ELEKTROTEHNIČKI FAKULTET UNIVERZITETA U BEOGRADU

JAVA APLIKACIJA ZA NALAŽENJE NAJKRAĆEG PUTA U MREŽI

– Diplomski rad –

Kandidat:

Vladimir Mihajlović 2008/187

Mentor:

doc. dr Zoran Čiča

Beograd, Oktobar 2015.

SADRŽAJ

SADRŽAJ **2**

1. UVOD **3**

2. ALGORITMI ZA NALAŽENJE NAJKRAĆEG PUTA **4**

2.1. DIJKSTRA ALGORITAM 4
2.2. FLOYD-WARSHALL ALGORITAM 5

3. JAVA APLIKACIJA **7**

3.1. RAZVOJNO OKRUŽENJE 7
3.2. REALIZACIJA ALGORITAMA U JAVI 10
3.2.1. Dijkstra algoritam *10*
3.2.2. Floyd-Warshall algoritam *12*

4. PRIMERI **13**

4.1. PRIMER 1 13
4.1.1. Grafičko rešenje primenom Dijkstra algoritma *13*
4.1.2. Rešenje u Java aplikaciji primenom Dijksra algoritma *17*
4.1.3. Rešenje u Java aplikaciji primenom Floyd-Warshall algoritma *17*
4.2. PRIMER 2 19
4.2.1. Grafičko rešenje primenom Dijkstra algoritma *19*
4.2.2. Rešenje u Java aplikaciji primenom Dijksra algoritma *22*
4.2.3. Rešenje u Java aplikaciji primenom Floyd-Warshall algoritma *23*
4.3. PRIMER 3 23
4.3.1. Grafičko rešenje primenom Dijkstra algoritma *23*
4.3.2. Rešenje u Java aplikaciji primenom Dijksra algoritma *24*
4.3.3. Rešenje u Java aplikaciji primenom Floyd-Warshall algoritma *25*

5. ZAKLJUČAK **26**

LITERATURA **27**

1. UVOD

Veliki broj problema optimizacije na realnim mrežama, kao što su električne, telekomunikacione, putne, može da se modelira nekim od karakterističnih problema na grafovima za koje postoje razvijeni efikasni algoritmi. Problem nalaženja najkraćeg puta u mreži je jedan od osnovnih i najjednostavnijih optimizacionih problema.

U ovom radu će biti realizovana dva algoritma za nalaženje najkraćeg puta, a to su: Dijkstra algoritam, Floyd-Warshall algoritam. Implementacija ovih algoritama obavljena je korišćenjem programskog jezika Java.

Teza se sastoji od 5 poglavlja:

- Prvo poglavlje predstavlja uvod
- U drugom poglavlju ćemo se upoznati sa algoritmima kroz teorijsko objašnjenje ovih algoritma za nalaženje najkraćeg puta;
- U trećem poglavlju opisana je instalacija alata potrebnih za pokretanje same aplikacije, kao i programski kod za realizaciju samih algoritama;
- Četvrto poglavlje rezervisano je za primere u kojima ćemo uporediti grafička rešenja i rešenja dobijena primenom same aplikacije;
- U petom poglavlju dat je zaključak teze.

2. ALGORITMI ZA NALAŽENJE NAJKRAĆEG PUTA

U ovome poglavlju biće data teorijska objašnjenja algoritama za nalaženje najkraćeg puta kroz mrežu korišćenih u ovoj tezi (Dijkstra, Floyd-Warshall algoritmi).

2.1. Dijkstra algoritam

Ovaj algoritam se smatra najefikasnijim za određivanje najkraćeg puta između dva čvora kada je ispunjen uslov da su dužine grana pozitivne, $c_{ij} > 0$, za svako $(i, j) \in L$.

Ideja algoritma i dokaz da se njime dolazi do optimalnog rešenja zasnivaju se na principu optimalnosti koji se za ovu priliku pojednostavljeno izražava stavom: optimalni put je i u delovima optimalan. U algoritmu se polazi od početnog čvora i iterativno približava krajnjem pri čemu se vodi računa da put kojim se ide zadovoljava princip optimalnosti.

U prvoj iteraciji se određuje čvor koji je najbliži početnom. U drugoj iteraciji treba odrediti čvor koji je sledeći po redu (drugi) najbliži čvor početnom čvoru. Do njega se može stići ili direktnom granom od početnog čvora ili preko čvora za koji je u prethodnoj iteraciji utvrđeno da je najbliži početnom. Na taj način su određeni najkraći putevi od početnog do sledećeg čvora u mreži. Dalje se proširuje skup čvorova do kojih su određeni najkraći putevi koristeći sledeće opšte pravilo: do nekog čvora se najkraćim putem dolazi ili direktno od početnog čvora ili preko nekog drugog čvora za koji je već određen najkraći put. Postupak se završava kada se utvrdi najkraći put do krajnjeg čvora.

U svrhu implementacije principa optimalnosti u Dijkstrinom algoritmu se koristi koncept obeležavanja čvorova. Obeležje ili oznaka $\mathrm{D}(j)$ čvora j može biti privremeno (promenljivo) i piše se $D^-(j)$ ili stalno (nepromenljivo), kada se piše $D^+(j)$. Ovo poslednje predstavlja dužinu najkraćeg puta od početnog čvora do čvora j.

Algoritam [1]:

1) Inicijalizacija

Čvorovima dodeljujemo početna obeležja na sledeći način:

- početnom čvoru s dodeljujemo stalno obeležje $D^+(s) = 0$;
- svim ostalim čvorovima dodeljujemo privremena obeležja $D^-(j) = \infty, j \in N \backslash \{s\}$;
- stavimo da je $i = s$.

2) Odrediti skup A_i čvorova koji slede čvor i i koji nemaju stalno obeležje

$$A_i = \{j \mid j \in \Gamma(i) \wedge D(j) = D^-(j)\}.$$

3) Za svako $j \in A_i$ odrediti nova privremena obeležja

$$\mathrm{D}^-(j) = min\{d^-(j), \quad D^+(i) + c_{ij}\}$$

4) Od svih čvorova na mreži koji su obeleženi privremenim obeležjem, samo jedan j^* dobija stalno obeležje i to onaj za koji je

$$D(j^*) = min_{j \in N}\{D^-(j)\},$$

pa je $D^+(j^*) = D^-(j^*)$.

5) Proveriti da li je $j^* = t$, tj. da li je završni čvor obeležen stalnim obeležjem.

Ako nije, staviti $i = j^*$ i vratiti se na korak 2).

Ako jeste, određena je dužina najkraćeg puta $D(t)$ i sada treba rekonstruisati najkraći put kojim se od s stiglo do t.

6) Najkraći put $p = (s, j_1, j_2, \dots, j_k, t)$ određujemo vraćajući se unazad od čvora t ka čvoru s: $t \longrightarrow j_k \longrightarrow j_{k-1} \longrightarrow \cdots \rightarrow s$, tako da važi:

$$\begin{aligned} j_k: &\quad D^+(t) - D^+(j_k) = c_{j_k t} \\ j_{k-1}: &\quad D^+(j_k) - D^+(j_{k-1}) = c_{j_{k-1} j_k} \\ &\quad \vdots \\ s: &\quad D^+(j_i) - D^+(s) = c_{s j_1} \end{aligned}$$

Jasno je da važi: $D^+(t) = c_{s j_1} + c_{j_1 j_2} + \dots + c_{j_{k-1} j_k} + c_{j_k t}$.

Napomena: Poslednji korak algoritma, određivanje najkraćeg puta, pojednostavljuje se ako se u koraku 4), pored stalnog obeležja, zapamti i indeks čvora koji prethodi posmatranom čvoru i na osnovu kojeg je dobijeno to obeležje.

2.2. Floyd-Warshall algoritam

Definisaćemo matricu rastojanja čvorova $C = (c_{ij})$ čiji elementi c_{ij} predstavljaju dužinu grana između čvorova i i j. Uvešćemo sledeće pretpostavke o dužinama grana, tj. o vrednostima elemenata matrice C:

1) $c_{ij} \neq 0 \; \forall \, (i,j) \in L$
2) $c_{ii} = 0 \; \forall \, i \in N$
3) $c_{ij} = \infty \; \forall \, (i, \; j) \notin L, i \neq j.$

U Floyd-Warshall algoritmu se koristi kvadratna matrica $C^k = (c_{ij}^k)_{n \times n}$, $k = 1, 2, \; \dots \; , n$, čiji elementi c_{ij}^k predstavljaju dužine najkraćih puteva između čvorova i i j, ali samo za $i, j \in \{1, 2, \dots, k\}$. Kada je $k = n$, tada će matrica C^n predstavljati dužine najkraćih puteva između svaka dva čvora u mreži.

Floyd-Warshall algoritam se može primeniti i kada su dužine grana negativne, ali u mreži ne sme da postoji kontura negativne dužine. Zato grane sa negativnim dužinama moraju biti orijentisane.

Algoritam [1]:

1) Postaviti $k = 0$ i formirati matricu C^0 takvu da je $C^0 = C$ tj. $c_{ij}^0 = c_{ij}$,

$$\forall \, i, j \; = 1, 2, \dots \, , n.$$

2) $k = k + 1$;
3) Odrediti skupove I i J takve da:

$$I = \left\{ i \middle| \, i \neq k \, , c_{ik}^k \neq \infty \right\}$$
$$J = \left\{ j \middle| \, j \neq k \, , c_{kj}^k \neq \; \infty \right\}$$

4) Izračunati elemente matrice C^k po formuli:

$$c_{ij}^k = min\{c_{ij}^{k-1}, c_{ik}^{k-1} + c_{kj}^{k-1}\} \ \forall\, i \in I \text{ i } \forall\, j \in J\ ,$$

dok ostali elementi ostaju nepromenjeni.

5) Mogu da nastupe tri slučaja:

- ako je $k < n$ i $c_{ij}^k \geq 0\ \forall\ i\ \in N$, ići na korak 2);
- ako je $k \leq n$ i $\exists\ i\ \in N : c_{ii}^k < 0$, tada u mreži postoji kontura negativne dužine i nije moguće dobiti konačno rešenje;
- ako je $k = n$ i $c_{ii}^n \geq 0\ \forall\ i\ \in N \rightarrow$ KRAJ;

Matrica C^n predstavlja rešenje problema.

3. JAVA APLIKACIJA

U prvom potpoglavlju ovog poglavlja biće dato uputstvo za instalaciju JDK (*Java SE Development Kit*), gde je SE oznaka za *Standard Edition*, Eclipse razvojnog okruženja, kao i uputstvo za pokretanje same aplikacije. U drugom potpoglavlju će biti dat deo programskog koda koji se odnosi na realizaciju algoritama za nalaženje najkraćeg puta u Javi.

3.1. Razvojno okruženje

Java je programski jezik koji se koristi za izradu i razvoj velikog broja aplikacija i njihovu implementaciju u najrazličitija multiplatformska okruženja. Java programiranje je jedno od najzastupljenijih danas, posebno imajući u vidu sve veću primenu u izradi aplikacija za mobilne telefone i tablet uređaje kroz Android operativni sistem. Java platforme imaju široku upotrebu u programiranju i primenjenim softverskim rešenjima, koje se kreću od najjednostavnijih digitalnih uređaja, mp3 plejera, mobilnih telefona, pa sve do kompleksnih veb servera i korporativnih aplikacija.

Postupak instalacije: [2]

Prvo treba daunloudovati Java SE Development Kit 8 (JDK8u60) sa sajta http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html

Selektovati opciju *Accept License Agreement* i zatim izabrati odgovarajući JDK za vaš operativni sistem (Slika 3.1.1.). Vodite računa da li je vaš operativni sistem 32-bitni (oznaka x86) ili 64-bitni (oznaka x64) .

Java SE Development Kit 8u60

You must accept the Oracle Binary Code License Agreement for Java SE to download this software.

Accept License Agreement Decline License Agreement

Product / File Description	File Size	Download
Linux ARM v6/v7 Hard Float ABI	77.69 MB	jdk-8u60-linux-arm32-vfp-hflt.tar.gz
Linux ARM v8 Hard Float ABI	74.64 MB	jdk-8u60-linux-arm64-vfp-hflt.tar.gz
Linux x86	154.66 MB	jdk-8u60-linux-i586.rpm
Linux x86	174.83 MB	jdk-8u60-linux-i586.tar.gz
Linux x64	152.67 MB	jdk-8u60-linux-x64.rpm
Linux x64	172.84 MB	jdk-8u60-linux-x64.tar.gz
Mac OS X x64	227.07 MB	jdk-8u60-macosx-x64.dmg
Solaris SPARC 64-bit (SVR4 package)	139.67 MB	jdk-8u60-solaris-sparcv9.tar.Z
Solaris SPARC 64-bit	99.02 MB	jdk-8u60-solaris-sparcv9.tar.gz
Solaris x64 (SVR4 package)	140.18 MB	jdk-8u60-solaris-x64.tar.Z
Solaris x64	96.71 MB	jdk-8u60-solaris-x64.tar.gz
Windows x86	180.82 MB	jdk-8u60-windows-i586.exe
Windows x64	186.16 MB	jdk-8u60-windows-x64.exe

Java SE Development Kit 8u60 Demos and Samples Downloads

You must accept the Oracle BSD License. to download this software.

Accept License Agreement Decline License Agreement

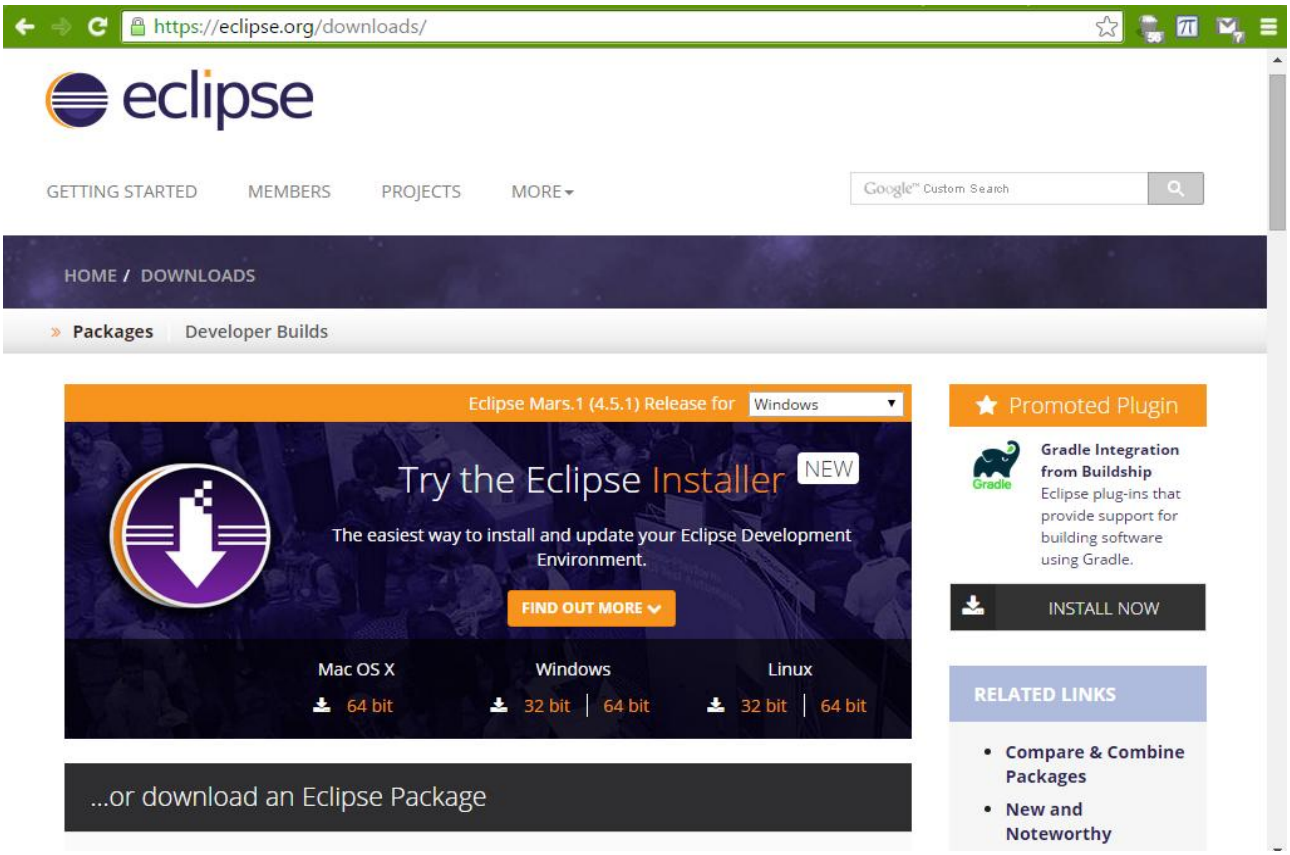
Product / File Description	File Size	Download
Linux ARM v6/v7 Hard Float ABI	9.95 MB	jdk-8u60-linux-arm32-vfp-hflt-demos.tar.gz
Linux ARM v8 Hard Float ABI	9.94 MB	jdk-8u60-linux-arm64-vfp-hflt-demos.tar.gz
Linux x86	52.65 MB	jdk-8u60-linux-i586-demos.rpm
Linux x86	52.51 MB	jdk-8u60-linux-i586-demos.tar.gz
Linux x64	52.7 MB	jdk-8u60-linux-x64-demos.rpm
Linux x64	52.56 MB	jdk-8u60-linux-x64-demos.tar.gz
Mac OS X	53.08 MB	jdk-8u60-macosx-x86_64-demos.zip
Solaris SPARC 64-bit	13.57 MB	jdk-8u60-solaris-sparcv9-demos.tar.Z
Solaris SPARC 64-bit	9.33 MB	jdk-8u60-solaris-sparcv9-demos.tar.gz
Solaris x64	13.57 MB	jdk-8u60-solaris-x64-demos.tar.Z
Solaris x64	9.29 MB	jdk-8u60-solaris-x64-demos.tar.gz
Windows x86	54.18 MB	jdk-8u60-windows-i586-demos.zip
Windows x64	54.28 MB	jdk-8u60-windows-x64-demos.zip

Slika 3.1.1. Izgled stranice za izbor odgovarajućeg JDK

Ako već imate inastaliranu Javu na računaru, preporučuje se da je prvo deinstalirate (opcija Add/Remove programs ili nešto slično). Po završetku daunlouda, instalirati JDK na vaš računar.

Ako želite, možete daunloudovati i primere (*Java SE Development Kit 8u60 Demos and Samples*) sa iste stranice (Slika 3.1.1.).

Zatim, daunloudovati *Eclipse*, sa sajta https://eclipse.org/downloads/ . Preporučuje se verzija *Eclipse IDE for Java Developers (Eclipse Mars (4.5))*



Slika 3.1.2. Izgled prozora za dounloud Eclipse Mars

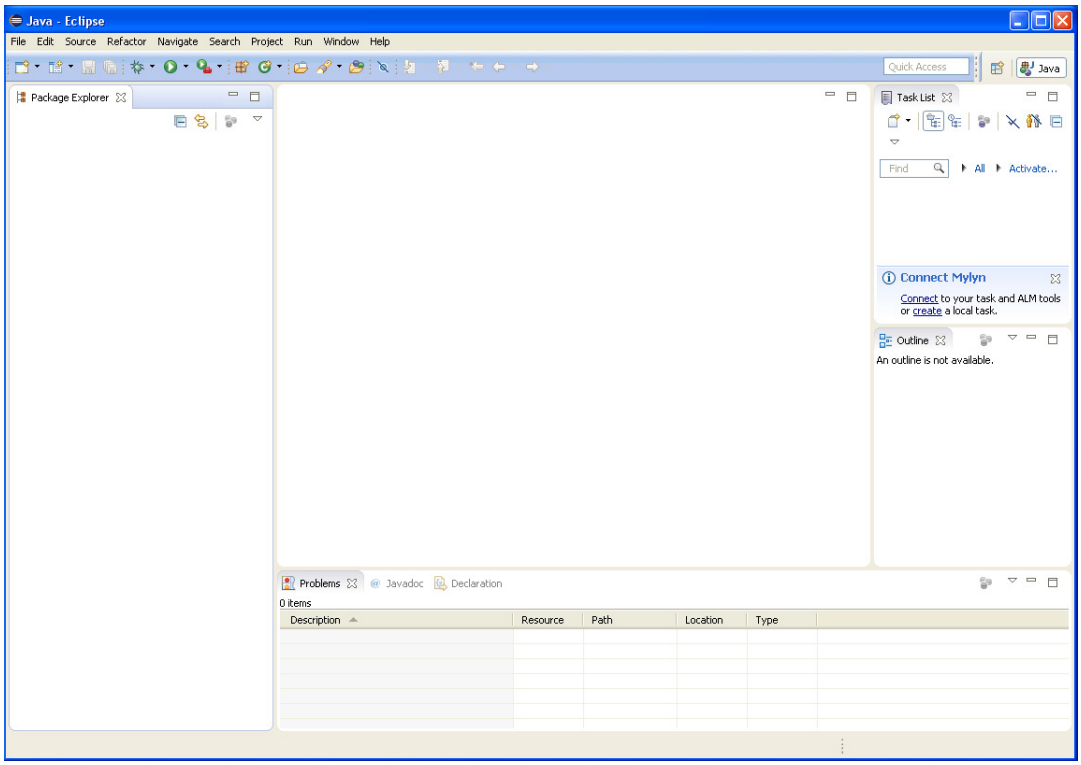
Izaberite iz padajuće liste (izgled prozora možete videti na Slici 3.1.2), "*Eclipse Mars.1 (4.5.1) Release for*" vaš operativni sistem, i vodite računa da li je vaš operativni sistem 32-bitni ili 64-bitni.

Eclipse ne zahteva instalaciju već se samo otpakuje u željeni folder.

Pokretanje Eclipse – dvostruki klik na eclipse.exe.

Po pokretanju alata, traži se da unesete lokaciju za vaš radni prostor (*workspace*). Možete imati više radnih prostora koji su smešteni u različitim folderima ili diskovima vašeg računara. Sačekajte da se učita Eclipce i u novootvorenom prozoru kliknite na *Workbench*.

Tipičan izgled radnog okruženja dat je na Slici 3.1.3. :



Slika 3.1.3. Izgled radnog okruženja

Otvaranje aplikacije:

U polju *Package Explorer* (nalazi se sa leve strane na Slici 3.1.3.) kliknite desnim klikom, pa izaberite opciju *Import*. U novootvorenom prozoru izaberite *General*, pa *Existing Projects into Workspace*. Zatim klikom na *Browse* odaberite putanju do projekta koji hoćete da otvorite i kliknite na *Finish*.

U *src* folderu se nalaze sva 4 programa. Za pokretanje aplikacije potrebno je 2 puta kliknuti na željeni algoritam i odabrati dugme *Run*.

3.2. Realizacija algoritama u Javi

3.2.1. Dijkstra algoritam

Programski kod za realizaciju Dijkstra algoritma:

```
import java.util.Arrays;

public class Dijkstra
{
    public static void main(String[] args)
    {
      //tezinska matrica koja predstavlja graf se menja ovde
        int[][] matrica =  {{0,3,10,0,0,0,0,0,0},
                            {0,0,0,7,3,0,0,0,0},
                            {0,0,0,0,0,8,0,0,0},
                            {0,0,0,0,6,0,4,2,3},
                            {0,0,2,0,0,0,0,7,0},
                            {0,0,0,0,5,0,0,6,0},
                            {0,0,0,0,0,3,0,0,4},
                            {0,0,0,0,0,0,0,0,5},
                            {0,0,0,0,0,0,0,0,0}};
        //ovde se menjaju pocetni i cvor koji zelite da posetite
        int pocetni = 1;
        int odredisni = 6;
        int[] prethodni = new int[matrica.length];

             for(int i=0; i<prethodni.length; i++)
                  prethodni[i] = -1;

        int[] rastojanje = new int[matrica.length];
        Arrays.fill(rastojanje,-1);
        int brojPosecenih = 0;
        boolean[] posecen = new boolean[matrica.length];


        int trenutni = pocetni-1;
        rastojanje[trenutni] = 0;
        posecen[trenutni] = true;
        brojPosecenih++;

        //dok ne posetimo sve cvorove trazimo minimalno rastojanje
        while(brojPosecenih!=matrica.length)
        {

                  int min = -1;
            for(int i=0;i<matrica.length;i++)
                if(matrica[trenutni][i]!=0 && !posecen[i])
                {
                if(rastojanje[i] == -1 || rastojanje[i] >
rastojanje[trenutni]+matrica[trenutni][i]){
                                          rastojanje[i] =
rastojanje[trenutni]+matrica[trenutni][i];
      prethodni[i]=trenutni;
            min = i;
                                  }
                }
```

```
            for(int i=0;i<matrica.length;i++)
            {
                if(!posecen[i])
                {
                    if(min==-1)
                        min = i;
                   else  if(rastojanje[i] != -1 & rastojanje[i]<rastojanje[min])
                        min = i;
                }
            }

            trenutni = min;
            posecen[trenutni] = true;
            brojPosecenih++;
        }
        //ispis rastojanja
        System.out.println("Sva rastojanja od pocetnog cvora: ");
        for(int i=1;i<=matrica.length;i++)
            System.out.println("Rastojanje od " + pocetni + " do " + i + " je: "
+ rastojanje[i-1]);

        System.out.println("Rastojanje do odabranog cvora " + odredisni + " je:
" + rastojanje[odredisni-1]);

        //ispis putanje
        int x =odredisni - 1;

             String ispis = "Putanja je: " + odredisni + " <- ";
        while(prethodni[x] != pocetni-1 && prethodni[x] != -1)
        {
            ispis += (prethodni[x]+1) + " <- ";
            x = prethodni[x];
        }
             if(prethodni[x] == -1)

                    ispis = "Nema puta izmedju " + pocetni + " i " + odredisni;
        else
                    ispis += pocetni;
            System.out.println(ispis);
    }
}
```

3.2.2. Floyd-Warshall algoritam

Programski kod za realizaciju Floyd-Warshall algoritma:

```
public class FloydWarshal {

	public static void main(String[] args){

		final int INFINITY = 9999;
//ovde se menja tezinska matrica koja predstavlja graf
		int[][] matrica = {
				{0, 2, INFINITY, INFINITY, 1, INFINITY, 3},
				{2, 0, 3, INFINITY, 2, INFINITY, 5},
				{INFINITY, 3, 0, 5, 3, 1, 4},
				{INFINITY, INFINITY, 5, 0, INFINITY, 2, 5},
				{1, 2, 3, INFINITY, 0, 1, INFINITY},
				{INFINITY, INFINITY, 1, 2, 1, 0, INFINITY},
				{3, 5, 4, 5, INFINITY, INFINITY, 0} };


//pozivamo funkciju i prosledjujemo joj ovu tezinsku matricu sto smo uneli
		Floyd(matrica);
	}

public static void Floyd(int[][] matrica){
	int minimum = 0;
//uvedemo pomocnu promenljivu k i prolazimo kroz celu matricu trazeci minimum

		for(int k=0; k<matrica.length; k++){
			for(int i=0; i<matrica.length; i++){
				for(int j=0; j<matrica.length; j++){
					minimum =  matrica[i][k]+matrica[k][j];
					if( minimum < matrica[i][j]){
						matrica[i][j] = minimum;}
				}
			}
		}
//ispisujemo matricu rastojanja
	System.out.println("Matrica rastojanja: ");
		for(int[] red: matrica){
			for(int kolona: red){
				System.out.print(kolona +"\t");
			}
			System.out.println();}
			}
	}
```

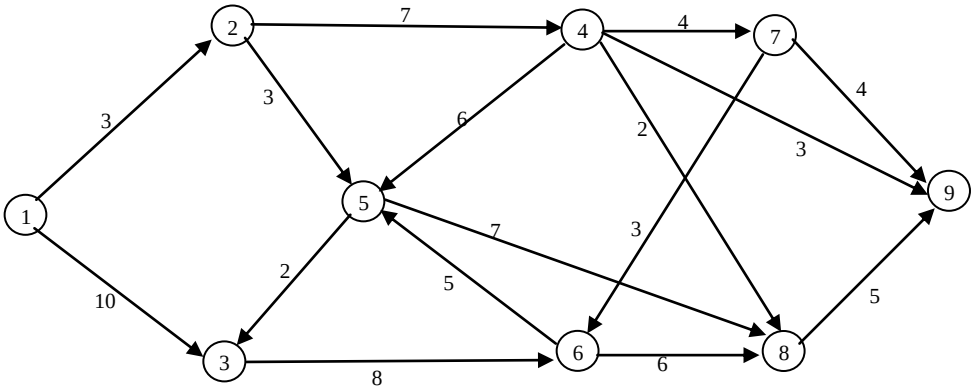
4. PRIMERI

U ovome poglavlju ćemo kroz niz primera pokušati da uporedimo rad aplikacije sa grafičkim rešavanjem problema nalaženja najkraćeg puta kroz mrežu.

4.1. Primer 1

4.1.1. Grafičko rešenje primenom Dijkstra algoritma

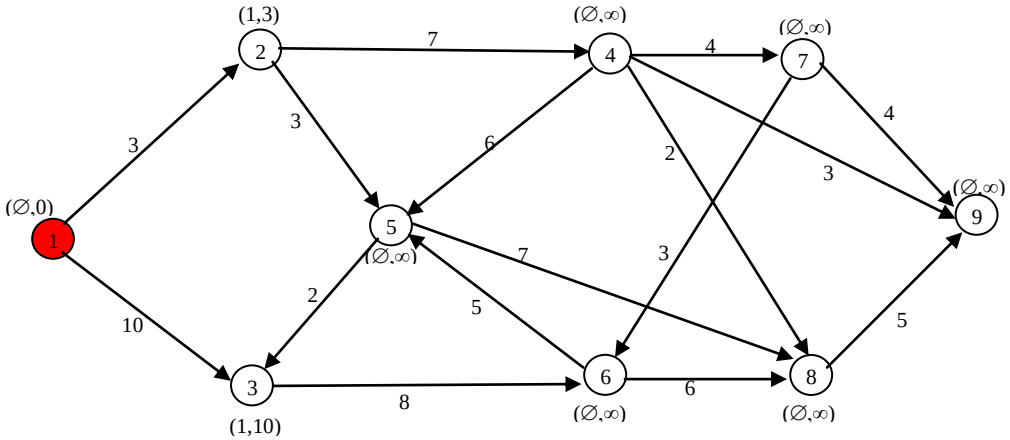
Za ispitivanje koda algoritma ćemo koristiti primer iz izvora [3] koji je prikazan u nastavku. Primer ćemo iskoristiti za poređenje rezultata aplikacije sa očekivanim rezultatom.



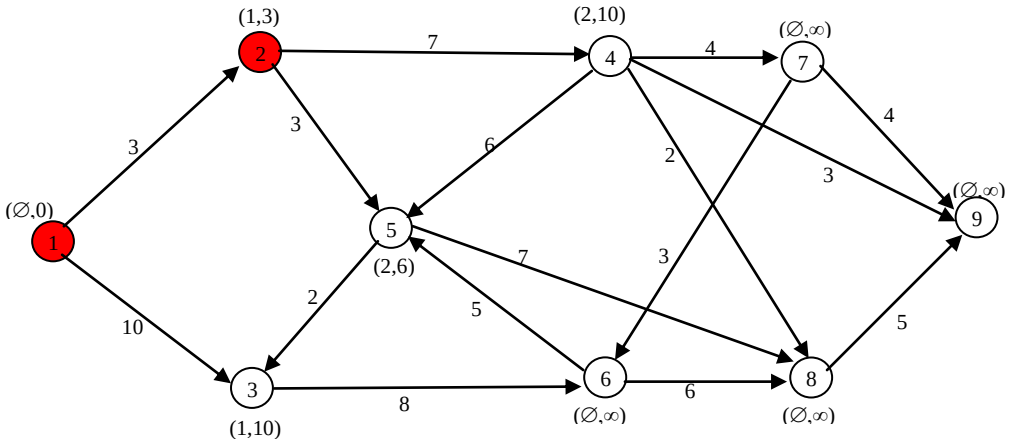
Slika 4.1.1.1. Graf za koji treba naći najkraći put

Rešenje:

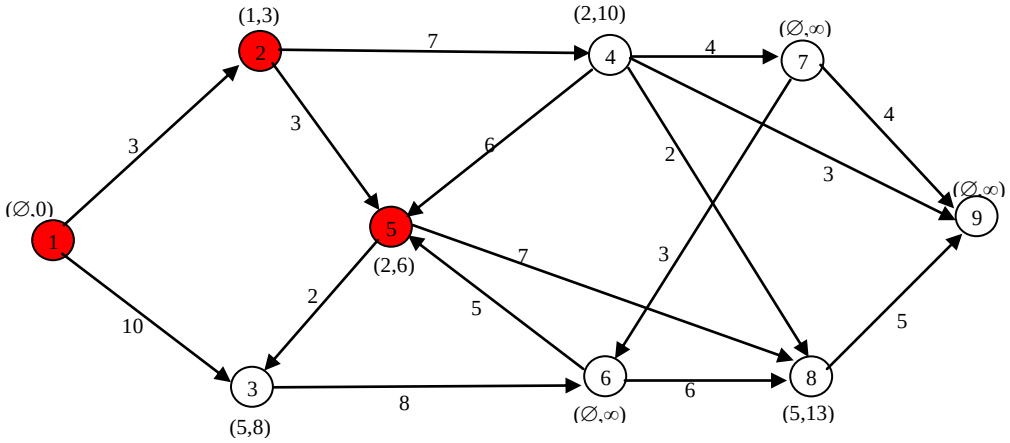
Posećeni čvorovi su obojeni odgovarajućom bojom radi lakšeg praćenja.



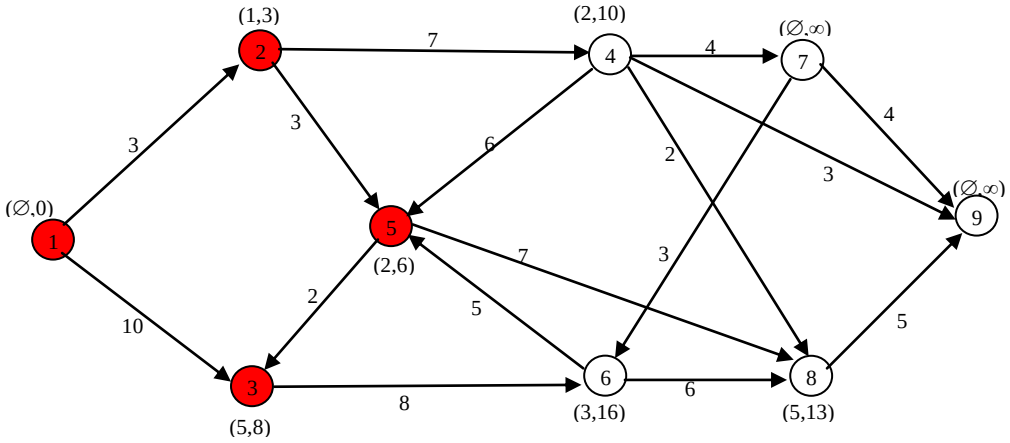
Slika 4.1.1.2. Označen je početni čvor grafa



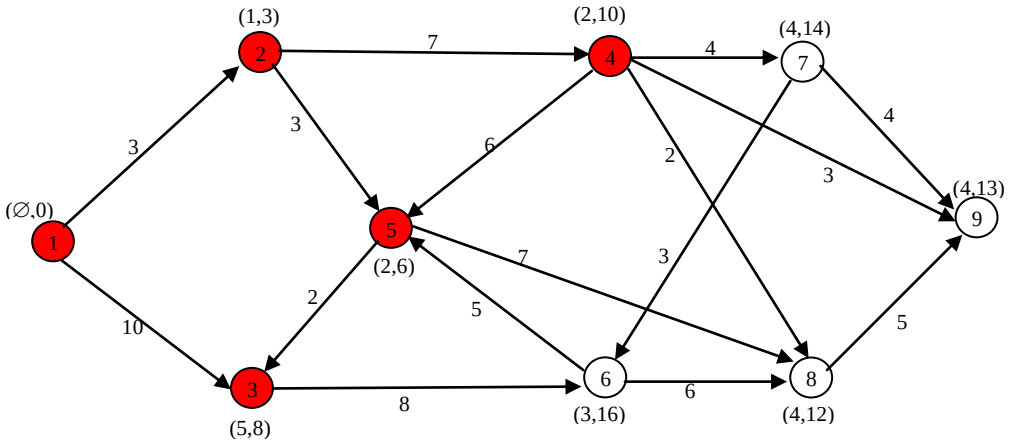
Slika 4.1.1.3. Slika sa izabranim čvorom 2 kao čvorom sa najkraćom putanjom do početnog čvora



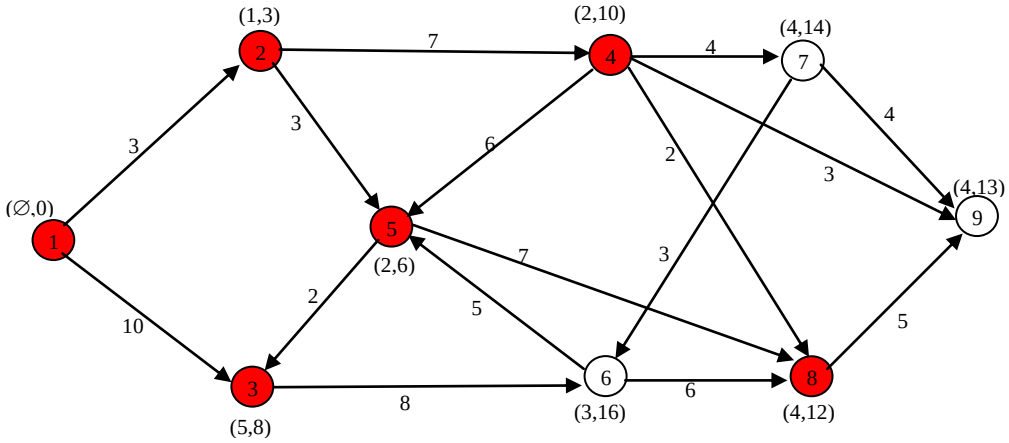
Slika 4.1.1.4. Slika sa označenim trećim čvorom po redu



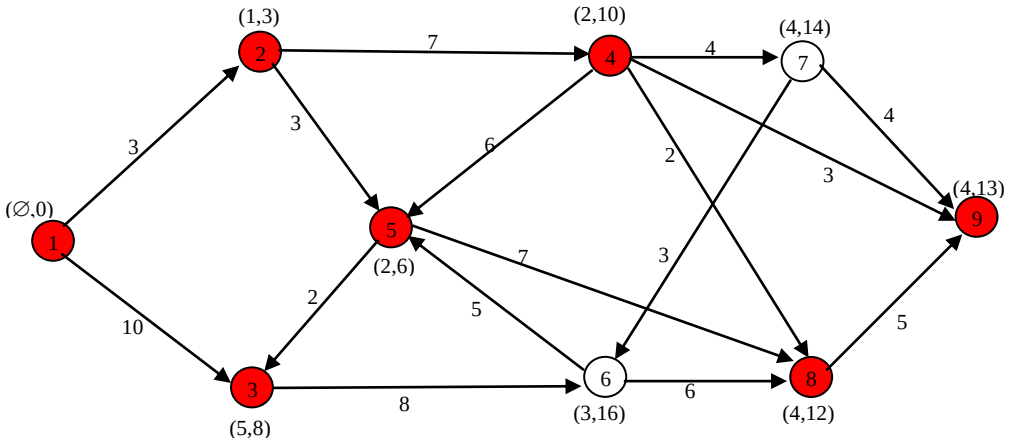
Slika 4.1.1.5 Slika sa označenim četvrtim čvorom po redu



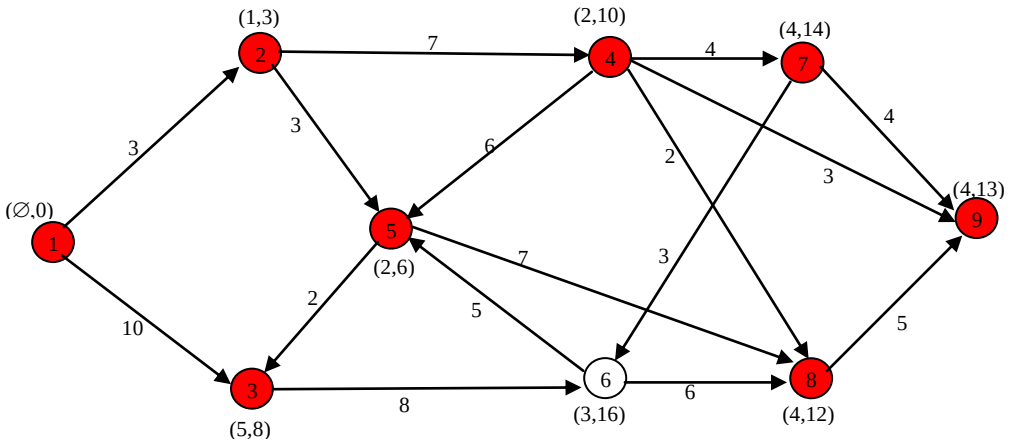
Slika 4.1.1.6. Slika sa označenim 5. čvorom po redu



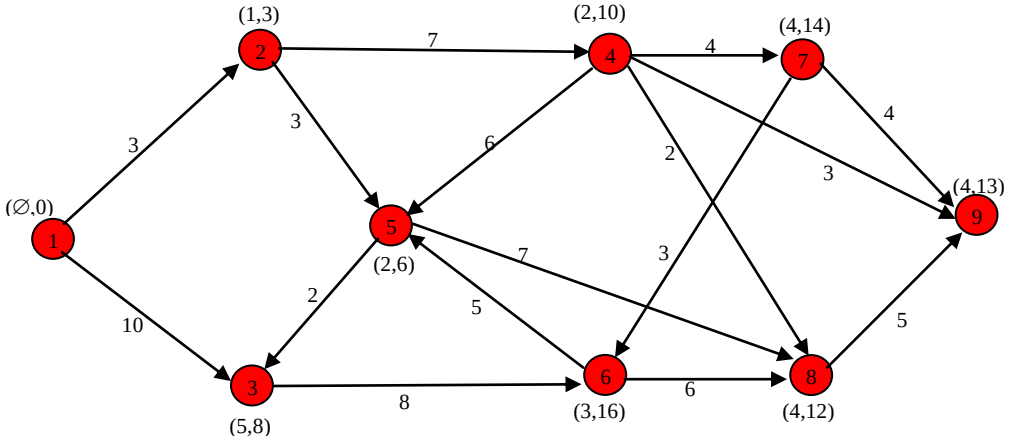
Slika 4.1.1.7. Slika sa označenim 6. čvorom po redu



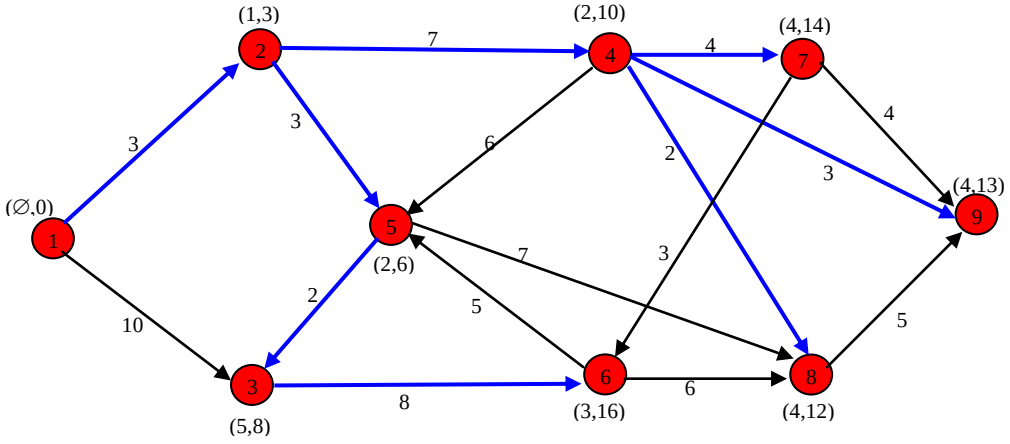
Slika 4.1.1.8. Slika sa označenim 7. čvorom po redu



Slika 4.1.1.9. Slika sa označenim 8. čvorom po redu



Slika 4.1.1.10. Slika sa označenim 9. čvorom po redu



Slika 4.1.1.11. Konačno rešenje sa prikazanim rastojanjima (drugi broj u zagradi pored čvora) svih čvorova u odnosu na početni čvor (1) , kao i najkraćim putanjama

4.1.2. Rešenje u Java aplikaciji primenom Dijksra algoritma

Kao početni čvor smo uzeli čvor 1, a za odredišni smo odabrali čvor 6.

Težinska matrica koju smo iskoristili za prikazivanje grafa sa Slike 4.1.1.1. ima sledeći oblik:

```
int[][] matrica =  {{0,3,10,0,0,0,0,0,0},
                    {0,0,0,7,3,0,0,0,0},
                    {0,0,0,0,0,8,0,0,0},
                    {0,0,0,0,6,0,4,2,3},
                    {0,0,2,0,0,0,0,7,0},
                    {0,0,0,0,5,0,0,6,0},
                    {0,0,0,0,0,3,0,0,4},
                    {0,0,0,0,0,0,0,0,5},
                   {0,0,0,0,0,0,0,0,0}};
```

Informacije dobijene iz komandnog prozora Java aplikacije nakon unosa težinske matrice i unosa početnog i odredišnog čvora su:

Sva rastojanja od pocetnog čvora:

Rastojanje od 1 do 1 je: 0

Rastojanje od 1 do 2 je: 3

Rastojanje od 1 do 3 je: 8

Rastojanje od 1 do 4 je: 10

Rastojanje od 1 do 5 je: 6

Rastojanje od 1 do 6 je: 16

Rastojanje od 1 do 7 je: 14

Rastojanje od 1 do 8 je: 12

Rastojanje od 1 do 9 je: 13

Rastojanje do odabranog čvora 6 je: 16

Putanja je: 6 <- 3 <- 5 <- 2 <- 1

Kada uporedimo grafičko rešenje sa rezultatima koje smo dobili pokretanjem Java aplikacije, možemo uočiti da su dobijena rastojanja ista. Takođe možemo zaključiti da se putanja od početnog do odredišnog čvora ispisuje obrnutim redosledom.

4.1.3. Rešenje u Java aplikaciji primenom Floyd-Warshall algoritma

Iskoristićemo isti graf (sa Slike 4.1.1.1.) kao i u prethodnom slučaju.

Potrebno je uneti sledeću težinsku matricu:

```
int   [][] matrica =
      {{0,3,10,INFINITY,INFINITY,INFINITY,INFINITY,INFINITY,INFINITY},

       {INFINITY,0,INFINITY,7,3,INFINITY,INFINITY,INFINITY,INFINITY},

       {INFINITY,INFINITY,0,INFINITY,INFINITY,8,INFINITY,INFINITY,INFINITY},

       {INFINITY,INFINITY,INFINITY,0,6,INFINITY,4,2,3},

       {INFINITY,INFINITY,2,INFINITY,0,INFINITY,INFINITY,7,INFINITY},

       {INFINITY,INFINITY,INFINITY,INFINITY,5,0,INFINITY,6,INFINITY},

       {INFINITY,INFINITY,INFINITY,INFINITY,INFINITY,3,0,INFINITY,4},

       {INFINITY,INFINITY,INFINITY,INFINITY,INFINITY,INFINITY,INFINITY,0,5},

       {INFINITY,INFINITY,INFINITY,INFINITY,INFINITY,INFINITY,INFINITY,INFINITY,0
};
```

Informacije dobijene iz komandnog prozora Java aplikacije nakon unosa parametara su sledeće:

Matrica rastojanja:

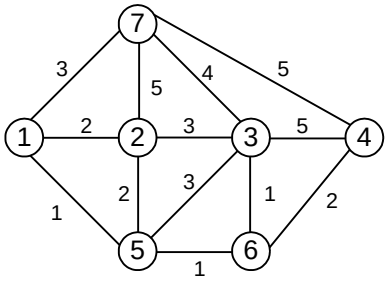
0	3	8	10	6	16	14	12	13
9999	0	5	7	3	13	11	9	10
9999	9999	0	9999	13	8	9999	14	19
9999	9999	8	0	6	7	4	2	3
9999	9999	2	9999	0	10	9999	7	12
9999	9999	7	9999	5	0	9999	6	11
9999	9999	10	9999	8	3	0	9	4
9999	9999	9999	9999	9999	9999	9999	0	5
9999	9999	9999	9999	9999	9999	9999	9999	0

Kao što možemo videti iz prve vrste matrice rastojanja Floyd-Warshall algoritam daje ista rastojanja između čvora 1 i ostalih čvorova u grafu kao i algoritam Dijkstra. Takođe se mogu videti i rastojanja između svih ostalih čvorova u mreži. Oznaka 9999 u matrici rastojanja ukazuje na to da između tih čvorova ne postoji putanja.

4.2. Primer 2

4.2.1. Grafičko rešenje primenom Dijkstra algoritma

Za ispitivanje koda algoritma ćemo koristiti primer iz izvora [4] koji je prikazan u nastavku. Primer ćemo iskoristiti za poređenje rezultata aplikacije sa očekivanim rezultatom.

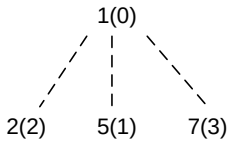


Slika 4.2.1.1. Graf za koji treba odrediti najkraće puteve između čvorova

Putevi su dvosmerni pri čemu je cena identična za oba smera.

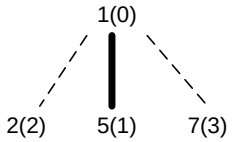
Rešenje:

a) U prvom koraku možemo da koristimo samo čvorove koji predstavljaju susede čvora 1, a to su čvorovi 2, 5 i 7 (Slika 4.2.1.1).



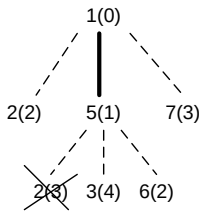
Slika 4.2.1.1. Korak 1

Pošto ovaj algoritam podrazumeva da se sukcesivno dodaju najbliži čvorovi jedan po jedan (najbliži po ceni) prvo se dodaje čvor 5 kao najbliži čvoru 1 (Slika 4.2.1.2.).



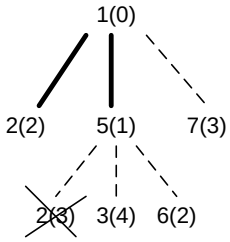
Slika 4.2.1.2. Korak 2

Zatim se pridodaju susedi čvora 5 da bi se utvrdilo da li postoji bolje rešenje do ostalih čvorova u mreži. Cena pridodatih čvorova je nađena cena do čvora 5 + cena linka od čvora 5 do njegovog odgovarajućeg suseda (Slika 4.2.1.3). Za sve čvorove koji još nisu dodati u dijagram se smatra da je rastojanje beskonačno što u stvari znači da još nije pronađen put do njih, ali kako algoritam bude odmicao i oni će biti uključeni u dijagram jer će se pronalaziti i put do njih.



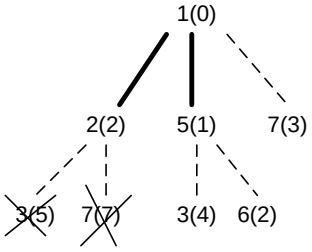
Slika 4.2.1.3. Korak 3

Kao što vidimo na Slici 4.2.1.4. nađen je nov put do čvora 2, ali cena tog puta je veća od onog koji je već bio prethodno pronađen pa se ovaj novi put odbacuje. Takođe, dijagramu su dodati čvorovi 3 i 6 jer je sada nađen put do njih. Sada treba ponovo odrediti koji je sada sledeći najbliži čvor. Vidimo da sada postoje dva kanididata ruteri 2 i 6. Svejedno je koji ćemo izabrati jer ionako će onaj drugi biti izabran u sledećem koraku. Izabraćemo ruter 2.



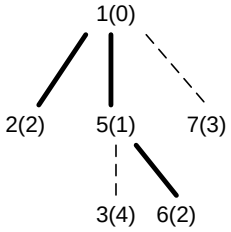
Slika 4.2.1.4. Korak 4

Sada pridodajemo susede čvora 2 dijagramu sa Slike 4.2.1.5 (ne pridodajemo čvorove 1 i 5 jer je za njih već određeno rešenje (tj. najkraći put)):



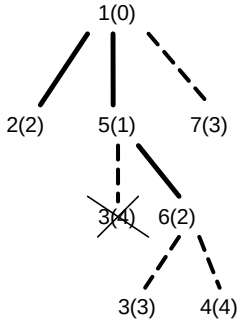
Slika 4.2.1.5. Korak 5

Kao što vidimo oba nova puta do čvorova 3 i 7 su odbačena kao nepovoljnija u odnosu na prethodno pronađene puteve. Sada pridodajemo čvor 6 rešenju (Slika 4.2.1.6.).



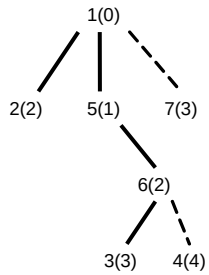
Slika 4.2.1.6. Korak 6

Opisana procedura se sada ponavlja (Slika 4.2.1.7.):



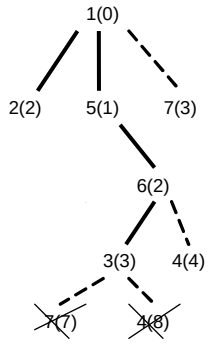
Slika 4.2.1.7. Korak 7

Sada smo imali slučaj da je do čvora 3 pronađen povoljniji put pa se prethodno određeni odbacuje, a uključuje se u dijagram novi put. Sada ćemo priključiti čvor 3 rešenju (Slika 4.2.1.8.):



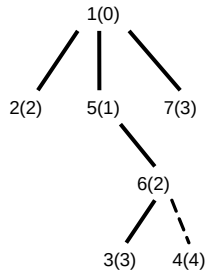
Slika 4.2.1.8. Korak 8

Priključujemo sada susede čvora 3 (Slika 4.2.1.9.):



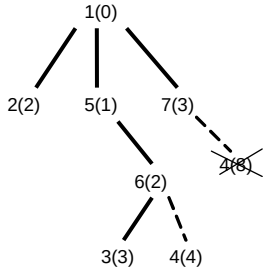
Slika 4.2.1.9. Korak 9

Rešenju sada dodajemo čvor 7 (Slika 4.2.1.10.):



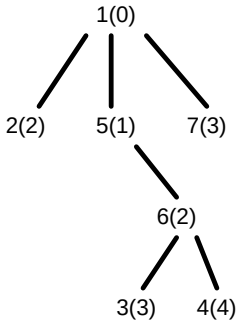
Slika 4.2.1.10. Korak 10

Priključujemo novi put do čvora 4 (jedini koji još nije uključen u konačno rešenje) kao na Slici 4.2.1.11.:



Slika 4.2.1.11. Korak 11

Sada još samo priključujemo čvor 4 u rešenje i time okončavamo algoritam (Slika 4.2.1.12.):



Slika 4.2.1.12. Konačno resenje

4.2.2. Rešenje u Java aplikaciji primenom Dijksra algoritma

Kao početni čvor smo uzeli čvor 1, a za odredišni smo odabrali čvor 3.

Težinska matrica koju smo iskoristili za prikazivanje grafa sa Slike 4.2.1.1. ima sledeći oblik:

```
int[][] matrica =  {{0,2,0,0,1,0,3},
                    {2,0,3,0,2,0,5},
                    {0,3,0,5,3,1,4},
                    {0,0,5,0,0,2,5},
                    {1,2,3,0,0,1,0},
                    {0,0,1,2,1,0,0},
                    {3,5,4,5,0,0,0}};
```

Informacije dobijene iz komandnog prozora Java aplikacije nakon unosa parametara su sledeće:

Sva rastojanja od početnog cvora:

Rastojanje od 1 do 1 je: 0

Rastojanje od 1 do 2 je: 2

Rastojanje od 1 do 3 je: 3

Rastojanje od 1 do 4 je: 4

Rastojanje od 1 do 5 je: 1

Rastojanje od 1 do 6 je: 2

Rastojanje od 1 do 7 je: 3

Rastojanje do odabranog čvora 3 je: 3

Putanja je: 3 <- 6 <- 5 <- 1

Upoređivanjem rezultata sa Slike 4.2.1.11. sa podacima koje smo dobili pokretanjem Java aplikacije možemo videti da su u oba slučaja dobijena ista rastojanja između odabranog čvora i ostalih čvorova u mreži.

4.2.3. Rešenje u Java aplikaciji primenom Floyd-Warshall algoritma

Kako bismo lakše uporedili dobijene rezultate iskoristićemo isti graf kao i u prethodnom slučaju (graf sa Slike 4.2.1.1).

Težinska matrica koju treba da simulira dati graf je sledeća:

```
int[][] matrica = {{0, 2, INFINITY, INFINITY, 1, INFINITY, 3},
                   {2, 0, 3, INFINITY, 2, INFINITY, 5},
                   {INFINITY, 3, 0, 5, 3, 1, 4},
                   {INFINITY, INFINITY, 5, 0, INFINITY, 2, 5},
                   {1, 2, 3, INFINITY, 0, 1, INFINITY},
                   {INFINITY, INFINITY, 1, 2, 1, 0, INFINITY},
                   {3, 5, 4, 5, INFINITY, INFINITY, 0} };
```

Informacije dobijene iz komandnog prozora Java aplikacije nakon unosa parametara su sledeće:

Matrica rastojanja:

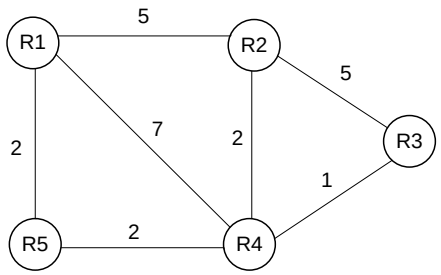
0	2	3	4	1	2	3
2	0	3	5	2	3	5
3	3	0	3	2	1	4
4	5	3	0	3	2	5
1	2	2	3	0	1	4
2	3	1	2	1	0	5
3	5	4	5	4	5	0

Kao što možemo videti iz prve vrste i prve kolone matrice (pošto su putevi dvosmerni, a cene puteva identične u oba slučaja) primenom Floyd-Warshall algoritma dobijamo ista rastojanja između čvora 1 i ostalih čvorova u grafu kao i primenom algoritama Dijkstra, što je i očekivano. Takođe se mogu videti i rastojanja između svih ostalih čvorova u mreži.

4.3. Primer 3

4.3.1. Grafičko rešenje primenom Dijkstra algoritma

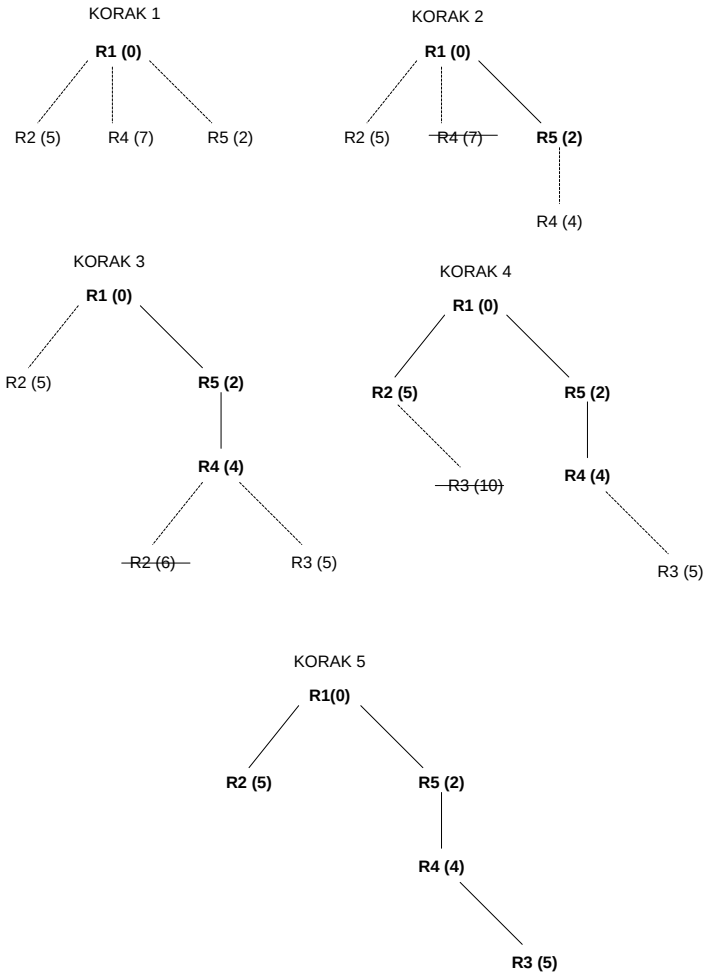
Za ispitivanje koda algoritma ćemo koristiti primer iz izvora [4] koji je prikazan u nastavku.



Slika 4.3.1.1. Graf za koji treba naći najkraći put

Rešenje:

Na Slici 4.3.1.2. su prikazani koraci na osnovu kojih se dolazi do rezultujućeg stabla, kao i samo rezultujuće stablo.



Slika 4.3.1.2. Rezultujuće stablo je stablo dato u koraku 5.

4.3.2. Rešenje u Java aplikaciji primenom Dijksra algoritma

Kao početni čvor smo uzeli čvor 1, a za odredišni smo odabrali čvor 3.

Težinska matrica koju smo iskoristili za prikazivanje grafa sa Slike 4.3.1.1. ima sledeći oblik:

```
int[][] matrica =  {{0,5,0,7,2},
                    {5,0,5,2,0},
                    {0,5,0,1,0},
                    {7,2,1,0,2},
                     {2,0,0,2,0}};
```

Informacije dobijene iz komandnog prozora Java aplikacije nakon unosa težinske matrice i unosa početnog i odredišnog čvora su:

Sva rastojanja od pocetnog čvora:

Rastojanje od 1 do 1 je: 0

Rastojanje od 1 do 2 je: 5

Rastojanje od 1 do 3 je: 5

Rastojanje od 1 do 4 je: 4

Rastojanje od 1 do 5 je: 2

Rastojanje do odabranog čvora 3 je: 5

Putanja je: 3 <- 4 <- 5 <- 1

Upoređivanjem grafičkog rešenja sa Slike 4.3.1.2 i rešenja dobijenog korišćenjem Java aplikacije možemo zaključiti da su rastojanja između izabranog početnog čvora 1 i ostalih čvorova iste. Takođe možemo videti i da putanja koju ispisuje program odgovara putanji u rezultujućem stablu između željenih čvorova u mreži (treba imati na umu da program putanju ispisuje u obrnutom redosledu).

4.3.3. Rešenje u Java aplikaciji primenom Floyd-Warshall algoritma

Iskoristićemo isti graf (sa Slike 4.3.1.1.) kao i u prethodnom slučaju.

Potrebno je uneti sledeću težinsku matricu:

```
int[][] matrica = {{0, 5, INFINITY, 7, 2},
                   {5, 0, 5, 2, INFINITY},
                   {INFINITY, 5, 0, 1, INFINITY},
                   {7, 2, 1, 0, 2},
                   {2, INFINITY, INFINITY, 2, 0} };
```

Informacije dobijene iz komandnog prozora Java aplikacije nakon unosa težinske matrice su:

Matrica rastojanja:

0	5	5	4	2
5	0	3	2	4
5	3	0	1	3
4	2	1	0	2
2	4	3	2	0

Upoređivanjem dobijenih rezultata sa rezultatima dobijenim korišćenjem Dijkstra algoritma možemo zaključiti da su dobijena rastojanja između odgovarajućih čvorova ista.

5. ZAKLJUČAK

Algoritmi za nalaženje najkraćeg puta u mreži, Dijkstra i Floyd-Warshall, su uspešno implementirani u programskom jeziku Java. Aplikacija uspešno pronalazi rastojanje između odabranog početnog čvora i svih ostalih čvora primenom Dijkstra algoritma, kao i najkrača rastojanja između svih čvorova u mreži primenom Floyd-Warshall algoritma.

Glavni nedostatak aplikacije je što se za svaku mrežu moramo ručno uneti težinsku matricu grafa što za velike mreže sa mnogo čvorova predstavlja veliki problem.

Dalji razvoj aplikacije mogao bi voditi ka implementaciji ove aplikacije izvršavanju na Android platformi. Moguće je implementirati i drugačiji unos strukture mreže, kao što je recimo unošenje težinske matrice iz tekstualnog fajla, što bi znatno olakšalo unos podataka.

LITERATURA

[1] http://tesla.pmf.ni.ac.rs/people/dragance/Knjiga1MO.pdf

[2] https://informatikacg.files.wordpress.com/2015/09/srednja-skola-instalacija-alata-2015.pdf

[3] https://www.google.rs/url?sa=t&rct=j&q=&esrc=s&source=web&cd=9&ved=0CFsQFjAIahUKEwjcl4i19qjIAhXMhiwKHXOMDyY&url=http%3A%2F%2Frutcor.rutgers.edu%2F~boliac%2FDijkstra.doc&usg=AFQjCNGdmVynyr_vUP5VhoUhjaerZFBp8Q&sig2=SK6v1D0w29zYlDNMp4NFfg&bvm=bv.104317490,d.bGg&cad=rja

[4] Materijali iz predmeta Računarske osnove i primena Interneta