Grafovi

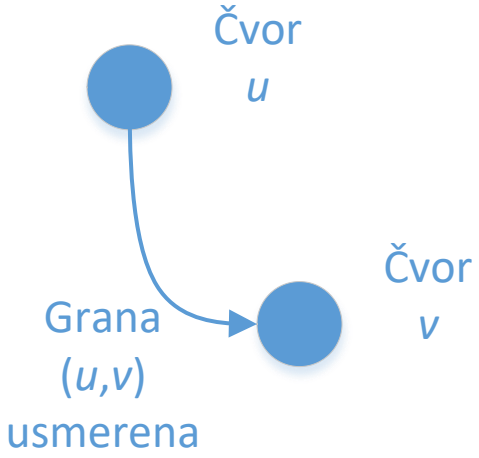
Predmet: Uvod u Algoritme 17 - ESI053
Studijski program: Primenjeno softversko inženjerstvo

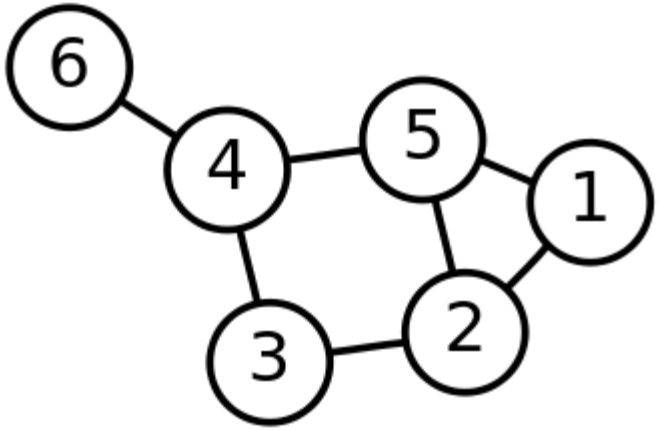
DEPARTMAN ZA RAČUNARSTVO I AUTOMATIKU

DEPARTMAN ZA ENERGETIKU, ELEKTRONIKU I KOMUNIKACIJE

Uvod

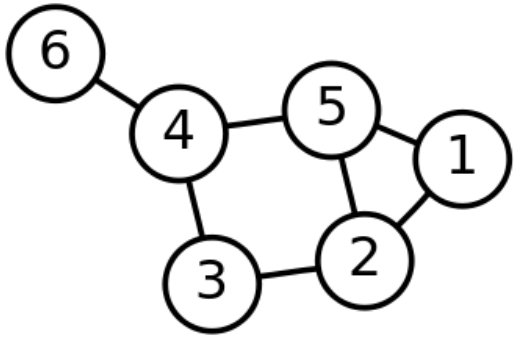
- Grafovi (eng. graphs) se izučavaju u teoriji grafova (oblasti: diskretne matematike i računarske nauke)
- Graf je matematička struktura za modelovanje odnosa između parova objekata.
- Graf se sastoji od:
  - čvorova (objekata) – (eng. *nodes* ili *vertices*), i
  - grana (koje povezuju parove objekata) – (eng. *edges*).





Matematička definicija grafa

- Graf $G = (V, E)$ se sastoji od skupa čvorova V i skupa grana E.
- Grane su 2-elementni podskup od V
- Red grafa je broj čvorova $|V|$
- Veličina grafa je broj grana $|E|$



V = {1, 2, 3, 4, 5, 6}
E = {{1, 2}, {1, 5}, {2, 3}, {2, 5}, {3, 4}, {4, 5}, {4, 6}}

$|V| = 6$
$|E| = 7$

Primena grafova

- Grafovi se upotrebljavaju za modelovanje (predstavljanje) mnogih praktičnih problema.
- Tipično modeliraju relacije i dinamiku procesa u fizičkim, bioločkim, socialnim i informatičkim sistemima.
- U računarstvu grafovi se koriste za predstavljanje računarske mreže, organizacije podataka, tokova podataka i sl.

- Primeri:
  - Sistem datoteka u operativnom sistemu: liči na stablo, ali se dodaju prečice (alijasi),
  - Struktura website-a: čvorovi su stranice, a usmerene grane su hiper linkovi,
  - Mreža puteva: čvorovi su raskrsnice, a grane su putevi.
  - Elektrodisributivna mreža: čvorovi su transformatorske stanice, a grane su vodovi,
  - ...

Tipovi grafova

Sa stanovišta usmerenosti grana:

- Neusmereni (neorijentisani)
- Usmereni (orijentisani)
- Mešoviti
  - deo grana je orijentisan

Sa stanovišta parametara grana:

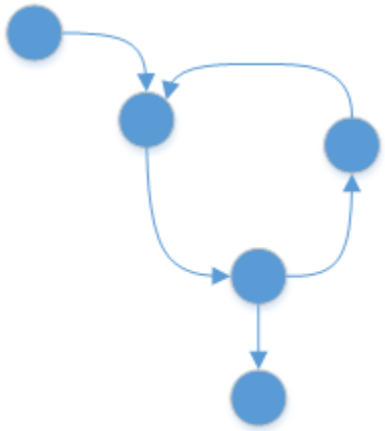
- Direktan graf
- Težinski graf
  - grane imaju parametre, tj. težine
  - ako se ne istakne, se se smatra da je graf direktan

Sa stanovišta postojanja „petlji“

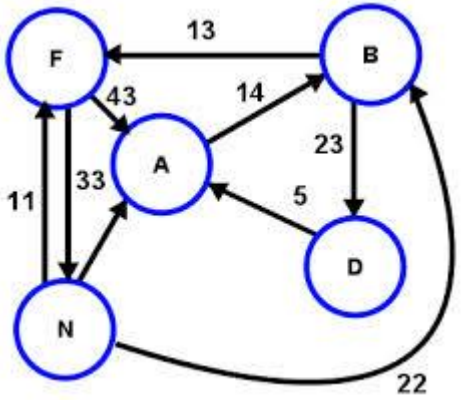
- Aciklični - nema „petlje“
- Ciklični

Potrebno je proveriti da li je graf cikličan.
Neki algoritmi rade samo sa acikljičnim grafovima.

Ciklični usmereni graf

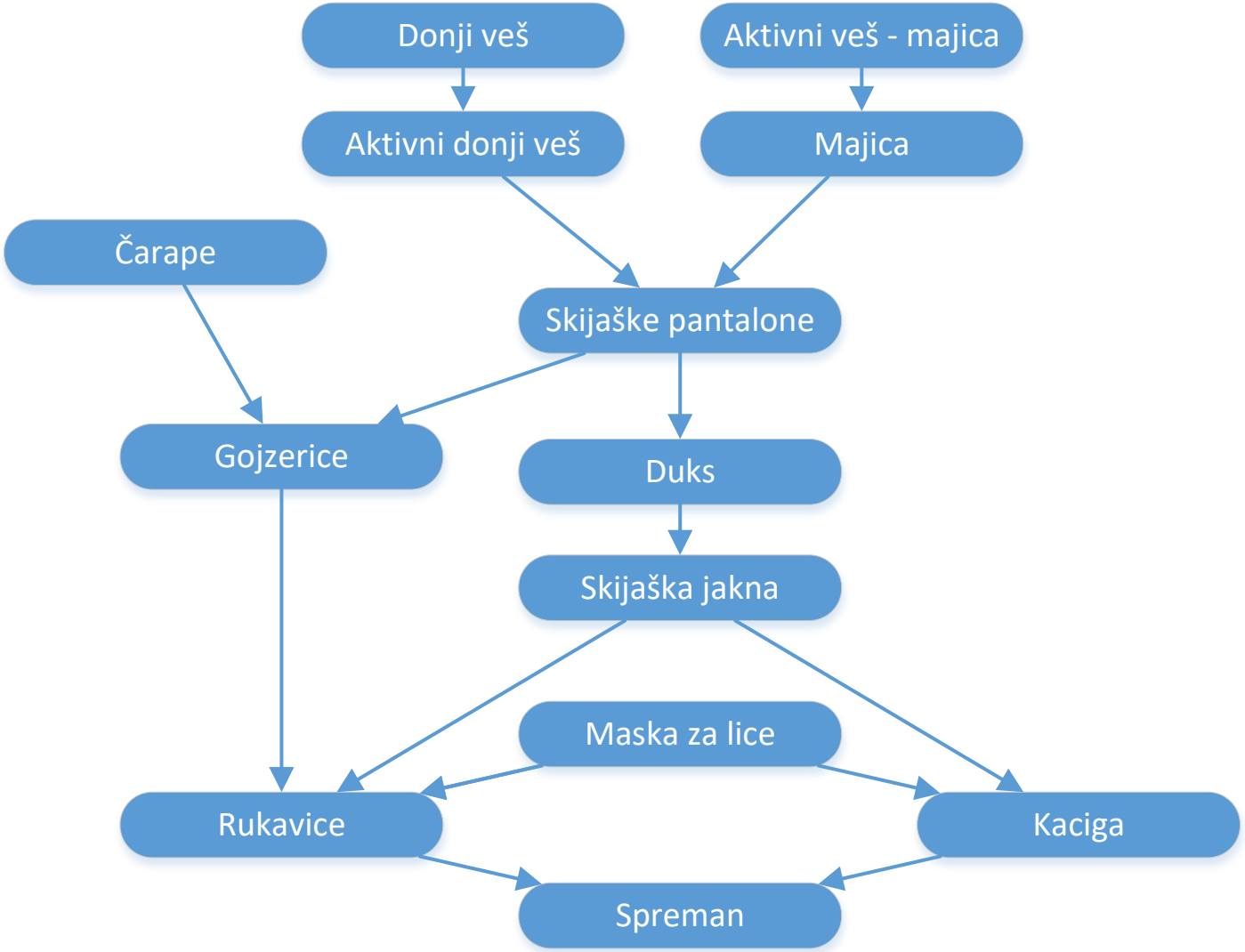


Težinski graf



Primer usmerenog acikličnog grafa

- Oblačenje skijaša ...



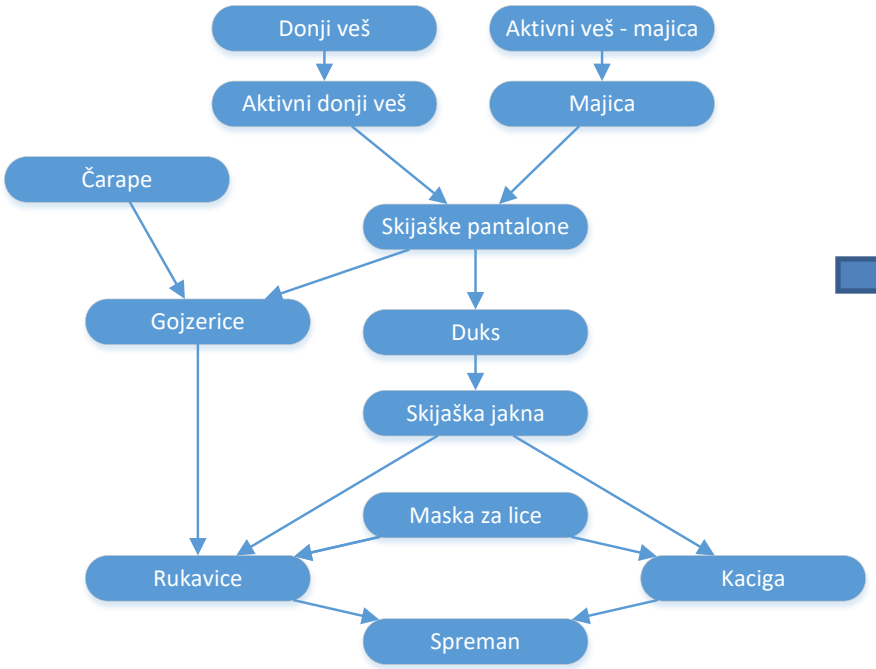
Topološko sortiranje

- Topološko sortiranje usmerenog acikličnog grafa proizvodi poredak čvorova gde je čvor u ispred čvora v kada ih spaja grana (u, v).
  - Rezultat je niz čvorova sortiran definisanim poretkom
  - Poredak ne mora biti jedinstven

- Primer: Redosled oblačenja skijaša
  1. Čarape, Donji veš, Aktivni veš – majica, Aktivni donji veš, Majica, Skijaške pantalone, Maska za lice, Gojzerice, Duks, Skijaška jakna, Kaciga, Rukavice, ili
  2. Donji veš, Aktivni veš – majica, Aktivni donji veš, Majica, Skijaške pantalone, Čarape, Duks, Skijaška jakna, Maska za lice, Gojzerice, Rukavice, Kaciga, ili
  3. ...

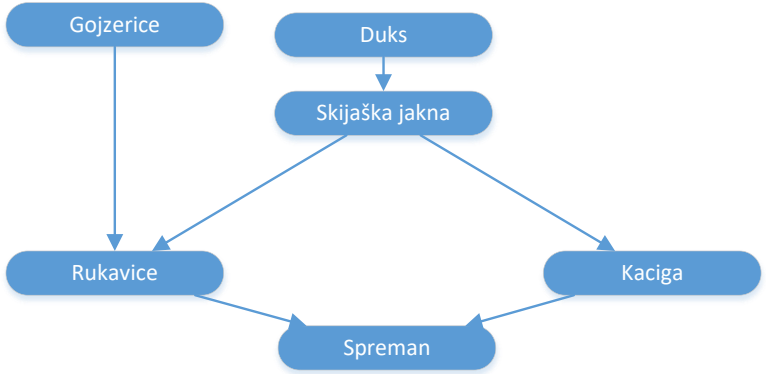
Topološko sortiranje – princip algoritma

1. Svakom čvoru pridružimo:
   - Broj ulaza (*in-degree*) == broj grana koje završavaju u čvoru
2. Biramo čvor bez ulaza (broj ulaza je 0)
   - Dodamo ga u izlazni niz
   - Uklonimo ga iz grafa
     Posledično: smanjimo broj ulaza u čvorovima sa kojima je povezan
3. Ponavljamo korak 2. (dok ima čvorova u njemu)



Primer:

1. Čarape
2. Donji veš
3. Aktivni veš – majica
4. Aktivni donji veš
5. Majica
6. Skijaške pantalone
7. Maska za lice
8. ...



Topološko sortiranje - algoritam

```
TOPOLOŠKO-SORTIRANJE(G)                    // G=G(V,E)
1  obraditi = ∅                            // skup čvorova bez ulaza koje treba „obraditi“
2  rez = []                                // rezultujući niz
3  for each u ∈ G.V                        // za svaki čvor prebroj ulaze
4    u.brul = 0
5  for each u ∈ G.V
6     for each v ∈ G.Susedi(u)             // susedni čvorovi – u koje se može otići iz u
7        v.brul = v.brul + 1
8  for each u ∈ G.V
9     if u.brul == 0                       // čvor bez ulaza ubaci se u obraditi
10       obraditi = obraditi ∪ u
11 while obraditi ≠ ∅                      // dok ima čvorova bez ulaza
12    u: obraditi = obraditi \ u           // uzmi jedan takav čvor
13    rez = [rez u]                        // dodaj ga na kraj rezultata
14    for each v ∈ G.Susedi(u)
15       v.brul = v.brul - 1
16       if v.brul == 0
17           obraditi = obraditi ∪ v
18 return rez
```

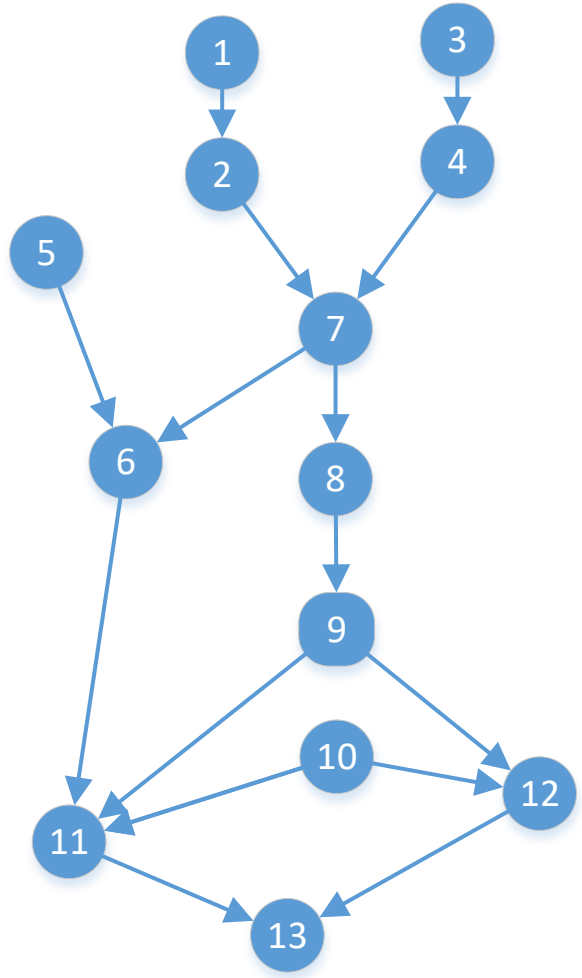
Predstavljanje grafova

Veze čvorova se mogu predstaviti:

1. **Matricom susedstva**
   - Jedna matrica za ceo graf
     - Memorijsko zauzeće $\Theta(|V|^2)$ - neefikasno za retke grafove (sadrži dosta nula)
   - Indeksi vrsta i kolona su indeksi čvorova
   - Sadrži $\{0, 1\}$ za direktne grafove, ili težine grana kod težinskih grafova
   - Simetrična je kod neusmerenih grafova
   - „Vide se“ čvorovi u koje se može otići iz posmatranog čvora, ali i čvorovi iz kojih se može doći u posmatrani čvor
2. **Listom susedstva**
   - Za svaki čvor se definiše lista čvorova sa kojima je povezan
     - Memorijsko zauzeće $\Theta(|V| + |E|)$
   - Memorijski efikasnije od matrice susedstva
   - „Vide se“ samo čvorovi u koje se može otići iz posmatranog čvora

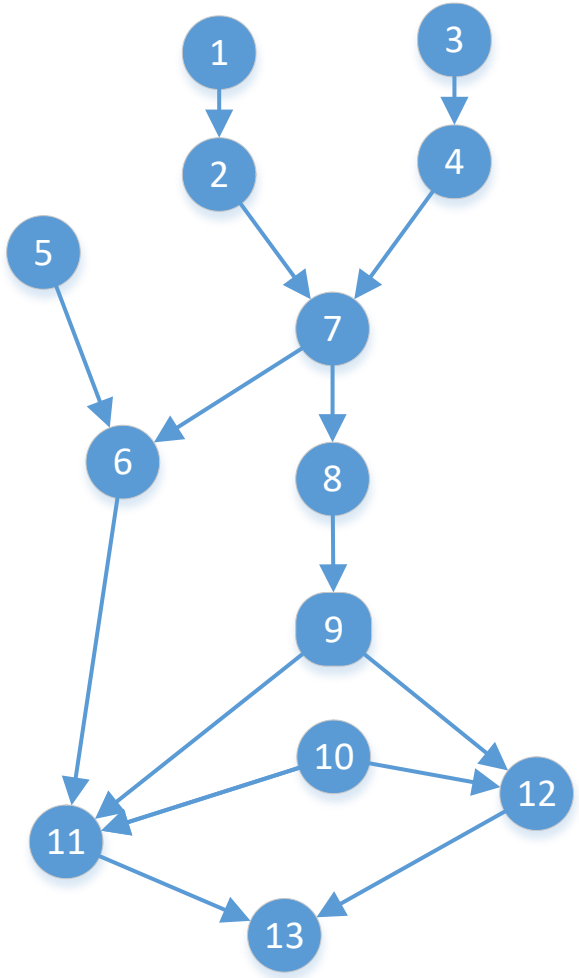
Matrica susedstva

od čvora \ ka čvoru	1	2	3	4	5	6	7	8	9	10	11	12	13
1	0	1	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	1	0	0	0	0	0	0
3	0	0	0	1	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	1	0	0	0	0	0	0
5	0	0	0	0	0	1	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	1	0	0
7	0	0	0	0	0	1	0	1	0	0	0	0	0
8	0	0	0	0	0	0	0	0	1	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0	1	1	0
10	0	0	0	0	0	0	0	0	0	0	1	1	0
11	0	0	0	0	0	0	0	0	0	0	0	0	1
12	0	0	0	0	0	0	0	0	0	0	0	0	1
13	0	0	0	0	0	0	0	0	0	0	0	0	0



Lista susedstva

čvor	ka čvorovima (lista)
1	2
2	7
3	4
4	7
5	6
6	11
7	6, 8
8	9
9	11, 12
10	11, 12
11	13
12	13
13	(prazno)



Vreme izvršavanja topološkog sortiranja

- Složenost algoritma je $\Theta(n+m), n = |V|, m = |E|$

```
TOPOLOŠKO-SORTIRANJE(G)                    // G=G(V,E)
1   obraditi = ∅
2   rez = []
3   for each u ∈ G.V                       // n prolaza
4     u.brul = 0
5   for each u ∈ G.V                       // n prolaza
6      for each v ∈ G.Susedi(u)            // ukupno m prolaza (nez obzira na n)
7         v.brul = v.brul + 1
8   for each u ∈ G.V                       // n prolaza
9      if u.brul == 0
10        obraditi = obraditi ∪ u         // dodaj u listu, O(1)
11  while obraditi ≠ ∅
12     u: obraditi = obraditi \ u          // n prolaza, u svakom se uklini iz liste O(1)
13     rez = [rez u]                       // O(1)
14     for each v ∈ G.Susedi(u)            // ukupno m prolaza
15        v.brul = v.brul - 1
16        if v.brul == 0
17            obraditi = obraditi ∪ v     // ukupno n – svaki čvor se jednom mora dodati
18  return rez
```

Algoritmi za obilazak grafa

- Obilazak grafa posećuje sve čvorove i grane grafa
- Algoritmi:
  - Pretraga u širinu - BFS (eng. *Breadth-First Search*)
  - Pretaraga u dubinu - DFS (eng. *Depth-First Search*)

Pretraga u širinu (BFS) - Osobine

- Jedan od najjednostavnijih algoritama pretrage grafova.
- Predstavlja osnovu drugim algoritmima (Dijkstrin algoritam, Primovo minimalno stablo razapinjanja, ...).

- Pretraga polazi od datog **izvornog čvora** i pokušava da dopre do svakog čvora koji je dostupan
- Nazvan je po načinu rada gde se „front“ pretrage širi tako da se nakon obilaska čvorova na rastojanju k od izvora nastavlja sa otkrivanjem čvorova na rastojanju $k + 1$.

- Za svaki doseziv čvor **daje najkraći put** (najmanji broj grana preko kojih se može doći u njega) polazeći od izvornog čvora.
- Algoritam radi i sa orijentisanim i neorijentisanim grafovima.

BFS način rada

- Pretraga polazi od zadatog čvora i obilazi sve njemu susedne čvorove (onekoji su na rastojanju 1), pa nastavlja sa njihovim susedima (koji su sada na rastojanju 2 od polaznog čvora) itd. Pri tome se pazi da se preskoče čvorovi koji su ranije posećeni.
- Stoga se tokom rada algoritma svaki čvor boji u:
  - **belo** – još nije uzet u obradu,
  - **sivo** – čeka na obradu, i
  - **crno** – obrađen.

  Tokom bojenja, svaki čvor (sem izvora) uvek tačno dva puta promeni boju: belo-sivo, sivo-crno.
- Algoritam izgrađuje **stablo pretrage u širinu** (eng. *Breadth-first tree*)
  - U korenu stabla je polazni (izvorni) čvor
  - Svaki drugi čvor ima vezu ka roditeljskom čvoru u stablu
    - Veza ka roditelju je predstavljena poljem *pred*

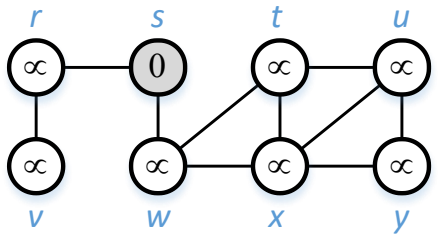
Čvorovi sadrže:
- boju - *boja*
- vezu ka prethodnom čvoru - *pred*
- rastojanje od polaznog čvora (u broju pređenih grana) - *d*

BFS algoritam

```
BFS(G, s)                          // s je polazni čvor
1   for each u ∈ G.V \ {s}         // za svaki čvor (sem s)
2      u.boja = BELA
3      u.d = ∞                     // d je rastojanje od izvora
4      u.pred = NIL
5   s.boja = SIVA                  // čvor izvora
6   s.d = 0
7   s.pred = NIL
8   Q = ∅                          // neobrađeni čvorovi
9   DODAJ(Q, s)                    // dodaj izvor u listu (red) neobrađenih
10  while Q ≠ ∅
11     u = PREUZMI(Q)              // uzmi prvi od neobrađenih
12     for each v ∈ G.Susedi(u)
13        if v.boja == BELA
14           v.boja = SIVA
15           v.d = u.d + 1
16           v.pred = u
17           DODAJ(Q, v)// dodaj u neobrađene
18     u.boja = CRNA
```

BFS primer

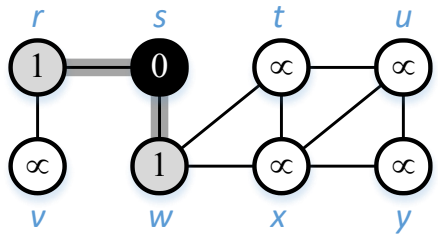
a)



Q: s

0

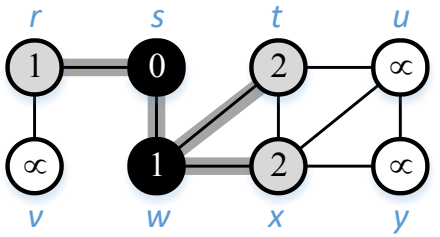
b)



Q: w r

1 1

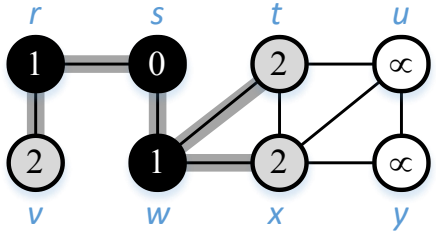
c)



Q: r t x

1 2 2

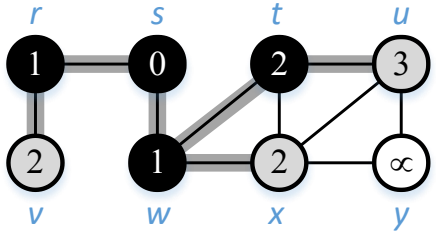
d)



Q: t x v

2 2 2

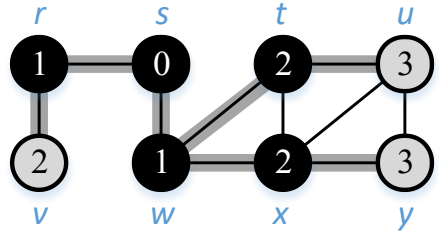
e)



Q: x v u

2 2 3

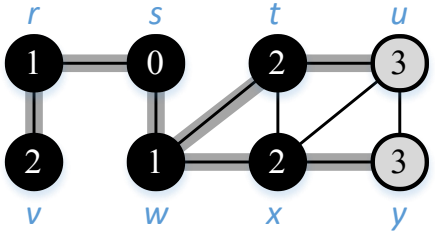
f)



Q: v u y

2 3 3

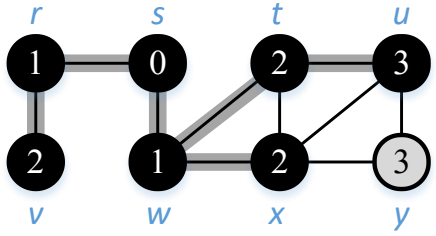
g)



Q: u y

3 3

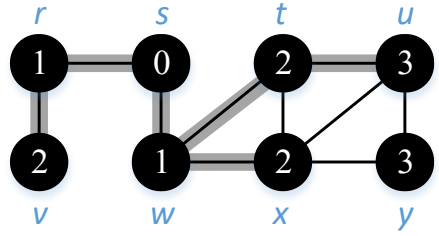
g)



Q: y

3

g)



Q: ∅

BFS vreme izvršavanja

- Posle inicijalizacije, algoritam nikada ne beli čvor, a na osnovu uslova (red #13) se dodaju u listu samo beli čvorovi.
- Ukupan broj DODAJ i PREUZMI operacija sa listom je n (svaka složenosti $O(1)$) .
- Kada se čvor ubaci u listu onda se skenira njegova lista susedstva. To se radi za svaki dostupan čvor te se skeniraju sve grane u grafu $O(m)$
- Ukupno vreme izvršavanja je $O(n+m)$, $n = |V|, m = |E|$

Stablo pretrage u širinu

- Stablo pretrage definiše putanje do svih (dosezivih) čvorova (polazeći od čvora s)
- BFS tokom rada izgrađuje stablo pretrage u širinu.
  - U primeru je prikazano debelim sivim vezama

- Formalna definicija na osnovu grafa $G = (V, E)$ i izvora s je:

$$G_\pi = (V_\pi, E_\pi)$$
$$V_\pi = \{v \in V : v.pred \neq \text{NIL}\} \cup \{s\}$$
$$E_\pi = \{(v.pred, v) : v \in V_\pi \setminus \{s\}\}$$

Stablo pretrage u širinu je takođe graf G_π opisan čvorovima V_π i granama E_π koji su podskupovi polaznog grafa $G = (V, E)$.

Ispisivanje putanje između čvorova

- Na osnovu rezultata BFS (stabla pretrage u širinu) i zatatog početnog i krajnjeg čvora može se ispisati putanja.

```
ISPIŠI-PUTANJU(G, s, v)
1  if v == s
2     print s
3  elseif  v.pred == NIL
4     print “nema putanje od “ s “ do “ v
5  else ISPIŠI-PUTANJU(G, s, v.pred)
6     print v
```

Pretraga u dubinu (DFS) – način rada

- Pretraga „prodire“ u graf što god dalje može udaljavajući se od izvornog (polaznog) čvora i ostavljajući „sa strane“ neistražene čvorove i grane. Kada dosegne do kraja pretraga se nastavlja od poslednje neistražene grane
- Algoritam formira stablo (šumu) pretrage u dubinu:

$$G_\pi = (V, E_\pi)$$

$$E_\pi = \{(v.pred, v) : v \in V \ \wedge \ v.pred \neq \text{NIL}\}$$

(primetiti da ovako definisano stablo pretrage uključuje sve čorove grafa V, za razliku of rezultata BFS, jer BFS kreće od zadatog čvora i mogu postojati delovi grafa koji nisu dosezivi – tj. graf je nepovezan)

- Ovde su dodate dve vremenske značke:
  - kada je čvor otkriven (.*d*) i
  - kada je obrada čvora završena (.*f*).
    (vremenske značke su brojevi 1,2,3...2|V|)

Čvorovi sadrže:
- boju – *boja*
- vezu ka prethodnom čvoru – *pred*
- trenutak kada je čvor posećen – *d*
- trenutak kada je završena obrada čvora – *f*

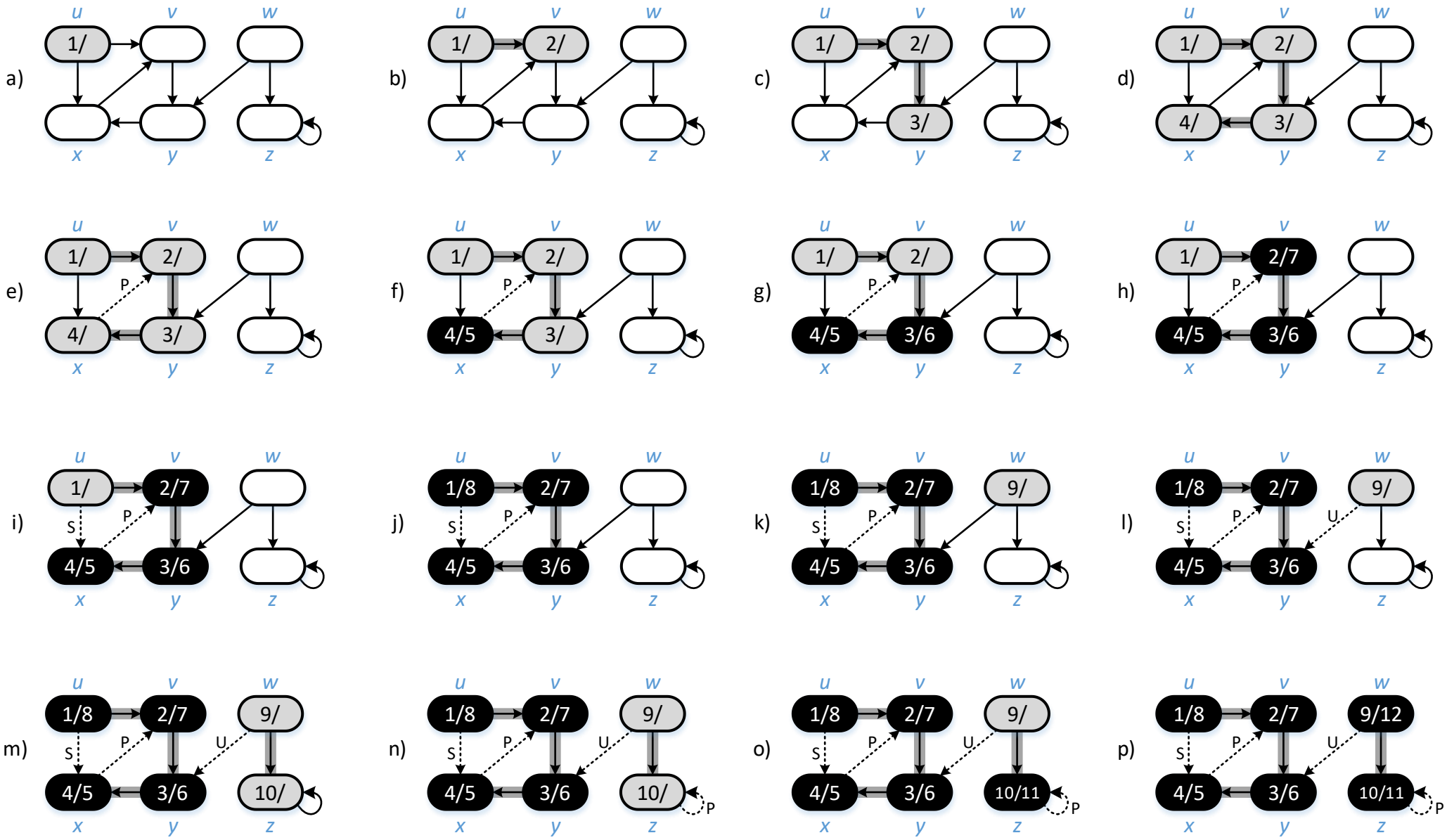
DFS algoritam

- Rekurzivno pretražuje graf
- Pazi da ne ponovi posetu ranije posećenim čvorovima (i ovde rešeno „bojenjem“)

```
DFS(G)
1  for each u ∈ G.V
2     u.boja = BELA
3     u.pred = NIL
4  vreme = 0
5  for each u ∈ G.V
6     if u.boja == BELA
7        DFS-POSETI(G,u)
```

```
DFS-POSETI(G,u)
1  vreme = vreme + 1
2  u.d = vreme
3  u.boja = SIVA
4  for each v ∈ G.Susedi(u)
5     if v.boja == BELA
6        v.pred = u
7        DFS-POSETI(G,v)
8  u.boja = CRNA
9  vreme = vreme + 1
10 u.f = vreme
```

DFS primer



DFS vreme izvršavanja

- Vreme izvršavanja DFS algoritma je $\Theta(n+m)$, $n = |V|$, $m = |E|$ - isto kao kod BFS (iz istih razloga).
  - Pri tome se smatra da se svi čvorovi nalaze u nizu i da su veze čvorova modelovane listama susedstva koje se takođe nalaze u nizu

DFS i stablo (šuma) pretrage u dubinu

Činjenice:

- Za svaki čvor u je $u.d < u.f$
- interval $[v.d, v.f]$ čvora v se nalazi unutar intervala $[u.d, u.f]$ čvora u ako je čvor v naslednik u (na istoj su putanji stabla pretrage u dubinu gde se od u dolazi u v), tj. važi da je $u.d < v.d < v.f < u.f$
- Ako se ovi intervali $[v.d, v.f]$ i $[u.d, u.f]$ ne seku (preklapaju) onda jedan čvor nije naslednik drugog čvora u stablu pretrage u dubinu

- Ove osobine se mogu upotrebiti za klasifikaciju grana ...

Klasifikacija grana na osnovu DFS

U usmerenom grafu:

- **Grane stabla** (eng. *tree edge*) – grane preko kojih se posećuju čvorovi (u primeru debele sive grane) – grana koja završava u belom čvoru
- **Povratne grane** (eng. *back edge*) – ka prethodnom čvoru u stablu (u primeru označene sa „P“) – grana koja završava u sivom čvoru
- **Preskočna grana** (eng. *forward edge*) – ka čvoru u stablu dalje od tekućeg (u primeru označene sa „S“) – grana (u, v) koja završava u crnom čvoru i $u.d < v.d$
- **Unakrsne grane** (eng. *cross edge*) – između dva podstabla (koja nemaju zajedničke čvorove - pretke) (u primeru označene sa „U“) – grana koja završava u crnom čvoru i $u.d > v.d$

Koji od tipova grana postoje u neusmerenom grafu?

Detekcija kružne putanje

- Kružna putanja postoji kada se kratanjem po grafu naiđe na ranije posećen čvor.
- Da li graf ima kružnu putanju (ciklus)? - je važno pitanje

- Detekciju vrši DFS
- Graf ima kružnu putanju ako postoji povratna grana!
  - Kružna putanja se dobija, polazeći od povratne grana, prateći grane stabla (*tree edge*)

Topološko sortiranje

- Topološko sortiranje **direktnog acikličnog grafa** (DAG) se može sprovesti na osnovu DFS algoritma - posmatraju se vremena završetaka kod obilaska čvorova
- Ideja: posmatra se grana (u, v) i mogući slučajevi su:
  - ako je v beo, onda je $v.f < u.f$ (prvo se završi obilazak v)
  - v ne može biti sivi čvor jer DAG nema kružne putanje (cikluse)!
  - ako je v crn, onda je $v.f < u.f$ (završen je obilazak v, a u još traje)

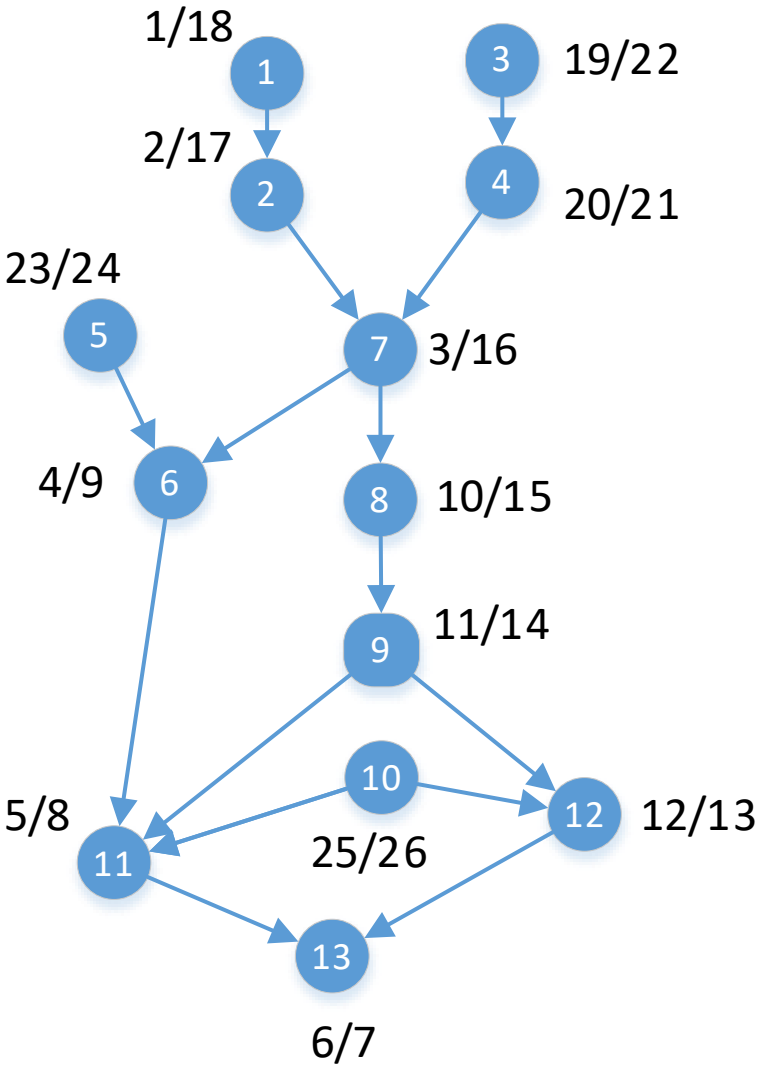
  Znači čvorovi u koje ulaze grane DAG-a imaju manji $.f$
- Algoritam:

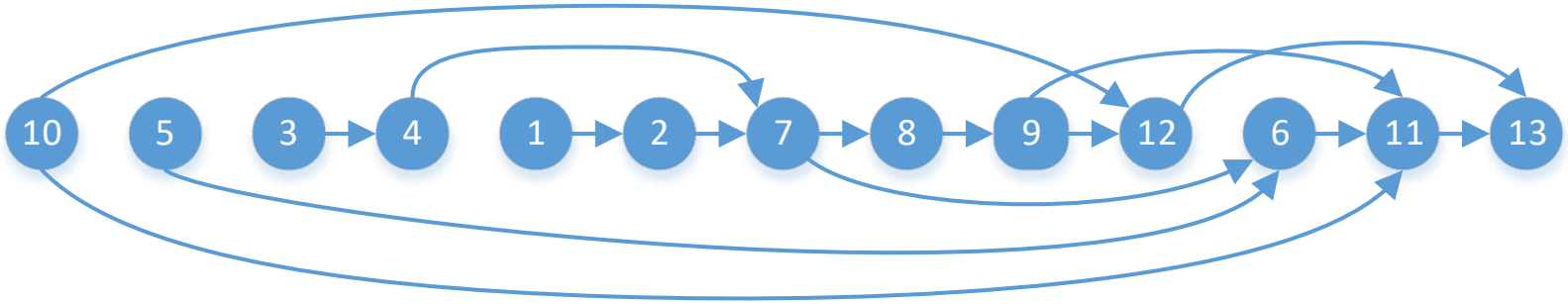
```
TOPOLOŠKO-SORTIRANJE(G)
1  Pozvati DFS da izračuna v.f za svaki čvor v.
2  Kako je čvor obrađen smestiti ga u listu L
3  Obrnutu listu L vratiti kao rezultat.
```

Ranije je rađen algoritam topološkog sortiranja, a ovo je drugačija ideja – algoritam.

- Vreme izvršavanja algoritma je $\Theta(n + m)$, $n = |V|$, $m = |E|$ zbog DFS poziva, dok je $O(1)$ trajanje smeštanja svakog od $|V|$ elemenata u listu

Primer topološkog sortiranja





1. Donji veš
2. Aktivni donji veš
3. Aktivni veš – majica
4. Majica
5. Čarape
6. Gojzerice
7. Skijaške pantalone
8. Duks
9. Skijaška jakna
10. Maska za lice
11. Rukavice
12. Kaciga
13. Spreman

→

10. Maska za lice
5. Čarape
3. Aktivni veš – majica
4. Majica
1. Donji veš
2. Aktivni donji veš
7. Skijaške pantalone
8. Duks
9. Skijaška jakna
12. Kaciga
6. Gojzerice
11. Rukavice
13. Spreman

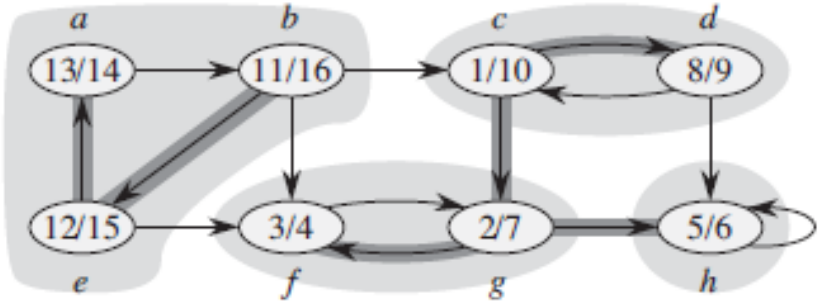
Čvrsto povezane komponente usmerenog grafa

(engl. *Strongly connected components*)

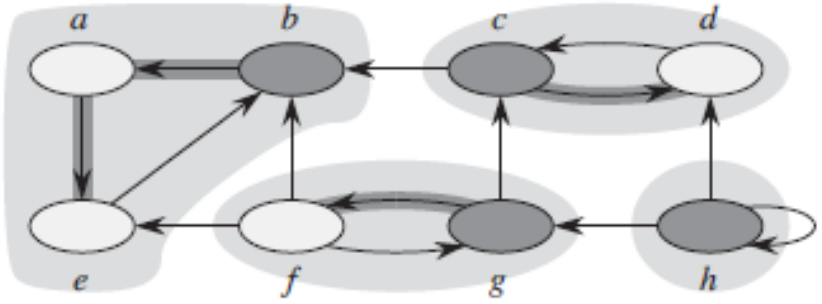
- Radi se o razlaganju grafa na grupe čvorova koju su „čvrsto povezani“ tako da se svaka takva grupa može predstaviti čvorom nastalog DAG-a
- Čvrsto povezani čvorovi u grupi imaju osobinu da postoji putanja između svakog para čvorova, tj. iz svakog čvora u grupi se može stići u svaki drugi

- Posledica: dobijeni graf grupa čvorova nema petlje (jer da ima onda bi ti čvorovi bili u zajedničkoj grupi)

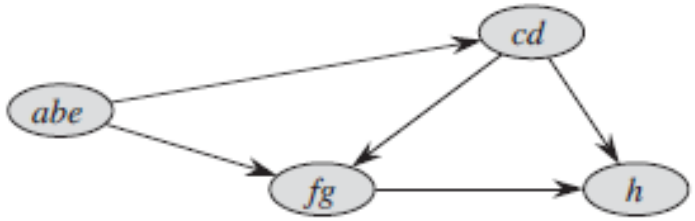
(a)



(b)



(c)



Algoritam za nalaženje čvrsto povezanih komponenti

- Ideja se zasniva na osobini da graf $G = (V, E)$ i od njega dobijen drugi graf $G^T = (V, E^T)$, gde su sve grane obrnule smer, imaju iste čvrsto povezane komponente

Algoritam:

ČVRSTO-POVEZANE-KOMPONENTE(G)

1 *Pozvati DFS da izračuna v.f za svaki čvor v.*
2 *Odrediti* G^T
3 *Ponovo pozvati DFS ali u glavnoj petlji obilaziti čvorove po opadajućem v.f*
4 *Ispisati grupu čvorova svakog stabla koje je posećeno u glavnoj petlji*

Objašnjenje načina rada alg. čvrsto povezanih komponenti

- U grafu $G = (V, E)$ posmatramo dve grupe čvrsto povezanih komponenti A i B i granu $(u, v) \in E$ koja jedina povezuje čvorove iz različitih grupa $u \in A, v \in B$.
- Kada se izvršava DFS prvi put i polazi iz nekog čvora $x \in A$ onda će $x.f$ biti najveće za sve čvorove u A i B. Ako se polazi iz nekog čvora $y \in B$ onda će svi čvorovi u B biti završeni pre nego se počne sa čvorovima u A, tj. opet je najveće $.f$ u A
- Kada se okrenu smerovi grana, onda $(v, u) \in E^T$
- Ponovo se izvršava DFS (2. put), ali se svakako kreće sa čvorom iz A (jer je njegovo $.f$ veće od svih u B). Kako i kod E^T grana postoje veze svih čvorova u A, svi oni će biti obiđeni pre nego se pređe na prvi čvor u B (nema grane ka B).
  Svi čvorovi obiđeni u ugnježdenim pozivima DFS-Poseti pripadaju grupi A.
- Nastavlja se sa nekim čvorom iz grupe B: obilaze se svi čvorovi u B i formiraju grupu B.
  Posećuje se i grana (v, u) ali se ignoriše jer završava u čvoru iz grupe A koja je već obiđena.

Minimalno stablo razapinjanja

(engl. *Minimum Spanning Tree* (MSP))

- Posmatra se **neusmeren težinski** povezan graf $G = (V, E)$, $w: E \to \mathbb{R}$
- Minimalno stablo razapinjanja je aciklični graf sačinjen od grana $T \subseteq E$ koje povezuju sve čvorove V tako da je suma težina u T minimalna, tj.

$$\min_T w(T) = \min_T \sum_{(u,v)\in T} w(u,v)$$

- Primeri:
  - Imamo grupu sela između kojih želimo da izgradimo mrežu puteva. Ako znamo cenu izgradnje puta između svakog para sela, možemo izgraditi deo tih puteva tako budu povezana sva sela ali da je cena izgradnje najmanja
  - Domaćinstva ruralnih krajeva povezujemo električnim vodovima ili cevovodima.
  - Povezujemo elektronske komponente „ožičavanjem“ na pločici.
  - …

Ideja izgradnje minimalnog stabla razapinjanja

- Do rešenja se može doći postepenom izgradnjom stabla razapinjanja, tj. dodavanjem po jedne grane u njega
  - u teoriji je ovo poznato kao „pohlepna“ strategija, gde se u datom trenutku pravi najbolji izbor
- Stablo razapinjanja $A \subseteq T$ se izgrađuje dok ne poveže sve čvorove
  - u osnovi, izgradnja može imati više stabala koje se povezuju u jedno
- Nezgodan deo je izbor „sigurne grane“ $(u, v) \in T$
  - sigurna grana ne može povezivati čvorove istog stabla A

```
MSR-GENERIČKI(G,w)
1  A = ∅
2  while A ≠ stablo razapinjanja
3     Pronađi „sigurnu granu“ (u,v)
4     A = A ∪ {(u,v)}
5  return A
```

Algoritmi minimalnog stabla razapinjanja

Posmatramo dva algoritma:

- **Kruskal** – na početku ima šumu stabala od po jednog čvora (bez grana) i bira (kao sigurnu granu) granu najmanje težine koja povezuje dva stabla
- **Prim** – izgrađuje jedno stablo tako što bira (kao sigurnu granu) najlakšu granu koja povezuje stablo sa čvorom koji nije u stablu

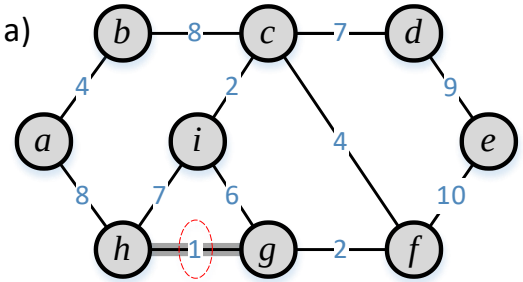
Kruskalov algoritam

```
MSR-KRUSKAL(G, w)
1  A = ∅
2  for each v ∈ G.V          // svaki čvor postavi u zaseban skup
3     NAPRAVI-SKUP(v)
4  Sortiraj sve grane po težini (u rastućem redosledu)
5  for each (u,v) ∈ sort(G.E)
6     if PRONAĐI-SKUP(u) ≠ PRONAĐI-SKUP(v)  // da li je grana sigurna?
7        A = A ∪ {(u,v)}
8        UNIJA-SKUPOVA(u,v)   // spoj skupove gde su bili čvorovi u i v
9  return A
```

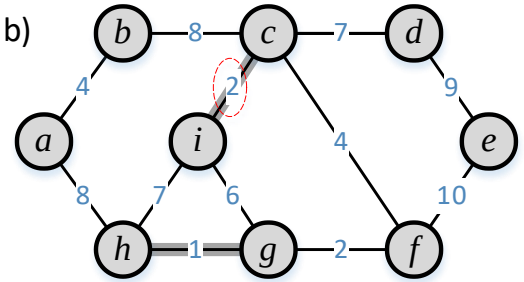
- Algoritam koristi operacije sa disjunktnim skupovima: formiranje, pretraga po skupovima, unija.
  - Složenost algoritma zavisi od implementacije ovih operacija
  - Može se postići složenost: $O(m \log_2 n)$, $|V| = n$, $|E| = m$

Primer Kruskal

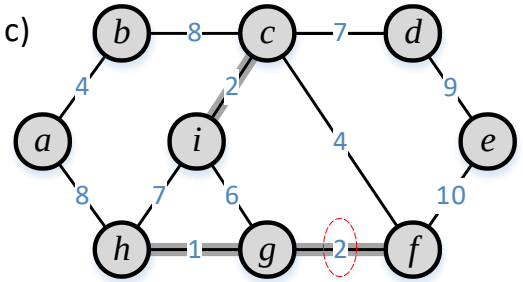
a)



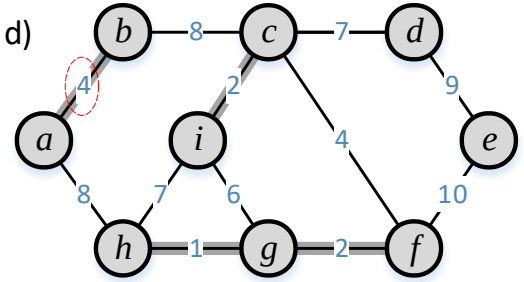
b)



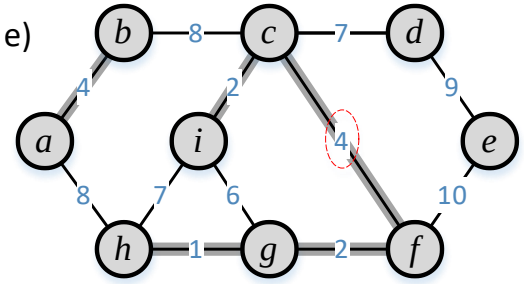
c)



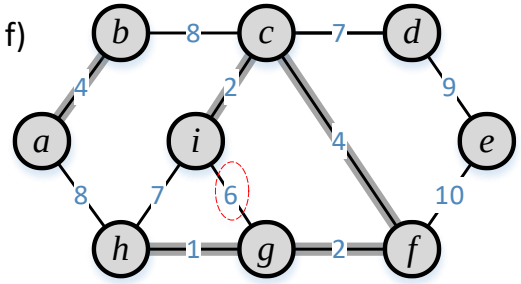
d)



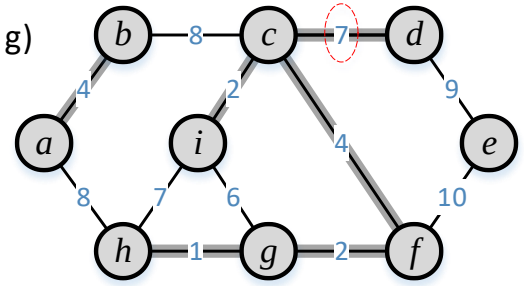
e)



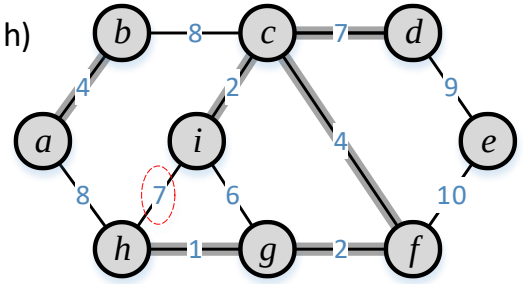
f)



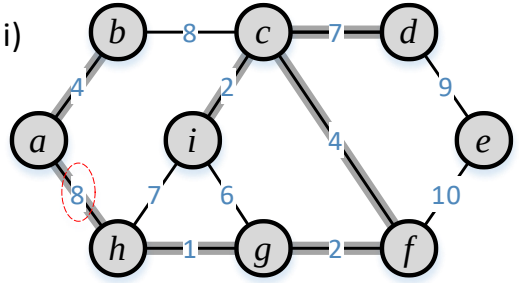
g)



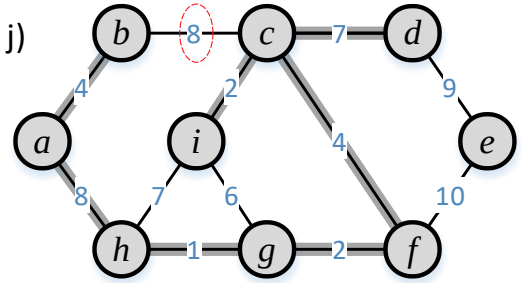
h)



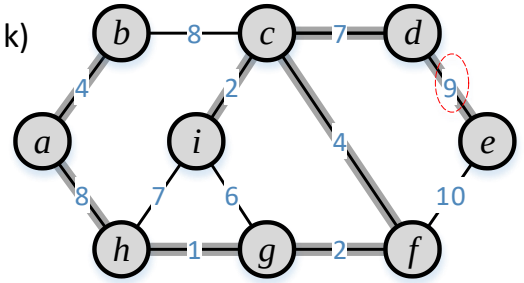
i)



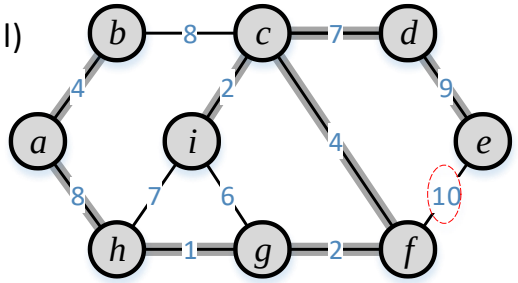
j)



k)



l)



$$\sum w = 37$$

Rad sa disjunktnim skupovima

- Posmatramo strukture podataka koje olakšavaju rad sa disjunktnim skupovima.
- Posmatraju se dinamički skupovi u kojima se ne ponavljaju elementi (objekti), tj. skup od skupova $S = \{S_1, S_2, \dots, S_k\}$
- Svaki skup $S_i, i = 1 \dots k$ se identifikuje jednim elementom (predstavnikom).
  - To može biti bilo koji element iz skupa (npr. sa najmanjim ključem), ali uvek isti kod uzastopnih poziva
- Bitne operacije (x je elemenat, tj. objekat):
  - NAPRAVI-SKUP(x) – napravi novi skup čiji je jedini element x
  - UNIJA(x,y) – objedinjuje elemente skupova x i y u jedan skup (a x i y logički uklanja)
  - PRONAĐI-SKUP(x) – vraća pokazivač na predstavnika (logički na skup u kome se nalazi)

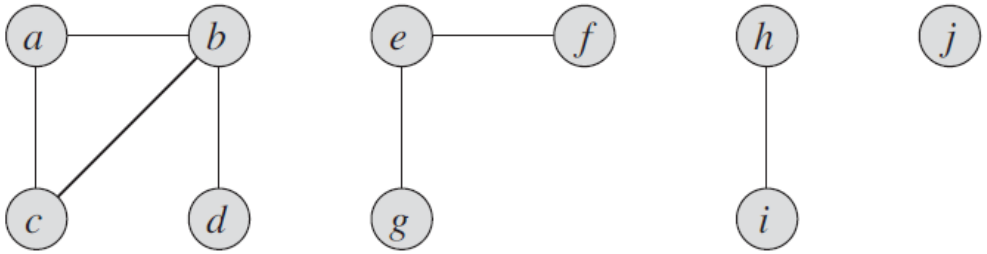
- Kako realizovati ove operacije da budu efikasne? (vidi naredne slajdove)

Primer rada sa disjunktnim skupovima

- Posmatra se neusmereni netežinski graf.
- Odrediti grupe povezanih čvorova.

```
POVEZANE-KOMPONENTE(G)
1  for each v ∈ G.V
2     NAPRAVI-SKUP(v)
3  for each (u,v) ∈ G.E
4     if PRONAĐI-SKUP(u) ≠ PRONAĐI-SKUP(v)
5        UNIJA(u,v)
```

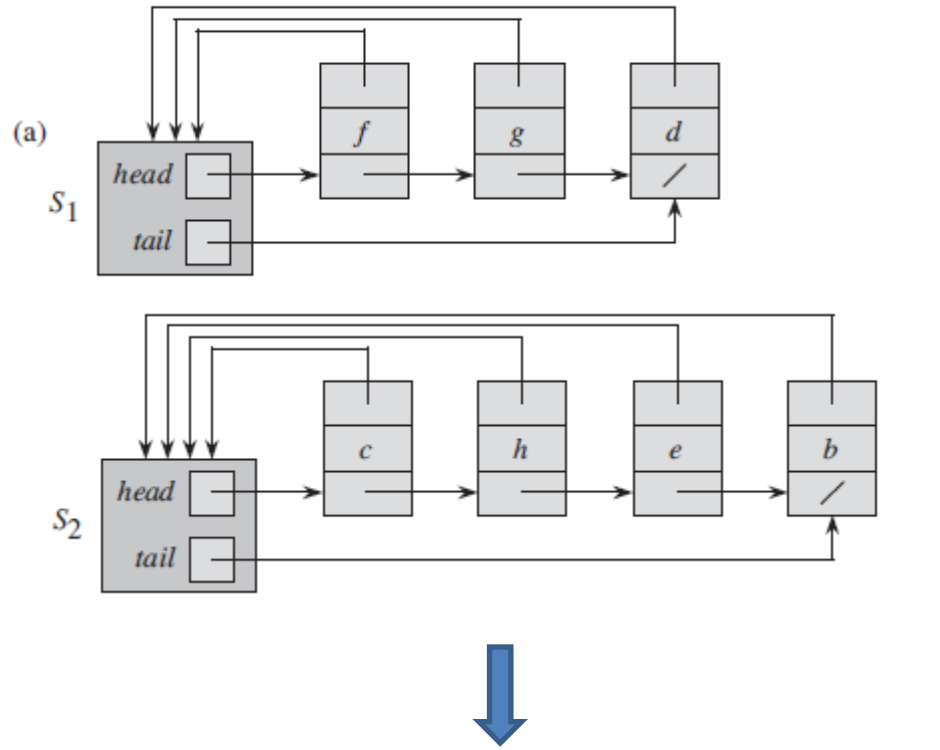
```
ISTE-KOMPONENTE(G)
1  if PRONAĐI-SKUP(u) == PRONAĐI-SKUP(v)
2     return true
3  else return false
```

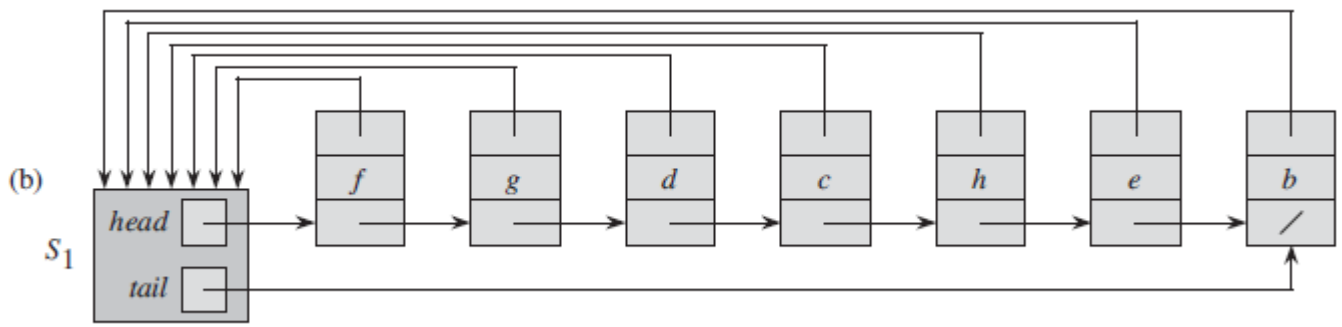


Primer:
Primeniti algoritam na graf koga čine čvorovi $\{a, b, c, d, e, f, g, h, i, j\}$ i grane $\{(a, b), (e, f), \ldots\}$

Realizacija disjunktnih skupova preko povezanih lista

- Svaki skup ima prilagođenu strukturu jednostruko spregnute liste elemenata
  - sa pokazivačem na gravu i pokazivačem na rep liste
  - dodatno, svaki elemenat pokazije na strukturu skupa
- Složenosti operacija:
  - NAPRAVI-SKUP(x): $O(1)$
  - UNIJA(x,y): $O(n)$ gde je n broj elemenata u skupu, jer se ažuriraju pokazivači elemanta na strukturu skupa
  - PRONAĐI-SKUP(x): $O(1)$ jer svaki elemenat ima pokazivač na strukturu, a onda npr. glava pokazuje na predstavnika (prvi elemenat u listi)



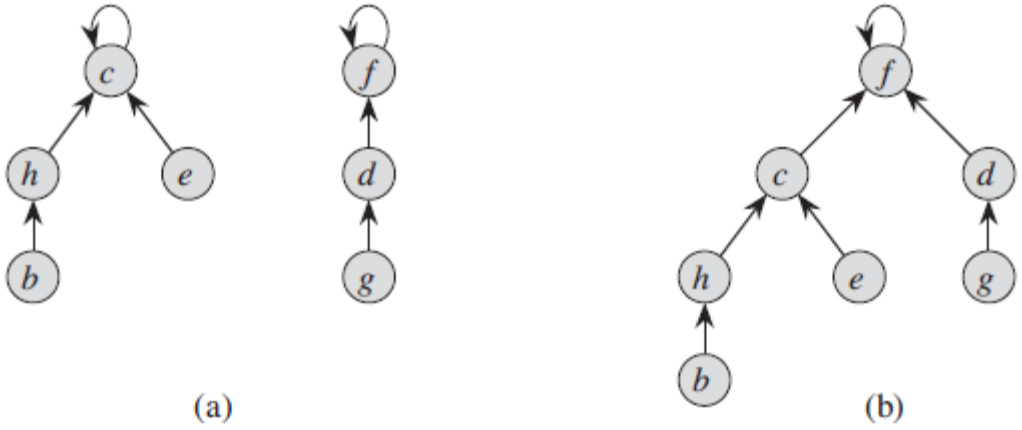


Realizacija disjunktnih skupova preko povezanih lista (2)

- Očito, unija je skupa operacija i može se „pametnije“ implementirati upotrebom heuristike:
  - vodi se evidencija o broju elemenata u skupu, tako da se elementi manjeg skupa prebaciju u veći
- Ovim se unija skupova jednakih veličina (i dalje) izvršava u $\Omega(n)$, ali se u tipičnoj primeni gde ima m (posmatra se sekvenca NAPRAVI-SKUP, UNIJA, PRONAĐI-SKUP) operacija sa n elemenata dobija ukupno vreme izvršavanja $O(m + n \log n)$
  - $O(m)$ puta se izvrše $O(1)$ operacije NAPRAVI-SKUP i PRONAĐI-SKUP, dok se unija n skupova primenom ove heuristike svodi na $O(n \log n)$

Realizacija disjunktnih skupova preko šume (stabala)

- Ovde se za svaki skup koristi stablo (sa jednim korenom) čiji elementi imaju samo pokazivač na roditelja.
  - Predstavnik skupa je elemenat u korenu stabla
  - Unija dva skupa koren jednog skupa postavi da pokazuje na koren drugog skupa (ili obrnuto)



Realizacija disjunktnih skupova preko šume (stabala) (2)

- Dodavanjem dve heuristike:
  - Visine svakog elementa u stablu (tzv. rang čvora), omogućava da se u uniji manje stablo poveže na veće.
  - „Kompresija putanje“: poziv PRONAĐI-SKUP se proširi tako da se kretanjem po putanji od elementa preko njegovih predaka dolazi do korena (predstavnika) i tokom tog kretanja se svaki procesirani roditeljski čvor postavi da pokazuje na korenski element.
  - Obe ove heuristike dovode do amortizovane složenosti (prethodno definisane sekvence m operacija) $O(m\,\alpha(n))$, gde je $\alpha(n)$ jako sporo rastuća funkcija (u praksi $\alpha(n) \leq 4$), što je u praksi vreme za m operacija linearno srazmerno sa m (striktno govoreći superlinearno)

```
NAPRAVI-SKUP(x)
1   x.r = x
2   x.rang = 0

UNIJA(x,y)
1   POVEŽI(PRONAĐI-SKUP(x), PRONAĐI-SKUP(y))

POVEŽI(x,y)
1   if x.rang > y.rang
2         y.r = x
3   else  x.r = y
4      if x.rang == y.rang
5         y.rang += 1

PRONAĐI-SKUP(x)
1   if x ≠ x.r
2      x.r = PRONAĐI-SKUP(x.r)
3   return x.r
```

Složenost Kruskal algoritma

```
MSR-KRUSKAL(G, w)
1   A = ∅                                          O(n)
2   for each v ∈ G.V                               O(1)
3      NAPRAVI-SKUP(v)                             O(m log m)
4   Sortiraj sve grane po težini                   redovi 5-8: O((n + m)α(n))
5   for each (u,v) ∈ sort(G.E)
6      if PRONAĐI-SKUP(u) ≠ PRONAĐI-SKUP(v)
7         A = A ∪ {(u,v)}
8         UNIJA-SKUPOVA(u,v)
9   return A
```

- Detaljnija analiza daje: $\alpha(n) = O(\log n) = O(\log m)$, $m \geq n - 1$ pa su ukupna složenost rada sa skupovima $O(m \log m)$, a to je i ukupna složenost $O(m \log m)$
- Dodatno za grafove koji nisu gusti $m < n^2$, $\log m = O(\log n)$ i složenost algoritma je: $\boldsymbol{O(m \log_2 n)}$, $|V| = n$, $|E| = m$

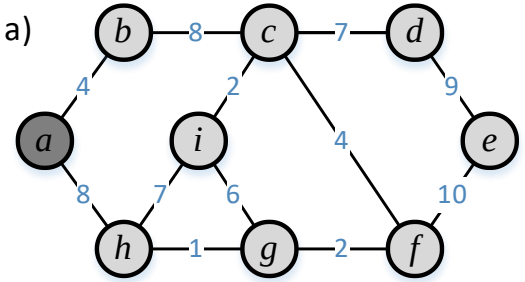
Primov algoritam

- Radi slično Dajsktra algoritmu

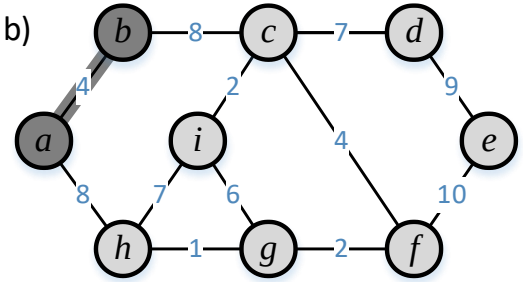
```
MSR-PRIM(G, w, r)              // r je polazni čvor
1   for each u ∈ G.V
2      u.ključ = ∝
3      u.pred = Nil
4   r.ključ = 0
5   Q = G.V
6   while Q ≠ ∅
7      u = IZDVOJ-MINIMUM(Q)     // po ključu
8      for each v ∈ G.Susedi(u)
9         if v ∈ Q & w(u,v) < v.ključ
10           v.pred = u
11           v.ključ = w(u,v)
```

Primer Prim

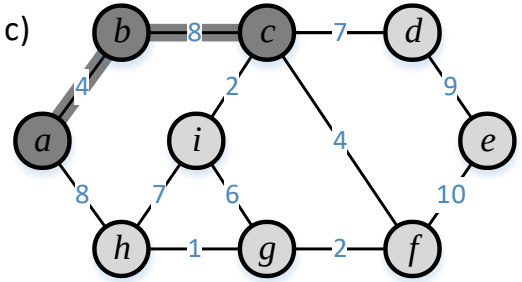
a)



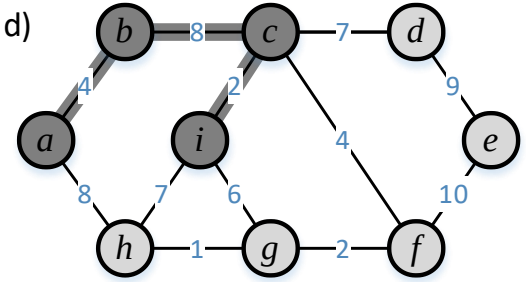
b)



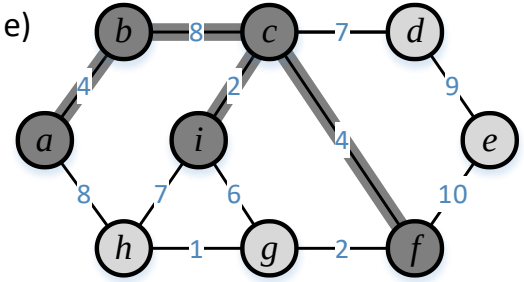
c)



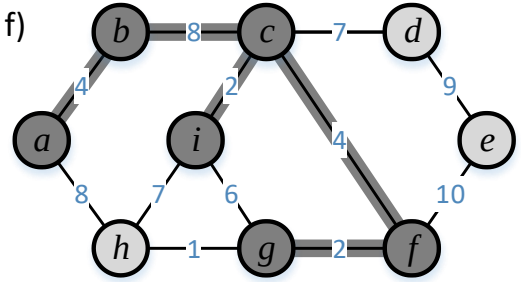
d)



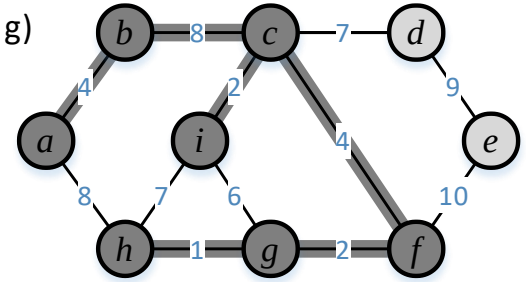
e)



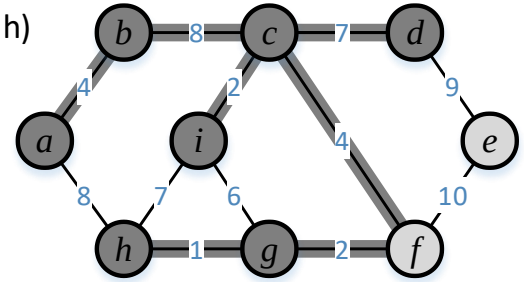
f)



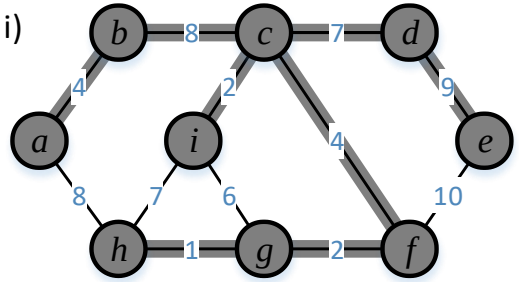
g)



h)



i)



$$\sum w = 37$$

Složenost Prim algoritma

```
MSR-PRIM(G, w, r)
1   for each u ∈ G.V
2       u.ključ = ∞
3       u.pred = Nil
4   r.ključ = 0
5   Q = G.V                                  Pretpostavka: koristimo min-hip. Izgradnja hipa je O(n)
6   while Q ≠ ∅                              petlja O(n)
7       u = IZDVOJ-MINIMUM(Q)                hip operacija O(log n)
8       for each v ∈ G.Susedi(u)             ukupno za sve čvorove ima 2m prolaza
9           if v ∈ Q & w(u,v) < v.ključ       v ∈ Q je provera u O(1) ako koristimo dodatni bit po čvoru
10              v.pred = u
11              v.ključ = w(u,v)             operacije umanjivanja ključa na hipu O(log n)
```

- Sumarno: redovi 7 u petlji $O(n \log n)$ i red 11 u petlji $O(m \log n)$ za $m > n - 1$ daju ukupno složenost $\boldsymbol{O(m \log_2 n)}$ kada se koristi min-hip struktura, $|V| = n$, $|E| = m$
- Uz upotrebu Fibonačijevog hipa dobija se $O(m + n \log_2 n)$

Težinski graf i najkraći put

- Posmatra je usmereni težinski graf $G = (V, E, w)$ gde su težine grana date funkcijom $w : E \to \mathbb{R}$
- Težina $w(p)$ putanje $p = \langle v_0, v_1, \dots, v_k \rangle$ je suma težina grana te putanje

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

- Najkraći put od čvora u do čvora v se definiše kao

$$\delta(u, v) = \begin{cases} \min\left\{w(p) : u \xrightarrow{p} v\right\}, ako\ postoji\ putanja \\ \infty, inače. \end{cases}$$

Problem: Naći p sa najmanjom težinom.

Najkraći put u grafu

BFS nalazi najkraći put u netežinskog grafu

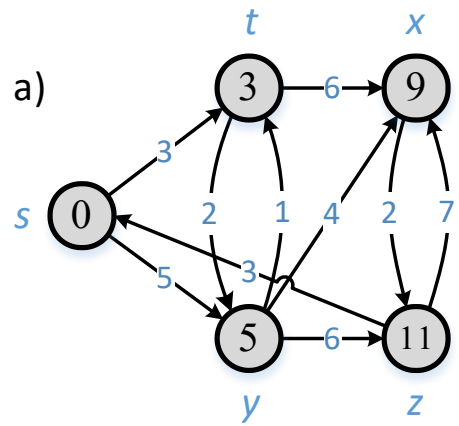
- razmatra se broj grana između čvorova

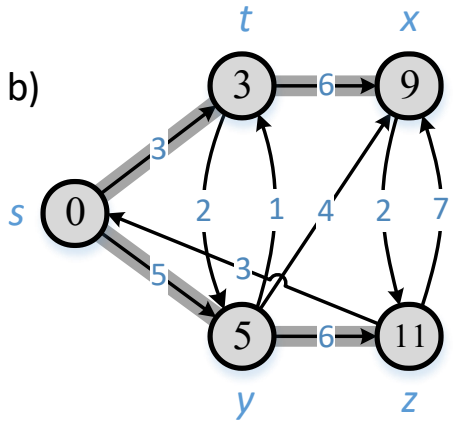
Najkraći put u težinskom grafu – računat od jednog čvora-izvora:

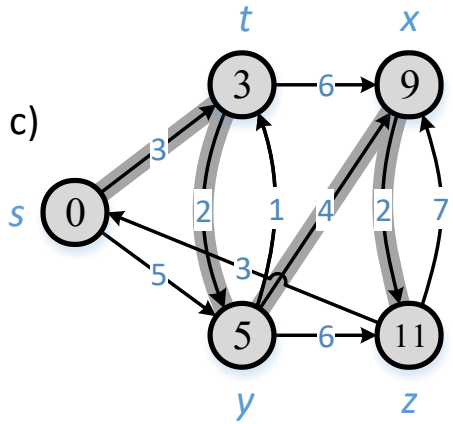
- **Dajkstra algoritam** (originalan naziv *Dijkstra*)
  - ograničava težine na nenegativne težine grana
  - složenost $O(|V|\log_2|V| + |E|)$
- **Belman-Ford algoritam** (originalan naziv *Bellman-Ford*)
  - graf može imati i pozitivne i negativne težine grana
  - složenost $O(|V||E|)$
  - detektuje kružne putanje negativnog pojačanja
- ne mora da ima najmanji broj grana
- složenost algoritma ne zavisi od težina w.

Najkraći put u težinskom grafu – računat od između svih čvorova

Primer najkraćeg puta







- Ažurira se tekuće rastojanje: $v.d$

Na kraju:

- Za najkraći put važi da je: $v.d = \delta(s, v)$
- Prethodni čvor je $v.pred \equiv \pi(v)$

  Najkraća putanja je $v - \pi(v) - \pi\big(\pi(v)\big) - \pi\Big(\pi\big(\pi(v)\big)\Big) - \cdots - s$

Varijante najkraćeg puta

1. Najkraći put od zadatog izvornog čvora do svih ostalih čvorova u grafu – osnovni problem.
2. Najkraći put do zadatog čvora od svih ostalih čvorova - problem koji se rešava kao osnovni problem.
3. Najkraći put između zadatih izvora i odredišta – rešava se kao osnovni problem (i očitava se rešenje za zadato odredište)
4. Najkraći put između svih čvorova u grafu – može se rešiti upotrebom osnovnog problema za svaki čvor u grafu, ali se rešava posebnim algoritmima (npr. *Floyd-Warshall*)

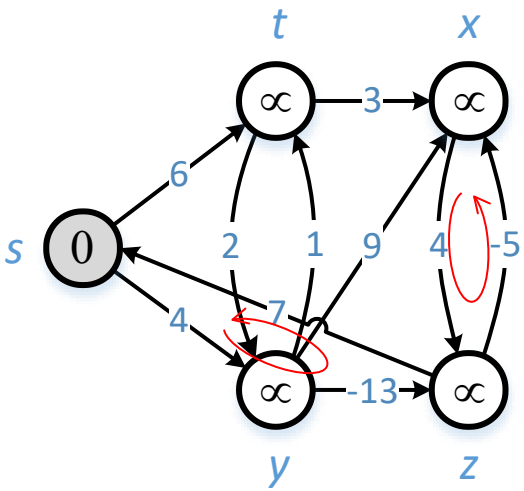
Poteškoće određivanja najkraćeg puta

- Grane sa negativnim težinama
  - Zašto postoje grane sa negativnim težinama?
    (da ih nema, bilo bi lakše)
  - Da li se nekom pogodnom transformacijom negativne težine mogu predstaviti pozitivnim?
- Kružne putanje sa negativnim pojačanjem
  - Da li mogu uzrokovati da se algoritam ne završi, jer se $v.d$ neprekidno smanjuje?

Primer kružnih putanja sa negativnim pojačanjem:

$$7 + 4 - 13 < 0$$

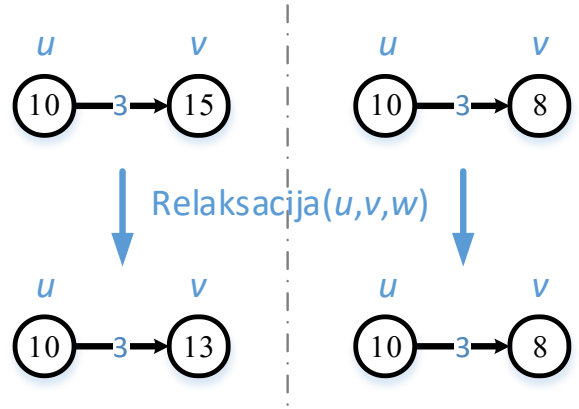
$$4 - 5 < 0$$



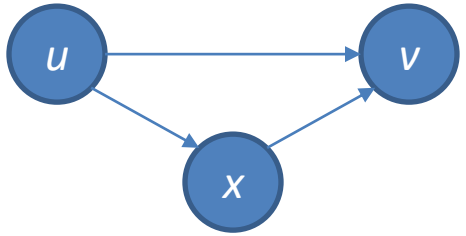
Generalna struktura algoritma najkraćeg puta

- Algoritam (ako nema negativnih zatvorenih putanja):

```
1  Inicijalijacija: for each v∈G.V, v.d=∞, v.pred=NIL
2                   s.d = 0
3  Ponavljaj: izaberi granu (u,v) // nekako?!
4     Relaksiraj granu (u,v): if v.d > u.d + w(u,v)
5                                v.d = u.d + w(u,v)
6                                v.pred = u
   dok ne bude za sve grane v.d ≤ u.d + w(u,v)
```



- Problem: „gruba sila“ - loš izbor grana može dovesti do eksponencijalnog trajanja algoritma
- Rešenje – primena dve osobine:
  - Delovi najkraćeg puta od u do t (npr. $p_{u,v}$) su takođe najkraći putevi
    $u \xrightarrow{p_{u,v}} v \xrightarrow{p_{v,w}} w \xrightarrow{p_{w,t}} t$
  - Nejednakost trougla: $\delta(u,v) \leq \delta(u,x) + \delta(x,v)$



Najkraći put u DAG

- DAG je usmeren acikličan graf (eng. *Directed Acyclic Graph*)

- Algoritam najkraćeg puta dozvoljava negativne težine grana.
- Kružne putanje (ciklusi) ne postoje u DAG (po definiciji) tako da ne mogu uticati na algoritam.

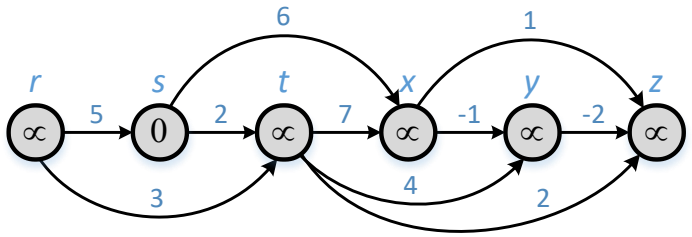
Najkraći put u DAG - algoritam

```
DAG-NAJKRAĆA-PUTANJA(G,w,s)
1  Topološki sortirati čvorove G u listu L
2  INICIJALIZACIJA-IZVORA(G,s)
3  for each čvor u, iz Liste L
4     for each v ∈ G.Susedi(u)
5        RELAKSACIJA(u,v,w)
```

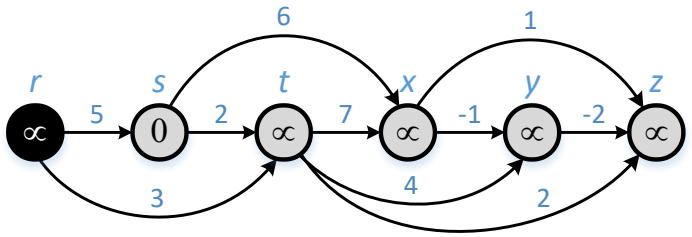
```
INICIJALIZACIJA-IZVORA(G,s)
1  for each v ∈ G.V
2     v.d = ∞
3     v.pred = NIL
4  s.d = 0
```

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

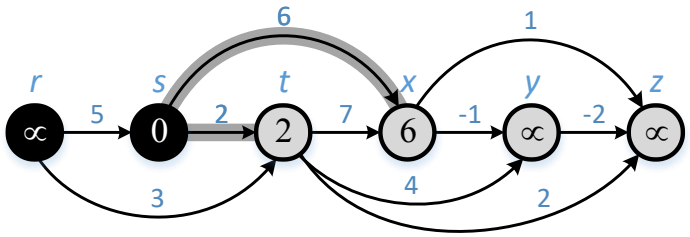
Primer najkraćeg puta u DAG



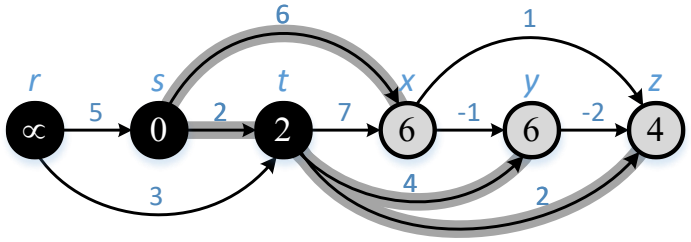
a)



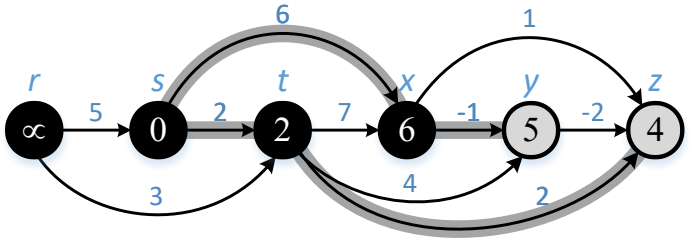
b)



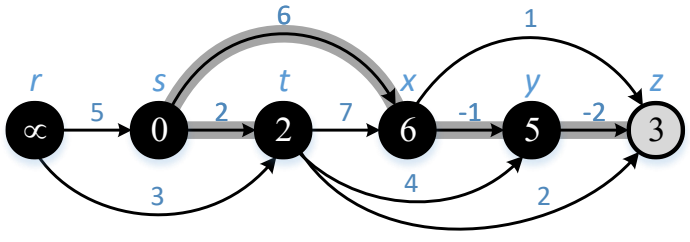
c)



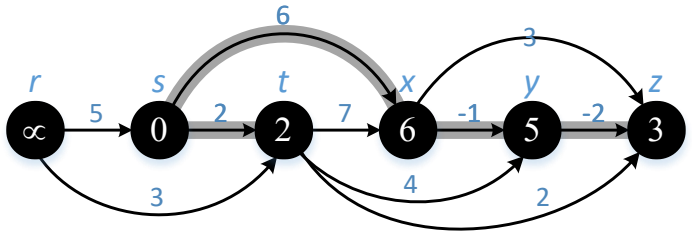
d)



e)



e)



f)

Principi algoritama najkraćeg puta

1. Nejednakost trougla: za svaku granu $(u, v) \in E$ važi $\delta(s, v) \leq \delta(s, u) + w(u, v)$
2. Gornja granica: $v.d \geq \delta(s, v)$ važi za sve čvorove $v \in V$, i jednom kada se postavi na $\delta(s, v)$ više se ne smanjuje
3. Nepostojeća putanja: ako čvor v nije doseziv iz s tada je $v.d = \delta(s, v) = \infty$
4. Konvergencija: ako je $s \to \cdots \to u \to v$ najkraća putanja za čvorove $u, v \in V$ i $u.d = \delta(s, u)$ je postavljeno pre relaksacije grane (u, v) tada je $v.d = \delta(s, v)$ posle relaksacije.
5. Relaksacija putanje: ako je $p = \langle v_0, v_1, \ldots, v_k \rangle$ najkraća putanje od $s = v_0$ do v_k i relaksiraju se grane p u redosledu $(v_0, v_1), (v_1, v_2), \ldots, (v_{k-1}, v_k)$ tada je $v_k.d = \delta(s, v_k)$. (Naravno, između se događaju relaksacije drugih grana, što ne smeta.)
6. Stablo prethodnih čvorova: Kada se postavi $v.d = \delta(s, v)$ za sve čvorove $v \in V$, onda ju najkraći putevi određeni stablom prethodnih čvorova sa korenom u s.

Ideja Belman-Ford algoritma

- Zasniva se na principu Relaksacije putanje
  - Najduža direktna putanja (u najgorem slučaju je to najkraća putanja) u grafu sa n čvorova čine svi ti čvorovi grafa i tada ona ima $n-1$ granu.
  - Ako se u $n-1$ prolaza relaksiraju sve grane grafa, onda će se posle prvog prolaza sigurno dobiti najkraće rastojanje $s \to v_1$ (relaksira se (v_0, v_1)), posle drugog prolaza $s \to v_2$ (relaksira se (v_1, v_2)), i tako redom, do poslednjeg prolaza $s \to v_k$ (relaksira se (v_{k-1}, v_k)).
    Time se dobilo $v_k.d = \delta(s, v_k)$, što je najkraće rastojanje.

- Algoritam:

```
BELMAN-FORD(G,w,s)
1  INICIJALIZACIJA-IZVORA(G,s)
2  for i = 1 to |V|-1
3     for each (u,v) ∈ E
4        RELAKSACIJA(u,v,w)
5  for each (u,v) ∈ E                  // provera
6     if v.d > u.d + w(u,v)
7        return „postoji negativna kružna putanja“
```

Osobine Belman-Ford (*Bellman-Ford*) algoritma

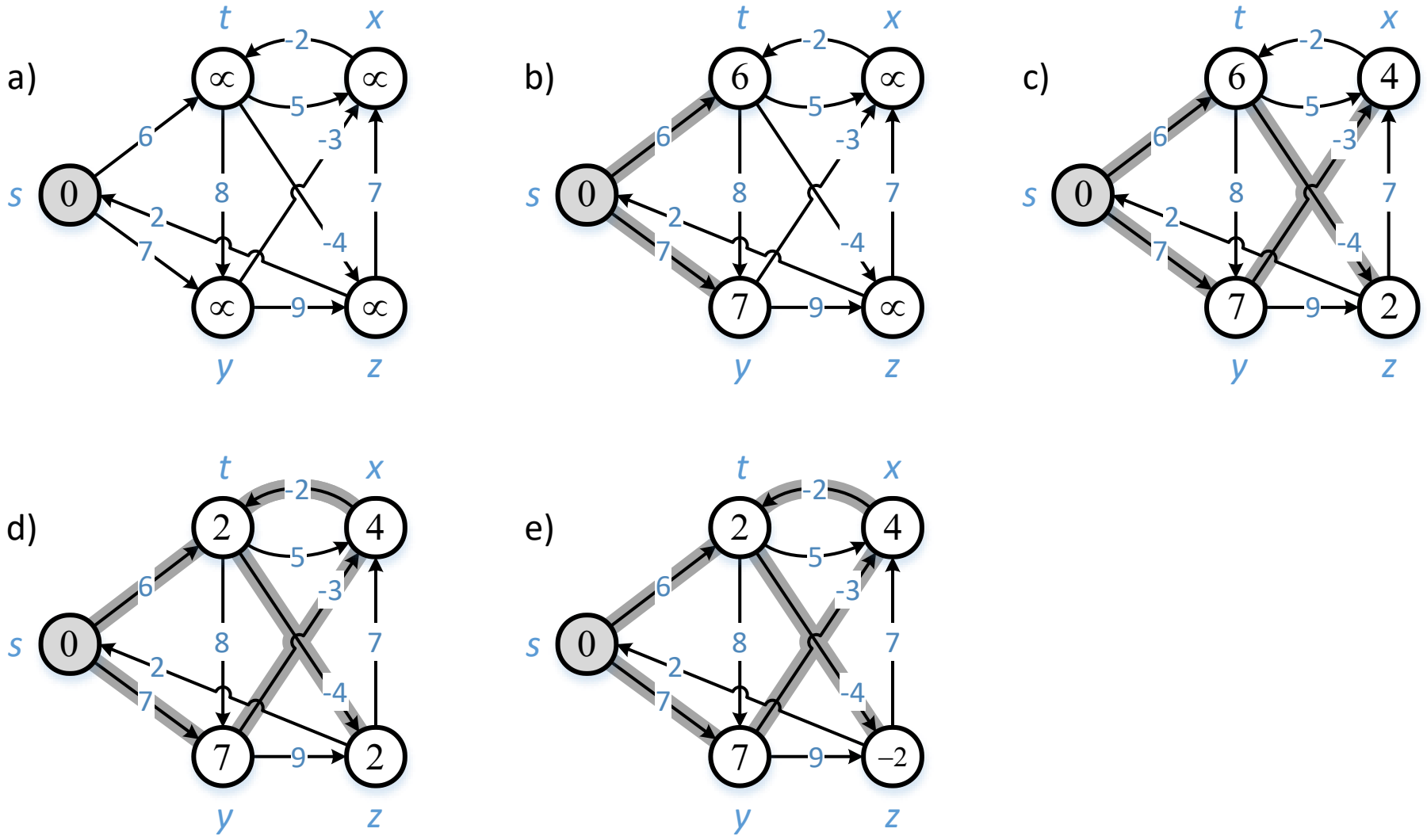
Osobine:

- Reševa problem traženja najkraćih puteva i kada postoje negativne težine grana
- Otkriva postojanje kružnih putanja negativnog pojačanja
- Složenost je $O(nm)$, $n = |V|$, $m = |E|$
  - Ukupan broj „prolaza“ (pokušaja smanjenja rastojanja, redovi 2,3,4) je $(n-1)m$
  - Jedan dodatan prolaz (n-ti, redovi 5,6,7) sa m operacija otkriva postojanje kružnih putanja negativnog pojačanja jer se samo u tom slučaju mogu dobiti kraća rastojanja.

Analiza:

```
BELMAN-FORD(G,w,s)
1  INICIJALIZACIJA-IZVORA(G,s)
2  for i = 1 to |V|-1                  // == (n-1) prolaza
3     for each (u,v) ∈ E               // == m prolaza
4        RELAKSACIJA(u,v,w)
5  for each (u,v) ∈ E                  // ≤ m prolaza
6     if v.d > u.d + w(u,v)
7        return „postoji negativna kružna putanja“
```

Belman-Ford – primer



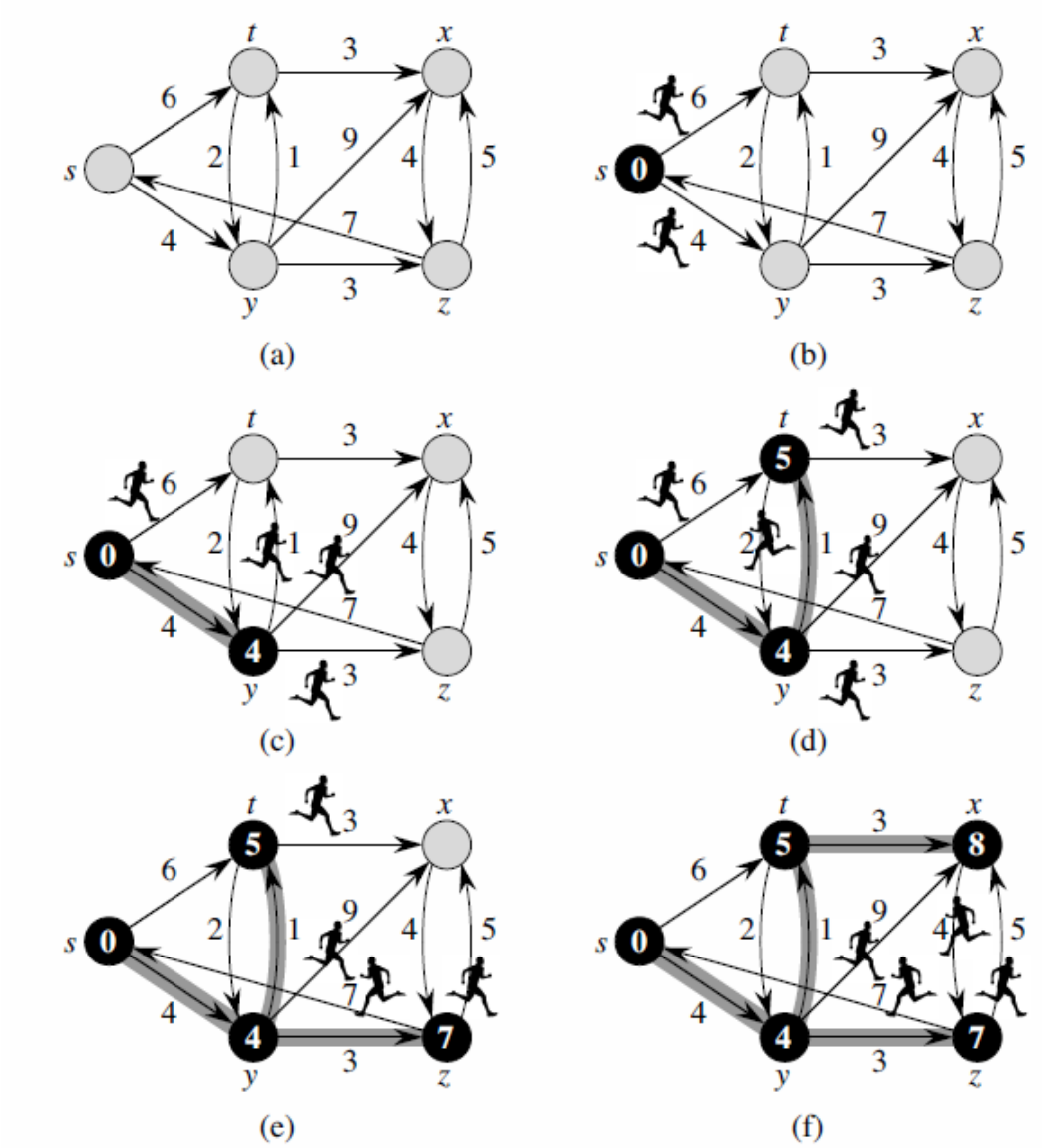
Dajkstra (*Dijkstra*) algoritam - osobine

- Dajkstrin algoritam (1956.) – autor: Edsger W. Dijkstra
- Nalazi **najkraći put od zadatog čvora** do svih ostalih čvorova u težinskom usmerenom grafu gde **sve težine grana nisu negativne**.
- Vreme izvršavanja je kraće od Belman-Ford algoritma.

Princip rada Dajkstra algoritma

1. Pretpostavimo da su težine grana dužine deonica
2. Iz polaznog čvora krenu trkači jednakim brzinama (u svim dopustivim granama)
3. Kada prvi trkač stigne u neposećen čvor
   a) zabeleži se vreme (to odgovara njegovoj najkraćoj razdaljini od polaznog čvora)
   b) preda se „štafeta“ drugim trkačima (kojih može biti više i trče svim dopustivim granama)

Primetiti da trkači dužim stazama trče duže od trkača koji je prvi stigao u čvor, što ne smeta. Kada stignu u čvor oni ne predaju štafetu jer je to već urađeno.



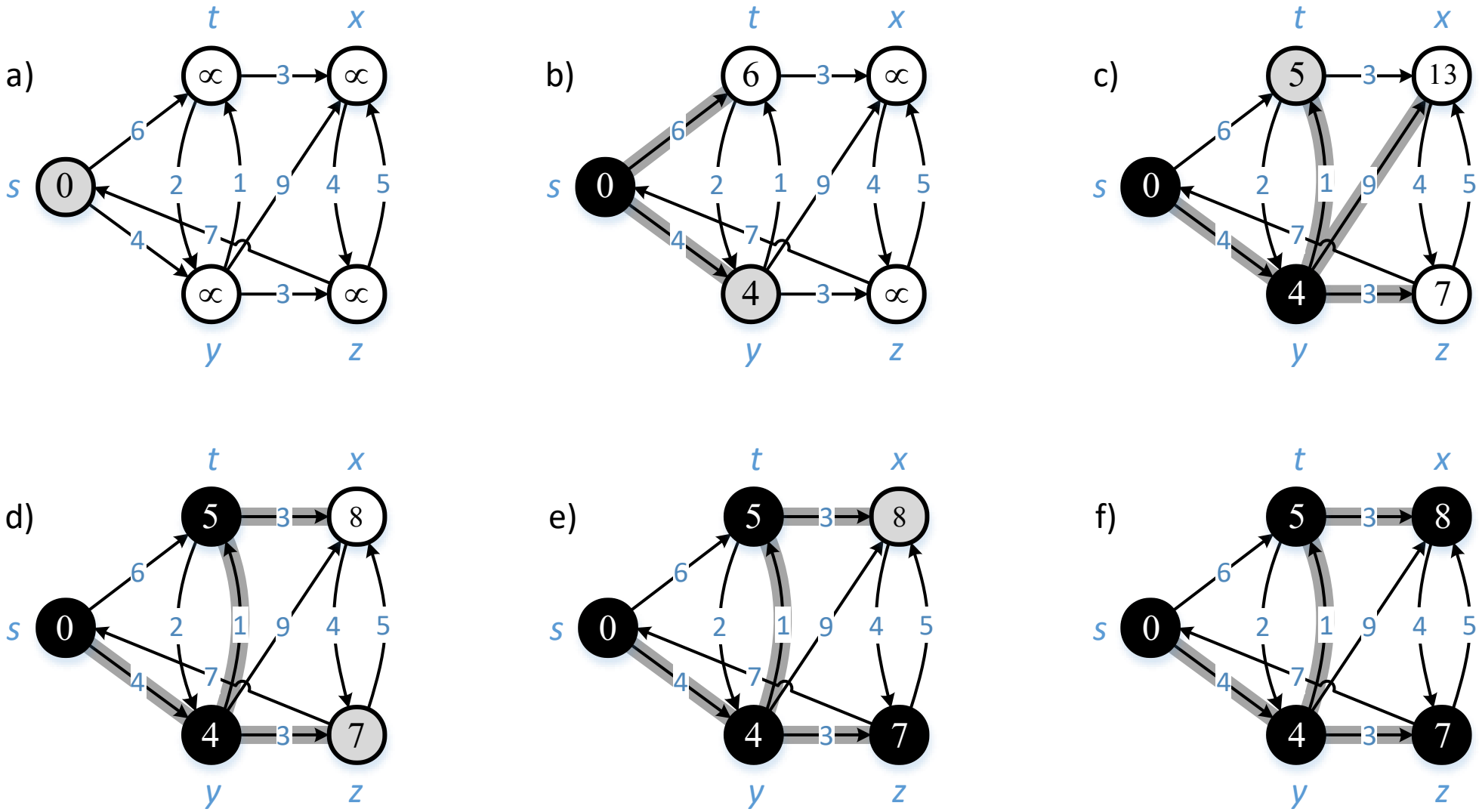
Dajkstra algoritam

- Algoritam održava skup čvorova S gde su rastojanja do polaznog čvora određena (neće se menjati). Zatim bira čvor $u \in V \backslash S$ sa najmanjim rastojanjem $u.d$, dodaje ga u S i koriguje (relaksira) rastojanja do svih čvorova preko grana koje izlaze iz u.

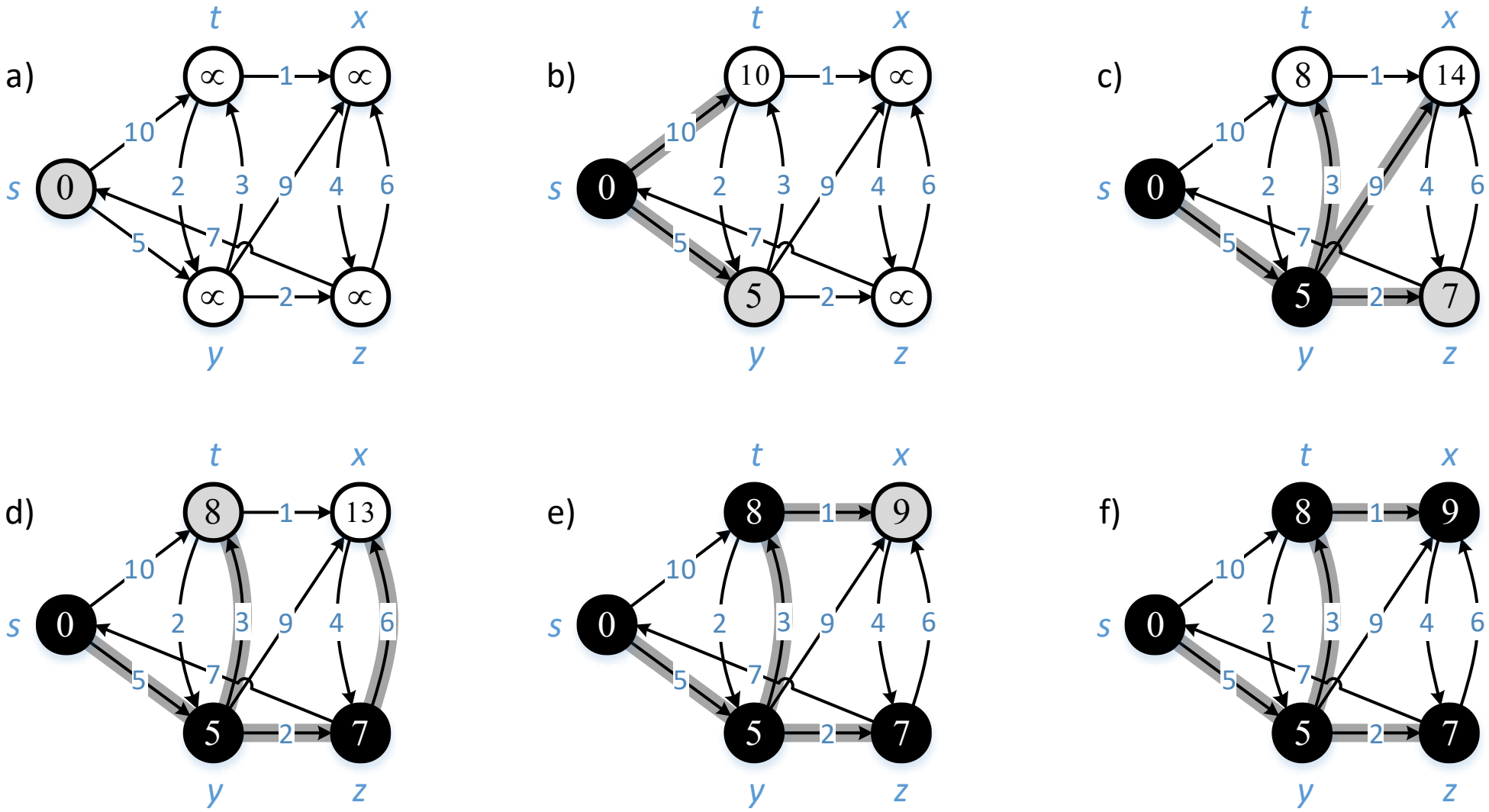
```
DAJKSTRA(G,w,s)
1  INICIJALIZACIJA-IZVORA(G,s)      // s ima minimalno rastojanje ==0
2  S = ∅                            // obrađeni čvorovi
3  Q = G.V                          // neorađeni čvorovi – treba uraditi za sve
4  while Q ≠ ∅                      // dok ima neobrađenih čvorova
5     u = IZDVOJ-MINIMUM(Q)         // trenutno najbliži s se sledeći obrađuje
6     S = S ∪ {u}                   // dodaj u obrađene
7     for each v ∈ G.Susedi(u)      // skrati rastojanja neobrađenih
8        RELAKSACIJA(u,v,w)
```

Uzima se trenutno najbliži jer ostali čvorovi mogu biti samo dalji pošto su sve težine grana pozitivne.
Skup neobrađenih se može realizovati kao *Min Heap*

Dajkstra – primer A



Dajkstra – primer B



Vreme izvršavanja Dajkstra algoritma

- Algoritam održava red sa minimalnim prioritetima Q (*min-priority queue*) $n = |V|, m = |E|$
  - poziva Insert operaciju (red #3) broj poziva n
  - poziva Izdvoj-Minimum (red #5) broj poziva n
  - poziva Relaksaciju broj poziva m
- Trajanje algoritma zavisi od načina implementacije Q

```
DAJKSTRA(G,w,s)
1  INICIJALIZACIJA-IZVORA(G,s)
2  S = ∅
3  Q = G.V
4  while Q ≠ ∅
5     u = IZDVOJ-MINIMUM(Q)
6     S = S ∪ {u}
7     for each v ∈ G.Susedi(u)
8        RELAKSACIJA(u,v,w)
```

Vreme izvršavanja Dajkstra algoritma (2)

- Implementacija Q nizom indeksiranim rednim brojem čvora $(1..n)$
  - Trajanja operacija:
    - Dodaj = $\Theta(1)$
    - Izdvoj-Minimum = $O(n)$ jer se pretražuje ceo niz
    - Relaksacija = $\Theta(1)$
  - Vreme izvršavanja je $n\Theta(1) + nO(n) + m\Theta(1) = O(n^2 + m) = O(n^2)$
    - Jer je broj grana u grafu je $m \leq n^2$

- implementacija Q upotrebom binarnog *min Heap*-a
  - Trajanja operacija:
    - Dodaj = $O(\log_2 n)$
    - Izdvoj-Minimum = $O(\log_2 n)$
    - Relaksacija = $O(\log_2 n)$ – zbog pronalaženja u *Heap*-u
  - Vreme izvršavanja $nO(\log_2 n) + nO(\log_2 n) + mO(\log_2 n) = O((m + n)\log_2 n)$
    - Svodi se na $O(m \log_2 n)$ kada su svi čvorovi dosezivi iz izvora

- Ako je graf redak (ima mali broj grana) onda je efikasniji način 2.

Vreme izvršavanja Dajkstra algoritma (3)

- implementacija Q upotrebom Fibonačijevog hipa (*Fibonacci heap*)
  - Trajanja operacija:
    - Izdvoj-Minimum = $O(\log_2 n)$
    - Relaksacija = $O(1)$ – amortizovano vreme pronalaženja
  - Vreme izvršavanja $O(m + n \log_2 n)$

- Ovde je upotrebljana **amortizovana složenost** $O(1)$: za n poziva je složenost $O(n)$ tako da se može smatrati da je prosečno trajanje poziva $O(1)$.
  - U opštem slučaju kod amortizivane složenosti, što ovde nije slučaj, pojedini pozivi mogu duže trajati, i za prosečno trajaje se ukupno vreme trajanja deli sa brojem poziva.

Najkraći put između svih čvorova u grafu

- Posmatra se usmereni graf $G = (V, E)$, gde su poznate težine grana w: $E \rightarrow \mathbb{R}$
- Najkraći put između svih čvorova u grafu se može odrediti ponavljanjem traženja najkraćeg puta od zadatog izvornog čvora do svih ostalih čvorova u grafu (osnovni problem) i tako za svaki čvor
  - Znači, prethodno posmatrani algoritmi treba da se ponove $|V|$ puta ($n = |V|$, $m = |E|$).
    - Belman-Ford: $O(n^2 m)$. Ako je graf "gust", tj. $m \approx n^2$ imamo $O(n^4)$
    - Dajkstra sa implementiranim Q kao
      - niz indeksiran rednim brojem čvora: $O(n^3)$
      - binarni min-hip: $O(mn \log_2 n)$
      - Fibonačijev hip: $O(mn + n^2 \log_2 n)$

- Ovde će se pomatrati efikasnije rešenje
- Za opis grafa će se upotrebiti matrica susedstva $W = [w_{ij}], i, j = 1,2, \ldots n$
  - (do sada je bila u upotrebi lista susedstva)

Rekurzivno računanje najkraćih rastojanja

- Rekurzivno rešenje određuje najkraće putanje između čvorova tako što produžava putanje od 1 do maksimalno $n-1$ grana
  - Definiše se matrica rastojanja između čvorova $L = [l_{ij}], i, j = 1,2, \ldots n$ koja se rekurzivno koriguje Posmatra je $L^{(m)}, m = 0,1, \ldots n-1$
  - Početna vrednost: $l_{ij}^{(0)} = \begin{cases} 0, i = j \\ \infty, i \neq j \end{cases}$
  - Za $m \geq 1$, $l_{ij}^{(m)}$ se računa na osnovu prethodne $l_{ij}^{(m-1)}$, tj. putanja sa m grana se određuje na osnovu putanje sa $m-1$ grana
- Traženje najkraćih rastojanja je algortiam dinamičkog programiranja

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

jer je $w_{jj} = 0$ za svako j.

- Ako graf nema negativne cikluse onda je $\delta(i,j) = l_{ik}^{(n-1)} = l_{ik}^{(n)} = l_{ik}^{(n+1)} = \cdots$

Rekurzivno računanje najkraćih rastojanja (2)

- Računaju se $L^{(1)} = W, L^{(2)}, L^{(3)}, \ldots L^{(n-1)}$
  - $L^{(m)}$ se računa na osnovu prethodnog $L^{(m-1)}$
- Složenost računanja je $\Theta(n^3)$

```
RAČUNANJEL(L,W)                          // sva rastojanja su iste dužine
1  n = L.kolona
2  L' = new [l'ij]                       // matrica n x n
3  for i = 1 to n                        // po vrstama i
4     for j = 1 to n                     // po kolonama => sva rastojanja
5        l'ij = ∝
6        for k = 1 to n                  // po svim potencijalnim vezama
7           l'ij = min(l'ij, lik+wkj)    // skrati rastojanje?
8  return L'
```

Algoritam računanja najkraćih rastojanja

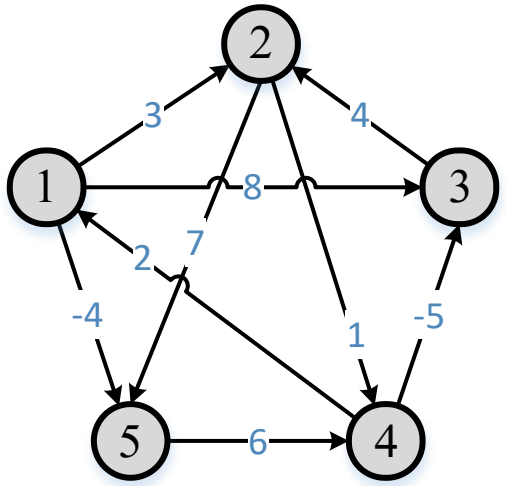
- Naivno (sporo) računanje ima složenost $\Theta(n^4)$

```
SPORORAČUNANJENAJKRAĆIHRASTOJANJA(W)
1  n = W.kolona
2  L(1) = W
3  for m = 2 to n-1          // za sve dužine putanja
4     L(m) = new [lij]
5     L(m) = RAČUNANJEL(L(m-1),W)
6  return L(m)
```

```
L(1) = RAČUNANJEL(L(0),W)
L(2) = RAČUNANJEL(L(1),W)
...
L(n-1) = RAČUNANJEL(L(n-2),W)
```

- Može brže ...

Primer



L(1)=

```
[0.0   3.0   8.0  Inf  -4.0
 Inf   0.0   Inf  1.0   7.0
 Inf   4.0   0.0  Inf   Inf
 2.0   Inf  -5.0  0.0   Inf
 Inf   Inf   Inf  6.0   0.0]
```

L(2)=

```
[0.0   3.0   8.0  2.0  -4.0
 3.0   0.0  -4.0  1.0   7.0
 Inf   4.0   0.0  5.0  11.0
 2.0  -1.0  -5.0  0.0  -2.0
 8.0   Inf   1.0  6.0   0.0]
```

L(3)=

```
[0.0   3.0  -3.0  2.0  -4.0
 3.0   0.0  -4.0  1.0  -1.0
 7.0   4.0   0.0  5.0  11.0
 2.0  -1.0  -5.0  0.0  -2.0
 8.0   5.0   1.0  6.0   0.0]
```

L(4)=

```
[0.0   1.0  -3.0  2.0  -4.0
 3.0   0.0  -4.0  1.0  -1.0
 7.0   4.0   0.0  5.0   3.0
 2.0  -1.0  -5.0  0.0  -2.0
 8.0   5.0   1.0  6.0   0.0]
```

Kraj.

Ako se nastavi dobija se L(5)==L(4)

L(5)=

```
[0.0   1.0  -3.0  2.0  -4.0
 3.0   0.0  -4.0  1.0  -1.0
 7.0   4.0   0.0  5.0   3.0
 2.0  -1.0  -5.0  0.0  -2.0
 8.0   5.0   1.0  6.0   0.0]
```

Algoritam ubrzanog računanja najkraćih rastojanja

- uvedemo operator ♣, koji je asocijativan
- $L^{(n-1)}$ se može izračunati u $k = \lceil \log_2(n-1) \rceil$ koraka
  - Računaju se $L^{(1)}, L^{(2)}, L^{(4)}, \ldots L^{(2^k)}$
  - Podsećamo da je $L^{(n-1)} = L^{(n)} = L^{(n+1)} = \cdots$ jer nema negativnih ciklusa
- Složenost $\Theta(n^3 \log_2 n)$

```
BRŽERAČUNANJENAJKRAĆIHRASTOJANJA(W)
1  n = W.kolona, m = 1
2  L(1) = W
3  while m < n-1
4      L(2m) = new [Lij]
5      L(2m) = RAČUNANJEL(L(m), L(m))
6      m = 2m
7  return L(m)
```

$L^{(1)}$ = RAČUNANJEL($L^{(0)}$,W)
$L^{(2)}$ = RAČUNANJEL($L^{(1)}$,W)
...
$L^{(n-1)}$ = RAČUNANJEL($L^{(n-2)}$,W)

$L^{(1)} = L^{(0)} \clubsuit W$
$L^{(2)} = L^{(1)} \clubsuit W$
...
$L^{(n-1)} = L^{(n-2)} \clubsuit W$

$L^{(1)} = W$
$L^{(2)} = L^{(1)} \clubsuit W = W \clubsuit W$
$L^{(3)} = L^{(2)} \clubsuit W = W \clubsuit W \clubsuit W$
$L^{(4)} = L^{(3)} \clubsuit W = (W \clubsuit W) \clubsuit (W \clubsuit W) = L^{(2)} \clubsuit L^{(2)}$
...
$L^{(8)} = L^{(4)} \clubsuit L^{(4)}$
...

Flojd-Varšal algoritam

- Flojd-Varšal (*Floyd-Warshall*) algoritam pronalazi najkraći put između svih čvorova u grafu
- Vreme izvršavanje je $\Theta(n^3)$
- Koristi dinamičko progamiranje i iterativno popravlja rastojanje između svaka dva čvora sve dok procena nije optimalna.
- Ideja: Posmatra se međučvor (unutrašnji čvor) u putanji $p = \langle v_0, v_1, \ldots, v_l \rangle$, tj. bilo koji čvor od v_1 do v_{l-1} i putanja se deli na dva dela: od polaznog čvora do međučvora, i od međučvora do krajnjeg čvora.
  - Drugim rečima, putanju čine dve kraće putanje
  - Primetiti da se u prethodnom algoritmu putanja produžavala dodavanjem jedne grane

Algoritam - ideja

- Posmatra se podskup čvorova $\{1,2, \dots k\}$ iz grafa $G = (V, E), |V| = n$
- za neke čvorove $i, j \in V$ posmatraju se putanje čiji su međučvorovi iz podskupa $\{1,2, \dots k\}$. Konkretnije, poznati su najkraći putevi sa međučvorovima $\{1,2, \dots k-1\}$ i posmatra se dodavanje novog čvora u putanju:
  - Ako je novi čvor iz skupa $\{1,2, \dots k\}$ (jedini preostali čvor k) onda je putanja sa njim najkraća (jer je to osobina putanja iz skupa)
  - Ako je novi čvor k van podskupa onda se putanja p deli na dva dela p_1 i p_2, tj. $i \overset{p_1}{\to} k \overset{p_2}{\to} j$ gde su i p_1 i p_2 najkraće putanje od i do k, odnosno od k do j.
- Na osnovu ovoga se formira rekurzivno računanje rastojanja sa međučvorovima $\{1,2, \dots k\}$

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

- Konačno, najduža putanja ima n čvorova, pa je $d_{ij}^{(n)} = \delta(i, j)$ za sve $i, j \in V$.

  Odnosno, formira se matrica rastojanja: $D_{ij}^{(n)} = [d_{ij}^{(n)}]$

Flojd-Varšal pseudokod

```
FLOJD-VARŠAL(W)
1  n = W.kolona
2  D(0) = W
3  for k = 1 to n              // određuje sve putanje od k čvorova
4     D(k) = new [dij(k)]      // matrica n x n
5     for i = 1 to n           // dve petlje uzimaju sva rastojanja
6        for j = 1 to n
7           dij(k) = min(dij(k-1), dik(k-1) + dkj(k-1))
8  return D(n)
```

- Složenost je $\Theta(n^3)$
- Algoritam detektuje negativne cikluse tako što se na dijagonali $D^{(n)}$ nađe negativna vrednost, tj. $d_{ii} < 0$ za neko $i = 1,2, \ldots n$
- Dodatno, mogu se rekonstruisati najkraće putanje na osnovu matrice prethodnika Π

Matrica prethodnika

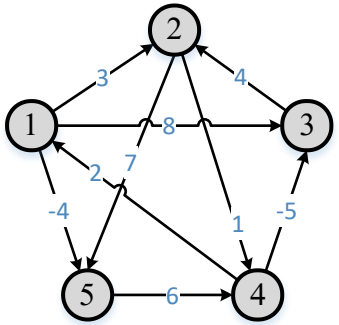
Ažuriranje $D^{(k)}$ prati ažuriranje $\Pi^{(k)}$

$$\Pi^{(0)}: \quad \pi_{ij}^{(0)} = \begin{cases} Nil, & i = j \mid w_{ij} = \infty \\ i, & i \neq j \mid w_{ij} < \infty \end{cases}$$

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

gde je sa π_{ij} označen indeks čvora koji prethodni j-tom čvoru kada se posmatra najkaća putanja koja polazi od čvora i

Primer 1



$D^{(m)}$ $\Pi^{(m)}$

(0)

0	3	8	Inf	-4	-	1	1	-	1
Inf	0	Inf	1	7	-	-	-	2	2
Inf	4	0	Inf	Inf	-	3	-	-	-
2	Inf	-5	0	Inf	4	-	4	-	-
Inf	Inf	Inf	6	0	-	-	-	5	–

(1)

0	3	8	Inf	-4	-	1	1	-	1
Inf	0	Inf	1	7	-	-	-	2	2
Inf	4	0	Inf	Inf	-	3	-	-	-
2	5	-5	0	-2	4	1	4	-	1
Inf	Inf	Inf	6	0	-	-	-	5	-

(2)

0	3	8	4	-4	-	1	1	2	1
Inf	0	Inf	1	7	-	-	-	2	2
Inf	4	0	5	11	-	3	-	2	2
2	5	-5	0	-2	4	1	4	-	1
Inf	Inf	Inf	6	0	-	-	-	5	-

$D^{(m)}$ $\Pi^{(m)}$

(3)

0	3	8	4	-4	-	1	1	2	1
Inf	0	Inf	1	7	-	-	-	2	2
Inf	4	0	5	11	-	3	-	2	2
2	-1	-5	0	-2	4	3	4	-	1
Inf	Inf	Inf	6	0	-	-	-	5	-

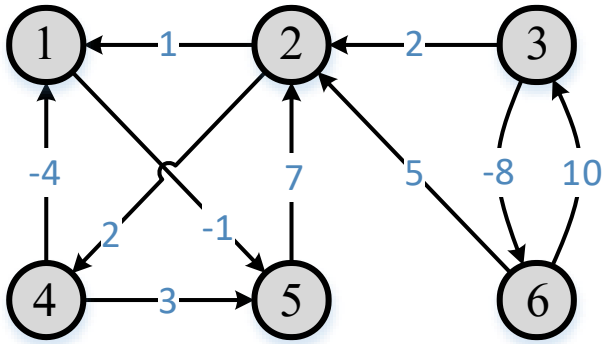
(4)

0	3	-1	4	-4	-	1	4	2	1
3	0	-4	1	-1	4	-	4	2	1
7	4	0	5	3	4	3	-	2	1
2	-1	-5	0	-2	4	3	4	-	1
8	5	1	6	0	4	3	4	5	-

(5)

0	1	-3	2	-4	-	3	4	5	1
3	0	-4	1	-1	4	-	4	2	1
7	4	0	5	3	4	3	-	2	1
2	-1	-5	0	-2	4	3	4	-	1
8	5	1	6	0	4	3	4	5	-

Primer 2



(0)

```
   0  Inf  Inf  Inf   -1  Inf     - - - - 1 -
   1    0  Inf    2  Inf  Inf     2 - - 2 - -
 Inf    2    0  Inf  Inf   -8     - 3 - - - 3
  -4  Inf  Inf    0    3  Inf     4 - - - 4 -
 Inf    7  Inf  Inf    0  Inf     - 5 - - - -
 Inf    5   10  Inf  Inf    0     - 6 6 - - -
```

(1)

```
   0  Inf  Inf  Inf   -1  Inf     - - - - 1 -
   1    0  Inf    2    0  Inf     2 - - 2 1 -
 Inf    2    0  Inf  Inf   -8     - 3 - - - 3
  -4  Inf  Inf    0   -5  Inf     4 - - - 1 -
 Inf    7  Inf  Inf    0  Inf     - 5 - - - -
 Inf    5   10  Inf  Inf    0     - 6 6 - - -
```

(2)

```
   0  Inf  Inf  Inf   -1  Inf     - - - - 1 -
   1    0  Inf    2    0  Inf     2 - - 2 1 -
   3    2    0    4    2   -8     2 3 - 2 1 3
  -4  Inf  Inf    0   -5  Inf     4 - - - 1 -
   8    7  Inf    9    0  Inf     2 5 - 2 - -
   6    5   10    7    5    0     2 6 6 2 1 -
```

(3)

```
   0  Inf  Inf  Inf   -1  Inf     - - - - 1 -
   1    0  Inf    2    0  Inf     2 - - 2 1 -
   3    2    0    4    2   -8     2 3 - 2 1 3
  -4  Inf  Inf    0   -5  Inf     4 - - - 1 -
   8    7  Inf    9    0  Inf     2 5 - 2 - -
   6    5   10    7    5    0     2 6 6 2 1 -
```

(4)

```
   0  Inf  Inf  Inf   -1  Inf     - - - - 1 -
  -2    0  Inf    2   -3  Inf     4 - - 2 1 -
   0    2    0    4   -1   -8     4 3 - 2 1 3
  -4  Inf  Inf    0   -5  Inf     4 - - - 1 -
   5    7  Inf    9    0  Inf     4 5 - 2 - -
   3    5   10    7    2    0     4 6 6 2 1 -
```

(5)

```
   0    6  Inf    8   -1  Inf     - 5 - 2 1 -
  -2    0  Inf    2   -3  Inf     4 - - 2 1 -
   0    2    0    4   -1   -8     4 3 - 2 1 3
  -4    2  Inf    0   -5  Inf     4 5 - - 1 -
   5    7  Inf    9    0  Inf     4 5 - 2 - -
   3    5   10    7    2    0     4 6 6 2 1 -
```

(6)

```
   0    6  Inf    8   -1  Inf     - 5 - 2 1 -
  -2    0  Inf    2   -3  Inf     4 - - 2 1 -
  -5   -3    0   -1   -6   -8     4 6 - 2 1 3
  -4    2  Inf    0   -5  Inf     4 5 - - 1 -
   5    7  Inf    9    0  Inf     4 5 - 2 - -
   3    5   10    7    2    0     4 6 6 2 1 -
```