TALTECH

Infotehnoloogia teaduskond

IT süsteemide arendus

Nikita Putyatin

194150IADB

Individuaal töö aines „Algoritmid ja andmestruktuurid“

Õppeaine: Algoritmid ja andmestruktuurid

Õppejõud: Jaanus Pöial

Tallinn 2020

Ülesanne

Ülesandeks oli koostada meetod tipust s tipuni t pikima suunatud tee leidmiseks atsüklilise orienteeritud graafis. See on ülesanne C-13.19, Data Structures and Algorithms in Java Michael T. Goodrich.

Lahendus

Etteantud ülesanne lahendamiseks oli vajalik topologiline sorterimine atsüklilise orienteeritud graafi.

Pärast seda ma natukene muutsin algselt antud juhuslikult genereeritud graaf. Sest mul oli vaja et graaf oli orienteeritud.

Kui programm ei leidnud tsüklit, siis ma saan lehe, kus mu tipud on sorteeritud, mis võimaldab meil tippe läbida lineariseeritud järjestuses vasakult paremale.

Nüüd mul on vaja kõigi tippude seast leidke täpselt minu jaoks vajalik rada etteantud punktide vahel. Kui minu lehel kõik tipud on juba sorteeritud. Ma pean lehe läbima vastupidises suunas ja kui tipul on target eelmise õige tipuga, siis võtame selle tulemuseks.

Selle ülesande jaoks kasutasin seda topoloogiline sorteerimine (Pöial, 2020) . Mida edasi teha, ma ei leidnud Internetist, nii et mõtlesin ise välja lahendused. Mul aitas probleemi lahendada see artikkel (Khan, 2011)

Selle allika algoritm aitas mul probleemi lahendada:

LONGEST-PATH(G)
▷ Input: Unweighted DAG $G = (V, E)$
▷ Output: Largest path cost in G
Topologically sort G
for each vertex $v \in V$ in linearized order
do $\text{dist}(v) = \max_{(u,v)\in E} \{\text{dist}(u) + 1\}$
return $\max_{v\in V} \{\text{dist}(v)\}$

Programmi kasutamisjuhend

Kasutame Graph meetod createRandomSimpleGraph(n, m), kus n on graafitippude arv ja m – kaarete arv. Graaf on tehtud, peaks seda ainult printima veel.

Nüüd peate sisestama 2 tippu, mille vahel soovite leida kõige pikema tee. Tipud tuleks sisestada järgmiselt : vn ja vm, kus n ja m teie 2 tippu. Tipud on vaja sisestada algul suur tipp, siis väiksem tipp. Näiteks - g.topolSort("v4", "v1");. Pärast seda programm printib kõige pikema tee.

Testimiskava

Esimene olukord: testin kõige lihtsalt grafi võimalus. Oodates 2-3 punktilist tee.

Tulemus:

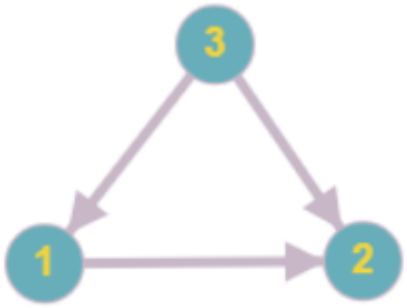
```
14      /** Actual main method to run examples and everything. */
15      public void run() {
16          Graph g = new Graph ( s: "G0");
17          g.createRandomSimpleGraph ( n: 3,  m: 3);
18          System.out.println (g);
19          // Example input: g.topolSort("v4", "v1");
20          System.out.println(g.topolSort( source: "v3",  last: "v1"));
```

GraphTask › run()

GraphTask

```
"D:\IntelliJ IDEA 2019.3.2\jbr\bin\java.exe" "-javaagent:D:\IntelliJ

G0
v1 --> av1_v2 (v1->v2)
v2 -->
v3 --> av3_v1 (v3->v1) av3_v2 (v3->v2)

Longest directed path: v3 --> v1

Process finished with exit code 0
```

Kontrollimiseks kasutan veebiservist GraphOnline. (Graph Online, 2020)



Teine olukord: Võtan graf koos 5 tipuga ja 7 kaarega.

Püüan ehitada pikkema tee tipust v5 tippu v2.

Tulemus:

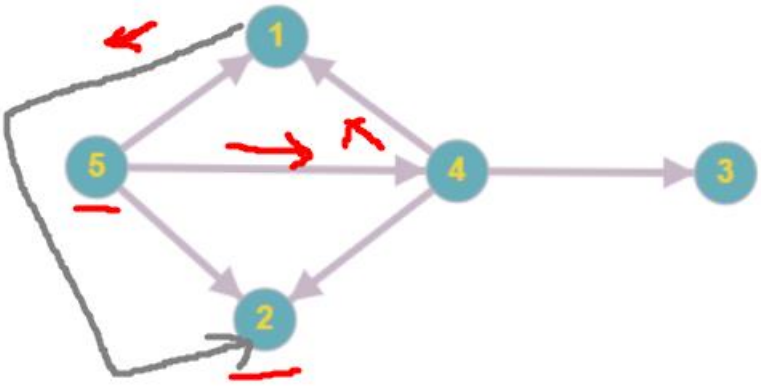
```
22      Graph g1 = new Graph ( s: "G1");
23      g1.createRandomSimpleGraph ( n: 5,  m: 7);
24      System.out.println (g1);
25      System.out.println(g1.topolSort( source: "v5",  last: "v2"));
```

GraphTask › run()

GraphTask

```
"D:\IntelliJ IDEA 2019.3.2\jbr\bin\java.exe" "-javaagent:D:\IntelliJ

G1
v1 --> av1_v2 (v1->v2)
v2 -->
v3 -->
v4 --> av4_v1 (v4->v1) av4_v2 (v4->v2) av4_v3 (v4->v3)
v5 --> av5_v2 (v5->v2) av5_v1 (v5->v1) av5_v4 (v5->v4)

Longest directed path: v5 --> v4 --> v1 --> v2
```

Kontroll:



Kolmas olukord: Nüüd võtan suur graf koos 10 tipuga ja 12 kaarega.

Püüan ehitada pikkema tee tipust v10 tippu v4. Oodates lühike tee 2-6 tippu. Tahat öelda, et kui rohkem tipud ja kaared, siis rohkem võimalus leida tsükkel.

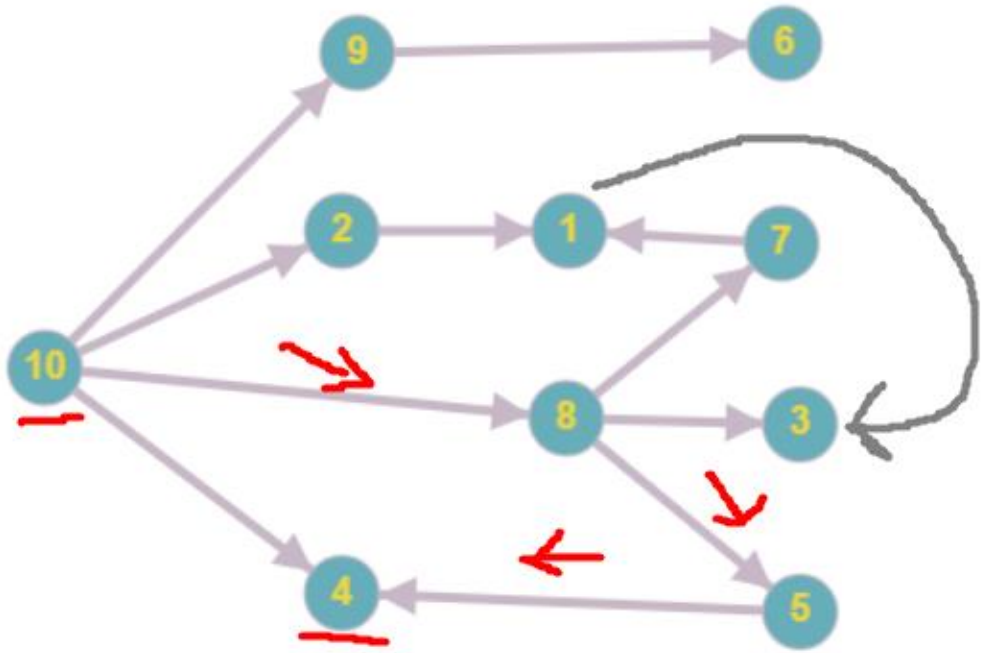
Tulemus:

```
27      Graph g2 = new Graph ( s: "G2");
28      g2.createRandomSimpleGraph ( n: 10,  m: 12);
29      System.out.println (g2);
30      System.out.println(g2.topolSort( source: "v10",  last: "v4"));
31
```

GraphTask › run()

GraphTask

```
G2
v1 --> av1_v3 (v1->v3)
v2 --> av2_v1 (v2->v1)
v3 -->
v4 -->
v5 --> av5_v4 (v5->v4)
v6 -->
v7 --> av7_v1 (v7->v1)
v8 --> av8_v3 (v8->v3) av8_v5 (v8->v5) av8_v7 (v8->v7)
v9 --> av9_v6 (v9->v6)
v10 --> av10_v4 (v10->v4) av10_v2 (v10->v2) av10_v8 (v10->v8) av10_v9 (v10->v9)

Longest directed path: v10 --> v8 --> v5 --> v4

Process finished with exit code 0
```



Neljas olukord: Võtan 5 tipud ja 10 kaared. Oodates tsükkel, sest grafis on liiga palju kaared. Kui tsükkel tuleb, siis ei saa ehita pikkem tee.

Tulemus:

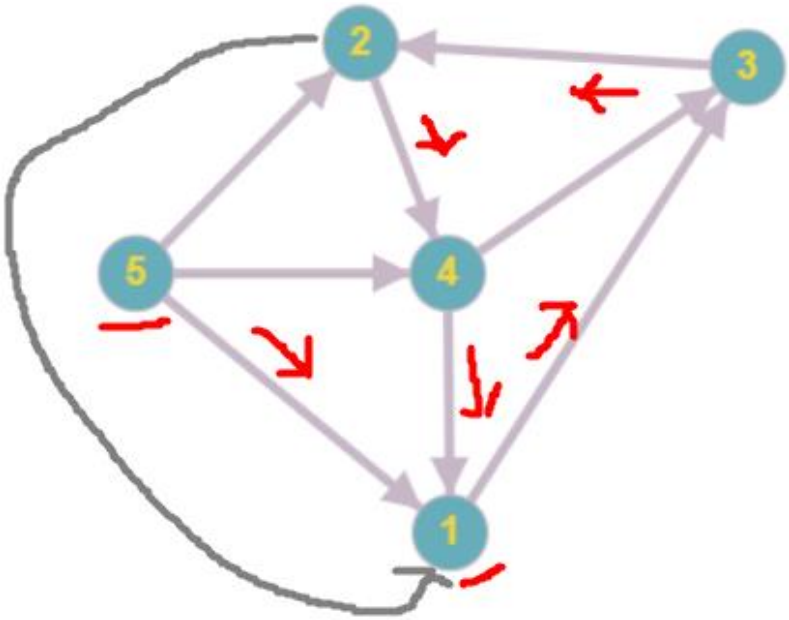
```
32      Graph g3 = new Graph ( s: "G3");
33      g3.createRandomSimpleGraph ( n: 5,  m: 10);
34      System.out.println (g3);
35      System.out.println(g3.topolSort( source: "v5",  last: "v1"));
```

GraphTask › run()

GraphTask

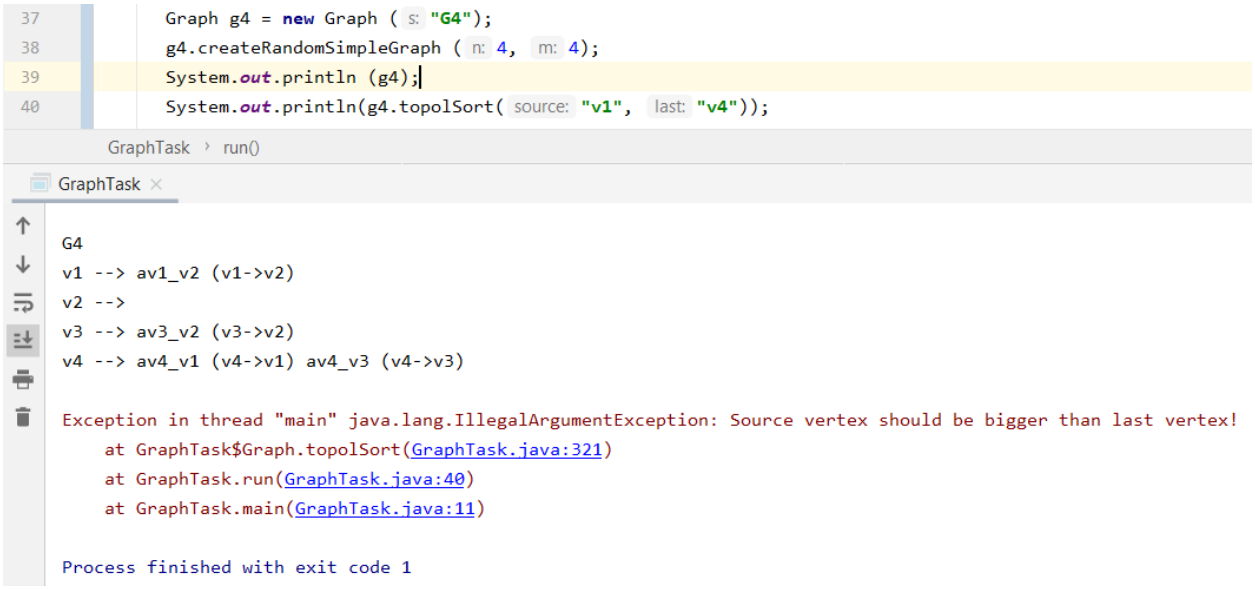
```
G3
v1 --> av1_v3 (v1->v3)
v2 --> av2_v1 (v2->v1) av2_v4 (v2->v4)
v3 --> av3_v5 (v3->v5) av3_v2 (v3->v2)
v4 --> av4_v1 (v4->v1) av4_v3 (v4->v3)
v5 --> av5_v1 (v5->v1) av5_v2 (v5->v2) av5_v4 (v5->v4)

Cycle was found!
```

Kontroll:



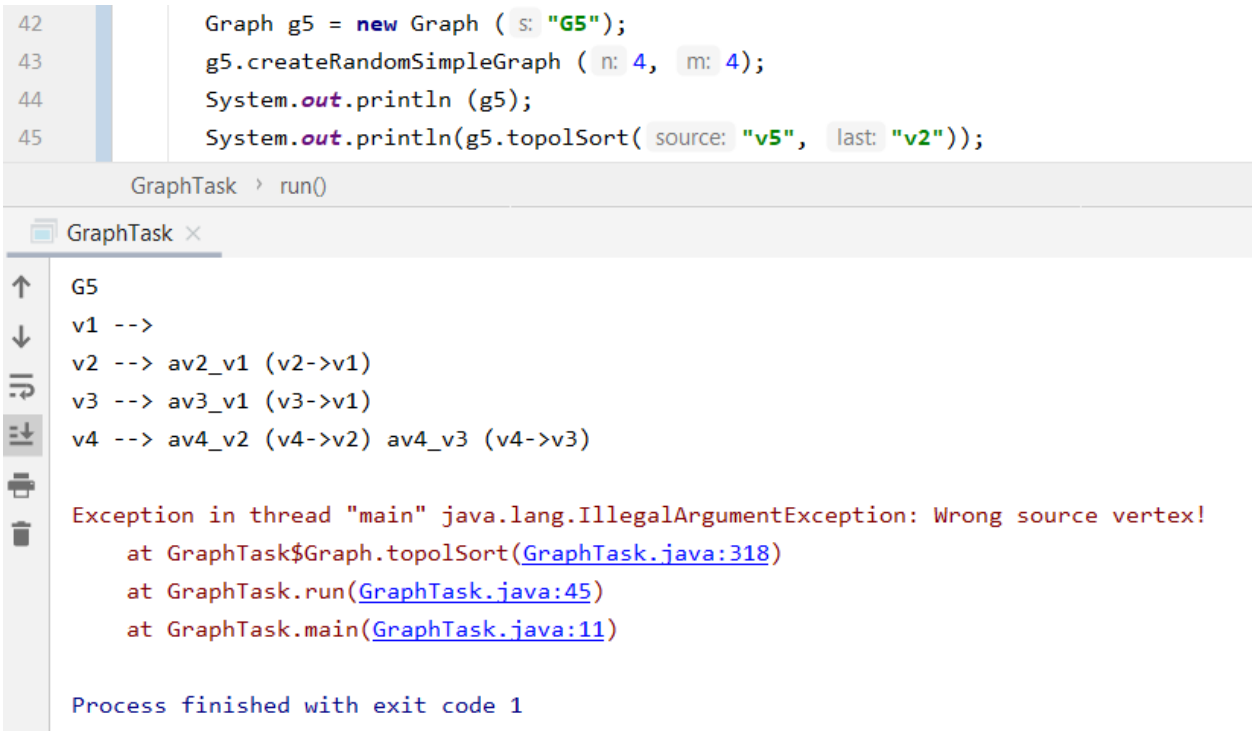
Viies olukord: Sisestan valed tipud. Ootan erandi, sest mul on atsükliline orienteeritud graaf.

Tulemus:



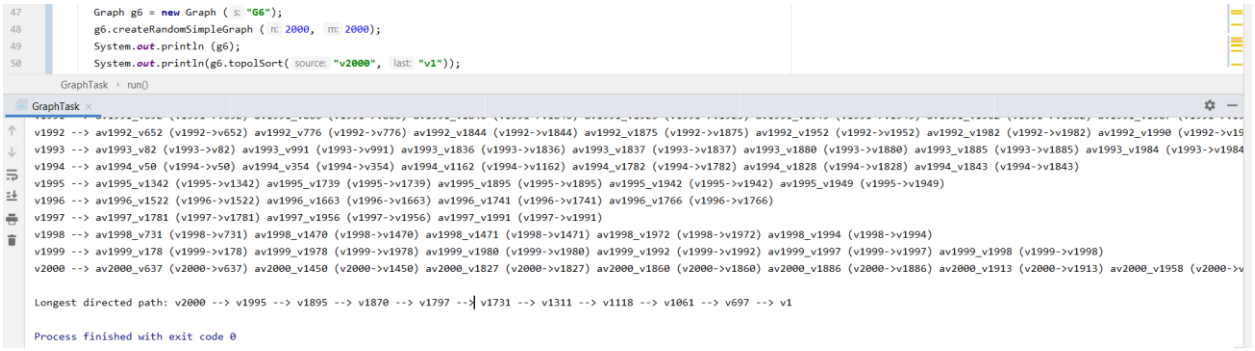
Kuues olukord: Sisetan vale esimene tipp. Tipud ei pea olema rohkem kui oli algselt.

Tulemus:



Seitsmes olukord: Graaf, kus on üle 2000 tuhande tippu. Oodatavalt programm lahendab seda.

Tulemus.



Autor ei saa seda kontrolli, aga võin öelda, et see töötab kiiresti (ümbes 2 sekundit) ja tulemus tundub veenev.

Kasutatud allikad

Graph Online. (2020). Retrieved from https://graphonline.ru/en/

Khan, M. (2011). *mathcs.emory.* Retrieved from mathcs.emory.edu: https://www.mathcs.emory.edu/~cheung/Courses/171/Syllabus/11-Graph/Docs/longest-path-in-dag.pdf

Pöial, J. (2020). *Topoloogiline sorteerimine*. Retrieved from https://enos.itcollege.ee/~jpoial/algoritmid/graafid.html

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) {
      GraphTask a = new GraphTask();
      a.run();
   }

   /** Actual main method to run examples and everything. */
   public void run() {
      Graph g = new Graph ("G0");
      g.createRandomSimpleGraph (3, 3);
      System.out.println (g);
      // Example input: g.topolSort("v4", "v1");
      System.out.println(g.topolSort("v3", "v1"));

//      Graph g1 = new Graph ("G1");
//      g1.createRandomSimpleGraph (5, 7);
//      System.out.println (g1);
//      System.out.println(g1.topolSort("v5", "v2"));
//
//      Graph g2 = new Graph ("G2");
//      g2.createRandomSimpleGraph (10, 12);
//      System.out.println (g2);
//      System.out.println(g2.topolSort("v10", "v4"));
//
//      Graph g3 = new Graph ("G3");
//      g3.createRandomSimpleGraph (5, 10);
//      System.out.println (g3);
//      System.out.println(g3.topolSort("v5", "v1"));
//
//      Graph g4 = new Graph ("G4");
//      g4.createRandomSimpleGraph (4, 4);
//      System.out.println (g4);
//      System.out.println(g4.topolSort("v1", "v4"));
//
//      Graph g5 = new Graph ("G5");
//      g5.createRandomSimpleGraph (4, 4);
//      System.out.println (g5);
//      System.out.println(g5.topolSort("v5", "v2"));
//
//      Graph g6 = new Graph ("G6");
//      g6.createRandomSimpleGraph (2000, 2000);
//      System.out.println (g6);
//      System.out.println(g6.topolSort("v2000", "v1"));


   }

   /** Vertex class for representing vertices in the graph. */
   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;
   }

   public int getVInfo() {
      return info;
   }

   public void setVInfo(int i) {
      this.info = i;
   }

   public boolean hasNext() {
      return next != null;
   }

   public Arc remove() {
       Arc arc = first;
       this.first.is_used = true;
       return arc;
   }

   public Vertex next() {
      return next;
   }

}


/** 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;
   private boolean is_used = false; // instead of remove

   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;
        }

        public boolean hasNext() {
            return next != null;
        }

        public Vertex getToVert() { // get target from arc (Vertex)
            return target;
        }

        public Arc next() { // get next arc
            return this.next;
        }

    }

    /** Graph class.
     */

    class Graph {

        private String id;
        private Vertex first;
        private int info = 0;
        public List<Arc> longest_path = Collections.synchronizedList(new LinkedList());
// represent path like List of Arc

        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) {
                    if (a.target != null) { //replace
                        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();
   }

   private void setGInfo(int i) {
      this.info = i;
   }

   public int getGInfo() {
      return this.info;
   }

   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], null); // replace
         } 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) { // replace
               if (a.target != 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, null);
//replace
            connected [j][i] = 1;
            edgeCount--;  // a new edge happily created
         }
      }
```

```java
    /**
     * Topological sort of vertices.
     * Source: https://enos.itcollege.ee/~jpoial/algoritmid/graafid.html
     * @param source first vertex
     * @param last last vertex
     * @return longest path between two vertex
     */
    public String topolSort(String source, String last) {
       boolean cycleFound = false;
       List<Vertex> order = new ArrayList<>();

       setGInfo (1); // count number of vertices
       Vertex vit = this.first;
       while (vit.hasNext()) {
          vit.next().setVInfo (0);
          setGInfo (getGInfo() + 1);
          vit = vit.next();
       }
       // two if statement for wrong inputs
       if (Integer.parseInt(source.substring(1)) > getGInfo()){
          throw new IllegalArgumentException("Wrong source vertex!");
       }
       if (Integer.parseInt(source.substring(1)) <
Integer.parseInt(last.substring(1)) ){
          throw new IllegalArgumentException("Source vertex should be bigger than
last vertex!");
       }
       vit = this.first;
       while (true) {
          Arc ait = vit.first;
          while (true) {
             if (ait.target != null) {
                Vertex v = ait.getToVert();
                // count number of incoming edges
                v.setVInfo(v.getVInfo() + 1);
             }
             if (!ait.hasNext()) {
                break;
             }
             ait = ait.next();
          }
          if (!vit.hasNext()) {
             break;
          }
          vit = vit.next();
       }
       List start = Collections.synchronizedList(new LinkedList());
        vit = this.first;
       while (true) {
          if (vit.getVInfo() == 0) {
             start.add (vit); // no incoming edges
          }
           if (!vit.hasNext()) {
               break;
           }
           vit = vit.next();
       }
       if (start.size() == 0) cycleFound = true;
       while ((!cycleFound)&(start.size() != 0)) {
          Vertex current = (Vertex)start.remove (0); // first vertex
          order.add (current);
```

```
            Arc ait = current.first;
            while (ait.target != null) {
                Vertex v = ait.getToVert();
                v.setVInfo (v.getVInfo() - 1);
                if (v.getVInfo() == 0) {
                   start.add (v); // no incoming edges anymore
                }
                if (!ait.hasNext()) {
                    break;
                }
                ait = ait.next();
            }
          }
         if (getGInfo() != order.size()) cycleFound = true;
          if (cycleFound) {
             return "Cycle was found!"; // if cycle was found then we can not do
topology sort
          }
          setGInfo (0);
          return findLongestPath(order, source, last);
       } // end of topolSort()

       /**
        * Search longest path in topologically sorted list.
        * @param order List of topologically sorted Vertex
        * @param source first vertex
        * @param last last vertex
        * @return representation of path as a string.
        */
       private String findLongestPath(List<Vertex> order, String source, String last)
{
          String result = "Longest directed path: ";
          Collections.reverse(order);
          List path = Collections.synchronizedList(new LinkedList());
          boolean is_start = false;
          for (Vertex v : order) {
             if (v.id.equals(last)) {
                is_start = true;
                path.add(v);
                continue;
             }
             if (is_start) {
                if (v.id.equals(source)) {
                   path.add(v);
                   Collections.reverse(path);
                   createArcPath(path); // create path of Arc for this Graph
                   result += showLongestPath(path);
                   return result;
                }
                Arc arc = v.first;
                Vertex target = (Vertex) path.get(path.size() - 1);
                while (true) {
                   if (arc.target == target) {
                      path.add(v);
                      break;
                   }
                   if (!arc.hasNext()){
                      break;
                   }
                   arc = arc.next();
                }
             }
```

```
            }
            throw new IllegalArgumentException("Can not build longest path because last
vertex is higher then first vertex!");
        }

        /** Create longest path as List of Arc for Graph. */
        private void createArcPath(List<Vertex> path) {
            for (int i = 0; i < path.size() - 1; i++) {
                Vertex v = path.get(i);
                Arc arc = v.first;
                Vertex target = path.get(i + 1);
                while (true) {
                    if (arc.target == target) {
                        longest_path.add(arc);
                        break;
                    }
                    if (!arc.hasNext()){
                        break;
                    }
                    arc = arc.next();
                }
            }
        }

        /** Represent longest path from List<Vertex>. */
        private String showLongestPath(List<Vertex> path) {
            String result = "";
            for (Vertex v : path) {
                if (path.get(path.size() - 1) == v){
                    result += v.id;
                    return result;
                }
                result += v.id + " --> ";
            }
            throw new IllegalArgumentException("Can not build longest path because last
vertex is higher then first vertex!");
        }
    }
}
```

Lahenduste näited:

```
14      /** Actual main method to run examples and everything. */
15      public void run() {
16          Graph g = new Graph ( s: "G0");
17          g.createRandomSimpleGraph ( n: 3,  m: 3);
18          System.out.println (g);
19          // Example input: g.topolSort("v4", "v1");
20          System.out.println(g.topolSort( source: "v3",  last: "v1"));
21
22          Graph g1 = new Graph ( s: "G1");
23          g1.createRandomSimpleGraph ( n: 5,  m: 7);
24          System.out.println (g1);
25          System.out.println(g1.topolSort( source: "v5",  last: "v2"));
26
27          Graph g2 = new Graph ( s: "G2");
28          g2.createRandomSimpleGraph ( n: 10,  m: 12);
29          System.out.println (g2);
30          System.out.println(g2.topolSort( source: "v10",  last: "v4"));
31
32          Graph g3 = new Graph ( s: "G3");
33          g3.createRandomSimpleGraph ( n: 5,  m: 10);
34          System.out.println (g3);
35          System.out.println(g3.topolSort( source: "v5",  last: "v1"));
36
37          Graph g4 = new Graph ( s: "G4");
38          g4.createRandomSimpleGraph ( n: 4,  m: 4);
39          System.out.println (g4);
40          System.out.println(g4.topolSort( source: "v1",  last: "v4"));
41
42          Graph g5 = new Graph ( s: "G5");
43          g5.createRandomSimpleGraph ( n: 4,  m: 4);
44          System.out.println (g5);
45          System.out.println(g5.topolSort( source: "v5",  last: "v2"));
```