алгоритмы тут не работают. Например, если переходить в клетку с максимальным значением, мы найдем другой путь, так как из клетки с ценой 13 (в начальной матрице) пойдем вниз, а не влево.

ЕГЭ, задание 18. Робот-сборщик монет.

Пример 7. Задание 18. № 27415.
Квадрат разлинован на $N \times N$ клеток ($1 < N < 17$). Исполнитель Робот может перемещаться по клеткам, выполняя за одно перемещение одну из двух команд: вправо или вниз. По команде вправо Робот перемещается в соседнюю правую клетку, по команде вниз — в соседнюю нижнюю. При попытке выхода за границу квадрата Робот разрушается. Перед каждым запуском Робота в каждой клетке квадрата лежит монета достоинством от 1 до 100. Посетив клетку, Робот забирает монету с собой; это также относится к начальной и конечной клетке маршрута Робота.

Задание 18

Откройте файл. Определите максимальную и минимальную денежную сумму, которую может собрать Робот, пройдя из левой верхней клетки в правую нижнюю. В ответ запишите два числа друг за другом без разделительных знаков — сначала максимальную сумму, затем минимальную.
Исходные данные представляют собой электронную таблицу размером $N \times N$, каждая ячейка которой соответствует клетке квадрата.
Пример входных данных:

1	8	8	4
10	1	1	3
1	3	12	2
2	3	5	6

Для указанных входных данных ответом должна быть пара чисел 41 и 22.

Решение. Задача аналогична предыдущей. Решение можно получить в Excel. Задача решена в отдельном файле.