МИНИСТЕРСТВО науки и высшего ОБРАЗОВАНИЯ РОссИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ

УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ

(национальный исследовательский университет)»

Институт №3 «Системы управления, информатика и электроэнергетика»

Кафедра № 304 «Вычислительные машины, системы и сети»

Структуры и алгоритмы обработки данных

Отчет по лабораторной работе

**Поиск кратчайших путей в графе.**

**Построение остовного дерева графа.**

Выполнил студент группы М3О-210Б-20  
Кильмишкин Никита Владимирович

Проверила доцент, к.т.н., Дмитриева Е.А.

Москва 2021 г.

**Оглавление**

[ТЗ 3](#_Toc88036636)

[Взвешенный ориентированный граф 4](#_Toc88036637)

[Структурная схема алгоритма 5](#_Toc88036638)

[main() 5](#_Toc88036639)

[Floyd\_Warshall() 6](#_Toc88036640)

[Crusc() 7](#_Toc88036641)

[Листинг программы 9](#_Toc88036642)

[Результаты 18](#_Toc88036643)

[Результат этапа №1 18](#_Toc88036644)

[Результат этапа №2 24](#_Toc88036645)

[Вывод 29](#_Toc88036646)

ТЗ

Для взвешенного ориентированного графа, состоящего как минимум из 10 вершин, реализовать по вариантам:

1. алгоритм поиска кратчайшего пути;
2. сделав тот же самый граф неориентированным, построить его остовное дерево минимальной стоимости.

Матрицу смежности ( значения весов каждого ребра) лучше определить в начале программы.

Должны быть представлены промежуточные результаты.

По каждому кратчайшему пути указать предшествующие вершины.

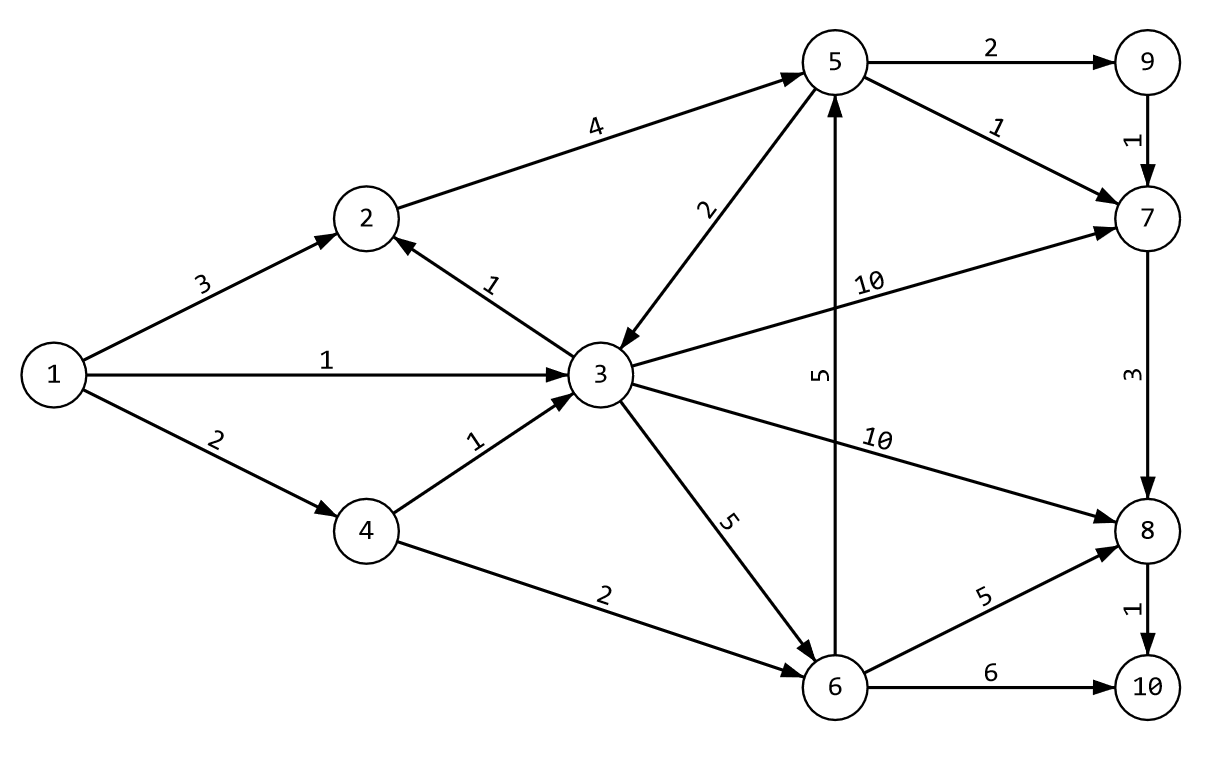
Граф и полученное остовное дерево должны быть изображены на рисунках в отчете.

Варианты заданий:

|  |  |  |
| --- | --- | --- |
| № вар. | Алгоритм, реализуемый в п.1 задания | Алгоритм, п.2 задания |
| 1 | Дейкстры | Прима |
| 2 | Беллмана-Форда | Крускала |
| 3 | Флойда-Уоршелла | Прима |
| 4 | Дейкстры | Крускала |
| 5 | Беллмана-Форда | Прима |
| 6 | Флойда-Уоршелла | Крускала |
| 7 | Дейкстры | Крускала |
| 8 | Беллмана-Форда | Прима |
| 9 | Флойда-Уоршелла | Крускала |
| 10 | Дейкстры | Прима |
| 11 | Флойда-Уоршелла | Крускала |
| 12 | Беллмана-Форда | Прима |

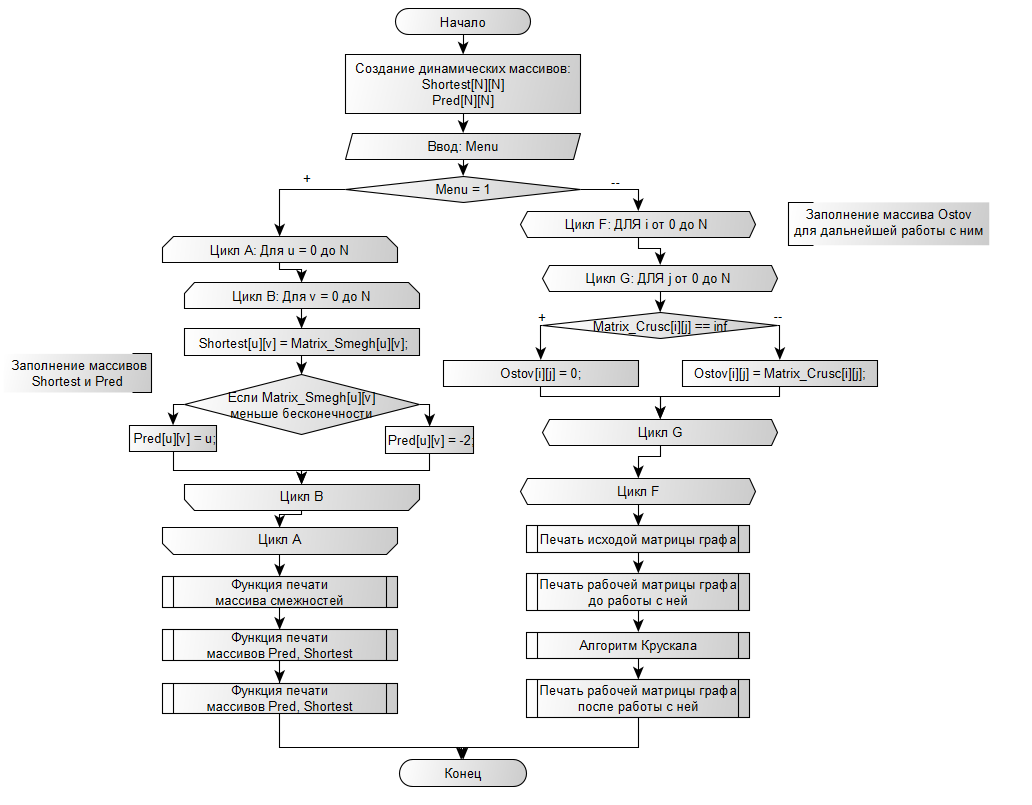
Взвешенный ориентированный граф

Вариант 11

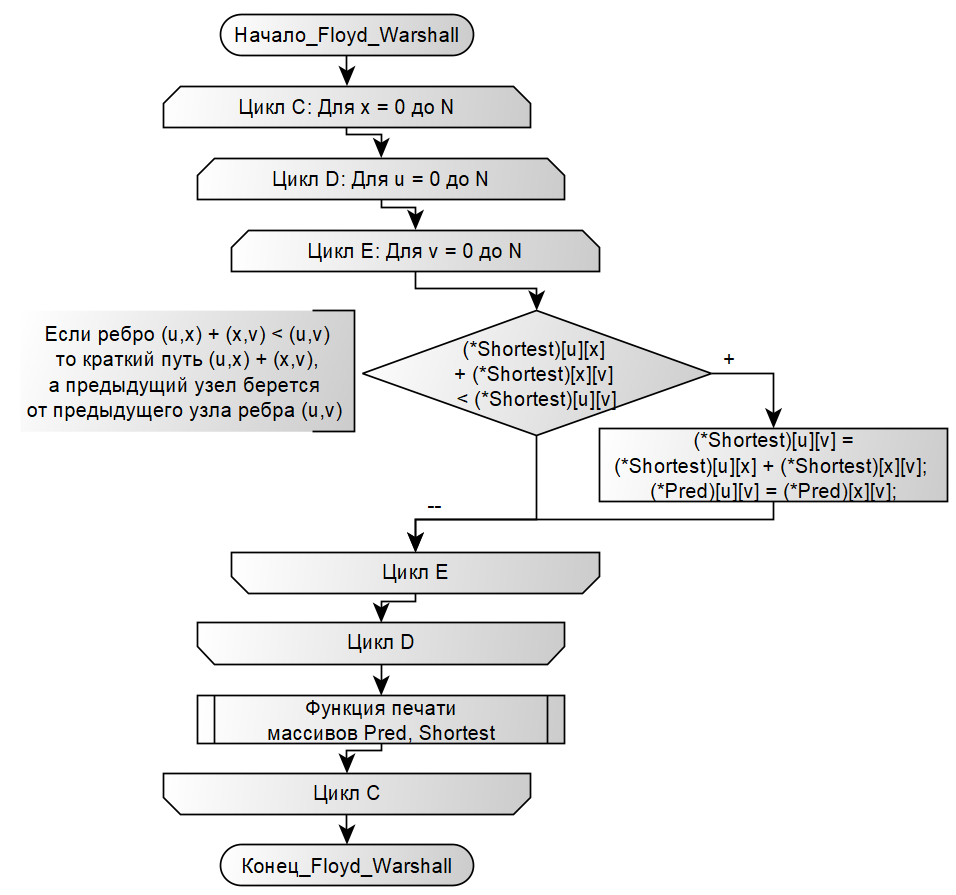


Структурная схема алгоритма

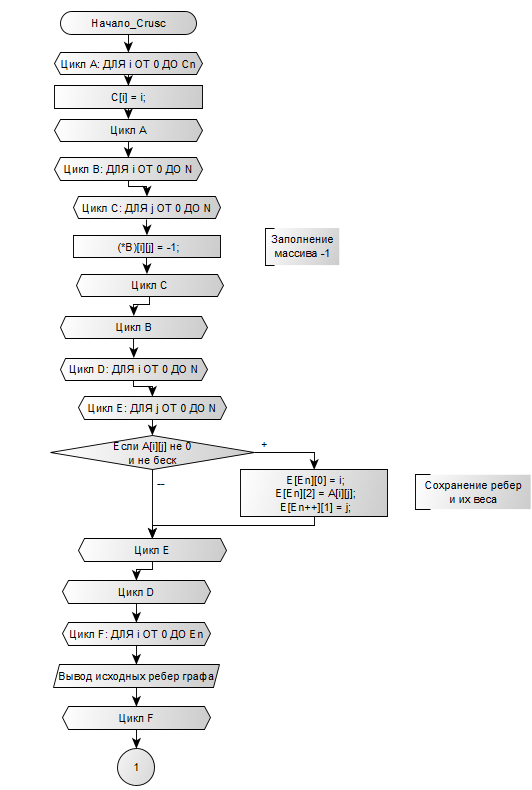
main()

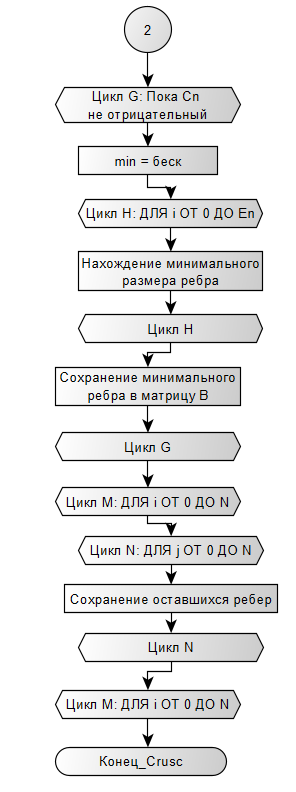
****

Floyd\_Warshall()



Crusc()





Листинг программы

#include <iostream>

#include <iomanip>

#include <algorithm>

#define inf 100000

using namespace std;

const int N = 10;

//int Matrix\_Smegh[N][N] =

//{

// // 1 2 3 4 5 6 7 8 9 10

// /\*x1\*/ { 0, 3, 4, 2, inf, inf, inf, inf, inf, inf },

// /\*x2\*/ { inf, 0, inf, inf, inf, 3, inf, inf, inf, inf },

// /\*x3\*/ { inf, inf, 0, inf, inf, 6, inf, inf, inf, inf },

// /\*x4\*/ { inf, inf, inf, 0, 5, 2, inf, inf, inf, inf },

// /\*x5\*/ { inf, inf, inf, inf, 0, inf, 6, inf, 12, inf },

// /\*x6\*/ { inf, inf, inf, inf, 1, 0, 12, 7, inf, inf },

// /\*x7\*/ { inf, inf, inf, inf, inf, inf, 0, inf, inf, 4 },

// /\*x8\*/ { inf, inf, inf, inf, inf, inf, inf, 0, inf, 3 },

// /\*x9\*/ { inf, inf, inf, inf, inf, inf, inf, 6, 0, 11 },

// /\*x10\*/ { inf, inf, inf, inf, inf, inf, inf, inf, inf, 0 }

//};

int Matrix\_Floyd[N][N] =

{

// 1 2 3 4 5 6 7 8 9 10

/\*x1\*/ { 0, 3, 1, 2, inf, inf, inf, inf, inf, inf },

/\*x2\*/ { inf, 0, inf, inf, 4, inf, inf, inf, inf, inf },

/\*x3\*/ { inf, 1, 0, inf, inf, 5, 10, 10, inf, inf },

/\*x4\*/ { inf, inf, 1, 0, inf, 2, inf, inf, inf, inf },

/\*x5\*/ { inf, inf, 2, inf, 0, inf, 1, inf, 2, inf },

/\*x6\*/ { inf, inf, inf, inf, 5, 0, inf, 5, inf, 6 },

/\*x7\*/ { inf, inf, inf, inf, inf, inf, 0, 3, inf, inf },

/\*x8\*/ { inf, inf, inf, inf, inf, inf, inf, 0, inf, 1 },

/\*x9\*/ { inf, inf, inf, inf, inf, inf, 1, inf, 0, inf },

/\*x10\*/ { inf, inf, inf, inf, inf, inf, inf, inf, inf, 0 }

};

int Matrix\_Crusc[N][N]

{

// 1 2 3 4 5 6 7 8 9 10

/\*x1\*/ { 0, 3, 1, 2, inf, inf, inf, inf, inf, inf },

/\*x2\*/ { 3, 0, 1, inf, 4, inf, inf, inf, inf, inf },

/\*x3\*/ { 1, 1, 0, 1, 2, 5, 10, 10, inf, inf },

/\*x4\*/ { 2, inf, 1, 0, inf, 2, inf, inf, inf, inf },

/\*x5\*/ { inf, 4, 2, inf, 0, 5, 1, inf, 2, inf },

/\*x6\*/ { inf, inf, 5, 2, 5, 0, inf, 5, inf, 6 },

/\*x7\*/ { inf, inf, 10, inf, 1, inf, 0, 3, 1, inf },

/\*x8\*/ { inf, inf, 10, inf, inf, 5, 3, 0, inf, 1 },

/\*x9\*/ { inf, inf, inf, inf, 2, inf, 1, inf, 0, inf },

/\*x10\*/ { inf, inf, inf, inf, inf, 6, inf, 1, inf, 0 }

};

//int Matrix\_Smegh[N][N] =

//{

// // 1 2 3 4 5 6 7 8 9 10

// /\*x1\*/ { 0, inf, 2, inf, inf, inf, inf, inf, inf, inf },

// /\*x2\*/ { 4, 0, inf, inf, 5, inf, inf, inf, inf, inf },

// /\*x3\*/ { inf, 1, 0, inf, inf, inf, inf, inf, inf, inf },

// /\*x4\*/ { inf, inf, 3, 0, inf, inf, inf, inf, inf, inf },

// /\*x5\*/ { inf, inf, inf, inf, 0, 5, inf, 6, inf, inf },

// /\*x6\*/ { 8, inf, inf, inf, inf, 0, 1, inf, inf, inf },

// /\*x7\*/ { inf, inf, inf, inf, 7, inf, 0, inf, inf, 3 },

// /\*x8\*/ { inf, inf, inf, 4, inf, inf, inf, 0, 5, inf },

// /\*x9\*/ { inf, inf, inf, inf, 2, inf, inf, inf, 0, inf },

// /\*x10\*/ { inf, inf, inf, inf, inf, inf, inf, inf, 1, 0 }

//};

//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Процедуры печати\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

void Print\_Pred(int\*\* Array);

void Print\_Smegh(int Matrix[N][N]);

void Print\_Help\_Matr(int\*\* Shortest);

//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Вспомогательные функции\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

void Way(int \*\*Pred);

int mini(int value1, int value2) { return ((value1 < value2) ? value1 : value2); };

void DeleteMas(int\* Array) { delete[] Array; }

void BotBorder() { cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << endl; }

//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Алгоритмы\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

void Floyd\_Warshall(int \*\*\*Shotest, int \*\*\*Pred);

void Crusc(int A[][N], int\*\*\* B, int R[][3]);

//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Для\_Таблиц\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

void Shapka() {

cout << setfill(char(196))

<< char(218) << setw(5); // "┌───────────"

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

cout << char(194) << setw(5); // "┬───────────"

cout << char(191) << endl; // "┐"

};

void Numberacya() {

cout << setfill(' ');

cout << char(179) << setw(4) << " "; // "│ "

for (int i = 1; i <= 10; i++)

cout << char(179) << setw(4) << i; // "│ 1"

cout << char(179); // "│"

}

void Mid() {

cout << endl << setfill(char(196))

<< char(195) << setw(5);

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

cout << char(197) << setw(5); // "├───────────"

cout << char(180) << endl; // "┤"

}

void Niz() {

cout << endl << setfill(char(196))

<< char(192) << setw(5); // "└───────────"

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

cout << char(193) << setw(5); // "┴───────────"

cout << char(217) << endl; // "┘"

}

//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

int main()

{

setlocale(LC\_ALL, "Rus");

system("color F0");

int Menu;

cout << " Введите число " << endl

<< " 1 -- Алгоритм Флойда-Уоршелла " << endl

<< " 2 -- Алгоритм Крускала (Краскала) " << endl;

cout << " Этап: "; cin >> Menu;

//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ЭТАП 1\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

if (Menu == 1) // Алгоритм Флойда\_Уоршелла

{

// Инициализация двумерных динамичeских массивов 10х10

int\*\* Shortest = new int\* [N];

for (int i = 0; i < N; i++) Shortest[i] = new int[N]();

int\*\* Pred = new int\* [N];

for (int i = 0; i < N; i++) Pred[i] = new int[N]();

// Заполнение массивов

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

{

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

{

Shortest[u][v] = Matrix\_Floyd[u][v]; // Копирование с матрицы смежностей

if (Matrix\_Floyd[u][v] < inf)

Pred[u][v] = u;

else

Pred[u][v] = -2;

}

}

// Печать массива смежностей

cout << "\n\t\tИсходная матрица смежности" << endl;

Print\_Smegh(Matrix\_Floyd);

BotBorder();

cout << "\t\t\tБыло:" << endl << endl;

Print\_Pred(Pred); // Печать матрицы Pred

Print\_Help\_Matr(Shortest); // Печать матрицы Shortest

BotBorder();

Floyd\_Warshall(&Shortest, &Pred); // Алгоритм Флойда-Уоршелла

BotBorder();

cout << "\t\t\tСтало:" << endl << endl;

Print\_Pred(Pred); // Печать матрицы Pred

Print\_Help\_Matr(Shortest); // Печать матрицы Shortest

BotBorder();

Way(Pred); // Функция нахождения пути

BotBorder();

} // if

//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ЭТАП 2\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

else // Алгоритм Крускала

{

// Инициализация двумерных динамичeского массива 10х10

int\*\* Ostov = new int\* [N];

for (int i = 0; i < N; i++) Ostov[i] = new int[N]();

int R[N - 1][3]; // Массив для записи ребер и их веса

// Заполнение массива

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

{

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

{

// Копирование с матрицы смежностей

if (Matrix\_Crusc[i][j] == inf)

Ostov[i][j] = 0;

else

Ostov[i][j] = Matrix\_Crusc[i][j];

} // for j

} // for i

cout << "\t\tИсходная матрица графа:" << endl;

Print\_Smegh(Matrix\_Crusc);

BotBorder();

cout << "\t\tМатрица Ostov до алгоритма:" << endl;

Print\_Help\_Matr(Ostov);

BotBorder();

cout << "\n Остовное дерево наименьшего веса по Крускалу";

Crusc(Matrix\_Crusc, &Ostov, R);

BotBorder();

cout << "\n Матрица смежности графа с остовным деревом : " << endl;

Print\_Help\_Matr(Ostov);

BotBorder();

}

} // main()

//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ФУНКЦИИ\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Алгоритм Флойда-Уоршелла\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

void Floyd\_Warshall(int\*\*\* Shortest, int\*\*\* Pred)

{

for (int x = 0; x < 10; x++)

{

for (int u = 0; u < 10; u++)

{

for (int v = 0; v < 10; v++)

{

if ((\*Shortest)[u][x] + (\*Shortest)[x][v] < (\*Shortest)[u][v])

{

(\*Shortest)[u][v] = (\*Shortest)[u][x] + (\*Shortest)[x][v];

(\*Pred)[u][v] = (\*Pred)[x][v];

}

}

}

cout << "\t\t\tx = " << x + 1 << endl;

Print\_Pred((\*Pred)); // Печать матрицы Pred

Print\_Help\_Matr(\*Shortest); // Печать матрицы Shortest

cout << " \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ " << endl;

}

}

//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Алгоритм Крускала\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

void Crusc(int A[][N], int \*\*\*B, int R[][3])

{

int k; // Счётчик

int xmin; // Индекс ребра

int min; // Минимальный вес

int En = 0;

int Cn = N; // Количество узлов

int C[N]; // Для избежания повторов

//int E[N \* (N - 1) / 2][3];

struct edge

{

int out;

int in;

int weight;

};

edge\* Edge = new edge[N \* (N - 1) / 2];

for (int i = 0; i < Cn; i++) // Заполнение массива

C[i] = i;

for (int i = 0; i < N; i++) // Заполнения матрицы -1

{

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

{

(\*B)[i][j] = -1;

} // for j

} // for i

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

{

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

{

if ((A[i][j]) && (A[i][j] != inf)) // Если не 0

{ // и не бесконечность

Edge[En].out = i; // Заполнение структуры

Edge[En].weight = A[i][j];

Edge[En++].in = j;

} // if

} // for j

} // for i

cout << "\n Ребра исходного графа : \n"; // Вывод ребер

cout << En << endl;

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

{

cout << " " << Edge[i].out + 1 << " –– "

<< Edge[i].in + 1 << " = "

<< Edge[i].weight << endl;

} // for i

while (Cn >= 0)

{

min = inf;

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

{

if ((Edge[i].weight < min) && (C[Edge[i].out] != C[Edge[i].in]))

{

xmin = i; // Находит индекс минимального ребра

min = Edge[i].weight; // Находит вес минимального ребра

} // if

} // for i

int u = Edge[xmin].out;

int v = Edge[xmin].in;

int weight = Edge[xmin].weight;

(\*B)[v][u] = (\*B)[u][v] = weight; // Вставляет в матрицу

C[u] = C[v] = mini(C[u], C[v]);

Cn--;

//печать

cout << "\t\t\tCn: " << Cn << endl

<< " \t\t Добавил " << Edge[xmin].out + 1 << " –– "

<< Edge[xmin].in + 1 << " = " << Edge[xmin].weight << endl;

Print\_Help\_Matr((\*B));

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

cout << " " << i + 1 << ": " << C[i] << endl;

} // while

k = 0;

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

{

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

{

if ((\*B)[i][j] > 0)

{

R[k][0] = i; R[k][1] = j;

R[k++][2] = (\*B)[i][j];

cout << " " << R[k - 1][0] + 1 << " -- "   
<< R[k-1][1] + 1 << " = " << R[k - 1][2] << endl;

Sum += R[k - 1][2]; } // if

} // for j

} // for i

cout << " Суммарный вес: " << Sum << endl;

} // Crusc()

//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Вывод пути\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

void Way(int\*\* Pred)

{

int k, m;

cout << " Введите откуда: "; cin >> k;

cout << " Введите куда : "; cin >> m;

cout << " Из " << k << " в " << m << ":" << endl;

int Temp = m - 1;

if (Pred[k - 1][Temp] > 0)

{

cout << " " << m;

while (Temp != k - 1)

{

cout << " <-- " << Pred[k - 1][Temp] + 1;

Temp = Pred[k - 1][Temp];

}

}

else cout << " Ошибка: Пути из " << k << " в " << m << " нет";

cout << endl;

}

//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Функции печати\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_//\*

void Print\_Pred(int \*\*Array)

{

cout << "\t\t\tPred" << endl;

setlocale(LC\_ALL, "C"); //ОТКЛЮЧЕНИЕ РУССКОГО ЯЗЫКА

Shapka();

Numberacya();

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

{

Mid();

cout << setfill(' ') << char(179) << setw(4) << u + 1 <<char(179);

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

{

if(Array[u][v] > -1)

cout << setw(4) << Array[u][v] + 1 << char(179);

else

cout << setw(4) << "NULL" << char(179);

}

}

Niz();

setlocale(LC\_ALL, "Rus");

cout << endl;

}

//\*

void Print\_Smegh(int Matrix[N][N])

{

setlocale(LC\_ALL, "C"); //ОТКЛЮЧЕНИЕ РУССКОГО ЯЗЫКА

Shapka();

Numberacya();

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

{

Mid();

cout << setfill(' ') << char(179) << setw(4) << u + 1 << char(179);

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

{

if (Matrix[u][v] == inf)

cout << setw(4) << "OO" << char(179);

else

cout << setw(4) << Matrix[u][v] << char(179);

}

}

Niz();

setlocale(LC\_ALL, "Rus");

cout << endl;

}

//\*

void Print\_Help\_Matr(int\*\* Matrix)

{

// cout << "\t\t\tShortest" << endl;

setlocale(LC\_ALL, "C"); //ОТКЛЮЧЕНИЕ РУССКОГО ЯЗЫКА

Shapka();

Numberacya();

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

{

Mid();

cout << setfill(' ') << char(179) << setw(4) << u + 1 << char(179);

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

{

switch (Matrix[u][v])

{

case -1:

{

cout << setw(4) << 0 << char(179);

break;

}

case inf:

{

cout << setw(4) << "OO" << char(179);

break;

}

default:

{

cout << setw(4) << Matrix[u][v] << char(179);

break;

}

}

}

}

Niz();

setlocale(LC\_ALL, "Rus");

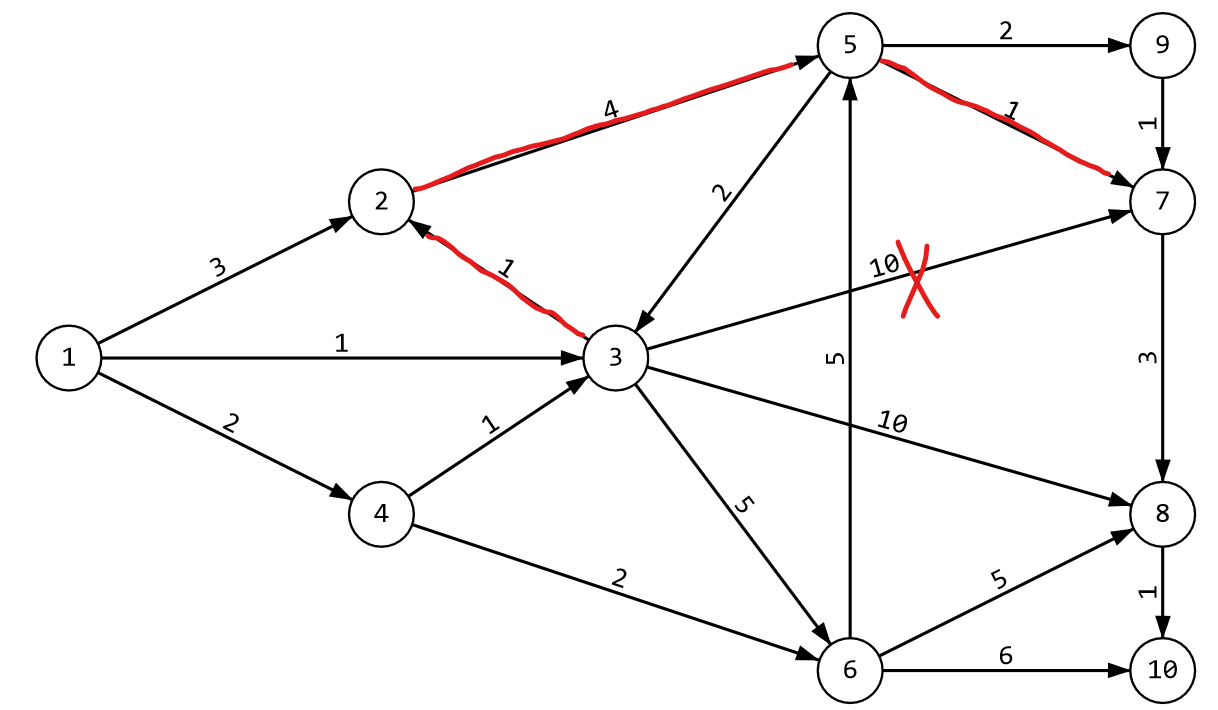
cout << endl;

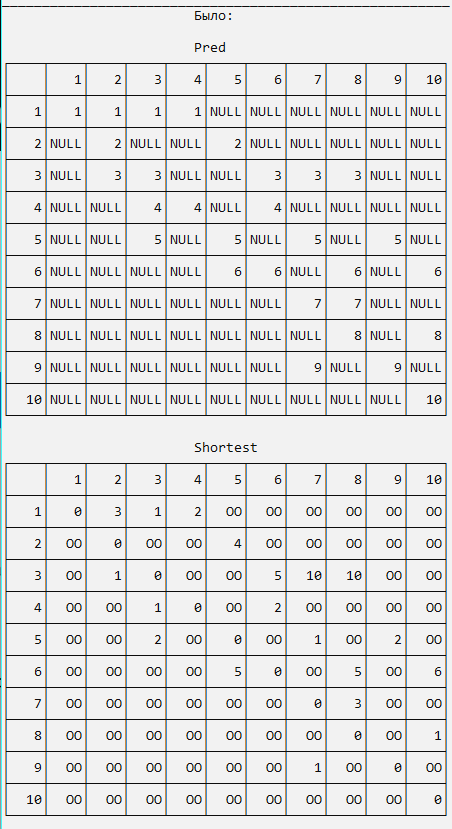
}

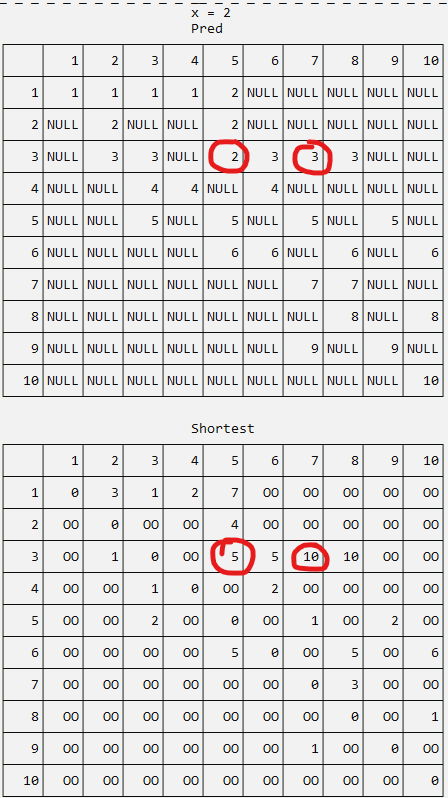
//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_//\*

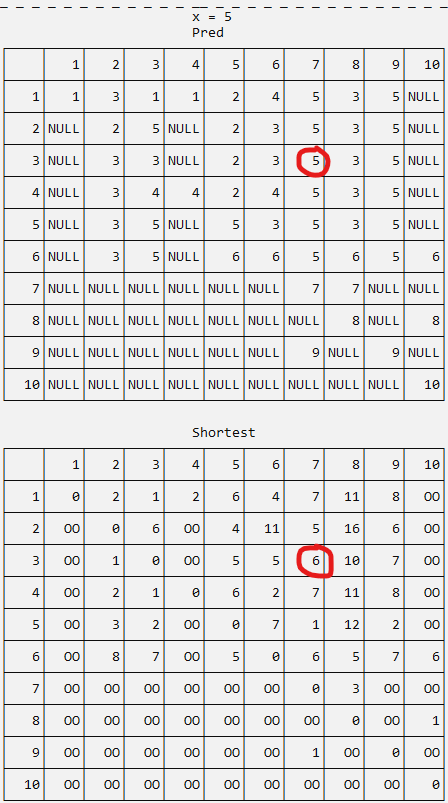
Результаты

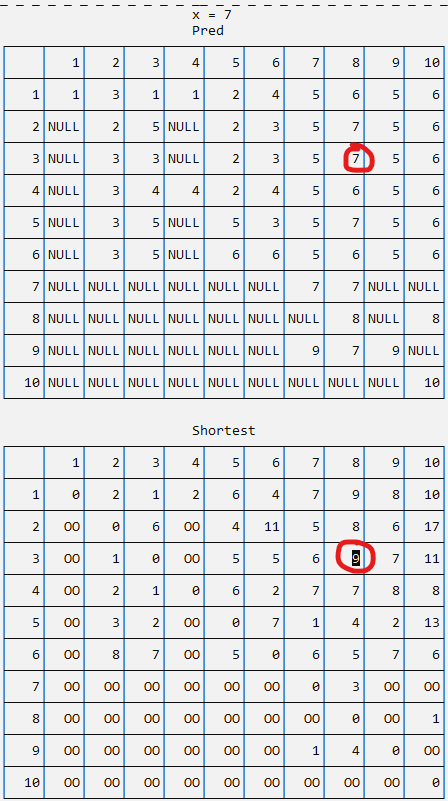
Результат этапа №1

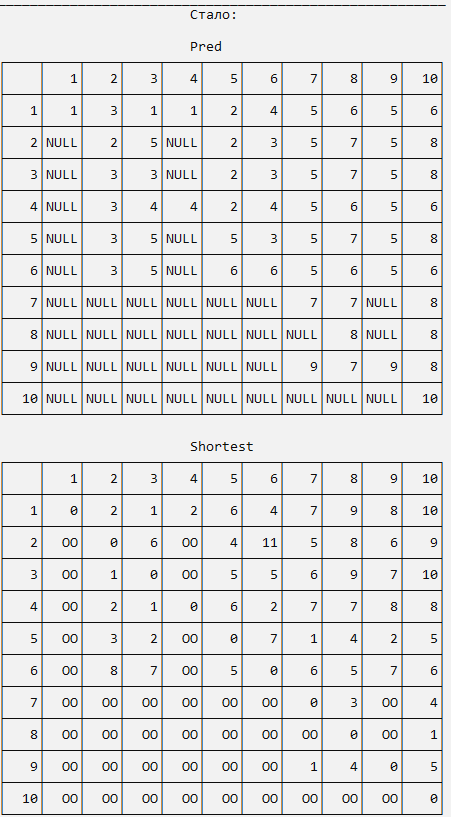


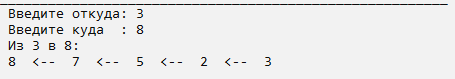




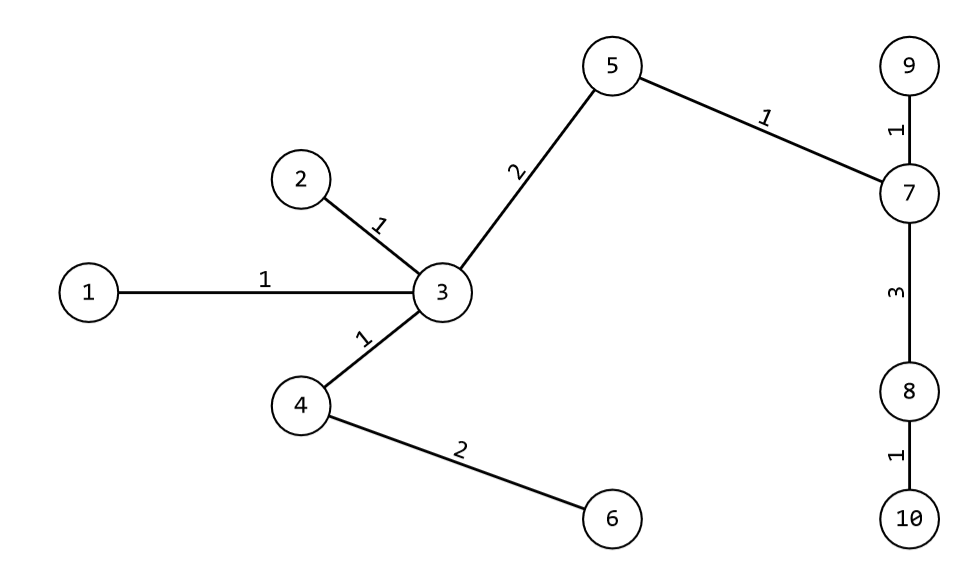


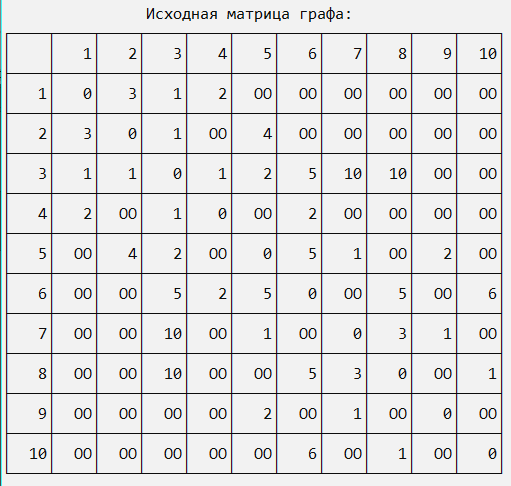
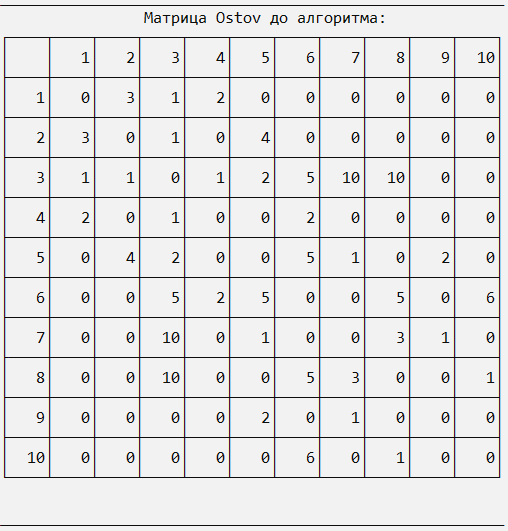


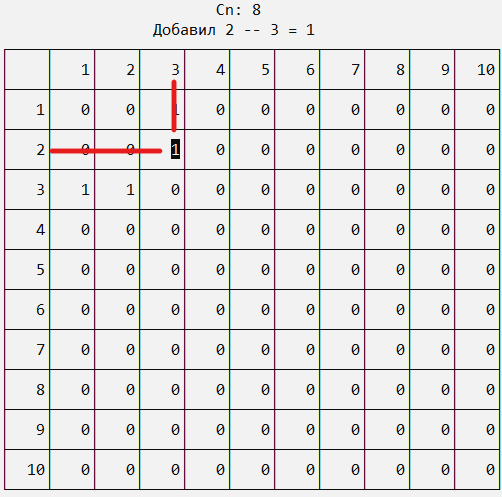
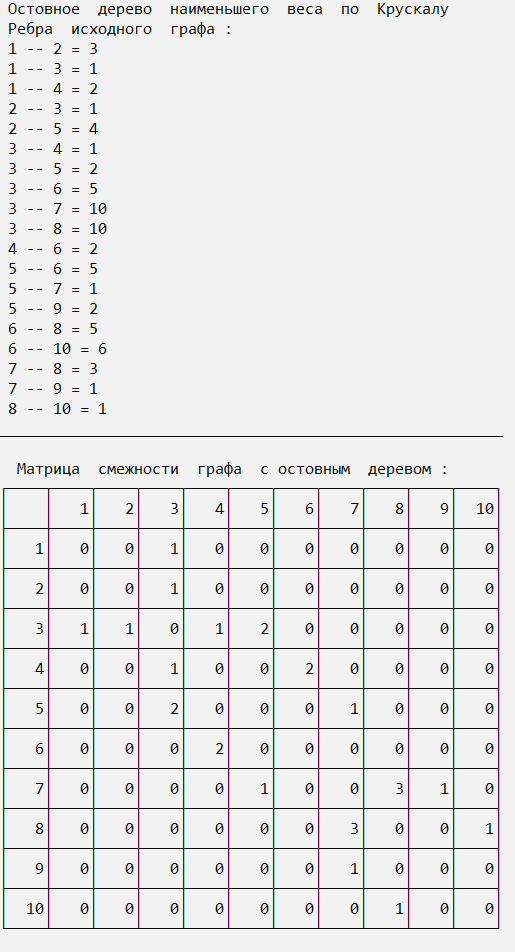


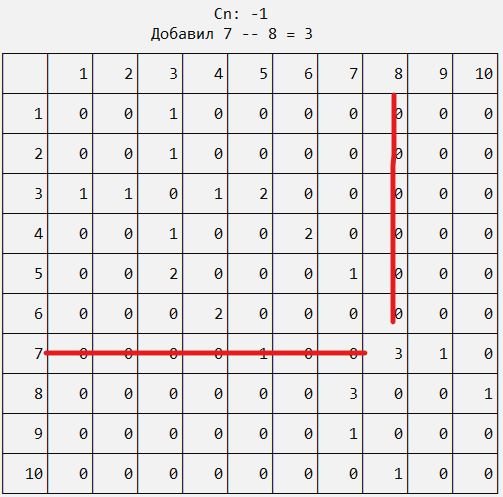
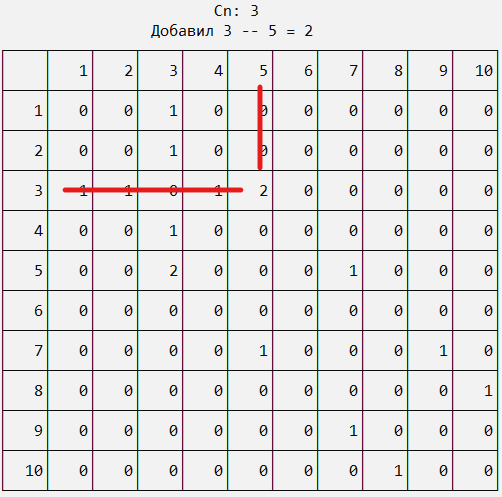


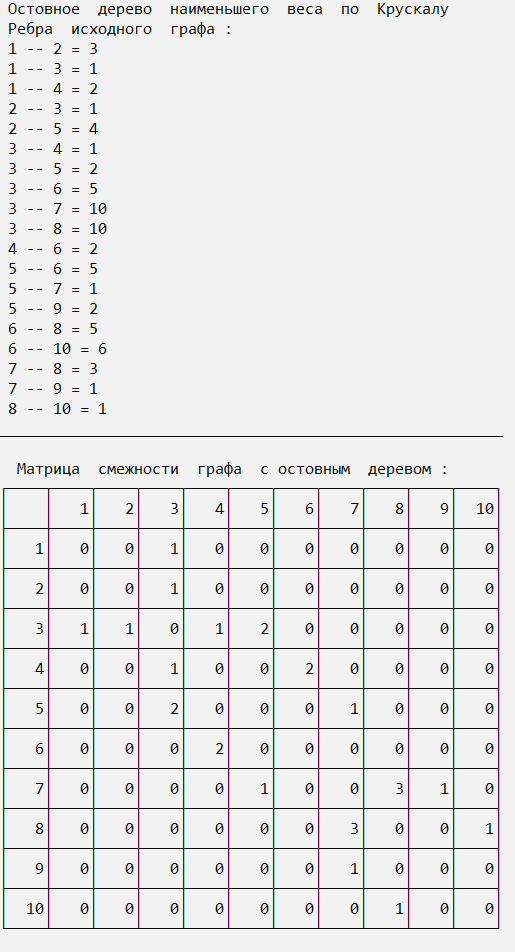
Результат этапа №2



****

****

****

****

Вывод

В лабораторной работе был реализован для взвешенного ориентированного графа, состоящего из 10 вершин, алгоритм поиска кратчайшего пути;

Также для того же самого граф, но неориентированного был построено его остовное дерево минимальной стоимости.

Матрица смежности была определена в начале программы.

Были представлены промежуточные результаты.

По каждому кратчайшему пути были указаны предшествующие вершины.

Граф и полученное остовное дерево были изображены на рисунках в отчете.