ALGORITMI ZA RAD SA GRAFOVIMA

UVOD

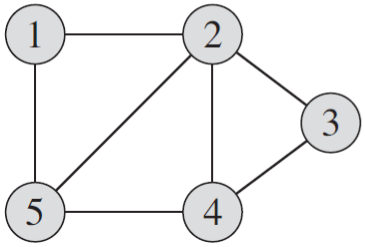
- Grafovi i grafovski algoritmi su stariji od računara
- Koriste za modelovanje kompleksnih odnosa između objekata
- Koriste se za modelovanje različitih situacija u različitim oblastima od arheologije do sociologije
- Šta se sve modeluje pomoću grafova:
  - Nalaženje dobrog puta do nekog odredišta u gradu
  - Proces izvršavanja programa kroz stanja koja prolazi, problem je odrediti koja stanja izvršavanja programa mogu dovesti do nepoželjnih stanja
  - Pravljenje rasporeda časova na fakultetu
  - Odnos u nekoj grupi u kojoj postoji više načina povezivanja (društvene mreže)

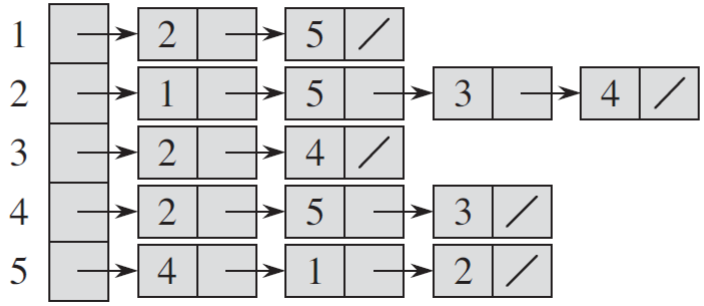
PREDSTAVLJANJE GRAFOVA …

- Graf se označava kao $\boldsymbol{G} = (\boldsymbol{V}, \boldsymbol{E})$ i sastoji se od skupa $\boldsymbol{V}$ čvorova i skupa $\boldsymbol{E}$ grana
- Grana odgovara vezi koja povezuje dva čvora ili relaciji između čvorova
  - Npr. graf može da predstavlja skup ljudi, a da grana povezuje dva čoveka ako se oni poznaju
- Graf je usmeren ako su mu grane uređeni parovi
  - Ako se usmereni graf predstavi crtežom, onda se dodaje strelica usmerene od prvog ka drugom čvoru u paru
  - Ako je potrebno predstaviti hijerarhiju, onda se grane mogu orijentisati od „korena“ (kao u korenskom stablu)
- Graf je neusmeren ako su mu grane neuređeni parovi

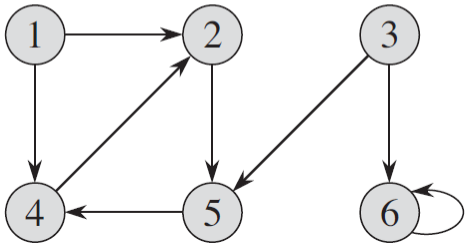
… PREDSTAVLJANJE GRAFOVA …

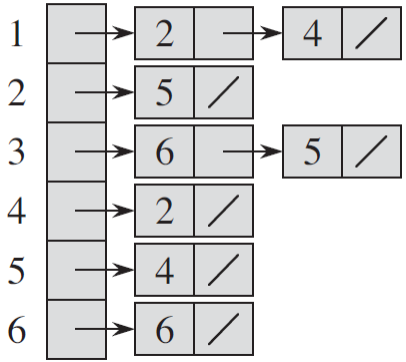
- Postoje 2 standardna načina predstavljanja grafova (i usmerenih i neusmerenih grafova) u računaru:
  - lista povezanosti
  - matrica povezanosti





	1	2	3	4	5
1	0	1	0	0	1
2	1	0	1	1	1
3	0	1	0	1	0
4	0	1	1	0	1
5	1	1	0	1	0





	1	2	3	4	5	6
1	0	1	0	1	0	0
2	0	0	0	0	1	0
3	0	0	0	0	1	1
4	0	1	0	0	0	0
5	0	0	0	1	0	0
6	0	0	0	0	0	1

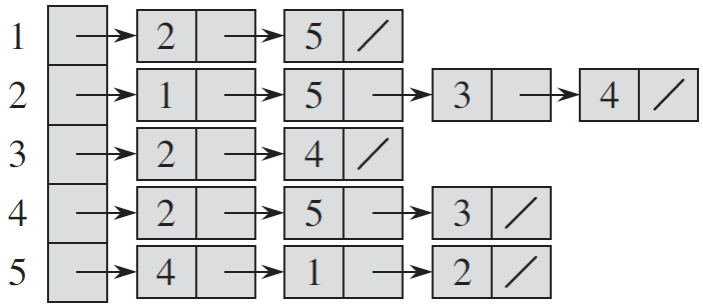
… PREDSTAVLJANJE GRAFOVA …

- Matrica povezanosti
  - Neka je $|V| = n$ i neka je $V = \{v_1, v_2, \ldots, v_n\}$
  - Matrica povezanosti grafa G je kvadratna matrica $A = (a_{ij})$ reda n
  - Elemenat a_{ij} ima vrednost jedan, $a_{ij} = 1$, ako i samo ako je $(v_i, v_j) \in E$, ostali elementi matrice imaju vrednost nula
  - Ako je graf neusmeren, matrica A je simetrična
  - Vrsta i ove matrice je vektor dužine n čija je j-ta koordinata jednaka 1 ako iz čvora v_i vodi grana u čvor v_j, 0 u suprotnom
  - Nedostatak matrice povezanosti je što ona uvek zauzima n^2 prostora, nezavisno od toga koliko grana ima graf
  - Ako je broj grana u grafu mali, većina elemenata matrice povezanosti će imati vrednost nula

	1	2	3	4	5
1	0	1	0	0	1
2	1	0	1	1	1
3	0	1	0	1	0
4	0	1	1	0	1
5	1	1	0	1	0

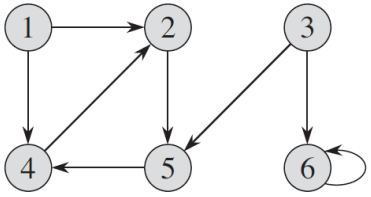
… PREDSTAVLJANJE GRAFOVA …

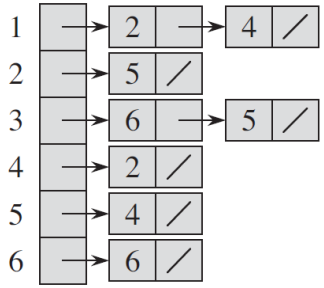
- Lista povezanosti
  - Umesto da se i sve nepostojeće grane prikažu u matrici povezanosti, mogu se formirati povezane liste od jedinica iz $\boldsymbol{i}$-te vrste, $\boldsymbol{i = 1, 2, \ldots, n}$
  - Svakom čvoru pridružuje se lista koja sadrži sve grane susedne čvoru (odnosno grane ka susednim čvorovima)
  - Može biti uređena prema rednim brojevima čvorova na krajevima njenih grana
  - Graf je predstavljen vektorom lista
  - Svaki element vektora sadrži ime (indeks) čvora i pokazivač na njegovu listu čvorova
  - Ako je graf statički, onda nisu dozvoljena umetanja i brisanja i graf se može predstaviti vektorima



… PREDSTAVLJANJE GRAFOVA

- Sa matricama povezanosti je lakše raditi
- Liste povezanosti su mnogo praktičnije za grafove sa manjim brojem grana
- U praksi se češće radi sa grafovima koji imaju znatno manje grana od maksimalno mogućeg broja, i tada je bolje koristiti liste povezanosti
- Maksimalan broj grana u neusmerenom grafu je $\boldsymbol{n(n-1)/2}$
- Maksimalan broj grana u usmerenom grafu je $\boldsymbol{n(n-1)}$
- Kod neusmerenog grafa je uvek $\boldsymbol{a_{ii} = 0}$, dok kod usmerenog grafa može postojati povratna grana na čvor





	1	2	3	4	5	6
1	0	1	0	1	0	0
2	0	0	0	0	1	0
3	0	0	0	0	1	1
4	0	1	0	0	0	0
5	0	0	0	1	0	0
6	0	0	0	0	0	1

GRAFOVI – POJMOVI …

- $G = (V, E)$, V su ***čvorovi*** i E su ***grane***
- ***Usmeren*** i ***neusmeren*** graf
- ***Stepen*** $d(v)$ čvora v je broj susednih grana čvoru v, odnosno broj grana koje povezuje v sa drugim čvorovima
  - Kod usmerenog grafa, razlikuje se broj ulaznih grana (***ulazni stepen***) i broj izlaznih grana (***izlazni stepen***)
- ***Put*** od čvora v_1 do čvora v_k je niz čvorova $v_1, v_2, \ldots, v_k$ povezanih granama (v_1, v_2), (v_2, v_3), …, (v_{k-1}, v_k) koje se smatraju delom puta
- Put je ***prost***, ako se svaki čvor pojavljuje samo jednom
- Za čvor u se kaže da je dostižan iz čvora v ako postoji put (usmeren ili neusmeren, zavisno od vrste grafa) od v do u
- Po definiciji čvor v je dostižan iz v

… GRAFOVI – POJMOVI …

- ***Ciklus*** je put čiji se prvi i poslednji čvor poklapaju
- Ciklus je prost, ako se, sem prvog i poslednjeg, ni jedan drug čvor u njemu ne ponavljaju
- ***Neusmeren oblik*** usmerenog grafa $\boldsymbol{G} = (\boldsymbol{V}, \boldsymbol{E})$ je isti graf bez smerova na granama
- Za graf se kaže da je ***povezan*** ako (u njegovom neusmerenom obliku) postoji put između proizvoljna dva čvora
- ***Šuma*** je graf koji (u svom neusmerenom obliku) ne sadrži cikluse
- ***Stablo*** ili ***drvo*** je povezana šuma
- ***Korensko stablo*** je usmereno stablo sa jednim posebnom izdvojenim čvorom koji se zove ***koren***, pri čemu su sve grane usmerene od korena

… GRAFOVI – POJMOVI …

- Graf $\boldsymbol{H} = (\boldsymbol{U}, \boldsymbol{F})$ je ***podgraf*** grafa $\boldsymbol{G} = (\boldsymbol{V}, \boldsymbol{E})$ ako je $\boldsymbol{U} \subset \boldsymbol{V}$ i $\boldsymbol{F} \subset \boldsymbol{E}$
- ***Povezujuće stablo*** neusmerenog grafa $\boldsymbol{G}$ je njegov podgraf koji je stablo i sadrži sve čvorove $\boldsymbol{G}$
- ***Povezujuća šuma*** neusmerenog grafa $\boldsymbol{G}$ je njegov podgraf koji je šuma i sadrži sve čvorove $\boldsymbol{G}$
- ***Indukovani podgraf*** grafa $\boldsymbol{G} = (\boldsymbol{V}, \boldsymbol{E})$ je njegov podgraf $\boldsymbol{H} = (\boldsymbol{U}, \boldsymbol{F})$ takav da je $\boldsymbol{U} \subset \boldsymbol{V}$ i $\boldsymbol{F}$ sadrži sve grane iz $\boldsymbol{E}$ čija su oba kraja u $\boldsymbol{U}$
- Ako graf $\boldsymbol{G} = (\boldsymbol{V}, \boldsymbol{E})$ nije povezan, onda se on može na jedinstven način razložiti u skup povezanih podgrafova, koji se zovu ***komponente povezanosti*** grafa
- Skup čvorova komponente povezanosti je klasa ekvivalencije u odnosu na relaciju dostižnosti za čvorove: komponentu povezanosti $\boldsymbol{G}$ kojoj pripada čvor $\boldsymbol{v}$ čine svi čvorovi dostižni iz $\boldsymbol{v}$

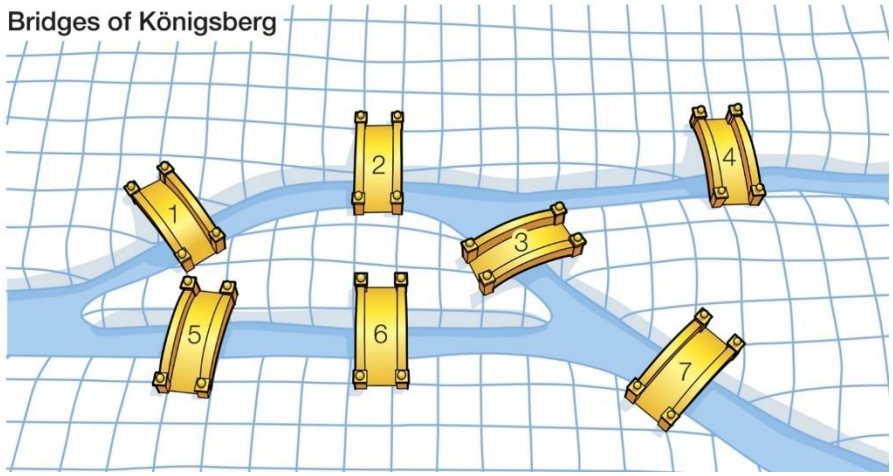
… GRAFOVI – POJMOVI

- Komponenta povezanosti grafa G je maksimalni povezani podgraf G (takav podgraf koji nije pravi podgraf ni jednog drugog povezanog podgrafa G)
- ***Bipartitni graf*** je graf čiji se čvorovi mogu podeliti na dva disjunktna podskupa tako da u grafu postoje samo grane između čvorova iz različitih podskupova
- ***Težinski graf*** je graf čijim su granama pridruženi realni brojevi (težine, cene, dužine)

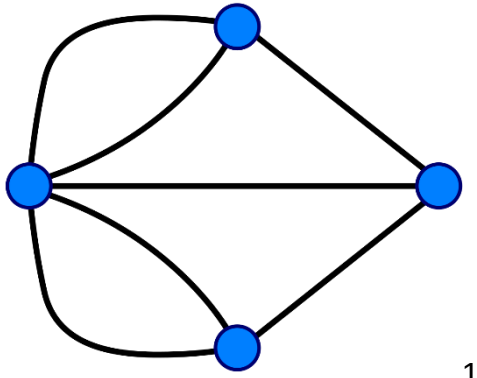
OJLEROVI GRAFOVI

OJLEROVI GRAFOVI …

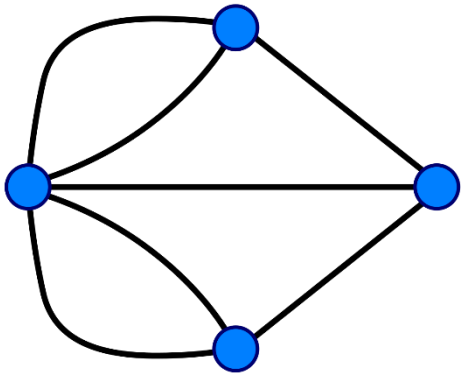
- Dovodi se u vezu sa prvim rešenim problemom teorije grafova
- Švajcarski matematičar, Leonard Ojler, problem iz 1736.
- U gradu Kenigsberg (danas Kalinjingrad) leži na obalama i dva ostvra na reci Pregel, koji su povezani sa 7 mostova



- Problem je bio da li je moguće početi šetnju iz bilo koje tačke u gradu i vratiti se u polaznu tačku, prelazeći svaki most samo jednom
- Rešenje se dobija uopštavanjem i primenom grafova
- Problem se može formulisati kao problem iz teorije grafova



.. OJLEROVI GRAFOVI …



- Da li je moguće u povezanom grafu pronaći ciklus, koji svaku granu sadrži tačno jednom (Ojlerov ciklus)
- Da li je moguće nacrtati ovaj graf a da se ne podigne olovka sa papira, tako da olovka završi put na mestu sa kojeg je i krenula?
- Ojler je rešio ovaj problem dokazavši da je to moguće samo ako je graf povezan i svi njegovi čvorovi imaju paran stepen
- Opisani graf je povezan, ali na žalost čvorovi nemaju paran stepen
- Teorema se dokazuje indukcijom
- Predstavlja i efikasan algoritam za nalaženje Ojlerovih ciklusa u gradu
- ***Problem:*** U zadatom neusmerenom povezanom grafu $\boldsymbol{G = (V, E)}$ čiji svi koreni imaju paran stepen, pronaći zatvoren put $\boldsymbol{P}$, takav da se u njemu svaka grana iz $\boldsymbol{E}$ pojavljuje samo jednom

.. OJLEROVI GRAFOVI …

- Lako je pokazati da u grafu ako postoji Ojlerov ciklus, onda svi čvorovi grafa moraju imati paran stepen
- Za vreme obilaska zatvorenog ciklusa, u svaki čvor se ulazi isto toliko puta koliko puta se iz njega izlazi
- Pošto se svaka grana prolazi tačno jednom, broj grana susednih proizvoljnom čvoru mora biti paran
- Da bi se dokazalo da je ovo dovoljno, moramo odabrati parametre po kojima će biti izvedena indukcija
- Izbor treba da dovede do smanjivanja problema, bez njegove promene
- Ako uklonimo čvor iz grafa, stepeni čvorova u dobijenom grafu više nisu svi parni (znači, nije najbolji kandidat za parametre i indukciju)

.. OJLEROVI GRAFOVI …

- Ako uklonimo grane, onda moramo da uklonimo takav skup grana $\boldsymbol{S}$, da za svaki čvor $\boldsymbol{v}$ grafa, broj grana iz $\boldsymbol{S}$ susednih $\boldsymbol{v}$ bude paran (i $\boldsymbol{0}$)
- Proizvoljan ciklus zadovoljava ovaj uslov, pa se postavlja pitanje da li Ojlerov graf uvek sadrži ciklus
- Pretpostavimo da smo započeli obilazak grafa iz proizvoljnog čvora $\boldsymbol{v}$ proizvoljnim redosledom, sigurno je da će se obilazak ranije ili kasnije završiti u čvoru $\boldsymbol{v}$, jer kad god uđemo u neki čvor, smanjujemo njegov stepen za jedan, činimo ga neparnim, pa ga uvek možemo i napustiti (ovakav obilazak ne mora da sadrži sve grane grafa)
- ***Induktivna hipoteza***
  Povezani graf sa $\boldsymbol{m}$ grana čiji svi čvorovi imaju paran stepen, sadrži Ojlerov ciklus, koji se može pronaći

.. OJLEROVI GRAFOVI

- ***Dokaz***
  - Posmatrajmo graf $\boldsymbol{G} = (\boldsymbol{V}, \boldsymbol{E})$ sa $\boldsymbol{m}$ grana
  - Neka je $\boldsymbol{P}$ neki ciklus u $\boldsymbol{G}$, i neka je $\boldsymbol{G'}$ graf dobijen uklanjanjem grana koje čine $\boldsymbol{P}$ iz grafa $\boldsymbol{G}$
  - Stepeni svih čvorova u $\boldsymbol{G'}$ su parni, jer je broj uklonjenih grana susednih bilo kom čvoru paran
  - Ipak se induktivna hipoteza ne može primeniti na graf $\boldsymbol{G'}$, jer on ne mora biti povezan
  - Neka su $\boldsymbol{G'_1, G'_2, \ldots, G'_k}$ komponente povezanosti grafa $\boldsymbol{G'}$
  - U svakoj komponenti stepeni svih čvorova su parni
  - Pored toga, broj grana u svakoj komponenti je manji od $\boldsymbol{m}$ (njihov ukupan broj grana manji je od $\boldsymbol{m}$)
  - Prema tome, induktivna hipoteza može se primeniti na sve komponente: u svakoj komponenti $\boldsymbol{G'_i}$ postoji Ojlerov ciklus $\boldsymbol{P'_i}$, i mi znamo da ga pronađemo

.. OJLEROVI GRAFOVI …

- ***Dokaz (nastavak)***
  - Potrebno je sada sve ove cikluse objediniti u jedan Ojlerov ciklus za graf G
  - Polazimo iz bilo kog čvora ciklusa P ("magistralnog puta") sve dok ne dođemo do nekog čvora v_j koji pripada komponenti G'_j
  - Tada obilazimo komponentu G'_j ciklusom P_j ("lokalnim putem") i vraćamo se u čvor v_j
  - Nastavljajući na taj način, obilazeći cikluse komponenti u trenutku nailaska na njih, na kraju ćemo se vratiti u polaznu tačku
  - U tom trenutku sve grane grafa G prođene su tačno jednom, što znači da je konstruisan Ojlerov ciklus

.. OJLEROVI GRAFOVI …

- **Ojlerov ciklus: Neusmereni graf sadrži Ojlerov ciklus ako su ispunjena sledeća dva uslova:**
  1. Svi čvorovi stepena većeg od nule su povezani (oni sa stepenom nula ne pripadaju ciklusu i ne zanimaju nas)
  2. Svi čvorovi imaju paran broj grana
- **Ojlerov put (putanja)*:* Neusmereni graf sadrži Ojlerov put ako su ispunjena sledeća dva uslova:**
  1. Isti je kao i 1. uslov za Ojlerov ciklus
  2. Svi čvorovi imaju paran broj ili samo dva čvora imaju neparan broj dok ostali imaju paran broj grana

.. OJLEROVI GRAFOVI …

- **Hierholzer's algorithm za pronalaženje Ojlerovog ciklusa:**
  1. Kreni od proizvoljnog čvora i prati trag preko grana sve dok se ne vratiš do čvora od kojeg si krenuo.
  2. Prvi korak možda nije pokrio sve grane, zato dok god postoji čvor koji je deo obilaska a čije grane nisu deo obilaska, započni novi obilazak iz tog čvora po nekoj od neposećenih grana. Na kraju spoji novi obilazak sa prethodnim.
  3. Pretpostavlja se da je originalni graf povezan, tako da će korak dva potrošiti sve grane.
- Ovaj algoritam je najbolje implementirati preko liste povezanosti
- Algoritam je linearan $\boldsymbol{O}(|\boldsymbol{E}|)$ ako se radi sa listom povezanosti

OBILASCI GRAFOVA

OBILASCI GRAFOVA

- Prvi problem na koji se nailazi pri konstrukciji bilo kog algoritma za obradu grafa je kako pregledati ulaz
- Ovaj problem u nizovima i skupovima je trivijalan zbog jednodimenzionalnosti ulaza (mogu se lako pregledati linearnim redosledom)
- Pregledanje grafa, odnosno njegov obilazak, nije trivijalan problem
- Postoje dva osnovna algoritma za obilazak grafa:
  - pretraga u dubinu
  - pretraga u širinu

PRETRAGA U ŠIRINU …

- Pretraga u širinu (ili BFS, što je skraćenica od breadth-first-search) je obilazak grafa na sistematičan način, nivo po nivo
- Jedan od najjednostavnijih algoritama za pretragu grafa i veoma važna osnova za druge važne grafovske algoritme
- Za dati graf $\boldsymbol{G} = (\boldsymbol{V}, \boldsymbol{E})$, počev od naglašenog izvorišnog čvora $\boldsymbol{s}$, najpre se posećuju svi susedi od $\boldsymbol{s}$ (deca u stablu pretrage, nivo jedan), zatim se dolazi do svih "unuka" (nivo dva), i tako dalje
- Za svaki čvor računa se rastojanje (najkraći put) koji treba preći od $\boldsymbol{s}$ do njega
- Paralelno sa tim se pravi breadth-first stablo koji sadrži sve dostižne čvorove od $\boldsymbol{s}$, počev od $\boldsymbol{s}$ kao korena stabla
- Za svaki čvor $\boldsymbol{v}$ koji je dostižan iz $\boldsymbol{s}$ prost put između $\boldsymbol{v}$ i $\boldsymbol{s}$ u breadth-first stablu odgovara najkraćem putu između $\boldsymbol{v}$ i $\boldsymbol{s}$ u $\boldsymbol{G}$
- Algoritam radi i za usmerene i za neusmerene grafove

… PRETRAGA U ŠIRINU …

- BFS pretraga se tako zove, jer širi front između otkrivenih i neotkrivenih čvorova uniformno preko linije (breadth) fronta, tj. algoritam otkriva sve čvorove na nivou $\boldsymbol{k}$, pre nego što otkrije sve čvorove na nivou $\boldsymbol{k+1}$
- Da bi se vodilo računa o progresu, algoritam boji svaki čvor u belo, sivo ili crno
  - Svi čvorovi kreću kao beli, kada se čvor otkrije, postaje siv ili crn
  - Postoje dve nijanse kako bi se obezbedilo da algoritam radi u breadth-first maniru
  - Crna označava da su svi susedni čvorovi otkriveni (sivi ili crni)
  - Siva označava da postoje susedni čvorovi koji još uvek nisu posećeni (beli)
  - Sivi čvorovi predstavljaju front

… PRETRAGA U ŠIRINU …

- BFS pretraga formira breadth-first stablo
  - Inicijalno sadrži samo koren, izvorni čvor s
  - Kad god algoritam otkrije novi, beli, čvor v tokom pretraživanja suseda čvora u (v je otkriven iz u), čvor v i grana (v, u) se dodaju u stablo
  - Kaže se da je u predak (nadređen čvor) čvora v u breadth-first stablu
  - Kako se čvor otkriva samo jednom, ima samo jednog pretka
  - Ako je čvor u na prostoj putanji od korena s do čvora v, onda je čvor u nadređen čvor čvoru v, i čvor v je podređen čvor čvoru u

… PRETRAGA U ŠIRINU …

- BFS algoritam
  - podrazumeva da je graf predstavljen listom povezanosti
  - Svakom čvoru su pridruženi dodatni atributi
  - Boja čvora u je smeštena u atributu $u.color$ a putanje do nadređenog čvora u $u.\pi$ (ako još nije određen nadređen čvor, ili ga nema, vrednost ovog atributa je NIL
  - Rastojanje od početnog čvora s do u čuva se u atributu $u.d$ (rastojanje se određuje tokom izvršavanja algoritma)
  - Takođe, koristi se FIFO red za rad sa sivim čvorovima

```
BFS(G, s)
 1  for each vertex u ∈ G.V − {s}
 2      u.color = WHITE
 3      u.d = ∞
 4      u.π = NIL
 5  s.color = GRAY
 6  s.d = 0
 7  s.π = NIL
 8  Q = ∅
 9  ENQUEUE(Q, s)
10  while Q ≠ ∅
11      u = DEQUEUE(Q)
12      for each v ∈ G.Adj[u]
13          if v.color == WHITE
14              v.color = GRAY
15              v.d = u.d + 1
16              v.π = u
17              ENQUEUE(Q, v)
18      u.color = BLACK
```

… PRETRAGA U ŠIRINU …

- BFS algoritam (nastavak)
  - Prvo se za sve čvorove osim ***s*** postave početne vrednosti
  - Čvor ***s*** se boji u sivo, jer se pretpostavlja da je on prethodno otkriven
  - Rastojanje čvora ***s*** od samog sebe je naravno 0 i pretpostavlja se da nema nadređeni čvor
  - FIFO red ***Q*** se inicijalizuje i u njega se doda samo čvor ***s***
  - ***while*** petlja od 10-18 linije prolazi kroz preostale sive čvorove čije povezanosti još nisu istražene, pretvara bele čvorove u sive, dodaje ih u FIFO red ***Q*** i kada istraži sve povezane čvorove čvora ***u***, njega prevodi u crno

```
BFS(G, s)
 1  for each vertex u ∈ G.V − {s}
 2       u.color = WHITE
 3       u.d = ∞
 4       u.π = NIL
 5  s.color = GRAY
 6  s.d = 0
 7  s.π = NIL
 8  Q = ∅
 9  ENQUEUE(Q, s)
10  while Q ≠ ∅
11       u = DEQUEUE(Q)
12       for each v ∈ G.Adj[u]
13            if v.color == WHITE
14                 v.color = GRAY
15                 v.d = u.d + 1
16                 v.π = u
17                 ENQUEUE(Q, v)
18       u.color = BLACK
```

… PRETRAGA U ŠIRINU …

- BFS algoritam (nastavak)
  - ***while*** petlja održava sledeće: u 10.oj liniji FIFO red Q nije prazan i sadrži sive čvorove
  - Očigledno je da ovo važi pre prve iteracija ***while*** petlje i da svaka iteracija održava ovo invarijantnim (u suprotnom izvršavanje petlje se prekida)
  - U startu Q sadrži samo s
  - U svakoj iteraciji ***while*** petlje čita se sledeći sivi čvor u
  - u liniji 12 ***for*** petljom se prolazi kroz sve čvorove povezane sa u i svaki beli čvor v se pretvara u sivi
  - čvoru v se kao nadređeni čvor podesi čvor u, rastojanje je očigledno rastojanje do u povećano za jedan (čvor v)

```
BFS(G, s)
 1  for each vertex u ∈ G.V − {s}
 2       u.color = WHITE
 3       u.d = ∞
 4       u.π = NIL
 5  s.color = GRAY
 6  s.d = 0
 7  s.π = NIL
 8  Q = ∅
 9  ENQUEUE(Q, s)
10  while Q ≠ ∅
11       u = DEQUEUE(Q)
12       for each v ∈ G.Adj[u]
13            if v.color == WHITE
14                 v.color = GRAY
15                 v.d = u.d + 1
16                 v.π = u
17                 ENQUEUE(Q, v)
18       u.color = BLACK
```

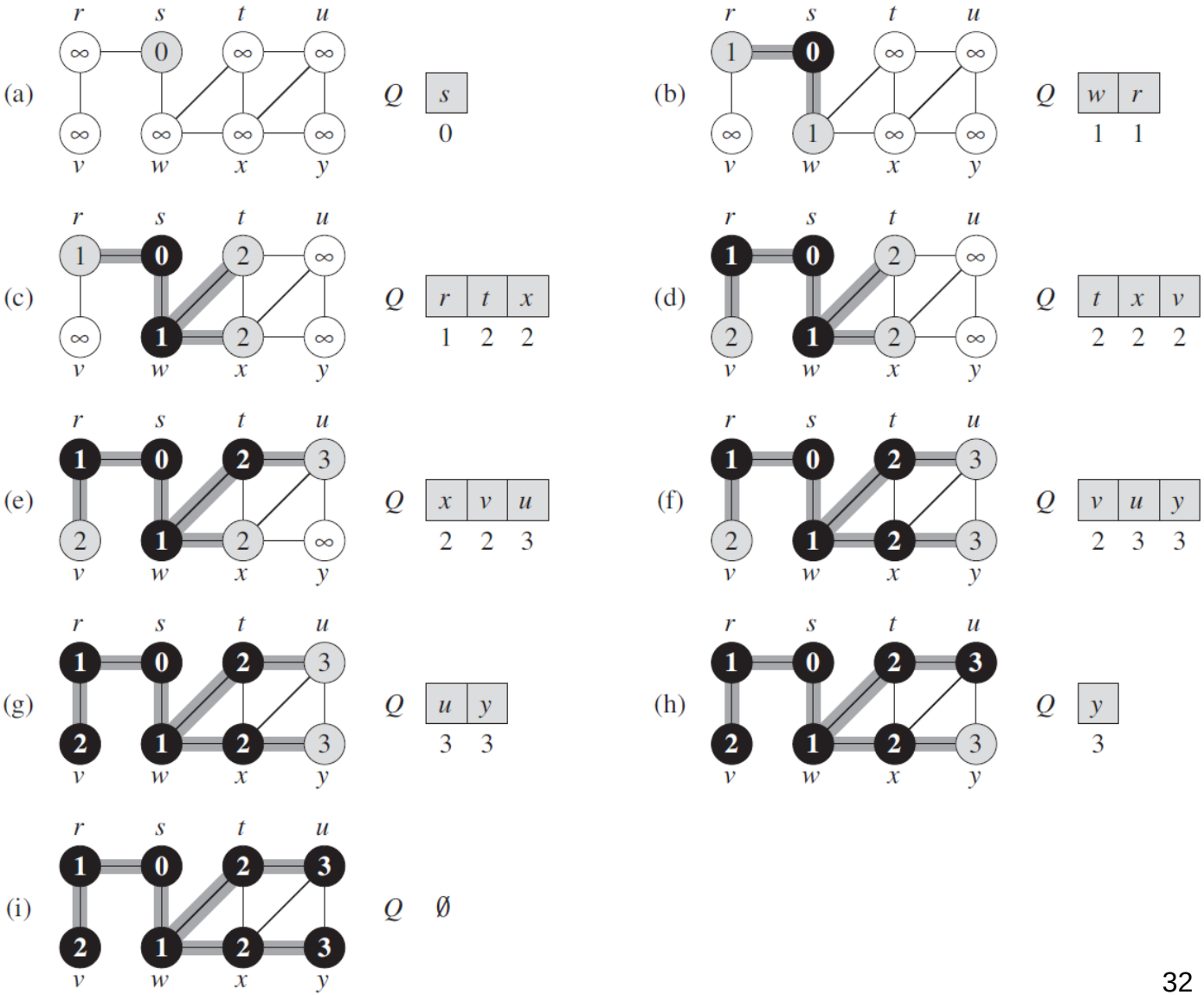
… PRETRAGA U ŠIRINU …

- BFS algoritam (nastavak)
  - Kada su svi čvorovi povezani sa u istraženi, čvor u se boji u crno
  - Invarijantnost petlje se održava time što svaki put kada se otkrije beli čvor, on se doda u red Q kao novi sivi čvor, takođe kada se čvor izvadi iz reda Q on se po završetku ***for*** petlje boji u crno
  - Rezultati BFS algoritma se mogu razlikovati u zavisnosti od redosleda pristupanja susedima datog čvora (linija 12)
  - breadth-first stablo se može razlikovati, ali su zato distance uvek iste

```
BFS(G, s)
 1  for each vertex u ∈ G.V − {s}
 2      u.color = WHITE
 3      u.d = ∞
 4      u.π = NIL
 5  s.color = GRAY
 6  s.d = 0
 7  s.π = NIL
 8  Q = ∅
 9  ENQUEUE(Q, s)
10  while Q ≠ ∅
11      u = DEQUEUE(Q)
12      for each v ∈ G.Adj[u]
13          if v.color == WHITE
14              v.color = GRAY
15              v.d = u.d + 1
16              v.π = u
17              ENQUEUE(Q, v)
18      u.color = BLACK
```

… PRETRAGA U ŠIRINU …

- BFS algoritam kako radi na primeru



… PRETRAGA U ŠIRINU …

- Analiza BFS algoritma
  - Svaki čvor se dodaje i izvlači iz $\boldsymbol{Q}$ tačno jednom (linija 13) i ta operacija traje $\boldsymbol{O(1)}$, što znači da operacije sa redom imaju kompleksnost $\boldsymbol{O(V)}$
  - Svaku listu povezanosti algoritma skenira samo jednom
  - Kako je ukupna dužina svih listi povezanosti jednaka $\Theta(\boldsymbol{E})$, onda je i kompleksnost skeniranje listi povezanosti $\boldsymbol{O(E)}$
  - Inicijalizacija takođe traje $\boldsymbol{O(V)}$
  - Pa se dobija da je BFS algoritam linearan i da je njegovo vreme izvršavanja $\boldsymbol{O(V+E)}$
  - Algoritam zavisi od veličine listi povezanosti

```
BFS(G, s)
 1  for each vertex u ∈ G.V − {s}
 2      u.color = WHITE
 3      u.d = ∞
 4      u.π = NIL
 5  s.color = GRAY
 6  s.d = 0
 7  s.π = NIL
 8  Q = ∅
 9  ENQUEUE(Q, s)
10  while Q ≠ ∅
11      u = DEQUEUE(Q)
12      for each v ∈ G.Adj[u]
13          if v.color == WHITE
14              v.color = GRAY
15              v.d = u.d + 1
16              v.π = u
17              ENQUEUE(Q, v)
18      u.color = BLACK
```

… PRETRAGA U ŠIRINU …

- Osobine BFS algoritma
  - Najkraće rastojanje $\boldsymbol{\delta(s,v)}$ od čvora $\mathbf{s}$ do čvora $\mathbf{v}$ u grafu $\mathbf{G=(V,E)}$ je onaj put grana između $\boldsymbol{s}$ i $\boldsymbol{v}$ koji sadrži najmanji broj grana u svom putu (ovo se menja kada se dodaju težine)
  - Ako ne postoji put od čvora $\boldsymbol{s}$ do čvora $\boldsymbol{v}$ u grafu $\boldsymbol{G=(V,E)}$ onda je $\boldsymbol{\delta(s,v)=\infty}$
  - $\boldsymbol{\delta(s,v)}$ se naziva i najkraćim putem
  - Važna osobina najkraćeg puta je sledeća:
    Neka $\boldsymbol{G=(V,E)}$ usmereni ili neusmereni graf, i neka je $\boldsymbol{s\in V}$ proizvoljan čvor iz $\boldsymbol{V}$, onda za svaku granu $\boldsymbol{(u,v)\in E}$ važi $\boldsymbol{\delta(s,v)\leq\delta(s,u)+1}$
  - Dokaz:
    Ako je čvor $\boldsymbol{u}$ dostižan iz $\boldsymbol{s}$, onda je i $\boldsymbol{v}$ (zašto?). I najkraći put od $\boldsymbol{s}$ do $\boldsymbol{v}$ ne može biti duži od najkraćeg put od $\boldsymbol{s}$ do $\boldsymbol{u}$ plus grana $\boldsymbol{(u,v)}$, tako da nejednakost važi. Ako čvor $\boldsymbol{u}$ nije dostižan iz $\boldsymbol{s}$, onda je $\boldsymbol{\delta(s,u)=\infty}$ i nejednakost opet važi

… PRETRAGA U ŠIRINU …


```
BFS(G, s)
1   for each vertex u ∈ G.V − {s}
2       u.color = WHITE
3       u.d = ∞
4       u.π = NIL
5   s.color = GRAY
6   s.d = 0
7   s.π = NIL
8   Q = ∅
9   ENQUEUE(Q, s)
10  while Q ≠ ∅
11      u = DEQUEUE(Q)
12      for each v ∈ G.Adj[u]
13          if v.color == WHITE
14              v.color = GRAY
15              v.d = u.d + 1
16              v.π = u
17              ENQUEUE(Q, v)
18      u.color = BLACK
```

- Osobine BFS algoritma (nastavak)
  - Želimo da pokažemo da je BFS algoritam za svaki čvor $\boldsymbol{v}$ generiše $\boldsymbol{v.d = \delta(s, v)}$
  - Prvo treba dokazati da važi sledeće:
    Neka $\boldsymbol{G = (V, E)}$ usmereni ili neusmereni graf, i neka je BFS obradio $\boldsymbol{G}$ počev od čvora $\boldsymbol{s \in V}$. Po završetku BFS algoritma, za svaki čvor $\boldsymbol{v \in V}$, vrednost $\boldsymbol{v.d}$ određena BFS algoritmom važi $\boldsymbol{v.d \geq \delta(s, v)}$
  - Dokaz se vrši indukcijom :
    - Osnova indukcije je stanje algoritam u liniji 9. Rastojanje čvora $\boldsymbol{s}$ od $\boldsymbol{s}$ je 0, a za sve ostale čvorove $\boldsymbol{v \in V - \{s\}}$ je $\boldsymbol{v.d = \infty \geq \delta(s, v)}$, uslov je tu zadovoljen
    - U sledećem koraku dokaza zamisliti beli čvor $\boldsymbol{v}$ do kojeg se stiglo pretragom čvora $\boldsymbol{u}$. Indukcija implicira da je $\boldsymbol{u.d \geq \delta(s, u)}$, odavde sledi: $\boldsymbol{v.d = u.d + 1 \geq \delta(s, u) + 1 \geq \delta(s, v)}$
    - Čvor $\boldsymbol{v}$ se izvlači iz reda i više se tamo ne vraća, pa njegov rastojanje više ne računa

… PRETRAGA U ŠIRINU …

- Osobine BFS algoritma (nastavak)
  - Ali da li je $v.d = \delta(s, v)$?
  - Sledeći korak je dokazati red u sebi sadrži najviše dve d vrednosti, sledeća lema:
    Pretpostaviti da tokom izvršavanja BFS algoritma nad grafom $G = (V, E)$, red Q sadrži čvorove $\langle v_1, v_2, \ldots, v_r \rangle$, gde je v_1 vrh reda, v_r je kraj reda, onda je $v_r.d \leq v_1.d + 1$ i $v_i.d \leq v_{i+1}.d$ za
    $$i = 1, 2, \ldots, r - 1$$
  - Dokaz se izvodi indukcijom i lema mora da važi i posle dodavanja u red i nakon brisanja iz reda:
    - Inicijalno kada red sadrži samo s, lema stoji
    - Kada se iz reda izvuče v_1, v_2 dolazi na vrh reda
    - U skladu sa hipotezom $v_1.d \leq v_2.d$, takođe ostaje $v_r.d \leq v_1.d + 1 \leq v_2.d + 1$, pa i dalje važi
    - Kada se u red doda novi čvor (linija 17), ono se dodaje na kraj i postaje v_{r+1}

```
BFS(G, s)
 1  for each vertex u ∈ G.V − {s}
 2      u.color = WHITE
 3      u.d = ∞
 4      u.π = NIL
 5  s.color = GRAY
 6  s.d = 0
 7  s.π = NIL
 8  Q = ∅
 9  ENQUEUE(Q, s)
10  while Q ≠ ∅
11      u = DEQUEUE(Q)
12      for each v ∈ G.Adj[u]
13          if v.color == WHITE
14              v.color = GRAY
15              v.d = u.d + 1
16              v.π = u
17              ENQUEUE(Q, v)
18      u.color = BLACK
```

… PRETRAGA U ŠIRINU …

- Osobine BFS algoritma (nastavak)
  - Dokaz (nastavak):
    - Kada se u red doda novi čvor (linija 17), ono se dodaje na kraj i postaje $\boldsymbol{v_{r+1}}$
    - $\boldsymbol{u}$ je već izvađeno iz reda i njegova se lista trenutno skenira, tako da za čvor na vrhu reda važi $\boldsymbol{u.d \leq v_1.d}$
    - Dalje sledi da je $\boldsymbol{v_{r+1}.d = v.d = u.d + 1 \leq v_1.d + 1}$
    - Dalje iz induktivne hipoteze sledi da je $\boldsymbol{v_r.d \leq u.d + 1 = v.d = v_{r+1}.d}$ i sve ostale jednakosti se i dalje drže
    - To znači da lema važi

```
BFS(G, s)
 1  for each vertex u ∈ G.V − {s}
 2      u.color = WHITE
 3      u.d = ∞
 4      u.π = NIL
 5  s.color = GRAY
 6  s.d = 0
 7  s.π = NIL
 8  Q = ∅
 9  ENQUEUE(Q, s)
10  while Q ≠ ∅
11      u = DEQUEUE(Q)
12      for each v ∈ G.Adj[u]
13          if v.color == WHITE
14              v.color = GRAY
15              v.d = u.d + 1
16              v.π = u
17              ENQUEUE(Q, v)
18      u.color = BLACK
```

… PRETRAGA U ŠIRINU …

- Osobine BFS algoritma (nastavak)
  - Iz prethodnog proizilazi da ako se čvorovi v_i i v_j nalaze u redu Q i ako je v_i dodat u red pre v_j, onda je $v_i.d \leq v_j.d$ u trenutku kada je v_j dodat u red
  - Na kraju se dolazi do sledeće teoreme: Neka $G = (V, E)$ usmereni ili neusmereni graf, i neka je BFS obradio G počev od čvora $s \in V$. Onda, tokom svog izvršavanja, BFS algoritam otkriva svaki čvor $v \in V$ dostižan iz s i nakon svog završetka $v.d = \delta(s, v)$ za svako $v \in V$. Pored svega za svaki čvor $v \neq s$ jedan od najkraćih puteva od s do v je put od s do $v.\pi$ plus grana po $(v.\pi, v)$
  - Dokaz teoreme se bazira na kontradikciji

```
BFS(G, s)
 1  for each vertex u ∈ G.V − {s}
 2      u.color = WHITE
 3      u.d = ∞
 4      u.π = NIL
 5  s.color = GRAY
 6  s.d = 0
 7  s.π = NIL
 8  Q = ∅
 9  ENQUEUE(Q, s)
10  while Q ≠ ∅
11      u = DEQUEUE(Q)
12      for each v ∈ G.Adj[u]
13          if v.color == WHITE
14              v.color = GRAY
15              v.d = u.d + 1
16              v.π = u
17              ENQUEUE(Q, v)
18      u.color = BLACK
```

… PRETRAGA U ŠIRINU …

- Osobine BFS algoritma (nastavak)
  - Dokaz:
    Neka se pretpostavi da postoji čvor $\boldsymbol{v \neq s}$ kod kojeg $\boldsymbol{v.d}$ nije $\boldsymbol{\delta(s,v)}$, već $\boldsymbol{v.d > \delta(s,v)}$
  - Čvor $\boldsymbol{v}$ je dostižan iz $\boldsymbol{s}$, jer bi u suprotnom bilo $\boldsymbol{\delta(s,v) = \infty \geq v.d}$
  - Neka je $\boldsymbol{u}$ čvor koji neposredno prethodi $\boldsymbol{v}$ i neka se on nalazi na najkraćem putu od $\boldsymbol{s}$ tako da je $\boldsymbol{u.d = \delta(s,u)}$, prateći sve prethodne dokaze $\boldsymbol{\delta(s,v) = \delta(s,u) + 1}$, i kada se sve spoji dobija se:
    $$\boldsymbol{v.d > \delta(s,v) = \delta(s,u) + 1 = u.d + 1}$$

```
BFS(G, s)
 1  for each vertex u ∈ G.V − {s}
 2      u.color = WHITE
 3      u.d = ∞
 4      u.π = NIL
 5  s.color = GRAY
 6  s.d = 0
 7  s.π = NIL
 8  Q = ∅
 9  ENQUEUE(Q, s)
10  while Q ≠ ∅
11      u = DEQUEUE(Q)
12      for each v ∈ G.Adj[u]
13          if v.color == WHITE
14              v.color = GRAY
15              v.d = u.d + 1
16              v.π = u
17              ENQUEUE(Q, v)
18      u.color = BLACK
```

… PRETRAGA U ŠIRINU …

- Osobine BFS algoritma (nastavak)
  - Dokaz (nastavak):

$$\boldsymbol{v.d > \delta(s,v) = \delta(s,u) + 1 = u.d + 1}$$

  - Ako se pogleda algoritam, vidi se sledeće:
    - Ako je čvor beo, u liniji 15 dobija se da je $\boldsymbol{v.d = u.d + 1}$, što je kontradikcija prethodnoj formuli
    - Ako je čvor crni, onda je on već izbačen iz reda i shodno tome $\boldsymbol{v.d \leq u.d}$, kontradikcija
    - Ako je čvor siv, onda je obojen u sivo tokom pretraživanja nekog čvora $\boldsymbol{w}$ koji prethodi čvoru $\boldsymbol{u}$ i koji je već izvađen iz reda i za koda važi da je $\boldsymbol{v.d = w.d + 1}$. Kako je $\boldsymbol{w.d \leq u.d}$, onda je i $\boldsymbol{v.d = w.d + 1 \leq u.d + 1}$, kontradikcija
  - Tako da je $\boldsymbol{v.d = \delta(s,v)}$ za svako $\boldsymbol{v \in V}$ i svi dostižni čvorovi moraju biti otkriveni, jer je u suprotnom $\boldsymbol{v.d = \infty > \delta(s,v)}$

```
BFS(G, s)
 1  for each vertex u ∈ G.V − {s}
 2      u.color = WHITE
 3      u.d = ∞
 4      u.π = NIL
 5  s.color = GRAY
 6  s.d = 0
 7  s.π = NIL
 8  Q = ∅
 9  ENQUEUE(Q, s)
10  while Q ≠ ∅
11      u = DEQUEUE(Q)
12      for each v ∈ G.Adj[u]
13          if v.color == WHITE
14              v.color = GRAY
15              v.d = u.d + 1
16              v.π = u
17              ENQUEUE(Q, v)
18      u.color = BLACK
```

… PRETRAGA U ŠIRINU

- Osobine BFS algoritma (nastavak)
  - Bez dokaza (može se naći u knjizi) uzećemo još i da breadth-first stablo, koje se gradi tokom BFS algoritma je takvo da su i sva njegova podstabla breadth-first stabla za čvor iz kojeg se posmatra, i graf koji se iz njega gradi tzv. ***predecessor subgraph***
  - Predecessor subgraph $\boldsymbol{G_\pi = (V_\pi, E_\pi)}$ gde je $\boldsymbol{V_\pi = \{v \in V : v.\pi \neq NIL\} \cup \{s\}}$ i $\boldsymbol{E_\pi = \{(v.\pi, v) : v \in V - \{s\}\}}$
  - Algoritam za ispis najkraćeg puta od $\boldsymbol{s}$ do $\boldsymbol{v}$

```
PRINT-PATH(G, s, v)
1  if v == s
2        print s
3  elseif v.π == NIL
4        print "no path from" s "to" v "exists"
5  else PRINT-PATH(G, s, v.π)
6        print v
```

PRETRAGA U DUBINU …

- Kod pretrage u dubinu (*depth-first search*), umesto da se ide nivo po nivo, ide se onoliko duboko koliko je to moguće
- Ideja je da se, počevši iz nekog čvora v, isprate sve grane dokle može da se stigne
- Kada su ispitane sve grane, algoritam se vraća na prethodni čvor iz koga se došlo do v i nastavlja sa ispitivanjem preostalih grana koje nisu istražene
- Suštinski algoritam se spušta do krajnjih čvorova (u odnosu na početni čvor) i vraća se unazad, pri čemu je mera pretraga grana između čvorova
- Kada su pregledane sve grane i čvorovi koji se mogu dostići iz početnog čvora, provera se da li postoji još neki čvor koji nije pretražen i ako ima algoritam kreće od njega ponovo, sve dok nisu otkriveni svi čvorovi

… PRETRAGA U DUBINU …

- Kao i kod pretrage u širinu, kad god se otkrije novi čvor, praćenjem liste povezanosti, postavi se vrednost njemu nadređenog čvora na čvor sa kojeg se došlo do njega
  - Ako se došlo do čvora $\boldsymbol{v}$ preko čvora $\boldsymbol{u}$, onda se $\boldsymbol{v.\pi}$ postavi na $\boldsymbol{u}$
- Za razliku od BFS algoritma, gde se za *predecessor subgraph* formira jedno stablo, ovde se za *predecessor subgraph* može formirati nekoliko stabla, jer se pretraga može ponoviti iz nekoliko izvora
- Ovde se sad *predecessor subgraph* definiše malo drugačije: $\boldsymbol{G_\pi = (V, E_\pi)}$ gde je $\boldsymbol{E_\pi = \{(v.\pi, v): v \in V \wedge v.\pi \neq NIL\}}$
- *Predecessor subgraph formira* ***depth-first šumu*** koja je sastavljena od nekoliko depth-first stabla
- $\boldsymbol{E_\pi}$ grane su zapravo grane stabla

… PRETRAGA U DUBINU …

- Kao i kod pretrage u širinu, čvorovi se farbaju kako bi se označilo u kojoj su fazi pretrage
  - Svi čvorovi kreću kao beli, kada se čvor otkrije, postaje siv ili crn
  - Siva označava da je čvor otkriven, ali da nisu istražene sve grane
  - Crna označava da su sve grane za dati čvor istražene
  - Obezbeđuje da se svaki čvor nađe samo jednom u depth-first stablu
- Dodatno DFS algoritam beleži još 2 vremena
  - Vreme kada je čvor v otkriven, atribut $v.d$ (postaje siv)
  - Vreme kada su proučene sve grane čvora v, atribut $v.f$ (postaje crn)
  - Ova vremena nose značajne informacije o strukturi grafa i olakšavaju analizu ponašanja DFS algoritma
  - Oba atributa su intidžeri čija vrednost može biti od 1 do $2|V|$, jer je svaki čvor samo jednom otkriven, i jednom završen (zacrnjen)
  - Važi: $v.d < v.f$, tj. $beo < v.d \leq siv < v.f \leq crn$

… PRETRAGA U DUBINU …

- DFS algoritam

```
DFS(G)
1  for each vertex u ∈ G.V
2      u.color = WHITE
3      u.π = NIL
4  time = 0
5  for each vertex u ∈ G.V
6      if u.color == WHITE
7          DFS-VISIT(G, u)
```

```
DFS-VISIT(G, u)
 1  time = time + 1          // white vertex u has just been discovered
 2  u.d = time
 3  u.color = GRAY
 4  for each v ∈ G.Adj[u]    // explore edge (u, v)
 5      if v.color == WHITE
 6          v.π = u
 7          DFS-VISIT(G, v)
 8  u.color = BLACK          // blacken u; it is finished
 9  time = time + 1
10  u.f = time
```

… PRETRAGA U DUBINU …

- DFS algoritam (nastavak)
  - Ulazni graf može biti usmeren i neusmeren
  - Promenljiva ***time*** je globalna promenljiva za praćenje vremena (brojanje koraka)
  - Algoritam u startu inicijalizuje sve čvorove u grafu
  - Zatim se inicijalizuje brojač
  - Na linijama od 5-7 ispituje se svaki čvor iz grafa i ako je on beo (nije pretražen), pristupa mu se i pretražuje se DFS-VISIT procedurom
  - Sa svakim pozivom DFS-VISIT procedure dati čvor $\boldsymbol{u}$ postaje koren novog stabla u depth-first šumi
  - Kad se DFS algoritam završi svaki čvor iz grafa imaće vreme otkrića i vreme završetka, $\boldsymbol{v.d}$ i $\boldsymbol{v.f}$

```
DFS(G)
1  for each vertex u ∈ G.V
2      u.color = WHITE
3      u.π = NIL
4  time = 0
5  for each vertex u ∈ G.V
6      if u.color == WHITE
7          DFS-VISIT(G, u)
```

```
DFS-VISIT(G, u)
 1  time = time + 1
 2  u.d = time
 3  u.color = GRAY
 4  for each v ∈ G.Adj[u]
 5      if v.color == WHITE
 6          v.π = u
 7          DFS-VISIT(G, v)
 8  u.color = BLACK
 9  time = time + 1
10  u.f = time
```

… PRETRAGA U DUBINU …

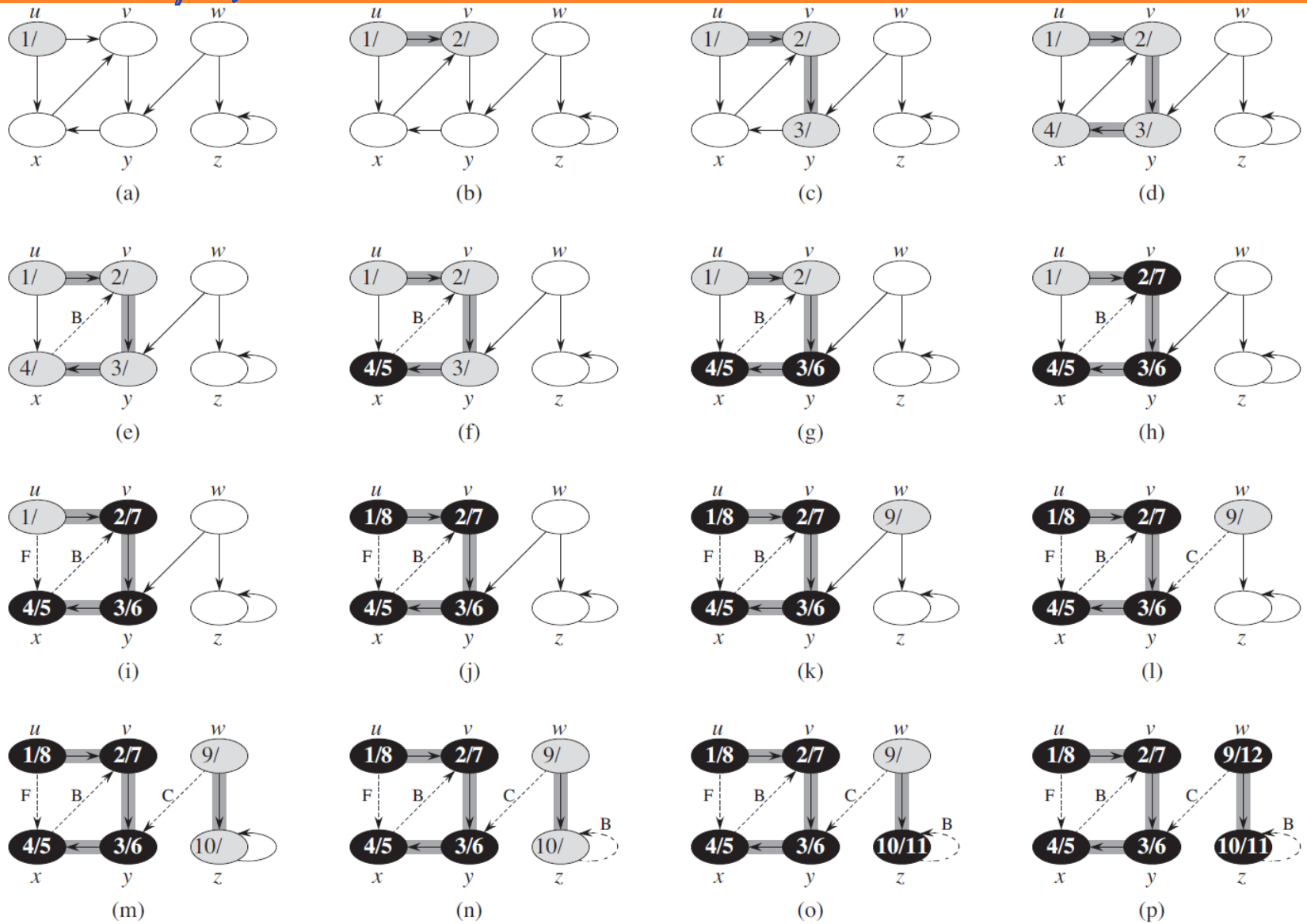
- DFS algoritam (nastavak)
  - Prilikom ulaska u DFS-VISIT proceduru, čvor u je beo, ali u liniji 3 postaje siv
  - U liniji 1 se inkrementira brojač
  - U liniji 2 se zabeleži vreme kada je postao siv
  - Zatim se pristupa listi povezanosti za čvor u
  - Ako je čvor v iz liste povezanosti čvora u beo, onda se čvor u postavlja za nadređeni čvor čvoru $v.\pi = u$ i poziva se procedura DFS-VISIT procedura za listu povezanosti čvora v
  - Kada su proučeni svi čvorovi iz liste povezanosti čvora u, čvor u postaje crn (linija 8)
  - Brojač se uveća za 1 i pamti se to vreme kao vreme završavanja obrade čvora u
  - U obe pretrage rezultati zavise od redosleda čvorova u grafu (DFS linija 5) i listi povezanosti (DFS-VISIT linija 4)

```
DFS(G)
1  for each vertex u ∈ G.V
2      u.color = WHITE
3      u.π = NIL
4  time = 0
5  for each vertex u ∈ G.V
6      if u.color == WHITE
7          DFS-VISIT(G, u)
```

```
DFS-VISIT(G, u)
 1  time = time + 1
 2  u.d = time
 3  u.color = GRAY
 4  for each v ∈ G.Adj[u]
 5      if v.color == WHITE
 6          v.π = u
 7          DFS-VISIT(G, v)
 8  u.color = BLACK
 9  time = time + 1
10  u.f = time
```

… PRETRAGA U DUBINU …

- DFS algoritam kako radi na primeru



… PRETRAGA U DUBINU …

- Analiza DFS algoritma
  - Obe ***for*** petlje u DFS algoritmu se izvršavaju tačno $\Theta(\boldsymbol{V})$ bez vremena izvršavanja DFS-VISIT procedure
  - Koristimo agregiranu tehniku određivanja kompleksnosti
  - DFS-VISIT se poziva tačno jednom za svako $\boldsymbol{v} \in \boldsymbol{V}$ (dok je čvor beo, a kako prvi poziv ove procedure farba čvor u sivo, neće se više pozvati za taj čvor)
  - Toko izvršavanja ***for*** petlje u DFS-VISIT proceduri petlja se izvršava onoliko puta koliko ima grana kojima se još nije pristupilo (jer se opet pristupa samo belim čvorovima), odavde sledi da je ukupan broj izvršavanja jednak broju grana koje treba obići $\Theta(\boldsymbol{E})$
  - Agregacijom se dobija kompleksnost $\Theta(\boldsymbol{V} + \boldsymbol{E})$ (linearno)

```
DFS(G)
1  for each vertex u ∈ G.V
2      u.color = WHITE
3      u.π = NIL
4  time = 0
5  for each vertex u ∈ G.V
6      if u.color == WHITE
7          DFS-VISIT(G, u)

DFS-VISIT(G, u)
 1  time = time + 1
 2  u.d = time
 3  u.color = GRAY
 4  for each v ∈ G.Adj[u]
 5      if v.color == WHITE
 6          v.π = u
 7          DFS-VISIT(G, v)
 8  u.color = BLACK
 9  time = time + 1
10  u.f = time
```

… PRETRAGA U DUBINU …

- Osobine DFS algoritma
  - Ima osobine koju su značajne u procesiranju grafa
  - Jedna od osnovnih je da *predecessor subgraph* formira šumu pošto struktura depth-first stabla oslikava strukturu rekurzivnih poziva DFS-VISIT procedure
  - $v.\pi = u$ samo ako je $DFS - VISIT(G, v)$ pristupio čvoru v, dok je u bio siv, tj. tokom pretraživanja liste povezanosti čvora v

```
DFS(G)
1  for each vertex u ∈ G.V
2      u.color = WHITE
3      u.π = NIL
4  time = 0
5  for each vertex u ∈ G.V
6      if u.color == WHITE
7          DFS-VISIT(G, u)

DFS-VISIT(G, u)
 1  time = time + 1
 2  u.d = time
 3  u.color = GRAY
 4  for each v ∈ G.Adj[u]
 5      if v.color == WHITE
 6          v.π = u
 7          DFS-VISIT(G, v)
 8  u.color = BLACK
 9  time = time + 1
10  u.f = time
```

… PRETRAGA U DUBINU …

- Osobine DFS algoritma (nastavak)
  - Druga važna osobina je ugnježdena struktura, tj. startno vreme i završno vreme obrade čvorova (trenutak kada neki čvor postaje siv i trenutak kada čvor postaje crn) ima ugnježdenu strukturu
  - Shodno tome, u bilo kom DFS algoritmu (i usmerenom i neusmerenom) za neki graf $\boldsymbol{G = (V, E)}$ za bilo koja dva čvora $\boldsymbol{u}$ i $\boldsymbol{v}$ važi jedno od sledeća tri stanja:
    - Intervali $[\boldsymbol{u.d, u.f}]$ i interval $[\boldsymbol{v.d, v.f}]$ se ne poklapaju (nemaju presečnih tačaka) i ni jedan od njih nije predak onom drugom u depth-first šumi
    - Interval $[\boldsymbol{u.d, u.f}]$ je u potpunosti sadržan u intervalu $[\boldsymbol{v.d, v.f}]$ i $\boldsymbol{u}$ je potomak od $\boldsymbol{v}$
    - Interval$[\boldsymbol{v.d, v.f}]$ je u potpunosti sadržan u intervalu $[\boldsymbol{u.d, u.f}]$ i $\boldsymbol{v}$ je potomak od $\boldsymbol{u}$

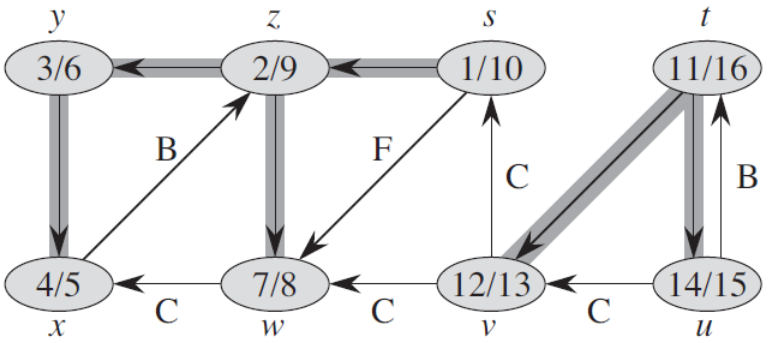
```
DFS(G)
1  for each vertex u ∈ G.V
2      u.color = WHITE
3      u.π = NIL
4  time = 0
5  for each vertex u ∈ G.V
6      if u.color == WHITE
7          DFS-VISIT(G, u)

DFS-VISIT(G, u)
 1  time = time + 1
 2  u.d = time
 3  u.color = GRAY
 4  for each v ∈ G.Adj[u]
 5      if v.color == WHITE
 6          v.π = u
 7          DFS-VISIT(G, v)
 8  u.color = BLACK
 9  time = time + 1
10  u.f = time
```

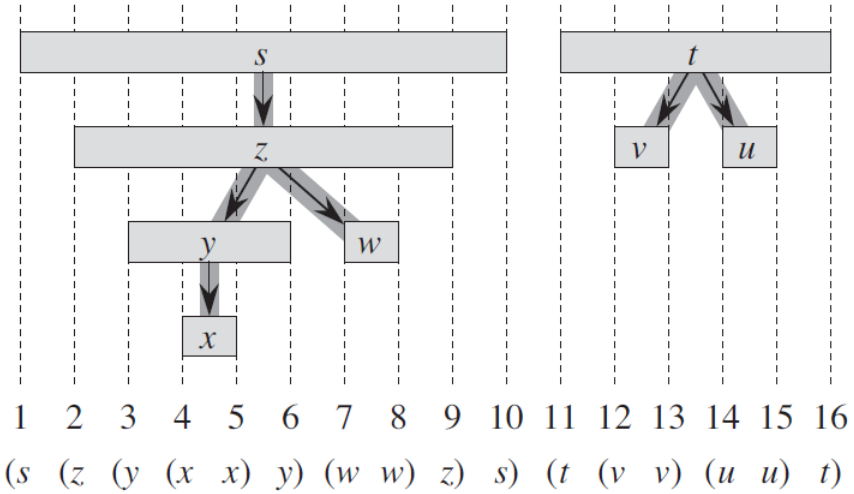
… PRETRAGA U DUBINU …

- Osobine DFS algoritma (nastavak)
  - Primer za ugnježdenu strukturu
  - Za dokaz probajte kombinacije:
    - $u.d < v.d$ (2 slučaja zavisno od toga da li je $v.d < u.f$ tačno)
    - $v.d < u.d$ (slično prethodnom samo u drugoj kombinaciji)
  - Kao rezultat prethodne teoreme sledi da je čvor v podređen čvoru u u depth-first stablu za (usmereni ili neusmereni) graf G ako i samo ako važi $u.d < v.d < v.f < u.f$

(a)



(b)



… PRETRAGA U DUBINU …

- Osobine DFS algoritma (nastavak)
  - Sledeća teorema kaže da je u depth-first šumi (usmerenog ili neusmerenog) grafa $\boldsymbol{G = (V, E)}$ čvor $\boldsymbol{v}$ potomak čvora $\boldsymbol{u}$ ako i samo ako u trenutku $\boldsymbol{u.d}$, kada je otkriven čvor $\boldsymbol{u}$, postoji putanja od $\boldsymbol{u}$ do $\boldsymbol{v}$ koji se sastoji isključivo od belih čvorova
  - Jasno da ako je $\boldsymbol{v = u}$, teorema važi (bojanje nastaje tek nakon inicijalizacije vrednosti $\boldsymbol{u.d}$)
  - A za ostale čvorove?
  - Ako je čvor $\boldsymbol{v}$ potomaka od $\boldsymbol{u}$ i nalazi se u depth-first stablu, onda mora da važi $\boldsymbol{u.d < v.d}$, pa će $\boldsymbol{v}$ biti u tom trenutku beo
  - Kako $\boldsymbol{v}$ može biti bilo koji potomak od $\boldsymbol{u}$, svi čvorovi na jedinstvenoj prostoj putanji od $\boldsymbol{u}$ do $\boldsymbol{v}$ moraju biti beli u trenutku $\boldsymbol{u.d}$

```
DFS(G)
1  for each vertex u ∈ G.V
2      u.color = WHITE
3      u.π = NIL
4  time = 0
5  for each vertex u ∈ G.V
6      if u.color == WHITE
7          DFS-VISIT(G, u)

DFS-VISIT(G, u)
 1  time = time + 1
 2  u.d = time
 3  u.color = GRAY
 4  for each v ∈ G.Adj[u]
 5      if v.color == WHITE
 6          v.π = u
 7          DFS-VISIT(G, v)
 8  u.color = BLACK
 9  time = time + 1
10  u.f = time
```

… PRETRAGA U DUBINU …

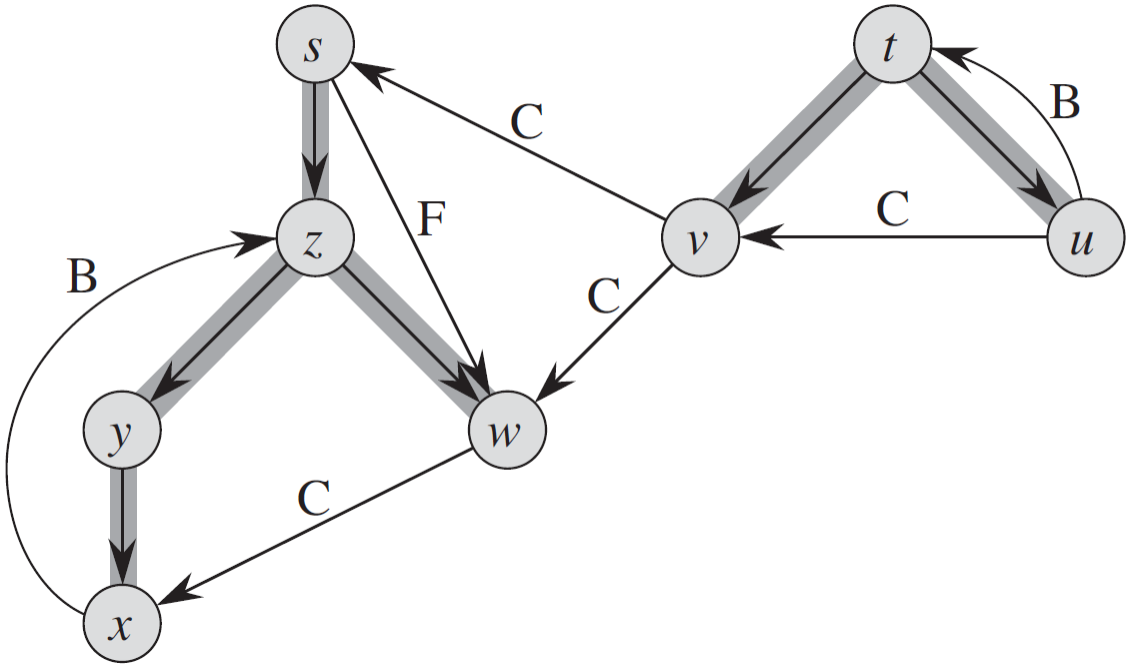
- Osobine DFS algoritma (nastavak)
  - DFS algoritam se može iskoristiti za klasifikaciju grana ulaznog grafa $\boldsymbol{G} = (\boldsymbol{V}, \boldsymbol{E})$, što može biti korisna informacija
  - Postoje četiri tipa grana spram *depth-first* šume $\boldsymbol{G_\pi}$ kreirane primenom DFS algoritma nad grafom $\boldsymbol{G}$
    - ***grana stabla*** – grana iz depth-first šume $\boldsymbol{G_\pi}$. $(\boldsymbol{u}, \boldsymbol{v})$ je stablo grana ako je čvor $\boldsymbol{v}$ otkriven u isto vreme kad i grana $(\boldsymbol{u}, \boldsymbol{v})$
    - ***povratna grana*** – grana $(\boldsymbol{u}, \boldsymbol{v})$ koje povezuje čvor $\boldsymbol{v}$ sa njemu nadređenim čvorom $\boldsymbol{u}$. Grana koja u usmerenom grafu vraća ka samom čvoru
    - ***direktna grana*** – ona grana koja nije stablo grana, ali koja povezuje nadređeni i podređeni čvor
    - ***poprečna grana*** – sve preostale grane koje ne spadaju u prethodne 3 kategorije. Npr. grana koja povezuje čvorove na istom nivou *depth-first* stabla ili između čvorova u šumi (sve dok čvorovi nisu u hijerarhiji)

```
DFS(G)
1  for each vertex u ∈ G.V
2      u.color = WHITE
3      u.π = NIL
4  time = 0
5  for each vertex u ∈ G.V
6      if u.color == WHITE
7          DFS-VISIT(G, u)

DFS-VISIT(G, u)
 1  time = time + 1
 2  u.d = time
 3  u.color = GRAY
 4  for each v ∈ G.Adj[u]
 5      if v.color == WHITE
 6          v.π = u
 7          DFS-VISIT(G, v)
 8  u.color = BLACK
 9  time = time + 1
10  u.f = time
```

… PRETRAGA U DUBINU …

- Osobine DFS algoritma (nastavak)
  - ***grana stabla – tree edge***
  - ***povratna grana – back edge***
  - ***direktna grana – forward edge***
  - ***poprečna grana – cross edge***




```
DFS(G)
1  for each vertex u ∈ G.V
2      u.color = WHITE
3      u.π = NIL
4  time = 0
5  for each vertex u ∈ G.V
6      if u.color == WHITE
7          DFS-VISIT(G, u)
```

```
DFS-VISIT(G, u)
 1  time = time + 1
 2  u.d = time
 3  u.color = GRAY
 4  for each v ∈ G.Adj[u]
 5      if v.color == WHITE
 6          v.π = u
 7          DFS-VISIT(G, v)
 8  u.color = BLACK
 9  time = time + 1
10  u.f = time
```

… PRETRAGA U DUBINU …

- Osobine DFS algoritma (nastavak)
  - DFS algoritam tokom obilaska grafa može da odredi vrstu grane
  - Kada se istražuje grana (u, v) boja čvora ukazuje na vrstu grane
    - Bela ukazuje na granu stabla
    - Siva ukazuje na povratnu granu
    - Crna ukazuje na direktnu ili poprečnu granu
  - Prvo je očigledno iz samog algoritma
  - Drugo proizilazi iz činjenice da sive formiraju lanac čvorova; siva znači da je taj čvor već posećen, što znači da se nalazi jedan nivo više
  - Treće pokriva preostale kombinacije ($u.d < v.d$ svakako)
    - Ako je $u.f > v.f$ ili $u.f$ nije još definisano, radi se o direktnoj grani
    - Ako je $u.f < v.d$, radi se o poprečnoj grani (da li može biti $u.d = v.d$?)

```
DFS(G)
1  for each vertex u ∈ G.V
2      u.color = WHITE
3      u.π = NIL
4  time = 0
5  for each vertex u ∈ G.V
6      if u.color == WHITE
7          DFS-VISIT(G, u)

DFS-VISIT(G, u)
1  time = time + 1
2  u.d = time
3  u.color = GRAY
4  for each v ∈ G.Adj[u]
5      if v.color == WHITE
6          v.π = u
7          DFS-VISIT(G, v)
8  u.color = BLACK
9  time = time + 1
10 u.f = time
```

… PRETRAGA U DUBINU

- Osobine DFS algoritma (nastavak)
  - U DFS algoritmu kojim se obilazi neusmereni graf G svaka grana je ili grana stabla ili povratna grana
  - Pretpostaviti da je $u.d < v.d$, onda algoritam mora prvo otkriti i završiti v, pre nego što završi u (dok je u siv), jer v deo liste povezanosti od u
  - Prvi put kada algoritam pristupi grani (u, v), onda ako ide iz pravca u ka v, v mora biti beo (ova grana još nije proučena), u suprotnom bi grana bila proučena iz pravca v ka u, tako grana (u, v) postaje grana stabla
  - Prvi put kada algoritam pristupi grani (u, v), ako ide iz pravca v ka u, onda je grana (u, v) povratna grana, jer je i u i v su sivi

```
DFS(G)
1  for each vertex u ∈ G.V
2      u.color = WHITE
3      u.π = NIL
4  time = 0
5  for each vertex u ∈ G.V
6      if u.color == WHITE
7          DFS-VISIT(G, u)

DFS-VISIT(G, u)
 1  time = time + 1
 2  u.d = time
 3  u.color = GRAY
 4  for each v ∈ G.Adj[u]
 5      if v.color == WHITE
 6          v.π = u
 7          DFS-VISIT(G, v)
 8  u.color = BLACK
 9  time = time + 1
10  u.f = time
```

TOPOLOŠKO SORTIRANJE

TOPOLOŠKO SORTIRANJE …

- Pretpostavimo da je zadat skup poslova na čiji redosled izvršavanja su zadata neka ograničenja (neki poslovi zavise od drugih, odnosno ne mogu se započeti pre nego što se ti drugi poslovi završe
- Sve zavisnosti su poznate, a cilj je napraviti takav redosled izvršavanja poslova koji zadovoljava sva zadata ograničenja (traži se takav raspored, da se po njemu svaki posao započinje tek kad budu završeni svi poslovi od kojih on zavisi)
- Potrebno je konstruisati efikasni algoritam za formiranje takvog rasporeda – problem se zove topološko sortiranje
- Zadatim poslovima i njihovim međuzavisnostima može se na prirodan način pridružiti graf
  - Svakom poslu pridružuje se čvor, a usmerena grana od posla u do posla v postoji ako se posao v ne može započeti pre završetka posla u
  - Jasno je da graf mora biti aciklički, (bez usmerenih ciklusa), jer se u protivnom neki poslovi nikada ne bi mogli započeti (directe acyclic graph – dag)

… TOPOLOŠKO SORTIRANJE …

- Problem koji treba rešiti je sledeći
  U zadatom usmerenom acikličkom grafu $\boldsymbol{G = (V, E)}$ sa *n* čvorova numerisati čvorove od $\boldsymbol{1}$ do $\boldsymbol{n}$, tako da ako je proizvoljan čvor $\boldsymbol{v}$ numerisan sa $\boldsymbol{k}$, onda svi čvorovi do kojih postoji usmereni put iz $\boldsymbol{v}$ imaju broj veći od $\boldsymbol{k}$
- Kod acikličnih čvorova važi:
  Umemo da numerišemo na zahtevani način čvorove svih usmerenih acikličkih grafova sa manje od n čvorova
  - Bazni slučaj jednog čvora, odnosno posla, je trivijalan
  - Posmatrajmo proizvoljni graf sa n čvorova, uklonimo jedan čvor, primenimo induktivnu hipotezu i pokušajmo da proširimo numeraciju na polazni graf
    - Imamo slobodu izbora n-tog čvora
    - Trebalo bi ga izabrati tako da ostatak posla bude što jednostavniji
    - Potrebno je numerisati čvorove. Koji čvor je najlakše numerisati?
    - Očigledno čvor koji ne zavisi od drugih, odnosno čvor sa ulaznim stepenom nula; njemu se može dodeliti broj 1 (može li to uvek?)

… TOPOLOŠKO SORTIRANJE …

- Kod acikličnih čvorova važi još i:
  Usmereni aciklički graf uvek ima čvor sa ulaznim stepenom nula
  - Ako bi svi čvorovi grafa imali pozitivne ulazne stepene, mogli bismo da krenemo iz nekog čvora "unazad" prolazeći grane u suprotnom smeru
  - Međutim, broj čvorova u grafu je konačan, pa se u tom obilasku mora u nekom trenutku naići na neki čvor po drugi put, što znači da u grafu postoji ciklus
  - To je suprotno pretpostavci da se radi o acikličkom grafu
  - Na sličan način zaključuje se da u grafu mora postojati i čvor sa izlaznim stepenom nula
- Usmereni graf $\boldsymbol{G}$ je acikličan ako i samo ako DFS algoritam od $\boldsymbol{G}$ ne generiše depth-first šumu/stablo sa povratnim granama
  - Dokaz?
  - Pa šta je ono prethodno?

… TOPOLOŠKO SORTIRANJE …

- Algoritam za topološko sortiranje:

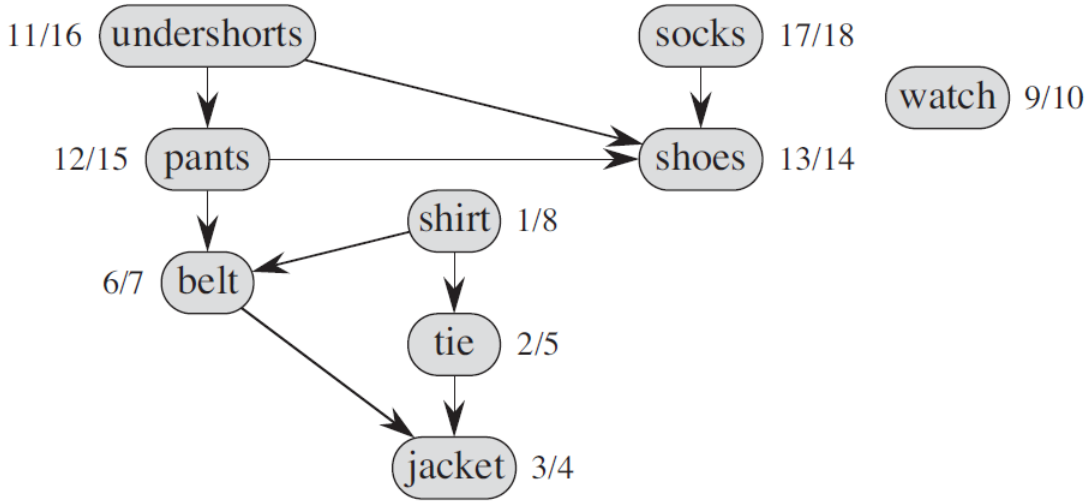
```
TOPOLOGICAL-SORT(G)
1  call DFS(G) to compute finishing times v.f for each vertex v
2  as each vertex is finished, insert it onto the front of a linked list
3  return the linked list of vertices
```

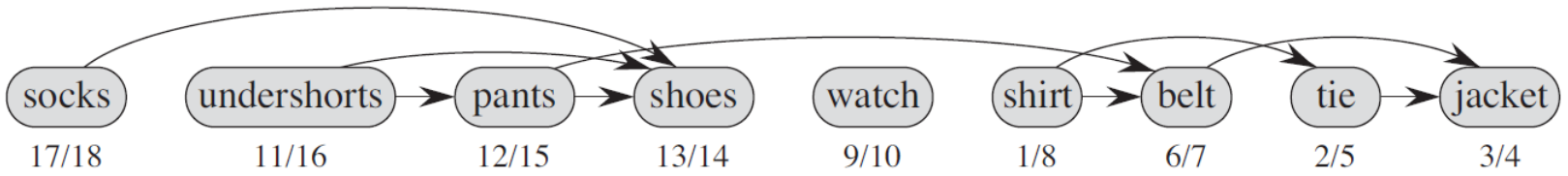
- Kako je $v.f$ jedinstveno za svaki čvor, ako nema povratnih grana, na osnovu prethodnih analiza jasno je da vreme završetka za čvor v manje od vremena svih čvorova koje mu prethode a preko kojih se može doći do njega; kako se čvorovi sa većim vremenom završetka dodaju pre njega u listu ispunjen je zahtev problema (problem rešen)

- Analiza algoritma za topološko sortiranje:
  - Kompleksnost DFS algoritma je $\Theta(V + E)$
  - Vreme potrebno da se čvor doda u listu je $O(1)$ za svako od $|V|$ čvorova
  - Tako da je ukupno vreme izvršavanja topološkog sortiranja $\Theta(V + E)$

… TOPOLOŠKO SORTIRANJE

- Primer:



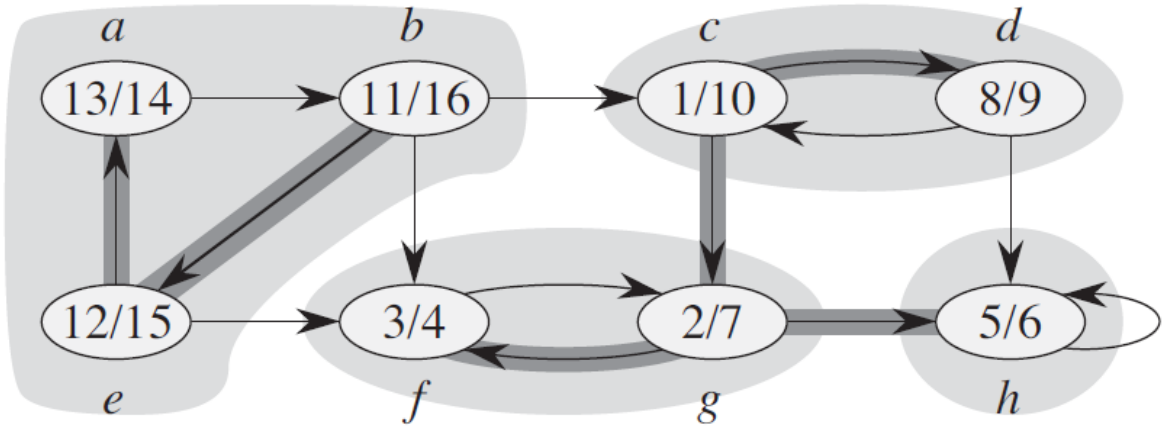


- Da li je jedinstven rezultat?

KOMPONENTE POVEZANOSTI

KOMPONENTE POVEZANOSTI …

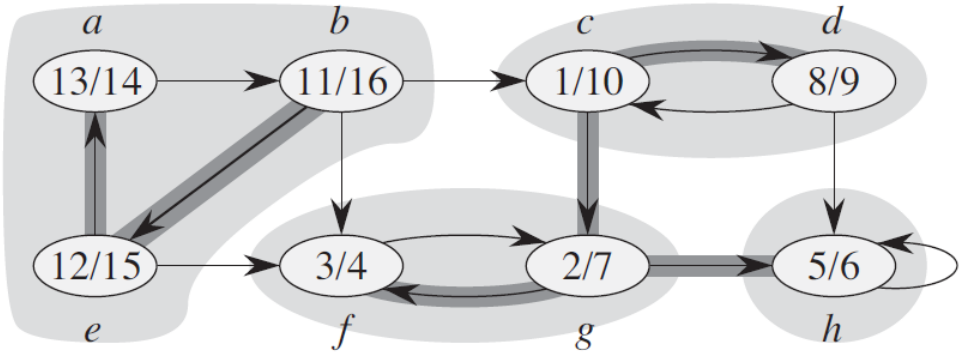
- Dekompozicija usmerenog grafa u komponente povezanosti
  - podgrafove sa tesno povezanim čvorovima
  - koji se posle odvojeno procesiraju (obrađuju), pa se rezultati spajaju u jedno rešenje
- Komponenta povezanosti usmerenog grafa $\boldsymbol{G} = (\boldsymbol{V}, \boldsymbol{E})$ je maksimalni skup čvorova $\boldsymbol{G} \subseteq \boldsymbol{V}$ tako da svaki par čvora ima usmereni put od $\boldsymbol{v}$ ka $\boldsymbol{u}$ i usmereni put od $\boldsymbol{u}$ ka $\boldsymbol{v}$
  - to znači da je $\boldsymbol{u}$ dostižno iz $\boldsymbol{v}$ i $\boldsymbol{v}$ dostižno iz $\boldsymbol{u}$



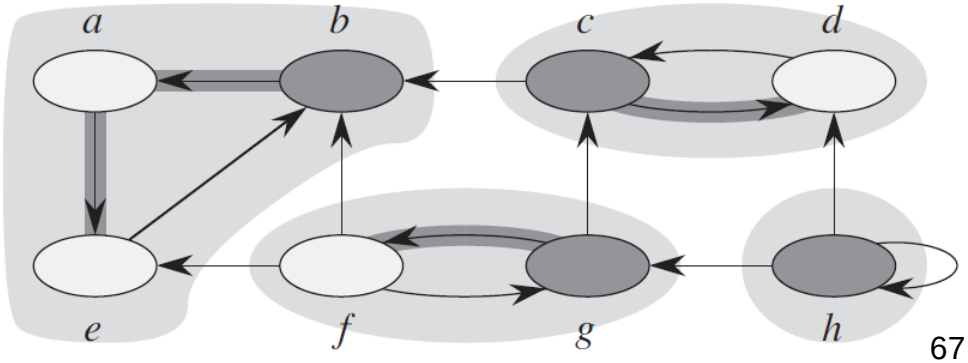
… KOMPONENTE POVEZANOSTI …

- Algoritam za pronalaženje komponenti povezanosti grafa $\boldsymbol{G} = (\boldsymbol{V}, \boldsymbol{E})$ oslanja se na transponovanje grafa $\boldsymbol{G}$:
  - $\boldsymbol{G^T} = (\boldsymbol{V}, \boldsymbol{E^T})$, gde je $\boldsymbol{E^T} = \{(\boldsymbol{u}, \boldsymbol{v}) : (\boldsymbol{v}, \boldsymbol{u}) \in \boldsymbol{E}\}$
  - $\boldsymbol{E^T}$ sadrži sve grane iz $\boldsymbol{E}$ samo sa granama obrnutog smera
- Vreme potrebno da se kreira $\boldsymbol{G^T}$ iznosi $\boldsymbol{O(V + E)}$
- Interesantno je da $\boldsymbol{G}$ i $\boldsymbol{G^T}$ imaju iste komponenti povezanosti
  - $\boldsymbol{u}$ i $\boldsymbol{v}$ su međusobno dostižni iz $\boldsymbol{G}$ ako i samo ako su međusobno dostižni i iz $\boldsymbol{G^T}$

(a)



(b)



… KOMPONENTE POVEZANOSTI …

- Algoritam za pronalaženje komponenti povezanosti grafa:

STRONGLY-CONNECTED-COMPONENTS(G)

1 call DFS(G) to compute finishing times $u.f$ for each vertex u
2 compute G^{T}
3 call DFS(G^{T}), but in the main loop of DFS, consider the vertices in order of decreasing $u.f$ (as computed in line 1)
4 output the vertices of each tree in the depth-first forest formed in line 3 as a separate strongly connected component

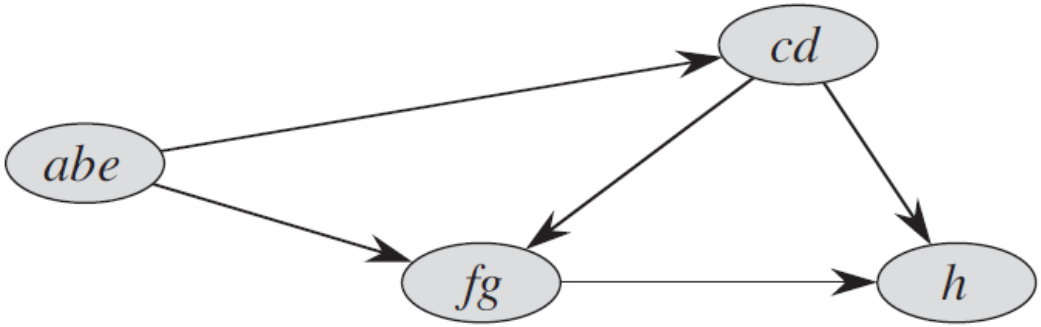
- Vreme izvršavanja ovog algoritma je $\Theta(\boldsymbol{V}+\boldsymbol{E})$
- Zašto?

DFS(G)

```
1  for each vertex u ∈ G.V
2      u.color = WHITE
3      u.π = NIL
4  time = 0
5  for each vertex u ∈ G.V
6      if u.color == WHITE
7          DFS-VISIT(G, u)
```

… KOMPONENTE POVEZANOSTI …

- Ideja algoritma dolazi iz grafa komponenti $G^{SCC} = (V^{SCC}, E^{SCC})$:
  - Neka G ima komponente povezanosti $C_1, C_2, \ldots, C_k$
  - Skup čvorova V^{SCC} sadrži čvorove $\{v_1, v_2, \ldots, v_k\}$, za svaku komponentu povezanosti C_i sadrži čvor v_i
  - Postoji grana $(v_i, v_j) \in E^{SCC}$ ako i samo ako postoji usmerena grana (x, y) koja povezuje neki čvor $x \in C_i$ sa čvorom $y \in C_j$



… KOMPONENTE POVEZANOSTI …

- Lema:
  Neka su $\boldsymbol{C}$ i $\boldsymbol{C'}$dve odvojene komponente povezanosti u usmerenom grafu $\boldsymbol{G} = (\boldsymbol{V}, \boldsymbol{E})$, neka su čvorovi $\boldsymbol{u}, \boldsymbol{v} \in \boldsymbol{C}$ i $\boldsymbol{u'}, \boldsymbol{v'} \in \boldsymbol{C'}$ i pretpostaviti da postoji grana $(\boldsymbol{v}, \boldsymbol{v'})$. Onda $\boldsymbol{G}$ ne može da sadrži i $(\boldsymbol{u}, \boldsymbol{u'})$

- Dokaz:
  Zar je zaista potreban?

… KOMPONENTE POVEZANOSTI …

- Lema:
  Neka su $\boldsymbol{C}$ i $\boldsymbol{C'}$ dve odvojene komponente povezanosti u usmerenom grafu $\boldsymbol{G} = (\boldsymbol{V}, \boldsymbol{E})$. Pretpostaviti da postoji grana $(\boldsymbol{u}, \boldsymbol{v}) \in \boldsymbol{E}$, gde je $\boldsymbol{u} \in \boldsymbol{C}$ i $\boldsymbol{v} \in \boldsymbol{C'}$, onda je $\boldsymbol{f}(\boldsymbol{C}) > \boldsymbol{f}(\boldsymbol{C'})$
  - $\boldsymbol{f}(\boldsymbol{C}) = \max\limits_{\boldsymbol{c} \in \boldsymbol{C}} \{\boldsymbol{c}.\boldsymbol{f}\}$ i $\boldsymbol{f}(\boldsymbol{C'}) = \max\limits_{\boldsymbol{c'} \in \boldsymbol{C'}} \{\boldsymbol{c'}.\boldsymbol{f}\}$
  - $\boldsymbol{d}(\boldsymbol{C}) = \min\limits_{\boldsymbol{c} \in \boldsymbol{C}} \{\boldsymbol{c}.\boldsymbol{f}\}$ i $\boldsymbol{d}(\boldsymbol{C'}) = \min\limits_{\boldsymbol{c'} \in \boldsymbol{C'}} \{\boldsymbol{c'}.\boldsymbol{f}\}$
  - Posmatra se vreme prvog DFS algoritma iz SCC algoritma
- Dokaz:
  Konsultovati knjigu

… KOMPONENTE POVEZANOSTI

- Posledica:
  Neka su $\boldsymbol{C}$ i $\boldsymbol{C'}$ dve odvojene komponente povezanosti u usmerenom grafu $\boldsymbol{G} = (\boldsymbol{V}, \boldsymbol{E})$. Pretpostaviti da postoji grana $(\boldsymbol{u}, \boldsymbol{v}) \in \boldsymbol{E^T}$, gde je $\boldsymbol{u} \in \boldsymbol{C}$ i $\boldsymbol{v} \in \boldsymbol{C'}$, onda je $\boldsymbol{f}(\boldsymbol{C}) < \boldsymbol{f}(\boldsymbol{C'})$

- Dokaz:
  Proizilazi iz prethodnog

- Sve ovo ukazuje da je algoritam korektan

MINIMALNO POVEZUJUĆE STABLO

MINIMALNO POVEZUJUĆE STABLO …

- Razmotrimo sistem računara koje treba povezati optičkim kablovima
  - Potrebno je obezbediti da postoji veza između svaka dva računara
  - Poznati su troškovi postavljanja kabla između svaka dva računara
  - Cilj je projektovati mrežu optičkih kablova tako da cena mreže bude minimalna
- Sistem računara može biti predstavljen grafom čiji čvorovi odgovaraju računarima, a grane – potencijalnim vezama između računara, sa odgovarajućom (pozitivnom) cenom
- Problem je pronaći povezani podgraf (sa granama koje odgovaraju postavljenim optičkim kablovima), koji sadrži sve čvorove, takav da mu suma cena grana bude minimalna

… MINIMALNO POVEZUJUĆE STABLO

- Problem se može modelovati na sledeći način
  - $\boldsymbol{G} = (\boldsymbol{V}, \boldsymbol{E})$ je graf koji modeluje moguće veze između računara
  - Vezu između dva čvora (dva računara) $\boldsymbol{u} \in \boldsymbol{G}$ i $\boldsymbol{v} \in \boldsymbol{G}$ predstavlja grana $(\boldsymbol{u}, \boldsymbol{v}) \in \boldsymbol{E}$
  - Težina $\boldsymbol{w}(\boldsymbol{u}, \boldsymbol{v})$ predstavlja cenu povezivanja računara $\boldsymbol{u}$ i $\boldsymbol{v}$
  - Cilj je pronaći takav aciklični podskup $\boldsymbol{T} \subseteq \boldsymbol{E}$ koji povezuje sve čvorove i čija je ukupna težina $\boldsymbol{W}(\boldsymbol{T}) = \sum_{(\boldsymbol{u},\boldsymbol{v})\in \boldsymbol{T}} \boldsymbol{w}(\boldsymbol{u}, \boldsymbol{v})$ minimalna
  - Pošto je $\boldsymbol{T}$ acikličan i povezuje sve čvorove ono je u formi stabla
  - Ovo stablo se zove povezujuće (*spanning*), jer povezuje sve čvorove
  - Sam problem se naziva problem minimalno povezujućeg stabla *(minimum spanning tree problem, ili minimum-weight spanning tree problem ili minimum-cost spanning tree problem*)
  - Skraćeno ćemo minimalno povezujuće stablo zvati MST

MST GENERIČKI ALGORITAM …

- Pretpostavimo da imamo povezan, neusmeren graf $\boldsymbol{G} = (\boldsymbol{V}, \boldsymbol{E})$ i da postoji funkcija težine $\boldsymbol{w}: \boldsymbol{E} \rightarrow \boldsymbol{R}$ i da želimo da pronađemo MST za $\boldsymbol{G}$
- Koristi se pohlepni pristup
- Generički algoritam održava skup grana $\boldsymbol{A}$ i održava sledeću invarijantu petlje
  - Pre svake iteracije, $\boldsymbol{A}$ je podskup nekog minimalnog povezujućeg stabla
- U svakom koraku se evaluira da li se grana $(\boldsymbol{u}, \boldsymbol{v})$ može dodati u $\boldsymbol{A}$ a da se ne pokvari invarijantnost petlje, u smislu da je i $\boldsymbol{A} \cup \{(\boldsymbol{u}, \boldsymbol{v})\}$ podskup nekog minimalnog povezujućeg stabla
- Grana $(\boldsymbol{u}, \boldsymbol{v})$ sa takvom osobinom naziva se sigurnom granom za $\boldsymbol{A}$ zato što se sa sigurnošću može dodati u $\boldsymbol{A}$ i pri tome očuvati invarijantnost petlje

… MST GENERIČKI ALGORITAM …

- Generički MST algoritam:

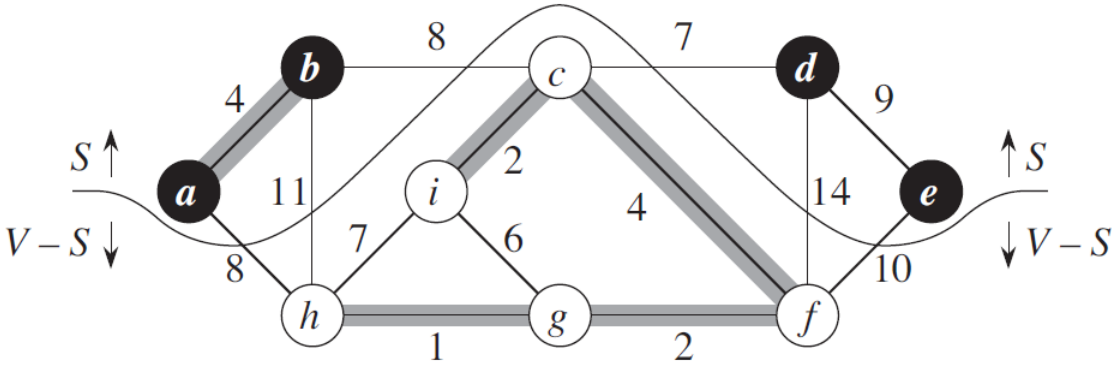
```
GENERIC-MST(G, w)
1  A = ∅
2  while A does not form a spanning tree
3       find an edge (u, v) that is safe for A
4       A = A ∪ {(u, v)}
5  return A
```

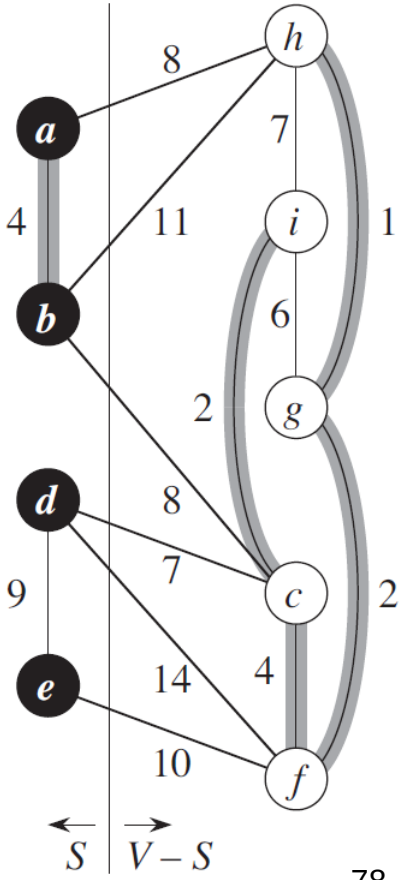
- Invarijantnost petlje se koristi na sledeći način:
  - **Inicijalizacija:** U liniji 1 skup A na trivijalan način zadovoljava invarijantnost
  - **Održivost:** Dodaju se samo sigurne grane, pa se invarijantnost održava
  - **Završavanje:** Svi čvorovi dodati u A su u minimalnom povezujućem stablu, tako da i skup A u liniji 5 mora biti minimalno povezujuće stablo
- Suština je pronaći sigurnu granu u liniji 3

… MST GENERIČKI ALGORITAM …

- Teorema:
  Neka je $G = (V, E)$ povezani, neusmereni graf sa funkcijom težine w definisanom nad E. Neka je A podskup od E i neka je A deo nekog minimalnog povezujućeg stabla od G. Neka je $(S, V - S)$ isečak iz G koji respektuje A, i neka je (u, v) laka grana koja prelazi $(S, V - S)$. U tom slučaju je (u, v) i siguran za A.

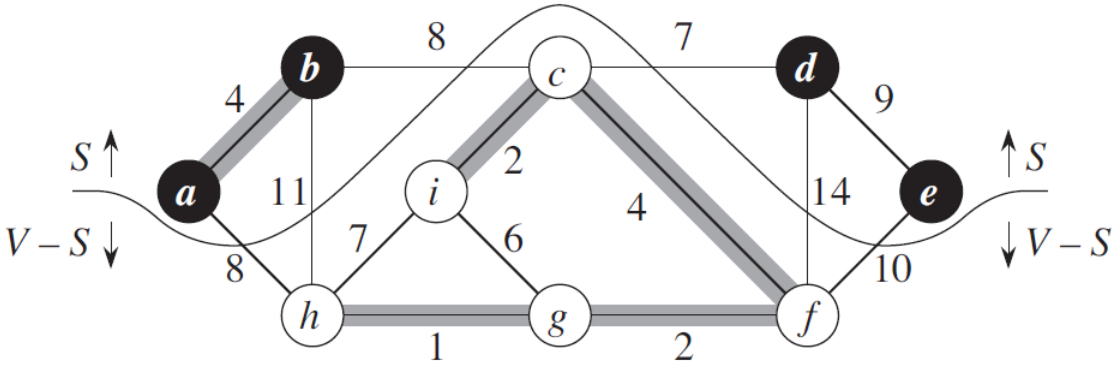
- Isečak $(S, V - S)$ iz grafa $G = (V, E)$ predstavlja deljenje skupa V na dva skupa čvorova S i $V - S$

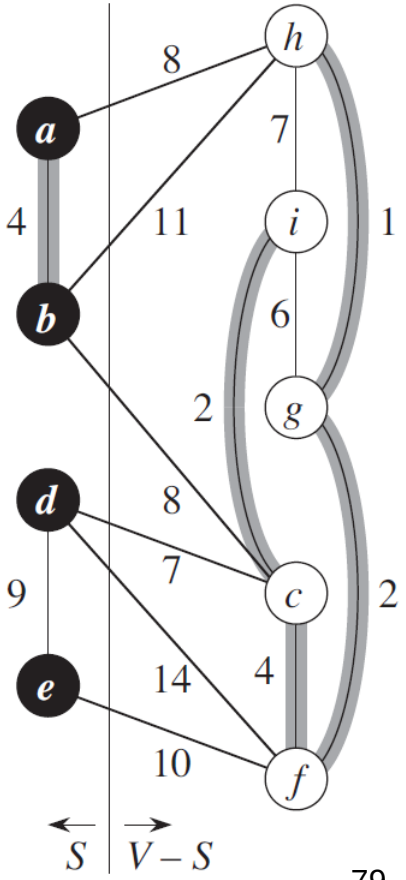




… MST GENERIČKI ALGORITAM …

- Grana $(u, v) \in E$ preseca isečak $(S, V - S)$ ako je jedan kraj grane u S a drugi u $V - S$
- Isečak respektuje skup grana A ako ni jedna grana iz A ne preseca isečak
- Grana je lagana grana ako prelazi preko isečka i ako je njena težina najmanja težina od svih grana koje prelaze preko isečka
- Može postojati više laganih grana





… MST GENERIČKI ALGORITAM

- Posledica:
  Neka je $G = (V, E)$ povezani, neusmereni graf sa funkcijom težine w definisanom nad E. Neka je A podskup od E i neka je A deo nekog minimalnog povezujućeg stabla od G. Neka je $C = (V_C, E_C)$ komponenta povezanosti u nekoj šumi $G_A = (V, A)$. Ako je (u, v) lagana grana koja povezuje C sa nekom drugom komponentom iz G_A onda je (u, v) siguran za A

- Dva algoritma za određivanje minimalnog povezujućeg stabla:
  - Kruskalov algoritma
  - Primov algoritam
  - Razlikuju se na osnovu pravila koje se koristi za određivanje sigurnih (lakih) grana u liniji 3

```
GENERIC-MST(G, w)
1  A = ∅
2  while A does not form a spanning tree
3      find an edge (u, v) that is safe for A
4      A = A ∪ {(u, v)}
5  return A
```

KRUSKALOV ALGORITAM …

- Kruskalov algoritam pronalazi sigurnu granu koju dodaje u rastuću šumu, tako što posmatra sve grane koje spajaju bilo koja dva stabla u šumi i uzima onu granu (u, v) koja ima najmanju težinu
- Pretpostaviti da imamo stabla C_1 i C_2 povezani granom (u, v), kako (u, v) mora biti lagana grana koja povezuje C_1 sa nekim drugim stablom, onda (u, v) mora biti i sigurna grana za C_1
- Kruskalov algoritam je pohlepan jer uvek dodaje u šumu granu sa najmanjom težinom
- Oslanja se na 3 procedure za rad sa disjunktnim skupovima čvorova:
  - ***MAKE-SET(x)*** – kreira novi skup čiji jedini član i predstavnik je ***x***
  - ***FIND-SET(x)*** – vraća pokazivač na predstavnika jedinstvenog skupa kojem pripada ***x***
  - ***UNION(x,y)*** – ujedinjuje dva disjunktna skupa S_x i S_y koji sadrže ***x*** i ***y***. Predstavnik skupa $S_x \cup S_y$ je bilo koji član novog skupa (može i da se odredi tačno jedan predstavnik prilikom spajanja), S_x i S_y se brišu

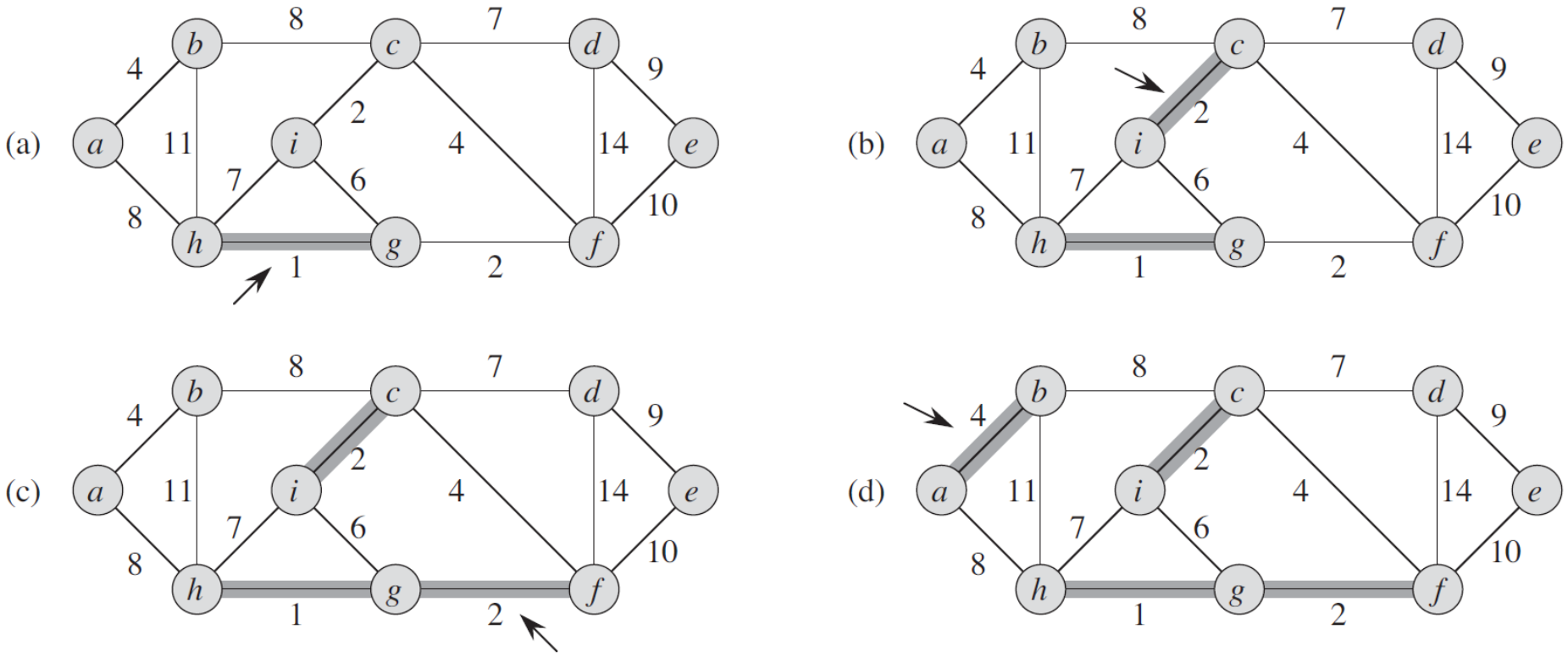
… KRUSKALOV ALGORITAM …

- Kruskalov algoritam

```
MST-KRUSKAL(G, w)
1  A = ∅
2  for each vertex v ∈ G.V
3       MAKE-SET(v)
4  sort the edges of G.E into nondecreasing order by weight w
5  for each edge (u, v) ∈ G.E, taken in nondecreasing order by weight
6       if FIND-SET(u) ≠ FIND-SET(v)
7            A = A ∪ {(u, v)}
8            UNION(u, v)
9  return A
```

… KRUSKALOV ALGORITAM …

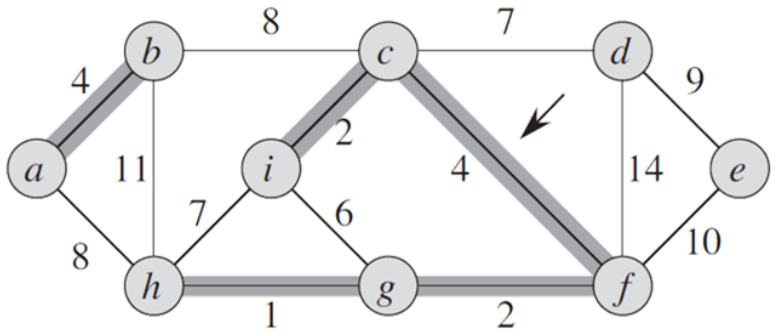
- Kako Kruskalov algoritam radi:



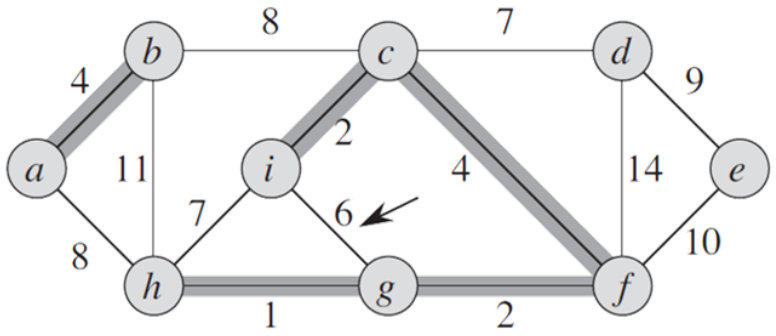
… KRUSKALOV ALGORITAM …

- Kako Kruskalov algoritam radi (nastavak):

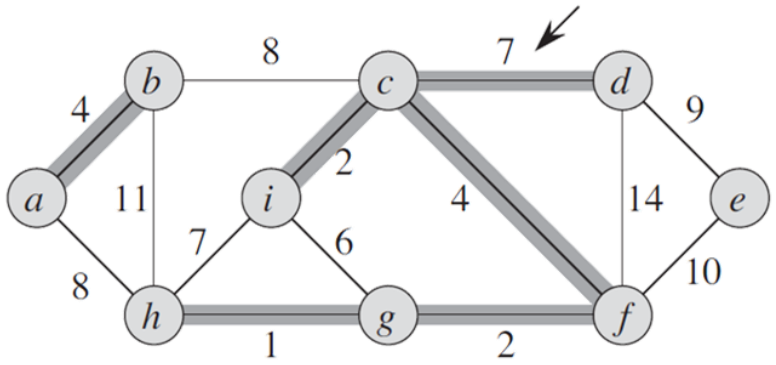
(e)



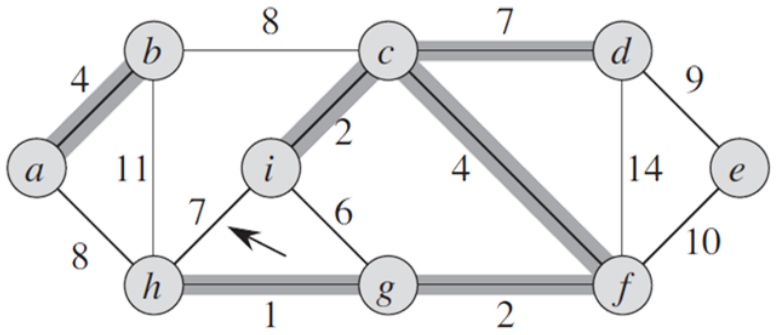
(f)



(g)



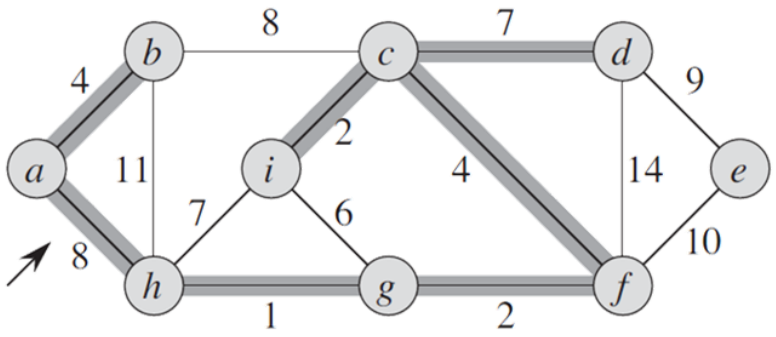
(h)



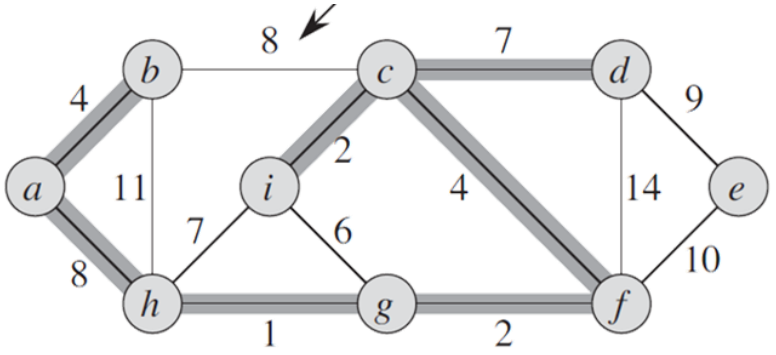
… KRUSKALOV ALGORITAM …

- Kako Kruskalov algoritam radi (nastavak):

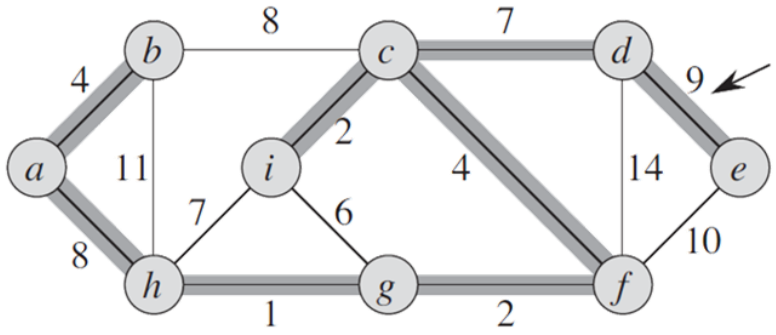
(i)



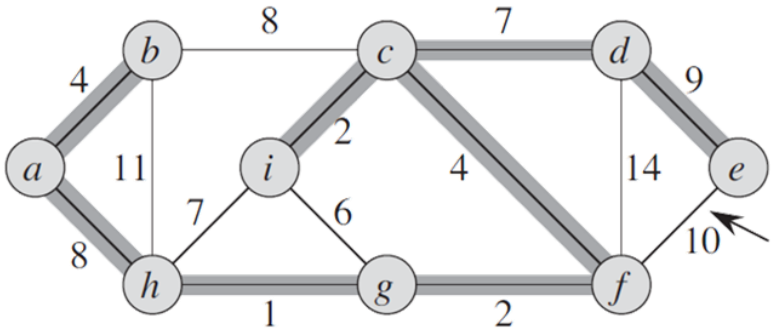
(j)



(k)



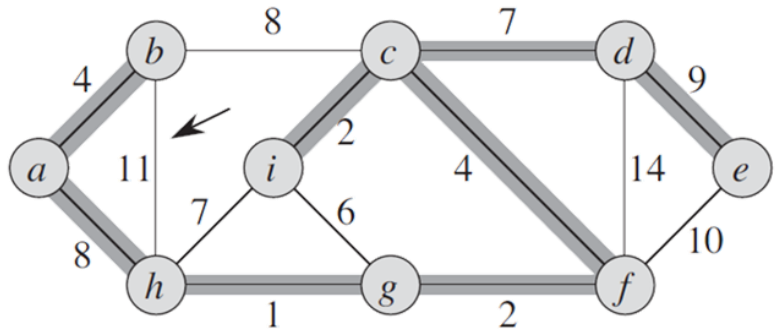
(l)



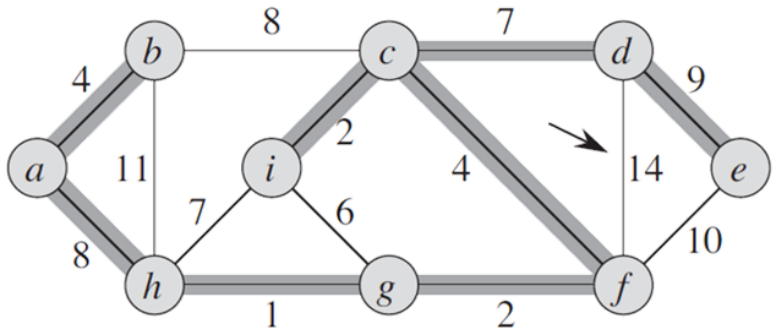
… KRUSKALOV ALGORITAM …

- Kako Kruskalov algoritam radi (nastavak):

(m)



(n)



… KRUSKALOV ALGORITAM …

- Kako Kruskalov algoritam radi (nastavak):
  - Skup A će sadržati grane (to je minimalno povezujuće stablo), u startu je prazan, linija 1
  - Prvo napravi $|V|$ disjunktnih stabla (svaki čvor za sebe), linija 2–3
  - U liniji 4 grane se sortiraju u rastućem redosledu njihovih težina
  - Zatim se ide po granama (linija 5–8) i proverava se da li su čvorovi na njihovim krajevima deo istog stabla (linija 6)
  - Ako nisu deo istog stabla, znači da grana prelazi preko isečka, da je najmanje težine, samim tim u pitanju je lagana i sigurna grana, te je sigurno dodati je u minimalno povezujuće stablo A (linija 7)
  - Skupovi čvorova kojima pripada grana se ujedinjuju kako bi se na taj način stvorilo povezujuće stablo (i označilo da su dati čvorovi već deo puta)

```
MST-KRUSKAL(G, w)
1  A = ∅
2  for each vertex v ∈ G.V
3      MAKE-SET(v)
4  sort the edges of G.E into nondecreasing order by weight w
5  for each edge (u, v) ∈ G.E, taken in nondecreasing order by weight
6      if FIND-SET(u) ≠ FIND-SET(v)
7          A = A ∪ {(u, v)}
8          UNION(u, v)
9  return A
```

… KRUSKALOV ALGORITAM

- Kruskalov algoritam analiza:
  - Vreme izvršavanja algoritma nad grafom $\boldsymbol{G = (V, E)}$ zavisi od toga kako je implementirana struktura podataka koja odgovara disjunktnim skupovima

```
MST-KRUSKAL(G, w)
1  A = ∅
2  for each vertex v ∈ G.V
3       MAKE-SET(v)
4  sort the edges of G.E into nondecreasing order by weight w
5  for each edge (u, v) ∈ G.E, taken in nondecreasing order by weight
6       if FIND-SET(u) ≠ FIND-SET(v)
7            A = A ∪ {(u, v)}
8            UNION(u, v)
9  return A
```

  - Inicijalizacija u liniji 1 traje $\boldsymbol{O(1)}$, sortiranje u liniji 4 traje $\boldsymbol{O(E \log E)}$
  - ***for*** petlja u liniji 5–8 traje $\boldsymbol{O(E)}$ ***FIND-SET*** i ***UNION*** operacija, zajedno sa $\boldsymbol{|V|}$ ***MAKE-SET*** to traje $\boldsymbol{O((E + V)\alpha(V))}$ vremena, gde je $\boldsymbol{\alpha}$ sporo-rastuća funkcija definisana u knjizi
  - Kako je $\boldsymbol{G}$ povezan graf, $\boldsymbol{|E| \geq |V| - 1}$, tako da operacije sa disjunktnim skupovima traju $\boldsymbol{O(E\alpha(V))}$
  - Takođe, $\boldsymbol{\alpha(|V|) = O(\log V) = O(\log E)}$, pa se dobija da je vreme izvršavanja MST-KRUSKAL algoritma $\boldsymbol{O(E \log E)}$,
  - kako je $\boldsymbol{|E| < |V|^2}$, sledi $\boldsymbol{\log|E| = O(\log V)}$, pa je: $\boldsymbol{O(E \log V)}$

PRIMOV ALGORITAM …

- Primov algoritam radi po principu da grane u skupu A uvek formiraju jedno stablo
- Stablo počinje u jednom proizvoljnom čvoru r i postepeno raste dok se ne dosegnu svi čvorovi u V
- Svaki korak dodaje u A laganu granu koja povezuje A sa izolovanim čvorom, jer je ta grana ujedno i sigurna grana za A
- Kako se prilikom obilaska grafa dodaju samo sigurne grane, na kraju algoritma je A zapravo minimalno povezujuće stablo
- Ovaj pristup je pohlepan, jer svaki korak dodaje u stablo granu koja za najmanju vrednost povećava ukupnu težinu stabla
- Ulaz algoritma su graf G i čvor r iz kojeg se gradi stablo
- Sam algoritam se oslanja na minimalni red prioriteta

… PRIMOV ALGORITAM …

- Svi čvorovi koji nisu u stablu nalaze se u minimalnom redu prioriteta $\boldsymbol{Q}$
- Sortirani su prema ključu $\boldsymbol{key}$, gde za svaki čvor $\boldsymbol{v}$ ključ ima vrednost koja odgovara najmanjoj (minimalnoj) težini neke grane koja povezuje $\boldsymbol{v}$ sa nekim čvorom iz stabla $\boldsymbol{A}$ (od svih grana bira se ona koja ima najmanju težinu)
- Ako čvor koji nije u stablu nema vezu ni prema jednom čvoru iz stabla, onda je njegov ključ $\boldsymbol{v.key} = \infty$
- Takođe, svaki čvor ima atribut $\boldsymbol{v.\pi}$ koji sadrži link ka nadređenom čvoru od $\boldsymbol{v}$
- Skup $\boldsymbol{A}$ se održava na sledeći način:

  $\boldsymbol{A = \{(v, v.\pi): v \epsilon V - \{r\} - Q\}}$

- Kada se algoritam završi, skup $\boldsymbol{A}$ će sadržati sledeće grane:

  $\boldsymbol{A = \{(v, v.\pi): v \epsilon V - \{r\}\}}$

… PRIMOV ALGORITAM …

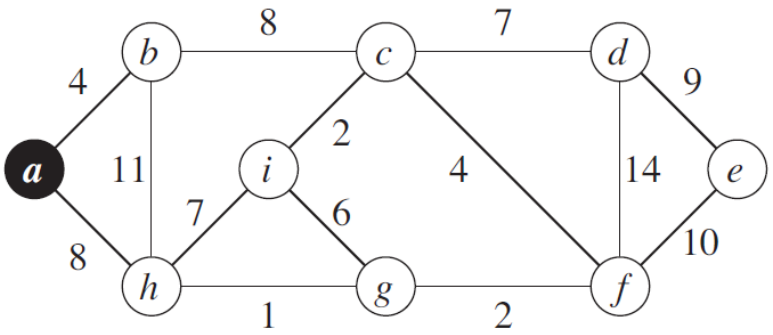
- Primov algoritam:

```
MST-PRIM(G, w, r)
 1  for each u ∈ G.V
 2      u.key = ∞
 3      u.π = NIL
 4  r.key = 0
 5  Q = G.V
 6  while Q ≠ ∅
 7      u = EXTRACT-MIN(Q)
 8      for each v ∈ G.Adj[u]
 9          if v ∈ Q and w(u, v) < v.key
10              v.π = u
11              v.key = w(u, v)
```

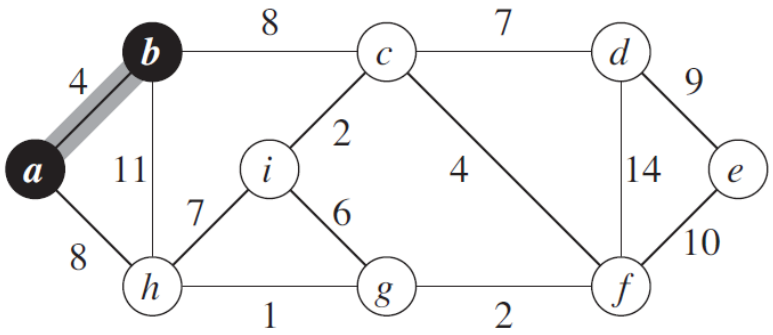
… PRIMOV ALGORITAM …

- Kako Primov algoritam radi:

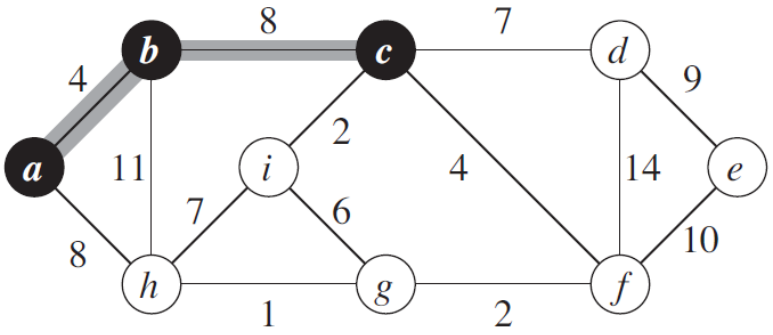
(a)



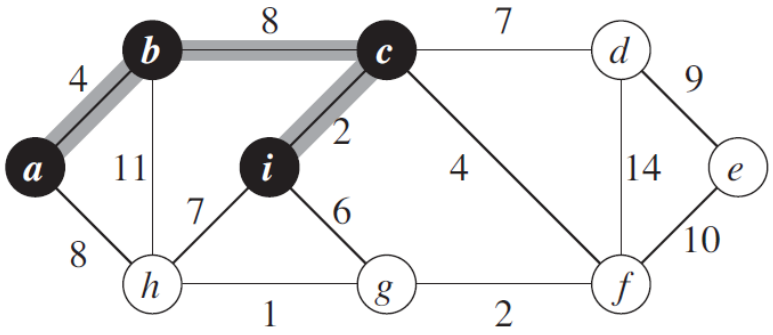
(b)



(c)



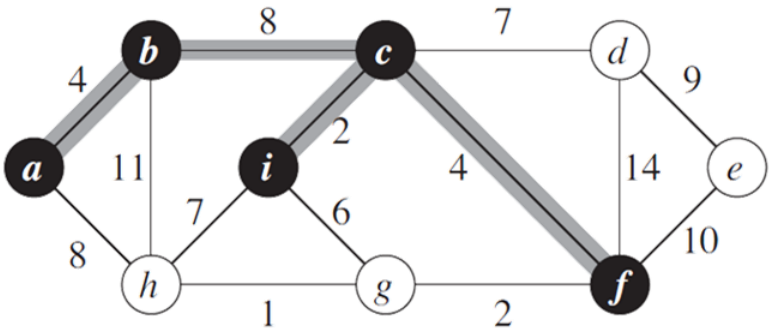
(d)



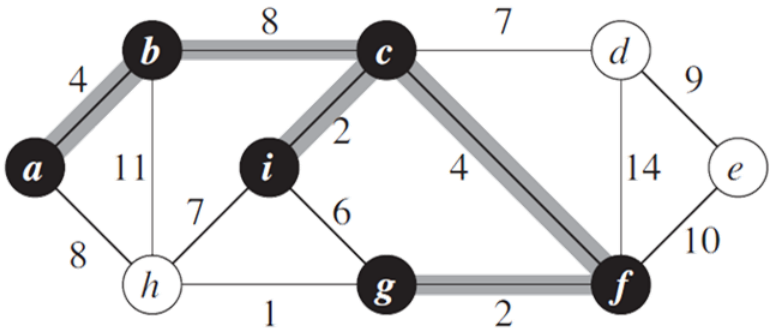
… PRIMOV ALGORITAM …

- Kako Primov algoritam radi (nastavak):

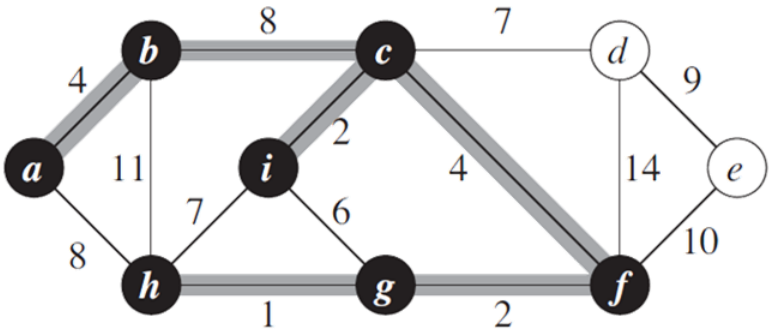
(e)



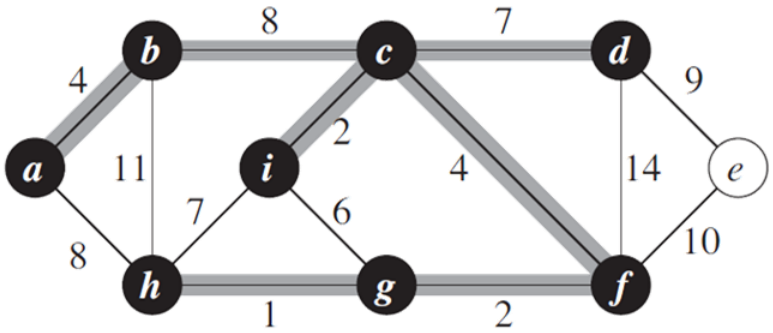
(f)



(g)



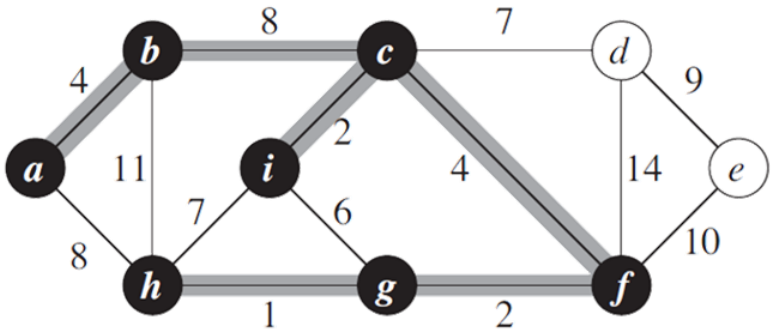
(h)



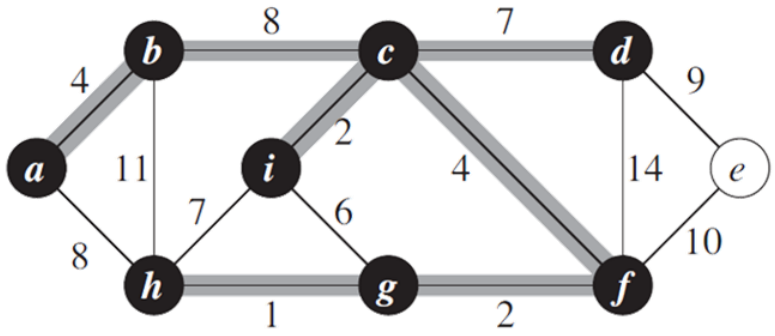
… PRIMOV ALGORITAM …

- Kako Primov algoritam radi (nastavak):

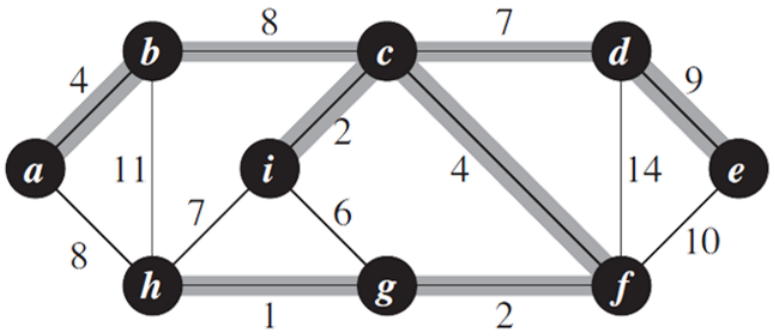
(g)



(h)



(i)



… PRIMOV ALGORITAM …

- Kako Primov algoritam radi (nastavak):
  - Linije 1–5 služe za inicijalizaciju čvorova i minimalnog reda prioriteta
  - Algoritam radi i održava sledeću invarijentnost petlje:
    - $\boldsymbol{A = \{(v, v.\pi): v \epsilon V - \{r\} - Q\}}$
    - Čvorovi koji se nalaze u MST stablu su čvorovi $\boldsymbol{V - Q}$
    - Za sve čvorove $\boldsymbol{v \epsilon Q}$ za koje važi $\boldsymbol{v.\pi \neq NIL}$ onda važi i $\boldsymbol{v.key} < \infty$ i $\boldsymbol{v.key}$ je težina laganog čvora $(\boldsymbol{v, v.\pi})$ koji povezuje $\boldsymbol{v}$ sa nekim čvorom iz $\boldsymbol{A}$
  - U liniji 7 preuzima se sledeći čvor iz reda prioriteta i on se iz $\boldsymbol{Q}$ prebacuje u $\boldsymbol{V - Q}$
  - U linijama 9–11 identifikuje se koja grana prelazi isečak $(\boldsymbol{V - Q, Q})$, izuzev u početnom slučaju kada je $\boldsymbol{u = r}$ i ažuriraju se $\boldsymbol{v.key}$ i $\boldsymbol{v.\pi}$ za sve čvorova povezane sa $\boldsymbol{u}$ a koji nisu u MST stablu

```
MST-PRIM(G, w, r)
 1  for each u ∈ G.V
 2      u.key = ∞
 3      u.π = NIL
 4  r.key = 0
 5  Q = G.V
 6  while Q ≠ ∅
 7      u = EXTRACT-MIN(Q)
 8      for each v ∈ G.Adj[u]
 9          if v ∈ Q and w(u, v) < v.key
10              v.π = u
11              v.key = w(u, v)
```

… PRIMOV ALGORITAM

- Primov algoritam analiza :
  - Vreme izvršavanja algoritma nad grafom $\boldsymbol{G = (V, E)}$ zavisi od toga kako je implementiran minimalni red prioriteta
  - Ako je to red zasnovan na heap strukturi, onda se linije 1–5 izvršavaju u $\boldsymbol{O(V)}$ vremena
  - Telo ***while*** petlje se izvršava $|\boldsymbol{V}|$ puta, a izvlačenje elementa iz reda traje $\boldsymbol{O(\log V)}$, tako da to traje ukupno $\boldsymbol{O(V \log V)}$
  - ***for*** petlja u linijama 8–11 se izvršava $\boldsymbol{O(E)}$ vremena
  - Linija 11 u sebi uključuje implicitno dekrementiranje ključa u redu minimalnog prioriteta što traje $\boldsymbol{O(\log V)}$
  - Tako se dolazi do toga da je ukupno vreme izvršavanja Primovog algoritma: $\boldsymbol{O(V \log V + E \log V) = O(E \log V)}$

```
MST-PRIM(G, w, r)
 1  for each u ∈ G.V
 2      u.key = ∞
 3      u.π = NIL
 4  r.key = 0
 5  Q = G.V
 6  while Q ≠ ∅
 7      u = EXTRACT-MIN(Q)
 8      for each v ∈ G.Adj[u]
 9          if v ∈ Q and w(u, v) < v.key
10              v.π = u
11              v.key = w(u, v)
```

NAJKRAĆI PUTEVI IZ ZADATOG ČVORA

NAJKRAĆI PUTEVI IZ ZADATOG ČVORA …

- Grafovi i grafovski algoritmi idealni su za rešavanje određenih životnih problema
- Jadan takav problem je kako doći iz tačke A do tačke B najkraćim mogućim putem i/ili za najkraće moguće vreme
- Kada imate takav problem jedno rešenje je da proučite sve moguće puteve između dve tačke, ali to nije nimalo efikasno
- Rešenje leži u grafovima, putni pravci mogu se opisati granama, a mesta na putu čvorovima
- Najbolje je koristi težinske grafove
- Neka je $\boldsymbol{G} = (\boldsymbol{V}, \boldsymbol{E})$ graf koji modeluje moguće veze između gradova i neka je svakoj grani pridružene težina (dužina puta između 2 grada, tj. čvora) opisana funkcijom težine $\boldsymbol{w}$: $\boldsymbol{E} \rightarrow \boldsymbol{R}$ koja mapira stvarne dužine puteva na težine grana

… NAJKRAĆI PUTEVI IZ ZADATOG ČVORA …

- Ukupna dužina nekog puta $\boldsymbol{w(p)}$ između 2 čvora koja obuhvata putanju $\boldsymbol{p = \langle v_0, v_1, \dots, v_k \rangle}$ i koja prolazi kroz čvorove $\boldsymbol{v_0, v_1, \dots, v_k}$
- Ova putanja obuhvata sumu težina svih grana na putanji od $\boldsymbol{v_0}$ do $\boldsymbol{v_k}$:

$$\boldsymbol{w(p) = \sum_{i=1}^{k} w(v_{i-1}, v_i)}$$

- Težinu najkraćeg puta $\boldsymbol{\delta(u, v)}$ od čvora $\boldsymbol{u}$ do čvor $\boldsymbol{v}$ definišemo:

$$\boldsymbol{\delta(u,v)} = \begin{cases} \min\left\{\boldsymbol{w(p)} : \boldsymbol{u \xrightarrow{p} v}\right\}, & \boldsymbol{ako\ postoji\ put\ izmedju\ u\ i\ v} \\ \infty, & \boldsymbol{ako\ ne\ postoji\ put\ izmedju\ u\ i\ v} \end{cases}$$

- Pod najkraćim putem između čvorova $\boldsymbol{u}$ i $\boldsymbol{v}$ smatra se bilo koji put $\boldsymbol{p}$ između njih sa težinom $\boldsymbol{w(p) = \delta(u, v)}$
- Težina može biti bilo šta, dužina puta, cena puta, potrošnja goriva

… NAJKRAĆI PUTEVI IZ ZADATOG ČVORA …

- BFS algoritam je po svojoj prirodi algoritam koji generiše najkraći put za ne-težinske grafove, tj. grafove kod kojih je težina 1 na svim granama
- Veliki broj koncepta vezanih za BFS algoritama se koriste i prilikom određivanja najkraćeg puta
- Postoji veliki broj varijanti ovog problema, ovde se koncentrišemo na situaciju kada za graf $\boldsymbol{G} = (\boldsymbol{V}, \boldsymbol{E})$ želimo da pronađemo najkraće puteve iz čvora $\mathbf{s} \in \boldsymbol{V}$ do svih preostalih čvorova iz $\boldsymbol{V}$
- Ovo može da reši gomilu drugih problema
  - Najkraći putevi iz svih čvorova do jednog čvora (**Single-destination shortest-paths problem)**
  - Najkraći put od čvora $\mathbf{u}$ do čvora $\mathbf{v}$ za čvorove $\mathbf{u}$ i $\mathbf{v}$ (**Single-pair shortest-path problem**)
  - Najkraći put čvora $\mathbf{u}$ do čvora $\mathbf{v}$ za sve parove čvorova iz $\boldsymbol{V}$ (**All-pairs shortest-paths problem**)

OPTIMALNA PODSTRUKTURA

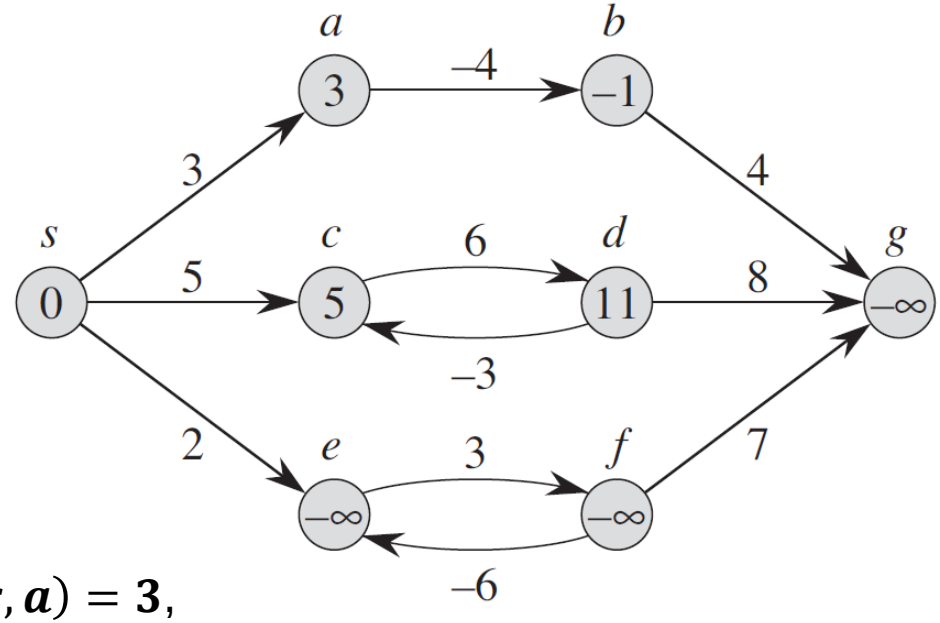
- Algoritmi za računanje najkraćeg puta između dva čvora, oslanjaju se na osobinu da najkraći put u sebi sadrži druge najkraće puteve
- Lema:
  Neka je $\boldsymbol{G = (V, E)}$ težinski, usmereni graf sa funkcijom težine $\boldsymbol{w: E \to R}$. Neka je $\boldsymbol{p = \langle v_0, v_1, \dots, v_k \rangle}$ najkraći put između $\boldsymbol{v_0}$ i $\boldsymbol{v_k}$. Neka za svako $\boldsymbol{i}$ i $\boldsymbol{j}$, za koje važi $\boldsymbol{0 \leq i \leq j \leq k}$, postoji put $\boldsymbol{p_{ij} = \langle v_i, v_{i+1}, \dots, v_j \rangle}$ koji povezuje $\boldsymbol{v_i}$ i $\boldsymbol{v_j}$ i koji je pod-put od $\boldsymbol{p}$.
  Tada je i $\boldsymbol{p_{ij}}$ najkraći put od $\boldsymbol{v_i}$ do $\boldsymbol{v_j}$.
- Dokaz?
- Može li neki drugi pod-put biti kraći? Ako može zašto nije on deo puta $\boldsymbol{p}$? Jer suštinski on bi učinio put $\boldsymbol{p}$ kraćim, što je kontradikcija sa početnom pretpostavkom

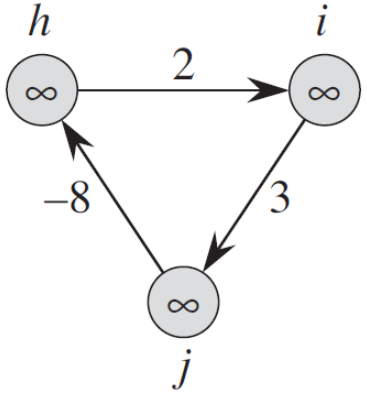
NEGATIVNE TEŽINE …

- Kod nekih grafova, mogu postojati i grane sa negativnim težinama
- Ako nema negativnih težina na putu od čvora s do v, onda u algoritmima $\delta(s, v)$ ostaje onakvo kako je definisano i sve je ok
- Ako postoje negativne grane i ako one formiraju ciklus (imaju povratne grane), onda $\delta(s, v)$ nije dobro definisano
- Ni jedna putanja od s nije najkraća, jer se uvek može napraviti povratak niz negativnu grane, čime se smanjuje ukupna cena puta $\delta(s, v)$
  - Za ovakve slučajeve se kaže da je $\delta(s, v) = -\infty$
- Ovaj problem je najlakše shvatiti na primeru

… NEGATIVNE TEŽINE …

- Problem negativnih težina je najlakše shvatiti na primeru





- $\boldsymbol{\delta(s,a)=3}$,
- $\boldsymbol{\delta(s,b)=\delta(s,a)+\delta(a,b)=3-4=-1}$ (samo jedan moguć, konačan put)
- $\boldsymbol{\delta(s,e)=2}$, ali i $\boldsymbol{\delta(s,e)=\delta(s,e)+\delta(e,f)+\delta(f,e)=2+3-6=-1}$, ali i $\boldsymbol{\delta(s,e)=\delta(s,e)+\delta(e,f)+\delta(f,e)+\delta(e,f)+\delta(f,e)=}$ $\boldsymbol{2+3-6+3-6=-4}$, gde je tu kraj?
- Čvorovi $\boldsymbol{h}$, $\boldsymbol{i}$, $\boldsymbol{j}$ nisu dostižni iz $\boldsymbol{s}$ za je $\boldsymbol{\delta(s,h)=\delta(s,i)=\delta(s,j)=\infty}$

… NEGATIVNE TEŽINE

- Neki algoritmi za najkraći put kao Dajsktrin algoritam pretpostavljaju da nema negativnih težina
- Drugi algoritmi, poput Belman-Fordovog ih uzimaju u obzir i rade korektno, sve dok nema povratnih grana sa negativnim vrednostima
  - Tipično, ako i postoje negativne povratne grane, algoritam to detektuje i prijavljuje

- GENERALNO, najkraći put ne može sadržati povratne grane (cikluse)
  - Jasno je za grafove sa negativnom težinom
  - Jasno je za grafove sa pozitivnim težinama
  - A grafovima čija je težina 0?
    - Uvek možemo da izbacimo graf čija je težina 0

RELAKSACIJA …

- Algoritmi za računanje najkraćeg puta koristi tehniku relaksacije grane (u, v), tokom koje se proverava da li se može unaprediti putanja od čvora s do v tako što se ide preko u
- Za svaki čvor v pamti se atribut $v.d$ u kojem se čuva gornja granica na ukupnu težinu puta od s do v
- Atribut $v.d$ naziva se procenom najkraćeg puta
- Inicijalizacija kreće na sledeći način:

```
INITIALIZE-SINGLE-SOURCE(G, s)
1  for each vertex v ∈ G.V
2      v.d = ∞
3      v.π = NIL
4  s.d = 0
```

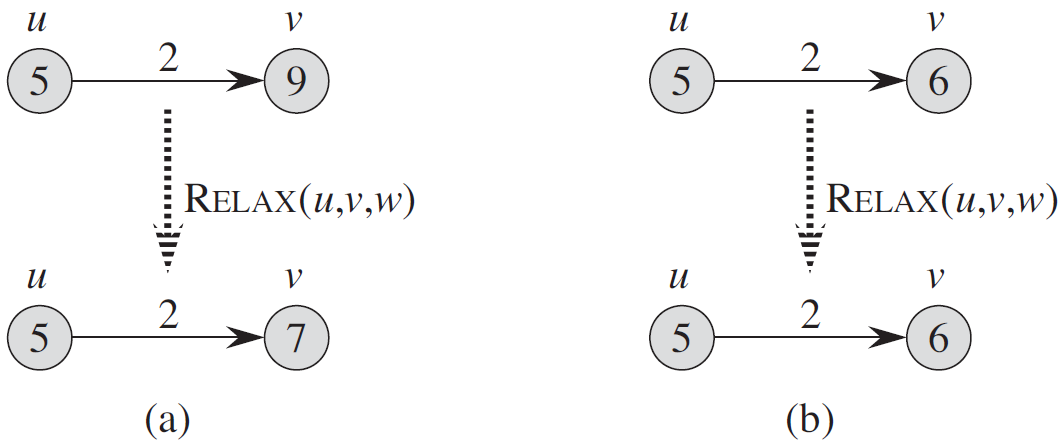
$v.\pi$ služi da se praćenjem pokazivača odredi koji sve čvorovi formiraju najkraći put

… RELAKSACIJA

- Relaksacija se oslanja na sledeću formulu: $v.d \leq u.d + w(u,v)$

```
RELAX(u, v, w)
1  if v.d > u.d + w(u, v)
2        v.d = u.d + w(u, v)
3        v.π = u
```

- Primer kako radi:



Posle a) će se osvežiti $v.d$, posle b) neće, jer se procena ne menja

OSOBINE NAJKRAĆEG PUTA I RELAKSACIJE …

- Da bi sledeće osobine važile potrebno je da čvorovi budu inicijalizovani ***INITIALIZESINGLE-SOURCE(G, s)*** procedurom
  - **Nejednakost trouglova:** Za svaku granu $(\boldsymbol{u}, \boldsymbol{v}) \in \boldsymbol{E}$, važi $\boldsymbol{\delta}(\boldsymbol{s}, \boldsymbol{v}) \leq \boldsymbol{\delta}(\boldsymbol{s}, \boldsymbol{u}) + \boldsymbol{w}(\boldsymbol{u}, \boldsymbol{v})$
  - **Osobina gornje granice:** Za svaki čvor $\boldsymbol{v} \in \boldsymbol{V}$ važi $\boldsymbol{v}.\boldsymbol{d} \geq \boldsymbol{\delta}(\boldsymbol{s}, \boldsymbol{v})$ i jednom kada $\boldsymbol{v}.\boldsymbol{d}$ dostigne vrednost $\boldsymbol{\delta}(\boldsymbol{s}, \boldsymbol{v})$, više se ne menja
  - **Osobina nepostojanja puta:** Ako ne postoji put od $\boldsymbol{s}$ do $\boldsymbol{v}$, onda je $\boldsymbol{v}.\boldsymbol{d} = \boldsymbol{\delta}(\boldsymbol{s}, \boldsymbol{v}) = \infty$
  - **Osobina konvergencije:** Ako je put $(\boldsymbol{s}, \boldsymbol{u})$ plus grana $(\boldsymbol{u}, \boldsymbol{v})$ najkraći put u grafu $\boldsymbol{G}$ od $\boldsymbol{u}$ do $\boldsymbol{v}$ za $\boldsymbol{u}, \boldsymbol{v} \in \boldsymbol{V}$, i ako je $\boldsymbol{u}.\boldsymbol{d} = \boldsymbol{\delta}(\boldsymbol{s}, \boldsymbol{u})$ u bilo kom trenutku pre relaksacije grane $(\boldsymbol{u}, \boldsymbol{v})$, onda je nakon relaksacije $\boldsymbol{v}.\boldsymbol{d} = \boldsymbol{\delta}(\boldsymbol{s}, \boldsymbol{v})$ do kraja rada algoritma

… OSOBINE NAJKRAĆEG PUTA I RELAKSACIJE

- Da bi sledeće osobine važile potrebno je da čvorovi budu inicijalizovani ***INITIALIZESINGLE-SOURCE(G, s)*** procedurom (nastavak)
  - **Osobina relaksacije puta:** Ako je put $\boldsymbol{p} = \langle \boldsymbol{v_0}, \boldsymbol{v_1}, \ldots, \boldsymbol{v_k} \rangle$ najkraći put od $\boldsymbol{s} = \boldsymbol{v_0}$ do $\boldsymbol{v_k}$ i ako vršimo relaksaciju grana u $\boldsymbol{p}$ u redosledu $(\boldsymbol{v_0}, \boldsymbol{v_1})$, $(\boldsymbol{v_1}, \boldsymbol{v_2})$, … , $(\boldsymbol{v_{k-1}}, \boldsymbol{v_k})$, onda će $\boldsymbol{v_k}.\boldsymbol{d} = \boldsymbol{\delta}(\boldsymbol{s}, \boldsymbol{v_k})$. Ovo važi bez obzira sve preostale relaksacije koje nastaju, čak i ako se dešavaju između relaksacija grana iz puta $\boldsymbol{p}$
  - **Osobina *predecessor subgraph*:** Jednom kada je $\boldsymbol{v}.\boldsymbol{d} = \boldsymbol{\delta}(\boldsymbol{s}, \boldsymbol{v})$ za svako $\boldsymbol{v} \in \boldsymbol{V}$, *predecessor subgraph* je stablo sa najkraćim putem sa korenom u $\boldsymbol{s}$

BELMAN-FORD ALGORITAM …

- Rešava na generički način problem najkraćeg puta iz zadatog čvora, pri čemu su dozvoljene i negativne težine
- Algoritam provera da li postoji putanja koja se može doseći iz startnog čvora, a koja u sebi sadrži ciklus sa negativnom težinom (vraća se boolean vrednost)
  - Ako postoji, algoritam obaveštava da se ne može odrediti najkraća putanja iz datog čvora
  - Ako nema takve putanje, algoritam vraća najkraće putanje (čvorove na njima) i njihove težine
- Algoritam relaksira grane, progresivno smanjujući predviđanje $v.d$ za najkraći put od čvora s do svakog čvora $v \in V$ sve dok se ne dosegne $\delta(s, v)$
- Tačna vrednost (*true*) vraća se ako i samo ako nema negativnih ciklusa koji se mogu doseći iz startnog čvora

… BELMAN-FORD ALGORITAM …

- Algoritam:

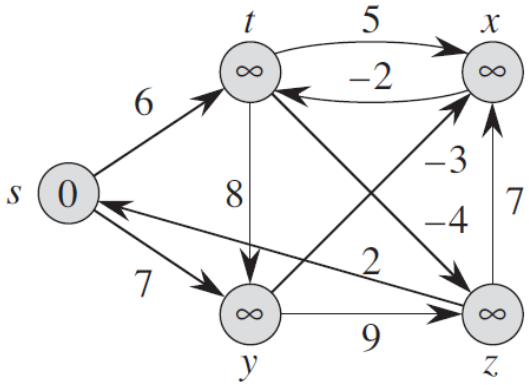
```
BELLMAN-FORD(G, w, s)
1  INITIALIZE-SINGLE-SOURCE(G, s)
2  for i = 1 to |G.V| − 1
3        for each edge (u, v) ∈ G.E
4              RELAX(u, v, w)
5  for each edge (u, v) ∈ G.E
6        if v.d > u.d + w(u, v)
7              return FALSE
8  return TRUE
```

```
INITIALIZE-SINGLE-SOURCE(G, s)
1  for each vertex v ∈ G.V
2        v.d = ∞
3        v.π = NIL
4  s.d = 0
```

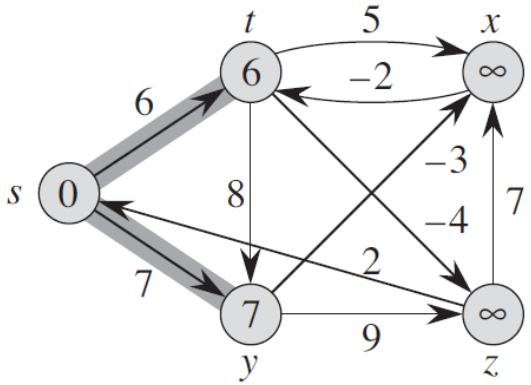
```
RELAX(u, v, w)
1  if v.d > u.d + w(u, v)
2        v.d = u.d + w(u, v)
3        v.π = u
```

… BELMAN-FORD ALGORITAM …

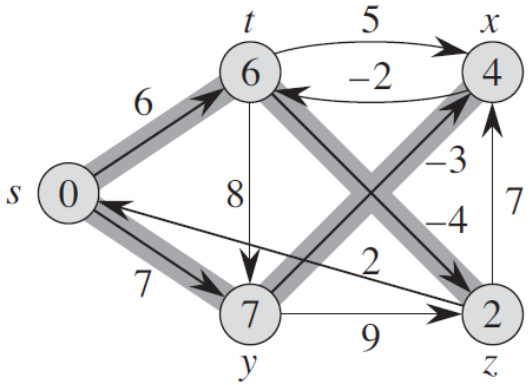
- Primer rada algoritma:



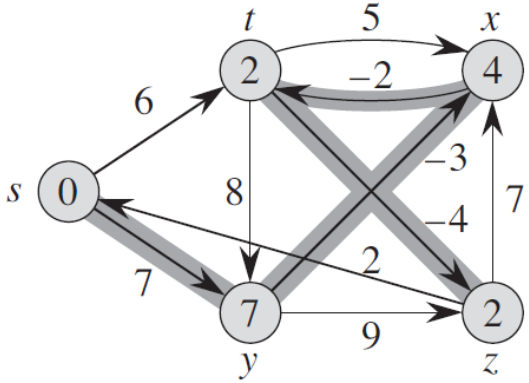
(a)



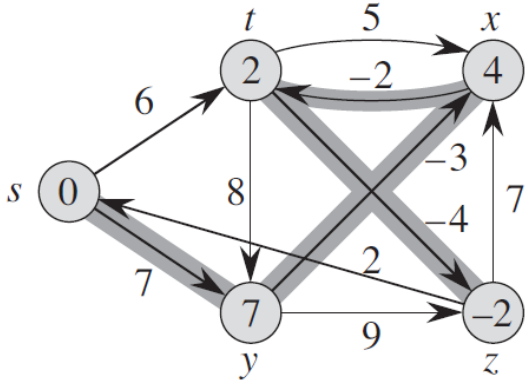
(b)



(c)



(d)



(e)

… BELMAN-FORD ALGORITAM …

- Kako Belman-Ford algoritam radi:
  - Nakon inicijalizacije čvorova, algoritam pravi $|V| - 1$ prolaza kroz grane u grafu, što suštinski odgovara proveri svih grana a svaki čvor grafa (osim startnog)
  - U svakoj iteraciji ***for*** petlje iz linije 2, pristupa se svakoj grani i relaksira se njena cena, kao i veza među čvorovima
  - Kako su najkraću putevi od startnog čvora do bilo kog drugog čvora koji mu je dostupan prosti, može biti najviše $|V| - 1$ čvorova

```
BELLMAN-FORD(G, w, s)
1  INITIALIZE-SINGLE-SOURCE(G, s)
2  for i = 1 to |G.V| − 1
3      for each edge (u, v) ∈ G.E
4          RELAX(u, v, w)
5  for each edge (u, v) ∈ G.E
6      if v.d > u.d + w(u, v)
7          return FALSE
8  return TRUE
```

```
RELAX(u, v, w)
1  if v.d > u.d + w(u, v)
2      v.d = u.d + w(u, v)
3      v.π = u
```

  - Svaka od $|V| - 1$ iteracija ***for*** petlje u liniji 2 relaksira svaku od $|E|$ grana, tako se u i-toj iteraciji relaksira i grana (v_{i-1}, v_i), tako da se na osnovu osobine relaksacije dobija da je $v.d = \delta(s, v)$
  - Nakon što je izvršena relaksacija čvorova, algoritam prolazi se kroz sve grane i proverava se da li postoji negativni ciklus

… BELMAN-FORD ALGORITAM

- Kako Belman-Ford algoritam radi (nastavak):
  - Posledica činjenice da je $v.d = \delta(s, v)$ je da ako nema negativnih grana, onda postoji put između s i v samo ako je po završetku Bellman-Ford algoritma $v.d < \infty$
  - Kompleksnost Belman-Ford algoritma iznosi $O(VE)$
  - Inicijalizacija se realizuje u $\Theta(V)$ vremena
  - Svaki od $|V| - 1$ prolaza se realizuje u $\Theta(E)$ vremena
  - ***for*** petlja u linijama 5-7 se realizuje u $O(E)$ vremena

```
BELLMAN-FORD(G, w, s)
1  INITIALIZE-SINGLE-SOURCE(G, s)
2  for i = 1 to |G.V| − 1
3      for each edge (u, v) ∈ G.E
4          RELAX(u, v, w)
5  for each edge (u, v) ∈ G.E
6      if v.d > u.d + w(u, v)
7          return FALSE
8  return TRUE
```

```
RELAX(u, v, w)
1  if v.d > u.d + w(u, v)
2      v.d = u.d + w(u, v)
3      v.π = u
```

DAJKSTRIN ALGORITAM …

- Rešava problem najkraćeg puta iz zadatog čvora za težinski, usmereni graf $\boldsymbol{G} = (\boldsymbol{V}, \boldsymbol{E})$ za slučaj kada ni jedna grana nema negativnih težina
- Podrazumeva se da je $\boldsymbol{w}(\boldsymbol{u}, \boldsymbol{v}) \geq \boldsymbol{0}$ za svaku granu $(\boldsymbol{u}, \boldsymbol{v}) \in \boldsymbol{E}$
- Kada je dobro implementiran Dajsktrin algoritam radi brže od Bellman-Fordovog
- Algoritam održava skup $\boldsymbol{S}$ čvorova čije su najkraće putanje od čvora $\boldsymbol{s}$ već određene
- Algoritam iterativno uzima neki čvor $\boldsymbol{u} \in \boldsymbol{V} - \boldsymbol{S}$ sa putem za koji je procenjeno da imam najmanju težinu, dodaje $\boldsymbol{u}$ u $\boldsymbol{S}$ i relaksira sve grane koje izlaze iz čvora $\boldsymbol{u}$
- U opisanoj implementaciji koriste se minimalni red prioriteta $\boldsymbol{Q}$ gde se kao ključ koristi atribut $\boldsymbol{d}$

… DAJKSTRIN ALGORITAM …

- Algoritam:

```
DIJKSTRA(G, w, s)
1  INITIALIZE-SINGLE-SOURCE(G, s)
2  S = ∅
3  Q = G.V
4  while Q ≠ ∅
5       u = EXTRACT-MIN(Q)
6       S = S ∪ {u}
7       for each vertex v ∈ G.Adj[u]
8            RELAX(u, v, w)
```

```
INITIALIZE-SINGLE-SOURCE(G, s)
1  for each vertex v ∈ G.V
2       v.d = ∞
3       v.π = NIL
4  s.d = 0
```

```
RELAX(u, v, w)
1  if v.d > u.d + w(u, v)
2       v.d = u.d + w(u, v)
3       v.π = u
```

… DAJKSTRIN ALGORITAM …

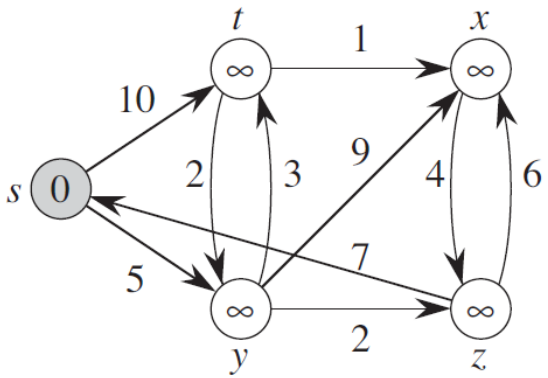
- Kako Dajsktrin algoritam radi:
  - Inicijalizuje se skup S čvorova do kojih je najkraći put određen (u početku je prazan)
  - Zatim se popuni minimalni red prioriteta Q
  - Ko će biti na vrhu reda Q na početku?
  - Sve dok se red Q ne isprazni:
    - uzima se čvor u sa najmanjom procenjenom cenom puta koji se nalazi na vrhu reda Q
    - taj se čvor dodaje u S
    - za sve čvorove povezane sa u relaksiraju se grane koje ih povezuju (promena u d podrazumeva promenu u samom redu prioriteta Q
    - Kako se novi čvorovi nikada ne dodaju u Q posle linije 3 i kako se u svakoj iteraciji izvlači samo jedan čvor iz Q, ***while*** petlja u liniji 4 će se izvršiti $|V|$ puta

```
DIJKSTRA(G, w, s)
1  INITIALIZE-SINGLE-SOURCE(G, s)
2  S = ∅
3  Q = G.V
4  while Q ≠ ∅
5      u = EXTRACT-MIN(Q)
6      S = S ∪ {u}
7      for each vertex v ∈ G.Adj[u]
8          RELAX(u, v, w)
```

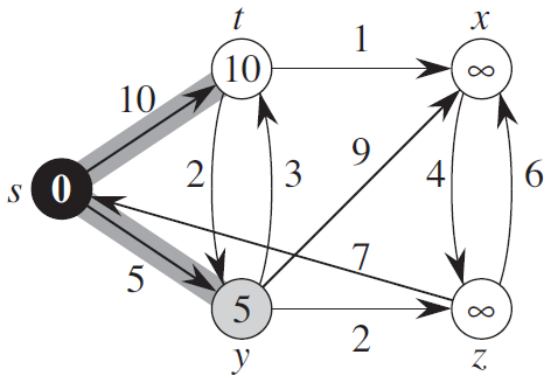
```
RELAX(u, v, w)
1  if v.d > u.d + w(u, v)
2      v.d = u.d + w(u, v)
3      v.π = u
```

… DAJKSTRIN ALGORITAM …

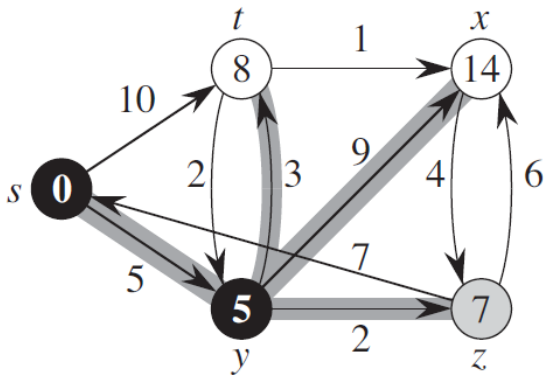
- Kako Dajsktrin algoritam radi - primer:



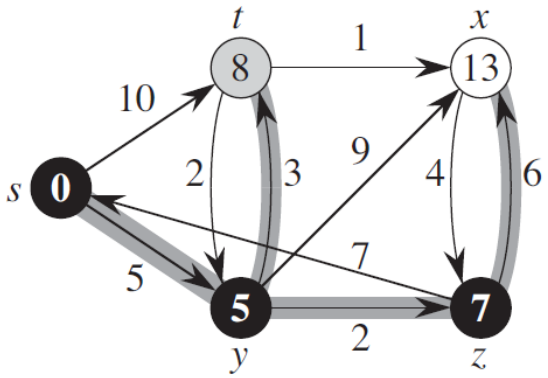
(a)



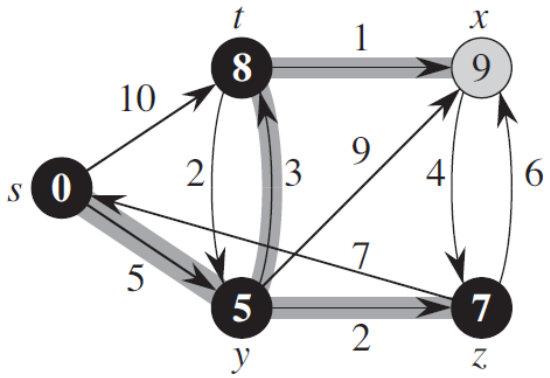
(b)



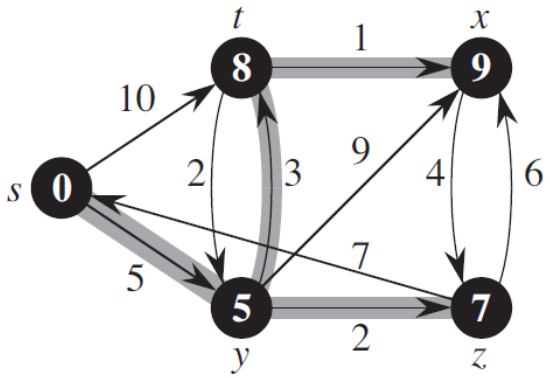
(c)



(d)



(e)



(f)

… DAJKSTRIN ALGORITAM …

- Kako Dajsktrin algoritam radi (nastavak):
  - Algoritam stalno bira čvor sa najmanjom težinom grane između njega i poslednje čvora dodatog u $\boldsymbol{S}$, tj. bira najlaganiji čvor iz $\boldsymbol{V} - \boldsymbol{S}$ i njega dodaje u $\boldsymbol{S}$
  - Ovaj izbor je u suštini pohlepan
  - Kada algoritam završi za svaki čvor $\boldsymbol{v} \in \boldsymbol{V}$ važi da je $\boldsymbol{v.d} = \boldsymbol{\delta(s, v)}$
  - Dokaz za prethodnu tvrdnju se oslanja na invarijantnost petlje u liniji 4 koja kaže da je pre svakog ulasku u petlju $\boldsymbol{v.d} = \boldsymbol{\delta(s, v)}$ za svaki čvor $\boldsymbol{v} \in \boldsymbol{S}$

```
DIJKSTRA(G, w, s)
1  INITIALIZE-SINGLE-SOURCE(G, s)
2  S = ∅
3  Q = G.V
4  while Q ≠ ∅
5      u = EXTRACT-MIN(Q)
6      S = S ∪ {u}
7      for each vertex v ∈ G.Adj[u]
8          RELAX(u, v, w)
```

```
RELAX(u, v, w)
1  if v.d > u.d + w(u, v)
2      v.d = u.d + w(u, v)
3      v.π = u
```

… DAJKSTRIN ALGORITAM …

- Kako Dajsktrin algoritam radi (nastavak):
  - Dokaz za prethodnu tvrdnju se oslanja na invarijantnost petlje u liniji 4 koja kaže da je pre svakog ulasku u petlju $v.d = \delta(s, v)$ za svaki čvor $v \in S$:
    - **Inicijalizacija:** $S = \emptyset$, trivijalno
    - **Održivost:** Treba dokazati da je $u.d = \delta(s, u)$. Ako ovo nije tačno, onda $u.d$ nije najmanje, pa ne može biti na vrhu reda prioriteta Q
    - **Završavanje:** $Q = \emptyset$, posledično, svi su čvorovi dodati i kako je $S = V - Q = S$, što implicira na osnovu **Inicijalizacija** i **Održivosti** $v.d = \delta(s, v)$ za sve čvorove iz $v \in V$
  - Na taj način je dokazana korektnost Dajkstrinog algoritma

```
DIJKSTRA(G, w, s)
1  INITIALIZE-SINGLE-SOURCE(G, s)
2  S = ∅
3  Q = G.V
4  while Q ≠ ∅
5      u = EXTRACT-MIN(Q)
6      S = S ∪ {u}
7      for each vertex v ∈ G.Adj[u]
8          RELAX(u, v, w)
```

```
RELAX(u, v, w)
1  if v.d > u.d + w(u, v)
2      v.d = u.d + w(u, v)
3      v.π = u
```

… DAJKSTRIN ALGORITAM …

- Kako Dajsktrin algoritam radi (nastavak):
  - Posledica svega prethodnog je da Dajkstrin algoritam primenjen na težinski, usmereni graf $\boldsymbol{G = (V, E)}$ sa nenegativnom težinskom funkcijom $\boldsymbol{w}$ počev iz čvora $\boldsymbol{s}$ generiše *predecessor subgraph* $\boldsymbol{G_\pi}$ je stablo najkraćih puteva sa korenom u $\boldsymbol{s}$

```
DIJKSTRA(G, w, s)
1  INITIALIZE-SINGLE-SOURCE(G, s)
2  S = ∅
3  Q = G.V
4  while Q ≠ ∅
5      u = EXTRACT-MIN(Q)
6      S = S ∪ {u}
7      for each vertex v ∈ G.Adj[u]
8          RELAX(u, v, w)
```

```
RELAX(u, v, w)
1  if v.d > u.d + w(u, v)
2      v.d = u.d + w(u, v)
3      v.π = u
```

… DAJKSTRIN ALGORITAM

- Analiza Dajsktrinog algoritma:
  - Algoritam održava red $\boldsymbol{Q}$ implicitnim pozivima: INSERT (jednom za svaki čvor), EXTRACT-MIN (jednom za svaki čvor) i DECREASE-KEY
  - Kako se svaki čvor $\boldsymbol{u} \in \boldsymbol{V}$ dodaje u $\boldsymbol{S}$ samo jednom, onda se i lista povezanosti $\boldsymbol{Adj[u]}$ čvora $\boldsymbol{u}$ proučava u liniji 7 tačno jednom u celom algoritmu
  - Kako ukupno ima $|\boldsymbol{E}|$ grana, ***for*** petlja u liniji 7 se izvršava tačno $|\boldsymbol{E}|$ puta, pa se poziva DECREASE-KEY najviše $|\boldsymbol{E}|$ puta

```
DIJKSTRA(G, w, s)
1  INITIALIZE-SINGLE-SOURCE(G, s)
2  S = ∅
3  Q = G.V
4  while Q ≠ ∅
5      u = EXTRACT-MIN(Q)
6      S = S ∪ {u}
7      for each vertex v ∈ G.Adj[u]
8          RELAX(u, v, w)
```

```
RELAX(u, v, w)
1  if v.d > u.d + w(u, v)
2      v.d = u.d + w(u, v)
3      v.π = u
```

  - Očigledno vreme izvršavanja Dajkstrinog algoritma zavisi od toga kako je implementiran red prioriteta $\boldsymbol{Q}$
  - Kada se sve uzme u obzir dobija se vreme od $\boldsymbol{O(V^2 + E) = O(V^2)}$
  - Algoritam se dodatno može unaprediti tako da se dobije vreme izvršavanja $\boldsymbol{O(V \log V + E)}$ – primenom Fibonačijevog hipa

SVI NAJKRAĆI PUTEVI

SVI NAJKRAĆI PUTEVI …

- U prethodnoj priči smo proučavali najkraće puteve iz jednog čvora ka svim ostalima, ali postoji i problematika kada se žele odrediti svi najkraći putevi iz svih čvorova
- Naravno, mogu se primeniti algoritmi za najkraće puteve iz nekog čvorova, pri čemu se algoritam primenjuje nad svakim od $|\boldsymbol{V}|$ čvorova
- To je u suštini vrlo nepraktično, jer
  - za slučajeve koje pokriva Dajkstrin algoritam potrebno je $\boldsymbol{O}(\boldsymbol{V^3} + \boldsymbol{VE}) = \boldsymbol{O}(\boldsymbol{V^3})$ vremena, u najboljem slučaju $\boldsymbol{O}(\boldsymbol{V^2} \log \boldsymbol{V} + \boldsymbol{VE})$
  - za slučajeve koje pokriva Belman-Ford algoritam porebno je $\boldsymbol{O}(\boldsymbol{V^2E})$, što za vrlo guste grafove iznosi čak $\boldsymbol{O}(\boldsymbol{V^4})$

… SVI NAJKRAĆI PUTEVI …

- Osim u nekim algoritmima, za određivanje svih najkraćih puteva koriste se matrice povezanosti za realizaciju algoritama
- Pretpostavlja se da su čvorovi numerisani redom $\mathbf{1}, \mathbf{2}, \ldots, |\boldsymbol{V}|$
- Ulaz je matrica $\boldsymbol{W} = (\boldsymbol{w_{ij}})$ dimenzija $|\boldsymbol{V}| \times |\boldsymbol{V}|$ koja reprezentuje težine grana između čvorova usmerenog graf $\boldsymbol{G} = (\boldsymbol{V}, \boldsymbol{E})$:

$$\boldsymbol{w_{ij}} = \begin{cases} \mathbf{0} & \boldsymbol{i = j} \\ \boldsymbol{\text{težina grane } (i,j)} & \boldsymbol{i \neq j, (i,j) \in E} \\ \infty & \boldsymbol{i \neq j, (i,j) \notin E} \end{cases}$$

- Rezultat se obično beleži u matricu (tabelu) $\boldsymbol{D} = (\boldsymbol{d_{ij}})$ dimenzija $|\boldsymbol{V}| \times |\boldsymbol{V}|$, gde $\boldsymbol{d_{ij}}$ predstavlja težinu najkraće putanje između čvora $\boldsymbol{i}$ i čvora $\boldsymbol{j}$
- To znači da bi na kraju algoritma trebalo da bude $\boldsymbol{d_{ij}} = \boldsymbol{\delta(i,j)}$

… SVI NAJKRAĆI PUTEVI …

- Pored težine najkraćeg puta, algoritmi trebaju da odrede ***predecessor matrix***, $\Pi = (\boldsymbol{\pi_{ij}})$
- $\boldsymbol{\pi_{ij}} = \boldsymbol{NIL}$ ili ako ne postoji veze između čvora $\boldsymbol{i}$ i čvora $\boldsymbol{j}$, ili ako je $\boldsymbol{i} = \boldsymbol{j}$
- Kada je $\boldsymbol{\pi_{ij}} \neq \boldsymbol{NIL}$, onda je $\boldsymbol{\pi_{ij}}$ čvor koji prethodi čvoru $\boldsymbol{j}$ na nekom od najkraćih puteva od $\boldsymbol{i}$
- Ovo znači da se iz reda $\boldsymbol{i}$ *predecessor* matrice može indukovati stablo najkraćih puteva za čvor $\boldsymbol{i}$ sa čvorom $\boldsymbol{i}$ kao korenom
- Za svaki čvor $\boldsymbol{i} \in \boldsymbol{V}$ definišemo ***predecessor subgraph*** od $\boldsymbol{G}$ za čvor $\boldsymbol{i}$ kao $\boldsymbol{G_{\pi,i}} = (\boldsymbol{V_{\pi,i}}, \boldsymbol{E_{\pi,i}})$ gde je:

$$\boldsymbol{V_{\pi,i}} = \{\boldsymbol{j} \in \boldsymbol{V} : \boldsymbol{\pi_{ij}} \neq \boldsymbol{NIL}\} \cup \{\boldsymbol{i}\}$$

$$\boldsymbol{E_{\pi,i}} = \{(\boldsymbol{\pi_{ij}}, \boldsymbol{j}) : \boldsymbol{j} \in \boldsymbol{V_{\pi,i}} - \{\boldsymbol{i}\}\}$$

… SVI NAJKRAĆI PUTEVI …

- Ako je $G_{\pi,i}$ *predecessor subgraph* od G za čvor i, najkraći put od čvora i do čvora j može se odštampati na sledeći način:

```
PRINT-ALL-PAIRS-SHORTEST-PATH(Π, i, j)
1  if i == j
2       print i
3  elseif π_ij == NIL
4       print "no path from" i "to" j "exists"
5  else PRINT-ALL-PAIRS-SHORTEST-PATH(Π, i, π_ij)
6       print j
```

- Rad ove procedure je vrlo jednostavan, kreće od čvora do kojeg se traži najkraći put i ide se od njega unazad sve do čvora od kojeg se kreće

… SVI NAJKRAĆI PUTEVI

- Nomenklatura:
  - Za graf $\boldsymbol{G} = (\boldsymbol{V}, \boldsymbol{E})$ pretpostavljamo da ima $\boldsymbol{n} = |\boldsymbol{V}|$ čvorova
  - Velika slova označavaju odgovarajuću matricu $\boldsymbol{W}$, $\boldsymbol{D}$
  - Individualni elementi matrice su označeni: $\boldsymbol{w_{ij}}$, $\boldsymbol{d_{ij}}$
  - Zapis sa uperskriptom, $\boldsymbol{D^{(m)}} = \left(\boldsymbol{d_{ij}^{(m)}}\right)$, označava $\boldsymbol{m}$-tu iteraciju (ili iteraciju)
  - Za neku matricu $\boldsymbol{A}$ dimenzija $\boldsymbol{n}$ postoji atribut $\boldsymbol{A.rows}$ koji čuva to $\boldsymbol{n}$

NAJKRAĆI PUTEVI I MATRIČNO MNOŽENJE …

- Predstavlja rešenje dinamičkim programiranjem
- **Definisanje strukture optimalnog rešenja**
  - Neka je povezani graf predstavljen matricom povezanosti $\boldsymbol{W} = (\boldsymbol{w_{ij}})$
  - Neka je $\boldsymbol{p}$ najkraći put od čvora $\boldsymbol{i}$ do čvora $\boldsymbol{j}$ i pretpostaviti da ima najviše $\boldsymbol{m}$ grana
  - Ako nema negativnih ciklusa, $\boldsymbol{m}$ je konačno
  - Ako je $\boldsymbol{i} = \boldsymbol{j}$, onda $\boldsymbol{p}$ ima težinu $\boldsymbol{0}$ i ni jednu granu
  - Ako su čvor $\boldsymbol{i}$ i čvor $\boldsymbol{j}$ različiti, onda se put može dekomponovati na putanju $\boldsymbol{p'} + (\boldsymbol{k}, \boldsymbol{j})$, gde je $\boldsymbol{p'}$ najkraći put od čvora $\boldsymbol{i}$ do čvora $\boldsymbol{k}$ i ona ima najviše $\boldsymbol{m} - \boldsymbol{1}$ granu

… NAJKRAĆI PUTEVI I MATRIČNO MNOŽENJE …

- **Rekurzivno rešenje**
  - Neka je $l_{ij}^{(m)}$ najmanja težina nekog puta od čvora i do čvora j koja sadrži m grana
  - Kada je $m = 0$ može postojati (najkraći) put od čvora i do čvora j koja sadrži 0 grana samo ako je $i = j$, tako da

$$l_{ij}^{(0)} = \begin{cases} 0 & i = j \\ \infty & i \neq j \end{cases}$$

  - Za $m \geq 1$ određujemo $l_{ij}^{(m)}$ tako što biramo manju vrednost između puta sa težinom $l_{ij}^{(m-1)}$ (koja predstavlja najkraći put od čvora i do čvora j koja sadrži $m - 1$ granu) i najmanjeg od puteva sa m grana koji nastaje kombinacijom puta sa težinom $l_{ij}^{(m-1)}$ i grana koje vode od poslednjeg čvora, k, iz tog puta do čvora j

- **Rekurzivno rešenje (nastavak)**
  - Za $\boldsymbol{m \geq 1}$ određujemo $\boldsymbol{l_{ij}^{(m)}}$ tako što biramo manju vrednost između puta sa težinom $\boldsymbol{l_{ij}^{(m-1)}}$ (koja predstavlja najkraći put od čvora $\boldsymbol{i}$ do čvora $\boldsymbol{j}$ koja sadrži $\boldsymbol{m-1}$ granu) i najmanjeg od puteva sa $\boldsymbol{m}$ grana koji nastaje kombinacijom puta sa težinom $\boldsymbol{l_{ij}^{(m-1)}}$ i grana koje vode od poslednjeg čvora, $\boldsymbol{k}$, iz tog puta do čvora $\boldsymbol{j}$, tako da se za rekurziju dobija:

$$\boldsymbol{l_{ij}^{(m)}} = \min\left(\boldsymbol{l_{ij}^{(m-1)}}, \min_{1\leq k\leq n}\left\{\boldsymbol{l_{ik}^{(m-1)}} + \boldsymbol{w_{ij}}\right\}\right) = \min_{1\leq k\leq n}\left\{\boldsymbol{l_{ik}^{(m-1)}} + \boldsymbol{w_{ij}}\right\}$$

  - Prethodna jednakost važi, jer je $\boldsymbol{w_{jj} = 0}$ za svako $\boldsymbol{j}$
  - Ako graf nema negativnih ciklusa, onda za svaki par čvorova $\boldsymbol{i}$ i $\boldsymbol{j}$ kod kojih je $\boldsymbol{\delta(i,j) < \infty}$, postoji direktan put, koji je najkraći i koji ima najviše $\boldsymbol{n-1}$ grana

… NAJKRAĆI PUTEVI I MATRIČNO MNOŽENJE …

- **Bottom up algoritam**
  - Kao ulaz stiže $\boldsymbol{W} = (\boldsymbol{w_{ij}})$
  - Računa seriju matrica $\boldsymbol{L^{(0)}}, \boldsymbol{L^{(1)}}, \dots, \boldsymbol{L^{(n-1)}}$, gde za svako $\boldsymbol{m} = \boldsymbol{1}, \boldsymbol{2}, \dots, \boldsymbol{n-1}$ imamo $\boldsymbol{L^{(m)}} = \left(\boldsymbol{l_{ij}^{(m)}}\right)$
  - Matrica $\boldsymbol{L}$ u poslednjem koraku $\boldsymbol{L^{(n-1)}}$ sadrži sve najkraće puteve i njihove težine
  - U prvom koraku će $\boldsymbol{L^{(1)}} = \boldsymbol{W}$, iz prostog razloga što je $\boldsymbol{l_{ij}^{(m)}} = \min\limits_{1 \le k \le n} \left\{\boldsymbol{l_{ik}^{(m-1)}} + \boldsymbol{w_{ij}}\right\}$ te je $\boldsymbol{l_{ij}^{(1)}} = \min\limits_{1 \le k \le n} \left\{\boldsymbol{l_{ik}^{(0)}} + \boldsymbol{w_{ij}}\right\} = \boldsymbol{w_{ij}}$ za svako $\boldsymbol{i}, \boldsymbol{j} \in \boldsymbol{V}$

… NAJKRAĆI PUTEVI I MATRIČNO MNOŽENJE …

- **Bottom up algoritam (nastavak)**
  - Osnova algoritma je sledeća procedura koja uzima matrice $\boldsymbol{L^{(m-1)}}$ i $\boldsymbol{W}$, i kao rezultat vraća matricu $\boldsymbol{L^{(m)}}$
  - U suštini proširuje najkraće puteve određene do tog trenutka za još jednu granu

```
EXTEND-SHORTEST-PATHS(L, W)
1  n = L.rows
2  let L' = (l'_ij) be a new n × n matrix
3  for i = 1 to n
4      for j = 1 to n
5          l'_ij = ∞
6          for k = 1 to n
7              l'_ij = min(l'_ij, l_ik + w_kj)
8  return L'
```

  - Ovo jako liči na množenje matrica, pa otud i naziv algoritma

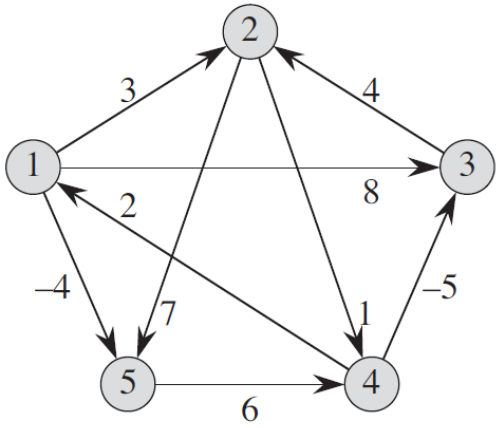
… NAJKRAĆI PUTEVI I MATRIČNO MNOŽENJE …

- **Bottom up algoritam (nastavak)**
  - Osnovni algoritam (njegova spora verzija) koji poziva proceduru za računanje najkraćih puteva u koraku $\boldsymbol{W}$

```
SLOW-ALL-PAIRS-SHORTEST-PATHS(W)
1  n = W.rows
2  L^(1) = W
3  for m = 2 to n − 1
4       let L^(m) be a new n × n matrix
5       L^(m) = EXTEND-SHORTEST-PATHS(L^(m−1), W)
6  return L^(n−1)
```

  - Ovo je sporija implementacija ovog algoritma
  - Ovaj algoritam se izvršava u $\Theta(\boldsymbol{n^4})$

- **Bottom up algoritam (nastavak)**
  - Primer rada



$$L^{(1)} = \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \infty & -5 & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \quad L^{(2)} = \begin{pmatrix} 0 & 3 & 8 & 2 & -4 \\ 3 & 0 & -4 & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & \infty & 1 & 6 & 0 \end{pmatrix}$$

$$L^{(3)} = \begin{pmatrix} 0 & 3 & -3 & 2 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 11 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix} \quad L^{(4)} = \begin{pmatrix} 0 & 1 & -3 & 2 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix}$$

- **Bottom up algoritam – ubrzanje**
  - Nije cilj izračunati svaku $\boldsymbol{L^{(m)}}$, već samo $\boldsymbol{L^{(n-1)}}$
  - U suštini za graf koji nema negativnih ciklusa jasno je da za bilo koje $\boldsymbol{m \geq n-1}$ je u suštini $\boldsymbol{L^{(m)} = L^{(n-1)}}$
  - Zbog komutativnosti matričnog množenja, $\boldsymbol{L^{(n-1)}}$ se može odrediti u $\lceil \log(\boldsymbol{n-1}) \rceil$ matričnih operacija, praćenjem sledeće logike:

    $\boldsymbol{L^{(1)} = W^1}$

    $\boldsymbol{L^{(2)} = W^2 = W \cdot W}$

    $\boldsymbol{L^{(4)} = W^4 = W^2 \cdot W^2}$

    $\boldsymbol{L^{(8)} = W^8 = W^4 \cdot W^4}$

    …

    $\boldsymbol{L^{(2^{\lceil \log(n-1) \rceil})} = W^{2^{\lceil \log(n-1) \rceil}} = W^{2^{\lceil \log(n-1) \rceil - 1}} \cdot W^{2^{\lceil \log(n-1) \rceil - 1}}}$
  - Kako je $\boldsymbol{2^{\lceil \log(n-1) \rceil} \geq n-1}$, finalni proizvod $\boldsymbol{L^{(2^{\lceil \log(n-1) \rceil})}}$ jednak je $\boldsymbol{L^{(n-1)}}$
  - Ova tehnika se zove ponovljeno stepenovanje (***repeated squaring***)

… NAJKRAĆI PUTEVI I MATRIČNO MNOŽENJE …

- **Bottom up algoritam – ubrzanje (nastavak)**
  - Procedura bazirana na ponovljenom stepenovanju:

```
FASTER-ALL-PAIRS-SHORTEST-PATHS(W)
1  n = W.rows
2  L^(1) = W
3  m = 1
4  while m < n − 1
5       let L^(2m) be a new n × n matrix
6       L^(2m) = EXTEND-SHORTEST-PATHS(L^(m), L^(m))
7       m = 2m
8  return L^(m)
```

… NAJKRAĆI PUTEVI I MATRIČNO MNOŽENJE

- **Bottom up algoritam – ubrzanje (nastavak)**
  - Analiza procedure bazirane na ponovljenom stepenovanju:
    - U svakoj iteraciji ***while*** petlje $\boldsymbol{L^{(2m)} = \left(L^{(m)}\right)^2}$, počev $\boldsymbol{m = 1}$
    - Na kraju svake iteracije vrednost $\boldsymbol{m}$ se duplira

```
FASTER-ALL-PAIRS-SHORTEST-PATHS(W)
1  n = W.rows
2  L^(1) = W
3  m = 1
4  while m < n − 1
5      let L^(2m) be a new n × n matrix
6      L^(2m) = EXTEND-SHORTEST-PATHS(L^(m), L^(m))
7      m = 2m
8  return L^(m)
```

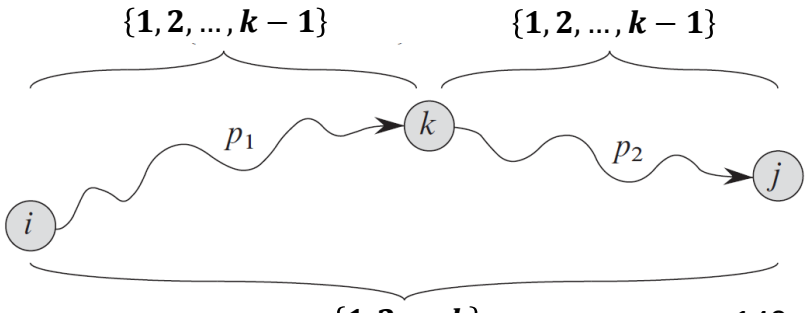
  - Krajnja iteracija izračunava $\boldsymbol{L^{(n-1)}}$ ali zapravo izračunava $\boldsymbol{L^{(2m)}}$, gde je $\boldsymbol{n - 1 \leq m < 2n - 2}$, a $\boldsymbol{L^{(2m)} = L^{(n-1)}}$
  - Kako se svaka od $\lceil \log(\boldsymbol{n - 1}) \rceil$ matrična operacija izvršava $\Theta(\boldsymbol{n^3})$, ova cela procedura traje $\Theta(\boldsymbol{n^3} \log \boldsymbol{n})$
  - Konstante su ovde male, jer nema nekih velikih struktura podataka

FLOYD-WARSHALL ALGORITAM …

- Promenom formulacije problema u dinamičkom programiranju može se doći do drugačijeg algoritma
- Floyd-Warshall algoritam dozvoljava negativne težine, ali se pretpostavlja da nema negativnih ciklusa
- **Definisanje strukture optimalnog rešenja**
  - Algoritam proučava među-čvorove nekog puta $\boldsymbol{p} = \langle \boldsymbol{v_1}, \boldsymbol{v_2}, \dots, \boldsymbol{v_l} \rangle$
  - Među-čvor su svi čvorovi iz puta $\boldsymbol{p}$ koji nisu čvor $\boldsymbol{v_1}$ i $\boldsymbol{v_l}$, znači svaki čvor iz skupa $\boldsymbol{v_1}\{\boldsymbol{v_2}, \dots, \boldsymbol{v_{l-1}}\}$
  - Algoritam se oslanja na to da ako graf $\boldsymbol{G}$ sadrži čvorove $\boldsymbol{V} = \{\boldsymbol{1}, \boldsymbol{2}, \dots, \boldsymbol{n}\}$ posmatrajmo podskup čvorova $\{\boldsymbol{1}, \boldsymbol{2}, \dots, \boldsymbol{k}\}$ za neko $\boldsymbol{k}$
  - Za bilo koji par čvorova $\boldsymbol{i}$ i $\boldsymbol{j}$ posmatramo puteve čiji su svi među-čvorovi iz skupa $\{\boldsymbol{1}, \boldsymbol{2}, \dots, \boldsymbol{k}\}$ i put $\boldsymbol{p}$ koji je prost i sa najmanjom težinom
  - Floyd-Warshall algoritam proučava odnos između $\boldsymbol{p}$ i najkraćih puteva koji povezuju $\boldsymbol{i}$ i $\boldsymbol{j}$ preko među-čvorova $\{\boldsymbol{1}, \boldsymbol{2}, \dots, \boldsymbol{k} - \boldsymbol{1}\}$
  - Međusobni odnosi zavise od toga da li je $\boldsymbol{k}$ među-čvor u putanji $\boldsymbol{p}$

… FLOYD-WARSHALL ALGORITAM …

- **Definisanje strukture optimalnog rešenja (nastavak)**
  - Međusobni odnosi zavise od toga da li je k među-čvor u putu p
  - Ako k nije među-čvor u putu p, onda se svi među-čvorovi iz puta p moraju nalazi u $\{1, 2, \dots, k-1\}$, samim tim se tu nalazi i najkraći put
  - Ako je k među-čvor u putu p, onda se put p može dekomponovati na put p_1 od čvora i do čvora k i put p_2 od čvora k do čvora j
    - u skladu sa svim prethodnim put p_1 mora biti najkraći put od čvora i do čvora k sa svim među-čvorovima iz $\{1, 2, \dots, k\}$
    - tačnije, kako k nije među čvor u putu p_1, onda se svi među-čvorovi puta p_1 nalaze u $\{1, 2, \dots, k-1\}$
    - to sve znači da je put p_1 najkraći put od čvora i do čvora k sa svim među-čvorovima iz $\{1, 2, \dots, k-1\}$
    - slično se može zaključiti i da je put p_2 najkraći put od čvora k do čvora j sa svim među-čvorovima iz $\{1, 2, \dots, k-1\}$



… FLOYD-WARSHALL ALGORITAM …

- **Rekurzivno rešenje**
  - Neka je $\boldsymbol{d_{ij}^{(k)}}$ težina puta od čvora $\boldsymbol{i}$ do čvora $\boldsymbol{j}$ sa svim među-čvorovima u $\{\boldsymbol{1}, \boldsymbol{2}, \dots, \boldsymbol{k}\}$
  - Kada je $\boldsymbol{k = 0}$, put od čvora $\boldsymbol{i}$ do čvora $\boldsymbol{j}$ ima najviše $\boldsymbol{0}$ među-čvorova i najviše jednu granu, tako da je $\boldsymbol{d_{ij}^{(0)} = w_{ij}}$
  - U skladu sa prethodnim, definiše se rekurentna jednačina:

$$\boldsymbol{d_{ij}^{(k)}} = \begin{cases} \boldsymbol{w_{ij}} & \boldsymbol{k = 0} \\ \min\left(\boldsymbol{d_{ij}^{(k-1)}}, \boldsymbol{d_{ik}^{(k-1)}} + \boldsymbol{d_{kj}^{(k-1)}}\right) & \boldsymbol{k \geq 1} \end{cases}$$

  - Kako su svi među-čvorovi nekog grafa iz skupa $\{\boldsymbol{1}, \boldsymbol{2}, \dots, \boldsymbol{n}\}$, matrica $\boldsymbol{D^{(n)} = \left(d_{ij}^{(n)}\right)}$ daje informacije o najkraćim putevima: $\boldsymbol{d_{ij}^{(n)} = \delta(i, j)}$ za svako $\boldsymbol{i, j \in V}$

… FLOYD-WARSHALL ALGORITAM …

- **Bottom up algoritam**
  - Algoritam radi povećavajući $\boldsymbol{k}$
  - Ulaz je matrica težina, $\boldsymbol{W} = (\boldsymbol{w_{ij}})$

```
FLOYD-WARSHALL(W)
1  n = W.rows
2  D^(0) = W
3  for k = 1 to n
4       let D^(k) = (d_ij^(k)) be a new n × n matrix
5       for i = 1 to n
6            for j = 1 to n
7                 d_ij^(k) = min(d_ij^(k-1), d_ik^(k-1) + d_kj^(k-1))
8  return D^(n)
```

  - Ukupno vreme izvršavanja algoritma je $\Theta(\boldsymbol{n^3})$. Zašto?

… FLOYD-WARSHALL ALGORITAM …

- **Primer rada algoritma:**

$$D^{(0)} = \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \infty & -5 & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \qquad D^{(3)} = \begin{pmatrix} 0 & 3 & 8 & 4 & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & -1 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix}$$

$$D^{(1)} = \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \qquad D^{(4)} = \begin{pmatrix} 0 & 3 & -1 & 4 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix}$$

$$D^{(2)} = \begin{pmatrix} 0 & 3 & 8 & 4 & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \qquad D^{(5)} = \begin{pmatrix} 0 & 1 & -3 & 2 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix}$$

… FLOYD-WARSHALL ALGORITAM …

- **Formiranje najkraćeg puta**
  - Najkraći put se može odrediti formiranjem određivanjem *predecessor* matrice Π dok se formira matrica $\boldsymbol{D^{(k)}}$
  - Određuje se sekvenca matrica $\Pi^{(0)}$, $\Pi^{(1)}$, …, $\Pi^{(n)}$, gde je $\Pi = \Pi^{(n)}$ i gde je $\boldsymbol{\pi_{ij}^{(k)}}$ predak čvora $\boldsymbol{j}$ na najkraćem putu od nekog čvor $\boldsymbol{i}$ sa među-čvorovima iz skupa $\{\boldsymbol{1, 2, \ldots, k}\}$
  - Kada je $\boldsymbol{k = 0}$, onda je očigledno da najkraći put od čvora $\boldsymbol{i}$ do čvora $\boldsymbol{j}$ nema među-čvorova, tako da je rekurentna jednačina za $\boldsymbol{\pi_{ij}^{(0)}}$

$$\boldsymbol{\pi_{ij}^{(0)}} = \begin{cases} \boldsymbol{NIL} & \boldsymbol{i = j \ \vee w_{ij} = \infty} \\ \boldsymbol{i} & \boldsymbol{i \neq j \ \wedge w_{ij} < \infty} \end{cases}$$

  - Za $\boldsymbol{k \geq 1}$ da se put od čvora $\boldsymbol{i}$ do čvora $\boldsymbol{j}$ može podeliti na put od čvora $\boldsymbol{i}$ do čvora $\boldsymbol{k}$ i put od čvora $\boldsymbol{k}$ do čvora $\boldsymbol{j}$, gde je $\boldsymbol{k \neq j}$

… FLOYD-WARSHALL ALGORITAM …

- **Formiranje najkraćeg puta (nastavak)**
  - Za $\boldsymbol{k \geq 1}$ da se put od čvora $\boldsymbol{i}$ do čvora $\boldsymbol{j}$ može podeliti na put od čvora $\boldsymbol{i}$ do čvora $\boldsymbol{k}$ i put od čvora $\boldsymbol{k}$ do čvora $\boldsymbol{j}$, gde je $\boldsymbol{k \neq j}$
  - Za pretka od čvora $\boldsymbol{j}$ se može uzeti:
    - Predak čvora $\boldsymbol{j}$ iz najkraćeg puta od čvora $\boldsymbol{i}$ do čvora $\boldsymbol{j}$ sa međučvorovima iz $\{\boldsymbol{1, 2, \ldots, k-1}\}$
    - Predak čvora $\boldsymbol{j}$ iz najkraćeg puta od čvora $\boldsymbol{k}$ do čvora $\boldsymbol{j}$ sa međučvorovima iz $\{\boldsymbol{1, 2, \ldots, k-1}\}$
  - Formalno:

$$\boldsymbol{\pi_{ij}^{(k)}} = \begin{cases} \boldsymbol{\pi_{ij}^{(k-1)}} & \boldsymbol{d_{ij}^{(k-1)} \leq d_{ik}^{(k-1)} + d_{kj}^{(k-1)}} \\ \boldsymbol{\pi_{kj}^{(k-1)}} & \boldsymbol{d_{ij}^{(k-1)} > d_{ik}^{(k-1)} + d_{kj}^{(k-1)}} \end{cases}$$

… FLOYD-WARSHALL ALGORITAM …

- **Primer rada algoritma sa *predecessor* matricom:**

$$D^{(0)} = \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \infty & -5 & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \quad \Pi^{(0)} = \begin{pmatrix} \text{NIL} & 1 & 1 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 2 & 2 \\ \text{NIL} & 3 & \text{NIL} & \text{NIL} & \text{NIL} \\ 4 & \text{NIL} & 4 & \text{NIL} & \text{NIL} \\ \text{NIL} & \text{NIL} & \text{NIL} & 5 & \text{NIL} \end{pmatrix}$$

$$D^{(1)} = \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \quad \Pi^{(1)} = \begin{pmatrix} \text{NIL} & 1 & 1 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 2 & 2 \\ \text{NIL} & 3 & \text{NIL} & \text{NIL} & \text{NIL} \\ 4 & 1 & 4 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 5 & \text{NIL} \end{pmatrix}$$

$$D^{(2)} = \begin{pmatrix} 0 & 3 & 8 & 4 & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \quad \Pi^{(2)} = \begin{pmatrix} \text{NIL} & 1 & 1 & 2 & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 2 & 2 \\ \text{NIL} & 3 & \text{NIL} & 2 & 2 \\ 4 & 1 & 4 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 5 & \text{NIL} \end{pmatrix}$$

… FLOYD-WARSHALL ALGORITAM

- **Primer rada algoritma sa *predecessor* matricom (nastavak):**

$$D^{(3)} = \begin{pmatrix} 0 & 3 & 8 & 4 & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & -1 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \quad \Pi^{(3)} = \begin{pmatrix} \text{NIL} & 1 & 1 & 2 & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 2 & 2 \\ \text{NIL} & 3 & \text{NIL} & 2 & 2 \\ 4 & 3 & 4 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 5 & \text{NIL} \end{pmatrix}$$

$$D^{(4)} = \begin{pmatrix} 0 & 3 & -1 & 4 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix} \quad \Pi^{(4)} = \begin{pmatrix} \text{NIL} & 1 & 4 & 2 & 1 \\ 4 & \text{NIL} & 4 & 2 & 1 \\ 4 & 3 & \text{NIL} & 2 & 1 \\ 4 & 3 & 4 & \text{NIL} & 1 \\ 4 & 3 & 4 & 5 & \text{NIL} \end{pmatrix}$$

$$D^{(5)} = \begin{pmatrix} 0 & 1 & -3 & 2 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix} \quad \Pi^{(5)} = \begin{pmatrix} \text{NIL} & 3 & 4 & 5 & 1 \\ 4 & \text{NIL} & 4 & 2 & 1 \\ 4 & 3 & \text{NIL} & 2 & 1 \\ 4 & 3 & 4 & \text{NIL} & 1 \\ 4 & 3 & 4 & 5 & \text{NIL} \end{pmatrix}$$