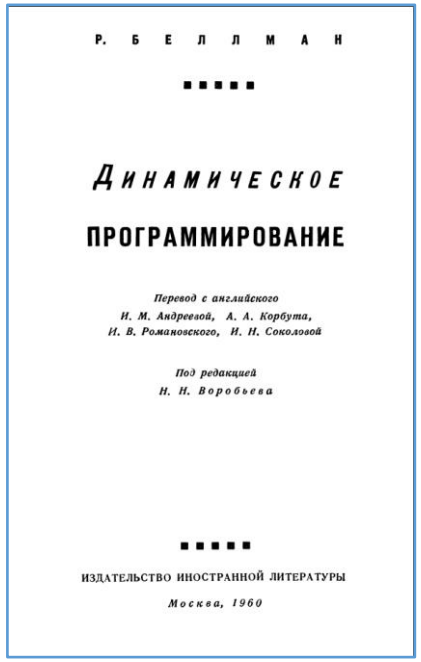
# Динамическое программирование в задачах обработки последовательностей ЕГЭ по информатике

1 часть

Старший методист ЦИТ
ГАУ ДПО ЯО ИРО
С.Ю. Белянчева

Ричард Эрнест Беллман




https://djvu.online/file/Y6aNgwIw9NxsP?ysclid=lpqndt7e2637577557

**Цель книги** – дать введение в математическую теорию многошаговых процессов решения.

**Процесс решения** – это процесс выбора преобразований, которым можно подвергнуть некоторую систему.

**Область применения**: оптимальное управление запасами, анализ баланса затрат и выпуска целого комплекса взаимозависимых отраслей, составление графика обслуживания пациентов или самолетов, задачи о капиталовложениях

# Динамическое программирование

https://stepik.org/course/104157/promo

**Цель данного курса** — научить слушателей решать задачи с помощью динамического программирования или определять, что решение данным методом затруднено.


Бесплатно

В курс входят

27 уроков

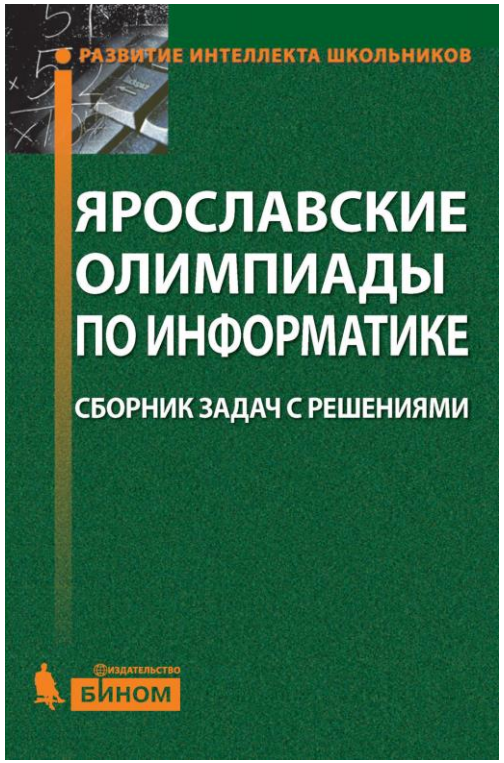
13 тестов

9 интерактивных задач

# Анализ литературы по теме «Динамическое программирование»

Ярославские олимпиады по информатике. Сборник задач с решениями. Волчёнков С.Г.
https://vk.com/doc191450968_580247024?hash=Xm2ijUYFeGWJAUpeX3ta0WVnaEiWRjaBPwh5aZrLovk&dl=MZzXElHgPNZk39wTCZw95UhLCla0vhlCvuhd2OZ07P8

Книга содержит задачи, предлагавшиеся на различных турах ярославских олимпиад по информатике с 1987 по 2009 г. Ко всем задачам приведены решения или краткие указания к решению.



# Анализ литературы по теме «Динамическое программирование»

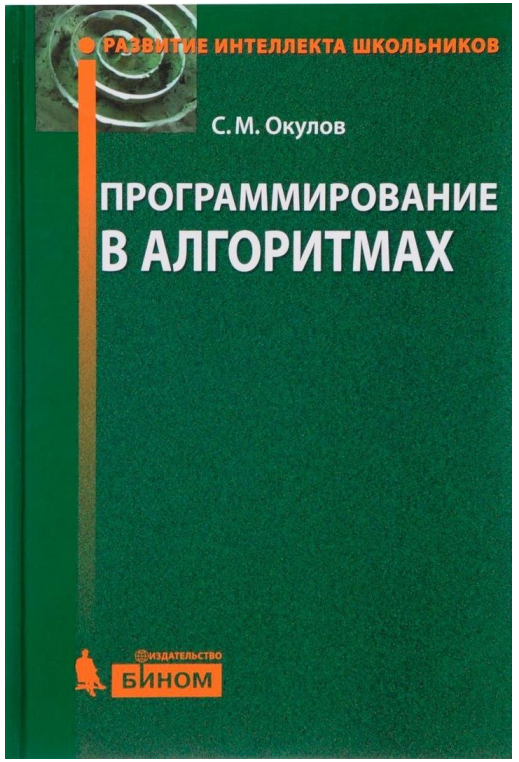
Учебное пособие В.М. Котова «Метод динамического программирования»
http://www.gmo-inf-ug.narod.ru/DswMedia/kotov.rar

1) Понятие задачи и подзадачи. Сведения задачи к подзадачам. Понятие рекуррентного соотношения. Правильные рекуррентные соотношения.
2) Использование таблиц для запоминания решения подзадач. Метод динамического программирования. Вычисление значений элементов одномерной таблицы.
3) Вычисление значений элементов двумерной таблицы. Восстановление структуры решения.

# Анализ литературы по теме «Динамическое программирование»

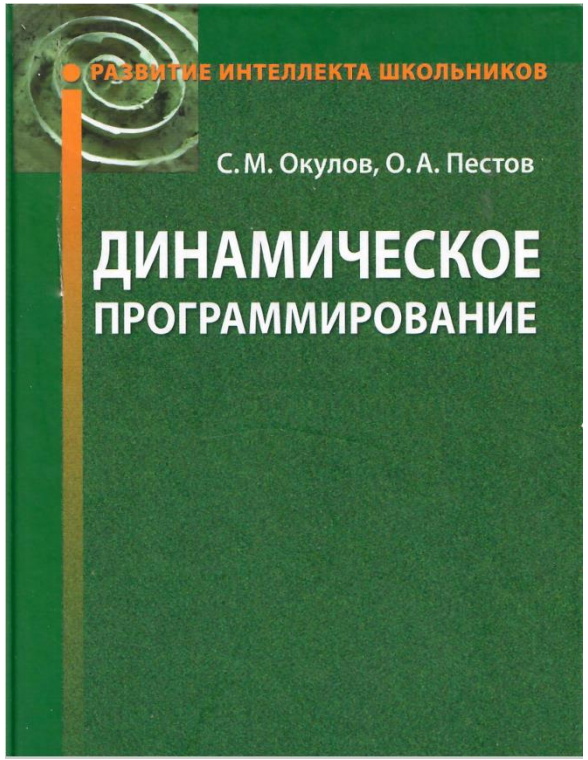
Программирование в алгоритмах. С. М. Окулов. https://vk.com/wall-173323234_1774?ysclid=lpv8lvvuh2430111225

В книге рассмотрены такие вопросы, как комбинаторные алгоритмы, перебор, алгоритмы на графах, алгоритмы вычислительной геометрии. В пособии также приводятся олимпиадные задачи по программированию с комментариями и практические рекомендации по тестированию программ.



# Анализ литературы по теме «Динамическое программирование»

Динамическое программирование. Окулов С.М., Пестов О.А.
https://vk.com/wall-220531567_53?ysclid=lpv6qmz1c3539165715

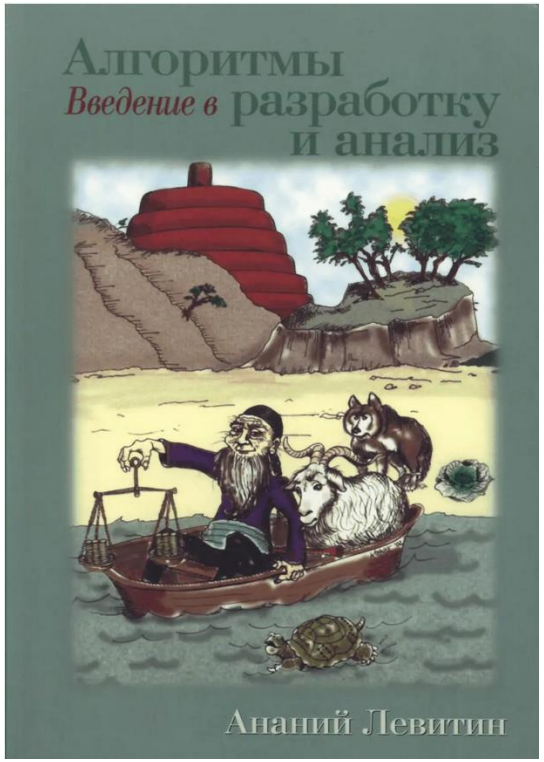



Глава 1. Простые задачи
Глава 2. Основной принцип и метод реализации на основе рекуррентных соотношений
Глава 3. Типы задач по динамическому программированию

# Анализ литературы по теме «Динамическое программирование»

«Алгоритмы. Введение в разработку и анализ» А. Левитин
https://djvu.online/file/9p2iXACf6NoIg?ysclid=lpqst1tk4x212761939

Глава 1. Понятие алгоритма, типы задач, базовые структуры данных
Глава 2. Основы анализа эффективности алгоритма
Глава 3. Метод «грубой силы»: сортировка, поиск, перебор
Глава 4. Метод декомпозиции
Глава 5. Метод уменьшения размера задачи
Глава 6. Метод преобразования
Глава 7. Пространственно-временной компромисс
Глава 8. Динамическое программирование
Глава 9. Жадные методы …

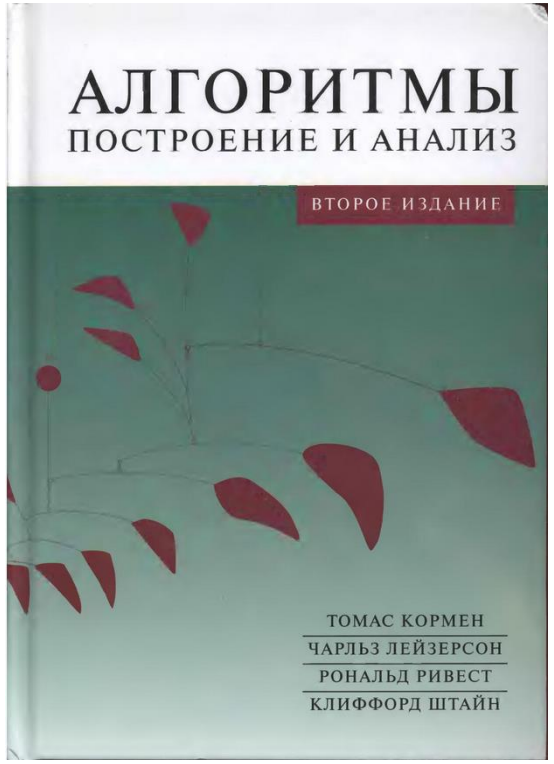


# Анализ литературы по теме «Динамическое программирование»

«Алгоритмы: построение и анализ» Томас Кормен
https://djvu.online/file/IWpjVy9EeMiZ3?ysclid=lpqthve4ds517992142
(https://vk.com/doc191450968_561608466?hash=6ZfSzXBV4R2z0lCDk2hd0Jfbm9uCT14JeEooJZ9Ss9X&dl=TPRxLokGe8hKecJyGVdQWNd6CaV1FDHhC6BKanr1s4P )

Этапы процесса разработки алгоритмов динамического программирования

1) Описание структуры оптимального решения
2) Рекурсивное определение значения, соответствующего оптимальному решению
3) Вычисление значения, соответствующего оптимальному решению, с помощью метода восходящего анализа.
4) Составление оптимального решения на основе информации, полученной на предыдущих этапах.



# Основные проблемы при решении задач методом динамического программирования

- Недостаточный опыт обучающихся в решении задач по динамическому программированию
- Незнание «подходов» к решению задач
- Стремление обучающихся решать задачи по динамическому программированию «в лоб»
- Сложности приведения задачи к математической формализации

# Примерный элективный курс

Цель: ознакомить учащихся с методом динамического программирования.

Курс проводится в виде мини-лекций и практических занятий.

Задачи сгруппированы по блокам в зависимости от тематики и сложности. Наполнение материала осуществляется и регулируется учителем с учетом способностей школьников, профиля класса.

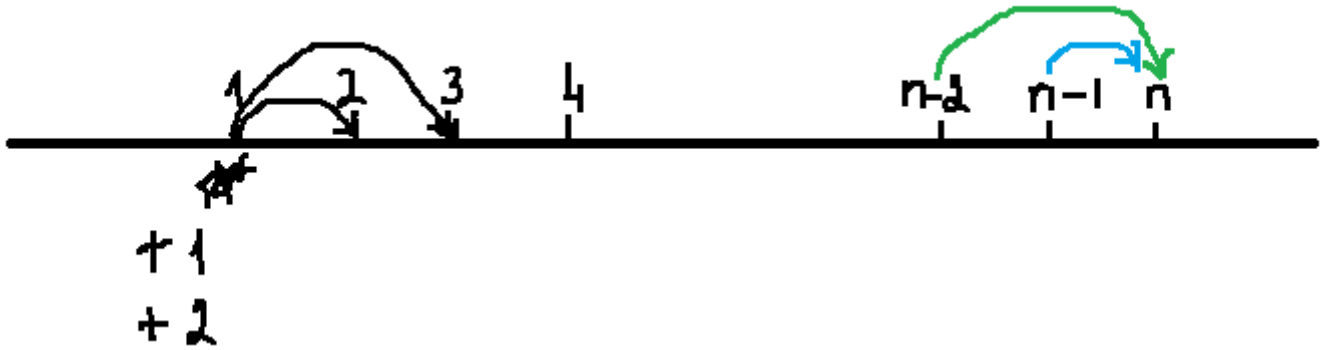
Каждый урок начинается с изложения теоретических сведений. После этого показываются способы решения задач.

Каждый урок заканчивается выдачей домашнего задания, содержащего задачи по рассматриваемой теме и указания к их решению.

# Динамическое программирование

Задача о кузнечике

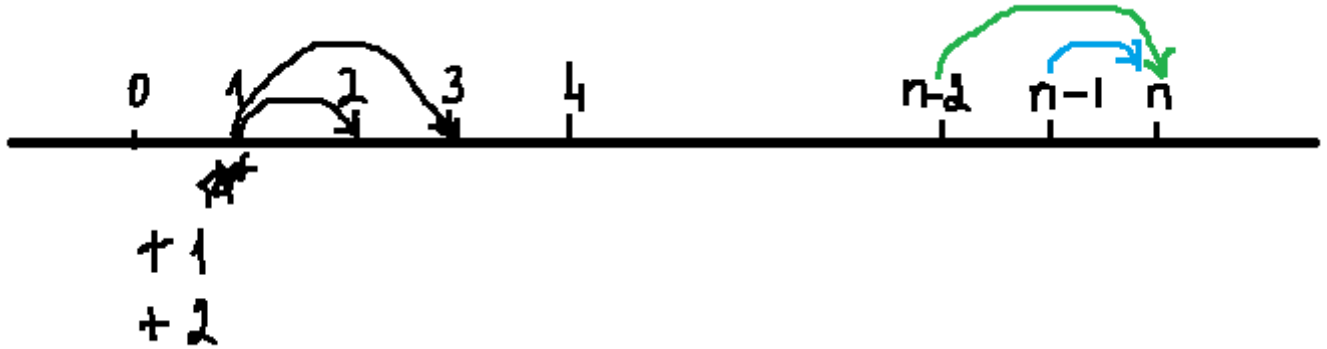
На числовой прямой сидит кузнечик, который может прыгать вправо на одну или на две единицы. Первоначально кузнечик находится в точке с координатой 1. Определите количество различных маршрутов кузнечика, приводящих его в точку с координатой n.




Количество способов попасть в точку n: $K_n = K_{n-1} + K_{n-2}$

Крайний случай - количество траекторий из 1 в 1: $K_1 = 1$.

Добавим точку 0 ($K_0 = 0$)




Начиная с точки 2, можно вычислять.

Рекуррентное соотношение $F(n) = F(n-2) + F(n-1)$, верное для всех $n \geq 2$ .

## Пример на языке Python

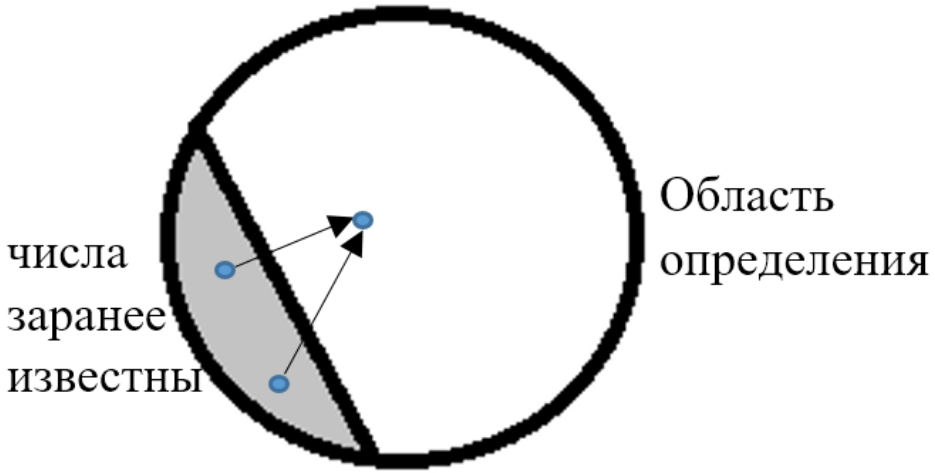
```
def f(n):
    if n<=1:
        return n
    else:
        return f(n-1) + f(n-2)

n = int(input())
print(f(n))
```

## Пример на языке Pascal

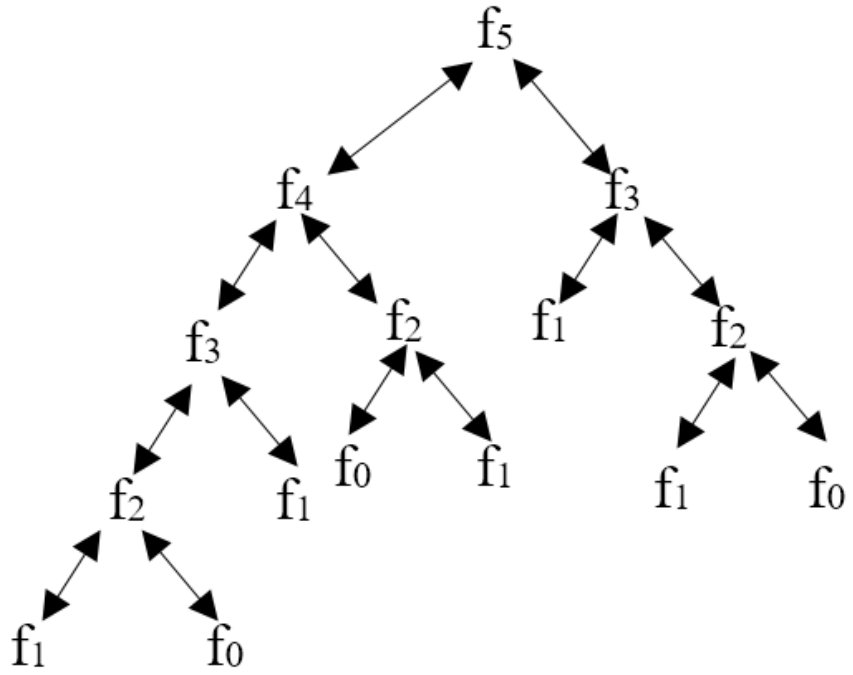
```
function f(n: longint): longint;
begin
  if n <= 1 then
    f := n
  else
    f := f(n-1) + f(n-2)
end;

var n: longint;

begin
  readln(n);
  writeln(f(n))
end.
```

**Числа Фибоначчи**: это элементы числовой последовательности 0, 1, 1, 2, 3, 5, 8, 13, 21, …, в которой каждое последующее число равно сумме двух предыдущих.

$f_n = f_{n-1} + f_{n-2}$ (область определения – неотрицательные числа)
$f_0 = 0$
$f_1 = 1$




Задача:
По заданному числу n вычислить $F_n$

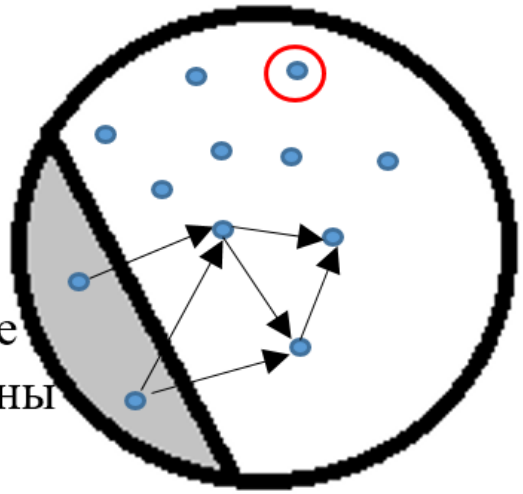



Для подсчета N-го числа Фибоначчи необходимо 2 раза посчитать (N-2)-е число, и это занимает в два раза больше времени, а значит это хотя бы $2^{N/2}$ действий.
Сложность $O(2^n)$

Нужен cash для хранения промежуточных вычислений.

```
def f(n):
    if cash[n] == -1:
        if n <= 1:
            result = n
        else:
            result = f(n-1) + f(n-2)
        cash[n] = result
    return cash[n]

n = int(input())
cash = [-1]*10000
print(f(n))
```

Алгоритм «сверху»



числа заранее известны

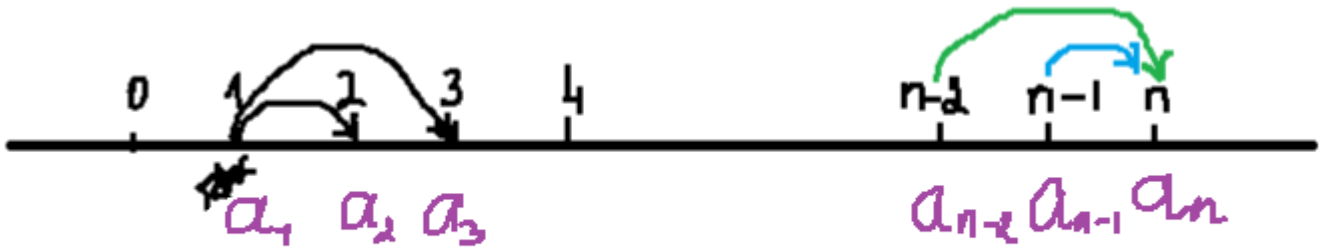
Область определения

```
def f(n):
    F = [0, 1] + [-1]*(n-1)
    for i in range(2, n+1):
        F[i] = F[i-1] + F[i-2]
    return F[n]

n = int(input())
print(f(n))
```

Алгоритм «снизу»

## Задача о кузнечике 1

На числовой прямой сидит кузнечик, который может прыгать вправо на одну или на две единицы.  В каждой точке кузнечик может получить награду или штраф (положительные и отрицательные числа). Первоначально кузнечик находится в точке с координатой 1. Определите наибольшую награду, которую получит кузнечик, перемещаясь в точку с координатой n.




$f_3 = a_3 + \max(a_1, a_2)$ $f_n = a_n + \max(a_{n-1}, a_{n-2})$

**Задача о кузнечике 2**

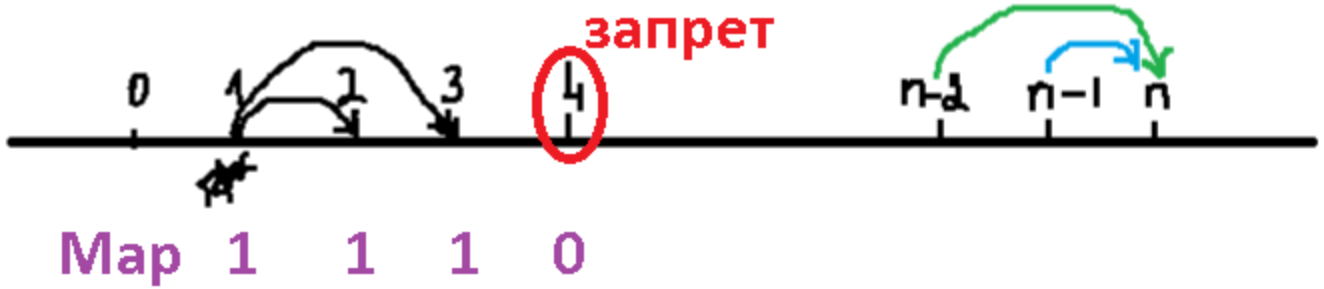
Пусть некоторые точки являются «запретными» для кузнечика, он не может прыгать в эти точки.

«Карта» запрещенных точек задается при помощи списка Map:

если Map[i] == 0 (для языка Pascal — массива Map), то в точку номер i кузнечик не может прыгать,

если Map[i] == 1, то данная точка является разрешенной для кузнечика.

Необходимо найти количество маршрутов в точку n.




Определение рекуррентного соотношения:

если Map[i] == 0, то F[i] = 0, то есть если точка — «запрещенная», то количество способов попасть в эту точку равно 0, так как нет ни одного допустимого маршрута, заканчивающегося в этой точке;

если Map[i] == 1, то значение F[i] вычисляется по тем же рекуррентным соотношениям, что и ранее.

**Максимальная сумма подпоследовательности.**

Дана последовательность, требуется найти максимальную сумму подпоследовательности.

A = [5, 2, 3, -7, 6, 8, -15, 2, -9, 20, 8, -24, 5, 2, -1, 2, 7]

Переборное решение, $O(n^2)$ :

```
m = -100000
for a in range(0,n):
   for b in range(a,n):
      s = sum(A[a:b])
      if s > m:
         m = s
```

Можно составить алгоритм за один проход?

A = [5, 2, 3, -7, 6, 8, -15, 2, -9, 20, 8, -24, 5, 2, -1, 2, 7]

Начнем считать суммы подпоследовательностей A[:i]

Сумма подпос-ти от начала до a:

```
Sa = [0] + [None]*len(A)
for i in range(1,n+1):
    Sa[i] = Sa[i] + A[i-1]
```

```
m = -100000
for a in range(0,n):
    for b in range(a,n):
        s = sum(A[a:b])
        if s > m:
            m = s
```

$$\sum_{i=a}^{b-1} A_i = \sum_{i=0}^{b-1} A_i - \sum_{i=0}^{a-1} A_i$$

max min

В процессе обхода последовательности будем кэшировать оптимальную вычитаемую сумму слева.

```
Sa_min = [0] + [None]*len(A)
for i in range(1,n+1):
    Sa_min[i] = min(Sa_min[i-1],Sa[i])
```

A = [5, 2, 3, -7, 6, 8, -15, 2, -9, 20, 8, -24, 5, 2, -1, 2, 7]

```
Sa = [0] + [None]*len(A)
for i in range(1,n+1):
   Sa[i] = Sa[i] + A[i-1]
Sa_min = [0] + [None]*len(A)
for i in range(1,n+1):
   Sa_min[i] = min(Sa_min[i-1],Sa[i])

Максимальная сумма подпос-ти:
ms = 0

Проход по всем числам:
for i in range(1,n+1):
   tmp = Sa[i]-Sa_min[i]
   ms = max(ms, tmp)
print(ms)
```

Запишем все в один цикл

```
Sa = [0] + [None]*len(A)
for i in range(1,n+1):
   Sa[i] = Sa[i] + A[i-1]
   Sa_min = [0] + [None]*len(A)
   Sa_min[i] = min(Sa_min[i-1] , Sa[i])
   tmp = Sa[i]-Sa_min[i]
   ms = max(ms, tmp)
print(ms)
```

Можно избавиться от массивов?

```
Sa = [0]; Sa_min = 0; ms = 0
for i in range(1,n+1):
    Sa = Sa + A[i-1]
    Sa_min = min(Sa_min , Sa)
    tmp = Sa - Sa_min
    ms = max(ms, tmp)
print(ms)
```

```
S = Smin = ms = 0
for i in range(n):
    S += A[i]
    Smin = min(Smin , S)
    ms = max(ms, S-Smin)
print(ms)
```

| | A = | 5 | 2 | 3 | -7 | 6 | 8 | -15 | 2 | -9 | 20 | 8 | -24 | 5 | 2 | -1 | 2 | 7 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| S | | 0 | 5 | 7 | 10 | 3 | 9 | 17 | 2 | 4 | -5 | 15 | 23 | -1 | 4 | 6 | 5 | 7 | 14 |
| Smin | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | -5 | -5 | -5 | -5 | -5 | -5 | -5 | -5 | -5 |
| S-Smin | | 0 | 5 | 7 | 10 | 3 | 9 | 17 | 2 | 4 | 0 | 20 | 28 | 4 | 9 | 11 | 10 | 12 | 19 |

Подпоследовательность с максимальной суммой [20,8]

**Пути на клеточном поле**

Дана прямоугольная таблица n x m, в клетках которой записаны целые числа. Черепашка находится в левой верхней клетке и ей необходимо попасть в правую нижнюю клетку. За один ход Черепашка может переместиться в соседнюю нижнюю или правую клетку. Требуется найти путь Черепашки с максимальной суммой элементов.

| | | | |
|---|---|---|---|
| 2 | 4 | 7 | 1 |
| 10 | 0 | 1 | 1 |
| 0 | 1 | 8 | 3 |
| 6 | 9 | 1 | 5 |

# Пути на клеточном поле

| | | | |
|---|---|---|---|
| 2 | 4 | 7 | 1 |
| 10 | 0 | 1 | 1 |
| 0 | 1 | 8 | 3 |
| 6 | 9 | 1 | 5 |

Создаем массив a. В ячейке `a[i][j]` - хранится максимальная сумма, которую мы соберем по дороге от [0,0] ячейки в ячейку [i,j].

Ответ будет храниться в ячейке a[n][m]

Заполняем каждую ячейку с помощью проверки сумм:

`a[i][j] = max(a[i-1][j],a[i][j-1])+a[i][j]` – максимальная сумма из предыдущей верхней и левой ячейки + число из текущей ячейки.

# Реализация

```
4 4
2 4 7 1
10 0 1 1
0 1 8 3
6 9 1 5
[[2, 6, 13, 14], [12, 12, 14, 15], [12, 13, 22, 25], [18, 27, 28, 33]]
```

1.py - C:/Users/svbel/AppData/Local/Programs/Python/Python311/1.py (3.11.4)

File Edit Format Run Options Window Help

```
n,m = map(int,input().split())
a = [list(map(int,input().split())) for i in range(n)]
for i in range(n):
    for j in range(m):
        if j==0 and i!=0:
            a[i][j]+=a[i-1][j]
        if i==0 and j!=0:
            a[i][j]+=a[i][j-1]
        if i>0 and j>0:
            a[i][j]=max(a[i-1][j],a[i][j-1])+a[i][j]
print(a)
```

# Динамическое программирование с двумя параметрами.

**Задача о рюкзаке.** (англ. Knapsack problem) — дано N предметов, $n_i$ предмет имеет массу $w_i>0$ и стоимость $p_i>0$. Необходимо выбрать из этих предметов такой набор, чтобы суммарная масса не превосходила заданной величины W (вместимость рюкзака), а суммарная стоимость была максимальна.


Пусто
0$

Одна гитара
1500$

Один ноутбук
2000$

Одна бензопила
3000$

Варианты решения

Задачу о рюкзаке можно решить несколькими способами:

- перебирать все подмножества набора из N предметов. Сложность такого решения $O(2^N)$
- разбить задачу пополам и решать всю задачу через частичный расчет половинок. Сложность решения $O(2^{N/2}N)$
- методом динамического программирования. Сложность — O(NW)

**Метод динамического программирования**

Пусть $A(k,s)$ - максимальная стоимость предметов, которые можно уложить в рюкзак вместимости $s$, если можно использовать только первые $k$ предметов.

$\{n1,n2,\ldots,nk\}$ - набор допустимых предметов для $A(k,s)$

$A(k,0)=0$

$A(0,s)=0$

**Метод динамического программирования**

Найдем $A(k,s)$. Возможны 2 варианта:

1. Если предмет $k$ не попал в рюкзак.

$A(k,s)$ равно максимальной стоимости рюкзака с такой же вместимостью и набором допустимых предметов $\{n1,n2,\ldots,n(k-1)\}$:

$A(k,s) = A(k-1,s)$

2. Если $k$ попал в рюкзак.

$A(k,s)$ равно максимальной стоимости рюкзака, где вес $s$ уменьшаем на вес $k$-ого предмета и набор допустимых предметов $\{n1,n2,\ldots,n(k-1)\}$ плюс стоимость $k$:

$A(k,s) = A(k-1,s-wk) + pk$

## Метод динамического программирования

$$A(k,s) = \begin{cases} A(k-1,\ s) & bk=0 \\ A(k-1,s-wk)+pk & bk=1 \end{cases}$$

То есть: $A(k,s) = \max(A(k-1,\ s),\ A(k-1,\ s-wk) + pk)$

Будем определять, входит ли *ni* предмет в искомый набор.

Начинаем с элемента $A(i,w)$, где $i=N$, $w=W$.
Для этого сравниваем $A(i,w)$ со следующими значениями:

1. Максимальная стоимость рюкзака с такой же вместимостью и набором допустимых предметов $\{n1,n2,\ldots,n(i-1)\}$, то есть $A(i-1,w)$

2. Максимальная стоимость рюкзака с вместимостью на $wi$ меньше и набором допустимых предметов $\{n1,n2,\ldots,n(i-1)\}$ плюс стоимость $pi$, то есть $A(i-1,w-wi)+pi$

Пусто
0$

Одна гитара
150$

Один ноутбук
2000$

Одна бенозопила
3000$

W = 4, N = 3
w1 = 1, p1 = 1500
w2 = 3, p2 = 2000
w3 = 4, p3 = 3000

Сравниваем
A[k-1, s] с
A[k-1, s-wk] + pk
и записываем в
A[k,s]

| Кол-во предметов | Вес | | | |
|---|---|---|---|---|
| | 1 | 2 | 3 | 4 |
| k = 0 | 0 | 0 | 0 | 0 |
| k = 1 | 1500 | 1500 | 1500 | 1500 |
| k = 2 | 1500 | 1500 | 1500 | 3500 |
| k = 3 | 1500 | 1500 | 1500 | 3500 |

# Домашнее задание: написать решение задач

1. Заполнение рюкзака: дано N предметов массой m1,…,mN и стоимостью c1,…,cN соответственно. Ими наполняют рюкзак, который выдерживает вес не более M. Какую наибольшую стоимость могут иметь предметы в рюкзаке?
2. Дано N золотых слитков массой m1,…,mN. Ими наполняют рюкзак, который выдерживает вес не более M. Можно ли набрать вес в точности M?
3. Дан набор гирек массой m1,…,mN. Можно ли их разложить на две чаши весов, чтобы они оказались в равновесии?

Решения присылать на почтовый адрес svbel@iro.yar.ru