TALLINNA TEHNIKAÜLIKOOL

Individuaaltöö aines "Algoritmid ja andmestruktuurid"

Diana Christine Tarro

IT süsteemide arendus

Juhendaja:

Jaanus Pöial

2020

1 ÜLESANDE PÜSTITUS

Antud individuaaltöö eesmärgiks oli koostada meetod etteantud graafi refleksiivse transitiivse sulundi leidmiseks. Etteantud graaf võis olla nii orienteeritud kui ka orienteerimata – selles osas piiranguid polnud.

Mõned mõisted:

- Graafi transitiivne sulund G^+ saadakse siis kui lähtegraafile G lisatakse kõik niisugused kaared (u, v), mille korral lähtegraafis leidub tee tipust u tippu v.
  Kui lähtegraaf on sidus graaf, siis selle transitiivne sulund on täisgraaf.
- Graafi G refleksiivne transitiivne sulund G^* saadakse transitiivsele sulundile kõigi silmuste lisamisel. [3]

2 LAHENDUSE KIRJELDUS

2.1 Külgnevusstuktuur

Graafi moodustamiseks arvutiprogrammis kasutatakse külgnevusstruktuuri („tippude loetelu, milles iga tipuga on seotud sellest lähtuvate kaarte loetelu (iga kaare kohta on teada tipp, kuhu ta suubub)“[3]). Külgnevusstruktuur luuakse *Graph* klassis kasutades *Vertex* ja *Arc* objekte tippude ning kaarte moodustamiseks. *Graph* klassis on väli, mis viitab esimesele tipule (*Vertex*). Iga tipp hoiab meeles järgmist tippu. *Vertex* klassis on ka väli, mis viitab esimesele kaarele (*Arc*) ning iga kaar hoiab meeles järgmist kaart ning seda, millisesse tippu antud kaar suubub.

2.2 Refleksiivse transitiivse sulundi maatriks

Selleks, et leida graafi refleksiivset transitiivset sulundit, leitakse kõigepealt *Graph* klassis oleva meetodiga *createReflexiveTransitiveClosureMatrix* maatriks, mis väljendab graafi tippude vahelisi ühendusi. Maatriksi read ja veerud tähistavad kõiki graafi tippe. Kui maatriksi mingil positsioonil on „1“, siis tähendab see seda, et sellele positsioonile vastava rea ehk ühe tipu ja veeru ehk teise tipu vahel on ühendus ehk esimesest tipust on võimalik liikuda teise. Orienteerimata graafi puhul tähendab see seda, et on võimalik liikuda ka teisest tipust esimesesse, kuid juhul kui tegu on orienteeritud graafiga, siis ei pruugi nii olla. Kui „1“ asetseb maatriksi peadiagonaalil, siis tähistab see tipus olevat silmust ehk tipu ühendust iseendaga. Kui aga mingil maatriksi positsioonil on „0“, siis tähendab see, et sellele positsioonile vastavast esimesest tipust antud positsioonile vastavasse teise tippu ühendust pole. Kui antud graaf on orienteerimata, siis pole ühendust ka teisest tipust esimesesse. Kui graaf on orienteeritud, siis võib sedapidi ühendus siiski olemas olla. „0“ maatriksi peadiagonaalil tähistab silmuse puudumist tipus.

2.3 Floyd-Warshalli algoritm maatriksis

Graafi refleksiivse transitiivse sulundi maatriksi leidmiseks kasutatakse lihtsustatud varianti Floyd-Warshalli algoritmist ehk maatriksis on kaarte pikkuste asemel ainult ühed ja nullid, mis tähistavad ühendust või selle puudumist tippude vahel. Kõigepealt kopeeritakse graafi külgnevusmaatriks, mis saadakse *createAdjMatrix* meetodiga, tühja maatriksisse, millest saab

hiljem refleksiivse transitiivse sulundi maatriks. Nii on juba soovitud maatriksis olemas otseühendused (ehk ilma vahetippudeta ühendused) tippude vahel. Seejärel kasutatakse mitmeid tsükleid, et leida graafis veel võimalikke ühendusi ning ühenduse leidmise korral väärtustakse „0“, mis on maatriksi antud positsioonil „1“-ks. Kuna tuleb tagastada refleksiivse transitiivse sulundi mitte lihtsalt transitiivse sulundi maatriks, siis väärtustatakse „1“-ks ka kõigi tippude ühendused iseendaga ehk kõik peadiagonaalil olevad väärtused.

2.4 Refleksiivse transitiivse sulundi graaf

Kui maatriks on arvutatud, siis kasutatakse seda *createReflexiveTransitiveClosureGraph* meetodis, mis tagastab graafi, millest antud meetodit välja kutsuti, refleksiivse transitiivse sulundi. Graafi refleksiivne transitiivne sulund on esitatud samasuguse külgnevusstuktuurina, mida on kirjeldatud punktis 2.1.

2.5 Maatriksist graafiks

Selleks, et teisendada refleksiivne transitiivne sulundi maatriks graafiks tehakse alguses valmis tühi graaf, millest saab hiljem etteantud graafi refleksiivne transitiivne sulund. Seejärel tehakse graafi tippude massiiv *Vertex*-tüüpi objektidest ning seotakse see refleksiivse transitiivse sulundiga ehk eelnevalt tehtud tühja graafi objektiga. Nüüd on tekkinud graafile tipud, kuid nende vahelisi seoseid veel pole. Selleks, et graafi tippude vahelised seosed ka graafi lisada, käiakse kahe tsükliga maatriks läbi ning kui maatriksi antud positsioonil on „1“, siis tähendab see ühendust ning see lisatakse graafi kaarena. Lõpuks tagastatakse ülesande lahendus ehk etteantud graafi refleksiivne transitiivne sulund. Kui etteantud graaf oli nullgraaf, siis tsükleid läbi ei käida ning tagastataksegi tühi graaf.

3 PROGRAMMI KASUTAMISJUHEND

Etteantud graafi refleksiivse transitiivse sulundi leidmiseks tuleb kõigepealt luua üks graaf, mis on *Graph*-tüüpi objekt. Seda võib teha nt *createRandomTree* või *createRandomSimpleGraph* meetodiga. Kumbki neist tagastab etteantud suurusega juhusliku graafi.

```
// directed graph
Graph g = new Graph( s: "Directed graph");
g.createRandomTree( n: 5,  isUndirected: false);
```

Joonis 1 Graafi loomine

Tekkinud graafi nägemiseks võib selle välja printida.

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

Joonis 2 Graafi printimine

Klassis *Graph* on meetod *toString*, mida automaatselt printimisel kasutatakse ja see teeb graafist ilusa ja arusaadava kuju.

```
Directed graph
v1 -->
v2 -->
v3 -->
v4 -->
v5 --> av5_v1 (v5->v1) av5_v2 (v5->v2) av5_v3 (v5->v3) av5_v4 (v5->v4)
```

Joonis 3 Graafi printimise tulemus terminalis

Seejärel saab kasutada *createReflexiveTransitiveClosureGraph* meetodit, mis tagastab uue *Graph* objekti, milleks on etteantud graafi refleksiivne transitiivne sulund. Antud meetodile ei ole vaja parameetreid ette anda.

```
Graph rtc = g.createReflexiveTransitiveClosureGraph();
```

Joonis 4 Etteantud graafi refleksiivse transitiivse sulundi loomine

Tekkinud refleksiivse transitiivse sulundi nägemiseks võib selle välja printida.

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

Joonis 5 Refleksiivse transitiivse sulundi printimine

```
Reflexive transitive closure of graph: Directed graph
v1 --> av1_v1 (v1->v1)
v2 --> av2_v2 (v2->v2)
v3 --> av3_v3 (v3->v3)
v4 --> av4_v4 (v4->v4)
v5 --> av5_v1 (v5->v1) av5_v2 (v5->v2) av5_v3 (v5->v3) av5_v4 (v5->v4) av5_v5 (v5->v5)
```

Joonis 6 Refleksiivse transitiivse sulundi printimise tulemus terminalis

4 TESTIMISKAVA

Igas lahendusnäites luuakse üks graaf ning esimeses neljas lahendusnäites prinditakse see koos graafi külgnevusmaatriksiga välja. Seejärel leitakse selle graafi refleksiivne transitiivne sulund ning esimeses neljas lahendusnäites prinditakse ka see koos külgnevusmaatriksiga välja. Viimase kahe näite lahendustulemusi välja ei prindita, kuna tegu on selleks liiga suurte graafidega. Nende puhul prinditakse välja programmi tööaeg. Selleks, et saada võimalikult täpset tööaega kommenteeriti teised testimiseks loodud lahendusnäited välja ning testitigi vaid ühte konkreetset graafi korraga. Tööaega testiti kolm korda ja neist arvutati keskmine tööaaeg.

4.1 Nullgraaf

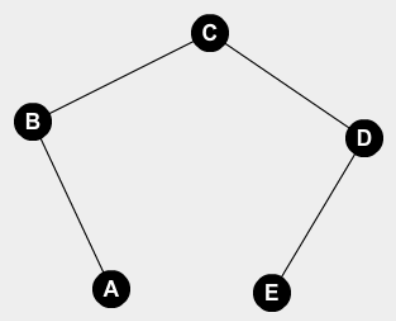
Kui etteantud graaf on nullgraaf, siis on selle refleksiivseks transitiivseks sulundiks samuti nullgraaf (Lisa 2.1).

4.2 Ühe tipuga graaf

Kui etteantud graaf on vaid ühe tipuga, siis selle refleksiivseks transitiivseks sulundiks on graaf, kus on endiselt üks tipp, kuid sellel silmus (Lisa 2.2).

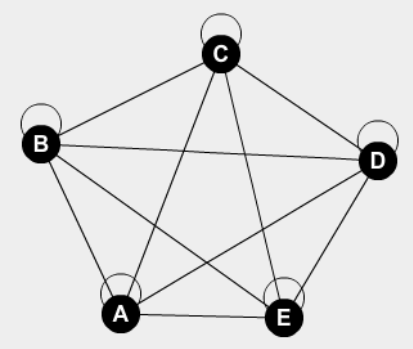
4.3 Orienteerimata graaf

Etteantud graafiks on viietipuline orienteerimata graaf (joonis 7).



Joonis 7 Orienteerimata graaf

Selle graafi refleksiivseks transitiivseks sulundiks saadi graaf joonisel 8.

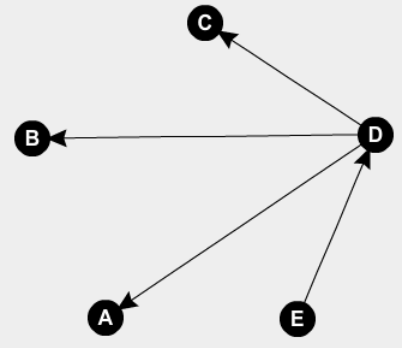


Joonis 8 Refleksiivne transitiivne sulund

Tulemus on õige, sest igal tipul on tekkinud ühendus iseendaga ning kuna etteantud graaf oli orienteerimata, siis on tekkinud ühendus ka kõikide graafi punktide vahel. (Lisa 2.3)

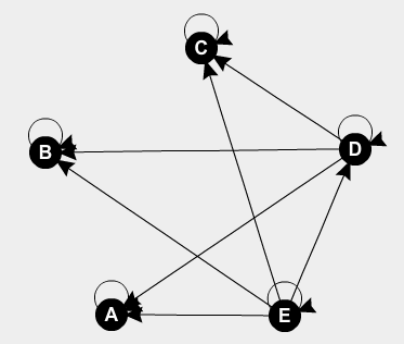
4.4 Orienteeritud graaf

Etteantud graafiks on viietipuline orienteeritud graaf (joonis 9).



Joonis 9 Orienteeritud graaf

Selle graafi refleksiivseks transitiivseks sulundiks saadi graaf joonisel 10.



Joonis 10 Refleksiivne transitiivne sulund

Tulemus on õige, sest igal tipul on tekkinud ühendus iseendaga ning lisaks on tekkinud viiendast tipust (E) ühendus igasse graafi tippu. (Lisa 2.4)

4.5 Väga suur orienteerimata graaf

Etteantud graaf on 2000 tipuline orienteerimata graaf. Testiti programmi tööaega. (Lisa 2.5)

1. tulemus: 15597 ms

2. tulemus: 15116 ms

3. tulemus: 13558 ms

Programmi keskmine tööaeg oli ~14757 ms.

4.6 Väga suur orienteeritud graaf

Etteantud graaf on 2000 tipuline orienteeritud graaf. Testiti programmi tööaega. (Lisa 2.6)

1. tulemus: 14657

2. tulemus: 14510

3. tulemus: 13270

Programmi keskmine tööaeg oli ~14146 ms.

5 KASUTATUD ALLIKAD

[1] https://bitbucket.org/itc_algorithms/kt6.git (viimati külastatud 19.11.10)

[2] https://www.geeksforgeeks.org/transitive-closure-of-a-graph/ (viimati külastatud 19.11.20)

[3] https://enos.itcollege.ee/~jpoial/algoritmid/graafid.html (viimati külastatud 20.11.20)

6 LISAD

Lisa 1 – Programmi kood

```java
import java.util.*;

/* ALLIKAD:
https://bitbucket.org/itc_algorithms/kt6.git
https://www.geeksforgeeks.org/transitive-closure-of-a-graph/
*/

/** 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() {
      // empty graph
      Graph g = new Graph("Empty graph");
      g.createRandomTree(0, true);
      System.out.println(g);
      System.out.println(Arrays.deepToString(g.createAdjMatrix()));  // empty list

      Graph rtc = g.createReflexiveTransitiveClosureGraph();
      System.out.println(rtc);
      System.out.println(Arrays.deepToString(rtc.createAdjMatrix()));  // empty list

      // one vertex graph
      g = new Graph("One vertex graph");
      g.createRandomTree(1, true);
      System.out.println(g);
      System.out.println(Arrays.deepToString(g.createAdjMatrix()));  // [[0]]

      rtc = g.createReflexiveTransitiveClosureGraph();
      System.out.println(rtc);
      System.out.println(Arrays.deepToString(rtc.createAdjMatrix()));  // [[1]]

      // undirected graph
      g = new Graph("Undirected graph");
      g.createRandomTree(5, true);
      System.out.println(g);
      System.out.println(Arrays.deepToString(g.createAdjMatrix()));

      rtc = g.createReflexiveTransitiveClosureGraph();
      System.out.println(rtc);
      System.out.println(Arrays.deepToString(rtc.createAdjMatrix()));

      // directed graph
      g = new Graph("Directed graph");
      g.createRandomTree(5, false);
      System.out.println(g);
      System.out.println(Arrays.deepToString(g.createAdjMatrix()));

      rtc = g.createReflexiveTransitiveClosureGraph();
```

Joonis 11 Programmi kood 1

```
        System.out.println(rtc);
        System.out.println(Arrays.deepToString(rtc.createAdjMatrix()));

        // huge undirected graph
        g = new Graph("Huge undirected graph");
        g.createRandomTree(2000, true);
        System.out.println(g);
        System.out.println(Arrays.deepToString(g.createAdjMatrix()));

        rtc = g.createReflexiveTransitiveClosureGraph();
        System.out.println(rtc);
        System.out.println(Arrays.deepToString(rtc.createAdjMatrix()));

        // huge directed graph
        g = new Graph("Huge directed graph");
        g.createRandomTree(2000, false);
        System.out.println(g);
        System.out.println(Arrays.deepToString(g.createAdjMatrix()));

        rtc = g.createReflexiveTransitiveClosureGraph();
        System.out.println(rtc);
        System.out.println(Arrays.deepToString(rtc.createAdjMatrix()));

    }


    /**
     * Vertex class.
     */
    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);
        }

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


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

        private String id;
        private Vertex target;
```

Joonis 12 Programmi kood 2

```
    private Arc next;

    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;
    }
 }


 /**
  * Graph class.
  */
 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, 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;
       }
```

Joonis 13 Programmi kood 3

```
      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, boolean isUndirected) {
      if (n <= 0)
         return;
      Vertex[] varray = new Vertex[n];
      for (int i = 0; i < n; i++) {
         varray[i] = createVertex("v" + (n - i));
         if (i > 0) {
            int vnr = (int) (Math.random() * i);
            createArc("a" + varray[vnr].toString() + "_"
                  + varray[i].toString(), varray[vnr], varray[i]);
            if (isUndirected) {
               createArc ("a" + varray [i].toString() + "_"
                     + varray [vnr].toString(), varray [i], varray [vnr]);
            }
         }
      }
   }

   /**
    * 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) {
```

Joonis 14 Programmi kood 4

```
            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, true);       // 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
      }
   }

   /**
    * Create reflexive transitive closure matrix of this graph.
    * It is later used to create reflexive transitive closure graph.
    * Side effect: corrupts info fields in the graph
    *
    * @return Matrix that represents reflexive transitive closure of this graph
    */
   public int[][] createReflexiveTransitiveClosureMatrix() {
      info = 0;
```

Joonis 15 Programmi kood 5

```
            Vertex v = first;
            while (v != null) {
               v.info = info++;
               v = v.next;
            }

            int[][] refTransMatrix = new int[info][info];
            int[][] adjMatrix = createAdjMatrix();

            // copy one matrix to another
            for (int i = 0; i < info; i++)
               for (int j = 0; j < info; j++)
                  refTransMatrix[i][j] = adjMatrix[i][j];

            for (int k = 0; k < info; k++) {
               for (int i = 0; i < info; i++) {
                  for (int j = 0; j < info; j++) {
                     refTransMatrix[i][j] = (refTransMatrix[i][j] != 0) || ((refTransMatrix[i][k]
!= 0) && (refTransMatrix[k][j] != 0)) || (i == j) ? 1 : 0;
                  }
               }
            }
            return refTransMatrix;
         }

         /**
          * Create reflexive transitive closure of this graph.
          * @return New graph object that is reflexive transitive closure of this graph
          */
         public Graph createReflexiveTransitiveClosureGraph() {
            int[][] matrix = createReflexiveTransitiveClosureMatrix();
            Graph reflexiveTransitiveClosureGraph = new Graph("Reflexive transitive closure of
graph: " + this.id);

            Vertex[] varray = new Vertex[matrix.length];
            for (int i = matrix.length-1; i >= 0; i--) {
               varray[i] = reflexiveTransitiveClosureGraph.createVertex("v" + (i + 1));
            }

            for (int i = matrix.length-1; i >= 0; i--) {
               for (int j = matrix.length-1; j >= 0; j--) {
                  if (matrix[i][j] == 1) {
                     reflexiveTransitiveClosureGraph.createArc("a" + varray[i].toString() + "_" +
varray[j].toString(), varray[i], varray[j]);
                  }
               }
            }
            return reflexiveTransitiveClosureGraph;
         }
      }
}
```

Joonis 16 Programmi kood 6

Lisa 2 – Lahendusnäidete tulemused

2.1 - Nullgraaf

```java
// empty graph
Graph g = new Graph( s: "Empty graph");
g.createRandomTree( n: 0,  isUndirected: true);
System.out.println(g);
System.out.println(Arrays.deepToString(g.createAdjMatrix()));  // empty list

Graph rtc = g.createReflexiveTransitiveClosureGraph();
System.out.println(rtc);
System.out.println(Arrays.deepToString(rtc.createAdjMatrix()));  // empty list
```

Joonis 17 Lahendusnäite kood

```
Empty graph

[]

Reflexive transitive closure of graph: Empty graph

[]
```

Joonis 18 Lahendusnäite tulemus terminalis

2.2 Ühe tipuga graaf

```java
// one vertex graph
g = new Graph( s: "One vertex graph");
g.createRandomTree( n: 1,  isUndirected: true);
System.out.println(g);
System.out.println(Arrays.deepToString(g.createAdjMatrix()));  // [[0]]

rtc = g.createReflexiveTransitiveClosureGraph();
System.out.println(rtc);
System.out.println(Arrays.deepToString(rtc.createAdjMatrix()));  // [[1]]
```

Joonis 19 Lahendusnäite kood

```
One vertex graph
v1 -->

[[0]]

Reflexive transitive closure of graph: One vertex graph
v1 --> av1_v1 (v1->v1)

[[1]]
```

Joonis 20 Lahendusnäite tulemus terminalis

2.3 Orienteerimata graaf

```
// undirected graph
g = new Graph( s: "Undirected graph");
g.createRandomTree( n: 5,  isUndirected: true);
System.out.println(g);
System.out.println(Arrays.deepToString(g.createAdjMatrix()));

rtc = g.createReflexiveTransitiveClosureGraph();
System.out.println(rtc);
System.out.println(Arrays.deepToString(rtc.createAdjMatrix()));
```

Joonis 21 Lahendusnäite kood

```
Undirected graph
v1 --> av1_v2 (v1->v2)
v2 --> av2_v1 (v2->v1) av2_v3 (v2->v3)
v3 --> av3_v2 (v3->v2) av3_v4 (v3->v4)
v4 --> av4_v3 (v4->v3) av4_v5 (v4->v5)
v5 --> av5_v4 (v5->v4)

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

Reflexive transitive closure of graph: Undirected graph
v1 --> av1_v1 (v1->v1) av1_v2 (v1->v2) av1_v3 (v1->v3) av1_v4 (v1->v4) av1_v5 (v1->v5)
v2 --> av2_v1 (v2->v1) av2_v2 (v2->v2) av2_v3 (v2->v3) av2_v4 (v2->v4) av2_v5 (v2->v5)
v3 --> av3_v1 (v3->v1) av3_v2 (v3->v2) av3_v3 (v3->v3) av3_v4 (v3->v4) av3_v5 (v3->v5)
v4 --> av4_v1 (v4->v1) av4_v2 (v4->v2) av4_v3 (v4->v3) av4_v4 (v4->v4) av4_v5 (v4->v5)
v5 --> av5_v1 (v5->v1) av5_v2 (v5->v2) av5_v3 (v5->v3) av5_v4 (v5->v4) av5_v5 (v5->v5)

[[1, 1, 1, 1, 1], [1, 1, 1, 1, 1], [1, 1, 1, 1, 1], [1, 1, 1, 1, 1], [1, 1, 1, 1, 1]]
```

Joonis 22 Lahendusnäite tulemus terminalis

2.4 Orienteeritud graaf

```
// directed graph
g = new Graph( s: "Directed graph");
g.createRandomTree( n: 5,  isUndirected: false);
System.out.println(g);
System.out.println(Arrays.deepToString(g.createAdjMatrix()));

rtc = g.createReflexiveTransitiveClosureGraph();
System.out.println(rtc);
System.out.println(Arrays.deepToString(rtc.createAdjMatrix()));
```

Joonis 23 Lahendusnäite kood

```
Directed graph
v1 -->
v2 -->
v3 -->
v4 --> av4_v1 (v4->v1) av4_v2 (v4->v2) av4_v3 (v4->v3)
v5 --> av5_v4 (v5->v4)

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

Reflexive transitive closure of graph: Directed graph
v1 --> av1_v1 (v1->v1)
v2 --> av2_v2 (v2->v2)
v3 --> av3_v3 (v3->v3)
v4 --> av4_v1 (v4->v1) av4_v2 (v4->v2) av4_v3 (v4->v3) av4_v4 (v4->v4)
v5 --> av5_v1 (v5->v1) av5_v2 (v5->v2) av5_v3 (v5->v3) av5_v4 (v5->v4) av5_v5 (v5->v5)

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

Joonis 24 Lahendusnäite tulemus terminalis

2.5 Väga suur orienteerimata graaf

```java
// huge undirected graph
long start = System.currentTimeMillis();
Graph g = new Graph( s: "Huge undirected graph");
g.createRandomTree( n: 2000,  isUndirected: true);
Graph rtc = g.createReflexiveTransitiveClosureGraph();
System.out.println(System.currentTimeMillis() - start);
```

Joonis 25 Lahendusnäite kood

2.6 Väga suur orienteeritud graaf

```java
// huge directed graph
long start = System.currentTimeMillis();
Graph g = new Graph( s: "Huge directed graph");
g.createRandomTree( n: 2000,  isUndirected: false);
Graph rtc = g.createReflexiveTransitiveClosureGraph();
System.out.println(System.currentTimeMillis() - start);
```

Joonis 26 Lahendusnäite kood