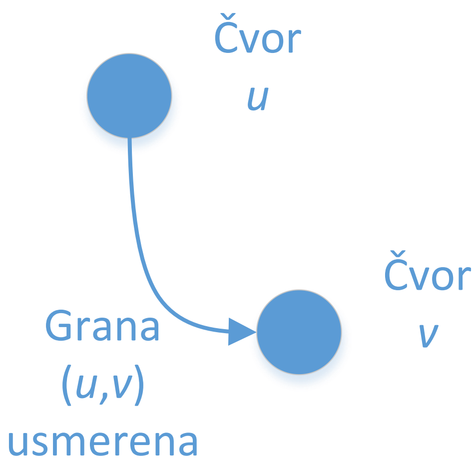
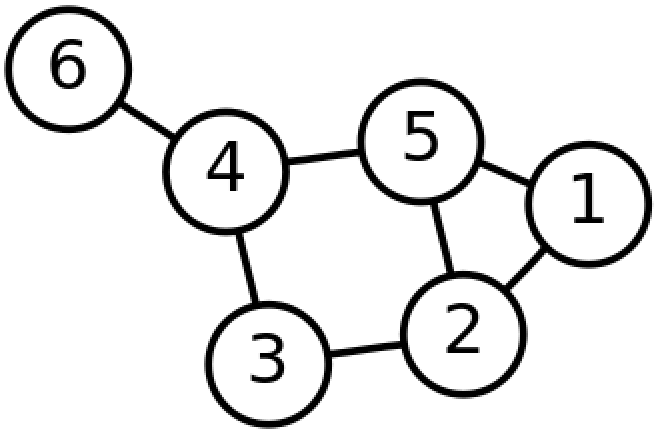
Algoritmi

Grafovi

Uvod

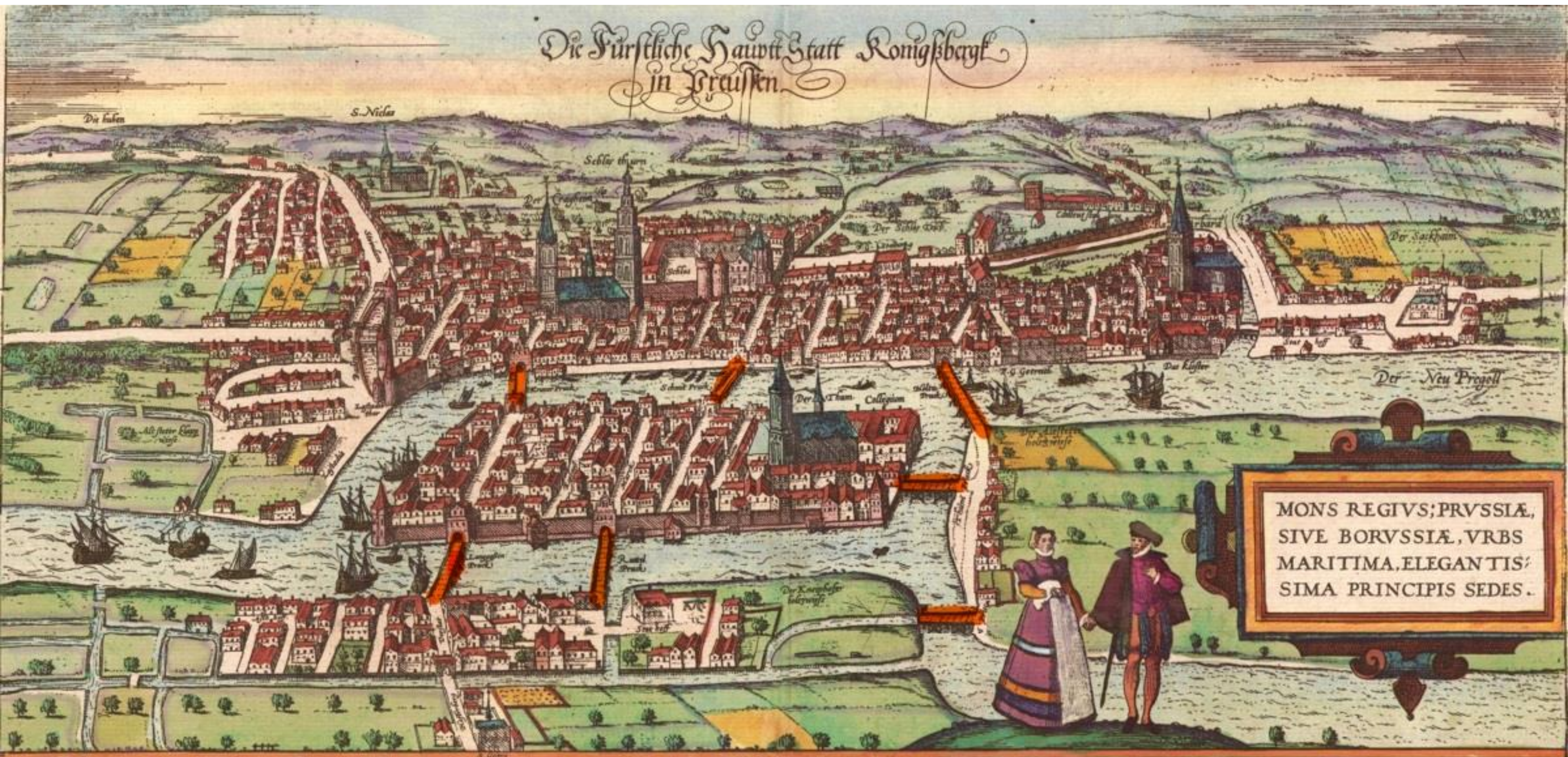
- Grafovi (*graphs*) se izučavaju u teoriji grafova (oblast diskretne matematike i računarske nauke)
- Graf je matematička struktura za modelovanje odnosa između parova objekata.
- **Graf** se sastoji od:
  - **čvorova** (objekata) – *nodes* ili *vertices*, i
  - **grana** (koje povezuju parove objekata)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





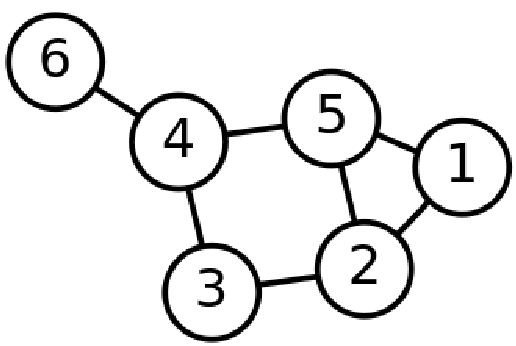
SEDAM MOSTOVA KENIGSBERGA (od 1946. **KALINJINGRADA)**

- Da li namernik u ruskom gradu Kalinjingradu može da obiđe svih sedam mostova (po jednom) i da se vrati na početak puta
- rešio je još 1735. godine švajcarski matematičar Leonard Ojler



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, biološkim, socialnim i informatičkim sistemima.
- U računarstvu grafovi se koriste za predstavljanje računarske mreže, organizacije podataka, tokova podataka i sl.

- Primeri:
  - Struktura website-a: čvorovi su stranice, a usmerene grane su hiper linkovi,
  - Mreža puteva: čvorovi su raskrsnice, a grane su putevi.
  - Elektrodistributivna mreža: čvorovi su transformatorske stanice, a grane su vodovi.

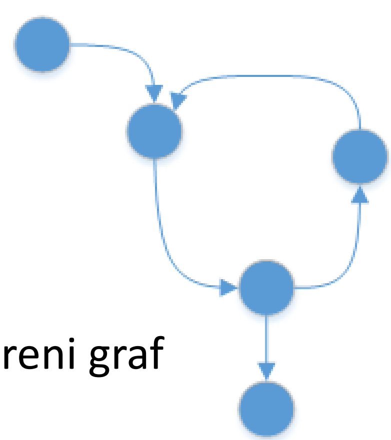
Tipovi grafova

Sa stanovišta usmerenosti grana:

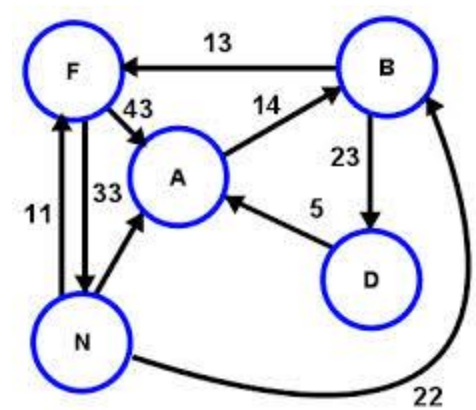
- **Neusmereni** (neorijentisani)
- **Usmereni** (orijentisani)
- Mešoviti
  – samo deo grana je orijentisan
- **Aciklični**
  – nema „petlje“

Sa stanovišta parametara grana:

- **Direktan** graf
- **Težinski** graf



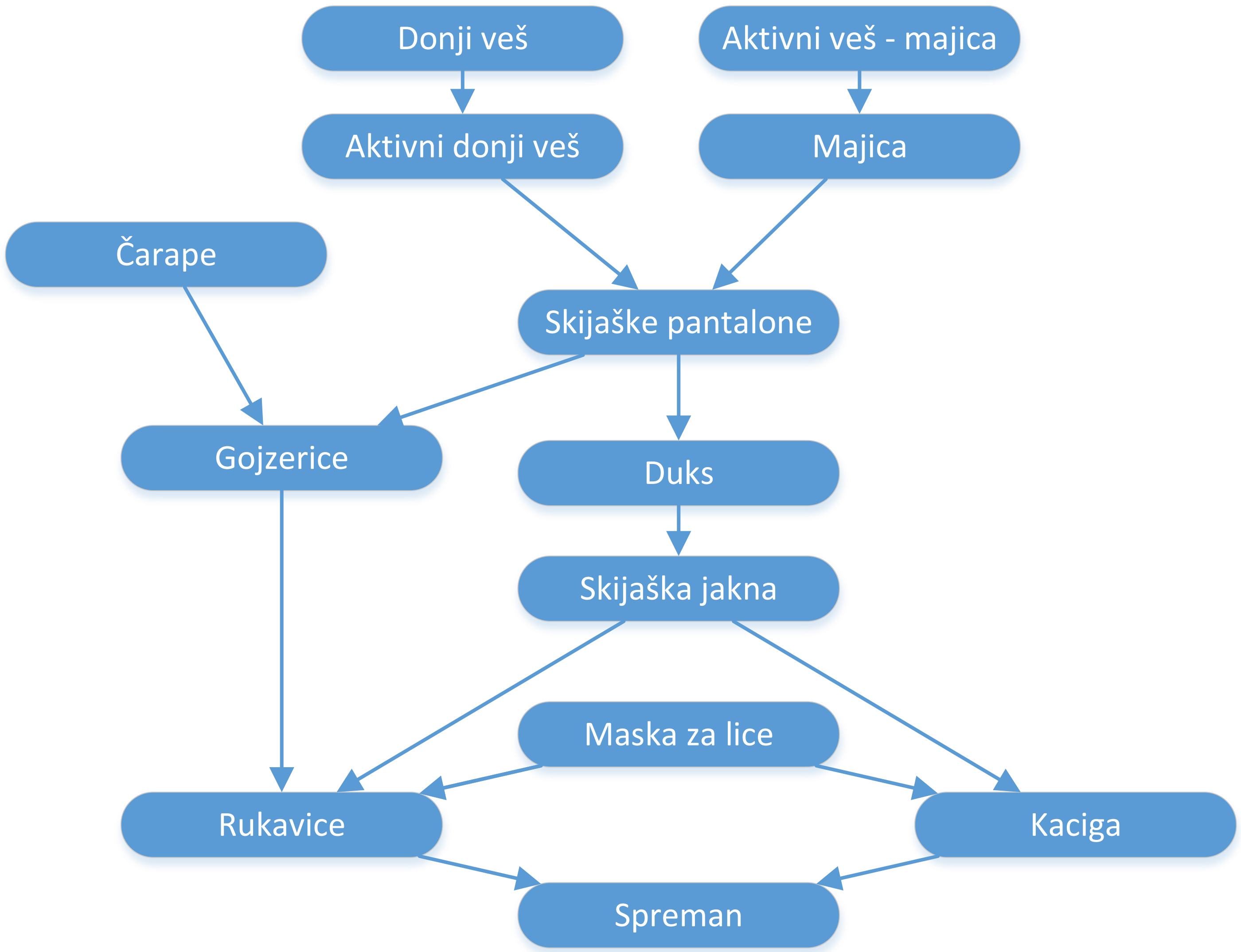
Ciklični usmereni graf



Težinski graf

Primer usmerenog acikličnog grafa

- Oblačenje skijaša ...



Topološko sortiranje

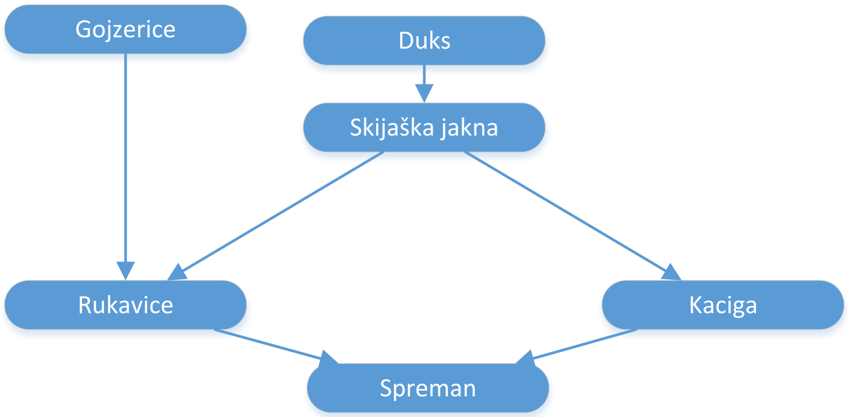
- Topološko sortiranje usmerenog acikličnog grafa proizvodi poredak gde je čvor u ispred čvora v kada ih spaja grana (u, v).
  - Poredak ne mora biti jedinstven

- 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

1. Svakom čvoru pridružujemo:
   - Broj ulaza (*in-degree*) = broj grana koje završavaju u čvoru
2. Biramo čvor bez ulaza (broj ulaza je 0)
   - Postavimo ga na kraj sortiranog reda
   - Uklonimo ga iz grafa
3. Ponavljamo korak 2. (dok ima čvorova u njemu)

Topološko sortiranje - 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

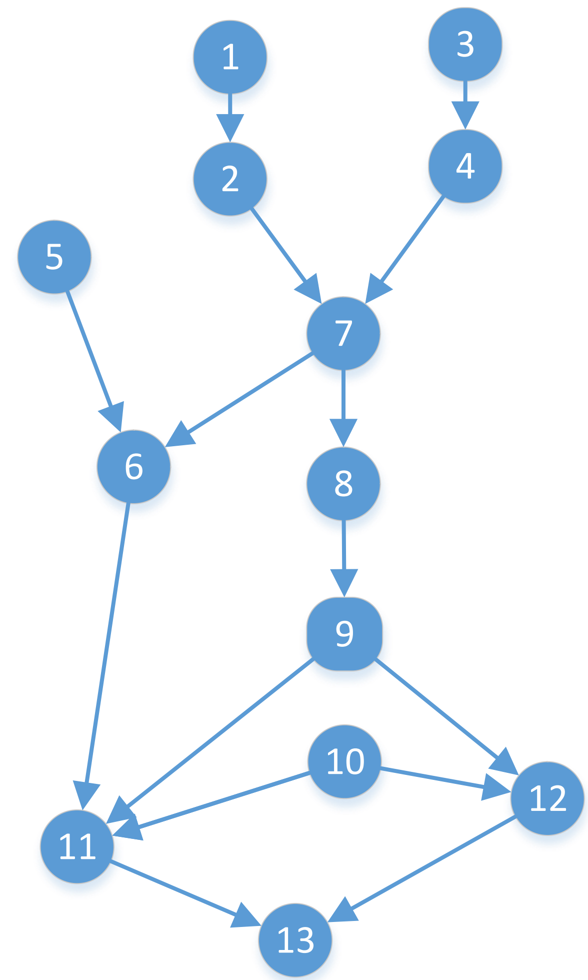
```
TOPOLOGICAL-SORT(G)    // G=G(V,E)
next = ∅               // skup čvorova bez ulaza
rez = []               // rezultujući niz
for each u ∈ G.V               // za svaki čvor prebroj ulaze
   u.in = 0
for each u ∈ G.V
   for each v ∈ G.Adj(u)       // Adj daje susede
      v.in = v.in + 1
for each u ∈ G.V
   if u.in == 0                // čvor bez ulaza ubaci u next
      next = next ∪ u
while next ≠ ∅                 // dok ima čvorova bez ulaza
   u: next = next \ u          // uzmi jedan takav čvor
   rez = [rez u]               // dodaj ga na kraj rezultata
   for each v ∈ G.Adj(u)
      v.in = v.in - 1
      if v.in == 0
         next = next ∪ v
return rez
```

Predstavljanje grafova

- Matricom susedstva
- Listom susedstva

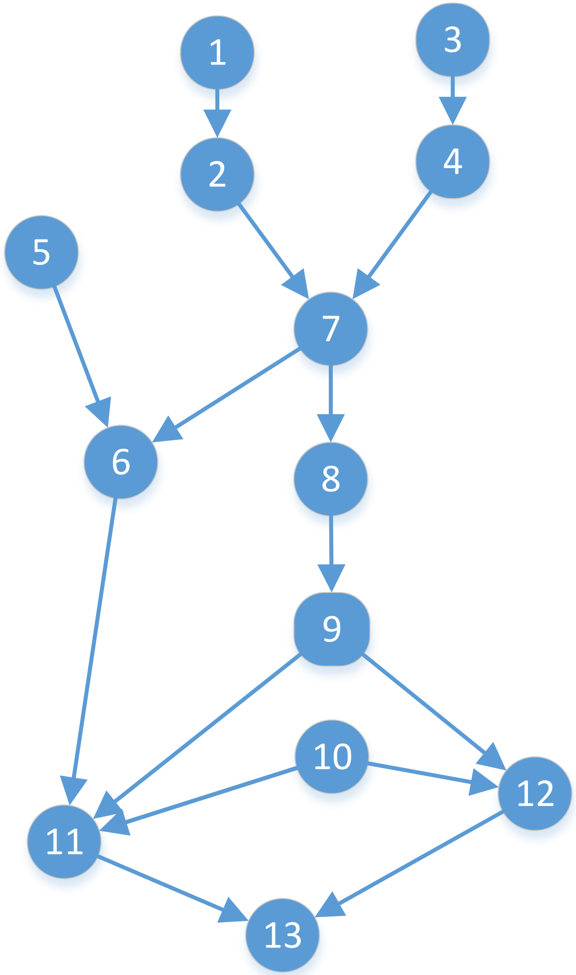
Matrica susedstva

	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

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 $\Theta(n+m), n=|V|, m=|E|$

```
TOPOLOGICAL-SORT(G)            // G=G(V,E)
next = ∅
rez = []
for each u ∈ G.V                         // n prolaza
    u.in = 0
for each u ∈ G.V                         // n prolaza
    for each v ∈ G.Adj(u)                // ukupno m prolaza
        v.in = v.in + 1
for each u ∈ G.V                         // n prolaza
    if u.in == 0
        next = next ∪ u                  // dodaj u listu next, O(1)
while next ≠ ∅
    u: next = next \ u                   // n prolaza, u svakom ukloni iz liste O(1)
    rez = [rez u]                        // O(1)
    for each v ∈ G.Adj(u)                // ukupno m prolaza
        v.in = v.in - 1
        if v.in == 0
            next = next ∪ v              // svaki čvor se jednom mora dodati u next
return rez
```

Obilazak grafa

- Obilazak grafa posećuje sve čvorove i grane grafa
- Algoritmi:
  - Pretraga u širinu (*Breadth-first search* - BFS)
  - Pretaraga u dubinu (*Depth-first search* - DFS)

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

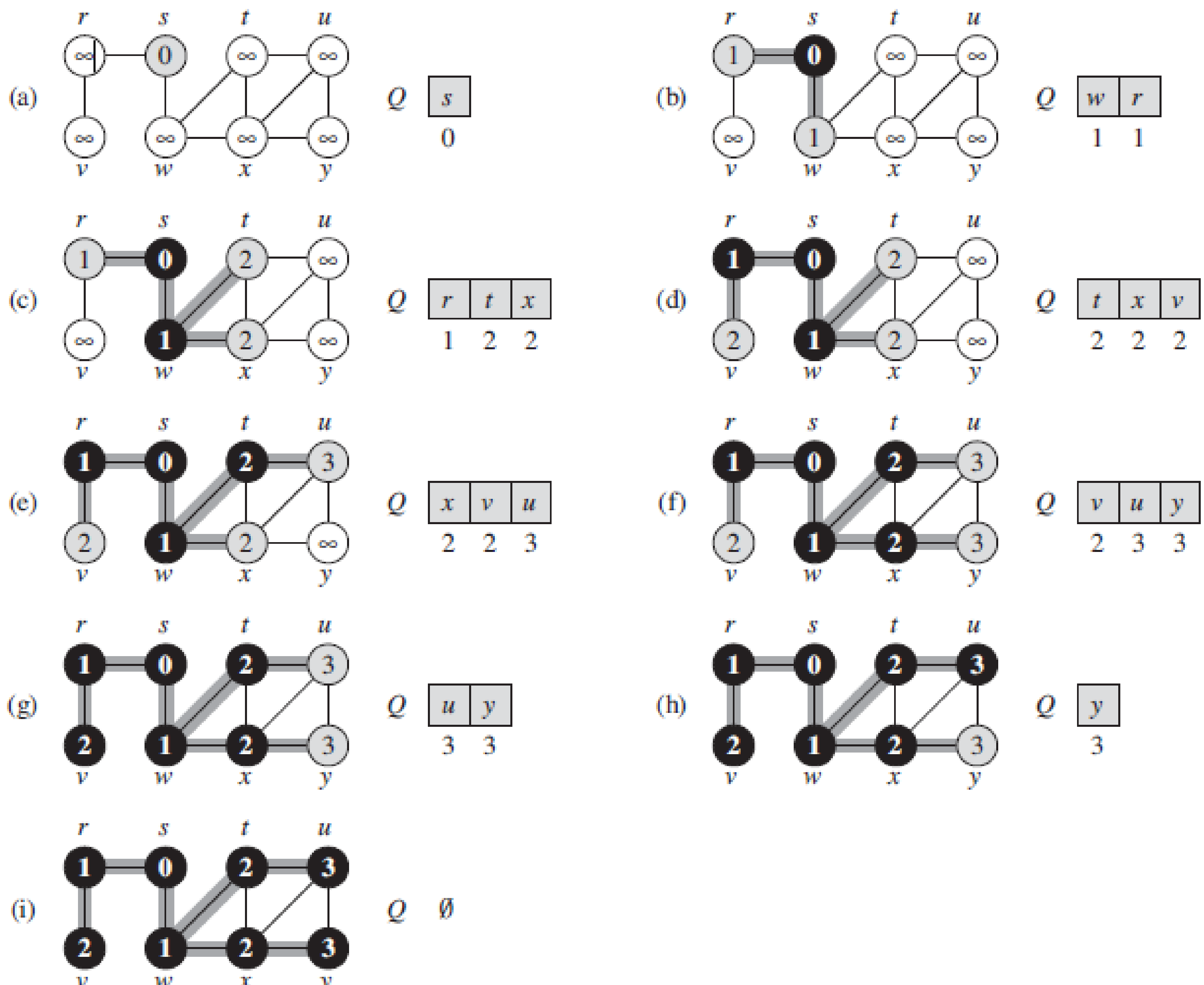
- Tokom rada algoritma svaki čvor se boji u:
  - **belo** – još nije uzet u obradu,
  - **sivo** – čeka na obradu, i
  - **crno** – obrađen.

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

BFS algoritam

```
BFS(G, s)
for each u ∈ G.V \ {s}   // za svaki čvor (sem s)
   u.color = WHITE
   u.d = ∞
   u.pred = NIL
s.color = GRAY           // čvor izvora
s.d = 0
s.pred = NIL
Q = ∅                    // neobrađeni čvorovi
ENQUEUE(Q, s)            // dodaj izvor u neobrađene
while Q ≠ ∅
   u = DEQUEUE(Q)        // uzmi prvi od neobrađenih
   for each v ∈ G.Adj(u)
      if v.color == WHITE
         v.color = GRAY
         v.d = u.d + 1     // udaljenost od izvora s
         v.pred = u
         ENQUEUE(Q, v)   // dodaj u neobrađene
   u.color = BLACK
```

BFS primer

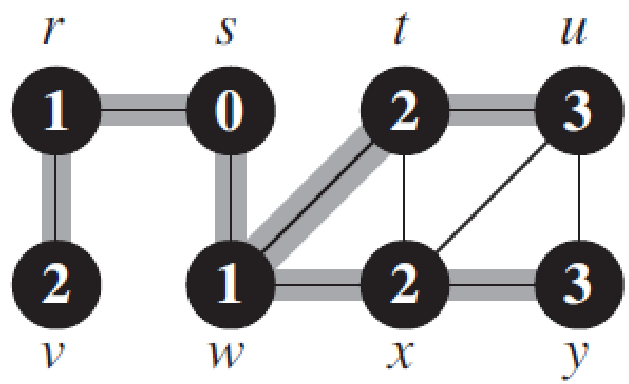

(a)
(b)
(c)
(d)
(e)
(f)
(g)
(h)
(i)
r s t u
v w x y
Q s
0
Q w r
1 1
Q r t x
1 2 2
Q t x v
2 2 2
Q x v u
2 2 3
Q v u y
2 3 3
Q u y
3 3
Q y
3
Q Ø

BFS vreme izvršavanja

- Posle inicijalizacije, algoritam nikada ne beli čvor, a na osnovu uslova (red #13) se dodaju u Queue samo beli čvorovi.
- Ukupan broj Enqueue i Deqeue operacija (složenosti $O(1)$) je n.
- Kada se čvor ubaci u *queue* 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 $O(n + m)$

Stablo pretrage u širinu

- 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 i s je (.π je oznaka za .pred):

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

Ispisivanje putanje između čvorova

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

```
PRINT-PATH(G, s, v)
 if v == s
    print s
 elseif  v.pred == NIL
    print “nema putanje od “ s “ do “ v
 else PRINT-PATH(G, s, v.pred)
    print v
```

Pretraga u dubinu (DFS) – način rada

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

$$G_\pi = (V, E_\pi)$$
$$E_\pi = \{(v.\pi, v) : v \in V \ \land v.\pi \neq \mathrm{NIL}\}$$

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

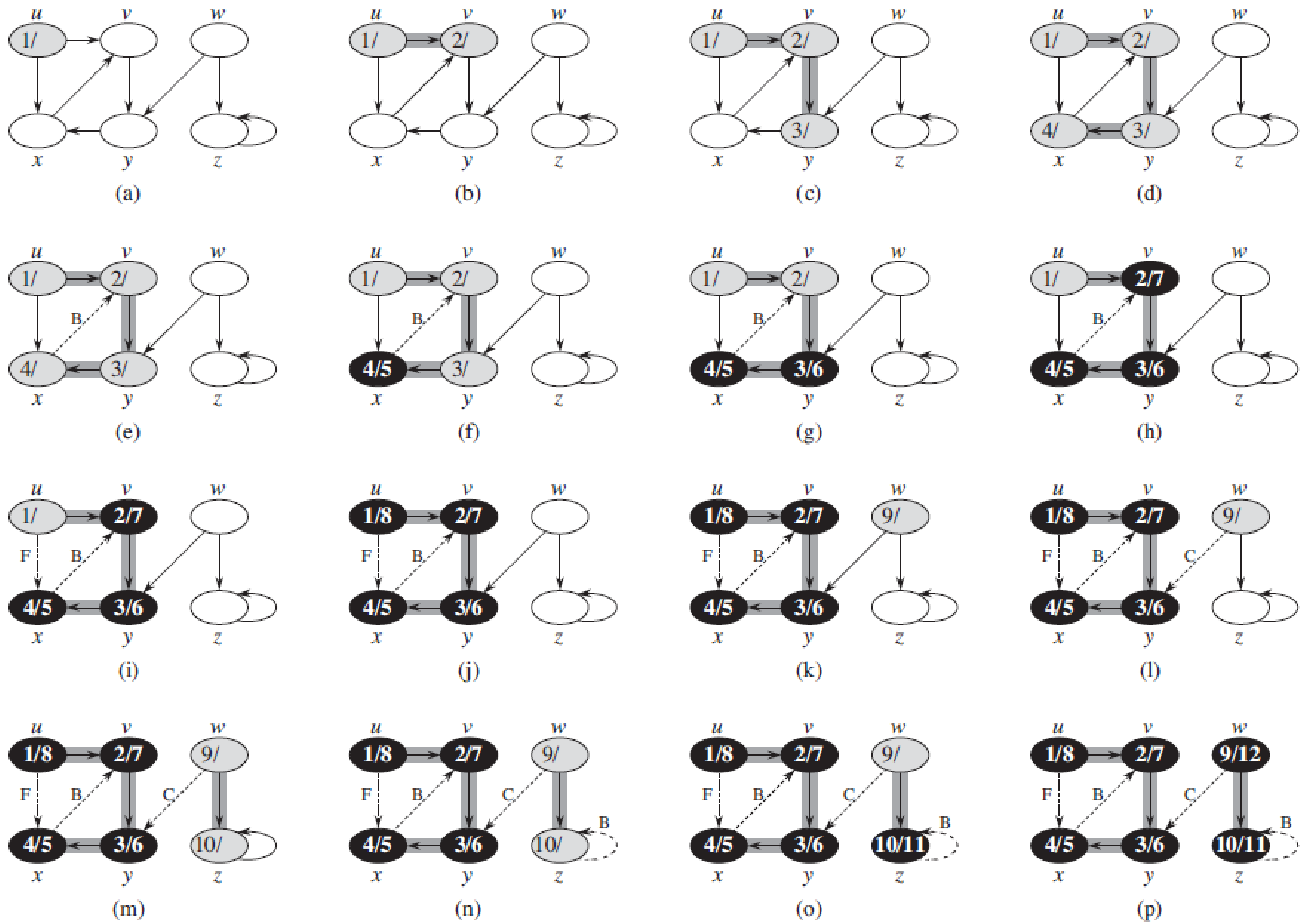
DFS algoritam

- Rekurzivno pretražuje graf
- Pazi da ne ponovi ranije posećene čvorove

```
DFS(G)
  for each u ∈ G.V
     u.color = WHITE
     u.pred = NIL
  time = 0
  for each u ∈ G.V
     if u.color == WHITE
        DFS-VISIT(G,u)
```

```
DFS-VISIT(G,u)
  time = time + 1
  u.d = time
  u.color = GRAY
  for each v ∈ G.Adj(u)
     if v.color == WHITE
        v.pred = u
        DFS-VISIT(G,v)
  u.color = BLACK
  time = time + 1
  u.f = time
```

DFS primer



u v w
1/
x y z
(a)
1/ 2/
(b)
1/ 2/ 3/
(c)
1/ 2/ 4/ 3/
(d)
1/ 2/ B 4/ 3/
(e)
1/ 2/ B 4/5 3/
(f)
1/ 2/ B 4/5 3/6
(g)
1/ 2/7 B 4/5 3/6
(h)
1/ 2/7 F B 4/5 3/6
(i)
1/8 2/7 F B 4/5 3/6
(j)
1/8 2/7 9/ F B 4/5 3/6
(k)
1/8 2/7 9/ F B C 4/5 3/6
(l)
1/8 2/7 9/ F B C 4/5 3/6 10/
(m)
1/8 2/7 9/ F B C B 4/5 3/6 10/
(n)
1/8 2/7 9/ F B C B 4/5 3/6 10/11
(o)
1/8 2/7 9/12 F B C B 4/5 3/6 10/11
(p)

DFS vreme izvršavanja

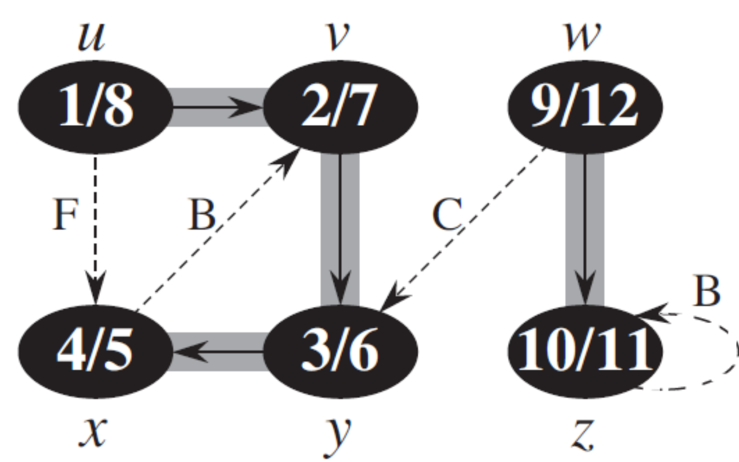
- Vreme izvršavanja DFS algoritma je $\Theta(n+m)$, $n=|V|$, $m=|E|$ - isto kao i kod BFS (iz istih razloga).

Klasifikacija grana na osnovu DFS

U usmerenom grafu:

- **Grane stabla** (*tree edge*) – grane preko kojih se posećuju čvorovi (u primeru debele sive grane)
- **Preskočna grana** (*forward edge*) – ka čvoru u stablu dalje od tekućeg
- **Povratne grane** (*backward edge*) – ka prethodnom čvoru u stablu
- **Unakrsne grane** (*cross edge*) – između dva podstabla (koja nemaju zajedničke čvorove - pretke)

Koji od tipova grana postoje u neusmerenom grafu?



Detekcija kružne putanje

- Da li graf ima kružu putanju (ciklus)?

- 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

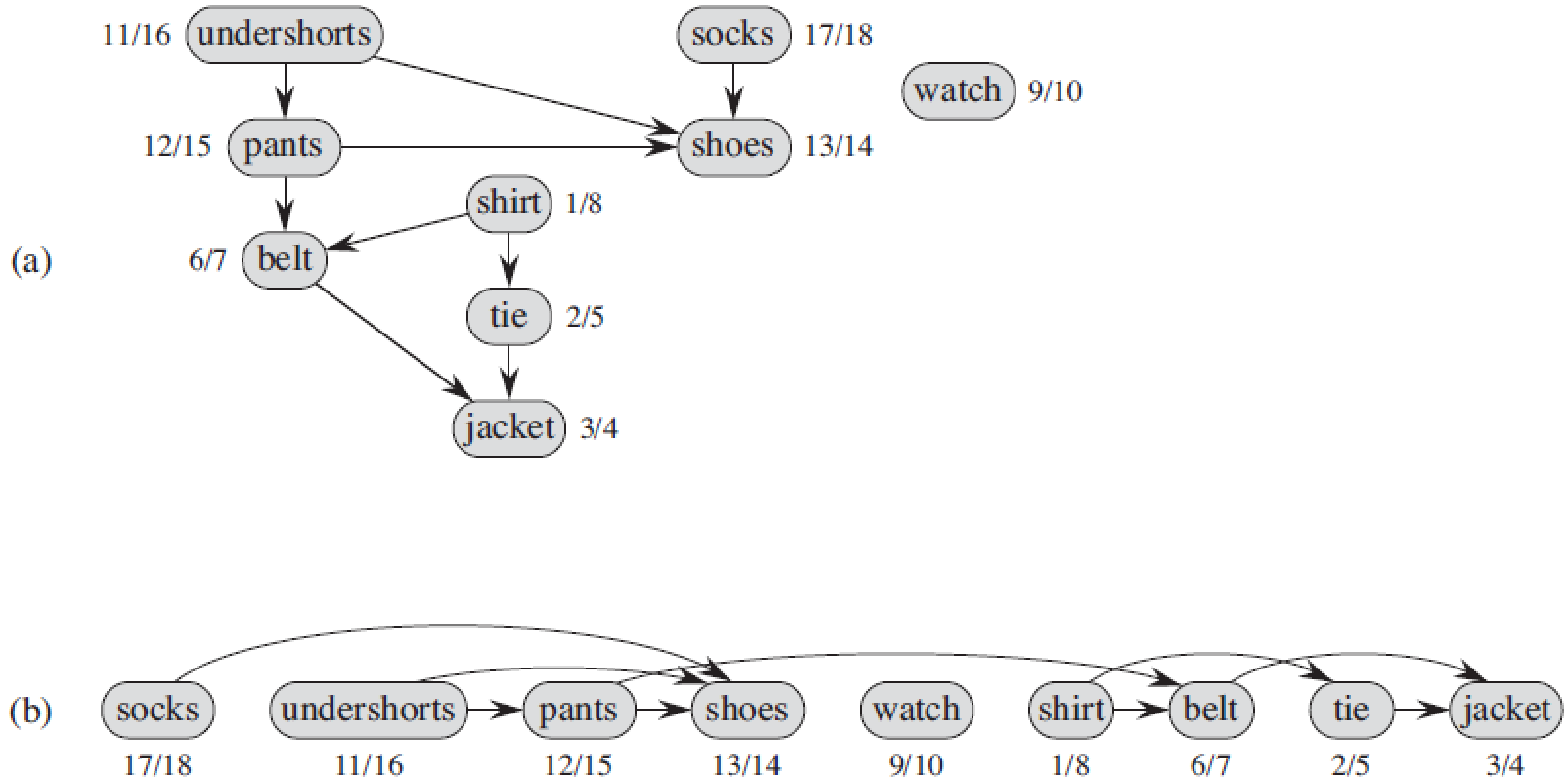
- Topološko sortiranje **direktnog acikličnog grafa** (DAG) se može sprovesti na osnovu DFS algoritma
  - DAG nema kružne putanje (cikluse)!

- Algoritam:

```
TOPOLOGICAL-SORT(G)
  Pozvati DFS da izračuna v.f za svaki čvor v
  Kako je čvor obrađen smestiti ga u listu L
  Obrnutu listu L vratiti kao rezultat
```

- Vreme izvršavanja algoritma je $\Theta(n + m)$

Primer topološkog sortiranja



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

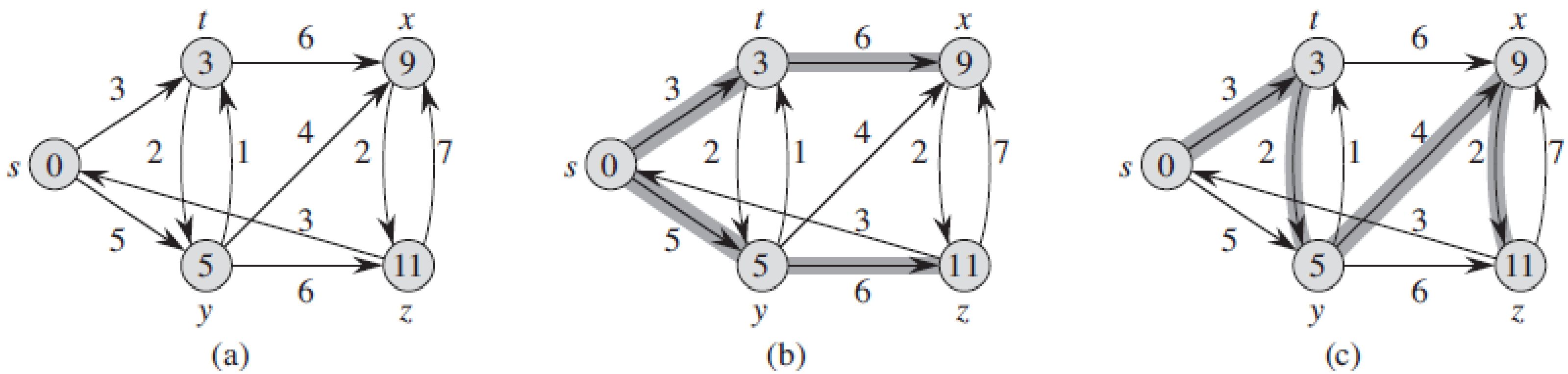
BFS nalazi najkraći put u netežinskog grafu

- razmatra se broj grana

Najkraći put u težinskom grafu:

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

Primer najkraćeg puta



Tekuća težina (rastojanje): $v.d$

Prethodni čvor: $v.pred \equiv \pi(v)$

Za najkraći put važi da je: $v.d = \delta(s, v)$

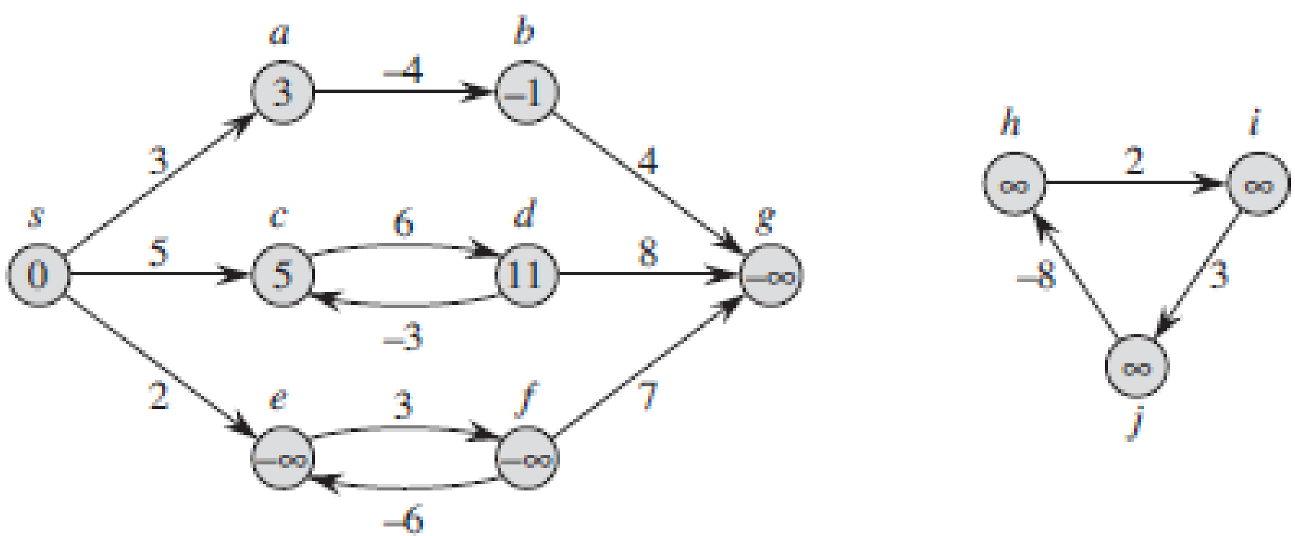
Najkraća putanja je od *s* do *v*: $v - \pi(v) - \pi\big(\pi(v)\big) - \pi\Big(\pi\big(\pi(v)\big)\Big) - \cdots - s$

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?



Generalna struktura algoritma najkraćeg puta

(Ako nema negativnih zatvorenih putanja)

Algoritam „grube sile“:

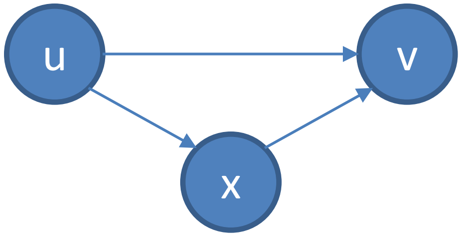
```
Inicijalizacija: for each v∈G.V, v.d=∞, v.pred=NIL
                 s.d = 0
Ponavljaj: izaberi granu (u,v) // nekako?!
  Relaksiraj granu (u,v): if v.d > u.d + w(u,v)
                             v.d = u.d + w(u,v)
                             v.pred = u
dok ne bude za sve grane v.d ≤ u.d + w(u,v)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

Generalna struktura algoritma najkraćeg puta

(nastavak)

- Problem: loš izbor grana može dovesti do eksponencijalnog trajanja algoritma (npr. $2^{\frac{n}{2}}$)

- 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 (DAG = *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

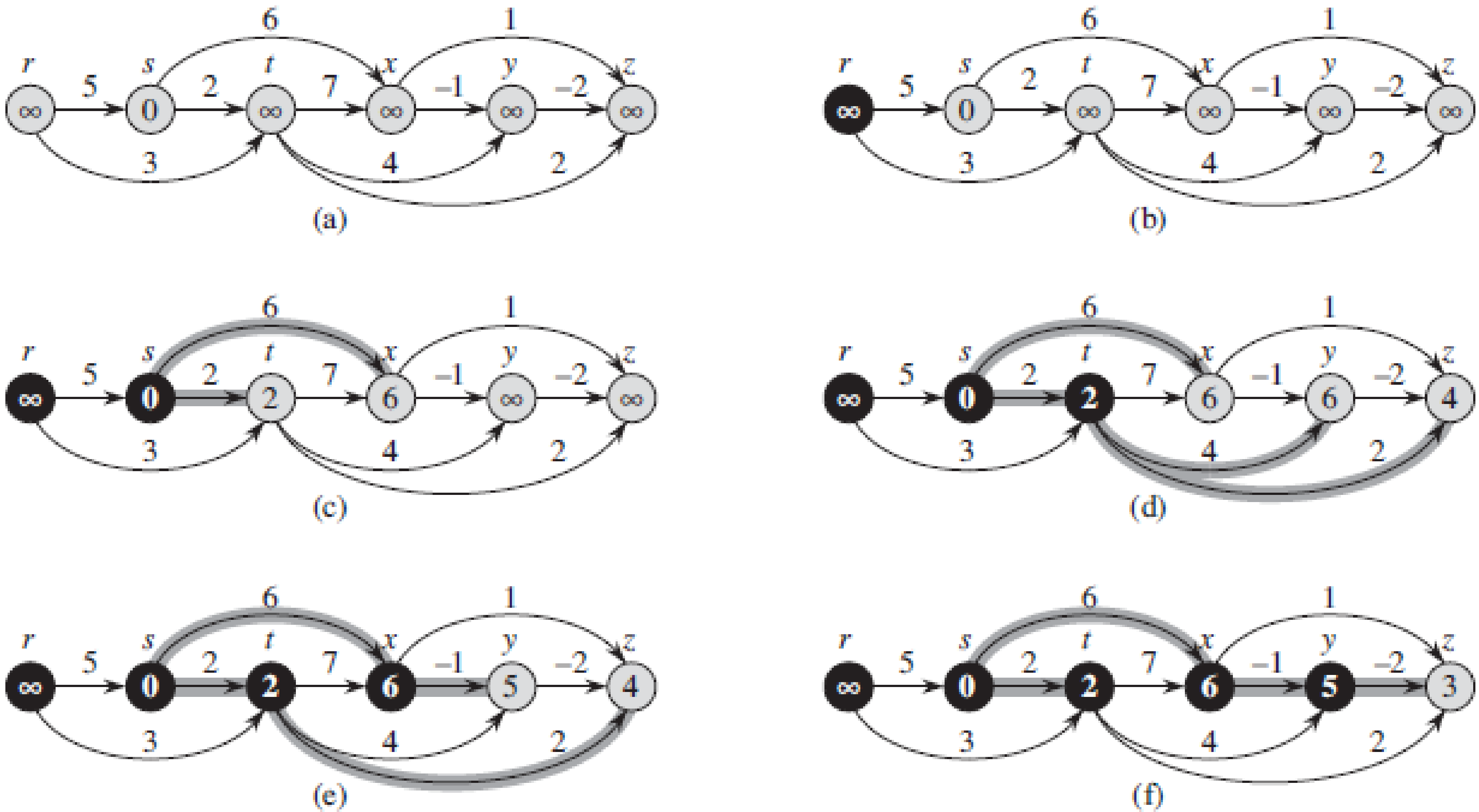
```
DAG-SHORTEST-PATHS(G,w,s)
Topološki sortirati čvorove G u listu L
INITIALIZE-SINGLE-SOURCE(G,s)
for each čvor u, iz Liste L
   for each v ∈ G.Adj(u)
      RELAX(u,v,w)
```

Najkraći put – rutina inicijalizacije i Relax

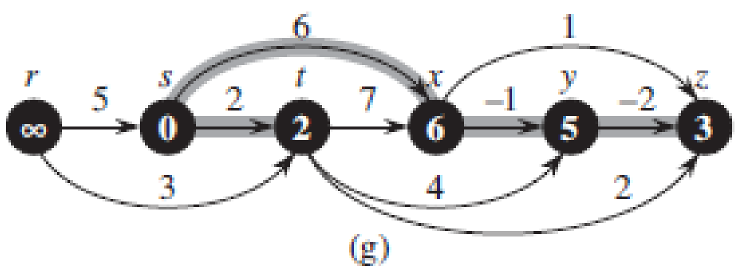
```
INITIALIZE-SINGLE-SOURCE(G,s)
  for each v ∈ G.V
     v.d = ∞
     v.pred = NIL
  s.d = 0
```

```
RELAX(u,v,w)
  if v.d > u.d + w(u,v)
     v.d = u.d + w(u,v)
     v.pred = u
```

Primer najkraćeg puta u DAG



(a) (b) (c) (d) (e) (f)



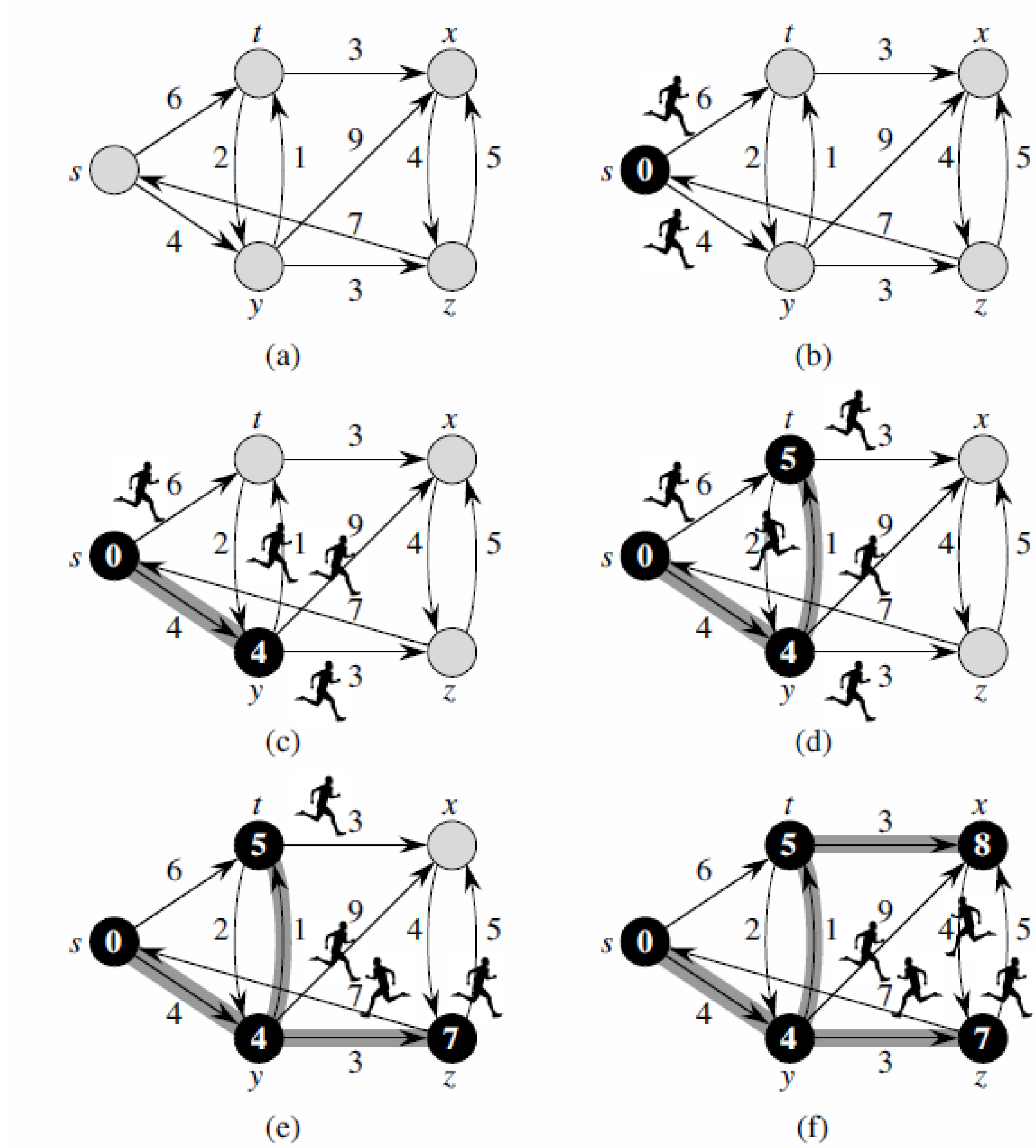
(g)

```
DAG-SHORTEST-PATHS(G,w,s)
    Topološki sortirati čvorove G u listu L
    INITIALIZE-SINGLE-SOURCE(G,s)
    for each čvor u, iz Liste L
        for each v ∈ G.Adj(u)
            RELAX(u,v,w)
```

Dijkstra algoritam - osobine

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

Princip rada *Dijkstra* algoritma

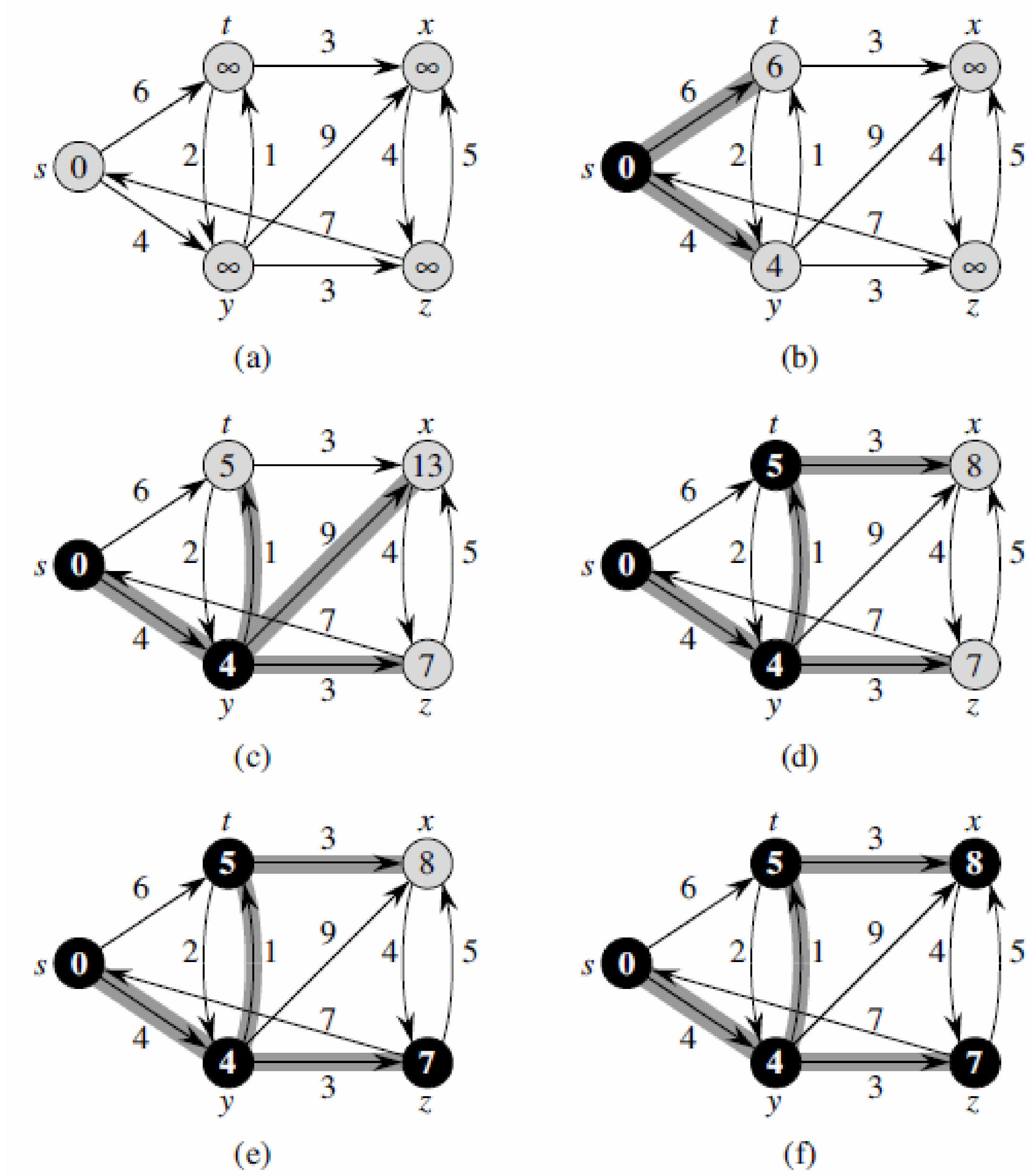


Dijkstra algoritam

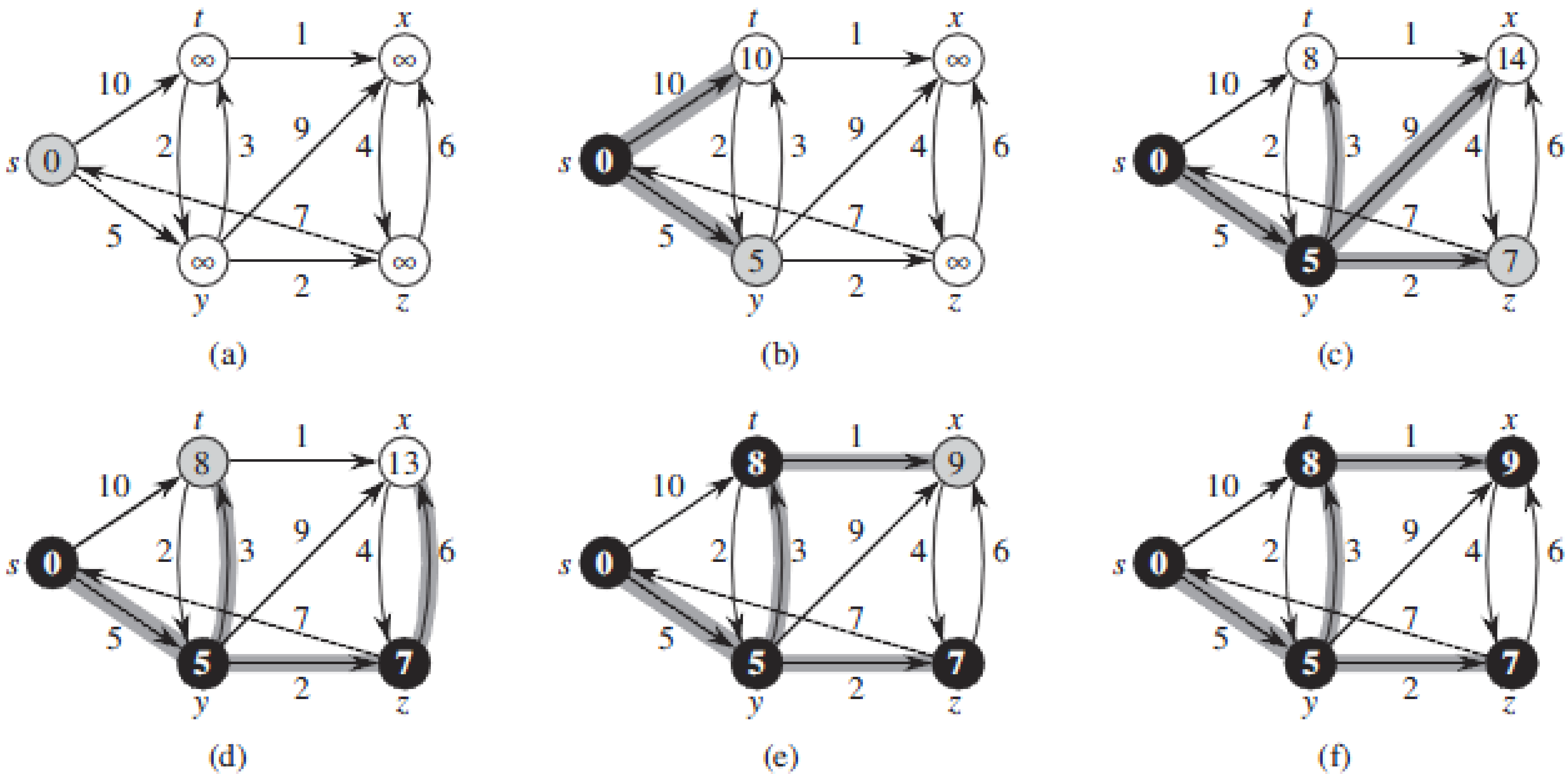
Algoritam održava skup čvorova S gde su rastojanja do polaznog čvora određena (neće se menjati). Zatim bira čvor $u \in V \setminus S$ sa najmanjim rastojanjem $u.d$, dodaje ga u S i koriguje (relaksira) rastojanja do svih čvorova preko grana koje izlaze iz u .

```
DIJKSTRA(G,w,s)
  INITIALIZE-SINGLE-SOURCE(G,s)
  S = ∅
  Q = G.V
  while Q ≠ ∅
     u = EXTRACT-MIN(Q)
     S = S ∪ {u}
     for each v ∈ G.Adj(u)
        RELAX(u,v,w)
```

Dijkstra – primer A



Dijkstra – primer B



Vreme izvršavanja *Dijkstra* algoritma

- Algoritam održava red sa minimalnim prioritetima Q (*min-priority queue*)
  $n = |V|, m = |E|$
  - poziva Insert operaciju (red #3)   broj poziva *n*
  - poziva Extract-Min (red #5)   broj poziva *n*
  - poziva Decrease-Key u Relax   broj poziva *m*
- Trajanje algoritma zavisi od načina implementacije Q

```
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 v ∈ G.Adj(u)
8           RELAX(u,v,w)
```

Vreme izvršavanja *Dijkstra* algoritma (2)

- Implementacija Q nizom indeksiranim rednim brojem čvora (1..*n*)
  - Trajanja operacija:
    - Insert = $\Theta(1)$
    - Extract-Min = $\Theta(n)$ jer se pretražuje ceo niz
    - Decrease-Key = $\Theta(1)$
  - Vreme izvršavanja je $n\Theta(1) + n\Theta(n) + m\Theta(1) = \Theta(n^2 + m) = \Theta(n^2)$

- implementacija Q upotrebom binarnog *min_heap*-a
  - Trajanja operacija:
    - Insert = $\Theta(\log_2 n)$
    - Extract-Min = $\Theta(\log_2 n)$
    - Decrease-Key = $\Theta(\log_2 n)$
  - Vreme izvršavanja $(2n + m)\Theta(\log_2 n) = \Theta((m + n)\log_2 n)$

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

Vreme izvršavanja *Dijkstra* algoritma (3)

- implementacija Q upotrebom *Fibonacci heap*-a
  - Trajanja operacija:
    - Extract-Min = $\Theta(\log_2 n)$ jer se pretražuje ceo niz
    - Decrease-Key = $\Theta(1)$ – amortizovano vreme
  - Vreme izvršavanja $\Theta(m + n\log_2 n)$

- Amortizovana složenost $\Theta(1)$: za n poziva je složenost $\Theta(n)$ tako da se može smatrati da je prosečno trajanje poziva $\Theta(1)$, mada se kod pojedinih poziva može desiti duže trajanje.

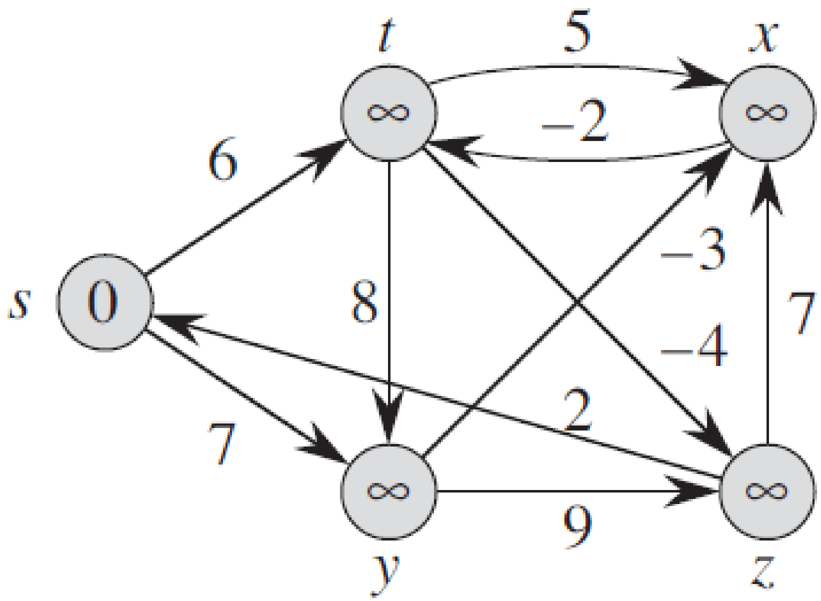
Bellman-Ford algoritam - osobine

- Rešava problem traženja najkraćih puteva i kada postoje negativne težine grana
- Otkriva postojanje kružnih putanja negativnog pojačanja

- Radi u polinomskom vremenu

Bellman-Ford algoritam

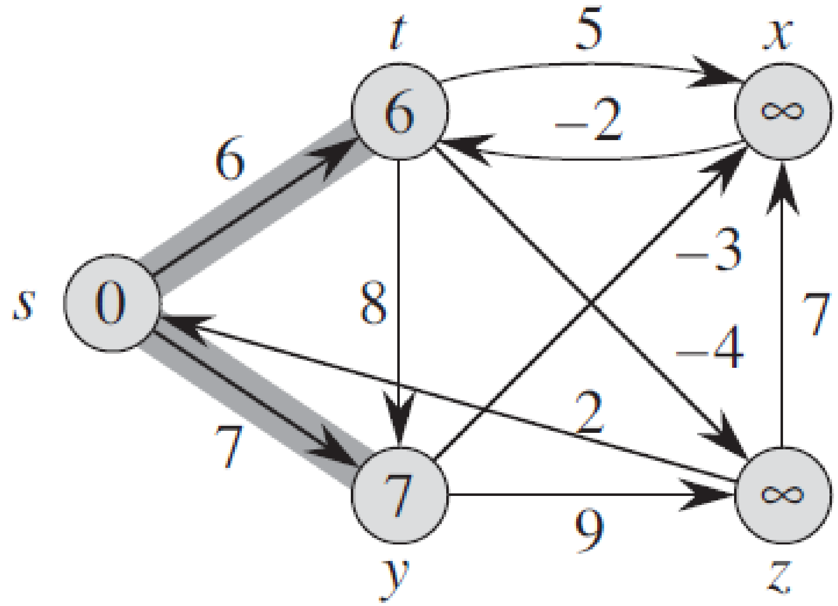
```
BELLMAN-FORD(G,w,s)
  INITIALIZE-SINGLE-SOURCE(G,s)
  for i = 1 to |V|-1
     for each (u,v) ∈ E
        RELAX(u,v,w)
  for each (u,v) ∈ E                // provera
     if v.d > u.d + w(u,v)
        return „postoji negativna kružna putanja“
```




```
BELLMAN-FORD(G,w,s)
    INITIALIZE-SINGLE-SOURCE(G,s)
    for i = 1 to |V|-1
       for each (u,v) ∈ E
          RELAX(u,v,w)
    for each (u,v) ∈ E                   // provera
       if v.d > u.d + w(u,v)
          return „postoji negativna kružna putanja“
```

Redosled grana

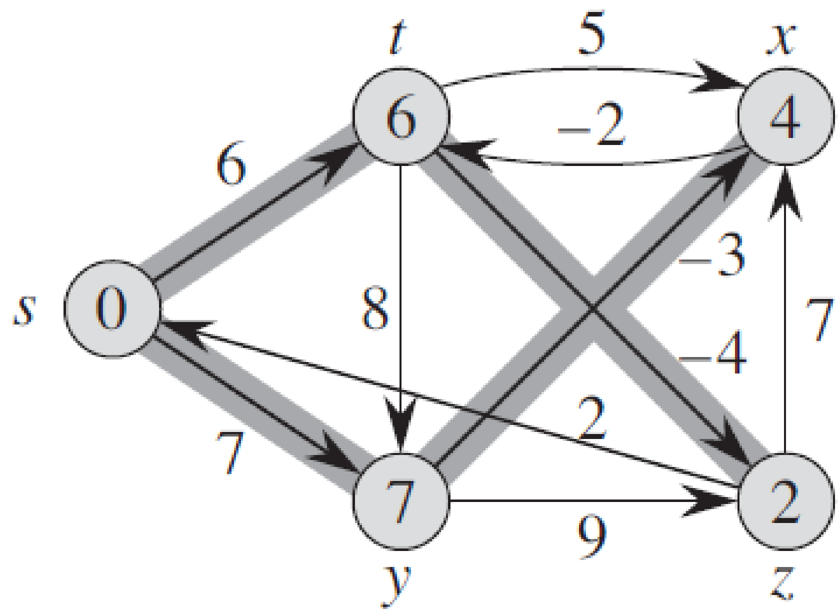
$(t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x),(z,s),(s,t),(s,y)$




```
BELLMAN-FORD(G,w,s)
   INITIALIZE-SINGLE-SOURCE(G,s)
   for i = 1 to |V|-1                       // i=1
      for each (u,v) ∈ E
         RELAX(u,v,w)
   for each (u,v) ∈ E                       // provera
      if v.d > u.d + w(u,v)
         return „postoji negativna kružna putanja“
```

Redosled grana

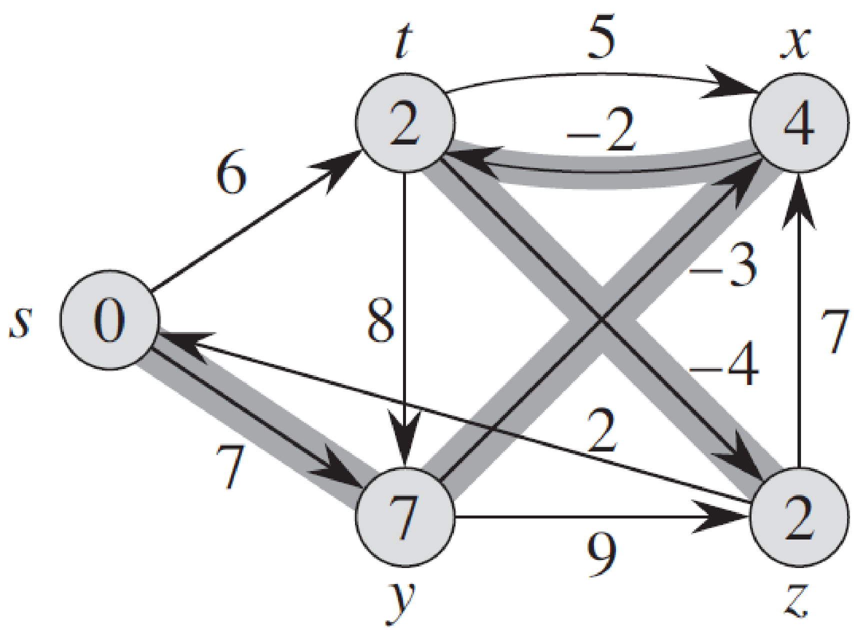
$(t, x), (t, y), (t, z), (x, t), (y, x), (y, z), (z, x), (z, s), (s, t), (s, y)$




```
BELLMAN-FORD(G,w,s)
  INITIALIZE-SINGLE-SOURCE(G,s)
  for i = 1 to |V|-1                          // i=2
     for each (u,v) ∈ E
        RELAX(u,v,w)
  for each (u,v) ∈ E                          // provera
     if v.d > u.d + w(u,v)
        return „postoji negativna kružna putanja“
```

Redosled grana

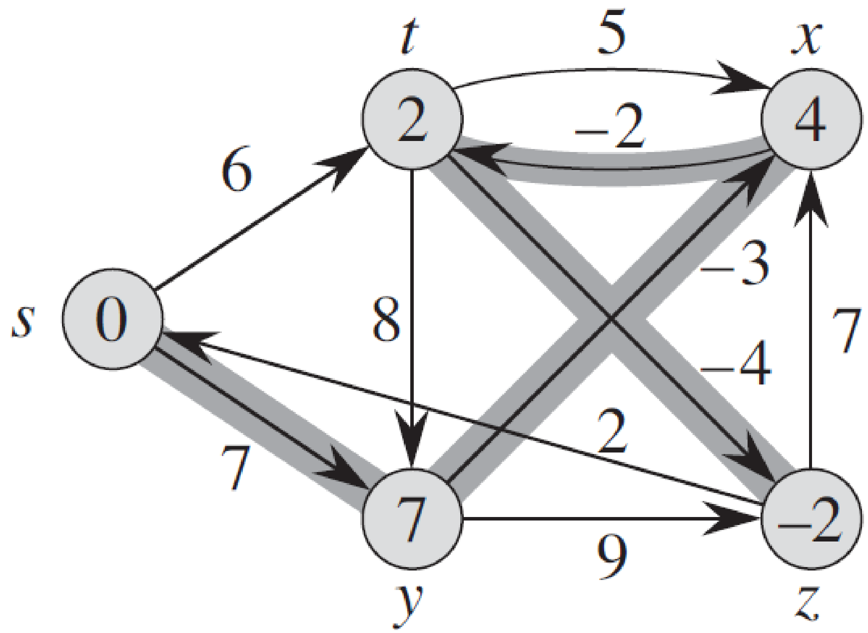
$(t,x), (t,y), (t,z), (x,t), (y,x), (y,z), (z,x), (z,s), (s,t), (s,y)$




```
BELLMAN-FORD(G,w,s)
  INITIALIZE-SINGLE-SOURCE(G,s)
  for i = 1 to |V|-1                          // i=4
     for each (u,v) ∈ E
        RELAX(u,v,w)
  for each (u,v) ∈ E                          // provera
     if v.d > u.d + w(u,v)
        return „postoji negativna kružna putanja“
```

Redosled grana

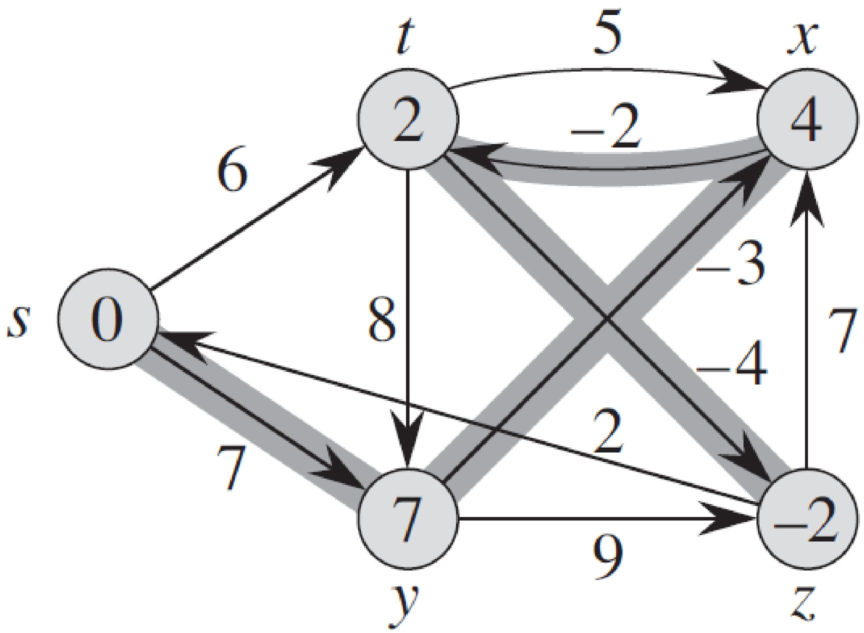
$(t, x), (t, y), (t, z), (x, t), (y, x), (y, z), (z, x), (z, s), (s, t), (s, y)$




```
BELLMAN-FORD(G,w,s)
    INITIALIZE-SINGLE-SOURCE(G,s)
    for i = 1 to |V|-1                          // i=4
        for each (u,v) ∈ E
            RELAX(u,v,w)
    for each (u,v) ∈ E                          // provera
        if v.d > u.d + w(u,v)
            return „postoji negativna kružna putanja“
```

Redosled grana

$(t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x),(z,s),(s,t),(s,y)$




```
BELLMAN-FORD(G,w,s)
  INITIALIZE-SINGLE-SOURCE(G,s)
  for i = 1 to |V|-1
     for each (u,v) ∈ E
        RELAX(u,v,w)
  for each (u,v) ∈ E                        // provera
     if v.d > u.d + w(u,v)
        return „postoji negativna kružna putanja“
```

Složenost *Bellman-Ford* algoritma

- Ukupan broj prolaza je $(n-1)m + m$: $O(nm)$
- Zašto $n-1$ prolaz?
  U najgorem slučaju najkraću putanju čine svi čvorovi grafa. Direktna putanja u kojoj je n čvorova ima $n-1$ granu. Svakim prolazom se pronalazi najkraće rastojanje do narednog $(i+1)$ čvora u najkraćoj putanji.

```
BELLMAN-FORD(G,w,s)
  INITIALIZE-SINGLE-SOURCE(G,s)
  for i = 1 to |V|-1                  // (n-1) prolaza
     for each (u,v) ∈ E               // m prolaza
        RELAX(u,v,w)
  for each (u,v) ∈ E                  // m prolaza
     if v.d > u.d + w(u,v)
        return „postoji negativna kružna putanja“
```