Учреждение образования

«Белорусский государственный технологический университет»

**Отчет по лабораторной работе №4**

**Тема «Динамическое программирование»**

Выполнила:

Студентка 2 курса 2 группы ФИТ

Максимова Вера Владимировна

Минск 2022

Цель работы. Освоить общие принципы решения задач методом динамического программирования, сравнить полученные решения задач с рекурсивным методом.

**Задание 1.**

На языке С++ сгенерировать случайным образом строку букв латинского алфавита  длиной  символов и длиной .

#include <algorithm>

#include <iostream>

#include <ctime>

#include <iomanip>

#include "Levenshtein.h"

int main()

{

setlocale(LC\_ALL, "rus");

clock\_t t1 = 0, t2 = 0, t3, t4;

char x[301], y[251];

srand(time(0));

for (int i = 0; i < 300; i++)

{

x[i] = ('a' + rand() % ('z' - 'a'));

std::cout << x[i];

}

std::cout << std::endl;

for (int i = 0; i < 250; i++)

{

y[i] = ('a' + rand() % ('z' - 'a'));

std::cout << y[i];

}

int lx = sizeof(x) - 1, ly = sizeof(y) - 1;

std::cout << std::endl;

std::cout << std::endl << "-- расстояние Левенштейна -----" << std::endl;

std::cout << std::endl << "--длина --- рекурсия -- дин.програм. ---" << std::endl;

for (int i = 0; i < 250; i++)

{

if (i < 13) { t1 = clock(); levenshtein\_r(i, x, i, y); t2 = clock(); }

t3 = clock(); levenshtein(i, x, i, y); t4 = clock();

std::cout << std::right << std::setw(2) << i << "/" << std::setw(3) << i

<< " ";

if (i < 13) std::cout << std::left << std::setw(10) << (t2 - t1);

else std::cout << std::left << std::setw(10) << "null";

std::cout << " " << std::setw(10) << (t4 - t3) << std::endl;

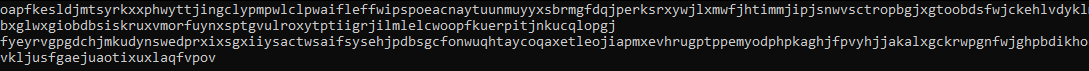
}

system("pause");

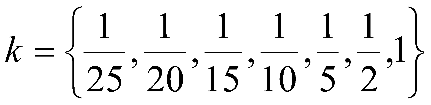
return 0;

}

Результат.



**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).

Код:

#include <algorithm>

#include <iostream>

#include <ctime>

#include <iomanip>

#include "Levenshtein.h"

using namespace std;

int main()

{

setlocale(LC\_ALL, "rus");

clock\_t t1 = 0, t2 = 0, t3, t4;

char x[25] = "abcdefghdd", y[] = "xyabcdeshrtrh";

int lx = sizeof(x) - 1, ly = sizeof(y) - 1;

std::cout << std::endl;

std::cout << std::endl << "-- расстояние Левенштейна -----" << std::endl;

std::cout << std::endl << "--длина --- рекурсия -- дин.програм. ---"

<< std::endl;

int u = 1;

for (int i = 6; i < std::min(lx, ly); i++)

{

t1 = clock(); levenshtein\_r(i, x, i - 2, y); t2 = clock();

t3 = clock(); levenshtein(i, x, i - 2, y); t4 = clock();

cout << right << setw(2) << 1 << "/" << setw(2) << u << " " << left << setw(10) << (t2 - t1) << " " << setw(10) << (t4 - t3) << endl;

if (u > 0) {

if (u == 20) { u = u + 5; }

if (u == 15) { u = u + 5; }

if (u == 10) { u = u + 5; }

if (u == 5) { u = u + 5; }

if (u == 2) { u = u + 3; }

if (u == 1) { u = u + 1; }

}

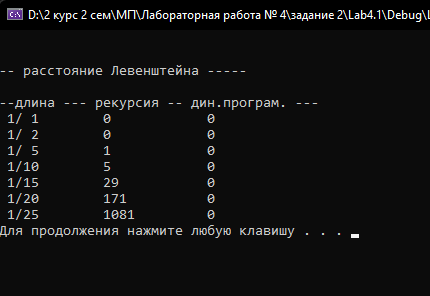
}

system("pause");

return 0;

}

Результат:



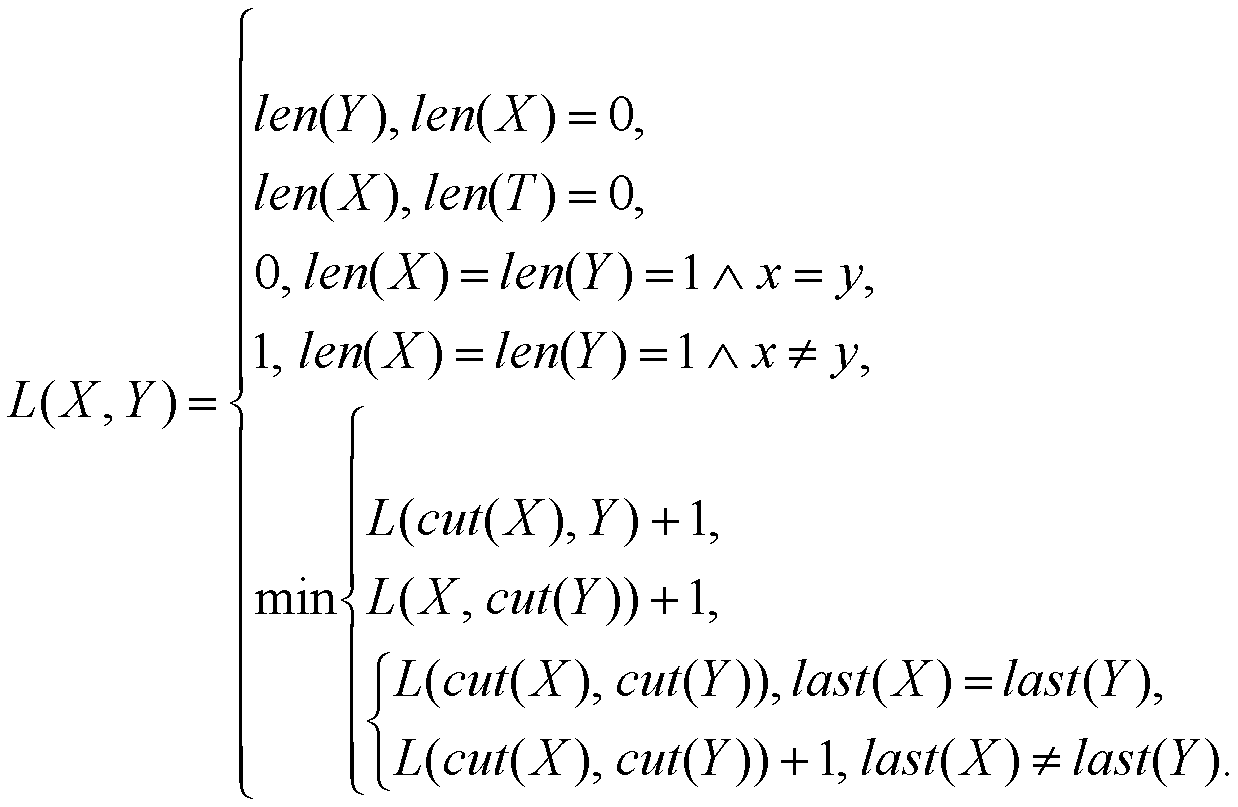
**Задание 3.**

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

**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).

Пусть и  – две символьные строки, тогда для вычисления дистанции Левенштейна  между ними может быть использовано следующее рекуррентное соотношение:



В предыдущем выражении используются символы   и  Разъясним их смысл:

 – количество символов в заданной строке. Например, 

 – заданная строка без последнего символа. Например, 

 – последний символ заданной строки. Например, 

|  |  |  |
| --- | --- | --- |
| 3 | Лом | Гомон |



























Операции:

Замена «л» на «г»

Добавление «о»

Добавление «н»

Следовательно дистанция Левинштейна: 3.

Реализация с помощью кода:

// - main

#include <iostream>

#include "Levenshtein.h"

int main()

{

setlocale(LC\_ALL, "rus");

char X[] = "лом", Y[] = "гомон";

std::cout << std::endl << "-- дистанция Левенштейна (рекурсия)";

std::cout << std::endl << X << " --> " << Y << " = "

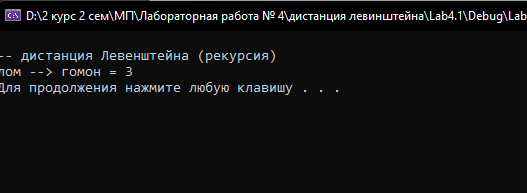
<< levenshtein\_r(sizeof(X) - 1, X, sizeof(Y) - 1, Y) << std::endl;

system("pause");

return 0;

}

Вывод:



**Задание 5. (нет решения, есть excel)**

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 k . Отобразить ход решения в отчете(по примеру из лекции) + код и копии экрана.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 3 | Лом | Гомон | ABCDFGI | EATUFI |

Код:

#include <iostream>

int main()

{

setlocale(LC\_ALL, "rus");

char X[] = "ABCDFGI", 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, // длина последовательности X

"ABCDFGI", // последовательность X

sizeof(Y) - 1, // длина последовательности Y

"EATUFI" // последовательность Y

);

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

//DYNAMIC

char z[100] = "";

char x[] = "ABCDFGI",

y[] = "EATUFI";

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: " << y;

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

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

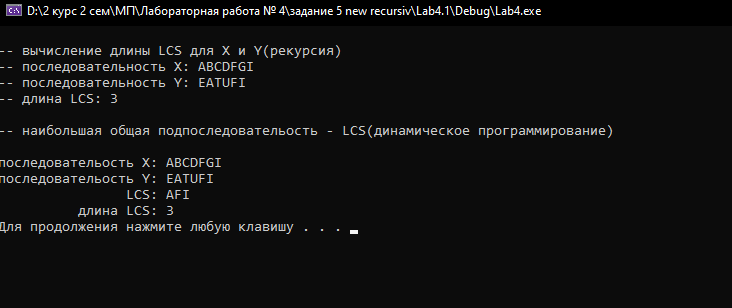
std::cout << std::endl;

system("pause");

return 0;

}

Вывод.



Порядок заполнения таблицы:

– первая строка и первый столбец таблицы заполняются нулями;

– каждый элемент последовательно заполняется по формуле для c[i, j]:

если символы для позиции i, j совпадают, то в неё записывается значение c[i−1, j −1]+1, иначе вычисляется максимум от соседей слева и сверху.

Элемент в правом нижнем углу показывает длину наибольшей общей

подпоследовательности.

Вторая матрица заполняется следующим образом: все ячейки кроме технических строк заполняются стрелками, направленными вверх.

Если если символы для позиции i, j совпадают, то стрелка меняется на лево-вверх.

Если числовое значение от соседа слева больше, чем от соседа сверху, то стрелка меняется на лево.

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

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | |  | E | A | T | U | F | I |
|  |  |  |  |  |  |  |
| A |  | ↑ | ↑ |  | ↑ | ↑ | ↑ | ↑ |
| B |  | ↑ | ↑ | ↑ | ← | ↑ | ↑ | ↑ |
| C |  | ↑ | ↑ | ↑ | ↑ | ← | ↑ | ↑ |
| D |  | ↑ | ↑ | ↑ | ↑ | ↑ | ← | ↑ |
| F |  | ↑ | ↑ | ↑ | ↑ | ↑ |  | ← |
| G |  | ↑ | ↑ | ↑ | ↑ | ↑ | ↑ | ← |
| I |  | ↑ | ↑ | ↑ | ↑ | ↑ | ↑ |  |

**Вывод.**

В ходе лабораторной работы были освоены методы динамического программирования. Полученные решения были сравнены с рекурсивным методом.