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):

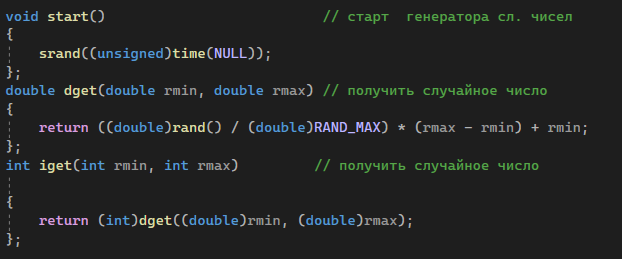


Рисунок 1 – Реализация функций

**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.

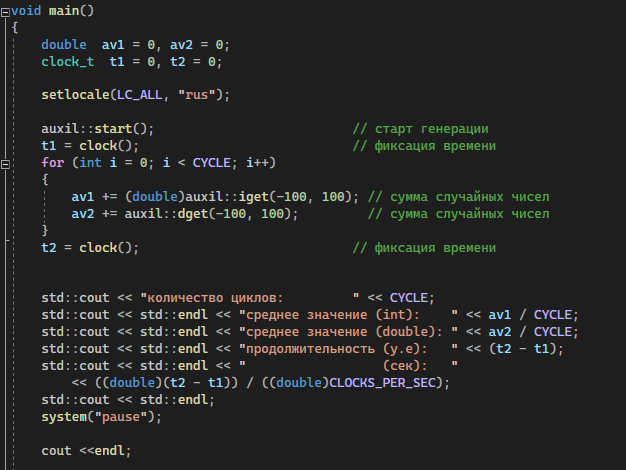


Рисунок 2 – вызов функции и измерение времени выполнения

**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

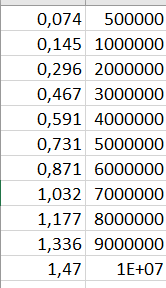
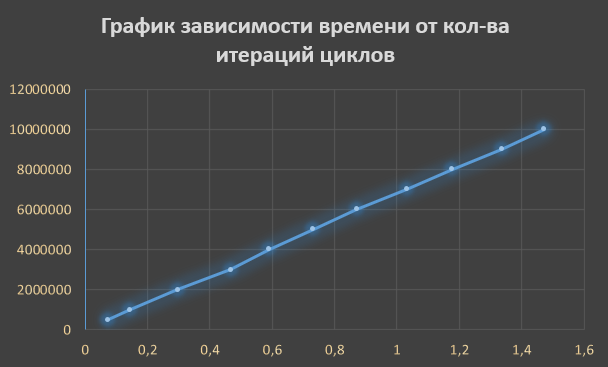


Рисунок 3 – график зависимости

Реализация Факториала в коде демонстрируется на рис. 4

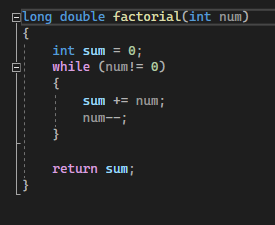


Рисунок 4 – функция для нахождения факториала

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

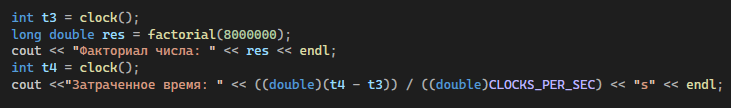


Рисунок 5 – вызов функции для подсчёта факториала

График зависимости:

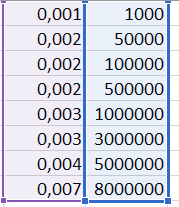
 

Рисунок 6 – график зависимости

**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.

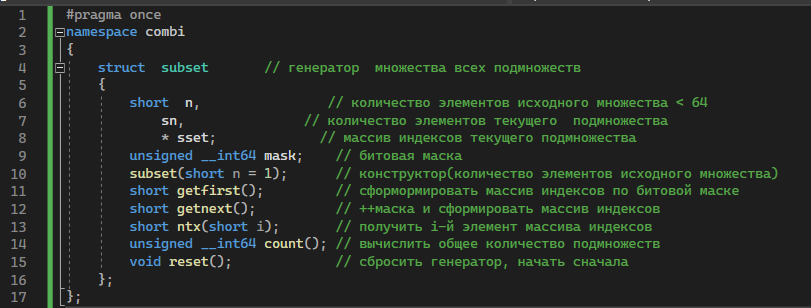


Рисунок 1 – Combi.h

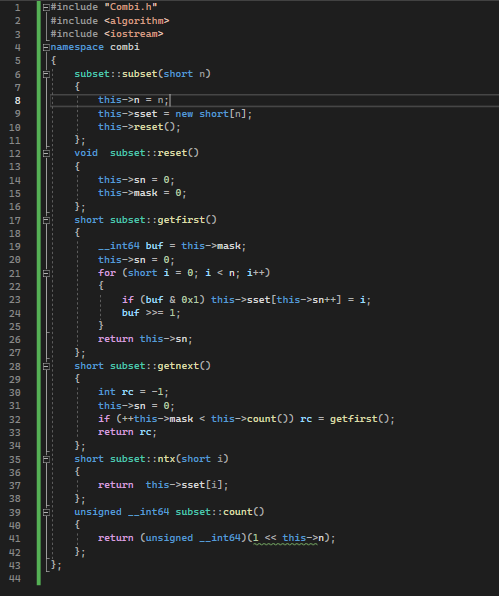


Рисунок 2 - Combi.cpp

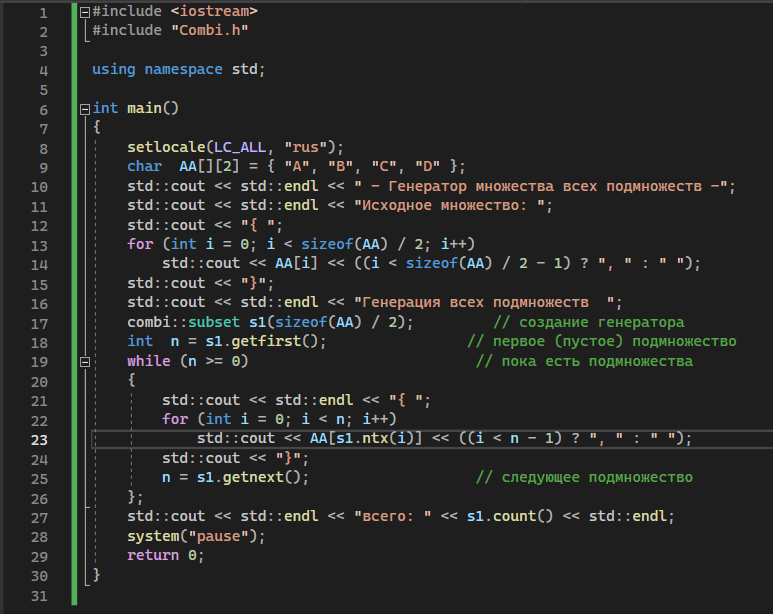


Рисунок 3 - Main\_ex1.cpp

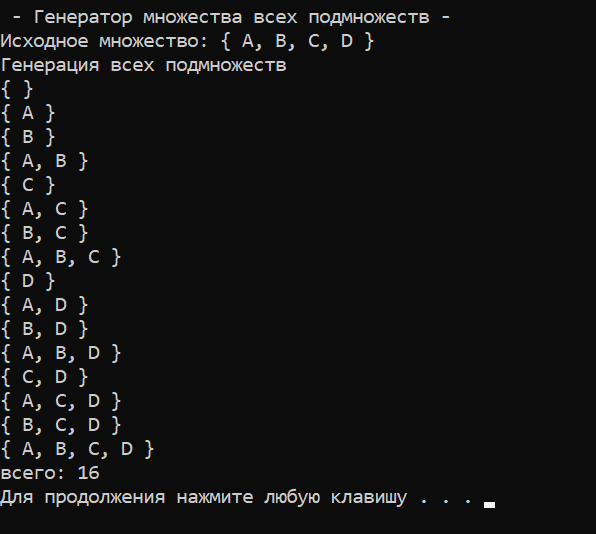


Рисунок 4 – Результат выполнения

**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.

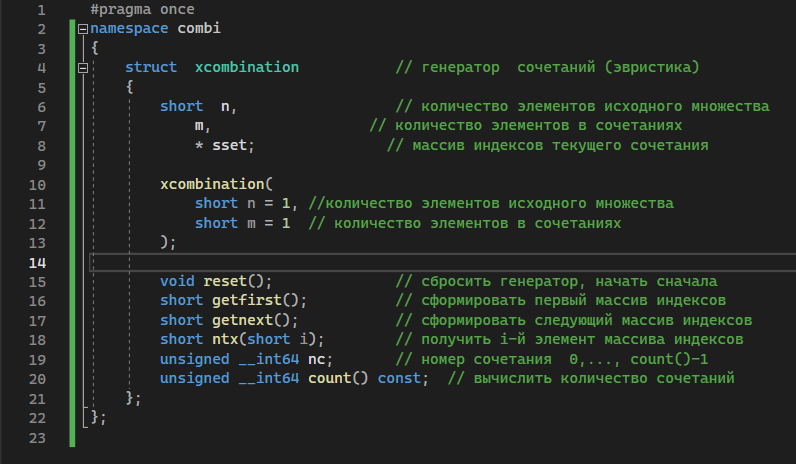


Рисунок 5 - Combi.h

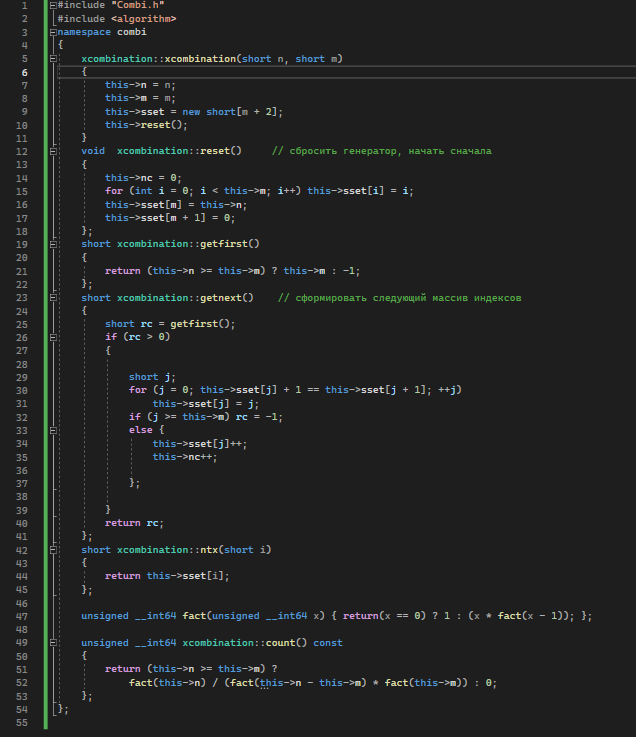


Рисунок 6 - Combi.cpp

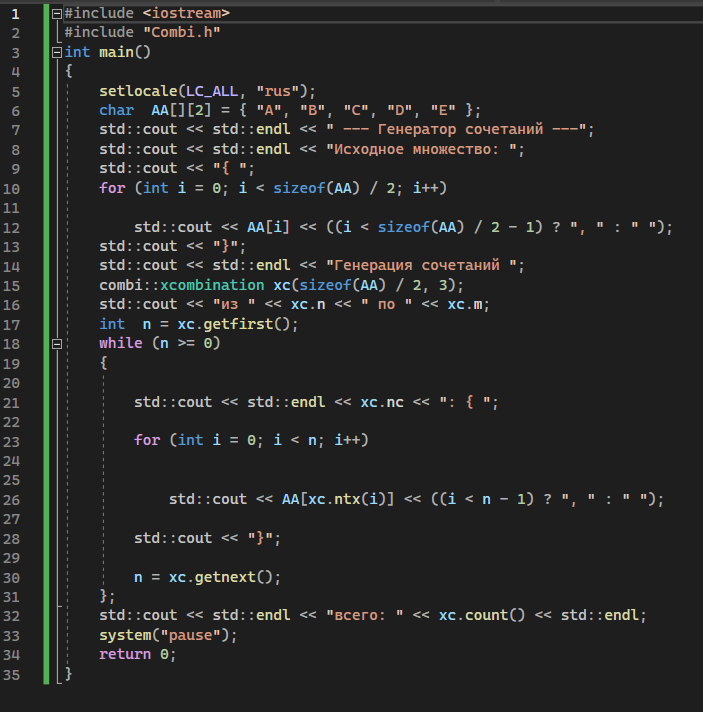


Рисунок 7 - Main\_ex2.cpp

Результат выполнения представлен на рисунке 8.

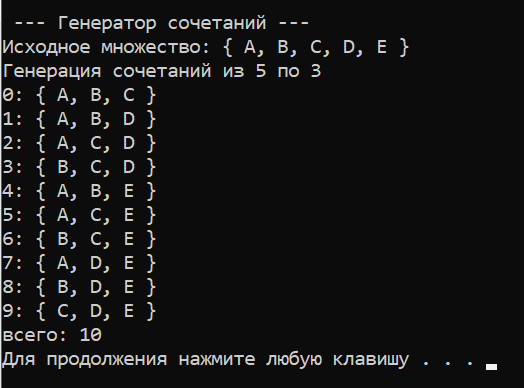


Рисунок 8 – результат выполнения программы

**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.

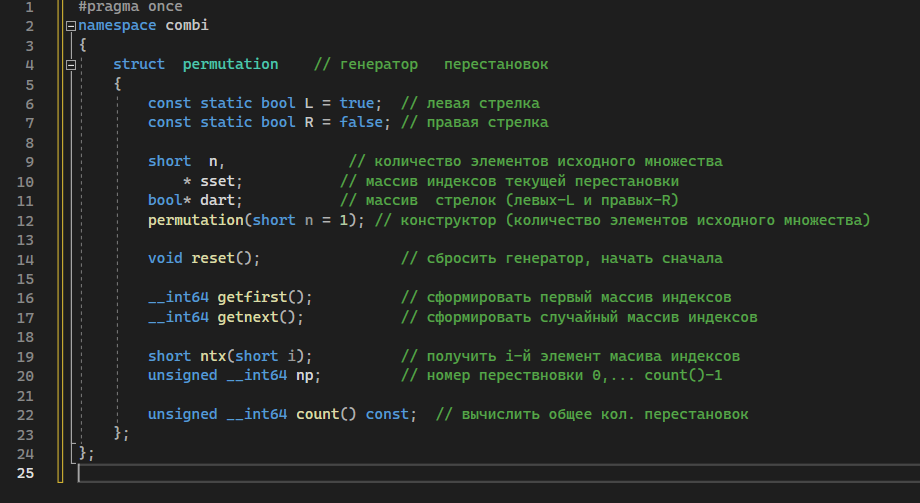


Рисунок 9 – Combi.h

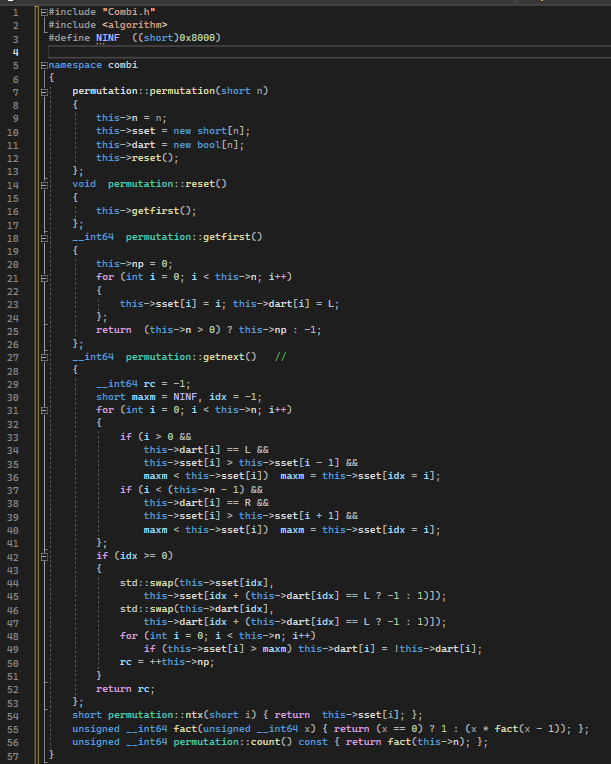


Рисунок 10 – Combi.cpp

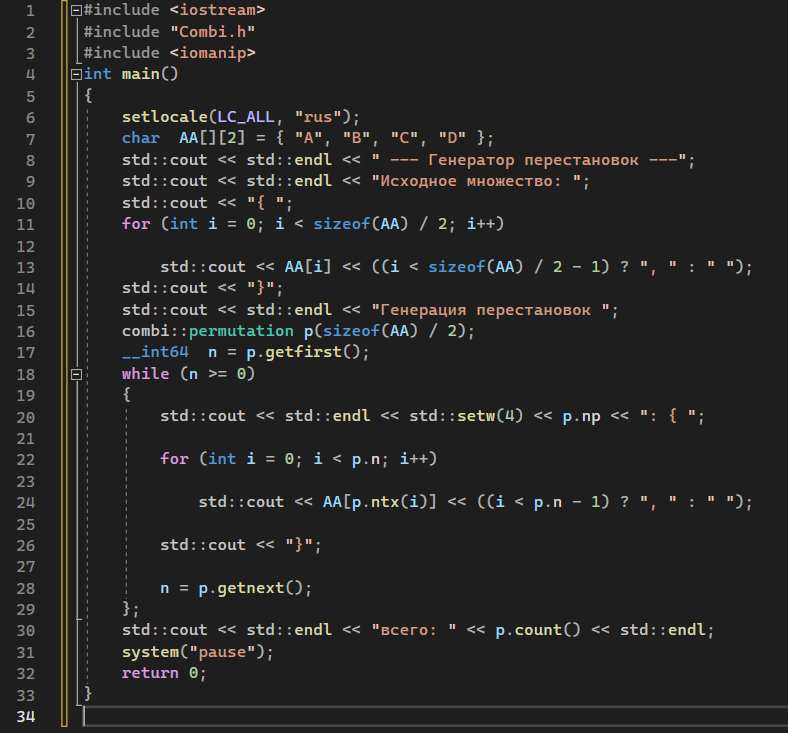


Рисунок 11 – Main\_ex3.cpp

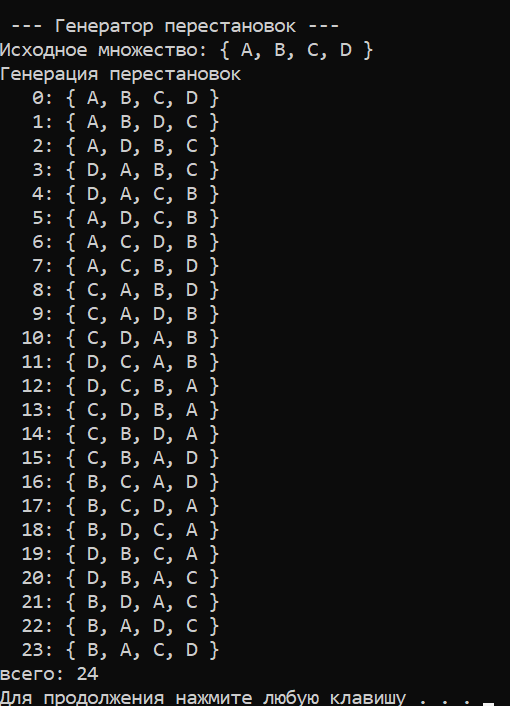


Рисунок 12 – результат выполнения программы

**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.

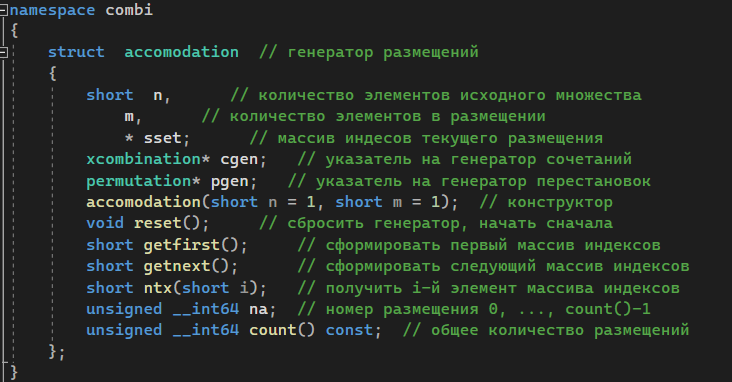


Рисунок 13 – Combi.h



Рисунок 14 – Combi.cpp

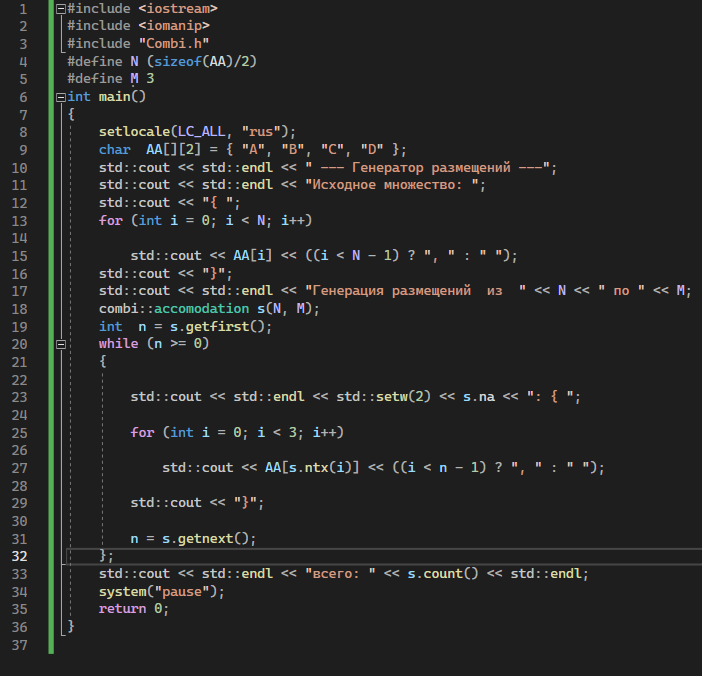


Рисунок 15 – Main\_ex4.cpp

Результат выполнения программы представлен на рисунке 16.

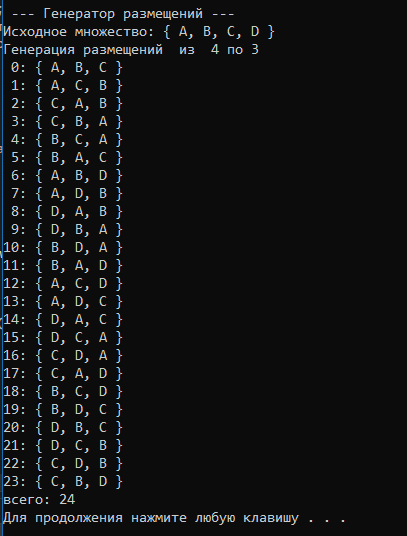


Рисунок 16 - результат выполнения программы

**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.);

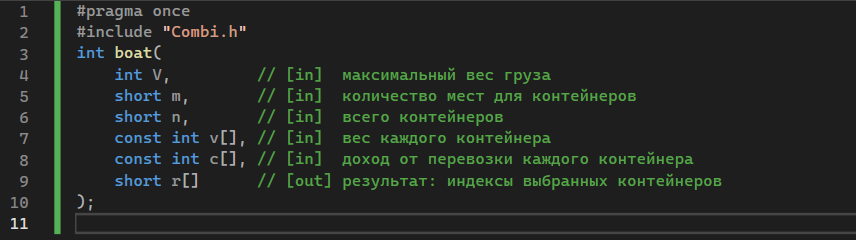


Рисунок 17 – Boat.h

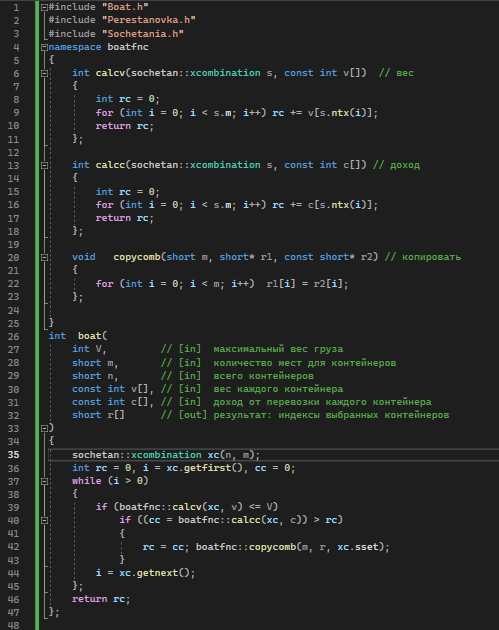


Рисунок 18 – Boat.cpp

Результат выполнения программы представлен на рисунке 19.

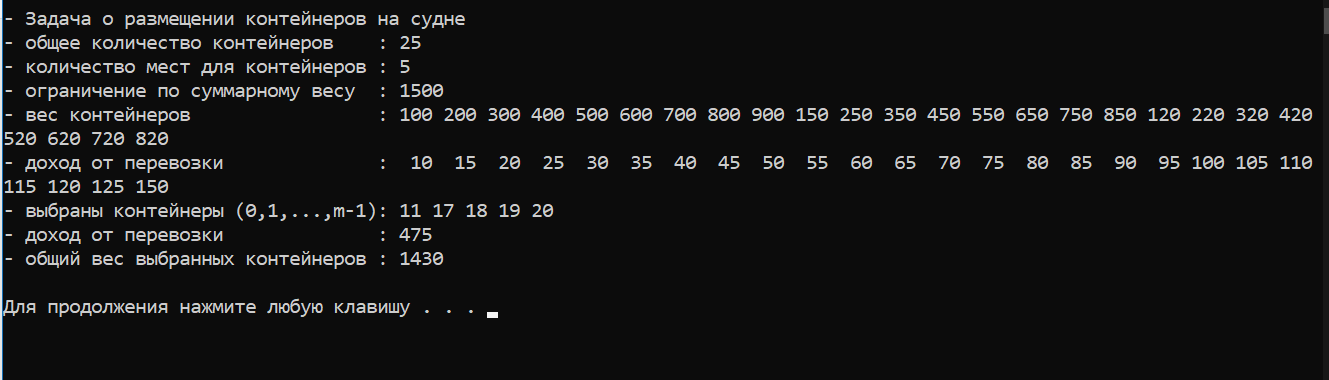


Рисунок 19 – результат работы программы

**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.

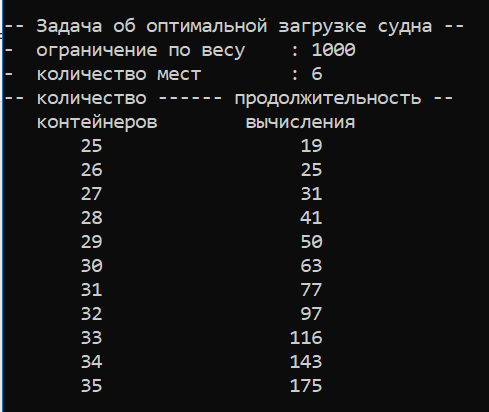


Рисунок 20 – Зависимость времени выполнения от количества контейнеров



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.

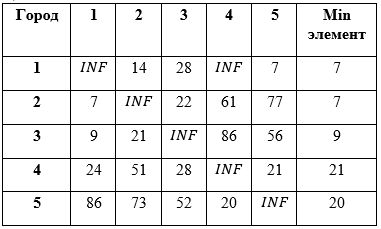
Более конкретно, параметр определяет максимально допустимый вес ребра в гамильтоновом цикле. Если в графе есть несколько ребер, вес которых больше параметра, то цикл, содержащий любое из этих ребер, не будет оптимальным.

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.

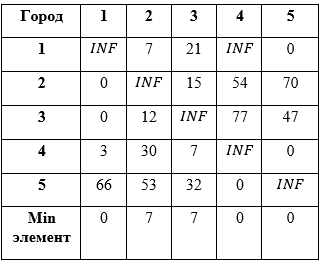
**Задание 2.**

Алгоритм решения (картинки идут последовательно, нумерация отсутствует):

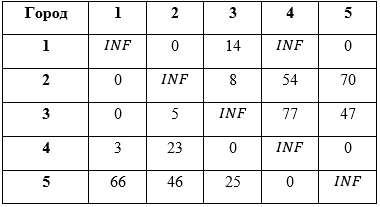
1)



2)

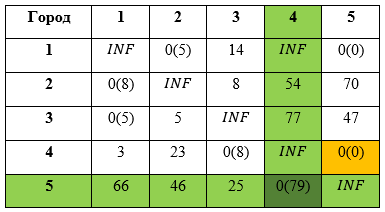


3)

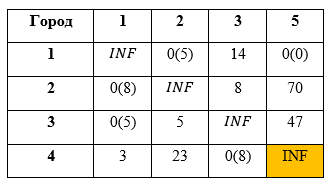


4)Н0 = 2\*7+9+21+20+14=78

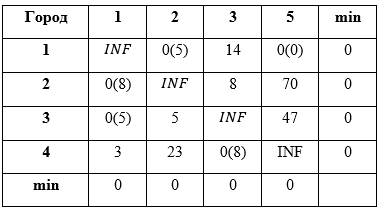
5)



6)



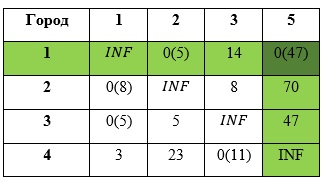
7)



8)H1=78+0 = 78

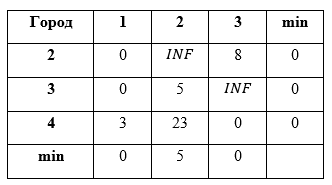
9)H1\*=78+79=156

10)

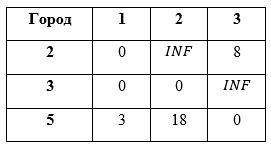


11)H2\*=78+47=125

12)

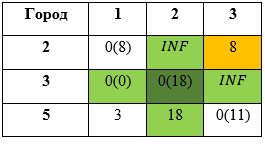


13)



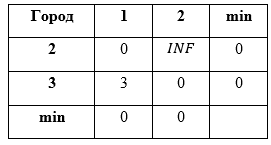
14)H2=78+5=83

15)



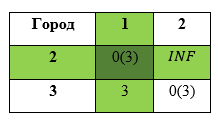
16)H3\*=83+18=101

17)



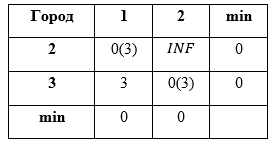
18)H3=83+0=83

19)

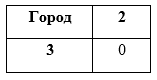


20) Н4\*=83+3=86

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.

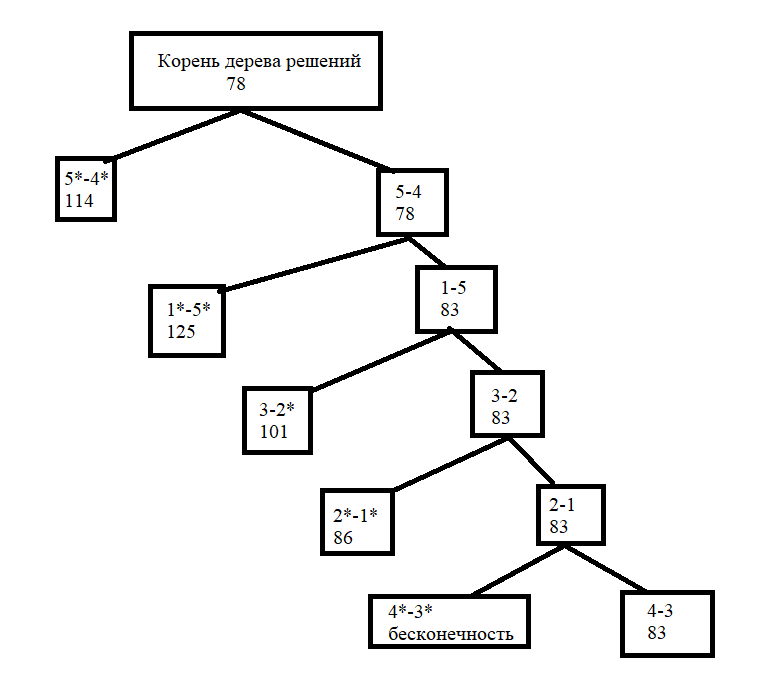


Рисунок 1 – иллюстрация метода для решения задачи

**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.

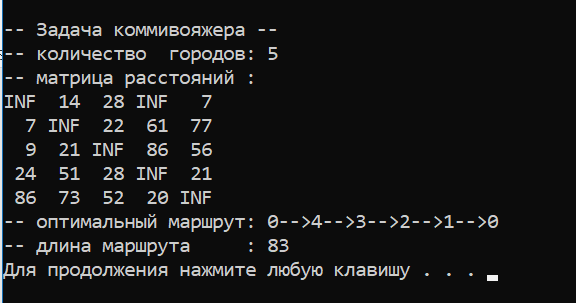


Рисунок 2 – демонстрация выполнения программы

**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:

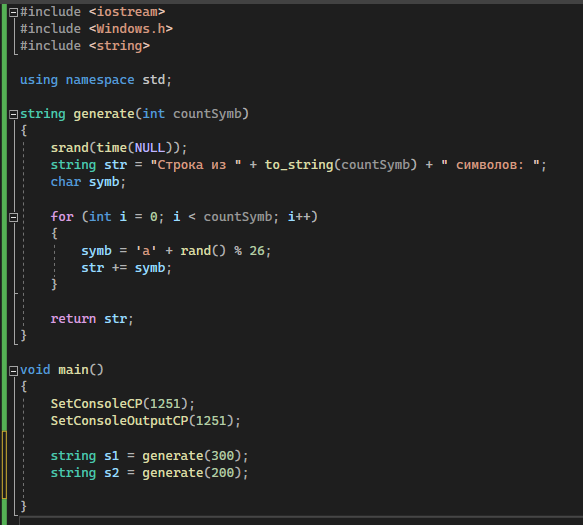


Рисунок 1 – Листинг кода генерации строк

**Задание 2**

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

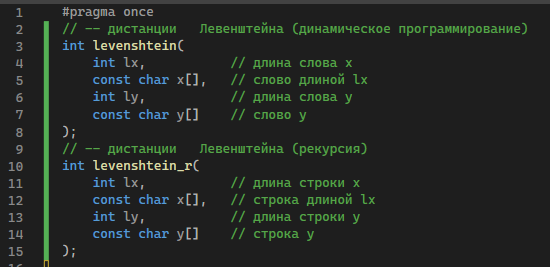


Рисунок 2 – Заголовочный файл с прототипами функций

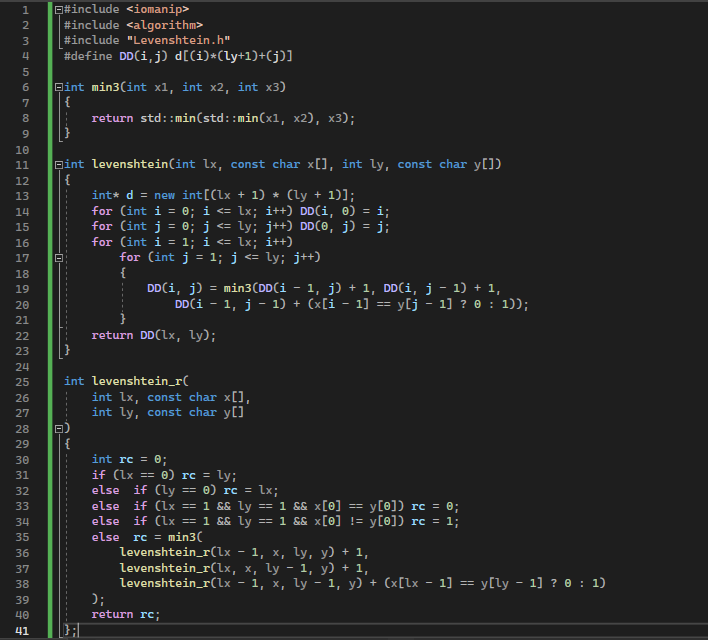


Рисунок 3 – Реализация функций

// --- 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.

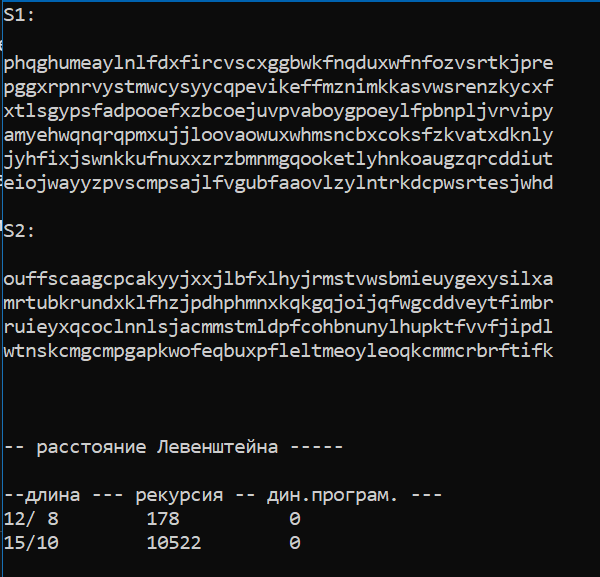


Рисунок 4 – Результат выполнения программы

**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.

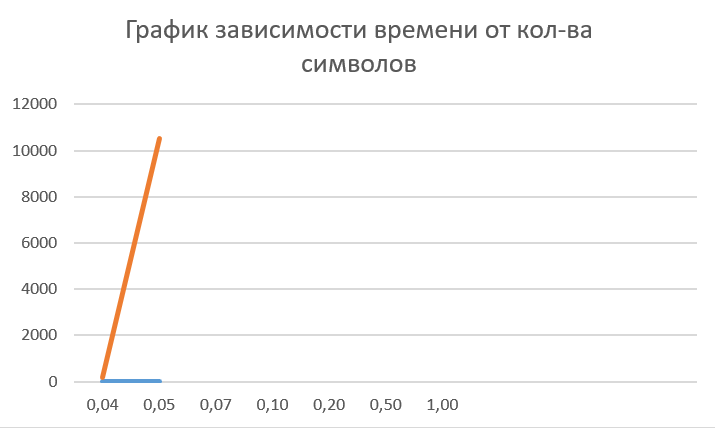


Рисунок 5 – График зависимости выполнения

**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.

1. L(“K”, “C”) = min(2,2,1) = 1
2. L(“Kо”, “C”) = min(2,3,2) = 2
3. L(“Kол”, “C) = min(4,3,3) = 3
4. L(“K”, “Cт”) = min(3,2,2) = 2
5. L(“Kо”, “Cт”) = min(3,3,2) = 2
6. L(“Kол”, “Cт”) = min(3,4,3) = 3
7. L(“K”, “Cто”) = min(4,3,3) = 3
8. L(“Kо”, “Cто”) = min(4,3,3) = 3
9. L(“Kол”, “Cто”) = min(4,4,3) = 3
10. L(“K”, “Cтол”) = min(5,4,4) = 4
11. L(“K”, “Cтолб”) = min(6,5,5) = 5
12. L(“Kо”, “Cтол”) = min(5,4,4) = 4
13. L(“Kол”, “Cтол”) = min(5,4,4) = 4
14. L(“Kо”, “Cтолб”) = min(6,5,5) = 5
15. L(“Kол”, “Cтолб”) = min(6,5,5) = 5

**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:

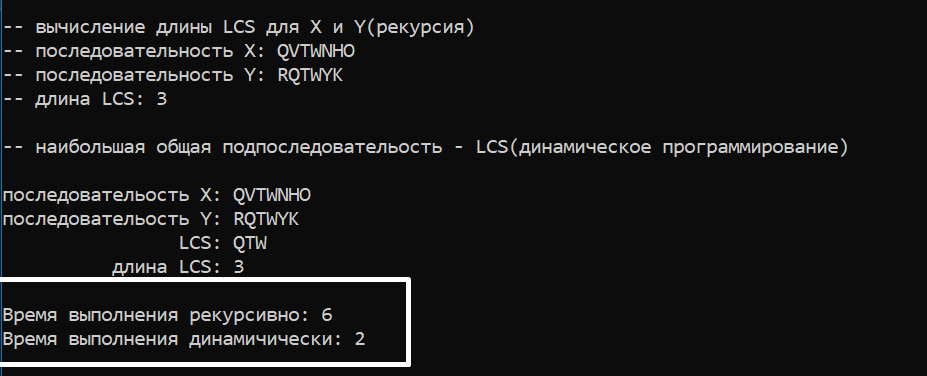


Рисунок 7 – выполнение программы по поиску наибольшей подпоследовательности

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

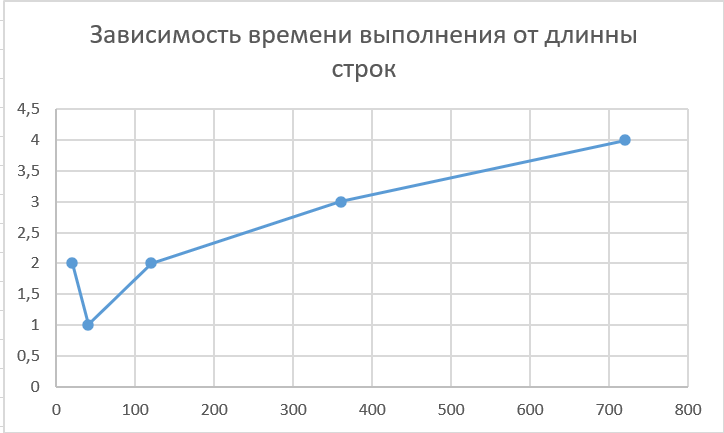


Рисунок 8 – График зависимости

**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.

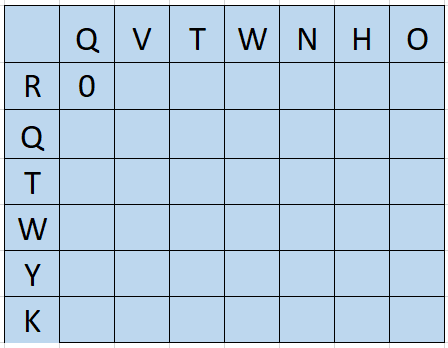
4. Если символ строки и столбца равны, то элемент массива на пересечении этой строки и столбца равен значению элемента на предыдущей диагонали плюс 1, т.е. с\_(𝑛,𝑝) = c\_(𝑛−1,𝑝−1) + 1.

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)).

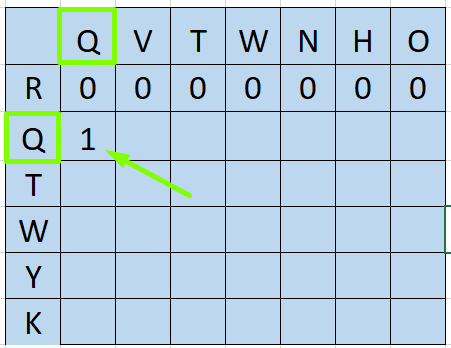
6. После прохода по всем элементам массива, наибольшая общая подпоследовательность будет равна значению элемента на пересечении последней строки и последнего столбца.

Алгоритм прохода будет выглядеть следующим образом:

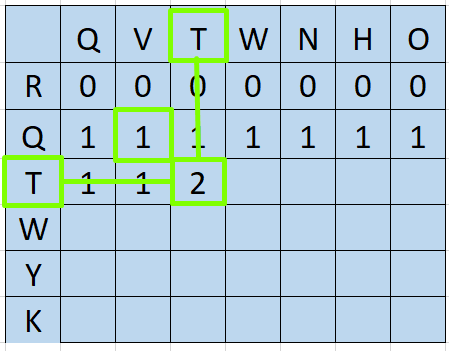
1)



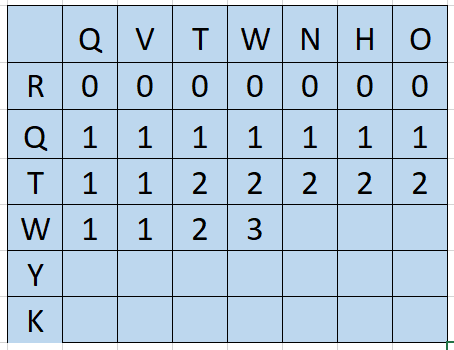
2)



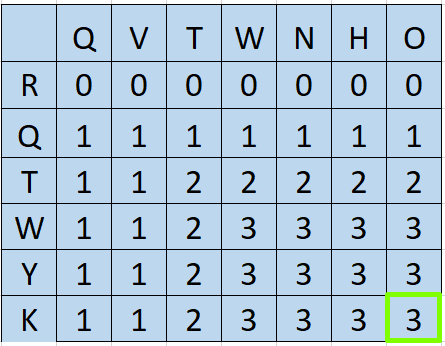
3)



4)



5)



**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.

**Лабораторная работа 5. ТРАНСПОРТНАЯ ЗАДАЧА**

Приобретение навыков решения открытой транспортной задачи

Необходимо составить таблицу (исходя из своего варианта) для решения транспортной задачи. Таблица представлена на таблице 1.

Таблица 1 – Исходная таблица для решения транспортной задачи

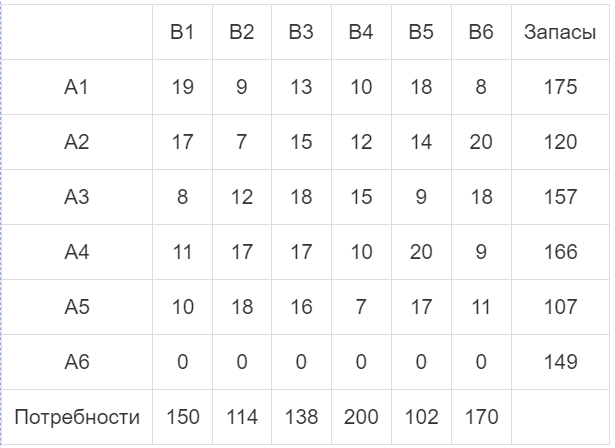
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| ПОТРЕБИТЕЛИ  ПОСТАВЩИКИ | 1 | 2 | 3 | 4 | 5 | 6 | ЗАПАСЫ |
| 1 | **19** | **9** | **13** | **10** | **18** | **8** | **175** |
| 2 | **17** | **7** | **15** | **12** | **14** | **20** | **120** |
| 3 | **8** | **12** | **18** | **15** | **9** | **18** | **157** |
| 4 | **11** | **17** | **17** | **10** | **20** | **9** | **166** |
| 5 | **10** | **18** | **16** | **7** | **17** | **11** | **107** |
| ПОТРЕБНОСТИ | **150** | **114** | **138** | **200** | **102** | **170** |  |

**Ход решения:**

* Проверить, открытая задача или закрытая;
* Составить опорный план;
* Применить метод потенциалов;

Решим задачу. Ниже будет приведен алгоритм решения задачи для таблицы 1:

Проверим необходимое и достаточное условие разрешимости задачи.  
∑a = 175 + 120 + 157 + 166 + 107 = 725  
∑b = 150 + 114 + 138 + 200 + 102 + 170 = 874  
Как видно, суммарная потребность груза в пунктах назначения превышает запасы груза на базах. Следовательно, модель исходной транспортной задачи является открытой. Чтобы получить закрытую модель, введем дополнительную (фиктивную) базу с запасом груза, равным 149 (725—874). Тарифы перевозки единицы груза из базы ко всем потребителям полагаем равны нулю.  
Занесем исходные данные в распределительную таблицу.



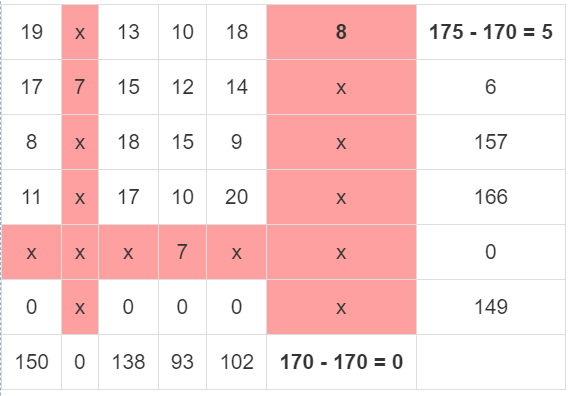
**Этап I. Поиск первого опорного плана**.  
1. Используя *метод наименьшей стоимости*, построим первый опорный план транспортной задачи.  
Суть метода заключается в том, что из всей таблицы стоимостей выбирают наименьшую, и в клетку, которая ей соответствует, помещают меньшее из чисел ai, или bj.  
Затем, из рассмотрения исключают либо строку, соответствующую поставщику, запасы которого полностью израсходованы, либо столбец, соответствующий потребителю, потребности которого полностью удовлетворены, либо и строку и столбец, если израсходованы запасы поставщика и удовлетворены потребности потребителя.  
Из оставшейся части таблицы стоимостей снова выбирают наименьшую стоимость, и процесс распределения запасов продолжают, пока все запасы не будут распределены, а потребности удовлетворены.  
Искомый элемент равен c22=7. Для этого элемента запасы равны 120, потребности 114. Поскольку минимальным является 114, то вычитаем его.  
x22 = min(120,114) = 114.



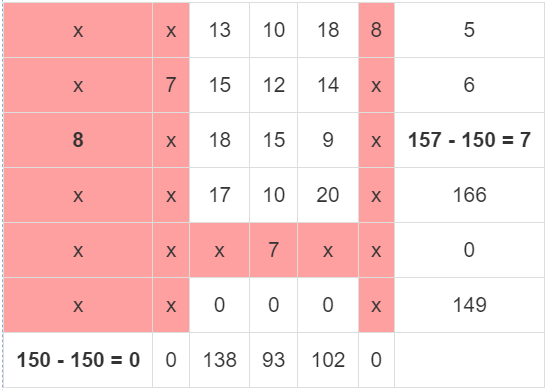
Искомый элемент равен c54=7. Для этого элемента запасы равны 107, потребности 200. Поскольку минимальным является 107, то вычитаем его.  
x54 = min(107,200) = 107.



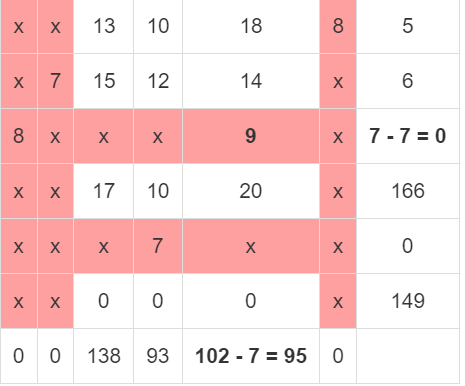
Искомый элемент равен c16=8. Для этого элемента запасы равны 175, потребности 170. Поскольку минимальным является 170, то вычитаем его.  
x16 = min(175,170) = 170.



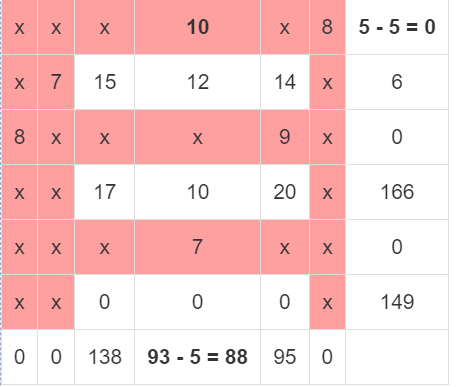
Искомый элемент равен c31=8. Для этого элемента запасы равны 157, потребности 150. Поскольку минимальным является 150, то вычитаем его.  
x31 = min(157,150) = 150.



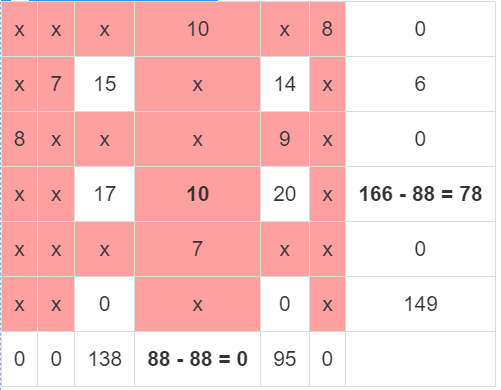
Искомый элемент равен c35=9. Для этого элемента запасы равны 7, потребности 102. Поскольку минимальным является 7, то вычитаем его.  
x35 = min(7,102) = 7.



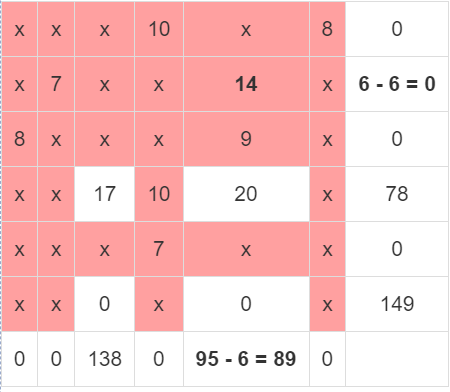
Искомый элемент равен c14=10. Для этого элемента запасы равны 5, потребности 93. Поскольку минимальным является 5, то вычитаем его.  
x14 = min(5,93) = 5.



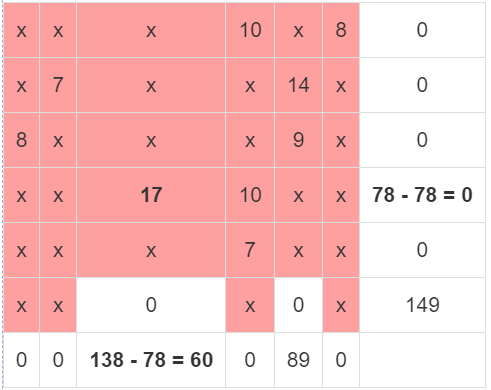
Искомый элемент равен c44=10. Для этого элемента запасы равны 166, потребности 88. Поскольку минимальным является 88, то вычитаем его.  
x44 = min(166,88) = 88.



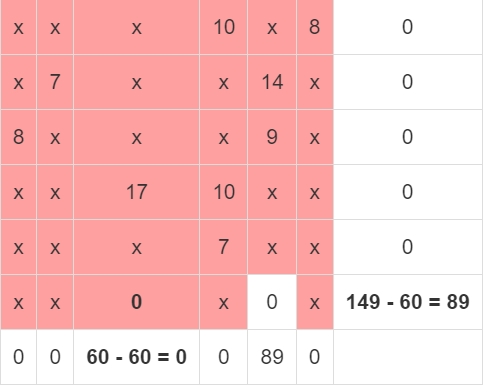
Искомый элемент равен c25=14. Для этого элемента запасы равны 6, потребности 95. Поскольку минимальным является 6, то вычитаем его.  
x25 = min(6,95) = 6.



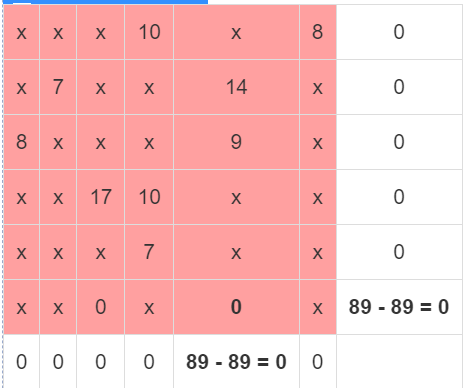
Искомый элемент равен c43=17. Для этого элемента запасы равны 78, потребности 138. Поскольку минимальным является 78, то вычитаем его.  
x43 = min(78,138) = 78.

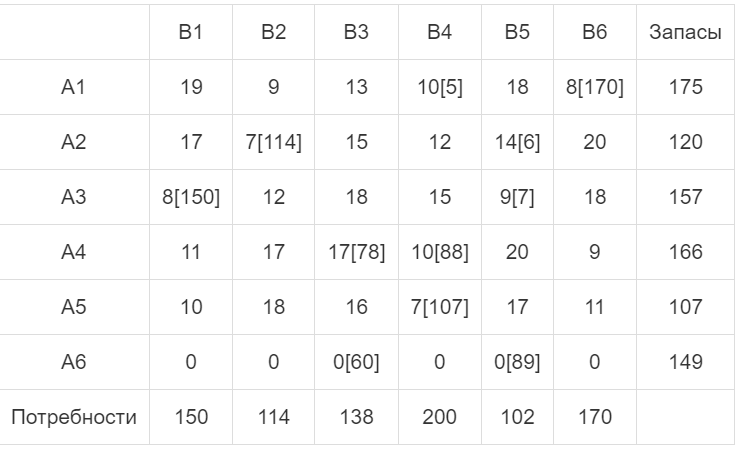


Искомый элемент равен c63=0. Для этого элемента запасы равны 149, потребности 60. Поскольку минимальным является 60, то вычитаем его.  
x63 = min(149,60) = 60.



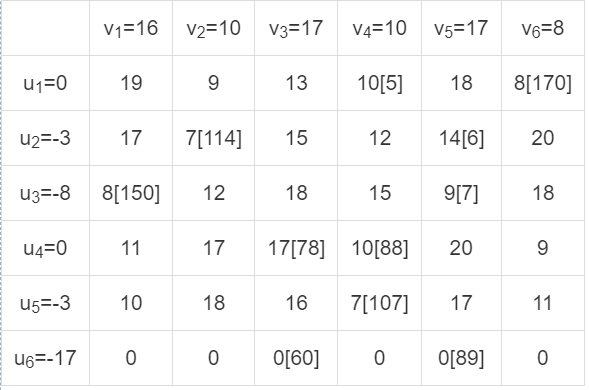
Искомый элемент равен c65=0. Для этого элемента запасы равны 89, потребности 89. Поскольку минимальным является 89, то вычитаем его.  
x65 = min(89,89) = 89.





В результате получен первый опорный план, который является допустимым, так как все грузы из баз вывезены, потребность потребителей удовлетворена, а план соответствует системе ограничений транспортной задачи.  
2. Подсчитаем число занятых клеток таблицы, их 11, а должно быть m + n - 1 = 11. Следовательно, опорный план является *невырожденным*.  
Значение целевой функции для этого опорного плана равно:  
F(x) = 10\*5 + 8\*170 + 7\*114 + 14\*6 + 8\*150 + 9\*7 + 17\*78 + 10\*88 + 7\*107 + 0\*60 + 0\*89 = 6510

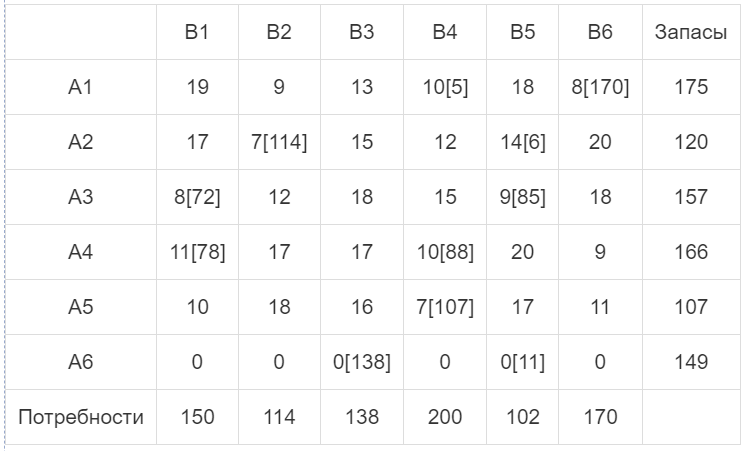
**Этап II. Улучшение опорного плана** **методом потенциалов**  
Проверим оптимальность опорного плана. Найдем *предварительные потенциалы* ui, vj. по занятым клеткам таблицы, в которых ui + vj = cij, полагая, что u1 = 0.  
u1 + v4 = 10; 0 + v4 = 10; v4 = 10  
u4 + v4 = 10; 10 + u4 = 10; u4 = 0  
u4 + v3 = 17; 0 + v3 = 17; v3 = 17  
u6 + v3 = 0; 17 + u6 = 0; u6 = -17  
u6 + v5 = 0; -17 + v5 = 0; v5 = 17  
u2 + v5 = 14; 17 + u2 = 14; u2 = -3  
u2 + v2 = 7; -3 + v2 = 7; v2 = 10  
u3 + v5 = 9; 17 + u3 = 9; u3 = -8  
u3 + v1 = 8; -8 + v1 = 8; v1 = 16  
u5 + v4 = 7; 10 + u5 = 7; u5 = -3  
u1 + v6 = 8; 0 + v6 = 8; v6 = 8



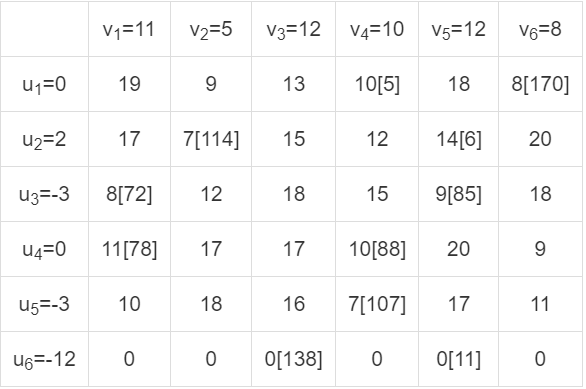
Опорный план не является оптимальным, так как существуют оценки свободных клеток, для которых ui + vj > cij  
(1;2): 0 + 10 > 9; ∆12 = 0 + 10 - 9 = 1 > 0  
(1;3): 0 + 17 > 13; ∆13 = 0 + 17 - 13 = 4 > 0  
(4;1): 0 + 16 > 11; ∆41 = 0 + 16 - 11 = 5 > 0  
(5;1): -3 + 16 > 10; ∆51 = -3 + 16 - 10 = 3 > 0  
max(1,4,5,3) = 5  
Выбираем максимальную оценку свободной клетки (4;1): 11  
Для этого в перспективную клетку (4;1) поставим знак «+», а в остальных вершинах многоугольника чередующиеся знаки «-», «+», «-».



Цикл приведен в таблице (4,1 → 4,3 → 6,3 → 6,5 → 3,5 → 3,1).  
Из грузов хij стоящих в минусовых клетках, выбираем наименьшее, т.е. у = min (4, 3) = 78. Прибавляем 78 к объемам грузов, стоящих в плюсовых клетках и вычитаем 78 из Хij, стоящих в минусовых клетках. В результате получим новый опорный план.



Проверим оптимальность опорного плана. Найдем *предварительные потенциалы* ui, vj. по занятым клеткам таблицы, в которых ui + vj = cij, полагая, что u1 = 0.  
u1 + v4 = 10; 0 + v4 = 10; v4 = 10  
u4 + v4 = 10; 10 + u4 = 10; u4 = 0  
u4 + v1 = 11; 0 + v1 = 11; v1 = 11  
u3 + v1 = 8; 11 + u3 = 8; u3 = -3  
u3 + v5 = 9; -3 + v5 = 9; v5 = 12  
u2 + v5 = 14; 12 + u2 = 14; u2 = 2  
u2 + v2 = 7; 2 + v2 = 7; v2 = 5  
u6 + v5 = 0; 12 + u6 = 0; u6 = -12  
u6 + v3 = 0; -12 + v3 = 0; v3 = 12  
u5 + v4 = 7; 10 + u5 = 7; u5 = -3  
u1 + v6 = 8; 0 + v6 = 8; v6 = 8



Опорный план является оптимальным, так все оценки свободных клеток удовлетворяют условию ui + vj ≤ cij.  
Минимальные затраты составят: F(x) = 10\*5 + 8\*170 + 7\*114 + 14\*6 + 8\*72 + 9\*85 + 11\*78 + 10\*88 + 7\*107 + 0\*138 + 0\*11 = 6120

**Анализ оптимального плана**.  
Из 1-го склада необходимо груз направить к 4-у потребителю (5 ед.), к 6-у потребителю (170 ед.)  
Из 2-го склада необходимо груз направить к 2-у потребителю (114 ед.), к 5-у потребителю (6 ед.)  
Из 3-го склада необходимо груз направить к 1-у потребителю (72 ед.), к 5-у потребителю (85 ед.)  
Из 4-го склада необходимо груз направить к 1-у потребителю (78 ед.), к 4-у потребителю (88 ед.)  
Из 5-го склада необходимо весь груз направить к 4-у потребителю.  
Потребность 3-го потребителя остается неудовлетворенной на 138 ед.  
Оптимальный план является вырожденным, так как базисная переменная x63=0.

Вывод: Потребность 5-го потребителя остается неудовлетворенной на 11 ед.  
Оптимальный план является вырожденным, так как базисная переменная x65=0.

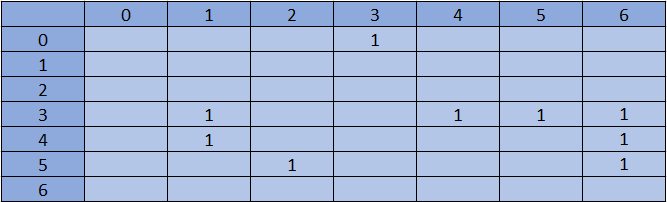
**Лабораторная работа 6. АЛГОРИТМЫ НА ГРАФАХ**

Освоить сущность и программную реализацию: а) способов представления графов; б) алгоритмов поиска в ширину и глубину; в) алгоритма топологической сортировки графов. Разобрать алгоритм Прима и алгоритм Крускала

**Задание 1.** Ориентированный граф **G** взять в соответствии с вариантом. Представить его в отчете в виде матрицы смежности, матрицы инцидентности, списка смежных вершин.



Матрица смежности:

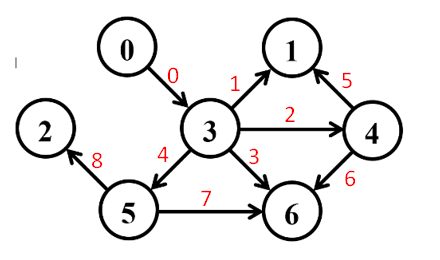


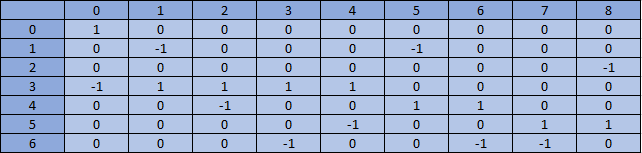
Список смежных вершин:

0 {3} 1 {-} 2 {-} 3 {1, 4, 5, 6} 4 {1, 6}

5 {2, 6}

Матрица инцидентности





***Задание 2.*** Осуществить алгоритмы поиска в ширину и глубину, а также алгоритма топологической сортировки аналогично примерам, рассмотренным на лекциях. Оформить отчет, включив в него **каждый** шаг выполнения алгоритмов.

***1.Поиск в ширину***



1.Посещённые вершины: {}

Очередь: {0}

Текущая вершина: {}

2.Посещённые вершины: {0}

Очередь: {}

Текущая вершина: {0}

3.Посещённые вершины: {0}

Очередь: {3}

Текущая вершина: {0}

4.Посещённые вершины: {0, 3}

Очередь: {1, 4, 5, 6}

Текущая вершина: {3}

5.Посещённые вершины: {0, 3, 1}

Очередь: {4, 5, 6}

Текущая вершина: {1}

6.Посещённые вершины: {0, 3, 1, 4}

Очередь: {5, 6}

Текущая вершина: {4}

7.Посещённые вершины: {0, 3, 1, 4, 5}

Очередь: {6, 2}

Текущая вершина: {5}

8.Посещённые вершины: {0, 3, 1, 4, 5, 6}

Очередь: {2}

Текущая вершина: {6}

9.Посещённые вершины: {0, 3, 1, 4, 5, 6, 2}

Очередь: {}

Текущая вершина: {2}

***2.Поиск в глубину***



1.Посещённые вершины: {}

Стек: {0}

Текущая вершина: {}

2.Посещённые вершины: {0}

Стек: {3}

Текущая вершина: {0}

3.Посещённые вершины: {0, 3}

Стек: {6, 5, 4, 1}

Текущая вершина: {3}

4.Посещённые вершины: {0, 3, 6}

Стек: {5, 4, 1}

Текущая вершина: {6}

5.Посещённые вершины: {0, 3, 6, 5}

Стек: {2, 4, 1}

Текущая вершина: {5}

6.Посещённые вершины: {0, 3, 6, 5, 2}

Стек: {4, 1}

Текущая вершина: {2}

7.Посещённые вершины: {0, 3, 6, 5, 2, 4}

Стек: {1}

Текущая вершина: {4}

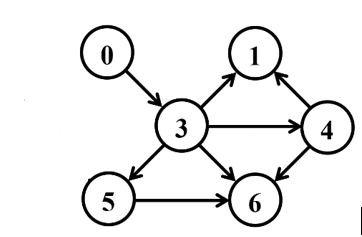
8.Посещённые вершины: {0, 3, 6, 5, 2, 4, 1}

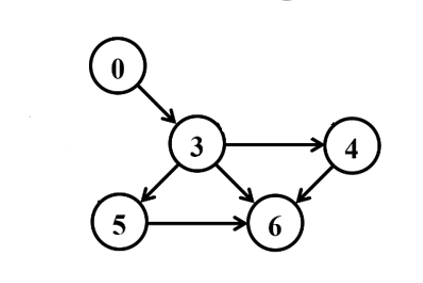
Стек: {}

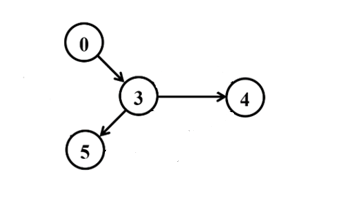
Текущая вершина: {1}

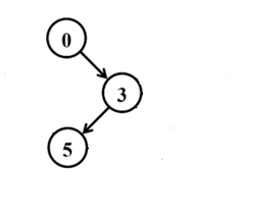
***3.Топологическая сортировка:***

2

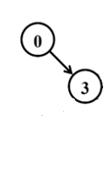
1

6

4



5



3

0

**Результат: 2-1-6-4-5-3-0**

***Задание 3.*** Осуществить программную реализацию алгоритмов на C++. Разработать структуры **AMatrix** и **АList**  для представления ориентированного графа матричным и списковым способом. Разработать функции преобразования из одного способа представления в другой. Разработать функцию **BFS** обхода вершин графа, используя метод поиска в ширину. Продемонстрировать работу функции. Копии экрана вставить в отчет.

Заголовочный файл для структур представления матричным и списковым способом представлен на рисунке 1:

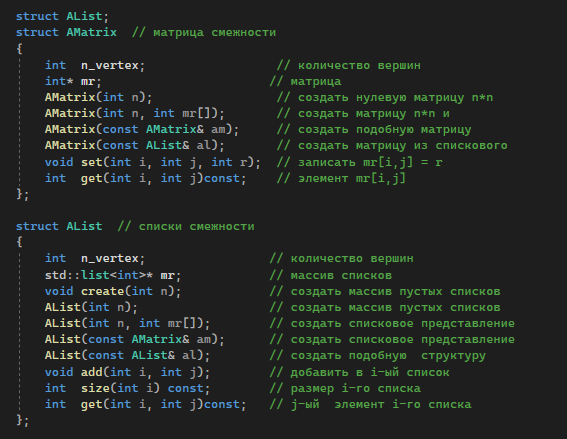


Рисунок 1 – struct of AList and AMatrix

В листинге 1 демонстрируется реализация структур AMatrix и AList:

#include "Graph.h"

namespace graph

{

AMatrix::AMatrix(int n)

{

this->n\_vertex = n;

this->mr = new int[this->n\_vertex \* this->n\_vertex];

for (int i = 0; i < n \* n; i++)mr[i] = 0;

};

AMatrix::AMatrix(int n, int mr[])

{

this->n\_vertex = n;

this->mr = mr;

};

AMatrix::AMatrix(const AMatrix& am)

{

this->n\_vertex = am.n\_vertex;

this->mr = new int[this->n\_vertex \* this->n\_vertex];

for (int i = 0; i < this->n\_vertex; i++)

for (int j = 0; j < this->n\_vertex; j++)

this->set(i, j, am.get(i, j));

};

AMatrix::AMatrix(const AList& al)

{

this->n\_vertex = al.n\_vertex;

this->mr = new int[this->n\_vertex \* this->n\_vertex];

for (int k = 0; k < this->n\_vertex \* this->n\_vertex; k++)mr[k] = 0;

for (int i = 0; i < this->n\_vertex; i++)

for (int j = 0; j < al.size(i); j++) this->set(i, al.get(i, j), 1);

};

void AMatrix::set(int i, int j, int r) { this->mr[i \* this->n\_vertex + j] = r; };

int AMatrix::get(int i, int j)const

{

return this->mr[i \* this->n\_vertex + j];

};

void AList::create(int n)

{

this->mr = new std::list<int>[this->n\_vertex = n];

};

AList::AList(int n) { create(n); }

AList::AList(const AMatrix& am)

{

this->create(am.n\_vertex);

for (int i = 0; i < this->n\_vertex; i++)

for (int j = 0; j < this->n\_vertex; j++)

if (am.get(i, j) != 0) this->add(i, j);

};

AList::AList(const AList& al)

{

this->create(al.n\_vertex);

for (int i = 0; i < this->n\_vertex; i++)

for (int j = 0; j < al.size(i); j++) this->add(i, al.get(i, j));

};

AList::AList(int n, int mr[])

{

this->create(n);

for (int i = 0; i < this->n\_vertex; i++)

for (int j = 0; j < this->n\_vertex; j++)

if (mr[i \* this->n\_vertex + j] != 0) this->add(i, j);

};

void AList::add(int i, int j) { this->mr[i].push\_back(j); };

int AList::size(int i) const { return (int)this->mr[i].size(); };

int AList::get(int i, int j)const

{

std::list<int>::iterator rc = this->mr[i].begin();

for (int k = 0; k < j; k++) rc++;

return (int)\*rc;

};

};

Листинг 1 – Graph.cpp

В листинге 2 показана реализация функции BFS:

void BFS::init(const graph::AList& al, int s)

{

this->al = &al;

this->c = new Color[this->al->n\_vertex];

this->d = new int[this->al->n\_vertex];

this->p = new int[this->al->n\_vertex];

for (int i = 0; i < this->al->n\_vertex; i++)

{

this->c[i] = WHITE;

this->d[i] = INF;

this->p[i] = NIL;

};

this->c[s] = GRAY;

this->q.push(s);

};

BFS::BFS(const graph::AList& al, int s) { this->init(al, s); };

BFS::BFS(const graph::AMatrix& am, int s)

{

this->init(\*(new graph::AList(am)), s);

};

int BFS::get()

{

int rc = NIL, v = NIL;

if (!this->q.empty())

{

rc = this->q.front();

for (int j = 0; j < this->al->size(rc); j++)

if (this->c[v = this->al->get(rc, j)] == WHITE)

{

this->c[v] = GRAY;

this->d[v] = this->d[rc] + 1;

this->p[v] = rc;

this->q.push(v);

};

this->q.pop();

this->c[rc] = BLACK;

};

return rc;

}

Листинг 2 – BFS.cpp

Результат выполнения программы представлен на рисунке 2:

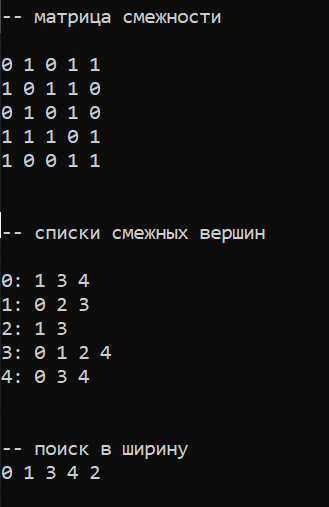


Рисунок 2 – результат выполнения программы

***Задание 4.*** Разработать функцию **DFS**  обхода вершин графа, используя метод поиска глубину. Продемонстрировать работу функции. Копии экрана вставить в отчет.

Заголовочный файл функции DFC.h:

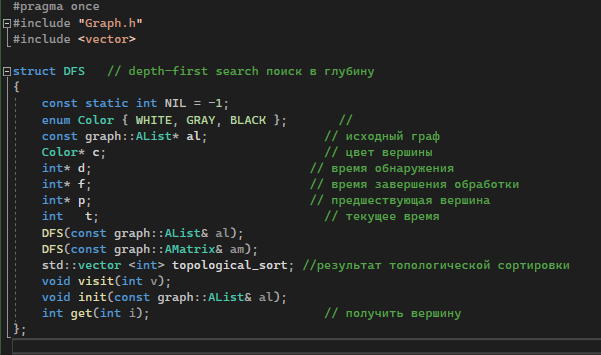


Рисунок 3 – DFC.h

Реализация заголовка представлена в листинге 3:

#include "DFS.h"

#define NINF 0x80000000

#define INF 0x7fffffff

void DFS::init(const graph::AList& al)

{

this->al = &al;

this->c = new Color[this->al->n\_vertex];

this->d = new int[this->al->n\_vertex];

this->f = new int[this->al->n\_vertex];

this->p = new int[this->al->n\_vertex];

this->t = 0;

for (int i = 0; i < this->al->n\_vertex; i++)

{

this->c[i] = WHITE;

this->d[i] = this->f[i] = 0;

this->p[i] = NIL;

};

for (int i = 0; i < this->al->n\_vertex; i++)

if (this->c[i] == WHITE)

{

this->visit(i);

this->topological\_sort.push\_back(i);

}

};

DFS::DFS(const graph::AList& al) { this->init(al); };

DFS::DFS(const graph::AMatrix& am)

{

this->init(\*(new graph::AList(am)));

};

void DFS::visit(int u)

{

int v = NIL;

this->c[u] = GRAY;

this->d[u] = ++(this->t);

for (int j = 0; j < this->al->size(u); j++)

if (this->c[v = this->al->get(u, j)] == WHITE)

{

this->p[v] = u;

this->visit(v);

this->topological\_sort.push\_back(v);

}

this->c[u] = BLACK;

this->f[u] = ++(this->t);

};

int DFS::get(int i)

{

int j = 0, min1 = INF, min2 = NINF, ntx = NIL;

for (int j = 0; j <= i; j++) // iая статистика

{

for (int k = 0; k < this->al->n\_vertex; k++)

if (this->f[k] < min1 && this->f[k] > min2)

{

min1 = this->f[k]; ntx = k;

};

min2 = min1; min1 = INF;

};

return ntx;

};

Листинг 3 - DFC.cpp

Результат выполнения программы представлен на рисунке 4:

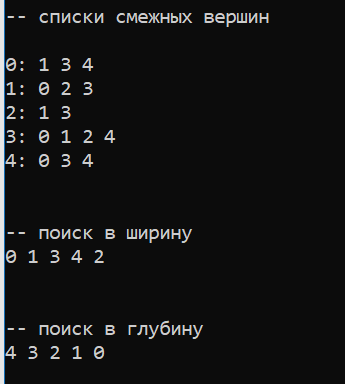


Рисунок 4 – результат выполнения программы

***Задание 5.*** Доработайте функцию **DFS**,для выполнения топологической сортировки графа. Продемонстрировать работу функции. Копии экрана вставить в отчет.

Реализация кода представлена в предыдущем задании, а результат выполнения – на рисунке 6.

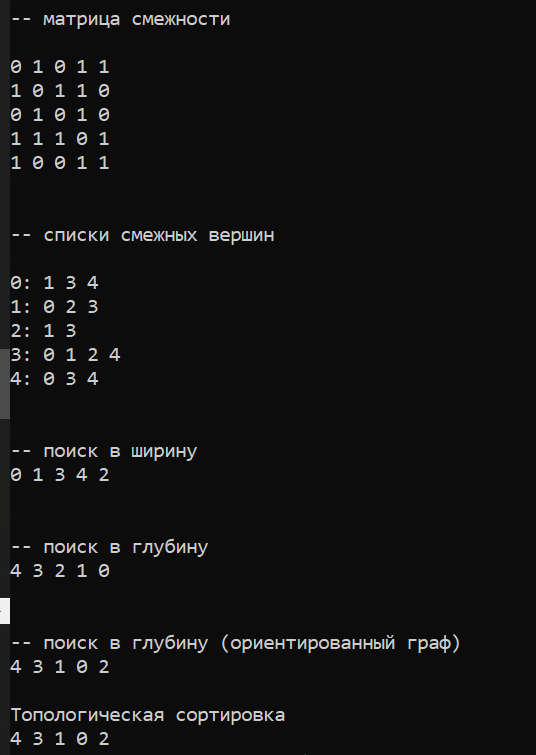
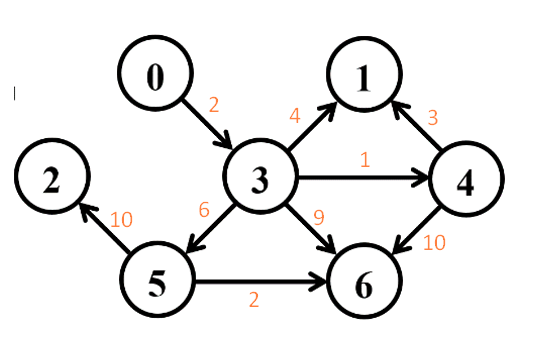
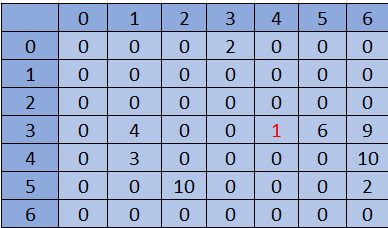


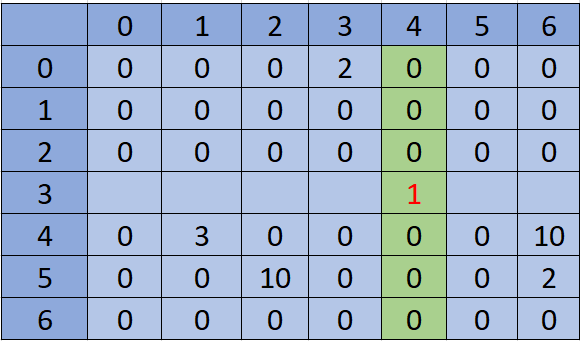
Рисунок 6 – результат выполнения программы

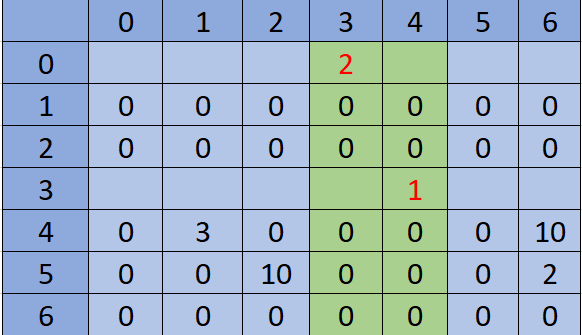
***Задание 6.*** По графу, соответствующему варианту составить минимальное остовное дерево по алгоритму Прима. Шаги построения отразить в отчете.

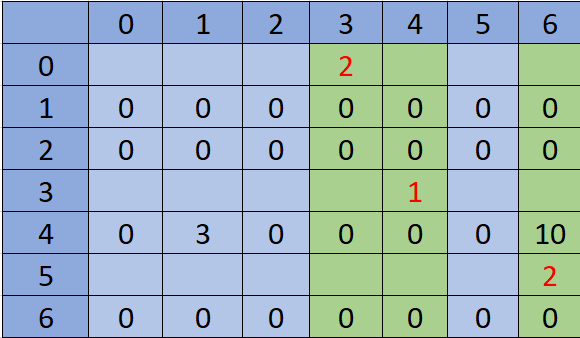


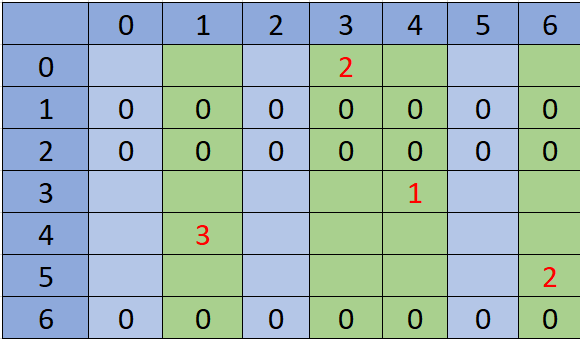
(Можно решать матрицей, вычёркивая строки и выделяя столбцы, можно через граф).

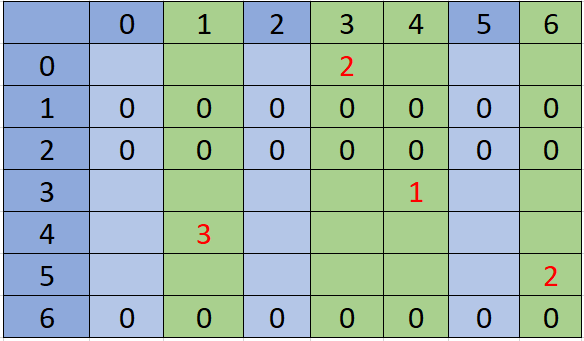












Т.е. 0-1-3-4-6

Теперь графически решим:

Для решения по алгоритму Прима необходимо выбрать произвольную начальную вершину и добавлять к остовному дереву ребра с наименьшей стоимостью, пока все вершины не будут включены в дерево.

Можно начать с вершины 0. Перечислим все ребра, связанные с этой вершиной и их стоимость:

* 0->3 (стоимость 2)

Добавляем это ребро к остовному дереву.

Теперь мы имеем две вершины в остовном дереве: 0 и 3. Необходимо выбрать следующее ребро с наименьшей стоимостью, которое связывает вершину из дерева и вершину, не принадлежащую дереву. Перечислим все такие ребра и их стоимость:

* 3->4 (стоимость 1)
* 3->5 (стоимость 6)
* 3->6 (стоимость 9)
* 3->1 (стоимость 4)

Выбираем ребро 3->4, так как оно имеет наименьшую стоимость. Добавляем вершину 4 в остовное дерево.

Теперь в дереве находятся вершины 0, 3 и 4. Необходимо выбрать следующее ребро с наименьшей стоимостью, которое связывает вершину из дерева и вершину, не принадлежащую дереву. Перечислим все такие ребра и их стоимость:

* 4->1 (стоимость 3)
* 4->6 (стоимость 10)
* 3->5 (стоимость 6)
* 3->6 (стоимость 9)
* 5->6 (стоимость 2)

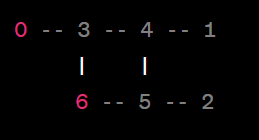
Выбираем ребро 5->6, так как оно имеет наименьшую стоимость. Добавляем вершину 6 в остовное дерево.

Теперь в дереве находятся вершины 0, 3, 4 и 6. Необходимо выбрать следующее ребро с наименьшей стоимостью, которое связывает вершину из дерева и вершину, не принадлежащую дереву. Перечислим все такие ребра и их стоимость:

* 5->2 (стоимость 10)
* 3->1 (стоимость 4)

Выбираем ребро 3->1, так как оно имеет наименьшую стоимость. Теперь в дереве находятся все вершины: 0, 3, 4, 6 и 1. Нет больше ребер, связывающих вершину из дерева и вершину, не принадлежащую дереву, поэтому алгоритм завершен.

Таким образом, остовное дерево, построенное по алгоритму Прима, имеет следующий вид:



Общая стоимость дерева равна 16.

\*Остовное дерево - это подграф, содержащий все вершины исходного графа и являющийся деревом (т.е. связным графом без циклов).

Однако, если вам нужно пройти все вершины графа, то минимальное остовное дерево может не включать все вершины. Это происходит, когда исходный граф не является связным, то есть имеет несколько компонент связности. В таком случае остовное дерево будет содержать только вершины из одной компоненты связности, а вершины из других компонент связности будут исключены из дерева.

Если нужно пройти все вершины графа, вам необходимо обходить все компоненты связности графа. Для этого можно использовать алгоритм поиска в глубину или алгоритм поиска в ширину, который позволит пройти все вершины графа.

***Задание 7.*** По графу, соответствующему варианту составить минимальное остовное дерево по алгоритму Крускала.

