# Занятие 18.

## Решение алгоритмических задач.

## План занятия

- повторение
- решение задач по всему блоку

### Сложность алгоритмов

$f(n) = \mathcal{O}(g(n)) \Leftrightarrow$ найдется константа C, что начиная с некоторого n: $f(n) \leq C \cdot g(n)$

$f(n) = \Omega(g(n)) \Leftrightarrow$ найдется константа C, что начиная с некоторого n: $f(n) \geq C \cdot g(n)$

$f(n) = \Theta(g(n)) \Leftrightarrow$ найдутся константы C_1б C_2, что начиная с некоторого n: $C_1 \cdot g(n) \leq f(n$

---

$$2n^2 + 5n = \mathcal{O}(4n^2)$$

при $c = 1, n \geq 3$

$$4n^2 = \mathcal{O}(2n^2 + 5n)$$

при $c = 2, n \geq 1$

$$2n^2 + 5n = \Theta(4n^2)$$

$$5n^2 = \Omega(3n \cdot log(n))$$

## Динамическое программирование

# Двумерные задачи

### Задача

1) Дана таблица cost, где *cost*[*i*][*j*] = стоимости перехода через эту ячейку. Требуется найти наименьший по стоимости путь от левой верхней клетки до правой нижней. Двигаться можно только вниз и вправо.

### Решение

- используем динамическое программирование:
  - строим новую таблицу A
  - в *A*[*i*][*j*] будет храниться наименьший путь от начала до (*i*, *j*)
  - *A*[*i*][*j*] = *min*(*A*[*i* – *j*][*j*], *A*[*i*][*j* – 1]) + *cost*[*i*][*j*]

In [5]:

```
n = 4
cost = [list(map(int, input().split())) for i in range(n)]
```

```
2 3 1 5
4 2 6 3
5 4 9 2
4 10 3 1
```

In [9]:

```
A = [[0]*n for i in range(n)]
A[0][0] = cost[0][0]
for i in range(1, n):
    A[0][i] = A[0][i-1] + cost[0][i]
for i in range(1,n):
    A[i][0] = A[i-1][0] + cost[i][0]
    for j in range(1,n):
        A[i][j] = min(A[i-1][j], A[i][j-1]) + cost[i][j]
print(A[-1][-1])
```

```
17
```

In [10]:

```
for row in cost:
    print(*row)
```

```
2 3 1 5
4 2 6 3
5 4 9 2
4 10 3 1
```

In [11]:

```
for row in A:
```

```
    print(*row)
```

```
2 5 6 11
6 7 12 14
11 11 20 16
15 21 23 17
```

Как восстановить последовательность шагов?

- начинаем двигаться с конца таблицы A
- шагаем в наименьшее из двух чисел слева и сверху
- продолжаем движение до начала и инвертируем список

**Упражнение**

- написать функцию, которая восстанавливает минимальный путь

**Задача**

2) Играют 2 игрока на поле размером $M \times N$. За 1 ход игрок может переставить фишку на неограниченное кол-во клеток вправо/вниз/по диагонали вниз. Игроки ходят по очереди. Проигрывает тот, чей ход начинается в нижней угловой клетке.

Кто выиграет при рациональной игре?

**Решение**

- Заметим, что если $M = N$, то выиграет 1-й игрок, сходив до конца по диагонали. Также при $M = 1$ или $N = 1$ выиграет 1-й
- Далее, пойдем с конца и нарисуем эти выигрышные стратегии буквой "В". Если из клетки нет пути, кроме как в "В", то это буква "П". Продолжаем заполнение, поднимаясь снизу вверх.
- Если в начальной клетке стоит буква "В", то выиграет 1-й игрок, иначе 2-й

---

In [12]:

```
n, m = map(int, input().split())
B = [[0]*m for i in range(n)]
# B[n-1][m-1] = 'L'

def check_win(n,m):
    for i in range(n):
        B[i][m] = 'W'
    for i in range(m):
        B[n][i] = 'W'
    for i in range(min(n,m), 0, -1):
        B[n - i][m - i] = 'W'
```

```
for i in range(n - 1, -1, -1):
    for j in range(m-1, -1, -1):
        if B[i][j] == 0:
            B[i][j] = 'L'
            check_win(i, j)

if B[0][0] == 'W':
    print(1)
else:
    print(0)
```

```
9 15
1
```

In [13]:

```
for row in B:
    print(*row)
```

```
W L W W W W W W W W W W W W W
W W W W W W W W W W L W W W W
W W W W L W W W W W W W W W W
W W W W W W W W W W W L W W W
W W W W W W W L W W W W W W W
W W W W W W W W L W W W W W W
W W W W W W W W W W W W W L W
W W W W W W W W W W W W L W W
W W W W W W W W W W W W W W L
```

## Задача об укладке рюкзака

Имеется $k$ вещей с массами $m_1, \ldots, m_k$ и стоимостями $c_1, \ldots, c_k$ соответственно. Грузоподъемность рюкзака $M$. Требуется наполнить рюкзак так, чтобы ценность была наибольшей

### Решение

- будем использовать ДП!
- строим табличку, где по одной оси мы будем увеличивать грузоподъемность рюкзака $W \in [0, M]$, а по другой рассматривать предметы $m_1, \ldots m_j$, где $j \in [1, k]$
- $a[j][w]$ --- максимальная стоимость для рюкзака весом W для предметов $m_1, \ldots m_j$

---

Рассмотрим j-й предмет:

- если его берем, то $a[j][w] = a[j-1][w - m[j]] + c[j]$

- если не берем $a[j][w] = a[j-1][w]$
- общая формула

$$a[j][w] = max(a[j-1][w-m[j]] + c[j], a[j-1][w])$$

---

In [14]:

```
m = [0, 4, 5, 3, 7, 6]
c = [0, 5, 7, 4, 9, 8]
n = len(m) - 1
M = 16

a = [[0]*(M + 1) for i in range(n + 1)]
```

In [15]:

```
for j in range(1, n +1):
    for w in range(1, M + 1):
        if w - m[j] >= 0:
            a[j][w] = max(a[j - 1][w - m[j]] + c[j], a[j - 1][w])
        else:
            a[j][w] = a[j - 1][w]
print(a[-1][-1])
```

```
21
```

In [18]:

```
a
```

Out[18]:

```
[[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5],
 [0, 0, 0, 0, 5, 7, 7, 7, 7, 12, 12, 12, 12, 12, 12, 12],
 [0, 0, 0, 4, 5, 7, 7, 9, 11, 12, 12, 12, 16, 16, 16, 16],
 [0, 0, 0, 4, 5, 7, 7, 9, 11, 12, 13, 14, 16, 16, 18, 20],
 [0, 0, 0, 4, 5, 7, 8, 9, 11, 12, 13, 15, 16, 17, 19, 20, 21]]
```

Как восстановить предметы из рюкзака?

- начинаем двигаться с конца таблицы A
- шагаем в наименьшее из двух чисел с индексами $[j-1][w-m[j]]$ и $[j-1][w]$
- продолжаем движение до начала

**Упражнение**

- написать функцию, которая восстанавливает предметы из рюкзака

## Линейные задачи

### Задача

- Дан массив неотрицательных чисел. Найти отрезок с фиксированной суммой M

### Решение

- Посчитать сумму на всех подотрезках и сравнить с M. Сложность $\mathcal{O}(N^2)$
- метод двух указателей
  - инициализируем в начале массива
  - двигаем правый, если сумма между указателями меньше M, иначе двигаем левый
  - продолжаем до конца массива, либо пока не получим M

In [34]:

```
x = [8, 1, 2, 10, 2, 7, 9, 2, 5]
M = 15

l = r = 0
cur_sum = 0
while r < len(x):
    if cur_sum == M:
        break
    elif cur_sum < M:
        cur_sum += x[r]
        r += 1
    else:
        cur_sum -= x[l]
        l += 1

if r == len(x):
    while l < len(x):
        if cur_sum == M:
            break
        cur_sum -= x[l]
        l += 1

if cur_sum == M:
    print('Yes')
else:
    print('No')
```

```
Yes
```

### Задача

- Задано множество символов. Необходимо вывести все перестановки этих символов

символов

### Решение

- всего перестановок будет $n!$, где $n$ - количество символов в множестве
- рекурсивный вызов функции f(my_set, strs)

In [41]:

```
myset = {'a', 'd', 'b'}

def permutations(my_set, strs=''):
    if len(my_set) > 0:
        for elem in my_set:
            permutations(my_set - {elem}, strs + str(elem))
    else:
        print(strs)
```

In [42]:

```
permutations(myset)
```

```
bad
bda
dba
dab
abd
adb
```