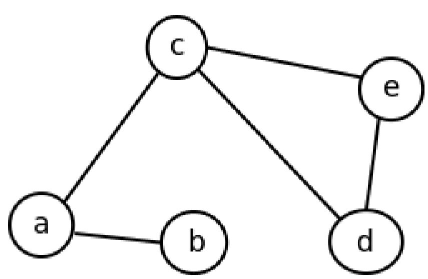
Grafovski algoritmi

Osnovni pojmovi

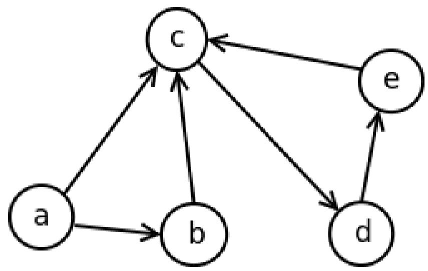
Grafovi su jedna od najkorisnijih struktura podataka. Jedan od najranijih primera grafova su mreže puteva i odgovarajuće mape. Poznat je slučaj srednjevekovne kopije mape iz petog veka koja je sadržala mrežu puteva Rimskog carstva sa imenima gradova i dužinama puteva između njih, dakle sa dovoljno informacija potrebnih da se pronađe najkraći put između dva grada. Još jedna od klasičnih primena grafova (specijalno drveta) je predstavljanje genealogija. Naime, porodična stabla su vekovima korišćena u svrhe odgovora na pravna pitanja poput pitanja dozvoljenih brakova, nasledstva i nasleđivanja vlasti. Naravno, postoji mnogo drugih poznatih primera grafova, kao što su predstavljanje strana i ivica poliedara, komunikacione mreže, električna kola, strukturne formule molekula, društvene igre, traženje izlaza iz lavirinta, a u današnje vreme i društvene mreže.

Graf $G = (V, E)$ se sastoji od skupa V *čvorova* i skupa E *grana*. Najčešće grana odgovara paru različitih čvorova, mada su ponekad dozvoljene i petlje, odnosno grane koje vode od čvora ka njemu samom. Graf može biti *neusmeren* (slika 1) ili *usmeren* (slika 2). Grane usmerenog grafa su uređeni parovi čvorova i kod njih je važan redosled čvorova koje grana povezuje. Ako se graf predstavlja grafički, onda se grane usmerenog grafa crtaju kao strelice usmerene od jednog čvora (početka) ka drugom čvoru (kraju grane). Grane neusmerenog grafa su neuređeni parovi čvorova: one se crtaju kao obične duži.



Slika 1: Primer neusmerenog grafa.

Susedom čvora v nazvaćemo sve čvorove u do kojih postoji grana iz čvora v. Susedi čvora c u neusmerenom grafu sa slike 1 su čvorovi a, d i e, dok je u usmerenom grafu sa slike 2 jedini sused čvora c čvor d. *Stepen* $d(v)$ čvora v je broj grana susednih čvoru v (odnosno broj grana koje čvor v povezuju sa nekim drugim čvorom). U usmerenom grafu razlikujemo *ulazni stepen* čvora v koji je



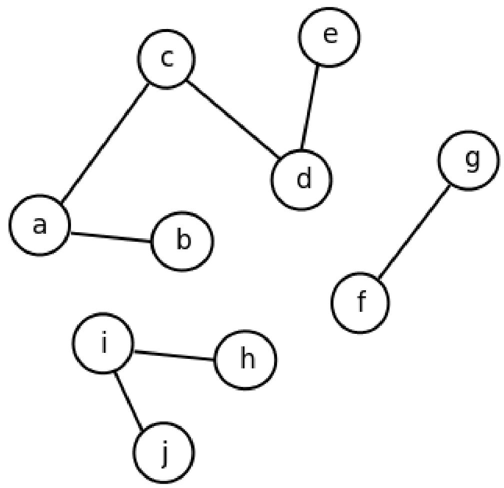
Slika 2: Primer usmerenog grafa.

jednak broju grana za koje je čvor v kraj, odnosno *izlazni stepen* čvora v koji je jednak broju grana za koje je čvor v početak. Na primer, stepen čvora c u neusmerenom grafu sa slike 1 je 3, dok za čvor c usmerenog grafa sa slike 2 važi da mu je ulazni stepen jednak 3, a izlazni stepen jednak 1.

Put od čvora v_1 do čvora v_k u grafu $G = (V, E)$ je niz čvorova grafa $v_1, v_2, \dots, v_k$ povezanih granama grafa $(v_1, v_2), (v_2, v_3), \dots, (v_{k-1}, v_k)$. Put je *prost* ako se svaki čvor u njemu pojavljuje samo jednom. Na primer, niz čvorova a, b, c, d, e u usmerenom grafu sa slike 2 predstavlja jedan prost put u tom grafu, dok put b, c, d, e, c nije prost. Za čvor u se kaže da je *dostižan* iz čvora v ako postoji put (usmeren, odnosno neusmeren, zavisno od grafa) od v do u. Npr. čvor e grafa sa slike 2 dostižan je iz čvora a. Po definiciji čvor v je dostižan iz v. *Ciklus* je put čiji se prvi i poslednji čvor poklapaju. Ciklus je *prost* ako se, sem prvog i poslednjeg čvora, ni jedan drugi čvor u njemu ne pojavljuje dva puta. Niz čvorova c, d, e predstavlja prost ciklus i u datom usmerenom i u datom neusmerenom grafu.

Neusmereni oblik usmerenog grafa $G = (V, E)$ je isti graf, bez smerova na granama (tako da su parovi čvorova u E neuređeni). Za neusmeren graf se kaže da je *povezan* ako postoji put između proizvoljna dva čvora u grafu. Graf sa slike 1 je povezan. Za usmerene grafove razlikujemo pojam slabe i jake povezanosti: usmereni graf je *slabo povezan* ako u njegovom neusmerenom obliku postoji put između svaka dva čvora u grafu, a *jako povezan* ako za svaka dva čvora u i v u grafu postoji usmeren put od čvora u do čvora v i usmeren put od čvora v do čvora u. Graf sa slike 2 je slabo povezan, ali nije jako povezan: naime, od čvora b nije moguće stići do čvora a. Neusmereni graf je *šuma* ako ne sadrži cikluse (slika 3). *Drvo* je povezana šuma.

Graf $H = (U, F)$ je *podgraf* grafa $G = (V, E)$ ako je $U \subseteq V$ i $F \subseteq E$. Na primer, graf sa skupom čvorova $\{a, b, c, d\}$ i skupom grana $\{(a, b), (a, c), (c, d)\}$ je jedan podgraf usmerenog grafa sa slike 2. *Povezujuće drvo* neusmerenog grafa G je njegov podgraf koji je drvo i sadrži sve čvorove grafa G. Na primer, jedno povezujuće drvo neusmerenog grafa sa slike 1 bi sadržalo sve čvorove grafa i skup grana $\{(a, b), (a, c), (c, d), (c, e)\}$. *Povezujuća šuma* neusmerenog grafa G je njegov podgraf koji je šuma i sadrži sve čvorove grafa G. Ako neusmereni graf $G = (V, E)$ nije povezan, onda se on može na jedinstven način razložiti



Slika 3: Primer grafa koji je šuma.

u skup povezanih podgrafova, koji predstavljaju klase ekvivalencije za relaciju dostižnosti i koji se nazivaju *komponente povezanosti* grafa G.

Predstavljanje grafa

Uobičajena su dva načina predstavljanja grafova. Prvi je *matricom povezanosti*, odnosno *matricom susedstva* grafa. Neka je $|V| = n$ i $V = \{v_0, v_1, \ldots, v_{n-1}\}$. Matrica povezanosti grafa G je kvadratna matrica $A = (a_{ij})$ reda n, sa elementima a_{ij} koji su jednaki 1 ako i samo ako $(v_i, v_j) \in E$, odnosno ako postoji grana od čvora v_i do čvora v_j; ostali elementi matrice A imaju vrednost nula. Ako je graf neusmeren, matrica A je simetrična. Vrsta i ove matrice je dakle niz dužine n čija je j-ta koordinata jednaka 1 ako iz čvora v_i vodi grana ka čvoru v_j, odnosno 0 u protivnom. Nedostatak matrice povezanosti je to što ona uvek zauzima prostor veličine n^2, nezavisno od toga koliko grana ima graf. Ako je broj grana u grafu mali, većina elemenata matrice povezanosti biće jednaka nula. Ako se za predstavljanje grafa koristi matrica povezanosti, složenost operacije dodavanja neke grane u graf, odnosno operacije uklanjanja neke grane iz grafa je $O(1)$, a takođe i ispitivanje da li su dva čvora u grafu povezana granom je složenosti $O(1)$. Listanje svih čvorova susednih datom čvoru v je složenosti $O(|V|)$.

U jeziku C++ graf predstavljen matricom povezanosti grafa možemo deklarisati na sledeći način:

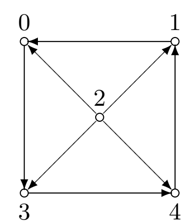
```
bool matricaPov[max][max]
```

ili, ako broj čvorova saznajemo tek u vreme izvršavanja programa:

```
vector<vector<bool>> matricaPov(n);
for (int i = 0; i < n; i++)
   matricaPov[i].resize(n);
```

Umesto da se i sve nepostojeće grane eksplicitno predstavljaju u matrici povezanosti, mogu se formirati povezane liste od jedinica iz i-te vrste, $i = 0, 1, \ldots, n-1$. Ovaj drugi način predstavljanja grafa naziva se *lista povezanosti*, odnosno *lista susedstva*. Svakom čvoru pridružuje se povezana lista, koja sadrži sve grane susedne tom čvoru, tačnije sve čvorove do kojih postoji grana iz tog čvora. Lista može biti uređena prema rednim brojevima čvorova na krajevima njenih grana. Graf je predstavljen nizom lista. Svaki element niza sadrži ime (indeks) čvora i pokazivač na njegovu listu suseda. Treba napomenuti da iako ime tako sugeriše, implementacija ovakve reprezentacije grafa ne mora biti zasnovana na listama, već se umesto povezanih listi može koristiti dinamički niz ili neka vrsta balansiranih binarnih drveta ili pak heš tabela.

	0	1	2	3	4
0	0	0	0	1	0
1	1	0	0	0	0
2	1	1	0	1	1
3	0	0	0	0	1
4	0	1	0	0	0

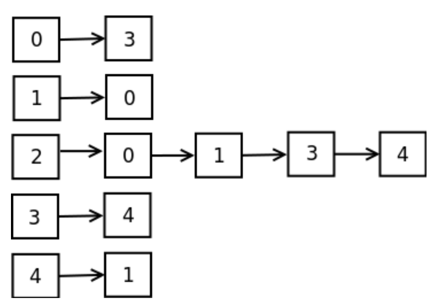


(a)

5	6	7	11	12	3	0	0	1	3	4	4	1
0	1	2	3	4	5	6	7	8	9	10	11	12

(b)

Slika 4: Predstavljanje grafa sa slike matricom povezanosti (a), odnosno listom povezanosti u vidu statičkog niza (b).



Slika 5: Predstavljanje grafa listom povezanosti.

Ako je graf *statički*, odnosno nisu dozvoljena umetanja i brisanja čvorova i grana, onda se graf može predstaviti statičkim nizom a dužine $|V| + |E|$ u slučaju usmerenog grafa (odnosno dužine $|V| + 2|E|$ u slučaju neusmerenog grafa). Prvih

$|V|$ članova niza su pridruženi čvorovima. Element niza a na poziciji i, $i < |V|$ je pridružen čvoru v_i i sadrži indeks početka spiska čvorova susednih čvoru v_i, $i = 0, 1, \ldots, n-1$, dok se na pozicijama i, $|V| \leq i < |V| + |E|$ nalaze informacije o susedima čvorova. Naime, susedi čvora v_i za $0 \leq i < n-1$ nalaze se u nizu a na pozicijama $[a[i], a[i+1])$, dok se susedi čvora v_{n-1} nalaze na pozicijama od $a[n-1]$ do kraja niza.

Na slikama 4 i 5 prikazana su na jednom primeru oba načina predstavljanja grafa u slučaju kada je graf statički. Čvorovi su numerisani brojevima od 0 do 4. Na poziciji $(1, 0)$ u matrici povezanosti nalazi se vrednost 1 jer postoji grana iz čvora 1 ka čvoru 0, dok se na poziciji $(0, 1)$ nalazi 0 jer ne postoji suprotno orijentisana grana u grafu. Lista povezanosti pridružena čvoru 1 je dužine 1 jer iz čvora 1 polazi tačno jedna grana, slično, lista povezanosti pridružena čvoru 2 je dužine 4 jer iz čvora 2 polaze četiri grane. Razmotrimo sada reprezentaciju statičkim nizom: na poziciji 0 nalazi se vrednost 5, a na poziciji 1 vrednost 6, što ukazuje na to da se na pozicijama $[5, 6)$ u ovom nizu nalaze susedi čvora 0 – u ovom slučaju to je samo čvor 3 jer iz čvora 0 postoji samo jedna grana i to ka čvoru 3. Susedi čvora 2 nalaze se na pozicijama $[7, 11)$ i to su redom $0, 1, 3$ i 4.

Sa matricama povezanosti je jednostavnije raditi. S druge strane, liste povezanosti su prostorno efikasnije za grafove sa malim brojem grana (njihova memorijska složenost je $O(|V| + |E|)$, za razliku od matrica povezanosti čija je memorijska složenost $O(|V|^2)$). U praksi se često radi sa grafovima koji imaju znatno manje grana od maksimalnog mogućeg broja ($n(n-1)/2$ za neusmereni, odnosno $n(n-1)$ za usmereni graf), i tada je efikasnije koristiti liste povezanosti. U slučaju kada se za implementaciju liste povezanosti koriste povezane liste, ispitivanje da li su dva čvora u grafu povezana je u najgorem slučaju složenosti $O(|V|)$, a takođe i uklanjanje grane iz grafa je u najgorem slučaju vremenske složenosti $O(|V|)$. U slučaju kada se za implementaciju lista povezanosti koriste heš tabele, očekivano vreme izvršavanja ovih operacija je $O(1)$. Listanje svih grana susednih čvoru v je složenosti $O(1 + d(v))$, gde je $d(v)$ stepen čvora v u slučaju neusmerenog grafa, odnosno izlazni stepen čvora v ako je graf usmeren. Dodavanje novog čvora u graf je jednostavnije nego u slučaju reprezentacije grafa matricom povezanosti.

U jeziku C++ za deklaraciju grafa predstavljenog listama povezanosti možemo iskoristiti naredni fragment koda:

```
vector<vector<int>> listaSuseda(n);
```

Na primer, usmereni graf sa slike 4 predstavićemo kao:

```
vector<vector<int>> listaSuseda {{3}, {0}, {0, 1, 3, 4}, {4}, {1}};
```

Broj čvorova grafa možemo dobiti kao broj elemenata u spoljašnjem vektoru:

```
int brCvorova = listaSuseda.size();
```

Novu granu `(cvorOd,cvorDo)` u graf možemo dodati na sledeći način:

```
listaSuseda[cvorOd].push_back(cvorDo);
```

dok kroz sve susede čvora možemo iterirati na sledeći način:

```
for (int cvorDo : listaSuseda[cvorOd])
    ...
```

Treba pomenuti da postoje i drugi načini za predstavljanje grafa, na primer graf se može čuvati kao niz grana, gde se za svaku granu čuva informacija sa kojim čvorovima je incidentna.

U narednim algoritmima smatraćemo da je graf sa kojim radimo dinamički i da je zadat listom povezanosti.

Obilasci grafova

Prvi problem na koji se nailazi pri konstrukciji bilo kog algoritma za obradu grafa je kako pregledati ulaz. U slučaju nizova i skupova taj problem je trivijalan zbog jednodimenzionalnosti ulaza — nizovi i skupovi mogu se lako pregledati linearnim redosledom. Pregledanje grafa, odnosno njegov *obilazak*, nije trivijalan problem. Postoje dva osnovna algoritma za obilazak, odnosno pretragu grafa: *pretraga u dubinu* i *pretraga u širinu*.

U opštem slučaju, obilazak grafa ima sledeći kostur:

```
dodaj cvor s u kolekciju C
sve dok C nije prazno
   uzmi cvor v iz C
   ako je v neoznaceno
     oznaci v
     za svaku granu (v,w)
        ubaci w u C
```

Ukoliko se kolekcija predstavi stekom, dobijamo algoritam pretrage u dubinu, ako kolekciju implementiramo kao red, dobijamo algoritam pretrage u širinu. Konačno, ako kolekciju implementiramo kao red sa prioritetom dobijamo pretragu prema prioritetu, gde bi se, ako prioritet razmatramo kao težinu grane dobilo minimalno povezujuće drvo datog grafa, a ako bismo kao prioritet razmatrali rastojanje od polaznog čvora mogli izračunati najkraći putevi od polaznog čvora do svih ostalih čvorova u datom grafu.

Pretraga u dubinu

Pretraga u dubinu (DFS, skraćenica od depth–first–search) je praktično ista za neusmerene i usmerene grafove. Međutim, pošto želimo da ispitamo neke osobine grafova koje nisu iste za neusmerene i usmerene grafove, razmatranje će biti podeljeno na dva dela: za neusmerene i usmerene grafove.

Neusmereni grafovi

Pretpostavimo da je zadat graf $G = (V, E)$. Želimo da izvršimo obilazak grafa tako da uvek kada je to moguće idemo dalje u dubinu pre nego što se vratimo unazad. Ovaj pristup zove se *pretraga u dubinu* (DFS). Osnovni razlog korisnosti pretrage u dubinu leži u njenoj jednostavnosti i lakoj realizaciji rekurzivnim algoritmom.

Razmotrimo problem pretrage u dubinu kada je graf zadat listom povezanosti i dat je čvor r grafa iz koga se započinje pretraga. Inicijalno su svi čvorovi neoznačeni. Čvor r se *označava* kao posećen. Zatim se u listi suseda čvora r pronalazi prvi neoznačeni sused r_1 čvora r, pa se iz čvora r_1 rekurzivno pokreće pretraga u dubinu. Iz nekog nivoa rekurzije, pokrenutog iz čvora v, izlazi se ako su svi susedi (ako ih ima) čvora v iz koga je pretraga pokrenuta već označeni. Ako su u trenutku završetka pretrage iz čvora r_1 svi susedi čvora r označeni, onda se pretraga za čvor r završava. U protivnom se u listi suseda čvora r pronalazi sledeći neoznačeni sused r_2, izvršava se pretraga polazeći od r_2, itd.

Na primer, u slučaju neusmerenog grafa prikazanog na slici 1 predstavljenog listom susedstva tako da su čvorovi u svakoj od listi numerisani leksikografski rastuće, ako bi se pokrenula pretraga u dubinu iz čvora c redom bi se obilazili čvorovi c, a, b, d, e, a ako bi se pokrenula iz čvora b, obilazili bi se redom čvorovi b, a, c, d, e.

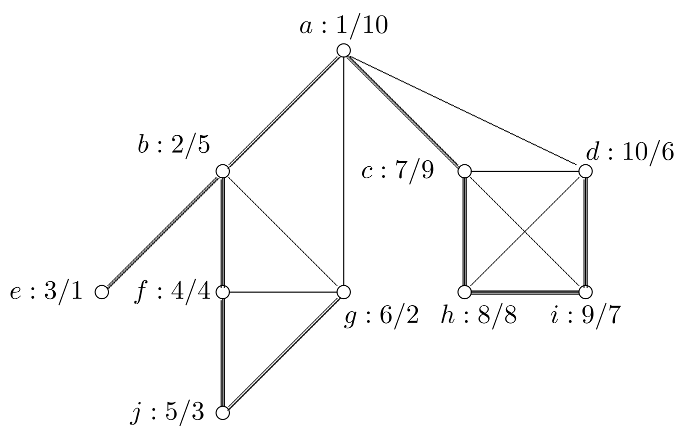
Pretraga grafa uvek se vrši sa nekim ciljem. Da bi se različite primene uklopile u pretragu u dubinu, poseti čvora ili grane pridružuju se dve vrste obrade, *ulazna obrada* i *izlazna obrada*. Ulazna obrada vrši se u trenutku označavanja čvora. Izlazna obrada vrši se na kraju, kada obradimo sve potomke datog čvora. Ulazna i izlazna obrada zavise od konkretne primene algoritma DFS. Na taj način moguće je rešavanje različitih problema jednostavnim definisanjem ulazne i izlazne obrade. Algoritam pretrage u dubinu dat je u nastavku.

Jednostavnosti radi, graf će u narednim kodovima biti deklarisan kao globalna promenljiva.

```
// reprezentacija grafa listom povezanosti
// graf je neusmeren, pa se svaka grana dva puta javlja u listi
vector<vector<int>> listaSuseda
     {{1, 2}, {0, 3, 4}, {0, 5}, {1}, {1, 6, 7},
     {2, 8}, {4}, {4}, {5}};

void dfs(int cvor, vector<bool> &posecen){
  posecen[cvor] = true;
  // ovde ide ulazna obrada

  // rekurzivno prolazimo kroz sve njegove susede
  // koje ranije nismo obisli
  for (auto sused : listaSuseda[cvor]){
```

```
    if (!posecen[sused])
      dfs(sused,posecen);
  }
  // ovde ide izlazna obrada
}

// funkcija koja vrsi DFS obilazak datog grafa iz datog cvora
void dfs(int cvor){
  int brojCvorova = listaSuseda.size();
  vector<bool> posecen(brojCvorova, false);
  dfs(cvor,posecen);
}

int main(){
  dfs(0);
  return 0;
}
```

Primer pretrage grafa u dubinu prikazan je na slici 6.



$a :\ b, g, c, d$
$b :\ a, e, f, g$
$c :\ a, h, i, d$
$d :\ a, c, h, i$
$e :\ b$
$f :\ b, j, g$
$g :\ a, b, f, j$
$h :\ c, i, d$
$i :\ d, c, h$
$j :\ f, g$

Slika 6: Primer pretrage grafa u dubinu kada je dat odgovarajućim listama povezanosti. Dva broja uz čvor jednaka su njegovim rednim brojevima pri dolaznoj, odnosno odlaznoj DFS numeraciji.

Lema: Ako je graf G povezan, onda su po završetku pretrage u dubinu svi čvorovi označeni, a sve grane grafa G su pregledane bar po jednom.

Dokaz: Pretpostavimo suprotno, i označimo sa U skup neoznačenih čvorova zaostalih posle izvršavanja algoritma. Pošto je graf G povezan, bar jedan čvor u iz U mora biti povezan granom sa nekim označenim čvorom w (skup označenih čvorova je neprazan jer sadrži bar čvor v). Međutim, ovako nešto je nemoguće, jer kad se poseti čvor w, moraju biti posećeni (pa dakle i označeni) svi njegovi neoznačeni susedi, dakle i čvor u. Pošto su svi čvorovi posećeni, a kad se čvor poseti, onda se pregledaju sve grane koje vode iz njega, zaključujemo da su i sve grane grafa pregledane. □

Prilikom izvršavanja DFS algoritma na neusmerenom grafu $G = (V, E)$ koji je zadat listom povezanosti, svaka grana se pregleda tačno dva puta, po jednom sa svakog kraja. Prema tome, ukupan broj izvršavanja tela `for` petlje u svim rekurzivnim pozivima algoritma DFS je $O|E|$. S druge strane, broj rekurzivnih poziva je $|V|$, pa se složenost DFS algoritma može opisati izrazom $O(|V| + |E|)$. Primetimo da složenost pretrage zavisi od reprezentacije grafa: naime, ako bi graf bio zadat matricom povezanosti, onda bi prolaz kroz spisak suseda jednog fiksiranog čvora podrazumevao prolaz kroz odgovarajuću vrstu matrice, što je složenosti $\Theta(n)$. Onda bi prolaz kroz susede svakog od čvorova bio složenosti $\Theta(n^2)$. Dakle, u slučaju algoritma pretrage efikasnija implementacija se dobija korišćenjem reprezentacije grafa listom povezanosti.

Za nepovezane grafove se algoritam DFS mora promeniti. Ako su svi čvorovi označeni posle prvog pokretanja opisanog algoritma, onda je graf povezan, i obilazak je završen. U protivnom, može se pokrenuti nova pretraga u dubinu polazeći od proizvoljnog neoznačenog čvora u grafu, itd. Prema tome, DFS se može iskoristiti da bi se ustanovilo da li je graf povezan, odnosno za pronalaženje svih njegovih komponenti povezanosti.

```
vector<vector<int>> listaSuseda
     {{1, 2}, {3}, {}, {}, {6, 7}, {8}, {}, {}, {}};

// obilazak u dubinu iz cvora cvor
void dfs(int cvor, vector<bool> &posecen,
         int brojKomponente, vector<int>& komponente) {
  // tekuci cvor pridruzujemo tekucoj komponenti
  posecen[cvor] = true;
  komponente[cvor] = brojKomponente;

  // rekurzivno prolazimo kroz sve njegove susede
  // koje ranije nismo obisli
  for (auto sused : listaSuseda[cvor])
    if (!posecen[sused])
      dfs(sused, posecen, brojKomponente, komponente);
}

// za svaki cvor grafa u vektor komponente upisuje
// redni broj komponente kojoj pripada i
// vraca ukupan broj komponenata povezanosti
int komponentePovezanosti(vector<int> &komponente) {
  int brojCvorova = listaSuseda.size();
  vector<bool> posecen(brojCvorova, false);
  int brojKomponente = 1;
  // pokrecemo novu DFS pretragu iz prvog neposecenog cvora
```

```
  for (int cvor = 0; cvor < brojCvorova; cvor++)
    if (!posecen[cvor]) {
      dfs(cvor, posecen, brojKomponente, komponente);
      brojKomponente++;
    }
  return brojKomponente;
}

int main() {
  // odredjujemo komponente povezanosti
  int brojCvorova = listaSuseda.size();
  vector<int> komponente(brojCvorova);
  int brojKomponenti = komponentePovezanosti(komponente);

  // ispisujemo rezultat
  cout << "Ukupan broj komponenti povezanosti je "
       << brojKomponenti - 1 << endl;
  for (int i = 0; i < brojCvorova; i++)
    cout << "Cvor " << i << " pripada komponenti "
         << komponente[i] << endl;
  return 0;
}
```

I ovaj algoritam bi bio vremenske složenosti $O(|V| + |E|)$.

Mi ćemo najčešće razmatrati samo povezane grafove, jer se u opštem slučaju problem svodi na posebnu obradu svake komponente povezanosti.

Konstrukcija DFS drveta i DFS numeracija

Prikazaćemo sada dve jednostavne a važne primene algoritma DFS — formiranje specijalnog povezujućeg drveta, takozvanog *DFS drveta* i numeraciju čvorova grafa *DFS brojevima*.

Prilikom obilaska grafa G u dubinu mogu se u petlji kojom se prolaze svi susedi čvora v izdvojiti sve grane ka novooznačenim čvorovima w. Preko izdvojenih grana dostižni su svi čvorovi povezanog neusmerenog grafa, pa je podgraf koga čine izdvojene grane povezan. Taj podgraf nema cikluse, jer se od svih grana koje vode u neki čvor, može izdvojiti samo jedna. Prema tome, izdvojene grane su grane podgrafa grafa G koji je u slučaju povezanog grafa povezujuće drvo, tzv. *DFS drvo* grafa G. Polazni čvor je koren DFS drveta. Čak i ako se drvo ne formira eksplicitno, mnoge algoritme je lakše razumeti razmatrajući DFS drvo.

U nastavku je dat algoritam konstrukcije DFS drveta datog grafa predstavljenog listom susedstva. DFS drvo je predstavljeno u vidu vektora grana.

```
vector<vector<int>> listaSuseda
    {{1, 2}, {3, 4}, {5}, {}, {6, 7}, {8}, {}, {}, {}};
```

```
void dfs(int cvor, vector<bool> &posecen,
     vector<vector<int>> &dfs_drvo){
  posecen[cvor] = true;

  // rekurzivno prolazimo kroz sve njegove susede
  // koje ranije nismo obisli
  for (auto sused : listaSuseda[cvor]){
    if (!posecen[sused]){
      // u DFS drvo dodajemo granu iz tekuceg ka novom cvoru
      // i njoj suprotno usmerenu granu (jer je graf neusmeren)
      dfs_drvo[cvor].push_back(sused);
      dfs_drvo[sused].push_back(cvor);
      dfs(sused,posecen,dfs_drvo);
    }
  }
}

// funkcija koja vrsi DFS obilazak datog grafa iz datog cvora
void dfs(int cvor){
  int brojCvorova = listaSuseda.size();
  vector<bool> posecen(brojCvorova, false);
  vector<vector<int>> dfs_drvo(brojCvorova);
  dfs(cvor,posecen,dfs_drvo);

  cout << "Grane DFS drveta su: ";
  for (int i = 0; i < dfs_drvo.size(); i++)
    for (int j = 0; j < dfs_drvo[i].size(); j++)
      cout << "(" << i << "," << dfs_drvo[i][j] << ")" << endl;

}

int main(){
  dfs(0);
  return 0;
}
```

Postoje dve varijante DFS numeracije: čvorovi se mogu numerisati prema redosledu označavanja čvorova (*dolazna DFS numeracija*, odnosno *preOrder* numeracija), ili prema redosledu napuštanja čvorova (*odlazna DFS numeracija*, odnosno *postOrder* numeracija). Primer grafa sa čvorovima numerisanim na dva načina prikazan je na slici 6. Numeracija čvorova će nam biti od koristi za rešavanje raznih problema nad grafovima.

Algoritmi kojima se čvorovi ispisuju u redosledu dolazne, odnosno odlazne numeracije opisani su narednim kodovima. Za primer sa slike 6 za dolaznu numeraciju bi se ispisali čvorovi u redosledu $a, b, e, f, j, g, c, h, i, d$, a za odlaznu numeraciju u redosledu $e, g, j, f, b, d, i, h, c, a$.

```
void dfs_preorder(int cvor, vector<bool> &posecen){
  posecen[cvor] = true;
  // stampamo naredni cvor u preOrder numeraciji
  cout << cvor << " ";

  // rekurzivno prolazimo kroz sve susede koje nismo obisli
  for (auto sused : listaSuseda[cvor]){
    if (!posecen[sused])
      dfs_preorder(sused,posecen);
  }
}

void dfs_postorder(int cvor, vector<bool> &posecen){
  posecen[cvor] = true;

  // rekurzivno prolazimo kroz sve susede koje nismo obisli
  for (auto sused : listaSuseda[cvor]){
    if (!posecen[sused])
      dfs_postorder(sused,posecen);
  }
  // stampamo naredni cvor u postOrder numeraciji
  cout << cvor << " ";
}
```

Alternativno, možemo implementirati funkcije kojima se za svaki čvor u grafu ispisuje njegova vrednost dolazne i odlazne numeracije.

```
int vreme_dolazna = 1;
int vreme_odlazna = 1;
vector<int> dolazna;
vector<int> odlazna;

void dfs(int cvor, vector<bool> &posecen) {
  posecen[cvor] = true;

  // cvor dobija narednu mogucu vrednost dolazne numeracije
  dolazna[cvor] = vreme_dolazna;
  vreme_dolazna++;

  // rekurzivno prolazimo kroz sve susede koje do sada nismo obisli
  for (auto sused : listaSuseda[cvor]) {
    if (!posecen[sused] == -1) {
      dfs(sused,posecen);
    }
  }
  // cvor dobija narednu mogucu vrednost odlazne numeracije
```

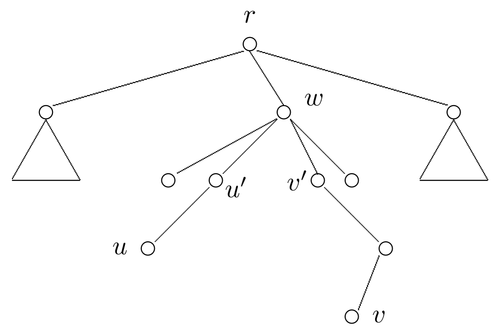
```
  odlazna[cvor] = vreme_odlazna;
  vreme_odlazna++;
}

void dfs(int cvor) {
  int brojCvorova = listaSuseda.size();
  dolazna.resize(brojCvorova, -1);
  odlazna.resize(brojCvorova, -1);
  vector<bool> posecen(brojCvorova, false);

  dfs(cvor,posecen);

  cout << "Dolazna i odlazna numeracija cvorova:" << endl;
  for (int i = 0; i < brojCvorova; i++)
    cout << "Cvor " << i << ": " << dolazna[i]
         << "/" << odlazna[i] << endl;
}
```

Čvor v nazivamo *pretkom* čvora w u DFS drvetu T sa korenom r ako je v na jedinstvenom putu od r do w u T. Ako je v predak w, onda je čvor w *potomak* čvora v. Na primer, za graf sa slike 6 važi da je čvor b predak čvora j u DFS drvetu, a j potomak čvora b. Ako je algoritam DFS pokrenut za čvor r, DFS pretraga iz čvora v počeće pre pretrage iz čvora w, pa je $v.Pre < w.Pre$. S druge strane, pretraga iz čvora v završava se posle pretrage iz čvora w, pa je $v.Post > w.Post$.

DFS drvo obuhvata sve čvorove povezanog grafa G. Redosled sinova svakog čvora u drvetu određen je listom povezanosti koja zadaje graf G, pa se za svaka dva sina istog čvora može reći koji je od njih levi (prvi po tom redosledu), a koji desni. Relacija levi – desni se prenosi na proizvoljna dva čvora u i v koji nisu u relaciji predak – potomak (videti ilustraciju na slici 7). Za čvorove u i v tada postoji jedinstveni zajednički predak w u DFS drvetu, kao i sinovi u' i v' čvora w takvi da je u' predak u i v' predak v. Kažemo da je u levo od v ako i samo ako je u' levo od v'. Geometrijska interpretacija ove relacije je jasna: DFS drvo ćemo iscrtavati naniže prilikom prelaska u nove – neoznačene čvorove (koraci u dubinu), odnosno sleva udesno prilikom dodavanja novih grana posle povratka u već označene čvorove. Ako je čvor u levo od čvora v, onda je on prilikom pretrage označen pre čvora v, pa je $u.Pre < v.Pre$. Obrnuto ne važi uvek: ako je $u.Pre < v.Pre$, onda je u levo od v, ili je u predak v u DFS drvetu (tj. iznad v). S druge strane, pretraga iz čvora u završava se pre pretrage iz čvora v, pa je $u.Post < v.Post$.

Lema: [Osnovna osobina DFS drveta neusmerenog grafa] Neka je $G = (V, E)$ povezan neusmeren graf, i neka je $T = (V, F)$ DFS drvo grafa G. Svaka grana grafa $e \in E$ pripada T (tj. $e \in F$) ili spaja dva čvora grafa G, od kojih je jedan predak drugog u T.

Dokaz: Neka je (v, u) grana u G, i pretpostavimo da je u toku DFS pretrage



Slika 7: Ilustracija relacije “levo – desno” na skupu čvorova DFS drveta.

čvor v posećen pre čvora u. Posle označavanja čvora v, u petlji se rekurzivno pokreće DFS pretraga iz svakog neoznačenog suseda čvora v. U trenutku kad dođe red na čvor u, ako je u označen, onda je u potomak čvora v u drvetu T, a u protivnom se iz u započinje DFS, pa u postaje sin čvora v u drvetu T i grana (v, u) pripada DFS drvetu T. $\square$

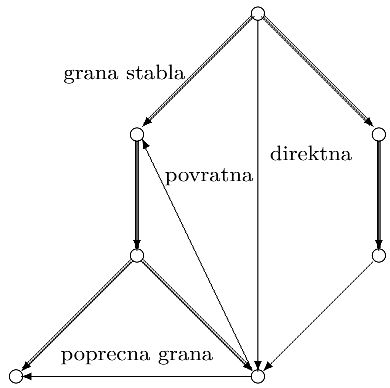
Tvrđenje leme može se preformulisati na sledeći način: grane grafa ne mogu biti *poprečne grane* u odnosu na DFS drvo, odnosno grane koje povezuju čvorove na razdvojenim putevima od korena (tj. takva dva čvora u i v koja su u odnosu levo – desno).

Usmereni grafovi

Procedura pretrage u dubinu usmerenih grafova ista je kao za neusmerene grafove. Međutim, usmerena DFS drveta imaju nešto drugačije osobine. Za njih, na primer, nije tačno da nemaju poprečne grane, što se može videti iz primera na slici 8. U odnosu na DFS drvo grane grafa pripadaju jednoj od četiri kategorije: *grane DFS drveta*, *povratne*, *direktne* i *poprečne* grane. Prve tri vrste grana povezuju dva čvora od kojih je jedan potomak drugog u DFS drvetu: grana DFS drveta povezuje oca sa sinom, povratna grana potomka sa pretkom, a direktna grana pretka sa potomkom (slika 8). Jedino poprečne grane povezuju čvorove koji nisu “srodnici” u drvetu. Poprečne grane, međutim, moraju biti usmerene “zdesna ulevo”, kao što pokazuje sledeća lema.

Lema: [Osnovna osobina DFS drveta usmerenog grafa] Neka je $G = (V, E)$ usmereni graf, i neka je $T = (V, F)$ DFS drvo grafa G. Ako je $(v, w) \in E$ grana grafa G za koju važi $v.Pre < w.Pre$, onda je čvor w potomak čvora v u drvetu T.

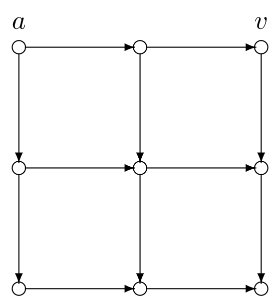
Dokaz: Pošto prema dolaznoj DFS numeraciji čvor v prethodi čvoru w, čvor w je označen posle čvora v. Grana grafa (v, w) mora biti razmatrana u toku rekurzivnog poziva algoritma DFS iz čvora v. Ako u tom trenutku čvor w nije označen, onda se grana (v, w) mora uključiti u DFS drvo, tj. $(v, w) \in F$,



Slika 8: DFS drvo usmerenog grafa. Klasifikacija grana u odnosu na DFS drvo.

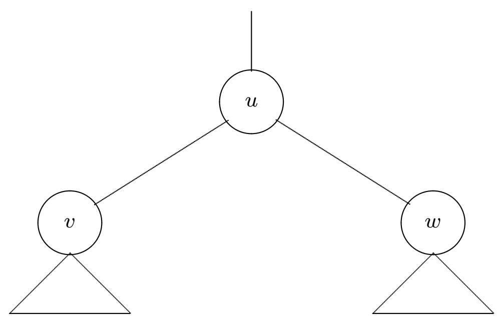
pa je tvrđenje leme tačno. U protivnom, čvor w je označen u toku izvođenja rekurzivnog poziva DFS iz v, pa je w potomak čvora v u DFS drvetu T. $\square$

Algoritam DFS za povezan neusmereni graf, započet iz proizvoljnog čvora, obilazi ceo graf. Analogno tvrđenje ne mora biti tačno za usmerene grafove. Preciznije, ovo tvrđenje će važiti samo ako je graf jako povezan. Posmatrajmo usmereni graf na slici 9. Ako se DFS pretraga započne iz čvora v, onda će biti dostignuti samo čvorovi u desnoj koloni. DFS pretraga može da dostigne sve čvorove grafa samo ako se započne iz čvora a. Ako se čvor a ukloni iz grafa zajedno sa dve grane koje izlaze iz njega, onda u datom grafu ne postoji čvor iz koga DFS algoritam obilazi ceo graf. Prema tome, uvek kad govorimo o DFS obilasku usmerenog grafa, smatraćemo da je DFS algoritam pokrenut onoliko puta koliko je potrebno da bi svi čvorovi bili označeni i sve grane bile razmotrene. Dakle, u opštem slučaju usmereni graf umesto DFS drveta ima *DFS šumu*.



Slika 9: Primer kad pretraga u dubinu usmerenog grafa ako se pokrene iz čvora v ne obilazi sve čvorove grafa.

Nepostojanje poprečnih grana u odnosu na DFS drvo koje idu sleva udesno govori nešto korisno o odlaznoj numeraciji čvorova grafa i o četiri vrste grana u odnosu na DFS drvo. Na slici 10(a) prikazana su tri čvora grafa u, v i w u okviru DFS drveta grafa. Čvorovi v i w su sinovi čvora u, a čvor w je desno od čvora v. Na slici 10(b) prikazani su vremenski intervali trajanja rekurzivnih poziva algoritma DFS za svaki od ovih čvorova. Zapažamo da je DFS algoritam pokrenut iz čvora v, potomka čvora u, aktivan samo u podintervalu vremena za koje je aktivan DFS algoritam iz čvora u (pretka čvora v). Specijalno, DFS pretraga iz v čvora završava se pre završetka DFS pretrage iz čvora u. Prema tome, iz činjenice da je v potomak čvora u sledi da je $v.Post < u.Post$. Pored toga, ako je čvor w desno od čvora v, onda poziv algoritma DFS iz čvora w ne može biti pokrenut pre nego što se završi DFS poziv iz čvora v. Prema tome, ako je v levo od w, onda je $v.Post < w.Post$. Iako to nije pokazano na slici 10, isti zaključak je tačan i ako su v i w u različitim drvetima DFS šume, pri čemu je drvo čvora v levo od drveta čvora w.



(a) Tri čvora u DFS stablu

(b) Aktivni intervali za njihove pozive DFS

Slika 10: Odnos između položaja čvorova u DFS drvetu i trajanja rekurzivnih poziva pokrenutih iz ovih čvorova.

Razmotrimo sada za proizvoljnu granu (u, v) grafa G odnos odlazne DFS numeracije čvorova u i v.

1. Ako je (u, v) grana drveta ili direktna grana, onda je čvor v potomak čvora u, pa važi $v.Post < u.Post$.

2. Ako je (u, v) poprečna grana, onda zbog toga što je čvor v levo od čvora u, ponovo važi $v.Post < u.Post$.

3. Ako je (u, v) povratna grana i $v \neq u$, onda je v pravi predak čvora u i $v.Post > u.Post$. Međutim, pošto je specijalni i petlja jedna vrsta povratne grane, za povratnu granu može da važi i $v = u$, pa samim tim i

$v.Post = u.Post$. Prema tome, u opštem slučaju za povratnu granu (u, v) važi $v.Post \geq u.Post$.

Prema tome, dokazano je naredno tvrđenje.

Lema: Grana (u, v) usmerenog grafa $G = (V, E)$ je povratna u odnosu na DFS drvo grafa ako i samo ako prema odlaznoj numeraciji čvor u prethodi čvoru v, odnosno $u.Post \leq v.Post$.

Na osnovu vrednosti dolazne i odlazne numeracije čvorova u grafu može se izvršiti klasifikacija grana u grafu u odnosu na DFS drvo. Naime za usmerenu granu $(u, v) \in E$ važi:

- ako je $u.Post \leq v.Post$, onda je grana (u, v) povratna,
- ako je $u.Post > v.Post$ i $u.Pre > v.Pre$, onda je grana (u, v) poprečna,
- ako je $u.Post > v.Post$ i $u.Pre < v.Pre$, onda ako je čvor u roditelj čvora v u DFS drvetu grana (u, v) je grana DFS drveta, a inače je (u, v) direktna grana.

Razmotrimo algoritam kojim se za svaku granu usmerenog grafa određuje njen tip u odnosu na DFS drvo, na osnovu vrednosti dolazne i odlazne numeracije čvorova.

```
vector<vector<int>> listaSuseda {{1, 2, 4}, {3, 4}, {5}, {},
                                 {0, 6, 7}, {8,3}, {}, {1}, {}};

int vreme_dolazna = 1;
int vreme_odlazna = 1;
vector<int> dolazna;
vector<int> odlazna;
vector<int> roditelj;

void dfs(int cvor) {
  // cvor dobija narednu mogucu vrednost dolazne numeracije
  dolazna[cvor] = vreme_dolazna;
  vreme_dolazna++;
  // rekurzivno prolazimo kroz sve susede koje do sada nismo obisli
  for (auto sused : listaSuseda[cvor]) {
    if (dolazna[sused] == -1) { // !posecen[sused]
      // 'cvor' je roditelj cvora 'sused' u DFS drvetu
      roditelj[sused] = cvor;
      dfs(sused);
    }
  }
  // cvor dobija narednu mogucu vrednost odlazne numeracije
  odlazna[cvor] = vreme_odlazna;
  vreme_odlazna++;
}
```

```
void klasifikuj_grane(int cvor) {
  int brojCvorova = listaSuseda.size();
  dolazna.resize(brojCvorova, -1);
  roditelj.resize(brojCvorova, -1);
  odlazna.resize(brojCvorova, -1);

  dfs(cvor);

  cout << "Dolazna i odlazna numeracija cvorova:" << endl;
  for (int i = 0; i < brojCvorova; i++)
    cout << "Cvor " << i << ": " << dolazna[i]
         << "/" << odlazna[i] << endl;

  cout << "Tipovi grana datog usmerenog grafa: " << endl;
  for (int i = 0; i < listaSuseda.size(); i++)
    for (int j = 0; j < listaSuseda[i].size(); j++){
      int k = listaSuseda[i][j];

      if (i == roditelj[k])
         cout << i << "-" << k << " je grana DFS drveta" << endl;
      else if (odlazna[i] <= odlazna[k])
         cout << i << "-" << k << " je povratna grana" << endl;
      else // odlazna[i] > odlazna[j]
         if (dolazna[i] < dolazna[k])
            cout << i << "-" << k << " je direktna grana" << endl;
         else
            cout << i << "-" << k << " je poprecna grana" << endl;
    }
}

int main(){
  klasifikuj_grane(0);
  return 0;
}
```

Pokazaćemo sada kako se algoritam DFS pretrage može iskoristiti za utvrđivanje da li je zadati graf aciklički.

Problem: Za zadati usmereni graf $G = (V, E)$ ustanoviti da li sadrži usmereni ciklus.

Lema: Neka je $G = (V, E)$ usmereni graf, i neka je T DFS drvo grafa G. Tada graf G sadrži usmereni ciklus ako i samo ako graf G sadrži povratnu granu u odnosu na DFS drvo T.

Dokaz: Pretpostavimo da graf G sadrži povratnu granu (u, v). Ona zajedno sa granama DFS drveta na putu od v do u čini ciklus, te važi jedan smer datog tvrđenja. Pokažimo da važi i suprotno tvrđenje, odnosno da ako u grafu postoji ciklus, da je nužno jedna od njegovih grana povratna. Zaista, pretpostavimo da u grafu G postoji ciklus koji čine grane $(v_1, v_2), (v_2, v_3), \ldots, (v_{k-1}, v_k), (v_k, v_1)$, od kojih ni jedna nije povratna u odnosu na DFS drvo T. Ako je $k = 1$, odnosno ciklus je petlja, onda je sama grana (v_1, v_1) povratna. Ako je pak $k > 1$, pretpostavimo da ni jedna od grana $(v_1, v_2), (v_2, v_3), \ldots, (v_{k-1}, v_k)$ nije povratna. Prema prethodnoj lemi važe nejednakosti $v_1.Post > v_2.Post > \cdots > v_k.Post$, iz kojih sledi da je $v_k.Post < v_1.Post$, pa je grana (v_k, v_1) prema prethodnoj karakterizaciji povratna — suprotno pretpostavci. Time je dokazano da u svakom ciklusu postoji povratna grana u odnosu na DFS drvo. □

Dakle, na osnovu prethodne leme, algoritam za proveru da li graf sadrži ciklus svodi se na određivanje odlazne DFS numeracije čvorova datog grafa i proveru postojanja povratne grane.

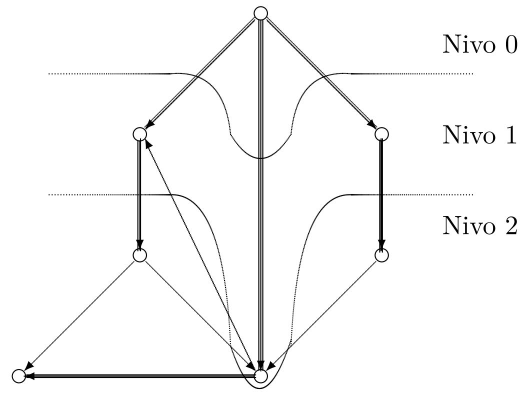
Pretraga u širinu

Pretraga u širinu (ili BFS, što je skraćenica od breadth–first–search) je obilazak grafa na sistematičan način, nivo po nivo, pri čemu se usput formira *drvo pretrage u širinu* (BFS drvo). Pretpostavimo da je graf zadat listom povezanosti. Ako BFS pretragu pokrenemo iz čvora v (v je koren BFS drveta), onda se najpre posećuju svi susedi čvora v redosledom određenim redosledom u listi povezanosti grafa (sinovi čvora v u BFS drvetu (čvorovi nivoa 1). Zatim se dolazi do svih njihovih neposećenih suseda, odnosno do "unuka" polaznog čvora (čvorovi nivoa 2), i tako dalje (videti primer na slici 11). Prilikom obilaska čvorovi se mogu numerisati *BFS brojevima*, slično kao pri pretrazi u dubinu. Preciznije, čvor w ima BFS broj k ako je on k-ti po redu čvor označen u toku obilaska algoritmom BFS. BFS drvo grafa može se formirati uključivanjem samo grana ka novooznačenim čvorovima: lako se pokazuje da dobijeni podgraf jeste povezan i da nema ciklus jer od svih grana koje vode nekom čvoru uključujemo tačno jednu, dok čvor nije posećen. Zapaža se da izlazna obrada kod pretrage u širinu, za razliku od pretrage u dubinu, nema smisla; pretraga nema povratak "naviše", već se, polazeći od korena, kreće samo naniže.

Kako realizovati pretragu u širinu? Čvorove obilazimo u željenom redosledu korišćenjem pogodne strukture podataka: sve neposećene susede tekućeg čvora smeštamo u datu kolekciju i potrebno ih je na isti način obraditi u redosledu dodavanja u tu kolekciju. U ove svrhe možemo iskoristiti strukturu podataka red.

Prikažimo na koji način se može realizovati pretraga u širinu ako je graf zadat listom povezanosti. Prilikom pretrage štampamo čvorove grafa u redosledu određenim BFS numeracijom čvorova i konstruišemo BFS drvo.

```
vector<vector<int>> listaSuseda {{1, 2}, {2, 3, 4}, {5},
                        {0, 4}, {6, 7}, {1, 8}, {}, {6}, {2}};

void bfs(int cvor) {
```



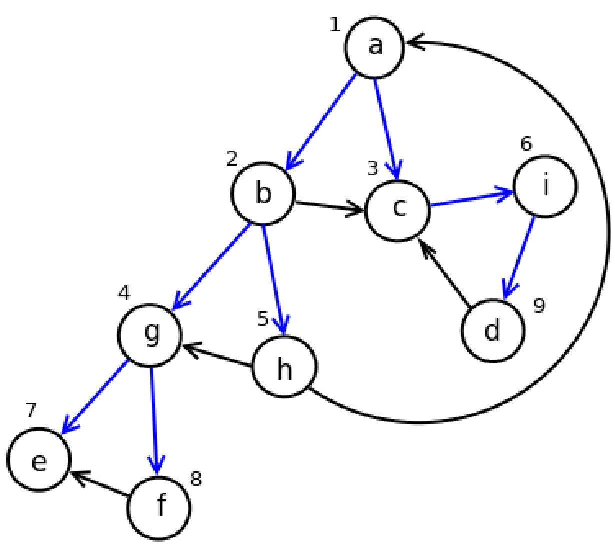
Slika 11: BFS drvo usmerenog grafa.

```
int brojCvorova = listaSuseda.size();
vector<bool> posecen(brojCvorova, false);

vector<vector<int>> bfs_drvo(brojCvorova);

// red u kom cuvamo cvorove u redosledu kojim ih treba posetiti
queue<int> red;
red.push(cvor);
posecen[cvor] = true;
// stampamo prvi cvor u redosledu BFS numeracije
cout << cvor << endl;
while (!red.empty()) {
  int cvor = red.front();
  red.pop();
  for (int sused : listaSuseda[cvor]) {
    // neposecene susede tekuceg cvora dodajemo u red
    if (!posecen[sused]) {
      posecen[sused] = true;
      // stampamo naredni cvor u redosledu BFS numeracije
      cout << sused << endl;
      // dodajemo granu u bfs drvo (pretpostavljamo da je graf usmeren)
      bfs_drvo[cvor].push_back(sused);
      red.push(sused);
    }
  }
}
cout << "Grane BFS drveta su: ";
for (int i = 0; i < bfs_drvo.size(); i++)
  for (int j = 0; j < bfs_drvo[i].size(); j++)
    cout << "(" << i << "," << bfs_drvo[i][j] << ")" << endl;
```

```
}

int main() {
  bfs(0);
  return 0;
}
```

Lako je uveriti se da se prilikom BFS obilaska grafa svaki čvor obrađuje po jednom i da se svaka grana pregleda po jednom u slučaju usmerenog grafa, odnosno dva puta ako je graf neusmeren. Stoga je vremenska složenost algoritma BFS $O(|V|+|E|)$. Slično kao i kod pretrage u dubinu, složenost algoritma bi bila veća da se koristi reprezentacija grafa matricom povezanosti jer je prolaz kroz sve susede nekog čvora složenosti $\Theta(n^2)$.

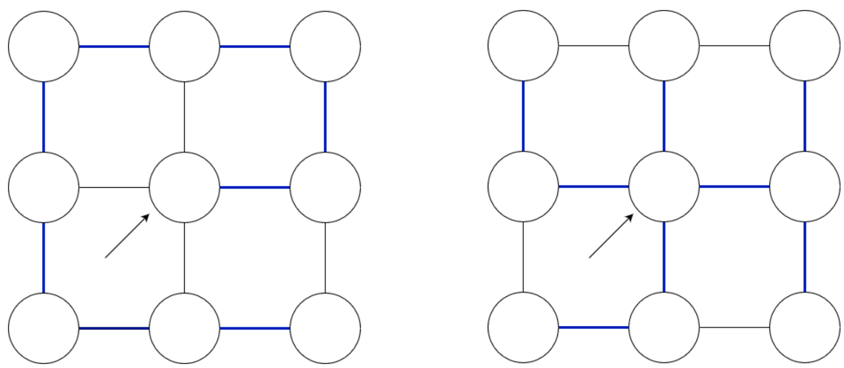
Razmotrimo primer izvršavanja algoritma pretrage u širinu pokrenutog iz čvora a na primeru grafa sa slike 12. Neka redosled suseda čvorova u listi povezanosti kojom je predstavljen graf odgovara leksikografskom poretku oznaka čvorova. Najpre posećujemo čvor a, a nakon njega njegove neposećene susede: čvor b, a zatim čvor c. Nakon toga posećujemo neposećene susede čvora b, odnosno čvorove g i h redom, a zatim neposećene čvorove suseda c: čvor i. Nakon toga posećujemo čvorove e i f kao susede čvora g, čvor h nema neposećenih suseda, a zatim posećujemo čvor d kao jedinog neposećenog suseda čvora i. Nakon ovoga redom razmatramo čvorove e, f i d i zaključujemo da nijedan od njih nema neposećenih čvorova i pretraga se završava. Grane BFS drveta su na slici 12 prikazane plavom bojom, a uz svaki čvor prikazan je njegov broj u BFS numeraciji.



Slika 12: Primer usmerenog grafa na kome ilustrujemo pretragu u širinu.

tekući čvor	sadržaj reda
-	a
a	b, c
b	c, g, h
c	g, h, i
g	h, i, e, f
h	i, e, f
i	e, f, d
e	f, d
f	d
d	-

Tabela 1: Ilustracija izvršavanja algoritma pretrage u širinu na primeru grafa sa slike 12.



Slika 13: Razlika između DFS i BFS drveta za pretragu pokrenutu iz središnjeg čvora. Plavom bojom označene su grane koje pripadaju DFS i BFS drvetu redom.

Razmotrimo nekoliko tvrđenja koja važe za pretragu u širinu, odnosno za BFS drvo.

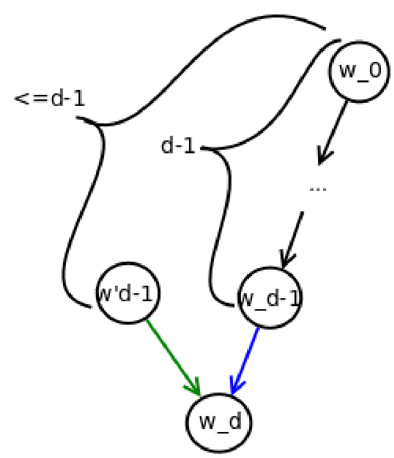
Lema: Ako grana (u, v) pripada BFS drvetu i čvor u je otac čvora v, onda čvor u ima najmanji BFS broj među čvorovima iz kojih postoji grana ka v.

Dokaz: Ako bi u grafu postojala grana (w, v), takva da čvor w ima manji BFS broj od čvora u, onda bi u trenutku obrade čvora w čvor v morao biti upisan u red, pa bi grana (w, v) morala biti uključena u BFS drvo. Međutim, prema pretpostavci grana (u, v) je uključen u BFS drvo, pa ne može istovremeno biti i grana (w, v) te dobijamo kontradikciju. $\square$

Definišimo *rastojanje* $d(u, v)$ između čvorova u i v kao dužinu najkraćeg puta od čvora u do čvora v, s tim da se pod dužinom puta podrazumeva broj grana koje čine taj put. Ovaj pojam odgovara nivou čvora u odnosu na BFS drvo.

Lema: Put od korena r BFS drveta do proizvoljnog čvora w kroz BFS drvo najkraći je put od čvora r do čvora w u grafu G.

Dokaz: Indukcijom po d dokazaćemo da do svakog čvora w na rastojanju d od korena r (jedinstveni) put kroz drvo od r do w ima dužinu d. Za $d = 1$ tvrđenje je tačno: grana (r, w) je nužno deo BFS drveta, pa između r i w postoji put kroz BFS drvo dužine 1. Pretpostavimo da je tvrđenje tačno za sve čvorove koji su na rastojanju manjem od d od korena, i neka je w neki čvor na rastojanju d od korena; drugim rečima, postoji niz čvorova $w_0 = r$, w_1, $w_2, \ldots, w_{d-1}, w_d = w$ koji čine put dužine d od r do w, i ne postoji kraći put od r do w. Pošto je dužina najkraćeg puta od r do w_{d-1} jednaka $d - 1$ prema induktivnoj hipotezi put od r do w_{d-1} kroz drvo ima dužinu $d - 1$. U trenutku obrade čvora w_{d-1}, ako čvor w_d nije označen, pošto u grafu G postoji grana (w_{d-1}, w_d), ta grana se uključuje u BFS drvo, pa do čvora w_d postoji put dužine d kroz drvo. U protivnom, ako je u tom trenutku čvor w_d već označen, onda do čvora w_d kroz BFS drvo vodi grana iz nekog čvora w'_{d-1}, označenog pre w_{d-1}, iz čega sledi da je nivo čvora w'_{d-1} najviše $d - 1$ i do njega po induktivnoj hipotezi postoji put dužine najviše $d - 1$ kroz BFS drvo, te i do čvora w vodi put kroz BFS drvo dužine najviše d. $\square$



Slika 14: Ilustracija uz dokaz leme.

Lema: Ako je graf $G = (V, E)$ neusmeren i $(v, w) \in E$ neka njegova proizvoljna grana, onda ta grana spaja dva čvora čiji se nivoi razlikuju najviše za jedan.

Dokaz: Pretpostavimo da je od čvorova v i w, čvor v prvi dostignut BFS pretragom i neka je njegov nivo d. Tada je nivo čvora w veći ili jednak od d. S druge strane, nivo čvora w nije veći od $d + 1$, jer do njega vodi grana BFS drveta ili iz čvora v, ili iz nekog čvora koji je označen pre v. Dakle, nivo čvora w je ili d ili $d + 1$, te tvrđenje leme važi. $\square$