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 5а

Студент гр. 3388 _____ Потапов Р.Ю.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург
2025

Цель работы:

Изучить алгоритмы Левенштейна для нахождения редукционного расстояния. Также реализовать задание по варианту.

Задание.

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*, $1 \le |S| \le 2550$).

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

Параметры выходных данных:

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

Sample Input:

pedestal

stien

Sample Output:

7

Реализация

Описание алгоритма Левенштейна:

Программа реализует алгоритм Левенштейна, который является классическим методом динамического программирования для вычисления редакционного расстояния между двумя строками. Редакционное расстояние — это минимальное количество операций редактирования (вставка, удаление, замена), необходимых для преобразования строки S в строку T.

Шаги алгоритма

Сначала определяется, сколько шагов нужно, чтобы из пустой строки сделать T, просто добавляя символы T один за другим.

Затем определяется, сколько шагов нужно, чтобы убрать все символы из S и получить пустую строку. Для первых i символов S это i удалений — i шагов.

Алгоритм проходит по всем возможным частям S (от одного символа до всей строки) и частям T (от одного символа до всей строки), сравнивая их символы. Для каждой пары позиций в S и T:

- Если текущие символы одинаковые, ничего не нужно делать — количество шагов остаётся таким же, как было для частей строк до этих символов.
- Если символы разные, рассматриваются три варианта:
    - **Замена:** поменять символ в S на символ из T. Это добавляет один шаг к тому, что было до этого.
    - **Вставка:** добавить символ из T в S. Это тоже добавляет один шаг к тому, что было для T без этого символа.
    - **Удаление:** убрать символ из S. Это добавляет один шаг к тому, что было для S без этого символа.
- Из этих трёх вариантов выбирается тот, который требует меньше всего шагов в сумме.

После того как алгоритм рассмотрит все символы S и T, он знает, сколько минимально шагов нужно, чтобы превратить всю строку S в T. Это число и есть ответ.

Описание функций:

1 main()

1.1 Считывание входных данных: первой строкой читаются четыре числовых значения (стоимости операций замены, вставки одного символа, удаления одного символа и удаления двух символов подряд) и переводятся в числа с плавающей точкой. Далее считываются две строки A и B.

1.2 Проверка тривиальных случаев:

– Если A пустая (n=0), то ответ равен m × cost_insert (стоимость вставки всех символов B). Выводится либо целое число (если произведение давало целое), либо дробное с точностью исходного умножения.

– Если B пустая (m=0), то ответ равен n × cost_delete (обычная стоимость удаления всех символов A). Аналогично форматируется вывод.

1.3 Инициализация двух массивов для динамического программирования:

– dp_prev2 длины (m+1) заполняется значениями j × cost_insert (стоимость вставки первых j символов B в пустую строку).

– dp_prev1 длины (m+1) создаётся пустым, но в dp_prev1[0] сразу записывается cost_delete (стоимость удаления первого символа A, когда B ещё пуст). Затем с помощью цикла по j от 1 до m считается dp_prev1[j] — минимальная стоимость перевода префикса A[:1] в префикс B[:j] без учёта операции удаления двух символов подряд (delete2): для каждой j вычисляются стоимости трёх классических операций (delete1, insert, replace) из предыдущего состояния dp_prev2 и dp_prev1.

1.4 Обработка случая n=1: если A состоит ровно из одного символа, результат уже накоплен в dp_prev1[m], и функция выводит его, завершив работу.

1.5 Основной цикл по i от 2 до n (работа с A[:i]):

– Создаётся новый массив current длины (m+1). В ячейке current[0] сохраняется минимальная стоимость перевода префикса A[:i] в пустую

строку B[:0]. Тут сравниваются две возможности: удалить i символов подряд по одной (i × cost_delete) или, если два последних символа A[i-2] и A[i-1] различаются, сделать удаление “двух сразу” (cost_delete2) плюс стоимость преобразования A[:i-2]→“”.

– Затем для каждой j от 1 до m считается current[j] как минимум из четырёх вариантов:

• delete1 = dp_prev1[j] + cost_delete (удалить очередной символ из A)

• insert = current[j-1] + cost_insert (вставить B[j-1] в A[:i])

• replace_val = dp_prev1[j-1] + (0 или cost_replace), в зависимости от совпадения A[i-1] и B[j-1]

• delete2_val = dp_prev2[j] + cost_delete2 (если A[i-2] ≠ A[i-1]).

Минимум из этих четырёх даёт current[j].

– После обработки всего j текущий уровень current становится новым dp_prev1, а прежний dp_prev1 (старый для i−1) сохраняется в dp_prev2 (для i−2).

1.6 В конце цикла результатом служит dp_prev1[m] — минимальная стоимость преобразования всей строки A в строку B. Выводится либо как целое (если число получилось целым), либо как дробное.

Таким образом, вся логика свёрстана в функции main. Переменные dp_prev2 и dp_prev1 хранят предыдущие два уровня динамики, а массив current рассчитывает следующий уровень, используя четыре возможных операции (replace, insert, delete один символ, delete два символа сразу).

Оценка сложности алгоритма:

Временная сложность

• Алгоритм проходит по всем i от 1 до n и для каждого i по всем j от 1 до m, выполняя для каждой пары (i,j) одну фиксированную сумму действий: вычисление стоимости удаления, вставки, замены и (при возможности) двойного удаления.

• На каждом шаге сравниваются два символа и выбирается минимум из

четырёх чисел, что требует O(1) времени.

• Плюс в начале выполняются единичные проходы длины m и длины n для базовых случаев, что даёт O(n+m).

Итого: O(n·m).

Пространственная сложность

• В любой момент хранятся три массива длины m+1: dp_prev2, dp_prev1 и current. То есть требуется O(m) памяти.

• Дополнительно используются постоянные по размеру переменные (счетчики, стоимости операций).

• Строки A и B сами по себе входят во входные данные и в подсчёт дополнительной памяти не включаются.

Итого: O(m).

Тестирование

Таблица 1. Тестирование.

Входные данные	Выходные данные
entrance reenterable	7
cat cat	0
cat cot	1
dog doing	2
 hello	5
kitten sitting	3

Вывод

В ходе лабораторной работы были написаны программы с использованием алгоритма Левенштейна.