TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Gert Kuldma 192847IADB

Individuaaltöö aines "Algoritmid ja andmestruktuurid"

Juhendaja: Jaanus Pöial

Tallinn 2021

Sisukord

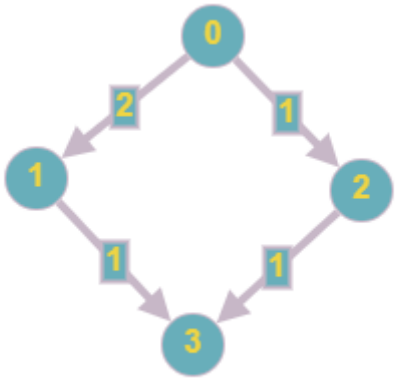
Jooniste loetelu

Koodi loetelu

1 Ülesande püstitus

Käesoleva töö ülesandeks on raamatus J.Kiho. „A&A ülesannete kogu“ ülessanne 6 leheküljel 14. Probleemiks on suunatud geograafiliste punktidega graafilt kahe punkti vahelise ohutuima tee leidmine. Ohutuim teekond on vastav teekond kus punktist x liikudes punkti y on lokaalne langus (langus punktide x ja y vahel) minimaalne. Seega on vajalik kirjutada algoritm mis läbiks graafi laiuti ja võrdleks igal sammul lokaalset langust ja valiks selle arvestusega ohutuima teekonna.

Joonis 1. Orienteeritud graafi näide



Joonis 1

Joonisel 1. on ohutuimaks teeks liikudes punktist 0->3 vastavalt ohutuima tee definitsioonile 0->2->3 selleks, et punktist puntki liikumisel oleks lokaalne langus minimaalne.

2 Programmi kirjeldus

2.1 Klassid

2.1.1 Vertex

Vertex (eesti keeles tipp) tähistab antud ülesandes geograafilist punkti. Klassil on 6 muutujad:

- Id – tippu identifitseeriv kood
- Next – Vertex objekt mis tähistab järgmist tippu
- First – Arc objekt kaar millega tipp on seotud
- Info – int tüüpi väli informatsiooni hoidmiseks
- Previous – Vertex object mis sisaldab infot eelmise punkti kohta
- Visited- Boolean tüüpi väärtus mis on kasutusel infosalvestuseks otsingu käigus külastuse märkimiseks.

Klassil Vertex on 2 meetodit ja 2 konstruktorit.

Meetodid:

- toString() – Kirjutab välja Vertex objekti id
- getAllArcs() – tagastab Vertex objektiga seotud Arc objektid loendina

Kontstruktorid:

- Saab luua Vertex objekti määrates parameetri: id
- Saab luua Vertex objekti määrates parameetrid: id, next ja first

Kood 1. Vertex klass

```
class Vertex {

   private String id;
   private Vertex next;
   private Arc first;
   private int info = 0;
   private Vertex previous = null;
   private Boolean visited = 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 String toString() { return id; }

   public Iterator getAllArcs(){
      List<Arc> result = new ArrayList<>();
      Arc a = first;
      while (a != null) {
         result.add(a);
         a = a.next;
      }
      return result.iterator();
   }
}
```

2.1.2 Arc

Arc (eesti keeles kaar) tähistab antud ülesandes kahe geograafilise punkti vahel asuvat ühesuunalist ühendust millele on antud väärtusena langus liikumisel puntkist x puntki y. Klassil on 6 muutujad:

- Id – kaart identifitseeriv kood
- Next – järgmine kaar mis on seotud sama Vertex objekti punktiga
- Target – Vertex objekt kuhu kaar on suunatud

- Info – int tüüpi väli informatsiooni hoidmiseks
- From – Vertex object mis sisaldab infot kaare algus punkti kohta
- declineValue - int tüüpi väärtus mis on kasutusel infosalvestuseks kaare languse märkimiseks.

Klassil Arc on 1 meetod ja 2 konstruktorit.

Meetodid:

- toString() – Kirjutab välja Arc objekti id ja tema languse väärtuse

Kontstruktorid:

- Saab luua Arc objekti määrates parameetri: id
- Saab luua Arc objekti määrates parameetrid: id, next ja target

Kood 2. Arc klass

```
class Arc {

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

   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 + " /" + declineValue + "/"; }
}
```

2.1.3 Graph

Graph (eesti keeles graaf) tähistab antud ülesandes geograafilise punktide kogumikku mis on omavahel ühendatud suunatud kaartega. Klassil on 3 muutujat:

- Id – graafi identifitseeriv kood
- First – esimene Vertex objekt graafil
- Info – int tüüpi väli informatsiooni hoidmiseks

Klassil Graph on 10 meetodit ja 2 konstruktorit.

Meetodid:

- toString() – Kirjutab välja graafi Vertex objektid ja nende vahelised Arc objektid

Kood 3. Graph.toString() meetod

```
@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();
}
```

- createVertex(String vid) – meetod uue Vertex objekti loomiseks ja selle liitmiseks graafi, tagastab loodud Vertex objekti

Kood 4. Graph.createVertex(Stirng vid) meetod

```
public Vertex createVertex (String vid) {
   Vertex res = new Vertex (vid);
   res.next = first;
   first = res;
   return res;
}
```

- createArc(String aid, int declineValue, Vertex from, Vertex to) – meetod uue Arc objekti loomiseks mis seob graafil kaks Vertex objekti omavahel. Tagastab loodud Arc objekti.

Kood 5. Graph. createArc(String aid, int declineValue, Vertex from, Vertex to) meetod

```
public Arc createArc (String aid, int declineValue, Vertex from, Vertex to) {
   Arc res = new Arc (aid);
   res.declineValue = declineValue;
   res.next = from.first;
   res.from = from;
   from.first = res;
   res.target = to;
   return res;
}
```

- createRandomTree(int n) – meetod suvalise orienteeritud puu loomiseks, sisendiks on tippude ehk Vertex objektide hulk soovitud puul.

Kood 6. Graph.createRandomTree(int n) meetod

```
public void createRandomTree (int n) {
   if (n <= 0)
      return;
   Vertex[] varray = new Vertex [n];
   int declineValue;
   for (int i = 0; i < n; i++) {
      varray [i] = createVertex ( vid: "v" + (n - i));
      declineValue = (int) (Math.random()*100);
      if (i > 0) {
         int vnr = (int)(Math.random()*i);
         createArc ( aid: "a" + varray [vnr].toString() + "_"
            + varray [i].toString(), declineValue, varray [vnr], varray [i]);
         } else {}
   }
}
```

- createAdjMatrix() – meetod graafi tippude omavaheliste seoste leidmiseks. Tagastab int[][] maatriksi kus on märgitud tipud ja nende vahelised seosed. Juhul

kui kaks tippu on omavahel ühendatud on vastavas punktis maatriksis väärtus 1 ja juhul kui otseühendus puudub siis 0.

Kood 7. Graph.createAdjMatrix()

```
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.declineValue;
         a = a.next;
      }
      v = v.next;
   }
   return res;
}
```

- createRandomSimpleGraph(int n, int m) – meetod suvalise orienteeritud graafi loomiseks. Sisendiks on tippude ehk Vertex objektide arv n ja kaarte ehk Arc objektide arv m. Meetod loob uue suvalise orienteeritud graafi kus on n tippu ja m tippe ühendavat kaart. Tippude arv on ette nähtud kuni 2000 ja kaarte arv on ette nähtud vastavalt seostele:

  1) m < n - 1

  2) m > n * (n – 1) / 2

  Antud seosed on vajalikud, et ei tekiks karate puudu ega ülejääki.

Kood 8. Graph. createRandomSimpleGraph(int n, int m)

```
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);

   int declineValue;

   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
      declineValue = (int) (Math.random()*100);
      Vertex vi = vert [i];
      Vertex vj = vert [j];
      createArc ( aid: "a" + vi.toString() + "_" + vj.toString(), declineValue, vi, vj);
      connected [i][j] = 1;
      edgeCount--;  // a new edge happily created
   }
}
```

- getAllVertices() – meetod mis tagastab kõik graafil olevad tipud iteraatori objektina

Kood 9. Graph.getAllVertices()

```
public Iterator getAllVertices(){
   List<Vertex> result = new ArrayList<>();
   Vertex v = first;
   while (v != null) {
      result.add(v);
      v = v.next;
   }
   return result.iterator();
}
```

- getNeighbours(Vertex v) – meetod mis saades sisendiks tipu, ehk Vertex objekti, tagastab iteraatori objekti kõigist antud tipu naabertippudest millega tipp on otse seotud.

Kood 10. Graph.getAllNeighbours(Vertex v)

```
public Iterator getNeighbours(Vertex v){
   List<Arc> result = new ArrayList<>();
   if (v.first != null){
      Arc a = v.first;
      result.add(a);
      while (a.next != null){
         result.add(a.next);
         a = a.next;
      }
   }
   return result.iterator();
}
```

- getSafestPathFromTo(String start, String finish) – meetod mille sisenditeks on start, sõne tüüpi muutuja mille väärtus on esimese tipu id, ja finish, sõne tüüpi muutuja mille väärtus on teekonna viimase tipu id kuhu on soovitud jõuda. Meetod tagastab loendi kõikide kaarte ehk Arc objektidega mida kasutatakse algus punktist lõpp punkti jõudmiseks.

Kood 11. Graph.getSafestPathFromTo(String start, String finish)

```
public List<Arc> getSafestPathFromTo(String start, String finish) {
    if (start.equals(finish)){
        throw new RuntimeException("Start point can not be same as finish: " + start);
    }
    int infinity = Integer.MAX_VALUE / 4;
    Iterator vertices = getAllVertices();
    boolean startExists = false;
    boolean finishExists = false;
    while (vertices.hasNext()){
        Vertex v = (Vertex) vertices.next();
        v.info = infinity;
        if (v.id.equals(start)){
            startExists = true;
        }
        if (v.id.equals(finish)){
            finishExists = true;
        }
    }
    if (!startExists){
        throw new RuntimeException("Start point does not exist with id: " + start);
    }
    if (!finishExists){
        throw new RuntimeException("Finish point does not exist with id: " + finish);
    }
    Vertex startVertex = getVertexById(start);
    startVertex.info = 0;
    List<Vertex> bufferList = new ArrayList<>();
    bufferList.add(startVertex);
```

```
while (bufferList.size()>0){
    int minLen = infinity+1;
    Vertex minChoice = null;
    Iterator bufferIterator = bufferList.iterator();
    while (bufferIterator.hasNext()){
        Vertex v = (Vertex) bufferIterator.next();
        if(v.info < minLen) {
            minChoice = v;
            minLen = v.info;
        }
    }
    if (minChoice == null) {
        break;
    }
    bufferList.remove(minChoice);
    bufferIterator = getNeighbours(minChoice);
    int arcCheckValue = infinity;
    Boolean allArcsChecked = false;
    Vertex saveVertex = null;
    Arc saveArc = null;

while (bufferIterator.hasNext()){
    Arc a = (Arc) bufferIterator.next();
    if (!bufferIterator.hasNext()){
        allArcsChecked = true;
    }
    minLen = a.declineValue;
    Vertex to = a.target;
    if (to.visited){
        continue;
    }
    if (to.info == infinity){
        bufferList.add(to);
    }
    if (minLen<arcCheckValue){
        arcCheckValue = minLen;
        saveVertex = to;
        saveArc = a;
    }
    if (allArcsChecked){
        saveVertex.info = minLen;
        saveVertex.previous = minChoice;
        saveVertex.previous.first = saveArc;
        saveVertex.visited = true;
    }
```

```
            if (saveVertex.id.equals(finish) && allArcsChecked){
                List<Arc> path = new ArrayList<>();
                while (saveVertex.previous != null){
                    if (saveVertex.previous.first != null){
                        if (path.contains(saveVertex.previous.first)){
                            break;
                        }
                        path.add(saveVertex.previous.first);
                    }
                    saveVertex = saveVertex.previous;
                }
                List<Arc> pathReverse = new ArrayList<>();
                for (int i = path.size()-1; i >= 0; i--){
                    pathReverse.add(path.get(i));
                }
                return pathReverse;
            }
        }
    }
    throw new RuntimeException("Path not found between " + start + "->" + finish);
}
```

Kontstruktorid:

- Saab luua Graph objekti määrates parameetri: id
- Saab luua Graph objekti määrates parameetrid: id ja first

3 Programmi kasutamine

3.1 Graafi loomine

Graafi loomiseks tuleb luua uus Graph object ja anda talle sisendiks kaasa nimetus ehk id sõne tüüpi muutujana.

Kood 12. uue Graph objekti loomine

```
Graph test = new Graph( s: "G");
```

3.2 Käsitsi tippude lisamine

Tippude ehk Vertex objektide lisamiseks on vajalik luua kõik uued Vertex objektid ja anda neile sisendina kaasa tipu unikaalne nimi ehk id sõne tüüpi sisendina. Uue tipu loomiseks kasutame Graph klassi meetodit createVertex(String vid). Kood 13. esindab näidet kus loome uue tipu klassi id-ga „v1“

Kood 13. Uue tipu loomine graafile

```
Vertex v1 = test.createVertex( vid: "v1");
```

3.3 Käsitsi kaarte lisamine

Kaarte ehk Arc objektide loomiseks on vajalik luua kõik uued Arc objektid ja anda neile sisendina kaasa kaare unikaalne nimi ehk id, arvväärtus kaare languse kohta ja tipud Vertex objektidena, mille vahel kaar paikneb. Kaare loomiseks ja graafile lisamiseks kasutama Graph klassi meetodit createArc(String aid, int declineValue, Vertex start, Vertex end). Kood 14. näide kirjeldab uue Arc objekti loomist mille id on „arc_v1_v5“, langus 2, algab punktist v1 ja lõppeb punktis v5.

Kood 14. Uue kaare loomine graafil kahe tipu vahele

```
test.createArc( aid: "arc_v1_v5",  declineValue: 2, v1, v5);
```

3.4 Graafi tippude ja kaarte lisamine juhuslikkuse alusel

Loomaks kindlal hulgal tippe ja neid ühendavaid kaari juhuslikkuse alusel graafile on kasutusel Graph klassi meetod createRandomSimpleGraph(int n, int m). Kood 15. kujutab käsku mis loob vastavalt soovitud hulga tippude ja kaartega uue suunatud graafi.

Kood 15. Juhuslikkuse alusel graafi loomine

```
Graph g = new Graph ( s: "G");
g.createRandomSimpleGraph (vertexCount, edgeCount);
```

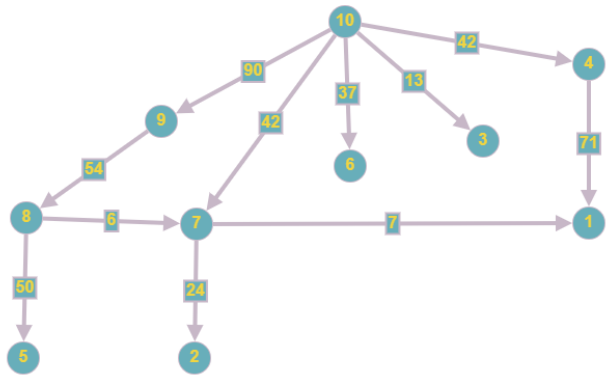
Joonisel 2 näeme antud juhuslikkuse alusel koostatud graafi väljatrükki kus on sisendandmetena näitena kasutatud 10 tippu ja 11 neid ühendavat kaart.

Joonis 2. Juhuslikkuse alusel koostatud graafi väljaprint

```
G
v1 -->
v2 -->
v3 -->
v4 --> av4_v1 /71/ (v4->v1)
v5 -->
v6 -->
v7 --> av7_v1 /7/ (v7->v1) av7_v2 /24/ (v7->v2)
v8 --> av8_v7 /6/ (v8->v7) av8_v5 /50/ (v8->v5)
v9 --> av9_v8 /54/ (v9->v8)
v10 --> av10_v3 /13/ (v10->v3) av10_v4 /42/ (v10->v4) av10_v6 /37/ (v10->v6) av10_v7 /42/ (v10->v7) av10_v9 /90/ (v10->v9)
```

Vastav graaf joonistatuna on kujutatud Joonis 3.

Joonis 3. Eelnevalt genereeritud graafi visuaalne kujutus



3.5 Ohutuima tee leidmine

Graafis ohutuima tee leidmiseks läbi tippude kasutades selleks nende vahel olevaid kaari on loodud Graph klassis meetod getSafestPathFromTo(String start, String finish).

Kasutamiseks on vaja peale graafi loomist, tippude lisamist ja nende vaheliste kaarte loomist käivitada antud meetod kahe sisendiga: sõne mis sisaldab algustipu id-d ja sõne mis sisaldab lõpp punkti id-d. Näide antud meetodi kasutamisest on toodud Kood 16.

Kood 16. Graph.getSafestPathFromTo(String start, String finish) kasutamine

```
List<Arc> safestPath = test.getSafestPathFromTo( start: "v1",  finish: "v9");
```

Kuna meetod tagastab meile loendi tee läbimisel kasutatud kaartest siis peame muutuja loomisel sellega arvestama. Vastus mille saame kujutab endast ohutuimat teed mida läbides on lokaalne langus punktist punkti minimaalne. Näide on toodu Joonis 4.

Joonis 4. Ohutuima tee meetodi väljatrükk

```
[arc_v1_v5 /2/, arc_v5_v8 /3/, arc_v8_v9 /4/]
```

Täpsen näide koos joonistega on välja toodud testide peatükis.

3.5.1 Ohutuima tee algoritm

Ohutuima tee leidmise meetod algab kontrollidega mis on vajalikud algseks sisendandmete ülevaatuseks. Esimene kontroll on, et start ei võrduks finish (Kood 17)

Kood 17. Kontroll, et start ei võrdu finish

```
if (start.equals(finish)){
    throw new RuntimeException("Start point can not be same as finish: " + start);
}
```

Edasi loome endale nii-õelda lõpmatuse väärtuse mis kujutab infona suurimat võimalikku eksisteerivat väärtust (Kood 18).

Kood 18. Lõpmatuse muutuja

. Lõpmatuse muutuja

```
int infinity = Integer.MAX_VALUE / 4;
```

Järgmise sammuna loome iteraator objekti kuhu kogume kokku kõik tipud loodud graafil kasutades meetodit Graph.getAllVertices() (Kood 9). Selleks, et veenduda tee leidmise võimalikkuses kontrollime start tipu ja finish tipu olemas olu (Kood 19).

Kood 19. Kõikide tippude iteraatori loomine ja stardi/finishi olemasolu kontroll

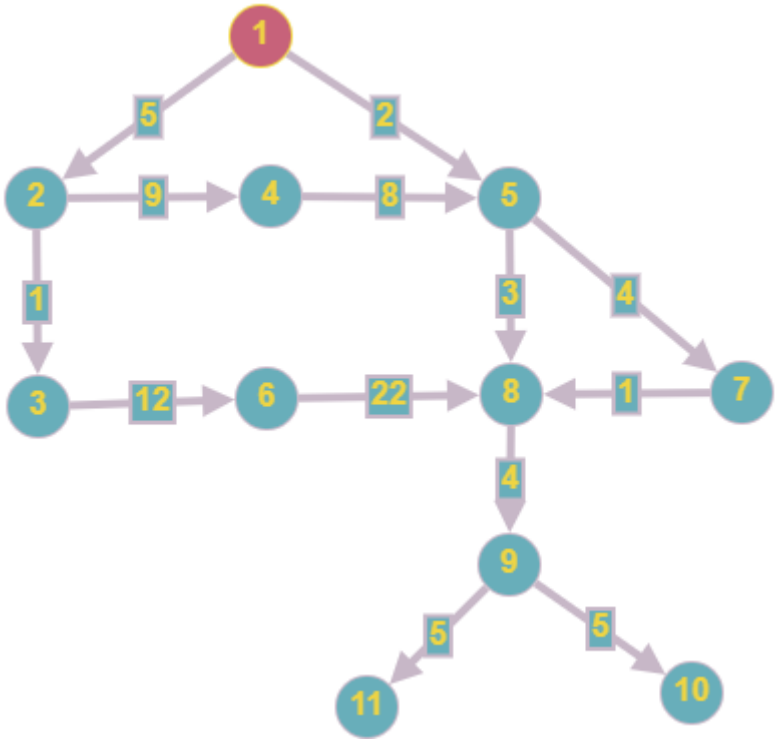
```
Iterator vertices = getAllVertices();
boolean startExists = false;
boolean finishExists = false;
while (vertices.hasNext()){
    Vertex v = (Vertex) vertices.next();
    v.info = infinity;
    if (v.id.equals(start)){
        startExists = true;
    }
    if (v.id.equals(finish)){
        finishExists = true;
    }
}
if (!startExists){
    throw new RuntimeException("Start point does not exist with id: " + start);
}
if (!finishExists){
    throw new RuntimeException("Finish point does not exist with id: " + finish);
}
```

Edasi leiame start id-le vastava Vertex objekti kasutades meetodit Graph.findVertexById(String id) ja loome mälu loendi tippude jaoks millega töötame (Kood 20). Graafi läbi töötamist alustame esimesest tipust milleks on start. Näide on toodud Joonis 5.

Kood 20. Start id põhjal Vertex objekti leidmine ja mälu loendi loomine

```
Vertex startVertex = getVertexById(start);
startVertex.info = 0;
List<Vertex> bufferList = new ArrayList<>();
bufferList.add(startVertex);
```

Joonis 5. Esimene tipp analüüsiks



Edasi alustatakse graafi läbimist laiuti mis annab teekonna 1->2->5->3->4->7->8->6->9->10->11. Teekond muutub vastavalt kui leitakse minimaalne langus ja liigutakse sealt edasi, juhul kui sihtkohani ei jõuta jätkatakse viimasest sobivast tipust. Iga läbitud tipu muutuja “visited” märgitakse vastavalt true, et jääks kirje sisse tipu kõlastamise kohta mis võimaldab hiljem tsüklis seda mitte kasutada. Teise etapi kontroll hõlmab tippe 2 ja 5 mis on esitatud Joonis 6. Kontrollimiseks loome muutuja number muutuja “minLen”, Vertex objekti “minChoice” ja mälu loendi põhjal iteraator objekti “bufferIterator”. Edasi leitakse uue while tsükliga “minChoice” tipp millega edasi teekonda jätkata. Juhul kui seda ei eksisteeri pole ka võimalik antud graafil teed leida (Kood 21). Tipp “minChoice” eemaldatakse tippude mälu loendist ja luuakse uus iterator objekt mis sisaldab antud tipu naabreid. Sellega on võimalik uurida kuhu oleks väimalik teha järgmine samm (Kood 22). Tipu naabrite töötlemisel kasutatakse käiakse läbi kõigi kaarte mis on algse tipuga seotud. Lisakontrollina, et hoida originaal graafi tipud vajaduseta muutumast on sisse viidud tõesus tüppi muutuja „allArcsChecked“ mis on vajalik puhuks kui mõni järgnev kaar on sobilikum siis muudame ainult vastava kaare väärtusi kui kõik on läbi uuritud. Selleks on kontoll mis jälgib kas on tulemas veel mõni kaar (Kood 23). Seame väärtuse muutujale „minLen“ mis tähistab hetkel uuritava kaare langust ja määrame järgmiseks punktiks hetkel uuritava naabri. Viime läbi kontrolli, et antud naaber ei oleks juba

külastatud (Kood 24). Juhul kui tipp on külastamat ja tema info ei sisalda teekonna miinimum langust oma „info“ muutujas sätestame selle vastavalt hetkel kehtivale „minLen“ väärtusele. Kontrollime kas „minLen“ on väiksem kui muutuja „arcCheckValue“, juhul kui võrdlus on tõene asendame mälus oleva miinimu langu jaoks vajaliku kaare (Kood 25). Juhul kui kõikide naabriteni viivad kaared on kontrollitud kirjutame üle sobilikuma kaare eelneva teekonna tipu, langu info ja sätestame, et tipp on külastatud (Kood 26). Viimase etapina antud tsükli lõpus kontrollitakse kas on täidetud kaks teekonna leidmise lõpetamiseks vajalikku tingimust:

1) Naaber tipu id võrdub finish väärtusega

2) Kõik naaber kaared on läbi uuritud

Juhul kui tingimused on täidetud luuakse uus vastus loend kõikide Arc objektidega mida teekonnal läbitakse (Kood 27). Juhul kui tingimused pole täidetud käikse graaf edasi läbi kuni antud tingimuste täitumiseni või juhul kui teed ei eksisteeri tõstatakse viga mis ütleb, et teed ei ole (Kood 28). Kuni on graafil tippe mida uurida käikse kõik vastavalt läbi (Joonis).

Kood 21. Vertex objekti “minChoice” leidmine

```
Iterator bufferIterator = bufferList.iterator();
while (bufferIterator.hasNext()){
    Vertex v = (Vertex) bufferIterator.next();
    if(v.info < minLen) {
        minChoice = v;
        minLen = v.info;
    }
}
if (minChoice == null) {
    break;
}
```

Kood 22. Tipu naabrite leidmine

```
bufferIterator = getNeighbours(minChoice);
int arcCheckValue = infinity;
Boolean allArcsChecked = false;
Vertex saveVertex = null;
Arc saveArc = null;
```

Kood 23. Järgneva naaber kaare olemasolu kontroll

```
Arc a = (Arc) bufferIterator.next();
if (!bufferIterator.hasNext()){
    allArcsChecked = true;
}
```

Kood 24. Järgmise tipu kontroll

```
minLen = a.declineValue;
Vertex to = a.target;
if (to.visited){
    continue;
}
```

Kood 25. Kaare miinimum väärtuse kontrollimine

```
if (minLen<arcCheckValue){
    arcCheckValue = minLen;
    saveVertex = to;
    saveArc = a;
}
```

Kood 26. Eelneva tipu määramine

```
if (allArcsChecked){
    saveVertex.info = minLen;
    saveVertex.previous = minChoice;
    saveVertex.previous.first = saveArc;
    saveVertex.visited = true;
}
```

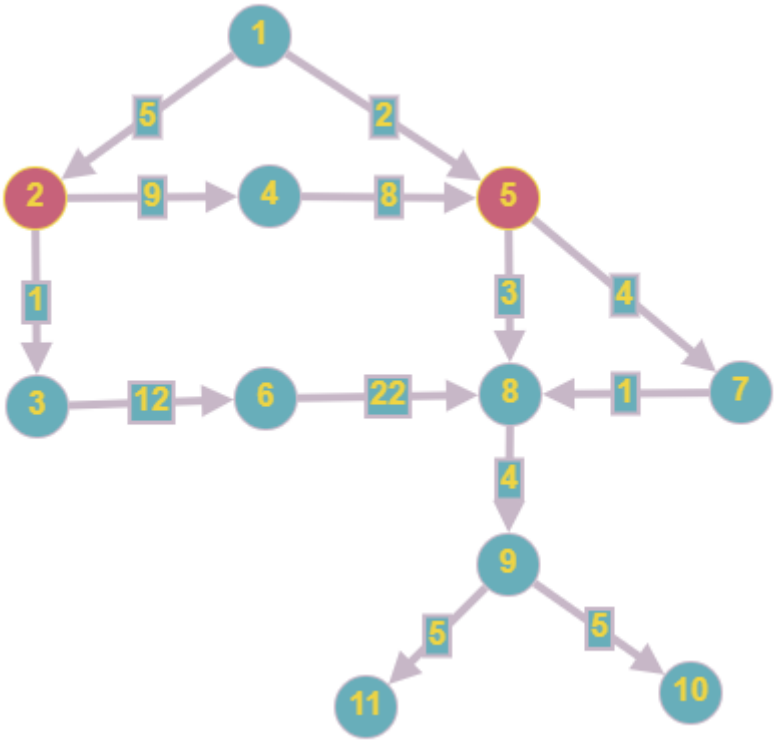
Kood 27. Teekonna lõpetamine ja vastuse genereerimine

```
if (saveVertex.id.equals(finish) && allArcsChecked){
    List<Arc> path = new ArrayList<>();
    while (saveVertex.previous != null){
        if (saveVertex.previous.first != null){
            if (path.contains(saveVertex.previous.first)){
                break;
            }
            path.add(saveVertex.previous.first);
        }
        saveVertex = saveVertex.previous;
    }
    List<Arc> pathReverse = new ArrayList<>();
    for (int i = path.size()-1; i >= 0; i--){
        pathReverse.add(path.get(i));
    }
    return pathReverse;
}
```

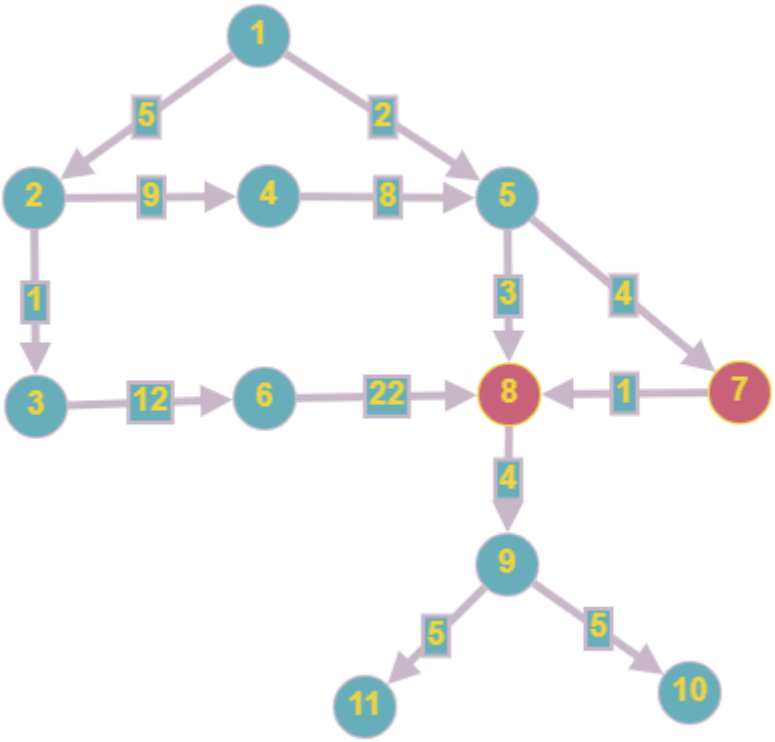
Kood 28. Vea heitmine juhul kui teed ei eksisteeri

```
throw new RuntimeException("Path not found between " + start + "->" + finish);
```

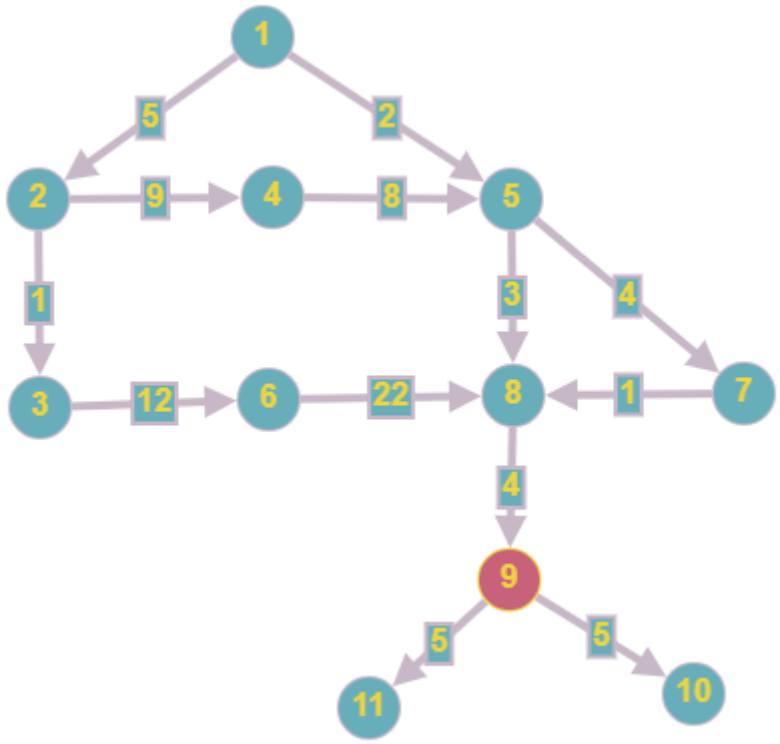
Joonis 6. Teise etapi punktid



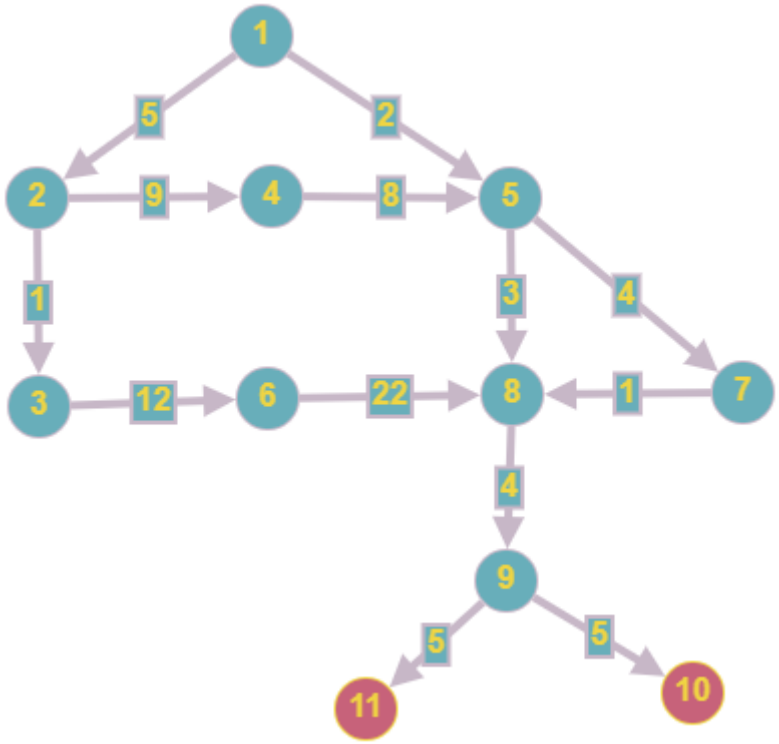
Joonis 7. Kolmanda etapi punktid



Joonis 8. Neljanda etapi punktid



Joonis 9. Viienda etapi punktid



4 Testid

4.1 Test 1

Loome graafi objekti, tipud ja kaared. Leiame ohutuima tee vastavalt Kood 29

Kood 29. Test graafi loomine

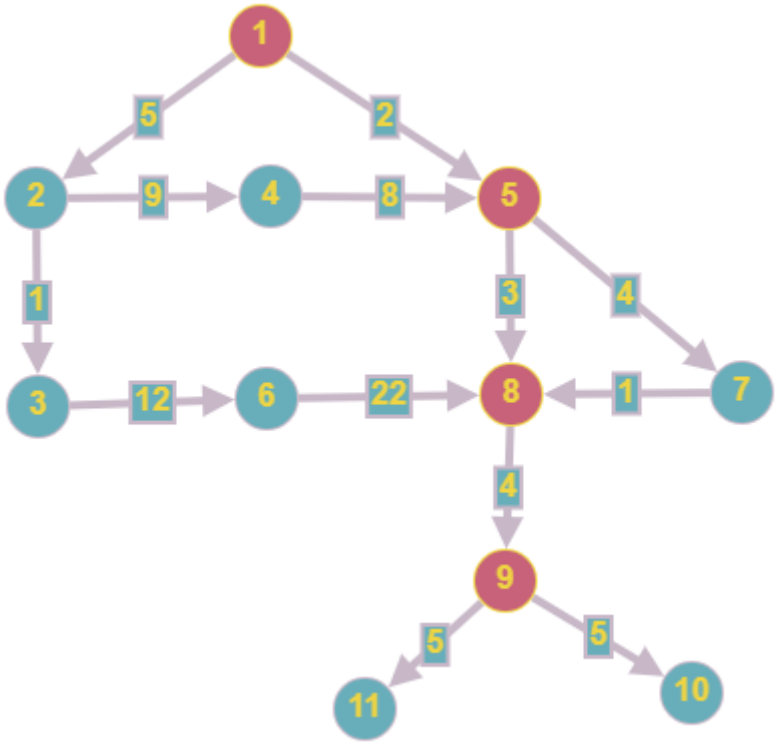
```
System.out.println("TEST 1");
System.out.println("Graph where Arc values are the decline from one point to another.");
Graph test = new Graph( s: "G");
Vertex v1 = test.createVertex( vid: "v1");
Vertex v2 = test.createVertex( vid: "v2");
Vertex v3 = test.createVertex( vid: "v3");
Vertex v4 = test.createVertex( vid: "v4");
Vertex v5 = test.createVertex( vid: "v5");
Vertex v6 = test.createVertex( vid: "v6");
Vertex v7 = test.createVertex( vid: "v7");
Vertex v8 = test.createVertex( vid: "v8");
Vertex v9 = test.createVertex( vid: "v9");
Vertex v10 = test.createVertex( vid: "v10");
Vertex v11 = test.createVertex( vid: "v11");

test.createArc( aid: "arc_v1_v5",  declineValue: 2, v1, v5);
test.createArc( aid: "arc_v1_v2",  declineValue: 5, v1, v2);
test.createArc( aid: "arc_v2_v3",  declineValue: 1, v2, v3);
test.createArc( aid: "arc_v2_v4",  declineValue: 9, v2, v4);
test.createArc( aid: "arc_v4_v5",  declineValue: 8, v4, v5);
test.createArc( aid: "arc_v5_v7",  declineValue: 4, v5, v7);
test.createArc( aid: "arc_v5_v8",  declineValue: 3, v5, v8);
test.createArc( aid: "arc_v3_v6",  declineValue: 12, v3, v6);
test.createArc( aid: "arc_v4_v6",  declineValue: 15, v4, v6);
test.createArc( aid: "arc_v7_v8",  declineValue: 1, v7, v8);
test.createArc( aid: "arc_v6_v8",  declineValue: 22, v6, v8);
test.createArc( aid: "arc_v8_v9",  declineValue: 4, v8, v9);
test.createArc( aid: "arc_v9_v11",  declineValue: 5, v9, v11);
test.createArc( aid: "arc_v9_v10",  declineValue: 5, v9, v10);
System.out.println(test);
List<Arc> safestPath = test.getSafestPathFromTo( start: "v1",  finish: "v9"); // V1->V5->V8->V9
System.out.println("Safest road on the graph were local decline from one point to next is minimal:");
System.out.println(safestPath);
```

Antud graaf on kujutad tekstipõhiselt Joonis 10 ja visuaalselt Joonis 11

Joonis 10. Test 1 graaf tekstiliselt

```
G
v11 -->
v10 -->
v9 --> arc_v9_v10 /5/ (v9->v10) arc_v9_v11 /5/ (v9->v11)
v8 --> arc_v8_v9 /4/ (v8->v9)
v7 --> arc_v7_v8 /1/ (v7->v8)
v6 --> arc_v6_v8 /22/ (v6->v8)
v5 --> arc_v5_v8 /3/ (v5->v8) arc_v5_v7 /4/ (v5->v7)
v4 --> arc_v4_v6 /15/ (v4->v6) arc_v4_v5 /8/ (v4->v5)
v3 --> arc_v3_v6 /12/ (v3->v6)
v2 --> arc_v2_v4 /9/ (v2->v4) arc_v2_v3 /1/ (v2->v3)
v1 --> arc_v1_v2 /5/ (v1->v2) arc_v1_v5 /2/ (v1->v5)
```

Joonis 11. Test 1 graaf visuaalselt koos ohutuima teega



Joonis 12. Test 1 vastuse väljatrükk

```
Safest road on the graph were local decline from one point to next is minimal:
[arc_v1_v5 /2/, arc_v5_v8 /3/, arc_v8_v9 /4/]
```

4.2 Test 2

Loome graafi objekti, tipud ja kaared. Leiame ohutuima tee vastavalt Kood 30

Kood 30. Test graafi loomine

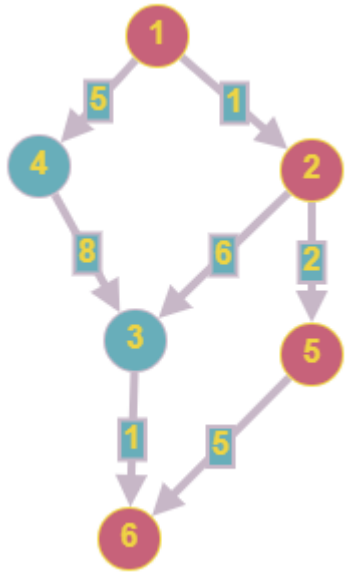
```
System.out.println("TEST 2");
System.out.println("Graph where Arc values are the decline from one point to another.");
Graph test = new Graph( s: "T");
Vertex v1 = test.createVertex( vid: "v1");
Vertex v2 = test.createVertex( vid: "v2");
Vertex v3 = test.createVertex( vid: "v3");
Vertex v4 = test.createVertex( vid: "v4");
Vertex v5 = test.createVertex( vid: "v5");
Vertex v6 = test.createVertex( vid: "v6");


test.createArc( aid: "arc_v1_v2",  declineValue: 1, v1, v2);
test.createArc( aid: "arc_v1_v4",  declineValue: 5, v1, v4);
test.createArc( aid: "arc_v2_v3",  declineValue: 6, v2, v3);
test.createArc( aid: "arc_v4_v3",  declineValue: 8, v4, v3);
test.createArc( aid: "arc_v2_v5",  declineValue: 2, v2, v5);
test.createArc( aid: "arc_v3_v6",  declineValue: 1, v3, v6);
test.createArc( aid: "arc_v5_v6",  declineValue: 5, v5, v6);
System.out.println(test);
List<Arc> safestPath = test.getSafestPathFromTo( start: "v1",  finish: "v6"); // V1->V2->V5->V6
System.out.println("Safest road on the graph were local decline from one point to next is minimal:");
System.out.println(safestPath);
```

Antud graaf on kujutad tekstipõhiselt Joonis 13 ja visuaalselt Joonis 14

Joonis 13. Test 2 graaf tekstiliselt

```
T
v6 -->
v5 --> arc_v5_v6 /5/ (v5->v6)
v4 --> arc_v4_v3 /8/ (v4->v3)
v3 --> arc_v3_v6 /1/ (v3->v6)
v2 --> arc_v2_v5 /2/ (v2->v5) arc_v2_v3 /6/ (v2->v3)
v1 --> arc_v1_v4 /5/ (v1->v4) arc_v1_v2 /1/ (v1->v2)
```

Joonis 14. Test 2 graaf visuaalselt koos ohutuima teega



Joonis 15. Test 2 vastuse väljatrükk

```
Safest road on the graph were local decline from one point to next is minimal:
[arc_v1_v2 /1/, arc_v2_v5 /2/, arc_v5_v6 /5/]
```

4.3 Test 3

Loome graafi objekti, tipud ja kaared. Leiame ohutuima tee vastavalt Kood 31

Kood 31. Test graafi loomine

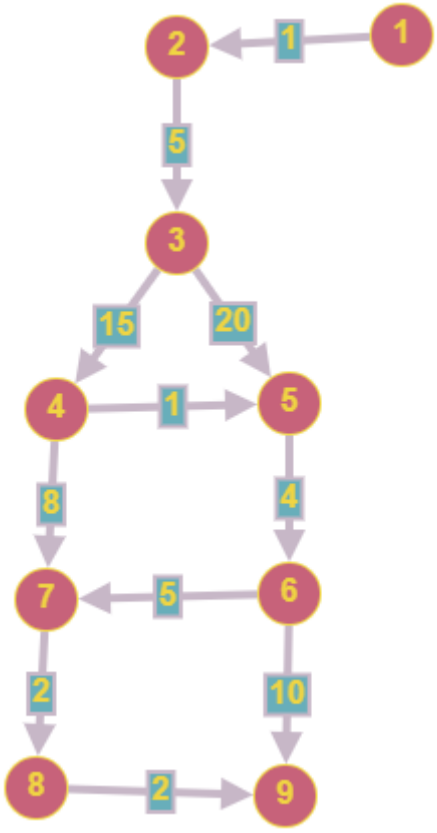
```
System.out.println("TEST 2");
System.out.println("Graph where Arc values are the decline from one point to another.");
Graph test = new Graph( s: "T");
Vertex v1 = test.createVertex( vid: "v1");
Vertex v2 = test.createVertex( vid: "v2");
Vertex v3 = test.createVertex( vid: "v3");
Vertex v4 = test.createVertex( vid: "v4");
Vertex v5 = test.createVertex( vid: "v5");
Vertex v6 = test.createVertex( vid: "v6");


test.createArc( aid: "arc_v1_v2",  declineValue: 1, v1, v2);
test.createArc( aid: "arc_v1_v4",  declineValue: 5, v1, v4);
test.createArc( aid: "arc_v2_v3",  declineValue: 6, v2, v3);
test.createArc( aid: "arc_v4_v3",  declineValue: 8, v4, v3);
test.createArc( aid: "arc_v2_v5",  declineValue: 2, v2, v5);
test.createArc( aid: "arc_v3_v6",  declineValue: 1, v3, v6);
test.createArc( aid: "arc_v5_v6",  declineValue: 5, v5, v6);
System.out.println(test);
List<Arc> safestPath = test.getSafestPathFromTo( start: "v1",  finish: "v6"); // V1->V2->V5->V6
System.out.println("Safest road on the graph were local decline from one point to next is minimal:");
System.out.println(safestPath);
```

Antud graaf on kujutad tekstipõhiselt Joonis 16 ja visuaalselt Joonis 17

Joonis 16. Test 3 graaf tekstiliselt

```
T
v6 -->
v5 --> arc_v5_v6 /5/ (v5->v6)
v4 --> arc_v4_v3 /8/ (v4->v3)
v3 --> arc_v3_v6 /1/ (v3->v6)
v2 --> arc_v2_v5 /2/ (v2->v5) arc_v2_v3 /6/ (v2->v3)
v1 --> arc_v1_v4 /5/ (v1->v4) arc_v1_v2 /1/ (v1->v2)
```

Joonis 17. Test 3 graaf visuaalselt koos ohutuima teega



Joonis 18. Test 3 vastuse väljatrükk

```
Safest road on the graph were local decline from one point to next is minimal:
[arc_v1_v2 /1/, arc_v2_v3 /5/, arc_v3_v4 /15/, arc_v4_v5 /1/, arc_v5_v6 /4/, arc_v6_v7 /5/, arc_v7_v8 /2/, arc_v8_v9 /2/]
```

4.4 Test 4

Kood 32. Test graafi loomine

Kood 32. Test graafi loomine

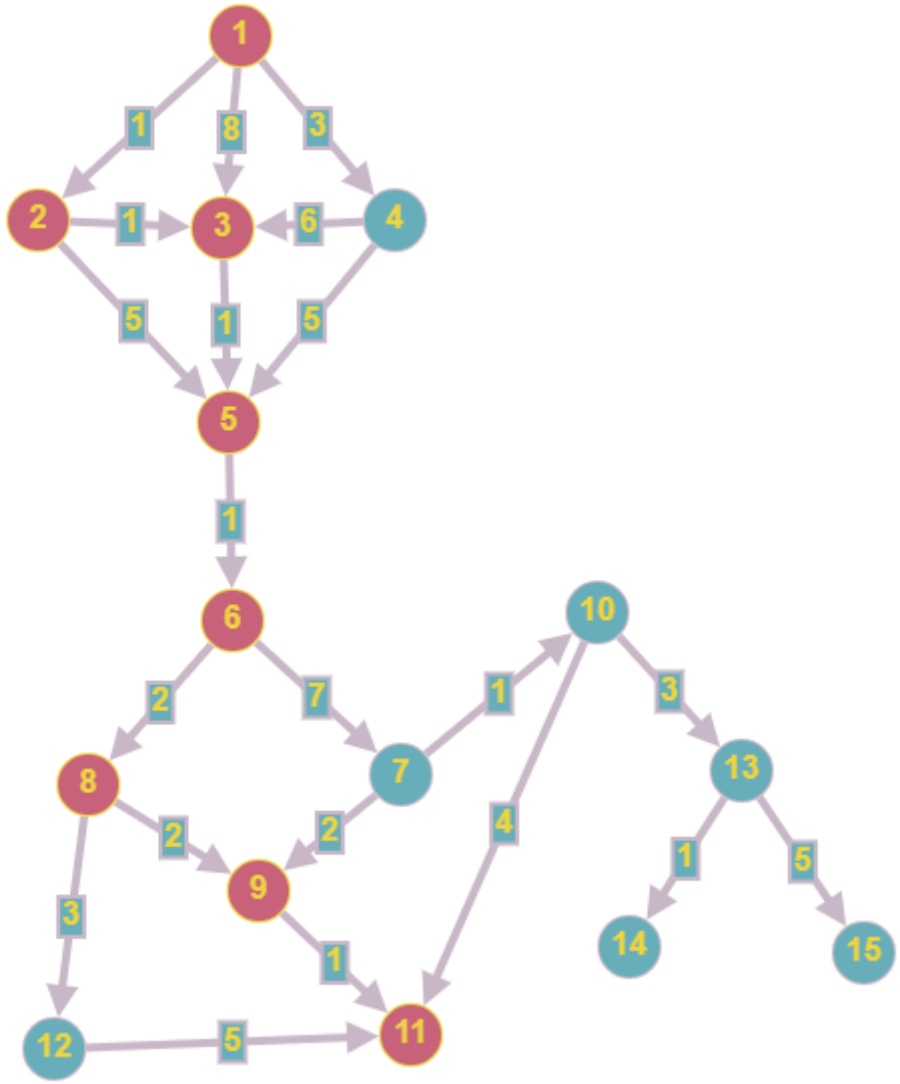
```
System.out.println("TEST 4");
System.out.println("Graph where Arc values are the decline from one point to another.");
Graph test = new Graph( s: "T");
Vertex v1 = test.createVertex( vid: "v1");
Vertex v2 = test.createVertex( vid: "v2");
Vertex v3 = test.createVertex( vid: "v3");
Vertex v4 = test.createVertex( vid: "v4");
Vertex v5 = test.createVertex( vid: "v5");
Vertex v6 = test.createVertex( vid: "v6");
Vertex v7 = test.createVertex( vid: "v7");
Vertex v8 = test.createVertex( vid: "v8");
Vertex v9 = test.createVertex( vid: "v9");
Vertex v10 = test.createVertex( vid: "v10");
Vertex v11 = test.createVertex( vid: "v11");
Vertex v12 = test.createVertex( vid: "v12");
Vertex v13 = test.createVertex( vid: "v13");
Vertex v14 = test.createVertex( vid: "v14");
Vertex v15 = test.createVertex( vid: "v15");

test.createArc( aid: "arc_v1_v2",  declineValue: 1, v1, v2);
test.createArc( aid: "arc_v1_v4",  declineValue: 3, v1, v4);
test.createArc( aid: "arc_v1_v3",  declineValue: 8, v1, v3);
test.createArc( aid: "arc_v2_v3",  declineValue: 1, v2, v3);
test.createArc( aid: "arc_v4_v3",  declineValue: 6, v4, v3);
test.createArc( aid: "arc_v2_v5",  declineValue: 5, v2, v2);
test.createArc( aid: "arc_v3_v5",  declineValue: 1, v3, v5);
test.createArc( aid: "arc_v4_v5",  declineValue: 5, v4, v2);
test.createArc( aid: "arc_v5_v6",  declineValue: 1, v5, v6);
test.createArc( aid: "arc_v6_v7",  declineValue: 7, v6, v7);
test.createArc( aid: "arc_v6_v8",  declineValue: 2, v6, v8);
test.createArc( aid: "arc_v8_v9",  declineValue: 2, v8, v9);
test.createArc( aid: "arc_v8_v12",  declineValue: 3, v8, v12);
test.createArc( aid: "arc_v12_v11",  declineValue: 5, v12, v11);
test.createArc( aid: "arc_v9_v11",  declineValue: 1, v9, v11);
test.createArc( aid: "arc_v7_v9",  declineValue: 2, v7, v9);
test.createArc( aid: "arc_v7_v10",  declineValue: 1, v7, v10);
test.createArc( aid: "arc_v10_v11",  declineValue: 4, v10, v11);
test.createArc( aid: "arc_v10_v13",  declineValue: 3, v10, v13);
test.createArc( aid: "arc_v13_v14",  declineValue: 1, v13, v14);
test.createArc( aid: "arc_v13_v15",  declineValue: 5, v13, v15);


System.out.println(test);
List<Arc> safestPath = test.getSafestPathFromTo( start: "v1",  finish: "v11"); // V1->V5->V8->V9
System.out.println("Safest road on the graph were local decline from one point to next is minimal:");
System.out.println(safestPath);
```

Antud graaf on kujutad tekstipõhiselt Joonis 19 ja visuaalselt Joonis 20

Joonis 19. Test 4 graaf tekstiliselt

```
T
v15 -->
v14 -->
v13 --> arc_v13_v15 /5/ (v13->v15) arc_v13_v14 /1/ (v13->v14)
v12 --> arc_v12_v11 /5/ (v12->v11)
v11 -->
v10 --> arc_v10_v13 /3/ (v10->v13) arc_v10_v11 /4/ (v10->v11)
v9 --> arc_v9_v11 /1/ (v9->v11)
v8 --> arc_v8_v12 /3/ (v8->v12) arc_v8_v9 /2/ (v8->v9)
v7 --> arc_v7_v10 /1/ (v7->v10) arc_v7_v9 /2/ (v7->v9)
v6 --> arc_v6_v8 /2/ (v6->v8) arc_v6_v7 /7/ (v6->v7)
v5 --> arc_v5_v6 /1/ (v5->v6)
v4 --> arc_v4_v5 /5/ (v4->v2) arc_v4_v3 /6/ (v4->v3)
v3 --> arc_v3_v5 /1/ (v3->v5)
v2 --> arc_v2_v5 /5/ (v2->v2) arc_v2_v3 /1/ (v2->v3)
v1 --> arc_v1_v3 /8/ (v1->v3) arc_v1_v4 /3/ (v1->v4) arc_v1_v2 /1/ (v1->v2)
```

Joonis 20. Test 4 graaf visuaalselt koos ohutuima teega



Joonis 21. Test 4 vastuse väljatrükk

```
Safest road on the graph were local decline from one point to next is minimal:
[arc_v1_v2 /1/, arc_v2_v3 /1/, arc_v3_v5 /1/, arc_v5_v6 /1/, arc_v6_v8 /2/, arc_v8_v9 /2/, arc_v9_v11 /1/]
```

4.5 Test 5

Viimane test on kasutades automaatset juhuslikkuse graafi, tippude ja kaarte loomise meetodit koos ohutuima tee leidmisega. Valitud tippude hulgaks on 2000 ja nende vaheliste kaarte hulgaks on 10000. Meetodi hindamiseks kasutatakse kulunud aega.

Kood 33. Automaatne juhuslikkuse graafi koostamine koos tippude ja nende vaheliste kaartega

```
int vertexCount = 2000;
int edgeCount = 10000;
//Because its graphical one direction vertex graph there is not al
while (true){
    try {
        Graph g = new Graph ( s: "G");
        g.createRandomSimpleGraph (vertexCount, edgeCount);
        String start = "v" + (Math.round(Math.random()*2000));
        String finish = "v" + (Math.round(Math.random()*2000));
        List<Arc> safestPath = g.getSafestPathFromTo(start, finish);
        System.out.println(safestPath);
        break;
    } catch (RuntimeException e){
        System.out.println(e.getMessage());
    }

}
```

Joonis 22. Test 5 kulunud aja väljatrükk

```
time (ms): 63
```

Kasutatud kirjandus

1. https://enos.itcollege.ee/~jpoial/algoritmid/graafid.html
2. http://www.cs.tlu.ee/~inga/AAS/graph_2020.pdf
3. https://en.wikipedia.org/wiki/Breadth-first_search
4. https://en.wikipedia.org/wiki/Dijkstra's_algorithm
5. https://enos.itcollege.ee/~jpoial/algorithms/examples/poeial_materials/src/FlowTask.java
6. https://www.yumpu.com/xx/document/read/14725903/algoritmid-ja-andmestruktuurid-ulesannete-kogu-tartu-ulikool

Lisa 1. Programmi kood

```java
import java.util.*;

/** Container class to different classes, that makes the whole
 * set of classes one class formally.
 */
// Used materials
// https://enos.itcollege.ee/~jpoial/algoritmid/graafid.html
// http://www.cs.tlu.ee/~inga/AAS/graph_2020.pdf
// https://en.wikipedia.org/wiki/Breadth-first_search
// https://en.wikipedia.org/wiki/Dijkstra's_algorithm
// https://enos.itcollege.ee/~jpoial/algorithms/examples/poeial_materials/src/FlowTask.java


/*
Ülesanne:

Olgu antud geograafiliste punktide graaf, mille iga kaarega (x,y) on
seotud väli .l – langus liikumisel punktist x punkti y. Ohutuimaks teeks kahe punkti vahel nimetame sellist teed, millel suurim lokaalne langus (punktist järgmisse
punkti) on minimaalne. Kirjutada graafi laiuti läbimisel põhinev algoritm, mille
käigus leitakse ohutuim tee antud punktist teise antud punkti.

 */
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() {
      long stime, ftime, diff;
      stime = System.nanoTime();
```

```
// Tests
// Test 1

System.out.println("TEST 1");
System.out.println("Graph where Arc values are the decline from one point to another.");
Graph test = new Graph( s: "G");
Vertex v1 = test.createVertex( vid: "v1");
Vertex v2 = test.createVertex( vid: "v2");
Vertex v3 = test.createVertex( vid: "v3");
Vertex v4 = test.createVertex( vid: "v4");
Vertex v5 = test.createVertex( vid: "v5");
Vertex v6 = test.createVertex( vid: "v6");
Vertex v7 = test.createVertex( vid: "v7");
Vertex v8 = test.createVertex( vid: "v8");
Vertex v9 = test.createVertex( vid: "v9");
Vertex v10 = test.createVertex( vid: "v10");
Vertex v11 = test.createVertex( vid: "v11");

test.createArc( aid: "arc_v1_v5",  declineValue: 2, v1, v5);
test.createArc( aid: "arc_v1_v2",  declineValue: 5, v1, v2);
test.createArc( aid: "arc_v2_v3",  declineValue: 1, v2, v3);
test.createArc( aid: "arc_v2_v4",  declineValue: 9, v2, v4);
test.createArc( aid: "arc_v4_v5",  declineValue: 8, v4, v5);
test.createArc( aid: "arc_v5_v7",  declineValue: 4, v5, v7);
test.createArc( aid: "arc_v5_v8",  declineValue: 3, v5, v8);
test.createArc( aid: "arc_v3_v6",  declineValue: 12, v3, v6);
test.createArc( aid: "arc_v4_v6",  declineValue: 15, v4, v6);
test.createArc( aid: "arc_v7_v8",  declineValue: 1, v7, v8);
test.createArc( aid: "arc_v6_v8",  declineValue: 22, v6, v8);
test.createArc( aid: "arc_v8_v9",  declineValue: 4, v8, v9);
test.createArc( aid: "arc_v9_v11",  declineValue: 5, v9, v11);
test.createArc( aid: "arc_v9_v10",  declineValue: 5, v9, v10);
System.out.println(test);
List<Arc> safestPath = test.getSafestPathFromTo( start: "v1",  finish: "v9"); // V1->V5->V8->V9
System.out.println("Safest road on the graph were local decline from one point to next is minimal:");
System.out.println(safestPath);
```

```
// Test 2

System.out.println("TEST 2");
System.out.println("Graph where Arc values are the decline from one point to another.");
Graph test = new Graph( s: "T");
Vertex v1 = test.createVertex( vid: "v1");
Vertex v2 = test.createVertex( vid: "v2");
Vertex v3 = test.createVertex( vid: "v3");
Vertex v4 = test.createVertex( vid: "v4");
Vertex v5 = test.createVertex( vid: "v5");
Vertex v6 = test.createVertex( vid: "v6");


test.createArc( aid: "arc_v1_v2",  declineValue: 1, v1, v2);
test.createArc( aid: "arc_v1_v4",  declineValue: 5, v1, v4);
test.createArc( aid: "arc_v2_v3",  declineValue: 6, v2, v3);
test.createArc( aid: "arc_v4_v3",  declineValue: 8, v4, v3);
test.createArc( aid: "arc_v2_v5",  declineValue: 2, v2, v5);
test.createArc( aid: "arc_v3_v6",  declineValue: 1, v3, v6);
test.createArc( aid: "arc_v5_v6",  declineValue: 5, v5, v6);
System.out.println(test);
List<Arc> safestPath = test.getSafestPathFromTo( start: "v1",  finish: "v6"); // V1->V2->V5->V6
System.out.println("Safest road on the graph were local decline from one point to next is minimal:");
System.out.println(safestPath);
```

```
// Test 3

System.out.println("TEST 3");
System.out.println("Graph where Arc values are the decline from one point to another.");
Graph test = new Graph( s: "T");
Vertex v1 = test.createVertex( vid: "v1");
Vertex v2 = test.createVertex( vid: "v2");
Vertex v3 = test.createVertex( vid: "v3");
Vertex v4 = test.createVertex( vid: "v4");
Vertex v5 = test.createVertex( vid: "v5");
Vertex v6 = test.createVertex( vid: "v6");
Vertex v7 = test.createVertex( vid: "v7");
Vertex v8 = test.createVertex( vid: "v8");
Vertex v9 = test.createVertex( vid: "v9");

test.createArc( aid: "arc_v1_v2",  declineValue: 1, v1, v2);
test.createArc( aid: "arc_v2_v3",  declineValue: 5, v2, v3);
test.createArc( aid: "arc_v3_v4",  declineValue: 15, v3, v4);
test.createArc( aid: "arc_v3_v5",  declineValue: 20, v3, v5);
test.createArc( aid: "arc_v4_v5",  declineValue: 1, v4, v5);
test.createArc( aid: "arc_v4_v7",  declineValue: 8, v4, v7);
test.createArc( aid: "arc_v5_v6",  declineValue: 4, v5, v6);
test.createArc( aid: "arc_v6_v7",  declineValue: 5, v6, v7);
test.createArc( aid: "arc_v7_v8",  declineValue: 2, v7, v8);
test.createArc( aid: "arc_v8_v9",  declineValue: 2, v8, v9);
test.createArc( aid: "arc_v6_v9",  declineValue: 10, v6, v9);
System.out.println(test);
List<Arc> safestPath = test.getSafestPathFromTo( start: "v1",  finish: "v9"); // V1->V5->V8->V9
System.out.println("Safest road on the graph were local decline from one point to next is minimal:");
System.out.println(safestPath);
```

```
// Test 4

System.out.println("TEST 4");
System.out.println("Graph where Arc values are the decline from one point to another.");
Graph test = new Graph( s: "T");
Vertex v1 = test.createVertex( vid: "v1");
Vertex v2 = test.createVertex( vid: "v2");
Vertex v3 = test.createVertex( vid: "v3");
Vertex v4 = test.createVertex( vid: "v4");
Vertex v5 = test.createVertex( vid: "v5");
Vertex v6 = test.createVertex( vid: "v6");
Vertex v7 = test.createVertex( vid: "v7");
Vertex v8 = test.createVertex( vid: "v8");
Vertex v9 = test.createVertex( vid: "v9");
Vertex v10 = test.createVertex( vid: "v10");
Vertex v11 = test.createVertex( vid: "v11");
Vertex v12 = test.createVertex( vid: "v12");
Vertex v13 = test.createVertex( vid: "v13");
Vertex v14 = test.createVertex( vid: "v14");
Vertex v15 = test.createVertex( vid: "v15");
```

```
test.createArc( aid: "arc_v1_v2",  declineValue: 1, v1, v2);
test.createArc( aid: "arc_v1_v4",  declineValue: 3, v1, v4);
test.createArc( aid: "arc_v1_v3",  declineValue: 8, v1, v3);
test.createArc( aid: "arc_v2_v3",  declineValue: 1, v2, v3);
test.createArc( aid: "arc_v4_v3",  declineValue: 6, v4, v3);
test.createArc( aid: "arc_v2_v5",  declineValue: 5, v2, v2);
test.createArc( aid: "arc_v3_v5",  declineValue: 1, v3, v5);
test.createArc( aid: "arc_v4_v5",  declineValue: 5, v4, v2);
test.createArc( aid: "arc_v5_v6",  declineValue: 1, v5, v6);
test.createArc( aid: "arc_v6_v7",  declineValue: 7, v6, v7);
test.createArc( aid: "arc_v6_v8",  declineValue: 2, v6, v8);
test.createArc( aid: "arc_v8_v9",  declineValue: 2, v8, v9);
test.createArc( aid: "arc_v8_v12",  declineValue: 3, v8, v12);
test.createArc( aid: "arc_v12_v11",  declineValue: 5, v12, v11);
test.createArc( aid: "arc_v9_v11",  declineValue: 1, v9, v11);
test.createArc( aid: "arc_v7_v9",  declineValue: 2, v7, v9);
test.createArc( aid: "arc_v7_v10",  declineValue: 1, v7, v10);
test.createArc( aid: "arc_v10_v11",  declineValue: 4, v10, v11);
test.createArc( aid: "arc_v10_v13",  declineValue: 3, v10, v13);
test.createArc( aid: "arc_v13_v14",  declineValue: 1, v13, v14);
test.createArc( aid: "arc_v13_v15",  declineValue: 5, v13, v15);
System.out.println(test);
List<Arc> safestPath = test.getSafestPathFromTo( start: "v1",  finish: "v11"); // V1->V5->V8->V9
System.out.println("Safest road on the graph were local decline from one point to next is minimal:");
System.out.println(safestPath);
```

```
// Test 5
// Test timing with big graph

System.out.println("TEST 5");
System.out.println("Graph where Arc values are the decline from one point to another.");
int vertexCount = 2000;
int edgeCount = 10000;
//Because its graphical one direction vertex graph there is not always path from one point to another we must try random graph multiple times for result.
while (true){
    try {
        Graph g = new Graph ( s: "G");
        g.createRandomSimpleGraph (vertexCount, edgeCount);
        String start = "v" + (Math.round(Math.random()*2000));
        String finish = "v" + (Math.round(Math.random()*2000));
        System.out.println("start: " + start);
        System.out.println("finish: " + finish);
        List<Arc> safestPath = g.getSafestPathFromTo(start, finish);
        System.out.println(safestPath);
        break;
    } catch (RuntimeException e){
        System.out.println(e.getMessage());
    }

}

ftime = System.nanoTime();
diff = ftime - stime;
System.out.println("time (ms): " + (diff / 1000000));
}
```

```java
/**
 * Class for vertex that in this work simulates a point in graph of geometric points.
 * Vertex contains information about id, connected arcs (last/next) and boolean value
 * of visitation.
 */
class Vertex {

    private String id;
    private Vertex next;
    private Arc first;
    private int info = 0;
    private Vertex previous = null;
    private Boolean visited = 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 String toString() { return id; }

    public Iterator getAllArcs(){
        List<Arc> result = new ArrayList<>();
        Arc a = first;
        while (a != null) {
            result.add(a);
            a = a.next;
        }
        return result.iterator();
    }
}
```

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

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

  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 + " /" + declineValue + "/"; }
}
```

```java
/**
 * Class Graph that contains all the methods for working with graph
 */
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); }

   @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();
    }

    /**
     * Method for generating new vertex
     *
     * @param vid String input for vertex id
     *
     * @return new Vertex object
     */
    public Vertex createVertex (String vid) {
        Vertex res = new Vertex (vid);
        res.next = first;
        first = res;
        return res;
    }

    /**
     * Method for generating new arc between two vertexes
     *
     * @param aid String input for vertex id
     * @param declineValue Integer value of decline between two vertexes
     * @param from Vertex input from the arc starts
     * @param to Vertex input where the arc connects
     *
     * @return new Arc
     */
    public Arc createArc (String aid, int declineValue, Vertex from, Vertex to) {
        Arc res = new Arc (aid);
        res.declineValue = declineValue;
        res.next = from.first;
        res.from = from;
        from.first = res;
        res.target = to;
        return res;
    }
```

```java
/**
 * 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];
   int declineValue;
   for (int i = 0; i < n; i++) {
      varray [i] = createVertex ( vid: "v" + (n - i));
      declineValue = (int) (Math.random()*100);
      if (i > 0) {
         int vnr = (int)(Math.random()*i);
         createArc ( aid: "a" + varray [vnr].toString() + "_"
            + varray [i].toString(), declineValue, varray [vnr], varray [i]);
         } 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.declineValue;
         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);

   int declineValue;

   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
      declineValue = (int) (Math.random()*100);
      Vertex vi = vert [i];
      Vertex vj = vert [j];
      createArc ( aid: "a" + vi.toString() + "_" + vj.toString(), declineValue, vi, vj);
      connected [i][j] = 1;
      edgeCount--;  // a new edge happily created
   }
```

```
    }

    /**
     * Find all vertexes on graph.
     *
     * @return Iterator of all vertexes on graph
     */
    public Iterator getAllVertices(){
        List<Vertex> result = new ArrayList<>();
        Vertex v = first;
        while (v != null) {
            result.add(v);
            v = v.next;
        }
        return result.iterator();
    }

    /**
     * Find all connected neighbours for vertex in interest.
     *
     * @param v vertex of interest
     *
     * @return Iterator of all vertex connected neighbours
     */
    public Iterator getNeighbours(Vertex v){
        List<Arc> result = new ArrayList<>();
        if (v.first != null){
            Arc a = v.first;
            result.add(a);
            while (a.next != null){
                result.add(a.next);
                a = a.next;
            }
        }
        return result.iterator();
    }
```

```java
/**
 * Method for finding safest path from start to end in graph.
 *
 * @param start String input of start vertex id
 * @param finish String input of finish vertex id
 *
 * @return List of arcs on safest path
 */
public List<Arc> getSafestPathFromTo(String start, String finish) {
    if (start.equals(finish)){
        throw new RuntimeException("Start point can not be same as finish: " + start);
    }
    int infinity = Integer.MAX_VALUE / 4;
    Iterator vertices = getAllVertices();
    boolean startExists = false;
    boolean finishExists = false;
    while (vertices.hasNext()){
        Vertex v = (Vertex) vertices.next();
        v.info = infinity;
        if (v.id.equals(start)){
            startExists = true;
        }
        if (v.id.equals(finish)){
            finishExists = true;
        }
    }
    if (!startExists){
        throw new RuntimeException("Start point does not exist with id: " + start);
    }
    if (!finishExists){
        throw new RuntimeException("Finish point does not exist with id: " + finish);
    }
    Vertex startVertex = getVertexById(start);
    startVertex.info = 0;
    List<Vertex> bufferList = new ArrayList<>();
    bufferList.add(startVertex);
```

```
while (bufferList.size()>0){
    int minLen = infinity+1;
    Vertex minChoice = null;
    Iterator bufferIterator = bufferList.iterator();
    while (bufferIterator.hasNext()){
        Vertex v = (Vertex) bufferIterator.next();
        if(v.info < minLen) {
            minChoice = v;
            minLen = v.info;
        }
    }
    if (minChoice == null) {
        break;
    }
    bufferList.remove(minChoice);
    bufferIterator = getNeighbours(minChoice);
    int arcCheckValue = infinity;
    Boolean allArcsChecked = false;
    Vertex saveVertex = null;
    Arc saveArc = null;
    while (bufferIterator.hasNext()){
        Arc a = (Arc) bufferIterator.next();
        if (!bufferIterator.hasNext()){
            allArcsChecked = true;
        }
        minLen = a.declineValue;
        Vertex to = a.target;
        if (to.visited){
            continue;
        }
        if (to.info == infinity){
            bufferList.add(to);
        }
        if (minLen<arcCheckValue){
            arcCheckValue = minLen;
            saveVertex = to;
            saveArc = a;
        }
```

```
                if (allArcsChecked){
                    saveVertex.info = minLen;
                    saveVertex.previous = minChoice;
                    saveVertex.previous.first = saveArc;
                    saveVertex.visited = true;
                }
                if (saveVertex.id.equals(finish) && allArcsChecked){
                    List<Arc> path = new ArrayList<>();
                    while (saveVertex.previous != null){
                        if (saveVertex.previous.first != null){
                            if (path.contains(saveVertex.previous.first)){
                                break;
                            }
                            path.add(saveVertex.previous.first);
                        }
                        saveVertex = saveVertex.previous;
                    }
                    List<Arc> pathReverse = new ArrayList<>();
                    for (int i = path.size()-1; i >= 0; i--){
                        pathReverse.add(path.get(i));
                    }
                    return pathReverse;
                }
            }
        }
        throw new RuntimeException("Path not found between " + start + "->" + finish);
    }

    /**
     * Method for geting Vertex from id string input
     *
     *  @param id String input for vertex id name
     *
     * @return Vertex with specific id
     */
    public Vertex getVertexById(String id) {
        Vertex v = first;
        while (!v.id.equals(id)){
            v = v.next;
        }
        return v;
    }
}

}
```