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. 3343 _____ Бондаренко Ф. А.

Преподаватель _____ Жангиров Т. Р.

Санкт-Петербург

2025

Цель работы.

Изучить и реализовать алгоритм Вагнера-Фишера для нахождения редакционного предписания.

Задание.

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.

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

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.

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

Индивидуальный вариант: 13.

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.

Выполнение работы.

Описание реализованных функций для алгоритма Вагнера-Фишера:

- *matrix_t getMatrix(const std::string& s1, const std::string& s2, const Cost& cost, bool record)* — функция для получения матрицы со стоимостью редакционных предписаний для всевозможных срезов двух строк. Редакционное расстояние для заданных двух строк *s1* и *s2* находятся в правом нижнем углу матрицы.

- *std::string getPrescription(const matrix_t& matrix, const std::string& s1, const std::string& s2, const Cost& cost, bool record)* — функция для получения редакционного предписания (превращения строки s1 в s2 самым "дешевым" образом). Редакционное предписание строится на основе матрицы расстояний: происходит обход строк s1 и s2 в обратном порядке, и сравниваются 2 последних символа: если они равны, то сдвигаем указатели в строках на 1 назад, иначе: смотрим на стоимость операций в матрице и выбираем подходящую.

- *void getAllPrescriprions(const matrix_t& matrix, const std::string& s1, const std::string& s2, const Cost& cost, std::vector<std::string>& prescriptions, std::string& currentPrescription, std::pair<int, int> pos, bool record)* — функция для получения всех редакционных предписаний: работает на основе алгоритма *backtracking* и алгоритма функции *getPrescription*: перед каждой

- *void show(const std::string& s1, const std::string& s2, const std::string& prescription)* — функция для отображения преобразования строки s1 в s2 по заданному редакционному предписанию.

- *matrix_t initMatrix(int n, int m, const Cost& cost)* — функция для инициализации начальной матрицы: первая строка — стоимости операций по превращению любой подстроки s1 в пустую строку; первый столбец — стоимости превращений пустой строки s1 в любую подстроку s2.

- *void printMatrix(const matrix_t& matrix)* — функция для отображения в терминале итоговой матрицы редакционных расстояний.

Описание рекуррентной функции Левенштейна:

D(a, b) - расстояние Левенштейна для двух строк a и b:

1. $D(a,b)=|a|$, если |b| == 0
2. $D(a,b)=|b|$, если |a| == 0
3. $D(a,b)=D(tail(a),tail(b))$, если a[0] == b[0]
4. $D(a,b)=1+min(D(a[1:],b);D(a,b[1:]);D(a[1:],b[1:]))$ в общем случае

Где:

- |a| — длина строки a;
- |b| — длина строки b.

Т.е. по рекуррентной формуле Левенштейна необходимо:

- либо рекурсивно "пройти" одну из строк до нулевого размера (воспользоваться пунктами (1) или (2) из формулы);
- либо рекурсивно "пройти" обе строки до нулевого размера (если они одинаковые) (т. е. воспользоваться пунктом (3) формулы).

Следовательно, будут происходить лишние итерации: если для данных подстрок уже вычислялось расстояние Левенштейна, то его надо будет заново получить.

Описание алгоритма Вагнера-Фишера:

Рекуррентная формула:

D[i, j] — элемент матрицы (превращаем s1[:i+1] в s2[:j+1] (т. е. превращаем срезы)):

1. $D[i,j]=0$, если i == 0 и j == 0
2. $D[i,j]=i\cdot cost_{delete}$, если j == 0 и i > 0
3. $D[i,j]=j\cdot cost_{insert}$, если i == 0 и j > 0
4. $D[i,j]=min(D[i,j-1]+cost_{insert}, D[i-1,j]+cost_{delete}, D[i-1,j-1]+cost_{replace})$ если i > 0 и j > 0

Где:

- Индексация строк в матрице начинается с единицы;
- D[i, j] = D(s1[:i+1], s2[:j+1]);
- D(s1, s2) = D[N, M].

Т.е. вместо возможного повторного вычисления расстояния Левенштейна для двух подстрок s1 и s2, значение расстояния берется из матрицы $n\times m$ (где n — длина s1, m — длина s2), куда оно было сохранено заранее (при первом вычислении). Сами редакционные расстояния вычисляются почти по той же формуле, что и рекуррентная формула Левенштейна.

Оценка сложности алгоритма.

- По времени: $O(n\cdot m)$, где n и m — длины строк: необходимо вычисление матрицы размера $n\times m$.
- По памяти: $O(n\cdot m)$, где n и m — длины строк: необходимо обеспечить хранение матрицы размера $n\times m$.

Тестирование.

Входные данные	Выходные данные	Комментарий
1 1 1 Helloworld worldHello	8	Успех (для редакционного расстояния)
1 1 1 Helloworld worldHello	RRRMRRRRMR Helloworld worldHello	Успех (для редакционного предписания)
1 1 1 aaaa aaaa	0	Успех (для редакционного расстояния)
1 1 1 aaaa aaaa	MMMM aaaa aaaa	Успех (для редакционного предписания)
2 1 3 Helloworld! worldHello!	RRRMRRRRMRM Helloworld! worldHello!	Успех (для редакционного предписания)
2 1 1 Helloworld! worldHello!	IIIIIMMMMDDMDDDM IIIIIMMMMMDDDDDM DDDDDMMMMMIIIIIM	Успех (для всех предписаний)

Вывод.

Изучен алгоритм Вагнера-Фишера для нахождения редакционного предписания для двух строк. Написана программа на языке C++ для нахождения редакционного предписания (всех редакционных предписаний) для заданных строк с возможностью вывода промежуточных результатов работы алгоритма.

ПРИЛОЖЕНИЕ А

Файл *main.cpp*:

```
#include <functional>
#include <iostream>
#include <map>
#include <string>

#include "../include/Cost.hpp"
#include "../include/Levenstein.hpp"

void runMode(const Cost& cost, const std::string& s1, const
std::string& s2) {
   levenstein::matrix_t matrix = levenstein::getMatrix(s1, s2, cost,
false);

  std::cout << std::endl;
  std::cout << "-------------------------" << std::endl;
  std::cout << matrix.back().back() << std::endl;
  std::cout << "-------------------------" << std::endl;
  std::cout << std::endl;

  std::string prescription =
      levenstein::getPrescription(matrix, s1, s2, cost, false);

  std::cout << std::endl;
  std::cout << "-------------------------" << std::endl;
  std::cout << prescription << std::endl;
  std::cout << s1 << std::endl;
  std::cout << s2 << std::endl;
  std::cout << "-------------------------" << std::endl;
  std::cout << std::endl;
}

void stepMode(const Cost& cost, const std::string& s1, const
std::string& s2) {
   levenstein::matrix_t matrix = levenstein::getMatrix(s1, s2, cost,
true);

  std::cout << std::endl;
  std::cout << "-------------------------" << std::endl;
  std::cout << "Редакционное расстояние: [" << matrix.back().back() <<
"]"
            << std::endl;
  std::cout << "-------------------------" << std::endl;
  std::cout << std::endl;

  std::string prescription =
      levenstein::getPrescription(matrix, s1, s2, cost, true);

  std::cout << std::endl;
  std::cout << "-------------------------" << std::endl;
  std::cout << "Редакционное предписание: [" << prescription << "]"
            << std::endl;
  levenstein::show(s1, s2, prescription);
  std::cout << "-------------------------" << std::endl;
  std::cout << std::endl;
}
```

```
void prescriptionsRunMode(const Cost& cost, const std::string& s1,
                          const std::string& s2) {
   levenstein::matrix_t matrix = levenstein::getMatrix(s1, s2, cost,
false);

  std::vector<std::string> prescriptions;
  std::string current = "";
  levenstein::getAllPrescriprions(matrix, s1, s2, cost, prescriptions,
current,
                                  {s1.length(), s2.length()}, false);

  for (size_t i = 0; i < prescriptions.size(); i++) {
    std::cout << std::endl;
    std::cout << "-------------------------" << std::endl;
        std::cout << "Предписание номер [" << i << "]: " <<
prescriptions[i]
              << std::endl;
    std::cout << "-------------------------" << std::endl;
  }
}

void prescriptionsStepMode(const Cost& cost, const std::string& s1,
                           const std::string& s2) {
   levenstein::matrix_t matrix = levenstein::getMatrix(s1, s2, cost,
true);

  std::vector<std::string> prescriptions;
  std::string current = "";
  levenstein::getAllPrescriprions(matrix, s1, s2, cost, prescriptions,
current,
                                  {s1.length(), s2.length()}, true);

  for (size_t i = 0; i < prescriptions.size(); i++) {
    std::cout << std::endl;
    std::cout << "-------------------------" << std::endl;
    std::cout << "Предписание номер [" << i << "]:" << std::endl;
    levenstein::show(s1, s2, prescriptions[i]);
    std::cout << "-------------------------" << std::endl;
  }
}

int main(int argc, char* argv[]) {
  if (argc < 2) {
    return 1;
  }

  std::string mode = argv[1];

  Cost cost;
  std::string s1, s2;

  std::cin >> cost.replaceCost >> cost.insertCost >> cost.deleteCost;
  std::cin >> s1 >> s2;

      std::map<std::string,   std::function<void(const   Cost&,   const
std::string&,
                                          const std::string&)>>
      modes = {{"run", runMode},
```

```
                {"step", stepMode},
                {"prescriptions-no-step", prescriptionsRunMode},
                {"prescriptions-with-step", prescriptionsStepMode}};

  auto it = modes.find(mode);
  if (it != modes.end()) {
    it->second(cost, s1, s2);
  }

  return 0;
}
```

Файл *Cost.hpp*:

```
#ifndef COST_HPP_
#define COST_HPP_

struct Cost {
  int insertCost;
  int deleteCost;
  int replaceCost;

  Cost() : insertCost(1), deleteCost(1), replaceCost(1) {}
  Cost(int insertCost, int deleteCost, int replaceCost)
      : insertCost(insertCost),
        deleteCost(deleteCost),
        replaceCost(replaceCost) {}
};

#endif  // COST_HPP_
```

Файл *Levenstein.hpp*:

```
#ifndef LEVENSTEIN_HPP_
#define LEVENSTEIN_HPP_

#include <string>
#include <utility>
#include <vector>

#include "Cost.hpp"

namespace levenstein {

using matrix_t = std::vector<std::vector<int>>;

matrix_t getMatrix(const std::string& s1, const std::string& s2,
                   const Cost& cost, bool record);

std::string getPrescription(const matrix_t& matrix, const std::string&
s1,
                            const std::string& s2, const Cost& cost,
                            bool record);

void getAllPrescriprions(const matrix_t& matrix_t, const std::string&
s1,
                         const std::string& s2, const Cost& cost,
                         std::vector<std::string>& prescriptions,
                         std::string& currentPresrciption,
                         std::pair<int, int> pos, bool record);
```

```
void show(const std::string& s1, const std::string& s2, const
std::string& prescription);

}  // namespace levenstein

#endif  // LEVENSTEIN_HPP_
```

Файл *Levenstein.cpp*:

```
#include "../include/Levenstein.hpp"

#include <algorithm>
#include <iomanip>
#include <iostream>

namespace levenstein {

namespace details {

const std::string red = "\033[31m";
const std::string reset = "\033[0m";

matrix_t initMatrix(int n, int m, const Cost& cost) {
  matrix_t matrix(n + 1, std::vector<int>(m + 1, 0));

  for (int i = 1; i < n + 1; i++) {
    matrix[i][0] = matrix[i - 1][0] + cost.deleteCost;
  }

  for (int j = 1; j < m + 1; j++) {
    matrix[0][j] = matrix[0][j - 1] + cost.insertCost;
  }

  return matrix;
}

void printMatrix(const matrix_t& matrix) {
  if (matrix.empty()) return;

  size_t rows = matrix.size();
  size_t cols = matrix[0].size();

  int maxVal = 0;
  for (auto& row : matrix) {
    for (int x : row) {
      maxVal = std::max(maxVal, x);
    }
  }

  size_t numberWidth = std::to_string(maxVal).size();
  size_t columnWidth = std::to_string(cols - 1).size();
  size_t rowWidth = std::to_string(rows - 1).size();

  size_t width = std::max({numberWidth, columnWidth, rowWidth}) + 1;

  std::cout << std::setw(width) << ' ';
  for (size_t j = 0; j < cols; j++) {
    std::cout << std::setw(width) << j;
  }
  std::cout << std::endl;
```

```
  for (size_t i = 0; i < rows; i++) {
    std::cout << std::setw(width) << i;
    for (size_t j = 0; j < cols; j++) {
      std::cout << std::setw(width) << matrix[i][j];
    }
    std::cout << std::endl;
  }
}

}  // namespace details

matrix_t getMatrix(const std::string& s1, const std::string& s2,
                   const Cost& cost, bool record) {
  int n = s1.length();
  int m = s2.length();

  if (record) {
    std::cout << std::endl;
    std::cout << "==========================" << std::endl;

    std::cout
              << "Алгоритм Фишера-Вагнера для нахождения расстояния
Левенштейна:"
        << std::endl;
    std::cout << "\t1) Инициализация матрицы:" << std::endl;
     std::cout << "\t\t- Первая строка - стоимость превращений любой
подстроки "
                 "s1 в пустую"
              << std::endl;
     std::cout << "\t\t- Первый столбец - стоимость превращений пустой
строки "
                 "s1 в любую подстроку s2"
              << std::endl;

    std::cout << "\t2) Заполнение матрицы по правилу:" << std::endl;
     std::cout << "\t\t- s1[i] != s2[j]: matrix[i][j]= min(matrix[i -
1][j] + "
                 "deleteCost, matrix[i][j - 1] + insertCost, "
                 "matrix[i - 1][j - 1] + replaceCost)"
              << std::endl;
     std::cout << "\t\t- s1[i] == s2[j]: matrix[i][j] = matrix[i-1][j-
1]"
              << std::endl;

    std::cout << "\t3) D(s1, s2) = matrix[n+1][m+1]" << std::endl;

    std::cout << "==========================" << std::endl;
    std::cout << std::endl;
  }

  matrix_t matrix = details::initMatrix(n, m, cost);

  for (int i = 1; i < n + 1; i++) {
    for (int j = 1; j < m + 1; j++) {
      if (s1[i - 1] != s2[j - 1]) {
        if (record) {
          std::cout << std::endl;
          std::cout << "-------------------------" << std::endl;
```

```
          std::cout << "s1[i-1] != s2[j-1]: " << std::endl;
                std::cout << "[s1]: [" << s1.substr(0, i - 1) <<
details::red
                     << s1[i - 1] << details::reset << s1.substr(i) <<
"]"
                   << std::endl;
                std::cout << "[s2]: [" << s2.substr(0, j - 1) <<
details::red
                     << s2[j - 1] << details::reset << s2.substr(j) <<
"]"
                   << std::endl;

          std::cout << "matrix[i][j] = min("
                    << matrix[i - 1][j] + cost.deleteCost << ", "
                    << matrix[i][j - 1] + cost.insertCost << ", "
                    << matrix[i - 1][j - 1] + cost.replaceCost << ")"
                    << std::endl;

          std::cout << "-------------------------" << std::endl;
          std::cout << std::endl;
        }

        matrix[i][j] = std::min({matrix[i - 1][j] + cost.deleteCost,
                                 matrix[i][j - 1] + cost.insertCost,
                                                    matrix[i - 1][j - 1] +
cost.replaceCost});
      } else {
        if (record) {
          std::cout << std::endl;
          std::cout << "-------------------------" << std::endl;

          std::cout << "s1[i-1] == s2[j-1]: " << std::endl;
                std::cout << "[s1]: [" << s1.substr(0, i - 1) <<
details::red
                     << s1[i - 1] << details::reset << s1.substr(i) <<
"]"
                   << std::endl;
                std::cout << "[s2]: [" << s2.substr(0, j - 1) <<
details::red
                     << s2[j - 1] << details::reset << s2.substr(j) <<
"]"
                   << std::endl;
             std::cout << "matrix[i][j] = " << matrix[i - 1][j - 1] <<
std::endl;

          std::cout << "-------------------------" << std::endl;
          std::cout << std::endl;
        }

        matrix[i][j] = matrix[i - 1][j - 1];
      }
    }
  }

  if (record) {
    std::cout << std::endl;
    std::cout << "=========================" << std::endl;
```

```
    std::cout << "Итоговая матрица: " << std::endl;
    details::printMatrix(matrix);

    std::cout << "==========================" << std::endl;
    std::cout << std::endl;
  }

  return matrix;
}

std::string getPrescription(const matrix_t& matrix, const std::string&
s1,
                            const std::string& s2, const Cost& cost,
                            bool record) {
  int i = s1.length();
  int j = s2.length();
  std::string prescription;

  if (record) {
    std::cout << std::endl;
    std::cout << "==========================" << std::endl;

    std::cout << "Алгоритм для нахождения редакционного предписания:"
              << std::endl;
    std::cout << "\t1) Идем с конца двух строк" << std::endl;
     std::cout << "\t2) Работаем по матрице Фишера-Вагнера в обратном
порядке"
              << std::endl;

    std::cout << "==========================" << std::endl;
    std::cout << std::endl;
  }

  while (i > 0 || j > 0) {
    if (i > 0 && j > 0 && s1[i - 1] == s2[j - 1]) {
      if (record) {
        std::cout << std::endl;
        std::cout << "--------------------------" << std::endl;

        std::cout << "Символы совпадают: " << s1[i - 1] << std::endl;
         std::cout << "i = [" << i << "]" << ", j = [" << j << "]" <<
std::endl;
        std::cout << "Действие: M" << std::endl;

        std::cout << "--------------------------" << std::endl;
        std::cout << std::endl;
      }

      prescription.push_back('M');
      i--;
      j--;
    } else if (j > 0 &&
                   (i == 0 || matrix[i][j - 1] + cost.insertCost ==
matrix[i][j])) {
      if (record) {
        std::cout << std::endl;
        std::cout << "--------------------------" << std::endl;
```

```
          std::cout << "Необходимо вставить символ: " << s2[j - 1] <<
std::endl;
          std::cout << "i = [" << i << "]" << ", j = [" << j << "]" <<
std::endl;
        std::cout << "Действие: I" << std::endl;

        std::cout << "-------------------------" << std::endl;
        std::cout << std::endl;
      }

      prescription.push_back('I');
      j--;
    } else if (i > 0 &&
                    (j == 0 || matrix[i - 1][j] + cost.deleteCost ==
matrix[i][j])) {
      if (record) {
        std::cout << std::endl;
        std::cout << "-------------------------" << std::endl;

          std::cout << "Необходимо удалить символ: " << s1[i - 1] <<
std::endl;
          std::cout << "i = [" << i << "]" << ", j = [" << j << "]" <<
std::endl;
        std::cout << "Действие: D" << std::endl;

        std::cout << "-------------------------" << std::endl;
        std::cout << std::endl;
      }

      prescription.push_back('D');
      i--;
    } else {
      if (record) {
        std::cout << std::endl;
        std::cout << "-------------------------" << std::endl;

          std::cout << "Необходимо заменить символ " << s1[i - 1] << "
на "
                    << s2[j - 1] << std::endl;
          std::cout << "i = [" << i << "]" << ", j = [" << j << "]" <<
std::endl;
        std::cout << "Действие: R" << std::endl;

        std::cout << "-------------------------" << std::endl;
        std::cout << std::endl;
      }

      prescription.push_back('R');
      i--;
      j--;
    }
  }

  std::reverse(prescription.begin(), prescription.end());

  if (record) {
    std::cout << std::endl;
    std::cout << "=========================" << std::endl;
```

```
        std::cout << "Итоговое предписание: " << prescription <<
std::endl;

    std::cout << "==========================" << std::endl;
    std::cout << std::endl;
  }

  return prescription;
}

void getAllPrescriprions(const matrix_t& matrix, const std::string&
s1,
                         const std::string& s2, const Cost& cost,
                         std::vector<std::string>& prescriptions,
                         std::string& currentPrescription,
                         std::pair<int, int> pos, bool record) {
  auto [i, j] = pos;

  if (i == 0 && j == 0) {
    std::string p = currentPrescription;
    std::reverse(p.begin(), p.end());

    if (record) {
      std::cout << std::endl;
      std::cout << "==========================" << std::endl;

      std::cout << "Найдено предписание: " << p << std::endl;

      std::cout << "==========================" << std::endl;
      std::cout << std::endl;
    }

    prescriptions.push_back(p);
    return;
  }

  if (i > 0 && j > 0 && s1[i - 1] == s2[j - 1]) {
    currentPrescription.push_back('M');

    if (record) {
      std::cout << std::endl;
      std::cout << "--------------------------" << std::endl;

      std::cout << "Символы совпадают: " << s1[i - 1] << std::endl;
       std::cout << "i = [" << i << "]" << ", j = [" << j << "]" <<
std::endl;
      std::cout << "Действие: M" << std::endl;
      std::cout << "Текущее предписание: "
                << currentPrescription.substr(0,
                                              currentPrescription.leng
th() - 1)
                    << details::red << currentPrescription.back() <<
details::reset
                << std::endl;

      std::cout << "--------------------------" << std::endl;
      std::cout << std::endl;
    }
```

```
    getAllPrescriprions(matrix, s1, s2, cost, prescriptions,
                        currentPrescription, {i - 1, j - 1}, record);
    currentPrescription.pop_back();
  }

   if (i > 0 && (j == 0 || matrix[i][j] == matrix[i - 1][j] +
cost.deleteCost)) {
    currentPrescription.push_back('D');

    if (record) {
      std::cout << std::endl;
      std::cout << "--------------------------" << std::endl;

        std::cout << "Необходимо удалить символ: " << s1[i - 1] <<
std::endl;
       std::cout << "i = [" << i << "]" << ", j = [" << j << "]" <<
std::endl;
      std::cout << "Действие: D" << std::endl;
      std::cout << "Текущее предписание: "
                << currentPrescription.substr(0,
                                              currentPrescription.leng
th() - 1)
                    << details::red << currentPrescription.back() <<
details::reset
                << std::endl;

      std::cout << "--------------------------" << std::endl;
      std::cout << std::endl;
    }

    getAllPrescriprions(matrix, s1, s2, cost, prescriptions,
                        currentPrescription, {i - 1, j}, record);
    currentPrescription.pop_back();
  }

   if (j > 0 && (i == 0 || matrix[i][j] == matrix[i][j - 1] +
cost.insertCost)) {
    currentPrescription.push_back('I');

    if (record) {
      std::cout << std::endl;
      std::cout << "--------------------------" << std::endl;

        std::cout << "Необходимо вставить символ: " << s2[j - 1] <<
std::endl;
       std::cout << "i = [" << i << "]" << ", j = [" << j << "]" <<
std::endl;
      std::cout << "Действие: I" << std::endl;
      std::cout << "Текущее предписание: "
                << currentPrescription.substr(0,
                                              currentPrescription.leng
th() - 1)
                    << details::red << currentPrescription.back() <<
details::reset
                << std::endl;

      std::cout << "--------------------------" << std::endl;
      std::cout << std::endl;
    }
```

```
    getAllPrescriprions(matrix, s1, s2, cost, prescriptions,
                        currentPrescription, {i, j - 1}, record);
    currentPrescription.pop_back();
  }

  if (i > 0 && j > 0 &&
      (matrix[i][j] == matrix[i - 1][j - 1] + cost.replaceCost)) {
    currentPrescription.push_back('R');

    if (record) {
      std::cout << std::endl;
      std::cout << "-------------------------" << std::endl;

       std::cout << "Необходимо заменить символ " << s1[i - 1] << " на
"
                 << s2[j - 1] << std::endl;
        std::cout << "i = [" << i << "]" << ", j = [" << j << "]" <<
std::endl;
      std::cout << "Действие: R" << std::endl;
      std::cout << "Текущее предписание: "
                << currentPrescription.substr(0,
                                              currentPrescription.leng
th() - 1)
                    << details::red << currentPrescription.back() <<
details::reset
                << std::endl;

      std::cout << "-------------------------" << std::endl;
      std::cout << std::endl;
    }

    getAllPrescriprions(matrix, s1, s2, cost, prescriptions,
                        currentPrescription, {i - 1, j - 1}, record);
    currentPrescription.pop_back();
  }
}

void show(const std::string& s1, const std::string& s2,
          const std::string& prescription) {
  std::string current = s1;
  int s1Pos = 0;
  int s2Pos = 0;

  std::cout << "Исходная строка: " << current << std::endl;

  for (const char& operation : prescription) {
    switch (operation) {
      case 'M':
        std::cout << "M: Совпадение " << current[s1Pos++] << " --> "
                  << s2[s2Pos++] << std::endl;
        break;

      case 'D':
        std::cout << "D: Удаляем " << current[s1Pos] << std::endl;
        current.erase(s1Pos, 1);
        break;

      case 'I':
```

```
        std::cout << "I: Вставляем " << s2[s2Pos] << std::endl;
        current.insert(s1Pos++, 1, s2[s2Pos++]);
        break;

      case 'R':
         std::cout << "R: Заменяем " << current[s1Pos] << " --> " <<
s2[s2Pos]
                   << std::endl;
        current[s1Pos++] = s2[s2Pos++];
        break;

      default:
            std::cout << "Неизвестная операция: " << operation <<
std::endl;
        break;
    }

    std::cout << "Текущая строка: " << current << std::endl;
  }

  std::cout << "Результат: " << current << std::endl;
}

}  // namespace levenstein
```

Файл *Makefile*:

```
CXX=g++
CXXFLAGS=-std=c++20 -Wall -Wextra -pedantic
EXT=cpp

SRC_DIR=src
OBJ_DIR=obj

SRC_FILES=$(wildcard $(SRC_DIR)/*.$(EXT))
OBJ_FILES=$(patsubst $(SRC_DIR)/%.$(EXT),$(OBJ_DIR)/%.o,$(SRC_FILES))

EXEC=$(OBJ_DIR)/exec
INCLUDES=-Iinclude

$(shell mkdir -p $(OBJ_DIR))

all: $(EXEC)

$(EXEC) : $(OBJ_FILES)
     $(CXX) $(CXXFLAGS) $(INCLUDES) -o $@ $^

$(OBJ_DIR)/%.o : $(SRC_DIR)/%.$(EXT)
     $(CXX) $(CXXFLAGS) $(INCLUDES) -c $< -o $@

run: $(EXEC)
     ./$(EXEC) run

step: $(EXEC)
     ./$(EXEC) step

prescriptions-no-step: $(EXEC)
     ./$(EXEC) prescriptions-no-step

prescriptions-with-step: $(EXEC)
```

```
	./$(EXEC) prescriptions-with-step

clean:
	rm -rf $(OBJ_DIR)/*.o $(EXEC)

.PHONY: all run step prescriptions-no-step prescriptions-with-step
clean
```