Tallinna Tehnikaülikool

Individuaaltöö aines “Algoritmid ja andmestruktuurid”

autor: Merilin Kalda
eriala: IT-süsteemide arendus
juhendaja: Jaanus Pöial

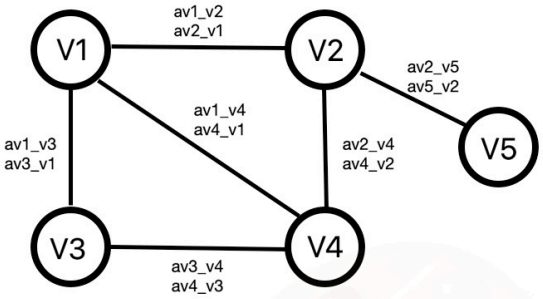
Tallinn 2020

Ülesande püstitus.

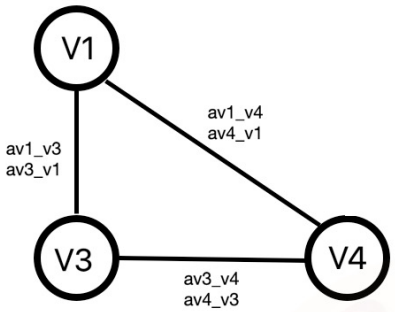
Ülesandeks oli koostada meetod etteantud tipu ning tema otseste järglaste eemaldamiseks etteantud graafist, kasutades graafi esitust külgnevusstruktuurina. Etteantud graafiks on juhuslik sidus lihtgraaf, milles ei ole suundasid, tsükleid ja kordseid servi. Servad on kokkuleppe kohaselt esitatud kahe kaarena algtipust lõpptippu ja vastupidi.
Etteantud aluspõhi, millele lisasin enda kirjutatud meetodid, pärineb https://bitbucket.org/itc_algorithms/kt6/src/master/src/GraphTask.java

Näiteks on meil algselt 5 tipuga graaf G, millest soovitakse eemaldada tipp V2. Tipu V2 eemaldamise tulemusena eemaldatakse ka tipu V2 järglane tipp V5 ning nendevaheline serv. Samuti eemaldatakse ka ülejäänud servad mis on otseses ühenduses eemaldatava tipuga V2. Peale eemaldamist peaks järele jääma kolme tipuga graaf.

```
G
v1 --> av1_v4 (v1->v4) av1_v3 (v1->v3) av1_v2 (v1->v2)
v2 --> av2_v5 (v2->v5) av2_v1 (v2->v1) av2_v4 (v2->v4)
v3 --> av3_v4 (v3->v4) av3_v1 (v3->v1)
v4 --> av4_v2 (v4->v2) av4_v1 (v4->v1) av4_v3 (v4->v3)
v5 --> av5_v2 (v5->v2)
```

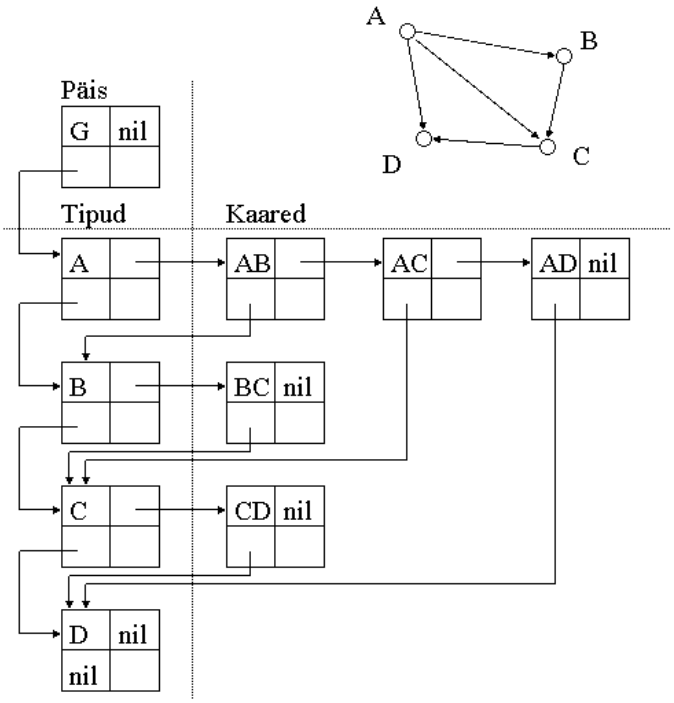



```
G
v1 --> av1_v4 (v1->v4) av1_v3 (v1->v3)
v3 --> av3_v4 (v3->v4) av3_v1 (v3->v1)
v4 --> av4_v1 (v4->v1) av4_v3 (v4->v3)
```



Graafi kujutamine külgnevusstruktuuri abil

E={ (A,B), (A,C), (A,D), (B,C), (C,D) }



allikas https://enos.itcollege.ee/%7ejpoial/algoritmid/graafid.html

Lahenduse kirjeldus

Lahenduse aluseks on võetud graafi laiuti läbimise algoritm(inglise keeles breadh first search, lühend BFS).

Graafi laiuti läbimise põhimõte (allikas https://enos.itcollege.ee/%7ejpoial/algoritmid/graafid.html)
Tipud võivad olla "valged" (töötlemata), "hallid" (järjekorras) või "mustad" (töödeldud).
Algselt on kõik tipud valged ja järjekord tühi.
Läbimine algab mingist tipust t: t lisatakse järjekorda ning värvitakse halliks.
Järgnevat tsüklit täidetakse niikaua, kuni järjekord pole tühi:
järjekorrast eemaldatakse tipp v,
v töödeldakse,
v värvitakse mustaks,
vaadatakse läbi kõik v vahetud järglased w:
kui w on valge, siis w lisatakse järjekorda ning värvitakse halliks.
Tippude töötlemise järjekord on FIFO ehk first in first out, mis tähendab et esimesena järjekorda lisatud tipud töödeldakse esimesena.

Tipu eemaldates tuli eemaldada lisaks ka kõik selle tipuga ühenduses olevad servad.
Selle jaoks tegin uue listi, kuhu kogusin servad, mida hiljem eemaldada.
Kui graafi läbimisel jõudsin servani mille otspunktiks oli etteantud eemaldtav tipp, siis lisasin servale viitena ka teise otspunkti ja selle serva panin listi.
Tipu eemaldamiseks muutsin eelneva tipu viite st. et külgnevusstruktuuris eelnev tipp ei viita enam eemaldatud tipule vaid eemaldatud tipust järgnevale või järgneva puudumisel on väärtuseks null.
Kui eemaldatud tipul oli järglasi, siis esimesele lisasin sildi, et tipp on eemaldatud tipu järglane.
Kui graaf oli läbi käidud, siis eemaldasin graafist kõik need servad, mille olin eelnevalt listi lisanud.
Serva eemaldamiseks kirjutasin eraldi meetodi, milleks oli vaja teada mõlemat tippu, mida serv ühendab.

Seejärel käisin läbi graafi külgnevusstruktuuri, ning muutsin ära viidad kõikidel sellistel tippudel, mis olid eemaldatud tipu järglased.

Juhul kui üritatakse graafist eemaldada tippu, mida antud graafis ei ole, annab programm veateate.

Programmi kasutusjuhend.

Programm koosneb klassidest Graph, Vertex, Arc ja GraphTask.

class Vertex

Vertex (String s, Vertex v, Arc e)
    Vertex objekti konstruktor
    parameetrid :
        s - tipu id
        v - viit järgmisele tipule
        e - viit esimesele kaarele
Vertex (String s)
    Vertex objekti konstruktor. Viit järgmisele tipule ja esimesel kaarele puuduvad.
    parameetrid :
        s - tipu id
public boolean equals(Object o)
    meetod mis tagastab tõeväärtuse, kas kaks tippu on võrdsed
    parameetrid:
        o - teine tipp millega võrreldakse
    tagastab :
        tõeväärtuse, true või false
public String toString()
    Vertex objekti sõne kujuline esitus.
    tagastab :
        tipu id
public void setColor(int i)
    meetod tipule värvi andmiseks. algselt kõik tipud valged. 0- valge, 1- hall, 2- must
    parameetrid:
        i - värv numbri kujul, kas 0, 1, või 2

public int getColor()

meetod tipu värvi saamiseks

tagastab:

tipu värvi nubri kujul, 0 - valge, 1 - hall, 2- must

public Iterator<Arc> outArcs()

Koostab listi kõikidest antud tipust väljuvatest kaartest ja tsükli üle kõigi nende kaarte.

tagastab:

Iteraatori üle kõigi antud tipust väljuvate kaarte

class Arc

Arc (String s, Vertex v, Arc a)

Arc objekti konstruktor

parameetrid :

s - kaare id

v - viit suubuvasse tippu

e - viit järgmisele kaarele

Arc (String s)

Arc objekti konstruktor, viidad tipule ja järgmisele kaarele puuduvad.

parameetrid :

s - kaare id

public String toString()

Arc objekti sõne kujuline esitus.

tagastab :

kaare id

public Vertex getToTarget()

Meetod, kaare lõpp punkti leidmiseks

tagastab:

tipu, kuhu antud kaar suubub.

class Graph

Graph (String s, Vertex v)

Graph objekti konstruktor

parameetrid :

s - graafi id

v - viit esimesse tippu

Graph (String s)

Graph objekti konstruktor. Viit esimesse tippu puudub.

parameetrid :

s - graafi id

public String toString()

Graph objekti sõne kujuline esitus. Igal real on tipp ning kõik sellest tipust väljuvad kaared.

tagastab :

graafi sõne kujul

public Vertex createVertex (String vid)

meetod uue tipu loomiseks. Uus loodud tipp pannakse antud graafi esimesele kohale ning senine esimene tipp lisatakse uuele tipule järgmise tipu viidaks

parameetrid:

vid - tipu id

tagastab:

uue tipu

public Arc createArc (String aid, Vertex from, Vertex to)

meetod uue kaare loomiseks. Uus loodud kaar pannake esimeseks, ning senine esimene lisatakse viitena uuele kaarele järgmiseks kaareks

parameetrid:

aid - kaare id

from - kaare algustipp

to - kaare lõpptipp

tagastab:

uue kaare

public void createRandomTree (int n)

meetod uue n- tipulise puu loomiseks. Iga uus tipp ühendatakse juhusliku olemasoleva tipu külge.

parameetrid:

n - tippude arv

public int[][] createAdjMatrix()

meetod graafi külgnemismaatriksi loomiseks.

tagastab:

graafi külgnevusmaatriksi

public void createRandomSimpleGraph (int n, int m)

meetod sidusa, orienteerimata, ilma tsükliteta ja ilma kordsete servadeta lihtgraafi loomiseks

parameetrid:

n - tippude arv

m - servade arv

public LinkedList<Vertex> getVertexList()

meetod kõigi graafis esinevate tippude listi genereerimiseks

tagastab:

listi tippudest

public void removeArc(Vertex source, Vertex target)

eemaldab etteantud kahe tipu vahelise kaare.

parameetrid:

source - kaare algtipp

target - kaare lõpptipp

public void setPreviousVertex()

lisab igale graafi tipule viida eelneva tipu kohta

public void removeVertex(Vertex vertexToRemove)

eemaldab etteantud lihtgraafist graafist etteantud tipu ja kõik selle tipuga seotud kaared ning ka tipu vahetud järglased

parameetrid:

vertexToRemove - tipp, mida soovitakse graafist eemaldada

Testimiskava

Enamus teste tegin väikeste graafidega, et oleks lihtsam hinnata, kas programm töötab nii nagu peaks. Testisin ka suuremõõtmelise graafiga milles on 2000 tippu ja sealjuures mõõtsin programmi töötamiseks kulunud aega.
Lisaks testisin olukordi, kus eemaldada soovitav tipp ei sisaldugi antud graafis.
Testin nii graafe milles eemaldataval tipul ei ole otseseid järglasi ja graafe mille eemaldaval tipul on otsesed järglased.

2 testi juhuslikult genereeritud väikseste graafiga

```
public void run() {
   Graph g = new Graph ( s: "G");
   g.createRandomSimpleGraph ( n: 6,  m: 9);
   System.out.println (g);

   g.removeVertex(new Vertex( s: "v3"));

   System.out.println (g);
```

GraphTask › run()

GraphTask

```
G
v1 --> av1_v4 (v1->v4) av1_v2 (v1->v2) av1_v3 (v1->v3)
v2 --> av2_v6 (v2->v6) av2_v1 (v2->v1) av2_v4 (v2->v4)
v3 --> av3_v6 (v3->v6) av3_v1 (v3->v1) av3_v4 (v3->v4)
v4 --> av4_v1 (v4->v1) av4_v2 (v4->v2) av4_v3 (v4->v3) av4_v5 (v4->v5)
v5 --> av5_v4 (v5->v4) av5_v6 (v5->v6)
v6 --> av6_v2 (v6->v2) av6_v3 (v6->v3) av6_v5 (v6->v5)


G
v1 --> av1_v4 (v1->v4) av1_v2 (v1->v2)
v2 --> av2_v6 (v2->v6) av2_v1 (v2->v1) av2_v4 (v2->v4)
v4 --> av4_v1 (v4->v1) av4_v2 (v4->v2) av4_v5 (v4->v5)
v5 --> av5_v4 (v5->v4) av5_v6 (v5->v6)
v6 --> av6_v2 (v6->v2) av6_v5 (v6->v5)
```

```java
/** Actual main method to run examples and everything. */
public void run() {
   Graph g = new Graph ( s: "G");
   g.createRandomSimpleGraph ( n: 6,  m: 9);
   System.out.println (g);

   g.removeVertex(new Vertex( s: "v6"));

   System.out.println (g);
```

GraphTask › run()

GraphTask

```
G
v1 --> av1_v6 (v1->v6) av1_v3 (v1->v3)
v2 --> av2_v5 (v2->v5) av2_v6 (v2->v6) av2_v3 (v2->v3)
v3 --> av3_v6 (v3->v6) av3_v1 (v3->v1) av3_v2 (v3->v2) av3_v4 (v3->v4)
v4 --> av4_v3 (v4->v3) av4_v5 (v4->v5)
v5 --> av5_v2 (v5->v2) av5_v4 (v5->v4) av5_v6 (v5->v6)
v6 --> av6_v3 (v6->v3) av6_v2 (v6->v2) av6_v1 (v6->v1) av6_v5 (v6->v5)


G
v1 --> av1_v3 (v1->v3)
v2 --> av2_v5 (v2->v5) av2_v3 (v2->v3)
v3 --> av3_v1 (v3->v1) av3_v2 (v3->v2) av3_v4 (v3->v4)
v4 --> av4_v3 (v4->v3) av4_v5 (v4->v5)
v5 --> av5_v2 (v5->v2) av5_v4 (v5->v4)
```

Olukord kus eemaldatav tipp ei sisaldu graafis:

```java
/** Actual main method to run examples and everything. */
public void run() {
   Graph g = new Graph ( s: "G");
   g.createRandomSimpleGraph ( n: 6,  m: 9);
   System.out.println (g);

   g.removeVertex(new Vertex( s: "v9"));

   System.out.println (g);
```

GraphTask › run()

GraphTask

```
/Library/Java/JavaVirtualMachines/jdk1.8.0_261.jdk/Contents/Home/bin/java ...

G
v1 --> av1_v2 (v1->v2) av1_v6 (v1->v6) av1_v5 (v1->v5)
v2 --> av2_v3 (v2->v3) av2_v1 (v2->v1) av2_v5 (v2->v5)
v3 --> av3_v2 (v3->v2) av3_v4 (v3->v4)
v4 --> av4_v5 (v4->v5) av4_v3 (v4->v3) av4_v6 (v4->v6)
v5 --> av5_v4 (v5->v4) av5_v1 (v5->v1) av5_v2 (v5->v2) av5_v6 (v5->v6)
v6 --> av6_v1 (v6->v1) av6_v4 (v6->v4) av6_v5 (v6->v5)

Exception in thread "main" java.lang.RuntimeException: Graph doesn't contain vertex v9
    at GraphTask$Graph.removeVertex(GraphTask.java:398)
    at GraphTask.run(GraphTask.java:20)
    at GraphTask.main(GraphTask.java:11)

Process finished with exit code 1
```

Käsitsi genereeritud graafid, et eemaldataval tipul oleks otsesed järglased.

Esimesel juhul tipu v2 eemaldades eemaldatakse ka tipu otsene järglane v5

Teisel juhul tipu v2 eemaldades eemaldatakse ka tipu v2 otsesed järglased v5 ja v6

```
g2.createArc( aid: "av1_v2", g2.first, g2.first.next);
g2.createArc( aid: "av1_v3", g2.first, g2.first.next.next);
g2.createArc( aid: "av1_v4", g2.first, g2.first.next.next.next);
g2.createArc( aid: "av4_v3", g2.first.next.next.next, g2.first.next.next);
g2.createArc( aid: "av2_v4", g2.first.next, g2.first.next.next.next);
g2.createArc( aid: "av2_v1", g2.first.next, g2.first);
g2.createArc( aid: "av3_v1", g2.first.next.next, g2.first);
g2.createArc( aid: "av4_v1", g2.first.next.next.next, g2.first);
g2.createArc( aid: "av3_v4", g2.first.next.next, g2.first.next.next.next);
g2.createArc( aid: "av4_v2", g2.first.next.next.next, g2.first.next);
g2.createArc( aid: "av2_v5", g2.first.next, g2.first.next.next.next.next);
g2.createArc( aid: "av5_v2", g2.first.next.next.next.next, g2.first.next);

System.out.println(g2);

g2.removeVertex(new Vertex( s: "v2"));

System.out.println(g2);
```

```
GraphTask › run()
GraphTask
/Library/Java/JavaVirtualMachines/jdk1.8.0_261.jdk/Contents/Home/bin/java ...

G
v1 --> av1_v4 (v1->v4) av1_v3 (v1->v3) av1_v2 (v1->v2)
v2 --> av2_v5 (v2->v5) av2_v1 (v2->v1) av2_v4 (v2->v4)
v3 --> av3_v4 (v3->v4) av3_v1 (v3->v1)
v4 --> av4_v2 (v4->v2) av4_v1 (v4->v1) av4_v3 (v4->v3)
v5 --> av5_v2 (v5->v2)


G
v1 --> av1_v4 (v1->v4) av1_v3 (v1->v3)
v3 --> av3_v4 (v3->v4) av3_v1 (v3->v1)
v4 --> av4_v1 (v4->v1) av4_v3 (v4->v3)


Process finished with exit code 0
```

```
g2.createArc( aid: "av1_v3", g2.first, g2.first.next.next);
g2.createArc( aid: "av1_v4", g2.first, g2.first.next.next.next);
g2.createArc( aid: "av4_v3", g2.first.next.next.next, g2.first.next.next);
g2.createArc( aid: "av2_v4", g2.first.next, g2.first.next.next.next);
g2.createArc( aid: "av2_v1", g2.first.next, g2.first);
g2.createArc( aid: "av3_v1", g2.first.next.next, g2.first);
g2.createArc( aid: "av4_v1", g2.first.next.next.next, g2.first);
g2.createArc( aid: "av3_v4", g2.first.next.next, g2.first.next.next.next);
g2.createArc( aid: "av4_v2", g2.first.next.next.next, g2.first.next);
g2.createArc( aid: "av2_v5", g2.first.next, g2.first.next.next.next.next);
g2.createArc( aid: "av5_v2", g2.first.next.next.next.next, g2.first.next);
g2.createArc( aid: "av2_v6", g2.first.next, g2.first.next.next.next.next.next);
g2.createArc( aid: "av6_v2", g2.first.next.next.next.next.next, g2.first.next);

System.out.println(g2);

g2.removeVertex(new Vertex( s: "v2"));

System.out.println(g2);
```

```
GraphTask › run()
GraphTask

G
v1 --> av1_v4 (v1->v4) av1_v3 (v1->v3) av1_v2 (v1->v2)
v2 --> av2_v6 (v2->v6) av2_v5 (v2->v5) av2_v1 (v2->v1) av2_v4 (v2->v4)
v3 --> av3_v4 (v3->v4) av3_v1 (v3->v1)
v4 --> av4_v2 (v4->v2) av4_v1 (v4->v1) av4_v3 (v4->v3)
v5 --> av5_v2 (v5->v2)
v6 --> av6_v2 (v6->v2)


G
v1 --> av1_v4 (v1->v4) av1_v3 (v1->v3)
v3 --> av3_v4 (v3->v4) av3_v1 (v3->v1)
v4 --> av4_v1 (v4->v1) av4_v3 (v4->v3)


Process finished with exit code 0
```

Test 2000 tipulise graafiga.

```
   public void run() {
      Graph g = new Graph ( s: "G");
      g.createRandomSimpleGraph ( n: 2000,  m: 3000);
      System.out.println(g.toString().contains("v1024"));

      long startTime = System.currentTimeMillis();
      g.removeVertex(new Vertex( s: "v1024"));
      long timeSpent = System.currentTimeMillis()-startTime;

      System.out.println("Time spent: " + timeSpent + " milliseconds");

//      System.out.println (g);

      System.out.println(g.toString().contains("v1024"));
```

GraphTask › run()

GraphTask

```
/Library/Java/JavaVirtualMachines/jdk1.8.0_261.jdk/Contents/Home/bin/java ...
true
Time spent: 21 milliseconds
false

Process finished with exit code 0
```

Programmi täielik tekst

```java
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) {
      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: 3000);
      System.out.println(g.toString().contains("v1024"));

      long startTime = System.currentTimeMillis();
      g.removeVertex(new Vertex( s: "v1024"));
      long timeSpent = System.currentTimeMillis()-startTime;

      System.out.println("Time spent: " + timeSpent + " milliseconds");

//      System.out.println (g);

      System.out.println(g.toString().contains("v1024"));

      //...

   }

   class Vertex {

      private String id;
      private Vertex next;
      private Vertex previous = null;
      private Arc first;
      private int info = 0;
      private int color = 0; //white
      private boolean isChildToRemovedVertex = false;

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

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

      @Override
      public boolean equals(Object o) {
         if (o instanceof Vertex) {
            Vertex v = (Vertex) o;
            if (v.id.equals(this.id)) {
               return true;
            }
         }
         return false;
      }

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

```
  public void setColor(int i) {
    color = i;
  }

  public int getColor() {
    return color;
  }

  public Iterator<Arc> outArcs() {
    List<Arc> arcList = new LinkedList<Arc>();
    Arc arc = this.first;
    while (arc != null) {
      arcList.add(arc);
      arc = arc.next;
    }
    return arcList.iterator();
  }
}


/** 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 Vertex source;
  private Arc next;
  private 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;
  }


  public Vertex getToTarget() {
    return target;
  }

}


class Graph {

  private String id;
  private Vertex first;
  private int info = 0;

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

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

```java
@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;
}
```

```java
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 ( vid: "v" + String.valueOf(n-i));
      if (i > 0) {
         int 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]);
      } else {}
   }
}
```

```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 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 ( 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
  }
}

/**
 * Get list of all the vertices in the graph.
 * @return LinkedList of vertices.
 *
 */
public LinkedList<Vertex> getVertexList() {
  LinkedList<Vertex> vertexList= new LinkedList<Vertex>();
  Vertex v = this.first;
  while (v != null) {
    vertexList.add(v);
    v = v.next;
  }
  return vertexList;
}
```

```
 /**
  * Remove arc between two vertices.
  * @param source source vertex of arc.
  * @param target target vertex of arc.
  */
 public void removeArc(Vertex source, Vertex target) {
    Iterator arcIterator = source.outArcs();
    Arc previousArc = null;
    while (arcIterator.hasNext()) {
       Arc a = (Arc) arcIterator.next();
       if (a.target.equals(target)) {
          if (previousArc != null) {
             previousArc.next = a.next;
          } else {
             source.first = a.next;
          }
       }
       previousArc = a;
    }
 }
```

```java
/**
 * Set previous vertex to each vertex in graph.
 */
public void setPreviousVertex() {
   Vertex s = this.first;
   Vertex next = s.next;
   while (next!=null) {
      next.previous = s;
      s = s.next;
      next = next.next;
   }
}


/**
* Method to remove given vertex and it's children from given random simple graph
 * Algorithm inspired by BFS from https://enos.itcollege.ee/%7ejpoial/algoritmid/graafid.html
* @param vertexToRemove vertex to remove from this graph.
*/
public void removeVertex(Vertex vertexToRemove) {
   if (vertexToRemove.id.equals(this.first.id)) {
      this.first = null;
   }
   if (this.first == null) return;
   if ((!getVertexList().contains(vertexToRemove))) {
      throw new RuntimeException ("Graph doesn't contain vertex " + vertexToRemove.id);
   }
   List vertexQueue = Collections.synchronizedList (new LinkedList());
   Vertex s = this.first;
   setPreviousVertex();
   vertexQueue.add (s);
   s.setColor(1); // "gray"
```

```java
   List<Arc> arcsToRemove = new ArrayList<Arc>();

   while (vertexQueue.size() > 0) {
      Vertex v = (Vertex)vertexQueue.remove( index: 0); // breadth == FIFO
      v.setColor(2); // "black"
      Iterator arcIterator = v.outArcs();
      while (arcIterator.hasNext()) {
         Arc a = (Arc)arcIterator.next();
         Vertex target = a.getToTarget();
         if (v.equals(vertexToRemove) && a.target.first.next==null) {
            a.target.isChildToRemovedVertex = true;
         }
         if (target.equals(vertexToRemove)) {
            a.source = v;
            arcsToRemove.add(a);
            target.previous.next = target.next;
         }
         if (target.getColor() == 0) {
            vertexQueue.add(target);
            target.setColor(1);
         }
      }
   }
   if (!arcsToRemove.isEmpty()) {
      for (Arc a: arcsToRemove) {
         removeArc(a.source, a.target);
      }
   }
```

```
        Vertex x = this.first;
        while (x!=null) {
            if (x.isChildToRemovedVertex) {
                if (x.previous!=null) {
                    x.previous.next = x.next;
                }
                if (x.next!=null) {
                    x.next.previous = x.previous;
                }
            }
            x = x.next;
        }
    }
}
```

Kasutatud allikad:

https://enos.itcollege.ee/%7ejpoial/algoritmid/graafid.html
https://bitbucket.org/itc_algorithms/kt6/src/master/src/GraphTask.java