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 №2

Бинарные деревья поиска

Выполнили студенты группы M3О-211Б-21

Плоцкий Б.А.

Раужев Ю.М.

Москва 2022 г.

Оглавление

[Задание 3](#_Toc121407254)

[Код программы 4](#_Toc121407255)

[Тестирование программы 18](#_Toc121407256)

[Тест 1 18](#_Toc121407257)

[Вывод 20](#_Toc121407258)

# Задание

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.

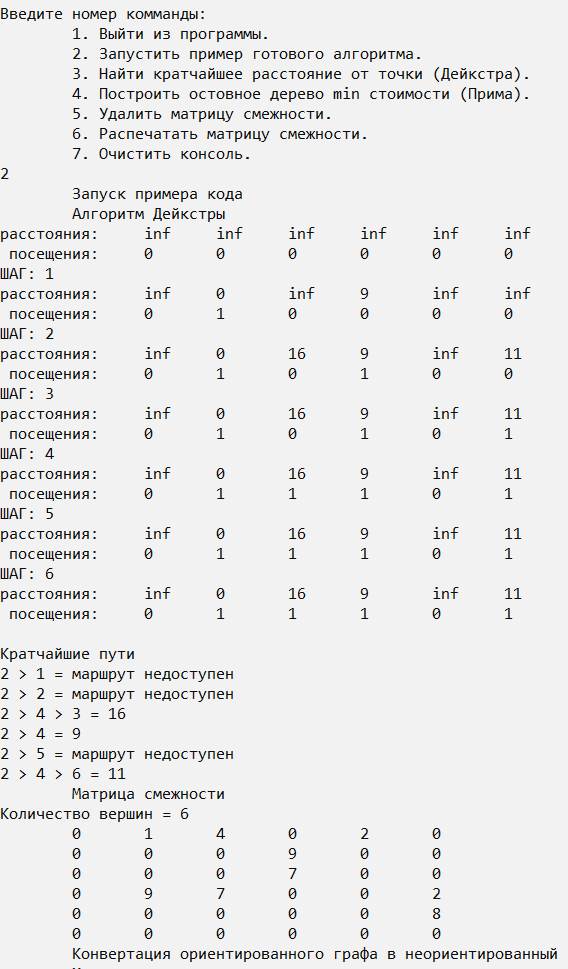
|  |  |  |
| --- | --- | --- |
| № вар. | Алгоритм, реализуемый в п.1 задания | Алгоритм, п.2 задания |
| **4** | **Дейкстры** | **Прима** |

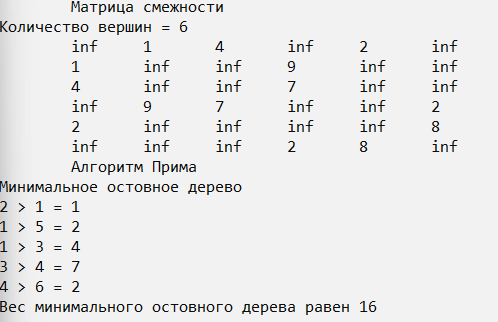
# Код программы

*/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\  
\*                    кафедра № 304 2 курс 3 семестр информатика         \*  
\*-----------------------------------------------------------------------\*  
\*   Project type : solution                                             \*  
\*   Project name : LW2\_GRAPH                                            \*  
\*   File name    : main.cpp                                             \*  
\*   Language     : c/c++                                                \*  
\*   Programmers  : Плоцкий Б.А. Раужев Ю. М.                            \*  
\*   Created      : 10/11/22                                             \*  
\*   Last revision: 30/11/22                                             \*  
\*   Comment(s)   :                                                      \*  
\*       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.                              \*  
\*                                                                       \*  
\\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
#include <iostream>  
//#include <iomanip>  
#include <fstream>  
#include <Windows.h> // для считывания кириллицы  
#include <string>****using******namespace****std;  
   
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
\*                      К О Н С Т А Н Т Ы                        \*  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
   
// если нужна печать по каждому действию  
#define NEED\_PRINT\_DEBUG 1  
   
// вывод в консоль сообщения  
#define INFO(str) if(NEED\_PRINT\_DEBUG)cout<<"\t"<<str<<"\n";  
   
// коды для взаимодействия пользователья с программой****enum******class****input\_codes  
{  
    exit = 0,  
    template\_program,  
    find\_short\_path\_diykstra,  
    create\_min\_spanning\_tree\_prim,  
    delete\_smej\_matr,  
    print\_smej\_matr,  
    clear\_console  
};  
   
// путь до графаа****const******string****graph\_file\_path = "graph.txt";  
   
// строка с коммандами****const******string****command\_str =  
"\nВведите номер комманды:\n\  
\t1. Выйти из программы.\n\  
\t2. Запустить пример готового алгоритма.\n\  
\t3. Найти кратчайшее расстояние от точки (Дейкстра).\n\  
\t4. Построить остовное дерево min стоимости (Прима).\n\  
\t5. Удалить матрицу смежности.\n\  
\t6. Распечатать матрицу смежности.\n\  
\t7. Очистить консоль.";  
   
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
\*               М А Т Р И Ц А   С М Е Ж Н О С Т И               \*  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/****struct****SmejMatr  
{****int****\*\* m\_smej\_matr;****int****m\_numb\_of\_vertexes;  
   
    // конструктор  
    SmejMatr()  
        :m\_smej\_matr(nullptr), m\_numb\_of\_vertexes(0)  
    {}  
    // деструктор  
    ~SmejMatr()  
    {****for****(****int****i = 0; i < m\_numb\_of\_vertexes; i++)  
        {  
            delete[] m\_smej\_matr[i];  
        }  
        delete[] m\_smej\_matr;  
    }  
};  
   
// чтение матрицы смежности из файла  
SmejMatr\* ReadGraphFromFile(****string****\_file\_path = graph\_file\_path);  
   
// печать матрицы смежности****void****PrintGraph(SmejMatr\*);  
   
// конвертация ориентированного графа в неориентированный****void****ConverOrientedIntoDisoriened(SmejMatr\*&);  
   
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
\*        В С П О М О Г А Т Е Л Ь Н Ы Е   Ф У Н К Ц И И          \*  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
   
// печать массива  
template<****class****T,****class****FUNC>****void****PrintArr(T\* \_arr,****int****\_size, FUNC \_prt);  
   
// обработка элемента из массива расстояний при печати****string****ChangeDistElem(****int****\_elem);  
   
// обработка элемента из массива посещений при печати****string****ChangeVisitElem(****int****\_elem);  
   
// установка одного значения во все ячейка матрицы  
template<****class****T>****void****SetMartixValue(T\*\* \_mat,****int****\_sizex,****int****\_sizey, T \_value);  
   
// проверка на номер команды****bool****CheckNumber(****int****\_num);  
   
// пример готовой программы****void****ExampleProgram();  
   
// выделение памяти под двумерный массив  
template<****class****T>  
T\*\* MemAlloc(****int****\_size\_x,****int****\_size\_y);  
   
// ввод и проверка значений  
template<typename T,****class****FUNC>  
T InputAndCheck(FUNC,****string****welcome\_str,****string****err\_str  
    = "Было введено некорректное значение");  
   
// функция ведения диалога с пользователем****void****Dialog();  
   
// Поиск кратчайшего пути****void****Diykstra(SmejMatr\* \_sm,****int****\_start\_vert);  
   
// построение Min остового дерева****void****Prima(SmejMatr\* \_sm,****int****\_st);  
   
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
\*                Г Л А В Н А Я   Ф У Н К Ц И Я                  \*  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/****int****main()  
{  
    // для считывания кириллицы  
    SetConsoleCP(1251);  
    SetConsoleOutputCP(1251);  
   
    // запуск диалога  
    Dialog();****return****0;  
}  
   
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
\*              Р Е А Л И З А Ц И Я   Ф У Н К Ц И Й              \*  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
   
SmejMatr\* ReadGraphFromFile(****string****\_file\_path)  
{  
    ifstream fin(\_file\_path);  
   
    // если файл не открылся****if****(!fin.is\_open())  
    {  
        INFO("ЧТЕНИЕ: файл не был открыт (9((");****return****nullptr;  
    }  
   
    // создаем структуру смежной матрицы  
    SmejMatr\* sm =****new****SmejMatr;  
   
    // количество вершин  
    sm->m\_numb\_of\_vertexes = 1;  
   
    // выяснение количества строк****int****numb\_of\_string = 0;****int****cur\_vert = 0;****while****(fin.peek() != EOF)  
    {****int****temp;  
        fin >> temp;****if****(temp != cur\_vert)  
        {  
            cur\_vert = temp;  
            sm->m\_numb\_of\_vertexes++;  
        }  
        fin.ignore(INT\_MAX, '\n');  
        numb\_of\_string++;  
    }  
   
    // перемещение указателя в начало файла  
    fin.close();  
    fin.open(\_file\_path);  
   
    //выделение памяти под матрицу  
    sm->m\_smej\_matr = MemAlloc<****int****>(sm->m\_numb\_of\_vertexes, sm->m\_numb\_of\_vertexes);  
    SetMartixValue(sm->m\_smej\_matr, sm->m\_numb\_of\_vertexes, sm->m\_numb\_of\_vertexes, 0);  
   
    // считываем матрицу смежности****for****(****int****i = 0; i < numb\_of\_string; i++)  
    {****int****from, to;  
        fin >> from >> to;  
        fin >> sm->m\_smej\_matr[to][from];  
    }  
   
    fin.close();****return****sm;  
}****void****PrintGraph(SmejMatr\* \_sm)  
{  
    INFO("Матрица смежности");  
    cout << "Количество вершин = " << \_sm->m\_numb\_of\_vertexes << endl;****for****(****int****i = 0; i < \_sm->m\_numb\_of\_vertexes; i++)  
    {****for****(****int****j = 0; j < \_sm->m\_numb\_of\_vertexes; j++)  
        {  
            cout << '\t' <<  
                (\_sm->m\_smej\_matr[i][j] == INT\_MAX ?  
                    "inf" : to\_string(\_sm->m\_smej\_matr[i][j]));  
        }  
        cout << endl;  
    }  
}****void****ConverOrientedIntoDisoriened(SmejMatr\*& \_sm)  
{  
    INFO("Конвертация ориентированного графа в неориентированный");****for****(****int****i = 0; i < \_sm->m\_numb\_of\_vertexes; i++)  
    {****for****(****int****j = 0; j < \_sm->m\_numb\_of\_vertexes; j++)  
        {****if****(\_sm->m\_smej\_matr[i][j] != 0)  
            {  
                \_sm->m\_smej\_matr[j][i] =  
                    \_sm->m\_smej\_matr[i][j];  
            }****else*** *{  
                \_sm->m\_smej\_matr[i][j] =  
                    \_sm->m\_smej\_matr[i][j] == 0 ? INT\_MAX : \_sm->m\_smej\_matr[i][j];  
            }  
        }  
    }  
}****string****ChangeDistElem(****int****\_elem)  
{****return****(\_elem == INT\_MAX ? "inf" : to\_string(\_elem));  
}****string****ChangeVisitElem(****int****\_elem)  
{****return****to\_string(\_elem) + " ";  
}  
   
// проверка на номер команды****bool****CheckNumber(****int****\_num)  
{****return****1 <= \_num && \_num <= 7;  
}  
   
// пример готовой программы****void****ExampleProgram()  
{  
    // считывание матрицы смежности из файла  
    SmejMatr\* sm = ReadGraphFromFile();  
   
    // позиция, от которой ищется путь****int****pos = 1;  
   
    // поиск кратчайшего пути  
    Diykstra(sm, pos);  
   
    // печать матрицы смежности  
    PrintGraph(sm);  
   
    // прима  
    ConverOrientedIntoDisoriened(sm);  
    PrintGraph(sm);  
    Prima(sm, pos);  
}  
   
template<****class****T =****int****>  
T\*\* MemAlloc(****int****\_size\_x,****int****\_size\_y)  
{  
    T\*\* arr =****new****T \* [\_size\_x];****for****(****int****i = 0; i < \_size\_x; i++)  
    {  
        arr[i] =****new****T[\_size\_y];  
    }****return****arr;  
}  
   
// ввод и проверка значений  
template<typename T,****class****FUNC>  
T InputAndCheck(FUNC \_comp,****string****welcome\_str,****string****err\_str)  
{  
    // размер массива  
    T symb;  
   
    // вывод сообщения  
    cout << welcome\_str << "\n";  
    cin >> symb;  
   
    // если было введено некорректное значение****if****(!\_comp(symb)) {  
        // если была введено не то, что нужно было****if****(cin.fail())  
        {  
            cin.clear();  
            cin.ignore(INT\_MAX, '\n');  
        }  
        cout << err\_str << "\n";  
   
        // рекурсивное обращение  
        symb = InputAndCheck<T, FUNC>(\_comp, welcome\_str, err\_str);  
    }****return****symb;  
}  
   
// функция ведения диалога с пользователем****void****Dialog()  
{  
    // переменная содержащая коды действий  
    input\_codes in\_code;  
   
    // временное число****int****temp;  
   
    // матрица смежности  
    SmejMatr\* sm = nullptr;****do*** *{  
        // запрос у пользователя следующих действий  
        in\_code = input\_codes(  
            InputAndCheck<****int****>(CheckNumber, command\_str)  
            - 1);  
   
        // запуск соответствующих функций****switch****(in\_code)  
        {****case****input\_codes::exit:  
            INFO("Произведен выход");****break****;****case****input\_codes::template\_program:  
            INFO("Запуск примера кода");  
            ExampleProgram();****break****;****case****input\_codes::find\_short\_path\_diykstra:  
            // счтитывание из файла  
            delete sm;  
            sm = ReadGraphFromFile();****if****(!sm)  
            {  
                INFO("ДИАЛОГ ДЕЙКСТРА: матрица смежности не существует");  
            }****else*** *{  
                temp = InputAndCheck<****int****>([&sm](****int****el)  
                    {****return****el >= 1 & el <= sm->m\_numb\_of\_vertexes;  
                    },  
                    "Введите номер вершины, от которой будет происходить поиск расстояния: ",  
                        "Номер должен быть в пределах: [1," + to\_string(sm->m\_numb\_of\_vertexes) + "]");  
                Diykstra(sm, temp - 1);  
            }****break****;****case****input\_codes::create\_min\_spanning\_tree\_prim:  
            INFO("Остовное дерево минимальной стоимости");****if****(!sm)  
            {  
                INFO("ДИАЛОГ ДЕЙКСТРА: матрица смежности не существует");  
            }****else*** *{  
                temp = InputAndCheck<****int****>([&sm](****int****el)  
                    {****return****el >= 1 & el <= sm->m\_numb\_of\_vertexes;  
                    },  
                    "Введите номер вершины, от которой будет строиться остовное дерево: ",  
                        "Номер должен быть в пределах: [1," + to\_string(sm->m\_numb\_of\_vertexes) + "]");  
                ConverOrientedIntoDisoriened(sm);  
                PrintGraph(sm);  
                Prima(sm, temp-1);  
            }****break****;****case****input\_codes::delete\_smej\_matr:  
            INFO("Удаление матрицы смежности");  
            delete sm;  
            sm = nullptr;****break****;****case****input\_codes::print\_smej\_matr:  
            INFO("Печать матрицы смежности");****if****(!sm) { INFO("ДИАЛОГ ПЕЧАТЬ: матрица смежности не существует"); }****else*** *{  
                PrintGraph(sm);  
            }****break****;****case****input\_codes::clear\_console:  
            system("cls");****break****;****default****:  
            INFO("Неизвестный код");****break****;  
        }  
   
    }****while****(  
        // пока пользователь не захотел выйти из программы  
        // или пока не запустил пример программыЫ  
        in\_code != input\_codes::exit &&  
        in\_code != input\_codes::template\_program  
        );  
}  
   
template<****class****T,****class****FUNC>****void****PrintArr(T\* \_arr,****int****\_size, FUNC \_prt)  
{****for****(****int****i = 0; i < \_size; i++)  
    {  
        cout << '\t' << \_prt(\_arr[i]);  
    }  
    cout << endl;  
}  
   
template<****class****T>****void****SetMartixValue(T\*\* \_mat,****int****\_sizex,****int****\_sizey, T \_value)  
{****for****(****int****i = 0; i < \_sizex; i++)  
    {****for****(****int****j = 0; j < \_sizey; j++)  
        {  
            \_mat[i][j] = \_value;  
        }  
    }  
}****void****Diykstra(SmejMatr\* \_sm,****int****\_start\_vert)  
{  
    INFO("Алгоритм Дейкстры");  
   
    // количество вершин****int****size = \_sm->m\_numb\_of\_vertexes;  
   
    // кратчайшие пути до вершин относительно вершины \_start\_vert****int****\* short\_distance =****new******int****[size];  
   
    // хранение информации о посещенных вершинах****bool****\* visited\_node =****new******bool****[size];  
   
    // минимальное расстояние****int****min = INT\_MAX;  
    // индекс элемента с минимальным расстоянием****int****minindex = INT\_MAX;  
   
    // изначально все вершины делаем непосещенными  
    // и расстояние до каждой делаем максимално возможным****for****(****int****i = 0; i < size; i++)  
    {  
        short\_distance[i] = INT\_MAX;  
        visited\_node[i] =****false****;  
    }  
    cout << "расстояния: ";  
    PrintArr(short\_distance, size, ChangeDistElem);  
   
    cout << " посещения: ";  
    PrintArr<****bool****>(visited\_node, size, ChangeVisitElem);  
   
    // расстояние для вершины, из которой идем, обнуляем  
    short\_distance[\_start\_vert] = 0;  
   
    // шаг алгоритма****for****(****int****count = 0; count < size; count++)  
    {  
        cout << "ШАГ: " << count + 1 << endl;  
   
        // поиск непосещенной вершины с минимальным расстоянием  
        min = INT\_MAX;****for****(****int****i = 0; i < size; i++)  
        {****if****(!visited\_node[i] && short\_distance[i] < min)  
            {  
                min = short\_distance[i];  
                minindex = i;  
            }  
        }  
   
        // отмечаем посещенным элемент с индексом minindex  
        visited\_node[minindex] =****true****;  
   
        // если у элемента есть какое-то расстояние****if****(short\_distance[minindex] != INT\_MAX)  
        {  
            // ищем смежные элементы с minindex  
            // и меняем минимальное расстояние до смежных вершин  
            // если это необходимо****for****(****int****i = 0; i < size; i++)   // i - вершина  
            {****if****(  
                    // если существует ребро между i и minindex  
                    \_sm->m\_smej\_matr[minindex][i] &&  
                    // если вершина не была посещена  
                    !visited\_node[i] &&  
                    // если расстояние от \_start\_vert до i >  
                    // расстояния от \_start\_vert до minindex +  
                    // от minindex до i  
                    short\_distance[minindex] + \_sm->m\_smej\_matr[minindex][i]  
                    < short\_distance[i]  
                    )  
                {  
                    short\_distance[i] = short\_distance[minindex] + \_sm->m\_smej\_matr[minindex][i];  
                }  
            }  
        }  
   
        cout << "расстояния: ";  
        PrintArr(short\_distance, size, ChangeDistElem);  
   
        cout << " посещения: ";  
        PrintArr<****bool****>(visited\_node, size, ChangeVisitElem);  
    }  
   
    // Восстановление путей до вершин  
    cout << "\nКратчайшие пути\n";  
    // массив с путем****int****\* out\_path =****new******int****[size];****for****(****int****i = 0; i < size; i++)  
    {  
        // конечная вершина****int****end = i;  
        // записываем конечную вершину в массив с путем  
        out\_path[0] = end + 1;  
        // счетчик, для правильной записи элементов в массив с путем****int****k = 1;  
        // длина пути до конечной вершины****int****weight = short\_distance[end];  
   
        // пока не дошли до начальной вершины и  
        // пока есть путь от начальной до конечной вершины****while****(end != \_start\_vert && short\_distance[end] != INT\_MAX)  
        {  
            // ищем смежные вершины с конечной****for****(****int****i = 0; i < size; i++)  
            {****if****(\_sm->m\_smej\_matr[i][end])  
                {  
                    // если нашли смежную вершину  
                    // проверяем совпадет ли вес текущей вершины  
                    // с весом конечной - ребро, смежное с конечной и текущей****if****(weight - \_sm->m\_smej\_matr[i][end] == short\_distance[i])  
                    {  
                        // вычитаем вес  
                        weight -= \_sm->m\_smej\_matr[i][end];  
                        // обновляем конечную вершину  
                        end = i;  
                        // добавляем эту вершину в массив с путем  
                        out\_path[k] = i + 1;  
                        k++;  
                    }  
                }  
            }  
        }  
   
        // вывод последовательности с длиной  
        // если путь существует до вершины  
        // и вершина i не является начальной****if****(short\_distance[i] != INT\_MAX && i != \_start\_vert)  
        {  
            // печатаем путь****for****(****int****j = k - 1; j >= 0; j--)  
            {  
                cout << out\_path[j];****if****(j > 0)   cout << " > ";  
            }  
            cout << " = " << short\_distance[i] << endl;  
        }  
        // иначе говорим, что пути нет****else*** *{  
            cout << \_start\_vert + 1 << " > " << i + 1 << " = " << "маршрут недоступен" << endl;  
        }  
    }  
   
    // чистим память  
    delete[] out\_path;  
    delete[] visited\_node;  
    delete[] short\_distance;  
}****void****Prima(SmejMatr\* \_sm,****int****\_st)  
{  
    INFO("Алгоритм Прима");  
    // количество вершин****int****size = \_sm->m\_numb\_of\_vertexes;  
   
    // пройденные вершины****bool****\* nodes =****new******bool****[size];****for****(****int****i = 0; i < size; i++)  
    {  
        nodes[i] =****false****;  
    }  
   
    //Количество рассмотренных вершин****int****num\_of\_nodes = 0;  
    //Берем первую вершину  
    nodes[\_st] =****true****;  
    //Связанные вершины, минимум****int****x, y, min;****int****weight = 0;  
    cout << "Минимальное остовное дерево" << endl;  
   
    //Алгоритм Прима****while****(num\_of\_nodes < size - 1) {  
        x = 0;  
        y = 0;  
        min = INT32\_MAX;****for****(****int****i = 0; i < size; i++) {****if****(nodes[i]) {****for****(****int****j = 0; j < size; j++) {  
                    //Еще не выбирали эту вершину и существует связь между ними****if****(!nodes[j] && \_sm->m\_smej\_matr[i][j] < INT\_MAX) {****if****(\_sm->m\_smej\_matr[i][j] < min) {  
                            min = \_sm->m\_smej\_matr[i][j];  
                            x = i;  
                            y = j;  
                        }  
                    }  
                }  
            }  
        }  
        weight += min;  
        cout << x + 1 << " > " << y + 1 << " = " << \_sm->m\_smej\_matr[x][y] << endl;  
        nodes[y] =****true****;  
        num\_of\_nodes++;  
    }  
    cout << "Вес минимального остовного дерева равен " << weight << endl;  
   
    // вывод матрицы  
   
}  
   
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* End Of main.cpp File \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/*

# Тестирование программы

### Тест 1





# Вывод

Работа программы завершена на основании:

1) Полученные результаты совпали с ожидаемыми;

2) Считаем набор тестов полным.