Komponente jake povezanosti grafa

Za usmereni graf kažemo da je *jako povezan* ako je svaki čvor grafa dostižan iz svakog drugog čvora u grafu.

Da se podsetimo:

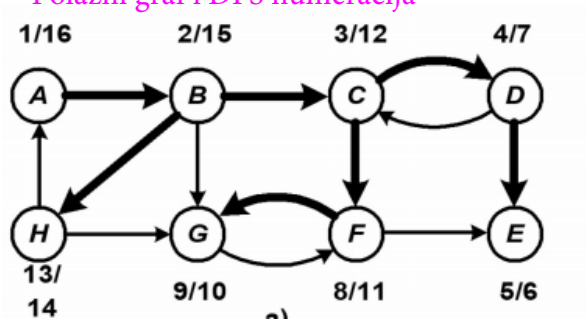
Određivanje povezanih komponenata
u usmerenom grafu

- ✓ DFS, zapamte se završna vremena
- ✓ napravi se transponovani graf *Gt* =(*V*, *Et*)
- ✓ DFS za *Gt* polazeći od čvora sa najvećim završnim vremenom
- ✓ svaki DFS_VISIT daje jednu jako povezanu komponentu

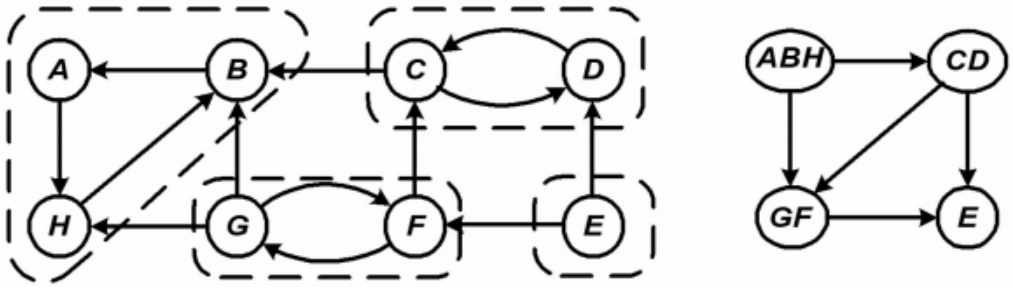
Izlaz

- ✓ redukovani graf
- ✓ međukomponentne grane

Polazni graf i DFS numeracija



DFS transponovanog grafa i kompresovanje jakih komponenti grafa



Na skupu čvorova usmerenog grafa $G = (V, E)$ može se definisati relacija $\sim$ *obostrane dostižnosti*: $u \sim v$ ako je čvor u dostižan iz čvora v i čvor v dostižan iz čvora u. Za ovu relaciju važi da je:

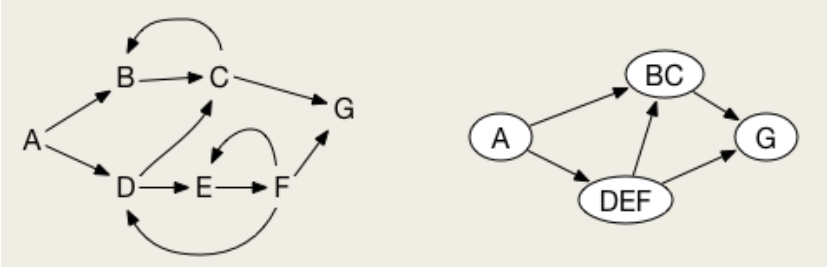
refleksivna – za svaki čvor $u \in V$ je $u \sim u$,

simetrična – za svaka dva čvora $u, v \in V$ važi $u \sim v$ akko $v \sim u$,

tranzitivna – za svaka tri čvora $u, v, w \in V$ iz $u \sim v$ i $v \sim w$ sledi i $u \sim w$.

Stoga je ona relacija ekvivalencije. Ona razlaže skup čvorova V u klase ekvivalencije koje nazivamo *komponente jake povezanosti* grafa G (eng. strongly connected components). Na slici 5 prikazan je usmereni graf koji ima četiri komponente jake povezanosti koje se sastoje redom od čvorova $\{A\}, \{B, C\}, \{D, E, F\}, \{G\}$. Primetimo da svi čvorovi nekog ciklusa pripadaju istoj komponenti jake povezanosti. Graf G se može "kompresovati" i razmatrati kao usmereni aciklički graf koji se sastoji od svojih komponenti jake povezanosti (slika 5, desno): svaki čvor u ovom grafu odgovara jednoj komponenti jake

povezanosti, a dva čvora su povezana granom samo ako i samo ako u polaznom grafu postoji grana između nekog čvora jedne komponente i nekog čvora druge komponente. Jasno je da ovaj graf mora biti aciklički jer ako bi u njemu postojao ciklus to bi značilo da se sve komponente koje pripadaju ciklusu mogu spojiti u jednu veću komponentu povezanosti.



Slika 5: Graf G i usmereni aciklički graf koji se sastoji od komponenti jake povezanosti grafa G.

Direktan način da bi se odredile komponente jake povezanosti sastojao bi se u tome da se za prvi čvor v_0 odredi koji čvorovi pripadaju njegovoj jakoj komponenti povezanosti, tako što bi se za sve preostale čvorove proveravalo da li su obostrano dostižni iz v_0 – to bi moglo da se uradi DFS pretragom iz čvora v_0 i iz svakog novog čvora čija se pripadnost komponenti ispituje. Nakon toga bi se sličan proces ponavljao za naredni čvor koji ne pripada jakoj komponenti kojoj pripada čvor v_0. Jasno je da bi ovo bilo veoma neefikasno.

Postoji nekoliko različitih algoritama linearne vremenske složenosti za određivanje komponenti jake povezanosti u grafu, a najpoznatiji među njima su Tardžanov algoritam i Kosaradžuov algoritam. Oba algoritma su zasnovana na DFS obilasku grafa, samo se kod prvog sve radi u jednom prolazu, dok se u drugom dva puta poziva algoritam DFS pretrage. U nastavku ćemo razmotriti prvi od ova dva algoritma.

Tardžanov algoritam

Prilikom DFS obilaska datog usmerenog grafa G implicitno se formira DFS drvo, odnosno šuma. Bez narušavanja opštosti možemo pretpostaviti da je graf takav da postoji čvor iz kog se on može u potpunosti obići, odnosno da ima DFS drvo. Nazovimo *baznim čvorom* neke jake komponente onaj čvor te komponente koji ima najmanju vrednost dolazne numeracije.

Lema: Neka je b bazni čvor jake komponente X. Tada za svako $v \in X$ važi da je v potomak čvora b u odnosu na DFS drvo i svi čvorovi na putu od b do v pripadaju komponenti X.

Dokaz: Dokažimo najpre prvo tvrđenje. Neka je v proizvoljni čvor iz X različit

od b. Važi da je (a) v potomak čvora b, (b) b potomak čvora v ili (c) nijedno od ova dva. Slučaj (b) nije moguć jer ako bi čvor b bio potomak čvora v onda bi on imao veću vrednost dolazne numeracije od v što je u suprotnosti sa pretpostavkom leme.

Pretpostavimo da važi (c). Put od čvora b do čvora v mora da postoji jer ova dva čvora pripadaju istoj jakoj komponenti. Razmotrimo jedan takav put p i neka je r najbliži zajednički predak svih čvorova na tom putu. Čvor r mora pripadati tom putu p, a čvorovi b i v moraju biti potomci različite dece čvora r. Označimo sa T_b poddrvo sa korenom u čvoru b, a sa T_v poddrvo sa korenom u čvoru v. S obzirom na to da je vrednost dolazne numeracije čvora b manja od vrednosti čvora v i s obzirom na to da su T_v i T_b disjunktna drveta, ne može postojati grana ni od jednog čvora iz T_b ka nekom čvoru iz T_v. Stoga je jedini put od b do v kroz čvor r. Međutim, s obzirom na to da je r predak čvora b on ima manju vrednost dolazne numeracije od čvora b a pripada istoj jakoj komponenti jer postoji put od b do r, a i od r do b kroz grane drveta. Ovo je u suprotonosti sa pretpostavkom da je b bazni čvor te komponente, pa slučaj (c) nije moguć. Stoga važi da je čvor v potomak čvora b.

Dokaz drugog dela leme je jednostavan. Neka je x čvor na putu od v do b. Postoji put od b do x kroz grane DFS drveta, a takođe i put od x do b tako što prvo idemo od x do v, pa od v do b. Stoga je x u istoj jakoj komponenti kao i čvor b.

Lema: Neka je b bazni čvor i neka su $b_1, b_2, \ldots b_k$ bazni čvorovi koji su potomci čvora b. Tada važi da je jaka komponenta kojoj pripada čvor b skup svih potomaka čvora b koji nisu potomci nijednog drugog čvora $b_1, b_2, \ldots, b_k$.

Pretpostavimo suprotno, odnosno da postoji čvor v koji je u istoj jakoj komponenti kao i b i koji je potomak i čvora b i čvora b_i za neko i, $1 \leq i \leq k$. Mora da postoji put od v do b, a takođe i put od čvora b preko čvora b_i do čvora v (koji se sastoji od grana DFS drveta). Odavde sledi da su b i b_i u istoj jakoj komponenti što je u suprotnosti sa pretpostavkom.

Lema: Čvor v je bazni čvor akko važi $v.Pre = v.minPre$.

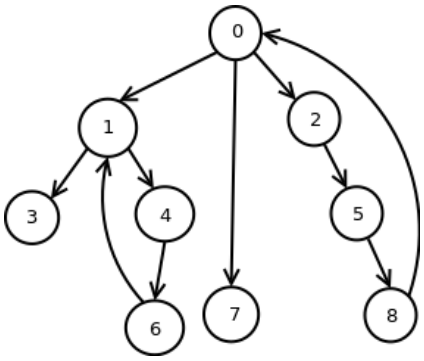
Da bismo izdvojili čvorove koji pripadaju poddrvetu sa korenom u datom baznom čvoru, možemo iskoristiti stek na koji ćemo stavljati čvor prilikom prve posete tokom DFS obilaska grafa. Kada tokom obilaska naiđemo na čvor koji se već nalazi na steku, znamo da će jednoj komponenti povezanosti pripadati svi čvorovi koji se nalaze na steku počev od tog čvora. Poprečne grane neće biti razmatrane jer kada stignemo do čvora koji je već posećen, vršimo obradu samo ako se on nalazi na steku (što neće biti slučaj sa krajnjim čvorom poprečne grane).

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

void dfs(int cvor, vector<int> &dolazna, vector<int> &min_pretka,
```

```
      stack<int> &redosledUObilasku, vector<int> &u_steku){
  dolazna[cvor] = min_pretka[cvor] = vreme;
  vreme++;
  redosledUObilasku.push(cvor);
  u_steku[cvor] = true;

  // rekurzivno prolazimo kroz sve susede koje nismo obisli
  for (auto sused : listaSuseda[cvor]){
    if (dolazna[sused] == -1){
       dfs(sused,dolazna,min_pretka,redosledUObilasku,u_steku);

       if (min_pretka[sused] < min_pretka[cvor])
          min_pretka[cvor] = min_pretka[sused];
    }
    // azuriramo vrednost minPre za cvor samo ako se sused nalazi u steku
    else if (u_steku[sused])
      if (dolazna[sused] < min_pretka[cvor])
         min_pretka[cvor] = dolazna[sused];
  }

  // ako je u pitanju koren komponente, stampamo sve cvorove te komponente
  if (dolazna[cvor] == min_pretka[cvor]){
    while(1){
      int cvor_komponente = redosledUObilasku.top();
      cout << cvor_komponente << " ";
      u_steku[cvor_komponente] = false;
      redosledUObilasku.pop();
      if (cvor_komponente == cvor){
         cout << "\n";
     break;
      }
    }
  }
}

void ispisi_komponente(int cvor){
  int brojCvorova = listaSuseda.size();
  vector<int> dolazna(brojCvorova,-1);
  vector<int> min_pretka(brojCvorova);
  stack<int> redosledUObilasku;
  vector<int> u_steku(brojCvorova,false);

  cout << "Komponente jake povezanosti su: " << endl;
  dfs(cvor,dolazna,min_pretka,redosledUObilasku,u_steku);
}
```

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

Razmotrimo izvršavanje algoritma na primeru grafa prikazanog na slici 6.



Slika 6: Primer grafa čije su komponente jake povezanosti $\{3\}$, $\{8\}$, $\{1,4,6\}$, $\{0,2,5,8\}$.

```
stek: 0

stek: 0,1

stek: 0,1,3
zavrsava se rekurzivni poziv iz cvora 3 i
s obzirom na to da za cvor 3 vazi uslov v.Pre=v.minPre=3
sa steka skidamo samo cvor 3 i on predstavlja zasebnu komponentu

stek:0,1

stek: 0,1,4

stek: 0,1,4,6
zavrsava se rekurzivni poziv iz cvora 6, ali za njega je v.Pre=5, a
v.minPre=2 pa on nije bazni cvor komponente

stek: 0,1,4,6,7
zavrsava se rekurzivni poziv iz cvora 7 i
s obzirom na to da za cvor 7 vazi uslov v.Pre=v.minPre=6
sa steka skidamo samo cvor 7 i on predstavlja zasebnu komponentu

stek: 0,1,4,6
zavrsava se rekurzivni poziv iz cvora 4, ali za njega je v.Pre=5, a
```

```
v.minPre=2 pa on nije bazni cvor komponente

zavrsava se rekurzivni poziv iz cvora 1 i
s obzirom na to da za cvor 1 vazi uslov v.Pre=v.minPre=2
sa steka skidamo sve cvorove do cvora 1, dakle cvorove 6,4,1
i oni predstavljaju zasebnu komponentu

stek: 0,2

stek: 0,2,5

stek: 0,2,5,8
zavrsava se rekurzivni poziv iz cvora 8, ali za njega je v.Pre=9, a
v.minPre=1 pa on nije bazni cvor komponente

zavrsava se rekurzivni poziv iz cvora 5, ali za njega je v.Pre=8, a
v.minPre=1 pa on nije bazni cvor komponente

zavrsava se rekurzivni poziv iz cvora 2, ali za njega je v.Pre=7, a
v.minPre=1 pa on nije bazni cvor komponente

zavrsava se rekurzivni poziv iz cvora 0 i
s obzirom na to da za cvor 0 vazi uslov v.Pre=v.minPre=1
sa steka skidamo sve cvorove do cvora 0, dakle cvorove 8,5,2,0
i oni predstavljaju zasebnu komponentu
```

Ovaj algoritam je zasnovan na DFS obilasku grafa i složenosti je $O(|V| + |E|)$.