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 11

Выполнил:
Тимаков Е.П. (фамилия имя)
К3141 (номер группы)

Проверила:
Артамонова В.Е.

Санкт-Петербург 2024 г.

Содержание отчета

Задачи по варианту

Задача №5. Высота дерева

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 $1 \le i1 < i2 < ... < ip \le n$, $1 \le j1 < j2 < ... < jp \le m$ 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. (именно листинг, а не скрины)

```
def lcs(a: list, b: list, c: list) -> int:
    matrix: list = [[[0 for _ in range(len(c) + 1)] for _ in range(len(b) + 1)] for _ in range(len(a) + 1)]
    for a_index, a_el in enumerate(a):
        for b_index, b_el in enumerate(b):
            for c_index, c_el in enumerate(c):
                if a_el == b_el == c_el:
                    matrix[a_index + 1][b_index + 1][c_index + 1] = matrix[a_index][b_index][c_index] + 1
                else:
                    matrix[a_index + 1][b_index + 1][c_index + 1] = max(matrix[a_index + 1][b_index][c_index],
                                                                        matrix[a_index][b_index + 1][c_index],
                                                                        matrix[a_index][b_index][c_index + 1],
                                                                        matrix[a_index + 1][b_index + 1][c_index],
```

```
                                                    matrix[a_index +
1][b_index][c_index + 1],
                                                    matrix[a_index][b_index +
1][c_index + 1])
    return matrix[-1][-1][-1]



input_arrays: list[list[int]] = [[], [], []]
with open("input.txt") as input_file:
    for i in range(3):
        input_file.readline()
        input_arrays[i] = list(map(int, input_file.readline().split(" ")))
with open("output.txt", "w") as output:
    print(lcs(input_arrays[0], input_arrays[1], input_arrays[2]), file=output)
```

Текстовое объяснение решения:

В данном коде мы создаем трехмерную матрицу для хранения всех трех длинн. Проходим по элементам списков a, b, c. Если элементы совпадают, то увеличиваем текущее значение матрицу на +1, иначе берем max значение из других ячеек матрицы.

Результат работы кода на примерах из текста задачи:

input.txt:

```
3
1 2 3
3
2 1 3
3
1 3 5
```

output.txt:

```
2
```

input.txt:

```
5
8 3 2 1 7
7
8 2 1 3 8 10 7
6
6 8 3 1 4 7
```

output.txt:

```
3
```

Результат работы кода на максимальных:

input.txt:

```
100
34872824 547383840 845059975 900108518 -496422477 312389614 685069461 -841541664 -860586678 -399058723 -124477987 13102893 3
100
262458077 -727106461 -294803825 -77577819 -354277228 -962971611 125538264 -895599963 -496496085 442356944 213769686 -7400882
100
-319247982 -418071564 297236925 903030440 -777864762 760198655 -705801119 -61163050 427730613 52421884 -527566709 -951049094
```

output.txt:

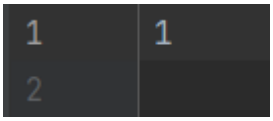
```
0
```

Результат работы кода на минимальных значениях:

input.txt:

```
1
1
1
1
1
1
```

output.txt:



	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.001	25
Пример из задачи	0.001	25
Пример из задачи	0.001	25
Верхняя граница диапазона значений входных данных из текста задачи	0.416	26

Вывод по задаче:

В данной задаче, мы ознакомились с структурой дерева. Также мы реализовали алгоритм подсчета высоты дерева.

Задача №6. Наибольшая возрастающая подпоследовательность

Дана последовательность, требуется найти ее наибольшую возрастающаю подпоследовательность.

• Формат ввода / входного файла (input.txt). В первой строке входных данных задано целое число n – длина последовательности ($1 \le n \le 300000$). Во второй строке задается сама последовательность. Числа разделяются пробелом. Элементы последовательности – целые числа, не превосходящие по модулю 109 . – Подзадача 1 (полегче). $n \le 5000$. – Общая подзадача. $n \le 300000$.

• Формат вывода / выходного файла (output.txt). В первой строке выведите длину наибольшей возрастающей подпоследовательности, а во второй строке выведите через пробел саму наибольшую возрастающаю подпоследовательность данной последовательности. Если ответов несколько - выведите любой.

• Ограничение по времени. 2 сек.

• Ограничение по памяти. 256 мб.

Листинг кода:

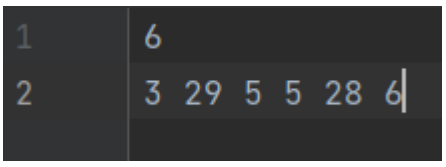
```
import bisect


def find_lis_length_and_sequence(sequence):
    if not sequence:
        return 0, []

    n = len(sequence)
    dp = []
    parent = [-1] * n
    lis_end_at = [-1] * n

    for i in range(n):
        pos = bisect.bisect_left(dp, sequence[i])
        if pos < len(dp):
            dp[pos] = sequence[i]
        else:
            dp.append(sequence[i])

        if pos > 0:
            parent[i] = lis_end_at[pos - 1]
        lis_end_at[pos] = i

    # Reconstruct the LIS
    lis_length = len(dp)
    lis = []
    k = lis_end_at[lis_length - 1]
    while k >= 0:
        lis.append(sequence[k])
        k = parent[k]
    lis.reverse()

    return lis_length, lis


def main():
    with open('input.txt', 'r') as f:
        n = int(f.readline().strip())
```

```
        sequence = list(map(int, f.readline().strip().split()))

    lis_length, lis = find_lis_length_and_sequence(sequence)

    with open('output.txt', 'w') as f:
        f.write(f"{lis_length}\n")
        f.write(" ".join(map(str, lis)) + "\n")


if __name__ == "__main__":
    main()
```

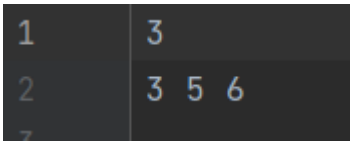
Текстовое объяснение решения:

В данный код проходится по каждому элементу последовательности и обновляется массив dp, который хранит последние значения наименьшей возрастающей подпоследовательности каждой длины. Также обновляются массивы parent и lis_end_at, которые отслеживают предыдущий элемент и индекс элемента, где заканчивается LIS каждой длины соответственно. После построения этих массивов, происходит восстановление LIS и возвращаются его длина и элементы.

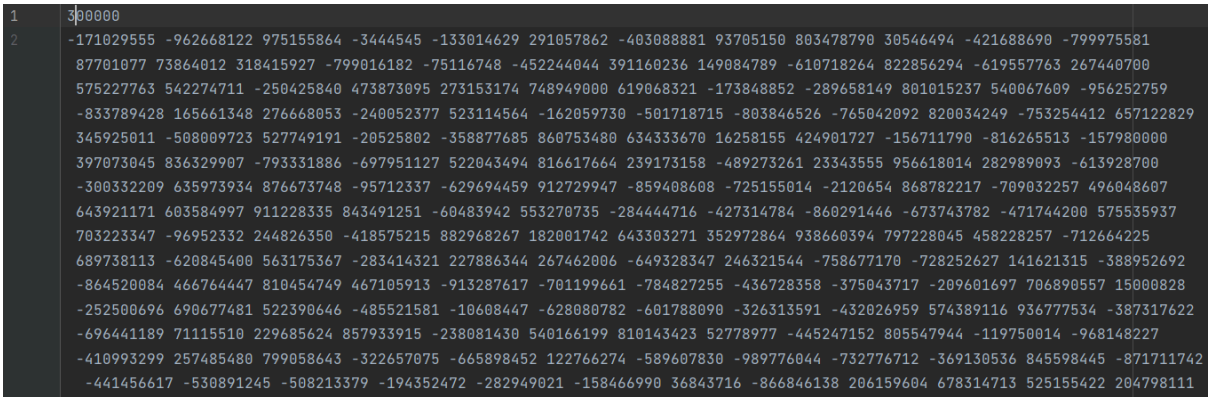
Результат работы кода на примерах из текста задачи:
input.txt:



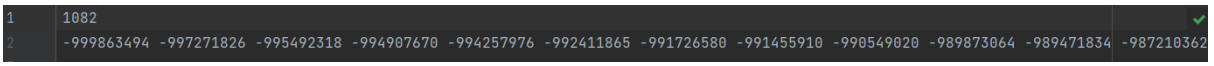
output.txt:



Результат работы кода на максимальных:
input.txt:

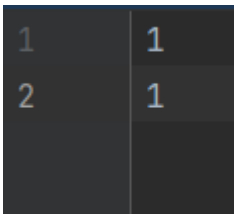


output.txt:

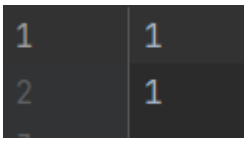


Результат работы кода на минимальных значениях:

input.txt:



output.txt:



	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.001	25
Пример из задачи	0.001	25
Верхняя граница диапазона значений входных данных из текста задачи	0.016	38

Вывод по задаче:

В данной задаче мы изучили и реализовали алгоритм динамического программирования. Также мы использовали алгоритм бинарного поиска для оптимизации.

Вывод

В данной лабораторной работе мы изучили и реализовали алгоритм динамического программирования.