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 1.

Студентка гр. 3388 _____ Титкова С.Д.

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

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

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 и структур:

- *int levenshtein_distance(const string& S, const string& T, const map<char, int>& special_replace_costs, const map<char, int>& special_insert_costs, bool debug = false)* – функция, которая определяет, сколько минимально действий (замен, вставок, удалений) нужно, чтобы превратить строку S в строку T, где каждое действие стоит ровно один шаг.

Оценка сложности алгоритма:

Временная сложность:

Сравнение строк:

- Алгоритм проверяет все комбинации частей S (длина n) и T (длина m), сравнивая символы и выбирая лучший вариант.
- Для каждой пары позиций делается фиксированная работа: сравнение символов и выбор минимума из трёх чисел.
- Начальная подготовка (вставки T и удаления S) занимает n+m.

Итог: O(n·m)

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.
- Строки S и T — часть входных данных, не считаются

Итог: O(n·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.

Исходный код программы см. в ПРИЛОЖЕНИИ А.

ПРИЛОЖЕНИЕ А.
ИСХОДНЫЙ КОД ПРОГРАММЫ

Levenshtein.cpp

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <map>
using namespace std;

int levenshtein_distance(const string& S, const string& T,
                         const map<char, int>& special_replace_costs,
                         const map<char, int>& special_insert_costs,
                         bool debug = false) {
    int n = S.size(), m = T.size();
    vector<vector<int>> dp(n + 1, vector<int>(m + 1));

    for (int j = 0; j <= m; ++j) {
        dp[0][j] = j;
    }
    for (int i = 0; i <= n; ++i) {
        dp[i][0] = i;
    }

    if (debug) {
        cout << "Начальная таблица (до заполнения):" << endl;
        for (int i = 0; i <= n; ++i) {
            for (int j = 0; j <= m; ++j) {
                cout << dp[i][j] << " ";
            }
            cout << endl;
        }
        cout << endl;
    }

    for (int i = 1; i <= n; ++i) {
        for (int j = 1; j <= m; ++j) {
            if (debug) {
                cout << "Заполняем dp[" << i << "][" << j << "],
сравниваем S[" << i-1 << "] = '" << S[i-1]
                     << "' с T[" << j-1 << "] = '" << T[j-1] << "': ";
            }

            if (S[i - 1] == T[j - 1]) {
                dp[i][j] = dp[i - 1][j - 1];
                if (debug) {
                    cout << "Совпадение, dp[" << i << "][" << j << "]
= dp[" << i-1 << "][" << j-1 << "] = " << dp[i][j] << endl;
                }
            } else {
                int del_cost = dp[i - 1][j] + 1;
```

```
                int ins_cost = dp[i][j - 1] +
(special_insert_costs.count(T[j - 1]) ? special_insert_costs.at(T[j -
1]) : 1);
                int sub_cost = dp[i - 1][j - 1] +
(special_replace_costs.count(S[i - 1]) ? special_replace_costs.at(S[i
- 1]) : 1);

                dp[i][j] = min({del_cost, ins_cost, sub_cost});
                if (debug) {
                    cout << "Нет совпадения, варианты: удаление=" <<
del_cost << ", вставка=" << ins_cost
                         << ", замена=" << sub_cost << ", dp[" << i <<
"][" << j << "] = " << dp[i][j] << endl;
                }
            }

            if (debug) {
                cout << "Таблица после заполнения dp[" << i << "][" <<
j << "]:" << endl;
                for (int x = 0; x <= n; ++x) {
                    for (int y = 0; y <= m; ++y) {
                        cout << dp[x][y] << " ";
                    }
                    cout << endl;
                }
                cout << endl;
            }
        }
    }

    if (debug) {
        cout << "Итоговое расстояние: " << dp[n][m] << endl;
    }
    return dp[n][m];
}

int main() {
    string S, T;
    map<char, int> special_replace_costs;
    map<char, int> special_insert_costs;

    cout << "Введите первую строку (S): ";
    cin >> S;
    cout << "Введите вторую строку (T): ";
    cin >> T;

    int num_replace;
    cout << "Введите количество особо заменяемых символов: ";
    cin >> num_replace;
    for (int i = 0; i < num_replace; ++i) {
        char symbol;
        int cost;
        cout << "Введите " << i + 1 << "-й особо заменяемый символ: ";
        cin >> symbol;
        cout << "Введите стоимость замены для символа '" << symbol <<
"': ";
        cin >> cost;
        special_replace_costs[symbol] = cost;
```

```
    }

    int num_insert;
    cout << "Введите количество особо добавляемых символов: ";
    cin >> num_insert;
    for (int i = 0; i < num_insert; ++i) {
        char symbol;
        int cost;
        cout << "Введите " << i + 1 << "-й особо добавляемый символ:
";
        cin >> symbol;
        cout << "Введите стоимость вставки для символа '" << symbol <<
"': ";
        cin >> cost;
        special_insert_costs[symbol] = cost;
    }

    int result = levenshtein_distance(S, T, special_replace_costs,
special_insert_costs, true);  // debug = true для отладки
    cout << "Расстояние Левенштейна: " << result << endl;

    return 0;
}
```