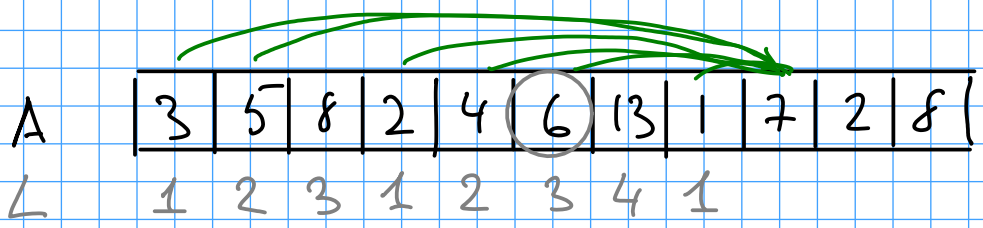
# Динамическое программирование

## Задача о самой длинной возрастающей подпоследовательности




1. $L[i]$ - длина самой длинной $\uparrow$ подп. заканчивающейся в ячейке $i$

2. $L[1] = 1$

$$L[i] = \max_{\substack{k<i \\ A[k]<A[i]}} L[k] + 1$$

3. Заполним $L$ слева направо $\Rightarrow$ $\max L[i] \longleftrightarrow$ самая длинная $\uparrow\uparrow$ подпосл.

4*. Восстановление: $P[i]$ - индекс предыдущего эл-та в подпосл.

```
L[N] = {1...1}, P[N] = {0...0}
L[1] = 1
for i = 2 to N:
    for k = 1 to i-1:
        if A[k] < A[i]:
            if L[k]+1 > L[i]:
                L[i] = L[k]+1
                P[i] = k
```

Время работы: $O(n^2)$

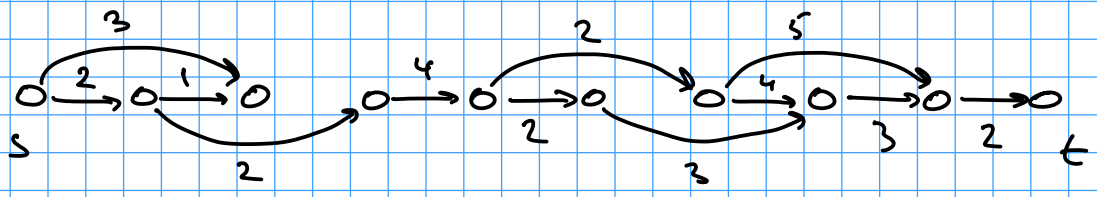
Память: $O(n)$

Можно: $O(n \log n)$

## Ход решения:

1. Выделить подзадачи
2. База и соотношение на подзадачах
3. Порядок вычисления

## Связь с кратчайшими путями:




1. D - массив с расстояниями от s до v
   D[v] - расстояние от s

2. D[s] = 0
   $D[v] = \min_{(u,v)} D[u] + w(u,v)$ ← все ребра (u,v)

3. Порядок: в порядке топологической сортировки

## Задача о рюкзаке

Рюкзак вместимости W

N предметов:

$w_1, \dots, w_N$ - веса предметов

$v_1, \dots, v_N$ - ценность предметов

Задача: максимизировать ценность предметов в рюкзаке

1. Непрерывный рюкзак

Предметы можно делить.

"Жадный алгоритм" из соображений удельной стоимости $\frac{v_i}{w_i}$

## 2. Рюкзак с повторениями / неограниченный рюкзак

„Жадный" алгоритм не работает.

$W = 10$ $w_1 = 6$ $v_1 = 24$

$w_2 = 5$ $v_2 = 15$

Жадное решение: 24

Оптимальное: 30

1. $V[i]$ – решение для рюкзака размера $i$
2. $V[0] = 0$

$$V[i] = \max_{1 \le k \le n} (v_k + V[i - w_k])$$

3. Порядок: слева направо.

Время: $O(N \cdot W)$ Память: $O(W)$

```
for i = 1 to W:
    for k = 1 to N:
        V[i] = max(V[i], V[i - w_k] + v_k)
```

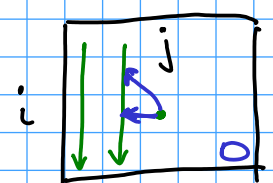
*

## 3. Рюкзак без повторений / ограниченный рюкзак

∀ предмет в единственном экземпляре

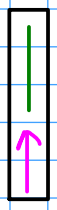
1. $V[i, j]$ – решение для рюкзака размера $i$, если использовать только предметы $\nu$ $1 \dots j$
2. $V[0, j] = 0$ $V[i, 0] = 0$

$$V[i, j] = \max(V[i - w_j, j-1] + v_j, V[i, j-1])$$

3. 


Время: $O(nW)$ Память: $O(nW)$

NB: Если хранить только последний столбец $\Rightarrow$ память $O(W)$, но не восстановить реш-е.

*  Если сделать перестановки в алг. для неогр. процесса $\Rightarrow$ алг. для ограниченного.

NB: $O(N \cdot W) = O(N \cdot 2^{|W|})$

Длина входа: $|W| + \sum_{k=1}^{N} (|w_k| + |v_k|)$

$|W| = \Theta(\log_2 W)$

Задача о перемножении матриц.

$$A \cdot (B \cdot C) = (A \cdot B) \cdot C$$

$A$ – $n \times m$
$B$ – $m \times k$
$C$ – $n \times k$

$A \cdot B = C$
Нужно: $n \times m \times k$

Вход: $A_1 \times A_2 \times \dots \times A_n$
Выход: min число умножений

Δ $A, B, C$ $\quad A - 10 \times 20 \quad B - 20 \times 40$
$C - 40 \times 100$

→ $(A \times B) \times C$ — $10 \times 20 \times 40 + 10 \times 40 \times 100 = 48\,000$
$A \times (B \times C)$ — $20 \times 40 \times 100 + 10 \times 20 \times 100 = 100\,000$

1. $C[i, j]$ – min # операций для $A_i \times \dots \times A_j$
2. $C[i, i] = 0$

Вход: $m_0, m_1, \dots m_n$, $A_i - m_{i-1} \times m_i$

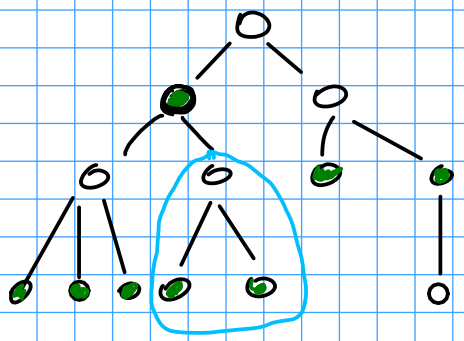
$$C[i, j] = \min_{i \le k < j} C[i, k] + C[k+1, j] + m_{i-1} \times m_k \times m_j$$

3. Порядок:

```
for s = 1 to N-1
    for i = 1 to N-s
        C[i, i+s] = min ...
```

Время: $O(N^3)$ Память: $O(N^2)$

## Максимальное независимое множество в деревьях



IS = independent set

Мн-во вершин: нет ребра, соед. вершины этого множества

1. IS[v] - макс. нез. мн-во в поддереве с корнем в v

2. IS[l] = 1, l - лист.

$$IS[v] = \max\left(\sum_{w\text{-сын } v} IS[w],\ 1 + \sum_{w\text{-внук } v} IS[w]\right)$$

(1 — сообщить. v)

3. Порядок: снизу вверх

Замечание: многие из этих алгоритмов можно реализовать рекурсивно + „мемоизация".