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

Выполнила:

Просветова Валерия

К3141

Проверил:

Афанасьев.А.В.

Санкт-Петербург

2024 г.

# Содержание отчета

Задача №1: Обмен монет

Задача №4: Наибольшая общая подпоследовательность двух последовательностей

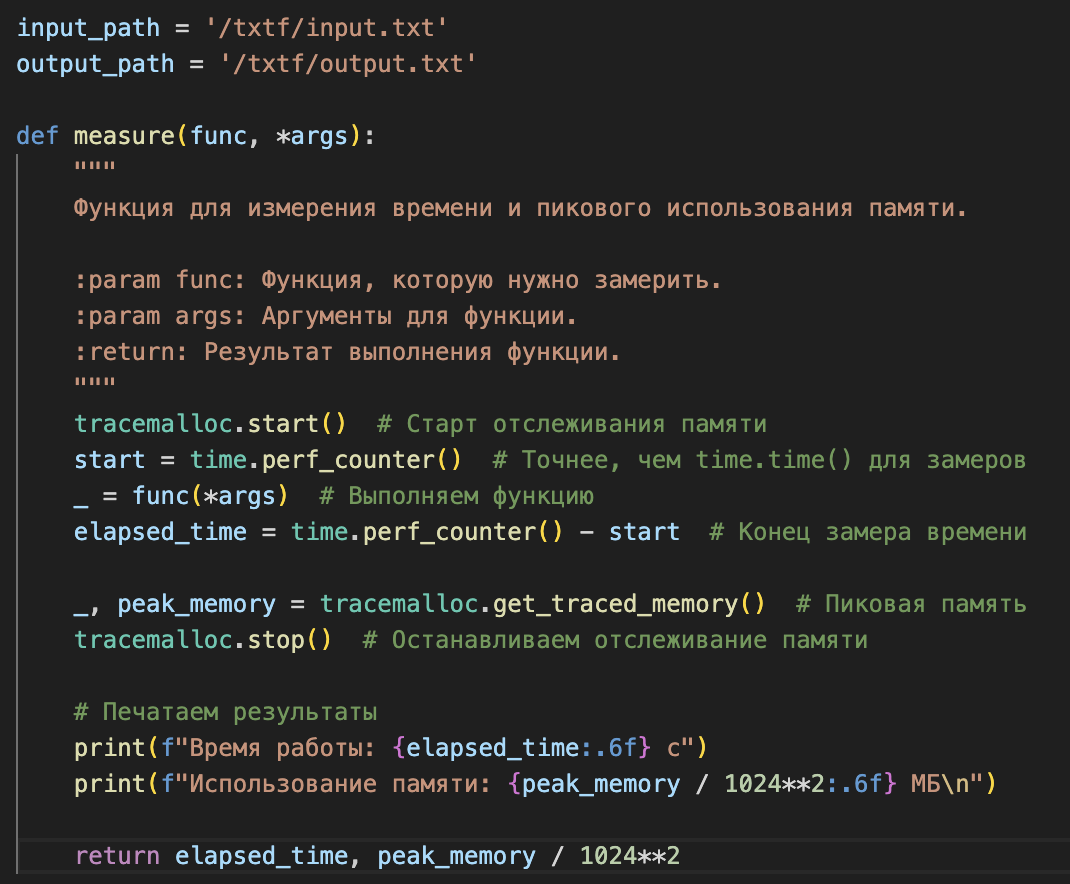
Задача №5: Наибольшая общая подпоследовательность трех последовательностей

Задача №6: Наибольшая возрастающая подпоследовательность

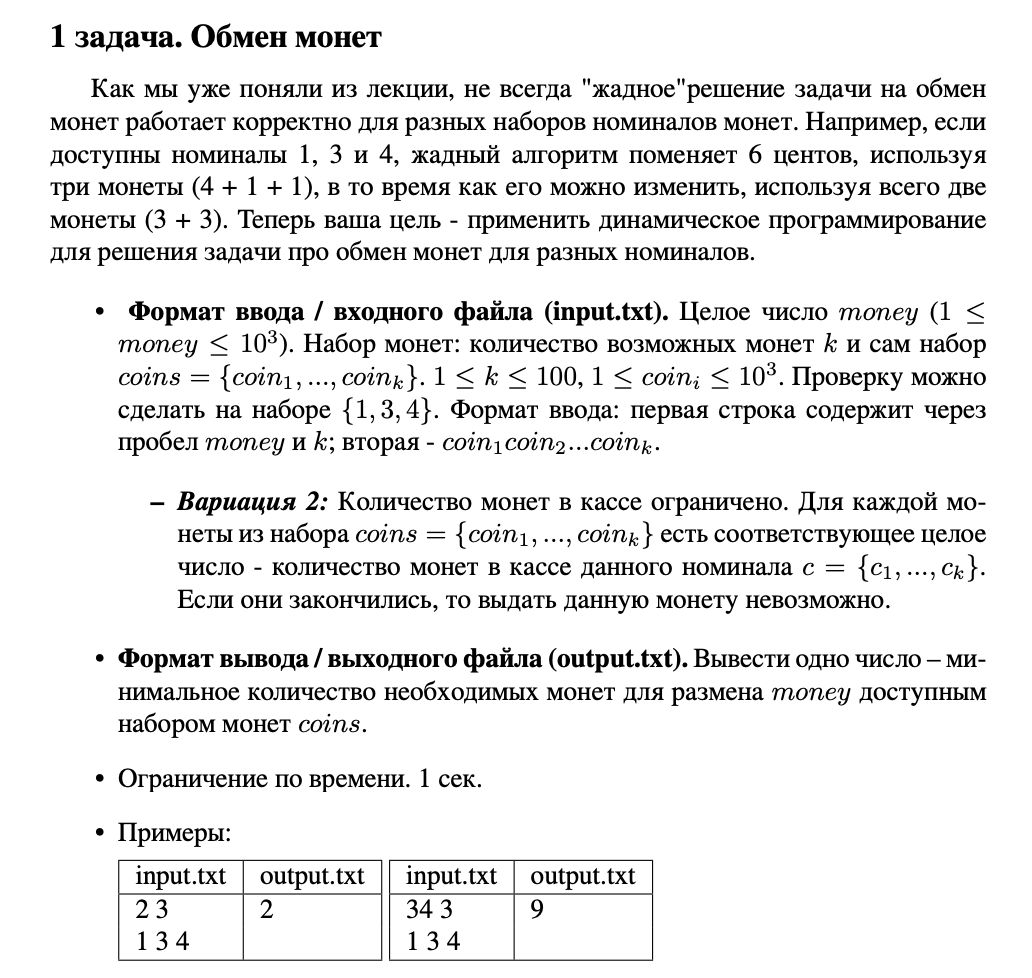
Вывод

Использовала файл utils.py для исключения повторения строчек кода:

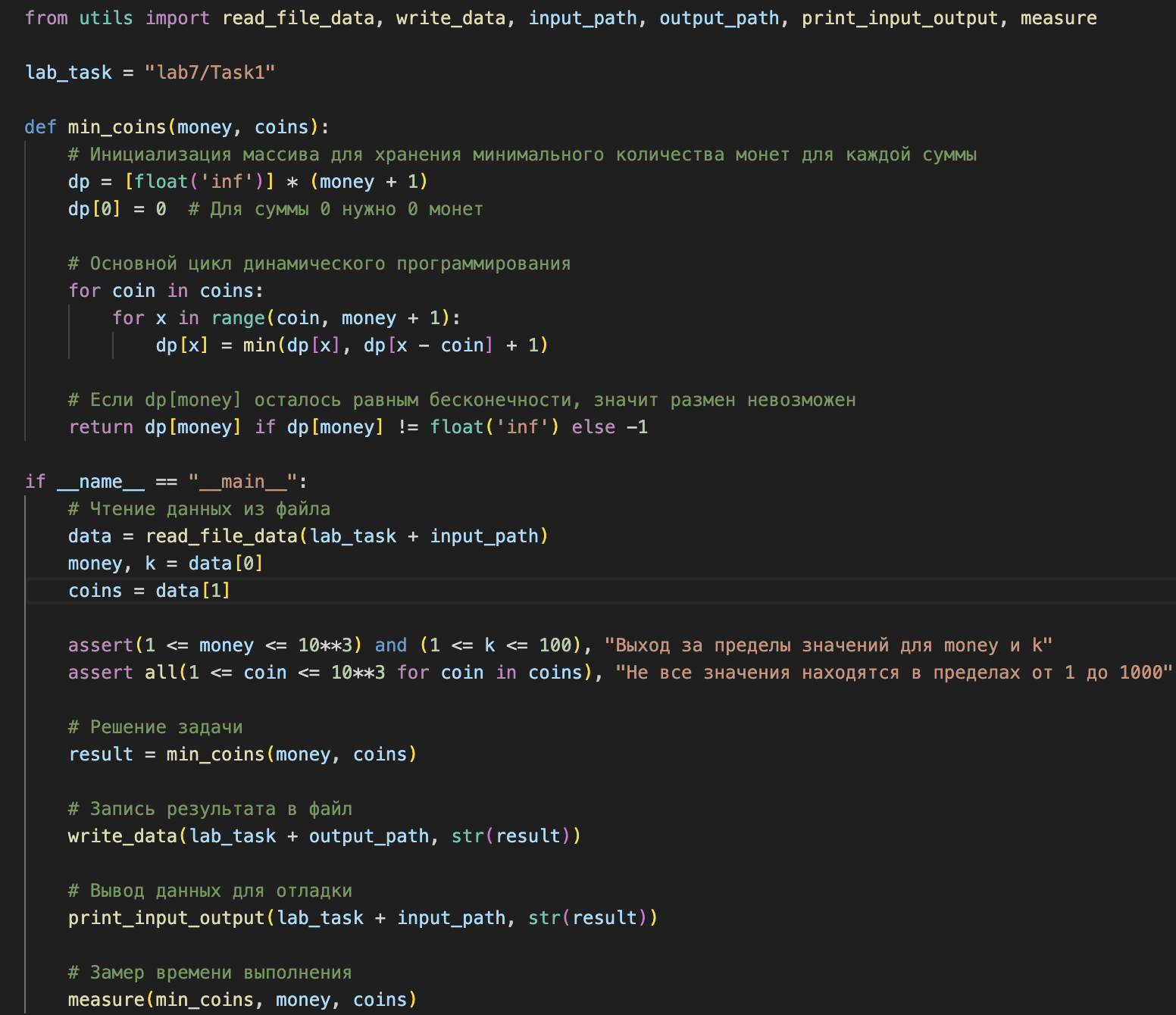




Задача №1:



Решение:



Этот код представляет собой функцию min\_coins, которая принимает два параметра:

* money: Сумма денег, которую нужно разменять.
* coins: Список номиналов монет.

Функция возвращает минимальное количество монет, необходимое для размена суммы money.

**Шаг 1:**

Первой строкой кода является инициализация массива dp с помощью списка из money + 1 элементов, каждый из которых равен float('inf'). Этот массив будет хранить минимальное количество монет для каждой суммы от 0 до money.

**Шаг 2:**

Далее инициализируется базовый случай, когда сумма равна 0. В этом случае нужно 0 монет, поэтому dp[0] устанавливается в 0.

**Шаг 3:**

В цикле for функция обрабатывает каждый номинал монеты coin из списка coins.

**Шаг 4:**

Внутри цикла for функция обрабатывает каждую сумму x от coin до money. Для каждой суммы x функция проверяет, можно ли получить эту сумму, используя монету coin.

**Шаг 5:**

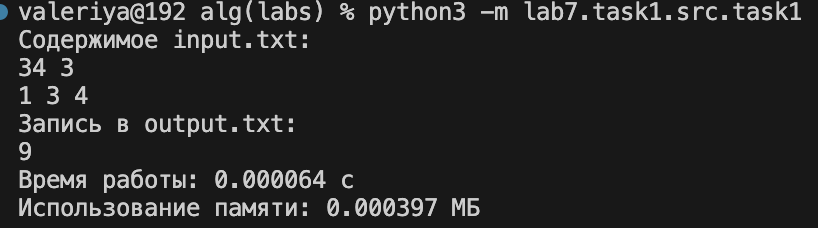
Если сумма x можно получить, используя монету coin, функция обновляет значение dp[x] минимальным количеством монет, необходимым для получения суммы x. Это делается с помощью выражения min(dp[x], dp[x - coin] + 1), которое выбирает минимальное значение между текущим значением dp[x] и значением dp[x - coin] + 1, где dp[x - coin] - минимальное количество монет, необходимое для получения суммы x - coin.

**Шаг 6:**

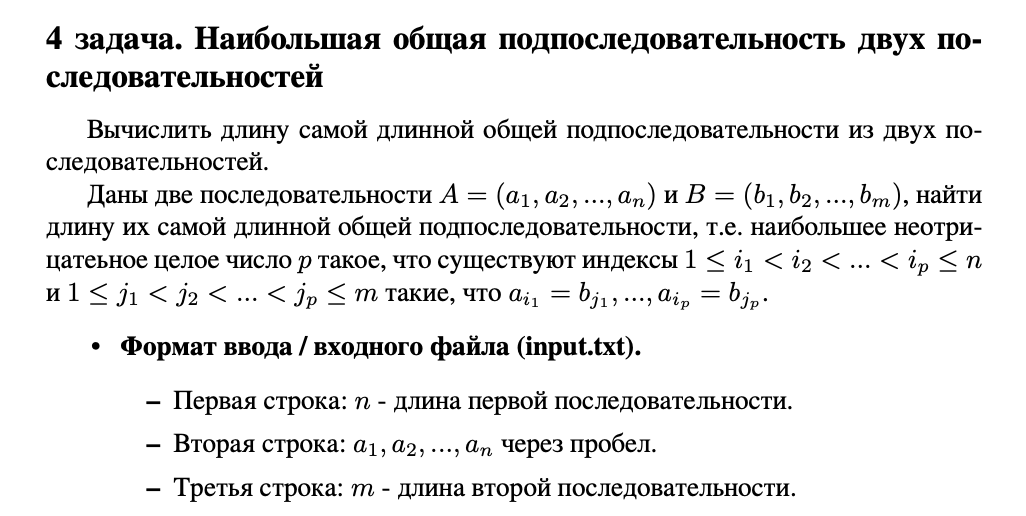
После завершения цикла for функция возвращает значение dp[money], которое представляет минимальное количество монет, необходимое для размена суммы money. Если dp[money] осталось равным float('inf'), это значит, что размен невозможен, и функция возвращает -1.

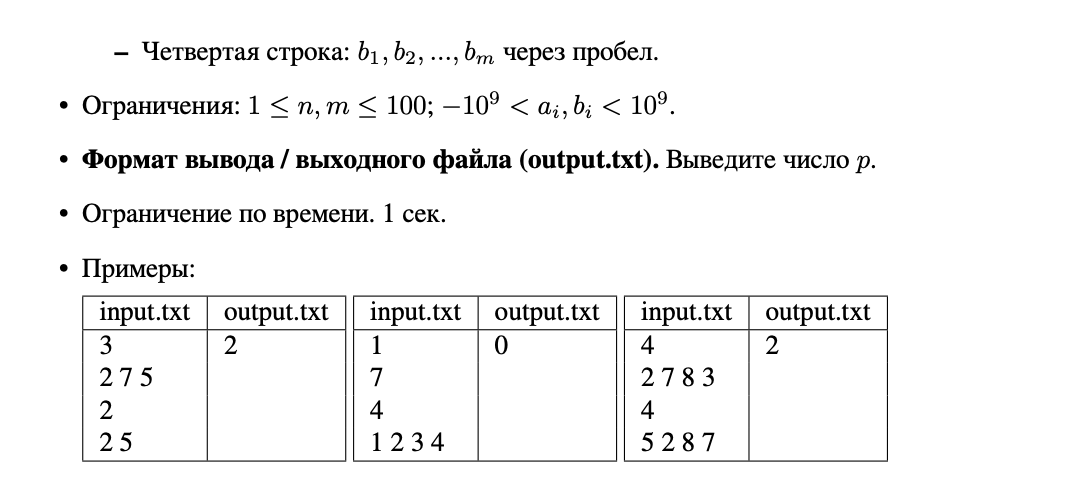
**Шаг 7:**

Считывание данных из input.txt и запись результата в output.txt.

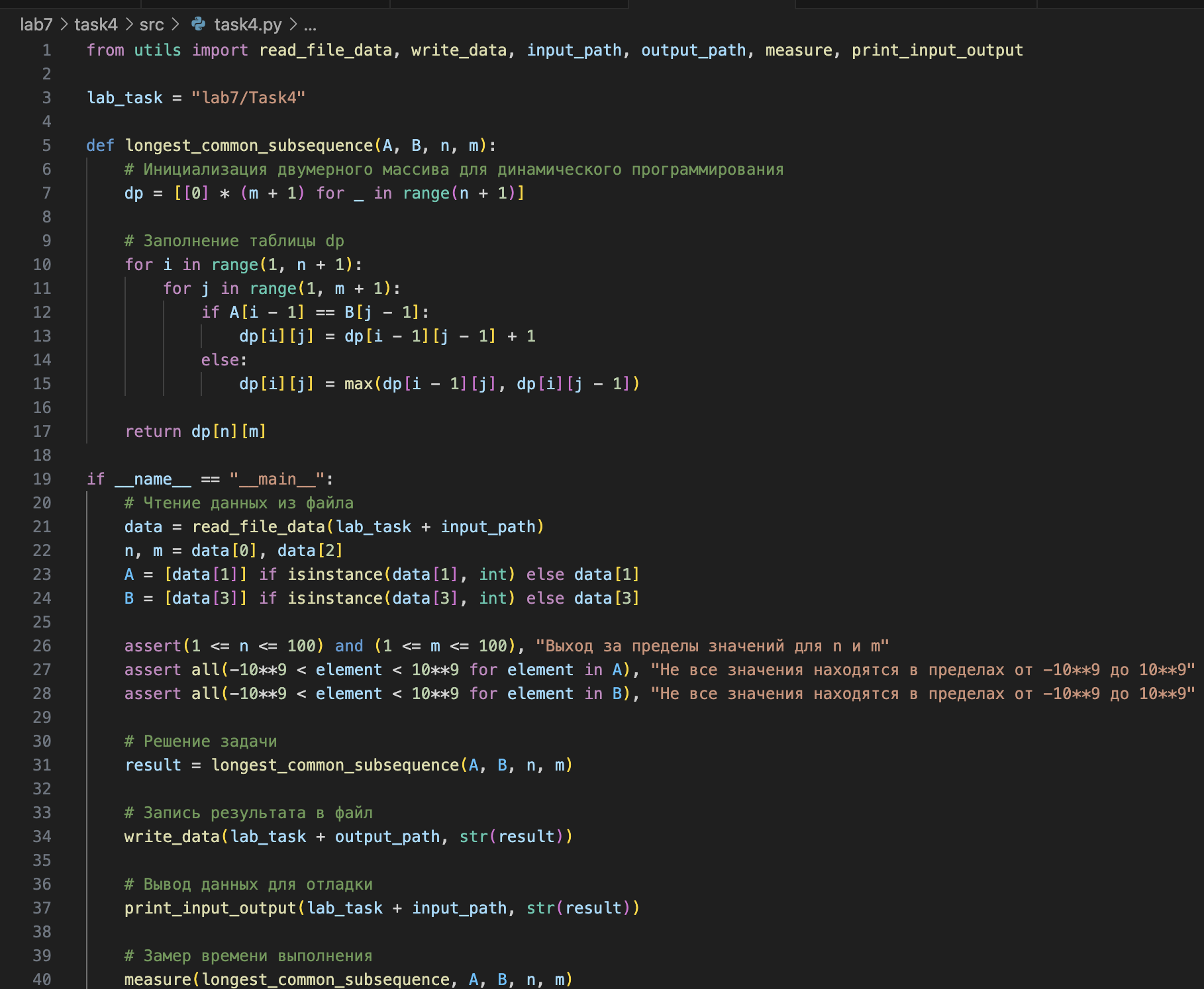


Задача №4





Решение:



Этот код представляет собой функцию longest\_common\_subsequence, которая принимает четыре параметра:

* A и B: Две строки, для которых нужно найти наибольшую общую подпоследовательность.
* n и m: Длины строк A и B соответственно.

Функция возвращает длину наибольшей общей подпоследовательности строк A и B.

**Шаг 1:**

Первой строкой кода является инициализация двумерного массива dp с помощью списка из n + 1 строк и m + 1 столбцов, каждый из которых заполнен нулями. Этот массив будет использоваться для хранения длины наибольшей общей подпоследовательности для каждой пары подстрок A и B.

**Шаг 2:**

В цикле for функция обрабатывает каждую пару подстрок A и B длины от 1 до n и от 1 до m соответственно.

**Шаг 3:**

Для каждой пары подстрок функция сравнивает текущие символы A[i - 1] и B[j - 1]. Если символы совпадают, функция увеличивает значение dp[i][j] на 1, добавляя 1 к длине наибольшей общей подпоследовательности для предыдущих подстрок A и B.

**Шаг 4:**

Если символы не совпадают, функция обновляет значение dp[i][j] максимальным значением между dp[i - 1][j] и dp[i][j - 1], что представляет длину наибольшей общей подпоследовательности для предыдущих подстрок A и B.

**Шаг 5:**

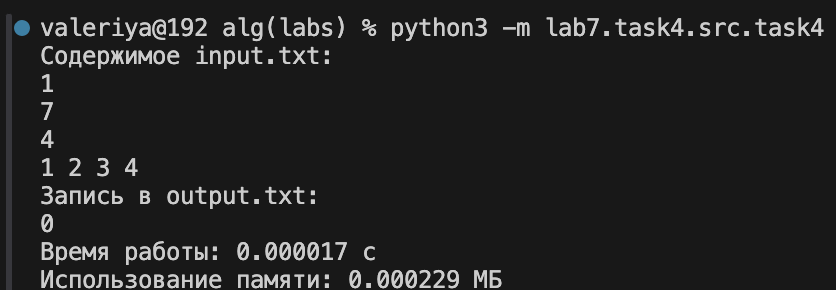
После завершения цикла for функция возвращает значение dp[n][m], которое представляет длину наибольшей общей подпоследовательности строк A и B.

**Шаг 6:**

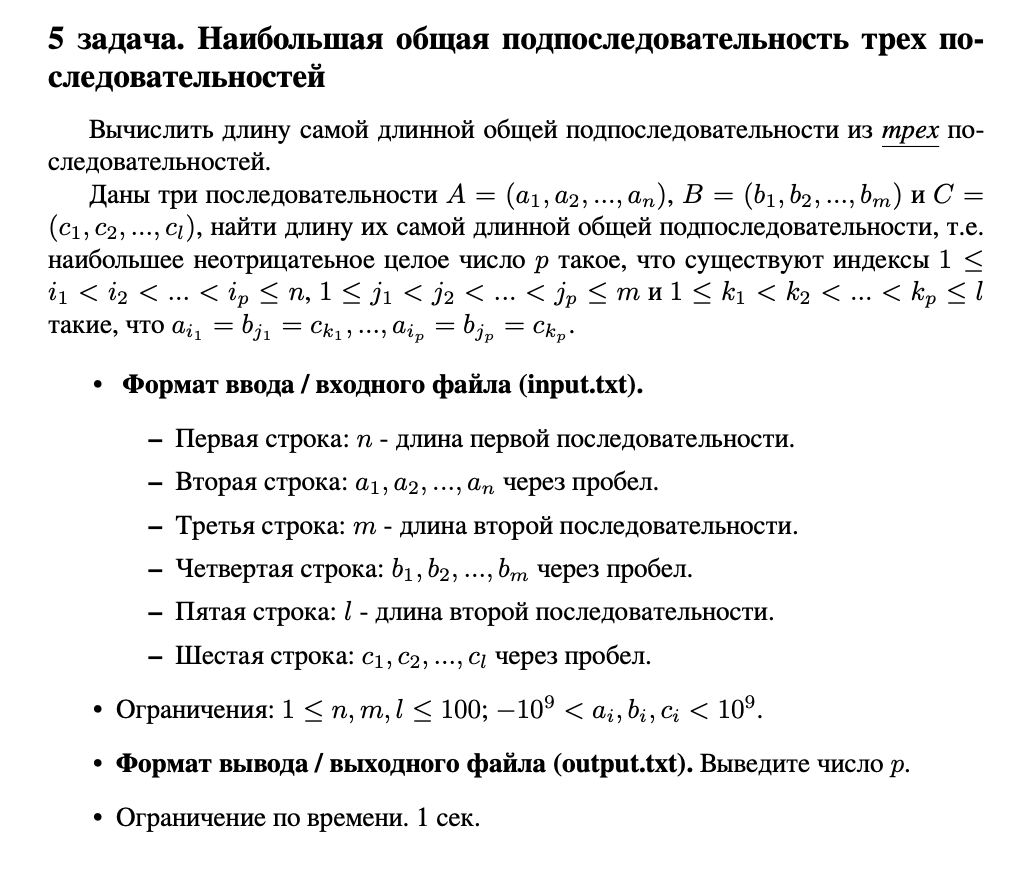
Считывание данных из input.txt и запись результата в output.txt.

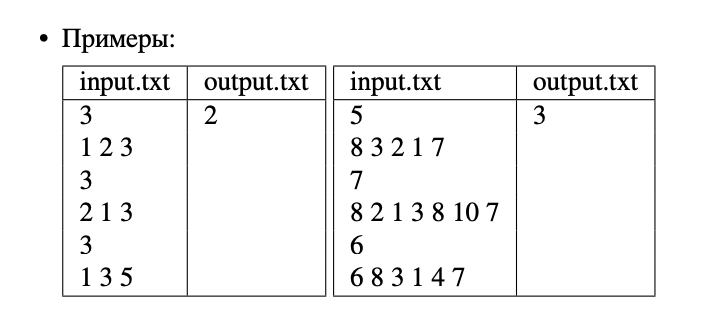
**Шаг 7:**

Считывание данных из input.txt и запись результата в output.txt.

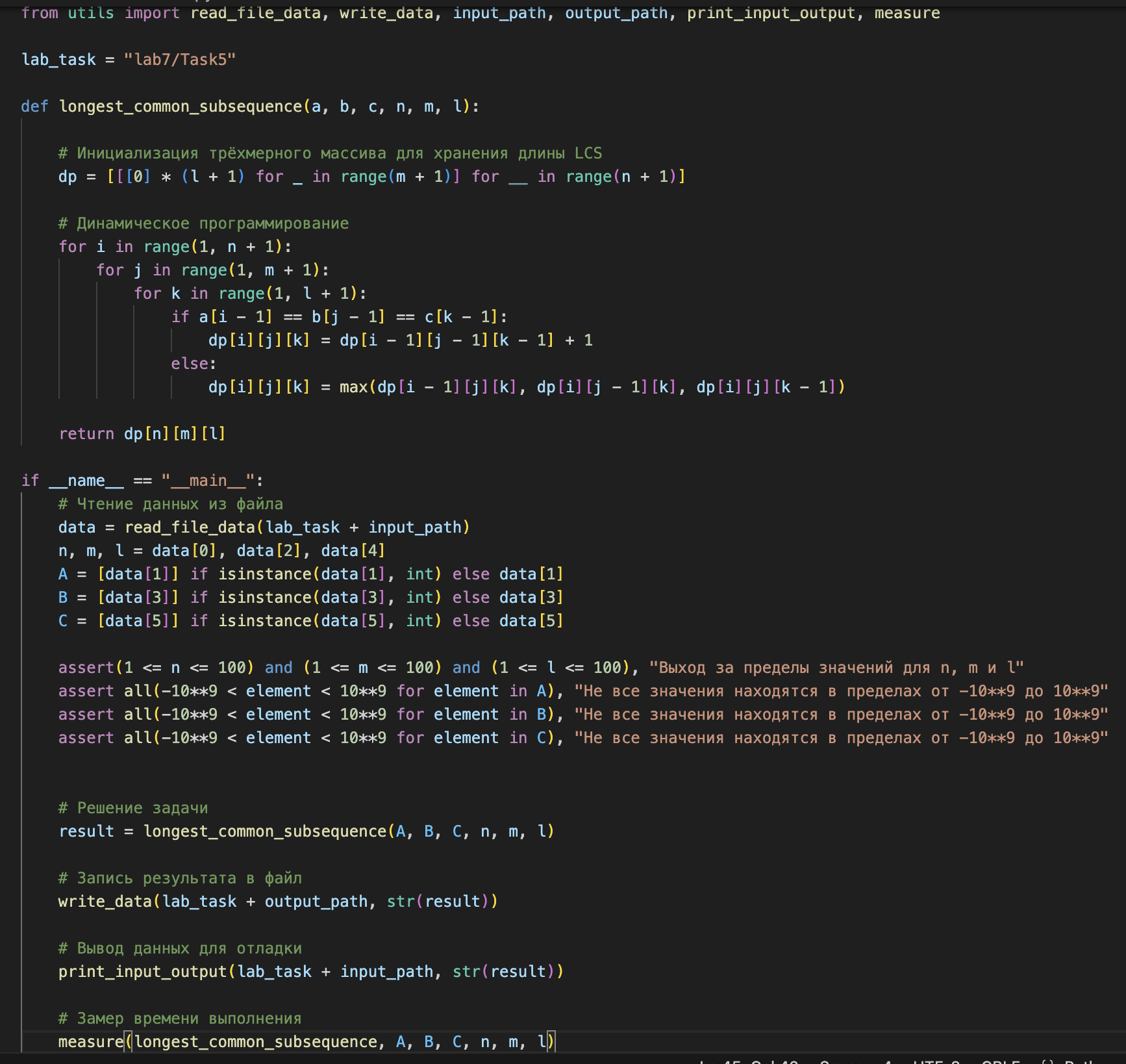


Задача №5:





Решение:

  
Этот код представляет собой функцию longest\_common\_subsequence, которая принимает шесть параметров:

* a, b и c: Три строки, для которых нужно найти наибольшую общую подпоследовательность.
* n, m и l: Длины строк a, b и c соответственно.

Функция возвращает длину наибольшей общей подпоследовательности строк a, b и c.

**Шаг 1:**

Первой строкой кода является инициализация трёхмерного массива dp с помощью списка из n + 1 строк, m + 1 столбцов и l + 1 глубин, каждый из которых заполнен нулями. Этот массив будет использоваться для хранения длины наибольшей общей подпоследовательности для каждой тройки подстрок a, b и c.

**Шаг 2:**

В цикле for функция обрабатывает каждую тройку подстрок a, b и c длины от 1 до n, от 1 до m и от 1 до l соответственно.

**Шаг 3:**

Для каждой тройки подстрок функция сравнивает текущие символы a[i - 1], b[j - 1] и c[k - 1]. Если символы совпадают, функция увеличивает значение dp[i][j][k] на 1, добавляя 1 к длине наибольшей общей подпоследовательности для предыдущих подстрок a, b и c.

**Шаг 4:**

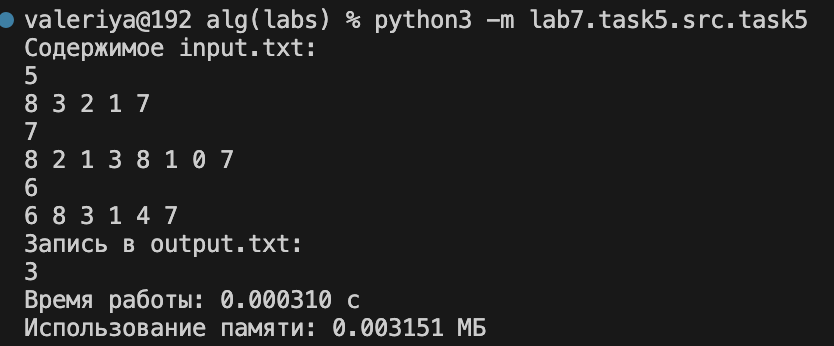
Если символы не совпадают, функция обновляет значение dp[i][j][k] максимальным значением между dp[i - 1][j][k], dp[i][j - 1][k] и dp[i][j][k - 1], что представляет длину наибольшей общей подпоследовательности для предыдущих подстрок a, b и c.

**Шаг 5:**

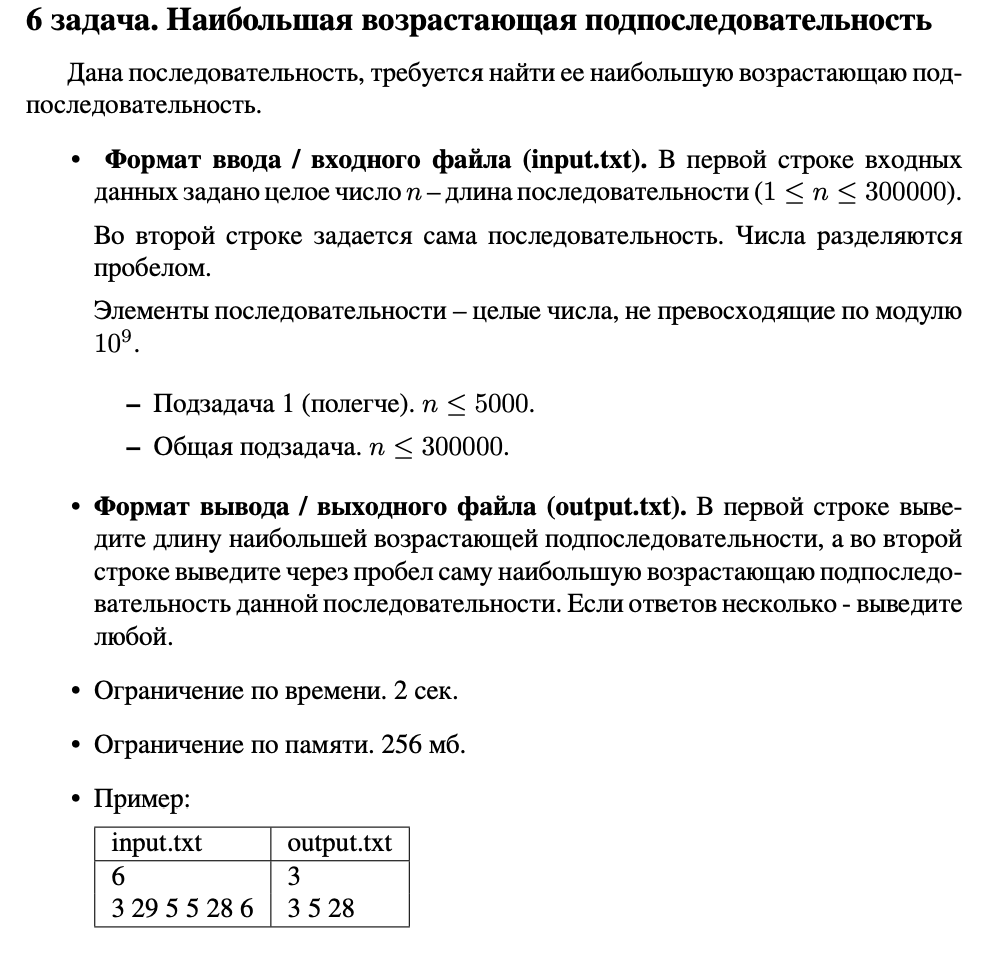
После завершения цикла for функция возвращает значение dp[n][m][l], которое представляет длину наибольшей общей подпоследовательности строк a, b и c.

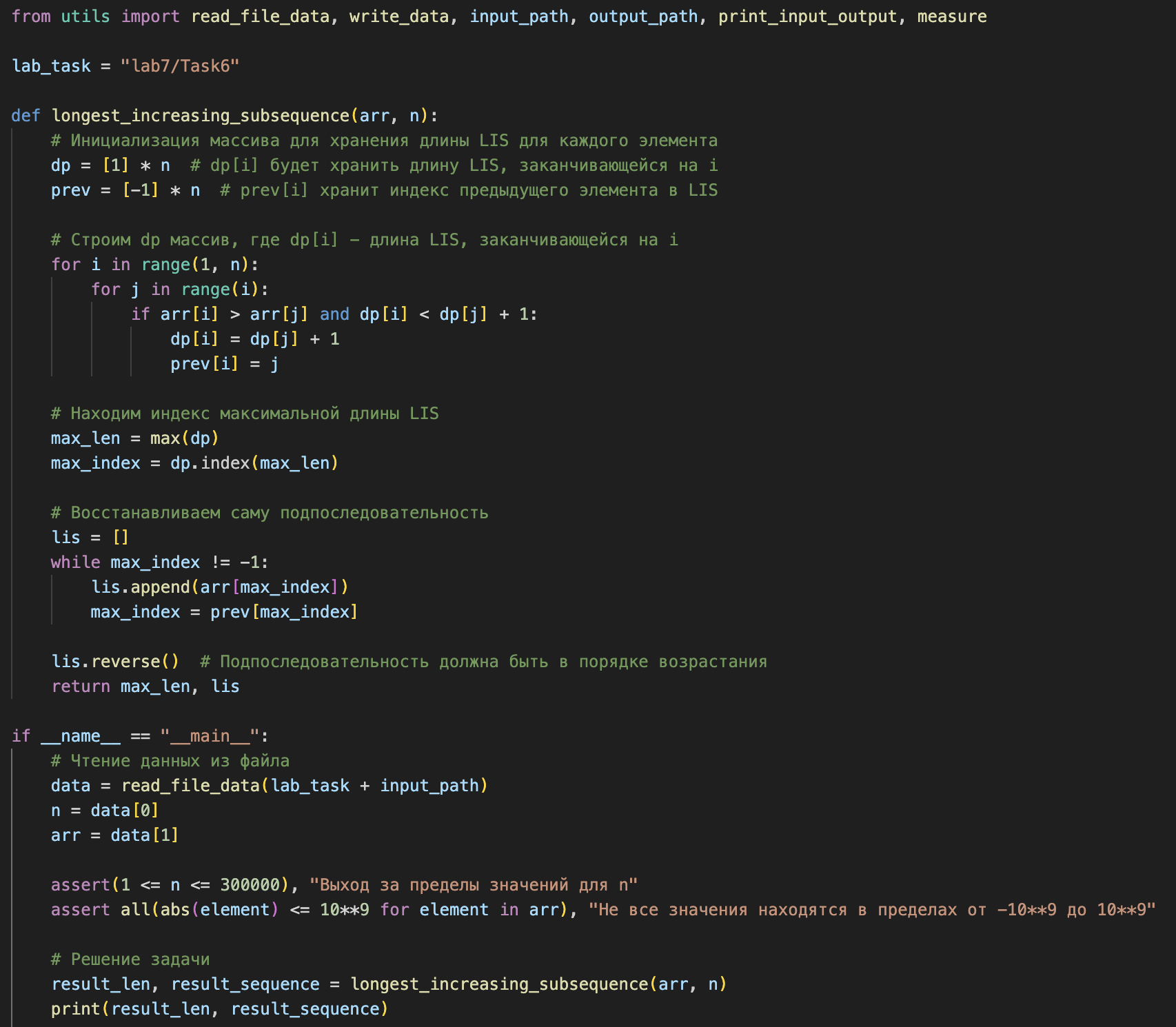
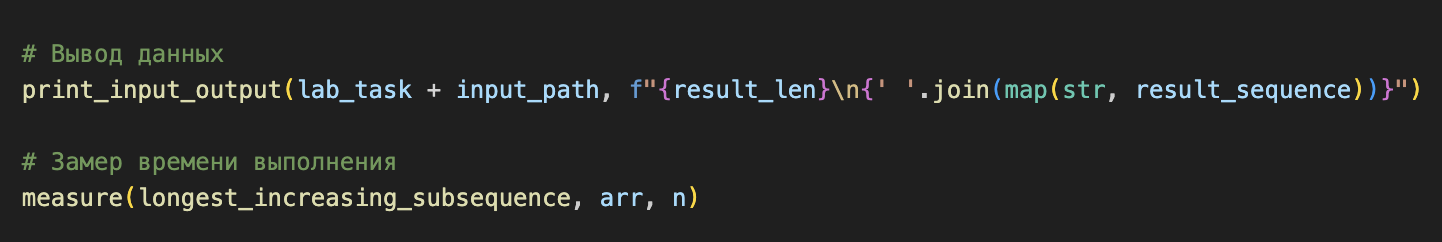
**Шаг 6:**

Считывание данных из input.txt и запись результата в output.txt.



Задача №6



Решение:

Этот код представляет собой функцию longest\_increasing\_subsequence, которая принимает два параметра:

* arr: Массив целых чисел.
* n: Длина массива arr.

Функция возвращает длину наибольшей возрастающей подпоследовательности (LIS) в массиве arr и саму подпоследовательность.

**Шаг 1:**

Первой строкой кода является инициализация двух массивов: dp и prev. Массив dp будет хранить длину LIS, заканчивающейся на каждом элементе массива arr, а массив prev будет хранить индекс предыдущего элемента в LIS.

**Шаг 2:**

В цикле for функция обрабатывает каждый элемент массива arr, начиная со второго элемента (индекс 1). Для каждого элемента arr[i] функция проверяет все предыдущие элементы arr[j] (индекс j от 0 до i-1). Если arr[i] больше arr[j] и длина LIS, заканчивающейся на arr[i], меньше длины LIS, заканчивающейся на arr[j], плюс 1, функция обновляет значение dp[i] и устанавливает индекс предыдущего элемента в LIS как j.

**Шаг 3:**

После построения массива dp функция находит индекс максимальной длины LIS с помощью функции max и метода index.

**Шаг 4:**

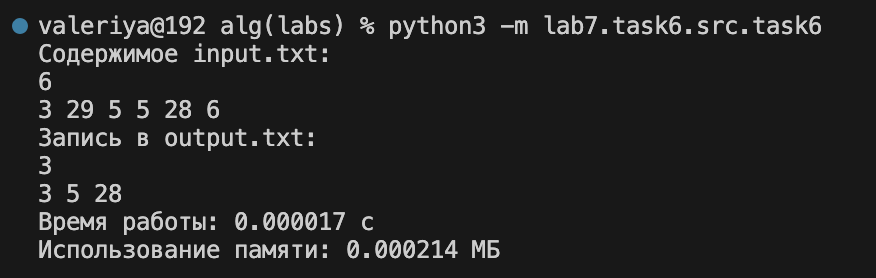
Функция восстанавливает саму подпоследовательность, начиная с индекса максимальной длины LIS и перемещаясь назад по массиву prev. Подпоследовательность хранится в списке lis.

**Шаг 5:**

После восстановления подпоследовательности функция возвращает длину наибольшей возрастающей подпоследовательности и саму подпоследовательность.

**Шаг 6:**

Считывание данных из input.txt и запись результата в output.txt.



Вывод:

В данной лабораторной работе для решения задач я применила динамическое программирование. Реализовала все задачи, которые передо мной поставлены.