МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ

ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Программное обеспечение информационных технологий

Дисциплина Математическое программирование

**Отчет по лабораторным работам по дисциплине**

**“Математическое программирование”**

Выполнил: студент 2курса 5 группы специальности “ПОИТ” Кравченко А.Д.

(Ф.И.О)

Минск 2023

**Лабораторная работа №1**

Вспомогательные функции

**Задание 1**

Разработайте три функции (start, dget и iget), используя следующие спецификации (рисунок 1):

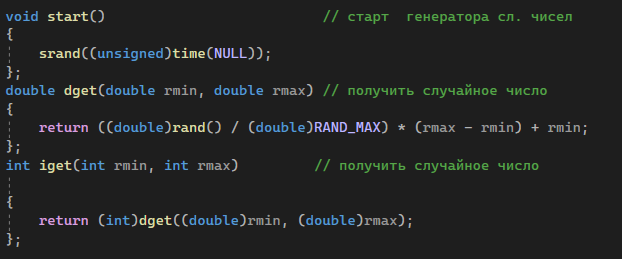


Рисунок 1 – Реализация функций

**Задание 2**

1.Реализовать пример 2.

2.Для проверки работоспособности разработанных функций и приобретения навыков замера продолжительности процесса вычисления реализуйте программу, приведенную в примере 2.

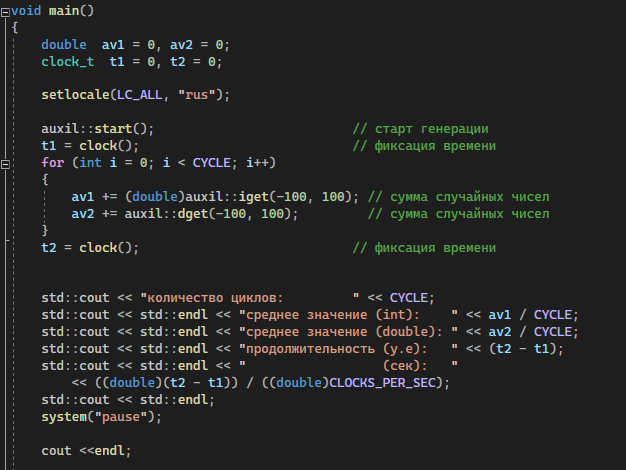


Рисунок 2 – вызов функции и измерение времени выполнения

**Задание 3**

Проведите необходимые эксперименты (разработать кодом) и постройте график зависимости (Excel) продолжительности процесса вычисления от количества циклов в примере 2.

Примечание: продолжительность вычисления измерять в условных единицах процессорного времени (функция clock). График представлен на рисунке 3

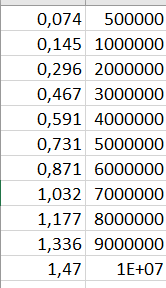
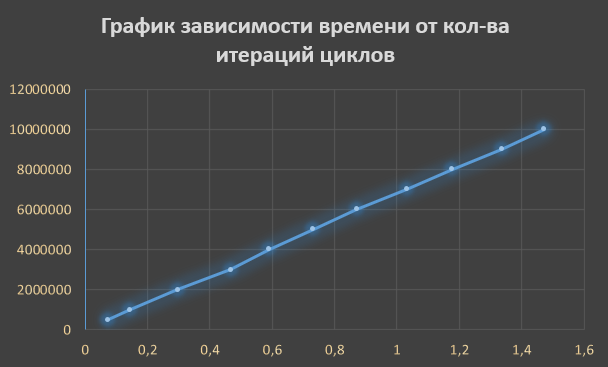


Рисунок 3 – график зависимости

Реализация Факториала в коде демонстрируется на рис. 4

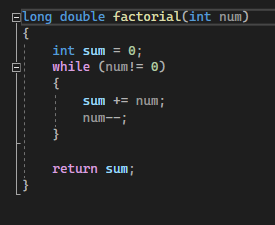


Рисунок 4 – функция для нахождения факториала

Вызов и подсчет времени для реализации факториала представлен на рисунке 5:

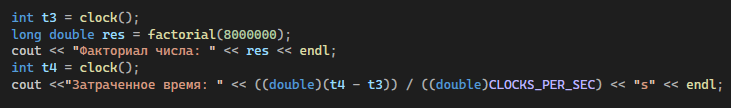


Рисунок 5 – вызов функции для подсчёта факториала

График зависимости:

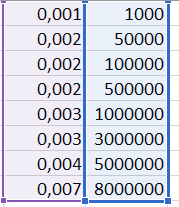
 

Рисунок 6 – график зависимости

**Вывод**: по полученным измерениям, можем сделать следующий вывод, что время выполнения программы линейно зависит от количества циклов (Значения факториала).

**Лабораторная работа №2**

Комбинаторные алгоритмы решения оптимизационных задач

**Задание 1.** Разобрать и разработать генератор подмножеств заданного множества.

В этом задании было необходимо разработать генератор подмножеств заданного множества. Ниже на рисунках представлены заголовочные файлы и файлы cpp с реализацией поставленной задачи.

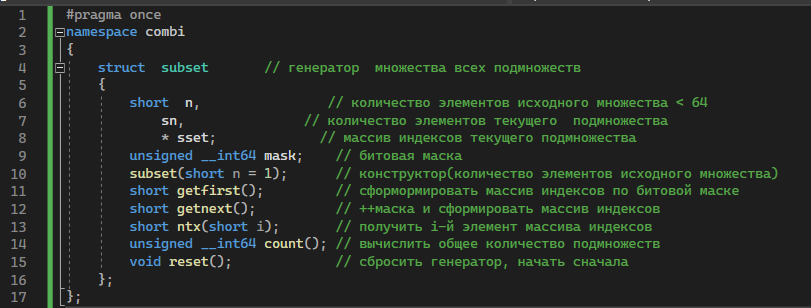


Рисунок 1 – Combi.h

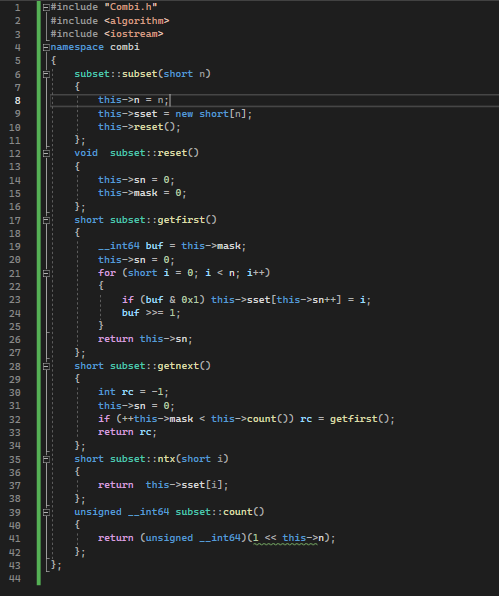


Рисунок 2 - Combi.cpp

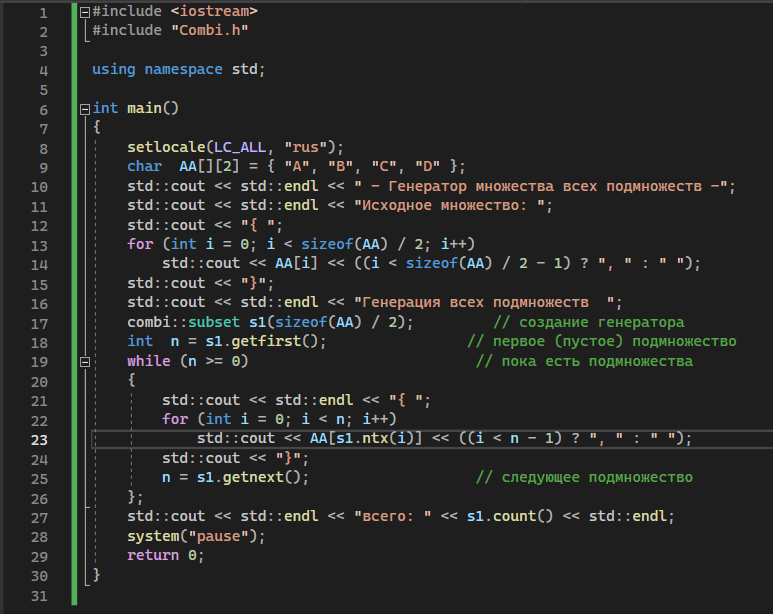


Рисунок 3 - Main\_ex1.cpp

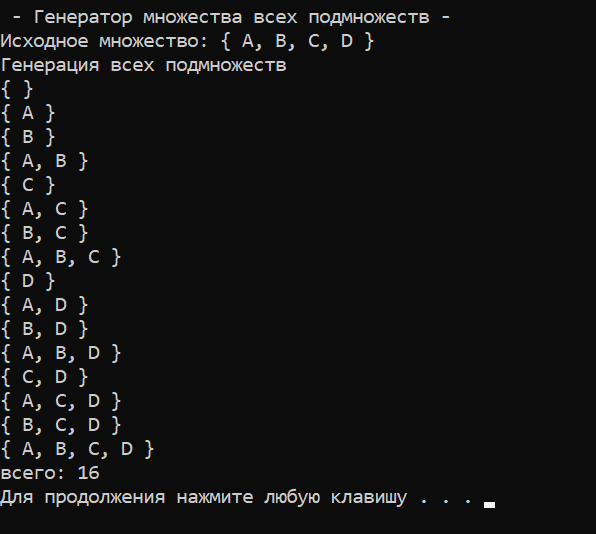


Рисунок 4 – Результат выполнения

**Задание 2.** Разобрать и разработать генератор сочетаний.

В этом задании было необходимо разработать генератор сочетаний. Ниже на рисунках представлены заголовочные файлы и файлы cpp с реализацией поставленной задачи.

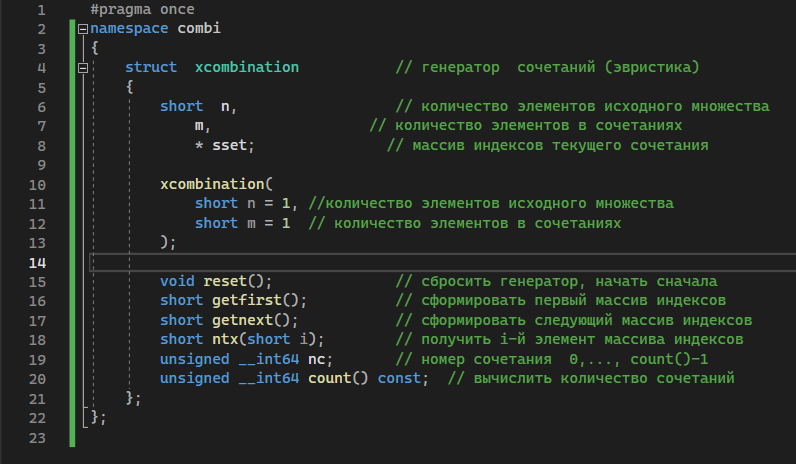


Рисунок 5 - Combi.h

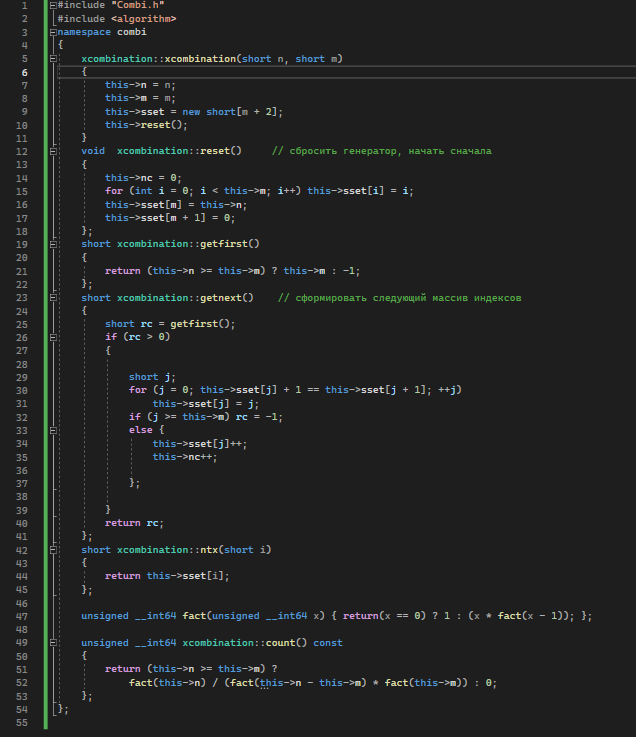


Рисунок 6 - Combi.cpp

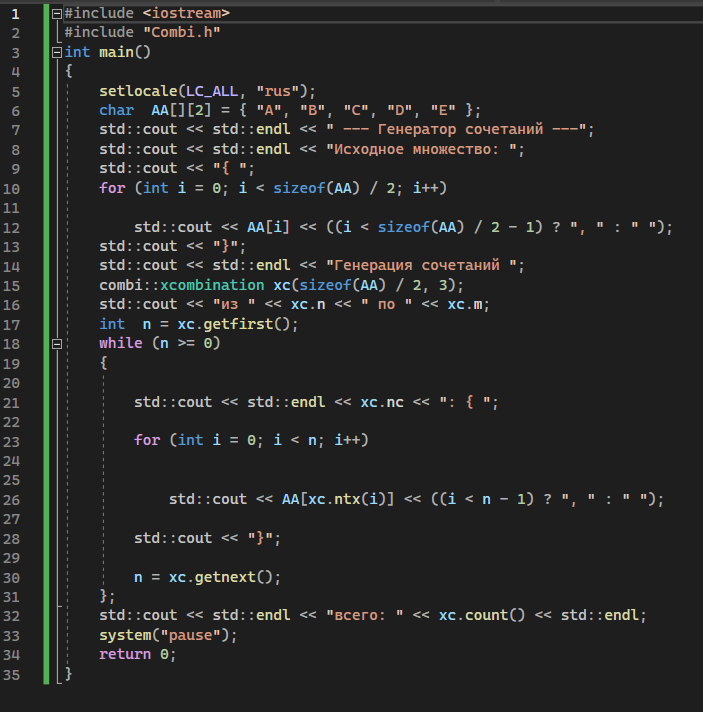


Рисунок 7 - Main\_ex2.cpp

Результат выполнения представлен на рисунке 8.

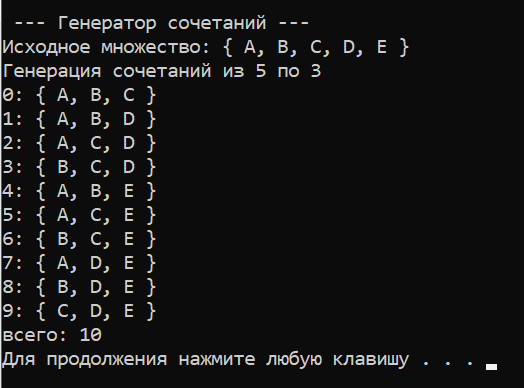


Рисунок 8 – результат выполнения программы

**Задание 3.** Разобрать и разработать генератор перестановок.

В этом задании было необходимо разработать генератор перестановок. Ниже на рисунках представлены заголовочные файлы и файлы cpp с реализацией поставленной задачи.

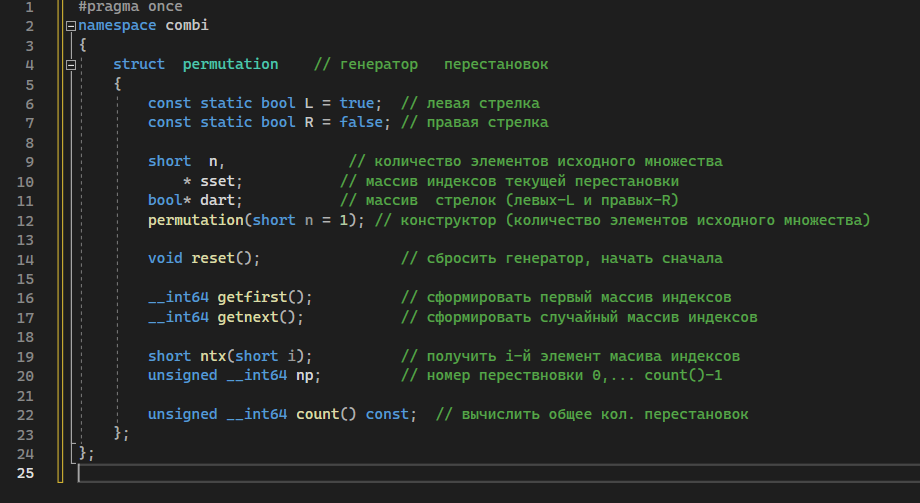


Рисунок 9 – Combi.h

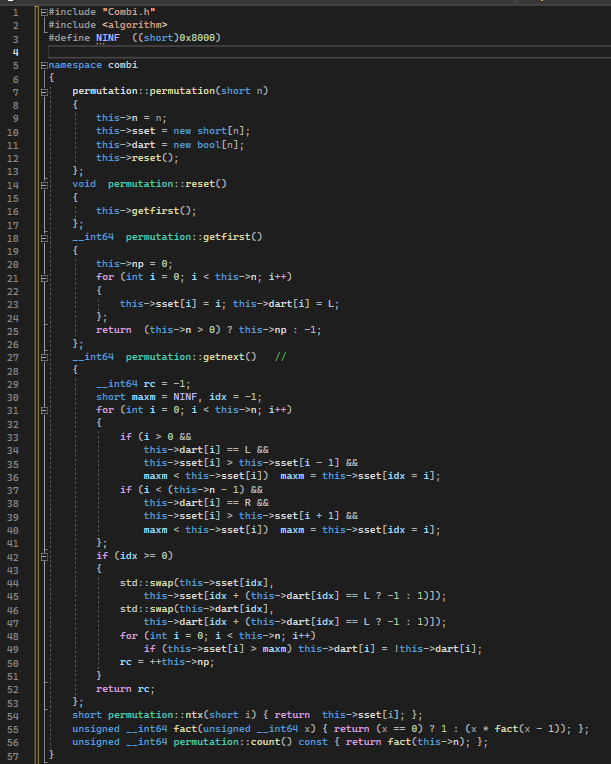


Рисунок 10 – Combi.cpp

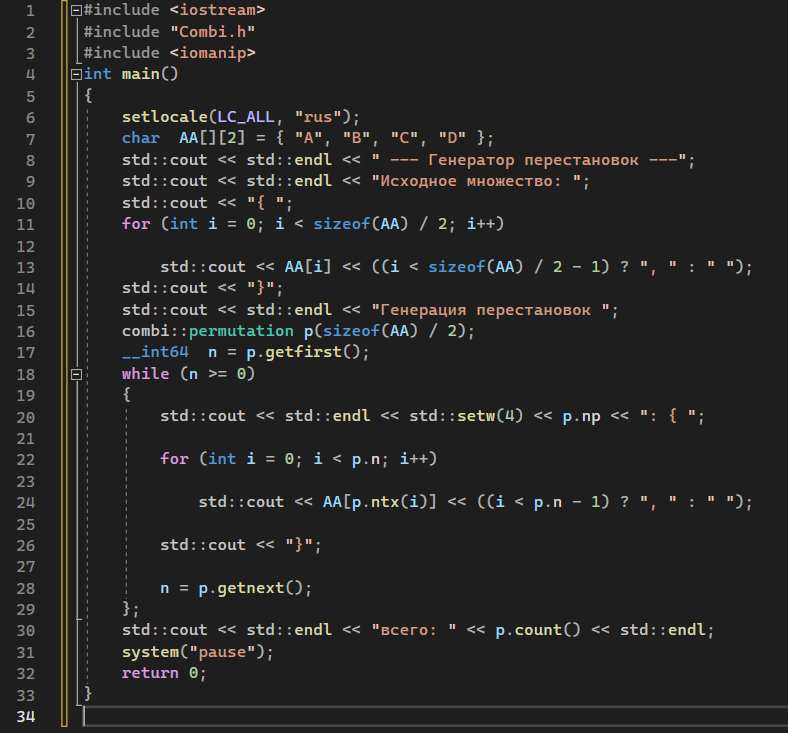


Рисунок 11 – Main\_ex3.cpp

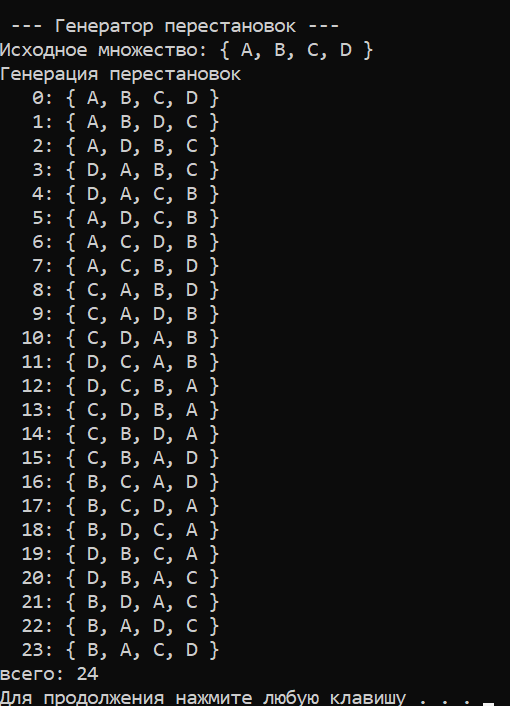


Рисунок 12 – результат выполнения программы

**Задание 4.** Разобрать и разработать генератор размещений.

В этом задании было необходимо разработать генератор размещений. Ниже на рисунках представлены заголовочные файлы и файлы cpp с реализацией поставленной задачи.

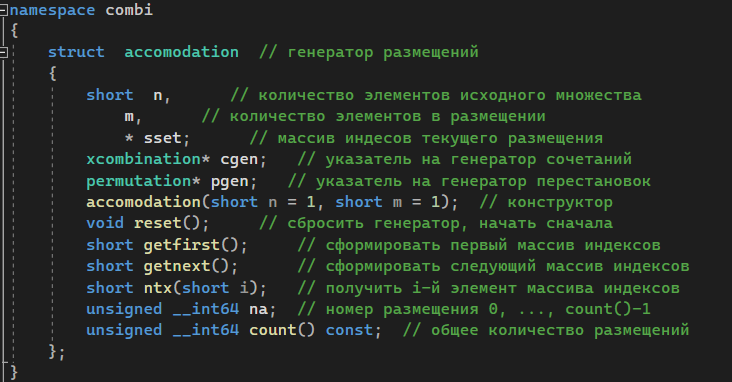


Рисунок 13 – Combi.h



Рисунок 14 – Combi.cpp

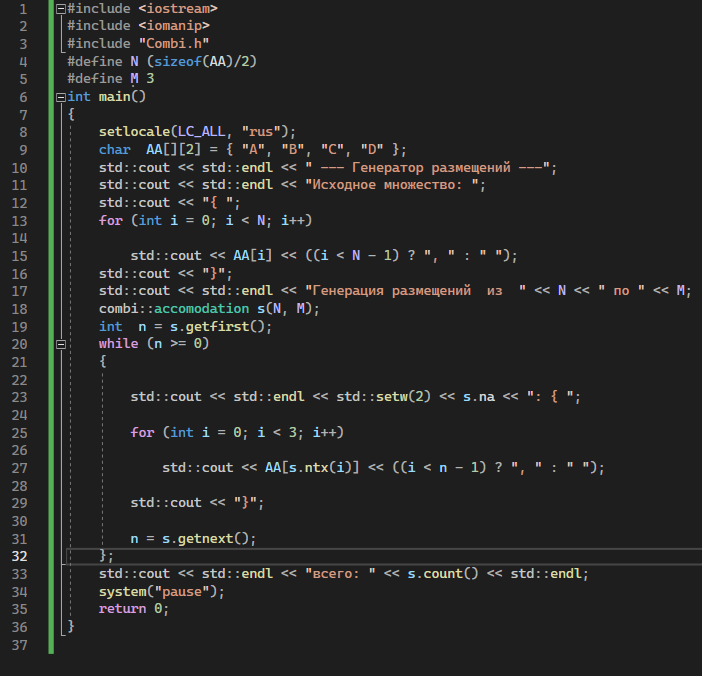


Рисунок 15 – Main\_ex4.cpp

Результат выполнения программы представлен на рисунке 16.

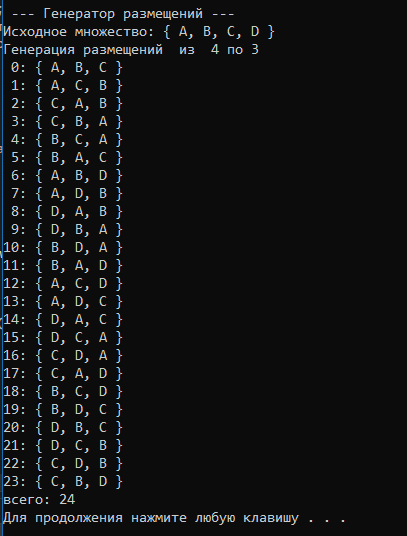


Рисунок 16 - результат выполнения программы

**Задание 5.** Решить в соответствии с вариантом задачу и результат занести в отчет. У меня 7ой вариант, поэтому условие задачи следующее:

об оптимальной загрузке судна (веса контейнеров сгенерировать случайным образом: ограничение по общему весу – 1500 кг., количество мест на судне для контейнеров – 5, количество контейнеров 25, веса контейнеров 100 – 900 кг., доход от перевозки 10 – 150 у.е.);

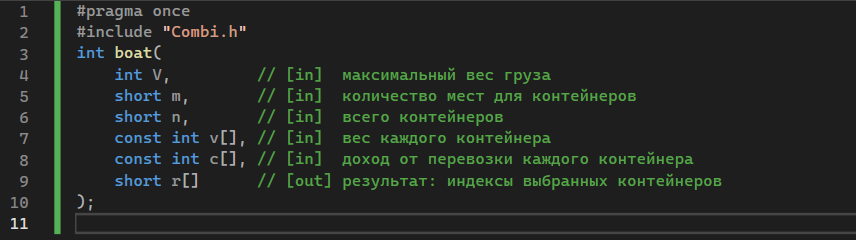


Рисунок 17 – Boat.h

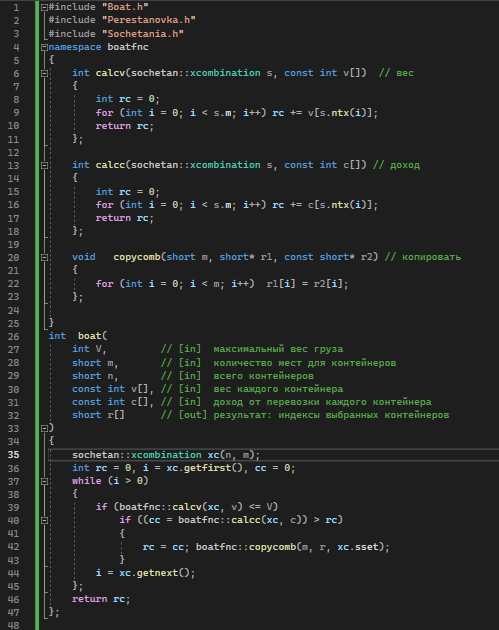


Рисунок 18 – Boat.cpp

Результат выполнения программы представлен на рисунке 19.

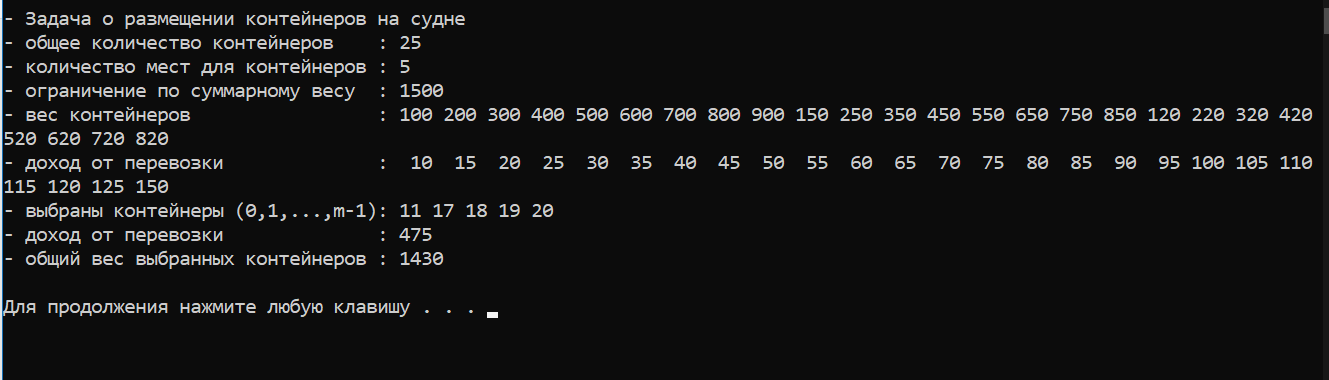


Рисунок 19 – результат работы программы

**Задание 6.** Исследовать зависимость времени вычисления необходимое для решения задачи от размерности задачи и результат в виде графика:

об оптимальной загрузке судна (количество мест на судне для контейнеров – 6, количество контейнеров 25 – 35

Результат работы программы на рисунке 20.

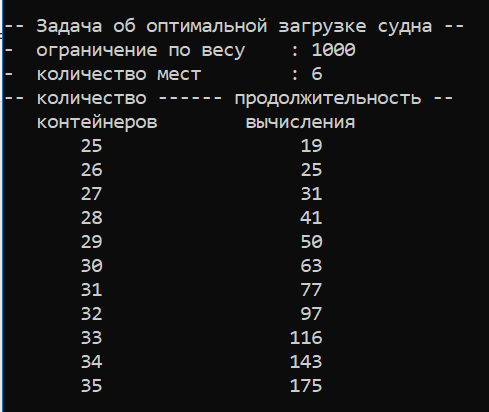


Рисунок 20 – Зависимость времени выполнения от количества контейнеров



Рисунок 21 – график зависимости

Вывод: исходя из полученных данных и графика, можно заметить, что скорость выполнения программы плавно возрастает при добавлении количества контейнеров.

**Лабораторная работа 3.**

Метод ветвей и границ. Задача коммивояжера и методы её решения.

**Задание 1.**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Город** | **1** | **2** | **3** | **4** | **5** |
| **1** |  | 14 | 28 |  | 7 |
| **2** | 7 |  | 22 | 61 | 77 |
| **3** | 9 | 21 |  | 86 | 56 |
| **4** | 24 | 51 | 28 |  | 21 |
| **5** | 86 | 73 | 52 | 20 |  |

*Вариант 7\**

Задача коммивояжера с параметром заключается в поиске гамильтонова цикла минимального веса в полном взвешенном графе, заданном матрицей расстояний, при условии, что в этом цикле есть ребро с максимальным весом не больше заданного параметра.

В данном случае матрица расстояний содержит расстояния между 5 городами, заданными номерами от 1 до 5. Значение INF означает отсутствие ребра между соответствующими городами. Задача коммивояжера с параметром для данной матрицы заключается в поиске гамильтонова цикла минимального веса в графе, при условии, что в этом цикле есть ребро с максимальным весом не больше заданного параметра.

Гамильтонов цикл – такой путь, который проходит через все вершины графа ровно один раз.

Параметр в задаче коммивояжера с параметром - это число, которое ограничивает вес ребра, которое может быть включено в найденный гамильтонов цикл.

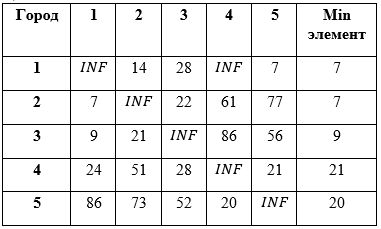
Более конкретно, параметр определяет максимально допустимый вес ребра в гамильтоновом цикле. Если в графе есть несколько ребер, вес которых больше параметра, то цикл, содержащий любое из этих ребер, не будет оптимальным.

Цель задачи коммивояжера с параметром - найти гамильтонов цикл минимального веса в заданном графе, который удовлетворяет ограничению на максимальный вес ребра. Значение параметра может быть задано заранее или получено в результате вычислений, и влияет на решение задачи.

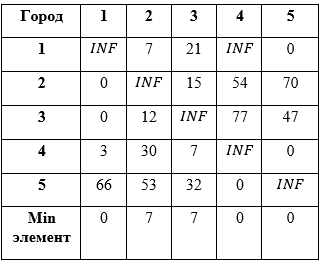
**Задание 2.**

Алгоритм решения (картинки идут последовательно, нумерация отсутствует):

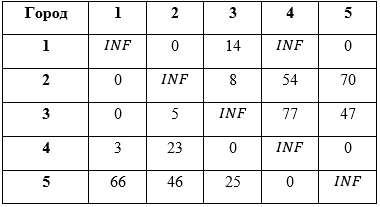
1)



2)

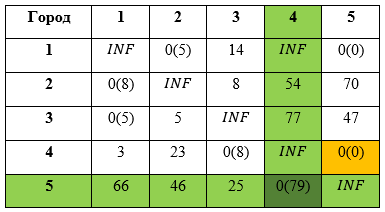


3)

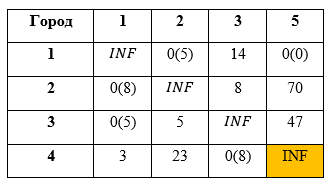


4)Н0 = 2\*7+9+21+20+14=78

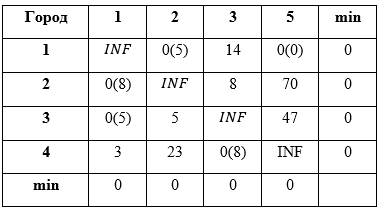
5)



6)



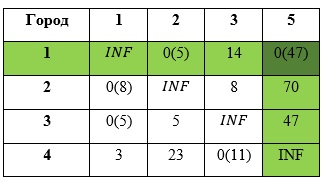
7)



8)H1=78+0 = 78

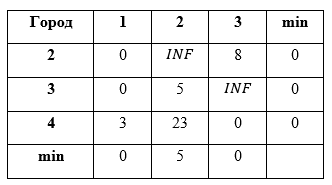
9)H1\*=78+79=156

10)

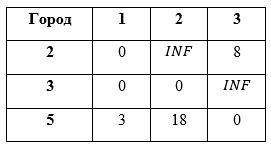


11)H2\*=78+47=125

12)

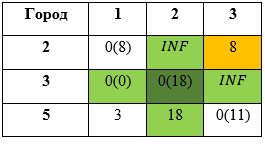


13)



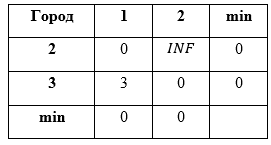
14)H2=78+5=83

15)



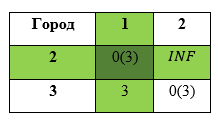
16)H3\*=83+18=101

17)



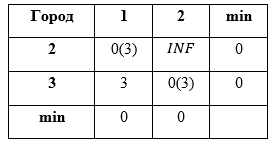
18)H3=83+0=83

19)

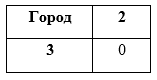


20) Н4\*=83+3=86

21)



22)H4=83+0=83



23)H5\*=б

24)H5=83+0\\\\\\\\\\\\\\\\\\\\\\

В результате мы получили следующий оптимальный маршрут: **83 (1->5->4->3->2->1)**

Граф обхода моей матрицы представлен на рисунке 1.

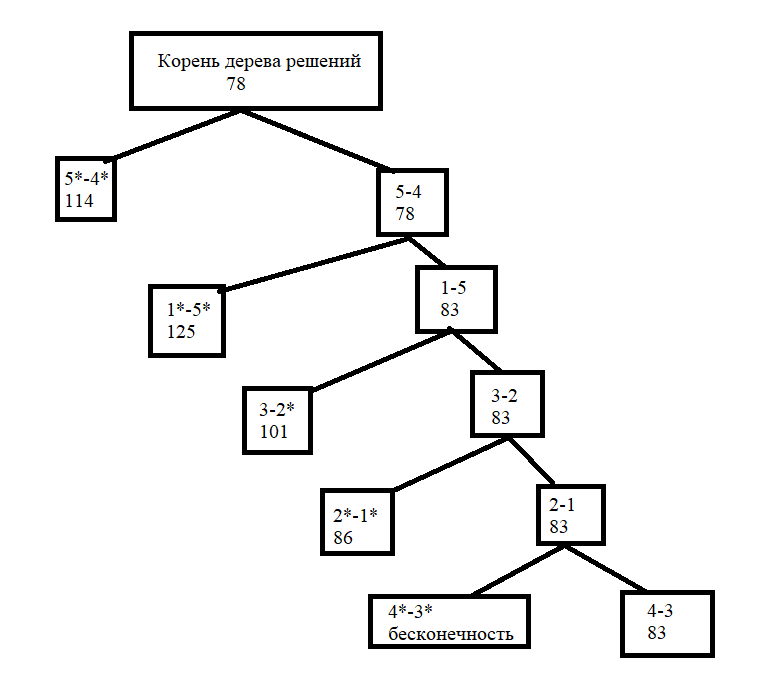


Рисунок 1 – иллюстрация метода для решения задачи

**Задание 3.**

После проверки результата при помощи генератора перестановок из лабораторной работы 2, полученные ответы совпадали с исходным решением. Демонстрация выполнения программы представлена на рисунке 2.

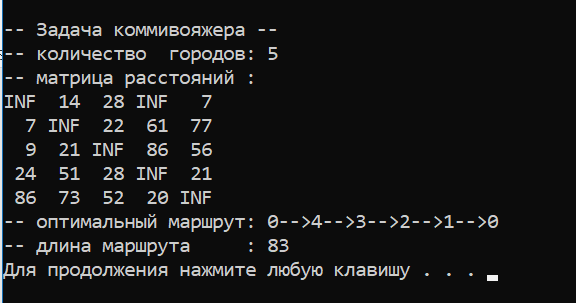


Рисунок 2 – демонстрация выполнения программы

**Вывод**: результаты после решения вручную и после использования генератора совпали.

**Лабораторная работа 4.**

Динамическое программирование. Решение задач методом динамического программирования.

**Задание 1**

На языке С++ сгенерировать случайным образом строку букв латинского алфавита S1 длиной 300 символов и S2 длиной 200.

На рисунке 1 представлен листинг кода:

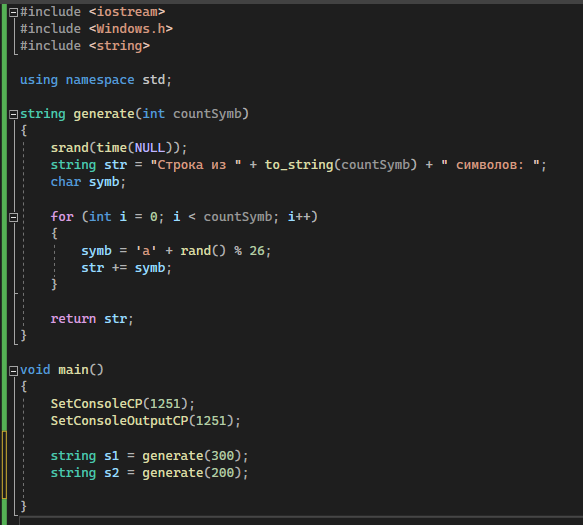


Рисунок 1 – Листинг кода генерации строк

**Задание 2**

Необходимо при помощи двух способов вычислить дистанцию Левенштейна для . Код выполнение программы представлен на рисунках ниже.

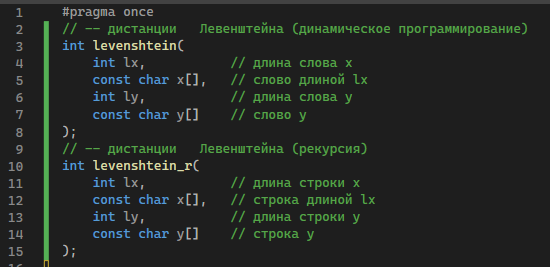


Рисунок 2 – Заголовочный файл с прототипами функций

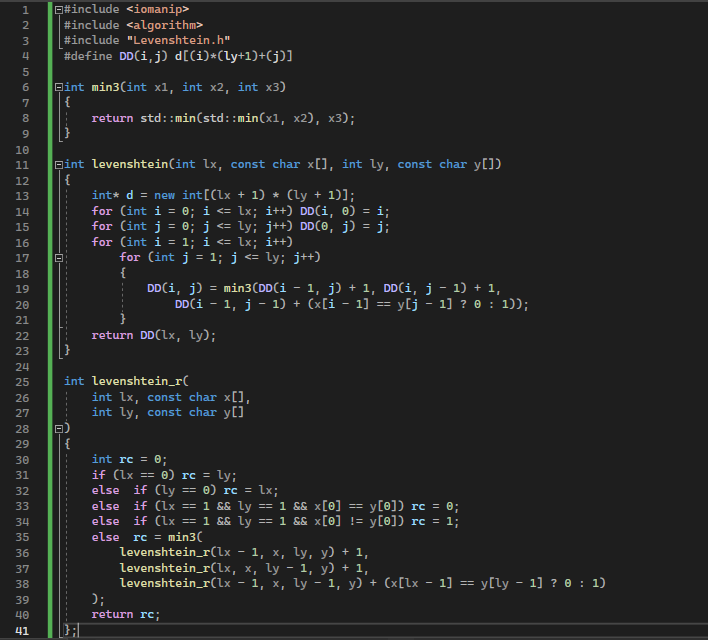


Рисунок 3 – Реализация функций

// --- main

// вычисление дистанции (расстояния) Левенштейна

#include <algorithm>

#include <iostream>

#include <ctime>

#include <iomanip>

#include "Levenshtein.h"

#include <Windows.h>

using namespace std;

char\* GenerateRandomString(int size)

{

char\* str = (char\*)malloc(sizeof(char) \* size);

for (int i = 0; i < size; i++) {

str[i] = rand() % 26 + 'a'; // 26 букв в алфавите

}

return str;

}

int main()

{

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

char\* s1 = GenerateRandomString(300);

cout << "S1: " << endl;

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

if (i % 50 == 0)

{

cout << "\n";

}

cout << s1[i];

}

cout << endl << endl;

srand(time(NULL) + 1);

char\* s2 = GenerateRandomString(200);

cout << "S2: " << endl;

for (int i = 0; i < 200; i++) {

if (i % 50 == 0)

{

cout << "\n";

}

cout << s2[i];

}

cout << endl << endl;

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

int lx = sizeof(s1);

int ly = sizeof(s2);

int s1\_size[]{ 300 / 25, 300 / 20, 300 / 15, 300 / 10, 300 / 5, 300 / 2, 300 };

int s2\_size[]{ 200 / 25, 200 / 20, 200 / 15, 200 / 10, 200 / 5, 200 / 2, 200 };

cout << "\n\n-- расстояние Левенштейна -----";

cout << "\n\n--длина --- рекурсия -- дин.програм. ---\n";

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

{

t1 = clock();

levenshtein\_r(s1\_size[i], s1, s2\_size[i], s2);

t2 = clock();

t3 = clock();

levenshtein(s1\_size[i], s1, s2\_size[i], s2);

t4 = clock();

cout << right << setw(2) << s1\_size[i] << "/" << setw(2) << s2\_size[i]

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

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

}

system("pause");

return 0;

}

Листинг 1 – функция main

Результат выполнения программы представлен на рисунке 4.

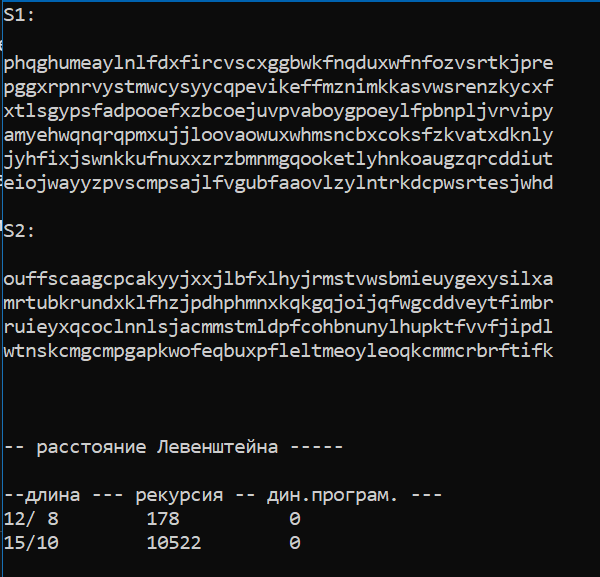


Рисунок 4 – Результат выполнения программы

**Задание 3**

Необходимо провести сравнительный анализ по затраченному на выполнение программы времени. На графике, который изображён на рисунке 5, нетрудно заметить, что использование динамического алгоритма в несколько раз эффективнее по затраченному времени, нежели рекурсивное выполнение.

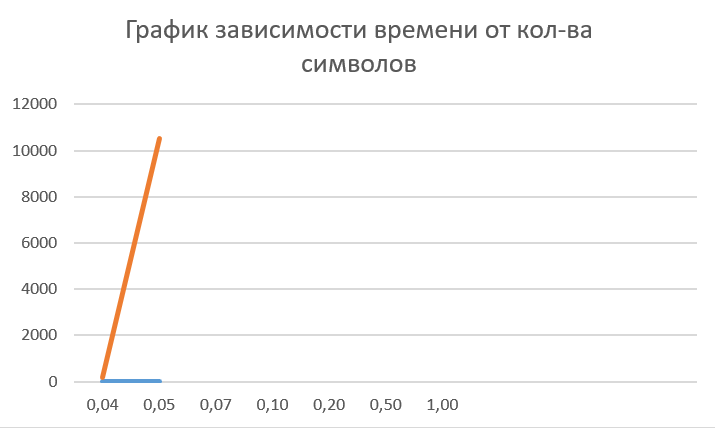


Рисунок 5 – График зависимости выполнения

**Задание 4**

Реализовать в соответствии с вариантом (рисунок 6) пример вычисления дистанции Левенштейна вручную, при помощи рекурсивного алгоритма. Ниже представлен алгоритм решения последовательно:



Рисунок 6 – Вариант для вычисления дистанции Левенштейна



= 5.

= 4.



= 4.

= 3.



= 3.

= 2.



= 2.

= 1.



= 3.

= 2.



= 2.

= 1.



= 1.

= 1.

= 0.

**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 в соответствии с вариантом. Построить графики зависимости времени вычисления от . **Отобразить ход решения в отчете**(по примеру из лекции) + код и копии экрана.

Листинги кода представлены ниже:

#pragma once

// -- 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

);

int lcsd(

const char x[], // последовательность X

const char y[], // последовательность Y

char z[] // наибольшая общая подпоследовательность

);

Листинг 2 – LCS.h

#include <algorithm>

#include "LCS.h"

#include <cstring>

#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 };

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

}

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

Листинг 3 - LCS.cpp

#include <iostream>

#include "LCS.h"

int main()

{

setlocale(LC\_ALL, "rus");

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

t1 = clock();

char X[] = "QVTWNHO", Y[] = "RQTWYK";

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

"QVTWNHO", // последовательность X

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

"RQTWYK" // последовательность Y

);

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

t2 = clock();

/// <summary>

/// </summary>

t3 = clock();

char z[100] = "";

char x[] = "QVTWNHO",

y[] = "RQTWYK";

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

std::cout << std::endl

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

<< "программирование)" << std::endl;

t4 = clock();

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;

std::cout << '\n' << "Время выполнения рекурсивно: " << t2 - t1;

std::cout << '\n' << "Время выполнения динамичически: " << t4 - t3 << '\n' << std::endl;

system("pause");

return 0;

}

Листинг 4 – Main.cpp

Выполнив сравнительный анализ времени затраченного на решение задачи о наибольшей общей подпоследовательности для двух методов решения, можно заметить, что динамический алгоритм работает быстрее, однако, в данном ситуации оба метода затрачивают для выполнения минимально времени. Результаты представлены на рисунке 7:

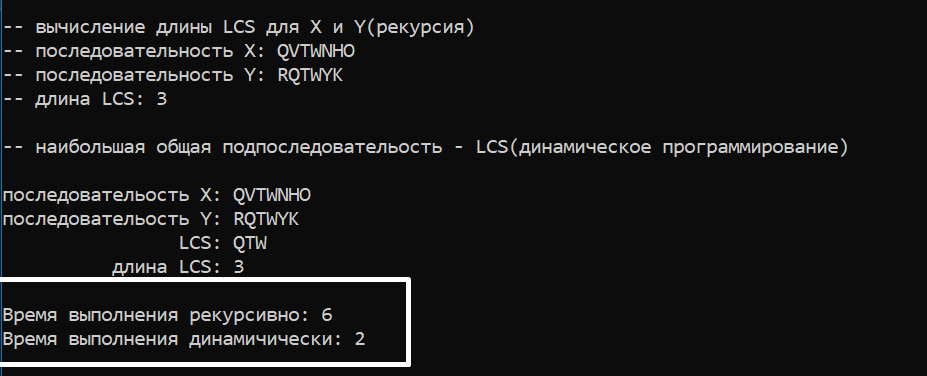


Рисунок 7 – выполнение программы по поиску наибольшей подпоследовательности

На рисунке 8 представлен график зависимости от суммарной длинны двух подстрок и необходимом времени выполнения при помощи динамического алгоритма:

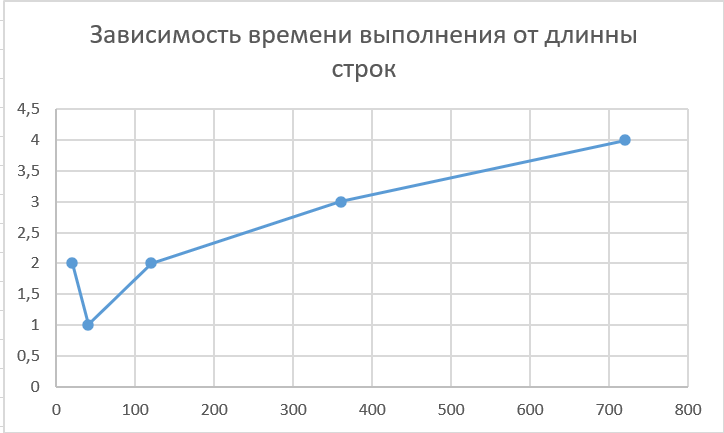


Рисунок 8 – График зависимости

**Алгоритм прохождения:**

1. Создаем двумерный массив размером (len(QVTWNHO) + 1) на (len(RQTWYK) + 1), где каждый элемент равен 0.

2. Проходим по каждой строке и столбцу массива, начиная с первого и до последнего.

3. Если значение строки или столбца равно 0, то элемент массива на пересечении этой строки и столбца остается равным 0.

4. Если символ строки и столбца равны, то элемент массива на пересечении этой строки и столбца равен значению элемента на предыдущей диагонали плюс 1, т.е. с\_(𝑛,𝑝) = c\_(𝑛−1,𝑝−1) + 1.

5. Если символ строки и столбца не равны, то элемент массива на пересечении этой строки и столбца равен максимуму между значением элемента на предыдущей строке и значением элемента на предыдущем столбце, т.е. c\_(𝑛,𝑝) = max⁡( с\_(𝑛,𝑝"−1" ), c\_(𝑛−1,𝑝)).

6. После прохода по всем элементам массива, наибольшая общая подпоследовательность будет равна значению элемента на пересечении последней строки и последнего столбца.

**Вывод:** динамический подход к решению задач позволяет выполнять их значительно быстрее, чем рекурсивный, особенно это будет заметно при решении задач с большим объёмом информации. Так же, я сделал следующие выводы:

1. Динамическое программирование - это мощный метод решения задач, позволяющий эффективно решать широкий спектр задач, которые не могут быть решены простыми алгоритмами.

2. Метод динамического программирования заключается в разбиении сложной задачи на более простые подзадачи, решение которых затем комбинируется в общее решение задачи.

3. Решение задач методом динамического программирования отличается высокой скоростью выполнения благодаря использованию кэширования вычислений и быстрой обработке данных.

4. В ходе выполнения лабораторной работы был исследован один из наиболее популярных методов динамического программирования - расстояние Левенштейна. Результаты экспериментов показали, что данная техника может быть очень эффективна для решения задач, связанных с обработкой текстовых данных.

5. Решение задач методом динамического программирования может быть осуществлено не только с помощью программирования на языке C++, но и на других языках, таких как Python или Java.