Лекция 12. Динамическое программирование I.

#вшпи #аисд #дп

Автор конспекта: Гридчин Михаил

Идея ДП

> **Def**
>
> **Динамическое программирование — подход к решению задач, где рассматривается сведение подзадачи к меньшей с оптимизированным перебором возможных вариантов.**

Пять шагов для решения задач на ДП:

1. **Что хранит в себе состояние динамики?**
2. **Какая база?**
3. **Как пересчитывать состояние?**
4. **В каком порядке вести пересчёт?**
5. **Где лежит ответ?**

Наивная идея: числа Фибоначчи

Рассмотрим код:

```
int Fib(int n) {
  if (n < 2) {
    return 1;
  }
  return Fib(n - 1) + Fib(n - 2);
}
```

Проблема данного подхода заключается в том, что при вычислении одной большой задачи мы также много раз вычисляем идентичные меньшие подзадачи. Асимптотика такой функции $O(\phi^n)$.

Числа Фибоначчи как ДП

Рассмотрим числа Фибоначчи с точки зрения динамического программирования.

Состояние динамики. Пусть $\mathrm{dp}[i] = F_i$.
База. $\mathrm{dp}[0] = \mathrm{dp}[1] = F_0 = F_1 = 1$.
Пересчёт. По определению $F_i = F_{i-1} + F_{i-2}$, так что $\mathrm{dp}[i] = \mathrm{dp}[i-1] + \mathrm{dp}[i-2]$.
Порядок пересчёта. По возрастанию i.
Ответ. $\mathrm{dp}[n]$.

Код:

```
int Fib(int n) {
  int dp[n + 1];
  dp[0] = dp[1] = 1;
  for (int i = 2; i <= n; ++i) {
    dp[i] = dp[i - 1] + dp[i - 2];
  }
  return dp[n];
}
```

Пример: задача о кузнечике

Постановка задачи

Дан массив $a_1, \ldots, a_n, a_i \in \mathbb{R}$. Находясь изначально в $a_0 = 0$, необходимо "допрыгать" до a_n, максимизировав сумму a_i, в которых побывал кузнечик. При этом он может прыгать на 1 или 2 клетки вперёд.

Пример.

a_0	a_1	a_2	a_3	a_4	a_5
0	-1	-2	-10	-100	0

Пример жадной стратегии

Давайте каждый раз выбирать максимум, доступный нам. Но тогда кузнечик в приведённом примере посетит вообще все клетки, что не оптимально, поэтому решать жадно такую задачу нельзя.

Решение

Построим следующую динамику:
Состояние динамики. $\mathrm{dp}[i]$ - величина ответа для первых i клеток, то есть, если бы цель кузнечика была допрыгать до a_i.
База. $\mathrm{dp}[0] = 0, \mathrm{dp}[1] = a_1$.
Пересчёт. $\mathrm{dp}[i] = a_i + \max\{\mathrm{dp}[i-1], \mathrm{dp}[i-2]\}$.

Порядок пересчёта. По увеличению i.
Ответ. $\mathrm{dp}[n]$.

Восстановление ответа в ДП

Для каждого состояния динамики будем хранить предка — состояние, откуда пришли в текущее. Зная лучшее состояние, можно, двигаясь по его предкам, восстановить ответ.

Пример: НВП

> **Задача о наибольшей возрастающей подпоследовательности**
>
> Дан массив $a_1, \ldots, a_n, a_i \in \mathbb{R}$. Требуется найти наибольшую по длине возрастающую подпоследовательность. То есть найти такие индексы $1 \le i_1 < i_2 < \ldots < i_k \le n$, что $a_{i_j} < a_{i_{j+1}}$ и k максимально.

Решение за $O(n^2)$

Построим следующую динамику:
Состояние динамики. $\mathrm{dp}[i]$ - длина НВП, заканчивающейся на индексе i.
База. $\mathrm{dp}[i] = 1 \; \forall i \in \{1, 2, \ldots, n\}$.
Пересчёт.

$$\mathrm{dp}[i] = 1 + \max_{j<i, a_j<a_i} \mathrm{dp}[j]$$

Порядок пересчёта. По увеличению i.
Ответ.

$$\max_{1 \le i \le n} \mathrm{dp}[i]$$

Решение за $O(n \log n)$

Улучшим предыдущее решение. Динамику будем поддерживать ту же. Давайте поддерживать дерево отрезков на индексах-a_i и значениях $\mathrm{dp}[i]$, будем хранить максимум на отрезке. Теперь для того, чтобы найти максимум среди всех $j < i, a_j < a_i$ просто сделаем запрос на префиксе к дереву отрезков, а чтобы добавить очередное значение, сделаем обновление в точке. Ответ - максимум на всём дереве отрезков. Каждая операция работает за $O(\log n)$, поэтому общая асимптотика решения $O(n \log n)$.

Пример: НОП

> **Постановка задачи**

Даны два массива $a_1, \ldots, a_n$ и $b_1, \ldots, b_m$. Необходимо извлечь наибольшую по длине общую последовательность. То есть найти такие индексы $1 \le i_1 < \ldots < i_k \le n$ и $1 \le j_1 < \ldots < j_k \le m$, что $a_{i_t} = b_{i_t}$ и k максимально.

Пример. $a = [1, 3, 5, 2, 1, 4]$, $b = [3, 1, 5, 1, 4]$. У них НОП равна $c = [3, 5, 1, 4]$ и её длина равна 4.

Решение за $O(n^2)$

Построим следующую динамику:
Состояние динамики. $\mathrm{dp}[i][j]$ - длина НОП для префиксов $a[:i], b[:j]$.
База. $\mathrm{dp}[i][j]$ = 0.
Пересчёт.

$$\mathrm{dp}[i][j] = \begin{cases} \mathrm{dp}[i-1][j-1] + 1, & \text{если } a_i = b_j \\ \max(\mathrm{dp}[i-1][j], \mathrm{dp}[i][j-1]), & \text{иначе} \end{cases}$$

Порядок пересчёта. По увеличению i, по увеличению j.
Ответ. $\mathrm{dp}[n][m]$.

Пример: 0-1 Рюкзак

⁝≡ Постановка задачи (линейно-алгебраическая)

Даны два вектора $w = \{w_0, \ldots, w_{n-1}\}$ и $c = \{c_0, \ldots, c_{n-1}\}$, оба вектора из $\mathbb{N}^n$, а также число $W \in \mathbb{N}$. Требуется построить битовый вектор $b \in \{0, 1\}^n$ такой, что

$$\begin{cases} (w, b) \le W \\ (c, b) \to \max \end{cases}$$

⁝≡ Постановка задачи (классическая)

Есть n предметов, каждый из них имеет вес w_i и стоимость c_i. Требуется взять какие-то предметы в рюкзак так, чтобы их суммарный вес не превосходил W, а стоимость была максимальна.

Note. Эта задача называется "0-1 рюкзак", потому что каждый из предметов можно или взять (один раз) в рюкзак, или не взять.

Решение за $O(nW)$ методом ДП

Построим следующую динамику:

Состояние динамики. Пусть $\mathrm{dp}[i][w]$ - это стоимость рюкзака весом ровно w. если разрешено брать только первые i предметов.

База. $\mathrm{dp}[0][0] = 0, dp[0][w] = -\infty$.

Пересчёт.

$$\mathrm{dp}[i][w] = \max \begin{cases} \mathrm{dp}[i-1][w], & i\text{-й предмет не берём} \\ \mathrm{dp}[i-1][w-w_i] + c_i, & i\text{-й предмет берём и } w \geq w_i \end{cases}$$

Таким образом,

$$\mathrm{dp}[i][w] = \begin{cases} \max(\mathrm{dp}[i-1][w], \mathrm{dp}[i-1][w-w_i] + c_i), & \text{если } w \geq w_i \\ \mathrm{dp}[i-1][w], & \text{иначе} \end{cases}$$

Порядок пересчёта. По увеличению i. по увеличению w.

Ответ.

$$\max_{w=0}^{W}\{dp[n][i]\}$$