NAPREDNE BAZE PODATAKA

PROJEKTNI ZADATAK

Primjena algoritama za pretraživanje grafova i pronalazak puta na grafovskoj bazi podataka u Neo4j

TIM THETA

Petra Škrabo

Ivan Štruklec

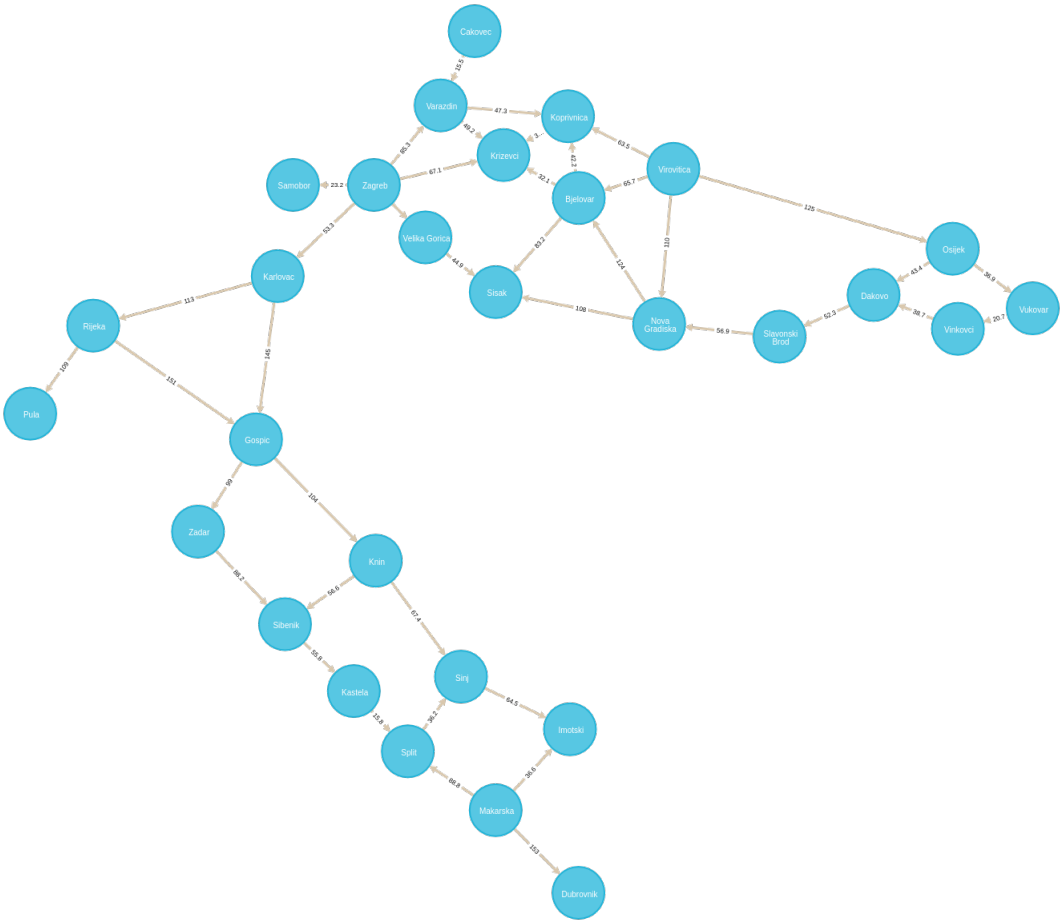
Katarina Šupe

Mateja Terzanović

Margarita Tolja

Sadržaj

1 Uvod



Slika 1: Grafovska baza odabranih hrvatskih gradova

2 Najkraći put

Algoritmi traženja najkraćeg puta u težinskom grafu pronalaze put koji spaja dva promatrana vrha s najmanjim mogućim zbrojem težina bridova na tom putu. Glavni algoritam za rješavanje problema ove vrste je *Dijkstrin algoritam* koji je ujedno implementiran i u standardnoj *shortestPath* funkciji za pronalazak najkraćeg puta Neo4j *Graph Algorithms* biblioteke. Postoje i dvije tehnike ubrzanja Dijkstrinog algoritma, *dvosmjerni Dijkstrin algoritam* i *algoritam* A^*, no mi ćemo se bazirati na općenitu verziju.

2.1 Dijkstrin algoritam

Osnovna ideja algoritma je iterativno računanje najkraćeg puta krenuvši od početnog čvora tako da se svakom iteracijom ažurira udaljenost puta do trenutno promatranog čvora sve dok ne obiđe sve čvorove.

Za bolje razumijevanje prvo navodimo pseudokod algoritma, a zatim njegov detaljniji opis.

Algoritam 1 Dijkstrin algoritam

```
1: za svaki vrh v u grafu ponavljaj
2:     udaljenost[v] := ∞
3:     posjećen[v] := nedefinirano
4: kraj za svaki

5: udaljenost[početni_vrh] := 0
6: Q := skup svih vrhova u grafu

7: sve dok Q nije prazan skup ponavljaj
8:     u := vrh v ∈ Q sa najmanjom udaljenosti udaljenost[v]
9:     ukloni u iz Q
10:    za svaki susjed v od u ponavljaj
11:        putanja := udaljenost[u] + udaljenost_između(u, v)
12:        ako putanja < udaljenost[v] onda
13:            udaljenost[v] := putanja
14:            posjećen[v] := u
15:        kraj ako
16:    kraj za svaki
17: kraj sve dok

18: vrati udaljenost[ ], posjećen[ ]
```

Udaljenost[v] čuva najkraću udaljenost vrha v od početnog vrha i predstavlja akumulirane težine između ta dva vrha, a *posjećen*[v] čuva informaciju o posjećenosti vrha v.

Na početku se svaki vrh v označava kao neposjećen, a njegova *udaljenost*[v] postavlja na beskonačnu vrijednost. Iznimno, udaljenost ishodišnog vrha postavlja se na nulu. U skup Q stavljaju se svi vrhovi grafa, budući da još niti jedan nije obrađen.

Sljedeći korak je pretražiti cijeli graf i za svaki vrh iz grafa odrediti njegovu udaljenost od početnog vrha tako da ona bude najmanja moguća. Prvo se određuje onaj vrh u čija je *udaljenost*[u] najmanja. Vidimo da će se u prvom koraku obraditi početni vrh budući da je njegova udaljenost nula, dok su sve druge udaljenosti postavljene na beskonačnu vrijednost. Nakon odabira vrha on se briše iz skupa Q. Algoritam ga zatim obrađuje na način da se *udaljenost*[v] svakog njegovog susjeda v ažurira ukoliko se on još uvijek nalazi u skupu Q i ako je dobivena *udaljenost*[v] preko promatranog čvora manja nego ona koju je v imao do tada. Ažuriranje se radi određivanjem zbroja

udaljenosti između susjednog vrha i vrijednosti trenutno promatranog vrha. Ako se susjedni vrh ne nalazi u Q to znači da ga je algoritam već obradio, odnosno da je njegova udaljenost najmanja moguća.

Kad se svi susjedi promatranog vrha obrade on se označava kao posjećen i algoritam nastavlja s obradom sljedećeg vrha, onog s trenutno najmanjom udaljenosti od početnog vrha.

Nakon završetka algoritma *posjećen* će imati strukturu koja zapravo podgraf orginalnog grafa i predstavlja najkraći put između početnog vrha i svih drugih vrhova.

Važno je napomenuti da Dijkstrin algoritam rješava problem najkraćeg puta u grafu s pozitivnim težinama bridova.

2.2 Primjena Dijkstrinog algoritma na promatranu bazu

U našem slučaju su težinama reprezentirane udaljenosti pa nam pozitivnost neće predstavljati problem. Pogledamo li napravljenu grafovsku bazu primjećujemo grad Zagreb kao veliko prometno čvorište. Uzmimo njega kao početni čvor i provedimo Dijkstrin algoritam. Znamo da ćemo kao rezultat dobiti najkraći put od Zagreba do svakog grada iz promatrane baze.

Prvo ćemo provesti algoritam „ručno".

Inicijalna udaljenost Zagreba je nula, dok su udaljenosti svih ostalih gradova promatrane baze postavljene na beskonačnu vrijednost. Dakle, prvo se obrađuje vrh Zagreb. Njegovi susjedni vrhovi su Velika Gorica (18.7), Samobor (23.2), Karlovac (53.3), Križevci (67.1) i Varaždin (85.3).

Vrh s najmanjom udaljenošću je Velika Gorica pa je upravo ona sljedeći vrh koji ćemo promatrati. Zagreb označavamo kao posjećen. Jedini susjed od Velike Gorice je Sisak čija se udaljenost od Zagreba postavlja na 63.6. Velika Gorica se označava kao posjećeni vrh.

Sljedeći vrh najbliži Zagrebu je Samobor (23.2). On nema susjeda, pa u ovom koraku samo Samobor svrstavamo među posjećene vrhove. Karlovac (53.3) trenutno je vrh s najmanjom udaljenošću pa promatramo njegove susjede Rijeku (166.3) i Gospić (198.3), a Karlovac označavamo ga kao posjećen.

U sljedećoj iteraciji algoritam za promatrani vrh uzima Sisak (63.6), a njegove susjede označava vrijednostima Bjelovar (146.8), Nova Gradiška (171.6). Sisak označava kao posjećen.

Početnom vrhu sada su najbliži Križevci (67.1). Njegovi susjedni vrhovi su Bjelovar (99.2), Koprivnica (99.3) i Varaždin (116.3). U ovoj iteraciji algoritam Varaždinu ne ažurira udaljenost budući da je u Varaždin bliže stići iz Zagreba za 85.3 kilometara, nego preko Križevaca za 116.3 kilometara. Križevci su sada posjećeni.

Na isti način nastavljamo provoditi algoritam sve dok svi gradovi nisu označeni kao posjećeni. Tada algoritam završava, a najkraći put iz Zagreba do svakog pojedinog grada iščitavamo iz koraka algoritma. Detaljna provedba algoritma prikazana je u —— tablici.

Tablica 1: Koraci Dijkstrinog algoritma sa Zagrebom kao početnim čvorom

Trenutni vrh	**Susjedni (neposjećen)**	**Prethodno posjećen**
Zagreb (0)	Velika Gorica (18.7), Samobor (23.2), Karlovac (53.3), Križevci (67.1), Varaždin (85.3)	X
Velika Gorica (18.7)	Sisak (63.6)	Zagreb
Samobor (23.2)	X	Velika Gorica
Karlovac (53.3)	Rijeka (166.3), Gospić (198.3)	Samobor
Sisak (63.6)	Bjelovar (146.8), Nova Gradiška (171.6)	Karlovac
Križevci (67.1)	Bjelovar (99.2), Koprivnica (99.3), Varaždin (116.3)	Sisak
Varaždin (85.3)	Čakovec (100.8), Koprivnica (132.6)	Križevaci
Bjelovar (99.2)	Nova Gradiška (223.2), Virovitica (164.9), Koprivnica (141.4)	Varaždin
Koprivnica (99.3)	Virovitica (162.8)	Bjelovar
Čakovec (100.8)	X	Koprivnica
Virovitica (162.8)	Nova Gradiška (272.8), Osijek (287.8)	Čakovec
Rijeka (166.3)	Pula (275.3), Gospić (317.3)	Virovitica
Nova Gradiška (171.6)	Slavonski Brod (228.5)	Rijeka
Gospić (198.3)	Zadar (297.3), Knin (302.3)	Nova Gradiška
Slavonski Brod (228.5)	Đakovo (280.8)	Gospić
Pula (275.3)	X	Slavonski Brod
Đakovo (280.8)	Osijek (324.2), Vinkovci (319.5)	Pula
Osijek (287.8)	Vukovar (324.7)	Đakovo
Zadar (297.3)	Šibenik (385.5)	Osijek
Knin (302.3)	Šibenik (358.9), Sinj (369.7)	Zadar
Vinkovci (319.5)	Vukovar (340.2)	Knin
Vukovar (324.7)	X	Vinkovci
Šibenik (358.9)	Kaštela (414.7)	Vukovar
Sinj (369.7)	Imotski (434.2), Split (405.9)	Šibenik
Split (405.9)	Kaštela (421.7), Makarska (494.7)	Sinj
Kaštela (414.7)	X	Split
Imotski (434.2)	Makarska (470.8)	Kaštela
Makarska (470.8)	Dubrovnik (623.8)	Imotski
Dubrovnik (623.8)	X	Makarska
X	X	Dubrovnik

Iz tablice sada lako iščitavamo najkraći put od Zagreba do bilo kojeg grada iz naše mreže gradova čitajući korake unatrag. Bitno je zanemariti gradove označene sivom bojom jer njihovu udaljenost dobivenu tom iteracijom ne ažuriramo, budući da već postoji kraći put od početnog čvora do njega. Osim toga, u stupcu **trenutni vrh** pokraj svakog grada nalazi se i njegova točna udaljenost od Zagreba. U stupcu **prethodno posjećen** možemo iščitati sve vrhove posjećene prije trenutnog tako da krenemo od vrha stupca i čitamo vrhove sve do retka koji u prvom stupcu sadrži trenutno posjećen vrh. Po završetku algoritma stupac **prethodno posjećen** sadrži sve gradove iz promatrane baze.

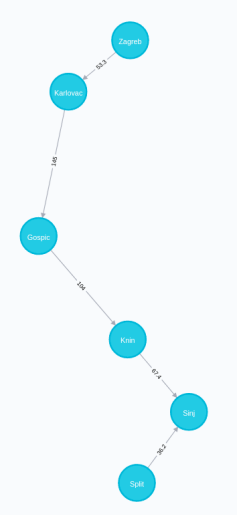
Primjeri najkraćih puteva od Zagreba do Splita, Rijeke i Osijeka. Čitamo:

1. Split←Sinj←Knin←Gospić←Karlovac←Zagreb
2. Rijeka←Karlovac←Zagreb
3. Osijek←Virovitica←Koprivnica←Križevci←Zagreb

Ovaj rezultat potvrđuje i *Cypher* pozivanjem *shortestPath* funkcije iz biblioteke s graf algoritmima.

Tablica 2: Tablica udaljenosti najkraćeg puta od Zagreba do Splita

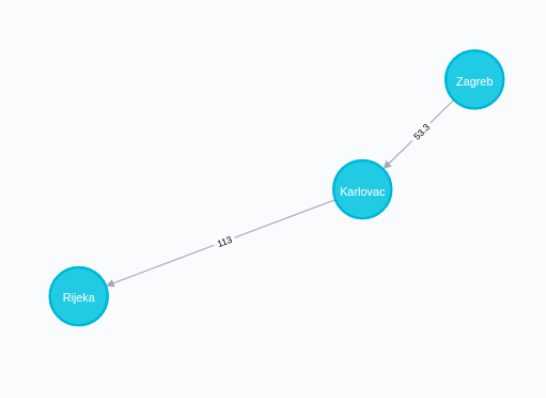
name	**cost**
"Zagreb"	0.0
"Karlovac"	53.3
"Gospic"	198.3
"Knin"	302.3
"Sinj"	369.70000000000005
"Split"	405.90000000000003



Slika 2: Najkraći put od Zagreba do Splita

Tablica 3: Tablica udaljenosti najkraćeg puta od Zagreba do Rijeke

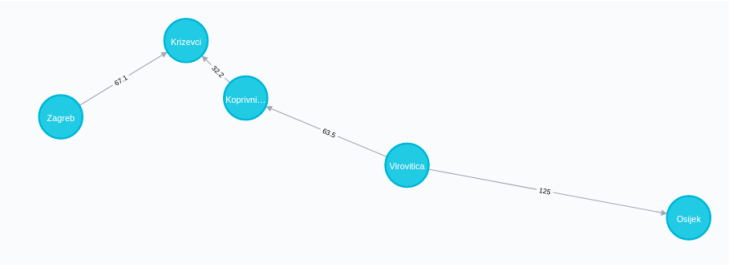
name	**cost**
"Zagreb"	0.0
"Karlovac"	53.3
"Rijeka"	166.3



Slika 3: Najkraći put od Zagreba do Rijeke

Tablica 4: Tablica udaljenosti najkraćeg puta od Zagreba do Osijeka

name	**cost**
"Zagreb"	0.0
"Krizevci"	67.1
"Koprivnica"	99.3
"Virovitica"	162.8
"Osijek"	287.8



Slika 4: Najkraći put od Zagreba do Osijeka

Do sada smo promatrali izvornu verziju algoritma, ali ona se lako može modificirati tako da algoritam pronalazi najkraći put između dvaju zadanih vrhova zaustavljajući se nakon što je najkraći put iz početnog do krajnjeg vrha određen. Dakle, proces ažuriranja susjednih vrhova u prethodno opisanom algoritmu se zaustavlja u trenutku kad se krajnji vrh označi kao posjećen.

2.3 Računanje udaljenosti između dvaju gradova Dijkstrinim algoritmom

Za primjer uzmimo pronalazak najkraćeg puta iz početnog vrha Zadra i krajnjeg vrha Bjelovara. Iteracije kojima ovaj algoritam dolazi do rješenja prikazane su u sljedećoj tablici, a postupak je jednak prethodno opisanom izuzev provjere je li vrh koji je trenutno posjećen jednak krajnjem vrhu.

Tablica 5: Koraci Dijkstrinog algoritma za put od Zadra do Bjelovara

Trenutni čvor	**Susjedni (neposjećen)**	**Prethodno posjećen**
Zadar (0)	Šibenik (88.2), Gospić (99)	X
Šibenik (88.2)	Kaštel (144), Knin (144.8)	Zadar
Gospić (99)	Knin (203), Karlovac (244), Rijeka (250)	Šibenik
Kaštel (144)	Split (159.8)	Gospić
Knin (144.8)	Sinj (212.2)	Kaštel
Split (159.8)	Sinj (196), Makarska (248,6)	Knin
Sinj (196)	Imotski (260.5)	Split
Karlovac (244)	Zagreb (297,3), Rijeka (357)	Sinj
Makarska (248.6)	Imotski (285,2), Dubrovnik (401.6)	Karlovac
Rijeka (250)	Pula (539)	Makarska
Imotski (260)	X	Rijeka
Zagreb (297.3)	Velika Gorica (316), Samobor (320.5), Križevci (364.4), Varaždin (382.6)	Imotski
Velika Gorica (316)	Sisak (360.9)	Zagreb
Samobor (320.5)	X	Velika Gorica
Pula (359)	X	Samobor
Križevci (364.4)	Bjelovar (396.5), Koprivnica (396.6), Varaždin (413.6)	Pula
Sisak (360.9)	Bjelovar (444.1), Nova Gradiška (468.9)	Križevaci
Varaždin (382.6)	Čakovec (398.1), Koprivnica(429,9)	Sisak
Bjelovar (396.5)	Koprivnica (438,7), Virovitica (462.2), Nova Gradiška (520)	Varaždin

Iz tablice čitamo da je najkraći put od Zadra do Bjelovara dan sljedećom rutom: Bjelovar ← Križevci ← Zagreb ← Karlovac ← Gospić ← Zadar. Ruta je duga 396.5 kilometara.

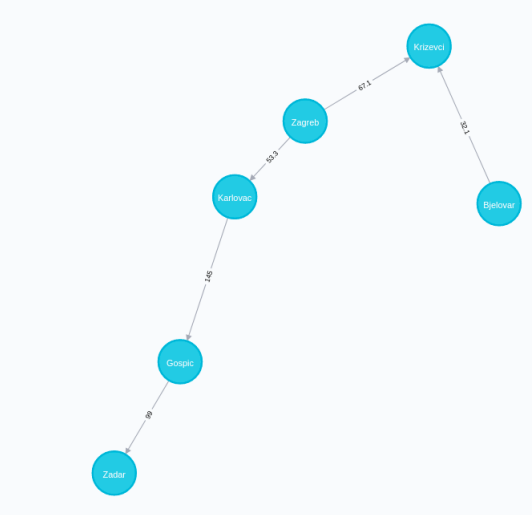
Sljedeći upit u *Cypheru* potvrđuje dobiveni rezultat.

```
MATCH (start:Cityname:"Zadar"), (end:Cityname:"Bjelovar")
CALL algo.shortestPath.stream(start, end, "distance")
YIELD nodeId, cost
```

```
MATCH (other:City) WHERE id(other) = nodeId
RETURN other.name AS name, cost
```

Tablica 6: Tablica udaljenosti najkraćeg puta od Zadra do Bjelovara

name	cost
"Zadar"	0.0
"Gospic"	99.0
"Karlovac"	244.0
"Zagreb"	297.3
"Krizevci"	364.4
"Bjelovar"	396.5



Slika 5: Najkraći put od Zadra do Bjelovara

2.4 Upotreba Dijkstrinog algoritma

Klasični primjer upotrebe ovog algoritma je računanje najbliže rute između dva udaljena mjesta. U našoj grafovskoj bazi Dijkstrin algoritam primjenjujemo u istu svrhu. Ali nismo jedini. I u većim bazama koje sadrže gradove kao vrhove, a na bridovima im leži udaljenost također se koristi ovaj algoritam u izračunima. Tako primjerice *Google Maps* koriste ovaj algoritam. On je doduše modificiran kako bi ga se ubrzalo, ali ideja ostaje ista. Još jedan primjer je i hrvatska softverska tvrtka *Mireo*, poznata po svojim sustavima GPS navigacije koja se isto tako služi ubrzanim Dijkstrinim algoritmom. Kako u cestovnom prometu, algoritam se može koristiti i u zračnom prometu. Težine bridova tada su zračne udaljenosti, a vrhovi zračne luke.

Bitnu primjenu algoritam pronalazi i u sustavu telekomunikacijskih mreža.

Prethodno opisane primjene su poprilično očite. No, algoritam rješava i brojne druge probleme. Tako se primjerice koristi u društvenim mrežama. Većina mreža nudi opciju „prijatelja koje možda poznajemo". Do tog rezultata dolaze primjenom Dijkstrinog algoritma na način da su vrhovi osobe, a težine bridova najčešće zajednički prijatelji.

3 Najkraći put među svim parovima

4 Minimalno razapinjuće stablo

Definirajmo najprije neke važne pojmove koji će nam pomoći u razumijevanju.

Definicija 1. *Podgraf* grafa $G = (V, E)$ je graf kojem su skup vrhova i skup bridova podskupovi od V i E, redom. *Inducirani podgraf* grafa G induciran skupom V' je podgraf $G' = (V', E')$, gdje se E' sastoji od svih bridova od G čija oba kraja leže u V', dok je V' neki zadani podskup od V. Razapinjući podgraf je podgraf oblika $G' = (V', E')$.

Definicija 2. *Šetnja* u grafu je niz $(v_0, e_1, v_1, e_2, v_2, \ldots, e_n, v_n)$, gdje je $e_i = \{v_{i-1}, v_i\}$, za $i = 1, \ldots, n$. Kažemo da je to šetnja od v_0 do v_n. Kažemo da je šetnja zatvorena ukoliko je $v_n = v_0$. *Staza* je šetnja u kojoj su svi bridovi različiti, dok je *put* šetnja u kojoj su svi vrhovi različiti (osim eventualno prvog i zadnjeg). Zatvoreni put zovemo *ciklus.*

Propozicija 3. *Za dva različita vrha x, y grafa G uvjeti da postoji šetnja, staza ili put između x i y su ekvivalentni.*

Definicija 4. Definiramo relaciju $\equiv$ na skupu vrhova V grafa G:
$x \equiv y$ ako postoji put (ili staza ili šetnja) od x do y. $\equiv$ je relacija ekvivalencije koja definira jednu particiju skupa V, te definiramo komponente povezanosti grafa kao podgrafove inducirane klasama ekvivalencije. Kažemo da je graf *povezan*, ukoliko postoji samo jedna komponenta .

Definicija 5. *Stablo* je povezan graf bez ciklusa. Šuma je graf bez ciklusa čije su komponente stabla.

Definicija 6. Neka je G graf. *Razapinjuća šuma* od G je razapinjući podgraf of G koji je šuma. *Razapinjuće stablo* od G je razapinjući podgraf od G koji je stablo.

Korolar 7. *Svaki povezani graf ima razapinjuće stablo.*

Sada kad smo se upoznali sa svim važnim definicijama i tvrdnjama, možemo konstruirati razapinjuće stablo grafa $G = (V, E)$:

Algoritam 2 Razapinjuće stablo

1: $S = \emptyset$
2: **sve dok** je graf (V, S) nepovezan **ponavljaj**
3: nađi brid e koji povezuje vrhove u različitim komponentama
4: dodaj e u S
5: **kraj sve dok**
6: **vrati** (V, S)

Međutim, što ako želimo naći povezani razapinjući podgraf s minimalnom težinom (tj. sumom težina bridova podgrafa)? Takav podgraf mora biti stablo, inače bismo mogli izbrisati neki brid, smanjujući težinu, a ne pokvarivši povezanost.

Definicija 8. *Minimalno razapinjuće stablo* (MST) je stablo koje povezuje sve vrhove nekog (težinskog) grafa, pri čemu je ukupna suma težina svih bridova minimalna.

Najpoznatiji algoritmi za pronalaženje MST-a su *Kruskalov* i *Primov*.

4.1 Kruskalov algoritam

Neka je $G = (V, E)$ povezan graf, ω nenegativna težinska funkcija na E. Tada Kruskalov algoritam možemo ovako zapisati:

Algoritam 3 Kruskalov algoritam

1: $S = \emptyset$
2: **sve dok** (V, S) nije povezan **ponavljaj**
3: odaberi brid $e \in E \setminus S$ minimalne težine, takav da $S \cup \{e\}$ nema ciklus
4: $S = S \cup \{e\}$
5: **kraj sve dok**
6: **vrati** (V, S)

Dakle, u svakom koraku odabiremo brid najmanje težine, takav da njegovo ubacivanje ne stvara ciklus. Mana ovog algoritma je to što nije baš jednostavno pronaći brid minimalne težine koji spaja vrhove iz različitih komponenti. Stoga postoji izmijenjen Kruskalov algoritam, gdje u svakom koraku biramo brid najmanje težine koji spaja neki vrh s nekim vrhom koji još nije spojen. Taj algoritam se naziva *Primov algoritam*.

4.2 Primov algoritam

Neka je $G = (V, E)$, $|V| = n$, povezan graf i ω nenegativna težinska funkcija na E. Primov algoritam možemo ovako zapisati:

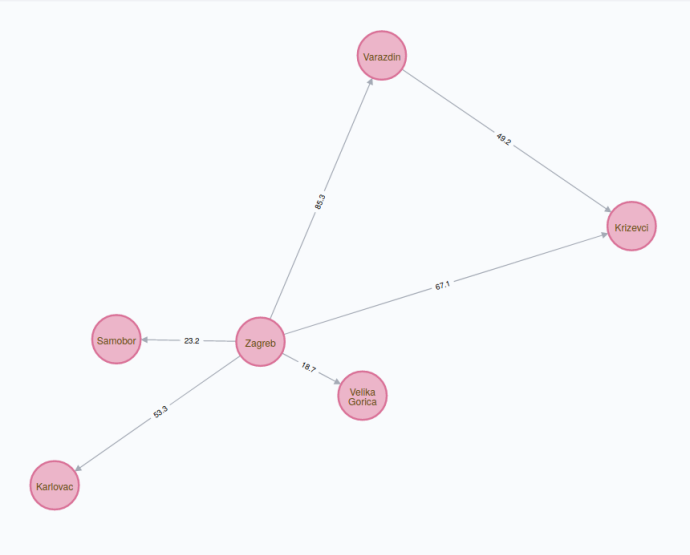
Algoritam 4 Primov algoritam

1: Odaberemo $v_0 \in V$ i definiramo $T = \{v_0\}$, $S = V \setminus \{v_0\}$, $F = \emptyset$
2: **sve dok** $|F| < n - 1$ **ponavljaj**
3: odaberi brid $e = \{v, w\}$ iz E minimalne težine, takav da je v iz T, w iz S
4: $T = T \cup \{w\}$, $F = F \cup \{e\}$, $S = S \setminus \{w\}$
5: **kraj sve dok**
6: **vrati** (V, F)

Kako je Primov algoritam poboljšani Kruskalov algoritam, koristeći njega ćemo pokazati na našoj bazi gradova kako pronaći minimalno razapinjuće stablo. Naglasimo da minimalno stablo ne mora biti jedinstveno, ali težina svakog minimalnog stabla uistinu je minimalna, tj. jedinstvena. Ovaj algoritam zahtjeva odabir proizvoljnog vrha za početak, a s obzirom na to da je Zagreb glavni grad Hrvatske, odaberimo upravo njega. Ovaj algoritam ćemo provesti u koracima *while* petlje iznad, stoga najprije uzimamo $v_0 =$ Zagreb, pa je $T = \{\text{Zagreb}\}$ te je $S = V \setminus \{v_0\}$, a $F = \emptyset$, gdje je V skup svih vrhova našeg grafa, tj. 29 gradova ($n = 29$). Prateći gornji algoritam, pogledajmo s kojim gradovima je grad Zagreb povezan.

```
MATCH (a name:'Zagreb')-[:CONNECTION]-(b) RETURN a,b;
```

Slika 6: Zagreb i njegovi susjedi

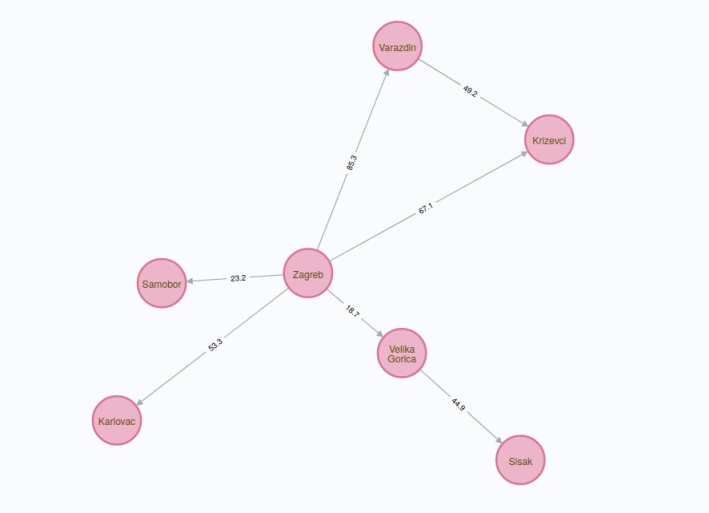


Sada odabiremo brid minimalne težine koji je povezan sa Zagrebom, stoga biramo brid koji povezuje Zagreb i Veliku Goricu (težine 18.7). Pogledajmo sada stanje naših skupova:

T	Zagreb, Velika Gorica
F	{{ Zagreb, Velika Gorica }}

Algoritam provodimo sve dok je $|F| < n - 1 = 28$, stoga nastavljamo. Sada promatramo sve susjede od Zagreba i Velike Gorice i među neodabranim bridovima biramo onaj minimalne težine.

Slika 7: Zagreb, Velika Gorica i njihovi susjedi



Vidimo da ćemo idući brid biti onaj koji povezuje Zagreb i Samobor čija je težina 23.2. Redom dalje ćemo odabirati bridove minimalne težine kao što je prikazano na tablici niže. Svaki odabrani brid ima svoju težinu ω. Dobivenom MST-u računamo ukupnu težinu tako što zbrajamo težine svih bridova u tom stablu.

$$\sum_{v \in V} \omega(v) = 1684 \tag{1}$$

T = posjećeni vrhovi	**F = posjećeni bridovi**	**S = neposjećeni vrhovi**
$v_0 = \{\text{Zagreb}\}$	$\emptyset$	$S = V \setminus v_0 = V \setminus \{\text{Zagreb}\}$
$T = T \cup \{\text{Velika Gorica}\}$	$F = F \cup \{\text{Zagreb, Velika Gorica}\}$	$S = S \setminus \{\text{Velika Gorica}\}$
$T = T \cup \{\text{Samobor}\}$	$F = F \cup \{\text{Zagreb, Samobor}\}$	$S = S \setminus \{\text{Samobor}\}$
$T = T \cup \{\text{Sisak}\}$	$F = F \cup \{\text{Velika Gorica, Sisak}\}$	$S = S \setminus \{\text{Sisak}\}$
$T = T \cup \{\text{Karlovac}\}$	$F = F \cup \{\text{Zagreb, Karlovac}\}$	$S = S \setminus \{\text{Karlovac}\}$
$T = T \cup \{\text{Križevci}\}$	$F = F \cup \{\text{Zagreb, Križevci}\}$	$S = S \setminus \{\text{Križevci}\}$
$T = T \cup \{\text{Bjelovar}\}$	$F = F \cup \{\text{Križevci, Bjelovar}\}$	$S = S \setminus \{\text{Bjelovar}\}$
$T = T \cup \{\text{Koprivnica}\}$	$F = F \cup \{\text{Križevci, Koprivnica}\}$	$S = S \setminus \{\text{Koprivnica}\}$
$T = T \cup \{\text{Varaždin}\}$	$F = F \cup \{\text{Koprivnica, Varaždin}\}$	$S = S \setminus \{\text{Varaždin}\}$
$T = T \cup \{\text{Čakovec}\}$	$F = F \cup \{\text{Varaždin, Čakovec}\}$	$S = S \setminus \{\text{Čakovec}\}$
$T = T \cup \{\text{Virovitica}\}$	$F = F \cup \{\text{Koprivnica, Virovitica}\}$	$S = S \setminus \{\text{Virovitica}\}$
$T = T \cup \{\text{Nova Gradiška}\}$	$F = F \cup \{\text{Sisak, Nova Gradiška}\}$	$S = S \setminus \{\text{Nova Gradiška}\}$
$T = T \cup \{\text{Slavonski Brod}\}$	$F = F \cup \{\text{Nova Gradiška, Slavonski Brod}\}$	$S = S \setminus \{\text{Slavonski Brod}\}$
$T = T \cup \{\text{Đakovo}\}$	$F = F \cup \{\text{Slavonski Brod, Đakovo}\}$	$S = S \setminus \{\text{Đakovo}\}$
$T = T \cup \{\text{Vinkovci}\}$	$F = F \cup \{\text{Đakovo, Vinkovci}\}$	$S = S \setminus \{\text{Vinkovci}\}$
$T = T \cup \{\text{Vukovar}\}$	$F = F \cup \{\text{Vinkovci, Vukovar}\}$	$S = S \setminus \{\text{Vukovar}\}$
$T = T \cup \{\text{Osijek}\}$	$F = F \cup \{\text{Vukovar, Osijek}\}$	$S = S \setminus \{\text{Osijek}\}$
$T = T \cup \{\text{Rijeka}\}$	$F = F \cup \{\text{Karlovac, Rijeka}\}$	$S = S \setminus \{\text{Rijeka}\}$
$T = T \cup \{\text{Pula}\}$	$F = F \cup \{\text{Rijeka, Pula}\}$	$S = S \setminus \{\text{Pula}\}$
$T = T \cup \{\text{Gospić}\}$	$F = F \cup \{\text{Karlovac, Gospić}\}$	$S = S \setminus \{\text{Gospić}\}$
$T = T \cup \{\text{Zadar}\}$	$F = F \cup \{\text{Gospić, Zadar}\}$	$S = S \setminus \{\text{Zadar}\}$
$T = T \cup \{\text{Šibenik}\}$	$F = F \cup \{\text{Zadar, Šibenik}\}$	$S = S \setminus \{\text{Šibenik}\}$
$T = T \cup \{\text{Kaštela}\}$	$F = F \cup \{\text{Šibenik, Kaštela}\}$	$S = S \setminus \{\text{Kaštela}\}$
$T = T \cup \{\text{Split}\}$	$F = F \cup \{\text{Kaštela, Split}\}$	$S = S \setminus \{\text{Split}\}$
$T = T \cup \{\text{Sinj}\}$	$F = F \cup \{\text{Split, Sinj}\}$	$S = S \setminus \{\text{Sinj}\}$
$T = T \cup \{\text{Knin}\}$	$F = F \cup \{\text{Šibenik, Knin}\}$	$S = S \setminus \{\text{Knin}\}$
$T = T \cup \{\text{Imotski}\}$	$F = F \cup \{\text{Sinj, Imotski}\}$	$S = S \setminus \{\text{Imotski}\}$
$T = T \cup \{\text{Makarska}\}$	$F = F \cup \{\text{Imotski, Makarska}\}$	$S = S \setminus \{\text{Makarska}\}$
$T = T \cup \{\text{Dubrovnik}\}$	$F = F \cup \{\text{Makarska, Dubrovnik}\}$	$S = S \setminus \{\text{Dubrovnik}\} = \emptyset$

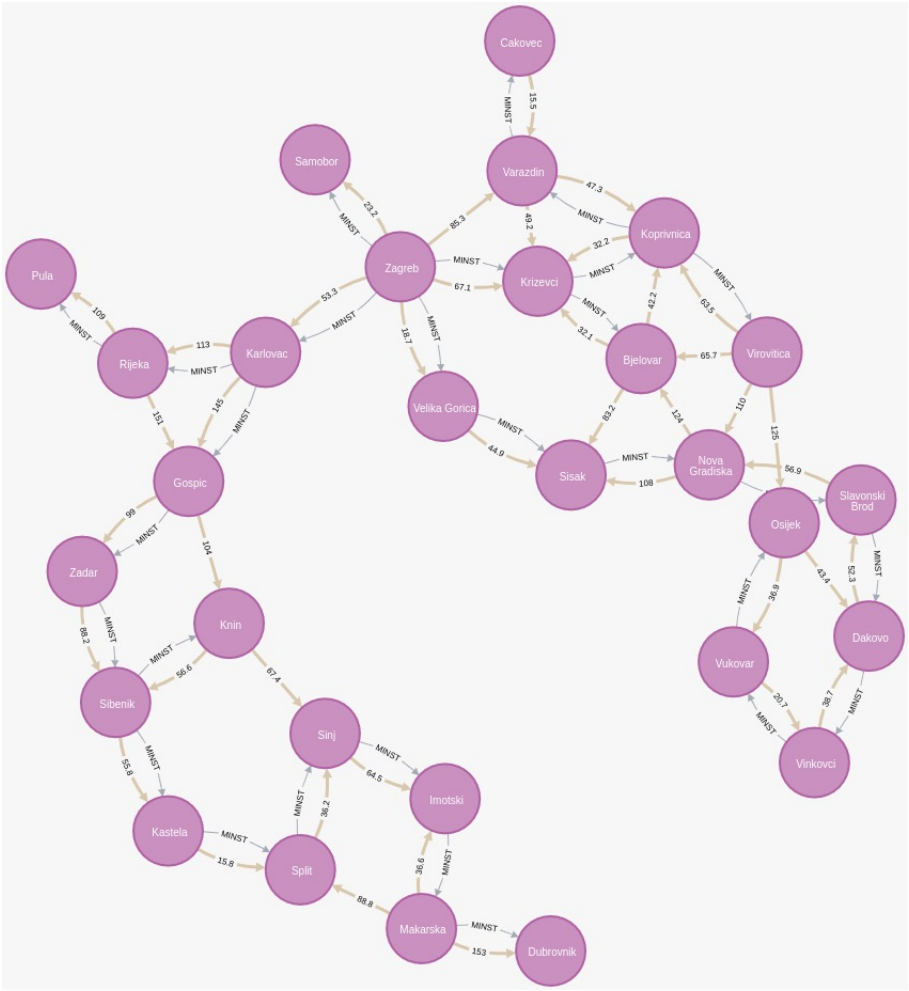
Dakle, dobiveno minimalno razapinjuće stablo sastoji se od dvih 29 vrhova te bridova iz dobivenog skupa F. Provjerimo sada kako izgleda minimalno razapinjuće stablo koje dobijemo koristeći se sljedećim *Cypher* upitom:

```
MATCH path = (n:City name:"Zagreb")-[:MINST*]-()
WITH relationships(path) AS rels
UNWIND rels AS rel
WITH DISTINCT rel AS rel
RETURN startNode(rel).name AS source, endNode(rel).name AS destination,
rel.distance AS cost;
```

Nakon toga upitom `MATCH (n) RETURN n;` dobijemo sliku niže, gdje se nalazi naš graf kojem su dodani bridovi koji će činiti MST. Pogledamo li malo bolje, to su upravo

bridovi koje smo i mi dobili Primovim algoritmom, stoga je težina ovog grafa također 1684, što potvrđuje činjenicu da smo uistinu dobili minimalno razapinjuće stablo.

Slika 8: Minimalno razapinjuće stablo



4.3 Primjena minimalnog razapinjućeg stabla

A zašto je ovo korisno? Recimo da se moraju postaviti telekomunikacijski kabeli između svaka dva grada. Telekomunikacijska tvrtka dakako razmišlja o optimalnom ulaganju, gdje bi potrošili što manje novca, a istovremeno uspjeli na neki način povezati sve gradove (između svaka dva grada postoji telekomunikacijski kabel, makar preko nekog

drugog grada). Pronalaženjem minimalnog razapinjućeg stabla ova tvrtka će dobiti savršen plan za minimiziranje svog troška.