

ИНФОРМАТИКА

9

класс

ДИНАМИЧЕСКОЕ ПРОГРАММИРОВАНИЕ

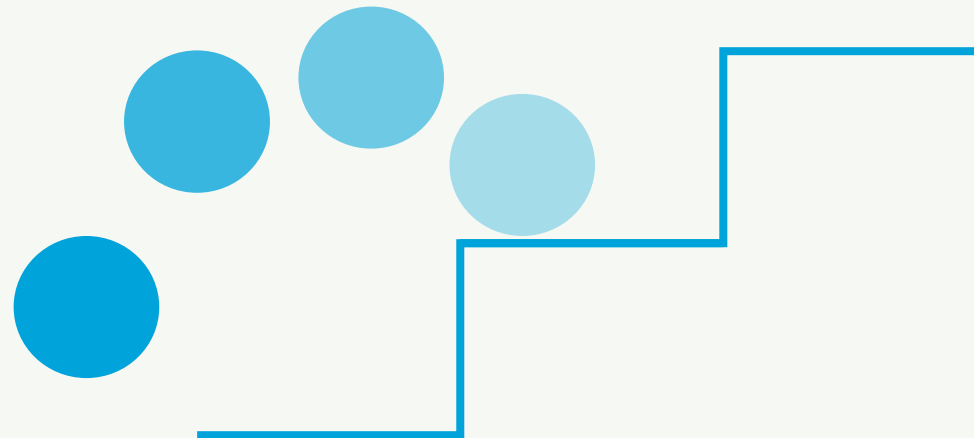
РАЗРАБОТКА АЛГОРИТМОВ И ПРОГРАММ

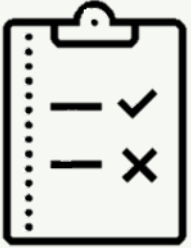
КЛЮЧЕВЫЕ СЛОВА

- ◆ динамическое программирование
- ◆ мемоизация

ДИНАМИЧЕСКОЕ ПРОГРАММИРОВАНИЕ

Динамическое программирование — это метод решения задач, состоящий в том, что исходная задача разбивается на более мелкие подзадачи, они — на ещё меньшие и т. д., пока не получатся простейшие задачи, решение которых уже известно. Решения простейших задач используются при решении задач более крупных и т. д., пока не будет получено решение исходной задачи.





ПРИМЕР 1

Вспомним известную вам задачу нахождения числа под номером n из последовательности чисел Фибоначчи. Напомним, что последовательность Фибоначчи — это последовательность чисел, в которой каждое следующее число, начиная с третьего, является суммой двух предыдущих:

1, 1, 2, 3, 5, 8, 13, 21, ...

Итак, чтобы найти число под номером $n > 2$, сначала надо найти числа под номерами $n - 1$ и $n - 2$:

$$f(n) = f(n - 1) + f(n - 2).$$

Вызовите следующую функцию при $n = 40$:

```
def fib(n):  
    if n == 1 or n == 2:  
        return 1  
    else:  
        return fib(n - 1) + fib(n - 2)
```

Обратите внимание: вычисления будут проходить достаточно долго.



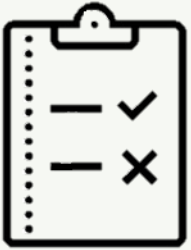
ВЫЧИСЛЕНИЕ ЗНАЧЕНИЙ

Рекурсия вычисляется достаточно долго, так как одни и те же значения вычисляются по несколько раз.

- ◆ для вычисления $f(6)$ надо вычислить $f(5)$ и $f(4)$;
- ◆ для вычисления $f(5)$ надо снова вычислить $f(4)$ и ещё $f(3)$;
- ◆ для вычисления $f(4)$ надо повторно вычислить $f(3)$ и ещё $f(2)$.

Чтобы не вычислять одни и те же значения многократно, часто используется **мемоизация** — сохранение результатов промежуточных вычислений. В случае если какое-то значение уже вычислялось, оно не вычисляется повторно, а используется ранее вычисленный результат.





Дополните следующую программу строкой `print(fib(k))` и вычислите с её помощью 40-й, 100-й, 400-й члены в последовательности Фибоначчи. Обратите внимание на скорость вычислений.

Создаётся массив из 1000 элементов,
заполненный нулями

① `F = [0] * 1000`

② `def fib(n):`

③ `if n == 1 or n == 2:`

④ `return 1`

⑤ `elif F[n] != 0`

⑥ `return F[n]`

⑦ `else:`

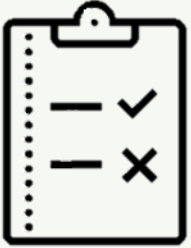
⑧ `F[n] = fib(n-1) + fib(n-2)`

⑨ `return F[n]`

Если элемент массива F, соответствующий любому n, отличному от 1 и 2, не равен нулю, то это значит, что он уже был вычислен ранее и его не надо вычислять повторно, а можно взять значение из массива

В противном случае мы проводим необходимые вычисления и возвращаем полученный результат

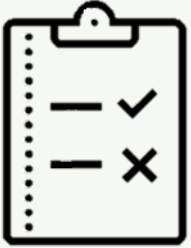




Поставленную задачу можно решить иначе:

```
n = int(input())
F = [0] * (n + 1)
F[1] = 1
F[2] = 1
for i in range(3, n + 1):
    F[i] = F[i-1] + F[i-2]
print(F[i])
```





ПРИМЕР 2

Алгоритм вычисления значения функции $F(n)$, где n — натуральное число, задан следующими соотношениями:

$$F(n)=1 \text{ при } n \leq 2;$$

$$F(n)=F(n-1)+3 \cdot F(n-2) \text{ при } n > 2.$$

Требуется выяснить, чему равно значение функции $F(7)$. По условию, $F(1) = F(2) = 1$.

$$F(3) = F(2) + 3 \cdot F(1) = 1 + 3 \cdot 1 = 4.$$

$$F(4) = F(3) + 3 \cdot F(2) = 4 + 3 \cdot 1 = 7.$$

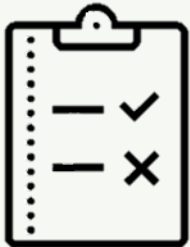
$$F(5) = F(4) + 3 \cdot F(3) = 7 + 3 \cdot 4 = 19.$$

$$F(6) = F(5) + 3 \cdot F(4) = 19 + 3 \cdot 7 = 40.$$

$$F(7) = F(6) + 3 \cdot F(5) = 40 + 3 \cdot 19 = 97$$

Подобные вычисления можно проводить в уме, а их результаты фиксировать в таблице:

n	1	2	3	4	5	6	7
$F(n)$	1	1	4	7	19	40	97



ПРИМЕР 3

На вершине лестницы, содержащей N ступенек, находится мячик, который начинает прыгать по ступенькам вниз, к основанию. Мячик может прыгнуть на предыдущую ступеньку, на ступеньку через одну или через две. (Например, если мячик лежит на 8-й ступеньке, то он может переместиться на 7-ю, 6-ю или 5-ю.) Требуется определить количество всевозможных «маршрутов» мячика с вершины на землю (земля — это «нулевая» ступенька).

Пусть мячик находится на некоторой ступеньке с номером i .

Тогда он может спрыгнуть на ступеньки с номерами $i-1$, $i-2$ и $i-3$. Если мы введём функцию $F(i)$, которая определяет количество маршрутов со ступеньки i до земли, то

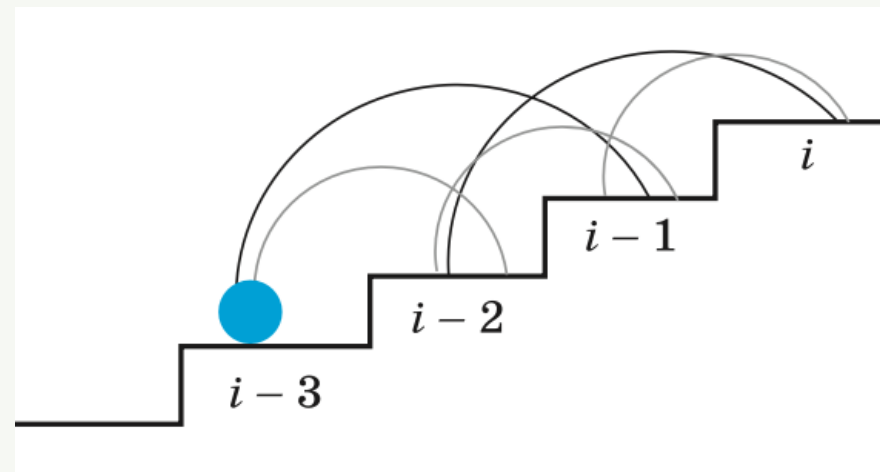
$$F(i) = F(i-1) + F(i-2) + F(i-3).$$

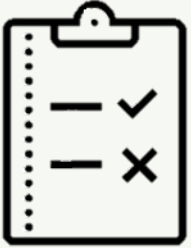
Вычислим вручную начальные значения:

$$F(1) = 1,$$

$$F(2) = 2 \text{ (очевидно),}$$

$$F(3) = 4 \text{ (3-2-1-0, 3-2-0, 3-1-0, 3-0).}$$





ПРИМЕР 3

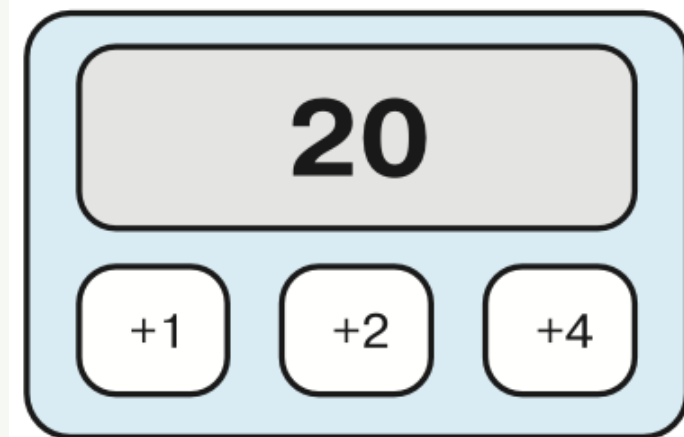
Программа, решающая поставленную задачу, имеет вид:

```
N = int(input())
F = [0] * 1000
def trek (N):
    if N == 1:
        return 1
    elif N == 2:
        return 2
    elif N == 3:
        return 4
    elif F[N] != 0:
        return F[N]
    else:
        F[N] = trek(N-1) + trek(N-2) + trek(N-3)
        return F[N]
print(trek(N))
```





ПРИМЕР 4

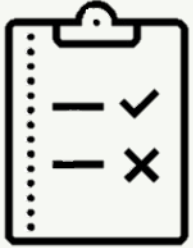


Исполнитель Плюс имеет следующую систему команд:

- 1) прибавь 1;
- 2) прибавь 2;
- 3) прибавь 4.

С помощью первой из них исполнитель увеличивает число на экране на 1, с помощью второй — на 2, с помощью третьей — на 4. Программа для исполнителя Плюс — это последовательность команд. Выясним, сколько разных программ, преобразующих число 20 в число 30, можно составить для этого исполнителя.





ПРИМЕР 4

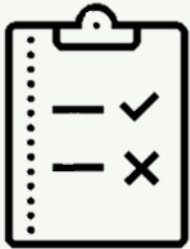
Количество программ, с помощью которых можно получить некоторое число n , будем рассматривать как функцию $K(n)$.

Число, меньшее 20, при заданных начальных условиях и системе команд исполнителя Плюс получить невозможно. Следовательно, при $n < 20$ $K(n) = 0$.

Для начального числа 20 количество программ равно 1: существует только одна пустая программа, не содержащая ни одной команды. Можем записать: $K(n) = 1$ при $n = 20$.

Любое число $n > 20$ может быть получено из чисел $n - 1$, $n - 2$ и $n - 4$ одной командой, входящей в систему команд исполнителя — прибавь 1, прибавь 2 и прибавь 4 соответственно. При этом каждая программа получения из исходного числа чисел $n - 1$, $n - 2$ и $n - 4$ удлинится на одну команду и будет приводить к числу n . Следовательно, $K(n) = K(n - 1) + K(n - 2) + K(n - 4)$.





ПРИМЕР 4

Запишем все соотношения, определяющие функцию

$$K(n): K(n)=0 \text{ при } n < 20;$$

$$K(n)=1 \text{ при } n=20;$$

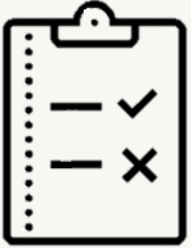
$$K(n)=K(n-1)+K(n-2)+K(n-4) \text{ при } n > 20.$$

Заполним по этой формуле таблицу для всех значений n от 20 до 30:

n	20	21	22	23	24	25	26	27	28	29	30
$K(n)$	1	1	2	3	6	10	18	31	55	96	169

Итак, существует 169 различных программ, с помощью которых исполнитель Плюс может преобразовать число 20 в 30.





ПРИМЕР 5

Пусть исполнитель Плюс должен преобразовать число 0 в число 15, причём за получение каждого из чисел от 1 до 15 ему будет начислено некоторое количество баллов от 1 до 3. Баллы за получение числа i задаются значением `Price[i]` массива `Price`.

Необходимо найти максимальное количество баллов, которое может набрать Плюс, преобразовав исходное число 0 в число-результат 15.

Создадим одномерный массив `Price` из 16 элементов и заполним его случайными числами от 1 до 3:

```
N = 16
Price = [0] * N
from random import randint
for i in range(1,N):
    Price[i] = randint(1, 3)
print(Price)
```





ПРИМЕР 5

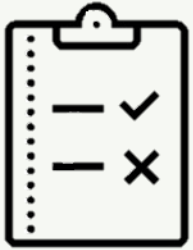
Максимальное количество баллов, которые можно собрать в процессе получения числа i , будем сохранять в элементе $B[i]$ массива B .

Так как число i получается из чисел $i - 1$ (командой прибавь 1), $i - 2$ (командой прибавь 2), $i - 4$ (командой прибавь 4), то значение $B[i]$ можно вычислить так:

$$B[i] = \max(B[i-1], B[i-2], B[i-4]) + \text{Price}[i]$$

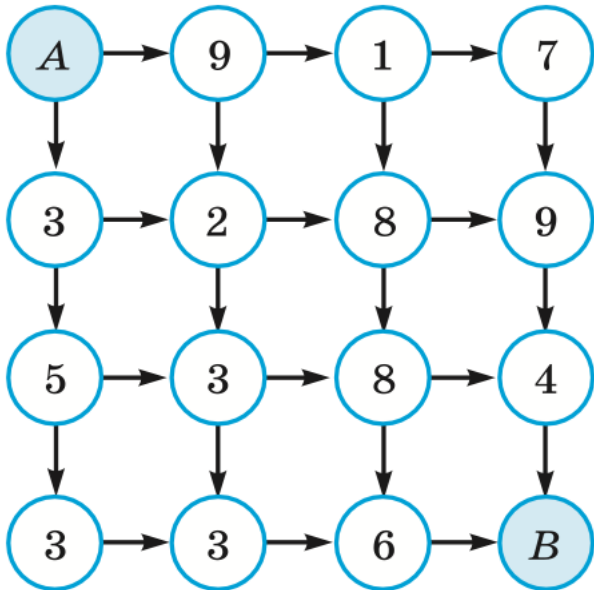
Соответствующий фрагмент программного кода имеет вид:

```
B = [0]*N
B[1] = Price[1]
B[2] = max(B[1], Price[2]) + Price[2]
B[3] = B[2] + Price[3]
for i in range(4, N):
    B[i] = max(B[i-1], B[i-2], B[i-4]) + Price[i]
print(B[15])
```

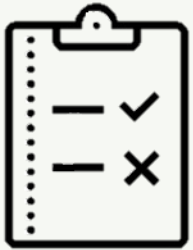


ПРИМЕР 6

Предположим, персонажу некоторой игры необходимо пройти по лабиринту из пункта *A* в пункт *B*, набрав при этом как можно меньше штрафных баллов, количество которых указано в клетках лабиринта, причём перемещаться можно только вниз или вправо.



0	9	10	17
3	5	13	22
8	8	16	20
11	11	17	17



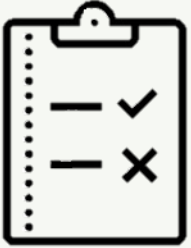
ПРИМЕР 6

```
① A = [[0, 9, 1, 7], [3, 2, 8, 9], [5, 3, 8, 4], [3, 3, 6, 0]]
② for i in range(1, 4):
③     A[i][0] += A[i-1][0]
④     for j in range(1, 4):
⑤         A[0][j] += A[0][j-1]
⑥     for i in range(1, 4):
⑦         for j in range(1, 4):
⑧             A[i][j] += min(A[i][j-1], A[i-1][j])
⑨ print(A[i][j])
```

Динамическое программирование — это метод решения задач, состоящий в том, что исходная задача разбивается на более мелкие подзадачи, они — на ещё меньшие и т. д., пока не получатся простейшие задачи, решение которых уже известно. Решения простейших задач используются при решении задач более крупных и т. д., пока не будет получено решение исходной задачи. Делается это, как правило, с помощью рекурсии.

При реализации рекурсивных алгоритмов одни и те же значения вычисляются по несколько раз. Такая организация вычислений нерациональна. Чтобы не вычислять одни и те же значения многократно, часто используется мемоизация — сохранение результатов промежуточных вычислений. В случае если какое-то значение уже вычислялось, оно не вычисляется повторно, а используется ранее вычисленный результат.

Вместо рекурсивных алгоритмов можно использовать циклические алгоритмы.



ВОПРОСЫ И ЗАДАНИЯ

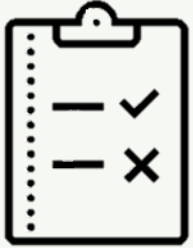
Алгоритм вычисления значения функции $F(n)$, где n — натуральное число, задан следующими соотношениями:

$$F(n) = 2 \text{ при } n \leq 2;$$

$$F(n) = F(n - 2) + F(n - 1) + F(n // 3) \text{ при } n > 2.$$

Чему равно значение функции $F(10)$?





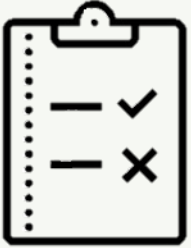
ВОПРОСЫ И ЗАДАНИЯ

На числовой прямой сидит Кузнечик, который может прыгать вправо на одну или на две единицы. Первоначально Кузнечик находится в точке с координатой 0. Определите количество различных маршрутов Кузнечика, приводящих его в точку с координатой 10.

Решите эту задачу двумя способами:

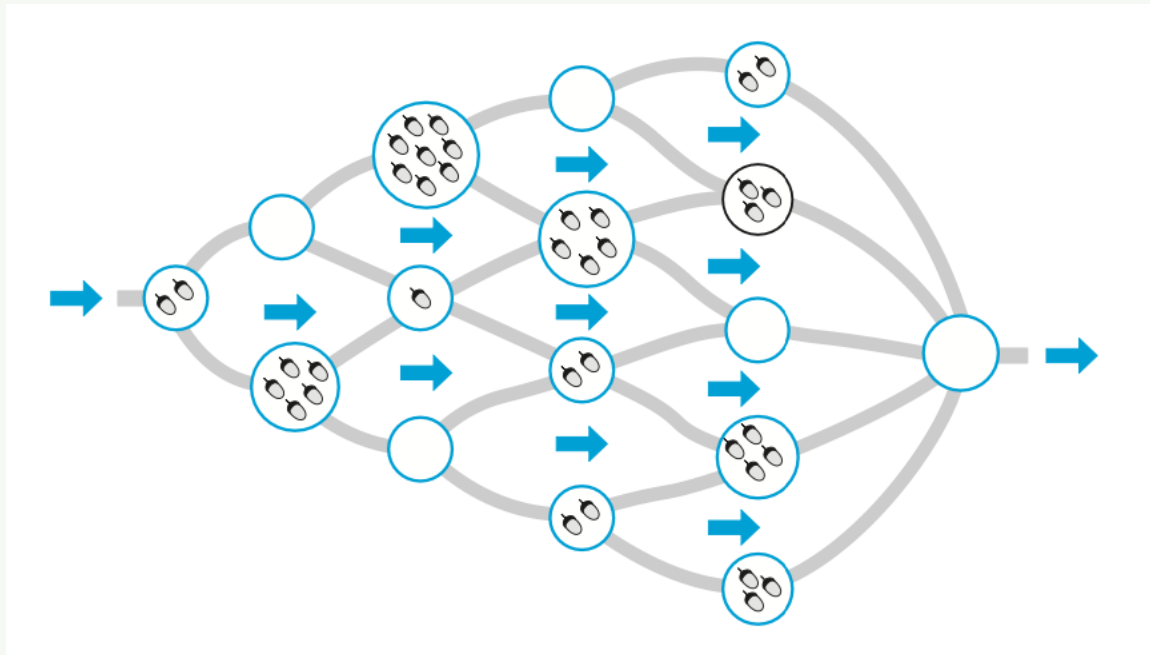
- 1) выполнив необходимые вычисления вручную;
- 2) написав программу и выполнив её на компьютере.

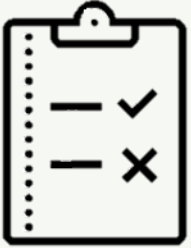




ВОПРОСЫ И ЗАДАНИЯ

В материалах международного конкурса по информатике «Бобёр» есть такая задача, предложенная разработчиками из Нидерландов. Бобёр Билли любит жёлуди. Он хочет поплыть по течению и собрать все жёлуди на островах, мимо которых будет проплывать. Какое максимальное количество желудей он сможет собрать?





ВОПРОСЫ И ЗАДАНИЯ

У исполнителя Калькулятор две команды, которым присвоены номера:

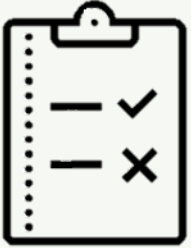
1. прибавь 2
2. умножь на 2

Сколько есть программ, которые число 1 преобразуют в число 24?

Решите эту задачу двумя способами:

- 1) выполнив необходимые вычисления вручную;
- 2) написав программу и выполнив её на компьютере.





ВОПРОСЫ И ЗАДАНИЯ

У исполнителя Калькулятор три команды, которым присвоены номера:

1. прибавь 2
2. прибавь 3
3. прибавь 5

Сколько существует программ, которые число 20 преобразуют в число 35?
Решите эту задачу двумя способами:

- 1) выполнив необходимые вычисления вручную;
- 2) написав программу и выполнив её на компьютере.





ВОПРОСЫ И ЗАДАНИЯ

Рассмотрим задачу на шахматной доске. В левом верхнем углу находится король. Король может перемещаться только вправо, вниз или по диагонали вправо-вниз на одну клетку. Необходимо определить количество различных маршрутов короля, приводящих его в правый нижний угол.

