**Санкт-Петербургский**

**Государственный электротехнический университет**

**Факультет Компьютерных Технологий и Информатики**

**Кафедра МО ЭВМ**

****

*Курсовая работа по теме*

*«Алгоритмы на графах, алгоритм Крускала»*

*4 семестр.*

**Выполнил студент: гр.2304 Комаров Д.Н.**

**Проверил преподаватель: Балтрашевич В.Э.**

***Санкт-Петербург***

***2014 год.***

Оглавление

[1. Постановка задачи: 3](#_Toc375518184)

[2. Формальная постановка задачи: 3](#_Toc375518185)

[3. Пример. 3](#_Toc375518186)

[4. Спецификация программы: 3](#_Toc375518187)

[5. Некоторые используемые функции 4](#_Toc375518188)

[6. Текст программы: 5](#_Toc375518189)

[7. Результаты работы программы: Интерфейс 8](#_Toc375518190)

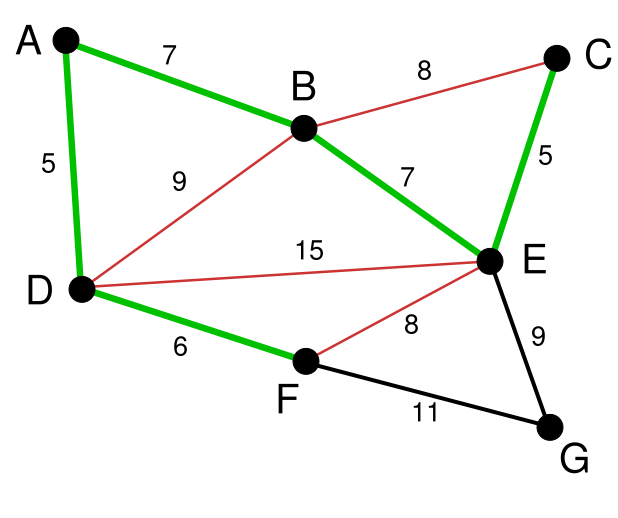
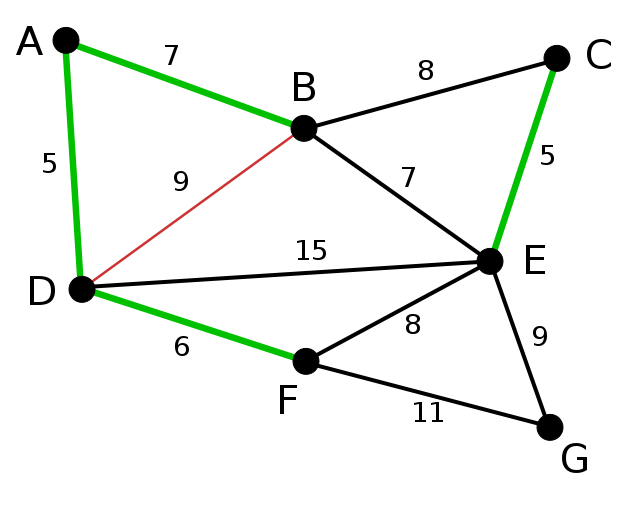
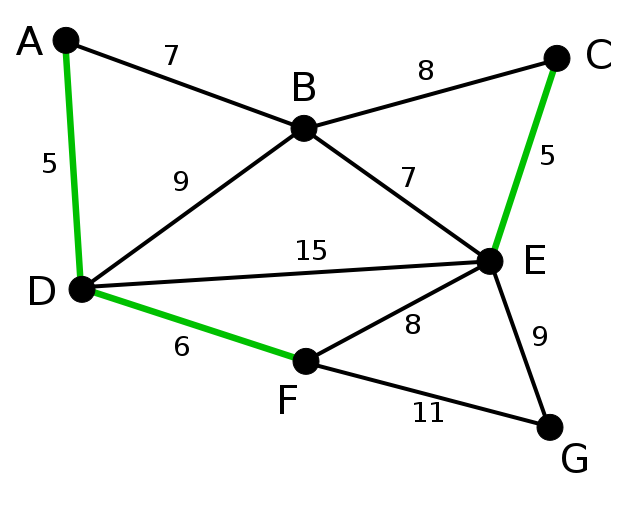
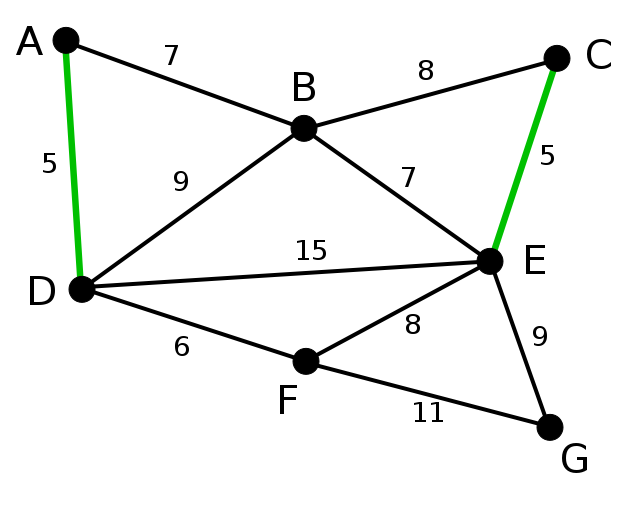
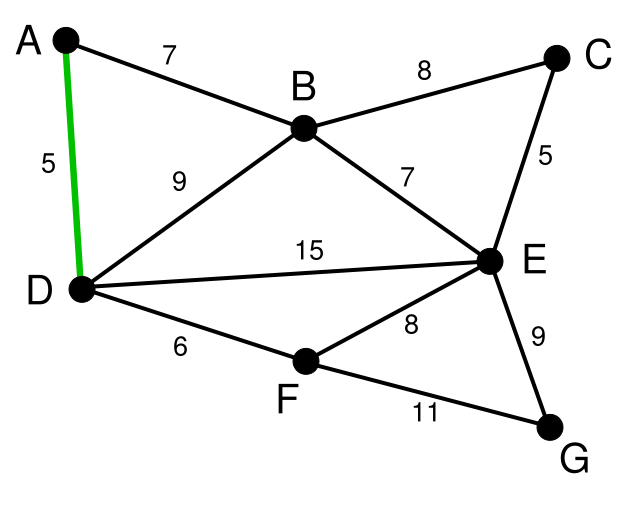
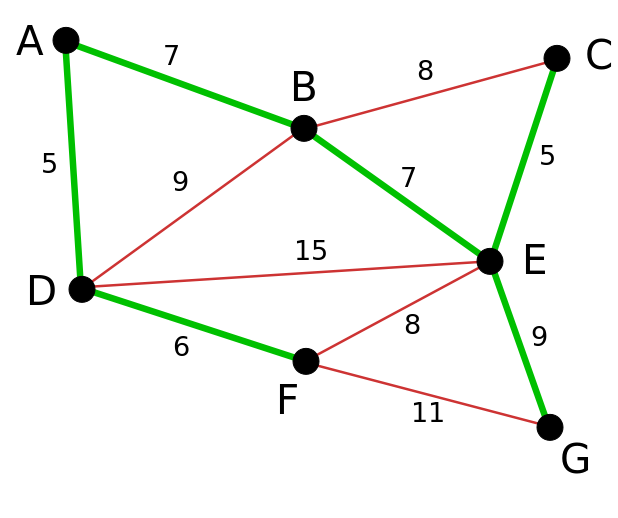
# Постановка задачи:

Минимальное остовное дерево (Kruskal). Реализовать жадный алгоритм Крускала построения минимального остовного дерева заданного графа.

# Формальная 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.

# Пример.

# Спецификация программы:

**Свойства минимального остова**

* Минимальный остов уникален, если веса всех рёбер различны. В противном случае, может существовать несколько минимальных остовов (конкретные алгоритмы обычно получают один из возможных остовов).
* Минимальный остов является также и остовом с минимальным произведением весов рёбер.

(доказывается это легко, достаточно заменить веса всех рёбер на их логарифмы)

* Минимальный остов является также и остовом с минимальным весом самого тяжелого ребра.

(это утверждение следует из справедливости алгоритма Крускала)

* Остов максимального веса ищется аналогично остову минимального веса, достаточно поменять знаки всех рёбер на противоположные и выполнить любой из алгоритм минимального остова.

**Алгоритм Крускала**

Данный алгоритм был описан Крускалом (Kruskal) в 1956 г.

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.

# Некоторые используемые функции

|  |  |
| --- | --- |
| int cmp(const void \*a,const void \*b) | Функция "сравнения" двух ребер, используемая для сортировки |
| int getColor(int n) | Функция получает цвет вершины n-й по порядку. если nodes[n] < 0, то вершина n имеет цвет nodes[n], если nodes[n] >= 0, то вершина n имеет цвет такой же, как и вершина с номером nodes[n] |
| void krusk(ifstream &input, ofstream &output, int NV, float xC, float xY) | Реализация алгоритма Крускала |
| Void set\_coord(coordEdge \*tree, int \_NV, float xC, float yC, int rad){ | Задает координаты вершинам графа |

# Текст программы:

Form1.h

private: System::Void button1\_Click(System::Object^ sender, System::EventArgs^ e) {

string path;

MarshalString(openFileDialog1->FileName, path);

if (openFileDialog1->FileName != "")

{

ifstream input(path);

ofstream output("output.txt");

int NV;

input.seekg(0);

input >> NV;

krusk(input, output, NV, 150, 150);

System::IO::StreamReader ^ sr = gcnew System::IO::StreamReader("output.txt");

textBox2->Text = sr->ReadToEnd();

sr->Close();

}

}

private: System::Void button2\_Click(System::Object^ sender, System::EventArgs^ e) {

if(openFileDialog1->ShowDialog() == System::Windows::Forms::DialogResult::OK)

{

System::IO::StreamReader ^ sr = gcnew

System::IO::StreamReader(openFileDialog1->FileName);

textBox1->Text = sr->ReadToEnd();

sr->Close();

}

}

private: System::Void Form1\_Load(System::Object^ sender, System::EventArgs^ e) {

openFileDialog1->FileName = "";

}

private: System::Void set\_coord(coordEdge \*tree, int \_NV, float xC, float yC, int rad){

float c\_iter = (2\*pi )/ \_NV;

float iter = c\_iter;

float mX = rad;

float mY = 0;

float m1;

float m2;

for (int i = 0; i < \_NV; i++)

{

m1 = (cos(iter)\*mX + sin(iter)\*mY);

m2 = (cos(iter)\*mY - sin(iter)\*mX);

tree[i].x = xC + m1;

tree[i].y = yC + m2;

iter = iter + c\_iter;

}

}

private: System::Void Draw(int from, int to, coordEdge \*coord, float xC, float xY, int \_NV, int rad, Graphics ^image) {

image->DrawLine(Pens::Green, coord[from].x, coord[from].y, coord[to].x, coord[to].y);

}

private: System::Void helpToolStripMenuItem\_Click(System::Object^ sender, System::EventArgs^ e) {

System::IO::StreamReader ^ sr = gcnew

System::IO::StreamReader("Help.txt", System::Text::Encoding::GetEncoding(1251));

MessageBox::Show(sr->ReadToEnd());

sr->Close();

}

private:System::Void krusk(ifstream &input, ofstream &output, int NV, float xC, float xY)

{

Bitmap ^uh = gcnew Bitmap(2\*xC, 2\*xY);

Graphics ^image = Graphics::FromImage(uh);

image->Clear(Color::White);

pictureBox1->Image = uh;

image->DrawEllipse(Pens::Red, 100, 100, 100, 100);

coordEdge \*treeEdge = new coordEdge[NV];

set\_coord(treeEdge, NV, 150, 150, 50);

Bitmap ^uhc = gcnew Bitmap(2\*xC, 2\*xY);

Graphics ^egami = Graphics::FromImage(uhc);

egami->Clear(Color::White);

pictureBox2->Image = uhc;

egami->DrawEllipse(Pens::Red, 100, 100, 100, 100);

int i;

// Считываем вход

input >> NE;

for(i = 0; i < NV; i++)

{

image->DrawString(Convert::ToString(i),System::Drawing::SystemFonts::DefaultFont, Brushes::Brown, treeEdge[i].x, treeEdge[i].y);

egami->DrawString(Convert::ToString(i),System::Drawing::SystemFonts::DefaultFont, Brushes::Brown, treeEdge[i].x, treeEdge[i].y);

nodes[i] = -1-i;

}

for(i = 0; i < NE; i++)

{

input >> edges[i].n1;

input >> edges[i].n2;

input >> edges[i].w;

Draw(edges[i].n1,edges[i].n2 , treeEdge, 150, 150, NV, 50, image);

}

input.close();

// Алгоритм Краскала

// Сортируем все ребра в порядке возрастания весов

qsort(edges, NE, sizeof(edge), cmp);

for(i = 0; i < NE; i++){

int c2 = getColor(edges[i].n2);

if ( getColor (edges[i].n1) != c2 ){

nodes [last\_n] = edges[i].n2;

output << edges[i].n1 << " ";

output << edges[i].n2 << " ";

output << edges[i].w << " ";

output << endl;

Draw(edges[i].n1,edges[i].n2 , treeEdge, 150, 150, NV, 50, egami);

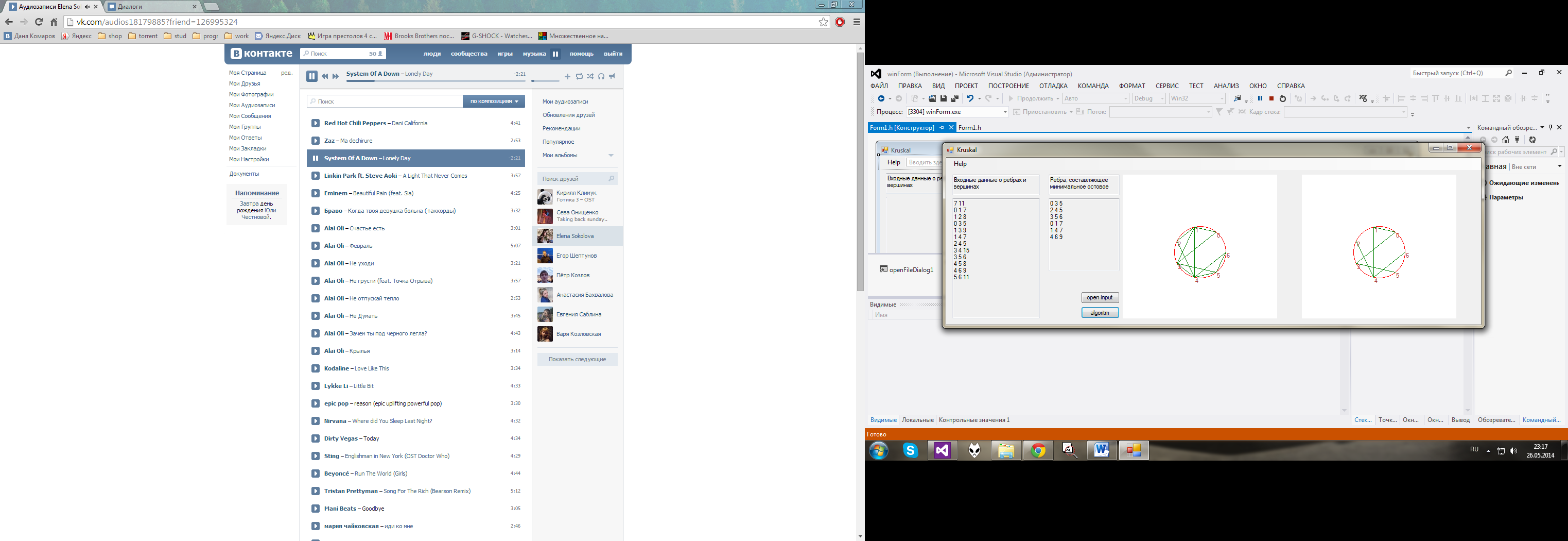
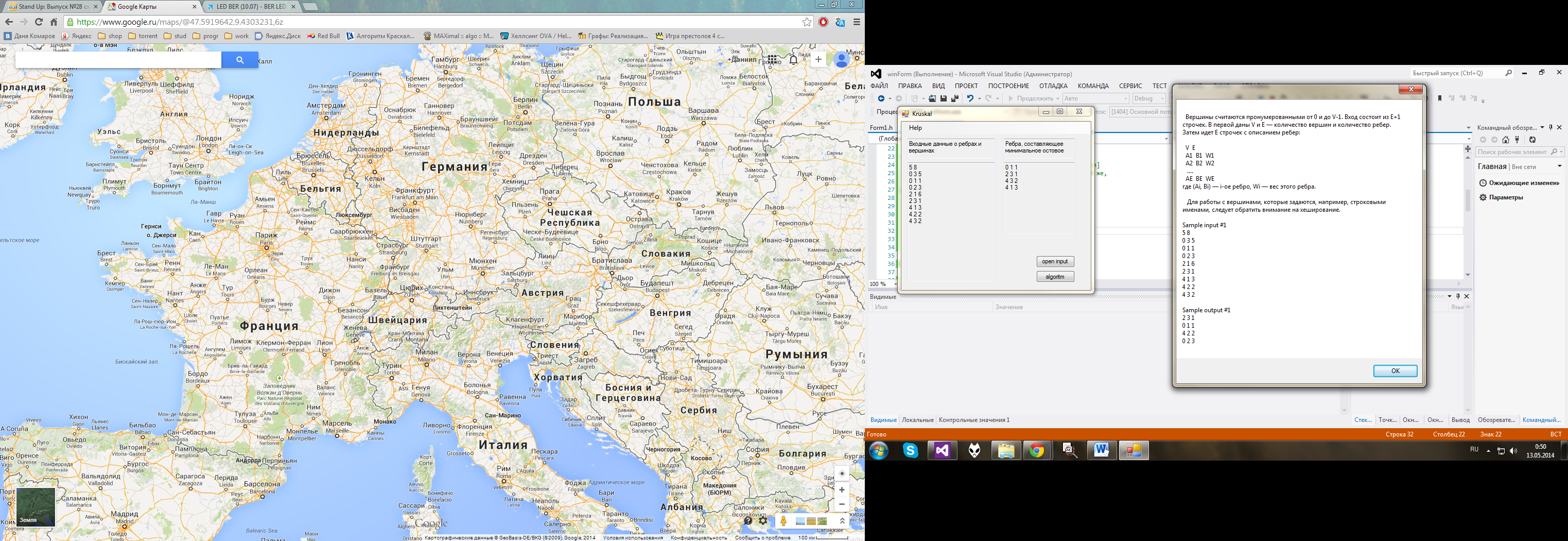
}

}

output.close();

}

# Результаты работы программы: Интерфейс

**Вывод:** Этот код самым непосредственным образом 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 - 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. Учитывая, что всего операций объединения будет N-1, мы и получаем асимптотику O (M log N + N2).