

САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ  
УНИВЕРСИТЕТ  
ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ  
ФАКУЛЬТЕТ ИНФОКОММУНИКАЦИОННЫХ ТЕХНОЛОГИЙ

Отчет по лабораторной работе №7  
по курсу «Алгоритмы и структуры данных»  
Тема: Динамическое программирование №1.  
Вариант 3

Выполнила:  
Блинова П. В.  
К3139 (номер группы)

Проверил:  
Афанасьев А. В.

Санкт-Петербург  
2024 г.

## Содержание отчета

Содержание отчета	2
Задачи по варианту	3
Задача №4. Наибольшая общая подпоследовательность двух последовательностей	3
Задача №7. Шаблоны	5
Дополнительные задачи	7
Задача №1. Обмен монет	7
Задача №5. Наибольшая общая подпоследовательность трех последовательностей	8
Вывод	10

## Задачи по варианту

### Задача №4. Наибольшая общая подпоследовательность двух последовательностей

#### Текст задачи.

##### 4 задача. Наибольшая общая подпоследовательность двух последовательностей

Вычислить длину самой длинной общей подпоследовательности из двух последовательностей.

Даны две последовательности  $A = (a_1, a_2, \dots, a_n)$  и  $B = (b_1, b_2, \dots, b_m)$ , найти длину их самой длинной общей подпоследовательности, т.е. наибольшее неотрицательное целое число  $p$  такое, что существуют индексы  $1 \leq i_1 < i_2 < \dots < i_p \leq n$  и  $1 \leq j_1 < j_2 < \dots < j_p \leq m$  такие, что  $a_{i_1} = b_{j_1}, \dots, a_{i_p} = b_{j_p}$ .

- **Формат ввода / входного файла (input.txt).**

- Первая строка:  $n$  - длина первой последовательности.
- Вторая строка:  $a_1, a_2, \dots, a_n$  через пробел.
- Третья строка:  $m$  - длина второй последовательности.

4

- Четвертая строка:  $b_1, b_2, \dots, b_m$  через пробел.

- Ограничения:  $1 \leq n, m \leq 100$ ;  $-10^9 < a_i, b_i < 10^9$ .

- **Формат вывода / выходного файла (output.txt).** Выведите число  $p$ .

- Ограничение по времени. 1 сек.

- Примеры:

input.txt	output.txt	input.txt	output.txt	input.txt	output.txt
3	2	1	0	4	2
2 7 5		7		2 7 8 3	
2		4		4	
2 5		1 2 3 4		5 2 8 7	

- В первом примере одна общая подпоследовательность – (2, 5) длиной 2, во втором примере две последовательности не имеют одинаковых элементов. В третьем примере - длина 2, последовательности – (2, 7) или (2, 8).

#### Листинг кода.

```
def longest_common_subsequence(n, m, seq1, seq2):
    table = [[0] * (m + 1) for _ in range(n + 1)]

    for i in range(1, n + 1):
        for j in range(1, m + 1):
            if seq1[i - 1] == seq2[j - 1]:
                table[i][j] = table[i - 1][j - 1] + 1
```

```
        else:
            table[i][j] = max(table[i - 1][j], table[i][j - 1])

    return table[n][m]
```

Текстовое объяснение решения.

Алгоритм основан на методе динамического программирования и использует двумерную таблицу для хранения длин НОП подпоследовательностей. Пусть seq1 и seq2 - две входные последовательности длиной n и m соответственно. Таблица table размера (n+1) x (m+1) инициализируется нулями. Элемент table[i][j] хранит длину НОП для префиксов seq1 длиной i и seq2 длиной j. Рекуррентное соотношение для заполнения таблицы: • Если seq1[i-1] == seq2[j-1], то символы совпадают, и длина НОП увеличивается на 1: table[i][j] = table[i-1][j-1] + 1. • Иначе символы не совпадают, и длина НОП равна максимуму из длин НОП для префиксов: table[i][j] = max(table[i-1][j], table[i][j-1]). Длина НОП для исходных последовательностей хранится в table[n][m].

Результат работы кода на примерах из текста задачи:

Входные данные:

3

2 7 5

2

2 5

Результат:

2

Время работы: 2.4700071662664413e-05

Память: 0.000335693359375 Мб

Process finished with exit code 0

Вывод по задаче:

В ходе работы над задачей мной была изучена работа с динамической таблицей.

## Задача №7. Шаблоны

Текст задачи.

### 7 задача. Шаблоны

Многие операционные системы используют шаблоны для ссылки на группы объектов: файлов, пользователей, и т. д. Ваша задача – реализовать простейший алгоритм проверки шаблонов для имен файлов.

В этой задаче алфавит состоит из маленьких букв английского алфавита и точки («.»). Шаблоны могут содержать произвольные символы алфавита, а также два специальных символа: «?» и «\*». Знак вопроса («?») соответствует ровно одному произвольному символу. Звездочка («\*») соответствует подстроке произвольной длины (возможно, нулевой). Символы алфавита, встречающиеся в шаблоне, отображаются на ровно один такой же символ в проверяемой строке. Строка считается подходящей под шаблон, если символы шаблона можно последовательно отобразить на символы строки таким образом, как описано выше. Например, строки «ab», «aab» и «beda.» подходят под шаблон «\*a?», а строки «bebe», «a» и «ba» – нет.

- **Формат ввода / входного файла (input.txt).** Первая строка входного файла определяет шаблон. Вторая строка  $S$  состоит только из символов алфавита. Ее необходимо проверить на соответствие шаблону. Длины обеих строк не превосходят 10 000. Строки могут быть пустыми – будьте внимательны!
- **Формат вывода / выходного файла (output.txt).** Если данная строка подходит под шаблон, выведите YES. Иначе выведите NO.
- Ограничение по времени. 2 сек.
- Ограничение по памяти. 256 мб.
- Пример:

input.txt	output.txt	input.txt	output.txt
k?t*n	YES	k?t?n	NO
kitten		kitten	

Листинг кода.

```
def match_pattern(pattern, string):
    n, m = len(string), len(pattern)

    table = [[False] * (m + 1) for _ in range(n + 1)]
    table[0][0] = True

    for j in range(1, m + 1):
        if pattern[j - 1] == '*':
            table[0][j] = table[0][j - 1]

    for i in range(1, n + 1):
        for j in range(1, m + 1):
            if pattern[j - 1] == '?' or pattern[j - 1] == string[i - 1]:
                table[i][j] = table[i - 1][j - 1]
            elif pattern[j - 1] == '*':
                table[i][j] = table[i - 1][j] or table[i][j - 1]

    return "YES" if table[n][m] else "NO"
```

Текстовое объяснение решения.

Алгоритм использует булеву таблицу `table` размера  $(n+1) \times (m+1)$ , где  $n$  – длина строки, а  $m$  – длина образца. Элемент `table[i][j]` равен `True`, если первые  $i$  символов строки соответствуют первым  $j$  символам образца, и `False` в противном случае.

Результат работы кода на примерах из текста задачи:

```
Входные данные:
```

```
k?t*n
```

```
kitten
```

```
Результат:
```

```
YES
```

```
Время работы: 5.019991658627987e-05
```

```
Память: 0.0005950927734375 M6
```

```
Входные данные:
```

```
12
```

```
add 911 police
```

```
add 76213 Mom
```

```
add 17239 Bob
```

```
find 76213
```

```
find 910
```

```
find 911
```

```
del 910
```

```
del 911
```

```
find 911
```

```
find 76213
```

```
add 76213 daddy
```

```
find 76213
```

```
Результат:
```

```
Mom
```

```
not found
```

```
police
```

```
not found
```

```
Mom
```

```
daddy
```

```
Время работы: 5.91999851167202e-05
```

```
Память: 0.0007581710815429688 M6
```

Вывод по задаче:

В ходе работы над задачей мной была изучена работа с динамической таблицей.

## Дополнительные задачи

### Задача №1. Обмен монет

Текст задачи.

#### 1 задача. Обмен монет

Как мы уже поняли из лекции, не всегда "жадное" решение задачи на обмен монет работает корректно для разных наборов номиналов монет. Например, если доступны номиналы 1, 3 и 4, жадный алгоритм поменяет 6 центов, используя три монеты (4 + 1 + 1), в то время как его можно изменить, используя всего две монеты (3 + 3). Теперь ваша цель - применить динамическое программирование для решения задачи про обмен монет для разных номиналов.

- **Формат ввода / входного файла (input.txt).** Целое число  $money$  ( $1 \leq money \leq 10^3$ ). Набор монет: количество возможных монет  $k$  и сам набор  $coins = \{coin_1, \dots, coin_k\}$ .  $1 \leq k \leq 100$ ,  $1 \leq coin_i \leq 10^3$ . Проверку можно сделать на наборе {1, 3, 4}. Формат ввода: первая строка содержит через пробел  $money$  и  $k$ ; вторая -  $coin_1 coin_2 \dots coin_k$ .

– **Вариация 2:** Количество монет в кассе ограничено. Для каждой монеты из набора  $coins = \{coin_1, \dots, coin_k\}$  есть соответствующее целое число - количество монет в кассе данного номинала  $c = \{c_1, \dots, c_k\}$ . Если они закончились, то выдать данную монету невозможно.

- **Формат вывода / выходного файла (output.txt).** Вывести одно число – минимальное количество необходимых монет для размена  $money$  доступным набором монет  $coins$ .
- Ограничение по времени. 1 сек.
- Примеры:

input.txt	output.txt	input.txt	output.txt
2 3 1 3 4	2	34 3 1 3 4	9

Листинг кода.

```
def coins_function(money, coins):
    table = [float('inf')] * (money + 1)
    table[0] = 0
    for coin in coins:
        for x in range(coin, money + 1):
            table[x] = min(table[x], table[x - coin] + 1)
    return table[money] if table[money] != float('inf') else -1
```

Текстовое объяснение решения.

Алгоритм использует одномерный массив `table` длиной `money + 1`, где `money` – целевая сумма денег. Элемент `table[i]` хранит минимальное количество монет, необходимых для составления суммы `i`. Массив инициализируется значением бесконечности (`float('inf')`), за исключением `table[0] = 0` (нулевая сумма не требует монет). Алгоритм перебирает все номиналы монет (`coins`). Для каждого номинала `coin` он итерируется по суммам от `coin` до `money`. Для каждой суммы `x` он обновляет значение `table[x]`, выбирая минимум между текущим значением `table[x]` и значением `table[x - coin] + 1` (количество монет для суммы `x - coin` плюс одна монета номинала `coin`). После перебора всех монет, `table[money]` содержит минимальное количество монет для составления целевой суммы. Если `table[money]` остаётся равным бесконечности, то целевую сумму составить невозможно.

Результат работы кода на примерах из текста задачи:

```
Входные данные:
```

```
34 3
```

```
1 3 4
```

```
Результат:
```

```
9
```

```
Время работы: 6.560003384947777e-05
```

```
Память: 0.00038909912109375 Мб
```

Вывод по задаче:

В ходе работы над задачей мной была изучена работа с динамической таблицей.

Задача №5. Наибольшая общая подпоследовательность трех последовательностей

Текст задачи.



## 5 задача. Наибольшая общая подпоследовательность трех последовательностей

Вычислить длину самой длинной общей подпоследовательности из *трех* последовательностей.

Даны три последовательности  $A = (a_1, a_2, \dots, a_n)$ ,  $B = (b_1, b_2, \dots, b_m)$  и  $C = (c_1, c_2, \dots, c_l)$ , найти длину их самой длинной общей подпоследовательности, т.е. наибольшее неотрицательное целое число  $p$  такое, что существуют индексы  $1 \leq i_1 < i_2 < \dots < i_p \leq n$ ,  $1 \leq j_1 < j_2 < \dots < j_p \leq m$  и  $1 \leq k_1 < k_2 < \dots < k_p \leq l$  такие, что  $a_{i_1} = b_{j_1} = c_{k_1}, \dots, a_{i_p} = b_{j_p} = c_{k_p}$ .

- **Формат ввода / входного файла (input.txt).**
  - Первая строка:  $n$  - длина первой последовательности.
  - Вторая строка:  $a_1, a_2, \dots, a_n$  через пробел.
  - Третья строка:  $m$  - длина второй последовательности.
  - Четвертая строка:  $b_1, b_2, \dots, b_m$  через пробел.
  - Пятая строка:  $l$  - длина второй последовательности.
  - Шестая строка:  $c_1, c_2, \dots, c_l$  через пробел.
- Ограничения:  $1 \leq n, m, l \leq 100$ ;  $-10^9 < a_i, b_i, c_i < 10^9$ .
- **Формат вывода / выходного файла (output.txt).** Выведите число  $p$ .
- Ограничение по времени. 1 сек.

Листинг кода.

```
def longest_common_subsequence(n, a, m, b, l, c):
    table = [[[0] * (l + 1) for _ in range(m + 1)] for __ in range(n + 1)]

    for i in range(1, n + 1):
        for j in range(1, m + 1):
            for k in range(1, l + 1):
                if a[i - 1] == b[j - 1] == c[k - 1]:
                    table[i][j][k] = table[i - 1][j - 1][k - 1] + 1
                else:
                    table[i][j][k] = max(table[i - 1][j][k], table[i][j - 1][k], table[i][j][k - 1])

    return table[n][m][l]
```

Текстовое объяснение решения.

Алгоритм использует трёхмерную таблицу `table` размера  $(n+1) \times (m+1) \times (l+1)$ , где  $n$ ,  $m$ , и  $l$  – длины трёх входных последовательностей  $a$ ,  $b$  и  $c$  соответственно. Элемент `table[i][j][k]` хранит длину НОП для префиксов последовательностей  $a$  (длиной  $i$ ),  $b$  (длиной  $j$ ) и  $c$  (длиной  $k$ ). Таблица инициализируется нулями. Алгоритм заполняет таблицу итеративно, используя следующее рекуррентное соотношение:

- Если  $a[i-1] == b[j-1] == c[k-1]$ : Символы на текущих позициях во всех трёх последовательностях совпадают. Длина НОП увеличивается на 1:  $table[i][j][k] = table[i-1][j-1][k-1] + 1$ .
- Иначе: Символы не совпадают. Длина НОП – это максимум из длин НОП для префиксов, полученных исключением последнего элемента из одной из

последовательностей:  $table[i][j][k] = \max(table[i-1][j][k], table[i][j-1][k], table[i][j][k-1])$ . После заполнения всей таблицы, значение  $table[n][m][l]$  содержит длину НОП для исходных трёх последовательностей.

Результат работы кода на примерах из текста задачи:

```
Входные данные:
```

```
3
```

```
1 2 3
```

```
3
```

```
2 1 3
```

```
3
```

```
1 3 5
```

```
Результат:
```

```
2
```

```
Время работы: 8.79999715834856e-05
```

```
Память: 0.00106048583984375 Мб
```

Вывод по задаче:

В ходе работы над задачей мной была изучена работа с динамической таблицей.

## Вывод

В ходе лабораторной работы был изучен метод работы с динамическим программированием.