**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МО ЭВМ**

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**по курсовой работе**

**по дисциплине «Построение и анализ алгоритмов»**

**Тема:** «**Алгоритм Краскала»**

|  |  |  |
| --- | --- | --- |
| Студент гр. 5382 | Никитин В.А. |  |
| Преподаватель | Шолохова О.М. |  |

Санкт-Петербург

2017

Содержание

[1. ИСХОДНОЕ ЗАДАНИЕ 3](#_Toc483757261)

[2. ОБЪЯСНЕНИЕ АЛГОРИТМА](#_Toc483757262) 3

[2.1. Формулировка](#_Toc483757263) 3

[2.2. Оценка 3](#_Toc483757264)

[3. ФОРМАЛЬНАЯ ПОСТАНОВКА ЗАДАЧИ 4](#_Toc483757267)

[4. ПОШАГОВАЯ РАБОТА АЛГОРИТМА](#_Toc483757262) 4

[5. СПЕЦИФИКАЦИЯ ПРОГРАММЫ 7](#_Toc483757275)

[6. ТЕСТИРОВАНИЕ](#_Toc483757262) 8

[7. ЗАКЛЮЧЕНИЕ 9](#_Toc483757285)

[Приложение А. Код программы..](#_Toc483757286) 10

1. Исходное задание.

В данной курсовой работе требуется написать программу, выполняющую поиск наименьшего по весу остовного дерева в графе с помощью алгоритма Краскала и выполнить исследование алгоритма.

1. Объяснение алгоритма.

**Алгоритм Краскала** — эффективный алгоритм построения минимального остовного дерева взвешенного связного неориентированного графа. Также алгоритм используется для нахождения некоторых приближений для задачи Штейнера. Алгоритм впервые описан Джозефом Краскалом в 1956 году.

**Формулировка**:

Вначале текущее множество рёбер устанавливается пустым. Затем, пока это возможно, проводится следующая операция: из всех рёбер, добавление которых к уже имеющемуся множеству не вызовет появление в нём цикла, выбирается ребро минимального веса и добавляется к уже имеющемуся множеству. Когда таких рёбер больше нет, алгоритм завершён. Подграф данного графа, содержащий все его вершины и найденное множество рёбер, является его остовным деревом минимального веса.

**Оценка:**

До начала работы алгоритма необходимо отсортировать рёбра по весу, это требует *O(E×log(E))* времени. После чего компоненты связности удобно хранить в виде системы непересекающихся множеств. Все операции в таком случае займут *O(E×α(E,V))*, где α — функция, обратная к функции Аккермана. Поскольку для любых практических задач *α(E,V) < 5*, то можно принять её за константу, таким образом, общее время работы алгоритма Краскала можно принять за *O(E\*log(E))*.

1. Формальная постановка задачи.

**Входные данные:** Количество вершин и ребер в графе. Из них строится список ребер графа, в котором отражены вершины, соединяющиеся данным ребром, и вес ребра.

**Выходные данные:** Время работы алгоритма, вес минимального остовного дерева. Так же возможен вывод на экран списка ребер графа и ребер остовного дерева.

1. Пошаговая работа алгоритма.
2. Имеем на входе граф и неотсортированный список ребер.

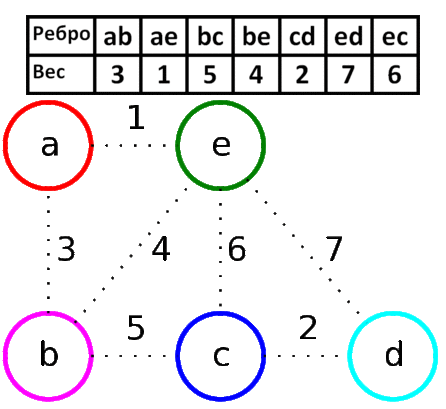


Рисунок 1.

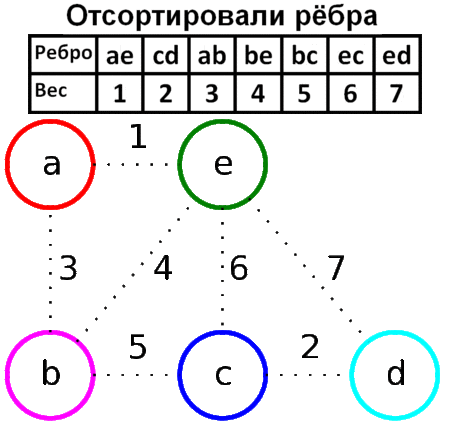
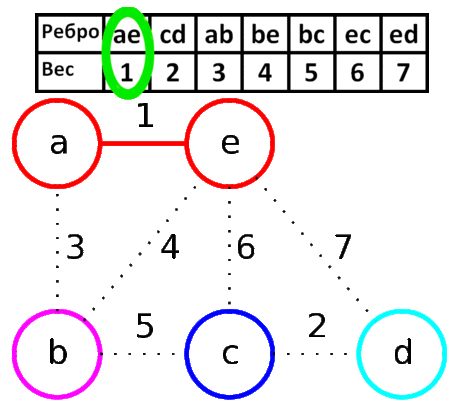
1. Сортируем список ребер в порядке возрастания их весов.

Рисунок 2.

1. В цикле начинаем поочередно проверять очередное ребро из списка и добавлять его в остовное дерево, если его добавление не создает циклов.

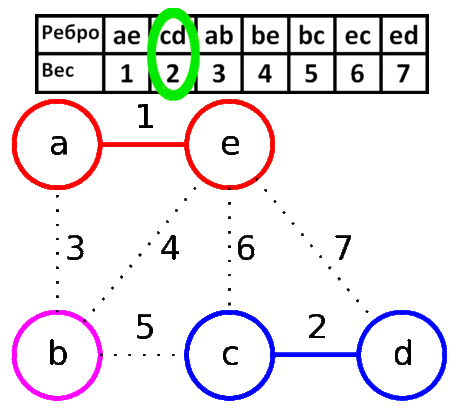
Рисунок 3.

Рисунок 4.

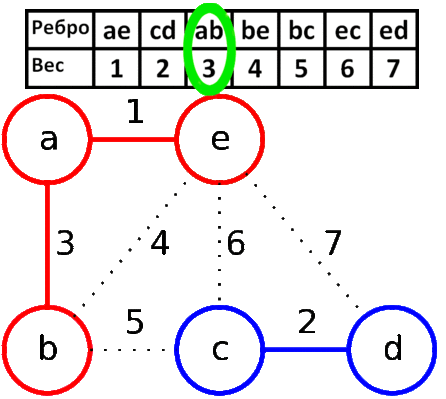


Рисунок 5.

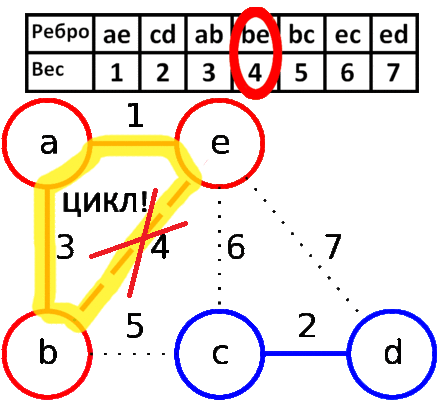


Рисунок 6.

1. Цикл работает до тех пор, пока все ребра не будут рассмотрены.

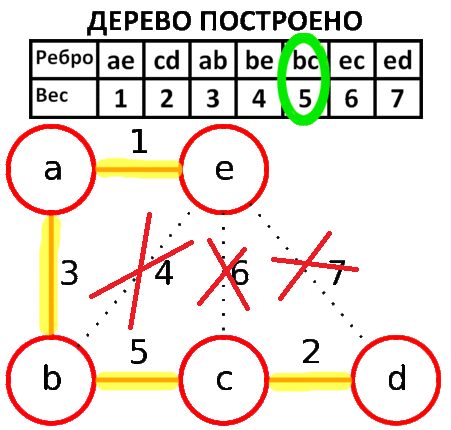


Рисунок 7.

1. Спецификация программы.

Struct edge; // Структура представляющая ребро. Содержит два поля вершин, которые соединяются данным ребром и поле веса ребра.

Bool cmp(const edge& a, const edge& b); // возвращает True если вес первого ребра меньше второго и наоборот.

Int getLeader(int x); // функция позволяющая узнать в каком подмножестве находится данная вершина

Bool unite(int x, int y); // функция возвращающая True если эти две вершины находятся в разных подмножествах и наоборот. Служит для определения можно ли брать текущее ребро или нет.

Void generate(vector <edge> &e, int m, int n); // функция генерации графа. Генерация происходит случайным образом. Сначала генерируются ребра между последовательными вершинами. Далее достраиваются остальные ребра в случайном порядке с проверкой на то, чтобы у каждой пары вершин было не более 1 ребра.

Void print(vector <edge> e, int size) // Функция вывода списка ребер на экран.

В main’е реализован минимальный диалог с пользователем, запрашивающий количество вершин и ребер в графе.

Рисунок 8.

Пример вывода результатов на экран.

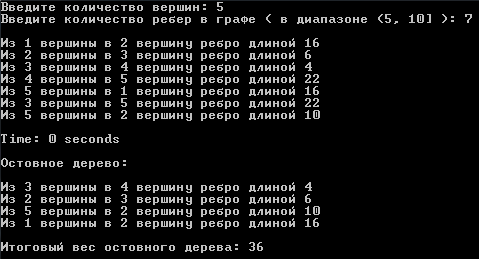


Рисунок 9.

1. Тестирование.

Таблица 1.

|  |  |
| --- | --- |
| Число ребер в графе | Время выполнения алгоритма в сек. |
| 100 | 0,001 |
| 500 | 0,001 |
| 1000 | 0,002 |
| 5000 | 0,006 |
| 10000 | 0,012 |

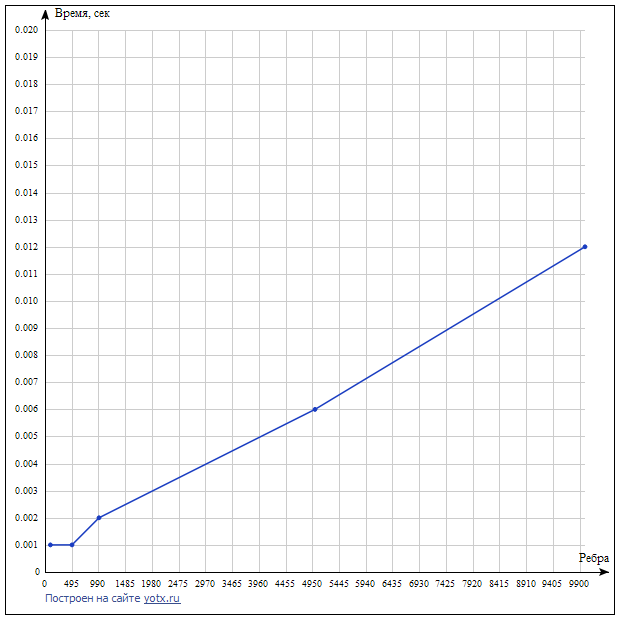


График 1.

Сложность алгоритма: O(nlog(n)).

1. Заключение.

В ходе написания данной курсовой работы получены теоретические знания об алгоритме Краскала и его применении на практике. Так же данный алгоритм был реализован на языке с++. Реализация была протестирована на графах различных размеров и было произведено временное исследование работы алгоритма.

**Приложение А. Код программы.**

#include "stdafx.h"

#include <iostream>

#include <cstdio>

#include <algorithm>

#include <fstream>

#include <vector>

#include <ctime>

using namespace std;

int heightdiap = 25;

struct edge{

int x, y, w;

edge(){}

edge(int x, int y, int w) :x(x), y(y), w(w){}

};

bool cmp(const edge& a, const edge& b){

return a.w < b.w;

}

vector <int> leader;

int getLeader(int x){

if (x == leader[x])return x;

return leader[x] = getLeader(leader[x]);

}

bool unite(int x, int y){

x = getLeader(x);

y = getLeader(y);

if (x == y)return false;

if (rand() % 2 == 0) swap(x, y);

leader[x] = y;

return true;

}

void generate(vector <edge> &e, int m, int n){

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

e[i].x = i;

if ((i + 1) == n) e[i].y = 0;

else e[i].y = i + 1;

e[i].w = rand() % heightdiap + 1;

}

for (int i = n; i < m; ++i){

e[i].x = rand() % n;

e[i].y = rand() % n;

while (e[i].y == e[i].x) e[i].y = rand() % n;

e[i].w = rand() % heightdiap + 1;

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

if (((e[j].x == e[i].x) && (e[j].y == e[i].y)) || ((e[j].x == e[i].y) && (e[j].y == e[i].x))){

i--; j = i - 1;

}

}

}

}

void print(vector <edge> e, int size){

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

cout << endl << "Из " << e[i].x+1 << " вершины в " << e[i].y+1 << " вершину ребро длиной " << e[i].w;

}

}

int \_tmain(int argc, \_TCHAR\* argv[])

{

setlocale(LC\_ALL, "Russian");

srand(time(0));

int n=0, m=0;

cout << "Введите количество вершин: ";

cin >> n;

cout << "Введите количество ребер в графе ( в диапазоне ("<<n<<","<<n\*(n-1)/2<<"] ): ";

cin >> m;

while ((m <= n) || (m>n\*(n - 1) / 2)){

cin.clear();

cin.ignore(225, '\n');

cout << "Недопустимое число, введите число удовлетворяющее диапазону (" << n << "," << n\*(n - 1) / 2 << "]" << endl << endl;

cin >> m;

}

vector <edge> e(m);

generate(e, m, n);

print(e, m);

float start = clock();

sort(e.begin(), e.end(), cmp);

leader.resize(n);

for (int i = 0; i < n; ++i)leader[i] = i;

vector <edge> ans;

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

int x = e[i].x, y = e[i].y;

if (unite(x, y)) ans.push\_back(e[i]);

}

float end = clock();

cout << endl << endl << "Time: " << (float)(end - start) / 1000 << " seconds";

cout << endl << endl << "Остовное дерево: " << endl;

print(ans, ans.size());

int sum = 0;

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

sum += ans[i].w;

cout <<endl<<endl<<"Итоговый вес остовного дерева: "<< sum << endl<<endl;

system("pause");

return 0;

}