Министерство образования Российской Федерации

Пензенский государственный университет

Кафедра «Вычислительная техника»

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

к курсовому проектированию

по курсу «Логика и основы алгоритмизации

в инженерных задачах»

На тему: «Реализация алгоритма Прима»

Выполнил:

студент группы 20ВВ2

Зиновьев Я.М.

Принял:

д.т.н. Митрохин М.А.

Пенза 2021

**Содержание**

[Реферат 3](#_Toc91705871)

[Введение 4](#_Toc91705872)

[Постановка задачи 5](#_Toc91705873)

[Теоретическая часть задания 6](#_Toc91705874)

[Описание алгоритма программы 7](#_Toc91705875)

[Описание программы 7](#_Toc91705876)

[Ручной расчет 10](#_Toc91705877)

[Тестирование 10](#_Toc91705878)

[Заключение 12](#_Toc91705879)

[Список литературы 13](#_Toc91705880)

[Листинг программы 14](#_Toc91705881)

# Реферат

Отчёт 16 страниц, 9 рисунков.

ГРАФ, ТЕОРИЯ ГРАФОВ, ОСТОВНОЕ ДЕРЕВО, АЛГОРИТМ ПРИМА.

Цель исследования – разработка программы, способной построить минимальное остовное дерево взвешенного связного неориентированного графа.

В курсовой работе будет рассмотрен алгоритм, при помощи которого можно найти минимальное остовное дерево графа, а также все правила работы с ним. Установлено, что с помощью данного алгоритма можно найти минимальное остовное дерево графа, при любой взвешенности (разнице в весах рёбер графа) и большом количестве вершин.

# Введение

Алгоритм Прима позволяет построить минимальное остовное дерево взвешенного связного неориентированного графа. Данный алгоритм был переоткрыт (открыт впервые, в 1930 г., Войтехом Ярником) Робертом Примом в 1957 г.

В качестве языка программирования был выбран Си++. Средой разработки была выбрана Microsoft Visual Studio 2019.

Целью данной курсовой работы является разработка программы реализующей алгоритма Прима на языке С++, который является широко используемым языком программирования. Именно с его помощью в курсовой работе будет реализован алгоритм Прима.

# Постановка задачи

Требуется разработать программу, которая будет выполнять алгоритм Прима, строить минимальное остовное дерево взвешенного связного неориентированного графа, а также выводить результат пользователю.

Исходные графы должны быть представлены в виде неориентированных взвешенных графов смежности, при генерации которых предусматриваются определённые ограничения по размеру. После выполнения алгоритма, на экран пользователя должно выводиться остовное дерево, которое получилось в результате работы данного алгоритма. Также, необходимо предусмотреть разные исходы расчёта алгоритма, чтобы в программе отсутствовали ошибки и некорректные результаты.

Устройство ввода – клавиатура и мышь.

# 

# Теоретическая часть задания

Алгоритм Прима — алгоритм поиска [минимального остовного дерева](https://neerc.ifmo.ru/wiki/index.php?title=%D0%9B%D0%B5%D0%BC%D0%BC%D0%B0_%D0%BE_%D0%B1%D0%B5%D0%B7%D0%BE%D0%BF%D0%B0%D1%81%D0%BD%D0%BE%D0%BC_%D1%80%D0%B5%D0%B1%D1%80%D0%B5#.D0.9C.D0.B8.D0.BD.D0.B8.D0.BC.D0.B0.D0.BB.D1.8C.D0.BD.D0.BE.D0.B5_.D0.BE.D1.81.D1.82.D0.BE.D0.B2.D0.BD.D0.BE.D0.B5_.D0.B4.D0.B5.D1.80.D0.B5.D0.B2.D0.BE) во взвешенном [неориентированном связном графе](https://neerc.ifmo.ru/wiki/index.php?title=%D0%9E%D1%81%D0%BD%D0%BE%D0%B2%D0%BD%D1%8B%D0%B5_%D0%BE%D0%BF%D1%80%D0%B5%D0%B4%D0%B5%D0%BB%D0%B5%D0%BD%D0%B8%D1%8F_%D1%82%D0%B5%D0%BE%D1%80%D0%B8%D0%B8_%D0%B3%D1%80%D0%B0%D1%84%D0%BE%D0%B2#.D0.9D.D0.B5.D0.BE.D1.80.D0.B8.D0.B5.D0.BD.D1.82.D0.B8.D1.80.D0.BE.D0.B2.D0.B0.D0.BD.D0.BD.D1.8B.D0.B5_.D0.B3.D1.80.D0.B0.D1.84.D1.8B).

В алгоритме Прима на каждом шаге рассматривается структура дерево. Вначале это дерево состоит из единственной вершины, в качестве которой может быть выбрана любая вершина графа. Затем к дереву последовательно добавляются ребра и вершины, пока не получится остовное дерево, т.е. каркас. Для того чтобы из текущего дерева при добавлении нового ребра опять получилось дерево, это новое ребро должно соединять вершину дерева с вершиной, еще не принадлежащей дереву. Такие ребра будем называть подходящими относительно рассматриваемого дерева. В алгоритме Прима применяется следующее правило выбора: на каждом шаге из всех подходящих ребер выбирается ребро наименьшего веса. Это ребро вместе с одной новой вершиной добавляется к дереву.

# Описание алгоритма программы

Для работы алгоритма, нам необходимо иметь связанный взвешенный неориентированный граф. Он может быть представлен в классической форме подобных графов – матрице смежности.

В начале работы алгоритма, берётся произвольная вершина графа и находится инцидентное ей ребро, которое обладает наименьшим весом. Если проецировать это на матрицу смежности, мы выбираем случайную строку (столбец) и начинаем поиск минимального значения в представленном ряду.

Данное ребро и вершины, соединяемые им, образуют начальное дерево. Затем, мы рассматриваем рёбра графа, один конец которых – уже принадлежащая дереву вершина, а другой конец – нет. Мы выбираем из данных рёбер наименьшие по весу. Когда такое ребро найдено, оно присоединяется к дереву. Рост остовного дерева будет продолжаться, пока все вершины графа не будут присоединены к дереву или пока не закончатся возможные рёбра (куда можно пройти).

В результате, мы получим минимальное (по весу) остовное дерево.

# Описание программы

Для написания данной программы использован язык программирования Си. Проект был создан в виде консольного приложения Win32 (Visual C++).

Данная программа является многомодульной, поскольку состоит из нескольких функций: printmatrix(), generatematrix().

В начале программы, пользователю предлагается ввести кол-во вершин в генерируемом графе:

printf("Введите количество вершин графа: ");

scanf\_s("%d", &N);



Рисунок 1. Ввод количества вершин графа

После пользовательского ввода, программа выделяет место под четыре массива данных: G – матрица смежности графа, used – массив пройденных вершин, min\_e – массив минимальных весов для i ребра и sel\_e – массив с концами (вершинами) наименьших рёбер (необходимо для конечного результата).

G = (int\*\*)malloc(N \* sizeof(int\*));

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

G[i] = (int\*)malloc(N \* sizeof(int));

used = (bool\*)malloc(N \* sizeof(bool));

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

used[i] = false;

min\_e = (int\*)malloc(N \* sizeof(int));

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

min\_e[i] = 1000000000;

sel\_e = (int\*)malloc(N \* sizeof(int));

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

sel\_e[i] = -1;

Когда программа выделила память под нужные массивы данных, она может приступать к генерации матрицы смежности. Это происходит в функции generatematrix():

generatematrix(10);

Для проверки правильности генерации, матрица, при помощи функции printmatrix(), выводится в консоль пользователя. При неудачной генерации, пользователь может перезапустить программу и начать генерацию заново.

printf("\nИсходная матрица графа G:\n");

printmatrix(G, N);

\_getch();

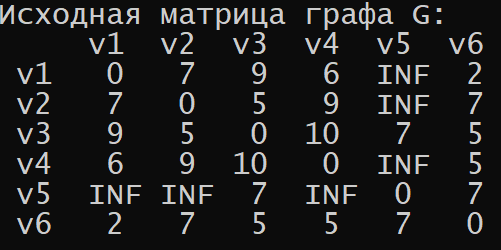


Рисунок 2. Вывод матрицы графа

После успешной генерации и проверки матрицы, мы можем приступить к выводу остовного дерева. Количество повторов поиска минимального ребра равно кол-ву вершин в графе (основной цикл):

min\_e[0] = 0;

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

{

Для начала, необходимо определить, какие вершины мы успели посетить. Мы выбираем нужную вершину при помощи следующего цикла:

int v = -1;

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

if (!used[j] && (v == -1 || min\_e[j] < min\_e[v]))

v = j;

Если длина минимальной вершины равна INF (несуществующий путь / 1000000000), то мы прекращаем строить дерево:

if (min\_e[v] == INF)

{

printf("\nNo MST!\n");

exit(0);

}

В ином случае, мы помечаем вершину, как посещённую и выводим данные о минимальном ребре (начальная вершина, вес, конечная вершина):

used[v] = true;

if (sel\_e[v] != -1)

cout <<"Нач. вершина: " << v << " Вес:" << min\_e[v] << " Кон. вершина: " << sel\_e[v]+1 << endl;

Имея информацию о конечной вершине, мы ищем её минимальное инцидентное ребро, которое не будет связано с уже посещённой вершиной.

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

if (G[v][to] < min\_e[to])

{

min\_e[to] = G[v][to];

sel\_e[to] = v;

}

}

}

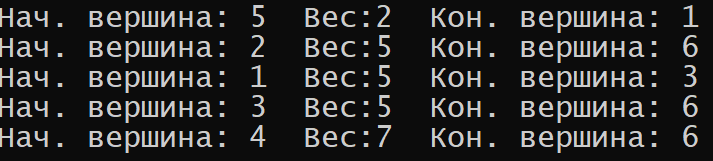


Рисунок 3. Вывод «ветвей» остовного дерева

В дальнейшем цикл будет повторяться до тех пор, пока он не пройдёт все возможные вершины или не упрётся в значение INF (несуществующий путь).

Полный код программы можно увидеть в разделе Листинг программы.

# Ручной расчет

Проведем проверку программы посредством ручных вычислений на

примере графа с 5 вершинами (рисунок 4).

Изначально остов состоит из вершины v5 из нее выбираем ребро v (v1:v5)= 3 имеющее минимальный вес, из вершины v1 выбираем ребро с минимальным весом (v1:v4) =4, v4 не имеет других ребер, из v1 по аналогии находим вершину v2 (v1:v2) =4 с минимальным весом, v2 имеет 2 ребра, так же оно связано с v3 (v2:v3) =3. Из v1 имеется путь в v3 (v1:v3) =9. Сравнив вес из v1 в v3 и вес пути v1:v2:v3 мы удаляем ребро v1:v3, так как путь v1:v2:v3 имеет меньший вес. Таким образом с помощью алгоритма Прима построили минимальное остовное дерево.

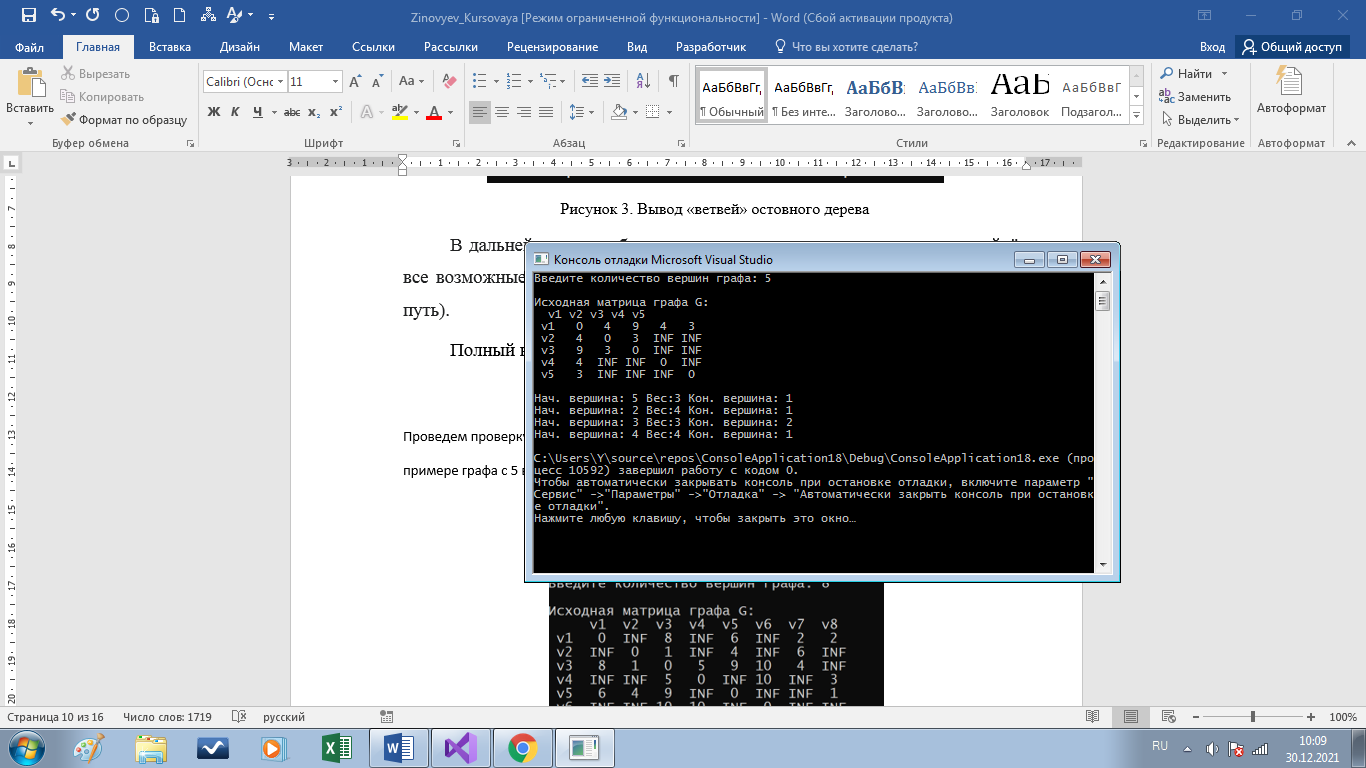


Рисунок 4. Тестирование работы программы

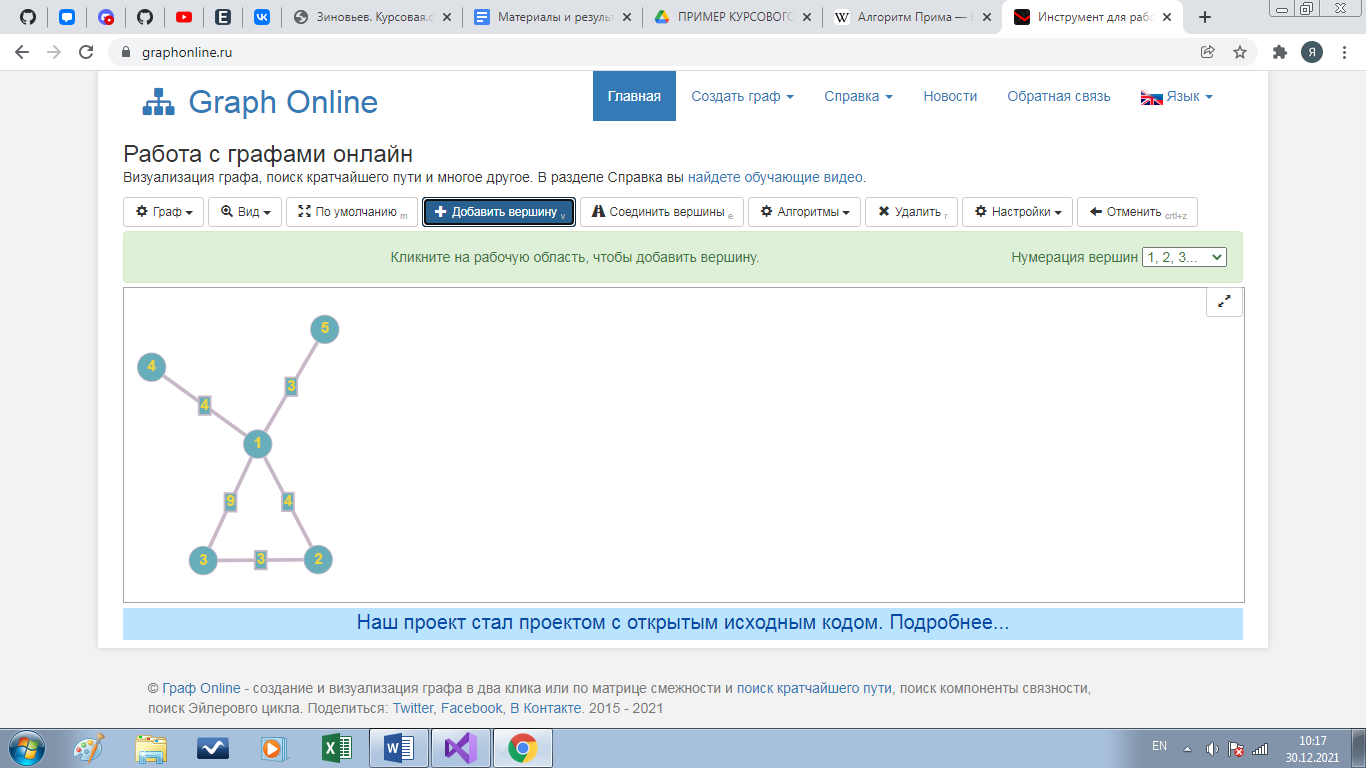


Рисунок 5. Неориентированный граф (по рисунку 4)

# Тестирование

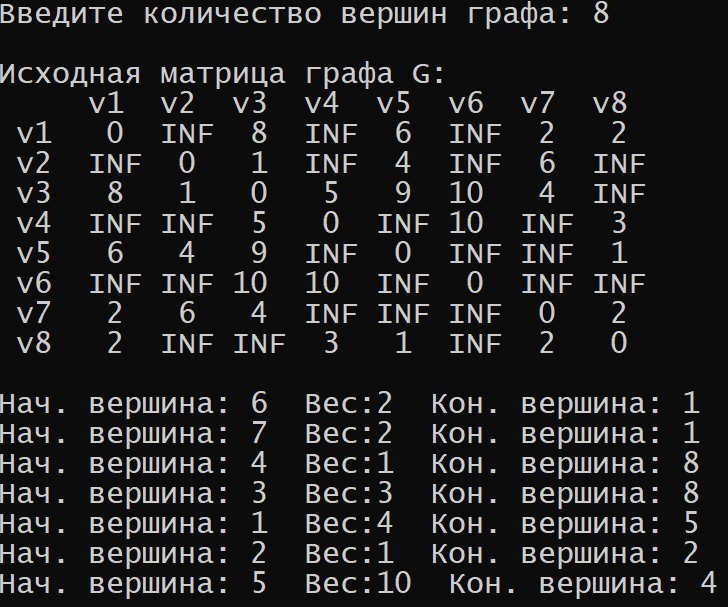


Рисунок 6. Тест №1

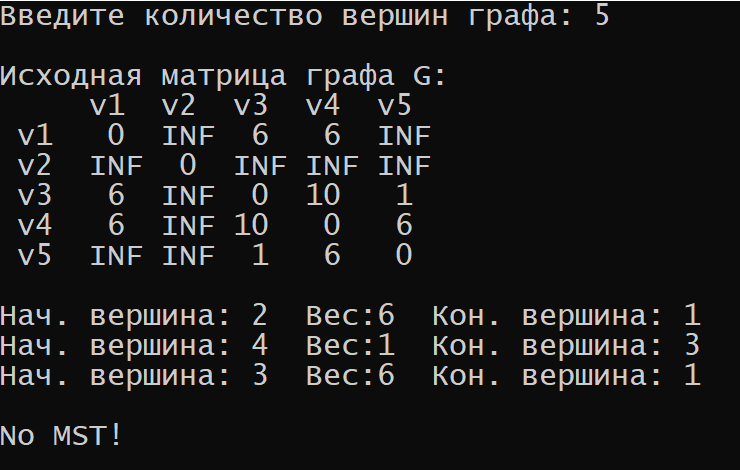


Рисунок 7. Тест №2

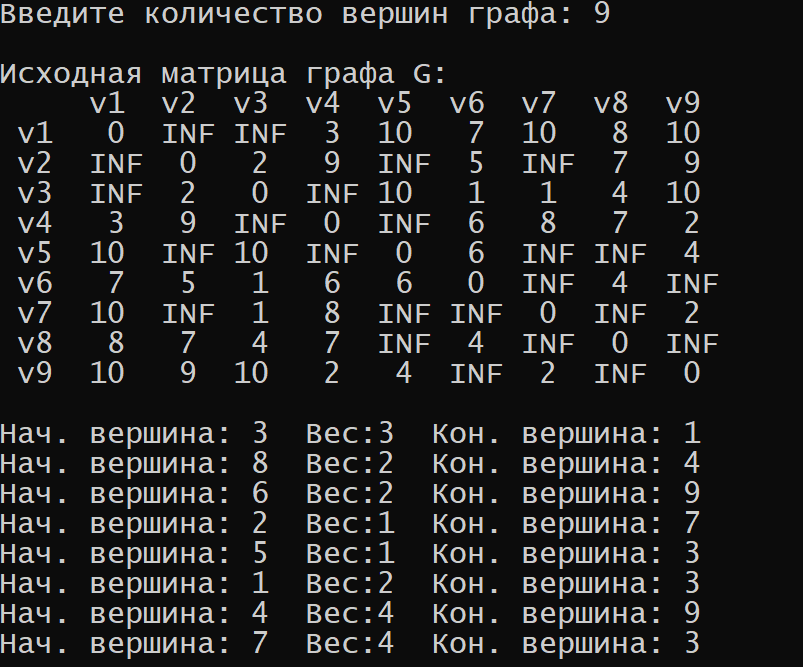


Рисунок 8. Тест №3

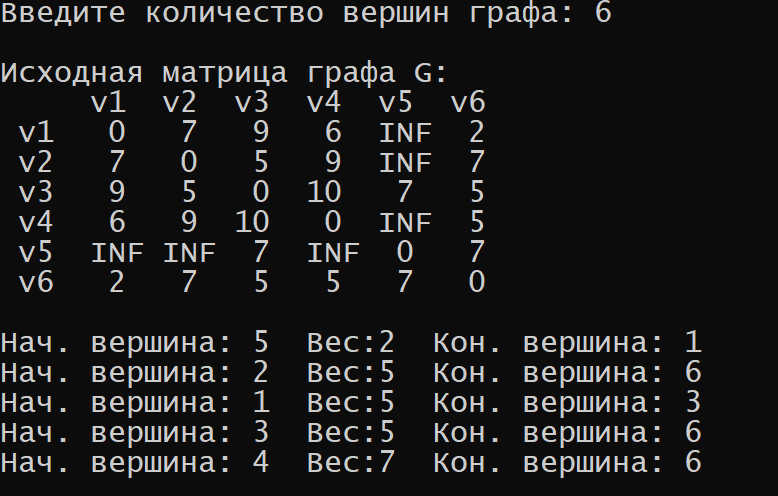


Рисунок 9. Тест №4

Таблица 1 – Описание тестирования программы

|  |  |  |  |
| --- | --- | --- | --- |
| Описание теста | Действие | Ожидаемый результат | Полученный результат |
| Запуск программы | Нажатие на ярлык | Вывод сообщения: «Введите кол-во вершин графа: » | Верно |
| Ввод данных | Данные о кол-ве вершин в графе, Enter | Вывод сгенерированной матрицы смежности | Верно |
| Пуск алгоритма | Enter | Вывод строк с «ветвями» минимального остовного дерева | Верно |
| Выход из программы | Enter | Программа успешно завершается | Верно |

# Заключение

Таким образом, в процессе создания данного проекта разработана программа, реализующая алгоритм Прима в Microsoft Visual Studio 2019.

При выполнении данной курсовой работы были получены навыки разработки программ и освоены приемы создания неориентированных взвешенных графов в матрицах смежности. Приобретены навыки построения остовных деревьев при помощи алгоритма Прима. Улучшены знания языка программирования C++.

Недостатком данной программы является достаточно примитивный пользовательский интерфейс. Из-за работы программы в консольном режиме, при подсчёте сложности, мы не учитываем программный интерфейс.

Программа имеет небольшой функционал, однако, он достаточен для выполнения поставленной задачи.

# Список литературы

1. Петрунин А. [Сколько деревьев в графе](http://kvant.mccme.ru/pdf/2018/2018-09.pdf) // [Квант](https://ru.wikipedia.org/wiki/%D0%9A%D0%B2%D0%B0%D0%BD%D1%82_(%D0%B6%D1%83%D1%80%D0%BD%D0%B0%D0%BB)). — 2018. — № 9. — С. 9—13.

2. [Кормен, Т.](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D1%80%D0%BC%D0%B5%D0%BD,_%D0%A2%D0%BE%D0%BC%D0%B0%D1%81), [Лейзерсон, Ч.](https://ru.wikipedia.org/wiki/%D0%9B%D0%B5%D0%B9%D0%B7%D0%B5%D1%80%D1%81%D0%BE%D0%BD,_%D0%A7%D0%B0%D1%80%D0%BB%D1%8C%D0%B7_%D0%AD%D1%80%D0%B8%D0%BA), [Ривест, Р.](https://ru.wikipedia.org/wiki/%D0%A0%D0%B8%D0%B2%D0%B5%D1%81%D1%82,_%D0%A0%D0%BE%D0%BD%D0%B0%D0%BB%D1%8C%D0%B4_%D0%9B%D0%B8%D0%BD%D0%BD), [Штайн, К.](https://ru.wikipedia.org/wiki/%D0%A8%D1%82%D0%B0%D0%B9%D0%BD,_%D0%9A%D0%BB%D0%B8%D1%84%D1%84%D0%BE%D1%80%D0%B4) Глава 23. Минимальные остовные деревья // Алгоритмы: построение и анализ = Introduction to Algorithms / Под ред. И. В. Красикова. — 2-е изд. — М.: Вильямс, 2005. — 1296 с

3. Язык Си: Б.В. Керниган, Д.М. Ричи - Санкт-Петербруг, Невский диалект, 2003г.

4. Буркатовския Ю. Б. Теория графов. — Томск: Издательство Томского политехнического университета, 2014. — Т. 1. — 200 с.

5. Кормен Т. М. и др. Часть VI. Алгоритмы для работы с графами // Алгоритмы: построение и анализ = Introduction to Algorithms. — 2-е изд. — М.: Вильямс, 2006. — С. 1296.

# Листинг программы

#include <iostream>

#include <stdio.h>

#include <Windows.h>

#include <time.h>

#include <conio.h>

#include <locale>

#include <stdlib.h>

#include <vector>

#define INF = 1000000000; // значение "бесконечность"

int\*\* G, \* min\_e, \* sel\_e;

bool\* used;

int N, z, i, j;

using namespace std;

void printmatrix(int\*\* source, int size);

void generatematrix(int range);

int main()

{

SetConsoleCP(1251); //Подключение русского языка

SetConsoleOutputCP(1251);

printf("Введите количество вершин графа: ");

scanf\_s("%d", &N); //Ввод кол-ва вершин графа

G = (int\*\*)malloc(N \* sizeof(int\*)); //Выделяем память под исходную матрицу графа

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

G[i] = (int\*)malloc(N \* sizeof(int));

used = (bool\*)malloc(N \* sizeof(bool));

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

used[i] = false;

min\_e = (int\*)malloc(N \* sizeof(int));

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

min\_e[i] = 1000000000;

sel\_e = (int\*)malloc(N \* sizeof(int));

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

sel\_e[i] = -1;

//Заполняем исходную матрицу для взвешенного неориентированного графа

generatematrix(10);

printf("\nИсходная матрица графа G:\n"); //Вывод матрицы смежности

printmatrix(G, N);

printf("\n");

\_getch();

min\_e[0] = 0;

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

{

int v = -1;

for (int j = 0; j < N; ++j) //Проверяем, является ли вершина пройденной или нет(true/false)

if (!used[j] && (v == -1 || min\_e[j] < min\_e[v]))

v = j;

if (min\_e[v] == 1000000000) //Если не можем строить дерево дальше, то завершаем программу

{

printf("\nNo MST!\n");

exit(0);

}

used[v] = true; //Отмечаем вершину, как посещённую

if (sel\_e[v] != -1)

cout <<"Нач. вершина: " << v << " Вес:" << min\_e[v] << " Кон. вершина: " << sel\_e[v]+1 << endl;

for (int to = 0; to < N; ++to) { //Ищем самое короткое ребро у концевой вершины

if (G[v][to] < min\_e[to])

{

min\_e[to] = G[v][to];

sel\_e[to] = v;

}

}

}

}

void printmatrix(int\*\* source, int size)

{

printf(" ");

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

printf(" v%d", i + 1);

printf("\n");

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

{

printf(" v%d ", i + 1);

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

if (G[i][j] == 1000000000)

printf(" INF");

else

printf("%3d ", source[i][j]);

printf("\n");

}

}

void generatematrix(int range) {

srand(time(NULL));

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

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

if (i == j) //Ячейки по диоганали 0, т.к нет циклов

G[i][j] = 0;

else

{

if (rand() % range < 6) //Вероятность заполнения числом 4/10

G[i][j] = 1 + rand() % range; //В ячейке будет от 0 до 10

else //Вероятность отсутсвия пути (0) 6/10

G[i][j] = 1000000000;

G[j][i] = G[i][j]; //Т.к граф - неориент., происходит зеркальное заполнение

}

}