Белорусский государственный технологический университет

Факультет информационных технологий

Кафедра информационных систем и технологий

Отчёт по лабораторной работе №4

«Динамическое программирование»

по дисциплине

«Математическое программирование»

Выполнила:

студентка 2-го курса спец. ПОИТ

Коржова В. С.

Вариант №3

Проверил:

ассистент кафедры

Барковский Е.В.

**Лабораторная работа 4. Динамическое программирование**

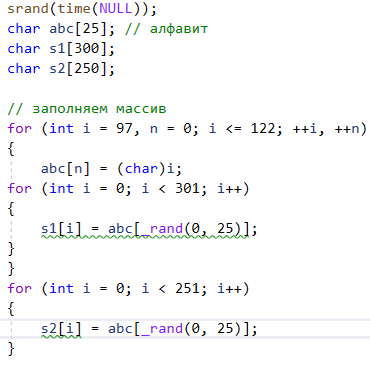
**Цель работы:** освоить общие принципы решения задач методом динамического программирования, сравнить полученные решения задач с рекурсивным методом.

**Задание для выполнения:**

**Задание 1.**

На языке С++ сгенерировать случайным образом строку букв латинского алфавита S1 длиной 300 символов и S2 длиной250.

**Решение:**

****

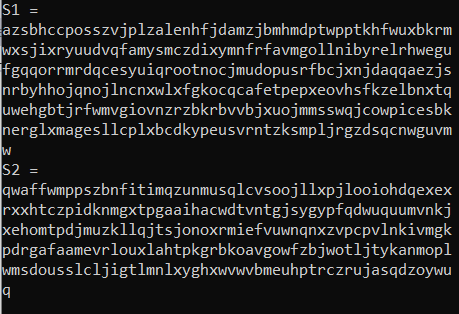


Рис. 1 – Пример генерации строк

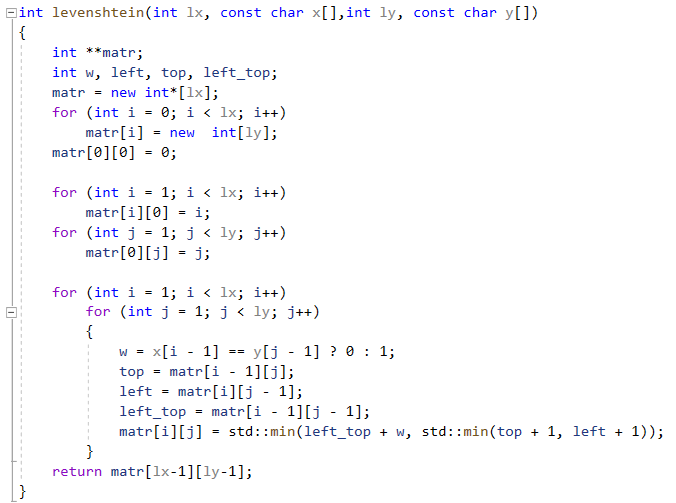
**Задание 2.**

Вычислить двумя способами (рекурсивно и с помощью динамического программирования)  – дистанцию Левенштейна для , где - длина строки ,  - строка состоящая из первых  символов строки . (копии экрана и код вставить в отчет).

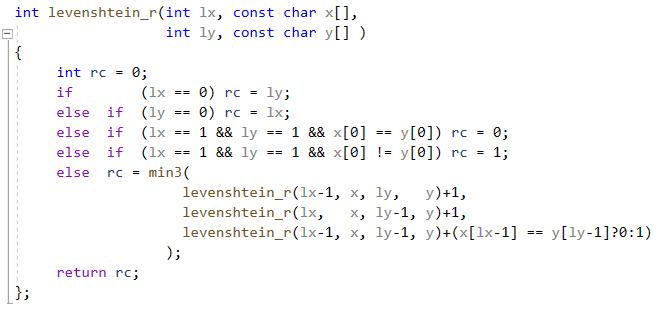
**Решение:**

Ниже приведены варианты реализации нахождения дистанции Левенштейна при помощи динамического программирования и при помощи рекурсивного алгоритма.

Исходный код реализации через динамическое программирование:



Пример реализации рекурсивным методом:

 На рисунке 2 представлены дистанции Левенштейна, вычисленные при помощи метода динамического программирования, а также рекурсивным алгоритмом.

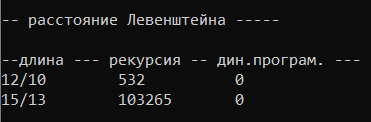


Рис. 2 – Проверка работоспособности решений

**Задание 3.**

Выполнить сравнительный анализ времени затраченного на вычисление дистанции Левенштейна для двух методов решения. Построить графики зависимости времени вычисления от k. (копии экрана и график вставить в отчет).

**Решение:**

Исходные данные для построения графика изображены на рисунке 3:

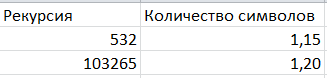


Рис. 3 – Исходные данные

Метод динамического программирования значительно эффективнее рекурсивного метода, т.к. выполняется намного быстрее.



Рис. 4 – График зависимости времени вычисления от К

Исходя из задания номер 2 и графика видно, что при больших значениях К, и, соответственно, при небольшой длине строк, метод динамического программирования является выигрышным вариантом по сравнению с методом рекурсии. Это происходит по той причине, что в методе ДП мы должны рассмотреть полиномиальное количество вариантов, пока не найдем решение, а в методе рекурсии перебор является экспоненциальным.

**Задание 4.**

Реализовать вручную пример вычисления дистанции Левенштейна при помощи рекурсивного алгоритма (в соответствии с вариантом) (каждый шаг алгоритма по примеру из лекции вставить в отчет).

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Вариант | Задание 4 | | Задание 5 | |
| 3 | Лом | Гомон | ABCDFGI | EATUFI |

**Решение:**

L(«ло», «гомон») + 1

1. L(“лом, гомон”) = min{ L(«лом», «гомо») + 1

L(«ло», «гомо»)

L(«л», «гомон») + 1

1. L(“ло, гомон”) = min{ L(«ло», «гомо») + 1

L(«л», «гомо») + 1

L(«ло», «гомо») + 1

1. L(“лом, гомо”) = min{ L(«лом», «гом») + 1

L(«ло», «гом») + 1

L(«л», «гомо») + 1

1. L(“ло, гомо”) = min{ L(«ло», «гом») + 1

L(«л», «гом») + 1

L(«», «гомон») + 1

1. L(“л, гомон”) = min{ L(«л», «гомо») + 1

L(«», «гомо») + 1

L(«», «гомон») = 5; L(«», «гомо») = 4;

L(«», «гомо») + 1

1. L(“л, гомо”) = min{ L(«л», «гом») + 1

L(«», «гом») + 1

L(«», «гомо») = 4; L(«», «гом») = 3;

L(«ло», «гом») + 1

1. L(“лом, гом”) = min{ L(«лом», «го») + 1

L(«ло», «го»)

L(«л», «гом») + 1

1. L(“ло, гом”) = min{ L(«ло», «го») + 1

L(«л», «го») + 1

L(«», «гом») + 1

1. L(“л, гом”) = min{ L(«л», «го») + 1

L(«», «го») + 1

L(«», «гом») = 3; L(«», «го») = 2;

L(«ло», «го») + 1

1. L(«лом», «го») = min{ L(«лом», «г») + 1

L(«ло», «г») + 1

L(«л», «го») + 1

1. L(«ло», «го») = min{ L(«ло», «г») + 1

L(«л», «г») + 1

L(«», «го») + 1

1. L(«л», «го») = min{ L(«л», «г») + 1

L(«», «г») + 1

L(«», «го») = 2; L(«», «г») = 1;

L(«ло», «г») + 1

1. L(«лом», «г») = min{ L(«лом», «») + 1

L(«ло», «») + 1

L(«лом», «») = 3; L(«ло», «») = 2;

L(«л», «г») + 1

1. L(«ло», «г») = min{ L(«ло», «») + 1

L(«л», «») + 1

L(«ло», «») = 2; L(«л», «») = 1;

L(«», «г») + 1

1. L(«л», «г») = min{ L(«л», «») + 1

L(«», «»)

L(«», «г») = 1; L(«л», «») = 1; L(«», «») = 0;

1. L(«л», «г») = min(2, 2, 0) = 0;
2. L(«ло», «г») = min(1, 3, 2) = 1;
3. L(«лом», «г») = min(2, 4, 3) = 2;
4. L(«л», «го») = min(3, 1, 2) = 1;
5. L(«ло», «го») = min(2, 2, 1) = 1;
6. L(«лом», «го») = min(2, 3, 2) = 2;
7. L(“л, гом”) = min(4, 2, 3) = 2;
8. L(“ло, гом”) = min(3, 2, 2) = 2;
9. L(“лом, гом”) = min(3, 3, 1) = 1;
10. L(“л, гомо”) = min(5, 3, 4) = 3;
11. L(“л, гомон”) = min(6, 4, 5) = 4;
12. L(“ло, гомо”) = min(4, 3, 3) = 3;
13. L(“лом, гомо”) = min(4, 2, 3) = 2;
14. L(“ло, гомон”) = min(5, 4, 4) = 4;
15. L(“лом, гомон”) = min(5, 3, 3) = 3;

Рекурсия

public static int LevensteinRecursion(string first, string second)

{

if (first == second)

{

return 0;

}

if (first == "")

{

return second.Length;

}

if (second == "")

{

return first.Length;

}

else

{

int firstLenght = first.Length, secondLenght = second.Length;

int[] result = new int[3];

result[0] = LevensteinRecursion(first.Substring(0, firstLenght - 1), second) + 1;

result[1] = LevensteinRecursion(first, second.Substring(0, secondLenght - 1)) + 1;

result[2] = LevensteinRecursion(first.Substring(0, firstLenght - 1), second.Substring(0, secondLenght - 1))

+ ((first[firstLenght - 1] == second[secondLenght - 1]) ? 0 : 1);

return result.Min();

}

}

Динамика

public static int LevensteinDynamic(string first, string second)

{

int firstLenght = first.Length, secondLenght = second.Length;

int[,] result = new int[(firstLenght + 1), (secondLenght + 1)];

for (int i = 0; i <= firstLenght; i++)

{

result[i, 0] = i;

}

for (int j = 0; j <= secondLenght; j++)

{

result[0, j] = j;

}

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

{

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

{

result[i, j] = GetMin(result[i - 1, j] + 1, result[i, j - 1] + 1,

result[i - 1, j - 1] + (first[i - 1] == second[j - 1] ? 0 : 1));

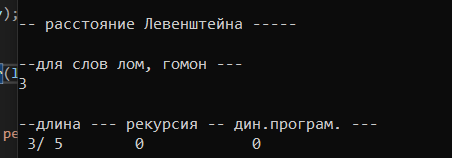
}

}

return result[firstLenght, secondLenght];

}

Проверка:



**Задание 5.**

**Нечетные варианты**. Выполнить сравнительный анализ времени затраченного на решение задачи о наибольшей общей подпоследовательности для двух методов решения (рекурсивное решение, динамическое программирование). Две последовательности взять в соответствии с вариантом. Построить графики зависимости времени вычисления от . **Отобразить ход решения в отчете**(по примеру из лекции) + код и копии экрана.

**Решение:**

// - LCS.h

// -- рекурсивное вычисление длины LCS

int lcs ( int lenx, // длина последовательности X

const char x[], // последовательность X

int leny, // длина последовательности Y

const char y[] // последовательность Y

);

// -- динамическое вычисление LCS

int lcsd( const char x[], // последовательность X

const char y[], // последовательность Y

char z[] // наибольшая общая подпоследовательность

);

// - LCS.cpp

// -- рекурсивное вычисление длины LCS

#include "stdafx.h"

#include <algorithm>

#include "LCS.h"

int lcs (int lenx, const char x[],

int leny, const char y[])

{

int rc = 0;

if (lenx > 0 && leny > 0)

{

if (x[lenx-1] == y[leny-1]) rc = 1 + lcs(lenx-1, x,leny-1, y);

else rc = std::max(lcs(lenx, x,leny-1, y), lcs(lenx-1, x,leny, y));

}

return rc; //длина LCS

}

#define LCS\_C(x1,x2) (C[(x1)\*(leny+1)+(x2)])

#define LCS\_B(x1,x2) (B[(x1)\*(leny+1)+(x2)])

#define LCS\_X(i) (x[(i)-1])

#define LCS\_Y(i) (y[(i)-1])

#define LCS\_Z(i) (z[(i)-1])

enum Dart{TOP,LEFT,LEFTTOP};

void getLCScontent( int lenx, int leny, const char x[],

const Dart\* B,

int n, int i, int j, char z[])

{

if ((i > 0 && j > 0 && n > 0 ))

{

if (LCS\_B(i,j) == LEFTTOP)

{

getLCScontent(lenx, leny,x, B, n-1, i-1, j-1, z);

LCS\_Z(n) = LCS\_X(i);

LCS\_Z(n+1) = 0;

}

else if (LCS\_B(i,j)== TOP)

getLCScontent(lenx, leny,x, B, n, i-1, j, z);

else getLCScontent(lenx, leny,x, B, n, i, j-1, z);

}

};

int lcsd(const char x[], const char y[], char z[])

{

int n;

int lenx = strlen(x), leny = strlen(x),

\*C = new int[(lenx+1)\*(leny+1)];

Dart\* B = new Dart[(lenx+1)\*(leny+1)];

memset(C,0,sizeof(int)\*(lenx+1)\*(leny+1));

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

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

if (LCS\_X(i) == LCS\_Y(j))

{LCS\_C(i,j) = LCS\_C(i-1,j-1)+1;

LCS\_B(i,j) = LEFTTOP;}

else if (LCS\_C(i-1,j) >= LCS\_C(i, j-1))

{

LCS\_C(i,j) = LCS\_C(i-1, j);

LCS\_B(i,j) = TOP;

}

else

{

LCS\_C(i,j) = LCS\_C(i, j-1);

LCS\_B(i,j) = LEFT;

}

getLCScontent(lenx, leny, x, B, LCS\_C(lenx,leny), lenx, leny, z);

return LCS\_C(lenx,leny);

}

#undef LCS\_Z

#undef LCS\_C

#undef LCS\_B

#undef LCS\_X

#undef LCS\_Y

// - main

// -- вычисления длины LCS

#include "stdafx.h"

#include <iostream>

#include "LCS.h"

int \_tmain(int argc, \_TCHAR\* argv[])

{

setlocale(LC\_ALL, "rus");

char z[100]="";

char X[]="ABCDFGI";

char Y[]="EATUFI";

std::cout<<std::endl<<"-- вычисление длины LCS для X и Y(рекурсия)";

std::cout<<std::endl<<"-- последовательность X: "<< X;

std::cout<<std::endl<<"-- последовательность Y: "<< Y;

int s = lcs(sizeof(X)-1, " BXWAFRE", sizeof(Y)-1, " XCDUFR" );

std::cout<<std::endl<< "-- длина LCS: "<<s<<std::endl;

// наибольшая общая подпоследовательность

int l = lcsd(x, y, z);

std::cout<<std::endl

<< "-- наибольшая общая подпоследовательость - LCS(динамическое"

<<"программирование)"<< std::endl;

std::cout<<std::endl<<"последовательость X: " << x;

std::cout<<std::endl<<"последовательость Y: " << x;

std::cout<<std::endl<<" LCS: " << z;

std::cout<<std::endl<<" длина LCS: " << l;

std::cout<<std::endl;

system("pause");

return 0;

}

На рисунке 3 представлена наибольшая общая подпоследовательность последовательностей Х и У, вычисленные при помощи метода динамического программирования, а также рекурсивным алгоритмом.

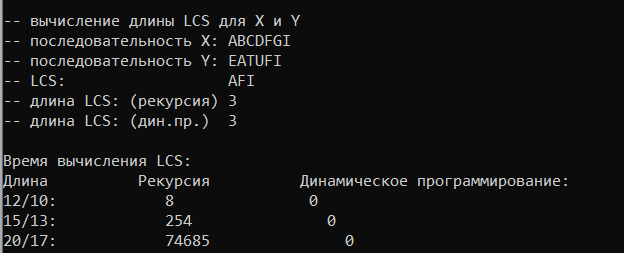


Рис. 3 – проверка работоспособности решений

На графике, представленном на рисунке 4, можно заметить, что выполненные с помощью динамического алгоритма, вычисления производятся в разы быстрее, чем с помощью рекурсивного алгоритма.

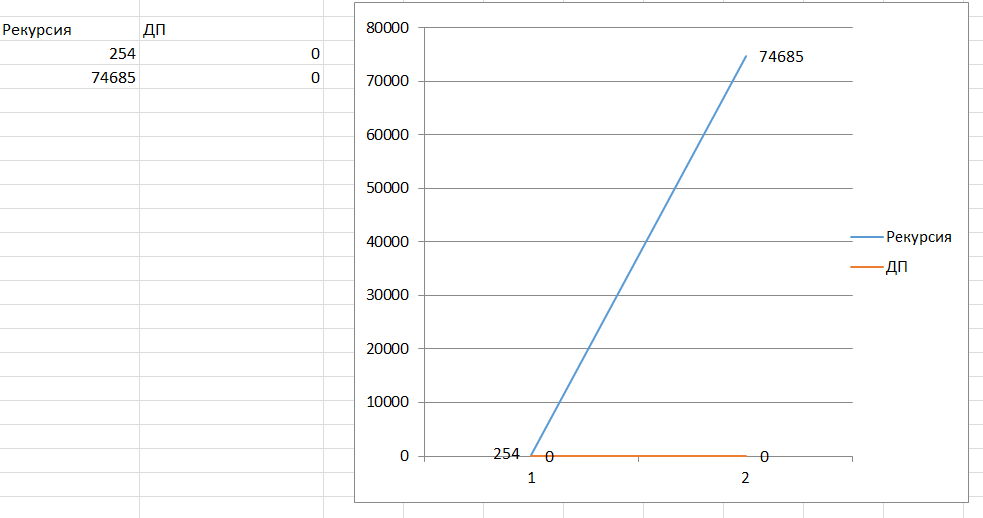


Рис. 4 – сравнительный анализ времени выполнения

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | E | A | T | U | F | I |  |  |  | E | **A** | T | U | **F** | **I** |
|  | 0 | 0 | 0 | 0 | 0 | 0 | 0 |  |  | - | - | - | - | - | - | - |
| A | 0 | 0 | 1 | 1 | 1 | 1 | 1 |  | **A** | - |  |  |  |  |  |  |
| B | 0 | 0 | 1 | 1 | 1 | 1 | 1 |  | B | - |  |  |  |  |  |  |
| C | 0 | 0 | 1 | 1 | 1 | 1 | 1 |  | C | - |  |  |  |  |  |  |
| D | 0 | 0 | 1 | 1 | 1 | 1 | 1 |  | D | - |  |  |  |  |  |  |
| F | 0 | 0 | 1 | 1 | 1 | 2 | 2 |  | **F** | - |  |  |  |  |  |  |
| G | 0 | 0 | 1 | 1 | 1 | 2 | 2 |  | G | - |  |  |  |  |  |  |
| I | 0 | 0 | 1 | 1 | 1 | 2 | **3** |  | **I** | - |  |  |  |  |  |  |

**Вывод:** в ходе работы были освоены общие принципы решения задач методом динамического программирования, проведено сравнение полученных решений задач с рекурсивным методом.