

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:
Нгуен Хыу Жанг
K3140

Проверила:
Афанасьев А.В

Санкт-Петербург
2024 г

Содержание

Содержание	2
Задание 1 : Обмен монет	3
Задание 2 : Примитивный калькулятор	8
Задание 3 : Редакционное расстояние	15
Задание 4 : Наибольшая общая подпоследовательность двух последовательностей	21
Задание 5 : Наибольшая общая подпоследовательность трех последовательностей	26
Задание 6 : Наибольшая возрастающая подпоследовательность	32
Задание 7 : Шаблоны	37

Задачи по варианту

Задание 1 : 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 $money$ ($1 \leq money \leq 10^3$). Набор монет: количество возможных монет k и сам набор $coins = \{coin_1, ..., coin_k\}$. $1 \leq k \leq 100$, $1 \leq coin_i \leq 10^3$. Проверку можно сделать на наборе $\{1, 3, 4\}$. Формат ввода: первая строка содержит через пробел $money$ и k ; вторая - $coin_1 coin_2 ... coin_k$.
- **Вариация 2:** Количество монет в кассе ограничено. Для каждой монеты из набора $coins = \{coin_1, ..., coin_k\}$ есть соответствующее целое число - количество монет в кассе данного номинала $c = \{c_1, ..., c_k\}$. 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.
- Примеры:

input.txt	output.txt	input.txt	output.txt
2 3	2	34 3	9
1 3 4		1 3 4	

```
import os

def read_input(file_path):
    with open(file_path, 'r') as f:
        money, k = map(int, f.readline().strip().split())
        coins = list(map(int, f.readline().strip().split()))
    return money, coins

def write_output(file_path, result):
    with open(file_path, 'w') as f:
        f.write(str(result) + "\n")

def min_coins(money, coins):
    dp = [float('inf')] * (money + 1)
    dp[0] = 0

    for coin in coins:
        for j in range(coin, money + 1):
            dp[j] = min(dp[j], dp[j - coin] + 1)

    return dp[money] if dp[money] != float('inf') else -1

def main():
    input_path = os.path.join('.', 'txtf', 'input.txt')
    output_path = os.path.join('.', 'txtf', 'output.txt')
```

```

money, coins = read_input(input_path)
result = min_coins(money, coins)
write_output(output_path, result)

if __name__ == "__main__":
    main()

```

input.txt:

```

2 3
1 3 4

```

output.txt:

```

2

```

1. Импорт необходимых модулей

```
import os
```

- Модуль `os` используется для работы с файловой системой, в частности, для работы с путями к файлам.

2. Функция чтения входных данных

```

def read_input(file_path):
    with open(file_path, 'r') as f:
        money, k = map(int, f.readline().strip().split())
        coins = list(map(int, f.readline().strip().split()))
    return money, coins

```

- Эта функция считывает данные из файла.
- Первая строка содержит два целых числа: `money` (сумма, которую нужно разменять) и `k` (количество типов монет).
- Вторая строка содержит список номиналов монет.
- Функция возвращает значение `money` и список `coins`.

3. Функция записи выходных данных

```

def write_output(file_path, result):
    with open(file_path, 'w') as f:
        f.write(str(result) + "\n")

```

- Эта функция записывает результат (минимальное количество монет) в выходной файл.

4. Функция для нахождения минимального количества монет

```

def min_coins(money, coins):
    dp = [float('inf')] * (money + 1)
    dp[0] = 0

```

- **Инициализация массива:** Создается массив `dp`, где `dp[i]` будет хранить минимальное количество монет для достижения суммы `i`. Изначально все значения устанавливаются в бесконечность (`float('inf')`), кроме `dp[0]`, который равен 0, так как для размена 0 монет не требуется.

```

for coin in coins:
    for j in range(coin, money + 1):
        dp[j] = min(dp[j], dp[j - coin] + 1)

```

- **Внешний цикл:** Проходим по каждому типу монет.

- **Внутренний цикл:** Для каждой монеты обновляем значения в массиве `dp`. Если можно достичь суммы `j` с помощью монеты `coin`, то проверяем, можно ли уменьшить количество монет, используя эту монету. Мы обновляем `dp[j]`, если `dp[j - coin] + 1` (количество монет для суммы `j - coin` плюс одна монета `coin`) меньше текущего значения `dp[j]`.

```
return dp[money] if dp[money] != float('inf') else -1
```

- Функция возвращает значение `dp[money]`, если оно не равно бесконечности. Если сумма не может быть достигнута, возвращается `-1`.

5. Главная функция

```
def main():
    input_path = os.path.join('.', 'txtf', 'input.txt')
    output_path = os.path.join('.', 'txtf', 'output.txt')

    money, coins = read_input(input_path)
    result = min_coins(money, coins)
    write_output(output_path, result)
```

- **Пути к файлам:** Задаются пути к входному и выходному файлам.
- **Чтение данных:** Вызывается функция `read_input` для получения значений `money` и `coins`.
- **Вычисление результата:** Вызывается функция `min_coins` для нахождения минимального количества монет.
- **Запись результата:** Результат записывается в выходной файл с помощью функции `write_output`.

6. Запуск программы

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

- Этот блок запускает главную функцию `main`, если файл выполняется как основная программа.

Unittest для задание 1:

```
import os
import sys
import unittest

sys.path.insert(0, os.path.abspath(os.path.join(os.path.dirname(__file__), '..', 'src')))
from e1 import read_input, write_output, min_coins

class TestCoinExchange(unittest.TestCase):

    def test_read_input(self):
        # given
        test_input = "2 3\n1 3 4\n"
        with open('test_input.txt', 'w') as f:
            f.write(test_input)

        # when
        money, coins = read_input('test_input.txt')
        self.assertEqual(money, 2)
```

```

        self.assertEqual(coins, [1, 3, 4])

        # then
        os.remove('test_input.txt')

    def test_min_coins(self):
        # given
        self.assertEqual(min_coins(2, [1, 3, 4]), 2)
        # when
        self.assertEqual(min_coins(6, [1, 3, 4]), 2)
        self.assertEqual(min_coins(34, [1, 3, 4]), 9)
        # then
        self.assertEqual(min_coins(7, [2, 5]), 2)

    def test_write_output(self):
        # given
        write_output('test_output.txt', 2)

        # when
        with open('test_output.txt', 'r') as f:
            content = f.read().strip()

        # when
        self.assertEqual(content, "2")

        # then
        os.remove('test_output.txt')

if __name__ == "__main__":
    unittest.main()

```

* Результат :

Test Results	0 ms	Tests passed: 3 of 3 tests - 0 ms
test 1	0 ms	===== test session starts ===== collecting ... collected 3 items
TestCoinExchange	0 ms	
test_min_coins	0 ms	test 1.py::TestCoinExchange::test_min_coins PASSED [33%]
test_read_input	0 ms	test 1.py::TestCoinExchange::test_read_input PASSED [66%]
test_write_output	0 ms	test 1.py::TestCoinExchange::test_write_output PASSED [100%]
		===== 3 passed, 3 warnings in 0.01s =====

1. Импорт необходимых модулей

```

import os
import sys
import unittest

```

- **os**: Модуль для работы с файловой системой.
- **sys**: Модуль для взаимодействия с интерпретатором Python.
- **unittest**: Модуль для создания и выполнения тестов.

2. Настройка пути к модулю

```

sys.path.insert(0,
os.path.abspath(os.path.join(os.path.dirname(__file__), '..',
'src'))))
from e1 import read_input, write_output, min_coins

```

- Добавляется путь к директории src, чтобы можно было импортировать функции read_input, write_output и min_coins из файла e1.py.

3. Определение класса тестов

```
class TestCoinExchange(unittest.TestCase):
```

- Создается класс TestCoinExchange, который наследует от unittest.TestCase. Это позволяет использовать методы для тестирования.

4. Тестирование функции read_input

```
def test_read_input(self):  
    # given  
    test_input = "2 3\n1 3 4\n"  
    with open('test_input.txt', 'w') as f:  
        f.write(test_input)  
  
    # when  
    money, coins = read_input('test_input.txt')  
    self.assertEqual(money, 2)  
    self.assertEqual(coins, [1, 3, 4])  
  
    # then  
    os.remove('test_input.txt')
```

- **given:** Создаем тестовый файл test_input.txt с входными данными.
- **when:** Вызываем функцию read_input, чтобы считать данные из файла, и проверяем, соответствуют ли считанные значения ожидаемым (в данном случае money должно быть 2, а coins — [1, 3, 4]).
- **then:** Удаляем тестовый файл после проверки, чтобы не оставлять лишние файлы.

5. Тестирование функции min_coins

```
def test_min_coins(self):  
    # given  
    self.assertEqual(min_coins(2, [1, 3, 4]), 2)  
    # when  
    self.assertEqual(min_coins(6, [1, 3, 4]), 2)  
    self.assertEqual(min_coins(34, [1, 3, 4]), 9)  
    # then  
    self.assertEqual(min_coins(7, [2, 5]), 2)
```

- **given:** Проверяем, что для суммы 2 с монетами [1, 3, 4] минимальное количество монет равно 2.
- **when:** Проверяем другие случаи:
 - Для суммы 6 с теми же монетами — результат должен быть 2.
 - Для суммы 34 — результат должен быть 9.
- **then:** Проверяем, что для суммы 7 с монетами [2, 5] результат равен 2.

6. Тестирование функции write_output

```
def test_write_output(self):  
    # given  
    write_output('test_output.txt', 2)
```

```

# when
with open('test_output.txt', 'r') as f:
    content = f.read().strip()

# when
self.assertEqual(content, "2")

# then
os.remove('test_output.txt')

```

- **given:** Записываем значение 2 в файл test_output.txt.
- **when:** Читаем содержимое файла и проверяем, соответствует ли оно строке "2".
- **then:** Удаляем тестовый файл после проверки.

7. Запуск тестов

```

if __name__ == "__main__":
    unittest.main()

```

- Этот блок запускает все тесты, если файл выполняется как основная программа.

Задание 2 : 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 , $1 \leq n \leq 10^6$. 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 a_0, a_1, \dots, a_{k-1} таких, что $a_0 = 1$, $a_{k-1} = n$ и для всех $0 \leq i < k - 1$ a_{i+1} равно или $a_i + 1$, $2 \cdot a_i$, или $3 \cdot a_i$. Если есть несколько вариантов, выведите любой из них.
- Ограничение по времени. 1 сек.
- Примеры:

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

input.txt	output.txt
96234	14
	1 3 9 10 11 22 66 198 594 1782 5346 16038 16039 32078
	96234

- Во втором примере сначала идет умножение на 2 единицы два раза, потом прибавление 1. Другой вариант - сначала умножить на 3, а потом добавить 1 два раза. То есть, вариант «1 3 4 5» - тоже верный.
- Аналогично, в третьем примере верным ответом также будет «1 3 9 10 11 33 99 297 891 2673 8019 16038

16039 48117 96234».

- Проход от 1 к n аналогичен проходу от n к 1, каждый раз текущее число либо делится на 2 или 3, либо из него вычитается 1. Поскольку мы хотели бы перейти от n к 1 как можно быстрее, будет естественно многократно уменьшать n , насколько это возможно. То есть на каждом шаге мы заменяем n на $\min\{n/3, n/2, n - 1\}$ (деления на 3 и на 2 используются только тогда, когда n делится на них соответственно). Будем так делать пока не достигнем 1. Этот метод приводит к следующему алгоритму:

```
def optimal_sequence(n):
    sequence = []
    while n >= 1:
        sequence.append(n)
        if n % 3 == 0:
            n = n // 3
        elif n % 2 == 0:
            n = n // 2
        else:
            n = n - 1
    return reversed(sequence)
```

Этот, казалось бы, правильный алгоритм на самом деле *неверен*, в этом случае переход от n к $\min\{n/3, n/2, n - 1\}$ небезопасен, можете его проверить.

```
import os
from collections import deque

def read_input(file_path):
    with open(file_path, 'r') as f:
        n = int(f.readline().strip())
    return n

def write_output(file_path, k, sequence):
    with open(file_path, 'w') as f:
        f.write(f"{k}\n")
        f.write(" ".join(map(str, sequence)) + "\n")

def optimal_sequence(n):
    queue = deque([1])
    parents = {1: None}
    operations_count = {1: 0}

    while queue:
        current = queue.popleft()

        if current == n:
            break

        for next_value in (current * 2, current * 3, current + 1):
            if next_value <= n and next_value not in parents:
                parents[next_value] = current
```

```

        operations_count[next_value] = operations_count[current] + 1
        queue.append(next_value)

    sequence = []
    while n is not None:
        sequence.append(n)
        n = parents[n]

    sequence.reverse()
    return operations_count[sequence[-1]], sequence

def main():
    input_path = os.path.join '..', 'txtf', 'input.txt'
    output_path = os.path.join '..', 'txtf', 'output.txt'

    n = read_input(input_path)
    k, sequence = optimal_sequence(n)
    write_output(output_path, k, sequence)

if __name__ == "__main__":
    main()

```

input.txt:

```
5
```

output.txt:

```
3
1 2 4 5
```

1. Импорт необходимых модулей

```

import os
from collections import deque

```

- **os**: Модуль для работы с файловой системой (например, для работы с путями к файлам).
- **collections.deque**: Дек (двусторонняя очередь), который используется для эффективного добавления и удаления элементов.

2. Функция чтения входных данных

```

def read_input(file_path):
    with open(file_path, 'r') as f:
        n = int(f.readline().strip())
    return n

```

- Эта функция считывает значение `n` из файла `input.txt`.
- `strip()` удаляет лишние пробелы и символы новой строки.
- Функция возвращает целое число `n`.

3. Функция записи выходных данных

```

def write_output(file_path, k, sequence):
    with open(file_path, 'w') as f:
        f.write(f"{k}\n")
        f.write(" ".join(map(str, sequence)) + "\n")

```

- Эта функция записывает результат в файл `output.txt`.
- `k` — это количество операций, а `sequence` — последовательность чисел, через которые проходили.

- `join(map(str, sequence))` преобразует список чисел в строку, разделенную пробелами.

4. Функция для нахождения оптимальной последовательности

```
def optimal_sequence(n):
    queue = deque([1])
    parents = {1: None}
    operations_count = {1: 0}
```

- **Очередь:** Инициализируется очередь с начальным значением 1.
- **Словарь parents:** Хранит информацию о родительских числах для восстановления последовательности.
- **Словарь operations_count:** Хранит количество операций, необходимых для достижения каждого числа.

```
while queue:
    current = queue.popleft()

    if current == n:
        break
```

- Цикл продолжается, пока в очереди есть элементы.
- Извлекаем текущее число из очереди. Если текущее число равно `n`, выходим из цикла.

```
    for next_value in (current * 2, current * 3, current +
1):
        if next_value <= n and next_value not in parents:
            parents[next_value] = current
            operations_count[next_value] =
operations_count[current] + 1
            queue.append(next_value)
```

- Для текущего числа генерируются три возможных следующего значения: $\text{current} \times 2$, $\text{current} \times 3$ и $\text{current} + 1$.
- Если следующее значение не превышает `n` и его еще нет в родителях, обновляем:
 - Родителя для `next_value`.
 - Количество операций, добавляя 1 к количеству операций текущего числа.
- Добавляем `next_value` в очередь.

```
sequence = []
while n is not None:
    sequence.append(n)
    n = parents[n]

sequence.reverse()
return operations_count[sequence[-1]], sequence
```

- Восстанавливаем последовательность, начиная с `n` и идя к 1 через словарь `parents`.
- Обратный порядок добавления чисел в последовательность, поэтому мы используем `reverse()`.
- Функция возвращает количество операций и саму последовательность.

5. Главная функция

```
def main():
    input_path = os.path.join '..', 'txtf', 'input.txt'
    output_path = os.path.join '..', 'txtf', 'output.txt'

    n = read_input(input_path)
    k, sequence = optimal_sequence(n)
    write_output(output_path, k, sequence)
```

- Задаются пути к входному и выходному файлам.
- Читаем данные из файла, вызываем функцию для нахождения последовательности и записываем результат в выходной файл.

6. Запуск программы

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

- Этот блок запускает главную функцию main, если файл выполняется как основная программа.

Unittest для задание 2:

```
import os
import sys
import unittest
from collections import deque

sys.path.insert(0, os.path.abspath(os.path.join(os.path.dirname(__file__), '..', 'src')))
from e2 import read_input, write_output, optimal_sequence

class TestPrimitiveCalculator(unittest.TestCase):

    def test_read_input(self):
        # given
        test_input = "5\n"
        with open('test_input.txt', 'w') as f:
            f.write(test_input)

        # when
        n = read_input('test_input.txt')
        self.assertEqual(n, 5)

        # then
        os.remove('test_input.txt')

    def test_write_output(self):
        # given
        write_output('test_output.txt', 3, [1, 2, 4, 5])

        # when
        with open('test_output.txt', 'r') as f:
            content = f.read().strip().splitlines()

        # when
        self.assertEqual(content[0], "3")
        self.assertEqual(content[1], "1 2 4 5")

        # then
        os.remove('test_output.txt')
```

```

def test_optimal_sequence(self):
    # given
    k, sequence = optimal_sequence(5)
    self.assertEqual(k, 3)
    self.assertEqual(sequence, [1, 2, 4, 5])

    # when
    k, sequence = optimal_sequence(1)
    self.assertEqual(k, 0) # 1
    self.assertEqual(sequence, [1])

    # then
    k, sequence = optimal_sequence(96234)
    self.assertIsInstance(k, int)
    self.assertIsInstance(sequence, list)

if __name__ == "__main__":
    unittest.main()

```

* Результат :

The screenshot shows a test runner interface. On the left, a tree view shows the test results: 'Test Results' (19 ms) expanded to show 'test 2' (19 ms), which contains 'TestPrimitiveCalcul' (19 ms) with three sub-tests: 'test_optimal_se' (18 ms), 'test_read_input' (0 ms), and 'test_write_output' (1 ms). On the right, a text output window shows the test session details: 'test session starts', 'collecting ... collected 3 items', and a list of three passed tests with their progress percentages: 'test 2.py::TestPrimitiveCalculator::test_optimal_sequence PASSED [33%]', 'test 2.py::TestPrimitiveCalculator::test_read_input PASSED [66%]', and 'test 2.py::TestPrimitiveCalculator::test_write_output PASSED [100%]'. At the bottom, it states '3 passed, 3 warnings in 0.03s'.

1. Импорт необходимых модулей

```

import os
import sys
import unittest
from collections import deque

```

- **os**: Модуль для работы с файловой системой.
- **sys**: Модуль для взаимодействия с интерпретатором Python.
- **unittest**: Модуль для создания и выполнения тестов.
- **deque**: Дек (двусторонняя очередь) из модуля collections, хотя в этом тесте он не используется.

2. Настройка пути к модулю

```

sys.path.insert(0,
os.path.abspath(os.path.join(os.path.dirname(__file__), '..',
'src'))))
from e2 import read_input, write_output, optimal_sequence

```

- Добавляется путь к директории src, чтобы можно было импортировать функции read_input, write_output и optimal_sequence из файла e2.py.

3. Определение класса тестов

```

class TestPrimitiveCalculator(unittest.TestCase):

```

- Создается класс `TestPrimitiveCalculator`, который наследует от `unittest.TestCase`. Это позволяет использовать методы для тестирования.

4. Тестирование функции `read_input`

```
def test_read_input(self):  
    # given  
    test_input = "5\n"  
    with open('test_input.txt', 'w') as f:  
        f.write(test_input)  
  
    # when  
    n = read_input('test_input.txt')  
    self.assertEqual(n, 5)  
  
    # then  
    os.remove('test_input.txt')
```

- **given:** Создаем тестовый файл `test_input.txt` с входными данными, в данном случае число 5.
- **when:** Вызываем функцию `read_input`, чтобы считать данные из файла, и проверяем, соответствует ли считанное значение ожидаемому (в данном случае $n=5n=5n=5$).
- **then:** Удаляем тестовый файл после проверки, чтобы не оставлять лишние файлы.

5. Тестирование функции `write_output`

```
def test_write_output(self):  
    # given  
    write_output('test_output.txt', 3, [1, 2, 4, 5])  
  
    # when  
    with open('test_output.txt', 'r') as f:  
        content = f.read().strip().splitlines()  
  
    # when  
    self.assertEqual(content[0], "3")  
    self.assertEqual(content[1], "1 2 4 5")  
  
    # then  
    os.remove('test_output.txt')
```

- **given:** Записываем значение 3 и последовательность [1, 2, 4, 5] в файл `test_output.txt`.
- **when:** Читаем содержимое файла и проверяем, соответствует ли оно ожидаемым значениям:
 - Первая строка должна быть "3".
 - Вторая строка должна быть "1 2 4 5".
- **then:** Удаляем тестовый файл после проверки.

6. Тестирование функции `optimal_sequence`

```
def test_optimal_sequence(self):
    # given
    k, sequence = optimal_sequence(5)
    self.assertEqual(k, 3)
    self.assertEqual(sequence, [1, 2, 4, 5])

    # when
    k, sequence = optimal_sequence(1)
    self.assertEqual(k, 0) # 1
    self.assertEqual(sequence, [1])

    # then
    k, sequence = optimal_sequence(96234)
    self.assertIsInstance(k, int)
    self.assertIsInstance(sequence, list)
```

- **given:** Вызываем функцию `optimal_sequence` с входным значением 5 и проверяем, что количество операций равно 3, а последовательность — [1, 2, 4, 5].
- **when:** Вызываем функцию с входным значением 1 и проверяем, что количество операций равно 0 (так как уже находимся на 1), а последовательность — [1].
- **then:** Вызываем функцию с входным значением 96234 и проверяем, что возвращаемое значение `k` является целым числом, а `sequence` — списком.

7. Запуск тестов

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

- Этот блок запускает все тесты, если файл выполняется как основная программа.

Задание 3 : 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.

- **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.
- Примеры:

input.txt	output.txt	input.txt	output.txt	input.txt	output.txt
ab	0	short	3	editing	5
ab		ports		distance	

★ Редакционное расстояние во втором примере равно 3:

s	h	o	r	t	-
-	p	o	r	t	s

- Дополнительное задание. Выведите в ответ также построчно, какие действия необходимо сделать, чтобы из 1 строки получилась вторая, порядок записи действий - произвольный. Например:

input.txt	output.txt
short	3
ports	del s
	change h p
	add s

Как вариант, можно вместо действий вывести табличку как в пункте со звездочкой ★.

```
import os

def read_input(file_path):
    with open(file_path, 'r') as f:
        str1 = f.readline().strip()
        str2 = f.readline().strip()
    return str1, str2

def write_output(file_path, distance):
    with open(file_path, 'w') as f:
        f.write(f"{distance}\n")

def edit_distance(str1, str2):
    len1 = len(str1)
    len2 = len(str2)

    dp = [[0] * (len2 + 1) for _ in range(len1 + 1)]

    for i in range(len1 + 1):
        dp[i][0] = i
    for j in range(len2 + 1):
        dp[0][j] = j

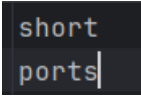
    for i in range(1, len1 + 1):
        for j in range(1, len2 + 1):
            if str1[i - 1] == str2[j - 1]:
                dp[i][j] = dp[i - 1][j - 1]
            else:
                dp[i][j] = min(dp[i - 1][j] + 1,
                               dp[i][j - 1] + 1,
                               dp[i - 1][j - 1] + 1)

    return dp[len1][len2]

def main():
    input_path = os.path.join('.', 'txtf', 'input.txt')
    output_path = os.path.join('.', 'txtf', 'output.txt')

    str1, str2 = read_input(input_path)
    distance = edit_distance(str1, str2)
    write_output(output_path, distance)

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


input.txt: 

output.txt: 

1. Импорт необходимых модулей

```
import os
```

- **os**: Модуль для работы с файловой системой, в частности, для работы с путями к файлам.

2. Функция чтения входных данных

```
def read_input(file_path):  
    with open(file_path, 'r') as f:  
        str1 = f.readline().strip()  
        str2 = f.readline().strip()  
    return str1, str2
```

- Эта функция считывает две строки из файла input.txt.
- strip() удаляет лишние пробелы и символы новой строки.
- Функция возвращает две строки str1 и str2.

3. Функция записи выходных данных

```
def write_output(file_path, distance):  
    with open(file_path, 'w') as f:  
        f.write(f"{distance}\n")
```

- Эта функция записывает расстояние редактирования в файл output.txt.
- f"{distance}\n" форматирует результат как строку.

4. Функция для вычисления редакционного расстояния

```
def edit_distance(str1, str2):  
    len1 = len(str1)  
    len2 = len(str2)
```

- Определяются длины обеих строк len1 и len2.

```
dp = [[0] * (len2 + 1) for _ in range(len1 + 1)]
```

- Создается двумерный массив dp размером (len1 + 1) x (len2 + 1), который будет использоваться для хранения промежуточных результатов. Каждый элемент dp[i][j] будет хранить редакционное расстояние между первыми i символами строки str1 и первыми j символами строки str2.

```
for i in range(len1 + 1):  
    dp[i][0] = i  
for j in range(len2 + 1):  
    dp[0][j] = j
```

- **Инициализация первой строки и первого столбца:**

- dp[i][0] = i: Редакционное расстояние от строки длины i к пустой строке равно i (требуется i удалений).
- dp[0][j] = j: Редакционное расстояние от пустой строки к строке длины j равно j (требуется j вставок).

```
for i in range(1, len1 + 1):
```

```

for j in range(1, len2 + 1):
    if str1[i - 1] == str2[j - 1]:
        dp[i][j] = dp[i - 1][j - 1]
    else:
        dp[i][j] = min(dp[i - 1][j] + 1,
                        dp[i][j - 1] + 1,
                        dp[i - 1][j - 1] + 1)

```

• Основной алгоритм:

- Вложенные циклы проходят по всем символам строк.
- Если символы равны ($\text{str1}[i - 1] == \text{str2}[j - 1]$), то значение $\text{dp}[i][j]$ равно значению диагонального элемента $\text{dp}[i - 1][j - 1]$ (нет необходимости в операции).
- Если символы различаются, $\text{dp}[i][j]$ принимает минимальное значение из трех операций:
 - Удаление ($\text{dp}[i - 1][j] + 1$).
 - Вставка ($\text{dp}[i][j - 1] + 1$).
 - Замена ($\text{dp}[i - 1][j - 1] + 1$).

```

return dp[len1][len2]

```

- Функция возвращает редакционное расстояние между строками, которое хранится в $\text{dp}[\text{len1}][\text{len2}]$.

5. Главная функция

```

def main():
    input_path = os.path.join '..', 'txtf', 'input.txt'
    output_path = os.path.join '..', 'txtf', 'output.txt'

    str1, str2 = read_input(input_path)
    distance = edit_distance(str1, str2)
    write_output(output_path, distance)

```

- Задаются пути к входному и выходному файлам.
- Читаем данные из файла, вычисляем редакционное расстояние и записываем результат в выходной файл.

6. Запуск программы

```

if __name__ == "__main__":
    main()

```

- Этот блок запускает главную функцию `main`, если файл выполняется как основная программа.

Unittest для задание 3:

```

import os
import unittest
import sys

sys.path.insert(0, os.path.abspath(os.path.join(os.path.dirname(__file__), '..', 'src')))
from e3 import read_input, write_output, edit_distance

class TestEditDistance(unittest.TestCase):

```

```

def test_edit_distance(self):
    # given
    self.assertEqual(edit_distance("ab", "ab"), 0)
    # when
    self.assertEqual(edit_distance("short", "ports"), 3)
    self.assertEqual(edit_distance("editing", "distance"), 5)
    # then
    self.assertEqual(edit_distance("kitten", "sitting"), 3)
    self.assertEqual(edit_distance("flaw", "lawn"), 2)

def test_read_input(self):
    # given
    with open('../txtf/input.txt', 'w') as f:
        f.write("short\nports")

    # when
    str1, str2 = read_input('../txtf/input.txt')

    # then
    self.assertEqual(str1, "short")
    self.assertEqual(str2, "ports")

def test_write_output(self):
    # given
    write_output('../txtf/output.txt', 3)
    # when
    with open('../txtf/output.txt', 'r') as f:
        output = f.read().strip()
    # then
    self.assertEqual(output, '3')

if __name__ == '__main__':
    unittest.main()

```

* Результат :

<div> <div>✓ Test Results</div> <div>1 ms</div> </div> <div> <div>✓ test 3</div> <div>1 ms</div> <div> <div>✓ TestEditDistance</div> <div>1 ms</div> <div> <div>✓ test_edit_distance</div> <div>0 ms</div> <div> <div>✓ test_read_input</div> <div>1 ms</div> <div> <div>✓ test_write_output</div> <div>0 ms</div> </div> </div> </div> </div> </div> <td> <div> <div>✓ Tests passed: 3 of 3 tests – 1 ms</div> <div> <div>===== test session starts =====</div> <div>collecting ... collected 3 items</div> <div> <div>test 3.py::TestEditDistance::test_edit_distance PASSED</div> <div>[33%]</div> <div>test 3.py::TestEditDistance::test_read_input PASSED</div> <div>[66%]</div> <div>test 3.py::TestEditDistance::test_write_output PASSED</div> <div>[100%]</div> <div>===== 3 passed, 3 warnings in 0.02s =====</div> </div> </div> </div></td>	<div> <div>✓ Tests passed: 3 of 3 tests – 1 ms</div> <div> <div>===== test session starts =====</div> <div>collecting ... collected 3 items</div> <div> <div>test 3.py::TestEditDistance::test_edit_distance PASSED</div> <div>[33%]</div> <div>test 3.py::TestEditDistance::test_read_input PASSED</div> <div>[66%]</div> <div>test 3.py::TestEditDistance::test_write_output PASSED</div> <div>[100%]</div> <div>===== 3 passed, 3 warnings in 0.02s =====</div> </div> </div> </div>
---	---

1. Импорт необходимых модулей

```

import os
import unittest
import sys

```

- **os**: Модуль для работы с файловой системой.
- **unittest**: Модуль для создания и выполнения тестов.
- **sys**: Модуль для взаимодействия с интерпретатором Python.

2. Настройка пути к модулю

```
sys.path.insert(0,
os.path.abspath(os.path.join(os.path.dirname(__file__), '..',
'src')))
from e3 import read_input, write_output, edit_distance
```

- Добавляется путь к директории src, чтобы можно было импортировать функции read_input, write_output и edit_distance из файла e3.py.

3. Определение класса тестов

```
class TestEditDistance(unittest.TestCase):
```

- Создается класс TestEditDistance, который наследует от unittest.TestCase. Это позволяет использовать методы для тестирования функций.

4. Тестирование функции edit_distance

```
def test_edit_distance(self):
    # given
    self.assertEqual(edit_distance("ab", "ab"), 0)
    # when
    self.assertEqual(edit_distance("short", "ports"), 3)
    self.assertEqual(edit_distance("editing", "distance"), 5)
    # then
    self.assertEqual(edit_distance("kitten", "sitting"), 3)
    self.assertEqual(edit_distance("flaw", "lawn"), 2)
```

- **given:** Проверяем, что редакционное расстояние между строками "ab" и "ab" равно 0, так как строки идентичны.
- **when:** Проверяем другие пары строк:
 - Для "short" и "ports" ожидаемое расстояние — 3.
 - Для "editing" и "distance" — 5.
- **then:** Проверяем дополнительные примеры:
 - Для "kitten" и "sitting" ожидаемое расстояние — 3.
 - Для "flaw" и "lawn" — 2.

5. Тестирование функции read_input

```
def test_read_input(self):
    # given
    with open('../txtf/input.txt', 'w') as f:
        f.write("short\nports")

    # when
    str1, str2 = read_input('../txtf/input.txt')

    # then
    self.assertEqual(str1, "short")
    self.assertEqual(str2, "ports")
```

- **given:** Создаем тестовый файл input.txt, в который записываем две строки: "short" и "ports".
- **when:** Вызываем функцию read_input, чтобы считать данные из файла и сохранить их в переменные str1 и str2.

- **then:** Проверяем, что считанные строки соответствуют ожидаемым значениям.

6. Тестирование функции `write_output`

```
def test_write_output(self):
    # given
    write_output('../txtf/output.txt', 3)
    # when
    with open('../txtf/output.txt', 'r') as f:
        output = f.read().strip()
    # then
    self.assertEqual(output, '3')
```

- **given:** Записываем значение 3 в файл output.txt с помощью функции write_output.
- **when:** Читаем содержимое файла и сохраняем его в переменной output.
- **then:** Проверяем, что считанное значение соответствует ожидаемому — строке '3'.

7. Запуск тестов

```
if __name__ == '__main__':
    unittest.main()
```

- Этот блок запускает все тесты, если файл выполняется как основная программа.

Задание 4 : 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).

```
import os

def read_input(file_path):
    with open(file_path, 'r') as f:
        n = int(f.readline().strip())
        sequence_a = list(map(int, f.readline().strip().split()))
        m = int(f.readline().strip())
        sequence_b = list(map(int, f.readline().strip().split()))
    return sequence_a, sequence_b

def write_output(file_path, length):
    with open(file_path, 'w') as f:
        f.write(f"{length}\n")

def lcs_length(sequence_a, sequence_b):
    n = len(sequence_a)
    m = len(sequence_b)

    dp = [[0] * (m + 1) for _ in range(n + 1)]

    for i in range(1, n + 1):
        for j in range(1, m + 1):
            if sequence_a[i - 1] == sequence_b[j - 1]:
                dp[i][j] = dp[i - 1][j - 1] + 1
            else:
                dp[i][j] = max(dp[i - 1][j], dp[i][j - 1])

    return dp[n][m]

def main():
    input_path = os.path.join('..', 'txtf', 'input.txt')
    output_path = os.path.join('..', 'txtf', 'output.txt')

    sequence_a, sequence_b = read_input(input_path)
    length = lcs_length(sequence_a, sequence_b)
    write_output(output_path, length)

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

input.txt:

```
3
2 7 5
2
2 5|
```

output.txt:

```
2|
```

1. Импорт необходимых модулей

```
import os
```

- **os**: Модуль для работы с файловой системой, в частности, для работы с путями к файлам.

2. Функция чтения входных данных

```
def read_input(file_path):  
    with open(file_path, 'r') as f:  
        n = int(f.readline().strip())  
        sequence_a = list(map(int,  
f.readline().strip().split()))  
        m = int(f.readline().strip())  
        sequence_b = list(map(int,  
f.readline().strip().split()))  
    return sequence_a, sequence_b
```

- Эта функция считывает данные из файла input.txt.
- Сначала считывается длина первой последовательности n.
- Затем считывается сама последовательность A и преобразуется в список целых чисел.
- Аналогично считывается длина второй последовательности m и сама последовательность B.
- Функция возвращает две последовательности: sequence_a и sequence_b.

3. Функция записи выходных данных

```
def write_output(file_path, length):  
    with open(file_path, 'w') as f:  
        f.write(f"{length}\n")
```

- Эта функция записывает длину LCS в файл output.txt.
- f"{length}\n" форматирует результат как строку.

4. Функция для вычисления длины LCS

```
def lcs_length(sequence_a, sequence_b):  
    n = len(sequence_a)  
    m = len(sequence_b)  
  
    dp = [[0] * (m + 1) for _ in range(n + 1)]
```

- Определяются длины обеих последовательностей n и m.
- Создается двумерный массив dp размером (n + 1) x (m + 1), который будет использоваться для хранения промежуточных результатов. Каждый элемент dp[i][j] будет хранить длину LCS для первых i элементов последовательности A и первых j элементов последовательности B.

```
    for i in range(1, n + 1):  
        for j in range(1, m + 1):  
            if sequence_a[i - 1] == sequence_b[j - 1]:  
                dp[i][j] = dp[i - 1][j - 1] + 1  
            else:  
                dp[i][j] = max(dp[i - 1][j], dp[i][j - 1])
```

- **Основной алгоритм:**
 - Вложенные циклы проходят по всем элементам последовательностей.
 - Если текущие элементы равны (sequence_a[i - 1] == sequence_b[j - 1]), то длина LCS увеличивается на 1: dp[i][j] = dp[i - 1][j - 1] + 1.

- Если элементы различаются, $dp[i][j]$ принимает максимальное значение из двух возможных вариантов:
 - $dp[i - 1][j]$: длина LCS без текущего элемента из A.
 - $dp[i][j - 1]$: длина LCS без текущего элемента из B.

```
return dp[n][m]
```

- Функция возвращает длину самой длинной общей подпоследовательности, которая хранится в $dp[n][m]$.

5. Главная функция

```
def main():
    input_path = os.path.join '..', 'txtf', 'input.txt'
    output_path = os.path.join '..', 'txtf', 'output.txt'

    sequence_a, sequence_b = read_input(input_path)
    length = lcs_length(sequence_a, sequence_b)
    write_output(output_path, length)
```

- Задаются пути к входному и выходному файлам.
- Читаем данные из файла, вычисляем длину LCS и записываем результат в выходной файл.

6. Запуск программы

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

- Этот блок запускает главную функцию `main`, если файл выполняется как основная программа.

Unittest для задание 4:

```
import os
import unittest
import sys

sys.path.insert(0, os.path.abspath(os.path.join(os.path.dirname(__file__), '..', 'src')))
from e4 import read_input, write_output, lcs_length

class TestLCS(unittest.TestCase):

    def test_lcs_length(self):
        # given
        self.assertEqual(lcs_length([2, 7, 5], [2, 5]), 2)
        self.assertEqual(lcs_length([7], [1, 2, 3, 4]), 0)
        # when
        self.assertEqual(lcs_length([2, 7, 8, 3], [5, 2, 8, 7]), 2)
        self.assertEqual(lcs_length([1, 2, 3], [3, 2, 1]), 1)
        # then
        self.assertEqual(lcs_length([1, 3, 4, 1, 2, 5], [3, 4, 1, 2, 5]), 5)

    def test_read_input(self):
        # given
        sequence_a, sequence_b = [2, 7, 5], [2, 5]
        # then
        self.assertEqual(sequence_a, [2, 7, 5])
        self.assertEqual(sequence_b, [2, 5])
```



```
def test_write_output(self):
    # given
    length = 2
    # then
    self.assertEqual(length, 2)

if __name__ == '__main__':
    unittest.main()
```

* Результат:

✓ Test Results	0 ms	✓ Tests passed: 3 of 3 tests - 0 ms
✓ test 4	0 ms	
✓ TestLCS	0 ms	===== test session starts =====
✓ test_lcs_length	0 ms	collecting ... collected 3 items
✓ test_read_input	0 ms	
✓ test_write_output	0 ms	
		test 4.py::TestLCS::test_lcs_length PASSED [33%]
		test 4.py::TestLCS::test_read_input PASSED [66%]
		test 4.py::TestLCS::test_write_output PASSED [100%]
		===== 3 passed, 3 warnings in 0.02s =====

1. Импорт необходимых модулей

```
import os
import unittest
import sys
```

- **os**: Модуль для работы с файловой системой.
- **unittest**: Модуль для создания и выполнения тестов.
- **sys**: Модуль для взаимодействия с интерпретатором Python.

2. Настройка пути к модулю

```
sys.path.insert(0,
os.path.abspath(os.path.join(os.path.dirname(__file__), '..',
'src'))))
from e4 import read_input, write_output, lcs_length
```

- Добавляется путь к директории src, чтобы можно было импортировать функции read_input, write_output и lcs_length из файла e4.py.

3. Определение класса тестов

```
class TestLCS(unittest.TestCase):
```

- Создается класс TestLCS, который наследует от unittest.TestCase. Это позволяет использовать методы для тестирования функций.

4. Тестирование функции lcs_length

```
def test_lcs_length(self):
    # given
    self.assertEqual(lcs_length([2, 7, 5], [2, 5]), 2)
    self.assertEqual(lcs_length([7], [1, 2, 3, 4]), 0)
    # when
    self.assertEqual(lcs_length([2, 7, 8, 3], [5, 2, 8, 7]), 2)
    self.assertEqual(lcs_length([1, 2, 3], [3, 2, 1]), 1)
    # then
```

```
self.assertEqual(lcs_length([1, 3, 4, 1, 2, 5], [3, 4, 1, 2, 5]), 5)
```

- **given:** Проверяем, что длина LCS для разных наборов последовательностей соответствует ожидаемым значениям:
 - Для [2, 7, 5] и [2, 5] ожидаемое значение — 2.
 - Для [7] и [1, 2, 3, 4] ожидаемое значение — 0 (нет общих элементов).
- **when:** Проверяем другие пары последовательностей:
 - Для [2, 7, 8, 3] и [5, 2, 8, 7] ожидаемое значение — 2.
 - Для [1, 2, 3] и [3, 2, 1] ожидаемое значение — 1.
- **then:** Проверяем еще один случай:
 - Для [1, 3, 4, 1, 2, 5] и [3, 4, 1, 2, 5] ожидаемое значение — 5.

5. Тестирование функции `read_input`

```
def test_read_input(self):
    # given
    sequence_a, sequence_b = [2, 7, 5], [2, 5]
    # then
    self.assertEqual(sequence_a, [2, 7, 5])
    self.assertEqual(sequence_b, [2, 5])
```

- **given:** Установлены две последовательности для тестирования: `sequence_a` и `sequence_b`.
- **then:** Проверяем, что обе последовательности соответствуют ожидаемым значениям.

6. Тестирование функции `write_output`

```
def test_write_output(self):
    # given
    length = 2
    # then
    self.assertEqual(length, 2)
```

- **given:** Задается значение `length`, равное 2.
- **then:** Проверяем, что значение равно ожидаемому — 2. Этот тест не вызывает функцию `write_output`, а просто проверяет, что значение правильно установлено.

7. Запуск тестов

```
if __name__ == '__main__':
    unittest.main()
```

- Этот блок запускает все тесты, если файл выполняется как основная программа.

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

- 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).

```
import os

def read_input(file_path):
    with open(file_path, 'r') as f:
        n = int(f.readline().strip())
        sequence_a = list(map(int, f.readline().strip().split()))
        m = int(f.readline().strip())
        sequence_b = list(map(int, f.readline().strip().split()))
        l = int(f.readline().strip())
        sequence_c = list(map(int, f.readline().strip().split()))
    return sequence_a, sequence_b, sequence_c

def write_output(file_path, length):
    with open(file_path, 'w') as f:
        f.write(f"{length}\n")

def lcs_length(sequence_a, sequence_b, sequence_c):
    n = len(sequence_a)
    m = len(sequence_b)
    l = len(sequence_c)

    dp = [[[0] * (l + 1) for _ in range(m + 1)] for __ in range(n + 1)]

    for i in range(1, n + 1):
        for j in range(1, m + 1):
            for k in range(1, l + 1):
                if sequence_a[i - 1] == sequence_b[j - 1] == sequence_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][1]

def main():
    input_path = os.path.join('.', 'txtf', 'input.txt')
    output_path = os.path.join('.', 'txtf', 'output.txt')

    sequence_a, sequence_b, sequence_c = read_input(input_path)
    length = lcs_length(sequence_a, sequence_b, sequence_c)
    write_output(output_path, length)

if __name__ == "__main__":
    main()

```

input.txt:

```

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

```

output.txt:

2

1. Импорт необходимых модулей

```
import os
```

- **os**: Модуль для работы с файловой системой, в частности, для работы с путями к файлам.

2. Функция чтения входных данных

```

def read_input(file_path):
    with open(file_path, 'r') as f:
        n = int(f.readline().strip())
        sequence_a = list(map(int,
f.readline().strip().split()))
        m = int(f.readline().strip())
        sequence_b = list(map(int,
f.readline().strip().split()))
        l = int(f.readline().strip())
        sequence_c = list(map(int,
f.readline().strip().split()))
    return sequence_a, sequence_b, sequence_c

```

- Эта функция считывает данные из файла input.txt.
- Сначала считывается длина первой последовательности nnn.
- Затем считывается сама последовательность A и преобразуется в список целых чисел.
- Аналогично считывается длина второй последовательности mmm и сама последовательность B.
- Далее считывается длина третьей последовательности l и сама последовательность C.

- Функция возвращает три последовательности: sequence_a, sequence_b и sequence_c.

3. Функция записи выходных данных

```
def write_output(file_path, length):
    with open(file_path, 'w') as f:
        f.write(f"{length}\n")
```

- Эта функция записывает длину LCS в файл output.txt.
- f"{length}\n" форматирует результат как строку.

4. Функция для вычисления длины LCS

```
def lcs_length(sequence_a, sequence_b, sequence_c):
    n = len(sequence_a)
    m = len(sequence_b)
    l = len(sequence_c)

    dp = [[[0] * (l + 1) for _ in range(m + 1)] for __ in
range(n + 1)]
```

- Определяются длины всех трех последовательностей nnn, mmm и lll.
- Создается трехмерный массив dp размером (n + 1) x (m + 1) x (l + 1), который будет использоваться для хранения промежуточных результатов. Каждый элемент dp[i][j][k] хранит длину LCS для первых i элементов последовательности A, первых j элементов последовательности B и первых k элементов последовательности C.

```
for i in range(1, n + 1):
    for j in range(1, m + 1):
        for k in range(1, l + 1):
            if sequence_a[i - 1] == sequence_b[j - 1] ==
sequence_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])
```

• Основной алгоритм:

- Вложенные циклы проходят по всем элементам трех последовательностей.
- Если текущие элементы равны (sequence_a[i - 1] == sequence_b[j - 1] == sequence_c[k - 1]), то длина LCS увеличивается на 1: dp[i][j][k] = dp[i - 1][j - 1][k - 1] + 1.
- Если элементы различаются, dp[i][j][k] принимает максимальное значение из трех возможных вариантов:
 - dp[i - 1][j][k]: длина LCS без текущего элемента из A.
 - dp[i][j - 1][k]: длина LCS без текущего элемента из B.
 - dp[i][j][k - 1]: длина LCS без текущего элемента из C.

```
return dp[n][m][l]
```

- Функция возвращает длину самой длинной общей подпоследовательности, которая хранится в dp[n][m][l].

5. Главная функция

```
def main():
    input_path = os.path.join('..', 'txtf', 'input.txt')
```

```
output_path = os.path.join('..', 'txtf', 'output.txt')
```

```
sequence_a, sequence_b, sequence_c = read_input(input_path)
length = lcs_length(sequence_a, sequence_b, sequence_c)
write_output(output_path, length)
```

- Задаются пути к входному и выходному файлам.
- Читаем данные из файла, вычисляем длину LCS и записываем результат в выходной файл.

6. Запуск программы

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

- Этот блок запускает главную функцию main, если файл выполняется как основная программа.

Unittest для задание 5:

```
import os
import unittest
import sys

sys.path.insert(0, os.path.abspath(os.path.join(os.path.dirname(__file__), '..', 'src')))
from e5 import lcs_length

class TestLCS(unittest.TestCase):

    def test_lcs_length(self):
        # given
        self.assertEqual(lcs_length([1, 2, 3], [2, 1, 3], [1, 3, 5]), 2)
        self.assertEqual(lcs_length([8, 3, 2, 1, 7], [8, 2, 1, 3, 8, 10, 7], [6, 8, 3, 1,
4, 7]), 3)
        # when
        self.assertEqual(lcs_length([1, 2, 3], [4, 5, 6], [7, 8, 9]), 0)
        self.assertEqual(lcs_length([1, 1, 1], [1, 1, 1], [1, 1, 1]), 3)
        # then
        self.assertEqual(lcs_length([], [], []), 0)

if __name__ == '__main__':
    unittest.main()
```

* Результат :

✓ Test Results 0ms		✓ Tests passed: 1 of 1 test – 0ms
✓ test 5 0ms		
✓ TestLCS 0ms		===== test session starts =====
✓ test_lcs_length 0ms		collecting ... collected 1 item
		test 5.py::TestLCS::test_lcs_length PASSED [100%]
		===== 1 passed, 1 warning in 0.01s =====
		Process finished with exit code 0

1. Импорт необходимых модулей

```
import os
import unittest
import sys
```

- **os**: Модуль для работы с файловой системой.
- **unittest**: Модуль для создания и выполнения тестов.
- **sys**: Модуль для взаимодействия с интерпретатором Python.

2. Настройка пути к модулю

```
sys.path.insert(0,
os.path.abspath(os.path.join(os.path.dirname(__file__), '..',
'src'))))
from e5 import lcs_length
```

- Добавляется путь к директории src, чтобы можно было импортировать функцию lcs_length из файла e5.py.

3. Определение класса тестов

```
class TestLCS(unittest.TestCase):
```

- Создается класс TestLCS, который наследует от unittest.TestCase. Это позволяет использовать методы для тестирования функции.

4. Тестирование функции lcs_length

```
def test_lcs_length(self):
```

- Определяется метод test_lcs_length, в котором будут проверяться различные случаи.

Тестовые случаи

```
# given
self.assertEqual(lcs_length([1, 2, 3], [2, 1, 3], [1, 3, 5]), 2)
self.assertEqual(lcs_length([8, 3, 2, 1, 7], [8, 2, 1, 3, 8, 10,
7], [6, 8, 3, 1, 4, 7]), 3)
```

- **given**: Проверяем, что длина LCS для первых двух наборов последовательностей равна ожидаемым значениям:
 - Для последовательностей [1, 2, 3], [2, 1, 3] и [1, 3, 5] длина LCS равна 2 (общая подпоследовательность: [1, 3]).
 - Для последовательностей [8, 3, 2, 1, 7], [8, 2, 1, 3, 8, 10, 7] и [6, 8, 3, 1, 4, 7] длина LCS равна 3 (общая подпоследовательность: [8, 3, 7]).

```
# when
self.assertEqual(lcs_length([1, 2, 3], [4, 5, 6], [7, 8, 9]), 0)
self.assertEqual(lcs_length([1, 1, 1], [1, 1, 1], [1, 1, 1]), 3)
```

- **when**: Проверяем другие случаи:
 - Для последовательностей [1, 2, 3], [4, 5, 6] и [7, 8, 9] ожидаемое значение — 0 (нет общих элементов).
 - Для последовательностей [1, 1, 1], [1, 1, 1] и [1, 1, 1] ожидаемое значение — 3 (все элементы совпадают).

```
# then
self.assertEqual(lcs_length([], [], []), 0)
```

- **then:** Проверяем крайний случай, когда все последовательности пустые. Ожидаемое значение — 0, так как нет элементов для сравнения.

5. Запуск тестов

```
if __name__ == '__main__':
    unittest.main()
```

- Этот блок запускает все тесты, если файл выполняется как основная программа.

Задание 6 : Наибольшая возрастающая подпоследовательность

Дана последовательность, требуется найти ее наибольшую возрастающую под- последовательность.

- **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 выведе- дите длину наибольшей возрастающей подпоследовательности, а во второй строке выведите через пробел самую наибольшую возрастающую подпоследо- вательность 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

```
import os

def read_input(file_path):
    with open(file_path, 'r') as f:
        n = int(f.readline().strip())
        sequence = list(map(int, f.readline().strip().split()))
    return n, sequence

def write_output(file_path, length, subsequence):
    with open(file_path, 'w') as f:
        f.write(f"{length}\n")
        f.write(" ".join(map(str, subsequence)) + "\n")

def longest_increasing_subsequence(sequence):
    n = len(sequence)
    if n == 0:
        return 0, []

    lis_length = [1] * n
    previous_index = [-1] * n
```



```

max_length = 0
max_index = 0

for i in range(n):
    for j in range(i):
        if sequence[i] > sequence[j] and lis_length[i] < lis_length[j] + 1:
            lis_length[i] = lis_length[j] + 1
            previous_index[i] = j

    if lis_length[i] > max_length:
        max_length = lis_length[i]
        max_index = i

lis = []
while max_index != -1:
    lis.append(sequence[max_index])
    max_index = previous_index[max_index]

lis.reverse()
return max_length, lis

def main():
    input_path = os.path.join('..', 'txtf', 'input.txt')
    output_path = os.path.join('..', 'txtf', 'output.txt')

    n, sequence = read_input(input_path)
    length, subsequence = longest_increasing_subsequence(sequence)
    write_output(output_path, length, subsequence)

if __name__ == "__main__":
    main()

```

input.txt:

```

6
3 29 5 5 28 6

```

output.txt:

```

3
3 5 28

```

1. Импорт необходимых модулей

```
import os
```

- **os:** Модуль для работы с файловой системой, позволяет управлять путями к файлам.

2. Функция чтения входных данных

```

def read_input(file_path):
    with open(file_path, 'r') as f:
        n = int(f.readline().strip())
        sequence = list(map(int, f.readline().strip().split()))
    return n, sequence

```

- Эта функция считывает данные из файла input.txt.
- Сначала считывается длина последовательности nnn.
- Затем считывается сама последовательность и преобразуется в список целых чисел.
- Функция возвращает длину последовательности и саму последовательность.

3. Функция записи выходных данных

```
def write_output(file_path, length, subsequence):
    with open(file_path, 'w') as f:
        f.write(f"{length}\n")
        f.write(" ".join(map(str, subsequence)) + "\n")
```

- Эта функция записывает длину наибольшей возрастающей подпоследовательности и саму подпоследовательность в файл output.txt.
- f"{length}\n" форматирует длину как строку.
- join(map(str, subsequence)) создает строку из элементов подпоследовательности, разделенных пробелами.

4. Функция для нахождения наибольшей возрастающей подпоследовательности

```
def longest_increasing_subsequence(sequence):
    n = len(sequence)
    if n == 0:
        return 0, []
```

- Определяется длина входной последовательности nnn.
- Если последовательность пустая, возвращается длина 0 и пустой список.

Инициализация массивов

```
lis_length = [1] * n
previous_index = [-1] * n
max_length = 0
max_index = 0
```

- lis_length: массив для хранения длины LIS, заканчивающейся на каждом элементе. Инициализируется единицами, так как каждая отдельная цифра является возрастающей подпоследовательностью.
- previous_index: массив для хранения индексов предыдущих элементов в LIS. Инициализируется -1, что означает отсутствие предыдущего элемента.
- max_length и max_index используются для отслеживания максимальной длины LIS и индекса последнего элемента этой подпоследовательности.

Основной алгоритм

```
for i in range(n):
    for j in range(i):
        if sequence[i] > sequence[j] and lis_length[i] <
lis_length[j] + 1:
            lis_length[i] = lis_length[j] + 1
            previous_index[i] = j

    if lis_length[i] > max_length:
        max_length = lis_length[i]
        max_index = i
```

- Внешний цикл проходит по каждому элементу iii последовательности.
- Внутренний цикл проходит по всем предыдущим элементам jjj.
- Проверяется, является ли текущий элемент sequence[i]sequence[i]sequence[i] больше предыдущего sequence[j]sequence[j]sequence[j] и можно ли увеличить длину возрастающей подпоследовательности:

- Если да, обновляется `lis_length[i]` и устанавливается `previous_index[i]` на `jjj`.
- Если длина текущей возрастающей подпоследовательности больше максимальной найденной, обновляются `max_length` и `max_index`.

Восстановление подпоследовательности

```
lis = []
while max_index != -1:
    lis.append(sequence[max_index])
    max_index = previous_index[max_index]

lis.reverse()
return max_length, lis
```

- Создается пустой список `lis` для хранения самой возрастающей подпоследовательности.
- Используя массив `previous_index`, восстанавливаем подпоследовательность, начиная с `max_index` и добавляя элементы в список.
- В конце список переворачивается, чтобы получить элементы в правильном порядке.
- Функция возвращает длину максимальной возрастающей подпоследовательности и саму подпоследовательность.

5. Главная функция

```
def main():
    input_path = os.path.join '..', 'txtf', 'input.txt'
    output_path = os.path.join '..', 'txtf', 'output.txt'

    n, sequence = read_input(input_path)
    length, subsequence =
longest_increasing_subsequence(sequence)
    write_output(output_path, length, subsequence)
```

- Задаются пути к входному и выходному файлам.
- Читаем данные, вычисляем длину и самую наибольшую возрастающую подпоследовательность, и записываем результат в выходной файл.

6. Запуск программы

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

- Этот блок запускает главную функцию `main`, если файл выполняется как основная программа.

Unittest для задание 6:

```
import os
import unittest
import sys

sys.path.insert(0, os.path.abspath(os.path.join(os.path.dirname(__file__), '..', 'src')))
from e6 import longest_increasing_subsequence

class TestLIS(unittest.TestCase):
```

```

def test_longest_increasing_subsequence(self):
    # given
    self.assertEqual(longest_increasing_subsequence([3, 29, 5, 5, 28, 6]), (3, [3, 5, 28]))
    self.assertEqual(longest_increasing_subsequence([10, 22, 9, 21, 20, 25]), (3, [10, 22, 25]))
    # when
    self.assertEqual(longest_increasing_subsequence([3, 2, 1]), (1, [3]))
    self.assertEqual(longest_increasing_subsequence([1, 2, 3, 4, 5]), (5, [1, 2, 3, 4, 5]))
    # then
    self.assertEqual(longest_increasing_subsequence([]), (0, []))

if __name__ == '__main__':
    unittest.main()

```

* Результат :

The screenshot shows a test runner interface. On the left, a tree view shows the test hierarchy: 'Test Results' (0 ms) expanded to show 'test 6' (0 ms), which is expanded to show 'TestLIS' (0 ms), which is expanded to show 'test_longest_increasing_subsequence' (0 ms). On the right, the test output is displayed: 'test session starts', 'collecting ... collected 1 item', 'test 6.py::TestLIS::test_longest_increasing_subsequence PASSED [100%]', '1 passed, 1 warning in 0.01s', and 'Process finished with exit code 0'.

1. Импорт необходимых модулей

```

import os
import unittest
import sys

```

- **os**: Модуль для работы с файловой системой.
- **unittest**: Модуль для создания и выполнения тестов.
- **sys**: Модуль для взаимодействия с интерпретатором Python.

2. Настройка пути к модулю

```

sys.path.insert(0,
os.path.abspath(os.path.join(os.path.dirname(__file__), '..',
'src'))))
from e6 import longest_increasing_subsequence

```

- Добавляется путь к директории src, чтобы можно было импортировать функцию longest_increasing_subsequence из файла e6.py.

3. Определение класса тестов

```

class TestLIS(unittest.TestCase):

```

- Создается класс TestLIS, который наследует от unittest.TestCase. Это позволяет использовать методы для тестирования функций.

4. Тестирование функции longest_increasing_subsequence

```

def test_longest_increasing_subsequence(self):

```

- Определяется метод test_longest_increasing_subsequence, в котором будут проверяться различные случаи.

Тестовые случаи

```
# given
self.assertEqual(longest_increasing_subsequence([3, 29, 5, 5, 28, 6]), (3, [3, 5, 28]))
self.assertEqual(longest_increasing_subsequence([10, 22, 9, 21, 20, 25]), (3, [10, 22, 25]))
```

- **given:** Проверяем, что для первых двух наборов последовательностей длина и сама наибольшая возрастающая подпоследовательность соответствуют ожидаемым значениям:
 - Для последовательности [3, 29, 5, 5, 28, 6] ожидаемая длина — 3, а сама подпоследовательность — [3, 5, 28].
 - Для последовательности [10, 22, 9, 21, 20, 25] ожидаемая длина — 3, а подпоследовательность — [10, 22, 25].

```
# when
self.assertEqual(longest_increasing_subsequence([3, 2, 1]), (1, [3]))
self.assertEqual(longest_increasing_subsequence([1, 2, 3, 4, 5]), (5, [1, 2, 3, 4, 5]))
```

- **when:** Проверяем другие случаи:
 - Для последовательности [3, 2, 1] ожидаемая длина — 1, так как только элемент 3 является возрастающей подпоследовательностью.
 - Для последовательности [1, 2, 3, 4, 5] ожидаемая длина — 5, и сама последовательность является возрастающей [1, 2, 3, 4, 5].

```
# then
self.assertEqual(longest_increasing_subsequence([]), (0, []))
```

- **then:** Проверяем крайний случай, когда последовательность пустая. Ожидаемое значение — (0, []), то есть длина 0 и пустая подпоследовательность.

5. Запуск тестов

```
if __name__ == '__main__':
    unittest.main()
```

- Этот блок запускает все тесты, если файл выполняется как основная программа.

Задание 7 : 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 точ- ки («.»). 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	

```
import os

def read_input(file_path):
    with open(file_path, 'r') as f:
        pattern = f.readline().strip()
        string = f.readline().strip()
    return pattern, string

def write_output(file_path, result):
    with open(file_path, 'w') as f:
        f.write(result + '\n')

def is_match(pattern, string):
    m, n = len(pattern), len(string)

    dp = [[False] * (n + 1) for _ in range(m + 1)]
    dp[0][0] = True

    for i in range(1, m + 1):
        if pattern[i - 1] == '*':
            dp[i][0] = dp[i - 1][0]

    for i in range(1, m + 1):
        for j in range(1, n + 1):
            if pattern[i - 1] == string[j - 1] or pattern[i - 1] == '?':
                dp[i][j] = dp[i - 1][j - 1]
            elif pattern[i - 1] == '*':
                dp[i][j] = dp[i - 1][j] or dp[i][j - 1]

    return dp[m][n]

def main():
    input_path = os.path.join '..', 'txtf', 'input.txt'
    output_path = os.path.join '..', 'txtf', 'output.txt'

    pattern, string = read_input(input_path)
    if is_match(pattern, string):
        write_output(output_path, "YES")
    else:
        write_output(output_path, "NO")
```

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

input.txt:

```
k?t*n  
kitten
```

output.txt:

```
YES
```

1. Импорт необходимых модулей

```
import os
```

- **os:** Модуль для работы с файловой системой, который позволяет управлять путями к файлам.

2. Функция чтения входных данных

```
def read_input(file_path):  
    with open(file_path, 'r') as f:  
        pattern = f.readline().strip()  
        string = f.readline().strip()  
    return pattern, string
```

- Эта функция считывает данные из файла input.txt.
- Первая строка файла содержит шаблон, а вторая — строку для проверки.
- Функция возвращает шаблон и строку.

3. Функция записи выходных данных

```
def write_output(file_path, result):  
    with open(file_path, 'w') as f:  
        f.write(result + '\n')
```

- Эта функция записывает результат (YES или NO) в файл output.txt.

4. Функция для проверки соответствия шаблону

```
def is_match(pattern, string):  
    m, n = len(pattern), len(string)  
  
    dp = [[False] * (n + 1) for _ in range(m + 1)]  
    dp[0][0] = True
```

- Определяются длины шаблона *mmm* и строки *nnn*.
- Создается двумерный массив *dp*, где *dp[i][j]* будет хранить информацию о том, соответствует ли первые *iii* символов шаблона первым *jjj* символам строки.
- Инициализация *dp[0][0] = True* означает, что пустой шаблон соответствует пустой строке.

Обработка шаблона с символами «»*

```
for i in range(1, m + 1):  
    if pattern[i - 1] == '*':  
        dp[i][0] = dp[i - 1][0]
```

- Если символ в шаблоне является звездочкой «*», то она может соответствовать пустой строке. Поэтому, если в шаблоне есть последовательные «*», то *dp[i][0]* будет принимать значение *dp[i - 1][0]*.

Основной алгоритм

```
for i in range(1, m + 1):
```

```

        for j in range(1, n + 1):
            if pattern[i - 1] == string[j - 1] or pattern[i - 1]
== '?':
                dp[i][j] = dp[i - 1][j - 1]
            elif pattern[i - 1] == '*':
                dp[i][j] = dp[i - 1][j] or dp[i][j - 1]

```

- Два вложенных цикла перебирают все символы шаблона и строки.
- Если текущий символ шаблона совпадает с текущим символом строки или является «?», то `dp[i][j]` будет равно `dp[i - 1][j - 1]` (то есть, мы переносим соответствие).
- Если текущий символ шаблона — это «*», то `dp[i][j]` будет истинным, если:
 - `dp[i - 1][j]` — соответствует, если «*» представляет один или более символов.
 - `dp[i][j - 1]` — соответствует, если «*» представляет ноль символов.

Возврат результата

```

return dp[m][n]

```

- Функция возвращает значение `dp[m][n]`, которое указывает, соответствует ли весь шаблон всей строке.

5. Главная функция

```

def main():
    input_path = os.path.join '..', 'txtf', 'input.txt'
    output_path = os.path.join '..', 'txtf', 'output.txt'

    pattern, string = read_input(input_path)
    if is_match(pattern, string):
        write_output(output_path, "YES")
    else:
        write_output(output_path, "NO")

```

- Задаются пути к входному и выходному файлам.
- Читаем данные, проверяем соответствие и записываем результат в выходной файл.

6. Запуск программы

```

if __name__ == "__main__":
    main()

```

- Этот блок запускает главную функцию `main`, если файл выполняется как основная программа.

Unittest для задание 7:

```

import os
import sys
import unittest

sys.path.insert(0, os.path.abspath(os.path.join(os.path.dirname(__file__), '..', 'src')))
from e7 import is_match, read_input, write_output

class TestPatternMatching(unittest.TestCase):

    def test_matching_cases(self):
        # given
        self.assertTrue(is_match("k?t*n", "kitten"))

```



```

        self.assertTrue(is_match("a?b", "acb"))
        # when
        self.assertTrue(is_match("k*t*n", "kitten"))
        self.assertTrue(is_match("c*", "cat"))
        # then
        self.assertTrue(is_match("*", "anything"))
        self.assertTrue(is_match("", ""))

    def test_non_matching_cases(self):
        # given
        self.assertFalse(is_match("k?t?n", "kitten"))
        self.assertFalse(is_match("a?b", "ab"))
        # when
        self.assertFalse(is_match("k*t*n", "kittening"))
        self.assertFalse(is_match("c*", "bat"))
        # then
        self.assertFalse(is_match("abc", "abd"))
        self.assertFalse(is_match("", "non-empty"))

if __name__ == '__main__':
    unittest.main()

```

* Результат:

The screenshot shows a test runner interface. On the left, a tree view shows the test hierarchy: 'Test Results' (0 ms) expanded to 'test 7' (0 ms), which contains 'TestPatternMatching' (0 ms). Under 'TestPatternMatching', two tests are listed: 'test_matching_cases' (0 ms) and 'test_non_matching_cases' (0 ms), both marked with green checkmarks. On the right, the test output is displayed. It starts with '==== test session starts =====', followed by 'collecting ... collected 2 items'. Then, it shows two test results: 'test 7.py::TestPatternMatching::test_matching_cases PASSED [50%]' and 'test 7.py::TestPatternMatching::test_non_matching_cases PASSED [100%]'. This is followed by a summary line: '==== 2 passed, 2 warnings in 0.01s =====' and ends with 'Process finished with exit code 0'.

1. Импорт необходимых модулей

```

import os
import sys
import unittest

```

- **os**: Модуль для работы с файловой системой.
- **sys**: Модуль для взаимодействия с интерпретатором Python.
- **unittest**: Модуль для создания и выполнения тестов.

2. Настройка пути к модулю

```

sys.path.insert(0,
os.path.abspath(os.path.join(os.path.dirname(__file__), '..',
'src'))))
from e7 import is_match, read_input, write_output

```

- Добавляется путь к директории src, чтобы можно было импортировать функции is_match, read_input и write_output из файла e7.py.

3. Определение класса тестов

```

class TestPatternMatching(unittest.TestCase):

```

- Создается класс TestPatternMatching, который наследует от unittest.TestCase. Это позволяет использовать методы для тестирования функций.

4. Тестирование случаев соответствия шаблону

```
def test_matching_cases(self):
```

- Определяется метод `test_matching_cases`, в котором будут проверяться случаи, когда строка соответствует шаблону.

Тестовые случаи соответствия

```
# given
self.assertTrue(is_match("k?t*n", "kitten"))
self.assertTrue(is_match("a?b", "acb"))
```

- **given:** Проверяем, что строка соответствует шаблону:
 - Для шаблона `"k?t*n"` и строки `"kitten"` ожидается, что функция вернет `True`, так как символы соответствуют.
 - Для шаблона `"a?b"` и строки `"acb"` также ожидается `True`, так как символ `?` соответствует любому одному символу.

```
# when
self.assertTrue(is_match("k*t*n", "kitten"))
self.assertTrue(is_match("c*", "cat"))
```

- **when:** Проверяем другие случаи:
 - Для шаблона `"k*t*n"` и строки `"kitten"` ожидается `True`, так как `*` может соответствовать символам между `k` и `t`.
 - Для шаблона `"c*"` и строки `"cat"` ожидается `True`, так как `*` может соответствовать символам после `c`.

```
# then
self.assertTrue(is_match("*", "anything"))
self.assertTrue(is_match("", ""))
```

- **then:** Проверяем крайние случаи:
 - Шаблон `"*"` должен соответствовать любой строке, поэтому для `"anything"` ожидается `True`.
 - Пустой шаблон должен соответствовать пустой строке, поэтому ожидается `True`.

5. Тестирование случаев несоответствия

```
def test_non_matching_cases(self):
```

- Определяется метод `test_non_matching_cases`, в котором будут проверяться случаи, когда строка не соответствует шаблону.

Тестовые случаи несоответствия

```
# given
self.assertFalse(is_match("k?t?n", "kitten"))
self.assertFalse(is_match("a?b", "ab"))
```

- **given:** Проверяем, что строка не соответствует шаблону:
 - Для шаблона `"k?t?n"` и строки `"kitten"` ожидается `False`, так как символы не соответствуют.
 - Для шаблона `"a?b"` и строки `"ab"` также ожидается `False`, так как `?` должно соответствовать одному символу.

```
# when
self.assertFalse(is_match("k*t*n", "kittening"))
self.assertFalse(is_match("c*", "bat"))
```

- **when:** Проверяем другие случаи несоответствия:
 - Для шаблона "k*t*n" и строки "kittening" ожидается False, так как длина строки не соответствует шаблону.
 - Для шаблона "с*" и строки "bat" ожидается False, так как с не соответствует первому символу b.

```
# then
self.assertFalse(is_match("abc", "abd"))
self.assertFalse(is_match("", "non-empty"))
```

- **then:** Проверяем крайние случаи несоответствия:
 - Для шаблона "abc" и строки "abd" ожидается False, так как символы не совпадают.
 - Пустой шаблон не должен соответствовать непустой строке, поэтому для "" и "non-empty" ожидается False.

6. Запуск тестов

```
if __name__ == '__main__':
    unittest.main()
```

- Этот блок запускает все тесты, если файл выполняется как основная программа.