SVEUČILIŠTE U ZAGREBU
Fakultet elektrotehnike i računarstva

Predmet: Informacijske mreže (34457)
Ak. godina: 2014./2015.

Laboratorijske vježbe:
I Z V J E Š T A J

Grupa: 11
Članovi grupe:
Danijela Barišić, 0036448426
Adrijan Jakšić, 0036450452
Melita Kerep, 0036448788
Ana Kikaš, 0036446619
Tin Novak, 0036457591

Sadržaj

Zadatak

Svaka grupa studenata treba u slobodno odabranom programskom jeziku (preporuka JAVA) kreirati programsku podršku koja će rješavati, mrežni algoritam. Mrežni algoritam koji je dodijeljen je Algoritam najkraćeg puta (engl.*Shortest Path Algorithms*) – Belman-Ford.

Programska podrška treba imati sljedeće funkcionalnosti:

a) Unos mrežne topologije (proizvoljan broj čvorova, grana poveznica među čvorovima te njihovih težina i usmjerenja (ako je potrebno)); Unos mrežne topologije treba biti realiziran preko tekstualne datoteke sljedeće strukture: „*čvor_1, čvor_2, težina_grane, usmjerenje_grane;“*. Svaki par čvorova i granja među njima treba biti definirana u jednom redu ulazne datoteke. *Čvor_1* i *čvor_2* predstavljaju oznake (brojevi) čvorova i poprimaju vrijednosti 0, 1, 2, 3, itd., *težina_grane* predstavlja težinu grane i može poprimit vrijednosti 0 (bez težine), 1, 2, 3, itd., usmjerenje_grane predstavlja je li grana usmjerena ili ne i može poprimit vrijednosti „u“ i „n“. Usmjerenje se određuje poretkom čvorova, prvi navedeni čvor je izvorište a drugi navedeni čvor je ishodište grane.
b) Implementaciju zadanog *mrežnog algoritma*;
c) Grafički prikaz kroz koji će se prikazati i objasniti svaki korak algoritma do konačnog rješenja;
d) Prikaz rezultata pojedinog algoritma.

U izvještaju treba biti na primjeru prikazano kako programska podrška rješava mrežnu topologiju od barem 6 čvorova.

Pseudokod algoritma

Prije samog algoritma imamo inicijalizirano polje udaljenost čije su vrijednosti na početku beskonačnost svima osim izvorišnom polju, njemu je 0. Također inicijalizirano je polje roditelj na početku s vrijednostima izvorišnog čvora.

```
Za  i =1 do broj čvorova -1 {
    Za j=0 do isključeno broj grana {
        Ako udaljenost do izvorišnog čvora j-te grane=beskonačno
            Nastavi u novu j iteraciju

        nova udaljenost = udaljenost do izvorišnog čvora j-te grane
                        + težina j-te grane

        Ako nova udaljenost < udaljenosti do odredišnog čvora j-te grane {
            udaljenost do odredišnog čvora j-te grane = nova udaljenost
            roditelj odredišta j-te grane = izvor j-te grane
        }
    }
    Predaj argumente metodi za ispis puta za ovu iteraciju i
}
```

Ovaj dio koda je ima vanjsku petlju koja se vrti jedan broj puta manje od broja čvorova kao po definiciji algoritma. Unutarnja petlja prolazi kroz sve grane u objektu klase edges koja sadrži samu topologiju mreže (izvor, odredište, težinu grane između njih dvoje). Unutar te petlje osigurali smo na početku da provjeravamo samo za grane čiji izvor nema udaljenost beskonačnost, a kasnije provjeravamo dal udaljenost do izvorišnog čvora u zbroju s težinom aktualne grane je manja od same upisane vrijednosti udaljenosti do odredišta u polju udaljenost. Ako je tako upisujemo tu novu vrijednost koja je manja. Također aktualni izvor od te grane upisujemo u polje roditelj kao prvi prethodni čvor prije aktualnog. To nam omogućava da slanjem tih argumenata u metodi path ispisujemo korake, tj. put od izvora do aktualnog odredišta i tako za svaku provjeru grane.

```
Za sve i=0 do broj udaljenosti
    Ako udaljenost < 0
        ima negativnih grana
```

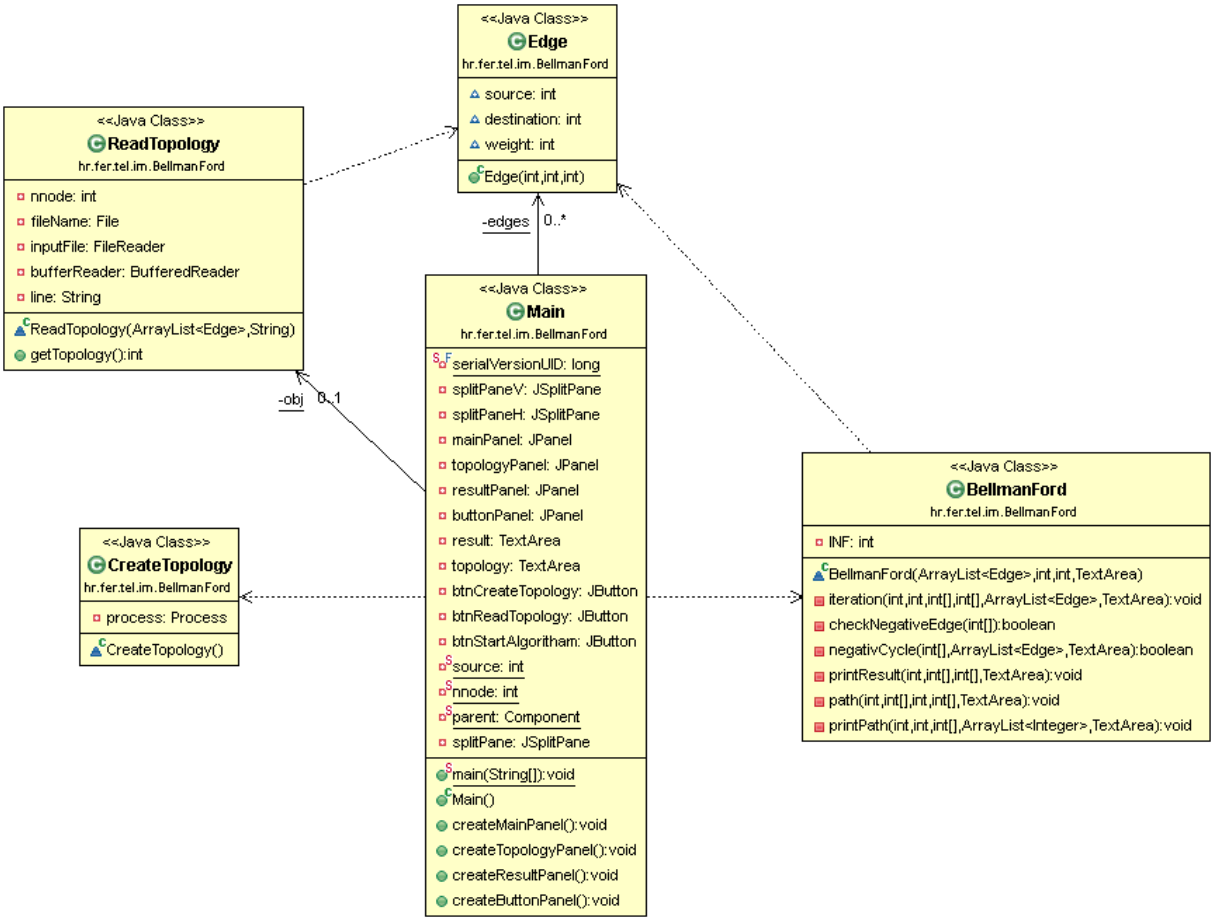
Provjeravamo ima li negativnih grana u topologiji

```
Ako postoji negativnih grana {
    ok = true
    Za j=0 do isključeno broj grana
        Ako udaljenost do izvora j-te grane != beskonačnosti i
        udaljenost odredišnog čvora j-te grane > udaljenosti izvorišnog čvora
        j-te grane + težine j-te grane {
            Zapiši u rezultat „Otkrivena petlja negativne težine!“
            ok = false
            izađi iz petlje
        }
    Vrati ok
}
```

Ako je bilo negativnih grana po definiciji algoritma trebamo još jedan korak algoritma ponoviti i ako dođe do promjene stanja udaljenosti javljamo da je otkrivena petlja negativne težine.

Dokumentacija

Opis strukture programskog rješenja i dijagram klasa



Kratki opis funkcionalnosti svake klase

Klasa Main

Klasa Main sadrži metodu main i konstruktor klase Main. Metoda main se poziva prilikom pokretanja programa. U metodi main stvaramo objekt klase Main, tako da se pozove konstruktor klase. U konstruktoru stvaramo grafičko sučelje aplikacije koje se sastoji od tri dijela. Prvi dio je izbornik s gumbima koji se nalazi s lijeve strane prozora, zatim imamo središnji dio gdje ispisujemo rezultat izvođenja programa i desni dio prozora u kojem se prikazuje učitana topologija. Na kreirane gumbe postavljamo *ActionListenere* koji prilikom pritiska gumba stvaraju objekte različitih klasa.

Pritiskom na gumb:

- *Kreiraj topologiju* stvara se objekt klase *CreateTopology* i poziva se konstruktor. Isječak koda u kojem je prikazano stvaranje objekta klase *CreateTopology* prikazan je u nastavku.

```
btnCreateTopology.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        new CreateTopology();
    }
});
```

- *Učitaj topologiju* otvara se izbornik *Chooser* u kojem biramo datoteku u kojoj se nalazi zadana topologija. Nakon što je topologija izabrana stvara se objekt klase *ReadTopology* i poziva se konstruktor kojem prosljeđujemo polje *Edge* i varijabla *path.* U polje *Edge* ćemo spremiti objekte klase *Edge* koje ćemo stvoriti prilikom čitanja datoteke. *Path* je apsolutni put do datoteke u kojoj se nalazi topologija. Isječak koda u kojem je prikazano pozivanje izbornika za biranje datoteka i stvaranje objekta klase *ReadTopology* prikazan je u nastavku.

```
btnReadTopology.addActionListener(new ActionListener() {
    @Override
    @SuppressWarnings("deprecation")
    public void actionPerformed(ActionEvent e) {
        JFileChooser chooser = new JFileChooser();
        chooser.showOpenDialog(parent);
        String path = chooser.getSelectedFile().getAbsolutePath();
        obj = new ReadTopology(edges, path);
        for (int i = 0; i < edges.size(); i++) {
           topology.appendText(edges.get(i).source + ", "
                               + edges.get(i).destination + ", "
                               + edges.get(i).weight + "\n");
        }
        nnode = obj.getTopology();
        for (int i = 0; i < nnode; i++) {
           sourceChoice.add(i + "\n");
        }
    }
});
```

- *Pokreni Algoritam* stvara se objekt klase *BellmanFord* i poziva se konstruktor kojem prosljeđujemo polje *Edge*, varijable *source i nnode* i TextArea *result.* U polju *Edge* se nalaze stvoreni objekti klase *Edge*. U varijabli *source* se nalazi izvorišni čvor, a u varijabli *nnode* se nalazi ukupan broj čvorova. TextArea *result* je prostor u GUI-u gdje ispisujemo rezultat izvođenja algoritma. Isječak koda u kojem je prikazano stvaranje objekta klase *BellmanFord* prikazan je u nastavku.

```
btnStartAlgoritham.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        if (nnode > 0 && edges.size() > 0 && source < nnode) {
            new BellmanFord(edges, source, nnode, result);
        }
        if (source >= nnode) {
            System.out.println("Vrijednost najvećeg čvora je "
                               + (nnode - 1));
        }
```

```
            if (nnode <= 0) {
               System.out.println("U učitanoj topologiji nema čvorova");
            }
            if (edges.size() <= 0) {
               System.out.println("Učitan je nepovezan graf. Niti jedan
                                   čvor nije povezan!");
            }
        }
    });
```

Klasa CreateTopology

Klasa *CreateTopology* sadrži samo konstruktor koji se poziva pritiskom na gumb *Kreiraj topologiju* kada se stvara objekt klase *CreateTopology.* U konstruktoru ove klase imamo *try/catch* blok, te u *try* dijelu bloka pozivamo program run.jar koji služi za generiranje topologije. Također osigurano je da se daljnje operacije u klasi ne izvode dok se program run.jar ne izvrši u potpunosti. Ako dođe do neke greške prilikom kreiranja topologije ispisuje se poruka o pogrešci. Opisani *try/catch* blok prikazan je u nastavku.

```
try {
   process = Runtime.getRuntime().exec("java -jar run.jar");
   System.out.println("Čekanje unosa mrežne toplogije ...");
   process.waitFor();
}
catch (InterruptedException | IOException e) {
   System.out.println("Dogodila se pogreška prilikom kreiranja
                       topologije:" + e.getMessage());
}
```

Klasa ReadTopology

Klasa *ReadTopology* sadrži konstruktor i metodu *getTopology.* Konstruktor se poziva pritiskom na gumb *Učitaj topologiju* kada se stvara objekt klase *ReadTopology.* Konstruktoru smo proslijedili polje *Edge* i varijablu *path.* U prazno polje *Edge* u koje ćemo spremiti objekte klase *Edge*, a u varijabli *path* se nalazi apsolutni put do datoteke. U konstruktoru ove klase koristi se *try*/*catch* blok, u *try* dijelu bloka učitavamo datoteku na temelju dobivenog apsolutnog puta iza varijable *path.* Zatim stvaramo objekt klase *FileReader* i stvaramo novi objekt klase *BufferReader.* Nakon toga čitamo redak po redak iz ulazne datoteke, te je razdvajamo svaku liniju po nizu znakova ", " i spremamo u polje stringova *var*. Za svaku tako učitanu liniju stvaramo objekt klase *Edge* i spremamo u polje *edges.* Također prilikom čitanja svih linija iz ulazne datoteke određujemo koliki je broj čvorova u zadanoj topologiji, tu vrijednost spremamo u varijablu *nnode*. Ako dođe do neke greške prilikom učitavanja topologije ispisuje se poruka o pogrešci. Opisani *try*/*catch* blok prikazan je u nastavku.

```
try {
   fileName = new File(path);
   inputFile = new FileReader(fileName);
   bufferReader = new BufferedReader(inputFile);
   while ((line = bufferReader.readLine()) != null) {
      String[] var = line.split(", ");
      edges.add(new Edge(Integer.parseInt(var[0]),
                         Integer.parseInt(var[1]),
                         Integer.parseInt(var[2])));
      if (nnode < Integer.parseInt(var[0])) {
         nnode = Integer.parseInt(var[0]);
      }
      if (nnode < Integer.parseInt(var[1])) {
         nnode = Integer.parseInt(var[1]);
      }
   }
   bufferReader.close();
```

```
} catch (IOException e) {
      System.out.println("Dogodila se pogreška prilikom čitanja
                          datoteke:" + e.getMessage());
}
```

Klasa Edge

Klasa *Edge* sadrži samo jedan konstruktor kojem se prosljeđuju parametri *s*(*source*), *d*(*destination*) i *w*(*weight*). Ovaj konstruktor se poziva s parametrima izvorišni čvor, odredišni čvor i težina između čvorova, prilikom stvaranja objekta ove klase u konstruktoru klase *ReadTopology*. Gdje za svaku učitanu liniju stvaramo objekt ove klase. Konstruktor klase *Edge* prikazan je u nastavku.

```
public Edge(int s, int d, int w) {
   source = s;
   destination = d;
   weight = w;
}
```

Klasa BellmanFord

Klasa *BellmanFord* je najvažnija klasa i u njoj se nalaze metode cijele logika algoritma, te metode za ispis konačnog rješenja. Klasa *BellmanFord* osim konstruktora sadrži sljedeće metode *iteration, checkNegativeEdge, negativeCycle, printResult, path, printPath.* U nastavku ćemo ukratko opisati prvo konstruktor, a zatim i svaku , te ćemo prikazati najvažnije dijelove konstruktora i opisanih metoda.

- Prilikom stvaranja objekta ove klase konstruktoru se prosljeđuju sljedeći parametri: lista *edges*, *source*, *nnode* i *result*. U konstruktoru inicijaliziramo

određene varijable i polja koje ćemo koristiti u daljnjim metodama ove klase. Važno je spomenuti polje *distance* koje je tipa integer, njegova veličina odgovara broju čvorova. Polje predstavlja udaljenost od početnog čvora do ostalih čvorova i te udaljenosti su postavljene na beskonačnu vrijednost, a udaljenost do početnog čvora postavljena je na 0. Također tu je polje *parent* koje je također tipa integer i njegova veličina isto odgovara broju čvorova. Ovo polje koristimo kako bi "zapamtili" preko kojih čvorova smo došli od izvorišnog do drugih čvorova. U nastavku je prikazan konstruktor ove klase.

```
BellmanFord(ArrayList<Edge> edges, int source, int nnode,
        TextArea result) {
   int[] distance = new int[nnode];
   Arrays.fill(distance, INF);
   int[] parent = new int[nnode];
   Arrays.fill(parent, source);
   distance[source] = 0;
   boolean negative;
   boolean ok = true;
   iteration(nnode, source, distance, parent, edges, result);
   negative = checkNegativeEdge(distance);
   if (negative) {
      ok = negativCycle(distance, edges, result);
   }
   if (ok) {
      printResult(source, distance, parent, result);
   }
}
```

- Iz gore prikazanog koda možemo vidjeti kako konstruktor prvo poziva metodu *iteration*. Metoda se poziva sa parametrima *nnode, source, distance, parent, edges* i *result.* U ovoj metodi je ostvarena funkcionalnost algoritma i to na način da prolazimo kroz dvije ugnježđene iteracije. Prva iteracija ide po čvorovima i to od 1 do V-1, a druga iteracija koja je ugđnježđena unutar prve ide po svim bridovima odnosno po polju objekata *edges* i to se ponavlja tako za svaki čvor u mreži. Unutar druge iteracije imamo nekoliko uvjeta. Prvi uvjet je da element u polju *distance* mora biti različit od beskonačno, ako nije tada se preskaču daljnji

koraci i gleda se sljedeći element. Ako je prethodni uvjet zadovoljen i element u polju *distance* je različit od beskonačno, tada računamo novu udaljenost do odredišnog čvora trenutnog objekta *edges,* tako da na vrijednost trenutnog elementa u polju(*distance*[*edges.get(j).source*]) dodajemo udaljenost do odredišnog čvora (*edges.get(j).weight*). U sljedećem koraku uspoređujemo novu udaljenost do odredišnog čvora, s onom udaljenosti koja je zapisana za odredišni čvor u polju *distance*. Ako je nova udaljenost manja od stare, tada u polje *distance* na mjesto koje odgovara elementu odredišnog čvora zapisujemo novu vrijednost. Dodatno u polje *parent* zapisujemo preko kojeg čvora smo došli do odredišnog čvora. Isječak metode *iteration* prikazana je u nastavku.

```
for (int i = 1; i <= nnode - 1; i++) {
   for (int j = 0; j < edges.size(); j++) {
      if (distance[edges.get(j).source] == INF) {
         continue;
      }
      int newDistance = distance[edges.get(j).source]
                        + edges.get(j).weight;

      if (newDistance < distance[edges.get(j).destination]) {
         distance[edges.get(j).destination] = newDistance;
         parent[edges.get(j).destination] = edges.get(j).source;
      }
   }
}
```

- Nakon što je algoritam prošao kroz V-1 iteracija završava prethodna metoda i možemo vidjeti u konstruktoru da je sljedeća metoda koju pozivamo *checkNegativeCycle*. Ovu metodu pozivamo s parametrom *distance* koji je potreban kako bi prošli kroz sve udaljenosti i utvrdili da li postoji negativna udaljenost. Ako u polju *distance* postoji negativna udaljenost tada metoda vraća logičku vrijednost false u suprotnom metoda vraća logičku vrijednost true. Ova metoda prikazana je u nastavku.

```
private boolean checkNegativeEdge(int[] distance) {
    boolean negative = false;
    for (int i = 0; i < distance.length; i++) {
        if (distance[i] < 0) {
            negative = true;
        }
    }
    return negative;
}
```

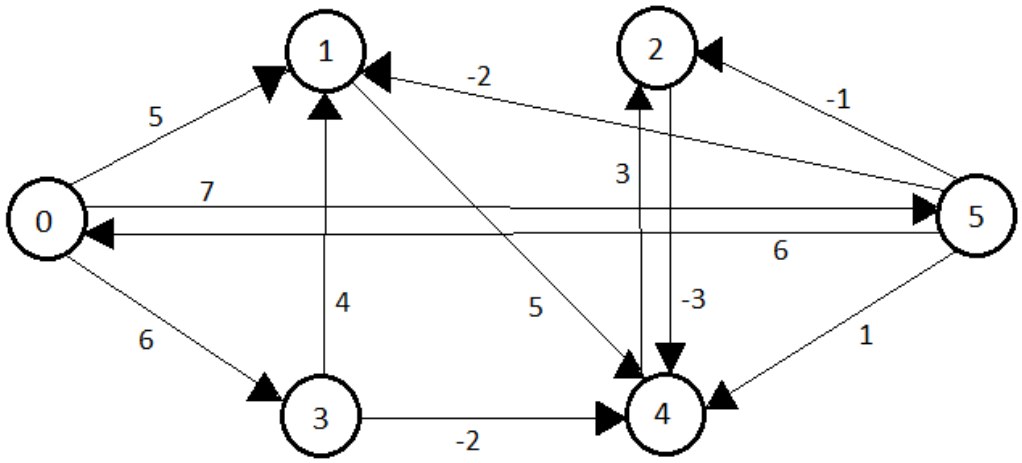
- Ako metoda *checkNegativeCycle* vrati logičku vrijednost true, tada u konstruktoru prolazi uvjet i poziva se metoda *negativeCycle* koja iterira po polju objekata *edges* i provjerava da li postoji negativna petlja. U slučaju da je prethodna metoda vratila logičku vrijednost false ovaj korak se preskače.

- Zadnja metoda koja se poziva u konstruktoru je metoda *printResult*, ova metoda se poziva samo u slučaju da ne postoji negativna petlja u mreži. Ako postoji negativna petlja, tada se ova metoda ne poziva. Ova metoda samo ispisuje konačni rezultat algoritma u zadanom obliku.

- Postoje još dvije metode koje se koriste kao pomoćne metode. To su metode *path* i *printPath.* Metoda *path* računa put preko kojih čvorova smo prolazili od izvorišnog čvora do ostalih čvorova u mreži, a metoda *printPath* ispisuje taj rezultat.

Grafički prikaz algoritma korak po korak

Generirana topologija:



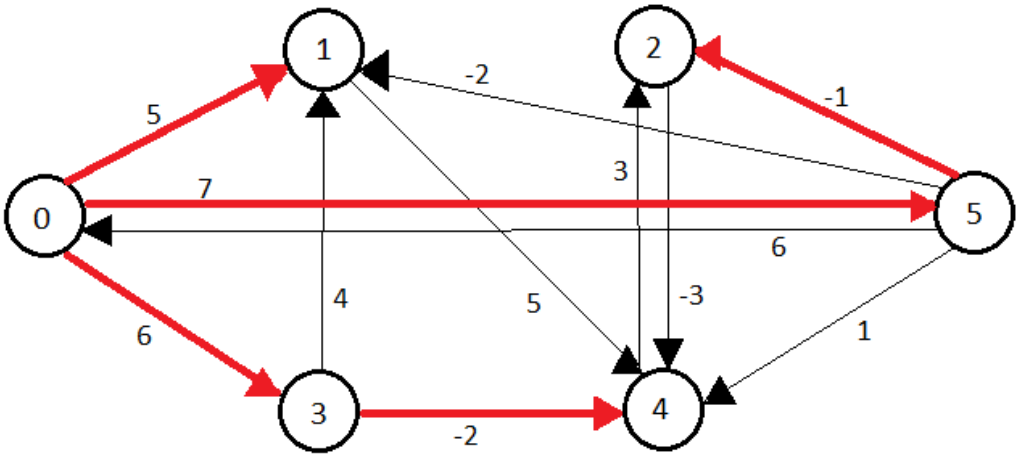
- **Broj iteracija i: |V|-1 = 6-1 = 5**

Početno stanje:

i	0	1	2	3	4	5
0	-	∞	∞	∞	∞	∞

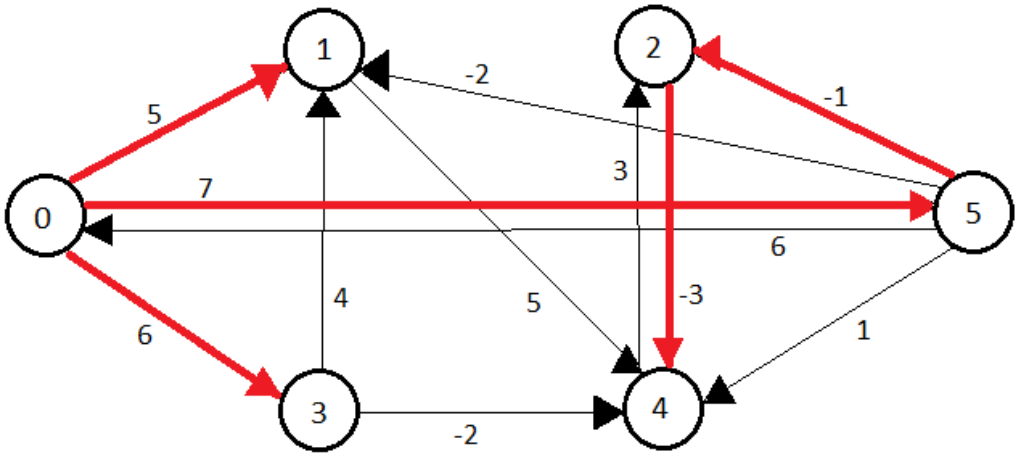
Prva iteracija:

i	0	1	2	3	4	5
0	-	∞	∞	∞	∞	∞
1	-	**0 -> 1: 5**	**0 -> 5 -> 2: 6**	**0 -> 3: 6**	**0 -> 3 -> 4: 4**	**0 -> 5: 7**



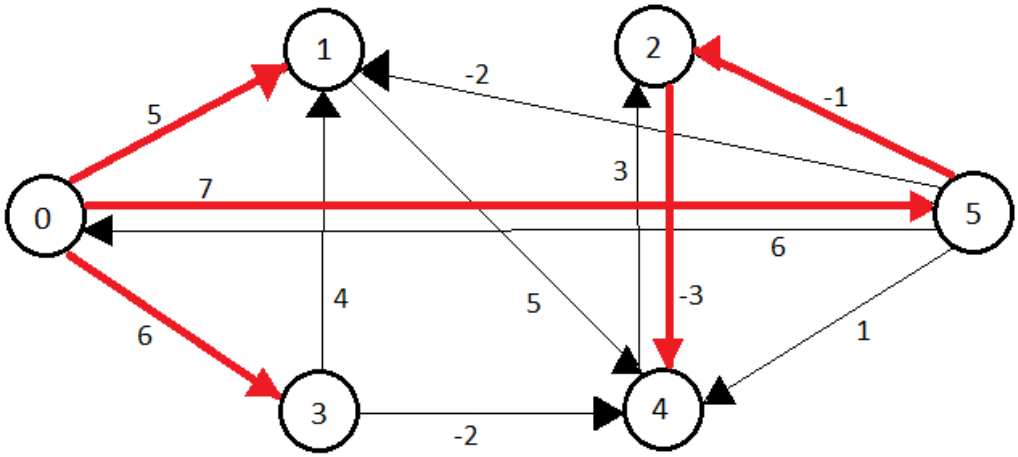
Druga iteracija:

i	0	1	2	3	4	5
0	-	∞	∞	∞	∞	∞
1	-	0 -> 1: 5	0 -> 5 -> 2: 6	0 -> 3: 6	0 -> 3 -> 4: 4	0 -> 5: 7
2	-	**0 -> 1: 5**	**0 -> 5 -> 2: 6**	**0 -> 3: 6**	**0 -> 5 -> 2 -> 4: 3**	**0 -> 5: 7**



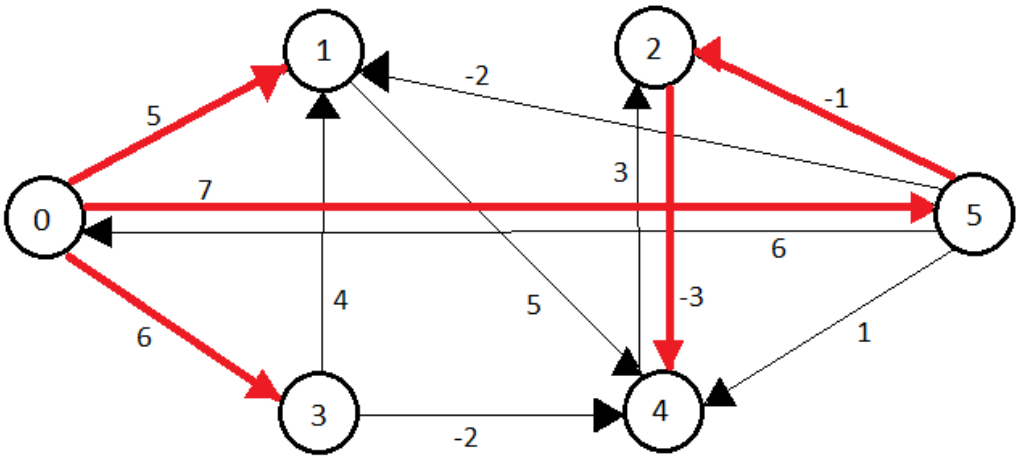
Treća iteracija:

i	0	1	2	3	4	5
0	-	∞	∞	∞	∞	∞
1	-	0 -> 1: 5	0 -> 5 -> 2: 6	0 -> 3: 6	0 -> 3 -> 4: 4	0 -> 5: 7
2	-	0 -> 1: 5	0 -> 5 -> 2: 6	0 -> 3: 6	0 -> 5 -> 2 -> 4: 3	0 -> 5: 7
3	-	**0 -> 1: 5**	**0 -> 5 -> 2: 6**	**0 -> 3: 6**	**0 -> 5 -> 2 -> 4: 3**	**0 -> 5: 7**



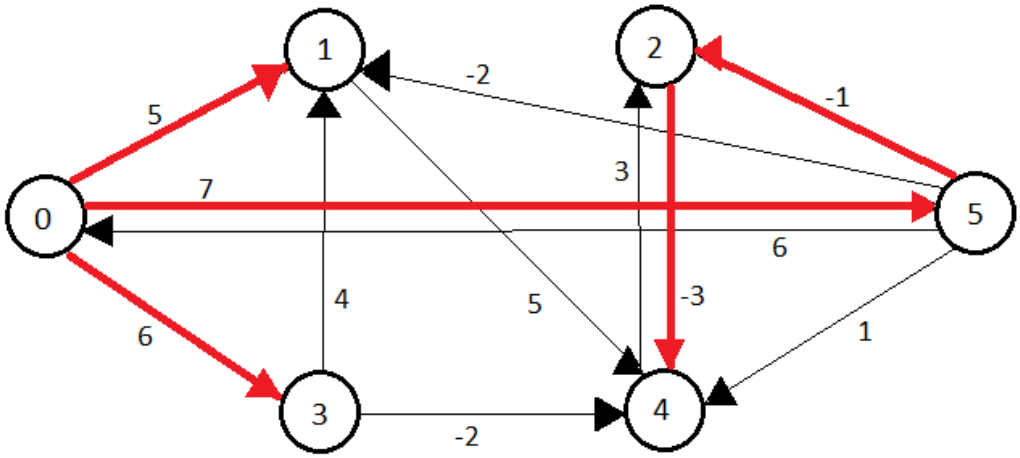
Četvrta iteracija:

i	0	1	2	3	4	5
0	-	∞	∞	∞	∞	∞
1	-	0 -> 1: 5	0 -> 5 -> 2: 6	0 -> 3: 6	0 -> 3 -> 4: 4	0 -> 5: 7
2	-	0 -> 1: 5	0 -> 5 -> 2: 6	0 -> 3: 6	0 -> 5 -> 2 -> 4: 3	0 -> 5: 7
3	-	0 -> 1: 5	0 -> 5 -> 2: 6	0 -> 3: 6	0 -> 5 -> 2 -> 4: 3	0 -> 5: 7
4	-	**0 -> 1: 5**	**0 -> 5 -> 2: 6**	**0 -> 3: 6**	**0 -> 5 -> 2 -> 4: 3**	**0 -> 5: 7**



Peta iteracija:

i	0	1	2	3	4	5
0	-	∞	∞	∞	∞	∞
1	-	0 -> 1: 5	0 -> 5 -> 2: 6	0 -> 3: 6	0 -> 3 -> 4: 4	0 -> 5: 7
2	-	0 -> 1: 5	0 -> 5 -> 2: 6	0 -> 3: 6	0 -> 5 -> 2 -> 4: 3	0 -> 5: 7
3	-	0 -> 1: 5	0 -> 5 -> 2: 6	0 -> 3: 6	0 -> 5 -> 2 -> 4: 3	0 -> 5: 7
4	-	0 -> 1: 5	0 -> 5 -> 2: 6	0 -> 3: 6	0 -> 5 -> 2 -> 4: 3	0 -> 5: 7
5	-	**0 -> 1: 5**	**0 -> 5 -> 2: 6**	**0 -> 3: 6**	**0 -> 5 -> 2 -> 4: 3**	**0 -> 5: 7**



REZULTAT IZVOĐENJA ALGORITMA

Put od izvorišta 0 do čvora 0: 0: 0

Put od izvorišta 0 do čvora 1: 0 -> 1: 5

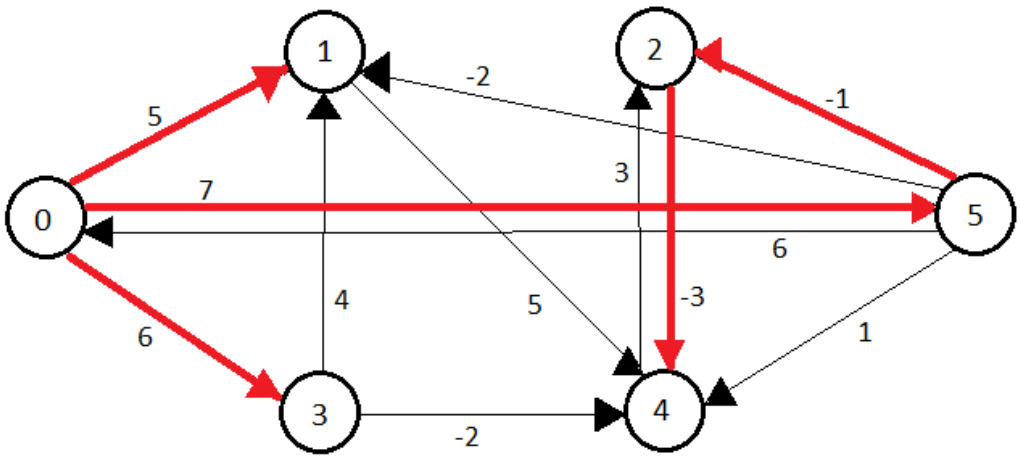
Put od izvorišta 0 do čvora 2: 0 -> 5 -> 2: 6

Put od izvorišta 0 do čvora 3: 0 -> 3: 6

Put od izvorišta 0 do čvora 4: 0 -> 5 -> 2 -> 4: 3

Put od izvorišta 0 do čvora 5: 0 -> 5: 7

Konačan graf najkraćeg puta:



JAVADOC

Bellman-Ford Javadoc

Package Summary		Page
hr.fer.tel.im.BellmanFord		22

Package hr.fer.tel.im.BellmanFord

Class Summary		Page
BellmanFord	Klasa BellmanFord sadrzi cijelu logiku algoritma i ispisuje sve korake.	22
CreateTopology	Klasa CreateTopology sadrzi samo konstruktor u kojem pozivamo java program run.jar: Program run.jar sluzi za generiranje .txt datoteke u kojoj se nalazi topologija.	27
Edge	Klasa u koju spremamo vrijednosti izvorisnog i odredisnog cvora, te tezinu izmedu njih.	28
Main		30
ReadTopology	U klasi ReadTopology ucitavamo generiranu topologiju	44

Class BellmanFord

hr.fer.tel.im.BellmanFord

```
java.lang.Object
  └ hr.fer.tel.im.BellmanFord.BellmanFord
```

```
public class BellmanFord
extends Object
```

Klasa BellmanFord sadrzi cijelu logiku algoritma i ispisuje sve korake.

Field Summary		Page
private int	**INF**	24

Constructor Summary		Page
	BellmanFord(ArrayList<Edge> edges, int source, int nnode, TextArea result) Konstruktor klase BellmanFord.	24

Method Summary		Page
private boolean	**checkNegativeEdge**(int[] distance) Metoda checkNegativeEdge provjerava da li postoji negativna udaljenost izmedu cvorova nakon V-1 iteracija.	24
private void	**iteration**(int nnode, int source, int[] distance, int[] parent, ArrayList<Edge> edges, TextArea result) Metoda iteration prolazi kroz sve cvorove u V-1 iteracija, te kroz bridove u broj cvorova - 1 iteracija.	24
private boolean	**negativCycle**(int[] distance, ArrayList<Edge> edges, TextArea result) Metoda negativeCycle prolazi jos jednom kroz sve cvorove, ako ustanovi da se neka od udaljenosti smanjila ispisuje se poruka o postojanju negativne petlje.	25
private void	**path**(int destination, int[] parent, int source, int[] distance, TextArea result) Metoda path racuna put od izvorisnog do ostalih cvorova.	25
private void	**printPath**(int source, int destination, int[] distance, ArrayList<Integer> rPath, TextArea result) Metoda printPath ispisuje put od izvorisnog do ostalih cvorova.	26
private void	**printResult**(int source, int[] distance, int[] parent, TextArea result) Metoda printResult ispisuje konacan rezultat izvodenja algoritma, ako ne postoje negativne petlje.	25

Field Detail

INF

```
private int INF
```

Constructor Detail

BellmanFord

```
BellmanFord(ArrayList<Edge> edges,
            int source,
            int nnode,
            TextArea result)
```

Konstruktor klase BellmanFord.

Parameters:

`edges` - polje objekata klase Edge
`source` - izvorisni cvor
`nnode` - ukupan broj cvorova u ucitanoj topologiji
`result` - textArea u koji ispisujemo rezultate izvodenja algoritma

Method Detail

iteration

```
private void iteration(int nnode,
                       int source,
                       int[] distance,
                       int[] parent,
                       ArrayList<Edge> edges,
                       TextArea result)
```

Metoda iteration prolazi kroz sve cvorove u V-1 iteracija, te kroz bridove u broj cvorova - 1 iteracija.

Parameters:

`nnode` - ukupan broj cvorova u ucitanoj topologiji
`source` - izvorisni cvor
`distance` - polje udaljenosti do svih cvorova u topologiji
`parent` - polje prethodnih cvorova
`edges` - polje objekata klase Edge
`result` - textArea u koji ispisujemo rezultate izvodenja algoritma

checkNegativeEdge

```
private boolean checkNegativeEdge(int[] distance)
```

Metoda checkNegativeEdge provjerava da li postoji negativna udaljenost izmedu cvorova nakon V-1 iteracija.

Parameters:

`distance` - polje udaljenosti do svih cvorova u topologiji

Returns:

negative - postavlja se na true ako postoji negativna udaljenost

---

negativCycle

```
private boolean negativCycle(int[] distance,
                             ArrayList<Edge> edges,
                             TextArea result)
```

Metoda negativeCycle prolazi jos jednom kroz sve cvorove, ako ustanovi da se neka od udaljenosti smanjila ispisuje se poruka o postojanju negativne petlje.

Parameters:

`distance` - polje udaljenosti do svih cvorova u topologiji
`edges` - polje objekata klase Edge
`result` - textArea u koji ispisujemo rezultate izvodenja algoritma

Returns:

ok - postavlja se na false ako postoji negativna petlja

---

printResult

```
private void printResult(int source,
                         int[] distance,
                         int[] parent,
                         TextArea result)
```

Metoda printResult ispisuje konacan rezultat izvodenja algoritma, ako ne postoje negativne petlje.

Parameters:

`source` - izvorisni cvor
`distance` - udaljenosti do svih cvorova u topologiji
`parent` - polje prethodnih cvora
`result` - textArea u koji ispisujemo rezultate izvodenja algoritma

---

path

```
private void path(int destination,
                  int[] parent,
                  int source,
                  int[] distance,
                  TextArea result)
```

Metoda path racuna put od izvorisnog do ostalih cvorova.

Parameters:

`destination` - odredisni cvor
`parent` - polje prethodnih cvorova
`source` - izvorisni cvor
`distance` - polje udaljenosti do svih cvorova u topologiji
`result` - textArea u koji ispisujemo rezultate izvodenja algoritma

---

printPath

```
private void printPath(int source,
                       int destination,
                       int[] distance,
                       ArrayList<Integer> rPath,
                       TextArea result)
```

Metoda printPath ispisuje put od izvorisnog do ostalih cvorova.

Parameters:

`source` - izvorisni cvor
`destination` - odredisni cvor
`distance` - polje udaljenosti do svih cvorova u topologiji
`rPath` - polje puta od izvorisnog do ostalih cvorova
`result` - textArea u koji ispisujemo rezultate izvodenja algoritma

Class CreateTopology

hr.fer.tel.im.BellmanFord

```
java.lang.Object
  └ hr.fer.tel.im.BellmanFord.CreateTopology
```

```
public class CreateTopology
extends Object
```

Klasa CreateTopology sadrzi samo konstruktor u kojem pozivamo java program run.jar: Program run.jar sluzi za generiranje .txt datoteke u kojoj se nalazi topologija.

Field Summary		Page
private Process	process	27

Constructor Summary		Page
	CreateTopology()	27

Field Detail

process

```
private Process process
```

Constructor Detail

CreateTopology

```
CreateTopology()
```

Class Edge

hr.fer.tel.im.BellmanFord

```
java.lang.Object
  └hr.fer.tel.im.BellmanFord.Edge
```

```
class Edge
extends Object
```

Klasa u koju spremamo vrijednosti izvorisnog i odredisnog cvora, te tezinu izmedu njih.

Field Summary		Page
int	destination	28
int	source	28
int	weight	28

Constructor Summary	Page
Edge(int s, int d, int w) Konstruktor preko kojeg prosljedujemo vrijednosti izvorisnog i odredisnog cvora, te tezinu izmedu njih.	29

Field Detail

source

```
int source
```

---

destination

```
int destination
```

---

weight

```
int weight
```

Constructor Detail

Edge

```
public Edge(int s,
            int d,
            int w)
```

Konstruktor preko kojeg prosljedujemo vrijednosti izvorisnog i odredisnog cvora, te tezinu izmedu njih.

Parameters:

`s` - izvorisni cvor
`d` - odredisni cvor
`w` - udaljenost izmedu izvorisnog i odredisnog cvora

Class Main

hr.fer.tel.im.BellmanFord

```
java.lang.Object
  └java.awt.Component
      └java.awt.Container
          └java.awt.Window
              └java.awt.Frame
                  └javax.swing.JFrame
                      └hr.fer.tel.im.BellmanFord.Main
```

All Implemented Interfaces:

Accessible, ImageObserver, MenuContainer, RootPaneContainer, Serializable, TransferHandler.HasGetTransferHandler, WindowConstants

---

```
public class Main
extends JFrame
```

Field Summary		Page
private JButton	btnCreateTopology	32
private JButton	btnReadTopology	32
private JButton	btnStartAlgoritham	32
private JPanel	buttonPanel	32
private static ArrayList<Edge>	edges	32
private JPanel	mainPanel	31
private static int	nnode	32
private static ReadTopology	obj	32
private static Component	parent	33
private TextArea	result	32
private JPanel	resultPanel	31
private static long	serialVersionUID	31
private static int	source	32
private JSplitPane	splitPane	33
private JSplitPane	splitPaneH	31
private JSplitPane	splitPaneV	31
private TextArea	topology	32
private JPanel	topologyPanel	31

Constructor Summary	Page
Main()	33

Method Summary		Page
void	createButtonPanel()	33
void	createMainPanel()	33
void	createResultPanel()	33
void	createTopologyPanel()	33
static void	main(String[] args)	33

Field Detail

serialVersionUID

```
private static final long serialVersionUID
```

---

splitPaneV

```
private JSplitPane splitPaneV
```

---

splitPaneH

```
private JSplitPane splitPaneH
```

---

mainPanel

```
private JPanel mainPanel
```

---

topologyPanel

```
private JPanel topologyPanel
```

---

resultPanel

```
private JPanel resultPanel
```

---

buttonPanel

```
private JPanel buttonPanel
```

---

result

```
private TextArea result
```

---

topology

```
private TextArea topology
```

---

btnCreateTopology

```
private JButton btnCreateTopology
```

---

btnReadTopology

```
private JButton btnReadTopology
```

---

btnStartAlgoritham

```
private JButton btnStartAlgoritham
```

---

edges

```
private static ArrayList<Edge> edges
```

---

obj

```
private static ReadTopology obj
```

---

source

```
private static int source
```

---

nnode

```
private static int nnode
```

parent

```
private static Component parent
```

---

splitPane

```
private JSplitPane splitPane
```

Constructor Detail

Main

```
public Main()
```

Method Detail

main

```
public static void main(String[] args)
```

---

createMainPanel

```
public void createMainPanel()
```

---

createTopologyPanel

```
public void createTopologyPanel()
```

---

createResultPanel

```
public void createResultPanel()
```

---

createButtonPanel

```
public void createButtonPanel()
```

Class ReadTopology

hr.fer.tel.im.BellmanFord

```
java.lang.Object
  └hr.fer.tel.im.BellmanFord.ReadTopology
```

```
public class ReadTopology
extends Object
```

U klasi ReadTopology ucitavamo generiranu topologiju

Field Summary		Page
private BufferedReader	bufferReader	35
private File	fileName	34
private FileReader	inputFile	34
private String	line	35
private int	nnode	34

Constructor Summary		Page
	ReadTopology(ArrayList<Edge> edges, String path) U konstruktoru ucitavamo generiranu topologiju, zatim ucitane podatke spremamo u varijable objekta Edge, te racunamo ukupan broj cvorova.	35

Method Summary		Page
int	getTopology() Metoda getTopology vraca broj cvorova ucitane topologije.	35

Field Detail

nnode

```
private int nnode
```

---

fileName

```
private File fileName
```

---

inputFile

```
private FileReader inputFile
```

---

bufferReader

```
private BufferedReader bufferReader
```

---

line

```
private String line
```

Constructor Detail

ReadTopology

```
ReadTopology(ArrayList<Edge> edges,
             String path)
```

U konstruktoru ucitavamo generiranu topologiju, zatim ucitane podatke spremamo u varijable objekta Edge, te racunamo ukupan broj cvorova.

Parameters:

`edges` - polje objekata klase Edge
`path` - apsolutni put do datoteke u kojoj se nalazi topologija koju zelimo ucitati

Method Detail

getTopology

```
public int getTopology()
```

Metoda getTopology vraca broj cvorova ucitane topologije.

Returns:

nnode - broj cvorova iz topologije

Dodatak

Izvorni kod rješenja

Main.java

```
package hr.fer.tel.im.BellmanFord;

import java.util.ArrayList;
import java.awt.*;
import java.awt.event.*;

import javax.swing.*;

public class Main extends JFrame {

	private static final long serialVersionUID = 1L;
	private JSplitPane splitPaneV;
	private JSplitPane splitPaneH;
	private JPanel mainPanel;
	private JPanel topologyPanel;
	private JPanel resultPanel;
	private JPanel buttonPanel;
	private TextArea result;
	private TextArea topology;
	private JButton btnCreateTopology;
	private JButton btnReadTopology;
	private JButton btnStartAlgoritham;

	private static ArrayList<Edge> edges = new ArrayList<Edge>();
	private static ReadTopology obj;
	private static int source;
	private static int nnode;
	private static Component parent;
	private JSplitPane splitPane;


	public static void main(String[] args) {
		new Main();
	}

	public Main() {
		setTitle("Bellman-Ford");

		JPanel panel = new JPanel();
		panel.setLayout(new BorderLayout());
		getContentPane().add(panel);

		createMainPanel();
		createTopologyPanel();
		createResultPanel();
		createButtonPanel();
```

```
        splitPaneV = new JSplitPane(JSplitPane.VERTICAL_SPLIT);
        panel.add(splitPaneV, BorderLayout.CENTER);

        splitPaneH = new JSplitPane(JSplitPane.HORIZONTAL_SPLIT);
        splitPaneH.setLeftComponent(mainPanel);

        splitPane = new JSplitPane();
        mainPanel.add(splitPane, BorderLayout.CENTER);
        splitPane.setRightComponent(resultPanel);

        splitPane.setLeftComponent(buttonPanel);

        splitPaneH.setRightComponent(topologyPanel);

        splitPaneV.setTopComponent(splitPaneH);

        setSize(700, 500);
        setLocationRelativeTo(null);
        setVisible(true);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }

    public void createMainPanel() {
        mainPanel = new JPanel();
        mainPanel.setLayout(new BorderLayout());
        mainPanel.setPreferredSize(new Dimension(500, 500));
    }

    public void createTopologyPanel() {
        topologyPanel = new JPanel();
        JLabel topologyLabel = new JLabel();
        topologyPanel.setLayout(new BorderLayout());
        topologyPanel.add(topologyLabel);

        topology = new TextArea();

        topologyLabel.add(topology);

        topology.setBounds(20, 20, 150, 420);

    }

    public void createResultPanel() {
        resultPanel = new JPanel();
        JLabel resultLabel = new JLabel();
        resultPanel.setLayout(new BorderLayout());
        resultPanel.add(resultLabel);

        JLabel textLabel = new JLabel(
                    "<html><b><font size = +1>Rezultat izvođenja
                      algoritma</font></b></html>");
        result = new TextArea();

        resultLabel.add(result);
        resultLabel.add(textLabel);

        result.setBounds(30, 70, 330, 360);
```

```
        textLabel.setBounds(30, 10, 300, 50);
    }

    public void createButtonPanel() {
        buttonPanel = new JPanel();
        JLabel buttonLabel = new JLabel();
        JLabel chooserLabel = new JLabel("Izaberite izvorište");
        buttonPanel.setLayout(new BorderLayout());
        buttonPanel.setPreferredSize(new Dimension(110, 500));
        buttonPanel.add(buttonLabel);
        final Choice sourceChoice = new Choice();

        btnCreateTopology = new JButton(
                    "<html><center>Kreiraj <br>
                      topologiju</center></html>");
        btnReadTopology = new JButton(
                    "<html><center>Učitaj <br>
                     topologiju</center></html>");
        btnStartAlgoritham = new JButton(
                    "<html><center>Pokreni <br>
                      algoritam</center></html>");

        buttonLabel.add(btnCreateTopology);
        buttonLabel.add(btnReadTopology);
        buttonLabel.add(btnStartAlgoritham);
        buttonLabel.add(chooserLabel);
        buttonLabel.add(sourceChoice);

        btnCreateTopology.setBounds(5, 10, 90, 40);
        btnReadTopology.setBounds(5, 80, 90, 40);
        btnStartAlgoritham.setBounds(5, 150, 90, 40);
        chooserLabel.setBounds(4, 220, 100, 20);
        sourceChoice.setBounds(4, 240, 100, 40);

        btnCreateTopology.addActionListener(new ActionListener() {
              @Override
              public void actionPerformed(ActionEvent e) {

                    new CreateTopology();
              }
        });

        btnReadTopology.addActionListener(new ActionListener() {
              @Override
              @SuppressWarnings("deprecation")
              public void actionPerformed(ActionEvent e) {
                    JFileChooser chooser = new JFileChooser();
                    chooser.showOpenDialog(parent);
                    String path =
                          chooser.getSelectedFile().getAbsolutePath();


                    obj = new ReadTopology(edges, path);

                    for (int i = 0; i < edges.size(); i++) {
                          topology.appendText(edges.get(i).source
                           + ", " + edges.get(i).destination
```

```
                                    + ", " + edges.get(i).weight + "\n");
                        }

                        nnode = obj.getTopology();
                        for (int i = 0; i < nnode; i++) {
                              sourceChoice.add(i + "\n");
                        }
                  }
            });

            btnStartAlgoritham.addActionListener(new ActionListener() {
                  @Override
                  public void actionPerformed(ActionEvent e) {

                        if (nnode > 0 && edges.size() > 0 && source <
                             nnode) {
                              new BellmanFord(edges, source, nnode,
                                          result);
                        }

                        if (source >= nnode) {
                              System.out.println("Vrijednost najvećeg
                                          čvora je " + (nnode - 1));
                        }
                        if (nnode <= 0) {
                              System.out.println("U učitanoj topologiji
                                          nema čvorova");
                        }
                        if (edges.size() <= 0) {
                              System.out.println("Učitan je nepovezan
                                    graf. Niti jedan čvor nije povezan!");
                        }
                  }
            });

            sourceChoice.addItemListener(new ItemListener() {
                  public void itemStateChanged(ItemEvent ie) {

                        //Izabiremo izvorišni čvor
                        source = sourceChoice.getSelectedIndex();
                  }
            });

      }
}
```

CrateTopology.java

```
package hr.fer.tel.im.BellmanFord;

import java.io.IOException;

public class CreateTopology {
    private Process process;

    CreateTopology() {
        try {

            System.out.println("###### UNOS MREŽNE TOPOLOGIJE ######");
            System.out.println("Kreirajte mrežnu topologiju");

            process = Runtime.getRuntime().exec("java -jar run.jar");

            System.out.println("Čekanje unosa mrežne toplogije ...");
            process.waitFor();
            System.out.println("Unos mrežne toplogije završen.\n");
        }

        catch (InterruptedException | IOException e) {
            System.out.println("Dogodila se pogreška prilikom kreiranja
                                topologije:" + e.getMessage());
        }
    }
}
```

ReadTopology.java

```
package hr.fer.tel.im.BellmanFord;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;

public class ReadTopology {
    private int nnode = 0;
    private File fileName;
    private FileReader inputFile;
    private BufferedReader bufferReader;
    private String line;

    ReadTopology(ArrayList<Edge> edges, String path) {

        System.out.println("###### UČITAVANJE KREIRANE MREŽNE TOPOLOGIJE
                           ######");
        System.out.println("Učitavanje kreirane topologije");
```

```
        try {
            fileName = new File(path);
            inputFile = new FileReader(fileName);

            bufferReader = new BufferedReader(inputFile);

            while ((line = bufferReader.readLine()) != null) {
                String[] var = line.split(", ");
                edges.add(new Edge(Integer.parseInt(var[0]),
                    Integer.parseInt(var[1]), Integer.parseInt(var[2])));

                if (nnode < Integer.parseInt(var[0])) {
                    nnode = Integer.parseInt(var[0]);
                }
                if (nnode < Integer.parseInt(var[1])) {
                    nnode = Integer.parseInt(var[1]);
                }
            }
            bufferReader.close();
            System.out.println("Topologija je učitana!\n");
        } catch (IOException e) {
            System.out.println("Dogodila se pogreška prilikom čitanja
                                datoteke:" + e.getMessage());
        }

        nnode = nnode + 1;
    }

    public int getTopology(){
        return nnode;
    }
}
```

Edge.java

```
package hr.fer.tel.im.BellmanFord;

class Edge {
    int source;
    int destination;
    int weight;

    public Edge(int s, int d, int w) {
        source = s;
        destination = d;
        weight = w;
    }
}
```

BellmanFord.java

```
package hr.fer.tel.im.BellmanFord;

import java.awt.TextArea;
import java.util.ArrayList;
import java.util.Arrays;

public class BellmanFord {
	private int INF = Integer.MAX_VALUE;

	BellmanFord(ArrayList<Edge> edges, int source, int nnode, TextArea
			result) {
		int[] distance = new int[nnode];
		Arrays.fill(distance, INF);
		int[] parent = new int[nnode];
		Arrays.fill(parent, source);
		distance[source] = 0;
		boolean negative;
		boolean ok = true;

		iteration(nnode, source, distance, parent, edges, result);

		negative = checkNegativeEdge(distance);

		if (negative) {
			ok = negativCycle(distance, edges, result);
		}

		if (ok) {
			printResult(source, distance, parent, result);
		}
	}

	@SuppressWarnings("deprecation")
	private void iteration(int nnode, int source, int[] distance, int[]
				parent, ArrayList<Edge> edges, TextArea result) {
		for (int i = 1; i <= nnode - 1; i++) {
			result.appendText("\n#####Korak " + i + ".
					iteracije#####\n");
			for (int j = 0; j < edges.size(); j++) {
				if (distance[edges.get(j).source] == INF) {
					continue;
				}

				int newDistance = distance[edges.get(j).source]
						+ edges.get(j).weight;

				if (newDistance <
				    distance[edges.get(j).destination]) {
					distance[edges.get(j).destination] =
							newDistance;
					parent[edges.get(j).destination] =
							edges.get(j).source;
				}
			}
```

```
            for(int j = 0; j <= nnode - 1; j++){
                  path(j, parent, source, distance, result);
            }

            System.out.println("\n");
      }
}

private boolean checkNegativeEdge(int[] distance) {
      boolean negative = false;

      for (int i = 0; i < distance.length; i++) {
            if (distance[i] < 0) {
                  negative = true;
            }
      }

      return negative;
}

@SuppressWarnings("deprecation")
private boolean negativCycle(int[] distance, ArrayList<Edge> edges,
            TextArea result) {
      boolean ok = true;

      for (int j = 0; j < edges.size(); j++) {
            if (distance[edges.get(j).source] != INF
               && distance[edges.get(j).destination] >
              distance[edges.get(j).source] + edges.get(j).weight){
                  result.appendText("\nOtkrivena petlja negativne
                                    težine!\n");
                  ok = false;
                  break;
            }
      }

      return ok;
}

@SuppressWarnings("deprecation")
private void printResult(int source, int[] distance, int[] parent,
            TextArea result) {
      result.appendText("\n##### REZULTAT IZVODENJA ALGORITMA
                        #####\n");

      for (int i = 0; i < distance.length; i++) {
            path(i, parent, source, distance, result);
      }
}

private void path(int destination, int[] parent, int source, int[]
                  distance, TextArea result) {
      ArrayList<Integer> rPath = new ArrayList<Integer>();
      rPath.add(destination);
```

```
            if (destination != source) {
                  int par = parent[destination];
                  int br = 0;

                  rPath.add(par);

                  while (par != source) {
                if(br != distance.length){
                    par = parent[par];
                    rPath.add(par);
                    br++;
                }
                else{
                    rPath.add(source);
                    break;
                }
            }

            }
            printPath(source, destination, distance, rPath, result);
      }

      @SuppressWarnings("deprecation")
      private void printPath(int source, int destination, int[] distance,
                  ArrayList<Integer> rPath, TextArea result) {
            result.appendText("Put od izvorišta " + source + " do čvora "
                              + destination + ": \n");
            for (int i = rPath.size() - 1; i >= 0; i--) {
                  result.appendText(rPath.get(i) + "");
                  if (i != 0) {
                        result.appendText(" -> ");
                  }
                  if (i == 0) {
                        if (distance[destination] == INF) {
                              result.appendText(": \u221E\t \n");
                        } else {
                              result.appendText(": " +
                                          distance[destination] + "\n");
                        }
                  }
            }
      }
}
```