UNIVERZITET U BEOGRADU
MATEMATIČKI FAKULTET

Nemanja D. Jelić

ALGORITMI ZA REŠAVANJE GRAFOVSKOG PROBLEMA NAJŠIREG PUTA

master rad

Beograd, 2022.

Mentor:

dr Vesna Marinković, docent
Univerzitet u Beogradu, Matematički fakultet

Članovi komisije:

dr Filip Marić, vanredni profesor
Univerzitet u Beogradu, Matematički fakultet

dr Mirko Spasić, docent
Univerzitet u Beogradu, Matematički fakultet

Datum odbrane: 26.09.2022.

Zahvaljujem se mentoru doc. dr Vesni Marinković na svim savetima i smernicama tokom izrade master rada.

Naslov master rada: Algoritmi za rešavanje grafovskog problema najšireg puta

Rezime: Grafovski algoritmi predstavljaju podoblast teorije algoritama koja se bavi konstrukcijom i analizom algoritama za rešavanje grafovskih problema. Jedan od problema koji se izučava u okviru oblasti grafovskih algoritama je i problem najšireg puta kod koga je zadatak da se u datom težinskom grafu pronađe put između dva data čvora čija je grana minimalne težine maksimalna moguća. Ovaj problem poznat je i pod nazivom problem maksimizacije kapaciteta puta. Cilj ovog rada je prikazati i programski realizovati nekoliko različitih algoritama za rešavanje problema najšireg puta.

Ključne reči: graf, najširi put, maksimalni kapacitet puta

Sadržaj

Glava 1

Uvod

Grafovski algoritmi predstavljaju podoblast teorije algoritama koja se bavi konstrukcijom i analizom algoritama za rešavanje grafovskih problema. Mnogi problemi iz realnog života mogu se modelovati u terminima grafovskih problema, pa su zato grafovi kao struktura podataka jako korisni i važni. Grafovi se najčešće koriste da predstave mreže, kao što su, na primer, putevi između gradova, telefonska mreža ili strujno kolo. Grafovi se, takođe, koriste za modelovanje društvenih mreža i odnosa koji u njima postoje, poput društvene mreže Facebook i profesionalne poslovne mreže LinkedIn.

Jedan od problema koji se izučava u okviru oblasti grafovskih algoritama je i *problem najšireg puta* (eng. *widest path problem*) kod koga je zadatak da se u datom težinskom grafu pronađe put između dva data čvora čija je grana minimalne težine maksimalna moguća. Ovaj problem poznat je i pod nazivom *problem maksimizacije kapaciteta puta* (eng. *maximum capacity path problem*). Problem najšireg puta pronalazi primenu u raznim oblastima. Neke od mogućih primena su za rešavanje problema rutiranja internet mreže, Šulcov metod (eng. *Schulze method*) koji odlučuje pobednika na izborima, proces montaže većeg broja slika u jednu (eng. *digital compositing*), modelovanje metaboličkih mreža, kao i problem maksimizacije toka.

Cilj ovog rada je prikazati i programski realizovati nekoliko različitih algoritama za rešavanje problema najšireg puta. Predstavljeni algoritmi, od efikasnijih do manje efikasnih, se upoređuju na testnom uzorku odabranih grafova.

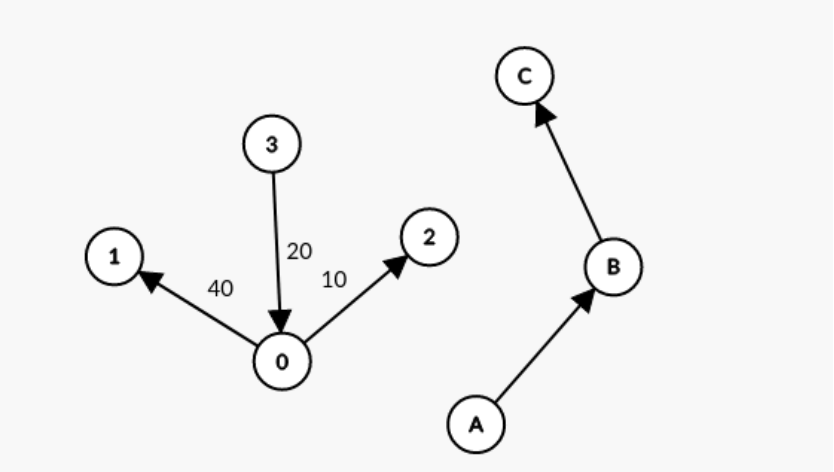
1.1 Osnovni pojmovi

Strukture podataka predstavljaju osnovnu gradivnu jedinicu softverskog inženjerstva. One definišu način na koji su podaci organizovani u memoriji. Strukture podataka se mogu podeliti na primitivne, osnovne i složene. Primitivne strukture podataka predstavljaju gradivne blokove za osnovne i složene strukture podataka i u njih spadaju celi brojevi, realni brojevi, karakteri itd. Osnovne strukture podataka su izgrađene od jedne ili više primitvnih struktura podataka. Primer osnovnih struktura podataka su nizovi i liste. Složene strukture podataka mogu sadržati u sebi i primitivne i osnovne strukture podataka. Mogu biti linearne i nelinearne. Linearne strukture podataka formiraju linearnu sekvencu sa jasno definisanim prethodnikom i sledbenikom za svaki element. Primeri linearnih struktura podataka su stek i red. Nelinearne strukture podataka ne formiraju linearnu sekvencu podataka i uređene su hijerarhijski. Drvo i graf su primeri nelinearnih složenih struktura podataka.

Graf je struktura podataka koja služi za opisivanje odnosa između objekata. Graf $G = (V, E)$ se sastoji od skupa čvorova, koji često označavamo sa V, i skupa grana, koji označavamo sa E. Grafovi se dele na usmerene i neusmerene. Grane usmerenog grafa su uređeni parovi čvorova i redosled dva čvora koje povezuje grana je bitan. Grane neusmerenog grafa su neuređeni parovi čvorova. Graf $G' = (V', E')$ je *podgraf* grafa $G = (V, E)$ ako i samo ako važi $V' \subseteq V$ i $E' \subseteq E$.

Težinski graf je graf čijim je granama pridružena težina koja može predstavljati cenu, rastojanje ili neku drugu jedinicu mere. Prednost težinskog grafa u odnosu na netežinski je ta što težinski graf može da predstavi složenije relacije među objektima, dok je očigledna mana što težinski graf zauzima više memorije od odgovarajućeg netežinskog grafa. Na slici 1.1 dat je primer jednog težinskog usmerenog grafa $G_1 = (\{0, 1, 2, 3\}, \{(0, 1), (0, 2), (3, 0)\})$, kao i jednog netežinskog usmerenog grafa $G_2 = (\{A, B, C\}, \{(A, B), (B, C)\})$.

Za težinski graf $G = (V, E)$ uvodimo funkciju $c : E \to \mathbb{Z}$, tako da $c(e)$, što ćemo često označavati sa c_e, predstavlja težinu grane $e \in E$. Kao kodomen ovog preslikavanja koristimo skup $\mathbb{Z}$ zbog jednostavnosti. Naime, većina algoritama o kojima će biti reči vraća tačan rezultat i kada su težine grana realni brojevi. Dakle, težinski graf se može predstaviti i kao uređena trojka $G = (V, E, c)$. U nastavku teksta za težinu grane e grafa koristićemo nekoliko odrednica kao što su c_e, cena, težina, dužina grane i sl. Ako je grana predstavljena kao uređeni par čvorova npr. (u, v), tada za težinu grane (u, v) koristimo i oznaku $c(u, v)$.

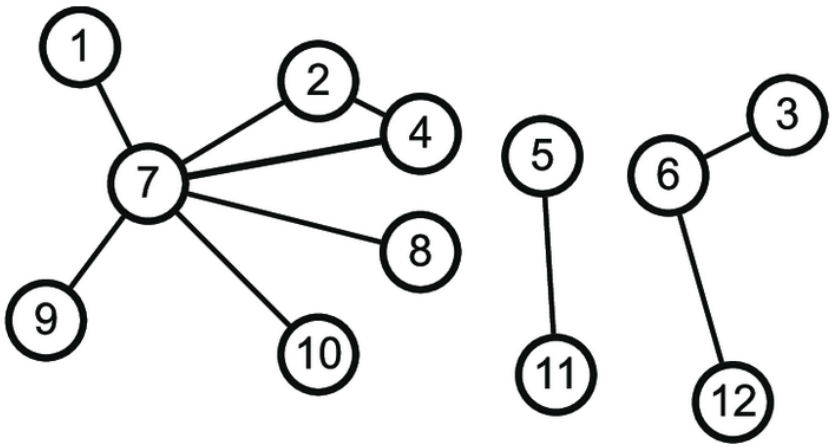


Slika 1.1: Primer jednog težinskog i jednog netežinskog usmerenog grafa

Stepen $d(v)$ čvora v je broj grana susednih čvoru v. U neusmerenom grafu to je broj grana koje povezuju čvor v sa nekim drugim čvorom. U usmerenom grafu razlikujemo ulazni stepen i izlazni stepen čvora: ako grana ulazi u dati čvor, ona se računa u ulazni stepen tog čvora, a ako grana polazi iz datog čvora, ona se onda računa u njegov izlazni stepen. Ulazni stepen čvora 0 težinskog grafa sa slike 1.1 jednak je 1, dok je izlazni stepen tog čvora jednak 2.

Put od čvora v_1 do čvora v_k je niz čvorova $v_1, v_2, \ldots, v_k$ povezanih granama $(v_1, v_2), (v_2, v_3), \ldots, (v_{k-1}, v_k)$. Na primer, put od čvora 3 do čvora 1 na slici 1.1 je $3, 0, 1$. Put je *prost*, ako se svaki čvor pojavljuje u njemu samo jednom. U ovom radu razmatraju se samo prosti putevi. Kada opisujemo neki algoritam, ako u tekstu stoji put, misli se na prost put. Čvor w je *dostižan* iz čvora v ako postoji put od v do w. Po definiciji, čvor v je dostižan iz v. *Ciklus* je put čiji se prvi i poslednji čvor poklapaju. Ciklus je *prost*, ako se sem prvog i poslednjeg čvora niti jedan drugi čvor ne pojavljuje u putu dva puta.

U teoriji grafova, *komponenta povezanosti* neusmerenog grafa G je podgraf grafa G u kom su proizvoljna dva čvora tog podgrafa međusobno povezana putem, i kome nijedan čvor nije povezan ni sa jednim dodatnim čvorom iz grafa G. Na primer, graf prikazan na slici 1.2 sadrži tri komponente povezanosti. Kada govorimo o komponentama povezanosti grafa uglavnom razmatramo neusmerene grafove. Moguće je ovaj pojam proširiti i na usmerene grafove, i tada bismo razlikovali slabu ili jaku povezanost grafa, međutim, u ovom radu taj slučaj nećemo obrađivati.



Slika 1.2: Graf koji sadrži tri komponente povezanosti

1.2 Definicija problema

Problemi koji se bave pronalaženjem optimalnog puta u grafu su veoma česti u oblasti informatike. Njihovo proučavanje datira još od nastanka ove nauke i nalaze se u srži implementacije velikog dela današnjih aplikacija. Ipak za većinu ovih problema mi ne znamo da li su pronađeni algoritmi optimalni ili postoji mogućnost za napredak. Na primer, poznat je problem traženja najkraćeg puta između dva čvora u usmerenom grafu, čijim su granama pridružene nenegativne vrednosti za dužinu. Trenutno najefikasniji poznat algoritam koji rešava ovaj problem je Dijkstrin algoritam [1]. Najbolja implementacija Dijkstrinog algoritma, koja koristi Fibonačijev hip, ima složenost $O(|E| + |V| \cdot \log |V|)$. Međutim, ostaje pitanje da li postoji algoritam koji rešava problem najkraćeg puta i čija je složenost linearna u odnosu na veličinu grafa. Zato je jako važno i dalje razmatrati probleme ovakvog tipa, naravno oslanjajući se na već poznate tehnike, sa otvorenošću za nove pokušaje da se poboljša efikasnost.

Problem traženja optimalnog puta u grafu ima tri varijante:

1. *jedan start - jedan cilj* (eng. *single source - single destination*) (koji se još naziva i $s - d$ *problem*):

   Za dati graf i njegova dva čvora s i d, pronaći optimalan put od s do d,

2. *jedan start* (eng. *single source*):

   Za dati graf i njegov čvor s, pronaći optimalne puteve od s do svakog drugog čvora grafa,

3. *svi parovi* (eng. *all pairs*):

   Za dati graf pronaći optimalan put između svaka dva njegova čvora.

Postoje različite mere optimalnosti puta i konkretna mera koja će biti korišćena zavisi od samog problema koji se rešava. U slučaju problema najšireg puta, optimalan put predstavlja put čiji je kapacitet najveći mogući (u smislu Definicije 1.2.1 koja je data u nastavku). Dakle, u ovom radu se bavimo maksimizacijom kapaciteta puta. Fokus će biti na prvoj varijanti problema: jedan start - jedan cilj. Kako će se pri rešavanju ovog problema koristiti Dijkstrin pristup, druga varijanta problema će na neki način biti pokrivena, ali je nećemo eksplicitno navoditi.

Kako je oblast pronalaženja optimalnog puta u grafu jako široka, uvek postoje dodatne varijacije problema. Na primer, moguće je zahtevati da se pronađe jedno rešenje ili sva moguća rešenja problema. Iako se većina algoritama lako može prilagoditi da pronađe sve optimalne puteve, u ovom radu, zbog jednostavnosti, fokus će biti na pronalasku samo jednog proizvoljnog optimalnog puta.

Definicija 1.2.1. Neka je dat težinski (usmeren ili neusmeren) graf $G = (V, E)$, sa celobrojnim težinama grana $c_e \in \mathbb{Z}$ za sve grane $e \in E$. *Kapacitet* b_p puta p je vrednost grane najmanje težine na tom putu tj.

$$b_p := min_{e \in p} c_e.$$

Ako fiksiramo startni čvor $s \in V$, *maxmin* čvora $v \in V$ ili $maxmin_v$ (koristi se i eng. naziv *bottleneck*) definišimo kao maksimalni kapacitet puta od s do v tj.

$$maxmin_v := max_p b_p,$$

posmatrajući sve puteve p koji vode od s do v u grafu G. Grana koja određuje kapacitet puta ili *maxmin* nekog čvora, naziva se *kritična grana* (eng. *critical edge*) puta ili čvora.

Napomena. Naziv termina *maxmin* je formiran od reči *max* i *min* jer predstavlja maksimalnu vrednost minimalne vrednosti težine grane na putu.

Definicija 1.2.2. *Najširi put* (eng. *widest path*) od čvora s do čvora d ($s, d \in V$) je put od s do d čiji je kapacitet najveći mogući.

Definicija 1.2.3. *Grafovski problem najšireg puta* ima zadatak da, za dati težinski graf $G = (V, E)$, sa težinama grana $c_e \in \mathbb{Z}$ ($e \in E$), i za dati startni čvor $s \in V$, kao i ciljni čvor $d \in V$ pronađe najširi put od s do d.

Kada razmatramo neki algoritam za traženje najšireg puta uvek fiksiramo polazni, odnosno startni čvor koji u radu označavamo sa s (eng. *source*). Često je fiksiran i ciljni čvor koji u radu označavamo sa d (eng. *destination*). Kažemo da je kapacitet čvora d zapravo kapacitet najšireg puta od s do d. Ako je $s = d$, tada po definiciji kažemo da prosti put od s ka samom sebi ne postoji. U daljem tekstu, gde god se spominju startni čvor s i ciljni čvor d, možemo pretpostaviti da su oni različiti.

U svakom algoritmu koji ćemo u ovom radu prikazati jedan od ulaznih argumenata je graf G. Poznato je da postoje različite tehnike za predstavljanje strukture podataka graf. Najpoznatije strukture podataka koje se koriste za implementaciju grafa su matrica susedstva i lista susedstva. S obzirom da se većina algoritama koji će biti prikazani u radu zasniva na obilasku grafa u dubinu i iteraciji kroz sve susede nekog čvora, lista susedstva se nameće kao bolja opcija za implementaciju grafa zbog uštede na vremenskoj i memorijskoj složenosti. Poznato je da se za graf $G = (V, E)$ koji je implementiran pomoću liste susedstva koristi $O(|V| + |E|)$ memorijskog prostora, dok se za graf koji je implementiran pomoću matrice susedstva koristi $O(|V|^2)$ memorijskog prostora. Takođe, vremenski je mnogo efikasnije proći kroz susede čvora kada je graf zadat listom povezanosti, nego prolaziti kroz celu vrstu matrice kada je graf zadat matricom susedstva. Prednost implementacije grafa pomoću matrice susedstva se zasniva na kostantnoj vremenskoj složenosti direktnog pristupa određenoj grani grafa. Međutim, implementacija algoritama u ovom radu ne zahteva efikasan direktan pristup grani između dva određena čvora grafa, već je fokus na efikasnom pristupu skupu svih grana susednih određenom čvoru i iteraciji kroz jedan po jedan element tog skupa, što implementacija pomoću liste susedstva i omogućava.

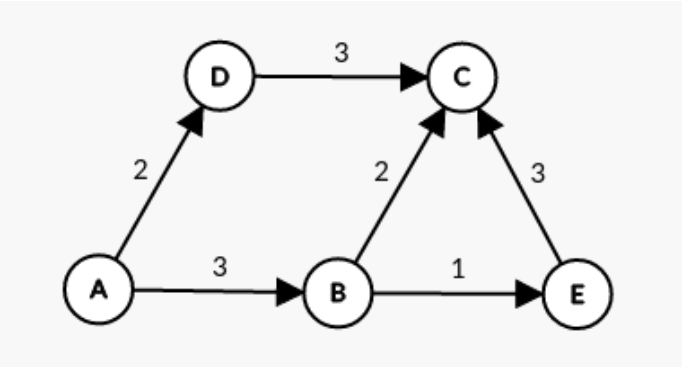
Problem pronalaženja najšireg puta u težinskom grafu je problem nalaženja puta koji ima najveći mogući kapacitet. Dakle, to je problem maksimizovanja kapaciteta puta.

Problem 1.2.1. Neka je dat težinski graf G sa skupom čvorova $V = \{v_1, v_2, \ldots, v_n\}$ i skupom grana $E = \{e_1, e_2, \ldots, e_m\}$. Neka su s i d dva različita čvora u grafu, s, $d \in V$. Pronaći najširi put od čvora s do čvora d u grafu G.

Primer 1.2.1. Posmatrajmo graf na slici 1.3. Najširi put od čvora A do čvora C je A, B, C kao i A, D, C, dok put A, B, E, C nije najširi put jer je njegov kapacitet manji od kapaciteta druga dva puta. Naime, postoje tri različita puta od čvora A

do čvora C i to su: A, B, C; A, B, E, C i A, D, C. Težina kritične grane na svakom od ovih puteva je sledeća: na putu A, B, C je 2, na putu A, B, E, C je 1, a na putu A, D, C je 2. Maksimum vrednosti težina kritičnih grana 2, 1, 2 je 2, tako da su putevi A, B, C i A, D, C upravo najširi putevi. U zavisnosti od potrebe možemo pronaći i vratiti sva rešenja ovog problema ili prvo rešenje na koje naiđemo, npr. put A, B, C.

Najširi put u grafu ne mora biti jedinstven, što prethodni primer i ilustruje. U implementaciji naših algoritama u narednim poglavljima fokus će biti na pronalaženju proizvoljnog najšireg puta, tj. u ovom radu nećemo se baviti pronalaženjem svih rešenja. Dati algoritam se lako modifikuje da umesto jednog vraća sve najšire puteve.



Slika 1.3: Primer nalaženja najšireg puta u grafu

1.3 Opis primena

Problem najšireg puta ima širok opseg primena u oblastima kao što su rutiranje internet mreže (eng. *network routing problem*), proces montaže većeg broja slika u jednu i teorija glasanja (eng. *voting theory*).

Neke poznate specifične primene problema najšireg puta su:

- Pronalazak rute sa najvećom brzinom prenosa u internet mreži
- Računanje najjačih lanaca (eng. *strongest candidate chains*) u Šulcovom metodu koji odlučuje pobednika na izborima
- Formiranje mozaika od nekoliko digitalnih slika

Pronalazak rute sa najvećom brzinom prenosa u internet mreži

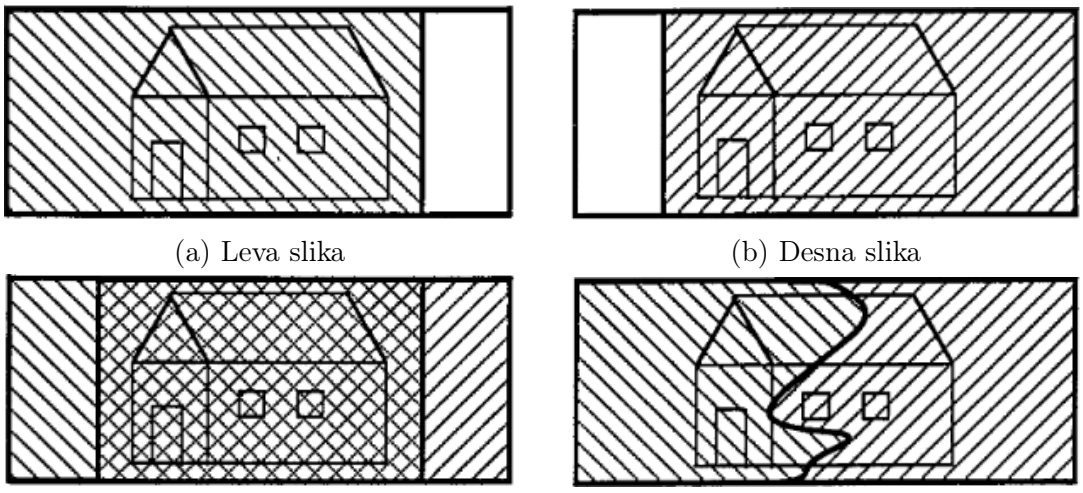
Ovaj problem se rešava direktno primenom algoritma za rešavanje problema najšireg puta. Pošto se prenos podataka u internet mreži vrši paket po paket, protok od tačke A do tačke C, koji se vrši preko tačke B, jednak je minimumu protoka od A do B i protoka od B do C. Na primer, ako je potrebno preneti 100 Mb podataka, pri čemu je protok od A do B 10 Mb/s, dok je protok od B do C 20 Mb/s, tada je protok od A do C jednak 10 Mb/s. Naime, prilikom prenosa podaci veličine 100 Mb dele se na pakete koji se, jedan po jedan, šalju kroz mrežu. Kada jedan paket pređe put od A do B, on se odmah šalje od B do C, ne čekajući prenos ostatka poruke, pa se zbog toga ukupan protok računa na taj način. Svaki ruter u mreži se predstavlja kao čvor u grafu. Ako postoji optički ili LAN kabl koji direktno spaja dva rutera tada između odgovarajućih čvorova u grafu postoji grana, a njena težina jednaka je protoku kabla koji spaja rutere. Pronalaženje optimalne rute prenosa podataka između dva rutera u internet mreži može se svesti na računanje najšireg puta između odgovarajućih čvorova u grafu.

Računanje najjačih lanaca u Šulcovom metodu koji odlučuje pobednika na izborima

Postoje različiti glasački sistemi. Na primer, često korišćen sistem je onaj u kome glasač bira jednog od svih ponuđenih kandidata. Međutim, ovakav sistem je neprecizan jer ne znamo šta glasač misli o kandidatima za koje nije glasao. Postoji sistem glasanja u kome glasač rangira sve kandidate od najboljeg do najgoreg tj. dodeljuje im brojeve od 1 do n (ako je n broj kandidata). Za ovakav sistem glasanja postoje raličiti metodi koji određuju pobednika, a jedan od najpreciznijih metoda je Šulcov metod. On funkcioniše po sledećem principu: formira se potpun usmeren težinski graf kod kog svaki čvor predstavlja jednog kandidata na izborima. Težina grane (A, B) jednaka je razlici broja glasača koji preferiraju kandidata A u odnosu na kandidata B i broja glasača koji preferiraju kandidata B u odnosu na kandidata A. Na primer, ako imamo pet glasača, od kojih tri preferiraju kandidata A u odnosu na kandidata B, dok dva preferiraju kandidata B u odnosu na kandidata A tada će težina grane (A, B) u grafu biti jednaka 1, dok će težina grane (B, A) biti jednaka -1. U formiranom grafu za svaka dva kandidata A i B računamo kapacitet k_1 najšireg puta od A do B (najširi put od A do B se u literaturi u opisu Šulcovog metoda naziva najjači lanac od A do B), kao i kapacitet k_2 najšireg puta od B do A. Ako

je $k_1 > k_2$ tada kažemo da A pobeđuje B. Kandidat koji pobeđuje sve ostale naziva se Šulcov pobednik i on uvek postoji i jedinstven je [7]. Upravo ovaj kandidat biće prema Šulcovom metodu pobednik izbora.

Formiranje mozaika od nekoliko digitalnih slika



(a) Leva slika (b) Desna slika

(c) Zajednički deo (d) Dobijen mozaik

Slika 1.4: Formiranje mozaika od dve digitalne slike

Pretpostavimo da imamo na raspolaganju veći broj digitalnih slika i da od njih želimo da formiramo panoramu. To možemo uraditi tehnikom sabijanja (eng. *merging*) dve mape u jednu koja će sadržati sve informacije. Izazov je u tome da date slike mogu imati različito osvetljenje, intenzitet boja itd. Jedan način da se formira mozaik je da se naprave aproksimacije tako da je svaki piksel rezultujuće slike predstavljen pikselom jedne od ulaznih slika. Cilj je pronaći optimalnu aproksimaciju. To se postiže korišćenjem problema najšireg puta. Na primer, ako spajamo dve slike koje prikazuju isti pejzaž u panoramu kao na slici 1.4, prvo pronalazimo njihov zajednički deo (end. common area). Piksele koji ne pripadaju zajedičkom delu slika uzimamo direktno za panoramu. Neka je, bez gubljenja opštosti, zajednički deo za date dve slike zapravo cela slika. Formirajmo matricu a gde a_{ij} predstavlja apsolutnu razliku intenziteta piksela (i, j) prve slike i piksela (i, j) druge slike. Nakon toga formiramo neusmeren težinski graf G u kom je svaki piksel predstavljen jednim čvorom. Dva piksela su susedna u grafu G ako dele istu vertikalnu ili horizontalnu granicu na slici. Težina grane između piksela (i, j) i piksela (k, l) u grafu G se postavlja na $\min\{-a_{ij}, -a_{kl}\}$. Definišimo PS-put kao prost put između proizvoljnog

piksela u prvoj vrsti matrice a i proizvoljnog piksela u poslednjoj vrsti matrice a. Cena PS-puta se definiše kao $f(PS) = \max a_{ij}$, $(i, j) \in PS$. Cilj je pronaći PS-put minimalne cene. Rezultujuća panorama će biti formirana od piksela prve slike koji se nalaze levo od piksela PS puta minimalne cene, i od piksela druge slike koji se nalaze desno od piksela PS puta minimalne cene. Lako je pokazati da je cena PS-puta minimalne cene jednaka maksimumu kapaciteta svih puteva između proizvoljnog čvora grafa G koji predstavlja piksel prve vrste matrice a i proizvoljnog čvora grafa G koji predstavlja piksel poslednje vrste matrice a. Detaljnije objašnjenje se može naći u [8].

U nastavku je dat kratak pregled rada kroz poglavlja.

- Osnovni grafovski algoritmi

  U ovom poglavlju predstavljen je opis i pseudokod osnovnih grafovskih algoritama koji se često koriste u daljem radu. Neki od prikazanih algoritama su pronalazak puta između dva čvora u grafu, pronalazak puta između dva čvora u grafu pri čemu se ignorišu grane čija je težina manja od neke vrednosti itd.

- Algoritmi za rešavanje problema najšireg puta

  U ovom poglavlju predstavljen je opis i pseudokod algoritama za rešavanje grafovskog problema najšireg puta.

  Algoritmi koji se obrađuju su: algoritam grube sile (NPGS), algoritam najšireg puta zasnovan na Dijkstrinoj ideji (NPD), algoritam najšireg puta koji koristi binarnu pretragu po dužini kritične grane ciljnog čvora (NPBP), algoritam najšireg puta zasnovan na sabijanju komponenti povezanosti (NPSK) i algoritam najšireg puta koji sortira težine grana (NPSG).

- Implementacija i evaluacija

  U ovom poglavlju implementirani algoritmi se upoređuju na testnom uzorku grafova. Vremena izvršavanja algoritama su grafički predstavljena u zavisnosti od veličine ulaznog grafa.

- Zaključak

  U ovom poglavlju izneti su glavni zaključci istraživanja i date smernice za budući rad.

Glava 2

Osnovni grafovski algoritmi

U ovoj glavi biće izloženi osnovni grafovski algoritmi koji služe kao gradivni element naprednijih algoritama predstavljenih u ovom radu. Pošto se ovi algoritmi često pozivaju u okviru drugih algoritama njihova efikasnost je od krucijalnog značaja. Implementacija osnovnih grafovskih algoritama u suštini se oslanja na obilazak grafa u dubinu, pri čemu se, u zavisnosti od potrebe, modifikuju ulazna i izlazna obrada čvora ili grane.

2.1 Ispitivanje da li je neki čvor dostižan iz nekog drugog

Prilikom razmatranja grafovskih algoritama često se postavlja pitanje da li je u datom grafu neki čvor dostižan iz nekog drugog, odnosno da li postoji put između data dva čvora.

Problem 2.1.1. Neka je dat (usmeren ili neusmeren) graf $G = (V, E)$, i njegova dva čvora $s, d \in V$. Ispitati da li je čvor d dostižan iz čvora s.

Algoritam 1 se može upotrebiti da reši problem 2.1.1.

Algoritam 1 Algoritam postojiPut

postojiPut(G, s, d)
Ulaz: graf $G = (V, E)$ i njegova dva čvora s i d
Izlaz: *true* ako postoji put u grafu G od čvora s do čvora d, inače $false$

```
1: begin
2: for v ∈ V do
3:     v.Oznaka ← false
4: end for
5: return postojiPutDFS(G, s, d)
6: end
```

U algoritmu `postojiPut` se inicijalno postavlja da nijedan čvor nije označen, a nakon toga vrši obilazak grafa u dubinu iz čvora s pozivom funkcije `postojiPutDFS`. Ako je čvor d dostižan iz čvora s tada će on biti obrađen tokom obilaska grafa, i funkcija `postojiPutDFS` će vratiti *true*.

Algoritam 2 Algoritam postojiPutDFS

postojiPutDFS(G, v, d)
Ulaz: graf $G = (V, E)$ i njegova dva čvora v i d
Izlaz: *true* ako važi $v = d$ ili ako postoji put u grafu G od čvora v do čvora d, inače $false$

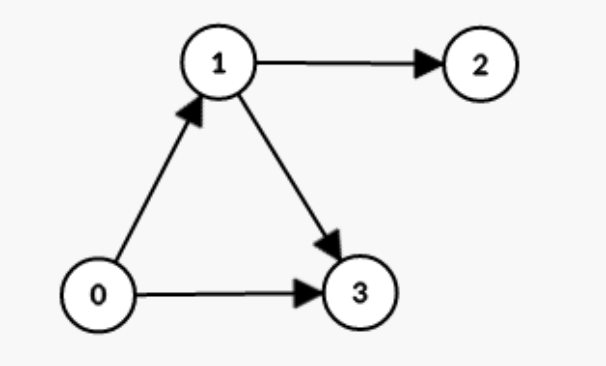
```
1: begin
2: v.Oznaka ← true
3: if v = d then
4:     return true
5: end if
6: for sve grane (v, w) do
7:     if w.Oznaka = false then
8:         if postojiPutDFS(G, w, d) then
9:             return true
10:        end if
11:    end if
12: end for
13: return false
14: end
```

Složenost. Algoritam `postojiPut` sastoji se od inicijalizacije polja *Oznaka* za svaki čvor, što se izvršava u vremenu $O(|V|)$, i od poziva funkcije `postojiPutDFS`, koja predstavlja obilazak grafa u dubinu. U algoritmu `postojiPutDFS` svaka grana se obilazi najviše jednom pa je njegova vremenska složenost $O(|E|)$. Zbog dubine

rekurzije prostorna složenost algoritma `postojiPutDFS` je $O(|V|)$. Dakle, vremenska složenost algoritma `postojiPut` je $O(|V| + |E|)$, a prostorna $O(|V|)$.

2.2 Formiranje puta od niza prethodnika

Prilikom obilaska grafa često je potrebno održavati niz prethodnika čvorova u kome se za svaki čvor pamti njegov prethodnik u DFS obilasku grafa, tj. roditelj u DFS stablu. Na primer, tokom obilaska grafa na slici 2.1 iz čvora 0, prvo obilazimo čvor 1. Sada obrađujemo susede čvora 1 - prvo čvor 2. Pošto čvor 2 nema suseda sada obrađujemo sledeći sused čvora 1, a to je 3. Dakle, niz prethodnika u ovom slučaju je $[-1, 0, 1, 1]$. Broj -1 označava da odgovarajući čvor nema prethodnika.



Slika 2.1: Formiranje niza prethodnika

Neka je dat startni čvor s iz koga pokrećemo DFS obilazak. Nakon izvršavanja DFS obilaska, u nizu prethodnika, za svaki čvor, nalazi se njegov prethodnik na putu od startnog čvora do njega. Da bismo rekonstruisali put od čvora s do čvora d koristimo niz prethodnika. Naime, potrebno je da se vratimo unazad nizom prethodnika od čvora d do čvora s. Način kako to možemo realizovati predstavljen je algoritmom 3.

Primer 2.2.1. Ako rekonstruišemo put od čvora 0 do čvora 3 u grafu datom na slici 2.1 na osnovu niza prethodnika $[-1, 0, 1, 1]$, krećemo od čvora 3. Njegov prethodnik je čvor 1. Sada je trenutni put $1, 3$. Na kraju, prethodnik čvora 1 je čvor 0 i formirani put je $0, 1, 3$.

Algoritam 3 Algoritam formirajPutOdNizaPrethodnika

$formirajPutOdNizaPrethodnika(s, d, pret)$
Ulaz: dva čvora s i d i niz prethodnika $pret$
Izlaz: put od s do d ako postoji, inače $null$

```
1: begin
2: Inicijalizuj prazan put p
3: if pret[d] = −1 then
4:     return null
5: end if
6: t ← d
7: while t ≠ s do
8:     dodaj t na početak puta p
9:     t ← pret[t]
10: end while
11: dodaj s na početak puta p
12: return p
13: end
```

Složenost. Ako odaberemo odgovarajuću strukuru podataka za realizaciju puta u grafu dodavanje elementa na početak puta se vrši u konstantnom vremenu, pa je složenost tela `while` petlje $O(1)$. Broj iteracija petlje jednak je dužini puta koja je u najgorem slučaju jednaka $|V|$. Dakle, vreme izvršavanja petlje pa i celog algoritma je $O(|V|)$. Pošto put čuvamo kao listu čvorova prostorna složenost algoritma je $O(|V|)$.

Napomena. U C++ implementaciji put je predstavljen preko strukture podataka *vector*. Put se u vektoru čuva u obrnutom redosledu. Samim tim, dodavanje čvora na početak puta se svodi na dodavanje elementa na kraj vektora što se izvršava u vremenskoj složenosti $O(1)$.

2.3 Pronalazak puta u grafu

Da bismo pronašli najširi put između dva data čvora u grafu potrebno je da umemo da pronađemo proizvoljni put između ta dva čvora. Dodatno, algoritam pronalaska puta mora biti efikasan. U daljem tekstu je predstavljen pseudokod algoritma `nadjiPut`.

Problem 2.3.1. Neka je dat (usmeren ili neusmeren) graf $G = (V, E)$, i njegova dva čvora $s, d \in V$. Pronaći proizvoljni put od čvora s do čvora d u grafu G. Ako traženi put ne postoji vratiti prazan put.

Algoritam 4 Algoritam nadjiPut

$nadjiPut(G, s, d)$
Ulaz: graf $G = (V, E)$ i njegova dva čvora s i d
Izlaz: put u grafu G od čvora s do čvora d ako postoji, inače *null*

```
1: begin
2: for v ∈ V do
3:     v.Oznaka ← false
4:     pret[v] ← −1
5: end for
6: nadjiPutDFS(G, s, pret)
7: put ← formirajPutOdNizaPrethodnika(s, d, pret)
8: return put
9: end
```

Algoritam `nadjiPut` inicijalizuje polje $Oznaka$ i niz prethodnika za svaki čvor i nakon toga vrši obilazak grafa u dubinu pozivom funkcije `nadjiPutDFS`. Na osnovu niza prethodnika, popunjenog tokom obilaska grafa, formira se put od čvora s do čvora d i vraća kao rezultat.

Postupak obilaska grafa u dubinu je opisan algoritmom `nadjiPutDFS`.

Algoritam 5 Algoritam nadjiPutDFS

$nadjiPutDFS(G, s, pret)$
Ulaz: graf $G = (V, E)$ i njegov čvor s, nepopunjen niz prethodnika $pret$
Izlaz: popunjen niz prethodnika $pret$

```
1: begin
2: s.Oznaka ← true
3: for sve grane (s, v) do
4:     if v.Oznaka = false then
5:         pret[v] ← s
6:         nadjiPutDFS(G, v, pret)
7:     end if
8: end for
9: end
```

Složenost. Algoritam `nadjiPut` sastoji se od inicijalizacije polja $Oznaka$ za svaki čvor i niza prethodnika, što se izvršava u vremenu $O(|V|)$, i od poziva funkcije `nadjiPutDFS`, koja predstavlja obilazak grafa u dubinu. U algoritmu `nadjiPutDFS` svaka grana se obilazi najviše jednom pa je njegova vremenska složenost $O(|E|)$. Zbog dubine rekurzije prostorna složenost algoritma `nadjiPutDFS` je $O(|V|)$. Dakle, vremenska složenost algoritma `nadjiPut` je $O(|V| + |E|)$, a prostorna $O(|V|)$.

2.4 Pronalazak puta u grafu, za dato donje ograničenje težine grane koje put sadrži

Nekada je potrebno da pronađemo put između dva čvora u grafu, tako da težine svih grana na putu budu težine veće ili jednake od neke vrednosti. Ako nam je, na primer, poznat kapacitet najšireg puta između dva čvora, možemo pronaći najširi put u grafu između ta dva čvora.

Problem. Neka je dat (usmeren ili neusmeren) težinski graf $G = (V, E)$, sa težinama grana c_e $(e \in E)$, i njegova dva čvora $s, d \in V$, kao i vrednost M. Odrediti put od čvora s do čvora d u grafu G, čije su sve težine grana veće ili jednake M.

Algoritam 6 Algoritam nadjiPutUPodgrafu

$nadjiPutUPodgrafu(G, s, d, M)$
Ulaz: težinski graf $G = (V, E)$, njegova dva čvora s i d, i vrednost M
Izlaz: put od čvora s do čvora d u grafu $G' = (V, E')$, gde je $E' = \{e \in E : c_e \geq M\}$ ako postoji, inače *null*

```
1: begin
2: for v ∈ V do
3:     v.Oznaka ← false
4:     pret[v] ← −1
5: end for
6: nadjiPutUPodgrafuDFS(G, s, M, pret)
7: put ← formirajPutOdNizaPrethodnika(s, d, pret)
8: return put
9: end
```

U algoritmu `nadjiPutUPodgrafu` se vrši inicijalizacija polja $Oznaka$ i niza prethodnika za svaki čvor, a nakon toga vrši se obilazak u dubinu podgrafa grafa G sa istim skupom čvorova i podskupom skupa grana čije su težine veće ili jednake M pozivom funkcije `nadjiPutUPodgrafuDFS`. Na osnovu niza prethodnika, popunjenog tokom obilaska grafa, formira se put i vraća rešenje.

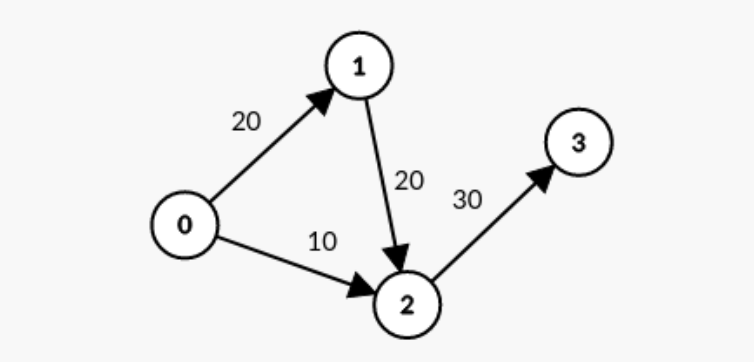
Postupak obilaska podgrafa u dubinu je opisan algoritmom `nadjiPutUPodgrafuDFS` i sličan je algoritmu `nadjiPutDFS`, s tim što prilikom razmatranja grana koje iz datog čvora vode do trenutno neoznačenih čvorova upoređujemo težinu trenutne grane sa M i ako je manja od M ne razmatramo je.

Algoritam 7 Algoritam nadjiPutUPodgrafuDFS

$nadjiPutUPodgrafuDFS(G, s, M, pret)$
Ulaz: graf $G = (V, E)$ i njegov čvor s, nepopunjen niz prethodnika $pret$
Izlaz: popunjen niz prethodnika $pret$

```
1: begin
2: s.Oznaka ← true
3: for sve grane (s, v) do
4:     if v.Oznaka = false then
5:         if c(s, v) ≥ M then
6:             pret[v] ← s
7:             nadjiPutUPodgrafuDFS(G, v, M, pret)
8:         end if
9:     end if
10: end for
11: end
```

Složenost. Vremenska složenost algoritma `nadjiPutUPodgrafu` iznosi $O(|V| + |E|)$, dok je prostorna složenost jednaka $O(|V|)$.



Slika 2.2: Primer pronalaska puta

Primer 2.4.1. Neka je dat težinski usmeren graf G na slici 2.2. Opišimo rad algoritma `nadjiPutUPodgrafu` za ulazne argumente $(G, 0, 3, 20)$.

Na početku izvršavanja algoritma su svi čvorovi neoznačeni i njihovi prethodnici nisu definisani. Najpre pozivamo funkciju `nadjiPutUPodgrafuDFS` za čvor 0 i on se označava. Njegovi neoznačeni susedi su 1 i 2. Pošto je težina grane $(0, 2)$ manja od 20, razmatra se samo grana ka čvoru 1. Postavljamo $pret[1] = 0$. Pozivamo funkciju `nadjiPutUPodgrafuDFS` za čvor 1. Čvor 1 se označava. Njegov jedini neoznačen sused je čvor 2. Pošto je težina grane $(1, 2)$ veća ili jednaka od 20, čvor 2 se sledeći obilazi. Postavljamo $pret[2] = 1$. Pozivamo `nadjiPutUPodgrafuDFS` za čvor 2. Čvor

2 se označava. Njegov jedini neoznačen sused je čvor 3. Pošto je težina grane $(2, 3)$ veća ili jednaka od 20, čvor 3 se sledeći obilazi. Postavljamo $pret[3] = 2$. Pozivamo `nadjiPutUPodgrafuDFS` za čvor 3 i on se označava. Pošto čvor 3 nema suseda, algoritam `nadjiPutUPodgrafuDFS` se ovde završava. Sledi formiranje puta od niza prethodnika. Poziva se `formirajPutOdNizaPrethodnika`$(0, 3, pret)$. Na početku je put prazan tj. $p = \{\}$. Promenljiva t je inicijalno jednaka 3. Sledi izvršavanje `while` petlje. Nakon izvršavanja prve iteracije petlje dobijamo $p = \{3\}$, $t = 2$. Nakon izvršavanja druge iteracije petlje je $p = \{2, 3\}$, $t = 1$. Nakon izvršavanja treće iteracije dobija se $p = \{1, 2, 3\}$, $t = 0$. Kako je $t = 0$ petlja se prekida. Na kraju dodajemo s na početak puta p, te, konačno, put postaje $p = \{0, 1, 2, 3\}$.

2.5 Nalaženje komponenti povezanosti neusmerenog grafa

Problem 2.5.1. Neka je dat neusmeren graf $G = (V, E)$. Odrediti njegove komponente povezanosti.

Algoritam 8 Algoritam komponentePovezanosti

komponentePovezanosti(G)
Ulaz: neusmeren graf $G = (V, E)$
Izlaz: $v.Komp$ za svaki čvor v dobija vrednost jednaku rednom broju komponente povezanosti koja sadrži v

```
1:  begin
2:  for v ∈ V do
3:      v.Oznaka ← false
4:  end for
5:  rbKomp ← 1                          ▷ tekući redni broj komponente
6:  for svi čvorovi v ∈ V do
7:      if v.Oznaka = false then
8:          komponentePovezanostiDFS(G, v, rbKomp)
9:          rbKomp ← rbKomp + 1
10:     end if
11: end for
12: end
```

U algoritmu `komponentePovezanosti` se inicijalno postavlja da nijedan čvor nije označen, zatim se postavlja redni broj komponente na 1 i nakon toga vrši obi-

lazak grafa u dubinu pozivom funkcije `komponentePovezanostiDFS`, dok god ima neoznačenih čvorova. Ako je neki čvor dostižan iz čvora koji se trenutno obilazi tada će on biti označen tokom obilaska grafa i biće mu dodeljen odgovarajući redni broj komponente. Postupak obilaska grafa u dubinu je opisan algoritmom `komponentePovezanostiDFS`.

Algoritam 9 Algoritam komponentePovezanostiDFS

$komponentePovezanostiDFS(G, s, rbKomp)$
Ulaz: graf $G = (V, E)$, njegov čvor s i $rbKomp$ redni broj komponente povezanosti
Izlaz: čvorovi koji su dostižni iz s su označeni i popunjeno im je polje $Komp$ brojem $rbKomp$

```
1: begin
2: s.Oznaka ← true
3: s.Komp ← rbKomp
4: for sve grane (s, v) do
5:     if v.Oznaka = false then
6:         komponentePovezanostiDFS(G, v, rbKomp)
7:     end if
8: end for
9: end
```

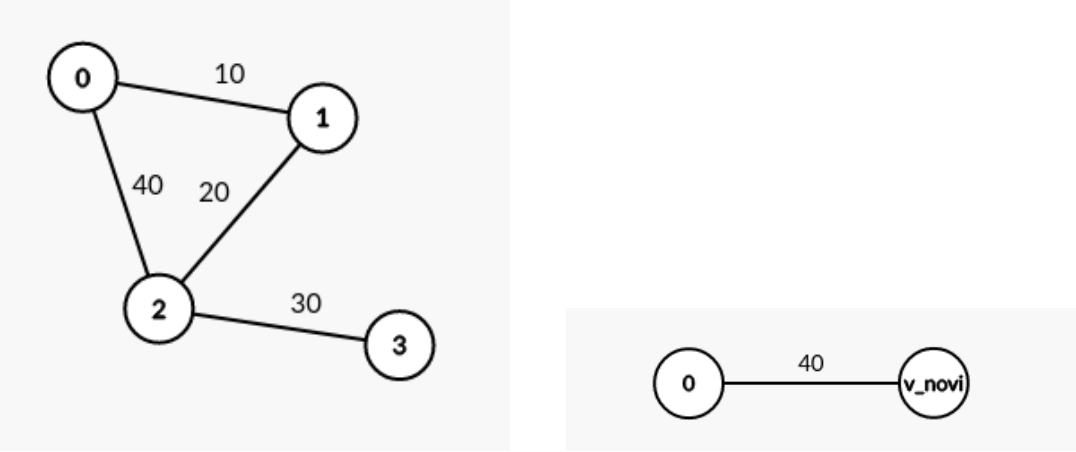
Složenost. Algoritam `komponentePovezanosti` sastoji se od inicijalizacije polja $Oznaka$ za svaki čvor, što se izvršava u vremenu $O(|V|)$, i od poziva funkcije `komponentePovezanostiDFS`, koja predstavlja obilazak grafa u dubinu. Algoritam `komponentePovezanostiDFS` se poziva dok svi čvorovi ne postanu označeni, najviše $|V|$ puta, ako je u pitanju graf bez grana. Izvršavanjem algoritma `komponentePovezanosti` svaki čvor se obilazi tačno jednom jer se čvorovi označavaju, a svaka grana obilazi tačno dva puta, po jednom sa svakog kraja, pa je njegova vremenska složenost $O(|V| + |E|)$. Zbog čuvanja oznake i rednog broja komponente za svaki čvor kao i zbog dubine rekurzije prostorna složenost algoritma je $O(|V|)$.

2.6 Sabijanje skupa čvorova grafa

Neka je dat neusmeren težinski graf $G = (V, E)$ i skup čvorova $S \subseteq V$. *Sabijanje skupa čvorova grafa* (eng. *node shrinking*) predstavlja formiranje novog grafa $G' = (V', E')$, tako da je $V' = (V \setminus S) \cup \{v_{novi}\}$. Naime, svi čvorovi iz skupa S u grafu G su predstavljeni jednim čvorom v_{novi} u grafu G', dok se skup E' formira na sledeći

način: čvorovi $u, v \in V \setminus S$ su susedni u G' ako i samo ako su susedni u G, pri čemu težina grane između ovih čvorova ostaje ista kao u G. S druge strane, čvor $u \in V \setminus S$ i čvor v_{novi} su susedni u G' ako i samo ako postoji čvor $w \in S$, tako da su u i w susedni u G. Težina grane u grafu G' između čvorova u i v_{novi} jednaka je maksimumu težina grana između u i svih čvorova iz skupa S u polaznom grafu G. Zbog jednostavnosti izlaganja razmotrićemo tehniku sabijanja skupa čvorova grafa samo u slučaju neusmerenog grafa.

Primer 2.6.1. Na slici 2.3a dat je težinski neusmeren graf G. Na slici 2.3b dat je graf G' koji se dobija sabijanjem skupa čvorova $S = \{1, 2, 3\}$ grafa G. Primetimo da je težina grane $(0, v_{novi})$ jednaka 40 zato što je vrednost 40 maksimum težina grana između čvora 0 i svih čvorova skupa S.



(a) G (b) G'

Slika 2.3: Sabijanje grafa G u odnosu na skup $\{1, 2, 3\}$

U Algoritmu 10 prikazan je postupak sabijanja skupa čvorova u grafu. Pretpostavljamo da je graf implementiran listom susedstva. Napomenimo da u slučaju neusmerenog grafa $G = (V, E)$, ako su neka dva čvora $u, v \in V$ susedna, tada se obe grane (u, v) i (v, u) nalaze u skupu E.

Algoritam 10 Algoritam sabijanje

sabijanje(G, S)
Ulaz: neusmeren težinski graf $G = (V, E, c)$ i skup čvorova S
Izlaz: novi graf $G^{'}$ koji je formiran od grafa G sabijanjem njegovih čvorova iz skupa S

```
1:  begin
2:  Inicijalizuj graf G' = (V', E', c'), sa skupom čvorova V' = (V \ S) ∪ {v_novi}, i praznim skupom grana
3:  for svaki čvor v ∈ V \ S do
4:      max ← −∞
5:      for svaka grana (v, w) u G do
6:          if w ∈ V \ S then
7:              dodati granu (v, w) u E'
8:              c'(v, w) ← c(v, w)
9:          else
10:             if max < c(v, w) then
11:                 max ← c(v, w)
12:             end if
13:         end if
14:     end for
15:     if max ≠ −∞ then
16:         dodaj grane (v, v_novi) i (v_novi, v) u E'
17:         c'(v, v_novi) ← max
18:         c'(v_novi, v) ← max
19:     end if
20: end for
21: return G'
22: end
```

Složenost. U spoljašnjoj `for` petlji kroz svaki čvor iz skupa $V \setminus S$ se prolazi tačno jednom. Algoritam obrađuje svaku granu skupa E najviše jednom, pa vremenska složenost algoritma iznosi $O(|V|+|E|)$. Obratimo pažnju da se grane između čvorova iz skupa S ne obrađuju u algoritmu.

Zbog potrebe formiranja novog grafa $G^{'}$ prostorna složenost algoritma iznosi $O(|V| + |E|)$.

Glava 3

Algoritmi za rešavanje problema najšireg puta

U daljem tekstu prikazani su algoritmi koji rešavaju Problem 1.2.1.

3.1 Algoritam grube sile

Tehnika konstrukcije algoritama grubom silom je osnovna i najjednostavnija tehnika konstrukcije algoritama. Koristeći ovu tehniku algoritam se često jednostavno implementira, ali se može koristiti samo za male dimenzije problema. Algoritmi grube sile su baš onakvi kako im i ime nalaže: zasnivaju se na direktnom pristupu rešavanja problema koji se oslanja na čistu računarsku snagu. Algoritam grube sile isprobava svaku mogućnost pri rešavanju problema, a retko koristi napredne tehnike ili trikove da bi unapredio efikasnost.

U daljem tekstu predstavljen je algoritam grube sile za pronalaženje najšireg puta u težinskom grafu od izvora s do cilja d.

Neka je dat graf $G = (V, E)$ i dva njegova čvora: izvor koji ćemo označiti sa s i cilj koji ćemo označiti sa d. Zadatak je pronaći najširi put od izvora s do cilja d. Naravno, ako ne postoji put između ova dva čvora onda ne postoji ni najširi put. U suprotnom najširi put postoji: do njega možemo doći sortiranjem konačnog skupa puteva po kapacitetu i odabirom puta najmanjeg kapaciteta. On, međutim, ne mora biti jedinstven. Ukoliko najširi put nije jedinstven zadatak je vratiti proizvoljni najširi put između ova dva čvora.

Ideja algoritma grube sile za rešavanje ovog problema je sledeća. Najpre je potrebno odrediti *maxmin* (kapacitet) čvora d. To se postiže rekurzivnim prolaskom kroz sve moguće puteve koji polaze iz čvora s. Usput pamtimo vrednost grane najmanje težine na svakom putu. Ako smo u rekurzivnom obilasku čvorova naišli na čvor d, taj put je kandidat za najširi put, i ako je njegova grana najmanje težine *minGrana* veća od trenutne vrednosti promenljive *maxmin*, tada promenljivoj *maxmin* dodeljujemo vrednost *minGrana*.

U nastavku teksta prikazan je algoritam kojim se može izračunati vrednost *maxmin* čvora d.

Algoritam 11 Algoritam maxminGrubaSila

maxminGrubaSila(G, s, d)
Ulaz: $G = (V, E)$ (težinski graf) i njegova dva čvora s i d
Izlaz: kapacitet najšireg puta od s do d
1: **begin**
2: $maxmin \leftarrow -\infty$
3: *maxminGrubaSilaPomocna*$(G, s, d, +\infty, maxmin)$
4: **return** *maxmin*
5: **end**

Algoritam `maxminGrubaSila` izračunava kapacitet najšireg puta između data dva čvora u grafu. Ovaj algoritam poziva funkciju `maxminGrubaSilaPomocna` koja rekurzivno obilazi sve moguće proste puteve iz s. Kada se neki put završava u čvoru d, ako je kapacitet puta veći od trenutne vrednosti *maxmin* čvora d, taj kapacitet se dodeljuje promenljivoj *maxmin*. Argument *minGrana* funkcije `maxminGrubaSilaPomocna` predstavlja trenutnu vrednost kapaciteta puta koji se obilazi rekurzijom i prenosi se po vrednosti. Pošto tražimo minimum, početna vrednost promenljive *minGrana* se postavlja na $+\infty$. Argument *maxmin* predstavlja najveći kapacitet na koji smo trenutno naišli tokom rekurzije i prenosi se po referenci. Pošto tražimo maksimum, početna vrednost promenljive *maxmin* se postavlja na $-\infty$.

Rekurzivni poziv algoritma `maxminGrubaSilaPomocna` kreće od čvora u i dalje razmatra njegove susede v, i poredi težinu grane (u, v) sa tekućom vrednošću kapaciteta, i pamti granu minimalne težine *minGrana* na putu koji obilazi. Rekurzija se završava u dva slučaja. Prvi slučaj je kada trenutni čvor u koji se obilazi nema više suseda. Drugi mogući slučaj je kada je trenutni čvor koji se razmatra baš ciljni čvor. Tada trenutni kapacitet upoređujemo sa vrednošću promenljive *maxmin*. Ako

je kapacitet veći od $maxmin$, postavljamo vrednost $maxmin$ na trenutni kapacitet puta. Da bismo izbegli slučaj beskonačne rekurzije svaki čvor koji se nalazi u trenutnom putu se označava. Na taj način izbegavamo cikluse, odnosno obrađujemo samo proste puteve što i jeste cilj. Kako smo obišli sve moguće puteve koji polaze iz u, da bi u narednim rekurzivnim pozivima, koji nisu obišli taj čvor, on bio neoznačen, na kraju algoritma skidamo oznaku sa čvora u.

Algoritam 12 Algoritam maxminGrubaSilaPomocna

$maxminGrubaSilaPomocna(G, u, d, minGrana, maxmin)$
Ulaz: $G = (V, E)$ (težinski graf), njegova dva čvora u i d, trenutna vrednost kapaciteta puta koji se obilazi $minGrana$, trenutni $maxmin$ čvora d
Izlaz: kapacitet najšireg puta od u do d biće sačuvan u promenljivoj $maxmin$

```
1: begin
2: u.Oznaka ← true
3: if u = d then
4:     if maxmin < minGrana then
5:         maxmin ← minGrana
6:     end if
7: else
8:     for sve grane (u, v) do
9:         if v.Oznaka = false then
10:            novaMinGrana ← minGrana
11:            if c(u, v) < minGrana then
12:                novaMinGrana ← c(u, v)
13:            end if
14:            maxminGrubaSilaPomocna(G, v, d, novaMinGrana, maxmin)
15:        end if
16:    end for
17: end if
18: u.Oznaka ← false
19: end
```

Sada prikazujemo ciljni algoritam ovog odeljka. Algoritam `najsiriPutGrubaSila` (NPGS) ima zadatak da, za dati težinski graf $G = (V, E)$ i data dva čvora $s, d \in V$, pronađe najširi put od s do d. Algoritam najpre poziva funkciju `maxminGrubaSila` i pronalazi kapacitet najšireg puta $maxmin$. Zatim nekim od standardnih obilazaka grafa (npr. obilaskom grafa u dubinu), tražimo put u grafu G od čvora s do čvora d, pri čemu ignorišemo grane čije su težine manje od vrednosti $maxmin$, i vraćamo taj put kao rešenje.

Algoritam 13 Algoritam najsiriPutGrubaSila (NPGS)

$najsiriPutGrubaSila(G, s, d)$
Ulaz: $G = (V, E)$ (težinski graf) i njegova dva čvora s i d
Izlaz: najširi put u grafu G od čvora s do čvora d
1: **begin**
2: $maxmin \leftarrow maxminGrubaSila(G, s, d)$
3: $najsiriPut \leftarrow nadjiPutUPodgrafu(G, s, d, maxmin)$
4: **return** $najsiriPut$
5: **end**

Složenost. Razmotrimo najpre vremensku složenost Algoritma 12. Rekurzija prolazi kroz svaki mogući (prost) put koji počinje u čvoru s i završava se u čvoru d, pri čemu je za formiranje puta dužine k bilo potrebno $k-1$ rekurzivnih poziva funkcije. U najgorem slučaju, kada je graf potput, dužina puta koji se formira rekurzijom je u intervalu od 2 do $|V|$, a broj svih puteva dužine k koji počinju u s i završavaju se u d je $\binom{|V|-2}{k-2} \cdot (k-2)!$. Broj rekurzivnih poziva je u najgorem slučaju jednak:

$$\sum_{k=2}^{|V|}((k-1)\cdot\binom{|V|-2}{k-2}\cdot(k-2)!) \leq (|V|-1)\cdot\sum_{k=2}^{|V|}\frac{(|V|-2)!}{(|V|-k)!} = (|V|-1)\cdot\sum_{k=0}^{|V|-2}\frac{(|V|-2)!}{k!}$$

Iz poznate jednakosti $\sum_{k=0}^{n}\frac{n!}{k!} = e \cdot \Gamma^1(n+1,1) = \lfloor e \cdot n! \rfloor$ [12] i prethodne nejednakosti sledi da vremenska složenost algoritma iznosi $O((|V|-1)!)$. Zbog dubine rekurzije prostorna složenost funkcije `maxminGrubaSilaPomocna` je $O(|V|)$. Maksimalna dubina rekurzije, odgovara maksimalnoj veličini puta (ovde se misli koliko put sadrži čvorova), koja iznosi $|V|$ jer obrađujemo samo proste puteve.

Funkcija `maxminGrubaSila` se sastoji od inicijalizacije promenljive *maxmin* i poziva funkcije `maxminGrubaSilaPomocna`, pa je složenost ove dve funkcije jednaka.

Konačno razmotrimo složenost algoritma NPGS. Ovaj algoritam se sastoji od poziva dve funkcije `maxminGrubaSila` i `nadjiPutUPodgrafu`, pa je njegova složenost jednaka maksimumu složenosti ove dve funkcije i iznosi $O((|V|-1)!)$. Prostorna složenost algoritma NPGS je jednaka $O(|V|)$.

Vremenska složenost algoritma NPGS jasno ukazuje da ovaj algoritam nije upotrebljiv za grafove iole veće dimenzije.

[1] Nekompletna Gama funkcija.

3.2 Algoritam najšireg puta zasnovan na Dijkstrinoj ideji

Dijkstrin algoritam [1] traži put između dva čvora koji ima minimalnu sumu težina grana na putu, dok algoritam najšireg puta traži put između dva čvora koji ima maksimalnu vrednost minimalne težine grane. Algoritam najšireg puta zasnovan na Dijkstrinoj ideji (NPD) koristi isti kostur kao Dijkstrin algoritam i samo su statistike koje se primenjuju drugačije. Zbog bolje efikasnosti koristi se verzija Dijkstre koja koristi red sa prioritetom.

Red sa prioritetom je apstraktna struktura podataka koja pruža sledeće operacije: dodaj proizvoljni element, povećaj prioritet nekom elementu, pronađi element sa maksimalnim prioritetom i obriši element sa maksimalnim prioritetom. Takođe, sve ove operacije su efikasne u odnosu na broj elemenata koji se čuva u redu sa prioritetom. Upravo zbog efikasnosti spomenutih operacija koristimo u implementaciji ovu strukturu podataka.

Ideja algoritma NPD je da razmatramo čvorove grafa redom prema njihovom kapacitetu (kapacitet čvora v ili $maxmin_v$ jednak je kapacitetu najšireg puta od čvora s do čvora v). Prvo se razmatra čvor čiji je kapacitet najveći. U toku izvršavanja algoritma računamo kapacitet i najširi put čvorova.

Neka je (s, x) najduža grana grafa koja polazi iz startnog čvora s. Dokažimo da je čvor x upravo čvor najvećeg kapaciteta u grafu i da je (s, x) najširi put čvora x. Pretpostavimo da u grafu postoji čvor y i put p od s do y tako da je kapacitet puta p veći od $c(s, x)$, pri čemu $y \in V \setminus \{s\}$ i $p \neq (s, x)$. Pošto put p polazi is čvora s on mora da sadrži neku granu koja polazi iz čvora s. Sve grane koje polaze iz s imaju manju ili jednaku dužinu kao grana (s, x). Kako put p sadrži granu koja je manje ili jednake dužine kao (s, x), sledi da je kapacitet puta p manji ili jednak vrednosti $c(s, x)$, što je u suprotnosti sa pretpostavkom i dobijamo kontradikciju. Dakle, čvor x jeste čvor najvećeg kapaciteta u grafu i (s, x) je najširi put od čvora s do čvora x.

Dalje, biramo maksimum dužina grana koje izlaze iz s, bez grane (s, x), i dužina grana koje izlaze iz x. Ako je to neka od grana koje izlaze iz s, recimo (s, w), tada je $maxmin_w$ upravo dužina od (s, w), a najširi put od s do w je (s, w). Ako je pak to neka od grana koje izlaze iz x, na primer (x, w), tada je $maxmin_w$ jednako minimumu dužina grana (s, x) i (x, w), a najširi put od s do w je (s, x, w). Kako smo pronašli najšire puteve do čvorova x i w kažemo da su čvorovi s, x i w obrađeni (čvor s je obrađen po definiciji). Na sličan način, traženjem grane najveće dužine

koja polazi iz već obrađenih čvorova i završava u proizvoljnom neobrađenom čvoru, pronalazimo sledeći čvor najvećeg kapaciteta. Pošto obrađujemo jedan po jedan čvor naslućujemo da bi se matematička indukcija mogla iskoristiti za dokazivanje korektnosti algoritma. Pošto smo pronašli čvor x i najširi put od čvora s do čvora x i pošto znamo da je kapacitet čvora x veći od kapaciteta ostalih čvorova, čvor x može da nam posluži kao baza indukcije. Formulišimo sledeću induktivnu hipotezu.

Induktivna hipoteza. Za zadati graf i njegov čvor s, umemo da pronađemo k čvorova, koji zajedno sa s čine skup V_k, kao i njihove najšire puteve, i pronađeni najširi putevi nemaju manji kapacitet od najširih puteva između s i proizvoljnog čvora koji pripada skupu $V \setminus V_k$.

Znamo da je čvor x čvor najvećeg kapaciteta u grafu i pronašli smo najširi put od čvora s do čvora x. Time je bazni slučaj (slučaj $k = 1$) rešen. Pretpostavimo da smo pronašli k čvorova koji zajedno sa s čine skup V_k i zadovoljavaju induktivnu hipotezu. Opišimo pronalazak $(k+1)$-vog čvora i najšireg puta do njega. Neka je skup $S_k = \{(a, b) \in E : a \in V_k, b \in V \setminus V_k\}$, podskup skupa grana grafa E. Neka je grana (v, w) grana najveće dužine iz skupa S_k. Tvrdimo da je čvor w upravo $(k+1)$-vi čvor i da je najširi put od s do w unija najšireg puta od s do v i grane (v, w). Tada očigledno važi $maxmin_w = \min\{maxmin_v, c(v, w)\}$. Pretpostavimo da u grafu postoji čvor y i put p od s do y tako da je kapacitet puta p veći od $maxmin_w$, pri čemu $y \in V \setminus V_k$ i $(v, w) \notin p$. Pošto put p polazi iz čvora s koji pripada skupu V_k, i ponire u y koji pripada $V \setminus V_k$, on mora da sadrži neku granu iz skupa S_k. Sve grane iz skupa S_k imaju manju ili jednaku dužinu kao grana (v, w). Kako put p sadrži granu koja je manje ili jednake dužine kao (v, w), sledi da je kapacitet puta p manji ili jednak vrednosti $c(v, w)$. Kako je po pretpostavci kapacitet puta p veći od $maxmin_w$ sledi da je kapacitet puta p veći od $maxmin_v$ tj. $maxmin_y > maxmin_v$, što je u suprotnosti sa induktivnim korakom i dobijamo kontradikciju. Kako smo y birali kao proizvoljan čvor skupa $V \setminus V_k$ (dakle, razmatrali smo i slučaj $y = w$), ovim je dokazano da je w $(k+1)$-vi čvor i da je najširi put od s do w unija najšireg puta od s do v i grane (v, w).

Dakle, u svakom koraku algoritma pronalazimo sledeći čvor u nerastućem redosledu kapaciteta od čvora najvećeg do čvora najmanjeg kapaciteta. To se ostvaruje

pronalaženjem grane maksimalne dužine od grana koje izlaze iz V_k i ulaze u $V \setminus V_k$. Međutim, da bi se odredio sledeći čvor nije neophodno razmatrati sve takve grane. Za svaki čvor w čuvamo vrednost $w.maxmin$ koja predstavlja vrednost trenutnog kapaciteta tog čvora. Nakon što odredimo najširi put i $maxmin$ nekog čvora v, prođemo kroz njegove susede w i ažuriramo njihove $maxmin$ vrednosti na sledeći način:

$$maxmin_w = \max\{maxmin_w, \min\{maxmin_v, c(v, w)\}\}.$$

Neoznačeni čvor koji trenutno ima najveću vrednost $maxmin$ je upravo $(k+1)$-vi čvor i njegov kapacitet je jednak $maxmin$. Sve tako obrađene susede čuvamo u strukturi red sa prioritetom, sa ključevima jednakim trenutnoj vrednosti $maxmin$. Na taj način biramo maksimum od svih vrednosti $w.maxmin$ u vremenu $O(1)$. Početna vrednost $maxmin$ za svaki čvor je $-\infty$, i menja se ako je čvor susedan nekom obrađenom čvoru. Početna vrednost $maxmin$ za s je $+\infty$ i taj čvor se prvi obrađuje. U jednoj iteraciji `while` petlje obrađuje se neoznačen čvor koji trenutno ima najveći kapacitet. Algoritam se završava kada obradimo ciljni čvor ili kada smo obradili sve čvorove dostižne iz startnog čvora.

Hip je pogodna struktura podataka za implementaciju reda sa prioritetom. Razmotrimo nekoliko vrsta hipa. Posmatrajmo sledeću tabelu 3.1 i uporedimo teorijske granice za osnovne operacije hipa. Kao što vidimo Fibonačijev hip se najbolje uklapa u naše potrebe, i upravo ovaj hip ćemo koristiti u implementaciji reda sa prioritetom.

Operacija	Binarni hip	Binomni hip	Fibonačijev hip
Nađi max	$O(1)$	$O(1)$	$O(1)$
Obriši max	$O(\log n)$	$O(\log n)$	$O(\log n)$
Ubaci element	$O(\log n)$	$O(\log n)$	$O(1)$
Povećaj ključ	$O(\log n)$	$O(\log n)$	$O(1)$
Spajanje	$O(n)$	$O(\log n)$	$O(1)$

Slika 3.1: Poređenje teorijskih granica za varijante max-hipa

Algoritam 14 Algoritam najšireg puta zasnovan na Dijkstrinoj ideji (NPD)

$najsiriPutDijkstra(G, s, d)$
Ulaz: $G = (V, E)$ (težinski graf) i njegova dva čvora s i d
Izlaz: najširi put u grafu G od s do d ili *null* ako put ne postoji

1: **begin**
2: inicijalizuj prazan red sa prioritetom Q
3: **for** sve čvorove v grafa G **do**
4: $v.Oznaka \leftarrow false$
5: $pret[v] \leftarrow -1$
6: $v.maxmin \leftarrow -\infty$
7: **end for**
8: $s.maxmin \leftarrow +\infty$
9: dodaj $(s.maxmin, s)$ u Q
10: **while** Q nije prazan **do**
11: skini maksimalni element v iz Q
12: $v.Oznaka \leftarrow true$
13: **if** $v = d$ **then**
14: **break**
15: **end if**
16: **for** sve grane (v, w) takve da je $w.Oznaka = false$ **do**
17: **if** $w.maxmin < v.maxmin$ **and** $w.maxmin < c(v, w)$ **then**
18: $pret[w] \leftarrow v$
19: **if** $v.maxmin < c(v, w)$ **then**
20: $w.maxmin \leftarrow v.maxmin$
21: **else**
22: $w.maxmin \leftarrow c(v, w)$
23: **end if**
24: **if** $w \in Q$ **then**
25: povećaj prioritet čvora w na $w.maxmin$ u Q
26: **else**
27: dodaj $(w.maxmin, w)$ u Q
28: **end if**
29: **end if**
30: **end for**
31: **end while**
32: **if** $pret[d] = -1$ **then**
33: **return** *null*
34: **else**
35: oformi put *put* od niza prethodnika
36: **return** *put*
37: **end if**
38: **end**

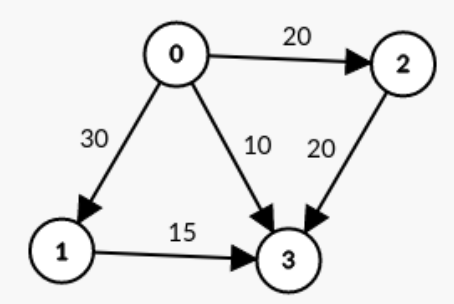
Složenost. Inicijalizacija praznog reda sa prioritetom ima složenost $O(1)$. Inicijalizacija početnih vrednosti za oznaku, prethodnika i kapacitet svakog čvora ima složenost $O(|V|)$. Složenost pronalaska maksimalnog elementa u redu sa prioritetom je $O(1)$. Neka je vremenska složenost brisanja maksimalnog elementa iz reda sa prioritetom T_b, složenost dodavanja elementa u red sa prioritetom T_d, a složenost povećanja prioriteta elementa u redu sa prioritetom T_u. Svaki čvor će biti obrađen tačno jednom u `while` petlji, pri čemu se u svakoj iteraciji `while` petlje briše maksimalni element, pa je složenost `while` petlje jednaka $O(|V| \cdot T_b)$. Svaka grana se obrađuje tačno jednom, i u zavisnosti od toga da li je ispunjen neki uslov, čvor u koji data grana ulazi se može dodati u red sa prioritetom ili se može uvećati prioritet tog čvora. Dakle, složenost obrade grana u grafu iznosi $O(|E| \cdot (T_d + T_u))$. Formiranje puta od liste prethodnika ima složenost $O(|V|)$. Stoga je ukupna vremenska složenost algoritma $O(|V| \cdot T_b + |E| \cdot (T_d + T_u))$. Razmotrimo nekoliko mogućih realizacija strukture podataka za čuvanje čvorova. Posmatrajmo tabelu 3.1. Ako za čuvanje čvorova koristimo Fibonačijev hip vremenska složenost je $O(|V| \cdot \log(|V|) + |E|)$. Ako za čuvanje čvorova koristimo binarni ili binomni hip vremenska složenost je $O((|V| + |E|) \cdot \log(|V|))$.

Prostorna složenost prikazanog algoritma je $O(|V|)$.

Napomena. U C++ implementaciji algoritma `najsiriPutDijkstra` koristi se `boost::heap::fibonacci_heap` kod koga su vremenske složenosti jednake: $T_d = O(1)$, $T_b = O(|V|)$, $T_u = O(\log |V|)$ [11]. Dakle, vremenska složenost algoritma NPD u C++ implementaciji je u najgorem slučaju $O(|V|^2)$. Međutim, pošto je amortizovana vremenska složenost brisanja iz hipa u boost biblioteci jednaka $O(\log |V|)$, dobijamo da je prosečna vremenska složenost algoritma NPD jednaka $O((|V| + |E|) \cdot \log(|V|))$.

Primer 3.2.1. Na slici 3.2 dat je primer izvršavanja algoritma NPD. Zadatak je pronaći najširi put u datom grafu od čvora 0 do čvora 3. Data je tabela koja opisuje vrednosti promenljivih na kraju svake iteracije `while` petlje. Iteracija 0 označava vrednosti promenljivih pre početka izvršavanja `while` petlje, dok iteracija 1 označava vrednosti promenljivih nakon izvršavanja prve iteracije petlje. Nakon izvršavanja `while` petlje formira se put od niza prethodnika, što smo obradili u prethodnoj glavi.

Iter.	Oznaka	pret	maxmin	Q
0	$\{0,0,0,0\}$	$\{-1,-1,-1,-1\}$	$\{+\infty,-\infty,-\infty,-\infty\}$	$[(+\infty,0)]$
1	$\{1,0,0,0\}$	$\{-1,0,0,0\}$	$\{+\infty,30,20,10\}$	$[(30,1),(20,2),(10,3)]$
2	$\{1,1,0,0\}$	$\{-1,0,0,1\}$	$\{+\infty,30,20,15\}$	$[(20,2),(15,3)]$
3	$\{1,1,1,0\}$	$\{-1,0,0,2\}$	$\{+\infty,30,20,20\}$	$[(20,3)]$
4	$\{1,1,1,1\}$	$\{-1,0,0,2\}$	$\{+\infty,30,20,20\}$	[]



Slika 3.2: Primer izvršavanja algoritma NPD

3.3 Algoritam najšireg puta koji koristi binarnu pretragu po dužini kritične grane ciljnog čvora

U literaturi se problem najšireg puta uglavnom razmatra za težinske grafove kod kojih su dužine grana celobrojne vrednosti: tada će i vrednost *maxmin* biti celobrojna. Primetimo da ukoliko je vrednost *maxmin* čvora d već poznata lako se pronalazi najširi put primenom algoritma `nadjiPutUPodgrafu`. Pošto je vremenska složenost algoritma `nadjiPutUPodgrafu` jednaka $O(|V|+|E|)$, ukoliko bismo umeli da izračunamo *maxmin* vrednost čvora d relativno brzo, tada bismo mogli da implementiramo efikasan algoritam.

Odredimo najpre težine minimalne i maksimalne grane u grafu – *minGrana* i *maxGrana*: njih možemo pronaći DFS obilaskom grafa u linearnom vremenu. Zatim, vršimo binarnu pretragu opsega $[minGrana, maxGrana]$ da bismo pronašli *maxmin* vrednost čvora d. Neka u datom trenutku pretražujemo opseg $[l,r]$. Najpre računamo srednju vrednost m intervala $[l,r]$. Zatim brišemo (ili, jednostavnije, ignorišemo) sve grane grafa čija je težina manja od m i proveravamo da li postoji put od s do d. Ako put postoji ažuriramo vrednost $maxmin = m$ kao i pronađeni put i dalje pretražujemo interval $[m+1,r]$. Ako put ne postoji, vraćamo obrisane grane (ili, jednostavnije, prestajemo da ih ignorišemo) i pretražujemo interval $[l,m-1]$. Pretraga se nastavlja dok je l manje ili jednako od r. Kada nejednakost prestane da

važi pretraga je završena i pronašli smo *maxmin* vrednost i najširi put.

Algoritam 15 Algoritam najšireg puta koji koristi binarnu pretragu po dužini kritične grane ciljnog čvora (NPBP)

$najsiriPutBinarnaPretraga(G, s, d)$
Ulaz: $G = (V, E)$ (težinski graf sa celobrojnim težinama grana) i njegova dva čvora s i d
Izlaz: najširi put u grafu G od s do d ili *null* ako put ne postoji

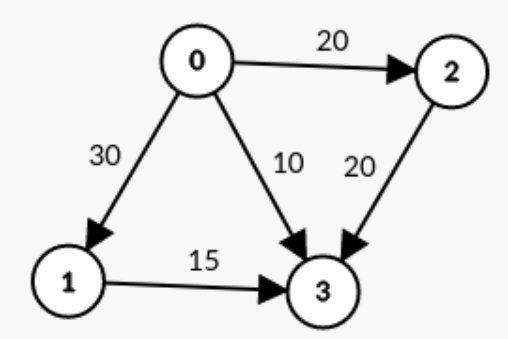
```
1: begin
2: minGrana ← dužina najmanje grane u grafu
3: maxGrana ← dužina najveće grane u grafu
4: najsiriPut ← null
5: l ← minGrana
6: r ← maxGrana
7: while l ≤ r do
8:     m ← (l+r)/2
9:     put ← nadjiPutUPodgrafu(G, s, d, m)
10:    if put ≠ null then
11:        najsiriPut ← put
12:        l ← m + 1
13:    else
14:        r ← m − 1
15:    end if
16: end while
17: return najsiriPut
18: end
```

Složenost. U telu `while` petlje ispitujemo da li postoji put u grafu što je linearne vremenske složenosti $O(|V| + |E|)$. Petlja se izvršava $\log(maxGrana - minGrana)$ puta, pa je ukupna vremenska složenost prethodnog algoritma $O((|V| + |E|) \cdot \log(maxGrana - minGrana))$. Pošto tokom izvršavanja algoritma održavamo tekući najširi put, prostorna složenost je $O(|V|)$.

Primer 3.3.1. Na slici 3.3 dat je primer izvršavanja algoritma NPBP. Zadatak je pronaći najširi put u datom grafu od čvora 0 do čvora 3. Data je tabela koja opisuje vrednosti promenljivih na kraju svake iteracije `while` petlje. Iteracija 0 označava vrednosti promenljivih pre početka izvršavanja `while` petlje, dok iteracija 1 označava vrednosti promenljivih nakon izvršavanja prve iteracije petlje. Kako je u iteraciji 1 vrednost m jednaka 20, i kako u datom grafu postoji put od čvora 0 do čvora 3 sa dužinama grana većim ili jednakim 20, čuvamo pronađeni put $0, 2, 3$ i ažuriramo vrednost l na 21. Primetimo da u svakoj narednoj iteraciji ne postoji put od čvora

0 do čvora 3 čije su grane sve dužina većih ili jednakih m, pa u svakom koraku ažuriramo vrednost r na $m - 1$, sve dok ne prestane da važi uslov `while` petlje.

Iteracija	najsiriPut	l	r	m
0	*null*	10	30	-
1	$0, 2, 3$	21	30	20
2	$0, 2, 3$	21	24	25
3	$0, 2, 3$	21	21	22
4	$0, 2, 3$	21	20	21



Slika 3.3: Primer izvršavanja algoritma NPBP

3.4 Algoritam najšireg puta zasnovan na sabijanju komponenti povezanosti

Neka je dat težinski neusmeren graf $G = (V, E)$ i njegova dva čvora s i d. Kao i u prethodno prikazanim algoritmima prvi cilj jeste odrediti *maxmin* vrednost čvora d. Algoritam `maxminSabijanjeKomponenti` pronalazi *maxmin* vrednost čvora d u neusmerenom težinskom grafu G za dati startni čvor s i ciljni čvor d. Osnovna ideja algoritma je da se čuva trenutni skup grana grafa i u svakom koraku traži *medijana* (broj iz skupa od m datih brojeva za koji važi da je $\lfloor m/2 \rfloor$ brojeva manje ili jednako od njega i da je $m - \lfloor m/2 \rfloor$ brojeva veće ili jednako od njega) težina trenutnog skupa grana i pita da li postoji put ako se ignorišu grane manje težine od dobijene medijane. Ako put postoji postavljamo vrednost *maxmin* na vrednost medijane i brišemo grane manje ili jednake težine vrednosti medijane. Ako put ne postoji brišu se grane veće od medijane tako što se sabija skup čvorova grafa koji čini jednu povezanu komponentu. Povezane komponente se dobijaju iz činjenice da ne postoji put od startnog do ciljnog čvora pod datim uslovima.

Poznato je [4, 5] da se medijana m brojeva može pronaći u linearnom vremenu tj. u $O(m)$ koraka [6, poglavlje 9.3]. U C++ implementaciji medijanu niza elemenata dobijamo koristeći STL funkciju `nth_element` koja ima linearnu prosečnu vremensku složenosti izvršavanja.

Sabijanje skupa čvorova grafa se može izvršiti u linearnom vremenu. Preciznije, ako je dat podskup $S \subseteq V$ skupa čvorova neusmerenog grafa $G = (V, E)$, možemo u linearnom vremenu konstruisati drugi graf sa skupom čvorova $(V \setminus S) \cup \{v_{novi}\}$ (gde v_{novi} sada predstavlja sabijen skup S tj. svi čvorovi iz skupa S su u novom grafu predstavljeni jednim čvorom), za koji važi da su čvorovi $v, w \in V \setminus S$ susedni ako i samo ako su v i w susedni u G (u ovom slučaju težina grane ostaje ista), a v_{novi} i $w \in V \setminus S$ su susedni ako i samo ako postoji neki čvor $v \in S$ tako da su v i w susedni čvorovi u G (u ovom slučaju težina grane uzima najveću vrednost od težina grana koje spajaju S i w u G).

Algoritam 16 Algoritam koji pronalazi vrednost maxmin zasnovan na sabijanju komponenti povezanosti

maxminSabijanjeKomponenti(G, s, d)
Ulaz: $G = (V, E)$ (težinski neusmereni graf) i njegova dva čvora s i d
Izlaz: vrednost najkraće grane na najširem putu od s do d tj. *maxmin*

1: **begin**
2: $brojIteracija \leftarrow 0$
3: **while** $brojIteracija < \lceil \log(|E|) \rceil$ **do**
4: $M \leftarrow$ medijana svih težina grana koje se trenutno nalaze u grafu G
5: $G' \leftarrow$[2]podgraf grafa G koji sadrži iste čvorove kao G, i grane od G koje su veće ili jednake M
6: **if** postoji put od s do d u grafu G' **then**
7: $maxmin \leftarrow M$
8: Izbriši sve grane grafa G koje imaju težinu manju ili jednaku M
9: **else**
10: Neka su $V_1, \ldots, V_k$ komponente povezanosti grafa G'
11: Sabij skupove čvorova $V_1, \ldots, V_k$ grafa G
12: **end if**
13: $brojIteracija \leftarrow brojIteracija + 1$
14: **end while**
15: **return** *maxmin*
16: **end**

[2]Graf G' se koristi da bismo lakše objasnili korake algoritma. U C++ implementaciji graf G' nećemo eksplicitno formirati već ćemo odgovarajuće operacije izvršavati na grafu G i prilikom obrade grana grafa G dodaćemo proveru da li je grana veća ili jednaka M.

U svakom koraku `while` petlje algoritma `maxminSabijanjeKomponenti` tražimo medijanu M dužina svih preostalih grana grafa i pitamo da li postoji put od čvora s do čvora d u podgrafu G' grafa G, koji se dobija od G brisanjem svih grana čije su težine manje od M. Ako postoji put od s do d u grafu G' tada postavljamo promenljivu *maxmin* na vrednost M i brišemo grane manje ili jednake M u grafu G. Pošto znamo da postoji put koji ima kapacitet M, grane manje ili jednake M nisu više od interesa za traženje puta čiji je kapacitet veći od M jer ako postoji put čiji je kapacitet veći od M taj put neće sadržati granu manju ili jednaku M. Kako se graf G u Algoritmu 16 prenosi po vrednosti, brisanje njegovih grana neće uticati na pronalazak puta u Algoritmu 17.

Ako ne postoji put između s i d u grafu G' tada izračunavamo komponente povezanosti $V_1, \ldots, V_k$ grafa G'. Nakon toga se vrši sabijanje skupova čvorova $V_1, \ldots, V_k$. Sabijanjem skupova čvorova $V_1, \ldots, V_k$ u grafu G ostaju samo grane između čvorova koji pripadaju različitim komponentama povezanosti $V_1, \ldots, V_k$. Ako bi u grafu G nakon sabijanja preostala neka grana (a, b) čija je težina veća ili jednaka M, tada bi ta grana pripadala i grafu G' pa bi čvorovi a i b pripadali istoj komponenti povezanosti, što je u suprotnosti sa tvrđenjem iz prethodne rečenice. Dakle, nakon sabijanja skupa čvorova u grafu G preostaju samo grane težine manje od M. U svakom podgrafu tj. komponenti povezanosti, koji se sabija u liniji 11, svaki čvor se može povezati sa svakim drugim, putem čiji je kapacitet barem M[3]. Samim tim, *maxmin* čvora d odnosno kapacitet čvora d (koji je manji od M, jer ne postoji put od s do d u grafu G'), ostaće isti u sabijenom grafu G.

Dakle, kada postoji put od s do d u grafu G' tada iz grafa G uklanjamo grane težine manje ili jednake vrednosti M, a kada put u grafu G' ne postoji, tada iz grafa G uklanjamo grane težine veće ili jednake vrednosti M i pritom vršimo sabijanje skupa čvorova grafa. Dakle, u svakoj iteraciji uklanjamo polovinu grana grafa G.

[3]Zbog ovog koraka algoritam nije primenjiv na usmerene grafove jer ne postoji garancija da je u okviru komponente povezanosti svaki čvor dostižan iz svakog. Naravno, algoritam se može modifikovati da korektno radi i za usmerene grafove ako bismo u liniji 10 tražili komponente jake povezanosti grafa. Međutim, tada bi algoritam bio neefikasan jer ne postoji garancija da bi se u koraku sabijanja skupa čvorova broj grana smanjio. To je zato što se neće svaka grana veća od medijane naći u jakoj komponenti povezanosti.

Algoritam 17 Algoritam najšireg puta zasnovan na sabijanju komponenti povezanosti (NPSK)

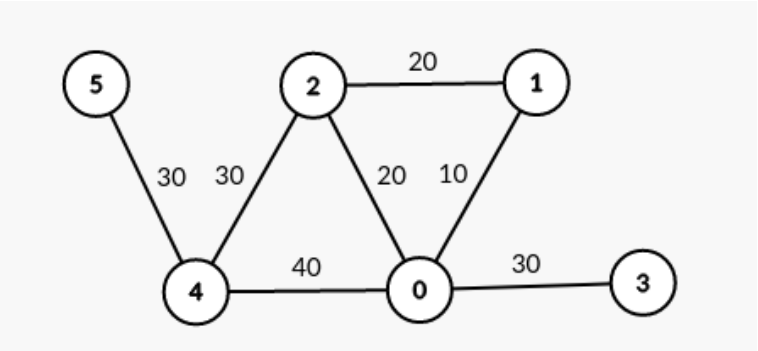
$najsiriPutSabijanjeKomponenti(G, s, d)$
Ulaz: $G = (V, E)$ (težinski neusmereni graf) i njegova dva čvora s i d
Izlaz: najširi put u grafu G od s do d
1: **begin**
2: $maxmin \leftarrow maxminSabijanjeKomponenti(G, s, d)$
3: $najsiriPut \leftarrow nadjiPutUPodgrafu(G, s, d, maxmin)$
4: **return** $najsiriPut$
5: **end**

Složenost. Razmotrimo najpre složenost algoritma `maxminSabijanjeKomponenti`. S obzirom da u svakoj iteraciji `while` petlje brišemo polovinu grana grafa, a telo petlje se izvršava u linearnom vremenu u odnosu na broj grana, ukupno vreme izvršavanja algoritma, posmatrajući u odnosu na broj grana, je $O(|E| + \frac{|E|}{2} + \frac{|E|}{4} + \ldots) = O(|E|)$. Brisanje grana grafa se u C++ implementaciji vrši funkcijom `deleteEdges` koja ima vremensku složenost $O(|E| + |V|)$ i prostornu složenost $O(1)$. U algoritmu `maxminSabijanjeKomponenti` vreme izvršavanja linije 6 u najgorem slučaju je $O(|E| + |V|)$ tj. zavisi linearno od trenutnog broja čvorova (oznake svih čvorova se moraju resetovati na $false$ da bi DFS pratio koji je čvor označen), i može se desiti da linija 6 uvek vraća $true$, tj. put uvek postoji, pa se broj čvorova nikada neće smanjiti (sabijanjem u `else` grani). Jedan od slučaja gde je uslov iz linije 6 Algoritma 16 uvek ispunjen, je kada je vrednost $maxmin$ jednaka maksimalnoj težini grane grafa. U najgorem slučaju u svakoj iteraciji prolazimo kroz $O(|V|)$ čvorova, a broj iteracija je $\log(|E|)$. Stoga je ukupna vremenska složenost algoritma `maxminSabijanjeKomponenti` $O(|E| + |V| \cdot \log(|E|))$. Složenost algoritma NPSK je maksimum složenosti pronalaska puta u grafu, a to je $O(|E| + |V|)$, i vremenske složenosti algoritma `maxminSabijanjeKomponenti`. Dakle, vremenska složenost NPSK je $O(|E| + |V| \cdot \log(|E|))$.

Pošto algoritam `maxminSabijanjeKomponenti` prenosi graf G kao argument funkcije po vrednosti prostorna složenost algoritma je linearna u odnosu na veličinu grafa. Dakle, prostorna složenost algoritma NPSK je $O(|V| + |E|)$.

Primer 3.4.1. Na slici 3.4 dat je primer izvršavanja Algoritma 16. Zadatak je pronaći kapacitet najšireg puta u datom grafu od čvora 0 do čvora 1. Data je tabela koja opisuje vrednosti promenljivih na kraju svake iteracije `while` petlje. Iteracija 0 označava vrednosti promenljivih pre početka izvršavanja `while` petlje, dok iteracija 1 označava vrednosti promenljivih nakon izvršavanja prve iteracije

brojIteracija	M	s	d	E	V	maxmin
0	-	0	1	(0,1,10) (0,2,20) (0,3,30) (0,4,40) (2,1,20) (4,2,30) (4,5,30)	$\{0,1,2,3,4,5\}$	-
1	30	0	1	(0,1,20)	$\{0,1\}$	-
2	20	0	1	-	$\{0,1\}$	20



Slika 3.4: Primer izvršavanja algoritma 16

petlje. Zbog sabijanja skupa čvorova grafa, nazivi čvorova će se menjati tokom izvršavanja algoritma. U ovom primeru, tokom izvršavanja prve iteracije `while` petlje, dobijamo da je medijana skupa težina grana grafa $M = 30$, dok je $G' = (\{0,1,2,3,4,5\}, \{(0,3),(0,4),(4,2),(4,5)\})$, pa kako ne postoji put od čvora 0 do čvora 1 u grafu G', izvršava se korak u liniji 10 Algoritma 16 i dobijaju se sledeće komponente povezanosti: $V_1 = \{0,2,3,4,5\}$ i $V_2 = \{1\}$. Njihovim sabijanjem dobijamo nova dva čvora koje ćemo nazvati 0 i 1, pri čemu čvor 0 odgovara skupu V_1, dok čvor 1 odgovara skupu V_2. S obzirom da se u polaznom grafu startni čvor nalazio u skupu V_1, a ciljni čvor nalazio u skupu V_2 nazivi startnog i ciljnog čvora ostaju 0 i 1. Trenutni nazivi za startni i ciljni čvor opisani su u kolonama s i d. U kolonama V i E prikazani su trenutni skup čvorova i trenutni skup grana grafa G. U koloni *maxmin* se nalazi vrednost kapaciteta najšireg puta koji smo pronašli posle iteracije opisane u koloni $brojIteracija$. U koloni M nalazi se medijana težina skupa čvorova grana grafa nakon date iteracije petlje.

3.5 Algoritam najšireg puta koji sortira težine grana

U ovoj sekciji predstavljen je algoritam koji rešava problem najšireg puta sortiranjem grana grafa prema njihovim težinama. Za početak tvrdimo da umemo da rešimo problem najšireg puta u linearnom vremenu ako su grane grafa sortirane po težini, odnosno ako je data funkcija uređenja grana l. Formalno, uređenje grana je funkcija nad skupom grana E težinskog grafa $G = (V, E)$, $l : E \to \{1, \ldots, m\}$, gde je $m = |E|$ broj grana, koja zadovoljava da iz $l(e_1) > l(e_2)$ sledi da je $c_{e_1} \geq c_{e_2}$. U nastavku teksta dat je algoritam koji pronalazi vrednost *maxmin* za težinski graf ako su grane grafa sortirane po težini. Ulazni argumenti algoritma su težinski graf $G = (V, E)$, njegova dva čvora s i d, funkcija uređenja l, kao i funkcija $T : l(E) \to \mathbb{Z}$ za koju važi $T(l(e)) = c_e$. Izlaz iz algoritma je kapacitet najšireg puta.

Razmotrimo algoritam `maxminGrafaSortiranihGrana`. Neka je $m = |E|$ broj grana grafa. Formiramo m skupova $B_1, \ldots, B_m$ u koje smeštamo čvorove u toku izvršavanja algoritma. Cilj nam je da se svaki čvor nađe u skupu čiji je indeks jednak redosledu kritične grane datog čvora. Strukturu podataka za realizaciju skupova $B_1, \ldots, B_m$ biramo tako da je vremenska složenost dodavanja čvora u skup, kao i biranja i brisanja čvora iz skupa jednaka $O(1)$, što se može realizovati povezanom listom, tako što dodajemo čvor na kraj liste i biramo čvor sa početka liste, dok se efikasno brisanje datog čvora iz liste postiže čuvanjem pokazivača[4]na mesto u listi gde se čvor nalazi. Uz svaki čvor održavaju se dva polja: f koje označava da li je neki čvor već obrađen, i b koje označava u kom se skupu trenutno nalazi čvor. Algoritam određuje kapacitet svakog čvora, tako što čvor većeg kapaciteta smešta u skup sa većim indeksom. Na taj način, smeštanjem čvora u odgovarajući skup se ostvaruje uređenje čvorova po kapacitetu. Dakle, $T(v.b)$ je zapravo trenutni kapacitet čvora v. Na početku algoritma prolazimo kroz susedne čvorove startnog čvora i smeštamo ih u odgovarajuće skupove $B_1, \ldots, B_m$. Naime, ako je neki čvor v susedan čvoru s tada čvor v smeštamo u skup $B_{l((s,v))}$. Obrađujemo redom sve skupove $B_1, \ldots, B_m$ počevši od onog sa najvećim indeksom m. Čvorovi koji odgovaraju granama sa većim težinama završiće u skupu sa većim indeksom i biće prvi obrađeni. Obrađujemo sve čvorove unutar jednog skupa.

[4]U C++ implementaciji svaki od skupova $B_1, \ldots, B_m$ se realizuje strukturom $std :: deque$. Pokazivač na mesto u listi gde se čvor nalazi se realizuje čuvanjem iteratora koji pokazuje na taj čvor. Preciznije, kreiramo dva vektora b i $addr$ čije su veličine jednake broju čvorova grafa. Kada dodamo neki čvor v u skup B_i postavljamo $b[v] = i$ i $addr[v] = B_i.begin()$.

Algoritam 18 Algoritam koji pronalazi vrednost maxmin za težinski graf ako su grane grafa već sortirane po težini

$maxminGrafaSortiranihGrana(G, s, d, l, T)$
Ulaz: $G = (V, E)$ (težinski graf), i njegova dva čvora s i d, l je funkcija koja svakoj grani dodeljuje njen redosled po težini, a T preslikava redosled grane u težinu grane
Izlaz: vrednost najkraće grane na najširem putu od s do d tj. $maxmin$

```
1: begin
2: m ← |E|
3: Inicijalizuj prazne skupove B_1, ..., B_m
4: for v ∈ V do
5:     v.b ← 0
6:     v.f ← 0
7: end for
8: s.f ← 1
9: for sve grane (s, v) do                ▷ prolazak kroz susedne čvorove startnog čvora
10:    B_{l((s,v))} ← B_{l((s,v))} ∪ {v}      ▷ trenutna kritična grana čvora v je (s, v)
11:    v.b ← l((s, v))                        ▷ v se smešta u skup čiji je indeks l((s, v))
12: end for
13: U ← m
14: while U > 0 do
15:    while B_U nije prazan do
16:        Izaberi v iz B_U
17:        B_U ← B_U \ {v}
18:        v.f ← 1
19:        if v = d then
20:            return T(d.b)
21:        else                          ▷ prolazak kroz neoznačene susede čvora v
22:            for sve grane (v, w) za koje je w.f = 0 do
23:                k ← min{v.b, l((v, w))}
24:                if k > w.b then    ▷ ako smo preko v dobili put većeg kapaciteta
25:                    B_{w.b} ← B_{w.b} \ {w}
26:                    B_k ← B_k ∪ {w}                    ▷ premeštamo w u B_k
27:                    w.b ← k          ▷ ažuriramo novi redosled w po kapacitetu
28:                end if
29:            end for
30:        end if
31:    end while
32:    U ← U − 1
33: end while
34: end
```

Biramo proizvoljni čvor v iz trenutnog skupa koji se obrađuje. Prolazimo kroz

sve njegove susede koji već nisu bili obrađeni (čija je oznaka f jednaka 0). Neka je w jedan od suseda čvora v. Kako smo čvor w dostigli preko čvora v, potencijalno mu dodeljujemo kapacitet jednak $T(\min\{v.b, l((v,w))\})$, ukoliko je taj kapacitet veći od kapaciteta $T(w.b)$. Ako smo putem preko čvora v dobili veći kapacitet za čvor w, tada menjamo i njegov redosled po kapacitetu na $\min\{v.b, l((v,w))\}$ i tu vrednost čuvamo u $w.b$. Kada dođemo do skupa koji sadrži ciljni čvor d, algoritam je završen i vraćamo kapacitet čvora d. Korektnost ovog algoritma se dokazuje slično kao korektnost Dijkstrinog algoritma i može se naći u jednom od prethodnih poglavlja (pogledati 3.2), ili se može pogledati u priloženoj literaturi [1].

Prikažimo sada algoritam za traženje najšireg puta od čvora s do čvora d koji koristi prethodno prikazani Algoritam 18 za računanje vrednosti $maxmin$. U opštem slučaju redosled grana po težini nije poznat. Mogli bismo samo sortirati sve grane prema težinama i za tako dobijeni skup grana pozvati prethodni algoritam. Međutim, sortiranje svih grana je složenosti $O(|E| \cdot \log|E|)$. Da bismo ostvarili bolju efikasnost uvodimo promenljive L i R i sortiramo samo grane iz skupa $S = \{e \in E : c_e > L, c_e \leq R\}$. Naime, cilj promenljivih L i R je da se kritična grana čvora d nađe u skupu S. Pošto se kritična grana nalazi u skupu S, grane iz skupa $E \setminus S$ ne treba sortirati. Ako bismo, na primer, svim granama iz skupa $A = \{e \in E : c_e \leq L\}$ promenili težinu na $-\infty$, a svim granama iz skupa $B = \{e \in E : c_e > R\}$ promenili težinu na $+\infty$, tada takva promena težina grana grafa ne bi uticala na pronalazak $maxmin$ čvora d. Neka je p proizvoljni put od čvora s do čvora d. Znamo da je kapacitet puta p u početnom grafu bio manji ili jednak vrednosti $maxmin_d$. Ako je kapacitet puta p u početnom grafu bio jednak vrednosti $maxmin_d$, tada taj put ne sadrži nijednu granu iz skupa A, i ako on sadrži neku granu iz skupa B, povećanje težina grana skupa B ne utiče na kapacitet puta p. Znamo da postoji bar jedan ovakav put i označimo ga sa p_{maxmin}. Naime, najširi put od čvora s do čvora d postoji akko postoji proizvoljni put od čvora s do čvora d. Možemo na početku algoritma NPSG dodati proveru da li je čvor d dostižan iz čvora s, što je i urađeno u implementaciji. U pseudokodu algoritama najšireg puta zbog jednostavnosti pretpostavljamo da postoji put od čvora s do čvora d. Za neki put p čiji je kapacitet manji od vrednosti $maxmin_d$, promena težina grana skupa A može doprineti da kapacitet tog puta postane $-\infty$. Međutim, zbog postojanja puta p_{maxmin}, vrednost $maxmin_d$ u novom grafu sa izmenjenim skupom težina grana, ostaće ista kao u početnom grafu. Takođe, svaki najširi put od čvora s do čvora d u početnom grafu, zadržaće svoj kapacitet nakon opisane promene težina grana iz

skupa $A \cup B$, pa će taj put ostati najširi put od čvora s do čvora d grafa nakon promene težina grana. Pošto ovakva promena težina grana ne utiče na rešenje, možemo izbeći sortiranje grana van skupa S tako što ćemo granama iz skupa A dodeliti najmanji prioritet prilikom kreiranja funkcije uređenja l i težinu $-\infty$ prilikom kreiranja funkcije T, dok granama iz skupa B dodeljujemo najveći prioritet prilikom kreiranja funkcije uređenja l i težinu $+\infty$ prilikom kreiranja funkcije T.

Pojasnimo sada formiranje skupa S. Na početku je skup S jednak E. U svakoj iteraciji `while` petlje tražimo medijanu M trenutnog skupa grana S koji se razmatra. Pitamo da li postoji put od s do d ako ignorišemo grane težine manje ili jednake vrednosti M. Ako traženi put postoji tada važi da je kapacitet čvora d veći od M, pa postavljamo $L = M$ i brišemo grane težine manje ili jednake M iz skupa S. Ako ne postoji traženi put od s do d to znači da je kapacitet čvora d manji ili jednak M, pa iz skupa S brišemo sve grane težine veće od M i postavljamo vrednost promenljive R na M. Broj iteracija `while` petlje zavisi od izabranog broja k. Pošto se u svakom koraku `while` petlje briše polovina grana iz skupa S, broj iteracija petlje je ograničen brojem $\log k$. Nakon izvršavanja petlje, sortiramo neopadajuće po težini grane iz skupa S nekim efikasnim algoritmom sortiranja. Znajući redosled grana po težini lako određujemo funkciju uređenja l i funkciju T. Nakon toga, izvršavanjem Algoritma 18 dobijamo vrednost *maxmin*. Znajući vrednost *maxmin* pronalazimo put u grafu G ignorišući grane težine manje od *maxmin*.

Složenost. Razmotrimo složenost izvršavanja Algoritma 18. Za izvršavanje se koristi m skupova $B_1, \ldots, B_m$ u kojima se mogu naći čvorovi grafa, svaki čvor u najviše jednom skupu. Zbog toga je prostorna složenost algoritma $O(|V| + |E|)$. Svaka grana se obilazi najviše jednom. Održavanje vrednosti oznaka f nam garantuje da se svaki čvor obrađuje najviše jednom. Dakle, vremenska složenost je linearna po veličini grafa tj. iznosi $O(|V| + |E|)$.

Razmotrimo sada složenost NPSG. Grane grafa najveće i najmanje težine mogu se odrediti DFS obilaskom grafa u linearnoj složenosti $O(|V| + |E|)$. Vremenska složenost izvršavanja `while` petlje zavisi od izabranog broja k. `While` petlja se izvršava $\log k$ puta, a telo `while` petlje ima vremensku složenost $O(|V| + |E|)$ pa je ukupna vremenska složenost petlje $O((|V| + |E|) \cdot \log k)$. Vremenska složenost sortiranja grana u liniji 17 takođe zavisi od izabrane vrednosti k. U svakoj iteraciji `while` petlje broj grana koje je potrebno sortirati u liniji 17 se smanjuje za polovinu jer u svakoj iteraciji petlje postavljamo jednu od promenljivih L i R na vrednost M tj. medijanu težina trenutnog skupa grana koji se razmatra.

Algoritam 19 Algoritam najšireg puta koji sortira grane po težini (NPSG)

$najsiriPutSortiraneGrane(G, s, d, k)$
Ulaz: $G = (V, E)$ (težinski graf) i njegova dva čvora s i d, i broj k
Izlaz: najširi put od s do d

1: **begin**
2: $brojIteracija \leftarrow 0$
3: $S \leftarrow E$
4: $L \leftarrow -\infty$
5: $R \leftarrow +\infty$
6: **while** $brojIteracija < \log k$ **do**
7: $M \leftarrow$ medijana skupa $\{c_e : e \in S\}$
8: **if** $nadjiPutUPodgrafu(G, s, d, M + 1) \neq null$ **then**
9: $S \leftarrow \{e \in S : c_e > M\}$
10: $L \leftarrow M$
11: **else**
12: $S \leftarrow \{e \in S : c_e \leq M\}$
13: $R \leftarrow M$
14: **end if**
15: $brojIteracija \leftarrow brojIteracija + 1$
16: **end while**
17: Sortirati neopadajuće po težini sve grane skupa S. Neka je $t = |S|$ i neka je skup S nakon sortiranja jednak $\{e_1, e_2, \ldots, e_t\}$ (Dakle, važi $c_{e_1} \leq c_{e_2} \leq \ldots \leq c_{e_t}$)
18:

$$l(e) = \begin{cases} 1, & e \in E,\ c_e \leq L \\ \text{i+1}, & e \in S,\ e = e_i \\ \text{t + 2}, & e \in E,\ c_e > R \end{cases}$$

19:

$$T(x) = \begin{cases} c_e, & x = l(e), x \neq 1,\ x \neq t + 2 \\ -\infty, & x = 1 \\ +\infty, & x = t + 2 \end{cases}$$

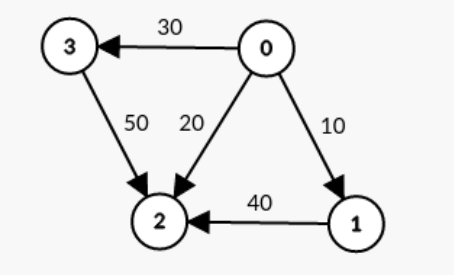
20: $maxmin \leftarrow maxminGrafaSortiranihGrana(G, s, d, l, T)$
21: $najsiriPut \leftarrow nadjiPutUPodgrafu(G, s, d, maxmin)$
22: **return** $najsiriPut$
23: **end**

Ako je ukupan broj grana grafa m, posle izvršavanja `while` petlje preostali broj grana će biti jednak $t = O(\frac{m}{k})$. Ako se sortiranje N elemenata vrši u vremenu $O(N \cdot \log\ N)$ tada je složenost linije 17 jednaka $O(\frac{m}{k} \cdot \log \frac{m}{k})$. Vremenska složenost izvršavanja Algoritma 18 kao i pronalaska puta u grafu je linearna. Ako za k izaberemo vrednost $\log m$, vremenska složenost algoritma NPSG biće jednaka $O(m \cdot \log \log$

m). Preciznije, složenost algoritma je $O((|V|+|E|)\cdot \log\log|E|)$. Prostorna složenost algoritma NPSG je linearna u odnosu na veličinu grafa i iznosi $O(|V|+|E|)$.

Napomena. U C++ implementaciji algoritmi NPSG i NPSK su optimizovani da razmatraju samo grane dostižne iz startnog čvora. Ova promena ne utiče na vremensku složenost u najgorem slučaju ali poboljšava prosečnu vremensku složenost izvršavanja algoritama. Očigledno se najširi put sastoji samo od grana koje su dostižne iz startnog čvora pa nije potrebno razmatrati grane koje nisu dostižne iz starnog čvora.

operacija	B_1	B_2	B_3	B_4	B_5	U
inicijalizacija	{}	{}	{}	{}	{}	5
obradi susede čvora 0	{1}	{2}	{3}	{}	{}	5
B_5 je prazan	{1}	{2}	{3}	{}	{}	5
B_4 je prazan	{1}	{2}	{3}	{}	{}	4
izaberi čvor 3 iz B_3	{1}	{2}	{}	{}	{}	3
obradi susede čvora 3	{1}	{}	{2}	{}	{}	3
izaberi čvor 2 iz B_3	{1}	{}	{}	{}	{}	3
2 je ciljni čvor, vrati $T(3)$ kao rešenje	{1}	{}	{}	{}	{}	3



Slika 3.5: Primer izvršavanja Algoritma 18

Primer 3.5.1. Na slici 3.5 dat je primer izvršavanja Algoritma 18. Zadatak je pronaći kapacitet najšireg puta u datom grafu od čvora 0 do čvora 2.

Funkcije l i T su date sa: $l((0,1)) = 1$, $l((0,2)) = 2$, $l((0,3)) = 3$, $l((1,2)) = 4$, $l((3,2)) = 5$ i $T(1) = 10$, $T(2) = 20$, $T(3) = 30$, $T(4) = 40$, $T(5) = 50$. Tabela prikazuje stanje skupova $B_1, \ldots, B_5$ i promenljive U nakon izvršavanja operacija u odgovarajućoj koloni te vrste. U linijama 9-12 Algoritma 18 obrađujemo susede čvora 0. Nakon toga postavljamo vrednost promenljive U na 5 i u svakoj iteraciji spoljašnje `while` petlje umanjujemo vrednost promenljive U za 1, dok u unutrašnjoj `while` petlji prolazimo kroz čvorove skupa B_U. Pošto su skupovi B_5 i B_4 prazni, najpre obrađujemo skup B_3. Biramo čvor 3 iz skupa B_3. Obrađujemo neoznačene

susede čvora 3. Jedini sused čvora 3 je čvor 2. U liniji 23 Algoritma 18 vrši se ažuriranje vrednosti k na sledeći način: $k = \min\{3, l((3,2))\}$ tj. $k = 3$. Kako se čvor 2 nalazi u skupu B_2 i $k > 2$, premeštamo čvor 2 u skup B_3. Dalje, obrađujemo sledeći čvor u skupu B_3, a to je čvor 2. Pošto je čvor 2 upravo ciljni čvor algoritam se završava i vraćamo $T(3)$ kao rešenje.

Glava 4

Implementacija i evaluacija

4.1 Implementacija

U radu je predstavljeno nekoliko različitih algoritama za rešavanje problema najšireg puta. Svi ovi algoritmi su realizovani u programskom jeziku C++. Programski jezik C++ je jedan od najefikasnijih i najbržih programskih jezika. Krasi ga i skup bogatih biblioteka. Za implementaciju algoritama korišćen je raznovrstan skup struktura podataka, dostupan kroz STL biblioteku. Koristi se i boost biblioteka zbog upotrebe Fibonačijevog hipa. Program sadrži 1480 linija koda. Izvorni kod je dostupan na adresi `https://github.com/githubnemanja/njsrpt`.

4.2 Rezultati testiranja

Algoritmi opisani u glavi 3 testirani su na testnom uzorku grafova. Testiranje je vršeno na računaru sa Intel i5 procesorom i 16 GB DDR4 RAM memorije. Kao testni primeri korišćeni su neusmereni težinski grafovi sa skupom grana generisanim na slučajan način. Grane su generisane funkcijom `generateEdges` koja iz random biblioteke koristi funkciju `std::uniform_int_distribution` za generisanje težine grane, kao i izlaznog i ulaznog čvora grane. Skup grafova nad kojima se vrši testiranje su neusmereni jer je cilj uporediti i ponašanje algoritma NPSK koji je implementiran samo za neusmerene grafove. Testiran je uzorak grafova kod kojih je broj grana reda veličine $O(n)$, $O(n \log n)$ kao i $O(n^2)$, gde je n broj čvorova grafa. Pod veličinom grafa, odnosno ulaza podrazumevaćemo broj čvorova u grafu koji se razmatra. Priloženi su rezultati testiranja ulaza veličine od 2 do 1000. Za sve prikazane algoritme, izvršena su merenja vremena izvršavanja, izražena u milisekundama. Za

svaku od veličina ulaza, merena su vremena izvršavanja za 100 nasumično odabranih grafova koji imaju zadat broj čvorova i grana sa težinama grana u intervalu (INT_MIN, INT_MAX) i pamti se najbolje, prosečno i najgore vreme izvršavanja prikazano redom u kolonama $t_{min}(ms)$, $t_{avg}(ms)$ i $t_{max}(ms)$.

Zbog velike vremenske složenosti izvršavanja NPGS se testira samo za male ulaze. Tabela 4.1 prikazuje vremena izvršavanja algoritma grube sile za rešavanje problema najšireg puta.

Graf		NPGS		
$\|V\|$	$\|E\|$	$t_{min}(ms)$	$t_{avg}(ms)$	$t_{max}(ms)$
2	2	0.001297	0.0025734	0.025591
2	4	0.001761	0.00252254	0.004815
10	10	0.001508	0.00439793	0.013854
10	33	0.010152	0.558311	2.75255
10	100	31.9702	33.9499	39.6748
11	11	0.001458	0.00627149	0.036129
11	38	0.132908	2.5177	16.6321
11	121	297.543	307.607	345.614
12	12	0.00173	0.00548986	0.025805
12	43	0.006947	14.8301	169.696
12	144	3078.44	3109.88	3374.29
13	13	0.002375	0.0142004	0.141082
13	48	3.19032	124.285	874.344
13	169	39514.3	40825.3	42769.9

Tabela 4.1: Prikaz vremena izvršavanja algoritma NPGS

U tabelama 4.2-4.5 prikazani su rezultati izvršavanja algoritama NPD, NPBP, NPSK i NPSG redom. Primetimo da se vremena izvršavanja algoritama za isti ulaz mogu drastično razlikovati za grafove koji nisu potpuni, dok su za potpune grafove vremena t_{min}, t_{avg} i t_{max} približno jednaka. Razlog tome se krije u činjenici da što je broj grana manji veća je verovatnoća da uopšte neće ni postojani nijedna izlazna grana iz startnog čvora pa se svaki algoritam u tom slučaju izvršava brzo [1]. Vremenska složenost izvršavanja algoritama zapravo ne zavisi od ukupnog broja grana

[1]Ovo je degenerisani slučaj. U ovom radu za testiranje su korišćeni nasumično generisani grafovi. Međutim, možda je bolje koristiti nasumično generisane povezane grafove.

grafa, već od broja grana koje su dostižne iz startnog čvora, pa zbog slučajnog generisanja grana taj broj može da varira. Takođe, vremenska složenost izvršavanja algoritma NPBP zavisi od razlike težine grane najveće težine u grafu i grane najmanje težine u grafu, pa pošto se težine grana grafa biraju na slučajan način, vreme izvršavanja ovog algoritma može da varira.

Graf		NPD		
$\|V\|$	$\|E\|$	$t_{min}(ms)$	$t_{avg}(ms)$	$t_{max}(ms)$
2	2	0.002163	0.00392479	0.023358
2	4	0.00318	0.0044244	0.018772
10	10	0.002679	0.0079872	0.02396
10	33	0.00662	0.0142929	0.034112
10	100	0.011349	0.0189804	0.052333
13	13	0.004002	0.016568	0.035927
13	48	0.011349	0.0189804	0.052333
13	169	0.039715	0.0748905	0.093531
100	100	0.002698	0.054555	0.174134
100	664	0.009191	0.151241	0.253138
100	10000	0.038887	0.263593	0.418131
500	500	0.007426	0.313598	0.718656
500	4483	0.098834	1.05689	1.95874
500	250000	0.328125	3.39283	5.60683
1000	1000	0.013925	0.77681	2.25789
1000	9966	0.088952	2.08702	4.41267
1000	1000000	1.29054	11.5178	19.3243

Tabela 4.2: Prikaz vremena izvršavanja algoritma NPD

Graf		NPBP		
$\|V\|$	$\|E\|$	$t_{min}(ms)$	$t_{avg}(ms)$	$t_{max}(ms)$
2	2	0.000516	0.00330284	0.009178
2	4	0.003262	0.00426293	0.004974
10	10	0.000572	0.0238647	0.072123
10	33	0.039084	0.067025	0.109478
10	100	0.049147	0.0847005	0.138061
13	13	0.000935	0.0593021	0.154175

13	48	0.068015	0.163141	0.327844
13	169	0.155803	0.236759	0.363352
100	100	0.00034	0.106989	0.280642
100	664	0.144018	0.685495	1.21483
100	10000	0.805582	3.05395	5.40156
500	500	0.000356	0.48044	1.3719
500	4483	1.55142	5.10164	9.55197
500	250000	26.9653	76.2925	115.242
1000	1000	0.000356	1.03713	3.95585
1000	9966	2.18766	10.4811	20.5875
1000	1000000	154.51	300.648	453.209

Tabela 4.3: Prikaz vremena izvršavanja algoritma NPBP

Graf		NPSK		
$\|V\|$	$\|E\|$	$t_{min}(ms)$	$t_{avg}(ms)$	$t_{max}(ms)$
2	2	0.000525	0.00416374	0.02432
2	4	0.003717	0.00525702	0.00663
10	10	0.00058	0.0163255	0.055166
10	33	0.026001	0.0493043	0.067922
10	100	0.048253	0.0754921	0.127872
13	13	0.000964	0.0457977	0.094994
13	48	0.082148	0.147417	0.2603
13	169	0.132159	0.221845	0.36283
100	100	0.000336	0.106717	0.253103
100	664	0.324069	0.442953	0.579312
100	10000	1.56319	1.80557	2.19561
500	500	0.000354	0.482408	1.12248
500	4483	2.4555	3.16414	4.61895
500	250000	37.7365	41.9707	45.8927
1000	1000	0.000356	1.01942	2.61224
1000	9966	5.31151	7.23448	14.6177
1000	1000000	152.955	168.981	181.428

Tabela 4.4: Prikaz vremena izvršavanja algoritma NPSK

Graf		NPSG		
$\|V\|$	$\|E\|$	$t_{min}(ms)$	$t_{avg}(ms)$	$t_{max}(ms)$
2	2	0.000456	0.00434556	0.020818
2	4	0.00396	0.0054258	0.008213
10	10	0.000579	0.0171699	0.065687
10	33	0.035271	0.062597	0.121381
10	100	0.045991	0.0881974	0.158425
13	13	0.00083	0.0468927	0.170123
13	48	0.073905	0.20789	0.382531
13	169	0.184446	0.300779	0.40704
100	100	0.000338	0.0510447	0.136285
100	664	0.253327	0.348941	0.488075
100	10000	1.18007	1.54564	2.10976
500	500	0.000338	0.223055	0.536479
500	4483	1.81491	2.65208	4.04352
500	250000	29.7384	36.1695	44.992
1000	1000	0.000341	0.47999	1.89413
1000	9966	4.384	5.83233	8.64431
1000	1000000	123.043	145.077	173.861

Tabela 4.5: Prikaz vremena izvršavanja algoritma NPSG

Kao što smo videli, test istance su kreirane tako da se može posmatrati uticaj broja čvorova i broja grana grafa na performanse algoritama. Posebno je važan broj grana u odnosu na broj čvorova grafa odnosno gustina grana grafa koji se testira. Zbog toga u narednom tekstu izdvajamo tri grupe uzoraka i grafički upoređujemo vremena izvršavanja algoritama. Tri grupe uzoraka su:

- grupa 1

  kada je $|E| = O(|V|)$,

- grupa 2

  kada je $|E| = O(|V| \cdot \log |V|)$,

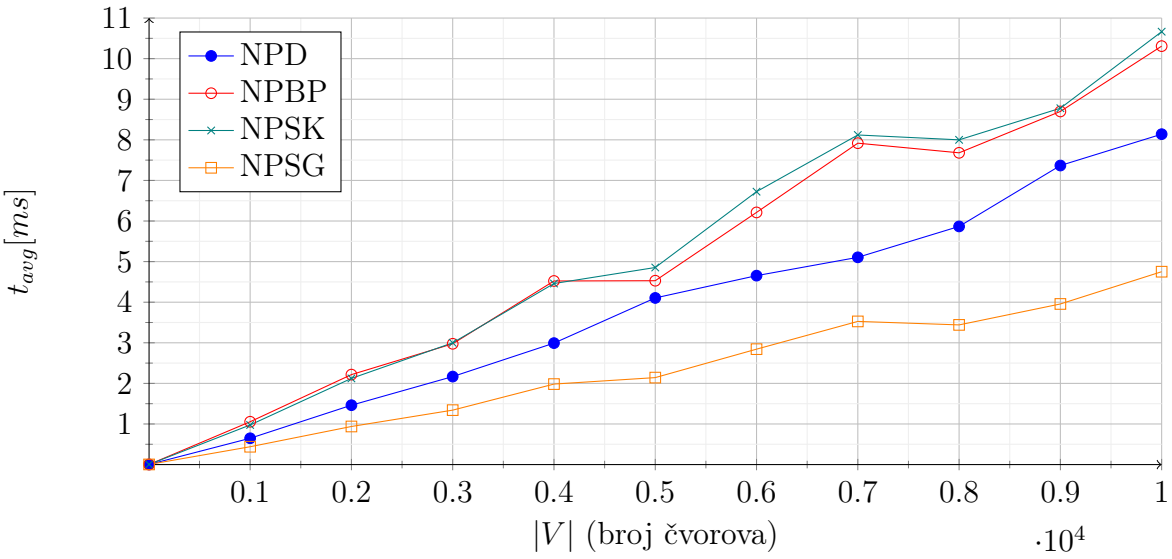
- grupa 3

  kada je $|E| = O(|V|^2)$.

Ako znamo gustinu grana možemo oceniti vremensku složenosti algoritma u odnosu na broj čvorova. Ako ulaz pripada grupi 1 ili grupi 2 uzoraka tada očekujemo približno linearan trend (tačnije očekujemo vremensku složenost $O(|V| \cdot \log |V|)$), dok za grupu 3 uzoraka očekujemo kvadratni trend.

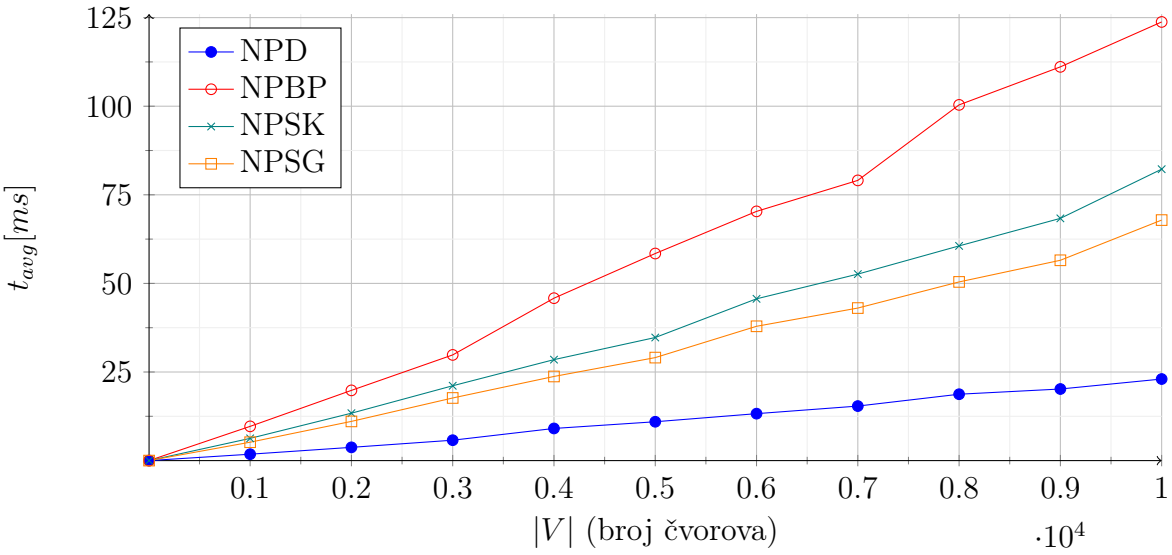
Vreme izvršavanja algoritama na grupi 1 uzorka prikazano je grafički na slici 4.1. X-osa predstavlja veličinu ulaza u intervalu od 2 do 10000. Y-osa predstavlja prosečno vreme izvršavanja algoritama na uzorku od 100 nasumično odabranih grafova (koji imaju X čvorova i pripadaju grupi 1 uzorka). Vreme je izraženo u milisekundama. Zbog jako slabih performansi u odnosu na ostale algoritme, NPGS nećemo grafički upoređivati.

Na slici 4.1 se vidi da se algoritmi NPBP i NPSG ponašaju slično za ulazne instance grupe 1, dok je NPD malo brži u proseku. Najbolji prosečan vremenski učinak ima algoritam NPSG.



Slika 4.1: Prikaz vremena izvršavanja algoritama za grupu 1 uzorka

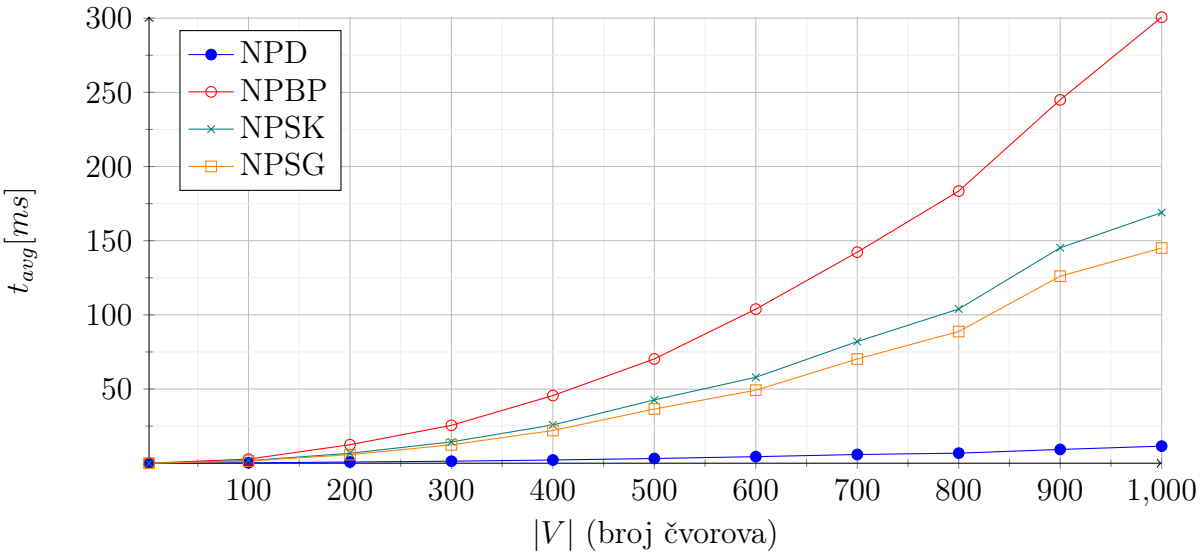
Vreme izvršavanja algoritama na grupi 2 uzorka prikazano je grafički na slici 4.2. X-osa predstavlja veličinu ulaza u intervalu od 2 do 10000. Y-osa predstavlja prosečno vreme izvršavanja algoritama na uzorku od 100 nasumično odabranih grafova. Vreme je izraženo u milisekundama.



Slika 4.2: Prikaz vremena izvršavanja algoritama za grupu 2 uzorka

Na slici 4.2 se vidi rangiranje algoritama po performansama za ulazne instance grupe 2 veličine od 2 do 10000. Najbolje performanse ima NPD, pa NPSK, pa NPSG, dok je NPBP najsporiji u proseku.

Posmatranjem grafika 4.1 i 4.2 deluje da je vremenska zavisnost izvršavanja algoritama u odnosu na veličinu ulaza približno linearna, što se uklapa u teorijski okvir.



Slika 4.3: Prikaz vremena izvršavanja algoritama za grupu 3 uzorka

Vreme izvršavanja algoritama na grupi 3 uzorka prikazano je grafički na slici 4.3.

Zbog kvadratne zavisnosti broja grana od broja čvorova u ovom slučaju testiraćemo veličinu ulaza od 2 do 1000. Za veće dimenzije ulaza javlja se memorijski problem[2]. Vremena izvršavanja algoritama i dalje prikazujemo u milisekundama, dok algoritme testiramo na uzorku od 100 nasumično odabranih grafova. U ovom slučaju uočavamo isto rangiranje po performansama kao na slici 4.2, s tim što NPD ima značajno bolje performanse od algoritama NPBP, NPSK i NPSG. Primetimo da kada je ulaz duplo veći, vreme izvršavanja je otprilike četiri puta veće. Dakle, zavisnost je kvadratna i odgovara našem očekivanju na osnovu teorijskog razmatranja složenosti.

[2]U radu je za obilazak grafa korišćen rekurzivan DFS. Moguće je da memorijski problem nastaje zbog prekoračenja steka. Međutim, moguće je i da sama veličina grafa kao i promenljivih koji se koriste u implementaciji algoritama pravi memorijski problem za velike ulaze. Naime, memorijski problem za grupu 3 ulaza se javlja za $|V| = 5000$ i $|E| = 25000000$. Dubina rekurzije zavisi od broja čvorova. Kako algoritmi savršeno rade za grupu ulaza 1 i 2 kada je $|V| = 5000$ zaključujemo da memorijski problem najverovatnije nastaje zbog same veličine grafa, a ne zbog rekurzije.

Glava 5

Zaključak

Predmet istraživanja ovog rada je grafovski problem najšireg puta. Kao i drugi algoritmi koji pronalaze optimalan put u grafu i algoritmi najšireg puta imaju veliki praktičan značaj u mnogim oblastima. U radu je predloženo pet algoritama za rešavanje problema pronalaska najšireg puta u grafu: algoritam grube sile, algoritam najšireg puta zasnovan na Dijkstrinoj ideji, algoritam najšireg puta koji koristi binarnu pretragu po dužini kritične grane ciljnog čvora, algoritam najšireg puta zasnovan na sabijanju komponenti povezanosti i algoritam najšireg puta koji sortira težine grana. Analiza performansi algoritama je pokazala da je predloženi algoritam najšireg puta zasnovan na Dijkstrinoj ideji uspešniji od preostala četiri algoritma u smislu brzine izvršavanja. Algoritam grube sile je neupotrebljiv za veće instance problema jer je prilično spor, dok je brzina izvršavanja ostalih algoritama zadovoljavajuća. Veće razlike između algoritama se vide za gušće grafove. Buduća istraživanja bi se mogla koncentrisati na dalje poboljšanje efikasnosti algoritama NPSK i NPSG, i implementiranje algoritma koji je efikasniji od algoritma NPD.

Bibliografija

[1] E. W. Dijkstra, A note on two problems in connexion with graphs, 1959.

[2] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and Clifford Stein. Introduction to Algorithms, Second Edition. MIT Press and McGraw-Hill, 2001. ISBN 0-262-03293-7. Section 22.3: Depth-first search, pp. 540–549, 2001.

[3] D. Cvetković. Teorija grafova i njene primene. Beograd: Naučna knjiga, 1990.

[4] M. Blum, R. W. Floyd, V. Pratt, R. L. Rivest, and R. E. Tarjan, Linear time bounds for median computations., in Proc. 4th ann. ACM Symp. Theory Comput., , pp. 119–124, Denver, 1972.

[5] M. Blum, R. W. Floyd, V. Pratt, R. L. Rivest, and R. E. Tarjan, Time bounds for selection., J. Comput. Syst. Sci. 7, pp. 448–461, 1973.

[6] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, Introduction to algorithms. 2nd ed., Cambridge, MA: MIT Press., 2001.

[7] M. Schulze, The Schulze Method of Voting, 1997.

[8] E. Fernandez, R. Garfinkel, R. Arbiol, Mosaicking of Aerial Photographic Maps Via Seams Defined by Bottleneck Shortest Paths. Operations Research 46(3):293-304, 1998.

[9] V. Kaibel, M. A. F. Peinhardt, On the bottleneck shortest path problem, ZIB-Report 06-22, Konrad-Zuse-Zentrum für Informationstechnik Berlin, 2006.

[10] M. Živković, Algoritmi, Matematički fakultet, 2000.

[11] T. Blechmann, Class template fibonacci_heap, www.boost.org, 2011.

[12] V. Strehl, Binomial Identities–Combinatorial and Algorithmic Aspects. Trends in Discrete Mathematics, 1994.