Tallinna Tehnikaülikool

Individuaaltöö aines „Algoritmid ja andmestruktuurid“

Koostaja: Silver-Ed Sillaots

Eriala: IT-süsteemide arendus

Rühm: IADB34

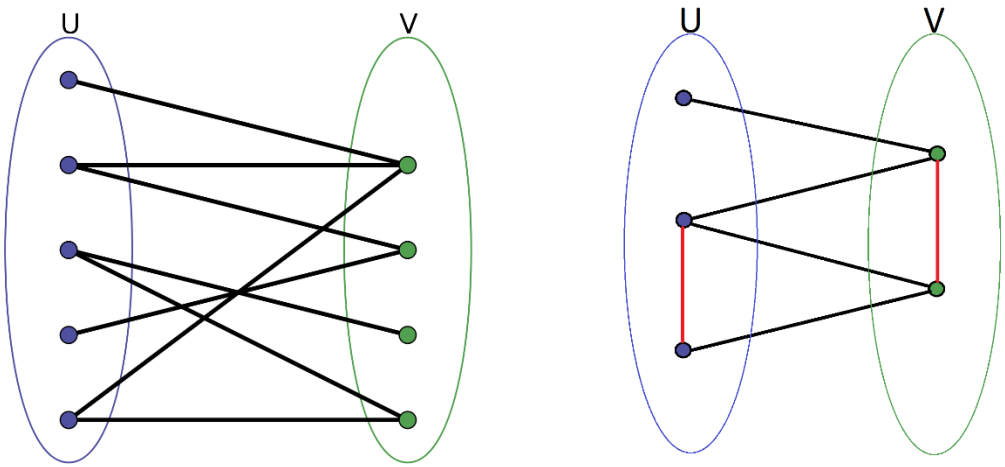
Juhendaja: Jaanus Pöial

Tallinn 2020

Sisukord

1.Ülesande püstitus

Ülesanne C-13.27 on võetud raamatust „Data Structures and Algorithms in Java“. Meil on suunamata graaf G, mis on kahepoolne, kui tema tipud saab jagada kahte kogumisse(U, V) nii, et iga selle graafi serva üks otspunkt ehk tipp asub ühes kogumis ning teine otspunkt teises kogumis. Koostada algoritm, mis teeb kindlaks, kas etteantud suvaliselt genereeritud lihtgraaf on kahepoolne ilma ette teadmata, millistesse kogumitesse tipud kuuluvad. Allpool olevas näites on ainult vasakpoolne graaf kahepoolne. Parempoolne graaf ei sobi punaseks märgitud servade tõttu, mille mõlemad otspunktid asuvad samas kogumis.

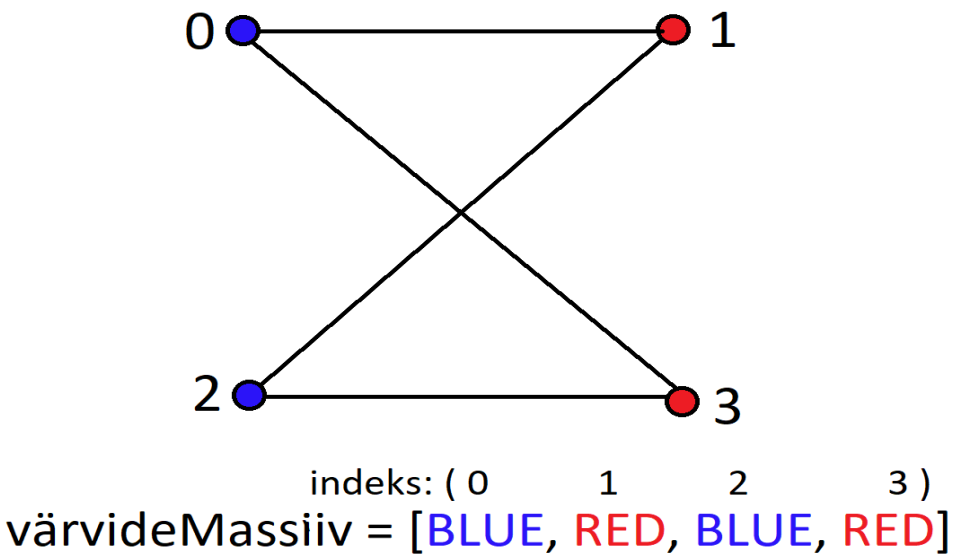


Allikas: Näide 1, lisatud ise juurde

2.Lahenduse kirjeldus

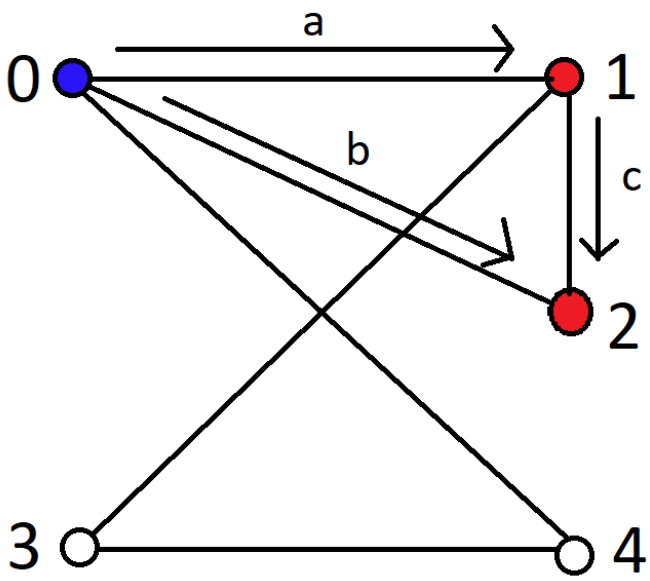
Leidmaks, kas etteantud suvaliselt genereeritud lihtgraaf on kahepoolne või mitte, tegin ma meetodi, mis kontrollib, kas graafi tippe on võimalik jagada (3)kahte kogumisse. Kõigepeal võtsin kasutusele külgnevuse mitmemõõtmelise massiivi ehk maatriksi, mille sain kaasaantud meetodilt ning lõin uue (2)värvide massiivi, mis sümboliseerib iga tipu värvi. Selle massiivi pikkuse seadsin võrdseks külgnevuse maatriksi pikkusega ning määrasin tippude värviks vaikimisi valge. Otsustasin tippude kogumid eristada värvidega, milleks on punane ning sinine.

Illustratiivne näide:



Kui massiivid loodud, hakkan töötama kahe tsükkliga. Esimene itereerib üle vahemiku 0 kuni värvi massiivi pikkus - 1. Kui meil on graafis näiteks viis tippu, oleks esimene vahemik 0 kuni 4. Teine tsükkel asub esimese sees ning itereerib üle vahemiku 0 kuni värvi massiivi pikkus, viie tipuga graafis oleks teine vahemik 0 kuni 5. Esimese tsükli väärtus sümboliseerib kahe tipu vahelise serva alguspunkti ning teise tsükli väärtus kahe tipu vahelise serva lõpp-punkti. Esialgu kasutan antud väärtuseid külgnevuse maatriksis olevatele elementidele viitamiseks, et kontrollida, kas kahe punkti vahel eksisteerib serv. Seejärel kui lõpp-punkt on jätkuvalt valge, värvin ta alguspunktiga vastupidist värvi. Kui näiteks alguspunkt on sinine, muutub lõpp-punkt punaseks, samamoodi vastupidi. Esimest värvimist teostades, kus alguspunkt on alati valge, värvitakse lõpppunkt alati punaseks.

Ilustratiivne näide2:



index: (0, 1, 2, 3, 4)

värvideMassiiv = [BLUE,RED,RED,WHITE,WHITE]

Kui tekib näiteks olukord(c), kus antud lõpp-punkt on juba varasemalt värvitud(b) ning on sama värvi, mis alguspunkt(a), lõpetatakse tsüklid koheselt ning meetod tagastab false väärtuse. Selline olukord saab tekkida ainult, siis kui antud serva alguspunkt ning lõpp-punkt satuvad vältimatult samasse kogumisse, mistõttu pole tegemist kahepoolse graafiga. Kui meetod jõuab tsüklite lõpuni ning värvi massiivis pole ühtegi vaikimisi väärtust ehk kõik väärtused on, kas sinised või punased, on tegemist kahepoolse graafiga ning meetod tagastab true.

3.Programmi kasutamisjuhend

Programmi kasutamiseks tuleb kõigepealt tekitada run() meetodis uus muutuja:

```
public void run() {
    Graph myGraphVariable = new Graph ( s: "GraphNameExample");
}
```

Muutuja nime(“myGraphVariable”) ning graafi nime(„GraphNameExample“) saab kasutaja ise valida.

Teise sammuna tuleb luua suvaline graaf selle jaoks mõeldud meetodiga createRandomSimpleGraph(), mille parameetriteks on: n = tippude arv, m = servade arv, bipartite = kahepoolsus. Kui seada bipartite parameetri väärtuseks true, tekitab meetod kahepoolse lihtgraafi, false väärtuse puhul tekitab meetod suvalise lihtgraafi, mis võib olla samuti kahepoolne, kuid ei pruugi.

```
public void run() {
    Graph myGraphVariable = new Graph ( s: "GraphNameExample");
    myGraphVariable.createRandomSimpleGraph( n: 12,  m: 40,  bipartite: false);
}
```

Pärast graafi loomist, soovides teada saada, kas tegemist on kahepoolse graafiga või mitte, tuleb jooksutada meetodit nimega checkIfGraphIsBipartite(). Selle meetodi jooksutamisel prinditakse vastus konsooli ning tagastatakse väärtus true või false.

```
public void run() {
    Graph myGraphVariable = new Graph ( s: "GraphNameExample");
    myGraphVariable.createRandomSimpleGraph( n: 12,  m: 40,  bipartite: false);
    myGraphVariable.checkIfGraphIsBipartite();
}
```

```
Graph: GraphNameExample with 12 vertices and 40 edges is not bipartite.
```

Antud näite puhul ei olnud tegemist kahepoolse graafiga.

4.Testimiskava

Testimise eesmärgil, et tagada järjepidevus lisasin juurde võimaluse meetodile createRandomSimpleGraph() genereerida alati kahepoolne graaf, kui on täidetud järgmine tingimus: Sisestatud servade arv ei ületa sisestatud tippude arvu maksimaalset servade arvu, et graaf oleks kahepoolne:

$m \leq \frac{n^2}{4}$, n = tippude arv, m = servade arv.

Seda valikut saab muuta andes meetodiga kaasa bipartite atribuudi väärtusega true või false.

```
GraphTask gt = new GraphTask();
```

TEST 1

```
@Test (timeout=20000)
public void testSmallBipartiteGraph() {
   GraphTask.Graph graph = gt.new Graph( s: "G");
   graph.createRandomSimpleGraph( n: 10, m: 25, bipartite: true);
   System.out.println(graph);
   long startTime = System.nanoTime();
   assertTrue ( message: "This graph is suppose to be bipartite.", graph.checkIfGraphIsBipartite());
   long endTime = System.nanoTime();
   long duration = endTime - startTime;
   System.out.println("Time taken: " + duration / 1000000 + " ms");
}
```

Antud testis sisestame checkIfGraphIsBipartite() meetodisse pisema suurusega kahepoolse graafi, mille tippude arv on 10 ning servade arv 25.

Ootus: Meetod tuvastab, et antud graaf on kahepoolne ning servade arv tuleb 25.

Output:

```
G
v1 --> av1_v10 (v1->v10) av1_v8 (v1->v8) av1_v6 (v1->v6) av1_v4 (v1->v4) av1_v2 (v1->v2)
v2 --> av2_v9 (v2->v9) av2_v7 (v2->v7) av2_v5 (v2->v5) av2_v1 (v2->v1) av2_v3 (v2->v3)
v3 --> av3_v10 (v3->v10) av3_v8 (v3->v8) av3_v6 (v3->v6) av3_v2 (v3->v2) av3_v4 (v3->v4)
v4 --> av4_v9 (v4->v9) av4_v7 (v4->v7) av4_v1 (v4->v1) av4_v3 (v4->v3) av4_v5 (v4->v5)
v5 --> av5_v10 (v5->v10) av5_v8 (v5->v8) av5_v2 (v5->v2) av5_v4 (v5->v4) av5_v6 (v5->v6)
v6 --> av6_v9 (v6->v9) av6_v3 (v6->v3) av6_v1 (v6->v1) av6_v5 (v6->v5) av6_v7 (v6->v7)
v7 --> av7_v10 (v7->v10) av7_v4 (v7->v4) av7_v2 (v7->v2) av7_v6 (v7->v6) av7_v8 (v7->v8)
v8 --> av8_v5 (v8->v5) av8_v3 (v8->v3) av8_v1 (v8->v1) av8_v7 (v8->v7) av8_v9 (v8->v9)
v9 --> av9_v6 (v9->v6) av9_v4 (v9->v4) av9_v2 (v9->v2) av9_v8 (v9->v8) av9_v10 (v9->v10)
v10 --> av10_v7 (v10->v7) av10_v5 (v10->v5) av10_v3 (v10->v3) av10_v1 (v10->v1) av10_v9 (v10->v9)

Graph: G with 10 vertices and 25 edges is bipartite.
Time taken: 35 ms
```

Ootus == Output

TEST 2

```java
@Test (timeout=20000)
public void testMediumBipartiteGraph() {
   GraphTask.Graph graph = gt.new Graph( s: "G");
   graph.createRandomSimpleGraph( n: 150, m: 5625, bipartite: true);
   System.out.println(graph);
   long startTime = System.nanoTime();
   assertTrue ( message: "This graph is suppose to be bipartite.", graph.checkIfGraphIsBipartite());
   long endTime = System.nanoTime();
   long duration = endTime - startTime;
   System.out.println("Time taken: " + duration / 1000000 + " ms");
}
```

Antud testis sisestame checkIfGraphIsBipartite() meetodisse keskmise suurusega kahepoolse graafi, mille tippude arv on 150 ning servade arv 5625.

Ootus: Meetod tuvastab, et antud graaf on kahepoolne ning servade arv tuleb 5626.

Output: Konsoolis välja prinditud servade arvu lisamine aruandesse, võtaks liigselt palju ruumi, mille tõttu selle astme jätan vahele. Konsoolis ikkagi esitatakse andmed.

```
Graph: G with 150 vertices and 5625 edges is bipartite.
Time taken: 38 ms
```

Ootus == Output

TEST 3

```
@Test (timeout=20000)
public void testLargeBipartiteGraph() {
   GraphTask.Graph graph = gt.new Graph( s: "G");
   graph.createRandomSimpleGraph( n: 2500, m: 1562500, bipartite: true);
   System.out.println(graph);
   long startTime = System.nanoTime();
   assertTrue ( message: "This graph is suppose to be bipartite.", graph.checkIfGraphIsBipartite());
   long endTime = System.nanoTime();
   long duration = endTime - startTime;
   System.out.println("Time taken: " + duration / 1000000 + " ms");
}
```

Antud testis sisestame checkIfGraphIsBipartite() meetodisse maksimaalse suurusega kahepoolse graafi, mille tippude arv on 2500 ning servade arv 1562500.

Ootus: Meetod tuvastab, et antud graaf on kahepoolne ning servade arvuks tuleb 1562500.

Output: Konsoolis välja prinditud servade arvu lisamine aruandesse, võtaks liigselt palju ruumi, mille tõttu selle astme jätan vahele. Konsoolis ikkagi esitatakse andmed.

```
Graph: G with 2500 vertices and 1562500 edges is bipartite.
Time taken: 96 ms
```

Ootus == Output

TEST 4

```
@Test (timeout = 20000)
public void testImpossibleBipartiteGraph() {
   GraphTask.Graph graph = gt.new Graph( s: "G");
   graph.createRandomSimpleGraph( n: 2000, m: 1562500, bipartite: false);
   assertFalse(graph.checkIfGraphIsBipartite());
}
```

Antud testis sisestame suvaliselt genereeritud lihtgraafi meetodisse checkIfGraphIsBipartite(), mis ületab 2000 tipuga kahepoolse graafi maksimaalset võimalikku servade arvu.

Ootus: Meetod tuvastab, et maksimaalselt võimalik servade arv kahepoolse graafi jaoks antud tippude arvuga on selle graafi puhul ületatud ning tagastab meile maksimaale võimaliku servade arvu 1000000, tippude arvu 2000 ning, et graaf ei ole kahepoolne.

Output: Konsoolis välja prinditud servade arvu lisamine aruandesse, võtaks liigselt palju ruumi, mille tõttu selle astme jätan vahele. Konsoolis ikkagi esitatakse andmed

```
Graph: G is not bipartite.
Impossible number of edges: 1562500 for a bipartite graph consisting of 2000 vertices.
Maximum number of edges for this graph to be bipartite would be: 1000000
```

Ootus == Output

TEST 5

```
@Test (timeout=20000)
public void testLargeRandomButMustBeBipartiteGraph() {
   GraphTask.Graph graph = gt.new Graph( s: "G");
   graph.createRandomSimpleGraph( n: 2500, m: 2499, bipartite: true);
   long startTime = System.nanoTime();
   assertTrue ( message: "This graph is suppose to be bipartite.", graph.checkIfGraphIsBipartite());
   long endTime = System.nanoTime();
   long duration = endTime - startTime;
   System.out.println("Time taken: " + duration / 1000000 + " ms");
}
```

Antud testis sisestame meetodisse checkIfGraphIsBipartite() suvaliselt genereeritud lihtgraafi, mille tippude arv on 2500 , servade arv 2499 ning, mis peab olema kahepoolne.

Ootus: Meetod tuvastab, et tegemist on kahepoolse graafiga ning tagastab meile tippude arvu ning servade arvu.

Output:

```
Graph: G with 2500 vertices and 2499 edges is bipartite.
Time taken: 11 ms
```

Ootus == Output

5.Kasutatud allikad

Näide 1: https://upload.wikimedia.org/wikipedia/commons/thumb/e/e8/Simple-bipartite-graph.svg/1200px-Simple-bipartite-graph.svg.png

2) https://www.geeksforgeeks.org/m-coloring-problem-backtracking-5/

3) https://www.geeksforgeeks.org/bipartite-graph/

6.Lisad

LISA 1: GraphTask.java

```java
import java.awt.*;
import java.util.Arrays;

/** Container class to different classes, that makes the whole
 * set of classes one class formally.
 */
public class GraphTask {

   /** Main method. */
   public static void main (String[] args) {
      GraphTask a = new GraphTask();
      a.run();
   }

   /** Actual main method to run examples and everything. */
   public void run() {
      Graph g = new Graph ( s: "G");
      g.createRandomSimpleGraph( n: 2000,  m: 875628,  bipartite: false);
      System.out.println(g);
      g.checkIfGraphIsBipartite();

   }

   /**
    * Ülesande kirjeldus:
    * A graph G is bipartite if its vertices can be partitioned into two sets X and Y
    * such that every edge in G has one end vertex in X and the other in Y. Design
    * and analyze an efficient algorithm for determining if an undirected graph G is
    * bipartite (without knowing the sets X and Y in advance).
    *
    * Kasutatud allikad:
    * https://www.geeksforgeeks.org/m-coloring-problem-backtracking-5/
    * https://www.geeksforgeeks.org/bipartite-graph/
    */
```

```java
/**
 * Vertex represents one vertex in the graph.
 */
class Vertex {

   private final String id;
   private Vertex next;
   private Arc first;
   private int info = 0;

   Vertex (String s, Vertex v, Arc e) {
      id = s;
      next = v;
      first = e;
   }

   Vertex (String s) {
      this (s, v: null, e: null);
   }

   @Override
   public String toString() {
      return id;
   }
}


/** Arc represents one arrow in the graph. Two-directional edges are
 * represented by two Arc objects (for both directions).
 */
class Arc {

   private final String id;
   private Vertex target;
   private Arc next;
   private final int info = 0;

   Arc (String s, Vertex v, Arc a) {
      id = s;
      target = v;
      next = a;
   }

   Arc (String s) {
      this (s, v: null, a: null);
   }

   @Override
   public String toString() {
      return id;
   }
}
```

```
class Graph {

   private final String id;
   private Vertex first;
   private int info = 0;
   private int verticeCount;
   private int edgeCount;

   Graph (String s, Vertex v) {
      id = s;
      first = v;
   }

   Graph (String s) { this (s,  v: null); }

   @Override
   public String toString() {
      String nl = System.getProperty ("line.separator");
      StringBuffer sb = new StringBuffer (nl);
      sb.append (id);
      sb.append (nl);
      Vertex v = first;
      while (v != null) {
         sb.append (v.toString());
         sb.append (" -->");
         Arc a = v.first;
         while (a != null) {
            sb.append (" ");
            sb.append (a.toString());
            sb.append (" (");
            sb.append (v.toString());
            sb.append ("->");
            sb.append (a.target.toString());
            sb.append (")");
            a = a.next;
         }
         sb.append (nl);
         v = v.next;
      }
      return sb.toString();
   }
```

```java
public Vertex createVertex (String vid) {
   Vertex res = new Vertex (vid);
   res.next = first;
   first = res;
   return res;
}

public Arc createArc (String aid, Vertex from, Vertex to) {
   Arc res = new Arc (aid);
   res.next = from.first;
   from.first = res;
   res.target = to;
   return res;
}

/**
 * Create a connected undirected random tree with n vertices or create a
 * connected undirected bipartite tree with n vertices.
 * Each new vertex is connected to some random existing vertex or to a
 * designated vertex when creating a bipartite graph.
 * @param n number of vertices added to this graph.
 * @param bipartite true if the graph created from this tree is going to be bipartite.
 */
public void createRandomTree (int n, boolean bipartite) {
   if (n <= 0)
      return;
   Vertex[] varray = new Vertex [n];
   for (int i = 0; i < n; i++) {
      varray [i] = createVertex ( vid: "v" + (n - i));
      if (i > 0) {
         int vnr = 0;
         if (bipartite) {
            vnr = i - 1;
         } else {
            vnr = (int) (Math.random() * i);
         }
         createArc( aid: "a" + varray[vnr].toString() + "_"
                + varray[i].toString(), varray[vnr], varray[i]);
         createArc( aid: "a" + varray[i].toString() + "_"
                + varray[vnr].toString(), varray[i], varray[vnr]);
      }
   }
}
```

```java
/**
 * Create an adjacency matrix of this graph.
 * Side effect: corrupts info fields in the graph
 * @return adjacency matrix
 */
public int[][] createAdjMatrix() {
   info = 0;
   Vertex v = first;
   while (v != null) {
       v.info = info++;
      v = v.next;
   }
   int[][] res = new int [info][info];
   v = first;
   while (v != null) {
      int i = v.info;
      Arc a = v.first;
      while (a != null) {
         int j = a.target.info;
         res [i][j]++;
         a = a.next;
      }
      v = v.next;
   }
   return res;
}
```

```java
/**
 * Create a connected simple (undirected, no loops, no multiple
 * arcs) random graph or connected simple bipartite graph with n vertices and m edges.
 * @param n number of vertices
 * @param m number of edges
 * @param bipartite boolean to choose if the graph created is going to be bipartite.
 */
public void createRandomSimpleGraph (int n, int m, boolean bipartite) {
   if (n <= 0)
      return;
   if (n > 2500)
      throw new IllegalArgumentException ("Too many vertices: " + n);
   if (m < n-1 || m > n*(n-1)/2)
      throw new IllegalArgumentException
         ("Impossible number of edges: " + m);
   this.verticeCount = n;
   this.edgeCount = m;
   first = null;
   createRandomTree (n, bipartite);       // n-1 edges created here
   Vertex[] vert = new Vertex [n];
   Vertex v = first;
   int c = 0;
   while (v != null) {
      vert[c++] = v;
      v = v.next;
   }
   int[][] connected = createAdjMatrix();
   int edgeCount = m - n + 1;  // remaining edges
   if (bipartite) {
      bipartiteEdgeAddition(edgeCount, connected, vert, n); //Fill remaining edges if bipartite graph.
   } else {
      while (edgeCount > 0) {
         int i = (int) (Math.random() * n);  // random source
         int j = (int) (Math.random() * n);  // random target
         if (i == j)
            continue;  // no loops
         if (connected[i][j] != 0 || connected[j][i] != 0)
            continue;  // no multiple edges
         Vertex vi = vert[i];
         Vertex vj = vert[j];
         createArc( aid: "a" + vi.toString() + "_" + vj.toString(), vi, vj);
         connected[i][j] = 1;
         createArc( aid: "a" + vj.toString() + "_" + vi.toString(), vj, vi);
         connected[j][i] = 1;
         edgeCount--;  // a new edge happily created
      }
   }
}
```

```java
/**
 * Add missing connections to the graph without making the graph non-bipartite.
 * @param edgeCount number of remaining edges to create
 * @param connected adjacency matrix
 * @param vert array of vertices
 * @param n number of vertices
 */
public void bipartiteEdgeAddition(int edgeCount, int[][] connected, Vertex[] vert, int n) {
   boolean pairCheck = false;
   for (int i = 0; i < n; i++) {
      pairCheck = i % 2 == 0;
      for (int j = 0; j < n - 1; j++) {
         if (edgeCount > 0) {
            int jr = j;
            if (pairCheck && j % 2 == 0) { jr++; }
            if (!pairCheck && j % 2 != 0 && i != 0) { jr++; }
            if (i == jr)
               continue;  // no loops
            if (connected[i][jr] != 0 || connected[j][i] != 0)
               continue;  // no multiple edges
            Vertex vi = vert[i];
            Vertex vj = vert[jr];
            createArc( aid: "a" + vi.toString() + "_" + vj.toString(), vi, vj);
            connected[i][jr] = 1;
            createArc( aid: "a" + vj.toString() + "_" + vi.toString(), vj, vi);
            connected[jr][i] = 1;
            edgeCount--;  // a new edge happily created
         }
      }
   }
}
```

```java
    /**
     * Check if the current graph is bipartite or not by dividing the vertices in the graph into
     * two separate sets differentiated by colors.
     * @return true if bipartite
     */
    public boolean checkIfGraphIsBipartite() {
        if (edgeCount > (verticeCount * verticeCount) / 4) { // Number of edges is too big to be bipartite.
            System.out.println("Graph: " + id + " is not bipartite.\nImpossible number of edges: " +
                    edgeCount + " for a bipartite graph consisting" +
                    " of " + verticeCount + " vertices.\nMaximum number of edges " +
                    "for this graph to be bipartite would be: " + (verticeCount * verticeCount) / 4);
            return false;
        }
        if (verticeCount - edgeCount == 1) { //Is always bipartite.
            System.out.println("Graph: " + id + " with " + verticeCount +
                    " vertices and " + edgeCount + " edges is bipartite. ");
            return true;
        }
        int[][] connections = createAdjMatrix();  //Array of connections.
        Color[] colors = new Color[connections.length]; //Array of colored vertices.
        Arrays.fill(colors, Color.WHITE);  //default color
        for (int v = 0; v < colors.length - 1; v++) {  //v stands for start point Vertex
              for (int e = 0; e < colors.length; e++) { //e stands for end point Vertex
                 if (connections[v][e] == 1) {      //Check if connection exists.
                    if (colors[e] == Color.WHITE){
                          if (colors[v] == Color.RED) {
                             colors[e] = Color.BLUE;
                          } else {
                           colors[e] = Color.RED;
                          }

                       if (colors[v] == colors[e]) {
                          System.out.println("Graph: " + id + " with " + verticeCount +
                                  " vertices and " + edgeCount + " edges is not bipartite. ");

                          return false;
                       }
                    }
                 }
              }
        }
        System.out.println("Graph: " + id + " with " + verticeCount +
                " vertices and " + edgeCount + " edges is bipartite. ");
        return true;
    }
}
}
```

LISA 2: GraphTaskTest.java

```java
import static org.junit.Assert.*;
import org.junit.Test;
import java.util.*;

/** Testklass.
 * @author jaanus
 */
public class GraphTaskTest {

   GraphTask gt = new GraphTask();

   @Test (timeout=20000)
   public void testSmallBipartiteGraph() {
      GraphTask.Graph graph = gt.new Graph( s: "G");
      graph.createRandomSimpleGraph( n: 10, m: 25, bipartite: true);
      System.out.println(graph);
      long startTime = System.nanoTime();
      assertTrue ( message: "This graph is suppose to be bipartite.", graph.checkIfGraphIsBipartite());
      long endTime = System.nanoTime();
      long duration = endTime - startTime;
      System.out.println("Time taken: " + duration / 1000000 + " ms");
   }

   @Test (timeout=20000)
   public void testMediumBipartiteGraph() {
      GraphTask.Graph graph = gt.new Graph( s: "G");
      graph.createRandomSimpleGraph( n: 150, m: 5625, bipartite: true);
      System.out.println(graph);
      long startTime = System.nanoTime();
      assertTrue ( message: "This graph is suppose to be bipartite.", graph.checkIfGraphIsBipartite());
      long endTime = System.nanoTime();
      long duration = endTime - startTime;
      System.out.println("Time taken: " + duration / 1000000 + " ms");
   }
```

```java
@Test (timeout=20000)
public void testLargeBipartiteGraph() {
    GraphTask.Graph graph = gt.new Graph( s: "G");
    graph.createRandomSimpleGraph( n: 2500, m: 1562500, bipartite: true);
    System.out.println(graph);
    long startTime = System.nanoTime();
    assertTrue ( message: "This graph is suppose to be bipartite.", graph.checkIfGraphIsBipartite());
    long endTime = System.nanoTime();
    long duration = endTime - startTime;
    System.out.println("Time taken: " + duration / 1000000 + " ms");
}

@Test (timeout = 20000)
public void testImpossibleBipartiteGraph() {
    GraphTask.Graph graph = gt.new Graph( s: "G");
    graph.createRandomSimpleGraph( n: 2000, m: 1562500, bipartite: false);
    assertFalse(graph.checkIfGraphIsBipartite());
}

@Test (timeout=20000)
public void testLargeRandomButMustBeBipartiteGraph() {
    GraphTask.Graph graph = gt.new Graph( s: "G");
    graph.createRandomSimpleGraph( n: 2500, m: 2499, bipartite: false);
    long startTime = System.nanoTime();
    assertTrue ( message: "This graph is suppose to be bipartite.", graph.checkIfGraphIsBipartite());
    long endTime = System.nanoTime();
    long duration = endTime - startTime;
    System.out.println("Time taken: " + duration / 1000000 + " ms");
}
```