ФЕДЕРАЛЬНОЕ АГЕНТСТВО СВЯЗИ

Федеральное государственное образовательное бюджетное учреждение

высшего профессионального образования

«Санкт-Петербургский государственный университет телекоммуникаций

им. проф. М. А. Бонч-Бруевича»

Отчет по лабораторной работе №4

**«Алгоритмы построения остовного**

**дерева сети»**

по дисциплине

«Алгоритмы и структуры данных»

Выполнили ст.группы: ИКПИ-74

ФИО:Шевцов Д. А.

Шмелёва В. Д.

Проверил: к.т.н, доцент кафедры ПИиВТ

Дагаев А.В.

**2019 г, СПб**

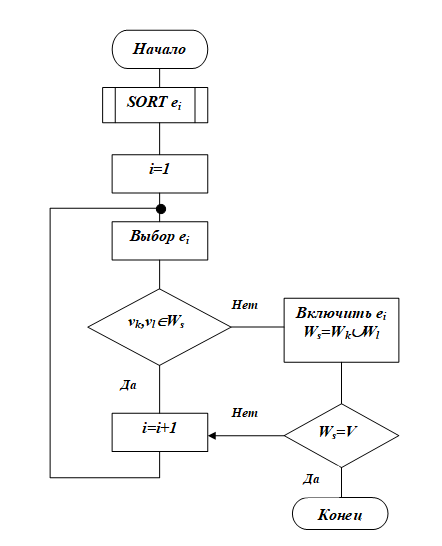
**Постановка задачи:**

Дан взвешенный неориентированный граф. Требуется найти такое поддерево этого графа, которое бы соединяло все его вершины, и при этом обладало наименьшим весом (т.е. суммой весов рёбер) из всех возможных. Такое поддерево называется минимальным остовным деревом или простом минимальным остовом.

**Алгоритм Крускала.**

**Описание:**

Алгоритм Крускала изначально помещает каждую вершину в своё дерево, а затем постепенно объединяет эти деревья, объединяя на каждой итерации два некоторых дерева некоторым ребром. Перед началом выполнения алгоритма, все рёбра сортируются по весу (в порядке неубывания). Затем начинается процесс объединения: перебираются все рёбра от первого до последнего (в порядке сортировки), и если у текущего ребра его концы принадлежат разным поддеревьям, то эти поддеревья объединяются, а ребро добавляется к ответу. По окончании перебора всех рёбер все вершины окажутся принадлежащими одному поддереву, и ответ найден.

**Блок-схема алгоритма:**

### Реализация:

Этот код реализует описанный выше алгоритм, и выполняется за **O (M log N + N2)**. Сортировка рёбер потребует O (M log N) операций. Принадлежность вершины тому или иному поддереву хранится просто с помощью массива tree\_id - в нём для каждой вершины хранится номер дерева, которому она принадлежит. Для каждого ребра мы за O (1) определяем, принадлежат ли его концы разным деревьям. Наконец, объединение двух деревьев осуществляется за O (N) простым проходом по массиву tree\_id. Учитывая, что всего операций объединения будет N-1, мы и получаем асимптотику **O (M log N + N2)**.

void kruskala(vector < pair < int, pair<int, int> > > g,int n,int m )

{

int cost = 0;

vector < pair<int, int> > res;

sort(g.begin(), g.end());

vector<int> tree\_id(n);

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

{

tree\_id[i] = i;

}

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

{

int a = g[i].second.first, b = g[i].second.second, l = g[i].first;

if (tree\_id[a] != tree\_id[b])

{

cost += l;

res.push\_back(make\_pair(a, b));

int old\_id = tree\_id[b], new\_id = tree\_id[a];

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

{

if (tree\_id[j] == old\_id)

{

tree\_id[j] = new\_id;

}

}

}

}

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

{

cout<<res[i].first<<" "<<res[i].second;

cout<<endl;

}

}

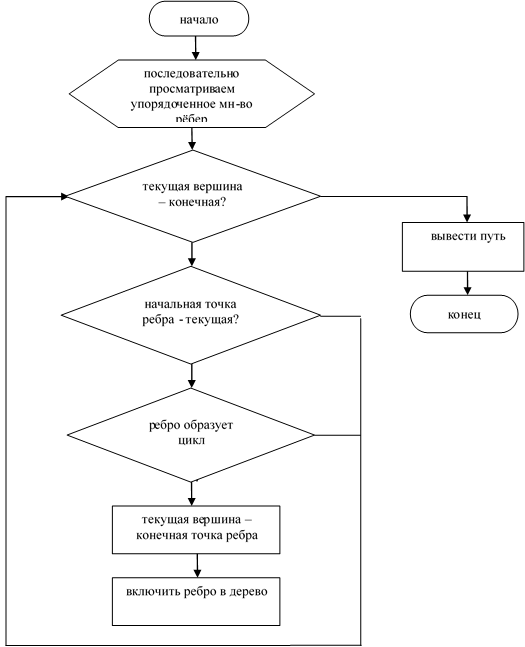
**Алгоритм Прима:**

**Описание:**

Искомый минимальный остов строится постепенно, добавлением в него рёбер по одному. Изначально остов полагается состоящим из единственной вершины (её можно выбрать произвольно). Затем выбирается ребро минимального веса, исходящее из этой вершины, и добавляется в минимальный остов. После этого остов содержит уже две вершины, и теперь ищется и добавляется ребро минимального веса, имеющее один конец в одной из двух выбранных вершин, а другой — наоборот, во всех остальных, кроме этих двух. И так далее, т.е. всякий раз ищется минимальное по весу ребро, один конец которого — уже взятая в остов вершина, а другой конец — ещё не взятая, и это ребро добавляется в остов (если таких рёбер несколько, можно взять любое). Этот процесс повторяется до тех пор, пока остов не станет содержать все вершины (или, что то же самое, n-1 ребро).

В итоге будет построен остов, являющийся минимальным. Если граф был изначально не связен, то остов найден не будет (количество выбранных рёбер останется меньше n-1).

**Блок-схема алгоритма:**

****

**Реализация:**

Для каждой ещё не выбранной будем хранить минимальное ребро, ведущее в уже выбранную вершину.

Тогда, чтобы на текущем шаге произвести выбор минимального ребра, надо просто просмотреть эти минимальные рёбра у каждой не выбранной ещё вершины — асимптотика составит O(n).

Но теперь при добавлении в остов очередного ребра и вершины эти указатели надо пересчитывать. Заметим, что эти указатели могут только уменьшаться, т.е. у каждой не просмотренной ещё вершины надо либо оставить её указатель без изменения, либо присвоить ему вес ребра в только что добавленную вершину. Следовательно, эту фазу можно сделать также за O(n).

Таким образом, мы получили вариант алгоритма Прима с асимптотикой O(n^2).

void prim(vector < vector<int> > g,int n)

{

vector<bool> used (n);

vector<int> min\_e (n, INF), sel\_e (n, -1);

min\_e[0] = 0;

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

{

int v = -1;

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

{

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

v = j;

}

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

{

cout << "No MST!";

exit(0);

}

used[v] = true;

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

cout << v << " " << sel\_e[v] << endl;

for (int to=0; to<n; ++to)

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

{

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

sel\_e[to] = v;

}

}

}

**Описание программы**

В данной программе все входные данные хранятся в такой структуре данных, как вектор.

Программа состоит из следующих функций:

1) Функция подсчета времени выполнения алгоритма Прима и Крускала

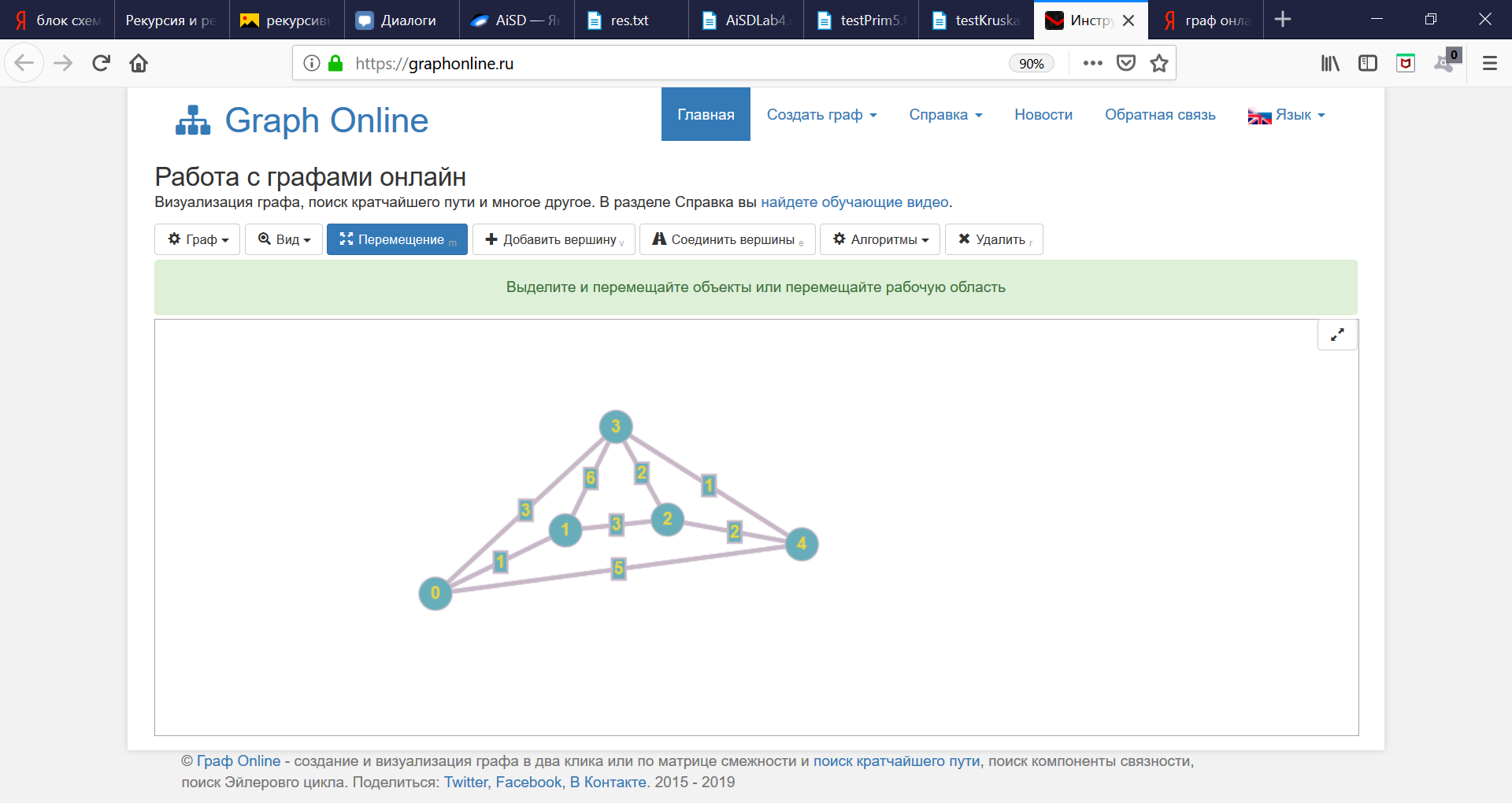
2) Функция с алгоритмом Крускала

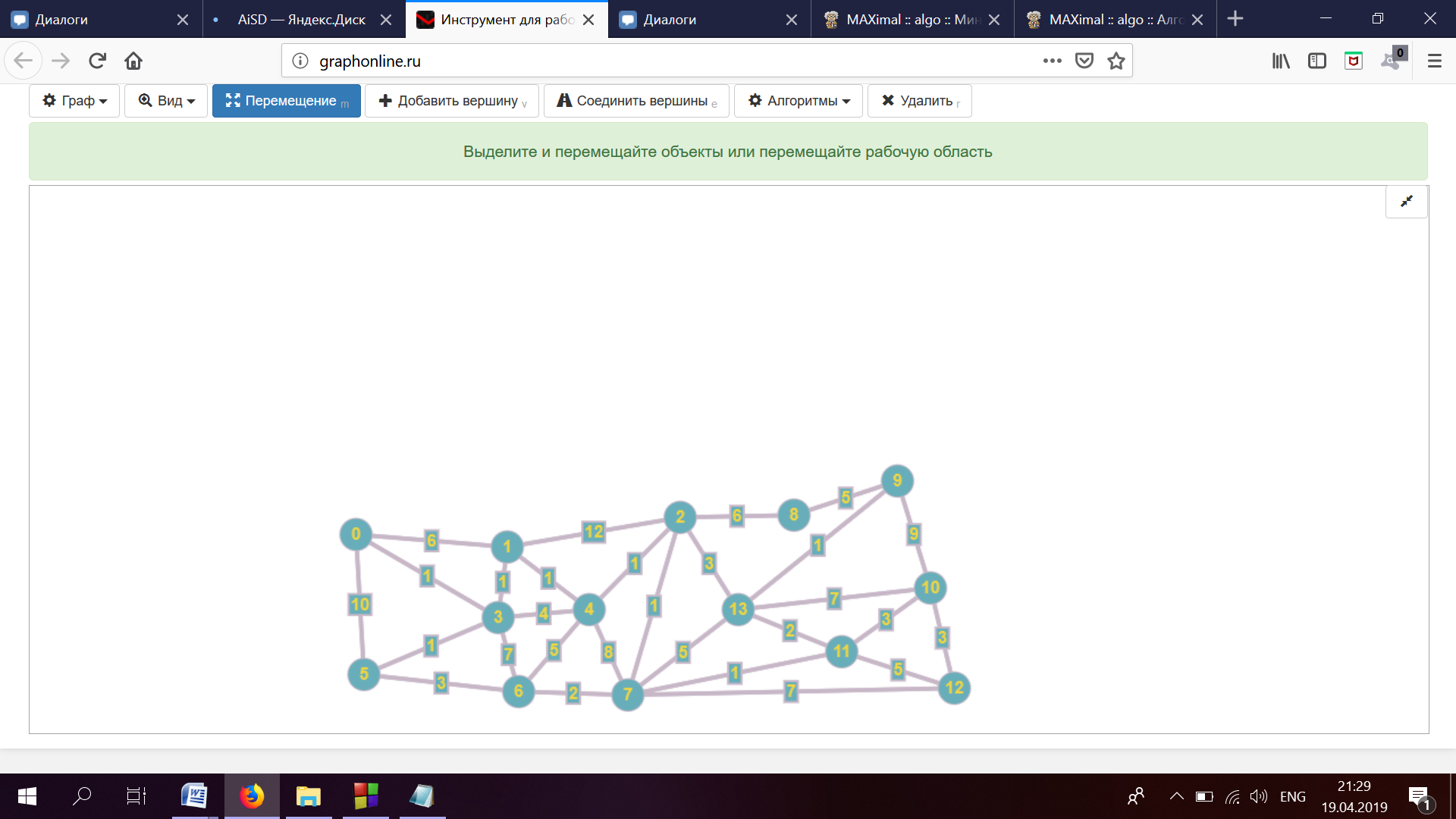
3) Функция с алгоритмом Прима

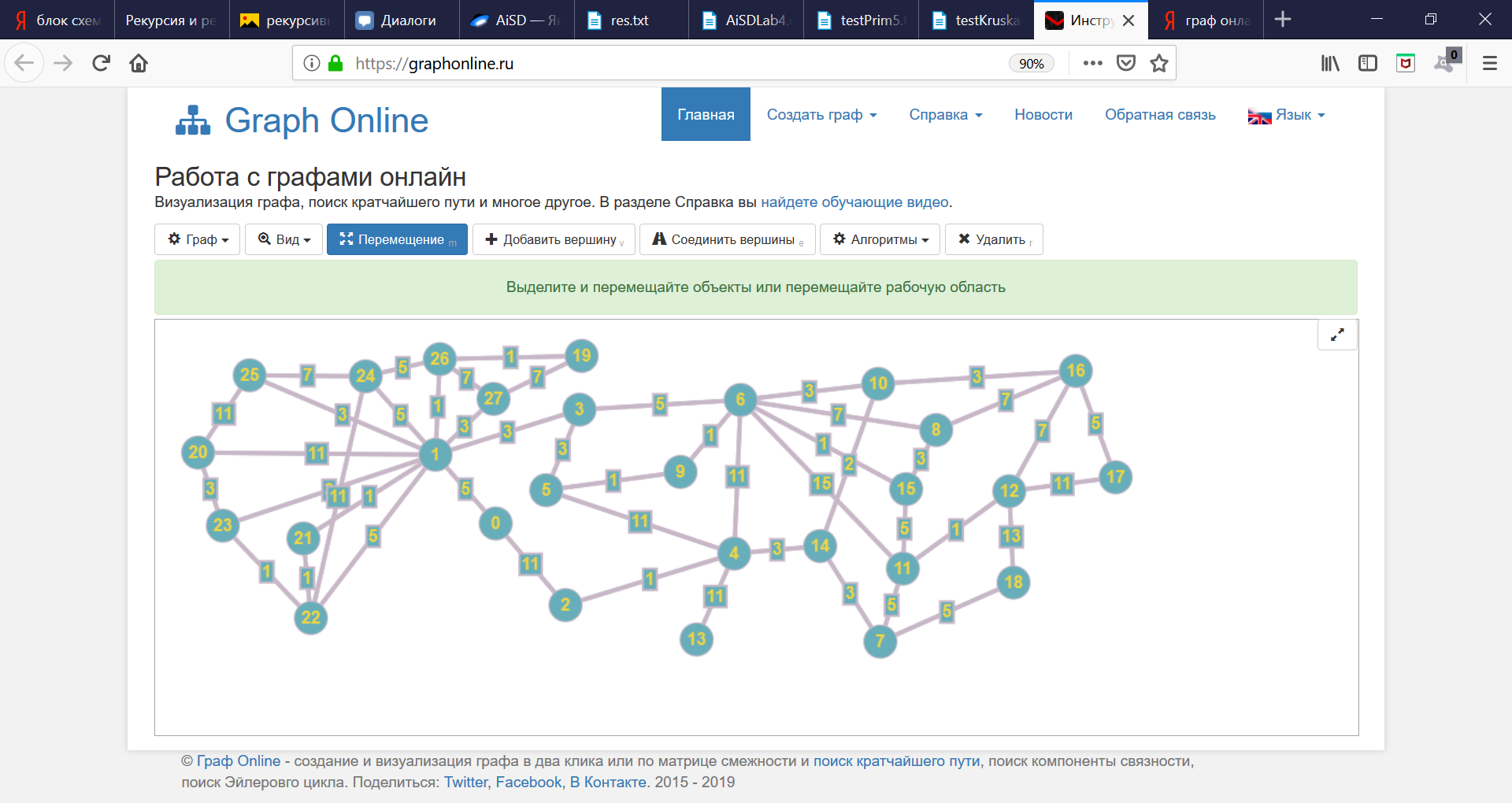
4) И основная функция, в которой происходит вызов вышесказанных функций и вывод результатов на экран.

**Тестирование алгоритмов:**

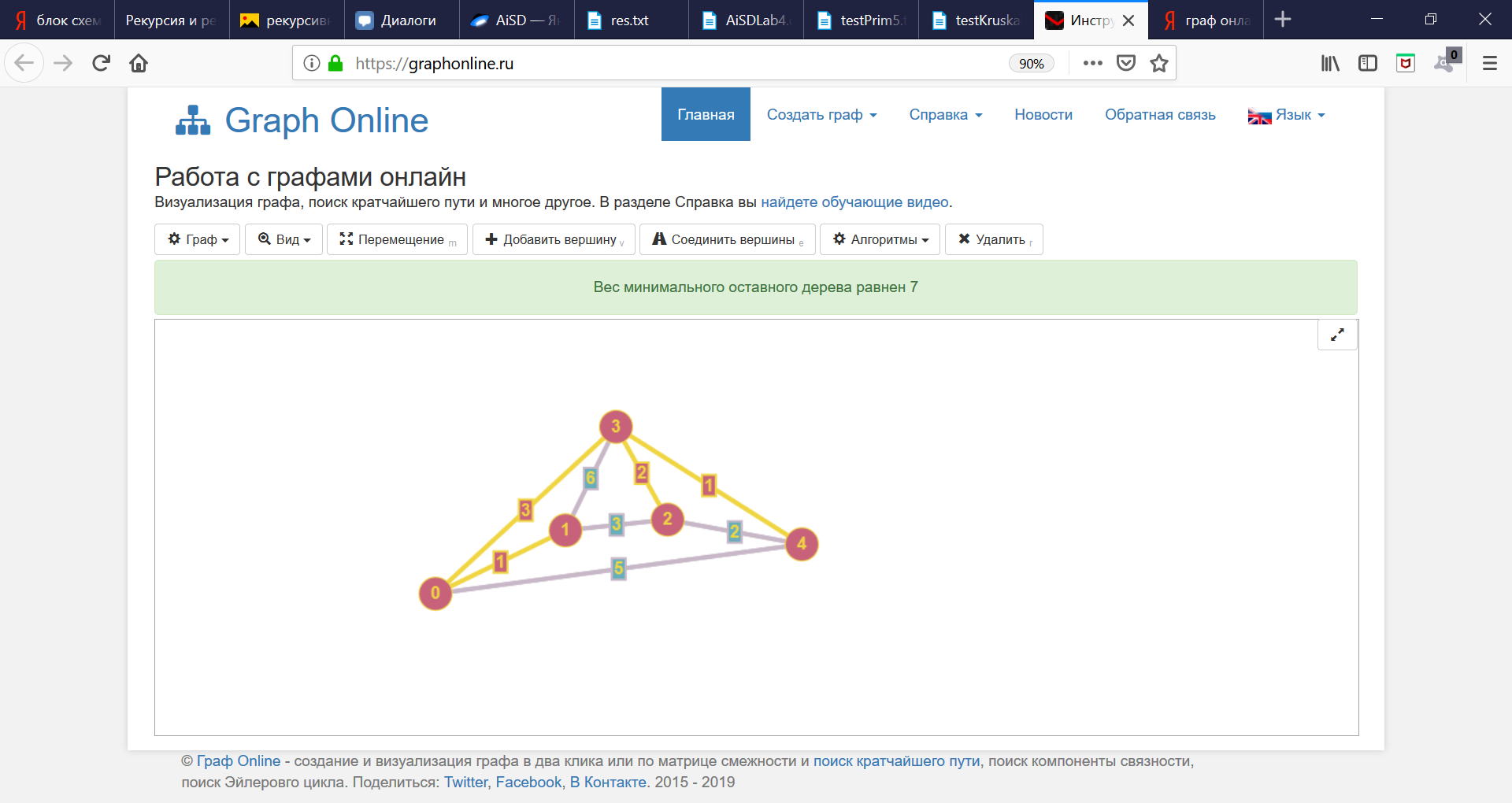
Для теста возьмем следующий граф:

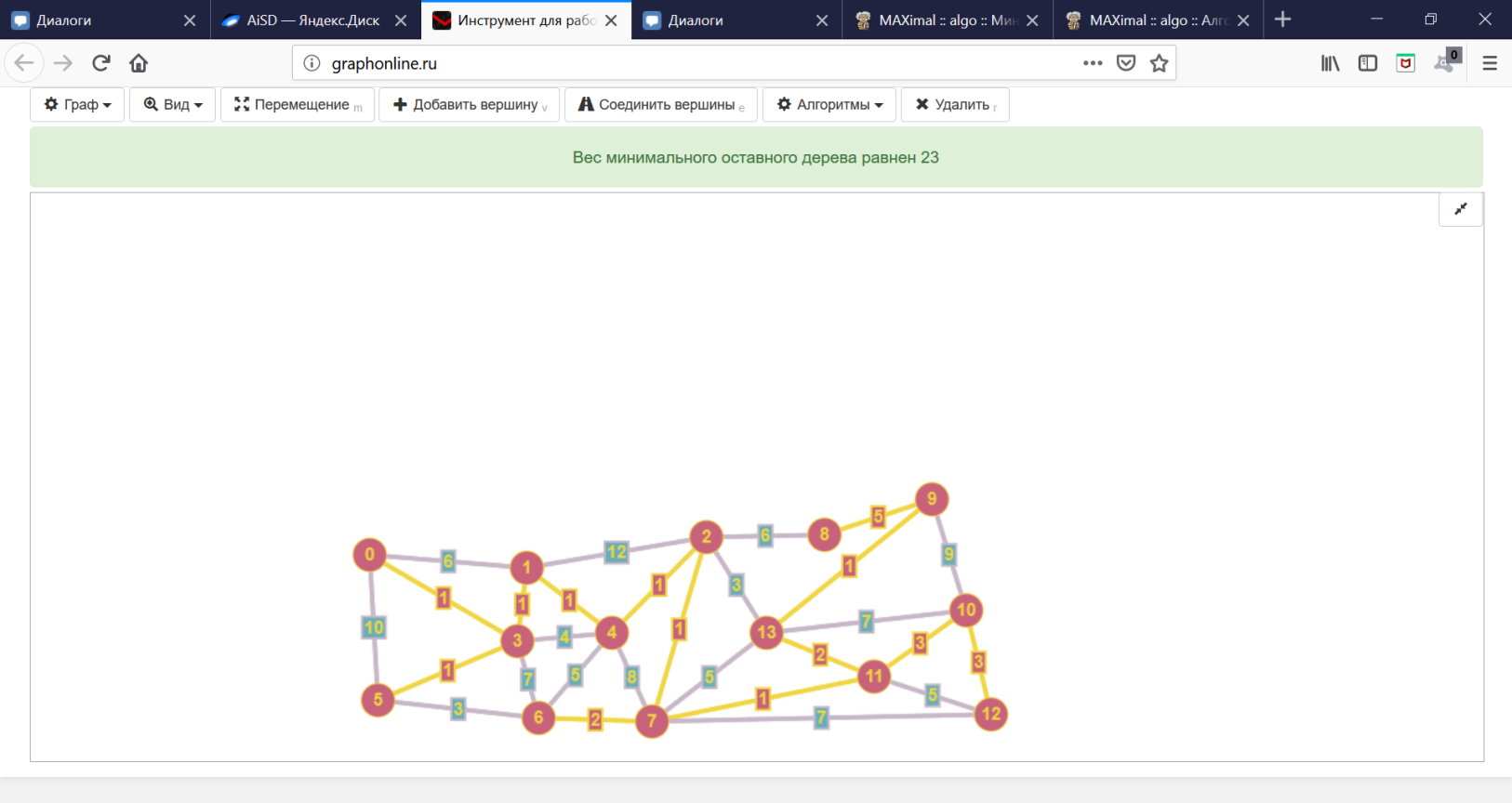


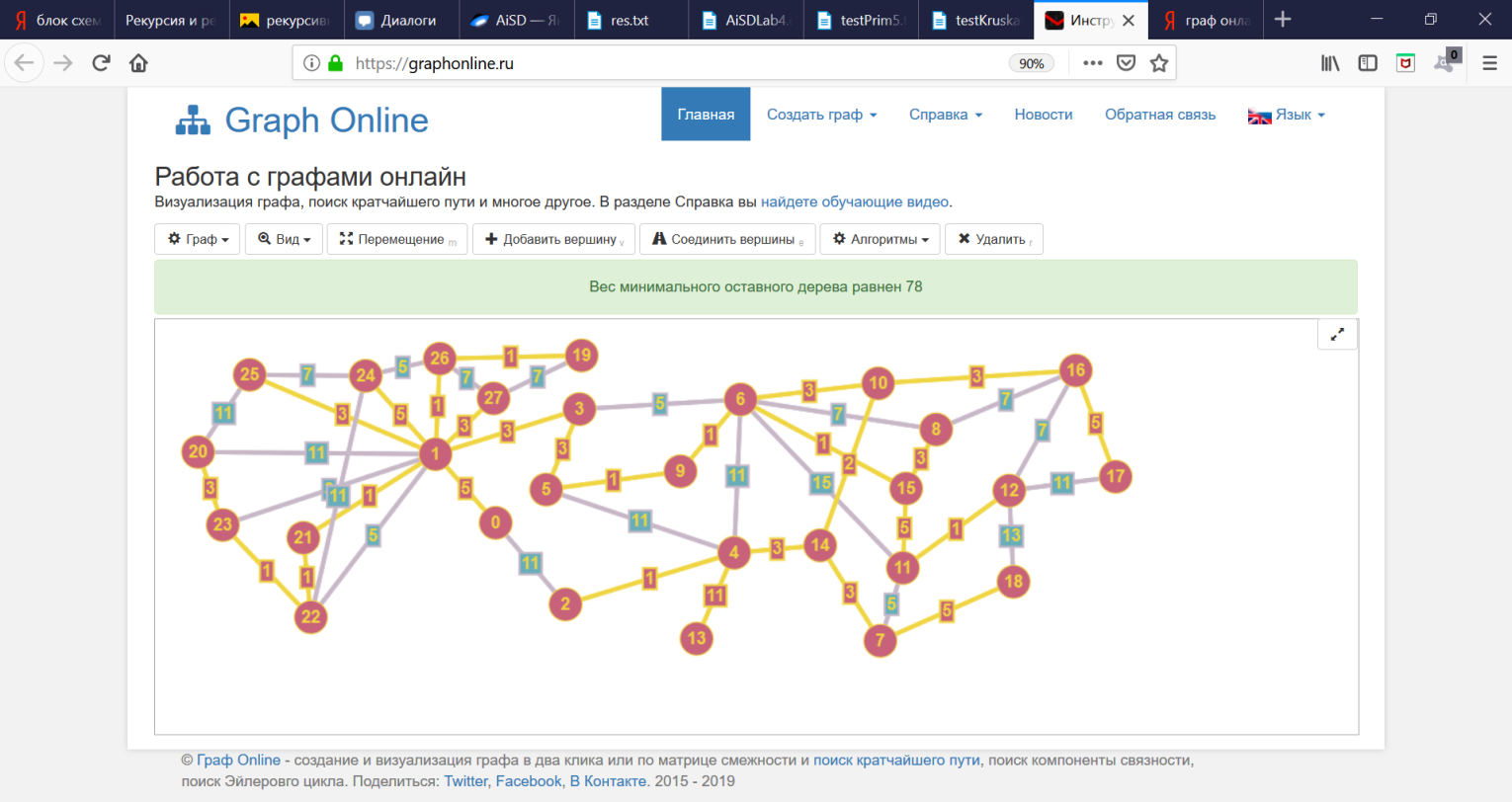




В результате работы программы получено следующее остовное дерево:







Затраченное время на нахождение остовного дерева.

|  |  |  |
| --- | --- | --- |
| Размера графа | Алгоритм Прима | Алгоритм Крускала |
| 5 | 0,014466 | 0,010268 |
| 14 | 0,02545 | 0,023644 |
| 19 | 0,03256 | 0,024412 |
| 28 | 0,05467 | 0,025 |

**Вывод:**

В ходе проведенной лабораторной работы были:

1) Изучены Алгоритм Крускала и Прима, которые строят остовное дерево.

2) Разработана программа на языке c++ в среде Codeblocks под систему Windows 10 x64 , считывающая граф из файла и считающая количество времени, за которые были построены остовные деревья данными алгоритмами.

3) И на основе полученных результатов был сделан вывод, что алгоритм Крускала быстрее Прима. Т.к асимптотика алгоритма Крускала **O (M log N + N2)**, а асимптотика алгоритма Прима **O (N2)**.

**Приложение. Код программы**

main.cpp

#include <vector>

#include <string>

#include <algorithm>

#include <fstream>

#include <iostream>

#include <cstdlib>

#include <ctime>

using namespace std;

const int INF = 999; // значение "бесконечность"

void prim(vector < vector<int> > g,int n)

{

vector<bool> used (n);

vector<int> min\_e (n, INF), sel\_e (n, -1);

min\_e[0] = 0;

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

{

int v = -1;

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

{

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

v = j;

}

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

{

cout << "No MST!";

exit(0);

}

used[v] = true;

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

cout << v << " " << sel\_e[v] << endl;

for (int to=0; to<n; ++to)

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

{

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

sel\_e[to] = v;

}

}

}

double go\_prim()

{

int n=14;

vector < vector<int> > g(n);

setlocale(0,"");

ifstream fin("in.txt");

int x;

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

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

{

fin>>x;

g.at(i).push\_back(x);

if(g[i][j]==0) g[i][j]=INF;

}

double start\_time,end\_time;

start\_time= clock();

prim(g,n);

end\_time=clock();

g.clear();

return end\_time-start\_time;

}

void kruskala(vector < pair < int, pair<int, int> > > g,int n,int m )

{

int cost = 0;

vector < pair<int, int> > res;

sort(g.begin(), g.end());

vector<int> tree\_id(n);

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

{

tree\_id[i] = i;

}

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

{

int a = g[i].second.first, b = g[i].second.second, l = g[i].first;

if (tree\_id[a] != tree\_id[b])

{

cost += l;

res.push\_back(make\_pair(a, b));

int old\_id = tree\_id[b], new\_id = tree\_id[a];

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

{

if (tree\_id[j] == old\_id)

{

tree\_id[j] = new\_id;

}

}

}

}

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

{

cout<<res[i].first<<" "<<res[i].second;

cout<<endl;

}

}

double go\_kruskal()

{

int n, m;

ifstream in("in1.txt");

in >> n;

in >> m;

vector < pair < int, pair<int, int> > > g(m);

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

{

in >> g[i].second.first;

in >> g[i].second.second;

in >> g[i].first;

}

in.close();

double start\_time,end\_time;

start\_time= clock();

kruskala(g,n,m);

end\_time=clock();

return end\_time-start\_time;

}

int main()

{

setlocale(0,"");

double srprim=0;

double srkruskal=0;

int N=500;

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

{

srprim+=go\_prim();

srkruskal+=go\_kruskal();

}

srprim/=N\*1000;

srkruskal/=N\*1000;

cout<<"Алгоритм Прима нашел оставное дерево за : "<<srprim<<" секунд"<<endl;

cout<<"Алгоритм Крускала нашел оставное дерево за : "<<srkruskal<<" секунд";

return 0;

}