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: Динамическое программирование

Студент гр. 3384 _____ Козьмин Н.В.

Преподаватель _____ Шевелева А.М.

Санкт-Петербург

2025

Цель работы.

Разработайте программу, которая решает задачи с редакционным расстоянием с помощью динамического программирования.

Задание.

№4.1. Над строкой ε (будем считать строкой непрерывную последовательность из латинских букв) заданы следующие операции:

1. *replace(ε, a, b)* – заменить символ *a* на символ *b*.
2. *insert(ε, a)* – вставить в строку символ *a* (на любую позицию).
3. *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.

№4.1.1. 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.

№4.1.2. 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.

№4.2. 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.

Описание работы.

Был выбран Python, так как не требуется высокая скорость выполнения и писать на этом ЯП быстрее и проще.

Сначала была создана генерация таблицы с весами минимальных изменений от каждой подстроки одного слова до каждой подстроки другого. Такой подход позволяет, хранить все нужные значения и оптимально их использовать для каждой новой ячейки, чтобы в итоге получить редакционное расстояния для двух целых слов в правом нижнем углу, тем самым решая задачу с помощью динамического программирования. Делалось это следующим образом: представляем первую исходную строку в качестве строк таблицы, другую – столбцов, добавляя пустой символ для строк и столбцов в начало, означающий изменения от или до пустого подслова. Для ускорения предварительно заполняем результат (таблицу) нулевыми элементами, чтобы задать размер. Затем сначала задаем значения по нулевым индексам, используя индексы другой оси, умножая их на соответствующие коэффициенты вставки (по горизонтали) или удаления (по вертикали). Далее задаем каждую другую ячейку, беря минимальное значения из {значения слева + коэффициент, значение сверху + коэффициент, значение слева сверху (по диагонали) + коэффициент замены (или без коэффициента, если менять символ не надо, то есть если они одинаковые в таблице)}. Проверить соответствие коэффициентов можно, используя пример с одной пустой исходной строкой и одной не пустой. Это описания дает понять, что расстояние Левенштейна является частным случаем расстояния Вагнера-Фишера, где коэффициенты равны единицам, что используется в решении.

Для редакционного предписания или, если проще, для инструкции минимального изменения также используется таблица. По уже составленной таблице нужно идти с правого нижнего угла, выбирая минимальные значения, воспроизводя в обратном порядке вставку, удаление, замену или игнорирование символов. Делается это до тех пор, пока не пройден путь до левого верхнего угла.

Описание функций.

min_diff_table - таблица с весами минимальных изменений от каждой подстроки одного слова до каждой подстроки другого, используя коэффициенты замены, вставки и удаления.

min_diff - редакционное расстояния для двух целых слов, используя коэффициенты замены, вставки и удаления для передачи их в min_diff_table (результат получается из правого нижнего угла таблицы)

min_diff_instruction - редакционное предписание из передачи аргументов в min_diff_table и обратном проходе по таблице

В main`е вызываются созданные функции в зависимости от задания.

Выводы.

В ходе выполнения работы основной целью было разработать программу, которая решает задачи с редакционным расстоянием и предписанием с помощью динамического программирования. Благодаря этому расстоянию можно не только определить операции для изменения одной строки в другую, но и сравнить их похожесть. В процессе было разобрано и аргументировано применение динамического подхода. Он позволяет решать задачу оптимальным способом, так как решение ставится таким образом, что не приходиться проходить лишние разы по возможным вариантам. За счёт этого в работе для сравнения двух строк размерами n и m используются затраты производительности O(n*m) (из-за составляемой таблицы). Также было объяснено, почему расстояние Левенштейна является частным случаем расстояния Вагнера-Фишера. За счёт этого работа была сделана без дублирования путём разбиения кода на функции, где составление таблицы используется в разных задачах. Реализованная программа успешно выполняет поставленные задачи. Разработанный код см. в приложении А. Он получился лаконичным и работающим эффективно в естественных ограничениях языка. Решение поставленных задач также было протестировано. Результаты тестирования см. в приложении Б. Точность работы ограничивается весом замены. Она должна быть меньше, чем значения вставки и удаления в сумме, иначе постановка задачи теряет логику и смысл. Так, например, если надо заменить все слово, то его проще удалить и вставить новое.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.py

```
    def min_diff_table(str_one, str_two, replace_val, insert_val,
delete_val):
        result = [[0 for _ in range(len(str_two)+1)] for _ in
range(len(str_one)+1)]
        for i in range(len(str_one)+1):
            for j in range(len(str_two)+1):
                if i == 0:
                    result[i][j] = j*insert_val
                elif j == 0:
                    result[i][j] = i*delete_val
                else:
                    result[i][j] = min(
                        result[i-1][j-1]+(replace_val if str_one[i-
1]!=str_two[j-1] else 0),
                        result[i-1][j]+delete_val,
                        result[i][j-1]+insert_val,
                    )
        return result


    def min_diff(str_one, str_two, replace_val, insert_val, delete_val):
        table = min_diff_table(str_one, str_two, replace_val, insert_val,
delete_val)
        return table[len(str_one)][len(str_two)]


    def min_diff_instruction(str_one, str_two, replace_val, insert_val,
delete_val):
        table = min_diff_table(str_one, str_two, replace_val, insert_val,
delete_val)
        cur_pos_i, cur_pos_j = len(str_one), len(str_two)
        result = ""
        while cur_pos_i != 0 or cur_pos_j != 0:
            _, add_i, add_j = min(
                (table[cur_pos_i-1][cur_pos_j], -1, 0) if cur_pos_i != 0
else (float("inf"), 0, 0),
                (table[cur_pos_i-1][cur_pos_j-1], -1, -1) if
cur_pos_i != 0 and cur_pos_j != 0 else (float("inf"), 0, 0),
                (table[cur_pos_i][cur_pos_j-1], 0, -1) if cur_pos_j != 0
else (float("inf"), 0, 0)
            )
            if (add_i, add_j) == (-1, -1):
                if table[cur_pos_i][cur_pos_j] == table[cur_pos_i-
1][cur_pos_j-1]:
                    result = "M" + result
                else:
                    result = "R" + result
            elif add_i == -1:
                result = "D" + result
            else:
                result = "I" + result
            cur_pos_i += add_i
```

```
            cur_pos_j += add_j
        return result


    def main():
        replace_val, insert_val, delete_val = map(int, input().split())
        # replace_val, insert_val, delete_val = 1, 1, 1
        str_one = input()
        str_two = input()
        print(min_diff(str_one,   str_two,   replace_val,   insert_val,
delete_val))
        print(min_diff_instruction(str_one,    str_two,    replace_val,
insert_val, delete_val))
        # print(str_one)
        # print(str_two)


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

ПРИЛОЖЕНИЕ Б

ТЕСТИРОВАНИЕ

В таблице Б.1 указаны результаты тестирования решения задачи №4.1.1.
В таблице Б.2 указаны результаты тестирования решения задачи №4.1.2.
В таблице Б.3 указаны результаты тестирования решения задачи №4.2.

Таблица Б.1 - Тестовые случаи задачи №4.1.1.

№ п/п	Выходные и входные данные	Комментарии
1.	1 1 2 a 2	
2.	2 1 2 a 1	
3.	1 1 1 аргумент рудимент 4	
4.	2 1 1 аргумент рудимент 4	
5.	1 1 1 entrance reenterable 5	

Таблица Б.2 - Тестовые случаи задачи №4.1.2.

№ п/п	Выходные и входные данные	Комментарии
1.	1 1 2 a D a	

Продолжение таблицы Б.2

№ п/п	Выходные и входные данные	Комментарии
2.	2 1 2 a I a	
3.	1 1 1 аргумент рудимент RRRRMMMM аргумент рудимент	
4.	2 1 1 аргумент рудимент DMDMIIMMMM аргумент рудимент	
5.	1 1 1 entrance reenterable IMIMMIMMRRM entrance reenterable	

Таблица Б.3 - Тестовые случаи задачи №4.2.

№ п/п	Выходные и входные данные	Комментарии
1.	 a 1	
2.	aba aba 0	
3.	pedestal stien 7	