**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: **Динамическое программирование**

|  |  |  |
| --- | --- | --- |
| Студент гр. 3342 |  | Песчатский С. Д. |
| Преподаватель |  | Виноградова Е. В. |

Санкт-Петербург

2025

## Задание

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

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.

| 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** |  |  |

**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

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≤∣*S*∣≤2550).  
Вторая строка входных данных содержит строку из строчных латинских букв.

(T, 1≤∣*T*∣≤2550).

**Параметры выходных данных:**

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

**Sample Input:**

pedestal

stien

**Sample Output:**

7

Индивидуальный вариант: 3а.  " Добавляется 4-я операция со своей стоимостью: последовательная вставка двух одинаковых символов.": цена добавления двух одинаковых символа отличается от обычной цены добавления. Цена добавления двух одинаковых символов— дополнительные входные данные.

**Описание алгоритма**

Расстояние Левенштейна - это минимальное количество операций редактирования, необходимых для преобразования одной строки в другую. Под операциями редактирования понимаются:

* Вставка символа (I)
* Удаление символа (D)
* Замена символа (R)

Алгоритм Вагнера-Фишера - это динамический подход к вычислению расстояния Левенштейна, который использует матрицу для хранения промежуточных результатов.

**Матрица динамического программирования**

Алгоритм строит матрицу VFM размером (m+1)×(n+1), где m и n - длины строк A и B соответственно. Каждая ячейка dp[i][j] содержит минимальную стоимость преобразования префикса A[0..i-1] в префикс B[0..j-1].

**Инициализация матрицы**

Матрица инициализируется следующим образом:

* VFM[0][0] = 0 (преобразование пустой строки в пустую)
* Первая строка: VFM[0][j] = VFM[0][j-1] + стоимость вставки B[j-1]
* Первый столбец: VFM[i][0] = VFM[i-1][0] + стоимость удаления A[i-1]

**Заполнение матрицы**

Для каждой пары символов A[i-1] и B[j-1]:

1. Если символы совпадают (A[i-1] == B[j-1]):  
   VFM[i][j] = VFM[i-1][j-1] (без дополнительной стоимости)
2. Если символы различаются:  
   VFM[i][j] = min(  
   VFM[i-1][j] + стоимость удаления A[i-1],  
   VFM[i][j-1] + стоимость вставки B[j-1],  
   VFM[i-1][j-1] + стоимости замены A[i-1] на B[j-1]  
   )

**Особенности реализации**

В представленной реализации учтены следующие особенности:

* Поддержка различных стоимостей операций для разных символов
* Восстановление последовательности операций по готовой матрице
* Подробный отладочный вывод для анализа работы алгоритма

**Сложность алгоритма**

Временная сложность: O(m×n) - требуется заполнить матрицу размером m×n  
Пространственная сложность: O(m×n) - хранение матрицы dp

**Пример работы**

Для строк A = "abc" и B = "cbagg" с параметрами:

* Стандартные стоимости: замена=1, вставка=1, удаление=1
* Вставка двух одинаковых символа=1

Алгоритм выполнит следующие шаги:

1. Инициализация матрицы
2. Последовательное заполнение всех ячеек
3. Восстановление оптимальной последовательности операций:
   * R: замена 'a'→'c' (стоимость=1)
   * M: совпадение 'b' (стоимость=0)
   * R: замена 'c'→'a' (стоимость=1)
   * T: двойная вставка 'g' (стоимость = 1)
4. Итоговая стоимость преобразования: 3

**Описание функций и структур данных**

* DEBUG: boolean - флаг включения отладочного режима (true/false)

**Int main()**  
Главный метод программы, выполняющий:

* Чтение входных данных (стоимости операций, строки A и B)
* Построение матрицы динамического программирования
* Вывод результатов (минимальная стоимость, операции, исходные строки)
* Отладочную информацию при включенном режиме DEBUG

**void LDTable(string A, string B, int lenA, int lenB, vector<vector<int>> & VFM , vector<vector<int>> & opers, vector<int> &cost)**

Построение матрицы динамического программирования.

Параметры:

* A: String - исходная строка
* B: String - целевая строка
* lenA: int – длина строки А
* lenB: int – длина строки B
* VFM: vector<vector<int>> - матрица стоимостей преобразований
* opers: vector<vector<int>> - матрица операций
* cost: vector<int> - стоимости операций

**void getOperations(int lenA, int lenB, vector<vector<int>> opers, string &res)**  
Восстановление последовательности операций.

Параметры:

* lenA: int – длина строки А
* lenB: int – длина строки B
* opers: vector<vector<int>> - матрица операций
* res: string – строка произведённых операций

**void printTable(int lenA, int lenB, vector<vector<int>> mat)**

Вывод матрицы динамического программирования.

Параметры:

* lenA: int – длина строки А
* lenB: int – длина строки B
* mat: vector<vector<int>>- матрица для вывода

**Тестирование**

Результаты тестирования представлены в табл. 1-4.

Таблица 1 – Результаты тестирования 1

|  |  |  |  |
| --- | --- | --- | --- |
| № п/п | Входные данные | Выходные данные | Комментарии |
|  | 1 2 3 4  duck  clack | 3  RRIMM  duck  clack | Верный вывод |
|  | 1 1 1 1  entrance  reenterable | 5  RTMMIMMRRM  entrance  reenterable | Верный вывод |

Таблица 2 – Результаты тестирования 2

|  |  |  |  |
| --- | --- | --- | --- |
| № п/п | Входные данные | Выходные данные | Комментарии |
|  | 1 1 1  kitten  sitting | 4  RRIMM  duck  clack | Верный вывод |
|  | 1 1 1 4  entrance  reenterable | 5  IIMMMIMMRRM  entrance  reenterable | Верный вывод |

Таблица 3 – Результаты тестирования 3

|  |  |  |  |
| --- | --- | --- | --- |
| № п/п | Входные данные | Выходные данные | Комментарии |
|  | 1 2 2 1  kid  children | 11  RIMIMIII  kid  children | Верный вывод |
|  | 3 2 2 1  entrance  reenterable | 12  RTMMIMMRRM  entrance  reenterable | Верный вывод |

Таблица 4 – Результаты тестирования 4

|  |  |  |  |
| --- | --- | --- | --- |
| № п/п | Входные данные | Выходные данные | Комментарии |
|  | 5 2 2 4  xyz  abc | 12  IIIDDD  xyz  abc | Верный вывод |
|  | 1 1 4 1  xyz  abc | 3  RRR  xyz  abc | Верный вывод |

# Приложение А Исходный код программы

Название файла: main.cpp

#include <iostream>

#include <vector>

using namespace std;

//глобальная константа, отвечающая за режим дебага

const bool DEBUG = true;

//функция, заполнаяющая матрицу для алгоритма Вагнера-Фишера

void LDTable(string A, string B, int lenA, int lenB, vector<vector<int>> & VFM , vector<vector<int>> & opers, vector<int> &cost){

    //проход по всей матрице VFM

    for(int j=1; j<lenB;j++){

        //заполнение первой строки стоимостью удаления

        VFM[0][j]=VFM[0][j-1]+cost[1];

        opers[0][j]=2;

    }

    int minvfm;

    for(int i=1; i<lenA; i++){

        //заполнение первого столбца стоимостью вставки

        VFM[i][0]=VFM[i-1][0]+cost[2];

        opers[i][0] = 3;

        for(int j=1; j<lenB; j++){

            //переменная для определения минимальной по стоимости операции на данном шаге

            minvfm =  min(min(VFM[i-1][j-1] + cost[0], VFM[i][j-1] + cost[1]), VFM[i-1][j] + cost[2]);

            //проверка на возможность двойной вставки

            if(j>=2 && B[j-1]==B[j-2] && (min(minvfm, VFM[i][j-2] + cost[3]) ==VFM[i][j-2] + cost[3])){

                VFM[i][j] = min(minvfm, VFM[i][j-2] + cost[3]);

                opers[i][j]=4;

                continue;

            }

            //информация о лучшей по стоимости операции, или о совпадении символов, заносится в VFM и opers

            if(A[i-1] != B[j-1]){

                VFM[i][j] = minvfm;

                if(VFM[i][j]==VFM[i-1][j-1] + cost[0]){

                    opers[i][j]=1;

                }

                if(VFM[i][j]==VFM[i][j-1] + cost[1]){

                    opers[i][j]=2;

                }

                if(VFM[i][j]==VFM[i-1][j]+cost[2]){

                     opers[i][j]=3;

                }

            }

        else{

                VFM[i][j] = VFM[i-1][j-1];

                opers[i][j]=0;

            }

        }

    }

}

//функция для получения строки операции для оптимального преобразования строк

void getOperations(int lenA, int lenB, vector<vector<int>> opers, string &res){

    //заполнение строки начинается с конца

    int i=lenA-1; int j=lenB-1;

    while(i>0 || j>0){

        if(opers[i][j]==0){

            res.append("M");

            i--;j--;

        }

        if(opers[i][j]==1){

            res.append("R");

            i--;j--;

        }

        if(opers[i][j]==2){

            if(j!=0){res.append("I");

            j--;}

        }

        if(opers[i][j]==3){

            if(i!=0){res.append("D");

            i--;}

        }

        if(opers[i][j]==4){

            if(j>=2){res.append("T");

                j--;j--;}

        }

    }

}

void printTable(int lenA, int lenB, vector<vector<int>> mat){

        for(int i=0; i<lenA; i++){

            for(int j=0; j<lenB; j++){

                cout<<mat[i][j]<<" ";

            }

            cout<<endl;

        }

}

int main(){

    //чтение стоимостей операций

    vector<int> cost(3);

    //cin>>cost[0]>>cost[1]>>cost[2]>>cost[3];

    cost[0] = 1; cost[1] = 1; cost[2] = 1; cost[3] = 125;

    //чтение строк A и B

    string A, B;

    cin>>A;

    cin>>B;

    int lenA = A.length()+1;

    int lenB = B.length()+1;

    //построение матриц VFM и opers

    //в матрицу opers записываются операции

    vector<vector<int>> VFM;

    vector<int> tmp(lenB);

    for(int j=0; j<lenA; j++){

        VFM.push\_back(tmp);

    }

    vector<vector<int>> opers = VFM;

    for(int i=0; i<lenA; i++){

        for(int j=0; j<lenB; j++){

            VFM.at(i).at(j)=0;

        }

    }

    //заполнение матриц VFM и opers

    LDTable(A, B, lenA, lenB, VFM, opers, cost);

    //получение строки необходимых оперций для преобразования строки A в строку B

    string res;

    getOperations(lenA, lenB, opers, res);

    //вывод матриц VFM и opers если программа в режиме дебага

    if(DEBUG){

        printTable(lenA, lenB, VFM);

        cout<<endl;

        printTable(lenA, lenB, opers);

    }

    //вывод расстояния Левенштейна

    cout<<VFM[lenA-1][lenB-1]<<endl;

    //вывод строки операций и строк A и B

    string ans=res;

    //строку операций необходимо отразить по скольку она восстанавливается в обратон порядке

    for(int i=0; i<size(res); i++){

        ans[i]=res[size(res)-i-1];

    }

    cout<<ans<<endl<<A<<endl<<B;

    return 0;

}