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: Хеширование. Хеш-таблицы.

Вариант 12

Выполнила:

Мкртчян.К.Г.

К3141

Проверил:

Афанасьев А.В.

Санкт-Петербург

2024 г.

Оглавление

[**Задачи по варианту** 3](#_Toc185627251)

[Задача №1. Множество. [N баллов] 3](#_Toc185627252)

[**Дополнительные задачи** 5](#_Toc185627253)

[Задача №2. Телефонная книга [N баллов] 5](#_Toc185627254)

[Задача №5. Выборы в США. [N баллов] 8](#_Toc185627255)

[Задача №6. Фибоначчи возвращается. [N баллов] 10](#_Toc185627256)

[**Вывод по лабораторной работе** 12](#_Toc185627257)

# **Задачи по варианту**

## **Задача №1. Обмен монет. [N баллов]**

Текст задачи:

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 ≤ money ≤ 10^3). Набор монет: количество возможных монет k и сам набор coins = {coin1, ..., coink}. 1 ≤ k ≤ 100, 1 ≤ coini ≤ 10^3. Проверку можно сделать на наборе {1, 3, 4}. Формат ввода: первая строка содержит через пробел money и k; вторая - coin1coin2...coink.

– Вариация 2: Количество монет в кассе ограничено. Для каждой монеты из набора coins = {coin1, ..., coink} есть соответствующее целое число - количество монет в кассе данного номинала c = {c1, ..., ck}. 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.

Решение:

def exchange(money: int, coins: list[int]) -> int:  
 dp = [0] + [float('inf')] \* money  
 for coin in coins:  
 for j in range(coin, money+1):  
 dp[j] = min(dp[j], dp[j - coin] + 1)  
 return dp[money]  
  
  
def exchange\_with\_amount(money: int, coins: dict[int: list[int]]) -> int:  
 dp = [0] + [float('inf')] \* money  
 for c in coins.keys():  
 denomination = c  
 amount = coins[c]  
 if amount > 0:  
 for j in range(denomination, money+1):  
 dp[j] = min(dp[j], dp[j - denomination] + 1)  
 return dp[money]

Объяснение решения:

Функция exchange предназначена для решения задачи о минимальном количестве монет, необходимых для того, чтобы составить заданную сумму денег. Она использует метод динамического программирования для достижения этой цели. Вот основные шаги ее работы:

Инициализация массива: Функция начинает с создания массива dp, где каждый элемент представляет наименьшее количество монет, необходимое для составления определенной суммы. Первым элементом массива устанавливается 0 (поскольку для суммы 0 не требуется ни одной монеты), а все остальные элементы инициализируются значением infinity, чтобы обозначить невозможные комбинации на начальном этапе.

- Внешний цикл по монетам: Функция перебирает каждую монету из входящего списка coins. Для каждой монеты выполняется внутренний цикл.

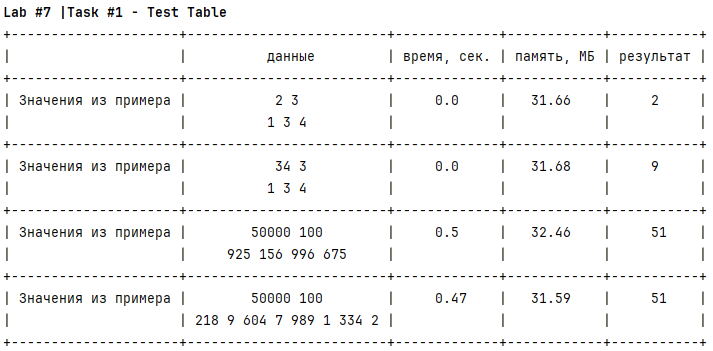
- Внутренний цикл по суммам: Внутренний цикл перебирает все суммы от значения текущей монеты до целевой суммы money. Этот цикл проверяет, можно ли улучшить текущее количество монет для суммы j, добавив одну текущую монету coin.

- Обновление массива dp: Внутри внутреннего цикла функция обновляет значение dp[j] как минимум между текущим значением dp[j] и значением dp[j - coin] + 1. Это означает, что если можно составить сумму j - coin, то при добавлении одной монеты coin количество монет для составления суммы j может быть уменьшено.

- Возврат результата: после завершения всех итераций возвращается значение dp[money], которое содержит минимальное количество монет для составления суммы money. Если это значение по-прежнему равно infinity, это означает, что невозможно составить указанную сумму с использованием данных монет.

- Таким образом, функция эффективно находит оптимальное решение для задачи обмена монет путем последовательного анализа возможностей составления каждой промежуточной суммы.

Время выполнения программы, затраты памяти и результат:



Вывод по задаче: Мы научились применять динамическое программирование для оптимизации решений, разбивая сложные задачи на более простые подзадачи.

## **Задача №5. НОП трех последовательностей. [N баллов]**

Текст задачи:

Вычислить длину самой длинной общей подпоследовательности из трех последовательностей. Даны три последовательности A = (a1, a2, ..., an), B = (b1, b2, ..., bm) и C = (c1, c2, ..., cl), найти длину их самой длинной общей подпоследовательности, т.е. наибольшее неотрицатеьное целое число p такое, что существуют индексы 1 ≤ i1 < i2 < ... < ip ≤ n, 1 ≤ j1 < j2 < ... < jp ≤ m и 1 ≤ k1 < k2 < ... < kp ≤ l такие, что ai1 = bj1 = ck1 , ..., aip = bjp = ckp .

• Формат ввода / входного файла (input.txt). – Первая строка: n - длина первой последовательности. – Вторая строка: a1, a2, ..., an через пробел. – Третья строка: m - длина второй последовательности. – Четвертая строка: b1, b2, ..., bm через пробел. – Пятая строка: l - длина второй последовательности. – Шестая строка: c1, c2, ..., cl через пробел.

• Ограничения: 1 ≤ n, m, l ≤ 100; −10^9 < ai , bi , ci < 10^9 .

• Формат вывода / выходного файла (output.txt). Выведите число p.

• Ограничение по времени. 1 сек.

Решение:

def longest\_common\_sub(n: int, arr\_a: list[int], m: int, arr\_b: list[int], l: int, arr\_c: list[int]) -> int:  
 matrix = [[[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):  
 matrix[i][j][k] = max(matrix[i - 1][j][k], matrix[i][j - 1][k], matrix[i][j][k - 1])  
 if arr\_a[i - 1] == arr\_b[j - 1] == arr\_c[k - 1]:  
 matrix[i][j][k] = max(matrix[i][j][k], matrix[i - 1][j - 1][k - 1] + 1)  
 return matrix[n][m][l]

Объяснение решения:

Функция longest\_common\_sub предназначена для поиска длины наибольшей общей подпоследовательности (НОП) трех последовательностей чисел. Она применяет метод динамического программирования для эффективного решения данной задачи. Рассмотрим этапы работы функции без примеров:

 Инициализация трехмерной матрицы:

 В начале создается трехмерный массив matrix, размером (n + 1) x (m + 1) x (l + 1), где n, m, и l - длины трех входных массивов. Этот массив будет хранить длины НОП для всех возможных комбинаций элементов этих массивов. Все элементы инициализируются нулем, что означает, что длина НГП для любых пустых последовательностей равна нулю.

 Тройной цикл по индексам:

 Функция использует три вложенных цикла для перебора всех возможных индексов i, j, и k, которые соответствуют элементам входных массивов arr\_a, arr\_b, и arr\_c соответственно.

 Эти циклы начинаются с 1, поскольку matrix[0][*][*], matrix[*][0][*] и matrix[*][*][0] уже инициализированы нулями, что помогает избежать проверки пустых последовательностей.

 Обновление значений в матрице:

 На каждой итерации в циклах функция обновляет значение matrix[i][j][k]. Это значение определяется как максимум из трех возможных вариантов:

 Длина НОП, если элемент arr\_a[i-1] не учитывается (то есть matrix[i - 1][j][k]).

 Длина НОП, если элемент arr\_b[j-1] не учитывается (то есть matrix[i][j - 1][k]).

 Длина НОП, если элемент arr\_c[k-1] не учитывается (то есть matrix[i][j][k - 1]).

 Проверка на совпадение элементов:

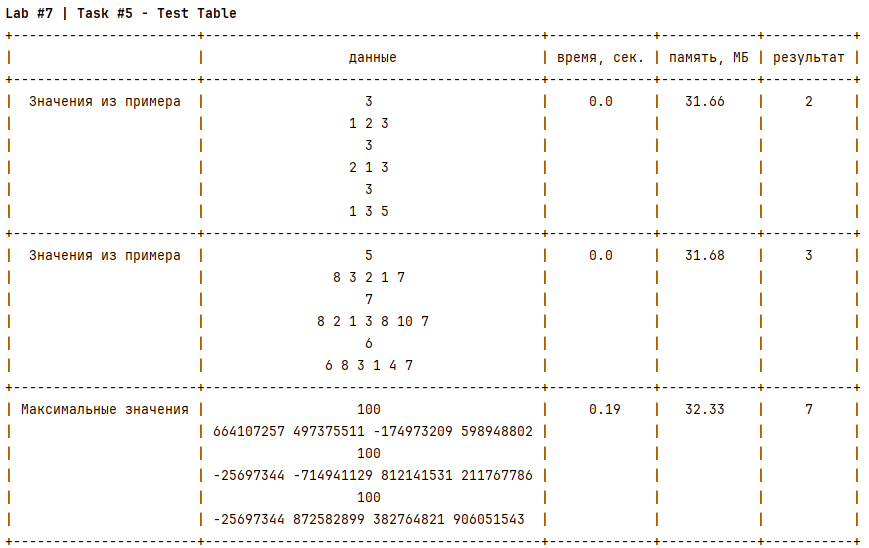
 Далее функция проверяет, совпадают ли текущие элементы всех трех массивов (arr\_a[i-1], arr\_b[j-1], и arr\_c[k-1]). Если совпадение есть, то длина НОП также может увеличиться. В этом случае значение matrix[i][j][k] обновляется как максимум между текущим значением и matrix[i - 1][j - 1][k - 1] + 1, что учитывает найденное совпадение.

 Возврат результата:

 После завершения всех тройных циклов функция возвращает значение matrix[n][m][l], которое представляет длину наибольшей общей подпоследовательности для трех входных массивов.

Таким образом, функция использует динамическое программирование для нахождения длины НОП, эффективно обрабатывая все возможные комбинации трех последовательностей и обновляя результаты в матрице.

Время выполнения программы, затраты памяти и результат:



Вывод по задаче: мы научились применять метод динамического программирования, который позволяет решать сложные задачи, разбивая их на более простые. Понимание принципа хранения промежуточных результатов для оптимизации времени выполнения алгоритма.

# **Дополнительные задачи**

## **Задача №4. НОП двух последовательностей. [N баллов]**

Текст задачи:

Вычислить длину самой длинной общей подпоследовательности из двух последовательностей. Даны две последовательности A = (a1, a2, ..., an) и B = (b1, b2, ..., bm), найти длину их самой длинной общей подпоследовательности, т.е. наибольшее неотрицатеьное целое число p такое, что существуют индексы 1 ≤ i1 < i2 < ... < ip ≤ n и 1 ≤ j1 < j2 < ... < jp ≤ m такие, что ai1 = bj1 , ..., aip = bjp .

• Формат ввода / входного файла (input.txt). – Первая строка: n - длина первой последовательности. – Вторая строка: a1, a2, ..., an через пробел. – Третья строка: m - длина второй последовательности. 4 – Четвертая строка: b1, b2, ..., bm через пробел.

• Ограничения: 1 ≤ n, m ≤ 100; −10^9 < ai , bi < 10^9 .

• Формат вывода / выходного файла (output.txt). Выведите число p. • Ограничение по времени. 1 сек.

Решение:

def longest\_common\_sub(n, arr\_a, m, arr\_b):  
 matrix = [[0] \* (m + 1) for \_ in range(n + 1)]  
 for i in range(1, n + 1):  
 for j in range(1, m + 1):  
 matrix[i][j] = max(matrix[i - 1][j], matrix[i][j - 1])  
 if arr\_a[i - 1] == arr\_b[j - 1]:  
 matrix[i][j] = max(matrix[i][j], matrix[i - 1][j - 1] + 1)  
 return matrix[n][m]

Объяснение решения:

Время выполнения, затраты памяти и результат:

Вывод по задаче:

## **Задача №6. Телефонная книга [N баллов]**

Текст задачи:

Решение:

Объяснение решения:

Время выполнения, затраты памяти и результат:

Вывод по задаче:

# **Вывод по лабораторной работе**

В ходе лабораторной работы мы узнали, как работают хэш-функции и хэш-таблицы, мы реализовали их и применили для решения различных задач. Также рассмотрели и реализовали разные способы борьбы с коллизиями.