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 10

Выполнил:

Абакар Иссака Малли

К3141

Проверил:

Афанасьев А. В.

Санкт-Петербург

2024 г.

**Содержание отчета**

**Оглавление**

[Содержание отчета 2](#_heading=h.30j0zll)

[1 задача. Обмен монет 3](#_heading=h.3znysh7)

[2 задача. Примитивный калькулятор 5](#_heading=h.tyjcwt)

[3 задача. Редакционное расстояние 6](#_heading=h.3dy6vkm)

[5 задача. Наибольшая общая подпоследовательность трех последовательностей 8](#_heading=h.1t3h5sf)

[Вывод 9](#_heading=h.4d34og8)

**Задачи по варианту**

**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 ≤ money ≤ 103 ). Набор монет: количество возможных монет k и сам набор coins = {coin1, ..., coink}. 1 ≤ k ≤ 100, 1 ≤ coini ≤ 103 . 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

Листинг кода

def get\_min\_cnt(amount, coin\_arr):  
 dp = [float('inf')] \* (amount + 1)  
 dp[0] = 0  
  
 for item in coin\_arr:  
 for i in range(item, amount + 1):  
 dp[i] = min(dp[i - item] + 1, dp[i])  
  
 return dp[amount]

Текстовое объяснение решения.

Создаем одномерную таблицу динамики, по порядку с меньшей до большей пересчитываем кол-во монет

Результат работы кода на примере из задачи:

Изображение выглядит как текст, Шрифт, снимок экрана, типография

Автоматически созданное описание

Результат работы кода на максимальных и минимальных значениях: Изображение выглядит как текст, Шрифт, снимок экрана

Автоматически созданное описаниеИзображение выглядит как текст, Шрифт, снимок экрана

Автоматически созданное описание

|  | Время выполнения | Затраты памяти |
| --- | --- | --- |
| Нижняя граница диапазона значений входных данных из текста задачи | 0.00078 sec | 15.36 Mb |
| Пример из задачи | 0.00082 sec | 15.38 Mb |
| Верхняя граница диапазона значений входных данных из текста задачи | 0.54783 sec | 38.51 Mb |

Вывод по задаче: Увеличение значений вводимых переменных в пределах ограничений увеличивает время работы программы в зависимости O(n) и используемую память.

**2 задача. Примитивный калькулятор**

Дан примитивный калькулятор, который может выполнять следующие три операции с текущим числом x: умножить x на 2, умножить x на 3 или прибавить 1 к x. Дано положительное целое число n, найдите минимальное количество операций, необходимых для получения числа n, начиная с числа 1. • Формат ввода / входного файла (input.txt). Дано одно целое число n, 1 ≤ n ≤ 106 . Посчитать минимальное количество операций, необходимых для получения n из числа 1. • Формат вывода / выходного файла (output.txt). В первой строке вывести минимальное число k операций. Во второй – последовательность промежуточных чисел a0, a1, ..., ak−1 таких, что a0 = 1, ak−1 = n и для всех 0 ≤ i < k – ai+1 равно или ai + 1, 2 · ai , или 3 · ai . Если есть несколько вариантов, выведите любой из них. 2 • Ограничение по времени. 1 сек.

Листинг кода

def prim\_calc(n):  
 order\_ans = [[] for i in range(n + 1)]  
 order\_ans[1] = [1]  
 cnt\_oper\_arr = [0] \* (n + 1)  
  
 for i in range(2, n + 1):  
 min\_cnt = float('inf')  
 ind\_min\_cnt = -1  
  
 if (i % 3 == 0) and cnt\_oper\_arr[i // 3] + 1 < min\_cnt:  
 min\_cnt, ind\_min\_cnt = cnt\_oper\_arr[i // 3] + 1, i //3  
  
 if (i % 2 == 0) and cnt\_oper\_arr[i // 2] + 1 < min\_cnt:  
 min\_cnt, ind\_min\_cnt = cnt\_oper\_arr[i // 2] + 1, i // 2  
  
 if cnt\_oper\_arr[i - 1] + 1 < min\_cnt:  
 min\_cnt, ind\_min\_cnt = cnt\_oper\_arr[i - 1] + 1, i - 1  
  
 cnt\_oper\_arr[i] = min\_cnt  
 order\_ans[i] = order\_ans[ind\_min\_cnt] + [i]  
  
 return cnt\_oper\_arr[-1], order\_ans[-1]

Текстовое объяснение решения.

Создаем таблицу для динамики и на каждом шаге пересчитываем минимальный путь, с которого можно прийти

Результат работы кода на примере из задачи:

Изображение выглядит как текст, Шрифт, снимок экрана, типография

Автоматически созданное описание

Результат работы кода на максимальных и минимальных значениях:

Изображение выглядит как текст, Шрифт, снимок экрана, дизайн

Автоматически созданное описание Изображение выглядит как текст, Шрифт, снимок экрана

Автоматически созданное описание

|  | Время выполнения | Затраты памяти |
| --- | --- | --- |
| Нижняя граница диапазона значений входных данных из текста задачи | 0.00128 sec | 15.11 Mb |
| Пример из задачи | 0.00137 sec | 15.21 Mb |
| Верхняя граница диапазона значений входных данных из текста задачи | 0.68457 sec | 37.84 Mb |

Вывод по задаче: Увеличение значений вводимых переменных в пределах ограничений увеличивает время работы программы в зависимости O(n) и используемую память.

**3 задача. Редакционное расстояние**

Редакционное расстояние между двумя строками – это минимальное количество операций (вставки, удаления и замены символов) для преобразования одной строки в другую. Это мера сходства двух строк. У редакционного расстояния есть применения, например, в вычислительной биологии, обработке текстов на естественном языке и проверке орфографии. Ваша цель в этой задаче – вычислить расстояние редактирования между двумя строками. 3 • Формат ввода / входного файла (input.txt). Каждая из двух строк ввода содержит строку, состоящую из строчных латинских букв. Длина обеих строк - от 1 до 5000. • Формат вывода / выходного файла (output.txt). Выведите расстояние редактирования между заданными двумя строками. • Ограничение по времени. 2 сек. • Ограничение по памяти. 256 мб

Листинг кода

def lev\_dist(st1, st2):  
  
 now\_row = [i for i in range(len(st1) + 1)]  
 for i in range(1, len(st2) + 1):  
 prev\_row, now\_row = now\_row, [i] + [0] \* len(st1)  
 for j in range(1, len(st1) + 1):  
 now\_row[j] = min(prev\_row[j] + 1, now\_row[j - 1] + 1, prev\_row[j - 1] + (st1[j - 1] != st2[i - 1]))  
  
 return now\_row[-1]

Текстовое объяснение решения.

Находим расстояние Левенштейна по специальному динамическому алгоритму

Результат работы кода на примере из задачи:

Изображение выглядит как текст, Шрифт, снимок экрана, типография

Автоматически созданное описание

Результат работы кода на максимальных и минимальных значениях: Изображение выглядит как текст, Шрифт, снимок экрана

Автоматически созданное описание Изображение выглядит как текст, Шрифт, снимок экрана

Автоматически созданное описание

|  | Время выполнения | Затраты памяти |
| --- | --- | --- |
| Нижняя граница диапазона значений входных данных из текста задачи | 0.00087 sec | 14.72 Mb |
| Пример из задачи | 0.00093 sec | 14.82 Mb |
| Верхняя граница диапазона значений входных данных из текста задачи | 0.56134 sec | 36.34 Mb |

Вывод по задаче: Увеличение значений вводимых переменных в пределах ограничений увеличивает время работы программы в зависимости времени работы O(n), а также используемую память.

**5 задача. Наибольшая общая подпоследовательность трех последовательностей**

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; −109 < ai , bi , ci < 109 . • Формат вывода / выходного файла (output.txt). Выведите число p. • Ограничение по времени. 1 сек.

Листинг кода

def get\_longest\_sub(arr1, arr2, arr3):  
  
 dp = [[[0] \* (len(arr3) + 1) for i in range(len(arr2) + 1)] for j in range(len(arr1) + 1)]  
  
 for i in range(1, len(arr1) + 1):  
 for j in range(1, len(arr2) + 1):  
 for k in range(1, len(arr3) + 1):  
 if arr1[i - 1] == arr2[j - 1] == arr3[k - 1]:  
 dp[i][j][k] = dp[i - 1][j - 1][k - 1] + 1  
 else:  
 dp[i][j][k] = max(dp[i - 1][j][k], dp[i][j - 1][k], dp[i][j][k - 1])  
  
 return dp[-1][-1][-1]

Текстовое объяснение решения.

Создаем трехмерную динамику, если символы совпадают увеличиваем значение по диагонали, иначе берем максимум с трех сторон

Результат работы кода на примере из задачи:

Изображение выглядит как текст, Шрифт, снимок экрана, типография

Автоматически созданное описание

Результат работы кода на максимальных и минимальных значениях:

Изображение выглядит как текст, Шрифт, снимок экрана

Автоматически созданное описание Изображение выглядит как текст, Шрифт, снимок экрана

Автоматически созданное описание

|  | Время выполнения | Затраты памяти |
| --- | --- | --- |
| Нижняя граница диапазона значений входных данных из текста задачи | 0.00095 sec | 14.87 Mb |
| Пример из задачи | 0.00107 sec | 14.98 Mb |
| Верхняя граница диапазона значений входных данных из текста задачи | 0.65414 sec | 35.42 Mb |

Вывод по задаче: Увеличение значений вводимых переменных в пределах ограничений увеличивает время работы программы в зависимости O(n) и используемую память.

**Вывод**

1. В этой лабораторной мы изучили мощный инструмент - динамическое программирование, а также реализовывали некоторые алгоритмы, основанные на динамическом подходе
2. Алгоритм O(n) и O(n logn) выполняется достаточно быстро, относительно квадратичной сложности, затраты памяти также прямо пропорциональны линейной.
3. С помощью методов *time.perf\_counter()* и *psutil.Process().memory\_info().rss* можно отслеживать ресурсозатратность алгоритмов.