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

ОТЧЕТ

по лабораторной работе №3

по дисциплине «Построение и анализ алгоритмов»

Тема: Редакционное расстояние

Студент гр. 3343 _____ Никишин С.А.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург

2025

Цель работы.

Изучить и реализовать алгоритм Вагнера-Фишера. Вычислить с его помощью редакционное расстояние и предписание.

Задание Общая часть.

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. Определите последовательность операций (редакционное предписание) с минимальной стоимостью, 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.

Задание 4.1.2.

Выходные данные: одно число – минимальная стоимость операций.

Sample Input:

```
1 1 1
entrance
reenterable
```

Sample Output:

```
5
```

Задание 4.1.3.

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**		

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
```

Описание алгоритма.

Алгоритм Вагнера-Фишера (также известный как алгоритм Вагнера-Фишера-Левенштейна) — это алгоритм для вычисления редакционного расстояния двух строк. Алгоритм позволяет построить предписание — последовательность операций, которая приводит к такому преобразованию.

Редакционное расстояние — это минимальное количество операций вставки, удаления и замены символов, необходимых для преобразования одной строки в другую.

Основные шаги алгоритма вычисления редакционного расстояния:

1. Создаётся матрицу размером (m+1) на (n+1)

2. Заполняем первые столбец и строку по формулам:

D[i][0] = i * deleteCost

D[0][j] = j * insertCost

3. Заполняем остальную матрицу:

3.1 Если символы строк равны, стоимость не увеличивается

3.2 Иначе выбираем минимальную из операций:

D[i-1][j] + deleteCost - удаление

D[i][j-1] + insertCost - вставка

D[i - 1][j - 1] + replaceCost - замена

4. Последняя ячейка матрицы будет содержать редакционное расстояние.

Основные шаги алгоритма вычисления редакционного предписания:

1. Строим матрицу (как при вычисление редакционного расстояния).

2. Начинаем с конца матрицы.

3. Проверяем случаи и сохраняем тип операции в последовательность:

3.1 Если символы равны:

Ничего не меняется, переходим к (i-1, j-1).

3.2 Иначе выбираем минимальную операцию:

3.2.1 Удаление, переходим (i-1, j).

3.2.2 Вставляем, переходим (i, j-1).

3.2.3 Изменяем, переход (i-1, j-1).

4. В конце переворачиваем последовательность и получаем редакционное предписание.

Алгоритмы реализующие функции имеет сложность по времени и памяти:

Функция	Вычисляет	Сложность по времени	Сложность по памяти
wagnerFischerDistance	Редакционное расстояние	O(m * n)	O(m * n)
wagnerFisherEditorialPrescription	Редакционное предписание	O(m * n)	O(m * n)

,где m – длина образца, n – длина текста

Исходные коды обоих программ указаны в приложении (приложение 1 и приложение 2 соответственно)

Описание функций и структур данных.

Структуры данных:

- *TrieNode* - класс узла бора, хранящий идентификатор, флаги терминальности, индексы шаблонов, дочерние узлы и ссылки

- *patterns* - список шаблонов для поиска в тексте (тип list[str])

- *results* - список найденных вхождений шаблонов с позициями и длинами (тип list[tuple])

Функции:

def __init__(self, patterns: list) - конструктор класса AhoCorasicAlgorithm, инициализирующий бор и строящий автомат Ахо-Корасика для заданных шаблонов

Параметры:

patterns - список шаблонов для поиска (тип list[str])

Сложность по времени: O(L), где L - суммарная длина всех шаблонов

def __add(self, pattern: str, index: int) - приватный метод добавления одного шаблона в бор

Параметры: pattern - шаблон для добавления (тип str), index - индекс шаблона (тип int)

Сложность по времени: O(m), где m - длина шаблона

def __makeLinks(self) - метод построения суффиксных и конечных ссылок в автомате

Сложность по времени: O(L), где L - суммарная длина всех шаблонов

def search(self, text) - метод поиска всех вхождений шаблонов в тексте с использованием автомата Ахо-Корасика

Параметры: text - текст для поиска (тип str)

Возвращает: отсортированный список найденных вхождений (тип list[tuple])

Сложность по времени: O(n + z), где n - длина текста, z - количество найденных вхождений

def getNodeCount(self) - метод возвращающий количество узлов в автомате

Возвращает: число узлов (тип int)

Сложность по времени: O(1)

def getMaxArcs(self) - метод возвращающий максимальное количество дуг из одной вершины бора

Возвращает: максимальное число исходящих дуг (тип int)

Сложность по времени: O(1)

def removeFoundPatterns(text, results, patterns) - функция вырезания найденных образцов из строки поиска

Параметры: text - исходный текст (тип str), results - найденные вхождения (тип list), patterns - шаблоны (тип list)

Возвращает: остаток строки после удаления шаблонов (тип str)

Сложность по времени: O(n + k), где n - длина текста, k - количество найденных вхождений

Функция	Сложность по времени	Сложность по памяти
wagnerFischerDistance	O(m * n)	O(m * n)
wagnerFisherEditorialPrescription	O(m * n)	O(m * n)
printMatrix	O(m * n)	O(1)

Тестирование.

Тестирование программ представлено на таблицах:

Таблица 1. Тестирование вычисления редакционного предписания

№ теста	Ввод	Вывод
1	1 1 1 entrance reenterable	5
2	2 2 2 cat car	2
3	1 1 1 abc cba	2
4	10 10 10 test test	0
5	5 5 5 abba	20
6	10 2 3 entrance reenterable	16

Таблица 2. Тестирование вычисления редакционного предписания

№ теста	Ввод	Вывод
1	1 1 1 entrance reenterable	IMIMMIMMRRM entrance reenterable
2	2 2 2 cat car	MMR cat car

3	1 1 1 abc cba	RMR abc cba
4	10 10 10 test test	MMMM test test
5	5 5 5 abba	DDDD abba
6	10 2 3 entrance reenterable	IMIMMIMMIIDDM entrance reenterable

Выводы.

Был реализован алгоритм Вагнера-Фишера. Реализованный алгоритм был использован для вычисления минимального редакционного расстояния и редакционного предписания. Полученные при тестировании ответы подтвердили корректность работы алгоритма.

Приложение

Приложение 1

```
import sys

def wagnerFischerDistance(string1: str, string2: str, replaceCost: int, insertCost: int,
deleteCost: int, doubleDeleteCost: int) -> int:
    """Ищет минимальное редакционное расстояние с учетом правила треугольника и операции
удаления двух символов"""
    n, m = len(string1), len(string2)

    # Проверяем правило треугольника для весов операций
    if replaceCost > deleteCost + insertCost:
        replaceCost = deleteCost + insertCost

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

    # Инициализация первой строки и столбца
    for i in range(n + 1):
        dp[i][0] = i * deleteCost
    for j in range(m + 1):
        dp[0][j] = j * insertCost

    # Заполнение матрицы
    for i in range(1, n + 1):
        for j in range(1, m + 1):
            operations = []

            # Удаление одного символа из string1
            operations.append(dp[i-1][j] + deleteCost)

            # Вставка одного символа в string1
            operations.append(dp[i][j-1] + insertCost)

            # Замена или совпадение символов
            if string1[i-1] == string2[j-1]:
                operations.append(dp[i-1][j-1])  # Совпадение
            else:
                operations.append(dp[i-1][j-1] + replaceCost)  # Замена

            # Операция удаления двух последовательных символов (4-я операция)
            if i >= 2 and string1[i-1] == string1[i-2]:
                operations.append(dp[i-2][j] + doubleDeleteCost)

            dp[i][j] = min(operations)

    return dp[n][m]

def wagnerFisherEditorialPrescription(string1: str, string2: str, replaceCost: int,
insertCost: int, deleteCost: int, doubleDeleteCost: int) -> str:
    """Определяет редакционное предписание с учетом операции удаления двух символов"""
    n, m = len(string1), len(string2)

    if replaceCost > deleteCost + insertCost:
        replaceCost = deleteCost + insertCost

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

    # Инициализация
    for i in range(n + 1):
```

```
        dp[i][0] = i * deleteCost
        prescript[i][0] = "D" * i

    for j in range(m + 1):
        dp[0][j] = j * insertCost
        prescript[0][j] = "I" * j

    # Заполнение матриц
    for i in range(1, n + 1):
        for j in range(1, m + 1):
            min_cost = float('inf')
            best_prescript = ""

            # Удаление одного символа
            cost_del = dp[i-1][j] + deleteCost
            if cost_del < min_cost:
                min_cost = cost_del
                best_prescript = prescript[i-1][j] + "D"

            # Вставка одного символа
            cost_ins = dp[i][j-1] + insertCost
            if cost_ins < min_cost:
                min_cost = cost_ins
                best_prescript = prescript[i][j-1] + "I"

            # Замена или совпадение
            if string1[i-1] == string2[j-1]:
                cost_rep = dp[i-1][j-1]
                if cost_rep < min_cost:
                    min_cost = cost_rep
                    best_prescript = prescript[i-1][j-1] + "M"
            else:
                cost_rep = dp[i-1][j-1] + replaceCost
                if cost_rep < min_cost:
                    min_cost = cost_rep
                    best_prescript = prescript[i-1][j-1] + "R"

            # Удаление двух символов (4-я операция со своей стоимостью)
            if i >= 2 and string1[i-1] == string1[i-2]:
                cost_dd = dp[i-2][j] + doubleDeleteCost
                if cost_dd < min_cost:
                    min_cost = cost_dd
                    best_prescript = prescript[i-2][j] + "DD"

            dp[i][j] = min_cost
            prescript[i][j] = best_prescript

    return prescript[n][m]

def printMatrix(string1: str, string2: str, replaceCost: int, insertCost: int, deleteCost:
int, doubleDeleteCost: int):
    """Выводит матрицу редакционных расстояний"""
    n, m = len(string1), len(string2)

    if replaceCost > deleteCost + insertCost:
        replaceCost = deleteCost + insertCost

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

    # Инициализация
    for i in range(n + 1):
        dp[i][0] = i * deleteCost
    for j in range(m + 1):
```

```
        DP[0][J] = J * INSERTCOST

    # ЗАПОЛНЕНИЕ МАТРИЦЫ
    FOR I IN RANGE(1, N + 1):
        FOR J IN RANGE(1, M + 1):
            OPERATIONS = []
            OPERATIONS.APPEND(DP[I-1][J] + DELETECOST)
            OPERATIONS.APPEND(DP[I][J-1] + INSERTCOST)

            IF STRING1[I-1] == STRING2[J-1]:
                OPERATIONS.APPEND(DP[I-1][J-1])
            ELSE:
                OPERATIONS.APPEND(DP[I-1][J-1] + REPLACECOST)

            IF I >= 2 AND STRING1[I-1] == STRING1[I-2]:
                OPERATIONS.APPEND(DP[I-2][J] + DOUBLEDELETECOST)

            DP[I][J] = MIN(OPERATIONS)

    # ВЫВОД МАТРИЦЫ
    PRINT("\NМАТРИЦА РЕДАКЦИОННЫХ РАССТОЯНИЙ:")
    PRINT("       ", END="")
    FOR J IN RANGE(M + 1):
        IF J == 0:
            PRINT("E    ", END="")
        ELSE:
            PRINT(F"{STRING2[J-1]:<4} ", END="")
    PRINT()

    FOR I IN RANGE(N + 1):
        IF I == 0:
            PRINT("E  ", END="")
        ELSE:
            PRINT(F"{STRING1[I-1]}  ", END="")

        FOR J IN RANGE(M + 1):
            PRINT(F"{DP[I][J]:<4} ", END="")
        PRINT()

DEF MAIN():
    TRY:
        # ЧТЕНИЕ ВХОДНЫХ ДАННЫХ (4 ЧИСЛА!)
        PRINT("ВВЕДИТЕ СТОИМОСТИ ОПЕРАЦИЙ (ЗАМЕНА ВСТАВКА УДАЛЕНИЕ ДВОЙНОЕ_УДАЛЕНИЕ):")
        COSTS = LIST(MAP(INT, INPUT().SPLIT()))
        IF LEN(COSTS) != 4:
            PRINT("ОШИБКА: НУЖНО ВВЕСТИ ЧЕТЫРЕ ЧИСЛА (СТОИМОСТИ ОПЕРАЦИЙ)")
            RETURN

        REPLACECOST, INSERTCOST, DELETECOST, DOUBLEDELETECOST = COSTS
        PRINT("ВВЕДИТЕ ПЕРВУЮ СТРОКУ:")
        STRING1 = INPUT().STRIP()
        PRINT("ВВЕДИТЕ ВТОРУЮ СТРОКУ:")
        STRING2 = INPUT().STRIP()

        # ВЫЧИСЛЕНИЕ РАССТОЯНИЯ
        DISTANCE = WAGNERFISCHERDISTANCE(STRING1, STRING2, REPLACECOST, INSERTCOST,
DELETECOST, DOUBLEDELETECOST)
        PRINT(F"\NМИНИМАЛЬНОЕ РЕДАКЦИОННОЕ РАССТОЯНИЕ: {DISTANCE}")

        # ПОЛУЧЕНИЕ РЕДАКЦИОННОГО ПРЕДПИСАНИЯ
        PRESCRIPTION = WAGNERFISHEREDITORIALPRESCRIPTION(STRING1, STRING2, REPLACECOST,
INSERTCOST, DELETECOST, DOUBLEDELETECOST)
        PRINT(F"РЕДАКЦИОННОЕ ПРЕДПИСАНИЕ: {PRESCRIPTION}")
```

```
        # Вывод матрицы
        printMatrix(string1, string2, replaceCost, insertCost, deleteCost, doubleDeleteCost)

        # Дополнительная информация
        print(f"\nДополнительная информация:")
        print(f"Длина первой строки: {len(string1)}")
        print(f"Длина второй строки: {len(string2)}")
        print(f"Стоимости операций: замена={replaceCost}, вставка={insertCost},
удаление={deleteCost}, двойное_удаление={doubleDeleteCost}")

    except ValueError as e:
        print(f"Ошибка ввода: {e}")
    except Exception as e:
        print(f"Произошла ошибка: {e}")

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

ПРИЛОЖЕНИЕ 2:

```
import sys

def wagnerFischerDistance(string1: str, string2: str, replaceCost: int, insertCost: int, deleteCost: int, doubleDeleteCost: int) -> int:
    """Ищет минимальное редакционное расстояние с учетом правила треугольника и операции удаления двух символов"""
    n, m = len(string1), len(string2)

    # Проверяем правило треугольника для весов операций
    if replaceCost > deleteCost + insertCost:
        replaceCost = deleteCost + insertCost

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

    # Инициализация первой строки и столбца
    for i in range(n + 1):
        dp[i][0] = i * deleteCost
    for j in range(m + 1):
        dp[0][j] = j * insertCost

    # Заполнение матрицы
    for i in range(1, n + 1):
        for j in range(1, m + 1):
            operations = []

            # Удаление одного символа из string1
            operations.append(dp[i-1][j] + deleteCost)

            # Вставка одного символа в string1
            operations.append(dp[i][j-1] + insertCost)

            # Замена или совпадение символов
            if string1[i-1] == string2[j-1]:
                operations.append(dp[i-1][j-1])  # Совпадение
            else:
                operations.append(dp[i-1][j-1] + replaceCost)  # Замена
```

```
            # Операция удаления двух последовательных символов (4-я операция)
            if i >= 2 and string1[i-1] == string1[i-2]:
                operations.append(dp[i-2][j] + doubleDeleteCost)

            dp[i][j] = min(operations)

    return dp[n][m]

def printMatrix(matrix: list):
    """Выводит на печать матрицу"""
    for line in matrix:
        print(line)
    print()

def main():
    # Ввод стоимостей операций (4 числа)
    replaceCost, insertCost, deleteCost, doubleDeleteCost = map(int, input().split())
    # Ввод строк
    string1 = input()
    string2 = input()

    with open("output.txt", "w", encoding="utf-8") as file:
        sys.stdout = file
        # Вычисление результата
        result = wagnerFischerDistance(string1, string2, replaceCost, insertCost, deleteCost,
doubleDeleteCost)
        sys.stdout = sys.__stdout__            # Возвращаем вывод в консоль

    print(result)   # Вывод результата

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