**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ**

федеральное государственное бюджетное образовательное учреждение высшего образования «Казанский национальный исследовательский

технический университет им. А.Н. Туполева-КАИ»

(КНИТУ-КАИ)

Институт компьютерных технологий и защиты информации

Кафедра систем автоматизированного проектирования

09.03.01. «Информатика и вычислительная техника»

КУРСОВАЯ РАБОТА

по дисциплине: «Основы систем автоматизированного проектирования»  
на тему: «Алгоритм Борувки для построения кратчайшего остовного дерева.»

Обучающийся 4214 \_\_\_\_\_\_\_\_\_\_\_\_ Бикмуллин А.Р.

(номер группы) (подпись, дата) (Ф.И.О.)

Руководитель старший преподаватель Суздальцев И.В.

(должность) (Ф.И.О.)

Курсовая работа (проект) зачтена (зачтен) с оценкой \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись, дата)

Казань 2022

**СОДЕРЖАНИЕ**

1. Введение......................................................................................................3
2. Содержательная постановка задачи..........................................................4
3. Математическая постановка задачи..........................................................5
4. Описание алгоритма решения задачи.......................................................7
5. Решение задачи на контрольном примере..............................................11
6. Описание программы................................................................................16
7. Примеры решения задач о нахождении минимального остовного

дерева с помощью нашей программы.....................................................18

1. Заключение................................................................................................25
2. Источники..................................................................................................26

10. Приложения...............................................................................................27

**Введение**

В данной курсовой работе проводится полный анализ алгоритма Борувки для построения минимального остовного дерева.

Выполнена содержательная постановка задачи, математическая постановка задачи, произведено описание алгоритма решения задачи и решение задачи на контрольном примере.

Также мы выяснили насколько алгоритм актуален в наши дни, и где он используется.

Выполнена разработка программы, которая способна решать задачу (строить минимальное остовное дерево) используя в основе алгоритм Борувки. Программа написана на языке разработки - С++

**2 Содержательная постановка задачи**

Исходными данными решаемой задачи являются взвешенный полный граф с определенными вершинами и весом ребер между ними. Вес ребра есть длина ребра, соединяющей одну вершину с другой.

Остовное дерево состоит из некоторого подмножества рёбер графа, таких, что из любой вершины графа можно попасть в любую другую вершину, двигаясь по этим рёбрам, и в нём нет циклов, то есть из любой вершины нельзя попасть в саму себя, не пройдя какое-то ребро дважды.

В неформальной форме задача о нахождении кратчайшего остовного дерева трактуется следующим образом: Связать все существующие в данном графе вершины так, чтобы суммарный вес связывающих ребер был минимальным. Важные условия: в получившимся графе не должно быть циклов и каждая вершина должна быть связана

Области применения:

Задача о нахождении минимального остовного дерева часто встречается решении логистических задач по транспортировке: допустим, есть *n* городов, которые необходимо соединить дорогами, так, чтобы можно было добраться из любого города в любой другой (напрямую или через другие города). Разрешается строить дороги между заданными парами городов и известна стоимость строительства каждой такой дороги. Требуется решить, какие именно дороги нужно строить, чтобы минимизировать общую стоимость строительства.

**3 Математическая постановка задачи**

Для математической (формальной) постановки задачи, введём следующие представленные в таблице 1.

|  |  |  |
| --- | --- | --- |
| **Обозначения элементов в математической модели задачи** | **Описание соответствующих элементов математической модели** | **Примечание** |
| G(V, U) | Исходный граф (карта коммивояжера) |  |
| V | Множества вершин (множество городов) |  |
| U | Множество ребер (множество дорог между городами) |  |
| n | Количество вершин графа (количество городов) |  |
| Dn×n | Матрица смежности графа (матрица расстояний между городами) |  |
| dij | Вес ребра, инцидентного i-ой и j-ой вершинам графа (длина дороги между i-ым и j-ым городами) | i = 1.. n;  j = 1.. n;  i≠j |
| Xn×n | Матрица, определяющая перечень рёбер, включённых в маршрут коммивояжера |  |
| xij | Элемент матрицы X | i = 1.. n;  j = 1.. n;  i≠j |
| F | Целевая функция |  |

Таб. 1. Перечень обозначений элементов в математической модели задачи.

Тогда математическая постановка задачи о построении остовного дерева минимальной длинны имеет следующий вид:

Остовное дерево D\* связного графа G называется кратчайшим, если его вес d(D\*) является наименьшим среди весов всех остовных деревьев графа G.

Алгоритм построения кратчайшего остовного дерева:

Вход: связный граф G = (V, U), |V| = n, |U| = q,

функция весов d : U → R+.

Выход: какое-то кратчайшее остовное дерево D\* = (V, U\*) графа G, U\* ⊆ U

F=

**4 Описание алгоритма решения задачи**

### **1) Общее описание алгоритма**

Алгоритм Борувки — алгоритм поиска [минимального остовного дерева](https://neerc.ifmo.ru/wiki/index.php?title=%D0%9E%D1%81%D1%82%D0%BE%D0%B2%D0%BD%D1%8B%D0%B5_%D0%B4%D0%B5%D1%80%D0%B5%D0%B2%D1%8C%D1%8F:_%D0%BE%D0%BF%D1%80%D0%B5%D0%B4%D0%B5%D0%BB%D0%B5%D0%BD%D0%B8%D1%8F,_%D0%BB%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) во взвешенном неориентированном связном графе. Впервые был опубликован в 1926 году Отакаром Борувкой.

Работа алгоритма состоит из нескольких итераций, каждая из которых состоит в последовательном добавлении рёбер к [остовному лесу](https://ru.wikipedia.org/wiki/%D0%9E%D1%81%D1%82%D0%BE%D0%B2%D0%BD%D1%8B%D0%B9_%D0%BB%D0%B5%D1%81) графа, до тех пор, пока лес не превратится в [дерево](https://ru.wikipedia.org/wiki/%D0%94%D0%B5%D1%80%D0%B5%D0%B2%D0%BE_(%D1%82%D0%B5%D0%BE%D1%80%D0%B8%D1%8F_%D0%B3%D1%80%D0%B0%D1%84%D0%BE%D0%B2)), то есть, лес, состоящий из одной компоненты связности.

### **2) Математическое описание алгоритма**

Пусть задан связный неориентированный граф G=(V,E) с весами рёбер f(e). Предполагается, что веса всех рёбер различны (если это не так, то можно упорядочить рёбра сначала по весу, а потом по номеру).

Алгоритм Борувки основан на следующих двух свойствах задачи:

* Минимальное ребро фрагмента. Пусть F – фрагмент минимального остовного дерева и eF – ребро наименьшего веса, исходящее из F (т.е. ровно один его конец является вершиной из F). Если ребро eF единственно, то оно принадлежит минимальному остовному дереву.
* Схлопывание фрагментов. Пусть F – фрагмент минимального остовного дерева графа G, а граф G′ получен из G склеиванием вершин, принадлежащих F. Тогда объединение F и минимального остовного дерева графа G′ даёт минимальное остовное дерево исходного графа G.

В начале работы алгоритма каждая вершина графа G является отдельным фрагментом. На очередном шаге у каждого фрагмента выбирается исходящее ребро минимального веса (если такое ребро существует). Выбранные рёбра добавляются в минимальное остовное дерево, а соответствующие фрагменты склеиваются.

### **3) Вычислительное ядро алгоритма**

Основными операциями являются:

1. Поиск минимального по весу исходящего ребра в каждом фрагменте.
2. Объединение фрагментов.

### **3) Схема реализации последовательного алгоритма**

В алгоритме Борувки фрагменты минимального остовного дерева наращиваются постепенно присоединением минимального ребра, выходящего из каждого фрагмента.

1. В самом начале каждая вершина является отдельным фрагментом.
2. На каждом шаге:
3. Для каждого фрагмента определяется минимальное по весу исходящее ребро.
4. Минимальные рёбра добавляются в минимальное остовное дерево, а соответствующие фрагменты объединяются.
5. Алгоритм останавливается, когда остаётся только один фрагмент либо когда ни у одного из фрагментов нет исходящих рёбер.

Поиск минимальных исходящих рёбер может выполняться независимо для каждого фрагмента. Таким образом, данную стадию вычислений можно эффективно параллелизовать

Аккуратный подсчёт количества активных фрагментов позволяет остановить алгоритм Борувки на один шаг раньше обычного:

1. В начале итерации счётчик активных фрагментов обнуляется.
2. На этапе поиска минимальных рёбер счётчик увеличивается на единицу для каждого фрагмента, у которого были исходящие рёбра.
3. На этапе объединения фрагментов счётчик уменьшается на единицу каждый раз, когда операция MERGE(u,v) вернула значение истина.

Если в конце итерации счётчик равен 0 или 1, то вычисления останавливаются. Параллелизм возможен на этапе сортировки рёбер по весу, однако основной ход алгоритма является последовательным.

**5) Доказательство.**

Пусть есть минимальный остов, в котором для какой-то вершины vv нет её минимального инцидентного ребра. Тогда, если добавить это ребро, образуется цикл, из которого можно удалить другое ребро, тоже инцидентное vv, но имеющее не меньший вес.

Алгоритм Борувки опирается на этот факт и заключается в следующем:

1. Для каждой вершины найдем минимальное инцидентное ей ребро.
2. Добавим все такие рёбра в остов (это безопасно — см. лемму) и сожмем получившиеся компоненты, то есть объединим списки смежности вершин, которые эти рёбра соединяют.
3. Повторяем шаги 1-2, пока в графе не останется только одна вершина-компонента.

Алгоритм может работать неправильно, если в графе есть ребра, равные по весу. Пример: «треугольник» с одинаковыми весами рёбер. Избежать такую ситуацию можно, введя какой-то дополнительный порядок на рёбрах — например, сравнивая пары из веса и номера ребра.

### **6) Сложность алгоритма**

На каждой итерации число деревьев в остовном лесу уменьшается по крайней мере в два раза, поэтому всего алгоритм совершает не более O(\log V) итераций. Каждая итерация может быть реализована со сложностью O(E) поэтому общее время работы алгоритма составляет O(E\log V)времени (здесь V и E— число вершин и рёбер в графе, соответственно).

Однако для некоторых видов графов, в частности, [планарных](https://ru.wikipedia.org/wiki/%D0%9F%D0%BB%D0%B0%D0%BD%D0%B0%D1%80%D0%BD%D1%8B%D0%B9_%D0%B3%D1%80%D0%B0%D1%84), оно может быть уменьшено до O(E). Существует также рандомизированный алгоритм построения [минимального остовного дерева](https://ru.wikipedia.org/wiki/%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), основанный на алгоритме Борувки, работающий в среднем за линейное время.

## Блок-схема данного алгоритма показана на (рис. 1) ниже.

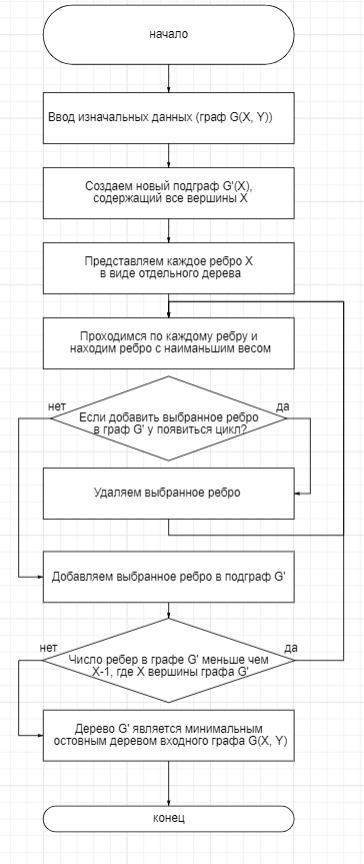


Рис.1. Блок-схема алгоритма Борувки

**5 Решение задачи на контрольном примере**

Дается неориентированный взвешенный граф (рис 2).

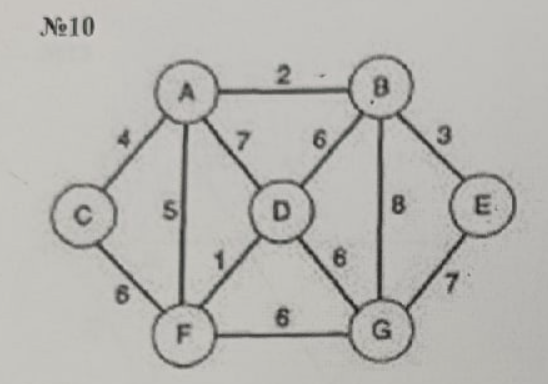


Рис. 2. Неориентированный взвешенный граф (Вариант 10).

**Решение:**

Решаем контрольный пример с помощью алгоритма Борувки.

Алгоритм состоит из нескольких шагов:

1. Изначально каждая вершина графа G— тривиальное дерево, а ребра не принадлежат никакому дереву.
2. Для каждого дерева T найдем минимальное инцидентное ему ребро. Добавим все такие ребра.
3. Повторяем шаг 2 пока в графе не останется только одно дерево T.

1) Обозначим все ребра графа как отдельное дерево

2) Найдем минимальное ребро и сразу добавляем его началом дерева

Это ребро F-D=1 (рис 3)

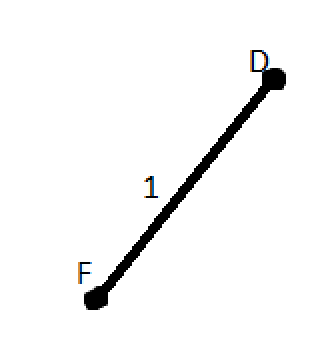


Рис. 3. Полученный граф на 2 стадии

3) Находим минимальное исходящее из вершин F и D ребра.

F-A=5 F-C=6 F-G=6 D-A=7 D-B=6 D-G=6

Самое выгодное ребро - F-A=5

4) Добавляем ребро в граф.(рис.4)

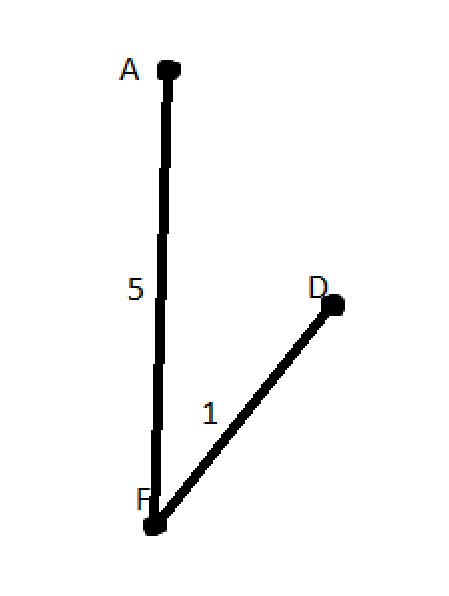


Рис. 4. Полученный граф на 4 стадии

5) Проверяем, обошли ли мы все вершины в графе.

Нет!

6) Находим минимальное исходящее из вершин A, F и D ребра.

Теперь нужно смотреть чтобы наше дерево не превратилось в цикл (то есть мы не можем из вершины A прийти в вершину D)

F-C=6 F-G=6 D-B=6 D-G=6 A-C=4 A-B=2

Самое выгодное ребро A-B=2

7) Добавляем ребро в граф.(рис. 5)

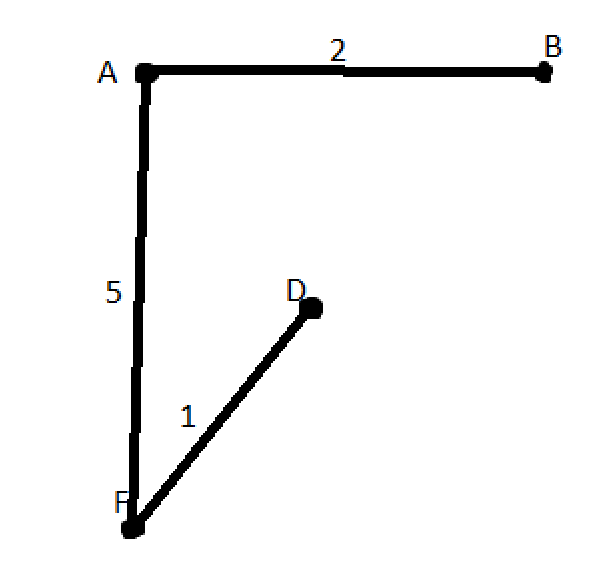


Рис. 5. Полученный граф на стадии 7

8) Проверяем, обошли ли мы все вершины в графе.

Нет!

9) Находим минимальное исходящее из вершин A,B, F и D ребра, исключая возможные циклы.

F-C=6 F-G=6 D-G=6 A-C=4 B-E=3 B-G=8

Самое выгодное ребро B-E=3.

10) Добавляем ребро в граф.(рис. 6)

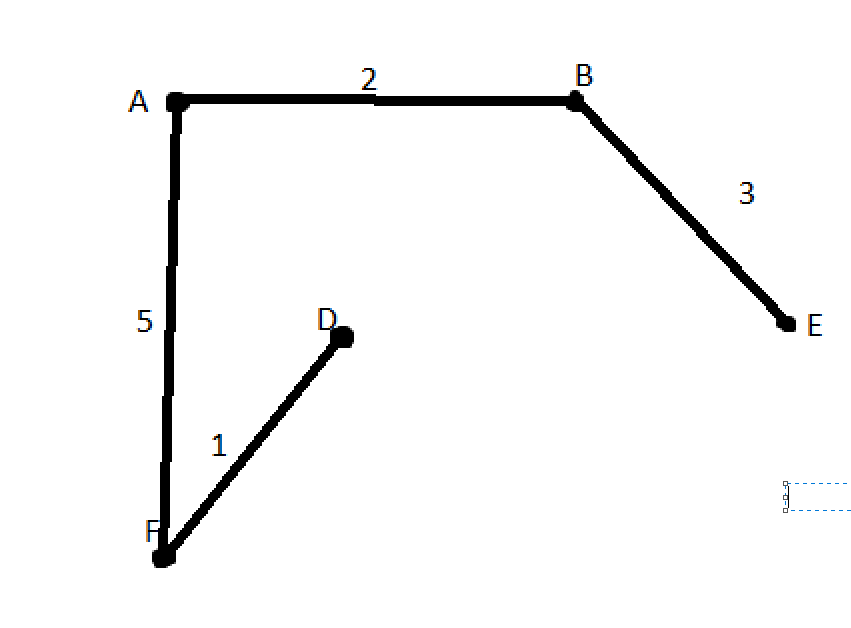


Рис. 6. Полученный граф на стадии 10

11) Проверяем, обошли ли мы все вершины в графе.

Нет!

12) Находим минимальное исходящее из вершин A,B,E,F и D ребра, исключая возможные циклы.

F-C=6 F-G=6 D-G=6 A-C=4 B-G=8 E-G=7

Самое выгодное ребро A-C=4.

13) Добавляем ребро в граф.(рис. 7)

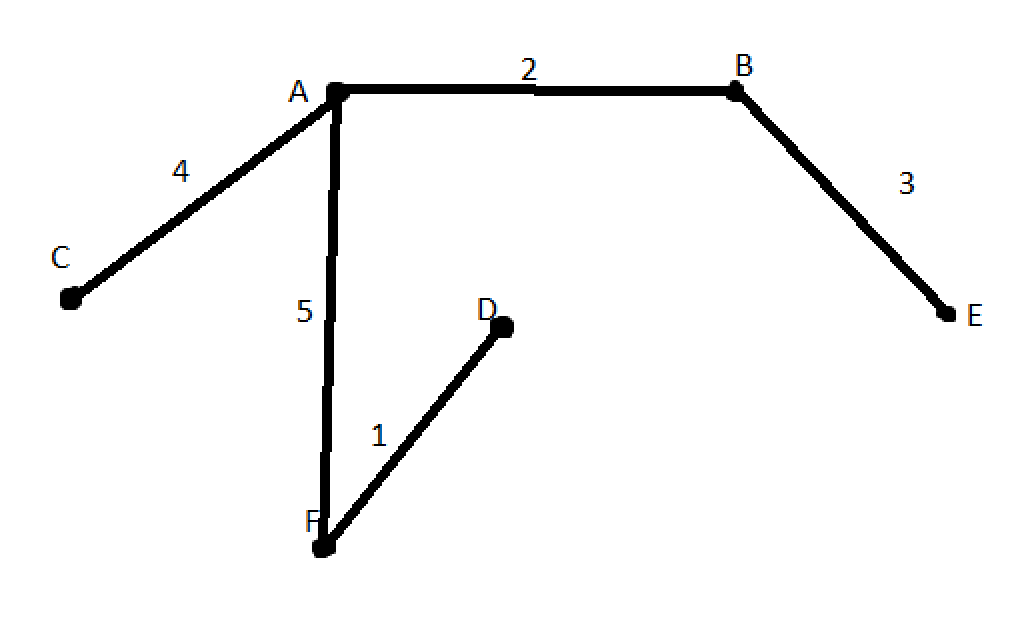


Рис. 7. Полученный граф на стадии 13

14) Проверяем, обошли ли мы все вершины в графе.

Нет!

15) Находим минимальное исходящее из вершин A,B,E,F,C и D ребра, исключая возможные циклы.

F-G=6 D-G=6 B-G=8 E-G=7

Минимальны ребра F-G и D-G весят одинаково поэтому можем прибавить любое из них. Вес получившегося дерева не изменится.

В случаях с одинаковыми весами ребер правилом хорошего тона будет правильным обозначить порядок обхода таких вершин. Выберем ребро D-G=6.

“Алгоритм может работать неправильно, если в графе есть ребра, равные по весу. Пример: «треугольник» с одинаковыми весами рёбер. Избежать такую ситуацию можно, введя какой-то дополнительный порядок на рёбрах — например, сравнивая пары из веса и номера ребра.”

16) Добавляем ребро в граф.(рис. 8)

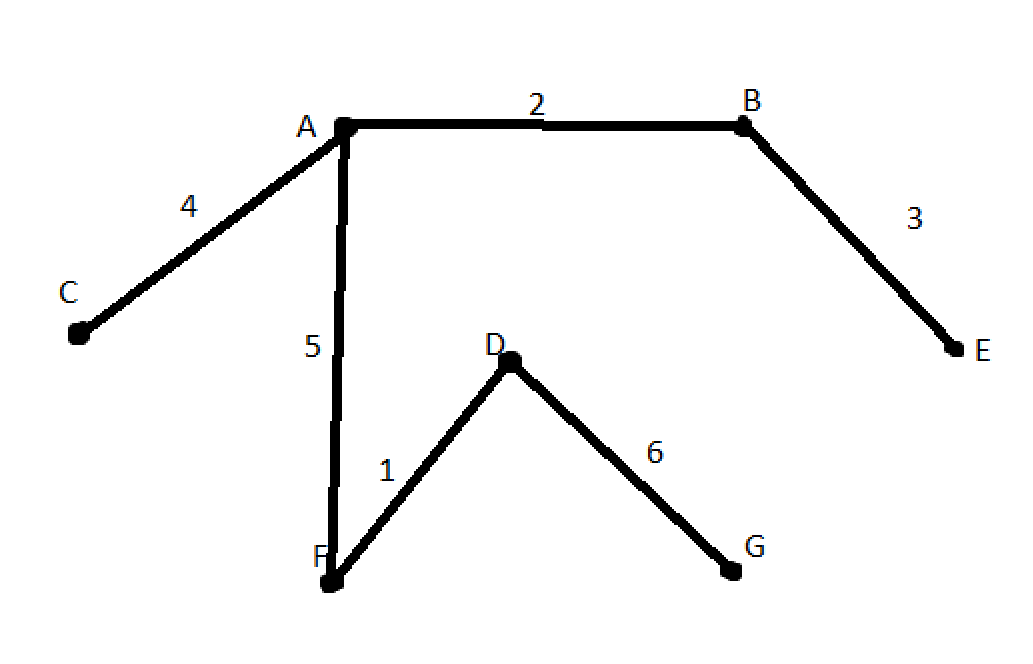


Рис. 8. Полученный граф на стадии 16

17) Проверяем, обошли ли мы все вершины в графе.

Да!

18) Считаем вес получившегося графа.

Минимальный вес получившегося дерева будет равен сумме входящих в него ребер. N = D-G + D-F + F-A + A-C + A-B + B-E =6+1+5+4+2+3= 21.

Задача решена! Мы нашли минимальное остовное дерево и его вес с помощью алгоритма Борувки.

**6 Описание работы программы.**

## Представим описание работы программы для решения задачи о поиске минимального осовного графа в виде структуры программы (с помощью UML-диаграммы классов программы).(рис. 9)

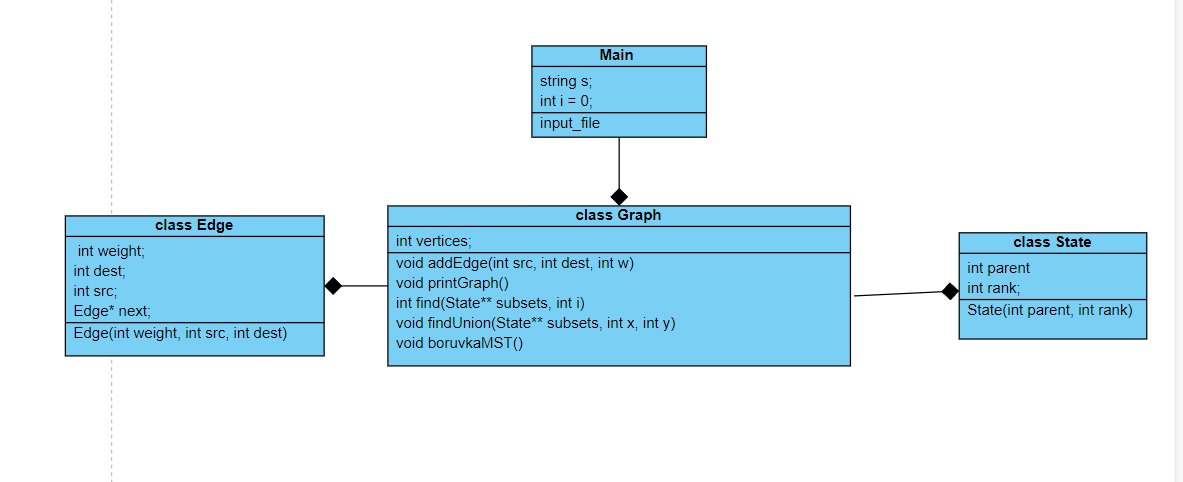


Рис. 9. UML диаграмма

## Программно-модифицированный алгоритм Борувки реализован в виде консольного приложения на языке C++. Описание классов и методов представлено ниже в таблице 2.



Таб. 2. Полное описание используемых методов

**7 Примеры решения задачи о поиске минимального остовного дерева.**

Пример №1

**Решение:**

1) Рассмотрим граф уже решенный выше (рис 10):

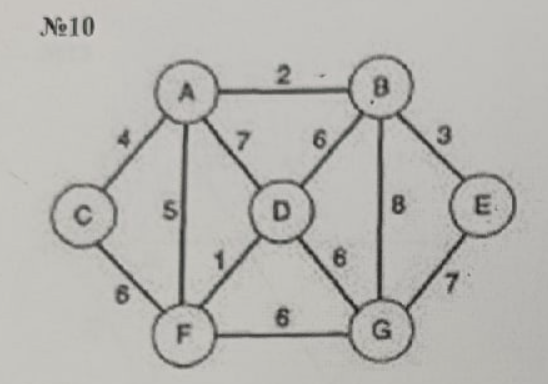


Рис. 10 Граф из варианта №10

2) Запишем граф в удобном для программы формате и пронумеруем вершины (рис 11):

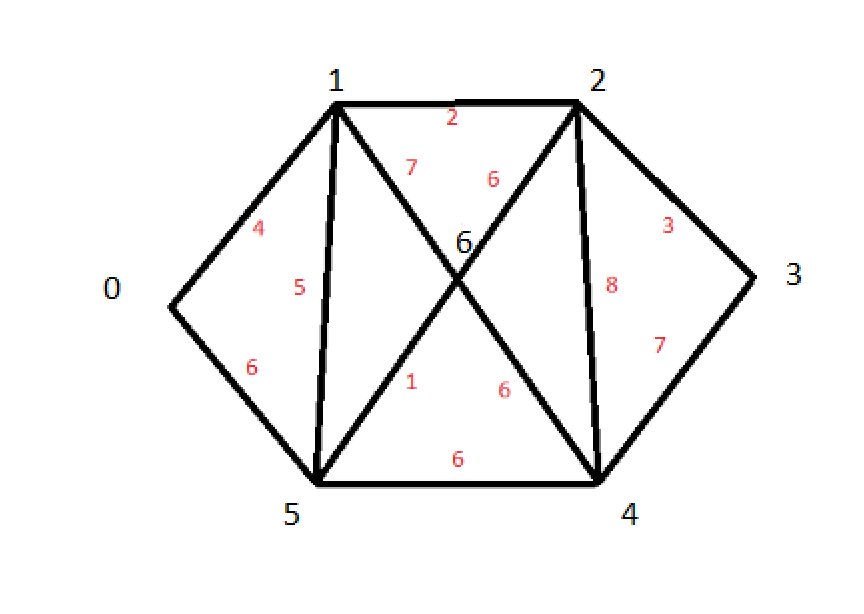


Рис. 11. Модернизированный граф

3) Забиваем данные в программу

4) Результат выполнения программы (рис 12):

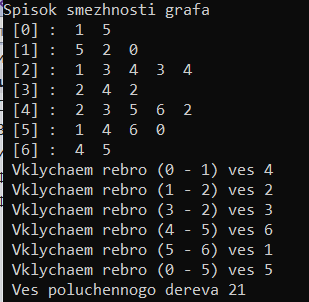


Рис. 12. Результат выполнения программы

5) Представим полученный результат в виде связанного дерева (рис 13):

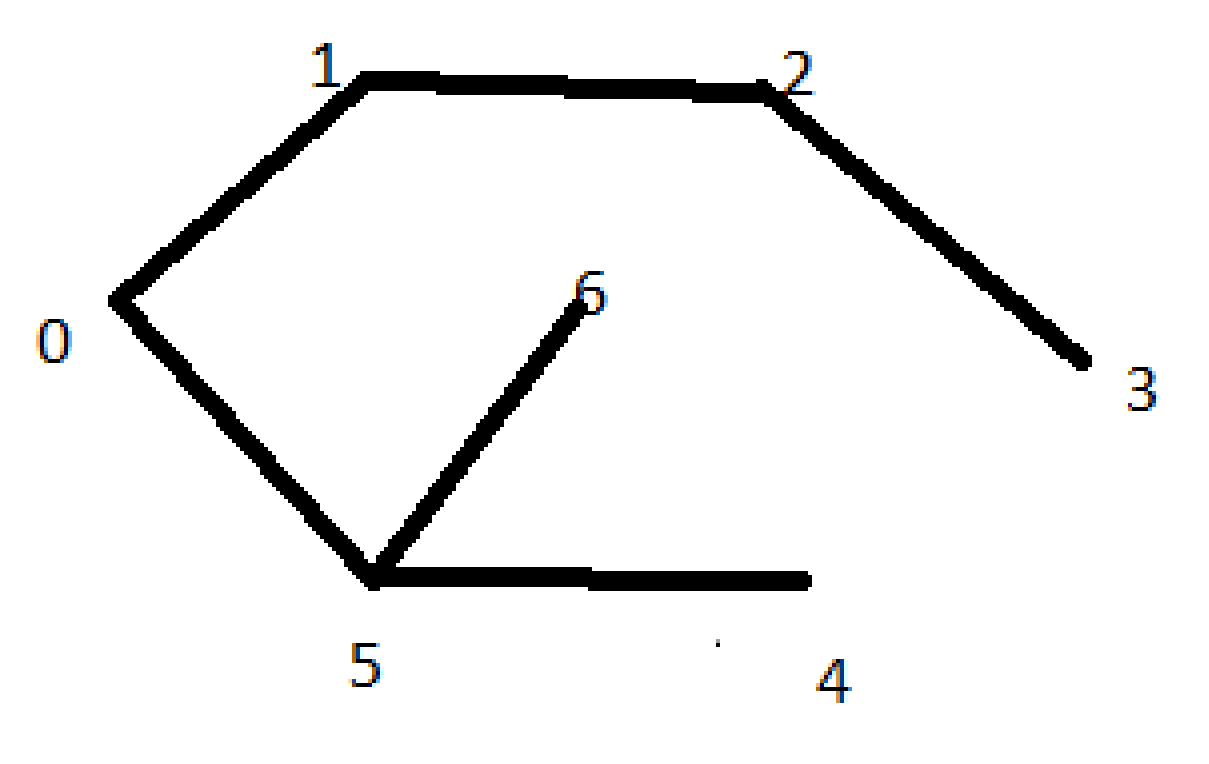


Рис. 13. Визуализированное решение программы.

Пример №2

1) Рассмотрим граф изображенный на рисунке 14.

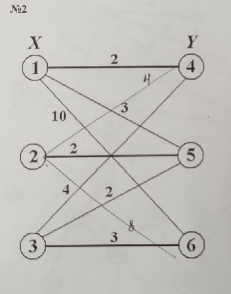


Рис. 14. Граф из варианта №2

2) Забиваем данные в программу

3) Результат выполнения программы (рис 15):

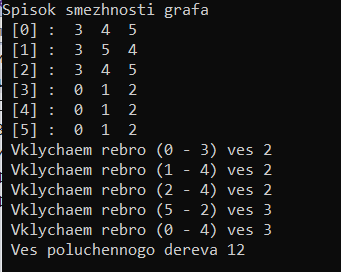


Рис. 15. Результат выполнения программы

4) Представим полученный результат в виде связанного дерева (рис 16):

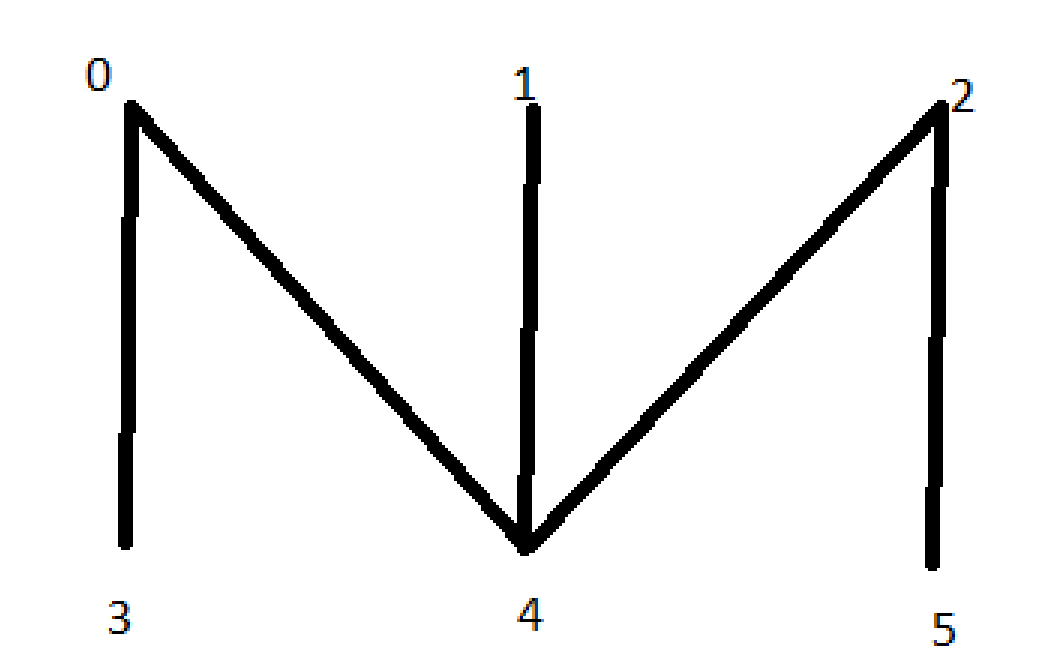


Рис. 16. Визуализированное решение программы.

Пример №3

1) Рассмотрим граф изображенный на рисунке 17.

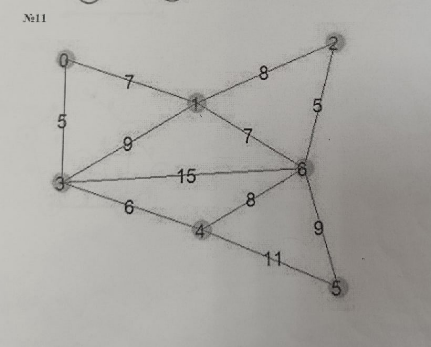


Рис. 17. Граф из варианта №11

2) Забиваем данные в программу

3) Результат выполнения программы (рис 18):

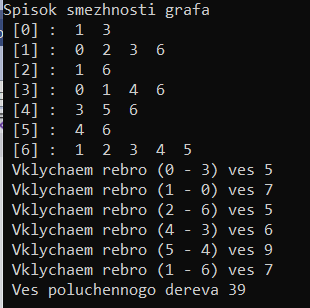


Рис. 18 Результат выполнения программы

4) Представим полученный результат в виде связанного дерева (рис 19):

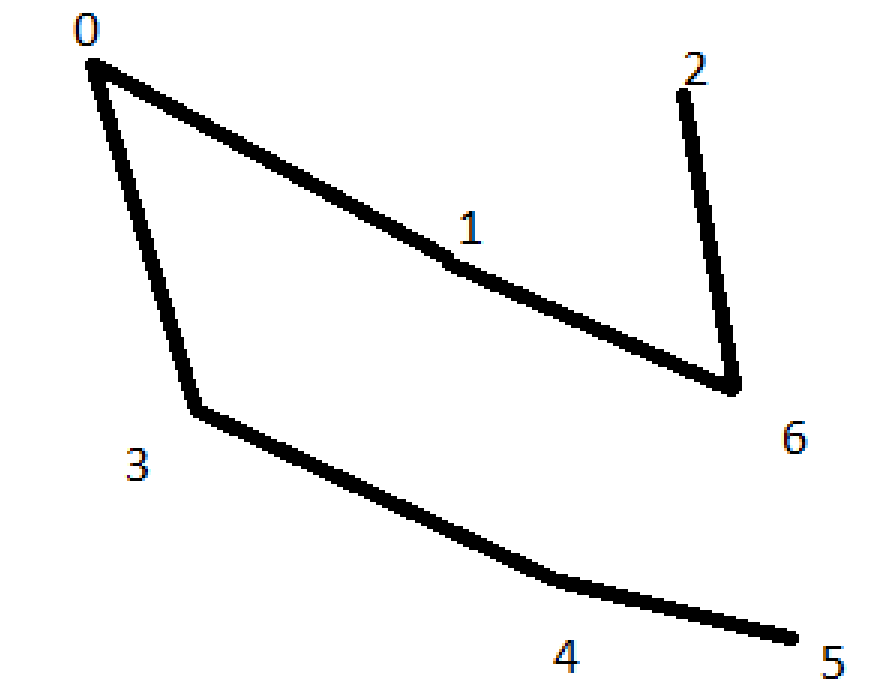


Рис. 19. Визуализированное решение программы.

Пример №4

1) Рассмотрим граф изображенный на рисунке 20.

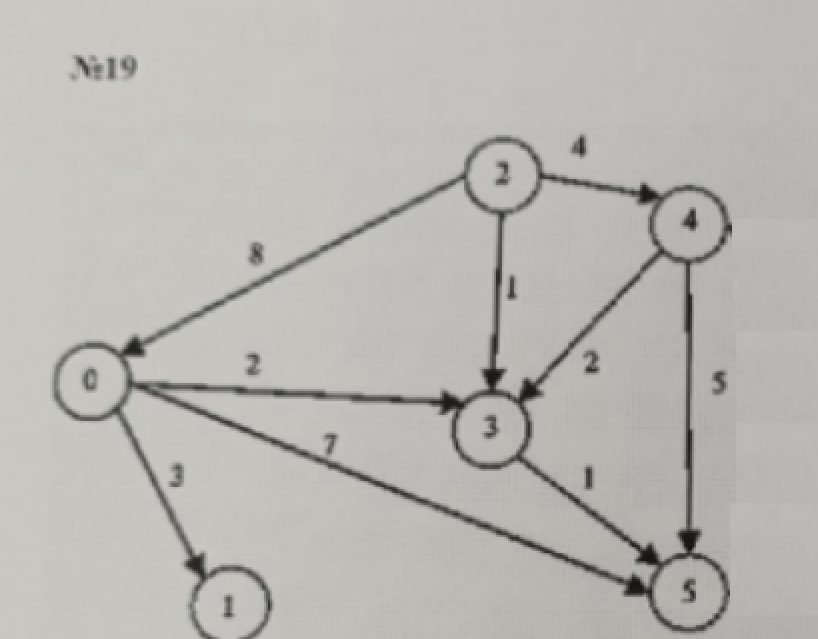


Рис. 20. Граф из варианта №23

2) Представим граф в виде неориентированного графа и забиваем данные в программу:

3) Результат выполнения программы (рис 21):

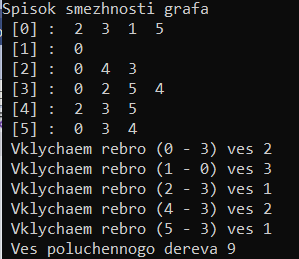


Рис. 21 Результат выполнения программы

4) Представим полученный результат в виде связанного дерева (рис 22):

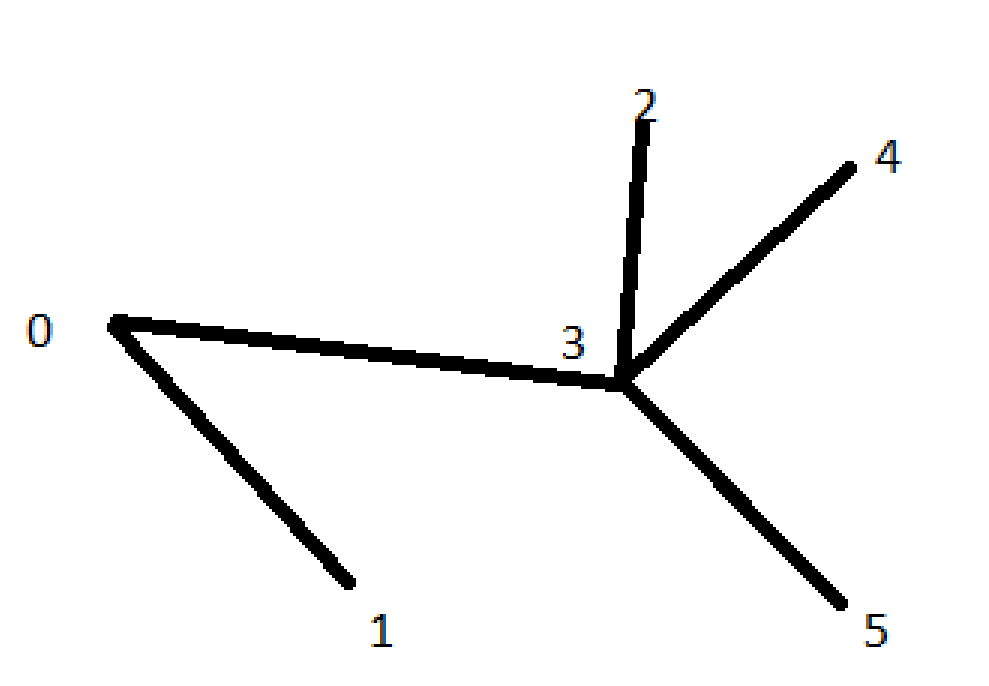


Рис. 22. Визуализированное решение программы.

**ЗАКЛЮЧЕНИЕ**

В данной курсовой работе мной была изучена задача о поиске минимального остовного графа на основе алгоритма Борувки.

Была сформулирована содержательная и математическая постановка задачи. Также мы произвели описание алгоритма решения задачи и решили задачу на контрольном примере.

Мы разработали программу на языке программирования С++, которая реализует поведение алгоритма Борувки. С помощью этой программы мы прорешали 4 варианта различных неориентированных, взвешанных графа.

Мы доказали актуальность алгоритма Борувки в наши дни.

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. Гальперин Александр Леонидович. Комбинаторные алгоритмы. Минимальный остов. 2018. Стр 18. Режим доступа: (creewick.github.io)
2. Галиев Ш. И. Дискретная математика. Казань: 2009.
3. Алгоритм Борувки. Научно-образовательный портал Викиконспекты [Электронный ресурс]:URL: [Алгоритм Борувки — Викиконспекты (ifmo.ru)](https://neerc.ifmo.ru/wiki/index.php?title=%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC_%D0%91%D0%BE%D1%80%D1%83%D0%B2%D0%BA%D0%B8)
4. Алгоритм Борувки. Научно-образовательный портал Википедия (алгоритм Борувки) [Электронный ресурс]:URL: [Алгоритм Борувки — Википедия (wikipedia.org)](https://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC_%D0%91%D0%BE%D1%80%D1%83%D0%B2%D0%BA%D0%B8)
5. Построение минимального остовного дерева алгоритмом Борувки. Программная реализация. Научно-образовательный портал Научные проблемы [Электронный ресурс]:URL: [Построение минимального остовного дерева алгоритмом Борувки. Программная реализация (scienceproblems.ru)](https://scienceproblems.ru/postroenie-minimalnogo/2.html)
6. Минимальное остовное дерево. Научно-образовательный портал Википедия (остовное дерево) [Электронный ресурс]:URL: [Минимальное остовное дерево — Википедия (wikipedia.org)](https://ru.wikipedia.org/wiki/%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)

## ПРИЛОЖЕНИЕ А

## **Листинг программы.**

#include <iostream>

#include <string>

#include <fstream>

#include <vector>

#include <sstream>

#include <algorithm>

using namespace std;

class Edge

{

public:

// вес ребра и тд

int weight;

int dest;

int src;

Edge\* next;

Edge(int weight, int src, int dest)

{

this->weight = weight;

this->dest = dest;

this->src = src;

this->next = NULL;

}

};

{

public: int parent;

int rank;

State(int parent, int rank)

{

this->parent = parent;

this->rank = rank;

}

};

class Graph

{

public: int vertices;

vector < vector < Edge\*> > graphEdge;

Graph(int vertices)

{

this->vertices = vertices;

for (int i = 0; i < this->vertices; ++i)

{

this->graphEdge.push\_back(vector < Edge\*>());

}

}

void addEdge(int src, int dest, int w)

{

if (dest < 0 || dest >= this->vertices ||

src < 0 || src >= this->vertices)

{

return;

}

// добавим ребро узла

this->graphEdge.at(src).push\_back(new Edge(w, src, dest));

if (dest == src)

{

return;

}

this->graphEdge.at(dest).push\_back(new Edge(w, dest, src));

}

void printGraph()

{

cout << "\Spisok smezhnosti grafa ";

for (int i = 0; i < this->vertices; ++i)

{

cout << " \n [" << i << "] :";

// итерировать ребра i узла

for (int j = 0; j < this->graphEdge.at(i).size(); ++j)

{

cout << " " << this->graphEdge.at(i).at(j)->dest;

}

}

}

int find(State\*\* subsets, int i)

{

if (subsets[i]->parent != i)

{

subsets[i]->parent = this->find(subsets, subsets[i]->parent);

}

return subsets[i]->parent;

}

void findUnion(State\*\* subsets, int x, int y)

{

int a = this->find(subsets, x);

int b = this->find(subsets, y);

if (subsets[a]->rank < subsets[b]->rank)

{

subsets[a]->parent = b;

}

else if (subsets[a]->rank > subsets[b]->rank)

{

subsets[b]->parent = a;

}

else

{

subsets[b]->parent = a;

subsets[a]->rank++;

}

}

void boruvkaMST()

{

// Содержит сумму веса в пути mst

int result = 0;

int selector = this->vertices;

State\*\* subsets = new State \* [this->vertices];

Edge\*\* cheapest = new Edge \* [this->vertices];

for (int v = 0; v < this->vertices; ++v)

{

subsets[v] = new State(v, 0);

}

while (selector > 1)

{

for (int v = 0; v < this->vertices; ++v)

{

cheapest[v] = NULL;

}

for (int k = 0; k < this->vertices; k++)

{

for (int i = 0; i < this->graphEdge.at(k).size(); ++i)

{

int set1 = this->find(subsets,

this->graphEdge.at(k).at(i)->src);

int set2 = this->find(subsets,

this->graphEdge.at(k).at(i)->dest);

if (set1 != set2)

{

if (cheapest[k] == NULL)

{

cheapest[k] = this->graphEdge.at(k).at(i);

}

else if (cheapest[k]->weight >

this->graphEdge.at(k).at(i)->weight)

{

cheapest[k] = this->graphEdge.at(k).at(i);

}

}

}

}

for (int i = 0; i < this->vertices; i++)

{

if (cheapest[i] != NULL)

{

int set1 = this->find(subsets, cheapest[i]->src);

int set2 = this->find(subsets, cheapest[i]->dest);

if (set1 != set2)

{

// уменьшить ребро

selector--;

this->findUnion(subsets, set1, set2);

// Отображение соединений ребер

cout << "\n Vklychaem rebro ("

<< cheapest[i]->src

<< " - "

<< cheapest[i]->dest

<< ") ves "

<< cheapest[i]->weight;

// добавим вес

result += cheapest[i]->weight;

}

}

}

}

cout << "\n Ves poluchennogo dereva "

<< result << endl;

}

};

int main()

{

string s;

int i = 0;

ifstream file("C:/Users/Админ/OneDrive/Рабочий стол/графы/4.txt");

if (file.is\_open())

{

getline(file, s);

i = static\_cast<int>(s[0]) - 48;

}

Graph\* g = new Graph(i);

if (file.is\_open())

{

while (getline(file, s))

{

int i1 = static\_cast<int>(s[0]) - 48;

int i2 = static\_cast<int>(s[1]) - 48;

int i3 = static\_cast<int>(s[2]) - 48;

g->addEdge(i1, i2, i3);

}

}

file.close();

g->printGraph();

g->boruvkaMST();

return 0;

}