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
Вариант 14б

Студент гр. 3388 _____ Потоцкий С.С.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург
2025

Цель работы

Изучить алгоритмы Вагнера-Фишера и Левенштейна. Реализовать алгоритм с дополнительной модификацией.

Задания.

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 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
```

Задание №2.

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.

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
```

Задание №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
stien
```

Sample Output:

```
7
```

Индивидуальный вариант

14б. Найти длину наибольшей общей подстроки двух строк, вывести и саму эту подстроку.

Описание алгоритма

Первое задание

Общее описание

Данный код реализует алгоритм динамического программирования для вычисления минимального редакционного расстояния между двумя строками A и B. Алгоритм находит минимальную стоимость преобразования одной строки в другую с использованием трех базовых операций редактирования (вставка, удаление, замена).

Входные данные

Стоимости операций: replaceCost, insertCost, deleteCost

Две строки: A (исходная) и B (целевая)

Операции редактирования:

Замена символа - изменение символа в позиции i на другой символ

Вставка символа - добавление нового символа в позицию i

Удаление символа - удаление символа из позиции i

Пошаговое описание алгоритма

1. Инициализация

Создается двумерная таблица editDistance[n+1][m+1], где: n = длина строки A, m = длина строки B, editDistance[i][j] = минимальная стоимость преобразования первых i символов строки A в первые j символов строки B

2. Базовые случаи

Первая строка (i=0): преобразование пустой строки в префикс B длины j требует j операций вставки, первый столбец (i=0): преобразование префикса A длины i в пустую строку требует i операций удаления

3. Заполнение таблицы

Для каждой пары позиций (i,j) рассматриваются два случая:

Случай 1: Символы совпадают (A[i-1] == B[j-1])

Дополнительных операций не требуется.

Случай 2: Символы не совпадают (A[i-1] != B[j-1]) Выбирается минимум из трех вариантов:

Замена: editDistance[i-1][j-1] + replaceCost

Вставка: editDistance[i][j-1] + insertCost

Удаление: editDistance[i-1][j] + deleteCost

4. Получение результата

Значение editDistance[n][m] содержит минимальную стоимость преобразования строки A в строку B.

Задание 2

Расширенная версия алгоритма из прошлого задания, которая дополнительно восстанавливает последовательность операций для преобразования строки A в строку B.

Дополнительный функционал:

В конце программы добавлен блок кода для обратного прохода по таблице (backtracking): восстановление пути: начиная с ячейки [n][m], программа движется назад к [0][0], определяя какая операция была выполнена на каждом шаге.

Кодирование операций:

M - совпадение символов (Match)

R - замена символа (Replace)

I - вставка символа (Insert)

D - удаление символа (Delete)

Задание 3

Алгоритм поиска расстояния Левенштейна и наибольшей общей подстроки

Общее описание - данный код реализует два алгоритма динамического программирования: вычисление расстояния Левенштейна между двумя строками, поиск наибольшей общей подстроки в двух строках

Входные данные - две строки: S и T для сравнения

Алгоритм 1: Расстояние Левенштейна

Цель - найти минимальное количество операций редактирования (вставка, удаление, замена) для преобразования строки S в строку T, где каждая операция имеет стоимость 1.

Принцип работы

Инициализация: создается таблица editDistance[n+1][m+1], первая строка: editDistance[0][j] = j (j вставок), первый столбец: editDistance[i][0] = i (i удалений)

Заполнение таблицы: если S[i-1] == T[j-1]: editDistance[i][j] = editDistance[i-1][j-1], если символы разные: editDistance[i][j] = min(замена, вставка, удаление) + 1

Результат: editDistance[n][m] - минимальное расстояние редактирования

Алгоритм 2: Наибольшая общая подстрока

Цель - найти самую длинную подстроку, которая встречается в обеих строках подряд.

Принцип работы

Инициализация: создается таблица substringEditDistance[n+1][m+1], заполненная нулями.

Заполнение таблицы: если S[i-1] == T[j-1]: substringEditDistance[i][j] = substringEditDistance[i-1][j-1] + 1, если символы разные: substringEditDistance[i][j] = 0 (прерывание последовательности).

Отслеживание максимума: ведется учет максимального значения в таблице, запоминается позиция окончания наибольшей подстроки

Восстановление результата: По найденной позиции извлекается сама подстрока

Оценка сложности алгоритма:

Временная сложность:

Алгоритм перебирает все пары префиксов строк S (длиной n) и T (длиной m). Для каждой пары символов выполняется фиксированное

количество операций: сравнение символов и выбор минимального значения из трёх возможных (вставка, удаление, замена). Инициализация первого столбца и первой строки занимает O(n + m), но не влияет на итоговую асимптотику.

Итоговая временная сложность — O(n * m), так как таблица размером n × m заполняется полностью.

Пространственная сложность

Хранение промежуточных результатов: нужно помнить количество шагов для всех комбинаций частей S и T — примерно n·m значений. Каждое значение — целое число, занимает фиксированное место.

Дополнительная память: Переменные n, m — фиксированная память.

Итог: O(n * m).

Вывод

В ходе выполнения лабораторной работы были реализованы алгоритмы для нахождения редакционного расстояния и расстояния Левенштейна.