UNIVERZITET U BEOGRADU
MATEMATIČKI FAKULTET

Katarina M. Dimitrijević

ALGORITMI ZA ODREĐIVANJE MINIMALNOG POVEZUJUĆEG DRVETA U GRAFU

master rad

Beograd, 2025.

Mentor:

dr Vesna Marinković, vanredni profesor
Univerzitet u Beogradu, Matematički fakultet

Članovi komisije:

prof. dr Filip Marić, redovan profesor
Univerzitet u Beogradu, Matematički fakultet

dr Mirko Spasić, docent
Univerzitet u Beogradu, Matematički fakultet

Datum odbrane: _____

Mami, tati i bratu

Naslov master rada: Algoritmi za određivanje minimalnog povezujućeg drveta u grafu

Rezime:

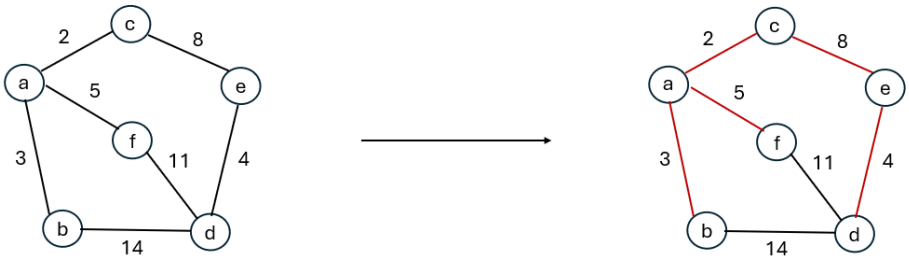
Ključne reči: graf, algoritam, drvo, složenost, minimalnost

Sadržaj

Glava 1

Uvod

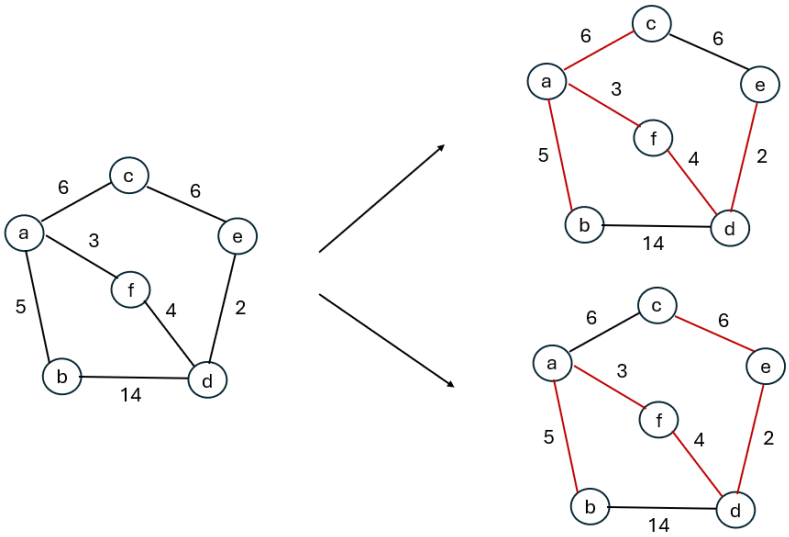
Grafovski algoritmi predstavljaju značajnu podoblast teorije algoritama. Jedan od fundamentalnih problema teorije grafova, koji se smatra osnovom kombinatorne optimizacije, je problem pronalaženja minimalnog povezujućeg drveta (eng. *Minimum Spanning Tree*) u neusmerenom težinskom grafu. U ostatku teksta će za minimalno povezujuće drvo često biti korišćena skraćenica MST.
Minimalno povezujuće (razapinjuće) drvo neusmerenog težinskog grafa predstavlja povezan podgraf zadatog grafa koji sadrži sve njegove čvorove, a čija je ukupna cena grana minimalna. Na slici 1 je prikazan primer neusmerenog težinskog povezanog grafa i njegovog minimalnog povezujućeg drveta [9].



Slika 1.1: Primer grafa i njegovog minimalnog povezujućeg drveta

U slučaju nepovezanih težinskih grafova za svaku od komponenti se može pronaći minimalno povezujuće drvo. Skup svih minimalnih povezujućih drveta nepovezanog težinskog grafa naziva se minimalna povezujuća šuma (eng. *Minimum Spanning Forest*). U nastavku će biti referisana uz pomoć skraćenice MSF.

Ukoliko su težine svih grana u grafu međusobno različite, rešenje problema konstrukcije MST je uvek jedinstveno. U suprotnom to ne mora biti slučaj. Primer povezanog težinskog grafa, kod koga se težine svih grana međusobno razlikuju, i njegovo jedinstveno rešenje ilustrovani su na slici 1. Na slici 1 je dat primer grafa kod koga postoje dve grane iste težine. U zavisnosti od toga koja od njih je u datom trenutku izabrana moguća su dva rešenja.



Slika 1.2: Primer povezanog težinskog grafa koji ima dve grane iste težine i njegova dva različita minimalna povezujuća drveta

Poput većine važnih algoritama, i algoritmi za rešavanje ovog problema, motivisani su problemima iz prakse. Ovaj problem datira još od davnina. Naime, ljudi su od davnina pokušavali da najpre minimizuju putanje i rastojanja između geografskih tačaka, naselja ili drugih objekata u prostoru, a kasnije i troškove vodovodnih, električnih, telekomunikacionih i drugih mreža. Formulacija problema konstrukcije MST predstavlja uopštenje svega navedenog. Prva istraživanja potiču još s početka dvadesetog veka. Naime, češki matematičar Boruvka (Otakar Borůvka) je, inspirisan tada aktuelnim problemom koji se odnosio na dizajn električne mreže, u regionu Južne Moravske u Češkoj, objavio rad 1926. godine koji se bavio ovom temom [1]. Tada je po prvi put formulisan problem konstrukcije MST u matematičkim terminima (odnosno njegovo uopštenje). Interesantno je da se rad sastojao od samo jedne strane, ali se, bez obzira na to, smatra ključnim za rešavanje problema konstrukcije MST. Originalna formulacija problema, data u ovom radu, glasi: Dato je n tačaka

u ravni ili u prostoru čija su sva međusobna rastojanja različita. Cilj je napraviti mrežu koja povezuje ove tačke tako da važi:

- svake dve tačke su povezane ili direktno ili posredstvom drugih tačaka
- ukupna dužine mreže je najmanja moguća [10].

Više detalja o ovom algoritmu biće dato u poglavlju 2.3. Kasnije je objavljeno još nekoliko radova na temu ovog algoritma, nezavisno od Boruvkinog istraživanja. Inspirisan Boruvkinim radom [10], Jarnik (Vojtěch Jarník) je 1930. godine objavio nešto drugačiji algoritam za rešavanje ovog problema [6]. Taj isti algoritam je objavio i Prim (Robert Prim) 1957. godine, a potom i Dijkstra (Edsger Dijkstra) 1958. godine. U literaturi ovaj algoritam je poznat kao Primov algoritam. Kruskal (Joseph Kruskal) je, 1956. godine [8] objavio algortiam zasnovan na dosta intuitivnoj ideji koristeći potpuno drugačiji pristup u odnosu na Primov. Svi do sada spomenuti algoritmi su složenosti $O(|E| \log |V|)$, gde je $|E|$ broj grana, a $|V|$ broj čvorova grafa. Naredni značajan pomak u ovoj oblasti, a i samoj teoriji grafova postigli su Fredman (Michael Fredman) i Tarjan (Bob Tarjan) 1984. godine [5] kada su konstruisali algoritam složenosti $O(|E| \log^* |V|)$[1] koristeći novu strukturu podataka – Fibonačijev hip. Sledeće veliko dostignuće postigao je Karger (David Karger) 1995. godine [7] sa randomizovanim algoritmom očekivane linearne vremenske složenosti. Međutim kako ovo nije bio deterministički algoritam, potraga za algoritmom manje vremenske složenosti je nastavljena. Godine 1997, Šazel (Bernard Chazelle) [2] predstavio je deterministički algoritam vremenske složenosti $O(|E|\alpha(\log |V|))$, gde je sa α označena inverzna Akermanova funkcija. Ova funkcija raste izuzetno sporo, sporije čak i od iterirane logaritamske funkcije. Ipak, ona i dalje teži beskonačnosti kada n teži beskonačnosti, tako da ovime i dalje nije pronađen deterministički algoritam linearne složenosti. Pored spomenutih, postoji još veliki broj algoritama koji predstavljaju varijacije ovih, uz manje vremenske uštede, ali za problem MST još uvek nije poznato da li postoji deterministički algoritam linearne vremenske složenosti.

Problem određivanja minimalnog povezujućeg drveta nalazi primene u različitim domenima. Jedna od tipičnih primena ovog problema je u dizajnu različitih vrsta komunikacionih mreža, kada je potrebno međusobno povezati čvorove mreže tako

[1] Ovde $\log^*$ označava iteriranu logaritamsku funkciju i predstavlja broj puta koliko je potrebno primeniti logaritamsku funkciju dok argument ne postane manji od 1.

da se ne obrazuje ciklus, a da cena mreže bude minimalno moguća. Pored komunikicionih, ovaj problem nalazi primenu i u dizajnu računarskih mreža, električnih i vodovodnih, kao i transportnih mreža.

Koristi se u problemu klasterovanja, kada je cilj grupisati slične tačke tako da se minimizuje ukupno rastojanje između klastera (kod hijerarhijskog tipa klasterovanja). Ideja je da se za zadati skup tačaka konstruiše MST korišćenjem nekog poznatog algoritma. MST se smatra jednim klasterom koji sadrži sve čvorove, a zatim se jedna po jedna grana uklanja (počevši od grane najveće težine) i ostaju grupisani elementi. Ovo je klasterovanje naniže, a uz pomoć MST se može postići i klasterovanje naviše.

Takođe, nalazi primenu i u razvijanju približnih algoritama za rešavanje nekih NP-teških problema, na primer, problema trgovačkog putnika (eng. *Traveling Salesman Problem*). Zadatak problema trgovačkog putnika je da se u datom grafu pronađe Hamiltonov ciklus sa minimalnom ukupnom cenom grana. MST se može iskoristiti za rešavanje jedne varijante problema trgovačkog putnika, kada težine grana odgovaraju euklidskim rastojanjima između čvorova. Dakle, potrebno je da trgovac obiđe n datih gradova, koje odgovaraju tačkama u ravni, tako da svaki od njih poseti tačno jednom a da pritom pređe što kraći put. Pritom se podrazumeva da su udaljenosti između svaka dva grada poznate i da odgovaraju rastojanju odgovarajućih tačaka u ravni. MST se u ovom algoritmu koristi za izgradnju približnog algoritma sa faktorom aproksimacije 2. Naime, udvostručavanjem grana MST grafa i skraćivanjem ponovljenih čvorova korišćenjem nejednakosti trougla dolazi se do obilaska grafa koji je u najgorem slučaju 2 puta duži od optimalnog rešenja problema TSP. Ovo rešenje je moguće dodatno popraviti.

Cilj ovog rada je implementacija i analiza različitih algoritama za rešavanje problema minimalnog povezujućeg drveta. Obrađeni algoritmi biće međusobno poređeni na testnom uzorku odabranih grafova.

Glava 2

Pregled algoritama za konstrukciju minimalnog povezujućeg drveta

2.1 Primov algoritam

Primov algoritam je osmišljen 1930. godine kao matematički algoritam u okviru teorije grafova, a 1957. godine je objavljen rad iz oblasti informatike u kome je, po drugi put, predstavljen isti algoritam.

Kako je minimalno povezujuće drvo određeno skupom grana datog grafa, prirodno se nameće ideja da se ono može konstruisati dodavanjem jedne po jedne grane. Naredna grana se bira tako da bude najmanje težine od svih grana koje povezuju do tada konstruisano drvo sa čvorovima koji još uvek nisu dodati u tekuće drvo. Postupak se završava kada se svi čvorovi dodaju u drvo. Dokaz korektnosti opisanog algoritma se može jednostavno sprovesti uz pomoć indukcije po broju grana dodatih u trenutni podgraf minimalnog povezujućeg drveta [9].

Induktivna hipoteza: Za povezan neusmeren težinski graf G, zadat skupom čvorova V i skupom grana E, u oznaci $G = (V, E)$, može se konstruisati drvo T_k koje sadrži k grana, $k < |V| - 1$, tako da je ono podgraf nekog MST-a zadatog grafa.

Baza indukcije: Za bazu indukcije se uzima slučaj kada je k=0, tada trenutno drvo T_0 ne sadrži nijednu granu i može sadržati proizvoljan čvor drveta. Pošto je zadati graf povezan, minimalno povezujuće drvo uvek postoji i sadrži sve čvorove zadatog grafa, te je baza indukcije ispunjena.

Induktivni korak: Pretpostavimo da imamo na raspolaganju drvo T_k koje zadovoljava induktivnu hipotezu i koje se sastoji od k grana. Postavlja se pitanje kako pronaći narednu granu minimalnog povezujućeg drveta koja bi proširila induktivnu

hipotezu.

Neka je G_k skup koji sadrži sve grane (a, b), tako da čvor a pripada T_k, a čvor b ne pripada T_k. Ideja je da se za narednu granu bira grana minimalne težine iz skupa G_k. Neka je grana minimalne težine grana e. Tvrdimo da svaka grana, izabrana na ovaj način, sigurno pripada nekom minimalnom povezujućem drvetu. Odnosno, drugim rečima, tvrdimo da postoji minimalno povezujuće drvo T' koje sadrži T_k i granu e.

Dokaz: Neka je T minimalno povezujuće drvo grafa G. Ukoliko grana e pripada drvetu T grafa G, u tom slučaju je $T = T'$. Ukoliko drvo T ne sadrži granu $e = (a, b)$ dokaz je malo komplikovaniji. Naime, pošto je T povezujuće drvo, mora postojati put od a do b kroz grane minimalnog povezujućeg drveta i on mora biti jedinstven, jer bi u suprotnom postojao ciklus. Takođe, zaključujemo da grana e ne sme pripadati putu od a do b u T. Posto čvor a pripada, a čvor b ne pripada drvetu T_k, na tom putu mora postojati bar još jedna grana $e' = (a', b')$, tako da se čvor a' nalazi u T_k, a čvor b' ne. Postavlja se pitanje odnosa težina grana e i e':

- Grana e' ne može biti manje težine od grane e jer je e odabrano kao grana minimalne težine iz skupa G_k.
- Međutim e' ne može biti ni veće težine od e jer bi se u tom slučaju, uklanjanjem grane e' iz drveta T i dodavanjem grane e, dobilo drvo manje ukupne cene od T, što bi dovelo do kontradikcije.
- Na osnovu prethodnog razmatranja jasno se zaključuje da u ovom slučaju grane e i e' moraju imati iste težine. Na taj način uklanjanjem grane e' i dodavanjem grane e u drvo T, ono ostaje minimalno povezujuće drvo grafa G. Odnosno, postoji *minimalno* drvo T koji *sadrži* granu e.

Primov algoritam polazi od praznog skupa grana i u svakoj iteraciji dodaje po jednu granu čime se drvo proširuje. Do svakog čvora, iz skupa čvorova koji još uvek ne pripadaju G_k, pamti se minimalna cena grane od čvorova iz G_k, a ukoliko takva grana ne postoji smatra se da težina iznosi $+\infty$. Nakon odabira naredne grane, proveravaju se cene grana od njenog krajnjeg čvora do svih susednih čvorova i ukoliko je cena neke od tih grana manja od cene tekuće minimalne grane do tog čvora, koja je u prethodnim iteracijama izračunata, minimalna tekuća cena grane do tog čvora se ažurira. Opisani postupak je skoro pa identičan Dajkstrinom algoritmu za računanje najkraćih puteva iz zadatog čvora, izuzev što se kod Dajkstrinog algoritma

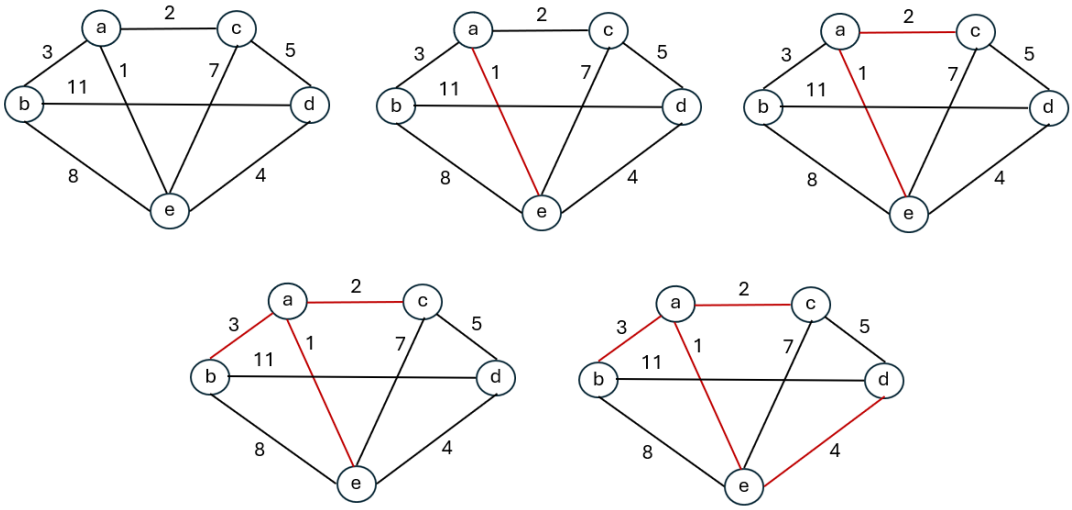
putanje minimizuju po ukupnoj ceni od početnog čvora, a kod Primovog od trenutno konstruisanog drveta.

Analiza složenosti: U implementaciji Primovog algoritma koristi se red sa prioritetom. Najčešće se za implementaciju reda sa prioritetom koristi binarni hip, koji je drvolika struktura podataka i kod koga operacije dodavanja proizvoljnog elementa i uklanjanja elementa sa najvećim prioritetom imaju u najgorem slučaju logaritamsku složenost. Kako se u korenu binarnog min-hipa čuva minimalni element, pristup minimalnom elementu se izvršava u konstantnom vremenu. Ova struktura podataka se koristi za čuvanje težina grana do svih suseda trenutnog drveta T_k, tako da se u svakom trenutku može efikasno pronaći grana minimalne težine koja se dodaje u trenutno minimalno povezujuće drvo. Svaki od čvorova se najviše jednom dodaje u red sa prioritetom, pa je vreme potrebno za dodavanje svih čvorova, kao i vreme potrebno za brisanje iz reda $O(|V| \log |V|)$. Tokom konstrukcije minimalnog povezujućeg drveta grane najmanje težine do svakog od čvorova van T_k se potencijalno ažuriraju, međutim tih operacija je ukupno najviše $|E|$, a vreme koje je potrebno za sva ažuriranja iznosi $O(|V| \log |V|)$. Stoga je složenost Primovog algoritma ekvivalentna složenosti Dajkstrinog i iznosi $O((|E| + |V|) \log |V|)$.

Umesto binarnog, može se koristiti neki drugi hip, npr. Fibonačijev hip i time se može poboljšati teorijska vremenska složenost. Ovakva poboljšanja postaju značajna u slučaju veoma velikih grafova. Primov algoritam se smatra prilično efikasnim algoritmom, naročito za retke grafove (one koji imaju mali broj grana) i tada složenost iznosi $O(|V| \log |V|)$.

Na slici 2.1 je ilustovan rad Primovog algoritma, korak po korak. Minimalno povezujuće drvo za zadati graf sa slike čine grane: (a, e), (a, c), (a, b) i (e, d), a njegova ukupna cena iznosi 11. Postupak čuvanja cena grana do svih čvorova i njihovo ažuriranje kroz iteracije može se videti u tabeli 2.1. Prvi element i-te vrste tabele odgovara čvoru koji je dodat u i-toj iteraciji, dok ostali elementi te vrste odgovaraju cenama minimalnih, do sada razmatranih, grana između njega i čvorova koji još uvek ne pripadaju konstruisanom drvetu. Vrednost beskonačno označava da put između neka dva čvora još uvek nije poznat.

Primov algoritam je primer *pohlepnog* algoritma, što znači da uvek bira trenutno najbolje rešenje, tj. lokalni minimum, koji vodi ka najboljem rešenju kompletnog



Slika 2.1: Primer izvršavanja Primovog algoritma

	a	b	c	d	e
a	-	$a(3)$	$a(2)$	∞	$a(1)$
b	-	-	$a(2)$	$b(11)$	$a(1)$
c	-	-	-	$e(4)$	$a(1)$
d	-	-	-	-	$a(1)$
e	-	-	-	-	-

Tabela 2.1: Cene minimalne grane do svakog od čvorova van T_k kroz iteracije

problema.

2.2 Kruskalov algoritam

Kruskal je 1956. godine osmislio algoritam za konstrukciju minimalnog povezujućeg drveta, koji je zasnovan na nešto drugačijoj ideji od Primovog.

Ideja koja stoji iza Kruskalovog algoritma je da se minimalno povezujuće drvo konstruiše dodavanjem jedne po jedne grane na trenutnu šumu, a ne na trenutno drvo, kao što je to slučaj u Primovom algoritmu. Sam algoritam započinje izvršavanje nad skupom čvorova, tako da svaki čvor predstavlja zasebno drvo, koje nema grana. U svakoj iteraciji se razmatra naredna grana iz neopadajuće sortiranog skupa grana, na osnovu njihovih cena. Ukoliko čvorovi, koji su incidenti sa tekućom granom,

pripadaju različitim drvetima tekuće šume, ta grana se dodaje u tekuću šumu, dok se u suprotnom preskače. Dodavanjem grane se vrši spajanje drveta kojima pripadaju čvorovi trenutne grane. Na ovaj način se dodavanjem svake grane broj drveta trenutne šume smanjuje za jedan. Lako se zaključuje da se postupak završava nakon dodavanja $|V| - 1$ grane, kada se formira jedinstveno drvo, odnosno podgraf polaznog grafa koji ne sadrži cikluse. Jasno je da ovaj postupak rezultira povezujućim drvetom, samo je još potrebno pokazati da je ono i minimalno [9]. Dokaz se može sprovesti indukcijom po broju obrađenih grana iz neopadajuće sortiranog niza grana na osnovu njihovih težina.

Induktivna hipoteza: Za zadati graf G i skup njegovih grana E (koje su neopadajuće sortirane po težinama), umemo da formiramo šumu K koja sadrži $k - 1$ grana iz skupa E, tako da K pripada minimalnom povezujućem drvetu T. Grane se biraju redom iz neopadajuće sortiranog skupa grana, pod uslovom da ne formiraju ciklus. Drugim rečima, postoji minimalno povezujuće drvo T koje sadrži sve grane iz K, a ne sadrži nijednu granu iz $E \setminus K$ (one koje su odbačene jer zatvaraju neki ciklus).

Baza indukcije: Za bazni slučaj se uzima da je $k = 0$, odnosno da nijedna grana još uvek nije obrađena, tako da je trenutna šuma $K = \emptyset$, ali je i skup odbačenih grana takođe prazan.

Induktivni korak: Ako je na osnovu induktivne hipoteze poznato kako kreirati tekuću šumu koja se sastoji od $k - 1$ grana, postavlja se pitanje kako na nju dodati narednu granu tako da induktivna hipoteza ostane ispunjena. Ako je sledeća grana koja se razmatra grana e, moguća su tri slučaja:

- Grana e može biti odbačena. Do odbacivanja dolazi ukoliko ona formira neki ciklus sa granama iz K. Pošto se sve grane iz K nalaze i u T, sledi da bi grana e formirala ciklus i u T. Kako u drvetu T nema ciklusa, grana e ne pripada T, tako da odbacivanjem grane e induktivna hipoteza ostaje ispunjena.

- Ukoliko grana e ne formira ciklus u K, ona se dodaje. U slučaju da se razmatrana grana nalazi u drvetu T induktivna hipoteza ostaje ispunjena i dokaz je završen.

- Ukoliko grana e ne formira ciklus u K, ona se dodaje, ali se može desiti da se ova grana ne nalazi u T. Kako se ona ne nalazi u T, znači da bi se njenim dodavanjem formirao neki ciklus. U tom ciklusu mora postojati neka grana e', koja se ne nalazi u K. Pošto se ona nalazi u T, znači da se ne nalazi u skupu odbačenih grana. Razmotrimo njenu težinu. Kako ona nije još uvek razmatrana

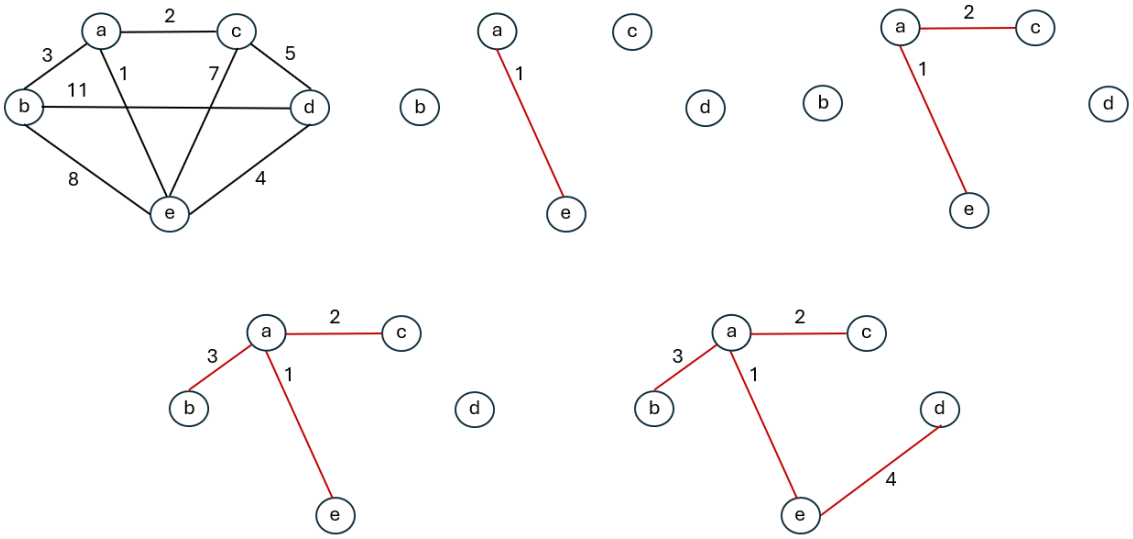
od strane algoritma, pa samim tim nije dodata u K, njena težina mora biti veća od težine grane e. Međutim, ako je veća onda se njenim izbacivanjem iz T i dodavanjem grane e dobija drvo manje težine što je kontradikcija jer je T već minimalno. Zaključujemo da su težine grana e i e' zapravo jednake. Na ovaj način zamenom grane e' granom e u drvetu T, ono ostaje minimalno i induktivna hipoteza ostaje ispunjena.

Analiza složenosti: u implementaciji ovog algoritma se za spajanje dva drveta, prilikom dodavanja grane, kao i za proveru da li krajnji čvorovi grana pripadaju istom ili različitim drvetima, može iskoristi efikasna namenska struktura DSU (eng. *Disjoint Set Union*), poznatija kao *union-find* [9]. Ova struktura podataka omogućava uniranje dva podskupa i određivanje kom podskupu pripada neki element u vremenskoj složenosti koja je logaritamska u odnosu na broj elemenata u skupu. Glavna petlja algoritma se izvršava $|E|$ puta. U petlji se pozivaju operacije pronalaženje predstavnika (eng. *find*) i spajanje drveta (eng. *union*) koje su složenosti $O(\log |V|)$. Takođe, na samom početku je potrebno inicijalizovati ovu pomoćnu strukturu što je vremenske složenosti $O(|V|)$, jer na samom početku svaki čvor predstavlja komponentu za sebe. Takođe, potrebno je izdvojiti i sortirati sve grane, što je u proseku složenosti $O(|E| \log |E|)$. Sumiranjem svega navedenog i korišćenjem poznate nejednakosti $|E| \leq |V|^2$, dolazimo do ukupne složenosti $O(|E| \log |V|)$. Slično kao i Primov, smatra se efikasnim algoritmom, ali naročito za retke grafove. Kruskalov algoritam je takođe primer *pohlepnog algoritma*.

Na slici 2.2 je prikazan primer izvršavanja Kruskalovog algoritma koji na ulazu dobija graf sa slike. Rad algoritma započinje nad skupom čvorova V, gde svaki od njih čini drvo za sebe i zajedno formiraju trenutnu šumu $\{a\}$, $\{b\}$, $\{c\}$ i $\{d\}$ i $\{e\}$. Uzima se jedna po jedna grana iz sortiranog niza i uključuje se samo ukoliko ne zatvara ciklus. Tekuću šumu nakon prve iteracije čine drveta sa skupom čvorova: $\{a, e\}$, $\{b\}$, $\{c\}$ i $\{d\}$, nakon druge: $\{a, c, e\}$, $\{b\}$ i $\{d\}$ i nakon treće $\{a, b, c, e\}$ i $\{d\}$. Na samom kraju formira se jedinstveno drvo sa skupom čvorova $\{a, b, c, d, e\}$ i skupom grana $\{(a, b), (a, c), (a, e), (d, e)\}$, odnosno, algoritam se zaustavlja kada se konstruiše drvo koje sadrži $|V| - 1$ granu.

2.3 Boruvkin algoritam

Još jedan algoritam za pronalaženje minimalnog povezujućeg drveta zadatog grafa, nastao 1926. godine, je Boruvkin algoritam. Ideja na kojoj se zasniva je veoma



Slika 2.2: Primer izvršavanja Kruskalovog algoritma

intuitivna i jednostavna. Polazi se od skupa čvorova ulaznog grafa, gde se svaki od njih posmatra kao zasebna komponenta povezanosti, odnosno, drvo, kao i kod Kruskalovog algoritma. Ideja je da se te komponente spajaju dok ne ostane samo jedna, koja će zapravo predstavljati traženo minimalno povezujuće drvo početnog grafa i tada se algoritam zaustavlja. Ono što se zapravo razlikuje u odnosu na Kruskalov algoritam je način odabira grana koje ulaze u minimalno povezujuće drvo, odnosno sama konstrukcije drveta.

U prvom koraku se za svaki čvor iz skupa V pronalazi grana najmanje cene koja je incidentna sa tim čvorom i ona se dodaje na trenutnu šumu. Ovime se broj komponenti smanjuje. Ovaj korak se ponavlja, s tim što se nakon prve iteracije više ne biraju grane koje povezuju pojedinačne čvorove, već grane koje povezuju novonastale komponente. Postupak se zaustavlja kada ostane samo jedna komponenta povezanosti. Drugim rečima, u svakom koraku se bira grana koja povezuje dve različite komponente povezanosti, čime se garantuje da neće doći do zatvaranja ciklusa, što je osnovni uslov za konstruisanje povezujućeg drveta grafa. Rezultat prethodno opisanog postupka je povezujuće drvo inicijalnog grafa. Ovo tvrđenje se jednostavno može dokazati indukcijom po broju komponenti povezanosti u trenutnoj šumi.

Ostaje pokazati da je ovo drvo ujedno i minimalno, odnosno drvo najmanje ukupne cene.

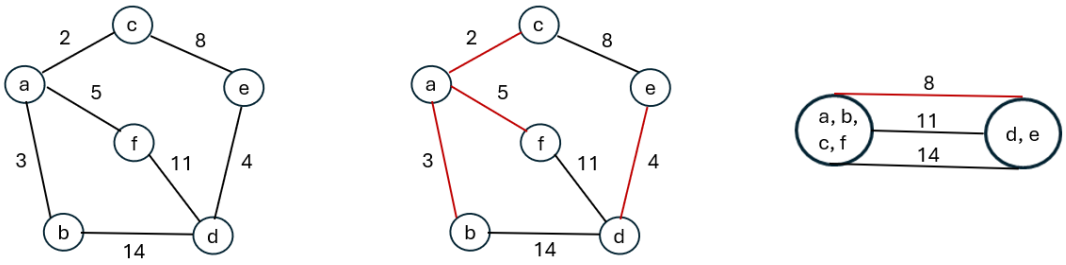
Dokaz: Pokažimo da je povezujuće drvo dobijeno opisanim postupkom minimalno svođenjem na kontradikciju. Neka je T minimalno povezujuće drvo zadatog grafa, a T' povezujuće drvo koje je konstruisao Boruvkin algoritam. Ako su sve cene grana grafa međusobno različite, tvrdimo da mora da važi $T = T'$. Pretpostavimo suprotno, tj. da se ova dva drveta razlikuju i neka je $e' = (a, b) \in T'$ prva grana drveta T' dobijena Boruvkinim algoritmom kao grana minimalne težine iz čvora a, za koju važi da ne pripada minimalnom povezujućem drvetu T. Kako je T povezujuce drvo, postoji put od cvora a do cvora b kroz grane drveta T. Taj put mora da sadrzi granu $e = (a, v)$ za $v \neq b$. Ova grana ne može biti manje težine od grane (a, b) jer je grana (a, b) izabrana u Boruvkinom algoritmu kao grana najmanje težine iz a. Stoga je težina grane e veća od težine grane e'. Međutim, onda važi da je i $T - e + e'$ manje ukupne cene od cene drveta T , što je u suprotnosti sa tim da je T MST grafa [10].

Prethodni dokaz važi u slučaju kada su sve grane zadatog grafa međusobno različite cene što je Boruvka i pretpostavio u svom prvom radu iz 1926. godine. Ukoliko bi graf mogao imati grane sa istim težinama, dokaz bi bio malo komplikovaniji, a primeri takvih dokaza dati su u poglavljima o Primovom i Kruskalovom algoritmu. Kada je nastao, ovaj algoritam je bio poznat pod nazivom *širenje šume* (eng. *parallel merging, forest growing*). Kao i prethodna dva algortima, ovo je takođe primer *pohlepnog algoritma.*

U implementaciji ovog algoritma se novoformirana komponenta identifikuje svojim predstavnikom, a sve grane unutar jedne komponente se eliminišu. Takođe, prilikom spajanja komponenti granom najmanje cene, sve ostale grane između njih mogu biti eliminisane. Za potrebe ovih operacija se koristi efikasna, već ranije spomenuta, union-find struktura.

Analiza složenosti: U svakoj iteraciji se na trenutnu šumu dodaje bar $|V|/2$ grana, čime se broj komponenti smanjuje bar dvostruko u odnosu na početni graf, te je broj iteracija algoritma $O(\log |V|)$. U svakoj od tih iteracija se svaka grana razmatra najviše jednom. Dakle, ukupna složenost Boruvkinog algoritma iznosi $O(|E| \log |V|)$. Iako ima istu složenost kao i Primov i Kruskalov algoritam, za razliku od njih, pokazuje se znatno efikasnijim na gustim grafovima. Takođe, njegova velika prednost je mogućnost paralelizacije.

Na slici 2.3 je prikazan primer izvršavanja Boruvkinog algoritma. U prvoj iteraciji se svaki čvor grafa iz skupa $V = \{a, b, c, d, e, f\}$ posmatra kao zasebna komponenta. Za svaki od njih se pronalazi njemu incidentna grana najmanje cene tako da ga ona povezuje sa drugom komponentom. Pregled takvih grana za svaku od komponenti,



Slika 2.3: Primer izvršavanja Boruvkinog algoritma

Komponenta	Grana	Cena
$\{a\}$	(a, c)	2
$\{b\}$	(a, b)	3
$\{c\}$	(a, c)	2
$\{d\}$	(d, e)	4
$\{e\}$	(d, e)	4
$\{f\}$	(a, f)	5

Tabela 2.2: Grane obeležene u prvoj iteraciji

Komponenta	Grana	Cena
$\{a, b, c, f\}$	(c, e)	8
$\{d, e\}$	(c, e)	8

Tabela 2.3: Grane obeležene u drugoj iteraciji

nakon završene prve iteracije, dat je u tabeli 2.3.

Ove grane se označavaju i tako se formiraju nove komponente koje sadrže njima incidentne čvorove. Primenom operacija unije nastaju nove komponente i one sada postaju ulaz u drugu iteraciju algoritma. Postupak se ponavlja. Grane koje su obeležene u drugoj iteraciji su date u tabeli 2.3.

Obeležavanjem poslednje grane nastaje jedinstvena komponenta i algoritam se zaustavlja. Prolaskom kroz označene grane se konstruiše minimalno povezujuće drvo polaznog grafa. Njega čine grane: (a, c), (a, b), (a, f), (c, e) i (a, c), ukupne cene 22.

2.4 Algoritam obrnutog brisanja grana

Još jedan algoritam iz grupe pohlepnih algoritama za pronalaženje minimalnog povezujućeg drveta je algoritam obrnutog brisanja grana (eng. *delete-reverse*) algo-

ritam. Algoritam na ulazu dobija neusmeren, težinski graf. Ideja algoritma je da se inicijalno grane sortiraju u opadajućem poretku (u opštem slučaju u nerastućem poretku), a da se zatim za svaku od njih razmatra da li nakon njenog uklanjanja graf ostaje povezan. Ukoliko je to slučaj, trenutna grana se uklanja i algoritam nastavlja dalje sa izvršavanjem. U protivnom se ta grana zadržava i prelazi se na sledeću granu iz sortiranog niza. Ideja algoritma je jednostavna i intuitivna i u velikoj meri odgovara ideji Kruskalovog algoritma [1]. Razlika je u tome što se u Kruskalovom algoritmu grane sortiraju u rastući (neopadajući) poredak, razmatraju jedna po jedna i dodaju na trenutnu šumu, osim ukoliko formiraju ciklus.

Iako se na osnovu samog opisa algoritma lako može zaključiti da on zaista vraća minimalno povezujuće drvo, u nastavku sledi dokaz. Prvi deo dokaza, odnosno tvrđenje da algoritam obrnutog brisanja grana nalazi povezujuće drvo je veoma jednostavano pokazati. Algoritam na ulazu dobija povezani, težinski graf G. Pošto je graf povezan, njegovo povezujuće drvo mora postojati. Uklanjanjem jedne po jedne grane, uz uslov da graf sve vreme ostaje povezan, na kraju ostaje graf koji sadrži inicijalni skup čvorova, ali iz koga su uklonjene sve grane koje formiraju neki ciklus, što je po definiciji povezujuće drvo grafa. Drugi deo dokaza je malo komplikovaniji i odnosi se na minimalnost ovako dobijenog drveta. Ovaj deo tvrđenja se može pokazati primenom matematičke indukcije.

Induktivna hipoteza: Prepostavimo da F, podgraf polaznog grafa G koji je dobijen obradom $k-1$ grane sa najvećom težinom iz G, sadrži neko njegovo minimalno povezujuće drvo T.

Baza indukcije: Ovo tvrđenje je tačno pre prve iteracije algoritma jer je inicijalno $F = G$, a kako je to povezan graf, kao takav uvek sadrži minimalno povezujuće drvo T.

Induktivni korak: Neka je F podgraf grafa G koji je nastao nakon $k-1$ iteracije algoritma. Neka je grana e $k-$ta po redu grana sortiranog niza. Moguća su tri slučaja:

- Grana e se ne izbacuje iz F. Pošto je na osnovu induktivne hipoteze F sadržalo neko MST T, kako se trenutna grana se ne izbacuje iz F, sledi da induktivna hipoteza ostaje nenarušena.

- Grana e se izbacuje iz F i ne pripada T. Ukoliko se grana ne nalazi u T, ona

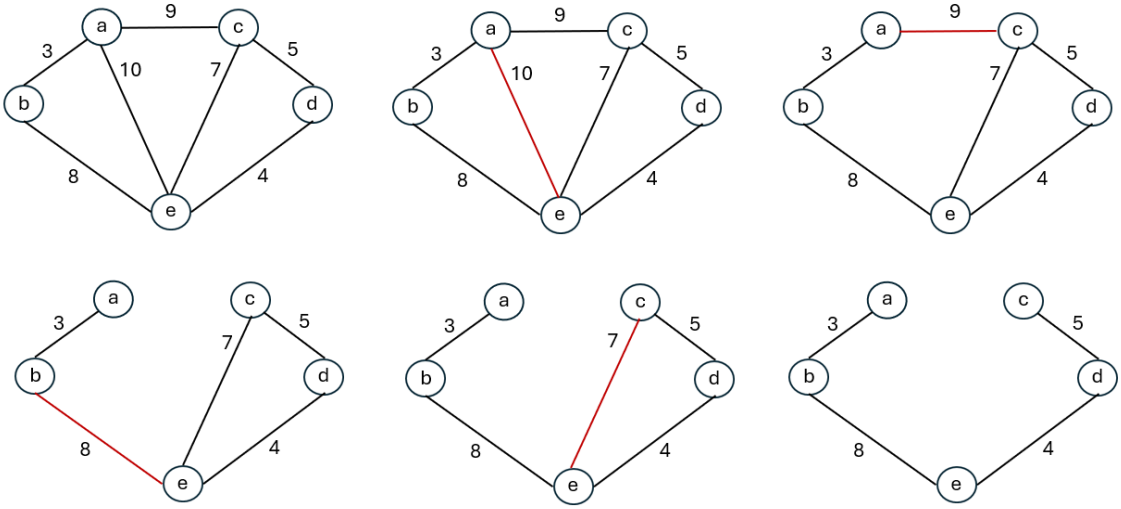
[1] I algoritam obrnutog brisanja grana i Kruskalov algoritam su objavljeni u istom radu 1956. godine, ali ih ne treba poistovetiti [8].

se može izbaciti iz F i time se ne narušava induktivna hipoteza jer i nakon izbacivanja trenutne grane, F i dalje sadrži neko MST T.

- Grana e se izbacuje iz F, ali pripada T. Pošto je ova grana izbačena, može se lako zaključiti da njenim izbacivanjem graf G ostaje povezan. Neka je $e = (u, v)$. U tom slučaju mora postojati neki drugi put od čvora u do čvora v. Jasno je da na tom putu postoji neka grana e' iz tog ciklusa koja se nalazi van T, jer u suprotnom T ne bi bilo povezujuće drvo. Ona mora pripadati nekom drugom drvetu T' koje je takođe podgraf od F i da važi $T' = T - e + e'$. Lako je pokazati da je da je T' zaista drvo jer nastalo izbacivanjem grane e iz drveta T i dodavanjem nove grane e', tako da se na taj način zadržavaju svojstva drveta. Sada možemo takođe pokazati i da je drvo T' minimalno. Posmatrajmo cene grana e i e'. Lako se pokazuje da nije moguće $w(e) > w(e')$ jer je T minimalno povezujuće drvo. Takođe, nije moguće ni $w(e) < w(e')$ jer algoritam uklanja grane u opadajućem redosledu što znači da bi onda grana e' bila razmatrana pre grane e i ovo dovodi do kontradikcije u razmatranom slučaju. Iz navedenog sledi da mora važiti da ove dve grane imaju jednake cene, odnosno da je $T' = T$, što znači da u svakoj iteraciji važi da F sadrži minimalno povezujuće drvo. Na samom kraju izvršavanja, sve grane koje učestvuju u nekom ciklusu su izbačene, što znači da je samo F drvo, odnosno, samo F je minimalno povezujuće drvo.

Analiza složenosti: vreme potrebno za sortiranje grana grafa je $O(|E| \log |E|)$. Kako bismo utvrdili da li bi nakon izbacivanja svake od grana graf ostao povezan, potrebno je na preostalom grafu pokrenuti neki vid pretrage (npr. DFS pretraga) čija složenost iznosi $O(|E| + |V|)$. Dakle, ukupna vremenska složenost iznosi: $O(|E| \log |E| + |E|(|E| + |V|))$.

Na slici 2.4 je ilustrovano izvršavanje algoritma. Grane se razmatraju jedna po jedna u opadajuće sortiranom redosledu težina (crvenom bojom su obeležene grane koje se trenutno razmatraju) i uklanjaju se sve one koje ulaze u sastav nekog ciklusa. Najpre se razmatra grana (a, e), a zatim i grana (a, c). Za obe se zaključuje da graf nakon njihovog uklanjanja ostaje povezan, te one ne pripadaju MST. Naredna grana koja se razmatra je grana (b, e), međutim, njenim brisanjem bi graf postao nepovezan tako da se ona preskače. Zatim se uklanja i grana (c, e). Sledeća je na redu (b, e), ali ni ona, niti ijedna od preostalih grana se ne smeju ukloniti jer nijedna od njih ne ulazi u sastav nekog ciklusa. Drugim rečima, grane koje su preostale čine minimalno



Slika 2.4: Primer izvršavanja algoritma obrnutog brisanja grana

povezujuće drvo.

2.5 KKT algoritam

Pokazuje se da je moguće razviti i efikasniji algoritam za računanje MST ukoliko uvedemo koncept nasumičnosti. Karger je objavio *randomizovani* algoritam 1992. godine i njegova očekivana vremenska složenost iznosi $O(|E| + |V| \log |V|)$. Klein i Tarjan su ga 1994. godine optimizovali i popravili njegovu vremensku složenost na $O(|E| + |V|)$. Kako je originalna ideja potekla od Kargera, algoritam je poznat i pod nazivom *Kargerov algoritam*. U nastavku će na ovaj algoritam biti referisano sa KKT (Karger-Klein-Tarjan). Ovaj algoritam spada u grupu tzv. podeli-pa-vladaj (eng. *divide-and-conquer*) algoritama [4].

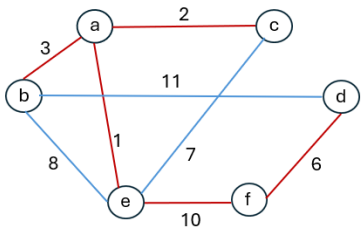
Ideja na kojoj je zasnovan KKT algoritam je sledeća: konstruisati nadgraf MST grafa G, a zatim iz njega odbaciti one grane koje ne pripadaju MST korišćenjem svojstva ciklusa, koje je objašnjeno u nastavku. U ostatku grafa G problem rekurzivno rešiti. Izdvajamo nekoliko bitnih svojstava:

- *Svojsvo ciklusa* (eng. *cycle property*) - za proizvoljan ciklus grafa G važi da grana najveće težine koja pripada tom ciklusu ne pripada minimalnom povezujućem drvetu tog grafa.

- *Svojsvo preseka* (eng. *cut property*) - za bilo koji podskup čvorova ulaznog grafa važi da od svih grana koje povezuju taj skup čvorova sa ostatkom grafa, ona koja ima najmanju težinu pripada minimalnom povezujućem drvetu tog grafa.

Za potrebe razmatranja algoritma uvodimo pojam teških i lakih grana u odnosu na neko drvo. Neka je F šuma, tako da je ona podgraf od G. Za granu e, koja pripada G, za koju važi da se njenim dodavanjem u F zatvara neki ciklus i da je ona grana najveće težine u njemu, kažemo da je ona *teška u odnosu na* F, tj. F-teška (eng. F-*heavy*). U suprotnom granu e nazivamo *lakom u odnosu na* F(eng. F-*light*), tj. F-lakom. Iz ove dve definicije se lako zaključuje da svaka grana koja pripada MST grafa G spada u F-lake. Ovo tvrđenje važi i u suprotnom smeru, odnosno, svaka F-laka grana pripada MST grafa G. Takođe, za F-teške grane važi da su one teške i u odnosu na minimalno povezujuće drvo grafa G, odnosno da ona predstavlja najtežu granu nekog ciklusa i u MST polaznog grafa.

Na slici 2.5 da je primer grafa na kome su obeležene lake grane (crvenom bojom) i one čine MST grafa i teške grane (plavom bojom). Teške grane nisu deo MST grafa i svaka od njih je grana najveće težine u nekom ciklusu trenutnog grafa.



Slika 2.5: Primer grafa na kome su označene F-lake i F-teške grana

U nastavku sledi opis samog algoritma. Najpre se iz skupa grana $E(G)$ grafa G nasumično odabere podskup grana $E'(G)$, a potom se iz njega uklone sve teške grane. Zatim se na ostatku grana rekurzivno rešava isti problem, ali manje dimenzije. Algoritam je sastavljen od kombinacija koraka Boruvkinog algoritma (eng. *Boruvka steps*) i nasumičnog uzorkovanja (eng. *random sampling*). Boruvkin algoritam se koristi kako bi se broj čvorova najmanje dvostruko smanjio, dok nasumično uzorkovanje redukuje broj grana. Koraci KKT algoritma:

- Inicijalno se nekoliko puta (obično tri puta) pokrene Boruvkin algoritam nad grafom $G(V, E)$, a kao rezultat ovog koraka dobija se graf $G'(V', E')$, tako da važi $V' \leq V/8$ i $E' \leq E$. Ovaj algoritam, kao što je već objašnjeno u poglavlju 2.3, za svaki čvor grafa pronalazi njemu incidentnu granu najmanje težine, a zatim sažima svaka dva čvora povezana ovim granama u jednu komponentu. U ovom koraku algoritma se uklanjaju sve petlje koje su postojale na čvorovima ulaznog grafa i zadržavaju se samo grane najmanje težine među svim granama koje povezuju novonastale komponente povezanosti, dok se sve ostale grane uklanjaju. U ovom koraku se broj čvorova smanjuje bar dvostruko. Ukoliko novonastalo drvo G' sadrži samo jedan jedini čvor, to znači da je rezultat Boruvkine faze ujedno i krajnji rezultat KKT algoritma.

- Na skupu grana E', bira se uzorak E_1, nezavisnim uzorkovanjem sa verovatnoćom odabira svake grane od 0.5.

- Algoritam se rekurzivno poziva na novoodabranom skupu grana E_1 čime se kao rezultat dobija minimalna povezujuća šuma $F1 = MST(V', E1)$.

- Pronalaze se i uklanjaju sve F_1-teške grane. Skup preostalih grana označićemo sa E_2.

- Algoritam se rekurzivno poziva na skupu grana E_2, čime se kao rezultat dobija minimalna povezujuća šuma $F_2 = MST(V', E_2)$.

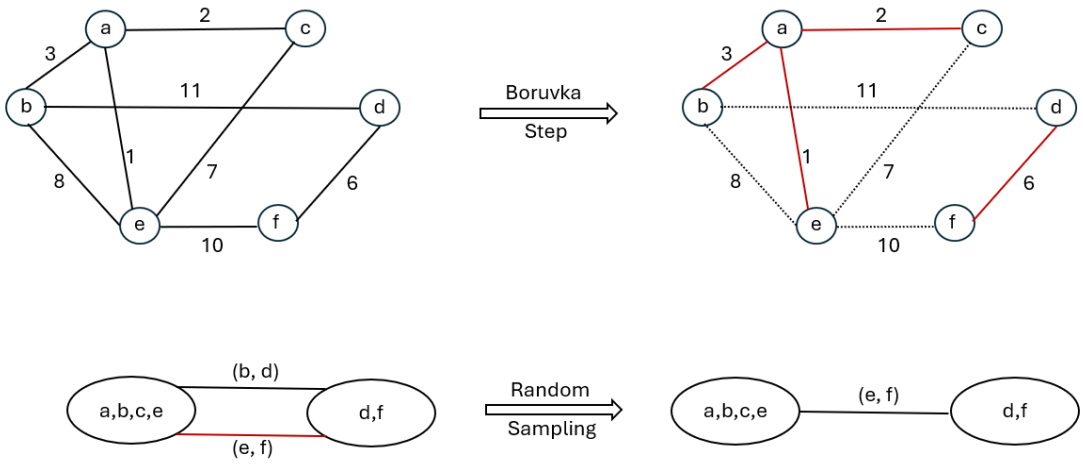
- Algoritam vraća uniju skupa grana koje su pronađene u Boruvkinom koraku i grana iz F_2. Ovaj skup grana čini MST inicijalnog grafa.

Dokaz korektnosti algoritma se može sprovesti korišćenjem matematičke indukcije po veličini grafa.

Induktivna hipoteza: Pretpostavimo da KKT algoritam korektno vraća MST za graf G' koji je strogo manjih dimenzija od grafa G.

Baza indukcije: Ukoliko graf G sadrži samo jedan čvor ili ne sadrži nijednu granu, na osnovu induktivne hipoteze, KKT algoritam vraća MST grafa G.

Induktivni korak: Na osnovu svojstva preseka važi da sve grane označene u prvom koraku (Boruvkin algoritam) pripadaju trenutnoj minimalnoj povezujućoj šumi grafa. Boruvkina faza vrši kontrakciju čvorova i vraća graf G' koji je manjih dimenzija od inicijalnog grafa G. Prema svojstvu ciklusa važi da nijedna teška grana ne ulazi u sastav MST originalnog grafa, tako da su sve uklonjene grane van MST.



Slika 2.6: Primer izvršavanja KKT algoritma

Na osnovu induktivne hipoteze se zaključuje da je MSF redukovanog grafa, odnosno grafa manje dimenzije, ispravno pronađena u svim rekurzuvnim pozivima, te konačno zaključujemo da algoritam vraća MST grafa.

Očekivana vremenska složenost: KKT algoritam se izvršava u očekivanom vremenu koje je linearno po veličini grafa, ali je dokaz ovog tvrđenja nešto komplikovaniji. Za potrebe dokaza biće korišćene leme date u nastavku:

Lema 2.5.1 *Očekivani broj nasumično odabranih grana jednak je polovini ukupnog broja grana.*

Lema 2.5.2 *Neka je G' podgraf grafa G koji je dobijen uključivanjem svake njegove grane sa verovatnoćom 0.5, a F minimalna povezujuća šuma grafa G'. Očekivani broj F-lakih grana u G nije veći od $2n$, gde je n broj čvorova grafa G.*

Lema 2.5.1 se matematički zapisuje kao $E[E_1] = \frac{|E|}{2}$. Ona se dokazuje na osnovu svojstva linearnosti matematičkog očekivanja.

Lema 2.5.2 se matematički zapisuje kao $E[E_2] \leq 2|V|$. Njen dokaz je nešto komplikovaniji i sledi u nastavku. Jasno je da se podgraf G' i njegova minimalna povezujuća šuma F konstruišu istovremeno. Za tu konstrukciju se koristi jedna varijanta Kruskalovog algoritma. Nakon utvrđivanja da li je grana F-teska ili F-laka, grana se sa verovatnoćom 0.5 dodaje u G', što se može interpretirati kao bacanje novčića.

Ukoliko je grana dodata u G' i proglašena F-lakom granom, ona se dodaje i u minimalnu povezujuću šumu F. Ovako dobijena šuma jeste minimalna povezujuća jer je dobijena primenom Kruskalovog algoritma na graf G'. Grane koje su proglašene F-teškim u trenutku obrade, ostaju teške do kraja jer se grane naknadno ne uklanjaju iz F. Slično važi i za F-lake grane, one koje su proglašene F-lakim u trenutku obrade, ostaju F-lake, jer sve grane koje su dodate posle njih imaju veću težinu. Jasno je da je prilikom bacanja novčića poznato da li je, trenutno razmatrana, grana F-teška ili F-laka. Pritom, ishodi bacanja za F-teške grane nisu relevanti jer te grane svakako neće biti dodate u F. Cilj je konstruisati MSF, a najveći mogući broj grana koji može biti dodat je $|V| - 1$, drugim rečima, potrebno je $|V| - 1$ uspešnih ishoda bacanja novčica u slučaju kada je trenutna grana laka. Ako bismo nastavili da bacamo novčić sve dok ne dobijemo željeni broj uspeha, gde je ukupan broj pozitivnih ishoda određen promenljivom Y, zaključujemo da Y predstavlja slučajnu promenljivu sa negativnom binomnom raspodelom, gde je ukupan broj bacanja jednak $|V|$, a verovatnoća uspeha je $p = 0.5$. Matematičko očekivanje slučajne promenljive sa ovakvom raspodelom je poznato i iznosi $2n$ [7].

Ostalo je još pokazati da je očekivano vreme izvršavanja ovog nedeterminističkog algoritma linearno i iznosi $O(|E| + |V|)$. Označimo očekivano vreme izvršavanja algoritma za graf G sa T_G. Najgore vreme izvršavanja nad svim grafovima G sa n čvorova i m grana se može predstaviti kao:

$$T_{m,n} := \max_{G=(V,E),\,|V|=n,\,|E|=m} \{T_G\}$$

Svi koraci, osim samog rekurzivnog poziva, imaju linearno vreme izvršavanja u funkciji broja čvorova i grana grafa, pa je dovoljno pokazati da se i vreme izvršavanja rekurzivnih poziva može odozgo ograničiti sa $c(n+m)$, gde je c konstanta. Neka je vreme izvršavanja algoritma u drugom i trećem koraku respektivno T_{F1} i T_{F2}. Tada važi:

$$T_G \leq c(m+n) + E[T_{F_1} + T_{F_2}] \leq c(m+n) + E[T_{m_1,n'} + T_{m_2,n'}]$$

Pretpostavićemo da važi: $T_G \leq 2c(m+n)$ za sve manje grafove, odnosno da je najgore vreme izvršavanja nad svim manjim grafovima linearno. U nastavku cemo

pokazati da to važi i za graf G [4].

$$\begin{aligned} T_G &\leq c(m+n) + E[2c(m_1+n')] + E[2c(m_2+n')] \\ &\leq c(m+n) + c(m'+2n') + 2c(2n'+n') \\ &= c(m+m'+n+8n') \\ &\leq 2c(m+n) \end{aligned}$$

Drugi red izvođenja se dobija primenom lema 2.5.1 i 2.5.2. Primenom nejednakosti $n' \leq n/8$ i $m' \leq m$ dolazimo do krajnje nejednakosti, čime se dokazuje da je najgore vreme izvršavanja za graf G linearno. Odavde sledi da se ovaj stohastički algoritam izvršava u linearnom vremenu.

Istaknimo i to da je ovo prvi algoritam za pronalaženje minimalnog povezujućeg drveta koji je uspešno paralelizovan tako da mu ukupno vreme izvršavanja bude linearno [3].

Na slici 2.5 je dat primer izvršavanja KKT algoritma. Prvi korak čini redukcija broja čvorova koja se vrši pomoću Boruvkinog algoritma. Nakon prvog koraka dobija se graf koji se sastoji od dve nepovezane komponente: $H_1 = \{a, b, c, e\}$ i $H_2 = \{d, f\}$. One nastaju tako što se za svaki čvor inicijalnog grafa odabere njemu incidenta grana najmanje težine. Grane (b, e) i (c, e) se eliminišu iz daljeg razmatranja pošto grade ciklus unutar komponente H_1. Nakon uspešne redukcije grafa, sledeći korak je nasumično uzorkovanje grana sa verovatnoćom 0.5. U ovom primeru su preostale samo dve grane: (b, d) i (e, f), a odabrana je grana (e, f). Algoritam se rekurzivno poziva nad ovom granom i ona se uključuje u krajnji rezultat. Treći korak je pronaći i eliminasati sve teške grane. Jedina takva je grana (b, d) i ona se eliminiše jer zatvara ciklus i predstavlja najtežu granu u njemu. Rezultat ovog postupka je unija grana dobijena u prvom koraku - (a, b), (a, c), (a, e), (d, f) i grane (e, f). Postupak izvršavanja algoritma je ovde ilustrovan na primeru veoma jednostavnog grafa, iako se u praksi on koristi za pronalaženje MST u znatno većim grafovima.

2.6 Algoritam Fredmana i Tarjana

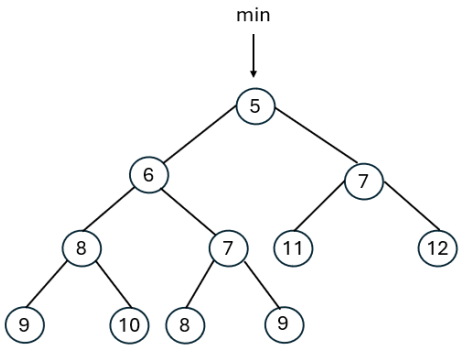
Fibonačijev hip

Fibonačijev hip (eng. *Fibonacci heap*) je struktura podataka koja se koristi u implementaciji reda sa prioritetom.

Postoje brojni primeri gde je ovakva struktura veoma korisna, a jedan od njih jeste internet ruter. Pretpostavimo da veliki broj poruka stiže, ali nisu sve podjednako bitne i neophodno je najpre obraditi one sa većim prioritetom. Svaka poruka se prima na ulazu u ruter i dodeljuje joj se neka vrednost koja se naziva *ključ*. On predstavlja prioritet koji poruka ima, a poruke se sortitaju unutar hipa na osnovu svojih ključeva sa ciljem da se obezbedi brz pristup onoj sa najvećim prioritetom.

Red sa prioritetom se može implementirati na različite načine, međutim jedan koji je jednostavan i često korišćen je binarni hip (eng. *binary heap*). Primer je dat na slici 2.6. On se implementira kao binarno drvo tako da ostaju zadovoljena dva svojstva:

- Svaki nivo je maksimalno popunjen i nivoi se popunjavaju sleva nadesno. Zahvaljujući ovom slojstvu, visina binarnog drveta zavisi logarimski od broja čvorova.
- Vrednosti u čvorovima dece su manje ili jednake vrednosti u čvoru roditelja.

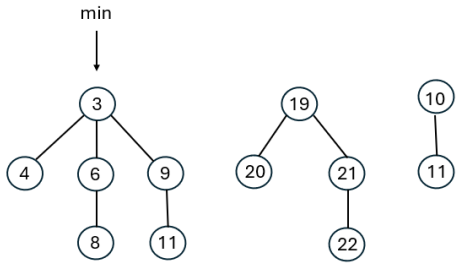


Slika 2.7: Primer binarnog hipa

Za pronalaženje željenih vrednosti u drvetu se koristi heš tabela, pa je taj proces veoma efikasan. Binarni hip podržava sledeće operacije:

- Pristupanje minimalnom elementu (eng. *GetMin*) - $O(1)$
- Umetanje elementa u hip (eng. *Insert*) - $O(\log n)$
- Brisanje elementa iz hipa (eng. *ExtractMin*) - $O(\log n)$
- Ažuriranje vrednosti elementa na hipu (eng. *Decrease key*) - $O(\log n)$

Fibonačijev hip je optimizovana verzija binarnog. On bi trebalo da obezbedi efikasan pristup minimalnom elementu, ali je neophodno i da ostale operacije budu što je moguće manje vremenski zahtevne. Ovo se može postići korišćenjem manje striktne strukture drveta, tako da kod ovako implementiranog hipa, prvo svojstvo koje se odnosi na strukturu, više ne mora da važi. Iako se drvo i dalje popunjava sleva nadesno, drvo ne mora biti binarno. Štaviše, hip se ne sastoji od jednog drveta, sada ih može biti više, primer je dat na slici 2.6. Ovime se sprečava da jedno drvo poraste previse i time naruši ukupnu složenost, ali se istovremeno kontroliše i broj drveta u hipu [5]. Glavna ideja na kojoj je zasnovana sama implementacija ovakve strukture podataka jeste da se operacije izvode *„lenjo“*. U nastavku sledi ugrubo obajsnjenje.



Slika 2.8: Primer Fibonačijevog hipa

Pristup minimumu ima istu ideju kao i kod binarnog hipa. Održava se pokazivač na čvor u kome se nalazi najmanja vrednost. Pošto se u korenu svakog od drveta Fibonačijevog hipa nalazi najmanja vrednost, minimum je zapravo minimum među vrednostima korena svih drveta. Jasno je da je za ovakav pristup potrebno $O(1)$ vremena.

Operacija ubacivanja se izvodi dosta jednostavnije u odnosu na binarni hip. Čvor se ubacuje u hip tako što se dodaje kao novo drvo koje se sastoji samo iz jednog jedinog čvora. Vreme koje se troši na ovu operaciju je takođe $O(1)$, ali se broj drveta u hipu uvećava za jedan.

Operacija uklanjanja sa hipa zahteva najpre da se deca podstabla odvoje od roditeljskog čvora i oni postaju stabla za sebe, ali ova operacija zahteva i dodatno vreme za tzv. „čišćenje“. Broj drveta se povećava za stepen[2] drveta, koji je ograničen nekim prirodnim brojem. Kako broj drveta ne bi previse porastao, potrebno je

[2]Stepen drveta označava maksimalni broj dece koji svaki čvor drveta može da ima.

da se stabla sa istim stepenom spajaju, što oduzima dodatno vreme. Pojedinačno izvođenje ove operacije je skupo, ali se deo posla može pripisati operaciji ubacivanja, pošto je ona sama po sebi veoma brza, čime se dobija znatno bolja amortizovana složenost. Detaljnijom analizom se zaključuje da amortizovana vremenska složenost zavisi samo od stepena drveta. Međutim, zahvaljujući svojstvu binomialnih drveta (eng. *binomial trees*) [5], dobija se da je složenost ove operacije $O(\log n)$, jer stepen drveta logarimaski zavisi od broja čvorova.

Ažuriranje vrednosti se veoma jednostavno može izvesti u logaritamskom vremenu. Međutim, moguće je svesti ažuriranje na konstantno vreme, ali zahteva najkompleksniju analizu do sada i ovde se koristi svojstvo Fibonačijevih brojeva. Naime, uvodi se ograničenje da svaki čvor može ostati bez najvise jednog deteta, jer se u suprotnom gubi svojstvo binomialnih drveta i time se narušava složenost operacije uklanjanja. Iako se ovime delimično kvari struktura binomialnih drveta, važi svojstvo koje je malo slabije, ali i dalje obezbeđuje logaritamsku složenost. To je svojstvo Fibonačijevih brojeva. Iako je ova operacija sporija kada se izvodi pojedinačno, ona u proseku daje konstantno vreme, tako da je amortizovana složenost i ove operacije iznosi $O(1)$.

Algoritam Fredmana i Tarjana

Sledeći u nizu algoritama za pronalaženje minimalnog povezujućeg drveta grafa je **algoritam Fredmana i Tarjana** iz 1987. godine. On se zasniva na, već dobro poznatom, Primovom algoritmu, uz nekoliko izmena u cilju postizanja optimizacije koja doprinosi boljoj ukupnoj vremenskoj složenosti. Ključna izmena u odnosu na Primov algoritam je korišćenje Fibonačijevog hipa, umesto binarnog, za implementaciju reda sa prioritetom koji se koristi za čuvanje susednih grana i brz pristup grani minimalne težine. Implementacija binarnog hipa zahteva $O(n \log n + m \log n)$ vremena, dok je vremenska složenost Fibonačijevog hipa $O(n \log n + m)$, tako da samo uvođenje Fibonačijevog hipa već dovodi do poboljšanja [4].

Ideja je da se na neki način logaritamski faktor smanji, kako bi ukupna složenost težila linearnoj. Ukoliko bi veličina hipa bila mala, postigla bi se manja cena izbacivanja elemenata iz hipa, što bi značajno uticalo na ukupno složenost algoritma. Primova verzija konstruiše samo jedno drvo tokom izvršavanja, čuvajući sve susede trenutnog drveta na hipu, tako da je dimenzija hipa ograničena brojem čvorova u grafu. Zaustavlja se kada se svi čvorovi dodaju. Optimizovana verzija koristi malo drugačiji pristup kako bi se veličina hipa održavala što manjom, odnosno, u zadatim

granicama. Iz ovako implementiranog hipa je moguće izbacivanje elementa u amortizovanoj složenosti $O(\log K)$, gde K predstavlja maksimalnu veličinu hipa, odnosno broja susednih čvorova trenutnom drvetu T_i.

Postavlja se pitanje kako održati veličinu hipa ispod nekog, unapred određenog, praga K, za koji ćemo najpre pretpostaviti da je fiksan. Fredman i Tarjan su primetili da je to moguće kontrolisanjem broja suseda trenutnog drveta na sledeći način: započeti izvršavanje Primovog algoritma i dodavati čvorove na hip sve dok njegova veličina ne dostigne zadati prag. Nakon toga, Primov algoritam se zaustavlja i bira se novi proizvoljani čvor iz kog se opisani postupak započinje iznova i to se ponavlja sve dok se svi čvorovi inicijalnog grafa ne dodaju u tekuću **šumu**. Čvorovi koji pripadaju novonastalom drvetu T_i kontrahuju u novi čvor, što je slična ideja kao kod Boruvkinog algortima. Potom se rekurzivno ova procedura poziva nad ostatkom grafa. Drugim rečima, ova procedura se može opisati kao niz usastopnih iteracija izvršavanja, pri čemu se svaka iteracija može razložiti na tri koraka:

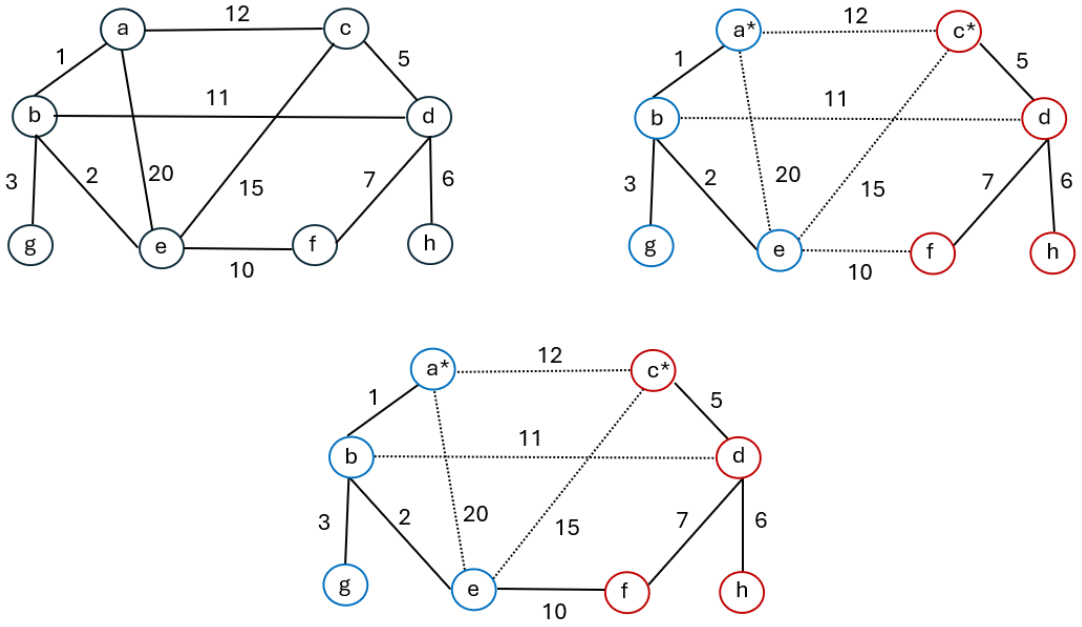
- Inicijalno su svi čvorovi neoznačeni. Nasumično se bira jedan neoznačen čvor i iz njega se pokreće Primov algoritam za konstrukciju MST. U Fibonačijevom hipu se čuvaju susedni čvorovi trenutnog drveta i time se obezbeđuje efikasan pristup najbližem od njih. Svaki čvor v za koji postoji grana $e(u, v)$, tako da $u \in T$, dok $v \notin T$ se naziva susedni čvor. Susedni čvorovi mogu biti već označeni u prethodnim iteracijama.

- Ukoliko u toku izvršavanja prethodnog koraka veličina hipa premaši unapred zadatu granicu ili se kao susedni, na hip, doda neki već ranije označeni čvor, postupak se zaustavlja. Svi čvorovi koji su dodati u tekuće drvo se označavaju i prethodni korak se ponavlja.

- Ukoliko su svi čvorovi dodati u tekuću šumu, nakon kontrahovanja čvorova, sledi rekurzivno izvršavanje na novonastalom grafu. [11]

Jasno je da je ovako opisan algoritam zadovoljava svojstvo korentnosti s obzirom da se u njegovoj osnovi nalazi Primov algoritam za koji je ranije dat dokaz. Tako nastale komponente povezanosti se dalje spajaju u rekurzivnim pozivima pomoću najlakših susednih grana, koje se skidaju sa hipa.

U nastavku sledi analiza složenosti izvršavanja algoritma. Intuitivno se može zaključiti kolika je složenost izvršavanja jedne iteracije algoritma. Prilikom konstrukcije drveta, svaka grana se razmatra najviše dva puta, po jednom sa oba kraja. Ako



Slika 2.9: Primer izvršavanja Fredman Tarjan algortima

označimo broj grana inicijalnog grafa sa m, faktor koji sigurno učestvuje je $O(m)$. Prilikom konstrukcije drveta dolazi do redukovanja broja čvorova, dok ne ostane jedna jedina komponenta koja predstavlja MST inicijalnog grafa i ovakvih spajanja je najvise n, koliko i čvorova inicijalnog drveta, dok sam pristup hipu, odnosno operacija uklanjanja elementa sa hipa, ima složenost $O(\log K)$. Iz svega navedog se može zaključiti da složenost jedne iteracije iznosi $O(m + n \log K)$. Jasno je da K predstavlja značajan faktor i ispostavlja se da on ne mora biti fiksirana vrednost, već se može uvesti kao parametar čija se vrednost menja iz iteracije u iteraciju. Za svaki označeni čvor koji se dodaje na trenutno drvo T_i, suma stepena svih čvorova koji pripadaju tom drvetu iznosi najmanje K. Čvor se označava u slučaju da njegovim dodavanjem veličina hipa premašuje K ili ukoliko je već bio označen, što znači da već pripada nekom drvetu (ono je već dostiglo maksimalni broj suseda) i tada dolazi do spajanja komponenti čime suma stepena svih čvorova postaje još veća. Ako se broj nastalih drveta u jednoj iteraciji označi sa l dolazi se do sledeće jednakosti:

$$2m = \sum_{v} d_v = \sum_{i=1}^{l} \sum_{v \in C_i} d_v \geq \sum_{i=1}^{l} K \geq Kl$$

Suma stepena svih čvorova u grafu jednaka je dvostrukom broju grana. Iz ovako

dobijene jednakosti se može izraziti broj nastalih drveta l u funkciji parametra K. Odnosno, broj nastalih drveta u iteracijama algoritma iznosi $l \leq \frac{2m}{K}$.

Broj nastalih komponenti zavisi od parametra K, a kako je poznato da se kao rezultat poslednje iteracije konstruiše MST inicijalnog grafa, odnosno, jedna jedina komponenta koja sadrži sve čvorove, intuitivno je jasno da se broj čvorova kroz iteracije smanjuje. Iz tog razloga se K može birati tako da raste kroz iteracije. Ideja je odabrati K na pogodan način tako da složenost ostane u okvirima linearne po broju grana. Ako se uzme da je u $i-toj$ iteraciji $K_i = 2^{\frac{2m}{n_i}}$, dolazimo do sledeće jednakosti [11]:

$$O(m_i + n_i \log K_i) = O\left(m_i + n_i \cdot \frac{2m}{n_i}\right) = O(m)$$

Ovako odabrano K neće pokvariti linearnu složenost, a dodatno, iz predhodne nejednakosti, koristeći da je $l \leq \frac{2m}{K_i}$, gde je l zapravo broj nastalih komponenti na kraju $i-te$ iteracije, odnosno one koje će se koristiti u narednoj, može se zaključiti sledeće:

$$K_i \leq \frac{2m}{n_{i+1}} = \log K_{i+1} \;\Rightarrow\; K_{i+1} \geq 2^{K_i}$$

Poslednja nejednakost pokazuje da K ima eksponencijalni rast kroz iteracije, dok n istim tim trendom opada. Zapravo, pojavljivanje višestrukih eksponenata ukazuje da je rast brži čak i od eksponencijalnog, tzv. *tetracioni rast*. Broj potrebnih iteracija da K dostigne vrednost n je $\beta(m,n)$, što je funkcija koja izuzetno sporo raste. Tada Primov algoritam može da sačuva sve preostale čvorove na hipu, bez da veličina hipa premaši granicu K i formira se minimalno povezujuće drvo inicijalnog grafa. **Vremenska složenost** algoritma iznosi $O(m\beta(m,n))$.

Primer izvršavanja algortima dat je na slici 2.6, za veličinu hipa $K = 2$. Izvršavanje počinje iz čvora a i on se na početku dodaje na hip. Pošto su grane najmanje cene susedne trenutnom drvetu T_1 $(a,b),(b,e)$ i (b,g), čvorovi a,b,g i e se dodaju na hip, čime se premašuje zadata veličina hipa. Oni se označavaju, hip se prazni i procedura počinje iznova. Na slici su svi čvorovi koji su dodati u prvoj iteraciji obeleženi plavom bojom i oni formiraju drvo T_1. Čvor u kome počinje druga iteracija je c. On se dodaje na hip, a potom i najbliže susedne grane $(c,d),(d,f)$ i (d,h), odnosno čvorovi c,d,f i h. Ovime se završava druga iteracija, dodati čvorovi se označavaju i hip se ponovo prazni. Na slici su ovo čvorovi označeni crvenom bojom i oni formiraju drvo T_2. Svi čvorovi su dodati u stablo i sada se stabla sažimaju u nove čvorove i procedura

se poziva rekurzivno na novom grafu koji se sastoji iz dva nova čvora i preostaih grana. Bira se ona sa najmanjom cenom i time se algortiam zaustavlja. Dobijeno minimalno povezujuće drvo čine grane: $(a, b), (b, e), (b, g), (c, d), (d, f), (d, h)$ i (e, f).

Gabov algoritam paketa

Algortitam Fredmana i Tarjana predstavlja značajno unapređenu verziju Primovog algoritma za nalaženje minimalnog povezujućeg drveta grafa. Međutim, u ovom poglavlju se odlazi korak dalje i izlaže se način na koji se Primov algoritam može dodatno optimizovati. Iako prethodno razmatrana verzija ima značajno bolju vremensku složenost zahvaljujući korišćenju Fibonačijevog hipa, operacija ažuriranja minimalnih rastojanja, koja se čuvaju na hipu, i dalje predstavlja najskuplju operaciju, odnosno tzv. *„usko grlo“* Fredman-Tarjan-ovog algoritma.

Njegova logika je takva da se svaka grana, kao potencijalni kandidat za dodavanje na trenutnu šumu, čuva na hipu, dok se ne premaši granica veličine hipa. Tada se sve sačuvane grane odbacuju i proces kreće iznova. Kako se neki kandidati tokom itecija veći broj puta odbacuju, može se desiti da neke od njih dođu na red tek u poslednjoj iteraciji, odnosno dodaju se čak $\beta(m, n)$ puta. U najgorem slučaju takvih grana može biti m, što nije zanemarljiv faktor u izrazu ukupne vremenske složenosti algoritma. Sve ove grane se stavljaju na hip i uklanjaju sa njega iz istog razloga, da bi se pronašla ona koja ima najmanju cenu među svim susednim granama tekuće šume. Na ovom mestu se javlja ideja da je moguće nekako optimizovati ove upite, tako da se ne moraju sve grane svaki put iznova brisati sa hipa kada se njegova veličina premaši, a potom dodavati da kako bi se sačuvala najlakša među njma.

Iako je tačno da je operacija dodavanja na hip veoma efikasna, odnosno konstanta, ne sme se zaboraviti da se to vreme potroši na svaki dodatu granu, a zatim se kao korisna uzima samo jedna, ona koja je minimalne cene, pa da se vreme uloženo u sve ostale koje nisu minimalne, smatra suvišno potrošenim. Prevazilaženjem limita veličine hipa se sve grane uklanjaju sa hipa, pa tada i vreme potrošeno na onu jednu koja je bila značajna, postaje uzalud potrošeno. Upravo se ovde javlja ideja da je potencijalno pametniji način čuvati samo minimum, a da se ostale grane, koje nisu u tom trenutku značajne, obrađuju kasnije. Odnosno, Gabov se dosetio da je put koji vodi optimizaciji upravo čuvanje samo grane najmanje cene na hipu, dok sve ostale, koje su teže od nje, ne moraju biti trenutno „vidljive“. Ovaj način optimizacije će biti detaljnije opisan u nastavku i predstavlja primer **dinamičkog programiranja**

i obezbeđuje složenost $O(m \log(\beta(m,n)))$.

Ovo je moguće ostvariti tako što se m grana rasporedi u m/p $p \in \mathbb{N}$ paketa. Grane incidentne jednom čvoru ili klasteru se stavljaju u isti paket, a svaki paket je sortiran po težinama grana. Da ovo sortiranje ne bi značajno uticalo na složenost, p bi trebalo da bude što manje, a složenost takvog sortiranja iznosi $O(m \log p)$. Sortiranje obezbeđuje da je početni element svakog paketa ujedno i grana minimalne cene, tako da je paket njome određen. Ostatak algoritma ostaje nepromenjen, osim što sada lista susedstva čvorova ne sadrži niz čvorova, vec niz paketa, a na hipu se čuvaju paketi koju su određeni svojim minimalnim elementima. Sve operacije na Fibonačijevim hipom i dalje ostaju, samo se više ne izvršavaju nad granama, nego nad paketima. Spajanjem drveta tekuće šume spajaju se i liste njihovih paketa, a novi minimum se određuje ka minumum minimuma svih paketa koji se trenutno nalaze na hipu.

Analiza složenosti:

- Inicijalno se grane particionišu u pakete i sortiraju, kao što je već objašnjeno ranije. Za to je potrebno $O(m + m \log p)$ vremena.
- U svakoj od iteracija, kojih ima $\beta(m,n)$, potrebno je $O(n_i \log k_i)$ vremena za skidanje elemenata sa hipa i $O(2m/p)$ za stavljanje svih paketa na hip. Ne treba zaboraviti da se neke grane brišu sa hipa, a samim tim i iz svojih paketa, ali se dimenzija paketa tokom vremena ne menja.
- Brisanja sa hipa može biti najvise $O(2m)$ jer se svaka grana može da naći u dva paketa, posmatrano sa oba kraja po jednom.
- Pošto svaki rekurzivni poziv sledi nakon brisanja elemenata sa hipa, takvih poziva je najvise $O(2m)$. Dodatno vreme za jedan rekurzivni poziv je konstantno.

Na osnovu prethodne analize sledi da ukupna vremenska složenost iznosi:

$$O\left(m + m \log p + \sum_{i=0}^{maxIter} (n_i \log k_i + 2m/p) + 2m\right)$$

Cilj je postići da vreme izvršavanja jedne iteracije bude $O(m/p)$, umesto $O(m)$. Ovo je moguće adekvatnim odabirom k_i tako da važi $n_i \log k_i = O(m/p)$, a ako se još dodatno odabere $p = \beta(m,n)$ postiže se sledeće:

$$O\left(m + m\log p + \sum_{i=0}^{p-1}(m/p) + 2m\right) = O(m\log p) = O\left(m\log\beta(m,n)\right)$$

Pogodno je uzeti da je $k = 2^{\frac{2m}{pn_i}}$, umesto $k = 2^{\frac{2m}{n_i}}$, jer u tom slučaju potrebno vreme za $n_i \log k_i$ iznosi $O(m/p)$, pa ukupno vreme koje se troši na sumu ostaje linearno.

Kroz analizu prethodnog algortima dolazi se do važnog zaključka. Naime, od ukupnog broja grana, što je $2m$, bar k_i njih biva dodato na trenutno drvo. Stoga, sledi da je takvih drveta najviše $2m/k_i$. Potrebno je uveriti se da ovo i dalje važi, uprkos manjim izmenama algortima. Zapravo, sada bar k_i paketa, od ukupno $2m/p$ njih biva dodato na trenutno drvo, tako da je na kraju konstruisano $2m/pk_i$ drveta što je čak i manji broj i ostaje da važi da k eksponencijalno raste, tako da je broj iteracija ograničen beta funkcijom [11].

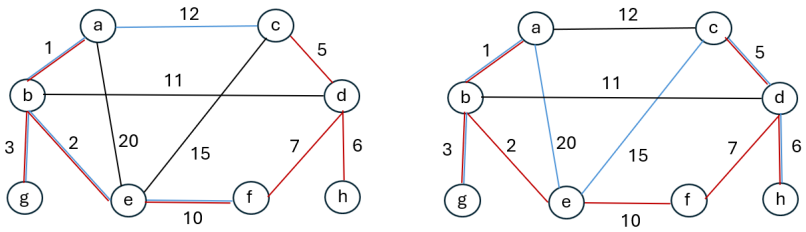
Još jedno bitno zapažanje jeste da se veliki deo vremena troši na inicijalno sortiranje elemenata unutar paketa. Međutim, zahvaljujući ovako sortiranim paketima, preostalo potrošeno vreme je linearno - $O(m/p)$. Ovo nikako ne znači da, pošto se kod prethodne implementacije sve grane odbacuju i ponovo dodaju na hip $\beta(m,n)$ puta, da se sada svaka grana razmatra tačno jednom. Logika je nepromenjena, i dalje je moguće odbaciti grane i razmatrati ih ponovo, ali se sada to radi samo sa predstavnicima paketa, dok se razmatranje odstalih odlaže za kasnije. Tako da se u svakoj iteraciji razmatra najvise $2m/p$ elemenata, a ukupno vreme je baš $O(m)$.

2.7 Algoritam Chazelle-a

Poslednji algoritam za određivanje minimalnog povezujućeg drveta grafa, koji će biti obrađen u ovom radu, objavio je Bernard Chazelle 1999. godine. Ovo je deterministički algoritam najbolje teorijske složenosti, od svih do sada obrađenih, koja iznosi $O(m\,\alpha(m,n))$, gde n predstavlja broj čvorova, m broj grana, a α funkciju koja izuzetno sporo raste, tj. inverz Akermanove funkcije.

Za razliku od prethodno analiziranih algoritama, koji su dosta intuitivniji, ovaj algoritam se zasniva na nešto složenijoj ideji i potpuno drugačijem pristupu. Jedan od ključnih koncepata jeste kontraktibilan (eng. *contractible*) podgraf. Za podraf C grafa G kažemo da je kontraktibilan, odnosno da se može kontrahovati u jedan jedini čvor, a da time ne utiče na odabir grana koje će biti dodate u $MST(G)$, ako

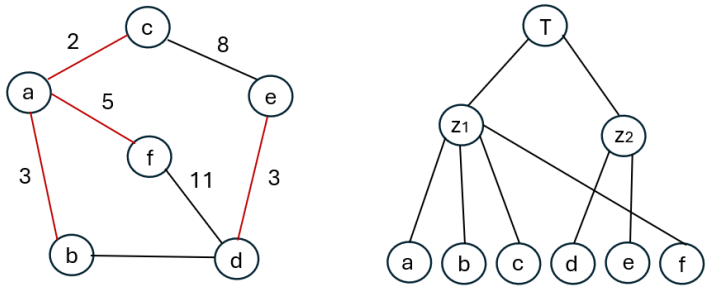
je njegov presek sa $MST(G)$ povezan. Primer je dat na slici 2.7 gde su crvenom bojom su označene grane koje pripadaju MST, dok su plavom označene grane podgrafa za koji se proverava da li je kontraktibilan. Ono što jedan ovakav podgraf čini značajnim jeste ideja da se $MST(G)$ može formirati pomoću $MST(C)$ i $MST(G')$, gde je G' graf koji nastaje od G kontrahovanjem podgrafa C. Detektovanje ovakvih podgrafa je bilo jednostavno u nekima od ranije obrađenih algoritama, ali se taj proces odvijao uporedo sa proširivanjem, odnosno formiranjem minimalnog povezujućeg drveta grafa. Sada se javlja ideja da se takvi podgrafi mogu pronaći ranije i da se kao takvi upotrebe pri konstrukciji MST. Podrazumeva se da taj proces mora biti veoma efikasan. Za potrebe rešenja ovog problema uvodi se nova strukturu podataka koja se zove soft hip (eng. *soft heap*).



Slika 2.10: Levo je primer podgrafa koji zadovoljava svojstvo kontraktibilnosti, dok podgraf koji se nalazi desno ne zadovoljava to svojstvo.

U nastavku paragrafa sledi ugrubo opis algoritma, dok će svi tehnički detalji biti objašnjeni nešto kasnije. Da bi se pronašlo $MST(G)$, ideja je da se najpre graf podeli na kontraktibilne podgrafe koji su međusobno disjunktni, a potom svaki od njih kontrahovati. Na taj način nastaje novi graf, koji se naziva minorni graf grafa G. Isti proces se može primeniti na novonastalom grafu i dalje se ponavljati sve dok ne ostane jedan jedini čvor koji predstavlja kontrakciju čitavog, inicijalnog, grafa G. Ovako nastala hijerarhija podgrafa tokom iteracija se može modelovati balansiranim stablom T u kome listovi odgovaraju originalnim čvorovima grafa G, dok se u korenu nalazi jedan čvor koje je zapravo kontrahovani graf G. Svaki unutrašnji čvor z ima neke potomke z_1, z_2, z_3 itd. i svaki od njih je jedan kontrahovani podgraf čvorova sa sledećeg nivoa (idući od korena nadole). Čvoru z odgovara kontrahovani podgraf C_z, čiji čvorovi su upavo potomci čvora z. Jasno je da svaki nivo stabla T predstavlja jedan minorni graf od G. Kada se završi konstrukcija drveta T, MST se računa

rekurzivno za svaki minorni graf C_z. Pošto u tom trenutku zasigurno znamo da svaki od C_z zadovoljava svojstvo kontraktibilnosti, jer se oni konstruišu da zadovoljavaju to svojstvo, $MST(G)$ se dobija spajanjem MST od prethodno izračunatih minora [2]. Na slici 2.7 dat je primer manjeg grafa (levo) i njegovog stabla (desno).



Slika 2.11: Primer konstrukcije drveta T

Drvo T se može opisati pomoću parametra d i broja čvorova n, gde je naročito značajan broj čvorova za svaku od komponenti n_z. Nakon dublje analize dolazi se do vremenske složenosti izgradnje drveta T od $O(m + d^3 n)$. U nastavku sledi malo detaljnija analiza i dokaz indukcijom. Posmatrajući ovu vremensku složenost, postavlja se pitanje kako odabrate visinu i broj čvorova svake komponente drveta, tako da izvršavanje ostane efikasno. Ukoliko se odabere mala visina drveta, konstrukcija stabla je izuzetno brza, ali su komponente na kojima se vrši rekurzija veće, imaju više čvorova, pa je vreme koje se troši na rekurziju veliko. Ukoliko se odabere velika visina drveta, konstrukcija drveta jeste značajnije sporija nego u prvom slučaju, ali su komponente čvorovi C_z značajno manje i rekurzija je mnogo brža. Ovde je potrebno naći kompromis kako bi ceo algoritam trošio razumno vreme na ceo proces izgradnje drveta i rekurziju. Spomenuti kompromis se postiže uvođenjem Akermanove funkcije, odnosno njenog inverza.

Izbor je sledeći: ako je $d_z = 1$ (za nivo na visini jedan, odnosno jedan iznad listova i definiše se kao broj grana na putu do listova), $n_z = S(t, 1)^3 = 8$, dok u slučaju kada je $d_z > 1$, $n_z = S(t - 1, S(t, d_z - 1))^3$, gde je t najmanji takav paramatar da važi $n \leq S(t, d)^3$, odnosno najmanji takav da ceo graf bude „smešten“ u korenu. Za d se uzima $d = c \left\lceil (m/n)^{1/3} \right\rceil$, za dovoljno veliku konstantu c. Funkcija S je definisana na sledeći način:

$$\begin{cases} S(1,j) = 2j, & \text{za svako } j > 0; \\ S(i,1) = 2, & \text{za svako } i > 0; \\ S(i,j) = S(i,j-1)\, S(i-1, S(i,j-1)), & \text{za svako } i,j > 1. \end{cases}$$

Ovakva funkcija obezbeđuje kontrolisani rast veličine n_z kako se približavamo korenu stabla T i upravo se zahvaljujući njoj dolazi do željene složenosti u kojoj figuriše funkcija α.

Indukcijom je lako pokazati da je broj temena, iz originalnog grafa G, koji završe u čvoru z, odnosno komponenti C_z, tačno $S(t,d_z)^3$: Broj čvorova u svakoj komponenti je već definisan kao $n_z = \big(S\big(t-1,\, S(t,d_z-1)\big)\big)^3$. Na osnovu definisane rekurzivne funkcije imamo:

$$S(t,d_z) = S(t,d_z-1) \cdot S\big(t-1,\, S(t,d_z-1)\big)$$

Stepenovanjem celog izraza na treći stepen dobija se:

$$S(t,d_z-1)^3 n_z = S(t,d_z-1)^3 \cdot \big(S\big(t-1,\, S(t,d_z-1)\big)\big)^3 = S(t,d_z)^3$$

Prethodno dokazana tvrdnja je tačna za svaki unutrašnji čvor z, pa samim tim i za koren stabla T. Ukoliko uzmemo da je $n = S(t,d)^3$, visina d odgovara ukupnoj visini drveta T. Moguće je jednostavno pokazati indukcijom po t da vreme potrebno za dobijanje minimalnog povezujućeg drveta originalnog grafa G iznosi $bt(m+d^3n)$. Baza indukcije je slučaj $t = 1$. Za induktivnu hipotezu se može uzeti slučaj kada broj čvorova trenutne komponente C_z iznosi $n_z = S(t-1, S(t,d_z-1))^3$ i vreme potrebno za nalaženje $MST(C_z)$ iznosi $b(t-1)(m_z + S(t,d_z-1)^3 n_z)$, za m_z grana komponente C_z. Sumiranjem po svakom z iz T dobijamo izraz:

$$b(t-1)\left(m + \sum_z S(t,d_z-1)^3\, S\big(t-1, S(t,d_z-1)\big)^3\right) = b(t-1)\left(m + \sum_z S(t,d_z)^3\right)$$

Svaki originalni čvor iz G se na svakom nivou prebroji tačno jednom. Ukupno ima d nivoa što nas dovodi do sledeće jednakosti:

$$= b(t-1)(m+dn)$$

Ovime se dobija ukupno vreme svih rekurzivnih poziva na nivou $t-1$. Dalje, kada se na ovo vreme koje se troši na rekurzivne pozive, doda vreme potrebno za konstrukciju drveta, za dovoljno veliko b dobijamo:

$$b(t-1)(m+dn) + O(m+d^3n) \leq bt(m+d^3n)$$

Indukcijom je pokazano koliko iznosi vreme potrebno na nivou t, pomoću $t-1$. Upravo zadavanjem parametra t preko Akermanove funkcije dobija se ukupna vremenska složenost od $O(m\alpha(m,n))$.

Pre detaljnije analize samog algoritma neophodno je uvesti pojam korumpiranih grana (eng. *corrupted edges*). Naime, tokom kreiranja stabla T cene nekih grana mogu biti uvećane i takve grane se nazivaju korumpirane grane. Cena se ne može promeniti bilo kojoj grani, već samo ukoliko je ona graničnog tipa. Do uvećanja cene dolazi usled uvođenje nove strukture podataka koja efikasno pronalazi minimum, ali uz određenu stopu greške dovodi do uvećanja vrednosti koje čuva, što su u našem slučaju cene grana. Detaljnije o ovoj strukturi i koje su to grane granične sledi u narednoj sekciji. Vratimo se na korumpirane grane. Nisu sve korumpirane grane problematične, već samo one koje su incidentne nekoj komponenti C_z koja je u tom trenutku kontrahovana u jedan jedini čvor. Takve grane nazivamo lošim (eng. *bad edges*). One se prilikom procesa kreiranja drveta T eliminišu iz razmatranja dok se ostale korumpirane smatraju prihvatljivim i prilikom svakom novog rekurzivnog poziva njihove cene se vraćaju na originalne. Dobra stvar je što je broj ovakvih grana ograničen.

Izvršavanje ovog algoritma se može opisati kroz nekoliko koraka:

- Izvršavanje počinje primenom nekoliko Boruvkinih faza. Kao rezultat se dobija skup grana koji ćemo označiti sa $\{Boruvka\}$.
- Kreiranje drveta T i konstruisanje grafa B koji se sastoji od tzv. loših grana.
- Za svaki unutrašnji čvor $z \in T$, rekurzivno rešiti problem manje dimenzije, ali bez loših grana $MST(C_z \setminus B)$. Uniranjem rezultata svih rekurzivnih poziva ovog oblika dobija se drvo F.
- Algoritam primeniti nad unijom prethodno dobijenog skupa grana F sa skupom loših grana B. Krajnji rezultat čini $MST(F \cup B) \cup \{Boruvka\}$.

Sledi detaljniji opis i analiza svakog od navedenih koraka.

Prvi korak

Jasno je da je ovo rekurzivan algoritam i samim tim se mogu se izdvojiti dva specijalna slučaja, kada je $n = O(1)$ i kada je $t = 1$. Oni su bazni slučajevi, odnosni

čine izlaze iz rekurzije. Prvi se odnosi na scenario kada ostane veoma mali broj čvorova i tada nije neophodno primenjivati ovaj složeni postupak rekurzivno, već je moguće izračunati MST direktno, korišćenjem nekog poznatog, jednostavnijeg algoritma, npr. Kruskalovog. Drugi slučaj se odnosi na scenario kada je parametar t, koji kontroliše dubinu rekurzije, jednak jedinici. U tom slučaju je u vremenu $O(n^2)$ moguće naći MST grafa samo uzastopnom primenom Boruvkinih faza dok na kraju ne ostane jedan jedini čvor. Što se tiče Boruvkinih faza, to su one iste faze koje su korišćene i u nekim od prethodni obrađenih algoritama. Ukratko, pronalazi se incidentra grana najmanje težine za svaki od čvorova. Nakon toga se povezani čvorovi kontrahuju u jedan, novi, čvor. Ovim postupkom se broj čvorova značajno smanjuje. U svakoj fazi se broj čvorova smanjuje bar upola, tako da se nakon c uzastiphin Boruvkinih faza, dobija novi graf G_0 u kome važi da je $n_0 \leq n/2^c$ i $m_0 \leq m$. Ova faza se izvršava u linearnom vremenu po broju grana i čvorova.

Drugi korak

U ovoj fazi algoritma veoma važan koncept predstavlja aktivna putanja (eng. *active path*). Neka je prilikom konstrukcije drveta T trenutni čvor z. On, kao i svaki unutrašnji čvor, nastaje kontrahovanjem čvorova sa nižeg nivoa, a u ovom slučaju neka to budu z_1, z_2, ... , z_k. Upravo oni čine spomenutu aktivnu putanju. Obrađivanjem jednog po jednog čvora, nastaju grafovi C_{z_1}, C_{z_2}, ... , C_{z_k}. Nakon sto je poslednji C_{z_k} konstruisan, on se kontrahuje u $C_{z_{k-1}}$. Tokom celog izvršavanja se održavaju dve invarijante:

- Za svako $i < k$ čuva se grana koja spaja te dve komponente C_{z_i} i $C_{z_{i+1}}$, odnosno uniju do sada obrađenih čvorova na aktivnoj putanji i narednog čvora $C_{z_{i+1}}$. Takva grana se zove lančana veze (eng. *chain-link*) i za nju važi da njena trenutna cena [3] na iznosi najviše kao cena neke granične grane[4]. Takođe, mora da važi da je njena radna cena manja od *radne cene*[5] bilo koje grane koja povezuje C_{z_i} i C_{z_j}, za $j \leq i$. Drugi uslov proveravamo zahvaljujući održavanju tzv. min-link veza koje su zapravo grane najmanje cene koje povezuju trenutnu

[3]Trenutna cena (eng. *current cost*) se uvodi jer prilikom izvršavanja algoritma može doći do povećanja cena u soft hipu.

[4]Granična grana (eng. *border edge*) je grana koja povezuje uniju do sada obrađenih komponenti na trenutno aktivnoj putanji sa ostatkom grafa. Drugim rečima, grana čiji se samo jedan kraj nalazi u obrađenom delu aktivne putanje, ona „izlazi“ napolje.

[5]Radna cena (eng. *working cost*) je trenutna cena ukoliko je cena grane izmenjena, odnosno originalna cena ukoliko nije.

komponentu sa nekom, već ranije obrađenom, komponentom. Iz svega navedenog sledi da se održavanjem ove invarijante obezbeđuje opadajući niz lančanih veza, što je ključni uslov za bezbedno kontrahovanje grafova. Bitno zapaženje je da se svakim odabirom lančane veze bira prvi čvor naredne komponente $C_{z_{i+1}}$. Kako komponenta raste, dodavanjem narednih čvorova, min-link grana može biti ažurirana na neku manju vrednost, tako se njena i cena lančane veze mogu znatno razlikovati.

- Za svako j granične ivice oblika (u, v), gde je $u \in C_{z_j}$, smeštaju se ili u soft hip sa oznakom $H(j)$ ili $H(i, j)$, gde je $i < j$. Svaka grana može pripadati tačno jednom hipu u datom trenutku. Hip oblika $H(j)$ se može posmatrati kao lokalni hip koji za svaku komponentu C_{z_j} čuva ivice koje vode ka čvorovima v koji su već povezani sa nekom ranijom komponentom. Dok hip oblika $H(i, j)$ čuva ivice tako da čvor v jeste sused komponente C_{z_i}, ali nije nijedne druge komponente C_{z_l} za $i < l < j$. Kreiranjem svih ovih hipova se održava druga invarijane i time se smanjuje efekat širenja „loših ivica“.

Obe invarijante se održavaju tokom celog procesa kreiranja drveta T koje se realizuje pomoću dve operacije: skraćivanje aktivne putanje (eng. *retraction*) i njeno proširivanje (eng. *extension*).

Operacija skraćivanja putanje

U trenutku kada i poslednji podgraf C_{z_k} sa aktivne putanje dostigne ciljanu veličinu, određenu brojem čvorova n_{z_k}, dolazi do retrakcije (skraćivanja putanje). Na taj način se kontroliše veličina komponenti i kraj aktive putanje se pomera na z_k. Kontrahovanjem komponente C_{z_k} u jedan jedini čvor, koga ćemo označiti sa a, komponenta postaje deo prethodno konstruisane komponente sa putanje $C_{z_{k-1}}$. Komponenta $C_{z_{k-1}}$ dobija novi čvor a, ali i nekoliko novih grana koje ga povezuju sa drugim čvorovima iz $C_{z_{k-1}}$. Među njima je i lančana grana koja je ranije povezivala komponente C_{z_k} i $C_{z_{k-1}}$ na aktivnoj putanji. Obe invarijante ostaju ispoštovane, iako za drugu to nije očigledno i zahteva detaljniju analizu. Ugrubo objašnjenje je da sada više nema smisla održavati hipove $H(k)$ i $H(k-1)$, već se oni uništavaju, a grane koje se nalaze u njima moraju biti adekvatno preraspoređene, osim loših, one se odbacuju. Sve one se prvo grupišu u klastere na osnovu spoljnog čvora, a zatim se iz svakog čuva samo ona koja ima najmanju cenu. Tada se grane raspoređuju u hipove. Ispostavlja se da ako grana dolazi iz $H(k)$ i deli čvor sa nekom granom iz $H(k-1, k)$ ili dolazi iz samog $H(k-1, k)$ može se smestiti u $H(k-1)$. Inače, grana

dolazi iz $H(k)$ i može se smestiti u $H(i,k)$ za $i < k-1$. Za svako $i < k-1$ hip $H(i,k)$ se može stopiti u $H(i,k-1)$.

Operacija proširivanja putanje

Potrebno je pronaći graničnu ivicu najmanje cene među svim granama koje se trenutno čuvaju u hipovima. Neka je to grana (u,v). Posmatrajući unazad komponente na aktivnoj putanji, traži se ona koja je najdalje na putanji, odnosno najranije dodata, a da ima radnu cenu min-link grane ne veću od cene $c(u,v)$. Odnosno, komponenta C_{z_i} za najmanje i koje ispunjava prethodno navedeni uslov. Ideja je da se prvo sve grane, čija se oba kraja nalaze posle komponente C_{z_i} na aktivnoj putanji, kontrahuju u jedan čvor, neka bude čvor b, a zatim se se izvrši kontrahovanje grane (a,b), gde je a kraj grane koji je incidentan sa komponentom C_{z_i}. Prethodno opisan postupak se naziva fuzija (eng. *fusion*) i ona je značajna kako bi prva invarijanta ostala ispunjena. Nakon toga je neophodno ažurirati sve min-link grane i ažurirati hipove koristeći istu logiku kao i u operaciji skraćivanja putanje, sa jednom bitnom razlikom. Ukoliko je došlo do fuzije, što ne mora uvek biti slučaj, potrebno je ažurirati ne samo $H(k)$ i $H(k-1,k)$, već i $H(i+1),...,H(k)$ kao i sve hipove oblika $H(j,j')$ za $i \leq j < j'$. Time druga invarijanta ostaje održana.

Sada kada je objašnjeno kako se realizuju operacije proširivanja i skraćivanja aktivne putanje, jasno je da se pomoću njih veoma jednostavno konstruiše stablo T. Ako je trenutni cvor z_k, aktivna putanja se proširuje sve dok je to moguće, odnosno dok komponenta C_{z_k} ne dostigne predviđenu veličinu. Takođe često se usput, po potrebi, dešava i fuzija. Dostizanjem predviđenog broja čvorova otpočinje operacija skraćivanje putanje.

Treći korak

Kao rezultat prethodnog koraka konstruisano je drvo T. Sada je potrebno rekurzivno pozvati algortitam za svaki njegov čvor z, odnosno za svaki podgraf C_z. Još u prethodnom koraku su odbačene neke grane, ali to su loše grane. Još uvek postoje neke kojima je hip izmenio vrednost, ali pošto nisu predstavljale opasnost pri kreiranju stabla, nisu odbačene. Sada je pre rekurzije potrebno vratiti originalne cene takvim granama. Tako da se algoritam zapravo poziva nad $C_z \setminus B$ i vrednost parametra t se takođe umanjuje što određuje visinu drveta koje se konstruiše. Rezultat unije ovakvih rekurzivnih poziva je skup F, skup ivica unutar podgrafa oblika C_z. Za svako z, jasno je da svi njegovi čvorovi predstavljaju kontrahovane grafove sa

nizeg nivoa, pa se zato algortiam rekuzivno poziva za svu njegovu decu. Međutim, postavlja se pitanje šta se dešava u slučaju da neki čvorovi predstavljaju rezultat fuzija. Situacija je zapravo ista jer svaki takav čvor predstavlja dete kao i svaki drugi unutar C_z, sa samo jednom razlikom, a to je da on ima svoju hierarhiju čvorova koja nastaje usled fuzije. Kako je prethodno objašnjeno, prilikom fuzija značajne su grane oblika (a, b) i one se zapravo dodaju u F, ali samo ukoliko nisu loše, u inače se njihova obrada ostavlja za neke naredne rekurzivne pozive.

Poslednji korak

Kao rezultat poslednjeg koraka dobija se finalni rezultat algoritma, odnosno, minimalno povezujuće drvo originalnog grafa G. U drugoj fazi je konstruisano drvo T i dobijen skup B što je skup grana koje se ne mogu odmah obraditi jer imaju izmenjene vrednosti cena i potencijalno narušavaju invarijante. U trećem koraku nastaje skup F, to je skup grana unutar kontrahovanih podgrafa koje sigurno pripadaju $MST(G)$. Sada, da bi se dobile ivice koje povezuju već obrađene komponente, algoritam se poziva za $F \cup B$. Rezultat ovog poziva, zajedno sa prethodno pronađenim ivicama u prvoj, Boruvkinoj, fazi, čini krajnji rezultat.

Korektnost

Za potrebe dokaza *korektnosti* ovog algoritma dovoljno je pokazati da važi sledeće: Ako grana originalnog grafa G nije proglašena lošom, niti je dodata u skup F, onda ona sigurno ne pripada $MST(G)$. U nastavku sledi skica dokaza kroz dva slučaja, posmatrajući neku granu e.

Prvi slučaj je da grana e, za koju važi da se njena oba kraja nalaze u nekom minoru C_i grafa G, nikada nije odbačena. Pošto ona nije u F, znači da nije deo lokalnog MST, odnosno ne pripada $MST(C_i)$. Pošto grana nije označena kao loša i nije u $MST(C_i)$, pa je na osnovu svojstva kompozicije[6] van $MST(G)$. Važno je prisetiti se da čim neka grana nije deo MST, to znači da ona gradi neki ciklus, a dodatno je i najskuplja grana u njemu. Ova analiza je izvršena u odnosu na radne cene grana, a kako grana e nije proglašena lošom, njena cena je zapravo sve vreme originalna. Ostalim granama se cene mogu vratiti na inicijalne, ali to znači da sada neke od njih mogu biti manje ili ostati iste, nikako veće. Ako je grana e

[6]Svojstvom kompozicije (eng. *composition property*) nazivamo svojstvo koje tvrdi da je $MST(G)$ unija svih $MST(G_i)$.

bila najskuplja u nekom ciklusu, ona to ostaje i nakon resetovanja cena na inicijalne vrednosti.

Drugi slučaj je da je $e(u, v)$ odbačena. Pre nego što je odbačena, postojala je grana $e'(u', v)$ manje cene, odnosno grana koja je delila čvor v sa granom e. Prilikom operacije skraćivanja putanje ili fuzije dolazi do kontrahovanja obe grane u jedan čvor. Pošto je taj čvor kontraktibilan u G, radne cene ovih grana se ponašaju kao originalne u odnosu na $MST(G)$. Nijedna od ove dve grane nema izmenjenu cenu, ali je e skuplja od e'. Za dve grane koje imaju zajednički čvor, sa originalnim cenama u G, važi da ona koja skuplja može biti izbačena i sigurno ne pripada $MST(G)$. Međutim, e' može promeniti vrednost u komponenti kojoj pripada i postati loša, tako da se ne može tvrditi da e ne pripada $MST(G \setminus B)$.

Vremenska složenost

Veoma je važno za analizu ukupne vremenske složenosti algoritma da je broj ivica, koje su obeležene kao loše prilikom konstrukcije drveta T, ograničen sa $|B| \leq m_0/2 + d^3 n_0$ [2]. Dokaz ove leme se može naći u literaturi i ona je ovde je navedena bez dokaza.

Dokaz složenosti ovog algoritma je dugačak i sadrži dosta tehničnih detalja, ali može se odrediti vremenska složenost svake faze algoritma i ispravnim odabirom t d dolazi se do ukupne vremenske složenosti od $O(m\alpha(m, n))$. Ona odgovara inverzu Akermanove funkcije, što je funkcija koja veoma brzo raste.

Bibliografija

[1] Otakar Boruvka. Otakar Borůvka on minimum spanning tree problem Translation of both the 1926 papers, comments, history, 1927. on-line at: https://www.sciencedirect.com/science/article/pii/S0012365X00002247.

[2] Bernard Chazelle. A minimum spanning tree algorithm with inverse-Ackermann type complexity, 2000. on-line at: https://mathscinet.ams.org/mathscinet/relay-station?mr=MR1866456.

[3] Richard Cole, Philip Klein, and Rober Tarjan. A linear-work parallel algorithm for finding minimum spanning trees, 1994. on-line at: chrome-extension://efaidnbmnnnibpcajpcglclefindmkaj/https://cs.brown.edu/research/pubs/pdfs/1994/Klein-1994-LWP.pdf.

[4] Jason Eisner. State-of-the-Art Algorithms for Minimum Spanning Trees, 1997. on-line at: www.cs.jhu.edu/~jason/papers/eisner.mst-tutorial.pdf.

[5] Michael Fredman and Bob Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms, 1987. on-line at: https://mathscinet.ams.org/mathscinet/relay-station?mr=MR0904195.

[6] Vojtěch Jarník. O jistém problému minimálním, 1930. on-line at: https://dml.cz/handle/10338.dmlcz/500725.

[7] David Karger, Philip Klein, and Rober Tarjan. A randomized linear-time algorithm to find minimum spanning trees, 1995. on-line at: https://cs.brown.edu/research/pubs/pdfs/1995/Karger-1995-RLT.pdf.

[8] Joseph Kruskal. On the shortest spanning subtree of a graph and the traveling salesman problem, 1956. on-line at: https://mathscinet.ams.org/mathscinet/relay-station?mr=MR0078686.

[9] Filip Marić and Vesna Marinković. Konstrukcija i analiza algoritama, 2024. on-line at: `https://poincare.matf.bg.ac.rs/~filip/kiaa/kiaa.pdf`.

[10] Jaroslav Nesetril and Helena Nesetrilova. The Origins of Minimal Spanning Tree Algorithms - Boruvka and Jarnik, 2010. on-line at: `https://www.cs.cmu.edu/~15850/notes/lec1.pdf`.

[11] CMU School of Computer Science. Minimum Spanning Trees, 2024. on-line at: `https://www.cs.cmu.edu/~15850/notes/lec1.pdf`.