

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

Содержание отчета	2
Задачи по варианту	2
Задача №2. Высота дерева	2
Задача №7. Снова сортировка	5
Вывод	9

Задачи по варианту

Задача №5. Высота дерева

Вычислить длину самой длинной общей подпоследовательности из трех последовательностей. Даны три последовательности $A = (a_1, a_2, \dots, a_n)$, $B = (b_1, b_2, \dots, b_m)$ и $C = (c_1, c_2, \dots, c_l)$, найти длину их самой длинной общей подпоследовательности, т.е. наибольшее неотрицательное целое число p такое, что существуют индексы $1 \leq i_1 < i_2 < \dots < i_p \leq n$, $1 \leq j_1 < j_2 < \dots < j_p \leq m$ и $1 \leq k_1 < k_2 < \dots < k_p \leq l$ такие, что $a_{i_1} = b_{j_1} = c_{k_1}$, ..., $a_{i_p} = b_{j_p} = c_{k_p}$.

- Формат ввода / входного файла (input.txt). – Первая строка: n – длина первой последовательности. – Вторая строка: a_1, a_2, \dots, a_n через пробел. – Третья строка: m – длина второй последовательности. – Четвертая строка: b_1, b_2, \dots, b_m через пробел. – Пятая строка: l – длина второй последовательности. – Шестая строка: c_1, c_2, \dots, c_l через пробел.

- Ограничения: $1 \leq n, m, l \leq 100$; $-109 < a_i, b_i, c_i < 109$.

- 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 длин. Проходим по элементам списков a, b, c. Если элементы совпадают, то увеличиваем текущее значение матрицы на +1, иначе берем значение из других ячеек матрицы.

Результат работы кода на примерах из текста задачи:

input.txt:

1	3	
2	1	2 3
3	3	
4	2	1 3
5	3	
6	1	3 5

output.txt:

1	2
2	

input.txt:

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

output.txt:

1	3
2	

Результат работы кода на максимальных:

input.txt:

1	100
2	34872824 547383840 845059975 900108518 -496422477 312389614 685069461 -841541664 -860586678 -399058723 -124477987 13102893 39
3	100
4	262458077 -727106461 -294803825 -77577819 -354277228 -962971611 125538264 -895599963 -496496085 442356944 213769686 -74008823
5	100
6	-319247982 -418071564 297236925 903030440 -777864762 760198655 -705801119 -61163050 427730613 52421884 -527566709 -951049094
7	

output.txt:

1	0
2	

Результат работы кода на минимальных значениях:

input.txt:

1	1
2	1
3	1
4	1
5	1
6	1

output.txt:

1	1
2	

	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 возрастающую подпоследовательность.

- Формат ввода / 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 . – Подзадача 1 (полегче). $n \leq 5000$. – Общая подзадача. $n \leq 300000$.

- Формат вывода / 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.

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:

1	6
2	3 29 5 5 28 6

output.txt:

1	3
2	3 5 6

Результат работы кода на максимальных:

input.txt:


```
1 30000
2 -171029555 -962668122 975155864 -3444545 -133014629 291057862 -403088881 93705150 803478790 30546494 -421688690 -799975581
87701077 73864012 318415927 -799016182 -75116748 -452244044 391160236 149084789 -610718264 822856294 -619557763 267440700
575227763 542274711 -250425840 473873095 273153174 748949000 619068321 -173848852 -289658149 801015237 540067609 -956252759
-833789428 165661348 276668053 -240052377 523114564 -162059730 -501718715 -803846526 -765042092 820034249 -753254412 657122829
345925011 -508009723 527749191 -20525802 -358877685 860753480 634333670 16258155 424901727 -156711790 -816265513 -157980000
397073045 836329907 -793331886 -697951127 522043494 816617664 239173158 -489273261 23343555 956618014 282989093 -613928700
-300332209 635973934 876673748 -95712337 -629694459 912729947 -859408608 -725155014 -2120654 868782217 -709032257 496048607
643921171 603584997 911228335 843491251 -60483942 553270735 -284444716 -427314784 -860291446 -673743782 -471744200 575535937
703223347 -96952332 244826350 -418575215 882968267 182001742 643303271 352972864 938660394 797228045 458228257 -712664225
689738113 -620845400 563175367 -283414321 227886344 267462006 -649328347 246321544 -758677170 -728252627 141621315 -388952692
-864520084 466764447 810454749 467105913 -913287617 -701199661 -784827255 -436728358 -375043717 -209601697 706890557 15000828
-252500696 690677481 522390646 -485521581 -10608447 -628080782 -601788090 -326313591 -432026959 574389116 936777534 -387317622
-696441189 71115510 229685624 857933915 -238081430 540166199 810143423 52778977 -445247152 805547944 -119750014 -968148227
-410993299 257485480 799058643 -322657075 -665898452 122766274 -589607830 -989776044 -732776712 -369130536 845598445 -871711742
-441456617 -530891245 -508213379 -194352472 -282949021 -158466990 36843716 -866846138 206159604 678314713 525155422 204798111
```

output.txt:

```
1 1082
2 -999863494 -997271826 -995492318 -994907670 -994257976 -992411865 -991726580 -991455910 -990549020 -989873064 -989471834 -987210362
```

Результат работы кода на минимальных значениях:

input.txt:

```
1 1
2 1
```

output.txt:

```
1 1
2 1
```

	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.

