Задание №1 На языке С++ сгенерировать случайным образом строку букв латинского алфавита длиной символов и длиной .

srand(time(NULL));

char abc[27]; // наш алфавит

char s1[301];

char s2[251];

// заполняем массив

for (int i = 97, n = 0; i <= 122; ++i, n++)

{

abc[n] = (char)i;

}

abc[26] = '\0';

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

{

s1[i] = abc[\_rand(0, 25)];

}

s1[300] = '\0';

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

{

s2[i] = abc[\_rand(0, 25)];

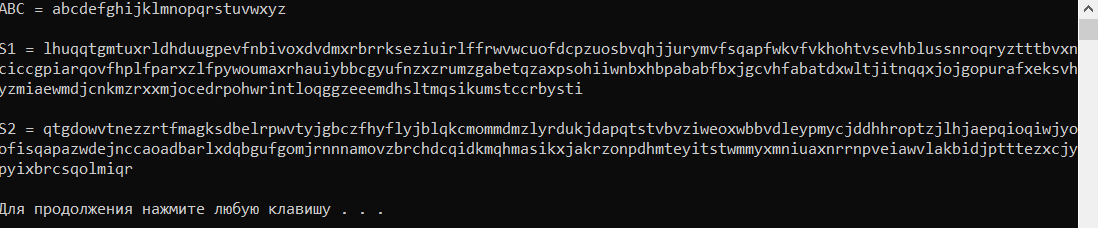
}

s2[250] = '\0';

cout << "ABC = " << abc << endl << endl;

cout << "S1 = " << s1 << endl << endl;

cout << "S2 = " << s2 << endl << endl;



Задание №2 Вычислить двумя способами (рекурсивно и с помощью динамического программирования) – Левенштейна для , где - длина строки , - строка, состоящая из первых символов строки . (копии экрана и код вставить в отчет).

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

Исходный код реализации через динамическое программирование:

int min3(int x1, int x2, int x3)

{

return std::min(std::min(x1, x2), x3);

}

int levenshtein(int lx, const char x[], int ly, const char y[])

{

int \*d = new int[(lx + 1)\*(ly + 1)];

for (int i = 0; i <= lx; i++) DD(i, 0) = i;

for (int j = 0; j <= ly; j++) DD(0, j) = j;

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

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

{

DD(i, j) = min3(DD(i - 1, j) + 1, DD(i, j - 1) + 1,

DD(i - 1, j - 1) + (x[i - 1] == y[j - 1] ? 0 : 1));

}

return DD(lx, ly);

}

Пример реализации рекурсивным методом:

int levenshtein\_r(

int lx, const char x[],

int ly, const char y[]

)

{

int rc = 0;

if (lx == 0) rc = ly;

else if (ly == 0) rc = lx;

else if (lx == 1 && ly == 1 && x[0] == y[0]) rc = 0;

else if (lx == 1 && ly == 1 && x[0] != y[0]) rc = 1;

else rc = min3(

levenshtein\_r(lx - 1, x, ly, y) + 1,

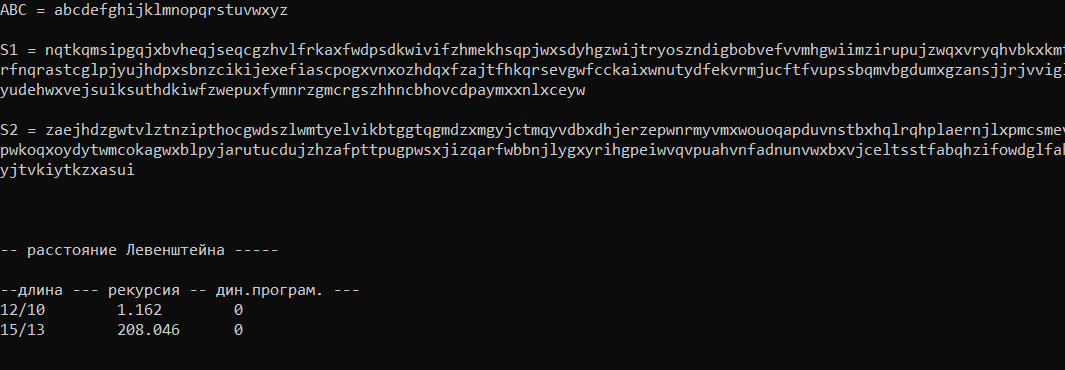
levenshtein\_r(lx, x, ly - 1, y) + 1,

levenshtein\_r(lx - 1, x, ly - 1, y) + (x[lx - 1] == y[ly - 1] ? 0 : 1)

);

return rc;

};



Задание №3

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).



= 5.

= 4.



= 4.

= 3.



= 3.

= 2.



= 2.

= 1.



= 3.

= 2.



= 2.

= 1.



Задание №5 Выполнить сравнительный анализ времени, затраченного на решение задачи о наибольшей общей подпоследовательности для двух методов решения (рекурсивное решение, динамическое программирование). Две последовательности взять с соответствии с вариантом. Построить графики зависимости времени вычисления от k. Отобразить ход решения в отчете (по примеру лекции) + код + копии экрана.

#include <algorithm>

#include <cstring>

#include "LCS.h"

//---рекурсивный

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

}

//---Динамичческое программирование

#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.cpp

char X[] = "HRWCYUJ", Y[] = "WLPCAU";

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

"HRWCYUJ", // последовательность X

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

"WLPCAU" // последовательность Y

);

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

char z[100] = "";

char x[] = "HRWCYUJ",

y[] = "WLPCAU";

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;

