**МИНОБРНАУКИ РОССИИ**

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**

**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**

**Кафедра МО ЭВМ**

**ОТЧЁТ**

**по лабораторной работе №4**

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

**Тема: Динамическое программирование.**

|  |  |  |
| --- | --- | --- |
| Студентка гр. 5382 |  | Лисс Н.И. |
| Преподаватель |  | Шолохова О.М. |

Санкт-Петербург

2017

**Цель работы.**

Изучение возможностей и использование динамического программирования.

**Постановка задачи.**

**«*Редакционное расстояние*».** Даны две строки *A* и *B*. Со строкой *A* разрешено выполнять следующие операции:

* удалить символ;
* вставить символ;
* заменить один символ на другой.

Какое наименьшее количество операций необходимо применить к строке *A*, чтобы получить строку *B*?

*Примечание*. Это количество операций принято называть *редакционным расстоянием.*

**Основные теоретические положения.**

**Динамическое программирование**  — способ решения сложных задач путём разбиения их на более простые подзадачи. Подход динамического программирования состоит в том, чтобы решить каждую подзадачу только один раз, сократив тем самым количество вычислений.

Редакционное расстояние, или расстояние Левенштейна — метрика, позволяющая определить «схожесть» двух строк — минимальное количество операций вставки одного символа, удаления одного символа и замены одного символа на другой, необходимых для превращения одной строки в другую.

**Спецификация программы.**

Программа ищет минимальное количество операций, применяемых к первой строке, чтобы получить вторую.

Программа написана на языке С++.

Входные данные:

Последовательность <a1…an>, где ai – символ. – первая строка.

Последовательность <b1…bn>, где bi – символ. – вторая строка.

Выходные данные: Редакционное расстояние.

**Реализация программы.**

Будем считать, что каждая из этих операций обладает определенной стоимостью. Ставится задача нахождения последовательности операций минимальной суммарной стоимости, в результате выполнения которой строка s1 преобразуется в строку s2. Для простоты мы будем рассматривать случай, когда все операции имеют одинаковую стоимость, равную единице; в таком случае стоимость преобразования строки s1 в строку s2 равна просто количеству совершенных операций. Искомое минимальное количество операций называется ***редакционным расстоянием*** между строками.

Пусть d(i,j) – это редакционное расстояние между подстроками s1[i…l-1] и s2[j…l2-1] , 0<=i<=l1, 0<=j<=l2 (случаи, когда индекс равен длине соответствующей строки, означают, что рассматривается пустой суффикс этой строки). Базой динамического программирования является состояние (l1,l2), для которого значение d равно нулю (как редакционное расстояние между пустыми строками). Переходы от данной подзадачи к меньшим основаны на рассмотрении следующих случаев:

1) Если i<l1 и j<l2, то можно заменить символ s1[i] символом s2[j] и исключить эти два символа из рассмотрения (редакционное расстояние будет равно d(i+1,j+1)+1).

2) Если i<l1, j<l2 и при этом дополнительно s1[i]=s2[j], то выполнять замену необходимости нет (редакционное расстояние будет равно d(i+1,j+1).

3) Если i<l1, то можно удалить символ s1[i] (редакционное расстояние будет равно d(i+1,j+1)).

4) Если j<l2, то можно добавить в начало строки s1[i…l1-1] символ s2[j] и исключить эти символы из рассмотрения (редакционное расстояние будет равно d(i,j+1)+1.

Из этих случаев нужно выбрать тот, который позволяет минимизировать получаемое редакционное расстояние.

Искомое редакционное расстояние между строками s1 и s2 будет содержаться в d(0,0). Для того чтобы восстановить само редакционное предписание, можно для каждого состояния (i,j) дополнительно сохранять величину p(i,j) – код операции (вставка, удаление, замена), которая позволяет минимизировать d(i,j), и после построения этих массивов «обратным ходом» найти искомую оптимальную последовательность.

**Тестирование.**

|  |  |
| --- | --- |
| Входные данные | Выходные данные |
| Строка 1: 24234  Строка 2: 123213 | Минимум операций : 4 |
| Строка 1: 12342  Строка 2: 12q13 | Минимум операций : 3 |
| Строка 1: 3422213  Строка 2: 22 | Минимум операций : 5 |

**Код программы.**

// Lab4\_PiAA.cpp: определяет точку входа для консольного приложения.

//

#include "stdafx.h"

#include <iostream>

#include <string>

#include <algorithm>

#include <fstream>

#include <cmath>

using namespace std;

fstream infile("input.txt");

const int INF = 1E9;

const int LEN = 1000;

int d[LEN + 1][LEN + 1];

int p[LEN + 1][LEN + 1];

void edit(int l1, int l2, string s1, string s2);

int main()

{

setlocale(LC\_ALL, "russian");

int count;

cout << "Выберите одно из предложенных действий:\nНажмите 1, чтобы ввести данные с клавиатуры.\nНажмите 2, чтобы прочитать из файла "<< endl;

cin >> count;

switch (count)

{

case 1: {

string s1, s2;

cout << "Введите перую строку:\n";

cin >> s1;

cout << "Введите вторую строку:\n";

cin >> s2;

int l1 = s1.length();

int l2 = s2.length();

d[l1][l2] = 0;

edit(l1, l2, s1, s2);

break;

}

case 2: {

string s1, s2;

infile >> s1;

infile >> s2;

int l1 = s1.length();

int l2 = s2.length();

d[l1][l2] = 0;

edit(l1, l2, s1, s2);

//d[l1][l2] = 0;

break;

}

}

system("pause");

return 0;

}

void edit(int l1, int l2, string s1, string s2)

{

for (int i = l1; i >= 0; --i)

for (int j = l2; j >= 0; --j)

{

if (i == l1 && j == l2)

continue;

d[i][j] = INF;

if (i < l1 && j < l2 && s1[i] == s2[j])

{

cout << "Равные элементы! " << s1[i] << " и " << s2[j] << endl;

d[i][j] = d[i + 1][j + 1];

p[i][j] = 3;

}

if (i + 1 <= l1)

{

if (d[i + 1][j] + 1 < d[i][j])

{

d[i][j] = d[i + 1][j] + 1;

p[i][j] = 0;

cout << "Различные элементы! " << d[i][j] << " Первый : " << s1[i] << " Второй : " << s2[j] << endl;

}

}

if (i + 1 <= l1 && j + 1 <= l2)

{

if (d[i + 1][j + 1] + 1 < d[i][j])

{

d[i][j] = d[i + 1][j + 1] + 1;

p[i][j] = 1;

}

}

if (j + 1 <= l2)

{

if (d[i][j + 1] + 1 < d[i][j])

{

d[i][j] = d[i][j + 1] + 1;

p[i][j] = 2;

}

}

}

cout << d[0][0] << endl;

int curi = 0, curj = 0;

int ch = 0;

while (curi < l1 || curj < l2)

{

switch (p[curi][curj])

{

case 0:

{

cout << "DELETE " << curi + ch << endl;

++curi;

--ch;

break;

}

case 1:

{

cout << "REPLACE " << curi + ch << ' ' << s2[curj] << endl;

++curi;

++curj;

break;

}

case 2:

{

cout << "INSERT " << curi + ch << ' ' << s2[curj] << endl;

++curj;

++ch;

break;

}

case 3:

{

++curi;

++curj;

}

}

}

}

**Вывод.**

В ходе данной лабораторной работы были освоены принципы динамического программирования и решена задача редакционного расстояния.