# Модуль 3 Динамическое программирование
# Лекция 3.6 Наибольшая общая подпоследовательность

Эта лекция посвящена одной задаче про последовательности.

Пусть у нас есть некоторая конечная последовательность чисел $a_1, a_2, \ldots, a_n$. Её *подпоследовательностью* называется набор чисел из последовательности, не обязательно идущих подряд, но с сохранением относительного порядка чисел. Формально: выбирается набор индексов $i_1, i_2, \ldots, i_k$, расположенных в порядке возрастания:

$$1 \leqslant i_1 < i_2 < \cdots < i_k \leqslant n.$$

Здесь $k$ — длина подпоследовательности. Подпоследовательностью будет новая последовательность чисел с этими индексами: $a_{i_1}, a_{i_2}, \ldots, a_{i_k}$.

На рис. 1 приведены примеры подпоследовательностей. Можно включить в подпоследовательность, например, числа через одно. Или подпоследовательность может состоять из одного числа. Можно сделать подпоследовательность из двух одинаковых чисел, если в последовательности встречаются одинаковые числа. Но порядок чисел в подпоследовательности сохраняется, перемешивать их нельзя.

**3** 2 **5** 1 **3** 7 **2** ⇒ 3 5 3 2
3 2 **5** 1 3 7 2 ⇒ 5
**3** 2 5 1 **3** 7 2 ⇒ 3 3
**3** **2** 5 1 3 **7** 2 ⇒ 3 2 7

Рис. 1. Примеры подпоследовательностей

Рассмотрим следующую задачу.

**Наибольшая общая подпоследовательность**
Даны две последовательности. Нужно найти их наибольшую общую подпоследовательность, то есть совпадающую подпоследовательность наибольшей длины.

Для примера на рис. 2 наибольшая общая подпоследовательность имеет вид 4 2 1 6. Ее длина равна 4.

3 2 **4** **2** **1** 7 **6**
**4** **2** 5 3 **1** **6** 5 2 3

Рис. 2. Пример к задаче «Наибольшая общая подпоследовательность»

**Упражнение 3.6.1**
Найдите длину наибольшей общей подпоследовательности последовательностей.
1 2 3 4
3 1 4 5 3 1 2 4

Решим задачу о нахождении наибольшей общей подпоследовательности методом динамического программирования. Пусть первая последовательность состоит из чисел $a_1, a_2, \ldots, a_n$, а вторая — из чисел $b_1, b_2, \ldots, b_m$. Их длины не обязательно совпадают, как и в приведённых примерах. Рассмотрим в первой последовательности префикс (начальный отрезок) длины $i$, а во второй — префикс длины $j$, и будем искать $d_{ij}$ — длину наибольшей общей подпоследовательности для этих префиксов. Для примера на рис. 3 эта подпоследовательность отмечена, и её длина равна 2. Таким образом, состояние динамики — это пара префиксов.

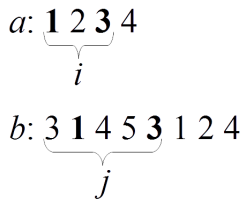


Рис. 3. Состояние динамики

Определим переходы в состояние $(i, j)$. Если последние элементы в префиксах $a_i$ и $b_j$ совпадают, то мы можем включить их в общую подпоследовательность, отбросить и найти наибольшую общую подпоследовательность для префиксов длины $(i-1)$ и $(j-1)$. Кроме того, мы можем просто не использовать какой-то из последних элементов. Если мы отбросим $a_i$, получим переход из состояния $(i-1, j)$. Если отбросим $b_j$, будет переход из состояния $(i, j-1)$ (см. рис. 4).

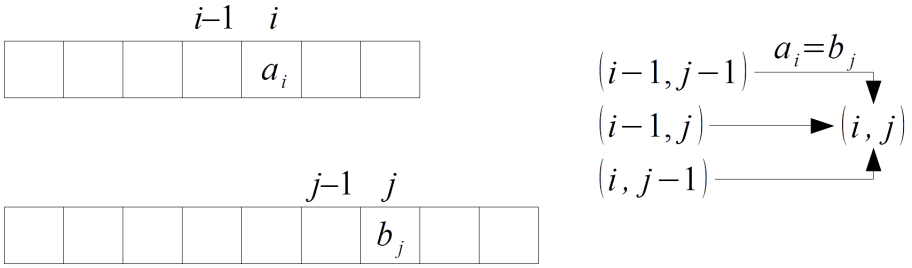


Рис. 4. Переходы динамики

Получаем следующую расчётную формулу:

$$d_{ij} = \max(d_{i-1,j-1} + 1, d_{i-1,j}, d_{i,j-1}). \quad (1)$$

На рис. 5 приведён код программы.

```
for (int i = 0; i <= n; i++)
    d[i][0] = 0;
for (int j = 0; j <= m; j++)
    d[0][j] = 0;

for (int i = 1; i <= n; i++)
    for (int j = 1; j <= m; j++)
    {
        d[i][j] = max(d[i - 1][j], d[i][j - 1]);
        if (a[i - 1] == b[j - 1])
            d[i][j] = max(d[i][j], d[i - 1][j - 1] + 1);
    }
cout << d[n][m] << endl;
```

Рис. 5. Решение задачи о наибольшей общей подпоследовательности

Первые два цикла — это инициализация.

**for** (**int** i = 0; i <= n; i++)
    d[i][0] = 0;
**for** (**int** j = 0; j <= m; j++)
    d[0][j] = 0;

Мы инициализируем все состояния динамики, у которых $i = 0$ или $j = 0$. У них один из префиксов пустой, поэтому наибольшая общая последовательность с любой последовательностью тоже будет пустая — из нуля элементов.

Следующие два вложенных цикла нужны для подсчёта по формуле (1).

**for** (**int** i = 1; i <= n; i++)
    **for** (**int** j = 1; j <= m; j++)

Сначала выбирается максимум по двум состояниям, для которых не надо прибавлять единицу.

d[i][j] = max(d[i - 1][j], d[i][j - 1]);

Потом сравниваются a[i-1] и b[j-1] (так получается, потому что в $a$ и $b$ 0-индексация).

**if** (a[i - 1] == b[j - 1])

Если символы совпали, мы присваиваем значение d[i][j] максимуму из его текущего значения и того, которое получается при переходе.

d[i][j] = max(d[i][j], d[i - 1][j - 1] + 1);

В результате ответ на задачу — d[n][m].

cout << d[n][m] << endl;

Как правило, в этой задаче требуется вывести саму общую подпоследовательность. Этот вопрос останется на самостоятельную доработку.

Подведём краткие итоги модуля «Динамическое программирование». Какие этой теме наиболее принципиальные моменты?

1. Во-первых, нужно придумать, что будет состоянием динамики. Это обычно самое сложное в задаче. Состояние динамики может включать одно число, два или больше. На контестах бывают задачи, где состояние динамики зависит от 4-5 параметров.

2. Дальше нужно понять, какие переходы существуют между состояниями.

3. Иногда удобнее делать динамику «вперёд», то есть рассматривать переходы не В данное состояние, и ИЗ него и обновлять будущие значения.

4. Инициализация. Нужно инициализировать все значения, необходимые для подсчёта по формуле. Бывают задачи, где требуется инициализировать более одного значения.

5. Ацикличность, отсутствие циклов. Для состояний необходим порядок, в котором их можно последовательно считать, чтобы каждое следующее состояние использовало только ранее посчитанные.

6. Вывод сертификата. Это дополнительная техническая задача, не очень сложная в идейном плане.