UDK 519.17
Stručni rad

Edin Kohnić

ALGORITMI U TEORIJI GRAFOVA (BFS, DFS I DIJKSTRIN ALGORITAM)

Sažetak

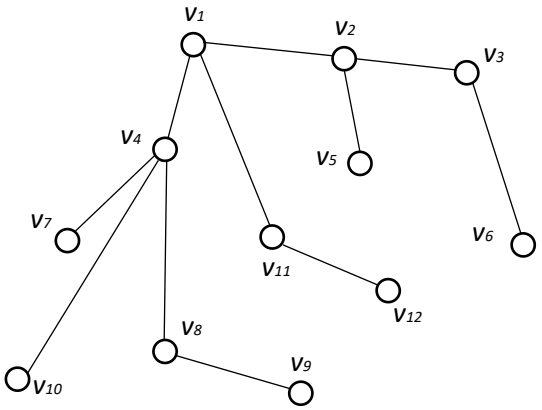
U ovom radu su predstavljeni algoritmi u teoriji grafova, i to: algoritam pretrage u dubinu, algoritam pretrage u širinu i Dijkstrin algoritam. Navedeni algoritmi su demonstrirani u vidu zapisa kodom kroz programske jezike, pored toga opisan je način na koji se svaki algoritam izvršava sa stajališta definicija izvršenja, te je istaknuta njihova praktična primjena kojom je pokazan značaj i široka upotreba navedenih algoritama.

Ključne riječi: *graf, vrh, ivica, algoritmi pretraživanja*

Algoritam pretrage u širinu – BFS

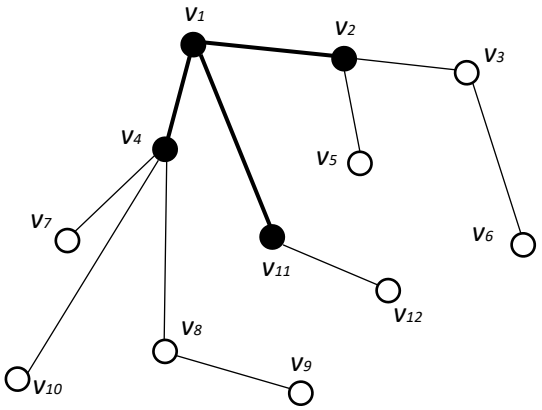
Algoritam pretrage grafa u širinu, poznatiji pod nazivom BFS (Breadth-First Search), kreiran je 1945. godine od strane njemačkog inženjera Konrada Zusea i američkog naučnika Edwarda Forrest-Moorea. [11] BFS predstavlja jedan od najjednostavnijih algoritama za pretragu grafova. Na samom početku je potrebno definisati dvije liste koje će sadržavati informacije o vrhovima koji su posjećeni i vrhovima koji tek trebaju biti posjećeni. Princip funkcionisanja algoritma se temelji na tome da se odabere jedan proizvoljan vrh grafa i označi kao početni, a zatim se vrši posjećivanje svih njegovih susjeda. Nakon toga se posjećuju svi susjedi vrhova koji su susjedi početnom vrhu i tako sve redom dok se ne posjete svi vrhovi grafa. Ako se radi o grafu koji nije povezan, svaka od komponenti grafa se posjećuje na isti način dok se ne izvrši obilazak svih komponenti datog grafa. [1] Posjećivanjem vrhova u svakoj iteraciji, moguće je dodijeliti svakom od vrhova brojnu oznaku koja odgovara oznaci iteracije. Ukoliko se početni vrh označi kao vrh *0*, tada se njegovi susjedi označavaju brojem *1*, susjedi vrhova oznake *1* označavaju se oznakom *2* i tako redom dok se ne označe svi vrhovi. Kako svaka oznaka vrha ujedno predstavlja i najkraći put od početnog vrha (vrha *0*) do tog vrha, tako se ovaj algoritam može koristiti za slučaj da je potrebno naći najkraći put od početnog vrha do nekog zadanog vrha (za graf koji nije težinski). [12]

U nastavku je dat graf na kojem će biti demonstriran algoritam pretraživanja u širinu.



Slika br. 1 Primjer- BFS pretraga na grafu

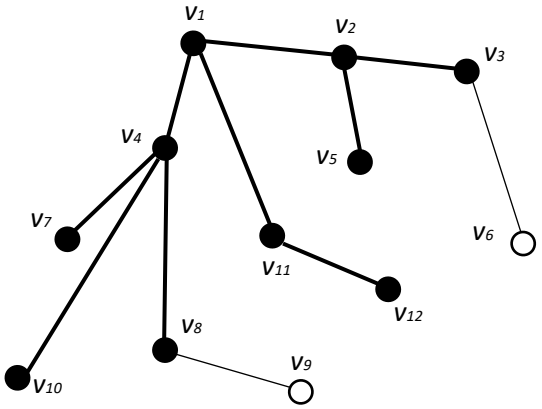
Dati graf se sastoji od 12 vrhova v_i pri čemu je $i = 1, 2, \ldots, 12$. Za početni vrh u postupku izvršavanja BFS algoritma uzet će se vrh v_1. Prvi korak podrazumijeva posjećivanje susjeda vrha v_1, a to su vrhovi v_2, v_{11} i v_4 (slika br. 2)



Slika br. 2 Prvi korak BFS pretrage

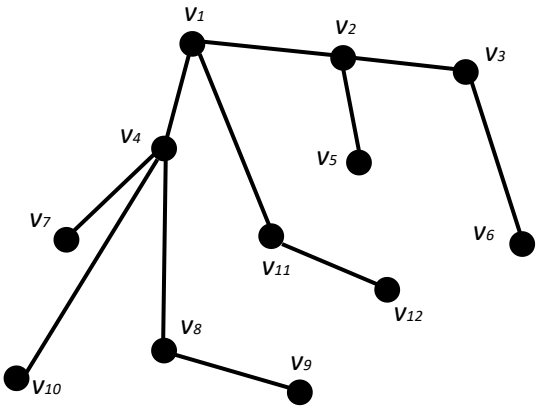
Nakon prvog koraka, potrebno je posjetiti sve susjede vrhova v_4, v_{11} i v_2. Prvo se mogu posjetiti susjedi vrha v_4, a to su vrhovi v_7, v_{10} i v_8. Zatim slijedi posjećivanje susjeda vrha v_{11},

a to je vrh v_{12}, te ostaje posjetiti susjede vrha v_2, a to su vrhovi v_5 i v_3 (slika br. 3).



Slika br. 3 Drugi korak BFS pretrage

Naredni vrhovi koje treba razmotriti su vrhovi v_7, v_{10}, v_8, v_{12}, v_5 i v_3. Na slici br. 3 se jasno vidi da samo vrhovi v_8 i v_3 imaju svog susjeda, a to su vrhovi v_9 i v_6, respektivno, te se posjećuju ti vrhovi.



Slika br. 4 Treći (finalni) korak BFS pretrage

Na ovaj način su posjećeni svi vrhovi datog grafa, jer nije ostao niti jedan vrh koji ima novog susjeda. Ekvivalentnim postupkom bi se vršilo pretraživanje grafa sa označavanjem vrhova kao koraka, odnosno iteracije u kojoj se određeni vrh posjetio. Ukoliko se vrh v_1 označi kao 0 (početni vrh), ekvivalentne oznake za ostale vrhove bi bile sljedeće:

$v_4 \longrightarrow 1$, $v_{11} \longrightarrow 1$, $v_2 \longrightarrow 1$

$v_7 \longrightarrow 2$, $v_{10} \longrightarrow 2$, $v_8 \longrightarrow 2$, $v_{12} \longrightarrow 2$, $v_5 \longrightarrow 2$, $v_3 \longrightarrow 2$

$v_9 \longrightarrow 3$, $v_6 \longrightarrow 3$

BFS algoritam - pseudokod

U prvom koraku algoritam prima kao ulaz graf *G* i početni vrh *v*. Zatim se definiše red *Q* i u red se dodaje početni vrh *v*. Početni vrh *v* se označava kao posjećen. Sve dok red *Q* nije prazan algoritam izvršava sljedeće korake: vrh *u* se uzima sa vrha reda *Q* i za svakog susjeda *w* vrha *u* u grafu *G* se provjerava da li je posjećen. Ukoliko vrh *w* nije posjećen, dodaje se u red *Q* i označava kao posjećen. Ovaj algoritam će se nastaviti sa ovim koracima sve dok red *Q* ne bude prazan. U krajnjem rezultatu, algoritam će posjetiti sve vrhove koji se nalaze u grafu *G* i oni će biti označeni kao posjećeni. [7]

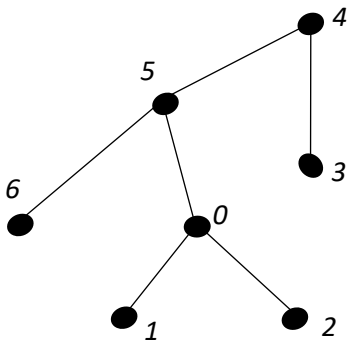
```
1   BFS (G, v)
2   Q je red
3   Q.dodaj(v)
4   označi v kao posjećen
5   sve dok red Q nije prazan
6           u = Q.izbaci( )
7           za sve susjede w od u u grafu G
8                   ako w nije posjećen
9                           Q.dodaj(w)
10                          označi w kao posjećen
```

Slika br. 5 Pseudokod za BFS algoritam

BFS u Pythonu

Na početku su definisane varijable *posjecen* i *red – posjecen* je lista za spremanje posjećenih vrhova, dok je *red* lista za vrhove koji se trebaju posjetiti. Metoda *BFS* prima 3 argumenta: *posjecen, graf* i *vrh*. Početni vrh se dodaje u liste *posjecen* i *red*, a zatim se kreće izvršavanje petlje sve dok lista red nije prazna. Kroz petlju se uzima vrh reda (trenutni vrh) i ispisuje, a zatim se provjeravaju njegovi susjedi. Ukoliko susjed nije posjećen, dodaje se u listu na čekanje. Na kraju se poziva funkcija *BFS* sa definisanim grafom i početnim vrhom 4. U nastavku je dat jedan od mogućih grafičkih prikaza ovog grafa (isti kao iz prethodnog primjera) i rezultat izvršavanja ovog koda. [8]

```
Redoslijed posjecivanja vrhova je sljedeci:
4 3 5 6 0 1 2
```

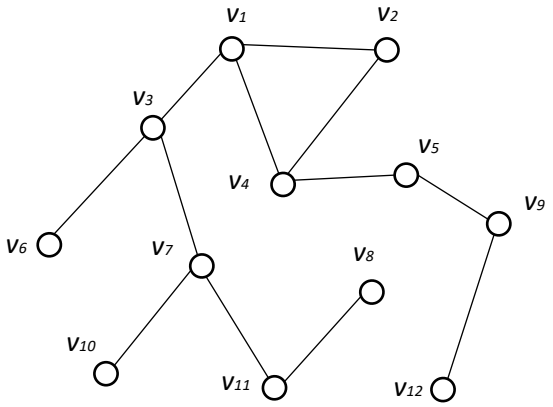


Slika br. 6 Rezultat pretrage BFS algoritma u Pythonu i grafički prikaz graf

Algoritam pretrage u dubinu – DFS

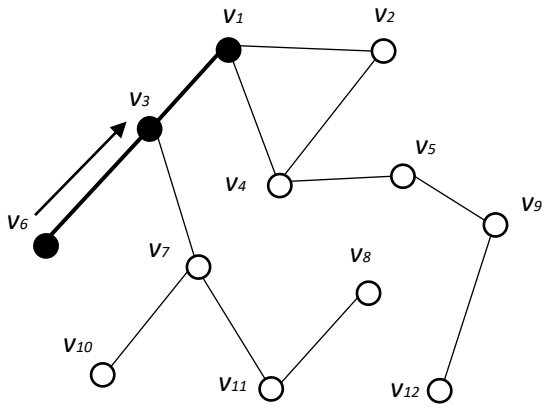
Algoritam pretrage u dubinu, poznatiji pod nazivom DFS (Depth-First Search), kreirao je francuski matematičar Charles Pierre Treamux u 19. vijeku kao strategiju za rješavanje popularne igre labirint. Njegov algoritam je zahtijevao crtanje linija kako bi se označila staza sve dok se ne dođe u slijepu ulicu ili dok se ne pronađe izlaz. [6] Pretraživanje grafa korištenjem DFS algoritma započinje na isti način kao i BFS algoritam, tako što se od svih vrhova grafa izabere jedan vrh kao početni. U narednom koraku se kreće dalje kroz graf i označava se jedan susjed početnog vrha kao posjećen. Nakon toga se vrši pretraživanje na način da se ponovno posjećuje susjed prethodno posjećenog vrha (susjeda početnog vrha) i tako redom dok se ne dođe u situaciju da dati vrh nema niti jednog susjeda. U toj situaciji se pretraživanje vraća u suprotan smjer do prvog vrha koji ima neposjećenog susjeda. Postupak se ponavlja dok se ne završi svaka mogućnost kretanja kroz graf. [1] Kao i kod BFS pretrage, u ovom slučaju je također moguće označavati posjećene vrhove oznakom ekvivalentne iteracije, tj. koraka posjećivanja, s tim da u ovom slučaju to neće predstavljati najkraću udaljenost datog vrha od početnog.

U nastavku je dat graf na kojem će biti demonstriran algoritam pretraživanja u dubinu.



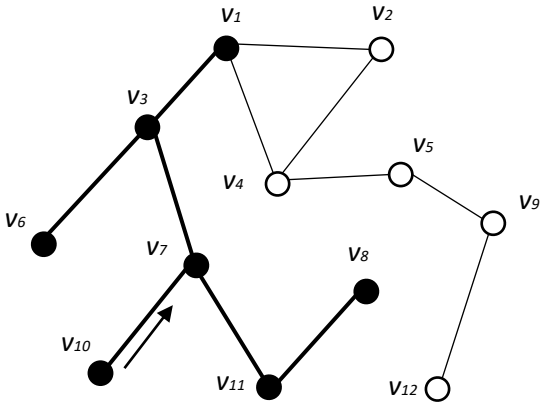
Slika br. 7 DFS pretraga na grafu

Kao početni vrh pretrage odabran je vrh v_1. Uočava se da taj vrh ima tri susjeda, i to: v_3, v_4 i v_2. Kako je uobičajeno da se prvo biraju lijeve ivice, a zatim desne, tako se u ovom slučaju prvo može posjetiti vrh v_3. Vrh v_3 ima dva susjeda: v_6 i v_7, te se prvo se posjećuje vrh v_6. Budući da vrh v_6 nema svojih susjeda, potrebno je vratiti se unazad do prvog vrha koji ima neposjećenog susjeda, a to je vrh v_3 (slika br. 8).



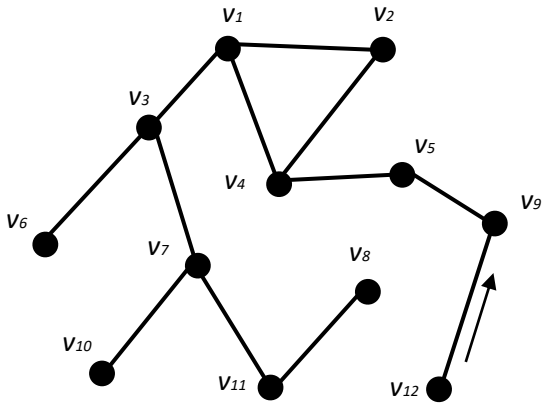
Slika br. 8 Prvi korak DFS pretrage

Nastavlja se posjećivanje susjeda vrha v_7. Budući da se ivica v_7v_{10} nalazi lijevo u odnosu na ivicu v_7v_{11}, prvo se posjećuje vrh v_{10}. Kako taj vrh nema svog susjeda, pretraživanja se vrši unazad, i to do vrha v_7 koji ima neposjećnog susjeda v_{11}. Vrh v_{11} ima jednog susjeda, vrh v_8, te vrši se njegovo posjećivanje (slika br. 9).



Slika br. 9 Drugi korak DFS pretrage

Vrh v_8 nema svojih susjeda, te se pretraga vrši unazad sve do vrha v_1 budući da je to prvi vrh na koji se nailazi na ovom putu, a koji ima neposjećene susjede. Posjećuje se susjed vrha v_1, vrh v_4 (nalazi se lijevo u odnosu na vrh v_2). Nakon toga se nastavlja posjećivanje njegovog susjeda v_5, zatim susjeda vrha v_5 tj. vrha v_9 i posjećivanje susjeda vrha v_9, vrha v_{12}. Kako vrh v_{12} nema susjeda, pretraživanje se odvija u suprotnom smjeru, sve do vrha v_4 koji ima neposjećnog susjeda. Posjećivanjem susjeda vrha v_4, vrha v_2, završava se pretraga grafa algoritmom DFS (slika br. 10).



Slika br. 10 Treći korak DFS pretrage

Ekvivalentni brojčani zapisi vrhova na grafu (oznake iteracije/koraka) bi bili sljedeći:

$v_1 \longrightarrow 0$, $v_3 \longrightarrow 1$, $v_6 \longrightarrow 2$, $v_7 \longrightarrow 3$, $v_{10} \longrightarrow 4$, $v_{11} \longrightarrow 5$, $v_8 \longrightarrow 6$, $v_4 \longrightarrow 7$, $v_5 \longrightarrow 8$, $v_9 \longrightarrow 9$, $v_{12} \longrightarrow 10$, $v_2 \longrightarrow 11$,

DFS algoritam – pseudokod

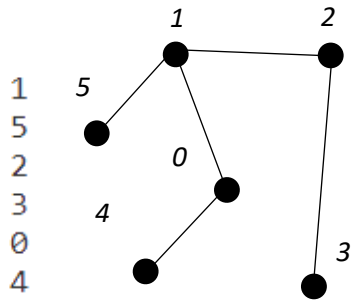
U prvom koraku se inicijalizira stek *S*. *S.ubaci()* služi za dodavanje vrha *v* na stek, nakon čega se on označava kao posjećen (početni vrh). Petlja će se izvršavati sve dok stek ne bude prazan – vrh *u* se označava kao onaj koji će biti uklonjen u narednoj iteraciji, budući da je zadnji ubačen u stek prvi će biti izbačen iz njega. Nakon toga se izvršava petlja za sve susjede vrha *u*, te ako njegov susjed, vrh *w*, nije posjećen, dodaje se na stek i označava kao posjećen. Vrhovi grafa će biti posjećeni prema pristupu pretraživanja u dubinu, sve dok stek ne bude prazan. [12]

```
1   DFS (G, v)
2   S je stek
3   S.ubaci(v)
4   označi v kao posjećen
5   sve dok S nije prazan
6           u = element sa vrha steka S
7           u = S.izbaci( )
8           za sve susjede w od u  grafa  G
9                   ako w nije posjećen
10                          stavi vrh w na vrh steka S
11                          označi w kao posjećen
```

Slika br. 11 Pseudokod za DFS algoritam

DFS algoritam u Pythonu

Funkcija *dfs* prima argumente: *graf* (predstavlja inicijalizaciju grafa, tj. povezanost vrhova i kreiranje ivica), *pocetak* (početni vrh pretrage grafa) i *posjecen* koji predstavlja skup vrhova koji su već posjećeni. Ukoliko pri prvom pozivu metode, *posjecen* nije definisan, njegova vrijednost se postavlja na prazan skup, a zatim se dodaje početni vrh u taj skup kako bi se označio kao posjećen. U *for* petlji se prolazi kroz susjede vrha *pocetak* koji nisu posjećeni, a koji su određeni kao razlika skupa susjednih vrhova (*graf[pocetak]*) i skupa posjećenih vrhova (*posjecen*). Za svaki neposjećen vrh se rekurzivno poziva funkcija *dfs* sa novim početnim vrhom i skupom *posjecen*, koji je ažuriran.
Metoda vraća skup posjećenih vrhova. Nakon kreirane metode, inicijalizira se graf i vrši se pretraživanje DFS algoritmom iz početnog čvora 1. U prilogu je dat jedan od mogućih grafičkih prikaza grafa i rezultat pretrage

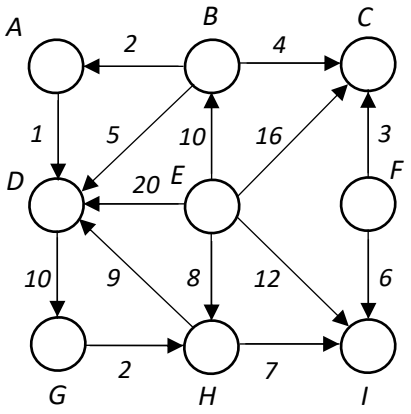


Slika br. 12 Rezultat DFS pretrage i grafički prikaz grafa

Dijkstrin algoritam

Dijkstrin algoritam, kako i samo ime upućuje, kreiran je 1959. godine od strane holandskog fizičara Edsgera W. Dijkstre, u momentima kada je razmišljao o tome kako pronaći najkraći put od Roterdama do Groningena. [5] Za razliku od BFS i DFS algoritma koji služe za oblizaka vrhova grafa, Dijkstrin algoritam se koristi za pronalaženje najkraćeg puta između dva vrha u težinskom grafu. Ovaj algoritam započinje dodjeljivanjem početnih vrijednosti za udaljenosti od početnog vrha do svakog vrha u datom grafu, da bi se u svakom narednom koraku pretpostavljena najkraća vrijednost ažurirala ukoliko se dođe da zaključka da postoji manje rastojanje između dva vrha.

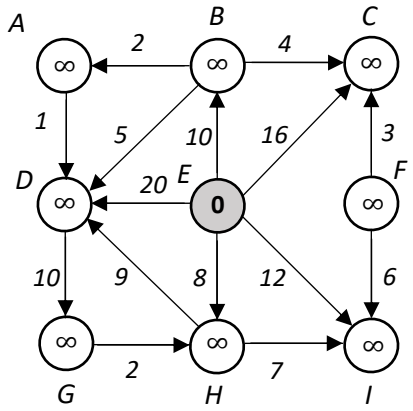
Na slici je dat primjer orijentisanog, težinskog grafa na kojem će biti objašnjen način funkcionisanja Dijkstrinog algoritma



Slika br. 13 Izvršenje Dijkstrinog algoritam na grafu

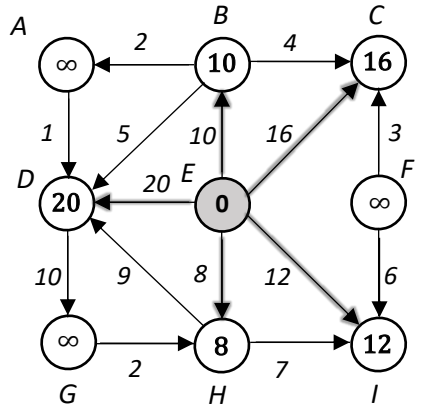
Za početni vrh je uzet vrh *E*. Potrebno je naći najkraći put do svakog drugog vrha u datom grafu. Kako je graf orijentisan, treba imati na umu da se ne može kretati svim ivicama grafa, već da je tačno određeno koji vrh je početni, a koji završni za svaku ivicu.

U svaki vrh se može upisati broj koji odgovara najkraćoj udaljenosti od vrha *E*, a za početak će to biti vrijednost beskonačnosti. Ta vrijednost će se mijenjati u zavisnosti od koraka pretrage u kojima će se definisati udaljenost od vrha *E* koja može da varira i prima manju vrijednost kako bi predstavljala najkraće rastojanje ili da bude ona koja je zapisana prema prvoj pretpostavci za svaki vrh. U vrh *E* se upisuje vrijednost *0*, kako je to početni vrh od kojeg kreće računanje rastojanja.



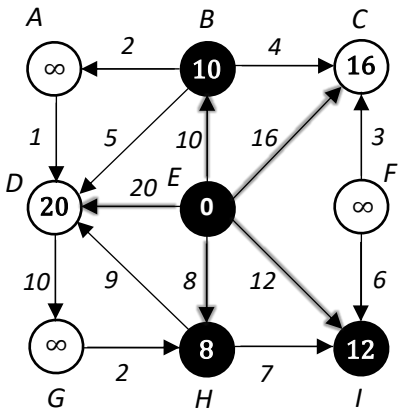
Slika br. 14 Obilježavanje početnog vrha pretrage

Iz vrha *E* se vrši kretanje prema ostalim vrhovima do kojih se može doći, a u prvom koraku to su vrhovi *B*, *C*, *D*, *H* i *I*. U te vrhove se upisuje prva pretpostavka najkraće udaljenosti od vrha *E* do svakog od tih vrhova.



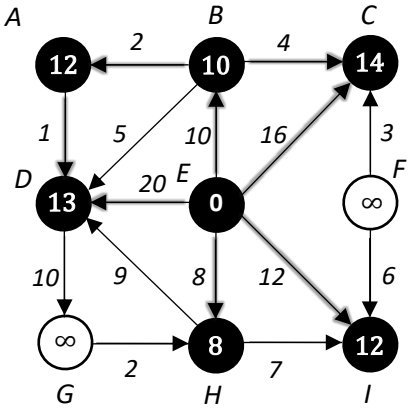
Slika br. 15 Pretraga susjeda početnog vrha

Ukoliko se razmotre ivice koje kao svoj početak imaju vrh *E*, može se zaključiti da je udaljenost vrha *E* od vrhova *B*, *H* i *I* ujedno i najkraća udaljenost od vrha *E* do tih vrhova, respektivno.



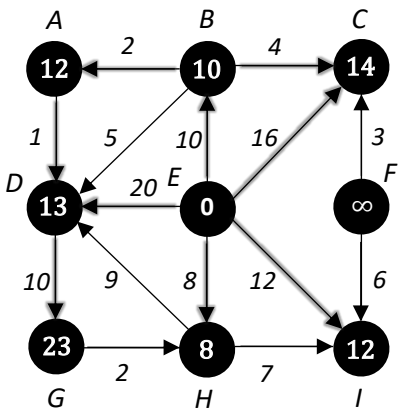
Slika br. 16 Obilježavanje najkraće udaljenosti za vrhove B, H i I

Kako je jedini put, samim tim i najkraći, od vrha *E* do vrha *A* jednak *E-B-A*, tako se zaključuje da je to najkraća udaljenost od vrha *E* do vrha *A*. Do vrha *C* se može doći kraćim putem, preko vrha *B*, slijedeći put *E-B-C*, pa se zbog toga umjesto prvobitne pretpostavke vrijednosti *16*, za vrh *C* uzima vrijednost *14* kao vrijednost najkraće udaljenosti od vrha *E* do tog vrha. Put od vrha *E* preko vrhova *A* i *B*, pa sve do vrha *D* je najkraći put od vrha *E* do tog vrha, te prvobitno usvojena vrijednost najkraće udaljenosti *20* se mijenja sa vrijednošću *13*.



Slika br. 17 Obilježavanje najkraće udaljenosti za vrhove A i D

Do vrha *G* se može doći prateći put *E-B-A-D-G* ili *E-D-G*. Kako je udaljenost prvog puta kraća od udaljenosti drugog, tako se za vrijednost udaljenosti od vrha *E* do vrha *G* uzima *23*. Vrh *F* je jedini vrh do kojeg se niti jednim putem ne može stići iz vrha *E*, budući da je on samo početak za ivice, odnosno ne postoji niti jedna ivica u datom grafu koja završava u vrhu *F*. Najkraća vrijednost ostaje jednaka prvobitnoj, tj. vrijednosti beskonačno (slika br. 18).



Slika br. 18 Posljednji korak pretrage i određivanja najkraće udaljenosti

Dijkstrin algoritam – pseudokod

Algoritam koristi nizove *dist* i *prev* koji se koriste za pohranjivanje najkraćih udaljenosti i prethodnika za svaki vrh. Na početku se za svaki vrh u grafu postavlja vrijednost BESKONAČNO za *dist* i NEPOZNATO za *prev*, što znači da još uvijek nije pronađen najkraći put. Svaki vrh se dodaje u skup *Q* koji sadrži vrhove koji još nisu posjećeni. Za početni vrh se postavlja udaljenost *0* (linija 5). Algoritam započinje petlju koja se izvodi sve dok skup *Q* nije prazan. U svakoj iteraciji petlje, algoritam pronalazi vrh u *Q* koji ima najmanju vrijednost *dist*. Taj vrh se uklanja iz skupa *Q* i za svakog susjeda tog vrha koji se još nalazi u *Q* (linija 10), provjerava se može li se do njega doći kraćim putem preko trenutnog vrha. Ukoliko je to moguće, ažurira se *dist*[*v*] i *prev*[*v*]. Na kraju petlje vratit će se *dist*[] i *prev*[] koji će sadržavati najkraće udaljenosti i prethodnike za svaki vrh u grafu. [5]

```
1   funkcija Dijkstra(Graf, izvor)
2       za svaki vrh v u Graf.vrhovi
3           dist[v] ← BESKONAČNO
4           prev[v] ← NEPOZNATO
5       dist[izvor] ← 0
6       sve dok lista Q nije prazna:
7           u ← vrh iz Q sa najmanjom udaljenosti dist[u]
8           izbaci u iz Q
9           za svakog susjeda v od u koji je još u Q
10              alt ← dist[u] + Graf.ivice(u, v)
11              if alt < dist[v]
12                  dist[v] ← alt
13                  prev[v] ← u
14  vrati dist[ ], prev[ ]
```

Slika br. 19 Pseudokod za Dijkstrin algoritam

Praktična primjena algoritama iz teorije grafova

Iako teorija grafova svoje karakteristike, pojmove i definicije pronalazi u matematici, ovo područje je široko rasprostranjeno u praksi. Pogrešno je mišljenje da algoritmi iz teorije grafove ne izlaze iz okvira matematike, naprotiv, svoju primjenu imaju u mnogobrojnim sferama djelovanja, te kao važan faktor utiču na rješenja velikog broja problema i pomažu pri izvedbama mnogih rješenja.

U nastavku rada će biti navedeni praktični primjeri upotrebe tri algoritma koja su opisana ovim radom, a to su: algoritam pretrage u širinu, algoritam pretrage u dubinu i Dijkstrin algoritam.

Upotreba BFS algoritma

Ovo su neke od praktičnih primjena BFS algoritma:

- **Peer to Peer mreža –** kada čvor u P2P mreži treba da komunicira sa drugim, potrebno je da pronađe stazu ili rutu do odredišnog čvora. BFS algoritam se može korsititi za pronalaženje najkraćeg puta između dva čvora. Svaki čvor održava listu susjednih čvorova u mreži, a kada čvor želi pronaći rutu do drugog čvora on šalje poruku svojim susjednim čvorovima tražeći od njih da proslijede poruku svojim susjedima. Svaki čvor koji primi poruku provjerava da li je to odredišni čvor, a ako nije, prosljeđuje poruku svojim susjednim čvorovima koji još nisu primili poruku. Svaki čvor prati čvorove koje je već posjetio kako bi se izbjegle petlje i spriječile suvišne poruke.
- **GPS navigacijski sistem –** u kontekstu GPS navigacije, karta je predstavljena kao graf gdje vrhovi predstavljaju raskrsnice ili čvorove, a ivice predstavljaju puteve ili veze koje ih povezuju. Korištenjem BFS algoritma vrši se pretraga mape ili pronalazak najkraćeg puta između 2 destinacije, odnosno početne tačke (početna pozicija kretanja) i krajnje tačke (odredišta).
- **Društvene mreže –** BFS je koristan alat za web stranice društvenih mreža za obavljanje različitih zadataka kao što su pronalazak prijatelja, preporuka novih konekcija i identifikacija zajednica. Web lokacije društvenih mreža često koriste BFS algoritam za pronalazak prijatelja već postojećih prijatelja. Počevši od neposrednih konekcija korisnika, BFS algoritam traži druge korisnike koji su direktno povezani na konekcije korisnika, itd. BFS se također može koristiti za preporuku novih veza korisnicima na osnovu zajedničkih interesa, hobija ili drugih faktora, za otkrivanje neželjene pošte ili lažnih profila na web stranicama društvenih mreža. Pretragom korisnika koji su povezani sa velikim brojem drugih korisnika, ali imaju malu ili nikakvu interakciju sa

njima, BFS može identificirati profile koji mogu biti lažni ili generisani od strane malicioznih korisnika.

- **Umjetna inteligencija –** jedan od osnovnih alata koji se koristi u AI-u za različite zadatke kao što su pretraživanje, pronalaženje puta, mašinsko učenje, itd. [2]

Upotreba DFS algoritma

Ovo su neke od praktičnih primjena DFS algoritma:

- **Rješavanje labirinta –** DFS je uobičajen način na koji se pristupa rješavanju problema poput labirinta. Prvo se izabere jedna staza koja se prati sve dok se ne dođe u slijepu ulicu ili do kraja labirinta. Ukoliko data staza ne funkcioniše (rezultat pretrage nije uspješan), potrebno je vratiti se unazad i ići alternativnim putem od prethodnog raskršća i ponoviti proces pretrage.
- **Sudoku –** jedan od načina na koje se može riješiti igra Sudoku jeste koristeći DFS algoritam. Potrebno je početi sa praznom mrežom polja i odabrati praznu ćeliju koja se želi popuniti. Nakon toga, potrebno je pokušati ubaciti sve brojeve iz skupa 1-9 i pronaći narednu praznu ćeliju. Ukoliko se dođe do tačke u kojoj se nijedna cifra ne može staviti u ćeliju bez kršenja pravila Sudoku igre, potrebno je vratiti se na prethodnu ćeliju i pokušati ubaciti drugu cifru. Ovaj proces se nastavlja dok se cijela mreža ne popuni.
- **Web pretraživači –** DFS je uobičajen algoritam koji se koristi u web pretraživačima za istraživanje i indeksiranje web stranica. Kada web pretraživač započne svoje indeksiranje, on obično počinje od datog početnog URL-a i uzima sadržaj web stranice. Zatim se izdvajaju sve veze na stranici i dodaju se na listu URL-ova koje treba posjetiti. DFS algoritam se zatim primjenjuje za posjetu URL-ova na listi, jednog po jednog. Ovaj proces se ponavlja sve dok se ne posjete sve web stranice u domeni. DFS se koristi u web pretraživačima jer omogućava indeksiranju da posjeti što je više moguće veza u datom domenu prije nego što pređe na sljedeću domenu. Ovo osigurava da pretraživač ne propusti niti jedan važan sadržaj i da može indeksirati cijelu domenu. [3]

Upotreba Dijkstrinog algoritma

Ovo su neke od praktičnih primjena Dijkstrinog algoritma:

- **Udaljenost između lokacija** – Dijkstrin graf je primjenjiv u situacijama poput pronalska udaljenosti ili puta od jedne tačke/lokacije do druge tačke/odredišta, npr. od jednog grada do drugog.
- **Društvene mreže** – kao i BFS algoritam, Dijkstrin algoritam također nalazi svoju primjenu u polju društvenih mreža za međusobno povezivanje korisnika, listu prijatelja koji su ponuđeni u sklopu dijela 'osobe koje možda poznajete', i slično.
- **Raspored avionskih linija i letova** – u ovom slučaju, može se pretpostaviti da je graf usmjeren sa određenim smjerovima – vrhovi predstavljaju aerodrome, a usmjerene ivice predstavljaju ivice sa 2 težine: vrijeme polaska i vrijeme dolaska. Uz pomoć Dijkstrinog algoritma moguće je pronaći najkraći put između 2 destinacije, odnosno najoptimalniju rutu za najkraći put i najefikasniji let.
- **Telefonska mreža** – u telefonskoj mreži svaka linija ima propusni opseg. Širina pojasne linije je najveća frekvencija koju linija može podržati. Generalno, ako je frekvencija signala viša u određenoj liniji, signal se smanjuje za tu liniju. Širina pojasa predstavlja količinu informacija koja se može prenijeti linijom, Ukoliko se grad zamisli kao graf, vrhovi predstavljaju komutacione stanice, ivice dalekovode, a težina ivica propusni opseg. Može se zaključiti da i ovo spada u problem najkraće udaljenosti, za koji se može koristiti Dijkstra algoritam.
- **Roboti i dronovi** – bespilotne letjelice/roboti koji su automatizirani i koji se koriste za dostavu paketa na određenu lokaciju ili za izvršavanje određenog zadatka, vode se ovim algoritamskim modulom tako da, kada su izvor i odredište poznati, robot/dron se kreće u naređenom smjeru slijedeći najkraći put da nastavi isporuku paketa ili obavi traženi zadatak u minimalnom vremenu. [4]

ZAKLJUČAK

Ovaj rad je prikazao algoritme u teoriji grafova, sa posebnim osvrtom na tri algoritma. Kroz rad su implementirani načini na koji svaki od tri opisana algoritma (pretraga u širinu, pretraga u dubinu, Dijkstra algoritam) funkcioniše, njihove osobenosti, te je predstavljen jedan mogući način izvršavanja datih algoritama u programskom jeziku (Python). Implementacija algoritama je pokazala koliko su oni značajni u okvirima izvan domena matematike, te koliki udio i kakav faktor predstavljaju u funkciji brojnih parametara svakodnevice. Kako se tehnologija razvija iz dana u dan, algoritmi teorije grafova imaju značajan udio u svemu novom što je trenutno tu među nama, ali i u onom što tek dolazi i sa čime se treba upoznati.

Ovaj rad je za rezultat ponudio osvrt na teoriju grafova i neke od njenih algoritama, te pokazao kako je matematika u jakoj sprezi sa drugim oblastima, a u današnjem naprednom tehnološkom dobu, ponajviše sa informacionim tehnologijama.

Literatura i izvori

1. A. Huskanović, H. Alajbegović, Teorija grafova, Diskretna matematika, Zenica, 2020. godine[1]

2. Applications Of Breadth First Search Traversal, GeeksforGeeks, Dostupno: https://www.geeksforgeeks.org/applications-of-breadth-first-traversal/, Januar, 2023[2]

3. Applications of Depth First Search, FACEPrep, Dostupno: https://www.geeksforgeeks.org/applications-of-breadth-first-traversal/, Mart, 2020[3]

4. Applications of Dijkstra's shortes path algorithm, GeeksforGeeks, Dostupno: https://www.geeksforgeeks.org/applications-of-dijkstras-shortest-path-algorithm/, Septembar[4]

5. F. Habibullaeav, Practical Dijkstra's Algorithm, Medium, Dostupno: https://farruh.io/practical-dijkstras-algorithm-b329ade79a1e, Novembar, 2019[5]

6. Isaac Computer Science, Dijkstra's Algorithm, Dostupno: https://isaaccomputerscience.org/concepts/dsa_search_dijkstra?examBoard=all&stage=all[6]

7. L. Ringer, R. Turner, G. Carroll, The Explanation & Comparison of Graph Searches, Dostupno: https://cse442-17f.github.io/A-Star-Search/[7]

8. P. Garg, Hackerearth, Algorithms: Depth First Search, Dostupno: https://www.hackerearth.com/practice/algorithms/graphs/depth-first-search/tutorial/[8]

9. Programiz, Breadth First Search, Dostupno: https://www.programiz.com/dsa/graph-bfs[9]

10. S. Kappagantula, All You Need To Know About Breadth First Search Algorithm, Medium, Dostupno: https://medium.com/edureka/breadth-first-search-algorithm-17d2c72f0eaa, September, 2019[10]

11. Z. Lateef, All You Need To Know About Breadth First Search Algorithm, Edureka, Dostupno: https://www.edureka.co/blog/breadth-first-search-algorithm/[11]

12. Ž. Jurić, Elementi teorije grafova, Radna skripta za kurs "Diskretna matematika" na Elektrotehničkom fakultetu u Sarajevu, Sarajevo, 2012/13. Godina[12]

ALGORITHMS IN GRAPH THEORY (BFS, DFS AND DIJKSTR'S ALGORITHM)

Abstract

In this paper, algorithms in the graph theory are presented, namely: depth-first search algorithm, breadth-first search algorithm and Dijkstra's algorithm. The mentioned algorithms are demonstrated in the form of written code through the programming languages, the execution method of each algorithm is described, their importance is highlighted, as well as the practical application which shows the importance and wide use of the mentioned algorithms.

Keywords: *graph, node, edge, search algorithms*