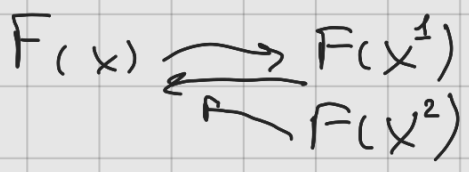
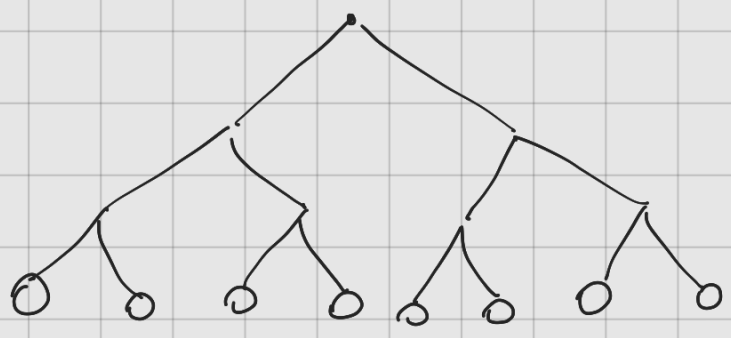
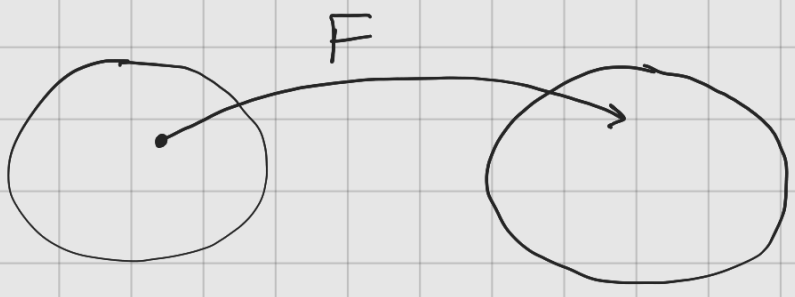
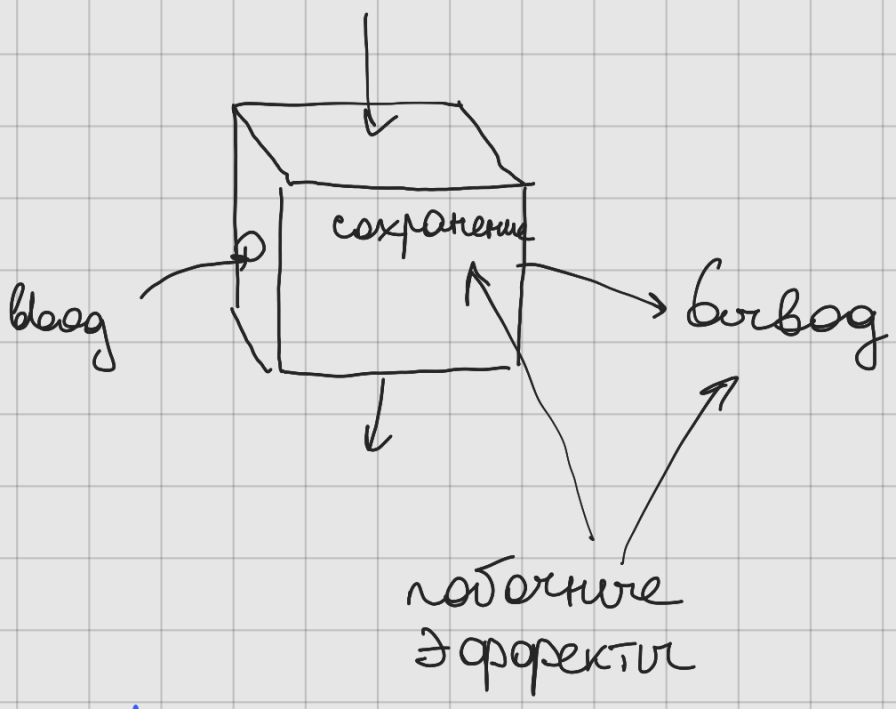
Динамичное программирование









@lru_cache

@cache

cache - проверяет, для каких x вычислялись значения

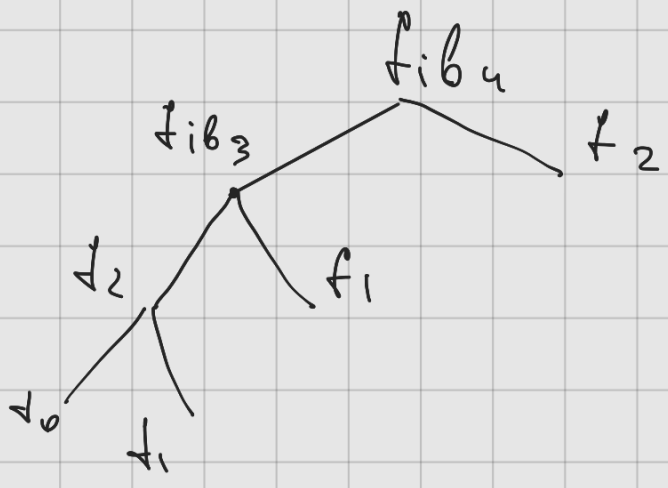
(кэширование)

@lru_cache(size=10000)

import functools

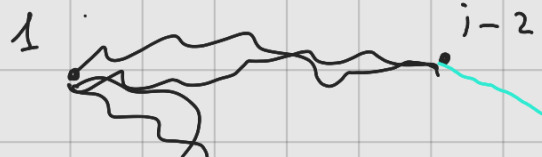
@functools.lru_cache

def

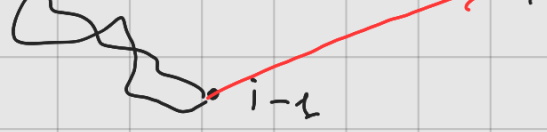


Динамическое программирование (снизу)

K_i → способов добраться от 1 до i



$K_i = K_{i-1} + K_{i-2}$



$K_0 = 0$

$K_1 = 1$

```
K = [None]*(n+1)
K[0] = 0
K[1] = 1
for i in range(2, n+1):
    K[i] = K[i-2] + K[i-1]
print(K[n])
```

Можем оформить в виде ф-ии.

C_i - min общая стоимость добраться от 1 до i-й клетки

$C_i = \min(C_{i-1}, C_{i-2}) + P_i$

```
def min_cost_path(price: list, n):
    C = [None]*(n+1)
    C[0] = float('inf')
```

```
C[1] = price[1]
for i in range(2, n+1):
    C[i] = min(C[i-1], C[i-2]) + price[i]
print(C[n])
path = [n]
i = n
while i != 1:
    if C[i] = C[i-1] + P[i]:
        i = i-1
    else:
        i = i-2
    path.append(i)
path.reverse()
return path.
```

Макс. сумма подотрезка

A = [1, -2, 3, -5, 0, 6, -3, 7, -9, 2, -15, 6, 3, -5] (i)

S = [0, 1, -1, 2, -3, -3, 3, 0, 7, -2, 0, -15, -9, 4, -1]

S[i] = S[i-1] + A[i]

Smin = [0, 0, -1, -1, -3, -3, -3, -3, -3, -3, -3, -15, -15, -15, -15]

$S[i] = sum(A[0:i])$

$S_i^{min} = min(S[0:i])$

$S_i^{min} = min(S_{i-1}^{min}, S_i)$

$sum(S[a:i]) = sum(S[:i]) - sum(S[:a])$

$S_i^{max} = S_i - S_i^{min}$

$A = [5, -7, 2, 3, -4, 9, -5, 6]$

$S = [0, 5, -2, 0, 3, -1, 8, 3, 9]$

$S_{min} = [0, 0, -2, -2, -2, -2, -2, -2, -2]$