Федеральное государственное автономное образовательное учреждение

«Санкт-Петербургский государственный электротехнический университет

«ЛЭТИ» им. В.И. Ульянова (Ленина)»

Отчет

по практическому заданию

Выполнили:

Диогенова К.А.

Казачкова А.Д.

Факультет: КТИ

Группа №3304

Преподаватель:

Фирсов М.А.

Подпись преподавателя

Санкт-Петербург

2015

1. **Формулировка задания**

Разработать программу, визуализирующую алгоритм нахождения минимального остовного дерева алгоритмом Крускала, на языке Java с использованием библиотек “Swing”, “AWT”. Программа должна иметь простой в использовании, удобный интерфейс.

1. **Спецификация**
2. *Внешняя*

Граф задается в файле. Работа алгоритма будет продемонстрирована следующим образом:

1 шаг:

5

9

3

11

2 шаг:

9

5

11

3

3 шаг:

9

11

5

3

4 шаг:

5

3

9

11

5 шаг:

5

11

9

3

Т.е. выбираемое на каждом шаге ребро подсвечивается голубым цветом. Ребра уже вошедшие в МОД подсвечиваются красным. В итоге получится выделенное красным цветом минимальное остовное дерево.

Если очередное рассматриваемое ребро, при включении его в МОД, образует цикл, то выводится соответствующее сообщение и ребро становится черным.

Предполагается, что окно программы будет выглядеть следующим образом:

Алгоритм Краскала

Граф

Далее

Область отрисовки графа

Выбрать файл, из которого следует считать граф, можно, нажав «Граф.. Выбрать файл».

Для генерации графа необходимо нажать «Граф.. Генерация». Для просмотра ребер нужно нажать «Граф.. Показать ребра». Для перехода к следующему шагу алгоритма надо нажать «Далее».

1. *Внутренняя*

Граф должен представляться классом, расширяющим класс JPanel и реализующим класс потока, также должен содержать такие члены-данные:

* Массив ребер, вошедших в МОД
* Массив вершин
* Число вершин
* Число ребер
* Массив ребер
* Текущий поток

И следующие методы:

* Реализация самого алгоритма Краскала
* 2 конструктора с параметрами
* Отрисовка самого графа

1. *Формат входного файла*

Файл должен содержать последовательно количество вершин, количество ребер, номер вершины из, номер вершины в, стоимость ребра.

Пример:

*5*

*3*

*0 2 6*

*0 1 5*

*2 4 7*

1. **Использование структур данных**

* Граф, описанный в спецификации

public class Graph extends JPanel implements Runnable{

public int numV;

public int numE;

public Thread graphThread;

public Vertex v[];

public Edge e[];

public Edge MFT[];

…

}

* Массив булевских переменных размером число вершин на число вершин. Этот массив используется в алгоритме Краскала для определения подграфа в котором содержится вершина

**Структура данных:**

Вершина: содержит следующие члены-данные

* Координаты x, y для отрисовки
* Свой порядковый номер в массиве вершин

class Vertex{

public int x, y; //координаты вершины

public int num;

}

Ребро:

* Ссылка на вершину от которого рисуется
* Ссылка на вершину к которому рисуется
* Стоимость ребра

class Edge{

Vertex from;

Vertex to;

int weight;

}

1. **Распределение обязанностей**

|  |  |
| --- | --- |
| Диогенова К.А. | * Реализация алгоритма Краскала * Создание репозитория * Оформление отчета * Проведение тестирования |
| Казачкова А.Д. | * Визуализация алгоритма Краскала * Раелизация графа (структура графа и конструкторы) * Оформление отчета |

1. **План разработки**

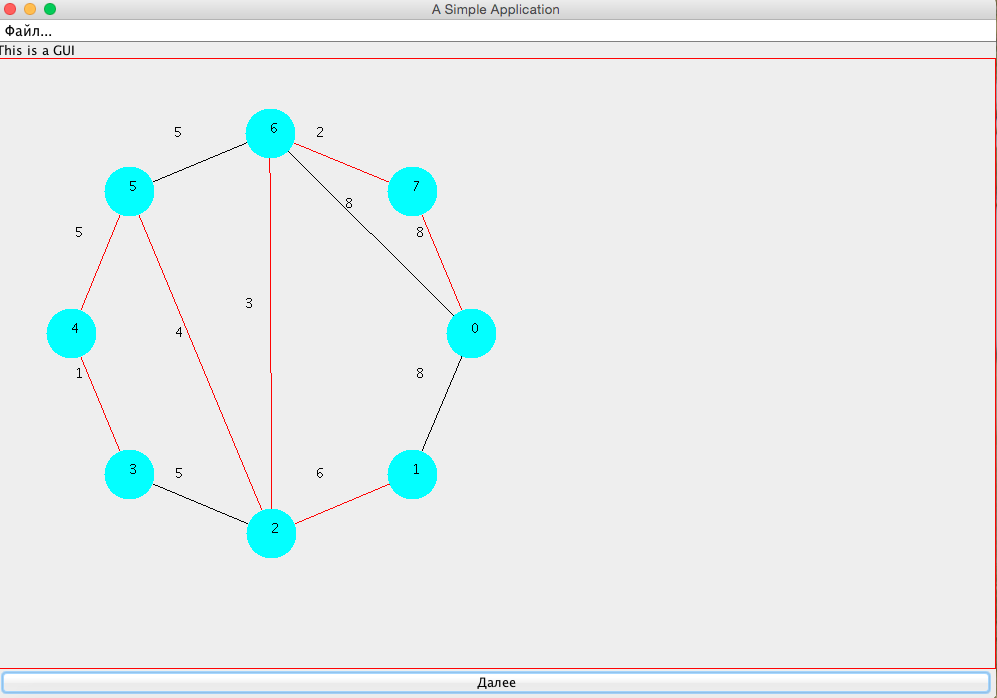
|  |  |
| --- | --- |
| 24.06 – 25.06 | * Оформление спецификации * Распределение обязанностей * Создание репозитория |
| 25.06 – 26.06 | * Написание прототипа программы (граф умеет отрисовываться и инициализироваться) |
| 27.06 – 29.06 | * Создание первой версии программы (рабочий алгоритм Краскала) |
| 29.06 – 1.07 | * Создание второй версии программы (визуализация алгоритма Краскала) |
| 2.07 | * Устранение возможных недочетов в программе и включение по возможности подсвечивания цикла графа |
| 3.07 | * Окончательная сдача финальной версии программы (включает при возможности подсветку циклов) |

**Уточнения ко второй версии, возникшие во время работы над первой:**

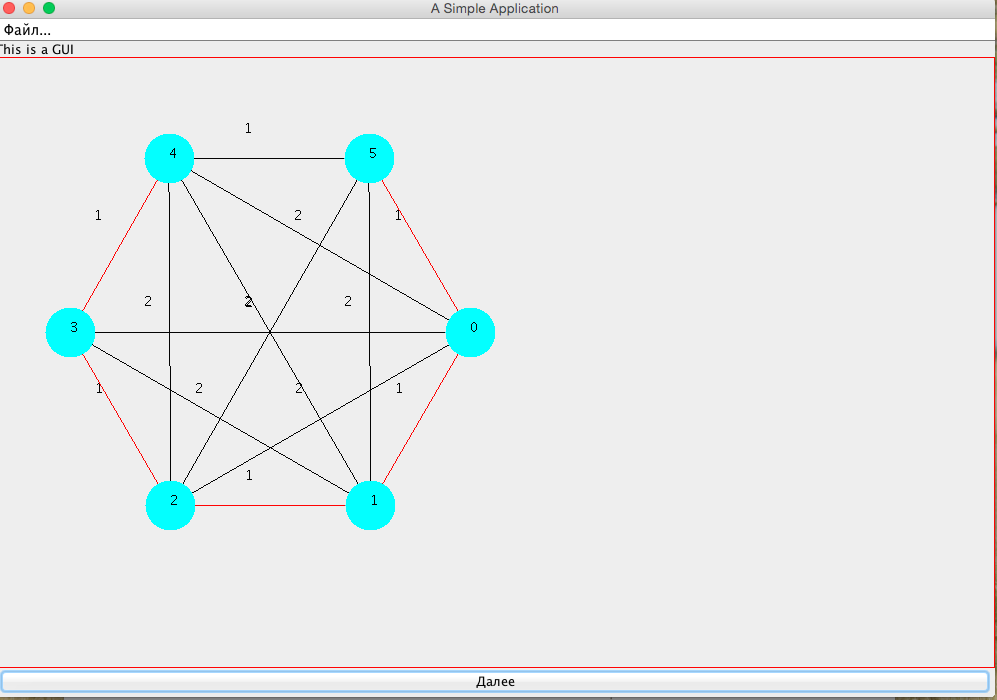
1. Улучшить формат входного файла;
2. Улучшить вывод веса ребер;
3. Загрузка файла и его выбор;
4. При большом числе изменять радиус вершины;
5. Генерация графа (количество вершин и процент ребер);
6. Добавить окно списка ребер.
7. **Тестирование первой версии программы (рабочий алгоритм Краскала)**

Красным выделено минимальное остовное дерево.

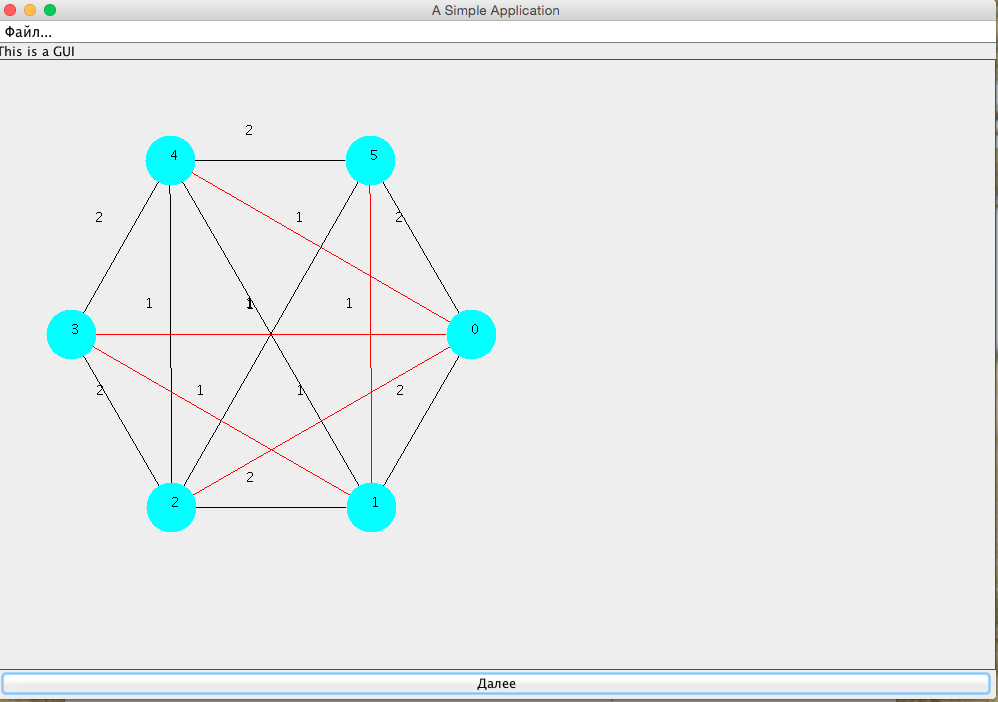
1. Пример с возможным циклом



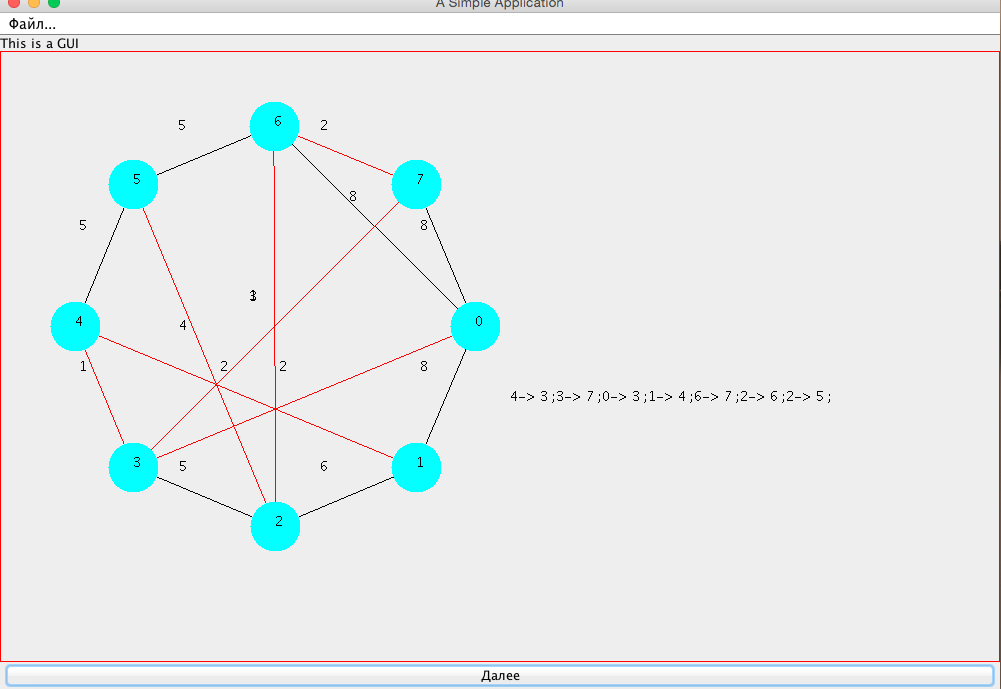
1. Пример с возможным циклом



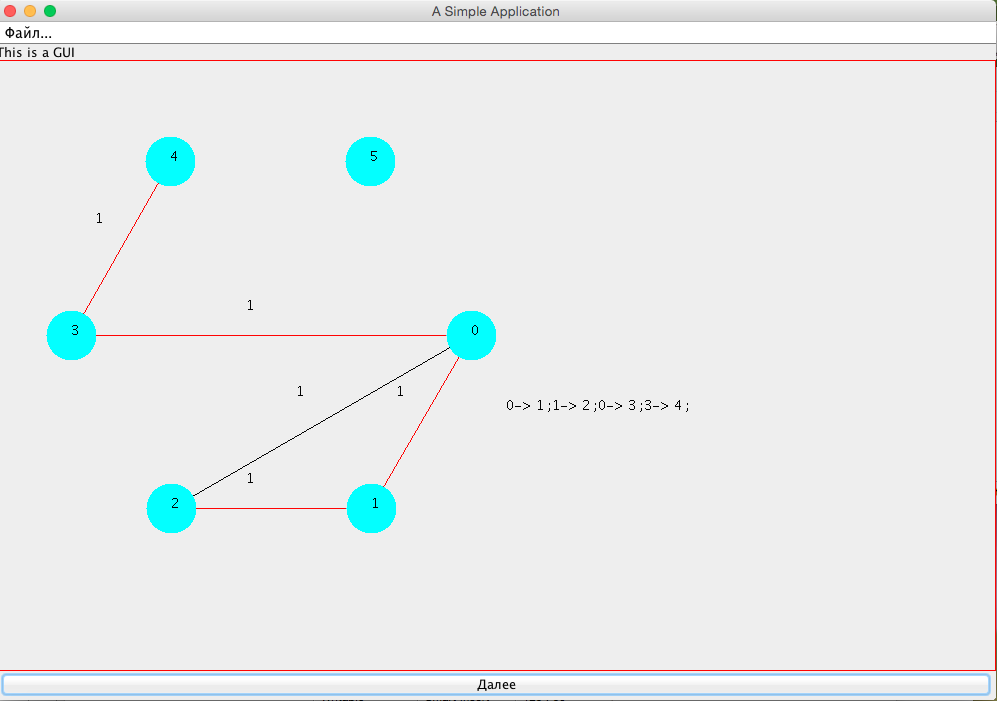
1. Пример с возможным циклом



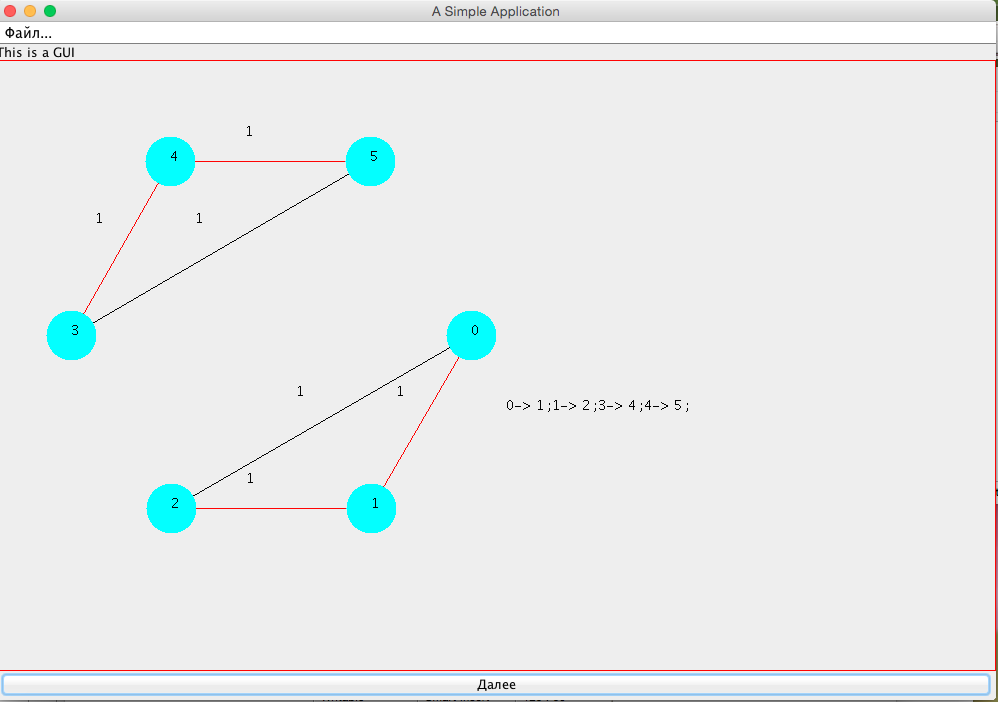
1. Пример с большим количеством ребер



1. Пример с изолированной вершиной

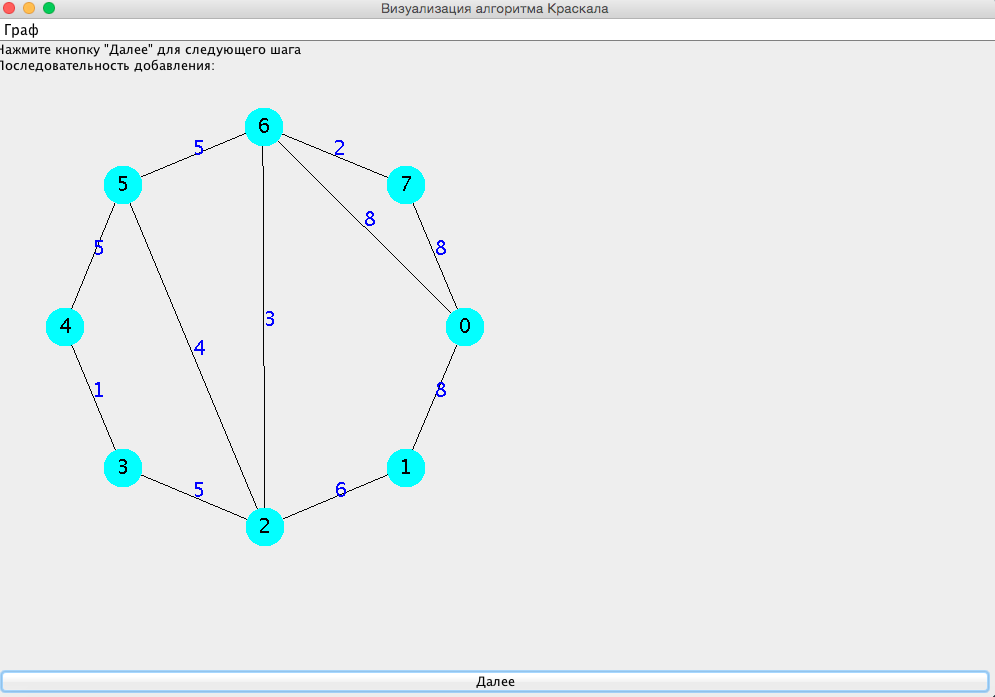


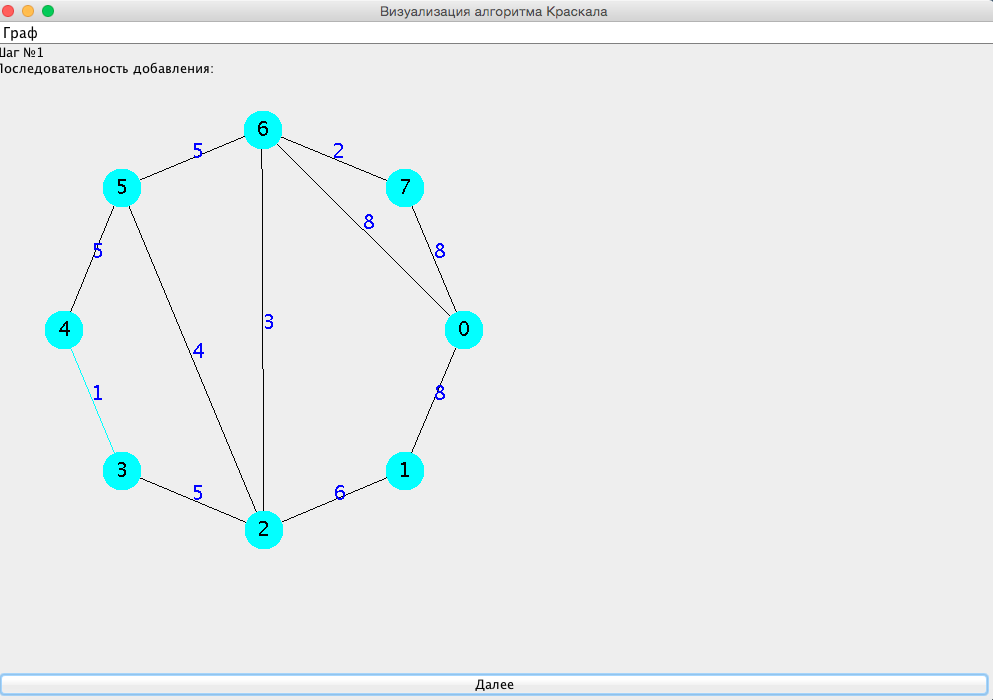
1. Пример графа в котором не все вершины соединены ребрами

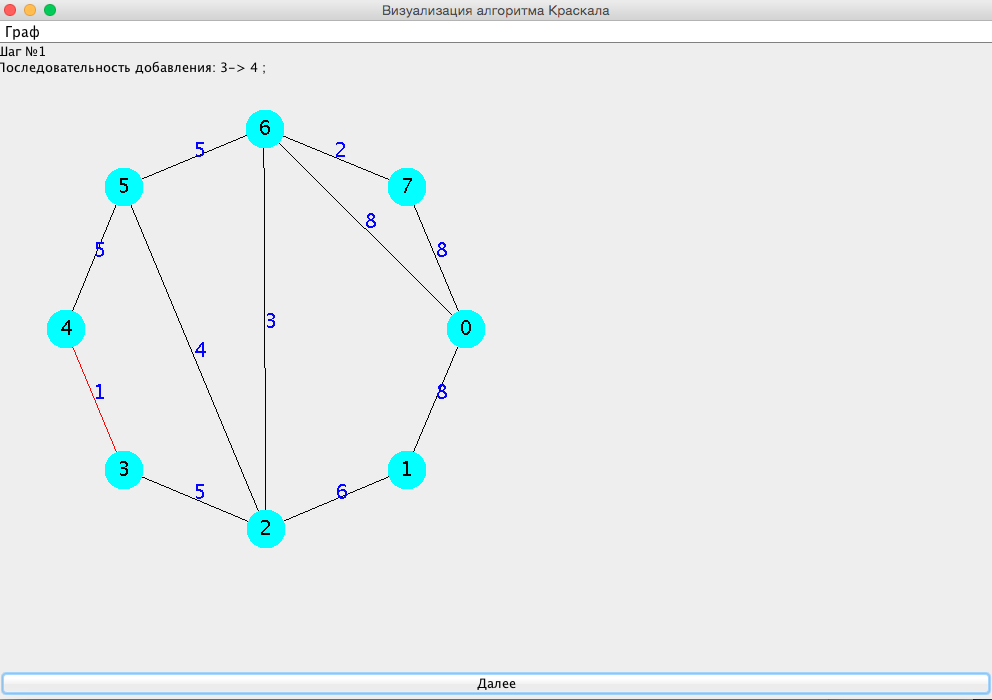


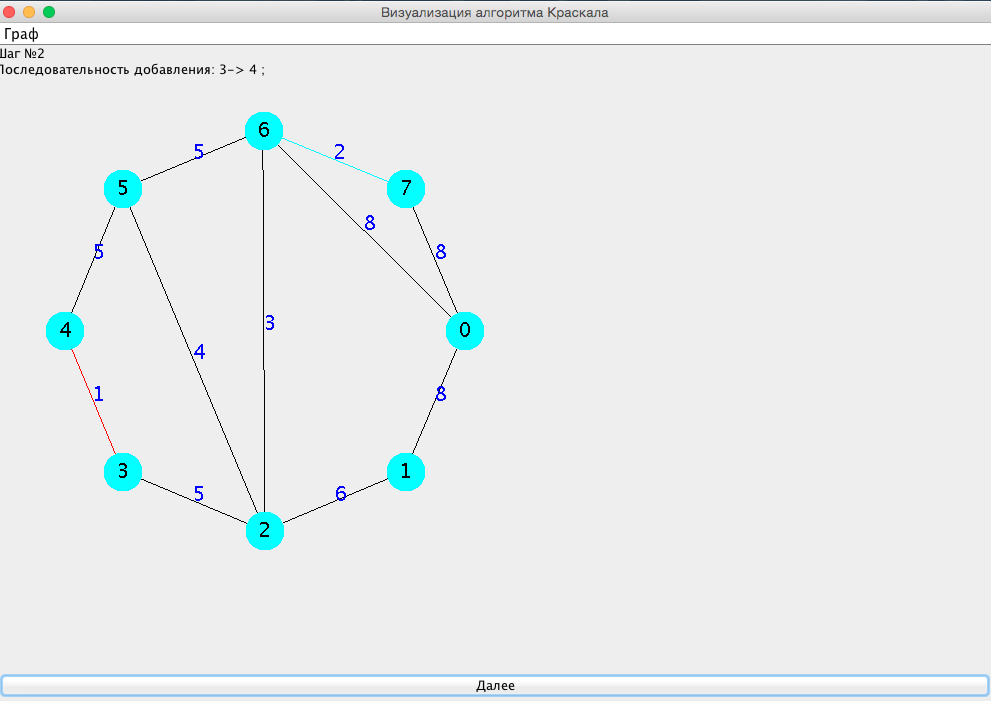
Как видим, алгоритм правильно находит МОД.

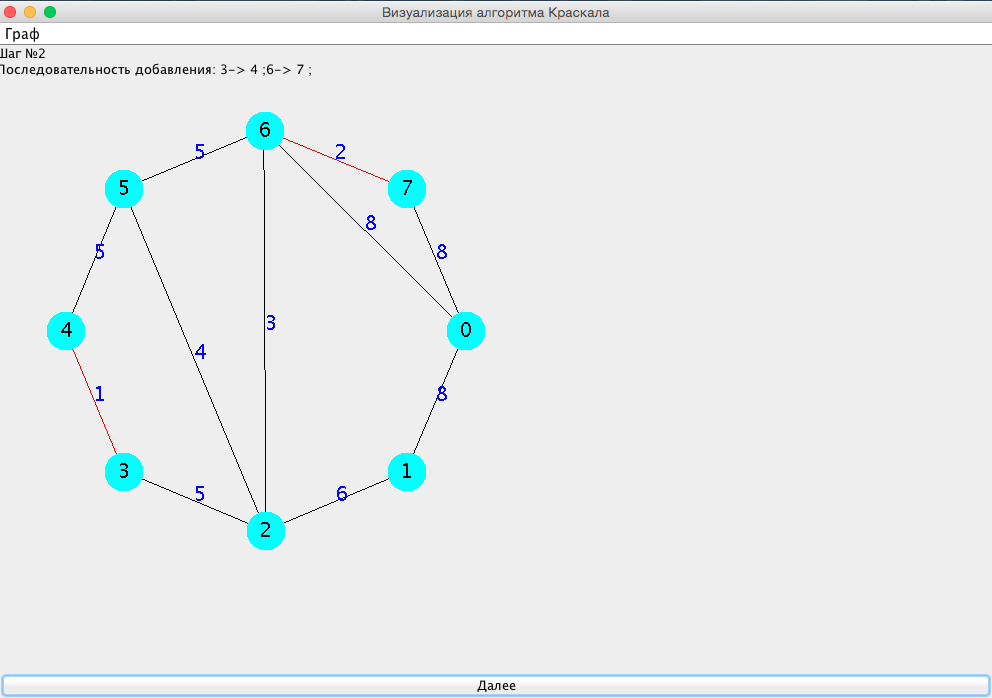
1. **Тестирование визуализации работы алгоритма**

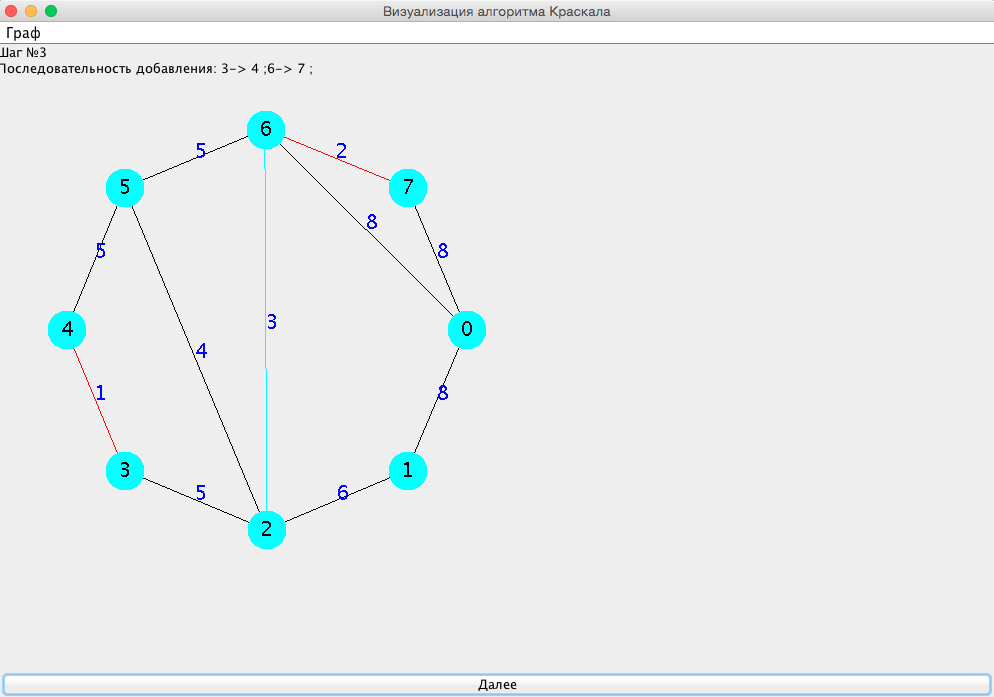
****

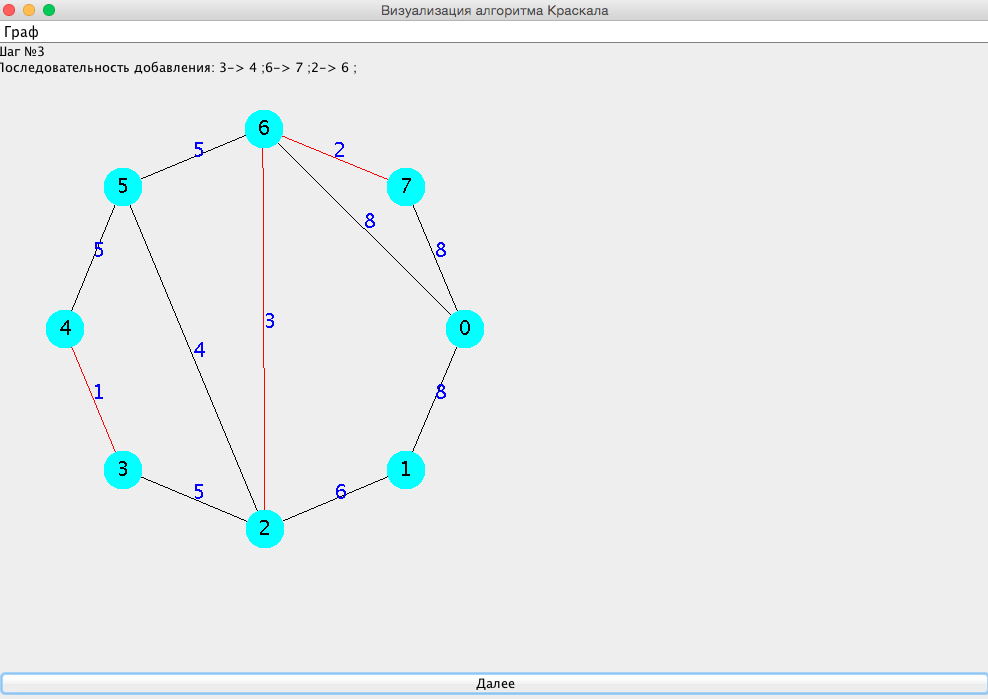
****

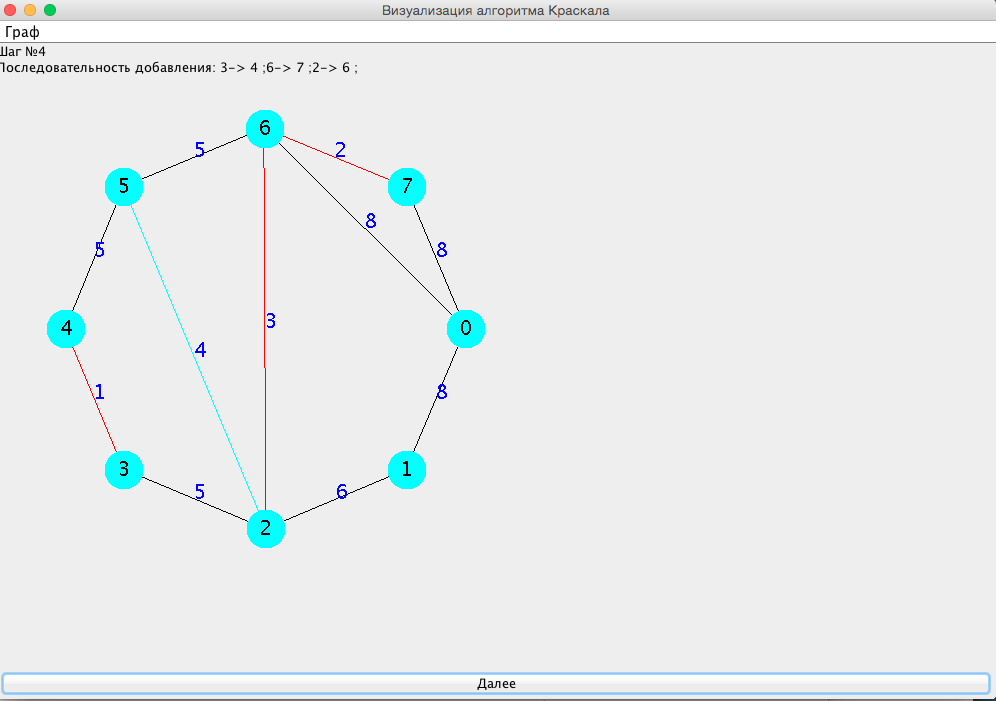
****

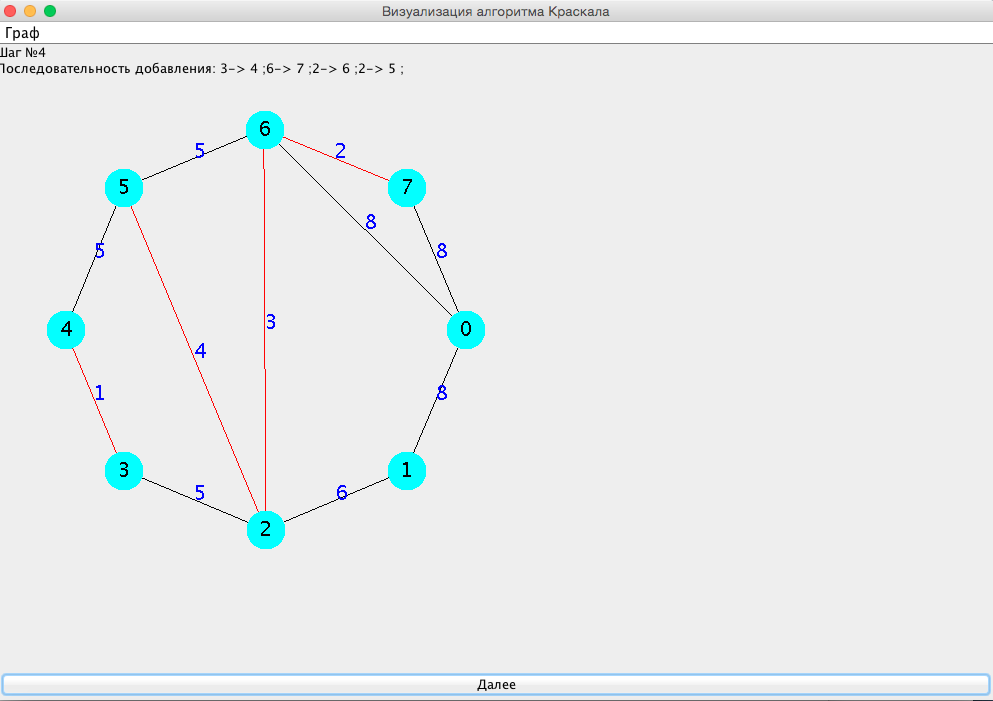
****

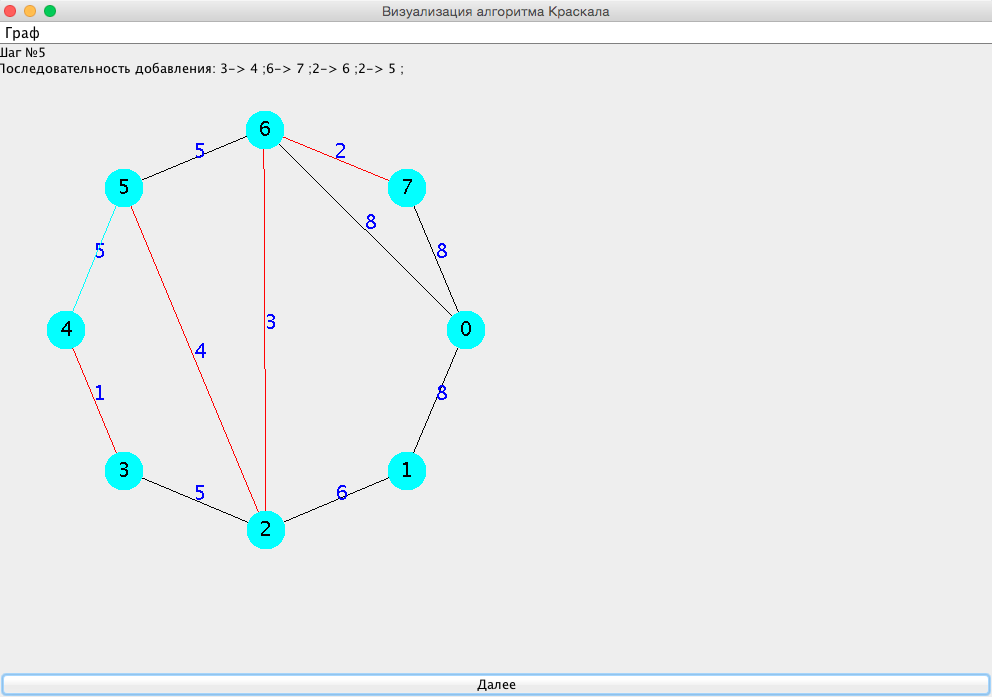
****

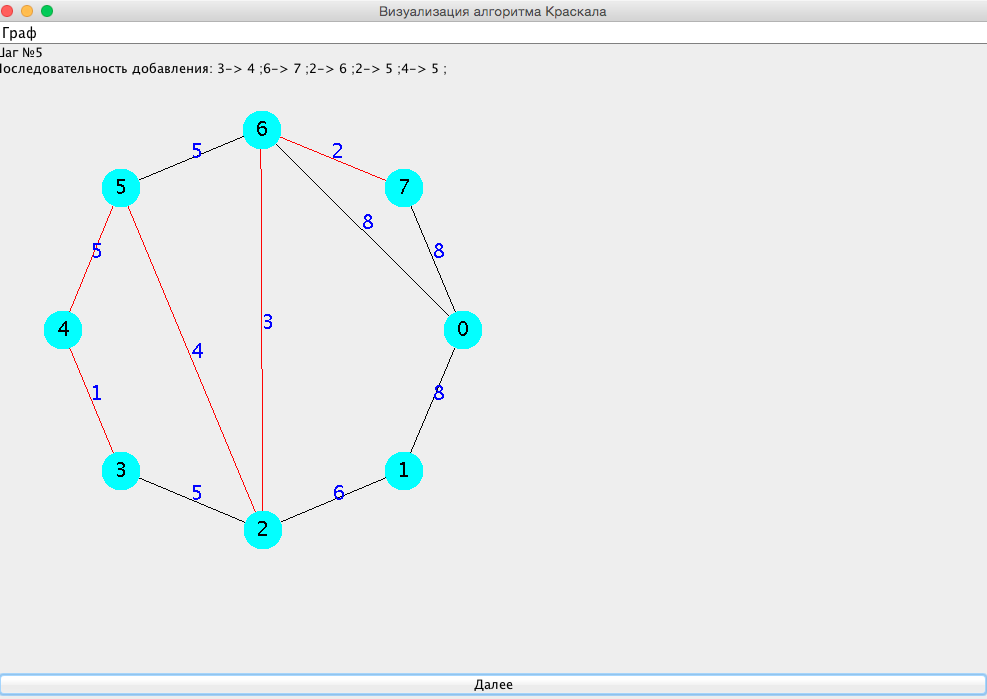
****

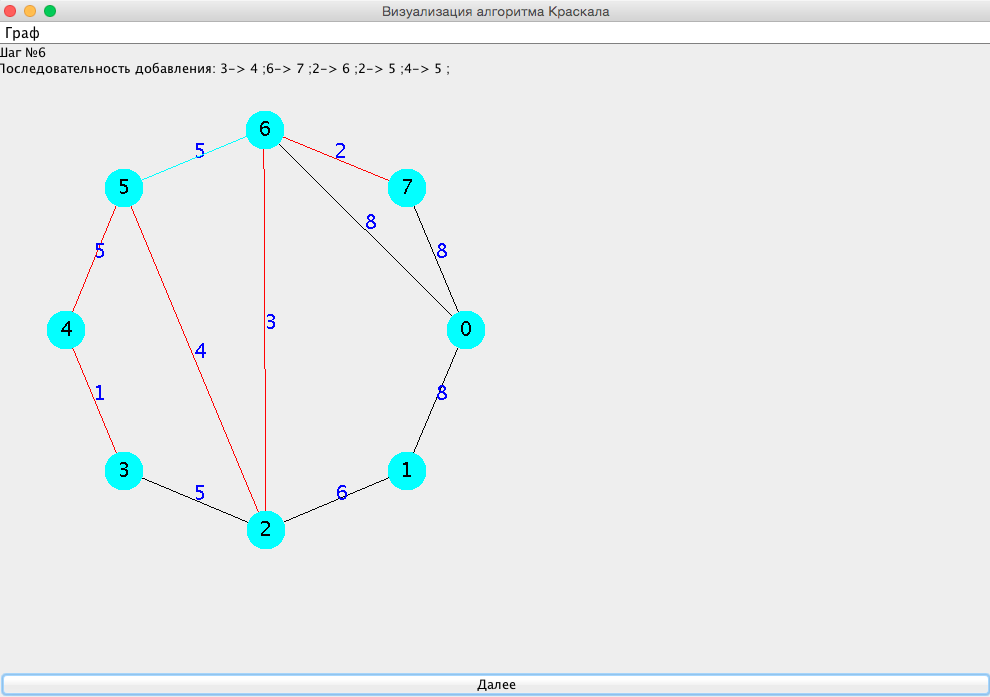
****

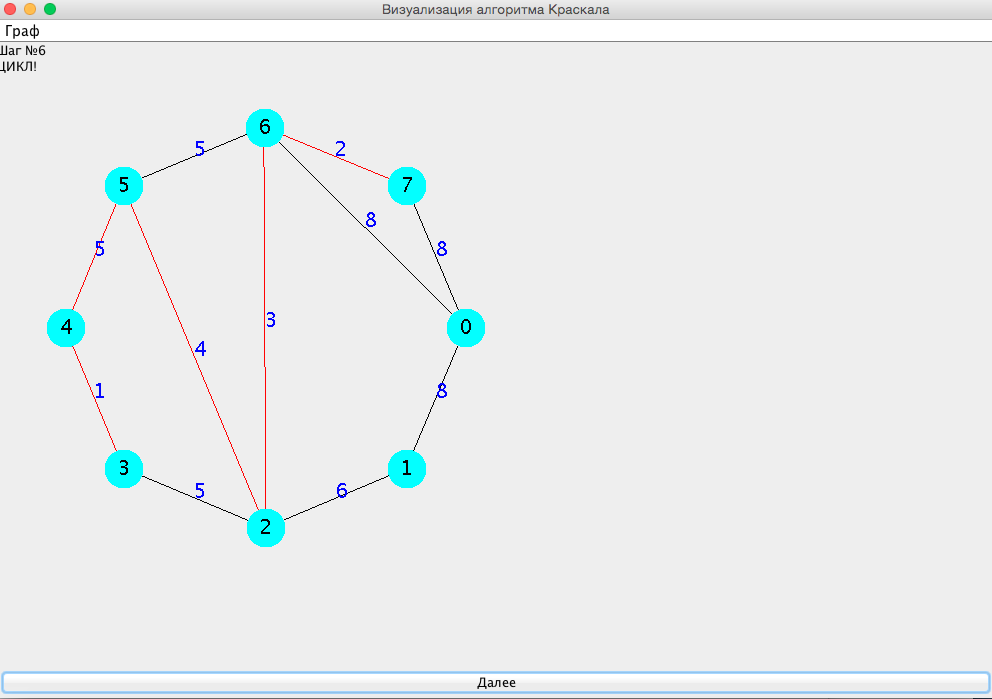
****

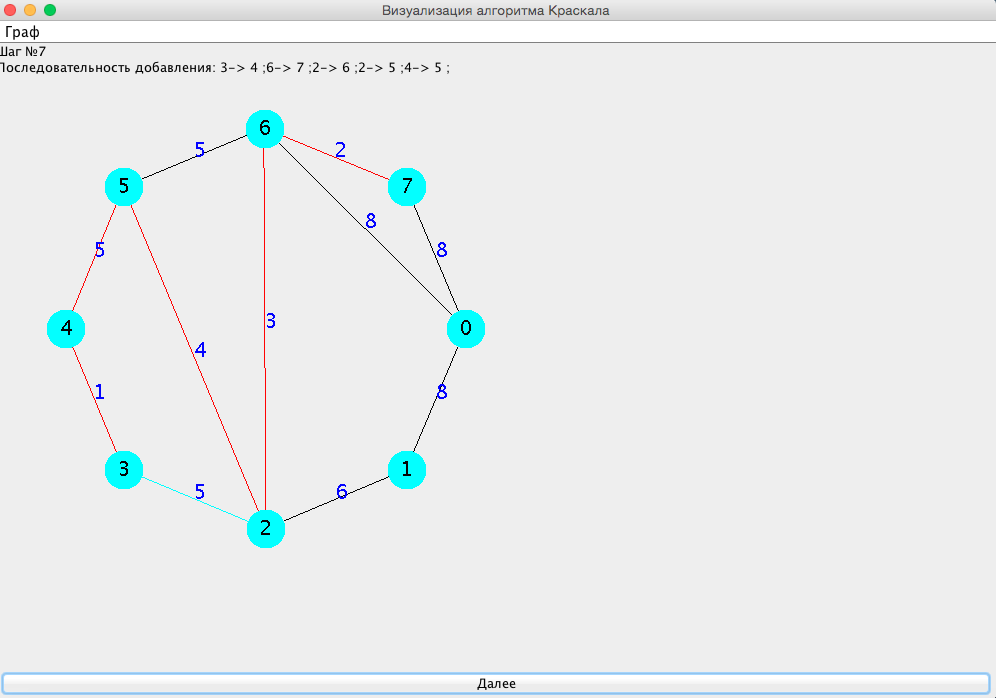
****

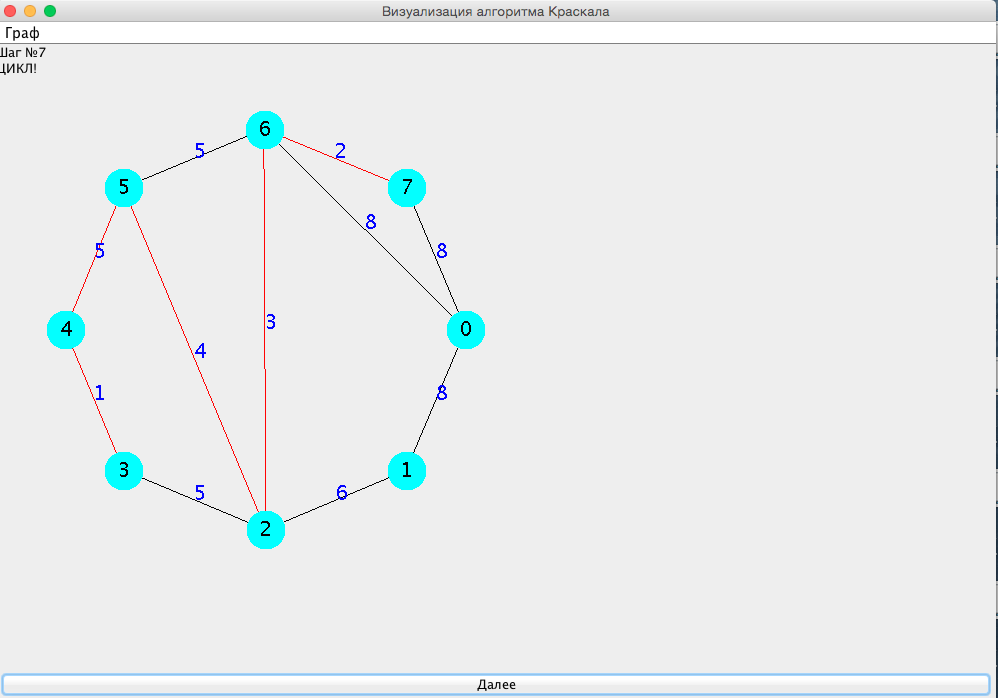
****

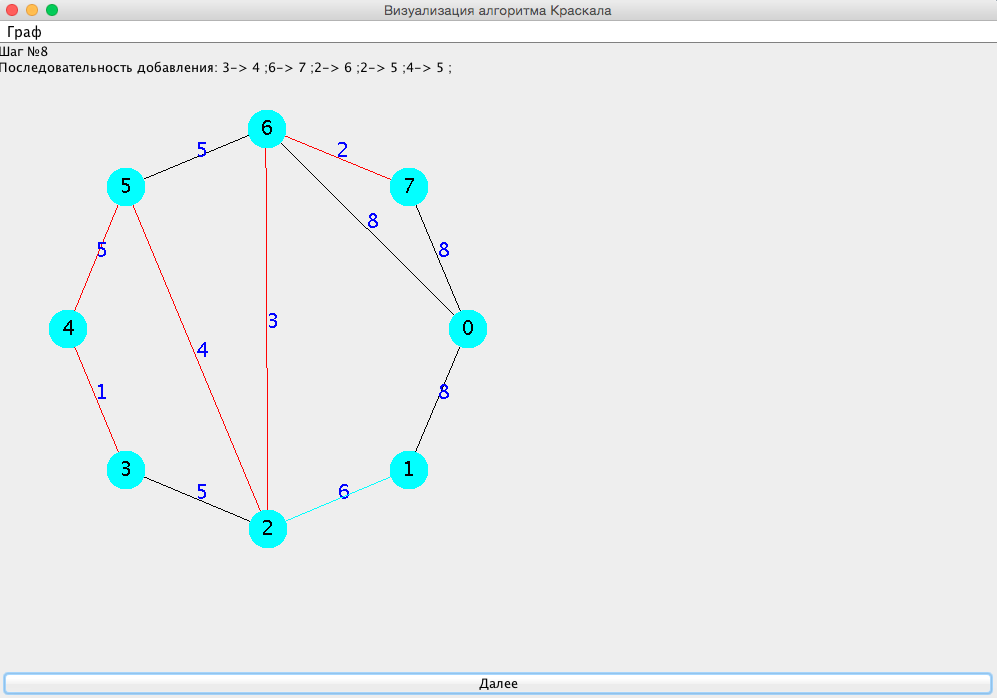
****

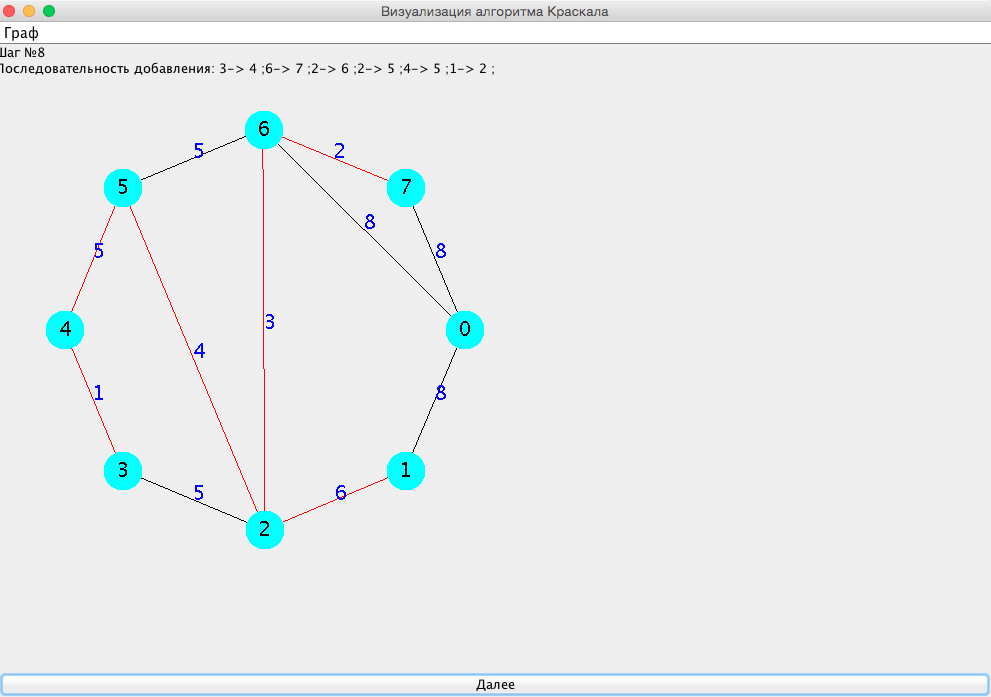
****

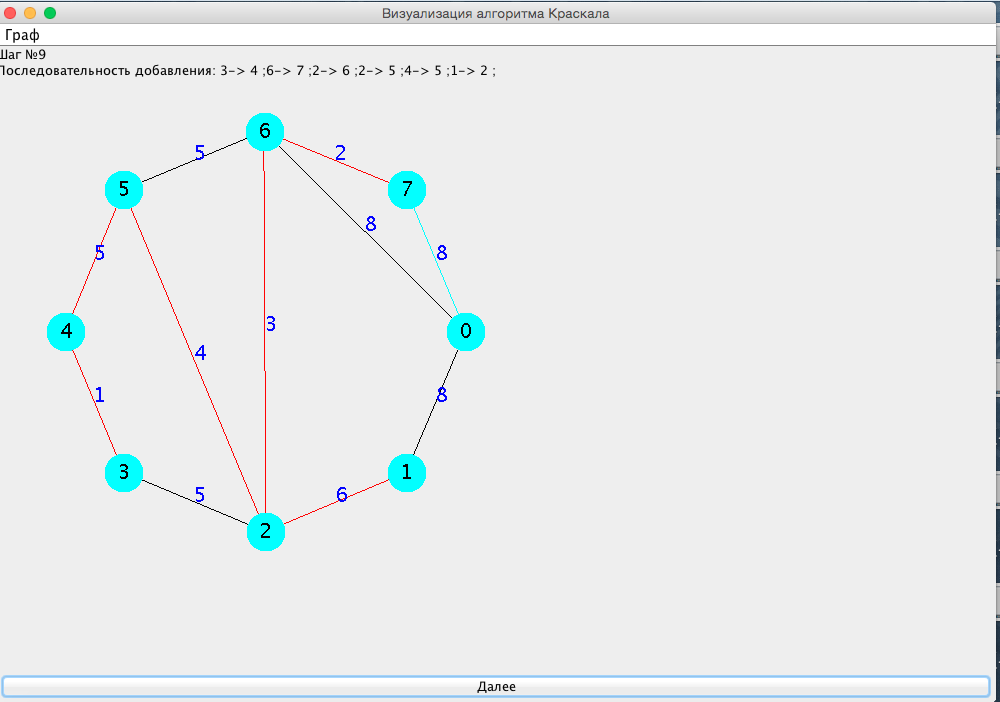
****

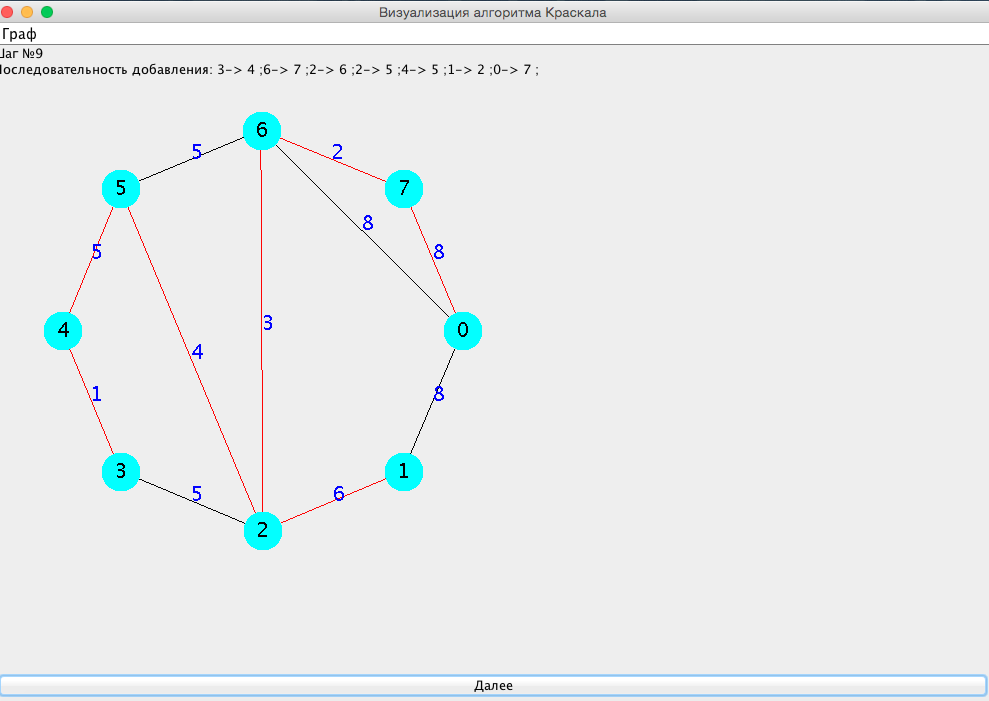
****

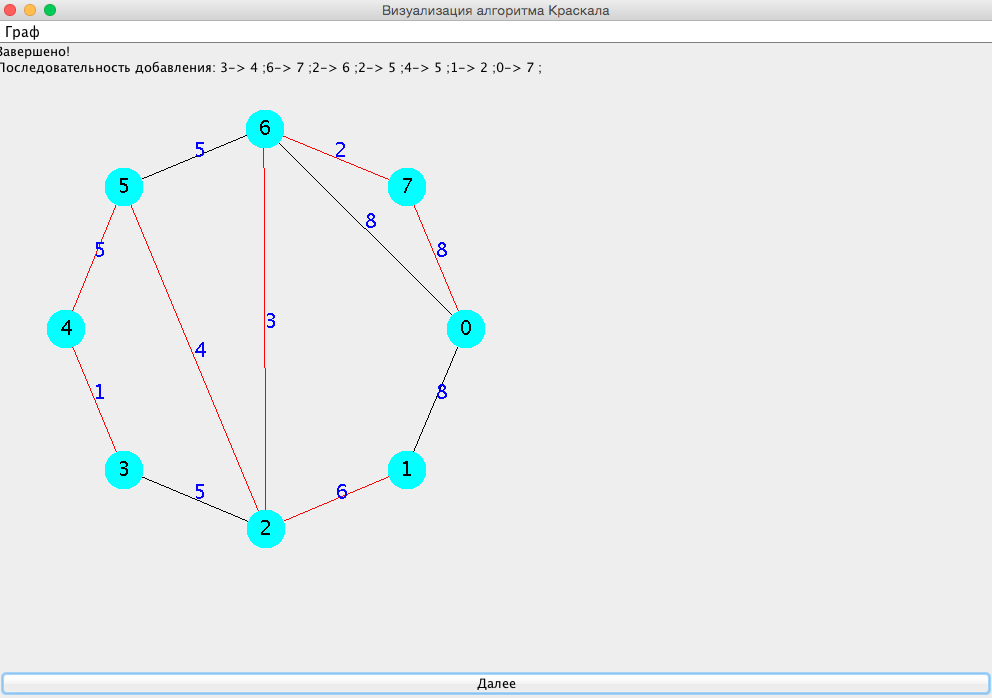
****

****

****

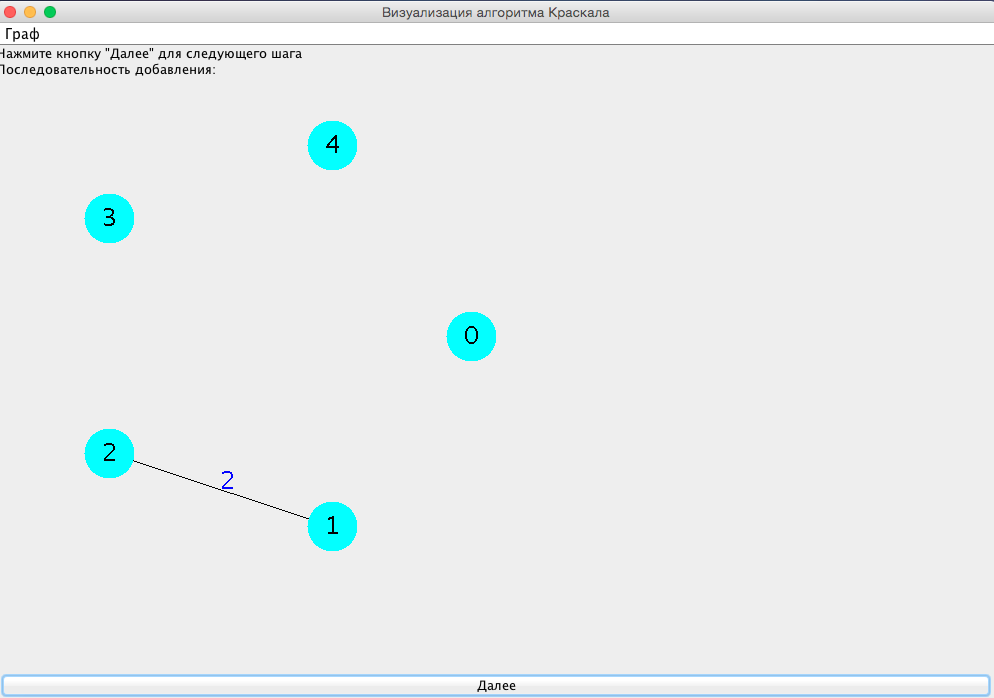
****

****

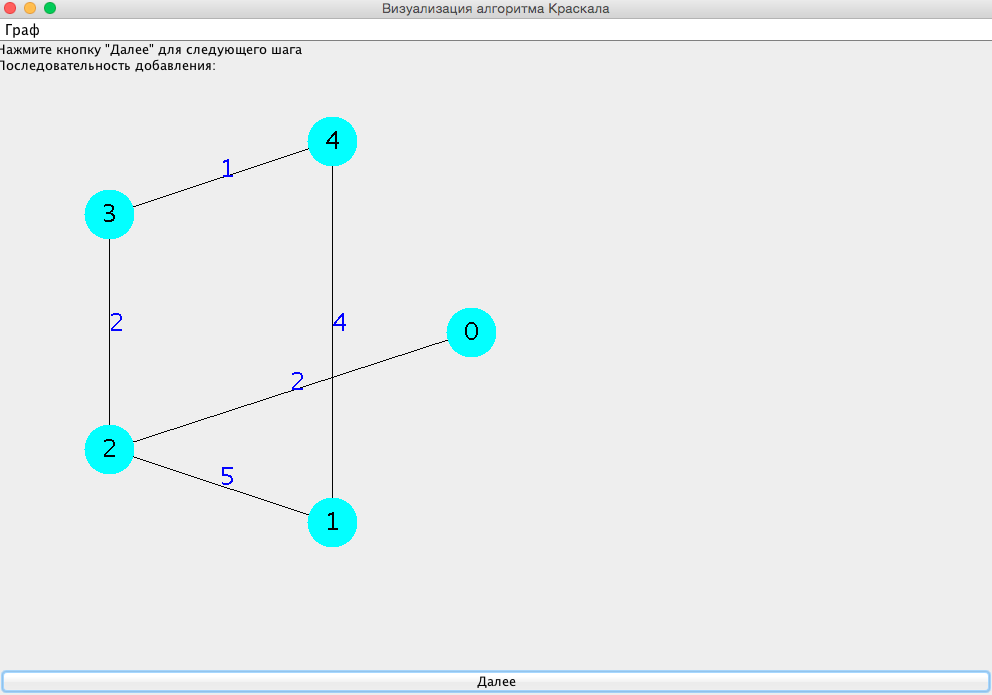
****

Как видим, визуализация алгоритма работает правильно.

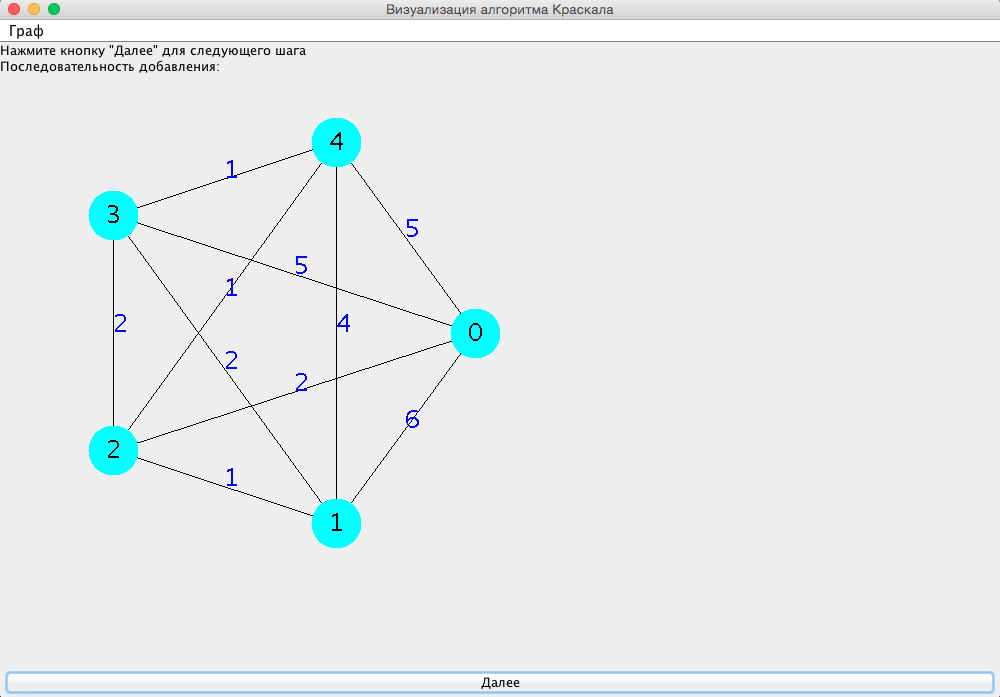
1. **Тестирование генерации**
2. Генерация графа из 5 вершин и 10 процентом ребер

****

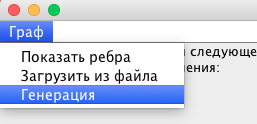
1. Генерация графа из 5 вершин и 50 процентов ребер



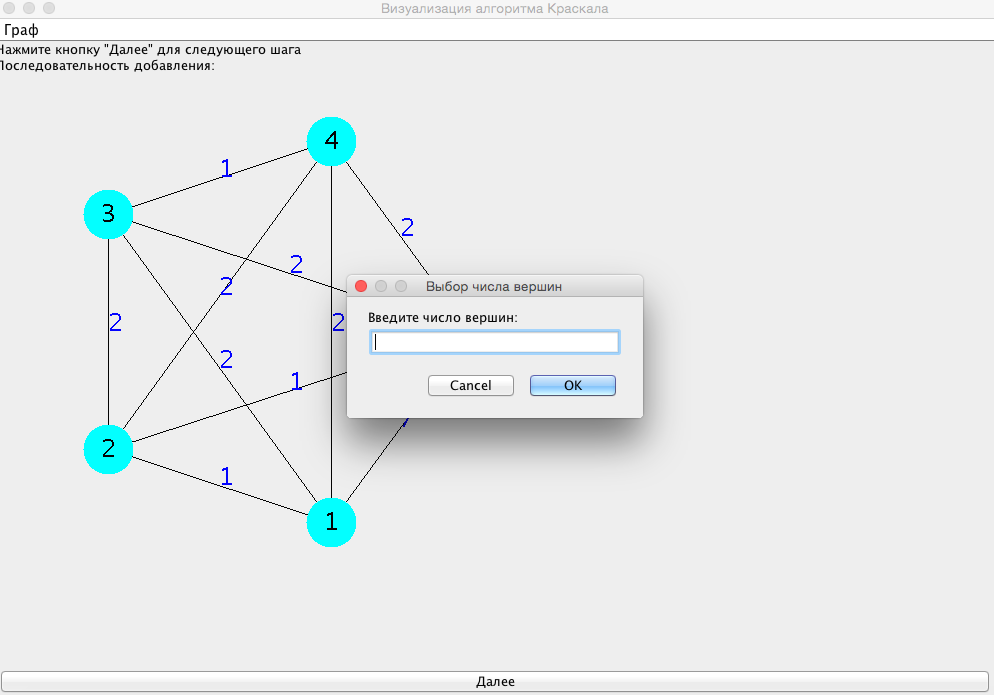
1. Генерация графа из 5 вершин и 100 процентов ребер



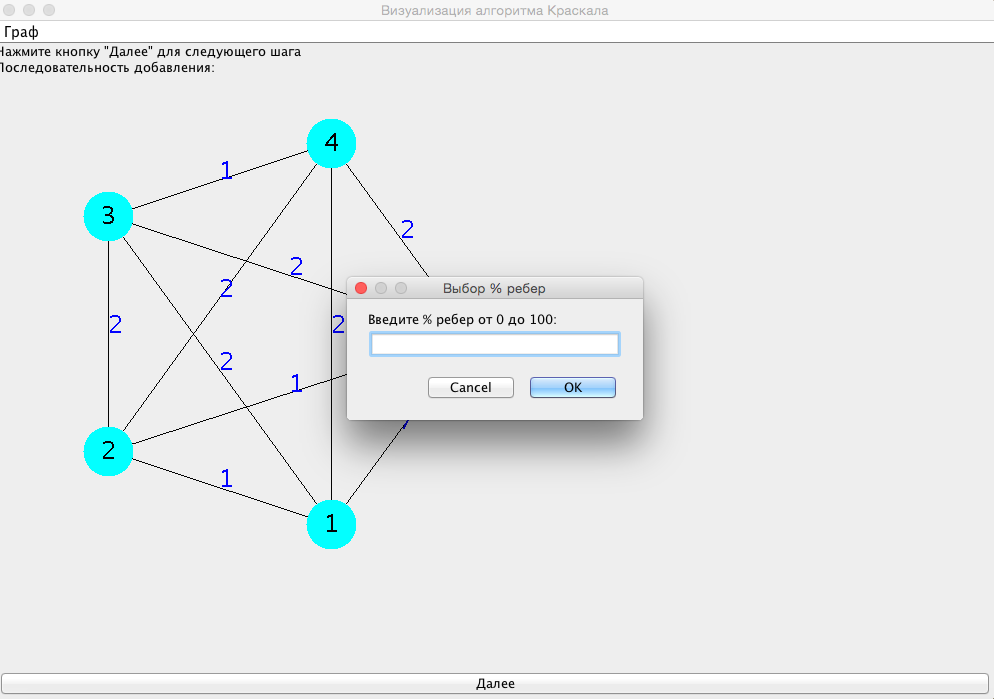
1. Кнопка генерации



1. Окно генерации «Выбор числа вершин»



1. Окно генерации «Выбор процента ребер»



1. Вывод:

Генерация происходит верно, так как в графе на 5 вершин максимальное число ребер 10, соответственно, каждые 10% ребер составляют одно ребро. Как видим, ровно нужное число ребер.

1. **Заключение**

Проект выполнен успешно. Учтены все требования.

1. **Приложение (код программы)**

Файл Graph.java

package practic;

import java.awt.BorderLayout;

import java.awt.Color;

import java.awt.Graphics;

import java.io.\*;

import java.util.Random;

import java.util.Scanner;

import java.util.concurrent.Semaphore;

import java.util.logging.Level;

import java.util.logging.Logger;

import javax.swing.\*;

class Vertex{

public int x, y; //координаты вершины

public int num;

public Vertex(int x, int y, int num){

this.x = x;

this.y = y;

this.num = num;

}

}

class Edge{

Vertex from;

Vertex to;

int weight;

Edge(Vertex from, Vertex to, int weight){

this.from = from;

this.to = to;

this.weight = weight;

}

}

public class Graph extends JPanel implements Runnable{

public int numV;

Semaphore sem;

public int numE;

public int numM;

int currentEdge;

JLabel doings;

public Thread graphThread;

public int r;

public Vertex v[];

public Edge e[];

public Edge MFT[];

public void run(){

doings.setText("");

repaint();

Kruskal();

}

protected void paintComponent(Graphics g){

super.paintComponent(g);

//настройка шрифта

g.setFont(g.getFont().deriveFont((float)r/2));

//отрисовка всех ребер графа

g.setColor(Color.BLACK);

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

g.drawLine(e[i].from.x+r/2, e[i].from.y+r/2,e[i].to.x+r/2,e[i].to.y+r/2);

}

//отрисовка ребер мод

g.setColor(Color.RED);

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

g.drawLine(MFT[i].from.x+r/2, MFT[i].from.y+r/2, MFT[i].to.x+r/2, MFT[i].to.y+r/2);

}

//отрисовка весов ребер

g.setColor(Color.BLUE);

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

String weight = Integer.toString(e[i].weight);

g.drawString(weight, (e[i].from.x + e[i].to.x +r)/2, (e[i].from.y + e[i].to.y +r)/2);

}

//отрисовка вершин

g.setColor(Color.CYAN);

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

g.fillOval(v[i].x,v[i].y, r, r);

}

if(currentEdge!=-1){

g.drawLine(e[currentEdge].from.x + r/2, e[currentEdge].from.y + r/2, e[currentEdge].to.x + r/2, e[currentEdge].to.y + r/2);

}

//отрисовка номеров вершин

g.setColor(Color.BLACK);

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

g.drawString(Integer.toString(i), v[i].x + 3\*r/8, v[i].y + 2\*r/3);

}

}

private void bothConstr(){

currentEdge = -1;

doings = new JLabel(""); //создание метки под вывод последовательности действий

graphThread = new Thread(this);

this.setLayout(new BorderLayout());

this.add(doings,BorderLayout.NORTH);

double angle = 6.2831853/numV; //вычиление расположения вершин

double nextangle = 0;

int R = 200;

r = (int)(6.28\*R)/(4\*numV);

if(r>50)r = 50;

if (r < 20){

//добавить скролл бар

r = 20;

}

int CentrX=250, CentrY=250;

int x, y;

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

x = (int) (CentrX + Math.cos(nextangle)\*R);

y = (int) (CentrY + Math.sin(nextangle)\*R);

nextangle += angle;

v[i] = new Vertex(x, y, i);

}

}

public Graph(int numvert, int proc, Semaphore sem){

//инициализация графа по заданым параметрам

int max = numvert\*(numvert-1)/2; //максимально возможное число верин в графе

int must = (int) (proc\*numvert\*(numvert-1)/100)/2; //необходимое нам число вершин

int counter = 0; //счетчик для массива вершин

numV = numvert; //инициализация полей графа

numE = must;

e = new Edge[must]; //создание массивов под вершины, ребра и мод

v = new Vertex[numvert];

MFT = new Edge[numV-1];

this.sem = sem;

bothConstr();

Random rand = new Random(System.currentTimeMillis()); //рандомные значения

Random weightrand = new Random(System.currentTimeMillis());

for(int i = 0; i < numvert && must > 0; i++){ //идем по вершинам, от которых можно провести ребро

for(int j = i+1; j < numvert && must > 0; j++){ //к которым можно его провести

if(rand.nextInt(max)< must){ //ставим ребро между i и j с вероятностью must/max

must--; //если ставим, уменьшаем оставшееся число ребер

e[counter] = new Edge(v[i],v[j],weightrand.nextInt(max)+1); //ставим ребро

counter++; //увеличиваем счетчик массива

}

max--; //уменьшаем максимально возможное оставшееся число ребер

}

}

graphThread.start();

}

public Graph(File file, Semaphore sem){

Scanner in;

try {

in = new Scanner(file);

this.sem = sem;

doings = new JLabel("");

numV = in.nextInt();

MFT = new Edge[numV-1];

v = new Vertex[numV];

numE = in.nextInt();

bothConstr();

e = new Edge[numE];

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

int from = in.nextInt();

int to = in.nextInt();

int weight = in.nextInt();

e[i] = new Edge(v[from], v[to], weight);

}

in.close();

graphThread.start();

} catch (FileNotFoundException ex) {

Logger.getLogger(Graph.class.getName()).log(Level.SEVERE, null, ex);

}

}

private void sortEdges(){

for(int i = 0; i <numE-1; i++){

for(int j = i+1; j<numE; j++){

if(e[i].weight >e[j].weight){

Edge help = e[i];

e[i]=e[j];

e[j]=help;

}

}

}

}

public void Kruskal(){

sortEdges();

String result = "Последовательность добавления: ";

doings.setText(result);

boolean unions[][] = new boolean[numV][numV];

int counter = 0;

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

unions[j][j] = true;

}

for(int i = 0; i<numE && (counter < numV-1); i++){

try {

sem.acquire();

} catch (InterruptedException e1) {

// TODO Auto-generated catch block

e1.printStackTrace();

}

currentEdge = i;

doings.setText(result);

this.repaint();

try {

graphThread.sleep(2000);

} catch (InterruptedException e1) {

// TODO Auto-generated catch block

e1.printStackTrace();

}

Edge help = e[i];

int from = e[i].from.num;

int to = e[i].to.num;

int h = -1;

int h2 = -1;

for(int j = 0; (j < from && h == -1); j++)

if(unions[from][j])

h=j;

for(int j = 0; (j<to &&h2==-1); j++)

if(unions[to][j])

h2=j;

if(h2!=-1)to = h2;

if(h!=-1)from = h;

if(to<from){

h = to;

to = from;

from = h;

}

boolean diffUnions = true;

for(int j = 0; j< numV && diffUnions; j++){

if((unions[j][from] == unions[j][to])&&(unions[j][from] == true)){

diffUnions = false;

}

}

if(diffUnions){

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

if(unions[j][to]){

unions[j][from] = true;

}

}

MFT[counter] = help;

counter++;

result+=help.from.num+"-> "+help.to.num+" ;";

doings.setText(result);

currentEdge = -1;

this.repaint();

numM = counter;

}

else{

doings.setText("ЦИКЛ!");

currentEdge = -1;

this.repaint();

}

}

}

}

Файл Demo.java

package practic;

import java.awt.\*;

import java.util.\*;

import java.util.concurrent.Semaphore;

import java.awt.event.\*;

import java.io.File;

import javax.swing.\*;

public class Demo {

Thread mainThread;

JLabel step;

static int count;

Semaphore sem;

JFrame mainWindow;

JButton next;

Graph gr;

Demo(){

mainWindow = new JFrame("Визуализация алгоритма Краскала"); //создается контейнер, содержащий окно

mainWindow.setSize(1000, 700); //задается его размер

mainWindow.setLayout(new BorderLayout()); //расположение объектов на нем

mainWindow.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE); //чтобы закрывалася на

mainWindow.setVisible(true); //становится видимым

mainThread = Thread.currentThread();

sem = new Semaphore(1); //создается семафор на 1 разрешение

try {

sem.acquire();

} catch (InterruptedException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

gr = new Graph(5,100,sem); //создается граф

mainWindow.add(gr, BorderLayout.CENTER); //добавляется на окно

step = new JLabel("Нажмите кнопку \"Далее\" для следующего шага"); //метка, отображающая номер шага

next = new JButton("Далее"); //оздается кнопка

next.addActionListener(new ActionListener(){ //задается действие при нажатии на кнопку

public void actionPerformed(ActionEvent ae){

if(gr.graphThread.isAlive()){

count++; //увеличивается счетчик шагов

step.setText("Шаг №"+Integer.toString(count)); //записывается номер шага

sem.release();

}

else {

step.setText("Завершено!");

}

}

});

JMenuBar menubar = new JMenuBar();

JMenu menu = new JMenu("Граф");

JMenuItem load = new JMenuItem("Загрузить из файла");

JMenuItem generate = new JMenuItem("Генерация");

JMenuItem show = new JMenuItem("Показать ребра");

menu.add(show);

//pp.setLayout( new BorderLayout());

menu.add(load);

menu.add(generate);

menubar.add(menu);

mainWindow.setJMenuBar(menubar);

show.addActionListener(new ActionListener(){

public void actionPerformed(ActionEvent AE){

JFrame edges = new JFrame("Ребра графа");

edges.setSize(500, 500); //задается его размер

edges.setLayout(new BorderLayout()); //расположение объектов на нем

edges.setDefaultCloseOperation(JFrame.HIDE\_ON\_CLOSE); //чтобы закрывалася на

edges.setVisible(true);

JTextArea list = new JTextArea();

//JLabel list = new JLabel();

String res = new String();

for(int i = 0; i < gr.numE; i++){

res+=" ИЗ "+gr.e[i].from.num+" В "+gr.e[i].to.num+" весом "+gr.e[i].weight+"\n";

}

list.setText(res);

list.setEditable(false);

//list.setText(res);

edges.add(list);

JScrollPane scroll = new JScrollPane(list);

edges.add(scroll);

}

});

load.addActionListener(new ActionListener(){

public void actionPerformed(ActionEvent ae){

FileDialog fileDialog = new FileDialog(mainWindow, "Выбор файла", FileDialog.LOAD);

fileDialog.show();

if(fileDialog.getFile()!=null){

count = 0;

step.setText("Нажмите кнопку \"Далее\" для следующего шага");

File file = new File(fileDialog.getDirectory()+ fileDialog.getFile());

gr.graphThread.stop();

sem = new Semaphore(1);

try {

sem.acquire();

} catch (InterruptedException e) {}

mainWindow.remove(gr);

gr = new Graph(file, sem);

mainWindow.add(gr);

}

}

});

generate.addActionListener(new ActionListener(){

public void actionPerformed(ActionEvent ae){

String numv =JOptionPane.showInputDialog(mainWindow,"Введите число вершин:","Выбор числа вершин", JOptionPane.PLAIN\_MESSAGE);

String proc = JOptionPane.showInputDialog(mainWindow,"Введите % ребер от 0 до 100:","Выбор % ребер", JOptionPane.PLAIN\_MESSAGE);

if(numv!=null &&proc != null ){

count = 0;

step.setText("Нажмите кнопку \"Далее\" для следующего шага");

gr.graphThread.stop();

sem = new Semaphore(1);

try {

sem.acquire();

} catch (InterruptedException e) {}

mainWindow.remove(gr);

gr = new Graph(Integer.parseInt(numv),Integer.parseInt(proc), sem);

mainWindow.add(gr);

}

}

});

mainWindow.add(next, BorderLayout.SOUTH);

mainWindow.add(step, BorderLayout.NORTH); //добавляем ее в контейнер

}

public static void main(String[] args) {

// TODO Auto-generated method stub

SwingUtilities.invokeLater(new Runnable(){

public void run(){

new Demo();

}

});

}

}