МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №4
по дисциплине «Построение и Анализ Алгоритмов»
Тема: Редакционное расстояние

Студент гр. 3384 _____ Рудаков А.Л.

Преподаватель _____ Шевелева А.М.

Санкт-Петербург
2025

Цель работы.

Написание алгоритмов поиска редакционного расстояния и редакционного предписания, а также алгоритма поиска расстояния Ливенштейна.

Задание.

1) Над строкой ε (будем считать строкой непрерывную последовательность из латинских букв) заданы следующие операции:

replace(ε, a, b) – заменить символ a на символ b.

insert(ε, a) – вставить в строку символ a (на любую позицию).

delete(ε, b) – удалить из строки символ b.

Каждая операция может иметь некоторую цену выполнения (положительное число).

Даны две строки A и B, а также три числа, отвечающие за цену каждой операции. Определите минимальную стоимость операций, которые необходимы для превращения строки A в строку B.

Входные данные: первая строка – три числа: цена операции replace, цена операции insert, цена операции delete; вторая строка – A; третья строка – B.

Выходные данные: одно число – минимальная стоимость операций.

Sample Input:

1 1 1

entrance

reenterable

Sample Output:

5

2) Над строкой ε (будем считать строкой непрерывную последовательность из латинских букв) заданы следующие операции:

replace(ε, a, b) – заменить символ a на символ b.

insert(ε, a) – вставить в строку символ a (на любую позицию).

delete(ε, b) – удалить из строки символ b.

Каждая операция может иметь некоторую цену выполнения (положительное число).

Даны две строки A и B, а также три числа, отвечающие за цену каждой операции. Определите последовательность операций (редакционное предписание) с минимальной стоимостью, которые необходимы для превращения строки A в строку B.

Пример (все операции стоят одинаково)

M	M	M	R	I	M	R	R
C	O	N	N		E	C	T
C	O	N	E	H	E	A	D

Пример (цена замены 3, остальные операции по 1)

M	M	M	D	M	I	I	I	I	D	D
C	O	N	N	E					C	T
C	O	N		E	H	E	A	D		

Входные данные: первая строка – три числа: цена операции replace, цена операции insert, цена операции delete; вторая строка – A; третья строка – B.

Выходные данные: первая строка – последовательность операций (M – совпадение, ничего делать не надо; R – заменить символ на другой; I – вставить символ на текущую позицию; D – удалить символ из строки); вторая строка – исходная строка A; третья строка – исходная строка B.

Sample Input:

1 1 1

entrance

reenterable

Sample Output:

IMIMMIMMRRM

entrance

reenterable

3) Расстоянием Левенштейна назовём минимальное количество операций вставки одного символа, удаления одного символа и замены одного символа на другой, необходимых для превращения одной строки в другую.

Разработайте программу, осуществляющую поиск расстояния Левенштейна между двумя строками.

Пример:

Для строк pedestal и stien расстояние Левенштейна равно 7:

Сначала нужно совершить четыре операции удаления символа: pedestal -> stal.

Затем необходимо заменить два последних символа: stal -> stie.

Потом нужно добавить символ в конец строки: stie -> stien.

Параметры входных данных:

Первая строка входных данных содержит строку из строчных латинских букв. (S, $1 \leq |S| \leq 2550$).

Вторая строка входных данных содержит строку из строчных латинских букв. (T, $1 \leq |T| \leq 2550$).

Параметры выходных данных:

Одно число L, равное расстоянию Левенштейна между строками S и T.

Sample Input:

pedestal

stein

Sample Output:

7

Выполнение работы.

Для первого задания реализован алгоритм, находящийся в файле first.py.

В нем реализованы функции *insert()* и *output()* для ввода начальных значений и вывода результата соответственно.

Кроме этого реализована функция *get_min_cost(..),* которая отвечает за возвращения минимальной стоимости преобразования строки A в строку B. Внутри создается матрица значений, первая строка которой заполняется значениями цены удаления текущего символа, первый столбец заполняется значениями добавления текущего символа. После этого происходит заполнение матрицы, путем выбора минимального из значений (замены, удаления, добавления или оставить без изменений). В конце возвращается значение левой нижней ячейки матрицы, оно и есть является минимальной ценой преобразования первой строки ко второй.

Тестирование приведено в табл. Б1.

Для второго задания реализован алгоритм, находящийся в файле second.py.

В нем реализованы функции *insert()* и *output()* для ввода начальных значений и вывода результата соответственно.

Также там реализована функция *get_cost_matrix(..),* которая вычисляет ту же матрицу, что и функция *get_min_cost(..)* из задания 1, но в каждой ячейке кроме самого значения записано то, какая ячейка была выбрана перед ней и какое действие было сделано для того, чтобы получить ее из предыдущей. Если текущие символы равны, записывается ‘M’, если выбрана замена (из левой верхней), записывается ‘R’, если удаление (из левой), записывается ‘D’, если вставка (из верхней), записывается ‘I’.

Функция *get_min_way(cost_matrix)* по полученной на предыдущем шаге матрице восстанавливает действия, выполняемые над строкой и возвращает результат.

Тестирование приведено в табл. Б2.

Для третьего задания реализован алгоритм, находящийся в файле third.py.

В нем реализованы функции *insert()* и *output()* для ввода начальных значений и вывода результата соответственно.

Функция *get_end_row(..)* возвращает последнюю строку матрицы, которая ищется в функции *get_min_cost(..)* из задания 1, но при этом тратит меньше памяти в силу того, что хранятся только 2 последние строки.

Функция *get_min_levinstein(..)* рекурсивно разбивает строки на более маленькие, вызывая для них *get_end_row(..)*, после чего находя лучший индекс для разбиения второй строки. Возвращает сумму рекурсивных вызовов самой себя для разделенных строк, тем самым возвращая суммарное значение расстояния Левенштейна.

Тестирование приведено в табл. Б3.

Разработанный программный код см. в приложении А.

Результаты тестирования см. в приложении Б.

Выводы.

В ходе работы были реализованы алгоритмы поиска редакционного расстояния от первой строки ко второй, редакционного предписания от первой строки ко второй и расстояния Левенштейна между двумя строками.

Алгоритм поиска редакционного расстояния от первой строки ко второй для поиска минимальной цены преобразования первой строки во вторую генерирует матрицу цен (расстояний) между символами строк, при вычислении текущей ячейки используется минимальная из верхней левой ячейки + цена замены, левой ячейки + цена удаления, правой ячейки + цена вставки. При этом если символы равны, то в поиске минимума участвует не левая верхняя ячейка + цена замены, а просто левая верхняя ячейка. В конце возвращается значения правой нижней ячейки, которое как раз таки хранит в себе значение минимальной стоимости преобразования первой строки ко второй.

Алгоритм поиска редакционного предписания для нахождения действий для минимального преобразования первой строки ко второй вычисляет ту же

матриц, что и первый алгоритм, но добавляет в нее информацию о том, из какой ячейки был получен текущий результат и какое действие для этого было совершено. Из действий возможны M – оставить символ, R – заменить символ, I – вставить символ и D – удалить символ. После вычисления данной матрицы происходит поиск пути, начиная с последней ячейки, после чего путь переворачивается и возвращается.

Алгоритм поиска расстояния Левенштейна между двумя строками представляет из себя улучшенную версию первого алгоритма, позволяя считать не одну большую таблицу, а несколько таблиц меньшего размера, в следствие чего на больших значениях он выполняется быстрее.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: first.py

```
    def insert():
        replace_cost, insert_cost, delete_cost = map(int, input().split())
        first_string = input()
        second_string = input()
        return replace_cost, insert_cost, delete_cost, first_string, second_string

    def get_min_cost(replace_cost, insert_cost, delete_cost, first_string, second_string):
        cost = [[0]]
        for i in range(len(first_string)):
            cost[0].append(cost[0][i] + delete_cost)
        for i in range(len(second_string)):
            cost.append([cost[i][0] + insert_cost])

        for i in range(1, len(second_string) + 1):
            for j in range(1, len(first_string) + 1):
                add_value = replace_cost
                if first_string[j - 1] == second_string[i - 1]:
                    add_value = 0
                cost[i].append(min(cost[i - 1][j - 1] + add_value, cost[i][j - 1] + delete_cost, cost[i - 1][j] + insert_cost))
        return cost[-1][-1]

    def output(cost_value):
        print(cost_value)

    def __main__():
        replace_cost, insert_cost, delete_cost, first_string, second_string = insert()
        cost_value = get_min_cost(replace_cost, insert_cost, delete_cost, first_string, second_string)
```

```
    output(cost_value)

if __name__ == '__main__':
    __main__()
```

Название файла: second.py

```
def insert():
    replace_cost, insert_cost, delete_cost = map(int, input().split())
    first_string = input()
    second_string = input()
    return replace_cost, insert_cost, delete_cost, first_string, second_string

def get_cost_matrix(replace_cost, insert_cost, delete_cost, first_string, second_string):
    cost = [[(0, (-1, -1), 'M')]]
    for i in range(len(first_string)):
        cost[0].append((cost[0][i][0] + delete_cost, (0, i), 'D'))
    for i in range(len(second_string)):
        cost.append([(cost[i][0][0] + insert_cost, (i, 0), 'I')])

    for i in range(1, len(second_string) + 1):
        for j in range(1, len(first_string) + 1):
            if first_string[j - 1] == second_string[i - 1]:
                cost[i].append((cost[i - 1][j - 1][0], (i - 1, j - 1), 'M'))
                continue
            current_value = min(cost[i - 1][j - 1][0] + replace_cost, cost[i][j - 1][0] + delete_cost, cost[i - 1][j][0] + insert_cost)

            if current_value == cost[i - 1][j - 1][0] + replace_cost:
                cost[i].append((current_value, (i - 1, j - 1), 'R'))
            elif current_value == cost[i][j - 1][0] + delete_cost:
                cost[i].append((current_value, (i, j - 1), 'D'))
            elif current_value == cost[i - 1][j][0] + insert_cost:
```

```
                    cost[i].append((current_value, (i - 1, j), 'I'))
        return cost

    def get_min_cost_way(cost_matrix):
        first_idx = len(cost_matrix) - 1
        second_idx = len(cost_matrix[0]) - 1
        answer_way = ""
        while first_idx != 0 or second_idx != 0:
            answer_way += cost_matrix[first_idx][second_idx][2]
            new_first_idx = cost_matrix[first_idx][second_idx][1][0]
            new_second_idx = cost_matrix[first_idx][second_idx][1][1]
            first_idx = new_first_idx
            second_idx = new_second_idx
        return answer_way[::-1]


    def output(cost_value, first_string, second_string):
        print(cost_value)
        print(first_string)
        print(second_string)

    def __main__():
        replace_cost,    insert_cost,    delete_cost,    first_string,
second_string = insert()
        cost_matrix   =   get_cost_matrix(replace_cost,   insert_cost,
delete_cost, first_string, second_string)
        cost_way = get_min_cost_way(cost_matrix)
        output(cost_way, first_string, second_string)

    if __name__ == '__main__':
        __main__()
```

Название файла: third.py

```
    def insert():
        first_string = input()
        second_string = input()
        return first_string, second_string
```

```
    def get_end_row(first_string, second_string):
        cost_first = [0]
        for i in range(len(first_string)):
            cost_first.append(cost_first[i] + 1)
        cost_second = [1]

        for i in range(1, len(second_string) + 1):
            for j in range(1, len(first_string) + 1):
                add_value = 1
                if first_string[j - 1] == second_string[i - 1]:
                    add_value = 0
                cost_second.append(min(cost_first[j - 1] + add_value,
cost_second[j - 1] + 1, cost_first[j] + 1))
            cost_first = cost_second
            cost_second = [cost_first[0] + 1]
        return cost_first


    def get_min_levinstein(first_string, second_string):
        if len(first_string) <= 1 or len(second_string) <= 1:
            return get_end_row(first_string, second_string)[-1]

        current_idx = 0
        if len(first_string) < len(second_string):
            first_left = first_string[:len(first_string) // 2]
            first_right = first_string[len(first_string) // 2 :]

            left_row = get_end_row(second_string, first_left)
            right_row = get_end_row(second_string[::-1],
first_right[::-1])

            for i in range(len(second_string) + 1):
                if left_row[i] + right_row[-i - 1] <
left_row[current_idx] + right_row[-current_idx - 1]:
                    current_idx = i

            second_left = second_string[:current_idx]
            second_right = second_string[current_idx:]
```

```
        else:
            second_left = second_string[:len(second_string) // 2]
            second_right = second_string[len(second_string) // 2:]

            left_row = get_end_row(first_string, second_left)
            right_row = get_end_row(first_string[::-1],
second_right[::-1])

            for i in range(len(first_string) + 1):
                if left_row[i] + right_row[-i - 1] <
left_row[current_idx] + right_row[-current_idx - 1]:
                    current_idx = i

            first_left = first_string[:current_idx]
            first_right = first_string[current_idx:]

        return get_min_levinstein(first_left, second_left) +
get_min_levinstein(first_right, second_right)


    def output(cost_value):
        print(cost_value)


    def __main__():
        first_string, second_string = insert()
        cost_value = get_min_levinstein(first_string, second_string)
        output(cost_value)


    if __name__ == '__main__':
        __main__()
```

ПРИЛОЖЕНИЕ Б
ТЕСТИРОВАНИЕ

Тесты для первого задания представлены в табл. Б1.

Таблица Б.1 - Примеры тестовых случаев задания 1.

№ п/п	Входные данные	Выходные данные	Комментарии
1.	1 1 1 entrance reenterable	5	Ok
2.	1 100 100 abcd abc	100	Ok
3.	100 100 1 abcd abc	1	Ok
4.	100 1 100 abc abcd	1	Ok
5.	100 100 100 abc abc	0	Ok
6.	100 1 1 abc asc	2	Ok

Тесты для второго задания представлены в табл. Б2.

Таблица Б.2 - Примеры тестовых случаев задания 2.

№ п/п	Входные данные	Выходные данные	Комментарии
1.	1 1 1 entrance reenterable	IIMMMIMMRRM entrance reenterable	Ok
2.	1 1 1 abc abc	MMM abc abc	Ok
3.	100 1 1 abc asc	MIDM abc asc	Ok
4.	1 1 1 a hfdsa	IIIIM a hfdsa	Ok
5.	1 1 1 dkjskjs d	MDDDDDD dkjskjs d	Ok
6.	1 1 1 abcds iolkj	RRRRR abcds iolkj	Ok
7.	1 1 1 abc asc	MRM abc asc	Ok

Тесты для третьего задания представлены в табл. Б3.

Таблица Б.3 - Примеры тестовых случаев задания 3.

№ п/п	Входные данные	Выходные данные	Комментарии
1.	pedestal stien	7	Ok
2.	abc abc	0	Ok
3.	chirik kosar	6	Ok
4.	abc adc	1	Ok
5.	abc ac	1	Ok
6.	abc uio	3	Ok
7.	ac abc	1	Ok