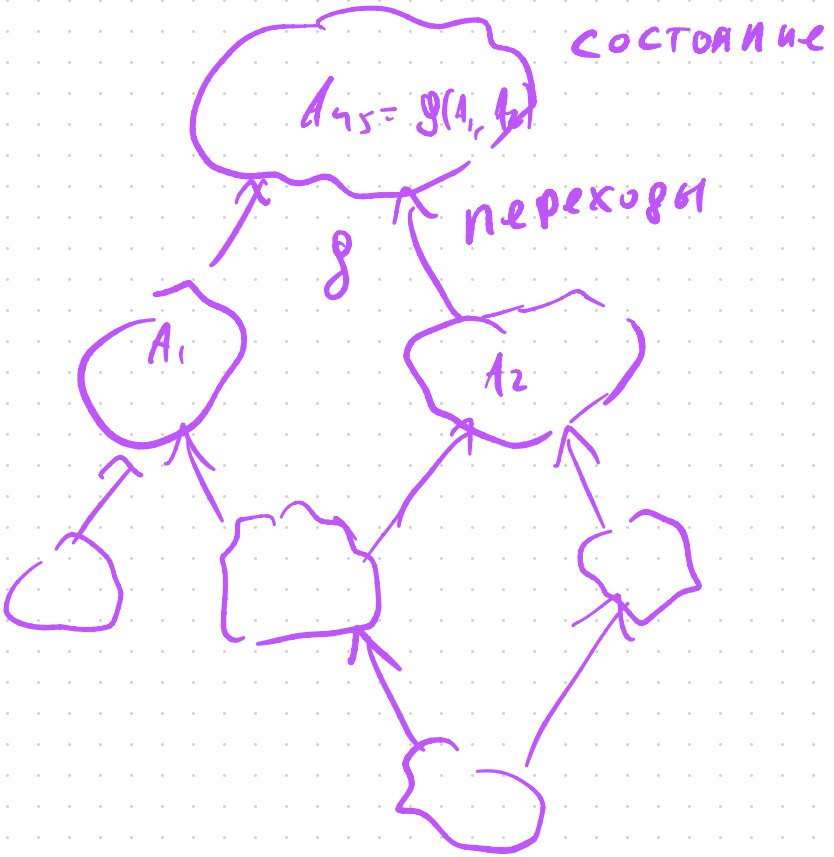
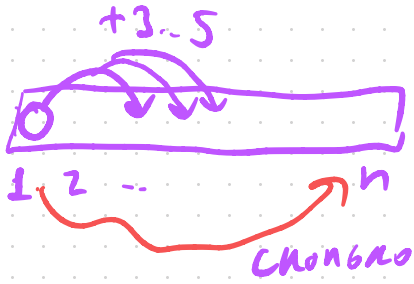
Динамическое Программирование

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

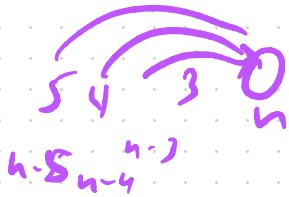
R. Bellman 1950x



Задача про полосну



Ans_n – число путей $1 \to n$



$$\begin{cases} Ans_n = Ans_{n-3} + Ans_{n-4} + Ans_{n-5} \\ Ans_1 = 1 \end{cases}$$

```
f = [0 for i in range(1, n+1)] # f[1..n] = 0

f[1] = 1
for i in range(2, n+1): # for i in 2..n:
    for step in [3,4,5]:
        if i > step:
            f[i] += f[i - step]
```

print f[n]

1) Время работы

$O(n)$ ← если мы считаем что $a+b$ работает за $O(1)$

$O(n^2)$ ← наши

$f_n = \Theta(c^n)$

2) Динамика

- Состояния
  f_n
- Переходы
  $f_n = f_{n-3} + f_{n-4} + f_{n-5}$
- Базовые состояния
  $f_1 = 1$

Динамика вперёд назад и лень

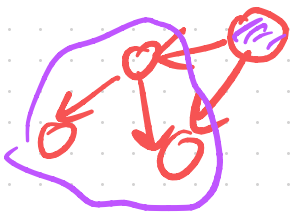
```
f = [0 for i in range(1, n+1)] # f[1..n] = 0

f[1] = 1
for i in range(2, n+1): # for i in 2..n:
    for step in [3,4,5]:
        if i > step:
            f[i] += f[i - step]
```

Динамика назад

$f_1 \dots f_{n-1}$ - уже вычислены

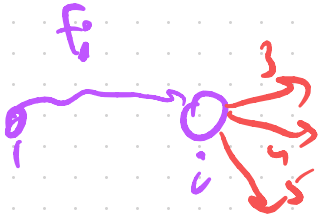
f_n - вычисляется через предыдущие



вперёд

```
f = [0 for i in range(1, n+1)] # f[1..n] = 0

f[1] = 1
for i in range(1, n+1): # for i in 1..n:
    for step in [3,4,5]:
        if i + step <= n:
            f[i + step] += f[i]
```



на момент итерации i

f_i - посчитано верно

Ленивая — маркер что состояние не посчитано

```python
f = [-1 for i in range(1, N+1)] # f[1..N] = -1

def calc_f(n):
    if f[n] != -1:
        return f[n]
    if n == 1:
        return 1 # база

    # общий случай
    result = 0
    for step in [3,4,5]:
        if n > step:
            result += calc_f(n-step)

    f[n] = result # запоминаем ответ в f[n] чтобы не считать дважды
    return result
```

$f_1 = 1$

$f_n = f_{n-3} + f_{n-4} + f_{n-5}$

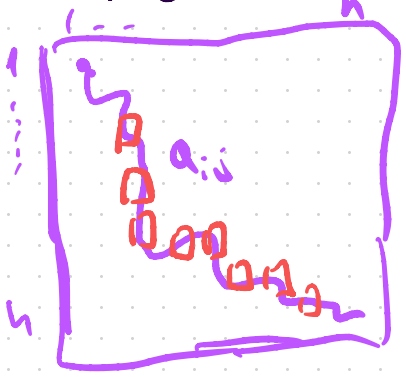
узнать f_n

Мемоизация

Плюс такого подхода, что не нужно упорядочивать состояния явно

$$\begin{cases} f_n = f_{n-1} + f_{n-2} \\ f_1 \end{cases} \leftarrow \text{Д.П.}$$

Пути на матрице



→ max

① Состояния

f_{ij} = лучший путь (вес) из (1,1) в (i,j)

② переходы

$$f_{ij} \leftarrow \max(f_{i-1,j} + a_{ij})$$

$$f_{ij} \leftarrow \max(f_{i,j-1} + a_{ij})$$

③ база

$$f_{1,1} = a_{1,1}$$

Вперёд

f = массив n×n

$f_{1,1} = a_{11}$

for i := 1 .. n

for j := 1 .. n

if i ≠ n:

$f_{i+1,j} = MAX(f_{i+1,j}; f_{ij} + a_{i+1,j})$

if j ≠ n

$f_{i,j+1} = MAX(f_{i,j+1}; f_{ij} + a_{i,j+1})$

$f_{i,j}$

f = максимальная сумма, которую можно собрать, чтобы попасть в клетку (i, j)

$f_{1,1} = a_{11}$

for $i := 1 .. n$

  for $j := 1 .. n$

    f_{ij} — нужное для максимума

    if $i \neq 1$:

      $f_{ij} = \max(f_{ij}, f_{i-1,j} + a_{ij})$

    if $j \neq 1$

      $f_{ij} = \max(f_{ij}, f_{i,j-1} + a_{ij})$

Восстановление ответа

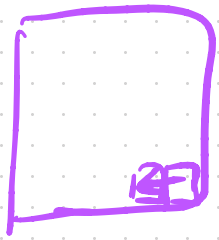
① Массив предков

$f_{ij} \leftarrow f_{i-1,j} + a_{ij}$

$f_{ij} \in f_{i,j-1} + a_{ij}$

$par_{ij} = 0/1$ в зависимости
L и от того где мах

② Без массива предков



Ещё раз вспомним,

$f_{ij} \leftarrow f_{i-1,j} + a_{ij}$

$f_{ij} \leftarrow f_{i,j-1} + a_{ij}$

Найдём где $\max / f_{ij}$

Subset Sum
(рюкзак без стоимостей)

$a_1 \dots a_n$ - веса различных предметов

$= S$ вес

$a_1 \dots a_n$ - веса
$p_1 \dots p_n$ - полезность

учесть max полезность

$\sum$ вес $\leq W$

2 5 7 4 → 13

Состояние:

$can_{i,w} = 0/1$

правда ли что можно набрать вес w используя первые i предметов (подмножество первых $a_1 \dots a_i$ предм)

Ответ:

$can_{n,S} = ответ$

переходы

$can_{i,S} \Leftarrow can_{i-1,S}$

$can_{i,S} \Leftarrow can_{i-1,S-a_i}$

База:

$can_{0,0} = 1$

$can_{0,s} = 0$
$s>0$

Сам

$O(nS)$ время

$O(nS)$ память

восстановление ответа
рюкзак без пер

→ цел: $O(S)$ память

сам: -1

сам i

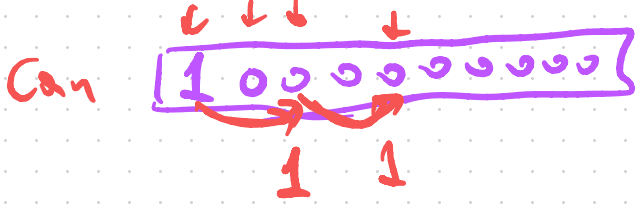
=> $O(S)$ память

без восст. ответа

O(S) памяти, но ответ восстановить.

```
can[0,0] = 1
for i = 1..n
    for s = 0..W:
        can[i,s] = can[i-1,s]
        if s ≥ a_i
            can[i,s] |= can[i-1,s-a_i]
```

```
can[0] = 1
for i = 1..n
    for s = W...0
        ~~can[s] = can[s]~~
        if s ≥ a_i
            can[s] |= can[s-a_i]
```

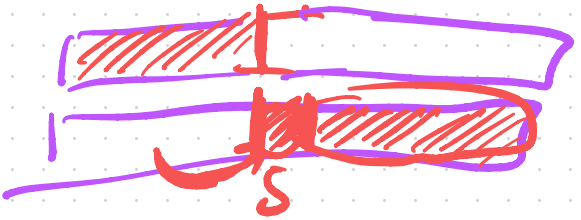


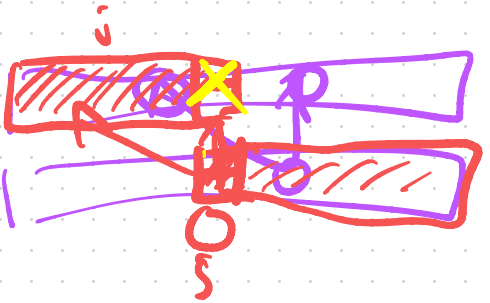
S = S - v

0 → 2 ; 2 → 4

V X

0 → 4





$can_0 = 1$

for $i = 1 \ldots n$

  for $s = W \ldots 0$

~~$can_s = can_s$~~

    if $s \geq a_i$ and can_{s-a_i} and $!can_s$

      $can_s = 1$

      $par_s = a_i$


```
cur = S
while cur ≠ 0:
    ans.append(par[cur])
    cur -= par[cur]
```

Наибольшая возр. подпосл.

$a_1 \dots a_n$

$a_{i_1} \; a_{i_2} \dots a_{i_k}$

k — max

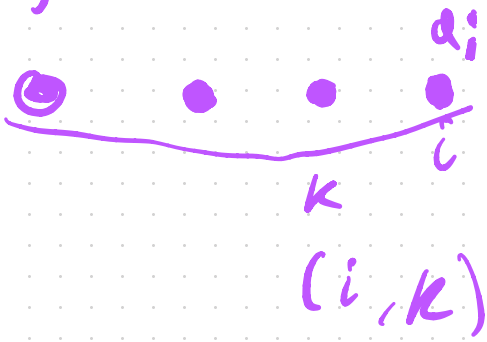
i_j ↗

a_{i_j} ↗

10 2 5 3 11

↳ 2 5 11

$O(n^2)$:



(i, k)

(i, k)

$\downarrow$ если $a_i < a_{i'}$

$(i', k+1)$ $\quad i < i'$

$O(n^3)$ $can_{i,k} = 0/1$

$can_{i',k} \xleftarrow[a_i < a_{i'}]{i < i'} can_{i,k-1}$

can_i 

$O(n^2)$ len_i = длина наиб. возр. подп. с концом в a_i

$len_i = \max\limits_{i < i',\ a_i < a_{i'}} len_{i'} + 1$

$len_i \leftarrow 1$

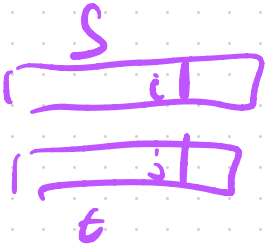
```
len = [1 ..... 1]
for i = 1 .. n:
    for i' = 1 .. i-1:
        if a_i' < a_i:
            len[i] = max(len[i],
                         len[i']+1)
```

Edit distance

abcd
↓ стереть
abd
↓ поменять
abe
↓ вставить
abef

dist(S, t) = ?

$O(n \cdot m)$ $n = |s|$ $m = |t|$



$s_1 \dots s_n$

$dp_{i,j} = edit(s_1 \dots s_i, t_1 \dots t_j)$

$dp_{0,0} = 0$ — база

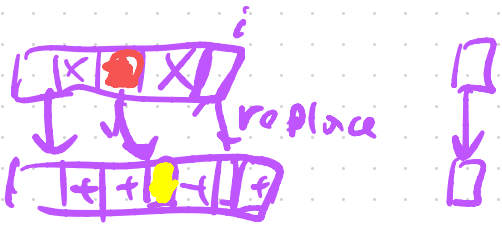
$dp_{n,m}$ — ответ

Переходы?

$dp_{i,j} \leftarrow dp_{i-1,j-1}$ $s_i = t_j$

$dp_{i,j} \leftarrow 1 + dp_{i-1,j}$

$dp_{i,j} \leftarrow 1 + dp_{i,j-1}$



$$dp_{i,j} \xleftarrow[s_i \neq t_j]{} dp_{i-1,j-1} + 1$$