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

Выполнил:
Волжева М.И.
К3141

Проверила:
Артамонова В.Е.

Санкт-Петербург
2022 г.

Содержание отчета

Задачи по варианту

Задача №4. Наибольшая общая подпоследовательность двух последовательностей

Вычислить длину самой длинной общей подпоследовательности из двух последовательностей.

- Формат 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. Ограничения: 1 ≤ n, m ≤ 100; -10^9 < ai, bi < 10^9
- Формат выходного файла (output.txt) (output.txt). Выведите число p.
- Ограничение по времени. 1 сек.
- Пример:

Input.txt	3 2 7 5 2 2 5	1 7 4 1 2 3 4	4 2 7 8 3 4 5 2 8 7
Output.txt	2	0	2

Листинг кода:

```
import time
import os, psutil


def find_max_subsequence(arr1, arr2):
    n = len(arr1)
    m = len(arr2)
    dp = [[0] * (m + 1) for _ in range(n + 1)]
    p = [[None] * (m + 1) for _ in range(n + 1)]

    for i in range(1, n + 1):
        for j in range(1, m + 1):
            if arr1[i - 1] == arr2[j - 1]:
                dp[i][j] = dp[i - 1][j - 1] + 1
                p[i][j] = (i - 1, j - 1, arr1[i - 1])
            else:
                if dp[i - 1][j] > dp[i][j - 1]:
                    dp[i][j] = dp[i - 1][j]
                    p[i][j] = (i - 1, j, "") # первые 2 элемента кортежа -
ссылки на предыдущий, третий - значение предыдущего
                else:
                    dp[i][j] = dp[i][j - 1]
                    p[i][j] = (i, j - 1, "")

    ans = []
```

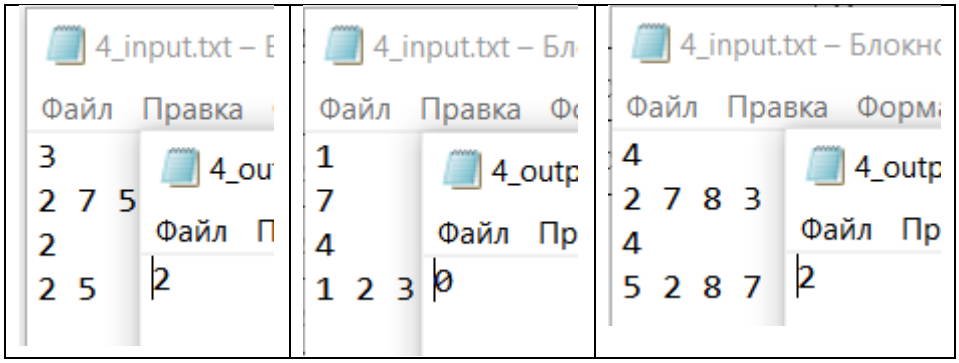
```
    cur = p[n][m]
    while cur is not None:
        if len(cur[2]) > 0:
            ans.append(int(cur[2]))
        cur = p[cur[0]][cur[1]]

    return ans[::-1]


t_start = time.perf_counter()
process = psutil.Process(os.getpid())
f = open("4_input.txt")
m = open("4_output.txt", "w")

count1 = int(f.readline())
string1 = f.readline()
elements1 = list(map(str, string1.split()))

count2 = int(f.readline())
string2 = f.readline()
elements2 = list(map(str, string2.split()))

m.write(str(len(find_max_subsequence(elements1, elements2))))


f.close()
m.close()

print("Time of working: %s second" % (time.perf_counter() - t_start))
print("Memory", process.memory_info().rss/(1024*1024), "mb")
```

Текстовое объяснение решения:

В функцию передаются два массива. Мы рассматриваем максимальную подпоследовательность частей массива, потом путем прибавления по 1 элементу из одной из последовательностей доходим до добавление всех элементов. Для этого мы создаём массив dp, в котором содержится следующая информация dp[i][j] – длина максимальной подпоследовательности, содержащая первые i элементов первой последовательности и перввые j элементов второй последовательности. При добавление нового элемента, то есть заполнения элемента dp, у нас есть 2 варианта – есть 2 равных элементов, тогда мы просто увеличиваем предыдущее на 1, или мы добавляем максимум от 2 предыдущих. В ответ у нас соответственно идёт самый последний элемент массива (крайний справа снизу).

Остальные операции, выполненные не в функции стандартные и описаны в предыдущих лабораторных.

Результат работы кода на примерах из текста задачи:



Тестирование алгоритма:

	Время выполнения	Затраты памяти
Пример 1 из задачи	0.002402499999999974 second	13.9453125 mb
Пример 2 из задачи	0.005738200000000138 second	13.9453125 mb
Пример 3 из задачи	0.009040700000000013 second	13.9453125 mb
Минимальные значения	0.0009274000000000227 second	13.94140625 mb
Максимальные значения	0.014985799999999994 second	14.953125 mb

Вывод по задаче: Мы узнали, что такое динамическое программирование и научились решать задачу о поиске максимальной общей подпоследовательности двух последовательностей.

Задача №6. Наибольшая возрастающую подпоследовательность

Дана последовательность, требуется найти ее наибольшую возрастающую подпоследовательность.

- Формат входного файла (input.txt) В первой строке входных данных задано целое число n – длина последовательности ($1 \leq n \leq 300000$). Во второй строке задается сама последовательность. Числа разделяются пробелом. Элементы последовательности – целые числа, не превосходящие по модулю 10^9
- Формат выходного файла (output.txt) В первой строке выведите длину наибольшей возрастающей подпоследовательности, а во второй строке выведите через пробел саму наибольшую возрастающую подпоследовательность данной последовательности. Если ответов несколько – выведите любой.
- Ограничение по времени. 2 сек.
- Ограничение по памяти. 256 мб.
- Пример:

Input.txt	6 3 29 5 5 28 6
Output.txt	3 3 5 28

Листинг кода:

```
import time
import os, psutil


def find_max_increasing_subsequence(arr, num):
    ans = [1 for _ in range(num)]
    ans_numbers = [None for _ in range(num)]
    for i in range(1, num):
        for j in range(0, i - 1):
            if arr[i] > arr[j]:
                if (ans[j] + 1) > ans[i]:
                    ans_numbers[i] = (j, arr[i]) # первый элемент кортежа - ссылка на предыдущий, второй - значение предыдущего
                    ans[i] = ans[j] + 1
                else:
                    ans[i] = ans[i]

    final_answer = []
    for i in range(len(ans)):
        if ans[i] == max(ans):
            cur = i
            break
```

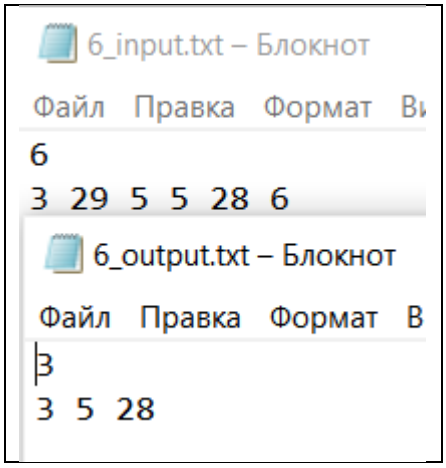
```
    while ans_numbers[cur] is not None:
        final_answer.append(ans_numbers[cur][1])
        cur = ans_numbers[cur][0]
    final_answer.append(arr[cur]) #добавляем первый элемент

    return final_answer[::-1]


t_start = time.perf_counter()
process = psutil.Process(os.getpid())
f = open("6_input.txt")
m = open("6_output.txt", "w")

count = int(f.readline())
string = f.readline()
elements = list(map(int, string.split()))


print(elements)
answer = find_max_increasing_subsequence(elements, count)
m.write(str(len(answer)) + "\n" )
for i in answer:
    m.write(str(i) + " ")
f.close()
m.close()

print("Time of working: %s second" % (time.perf_counter() - t_start))
print("Memory", process.memory_info().rss/(1024*1024), "mb")
```

Текстовое объяснение решения:

В функцию передаются массив и количество элементов. В массиве ans – длина максимальной возрастающей подпоследовательности до текущего элемента. В массиве ans_numbers – кортеж, первый элемент кортежа - ссылка на предыдущий элемент подпоследовательности, второй - значение предыдущего элемента подпоследовательности. Идёт последовательное заполнение массивов, ответом будет являться максимальное значение массива ans. После этого нам нужно распаковать массив ans_numbers, использую ссылки представленные в кортеже. Мы переворачиваем массив, потому что распаковка проходила с конца.

Остальные операции, выполненные не в функции стандартные и описаны в предыдущих лабораторных.

Результат работы кода на примерах из текста задачи:



Тестирование алгоритма:

	Время выполнения	Затраты памяти
Пример 1 из задачи	0.0012595999999999719 second	14.0 mb
Минимальные значения	0.001257399999999964 second	13.98046875 mb
Максимальные значения	3.4492016000000003 second	14.96875 mb

Вывод по задаче: Мы научились реализовывать задачу п поиску максимальной возрастающей подпоследовательности с помощью динамического программирования.

Вывод

Мы узнали, что такое динамическое программирование и научились реализовывать некоторые задачи с его использованием.