МИНОБРНАУКИ РОССИИ

Санкт-Петербургский государственный электротехнический университет «ЛЭТИ» им. В.И. Ульянова (Ленина)

Кафедра МОЭВМ

Отчет по практике

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

Тема: Язык программирования Java. Алгоритм Брона-Кербоша. Визуализация.

Выполнили: студентки гр.4383

Козловских Е.Б.

Хафизова А.А.

Преподаватель: Шолохова О.М.

Санкт-Петербург

2016

# **ИСХОДНАЯ ПОСТАНОВКА ЗАДАЧИ**

Пошаговая визуализация алгоритма Брона-Кербоша построения максимальной клики на языке программирования Java. Совместная разработка проекта с использованием распределенной системы контроля версий Git.

# **АНАЛИЗ И ПРИМЕР РЕШЕНИЯ ЗАДАЧИ**

Программа на вход получает текстовый файл с исходным данными в следующем порядке: количество вершин в графе, количество рёбер в графе, две инцидентные ребру вершины. Это минимальный набор данных, необходимый для реализации алгоритма. Для алгоритма требуется 4 множества:

Множество comsub — множество, содержащее на каждом шаге рекурсии полный подграф для данного шага. Строится рекурсивно;

Множество candidates — множество вершин, которые могут увеличить comsub;

Множество not — множество вершин, которые уже использовались для расширения comsub на предыдущих шагах алгоритма;

Множество max\_comsub - множество для максимальной клики (начальное значение — пустое множество);

Алгоритм является рекурсивной процедурой, применяемой к этим четырём множествам.

ПРОЦЕДУРА extend (comsub, max\_comsub, candidates, not):

ПОКА candidates НЕ пусто И not НЕ содержит вершины, СОЕДИНЕННОЙ СО ВСЕМИ вершинами из candidates,

ВЫПОЛНЯТЬ:

1. Выбираем вершину v из candidates и добавляем её в comsub;
2. Формируем new\_candidates и new\_not, удаляя из candidates и not вершины, не СОЕДИНЕННЫЕ с v;
3. ЕСЛИ new\_candidates и new\_not пусты;
4. ТО comsub – клика И ЕСЛИ размер comsub (количество вершин) больше размера max\_comsub;
5. ТО поместить comsub в max\_comsub;
6. ИНАЧЕ ничего не делать;
7. ИНАЧЕ рекурсивно вызываем extend (comsub, max\_comsub, new\_candidates, new\_not);
8. Удаляем v из compsub и candidates, и помещаем в not;

Построение графа выполняется по кругу, координаты формируются автоматически.

**Пример выполнения.**

Входные данные:

4

4

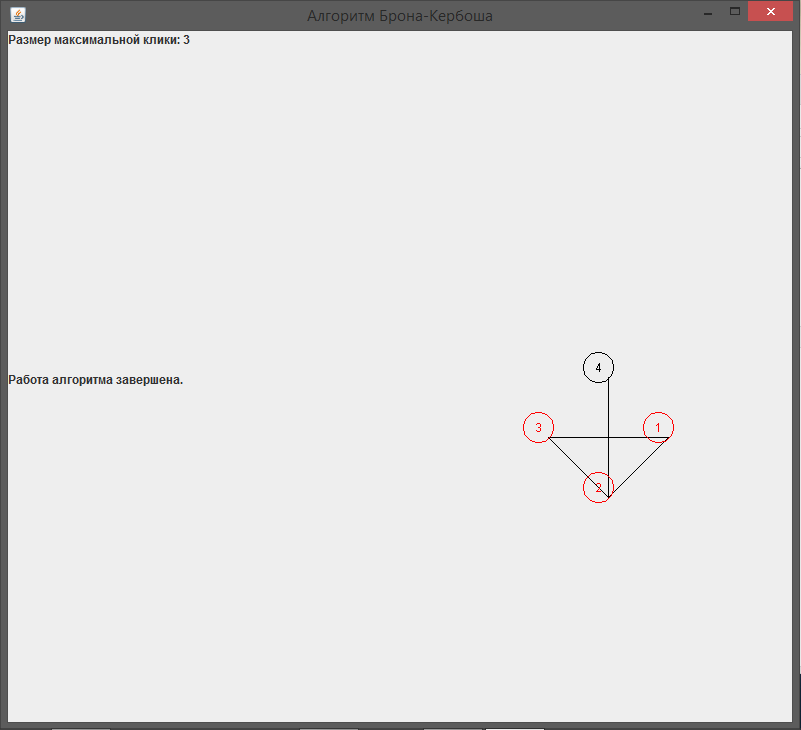
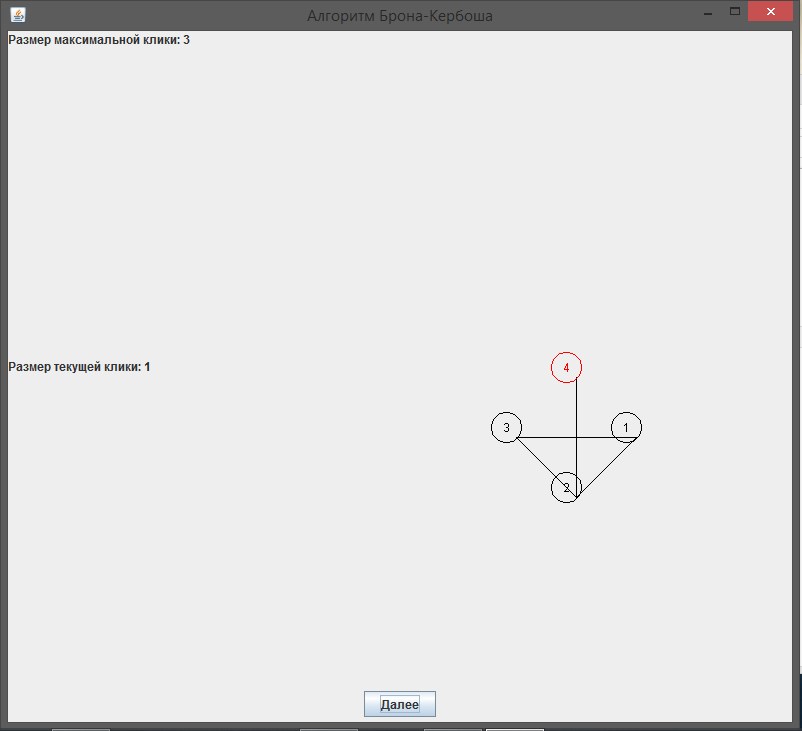
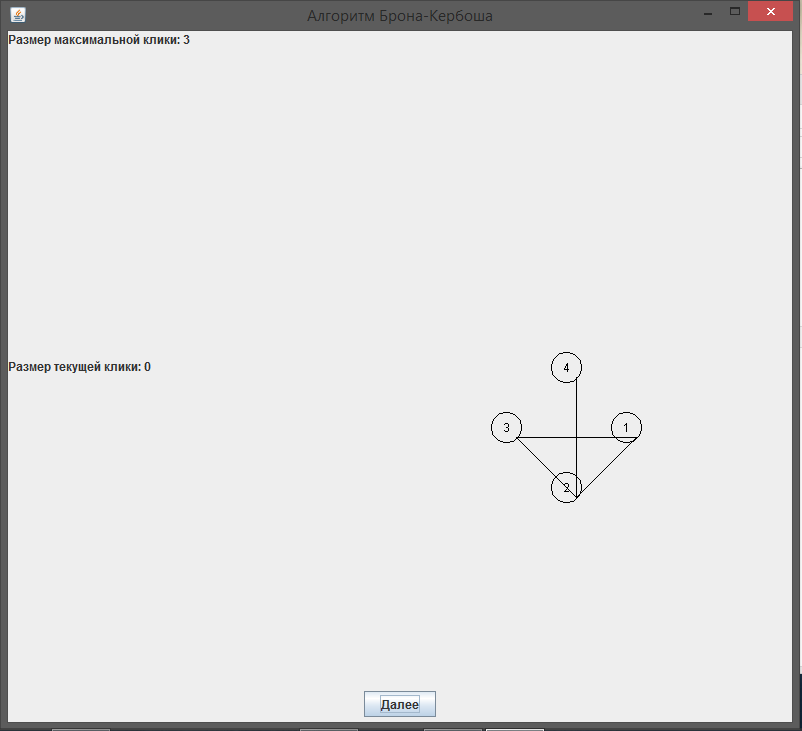
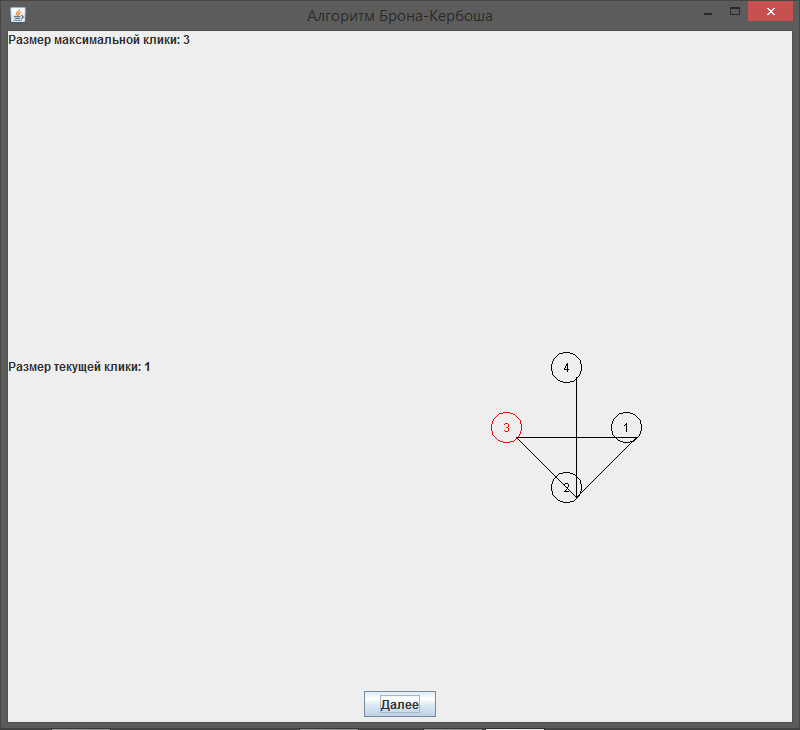
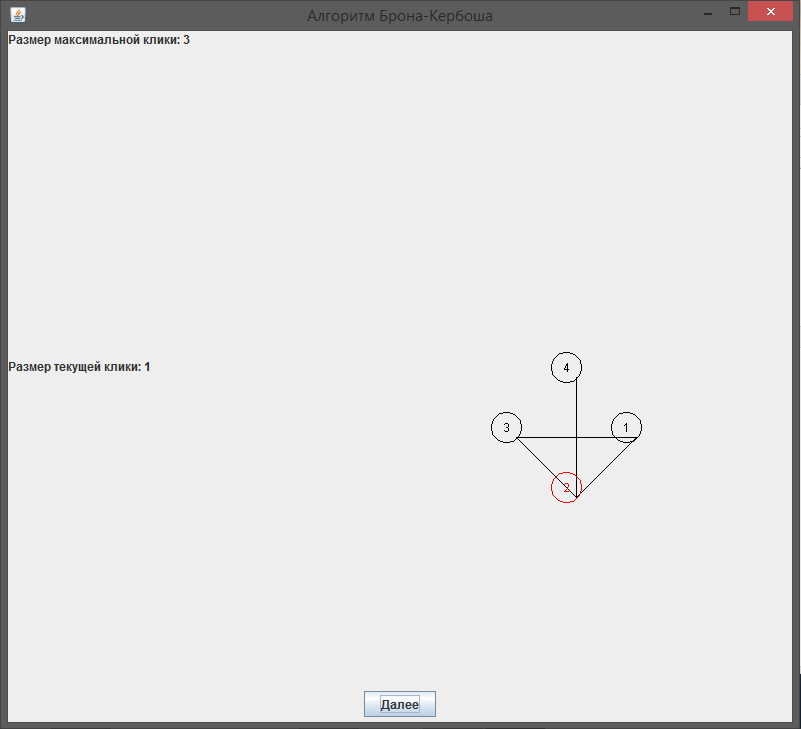
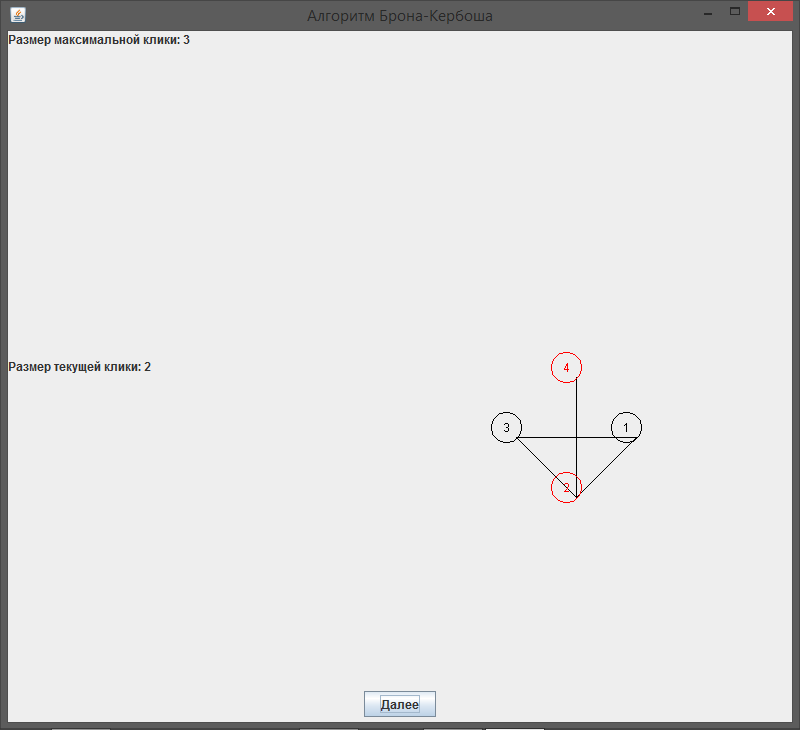
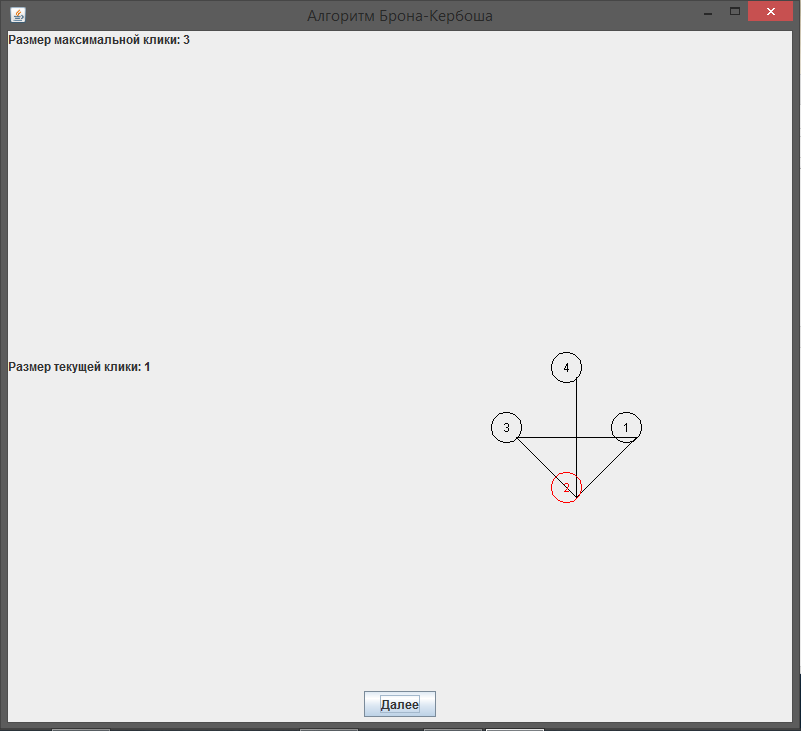
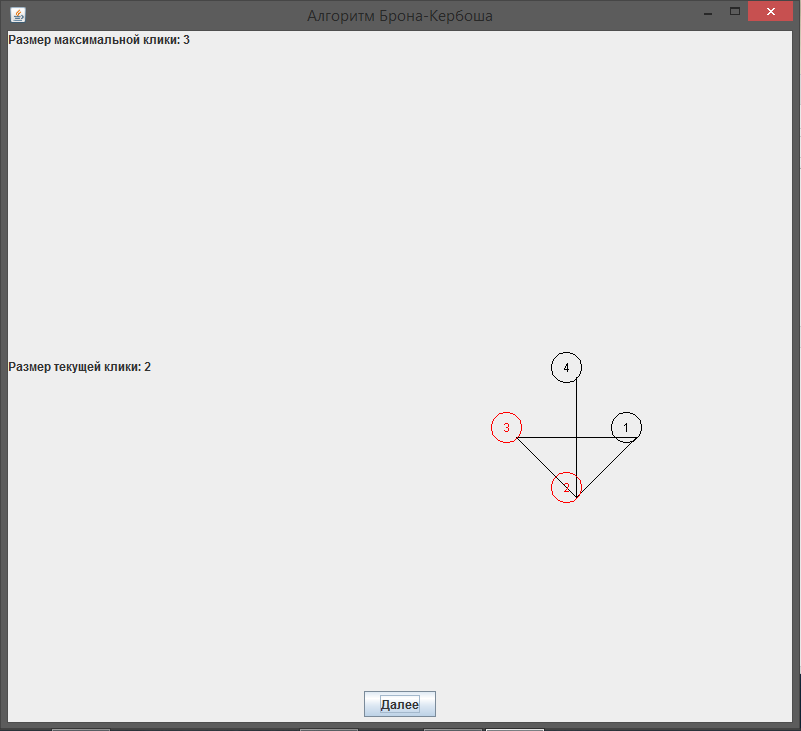
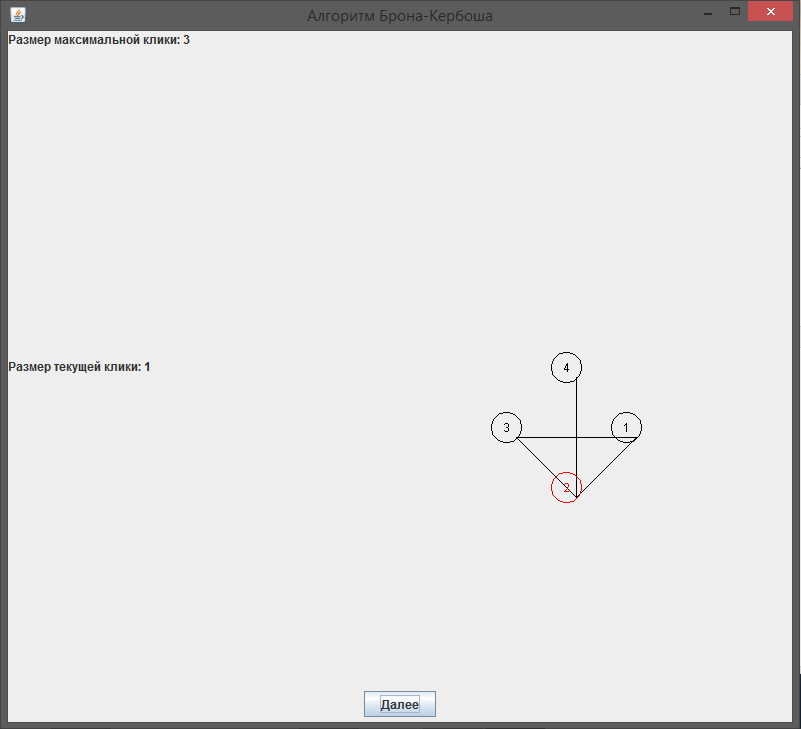
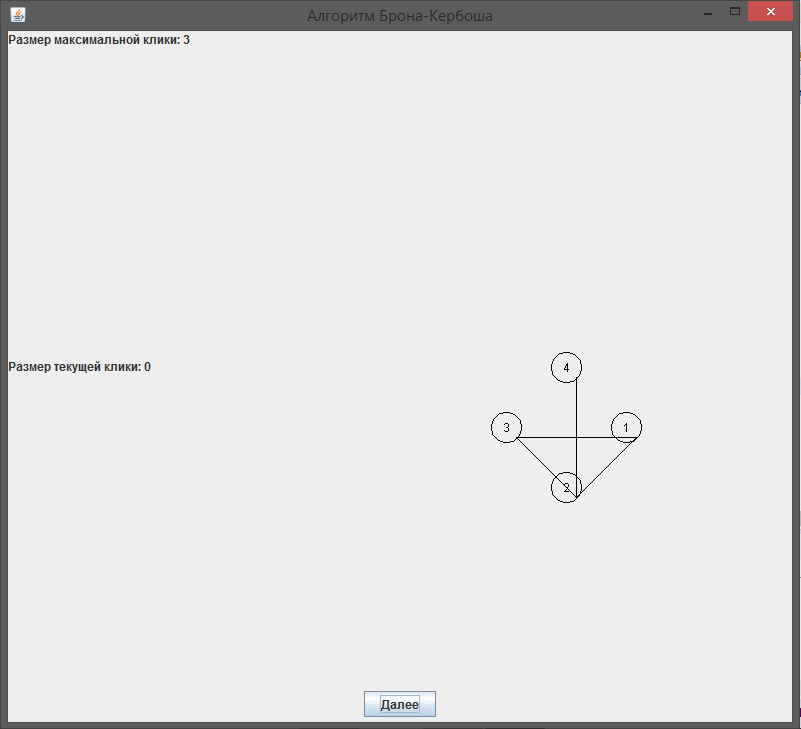
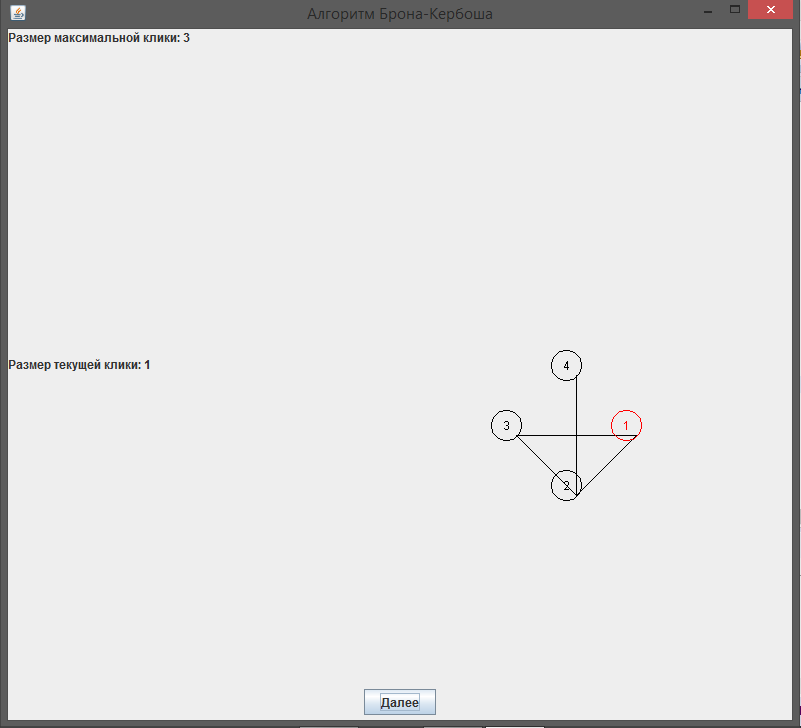
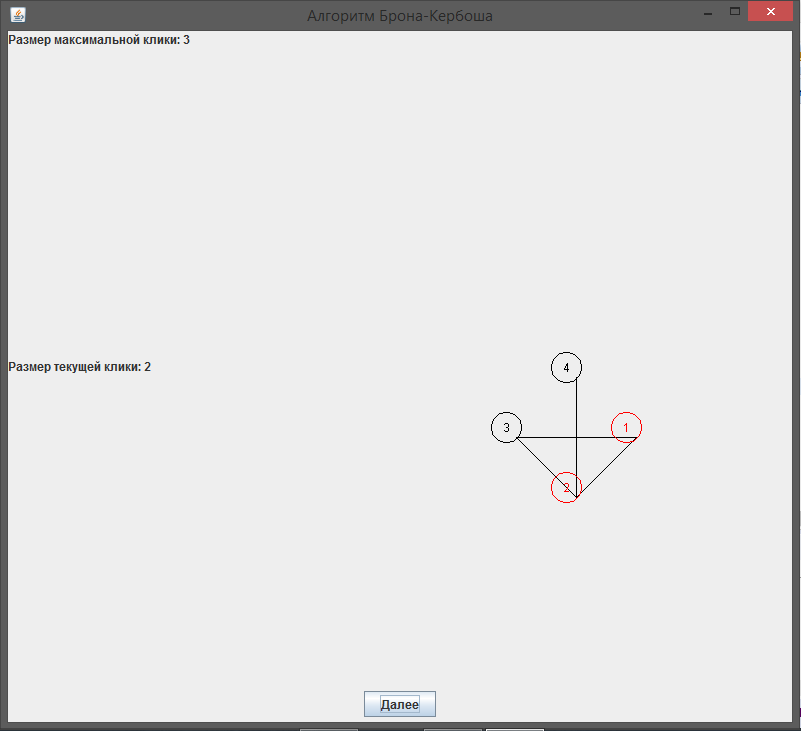
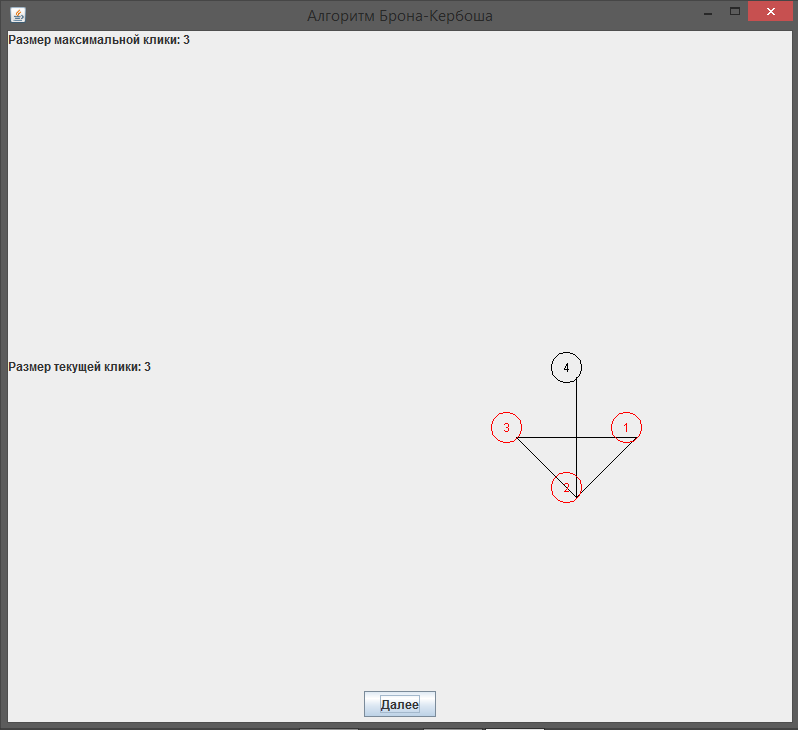
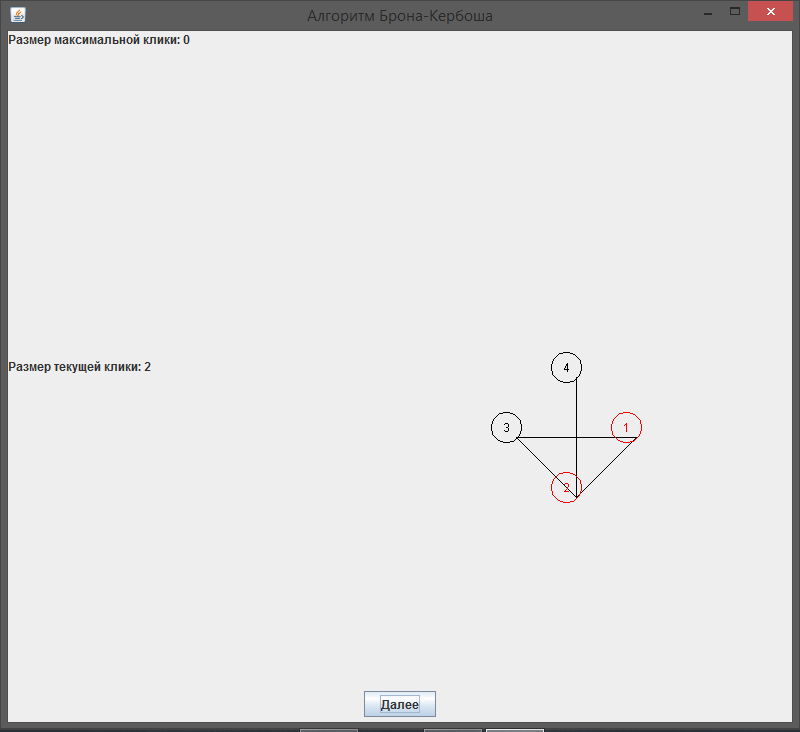
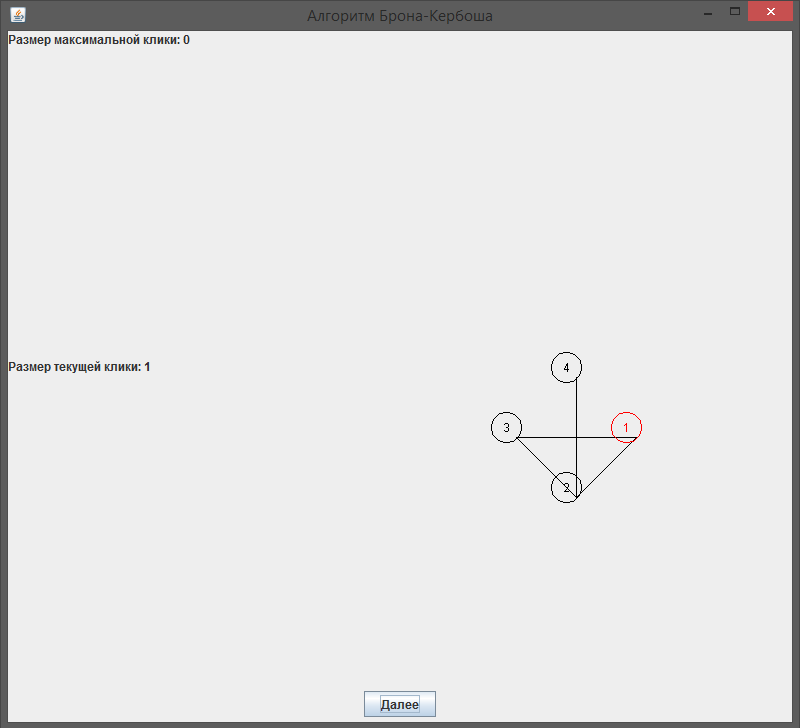
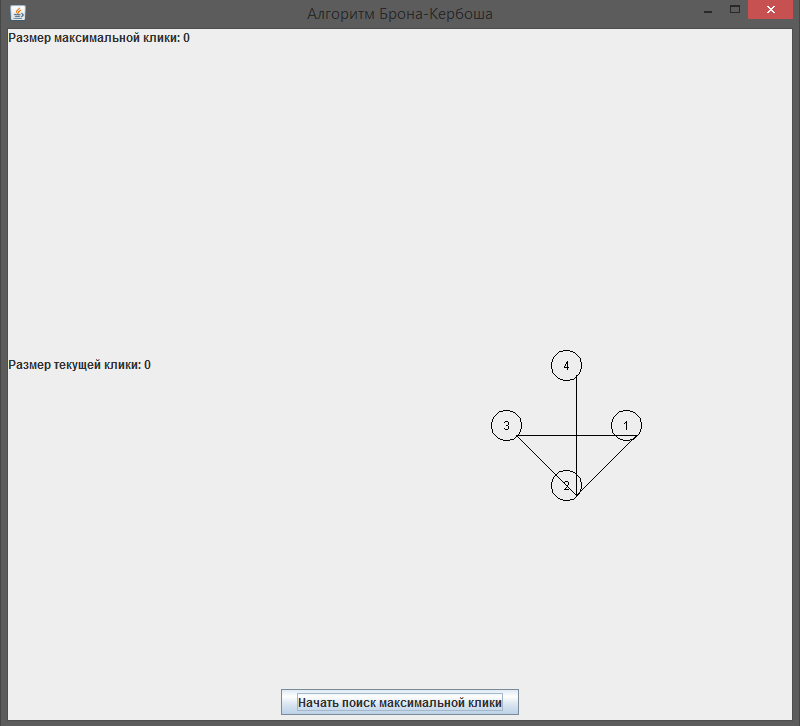
1 2

1 3

2 3

2 4

Выполнение алгоритма:



* **ФОРМАЛЬНАЯ ПОСТАНОВКА ЗАДАЧИ**
* Исходные данные:

Исходные данные находятся в текстовом файле в следующем порядке: количество вершин в графе, количество рёбер в графе, две инцидентные ребру вершины.

# Ограничения на исходные данные:

Данные типа int (целочисленные).

# Результирующие выходные данные:

Результатом выполнения программы является построение максимальной клики заданного графа.

* **ОПИСАНИЕ ОСНОВНЫХ КЛАССОВ**

|  |  |
| --- | --- |
| **Класс** | **Описание** |
| Graph | Класс, реализующий основную структуру данных, граф |
| GraphInterface | Интерфейс графа |
| BronKerbosh | Класс, реализующий алгоритм Брона-Кербоша |
| BronKerboshInterface | Интерфейс алгоритма |
| GraphGraphic | Класс, описывающий панель для отрисовки графа |
| Window | Класс, реализующий окно |
| Test | Класс тестирования, содержит функцию main |

# **ТЕСТИРОВАНИЕ ПРОГРАММЫ**

Входные данные:

5

10

1 2

1 3

1 4

1 5

2 3

2 4

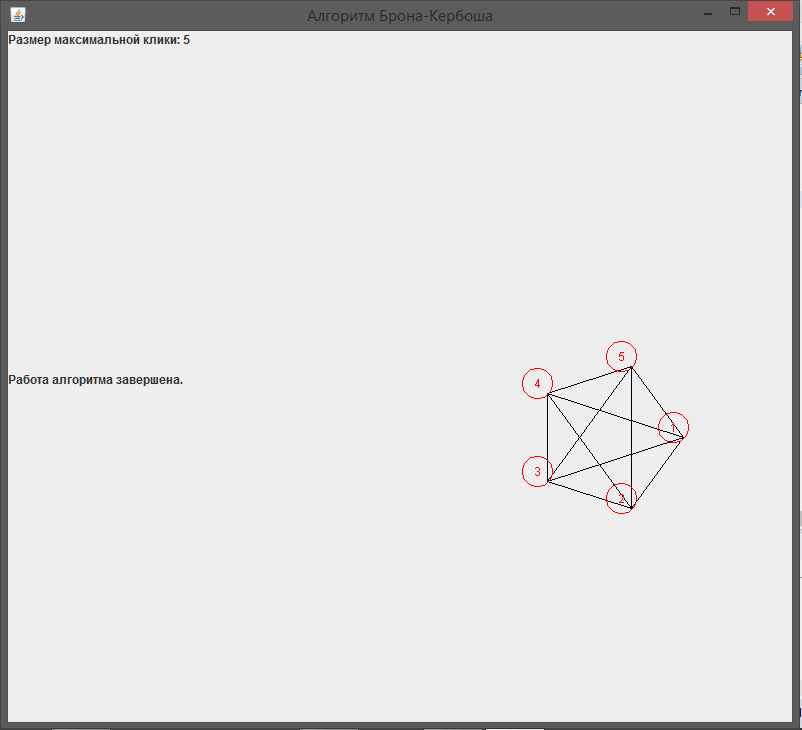
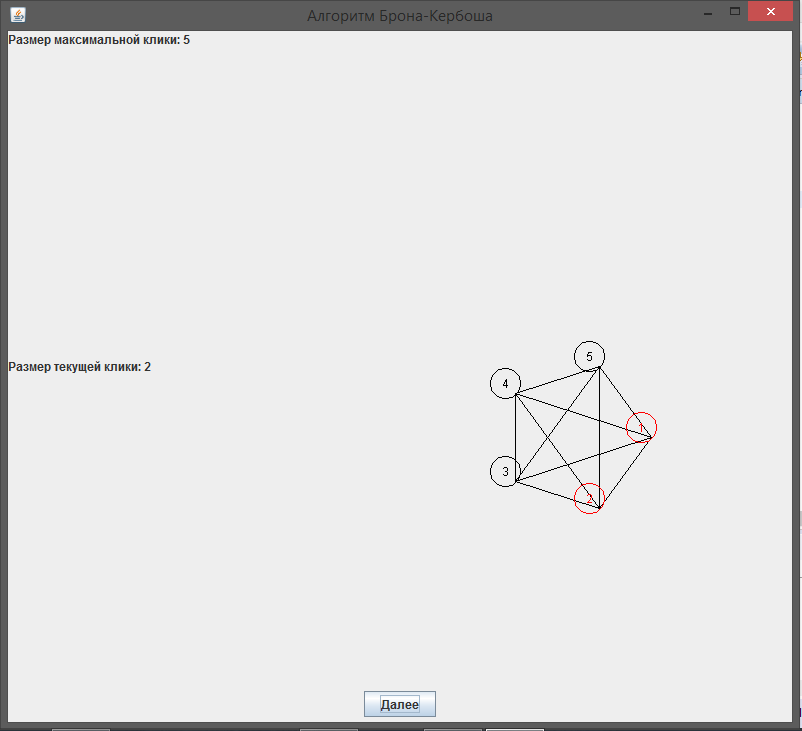
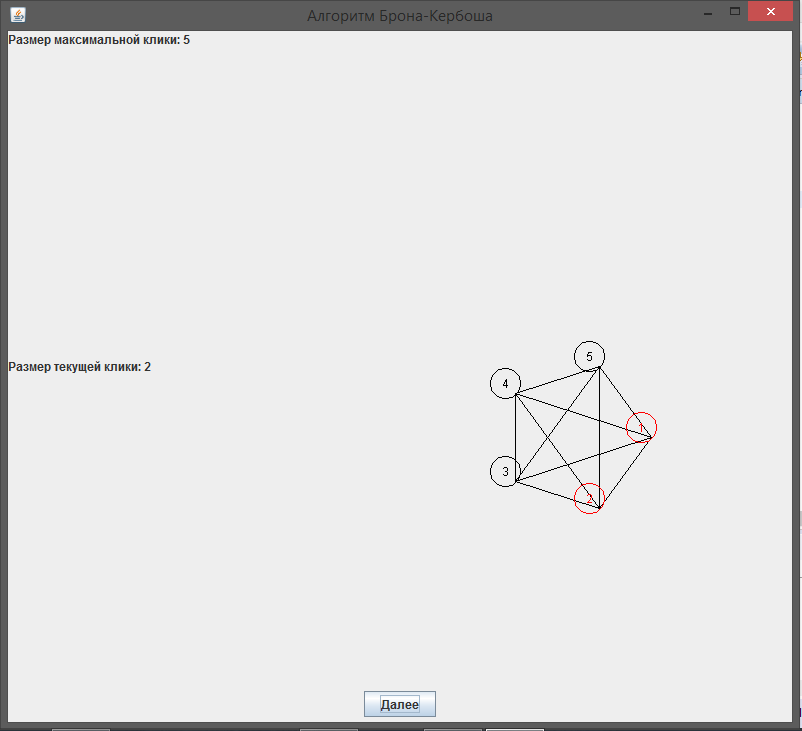
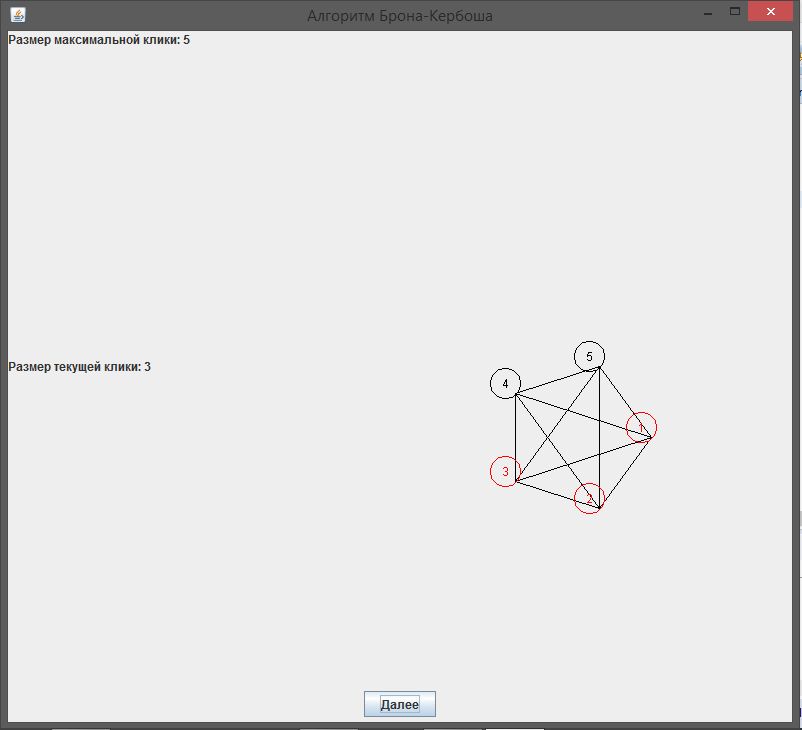
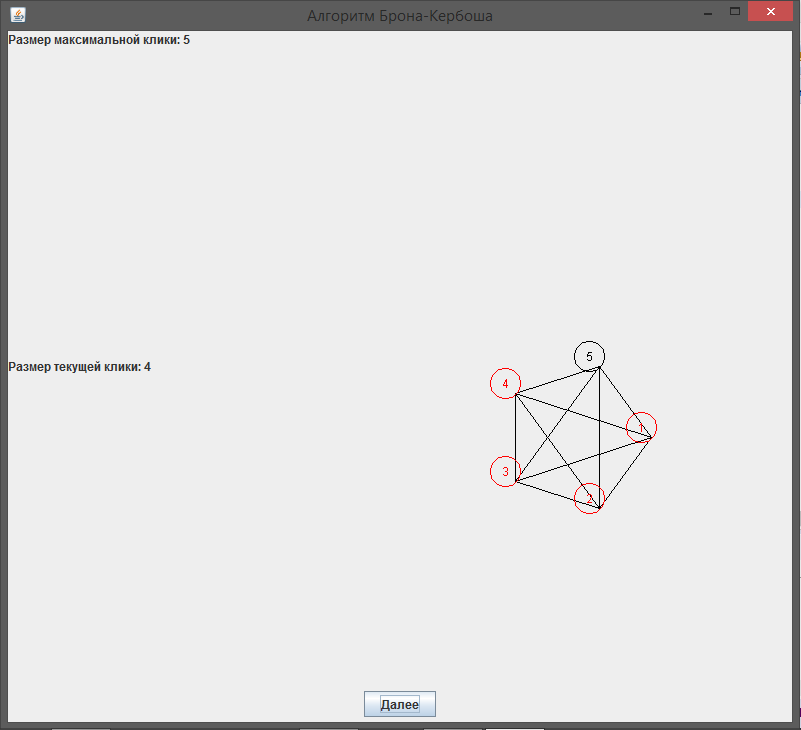
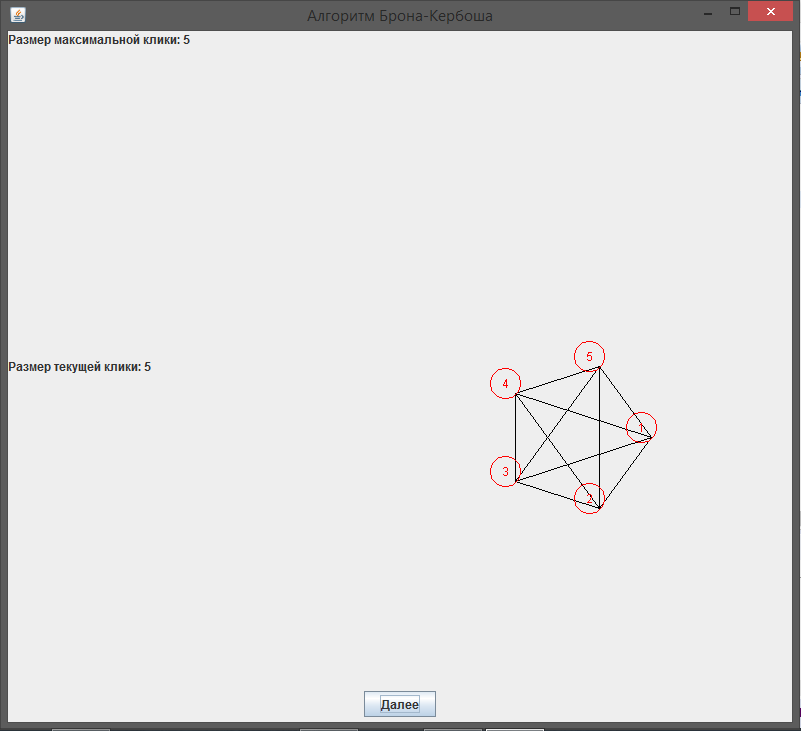
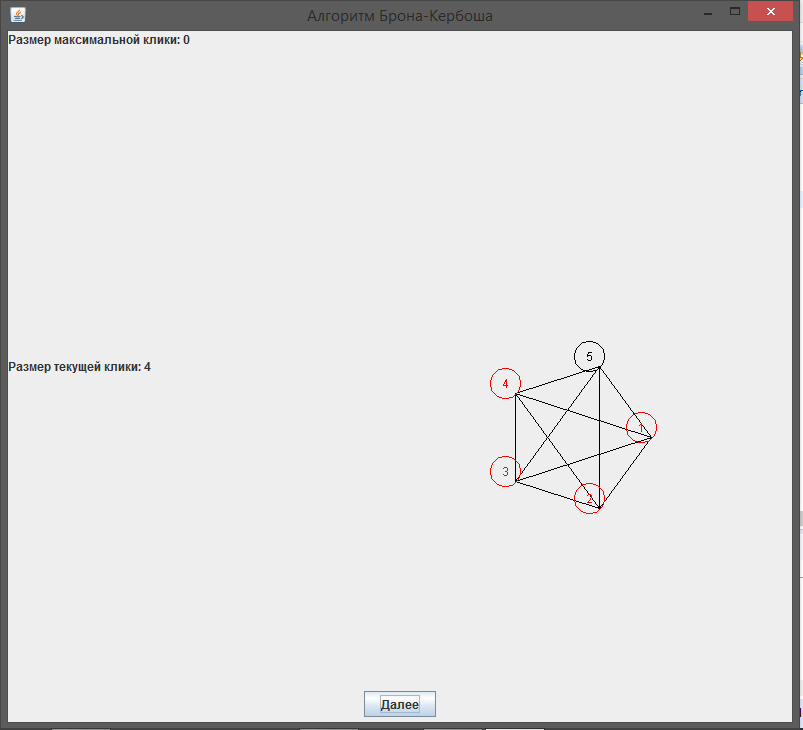
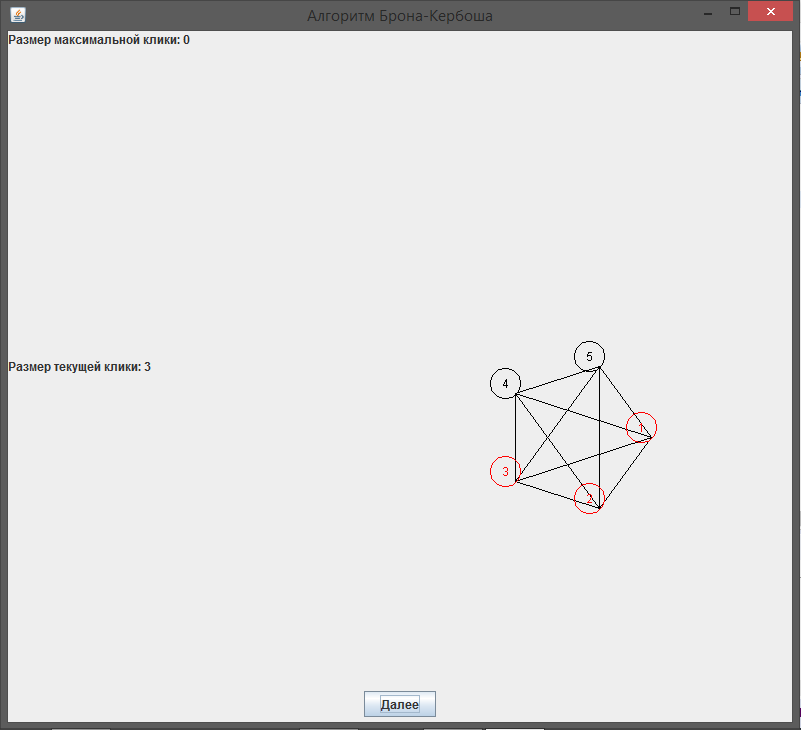
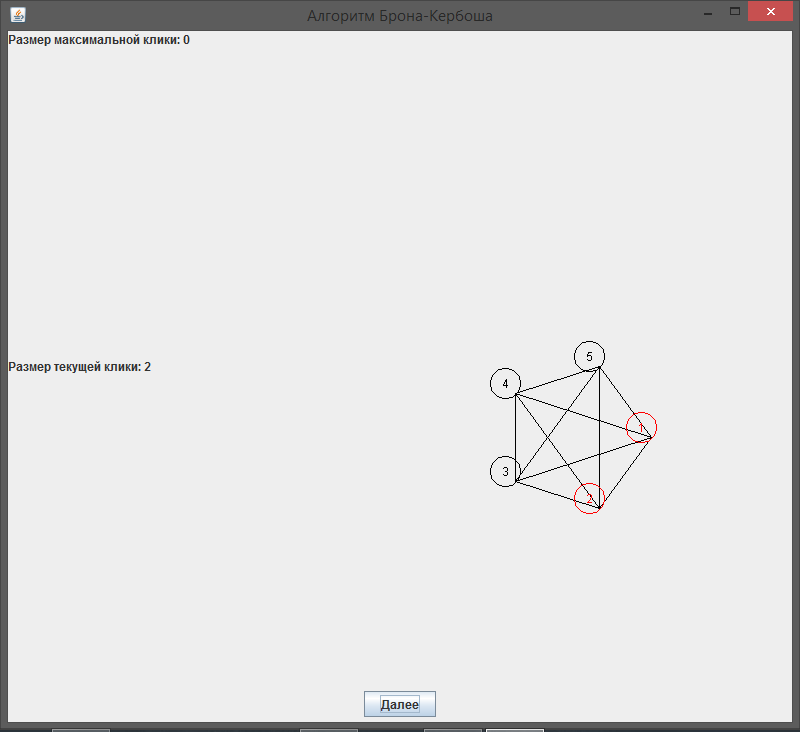
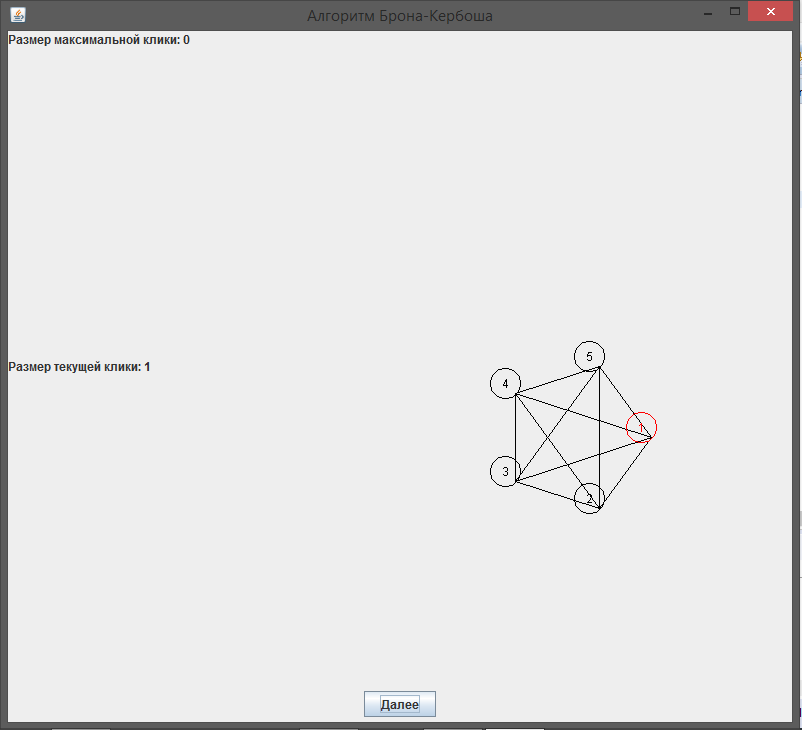
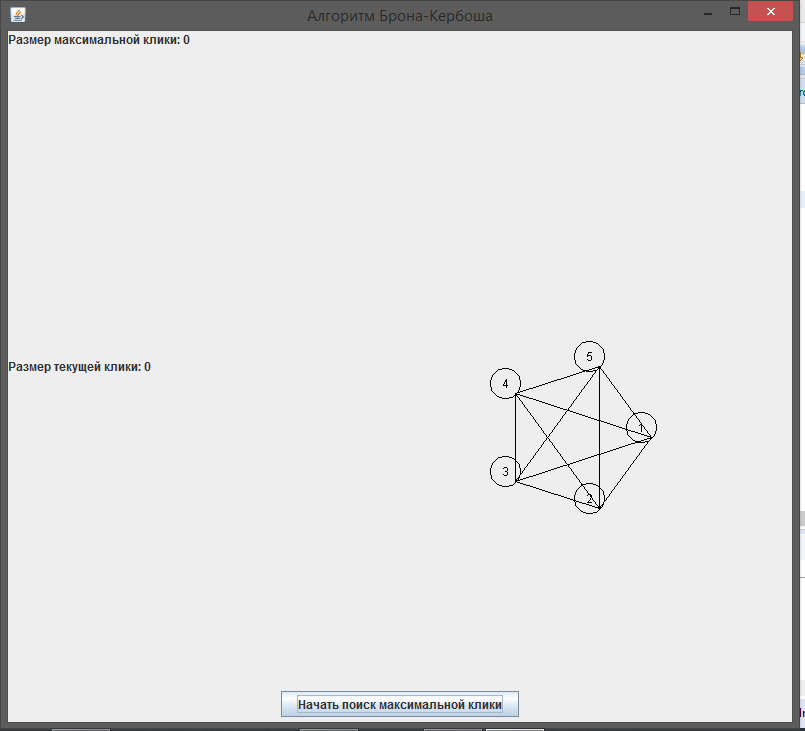
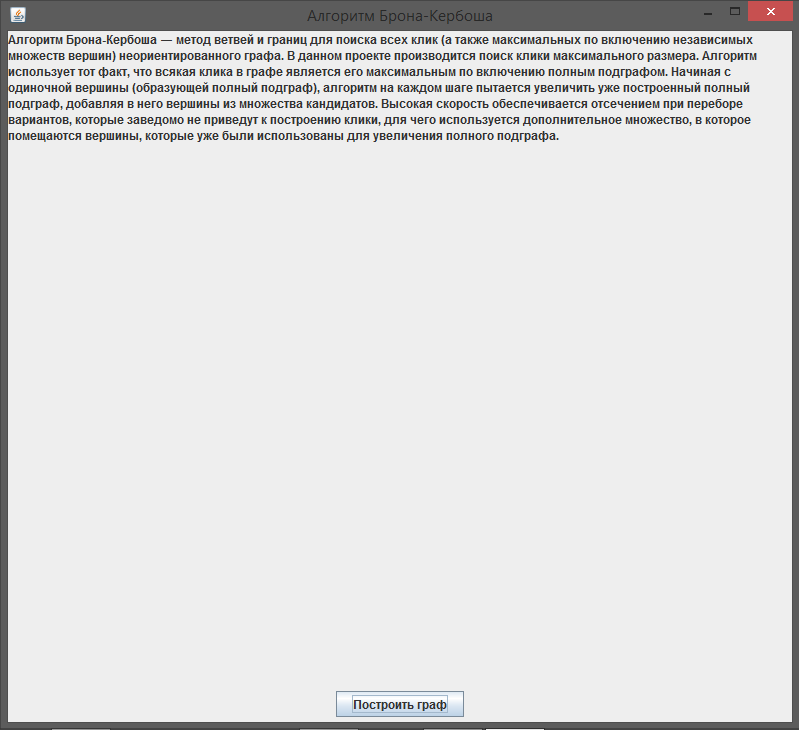
2 5

3 4

3 5

4 5

Выполнение алгоритма:



# **КОД ПРОГРАММЫ**

Graph.java

**import** java.io.File;  
**import** java.io.FileNotFoundException;  
**import** java.util.Scanner;  
**import** java.util.Vector;  
  
**public class** Graph **implements** GraphInterface {  
 **public static class** Edge {  
 **public** Integer **x**;  
 **public** Integer **y**;  
   
 **public** Edge(Integer x, Integer y) {  
 **this**.**x** = x;  
 **this**.**y** = y;  
 }  
 }  
 **public** Vector <Edge> **edges**;  
 **public** Integer **numVert**;  
 **public** Integer **numEdge**;  
   
 **public** Graph() {  
 **edges** = **new** Vector<>();  
 File f = **new** File(**"6.txt"**);  
 **try** {  
 Scanner scan = **new** Scanner(f);  
 **numVert** = scan.nextInt();  
 **numEdge** = scan.nextInt();  
 **int** a, b;  
 **for** ( **int** i = 0; i < **numEdge**; i++ ) {  
 a = scan.nextInt();  
 b = scan.nextInt();  
 addEdge(a, b);  
 }  
 } **catch** (FileNotFoundException e) {  
 e.printStackTrace();  
 }  
 }  
  
 **public void** clear() {  
 **edges**.clear();  
 }  
 **public void** addEdge(Integer x, Integer y) {  
 Edge e = **new** Edge(x, y);  
 **edges**.add(e);  
 }  
 **public int** GetNumEdges()  
 {  
 **return numEdge**;  
 }  
 **public int** GetNumVert()  
 {  
 **return numVert**;  
 }  
 **public int** GetEdgesV1(**int** num)  
 {  
 **return edges**.elementAt(num).**x**;  
 }  
 **public int** GetEdgesV2(**int** num)  
 {  
 **return edges**.elementAt(num).**y**;  
 }  
 **public void** output() {  
 System.***out***.println(**"edges.size(): "** + **edges**.size());  
 **for** ( **int** i = 0; i < **edges**.size(); i++ ) {  
 System.***out***.println(**"edges["** + i + **"]: "** + **edges**.get(i).**x** + **"; edges["** + i + **"]: "** + **edges**.get(i).**y** );  
 }  
 }  
}

GraphInterface.java

**public interface** GraphInterface {  
 **public void** clear();  
 **public int** GetEdgesV1(**int** num);  
 **public int** GetEdgesV2(**int** num);  
 **public int** GetNumEdges();  
 **public int** GetNumVert();  
 **public void** addEdge(Integer x, Integer y);  
 **public void** output();  
}

BronKerbosh.java

**import** java.io.FileNotFoundException;  
**import** java.io.FileOutputStream;  
**import** java.io.FileWriter;  
**import** java.io.IOException;  
**import** java.io.OutputStreamWriter;  
**import** java.io.PrintWriter;  
**import** java.util.ArrayList;  
**import** java.util.Scanner;  
  
**public class** BronKerbosh **implements** BronKerboshInterface {  
 **private** ArrayList<Integer> **comsub**, **candidates**, **not**, **maxComsub**;  
 **public** Graph **graph**;  
   
 **public** BronKerbosh() {  
 **comsub** = **new** ArrayList<>();  
 **candidates** = **new** ArrayList<>();   
 **not** = **new** ArrayList<>();  
 **maxComsub** = **new** ArrayList<>();  
 **graph** = **new** Graph();  
 **for**( **int** i = 1; i < **graph**.**numVert** + 1; i++ )   
 **candidates**.add(i);  
 }   
 **public boolean** edgeExist(**int** a, **int** b) {  
 **for** ( **int** k = 0; k < **graph**.**edges**.size(); k++ ) {  
 **if** ( **graph**.**edges**.get(k).**x**.equals(a) && **graph**.**edges**.get(k).**y**.equals(b) )  
 **return true**;  
 **if** ( **graph**.**edges**.get(k).**x**.equals(b) && **graph**.**edges**.get(k).**y**.equals(a) )  
 **return true**;  
 }  
 **return false**;  
 }  
 **public boolean** haveAnEdge(ArrayList<Integer> newNot, ArrayList<Integer> newCand) {  
 **if** ( newNot.isEmpty() )   
 **return false**;  
   
 **for** ( **int** i = 0; i < newNot.size(); i++ ) {  
 **int** a = newNot.get(i);  
 **for** ( **int** j = 0; j < newCand.size(); j++ ) {  
 **int** b = newCand.get(j);  
 **if** ( !edgeExist(a, b) ) **return false**;  
 }  
 }  
 **return true**;  
 };  
 **public void** base() {  
 **try** {  
 PrintWriter out = **new** PrintWriter(**"note.txt"**);  
 out.close();  
 } **catch** (FileNotFoundException e) {  
 e.printStackTrace();  
 }  
 algBK(**candidates**, **not**);  
 }  
 **public void** check(**int** v, ArrayList<Integer> newNot, ArrayList<Integer> newCand, **int** j) {  
 **if** ( !edgeExist(v, **graph**.**edges**.get(j).**x**) ) {  
 **if** ( newCand.contains( **graph**.**edges**.get(j).**x** ) ) {   
 newCand.remove( **graph**.**edges**.get(j).**x** );   
 newCand.trimToSize();   
 }  
 **if** ( newNot.contains( **graph**.**edges**.get(j).**x** ) ) {   
 newNot.remove( **graph**.**edges**.get(j).**x** );   
 newNot.trimToSize();   
 }   
 }  
 **if** ( !edgeExist(v, **graph**.**edges**.get(j).**y**) ) {  
 **if** ( newCand.contains( **graph**.**edges**.get(j).**y** ) ) {   
 newCand.remove( **graph**.**edges**.get(j).**y** );   
 newCand.trimToSize();   
 }  
 **if** ( newNot.contains( **graph**.**edges**.get(j).**y** ) ) {   
 newNot.remove( **graph**.**edges**.get(j).**y** );   
 newNot.trimToSize();   
 }  
 }  
 }  
 **public void** remove(ArrayList<Integer> newNot, ArrayList<Integer> newCand, Object s) {  
 **if** ( newCand.contains(s) ) {   
 newCand.remove(s);  
 newCand.trimToSize();   
 }  
 **if** ( newNot.contains(s) ) {   
 newNot.remove(s);   
 newNot.trimToSize();   
 }  
 }  
 **public void** print(ArrayList<Integer> list) {  
 **try** {  
 FileOutputStream fos = **new** FileOutputStream(**"note.txt"**, **true**);  
 OutputStreamWriter osr = **new** OutputStreamWriter(fos);  
 PrintWriter out = **new** PrintWriter(osr);  
 out.print( list.size() + **" "** );  
 **for** ( **int** i = 0; i < list.size(); i++ ) {  
 out.print(list.get(i) + **" "**);  
 }  
 out.println();  
 out.close();  
 } **catch** (FileNotFoundException e) {  
 e.printStackTrace();  
 }  
 }  
 **public void** algBK(ArrayList<Integer> cand, ArrayList<Integer> not) {  
 **while** ( !cand.isEmpty() && !haveAnEdge(not, cand) ) {  
 **int** v = cand.get(0);  
 Object s = (Object) v;   
 **comsub**.add(v);   
 print(**comsub**);  
   
 ArrayList<Integer> newCand = **new** ArrayList<>();   
 ArrayList<Integer> newNot = **new** ArrayList<>();  
   
 newCand.addAll(cand);   
 newNot.addAll(not);   
 remove(newNot, newCand, s);  
   
 **for** ( **int** j = 0; j < **graph**.**edges**.size(); j++ )   
 check(v, newNot, newCand, j);  
 **if** ( newCand.isEmpty() && newNot.isEmpty() ) {   
 **if** ( **maxComsub**.isEmpty() ) {  
 **maxComsub**.addAll(**comsub**);  
 print(**maxComsub**);  
 }  
 **else** {  
 **if** ( **comsub**.size() > **maxComsub**.size() ) {   
 **maxComsub**.clear();   
 **maxComsub**.addAll(**comsub**);   
 print(**maxComsub**);  
 } **else** print(**comsub**);  
 }  
 }  
 **else** {   
 algBK(newCand, newNot);  
 }  
 **comsub**.remove(s);   
 **comsub**.trimToSize();   
 print(**comsub**);  
   
 cand.remove(s);   
 cand.trimToSize();   
 not.add(v);  
 }  
 }  
}

BronKerboshInterface.java

**import** java.util.ArrayList;  
  
**public interface** BronKerboshInterface {  
 **public boolean** edgeExist(**int** a, **int** b);  
 **public boolean** haveAnEdge(ArrayList<Integer> newNot, ArrayList<Integer> newCand);  
 **public void** base();  
 **public int** GetNumEdges();  
 **public int** GetNumVert();  
 **public int** GetEdgesV1(**int** num);  
 **public int** GetEdgesV2(**int** num);  
 **public void** print(ArrayList<Integer> list);  
 **public void** check(**int** v, ArrayList<Integer> newNot, ArrayList<Integer> newCand, **int** j);  
 **public void** remove(ArrayList<Integer> newNot, ArrayList<Integer> newCand, Object s);  
 **public void** algBK(ArrayList<Integer> cand, ArrayList<Integer> not);  
}

GraphGraphic.java

**import** javax.swing.\*;  
**import** java.awt.\*;  
**import** java.util.ArrayList;  
  
**public class** GraphGraphic **extends** JPanel {  
 **private class** MyPoint {  
 **int x**, **y**;  
 MyPoint (**int** x, **int** y) {  
 **this**.**x** = x;  
 **this**.**y** = y;  
 }  
 }  
 **public static** Graphics2D *p*;  
 **private** ArrayList<Integer> **clique**;  
 MyPoint[] **coordinates**;  
 **private** BronKerbosh **gr**;  
 **private boolean check**;  
  
 GraphGraphic() {  
 **gr** = **new** BronKerbosh();  
 **gr**.base();  
 **coordinates** = **new** MyPoint[ **gr**.**graph**.**numVert** ];  
 **check** = **false**;  
 **clique** = **new** ArrayList<>();  
 }  
  
 **public void** SetCoordinate() {  
 Double alpha = 2 \* Math.***PI*** / **gr**.**graph**.**numVert**;  
 Double alpha0 = 0.0;  
 Integer R = 15 \* **gr**.**graph**.**numVert**;  
 Integer x0 = 400;  
 Integer y0 = 365;  
 **int** a, b;  
 **for** ( **int** i = 0; i < **gr**.**graph**.**numVert**; i++ ) {  
 a =(**int**)Math.*round*(Math.*cos*(alpha0));  
 b=(**int**)Math.*round*(Math.*sin*(alpha0));  
 a= x0 + (**int**)(Math.*round*(Math.*cos*(alpha0) \* R));  
 b = y0 + (**int**)(Math.*round*(Math.*sin*(alpha0) \* R));  
 MyPoint tmp = **new** MyPoint(a, b);  
 **coordinates**[i] = tmp;  
 alpha0 = alpha0 + alpha;  
 }  
 }  
 **public void** setClique(ArrayList<Integer> curClique, **boolean** check) {  
 **clique**.clear();  
 **clique**.addAll(curClique);  
 **this**.**check** = check;  
 }  
 **public void** DrawVert(**int** x, **int** y, Graphics p, **int** num) {  
 String str = Integer.*toString*(num);  
 p.drawOval(x, y, 30, 30);  
 p.drawString(str, x + 12, y + 20);  
 }  
 **public void** DrawEdge(**int** x, **int** y, Graphics p) {  
 p.drawLine(**coordinates**[x - 1].**x** + 25, **coordinates**[x - 1].**y** + 25, **coordinates**[y - 1].**x** + 25, **coordinates**[y - 1].**y** + 25);  
 }  
 **public void** paint(Graphics g) {  
 SetCoordinate();  
 g.setColor(**new** Color(0, 0, 0));  
   
 **for** ( **int** i = 0; i < **gr**.**graph**.**numEdge**; i++ ) {  
 DrawEdge(**gr**.**graph**.**edges**.get(i).**x**, **gr**.**graph**.**edges**.get(i).**y**, g);  
 }  
 **for** ( **int** i = 0; i < **gr**.**graph**.**numVert**; i++ ) {  
 DrawVert(**coordinates**[i].**x**, **coordinates**[i].**y**, g, i + 1);  
 }  
   
 **if** ( **check** ) g.setColor(**new** Color(255, 0, 0));  
 **else** g.setColor(**new** Color(0, 0, 0));  
  
 **for** ( **int** i = 0; i < **clique**.size(); i++ ) {  
 DrawVert(**coordinates**[**clique**.get(i)-1].**x**, **coordinates**[**clique**.get(i)-1].**y**, g, **clique**.get(i));  
 }  
 }  
}

Window.java

**import** java.awt.\*;  
**import** java.awt.event.\*;  
**import** java.io.File;  
**import** java.io.FileNotFoundException;  
**import** java.util.ArrayList;  
**import** java.util.Scanner;  
  
**import** javax.swing.\*;  
  
**public class** Window **extends** JFrame {  
 **private** JLabel **text1**, **text2**;  
 **private** JButton **construct**, **start**, **next**;  
 **private** JPanel **buttonsPanel**;  
 **private** GraphGraphic **gg**;  
 **private** Container **c**;  
 **private boolean flag**;  
   
 **private** ArrayList<Integer> **list1**;  
 **private** ArrayList<Integer> **list2**;  
 **private** ArrayList<Integer> **maxClique**;  
 **private** File **f**;   
 **private** Scanner **scan**;   
 **private boolean y**, **x**;  
 **public** Window() {  
 **super**(**"Алгоритм Брона-Кербоша"**);  
   
 **c** = getContentPane();  
 **gg** = **new** GraphGraphic();  
   
 **buttonsPanel** = **new** JPanel(**new** FlowLayout());  
 **text1** = **new** JLabel(**"<html> Алгоритм Брона-Кербоша — метод ветвей и границ для поиска всех клик (а также максимальных по включению независимых <br> множеств вершин) неориентированного графа. В данном проекте производится поиск клики максимального размера. Алгоритм использует тот факт, что всякая клика в графе является его максимальным по включению полным подграфом. Начиная с одиночной вершины (образующей полный подграф), алгоритм на каждом шаге пытается увеличить уже построенный полный подграф, добавляя в него вершины из множества кандидатов. Высокая скорость обеспечивается отсечением при переборе вариантов, которые заведомо не приведут к построению клики, для чего используется дополнительное множество, в которое помещаются вершины, которые уже были использованы для увеличения полного подграфа. </html>"**);  
 **text2** = **new** JLabel(**" "**);  
   
 **construct** = **new** JButton(**"Построить граф"**);  
 **start** = **new** JButton(**"Начать поиск максимальной клики"**);  
 **next** = **new** JButton(**"Далее"**);  
   
 **list1** = **new** ArrayList<>();  
 **list2** = **new** ArrayList<>();  
 **maxClique** = **new** ArrayList<>();  
   
 **y** = **true**; **x** = **true**;  
 File f = **new** File(**"note.txt"**);  
 **try** {  
 **scan** = **new** Scanner(f);  
 } **catch** (FileNotFoundException e) {  
 e.printStackTrace();  
 }  
   
 add(**text1**, BorderLayout.***NORTH***);   
 add(**text2**, BorderLayout.***BEFORE\_LINE\_BEGINS***);  
  
 **construct**.setToolTipText(**"Нажмите здесь, чтоб построить граф"**);  
 **buttonsPanel**.add(**construct**);  
 **c**.add(**buttonsPanel**, BorderLayout.***SOUTH***);  
 initListeners();  
   
 setBounds(0, 0, 800, 730);  
 setDefaultCloseOperation(JFrame.***EXIT\_ON\_CLOSE***);  
 }  
 **private void** read(ArrayList<Integer> list) {  
 **if** ( **scan**.hasNextInt() ) {  
 list.clear();  
 **int** size = **scan**.nextInt();  
 **for** ( **int** i = 0; i < size; i++ ) {  
 **int** a = **scan**.nextInt();  
 list.add(a);  
 };  
 }  
 **else x** = **false**;  
 }  
 **private void** initListeners() {  
 **construct**.addActionListener(**new** ActionListener() {  
 **public void** actionPerformed(ActionEvent e) {  
 **buttonsPanel**.remove(**construct**);  
 **start**.setToolTipText(**"Нажмите здесь, чтоб начать работу алгоритма"**);  
 **buttonsPanel**.add(**start**);  
 revalidate(); repaint();  
 **text1**.setText(**"Размер максимальной клики: "** + 0);  
 **text2**.setText(**"Размер текущей клики: "** + 0);   
 **c**.add(**gg**);  
 }  
 });  
 **start**.addActionListener(**new** ActionListener() {  
 **public void** actionPerformed(ActionEvent e) {  
 **buttonsPanel**.remove(start);  
 next.setToolTipText(**"Нажмите здесь, чтоб начать поиск клики"**);  
 buttonsPanel.add(next);  
 revalidate(); repaint();  
 text1.setText(**"Размер максимальной клики: "** + 0);  
 text2.setText(**"Размер текущей клики: "** + 0);  
 }  
 });  
 next.addActionListener(**new** ActionListener() {  
 **public void** actionPerformed(ActionEvent e) {  
 **if** ( y ) {  
 read(list1);  
 read(**list2**);   
 **y** = **false**;  
 }   
 **else** {  
 **list1**.clear();   
 **list1**.addAll(**list2**);   
   
 **if** ( **x** ) read(**list2**);   
 **if** ( **list1**.size() == **list2**.size() && !**list1**.isEmpty() ) {  
 **maxClique**.clear();  
 **maxClique**.addAll(**list2**);  
 }  
 }  
 revalidate(); repaint();  
 **text2**.setText(**"Размер максимальной клики: "** + **list1**.size() );  
 **text1**.setText(**"Размер текущей клики: "** + **maxClique**.size() );  
 **flag** = **true**;  
 **gg**.setClique(**list1**, **flag**);  
 **gg**.repaint();  
   
 **if** ( !**x** ) {  
 **gg**.setClique(**maxClique**, **flag**);  
 **buttonsPanel**.remove(**next**);  
 revalidate(); repaint();  
 **text1**.setText(**"Размер максимальной клики: "** + **maxClique**.size() );  
 **text2**.setText(**"Работа алгоритма завершена."**);  
 }  
 }  
 });  
 }  
}

Test.java

**import** java.awt.\*;  
  
**public class** Test {  
 **public static void** main(String[] args) **throws** InterruptedException {BronKerboshInterface i = **new** BronKerbosh();  
 i.base();  
 Window app = **new** Window();  
 app.setVisible(**true**);  
  
 }  
}

# **СОВМЕСТНАЯ РАЗРАБОТКА**

Совместная разработка производилась с использованием распределенной системы контроля версий.

|  |  |  |
| --- | --- | --- |
| **Задача** | **Исполнитель** | **Срок реализации** |
| [[Doc] Создать wiki c описанием алгоритма](https://github.com/ElizavetaKozlovskykh/Bron-Kerbosch-algorithm/issues/1) | Хафизова Айгуль | 27 июня |
| [[Doc] Создать wiki c требованиями к проекту](https://github.com/ElizavetaKozlovskykh/Bron-Kerbosch-algorithm/issues/2) | Козловских Елизавета | 27 июня |
| [[FR] Создать GUI прототип пректа](https://github.com/ElizavetaKozlovskykh/Bron-Kerbosch-algorithm/issues/3) | Хафизова Айгуль | 27 июня |
| [[FR] Создать класс основной структуры данных (граф/дерево)](https://github.com/ElizavetaKozlovskykh/Bron-Kerbosch-algorithm/issues/4) | Хафизова Айгуль | 26 июня |
| [[FR] Отрисовка основной структуры данных(граф/дерево)](https://github.com/ElizavetaKozlovskykh/Bron-Kerbosch-algorithm/issues/5) | Козловских Елизавета | 3 июля |
| [[FR] Реализация алгоритма](https://github.com/ElizavetaKozlovskykh/Bron-Kerbosch-algorithm/issues/6) | Хафизова Айгуль | 2 июля |
| [[FR] Считывание/генерация основной структуры данных](https://github.com/ElizavetaKozlovskykh/Bron-Kerbosch-algorithm/issues/7) | Хафизова Айгуль  Козловских Елизавета | 30 июня |
| [[FR] Визуализация выделения максимальной клики](https://github.com/ElizavetaKozlovskykh/Bron-Kerbosch-algorithm/issues/9) | Козловских Елизавета | 4 июля |
| [[FR] Реализация класса состояния строящейся клики](https://github.com/ElizavetaKozlovskykh/Bron-Kerbosch-algorithm/issues/12) | Хафизова Айгуль | 3 июля |

# **ВЫВОД**

В ходе выполнения данной работы были изучены основы языка программирования Java, получены практические навыки работы с графической библиотекой Swing и распределенной системой контроля версий Git.