TALLINNA TEHNIKAÜLIKOOL

Infotehnoloogia teaduskond

IT Kolledž

Individuaaltöö aines
Algoritmid ja andmestruktuurid - **ICD0001**

Üliõpilane: Kristjan Koemets

Õpperühm: EAEI

Matrikli nr: 122599

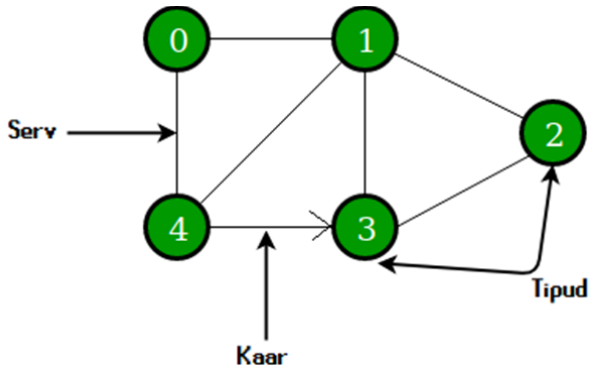
Juhendaja: Jaanus Pöial

Tallinn, **2018**.a.

SISUKORD

1. ÜLESANDE PÜSTITUS

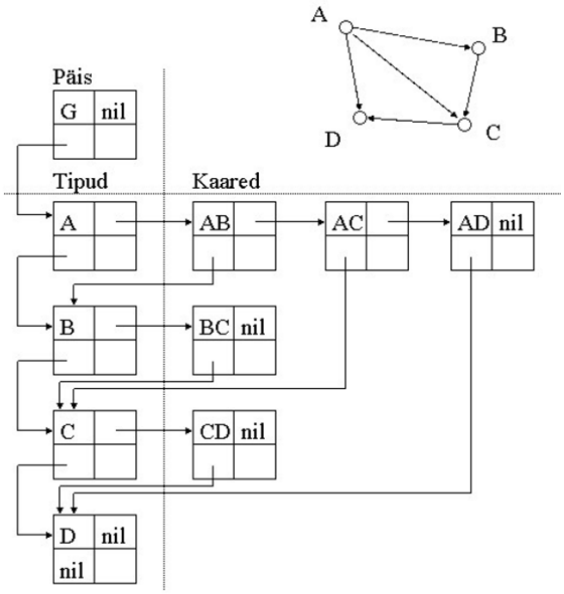
Graaf on andmestruktuur, mis koosneb tippudest, servadest ja/või kaartest, illustreerivalt on näidatud joonisel 1.1. Servad ja kaared seovad omavahel tipupaare, nende erinevuseks on see, et esimesel on liikumissuund kahesuunaline, siis teisel on ühesuunaline.



Joonis 1.1 Graafiline selgitus servade, kaarte ja tippude kohta [1]

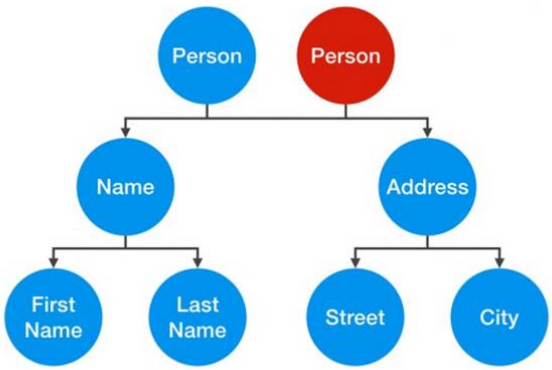
Lihtsaim näide graafi rakendusest on GPS süsteemid. Lähte- ja sihtpunktidena võib vaadelda tippe ning servasid/kaari kui neid ühendavaid teid.

Käesoleva töö eesmärgiks on koostada graafi sügava kloonimise meetod, mis on kooskõlas Java konventsioonidega. Tulemuses peavad olema kõik *Vertex* ja *Arc* tüüpi objektid samuti kloonitud. Ülesande lahendamisel tuleb kasutada etteantud programmitoorikut, mis eeldab, et graafi kujutatakse külgnevusstruktuuriga, põhimõte esitatud joonisel 1.2.



Joonis 1.2 Graafi kujutamine külgnevusstruktuuri abil [2]

Programeerimiskeeles Java defineerib meetodi *clone* juurülemklass java.lang.Object. Alamklassid, mis kasutavad liidest java.lang.Cloneable, tagastavad objektist koopia. Kuigi meetodi ülekatmisel on vaba valik keerukamate algoritmide loomisel, siis Java konventsioon näeb ette, et meetod *clone* tagastab madalkoopia, see tähendab, et luuakse koopia kloneeritavast objektist, kuid koopia *Object* tüüpi isendimuutujad viitavad ikka kloneeritava objekti *Object* tüüpi isendimuutujatele, vt joonist 1.3. Tulemuseks on see, et muutes *Object* tüüpi isendimuutujaid ühes objektis, kajastuvad muutused ka teises objektis. [3]



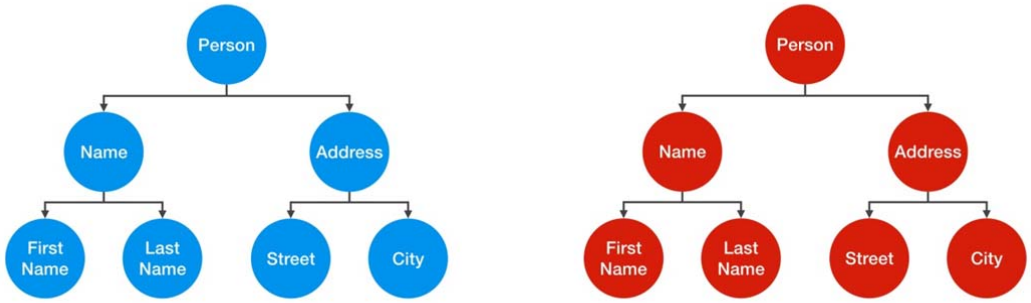
Joonis 1.3 Madalkloneerimisel uue objekti isendimuutujad refereerivad kloneeritud objekti isendimuutujatele [4]

kus sinine ring – objekt enne kloneerimist,

punane ring – objekt peale kloneerimist,

nooled – viidad.

Sügaval kloneerimisel luuakse samuti koopia kloneeritavast objektist, kuid koopia *Object* tüüpi isendimuutujad on erinevad kloneeritava objekti omadest. Objekti koopia isendimuutujad ei viita enam samadele isendimuutujatele, vaid luuakse igast *Object* tüüpi isendimuutujast uus koopia, kui isendimuutujal on omakorda *Object* tüüpi isendimuutujad, siis luuakse ka neist uued koopiad, vt joonist 1.4.



Joonis 1.4 Sügavkloonimisel on kloneeritud objektil isendimuutujad erinevad [4]

Arvestades seda, et lähteülesande tingimustes oli nõutud graafi sügava kloonimise meetodi kooskõlasust Java konventsioonidega, siis meetodi realiseerimisel ei oleks õige meetodi *clone* ülekatmine, kuna meetod eeldab madalkloneerimist. Kasutusele tuleks võtta mingi muu meetodi nimi, näiteks **deepCopy**.

2. LAHENDUSE KIRJELDUS

Programmitoorikus on tipud esitatud klassiga *Vertex*. Selleks, et võrrelda *Vertex* tüüpi objekte oleks vaja katta üle meetodit *equals*. Tippude võrdlemisel võrreldakse *Vertex* klassis defineeritud *String* tüüpi isendimuutujat *id*.

Kaared on programmitoorikus esitatud klassiga *Arc*. Selleks, et võrrelda *Arc* tüüpi objekte oleks vaja katta üle meetodit *equals*. Kaarte võrdlemisel võrreldakse *Arc* klassis defineeritud *String* tüüpi isendimuutujat *id* ja *Vertex* tüüpi isendimuutujat *target*.

Samas programmitoorikus on esitatud graaf klassiga *Graph*. Klassis on defineeritud *String* tüüpi isendimuutuja *id* ja *Vertex* tüüpi isendimuutuja *first*, *int*-tüüpi isendimuutujat *info* antud ülesandes ei käsitleta.

Graafi sügavaks kloneerimiseks on defineeritud meetod *deepCopy*, mille nähtavus on public. Meetod *deepCopy* on jagatud väiksemateks ülesanneteks ehk meetoditeks, mis on *private* nähtavusega meetodid *oldAndCloneVertices* ja *createAndConnectCloneArcs*.

Meetodi *deepCopy* rakendamisel luuakse alguses uus *Graph* tüüpi objekt. Seejärel luuakse paisktabel vanade ja uute tippudega ning omistatakse uuele *Graph* tüüpi objektile *Vertex* tüüpi isendimuuja *first*, milleks on uue tipuahela algus. Graafi lõpliku struktuuri loomiseks luuakse *while*-tsükliga uutele tippudele kaared. Meetod lõpetab graafi koopia tagastamisega.

Meetodi *oldAndCloneVertices* parameetriks on *Graph* tüüpi objekti isendimuutuja *first* ehk tippude ahela esimene tipp. Meetodi ülesandeks on luua uus ahel ning siduda uued tipud vanadega paisktabelis, kus võtmeks on vana tipp (*Vertex* tüüpi) ja väärtuseks on uus tipp (*Vertex* tüüpi). Meetod lõpetab paisktabeli tagastamisega.

Meetodi *createAndConnectCloneArcs* parameetriteks on *Arc* tüüpi objekt, mis on kopeeritava kaarteahela esimene element(info salvestatud tipus) ning paisktabel, mille tagastas meetod o*ldAndCloneVertices*. Meetodi ülesandeks on luua tipule uus kaarteahel, paisktabel aitab leida tippude ahelast uuele kaarele *Vertex* tüüpi isendimuutuja *target*. Meetod lõpetab tipuahela alguse tagastamisega.

3. PROGRAMMI KASUTAMISJUHEND

Graafi sügavkloneerimiseks on klassis *Graph* defineeritud sisendparameetrita isendimeetod **deepCopy**. Meetod tagastab uue *Graph*-tüüpi objekti, mille sisemine struktuur on küll identne objektiga (graafiga), millele isendimeetodit rakendati, kuid kõik tipud ja kaared on uued objektid.

Graafide isomorfismi ehk struktuuri üks-ühene vastavus on kontrollitav klassis *Graph* defineeritud isendimeetodiga **isIsomorphicTo**, millel antakse ette *Graph*-tüüpi sisendparameeter, millega soovitakse võrrelda objekti (graafi), mis rakendas meetodit. Tagastusväärtus on *boolean*-tüüpi, tõeväärtuse korral on graafid isomorfsed.

4. TESTIMISKAVA

Klassi *Vertex* isendimeetodi *equals* toimivuse kontroll:

```
/*
 * Kontrolli Vertexi equals meetodit erinevate konstruktoritega
 */
Vertex v1 = new Vertex("v1");
Vertex v2 = new Vertex("v1");
System.out.println("Peaks tagastama true, tagastab: " + v1.equals(v2));
//
v1 = new Vertex("v1", null, null);
v2 = new Vertex("v1", new Vertex("v1"), new Arc("a1"));
System.out.println("Peaks tagastama true, tagastab: " + v1.equals(v2));
//
v1 = new Vertex("v1");
v2 = new Vertex("v2");
System.out.println("Peaks tagastama false, tagastab: " + v1.equals(v2));
//
v1 = new Vertex("v1", null, null);
v2 = new Vertex("v101", new Vertex("v1"), new Arc("a1"));
System.out.println("Peaks tagastama false, tagastab: " + v1.equals(v2));

TULEMUS KÄSUREALT:
Peaks tagastama true, tagastab: true
Peaks tagastama true, tagastab: true
Peaks tagastama false, tagastab: false
Peaks tagastama false, tagastab: false
```

Klassi *Arc* isendimeetodi *equals* toimivuse kontroll:

```
/*
 * Kontrolli Arc equals meetodit erinevate konstruktoritega
 */
Arc a1 = new Arc("a1");
Arc a2 = new Arc("a1");
try {
   a1.equals(a2);
} catch (NullPointerException e) {
   System.out.println("Viskab erindi NullPointerExceptioni, kuna target on null
väärtusega kaartel");
}
//
a1 = new Arc("a1", new Vertex("v1"), null);
a2 = new Arc("a1", new Vertex("v1"), new Arc("a3"));
System.out.println("Peaks tagastama true, tagastab: " + a1.equals(a2));
//
a1 = new Arc("a1", new Vertex("v3"), null);
a2 = new Arc("a101", new Vertex("v1"), new Arc("a1"));
System.out.println("Peaks tagastama false, tagastab: " + a1.equals(a2));

TULEMUS KÄSUREALT:
Viskab erindi NullPointerExceptioni, kuna target on null väärtusega kaartel
Peaks tagastama true, tagastab: true
Peaks tagastama false, tagastab: false
```

Klassi *Graph* isendimeetodi *deepCopy* ja *isIsomorphicTo* toimivuse kontroll:

```
/*
 * Kontrolli meetodi deepCopy() toimivust
 */
Graph g = new Graph("g");
g.createRandomSimpleGraph(50,1225);
GraphTask.Graph gClone = g.deepCopy();
GraphTask.Vertex v0 = g.getFirst();
GraphTask.Vertex v1 = gClone.getFirst();
while (v0 != null) {
   if (v0 == v1) {
      throw new IllegalStateException("Kloonitud tipud ei tohiks viidata samale
mäluaadressile.");
   }
   if (!v0.equals(v1)) {
      throw new IllegalStateException("Kloonitud tipud ei tohiks olla erinevate
isendimuutujatega.");
   }
   GraphTask.Arc a0 = v0.getFirst();
   GraphTask.Arc a1 = v1.getFirst();
   while (a0 != null) {
      if (a0 == a1) {
         throw new IllegalStateException("Kloonitud kaared ei tohiks viidata
samale mäluaadressile.");
      }
      if (!a0.equals(a1)) {
         throw new IllegalStateException("Kloonitud kaared ei tohiks olla
erinevate isendimuutujatega.");
      }
      a0 = a0.getNext();
      a1 = a1.getNext();
   }
   v0 = v0.getNext();
   v1 = v1.getNext();
}

/*
 * Kontrolli meetodi isIsomorphicTo() toimivust
 */
if (!g.isIsomorphicTo(gClone)) {
   throw new IllegalStateException("Meetod isIsomorphicTo() peaks tagastama true
väärtuse");
}

TULEMUS KÄSUREALT:
Process finished with exit code 0
```

Klassi *Graph* isendimeetodi *deepCopy* lahendusajad:

```
/*
 * Mõõda meetodi deepCopy aega
 */
Graph g = new GraphTask().new Graph("g");
int n = 500;
while (n <= 2500) {
   int m = n * (n - 1) / 2;
   g.createRandomSimpleGraph(n, m);
   long startTime = System.currentTimeMillis();
   GraphTask.Graph gClone = g.deepCopy();
   System.out.println("Aega kulus " + n + " tipuga " + "ja " + m + " servaga: " +
(System.currentTimeMillis() - startTime) + " ms");
   n += 500;
}

TULEMUS KÄSUREALT:
Aega kulus 500 tipuga ja 124750 servaga: 42 ms
Aega kulus 1000 tipuga ja 499500 servaga: 58 ms
Aega kulus 1500 tipuga ja 1124250 servaga: 89 ms
Aega kulus 2000 tipuga ja 1999000 servaga: 314 ms
Aega kulus 2500 tipuga ja 3123750 servaga: 390 ms
```

Klassi *Graph* isendimeetodi *isIsomorphicTo* lahendusajad:

```
/*
 * Mõõda meetodi isIsomorphicTo aega
 */
Graph g = new GraphTask().new Graph("g");
int n = 500;
while (n <= 2500) {
   int m = n * (n - 1) / 2;
   g.createRandomSimpleGraph(n, m);
   GraphTask.Graph gClone = g.deepCopy();
   long startTime = System.currentTimeMillis();
   gClone.isIsomorphicTo(g);
   System.out.println("Aega kulus " + n + " tipuga " + "ja " + m + " servaga: " +
(System.currentTimeMillis() - startTime) + " ms");
   n += 500;
}

TULEMUS KÄSUREALT:
Aega kulus 500 tipuga ja 124750 servaga: 135 ms
Aega kulus 1000 tipuga ja 499500 servaga: 253 ms
Aega kulus 1500 tipuga ja 1124250 servaga: 681 ms
Aega kulus 2000 tipuga ja 1999000 servaga: 1731 ms
Aega kulus 2500 tipuga ja 3123750 servaga: 4599 ms
```

KASUTATUD ALLIKAD

[1] [WWW] https://www.geeksforgeeks.org/applications-of-graph-data-structure/

[2] [WWW] http://enos.itcollege.ee/~jpoial/algoritmid/graafid.html

[3] [WWW] http://javatechniques.com/blog/faster-deep-copies-of-java-objects/?fbclid=IwAR09jtPaAJKDq2dQSmOm1ihAjvRFV0-ghlvLlwR4V9smVL94f4HBP9u74hY

[4] [WWW] https://dzone.com/articles/java-copy-shallow-vs-deep-in-which-you-will-swim

LISAD

```
import java.util.*;

/** 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) throws
CloneNotSupportedException {
      GraphTask a = new GraphTask();
      a.run();
   }

   /** Actual main method to run examples and everything. */
   public void run() throws CloneNotSupportedException {
      /*
       * Kontrolli meetodi deepCopy() toimivust
       */
      Graph g = new Graph("g");
      g.createRandomSimpleGraph(50,1225);
      GraphTask.Graph gClone = g.deepCopy();
      GraphTask.Vertex v0 = g.getFirst();
      GraphTask.Vertex v1 = gClone.getFirst();
      while (v0 != null) {
         if (v0 == v1) {
            throw new IllegalStateException("Kloonitud tipud ei
tohiks viidata samale mäluaadressile.");
         }
         if (!v0.equals(v1)) {
            throw new IllegalStateException("Kloonitud tipud ei
tohiks olla erinevate isendimuutujatega.");
         }
         GraphTask.Arc a0 = v0.getFirst();
         GraphTask.Arc a1 = v1.getFirst();
         while (a0 != null) {
            if (a0 == a1) {
               throw new IllegalStateException("Kloonitud kaared
ei tohiks viidata samale mäluaadressile.");
            }
            if (!a0.equals(a1)) {
               throw new IllegalStateException("Kloonitud kaared
ei tohiks olla erinevate isendimuutujatega.");
            }
            a0 = a0.getNext();
            a1 = a1.getNext();
         }
```

```
         v0 = v0.getNext();
         v1 = v1.getNext();
      }

      /*
       * Kontrolli meetodi isIsomophicTo() toimivust
       */
      if (!g.isIsomophicTo(gClone)) {
         throw new IllegalStateException("Meetod isIsomophicTo()
peaks tagastama true väärtuse");
      }
   }

   // TODO!!! add javadoc relevant to your problem
   class Vertex  {

      private 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, null, null);
      }

      // TODO!!! Your Vertex methods here!

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

      /** Võrdle tippe
       * @param o võrreldav tipp
       * @return true, kui isendimuutujad 'id' on samased
       */
      @Override
      public boolean equals(Object o) {
         if (this == o) return true;
         if (!(o instanceof Vertex)) return false;

         Vertex vertex = (Vertex) o;
```

```
         return id != null ? id.equals(vertex.id) : vertex.id ==
null;
      }

      /** Tagasta kaarteahela esimene kaar
       * @return esimene kaar
       */
      public Arc getFirst() {
         return first;
      }

      @Override
      public int hashCode() {
         return id != null ? id.hashCode() : 0;
      }

      /** Tagasta järgmine tipp
       * @return järgmine tipp
       */
      public Vertex getNext() {
         return next;
      }
   }


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

      public String id;
      public Vertex target;
      public Arc next;
      public int info = 0;

      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!

      /** Tagasta järgmine kaar
       * @return järgmine kaar
       */
      public Arc getNext() {
         return next;
      }

      /** Võrdle kaari. Ei arvesta lähtetippu
       * @param o võrreldav kaar
       * @return true, kui isendimuutujad 'target' ja 'id' on
samased
       */
      @Override
      public boolean equals(Object o) {
          if (this == o) return true;
          if (!(o instanceof Arc)) return false;

          Arc arc = (Arc) o;

          if (!id.equals(arc.id)) return false;
          return target.equals(arc.target);
      }

       @Override
       public int hashCode() {
           int result = id.hashCode();
           result = 31 * result + target.hashCode();
           return result;
       }
   }


   class Graph implements java.lang.Cloneable {

      private String id;
      private Vertex first; // graafi algus
      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;
         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;
      }

      private 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.
```

```
      /** Sügavkloneeri graaf
       * @return graafi sügavkloon
       * @throws CloneNotSupportedException
       */
      public Graph deepCopy() throws CloneNotSupportedException {
         Graph gClone = new Graph(id);
         if (first == null) return gClone;
         Map<Vertex, Vertex> vertices =
oldAndCloneVertices(first); // hashmap, et kaartele panna targetid
hiljem
         gClone.first = vertices.get(first);
         Vertex v = first;
         while (v != null) {
            if (v.first == null) continue;
            vertices.get(v).first =
createAndConnectCloneArcs(v.first, vertices);
            v = v.getNext();
         }
         return gClone;
      }

      /** Loo vanade tippude ahela sügavkloon
       * @param first sügavkloneeritava graafi esimene tipp
       * @return  paisktabel, kus: võti - kloneeritav tipp,
väärtus - sügavkloonitud tipu koopia ilma kaarteta
       */
      private Map<Vertex, Vertex> oldAndCloneVertices(Vertex
first) {
         Map<Vertex, Vertex> vertices = new HashMap<>();
         Vertex oldV = first;
         Vertex newV = new Vertex(oldV.id);
         vertices.put(oldV, newV);
         while (oldV.next != null) {
            oldV = oldV.next;
            newV.next = new Vertex(oldV.id);
            vertices.put(oldV, newV.next);
            newV = newV.next;
         }
         return vertices;
      }

      /** Loo vana kaarteahela sügavkloon
       * @param first sügavkloneeritava kaarte ahela esimene
element
       * @param vertices   paisktabel, kus võti - kloneeritav
tipp, väärtus - sügavkloonitud koopia
       * @return  kaarte ahela esimene element
       */
```

```
      private Arc createAndConnectCloneArcs(Arc first, Map<Vertex,
Vertex> vertices) {
         Arc oldA = first;
         Arc newA = new Arc(oldA.id, vertices.get(oldA.target),
null);
         Arc newFirst = newA;
         while (oldA.next != null) {
            oldA = oldA.next;
            newA.next = new Arc(oldA.id,
vertices.get(oldA.target), null);
            newA = newA.next;
         }
         return newFirst;
      }

      /** Kontrolli kahe graafi isomorfismi
       * @param g võrreldav graaf
       * @return  true, kui graafidel on tipud, kaared ja
nendevahelised seosed samased
       */
      public boolean isIsomophicTo(Graph g) {
          if (!hasSameVertices(g)) return false;
          if (!hasSameArcs(g)) return false;
          return true;
      }

      /** Võrdle graafide tippe
       * @param g võrreldav graaf
       * @return true, kui tippe on sama palju ja samased (equals)
       */
      private boolean hasSameVertices(Graph g) {
          HashSet<Vertex> thisGraphVertices = new HashSet<>();
          Vertex v = first;
          while (v != null) {
              thisGraphVertices.add(v);
              v = v.next;
          }
          int n = thisGraphVertices.size();
          v = g.first;
          // kontrollib, kas tipud on samad
          while (v != null) {
              if (thisGraphVertices.contains(v)) n--; else return
false;
              v = v.next;
          }
          if (n > 0) return false; // kui jääb üle, siis ei ole
võrdsed
          return true;
      }
```

```
      /** Võrdle graafide kaari
       * @param g võrreldav graaf
       * @return true, kui kaari on sama palju ja samased (equals)
       */
      private boolean hasSameArcs(Graph g) {
          HashSet<Arc> thisGraphArcs = new HashSet<>();
          Vertex v = first;
          int n = 0;
          while (v != null) {
              Arc a = v.first;
              while (a != null) {
                  thisGraphArcs.add(a);
                  a = a.next;
              }
              v = v.next;
          }
          v = g.first;
          n = thisGraphArcs.size();
          // kontrollib, kas servad on samad
          while (v != null) {
              Arc a = v.first;
              while (a != null) {
                  if (thisGraphArcs.contains(a)) n--; else return
false;
                  a = a.next;
              }
              v = v.next;
          }
          if (n > 0) return false; // kui jääb üle, siis ei ole
võrdsed
          return true;
      }

      /** Tagasta graafi tippude ahela esimene tipp
       * @return ahela esimene tipp
       */
      public Vertex getFirst() {
         return first;
      }

      @Override
      protected Object clone() throws CloneNotSupportedException {
         throw new UnsupportedOperationException();
      }

      @Override
      public boolean equals(Object o) {
         throw new UnsupportedOperationException();
```

```
        }

        @Override
        public int hashCode() {
           throw new UnsupportedOperationException();
        }


    }
}
```