**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МОЭВМ**

отчет

**по лабораторной работе №3**

**по дисциплине «Построение и анализ алгоритмов»**

Тема: «**Редакционное расстояние**»

Вариант 13

|  |  |  |
| --- | --- | --- |
|  |  | Гусакова К.А. |
| Студент гр. 3388 |  |  |
| Преподаватель |  | Жангиров Т.Р. |

Санкт-Петербург

2025**ЗАДАНИЕ НА ЛАБОРАТОРНУЮ РАБОТУ**

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 на вариант:

Вывести не одно, а все редакционные предписания с минимальной стоимостью. Для одного из предписаний продемонстрировать его применение для преобразования 1-ой строки во 2-ую.

**Описание алгоритма для решения задачи**

Алгоритм, реализованный в коде, решает задачу поиска самого дешевого преобразования одной строки в другую, используя операции с учётом их стоимости. Операции: замена символа, вставка и удаление. Каждая операция имеет заданную стоимость, и цель – получить строку B из строки A с минимальной суммарной стоимостью.

В основе алгоритма лежит метод динамического программирования, который строит таблицу (dp), где каждая ячейка содержит минимальную стоимость преобразования префикса строки A в префикс строки B, а также информацию о том, какая операция была применена (замена, вставка и т. д.).

Вот как работает алгоритм шаг за шагом:

Сначала инициализируется таблица dp, где dp[i][j] – это структура, содержащая:

* cost: минимальная стоимость преобразования первых i символов строки A в первые j символов строки B;
* operation: символ, обозначающий последнюю операцию на этом шаге (M – match, R – замена, I – вставка, D – удаление).

Пример начала инициализации:

dp[0][0] = { 0, 'M' }; // Нулевая стоимость: ничего не преобразуем

for (int i = 1; i <= m; ++i)

dp[i][0] = { dp[i - 1][0].cost + cost\_delete, 'D' }; // Удаляем все символы A

for (int j = 1; j <= n; ++j)

dp[0][j] = { dp[0][j - 1].cost + cost\_insert, 'I' }; // Вставляем все символы B

Это задаёт начальные состояния: чтобы получить пустую строку из A, нужно удалить все её символы; чтобы получить B из пустой строки, нужно вставить все её символы.

Далее заполняется остальная часть таблицы. На каждом шаге рассматриваются три варианта:

* Если символы A[i-1] и B[j-1] совпадают, операция – M, и стоимость не увеличивается.
* Иначе выбирается минимальная по стоимости из трёх операций:

R (замена) – берётся стоимость ячейки dp[i-1][j-1] + cost\_replace;

I (вставка) – берётся dp[i][j-1] + cost\_insert;

D (удаление) – берётся dp[i-1][j] + cost\_delete.

Пример фрагмента:

if (A[i - 1] == B[j - 1]) {

dp[i][j] = { dp[i - 1][j - 1].cost, 'M' };

} else {

int replace\_cost = dp[i - 1][j - 1].cost + cost\_replace;

int insert\_cost = dp[i][j - 1].cost + cost\_insert;

int delete\_cost = dp[i - 1][j].cost + cost\_delete;

if (replace\_cost <= insert\_cost && replace\_cost <= delete\_cost) {

dp[i][j] = { replace\_cost, 'R' };

} else if (insert\_cost <= replace\_cost && insert\_cost <= delete\_cost) {

dp[i][j] = { insert\_cost, 'I' };

} else {

dp[i][j] = { delete\_cost, 'D' };

}

}

После того как таблица dp полностью построена, итоговая стоимость преобразования – это значение dp[m][n].cost, где m и n – длины строк A и B соответственно.

Затем запускается отдельная рекурсивная функция, которая восстанавливает все возможные последовательности операций, давшие минимальную стоимость. Это делается путём обхода таблицы dp в обратном порядке – от dp[m][n] к dp[0][0].

На каждом шаге проверяется, какая операция могла привести к текущей ячейке при данной стоимости, и строится строка операций в обратном порядке. Если найден конец пути (ячейка dp[0][0]), строка операций сохраняется.

Пример восстановления пути:

if (A[i - 1] == B[j - 1] && dp[i][j].cost == dp[i - 1][j - 1].cost) {

find\_all\_paths(dp, A, B, i - 1, j - 1, current + 'M', all\_paths, ...);

}

if (A[i - 1] != B[j - 1] && dp[i][j].cost == dp[i - 1][j - 1].cost + cost\_replace) {

find\_all\_paths(dp, A, B, i - 1, j - 1, current + 'R', all\_paths, ...);

}

Затем показывается применение одной из таких последовательностей. Строка A по шагам превращается в B, и после каждой операции (замена, вставка, удаление) выводится текущее состояние строки. При этом используется индекс i, который указывает на текущую позицию в строке, чтобы изменения применялись строго слева направо и только к оригинальным символам, а не к тем, что были добавлены во время.

**Оценка сложности алгоритма по времени и памяти**

**Временная сложность:**

### **build\_dp()**

for (int i = 1; i <= m; ++i)

dp[i][0] = { dp[i - 1][0].cost + cost\_delete, 'D' };

for (int j = 1; j <= n; ++j)

dp[0][j] = { dp[0][j - 1].cost + cost\_insert, 'I' };

for (int i = 1; i <= m; ++i) {

for (int j = 1; j <= n; ++j) {

}

}

Двойной цикл по i и j **–** O(m \* n)

Каждый элемент вычисляется за O(1) времени   
 Временная сложность: O(m \* n)

**2. find\_all\_paths() – Нахождение всех путей преобразований**

Рекурсивно находит все возможные пути с минимальной стоимостью редактирования от (m, n) до (0, 0) по матрице dp.

if (A[i - 1] == B[j - 1] && ...) // M

if (A[i - 1] != B[j - 1] && ...) // R

if (dp[i][j].cost == dp[i][j - 1].cost + cost\_insert) // I

if (dp[i][j].cost == dp[i - 1][j].cost + cost\_delete) // D

O(3(m + n))в худшем случае, в каждой точке (i, j) может быть до трех рекурсивных вызовов (если все три операции допустимы и равны по стоимости). Глубина рекурсии максимум m + n(максимальная проходка).

Временная сложность: O(3(m + n))

**3. apply\_operations() – пошаговое применение одного пути преобразования**

O(m + n),  
где m = A.length(), n = B.length()

Обрабатываем operations, длина которой не больше m + n (максимум m удалений + n вставок).

Каждая операция:

M, R **–** O(1)

I, D **–** вставка/удаление **–** O(k), где k **–** длина строки

Тк идет вывод каждой строки, то (m+n)(m+n)

Временная сложность**:** O(m + n)2

**Общая временная сложность программы**:

O(3(m+n))

**Пространственная сложность:**

**1. build\_dp() – Построение таблицы DP**

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

* Хранится таблица размером (m + 1) × (n + 1).
* Каждый элемент (Cell) занимает **O(1)** памяти.

Пространственная сложность:  
O(m \* n) – память под таблицу.

**2. find\_all\_paths() – Нахождение всех путей преобразований**

Основные ресурсы:

* Рекурсивный стек глубиной до m + n.
* all\_paths – вектор строк, каждая длиной до m + n.
* Строка current – создается на каждом уровне рекурсии.

В худшем случае число путей экспоненциальное: до 3^(m+n).

Пространственная сложность:  
O(p \* (m + n)), где p – количество оптимальных путей

**3. apply\_operations() – Применение одного пути преобразования**

Используемая память:

* result – копия строки A O(m)
* operations – длина до m + n
* Дополнительно: временные строки при вставках/удалениях (в пределах O(m + n))

Пространственная сложность:  
O(m + n) – максимум длина промежуточной строки result.

**Общая пространственная сложность программы**:

O(3m+n \*(m+n))

**Описание способа хранения частичных решений.**

Частичные решения хранятся в таблице dp типа vector<vector<Cell>>.

struct Cell {

int cost

char operation;

};

**Использованные оптимизации.**

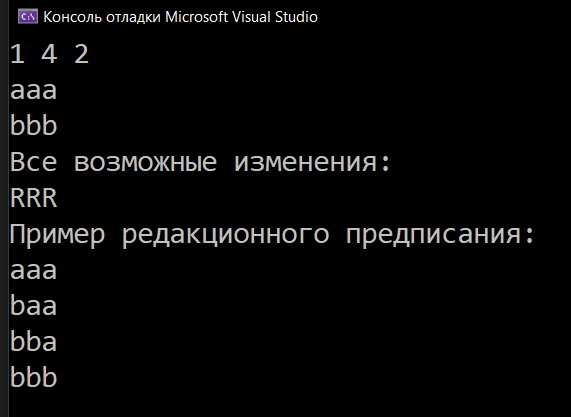
Сокращение глубины вызовов.

В find\_all\_paths() используется проверка:

if (i == 0 && j == 0)

которая завершает рекурсию как только достигнут старт пути (нижний угол таблицы).

**Тестирование. Показ граничных случаев алгоритма.**



A screenshot of a computer

AI-generated content may be incorrect.Рисунок 1 – показ базового случая с разной стоимостью

Рисунок 2 – показ с одинаковой стоимостью

A screen shot of a computer

AI-generated content may be incorrect.Рисунок 3 – показ случая с высокой стоимостью одной операции.

**ВЫВОД**

В ходе лабораторной работы был реализован алгоритм редактирования строк с учетом различных стоимостей операций, основанный на динамическом программировании. Алгоритм корректно определяет оптимальные пути преобразования и применяет их к строкам. Проведённый анализ подтвердил его эффективность и соответствие заявленной вычислительной сложности.

**ПРИЛОЖЕНИЕ. ИСХОДНЫЙ КОД ПРОГРАММЫ**

#include <iostream>  
#include <string>  
#include <vector>  
#include <algorithm>  
  
using namespace std;  
  
  
struct Cell {  
 int cost;  
 char operation;  
};  
  
  
vector<vector<Cell>> build\_dp(const string& A, const string& B, int cost\_replace, int cost\_insert, int cost\_delete) {  
 int m = A.length();  
 int n = B.length();  
 vector<vector<Cell>> dp(m + 1, vector<Cell>(n + 1));  
  
 dp[0][0] = { 0, 'M' };  
 for (int i = 1; i <= m; ++i)  
 dp[i][0] = { dp[i - 1][0].cost + cost\_delete, 'D' };  
 for (int j = 1; j <= n; ++j)  
 dp[0][j] = { dp[0][j - 1].cost + cost\_insert, 'I' };  
  
 for (int i = 1; i <= m; ++i) {  
 for (int j = 1; j <= n; ++j) {  
 if (A[i - 1] == B[j - 1]) {  
 dp[i][j] = { dp[i - 1][j - 1].cost, 'M' };  
 }  
 else {  
 int replace\_cost = dp[i - 1][j - 1].cost + cost\_replace;  
 int insert\_cost = dp[i][j - 1].cost + cost\_insert;  
 int delete\_cost = dp[i - 1][j].cost + cost\_delete;  
  
 if (replace\_cost <= insert\_cost && replace\_cost <= delete\_cost) {  
 dp[i][j] = { replace\_cost, 'R' };  
 }  
 else if (insert\_cost <= replace\_cost && insert\_cost <= delete\_cost) {  
 dp[i][j] = { insert\_cost, 'I' };  
 }  
 else {  
 dp[i][j] = { delete\_cost, 'D' };  
 }  
 }  
 }  
 }  
  
 return dp;  
}  
  
void find\_all\_paths(const vector<vector<Cell>>& dp, const string& A, const string& B,  
 int i, int j, string current, vector<string>& all\_paths,  
 int cost\_replace, int cost\_insert, int cost\_delete) {  
 if (i == 0 && j == 0) {  
 reverse(current.begin(), current.end());  
 all\_paths.push\_back(current);  
 return;  
 }  
  
 if (i > 0 && j > 0) {  
 if (A[i - 1] == B[j - 1] && dp[i][j].cost == dp[i - 1][j - 1].cost) {  
 find\_all\_paths(dp, A, B, i - 1, j - 1, current + 'M', all\_paths, cost\_replace, cost\_insert, cost\_delete);  
 }  
 if (A[i - 1] != B[j - 1] && dp[i][j].cost == dp[i - 1][j - 1].cost + cost\_replace) {  
 find\_all\_paths(dp, A, B, i - 1, j - 1, current + 'R', all\_paths, cost\_replace, cost\_insert, cost\_delete);  
 }  
 }  
 if (j > 0 && dp[i][j].cost == dp[i][j - 1].cost + cost\_insert) {  
 find\_all\_paths(dp, A, B, i, j - 1, current + 'I', all\_paths, cost\_replace, cost\_insert, cost\_delete);  
 }  
 if (i > 0 && dp[i][j].cost == dp[i - 1][j].cost + cost\_delete) {  
 find\_all\_paths(dp, A, B, i - 1, j, current + 'D', all\_paths, cost\_replace, cost\_insert, cost\_delete);  
 }  
}  
  
void apply\_operations(const string& A, const string& B, const string& operations) {  
 string result = A;  
 int i = 0, j = 0;   
  
   
 cout << result << endl;  
  
 for (char op : operations) {  
 if (op == 'M') {  
 ++i;  
 ++j;  
 }  
 else if (op == 'R') {  
 result[i] = B[j];  
 ++i;  
 ++j;  
 cout << result << endl;  
 }  
 else if (op == 'I') {  
 result.insert(result.begin() + i, B[j]);  
 ++i;  
 ++j;  
 cout << result << endl;  
 }  
 else if (op == 'D') {  
 result.erase(result.begin() + i);  
 cout << result << endl;  
 }  
 }  
}  
  
int main() {  
 setlocale(LC\_ALL, "Russian");  
 int cost\_replace, cost\_insert, cost\_delete;  
 string A, B;  
  
 cin >> cost\_replace >> cost\_insert >> cost\_delete;  
 cin >> A >> B;  
  
 auto dp = build\_dp(A, B, cost\_replace, cost\_insert, cost\_delete);  
  
 vector<string> all\_paths;  
 find\_all\_paths(dp, A, B, A.length(), B.length(), "", all\_paths, cost\_replace, cost\_insert, cost\_delete);  
  
 cout << "Все возможные изменения:" << endl;  
 for (const string& path : all\_paths)  
 cout << path << endl;  
  
 if (!all\_paths.empty()) {  
 cout << "Пример редакционного предписания:" << endl;  
 apply\_operations(A, B, all\_paths[0]);  
 }  
  
 return 0;  
}