**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.**

Выполнили студент группы М3О-210Б-21  
Пысларь Александр Игоревич

Осминнов Никита Михайлович

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

Москва 2022 г.

**Оглавление**

[ТЗ 3](#_heading=h.30j0zll)

[Взвешенный ориентированный граф 4](#_heading=h.1fob9te)

[Структурная схема алгоритма 5](#_heading=h.3znysh7)

[main() 5](#_heading=h.2et92p0)

f[loydWarshall() 6](#_heading=h.tyjcwt)

k[ruscal() 7](#_heading=h.3dy6vkm)

[Листинг программы 9](#_heading=h.1t3h5sf)

[Результаты 18](#_heading=h.4d34og8)

[Результат этапа №1 18](#_heading=h.2s8eyo1)

[Результат этапа №2 24](#_heading=h.17dp8vu)

[Вывод 29](#_heading=h.3rdcrjn)

**ТЗ**

**Лабораторная работа.**

**Поиск кратчайших путей в графе. Построение остовного дерева графа.**

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.

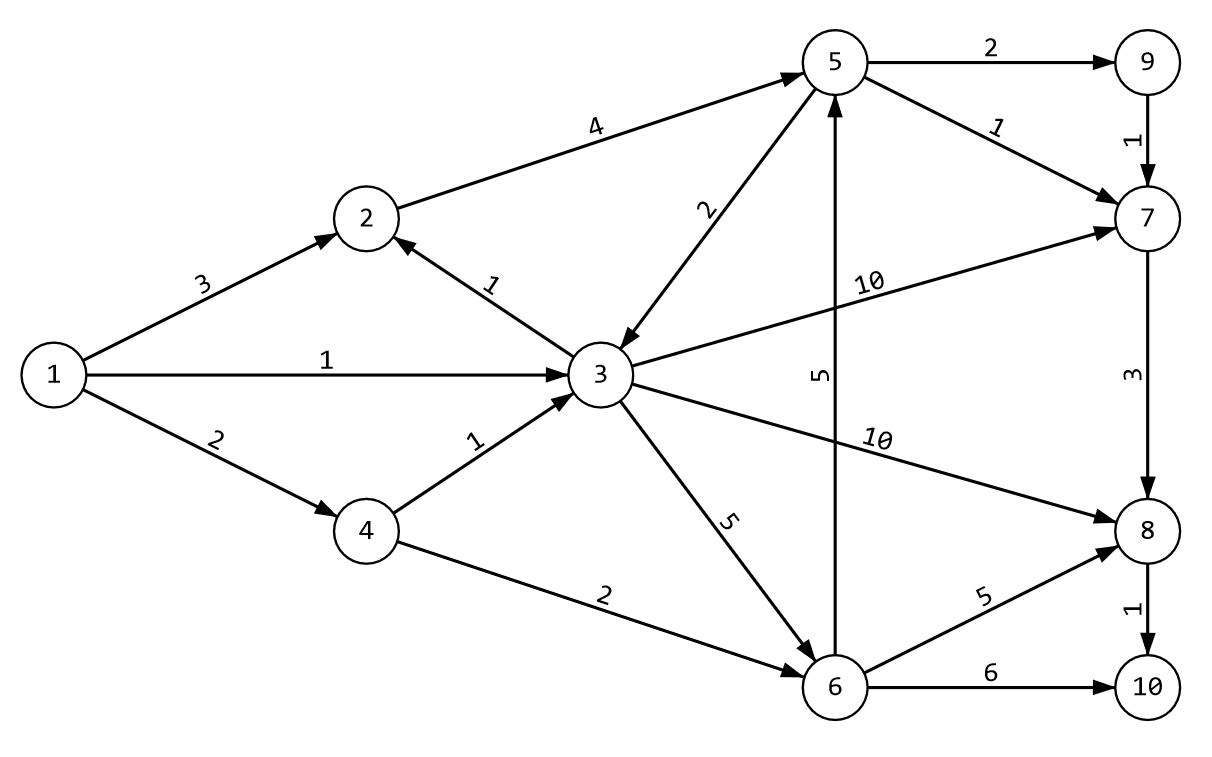
Граф и полученное остовное дерево должны быть изображены на рисунках в отчете.

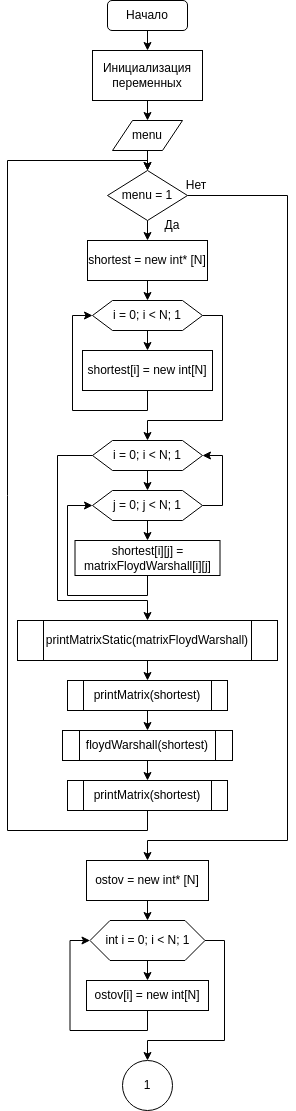
Варианты заданий:

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

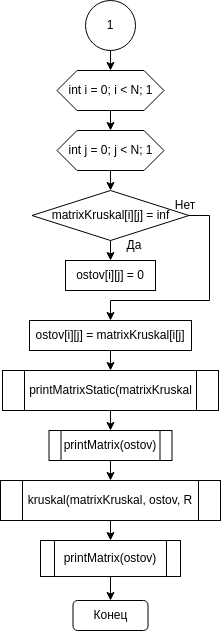
**Взвешенный ориентированный граф**

Вариант 6

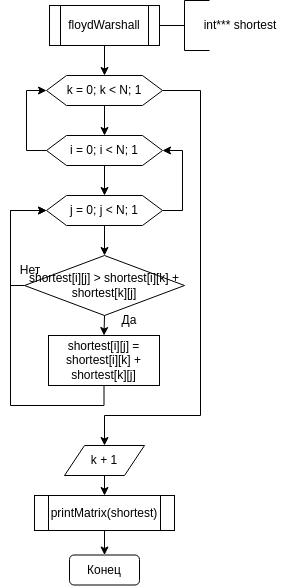


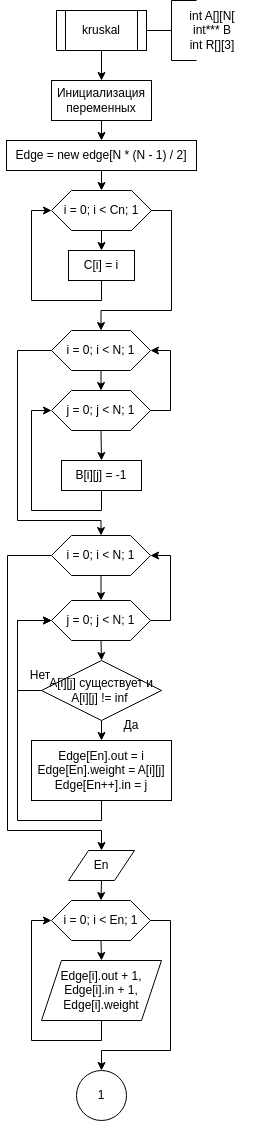
**Структурная схема алгоритма** 

main()

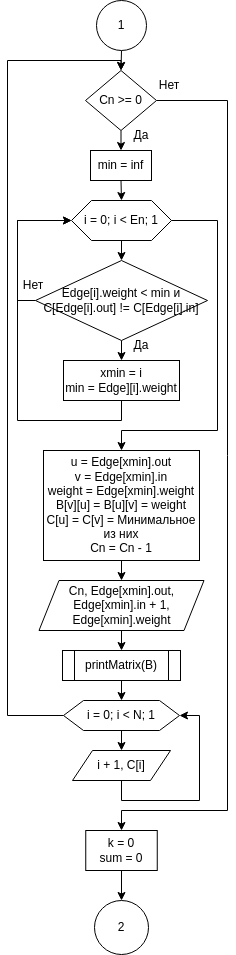


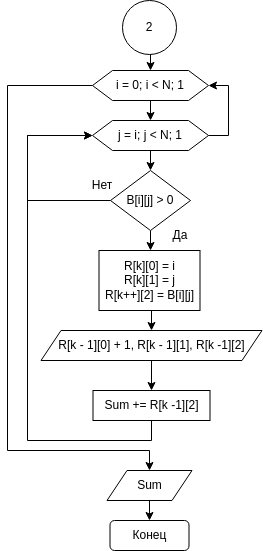
floydWarshall()





kruscal()



****

**Листинг программы**

**main.cpp**

#include <iostream>

#include <iomanip>

#include <algorithm>

#include "functions.h"

using namespace std;

int matrixFloydWarshall[N][N] = { // Матрица для работы с алгоритмом Флойда-Уоршелла

{ 0, 3, 1, 2, inf, inf, inf, inf, inf, inf},

{ inf, 0, inf, inf, 4, inf, inf, inf, inf, inf},

{ inf, 1, 0, inf, inf, 5, 10, 10, inf, inf},

{ inf, inf, 1, 0, inf, 2, inf, inf, inf, inf},

{ inf, inf, 2, inf, 0, inf, 1, inf, 2, inf},

{ inf, inf, inf, inf, 5, 0, inf, 5, inf, 6},

{ inf, inf, inf, inf, inf, inf, 0, 3, inf, inf},

{ inf, inf, inf, inf, inf, inf, inf, 0, inf, 1},

{ inf, inf, inf, inf, inf, inf, 1, inf, 0, inf},

{ inf, inf, inf, inf, inf, inf, inf, inf, inf, 0}

};

int matrixKruskal[N][N] = { // Матрица для работы с алгоритмом Краскала

{ 0, 3, 1, 2, inf, inf, inf, inf, inf, inf},

{ 3, 0, 1, inf, 4, inf, inf, inf, inf, inf},

{ 1, 1, 0, 1, 2, 5, 10, 10, inf, inf},

{ 2, inf, 1, 0, inf, 2, inf, inf, inf, inf},

{ inf, 4, 2, inf, 0, 5, 1, inf, 2, inf},

{ inf, inf, 5, 2, 5, 0, inf, 5, inf, 6},

{ inf, inf, 10, inf, 1, inf, 0, 3, 1, inf},

{ inf, inf, 10, inf, inf, 5, 3, 0, inf, 1},

{ inf, inf, inf, inf, 2, inf, 1, inf, 0, inf},

{ inf, inf, inf, inf, inf, 6, inf, 1, inf, 0}

};

int main() {

int menu; // Выбор действия

cout << "Enter the number:" << endl

<< "1 -- Floyd-Warshall algorithm" << endl

<< "2 -- Kruskal algorithm" << endl;

cout << "Stage: ";

cin >> menu; // Ввод пользовательского действия

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]();

// Заполнение массивов

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

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

shortest[i][j] = matrixFloydWarshall[i][j]; // Копирование с матрицы смежностей

}

}

// Печать массива смежностей

cout << "Initial Adjacency Matrix:" << endl;

printMatrixStatic(matrixFloydWarshall);

// Печать матрицы Shortest

cout << "Before:" << endl;

printMatrix(shortest);

floydWarshall(&shortest); // Алгоритм Флойда-Уоршелла

// Печать матрицы Shortest

cout << "After:" << endl;

cout << "Shortest" << endl;

printMatrix(shortest);

}

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 (matrixKruskal[i][j] == inf)

ostov[i][j] = 0;

else

ostov[i][j] = matrixKruskal[i][j];

}

}

cout << "Source graph matrix:" << endl;

printMatrixStatic(matrixKruskal);

cout << "Ostov matrix before algorithm:" << endl;

printMatrix(ostov);

cout << "Least Kruskal weight spanning tree:" << endl;

kruskal(matrixKruskal, &ostov, R);

cout << "Adjacency matrix of graph with spanning tree:" << endl;

printMatrix(ostov);

}

}

**functions.cpp**

#include <iostream>

#include <iomanip>

#include <algorithm>

#include "functions.h"

using namespace std;

// Метод печати

void printMatrixStatic(int matrix [][N]) { // Печать статической матрицы

tableHeader(); // Заголовок таблицы

tableValues(); // Значения таблицы

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

tableCenter(); // Центр таблицы

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

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

switch (matrix[i][j]) {

case -2: {

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

break;

}

case inf: {

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

break;

}

default: {

cout << setw(4) << matrix[i][j] << char(179);

break;

}

}

}

}

tableBottom(); // Низ таблицы

cout << endl;

}

void printMatrix(int\*\* matrix) { // Печать динамической матрицы

tableHeader(); // Заголовок таблицы

tableValues(); // Значения таблицы

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

tableCenter(); // Центр таблицы

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

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

switch (matrix[i][j]) {

case -2: {

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

break;

}

case 100000: {

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

break;

}

default: {

cout << setw(4) << matrix[i][j] << char(179);

break;

}

}

}

}

tableBottom(); // Низ таблицы

cout << endl;

}

// Методы для построения таблиц

void tableHeader() { // Заголовок таблицы

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 tableValues() { // Значения таблицы

cout << setfill(' ');

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

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

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

cout << char(179); // "│"

}

void tableCenter() { // Центр таблицы

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 tableBottom() { // Низ таблицы

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; // "┘"

}

// Методы алгоритмов

void floydWarshall(int\*\*\* shortest) { // Алгоритм Флойда-Уоршелла

for (int k = 0; k < N; k++) {

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

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

if ((\*shortest)[i][j] >(\*shortest)[i][k] + (\*shortest)[k][j]) {

(\*shortest)[i][j] = (\*shortest)[i][k] + (\*shortest)[k][j];

}

}

}

cout << "x = " << k + 1 << endl;

printMatrix(\*shortest); // Печать матрицы Shortest

}

}

void kruskal(int A[][N], int\*\*\* B, int R[][3]) { // Алгоритм Краскала

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

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 (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;

}

}

}

cout << "Edges of the original graph: "; // 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;

}

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

}

}

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] = std::min(C[u], C[v]);

Cn--;

cout << "Cn: " << Cn << endl

<< "Added " << Edge[xmin].out + 1 << " -- "

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

printMatrix((\*B));

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

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

}

k = 0;

int Sum = 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];

}

}

}

cout << "Total weight: " << Sum << endl;

}

**functions.h**

#ifndef FUNCTIONS\_H

#define FUNCTIONS\_H

#define inf 100000

const int N = 10;

using namespace std;

// Методы печати

void printMatrixStatic(int matrix[][N]); // Печать статической матрицы

void printMatrix(int\*\* matrix); // Печать динамической матрицы

// Методы для построения таблиц

void tableHeader(); // Заголовок таблицы

void tableValues(); // Значения таблицы

void tableCenter(); // Центр таблицы

void tableBottom(); // Низ таблицы

// Методы алгоритмов

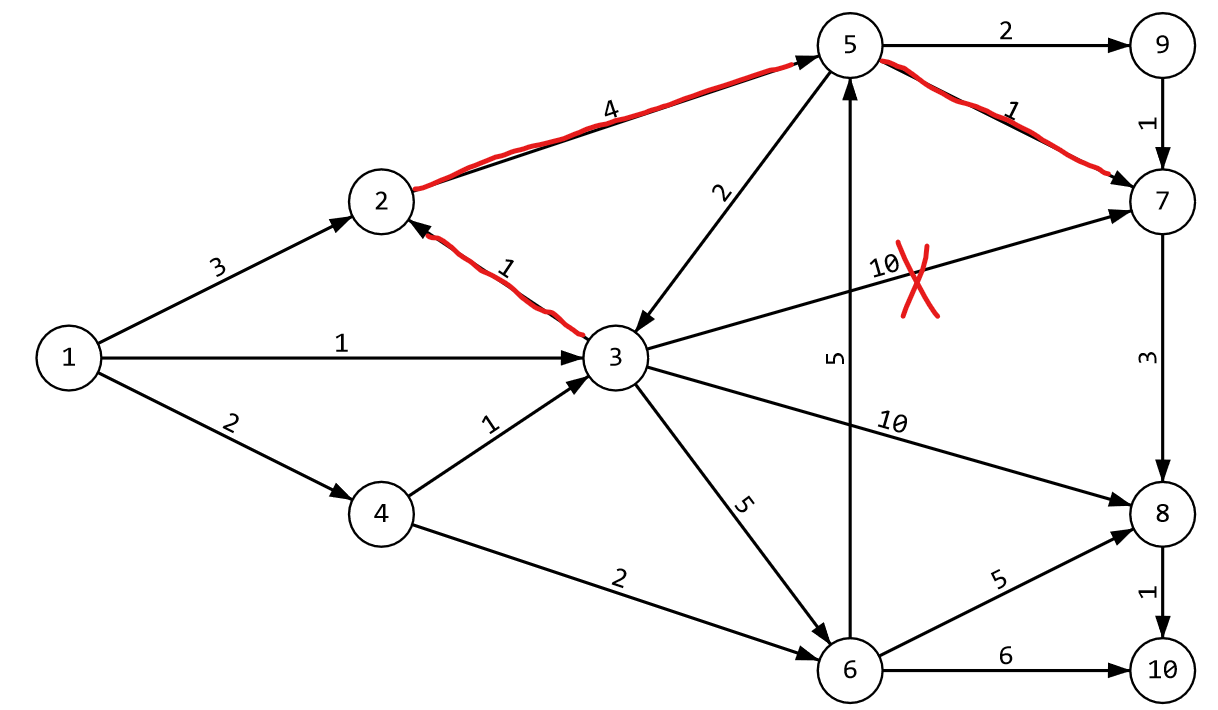
void floydWarshall(int\*\*\* shotest); // Алгоритм Флойда-Уоршелла

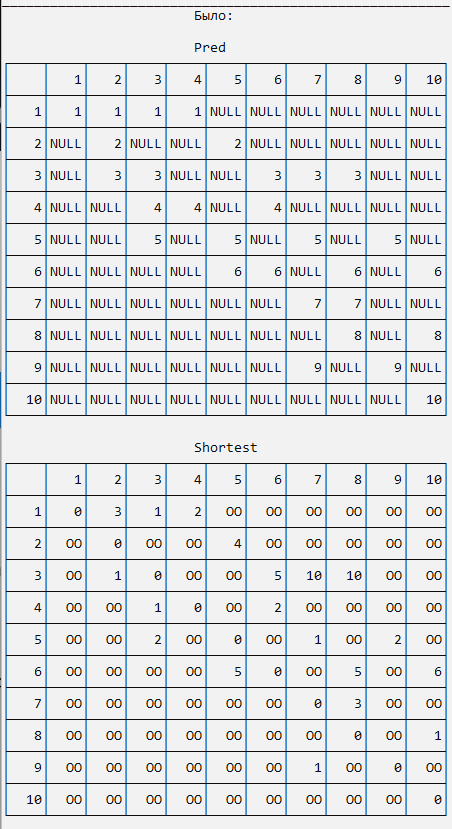
void kruskal(int A[][N], int\*\*\* B, int R[][3]); // Алгоритм Краскала

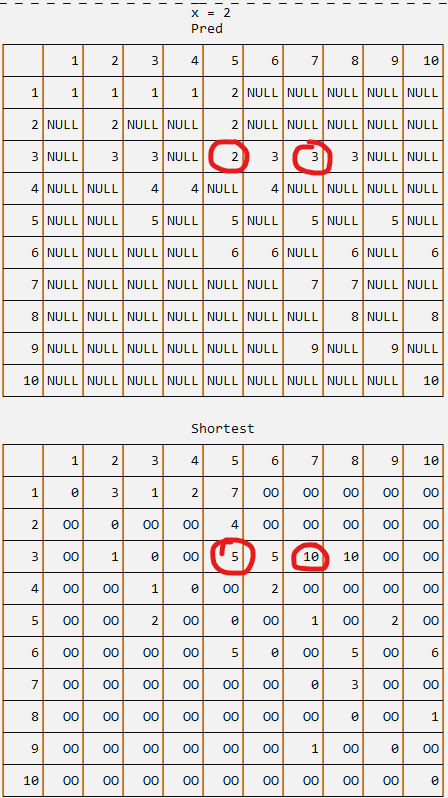
#endif

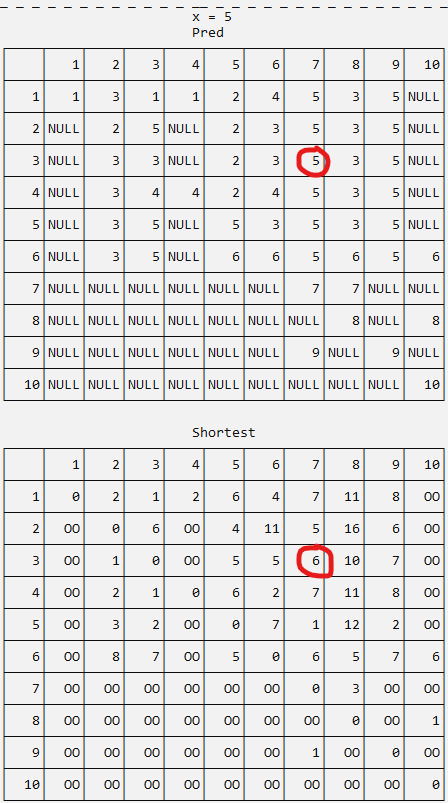
**Результаты**

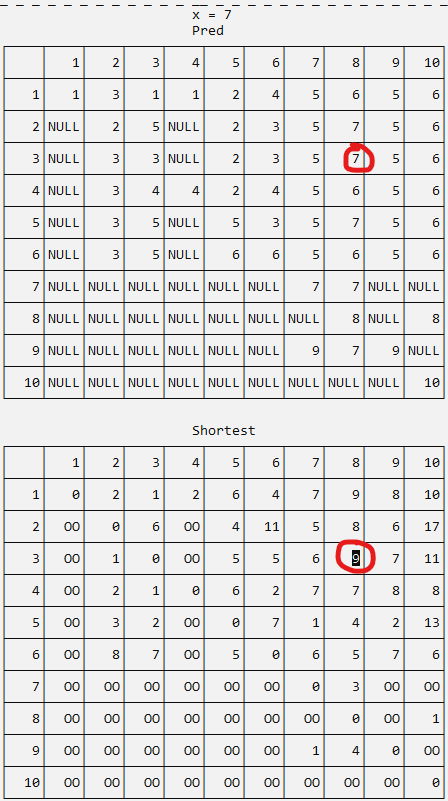
Результат этапа №1

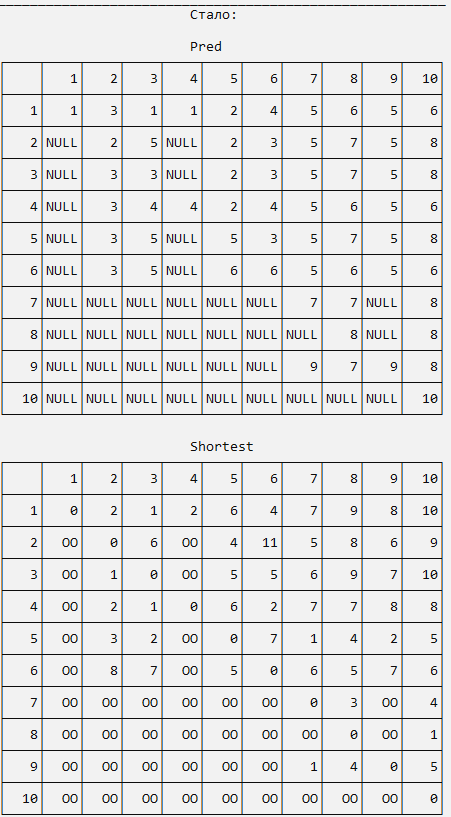


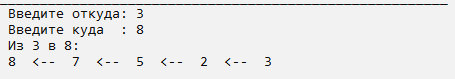




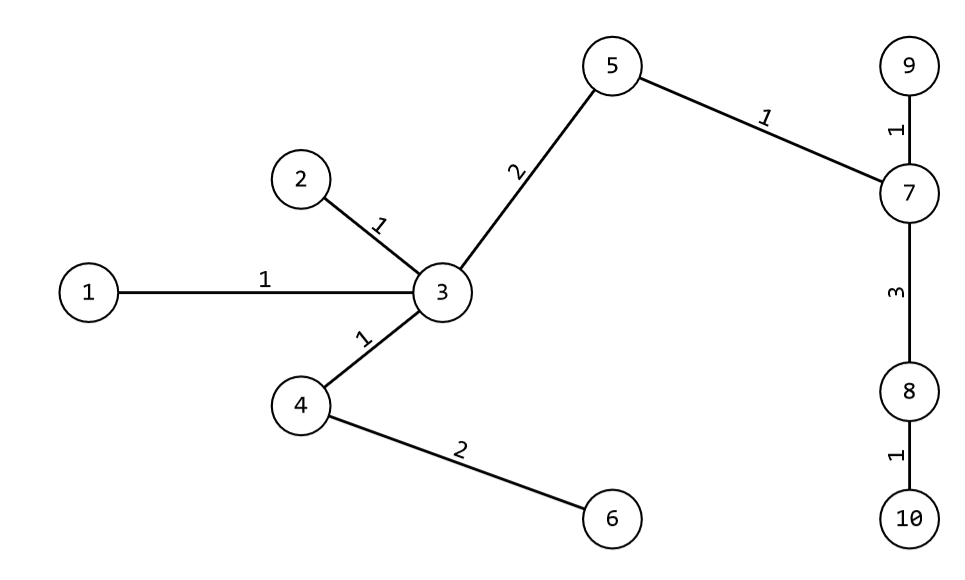


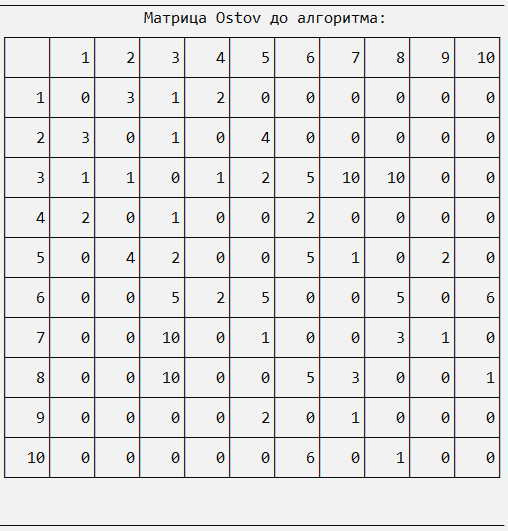
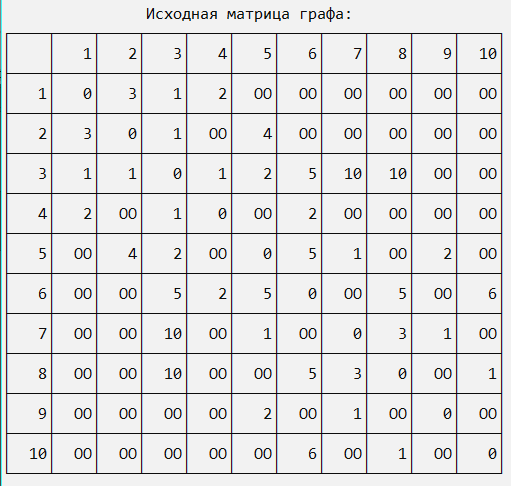


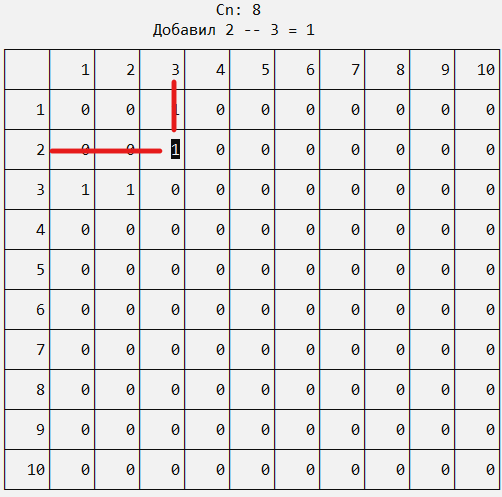
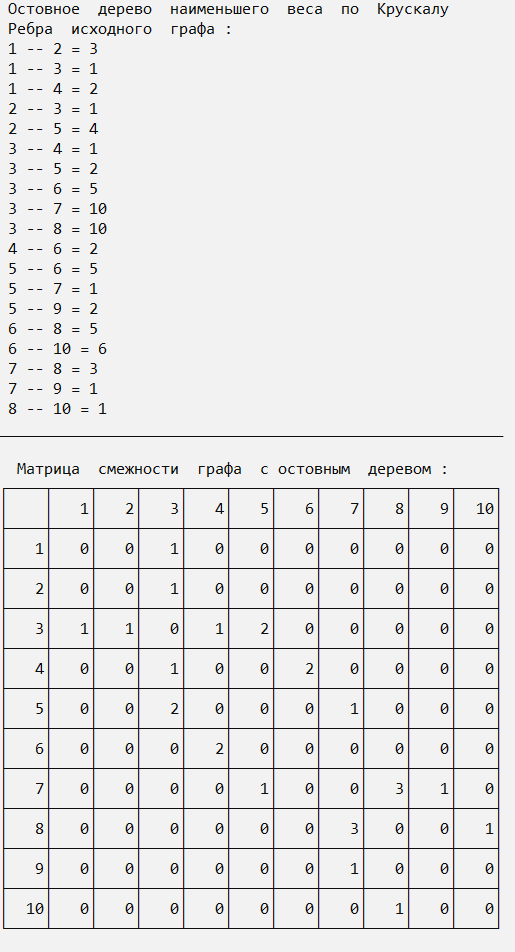


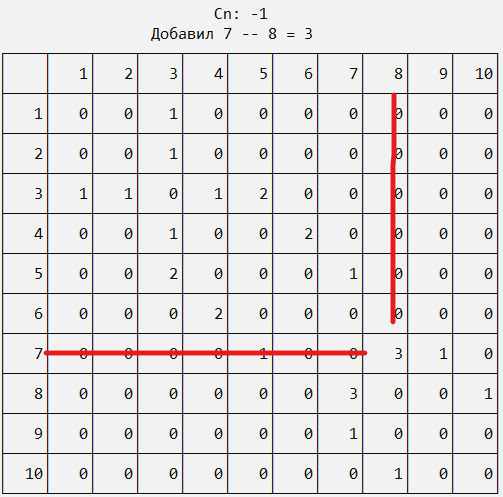
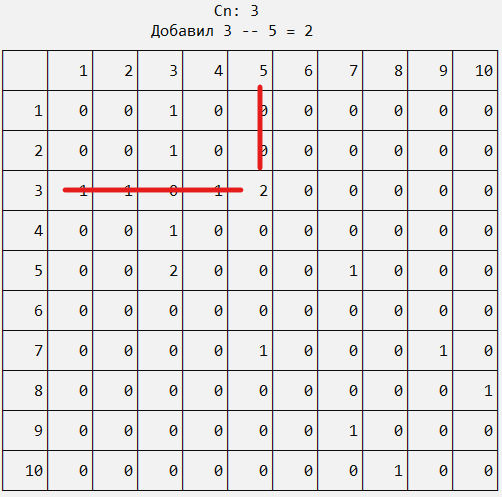


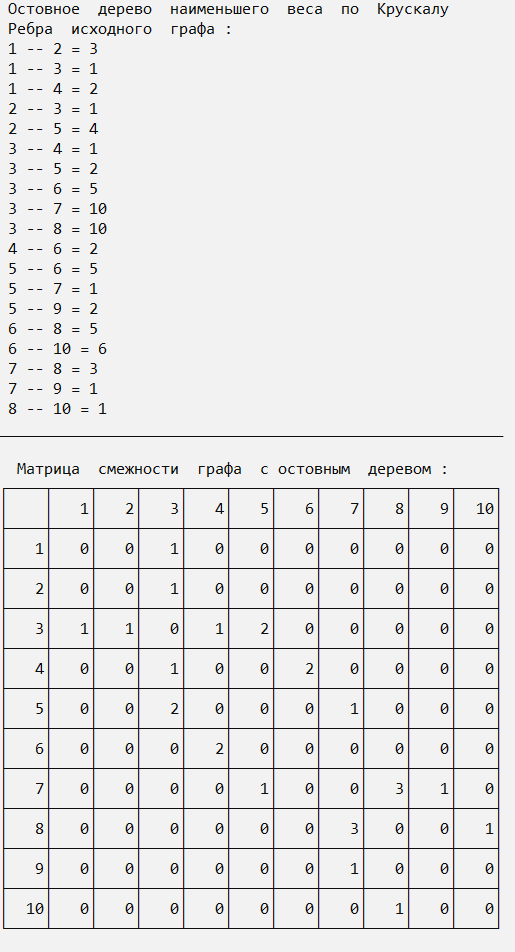
Результат этапа №2











**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.