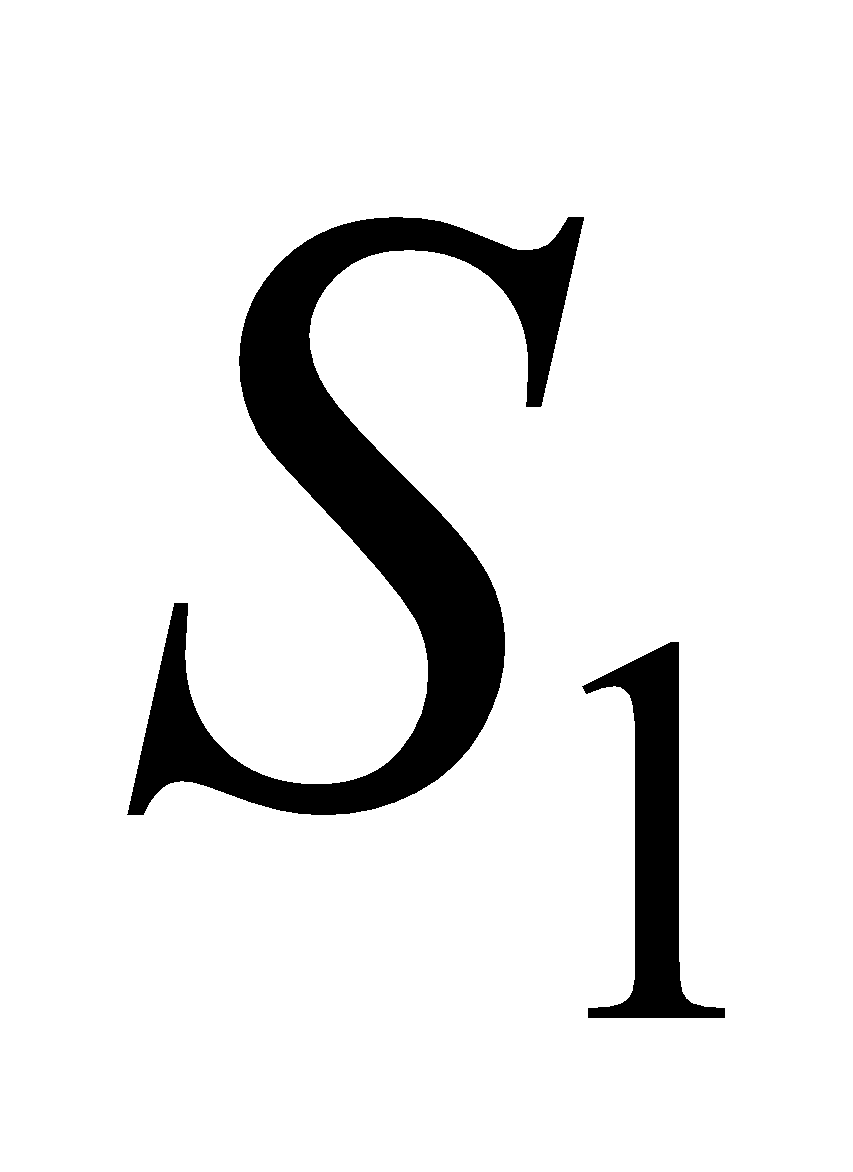
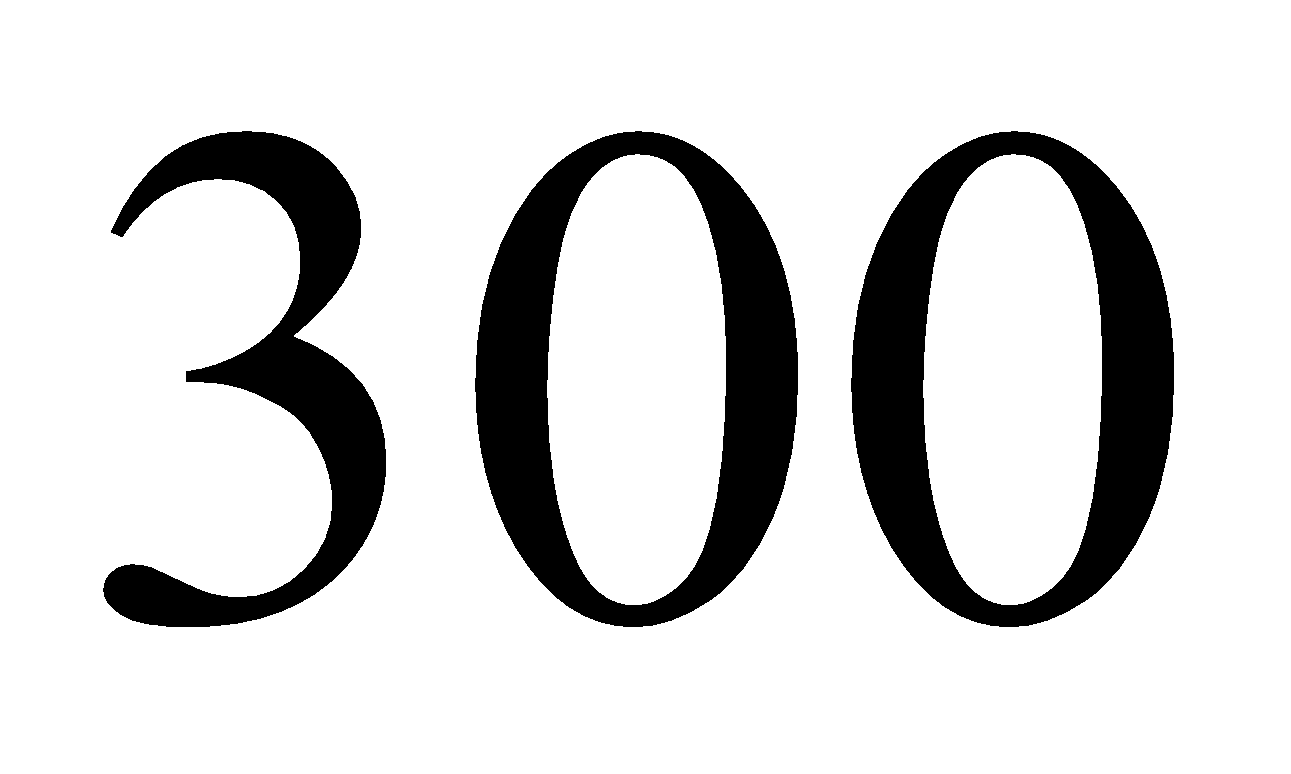
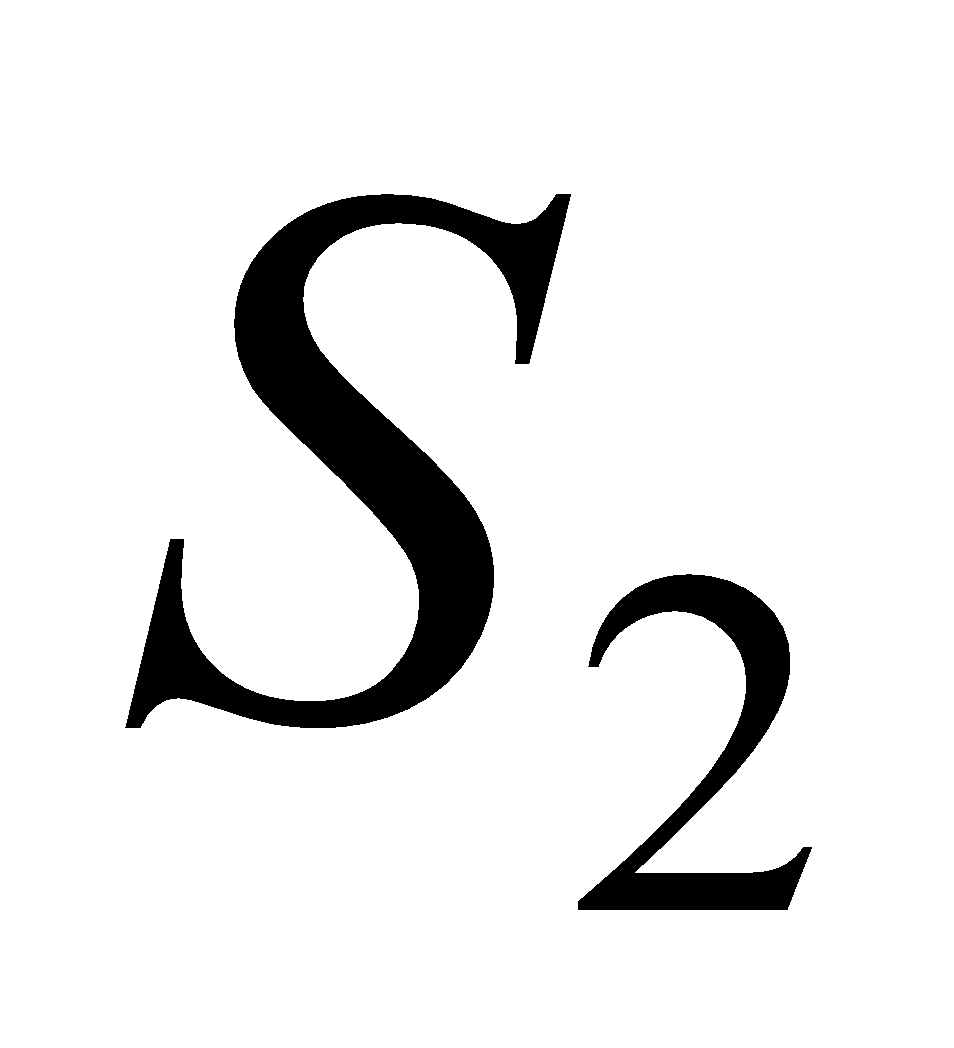
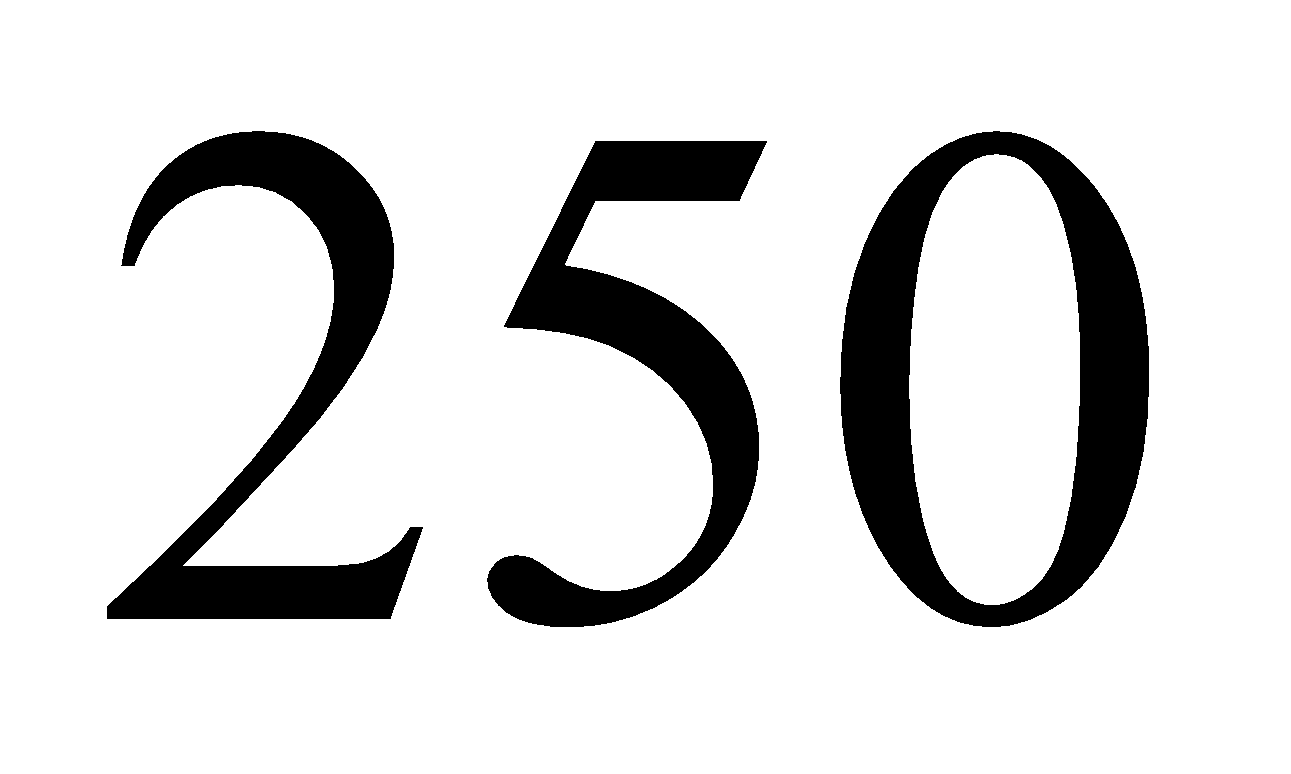
**Лабораторная работа 4**

**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  длиной  символов и длиной .

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

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

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

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

std::cout << "S1: ";

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

{

std::cout << x[i];

}

std::cout << "\n";

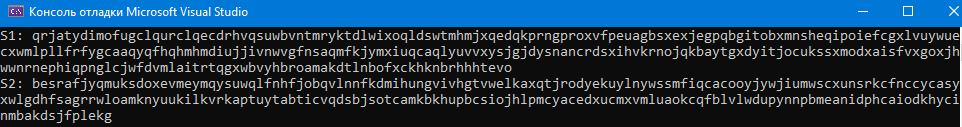
std::cout << "S2: ";

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

{

std::cout << y[i];

}



**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).

// - Levenshtein.h

// -- дистанции Левенштeйна (динамическое программирование)

int levenshtein(

int lx, // длина слова x

const char x[], // слово длиной lx

int ly, // длина слова y

const char y[] // слово y

);

// -- дистанции Левенштeйна (рекурсия)

int levenshtein\_r(

int lx, // длина строки x

const char x[], // строка длиной lx

int ly, // длина строки y

const char y[] // строка y

);

// - Levenshtein.cpp

#include "stdafx.h"

#include <iomanip>

#include <algorithm>

#include "Levenshtein.h"

#define DD(i,j) d[(i)\*(ly+1)+(j)]

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);

}

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;

};

// --- main

// вычисление дистанции (расстояния) Левенштейна

#include "stdafx.h"

#include <algorithm>

#include <iostream>

#include <ctime>

#include <iomanip>

#include "Levenshtein.h"

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

{

setlocale(LC\_ALL, "rus");

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

char x[] = "abcdefghklmnoxm", y[] = "xyabcdefghomnkm";

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 = 8; 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();

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

<< " "<<std::left<<std::setw(10)<<(t2-t1)

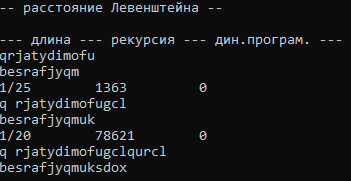
<<" "<<std::setw(10)<<(t4-t3)<<std::endl;

}

system("pause");

return 0;

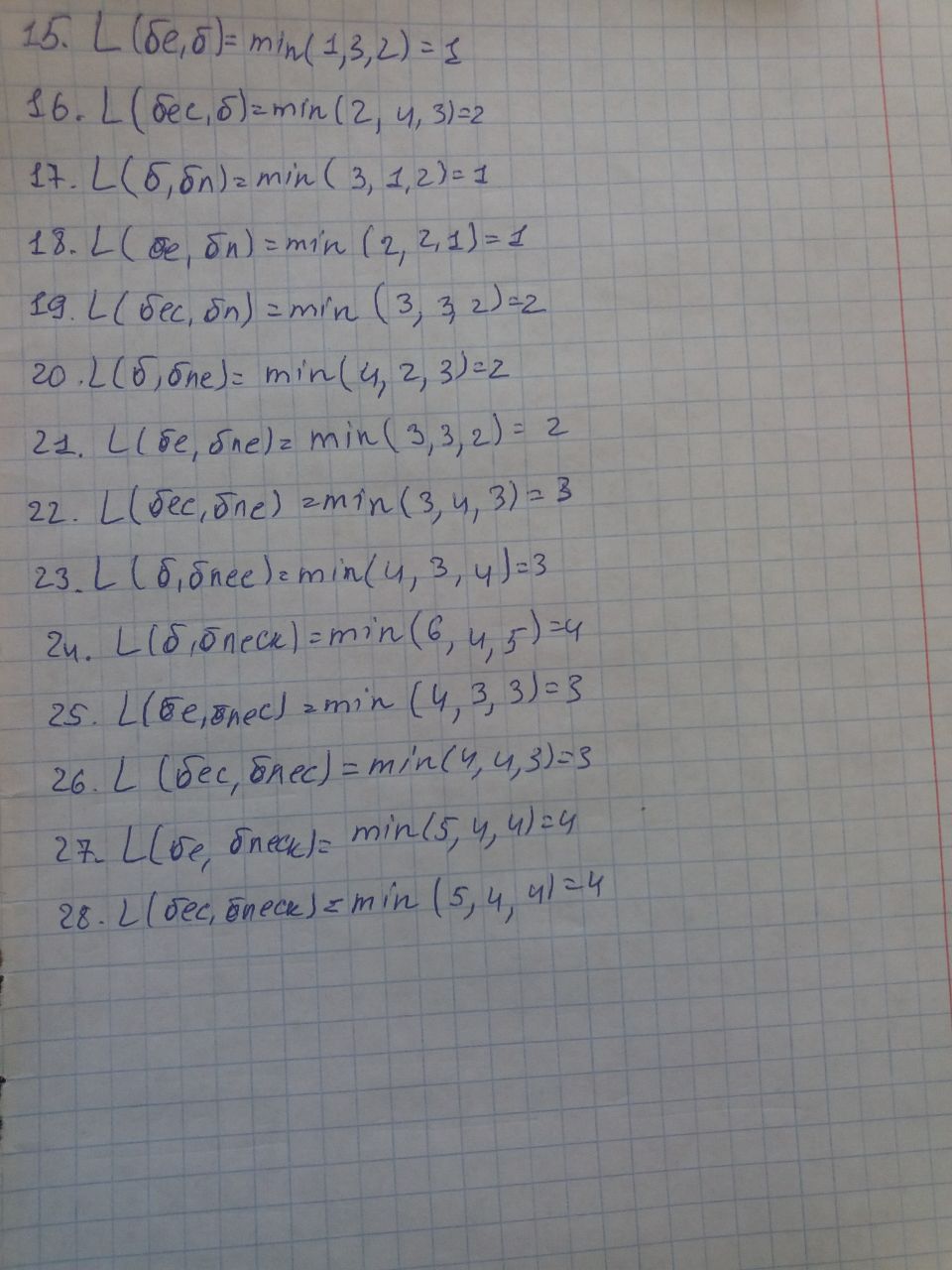
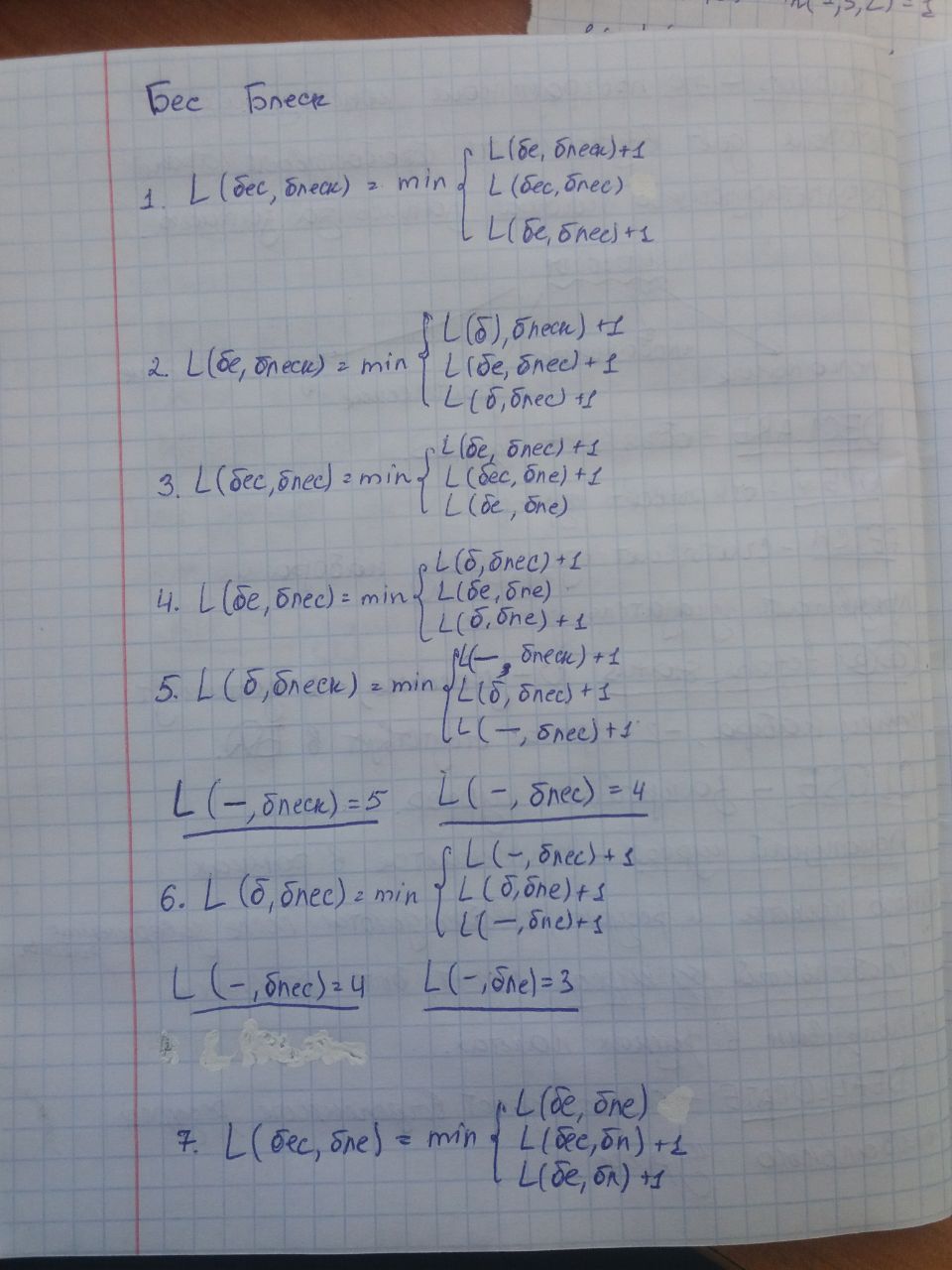
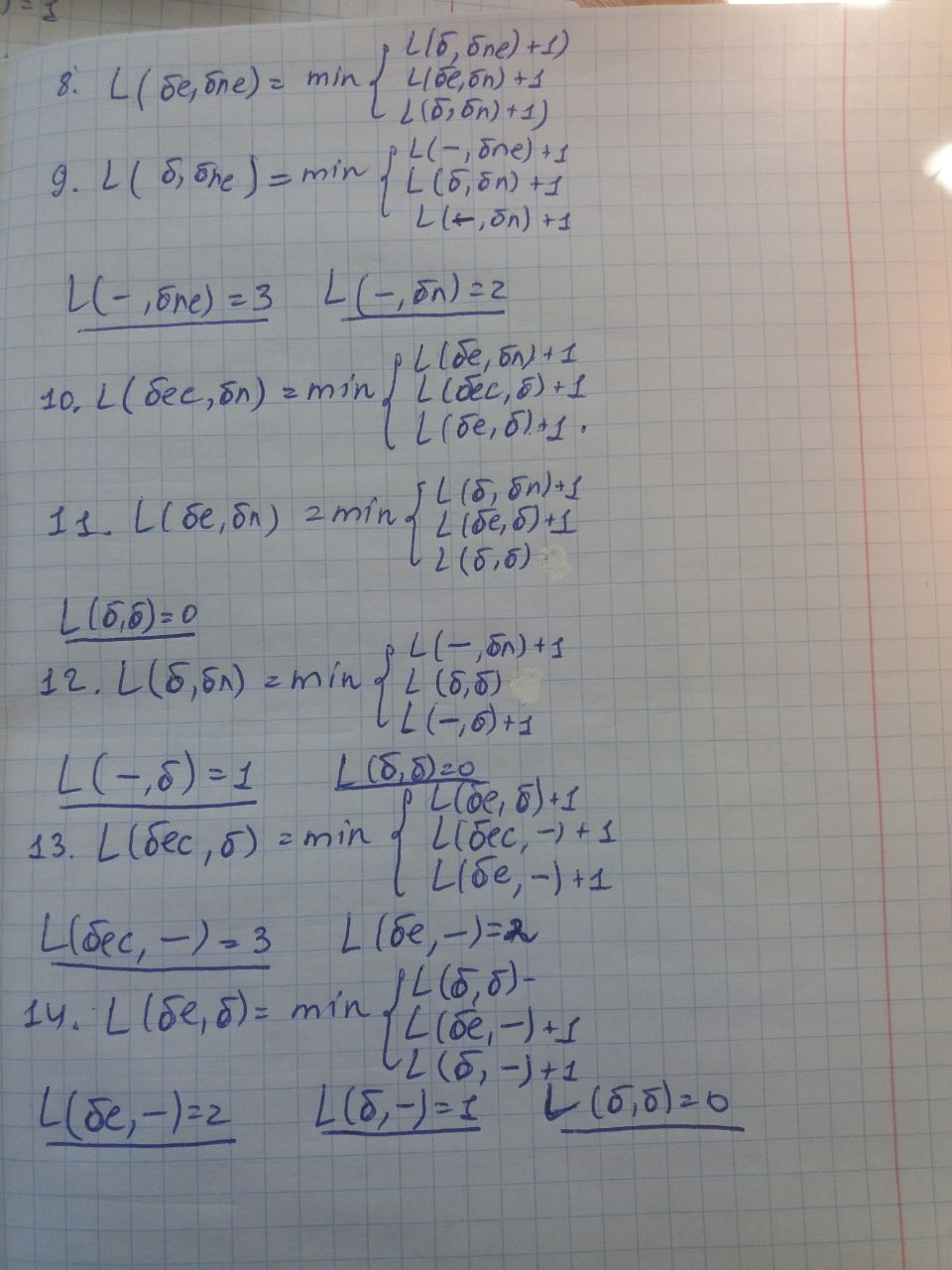
}



**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 отч ет).

**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).

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

**Рекурсивное решение**

#pragma once

// - 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

);

#include <algorithm>

#include <tchar.h>

#include <iostream>

#include "LCS.h"

// - main

// -- вычисления длины LCS

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

{

setlocale(LC\_ALL, "rus");

char X[] = "ALBDACD", Y[] = "CDLDCA";

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

"ALDC", // последовательность X

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

"LADCM" // последовательность Y

);

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

system("pause");

return 0;

}

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

int leny, const char y[])

{

int rc = 0;

if (lenx > 0 && leny > 0)

{

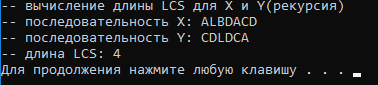
if (x[lenx - 1] == y[leny - 1]) rc = 1 + lcs(lenx - 1, x, leny - 1, y);

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

}

return rc; //длина LCS

}

****

**Динамическое программирование**

//- LCH.h

int lcsd(

const char x[], // последовательность X

const char y[], // последовательность Y

char z[] // наибольшая общая подпоследовательность

);

#pragma once

//- LCS.cpp

#include <cstring>

#include <iostream>

#include <tchar.h>

#include "LCS.h"

#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

// наибольшая общая подпоследовательность

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

{

setlocale(LC\_ALL, "rus");

char z[100] = "";

char x[] = "ALBDACD",

y[] = "CDLDCA";

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;

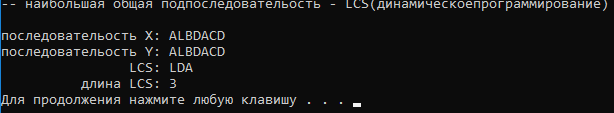
system("pause");

return 0;

}

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | A | L | B | D | A | C | D |
|  | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| C | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| D | 0 | 0 | 0 | 0 | 1 | 1 | 2 | 2 |
| L | 0 | 0 | 1 | 1 | 1 | 2 | 2 | 2 |
| D | 0 | 0 | 1 | 1 | 2 | 3 | 3 | 4 |
| C | 0 | 0 | 1 | 1 | 2 | 3 | 4 | 4 |
| A | 0 | 1 | 2 | 2 | 3 | 4 | 5 | 5 |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | A | L | B | D | A | C | D |
|  |  |  |  |  |  |  |  |  |
| C |  | ↑ | ↑ | ↑ | ↑ | ↑ | ↖ | ↑ |
| D |  | ↑ | ↑ | ↑ | ↖ | ← | ↑ | ↖ |
| L |  | ↑ | ↖ | ← | ↑ | ↑ | ↑ | ↑ |
| D |  | ↑ | ↑ | ↑ | ↖ | ↑ | ← | ← |
| C |  | ↑ | ↑ | ↑ | ↑ | ↑ | ↖ | ↑ |
| A |  | ↖ | ↑ | ← | ↑ | ↖ | ↑ | ← |



**Вывод:** освоили общие принципы решения задач методом динамического программирования, сравнили полученные решения задач с рекурсивным методом.