Algoritmi pretrage u stablu

Pintur, Antonela

PRIRODOSLOVNO–MATEMATIČKI FAKULTET
SVEUČILIŠTA U SPLITU

ANTONELA PINTUR

ALGORITMI PRETRAGE U STABLU

DIPLOMSKI RAD

Split, svibanj 2023.

PRIRODOSLOVNO–MATEMATIČKI FAKULTET
SVEUČILIŠTA U SPLITU

ODJEL ZA MATEMATIKU

ALGORITMI PRETRAGE U STABLU

DIPLOMSKI RAD

Studentica:
Antonela Pintur

Mentorica:
doc.dr.sc. Tanja Vojković

Split, svibanj 2023.

Uvod

Teorija grafova je danas iznimno popularna grana moderne matematike koja ima široke primjene. Grafove koristimo za modeliranje raznih problema kao što su na primjer mreže interneta, vodovoda, struje, prometa i slično. Vrhovi u grafovima predstavljaju čvorove tih mreža, a bridovi kabele, cijevi, ceste i slično, ovisno o problemu koji želimo riješiti. Prvi spomen teorije grafova bio je 1736. godine u Eulerovom rješenju problema šetnje koenigsberškim mostovima. Naime, ideja tog problema je bila vidjeti može li se i kako prošetati gradom Koenigsbergom tako da se svaki od 7 mostova pređe točno jednom i da se šetnja završi u polaznoj točki. I danas se grafovima pokušavaju riješiti slični problemi. Pronalaskom minimalnog razapinjućeg stabla pokušava se, na primjer, minimizirati trošak prijevoza u transportu ili se u prometu želi minimizirati vrijeme dolaska od točke A do točke B. Navedene probleme nazivamo optimizacijskim problemima i njih je uglavnom lako formulirati, ali ih nije baš jednostavno riješiti. Potrebno je naći primjerene matematičke modele za traženje njihovih rješenja. U ovom radu upoznat ćemo se s nekim osnovnim pojmovima teorije grafova, s pojmom stabla, a zatim ćemo upoznati neke algoritme pretrage koji su jako primjenjivi u stvarnom životu.

Sadržaj

Poglavlje 1

Uvod u teoriju grafova

1.1 Definicije osnovnih pojmova

Za početak, definiraju se neki osnovni pojmovi iz područja teorije grafova.

Definicija 1.1 ***Graf*** *G je uređena trojka $G = (V, E, \rho)$, gdje je V neprazan skup čije elemente zovemo* ***vrhovima*** *(engl. vertex), E je skup disjunktan s V čije elemente zovemo* ***bridovima*** *(engl. edge) i ρ preslikavanje koje svakom bridu pridružuje neuređeni par (ne nužno različitih) vrhova. Preslikavanje ρ naziva se* ***incidencijska funkcija*** *grafa G.*

Ako je $\rho(e) = \{u, v\}$, za brid se koristi oznaka $e = \{u, v\}$ ili $e = uv$. Sada se graf može promatrati kao uređeni par $G = (V, E)$. Skup vrhova i skup bridova se označava kao $V(G)$ i $E(G)$ respektivno (ili V_G i E_G).
Graf je **konačan (beskonačan)** ako su skupovi V i E konačni (bar jedan beskonačan). Nadalje ćemo razmatrati samo *konačne grafove.*
Broj vrhova $|V_G|$ se zove **red grafa** i koristi se oznaka $|V_G| = n$, a broj bridova $|E_G|$ **veličina grafa** i označava kao $|E_G| = m$.

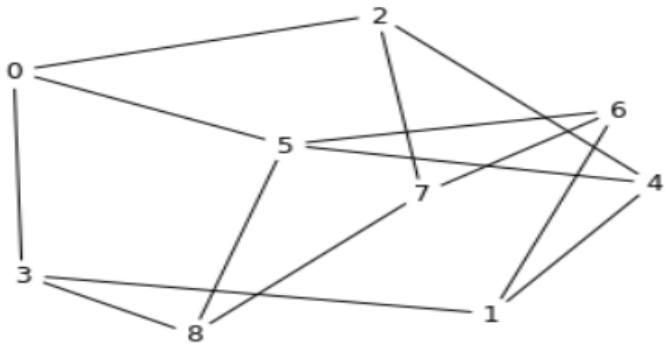
Definicija 1.2 *Kada je $\rho(e) = \{u, v\}$ kažemo da su vrhovi u i v* ***krajevi*** *brida e i međusobno* ***susjedni****. Kažemo također da su vrhovi u i v* ***incidentni*** *s bridom e i obrnuto, te, da su vrhovi u i v spojeni bridom e. Bridovi su susjedni ako su incidentni istom vrhu.* ***Petlja*** *je brid kojemu su krajevi isti vrh. Ako e nije petlja, kažemo da je* ***pravi brid****. Dva ili više različitih bridova međusobno su* ***paralelni*** *ako imaju iste krajeve.* ***Višestruki*** *brid je skup međusobno paralelnih bridova.* ***Jednostavni graf*** *je graf bez višestrukih bridova i petlji. Graf koji ima samo jedan vrh i nema niti jedan brid zovemo* ***trivijalan****.*

Skup svih vrhova grafa G susjednih vrhu v se označava s $N_G(v)$.

Definicija 1.3 *Označimo jedan graf s G, a drugi s H. Ako je $V_H \subseteq V_G$, $E_H \subseteq E_G$ i svaki brid grafa H ima iste krajeve u H kao što ih ima i u G, onda kažemo da je H* ***podgraf*** *(engl. subgraph) grafa G i pišemo $H \subseteq G$. Graf G u tom slučaju zovemo* ***nadgraf*** *grafa H. Ako je $H \subseteq G$ i $V_G = V_H$, kažemo da je H* ***razapinjući*** *(engl. spanning) podgraf grafa G.*

Podgrafovi se mogu dobiti uklanjanjem pojedinih bridova danog grafa. Ako je e brid grafa G, $G - e$ označava graf dobiven uklanjanjem brida e iz grafa G. Općenito, ako je F neki neprazan skup bridova grafa G, graf koji se dobije kad se uklone svi bridovi skupa F iz G označava se s $G - F$.
Brid $e \in E_G$ je **kontrahiran** ako je on uklonjen, a njegovi krajevi u i v identificirani s novim vrhom tako da je rezultirajući vrh incidentan s bridovima (osim e) s kojima su izvorno bili incidentini u ili v. Graf koji se dobije kontrakcijom brida e se označava kao G/e.
Susjednost vrhova i incidencija vrhova i bridova u grafu se mogu zapisati matrično. Takav prikaz je prikladan za pohranu grafa u računalo. Zbog toga se definira matrica susjedstva te incidencije.

Definicija 1.4 *Neka je G graf s vrhovima $v_1, \ldots, v_n$ i bridovima $e_1, \ldots, e_m$ u zadanom poretku.* ***Matrica susjedstva*** *(engl. adjacency matrix) grafa G je $n \times n$ matrica $A = A(G) = [a_{ij}]$, gdje je a_{ij} broj bridova koji spajaju vrhove v_i i v_j.* ***Matrica incidencije*** *(engl. incidence matrix) grafa G je $n \times m$ matrica $B = B(G) = [b_{ij}]$, gdje je b_{ij} broj incidencija vrha v_i s bridom e_j; $b_{ij} \in \{0, 1, 2\}$.*



Slika 1.1: Graf G za matricu susjedstva i incidencije

Primjer 1.5 (MATRICA SUSJEDSTVA)

$$A = \begin{bmatrix} 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \end{bmatrix}$$

Primjer 1.6 (MATRICA INCIDENCIJE)

$$B = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \end{bmatrix}$$

Osim matrice susjedstva, susjednost vrhova u grafu G možemo još promatrati preko *liste susjedstva* (engl. *adjacency list*).

Primjer 1.7 (LISTA SUSJEDSTVA) *Svaki redak u matrici susjedstva iz primjera 1.5 predstavlja listu susjedstva vrhova* $0, 1, \ldots, 8$ *redom, odnosno, za svaki vrh grafa* G *imamo sljedeće liste susjeda:*

$$L(0) = \{2, 3, 5\}$$

$$L(1) = \{3, 4, 6\}$$

$$L(2) = \{0, 4, 7\}$$

$$L(3) = \{0, 1, 8\}$$

$$L(4) = \{1, 2, 5\}$$

$$L(5) = \{0, 4, 6, 8\}$$

$$L(6) = \{1, 5, 7\}$$

$$L(7) = \{2, 6, 8\}$$

$$L(8) = \{3, 5, 7\}$$

Kako je rečeno u uvodu, teorija grafova je jedna od primjenjivijih grana matematike. Često se koristi u situacijama u kojima je bridu potrebno pridružiti neku vrijednost. Iz tog razloga uvodi se pojam *težinskog grafa*.

Definicija 1.8 ***Težinski graf*** *(engl. weighted graph) je uređeni par* (G, w) *gdje je* G *graf i funkcija* $w\colon E_G \to \mathbb{R}^+$ *svakom bridu grafa* G *pridružuje nenegativan broj* $w(e)$*. Vrijednost* $w(e)$ *naziva se* ***težina brida*** e*. Težina podgrafa težinskog grafa je ukupna težina njegovih bridova.*

Da bi se mogao uvesti pojam povezanih grafova, najprije ćemo vidjeti kako se možemo "kretati" po grafu, odnosno, definirat ćemo pojmove kao što su šetnja, staza, ciklus i put.

Definicija 1.9 ***Šetnja*** *(engl. walk)* W *u grafu* G *je konačan niz vrhova* v_i *i bridova* e_i *oblika* $v_0, e_1, v_1, e_2, \ldots, e_l, v_l$ *pri čemu su krajevi brida* e_i *vrhovi* v_{i-1} *i* v_i*. Vrh* v_0 *je početni vrh,* v_l *je završetak, a* $v_1, v_2, \ldots, v_{l-1}$ *unutarnji vrhovi. Broj bridova* l *zove se* ***duljina šetnje****.* ***Staza*** *(engl. trail) u grafu je šetnja čiji su svi bridovi međusobno različiti. Staza s međusobno različitim vrhovima naziva se* ***put*** *(engl. path).* ***Ciklus*** *u grafu je zatvorena staza koja sadrži barem jedan brid i ima međusobno različite unutarnje vrhove.*

Definicija 1.10 *Graf* G *je* ***povezan*** *ako su svaka dva njegova vrha povezana putem, inače je* ***nepovezan****. Povezani podgraf koji nije pravi podgraf niti jednog drugog povezanog podgrafa grafa* G *zove se* ***komponenta povezanosti*** *grafa* G*. Broj komponenti grafa* G *označavamo s* $c(G)$*.*

Povezani graf smo definirali kao onaj graf u kojem su svaka dva njegova vrha povezana nekim putem, pa je intuitivno jasno da se uklanjanjem nekih njegovih bridova mogu dobiti barem dva vrha tog grafa između kojih više ne postoji put, odnosno, na taj način se dobije nepovezani graf. Postavlja se

pitanje koliko je najmanje bridova potrebno ukloniti da bi povezani graf postao nepovezan ili koliko najmanje bridova treba ukloniti da se nepovezanom grafu poveća broj komponenti povezanosti. Iz tog razloga se definira bridni rez.

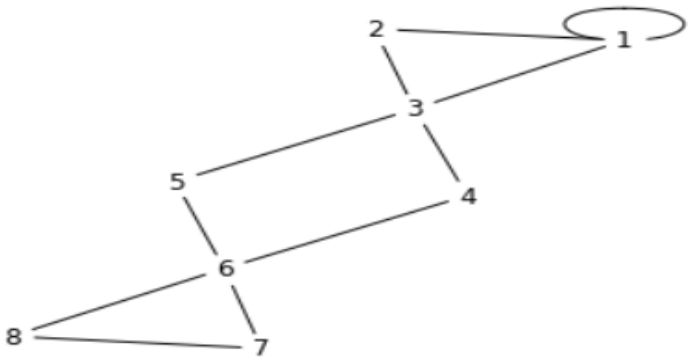
Definicija 1.11 *Neka je dan graf $G = (V, E)$ i neka su $X, Y \subseteq V$ skupovi vrhova grafa G. Sa $E[X, Y]$ označimo skup bridova kojima je jedan kraj u skupu X, a drugi u Y. Ako je $X = Y$ pišemo $E(X)$ za $E[X, X]$. Kad je $Y = V \backslash X$, skup $E[X, Y]$ nazivamo **bridni rez** grafa G određen skupom X i označavamo ga s $\partial(X)$.*

Kako znamo da uklanjanje vrhova u grafu rezultira uklanjanjem bridova koji su incidentni s tim vrhom, iz istog razloga kao prije definira se i vršni rez.

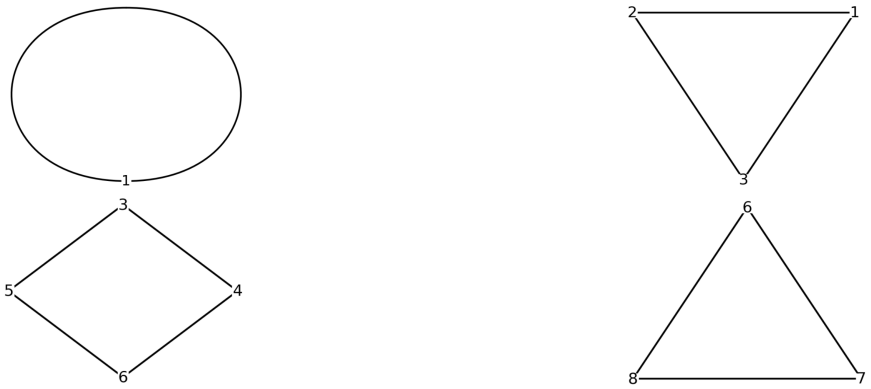
Definicija 1.12 ***Vršni rez*** *grafa G je skup S vrhova grafa čijim uklanjanjem se dobije graf s povećanim brojem komponenti povezanosti, $c(G - S) > c(G)$. Za vršni rez kažemo da je **minimalan** ako nijedan njegov pravi podskup nije vršni rez.*

Definicija 1.13 ***Blok grafa*** *bez petlji je maksimalni povezani podgraf koji nema reznih vrhova. Ako graf G ima petlji, uzima se da je svaka pojedina petlja s incidentnim vrhom jedan blok te da su ostali blokovi kao u grafu dobivenom iz G uklanjanjem petlji.*

Blok grafa po definiciji nema vlastiti rezni vrh, ali blok može sadržavati rezni vrh cijelog grafa.
Nadalje, kako bi se mogli matematički modelirati različiti praktični problemi, uvode se *usmjereni grafovi*, odnosno, grafovi u kojima je važan poredak vrhova koji su povezani bridom. Usmjerene grafove ćemo u nastavku zvati *digrafovi*, a neusmjerene grafove samo *grafovi*.



Slika 1.2: Graf G



Slika 1.3: Blokovi grafa G

Definicija 1.14 ***Usmjereni graf*** *ili* ***digraf*** *(engl. directed graph, digraph) D je uređena trojka (V, A, ψ), $D = (V, A, \psi)$, gdje je V neprazni skup čije elemente nazivamo* ***vrhovima****, A je skup disjunktan s V čije elemente nazivamo* ***lukovima*** *(engl. arc) ili* ***usmjerenim bridovima*** *i ψ funkcija koja svakom luku $a \in A$ pridružuje uređeni par $(u, v) \in V \times V$, pri čemu $u, v \in V$ nisu nužno različiti. Kažemo da je vrh u* ***početak*** *(engl. tail), a v* ***kraj*** *(engl. head) luka a te da je* ***orijentacija*** *luka a od u prema v.*

Kao i kod grafova, i ovdje ćemo promatrati samo slučajeve kad su skupovi V i A konačni, odnosno, promatramo samo konačne digrafove.
Lukove u digrafu promatramo kao bridove u grafu pa se svojstva bridova pre-

nose na svojstva lukova. Posebno, operacije nad bridovima u grafu možemo prenijeti na operacije nad lukovima u digrafu. Dakle, operacije uklanjanja vrhova i lukova definiramo na isti način kao prije. Uklanjanjem vrhova i lukova digrafa D možemo povećati broj komponenti povezanosti od D pa se uvodi pojam ***izlaznog i ulaznog lučnog reza*** (engl. *outcut, incut*).

Definicija 1.15 *Neka je dan digraf $D = (V, A)$ i neka su $X, Y \subseteq V$ skupovi vrhova digrafa D, ne nužno disjunktni. Skup lukova kojima je početak u skupu X, a kraj u skupu Y označavamo s $A[X, Y]$. Ako je $X = Y$, umjesto $A[X, X]$ pišemo $A[X]$. Kad je $Y = V \backslash X$, skup $A[X, Y]$ se zove **izlazni lučni rez** (engl. outcut) digrafa D određen skupom X i označavamo ga kao $\partial^+(X)$. Analogno, skup $A[Y, X]$ zovemo **ulazni lučni rez** (engl. incut) digrafa D određen skupom X i označavamo ga kao $\partial^-(X)$.*

Analogno kao i za grafove, zanima nas "kretanje" i po digrafu pa imamo sljedeću definiciju.

Definicija 1.16 ***Usmjerena šetnja*** *u digrafu D je niz $W = v_0, a_1, v_1, \ldots, a_k, v_k$ čiji su članovi naizmjenično vrhovi i lukovi tako da je vrh v_{i-1} početak, a vrh v_i kraj luka a_i, $i \in \{1, \ldots, k\}$. Usmjerena šetnja je zatvorena ako je $v_0 = v_k$. Usmjerenu šetnju u kojoj su svi lukovi međusobno različiti zovemo **usmjerena staza**. Usmjerenu stazu kojoj su svi vrhovi međusobno različiti zovemo **usmjereni put**.*

Treba napomenuti da postojanje usmjerene šetnje (v_k, v_l) ne znači nužno i postojanje usmjerene šetnje (v_l, v_k), ali znači postojanje usmjerenog puta (v_k, v_l).

1.2 Uvod u stabla

Stabla su posebno važni grafovi jer imaju najveću primjenu i, osim što imaju svojstva koja imaju svi grafovi, imaju i neka dodatna koja ih karakteriziraju. Za početak ćemo definirati što su to stabla.

Definicija 1.17 ***Stablo*** *(engl. tree) je povezani graf bez ciklusa. Graf bez ciklusa je* ***šuma*** *(engl. forest).*

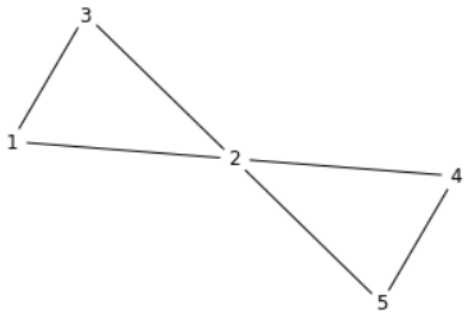
Definicija 1.18 *Označimo s r izabrani vrh stabla T koji ćemo nazvati* ***korijen*** *(engl. root) stabla. Tada se T naziva* ***stablo s korijenom r*** *ili* ***korijensko stablo*** *(engl. rooted tree).*

Stablo s korijenom r još zovemo i *r-stablo*.

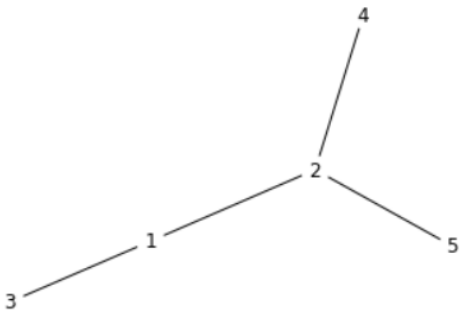
Nazivi nekih pojmova u teoriji grafova su posuđeni iz genetike. Neka je r korijen stabla. **Predak** vrha v je svaki vrh na putu rTv, uključujući i vrh v, a svaki vrh kojemu je v predak, zove se **potomak** od v. Ako je taj vrh različit od v, onda je to **pravi** predak (potomak). Reći ćemo da su dva vrha **povezana** u stablu ako je jedan predak od drugog. Neposredni predak vrha v se zove **roditelj** od v i označavamo ga s $p(v)$, a neposredni potomak je **dijete** od v.

Uklanjanjem po jednog brida iz svakog ciklusa povezanog grafa G, dobije se graf koji nema ciklusa, ali je i dalje povezan i sadrži sve vrhove od G. Na taj način dobije se stablo koje je razapinjući podgraf grafa G. Analogno možemo u nepovezanom grafu postupiti sa svakom komponentom povezanosti tog grafa i rezultat će biti šuma koja sadrži sve njegove vrhove.

Definicija 1.19 *Razapinjući podgraf grafa G koji je stablo zove se* ***razapinjuće stablo*** *(engl. spanning tree) grafa G.*



Slika 1.4: Graf G



Slika 1.5: Jedno razapinjuće stablo grafa G

Teorem 1.20 *Graf je povezan ako i samo ako ima razapinjuće stablo.*

Postoje mnoge karakterizacije stabala od kojih se neke koriste i kao alternativne definicije. Sljedeći teoremi daju neke od njih.

Teorem 1.21 *Povezani graf je stablo ako i samo ako je svaki njegov brid rezni.*

Teorem 1.22 *Graf G s n vrhova je stablo ako i samo ako je G povezan i ima $n-1$ bridova.*

Sljedeći teorem koristit će nam kasnije u analizi algoritma pretrage po dubini.

Teorem 1.23 *Razapinjući podgraf T povezanog grafa G je razapinjuće stablo od G ako i samo ako je $T \cap B$ razapinjuće stablo od B, za svaki blok B od G.*

Poglavlje 2

Algoritmi pretrage po stablima

2.1 Pretraga po stablima

Povezanost je osnovno svojstvo grafova, pa nas zanima kako odrediti je li graf povezan ili nije. U slučaju manjih grafova to se može provjeriti na način da provjeravamo vrh po vrh i tražimo puteve između njih, ali za velike grafove bi nam u tom slučaju bilo potrebno puno vremena jer možemo imati jako veliki broj bridova. Zbog toga je poželjno pronaći brži način na koji ćemo to napraviti, odnosno, pronaći *algoritam* (engl. *algorithm*) koji je efikasan i primjenjiv na grafovima.

Neka je H podgraf grafa G. Prisjetimo se da je $\partial(V(H))$ bridni rez određen podgrafom H koji označavamo s $\partial(H)$.

Neka je T podgraf grafa G takav da je T stablo. Ako je $V(T) = V(G)$, onda je T razapinjuće stablo grafa G. Iz teorema 1.20 možemo zaključiti da je G povezan graf. Ako je $V(T) \subset V(G)$ razlikujemo dva slučaja. Prvi slučaj je da je $\partial(T) = \emptyset$ i u tom slučaju je G nepovezan. Drugi slučaj je da je $\partial(T) \neq \emptyset$ i tada za svaki brid $xy \in \partial(T)$, gdje je $x \in V(T)$, $y \in V(G) \backslash V(T)$, podgraf grafa G dobiven dodavanjem vrha y i brida xy stablu T je ponovno

stablo.

Koristeći gornju ideju, može se generirati niz korijenskih stabala u G, počevši s trivijalnim stablom koje se sastoji samo od korijena r i završavajući ili s razapinjućim stablom grafa ili s nerazapinjućim čiji je pridruženi bridni rez prazan skup.

U praksi to uključuje provjeru lista susjedstva vrhova koji su već u stablu, jedan po jedan, da bi se vidjelo koji vrh i brid se još mogu dodati u stablo. Ovakav postupak se zove **pretraga po stablima** (engl. *tree-search*), a rezultat je **stablo pretrage** (engl. *search tree*). Svako pretraživanje po stablima može odrediti je li stablo povezano ili nije i u tom slučaju nas ne zanima redoslijed pretrage vrhova.

Koristeći različita pretraživanja po stablima možemo doznati različite informacije o strukturi grafa. Na primjer, pretraga po širini se može koristiti za pronalaženje udaljenosti u grafu, a pretraga po dubini za pronalazak vršnog reza.

Neka je T r-stablo. **Dubina** vrha v u stablu T je duljina puta rTv. Svaki brid od T pridružen je vrhovima na uzastopnim dubinama i uobičajeno je smatrati da su bridovi orijentirani od manje dubine prema većoj. Primjetimo da se skup bridova korijenskog stabla T kojemu je p *roditeljska funkcija*, može označavati i kao

$$E(T) = \{(p(v), v) : v \in V(T)\backslash\{r\}\}$$

gdje je r korijen stabla T.

Često korijensko stablo opisujemo s njegovim skupom vrhova i roditeljskom funkcijom. Nadalje ćemo, bez smanjenja općenitosti, smatrati da su grafovi povezani.

2.2 Pretraga po širini (BFS)

U većini pretraživanja po stablu, vrhovi se uglavnom pretražuju redoslijedom kojim su i dodani u to stablo. Pretraživanje u kojem se lista susjedstva (engl. *adjacency list*) vrhova od T gleda kao *'first in, first out'*, odnosno, pretraživanje po rastućem redoslijedu dodavanja u T, zove se **pretraga po širini** (engl. *breadth-first search, BFS*).

Da bi mogli učinkovito implementirati ovaj algoritam, vrhove stabla spremamo u red (lista Q koju proširujemo ili na način da dodajemo novi element na kraj liste ili da maknemo element s početka liste). U svakom koraku red Q sadrži sve vrhove od kojih možemo izgraditi potencijalno stablo. Na početku (u vremenu $t = 0$) red Q je prazan. Kad god stablu dodamo novi vrh, on se doda i u Q.

U svakom koraku provjerava se lista susjedstva vrha s početka reda Q kako bi se pronašli njegovi mogući susjedi i dodali u stablo.

Ako su mu svi susjedi već u stablu, uklanjamo taj vrh iz Q. Algoritam staje kad se red Q isprazni.

Osim što vraća stablo (dano preko roditeljske funkcije p), algoritam vraća i funkciju $l\colon V \to \mathbb{N}$, koja označava dubinu svakog vrha u stablu i udaljenost od r u G. Također, vraća i funkciju $t\colon V \to \mathbb{N}$ koja 'mjeri' vrijeme dodavanja svakog vrha stablu T. Svaki posjećeni vrh obojimo kako bi mogli pratiti pretragu.

Oznaka $G(X)$ označava graf G s istaknutnim vrhom (ili korijenom) x. Prisjetimo se da je *x-stablo* stablo koje ima korijen u vrhu x.

Algoritam 2.1 (PRETRAGA PO ŠIRINI (BFS))

INPUT: povezani graf $G(r)$

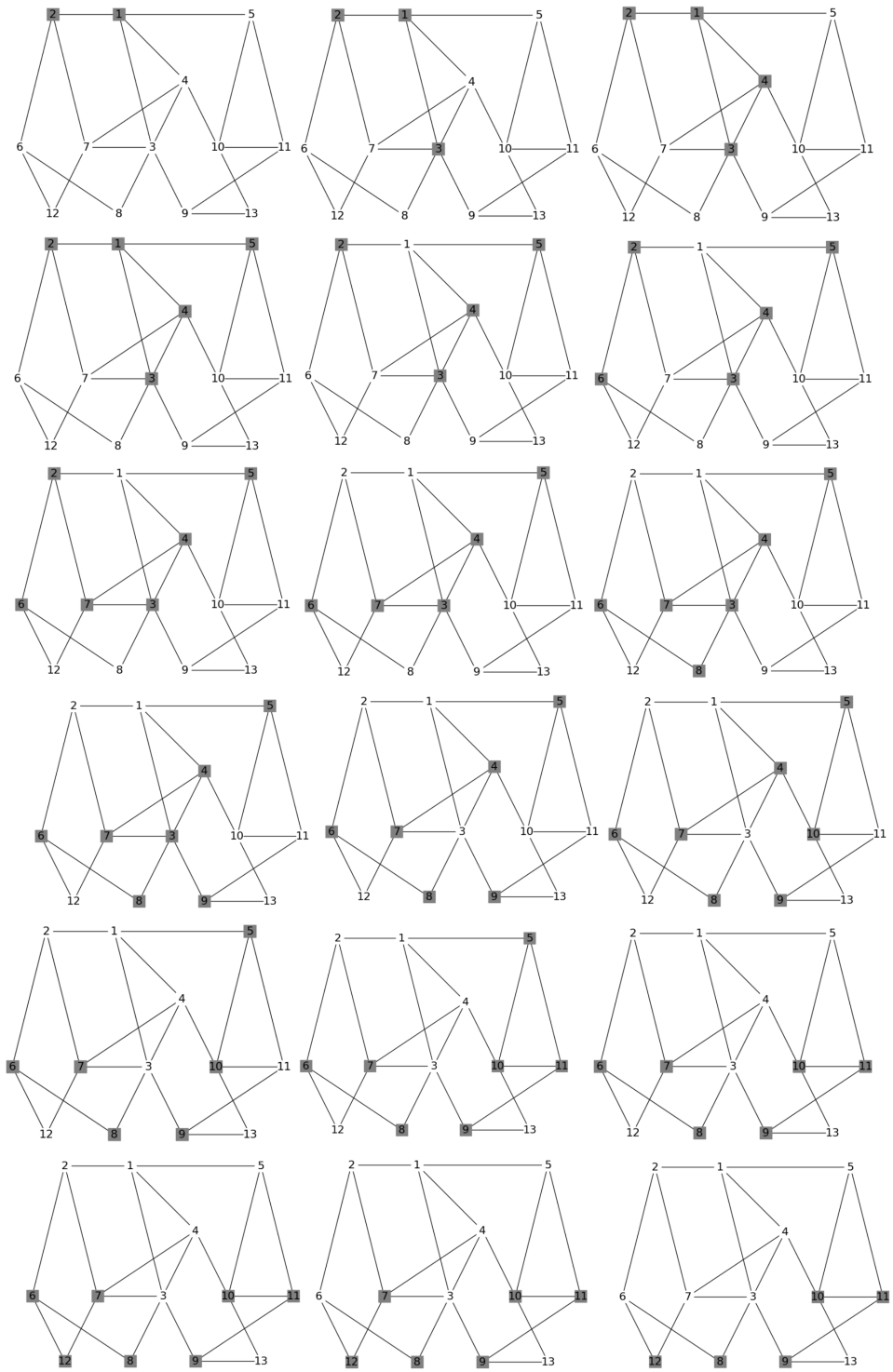
OUTPUT: r-stablo T u G s roditeljskom funkcijom p, funkcijom dubine l takvom da je $l(v) = d_G(r, v)$ za sve $v \in V$ i vremenskom funkcijom t
postavi $i := 0$, $Q := \emptyset$
povećaj i *za* 1
obojaj r *u crno*
postavi $l(r) := 0$ i $t(r) := i$
dodaj r *u* Q
while Q *nije prazan* ***do***
    uzmi vrh x *s početka* Q
    if x *ima neobojanog susjeda* y ***then***
        povećaj i *za* 1
        obojaj y *u crno*
        postavi $p(y) := x$, $l(y) := l(x) + 1$, $t(y) := i$
        dodaj y *u* Q
    else
        ukloni x *iz* Q
    end if
end while
vrati (p, l, t)

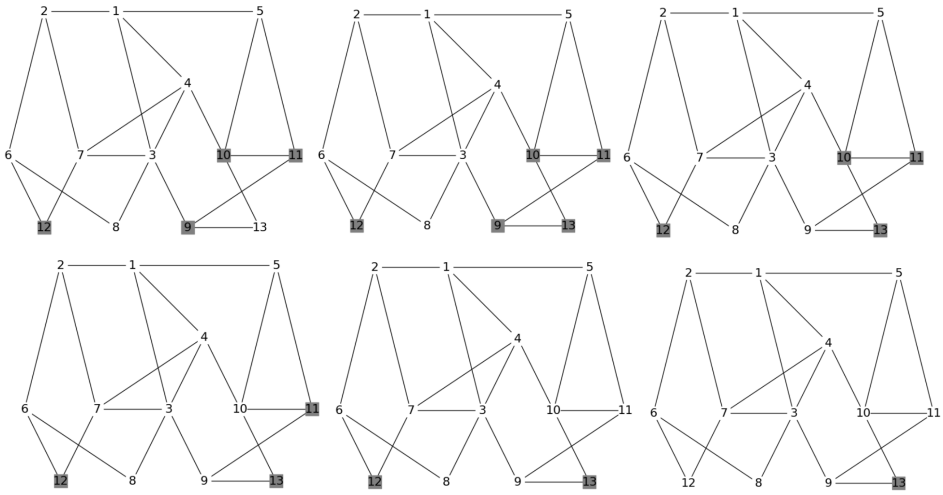
Razapinjuće stablo T, koje je rezultat *BFS*-a, zove se ***BFS-stablo*** (engl. *BFS-tree*) od G.

Primjer 2.2 (PRIMJER *BFS-STABLA*)

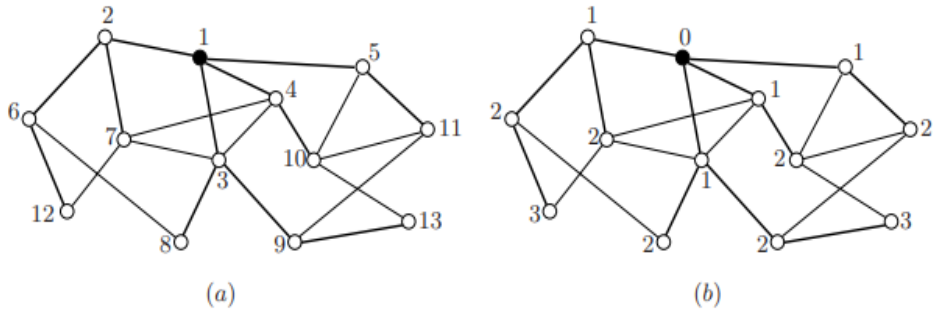
$\emptyset \to 1 \to 1\,2 \to 1\,2\,3 \to 1\,2\,3\,4 \to 1\,2\,3\,4\,5 \to 2\,3\,4\,5 \to 2\,3\,4\,5\,6 \to 2\,3\,4\,5\,6\,7 \to 3\,4\,5\,6\,7 \to 3\,4\,5\,6\,7\,8 \to 3\,4\,5\,6\,7\,8\,9 \to 4\,5\,6\,7\,8\,9 \to 4\,5\,6\,7\,8\,9\,10 \to 5\,6\,7\,8\,9\,10 \to 5\,6\,7\,8\,9\,10\,11 \to 6\,7\,8\,9\,10\,11 \to 6\,7\,8\,9\,10\,11\,12 \to 7\,8\,9\,10\,11\,12 \to 8\,9\,10\,11\,12 \to 9\,10\,11\,12 \to 9\,10\,11\,12\,13 \to 10\,11\,12\,13 \to 11\,12\,13 \to 12\,13 \to 13 \to \emptyset$

2.2. Pretraga po širini (BFS)





Slika 2.1: Primjer *BFS* povezanog grafa



Slika 2.2: a) vremenska funkcija t b) funkcija dubine l

BFS-stabla imaju dva glavna svojstva od kojih prvo opravdava naš naziv funkcije l kao funkcije dubine.

Teorem 2.3 *Neka je T BFS-stablo povezanog grafa G s korijenom r. Vrijedi:*

(i) Za svaki vrh v grafa G, $l(v) = d_T(r, v)$ je dubina od v u T;

(ii) Svaki brid grafa G je pridružen vrhovima na neposrednim dubinama od T, odnosno, za sve $uv \in E$ vrijedi $|l(u) - l(v)| \leq 1$.

Dokaz.

(i) Trivijalno slijedi iz definicije od l. Naime, l je funkcija dubine, a dubinu smo definirali kao duljinu puta rTv.

(ii) Da bi dokazali drugu tvrdnju, dovoljno je dokazati da iz $uv \in E$ i $l(u) < l(v)$ slijedi $l(u) = l(v) - 1$. Najprije ćemo indukcijom po $l(u)$ dokazati da ako su u i v proizvoljni vrhovi takvi da je $l(u) < l(v)$, onda je u dodan u Q prije vrha v.

Ako je $l(u) = 0$, u je korijen od T, pa tvrdnja očito vrijedi.

Pretpostavimo da je tvrdnja istinita za sve $l(u) < k$, gdje je $k > 0$.

Neka je $l(u) = k$. Stavimo $x := p(u)$, $y := p(v)$. Iz algoritma 2.1 (linija 11) slijedi da je $l(x) = l(u) - 1 < l(v) - 1 = l(y)$. Prema pretpostavci indukcije, x je dodan u Q prije y. Slijedi da je u, koji je susjed od x, u Q dodan prije vrha v.

Sad pretpostavimo da je $uv \in E$ i $l(u) < l(v)$. Ako je $u = p(v)$, onda je $l(u) = l(v) - 1$ (također iz algoritma 2.1, linija 11). Ako nije, stavimo $y := p(v)$. Kako je v dodan u T preko brida yv, a ne preko brida uv, vrh y je dodan u Q prije u, stoga je $l(y) \leq l(u)$ prema prethodno dokazanoj tvrdnji. Dakle, $l(v) - 1 = l(y) \leq l(u) \leq l(v) - 1$, iz čega slijedi $l(u) = l(v) - 1$.

■

Sljedeći teorem nam govori o točnosti *BFS algoritma*.

Teorem 2.4 *Neka je G povezani graf. Tada su vrijednosti funkcije l u svakom vrhu v, koji su dobiveni BFS algoritmom, jednake udaljenosti v od korijena r u grafu G.*

Dokaz. Prema tvrdnji i) u teoremu 2.3, vrijedi $l(v) = d_T(r, v)$. Štoviše, $d_T(r, v) \geq d_G(r, v)$ jer je T podgraf od G. Slijedi, $l(v) \geq d_G(r, v)$. Dokazat

ćemo suprotnu nejednakost indukcijom po duljini najkraćeg puta (r, v). Neka je P najkraći (r, v) – put u G, gdje je $v \neq r$ i neka je u roditelj od v na P. Tada je rPu najkraći (r, u) – put i $d_G(r, u) = d_G(r, v) - 1$. Prema pretpostavci indukcije, $l(u) \leq d_G(r, u)$ i prema tvrdnji ii) teorema 2.3, $l(v) - l(u) \leq 1$. Dakle,

$$l(v) \leq l(u) + 1 \leq d_G(r, u) + 1 = d_G(r, v).$$

■

2.3 Pretraga po dubini (DFS)

Pretraga po dubini (engl. *depth-first search, DFS*) je pretraga po stablu T u kojoj se tom stablu dodaje susjed vrha koji je prethodno dodan u stablo. Drugim riječima, najprije provjerimo listu susjedstva zadnjeg dodanog vrha x kako bi pronašli njegove susjede koji nisu u T. Ako takav vrh postoji, dodamo ga u T. Ako ne, vratimo se na vrh koji je dodan prije vrha x i ponovimo postupak. Postupak nastavljamo sve dok ne provjerimo sve vrhove iz liste.

Rezultirajuće stablo zovemo ***DFS-stablo*** (engl. *DFS-tree*).

Ovaj algoritam se može učinkovito implementirati ažurirajući vrhove od T čije se liste susjedstva još trebaju u potpunosti provjeriti, ovog puta koristeći stog, a ne red kao u *BFS algoritmu.*

Stog je obična lista čiji kraj zovemo *vrh stoga* (može se promijeniti ili dodavanjem novog elementa kao njegovog novog vrha ili uklanjanjem postojećeg vrha).

U pretraživanju po dubini, stog S je na početku prazan. Kad dodajemo novi vrh stablu T, dodajemo ga i u S. U svakom koraku, lista susjedstva vrha stoga se provjerava i traže se njegovi susjedi koji se potom dodaju u T. Ako

se svi njegovi susjedi već nalaze u T, taj vrh uklanjamo iz S. Algoritam staje kad se S isprazni.

Kao i u *BFS algoritmu*, vrhove koje smo pregledali obojimo crno. Za svaki vrh v iz G računaju se i dva vremena: vrijeme $f(v)$ koje označava kad je v dodan u stablo T (odnosno, dodan u stog S) i vrijeme $l(v)$ koje označava kad se vrh v uklanja iz S, odnosno, kad su pronađeni svi njegovi susjedi. Svaki put kad promijenimo stog vrijeme se poveća za 1. Točnije, $f(r) = 1$, $l(v) = f(v) + 1$ za svaki list v od T i $l(r) = 2n$.

NAPOMENA! Funkcija $l(v)$ u *DFS algoritmu* nije ista kao funkcija $l(v)$ u *BFS*).

Algoritam 2.5 (PRETRAGA PO DUBINI (DFS))

INPUT: povezani graf G

OUTPUT: razapinjuće korijensko stablo od G s roditeljskom funkcijom p i dvije vremenske funkcije f i l

postavi $i := 0$, $S := \emptyset$

odaberi bilo koji vrh x (za korijen)

povećaj i za 1

obojaj r u crno

postavi $f(r) := i$

dodaj r u S

while *S nije prazan* ***do***

    uzmi x (vrh stoga S)

    povećaj i za 1

    if *x ima neobojanog susjeda y* ***then***

        obojaj y u crno

        postavi $p(y) := x$, $f(y) := i$

dodaj y na vrh S

else

postavi $l(x) = i$

ukloni x iz S

end if

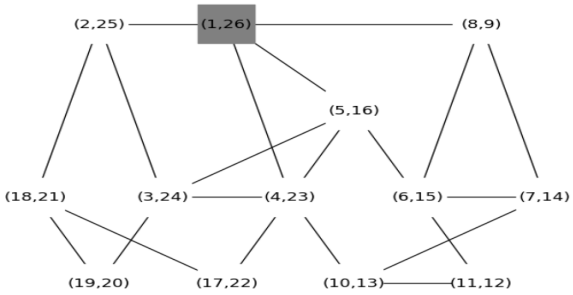
end while

vrati (p, f, l)

DFS-stablo povezanog grafa možemo vidjeti u primjeru ispod. Svaki vrh stabla je označen uređenim parovima $(f(v), l(v))$.

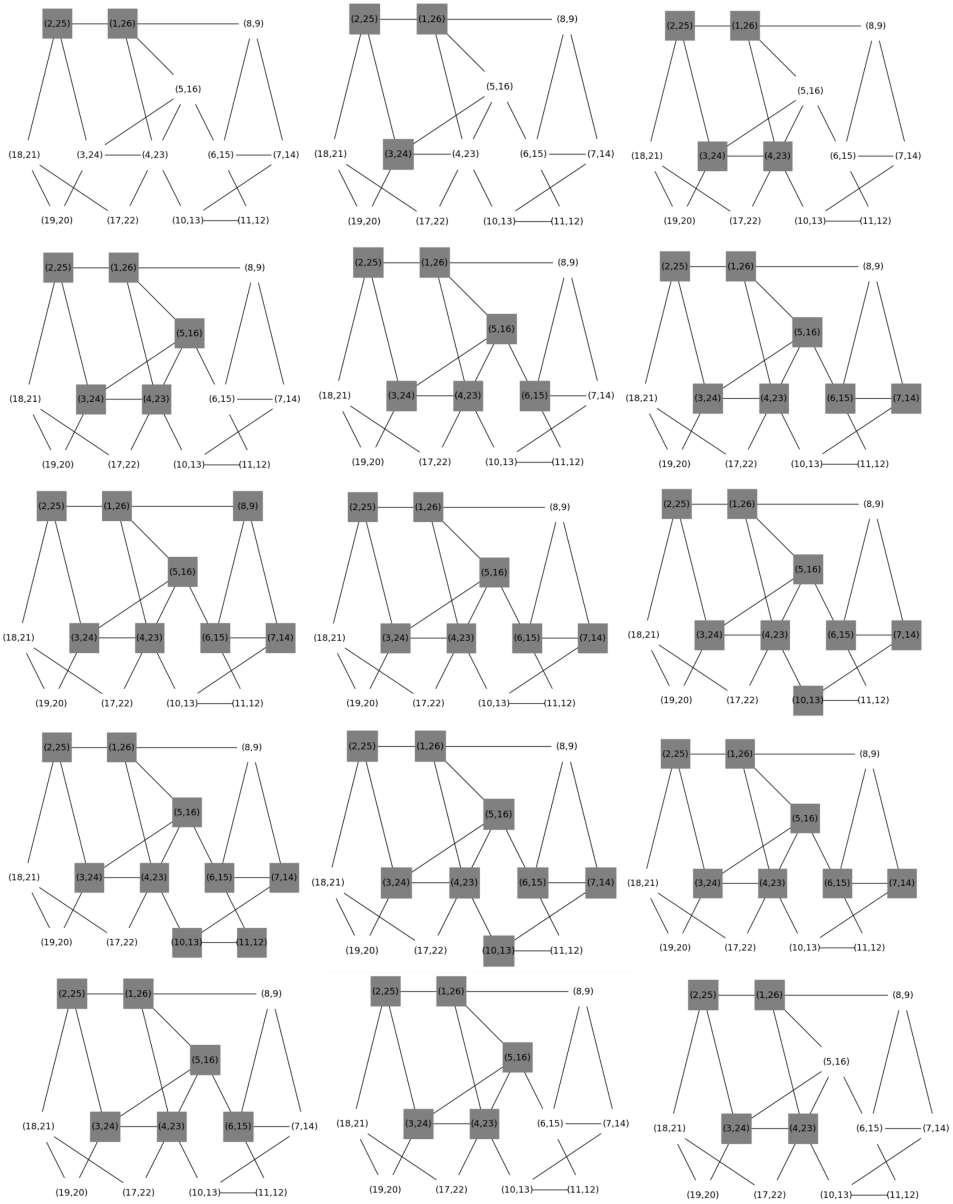
Primjer 2.6 (PRIMJER *DFS-STABLA*)

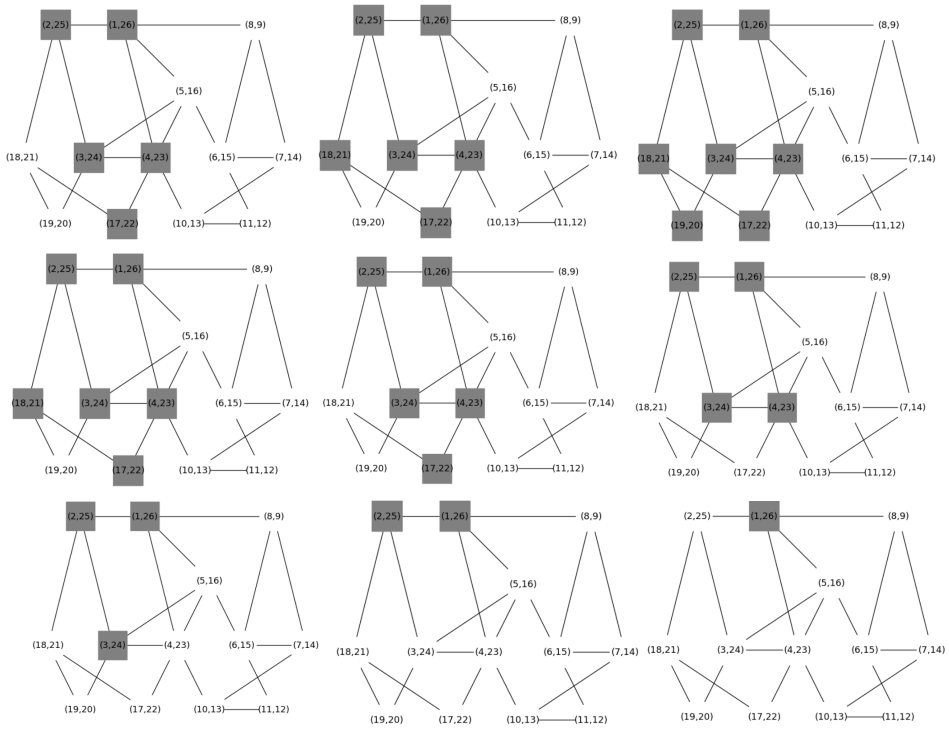
$\emptyset \to 1 \to 12 \to 123 \to 1234 \to 12345 \to 123456 \to 1234567 \to$
$12345678 \to 1234567 \to 123456710 \to 12345671011 \to 123456710 \to$
$1234567 \to 123456 \to 12345 \to 1234 \to 12341 7 \to 123417 18 \to$
$12341718 19 \to 123417 18 \to 12341 7 \to 1234 \to 123 \to 12 \to 1 \to \emptyset$



Slika 2.3: Graf G za *DFS* pretragu

2.3. Pretraga po dubini (DFS)





Slika 2.4: Primjer *DFS* povezanog grafa

Sljedeća propozicija nam daje vezu između grafa G, njemu pripadnog *DFS-stabla* i dvije vremenske funkcije f i l koje se dobiju iz algoritma 2.5.

Propozicija 2.7 *Neka su u i v dva vrha grafa G i neka je $f(u) < f(v)$. Vrijedi:*

(i) Ako su u i v susjedi u G, onda je $l(v) < l(u)$;

(ii) u je predak od v u T ako i samo ako je $l(v) < l(u)$.

Dokaz.

(i) Prema linijama $8-12$ algoritma 2.5, vrh u se uklanja iz stoga S jedino onda kad mu se sva moguća djeca (neobojani susjedi) provjere za dodavanje u S. Jedan od tih susjeda je i v jer je $f(u) < f(v)$. Stoga je

v dodan u stog S dok je u još uvijek u S i u se ne može ukloniti iz S prije nego uklonimo v. Iz čega slijedi da je $l(v) < l(u)$.

(ii) Pretpostavimo da je u predak od v u T. Prema linijama 9 i 12 u *DFS algoritmu*, kako se pomičemo po putu uTv, vrijednosti od f rastu. Ako primjenimo tvrdnju i) na svaki brid tog puta, dobijemo nejednakost $l(v) < l(u)$.

Sad pretpostavimo da u nije predak od v. Jer je $f(u) < f(v)$, niti v nije predak od u. Stoga se u ne nalazi na putu rTv i v se ne nalazi na putu rTu. Neka je s zadnji zajednički vrh puteva rTu i rTv. Kako je $f(u) < f(v)$, pravi potomci od s na putu rTv se mogu dodati u stog S samo onda kad se svi potomci od s na putu rTu uklone iz S i tako ostave s kao vrh stoga. Točnije, v možemo dodati u S samo onda kad se ukloni u, pa je $l(u) < f(v)$. Jer je $f(v) < l(v)$, možežmo zaključiti $l(u) < l(v)$.

■

Vidjeli smo ranije (u teoremu 2.3b) da *BFS-stabla* karakterizira svojstvo da je svaki brid grafa incidentan vrhovima na istim ili susjednim dubinama. Glavno svojstvo *DFS-stabala* je opisano sljedećim teoremom.

Teorem 2.8 *Neka je T DFS-stablo grafa G. Tada je svaki brid grafa G incidentan povezanim vrhovima u T.*

Dokaz. Ova tvrdnja direktno slijedi iz propozicije 2.7. Neka je uv brid u grafu G. Bez smanjenja općenitosti pretpostavimo da je $f(u) < f(v)$. Prema propoziciji 2.7a je $l(v) < l(u)$. Sada, prema istoj propoziciji 2.7b slijedi da je u predak od v, pa su u i v povezani u T. ■

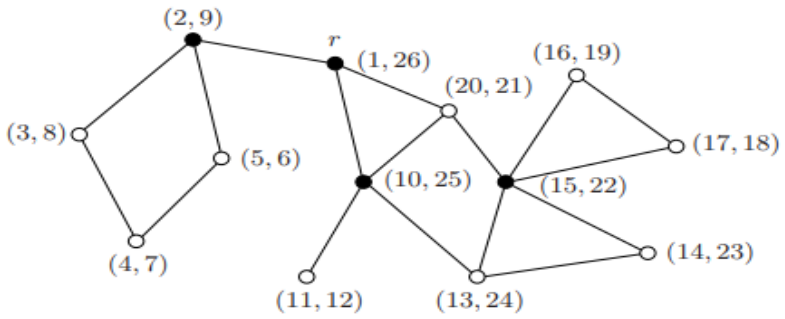
2.4 Pronalaženje vršnog reza i blokova grafa

Prilikom provođenja *DFS* u grafu G, uobičajeno je orijentirati bridove od G kao u *DFS-stablu*. Svaki brid stabla orijentiramo od roditelja prema djetetu, a svaki brid koji ne pripada stablu (čiji su krajevi povezani u T, prema teoremu 2.8) od potomka prema pretku. Ti bridovi se zovu **stražnji bridovi** (engl. *back edges*).
Sljedeći teorem je izravna posljedica teorema 2.8.

Teorem 2.9 *Neka je T DFS-stablo povezanog grafa G. Korijen r od T je rezni vrh od G ako i samo ako ima barem dva djeteta. Za svaki vrh v od T, takav da je $v \neq r$, vrijedi v je rezni vrh od G ako i samo ako v ima dijete čiji potomak nije orijentiran (po stražnjim bridovima) prema pravom pretku tog vrha.*

Pogledajmo kako možemo iskoristiti *DFS* za pronalazak reznih vrhova i blokova grafa.
Neka je T *DFS-stablo* povezanog grafa G i neka je B blok od G. Tada je $T \cap B$ stablo u G (prema teoremu 1.23). Štoviše, kako je T korijensko stablo, možemo B poistovjetiti s jedinstvenim vrhom, korijenom, stabla $T \cap B$. Taj vrh se zove *korijen od B određen s T* i njega prvog dodajemo u T.
Primjetimo da su rezni vrhovi od G zapravo korijeni blokova. Stoga, kako bi odredili vršni rez i blokove od G, dovoljno je odrediti te korijene. Pokaže se da je to moguće napraviti pomoću *DFS*.
Definiramo funkciju $f^*\colon V \to \mathbb{N}$ na sljedeći način. Ako možemo dohvatiti nekog pravog pretka vrha v preko v koristeći usmjereni put koji se sastoji od bridova stabla i jednog stražnjeg brida, $f^*(v)$ se definira kao zadnja f-vrijednost takvog pretka. Inače, stavimo $f^*(v) := f(v)$. Primjetimo da je vrh v korijen bloka ako i samo ako ima dijete w tako da vrijedi $f^*(w) \geq f(v)$.

Funkciju f^* možemo izračunati dok izvršavamo *DFS* i istovremeno se mogu provjeravati i korijeni blokova. Na taj način se korijeni blokova od G, a onda i blokovi, mogu odrediti u linearnom vremenu. Korijeni blokova grafa određenih s *DFS-stablom* se mogu vidjeti na slici ispod. Za svaki vrh v imamo par $(f(v), l(v))$.



Slika 2.5: Pronalazak vršnog reza i blokova grafa koristeći *DFS*

2.5 Minimalno razapinjuće stablo

Na početku smo definirali razapinjuća stabla (definicija 1.19). **Minimalno razapinjuće stablo** je razapinjuće stablo minimalne težine. Promotrimo sljedeći problem:

Problem 2.10 (MINIMALNO RAZAPINJUĆE STABLO)
Zadan je povezani težinski graf.
Treba naći minimalno razapinjuće stablo T od G.

Minimalno razapinjuće stablo ćemo zvati *optimalno stablo*.

2.5.1 Jarník–Prim algoritam

Zahvaljujući Jarník-u i Prim-u, problem minimalnog razapinjućeg stabla može se riješiti pomoću pretraživanja po stablu.

U algoritmu, koji zovemo ***Jarník-Prim algoritam***, za korijen od T se odabere proizvoljni vrh r. U svakom koraku brid koji se dodaje u stablo T je brid najmanje težine u bridnom rezu određenom s T. Kao i u *BFS* i *DFS*, bojaju se vrhovi od T.

Također, da bi se učinkovito implementiralo navedeno pretraživanje po stablu, svakom neobojanom vrhu v se pridruži *privremeni trošak* (engl. *provisional cost*) $c(v)$. To je najmanja težina brida koji je incidentan vrhu v i nekom obojanom vrhu u, ako takav brid postoji, i u tom slučaju vrh u se zove *privremeni roditelj* (eng. *provisional predecessor*) od v, u oznaci $p(v)$. Inicijalno, svaki vrh ima beskonačan trošak i nema roditelja. Te se dvije privremene oznake ažuriraju u svakom koraku algoritma.

Algoritam 2.11 (JARNÍK–PRIM ALGORITAM)

INPUT: težinski povezani graf (G, w)

OUTPUT: optimalno stablo T *od* G *s roditeljskom funkcijom* p *i težinom* $w(T)$

postavi $p(v) := \emptyset$, $c(v) := \infty$, $v \in V$, $w(T) := 0$

odaberi bilo koji vrh x *(za korijen)*

zamijeni $c(r)$ *s* 0

while *ima neobojanih vrhova* ***do***

  odaberi vrh u *minimalnog troška* $c(u)$

  obojaj u *u crno*

  for *svaki neobojani vrh* v *takav da je* $w(uv) < c(v)$ ***do***

    zamijeni $p(v)$ *s* u *i* $c(v)$ *s* $w(uv)$

    zamijeni $w(T)$ *s* $w(T) + c(u)$

end for

end while

vrati $(p, w(T))$

U praksi, skup neobojanih vrhova i njihov trošak se spremaju u *prioritetni red*(engl. *priority queue*). Iako to nije pravi red, vrh minimalnog troška se uvijek nalazi na početku reda i na taj način mu se može odmah pristupiti. Nadalje, red je strukturiran na način da se može brže ažurirati kad se taj vrh ukloni (oboji) ili kad se promijeni trošak.
Rezultat Jarník-Prim-ovog algoritma je korijensko razapinjuće stablo koje se zove ***Jarník-Prim-stablo***.
Sljedeći teorem nam govori o točnosti algoritma.

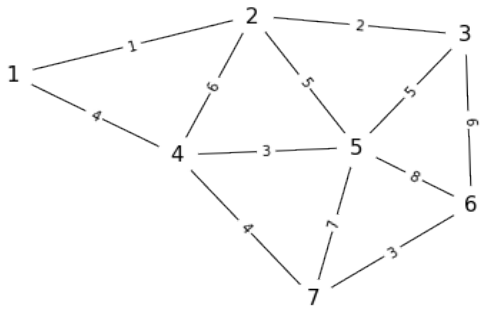
Teorem 2.12 *Svako Jarník-Prim-stablo je optimalno stablo.*

Dokaz. Neka je T Jarník-Prim-stablo s korijenom r. Indukcijom po broju vrhova od T, $v(T)$, ćemo dokazati da je T optimalno stablo.
Prvi brid koji je dodan u T je brid e najmanje težine u bridnom rezu određenom s $\{r\}$, odnosno, $w(e) \leq w(f)$ za svaki brid f incidentan s r. Za početak ćemo pokazati da neka optimalna stabla uključuju brid e.
Neka je T^* optimalno stablo. Pretpostavimo da $e \notin E(T^*)$. Tada $T^* + e$ sadrži jedinstveni ciklus C. Neka je f neki drugi brid iz C koji je incidentan s r. Tada je $T^{**} := (T^* + e)/f$ razapinjuće stablo od G. Štoviše, jer je $w(e) \leq w(f)$, $w(T^{**}) = w(T^*) + w(e) - w(f) \leq w(T^*)$.
Kako je T^* optimalno stablo, mora vrijediti jednakost, pa je T^{**} također optimalno stablo. Štoviše, T^{**} sadrži e.
Sad promotrimo graf $G' := G/e$ i označimo s r' vrh koji nastane kontrakcijom brida e. Imamo 1 na 1 korespondenciju između skupa razapinjućih stabala od G koji sadrže e i skupa svih razapinjućih stabala od G'. Stoga, da bi

pokazali da je konačno stablo T optimalno stablo od G, dovoljno je pokazati da je $T' = T/e$ optimalno stablo od G'.

Tvrdimo da je T' Jarník-Prim-stablo od G' s korijenom r'.

Promotrimo stablo T u nekom koraku Jarník-Prim algoritma. Pretpostavimo da T ne sadrži samo vrh r. Dakle onda sadrži i brid e. Neka je $T' = T/e$. Tada je $\partial(T) = \partial(T')$, pa je brid minimalne težine u $\partial(T)$ jednak onom minimalne težine u $\partial(T')$. Kako je konačno stablo T Jarník-Prim-stablo od G, zaključujemo da je T' Jarník-Prim stablo od G'. Kako G' ima manje vrhova od G, po indukciji slijedi da je T' optimalno stablo od G'. Dakle, T je optimalno stablo od G. ■



Slika 2.6: Težinski graf G

Primjer 2.13 *Koristeći Jarník-Prim algoritam, pronađite minimalno razapinjuće stablo grafa* $G = (V, E)$ *sa slike 2.6.* $V_G = \{1, 2, 3, 4, 5, 6, 7\}$*, a bridovi sa svojim težinama su sljedeći:*

- $w(\{1, 2\}) = 1$
- $w(\{2, 3\}) = 2$
- $w(\{1, 4\}) = 4$
- $w(\{2, 4\}) = 4$
- $w(\{2, 5\}) = 5$
- $w(\{3, 5\}) = 5$
- $w(\{3, 6\}) = 6$
- $w(\{4, 5\}) = 3$
- $w(\{4, 7\}) = 4$
- $w(\{5, 6\}) = 8$
- $w(\{5, 7\}) = 7$
- $w(\{6, 7\}) = 3$

IZABRANI BRID	SKUP POVEZANIH VRHOVA
{1,2}	{1,2}
{2,3}	{1,2, 3}
{1,4}	{1,2, 3, 4}
{4,5}	{1,2,3,4,5}
{4,7}	{1,2,3,4,5,7}
{6,7}	{1,2,3,4,5,6,7}

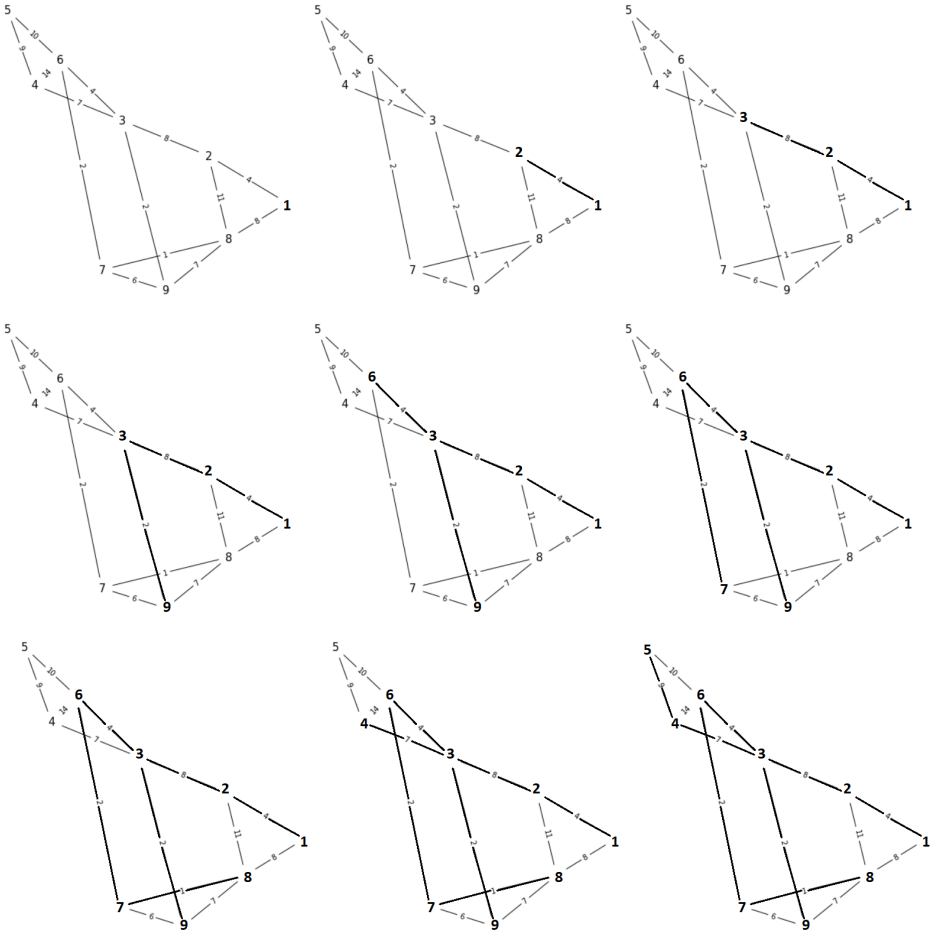
Tablica 2.1: *Jarník-Prim* algoritam

$$w = w(\{1, 2\}) + w(\{2, 3\}) + w(\{1, 4\}) + w(\{4, 5\}) + w(\{4, 7\}) + w(\{6, 7\}) =$$

$$= 1 + 2 + 4 + 3 + 4 + 3 = 17.$$

Primjer 2.14 *Koristeći Jarník-Prim algoritam, pronađite minimalno razapinjuće stablo grafa* $G = (V, E)$ *kojemu je skup vrhova* $V_G = \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$*, a bridovi sa svojim težinama su sljedeći:*

- $w(\{1,2\}) = 4$
- $w(\{1,8\}) = 8$
- $w(\{2,8\}) = 11$
- $w(\{2,3\}) = 8$
- $w(\{3,4\}) = 7$
- $w(\{3,6\}) = 4$
- $w(\{3,9\}) = 2$
- $w(\{4,5\}) = 9$
- $w(\{4,6\}) = 14$
- $w(\{5,6\}) = 10$
- $w(\{6,7\}) = 2$
- $w(\{7,8\}) = 1$
- $w(\{7,9\}) = 6$
- $w(\{8,9\}) = 7$



Slika 2.7: Minimalno razapinjuće stablo iz primjera 2.14

2.5.2 Boruvka-Kruskal algoritam

Jarník-Prim algoritam počinje s korijenom i određuje ugniježdeni niz stabala, a završava s minimalnim razapinjućim stablom.

Osim njega, za rješavanje problema minimalnog stabla, često se koristi i *Boruvka-Kruskal algoritam*. On počinje s praznim razapinjućim podgrafom i pronalazi ugniježdeni niz šuma završavajući s optimalnim stablom. Ovaj niz se konstruira dodavanjem bridova na način da u svakom koraku dodamo

brid najmanje težine, pod uvjetom da je rezultirajući podgraf i dalje šuma.

Algoritam 2.15 (BORUVKA-KRUSKAL ALGORITAM)

INPUT: težinski povezani graf (G, w)

OUTPUT: optimalno stablo $T = (V, F)$ *od* G *s težinom* $w(F)$

postavi $F := \emptyset$, $w(F) := 0$ *(*F *označava skup bridova trenutne šume)*

while *postoji brid* $e \in E \backslash F$ *takav da je* $F \cup \{e\}$ *skup bridova od šume* ***do***

odaberi brid e *minimalne težine*

zamijeni F *s* $F \cup \{e\}$ *i* $w(F)$ *s* $w(F) + w(e)$

end while

vrati $((V, F), w(F))$

Pretpostavimo da je G povezan, šuma (V, F) koju dobijemo kao rezultat algoritma 2.15 je razapinjuće stablo od G. Zovemo je *Boruvka-Kruskal-stablo*. Da bi mogli učinkovito implementirati *Boruvka-Kruskal algoritam*, trebamo provjeravati povezuje li brid, kojeg želimo dodati, vrhove iz različitih komponenti šume. To možemo napraviti bojanjem vrhova iz iste komponente istom bojom, dok one vrhove iz različitih komponenti obojamo različitim bojama. Tada je dovoljno provjeriti imaju li krajevi bridova različite boje. Kad se brid doda u šumu, svi vrhovi iz dvije spojene komponente se oboje bojom druge komponente.
Sljedeći teorem nam govori o točnosti *Boruvka-Kruskal algoritma*

Teorem 2.16 *Svako Boruvka-Kruskal-stablo je optimalno stablo.*

Primjer 2.17 *Koristeći Boruvka-Kruskal algoritam, pronađite minimalno razapinjuće stablo grafa* $G = (V, E)$ *sa slike 2.6.* $V_G = \{1, 2, 3, 4, 5, 6, 7\}$*, a bridovi sa svojim težinama su sljedeći:*

- $w(\{1,2\}) = 1$
- $w(\{2,3\}) = 2$
- $w(\{1,4\}) = 4$
- $w(\{2,4\}) = 4$
- $w(\{2,5\}) = 5$
- $w(\{3,5\}) = 5$
- $w(\{3,6\}) = 6$
- $w(\{4,5\}) = 3$
- $w(\{4,7\}) = 4$
- $w(\{5,6\}) = 8$
- $w(\{5,7\}) = 7$
- $w(\{6,7\}) = 3$

Najprije treba sortirati bridove uzlazno po težini: $\{1,2\}$, $\{2,3\}$, $\{4,5\}$, $\{6,7\}$, $\{1,4\}$, $\{2,5\}$, $\{4,7\}$, $\{3,5\}$, $\{2,4\}$, $\{3,6\}$, $\{5,7\}$, $\{5,6\}$.

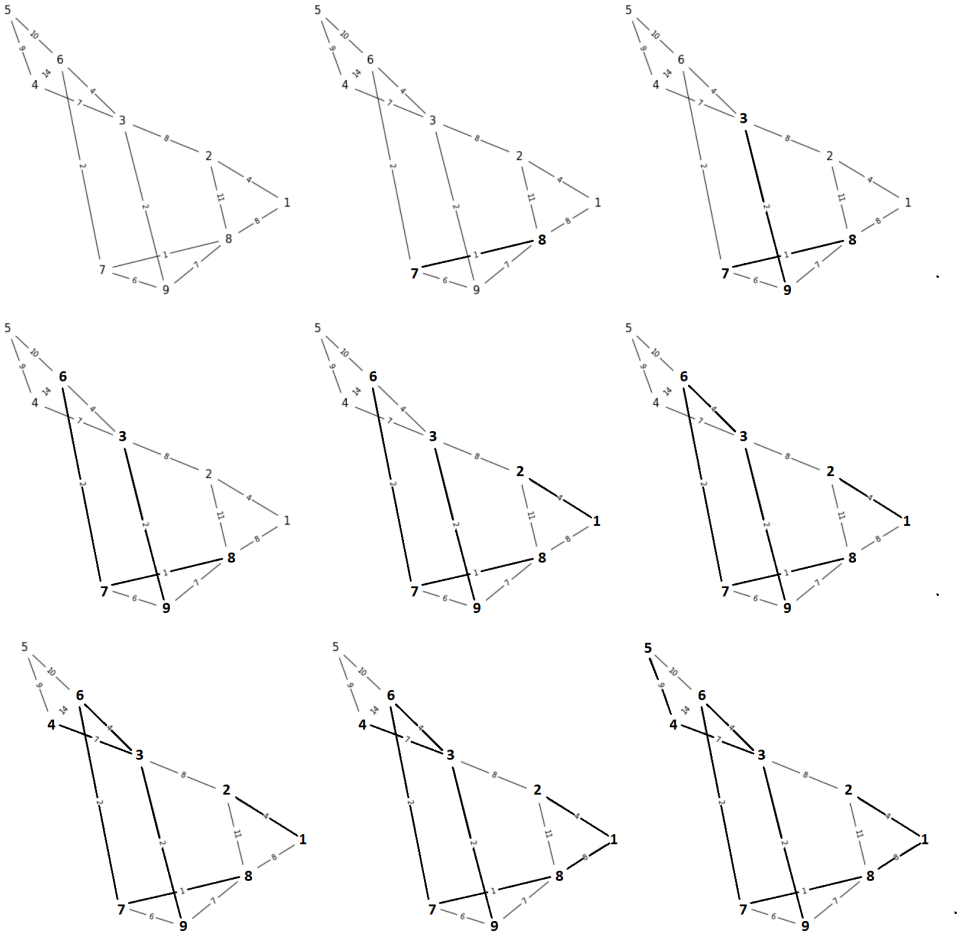
IZABRANI BRID	**SKUP POVEZANIH VRHOVA**
–	{1},{2},{3},{4},{5},{6},{7}
$\{1,2\}$	$\{1,2\}$,{3},{4},{5},{6},{7}
$\{2,3\}$	$\{1,2,3\}$,{4},{5},{6},{7}
$\{4,5\}$	$\{1,2,3\}$,$\{4,5\}$,{6},{7}
$\{6,7\}$	$\{1,2,3\}$,$\{4,5\}$,$\{6,7\}$
$\{1,4\}$	$\{1,2,3,4,5\}$,$\{6,7\}$
$\{2,5\}$	odbacuje se jer su vrhovi u istoj komponenti
$\{4,7\}$	$\{1,2,3,4,5,6,7\}$

Tablica 2.2: *Boruvka-Kruskal* algoritam

$$w = w(\{1,2\}) + w(\{2,3\}) + w(\{4,5\}) + w(\{6,7\}) + w(\{1,4\}) + w(\{4,7\}) =$$
$$= 1 + 2 + 3 + 3 + 4 + 4 = 17.$$

Primjer 2.18 *Koristeći Boruvka-Kruskal algoritam, pronađite minimalno razapinjuće stablo grafa* $G = (V, E)$ *kojemu je skup vrhova* $V_G = \{1,2,3,4,5,6,7,8,9\}$, *a bridovi sa svojim težinama su sljedeći:*

- $w(\{1,2\}) = 4$
- $w(\{1,8\}) = 8$
- $w(\{2,8\}) = 11$
- $w(\{2,3\}) = 8$
- $w(\{3,4\}) = 7$
- $w(\{3,6\}) = 4$
- $w(\{3,9\}) = 2$
- $w(\{4,5\}) = 9$
- $w(\{4,6\}) = 14$
- $w(\{5,6\}) = 10$
- $w(\{6,7\}) = 2$
- $w(\{7,8\}) = 1$
- $w(\{7,9\}) = 6$
- $w(\{8,9\}) = 7$



Slika 2.8: Minimalno razapinjuće stablo iz primjera 2.18

2.6 Primjena algoritama pretrage

POVEZIVANJE RURALNIH DIJELOVA NIGERIJE

Zbog razvoja tehnologije, projektiranje mreže za povezivanje sela je postalo ekonomski izvedivo. U projektiranju, ulaz je skup sela sa udaljenostima među njima. Trošak mreže je trošak goriva koje se koristi za putovanje po selima. Podskupove sela određujemo na način da su sva sela s postojećom mrežom povezana na isplativ način.

Pravila usluge autobusnog prijevoza su sljedeća:

1. jedno vozilo može posjetiti skup sela točno jednom prilikom putovanja
2. vozač želi krenuti iz određenog sela i završiti put na određenoj destinaciji
3. mora biti određena minimalna cijena puta

Neka je $G = (V, E)$ neusmjereni graf s vrhom i težinama kako je definirano na početku. Problem minimalnog razapinjućeg stabla se sastoji u pronalasku povezanog podgrafa $T = (V_T, E_T)$ bez ciklusa, gdje je $V_T \subseteq V$, $E_T \subseteq E$.

Prikupljena je mapa prometnica ruralnih naselja u Nigeriji i digitalizirana je u mrežni dijagram koji čini graf od n vrhova. Svaki vrh predstavlja jedno selo, a svaki brid predstavlja prometnicu koja povezuje sela. Težine su udaljenosti između sela.

Cilj ovog algoritma je dobiti informacije kako najbolje pružiti mogućnost ljudima da dođu do ruralnih dijelova zemlje, zatim dati informacije o najkraćem putu za vodovodne cijevi, struju, medicinske usluge i slično radi isplativosti te pomoći pružateljima usluga prijevoza po pitanju pristupačnosti.

Pretpostavimo da imamo cestu od mjesta A do mjesta B. Neka je m duljina najkraćeg takvog puta. Označimo tu cestu s $x_0, x_1, \ldots, x_{m-1}, x_m$ gdje

je $x_0 = A$ i $x_m = B$. Pretpostavimo da je $A = B$ i $m > n$. Kako imamo n gradova, prema Dirichletovom principu, među gradovima $x_0, x_1, \ldots, x_{m-1}$ se nalaze barem 2 ista grada.

Neka je $x_i = x_j$, za $0 \leq i < j \leq m-1$. Tada cesta sadrži ciklus od x_i do x_j, odnosno, do samog sebe. Taj ciklus možemo ukloniti i onda nam ostaje cesta $x_0, x_1, \ldots, x_i, x_{j+1}, \ldots, x_m$ od A do B koja je manje duljine, stoga najkraći put mora biti duljine najviše n.

Pretpostavimo sad da je $x_i \neq x_j$. Trebamo pronaći početak i kraj puta zajedno sa usputnim stajalištima. Na početku mapa s n sela ima $n-1$ cestu, pa je onda i stablo (prema teoremu 1.22). Neka je i broj usputnih sela, a d broj odredišta na karti. Kad znamo n, i i d, možemo odrediti parametre m i mi, gdje je m broj djece svakog od i usputnih sela, a ukupno imamo mi sela na mapi (ne računajući početno selo). Dakle, mapa ima ukupno $n = mi + 1$ selo.

Koristeći Jarník-Prim algoritam, odredit ćemo najkraći put između svaka dva sela na mapi. Kao što je i rečeno prije, algoritam gradi razapinjuće stablo na način da kreće od jednog vrha (početka puta) i dodaje jedan po jedan brid (cesta) po pravilima algoritma. Postupak se ponavlja dok ne dodamo sve vrhove (sela).

U text dokumentu su spremljene informacije o 88 gradova i udaljenostima među njima. Prilikom izvršavanja algoritma, zahtijeva se unos korijena (početak puta) i algoritam generira sve veze između polazišta i ostalih sela. Autobus koji putuje iz bilo kojeg polaznog mjesta do neke destinacije samo treba pratiti put kojeg je generirao algoritam.

Algoritam je vrlo učinkovit u pronalaženju najkraćeg puta. Reducira trošak goriva i vremena za prijevoz putnika što omogućava prihvatljive cijene prijevoza u Nigeriji. Također, ovaj algoritam u budućnosti može poslužiti za

smanjenje troškova vode, struje, telekomunikacije i slično.

Slika 2.9: Primjer mreže prometnica u Odedi (Federalna država Ogun, Nigerija)

Poglavlje 3

Pretraga grananjem

Do sada smo proučavali i pretraživali grafove, a sada ćemo vidjeti da to isto možemo raditi i s *digrafovima*.

Iz definicije digrafa znamo da svaki njegov luk pokazuje orijentaciju od jednog vrha prema drugom, stoga takvi grafovi imaju veću primjenu u modeliranju problema. Na primjer, ako promatramo gradski promet i želimo ga prikazati kao mrežu ulica koja će nam služiti kao auto karta, prikazat ćemo ga kao digraf. Naime, znamo kako ulice mogu biti jednosmjerne i dvosmjerne, odnosno, nekim ulicama se može prometovati samo u jednom smjeru, dok je u nekim to moguće u oba pa nam je ideja koristiti usmjerene bridove (lukove). Vrhovi tog digrafa će predstavljati mjesta gdje se ulice sijeku, a lukovi ulice. Pretpostavimo da u tom gradu poštar mora dostaviti poštu u što kraćem vremenu sa svojim automobilom. Kako bi skratio vrijeme i svima dostavio njihovu poštu, trebao bi svakom ulicom proći točno jednom. Njegov problem može se riješiti koristeći teoriju grafova.

U ovakvim pretraga koristimo *grananje*, odnosno, stabla kod kojih su svi usmjereni putevi oblika (r, v_k), gdje je $r \in V_T$ korijen promatranog stabla T, a $v_k \in V$ takav vrh iz V_T da je $v_k \neq r$.

Kod grananja se kreće od korijena r i dodaje se jedan po jedan luk. Lukovi se odabiru iz izlaznog lučnog reza određenog trenutnim skupom vrhova u grananju.

Postupak završava ili razapinjućim grananjem digrafa D ili nerazapinjućim grananjem čiji je pripadni izlazni lučni rez prazan skup. Konačni skup vrhova grananja sadrži vrhove $v_k \in V_D$ za koje postoji put (r, v_k) u D. Takav postupak zovemo **pretraga grananjem** (engl. *branching search*).

Kao i kod korijenskih stabala u grafovima, tako i u grananju možemo imati različite načine pretrage, ovisno o redoslijedu odabira lukova. Stoga i ovdje imamo pretrage po širini (*BFS*) i pretrage po dubini (*DFS*). Sada ćemo reći nešto više o svakoj od njih.

3.1 Usmjerena pretraga po širini (*BFS*)

Pretpostavimo da imamo problem poštara kao u uvodu ovog poglavlja. Problem se svodi na pronalazak usmjerenog puta najmanje duljine koji povezuje neka dva vrha u *težinskom digrafu*.

Problem 3.1 (PROBLEM NAJKRAĆEG PUTA)
Zadan je težinski digraf (D, w) *i njegova dva vrha* x *i* y*.*
Treba naći minimalni težinski usmjereni put (x, y) *u* D*.*

U nastavku će se umjesto pojma *težina* koristiti pojam *duljina*. Također, *minimalna težina* postaje *najkraći usmjereni put*, a *težina* od x do y postaje *udaljenost* od x do y i označava se kao $d(x, y)$. Ako digraf sadrži luk težine 0, uvijek ga možemo kontrahirati. Jedini problem mogu predstavljati lukovi negativnih težina jer, ako digraf sadrži usmjereni ciklus negativnih težina, onda može postojati $(x, y)-$ šetnja koja je kraća od bilo kojeg $(x, y)-$ puta pa nam ovi algoritmi pretrage neće koristiti.

Dakle, pretpostavimo da su sve težine pozitivne. U tom slučaju, za rješavanje problema, možemo koristiti *Dijkstrin* algoritam pretrage.

3.1.1 Dijkstrin algoritam pretrage

Iako se koristi u *BFS* za digrafove, *Dijkstrin* algoritam je jako sličan *Jarník-Prim* algoritmu zbog dodijeljivanja privremenih oznaka vrhovima. U svakom koraku, svaki vrh v trenutnog grananja B se označi pomoću njegovog roditelja, $p(v)$, u B i udaljenosti od korijena r, koristeći funkciju $l(v) := d_B(r, v)$. Svaki vrh v koji nije u grananju B, ali za koji postoji neki luk kojemu je početak u nekom vrhu u od B, a kraj u v, označava se privremenim funkcijama $p(v)$ i $l(v)$. Ovakve vrhove v ćemo zvati *vanjskim susjedima od* u. Pravilo za biranje sljedećeg vrha i luka za grananje ovisi samo o privremenim udaljenostima.

Algoritam 3.2 (DIJKSTRIN ALGORITAM)

INPUT: pozitivni težinski digraf (D, w) *s istaknutim vrhom* r

OUTPUT: r-grananje u D *s roditeljskom funkcijom* p *i funkcijom* $l\colon V \to \mathbb{R}^+$ *za koju vrijedi* $l(v) = d_D(r, v)$*, za sve* $v \in V$

postavi $p(v) := \emptyset$*,* $v \in V$*,* $l(r) := 0$*,* $l(v) := \infty$*,* $v \in V \backslash r$

while *postoji neobojani vrh* u *takav da je* $l(u) < \infty$ ***do***

odaberi vrh u *minimalne duljine* $l(u)$

obojaj u *u crno*

for *svaki neobojani vanjski susjed* v *od* u *za kojeg vrijedi* $l(v) > l(u) + w(u, v)$ ***do***

zamijeni $p(v)$ *s* u *i* $l(u)$ *s* $l(u) + w(u, v)$

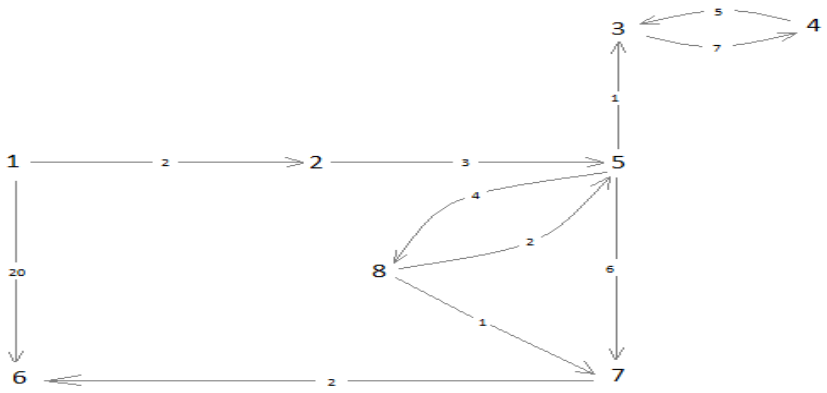
end for

end while

vrati (p, l)

3.1. Usmjerena pretraga po širini (*BFS*)

Kao i *Jarník-Prim* algoritam, i *Dijkstrin* algoritam se može implementirati na način da neobojane vrhove i njihove udaljenosti spremamo u red.



Slika 3.1: Primjer digrafa D za primjenu Dijkstrinog algoritma

Primjer 3.3 *Koristeći Dijkstrin algoritam pretrage, pronađite grananje digrafa D sa slike 3.1*

Inicijalno ćemo staviti $S := \emptyset$, gdje S sadrži posjećene vrhove digrafa D.

Sve korake algoritma možemo pratiti u tablici 3.1.

Odaberimo neki vrh kao korijen. Na primjer, možemo uzeti vrh "1". *Tablica udaljenosti (od korijena) na početku izgleda kao u koraku 1. Dakle, samo korijen* "1" *ima udaljenost* 0, *svi ostali imaju udaljenost* ∞.

Sada promotrimo udaljenosti izlaznih susjednih vrhova korijena "1", *pogledajmo jesu li one manje od vrijednosti zapisanih u tablici i, ako jesu, popunimo tablicu s tim vrijednostima.*

Dodamo vrh "1" *u S.*

Nakon toga iz tablice udaljenosti biramo vrh s najmanjom udaljenosti koji je

do sada neposjećen i promatramo njegove neposječene izlazne susjede. Vidimo da je to vrh "2" *i da je njegov jedini izlazni susjed vrh* "5"*, pa dodajemo i udaljenost od* "5" *u tablicu ako je manja od trenutno zapisane.*
Dodamo vrh "2" *u* S*.*
Ponovno tražimo neposjećeni vrh s najmanjom udaljenosti od korijena i vidimo da je sada to vrh "5"*. Promotrimo njegove izlazne neposjećene susjede i ako im je udaljenost do korijena manja od one koja je zapisana u tablici, promijenimo vrijednosti u tablici.*
Dodamo vrh "5" *u* S*.*
Sljedeći vrh kojeg biramo po istom pravilu kao i prošle je vrh "3"*. Opet, promotrimo izlazne susjede i ažuriramo tablicu ako je potrebno.*
Dodamo vrh "3" *u* S*.*
Nakon njega, promatramo vrh "8" *i opet ažuriramo tablicu udaljenosti.*
Dodamo vrh "8" *u* S*.*
Sljedeći vrh kojeg uzimamo je "7"*. Ažuriramo tablicu.*
Dodamo vrh "7" *u* S*.*
Sad uzimamo vrh "6" *i vidimo da za njega ne trebamo mijenjati ništa u tablici jer on nema izlaznih susjeda.*
Dodamo vrh "6" *u* S*.*
Ostaje nam još vrh "4" *za posjetiti. Njegov jedini izlazni susjed je vrh* "3"*. Ako promatramo udaljenosti, nova udaljenost do vrha* "3" *bi bila* $13+5=18$*, a trenutna u tablici je* 6*, pa ne mijenjamo ništa u tablici.*
Dodamo vrh "4" *u* S*.*
Sada imamo $S = V_D$*, pa algoritam staje. Konačne udaljenosti su zapisane u tablici u koraku 7.*

KORAK	**VRH**	1	2	3	4	5	6	7	8
1	**UDALJENOST**	0	∞	∞	∞	∞	∞	∞	∞
	RODITELJ	–	–	–	–	–	–	–	–
2	**UDALJENOST**	0	2	∞	∞	∞	20	∞	∞
	RODITELJ	–	1	–	–	–	1	–	–
3	**UDALJENOST**	0	2	∞	∞	5	20	∞	∞
	RODITELJ	–	1	–	–	2	1	–	–
4	**UDALJENOST**	0	2	6	∞	5	20	11	9
	RODITELJ	–	1	5	–	2	1	5	5
5	**UDALJENOST**	0	2	6	13	5	20	11	9
	RODITELJ	–	1	5	3	2	1	5	5
6	**UDALJENOST**	0	2	6	13	5	20	10	9
	RODITELJ	–	1	5	3	2	1	8	5
7	**UDALJENOST**	0	2	6	13	5	12	10	9
	RODITELJ	–	1	5	3	2	7	8	5

Tablica 3.1: *Dijkstrin* algoritam pretrage

3.2 Usmjerena pretraga po dubini (*DFS*)

Usmjerena pretraga po širini je dosta slična standardnoj pretrazi po širini, ali to ne vrijedi i za pretragu po dubini. Usmjerena pretraga po dubini se razlikuje od standardne. Naime, kad god grananje stane, odabere se novi neobojani vrh i pretraga se nastavlja uzimajući taj vrh kao novi korijen. Rezultat je razapinjuća razgranata šuma (engl. *spanning branching forest*) digrafa D koju zovemo ***DFS-razgranata šuma***.

Algoritam 3.4 (USMJERENA PRETRAGA PO DUBINI)

INPUT: digraf D

OUTPUT: razapinjuća razgranata šuma od D s roditeljskom funkcijom p i dvije vremenske funkcije f i l

postavi $i := 0$, $S := \emptyset$

while *postoji neobojani vrh* ***do***

  odaberi bilo koji neobojani vrh r kao korijen

  povećaj i za 1

  obojaj r u crno

  postavi $f(r) := i$

  dodaj r u S

  while *S nije prazan* ***do***

    uzmi vrh x stoga S

    povećaj i za 1

    if *x ima neobojanog vanjskog susjeda y* ***then***

      obojaj y u crno

      postavi $p(y) := x$, $f(y) := i$

      dodaj y na vrh stoga S

    else

postavi $l(x) := i$

ukloni x *iz* S

end if

end while

end while

vrati (p, f, l)

Da bi lakše razumijeli sljedeći teorem, definirat ćemo pojam **izlaznog stražnjeg, ulaznog stražnjeg i ukrštenog** luka (engl. *forward arc, back arc, cross arc*).
Reći ćemo da je luk $(u, v) \in A(D)\backslash A(F)$, gdje je F dobivena *DFS-razgranata* šuma, izlazni stražnji luk ako je vrh u predak od v u F. Luk (u, v) je ulazni stražnji luk ako je vrh u potomak od vrha v, a ako vrhovi u i v nisu susjedni u F i vrh u je pronađen nakon v, onda takav luk zovemo ukršteni luk. Ako te definicije pogledamo preko pojmova vremenske funkcije f i duljine l, onda imamo sljedeće:

- (u, v) je izlazni stražnji luk ako je $f(u) < f(v)$ i $l(v) < l(u)$
- (u, v) je ulazni stražnji luk ako je $f(v) < f(u)$ i $l(u) < l(v)$
- (u, v) je ukršteni luk ako je $l(v) < f(u)$

Sljedeći teorem nam daje particiju skupa $A(D)\backslash A(F)$.

Teorem 3.5 *Neka je F DFS-razgranata šuma digrafa D. Svaki luk od $A(D)\backslash A(F)$ je ili izlazni stražnji luk ili ulazni stražnji luk ili ukršteni luk.*

Poglavlje 4

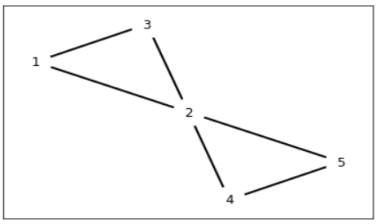
Implementacija algoritama - Python

Prije nego što vidimo kako sve ove algoritme možemo implementirati u programskom jeziku Python, vidjet ćemo kako nacrtati graf u Pythonu. Za to imamo sljedeći kod:

```python
# CRTANJE GRAFA

import networkx as nx
import matplotlib.pyplot as plt

class GraphVisualization:
    def __init__(self):
        self.visual = [] # u listu 'visual' spremamo bridove

    def addEdge(self, a, b): # bridovi se definiraju preko vrhova i dodaju u listu 'visual'
        self.visual.append([a, b])

    def visualize(self):
        G = nx.Graph() # G je objekt klase Graph
        G.add_edges_from(self.visual)
        nx.draw_networkx(G, node_color = "white",node_size=800, width = 2) # crtanje grafa
        plt.show() # prikazuje graf

G = GraphVisualization() # G je objekt gore definirane klase
G.addEdge(1, 2)
G.addEdge(1, 3)
G.addEdge(2, 3)
G.addEdge(2, 4)
G.addEdge(2, 5)
G.addEdge(4, 5)
G.visualize()
```

Slika 4.1: Kod za crtanje grafa - Python

Kao rezultat danog koda imamo sljedeći graf:



Sada slijede implementacije *BFS*, *DFS*, *Jarník-Prim algoritma* te *Boruvka-Kruskal algoritma* redom, zajedno sa pripadnim rezultatima svakog od algoritama.

```
# PRETRAGA PO ŠIRINI (BSF)

import collections # zbog reda

def bfs(graf, korijen):
    posjeceni = set() # posjećene ('obojane') vrhove ćemo spremati u ovaj skup
    Q = collections.deque([korijen]) # Q na početku sadrži samo korijen
    posjeceni.add(korijen) # 'bojanje' korijena

    while Q: # dok Q nije prazan
        vrh = Q.popleft() # izbacuje prvi element slijeva, tj ispisuje posjećene vrhove redom
        print(str(vrh) + " ", end="")

        # graf je spremljen u dictionary, vrh je ključ, a vrijednosti su njegovi susjedi
        for susjed in graf[vrh]:
            if susjed not in posjeceni: # ako susjed nije posjećen, onda ga dodaj u listu posjećenih i u Q
                posjeceni.add(susjed)
                Q.append(susjed) # append dodaje susjeda na desni kraj od Q

graf = {0: [1, 2, 3, 4], 1: [0,5,6], 2: [3], 3: [0, 3, 6, 7, 8],
            4: [0, 9, 10], 5: [1, 7, 11], 6: [1, 2, 3, 11], 7: [2, 5],
            8: [2, 10, 12], 9: [3, 4, 10, 12], 10: [4, 8, 9],
            11: [5, 6], 12: [8, 9]}
print("Pretraga stabla BFS: ")
bfs(graf, 0)
```

```
Pretraga stabla BFS:
0 1 2 3 4 5 6 7 8 9 10 11 12
```

Slika 4.2: Kod za BFS - Python

```python
# PRETRAGA PO DUBINI (DFS)

import collections # zbog reda

posjeceni = set() # skup posjećenih vrhova

def dfs(posjeceni, graf, vrh):
    if vrh not in posjeceni:
        print (str(vrh) + " ", end="")
        posjeceni.add(vrh)
        for susjed in graf[vrh]:
            dfs(posjeceni, graf, susjed)

graf = {0: [1, 2, 3, 4], 1: [0,5,6], 2: [3], 3: [0, 3, 6, 7, 8],
             4: [0, 9, 10], 5: [1, 7, 11], 6: [1, 2, 3, 11], 7: [2, 5],
             8: [2, 10, 12], 9: [3, 4, 10, 12], 10: [4, 8, 9],
             11: [5, 6], 12: [8, 9]}

print("Pretraga stabla DFS: ")
dfs(posjeceni, graf, 0)
```

```
Pretraga stabla DFS:
0 1 5 7 2 3 6 11 8 10 4 9 12
```

Slika 4.3: Kod za DFS - Python

```python
# PRIM ALGORITAM

INF = 9999999 # neka vrijednost koju vjerojatno nećemo imati u grafu
N = 9 # broj vrhova u grafu
A = [[0, 4, 0, 0, 0, 0, 0, 8, 0],  # matrica susjedstva
     [4, 0, 8, 0, 0, 0, 0, 11, 0], # (umjesto broja incidentnih bridova, imamo težine)
     [0, 8, 0, 7, 0, 4, 0, 0, 2],
     [0, 0, 7, 0, 9, 14, 0, 0, 0],
     [0, 0, 0, 9, 0, 10, 0, 0, 0],
     [0, 0, 4, 14, 10, 0, 2, 0, 0],
     [0, 0, 0, 0, 0, 2, 0, 1, 6],
     [8, 11, 0, 0, 0, 0, 1, 0, 7],
     [0, 0, 2, 0, 0, 0, 6, 7, 0]]

odabrani_brid = [0, 0, 0, 0, 0, 0, 0, 0, 0]
broj_bridova = 0
odabrani_brid[0] = True
```

```python
print("Brid :Težina\n")
while (broj_bridova < N - 1):
    minimum = INF
    a = 0
    b = 0
    for m in range(N):
        if odabrani_brid[m]:
            for n in range(N):
                if ((not odabrani_brid[n]) and A[m][n]): # ako postoji brid
                    if minimum > A[m][n]:
                        minimum = A[m][n]
                        a = m
                        b = n
    print(str(a) + "-" + str(b) + ":" + str(A[a][b]))
    odabrani_brid[b] = True
    broj_bridova += 1
```

```
Brid :Težina

0-1:4
0-7:8
7-6:1
6-5:2
5-2:4
2-8:2
2-3:7
3-4:9
```

Slika 4.4: Jarník-Prim algoritam - Python

```python
# KRUSKAL ALGORITAM

class Graph:
    def __init__(self, vertices):
        self.V = vertices
        self.graph = []

    def add_edge(self, u, v, w): # dodavanje bridova
        self.graph.append([u, v, w])

    def find(self, roditelj, i):
        if roditelj[i] == i:
            return i
        return self.find(roditelj, roditelj[i])

    def apply_union(self, roditelj, rang, x, y):
        xroot = self.find(roditelj, x)
        yroot = self.find(roditelj, y)
        if rang[xroot] < rang[yroot]:
            roditelj[xroot] = yroot
        elif rang[xroot] > rang[yroot]:
            roditelj[yroot] = xroot
        else:
            roditelj[yroot] = xroot
            rang[xroot] += 1
```

```
    def kruskal_algo(self):
        rez = []
        i= 0
        e = 0
        self.graph = sorted(self.graph, key=lambda item: item[2])
        roditelj = []
        rang = []
        for vrh in range(self.V):
            roditelj.append(vrh)
            rang.append(0)
        while e < self.V - 1:
            u, v, w = self.graph[i]
            i = i + 1
            x = self.find(roditelj, u)
            y = self.find(roditelj, v)
            if x != y:
                e = e + 1
                rez.append([u, v, w])
                self.apply_union(roditelj, rang, x, y)
        for u, v, weight in rez:
            print("%d - %d: %d" % (u, v, weight))
```

```
g = Graph(9) # navodimo broj vrhova
g.add_edge(0, 1, 4)
g.add_edge(0, 7, 8)
g.add_edge(1, 7, 11)
g.add_edge(1, 2, 8)
g.add_edge(2, 3, 7)
g.add_edge(2, 5, 4)
g.add_edge(2, 8, 2)
g.add_edge(3, 4, 9)
g.add_edge(3, 5, 14)
g.add_edge(4, 5, 10)
g.add_edge(5, 6, 2)
g.add_edge(6, 7, 1)
g.add_edge(6, 8, 6)
g.add_edge(7, 8, 7)
g.kruskal_algo()
```

```
6 - 7: 1
2 - 8: 2
5 - 6: 2
0 - 1: 4
2 - 5: 4
2 - 3: 7
0 - 7: 8
3 - 4: 9
```

Slika 4.5: Boruvka-Kruskal algoritam - Python

```python
# DIJKSTRIN ALGORITAM

import sys

vrhovi = [[0, 1, 0, 0, 0, 1, 0, 0], # matrica incidencije
          [0, 0, 0, 0, 1, 0, 0, 0],
          [0, 0, 0, 1, 0, 0, 0, 0],
          [0, 0, 1, 0, 0, 0, 0, 0],
          [0, 0, 1, 0, 0, 0, 1, 1],
          [0, 0, 0, 0, 0, 0, 0, 0],
          [0, 0, 0, 0, 0, 1, 0, 0],
          [0, 0, 0, 0, 1, 0, 1, 0]]

bridovi_tezine = [[0, 2, 0, 0, 0, 20, 0, 0],  # incidencija s težinama
          [0, 0, 0, 0, 3, 0, 0, 0],
          [0, 0, 0, 7, 0, 0, 0, 0],
          [0, 0, 5, 0, 0, 0, 0, 0],
          [0, 0, 1, 0, 0, 0, 6, 4],
          [0, 0, 0, 0, 0, 0, 0, 0],
          [0, 0, 0, 0, 0, 2, 0, 0],
          [0, 0, 0, 0, 2, 0, 1, 0]]

def neposjeceni():
    global posj_udaljenost # globalna jer ju koristimo i izvan ove funkcije
    p = -10
    for i in range(broj_vrhova):
        if posj_udaljenost[i][0] == 0 and (p < 0 or posj_udaljenost[i][1] <=
                posj_udaljenost[p][1]):
            p = i
    return p
```

```python
broj_vrhova = len(vrhovi[0])

posj_udaljenost = [[0, 0]]
for i in range(broj_vrhova-1):
    posj_udaljenost.append([0, sys.maxsize])

for vrh in range(broj_vrhova):
    posjetiti = neposjeceni()
    for susjed in range(broj_vrhova):
        if vrhovi[posjetiti][susjed] == 1 and posj_udaljenost[susjed][0] == 0:
            nova_d = posj_udaljenost[posjetiti][1] + bridovi_tezine[posjetiti][susjed]
            if posj_udaljenost[susjed][1] > nova_d:
                posj_udaljenost[susjed][1] = nova_d

        posj_udaljenost[posjetiti][0] = 1

i = 0

for d in posj_udaljenost:
    print("Udaljenost ", 0 + i,
          " od korijena: ", d[1])
    i = i + 1
```

```
Udaljenost  0  od korijena:  0
Udaljenost  1  od korijena:  2
Udaljenost  2  od korijena:  6
Udaljenost  3  od korijena:  13
Udaljenost  4  od korijena:  5
Udaljenost  5  od korijena:  12
Udaljenost  6  od korijena:  10
Udaljenost  7  od korijena:  9
```

Slika 4.6: Dijkstra algoritam - Python

Poglavlje 5

Zaključak

Kako je i napisano u uvodu rada, ideja je bila upoznavanje s teorijom grafova kako bi ju znali primijeniti u praktičnim situacijama. Upoznali smo se s različitim vrstama pretraga po stablima, kao što su pretraživanje po širini (*BFS*), pretraživanje po dubini (*DFS*) i s različitim načinima za pronalazak minimalnog razapinjućeg stabla T grafa G, a to su *Jarník-Prim* algoritam, *Boruvka-Kruskal* algoritam i *Dijkstrin* algoritam. Svi navedeni algoritmi su dosta primjenjivi danas u raznim situacijama i ispravno je koristiti bilo koji od njih. Teorija grafova će zbog svoje jednostavne implementacije u računalu sigurno i dalje ostati primjenjiva i korištena za modeliranje problema.

Literatura

[1] O. T. Arogundade and A. T. Akinwale, "Application of Prim's Algorithm to a Profit-Oriented Transportation System in Rural Areas of Nigeria." University of Agriculture, Abeokuta 37 (2009): 4-9.

[2] J.A. Bondy and U.S.R. Murty, "Graph Theory, Series." Graduate Texts in Mathematics 244.

[3] Anka Golemac, Teorija grafova, nastavni tekst, 2021.

TEMELJNA DOKUMENTACIJSKA KARTICA

PRIRODOSLOVNO–MATEMATIČKI FAKULTET
SVEUČILIŠTA U SPLITU
ODJEL ZA MATEMATIKU

DIPLOMSKI RAD

ALGORITMI PRETRAGE U STABLU

Antonela Pintur

Sažetak:

Teorija grafova je danas iznimno popularna grana matematike sa širokim primjenama. Najčešće primjene su u rješavanju problema pronalaska najkraćih puteva kako bi se minimizirali razni troškovi pa je korisno pogledati različite algoritme pretrage koje imamo u stablima. Najpoznatiji algoritmi su Jarnik-Prim algoritam i Boruvka-Kruskal kojima je jedina razlika u redoslijedu odabira bridova ("puteva"), a kao rezultat daju jednako stablo.

Ključne riječi:

graf, minimalno razapinjuće stablo, Jarnik-Prim, Boruvka-Kruskal, Dijkstra

Podatci o radu:

51 stranica, 26 slika, 3 tablice, 3 literaturna navoda, hrvatski jezik

Mentorica: *doc.dr.sc. Tanja Vojković*

Članovi povjerenstva:

dr. sc. Ana Laštre, pred.

doc.dr.sc. Goran Erceg

Povjerenstvo za diplomski rad je prihvatilo ovaj rad *11. svibnja 2023.*

TEMELJNA DOKUMENTACIJSKA KARTICA

FACULTY OF SCIENCE, UNIVERSITY OF SPLIT
DEPARTMENT OF MATHEMATICS

MASTER'S THESIS

TREE-SEARCH ALGORITHMS

Antonela Pintur

Abstract:

Graph theory is lately an extremely popular branch of mathematics with wide applications. The most common applications are in solving problems of finding the shortest paths to minimize various costs, so it is useful to look and understand different types of search algorithms that are used in trees. The most famous algorithms are Jarnik-Prim algorithm and Boruvka-Kruskal. The only difference between these two is in the order selection of edges („paths") but the result is the same tree in both cases.

Key words:

graph, minimal spanning tree, Jarnik-Prim, Boruvka-Kruskal, Dijkstra

Specifications:

51 pages, 26 images, 3 tables, 3 references, Croatian language

Mentor: *assisstant professor Tanja Vojković*

Committee:

Ana Laštre, Phd, Lecturer

assisstant professor Goran Erceg

This thesis was approved by a Thesis commettee on *May 11, 2023.*