Динамическое программирование

Билет № ?

12 декабря 2025 г.

1 Задача о наибольшей возрастающей подпоследовательности (НВП)

Пусть дан массив a_1, a_2, ... $a_n \in \mathbb{R}$. Необходимо найти такую наибольшую по длине последовательность $i_1, i_2, ...i_l : i_1 < i_2 < ... < i_l; a_{i_1} < a_{i_2} < ... < a_{i_l}$.

Будем хранить массив dp размера n и в k-ой ячейке будем хранить состояние "длина НВП, которая заканчивается на k-ой позиции". База: dp[k] = 1.

Тогда формула пересчета выглядит так:

$$dp[k] = max\{dp[k']\} + 1,$$
$$где\ k' < k, a_{k'} < a_k.$$

Require: $A = [a_1, a_2, \ldots, a_n]$

function LIS(A)
    $n \leftarrow |A|$
    $dp \leftarrow [1, 1, \ldots, 1]$ ▷ массив длины n
    for k in range(2, n) **do**
        for k' in range (0, k - 1) **do**
            if $a_k > a_{k'}$ **then**
                $dp[k] = \max\{dp[k], dp[k'] + 1\}$
            end if
        end for
    end for
    return max(dp)
end function

Но это решение за $O(n^2)$: (Надо уйти от условия $a_{k'} < a_k$. Давайте отсортируем массив $(a_1, 1), (a_2, 2)...(a_n, n)$ первым приоритетом по a_k по возрастанию, а вторым по k по убыванию.

Algorithm 1 Восстановление НВП

```
1: procedure RestoreLIS(A, dp)
2:     n ← length(A)
3:     max_len ← max(dp)
4:     current_pos ← arg max(dp)              ▷ индекс с максимальным dp
5:     LIS ← [ ]
6:     current_len ← max_len
7:     LIS.add(A[current_pos])                ▷ добавляем в начало
8:     current_len − −
9:     while current_len > 0 do
10:        for i in range(current_pos-1, 1, -1) do
11:            if dp[i] = current_len and A[i] < A[current_pos] then
12:                current_pos ← i
13:                break
14:            end if
15:        end for
16:    end while
17:    return LIS
18: end procedure
```

Примечание. max() - поиск максимума на подотрезке с помощью кучи (O(log(n)). Итого: асимптотика O(n*log(n)). Псевдокод убежал на 3 стр.

2 Задача о наибольшей общей подпоследовательности (НОП)

Пусть даны два массива A = $\{a_1, a_2, ...a_n\}$, B = $\{b_1, b_2, ...b_m\}$. Надо найьт ьакле максимальное p, что найдутся последовательности $i_1 < i_2 < ... < i_p$ и $j_1 < i_2 < ... < j_p$, что $\forall k \in 1...p \mapsto a_{i_k} = b_{j_k}$.
Заведем матрицу dp размера $m * n$, изначально заполненный нулями. В клетке (k, t) будем хранить длину НОП для a[:k], b[:t]. Тогда пересчет будет таков:

$$dp[k][t] = \begin{cases} \text{dp[k - 1][t - 1] + 1, if } a_k = b_t \\ \text{max\{dp[k][t-1], dp[k-1][t]\}, else} \end{cases}$$

Algorithm 2 Improved LIS

```
1: function LIS(A)
2:     A^S ← sort((a_i, i) for i = 1..n) ▷ По возрастанию a_k, по убыванию k
3:     ST ← [-∞, −∞, ... −∞]
4:     dp ← [0] × n
5:     for (a_k, k) ∈ A^S do
6:         dp[k] ← max(dp[k], 1 + ST.query(1, k − 1))
7:         ST.update(k, dp[k])
8:     end for
9:     return max(dp)
10: end function
```

Примечание. Сложность O(nm). Ответ в dp[n][m].

3 Задача о рюкзаке

Есть n предметов, каждый из них имеет вес w_i и смоимость c_i. Надо взять предметы в рюкзак так, чтобы их суммарный вес не превосходил W, а стоимость максимальна. Будем хранить в dp[i][w] результат для подзадачи, когда берем первые i элементов, а вместимость рюкзака - w. База: всё заполнено нулями. Тогда пересчёт:

$$\text{dp[i][w]} = \max\begin{cases} dp[i-1][w-w_i] + c_i \text{ if i-ый взяли} \\ \text{dp[i - 1][w], else} \end{cases}$$

Тогда ответ в dp[n][W]