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 1

Выполнил:
Бен Шамех Абделазиз
К3239

Проверил:
Ромакина Оксана Михайловна

Санкт-Петербург
2025 г.

Содержание отчета

Оглавление

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 ($1 \leq money \leq 103$). Набор монет: количество возможных монет k и сам набор coins = {coin1, ..., coink}. $1 \leq k \leq 100$, $1 \leq coini \leq 103$. 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

```python
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]
```

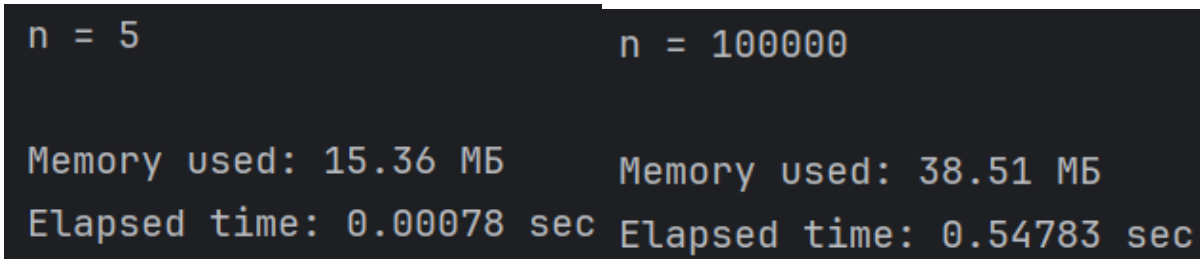
Текстовое объяснение решения.

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

Результат работы кода на примере из задачи:

```
Memory used: 15.38 МБ
Elapsed time: 0.00082 sec
```

Результат работы кода на максимальных и минимальных значениях:



	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, $1 \le n \le 106$. 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 $0 \le i < k$ – ai+1 равно или ai + 1, $2 \cdot ai$, или $3 \cdot ai$. 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]
```

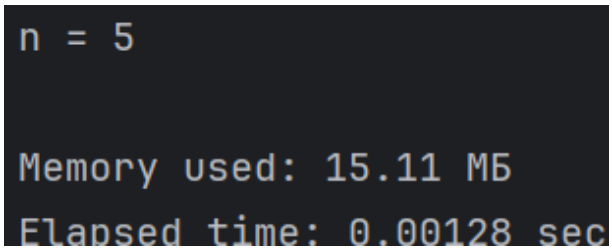
Текстовое объяснение решения.

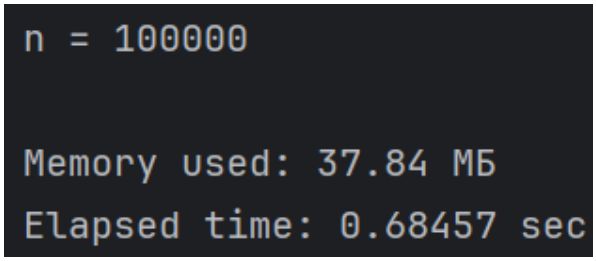
Создаем таблицу для динамики и на каждом шаге пересчитываем минимальный путь, с которого можно прийти

Результат работы кода на примере из задачи:

```
Memory used: 15.21 МБ
Elapsed time: 0.00137 sec
```

Результат работы кода на максимальных и минимальных значениях:





	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

```python
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:

```
Memory used: 14.82 МБ
Elapsed time: 0.00093 sec
```

Результат работы кода на максимальных и минимальных значениях:

```
n = 5

Memory used: 14.72 МБ
Elapsed time: 0.00087 sec
```

```
n = 100000

Memory used: 36.34 МБ
Elapsed time: 0.56134 sec
```

	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	0.56134 sec	36.34 Mb

входных данных из текста задачи		

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 неотрицательное целое число p такое, что существуют индексы $1 \le i1 < i2 < ... < ip \le n$, $1 \le j1 < j2 < ... < jp \le m$ и $1 \le k1 < k2 < ... < kp \le l$ 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: $1 \le n, m, l \le 100$; $-109 < ai , bi , ci < 109$. • 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:

```
Memory used: 14.98 МБ
Elapsed time: 0.00107 sec
```

Результат работы кода на максимальных и минимальных значениях:

```
n = 3

Memory used: 14.87 МБ
Elapsed time: 0.00095 sec
```

```
n = 100000

Memory used: 35.42 МБ
Elapsed time: 0.65414 sec
```

	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.