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. 3388 _____ Снигирев А.А

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург

2025

Цель работы

Изучить алгоритмы Левенштейна и Вагнера-Фишера и принцип расчета редакционного расстояния. Реализовать алгоритм с дополнительной модификацией.

Задание 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. 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.

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.

Индивидуализация

Вариант 2

"Особый заменитель и особо удаляемый символ": цена замены на определённый символ отличается от обычной цены замены; цена удаления другого (или того же) определённого символа отличается от обычной цены удаления. Особый заменитель и цена замены на него, особо удаляемый символ и цена его удаления — дополнительные входные данные.

Выполнение работы

В сущности, интерес представляет только задание номер 2. Первое задание считает только цену операций без предписания, а третье еще и использует фиксированные цены. Поэтому объяснено будет только второе задание и модификация.

Редакционное расстояние - метрика, измеряющая по модулю разность между двумя последовательностями символов. Она определяется как минимальное количество односимвольных операций, необходимых для превращения одной последовательности символов в другую.

Алгоритм Левенштейна предполагает, что все операции имеют цену 1.

Алгоритм Вагнера-Фишера — обобщение, позволяющее цены назначать.

Идея алгоритма

Алгоритм строит матрицу размером N x M, где N – длина исходной строки, M – длина строки, до которой нужно вычислить редакционное расстояние. Строит он её динамически, используя уже вычисленные ранее значения. Конкретнее - «уголок», образуемый элементами (i-1,j-1), (i, j-1), (j, i-1).

Алгоритм считает сумму элемента и соответствующей операции и берет минимальное из трех значений для записи в ячейку.

Подробнее:

DP[i][j] = min(DP[i-1][j-1]+replace_cost, DP[i][j-1]+ins_cost, DP[i-1][j])

Если символы-проекции с текущей клетки равны, то просто DP[i][j] = DP[i-1][j-1].

Для расчета редакционного расстояния этого достаточно — ответ будет лежать в нижнем правом углу таблицы DP.

В коде за построение таблицы отвечает функция *int editDistance,* точнее ее первые три цикла.

Редакционное предписание

Редакционное предписание — более сложная задача. Для решения была создана дополнительная таблица *prev*, которая хранит математические векторы, что показывают, из какого состояния получилось текущее.

Алгоритм как бы прокладывает дорогу из правого нижнего в левый верхний угол, перемещаясь по векторам.

Конкретные операции также хранятся в таблице и копируются в массив операций на каждой пройденной клетке.

Индивидуализация

Главная часть алгоритма — построение таблицы. Таблица строится по довольно простым правилам и каждый элемент напрямую зависит от веса операции. Это предоставляет простор для обработки частных случаев строк и интересных операций. Главное — правильно прописать в коде новые условия и функции.

В конкретно данном варианте особенность следующая:

Цена замены и удаления не фиксированная, а зависит от операндов. Поэтому в обработке этих операций требуется дополнительно проверять текущие символы-проекции на совпадение с «особыми символами», а не только «уголок», чтобы правильно все расчитать.

Исходный код программы см. в **ПРИЛОЖЕНИИ А**

Оценка сложности алгоритма:

Построение таблицы занимает большую часть времени и ресурсов:

Время — O(N*M), именно столько операций требуется для заполнения матрицы DP.

Память — **O(N*M)**, столько памяти требуют таблицы DP, prev и operations.

Буферы для строк асимптотически занимают меньше памяти.

Редакционное предписание выполняется за **O(N+M)** операций.

Модификация немного увеличивает константные затраты на расчет каждой ячейки, но на аппроксимацию влияния не имеет.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

```
#include <iostream>
#include <vector>
#include <string>
#include <climits>
#include <iomanip>

using namespace std;


struct Operation {
    string type;
    int pos;
    char a, b;
};


int editDistance(const string& A, const string& B, int cr, int
ci, int cd,
                 char special_replace_char, int
special_replace_cost,
                 char special_delete_char, int
special_delete_cost) {
    int n = A.length();
    int m = B.length();


    cout << "=== Начало работы алгоритма ===" << endl;
    cout << "Строка A: " << A << " (длина = " << n << ")" <<
endl;
    cout << "Строка B: " << B << " (длина = " << m << ")" <<
endl;
    cout << "Стоимости: замена = " << cr << ", вставка = " <<
ci << ", удаление = " << cd << endl;
    cout << "Особый заменитель: '" << special_replace_char <<
"' (стоимость = " << special_replace_cost << ")" << endl;
```

```
        cout << "Особый удаляемый символ: '" << special_delete_char
<< "' (стоимость = " << special_delete_cost << ")" << endl;



        vector<vector<long long>> dp(n + 1, vector<long long>(m +
1, LLONG_MAX));

        vector<vector<pair<int, int>>> prev(n + 1, vector<pair<int,
int>>(m + 1, {-1, -1}));

        vector<vector<string>> op(n + 1, vector<string>(m + 1,
""));



        dp[0][0] = 0;
        cout << "\nБазовые случаи:" << endl;
        cout << "dp[0][0] = 0 (пустые строки)" << endl;
        for (int i = 1; i <= n; ++i) {
            long long delete_cost = (A[i-1] == special_delete_char)
? special_delete_cost : cd;
            string marker = (A[i-1] == special_delete_char) ?
"special_delete" : "delete";
            dp[i][0] = dp[i-1][0] + delete_cost;
            prev[i][0] = {i - 1, 0};
            op[i][0] = marker;
            cout << "dp[" << i << "][0] = " << dp[i][0] << " (" <<
marker << " '" << A[i-1] << "', стоимость = " << delete_cost << ")"
<< endl;
        }
        for (int j = 1; j <= m; ++j) {
            dp[0][j] = dp[0][j-1] + ci;
            prev[0][j] = {0, j - 1};
            op[0][j] = "insert";
            cout << "dp[0][" << j << "] = " << dp[0][j] << "
(insert '" << B[j-1] << "', стоимость = " << ci << ")" << endl;
        }
```

```
        cout << "\nЗаполнение"<<endl;

        for (int i = 1; i <= n; ++i) {
            for (int j = 1; j <= m; ++j) {
                cout << "\nРассмотрение dp[" << i << "][" << j <<
"] (A[" << i-1 << "] = '" << A[i-1] << "', B[" << j-1 << "] = '" <<
B[j-1] << "')" << endl;
                long long min_cost = dp[i][j];

                if (A[i-1] == B[j-1]) {
                    if (dp[i-1][j-1] < dp[i][j]) {
                        dp[i][j] = dp[i-1][j-1];
                        prev[i][j] = {i-1, j-1};
                        op[i][j] = "match";
                        cout << "  Совпадение: dp[" << i << "][" <<
j << "] = " << dp[i][j] << " (match)" << endl;
                    }
                } else {

                    long long replace_cost = (B[j-1] ==
special_replace_char) ? special_replace_cost : cr;
                    string marker = (B[j-1] ==
special_replace_char) ? "special_replace" : "replace";
                    if (dp[i-1][j-1] + replace_cost < dp[i][j]) {
                        dp[i][j] = dp[i-1][j-1] + replace_cost;
                        prev[i][j] = {i-1, j-1};
                        op[i][j] = marker;
                        cout << "  Замена: dp[" << i << "][" << j
<< "] = " << dp[i][j] << " (" << marker << ", стоимость = " <<
replace_cost << ")" << endl;
                    }


                    long long delete_cost = (A[i-1] ==
special_delete_char) ? special_delete_cost : cd;
                    long long delete_insert_cost = delete_cost +
ci;
```

```
                    if (dp[i-1][j-1] + delete_insert_cost < dp[i]
[j]) {
                        dp[i][j] = dp[i-1][j-1] +
delete_insert_cost;
                        prev[i][j] = {i-1, j-1};
                        op[i][j] = "delete_insert";
                        cout << "  Удаление+вставка: dp[" << i <<
"][" << j << "] = " << dp[i][j] << " (delete_insert, стоимость = "
<< delete_insert_cost << ")" << endl;
                    }
                }

                if (dp[i][j-1] + ci < dp[i][j]) {
                    dp[i][j] = dp[i][j-1] + ci;
                    prev[i][j] = {i, j-1};
                    op[i][j] = "insert";
                    cout << "  Вставка: dp[" << i << "][" << j <<
"] = " << dp[i][j] << " (insert '" << B[j-1] << "', стоимость = " <<
ci << ")" << endl;
                }

                long long delete_cost = (A[i-1] ==
special_delete_char) ? special_delete_cost : cd;
                string marker = (A[i-1] == special_delete_char) ?
"special_delete" : "delete";
                if (dp[i-1][j] + delete_cost < dp[i][j]) {
                    dp[i][j] = dp[i-1][j] + delete_cost;
                    prev[i][j] = {i-1, j};
                    op[i][j] = marker;
                    cout << "  Удаление: dp[" << i << "][" << j <<
"] = " << dp[i][j] << " (" << marker << " '" << A[i-1] << "',
стоимость = " << delete_cost << ")" << endl;
                }
            }
        }


        cout << "\nТаблица DP:" << endl;
```

```
        cout << setw(8) << " ";
        for (int j = 0; j <= m; ++j) {
            cout << setw(8) << (j == 0 ? " " : string(1, B[j-1]));
        }
        cout << endl;
        for (int i = 0; i <= n; ++i) {
            cout << setw(8) << (i == 0 ? " " : string(1, A[i-1]));
            for (int j = 0; j <= m; ++j) {
                if (dp[i][j] == LLONG_MAX) {
                    cout << setw(8) << "INF";
                } else {
                    cout << setw(8) << dp[i][j];
                }
            }
            cout << endl;
        }



        long long minCost = dp[n][m];
        cout << "\nМинимальная стоимость: " << minCost << endl;



        cout << "\nВосстановление операций:" << endl;
        vector<Operation> operations;
        int i = n, j = m;
        int step = 0;
        while (i > 0 || j > 0) {
            int pi = prev[i][j].first;
            int pj = prev[i][j].second;
            string operation = op[i][j];
            cout << "Шаг " << step++ << ": (" << i << "," << j <<
") -> (" << pi << "," << pj << "), операция = " << operation;

            if (operation == "match") {
                cout << " (символы '" << A[i-1] << "' совпадают)"
<< endl;
                operations.push_back({"match", i-1, 'n', 'n'});
            } else if (operation == "replace") {
```

```
                cout << " (замена '" << A[i-1] << "' на '" << B[j-
1] << "', стоимость = " << cr << ")" << endl;
                operations.push_back({"replace", i-1, A[i-1], B[j-
1]});
            } else if (operation == "special_replace") {
                cout << " (специальная замена '" << A[i-1] << "' на
'" << B[j-1] << "', стоимость = " << special_replace_cost << ")" <<
endl;
                operations.push_back({"special_replace", i-1, A[i-
1], B[j-1]});
            } else if (operation == "insert") {
                cout << " (вставка '" << B[j-1] << "', стоимость =
" << ci << ")" << endl;
                operations.push_back({"insert", j-1, B[j-1], ' '});
            } else if (operation == "delete") {
                cout << " (удаление '" << A[i-1] << "', стоимость =
" << cd << ")" << endl;
                operations.push_back({"delete", i-1, A[i-1], ' '});
            } else if (operation == "special_delete") {
                cout << " (специальное удаление '" << A[i-1] << "',
стоимость = " << special_delete_cost << ")" << endl;
                operations.push_back({"special_delete", i-1, A[i-
1], ' '});
            } else if (operation == "delete_insert") {
                cout << " (удаление '" << A[i-1] << "' + вставка '"
<< B[j-1] << "', стоимость = "
                     << ((A[i-1] == special_delete_char) ?
special_delete_cost : cd) << " + " << ci << ")" << endl;
                operations.push_back({"insert", j-1, B[j-1], ' '});
                operations.push_back({"delete", i-1, A[i-1], ' '});
            }

            i = pi;
            j = pj;
        }
```

```
        cout << "\nПоследовательность операций (в обратном
порядке):" << endl;
        for (int k = operations.size() - 1; k >= 0; --k) {
            auto& op = operations[k];
            if (op.type == "replace") {
                cout << "R";
            } else if (op.type == "special_replace") {
                cout << "r";
            } else if (op.type == "insert") {
                cout << "I";
            } else if (op.type == "delete") {
                cout << "D";
            } else if (op.type == "special_delete") {
                cout << "d";
            } else if (op.type == "match") {
                cout << "M";
            }
        }
        cout << endl;

        return minCost;
    }

    int main() {
        string A, B;
        int cr, ci, cd, other_cr, other_cd;
        char special_ch_r, special_ch_d;
        cout<<"Цены стандартных операций: <замена> <вставка>
<удаление>"<<endl;
        cin>>cr>>ci>>cd;
        cout<<"Особая замена: <цена> <символ>"<<endl;
        cin>>other_cr>>special_ch_r;
        cout<<"Особое удаление: <цена> <символ>"<<endl;
        cin>>other_cd>>special_ch_d;
        cout<<"Строка 1:"<<endl;
        cin >> A;
        cout<<"Строка 2:"<<endl;
        cin >> B;
```

```
        int minCost = editDistance(A, B, cr, ci, cd, special_ch_r,
other_cr, special_ch_d, other_cd);
        cout<<minCost<<endl;

        return 0;
    }
```