TALLINNA TEHNIKAÜLIKOOL

Infotehnoloogia teaduskond

Vladislav Konstantinov

193598IADB

Find two vertices that have the longest distance between them

Individuaaltöö aines "Algoritmid ja andmestruktuurid"

Juhendaja: Jaanus Pöial

Tallinn 2020

1.Ülesande püstitus

Probleemi sõnastamine:

Inglise keeles:

Find two vertices that have the longest distance between them in undirected, unweighted, simple graph.

Eesti keeles:

Leia kaks tippu, mille vahemaa on kõige pikem oreienteerimatas, kaalumatas, lihtsas graafis.

Definitsioonid:

Oreinteerimata graaf - Graaf, mille servad(Arc) ei ole suunatud.

Kaalumata graaf – Graaf, mille seosed(Arc) ei omistatud mingi väärtusi.

Lihtne graaf – Graaf, millel pole silmuseid ja mitmekordseid serva.

Selgitus:

Pikim tee probleem on: Antud graafis maksimaalse pikkusega lihtsa tee leidmise. Tee nimetatakse lihtsaks, kui selles pole korduvaid tippe. Tee pikkust saab mõõta servade arvu kaalumatas graafis.

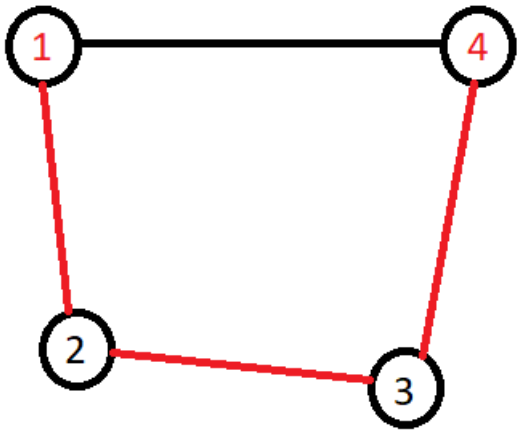
Näide:

Sellis graafis tippudega 1, 2, 3, 4

Kõige pikem tee same ettekujutada nagu

1 → 2 → 3 → 4

Punktide 1 ja 4 vahel, ja tema pikkus on 3.



3.Lahenduse kirjeldus

Kirjutasin minu algoritmi variant, mis leaib kõige kauem tee.

Peamise idee jaoks võtsin seda https://foxford.ru/wiki/informatika/postroenie-gamiltonova-tsikla veebi lehel esitatud algoritmi, mis leiab antud tipust graafilt Hamiltoni tee. Kuid muutsin algoritmi nii, et see leiab graafilt antud tipust lähtuvalt võimalikult pika tee ja kirjutab selle üles, isegi kui see pole Hamiltoni tee.

a.See funktsioon on abistav ja aitab leida kauem tee funktsioonile findlongestway (), funktisoon kasutab Adjugate matrix.

```
/**
 * Find longest path from start point curr.
 * Auxiliary function
 * @param curr - Start point.
 * @param visited - list with visited graphs.
 * @param path - path that was already passed.
 * @return boolean
 */
      public boolean findlong(int curr, boolean[] visited, List<Integer> path) {
         int [][] matrix = this.createAdjMatrix();
         path.add(curr);
         visited[curr] = true;
         int n = matrix.length;
         if (path.size() == matrix.length) {
            return true;
         }

         for (int i = 0; i < n; i++) {
            if (matrix[curr][i] == 1 && !visited[i]) {
               if (findlong(i,  visited, path)) {
                  return true;
               }

            }
         }
         visited[curr] = false;
         return true;
      }
```

b.Selles funktioonis kasutatakse funktsioon findlong(), seda funktioon leiab kõik grafi teed mis on pikem ja return neid nagu List<List<Arc>>.

```
      /**
       * Find all longest ways in this graph.
       * @return list with lists of Arc that form longest ways in this graph.
       */
      public List<List<Arc>> findlongestway(){
         Graph g = this;
         int[][] matrix = g.createAdjMatrix();
         boolean[] visited;

         ArrayList<Integer> path = new ArrayList<>();
         List<Integer> longestpath = new ArrayList<>();
         List<List<Integer>> pathes = new ArrayList<>();
```

```
            List<List<Integer>> longestpathes = new ArrayList<>();

            for(int i = 0; i < matrix.length; i++) {
               path.clear();
               visited = new boolean[matrix.length];

               findlong(i, visited, path);

               if(i == 1) {
                  longestpath = (List<Integer>) path.clone();
               } else if(longestpath.size() < path.size()){
                  longestpath = (List<Integer>) path.clone();
               }
               pathes.add((List<Integer>) path.clone());

            }
            int max = 0;
            for(List<Integer> list: pathes) {
               if (list.size() > max) {
                  max = list.size();
               }
            }
            for(List<Integer> list: pathes) {
               if (list.size() == max) {
                  longestpathes.add(list);
               }
            }

            List<List<Arc>> pathsarcs = new ArrayList<>();
            for(List<Integer> firstpath: longestpathes) {
               List<Arc> arcs = new ArrayList<>();
               for (int arci = 0; arci < firstpath.size(); arci++) {
                  Vertex v = first;
                  while (v.next != null && firstpath.get(arci) != v.info) {
                     v = v.next;
                  };
                  if(arci < firstpath.size() - 1) {

                     Arc a = v.first;
                     while (a != null && a.target != null && a.target.info !=
firstpath.get(arci + 1)) {
                        a = a.next;
                     }

                     arcs.add(a);

                  }
               }

               pathsarcs.add(arcs);
            }
//             System.out.println(pathsarcs.toString());
            return pathsarcs;
         }
```

Põhi idee on selline:
Lõin `boolean[] visited`, kuhu on märgitatud juba külastatud tipud(juba külastatud tipud on märgitud nagu true), listi indeksid ja tipude id on seotud.
Lõin `List<List<Integer>> pathes`, kus ok kõik teed, mis leia funksion `findlong()`.

Lõin `List<Integer> longestpath`, kuhu on salvestatakse kõige pikem tee ühest tipust.
Lõin `List<List<Integer>> longestpathes`, kus salvestatakse kõik kõige pikem teed.

1) Algoritm viib läbi graafi kõik tipud ja saadab need funktsioonile findlong (), samuti on juba külastatud tipud märgitud boolean [] visited loendis ja läbitud path loendis List<Integer> path. Ta salvestab graafi teed `List<List<Integer>> pathes`
2) Pärast seda ta leidis kõige pikem teed.
3) Kui kõik teed on leitud, algoritm teisendab tee tippude loendist Arci loendiks ja salvestab neid listide `longestpathes`.
4) `Return neid nagu List<List<Arc>>.`

   `c.Seda funktions represent meie resultaadid mis leiti funktsioonist findlongestway() nagu string, return samuti List<List<Arc>> kus on leitud teed.`

```
public List<List<Arc>> representLongestway() {
         Graph g = this;
         String nl = System.getProperty("line.separator");
         StringBuffer sb = new StringBuffer(nl);
         List<List<Arc>> listofpathes = findlongestway();
         sb.append(nl);
         sb.append("The longest ways length in not oriented, unweighted graph ");
         sb.append(g.id);
         sb.append(" is: ");
         sb.append(listofpathes.get(0).size() - 1);
         sb.append(" arcs");
         sb.append(nl);
         int count = 0;
         for(List<Arc> firstpath: listofpathes) {
            StringBuffer sb2 = new StringBuffer(nl);
            count++;
            sb.append(count);
            sb.append(String.format(". way is: "));
            sb.append(nl);
            for (int arci = 0; arci < firstpath.size(); arci++) {
               if(arci == 0){
                  sb2.append("Vertexes: ");
                  sb2.append(firstpath.get(arci).target.id);
                  sb2.append(" ==> ");
               }
               if(arci == firstpath.size() - 1){
                  sb2.append(firstpath.get(arci).target.id);
               }
               if (arci != 0) {
                  sb.append(" --> ");
               }
               sb.append(firstpath.get(arci).target.id);
               if(arci < firstpath.size() - 1) {

                  sb.append(" (");
                  sb.append(firstpath.get(arci + 1).id);
                  sb.append(")");
               }
            }
            sb.append(sb2);
            sb.append(nl);
//             sb.append(nl);
         }
         System.out.println(sb.toString());
         return listofpathes;
      }
```

4.Programmi kasutamisjuhend

1. Kõigepealt peate looma graafi klass, kus parameter on grrafi nimi.

   Näide:
   ```
   Graph g = new Graph( s: "G");
   ```

2. Nüüd luuate oreienteerimata, kaalumata, lihtne graaf funktsiooniga createRandomSimpleGraph(n, m) parametriga n ja m.
   Kus n see on tipude arv.
   m on arci arv.

   Näide:
   ```
   g.createRandomSimpleGraph( n: 6,  m: 9);
   ```
   NB!
   Arc (m) arv peab olema vähem või võrdne kui n * (n – 1) / 2.
   n ei saa olla vähem kui 1.
   m peab olema vähem või võrde 2500.

3.
   a. Kui soovite vastuse stringi esitust, siis kasutage funktsioon representLongestway(), et leida tippud, mille vahemaa on kõige pikem ja printida neid.
      Näide:
      ```
      g.representLongestway();
      ```
   b. Kui teil on vaja ainult vastus nagu list kus on tee mis esitatud nagu graafide arcid, Siis sate kasutada funktsioon `findlongestway()`.

      Näide:
      ```
      g.findlongestway();
      ```

5.Testimiskava

NB!

Sellepärast, et algritmi keerukus on np-keeruline, on saadud algoritmi vastuste õigsust võimalik kontrollida ainult läbides graafikus kõik võimalikud teed ja leida pikim ning võrrelda saadud algoritmi vastusega mis eeldab teist "usaldusväärset" algoritmi millega on võimalik võrrelda vastused.

Need testid kontrollivad ainult programmi toimivust.

1.Graaf G 2 tippuga ja 1 arciga.

Graaf:

```
G
v1 --> av1_v2 (v1->v2)
v2 --> av2_v1 (v2->v1)
```

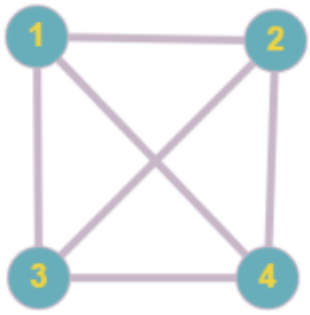
Programmi vastus:

```
The longest ways length in not oriented, unweighted graph G is: 1 arcs
1. way is:
v1 (av1_v2) --> v2
Vertexes: v1 ==> v2
2. way is:
v2 (av2_v1) --> v1
Vertexes: v2 ==> v1
```

2.Graaf G 4 tippuga ja 6 arciga.

Graaf:

```
G
v1 --> av1_v3 (v1->v3) av1_v4 (v1->v4) av1_v2 (v1->v2)
v2 --> av2_v4 (v2->v4) av2_v1 (v2->v1) av2_v3 (v2->v3)
v3 --> av3_v1 (v3->v1) av3_v2 (v3->v2) av3_v4 (v3->v4)
v4 --> av4_v2 (v4->v2) av4_v1 (v4->v1) av4_v3 (v4->v3)
```



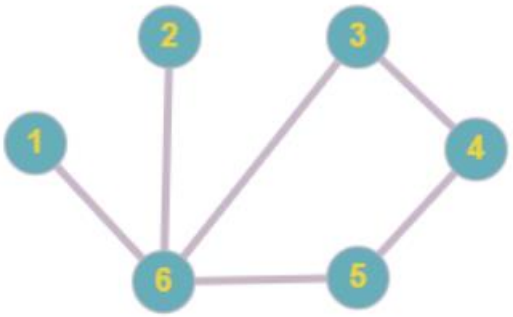
Programmi vastus:

```
The longest ways length in not oriented, unweighted graph G is: 3 arcs
1. way is:
v1 (av1_v2) --> v2 (av2_v3) --> v3 (av3_v4) --> v4
Vertexes: v1 ==> v4
2. way is:
v2 (av2_v1) --> v1 (av1_v3) --> v3 (av3_v4) --> v4
Vertexes: v2 ==> v4
3. way is:
v3 (av3_v1) --> v1 (av1_v2) --> v2 (av2_v4) --> v4
Vertexes: v3 ==> v4
4. way is:
v4 (av4_v1) --> v1 (av1_v2) --> v2 (av2_v3) --> v3
Vertexes: v4 ==> v3
```

3.Graaf G 6 tippuga ja 6 arciga.

Graaf:

```
G
v1 --> av1_v6 (v1->v6)
v2 --> av2_v6 (v2->v6)
v3 --> av3_v4 (v3->v4) av3_v6 (v3->v6)
v4 --> av4_v3 (v4->v3) av4_v5 (v4->v5)
v5 --> av5_v4 (v5->v4) av5_v6 (v5->v6)
v6 --> av6_v1 (v6->v1) av6_v2 (v6->v2) av6_v3 (v6->v3) av6_v5 (v6->v5)
```



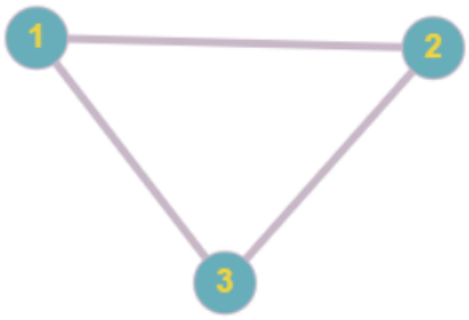
Programmi vastus:

```
The longest ways length in not oriented, unweighted graph G is: 4 arcs
1. way is:
v3 (av3_v4) --> v4 (av4_v5) --> v5 (av5_v6) --> v6 (av6_v1) --> v1
Vertexes: v3 ==> v1
2. way is:
v5 (av5_v4) --> v4 (av4_v3) --> v3 (av3_v6) --> v6 (av6_v1) --> v1
Vertexes: v5 ==> v1
```

4.Graaf G 3 tippuga ja 3 arciga.

Graaf:

```
G
v1 --> av1_v2 (v1->v2) av1_v3 (v1->v3)
v2 --> av2_v1 (v2->v1) av2_v3 (v2->v3)
v3 --> av3_v1 (v3->v1) av3_v2 (v3->v2)
```



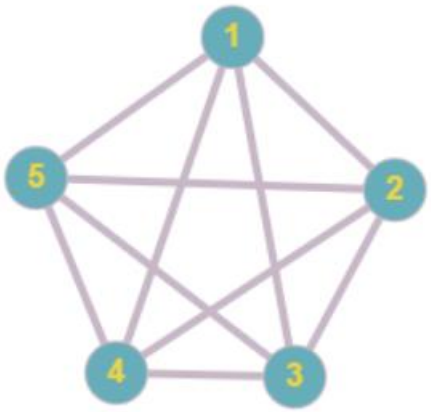
Programmi vastus:

```
The longest ways length in not oriented, unweighted graph G is: 2 arcs
1. way is:
v1 (av1_v2) --> v2 (av2_v3) --> v3
Vertexes: v1 ==> v3
2. way is:
v2 (av2_v1) --> v1 (av1_v3) --> v3
Vertexes: v2 ==> v3
3. way is:
v3 (av3_v1) --> v1 (av1_v2) --> v2
Vertexes: v3 ==> v2
```

5.Graaf G 5 tippuga ja 10 arciga.

Graaf:

```
G
v1 --> av1_v2 (v1->v2) av1_v4 (v1->v4) av1_v5 (v1->v5) av1_v3 (v1->v3)
v2 --> av2_v5 (v2->v5) av2_v1 (v2->v1) av2_v4 (v2->v4) av2_v3 (v2->v3)
v3 --> av3_v4 (v3->v4) av3_v1 (v3->v1) av3_v2 (v3->v2) av3_v5 (v3->v5)
v4 --> av4_v3 (v4->v3) av4_v2 (v4->v2) av4_v1 (v4->v1) av4_v5 (v4->v5)
v5 --> av5_v2 (v5->v2) av5_v1 (v5->v1) av5_v3 (v5->v3) av5_v4 (v5->v4)
```



Programmi vastus:

```
The longest ways length in not oriented, unweighted graph G is: 4 arcs
1. way is:
v1 (av1_v2) --> v2 (av2_v3) --> v3 (av3_v4) --> v4 (av4_v5) --> v5
Vertexes: v1 ==> v5
2. way is:
v2 (av2_v1) --> v1 (av1_v3) --> v3 (av3_v4) --> v4 (av4_v5) --> v5
Vertexes: v2 ==> v5
3. way is:
v3 (av3_v1) --> v1 (av1_v2) --> v2 (av2_v4) --> v4 (av4_v5) --> v5
Vertexes: v3 ==> v5
4. way is:
v4 (av4_v1) --> v1 (av1_v2) --> v2 (av2_v3) --> v3 (av3_v5) --> v5
Vertexes: v4 ==> v5
5. way is:
v5 (av5_v1) --> v1 (av1_v2) --> v2 (av2_v3) --> v3 (av3_v4) --> v4
Vertexes: v5 ==> v4
```

6.Kasutatud allikad

1. https://foxford.ru/wiki/informatika/postroenie-gamiltonova-tsikla
2. https://en.wikipedia.org/wiki/Critical_path_method
3. McHugh,A. James , Algorithmic Graph Theory (2010)
4.https://et.wikipedia.org/wiki/Graaf
5. https://graphonline.ru/

7.Programmi täielik tekst

https://github.com/Vlagod/Algoritmid/blob/main/kt6/src/GraphTask.java

```
import java.util.*;
//Find two vertices that have the longest distance between them
/** 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();
//      throw new RuntimeException ("Nothing implemented yet!"); // delete this
   }

   /**
    * Actual main method to run examples and everything.
    */
   public void run() {
      Graph g = new Graph("G");
      g.createRandomSimpleGraph(5, 10);
      System.out.println(g);
      System.out.println(Arrays.deepToString(g.createAdjMatrix()));
//      System.out.println(Arrays.deepToString(g.createAdjMatrix()));
//      g.findlongestway();
      long time = System.currentTimeMillis();
//      System.out.println(g.findlongestway());
      System.out.println(g.representLongestway());
      System.out.println(String.format("Time: %s milli seconds",
System.currentTimeMillis() - time));

      Graph c = new Graph("C");
      c.createRandomSimpleGraph(50, 51);
      System.out.println(c);
      time = System.currentTimeMillis();
//      System.out.println(c.findlongestway());
      System.out.println(c.representLongestway());
      System.out.println(String.format("Time: %s milli seconds",
System.currentTimeMillis() - time));

      Graph b = new Graph("B");
      b.createRandomSimpleGraph(100, 500);
      System.out.println(b);
      time = System.currentTimeMillis();
//      System.out.println(b.findlongestway());
      System.out.println(b.representLongestway());
      System.out.println(String.format("Time: %s milli seconds",
System.currentTimeMillis() - time));

g.representLongestway();
```

```
g.findlongestway();

      Graph d = new Graph("D");
      d.createRandomSimpleGraph(71, 2485);
      System.out.println(d);
      time = System.currentTimeMillis();
//      System.out.println(d.findlongestway());
      System.out.println(d.representLongestway());
      System.out.println(String.format("Time: %s milli seconds",
System.currentTimeMillis() - time));


      Graph a = new Graph("A");
      a.createRandomSimpleGraph(500, 1000);
      System.out.println(a);
//      time = System.currentTimeMillis();
////      a.findlongestway();
//      System.out.println(String.format("Time: %s milli seconds",
System.currentTimeMillis() - time));
      time = System.currentTimeMillis();
//      System.out.println(a.findlongestway());
      System.out.println(a.representLongestway());
      System.out.println(String.format("Time: %s milli seconds",
System.currentTimeMillis() - time));

      // TODO!!! Your experiments here
   }

   // TODO!!! add javadoc relevant to your problem
   /**
    * Program implements an application that
    * Create random simple graph
    * Finds critical paths in the graph
    * Since the complexity algorithm is np algorithm can take a relatively long time
to solve
    * @author  Vladislav Konstantinov
    * @author  Jaanus Pöial
    * @version 1.0
    * @since   2020-11-20
    */
   class Vertex {

      private String id;
      private Vertex next;
      private Arc first;
      private int info = 0;
      // You can add more fields, if needed

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

      Vertex(String s) {
         this(s, null, null);
      }

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

```
        }

        // TODO!!! Your Vertex methods here!
    }


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

        private String id;
        private Vertex target;
        private Arc next;
        private int info = 0;
        // You can add more fields, if needed

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

        Arc(String s) {
            this(s, null, null);
        }

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

        // TODO!!! Your Arc methods here!
    }


    class Graph {

        private String id;
        private Vertex first;
        private int info = 0;
        // You can add more fields, if needed

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

        Graph(String s) {
            this(s, 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;
//          System.out.println(v.toString());
```

```
        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();
    }

    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.
     * Each new vertex is connected to some random existing vertex.
     *
     * @param n number of vertices added to this graph
     */
    public void createRandomTree(int n) {
        if (n <= 0)
            return;
        Vertex[] varray = new Vertex[n];
        for (int i = 0; i < n; i++) {
            varray[i] = createVertex("v" + String.valueOf(n - i));
            if (i > 0) {
                int vnr = (int) (Math.random() * i);
                createArc("a" + varray[vnr].toString() + "_"
                        + varray[i].toString(), varray[vnr], varray[i]);
                createArc("a" + varray[i].toString() + "_"
                        + varray[vnr].toString(), varray[i], varray[vnr]);
            } else {
            }
        }
    }

    /**
     * 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;
    }

    /**
     * Create a connected simple (undirected, no loops, no multiple
     * arcs) random graph with n vertices and m edges.
     *
     * @param n number of vertices
     * @param m number of edges
     */
    public void createRandomSimpleGraph(int n, int m) {
       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);
       first = null;
       createRandomTree(n);       // 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
       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("a" + vi.toString() + "_" + vj.toString(), vi, vj);
          connected[i][j] = 1;
          createArc("a" + vj.toString() + "_" + vi.toString(), vj, vi);
```

```
            connected[j][i] = 1;
            edgeCount--;  // a new edge happily created
         }
      }



      // TODO!!! Your Graph methods here! Probably your solution belongs here.
      /**
       * Represent all longest ways in this graph that was findby function
findlongestway(g).
       * @return String with information about longest ways in this graph g.
       */
      public List<List<Arc>> representLongestway() {
         Graph g = this;
         String nl = System.getProperty("line.separator");
         StringBuffer sb = new StringBuffer(nl);
         List<List<Arc>> listofpathes = findlongestway();
         sb.append(nl);
         sb.append("The longest ways length in not oriented, unweighted graph ");
         sb.append(g.id);
         sb.append(" is: ");
         sb.append(listofpathes.get(0).size());
         sb.append(" arcs");
         sb.append(nl);
         int count = 0;
         for(List<Arc> firstpath: listofpathes) {
            StringBuffer sb2 = new StringBuffer(nl);
            count++;
            sb.append(count);
            sb.append(String.format(". way is: "));
            sb.append(nl);
            for (int arci = 0; arci < firstpath.size(); arci++) {
               if(arci == 0){
                  sb2.append("Vertexes: ");
                  sb2.append(firstpath.get(arci).toString().substring(1,
firstpath.get(arci).toString().indexOf("_")));
                  sb2.append(" ==> ");
//                  sb.append(firstpath.get(arci).next.toString());
                  sb.append(firstpath.get(arci).toString().substring(1,
firstpath.get(arci).toString().indexOf("_")));
                  sb.append(" (");
                  sb.append(firstpath.get(arci).id);
                  sb.append(")");
                  sb.append(" --> ");
               }
               if(arci == firstpath.size() - 1){
                  sb2.append(firstpath.get(arci).target.id);
               }
               if (arci != 0) {
                  sb.append(" --> ");
               }
               sb.append(firstpath.get(arci).target.id);
               if(arci < firstpath.size() - 1) {

                  sb.append(" (");
                  sb.append(firstpath.get(arci + 1).id);
                  sb.append(")");
               }
            }
            sb.append(sb2);
            sb.append(nl);
```

```
//             sb.append(nl);
         }
         System.out.println(sb.toString());
         return listofpathes;
      }

      /**
       * Find all longest ways in this graph.
       * @return list with lists of Arc that form longest ways in this graph.
       */
      public List<List<Arc>> findlongestway(){
         Graph g = this;
         int[][] matrix = g.createAdjMatrix();
         boolean[] visited;

         ArrayList<Integer> path = new ArrayList<>();
         List<Integer> longestpath = new ArrayList<>();
         List<List<Integer>> pathes = new ArrayList<>();
         List<List<Integer>> longestpathes = new ArrayList<>();

         for(int i = 0; i < matrix.length; i++) {
            path.clear();
            visited = new boolean[matrix.length];

            findlong(i, visited, path);

            if(i == 1) {
               longestpath = (List<Integer>) path.clone();
            } else if(longestpath.size() < path.size()){
               longestpath = (List<Integer>) path.clone();
            }
            pathes.add((List<Integer>) path.clone());

         }
         int max = 0;
         for(List<Integer> list: pathes) {
            if (list.size() > max) {
               max = list.size();
            }
         }
         for(List<Integer> list: pathes) {
            if (list.size() == max) {
               longestpathes.add(list);
            }
         }

         List<List<Arc>> pathsarcs = new ArrayList<>();
         for(List<Integer> firstpath: longestpathes) {
            List<Arc> arcs = new ArrayList<>();
            for (int arci = 0; arci < firstpath.size(); arci++) {
               Vertex v = first;
               while (v.next != null && firstpath.get(arci) != v.info) {
                  v = v.next;
               };
               if(arci < firstpath.size() - 1) {

                  Arc a = v.first;
                  while (a != null && a.target != null && a.target.info !=
firstpath.get(arci + 1)) {
                     a = a.next;
                  }
```

```
                    arcs.add(a);

                }
            }

            pathsarcs.add(arcs);
        }
//          System.out.println(pathsarcs.toString());
        return pathsarcs;
    }

    /**
     * Find longest path from start point curr.
     * Auxiliary function
     * @param curr - Start point.
     * @param visited - list with visited graphs.
     * @param path - path that was already passed.
     * @return boolean
     */
//
//inspireeritud sellest koodist https://foxford.ru/wiki/informatika/postroenie-gamiltonova-tsikla, kuid muutis seda palju minu ülesandade jaoks.
    public boolean findlong(int curr, boolean[] visited, List<Integer> path) {
        int [][] matrix = this.createAdjMatrix();
        path.add(curr);
        visited[curr] = true;
        int n = matrix.length;
        if (path.size() == matrix.length) {
            return true;
        }

        for (int i = 0; i < n; i++) {
            if (matrix[curr][i] == 1 && !visited[i]) {
                if (findlong(i,  visited, path)) {
                    return true;
                }

            }
        }
        visited[curr] = false;
        return true;
    }

}
```

8.lahendusnäidete tulemused

Test 1:

```
G
v1 --> av1_v8 (v1->v8)
v2 --> av2_v6 (v2->v6) av2_v3 (v2->v3)
v3 --> av3_v7 (v3->v7) av3_v2 (v3->v2) av3_v5 (v3->v5)
v4 --> av4_v7 (v4->v7)
v5 --> av5_v8 (v5->v8) av5_v3 (v5->v3) av5_v7 (v5->v7)
v6 --> av6_v2 (v6->v2) av6_v7 (v6->v7)
v7 --> av7_v3 (v7->v3) av7_v4 (v7->v4) av7_v5 (v7->v5) av7_v6 (v7->v6) av7_v8 (v7->v8)
v8 --> av8_v5 (v8->v5) av8_v1 (v8->v1) av8_v7 (v8->v7)

[[0, 0, 0, 0, 0, 0, 0, 1], [0, 0, 1, 0, 0, 1, 0, 0], [0, 1, 0, 0, 1, 0, 1, 0], [0, 0, 0, 0, 0, 0, 1, 0], [0, 0, 1, 0, 0, 0, 1, 1]


The longest ways length in not oriented, unweighted graph G is: 7 arcs
1. way is:
v1 (av1_v8) --> v8 (av8_v5) --> v5 (av5_v3) --> v3 (av3_v2) --> v2 (av2_v6) --> v6 (av6_v7) --> v7 (av7_v4) --> v4
Vertexes: v1 ==> v4

[[av1_v8, av8_v5, av5_v3, av3_v2, av2_v6, av6_v7, av7_v4]]
Time: 36 milli seconds
```

Test 2:

```
G
v1 --> av1_v4 (v1->v4)
v2 --> av2_v5 (v2->v5)
v3 --> av3_v5 (v3->v5)
v4 --> av4_v1 (v4->v1) av4_v5 (v4->v5)
v5 --> av5_v2 (v5->v2) av5_v3 (v5->v3) av5_v4 (v5->v4)

[[0, 0, 0, 1, 0], [0, 0, 0, 0, 1], [0, 0, 0, 0, 1], [1, 0, 0, 0, 1], [0, 1, 1, 1, 0]]


The longest ways length in not oriented, unweighted graph G is: 3 arcs
1. way is:
v1 (av1_v4) --> v4 (av4_v5) --> v5 (av5_v2) --> v2
Vertexes: v1 ==> v2

[[av1_v4, av4_v5, av5_v2]]
Time: 28 milli seconds
```

Test 3:

```
G
v1 --> av1_v3 (v1->v3)
v2 --> av2_v3 (v2->v3)
v3 --> av3_v1 (v3->v1) av3_v2 (v3->v2)

[[0, 0, 1], [0, 0, 1], [1, 1, 0]]


The longest ways length in not oriented, unweighted graph G is: 2 arcs
1. way is:
v1 (av1_v3) --> v3 (av3_v2) --> v2
Vertexes: v1 ==> v2
2. way is:
v2 (av2_v3) --> v3 (av3_v1) --> v1
Vertexes: v2 ==> v1

[[av1_v3, av3_v2], [av2_v3, av3_v1]]
Time: 39 milli seconds
```