Pregled algoritama za traženje najkraćeg puta u grafu

Žunić, Salim

SVEUČILIŠTE U ZAGREBU
FAKULTET PROMETNIH ZNANOSTI

Salim Žunić

PREGLED ALGORITAMA ZA TRAŽENJE NAJKRAĆEG PUTA U GRAFU

ZAVRŠNI RAD

Zagreb, 2018.

SVEUČILIŠTE U ZAGREBU
FAKULTET PROMETNIH ZNANOSTI
ODBOR ZA ZAVRŠNI RAD

Zagreb, 5. travnja 2018.

Zavod: **Zavod za inteligentne transportne sustave**
Predmet: **Algoritmi i programiranje**

ZAVRŠNI ZADATAK br. 4862

Pristupnik: **Salim Žunić (0135216831)**
Studij: Promet
Smjer: Informacijsko-komunikacijski promet

Zadatak: **Pregled algoritama za traženje najkraćeg puta u grafu**

Opis zadatka:

Cilj rada je dati pregled algoritama za traženje najkraćeg puta u grafu koji opisuje prometnu mrežu u informacijsko komunikacijskom prometu. Opisat će se algoritmi te će se implementirati barem jedan algoritam. Rad algoritma prikazat će se na primjeru manje prometne mreže.

Mentor:

dr. sc. Juraj Fosin

Predsjednik povjerenstva za završni ispit:

Sveučilište u Zagrebu
Fakultet prometnih znanosti

ZAVRŠNI RAD

Pregled algoritama za traženje najkraćeg puta u grafu

Overview of Algorithms for the Shortest Path Problem

Mentor: dr.sc. Juraj Fosin
Student: Salim Žunić, 0135216831

Zagreb, rujan 2018.

PREGLED ALGORITAMA ZA TRAŽENJE NAJKRAĆEG PUTA U GRAFU

SAŽETAK

U ovom radu prikazati će se pregled algoritama za traženje najkraćeg puta u grafu. Prikazani su Dijsktrin algoritam i Floyd-Warshallov algoritam, te su objašnjeni na primjerima manjih grafova. U prvom dijelu rada objašnjena je definicija grafova, osnovni načini prikaza grafova, te njihova osnovna podjela. Implementiran je Dijkstrin algoritam u programskom jeziku Java. Program je napravljen u razvojnom okruženju *InteliJ*. Algoritam je implementiran na primjeru grafa koji se sastoji od osam čvorova i simbolizira manju prometnu mrežu u informacijsko komunikacijskom prometu. Čvorovi grafa predstavljaju usmjerivače *(engl. Router).* Cilj programa je izračunati najmanje moguće količine kašnjenja od početnog čvora, do ostalih čvorova u mreži, odnosno grafu. Usmjeravanje u Internet prometu je proces pronalaženja puta kojim će se slati paketi od izvorišnog usmjerivača do odredišta. Cilj je da paketi na odredište stignu što brže, te se pri odlučivanju koriste algoritmi za traženje najkraćeg puta, poput Dijkstrinog algoritma implementiranog u ovom radu.

KLJUČNE RIJEČI: Java, Graf, Dijkstrin algoritam, Problem najkraćeg puta

OVERVIEW OF ALGORITHMS FOR THE SHORTEST PATH PROBLEM

SUMMARY

This paper presents overview of algorithms for the shortest path problem. It explains Dijkstra algorithm and Floyd-Warshalls algorithm that are explained on examples of smaller graphs. Definition of graphs, basic methods of graph presentation and their basic division are explained in fist part of the paper. Dijkstra algorithm is implemented in Java programing language. Program was made in *InteliJ* development environment. Algorithm is implemented on the example of graph which is made of eight nodes. Graph symbolizes smaller information and communication traffic network. Goal of program is to calculate delays from source node to all other nodes in network. Routing, in Internet traffic, is the process of choosing a path over which to send packets from source router to destination. Goal is for packets to be delivered as soon as possible, for decision making we are using shortest path algorithms as Dijsktra algorithm implemented in this paper.

KEYWORDS: Java, Graph, Dijkstra algorithm, Shortest path problem

SADRŽAJ

1 UVOD

Cilj rada je dati pregled algoritama za traženje najkraćeg puta u grafu koji opisuje prometnu mrežu u informacijsko komunikacijskom prometu. Opisat će se algoritmi te će se implementirati barem jedan algoritam. Rad algoritma će se prikazati na primjeru manje prometni mreže.

Završni rad sastoji se od šest funkcionalno povezanih dijelova ili teza:

1. Uvod
2. Opis problema traženja nakraćeg puta u grafu
3. Algoritmi za rješavanje problema najkraćeg puta u grafu
4. Implementacija algoritma
5. Rezultati
6. Zaključak

Prvo poglavlje završnog rada je *Uvod* u kojem se iznose predmet i cilj rada te njegova struktura.

Drugo poglavlje pod nazivom *Opis problema traženja najkraćeg puta u grafu* prikazuje problem traženja najkraćeg puta u grafu, te pobliže objašnjava pojmove grafova i vrste grafova

U trećem poglavlju rada pod nazivom *Algoritmi za rješavanje problema najkraćeg puta u grafu* objašnjena su dva algoritma, Dijkstrin algoritam i Floyd-Warshallov algoritam.

U četvrtom poglavlju pod nazivom *Implementacija algoritama* pokazan je način na koji se može pomoću Java programskog jezika, implementirati Dijkstrin algoritam.

Peti dio rada su *Rezultati* programa pisanog u Java programskom jeziku, te u *InteliJ* razvojnom okruženju.

Šesti dio rada je *Zaključak* koji je donesen na temelju istraživanja i vlastitih promišljanja.

2 OPIS PROBLEMA TRAŽENJA NAJKRAĆEG PUTA U GRAFU

U teoriji grafova, problem najkraćeg puta je problem pronalaženja puta između dva čvora u grafu tako da je zbroj težina njegovih bridova minimalan.

U problemu najkraćeg puta traži se put u težinskom grafu koji povezuje dva čvora tako da zbroj težina bridova na tom putu bude najmanji mogući. Problem postoji i u netežinskom grafu, gdje se traži put s najmanjim brojem bridova koji povezuje dvije točke u grafu.

2.1 Definicija grafova

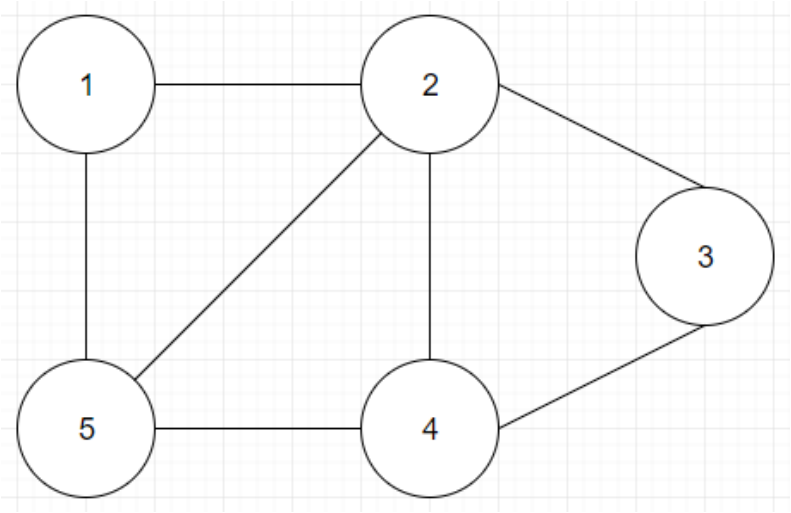
Graf je apstraktna struktura podataka koja se sastoji od čvorova i bridova koji ih povezuju. Formalno se definira kao uređen par $G = (V, E)$ skupa čvorova V i skupa bridova E. Mnoge situacije iz realnog života možemo predstaviti grafovima. Čvorovi, na primjer, mogu predstavljati gradove, a bridovi puteve koji povezuju te gradove.

Grafovi se mogu prezentirati na dva načina, kao lista (*engl. Adjacency list*) ili kao matrica (*engl. Adjacency matrix*). Oba dva načina odnose se na usmjerene i neusmjerene grafove.[10]

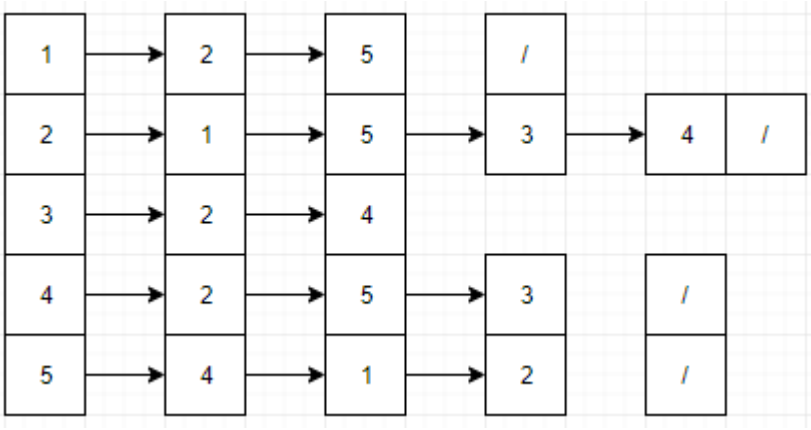
2.1.1 Lista

Prikaz grafa $G = (V, E)$ pomoću liste sastoji se od redova koji prikazuju svaki čvor u grafu i njegove susjedne čvorove.[4]

Na primjeru malog grafa G od 5 čvorovaova pokazat će se prikaz pomoću liste. Radi se o neusmjerenom grafu.



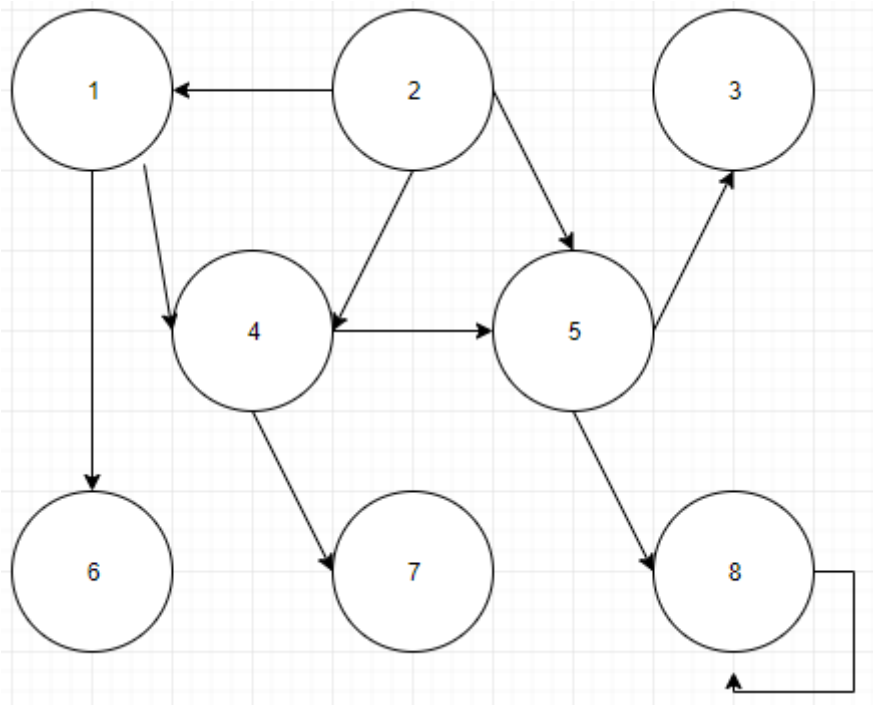
Slika 1. Prikaz grafa G s 5 čvorova



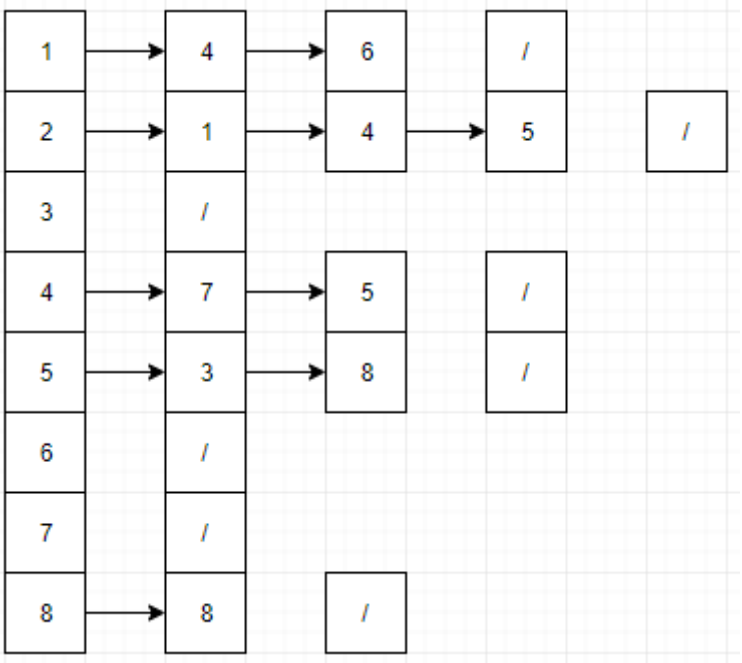
Slika 2. Prikaz grafa G pomoću liste

Prvi stupac liste prikazane na slici 2, simbolizira svaki čvor u grafu *G* prikazanog na slici 1. Drugi stupac prikazuje prvi susjedni čvor i tako dalje svaki stupac prikazuje susjedne čvorove od čvora iz prvog stupca. Kao što se vidi u grafu, čvor *1* je povezan s čvorom *2* i čvorom *5*, zato u listi nakon čvora *1* slijede čvorovi *2* i *5*. Isto vrijedi i za ostale čvorove. Čvor *2* je povezan sa svim čvorovima, te zbog toga u listi nakon čvora *2* slijede čvorovi *1, 5, 3* i *4*. Ovo je bio primjer za neusmjereni graf.

Na primjeru grafa *D*, sa slike 3, koji se sastoji od osam čvorova prikazat će se lista za usmjereni graf.



Slika 3. Usmjereni graf D



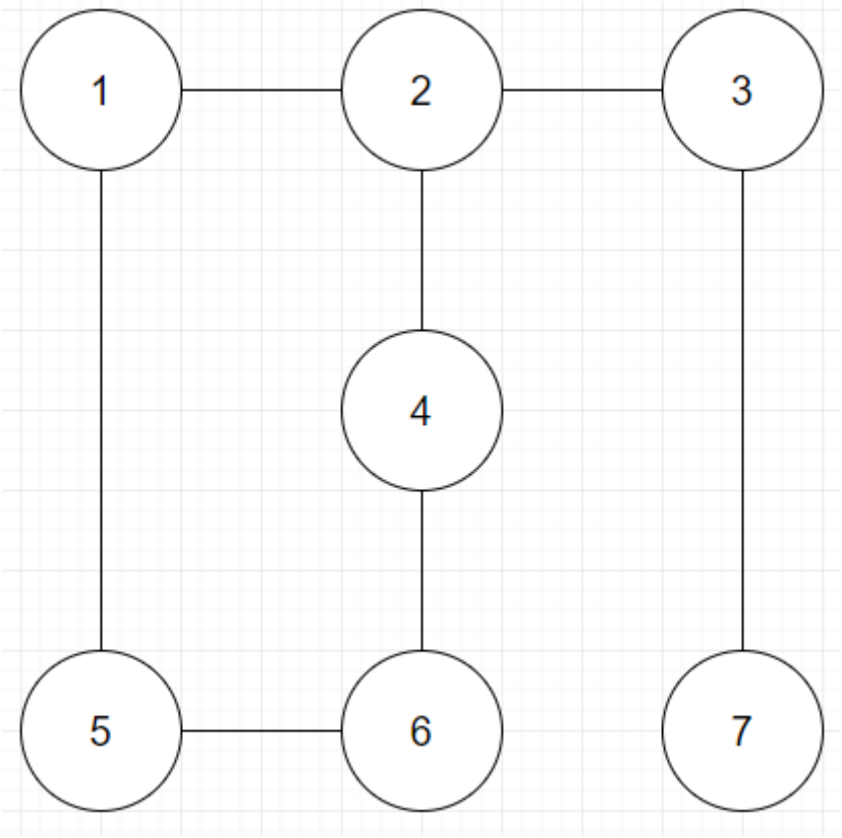
Slika 4. Prikaz grafa D pomoću liste

Prvi stupac liste na slici 4 prikazuje sve čvorove u grafu *D*, od *1* do *8*. Zatim u listi kod usmjerenog grafa ne prikazuju se svi susjedni čvorovi, nego samo oni kojima je čvor početna točka i prikazuje dalje čvorove prema kojima je usmjeren. Npr. čvor *1* je kao što se vidi u grafu usmjeren prema čvoru *4* i *6*, zato su čvorovi *4* i *6* prikazani u listi, dok čvor *2* nije. Čvor *2* je usmjeren prema čvoru *1*, a ne obratno.

2.1.2 Matrica

Matrica udaljenosti u grafu $G = (V, E)$ je $n \times n$ matrica $D = (d_{ij})$ gdje je n broj čvorova u grafu G i d_{ij} je broj bridova između čvorova. $d_{ij} = 0$ ako dva čvora u grafu nisu povezana.[4]

Na primjeru grafa *G*, na slici 5, koji se sastoji od sedam čvorova, prikazat će se matrica.



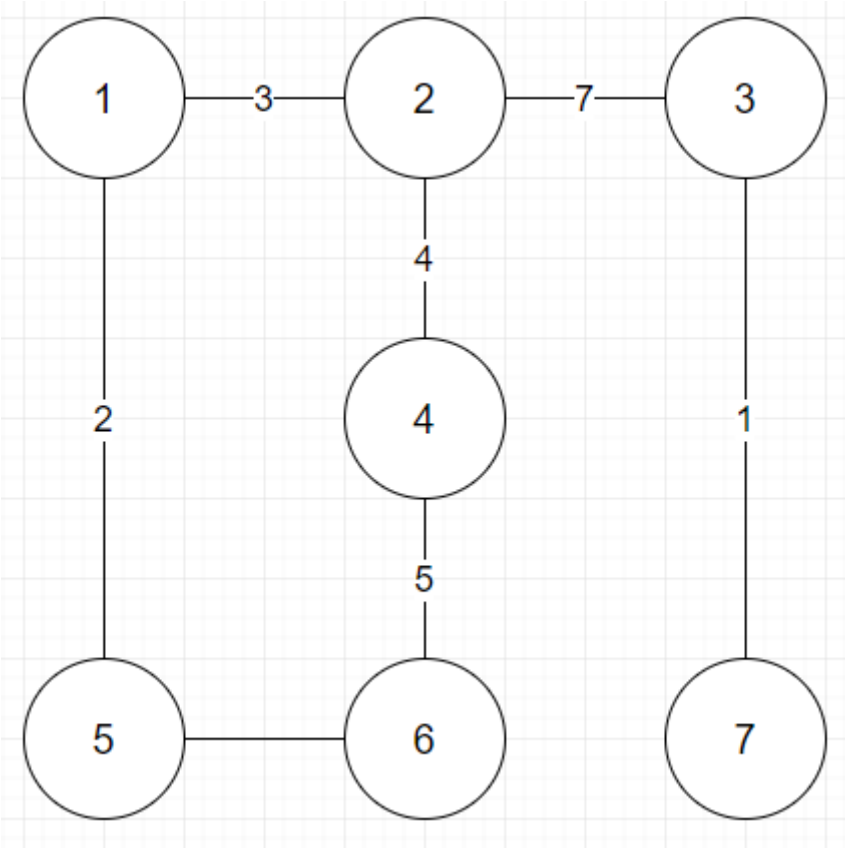
Slika 5. Prikaz grafa G s sedam čvorova

	1	2	3	4	5	6	7
1	0	1	0	0	1	0	0
2	1	0	1	1	0	0	0
3	0	1	0	0	0	0	1
4	0	1	0	0	0	1	0
5	1	0	0	0	0	1	0
6	0	0	0	1	1	0	0
7	0	0	1	0	0	0	0

Slika 6. Prikaz grafa G pomoću matrice

Matrica je odličan način prikazivanja grafa jer se jednostavno i jasno vidi koji čvorovi su međusobno povezani. Slika 6 prikazuje matricu grafa *G*. Na primjeru čvora *1* sve vrijednosti su nula, osim vrijednosti kod čvora *2* i čvora *5*, zato što je s tim čvorovima povezan. Za čvor *2* sve vrijednosti su nula, osim vrijednosti kod čvora *1*, *3*, i *4*, zato što je s tim čvorovima povezan. Tako redom do čvora *7* koji ima vrijednost jedan samo kod čvora *3*, zato što je samo s tim čvorom povezan.

Na isti graf *G*, kao u primjeru prije dodati će se vrijednosti na bridove i napraviti matrica. Što je prikazano na slici 7.



Slika 7. Graf G s dodanim vrijednostima

	1	2	3	4	5	6	7
1	0	3	0	0	2	0	0
2	3	0	7	4	0	0	0
3	0	7	0	0	0	0	1
4	0	4	0	0	0	5	0
5	2	0	0	0	0	2	0
6	0	0	0	5	2	0	0
7	0	0	1	0	0	0	0

Slika 8. Prikaz težinskog grafa G pomoću matrice

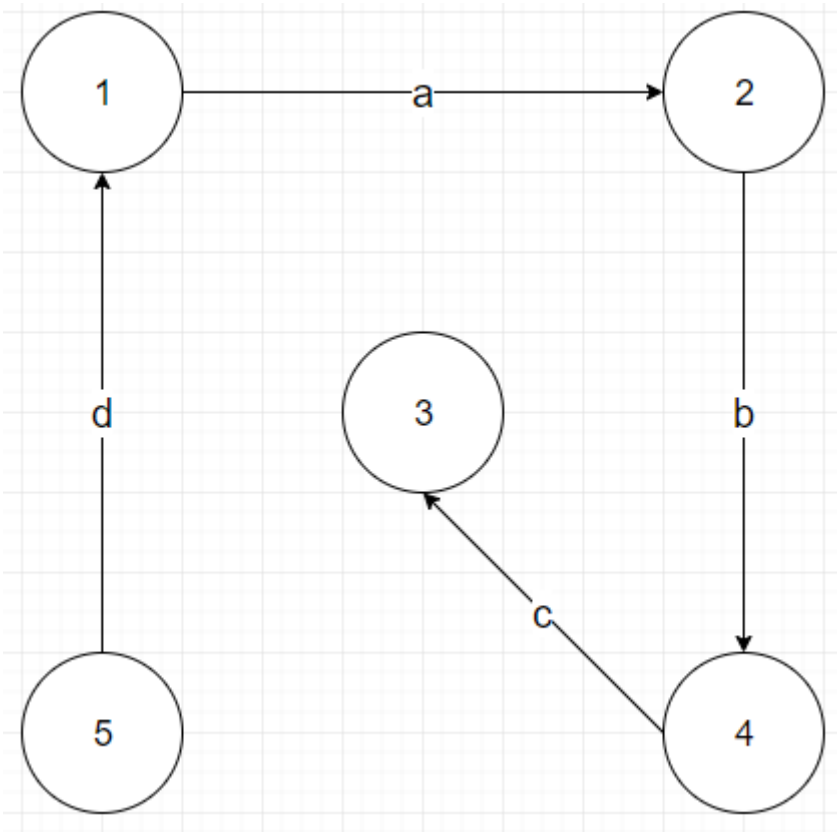
Slika 8 prikazuje matricu koja ne predstavlja broj bridova između čvorova, nego njihove težine, tj. vrijednosti. Za čvor *1*, vrijednost kod čvora *2* jednaka je tri, zato što je tolika težina brida koja ih spaja. Isto tako kod čvora *5* je vrijednost dva. Za čvor *2*, vrijednost tri je kod čvora *1*, zato što je tolika vrijednost brida koja ih povezuje, vrijednost sedam je kod čvora *3*, a vrijednost četiri kod čvora *4*. I tako redom do čvora *7* koji je povezan samo s čvorom *3* i težina njihovog brida jednaka je jedan.

2.2 Vrste grafova

Graf može biti usmjeren (orijentiran) i neusmjeren (neorijentiran). U neusmjerenom grafu, ako postoji brid od čvora *x* do čvora *y*, postoji i od *y* do *x*. U usmjerenom grafu, ako postoji brid od *x* do *y*, to ne znači da postoji i brid od *y* do x. U usmjerenom grafu čvorove možemo promatrati kao uređene parove *(x,y)*, *(x,y pripadaju V)* i crtamo ih korištenjem strelica usmjerenih od čvora *x* ka čvoru *y*, a u neusmjerenom grafu čvorovi su dvočlani skupovi *{x,y}* i crtamo ih kao obične linije.[5]

2.2.1 Usmjereni graf

Usmjereni graf je graf sastavljen od čvorova povezanih s usmjerenim bridovima. Usmjereni graf $G = (V, E)$, gdje je V set čvorova, a E set bridova. Razlika s neusmjerenim grafovima je ta što su elementi unutar E uređeni parovi. Brid od čvora u do čvora v je zapisan kao par (u, v), a par (v, u) predstavlja brid u suprotnom smjeru. Čvor u je početni čvor brida, a čvor v je završni čvor brida. Objasnit će se na primjeru grafa $G= V\{1, 2, 3, 4, 5\}, E\{a, b, c, d\}$ prikazanog na slici 9.

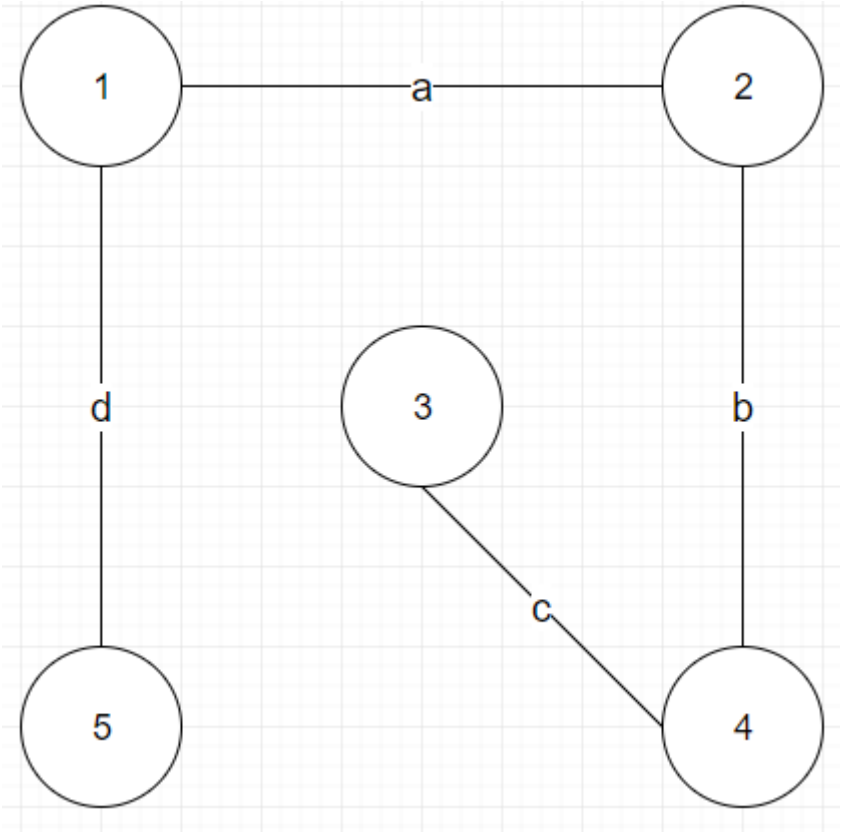


Slika 9. Primjer usmjerenog grafa G

Čvor *1* je povezan s čvorovima *2* i *5*, ali je usmjeren prema čvoru *2*, stoga brid *a* ima zapis *a (1, 2).* Gdje je *1* početni čvor brida *a*, a *2* završni čvor brida *a*. Brid *b* ima zapis *b (2, 4)*, gdje je *2* početni čvor brida *b*, a čvor *4* završni čvor brida *b*. Brid *c* ima zapis *c (4, 3)*, gdje je *4* početni čvor brida *c*, a *3* završni čvor brida *c*. Brid *d* ima zapis *d (5, 1)*, gdje je čvor *5* početni čvor brida *d*, a čvor *1* završni čvor brida *d*.[5]

2.2.2 Neusmjereni graf

Neusmjereni graf (*engl. undirected graph*) je uređeni par skupova *(V, E)*. Elemente skupa *V* nazivamo čvorovima, a one skupa *E* bridovima grafa. Brid grafa je par elemenata iz skupa *V*. Za neku granu kažemo da povezuje dva čvora iz skupa *V*. Objasnit će se na primjeru grafa *G= V{1, 2, 3, 4, 5}, E{a, b, c, d}* prikazanog na slici 10. [5]



Slika 10. Primjer neusmjerenog grafa G

Za granu *a* može se reći da povezuje čvorove *1* i *2*. Time su čvor *1* i čvor *2* susjedni čvorovi. Brid *b* povezuje čvorove *2* i *4*. Brid *c* povezuje čvorove *4* i *3*. Brid *d* povezuje čvorove *5* i *1*. Graf može biti povezan ili nepovezan. Graf je povezan ako postoji put od bilo kojeg čvora do bilo kojeg drugog čvora u grafu. Ako takav put ne postoji, graf je nepovezan. Budući da u grafu *G* postoji put između svih čvorova, graf *G* je povezan.

3 ALGORITMI ZA RJEŠAVANJE PROBLEMA NAJKRAĆEG PUTA U GRAFU

Neformalno, algoritam je bilo koji dobro definiran računalni postupak koji uzima neku vrijednost, ili skup vrijednosti kao ulaze i proizvodi neku vrijednost, ili skup vrijednosti, kao izlaze. Algoritam je tako slijed računalnih koraka koji pretvaraju ulaz u izlaz. Također algoritam se može vidjeti kao alat za rješavanje dobro određenog računalnog problema. Izjava o problemu općenito opisuje željeni odnos ulaza i izlaza. Algoritam opisuje specifičan računalni postupak za postizanje tog odnosa ulaza i izlaza.

Na primjer, mora se poredati niz brojeva u redoslijed od najmanjeg do najvećeg. Ovaj problem nastaje često u praksi i pruža plodno tlo za uvođenje mnogih standardnih tehnika dizajna i alata za analizu. Formalno definiranje problema sortiranja:

· **Ulaz:** Slijed *n* brojeva a_1, a_2,……,a_n
· **Izlaz:** Permutacija ulaza u redosljed takav da je $a_1 <= a_2$ ……$<=a_n$

Na primjer, s obzirom na ulazni niz *(31, 41, 59, 26, 41, 58)* vraća se sortirani niz kao izlazni niz. Takav ulazni slijed zove se instanca problema sortiranja. Općenito, instanca problema sastoji se od ulaza potrebnog za izračunavanje rješenja problema.

Za algoritam se kaže da je ispravan ako, za svaku instancu unosa, završi s ispravnim izlazom.

Kažemo da ispravni algoritam rješava zadani računalni problem. Neispravni algoritam uopće ne može završiti sa nekim odgovorom, ili može završiti s različitim odgovorom od željenog.

Analiziranje algoritma je predviđanje resursa koje algoritam zahtijeva. Povremeno, resursi poput memorije, komunikacijske propusnosti ili računalni hardver su od primarne važnosti, ali najčešće se mjeri vrijeme potrebno da se algoritam izvrši.[7]

3.1 Dijkstrin algoritam

Dijkstrin algoritam, koji je izradio nizozemski informatičar Edsger Dijkstra 1956. godine i objavio 1959. godine, je algoritam pretraživanja grafa koji rješava problem najkraćeg puta za graf s pozitivnim vrijednostima bridova. Ovaj se algoritam često koristi u usmjeravanju i kao potprogram u drugim algoritmima.[8]

```
1 funkcija Dijkstra(Graf, izvor):
2
3   izradi skup čvorova Q
4
5   za svaki čvor v u Grafu:
6     udaljenost[v] ← BESKONAČNO
7     prethodni[v] ← NEODREĐEN
8     dodaj v u Q
9
10   udaljenost[izvor] ← 0
11
12   dok Q nije prazan:
13     u ← čvor u Q sa min udaljenosti[u]
14     ukloni u iz Q
15
16     za svaki susjedni v od u:
17       alt ← udaljenost[u] + duljina(u, v)
18       ako alt < udaljenost[v]:
19         udaljenost[v] ← alt
20         prethodni[v] ← u
21
22   vrati udaljenost[], prethodni[]
```

Algoritam 1. Pseudokod Dijkstrinog algoritma[4]

Algoritam 1 prikazuje pseudokod Dijkstrinog algoritama koji radi sljedeće. Na početku se izradi prazan skup čvorova *Q*. Zatim se postavlja udaljenost svih čvorova na početne udaljenosti koja iznosi beskonačno. Čvorovi *v* se dodaju u skup *Q*. Prethodnici svakog čvora se postavljaju na vrijednost *ništa*. Dalje u liniji 10 udaljenost odredišnog čvora se postavlja na nulu. Počevši u liniji 12, sve dok je *Q* neprazan skup. Minimalna vrijednost čvora u skupu *Q* se postavlja u varijablu *u*. Za vrijeme dok ima čvorova u skupu *Q*, algoritam s njegovog početka uzima najmanju vrijednost čvora *u*. Zatim u liniji 16 za svakog susjeda *v*

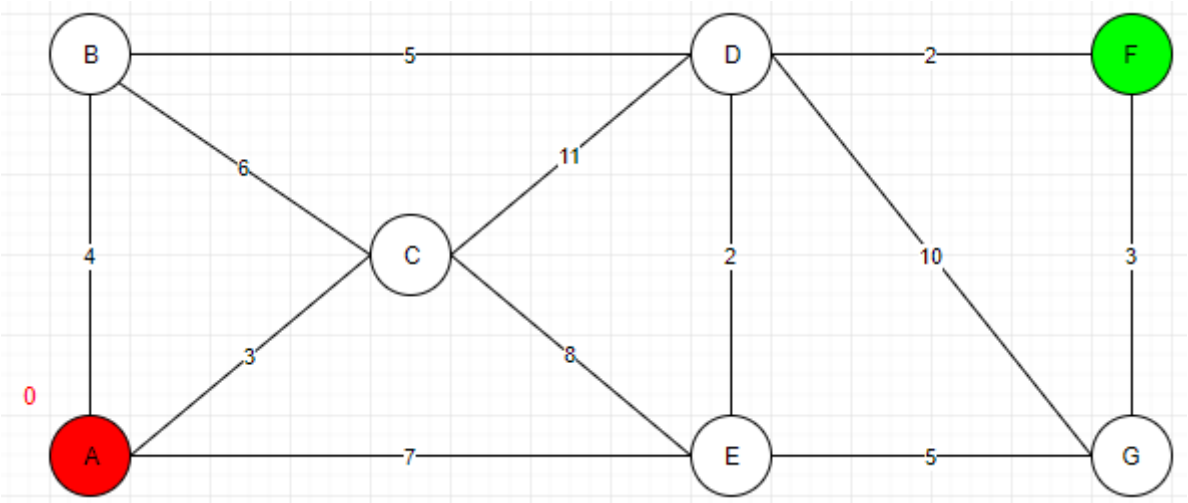
od vrijednosti *u*, koja je trenutna najmanja vrijednost radi sljedeće: prvo se varijabla *alt* postavlja na zbroj trenutne minimalne vrijednosti *u* i udaljenosti *u* od susjeda *v*, ta udaljenost predstavlja brid, koja se razmatra. Zatim ako je udaljenost *alt* manja od vrijednosti susjednog čvora *v*, tom susjednom čvoru se mijenja vrijednost na iznos *alt*, a samim time njegov prethodnik postaje najmanji čvor te se izbacuje iz skupa *Q*. Petlja se ponavlja sve dok se skup *Q* ne isprazni.

Za određeni čvor u grafu algoritam pronalazi put s najmanjim troškom (tj. najkraćim putem) između tog čvora i svakog drugog čvora. Može se koristiti i za pronalaženje najkraćih puteva iz jednog čvora do samo jednog odredišnog čvora zaustavljajući algoritam nakon što je najkraći put do odredišnog čvora određen. Na primjer, ako čvorovi grafa predstavljaju gradove i troškovi bridova predstavljaju udaljenosti vožnje između gradova povezanih izravnim putem, Dijkstrin algoritam može se koristiti za pronalaženje najkraćeg puta između jednog grada i svih drugih gradova. Kao rezultat toga, najkraći put prvo se naširoko koristi u protokolima mrežnog usmjeravanja.[1]

Pretpostavimo da se želi pronaći najkraći put između dva raskrižja na karti grada, polazište i odredište. Za početak potrebno je označiti udaljenost svakog raskrižja na karti znakom beskonačnosti. To ne znači da postoji beskonačna udaljenost, nego se označava da to raskrižje još nije posjećeno. Neke varijante ove metode ostavljaju raskrižje bez oznaka. Nadalje, u svakoj budućoj iteraciji, odabire se trenutno raskrižje. Za prvu iteraciju trenutno raskrižje će biti početna točka i udaljenost do njega (oznake raskrižja) bit će nula. Za kasnije iteracije (poslije prve) trenutačno raskrižje će biti najbliže neposjećeno raskrižje od polazne točke. Od trenutnog raskrižja ažurira se udaljenost do svakog neposjećenog raskrižja koje je izravno povezano s njim. To se radi određivanjem zbroja udaljenosti između neposjećenog raskrižja i vrijednosti trenutnog raskrižja, a ako je vrijednost neposjećenog raskrižja manja od trenutne vrijednosti, potrebno ga je ponovno označiti. Put se mijenja ako je put do neposjećenog raskrižja, kroz trenutno raskrižje, kraći od prethodno poznatih puteva. Nakon što se ažuriraju udaljenosti svakom susjednom raskrižju, označi se trenutačno posjećeno raskrižje i odabire se neposjećeno raskrižje s najmanjom udaljenosti kao trenutno raskrižje. Čvorovi vidljivi kao posjećeni označeni su oznakom najkraćeg puta od početne točke do njega i neće se vraćati prema njemu. Nastavlja se s ovim procesom ažuriranja susjednih raskrižja s

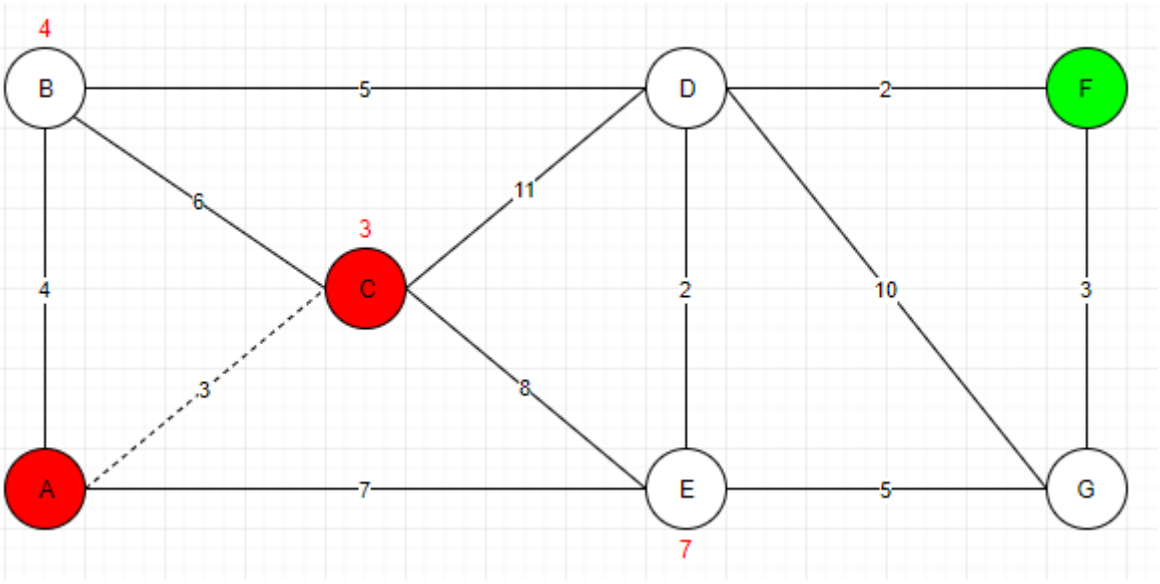
najkraćim udaljenostima, a zatim se označava trenutno raskrižje kao posjećeno i kreće do najbližeg neposjećenog raskrižja dok se ne označi odredište kao posjećeno. Nakon što se označi odredište kao posjećeno (kao što je slučaj s bilo kojim raskrižjem koje se posjetilo) dobiva se najkraći put do njega od početne točke i može se pratiti put natrag sljedeći oznake unatrag. Ovaj algoritam jedino razmatra određivanje sljedećeg "trenutnog" križanja i njegovu udaljenost od početne točke. Ovaj se algoritam stoga "širi prema van" od početne točke, te razmatra svaki čvor koji je bliže u smislu najkraće udaljenosti do odredišta. Algoritam pronalazi najkraći put, a mana mu je relativna sporost u nekim topologijama.[1]

Sada će se prikazati način na koji Dijkstrin algoritam dolazi od čvora *A* do čvora *F* na ponuđenom grafu.



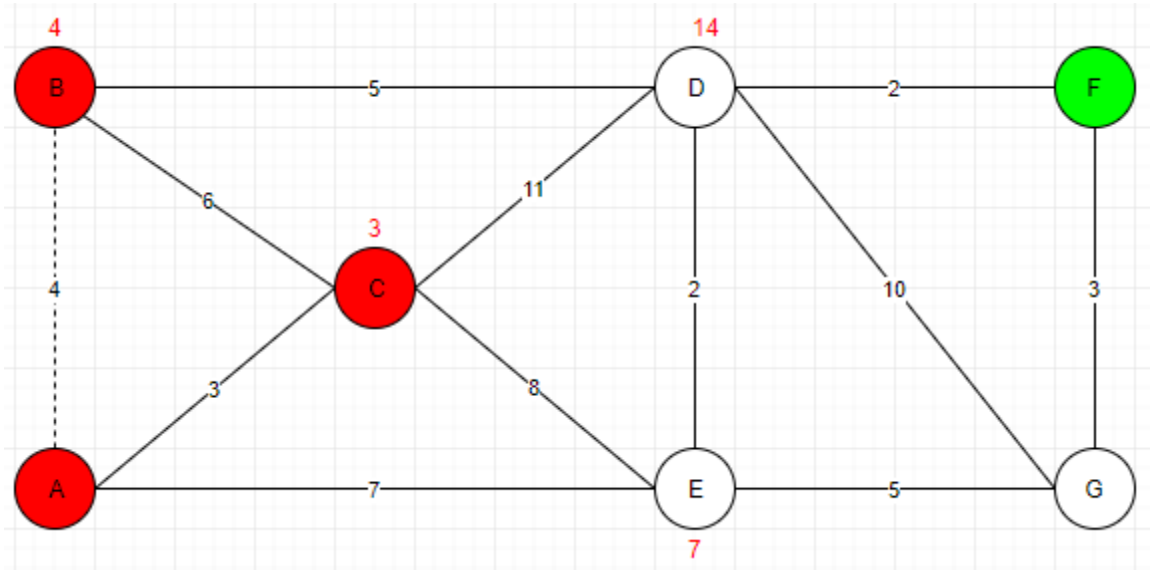
Slika 11. Početni čvor grafa

Na slici 11 vidi se da je početni čvor ovog grafa je čvor *A*, zato je njegova trenutna vrijednost nula. Dalje se gleda koji susjedni čvor ima najmanju vrijednost. U ovom slučaju to je čvor *C*, njegova vrijednost je tri, tako da se pomiče sa čvora *A* na čvor *C* što je prikazano na slici 12. Crvenom bojom će se označavati čvorovi koji su posjećeni.



Slika 12. Pomak na čvor C

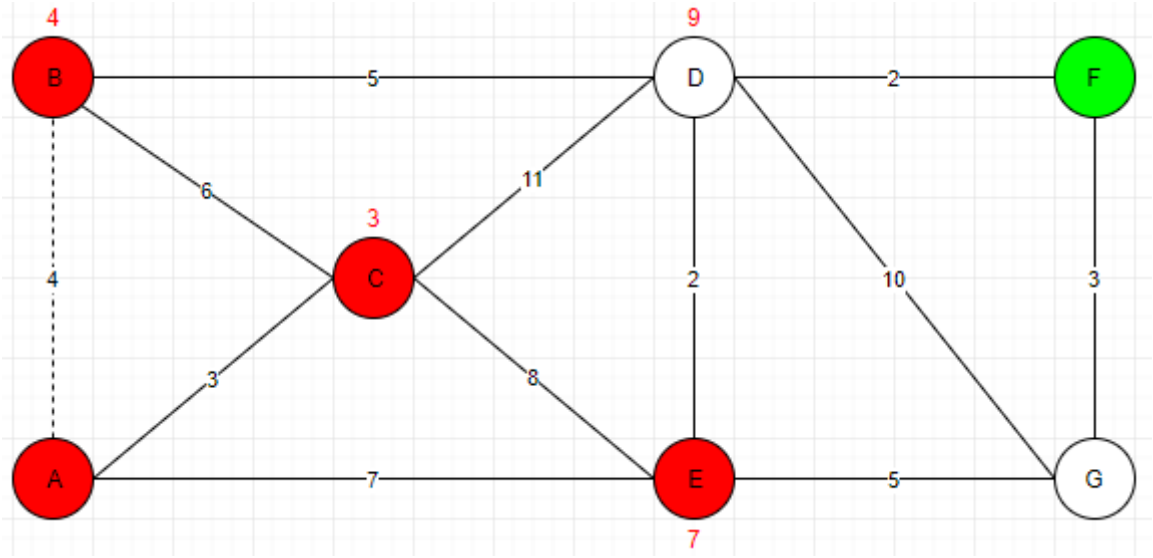
Od čvora *C* do čvora *B* ukupna vrijednost puta bi iznosila devet, budući da već postoji vrijednost četiri na čvoru *B*, taj čvor se ignorira. Također od čvora *C* do čvora *E* vrijednost bi bila jedanaest što je veće od trenutne vrijednosti sedam tako se i taj čvor ignorira. Od čvora *C* do čvora *D* vrijednost je četrnaest. Trenutna najmanja vrijednost je četiri na čvoru *B*, tako da se pomiče od čvora *A* do čvora *B* kao što je prikazano na slici 13.



Slika 13. Pomak na čvor B

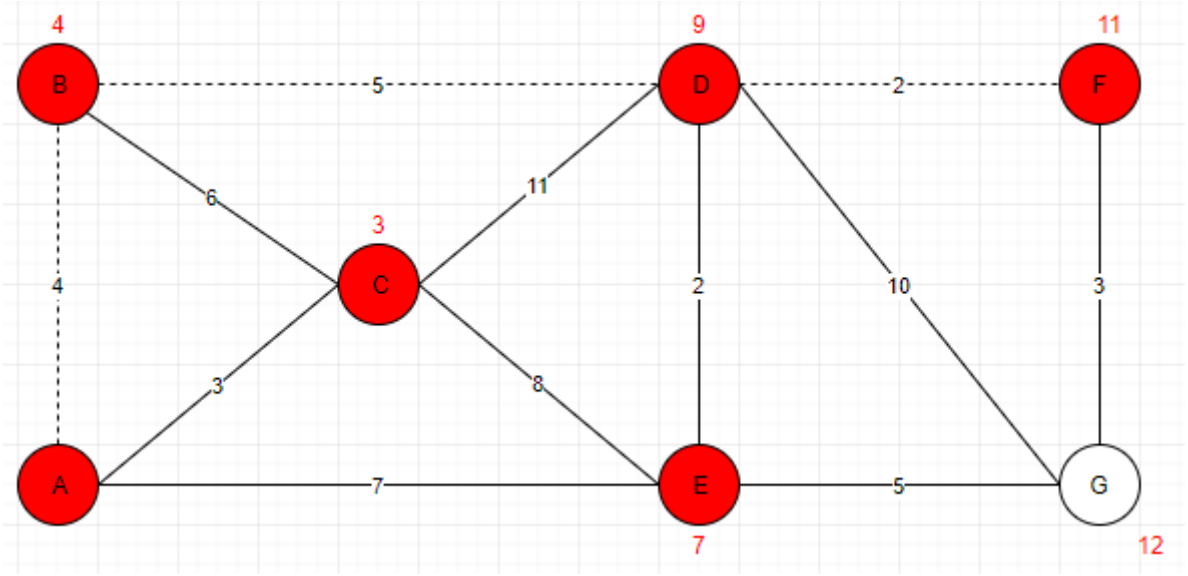
Sada u čvoru *B* neposjećeni čvorovi su *E* i *D*. Čvor *D* trenutno ima vrijednost četrnaest, ali iz čvora *B* se do čvora *D* može doći za ukupnu vrijednost devet. Tako da se

vrijednost čvora *D* mijenja na devet. Čvor *E* ima najmanju vrijednost sedam. Tako da se pomiče sa čvora *A* na čvor *E*, kao što je prikazano na slici 14.



Slika 14. Pomak na čvor E

Slika 15 prikazuje da se iz čvora *E* izračunala vrijednost za čvor *G*, a to je dvanaest. Iz čvora *E* i iz čvora *B* pomak na čvor *D* je devet, što je trenutno najmanja vrijednost, tako da se pomiče na čvor *D*.



Slika 15. Pomak na čvorove D i F

Nakon što se pomaknulo u čvor *D*, dobivamo vrijednost odredišnog čvora *F* a to je jedanaest. Budući da je vrijednost čvora *G* dvanaest, a čvora *F* jedanaest pomiče se odmah u odredišni čvor *F*. Najkraći put do čvora *F* je *A, B, D, F*.

3.2 Floyd – Warshallov algoritam

Floyd-Warshallov algoritam (također poznat kao Floydov algoritam, Roy-Warshallov algoritam,) je algoritam grafičke analize za pronalaženje najkraćih puteva u težinskom grafu s pozitivnim ili negativnim težinama bridova (ali bez negativnih ciklusa). Jedno pokretanje algoritma će naći duljine (sažete težine) najkraćih puteva između svih parova čvorova, iako ne vraćaju detalje o samim putevima. Algoritam je primjer dinamičkog programiranja. Objavio ga je u svom trenutno priznatom obliku Robert Floyd 1962. godine. Međutim, u suštini je isti kao i algoritmi koje su prethodno objavili Bernard Roy 1959. i Stephen Warshall 1962. godine za pronalaženje tranzitivnog zatvaranja grafa. Modernu formulaciju Warshallovog algoritma kao tri ugnježđene petlje prvi je opisao Peter Ingerman, također 1962. godine.[9]

Floyd-Warshallov algoritam može se koristiti za rješavanje sljedećih problema:

• Najkraći put u usmjerenim grafovima (Floydov algoritam),

• Prijelazno zatvaranje usmjerenih grafova (Warshallov algoritam). U Warshallovoj izvornoj formulaciji algoritma, graf je težinski i prikazan je matričnom logikom Booleova susjedstva. Zatim se zamjenjuje postupak dodavanja logičkim spojem (AND) i minimalnim radom logičkim razdvajanjem (OR),

• Testiranje je li neizravni graf bipartitan,

• Brzo izračunavanje Pathfinder mreža,

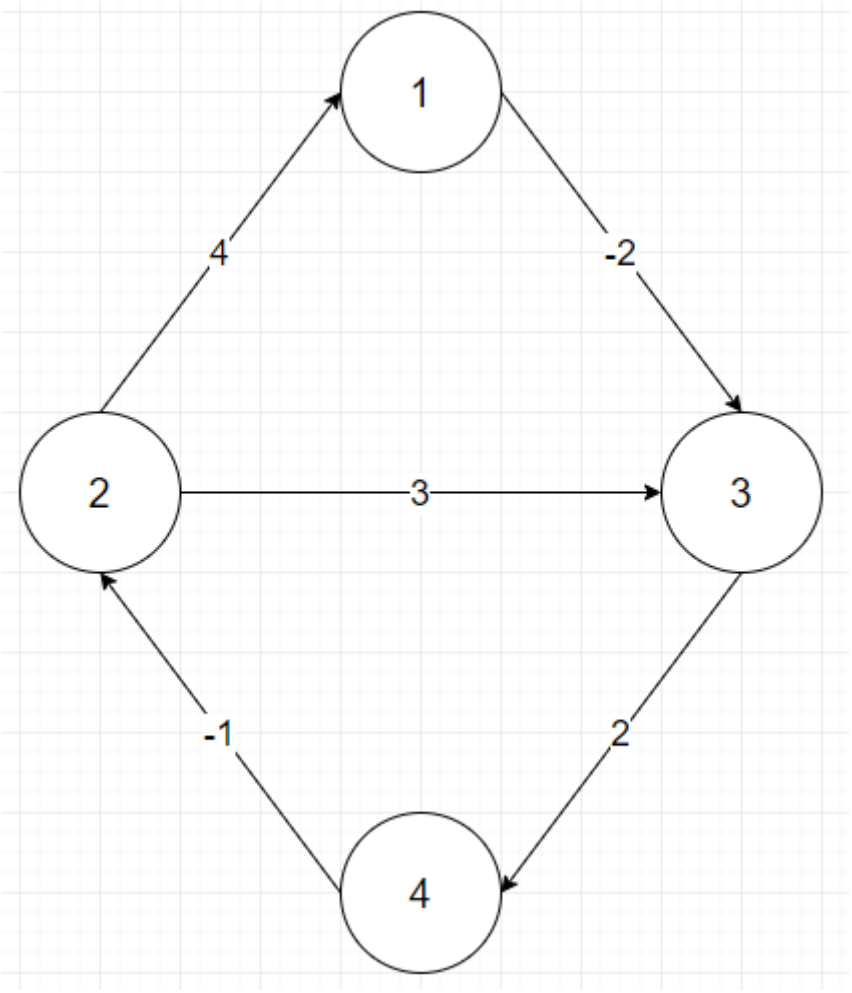
• Najširi putevi / Maksimalni putevi propusnosti.[1]

```
1 neka udaljenost bude V × V polje minimalnih udaljenosti postavljenih
u ∞ (beskonačno)
2 za svaki čvor v
3  udaljenost[v][v] ← 0
4 za svaki brid (u,v)
5  udaljenost[u][v] ← w(u,v) // težina brida (u,v)
6 za k od 1 do V
7  za i od 1 do V
8    za j od 1 do V
9     ako je udaljenost[i][j] > udaljenost[i][k] + udaljenost[k][j]
10       udaljenost[i][j] ← udaljenost[i][k] + udaljenost[k][j]
11     kraj
```

Algoritam 2. Pseudokod Floyd – Warshallovog algoritma[1]

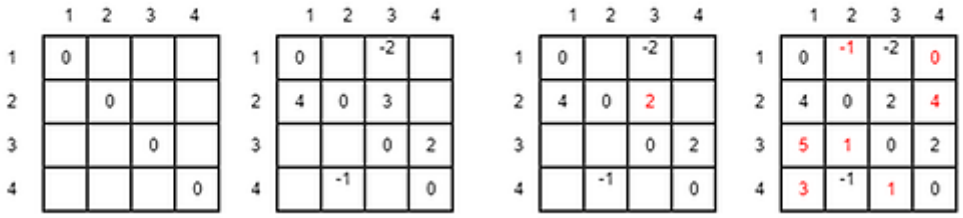
Algoritam 2 prikazuje pseudokod Floyd – Warshallovog algoritma koji funkcionira na sljedeći način. Napravi se polje *V x V*, gdje *V* predstavlja broj čvorova u grafu. Zatim se inicijalizira put svakog čvora do samog sebe na *0*. Zatim se u linijama 4 i 5 određuju težine bridova između čvorova, a one su jednake udaljenostima između čvorova. Zatim se za vrijednosti *k, i* i *j* koje simboliziraju čvorove grafa, gdje je njegova vrijednost od 1 do ukupnog broja čvorova *V*, provjerava da li je udaljenost od čvora *i* do čvora *j*, veća od zbroja udaljenosti od čvora *i* do čvora *k* i udaljenosti od čvora *k* do čvora *j*. Ako je udaljenost veća i uvjet je zadovoljen, udaljenost od čvora *i* do čvora *j* dobiva novu vrijednost a to je zbroj udaljenosti od čvora *i* do čvora *k* i udaljenosti od čvora *k* do čvora *j*. Nakon što se odrede sve udaljenosti algoritam je završio.

Floyd – Warshallov algoritam pobliže će se objasniti na primjeru grafa *G* sastavljenog od 4 čvora, prikazanog na slici 16.



Slika 16. Graf G pomoću kojeg je objašnjen Floyd – Warshallov algoritam

Da bi se izračunao najkraći put od jedne točke do druge, potrebno je kreirati *V* * *V* tablicu, gdje *V* predstavlja broj čvorova. U ovom slučaju *V* je jednako 4. Kreira se tablica i postavlja se udaljenost čvorova samih od sebe na 0. Zatim se napravi krug po grafu i upišu se inicijalne vrijednosti u tablicu. Zatim se gleda da li je uvjet $dist_{[i][j]} > dist_{[i][k]} + dist_{[k][j]}$, gdje *i*, *k* i *j* predstavljaju čvorove grafa, zadovoljen. Razmatrat će se slučajevi gdje su vrijednosti *i*, *k* i *j* od 1 do 4. Radi jednostavnosti prikaza, na slici 17, razmotrit će se samo slučajevi koji zadovoljavaju uvjet.

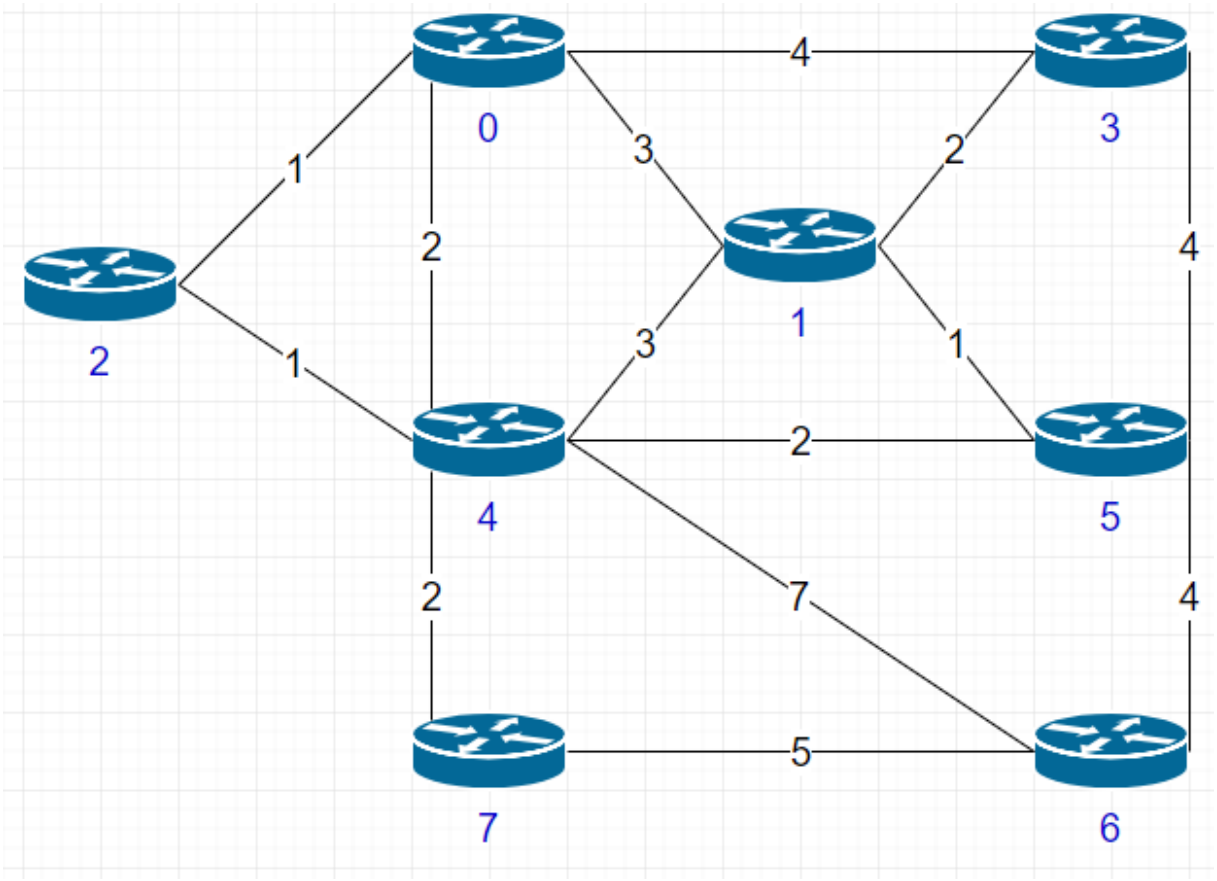


Slika 17. Floyd – Warshall algoritam, prikazan tablicom

Prvi slučaj gdje se zadovoljava uvjet je kod vrijednosti $i = 2, j = 3$ i $k = 1$. $dist_{[i][j]} > dist_{[i][k]} + dist_{[k][j]}$, $3 > 4 + -2, 3 > 2$. i, j i k simboliziraju retke i stupce u tablici, gdje se promatra da li je iznos u polju $dist_{[i][j]}$ veći od zbroja $dist_{[i][k]}$ i $dist_{[k][j]}$. U ovom slučaju uvjet je zadovoljen tako da vrijednost tri prelazi u vrijednost dva. Te se dalje po istom principu popunjava tablica i dobivaju se konačni rezultati.

4 IMPLEMENTACIJA ALGORITAMA

Zadatak ovog rada je implementirati algoritam za traženje najkraćeg puta u grafu u nekoj telekomunikacijskoj mreži. Implementirat će se Dijkstrin algoritam u mreži s komutacijom paketa, da bi se odredio put s najmanjom količinom kašnjenja, odnosno da bi se odredio iznos kašnjenja od točke A do točke B. Algoritam je izveden u programskom jeziku Java.



Slika 18. Primjer mreže sa komutacijom paketa

Na slici 18 možemo vidjeti graf koji simbolizira čvorišta u mreži sa komutacijom paketa, odnosno usmjernike koji su označeni s plavim ikonama sa četiri strelice. Na bridovima grafa prikazana je količina kašnjenja između svakog čvora. Cilj je odrediti minimalnu količinu kašnjenja od čvora 0 do svih ostalih čvorova.

4.1 Java programski jezik

Java je programski jezik razvijen početkom 90-ih godina prošlog stoljeća u tvrtki Sun Microsystems. Osnovna ideja bila je razviti novi jezik koji bi sintaksom sličio tadašnjim najpopularnijim jezicima C i C++, ali koji bi s druge strane donio i neka nova svojstva. Jedno od najvažnijih svojstava Jave je portabilnost. Ideja je da se Java aplikacije razvijene na jednom operativnom sustavu mogu nepromijenjene izvršavati i na ostalim operativnim sustavima koji podržavaju Javu. Na taj je način ušteda vremena i resursa kod razvoja i održavanja velikih aplikacija na različitim platformama velika.

Java je objektno orijentirani programski jezik. Objektno orijentirane aplikacije jednostavnije su za razumijevanje i održavanje od aplikacija koje nisu razvijane objektno orijentiranim pristupom.[11]

4.1.1 Varijable i izrazi

Varijabla je dio memorije kojem se daje neko simboličko ime i koja može čuvati vrijednost odgovarajućeg podatkovnog tipa. Najčešći oblici varijabla su: cjelobrojne, decimalne, znakovne i logičke.

Postoje četiri vrste različitih varijabli u Javi koje možemo spremiti u cjelobrojne vrijednosti, a to su *Byte*, *Short*, *Integer* i *Long*. Najčešće je korištena varijabla Integer. Varijabla Integer može spremiti cijeli broj u rasponu od -2147483648 do 2147483647 i njeno zauzeće memorije iznosi četiri *byta*.

Da bi se varijabla mogla koristiti ona se prvo mora deklarirati, (npr. int *pocetniCvor*;), ovim kodom u Javi se rezervira prostor u memoriji pod nazivom *pocetniCvor* i u njega možemo spremiti vrijednost oblika *Integer*.

Operacije koje se u Javi mogu koristiti nad cjelobrojnim vrijednostima su zbrajanje, oduzimanje, množenje, cjelobrojno dijeljenje i ostatak pri dijeljenju. Jako korisne u Javi su i logičke varijable. Varijabla boolean koja može imati samo dvije različite vrijednosti, istina i laž. Koristi se kada želimo da se neki dio koda izvede, ako je neka tvrdnja istinita ili lažna.[6]

4.1.2 Petlje i kontrola tijeka

Najčešće korištene petlje i iskazi kontrole tijeka su iskazi *if – else* i *switch*, te petlje *While*, *Do While* i *for*. Iskaz *if - else* opisuje grananje temeljeno na logičkom uvijetu. Logički uvjet je izraz koji se testira na početku iskaza *if* i kao rezultat vraća logičku konstantu istina ili laž. Samo ako je taj uvjet istinit, izvodi se iskaz koji slijedi. Iskaz se zapisuje navođenjem ključne riječi *if*, iza koje unutar zagrada slijedi izraz koji se testira te zatim iskaz koji se izvršava ako je uvjet istinit. Ako logički uvjet nije istinit, tijelo *if* izraza se preskače.

```
if (logicki_izraz){
Iskaz;
}
```

Kod rješavanja raznih problema često se pojavljuje potreba za ponavljanjem jedne ili više radnji. Radnje se ponavljaju određen broj puta, sve dok logički uvjet ne bude ispunjen ili sve dok se zbog bilo kojeg razloga ne prekine ponavljanje radnji. U takvim situacijama koriste se petlje. *While* petlja se koristi kada ne znamo broj ponavljanja. Na primjer treba se ispisati prva četiri cijela broja.

```
int n = 1;
while (n<5){
System.out.println(n);
n = n+1;
}
```

Prvo se deklarira varijabla *n* tipa *Integer* i postavi se vrijednost u jedan. Zatim *While* petlja radi sljedeće. Dok je zadovoljen uvjet da je broj manji od pet, petlja se pokreće. Prvo u petlju ulazi broj jedan, on se ispisuje (funkcija *System.out.println* ispisuje izraz unutar zagrada), te se broj jedan zbraja s jedan te tako postaje dva. Kako je broj dva i dalje manji od pet, opet se pokreće petlja. Ulazi broj dva, ispisuje se i zbraja s jedan te tako postaje tri. I tako redom dok se ne dođe do broja pet, kako broj pet nije manji od pet, uvjet nije zadovoljen i petlja se više ne pokreće.

For petlja donosi alternativni način definiranja iteracija. Iako ne donosi ništa funkcionalno novo u odnosu na *While* petlju, mnogo je bolja za rješavanje situacija kad se zna broj ponavljanja. Petlja *for* ima sljedeću strukturu:

for (inicijalizator; logicki_izraz; iterator){

}

Inicijalizator definira početnu vrijednost kontrolne varijable, *logicki_izraz* mora biti istinit da bi se petlja izvršavala, te *iterator* mijenja vrijedost kontrolne varijable.

Na primjer želimo riječ graf ispisati deset puta.

for (int i = 0; i < 10; i++){

System.out.println("Graf");

}

Petlja radi sljedeće. Za vrijeme dok je varijabla *i* manja od deset izvršava naredbu unutar iskaza. Naredba je u ovom slučaju da se ispiše riječ Graf. Prva vrijednost varijable *i* jednaka je nula, uvjet da je *i* manje od deset je zadovoljen i petlja se pokreče, zatim se varijabla poveća za jedan. I tako redom dok uvjet da je varijabla *i* manja od deset bude neistinit.[6]

4.1.3 Polja

U prethodnim poglavljima opisali su se tipovi podataka kod kojih svaki identifikator odgovara jednoj varijabli. Postoje situacije kada se želi obraditi skup vrijednosti istog tipa, tada se koristi polje. Polje je objekt koji čuva skup vrijednosti. Ono ima svoj naziv koji je identifikator kao i za bilo koju drugu varijablu. Taj naziv može se gledati kao zajedničko ime cijelog skupa elemenata. Za svaki element tog skupa kaže se da je element polja. Elementima u polju pristupa se koristeći njihovu poziciju odnosno indeks. Prvi element ima poziciju 0 i tako do pozicije n-1, ako se s n označi ukupan broj elemenata u polju. Polje se u kodu kreira na sljedeći način:

int [] čvorovi = new int [5];

Na ovaj način kreiralo se polje imena *čvorovi*, cjelobrojnog tipa, koje sadrži 5 elemenata.

Elementima u polju pristupa se na sljedeći način. U polje čvorovi dodaje se 5 elemenata.

int[] čvorovi = {5, 3, 4, 2, 7};

Elementi se nalaze redom na pozicijama, pet na poziciji 0, tri na poziciji *1*, četiri na poziciji *2*, dva na poziciji *3* i sedam na poziciji *4*. Četvrtom elementu polja, gdje je zapisana vrijednost dva, pristupamo preko indeksa *3*:

čvorovi [3];

Upiše se ime polja i zatim u uglatim zagradama pozicija elementa.[6]

4.1.4 Klase

Klasa je predložak za kreiranje pojedinih vrsta objekata. Klasa također definira i novi tip podatka. Objekti kreirani po predlošku neke klase sadržavaju sve sastavnice navedene u toj klasi. Na klase i objekte možemo gledati kao na tipove i varijable. Jedno od značajnijih obilježja klasa je da su u njima osobine i sposobnosti nekog pojma učahurene (*engl. Encapsulated*) u jedinstvenoj jedinici programskog koda. Osnovni elementi klase su metode i polja. Pojam polja spomenut u kontekstu klasa u prethodnoj rečenici nije isti kao pojam polje opisan u poglavlju 4.1.3. Dakle, polje u kontekstu definicije klase ne predstavlja niz elemenata, nego varijablu definiranu u klasi, tzv. člansku varijablu. Objasnit će se na primjeru klase *Brid*.[6]

Public class Brid {

Private int pocetniCvor;

Private int zavrsniCvor;

Private int kasnjenje;

Varijable *pocetniCvor*, *zavrsniCvor* i *kasnjenje* su polja u prethodnom primjeru klase *Brid*.

U programskom jeziku Java klasa se deklarira pomoću modifikatora pristupa, zatim ključne riječi Class nakon koje slijedi proizvoljan naziv klase. Unutar blokova vitičastih zagrada slijede definicije članova od kojih se sastoji tijelo klase.

Modifikator pristupa definira dostupnost elemenata klase prema drugim objektima. U prethodnom primjeru modifikator pristupa *public* ispred ključne riječi *class* označava klasu *Brid* kao javno dostupnu.

Pod tim se podrazumijeva da bilo koji objekt iz bilo kojeg paketa može vidjeti klasu *Brid* i iz nje instancirati novi objekt. No suprotno tomu, modifikator pristupa *private* čini da elementi klase ne budu javno dostupni, nego da su privatni i da se mogu koristiti jedino unutar te određene klase.

4.2 Bridovi

Prvo što moramo napraviti je stvoriti Java klasu *Brid*. Svaki brid ima svoju vrijednost koja simbolizira kašnjenje između dva čvora koja povezuje.

```
package com.company;

public class Brid {


    private int pocetniCvor;
    private int zavrsniCvor;
    private int kasnjenje;

    public Brid(int pocetniCvor, int zavrsniCvor, int kasnjenje) {

        this.pocetniCvor = pocetniCvor;
        this.zavrsniCvor = zavrsniCvor;
        this.kasnjenje = kasnjenje;


    }

    public int getPocetniCvor() { return pocetniCvor; }

    public int getZavrsniCvor() { return zavrsniCvor; }

    public int getKasnjenje() { return kasnjenje; }

    public int getSusjedniČvor(int čvor) {
        if (this.pocetniCvor == čvor) {
            return this.zavrsniCvor;
        } else {
            return this.pocetniCvor;
        }
    }

}
```

Slika 19. Java kod klase Brid

Slika 19 prikazuje Java klasu *Brid*. Inicijaliziraju se varijable *pocetniCvor* i *zavrsniCvor* koje simboliziraju početni čvor i završni čvor brida, te varijabla kašnjenje, koja je jednaka težini brida, ali u ovom slučaju simbolizira kašnjenje u mreži. Kreira se klasa *Brid* koja se sastoji od varijabla tipa *Integer pocetniCvor*, zavrsniCvor i kašnjenje.

4.3 Čvorovi

Nakon što su se definirali bridovi, potrebno je isprogramirati čvorove grafa. Klasa *Čvor*, prikazana na slici 20, mora sadržavati listu bridova koji su povezani snjim. Uz to mora sadržavati polje kojim se određuje da li je čvor bio posjećen ili ne.

```
import java.util.ArrayList;

public class Čvor {

    private int udaljenostOdIzvora = Integer.MAX_VALUE;
    private boolean posjecen;
    private ArrayList<Brid> bridovi = new ArrayList<Brid>();

    public int getUdaljenostOdIzvora() { return udaljenostOdIzvora; }

    public void setUdaljenostOdIzvora(int udaljenostOdIzvora) {
        this.udaljenostOdIzvora = udaljenostOdIzvora;
    }

    public boolean isPosjecen() { return posjecen; }

    public void setPosjecen(boolean posjecen) { this.posjecen = posjecen; }

    public ArrayList<Brid> getBridovi() { return bridovi; }

    public void setBridovi(ArrayList<Brid> bridovi) { this.bridovi = bridovi; }
}
```

Slika 20. Java kod klase Čvor

Inicijaliziraju se varijable *udaljenostOdIzvora*, te varijabla posjećen koja je tipa *boolean*, odnosno istina ili laž.

4.4 Graf

Sada kada su definirani čvorovi i bridovi, može se definirati graf koji mora sadržavati čvorove i bridove. Također na kraju ove klase nalazi se i naredba za ispis rezultata.

```
public class Graf {


    private Čvor [] čvorovi;
    private int brojČvorova;
    private Brid [] bridovi;
    private int brojBridova;


    public Graf(Brid[] bridovi) {
        this.bridovi = bridovi;
        this.brojČvorova = izracunajBrojČvorova(bridovi);
        this.čvorovi = new Čvor[this.brojČvorova];

        for (int i = 0; i < this.brojČvorova; i++) {
            this.čvorovi[i] = new Čvor();
        }

        this.brojBridova = bridovi.length;

        for (int dodajBrid = 0; dodajBrid < this.brojBridova; dodajBrid++) {
            this.čvorovi[bridovi[dodajBrid].getPocetniCvor()].getBridovi().add(bridovi[dodajBrid]);
            this.čvorovi[bridovi[dodajBrid].getZavrsniCvor()].getBridovi().add(bridovi[dodajBrid]);
        }

    }

    private int izracunajBrojČvorova(Brid[] bridovi) {
        int brojČvorova = 0;

        for (Brid e : bridovi) {
            if (e.getZavrsniCvor() > brojČvorova)
                brojČvorova = e.getZavrsniCvor();
            if (e.getPocetniCvor() > brojČvorova)
                brojČvorova = e.getPocetniCvor();
        }

        brojČvorova++;

        return brojČvorova;
```

Slika 21. Java kod klase Graf

Slika 21 prikazuje izradu grafa u Java kodu. Prvo se napravi klasa *Graf*, koji sadrži polje *Čvor* i polje *Brid*, ta dva polja su prethodno definirana u klasama *Čvor* i *Brid*.

Zatim kreiramo objekt *Graf* u kojem će biti implementiran Dijkstrin algoritam. Sada se moraju kreirati čvorovi i povezati bridovi i čvorovi. Postavlja se vrijednost varijable *brojCvorova* na rezultat implementirane metode *izracunajBrojCvorova*. Metoda radi na

sljedeći način. Varijabla *brojCvorova* postavlja se na nula. Zatim *for* petlja prolazi kroz polje *Brid* i za svaki element u polju ako je vrijednost brida *zavrsniCvor* veća od trenutnog broja čvorova, varijablu *brojCvorova* postavlja na taj iznos. Isto tako radi i za varijablu *pocetniCvor*, zato što varijable *pocetniCvor* i *zavrsniCvor* zapravo predstavljaju čvorove. Kada *for* petlja prođe jednu iteraciju *brojČvorova* poveća za jedan i tako ponavlja sve dok varijable *pocetniCvor* i *zavrsniCvor* ne budu jednake varijabli *brojČvorova*. Na kraju metoda vraća krajnji broj čvorova.

Nakon što metoda izračuna broj čvorova, *for* petlja puni polje *Čvor* tako da kreće od nula i sve dok je zadovoljen uvjet da je sljedeći broj manji od maksimalnog broja čvorova upisuje čvorove u polje. Kada su se povezali bridovi i čvorovi može se krenuti s implementacijom algoritma kao što je prikazano slikama 22 i 23.

```
public void dijkstra() {

    this.čvorovi[0].setUdaljenostOdIzvora(0);
    int sljedeciČvor = 0;


    for (int i = 0; i < this.čvorovi.length; i++) {

        ArrayList<Brid> trenutniBridČvora = this.čvorovi[sljedeciČvor].getBridovi();

        for (int pridruzeniBrid = 0; pridruzeniBrid < trenutniBridČvora.size(); pridruzeniBrid++) {
            int susjedniČvor = trenutniBridČvora.get(pridruzeniBrid).getSusjedniČvor(sljedeciČvor);


            if (!this.čvorovi[susjedniČvor].isPosjecen()) {
                int privremeni = this.čvorovi[sljedeciČvor].getUdaljenostOdIzvora()
                        + trenutniBridČvora.get(pridruzeniBrid).getKasnjenje();

                if (privremeni < čvorovi[susjedniČvor].getUdaljenostOdIzvora()) {
                    čvorovi[susjedniČvor].setUdaljenostOdIzvora(privremeni);
                }
            }
        }


        čvorovi[sljedeciČvor].setPosjecen(true);


        sljedeciČvor = najbliziCvor();
```

Slika 22. Implementacija Dijsktrinog algoritma

```
private int najbliziCvor() {
    int spremljeniČvor = 0;
    int spremljenaUdaljenost = Integer.MAX_VALUE;

    for (int i = 0; i < this.čvorovi.length; i++) {
        int trenutnaUdaljenost = this.čvorovi[i].getUdaljenostOdIzvora();

        if (!this.čvorovi[i].isPosjecen() && trenutnaUdaljenost < spremljenaUdaljenost) {
            spremljenaUdaljenost = trenutnaUdaljenost;
            spremljeniČvor = i;
        }
    }

    return spremljeniČvor;
}

public void printResult() {
    String IZLAZ = "Broj čvorova = " + this.brojČvorova;
    IZLAZ += "\nBroj putanja = " + this.brojBridova;

    for (int i = 0; i < this.čvorovi.length; i++) {
        IZLAZ += ("\nNajmanja količina kašnjenja od čvora 0 do čvora " + i + " je "
                + čvorovi[i].getUdaljenostOdIzvora());
    }

    System.out.println(IZLAZ);
}
public Čvor[] getČvorovi() { return čvorovi; }
public int getBrojČvorova() { return brojČvorova; }
public Brid[] getBridovi() { return bridovi; }
public int getBrojBridova() { return brojBridova; }
```

Slika 23. Nastavak implementacije Dijsktrinog algoritma

Početni čvor se postavlja na nulu. Zatim *for* petlja kruži oko bridova trenutnog čvora, te pomoću sljedeće *for* petlje traži najmanju vrijednost. Petlja radi tako da za vrijeme dok je zadovoljen uvjet da je trenutni pridruženi brid manje vrijednosti od trenutnog brida, taj pridruženi brid postaje put do sljedećeg susjednog čvora.

Zatim ako taj trenutni najmanji čvor nije posjećen on se postavlja kao sljedeći susjedni čvor na kojeg se pomiče.

Metoda *najbliziCvor* radi tako da prvo čvor postavi na vrijednost nula. Zatim za svaki čvor ako vrijedi da je čvor manji od ukupnog broja čvorova uzima se da je taj čvor trenutna udaljenost od izvora. Zatim ako taj čvor zadovoljava uvjet da nije posjećen i da je trenutna udaljenost manja od maksimalne spremljene udaljenosti, taj čvor postaje najbliži.

4.5 Upisivanje vrijednosti s grafa

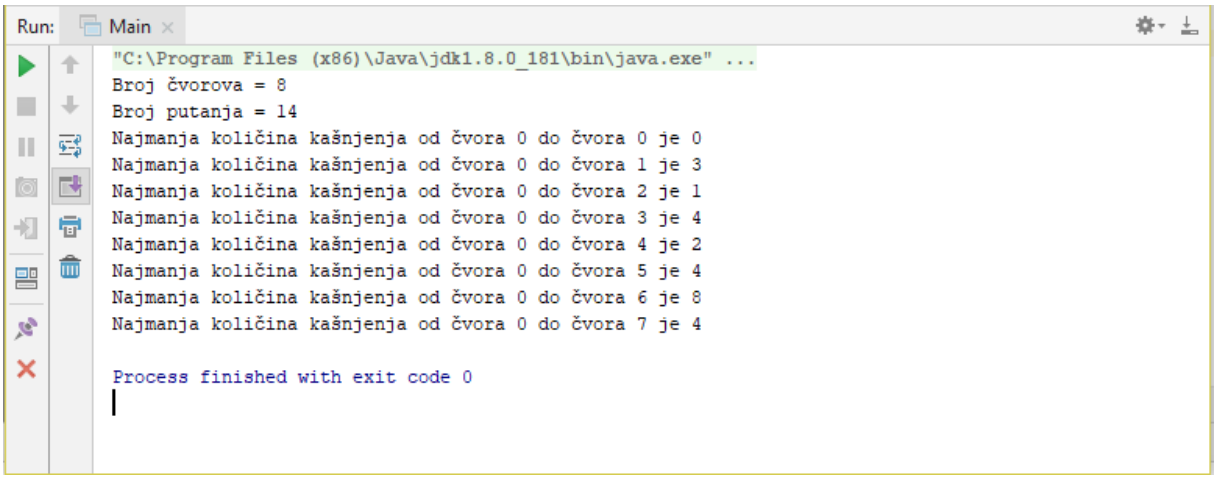
```
public class Main {

    public static void main(String[] args) {

        Brid[] bridovi = {
                new Brid( pocetniCvor: 0,  zavrsniCvor: 2,  kasnjenje: 1),
                new Brid( pocetniCvor: 0,  zavrsniCvor: 3,  kasnjenje: 4),
                new Brid( pocetniCvor: 0,  zavrsniCvor: 4,  kasnjenje: 2),
                new Brid( pocetniCvor: 0,  zavrsniCvor: 1,  kasnjenje: 3),
                new Brid( pocetniCvor: 1,  zavrsniCvor: 3,  kasnjenje: 2),
                new Brid( pocetniCvor: 1,  zavrsniCvor: 4,  kasnjenje: 3),
                new Brid( pocetniCvor: 1,  zavrsniCvor: 5,  kasnjenje: 1),
                new Brid( pocetniCvor: 2,  zavrsniCvor: 4,  kasnjenje: 1),
                new Brid( pocetniCvor: 3,  zavrsniCvor: 5,  kasnjenje: 4),
                new Brid( pocetniCvor: 4,  zavrsniCvor: 5,  kasnjenje: 2),
                new Brid( pocetniCvor: 4,  zavrsniCvor: 6,  kasnjenje: 7),
                new Brid( pocetniCvor: 4,  zavrsniCvor: 7,  kasnjenje: 2),
                new Brid( pocetniCvor: 5,  zavrsniCvor: 6,  kasnjenje: 4),
                new Brid( pocetniCvor: 6,  zavrsniCvor: 7,  kasnjenje: 5)
        };

        Graf g = new Graf(bridovi);
        g.dijkstra();
        g.printResult();
```

Slika 24. Java kod kreiranja grafa

Slika 24 prikazuje izradu grafa prema zadanim vrijednostima. U klasi *Main* kreira se polje *Brid* klase *Bridovi* i puni se vrijednostima zadanim na grafu. Zatim se pozivaju metode *dijkstra* i metoda *printResult* za ispis rezultata.

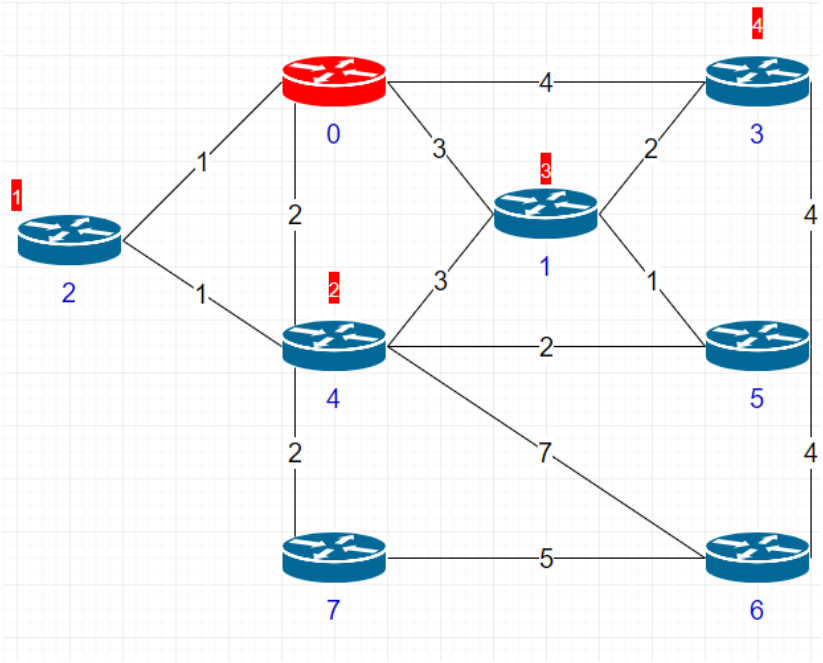
5 REZULTATI

Program je pisan u razvojnom okruženju *InteliJ*, te za kod koji je prikazan u prethodnom poglavlju ima sljedeći ispis rezultata prikazan slikom 25.



Slika 25. Ispis rezultata u *InteliJ* razvojnom okruženju

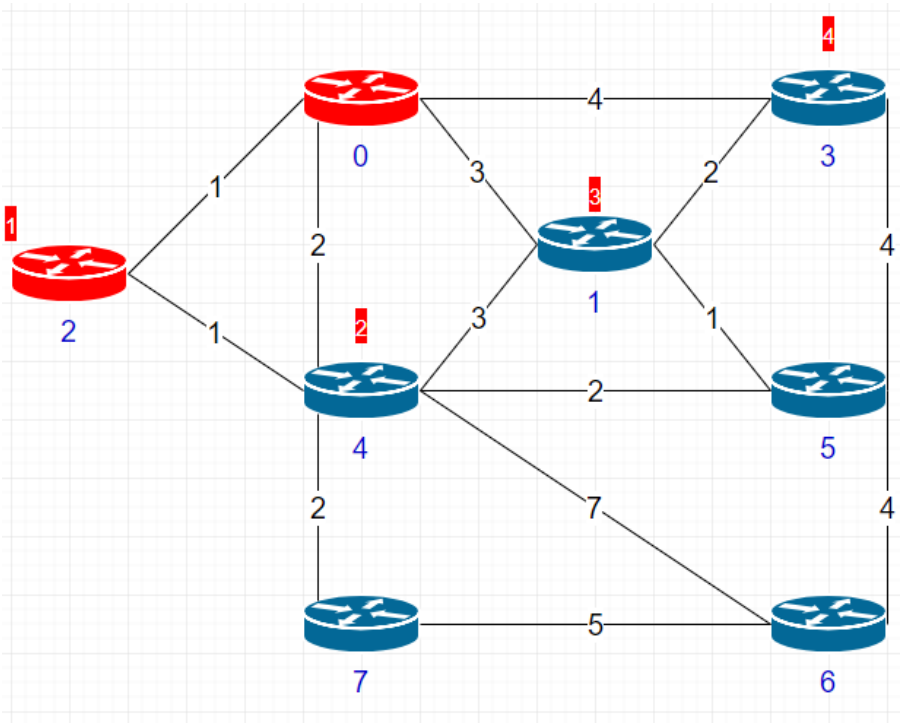
Sada će se provjeriti da li je dobro izračunata vrijednost kašnjenja od čvora 0 do čvora 7.



Slika 26. Provjera rezultata programa

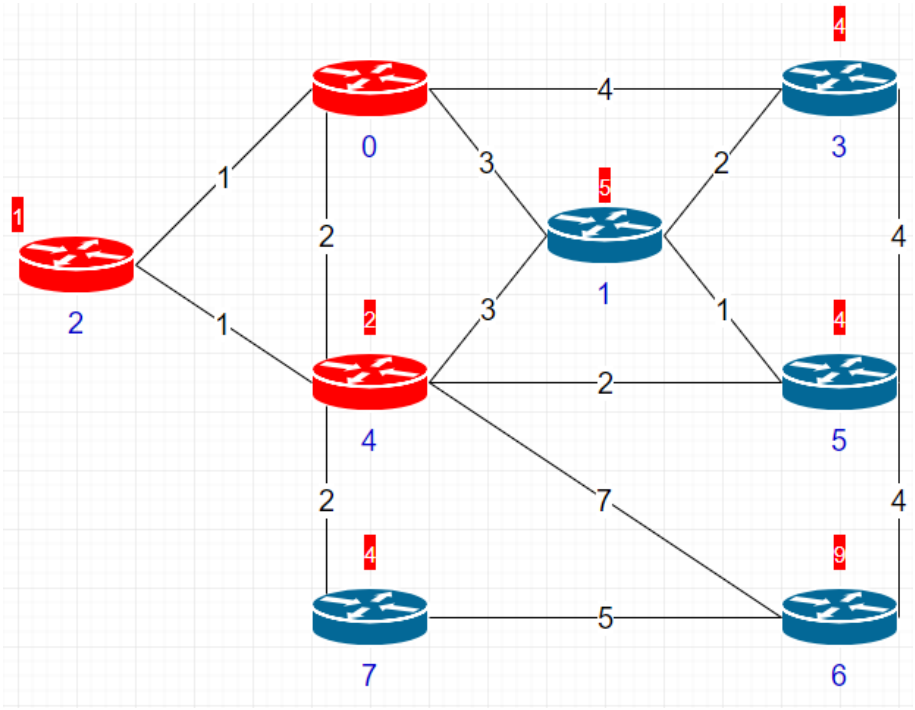
Na slici 26 je prikazano da je početna točka, točka *0*. Dalje se gleda koji susjedni čvor ima najmanju vrijednost, odnosno brid koji ih povezuje. Vrijednosti čvorova označene su bijelim

brojem sa crvenom pozadinom. Čvor *1* ima vrijednost tri, čvor *2* ima vrijednost jedan, čvor *3* ima vrijednost četiri, čvor *4* ima vrijednost četiri. Budući da je najmanja vrijednost u čvoru *2*, na njega se pomičemo.



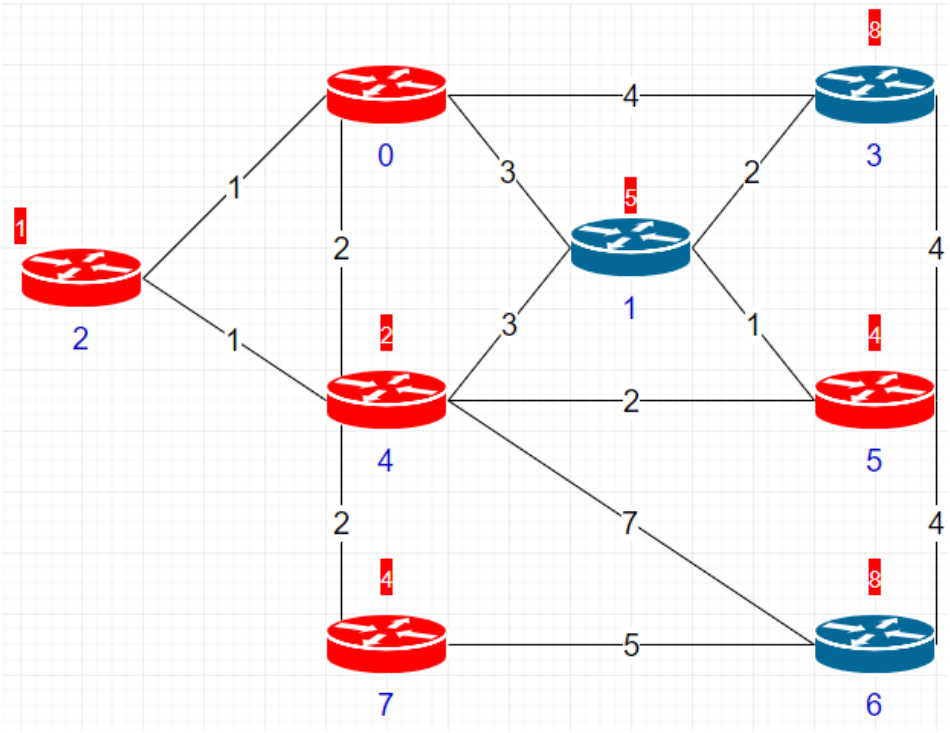
Slika 27. Provjera rezultata, čvor 2

Slika 27 prikazuje poziciju na čvoru *2*, iz čvora *2* najmanja vrijednost je u čvoru *4* te ona iznosi dva, tako da se pomičemo na čvor *4*.



Slika 28. Provjera rezultata, čvor 4

Slika 28 prikazuje poziciju na čvoru *4*. Trenutne vrijednosti susjednih čvorova su sljedeće: čvor *1* ima vrijednost pet, čvor *5* ima vrijednost četiri, čvor *6* ima vrijednost devet i čvor *7* ima vrijednost četiri. Budući da čvorovi *5* i *7* imaju trenutno najmanje vrijednosti na njih se pomičemo.



Slika 29. Provjera rezultata, čvorovi 5 i 7

Slika 29 prikazuje pozicije na čvorovima *5* i *7*. Iz čvora *5*, čvor *3* ima vrijednost osam, čvor *1* ima vrijednost pet i čvor *6* ima vrijednost osam. Budući da su sve vrijednosti veće od četiri koliko trenutno ima odredišni čvor *7*, našli smo njegovu najmanju vrijednost i ona iznosi četiri. Ranije u tekstu na slici 24 je vidljivo da je program također izračunao da je najmanja količina kašnjenja za čvor *7* jednaka četiri.

6 ZAKLJUČAK

Spoj Java programskog jezika i Dijsktrinog algoritma može biti koristan u svim granama prometa. Kao što se pokazalo u ovom radu vrlo brzo se može izračunati kašnjenje unutar telekomunikacijske mreže, isto tako bi se lako mogla izračunati najkraća udaljenost između dva grada cestovnom mrežom.

Internet omogućuje ljudima diljem svijeta da brzo pristupe i povuku veliku količinu podataka i informacija. Kako bi to mogli učiniti, pametni algoritmi se koriste kako bi se upravljalo i manipuliralo tom velikom količinom podataka. Primjeri problema koji se mogu pojaviti i koji se moraju riješiti je problem pronalaženja dobrih ruta kojima će podatci putovati. U takvim problemima do izražaja dolaze algoritmi za traženje najkraćeg puta, te omogućuju da se internet promet odvija maksimalno brzo i učinkovito.

U radu su se objasnile implementacije Dijkstrinog algoritma i Floyd – Warshallovog algoritma za traženje najkraćih puteva u grafu, implementacije su se prikazale na primjerima manjih grafova.

Također se prikazao način implementacije Dijkstrinog algoritma na primjeru manje prometne mreže, koristeći Java programski jezik unutar *InteliJ* razvojnog okruženja. Jednako tako ovaj algoritam se može implementirati na mnogo većim mrežama u stvarnom životu.

LITERATURA

[1]http://www.cse.unt.edu/~tarau/teaching/AnAlgo/Dijkstra%27s%20algorithm.pdf[Pristupljeno: kolovoz 2018.]

[2] http://grafovi.weebly.com/uvod-u-teoriju-grafova[Pristupljeno: kolovoz 2018.]

[3] https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm#Pseudocode[Pristupljeno: kolovoz 2018.]

[4] Thomas H. Cormen, Charles R. Leiserson, Ronald L. Rivest, Clifford Stein. Introduction to Algorithms Third Edition. Massachusetts Institute of Technology 2009

[5] Keijo Ruohonen. Graph Theory. Preuzeto sa: http://math.tut.fi/ [Pristupljeno: kolovoz 2018.]

[6] Filip Tomić, Danijel Kučak. Osnove Java programiranja za Android. Algebra učilište, Zagreb 2013.

[7] Thomas H. Cormen, Charles R. Leiserson, Ronald L. Rivest, Clifford Stein. Introduction to Algorithms Second Edition. Massachusetts Institute of Technology 2001

[8] E. W. Dijkstra. A note on two problems in connexion with graphs, Mathematisch Centrum, Amsterdam 1959.

[9] Robert W. Floyd. Algorithm 97: Shortest path, Magazine Communications of the ACM, New York 1962.

[10] V. K. Balakrishnan, Schaums outline of graph theory, McGraw – Hill Education, New York 1997

[11] Tim Lindholm, Frank Yelin, Java virtual machine specification, Addison – Wesley Longman Publishing, Boston 1999.

POPIS SLIKA

POPIS ALGORITAMA

Sveučilište u Zagrebu
Fakultet prometnih znanosti
10000 Zagreb
Vukelićeva 4

IZJAVA O AKADEMSKOJ ČESTITOSTI I SUGLASNOST

Izjavljujem i svojim potpisom potvrđujem kako je ovaj završni rad isključivo rezultat mog vlastitog rada koji se temelji na mojim istraživanjima i oslanja se na objavljenu literaturu što pokazuju korištene bilješke i bibliografija.

Izjavljujem kako nijedan dio rada nije napisan na nedozvoljen način, niti je prepisan iz necitiranog rada, te nijedan dio rada ne krši bilo čija autorska prava.

Izjavljujem također, kako nijedan dio rada nije iskorišten za bilo koji drugi rad u bilo kojoj drugoj visokoškolskoj, znanstvenoj ili obrazovnoj ustanovi.

Svojim potpisom potvrđujem i dajem suglasnost za javnu objavu završnog rada pod naslovom **Pregled algoritama za traženje najkraćeg puta u grafu**

na internetskim stranicama i repozitoriju Fakulteta prometnih znanosti, Digitalnom akademskom repozitoriju (DAR) pri Nacionalnoj i sveučilišnoj knjižnici u Zagrebu.

U Zagrebu, 10.9.2018

Student/ica:

(potpis)