Министерство образования и науки Российской федерации

Санкт-Петербургский государственный электротехнический университет

“ЛЭТИ”

кафедра математического обеспечения и применения ЭВМ

**Спецификация**

**Преподаватель: Фирсов М.А.**

Факультет: КТИ

Выполнили:

Марущак Ю.В.

Левицкий Д.В.

Карпенко Д.Р.

Санкт-Петербург

2016

**Список терминов:**

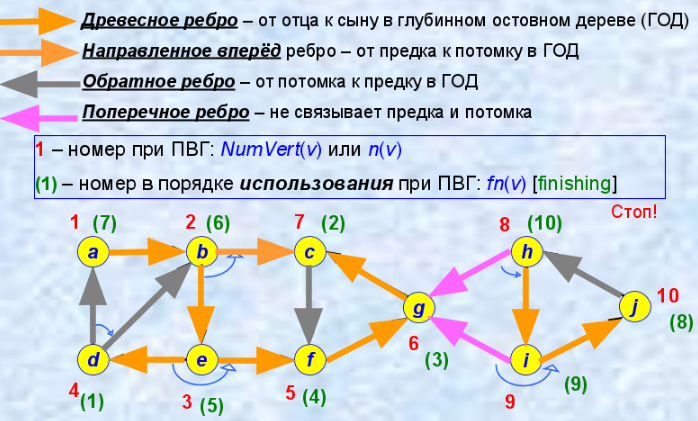
**Граф** — основной объект изучения математической теории графов, совокупность непустого множества вершин и наборов пар вершин (связей между вершинами).

**Ориентированный граф** (кратко **орграф**) — (мульти) граф, рёбрам которого присвоено направление.

**Направленный ациклический граф**(*ориентированный ациклический граф*) — орграф, в котором отсутствуют направленные циклы, то есть пути, начинающиеся и кончающиеся в одной и той же вершине. Направленный ациклический граф является обобщением дерева.

**Топологическая сортировка** — упорядочивание вершин направленного ациклического графа согласно частичному порядку, заданному ребрами орграфа на множестве его вершин.

**Виды ребер в орг. Графе**:



**Цель:**

Реализовать алгоритм Топологической сортировки на базе языка Java и выполнить топологическую сортировку для заданного направленного ациклического графа и визуализировать его работу и интерфейс, для удобства использования программы.

**Алгоритм Тарьяна:**

Изначально все вершины «белые». Производим серию обходов в глубину, пока каждая вершина не будет посещена.

* При входе в вершину делаем её «серой», при выходе — «чёрной» и одновременно заносим в окончательный список.
* Если вдруг вошли в серую вершину — найден цикл, топологическая сортировка невозможна.

**Цвет.** Во время обхода в глубину используется 3 цвета. Изначально все вершины белые. Когда вершина обнаружена, красим ее в серый цвет. Когда просмотрен список всех смежных с ней вершин, красим ее в черный цвет.

**Сложность.** O(n)

**Наш псевдокод:**

void main()

{

Graph G;

вврод графа G;

Node \* final\_list;

для каждой вершины графа G вызываем : algorithm(curr\_top, final\_list)

}

void algorithm(Node curr\_top, Node \* final\_list)

{

if (curr.color == Gray)

{

Невозможно отсортирвоать, т.к.найден цикл.

}

curr\_top.color = Gray;

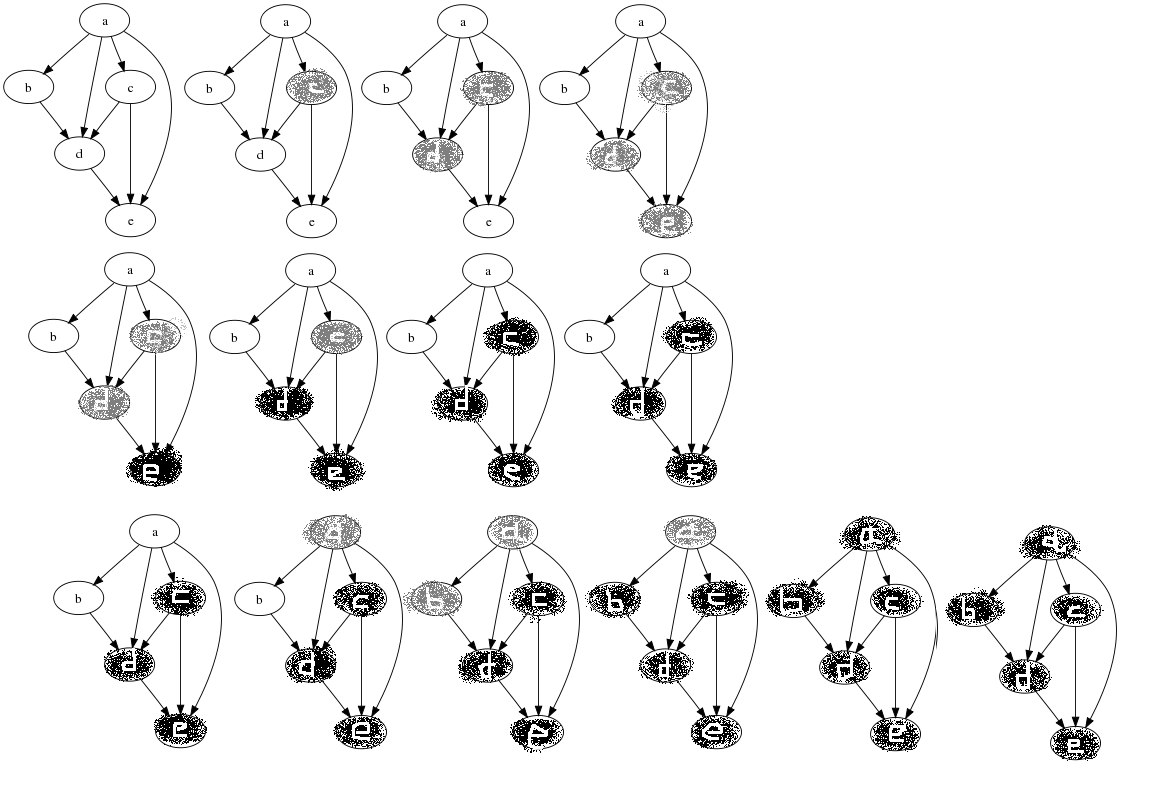
для каждого сына curr\_top вызываем функцию algorithm(текущий сын curr\_top, final\_list)

curr\_top.color = Black;

добавляем curr\_top в final\_list;

}

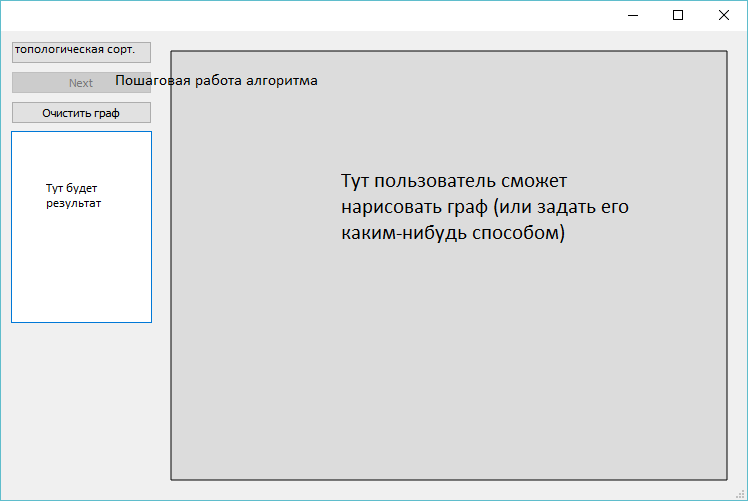
**Наш пример:**



a,b,c,d,e|a,c,b,d,e

**Примерный вид интерфейса:**

Поле для вывода работы алгоритма



Пользователь будет иметь возможность вводить граф графически(например: Левая кнопка – создание вершины, правая кнопка – провести ребро), и будет иметь возможность удалять и перемещать вершины. Так же вводить граф с файла вручную или сгенерировать граф

В программе будет возможность выделить вершину кликнув по ней левой кнопкой, таким образом чтобы провести ребро, нужно выделить 2 вершины и затем кликнуть правой кнопкой мыши по пустому месту ( в том порядке, в которым были выбраны вершины и будет проведено ребро)

Удаление будет сделано аналогично, выделяются вершины и какой-нибудь горячей клавишей они будут удалены.

С файла будет в виде:

{1,2,3,4}(1,2)(1,4){список всех вершин}(связи)

Результат: отсортированный список вершин

Генерация(начальная версия): Будет происходить случайно(но будет проверка на цикличность) если граф будет с циклом генерация будет происходить заново(входные параметры кол-во вершин и кол-во ребер).

**План работы:**

**25.06.2016** Встреча группы для обсуждения алгоритмов, создания репозитория и четкого разделения труда.

**26.06.2016** Создание Прототипа(отдельно работающий графический интерфейс и алгоритм).

**28.06.2016** Попытка соединения между собой Алгоритма и граф. интерфейса. Необходимо, чтобы присутствовал хотя бы примитивный способ задания графа (с граф. выводом на экран) и вывод результата в интерфейс(без демонстрации работы алгоритма).

**30.06.2016** Графическая демонстрация работы алгоритма, с реализованным алгоритмом выделения различных видов ребер, и несколькими способами задания графа пользователем(в том числе с случайной генерацией).

**3.07.2016** Написание отчета по финальной версии проекта.

Изменение: 3.07.2016 завершить финальную версию( ввод из файла , случайная генерация, выделение графически различных видов ребер)

Дополнение : к 30.06.2016 доделать проверку на циклы и выдачу соответствующих сообщений

**Разделение труда:**

**Группа :** Обсуждение выбора алгоритма и тонкостей его реализации

Юлия: Разработка алгоритма на Java ( топологическая сортировка и разделение различных видов ребер)

Project manage (слежение за планом, создание репозитория)

Даниил: Создание интерфейса и графического способа задания графа, визуализация различных видов ребер

Денис: Графическая визуализация алгоритма и создание ввода графа из файла и генерации графа

**Использованные структуры данных:**

В нашей программе для хранения графа использовались следующие классы:

**class** Vertex*// Класс вершина*{  
 **int x**, **y**;//Координаты расположения на поле  
 **char color**;*//Цвет ( 'g' - серый , 'b' - черный, 'w' - белый )* **int name**; *// номер вершины* ArrayList<Edge> **edg**= **new** ArrayList<Edge>(); *//Список ребер из вершины* Vertex(**int** x,**int** y,**int** name) *// конструктор* {  
 **this**.**x**=x;  
 **this**.**y**=y;  
 **this**.**color**= **'w'**; *//Вначале у всех цвет - белый* **this**.**name**=name; *// Записали номер вершины* }  
};

Для хранения ребер идущих от вершины используется вектор, т.к. мы часто получаем доступ ко всем элементам(пробегаем весь вектор).

**class** Edge *// Класс ребро*{  
 Vertex **v**; *// В какую вершину оно идет* **char type**; *// 'n'- не использованное , 'd' - древесное , 'u' - направленное вперед , 'b' - направленное назад , 'p' - поперечное* **boolean used**; *// Ходили ли мы через него* Edge(Vertex v) *// Конструктор* {  
 **this**.**type**=**'n'**;  
 **this**.**v** = v;  
 **this**.**used**=**false**;  
 }  
};

Used = true - если прошли по вершине , false - не прошли

**class** Tree *//Класс деерво*{  
 **static char** *White*=**'w'**; *//Введем константные цвета* **static char** *Black*=**'b'**;  
 **static char** *Gray*=**'g'**;  
 **static int** *MAX\_VERTICES*=1000;  
 ArrayList<Vertex> **vertices**= **new** ArrayList<Vertex>(); *// Список вершин* ArrayList<Vertex> **RTopSort**= **new** ArrayList<Vertex>(); *//Нужный список вершин, записанный в обратном порядке* **static** Random *random\_integer* = **new** Random();  
 **void** removeVertex(Vertex v)  
 {  
 **for** (**int** i = 0; i < **vertices**.size(); i++)  
 {  
 **for**(**int** j=0;j<**vertices**.get(i).**edg**.size();j++)  
 {  
 **if**(**vertices**.get(i).**edg**.get(j).**v**==v)  
 {  
 **vertices**.get(i).**edg**.remove(j);  
 j--;  
 }  
 }  
 }  
 v.**edg**.clear();  
 **int** i;  
 **for**(i=0;i<**vertices**.size();i++)  
 **if**(**vertices**.get(i)==v) **break**;  
 **vertices**.remove(i);  
 }  
 **public boolean** add\_edge(**int** from, **int** to)  
 {  
 Vertex vertex\_from = **null**;  
 **for** (**int** k = 0; k < **vertices**.size(); k++)  
 **if** (**vertices**.get(k).**name** == from) {  
 vertex\_from = **vertices**.get(k);  
 }  
 **if**(vertex\_from == **null**) *// если в дереве нет вершины, с которой хотят соединить путь* {  
 **return false**;  
 }  
 Vertex vertex\_to = **null**;  
 **for** (**int** k = 0; k <**vertices**.size(); k++)  
 **if** (**vertices**.get(k).**name** == to) {  
 vertex\_to = **vertices**.get(k);  
 }  
 **if**(vertex\_to == **null**) *// если в дереве нет вершины, с которой хотят соединить путь* {  
 **return false**;  
 }  
  
 vertex\_from.**edg**.add(**new** Edge(vertex\_to));  
 **return true**;  
 }  
 **void** addVertex(**int** x, **int** y) {  
 **int** name = 1;  
 **while** (name <= *MAX\_VERTICES*) {  
 **boolean** contain = **false**;  
 **for** (**int** j = 0; j < **vertices**.size(); j++)  
 **if** (**vertices**.get(j).**name** == name) contain = **true**;  
 **if** (!contain) **break**;  
 name++;  
 }  
 **vertices**.add(**new** Vertex(x, y, name));  
 }  
 **void** addVertex(**int** name) {  
 *// int name = 1;* **for** (**int** j = 0; j < **vertices**.size(); j++)  
 **if** (**vertices**.get(j).**name** == name) **return**; *// такая вершина уже содержится* **int** x = 50 + (abs(*random\_integer*.nextInt()) % 330) ;  
 **int** y = 50 + (abs(*random\_integer*.nextInt()) % 330) ;  
 **vertices**.add(**new** Vertex(x, y, name));  
  
 }  
 **void** addVertex( **int** x, **int** y, **int** name) {  
 *// int name = 1;* **for** (**int** j = 0; j < **vertices**.size(); j++)  
 **if** (**vertices**.get(j).**name** == name) **return**; *// такая вершина уже содержится* **vertices**.add(**new** Vertex(x, y, name));  
  
 }  
 **private int** abs(**int** g) {  
 **if**(g>0) **return** g; **else return** -g;  
 }  
 **public boolean** have\_cycle()  
 {  
 *// if(RTopSort.size()>0) RTopSort.clear(); //Очистим список вершин  
 // RTopSort.clear(); //Очистим список вершин* **for** (**int** i=0; i<**vertices**.size(); i++)  
 {  
 **if**((**vertices**.get(i).**color**==*White*)) *//Если вершина не пройдена* {  
 **if**(!dfs1(**vertices**.get(i))) {  
 **for** (**int** j = 0; j<**vertices**.size(); j++)  
 {  
 **vertices**.get(j).**color**=*White*;  
 }  
 **return true**; *// запускаем поиск в глубину* }  
 *// if(!Ch) return true;* }  
 }  
 **for** (**int** j = 0; j<**vertices**.size(); j++)  
 {  
 **vertices**.get(j).**color**=*White*;  
 }  
 **return false**;  
 }  
 **public boolean** dfs1(Vertex cur) *//Поиск в глубину* {  
 **if**(cur.**color** == *Gray*) **return false**;  
 cur.**color**=*Gray*; *//Красим вершину в серый* **for** (**int** i = 0; i<cur.**edg**.size(); i++)  
 {  
 **if** (!(cur.**edg**.get(i).**v**.**color**==*Black*))  
 {  
 **if**(!dfs1(cur.**edg**.get(i).**v**)) **return false**;*//переходим к след. вершине* }  
 }  
 cur.**color**=*Black*;  
 **return true**;  
 }  
 **void** dellete\_edge(**int** from, **int** to)  
 {  
 Vertex curr\_vert= **null**;  
 **for** (**int** j = 0; j < **vertices**.size(); j++)  
 **if** (**vertices**.get(j).**name** == from) {  
 curr\_vert = **vertices**.get(j);  
 **break**;  
 }  
 **if**(curr\_vert!=**null**)  
 {  
 Vertex to\_vert=**null**;  
 **for** (**int** j = 0; j < **vertices**.size(); j++)  
 **if** (**vertices**.get(j).**name** == from) {  
 curr\_vert = **vertices**.get(j);  
 **break**;  
 }  
 **if**(to\_vert == **null**)  
 **return**;  
 curr\_vert.**edg**.remove(to\_vert);  
 }  
  
  
 *// vertices.add(new Vertex(x, y, name));* }  
  
};

В дереве используется вектор вершин, по аналогичной причине, что и выше.

void removeVertex (Vertex v) - удаляет вершину v из дерева

void addVertex(x,y) - добавляет вершину в дерево с координатами x,y

boolean have\_cycle() - проверяет наличие цикла в дереве

boolean dfs1(Vertex cur) - Поиск в глубину для поисков циклов

void dellete\_edge(int from, int to) - Удалить ребро из from в to

В MainClass:

**public void** paintEvent(QPaintEvent e) – Перерисовка в случае repaint()

**public void** mousePressEvent(QMouseEvent e)- Нажатие кнопки мыши

**public void** mouseMoveEvent(QMouseEvent e)- передвижение элементов мышью

**public void** mouseReleaseEvent(QMouseEvent e)-Остановка мыши на месте

**public void** keyPressEvent(QKeyEvent e)- нажатие клавиши

**public void** resizeEvent(QResizeEvent e)- Изменение размеров окна

**void** Wait()- ожидание всех процессов

**public void** dfs1(Vertex cur)- обход в глубину

**void** TopSort()- Топологическая сортировка

**protected void** closeEvent( QCloseEvent e ) – закрытие окна

**Код MainClass:**

**import** com.trolltech.qt.core.QCoreApplication;  
**import** com.trolltech.qt.core.QPointF;  
**import** com.trolltech.qt.core.QRect;  
**import** com.trolltech.qt.core.Qt;  
**import** com.trolltech.qt.gui.\*;  
**import** java.util.ArrayList;  
**import** java.util.Random;  
  
**class** Vertex*// Класс вершина*{  
 **int x**, **y**;  
 **char color**;*//Цвет ( 'g' - серый , 'b' - черный, 'w' - белый )* **int name**; *// номер вершины* ArrayList<Edge> **edg**= **new** ArrayList<Edge>(); *//Список ребер из вершины* Vertex(**int** x,**int** y,**int** name) *// конструктор* {  
 **this**.**x**=x;  
 **this**.**y**=y;  
 **this**.**color**= **'w'**; *//Вначале у всех цвет - белый* **this**.**name**=name; *// Записали номер вершины* }  
};  
  
**class** Edge *// Класс ребро*{  
 Vertex **v**; *// В какую вершину оно идет* **char type**; *// 'n'- не использованное , 'd' - древесное , 'u' - направленное вперед , 'b' - направленное назад , 'p' - поперечное* **boolean used**; *// Ходили ли мы через него* Edge(Vertex v) *// Конструктор* {  
 **this**.**type**=**'n'**;  
 **this**.**v** = v;  
 **this**.**used**=**false**;  
 }  
};  
  
**class** Tree *//Класс деерво*{  
 **static char** *White*=**'w'**; *//Введем константные цвета* **static char** *Black*=**'b'**;  
 **static char** *Gray*=**'g'**;  
 **static int** *MAX\_VERTICES*=1000;  
 ArrayList<Vertex> **vertices**= **new** ArrayList<Vertex>(); *// Список вершин* ArrayList<Vertex> **RTopSort**= **new** ArrayList<Vertex>(); *//Нужный список вершин, записанный в обратном порядке* **static** Random *random\_integer* = **new** Random();  
 **void** removeVertex(Vertex v)  
 {  
 **for** (**int** i = 0; i < **vertices**.size(); i++)  
 {  
 **for**(**int** j=0;j<**vertices**.get(i).**edg**.size();j++)  
 {  
 **if**(**vertices**.get(i).**edg**.get(j).**v**==v)  
 {  
 **vertices**.get(i).**edg**.remove(j);  
 j--;  
 }  
 }  
 }  
 v.**edg**.clear();  
 **int** i;  
 **for**(i=0;i<**vertices**.size();i++)  
 **if**(**vertices**.get(i)==v) **break**;  
 **vertices**.remove(i);  
 }  
 **public boolean** add\_edge(**int** from, **int** to)  
 {  
 Vertex vertex\_from = **null**;  
 **for** (**int** k = 0; k < **vertices**.size(); k++)  
 **if** (**vertices**.get(k).**name** == from) {  
 vertex\_from = **vertices**.get(k);  
 }  
 **if**(vertex\_from == **null**) *// если в дереве нет вершины, с которой хотят соединить путь* {  
 **return false**;  
 }  
 Vertex vertex\_to = **null**;  
 **for** (**int** k = 0; k <**vertices**.size(); k++)  
 **if** (**vertices**.get(k).**name** == to) {  
 vertex\_to = **vertices**.get(k);  
 }  
 **if**(vertex\_to == **null**) *// если в дереве нет вершины, с которой хотят соединить путь* {  
 **return false**;  
 }  
  
 vertex\_from.**edg**.add(**new** Edge(vertex\_to));  
 **return true**;  
 }  
 **void** addVertex(**int** x, **int** y) {  
 **int** name = 1;  
 **while** (name <= *MAX\_VERTICES*) {  
 **boolean** contain = **false**;  
 **for** (**int** j = 0; j < **vertices**.size(); j++)  
 **if** (**vertices**.get(j).**name** == name) contain = **true**;  
 **if** (!contain) **break**;  
 name++;  
 }  
 **vertices**.add(**new** Vertex(x, y, name));  
 }  
 **void** addVertex(**int** name) {  
 *// int name = 1;* **for** (**int** j = 0; j < **vertices**.size(); j++)  
 **if** (**vertices**.get(j).**name** == name) **return**; *// такая вершина уже содержится* **int** x = 50 + (abs(*random\_integer*.nextInt()) % 330) ;  
 **int** y = 50 + (abs(*random\_integer*.nextInt()) % 330) ;  
 **vertices**.add(**new** Vertex(x, y, name));  
  
 }  
 **void** addVertex( **int** x, **int** y, **int** name) {  
 *// int name = 1;* **for** (**int** j = 0; j < **vertices**.size(); j++)  
 **if** (**vertices**.get(j).**name** == name) **return**; *// такая вершина уже содержится* **vertices**.add(**new** Vertex(x, y, name));  
  
 }  
 **private int** abs(**int** g) {  
 **if**(g>0) **return** g; **else return** -g;  
 }  
 **public boolean** have\_cycle()  
 {  
 *// if(RTopSort.size()>0) RTopSort.clear(); //Очистим список вершин  
 // RTopSort.clear(); //Очистим список вершин* **for** (**int** i=0; i<**vertices**.size(); i++)  
 {  
 **if**((**vertices**.get(i).**color**==*White*)) *//Если вершина не пройдена* {  
 **if**(!dfs1(**vertices**.get(i))) {  
 **for** (**int** j = 0; j<**vertices**.size(); j++)  
 {  
 **vertices**.get(j).**color**=*White*;  
 }  
 **return true**; *// запускаем поиск в глубину* }  
 *// if(!Ch) return true;* }  
 }  
 **for** (**int** j = 0; j<**vertices**.size(); j++)  
 {  
 **vertices**.get(j).**color**=*White*;  
 }  
 **return false**;  
 }  
 **public boolean** dfs1(Vertex cur) *//Поиск в глубину* {  
 **if**(cur.**color** == *Gray*) **return false**;  
 cur.**color**=*Gray*; *//Красим вершину в серый* **for** (**int** i = 0; i<cur.**edg**.size(); i++)  
 {  
 **if** (!(cur.**edg**.get(i).**v**.**color**==*Black*))  
 {  
 **if**(!dfs1(cur.**edg**.get(i).**v**)) **return false**;*//переходим к след. вершине* }  
 }  
 cur.**color**=*Black*;  
 **return true**;  
 }  
 **void** dellete\_edge(**int** from, **int** to)  
 {  
  
 *// System.out.println("Попытка удалить " + from + " " +to);* Vertex curr\_vert= **null**;  
 **for** (**int** j = 0; j < **vertices**.size(); j++)  
 **if** (**vertices**.get(j).**name** == from) {  
 curr\_vert = **vertices**.get(j);  
 **break**;  
 }  
 **if**(curr\_vert!=**null**)  
 {  
 Vertex to\_vert=**null**;  
 **int** index=0;  
 **for** (**int** j = 0; j < curr\_vert.**edg**.size(); j++) {  
 **if** (curr\_vert.**edg**.get(j).**v**.**name** == to) {  
 *// System.out.println("Сравниваем " + vertices.get(j).name +" " + to);* to\_vert = curr\_vert.**edg**.get(j).**v**;  
 index = j;  
 **break**;  
 }  
 }  
 **if**(to\_vert == **null**) {  
 **return**;  
 }  
 curr\_vert.**edg**.remove(index);  
 }  
 *// vertices.add(new Vertex(x, y, name));* }  
  
  
};  
  
**public class** MainClass **extends** QMainWindow  
{  
  
 **static** Ui\_MainWindow *MyWindow* = **new** Ui\_MainWindow();  
 **static** Ui\_AddFileForm *AddFileForm* = **new** Ui\_AddFileForm();  
 **static char** *White*=**'w'**; *//Введем константные цвета* **static char** *Black*=**'b'**;  
 **static char** *Gray*=**'g'**;  
 */\*static QColor first = QColor.fromRgb(0, 0, 0, 255); // по умолчанию белый  
 static QColor second = QColor.fromRgb(105, 105, 105, 255); // по умолчанию серый  
 static QColor fird = QColor.fromRgb(240, 248, 255, 255); // по умолчанию черный\*/* **static** Tree *tree*=*MyWindow*.*save\_tree*; *//Создадим дерево* **static int** *TREE\_FIELD\_X* = 170;  
 **static int** *TREE\_FIELD\_Y* = 20;  
 **static int** *arrowWidth*=15;  
 **static int** *arrowHeight*=15;  
 **static int** *TREE\_VERTEXES\_RADIUS* = 15;  
 **int TREE\_FIELD\_WIDTH** = 600;  
 **int TREE\_FIELD\_HEIGHT** = 400;  
  
 ArrayList<Vertex> **selected**= **new** ArrayList<Vertex>();  
 **boolean processing**;  
 **boolean dragging**;  
 **static boolean** *k*;  
 **boolean algorithm**; *// отслеживает, запущен ли агоритм сортировки* Vertex **draggable**;  
  
  
  
  
  
 **public** MainClass()  
 {  
 **algorithm** =**false**;  
 *MyWindow*.setupUi(**this**);  
 *MyWindow*.**save**.**triggered**.connect(*MyWindow*, **"on\_actionSave\_clicked()"**);  
 *MyWindow*.**from\_file**.**triggered**.connect(*MyWindow*, **"on\_actionFrom\_file\_clicked()"**);  
 *MyWindow*.**option**.**triggered**.connect(*MyWindow*, **"on\_actionOption\_clicked()"**);  
 *MyWindow*.**generate**.**triggered**.connect(*MyWindow*, **"on\_actionGenerate\_clicked()"**);  
 show();  
 }  
  
 **public static void** main(String ... arg)  
 {  
 QApplication.*initialize*(arg);  
 *//  
  
 //* MainClass text = **new** MainClass();  
  
 QApplication.*execStatic*();  
 }  
  
 **public void** paintEvent(QPaintEvent e)  
 {  
 **if**(!**algorithm**)  
 {  
 QPainter p = **new** QPainter(**this**);  
 p.setRenderHint(QPainter.RenderHint.***Antialiasing***, **true**);  
 p.setBrush(**new** QBrush(QColor.*fromRgb*(220, 220, 220, 255)));  
 p.drawRect(*TREE\_FIELD\_X*, *TREE\_FIELD\_Y*, **TREE\_FIELD\_WIDTH**, **TREE\_FIELD\_HEIGHT**);  
  
 **for** (**int** i = 0; i < *tree*.**vertices**.size(); i++) {  
 QPen pen = **new** QPen();  
 pen.setBrush(**new** QBrush(QColor.*fromRgb*(0, 0, 0, 255)));  
 pen.setWidth(1);  
 p.setPen(pen);  
 **for** (**int** j = 0; j < *tree*.**vertices**.get(i).**edg**.size(); j++) {  
 QPointF p0 = **new** QPointF(*TREE\_FIELD\_X* + *tree*.**vertices**.get(i).**x**, *TREE\_FIELD\_Y* + *tree*.**vertices**.get(i).**y**);  
 QPointF p1 = **new** QPointF(*TREE\_FIELD\_X* + *tree*.**vertices**.get(i).**edg**.get(j).**v**.**x**, *TREE\_FIELD\_Y* + *tree*.**vertices**.get(i).**edg**.get(j).**v**.**y**);  
 pen.setBrush(**new** QBrush(QColor.*fromRgb*(0, 0, 0, 255)));  
 p.drawLine(p0, p1);  
  
 **double** dx, dy;  
 **int** m;  
 **if** (p1.x() > p0.x()) {  
 dx = p1.x() - p0.x();  
 dy = p1.y() - p0.y();  
 m = 1;  
 } **else** {  
 dx = -p1.x() + p0.x();  
 dy = -p1.y() + p0.y();  
 m = -1;  
 }  
 **double** angle = dy / dx;  
 angle = Math.*atan*(angle);  
  
 QPointF p5 = **new** QPointF(p1.x() - m \* (*TREE\_VERTEXES\_RADIUS*) \* Math.*cos*(angle), p1.y() - m \* (*TREE\_VERTEXES\_RADIUS*) \* Math.*sin*(angle));  
 QPointF p2 = **new** QPointF(p1.x() - m \* (*arrowHeight* + *TREE\_VERTEXES\_RADIUS*) \* Math.*cos*(angle), p1.y() - m \* (*arrowHeight* + *TREE\_VERTEXES\_RADIUS*) \* Math.*sin*(angle));  
 angle = Math.*atan*(((-1) \* dx) / dy);  
 QPointF p3 = **new** QPointF(p2.x() - (*arrowWidth* / 2) \* Math.*cos*(angle), p2.y() - (*arrowWidth* / 2) \* Math.*sin*(angle));  
  
 QPointF p4 = **new** QPointF(p2.x() + (*arrowWidth* / 2) \* Math.*cos*(angle), p2.y() + (*arrowWidth* / 2) \* Math.*sin*(angle));  
 QPolygonF arrow = **new** QPolygonF();  
 arrow.add(p5);  
 arrow.add(p3);  
 arrow.add(p4);  
 p.drawConvexPolygon(arrow);  
 pen.setBrush(**new** QBrush(QColor.*fromRgb*(0, 0, 0, 255)));  
 pen.setWidth(1);  
 p.setPen(pen);  
 }  
 }  
 **for** (**int** i = 0; i < *tree*.**vertices**.size(); i++) {  
 QRect rect = **new** QRect(*TREE\_FIELD\_X* + *tree*.**vertices**.get(i).**x** - *TREE\_VERTEXES\_RADIUS*,  
 *TREE\_FIELD\_Y* + *tree*.**vertices**.get(i).**y** - *TREE\_VERTEXES\_RADIUS*,  
 *TREE\_VERTEXES\_RADIUS* \* 2, *TREE\_VERTEXES\_RADIUS* \* 2);  
 **if** (**selected**.contains(*tree*.**vertices**.get(i)))  
 p.setBrush(**new** QBrush(QColor.*fromRgb*(180, 230, 90, 255)));  
 **else** p.setBrush(**new** QBrush(QColor.*fromRgb*(160, 230, 255, 255)));  
 p.drawEllipse(rect);  
 p.drawText(rect, 4, Integer.*toString*(*tree*.**vertices**.get(i).**name**)); *//* }  
 p.end();  
 }**else** {  
 QPainter p = **new** QPainter(**this**);  
 p.setRenderHint(QPainter.RenderHint.***Antialiasing***, **true**);  
 p.setBrush(**new** QBrush(QColor.*fromRgb*(220, 220, 220, 255)));  
 p.drawRect(*TREE\_FIELD\_X*, *TREE\_FIELD\_Y*, **TREE\_FIELD\_WIDTH**, **TREE\_FIELD\_HEIGHT**);  
  
 **for** (**int** i = 0; i < *tree*.**vertices**.size(); i++) {  
 QPen pen = **new** QPen();  
 pen.setBrush(**new** QBrush(QColor.*fromRgb*(0, 0, 0, 255)));  
 pen.setWidth(1);  
 p.setPen(pen);  
 **for** (**int** j = 0; j < *tree*.**vertices**.get(i).**edg**.size(); j++) {  
  
 QPointF p0 = **new** QPointF(*TREE\_FIELD\_X* + *tree*.**vertices**.get(i).**x**, *TREE\_FIELD\_Y* + *tree*.**vertices**.get(i).**y**);  
 QPointF p1 = **new** QPointF(*TREE\_FIELD\_X* + *tree*.**vertices**.get(i).**edg**.get(j).**v**.**x**, *TREE\_FIELD\_Y* + *tree*.**vertices**.get(i).**edg**.get(j).**v**.**y**);  
 **if**(*tree*.**vertices**.get(i).**edg**.get(j).**type**==**'d'**) *// древесное - толстое* {  
 pen.setWidth(4);  
 p.setPen(pen);  
 }  
 **if**(*tree*.**vertices**.get(i).**edg**.get(j).**type**==**'u'**) *// направленное вперед - зеленое* {  
 pen.setBrush(**new** QBrush(QColor.***green***));  
 p.setPen(pen);  
 }  
 **if**(*tree*.**vertices**.get(i).**edg**.get(j).**type**==**'b'**) *// направленное назад - красное* {  
 pen.setBrush(**new** QBrush(QColor.***red***));  
 p.setPen(pen);  
 }  
 **if**(*tree*.**vertices**.get(i).**edg**.get(j).**type**==**'p'**) *// поперечное - серое* {  
 pen.setBrush(**new** QBrush(QColor.***magenta***));  
 p.setPen(pen);  
 }  
 p.drawLine(p0, p1);  
  
 **double** dx, dy;  
 **int** m;  
 **if** (p1.x() > p0.x()) {  
 dx = p1.x() - p0.x();  
 dy = p1.y() - p0.y();  
 m = 1;  
 } **else** {  
 dx = -p1.x() + p0.x();  
 dy = -p1.y() + p0.y();  
 m = -1;  
 }  
 **double** angle = dy / dx;  
 angle = Math.*atan*(angle);  
  
 QPointF p5 = **new** QPointF(p1.x() - m \* (*TREE\_VERTEXES\_RADIUS*) \* Math.*cos*(angle), p1.y() - m \* (*TREE\_VERTEXES\_RADIUS*) \* Math.*sin*(angle));  
 QPointF p2 = **new** QPointF(p1.x() - m \* (*arrowHeight* + *TREE\_VERTEXES\_RADIUS*) \* Math.*cos*(angle), p1.y() - m \* (*arrowHeight* + *TREE\_VERTEXES\_RADIUS*) \* Math.*sin*(angle));  
 angle = Math.*atan*(((-1) \* dx) / dy);  
 QPointF p3 = **new** QPointF(p2.x() - (*arrowWidth* / 2) \* Math.*cos*(angle), p2.y() - (*arrowWidth* / 2) \* Math.*sin*(angle));  
  
 QPointF p4 = **new** QPointF(p2.x() + (*arrowWidth* / 2) \* Math.*cos*(angle), p2.y() + (*arrowWidth* / 2) \* Math.*sin*(angle));  
 QPolygonF arrow = **new** QPolygonF();  
 arrow.add(p5);  
 arrow.add(p3);  
 arrow.add(p4);  
 p.drawConvexPolygon(arrow);  
 pen.setBrush(**new** QBrush(QColor.*fromRgb*(0, 0, 0, 255)));  
 pen.setWidth(1);  
 p.setPen(pen);  
 }  
 }  
 **for** (**int** i = 0; i < *tree*.**vertices**.size(); i++) {  
 QRect rect = **new** QRect(*TREE\_FIELD\_X* + *tree*.**vertices**.get(i).**x** - *TREE\_VERTEXES\_RADIUS*,  
 *TREE\_FIELD\_Y* + *tree*.**vertices**.get(i).**y** - *TREE\_VERTEXES\_RADIUS*,  
 *TREE\_VERTEXES\_RADIUS* \* 2, *TREE\_VERTEXES\_RADIUS* \* 2);  
 **if** (*tree*.**vertices**.get(i).**color** == *White*) {  
 p.setBrush(**new** QBrush(*MyWindow*.*fird*));  
 }  
 **else** {  
 **if** (*tree*.**vertices**.get(i).**color** == *Gray*) {  
 p.setBrush(**new** QBrush(*MyWindow*.*second*));  
 }  
 **else if** (*tree*.**vertices**.get(i).**color** == *Black*) {  
 p.setBrush(**new** QBrush(*MyWindow*.*first*));  
 }  
 }  
 p.drawEllipse(rect);  
  
  
 **if**(*tree*.**vertices**.get(i).**color** == *Black*) { *// возвращаем цвет линий черным* QPen pen = **new** QPen();  
 pen.setBrush(**new** QBrush(*MyWindow*.*fird/\*QColor.fromRgb(240, 248, 255, 255)\*/*));*// цвет линий задаётся белым, чтобы номер вершины был виден, таже при закрашенной в черный вершине* pen.setWidth(1);  
 p.setPen(pen);  
 p.drawText(rect, 4, Integer.*toString*(*tree*.**vertices**.get(i).**name**));  
 *// QPen pen = new QPen();* pen.setBrush(**new** QBrush(QColor.*fromRgb*(0,0,0, 255)));  
 *//pen.setBrush(new QBrush(MyWindow.first));* pen.setWidth(1);  
 p.setPen(pen);  
 }**else** p.drawText(rect, 4, Integer.*toString*(*tree*.**vertices**.get(i).**name**));  
 }  
 p.end();  
 }  
 }  
 **public void** mousePressEvent(QMouseEvent e)  
 {  
 **if** (**processing**) **return**;  
 *//обработка нажатия правой кнопки мыши* Vertex z;  
 **if** (e.button() == Qt.MouseButton.***RightButton***)  
 {  
 **for** (**int** i = 0; i < *tree*.**vertices**.size(); i++)  
 *//если курсор указывает на вершину* **if** ((Math.*abs*(*tree*.**vertices**.get(i).**x** + *TREE\_FIELD\_X* - e.x()) <= *TREE\_VERTEXES\_RADIUS* \* 2) &&  
 (Math.*abs*(*tree*.**vertices**.get(i).**y** + *TREE\_FIELD\_Y* - e.y()) <= *TREE\_VERTEXES\_RADIUS* \* 2))  
 {  
 *//удаление вершины из списка выбранных и из списка вершин* z=*tree*.**vertices**.get(i);  
 **for**(**int** j=0;j<**selected**.size();j++)  
 **if**(**selected**.get(j)==z) **selected**.remove(j);  
 *tree*.removeVertex(z);  
 repaint();  
 **return**;  
 }  
 *//добавление ребер (в случае, если курсор не указывает на вершину)* **if**(**selected**.size()==2)  
 {  
 Vertex u=**selected**.get(0);  
 Vertex v=**selected**.get(1);  
 **for** (**int** j = 0; j < u.**edg**.size(); j++)  
 **if** (u.**edg**.get(j).**v** == v) **return**;  
 u.**edg**.add(**new** Edge(v));  
 }  
 }  
 repaint();  
 }  
  
 **public void** mouseMoveEvent(QMouseEvent e)  
 {  
 **if** (**processing**) **return**;  
 *//обработка начала перемещения вершины* **for** (**int** i = 0; i < *tree*.**vertices**.size(); i++) {  
 **if** (**dragging** == **true**) **break**;  
 *//если курсор указывает на вершину* **if** ((Math.*abs*(*tree*.**vertices**.get(i).**x** + *TREE\_FIELD\_X* - e.x()) <= *TREE\_VERTEXES\_RADIUS* \* 2) &&  
 (Math.*abs*(*tree*.**vertices**.get(i).**y** + *TREE\_FIELD\_Y* - e.y()) <= *TREE\_VERTEXES\_RADIUS* \* 2)) {  
  
 **dragging** = **true**;  
 **draggable** = *tree*.**vertices**.get(i);  
 **break**;  
 }  
 }  
 *//обработка процесса перемещения вершины* **if** (**dragging**)  
 {  
 *//изменение координат* **draggable**.**x** = e.x() - *TREE\_FIELD\_X*;  
 **draggable**.**y** = e.y() - *TREE\_FIELD\_Y*;  
  
 *//поправка на границы* **if** (e.x() > *TREE\_FIELD\_X* + **TREE\_FIELD\_WIDTH** - *TREE\_VERTEXES\_RADIUS*)  
 **draggable**.**x** = **TREE\_FIELD\_WIDTH** - *TREE\_VERTEXES\_RADIUS*;  
 **if** (e.x() < *TREE\_FIELD\_X* + *TREE\_VERTEXES\_RADIUS*)  
 **draggable**.**x** = *TREE\_VERTEXES\_RADIUS*;  
 **if** (e.y() > *TREE\_FIELD\_Y* + **TREE\_FIELD\_HEIGHT** - *TREE\_VERTEXES\_RADIUS*)  
 **draggable**.**y** = **TREE\_FIELD\_HEIGHT** - *TREE\_VERTEXES\_RADIUS*;  
 **if** (e.y() < *TREE\_FIELD\_Y* + *TREE\_VERTEXES\_RADIUS*)  
 **draggable**.**y** = *TREE\_VERTEXES\_RADIUS*;  
 repaint();  
 }  
 }  
  
 **public void** mouseReleaseEvent(QMouseEvent e)  
 {  
 **if** (**processing**) **return**;  
 *//обработка окончания перетаскивания* **if** (**dragging**)  
 {  
 **dragging** = **false**;  
 **draggable** = **null**;  
 repaint();  
 **return**;  
 }  
  
 *//обработка нажатия левой кнопки мыши* **if** (e.button() == Qt.MouseButton.***LeftButton***)  
 {  
 *//проверка на границы* **if** ((e.x() < *TREE\_FIELD\_X* + *TREE\_VERTEXES\_RADIUS*) ||  
 (e.x() > *TREE\_FIELD\_X* + **TREE\_FIELD\_WIDTH** - *TREE\_VERTEXES\_RADIUS*) ||  
 (e.y() < *TREE\_FIELD\_Y* + *TREE\_VERTEXES\_RADIUS*) ||  
 (e.y() > *TREE\_FIELD\_Y* + **TREE\_FIELD\_HEIGHT** - *TREE\_VERTEXES\_RADIUS*))  
 **return**;  
  
 **for** (**int** i = 0; i < *tree*.**vertices**.size(); i++)  
  
 *//если курсор указывает на вершину* **if** ((Math.*abs*(*tree*.**vertices**.get(i).**x** + *TREE\_FIELD\_X* - e.x()) <= *TREE\_VERTEXES\_RADIUS* \* 2) &&  
 (Math.*abs*(*tree*.**vertices**.get(i).**y** + *TREE\_FIELD\_Y* - e.y()) <= *TREE\_VERTEXES\_RADIUS* \* 2))  
 {  
 *//добавление/удаление из списка выбранных* **if** (**selected**.contains(*tree*.**vertices**.get(i)))  
 **selected**.remove(**selected**.indexOf(*tree*.**vertices**.get(i)));  
 **else** {  
 **if**(**selected**.size()<2)  
 {  
 **selected**.add(*tree*.**vertices**.get(i));  
 }  
 **else** {  
 **selected**.remove(**selected**.indexOf(**selected**.get(**selected**.size()-1)));  
 **selected**.add(*tree*.**vertices**.get(i));  
 }  
 }  
 repaint();  
 **return**;  
 }  
 *//добавление вершины (в случае, если курсор не указывает на вершину)  
 tree*.addVertex(e.x() - *TREE\_FIELD\_X*, e.y() - *TREE\_FIELD\_Y*);  
 }  
 repaint();  
 }  
  
 **public void** keyPressEvent(QKeyEvent e)  
 {  
 **if** (**processing**) **return**;  
 *//удаление ребер, связывающих выбранные вершины, по нажатию пробе* **if** (e.key() ==Qt.Key.***Key\_Backspace***.value())  
 {  
 **if**(**selected**.size()==2)  
 {  
 **for**(**int** i=0;i<**selected**.get(0).**edg**.size();i++)  
 **if**(**selected**.get(0).**edg**.get(i).**v**==**selected**.get(1)) **selected**.get(0).**edg**.remove(i);  
 }  
 }  
 repaint();  
 }  
  
 **public void** resizeEvent(QResizeEvent e)  
 {  
 **TREE\_FIELD\_WIDTH** = e.size().width() - *TREE\_FIELD\_X* - 20;  
 **TREE\_FIELD\_HEIGHT** = e.size().height() - *TREE\_FIELD\_Y* - 20;  
 }  
  
 **void** Wait()  
 {  
 **if**(*MyWindow*.*visualisation*) {  
 *MyWindow*.**Next**.setEnabled(**true**);  
 **while** (!*k*) {  
 QCoreApplication.*processEvents*();  
 }  
 *k* = **false**;  
 *MyWindow*.**Next**.setEnabled(**false**);  
 }  
 }  
 **static boolean** *Ch*=**true**;  
 **public void** dfs1(Vertex cur) *//Поиск в глубину* {  
 **if**(*MyWindow*.*visualisation*)*MyWindow*.**textBrowser**.append(**"Вошли в вершину "**+ cur.**name** +**". Красим её в серый."**);  
 *MyWindow*.*write\_massage\_to\_file*(**"Вошли в вершину "**+ cur.**name** +**". Красим её в серый."**, *MyWindow*.*visualisation*);  
 cur.**color**=*Gray*; *//Красим вершину в серый* repaint();  
 Wait();  
 **for** (**int** i = 0; i<cur.**edg**.size(); i++) *// Проходим по всем ребрам идущим из вершины* {  
 **if** (!(cur.**edg**.get(i).**v**.**color**==*Black*)) *// Если ребро ведет не в черную вершину* {  
 **if**(cur.**edg**.get(i).**v**.**color**==*Gray*) *// Если ребро ведет в уже серую вершины -> цикл* {  
 cur.**edg**.get(i).**type**=**'b'**;  
 cur.**edg**.get(i).**used**=**true**;  
  
 **if**(*MyWindow*.*visualisation*)*MyWindow*.**textBrowser**.append(**"Переход от вершины "**+ cur.**edg**.get(i).**v**.**name** +**" к её сыну "** + cur.**edg**.get(i).**v**.**name** + **"."**);  
 *MyWindow*.*write\_massage\_to\_file*(**"Переход от вершины "**+ cur.**name** +**" к её сыну "** + cur.**edg**.get(i).**v**.**name** + **"."**, *MyWindow*.*visualisation*);  
 **if**(*MyWindow*.*visualisation*)*MyWindow*.**textBrowser**.append(**"Сын "** + cur.**edg**.get(i).**v**.**name** + **" является серой вершиной. "**);  
 *MyWindow*.*write\_massage\_to\_file*(**"Сын "** + cur.**edg**.get(i).**v**.**name** + **" является серой вершиной. "**, *MyWindow*.*visualisation*);  
 *MyWindow*.**textBrowser**.append(**"Ошибка1!"**);  
 *Ch*=**false**; *// Ch=false - завершаем алгоритм* **return**;  
 }  
 cur.**edg**.get(i).**used** = **true**;*// прошли по ребру* cur.**edg**.get(i).**type**=**'d'**; *// Ребро - древесное* **if**(*MyWindow*.*visualisation*)*MyWindow*.**textBrowser**.append(**"Переход от вершины "**+ cur.**name** +**" к её сыну "** + cur.**edg**.get(i).**v**.**name** + **"."**);  
 *MyWindow*.*write\_massage\_to\_file*(**"Переход от вершины "**+ cur.**name** +**" к её сыну "** + cur.**edg**.get(i).**v**.**name** + **"."**, *MyWindow*.*visualisation*);  
  
 dfs1(cur.**edg**.get(i).**v**);*//Запускаем алгоритм рекурсивно для следующей вершнины* repaint();  
 Wait();  
 }  
 }  
 **if**(*MyWindow*.*visualisation*)*MyWindow*.**textBrowser**.append(**"Выходим из вершины "**+ cur.**name** +**". Красим её в черный."**);  
 *MyWindow*.*write\_massage\_to\_file*(**"Выходим из вершины "**+ cur.**name** +**". Красим её в черный."**, *MyWindow*.*visualisation*);  
 **for**(**int** j=0;j<cur.**edg**.size();j++) { *// Проверяем все ребра из текущей вершины(из которой мы сейчас вышли)* **if** (!cur.**edg**.get(j).**used**) {  
 **if** (cur.**edg**.get(j).**v**.**color** == *Gray*) cur.**edg**.get(j).**type** = **'b'**;*// Если ребро идет в серую вершину -> оно направлено назад* **if** (cur.**edg**.get(j).**v**.**color** == *Black*) *// Если ребро идет в черную вершину -> оно направлено вперед* {  
 cur.**edg**.get(j).**type** = **'u'**;  
 }  
 }  
 }  
 cur.**color**=*Black*;*//Вышли из вершины - покрасили в черный* repaint();  
 Wait();  
 **for**(**int** i=0;i<*tree*.**vertices**.size();i++) *//Проверим все ребра в графе, если есть ребро Белой->Черную - тогда это поперечные ребра* {  
 **if**(*tree*.**vertices**.get(i).**color**==*White*)  
 {  
 **for** (**int** k = 0; k < *tree*.**vertices**.get(i).**edg**.size(); k++)  
 **if**(*tree*.**vertices**.get(i).**edg**.get(k).**v**.**color** == *Black*)  
 {  
 *tree*.**vertices**.get(i).**edg**.get(k).**used**=**true**;  
 *tree*.**vertices**.get(i).**edg**.get(k).**type** = **'p'**;  
 }  
 }  
 }  
 *tree*.**RTopSort**.add(cur);*//Добавили её в список* }  
  
 **void** TopSort() *//Топологическая сортировка* {  
 *Ch*=**true**;*//В начале работы Ch=true - циклов нет  
 tree*.**RTopSort**.clear(); *//Очистим список вершин* **int** m=0; *// m -номер вершины для которой будем вызывать поиск в глубину* **while**(*tree*.**RTopSort**.size()<*tree*.**vertices**.size()) *//пока все вершины не попали в нужый список* {  
 **if**((*tree*.**vertices**.get(m).**color**==*White*)) *//Если вершина не пройдена* {  
 dfs1(*tree*.**vertices**.get(m)); *// запускаем поиск в глубину для неё* **if**(!*Ch*) **return**;*// Если нашелся цикл, завершаем работу алгоритма* }  
 **else** m++;*//иначе переходим к следуюещей* }  
 }  
  
  
 **public void** on\_Next\_clicked()  
 {  
 *k*=**true**;  
 }  
 **public void** on\_step\_clicked()  
 {  
 **algorithm**=**true**;  
 **processing** = **true**;  
 **selected**.clear();  
 *MyWindow*.**step**.setEnabled(**false**);  
 *MyWindow*.**clear**.setEnabled(**false**);  
 **int** m = 0;  
 TopSort();  
 **for**(**int** i=0;i<*tree*.**vertices**.size();i++)  
 **for**(**int** j=0;j<*tree*.**vertices**.get(i).**edg**.size();j++)*// 'u' - направленное вперед , 'b' - направленное назад , 'p' - поперечное* {  
 **if** (*tree*.**vertices**.get(i).**edg**.get(j).**type** == **'n'**)  
 *MyWindow*.**Result**.append(Integer.*toString*(*tree*.**vertices**.get(i).**name**) + **"->"** + Integer.*toString*(*tree*.**vertices**.get(i).**edg**.get(j).**v**.**name**) + **"Неиспользованное"**);  
 **if** (*tree*.**vertices**.get(i).**edg**.get(j).**type** == **'d'**)  
 *MyWindow*.**Result**.append(Integer.*toString*(*tree*.**vertices**.get(i).**name**) + **"->"** + Integer.*toString*(*tree*.**vertices**.get(i).**edg**.get(j).**v**.**name**) + **"Древесное"**);  
 **if** (*tree*.**vertices**.get(i).**edg**.get(j).**type** == **'u'**)  
 *MyWindow*.**Result**.append(Integer.*toString*(*tree*.**vertices**.get(i).**name**) + **"->"** + Integer.*toString*(*tree*.**vertices**.get(i).**edg**.get(j).**v**.**name**) + **"Направленное вперед"**);  
 **if** (*tree*.**vertices**.get(i).**edg**.get(j).**type** == **'p'**)  
 *MyWindow*.**Result**.append(Integer.*toString*(*tree*.**vertices**.get(i).**name**) + **"->"** + Integer.*toString*(*tree*.**vertices**.get(i).**edg**.get(j).**v**.**name**) + **"Поперечное"**);  
 **if** (*tree*.**vertices**.get(i).**edg**.get(j).**type** == **'b'**)  
 *MyWindow*.**Result**.append(Integer.*toString*(*tree*.**vertices**.get(i).**name**) + **"->"** + Integer.*toString*(*tree*.**vertices**.get(i).**edg**.get(j).**v**.**name**) + **"Направленное назад"**);  
 }  
 String str=**"Список отсортированных вершин: "**;  
 **for**(**int** i=0;i<*tree*.**RTopSort**.size();i++) {  
 str += **" "**;  
 str += *tree*.**RTopSort**.get(*tree*.**RTopSort**.size() - i - 1).**name**;  
 }  
 **if**(*Ch*) *MyWindow*.**Result**.append(str);  
 **else** *MyWindow*.**Result**.append(**"Есть цикл!"**);  
 **if**(*MyWindow*.*visualisation*) {  
 repaint();  
 Wait();  
 }  
 **else** {  
 *MyWindow*.*visualisation*=**true**;  
 repaint();  
 Wait();  
 *MyWindow*.*visualisation*=**false**;  
 }  
 **for**(**int** i=0;i<*tree*.**vertices**.size();i++) {  
 *tree*.**vertices**.get(i).**color** = *White*;  
 **for** (**int** j = 0; j < *tree*.**vertices**.get(i).**edg**.size(); j++) {  
 *tree*.**vertices**.get(i).**edg**.get(j).**used** = **false**;  
 *tree*.**vertices**.get(i).**edg**.get(j).**type** = **'n'**;  
 }  
 }  
 **algorithm**=**false**;  
 *MyWindow*.**step**.setEnabled(**true**);  
 *MyWindow*.**clear**.setEnabled(**true**);  
 **processing** = **false**;  
 repaint();  
 }  
 **public void** on\_clear\_clicked()  
 {  
 **selected**.clear();  
 *tree*.**RTopSort**.clear();  
 *tree*.**vertices**.clear();  
 repaint();  
 }  
 **protected void** closeEvent( QCloseEvent e ) *// переопределено закрытие файла, т.к. нужно чтобы файл закрывался( иначе не сохраняются данные)* {  
 **if**(*MyWindow*.*write\_file\_exist*)  
 *MyWindow*.*outFile*.close();  
 **if**(*MyWindow*.*read\_file\_exist*)  
 *MyWindow*.*fromFile*.close();  
 e.accept();  
 }  
}

**UI\_MainWindow:**

*/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
 \*\* Form generated from reading ui file 'mainwindow.jui'  
 \*\*  
 \*\* Created by: Qt User Interface Compiler version 4.8.6  
 \*\*  
 \*\* WARNING! All changes made in this file will be lost when recompiling ui file!  
 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/***import** com.trolltech.qt.QSignalEmitter;  
**import** com.trolltech.qt.core.\*;  
**import** com.trolltech.qt.gui.\*;  
**import** java.util.Random;  
**import** java.math.\*;  
**import** javafx.scene.transform.Transform;  
**import** sun.font.TrueTypeFont;  
**import** sun.misc.Signal;  
  
  
  
**public class** Ui\_MainWindow **implements** com.trolltech.qt.QUiForm<QMainWindow>  
{  
 **public** QWidget **centralWidget**;  
 **public** QPushButton **step**;  
 **public** QPushButton **clear**;  
 **public** QPushButton **Next**;  
 **public** QTextBrowser **textBrowser**;  
 **public** QTextBrowser **Result**;  
 **public** QStatusBar **statusBar**;  
 *//* **public** QMenu **file**;  
 **public** QAction **from\_file**;  
 **public** QAction **save**;  
 **public** QAction **generate**;  
 **public** QMenu **option**;  
 **public** QAction **option\_of\_graph**;  
 **public** QMenuBar **menubar**;  
  
 *//public String answer;* **public static** Ui\_AddFileForm *form* = **new** Ui\_AddFileForm()*/\*null\*/*;  
 **public static** Ui\_GenerateForm *generate\_form* = **new** Ui\_GenerateForm();  
 **public static boolean** *write\_file\_exist* = **false**;  
 **public static** Ui\_OptionForm *option\_form* =**null**;  
 **public static** QFile *outFile*;  
 **public static** QTextStream *out*;  
  
 **public static boolean** *read\_file\_exist* = **false**;  
 **public static** QFile *fromFile*;  
 **public static** QTextStream *fromF*;  
  
  
 **public static boolean** *visualisation* = **true**;  
  
 **static** QColor *first* = QColor.*fromRgb*(0, 0, 0, 255); *// по умолчанию черный* **static** QColor *second* = QColor.*fromRgb*(105, 105, 105, 255); *// по умолчанию серый* **static** QColor *fird* = QColor.*fromRgb*(240, 248, 255, 255); *// по умолчанию белый* **static** QColor *fourth* = QColor.*fromRgb*(0, 100, 0, 255); *// зелёный* **public static** Tree *save\_tree* = **new** Tree();  
 *//* **public** Ui\_MainWindow() { **super**(); }  
 *// public ~Ui\_MainWindow(){ outFile.close();}* **public void** setupUi(QMainWindow MainWindow)  
 {  
 MainWindow.setObjectName(**"MainWindow"**);  
 MainWindow.resize(**new** QSize(746, 500*/\*469\*/*).expandedTo(MainWindow.minimumSizeHint()));  
 **centralWidget** = **new** QWidget(MainWindow);  
 **centralWidget**.setObjectName(**"centralWidget"**);  
 **step** = **new** QPushButton(**centralWidget**);  
 **step**.setObjectName(**"step"**);  
 **step**.setGeometry(**new** QRect(10, 30, 141, 23));  
 **step**.setCheckable(**false**);  
 **step**.setChecked(**false**);  
 **step**.setFlat(**false**);  
 **clear** = **new** QPushButton(**centralWidget**);  
 **clear**.setObjectName(**"clear"**);  
 **clear**.setGeometry(**new** QRect(10, 90, 141, 23));  
 **clear**.setCheckable(**false**);  
 **clear**.setChecked(**false**);  
 **clear**.setFlat(**false**);  
 **Next** = **new** QPushButton(**centralWidget**);  
 **Next**.setObjectName(**"Next"**);  
 **Next**.setGeometry(**new** QRect(10, 60, 141, 23));  
 **textBrowser** = **new** QTextBrowser(**centralWidget**);  
 **textBrowser**.setObjectName(**"textBrowser"**);  
 **textBrowser**.setGeometry(**new** QRect(10, 120, 141, 192));  
 **Result** = **new** QTextBrowser(**centralWidget**);  
 **Result**.setObjectName(**"Result"**);  
 **Result**.setGeometry(**new** QRect(10, 320, 141, 141));  
 MainWindow.setCentralWidget(**centralWidget**);  
 **statusBar** = **new** QStatusBar(MainWindow);  
 **statusBar**.setObjectName(**"statusBar"**);  
  
  
  
  
  
 *//  
  
  
  
 //  
 // menubar = QMenuBar.* **menubar** = **new** QMenuBar(MainWindow);  
 *//файл:* **file** = **new** QMenu(**menubar**);  
 **file**.setObjectName(**"file"**);  
 **file** = **menubar**.addMenu(**"Файл"**);  
 **from\_file** = **new** QAction(MainWindow);  
 **from\_file**.setObjectName(**"in\_file"***/\*"Ввести граф из файла"\*/*);  
  
 **menubar**.addAction(**file**.menuAction());  
  
 **file**.addSeparator();  
 **file**.addAction(**from\_file**);  
  
 **save** = **new** QAction(MainWindow);  
  
 **save**.setObjectName(**"save"**);  
 **file**.addAction(**save**);  
  
 **generate** = **new** QAction(MainWindow);  
 **generate**.setObjectName(**"generate"**);  
 **file**.addAction(**generate**);  
  
  
 *//найстройки:* **option** = **new** QMenu(**menubar**);  
 **option**.setObjectName(**"option"**);  
 **option** = **menubar**.addMenu(**"Настойки"**);  
 **option\_of\_graph**= **new** QAction(MainWindow);  
 **option**.addAction(**option\_of\_graph**);  
 **option\_of\_graph**.setObjectName(**"option\_of\_graph"**);  
  
  
 **menubar**.setGeometry(**new** QRect(0, 0, 117, 23));  
  
  
 *//* MainWindow.setStatusBar(**statusBar**);  
 retranslateUi(MainWindow);  
  
  
 *//  
 //.clicked.connect(Form, "close()");  
  
 // QDialogButtonBox.StandardButton.Close.  
 //* MainWindow.connectSlotsByName();  
 } *// setupUi* **void** retranslateUi(QMainWindow MainWindow)  
 {  
 MainWindow.setWindowTitle(com.trolltech.qt.core.QCoreApplication.*translate*(**"MainWindow"**, **"\u0422\u043e\u043f\u043e\u043b\u043e\u0433\u0438\u0447\u0435\u0441\u043a\u0430\u044f \u0421\u043e\u0440\u0442\u0438\u0440\u043e\u0432\u043a\u0430"**, **null**));  
 **step**.setText(com.trolltech.qt.core.QCoreApplication.*translate*(**"MainWindow"**, **"\u0417\u0430\u043f\u0443\u0441\u0442\u0438\u0442\u044c \u0422\u043e\u043f.\u0421\u043e\u0440\u0442."**, **null**));  
 **clear**.setText(com.trolltech.qt.core.QCoreApplication.*translate*(**"MainWindow"**, **"\u041e\u0447\u0438\u0441\u0442\u0438\u0442\u044c \u0433\u0440\u0430\u0444"**, **null**));  
 **Next**.setText(com.trolltech.qt.core.QCoreApplication.*translate*(**"MainWindow"**, **"Next"**, **null**));  
 **file**.setTitle(com.trolltech.qt.core.QCoreApplication.*translate*(**"MainWindow"**, **"Файл"**, **null**));  
 **from\_file**.setText(com.trolltech.qt.core.QCoreApplication.*translate*(**"MainWindow"**,**"Ввод из файла"**, **null**));  
 **save**.setText(com.trolltech.qt.core.QCoreApplication.*translate*(**"MainWindow"**,**"Вывод результата в файл"**, **null**));  
 **generate**.setText(com.trolltech.qt.core.QCoreApplication.*translate*(**"MainWindow"**,**"Сгенерировать граф"**, **null**));  
  
 **option\_of\_graph**.setText(com.trolltech.qt.core.QCoreApplication.*translate*(**"MainWindow"**,**"Найстройки графа"**, **null**));  
  
 } *// retranslateUi* **public void** on\_actionSave\_clicked()  
 {  
 *// this.textBrowser.append("save нажат");  
 form* = **null**;  
 QWidget new\_widget = **new** QWidget();  
 *form* = **new** Ui\_AddFileForm();  
 *form*.setupUi(new\_widget);  
 *// this.textBrowser.append(form.lineEdit.text());  
 form*.*Ok*.**clicked**.connect(**this**, **"write\_to\_file()"**);  
 new\_widget.show();  
  
 *//form.show();* }  
 **public void** on\_actionGenerate\_clicked()  
 {  
 *// this.textBrowser.append("save нажат");  
 generate\_form* = **null**;  
 QWidget new\_widget = **new** QWidget();  
 *generate\_form* = **new** Ui\_GenerateForm();  
 *generate\_form*.setupUi(new\_widget);  
 *// this.textBrowser.append(form.lineEdit.text());  
 generate\_form*.**Ok**.**clicked**.connect(**this**, **"generate\_new\_tree()"**);  
 new\_widget.show();  
  
 *//form.show();* }  
 **public void** on\_actionFrom\_file\_clicked()  
 {  
 *//this.textBrowser.append("save нажат");  
  
 form* = **null**;  
 QWidget new\_widget = **new** QWidget();  
 *form* = **new** Ui\_AddFileForm();  
 *form*.setupUi(new\_widget);  
 *// this.textBrowser.append(form.lineEdit.text());  
 form*.*Ok*.**clicked**.connect(**this**, **"read\_from\_file()"**);  
 new\_widget.show();  
 *//form.show();* }  
 **public void** on\_actionOption\_clicked()  
 {  
 *//this.textBrowser.append("save нажат");* QWidget new\_widget = **new** QWidget();  
 *option\_form* = **new** Ui\_OptionForm();  
 *option\_form*.setupUi(new\_widget);  
 *option\_form*.**buttonBox**.**clicked**.connect(**this**, **"vis()"**);  
 new\_widget.show();  
 *//form.show();* }  
 **public void** read\_from\_file()  
 {  
  
 **if**(*read\_file\_exist*)  
 {  
 *fromFile*.close();  
 *read\_file\_exist* = **false**;  
 }  
 *// this.textBrowser.append("Попытка найти файл с именем: " + form.inputString);  
 fromFile* = **new** QFile( *form*.*inputString* + **".txt"** */\*".//OutputInfo.txt"\*/*);  
 **if**(!*fromFile*.open(QIODevice.OpenModeFlag.***ReadOnly***, QIODevice.OpenModeFlag.***Text***))  
 {  
 **this**.**textBrowser**.append(**"Файл с именем "** + *form*.*inputString* + **".txt "** + **"не найден."**);  
 *read\_file\_exist* = **false**;  
  
 }  
 **else** {  
 *read\_file\_exist* = **true**;  
 **this**.**textBrowser**.append(**"Граф будет введён из файла: "** + *form*.*inputString* + **".txt"** );  
 *fromF* = **new** QTextStream(*fromFile*);  
 update\_tree();  
 *//outFile.w  
 //out.writeString( "Hi! " + form.inputString);* }  
 *// outFile.close();* }  
 **public void** write\_to\_file()  
 {  
  
 **if**(*write\_file\_exist*)  
 {  
 *outFile*.close();  
 *write\_file\_exist* = **false**;  
 }  
 *// this.textBrowser.append("Попытка найти файл с именем: " + form.inputString);  
 outFile* = **new** QFile(**".//"** + *form*.*inputString* + **".txt"** */\*".//OutputInfo.txt"\*/*);  
 **if**(!*outFile*.open(QIODevice.OpenModeFlag.***WriteOnly***, QIODevice.OpenModeFlag.***Text***))  
 {  
 **this**.**textBrowser**.append(**"Файл с именем "** + *form*.*inputString* + **".txt "** + **"не найден."**);  
 *write\_file\_exist* = **false**;  
  
 }  
 **else** {  
 QSignalEmitter.Signal1<Boolean> answer;  
 *write\_file\_exist* = **true**;  
 **this**.**textBrowser**.append(**"Теперь все действия будут дублироваться в файл: "** + *form*.*inputString* + **".txt"** );  
 *out* = **new** QTextStream(*outFile*);  
 *//outFile.w  
 //out.writeString( "Hi! " + form.inputString);  
 //return QSignalEmitter.signalSender().s;  
 // return QSignalEmitter.AbstractSignal;* }  
 *// outFile.close();* }  
 **public static void** write\_massage\_to\_file( String massege, **boolean** visualisation)  
 {  
 **if**(*write\_file\_exist* && visualisation) {  
 *out*.writeString(massege);  
 *out*.writeString(**"\n"**);  
 }  
 }  
 **public static void** close\_file()  
 {  
 **if**(*write\_file\_exist*)  
 {  
 *outFile*.close();  
 *write\_file\_exist* = **false**;  
 }  
 **if**(*read\_file\_exist*)  
 {  
 *fromFile*.close();  
 *read\_file\_exist* = **false**;  
 }  
  
 }  
 **public static void** vis()  
 {  
 **if**(!*option\_form*.*first*) {  
 *visualisation* = *option\_form*.*checkBox*.isChecked();  
 **if**(*option\_form*.*Green\_yellow*.isChecked())  
 {  
 *first* = QColor.*fromRgb*(0, 100, 0, 255); *//зелёный  
 second* = QColor.*fromRgb*(255,215,0*/\*238,0,0\*/*, 255); *// красный  
 fird* = QColor.*fromRgb*(240, 248, 255, 255); *// белый* }  
 **else** {  
 *first* = QColor.*fromRgb*(0, 0, 0, 255); *// черный  
 second* = QColor.*fromRgb*(105, 105, 105, 255); *//серый  
 fird* = QColor.*fromRgb*(240, 248, 255, 255); *// белый* }  
 }  
 }  
  
 **protected void** closeEvent( QCloseEvent e )  
 {  
  
 *outFile*.close();  
 *fromFile*.close();  
 e.accept();  
 *//e.ignore();* }  
  
  
  
 **public boolean** update\_tree()  
 {  
 **if**(!*read\_file\_exist*)  
 {  
 **return false**;  
 }  
 *save\_tree*.**RTopSort**.clear();  
 *save\_tree*.**vertices**.clear();  
 String all\_information;  
 all\_information = *fromF*.readAll();  
 **textBrowser**.append(all\_information);  
  
 *// String curr\_elem = new String();  
 // добавление узлов графа* **int** save\_int=0;  
 **for**(**int** i=save\_int; i<all\_information.length(); i++)  
 {  
 **if**(all\_information.charAt(i) != **' '**)  
 {  
 **if**(all\_information.charAt(i) == **'{'**)  
 {  
 save\_int = i+1;  
 **break**;  
 }**else** {  
 *save\_tree*.**RTopSort**.clear();  
 *save\_tree*.**vertices**.clear();  
 **textBrowser**.append(**"В файле присутствует неверный синтаксис. Нет открывающей фигурной скобки: "** +all\_information.charAt(i));  
 *fromFile*.close();  
 *read\_file\_exist*=**false**;  
 **return false**;  
 }  
 }  
 }  
 **int** curr\_elem =0;  
 **for**(**int** i=save\_int; i<all\_information.length(); i++) *// добавление вершин* {  
 save\_int = i+1;  
 **if**(all\_information.charAt(i) != **' '**)  
 {  
 **if**(Character.*isDigit*(all\_information.charAt(i)))  
 {  
 curr\_elem = curr\_elem\*10 +all\_information.charAt(i)-**'0'**;  
  
 }**else  
 if**(all\_information.charAt(i) == **','** || all\_information.charAt(i) == **'}'**)  
 {  
 **if**(curr\_elem!=0)  
 {  
 *save\_tree*.addVertex(curr\_elem);  
 curr\_elem = 0;  
 **if**(all\_information.charAt(i) == **'}'**)  
 {  
 **break**;  
 }  
 }**else** {  
 *save\_tree*.**RTopSort**.clear();  
 *save\_tree*.**vertices**.clear();  
 **textBrowser**.append(**"В файле присутствует неверный синтаксис. Нет цифры перед: "** +all\_information.charAt(i));  
 *fromFile*.close();  
 *read\_file\_exist*=**false**;  
 **return false**;  
 }  
 }  
 **else** {  
  
 *save\_tree*.**RTopSort**.clear();  
 *save\_tree*.**vertices**.clear();  
 **if**(all\_information.charAt(i) != **','**)  
 **textBrowser**.append(**"В файле присутствует неверный синтаксис. Тут должна быть закрывающая фигураня скобка или заяптая: "** +all\_information.charAt(i));  
 *fromFile*.close();  
 *read\_file\_exist*=**false**;  
 **return false**;  
 }  
 }  
 }  
 **for**(**int** i=save\_int; i<all\_information.length(); i++) *// добавление вершин* {  
 **while**(i<all\_information.length() && all\_information.charAt(i) ==**' '**)  
 {  
 i++;  
 }  
 **if**(i<all\_information.length() && all\_information.charAt(i) == **'('**)  
 {  
 i++;  
 **while**(all\_information.charAt(i) == **' '**) i++;  
 **if**(!Character.*isDigit*(all\_information.charAt(i)))  
 {  
 *save\_tree*.**RTopSort**.clear();  
 *save\_tree*.**vertices**.clear();  
 **textBrowser**.append(**"В файле присутствует неверный синтаксис. Не цифра: "** +all\_information.charAt(i));  
 *fromFile*.close();  
 *read\_file\_exist*=**false**;  
 **return false**;  
 }  
 **int** from=0;  
 **while**(i<all\_information.length() && Character.*isDigit*(all\_information.charAt(i)))  
 {  
 from = from\*10 +all\_information.charAt(i)-**'0'**;  
 i++;  
 **while**(all\_information.charAt(i) == **' '**) i++;  
 }  
 **while**(all\_information.charAt(i) == **' '**) i++;  
 **if**(i>all\_information.length() ||all\_information.charAt(i) !=**','**)  
 {  
 *save\_tree*.**RTopSort**.clear();  
 *save\_tree*.**vertices**.clear();  
 **if**(all\_information.charAt(i) !=**','**)  
 **textBrowser**.append(**"В файле присутствует неверный синтаксис. Нет запятой: "** +all\_information.charAt(i));  
 **else  
 textBrowser**.append(**"В файле присутствует неверный синтаксис. Нет запятой. "**);  
 *fromFile*.close();  
 *read\_file\_exist*=**false**;  
 **return false**;  
 }  
 i++;  
 **while**(all\_information.charAt(i) == **' '**) i++;  
 **int** to=0;  
 **while**(i<all\_information.length() && Character.*isDigit*(all\_information.charAt(i)))  
 {  
 to = to\*10 +all\_information.charAt(i)-**'0'**;  
 i++;  
 **while**(all\_information.charAt(i) == **' '**) i++;  
 }  
 **if**(i>all\_information.length() ||all\_information.charAt(i) !=**')'**)  
 {  
 *save\_tree*.**RTopSort**.clear();  
 *save\_tree*.**vertices**.clear();  
 **textBrowser**.append(**"В файле присутствует неверный синтаксис. Запись рёбер должна заканчиваться закрывающй скобкой: "** +all\_information.charAt(i));  
 *fromFile*.close();  
 *read\_file\_exist*=**false**;  
 **return false**;  
 }  
 */// добавление ребра  
  
 /\*for (int j = 0; j < u.edg.size(); j++)  
 if (u.edg.get(j).v == v) return;  
 u.edg.add(new Edge(v));\*/  
 /\*\*textBrowser.append("Все вершины дерева: ");  
 for (int k = 0; k < save\_tree.vertices.size(); k++)  
 {  
 textBrowser.append(Integer.toString(save\_tree.vertices.get(k).name));  
 }\*/* Vertex vertex\_from = **null**;  
 **for** (**int** k = 0; k < *save\_tree*.**vertices**.size(); k++)  
 **if** (*save\_tree*.**vertices**.get(k).**name** == from) {  
 vertex\_from = *save\_tree*.**vertices**.get(k);  
 }  
 **if**(vertex\_from == **null**) *// если в дереве нет вершины, с которой хотят соединить путь* {  
 *save\_tree*.**RTopSort**.clear();  
 *save\_tree*.**vertices**.clear();  
 **textBrowser**.append(**"В файле присутствует неверный синтаксис. Данная вершина не найдена: "** +from);  
 *fromFile*.close();  
 *read\_file\_exist*=**false**;  
 **return false**;  
 }  
 Vertex vertex\_to = **null**;  
 **for** (**int** k = 0; k < *save\_tree*.**vertices**.size(); k++)  
 **if** (*save\_tree*.**vertices**.get(k).**name** == to) {  
 vertex\_to = *save\_tree*.**vertices**.get(k);  
 }  
 **if**(vertex\_to == **null**) *// если в дереве нет вершины, с которой хотят соединить путь* {  
 *save\_tree*.**RTopSort**.clear();  
 *save\_tree*.**vertices**.clear();  
 **textBrowser**.append(**"В файле присутствует неверный синтаксис. Данная вершина не найдена: "** +to); *fromFile*.close();  
 *read\_file\_exist*=**false**;  
 **return false**;  
 }  
 vertex\_from.**edg**.add(**new** Edge(vertex\_to));  
 *///* }**else** {  
 *save\_tree*.**RTopSort**.clear();  
 *save\_tree*.**vertices**.clear();  
 **textBrowser**.append(**"В файле присутствует неверный синтаксис. Запись рёбер должна начинаться с открывающей скобки: "** +all\_information.charAt(i));  
 *fromFile*.close();  
 *read\_file\_exist*=**false**;  
 **return false**;  
 }  
 }  
 **textBrowser**.append(**"Дерево было считано."**);  
 **return true**;  
 }  
 **private int** abs(**int** i)  
 {  
 **if**(i>0) **return** i;  
 **else return** -i;  
 }  
 **private double** abs(**double** i)  
 {  
 **if**(i>0) **return** i;  
 **else return** -i;  
 }  
 **public void** generate\_new\_tree()  
 {  
 *// int curr\_ele  
 //save\_tree.addVertex(curr\_elem);* **do** {  
 *save\_tree*.**RTopSort**.clear();  
 *save\_tree*.**vertices**.clear();  
 **for**(**int** i=0;i<*save\_tree*.**vertices**.size();i++) {  
 *save\_tree*.**vertices**.get(i).**color** = **'w'**;  
 **for** (**int** j = 0; j < *save\_tree*.**vertices**.get(i).**edg**.size(); j++) {  
 *save\_tree*.**vertices**.get(i).**edg**.get(j).**used** = **false**;  
 *save\_tree*.**vertices**.get(i).**edg**.get(j).**type** = **'n'**;  
 }  
 }  
 *// обычная случайная генерация:* Random random\_int = **new** Random();  
 **if**(!*generate\_form*.**round\_painting**.isChecked())  
 {  
 **for** (**int** k = 1; k <= abs(*generate\_form*.**count\_of\_top**.value()); k++) {  
 *save\_tree*.addVertex(k);  
 }  
 }**else** {  
  
 *// генерация по кругу:* **double** Radious = 60 \* ((**double**) *generate\_form*.**count\_of\_top**.value()) / (2 \* 3.14); *// чтобы параметр был равен кол-ву вершин(тогда расстояние между вершинами равно 1)* **int** Start = 200;  
 **double** x = 0;  
 **double** y = 0;  
 **for** (**int** k = 0; k < abs(*generate\_form*.**count\_of\_top**.value()); k++) {  
 x = Radious \* Math.*cos*(Math.*toRadians*((k \* 360) / *generate\_form*.**count\_of\_top**.value()));  
 y = Radious \* Math.*sin*(Math.*toRadians*((k \* 360) / *generate\_form*.**count\_of\_top**.value()));  
 *save\_tree*.addVertex(Start + (**int**) x, Start + (**int**) y, k + 1);  
 *//textBrowser.append(k + ") " + x + " " + y);* **textBrowser**.append(k + **") "** + x + **" "** + y);  
 **textBrowser**.append(**"Угол: "** + (k \* 180) / *generate\_form*.**count\_of\_top**.value());  
  
 }  
 }  
  
  
  
 **int** count = 0;  
 **while** (count < *generate\_form*.**count\_of\_edges**.value()) {  
 **int** from = 1 + (abs(random\_int.nextInt()) % (*generate\_form*.**count\_of\_top**.value()));  
 **int** to = 1 + (abs(random\_int.nextInt()) % (*generate\_form*.**count\_of\_top**.value()));  
 **if** (*save\_tree*.add\_edge(from, to))  
 count++;  
 **if**(*generate\_form*.**without\_cycle**.isChecked() && *save\_tree*.have\_cycle() */\*save\_tree.have\_cycle()\*/*)  
 {  
 **textBrowser**.append(**" Есть цикл. Удаляем ребро между "** + Integer.*toString*(from) + **" и "**+ Integer.*toString*(to));  
 *save\_tree*.dellete\_edge(from,to);  
 count--;  
 }  
 **else** {  
 **if**(*generate\_form*.**without\_cycle**.isChecked())  
 **textBrowser**.append(**" Цикла нет. Добавлено ребро "** + Integer.*toString*(from) + **" и "**+ Integer.*toString*(to));  
 }  
 }  
 */\* if(generate\_form.without\_cycle.isChecked())  
 textBrowser.append("Должно быть без цикла");  
 else  
 textBrowser.append("Возможен цикл");  
  
 if(save\_tree.have\_cycle())  
 textBrowser.append(" и цикл есть.");  
 else  
 textBrowser.append(" и цикла нет.");\*/* }**while**(*generate\_form*.**without\_cycle**.isCheckable() && *save\_tree*.have\_cycle());  
  
 **textBrowser**.append(**"Граф сгенерирован."**);  
 **for**(**int** i=0;i<*save\_tree*.**vertices**.size();i++) {  
 *save\_tree*.**vertices**.get(i).**color** = **'w'**;  
 **for** (**int** j = 0; j < *save\_tree*.**vertices**.get(i).**edg**.size(); j++) {  
 *save\_tree*.**vertices**.get(i).**edg**.get(j).**used** = **false**;  
 *save\_tree*.**vertices**.get(i).**edg**.get(j).**type** = **'n'**;  
 }  
 }  
 **step**.setEnabled(**true**);  
 **clear**.setEnabled(**true**);  
  
 *// распознование, является ли дерево цикличным или нет* }  
}

**Ui\_AddFile:**

*/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
 \*\* Form generated from reading ui file 'AddFileForm.jui'  
 \*\*  
 \*\* Created by: Qt User Interface Compiler version 4.8.6  
 \*\*  
 \*\* WARNING! All changes made in this file will be lost when recompiling ui file!  
 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
  
  
  
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
 \*\* Form generated from reading ui file 'AddFileForm.jui'  
 \*\*  
 \*\* Created by: Qt User Interface Compiler version 4.8.6  
 \*\*  
 \*\* WARNING! All changes made in this file will be lost when recompiling ui file!  
 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/***import** com.trolltech.qt.core.\*;  
**import** com.trolltech.qt.gui.\*;  
  
**public class** Ui\_AddFileForm **implements** com.trolltech.qt.QUiForm<QWidget>  
{  
 **public** QWidget **widget**;  
 **public** QVBoxLayout **verticalLayout\_3**;  
 **public** QVBoxLayout **verticalLayout\_2**;  
 **public** QVBoxLayout **verticalLayout**;  
 **public** QHBoxLayout **horizontalLayout**;  
 **public** QLabel **label**;  
 **public** QSpacerItem **horizontalSpacer**;  
 **public** QLineEdit **lineEdit**;  
 **public** QHBoxLayout **horizontalLayout\_2**;  
 **public static** QPushButton *Ok*;  
 **public** QPushButton **Cansel**;  
 **public static** String *inputString*;  
  
 **public** Ui\_AddFileForm() { **super**(); }  
  
 **public void** setupUi(QWidget Form)  
 {  
 Form.setObjectName(**"Form"**);  
 Form.resize(**new** QSize(236, 87).expandedTo(Form.minimumSizeHint()));  
 **widget** = **new** QWidget(Form);  
 **widget**.setObjectName(**"widget"**);  
 **widget**.setGeometry(**new** QRect(0, 0, 234, 85));  
 **verticalLayout\_3** = **new** QVBoxLayout(**widget**);  
 **verticalLayout\_3**.setObjectName(**"verticalLayout\_3"**);  
 **verticalLayout\_2** = **new** QVBoxLayout();  
 **verticalLayout\_2**.setObjectName(**"verticalLayout\_2"**);  
 **verticalLayout** = **new** QVBoxLayout();  
 **verticalLayout**.setObjectName(**"verticalLayout"**);  
 **horizontalLayout** = **new** QHBoxLayout();  
 **horizontalLayout**.setObjectName(**"horizontalLayout"**);  
 **label** = **new** QLabel(**widget**);  
 **label**.setObjectName(**"label"**);  
  
 **horizontalLayout**.addWidget(**label**);  
  
 **horizontalSpacer** = **new** QSpacerItem(40, 20, com.trolltech.qt.gui.QSizePolicy.Policy.***Expanding***, com.trolltech.qt.gui.QSizePolicy.Policy.***Minimum***);  
  
 **horizontalLayout**.addItem(**horizontalSpacer**);  
  
  
 **verticalLayout**.addLayout(**horizontalLayout**);  
  
 **lineEdit** = **new** QLineEdit(**widget**);  
 **lineEdit**.setObjectName(**"lineEdit"**);  
  
 **verticalLayout**.addWidget(**lineEdit**);  
  
  
 **verticalLayout\_2**.addLayout(**verticalLayout**);  
  
  
 **verticalLayout\_3**.addLayout(**verticalLayout\_2**);  
  
 **horizontalLayout\_2** = **new** QHBoxLayout();  
 **horizontalLayout\_2**.setObjectName(**"horizontalLayout\_2"**);  
 *Ok* = **new** QPushButton(**widget**);  
 *Ok*.setObjectName(**"Ok"**);  
  
 **horizontalLayout\_2**.addWidget(*Ok*);  
  
 **Cansel** = **new** QPushButton(**widget**);  
 **Cansel**.setObjectName(**"Cansel"**);  
  
 **horizontalLayout\_2**.addWidget(**Cansel**);  
  
  
 **verticalLayout\_3**.addLayout(**horizontalLayout\_2**);  
  
 retranslateUi(Form);  
  
 **if**(**lineEdit**.isModified())  
 {  
 **lineEdit**.text();  
 }  
 **Cansel**.**clicked**.connect(Form, **"close()"**);  
 *//if(Ok.clicked.)inputString = lineEdit.text();  
 Ok*.**clicked**.connect(**this**, **"read\_line()"**);  
 *Ok*.**clicked**.connect(Form, **"close()"**);  
  
 Form.connectSlotsByName();  
 } *// setupUi* **void** retranslateUi(QWidget Form)  
 {  
 Form.setWindowTitle(com.trolltech.qt.core.QCoreApplication.*translate*(**"Form"**, **"Form"**, **null**));  
 **label**.setText(com.trolltech.qt.core.QCoreApplication.*translate*(**"Form"**,**"Введите имя файла( без расширения)"** */\*"\u0412\u0432\u0435\u0434\u0438\u0442\u0435 \u043f\u0443\u0442\u044c \u043a \u0442\u0435\u043a\u0441\u0442\u043e\u0432\u043e\u043c\u0443 \u0444\u0430\u0439\u043b\u0443:"\*/*, **null**));  
 *Ok*.setText(com.trolltech.qt.core.QCoreApplication.*translate*(**"Form"**, **"\u041e\u043a"**, **null**));  
 **Cansel**.setText(com.trolltech.qt.core.QCoreApplication.*translate*(**"Form"**, **"\u041e\u0442\u043c\u0435\u043d\u0430"**, **null**));  
 } *// retranslateUi* **void** read\_line()  
 {  
 *inputString* = **lineEdit**.text();  
 }  
  
}

**Ui\_GenerateForm:**

*/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
 \*\* Form generated from reading ui file 'generate\_form.jui'  
 \*\*  
 \*\* Created by: Qt User Interface Compiler version 4.8.6  
 \*\*  
 \*\* WARNING! All changes made in this file will be lost when recompiling ui file!  
 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/***import** com.trolltech.qt.core.\*;  
**import** com.trolltech.qt.gui.\*;  
  
**public class** Ui\_GenerateForm **implements** com.trolltech.qt.QUiForm<QWidget>  
{  
 **public** QWidget layoutWidget;  
 **public** QVBoxLayout verticalLayout;  
 **public** QLabel label;  
 **public** QLabel label\_5;  
 **public** QHBoxLayout horizontalLayout\_3;  
 **public** QLabel label\_2;  
 **public** QSlider count\_of\_top;  
 **public** QLabel label\_4;  
 **public** QLabel label\_6;  
 **public** QHBoxLayout horizontalLayout\_2;  
 **public** QLabel label\_3;  
 **public** QSlider count\_of\_edges;  
 **public** QLabel label\_7;  
 **public** QCheckBox without\_cycle;  
 **public** QCheckBox round\_painting;  
 **public** QHBoxLayout horizontalLayout;  
 **public** QPushButton Ok;  
 **public** QPushButton Cansel;  
  
 **public** Ui\_GenerateForm() { **super**(); }  
  
 **public void** setupUi(QWidget Form)  
 {  
 Form.setObjectName(**"Form"**);  
 Form.resize(**new** QSize(241, 200).expandedTo(Form.minimumSizeHint()));  
 layoutWidget = **new** QWidget(Form);  
 layoutWidget.setObjectName(**"layoutWidget"**);  
 layoutWidget.setGeometry(**new** QRect(20, 0, 201, 190));  
 verticalLayout = **new** QVBoxLayout(layoutWidget);  
 verticalLayout.setObjectName(**"verticalLayout"**);  
 label = **new** QLabel(layoutWidget);  
 label.setObjectName(**"label"**);  
  
 verticalLayout.addWidget(label);  
  
 label\_5 = **new** QLabel(layoutWidget);  
 label\_5.setObjectName(**"label\_5"**);  
  
 verticalLayout.addWidget(label\_5);  
  
 horizontalLayout\_3 = **new** QHBoxLayout();  
 horizontalLayout\_3.setObjectName(**"horizontalLayout\_3"**);  
 label\_2 = **new** QLabel(layoutWidget);  
 label\_2.setObjectName(**"label\_2"**);  
  
 horizontalLayout\_3.addWidget(label\_2);  
  
 count\_of\_top = **new** QSlider(layoutWidget);  
 count\_of\_top.setObjectName(**"count\_of\_top"**);  
 count\_of\_top.setOrientation(com.trolltech.qt.core.Qt.Orientation.Horizontal);  
 count\_of\_top.setMaximum(50);  
 count\_of\_top.setValue(10);  
  
 horizontalLayout\_3.addWidget(count\_of\_top);  
  
 label\_4 = **new** QLabel(layoutWidget);  
 label\_4.setObjectName(**"label\_4"**);  
  
 horizontalLayout\_3.addWidget(label\_4);  
  
  
 verticalLayout.addLayout(horizontalLayout\_3);  
  
 label\_6 = **new** QLabel(layoutWidget);  
 label\_6.setObjectName(**"label\_6"**);  
  
 verticalLayout.addWidget(label\_6);  
  
 horizontalLayout\_2 = **new** QHBoxLayout();  
 horizontalLayout\_2.setObjectName(**"horizontalLayout\_2"**);  
 label\_3 = **new** QLabel(layoutWidget);  
 label\_3.setObjectName(**"label\_3"**);  
  
 horizontalLayout\_2.addWidget(label\_3);  
  
 count\_of\_edges = **new** QSlider(layoutWidget);  
 count\_of\_edges.setObjectName(**"count\_of\_edges"**);  
 count\_of\_edges.setOrientation(com.trolltech.qt.core.Qt.Orientation.Horizontal);  
 count\_of\_edges.setMaximum(50);  
 count\_of\_edges.setValue(10);  
  
 horizontalLayout\_2.addWidget(count\_of\_edges);  
  
 label\_7 = **new** QLabel(layoutWidget);  
 label\_7.setObjectName(**"label\_7"**);  
  
 horizontalLayout\_2.addWidget(label\_7);  
  
  
 verticalLayout.addLayout(horizontalLayout\_2);  
  
 without\_cycle = **new** QCheckBox(layoutWidget);  
 without\_cycle.setObjectName(**"without\_cycle"**);  
  
 verticalLayout.addWidget(without\_cycle);  
  
 round\_painting = **new** QCheckBox(layoutWidget);  
 round\_painting.setObjectName(**"round\_painting"**);  
  
 verticalLayout.addWidget(round\_painting);  
  
 horizontalLayout = **new** QHBoxLayout();  
 horizontalLayout.setObjectName(**"horizontalLayout"**);  
 Ok = **new** QPushButton(layoutWidget);  
 Ok.setObjectName(**"Ok"**);  
  
 horizontalLayout.addWidget(Ok);  
  
 Cansel = **new** QPushButton(layoutWidget);  
 Cansel.setObjectName(**"Cansel"**);  
  
 horizontalLayout.addWidget(Cansel);  
  
  
 verticalLayout.addLayout(horizontalLayout);  
  
 retranslateUi(Form);  
  
 *//* count\_of\_edges.valueChanged.connect(**this**, **"setCountOfTop()"**);  
 count\_of\_top.valueChanged.connect(**this**, **"setCountOfEdges()"**);  
 Cansel.clicked.connect(Form, **"close()"**);  
 *//if(Ok.clicked.)inputString = lineEdit.text();* Ok.clicked.connect(Form, **"close()"**);  
 *//* Form.connectSlotsByName();  
 } *// setupUi* **private void** setCountOfTop()  
 {  
 **if**( count\_of\_top.value() < count\_of\_edges.value() )  
 {  
 count\_of\_top.setValue(count\_of\_edges.value());  
 }  
  
 }  
 **private void** setCountOfEdges()  
 {  
  
 **if**( count\_of\_top.value() < count\_of\_edges.value() )  
 {  
 count\_of\_edges.setValue(count\_of\_top.value());  
 }  
 *// count\_of\_top.setValue(curr\_value +1);* }  
 **void** retranslateUi(QWidget Form)  
 {  
 Form.setWindowTitle(com.trolltech.qt.core.QCoreApplication.translate(**"Form"**, **"Form"**, **null**));  
 label.setText(com.trolltech.qt.core.QCoreApplication.translate(**"Form"**, **"\u0420\u0430\u0437\u043c\u0435\u0440 \u0433\u0440\u0430\u0444\u0430:"**, **null**));  
 label\_5.setText(com.trolltech.qt.core.QCoreApplication.translate(**"Form"**, **"\u041a\u043e\u043b\u0438\u0447\u0435\u0441\u0442\u0432\u043e \u0432\u0435\u0440\u0448\u0438\u043d:"**, **null**));  
 label\_2.setText(com.trolltech.qt.core.QCoreApplication.translate(**"Form"**, **"0"**, **null**));  
 label\_4.setText(com.trolltech.qt.core.QCoreApplication.translate(**"Form"**, **"50"**, **null**));  
 label\_6.setText(com.trolltech.qt.core.QCoreApplication.translate(**"Form"**, **"\u041a\u043e\u043b\u0438\u0447\u0435\u0441\u0442\u0432\u043e \u0440\u0451\u0431\u0435\u0440:"**, **null**));  
 label\_3.setText(com.trolltech.qt.core.QCoreApplication.translate(**"Form"**, **"0"**, **null**));  
 label\_7.setText(com.trolltech.qt.core.QCoreApplication.translate(**"Form"**, **"50"**, **null**));  
 without\_cycle.setText(com.trolltech.qt.core.QCoreApplication.translate(**"Form"**, **"\u0413\u0440\u0430\u0444 \u0441\u043e\u0437\u0434\u0430\u0451\u0442\u0441\u044f \u0431\u0435\u0437 \u0446\u0438\u043a\u043b\u043e\u0432"**, **null**));  
 round\_painting.setText(com.trolltech.qt.core.QCoreApplication.translate(**"Form"**, **"\u0413\u0440\u0430\u0444 \u0440\u0438\u0441\u0443\u0435\u0442\u0441\u044f \u043f\u043e \u043a\u0440\u0443\u0433\u0443"**, **null**));  
 Ok.setText(com.trolltech.qt.core.QCoreApplication.translate(**"Form"**, **"\u041e\u043a"**, **null**));  
 Cansel.setText(com.trolltech.qt.core.QCoreApplication.translate(**"Form"**, **"\u041e\u0442\u043c\u0435\u043d\u0430"**, **null**));  
 } *// retranslateUi*}

**Ui\_OptionForm:**

*/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
 \*\* Form generated from reading ui file 'Option.jui'  
 \*\*  
 \*\* Created by: Qt User Interface Compiler version 4.8.6  
 \*\*  
 \*\* WARNING! All changes made in this file will be lost when recompiling ui file!  
 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/***import** com.trolltech.qt.core.\*;  
**import** com.trolltech.qt.gui.\*;  
  
**public class** Ui\_OptionForm **implements** com.trolltech.qt.QUiForm<QWidget>  
{  
 **public static boolean** first = **true**;  
 **public** QWidget layoutWidget;  
 **public** QVBoxLayout verticalLayout\_3;  
 **public** QHBoxLayout horizontalLayout\_3;  
 **public** QVBoxLayout verticalLayout\_2;  
 **public** QHBoxLayout horizontalLayout\_2;  
 **public** QLabel label;  
 **public** QSpacerItem horizontalSpacer;  
 **public** QSpacerItem horizontalSpacer\_2;  
 **public static** QCheckBox checkBox;  
 **public** QSpacerItem verticalSpacer;  
 **public** QVBoxLayout verticalLayout;  
 **public** QLabel label\_2;  
 **public** QHBoxLayout horizontalLayout;  
 **public static** QRadioButton Green\_yellow;  
 **public static** QRadioButton White\_dark;  
 **public** QSpacerItem verticalSpacer\_2;  
 **public** QHBoxLayout horizontalLayout\_4;  
 **public** QDialogButtonBox buttonBox;  
 **public** QSpacerItem horizontalSpacer\_3;  
  
 **public** Ui\_OptionForm() { **super**(); }  
  
 **public void** setupUi(QWidget Form)  
 {  
 Form.setObjectName(**"Form"**);  
 Form.resize(**new** QSize(269, 231).expandedTo(Form.minimumSizeHint()));  
 layoutWidget = **new** QWidget(Form);  
 layoutWidget.setObjectName(**"layoutWidget"**);  
 layoutWidget.setGeometry(**new** QRect(30, 30, 213, 171));  
 verticalLayout\_3 = **new** QVBoxLayout(layoutWidget);  
 verticalLayout\_3.setObjectName(**"verticalLayout\_3"**);  
 horizontalLayout\_3 = **new** QHBoxLayout();  
 horizontalLayout\_3.setObjectName(**"horizontalLayout\_3"**);  
 verticalLayout\_2 = **new** QVBoxLayout();  
 verticalLayout\_2.setObjectName(**"verticalLayout\_2"**);  
 horizontalLayout\_2 = **new** QHBoxLayout();  
 horizontalLayout\_2.setObjectName(**"horizontalLayout\_2"**);  
 label = **new** QLabel(layoutWidget);  
 label.setObjectName(**"label"**);  
  
 horizontalLayout\_2.addWidget(label);  
  
 horizontalSpacer = **new** QSpacerItem(68, 20, com.trolltech.qt.gui.QSizePolicy.Policy.Expanding, com.trolltech.qt.gui.QSizePolicy.Policy.Minimum);  
  
 horizontalLayout\_2.addItem(horizontalSpacer);  
  
  
 verticalLayout\_2.addLayout(horizontalLayout\_2);  
  
  
 horizontalLayout\_3.addLayout(verticalLayout\_2);  
  
 horizontalSpacer\_2 = **new** QSpacerItem(48, 20, com.trolltech.qt.gui.QSizePolicy.Policy.Expanding, com.trolltech.qt.gui.QSizePolicy.Policy.Minimum);  
  
 horizontalLayout\_3.addItem(horizontalSpacer\_2);  
  
  
 verticalLayout\_3.addLayout(horizontalLayout\_3);  
  
 **if**(first) {  
 checkBox = **new** QCheckBox(layoutWidget);  
 checkBox.click();  
  
 }  
 checkBox.setObjectName(**"checkBox"**);  
  
 verticalLayout\_3.addWidget(checkBox);  
  
 verticalSpacer = **new** QSpacerItem(20, 18, com.trolltech.qt.gui.QSizePolicy.Policy.Minimum, com.trolltech.qt.gui.QSizePolicy.Policy.Expanding);  
  
 verticalLayout\_3.addItem(verticalSpacer);  
  
 verticalLayout = **new** QVBoxLayout();  
 verticalLayout.setObjectName(**"verticalLayout"**);  
 label\_2 = **new** QLabel(layoutWidget);  
 label\_2.setObjectName(**"label\_2"**);  
  
 verticalLayout.addWidget(label\_2);  
  
 horizontalLayout = **new** QHBoxLayout();  
 horizontalLayout.setObjectName(**"horizontalLayout"**);  
 **if**(first) {  
 Green\_yellow = **new** QRadioButton(layoutWidget);  
 Green\_yellow.setEnabled(**true**);  
 Green\_yellow.setAcceptDrops(**false**);  
 }  
 Green\_yellow.setObjectName(**"Green\_Red"**);  
  
 horizontalLayout.addWidget(Green\_yellow);  
  
 **if**(first) {  
 White\_dark = **new** QRadioButton(layoutWidget);  
 White\_dark.setMouseTracking(**true**);  
 White\_dark.click();  
 }  
 White\_dark.setObjectName(**"White\_dark"**);  
  
 horizontalLayout.addWidget(White\_dark);  
  
  
 verticalLayout.addLayout(horizontalLayout);  
  
  
 verticalLayout\_3.addLayout(verticalLayout);  
  
 verticalSpacer\_2 = **new** QSpacerItem(20, 13, com.trolltech.qt.gui.QSizePolicy.Policy.Minimum, com.trolltech.qt.gui.QSizePolicy.Policy.Expanding);  
  
 verticalLayout\_3.addItem(verticalSpacer\_2);  
  
 horizontalLayout\_4 = **new** QHBoxLayout();  
 horizontalLayout\_4.setObjectName(**"horizontalLayout\_4"**);  
  
 buttonBox = **new** QDialogButtonBox(layoutWidget);  
 buttonBox.setStandardButtons(com.trolltech.qt.gui.QDialogButtonBox.StandardButton.createQFlags(com.trolltech.qt.gui.QDialogButtonBox.StandardButton.Ok));  
  
 buttonBox.setObjectName(**"buttonBox"**);  
  
 horizontalLayout\_4.addWidget(buttonBox);  
  
 horizontalSpacer\_3 = **new** QSpacerItem(128, 20, com.trolltech.qt.gui.QSizePolicy.Policy.Expanding, com.trolltech.qt.gui.QSizePolicy.Policy.Minimum);  
  
 horizontalLayout\_4.addItem(horizontalSpacer\_3);  
  
  
 verticalLayout\_3.addLayout(horizontalLayout\_4);  
  
 retranslateUi(Form);  
 first =**false**;  
 *// checkBox.clicked.connect(this, "vis()");  
 //checkBox.isChecked();* buttonBox.clicked.connect(Form, **"close()"**);  
  
 Form.connectSlotsByName();  
 } *// setupUi* **void** retranslateUi(QWidget Form)  
 {  
 Form.setWindowTitle(com.trolltech.qt.core.QCoreApplication.translate(**"Form"**, **"Настройки"**, **null**));  
 label.setText(com.trolltech.qt.core.QCoreApplication.translate(**"Form"**, **"\u041d\u0430\u0441\u0442\u0440\u043e\u0439\u043a\u0438:"**, **null**));  
 checkBox.setText(com.trolltech.qt.core.QCoreApplication.translate(**"Form"**, **"С визуализацией"**, **null**));  
 label\_2.setText(com.trolltech.qt.core.QCoreApplication.translate(**"Form"**, **"\u0420\u0430\u0441\u043a\u0440\u0430\u0441\u043a\u0430 \u0433\u0440\u0430\u0444\u043e\u0432 \u043f\u0440\u0438 \u0432\u0438\u0437\u0443\u0430\u043b\u0438\u0437\u0430\u0446\u0438\u0438:"**, **null**));  
 Green\_yellow.setText(com.trolltech.qt.core.QCoreApplication.translate(**"Form"**, **"Жёлто-зелёный"**, **null**));  
 White\_dark.setText(com.trolltech.qt.core.QCoreApplication.translate(**"Form"**, **"\u0427\u0435\u0440\u043d\u043e-\u0431\u0435\u043b\u0430\u044f"**, **null**));  
 } *// retranslateUi  
  
 /\*void vis()  
 {  
 visualisation = !visualisation;  
 }\*/*}

**Тест кейсы:**

Конечная версия:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | Работа алгоритма | Визуализация алгоритма  Гр Рез Алг | Визуализация интерфейса | Работа кнопок | Работа алгоритма различных видов ребер | Визуализация различных видов ребер | Способ задания графов  Г Ф |
| **Тест 1** | **+** | + + + | **+** |  | + | + | + + |
| **Тест 2** | + | + + + | **+** |  | + | + | + + |
| **Тест 3** | + | + + + | **+** |  | + | + | + + |
| **Тест 4** |  |  | **+** | + |  |  |  |
| **Тест 5** |  |  | **+** | + |  |  |  |
| **Тест 6** | + | + + + | **+** |  | + | + | + + |

Гр- граф ,Рез- результат, Алг- алгоритм, К – клавиатура, Ф – файл, Г – графически

Тест1:

1->2 , 1->3 , 2->4 , 5->1 ( обычный тест) ( Ожидаемый результат: 5,1,3,2,4)

4->1 , 4->2 , 4->3 , 5->3 ( Ожидаемый результат: 4,3,5,2,1)

2->1 , 2->3 ,4->2 , 5->2

Тест2:

Тест из спецификации (Ожидаемый результат: 1,2,3,4,5|1,3,2,4,5)

1->8 , 8->2 , 7->2 , 7->3 , 6->3 , 6->7 , 4->3 , 5->4 ( усложненные) (Ожидаемый результат: 6,7,5,4,3,1,8,2)

Тест3:

Ожидаемые результаты: Ошибки

1->2 , 2->3 , 3->1 ( циклический тест)

1->2 , 3->2 , 2->4 , 4->3

1->2 , 1->4 , 1->3 , 3->3 , 3->4

Тест4: Проверка работы всех кнопок на основе теста 1.1

Тест5: Проверка всех кнопок на основе теста 3.1

Тест6:

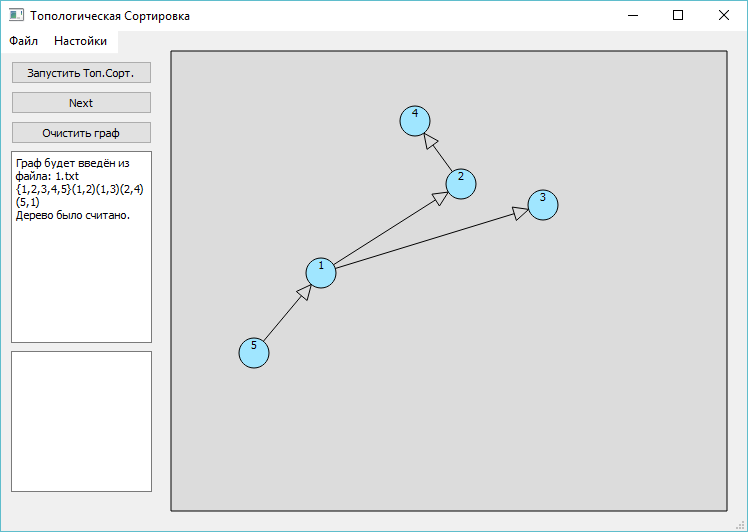
Ожидаемые результаты: Ошибки

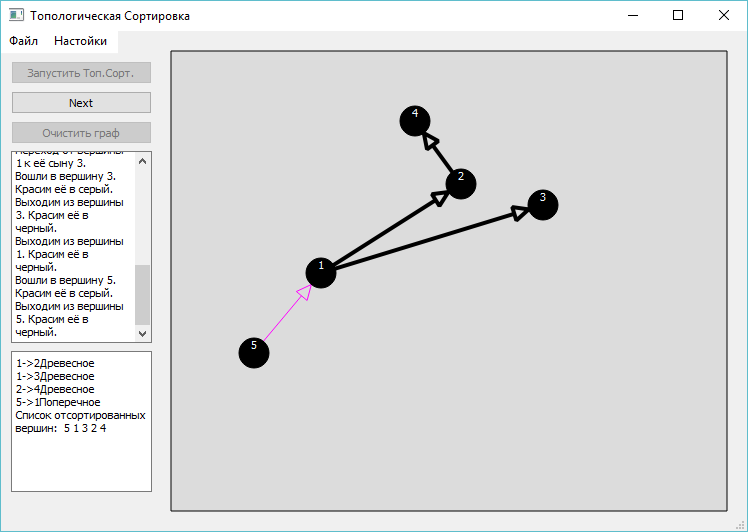
2->1 , 3->2 , 4->2 , 4->5 , 2->5 , 5->6 , 6->3 (усложненный циклический тест)

1->5 , 2->5 , 6->2 , 3->2 , 2->7 , 6->7 , 7->3 , 7->8 , 8->4

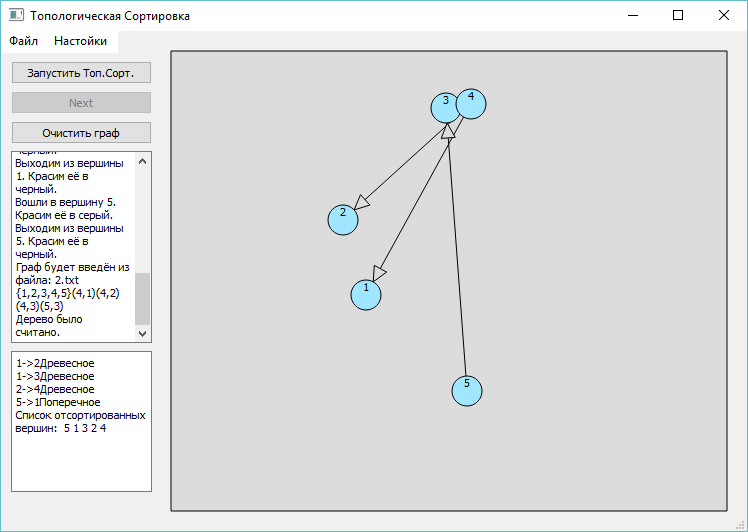
Тесты 1,2,3,6 (ввод из файла)

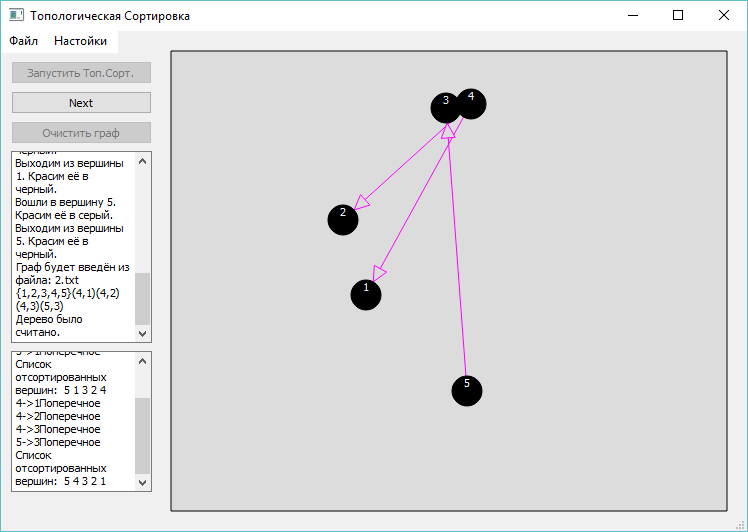
1.1



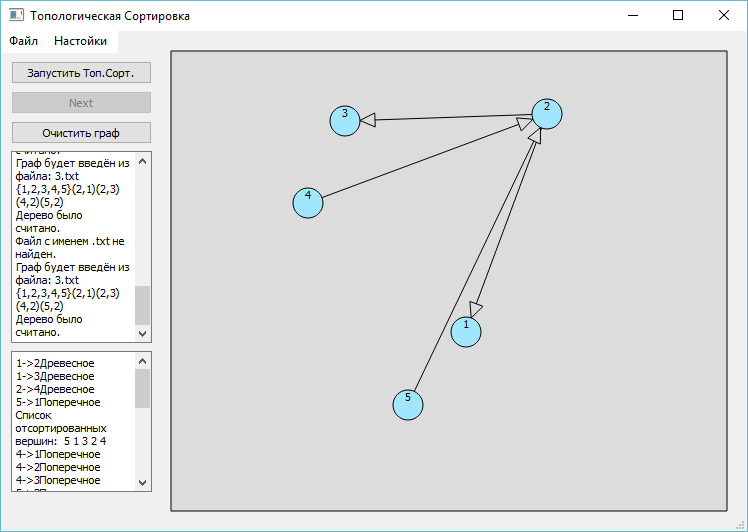


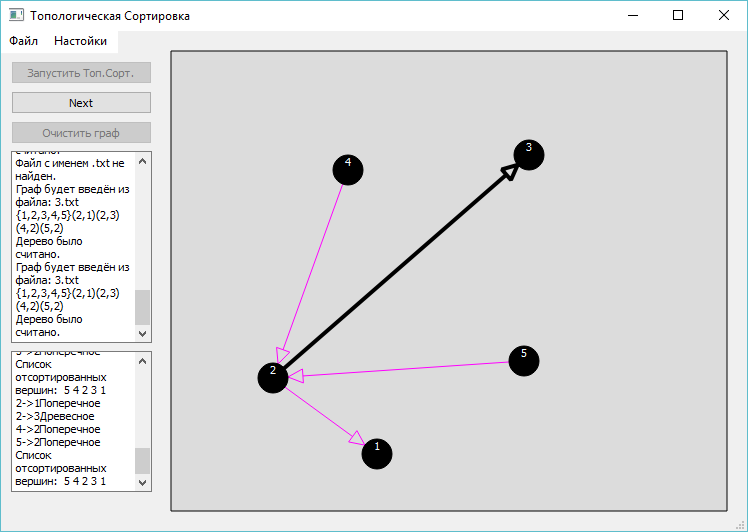
Тест 1.2:



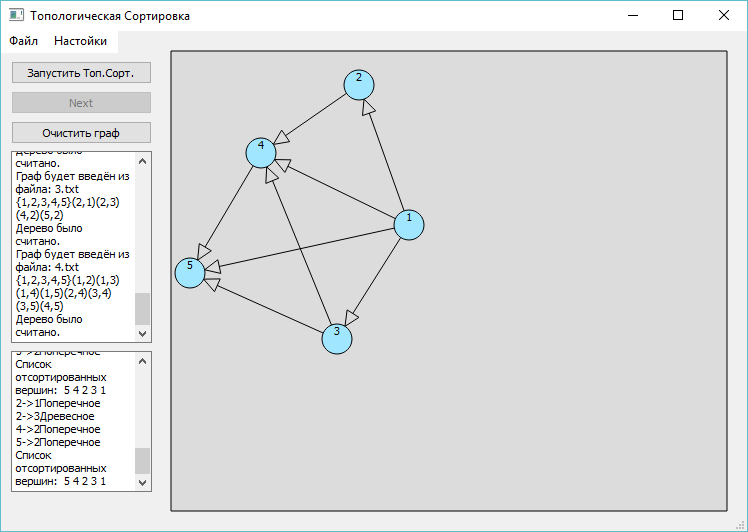


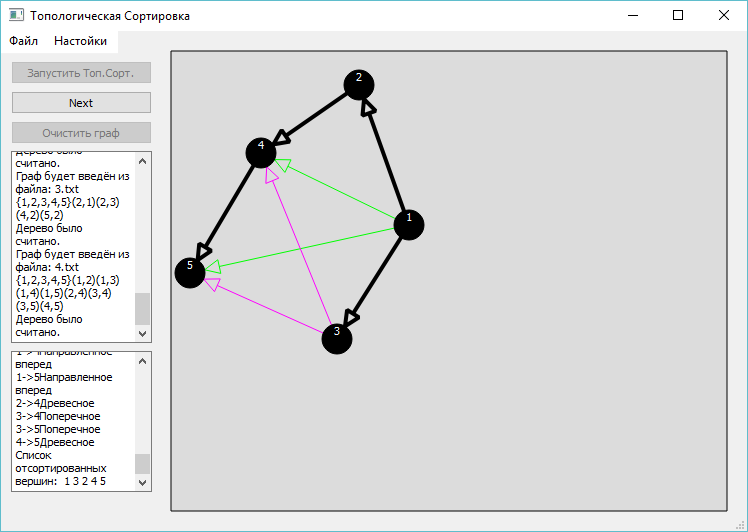
Тест 1.3:



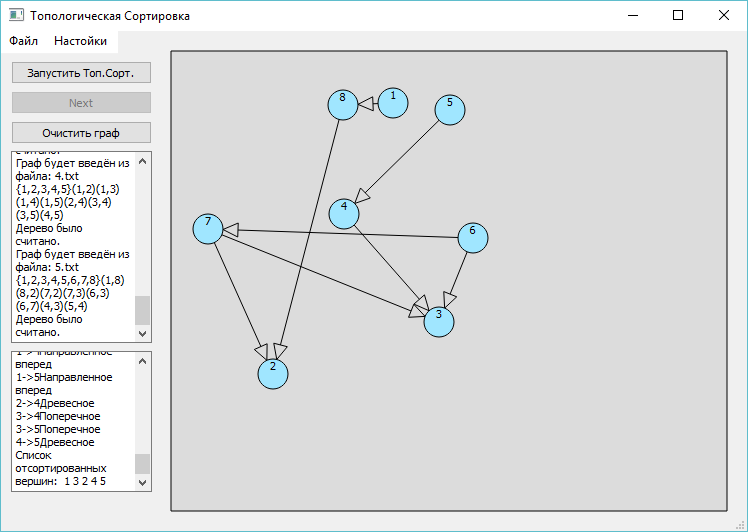


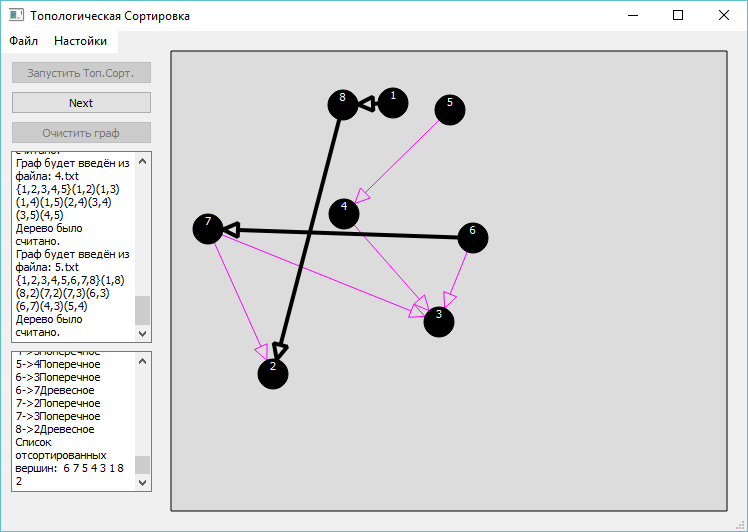
Тест 2.1



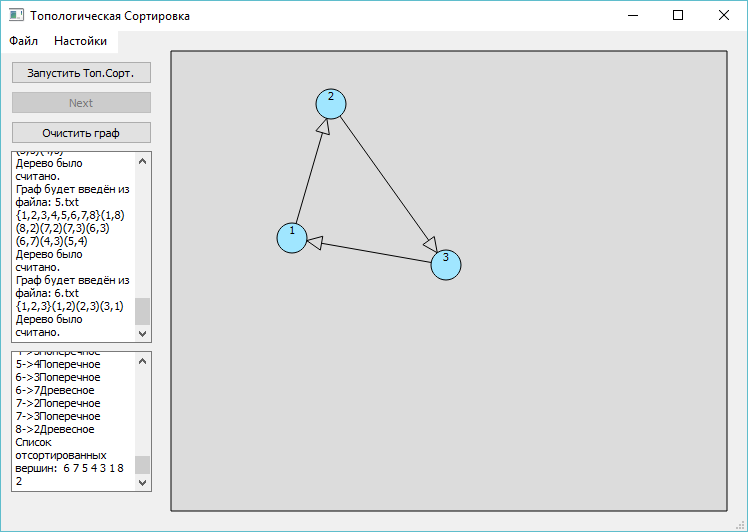


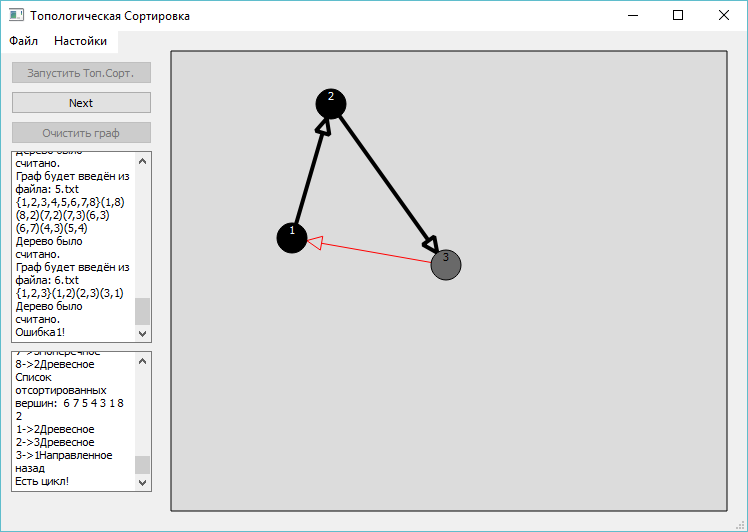
Тест 2.2



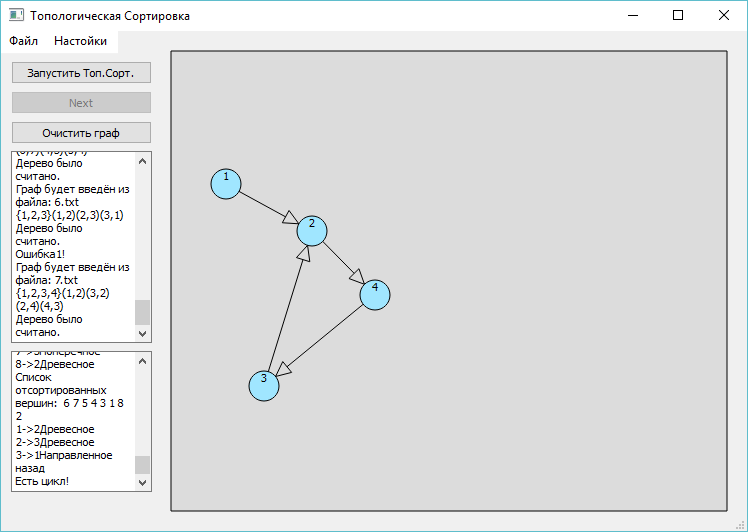


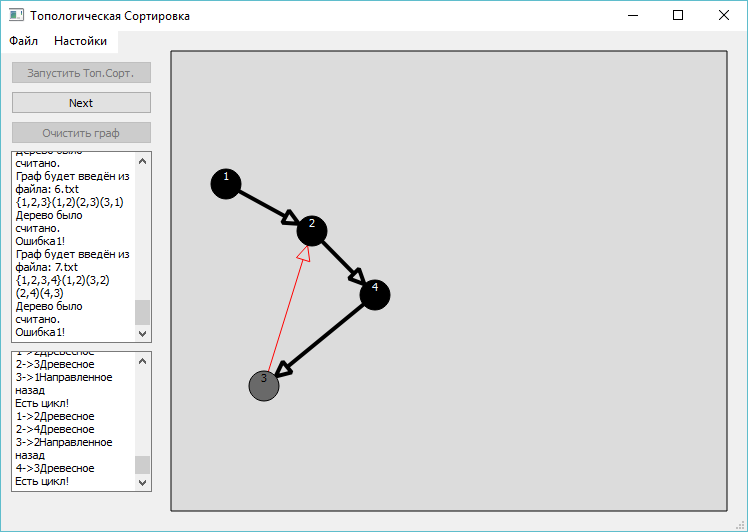
Тест 3.1



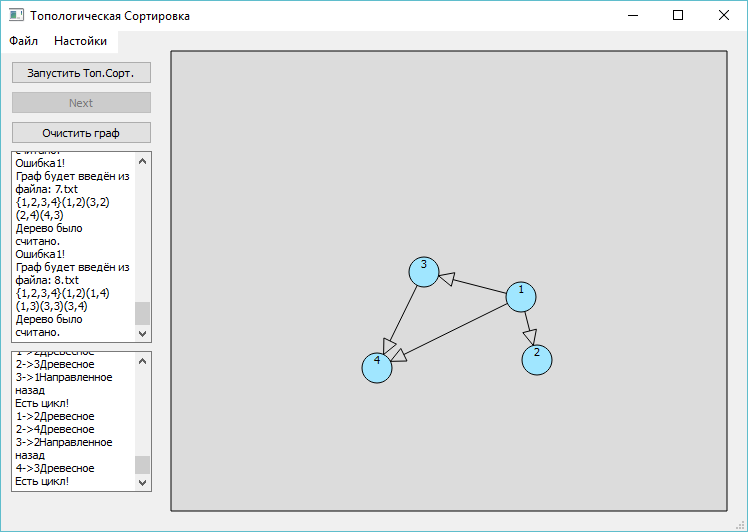


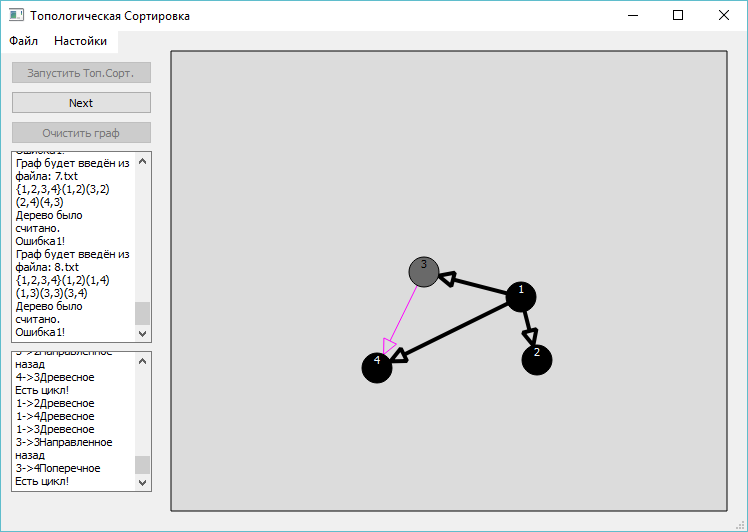
Тест 3.2



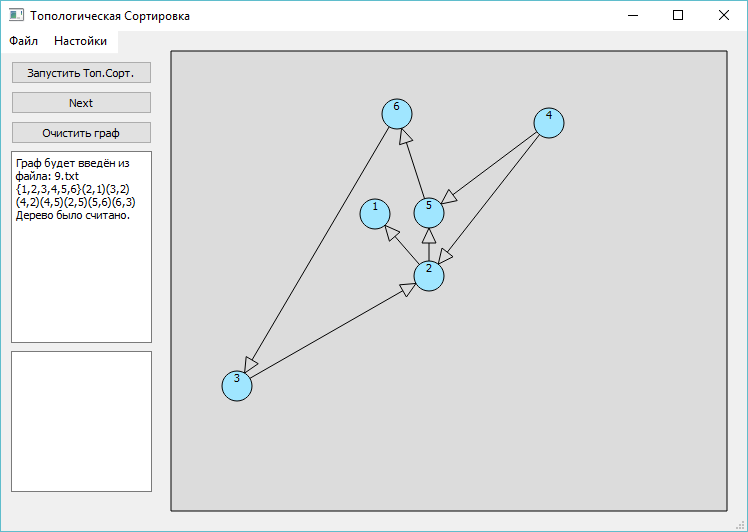


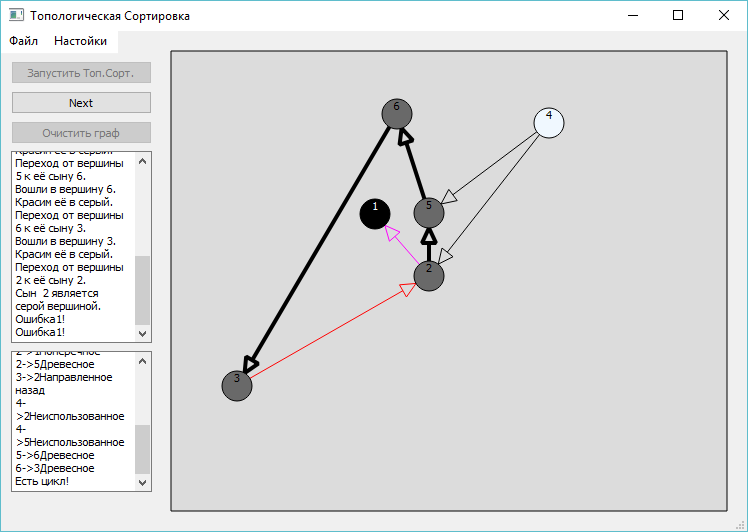
Тест 3.3



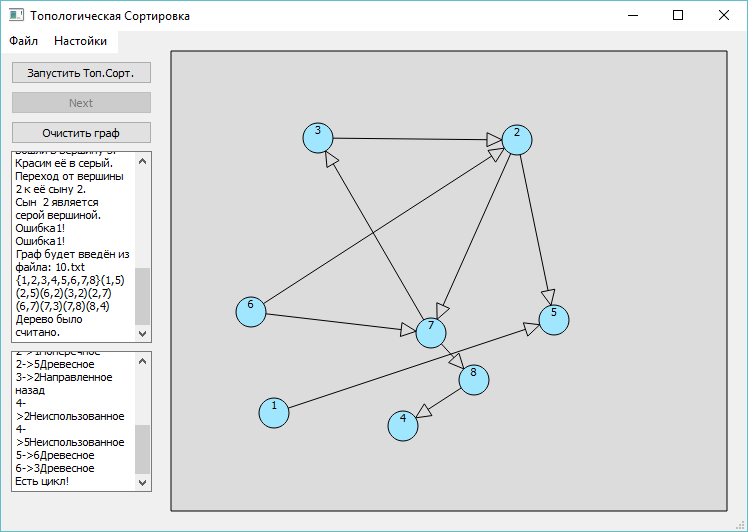


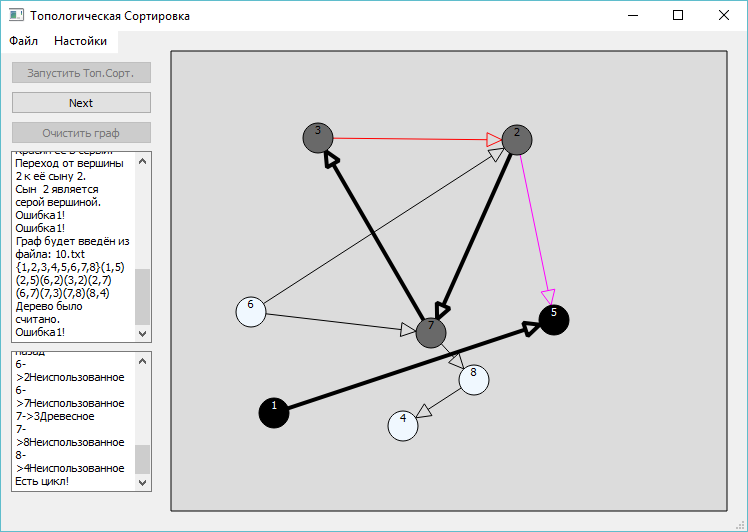
Тест 6.1





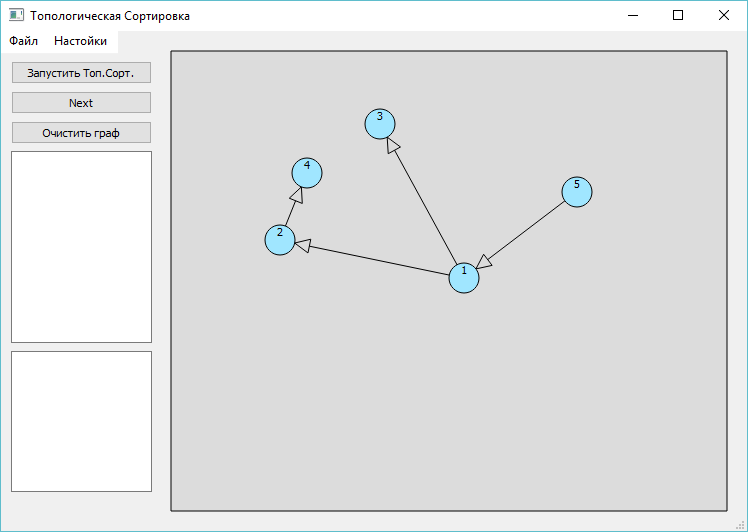
Тест 6.2



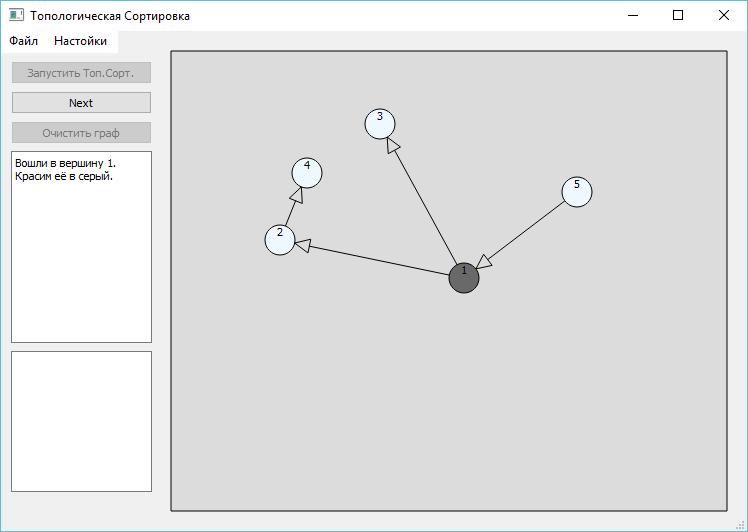


Тест 4:

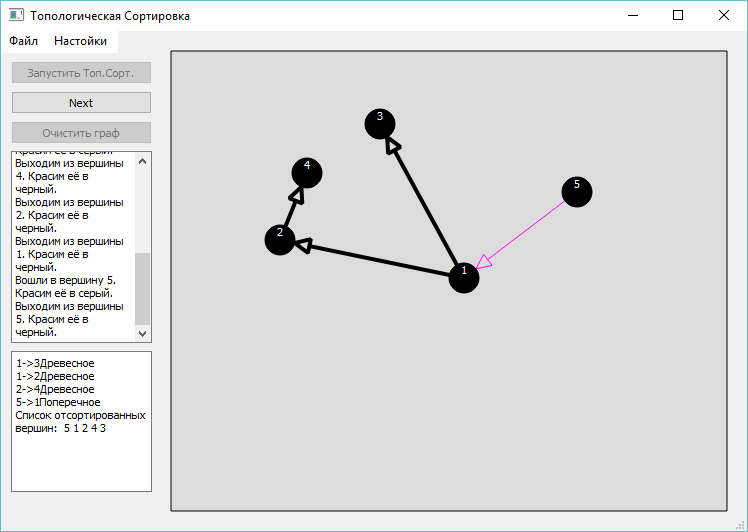
Ввели граф:



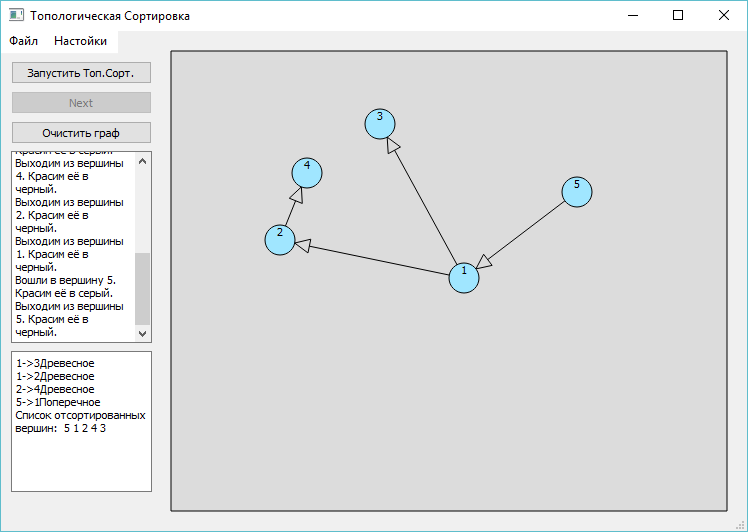
Нажатие кнопки Запустить Топ.Сорт.



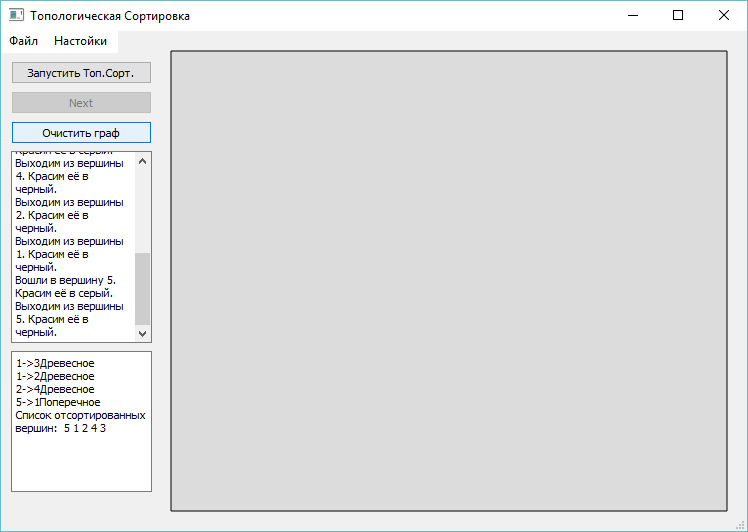
Нажимаем кнопку Next. До завершения работы алгоритмма:



Завершаем алгоритм кнопкной Next:



Нажатие кнопки Очистить Граф



**Тест кейсы:**

Первая версия:

Тест1: 1->2 , 1->3 , 2->4 , 5->1 ( обычный тест) ( Ожидаемый результат: 5,1,3,2,4)

4->1 , 4->2 , 4->3 , 5->3 ( Ожидаемый результат: 4,3,5,2,1)

Тест2: Тест из спецификации (Ожидаемый результат 1,3,2,4,5 | 1,2,3,4,5)

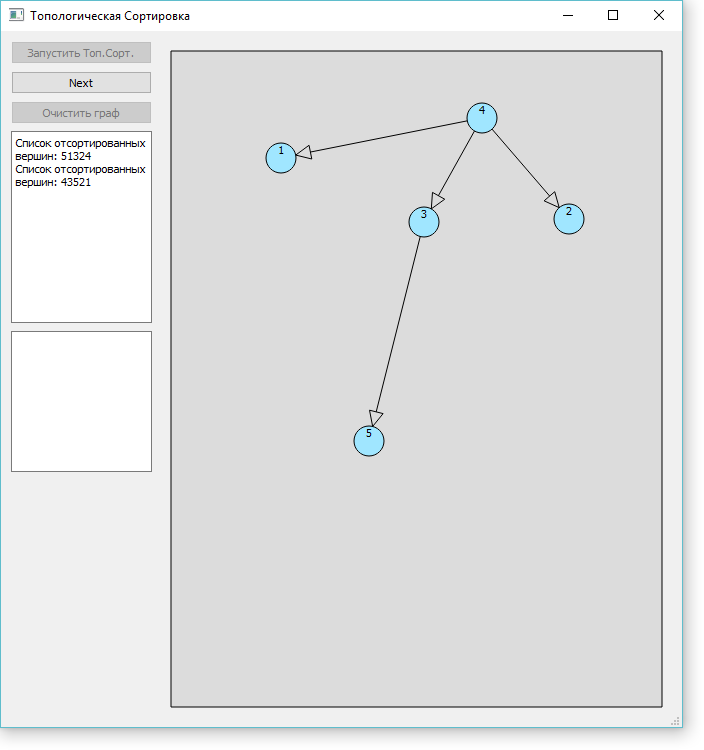
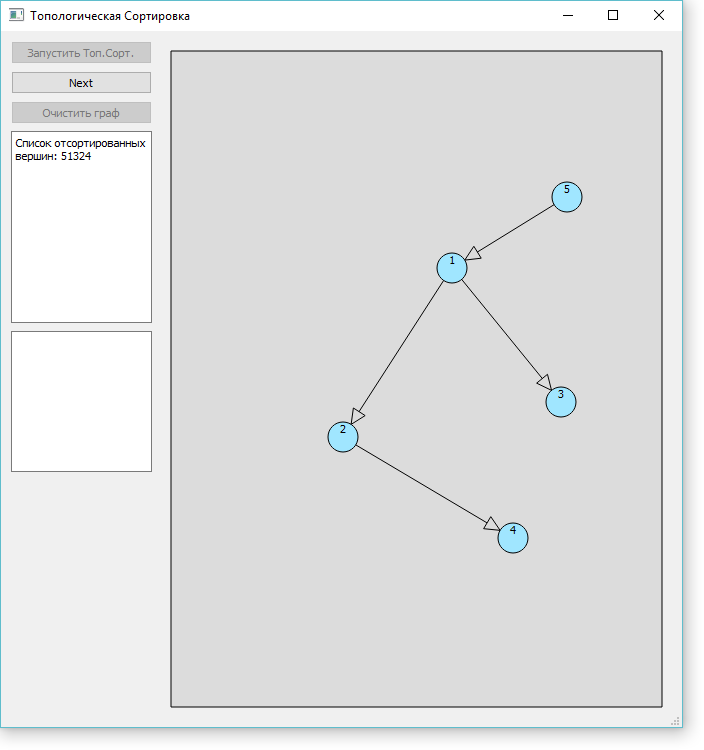
1->8 , 8->2 , 7->2 , 7->3 , 6->3 , 6->7 , 4->3 , 5->4 ( усложненные) ( Ожидаемый результат: 6,7,5,4,3,1,8,2)

Тест4: Проверка работы всех кнопок на основе теста 1.1

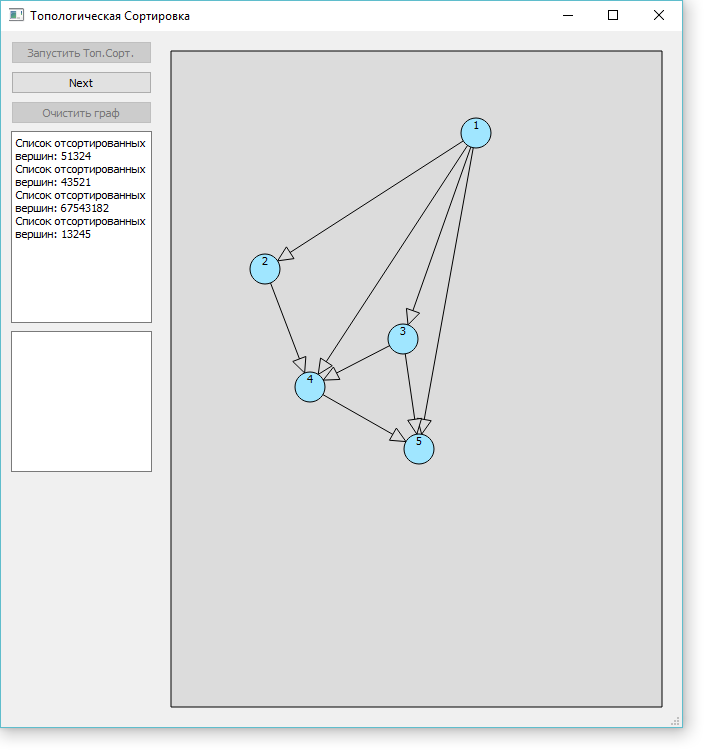
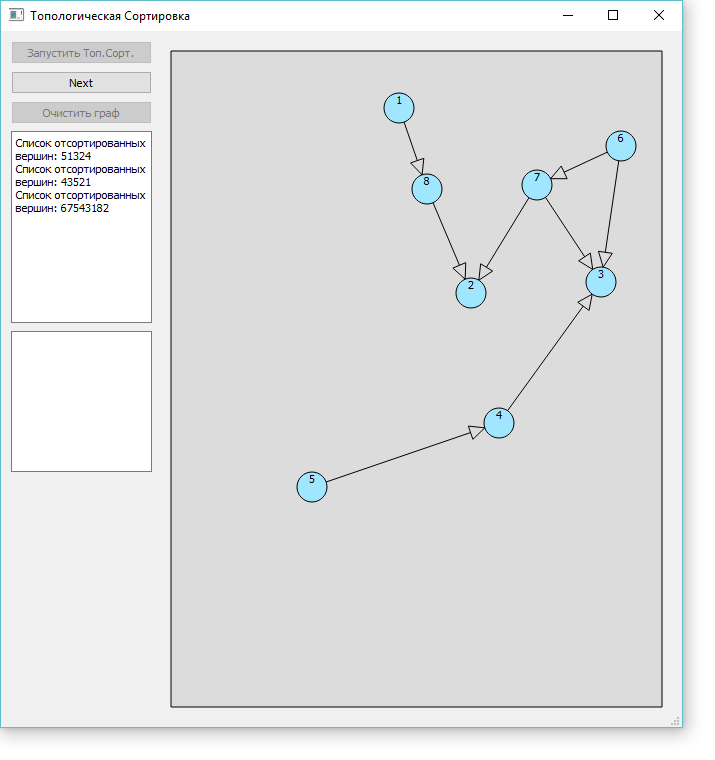
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Работа алгоритма | Визуализация алгоритма  Гр Рез Алг | Визуализация интерфейса | Работа кнопок | Способ задания графов  Г |
| **Тест 1** | **+** | + + | **+** |  | + |
| **Тест 2** | + | + + | **+** |  | + |
| **Тест 4** |  |  | **+** | + |  |

Тестирование программы 1:

**Тест1:**

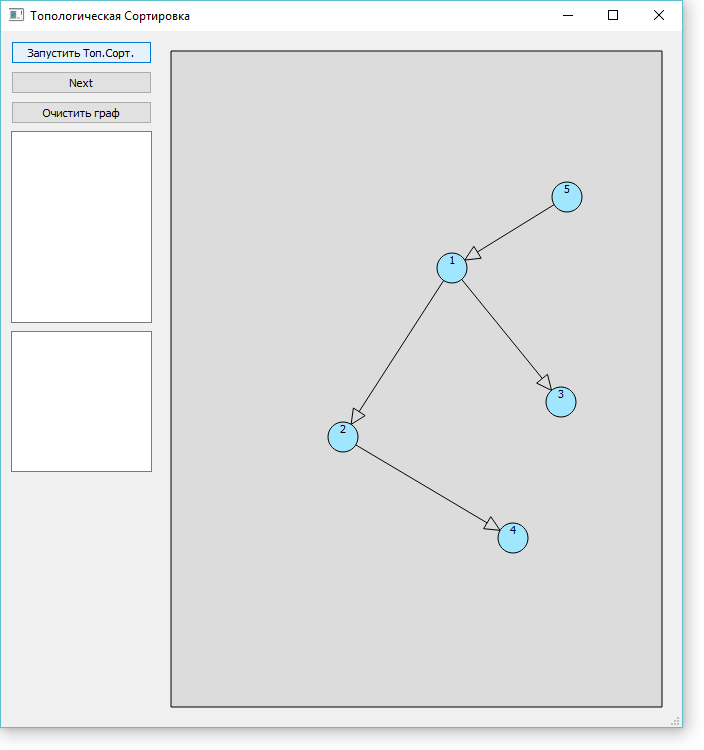


**Тест 2:**

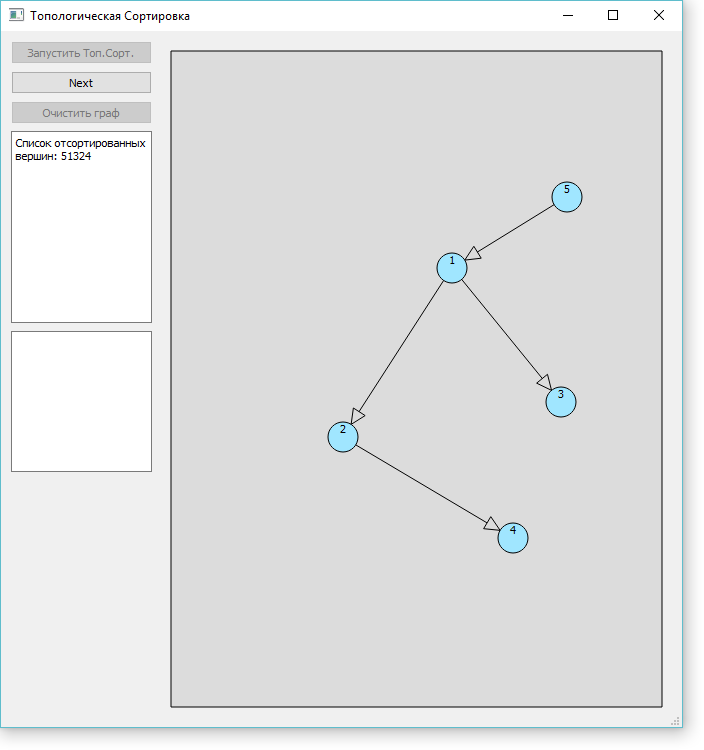


**Тест 3 :**

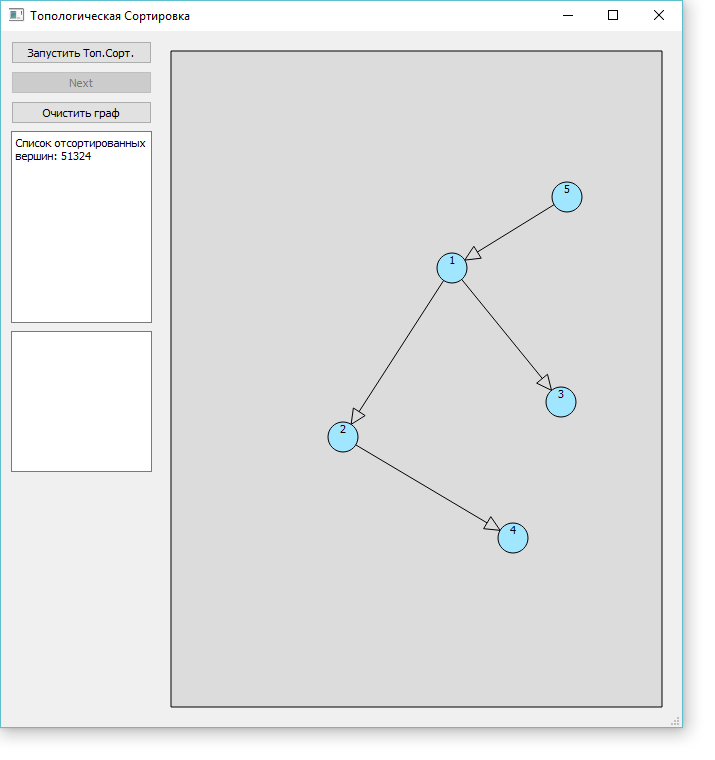
Ввели граф:



Нажатие кнопки Запустить Топ.Сорт.



Нажатие кнопки Next(завершает работу алгоритма)



Нажатие кнопки Очистить Граф

