UNIVERZITET U NOVOM SADU
PRIRODNO-MATEMATIČKI FAKULTET
DEPARTMAN ZA MATEMATIKU I INFORMATIKU

Evelin Kurta

Algoritmi za pronalaženje minimalnog pokrivajućeg stabla

Master rad

Mentor:
prof. dr Maja Pech

Novi Sad, 2016

Sadržaj

Uvod i istorija MPS **3**

1 O minimalnom pokrivajućem stablu iz ugla teorije grafova **5**
1.1 Osnovni pojmovi iz teorije grafova 5
1.2 Kako napraviti minimalno pokrivajuće stablo 9
1.2.1 T-teške i T-lake grane 9
1.2.2 Crveno i plavo pravilo odnosno pravilo konture i reza . 9
1.2.3 Postojanje i jedinstvenost MPS 12
1.3 Pretraživanje grafa: breadth-first search i depth-first search . . 13

2 Kompleksnost algoritama **17**
2.1 Tjuringova mašina 17
2.2 Vremenska kompleksnost algoritama 18
2.3 Notacija O 18
2.4 Klase vremenske kompleksnosti 19

3 Algoritmi za pronalaženje minimalnog pokrivajućeg stabla u polinomnom vremenu **20**
3.1 Borůvkin algoritam 20
3.1.1 Opis algoritma 20
3.1.2 Ispravnost algoritma 21
3.1.3 Kompleksnost algoritma 22
3.2 Primov algoritam 24
3.2.1 Opis algoritma 24
3.2.2 Ispravnost algoritma 25
3.2.3 Kompleksnost algoritma 25
3.3 Kruskalov algoritam 28
3.3.1 Opis algoritma 28
3.3.2 Ispravnost algoritma 29
3.3.3 Kompleksnost algoritma 30
3.4 Optimizovani Kruskalov algoritam 32

3.5 Verifikacija MPS 33

4 Empirijska analiza klasičnih algoritama 35
4.1 Kreiranje proizvoljnih grafova 35
4.2 Analiza Borůvkinog i Primovog algoritma 36
4.3 Analiza Kruskalovog algoritma 38

5 Algoritam za pronalaženje minimalnog pokrivajućeg stabla u linearnom vremenu 42
5.1 Nasumični algoritam 42
5.1.1 Opis algoritma 42
5.1.2 Ispravnost algoritma 46
5.1.3 Kompleksnost algoritma 47

6 Primena MPS na deviznom tržištu 49
6.1 Od deviznog kursa do grafa 49
6.2 MPS 52

Zaključak 54

Prilog 55

Literatura 61

Biografija 63

Uvod i istorija MPS

Problem pronalaženja minimalnog pokrivajućeg stabla jedan je od najpoznatijih problema u kombinatornoj optimizaciji. Posmatrani problem je interesantan jer ima široku primenu: u projektovanju mreža (telefonskih, električnih, računarskih), u klaster analizi (grupisanje tačaka u ravni, grupisanje podataka genske ekspresije), povezivanju ostrva, prepoznavanju rukopisa matematičkih izraza, kreiranju lavirinata, itd.

Minimalno pokrivajuće stablo (MPS) je pokrivajuće stablo datog grafa sa najmanjom težinom. Jedan graf može da ima nekoliko različitih MPS. Problem pronalaženja MPS za dati graf rešava se primenom različitih algoritama.

Češki matematičar Otakar Borůvka je 1926. godine definisao problem pronalaženja MPS kao problem izgradnje efikasne električne mreže u Moraviji u istočnoj Češkoj, i opisao je algoritam za rešavanje ovog problema. Drugi algoritam dao je 1930. godine Vojtěch Jarník, takođe češki matematičar. Isti algoritam nezavisno od Jarníka opisali su Robert Prim (1957.) i Edsger Dijkstra (1959.) pa je ovaj algoritam poznat i pod imenima DJP i Prim-Jarník. Mi ćemo ga u ovom radu zvati Primov algoritam. 1956. godine Joseph Kruskal dao je treći algoritam za rešavanje problema pronalaženja MPS. Ova tri klasična algoritma izvršavaju se u polinomnom vremenu. Neke od bržih algoritama razvili su Yao (1975.), Chariton i Tarjan (1976.), Fredman i Tarjan (1984.), Gabow, Galil, Spencer i Tarjan (1986.). 1995. godine Karger, Klein i Tarjan razvili su nasumični algoritam koji ima očekivano linearno vreme izvršavanja. 1997. Chazelle je opisao deterministički algoritam koji se izvršava u vremenu $O(m\alpha(m,n))$, gde je α inverzna Akermanova funkcija. Pettie i Ramachandran 1998. godine opisali su optimalni deterministički algoritam.

U prvoj glavi ovog rada dat je pregled osnovnih pojmova iz teorije grafova i jedan opšti postupak za građenje MPS. U drugoj glavi pažnja je posvećena vremenskoj kompleksnosti algoritama. U trećem delu rada opisana su tri klasična algoritma za pronalaženje MPS: Borůvkin, Primov i Kruskalov. Četvrta glava predstavlja empirijsku analizu prethodno opisanih algoritama a koji su implementirani u programskom paketu R. U petoj glavi prikazan

je nasumični algoritam čije je očekivano vreme izvršavanja linearno. Šesta, poslednja glava, opisuje primenu MPS na deviznom tržištu. Na kraju rada dat je kratak zaključak.

Glava 1

O minimalnom pokrivajućem stablu iz ugla teorije grafova

Ovo poglavlje predviđeno je za uvođenje neophodnih pojmova iz teorije grafova. Većina navedenih definicija i rezultata može se naći u [9].

1.1 Osnovni pojmovi iz teorije grafova

Definicija 1 ***Graf*** *je uređeni par $G = (V, E)$, gde je V neprazan konačan skup, a E proizvoljan podskup skupa $V^{(2)} = \{\{u, v\} \subseteq V : u \neq v\}$. Elemente skupa V zovemo* ***čvorovi*** *grafa G, dok elemente skupa E zovemo* ***grane*** *grafa G.*

Broj čvorova grafa označavaćemo sa n, a ukupan broj grana grafa sa m. Dakle, $n = |V|$, $m = |E|$.

Definicija 2 *Ako je $e = \{u, v\}$ grana grafa, tada kažemo da su čvorovi u i v* ***susedni*** *i da je grana e* ***incidentna*** *sa u i v. Takođe kažemo da je čvor u* ***sused*** *čvora v i da grana e* ***spaja*** *čvorove u i v.*

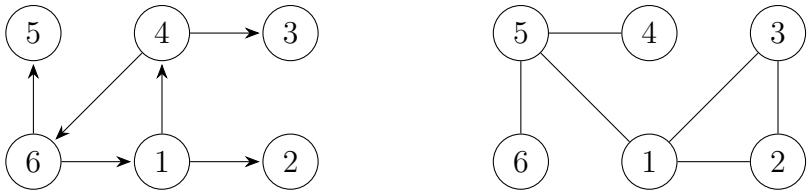
Graf možemo grafički predstaviti u ravni: čvorove predstavljamo tačkama ili krugovima, a grane linijama koje povezuju čvorove. Grane grafa mogu da budu *usmerene* i tada kažemo da je graf usmeren (asimetričan ili orijentisan) i *neusmerene* i tada je graf neusmeren (simetričan, neorijentisan). Ako je graf neusmeren tada su grane neuređeni parovi čvorova i grana $\{u, v\}$ je isto što i grana $\{v, u\}$. U tom slučaju grane crtamo kao linije bez strelica. U slučaju usmerenih grafova grane su uređeni parovi čvorova i crtamo ih kao linije sa strelicom usmerenom od prvog čvora do drugog, i zato je redosled u kojem navodimo čvorove bitan. Ako je grana (u, v) usmerena to znači da

postoji veza od u do v ali ne i da postoji veza od v do u. Na slici 1 prikazan je jedan usmeren i jedan neusmeren graf.

Nas će zanimati neusmereni grafovi, redosled u kojem navodimo čvorove nije bitan i zato ćemo granu $\{u, v\}$ označavati sa uv.

Definicija 3 *Skup svih čvorova grafa G koji su susedni čvoru v označavamo sa $N(v)$ i zovemo* ***susedstvo*** *čvora v.*

PRIMER 1 *Na Slici 1.1 prikazan je jedan usmeren i jedan neusmeren graf. U neusmerenom grafu na Slici 1.1b susedstvo čvora 1 čine čvorovi 2, 3 i 5.*



(a) Usmeren graf (b) Neusmeren graf

Slika 1.1: Primer 1

Definicija 4 *Graf $H = (W, E')$ je* ***podgraf*** *grafa $G = (V, E)$, pišemo $H \leqslant G$, ako je $W \subseteq V$ i $E' \subseteq E$. Graf H je* ***indukovani podgraf*** *grafa G ako je $E' = E \cap W^{(2)}$.*

Grane indukovanog podgrafa grafa G su sve grane grafa G čija oba kraja pripadaju skupu W.

Definicija 5 *Niz grana oblika $v_0v_1, v_1v_2, ..., v_{r-1}v_r$ grafa G, ili skraćeno $w = v_0v_1...v_{r-1}v_r$ zove se* ***šetnja*** *dužine r u grafu G. Čvor v_0 zove se* ***početni****, a čvor v_r* ***završni*** *čvor šetnje. Čvorovi $v_1, ..., v_{r-1}$ su* ***unutrašnji čvorovi*** *šetnje. Šetnja sa početnim čvorom v_0 i završnim čvorom v_r zove se $v_0...v_r$* ***šetnja*** *i kaže se da* ***spaja*** *čvorove v_0 i v_r.*

Definicija 6 *Ako su sve grane šetnje $v_0...v_r$ različite ona se zove* ***staza*** *dužine r. Ako su svi čvorovi šetnje (a samim tim i sve grane) različiti, tada se šetnja zove* ***put*** *dužine r.*

Definicija 7 *Ako su u i v čvorovi grafa G, dužina najkraćeg puta između njih zove se* ***rastojanje*** *između čvorova u i v i označava se sa $d(u, v)$.*

Definicija 8 *Šetnja ili staza je* ***zatvorena*** *ako je $v_0 = v_r$. Šetnja u kojoj su čvorovi $v_0, v_1, ..., v_r$ svi različiti sa izuzetkom početnog i krajnjeg čvora koji se poklapaju zove se* ***kontura****. Graf koji ne sadrži nijednu konturu je* ***acikličan****.*

Definicija 9 ***Put sa n čvorova****, u oznaci P_n, je graf kod koga je prvi čvor susedan sa drugim, drugi sa trećim, i tako dalje, pretposlednji susedan sa poslednjim, a poslednji nije susedan sa prvim. Kažemo da put sa n čvorova ima dužinu $n-1$.*

Put je šetnja kod koje se ne ponavljaju ni čvorovi ni grane.

Kontura je zatvorena šetnja kod koje se ne ponavljaju ni čvorovi ni grane, osim prvog čvora koji se pojavljuje na kraju šetnje.

Definicija 10 *Graf G je* ***povezan*** *ako između svaka dva njegova čvora postoji put. Ako postoje čvorovi koji se ne mogu povezati putem, graf je* ***nepovezan****.*

Nepovezan graf se sastoji od dva ili više odvojenih delova. Ovi odvojeni delovi nazivaju se **povezane komponente** grafa.

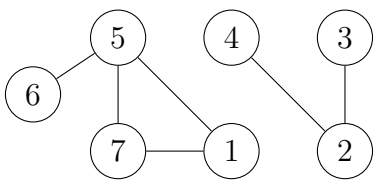
Karakterizacija stabala. Neka je T graf sa n čvorova. Sledeća tvrđenja su ekvivalentna:

1. T je stablo.
2. T je acikličan i ima $n-1$ granu.
3. T je povezan i ima $n-1$ granu.
4. Svaka dva čvora u T povezana su tačno jednim putem.
5. T nema kontura, ali dodavanjem bilo koje nove grane dobija se graf sa tačno jednom konturom.

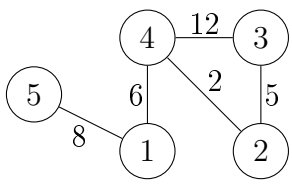
Definicija 11 ***Šuma*** *je acikličan graf, a povezana šuma naziva se* ***stablo****, u oznaci T.*

Povezane komponente šume su stabla.

Na slici 1.2 prikazan je jedan nepovezan graf koji se sastoji od dve povezane komponente.



Slika 1.2: Nepovezan graf



Slika 1.3: Povezan težinski graf

Definicija 12 *Stablo je* ***netrivijalno*** *ako ima više od jednog čvora.* ***Trivijalno stablo*** *sadrži samo jedan čvor.*

Definicija 13 ***Težinski graf*** *je graf u kome je svakoj grani dodeljen neki broj, tj. grafu $G = (V, E)$ pridruženo je preslikavanje $w : E \to \mathbb{R}$ koje svakoj grani $e \in E$ dodeljuje broj $w(e)$ kao težinu. Funkcija w se zove* ***težinska funkcija grafa****.* ***Težina grafa*** *je zbir težina svih grana grafa.*

U ovom radu bavićemo se isključivo težinskim grafovima. Težina grane može da predstavlja fizičku udaljenost, propusnu moć, kapacitet, cenu koštanja, itd.

Težinski graf G, tj. uređena trojka $G = (V, E, w)$ često se naziva i mreža. U praksi njima odgovaraju konkretne fizičke mreže kao što su telekomunikacione, računarske, saobraćajne i druge mreže.

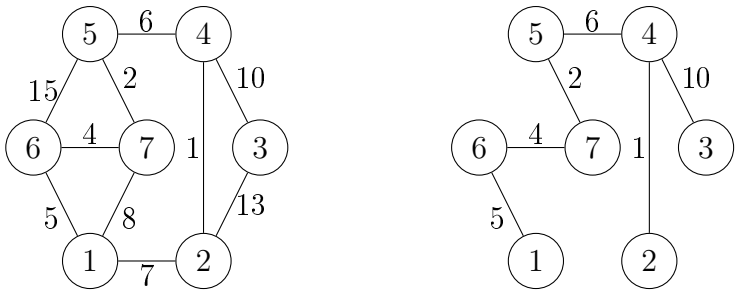
Uvešćemo oznaku d_{ij} za težinu grane $e = v_i v_j$ koja spaja i-ti i j-ti čvor, $d_{ij} = w(e)$. Od veličina d_{ij} možemo formirati kvadratnu matricu $D = (d_{ij})$ koju nazivamo težinskom matricom.

Definicija 14 ***Pokrivajuće stablo*** *T povezanog, neusmerenog grafa G je podgraf koji sadrži sve čvorove i neke od grana grafa G takve da je T stablo.* ***Minimalno pokrivajuće stablo*** *(MPS) je ono pokrivajuće stablo koje od svih mogućih pokrivajućih stabala datog težinskog grafa ima najmanju težinu.*

Tvrđenje 1 *[9] Graf je povezan ako i samo ako ima pokrivajuće stablo.*

Problem pronalaženja minimalnog pokrivajućeg stabla

Neka je $G = (V, E, w)$ povezan težinski neusmeren graf, gde je $w : E \to \mathbb{R}$ težinska funkcija. Podgraf $T = (V, E_T)$ grafa G koje je stablo je pokrivajuće stablo grafa G. Težina pokrivajućeg stabla T grafa G je $w(T) = \sum_{e \in E_T} w(e)$. Treba da pronađemo ono pokrivajuće stablo grafa $G = (V, E)$ koje ima najmanju težinu.



Slika 1.4: Graf i njegovo minimalno pokrivajuće stablo

1.2 Kako napraviti minimalno pokrivajuće stablo

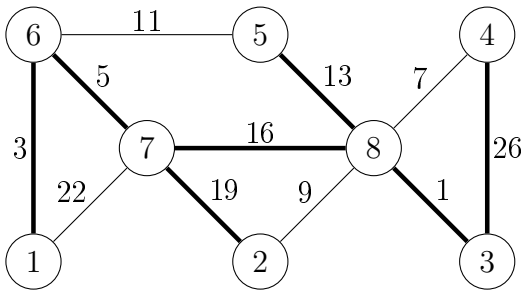
Postoji više pristupa problemu građenja minimalnog pokrivajućeg stabla u grafu, a većina njih se zasniva na pravilu konture i pravilu reza koji su poznati još i pod imenima crveno i plavo pravilo. Za početak ćemo uvesti pojmove T-teških i T-lakih grana a nakon toga ćemo opisati crveno i plavo pravilo.

1.2.1 T-teške i T-lake grane

Definicija 15 ($w_T(e)$) *Neka je $G = (V, E, w)$ težinski neusmeren graf i neka je T pokrivajuće stablo grafa G. Za granu $e = uv \in E$ sa $w_T(e)$ ćemo označavati težinu najteže grane na jedinstvenom putu u T koji povezuje u i v.*
Ako $e \in T$ onda je $w_T(e) = w(e)$.

Definicija 16 (T-teške i T-lake grane) *Neka je $G = (V, E, w)$ težinski neusmeren graf i neka je T pokrivajuće stablo grafa G. Kažemo da je grana $e \in E$* ***T-teška*** *ako i samo ako je $w(e) > w_T(e)$, a u suprotnom kažemo da je e* ***T-laka*** *grana.*

PRIMER 2 *U grafu na slici 1.5 grane koje pripadaju pokrivajućem stablu T prikazane su debljim linijama. Grana $e_1 = \{4, 8\}$ je T-laka jer je $w_T(e_1) = 26$ a $w(e_1) = 7$. Grana $e_2 = \{1, 7\}$ je T-teška jer je $w_T(e_2) = 5$ a $w(e_2) = 22$.*



Slika 1.5: Primer 2

1.2.2 Crveno i plavo pravilo odnosno pravilo konture i reza

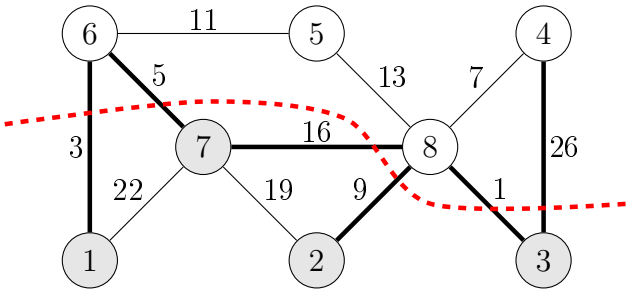
Crveno-Plavi algoritam koji ćemo opisati u ovom delu zasniva se na crvenom i plavom pravilu. U [1] prikazan je malo drugačiji pristup građenju

MPS koji se zasniva na tzv. opštem MPS postupku u kojem treba da se nađe *bezbedna grana* koja se dodaje u MPS. Bezbedna grana u stvari predstavlja najlakšu granu koja preseca rez, a postupak odgovara plavom pravilu.

Definicija 17 ***Rez*** *grafa $G = (V, E)$ je $(S, \bar{S})$, gde je S neprazan podskup skupa čvorova V, a $\bar{S} = V - S$ njegov komplement.*

Dakle, važi da je: $S, \bar{S} \neq \emptyset$, $S \cap \bar{S} = \emptyset$ i $S \cup \bar{S} = V$. Rez grafa treba da zamislimo kao podelu skupa čvorova u dva neprazna skupa.

Definicija 18 *Grana e* ***preseca*** *rez $(S, \bar{S})$ grafa $G = (V, E)$ ako je jedan od čvorova incidentnih sa e u S a drugi u $\bar{S}$.*



Slika 1.6: Rez $(\{1, 7, 2, 3\}, \{4, 8, 5, 6\})$

Većina algoritama za pronalaženje MPS stabla može da se opiše kao specijalan slučaj sledećeg postupka.

Crveno-Plavi algoritam:

1. Inicijalizacija: sve grane u grafu su crne.
2. Primenjivati sledeće pravilo sve dok ima crnih grana u grafu:
   (a) **(Plavo pravilo)** Odabrati rez $(S, \bar{S})$ takav da najlakša grana koja preseca ovaj rez nije plava i obojiti je u plavo.
   (b) **(Crveno pravilo)** Odabrati konturu C takvu da najteža grana u konturi nije crvena i obojiti je u crveno.
3. MPS čine grane obojene u plavo.

Pokazaćemo da je ovaj algoritam ispravan i da kao rezultat daje upravo MPS ulaznog grafa.

Lema 1 (Pravilo reza) *[8] Najlakša grana svakog reza je u MPS.*

Dokaz. Pokazaćemo da ovo važi svođenjem na kontradikciju. Neka je e najlakša grana koja preseca rez $(S, \bar{S})$. Ako grana e ne pripada MPS to znači da postoji grana e' koja pripada MPS a koja preseca rez $(S, \bar{S})$. Obeležimo ovo MPS sa T'. Znamo da ovakva grana postoji jer mora da postoji grana koja povezuje S i $\bar{S}$ da bi T' bio povezan. Neka je $w(T')$ težina MPS koje sadrži e'. Ako iz MPS T' obrišemo granu e' dobićemo nepovezan graf sa dve povezane komponente. Ako dodamo e umesto e' tada ćemo ponovo dobiti povezan graf. Težina novog grafa T je $w(T) = w(T') - w(e') + w(e)$. Grana e je najlakša grana koja preseca rez $(S, \bar{S})$ pa sledi da je $w(e) < w(e')$, a odavde sledi da je $w(T) < w(T')$ što znači da T' nije MPS.

■

Lema 2 (Pravilo konture) *[8] Grana e nije u MPS ako i samo ako je ona najteža na nekoj konturi.*

Dokaz. Pretpostavimo prvo da grana e nije u MPS T grafa G. Tada je e T-teška grana i ona je najteža grana na konturi koju čine grana e i put u T koji povezuje krajnje čvorove grane e.

Pretpostavimo sada da je grana $e = uv$ najteža na konturi C i da grana e pripada MPS T grafa G. Ako granu e uklonimo iz T tada ćemo T rastaviti na dve komponente: T_u i T_v. Sada su neki od čvorova iz C u T_u a neki u T_v, i znamo da postoji grana $e' \neq e$, koja takođe pripada konturi C a čija jedna krajnja tačka leži u T_u a druga u T_v. Pošto je e najteža grana na ovoj konturi, sledi da je $w(e') < w(e)$. Ako umesto grane e stavimo granu e' dobićemo pokrivajuće stablo T' koje ima manju težinu od stabla T: $w(T') = w(T) - w(e) + w(e') < w(T)$ pa sledi da T nije MPS grafa G a to je u kontradikciji sa pretpostavkom.

■

Lema 3 (Crno pravilo) *[8] Sve dok ima crnih grana u grafu možemo da primenimo bar jedno od pravila: plavo, crveno.*

Dokaz. Neka je $e = uv$ crna grana. Označimo sa M skup čvorova do kojih možemo doći iz u preko plavih grana.

Ako čvor v pripada skupu M, onda grana e zajedno sa plavim granama preko kojih možemo doći do v čini konturu. Na toj konturi je grana e najteža (jer sve plave grane pripadaju MPS i na konturi ne može da bude više T-lakih grana) pa možemo da primenimo crveno pravilo.

Ako čvor v ne pripada skupu M onda možemo napraviti rez $(M, V \backslash M)$ koji ne sadrži ni jednu plavu granu pa sledi da možemo primeniti plavo pravilo.

■

Sada smo spremni za dokaz glavne teoreme o ispravnosti gore navedenog algoritma.

Teorema 1 (Ispravnost Crveno-Plavog algoritma) *[8] Za bilo koji izbor pravila, Crveno-Plavi algoritam se izvršava i plave grane čine MPS ulaznog grafa.*

Dokaz. Primetimo da su na početku sve grane crne, a tokom izvršavanja algoritma svaku od grana bojimo samo jednom: u plavo ili crveno. Nijednu od grana ne bojimo ponovo u istu boju (ovo je obezbeđeno pravilima bojenja), a niti joj menjamo boju (ako bismo joj menjali boju to bi na osnovu Lema 1, 2 i 3 značilo da je ta grana istovremeno i u MPS i van MPS, što je nemoguće). Algoritam će se zaustaviti onog trenutka kad više ne bude crnih grana.

Kada više ne možemo da primenimo nijedno od pravila tada na osnovu Crnog pravila znamo da su sve grane obojene. Na osnovu Pravila reza sledi da su sve plave grane u MPS i na osnovu Pravila konture sledi da sve crvene grane nisu u MPS. Možemo da zaključimo da plave grane čine upravo MPS ulaznog grafa.

■

1.2.3 Postojanje i jedinstvenost MPS

Pokazaćemo da MPS postoji za sve povezane grafove i da je ono jedinstveno ako su težine grana međusobno različite. Za početak ćemo uvesti neke operacije na grafovima koje koristimo u dokazu jedinstvenosti.

Neka su dati grafovi $G_1 = (V_1, E_1)$ i $G_2 = (V_2, E_2)$. **Unija grafova** G_1 i G_2, u oznaci $G_1 \cup G_2$, je graf čiji je skup čvorova $V_1 \cup V_2$ a skup grana $E_1 \cup E_2$. **Presek grafova** G_1 i G_2, u oznaci $G_1 \cap G_2$, je graf čiji je skup čvorova $V_1 \cap V_2$ a skup grana $E_1 \cap E_2$.

Neka je dat graf $G = (V, E)$ i njegov podgraf $H = (W, E')$. Graf $G - H$ je graf čiji je skup čvorova V a skup grana $E \backslash E'$.

Tvrđenje 2 (Postojanje i jedinstvenost MPS) *[7] Svaki povezan graf G ima MPS i ukoliko grane imaju međusobno različite težine tada je MPS jedinstveno.*

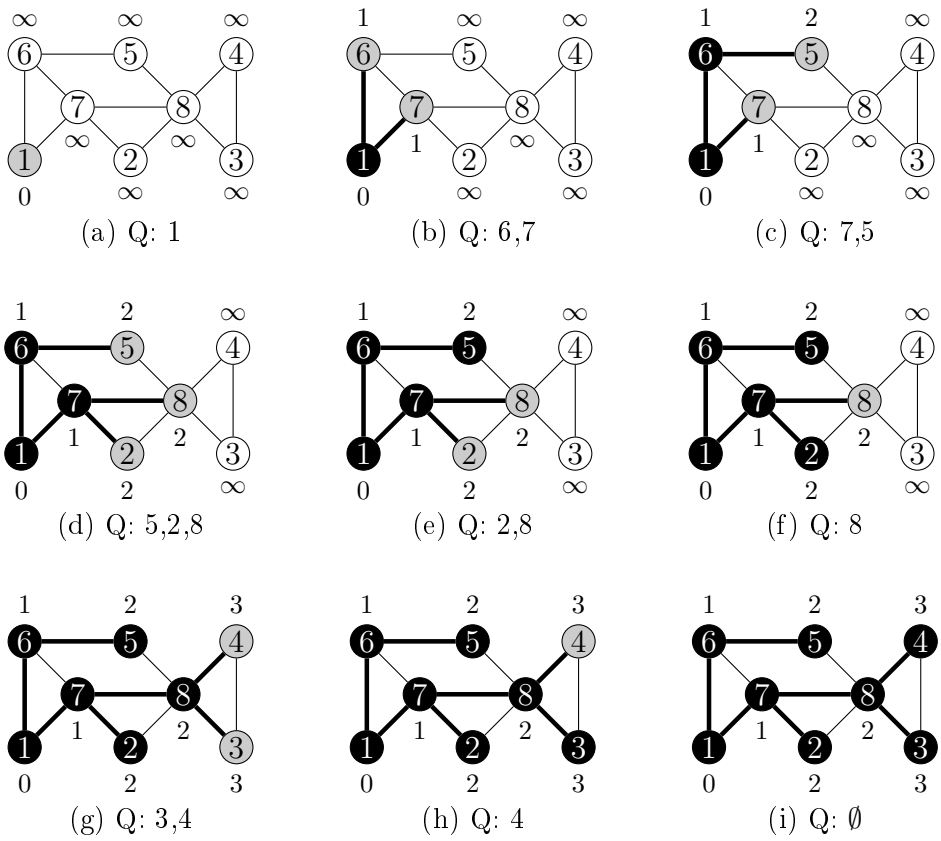
Dokaz. Postojanje: Pokrivajuće stablo grafa G postoji jer možemo da primenjujemo crveno pravilo sve dok u G ima kontura. Kad ponestane kontura

dobili smo pokrivajuće stablo. Uklanjanjem bilo koje od grana koje su u konturi nećemo narušiti povezanost grafa jer svaki put koji je prolazio kroz datu granu može da prođe i kroz ostatak konture. Takođe postoji i pokrivajuće stablo minimalne težine. Pošto su grafovi koje posmatramo konačni, sledi da možemo da numerišemo sva pokrivajuća stabla i njihove težine. Skup pokrivajućih stabala i njihovih težina je takođe konačan pa sledi da mora da postoji minimum.
Jedinstvenost: Pretpostavimo da imamo dva minimalna pokrivajuća stabla T_1 i T_2 težine $w(T_1) = w(T_2)$. Neka je grana e najteža grana u $T_1 \cup T_2 - T_1 \cap T_2$. Bez umanjenja opštosti možemo da pretpostavimo da grana $e \in T_1$. Ako obrišemo granu e iz T_1 dobićemo dve povezane komponente. Ove dve komponente ne povezuje ni jedna grana iz T_1 osim e jer bi tada postojala kontura u T_1. Pošto pokrivajuće stablo mora da bude povezano sledi da postoji bar jedna grana iz T_2 koja preseca ovaj rez. Neka je e' jedna takva grana. Znamo da e' ne pripada T_1 pa sledi da e' mora da pripada $T_1 \cup T_2 - T_1 \cap T_2$ i pošto je grana e bila najteža u ovom skupu sledi da je $w(e') < w(e)$. Sada ako umesto grane e stavimo granu e' u T_1 ponovo ćemo imati povezan graf. Dobijen graf je takođe acikličan jer nakon uklanjanja grane e iz T_1 nije postojao put između dve povezane komponente, pa dodavanjem grane e' u T_1 nismo dobili konturu. Možemo da zaključimo da smo zamenjivanjem grane e sa e' dobili pokrivajuće stablo koje ima manju težinu od težine $w(T_1) = w(T_2)$ pa sledi da nijedno od ovih pokrivajućih stabala nije minimalno. Dakle, sledi da pokrivajuće stablo minimalne težine mora da bude jedino pokrivajuće stablo sa tom težinom.

■

1.3 Pretraživanje grafa: breadth-first search i depth-first search

Algoritmi za pretraživanje grafova su nam važni jer se pomoću njih u linearnom vremenu mogu odrediti povezane komponente grafa, a takođe Primov algoritam za određivanje MPS zasniva se na ideji sličnoj onoj u breadth-first pretrazi. Daćemo kratak opis dva algoritma za pretraživanje grafova, breadth-first (pretraga po širini) i depth-first (pretraga po dubini), a više o njima može da se nađe u [1]. U primerima ćemo koristiti neusmerene grafove ali ovi algoritmi mogu da se koriste i na usmerenim grafovima.



Slika 1.7: Postupak izvršavanja Breadth-first algoritma. Brojevi izvan čvorova označavaju udaljenost. U Q možemo da pratimo koji čvorovi su u datom trenutku sivi.

Breadth-first search

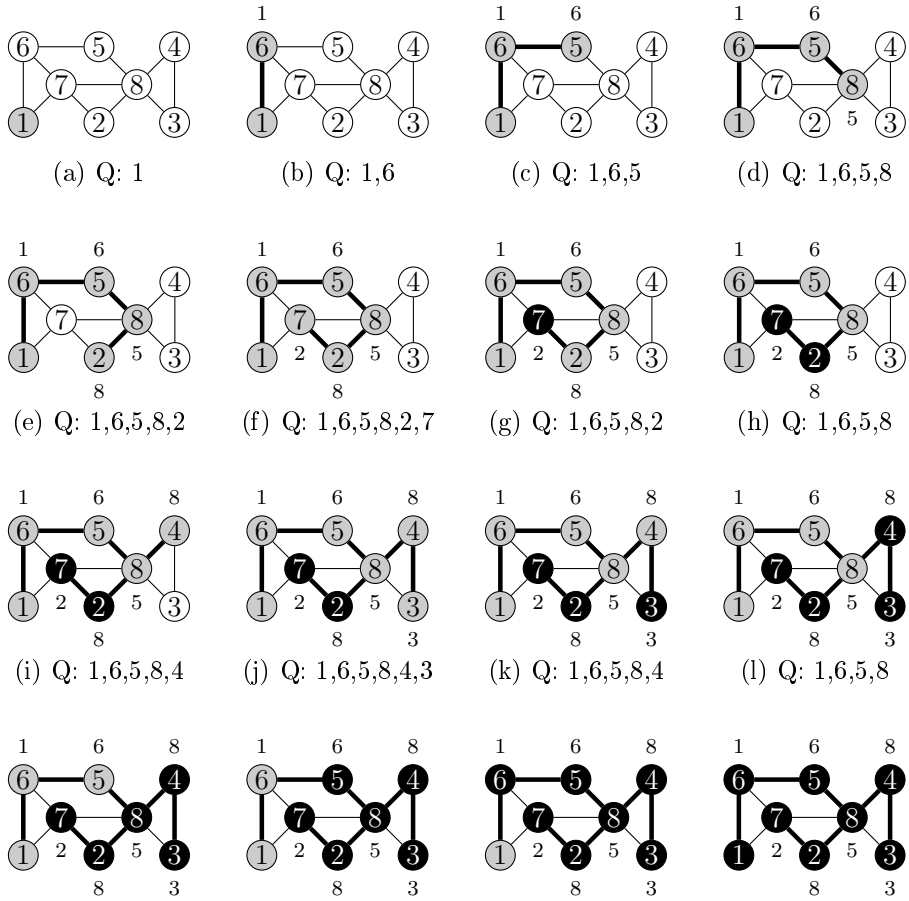
Ovaj algoritam za zadati čvor s i graf G pronalazi sve one čvorove u G do kojih može da se dođe iz s. On takođe računa minimalnu udaljenost od s do svakog od tih čvorova. Kao rezultat ovog algoritma dobijamo i bredth-first stablo u kojem je s koren stabla i ono sadrži sve čvorove do kojih može da se dođe iz s. Udaljenost od bilo kog čvora v do s u ovom stablu odgovara minimalnoj udaljenosti od v do s u grafu G.

1. Svim čvorovima dodeljujemo tri atributa: boju, roditelja i udaljenost. Na početku je boja bela, roditelj je NULL a udaljenost ∞.

2. Odaberemo čvor s koji će biti koren breadth-first stabla i promenimo njegovu boju u sivu i udaljenost u 0.

3. Definišemo red čekanja (eng. *queue*) $Q = \emptyset$ i dodamo s u Q.

4. Sve dok je $Q \neq \emptyset$ raditi sledeće:
   (a) Uzmemo prvi elemenat $u \in Q$.
   (b) Za sve susede v čvora u čija je boja bela: promenimo boju u sivu, promenimo udaljenost u udaljenost(u) + 1, promenimo roditelja u u i dodamo na kraj reda Q.
   (c) Promenimo boju čvora u u crnu.


(a) Q: 1
(b) Q: 1,6
(c) Q: 1,6,5
(d) Q: 1,6,5,8
(e) Q: 1,6,5,8,2
(f) Q: 1,6,5,8,2,7
(g) Q: 1,6,5,8,2
(h) Q: 1,6,5,8
(i) Q: 1,6,5,8,4
(j) Q: 1,6,5,8,4,3
(k) Q: 1,6,5,8,4
(l) Q: 1,6,5,8
(m) Q: 1,6,5
(n) Q: 1,6
(o) Q: 1
(p) Q: $\emptyset$

Slika 1.8: Postupak izvršavanja Depth-first algoritma.
Brojevi izvan čvorova označavaju roditelja.

Depth-first search

Ideja ovog algoritma je da polazeći od prvog čvora, koji predstavlja koren depth-first stabla, pronalazi nove čvorove duž svih grana koliko je god to

moguće pre nego što se vrati u prethodni čvor.

1. Svim čvorovima dodeljujemo dva atributa: boju i roditelja. Na početku je boja bela a roditelj NULL.

2. Za sve čvorove $u \in G$ čija je boja bela pozvati algoritam *Poseta*(G, u).

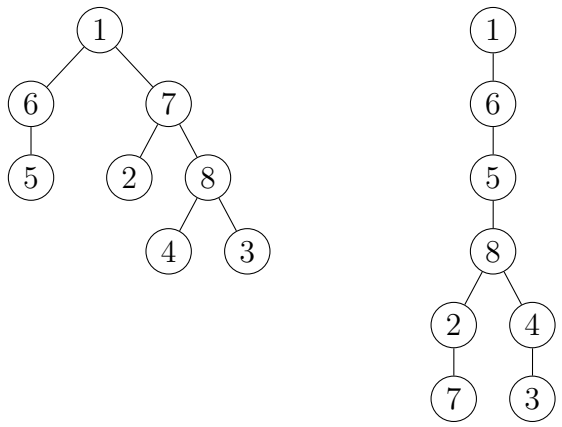
3. *Poseta*(G, u)

   (a) Promeniti boju čvora u u sivu.

   (b) Za sve susede v čvora u čija je boja bela: promenimo roditelja u u i pozovemo algoritam *Poseta*(G, v).

   (c) Promenimo boju čvora u u crnu.

Kao rezultat dobijamo Breadth-first i Depth-first stabla (slika 1.9).



Slika 1.9: Breadth-first stablo (levo) i Depth-first stablo (desno).

Glava 2

Kompleksnost algoritama

2.1 Tjuringova mašina

Tjuringova mašina (TM) predstavlja apstraktni matematički model računara koji je 1936. godine opisao engleski matematičar Alan M. Tjuring (1912 - 1954). TM se sastoji od *glave*, *trake*, *azbuke*, *skupa stanja* i *programa*. Traku treba da zamislimo kao beskonačan papir izdeljen na polja, a u svakom polju se nalazi po jedan simbol iz unapred određene konačne azbuke Σ. Iako je ova traka beskonačna, ona uvek sadrži samo konačno mnogo simbola a sva ostala polja su prazna (sadrže prazan simbol). Traka služi za skladištenje ulaznih, izlaznih podataka ali i za čuvanje rezultata u toku izračunavanja. Glava se nalazi iznad jednog polja trake i može da čita, briše i piše simbol u to polje i može da pomera traku za jedno polje levo ili desno. Mašina se u svakom trenutku nalazi u jednom od konačnog broja stanja Q, a na početku u početnom stanju $q_0 \in Q$. Program je konačan skup uputstava koja na osnovu trenutnog stanja i trenutnog simbola na traci definišu naredno stanje i akciju koju glava treba da izvrši. Dakle algoritam kreće od početnog stanja q_0 a njegov dalji rad zavisi od programa, funkcije prelaza $\delta : Q \times \Sigma \to (\Sigma \cup \{L, R, H\} \times Q)$, gde simboli L, R, H redom predstavljaju uputstva 'pomeri traku levo', 'pomeri traku desno', 'zaustavi mašinu'. Na primer, neka je trenutno stanje x i simbol ispod glave y. Jedno od uputstava može da glasi: ukoliko je trenutno stanje x i simbol y, napiši z u polje ispod glave, promeni stanje u $x2$ i pomeri traku za jedno polje levo.

Ovim je definisan rad determinističke TM kod koje je svako naredno stanje deterministički (jednoznačno) određeno. Takođe postoje i nedeterminističke TM kod kojih u svakom koraku mašina ima više mogućnosti od kojih može da izabere sledeće stanje.

2.2 Vremenska kompleksnost algoritama

Vremenska kompleksnost TM je funkcija, u oznaci $f(n)$, koja za ulaz dužine n meri maksimalan broj komandi koji će mašina izvršiti dok se ne zaustavi. Pod maksimalnim brojem podrazumevamo broj koraka u najgorem slučaju.

Pod korakom algoritma podrazumevamo izvršenje jedne komande TM koja implementira dati problem. Vremenska kompleksnost posmatranog problema predstavlja broj koraka koji je potreban da bi se problem rešio na TM, kao funkcija veličine ulaznih podataka. Kompleksnost želimo da opišemo kao funkciju veličina ulaznih parametara, broja grana m i broja čvorova n. Pored vremenske kompleksnosti postoji i prostorna kompleksnost a ona se meri maksimalnom količinom memorije koju algoritam zauzima.

2.3 Notacija O

Kompleksnost algoritma se obično procenjuje u asimptotskom smislu i funkciju kompleksnosti procenjujemo za velike veličine ulaza. To znači da nas ne zanima tačan broj koraka algoritma već hoćemo da procenimo red veličine te kompleksnosti. Za ove procene koristimo tzv. O notaciju (eng. *Big O notation*). Ova notacija opisuje ograničavajuće ponašanje funkcije kada argumenti (n, m) teže određenoj vrednosti ili beskonačnosti.

Definicija 19 *Neka su $f(x)$ i $g(x)$ funkcije definisane na nekom podskupu skupa realnih brojeva. $f(x) = O(g(x))$ kada $x \to \infty$ ako i samo ako $\exists M \in \mathbb{R}, M > 0$ i $x_0 \in \mathbb{R}$ takvi da za $\forall x > x_0$ važi: $|f(x)| \leq M|g(x)|$.*

Gornja definicija nam govori da ako je $f = O(g)$ to znači da je funkcija f reda funkcije g odnosno da je funkcija g asimptotska gornja granica funkcije f.

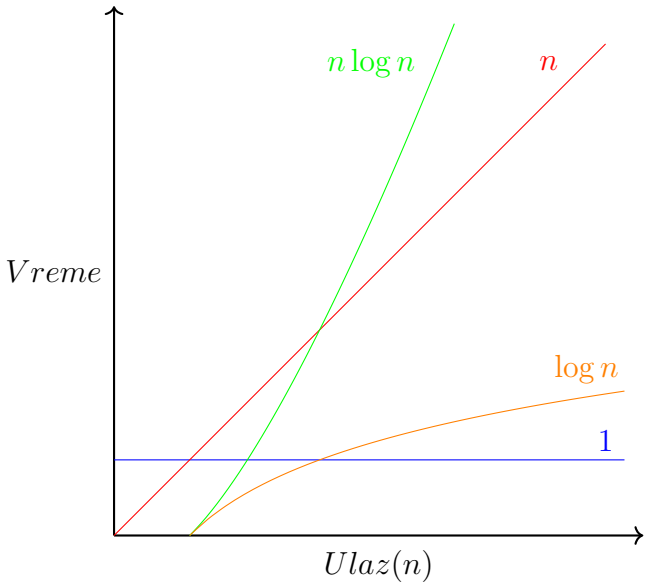
Navešćemo neke osobine O:

1. Za sve konstante $c > 0$ važi: $cf(x) = O(f(x))$.
2. Ako je $f_1(x) = O(g_1(x))$ i $f_2(x) = O(g_2(x))$, tada je $f_1(x) + f_2(x) = O(max\{g_1(x), g_2(x)\})$.
3. Ako je $f_1(x) = O(g_1(x))$ i $f_2(x) = O(g_2(x))$ tada je $f_1(x)f_2(x) = O(g_1(x)g_2(x))$.

2.4 Klase vremenske kompleksnosti

Problemi koji imaju sličnu vremensku kompleksnost pripadaju istoj klasi kompleksnosti. Jedna od najosnovnijih klasa kompleksnosti je klasa **P**. Ona sadrži polinomne algoritme, tj. algoritme koji su odlučivi u vremenu koje se može oceniti polinomnom funkcijom po veličini ulazne reči. To znači da postoji polinom p takav da se algoritam izvršava za najviše $p(n)$ vremena za ulaz dužine n. Algoritmi koji pripadaju klasi **P** smatraju se efikasnim. Pored klase **P** postoji i klasa problema odlučivanja koji mogu da se izvrše u polinomnom vremenu na nedeterminističkoj Tjuringovoj mašini, ovu klasu označavamo sa **NP**. **NP** sadrži probleme odlučivanja čije se rešenje može proveriti u polinomnom vremenu.

Neke od funkcija sa kojima se često srećemo u analizi algoritama prikazane su na slici 2.1. Ove funkcije predstavljaju asimptotska vremena izvršavanja algoritama. Konstantna funkcija 1 odnosi se na algoritme koji imaju konstantno vreme izvršavanja, oznaka $O(1)$. Analogno, imamo logaritamsko $O(\log(n))$, linearno $O(n)$ i linearno-logaritamsko $O(n \log n)$ vreme izvršavanja.



Slika 2.1: Funkcije vremena izvršavanja u odnosu na veličinu ulaza

Glava 3

Algoritmi za pronalaženje minimalnog pokrivajućeg stabla u polinomnom vremenu

U ovom delu rada predstavićemo tri klasična algoritma za pronalaženje MPS koji su odlučivi u polinomnom vremenu. Za svaki od algoritama opisaćemo sam algoritam, dokazati da algoritam kao rezultat daje upravo MPS i daćemo dokaz kompleksnosti algoritma. Takođe ćemo kroz jedan primer prikazati postupak određivanja MPS pomoću ova tri algoritma.

3.1 Borůvkin algoritam

3.1.1 Opis algoritma

Pretpostavka Borůvkinog algoritma je da su sve težine grana međusobno različite. Ovaj uslov možemo lako da prevaziđemo time što ćemo poređati grane u neopadajući niz i numerisati ih. Ako ima više grana koje imaju istu težinu onda ona koju smo prvu naveli posmatramo kao 'lakšu'. Ipak ćemo radi jednostavnosti pretpostaviti da sve grane imaju različite težine.

Ideja algoritma je sledeća: na početku svaki čvor posmatramo kao trivijalno stablo, a skup svih trivijalnih stabala kao jednu šumu. U svakoj iteraciji za svako od ovih stabala biramo najlakšu od svih grana koje imaju jedan krajnji čvor u posmatranom stablu a drugi krajnji čvor u bilo kom drugom stablu i ovu granu dodamo u šumu. Ovo radimo sve dok šuma ne postane stablo.

Sada ćemo algoritam navesti u obliku pseudo-koda radi lakšeg praćenja dokaza ispravnosti i kompleksnosti.

1. T je šuma koju čine čvorovi grafa G: $T = T_1 \cup T_2 \cup \ldots \cup T_n$, $T_i = \{u_i\}$, $i = 1, 2, \ldots, n$

2. Sve dok je T nepovezan raditi sledeće:

   (a) Za svaku komponentu (stablo) T_i šume T odabrati najlakšu granu e_i koja preseca rez $(T_i, V \backslash T_i)$.

   (b) Dodati e_i u T, za svako i.

3. T je MPS.

3.1.2 Ispravnost algoritma

Treba da pokažemo da ovaj algoritam zaista kao rezultat daje MPS početnog grafa.

Tvrđenje 3 *[8] Borůvkin algoritam kao rezultat vraća MPS ulaznog grafa.*

Dokaz. Primetimo da je korak 2(a) u stvari primena Plavog pravila. Sledi da su grane e_i koje dodajemo u T plave pa je onda i T plava šuma. Algoritam se zaustavlja onog trenutka kada T postane (plavo) stablo pa nema potrebe da primenjujemo Crveno pravilo za 'izbacivanje' grana iz T.

Treba još da pokažemo da se dodavanjem grana u T ne formira kontura. Pretpostavimo suprotno: da smo u nekoj iteraciji j dobili konturu C u T. Svi T_i-ovi su stabla i za svako stablo T_i smo izabrali najlakšu granu e_i koja preseca rez $(T_i, V \backslash T_i)$. e_1 je grana koja je bila izabrana za stablo T_1 jer je e_1 bila najlakša grana koja preseca rez $(T_1, V \backslash T_1)$. e_2 je grana koja je bila izabrana za stablo T_2 kao najlakša koja preseca rez $(T_2, V \backslash T_2)$. Uopšteno, grana e_i je u ovoj iteraciji bila najlakša grana koja preseca rez $(T_i, V \backslash T_i)$. Konturu C možemo predstaviti u sledećem obliku:
$C = T_1[u_1, v_1]v_1u_2T_2[u_2, v_2]v_2u_3T_3[u_3, v_3]v_3u_4 \ldots T_k[u_k, v_k]v_ku_1$, gde su v_iu_{i+1} grane dodate u iteraciji j, a $T_i[u_i, v_i]$ je put, tj. deo stabla T_i koji povezuje čvorove u_i i v_i. Grana e_i je v_iu_{i+1} ili $v_{i-1}u_i$. Pretpostavimo bez umanjenja opštosti da je $e_1 = v_1u_2$, tj. da grana e_1 povezuje T_1 sa T_2. Tada je $e_2 = v_2u_3$ i važi da je $w(e_2) < w(e_1)$. To znamo jer bismo u suprotnom kada biramo najlakšu granu koja preseca rez $(T_2, V \backslash T_2)$ odabrali e_1 a ne e_2. Ovde je važno napomenuti da ovo važi samo u slučaju da su sve težine grana međusobno različite. Koristeći ovu logiku za ostale grane, dobijamo sledeću nejednakost: $w(e_1) > w(e_2) > w(e_3) > \ldots > w(e_k) > w(e_1)$ što je nemoguće. Dakle, pretpostavka da smo u nekoj od iteracija dobili konturu je netačna.

■

3.1.3 Kompleksnost algoritma

Borůvkin algoritam se izvršava za $O(m \log n)$ jedinica vremena. Prvo ćemo navesti dve leme na osnovu kojih će slediti tvrđenje o kompleksnosti samog algoritma.

Lema 4 *[8] Broj stabala u T se bar prepolovi u svakoj iteraciji algoritma.*

Dokaz. Svako stablo se u svakoj iteraciji povezuje sa bar jednim od njegovih susednih stabala. Sledi da svako novo stablo sadrži bar dva stabla iz prethodne iteracije.

■

Posledica 4 *[8] Borůvkin algoritam se zaustavlja nakon najviše $O(\log n)$ iteracija.*

Dokaz. U prvoj iteraciji imamo n trivijalnih stabala. Na osnovu Leme 4 sledi da je u drugoj iteraciji broj stabala najviše $\frac{n}{2}$, u trećoj iteraciji najviše $\frac{n}{2^2}$ i tako dalje. Algoritam se izvršava sve dok ne dobijemo jedno stablo, odnosno sve dok je $\frac{n}{2^k} > 1$, gde je k broj iteracija. Dakle, tražimo najmanji broj k takav da je $\frac{n}{2^k} \leq 1$. Kada sredimo prethodnu nejednakost dobijemo da je $k \geq \frac{\log n}{\log 2}$, odakle sledi da je maksimalan broj iteracija $O(\log n)$.

■

Lema 5 *[8] Svaka iteracija Borůvkinog algoritma se izvršava za $O(m)$.*

Dokaz. Da se podsetimo šta treba da uradimo u svakoj iteraciji: za svaku povezanu komponentu (stablo) T_i šume T treba odabrati najlakšu granu e_i koja preseca rez $(T_i, V \backslash T_i)$. Takođe treba da odredimo i povezane komponente za sledeću iteraciju.

Koristićemo jedan pomoćni graf G_j, gde j predstavlja redni broj iteracije. Pretpostavimo da smo na početku j-te iteracije. Čvorovi pomoćnog grafa predstavljaju povezane komponente grafa T koji sadrži sve do sada odabrane grane za MPS. Grane pomoćnog grafa su sve grane koje povezuju različite povezane komponente.

Neka je data lista suseda za G_j, malo kasnije ćemo objasniti kako se ona određuje. Za svaki čvor prođemo kroz sve grane koje su incidentne sa datim čvorom i odaberemo granu minimalne težine. Sve ove grane dodamo u T. Sada treba da odredimo povezane komponente T_i grafa T za narednu iteraciju.

Dok tražimo povezane komponente sačuvaćemo informaciju kojem T_i-u pripada svaki od čvorova. U narednoj iteraciji $j+1$ ove komponente će postati čvorovi pomoćnog grafa G_{j+1}.

Da bismo napravili listu suseda za G_{j+1} treba da prođemo kroz sve grane $e = uv$ iz grafa G_j. Ako čvorovi u i v pripadaju različitim komponentama, recimo T_{i_l} i T_{i_k} tada granu e dodamo u G_{j+1}. U G_{j+1} grana e će povezivati čvorove koji predstavljaju povezane komponente T_{i_l} i T_{i_k}.

Povezane komponente grafa možemo da izračunamo u linearnom vremenu u odnosu na broj grana grafa (koristeći jednu od metoda breadth-first search ili depth-first search). Dakle, za izvršavanje jedne iteracije potrebno je $O(m)$ vremena.

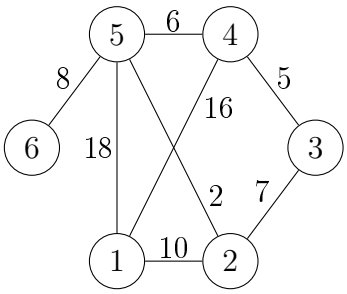
■

Na osnovu prethodnih Lema sledi:

Tvrđenje 5 *[8] Borůvkin algoritam se izvršava za $O(m \log n)$ vremena.*

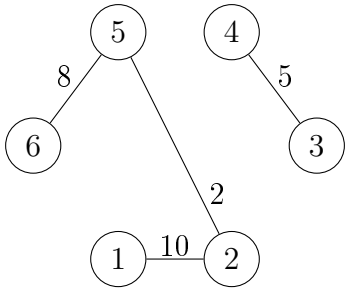
Primer

PRIMER 3 *Odredićemo MPS datog grafa Borůvkinim algoritmom.*

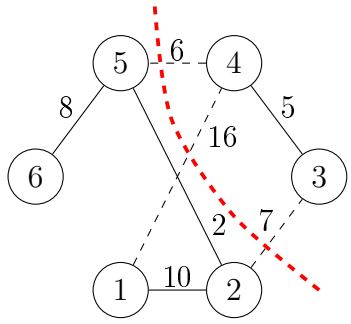


U prvoj iteraciji za svaki čvor treba da pronađemo granu incidentnu sa posmatranim čvorom koja ima najmanju težinu. Kao rezultat dobijamo šumu sa dve povezane komponente:

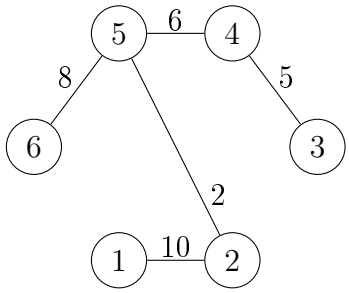
U drugoj iteraciji treba da nađemo najlakšu granu koja preseca rez $(\{1,2,5,6\},\{4,3\})$. *Tri grane presecaju ovaj rez, a najmanju težinu ima grana koja povezuje čvorove 5 i 4, pa nju dodajemo u MPS. Dobili smo povezan graf pa se ovde algoritam završava.*



(a) Rezultat nakon jedne iteracije Borůvkinog algoritma



(b) Rez $(\{1, 2, 5, 6\}, \{4, 3\})$ presecaju tri grane



Slika 3.2: MPS određeno Borůvkinim algoritmom

3.2 Primov algoritam

Kod Primovog algoritma ne mora da važi pretpostavka o međusobno različitim težinama grana. Ideja algoritma je sledeća: polazimo od proizvoljnog čvora u koji dodamo u T. MPS gradimo tako što u svakom koraku dodajemo granu koja povezuje jedan čvor iz T sa čvorom koji nije u T. Na ovaj način možemo biti sigurni da se neće formirati kontura u T. Algoritam se zaustavlja nakon što smo dodali $n-1$ granu u T, tj. nakon što smo povezali sve čvorove.

3.2.1 Opis algoritma

1. T je trivijalno stablo koje sadrži proizvoljan čvor iz grafa G.
2. Sve dok postoji čvor koji nije u T raditi sledeće:
   (a) Odabrati najlakšu granu uv takvu da važi: $u \in V(T)$ i $v \notin V(T)$.
   (b) Dodati granu uv u T: $T = T \cup uv$.
3. T je MPS.

3.2.2 Ispravnost algoritma

Tvrđenje 6 *[8] Primov algoritam kao rezultat vraća MPS ulaznog grafa.*

Dokaz. Korak $2a$ odgovara primeni plavog pravila ako posmatramo rez koji razdvaja T od ostatka početnog grafa. Sledi da je T plavo stablo u svakoj iteraciji algoritma. Algoritam se zaustavlja nakon tačno $n-1$ iteracija, odnosno nakon dodavanja $n-1$ plavih grana u T.

■

3.2.3 Kompleksnost algoritma

Kod Primovog algoritma vrlo je važno da najlakšu granu koja povezuje čvor iz T sa čvorom koji nije u T pronađemo što je brže moguće. Označimo skup čvorova koji nisu u T sa Q. Pronalaženje najlakše grane možemo da posmatramo i kao problem određivanja čvora iz Q koji je povezan sa nekim čvorom iz T i pritom grana koja ih povezuje ima najmanju težinu od svih grana koje povezuju neki čvor iz T sa čvorom iz Q. Da bismo ovaj problem efikasno rešili koristićemo (binarnu) hip strukturu podataka i prikaz reda sa prioritetom pomoću hipa.

Hip struktura podataka i redovi sa prioritetom

Navešćemo neke osnovne pojmove u vezi sa hip strukturom podataka i redovima sa prioritetom da bismo razumeli dokaz kompleksnosti Primovog algoritma. Više detalja o hip strukturi, o drugim strukturama podataka, primenama kao i dokaz kompleksnosti može da se nađe u [1].

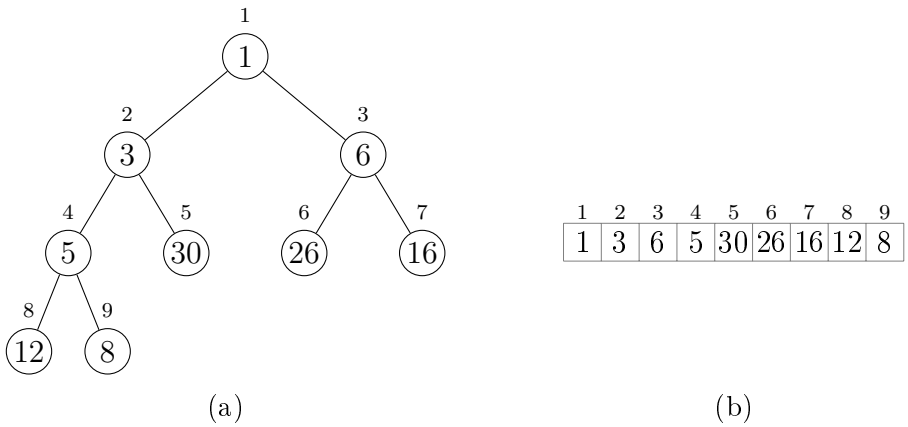
Hip je struktura koja omogućuje da binarno stablo predstavimo u obliku niza. Svakom čvoru stabla odgovara jedan elemenat niza. Ova struktura ima mnogo primena a za nas je posebno interesantno to što pomoću hipa možemo da predstavimo red sa prioritetom.

Red sa prioritetom je struktura podataka za održavanje skupa elemenata Q, gde je svakom elementu skupa Q pridružen atribut koji se zove *ključ*. Koristićemo min-red sa prioritetom (postoji još i max-red sa prioritetom) u kojem važi sledeći uslov: ključ svakog čvora je manji ili jednak od ključeva njegove dece a koren stabla ima najmanji ključ.

Ako red indeksiramo tako da prvom elementu odgovara indeks 1, onda važi da je za i-ti čvor:

- čvor sa indeksom 2^*i njegovo levo dete,
- čvor sa indeksom 2^*i+1 njegovo desno dete,

- čvor sa indeksom $\lfloor i/2 \rfloor$ njegov roditelj.



Slika 3.3: Min-red prikazan u obliku stabla i u obliku niza. Brojevi prikazani u krugu čvora u stablu predstavljaju ključ tog čvora, a broj iznad čvora je njegov indeks.

Da bismo mogli da koristimo min-red svakom čvoru ćemo dodeliti dva nova atributa: *udaljenost* i *roditelj*. *Udaljenost* čvora $u \in Q$ je težina najlakše grane e koja je incidentna sa čvorom u i koja preseca rez (T, Q). *Roditelj* čvora u je čvor iz T sa kojim je u povezan granom e. Možemo da zaključimo da će udaljenost čvora biti ključ u min-red strukturi. Čvor koji tražimo je onaj koji ima minimalnu udaljenost - on je koren binarnog stabla - on je prvi elemenat u redu.

Neke od operacija koje podržava min-red struktura su:

- **vrati-min**(Q): vraća prvi elemenat reda - elemenat koji ima najmanji ključ i uklanja ga iz reda. To znači da *vrati-min* uklanja koren stabla i treba da ažuriramo stablo tako da ponovo važi uslov da je ključ svakog čvora manji ili jednak od ključeva njegove dece, a koren ima najmanji ključ. Ukupno vreme potrebno za izvršavanje operacije *vrati-min*(Q) na Q koji ima n elemenata je $O(\log n)$.

- **smanji-ključ**$(Q, u, w(uv))$: smanjuje vrednost ključa čvora u dodeljujući mu novu vrednost $w(uv)$ za koju pretpostavljamo da je manja od trenutne vrednosti ključa čvora u. Vreme potrebno za izvršavanje ove operacije na redu sa prioritetom Q koji ima n elemenata je $O(\log n)$.

Dokaz kompleksnosti

Tvrđenje 7 *[1] Primov algoritam izračunava MPS grafa sa n čvorova i m grana za $O(m \log n)$ jedinica vremena.*

Dokaz. Prvo ćemo navesti pseudo-kod koristeći min-red operacije a nakon toga ćemo izračunati vreme izvršavanja algoritma.

1. $T = \emptyset$.
2. **for** $i = 1 : n$
   roditelj$(u_i) = NULL$
   udaljenost$(u_i) = \infty$
3. udaljenost$(u^*) = 0$
4. $Q = \{u_1, u_2, \ldots, u_n\}$
5. **while** $Q \neq \emptyset$
   $u =$ vrati-min(Q)
   dodaj granu $e = \{u, roditelj(u)\}$ u T
   for each $v \in$ lista-suseda(u)
   if $v \in Q$ i $w(uv) < udaljenost(v)$
   $roditelj(v) = u$
   $udaljenost(v) = w(uv)$

Na početku svakom čvoru dodeljujemo dva atributa: *udaljenost* i *roditelj*. Svaki čvor na početku ima *udaljenost* beskonačno i *roditelja* $NULL$. Odaberemo proizvoljan čvor u^*, dodamo ga u T (to je čvor od kojeg počinjemo da gradimo MPS) i dodelimo mu *udaljenost* 0. Za izvršavanje koraka 1-4 potrebno je $O(n)$ vremena jer ukupno ima n čvorova.

Svi oni čvorovi koji nisu u T su u min-redu sa prioritetom Q, poređani na osnovu njihove udaljenosti. Na početku su svi čvorovi u Q, a kad se algoritam završi Q je prazan.

Peti korak opisuje petlju koja se izvršava n puta. U svakoj iteraciji treba da nađemo najlakšu granu koja povezuje T i Q, tj. treba da nađemo čvor iz Q koji ima minimalnu *udaljenost* a to je upravo prvi čvor u Q. Ovaj čvor i granu incidentnu sa njom prebacimo u T i obrišemo iz Q. Označimo taj čvor sa u. Treba da ažuriramo *udaljenost* i *roditelja* svakog od čvorova povezanih sa u, tj. svakog od čvorova iz liste suseda čvora u. Označimo jedan od tih čvorova sa v. *Udaljenost* ažuriramo na sledeći način: ako je $v \in Q$ i $w(uv) < udaljenost(v)$ onda je nova $udaljenost(v) = w(uv)$ i novi $roditelj(v) = u$. Ovaj postupak odgovara upravo operaciji *smanji-ključ*.

Sada pošto smo detaljno opisali algoritam možemo da analiziramo vreme izvršavanja. Ono direktno zavisi od toga na koji način ćemo da implementiramo red sa prioritetom Q. Koristeći binarni min-hip ćemo dobiti najbolji rezultat. *While* petlja u petom koraku se izvršava n puta, a za jedno izvršavanje operacije *vrati-min* je potrebno $O(\log n)$ vremena pa je to ukupno

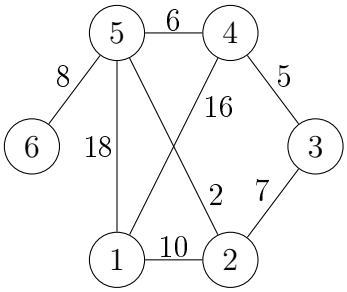
$O(n \log n)$. Već smo zaključili da poslednje dve komande u *if* petlji odgovaraju operaciji *smanji-ključ*. Ove komande će se izvršiti ukupno $2m$ puta jer je broj svih suseda $2m$ (ovo važi jer se za svaku granu $e = (uv)$ čvor u pojavljuje u listi suseda čvora v i čvor v se pojavljuje u listi suseda čvora u), a za jedno izvršavanje operacije *smanji-ključ* je potrebno $O(\log n)$ vremena pa ukupno imamo $O(m \log n)$.

Kada saberemo dobijamo: $O(n) + O(n \log n) + O(m \log n) = O(m \log n)$.

■

Primer

PRIMER 4 *Odrediti MPS datog grafa Primovim algoritmom.*



Čvorove koje dodamo u MPS obojićemo u sivo a grane ćemo obeležiti debljom linijom. Počećemo da gradimo MPS od čvora 1. Treba da nađemo najlakšu granu koja preseca rez $(\{1\}, \{2, 3, 4, 5, 6\})$. *To je grana* $\{1, 2\}$. *Nju dodamo u MPS. Najlakša grana koja preseca rez* $(\{1, 2\}, \{3, 4, 5, 6\})$ *je* $\{2, 5\}$ *i nju dodamo u MPS. Nastavljamo sve dok ima čvorova koji nisu sivi.*

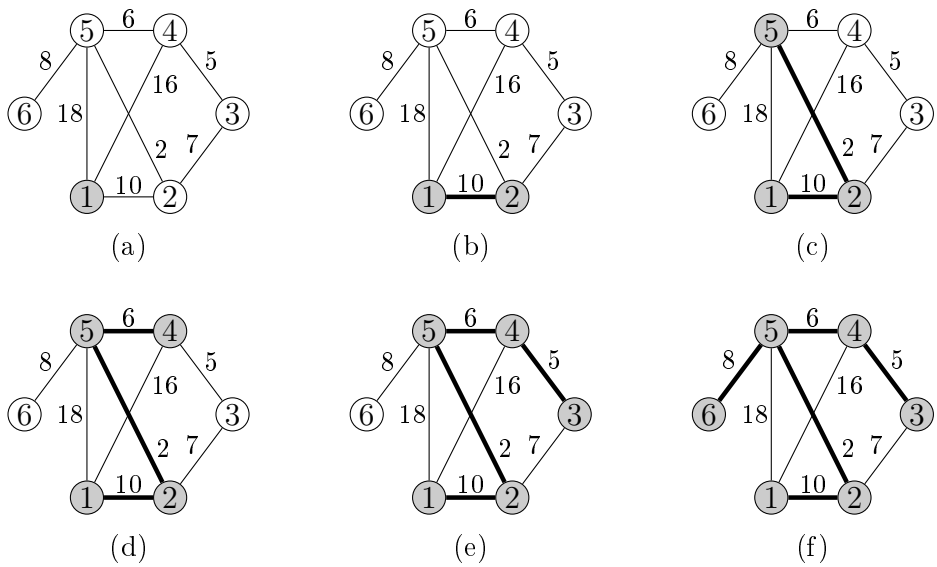
3.3 Kruskalov algoritam

3.3.1 Opis algoritma

Kruskalov algoritam, kao i Borůvkin, gradi MPS tako što kreće od prazne šume T, odnosno šume čiji su elementi svi čvorovi ulaznog grafa. Zatim poređa grane u neopadajući niz po težinama i za svaku granu proverava da li će se njenim dodavanjem u šumu T formirati kontura. Ukoliko dodavanje posmatrane grane ne prouzrokuje formiranje konture tada je dodajemo u T.

1. Poređati grane u neopadajući niz po težinama.

2. T je šuma koju čine čvorovi grafa G: $T = \{u_1, u_2, \ldots u_n\}$.



Slika 3.4: Postupak određivanja MPS Primovim algoritmom.

3. Za sve grane e (koje su poređane u neopadajući niz po težinama) raditi sledeće: Ako je $T = T+e$ acikličan dodati e u T. U suprotnom izostaviti e.

4. T je MPS.

3.3.2 Ispravnost algoritma

Tvrđenje 8 *[1] Kruskalov algoritam kao rezultat vraća MPS ulaznog grafa.*

Dokaz. Analizirajmo treći korak algoritma. Ovaj postupak odgovara bojenju grana u plavo i crveno, odnosno primeni crvenog i plavog pravila.

Ako je $T+e$ acikličan to znači da je e grana na nekom rezu koji razdvaja komponente T. Pri tome je e najlakša grana koja preseca ovaj rez jer sve ostale grane ovog reza još nismo obradili jer su one teže od e. Dakle dodavanje grane e odgovara bojenju grane e u plavo. Odatle sledi da su sve grane koje su u T plave.

Ako je $T+e$ cikličan, odnosno dodavanjem grane e u T dobijamo konturu, onda znamo da je grana e najteža na ovoj konturi. Ovo sledi iz toga što su sve grane koje su prethodno dodate u T lakše od e. Možemo da zaključimo da izostavljanje grane e odgovara primeni crvenog pravila, odnosno bojenju grana u crveno.

Na kraju algoritma sve grane su obojene, sve grane u T su plave pa T mora da bude MPS.

■

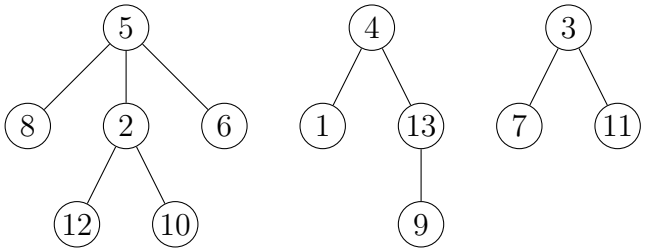
3.3.3 Kompleksnost algoritma

Struktura podataka disjunktnih skupova

Struktura disjunktnih skupova održava kolekciju disjunktnih dinamičkih skupova. Svaki od disjunktnih skupova ima jedinstvenog predstavnika, a on je jedan od elemenata skupa.

Prilikom građenja MPS pomoću Kruskalovog algoritma, u T ćemo imati više povezanih komponenti. Svaka od povezanih komponenti grafa T je jedan disjunktni skup čvorova i jednog od čvorova iz skupa ćemo proglasiti predstavnikom skupa.

Disjunktne skupove ćemo predstavljati pomoću kolekcije korenskih stabala u kojem svaki čvor stabla odgovara jednom čvoru grafa T a predstavnik stabla je čvor koji se nalazi u korenu.



Slika 3.5: Primer disjunktnih skupova koji su predstavljeni u obliku korenskih stabala. Predstavnici ova tri disjunktna skupa su čvorovi koji se nalaze u korenu stabala, a to su čvorovi 5, 4 i 3.

Ova struktura je važna jer podržava neke od operacija koje ćemo koristiti u Kruskalovom algoritmu:

1. **napravi-skup(u)**: postavlja u za predstavnika odnosno za koren jednočlanog skupa $\{u\}$.

2. **pronađi(u)**: vraća koren stabla u kojem se nalazi čvor u. Ova operacija je korisna da bismo utvrdili da li su dva čvora u istoj povezanoj komponenti (u istom disjunktnom skupu). Ako je koren isti za oba čvora onda se oni nalaze u istoj povezanoj komponenti i dodavanje grane koja ih povezuje bi prouzrokovalo stvaranje konture u T.

3. **unija(u,v)**: spaja skupove koji sadrže u i v u jedan skup.

Da bismo mogli da koristimo prethodno navedene operacije svakom čvoru ćemo dodeliti dva atributa: *rang* i *roditelj*. *Rang* čvora predstavlja maksimalnu moguću visinu stabla u kom se čvor nalazi. *Roditelj* čvora je čvor koji se nalazi iznad njega u stablu. Čvor koji je koren stabla (predstavnik) je sam svoj roditelj.

Operacija *napravi-skup(u)* čvoru u dodeljuje rang 0 i postavlja u za svog roditelja. Ova operacija se izvršava u konstantnom vremenu. Operacija *unija(u,v)* postavlja u za roditelja čvora v i povećava rang čvora u za jedan (ili postavlja v za roditelja čvora u i povećava rang čvora v za jedan). Ova operacija se takođe izvršava u konstantnom vremenu. Operacija *pronađi(u)* prolazi kroz čvorove na putu od u do korena stabla. Vreme izvršavanja zavisi od visine stabla, odnosno od ranga korena stabla. Može se pokazati ([4]) da je vreme potrebno za prelaženje puta od nekog čvora u stablu do korena stabla $O(\log k)$, gde je k broj čvorova u datom stablu. Znamo da je ukupan broj čvorova u svim disjunktnim skupovima n, $n \geq k$, pa sledi da je vreme izvršavanja operacije *pronađi* $O(\log n)$.

Dokaz kompleksnosti

Tvrđenje 9 *Kruskalov algoritam izračunava MPS grafa sa m grana i n čvorova za $O(m \log n)$ jedinica vremena.*

Dokaz. Prvo ćemo navesti pseudo-kod koji će nam pomoći da izračunamo ukupno vreme potrebno za izvršavanja algoritma.

1. $T = \emptyset$.

2. **for** $i = 1 : n$
   napravi-skup(u_i)

3. Poređati grane u neopadajući niz po težinama.

4. **for** $j = 1 : m$ (za svaku granu $e_j = uv$)
   if pronađi-skup(u) $\neq$ pronađi-skup(v)
   $T = T + e_j$
   unija(u,v)

Za izvršavanje n *napravi-skup* operacija potrebno je $O(n)$ vremena.

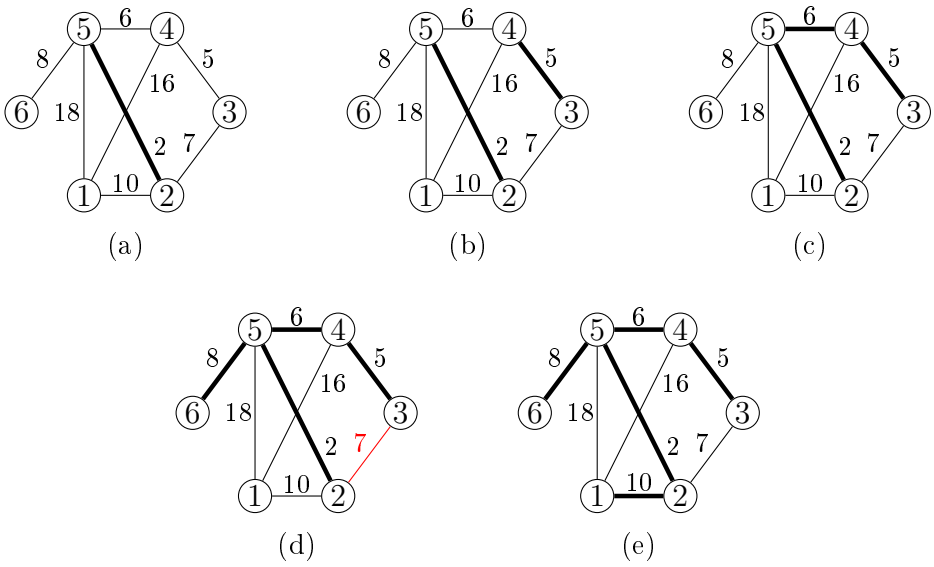
Da bismo poređali grane u neopadajući niz po težinama potrebno je $O(m \log m)$ vremena. Ovde se nećemo baviti implementacijom algoritama za sortiranje, više o njima može da se nađe u [1]. Primetimo da je $m < \frac{n^2}{2}$, pa je $O(m \log m) = O(m \log n)$.

For petlja u četvrtom koraku izvršava se ukupno m puta a svaka iteracija se izvršava za $O(\log n)$ vremena pa je to ukupno $O(m \log n)$ vremena. Kada sve saberemo dobijamo: $O(n) + O(m \log n) + O(m \log n) = O(m \log n)$.

■

Primer

PRIMER 5 *Odrediti MPS datog grafa Kruskalovim algoritmom. Grane analiziramo u rastućem redosledu. Ukoliko se dodavanjem posmatrane grane ne formira kontura u MPS tada je dodajemo.*



Slika 3.6: Postupak određivanja MPS Kruskalovim algoritmom. Crvenom bojom na slici (d) obeležena je grana čijim bi se dodavanjem formirala kontura i zato ona nije postala deo MPS.

3.4 Optimizovani Kruskalov algoritam

Optimizacija operacija na strukturi disjunktnih skupova

1. Unija po rangu (*eng. union by rank*). Ideja je da prilikom spajanja skupova u operaciji *unija* uzmemo u obzir rangove korenova skupova koje želimo da spojimo. Ako su rangovi različiti onda čvor sa većim rangom postaje roditelj čvora sa manjim rangom i rang oba čvora ostaje nepromenjen. Ako su rangovi isti onda na proizvoljan način odaberemo

čvor koji će postati roditelj drugog čvora i njegov rang se poveća za jedan.

2. Kompresija puta *(eng. path compression)*. Ideja je da svaki čvor umesto roditelja pokazuje direktno na predstavnika skupa, tj. na koren stabla u kojem se nalazi. Ova operacija ne menja rang čvorova.

Ono što je bitno za nas je vreme izvršavanja operacija ukoliko implementiramo prethodno navedene optimizacije. Može se pokazati ([1]) da je za izvršavanje niza od m *napravi-skup*, *unija* i *pronađi* operacija potrebno $O(m\alpha(n))$ vremena gde je $\alpha(n)$ poznata inverzna Akermanova funkcija. Akermanova funkcija je veoma brzo rastuća funkcija, pa je njena inverzna veoma sporo rastuća i ona zadovoljava sledeću nejednakost: $\alpha(n) \leq 4$, za $n \leq 10^{80}$.

Tvrđenje 10 (Kompleksnost optimizovanog Kruskalovog algoritma) *Koristeći strukturu disjunktnih skupova sa unijom po rangu i kompresijom puta Kruskalov algoritam izračunava MPS zadatog grafa za $O(m\alpha(n) + m \log n)$ vremena. Ako su grane već poređane u neopadajući niz po težinama tada je vreme izvršavanja $O(m\alpha(n))$.*

Dokaz tvrđenja prevazilazi okvire ovog rada, on može da se nađe u [1].

3.5 Verifikacija MPS

U ovom kratkom delu posmatramo problem koji je usko vezan za problem pronalaženja MPS grafa, a to je problem provere da li je dato pokrivajuće stablo T težinskog grafa $G = (V, E, w)$ minimalno.

Tarjan je 1979. godine opisao verifikacioni algoritam koji se izvršava za $O(m\alpha(m, n))$ vremena, gde je α inverzna Akermanova funkcija. 1985. godine Komlós je pokazao da je za verifikaciju potreban linearan broj poređenja težina grana, ali sa nelinearnim vremenom potrebnim za odabir grana koje treba da se uporede. Dixon, Rauch i Tarjan su 1992. kombinacijom prethodno pomenutih algoritama dobili verifikacioni algoritam koji se izvršava u linearnom vremenu. 1993. Valerie King opisala je jednostavniji verifikacioni algoritam koji se takođe izvršava u linearnom vremenu.

Mi ćemo ovde navesti samo rezultat prethodno pomenutih radova.

Podsetimo se, za granu $e = uv \in E$ sa $w_T(e)$ označavamo težinu najteže grane na jedinstvenom putu u T koji povezuje u i v. Ako $e \in T$ onda je $w_T(e) = w(e)$.

Lema 6 *Pokrivajuće stablo T grafa $G = (V, E, w)$ je MPS ako i samo ako za svaku granu $e \notin T$ važi da je $w(e) \geq w_T(e)$.*

Tvrđenje 11 *Postoji deterministički algoritam odlučiv u linearnom vremenu koji za dati težinski neusmereni graf* $G = (V, E, w)$ *i za pokrivajuće stablo* T *grafa* G *izračunava* $w_T(e)$ *za svaku granu* $e \in E$.

Posledica 12 *Postoji deterministički algoritam odlučiv u linearnom vremenu koji za dati težinski neusmereni graf* $G = (V, E)$ *i pokrivajuće stablo* T *proverava da li je* T *MPS grafa* G.

Glava 4

Empirijska analiza klasičnih algoritama

Četiri algoritma opisana u prethodnoj glavi implementirana su u programskom jeziku R. Cilj analize je da utvrdimo koji od algoritama je najbrži za proizvoljne grafove i da odredimo vezu između broja grana i čvorova u ulaznom grafu i vremena izvršavanja algoritama. U ovom delu ćemo predstaviti rezultate analize Primovog, Borůvkinog i četiri verzije Kruskalovog algoritma. Pokazalo se da je Kruskalov algoritam znatno brži od Primovog i Borůvkinog i zato smo analizu podelili u dva dela: u prvom poredimo Borůvkin i Primov algoritam jer su njihova vremena izvršavanja približna, a u drugom delu ćemo da uporedimo više verzija Kruskalovog algoritma. Svi korišćeni kôdovi mogu da se nađu u Prilogu rada.

Tokom eksperimenta pokazalo se da je vreme izvršavanja algoritma različito kada se algoritam pokrene više puta za isti graf. Iako su ove razlike relativno male, da bismo dobili preciznije rezultate za svaki graf smo svaki od algoritama pokrenuli po 10 puta. Takođe smo za svaki željeni par gustina-/broj čvorova kreirali 10 različitih grafova. Kao konačno vreme izvršavanja algoritma za određenu gustinu/broj čvorova uzet je prosek 100 vremena (10 različitih grafova, po 10 izvršavanja za isti graf).

Pre poređenja algoritama opisaćemo algoritam za kreiranje proizvoljnih grafova.

4.1 Kreiranje proizvoljnih grafova

Za testiranje algoritama potrebni su grafovi, a za njihovo kreiranje bilo je potrebno napisati funkciju (kôd) koji za zadate parametre kao rezultat vraća graf. Ulazni parametri su *broj čvorova*, *maksimalna težina grane* i *gustina*

grafa. Za maksimalnu težinu grane uzeli smo 10^9. Izlaz je matrica sa tri kolone a svaki red predstavlja jednu granu grafa. U prvoj koloni nalazi se prvi čvor, u drugoj koloni drugi čvor a u trećoj koloni težina grane koja povezuje ova dva čvora.

randomgraph, funkcija koja kreira graf, polazi od prazne kvadratne matrice formata *broj čvorova* x *broj čvorova*. Ova matrica predstavlja težinsku matricu grafa, D. Svakom polju matrice D sa verovatnoćom *gustina grafa* funkcija dodeljuje proizvoljan ceo broj iz intervala [1, *maksimalna težina grane*], a sa verovatnoćom 1-*gustina grafa* datom polju dodeljuje vrednost 0. Ukoliko je u polju (i, j) matrice D vrednost 0 to znači da čvor i i čvor j nisu povezani granom. Ukoliko je vrednost različita od nule, to znači da su čvorovi i i j povezani granom težine $D(i, j)$. Zatim se svi potrebni podaci prebacuju u matricu sa tri kolone: ukoliko je $D(i, j) > 0$, u matricu se dodaje red sa vrednostima i j $D(i, j)$. Ovo se izvršava samo za $j > i$, odnosno samo za polja iznad glavne dijagonale jer nas zanimaju samo neusmereni grafovi.

Na ovaj način izbegli smo mogućnost da se kreira više grana koje povezuju isti par čvorova. Na kraju se još od svih grana brišu one koje imaju istu težinu jer je kod Borůvkinog algoritma uslov da grane imaju međusobno različite težine. Možemo da zaključimo da graf koji dobijemo kao rezultat ove funkcije neće imati tačno zadatu gustinu već može da dođe do manjih odstupanja, a koja možemo da zanemarimo.

4.2 Analiza Borůvkinog i Primovog algoritma

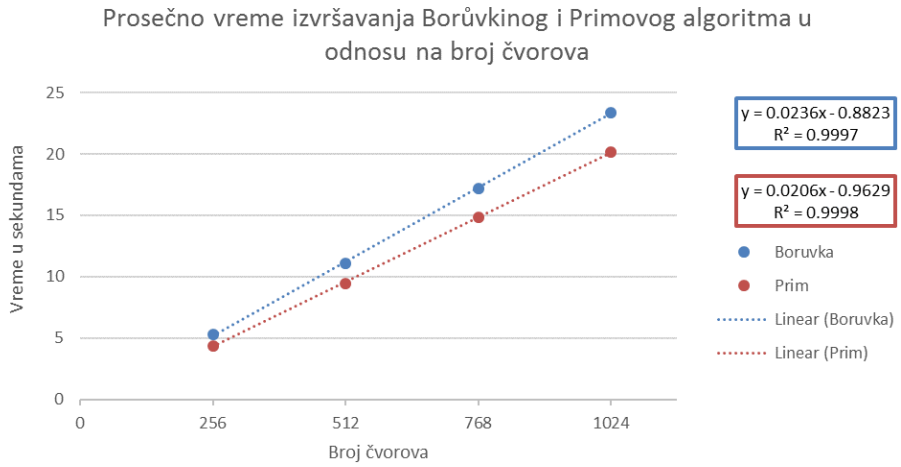
Vremena izvršavanja Borůvkinog i Primovog algoritma uporedili smo prvo na grafovima koji imaju približno isti broj grana a različit broj čvorova, a zatim u drugom delu na grafovima sa 1024 čvora a različitim brojem grana.

Kreirali smo proizvoljne grafove sa 256, 512, 768 i 1024 čvora tako da svaki od njih ima približno 31500 grana.

Broj čvorova	Gustina	Prosečan broj grana	PVI	
			Borůvka	Prim
256	0.96	31316	5.28	4.39
512	0.24	31386	11.05	9.47
768	0.11	32318	17.22	14.82
1024	0.06	31449	23.38	20.17

Tabela 4.1: Rezultati analize Borůvkinog i Primovog algoritma na grafovima sa približno 31500 grana

U Tabeli 4.1 vidimo rezultate prvog eksperimenta, gde je u koloni PVI prikazano prosečno vreme izvršavanja algoritama izraženo u sekundama. Primetimo da je Primov algoritam brži od Borůvkinog. Da bismo preciznije odredili vezu između broja čvorova i vremena izvršavanja algoritama koristili smo linearnu regresiju.



Slika 4.1: Rezultati analize Borůvkinog i Primovog algoritma na grafovima sa približno 31500 grana

Na Slici 4.1 prikazani su rezultati iz prethodne tabele i takođe je dodata linearna regresiona linija. Na grafovima koji imaju 31500 grana, ukoliko povećamo broj čvorova za 100 možemo da očekujemo da će se vreme izvršavanja Borůvkinog algoritma povećati za 2.36 sekundi, a Primovog algoritma za 2.06 sekundi.

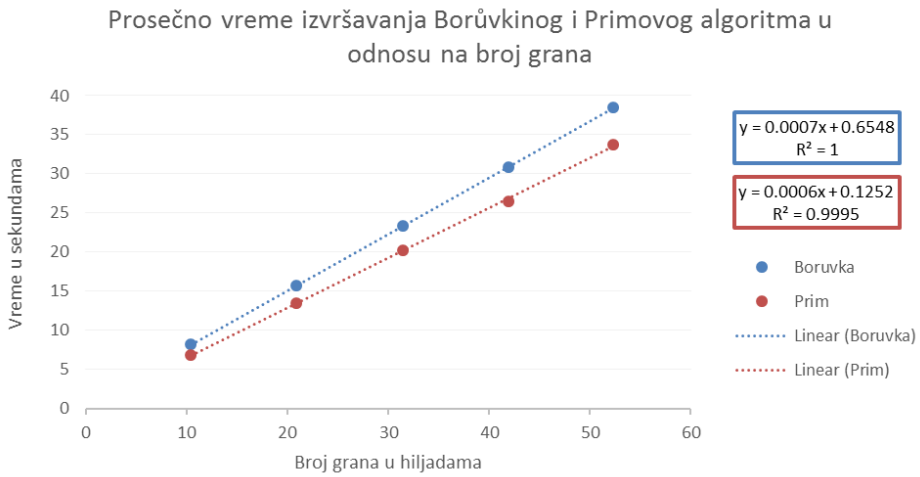
		PVI	
Gustina	Prosečan broj grana	Borůvka	Prim
0.02	10415	8.16	6.87
0.04	20855	15.75	13.43
0.06	31449	23.38	20.17
0.08	41890	30.83	26.49
0.1	52340	38.49	33.80

Tabela 4.2: Rezultati analize Borůvkinog i Primovog algoritma na grafovima sa 1024 čvora

U drugom delu eksperimenta fiksirali smo broj čvorova na 1024, a menjali smo broj grana. Broj grana određen je na osnovu zadate gustine grafa, za

pet različitih gustina, od 0.02 do 0.1 sa korakom 0.02.

U Tabeli 4.2 prikazani su rezultati analize. Možemo da primetimo da se vreme izvršavanja povećava sa povećanjem broja grana i da je Primov algoritam i u ovom slučaju brži od Borůvkinog. Ponovo smo koristili linearnu regresiju da bismo tačno odredili vezu između broja grana i vremena izvršavanja algoritama. Na Slici 4.2 prikazana je linearna regresiona linija za oba algoritma, kao i njena jednačina.



Slika 4.2: Rezultati analize Borůvkinog i Primovog algoritma na grafovima sa 1024 čvora

Na osnovu jednačina vidimo da za svakih dodatnih 10000 grana možemo da očekujemo da se vreme izvršavanja Borůvkinog algoritma poveća za 7 sekundi a Primovog algoritma za 6 sekundi.

4.3 Analiza Kruskalovog algoritma

Četiri verzije Kruskalovog algoritma implementirane su u *R*-u. Analiza je rađena na grafovima sa 1024 čvora, sa brojem grana od 10 do 52 hiljade, i na grafovima sa 2048 čvorova, sa brojem grana od 42 do 209 hiljada. Prvo ćemo opisati po čemu se razlikuju četiri implementirane verzije a zatim ćemo predstaviti rezultate analize.

1. **Osnovni** - algoritam opisan u delu 3.3.

2. **UBR** - osnovni Kruskalov algoritam opisan u delu 3.3 sa unijom po rangu iz dela 3.4.

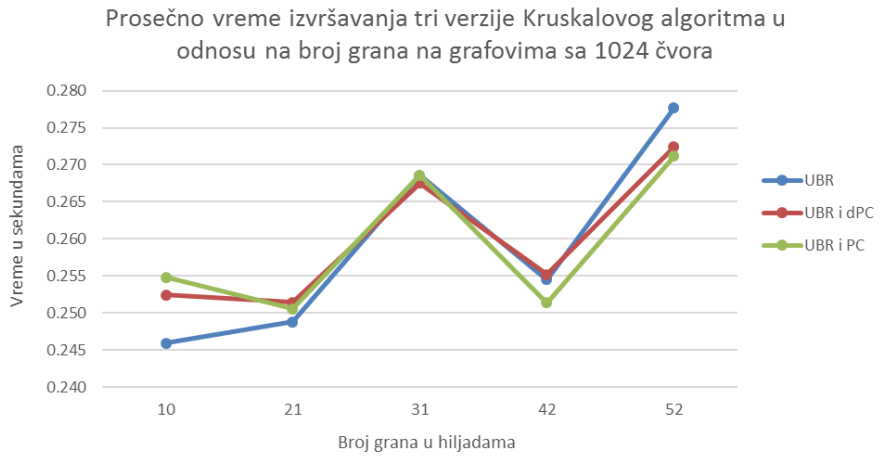
3. **UBR i dPC** - UBR i delimična kompresija puta iz dela 3.4. Delimična zbog toga što kada pronađemo koren stabla u kojem se nalazi čvor u, tada promenimo roditelja čvora u u koren stabla. Za potpunu kompresiju puta potrebno je promeniti roditelja svih čvorova na putu od u do korena u koren stabla.

4. **UBR i PC** - osnovni Kruskalov algoritam sa unijom po rangu i potpunom kompresijom puta.

Gustina	Prosečan broj grana	PVI			
		Osnovni	UBR	UBR i dPC	UBR i PC
0.02	10415	1.116	0.246	0.252	0.255
0.04	20855	1.075	0.249	0.251	0.251
0.06	31449	1.155	0.269	0.268	0.269
0.08	41890	1.040	0.255	0.255	0.251
0.1	52340	1.154	0.278	0.272	0.271

Tabela 4.3: Rezultati analize Kruskalovog algoritma na grafovima sa 1024 čvora

U tabeli 4.3 prikazana su prosečna vremena izvršavanja ove četiri verzije Kruskalovog algoritma. Kao što smo i očekivali osnovni algoritam je znatno sporiji od ostala tri u kojima smo primenili neki od oblika optimizacije. Njegovo vreme izvršavanja je oko četiri puta duže od vremena izvršavanja poboljšanih verzija algoritma.



Slika 4.3: Rezultati Kruskalovog algoritma na grafovima sa 1024 čvora

Poboljšane verzije imaju približno jednako vreme izvršavanja i na osnovu ovih rezultata ne možemo da zaključimo koji od njih je brži u opštem slučaju.

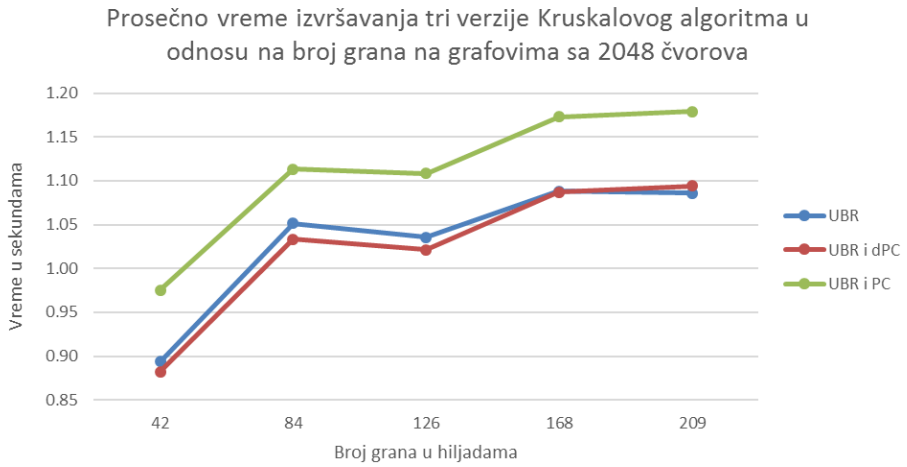
Na Slici 4.3 možemo da vidimo da se vreme izvršavanja nužno ne povećava sa povećanim brojem grana u grafu, za razliku od Primovog i Borůvkinog algoritma gde je veza između broja grana i vremena izvršavanja očigledna.

U tabeli 4.4, gde su prikazana vremena izvršavanja četiri verzije Kruskalovog algoritma na grafovima sa 2048 čvorova, možemo da vidimo da je osnovni algoritam čak do 5.5 puta sporiji od ostala tri.

Gustina	Prosečan broj grana	PVI			
		Osnovni	UBR	UBR i dPC	UBR i PC
0.02	41835	4.450	0.894	0.882	0.976
0.04	84000	5.590	1.052	1.034	1.113
0.06	125636	4.877	1.035	1.022	1.109
0.08	167799	5.439	1.088	1.087	1.173
0.1	209433	5.261	1.086	1.094	1.179

Tabela 4.4: Rezultati analize Kruskalovog algoritma na grafovima sa 2048 čvorova

Na Slici 4.4 vidimo da je algoritam *UBR i PC* najsporiji, a u njemu su uključena oba oblika optimizacije, unija po rangu i kompresija puta. Između vremena izvršavanja algoritama *UBR* i *UBR i dPC* razlika je minimalna.



Slika 4.4: Rezultati Kruskalovog algoritma na grafovima sa 2048 čvorova

Na osnovu ovih analiza vidimo da je implementacija unije po rangu doprinela boljem rezultatu algoritma ali kada smo dodali i delimičnu kompresija puta nismo došli do bržeg određivanja MPS. Takođe vidimo da ukoliko pored unije po rangu implementiramo i kompresiju puta u obliku u kojem smo je opisali u teorijskom delu rada dobijamo slabije rezultate.

Uporedimo sada vremena izvršavanja na grafovima sa 1024 čvora i prosečnim brojem grana 41890 sa vremenima izvršavanja na grafovima sa 2048 čvorova i prosečnim brojem grana 41835.

Broj čvorova	Prosečan broj grana	PVI			
		Osnovni	UBR	UBR i dPC	UBR i PC
1024	41890	1.040	0.255	0.255	0.251
2048	41835	4.450	0.894	0.882	0.976

Tabela 4.5: Poređenje rezultata na grafovima sa 42 hiljade grana

Ulazni grafovi imaju približno isti broj grana, dok se vremena drastično razlikuju, što možemo da vidimo u Tabeli 4.5.

Možemo da zaključimo da je vreme potrebno za određivanje MPS Kruskalovim algoritmom znatno duže na grafovima sa više čvorova, dok broj grana nema značajan uticaj na vreme izvršavanja.

Glava 5

Algoritam za pronalaženje minimalnog pokrivajućeg stabla u linearnom vremenu

5.1 Nasumični algoritam

Algoritam koji ćemo predstaviti u ovom delu su opisali Karger, Klein i Tarjan u [6]. Ponovo pretpostavljamo da su sve težine grana međusobno različite. Ovaj algoritam je specifičan po tome što ako ga pokrenemo više puta za isti ulazni graf algoritam uvek vraća isto rešenje ali vreme izvršavanja može da bude različito.

U [7] mogu se naći sledeći rezultati vezani za vreme izvršavanja nasumičnog algoritma. Neka je dat proizvoljan povezan neusmeren graf G sa m grana i n čvorova. Za svaki takav graf G nasumični algoritam ima linearno očekivano vreme izvršavanja: $O(m+n)$. Ali postoji mogućnost da će vreme izvršavanja algoritma biti čak $O(m \log n + n^2)$. Ovo znači da postoji broj c takav da kada se algoritam izvršava na grafu G dovoljan broj puta tada je prosečno vreme izvršavanja algoritma manje ili jednako od $c * (m+n)$ jedinica vremena. Ali postoji i broj d takav da će se algoritam sigurno izvršiti za manje od $d * (m \log n + n^2)$ jedinica vremena.

5.1.1 Opis algoritma

Ovaj algoritam je rekurzivan. Grafovi na kojima se primenjuje rekurzija mogu da budu i nepovezani, čak i ako je početni graf povezan. Zato ćemo algoritam opisati kao algoritam za pronalaženje minimalne pokrivajuće šume grafa koji ne mora da bude povezan.

Definicija 20 ***Pokrivajuća šuma*** *grafa $G = (V, E)$ je podgraf koji čine pokrivajuća stabla svake od povezanih komponenti početnog grafa.*
Minimalna pokrivajuća šuma *(MPŠ) je pokrivajuća šuma koja ima najmanju težinu.*
Podšuma *grafa $G = (V, E)$ je podgraf grafa G koji je šuma, tj. to je podgraf koji ne sadrži konture.*

Podšuma grafa ne mora da bude pokrivajuća.

Prethodno smo veličinu $w_T(e)$ definisali na stablu T. Šta ako T nije stablo nego šuma? U tom slučaju definiciju $w_T(e)$ možemo da dopunimo tako da važi i za šume:

Definicija 21 *Neka je $G = (V, E, W)$ težinski neusmeren graf i neka je F podšuma od G. Tada za granu $e = uv \in E$ definišemo veličinu $w_F(e)$ koja predstavlja težinu najteže grane na jedinstvenom putu u F koja povezuje u i v ako su u i v u istom stablu šume F, a ako nisu onda je $w_F(e) = +\infty$.*

Definicija 22 (F-teške i F-lake grane) *[6] Neka je $G = (V, E, w)$ težinski neusmeren graf i neka je F podšuma grafa G. Kažemo da je grana $e \in E$* ***F-teška*** *ako i samo ako je $w(e) > w_F(e)$, a u suprotnom kažemo da je E* ***F-laka*** *grana.*

Ako je e u F ili povezuje dva različita stabla u F onda je e F-laka grana.

Lema 7 *[6] Neka je $G = (V, E, w)$ težinski neusmeren graf i neka je F proizvoljna podšuma grafa G. Tada MPŠ grafa G ne sadrži ni jednu F-tešku granu.*

Dokaz. Sve F-teške grane možemo da obojimo u crveno koristeći crveno pravilo.

■

Nasumični algoritam koji želimo da opišemo oslanja se na sledeću 'lemu uzorkovanja'.

Lema 8 *[6] Neka je $G = (V, E, w)$ težinski neusmeren graf i neka je $0 < p < 1$. Neka je $G' = (V, E', w)$ proizvoljan podgraf grafa G koji je dobijen tako što je svaka grana u grafu G' dodata sa verovatnoćom p, nezavisno od ostalih grana. Neka je F MPŠ grafa G'. Tada je očekivani broj F-lakih grana u G najviše $\frac{n}{p}$, gde je $n = |V|$.*

Primetimo da očekivani broj F-lakih grana zavisi samo od n, broja čvorova, a ne od m, broja grana u početnom grafu G.

Dokaz. Prvo ćemo da napišemo pomoćni algoritam na kojem se zasniva dokaz leme. Kao rezultat ovog algoritma dobijamo proizvoljan podgraf početnog grafa, MPŠ podgrafa i skup F-lakih grana početnog grafa.

Pomoćni algoritam:

1. Poređati grane grafa G u neopadajući niz po težinama, odnosno tako da važi: $w(e_1) \leq w(e_2) \leq \cdots \leq w(e_m)$.

2. E', F, $L = \emptyset$.

3. **for** i=1:m
   Sa verovatnoćom p: $E' = E' \cup \{e_i\}$
   if $\{e_i\}$ povezuje dva različita stabla u F
   $L = L \cup \{e_i\}$
   if $\{e_i\} \in E'$
   $F = F \cup \{e_i\}$

Prvi korak pomoćnog algoritma je jednostavan: poređamo grane u neopadajući niz po težinama. Drugi korak predstavlja inicijalizaciju: definišemo skupove koji će nam biti potrebni i za početak oni su prazni skupovi. U svakoj iteraciji *for* petlje posmatramo jednu od grana početnog grafa G i sa verovatnoćom p je stavljamo u E'. Na ovaj način ćemo u skupu E' na kraju algoritma imati sve grane koje čine podgraf G'. U *if* naredbi koristeći Kruskalov algoritam dobijamo MPŠ F podgrafa $G' = (V, E')$. Pored toga što konstruišemo MPŠ F, u telu *if* naredbe konstruišemo i skup grana $L \subseteq E$ koje čine F-lake grane početnog grafa $G = (V, E)$.

Kao i Kruskalov algoritam, i ovaj algoritam obrađuje grane u rastućem redosledu po težinama. Svaku granu e_i dodajemo u E' sa verovatnoćom p. Ako e_i povezuje dva različita stabla u F (odnosno povezuje dve različite povezane komponente u F) onda je e_i F-laka grana i dodajemo je u L. Primetimo da je u tom slučaju e_i F-laka grana bez obzira da li smo je prethodno dodali u E' ili ne. U trenutku obrađivanja grane e_i ona povezuje dva različita stabla u F. Težina grane e_i je veća od težine bilo koje grane koja je trenutno u F. Ako dva stabla koja povezuje e_i ne bi bila povezana u jedno stablo u krajnjoj MPŠ F, tada bi e_i bila F-laka grana. Ako bi ova dva stabla na kraju bila povezana kao rezultat dodavanja grane e_i u E' pa i u F, onda bi e_i bila grana koja pripada šumi, a time bi ona bila i F-laka. Ako grana e_i ne bi bila dodata u F, a dva stabla koja ona povezuje bi na kraju ipak bila povezana, tada bi grana koja bi ih povezivala bila teža od e_i, pa bi onda e_i opet bila F-laka

grana. Primetimo i da ako su čvorovi koje povezuje grana e_i u istom stablu grafa F u trenutku obrađivanja grane e_i, tada je e_i F-teška grana jer su sve grane koje su trenutno u F lakše od e_i.

Pošto je gore navedeni algoritam slučajan, sledi da su $|L|$ i $|F|$ slučajne promenljive. Kada neku granu e dodamo u L, nju takođe dodajemo u F sa verovatnoćom p. Sledi da je očekivani broj grana u L jednak očekivanom broju grana u L pomnožen sa p, $\mathbb{E}[|F|] = p\mathbb{E}[|L|]$.

Pošto je F MPŠ grafa $G' = (V, E')$ sledi da je broj grana u F strogo manji od ukupnog broja čvorova u G', a to je n. Dakle imamo da je $|F| \leq n - 1$ i da je $\mathbb{E}[|F|] \leq n - 1$.

Sada imamo da je

$$p\mathbb{E}[|L|] = \mathbb{E}[|F|] \leq n - 1$$

odnosno

$$\mathbb{E}[|L|] \leq \frac{n-1}{p} \leq \frac{n}{p}$$

što smo i trebali pokazati.

■

Sada ćemo da opišemo nasumični algoritam za pronalaženje MPŠ (NasumičniMPŠ).

Ako je skup grana E početnog grafa prazan, onda je i MPŠ takođe prazan skup. U suprotnom, izvršava se sledeći algoritam koji ćemo opisati u vidu pseudo-koda radi lakšeg pregleda, a nakon toga ćemo detaljno opisati svaki korak.

1. $(G', F') = BoruvkaKorak(G)$
2. $(G'', F'') = BoruvkaKorak(G')$
3. $G_1 = ProizvoljanPodgraf(G'', \frac{1}{2})$
4. $F_1 = NasumičniMPŠ(G_1)$
5. $E_2 = LakeGrane(G'', F_1)$
6. $G_2 = (V[G''], E_2, w)$
7. $F_2 = NasumičniMPŠ(G_2)$
8. Rešenje: $F' \cup F'' \cup F_2$

Prvo primenimo dva uzastopna Borůvkina koraka (koraci 1 i 2). Pod Borůvkinim korakom podrazumevamo sledeće: za svaki čvor odabrati granu minimalne težine koja je incidentna sa posmatranim čvorom. Zatim "kontrakovati" sve odabrane grane. To znači da svaku povezanu komponentu dobijenog grafa treba da zamenimo jednim superčvorom, obrišemo izolovane čvorove, grane koje povezuju čvorove koji pripadaju istom superčvoru i sve grane koje povezuju različite superčvorove osim onih koje imaju minimalnu težinu. Ovim ćemo smanjiti broj čvorova bar 4 puta, dokaz je dat u Lemi 9.

Sa F' i F'' ćemo označiti skup grana koje smo kontrakovali u ove dve iteracije Borůvkinog koraka (F' - skup grana kontrakovanih u prvoj iteraciji, F'' - skup grana kontrakovanih u drugoj iteraciji). Sa G'' ćemo označiti dobijeni kontrakovani graf. U koraku 3 biramo proizvoljan podgraf $G_1 = (V, E_1)$ kontrakovanog grafa G'' u koji svaku granu dodajemo sa verovatnoćom $\frac{1}{2}$, nezavisno od ostalih grana. U koraku 4 prvi put rekurzivno pozivamo nasumični algoritam, i to za graf G_1 da bismo odredili MPŠ F_1 grafa G_1. Zatim u koraku 5 koristimo linearni algoritam za određivanje skupa lakih grana E_2 grafa G'' (Dixon-Rauch-Tarjan, 1992., $O(m)$). Sve ostale grane možemo da uklonimo koristeći pravilo konture (crveno pravilo). Sada definišemo graf G_2: čvorovi ovog grafa su čvorovi grafa G''; grane ovog grafa su lake grane koje su u skupu E_2, a dobijene u prethodnom koraku; težine grana su težine grana iz početnog grafa. U koraku 7 za graf G_2 rekurzivno pozovemo nasumični algoritam da bismo dobili njegovu MPŠ F_2. Na kraju imamo da skup grana $F' \cup F'' \cup F_2$ čini MPŠ početnog grafa $G = (V, E)$.

Lema 9 *U svakoj iteraciji Borůvkinog koraka broj stabala u G se bar prepolovi.*

Dokaz. Svako stablo (tj. superčvor) se u svakoj iteraciji povezuje sa bar jednim od svojih suseda, pa svako novo stablo sadrži bar 2 ili više 'starih' stabala (superčvorova).

■

5.1.2 Ispravnost algoritma

Tvrđenje 13 *[6] NasumičniMPŠ algoritam vraća MPŠ ulaznog grafa.*

Dokaz. Ispravnost algoritma sledi iz pravila reza i konture (plavo i crveno pravilo).

Sve grane koje su u F' i F'' a koje su odabrane primenom dva Borůvkina koraka možemo da obojimo u plavo koristeći pravilo reza i zbog toga one pripadaju MPŠ.

Sve F_1-teške grane, a koje su uklonjene iz G_2 pre drugog rekurzivnog poziva, možemo da obojimo u crveno koristeći pravilo konture i zato one ne pripadaju MPŠ.

Ostatak pokazujemo indukcijom.

■

5.1.3 Kompleksnost algoritma

Tvrđenje 14 *[6] Prosečno vreme izvršavanja algoritma NasumičniMPŠ je $O(m+n)$.*

Dokaz. Neka je $G = (V, E)$ proizvoljan graf i neka je $m = |E|$ broj grana i $n = |V|$ broj čvorova u grafu G. Pošto je graf G proizvoljan, sledi da su m i n slučajne promenljive. Neka je $\bar{m} = \mathbb{E}[m]$ očekivani broj grana i $\bar{n} = \mathbb{E}[n]$ očekivani broj čvorova. Neka su grafovi $G_1 = (V_1, E_1)$ i $G_2 = (V_2, E_2)$ proizvoljni grafovi za koje smo rekurzivno pozvali NasumičniMPŠ algoritam prilikom izvršavanja istog algoritma na grafu G. Neka su $m_i = |E_i|$, $n_i = |V_i|$, $\bar{m_i} = \mathbb{E}[m_i]$, $\bar{n_i} = \mathbb{E}[n_i]$ za $i = 1, 2$.

Pošto su grafovi G_1 i G_2 podgrafovi grafa G'' a koji je dobijen primenom dva uzastopna Borůvkina koraka na G, a na osnovu Leme 9 sledi da je $n_1, n_2 \leq \frac{n}{4}$, a odatle sledi i da je $\bar{n_1}, \bar{n_2} \leq \frac{\bar{n}}{4}$.

Graf G_1 je dobijen od grafa G'' tako što je svaka grana izabrana nezavisno od ostalih grana, sa verovatnoćom $\frac{1}{2}$, pa sledi da je $\bar{m_1} \leq \frac{\bar{m}}{2}$. Takođe znamo da važi da je $m_1 < m$ jer su neke grane kontrakovane prilikom primene koraka Borůvkinog algoritma. Na osnovu 'leme uzorkovanja' sledi da je $\bar{m_2} \leq \bar{n_2}/\frac{1}{2} \leq (\frac{\bar{n}}{4})/(\frac{1}{2}) = \frac{\bar{n}}{2}$.

Neka je $T(G)$ vreme izvršavanja algoritma na proizvoljnom grafu G. Tada $T(G)$ čine: vreme potrebno za izvršavanje dva uzastopna Borůvkina koraka, vreme potrebno za izvršavanje algoritma koji pronalazi lake grane grafa G'' i vreme za izvšavanje istog ovog algoritma na grafovima G_1 i G_2:

$$T(G) = a(m+n) + T(G_1) + T(G_2),$$

gde je $a > 1$ konstanta koja je dovoljno velika da bi $a(m+n)$ pokrio linearno vreme izvršavanja dva Borůvkina koraka i izvršavanje algoritma koji pronalazi lake grane $LakeGrane(G'', F_1)$.

Pokazaćemo indukcijom po m i n da ako je G proizvoljan graf sa najviše m grana i n čvorova, da onda važi

$$\mathbb{E}[T(G)] \leq 2a(\bar{m} + 2\bar{n}).$$

Baza indukcije: Ako nakon dva uzastopna koraka Borůvkinog algoritma dobijemo MPŠ početnog grafa G, tada je $T(G) = a(m+n)$ pa važi tvrđenje.

Induktivna pretpostavka: Neka važi tvrđenje: $\mathbb{E}[T(G)] \leq 2a(\bar{m} + 2\bar{n})$ za sve vrednosti manje od m i n.

Induktivni korak: Neka je G proizvoljan graf sa najviše m grana i n čvorova. Induktivna pretpostavka važi za grafove G_1 i G_2 (jer oni sigurno imaju manje grana od G), pa imamo da je:

$$\begin{aligned}
\mathbb{E}[T(G)] &= a(\bar{m} + \bar{n}) + \mathbb{E}[T(G_1)] + \mathbb{E}[T(G_2)] \\
&\leq a(\bar{m} + \bar{n}) + 2a(\bar{m_1} + 2\bar{n_1}) + 2a(\bar{m_2} + 2\bar{n_2}) \\
&\leq a(\bar{m} + \bar{n}) + 2a(\frac{\bar{m}}{2} + 2 \cdot \frac{\bar{n}}{4}) \\
&= 2a(\bar{m} + 2\bar{n}).
\end{aligned}$$

■

Glava 6

Primena MPS na deviznom tržištu

U ovom delu rada opisaćemo jednu od mnogobrojnih primena MPS, a to je primena na deviznom tržištu. Na deviznom tržištu se trguje stranim valutama i ono predstavlja najveće i najlikvidnije finansijsko tržište. Prosečan dnevni promet na globalnom deviznom tržištu se povećava iz godine u godinu i procenjuje se da je 2013. iznosio \$5.3 biliona što predstavlja rast od oko 35% u odnosu na \$4 biliona u 2010. (videti [3]). Mi ćemo se baviti analizom deviznih kurseva u cilju razumevanja međusobnog položaja različitih valuta. Devizni kurs je cena jedne valute izražena u drugoj valuti i ima značajnu ulogu u međunarodnoj razmeni svih država i multinacionalnih kompanija jer utiče na nivo izvoza i uvoza, investitorima i menadžerima fondova omogućava pristup inostranim hartijama od vrednosti a takođe može da predstavlja i izvor zarade.

U prvom delu ćemo opisati kako se od istorijskih podataka deviznih kurseva može dobiti graf za koji ćemo zatim odrediti MPS pomoću Kruskalovog algoritma u R-u. U drugom delu ćemo prikazati i analizirati rezultate.

6.1 Od deviznog kursa do grafa

Posmatraćemo dnevni devizni kurs 18 valuta u odnosu na evro (EUR) u 2015. godini. Istorijski podaci dobijeni su sa stranice `http://www.usforex.com`, a spisak posmatranih valuta dat je u tabeli 6.1.

Postavlja se pitanje kako od istorijskih podataka dobiti graf koji prikazuje veze između valuta. Koristićemo metodologiju koja je opisana u [11] a koja predlaže da svaki od čvorova u grafu predstavlja jednu od valuta. Grane će predstavljati vezu između valuta i želimo da valute koje su 'bliske' budu

povezane granom manje težine.

AUD	Australijski dolar	JPY	Japanski jen
CAD	Kanadski dolar	KRW	Južnokorejski von
CHF	Švajcarski franak	NOK	Norveška kruna
CNY	Kineski juan	NZD	Novozelandski dolar
CZK	Češka kruna	PLN	Poljski zlot
DKK	Danska kruna	RUB	Ruska rublja
GBP	Britanska funta	SEK	Švedska kruna
HKD	Hongkong dolar	SGD	Singapurski dolar
HUF	Mađarska forinta	USD	Američki dolar

Tabela 6.1: Spisak posmatranih nacionalnih valuta sa simbolima

Težinu grana dobijamo na sledeći način:

1. Izračunati logaritamski prinos za svaku od valuta pomoću formule $r_i^t = \ln K_i^t - \ln K_i^{t-1}$, gde je K_i^t kurs i-te valute u odnosu na EUR na dan t. Koristimo logaritamske prinose jer su za njih ispunjene pretpostavke koje treba da važe da bismo mogli koristiti Pirsonov koeficijent korelacije u sledećem koraku. Pretpostavke su: normalnost, linearnost i homoskedastičnost.

2. Na osnovu logaritamskih prinosa izračunati koeficijent korelacije

$$\rho_{ij} = \frac{\langle r_i r_j \rangle - \langle r_i \rangle \langle r_j \rangle}{\sqrt{(\langle r_i^2 \rangle - \langle r_i \rangle^2)(\langle r_j^2 \rangle - \langle r_j \rangle^2)}}$$

   gde je $\langle \ldots \rangle$ prosečna vrednost za posmatrani period.

3. Izračunati udaljenosti odnosno težine grana pomoću formule $d_{ij} = \sqrt{2(1-\rho_{ij})}$. Dakle, d_{ij} su elementi težinske matrice D.

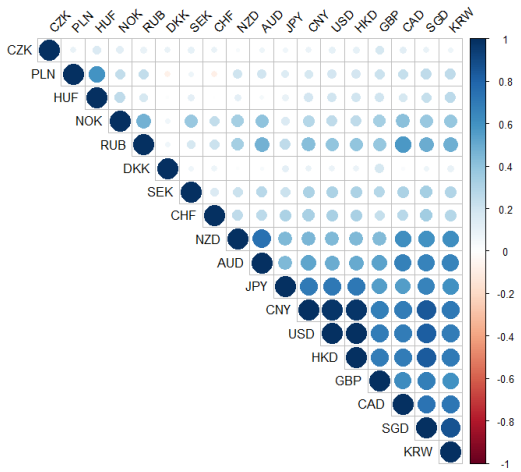
Podsetimo se, koeficijent korelacije ρ je broj između -1 i 1 i pokazuje u kojoj meri su promene vrednosti jedne promenljive povezane sa promenama vrednosti druge promenljive. -1 označava savršenu negativnu korelaciju a 1 savršenu pozitivnu korelaciju. Koeficijent korelacije se uglavnom tumači na sledeći način: vrednosti između -0.25 i 0.25 znače da nema povezanosti između promenljivih, od 0.25 do 0.5 i od -0.5 do -0.25 slaba povezanost, od 0.5 do 0.75 i od -0.75 do -0.5 umerena povezanost, od 0.75 do 1 i od -1 do -0.75 jaka povezanost. U našem primeru koeficijent korelacije govori u kojoj meri su promene logaritamskih prinosa jednog valutnog para, npr.

USDEUR, povezane sa promenama logaritamskih prinosa drugog valutnog para, npr. JPYEUR. U tabeli 6.2 ovu vrednost možemo da nađemo ispod glavne dijagonale, za posmatrani primer USDEUR i JPYEUR iznosi 0.71.

	USD	SGD	SEK	PLN	NZD	NOK	RUB	CNY	KRW	JPY	HUF	HKD	GBP	DKK	CZK	CHF	CAD	AUD
USD		***	***	***	***	***	***	***	***	***	***	***	***	**	,	***	***	***
SGD	0.83		***	***	***	***	***	***	***	***	***	***	***	*	,	***	***	***
SEK	0.32	0.34			***	***	**	***	***	***	*	***	***	,	,	**	***	***
PLN	0.19	0.26	0.07		***	***	***	**	***	*	***	***	***		,	*	***	***
NZD	0.44	0.60	0.22	0.20		***	***	***	***	***	*	***	***			***	***	***
NOK	0.26	0.38	0.37	0.25	0.34		***	***	***	**	***	***	***		*	**	***	***
RUB	0.40	0.50	0.17	0.24	0.34	0.47		***	***	***	**	***	***		,	*	***	***
CNY	0.97	0.84	0.31	0.17	0.45	0.29	0.42		***	***	**	***	***	**		***	***	***
KRW	0.70	0.86	0.30	0.25	0.61	0.38	0.48	0.72		***	***	***	***	*		***	***	***
JPY	0.71	0.66	0.21	0.14	0.44	0.15	0.25	0.70	0.61		,	***	***	*		***	***	***
HUF	0.20	0.23	0.12	0.59	0.11	0.26	0.18	0.18	0.26	0.10		***	***		**		**	
HKD	1.00	0.83	0.32	0.19	0.44	0.26	0.40	0.97	0.70	0.71	0.20		***	**	,	***	***	***
GBP	0.69	0.68	0.28	0.22	0.43	0.33	0.38	0.69	0.61	0.56	0.19	0.69		***	**	***	***	***
DKK	0.09	0.07	0.07	-0.08	0.02	0.06	0.07	0.09	0.10	0.12	0.02	0.09	0.18			,		
CZK	0.10	0.10	0.10	0.10	0.04	0.12	0.10	0.09	0.07	0.07	0.15	0.10	0.17	0.09			*	
CHF	0.32	0.35	0.15	-0.08	0.25	0.24	0.21	0.33	0.29	0.31	0.00	0.32	0.24	0.08	0.09		***	***
CAD	0.69	0.74	0.31	0.23	0.62	0.42	0.58	0.69	0.73	0.55	0.17	0.70	0.63	0.03	0.12	0.28		***
AUD	0.50	0.66	0.27	0.20	0.75	0.40	0.47	0.52	0.66	0.45	0.05	0.50	0.53	0.04	0.07	0.26	0.68	

Tabela 6.2: Korelacije i p-vrednost

Koeficijent korelacije možemo da prikažemo grafički, na primer kao na slici 6.1, gde je plavom bojom prikazana pozitivna korelacija a crvenom bojom negativna korelacija. Što je krug veći to je korelacija jača.



Slika 6.1: Grafički prikaz korelacija

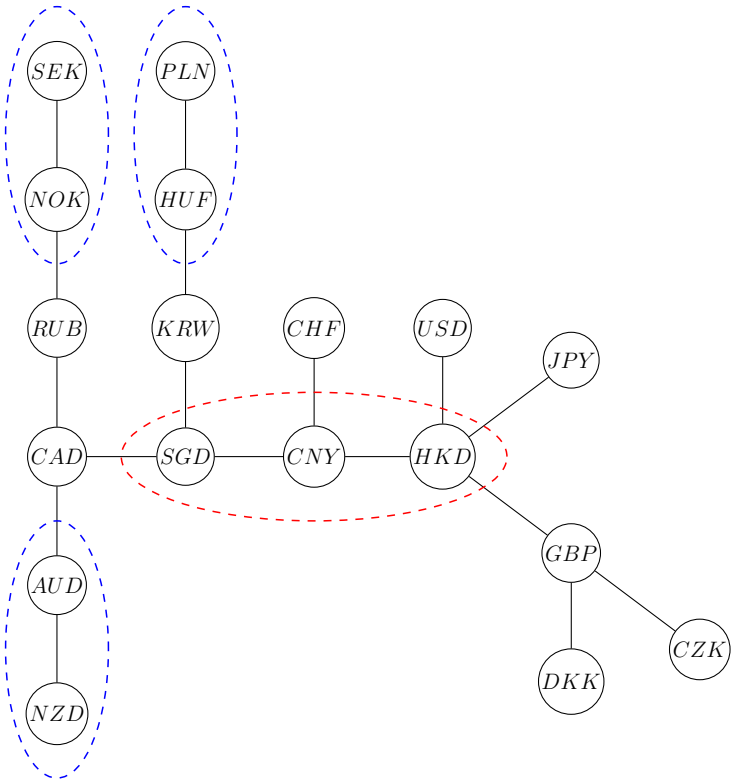
Iznad glavne dijagonale u tabeli 6.2 označene su p-vrednosti. P-vrednost daje odgovor na pitanje da li je korelacija između dve promenljive statistički

značajna. Sa tri zvezdice označena je p-vrednost <0.001, sa dve zvezdice <0.01, sa jednom zvezdicom <0.05 i sa zarezom <0.1. Vidimo da je korelacija između USDEUR i JPYEUR statistički značajna (p-vrednost je <0.01).

P-vrednost je izuzetno važna jer ukoliko je ona veća od postavljene granice značajnosti, uglavnom 0.05, tada koeficijent korelacije nije statistički značajan. U primeni to bi značilo da ukoliko je neka od p-vrednosti veća od 0.05 tada ne bi trebalo da se oslanjamo na koeficijent korelacije, a samim tim ni na težinu grane koja povezuje odgovarajuće valute.

6.2 MPS

Vrednosti ρ_{ij} su između -1 i 1 pa sledi da će težine grana d_{ij} biti između 0 i 2. Matrica D će biti simetrična matrica sa nulama na dijagonali. Primetimo da ukoliko je korelacija negativna, na primer -1, tada će težina grane biti 2. Možemo da zaključimo da će MPS prikazati povezanost između onih valutnih parova između čijih logaritamskih prinosa postoji stroga pozitivna korelacija.



Slika 6.2: MPS dobijeno na osnovu logaritamskih prinosa 18 valuta

Na slici 6.2 prikazano je MPS koje smo odredili za 18 valuta. Autori rada

[11] zaključili su iz MPS koji je dobijen na osnovu podataka iz 1999 - 2012. da zemlje čija je privreda povezana sa Kinom, kao što su Hongkong i Singapur imaju centralnu poziciju. Na osnovu podataka za 2015. godinu možemo da vidimo da su ove valute u centru MPS, a pored njih i kanadski dolar. Takođe možemo da primetimo i grupe geografski bliskih zemalja, AUD i NZD, HUF i PLN, NOK i SEK. Interesantno je što većina listova u MPS predstavljaju valute evropskih država.

Ovim smo pokazali kako se prilično velika količina podataka, čije je razumevanje u neobrađenom obliku gotovo nemoguće, može prikazati na jednostavan i informativan način pomoću MPS.

Zaključak

Problem određivanja MPS definisao je češki matematičar Otakar Borůvka 1926. godine i opisao algoritam koji se izvršava u polinomnom vremenu. U ovom radu su pored Borůvkinog algoritma prikazani Primov i Kruskalov algoritam, koji se takođe izvršavaju u polinomnom vremenu. U radu je opisan i tzv. nasumični algoritam čije je vreme izvršavanja očekivano linearno.

Tri pomenuta klasična algoritma i tri verzije Kruskalovog algoritma koje uključuju različite optimizacije implementirali smo u programskom paketu *R*. Zatim smo za proizvoljne grafove pomoću implementiranih algoritama odredili MPS i analizirali dobijene rezultate. Cilj ove empirijske analize bio je da zaključimo na koji način povećanje broja grana i broja čvorova utiče na vreme izvršavanja ovih algoritama, a takođe i koji od ovih algoritama je najbrži. Zaključili smo da je Kruskalov algoritam znatno brži od Primovog i Borůvkinog algoritma. Kod Primovog i Borůvkinog algoritma sa većim brojem čvorova i grana povećava se i vreme izvršavanja dok kod Kruskalovog algoritma uticaj povećanja broja grana nije toliko očigledan. Dodavanjem jedne od optimizacija, unije po rangu, vreme izvršavanja Kruskalovog algoritma se značajno poboljšalo dok dodavanjem kompresije puta nismo dobili bolji rezultat.

Problem određivanja MPS je interesantan jer ima široku primenu, a u radu je opisana primena na deviznom tržištu. U cilju razumevanja međusobnog položaja različitih valuta analizirali smo istorijske podatke deviznih kurseva osamnaest valuta izraženih u EUR za 2015. godinu. Iz istorijskih podataka odredili smo MPS i zaključili smo da valute zemalja čija je privreda povezana sa Kinom, kao što su Hongkong i Singapur, a pored njih i kanadski dolar imaju centralnu poziciju. Takođe smo videli da su valute geografski bliskih zemalja bliske i u dobijenom MPS.

Prilog

Kodovi koji su dati ispod napisani su u programskom paketu *R* i predstavljaju implementaciju tri klasična algoritma za određivanje MPS: Borůvkin, Primov i Kruskalov. Takođe je dat kod za tri verzije poboljšanog Kruskalovog algoritma, u svakom od njih je implementirana neka vrsta optimizacije. Data su i dva pomoćna algoritma, jedan za kreiranje proizvoljnih povezanih, neusmerenih grafova sa različitim težinama grana i jedan za određivanje povezanih komponenti grafa.

Kreiranje proizvoljnih grafova:

```
randomgraph <- function(brojcvorova, maxtezina, gustina) {
  brojcvorova <- brojcvorova*brojcvorova
  brojevi <- rbinom(brojcvorova,1,gustina)*round(runif(brojcvorova,0,maxtezina))
  G <- matrix(brojevi, nrow = sqrt(brojcvorova), ncol = sqrt(brojcvorova))
  G[lower.tri(G)] <- 0
  diag(G) <- 0
  cvorovi <- 1:nrow(G)
  grane <- matrix(ncol=3)[-1,]
  for (i in 1:nrow(G)) {
    for (j in i:nrow(G)) {
      if (G[i, j] > 0) {
        grane <- rbind(grane, matrix(c(i,j,G[i,j]),ncol=3))
      }
    }
  }
  grane <<- grane[!duplicated(grane[,3]),]
  cvorovi <<- cvorovi
}
```

Borůvkin algoritam:

```
Boruvka <- function (cvorovi, grane) {
  t0 <- Sys.time()
  cvorovi <- matrix(c(cvorovi, cvorovi), ncol = 2)
  MPS <- matrix(ncol = 3)[-1, ]
  while (length(unique(cvorovi[,2]) ) > 1) {
```

```
    pomocni <- matrix(ncol = 5)[-1,]
    for (i1 in 1:length(unique(cvorovi[,2]))) {
      i <- cvorovi[cvorovi[,2]==i1,1]
      j <- cvorovi[cvorovi[,2]!=i1,1]
      vaznegrane <- matrix(grane[(grane[, 1] %in% i & grane[, 2] %in% j) |
                                 (grane[, 2] %in% i & grane[, 1] %in% j)], ncol = 3)
      mingrana <- matrix(vaznegrane[vaznegrane[, 3] == min(vaznegrane[, 3]), ], ncol
          = 3)[1, ]
      if (mingrana[1] > mingrana[2]) {
        mingrana[c(1,2)] <- mingrana[c(2,1)]
      }
      mingrana <- matrix(c(mingrana, cvorovi[mingrana[1],2], cvorovi[mingrana[2],2]),
          ncol = 5)
      pomocni <- rbind(pomocni, mingrana)
    }
    pomocni <- unique(pomocni)
    MPS <- rbind(MPS, pomocni[,1:3])
    komponente(unique(cvorovi[,2]), pomocni[,4:5])
    cvorovi <- cbind(cvorovi, 0)
    for (i in cvorovi[,1]) {
      cvorovi[i,3] <- povkomp[cvorovi[i,2],2]
    }
    cvorovi <- cvorovi[,-2]
  }
  MPSBoruvka <<- MPS
  t1 <- Sys.time()
  vreme_izvrsavanja <<- t1 - t0
}
```

Određivanje povezanih komponenti:

```
komponente <- function (cvorovi, grane) {
  komp <- 1
  povkomp <- cbind(cvorovi,0)
  grane <- matrix(grane, ncol = 2)
  while (length(grane) > 1) {
    i <- grane[1]
    stack <- c(i)
    povkomp[i,2] <- komp
    while (length(stack) > 0) {
      i <- stack[1]
      sused <- c(grane[grane[,1]==i,2], grane[grane[,2]==i,1]) [1]
      if (!is.na(sused) > 0) {
        row <- which(grane[,1]==i & grane[,2]==sused)
        if (length(row) < 1) {row <- which(grane[,1]==sused & grane[,2]==i)}
        grane <- matrix(grane[-row,],ncol=2)
        stack <- c(sused, stack)
        povkomp[sused,2] <- komp
      } else {
```

```
        stack <- stack[-1]
      }
    }
    komp <- komp + 1
  }
  povkomp <<- povkomp
}
```

Primov algoritam:

```
Prim <- function(cvorovi, grane) {
  t0 <- Sys.time()
  cvorovi <- cbind(cvorovi, 1)
  MPS <- matrix(ncol = 3)[-1, ]
  cvorovi[1,2] <- 0
  while (sum(cvorovi[,2] > 0)) {
    i <- cvorovi[cvorovi[,2]==1,1]
    j <- cvorovi[cvorovi[,2]==0,1]
    vaznegrane <- matrix(grane[(grane[, 1] %in% i & grane[, 2] %in% j) |
                               (grane[, 2] %in% i & grane[, 1] %in% j)], ncol = 3)
    mingrana <- matrix(vaznegrane[vaznegrane[, 3] == min(vaznegrane[, 3]), ], ncol =
        3)[1, ]
    cvorovi[mingrana[1],2] <- 0
    cvorovi[mingrana[2],2] <- 0
    MPS <- rbind(MPS, mingrana)
  }
  MPSPrim <<- MPS
  t1 <- Sys.time()
  vreme_izvrsavanja <<- t1 - t0
}
```

Osnovni Kruskalov algoritam:

```
Kruskalosnovni <- function(cvorovi, grane) {
  t0 <- Sys.time()
  grane <- grane[sort.list(grane[,3], method = "quick", na.last = NA),]
  cvorovi <- matrix(c(cvorovi, cvorovi), ncol = 2)
  MPS <- matrix(ncol = 3)[-1, ]
  i <- 1
  while (length(MPS[,1]) < (length(cvorovi[,1]) -1)) {
    u <- grane[i,1]
    v <- grane[i,2]
    pu <- u
    while (pu != cvorovi[pu,2]) {
      pu <- cvorovi[pu,2]
    }
    pv <- v
    while (pv != cvorovi[pv,2]) {
      pv <- cvorovi[pv,2]
```

```
    }
    if (pu != pv) {
      MPS <- rbind(MPS, grane[i,])
      cvorovi[pv,2] <- pu
    }
    i <- i+1
  }
  MPSKruskalosnovni <<- MPS
  t1 <- Sys.time()
  vreme_izvrsavanja <<- t1 - t0
}
```

Kruskal UBR:

```
KruskalUBR <- function(cvorovi, grane) {
  t0 <- Sys.time()
  grane <- grane[sort.list(grane[,3], method = "quick", na.last = NA),]
  cvorovi <- cbind(matrix(c(cvorovi, cvorovi), ncol = 2), 0)
  MPS <- matrix(ncol = 3)[-1, ]
  i <- 1
  while (length(MPS[,1]) < (length(cvorovi[,1]) -1)) {
    u <- grane[i,1]
    v <- grane[i,2]
    pu <- u
    while (pu != cvorovi[pu,2]) {
      pu <- cvorovi[pu,2]
    }
    pv <- v
    while (pv != cvorovi[pv,2]) {
      pv <- cvorovi[pv,2]
    }
    if (pu != pv) {
      MPS <- rbind(MPS, grane[i,])
      if (cvorovi[pu,3] > cvorovi[pv,3]) {
        cvorovi[pv,2] <- pu
      } else if (cvorovi[pu,3] < cvorovi[pv,3]) {
        cvorovi[pu,2] <- pv
      } else {
        cvorovi[pv,2] <- pu
        cvorovi[pu,3] <- cvorovi[pu,3] + 1
      }
    }
    i <- i+1
  }
  MPSKruskalUBR <<- MPS
  t1 <- Sys.time()
  vreme_izvrsavanja <<- t1 - t0
}
```

Kruskal UBR i dPC:

```
KruskalUBRidPC <- function(cvorovi, grane) {
  t0 <- Sys.time()
  grane <- grane[sort.list(grane[,3], method = "quick", na.last = NA),]
  cvorovi2 <- cbind(matrix(c(cvorovi, cvorovi), ncol = 2), 0)
  MPS <- matrix(ncol = 3)[-1, ]
  i <- 1
  while (length(MPS[,1]) < (length(cvorovi2[,1]) -1)) {
    u <- grane[i,1]
    v <- grane[i,2]
    pu <- u
    while (pu != cvorovi2[pu,2]) {
      pu <- cvorovi2[pu,2]
    }
    cvorovi2[u,2] <- pu
    pv <- v
    while (pv != cvorovi2[pv,2]) {
      pv <- cvorovi2[pv,2]
    }
    cvorovi2[v,2] <- pv
    if (pu != pv) {
      MPS <- rbind(MPS, grane[i,])
      if (cvorovi2[pu,3] > cvorovi2[pv,3]) {
        cvorovi2[pv,2] <- pu
      } else if (cvorovi2[pu,3] < cvorovi2[pv,3]) {
        cvorovi2[pu,2] <- pv
      } else {
        cvorovi2[pv,2] <- pu
        cvorovi2[pu,3] <- cvorovi2[pu,3] + 1
      }
    }
    i <- i+1
  }
  MPSKruskalUBRidPC <<- MPS
  t1 <- Sys.time()
  vreme_izvrsavanja <<- t1 - t0
}
```

Kruskal UBR i PC:

```
KruskalUBRiPC <- function(cvorovi, grane) {
  t0 <- Sys.time()
  grane <- grane[sort.list(grane[,3], method = "quick", na.last = NA),]
  cvorovi <- cbind(matrix(c(cvorovi, cvorovi), ncol = 2), 0)
  MPS <- matrix(ncol = 3)[-1, ]
  i <- 1
  while (length(MPS[,1]) < (length(cvorovi[,1]) -1)) {
    u <- grane[i,1]
```

```
    v <- grane[i,2]
    pu <- cvorovi[u,2]
    pv <- cvorovi[v,2]
    if (pu != pv) {
      MPS <- rbind(MPS, grane[i,])
      if (cvorovi[pu,3] > cvorovi[pv,3]) {
        cvorovi[cvorovi[,2] == pv, 2] <- pu
      } else if (cvorovi[pu,3] < cvorovi[pv,3]) {
        cvorovi[cvorovi[,2] == pu, 2] <- pv
      } else {
        cvorovi[cvorovi[,2] == pv, 2] <- pu
        cvorovi[pu,3] <- cvorovi[pu,3] + 1
      }
    }
    i <- i+1
  }
  MPSKruskalUBRiPC <<- MPS
  t1 <- Sys.time()
  vreme_izvrsavanja <<- t1 - t0
}
```

Literatura

[1] Thomas H. Cormen, Charles Eric Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to algorithms*. MIT press, 2009.

[2] Igor Dolinka. *Kratak uvod u analizu algoritama*. Prirodno-matematički fakultet u Novom Sadu, 2008.

[3] Bank for International Settlements. Triennial Central Bank Survey Foreign exchange turnover in April 2013: preliminary global results. `http://www.bis.org/publ/rpfx13fx.pdf`, 2003. [Online; accessed 12-February-2016].

[4] Dejan Živković. *Uvod u algoritme i strukture podataka*. Univerzitet Singidunum, 2010.

[5] Neil F Johnson, Mark McDonald, Omer Suleman, Stacy Williams, and Sam Howison. What shakes the fx tree? understanding currency dominance, dependence, and dynamics (keynote address). In *SPIE Third International Symposium on Fluctuations and Noise*, pages 86–99. International Society for Optics and Photonics, 2005.

[6] David R Karger, Philip N Klein, and Robert E Tarjan. A randomized linear-time algorithm to find minimum spanning trees. *Journal of the ACM (JACM)*, 42(2):321–328, 1995.

[7] Jinna Lei. *Three minimum spanning tree algorithms*. Senior honors thesis, University of California, Berkeley, 2010.

[8] Martin Mareš. The saga of minimum spanning trees. *Computer Science Review*, 2(3):165–221, 2008.

[9] Dragan Mašulović. *Odabrane teme diskretne matematike*. Prirodno-matematički fakultet u Novom Sadu, 2007.

[10] Mark McDonald, Omer Suleman, Stacy Williams, Sam Howison, and Neil F Johnson. Detecting a currency's dominance or dependence using foreign exchange network trees. *Physical Review E*, 72(4):046106, 2005.

[11] Marcel Rešovskỳ, Denis Horváth, Vladimír Gazda, and Marianna Siničáková. Minimum spanning tree application in the currency market. *Biatec*, 21(7):21–23, 2013.

[12] Wikipedia. Foreign exchange market — wikipedia, the free encyclopedia, 2016. [Online; accessed 12-February-2016].

Biografija

Evelin Kurta rođena je 8. septembra 1990. godine u Somboru. Završila je osnovnu školu "Ivan Goran Kovačić" u Stanišiću 2005. godine i upisala Srednju ekonomsku školu u Somboru. Po završetku srednje škole, 2009. godine, upisala je osnovne akademske studije na Prirodno-matematičkom fakultetu u Novom Sadu, smer Primenjena matematika, modul Matematika finansija, koje je završila 2012. godine. U oktobru iste godine upisala je master studije na istom smeru. Položila je sve ispite predviđene nastavnim planom i programom master studija u oktobru 2014. godine i time stekla uslov za odbranu master rada.

U okviru TEMPUS projekta "Visuality and Mathematics", tokom 2014. godine provela je mesec dana u Egeru, Mađarska.

Od januara 2015. godine zaposlena je na poziciji analitičara kreditnog rizika u *Morgan Stanley Hungary Analytics Ltd.* u Budimpešti.

UNIVERZITET U NOVOM SADU
PRIRODNO - MATEMATIČKI FAKULTET
KLJUČNA DOKUMENTACIJSKA INFORMACIJA

Redni broj:
RBR
Identifikacioni broj:
IBR
Tip dokumentacije: Monografska dokumentacija
TD
Tip zapisa: Tekstualni štampani materijal
TZ
Vrsta rada: Master rad
VR
Autor: Evelin Kurta
AU
Mentor: prof. dr Maja Pech
MN
Naslov rada: Algoritmi za pronalaženje minimalnog pokrivajućeg stabla
NR
Jezik publikacije: Srpski (latinica)
JP
Jezik izvoda: s / e
JI
Zemlja publikovanja: Republika Srbija
ZP
Uže geografsko područje: Vojvodina
UGP
Godina: 2016.
GO
Izdavač: Autorski reprint
IZ
Mesto i adresa: Departman za matematiku i informatiku, Prirodno-matematički fakultet, Univerzitet u Novom Sadu, Trg Dositeja Obradovića 4, Novi Sad,
MA
Fizički opis rada: (6 glava, 63 strane, 7 tabela, 22 slike, 12 referenci, 1 prilog)
FO
Naučna oblast: Matematika
NO

Naučna disciplina: Teorija grafova
ND
Ključne reči: grafovi, minimalno pokrivajuće stablo, algoritmi, polinomno vreme, linearno vreme
PO
UDK
Čuva se: Biblioteka Departmana za matematiku i informatiku Prirodno-matematičkog fakulteta Univerziteta u Novom Sadu
ČU
Važna napomena:
VN
Izvod: Ovaj rad bavi se algoritmima za određivanje minimalnog pokrivajućeg stabla (MPS) povezanih, neusmerenih, težinskih grafova. Definisani su osnovni pojmovi iz teorije grafova i predstavljen je problem pronalaženja MPS. Opisan je jedan opšti postupak za određivanje MPS i uslovi za postojanje i jedinstvenost. U drugom poglavlju uveden je pojam i klase vremenske kompleksnosti algoritama na grafovima. U trećem poglavlju predstavljena su tri klasična algoritma za određivanje MPS: Borůvkin, Primov i Kruskalov. Za svaki od algoritama dat je opis algoritma, dokazana je ispravnost i kompleksnost. Ova tri algoritma određuju MPS u polinomnom vremenu. Četvrto poglavlje predstavlja empirijsku analizu algoritama predstavljenih u trećoj glavi. U petom poglavlju predstavljen je nasumični algoritam koji određuje MPS grafa u očekivanom linearnom vremenu. Za ovaj algoritam je takođe dokazana ispravnost i kompleksnost. Šesto poglavlje predstavlja primenu MPS na tržištu stranih valuta.
IZ
Datum prihvatanja teme od strane NN veća: 23.10.2014.
DP
Datum odbrane: Mart 2016.
DO
Članovi komisije:
KO
Predsednik: dr Dragan Mašulović, redovni profesor, Prirodno-matematički fakultet, Univerzitet u Novom Sadu
Mentor: dr Maja Pech, vanredni profesor, Prirodno-matematički fakultet, Univerzitet u Novom Sadu
Član: dr Mirjana Mikalački, docent, Prirodno-matematički fakultet, Univerzitet u Novom Sadu

UNIVERSITY OF NOVI SAD
FACULTY OF SCIENCE
KEY WORDS DOCUMENTATION

Accession number:
ANO
Identification umber:
INO
Document type: Monograph type
DT
Type of record: Textual printed material
TR
Contents Code: Master thesis
CC
Author: Evelin Kurta
AU
Mentor: Maja Pech, Ph.D.
MN
Title: Algorithms for finding a minimum spanning tree
TI
Language of text: Serbian (Latin)
LT
Language of abstract: en / s
LA
Country of publication: Republic of Serbia
CP
Locality of publication: Vojvodina
LP
Publication year: 2016.
PY
Publisher: Author's reprint
PU
Publ. place: Department of Mathematics and Informatics, Faculty of Sciences, University of Novi Sad, Trg Dositeja Obradovića 4, Novi Sad
PP
Physical description: (6 chapters, 63 pages, 7 tables, 22 pictures, 12 references, 1 appendix)
PD

Scientific field: Mathematics
SF
Scientific discipline: Graph theory
SD
Subject / Key words: graphs, minimum spanning tree, algorithms, polynomial time, linear time
SKW
UC
Holding data: Library of the Department of Mathematics and Informatics, Faculty of Sciences, University of Novi Sad
HD
Note:
N
Abstract: This thesis deals with algorithms for finding a minimum spanning tree (MST) of connected, undirected, weighted graphs. The first chapter provides basic concepts and terms in graph theory and defines the MST problem. A generic algorithm for growing a MST is described and we show the conditions for existence and uniqueness of a MST. In the second chapter time complexity of algorithms and complexity classes are explained. Third chapter deals with three classical algorithms for finding a MST which run in polynomial time. These are Borůvka's, Prim's and Kruskal's algorithm. For each of them we describe the algorithm, prove its correctness and complexity. In the fourth chapter we compared the classical algorithms through results of an empirical analysis. Fifth chapter deals with a randomized algorithm which has linear expected running time. In the last chapter we describe an implementation of MST in the FX market.
AB
Accepted by the Scientific Board on: 23.10.2014.
ASB
Defended: March 2016.
DE
Thesis defend board:
President: Dragan Mašulović, Ph.D., full professor, Faculty of Sciences, University of Novi Sad
Mentor: Maja Pech, Ph.D., associate professor, Faculty of Sciences, University of Novi Sad
Member: Mirjana Mikalački, Ph.D., assistant professor, Faculty of Sciences, University of Novi Sad