Tallinna Tehnikaülikool

Euleri teekonna leidmine ning graafi tippude nummerdamine teekonna olemasolul

Liisa Heinla
194147IADB

Tallinn 2020

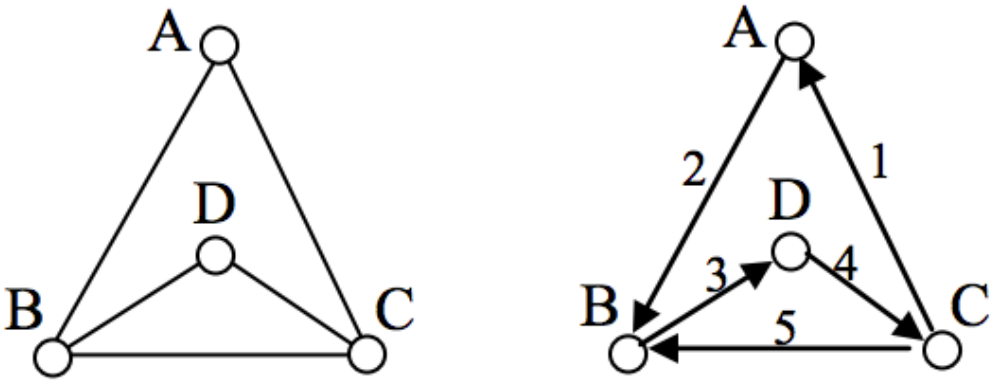
1. Probleemi püstitus

Ülesande püstitus on järgmine: koostada meetod, mis teeb kindlaks, kas etteantud sidusas lihtgraafis leidub Euleri tee ning selle leidumisel nummerdab kõik tipud vastavalt nende järjekorrale Euleri tee läbimisel kasutades Fleury' algoritmi.

Euleri tee graafil on tee, kus iga graafi kaar on läbitud ainult üks kord, moodustades tervikliku teekonna. Lihtsamalt öeldes on see sama olukord, kus tuleb ühendada graafi tippe kaarte kaudu, kasutades igat kaart ainult ühe korra. Fleury' algoritm aitab sellise tee leida. Algoritm näeb ette, et pärast iga kaare läbimist on kasulik see kaar graafilt eemaldada. Seda tehes pole võimalik enam seda kaart kasutada ning on kergem leida mittekasutatud kaari, mille kaudu järgmisesse tippu liikuda.

Euleri tee olemasolu saab kindlaks määrata iga graafi tipust väljuvate kaarte kaudu – kui graafil on tippe, millest väljub paaritu arv kaari, kaks või mitte ühtki, leidub graafil Euleri tee.

Seega oli tarvis läbida kaks sammu: kindlaks määrata, kas tegemist on Euleri teega ning seejärel leida Euleri tee läbimise kaared graafil ning need kirja panna (vt 1.1)



1.1 Euleri tee graafil

2. Protsess – algoritmid ja meetodid

Alustuseks tegin kokku 3 abimeetodit. Kõigepealt lõin meetodi nimega makeMap(), mis, kasutades Arrays.streami, loob mapi, kuhu on koondatud võtmetena tippude nimed. Tippudele pannakse väärtuseks tippudest väljuvate kaarte hulk. Kuna iga kaar on kirjeldatud mõlemas tipus, kus see alguse saab või lõpeb, on kaari iseloomustavaid Vertex objekte kaks korda rohkem. Selle tõttu jagan mapi pannes tippude arvu kahega (Lisa 1). Teine abimeetod, mille lõin ja kasutan, on getStringArray(), mis teeb ArrayListi stringidest String[]-kujule. See teeb koopia Arraylistist ning lisab selle String[]-listi. Kolmas abimeetod on countTips(), mis loeb

kokku nii Vertexid (stringi kujul) kui ka Arcid ning lisab need mõlemad Graph konstruktoris loodud listidesse. Selles meetodis kasutatakse while loope (Lisa 6).

Kaks põhimeetodit, hasEulerPath() ja enumerateEulerPath(), kasutavad suuremal või väiksemal määral eelmainitud abimeetodeid. Kuna selleks, et Euleri graafis oleks olemas Euleri teekond, on vaja lugeda igast tipust väljuvad kaared kokku. Euleri teekonnale on iseloomulik, et igast tipust väljub paarisarv kaari või on graafi peal kokku kaks tippu, millest väljub paaritu arv kaari. Seega kasutab hasEulerMap() meetodeid getStringArray() ning makeMap(), et lugeda kokku igast tipust väljuvate kaarte arv ning panna need mapi. Seejärel käiakse for loopiga map läbi, et lugeda kokku tippude arv, kust väljub paaritu arv kaari. hasEulerPath() väljastab booleani väärtuse, mis ütleb, kas mapis oli kaks või mitte ühtegi paarituarvulise kaarte arvuga tippe (Lisa 2).

Graafi loomiseks kasutan etteantud meetodeid createVertex() ja createArc(). Selleks, et graafi luua, peavad enne olema loodud tipud ehk Vertexid. Vertexisse on tarvis lisada tipu nimi. createArc kasutab loodud tippe ning ühendab need. createArc() meetod nõuab kaare nime, ning kaare algus- ja lõpp-punkti. Mõistlik on luua kaks erisuunalist kaart, et Euleri teekonna leidmisel oleks võimalus liikuda kaarel nii üht- kui teistpidi (Lisad 3 ja 4).

Põhiline funktsioon programmis on enumerateEulerPath(). Kõigepealt loeb see kõik kaared ja tipud kokku, kasutades countTips() meetodit.

```
public void countTips() {
    Vertex v = this.first;
    while (v != null) {
        ArrayList<Arc> x = new ArrayList<>();
        Arc a = v.first;
        while (a != null) {
            this.arcs.add(a);
            this.vertexes.add(v.toString());
            this.vertexes.add(a.target.toString());
            x.add(a);
            a = a.next;
        }
        this.mapOfArcs.put(v.toString(), x);
        v = v.next;
    }
}
```

Seejärel loob see Arraylisti Arc tüüpi objektidest, kuhu programm hakkab lisama Euleri teekonna tippe. Juhul kui graafil on Euleri teekond (mida kontrollitakse hasEulerPath() meetodiga), leitakse kõigepealt üles alguspunkt. Reegel on, et kui paaritu kaartearvuga tippe on null, siis võib stardipunkt olla ükskõik milline tipp. Kui neid on kaks, siis tuleb valida üks neist tippudest oma stardipunktiks. Ehk programm käib läbi mapi, kus on kirjas palju igast tipust väljuvaid kaari on ning võtab esimese, millel on paaritu arv. Kui paaritut arvu pole, võetakse esimene mapi võti alguspunktiks. Alguskaar lisatakse Arcide listi. Fleury algoritmi põhjal on võimalik leida Euleri teekonda nii, et iga kasutatud kaar eemaldatakse.

Programm hakkab eemaldama graafi konstruktori arcs (kõik kaared graafis) kaari. Esimesena leitakse arcs nimekirjast kõik kaared, mille alguspunkt on start.

```
if (arcs.isEmpty()) {
    throw new RuntimeException("Please create a graph");
}
ArrayList<Arc> path = new ArrayList<>();
if (hasEulerPath()) {
    String start = "";
    for (Map.Entry<String, ArrayList<Arc>> entry : mapOfArcs.entrySet()) {
        if (entry.getValue().size() % 2 == 1) {
            start = entry.getKey();
            break;
        }
    }
    if (start.equals("")) {
        start = mapOfArcs.keySet().iterator().next();
    }
```

2.1 Start punkti leidmine

Seejärel võetakse esimene leitud kaar ning kontrollitakse, kas selle kaare sihtpunkt katkestaks ühenduse ülejäänud graafiga (tippu suubub üks kaar ning ei välju mitte ühtki ehk tipu kaarte arv on üks). Kui katkestab, siis võetakse järgmine kaar ning kontrollitakse uuesti. Kui ei katkesta, siis eemaldatakse see kaar arcs listist.

Äsja eemaldatud kaare sihtpunkt määratakse uueks start punktiks. Eemaldatakse for loopiga arcsist ka teine kaar, millel sama asukoht kuid siht ja alguspunkt on vahetunud, et vältida kaare mitmekordset kasutust, mis pole lubatud. Kogu protsess toimub while loopi sees tingimusel, et arci listi suurus on suurem nullist ehk kõik valikus olevad kaared saaksid kasutatud (Lisa 5). While loopi lõppedes tagastatakse StirngBuilder, kus on märgitud Euleri teekond.

```
while (arcs.size() != 0) {
    ArrayList<Arc> x = new ArrayList<>();
    for (Arc element : arcs) {
        if (element.id.replaceAll( regex: "a",  replacement: "").split( regex: "_")[0].equals(start))
        x.add(element);
    }
    for (Arc item : x) {
        String[] y = item.id.replaceAll( regex: "a",  replacement: "").split( regex: "_");
        if (this.makeMap(getStringArray(vertexes)).get(y[1]) > 1) {
            arcs.remove(item);
            path.add(item);
            for (Arc a : copy) {
                if (a.target.id.equals(y[0]) && a.id.replaceAll( regex: "a",  replacement: "").split( regex: "_")[0].equals(y[1])) {
                    arcs.remove(a);
                    break;
                }
            }
            start = item.target.id;
            break;
        }
    }
}
else {
    return ("The graph has no euler path, therefore the verteces cannot be enumerated!");
}
return path;
```

Erindid meetodis hasEulerPath() – visatakse RunTimeException, kui graafil puuduvad tipud ja kaared.

```
int uneven = 0;
Map<String, Integer> countedTips = this.makeMap(this.getStringArray(this.vertexes));
if (countedTips.size() == 0) {
    throw new RuntimeException("Please create a valid graph with valid verteces");
}
```

Erindid meetodis enumerateEulerPath() – visatakse RunTimeException, kui graafil puuduvad tipud ja kaared.

```
if (arcs.isEmpty()) {
    throw new RuntimeException("Please create a graph");
}
```

3. Kasutusjuhend

Selleks, et programmi käivitada, on vaja luua uus graafi objekt Graph klassi meetodis run():

```
class Graph {
    private String id;
    private Vertex first;
    private int info = 0;
    private Map<String, ArrayList<Arc>> mapOfArcs = new HashMap<>();
    private ArrayList<String> vertexes = new ArrayList<>();
    private ArrayList<Arc> arcs = new ArrayList<>();

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

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

3.1 Graph klass

```
Graph a = new Graph ( s: "G1");
```

Seejärel tuleb lisada graafile tipud ning panna neile nimed kasutades createVertex meetodit:

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

3.1 createVertex meetod

NB! Tipu nimi tuleb esitada vormis „v“ + tipu number

```
Vertex v1 = a.createVertex( vid: "v1");
Vertex v2 = a.createVertex( vid: "v2");
Vertex v3 = a.createVertex( vid: "v3");
Vertex v4 = a.createVertex( vid: "v4");
```

Tipud tuleb omavahel ära ühendada kasutades createArc meetodit:

```
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;
}
```

3.2 createArc meetod

NB! Kaare nimi tuleb esitada vormis „a“ + algustipp + „_“ + lõpp-tipp

Meetodi väljad: (kaare nimi, alguspunkt, lõpp-punkt)

```
a.createArc( aid: "av1_v2", v1, v2);
a.createArc( aid: "av2_v1", v2, v1);
a.createArc( aid: "av2_v3", v2, v3);
a.createArc( aid: "av3_v2", v3, v2);
a.createArc( aid: "av3_v1", v3, v1);
a.createArc( aid: "av1_v3", v1, v3);
a.createArc( aid: "av3_v4", v3, v4);
a.createArc( aid: "av4_v3", v4, v3);
a.createArc( aid: "av4_v2", v4, v2);
a.createArc( aid: "av2_v4", v2, v4);
```

NB! Selleks, et programm toimiks, on vaja lisada igast kaarest kaks eksemplari, kus sihtpunkt ja alguspunkt on vahetunud.

Et välja kutsuda hasEulerPath() ja enumerateEulerPath(), on tarvis välja kutsuda countTips() meetod, mis salvestab tulemused:

```
a.countTips();
```

countTips tuleb välja kutsuda pärast tippude ja kaarte lisamist, kuid enne hasEulerPath() ja enumerateEulerPath() meetodite väljakutsumist.

Kui tahetakse testida, kas graafis on Euleri teekond, siis on tuleb välja kutsuda meetod hasEulerPath(), mis tagastab booleani:

```
System.out.println(a.hasEulerPath());
```

Selleks, et enumerateEulerPathi() välja kutsuda, on tarvis teha nii:

```
System.out.println(a.enumerateEulerPath());
```

Programm enumerateEulerPath() tagastab Euleri teekonn Arcide listina.

```
[av6_v5, av5_v1, av1_v6, av6_v3, av3_v4, av4_v5, av5_v3, av3_v2, av2_v5]
```

Iga graaf annab väljaprinditud kuju selliselt (mitte seotud eelmise graafiga G1):

```
G3
v1 --> av1_v3 (v1->v3) av1_v2 (v1->v2) av1_v5 (v1->v5)
v2 --> av2_v1 (v2->v1) av2_v3 (v2->v3)
v3 --> av3_v1 (v3->v1) av3_v6 (v3->v6) av3_v2 (v3->v2) av3_v5 (v3->v5)
v4 --> av4_v6 (v4->v6) av4_v5 (v4->v5)
v5 --> av5_v1 (v5->v1) av5_v3 (v5->v3) av5_v4 (v5->v4) av5_v6 (v5->v6)
v6 --> av6_v4 (v6->v4) av6_v3 (v6->v3) av6_v5 (v6->v5)
```

Terve programm on leitav Lisa 7 alt. Programmis sisalduvad meetodid createRandomTree(), createAdjMatrix() ja createRandomSimpleGraph on loodud graafi testimise eesmärgil ning rakendust tavakasutaja poolt ei leia.

4. Testimiskava:

Test 1 – nelja tipuga graaf

```
public void run() {
    Graph a = new Graph ( s: "G3");

    Vertex v1 = a.createVertex( vid: "v1");
    Vertex v2 = a.createVertex( vid: "v2");
    Vertex v3 = a.createVertex( vid: "v3");
    Vertex v4 = a.createVertex( vid: "v4");

    a.createArc( aid: "av1_v2", v1, v2);
    a.createArc( aid: "av2_v1", v2, v1);
    a.createArc( aid: "av2_v3", v2, v3);
    a.createArc( aid: "av3_v2", v3, v2);
    a.createArc( aid: "av3_v1", v3, v1);
    a.createArc( aid: "av1_v3", v1, v3);
    a.createArc( aid: "av3_v4", v3, v4);
    a.createArc( aid: "av4_v3", v4, v3);
    a.createArc( aid: "av4_v2", v4, v2);
    a.createArc( aid: "av2_v4", v2, v4);

    a.countTips();
    System.out.println(a);
    System.out.println(a.hasEulerPath());
    System.out.println(a.enumerateEulerPath());
```

Tulemus peab tulema: av2_v4, av4_v3, av3_v1, av1_v2, av2_v3

Väljund:

```
G3
v4 --> av4_v2 (v4->v2) av4_v3 (v4->v3)
v3 --> av3_v4 (v3->v4) av3_v1 (v3->v1) av3_v2 (v3->v2)
v2 --> av2_v4 (v2->v4) av2_v3 (v2->v3) av2_v1 (v2->v1)
v1 --> av1_v3 (v1->v3) av1_v2 (v1->v2)

true
[av2_v4, av4_v3, av3_v1, av1_v2, av2_v3]
```

Selgitus: teekonda alustatakse esimesest paaritust tipust v2 ning liigutakse sealt edasi esimese tipuni v4 jne kuni saadakse kõik kaared kasutatud

Test 2 – kuue tipuga graaf

```
public void run() {
    Graph a = new Graph ( s: "G3");
    Vertex v1 = a.createVertex( vid: "v1");
    Vertex v2 = a.createVertex( vid: "v2");
    Vertex v3 = a.createVertex( vid: "v3");
    Vertex v4 = a.createVertex( vid: "v4");
    Vertex v5 = a.createVertex( vid: "v5");
    Vertex v6 = a.createVertex( vid: "v6");
    a.createArc( aid: "av1_v6", v1, v6);
    a.createArc( aid: "av1_v5", v1, v5);
    a.createArc( aid: "av2_v3", v2, v3);
    a.createArc( aid: "av2_v5", v2, v5);
    a.createArc( aid: "av3_v2", v3, v2);
    a.createArc( aid: "av3_v5", v3, v5);
    a.createArc( aid: "av3_v4", v3, v4);
    a.createArc( aid: "av3_v6", v3, v6);
    a.createArc( aid: "av4_v3", v4, v3);
    a.createArc( aid: "av4_v5", v4, v5);
    a.createArc( aid: "av5_v2", v5, v2);
    a.createArc( aid: "av5_v3", v5, v3);
    a.createArc( aid: "av5_v4", v5, v4);
    a.createArc( aid: "av5_v6", v5, v6);
    a.createArc( aid: "av5_v1", v5, v1);
    a.createArc( aid: "av6_v3", v6, v3);
    a.createArc( aid: "av6_v1", v6, v1);
    a.createArc( aid: "av6_v5", v6, v5);

    a.countTips();
    System.out.println(a);
    System.out.println(a.hasEulerPath());
    System.out.println(a.enumerateEulerPath());
```

Väljundiks peab tulema [av6_v5, av5_v1, av1_v6, av6_v3, av3_v4, av4_v5, av5_v3, av3_v2, av2_v5]

Väljund:

```
G3
v6 --> av6_v5 (v6->v5) av6_v1 (v6->v1) av6_v3 (v6->v3)
v5 --> av5_v1 (v5->v1) av5_v6 (v5->v6) av5_v4 (v5->v4) av5_v3 (v5->v3) av5_v2 (v5->v2)
v4 --> av4_v5 (v4->v5) av4_v3 (v4->v3)
v3 --> av3_v6 (v3->v6) av3_v4 (v3->v4) av3_v5 (v3->v5) av3_v2 (v3->v2)
v2 --> av2_v5 (v2->v5) av2_v3 (v2->v3)
v1 --> av1_v5 (v1->v5) av1_v6 (v1->v6)

true
[av6_v5, av5_v1, av1_v6, av6_v3, av3_v4, av4_v5, av5_v3, av3_v2, av2_v5]
```

Selgitus: teekonda alustatakse esimesest paaritust tipust v6 ning liigutakse sealt edasi esimese tipuni v5 jne kuni saadakse kõik kaared kasutatud

Test 3 – puudub Euleri tee

```
public void run() {
    Graph a = new Graph ( s: "G3");
    Vertex v1 = a.createVertex( vid: "v1");
    Vertex v2 = a.createVertex( vid: "v2");
    Vertex v3 = a.createVertex( vid: "v3");
    Vertex v4 = a.createVertex( vid: "v4");
    Vertex v5 = a.createVertex( vid: "v5");
    Vertex v6 = a.createVertex( vid: "v6");
    a.createArc( aid: "av1_v5", v1, v5);
    a.createArc( aid: "av2_v3", v2, v3);
    a.createArc( aid: "av2_v5", v2, v5);
    a.createArc( aid: "av3_v2", v3, v2);
    a.createArc( aid: "av3_v5", v3, v5);
    a.createArc( aid: "av3_v4", v3, v4);
    a.createArc( aid: "av4_v3", v4, v3);
    a.createArc( aid: "av4_v5", v4, v5);
    a.createArc( aid: "av5_v2", v5, v2);
    a.createArc( aid: "av5_v3", v5, v3);
    a.createArc( aid: "av5_v4", v5, v4);
    a.createArc( aid: "av5_v6", v5, v6);
    a.createArc( aid: "av5_v1", v5, v1);
    a.createArc( aid: "av6_v5", v6, v5);

    a.countTips();
    System.out.println(a);
    System.out.println(a.hasEulerPath());
    System.out.println(a.enumerateEulerPath());
```

Väljundiks peab tulema tekst: „The graph has no euler path, therefore the verteces cannot be enumerated!“

Väljund:

```
G3
v6 --> av6_v5 (v6->v5)
v5 --> av5_v1 (v5->v1) av5_v6 (v5->v6) av5_v4 (v5->v4) av5_v3 (v5->v3) av5_v2 (v5->v2)
v4 --> av4_v5 (v4->v5) av4_v3 (v4->v3)
v3 --> av3_v4 (v3->v4) av3_v5 (v3->v5) av3_v2 (v3->v2)
v2 --> av2_v5 (v2->v5) av2_v3 (v2->v3)
v1 --> av1_v5 (v1->v5)

false
The graph has no euler path, therefore the verteces cannot be enumerated!
```

Selgitus: Paarituid tippe on graafil neli, seega Euleri teed ei eksisteeri.

Test 4 – tühi graaf ja enumerateEulerPath() meetod

```
Graph a = new Graph ( s: "G3");
System.out.println(a);
System.out.println(a.enumerateEulerPath());
```

Oodatav tulemus: RunTimeException

Tulemus:

```
G3

Exception in thread "main" java.lang.RuntimeException: Please create a graph
    at GraphTask$Graph.enumerateEulerPath(GraphTask.java:325)
    at GraphTask.run(GraphTask.java:26)
    at GraphTask.main(GraphTask.java:19)
```

Selgitus: tühjal graafi objektil (kaarte ja tippudeta) ei ole Euleri teed võimalik nummerdada

Test 5 . tühi graaf ja hasEulerPath() meetod

```
Graph a = new Graph ( s: "G3");
System.out.println(a);
System.out.println(a.hasEulerPath());
```

Oodatav tulemus: RunTimeException

Tulemus:

```
G3

Exception in thread "main" java.lang.RuntimeException: Please create a valid graph with valid verteces
    at GraphTask$Graph.hasEulerPath(GraphTask.java:303)
    at GraphTask.run(GraphTask.java:26)
    at GraphTask.main(GraphTask.java:19)
```

Selgitus: Euleri tee ei eksisteeri, kui graafil puuduvad tipud ja kaared.

Lisad

Lisa 1. makeMap() meetod

```
public Map<String, Integer> makeMap(String[] array) {
    Map<String, Integer> countedTips = new HashMap<>();
    Arrays.stream(array)
            .collect(Collectors.groupingBy(s -> s))
            .forEach((k, v) -> countedTips.put(k,  v: v.size()/2));
    return countedTips;
}
```

Lisa 2. hasEulerPath() meetod

```
public boolean hasEulerPath() {
    /**
     * checks if graph has euler path
     */
    int uneven = 0;
    Map<String, Integer> countedTips = this.makeMap(this.getStringArray(this.vertexes));
    if (countedTips.size() == 0) {
        throw new RuntimeException("Please create a valid graph with valid verteces");
    }
    for (Map.Entry<String, Integer> entry : countedTips.entrySet()) {
        if (entry.getValue() % 2 == 1) {
            uneven += 1;
        }
    }
    return uneven == 2 || uneven == 0;
}
```

Lisa 3. createVertex() meetod

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

Lisa 4. createArc() meetod

```
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;
}
```

Lisa 5. enumerateEulerPath() meetod

```
public Serializable enumerateEulerPath() {
    /**
     * enumerates Euler's path and returns a string of the result
     */
    if (arcs.isEmpty()) {
        throw new RuntimeException("Please create a graph");
    }
    StringBuilder sb = new StringBuilder();
    if (hasEulerPath()) {
        String start = "";
        for (Map.Entry<String, ArrayList<Arc>> entry : mapOfArcs.entrySet()) {
            if (entry.getValue().size() % 2 == 1) {
                start = entry.getKey();
                sb.append(start);
                sb.append("->");
                break;
            }
        }
        if (start.equals("")) {
            start = mapOfArcs.keySet().iterator().next();
        }
        ArrayList<Arc> copy = new ArrayList<>(arcs);
```

```
        while (arcs.size() != 0) {
            ArrayList<Arc> x = new ArrayList<>();
            for (Arc element : arcs) {
                if (element.id.replaceAll( regex: "a",  replacement: "").split( regex: "_")[0].equals(start))
                x.add(element);
            }
            for (Arc item : x) {
                String[] y = item.id.replaceAll( regex: "a",  replacement: "").split( regex: "_");
                if (this.makeMap(getStringArray(vertexes)).get(y[1]) > 1) {
                    arcs.remove(item);
                    sb.append(y[1]);
                    for (Arc a : copy) {
                        if (a.target.id.equals(y[0]) && a.id.replaceAll( regex: "a",  replacement: "").split( regex: "_")[0].equals(y[1])) {
                            arcs.remove(a);
                            break;
                        }
                    }
                    start = item.target.id;
                    if (arcs.size() > 0) {
                        sb.append("->");
                    }
                    break;
                }
            }
        }
    }
    else {
        return ("The graph has no euler path, therefore the verteces cannot be enumerated!");
    }
    return sb;
}
```

Lisa 6. Graafi objekti konstruktor

```
class Graph {
    private String id;
    private Vertex first;
    private int info = 0;
    private Map<String, ArrayList<Arc>> mapOfArcs = new HashMap<>();
    private ArrayList<String> vertexes = new ArrayList<>();
    private ArrayList<Arc> arcs = new ArrayList<>();
```

Lisa 7. Terve programm:

```
import java.io.Serializable;
import java.util.*;
import java.util.stream.Collectors;

/** Container class to different classes, that makes the whole
 * set of classes one class formally.
 */
public class GraphTask {

   /**
    * Koostada meetod, mis teeb kindlaks,
    * kas etteantud sidusas lihtgraafis leidub Euleri tee,
    * ning selle leidumisel nummerdab kõik tipud vastavalt nende järjekorrale
Euleri tsükli
    * läbimisel (vt. ka Fleury' algoritm)
    */
   public static void main (String[] args) {
      GraphTask a = new GraphTask();
      a.run();
   }

   /** Actual main method to run examples and everything. */
   public void run() {
      Graph a = new Graph ("G3");
      a.createRandomSimpleGraph(6, 9);

      a.countTips();
      System.out.println(a);
      System.out.println(a.hasEulerPath());
      System.out.println(a.enumerateEulerPath());

      // TODO!!! Your experiments here
   }

   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;
      }

      // TODO!!! Your Vertex methods here!
   }

   /** 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;
   // You can add more fields, if needed

   Arc (String s, Vertex v, Arc a) {
      id = s;
      target = v;
      next = a;
   }

   Arc (String s) {
      this (s, null, null);
   }

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

   // TODO!!! Your Arc methods here!
}

class Graph {
   private String id;
   private Vertex first;
   private int info = 0;
   private Map<String, ArrayList<Arc>> mapOfArcs = new HashMap<>();
   private ArrayList<String> vertexes = new ArrayList<>();
   private ArrayList<Arc> arcs = new ArrayList<>();

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

   Graph(String s) {
      this(s, null);
   }

   @Override
   public String toString() {
      String nl = System.getProperty("line.separator");
      StringBuffer sb = new StringBuffer(nl);
      sb.append(id);
      sb.append(nl);
      Vertex v = first;
      while (v != null) {
         sb.append(v.toString());
         sb.append(" -->");
         Arc a = v.first;
         while (a != null) {
            sb.append(" ");
            sb.append(a.toString());
            sb.append(" (");
            sb.append(v.toString());
            sb.append("->");
            sb.append(a.target.toString());
            sb.append(")");
            a = a.next;
         }
         sb.append(nl);
         v = v.next;
```

```
   }
   return sb.toString();
}

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

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], varray[vnr]);
      } else {
      }
   }
}

/**
 * Create an adjacency matrix of this graph.
 * Side effect: corrupts info fields in the graph
 *
 * @return adjacency matrix
 */
public int[][] createAdjMatrix() {
   info = 0;
   Vertex v = first;
   while (v != null) {
      v.info = info++;
      v = v.next;
   }
   int[][] res = new int[info][info];
   v = first;
   while (v != null) {
      int i = v.info;
      Arc a = v.first;
      while (a != null) {
         int j = a.target.info;
         res[i][j]++;
         a = a.next;
      }
      v = v.next;
```

```
   }
   return res;
}

/**
 * Create a connected simple (undirected, no loops, no multiple
 * arcs) random graph with n vertices and m edges.
 *
 * @param n number of vertices
 * @param m number of edges
 */
public void createRandomSimpleGraph(int n, int m) {
   if (n <= 0)
      return;
   if (n > 2500)
      throw new IllegalArgumentException("Too many vertices: " + n);
   if (m < n - 1 || m > n * (n - 1) / 2)
      throw new IllegalArgumentException
              ("Impossible number of edges: " + m);
   first = null;
   createRandomTree(n);// n-1 edges created here
   Vertex[] vert = new Vertex[n];
   Vertex v = first;
   int c = 0;
   while (v != null) {
      vert[c++] = v;
      v = v.next;
   }
   int[][] connected = createAdjMatrix();
   int edgeCount = m - n + 1;  // remaining edges
   while (edgeCount > 0) {
      int i = (int) (Math.random() * n);  // random source
      int j = (int) (Math.random() * n);  // random target
      if (i == j)
         continue;  // no loops
      if (connected[i][j] != 0 || connected[j][i] != 0)
         continue;  // no multiple edges
      Vertex vi = vert[i];
      Vertex vj = vert[j];
      createArc("a" + vi.toString() + "_" + vj.toString(), vi, vj);
      connected[i][j] = 1;
      createArc("a" + vj.toString() + "_" + vi.toString(), vj, vi);
      connected[j][i] = 1;
      edgeCount--;  // a new edge happily created
   }
}


/**
 * counts the arcs and vertexes and puts them to a list
 */
public void countTips() {
   Vertex v = this.first;
   while (v != null) {
      ArrayList<Arc> x = new ArrayList<>();
      Arc a = v.first;
      while (a != null) {
         this.arcs.add(a);
         this.vertexes.add(v.toString());
         this.vertexes.add(a.target.toString());
         x.add(a);
         a = a.next;
      }
      this.mapOfArcs.put(v.toString(), x);
      v = v.next;
   }
```

```
	}

	/**
	 * creates string array from Arraylist
	 */
	public String[] getStringArray(ArrayList<String> arr)

	{
	   Object[] objArr = arr.toArray();

	   String[] str = Arrays
	           .copyOf(objArr, objArr
	                           .length,
	                   String[].class);
	   return str;
	}


	/**
	 * checks if graph has euler path
	 */
	public boolean hasEulerPath() {
	   int uneven = 0;
	   Map<String, Integer> countedTips =
this.makeMap(this.getStringArray(this.vertexes));
	   if (countedTips.size() == 0) {
	      throw new RuntimeException("Please create a valid graph with valid
verteces");
	   }
	   for (Map.Entry<String, Integer> entry : countedTips.entrySet()) {
	      if (entry.getValue() % 2 == 1) {
	         uneven += 1;
	      }
	   }
	   return uneven == 2 || uneven == 0;
	}

	/**
	 * creates a map with the vertex as key and the count of its arcs as value
	 */
	public Map<String, Integer> makeMap(String[] array) {
	   Map<String, Integer> countedTips = new HashMap<>();
	   Arrays.stream(array)
	           .collect(Collectors.groupingBy(s -> s))
	           .forEach((k, v) -> countedTips.put(k, v.size()/2));
	   return countedTips;
	}


	/**
	 * enumerates Euler's path and returns a list of the result
	 */
	public Serializable enumerateEulerPath() {
	   if (arcs.isEmpty()) {
	      throw new RuntimeException("Please create a graph");
	   }
	   ArrayList<Arc> path = new ArrayList<>();
	   if (hasEulerPath()) {
	      String start = "";
	      for (Map.Entry<String, ArrayList<Arc>> entry : mapOfArcs.entrySet()) {
	         if (entry.getValue().size() % 2 == 1) {
	            start = entry.getKey();
	            break;
	         }
	      }
	      if (start.equals("")) {
```

```
                start = mapOfArcs.keySet().iterator().next();
            }
            ArrayList<Arc> copy = new ArrayList<>(arcs);
            while (arcs.size() != 0) {
                ArrayList<Arc> x = new ArrayList<>();
                for (Arc element : arcs) {
                    if (element.id.replaceAll("a", "").split("_")[0].equals(start))
                    x.add(element);
                }
                for (Arc item : x) {
                    String[] y = item.id.replaceAll("a", "").split("_");
                    if (this.makeMap(getStringArray(vertexes)).get(y[1]) > 1) {
                        arcs.remove(item);
                        path.add(item);
                        for (Arc a : copy) {
                            if (a.target.id.equals(y[0]) && a.id.replaceAll("a",
"").split("_")[0].equals(y[1])) {
                                arcs.remove(a);
                                break;
                            }
                        }
                        start = item.target.id;
                        break;
                    }
                }
            }
        }
        else {
            return ("The graph has no euler path, therefore the verteces cannot be
enumerated!");
        }
        return path;
    }
  }
}
```

Kasutatud allikad:

1. https://www.geeksforgeeks.org/fleurys-algorithm-for-printing-eulerian-path/
2. https://en.wikipedia.org/wiki/Eulerian_path
3. https://www.geeksforgeeks.org/eulerian-path-and-circuit/
4. https://research.cyber.ee/~peeter/teaching/graafid03s/graafid.pdf