Tallinna Tehnikaülikool

Individuaaltöö aines „Algoritmid ja andmestruktuurid“

Autor: Marko Bode

Eriala: IT süsteemide arendus

Juhendaja: Jaanus Pöial

2020 Tallinn

Sisukord

1. Ülesande püstitus ... 3
2. Lahenduse kirjeldus ... 4
   2.1 Klassid ... 4
      2.1.1 Arc ... 4
      2.1.2 Vertex ... 4
      2.1.3 Graph ... 4
   2.2 Lahendus ülesande püstitusele ... 6
3. Programmi kasutamisjuhend ... 8
4. Testimiskava ... 9
5. Kasutatud allikad ... 13
Lisad ... 14
   Lisa 1. Programmi täielik tekst ... 14

1. Ülesande püstitus

Ülesanne on koostada meetod, mis leiab etteantud sidusa lihtgraafi G tsentri.

Graaf on lihtgraaf kui see graaf on orienteerimata ja selles graafis pole silmuseid ega kordseid servasid. [1]

Graaf on orienteerimata kui tema servad pole suunatud. [1]

Orienteerimata graaf on sidus kui igast tipust on võimalik koostada tee igasse tippu. [1]

Graafi tsentriks nimetakse sellist tippude hulka, mille ekstsentrilisused on väikseimad. [2, lk 777]

Tipu ekstsentrilisus on lühim tee kõige kaugemasse tippu. [2, lk 778]

2. Lahenduse kirjeldus

Lahenduse koodi nägemiseks vaata Lisa 1.

2.1 Klassid

Lahenduses on 3 klassi: Arc, Vertex, Graph.

2.1.1 Arc

Klassil Arc(eesti keeles Kaar) on 3 klassimuutujat: id, target ja next. Id on Arc objekti nimi ning see peaks olema unikaalne, sest meetod toString() tagastab selle. Target on Arc objekti lõpp-punkt, milleks on Vertex objekt. Next on järgmine kaar, mis on sama Vertex objekti küljes.

Klassil Arc on 2 konstruktorit: ühega saab Arc objekti luua määrates kõik 3 klassimuutujat ja teisega saab objekti luua nii, et määratud on ainult objekti id.

Meetod toString() tagastab Arc objekti id.

2.1.2 Vertex

Klassil Vertex(eesti keeles Tipp) on 5 klassimuutujat: id, next, first, info ja eccentricity. Id on Vertex objekti nimi ning see peaks olema unikaalne, sest meetod toString() tagastab selle. Next on järgmine Vertex objekt, mis ühenduses selle Vertex objektiga. First on Arc objekt, mida loetakse esimeseks sellest Vertex objektist välja minevaks kaareks. Info on arv, mida saab kasutada mõne meetodi töö kergendamiseks ning selle vaikeväärtus on 0. Eccentricity on selle Vertex objekti ekstsentrilisus.

Klassil Vertex on 2 konstruktorit: ühega saab Vertex objekti luua määrates objektile id, nexti ja firsti ning teistega saab luua objekti määrates ainult id. Väljad eccentricity ja info jäävad objekti loomisel määramata.

Meetod toString() tagastab Vertex objekti id.

2.1.3 Graph

Klassil Graph(eesti keeles Graaf) on 3 klassimuutujat: id, first, vertices. Id on Graph objekti nimi ning see võiks olla unikaalne, et erinevate Graph objektide vahel kergelt vahet teha. First on selle

graafi esimene Vertex. Vertices on list kõigist selle Graph objekti Vertex objektidest. Valitud on andmestruktuur List, sest Vertex objekte saab graafi juurde lisada.

Meetod toString() tagastab Graph objekti sõne kujul nii, et alguses on Graph objekti id, millele järgnevad kõik graafi Vertex objektid. Iga Vertex objekt on omal real. Iga rida koosneb Vertex objekti id-st, millele järgnevad kõik Arc objektid ehk kaared, mis sellest tipust väljuvad. Kui kaar on orienteerimata, siis on kaar topelt- Vertexist A Vertexisse B ning Vertexist B Vertexisse A.

Meetod createVertex(String vid) teeb uue Vertex objekti ja lisab selle graafi. Meetod saab sisendiks loodava Vertex objekti id. Meetod tagastab loodud Vertex objekti.

Meetod createArc(String aid, Vertex from, Vertex to) teeb uue Arc objekti ja ühendab graafis kaks Vertex objekti. Meetod saab sisendiks loodava Arc objekti id ning algus ja lõpp Vertex objektd. Meetod tagastab loodud Arc objekti.

Meetod createRandomTree(int n) genereerib suvalise orienteerimata, sidusa ja tsükliteta graafi ehk puu, millel on meetodi sisendiks olev n arv Vertex objekte.

Meetod createAdjMatrix() loob graafile ühenduste maatiksi. Kui kahe Vertex objekti vahel on Arc objekt siis on maatriksis väärtus 1, kui Kahe Vertex objekti vahel ei saa liikuda ühe sammuga, siis on maatriksis väärtus 0. Meetod tagastab ühenduste maatriksi massiivina, kus iga massiivi element on massiiv, mis koosneb määratud väärtustest 0 ja 1. Meetodi töö tulemusena on graafi Vertex objektidel info väljas mingi väärtus.

Meetod createRandomSimpleGraph(int n, int m) loob graafile n arvu Vertex objekte ning m arvu Arc objekte nende Vertex objektide vahele. Väärtused m ja n on meetodi sisendiks. Saadud graaf on orienteerimata, sidus ja tsükliteta. Suurim n väärtus on 2500 ning kaarte arv peab olema suurem kui n – 1 ja väiksem kui n *(n – 1) / 2. Väiksema m väärtuse korral ei ole võimalik ühendada kõiki tippe mingi teise tipuga ja graaf poleks enam sidus. Suurema m väärtuse korral oleksid vajalikud tsüklid.

Ülejäänud meetodid on seotud ülesande püstitusega ning on kirjeldatud järgmises punktis.

2.2 Lahendus ülesande püstitusele

Lahendus ülesande püstitusele koosneb viiest meetodist Graph klassis, kus kasutatakse eelnevalt kirjeldatud klassimuutujaid ja meetodeid. Kirjeldus algab n-ö kõige sügavamast meetodist liikudes edasi välimistesse.

Selleks, et leida graafi tsenter, peab eelnevalt leidma kõikide graafi tippude ekstsentrilisused. Selleks, et leida graafi tipu ekstsentrilisus, peab leidma tipu lühima tee kaugeimasse tippu. Selleks, et leida lühimat teed kasutan Floyd-Warshalli algoritmi.

Meetod getDistMatrix(int[][] adjMatrix) saab sisendiks graafi ühenduste maatriksi ning tagastab graafi kauguste maatriksi. Kui kahe Vertex objekti vahe on ühendus olemas, siis pannakse kauguste maatriksisse väärtus 1. Kui Arc objektidel oleks määratud pikkus, siis tuleks panna see väärtuseks. Kui kahe Vertex objekti vahel pole ühendust, siis pannakse kauguste maatriksisse võimatult suur arv. Määramata pikkuste korral piisab arvust 2 * graafi tippude arv + 1. Määratud pikkuste korral oleks hea kasutada Integer.MAX_VALUE / 2. Sellised positsioonid maatriksis, kus x telje Vertex objekt ja y telje Vertex objekt on võrdsed pannakse kauguste maatriksisse väärtus 0 ehk tipust iseendasse on kaugus alati 0, sest ülesande püstituses on keelatud tsüklid.

Meetod getShortestPaths() teeb graafile kauguste maatriksi kasutades meetodit getDistMatrix ning arvutab Floyd-Warshalli algoritmiga lühimad vahemaad kõikide tippude vahel. Algoritm kasutab kolmekordset tsüklit ja on kuupkeerukusega. Algoritm võrdleb igal tsüklil seni teada olevat teed punktis A punkti B uue teega, mis kasutab vahepunkti C. Kui tee punktist A punkti C ja seejärel punktist C punkti B on lühem kui tee punktist A punkti B, siis kauguste maatriksis asendatakse A ja B vaheline kaugus uue ja lühema kaugusega. Nii asenduvad ka kõik määramata pikkused, mis on kirjeldatud eelmise meetodi seletuses, mingisuguse määratud pikkusega, sest graaf on sidus.

Meetod getEccentricity(Vertex Vertex, int[][] distMatrix) saab sisendiks Vertex objekti, mille ekstsentrilisust otsitakse ning lühimate kauguste maatriksi, kust seda ekstsentrilisust otsida. Vaadatakse kõiki teepikkusi antud Vertex objektist teistesse selle graafi Vertex objektidesse ning ekstsentrilisuseks saab pikim vahemaa.

Meetod getCenter() leiab graafi tsentri. Kui graafil on 0 tippu, siis graafil pole tsentrit. Kui graafil on 1 tipp, siis graafi tsenter ongi see sama tipp. Kui graafil on rohkem kui 1 tipp, siis leitakse iga tipu ekstsentrilisus meetodiga getEccentricity, kuhu antakse kaasa lühimate kauguste maatriks, mis

saadakse meetodist getShortestPaths. Tsentriks saab tipp või tipud, mille ekstsentrilisused on madalaimad. Meetod tagastab List andmestruktuuri, kus on sees graafi tsentrisse kuuluvad tipud.

Meetod getCenterAsString() leiab graafi tsentri kasutades meetodit getCenter() ja väljastab tipu sõnena, et tsenter oleks lihtsamini loetaval kujul. Väljastatud lause sisaldab endas Graph objekti enda sõnevormi, millele järgneb tippude nimekiri, mis kuuluvad tsentrisse. Vertex objektid on kujutatud oma toString() meetodiga.

3. Programmi kasutamisjuhend

Selleks, et leida Graph objekti tsenter, peab välja kutsuma Graph objekti meetodit getCenter() või getCenterAsString(), parameetreid kaasa pole vaja anda. Meetodit getCenter() välja kutsuda kui kasutaja soovib saada tsentrit masintöödeldava objektina(antud lahenduses andmetüüp List, mis koosneb Vertex objektidest). Meetodit getCenterAsString() välja kutsuda kui kasutaja soovib tsentrit näha lihtsalt loetaval kujul.

4. Testimiskava

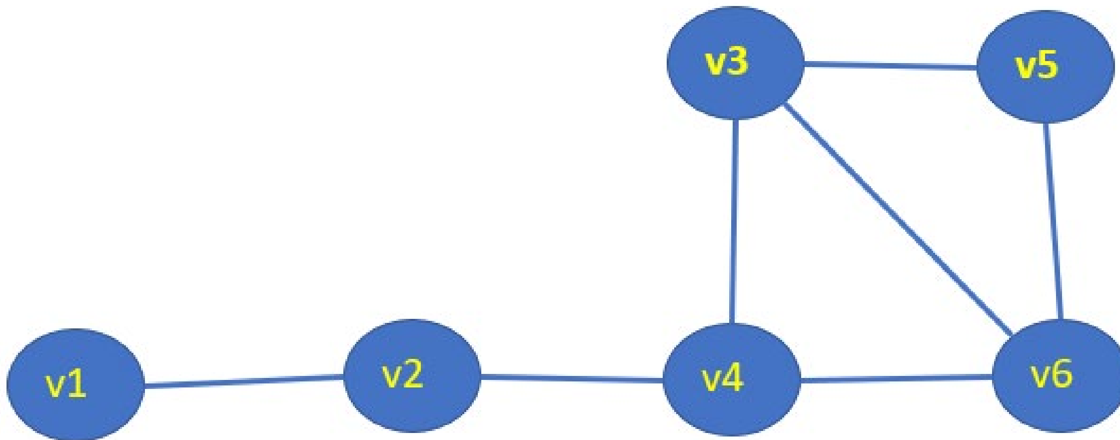
Testitakse erijuhtu, kus Graph objektil pole ühetegi Vertex objekti ehk graafil pole tippe. Meetod getCenter() väljastab tühja Listi ning getCenterAsString väljastab:

```
G

Graph with no vertices does not have a center!
```

Testitakse erijuhtu, kus Graph objektil on ainult 1 Vertex objekt ehk graafil on 1 tipp. Meetod getCenter() väljastab Listi, kus on sees ainult see 1 tipp. Meetod getCenterAsString() väljastab:

```
G
v1 -->

The center of Graph G(shown above) is v1
```

Testitakse juhtu, kus Graph objektil on palju tippe ja kaari ning graafi tsentriks tuleb ainult 1 tipp.



Joonisel on graaf, millel on 6 tippu ja 7 kaart.

Ekstsentrilisused:

v1 - 4

v2 - 3

v3 - 3

v4 - 2

v5 - 4

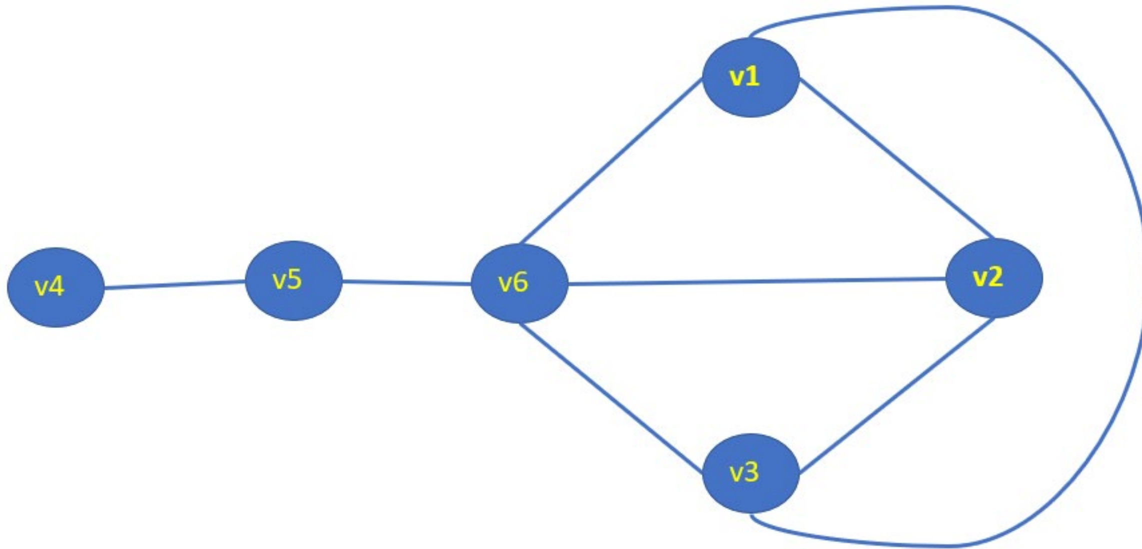
v6 - 3

Antud graafi tsentriks on tipp v4, sest tipu v4 ekstsentrilisus on väikseim.

Programmi meetod getCenterAsString() väljastab:

```
G
v1 --> av1_v2 (v1->v2)
v2 --> av2_v1 (v2->v1) av2_v4 (v2->v4)
v3 --> av3_v6 (v3->v6) av3_v4 (v3->v4) av3_v5 (v3->v5)
v4 --> av4_v3 (v4->v3) av4_v2 (v4->v2) av4_v6 (v4->v6)
v5 --> av5_v3 (v5->v3) av5_v6 (v5->v6)
v6 --> av6_v3 (v6->v3) av6_v4 (v6->v4) av6_v5 (v6->v5)

The center of Graph G(shown above) is v4
```

Testitakse juhtu, kus Graph objektil on palju tippe ja kaari ning graafi tsentriks tulevad mitu Vertex objekti.



Joonisel on graaf, millel on 6 tippu ja 8 kaart.

Ekstsentrilisused:

v1 - 3

v2 - 3

v3 - 3

v4 - 3

v5 - 2

v6 - 2

Antud graafi tsentriks on tipud v5 ja v6, sest nende tippude ekstsentrilisus on väikseimad.

Programmi meetod getCenterAsString() väljastab:

```
G
v1 --> av1_v3 (v1->v3) av1_v6 (v1->v6) av1_v2 (v1->v2)
v2 --> av2_v3 (v2->v3) av2_v1 (v2->v1) av2_v6 (v2->v6)
v3 --> av3_v1 (v3->v1) av3_v2 (v3->v2) av3_v6 (v3->v6)
v4 --> av4_v5 (v4->v5)
v5 --> av5_v4 (v5->v4) av5_v6 (v5->v6)
v6 --> av6_v1 (v6->v1) av6_v2 (v6->v2) av6_v3 (v6->v3) av6_v5 (v6->v5)

The center of Graph G(shown above) are vertices v5, v6
```

Testitakse juhtu, kus Graph objektil on 2000 tippu ja 3000 kaart. Mõõdetakse lahendustulemuse aega. Genereeriti 3 suvalist sellist graafi ja leiti iga graafi tsenter. Programmi väljund näitab keskmist aega 1 sellise graafi tsentri leidmiseks:

```
Average time to find the center is 19790 milliseconds
```

5. Kasutatud allikad

1. Jaanus Pöidla aine „Algoritmid ja andmestruktuurid“ graafiteooria materjalid. https://enos.itcollege.ee/~jpoial/algoritmid/graafid.html (21.11.2020)

2. Rosen, K. (2012) „Discrete Mathematics and Its Applications“.
https://notendur.hi.is/mbh6/html/_downloads/Discrete%20Mathematics%20and%20Its%20Applications%20-%20Kenneth%20Rosen%20(2012).pdf (21.11.2020)

Lisad

Lisa 1. Programmi täielik tekst

```java
import java.util.*;

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

    /**
     * Main method.
     */
    public static void main(String[] args) {
        GraphTask a = new GraphTask();
        a.run();
    }

    /**
     * Actual main method to run examples and everything.
     */
    public void run() {
        List<Long> times = new ArrayList<>();

        for (int i = 0; i < 2; i++) {
            Graph g = new Graph("G");
            g.createRandomSimpleGraph(2000, 3000);

            long startTime = System.currentTimeMillis();

            g.getCenterAsString();

            long endTime = System.currentTimeMillis();

            times.add(endTime - startTime);
        }
        long total = 0;
        for (long l :
                times) {
            total += l;
        }
        System.out.println("Average time to find the center is " + total / 2 + " milliseconds");
    }

    /**
     * Vertex class to define vertices for Graph class.
     */
    class Vertex {

        /**
         * ID of the vertex that should be unique.
         */
        private final String id;

        /**
         * Next Vertex that is connected to this Vertex.
         */
```

```java
    private Vertex next;

    /**
     * Arc that is between next Vertex and this Vertex.
     */
    private Arc first;

    /**
     * Info field to help different methods operations.
     */
    private int info = 0;

    /**
     * Eccentricity of the vertex.
     * If 0 then no eccentricity is defined yet.
     */
    private int eccentricity;

    /**
     * Constructor for Vertex class.
     *
     * @param s id for the Vertex
     * @param v first Vertex connected to this Vertex
     * @param e Arc between the two vertices
     */
    Vertex(String s, Vertex v, Arc e) {
        id = s;
        next = v;
        first = e;
    }

    /**
     * Second constructor to construct Vertex with only id.
     *
     * @param s id for the Vertex
     */
    Vertex(String s) {
        this(s, v: null, e: null);
    }

    /**
     * String representation of the Vertex.
     *
     * @return the id of the Vertex
     */
    @Override
    public String toString() {
        return id;
    }
}
```

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

    /**
     * ID of the Arc that should be unique.
     */
    private final String id;

    /**
     * The destination of the Arc
     */
    private Vertex target;

    /**
     * Next Arc for this Arc
     */
    private Arc next;

    /**
     * Constructor for Arc
     *
     * @param s id of the Arc
     * @param v target of this Arc
     * @param a next Arc for this Arc
     */
    Arc(String s, Vertex v, Arc a) {
        id = s;
        target = v;
        next = a;
    }

    /**
     * Second constructor to construct Arc with only id.
     *
     * @param s id of the Arc
     */
    Arc(String s) {
        this(s, v: null, a: null);
    }

    /**
     * String representation of the Arc.
     *
     * @return id of the Arc
     */
    @Override
    public String toString() {
        return id;
    }
}
```

```java
/**
 * Graph consists of Vertices and Arcs.
 */
class Graph {

    /**
     * ID of the Graph which should be unique.
     */
    private final String id;

    /**
     * First Vertex of this Graph.
     */
    private Vertex first;

    /**
     * List of all the Vertices that are present in this Graph.
     */
    private final List<Vertex> vertices = new ArrayList<>();

    /**
     * Constructor for Graph.
     *
     * @param s id for the Graph
     * @param v first Vertex of this graph
     */
    Graph(String s, Vertex v) {
        id = s;
        first = v;
    }

    /**
     * Second constructor to construct the Graph with only id.
     *
     * @param s id for the Graph
     */
    Graph(String s) {
        this(s, v: null);
    }

    /**
     * String representation of the Graph beginning with the id of the Graph.
     * Then followed by each Vertex in this Graph and each Vertex`s Arcs.
     * If Arc does not have a direction then the Arc is represented twice- from A to B and then from B to A.
     *
     * @return graph as a string
     */
    @Override
    public String toString() {
        String nl = System.getProperty("line.separator");
        StringBuilder sb = new StringBuilder(nl);
        sb.append(id);
        sb.append(nl);
        Vertex v = first;
        while (v != null) {
            sb.append(v.toString());
            sb.append(" -->");
            Arc a = v.first;
```

```java
221            while (a != null) {
222                sb.append(" ");
223                sb.append(a.toString());
224                sb.append(" (");
225                sb.append(v.toString());
226                sb.append("->");
227                sb.append(a.target.toString());
228                sb.append(")");
229                a = a.next;
230            }
231            sb.append(nl);
232            v = v.next;
233        }
234        return sb.toString();
235    }
236
237    /**
238     * Create new Vertex for this Graph.
239     *
240     * @param vid new Vertex`s id
241     * @return created Vertex
242     */
243    public Vertex createVertex(String vid) {
244        Vertex res = new Vertex(vid);
245        res.next = first;
246        first = res;
247        return res;
248    }
249
250    /**
251     * Create new Arc for this Graph.
252     *
253     * @param aid  new Arc`s id
254     * @param from Arc`s start point
255     * @param to   Arc`s destination
256     */
257 @  public void createArc(String aid, Vertex from, Vertex to) {
258        Arc res = new Arc(aid);
259        res.next = from.first;
260        from.first = res;
261        res.target = to;
262    }
263
264    /**
265     * Create a connected undirected random tree with n vertices.
266     * Each new vertex is connected to some random existing vertex.
267     *
268     * @param n number of vertices added to this graph
269     */
270    public void createRandomTree(int n) {
271        if (n <= 0)
272            return;
273        Vertex[] varray = new Vertex[n];
274        for (int i = 0; i < n; i++) {
275            varray[i] = createVertex( vid: "v" + (n - i));
276            if (i > 0) {
277                int vnr = (int) (Math.random() * i);
```

```java
                createArc( aid: "a" + varray[vnr].toString() + "_"
                        + varray[i].toString(), varray[vnr], varray[i]);
                createArc( aid: "a" + varray[i].toString() + "_"
                        + varray[vnr].toString(), varray[i], varray[vnr]);
            }
        }
    }

    /**
     * Create an adjacency matrix of this graph.
     * Side effect: corrupts info fields in the graph
     *
     * @return adjacency matrix
     */
    public int[][] createAdjMatrix() {
        int 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;
```

```java
        while (v != null) {
            vertices.add(v);
            vert[c++] = v;
            v = v.next;
        }
        int[][] connected = createAdjMatrix();
        int edgeCount = m - n + 1;  // remaining edges

        while (edgeCount > 0) {
            int i = (int) (Math.random() * n);  // random source
            int j = (int) (Math.random() * n);  // random target
            if (i == j)
                continue;  // no loops
            if (connected[i][j] != 0 || connected[j][i] != 0)
                continue;  // no multiple edges
            Vertex vi = vert[i];
            Vertex vj = vert[j];
            createArc( aid: "a" + vi.toString() + "_" + vj.toString(), vi, vj);
            connected[i][j] = 1;
            createArc( aid: "a" + vj.toString() + "_" + vi.toString(), vj, vi);
            connected[j][i] = 1;
            edgeCount--;  // a new edge happily created
        }
    }

    /**
     * Create a distance matrix of this graph.
     *
     * @param adjMatrix adjacency matrix of the graph
     * @return distance matrix of the graph
     */
    public int[][] getDistMatrix(int[][] adjMatrix) {
        int vertexes = this.vertices.size(); // number of vertices

        int[][] result = new int[adjMatrix.length][adjMatrix.length];
        if (vertexes < 1) return result;
        int noPath = 2 * vertexes + 1; // something impossible
        for (int i = 0; i < vertexes; i++) {
            for (int j = 0; j < vertexes; j++) {
                if (i == j) {
                    result[i][j] = 0;
                }
                if (adjMatrix[i][j] == 0) {
                    result[i][j] = noPath; // no path between vertices i and j
                } else {
                    result[i][j] = 1; // value is the length of the arc
                }
            }
        }
        return result;
    }

    /**
     * Floyd-Warshall`s algorithm to get the shortest paths from each vertex to any other vertex.
     *
     * @return distance matrix where each distance is as short as possible
     */
```

```
391    public int[][] getShortestPaths() {
392        // Algorithm`s base taken from https://enos.itcollege.ee/~jpoial/algoritmid/graafid.html
393        // Modified this and getDistMatrix() to fit my requirements.
394
395        int n = this.vertices.size(); // number of vertices
396
397        int[][] distMatrix = getDistMatrix(createAdjMatrix());
398        for (int k = 0; k < n; k++) {
399            for (int i = 0; i < n; i++) {
400                for (int j = 0; j < n; j++) {
401                    int newLength = distMatrix[i][k] + distMatrix[k][j];
402                    if (distMatrix[i][j] > newLength) {
403                        distMatrix[i][j] = newLength; // new path is shorter
404                    }
405                }
406            }
407        }
408        return distMatrix;
409    }
410
411    /**
412     * Find the distance between given vertex and the vertex furthest from the given vertex.
413     *
414     * @param vertex     the vertex which eccentricity is wanted
415     * @param distMatrix graphs distance matrix
416     * @return eccentricity of the vertex
417     */
418 @  public int getEccentricity(Vertex vertex, int[][] distMatrix) {
419        int index = vertices.indexOf(vertex);
420        int eccentricity = 0;
421        for (int i = 0; i < distMatrix.length; i++) {
422            if (distMatrix[index][i] > eccentricity) {
423                eccentricity = distMatrix[index][i];
424            }
425
426        }
427        return eccentricity;
428    }
429
430    /**
431     * Find the center of this graph. Center is the vertex which eccentricity is the smallest.
432     * If several vertices share the smallest eccentricity then all these vertices are a part of the center.
433     *
434     * @return center of the graph as a list of vertices
435     */
436    public List<Vertex> getCenter() {
437        List<Vertex> center = new ArrayList<>();
438
439        if (vertices.size() < 1) {
440            return center;
441        } else if (vertices.size() == 1) {
442            center.add(vertices.get(0));
443            return center;
444        }
445
446        int[][] distMatrix = getShortestPaths();
447
```

```
448            for (Vertex vertex : vertices) {
449                vertex.eccentricity = getEccentricity(vertex, distMatrix);
450                if (center.size() == 0) {
451                    center.add(vertex);
452                } else if (vertex.eccentricity < center.get(0).eccentricity) {
453                    center = new ArrayList<>();
454                    center.add(vertex);
455                } else if (vertex.eccentricity == center.get(0).eccentricity) {
456                    center.add(vertex);
457                }
458            }
459            return center;
460        }
461
462        /**
463         * Get the center of this Graph as a string.
464         *
465         * @return sentence to describe the center of this Graph
466         */
467        public String getCenterAsString() {
468            List<Vertex> center = getCenter();
469            String result = this.toString() + "\n";
470            if (center.size() == 0) {
471                result += "Graph with no vertices does not have a center!";
472            }
473            else if (center.size() == 1) {
474                result += "The center of Graph " + id + "(shown above) is " + center.get(0);
475            } else {
476                StringBuilder vertices = new StringBuilder();
477                for (Vertex vertex : center) {
478                    if (center.indexOf(vertex) != center.size() - 1) {
479                        vertices.append(vertex);
480                        vertices.append(", ");
481
482                    } else {
483                        vertices.append(vertex);
484                    }
485
486                }
487                result += "The center of Graph " + id + "(shown above) are vertices " + vertices.toString();
488            }
489            return result;
490        }
491    }
492 }
493
```