

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  
Вариант 5

Выполнил:  
Артемов И. В.  
К3141

Проверил:  
Афанасьев А. В.

Санкт-Петербург  
2024 г.

# Содержание отчета

Содержание отчета	2
Задачи по варианту	3
Задача №5. Наибольшая общая подпоследовательность трех последовательностей	3
Задача №7. Шаблоны	7
Дополнительные задачи	11
Задача №4. Наибольшая общая подпоследовательность двух последовательностей	11
Задача №6. Наибольшая возрастающая подпоследовательность	15
Вывод	19

## Задачи по варианту

### Задача №5. Наибольшая общая подпоследовательность трех последовательностей

Текст задачи.

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

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  $a_{i_1} = b_{j_1} = c_{k_1}, \dots, a_{i_p} = b_{j_p} = c_{k_p}$ .

- **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:  $1 \leq n, m, l \leq 100$ ;  $-10^9 < a_i, b_i, c_i < 10^9$ .
- **Формат вывода / выходного файла (output.txt).** Выведите число  $p$ .
- Ограничение по времени. 1 сек.
- Примеры:

input.txt	output.txt	input.txt	output.txt
3	2	5	3
1 2 3		8 3 2 1 7	
3		7	
2 1 3		8 2 1 3 8 10 7	
3		6	
1 3 5		6 8 3 1 4 7	

- В первом примере одна общая подпоследовательность – (1, 3) длиной 2. Во втором примере есть две общие последовательности длиной 3 элемента – (8, 3, 7) и (8, 1, 7).

Листинг кода.

```
# Импортируем библиотеки для отслеживания памяти и времени выполнения программы
import tracemalloc
```

```

import time
from lab7.utils import *

# Запускаем таймер для измерения времени работы программы
t_start = time.perf_counter()

# Включаем отслеживание памяти
tracemalloc.start()

current_dir = os.path.dirname(os.path.abspath(__file__)) # Текущая
директория
txtf_dir = os.path.join(os.path.dirname(current_dir), "txtf") # Директория
txtf
input_path = os.path.join(txtf_dir, "input.txt")

def longest_common_subsequence_3(a, b, c):
    """
    Вычисляет длину самой длинной общей подпоследовательности для трёх
    последовательностей.
    """
    n, m, l = len(a), len(b), len(c)

    # Инициализируем 3D массив dp размером (n+1) x (m+1) x (l+1)
    dp = [[[0] * (l + 1) for _ in range(m + 1)] for __ in range(n + 1)]

    # Заполнение dp массива
    for i in range(1, n + 1):
        for j in range(1, m + 1):
            for k in range(1, l + 1):
                if a[i - 1] == b[j - 1] == c[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[n][m][l]

# Основной блок программы
if __name__ == "__main__":
    # Читаем данные из файла input.txt с помощью функции open_file
    input_data = open_file(input_path)

    # Разбираем данные
    n = int(input_data[0][0])
    m = int(input_data[2][0])
    l = int(input_data[4][0])
    a = list(map(int, input_data[1].split()))
    b = list(map(int, input_data[3].split()))
    c = list(map(int, input_data[5].split()))

    # Проверяем корректность данных
    if all(1 <= val <= 100 for val in (n, m, l)) and \
        all(-10**9 < x < 10**9 for x in a + b + c):
        print(f"Task 5\nInput:\nA: {a}\nB: {b}\nC: {c}")
        delete_prev_values(5)

    # Вычисляем результат
    result = longest_common_subsequence_3(a, b, c)

    # Записываем результат в файл output.txt
    output_path = get_output_path(5)
    write_file(str(result), output_path)

```

```

print_output_file(5)

else:
    # Выводим сообщение об ошибке, если данные некорректны
    print("Введите корректные данные")

# Выводим время работы программы
print("Время работы: %s секунд" % (time.perf_counter() - t_start))
# Выводим количество памяти, затраченной на выполнение программы
print("Затрачено памяти:", tracemalloc.get_traced_memory()[1], "байт")

# Останавливаем отслеживание памяти
tracemalloc.stop()

```

Текстовое объяснение решения.

1. Импортирует необходимые библиотеки для работы с файлами, измерения времени и памяти.
2. Считывает входные данные из файла `input.txt` через функцию `open\_file`.
3. Преобразует данные в три числовые последовательности A, B, и C.
4. Проверяет корректность входных данных (диапазон значений, размеры последовательностей).
5. Рассчитывает длину самой длинной общей подпоследовательности (LCS) для трёх последовательностей с использованием трёхмерного динамического программирования.
6. Записывает результат в файл `output.txt` через функцию `write\_file`.
7. Выводит результат на экран и отображает содержимое выходного файла.
8. Измеряет и выводит время выполнения программы.
9. Измеряет и выводит максимальный объём использованной памяти.
10. Останавливает отслеживание использования памяти.

Результат работы кода на примере из текста задачи:

task5.py		test_task5.py		input.txt		ou	
1	3						
2	1 2 3						
3	3						
4	2 1 3						
5	3						
6	1 3 5						

Результат работы кода на максимальных и минимальных значениях:

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.0006962090265005827 секунд	15496 байт
Пример из задачи	0.0010064170055557042 секунд	15550 байт
Верхняя граница диапазона значений входных данных из текста задачи	0.4109084579977207 секунд	430401 байт

Вывод по задаче: мною был изучен алгоритм нахождения наибольшей общей подпоследовательности трех последовательностей.

## Задача №7. Шаблоны

Текст задачи.

### 7 задача. Шаблоны

Многие операционные системы используют шаблоны для ссылки на группы объектов: файлов, пользователей, и т. д. Ваша задача – реализовать простейший алгоритм проверки шаблонов для имен файлов.

В этой задаче алфавит состоит из маленьких букв английского алфавита и точки («.»). Шаблоны могут содержать произвольные символы алфавита, а также два специальных символа: «?» и «\*». Знак вопроса («?») соответствует ровно одному произвольному символу. Звездочка «\*» соответствует подстроке произвольной длины (возможно, нулевой). Символы алфавита, встречающиеся в шаблоне, отображаются на ровно один такой же символ в проверяемой строке. Строка считается подходящей под шаблон, если символы шаблона можно последовательно отобразить на символы строки таким образом, как описано выше. Например, строки «ab», «aab» и «beda.» подходят под шаблон «\*a?», а строки «bebe», «a» и «ba» – нет.

- **Формат ввода / входного файла (input.txt).** Первая строка входного файла определяет шаблон. Вторая строка  $S$  состоит только из символов алфавита. Ее необходимо проверить на соответствие шаблону. Длины обеих строк не превосходят 10 000. Строки могут быть пустыми – будьте внимательны!
- **Формат вывода / выходного файла (output.txt).** Если данная строка подходит под шаблон, выведите YES. Иначе выведите NO.
- Ограничение по времени. 2 сек.
- Ограничение по памяти. 256 мб.
- Пример:

input.txt	output.txt	input.txt	output.txt
k?t*n	YES	k?t?n	NO
kitten		kitten	

Листинг кода.

```
# Импортируем библиотеки для отслеживания памяти и времени выполнения программы
import tracemalloc
import time
import os
from lab7.utils import *

# Запускаем таймер для измерения времени работы программы
t_start = time.perf_counter()

# Включаем отслеживание памяти
tracemalloc.start()

# Определяем пути к файлам
current_dir = os.path.dirname(os.path.abspath(__file__)) # Директория task/src
txtf_dir = os.path.join(os.path.dirname(current_dir), "txtf") # Директория task/txtf
```

```

input_path = os.path.join(txtf_dir, "input.txt")

def matches_pattern(pattern: str, string: str) -> str:
    """
    Проверяет, соответствует ли строка string шаблону pattern.

    :param pattern: Шаблон строки с использованием символов алфавита, '?' и '*'
    :param string: Проверяемая строка.
    :return: 'YES', если строка соответствует шаблону, иначе 'NO'.
    """
    n, m = len(pattern), len(string)
    dp = [[False] * (m + 1) for _ in range(n + 1)]

    # Пустой шаблон соответствует только пустой строке
    dp[0][0] = True

    # Инициализация строки, если шаблон начинается с '*'
    for i in range(1, n + 1):
        if pattern[i - 1] == '*':
            dp[i][0] = dp[i - 1][0]
        else:
            break

    # Заполняем таблицу
    for i in range(1, n + 1):
        for j in range(1, m + 1):
            if pattern[i - 1] == string[j - 1] or pattern[i - 1] == '?':
                dp[i][j] = dp[i - 1][j - 1]
            elif pattern[i - 1] == '*':
                # '*' может соответствовать пустой строке или одному или
                # нескольким символам
                dp[i][j] = dp[i - 1][j] or dp[i][j - 1]

    # Результат в последней ячейке
    return "YES" if dp[n][m] else "NO"

# Основной блок программы
if __name__ == "__main__":
    # Читаем данные из файла input.txt с помощью функции open_file
    lines = open_file(input_path)
    pattern = lines[0] if len(lines) > 0 else ""
    string = lines[1] if len(lines) > 1 else ""

    # Проверяем данные и записываем результат
    if len(pattern) <= 10_000 and len(string) <= 10_000:
        print(f"\nTask 7:\nPattern: {pattern}\nString: {string}")
        delete_prev_values(7)
        result = matches_pattern(pattern, string)
        output_path = get_output_path(7)
        write_file(result, output_path)
        print_output_file(7)
    else:
        print('Введите корректные данные')

    # Выводим время работы программы
    print("Время работы: %s секунд" % (time.perf_counter() - t_start))
    # Выводим количество памяти, затраченной на выполнение программы
    print("Затрачено памяти:", tracemalloc.get_traced_memory()[1], "байт")

    # Останавливаем отслеживание памяти
    tracemalloc.stop()

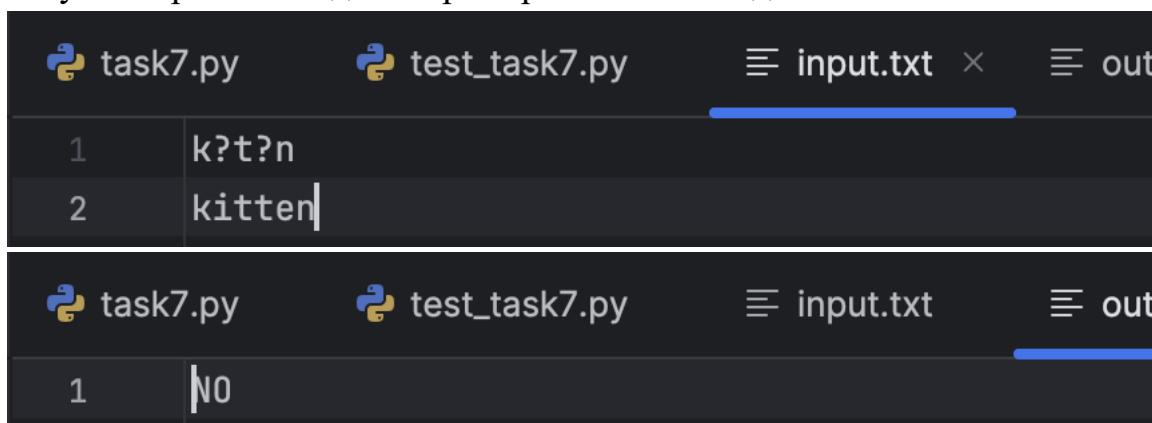
```



Текстовое объяснение решения.

1. Импортирует необходимые библиотеки для работы с файлами, измерения времени и памяти.
2. Считывает входные данные из файла `input.txt` через функцию `open\_file`.
3. Преобразует входные данные в шаблон `pattern` и строку `string`.
4. Проверяет корректность данных (длины не превышают 10,000 символов).
5. Проверяет соответствие строки `string` шаблону `pattern`, используя динамическое программирование.
6. Записывает результат проверки (`YES` или `NO`) в файл `output.txt` через функцию `write\_file`.
7. Выводит результат на экран и отображает содержимое выходного файла.
8. Измеряет и выводит время выполнения программы.
9. Измеряет и выводит максимальный объём использованной памяти.
10. Останавливает отслеживание использования памяти.

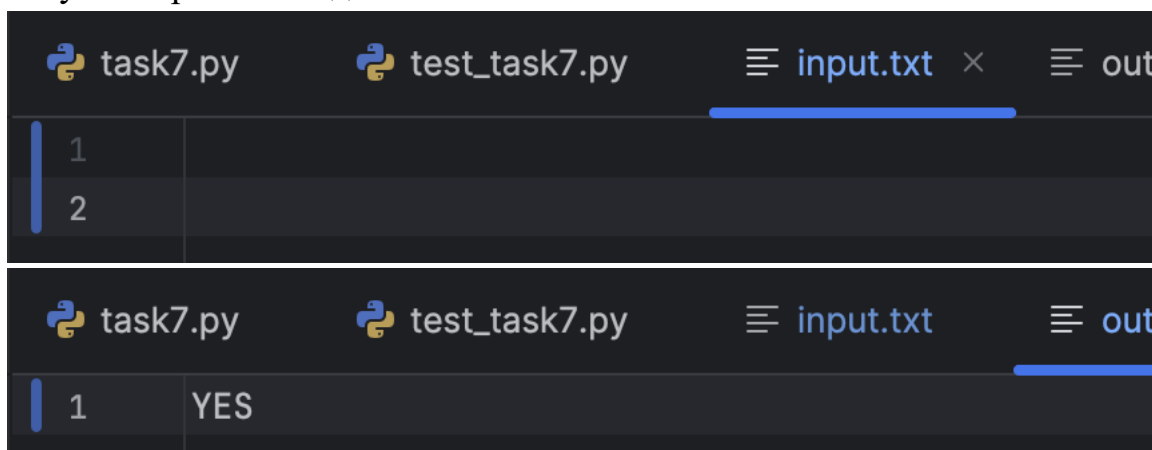
Результат работы кода на примере из текста задачи:



```
task7.py test_task7.py input.txt x out
1 k?t?n
2 kitten

task7.py test_task7.py input.txt out
1 NO
```

Результат работы кода на максимальных и минимальных значениях:



```
task7.py test_task7.py input.txt x out
1 
2 

task7.py test_task7.py input.txt out
1 YES
```

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.00047566700959578156 секунд	15249 байт
Пример из задачи	0.001013792003504932 секунд	15375 байт
Верхняя граница диапазона значений входных данных из текста задачи	0.1820174169843085 секунд	10481917 байт

Вывод по задаче: мною был изучен алгоритм реализации проверки строки на соответствие шаблону.

# Дополнительные задачи

## Задача №4. Наибольшая общая подпоследовательность двух последовательностей

Текст задачи.

### 4 задача. Наибольшая общая подпоследовательность двух последовательностей

Вычислить длину самой длинной общей подпоследовательности из двух последовательностей.

Даны две последовательности  $A = (a_1, a_2, \dots, a_n)$  и  $B = (b_1, b_2, \dots, b_m)$ , найти длину их самой длинной общей подпоследовательности, т.е. наибольшее неотрицательное целое число  $p$  такое, что существуют индексы  $1 \leq i_1 < i_2 < \dots < i_p \leq n$  и  $1 \leq j_1 < j_2 < \dots < j_p \leq m$  такие, что  $a_{i_1} = b_{j_1}, \dots, a_{i_p} = b_{j_p}$ .

- **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.
- Ограничения:  $1 \leq n, m \leq 100$ ;  $-10^9 < a_i, b_i < 10^9$ .
- **Формат вывода / выходного файла (output.txt).** Выведите число  $p$ .
- Ограничение по времени. 1 сек.
- Примеры:

input.txt	output.txt	input.txt	output.txt	input.txt	output.txt
3	2	1	0	4	2
2 7 5		7		2 7 8 3	
2		4		4	
2 5		1 2 3 4		5 2 8 7	

- В первом примере одна общая подпоследовательность – (2, 5) длиной 2, во втором примере две последовательности не имеют одинаковых элементов. В третьем примере - длина 2, последовательности – (2, 7) или (2, 8).

Листинг кода.

```
# Импортируем библиотеки для отслеживания памяти и времени выполнения программы
import tracemalloc
import time
from lab7.utils import *

# Запускаем таймер для измерения времени работы программы
t_start = time.perf_counter()

# Включаем отслеживание памяти
```

```

tracemalloc.start()

current_dir = os.path.dirname(os.path.abspath(__file__)) # Директория
task/src
txtf_dir = os.path.join(os.path.dirname(current_dir), "txtf") # Директория
task/txtf
input_path = os.path.join(txtf_dir, "input.txt")

def longest_common_subsequence(n, A, m, B):
    """
    Вычисляет длину самой длинной общей подпоследовательности для
    последовательностей A и B.
    """
    # Инициализация DP-таблицы
    dp = [[0] * (m + 1) for _ in range(n + 1)]

    # Заполнение таблицы
    for i in range(1, n + 1):
        for j in range(1, m + 1):
            if A[i - 1] == B[j - 1]: # Элементы совпадают
                dp[i][j] = dp[i - 1][j - 1] + 1
            else: # Элементы не совпадают
                dp[i][j] = max(dp[i - 1][j], dp[i][j - 1])

    # Результат — последняя ячейка таблицы
    return dp[n][m]

# Основной блок программы
if __name__ == "__main__":
    # Читаем данные из файла input.txt с помощью функции open_file
    input_data = open_file(input_path)

    # Преобразуем входные данные
    n = int(input_data[0].strip()) # Длина первой последовательности
    A = list(map(int, input_data[1].strip().split())) # Первая
последовательность
    m = int(input_data[2].strip()) # Длина второй последовательности
    B = list(map(int, input_data[3].strip().split())) # Вторая
последовательность

    # Проверка корректности входных данных
    if (1 <= n <= 100) and (1 <= m <= 100) and (all(-10**9 < x < 10**9 for
x in A + B)):
        print(f"Task 4\nInput:\n{n} {A}\n{m} {B}")
        delete_prev_values(4)

        # Вычисляем длину самой длинной общей подпоследовательности
        result = longest_common_subsequence(n, A, m, B)

        output_path = get_output_path(4)
        # Записываем результат в файл output.txt
        write_file(str(result), output_path)
        print_output_file(4)
    else:
        # Выводим сообщение об ошибке, если данные некорректны
        print("Введите корректные данные")

    # Выводим время работы программы
    print("Время работы: %s секунд" % (time.perf_counter() - t_start))
    # Выводим количество памяти, затраченной на выполнение программы
    print("Затрачено памяти:", tracemalloc.get_traced_memory()[1], "байт")

```

```
# Останавливаем отслеживание памяти
tracemalloc.stop()
```

Текстовое объяснение решения.

1. Импортирует необходимые библиотеки для работы с файлами, измерения времени и памяти.
2. Считывает входные данные из файла `input.txt` через функцию `open\_file`.
3. Преобразует данные в две числовые последовательности A и B.
4. Проверяет корректность входных данных (диапазон значений, размеры последовательностей).
5. Рассчитывает длину самой длинной общей подпоследовательности (LCS) двух последовательностей с использованием динамического программирования.
6. Записывает результат в файл `output.txt` через функцию `write\_file`.
7. Выводит результат на экран и отображает содержимое выходного файла.
8. Измеряет и выводит время выполнения программы.
9. Измеряет и выводит максимальный объём использованной памяти.
10. Останавливает отслеживание использования памяти.

Результат работы кода на примере из задачи:

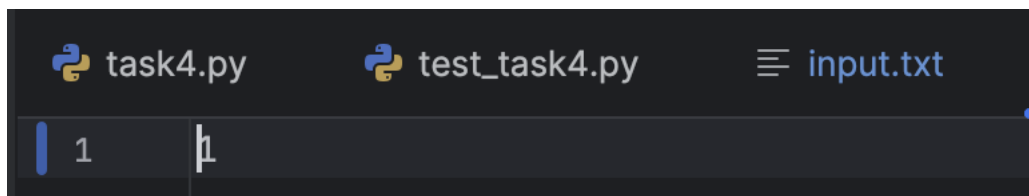
task4.py	test_task4.py	input.txt
1	3	
2	2 7 5	
3	2	
4	2 5	

task4.py	test_task4.py	input.txt
1	2	

Результат работы кода на максимальных и минимальных значениях:

task4.py	test_task4.py	input.txt
1	1	
2	1	
3	1	
4	1	



	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.0006371669878717512 секунд	15410 байт
Пример из задачи	0.0009567080123815686 секунд	15458 байт
Верхняя граница диапазона значений входных данных из текста задачи	0.4109084579977207 секунд	10765530 байт

Вывод по задаче: мною был изучен алгоритм нахождения наибольшей общей подпоследовательности двух последовательностей.

## Задача №6. Наибольшая возрастающая подпоследовательность

Текст задачи.

### 6 задача. 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.
- Пример:

input.txt	output.txt
6	3
3 29 5 5 28 6	3 5 28

Листинг кода.

```
import tracemalloc
import time
from lab7.utils import *
import os
from bisect import bisect_left

# Запуск таймера для измерения времени выполнения программы
t_start = time.perf_counter()

# Включаем отслеживание памяти
tracemalloc.start()

# Определяем пути к файлам
current_dir = os.path.dirname(os.path.abspath(__file__)) # Директория task1/src
txtf_dir = os.path.join(os.path.dirname(current_dir), "txtf") # Директория task1/txtf
input_path = os.path.join(txtf_dir, "input.txt")
```

```

def find_lis(n, sequence):
    """Функция для нахождения LIS и её длины."""
    # Массивы для вычисления LIS
    lis = [] # Концы подпоследовательностей
    parent = [-1] * n # Индексы предыдущих элементов
    indices = [] # Индексы элементов LIS

    # Поиск LIS с использованием бинарного поиска
    for i in range(n):
        pos = bisect_left(lis, sequence[i])
        if pos == len(lis):
            lis.append(sequence[i])
            indices.append(i)
        else:
            lis[pos] = sequence[i]
            indices[pos] = i

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

    # Восстановление LIS
    lis_length = len(lis)
    lis_sequence = []
    current_index = indices[-1]

    for _ in range(lis_length):
        lis_sequence.append(sequence[current_index])
        current_index = parent[current_index]

    lis_sequence.reverse() # Переворачиваем последовательность
    return lis_length, lis_sequence

# Основной блок программы
if __name__ == "__main__":
    # Читаем данные из файла input.txt с помощью функции open_file
    input_data = open_file(input_path)
    n = int(input_data[0][0]) # Количество элементов
    sequence = list(map(int, input_data[1].strip().split())) #
    Последовательность

    # Проверка корректности входных данных
    if (1 <= n <= 10**5) and (all(abs(x) <= 10**9 for x in sequence)):
        print(f"\nTask 6\nInput:\n{n}\n{sequence}")
        delete_prev_values(6)
        # Нахождение LIS
        lis_length, lis_sequence = find_lis(n, sequence)
        output_path = get_output_path(6)
        # Записываем длину LIS и саму последовательность в файл
        write_file(f"{lis_length}\n" + " ".join(map(str, lis_sequence)),
        output_path)
        print_output_file(6)
    else:
        print("Введите корректные данные")

    # Выводим время работы программы
    print("Время работы: %s секунд" % (time.perf_counter() - t_start))
    # Выводим количество памяти, затраченной на выполнение программы
    print("Затрачено памяти:", tracemalloc.get_traced_memory()[1], "байт")

    # Останавливаем отслеживание памяти
    tracemalloc.stop()

```



Текстовое объяснение решения.

1. Импортирует библиотеки для работы с файлами, измерения времени, памяти и алгоритмов.
2. Считывает входные данные из файла `input.txt` через функцию `open\_file`.
3. Преобразует входные данные в количество элементов `n` и последовательность `sequence`.
4. Проверяет корректность данных (длина последовательности и значение элементов).
5. Находит длину и элементы **\*\*наибольшей** возрастающей подпоследовательности (LIS)\*\* с использованием алгоритма, основанного на бинарном поиске.
6. Результат состоит из двух строк: длина LIS и элементы LIS, записываемые в файл `output.txt` через функцию `write\_file`.
7. Выводит входные данные, результат и содержимое выходного файла на экран.
8. Измеряет и выводит время выполнения программы.
9. Измеряет и выводит максимальный объём использованной памяти.
10. Останавливает отслеживание использования памяти.

Результат работы кода на примере из задачи:

task6.py		test_task6.py		input.txt		output.txt	
1	6						
2	3 29 5 5 28 6						

task6.py		test_task6.py		input.txt		output.txt	
1	3						
2	3 5 6						

Результат работы кода на максимальных и минимальных значениях:

task6.py		test_task6.py		input.txt		output.txt	
1	1						
2	1						

```

task6.py  test_task6.py  input.txt  out
1 1
2 1

```

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.0006219159986358136 секунд	15324 байт
Пример из задачи	0.0008923330169636756 секунд	15378 байт
Верхняя граница диапазона значений входных данных из текста задачи	0.4109084579977207 секунд	10765530 байт

Вывод по задаче: мною был изучен алгоритм нахождения наибольшей возрастающей подпоследовательности.

## Вывод

В ходе лабораторной работы я узнал про динамическое программирование.