Strukture podataka
Letnji semestar

Grafovi

Prof. Dr Leonid Stoimenov

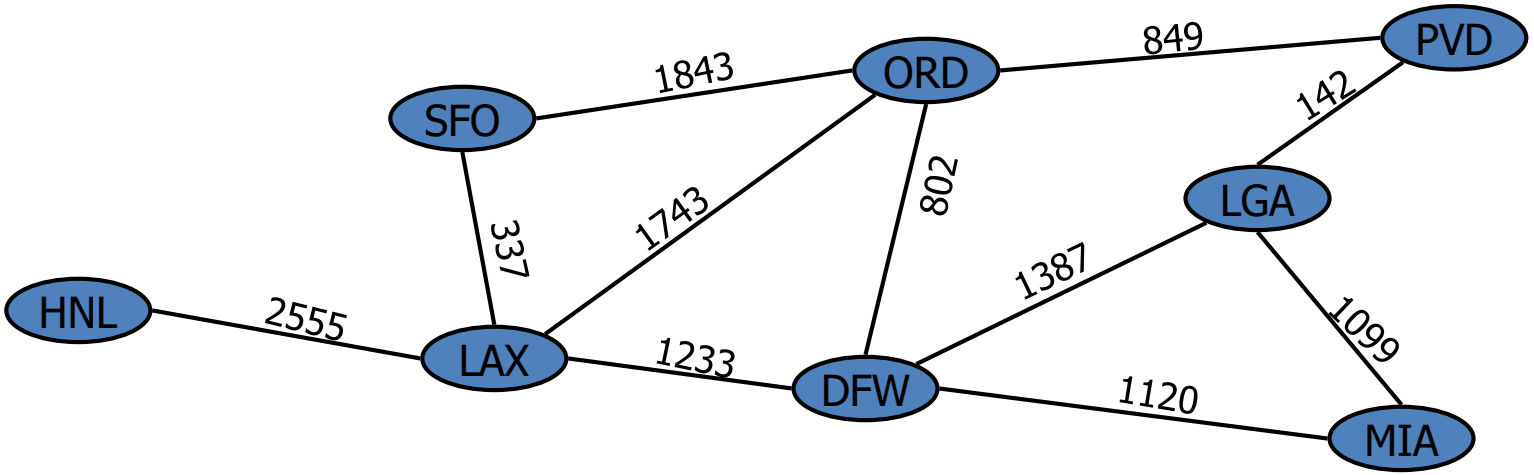
Katedra za računarstvo
Elektronski fakultet u Nišu

Graf - Pregled

- Graf
  - Definicije
  - Terminologija
  - Memorijska reprezentacija
    - Sekvencijalno - matrice
    - Lančano
- Operacije za rad sa grafom
  - ADT
  - Osnovne operacije
  - Traženje puteva
  - Obilazak grafa
  - Topološko sortiranje

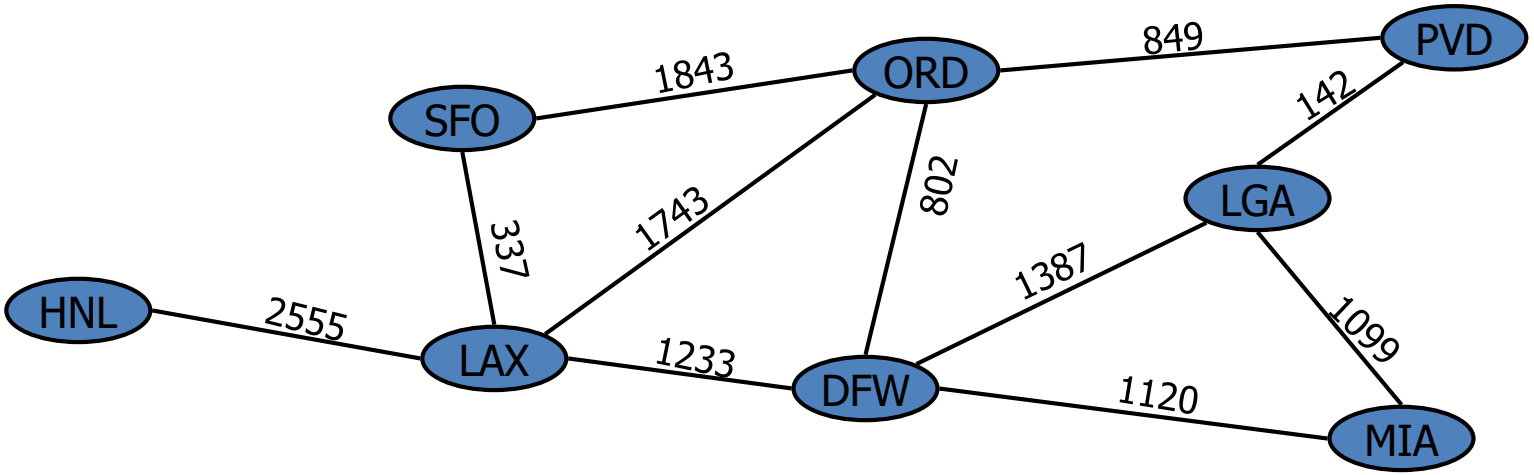
Graf – primer

- Šta je graf:
  - Graf se sastoji od niza čvorova i niza potega.
  - Svaki poteg u grafu je određen parom čvorova.
- Primer:
  - Čvorovi su aerodromi i sadrže kod aerodroma
  - Poteg predstavlja rutu leta između dva aerodorma i sadrži vazdušno rastojanje



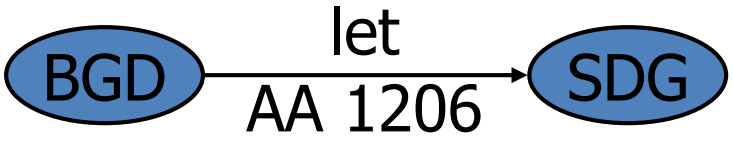
GRAF - DEFINICIJE

- Graf je uređeni par (***V, E***), gde je
  - ***V*** skup temena *v*, koji se zovu i čvorovi
  - ***E*** je kolekcija parova čvorova *e=[u,v]*, koji čine potege ili grane
- Čvorovi i grane čuvaju odgovarajuće vrednosti (graf je obeležen)
- Svakom potegu u grafu može biti pridružen broj i takav graf se naziva težinski graf ili mreža.
- Nenegativan broj pridružen potegu naziva se težina.

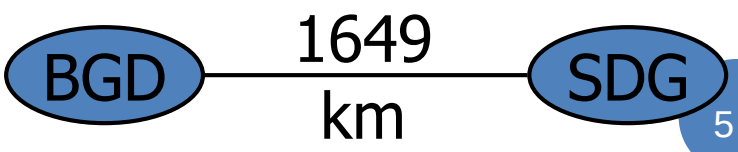


TERMINOLOGIJA

- **Orijetisani poteg**
  - Uređeni par čvorova (***u***,***v***)
  - Prvi čvor ***u*** je **početni**
  - Poteg iz njega **izvire**
  - drugi čvor ***v*** je **odredišni**
  - Poteg u njega **uvire**
  - Primer: avio let
- Orijetisani graf - digraf
  - Svi potezi su orijentisani
  - Primer: Mreža avio ruta

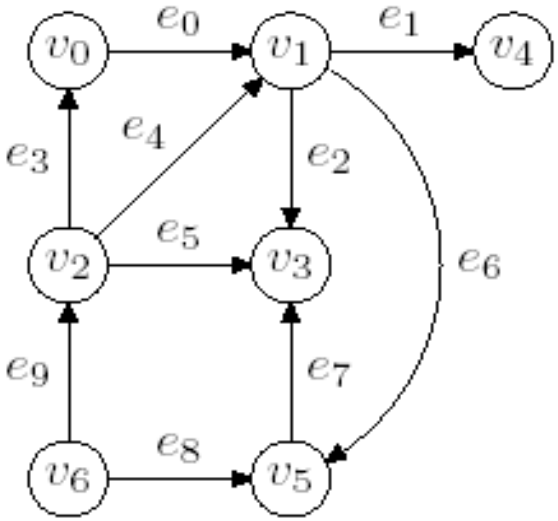


- **Neorijetisani poteg**
  - Neuređeni par (***u***,***v***)
  - Primer: trasa avio leta
- Neorijetisani graf
  - Svi potezi su neorijentisani
  - Primer: Mreža letova između gradova



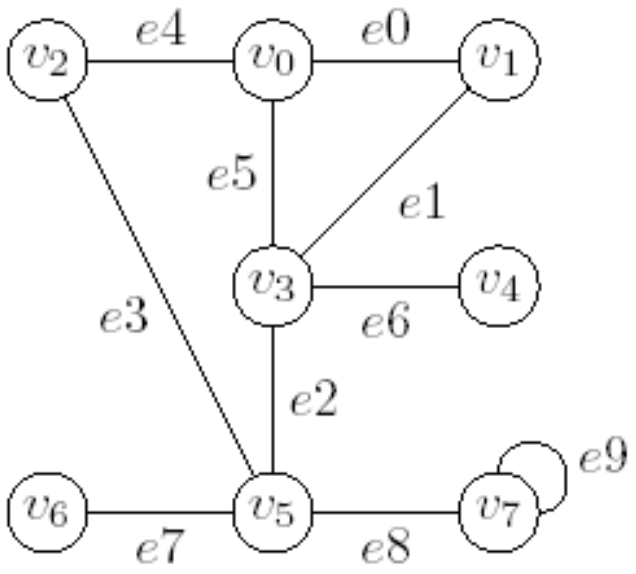
Primer grafova

- Orijentisani



V = {v0, v1, v2, v3, v4, v5, v6}
E = {e0 = (v0, v1), e1 = (v1, v4),
e2 = (v1, v3), e3 = (v2, v0), e4 = (v2, v1),
e5 = (v2, v3), e6 = (v1, v5), e7 = (v5, v3),
e8 = (v6, v5)}

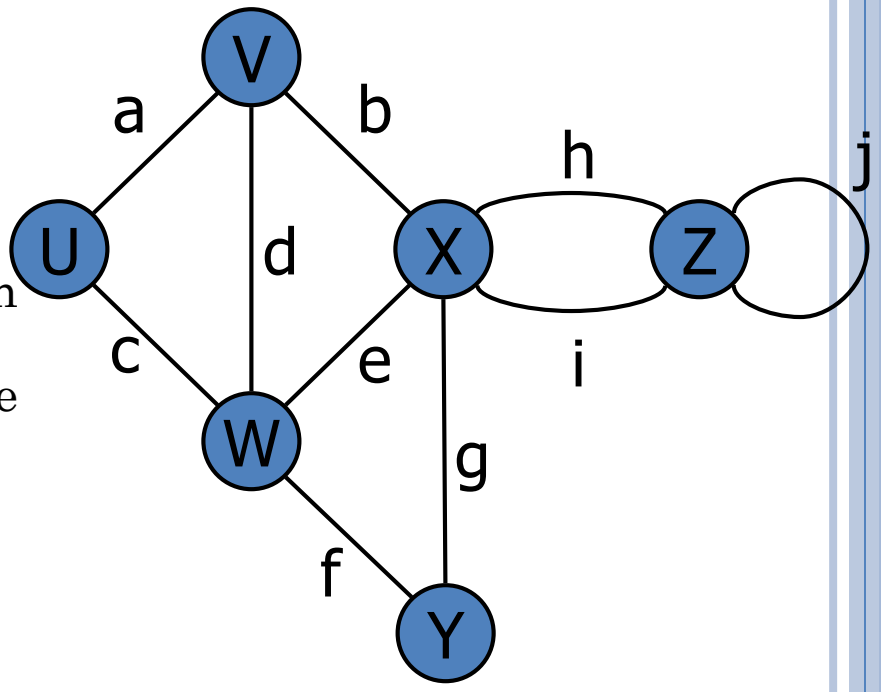
V = {v0, v1, v2, v3, v4, v5, v6, v7}
E = {e0 = {v0, v1}, e1 = {v1, v3},
e2 = {v3, v5}, e3 = {v2, v5}, e4 = {v0, v2},
e5 = {v0, v3}, e6 = {v3, v4}, e7 = {v5, v6},
e8 = {v5, v7},
e9 = {v7, v7}}



- Neorijentisani

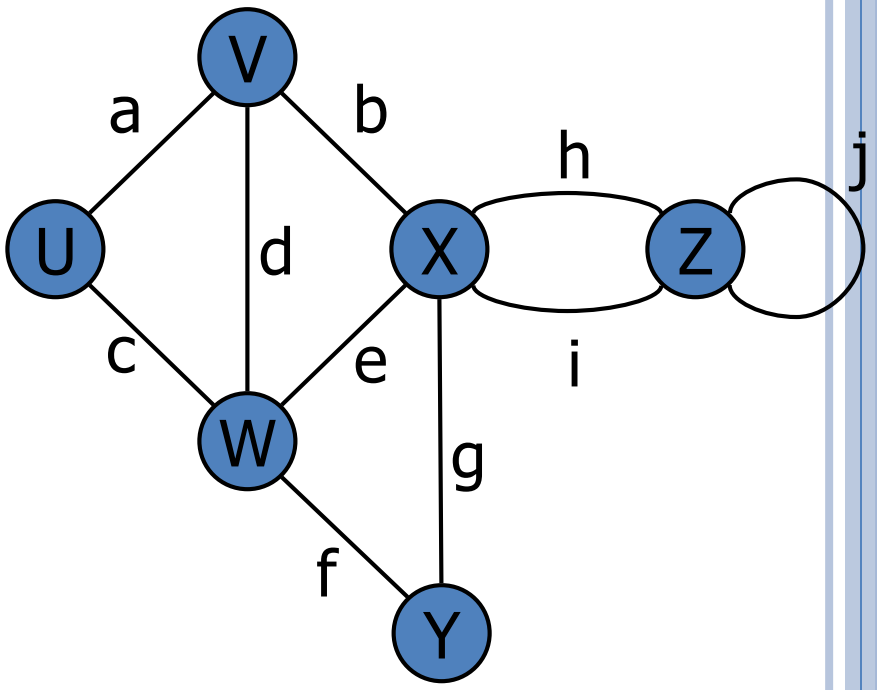
Terminologija (nast.)

- **Završni čvorovi** nekog potega
  - Krajnje tačke potega
  - U i V su završni čvorovi za a
- **Incidentnost** potega i čvorova
  - Grana e je incidentna (vezana) za čvor v ukoliko je v jedna od krajnjih tačaka potega
  - Čvor v je incidentan potegu x ako je v jedan od čvorova u uređenom paru čvorova koji čine poteg x.
  - a, d, i b su *incidentni* za V
- **Susedni čvorovi**
  - Čvor u je susedan čvoru v ako postoji poteg od u do v.
  - Ako je čvor u susedan čvoru v onda se v naziva sledbenik čvora u, a čvor u je prethodnik čvora v.
  - U i V su susedni



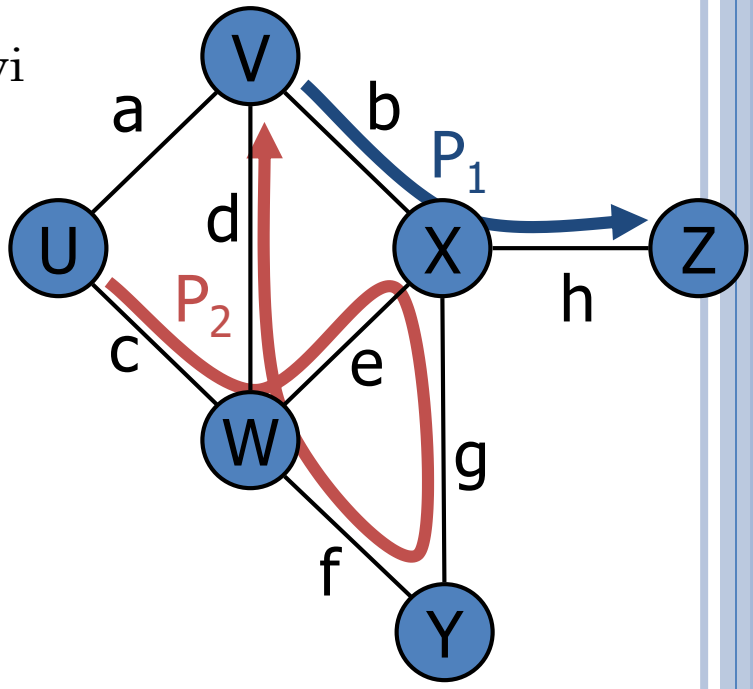
Terminologija (3)

- **Stepen čvora** deg(u)
  - broj potega incidentnih njemu
  - X ima *stepen* deg(X)=5
- **Izolovan čvor**
  - Ako je deg(u)=0
- **Orijetisani graf**
  - Ulazni stepen čvora indeg(u)
  - Izlazni stepen čvora outdeg(u)
- **Paralelni potezi**
  - Potezi između dva čvora
  - Potezi h, i su *paralelni potezi*
- **Petlja** (Self-loop)
  - Poteg koji počinje i završava se u istom čvoru.
  - j je *petlja*



Terminologija (4)

- **Put u grafu**
  - Sekvenca čvorova i potega
  - Počinje čvorom
  - Završava se čvorom
  - Za svaki poteg su poznati završni čvorovi
- ***Put dužine n*** od čvora *u* do čvora *v* se definiše kao sekvenca od n+1 čvorova ($v_0,v_1,v_2,\ldots,v_n$) tako da je $\mathbf{v_0=u}$, $\mathbf{v_n=v}$.
  - Za svaki čvor i između 1 i k važi da su čvorovi v_i i v_{i+1} susedni.
  - Dužina puta = broja grana na putu n
- **Prost put** (Simple path)
  - Put kod koga su svi čvorovi i potezi različiti
- Primer
  - P_1=(V,b,X,h,Z) je prost put
  - P_2=(U,c,W,e,X,g,Y,f,W,d,V) je put koji nije prost

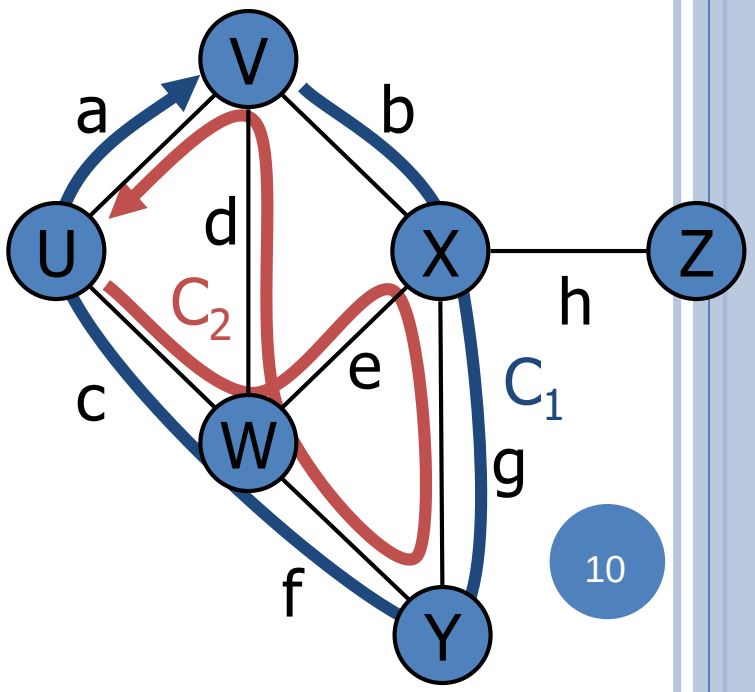


Terminologija (5)

- **Zatvoreni put**
  - Kružni put
  - $v_0=v_n$
  - Prost zatvoren put
- **Ciklus** (Cycle)
  - Cirkularna sekvenca čvorova i potega
  - Svaki poteg je definisan završnim čvorovima
  - Put od čvora do njega samog se naziva *ciklus*.
- **Prost ciklus**
  - Ciklus kod koje su svi čvorovi i potezi različiti
- Def: Zatvoren prost put dužine >=3
- **K-ciklus**
  - Ciklus dužine k
- Ciklus dužine 1 = petlja

Primer

C1=(V,b,X,g,Y,f,W,c,U,a,↵)
je prost ciklus
C2=(U,c,W,e,X,g,Y,f,W,d,V,a,↵)
je ciklus koji nije prost

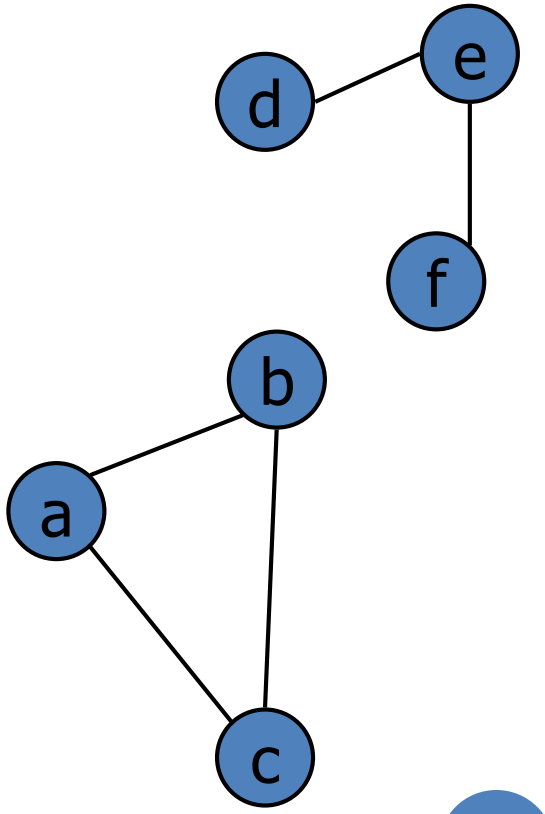


Terminologija (6)

- Aciklični graf
  - Graf koji nema cikluse
- Orijetisani aciklični graf
  - Orijentisani graf bez ciklusa
- Povezani graf
  - **akko** postoji prost put između bilo koja dva njegova čvora
- Strogo povezani graf
  - **akko** postoji samo po jedan put iz svakog čvora do svih ostalih čvorova

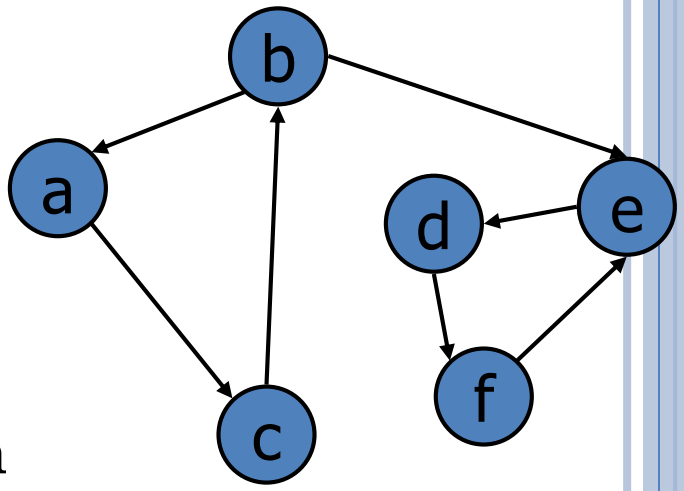
Terminologija (7)

- Neorijentisani graf je povezan ako postoji put u grafu između svakog para čvorova
- Primer: Nepovezan graf (ne postoji put između a i d)
  - V={a,b,c,d,e,f}
  - E={(a,b),(a,c),(b,c),(d,e),(e,f)}
- **Povezane** komponente – povezani podgrafovi grafa
- **Izolovane** komponente – izdvojeni podgrafovi grafa

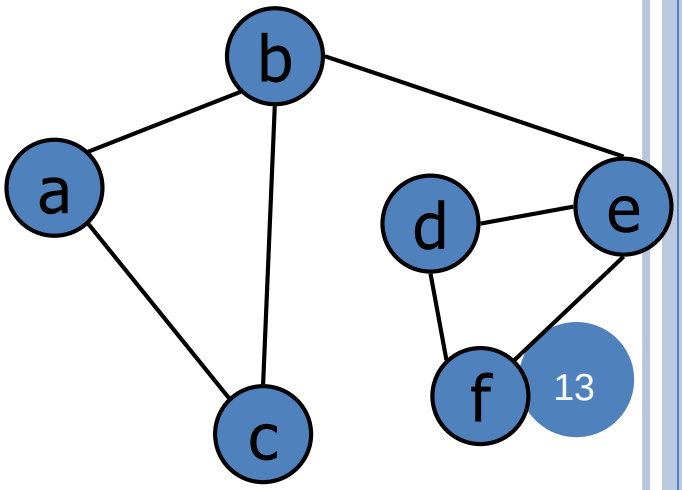


Terminologija (8)

- Orijentisani graf – strogo i slabo povezan
- Orijentisani graf je **strogo povezan** ako postoji put u grafu između svakog para čvorova
- Orijentisani graf je **slabo povezan** ako je odgovarajući neorijetisani graf povezan

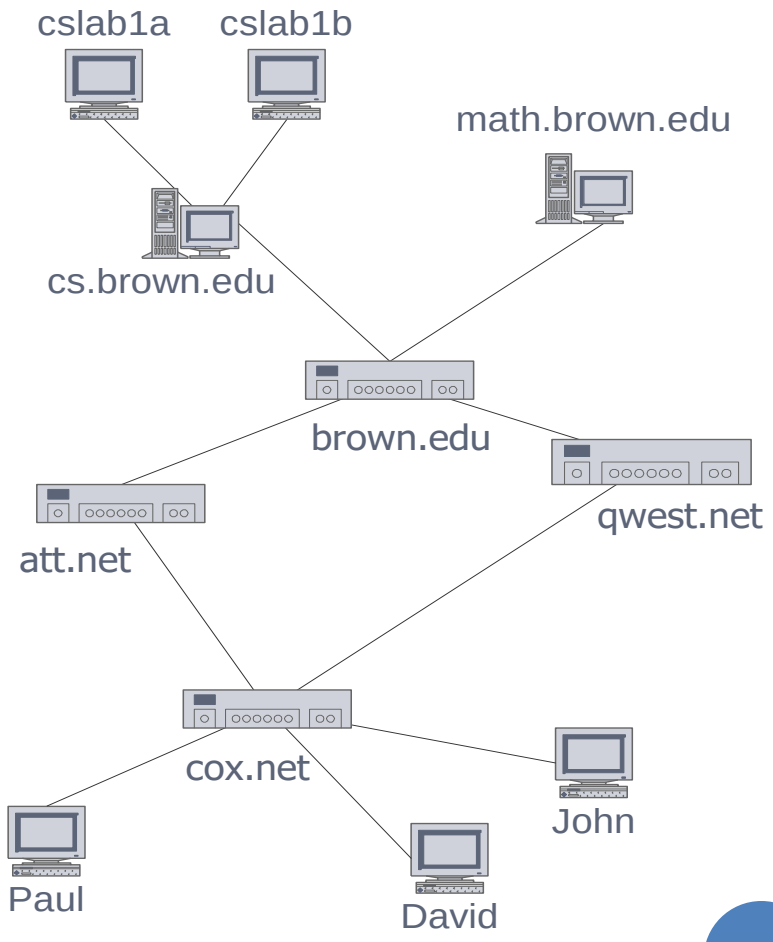


Primer: Graf nije strogo povezan, ali je slabo povezan (ne postoji put između nekog od čvorova d,e,f i čvorova a,b,c)



Primena grafova

- Elektronska kola
  - Štampane ploče
  - Integrisana kola
- Transportne mreže
  - Mreža puteva
  - Mreža letova
- Računarske mreže
  - LAN
  - Internet
  - Web
- Baze podataka
  - Entity-relationship dijagram
- ...



Sekvencijalna reprezentacija – matrica susedstva

- Matrica susedstva
  - Dimenzije: nxn, gde je n broj čvorova
  - Elementi m_{ij} matrice definišu broj potega koji spajaju čvorove i i j:

$$A_{i,j} = \begin{cases} 1 & (v_i, v_j) \in E \\ 0 & (v_i, v_j) \notin E \end{cases}$$

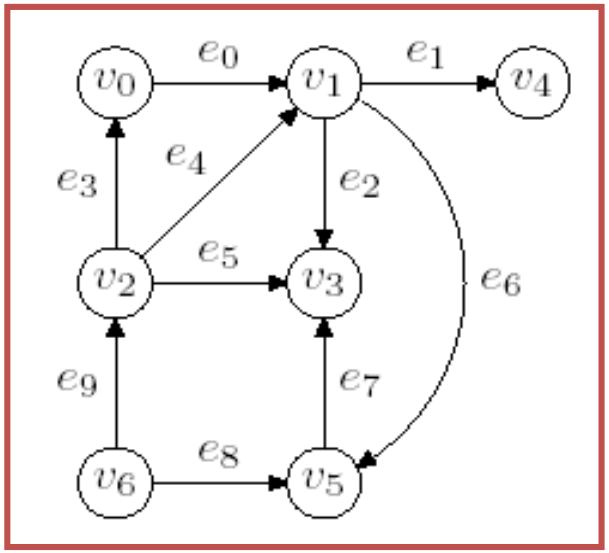
- Za neorijentisani graf matrica susedstva je simetrična u odnosu na glavnu dijagonalu
- Za digrafove to ne mora biti slučaj.
- Prema gornjoj definiciji, ako u čvoru i ne postoji petlja, tada je element $m_{ii} = 0$

Matrica susedstva (nast.)

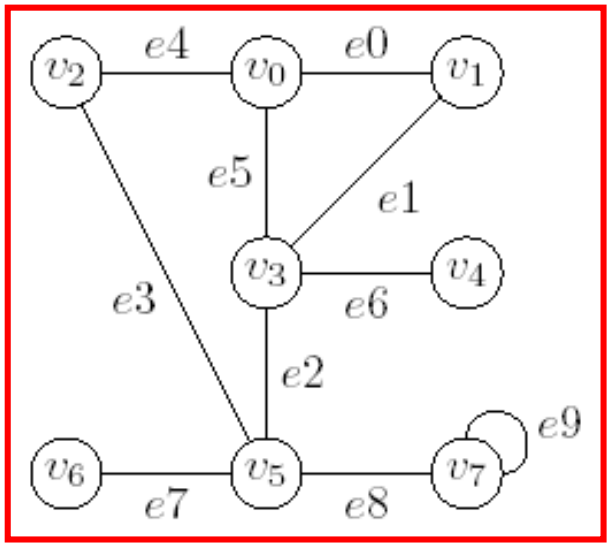
- Generalizacija matrice susedstva:
  - Element matrice m_{ij} je jednak broju grana između dva čvora v_i i v_j
  - A_{ij} = broj grana između v_i i v_j

- Graf sa relativno malo potega je redak (sparse),
- Graf sa mnogo potega je gust (dense)
- Redak graf G=(V,E): $|E|=O(|V|)$
- Gust graf G=(V,E): $|E|=O(|V|^2)$
  - $|V|$ - broj čvorova
  - $|E|$ - broj grana

Primer matrice susedstva



$$\begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{array} \begin{array}{c} \begin{array}{ccccccc} 0 & 1 & 2 & 3 & 4 & 5 & 6 \end{array} \\ \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 \end{pmatrix} \end{array}$$



$$\begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \end{array} \begin{array}{c} \begin{array}{cccccccc} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \end{array} \\ \begin{pmatrix} 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \end{pmatrix} \end{array}$$

Matrica težina

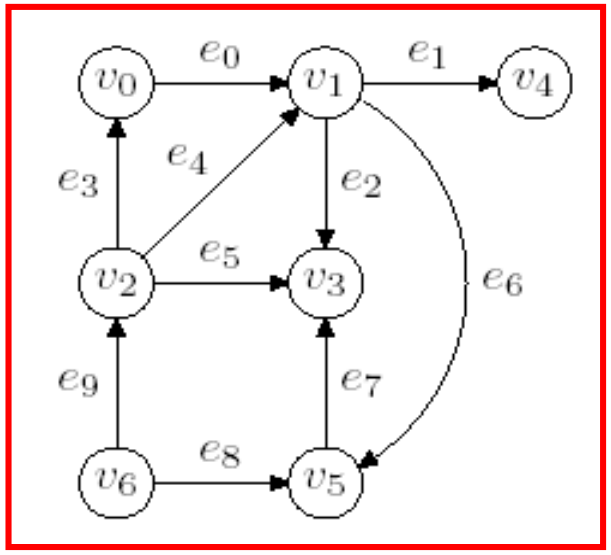
- Težinski graf
  - Ako je svakom potegu dodeljena težina
  - Matrica susedstva -> matrica težina
  - Vrednost elementa matrice težina w_{ij} koji definiše poteg od čvora i do čvora j:
    - umesto 1 sadrži vrednost težine tog potega
    - 0 ili ∞, u zavisnosti od implementacije

$$w_{ij} = \begin{cases} t & (v_i, v_j) \in E \\ \infty & (v_i, v_j) \notin E \end{cases}$$

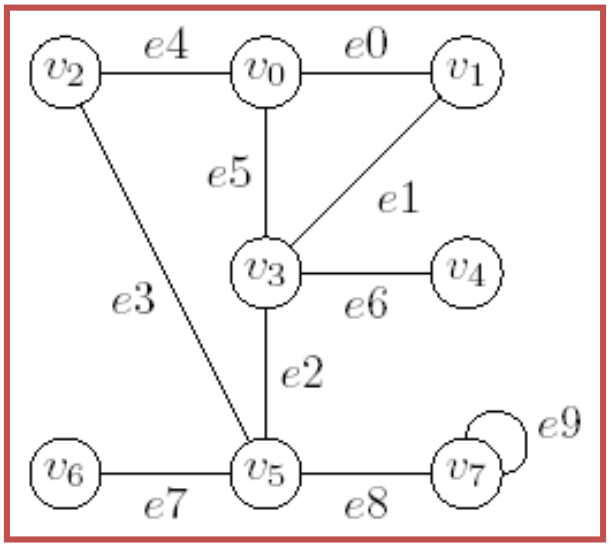
Sekvencijalna reprezentacije - Matrica incidencije

- Matrica incidencije – zasniva se na incidenciji čvorova i potega
- Neka graf G ima n čvorova ($v_0, \ldots, v_{n-1}$) i m potega ($e_0, \ldots, e_{m-1}$)
- Dimenzije matrice su n x m (|V| x |E|)
- Matrica incidencije je matrica čiji je element m_{ij} broj koliko puta su čvor v_i i poteg e_j incidentni.
- Vrednosti elementa m_{ij}:
  - 0 - nisu incidentni,
  - 1 - incidentni
  - [2 - čvor v_i spaja sam sebe (petlja) potegom e_j].
- Ako je graf usmeren, broj može biti +1 (ulazak u čvor) ili -1 (izlazak iz čvora).

Primer matrice incidencije



$$
\begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{array}
\begin{array}{c}
\begin{array}{cccccccccc} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \end{array} \\
\begin{pmatrix}
-1 & 0 & 0 & +1 & 0 & 0 & 0 & 0 & 0 & 0 \\
+1 & -1 & -1 & 0 & +1 & 0 & -1 & 0 & 0 & 0 \\
0 & 0 & 0 & -1 & -1 & -1 & 0 & 0 & 0 & +1 \\
0 & 0 & +1 & 0 & 0 & +1 & 0 & +1 & 0 & 0 \\
0 & +1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & +1 & -1 & +1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & -1
\end{pmatrix}
\end{array}
$$

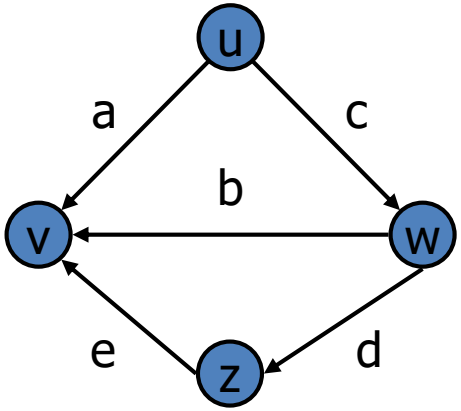


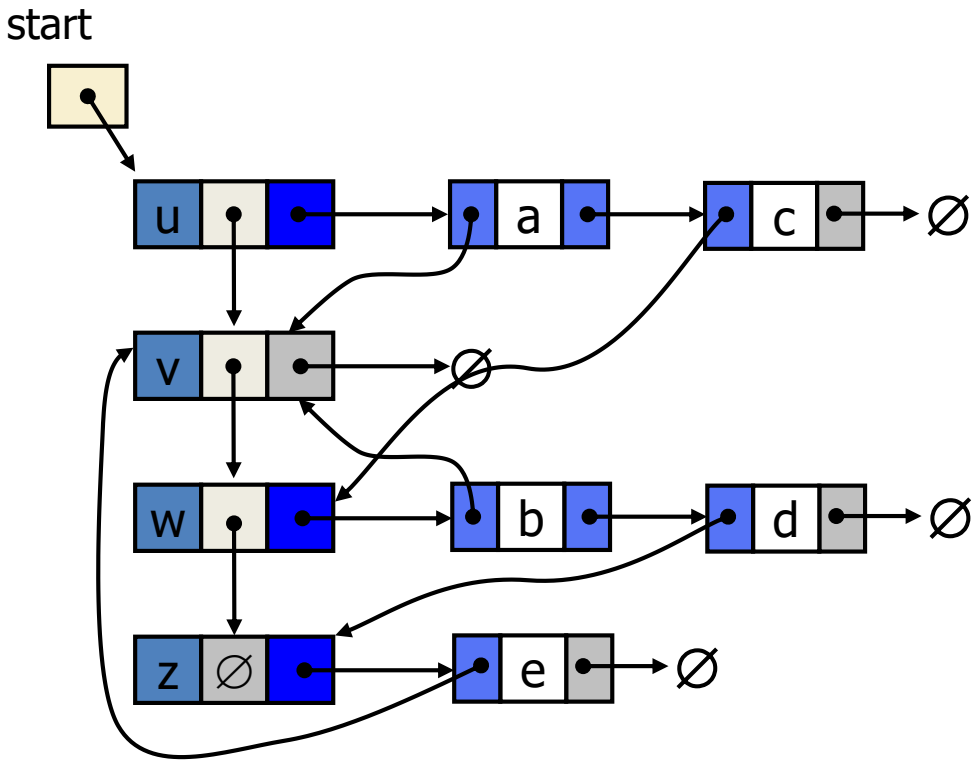
$$
\begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \end{array}
\begin{array}{c}
\begin{array}{cccccccccc} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \end{array} \\
\begin{pmatrix}
1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\
1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 2
\end{pmatrix}
\end{array}
$$

Lančana reprezentacija grafa

- Pogodna za retke grafove
- Lančane liste
  - Jedna lančana lista za čvorove
  - Po jedna lančana lista za potege nekog čvora
- Element za čvor:
  - node – vrenost čvora
  - next – pokazivač na sledeći el. u listi
  - adj – pokazivač na listu potega za taj čvor
- Element za poteg:
  - dest – pointer na odredište potega
  - link – sledeći u listi potega
  - Opciono: vrednost potega (napr weight)

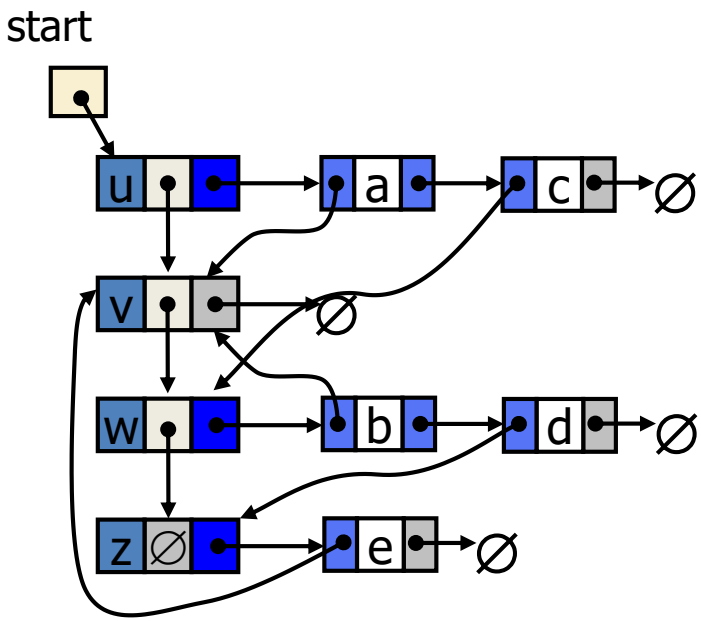
Primer lančane reprezentacije grafa





Graf - Operacije

- Traženje puta
  - proizvoljnog puta
  - najkraćeg puta
- Traženje elemenata grafa
  - traženje potega
  - traženje čvora
- Umetanje
  - potega
  - čvora
- Brisanje
  - potega
  - čvora
- Obilazak grafa
- Topološko sortiranje
- Testiranje povezanosti grafa
- Da li graf ima cikluse
- ...



Graf ADT - Osnovne metode

- Čvorovi i potezi
  - Pozicije
  - Čuvaju elemente
- Accessor metode
  - aVertex()
  - incidentEdges(v)
  - endVertices(e)
  - isDirected(e)
  - origin(e)
  - destination(e)
  - opposite(v, e)
  - areAdjacent(v, w)
- Update metode
  - insertVertex(o)
  - insertEdge(v, w, o)
  - insertDirectedEdge(v, w, o)
  - removeVertex(v)
  - removeEdge(e)
- Generičke metode
  - numVertices()
  - numEdges()
  - vertices()
  - edges()

Pogledati Praktikum !!

Traženje puteva u grafu
sekvencijalna reprezentacija grafa

- Element a_k matrice A^k jednak je broju puteva dužine k od čvora v_i do čvora v_j
- Element $b_r(i,j)$ matrice $B_r = A+A^2+A^3+\ldots+A^r$ jednak je broju puteva dužine $\leq r$ od čvora v_i do čvora v_j
- Matrica puta ili matrica dostupnosti $P=[p_{ij}]$ definiše se kao:

$$p_{ij} = \begin{cases} 1, \text{ ako postoji put iz } v_i \text{ u } v_j \\ 0, \text{ u ostalim slučajevima} \end{cases}$$

TRAŽENJE PUTEVA U GRAFU
SEKVENCIJALNA REPREZENTACIJA GRAFA

Ako je za orijentisani graf sa ***m*** čvorova

- A=[a_{ij}] matrica susedstva i
- P=[p_{ij}] matrica puta,

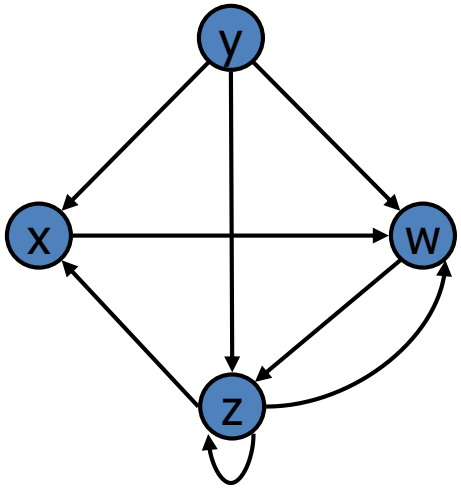
Tada je

- **p_{ij}=1** ako i samo ako matrica $B_m = A+A^2+A^3+\ldots+A^m$ ima **nenulti element b_{ij}**

- Ako graf ima *m* čvorova tada prost put ili ciklus mora biti dužine $\leq m$

- Matrica P strogo povezanog grafa nema nultih elemenata

Primer izračunavanja matrice puta P



$$A = \begin{array}{c|cccc} & x & y & z & w \\ \hline x & 0 & 0 & 0 & 1 \\ y & 1 & 0 & 1 & 1 \\ z & 1 & 0 & 1 & 1 \\ w & 0 & 0 & 1 & 0 \end{array}$$

$$A^2 = \begin{array}{c|cccc} & x & y & z & w \\ \hline x & 0 & 0 & 1 & 0 \\ y & 1 & 0 & 1 & 2 \\ z & 0 & 0 & 1 & 1 \\ w & 1 & 0 & 0 & 1 \end{array}$$

$$A^3 = \begin{array}{c|cccc} & x & y & z & w \\ \hline x & 1 & 0 & 0 & 1 \\ y & 1 & 0 & 2 & 2 \\ z & 1 & 0 & 1 & 1 \\ w & 0 & 0 & 1 & 1 \end{array}$$

$$A^4 = \begin{array}{c|cccc} & x & y & z & w \\ \hline x & 0 & 0 & 1 & 1 \\ y & 2 & 0 & 2 & 3 \\ z & 1 & 0 & 1 & 2 \\ w & 1 & 0 & 1 & 1 \end{array}$$

$$B_4 = A+A^2+A^3+A^4 = \begin{array}{c|cccc} & x & y & z & w \\ \hline x & 1 & 0 & 2 & 3 \\ y & 5 & 0 & 6 & 8 \\ z & 3 & 0 & 3 & 5 \\ w & 2 & 0 & 3 & 3 \end{array} \Rightarrow \mathbf{P} = \begin{array}{c|cccc} & \mathbf{x} & \mathbf{y} & \mathbf{z} & \mathbf{w} \\ \hline \mathbf{x} & \mathbf{1} & \mathbf{0} & \mathbf{1} & \mathbf{1} \\ \mathbf{y} & \mathbf{1} & \mathbf{0} & \mathbf{1} & \mathbf{1} \\ \mathbf{z} & \mathbf{1} & \mathbf{0} & \mathbf{1} & \mathbf{1} \\ \mathbf{w} & \mathbf{1} & \mathbf{0} & \mathbf{1} & \mathbf{1} \end{array}$$

y se ne može doseći!

Algoritam za nalaženje matrice puta P

- Na osnovu definicije matrice P:
  - **Algoritam G.1**. Matrica P za graf sa m čvorova
    1. Naći matricu susedstva A
    2. Naći redom matrice A^2, A^3, ..., A^m
    3. Naći matricu $B = [b_{ij}] = A+A^2+A^3+ ...+A^m$
    4. Generisati matricu P korišćenjem matrice B
       i. $p_{ij}=1$, ako je $b_{ij} \neq 0$
       ii. $p_{ij}=0$, ako je $b_{ij} = 0$
- Bolja varijanta: Warshall-ov algortiam

WARSHALL-OV ALGORITAM

- Definišemo matrice $P_0, P_1, \dots P_m$, tako da je matrica P_k:
  - **$P_k[i,j]=1$,** ako postoji prost put od čvora v_i do čvora v_j koji ne koristi nijedan drugi čvor osim eventualno čvorova $v_1, v_2, \dots, v_k$
  - **$P_k[i,j]=0$,** u ostalim slučajevima.
- Primer:
  - $P_0[i,j]=1$, ako postoji poteg od v_i do v_j
  - $P_1[i,j]=1$, ako postoji poteg od v_i do v_j, koji ne koristi nijedan drugi čvor osim možda v_1
  - $P_2[i,j]=1$, ako postoji poteg od v_i do v_j, koji ne koristi nijedan drugi čvor osim možda v_i i v_j

WARSHALL-OV ALGORITAM (2)

- Može se uočiti da je:
  - **P_0=A,**
    pošto je jedini put od čvora i do čvora j bez prolaska kroz druge čvorove direktan put od i do j
  - **P_m=P,**
    pošto put može da prođe kroz bilo koji čvor obeležen od 1 do m.

WARSHALL-OV ALGORITAM (3)

- Tada važi da je
  $P_k[i,j]=1$,
  ako i samo ako važi jedan od sledeća dva uslova:
  - Ako postoji **prost put** od **v_i do v_j** koji ne koristi nijedan drugi čvor osim možda v_1, v_2, … , v_{k-1}:
    $$P_{k-1}[i,j]=1,$$
  - Ako postoji prost put od **v_i do v_k** i prost put od **v_k do v_j** i ako oba ova puta ne koriste nijedan drugi čvor osim možda v_1, v_2, … , v_{k-1}:
    $$P_{k-1}[i,k]=1 \text{ i } P_{k-1}[k,j]=1$$

Warshall-ov algoritam (4)

Posledica:

- Matrica P_k se može dobiti na osnovu prethodne matrice P_{k-1}
- Element $P_k[i,j]$ se može izračunati na osnovu prethodno ozračunatih vrednosti:

$P_k[i,j] = P_{k-1}[i,j]$ **OR**

$(P_{k-1}[i,k]$ **AND** $P_{k-1}[k,j])$

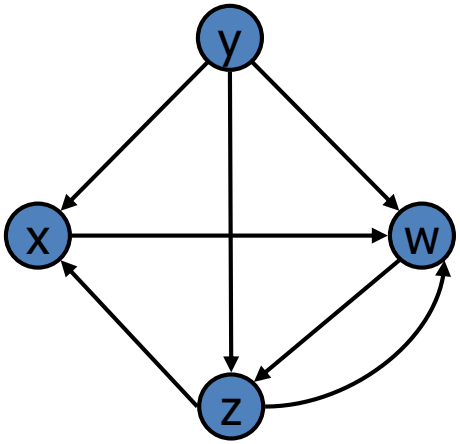
WARSHALL-OV ALGORITAM PSEUDOKOD

Algoritam G.2. Warshall-ov algoritam

Warshall(A,m)

```
1.  {// data je matrica susedstva A i broj čvorova m
     // algoritam generiše matricu P
2.  repeat for (i=1,m)      //inicijalizacija matrice P0
3.      { repeat for (j=1,m)
4.          { if (A[i,j]=0)
5.                  then P[i,j ]=0
6.                  else P[i,j ]=1 }}
7.   repeat for k=1,m      //Azuriranje P
8.      {repeat for i=1,m
9.          {repeat for j=1,m
10.         P[i,j ]=P[i,j] or (P[i,k] and P[k,j]) }}
11. return }
```

WARSHALL-OV ALGORITAM - PRIMER



$P_0 = A =$

	x	y	z	w
x	0	0	0	1
y	1	0	1	1
z	1	0	1	1
w	0	0	1	0

$P_1 =$

	x	y	z	w
x	0	0	0	1
y	1	0	1	1
z	1	0	1	1
w	0	0	1	0

put preko x

$P_2 =$

	x	y	z	w
x	0	0	0	1
y	1	0	1	1
z	1	0	0	1
w	0	0	1	0

put preko x,y

$P_3 =$

	x	y	z	w
x	0	0	0	1
y	1	0	0	1
z	1	0	1	1
w	1	0	1	1

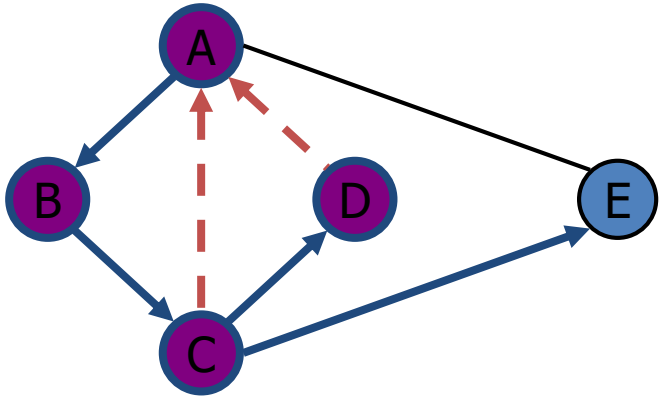
put preko x,y,z

$P = P_4 =$

	x	y	z	w
x	1	0	1	1
y	1	0	1	1
z	1	0	1	1
w	1	0	1	1

putevi preko x,y,z,w

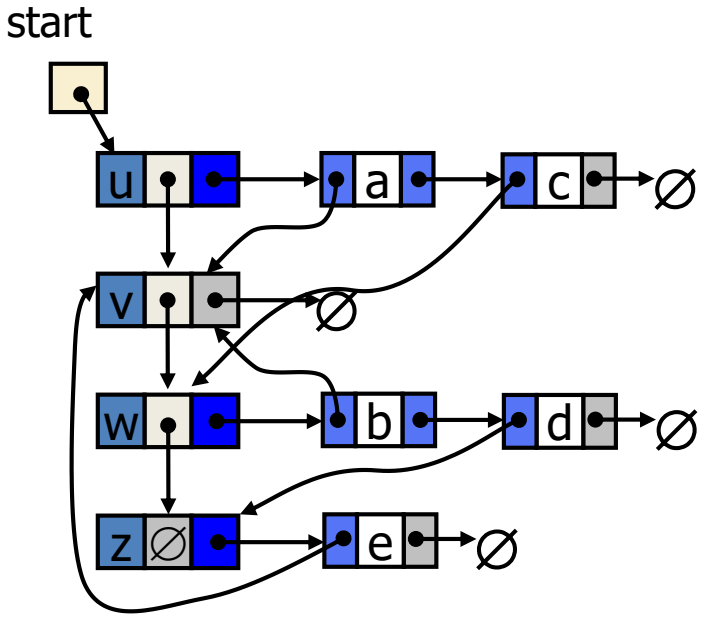
Ostale osnovne operacije (lančana reprezentacija)



Traženje čvora i potega
Dodavanje čvora i potega
Brisanje čvora i potega
Depth-First Search

Traženje čvora i Traženje potega

- Lančana reprezentacija grafa
- **Traženje čvora**
  - Zadata vrednost čvora koji se traži
  - Procedura vraća lokaciju čvora
- **Traženje potega**
  - Zadati završni čvorovi potega A i B
  - Rezultat je lokacija čvora B u listi grana čvora A



TRAŽENJE U GRAFU – PSEUDOKOD

Algoritam G.6. Traženje čvora
findNode(start, A, loc)

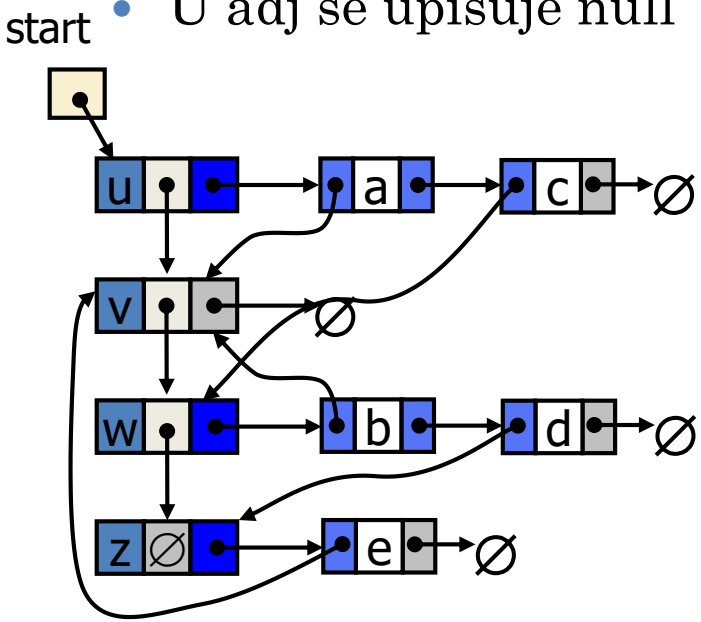
```
1.  {
2.    pok = start
3.    repeat while (pok <> null)
4.    {    if (pok.info=A)
5.         then      {
6.           loc=pok
7.           return  }
8.         else pok=pok.link
9.    }
10.    loc= null
11.    return
12.}
```

Algoritam G.7. Traženje potega
findEdge(start, A, B, loc)

```
1.  { call findNode(start, A, locA)
2.     call findNode(start, B, locB)
3.     if (locA=null or locB=null)
4.     then loc=null
5.     else {
6.          pok = locA.adj
7.          repeat while (pok <> null)
8.          {   if (pok.dest=locB)
9.               then          {
10.                   loc=pok
11.                   return     }
12.              else pok=pok.link
13.         }
14.     loc= null
15.     return
16. }
```

Operacije umetanja – dodavanje čvora ili grane

- Dodavanje čvora
  - Čvor se dodaje na početak liste čvorova
  - Odogovara operaciji dodavanja elementa na početak liste
  - U adj se upisuje null

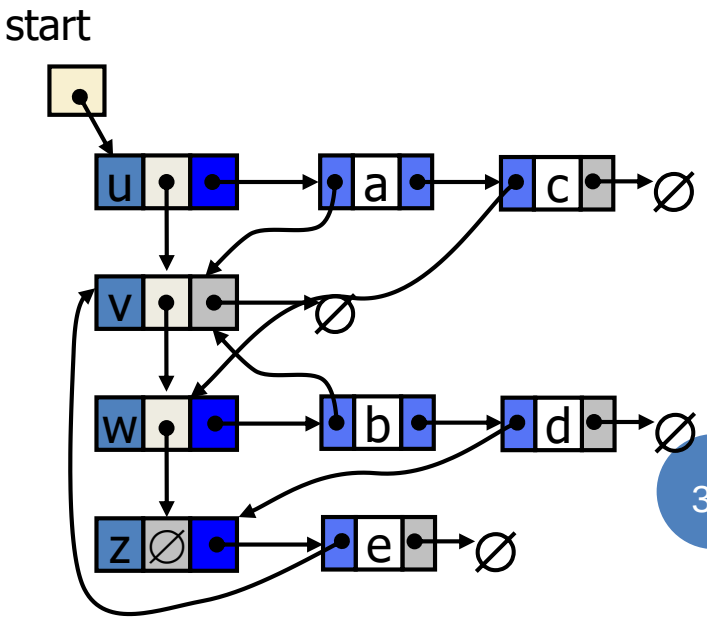


- Dodavanje potega
  - Podrazmeva se da čvorovi A i B postoje
  - Zadati su svojom vrednošću, pa se najpre pozivom *findNode* određuju lokacije u listi čvorova
  - Novi poteg se dodaje kao prvi element u listu potega prvog čvora
  - U *dest* se upisuje lokacija drugog čvora

Brisanje potega

- Brišemo poteg između dva zadata čvora A i B
- Nalazimo lokacije oba čvora
- U listi potega prvog čvora brišemo element koji ukazuje na drugi čvor
- Ovaj deo odgovara operaciji brisanja zadatog elementa liste

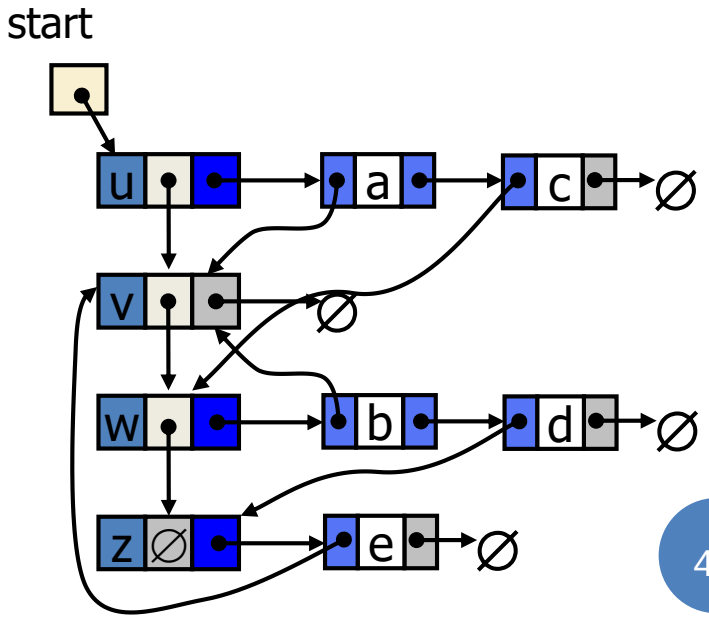
- locA=findNode(A)
- locB=findNode(B)
- Brisanje
  - Iz liste locA.adj
  - Element locB



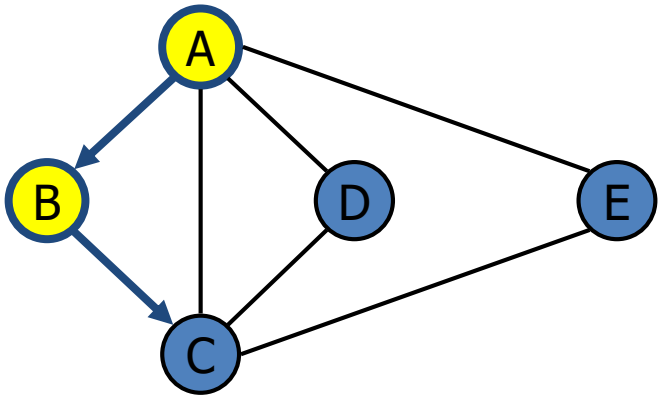
Brisanje čvora

- Nalazimo lokaciju čvora N
- Obrisati sve potege koji se završavaju na čvoru N
  - Zahteva obilazak celog grafa, tj liste čvorova, i za svaki čvor obilazak njegove liste potega
  - Obrisati poteg prema N iz listi potega svih čvorova
- Obrisati listu potega čvora N
  - Odogovara brisanju cele liste
- Obrisati čvor N iz liste čvorova
  - Ovaj deo odgovara operaciji brisanja zadatog elementa liste

- Neophodne operacije
  - Obilazak liste čvorova
  - Obilazak liste potega
  - Brisanje zadatog elementa iz liste potega
  - Brisanje cele liste potega
  - Brisanje zadatog elementa iz liste čvorova



Obilazak Grafa



Obilazak grafa

- Sistematski se ispituju svi čvorovi i grane grafa
- Svaki čvor se obilazi samo jednom
- Obilazak po širini – BFS
  - **Red** kao pomoćna struktura
- Obilazak po dubini – DFS
  - **Magacin** kao pomoćna struktura
- Status čvorova
  - 1 (spreman): inicijalno stanje
  - 2 (čekanje): čvor čeka na obradu
  - 3 (obrađen): čvor je obrađen
- Ako neki od čvorova nisu obiđeni, ponoviti postupak počev od prvog čvora kome je status ostao 1.

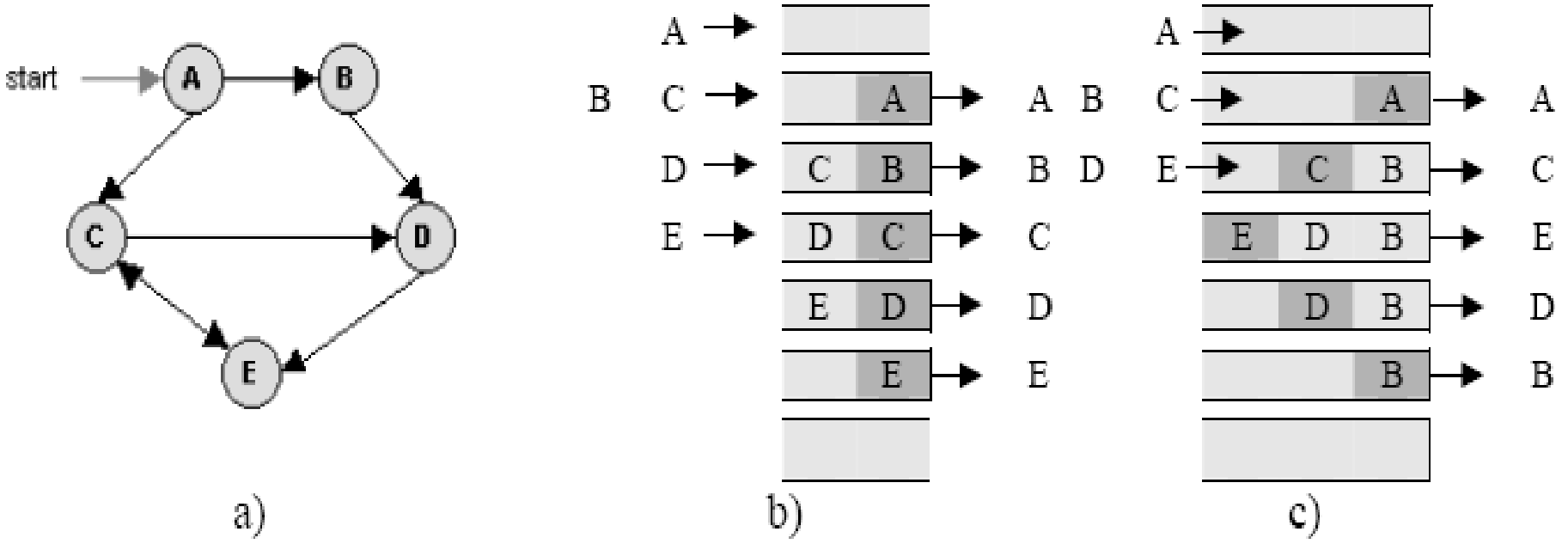
OBILAZAK PO ŠIRINI/DUBINI

BFS/DFS – razlika je u pomoćnoj strukturi !!

Algoritam G.8 Obilazak po **širini** / **dubini**

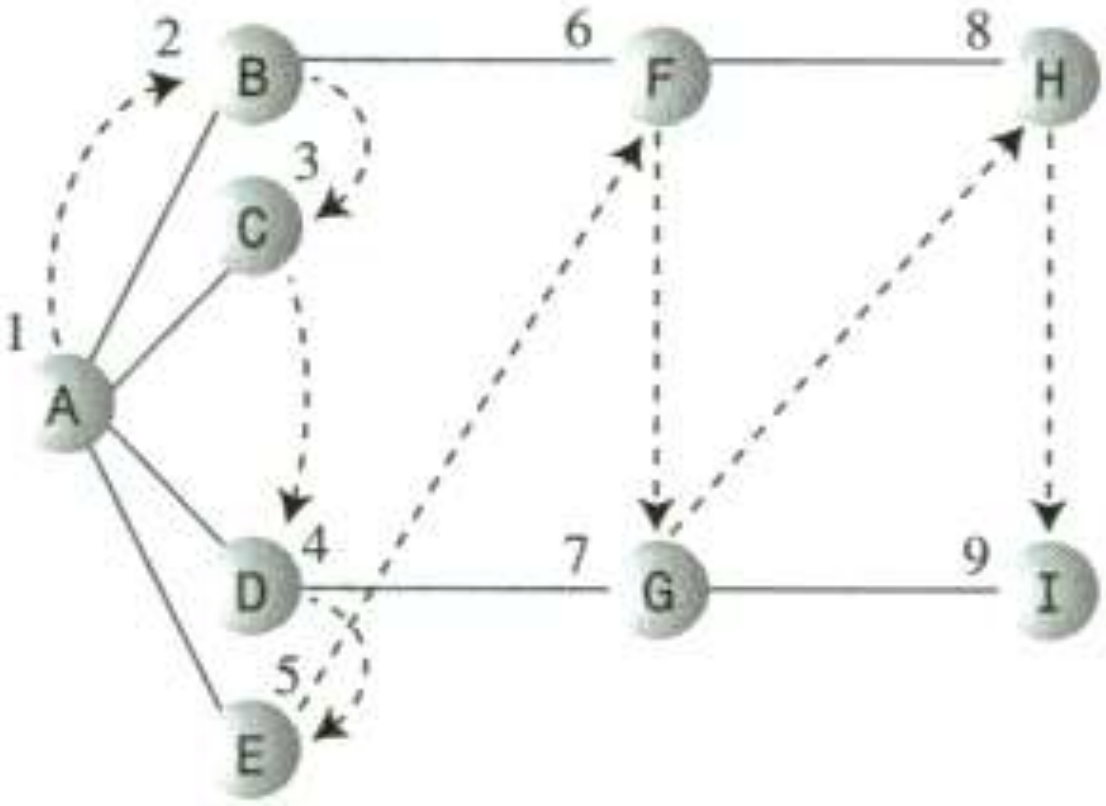
1. Postaviti sve čvorove u STATUS=1
2. Upisati prvi čvor u **RED** / **MAGACIN** i promeniti mu status na STATUS=2
3. Sve dok **RED** / **MAGACIN** ne bude prazan
   a) Uzeti čvor sa početka **REDa** / **MAGACINa**.
      Obraditi N u promeniti mu STATUS=3
   b) Dodati u **RED** / **MAGACIN** sve susede čvora N čiji je STATUS=1.
      Promeniti im STATUS=2
4. Kraj.

ILUSTRACIJA RADA DFS/BFS



Obilazak grafa: a) primer grafa, b) obilazak po širini, c) obilazak po dubini

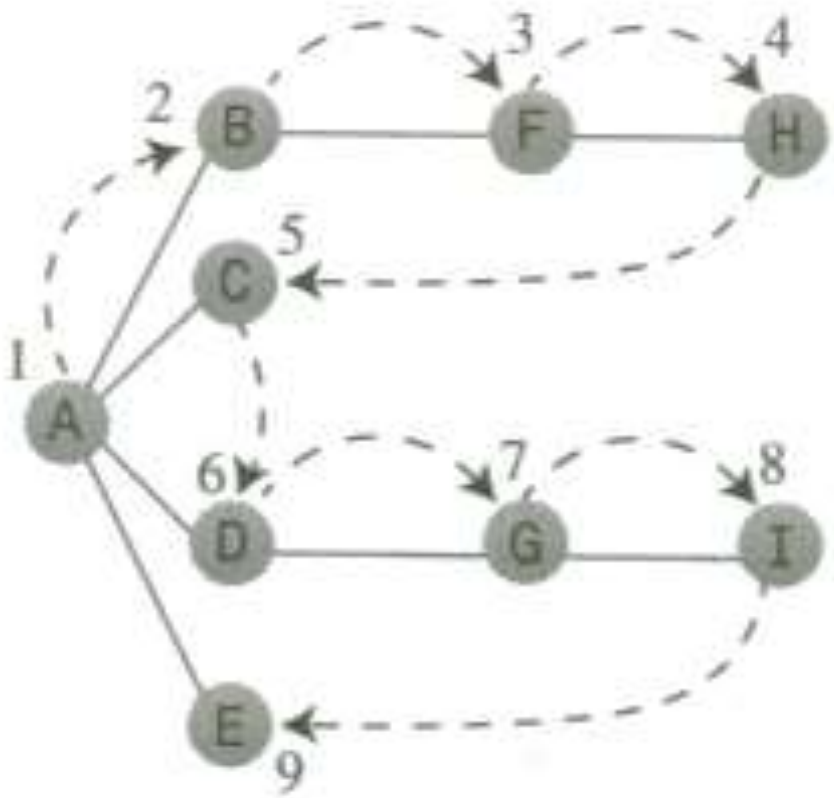
Redosled obilaska po BFS



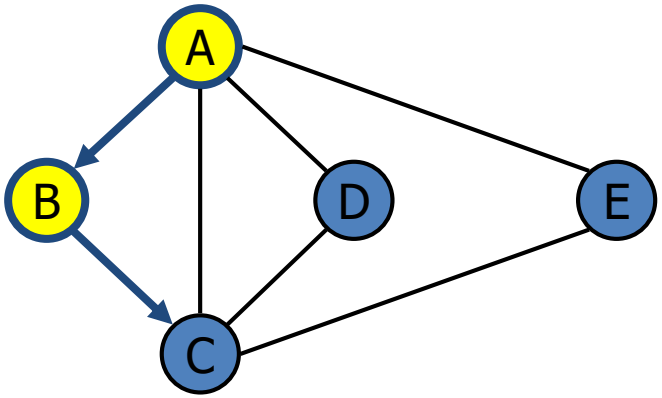
DFS

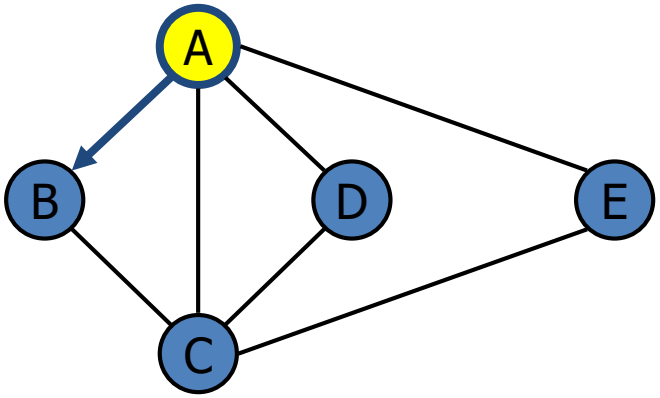
- DFS je generalna tehnika za obilazak grafa
- DFS obilazak
  - Obiđi sve čvorove i potege grafa G
  - Određuje da li je graf povezan
  - Određuje povezane komponente grafa G
  - Određuje *spanning forest* grafa G
- DFS za graf sa n čvorova i m potega zahteva $O(n + m)$
- DFS se može proširiti da reši neke probleme kod grafa
  - Naći i prikazati put između dva zadata potega
  - Pronaći cikluse u grafu

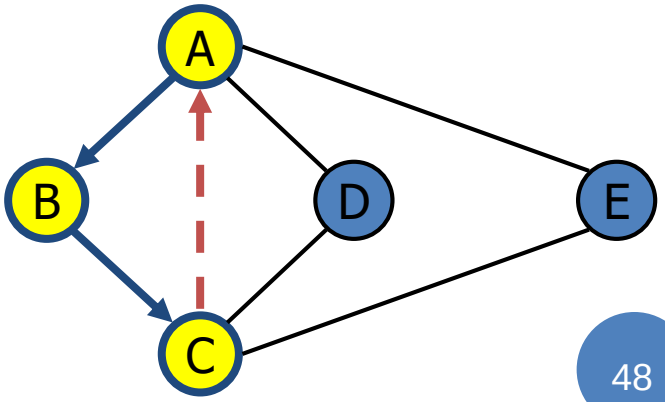
Redosled obilaska po DFS



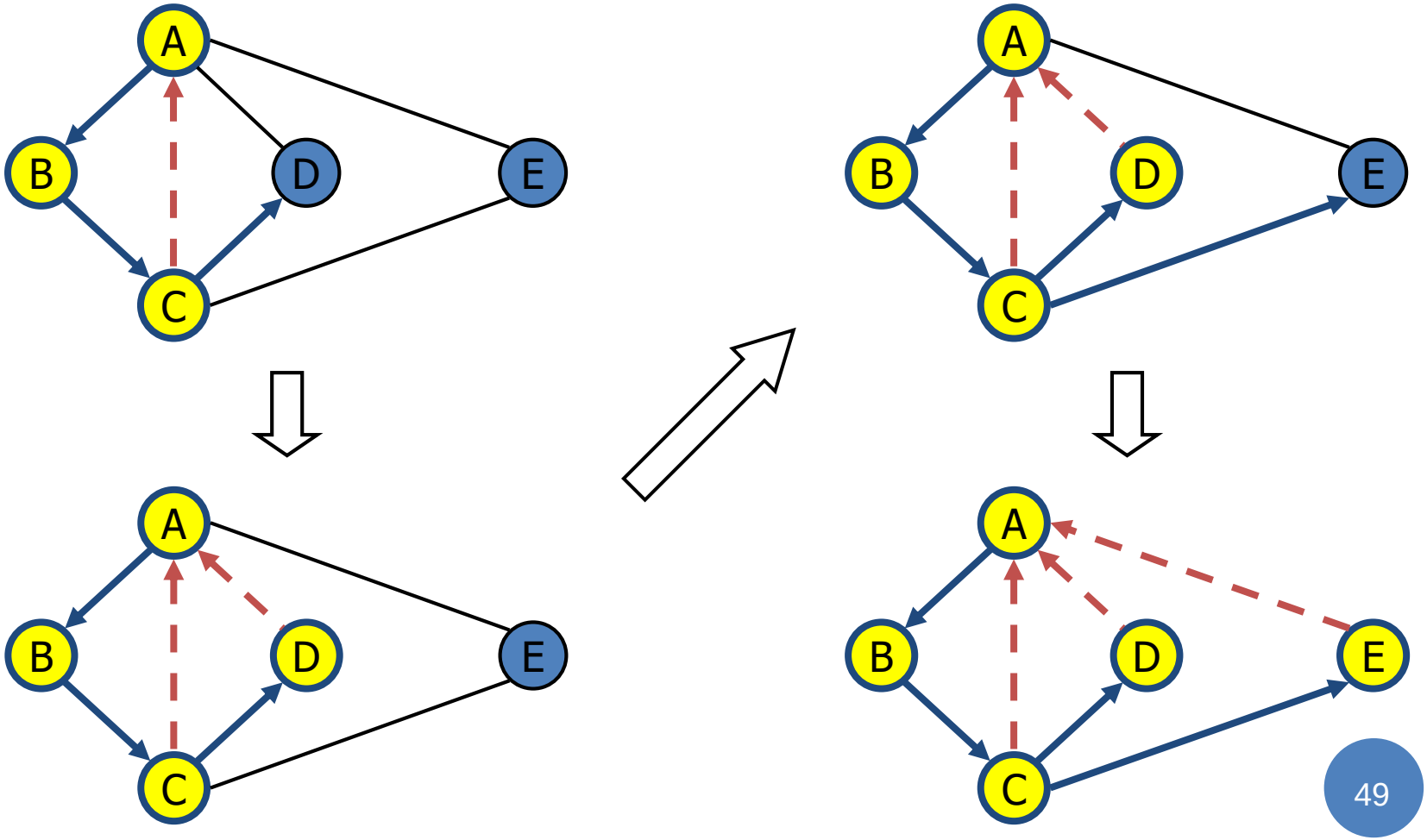
Primer za DFS



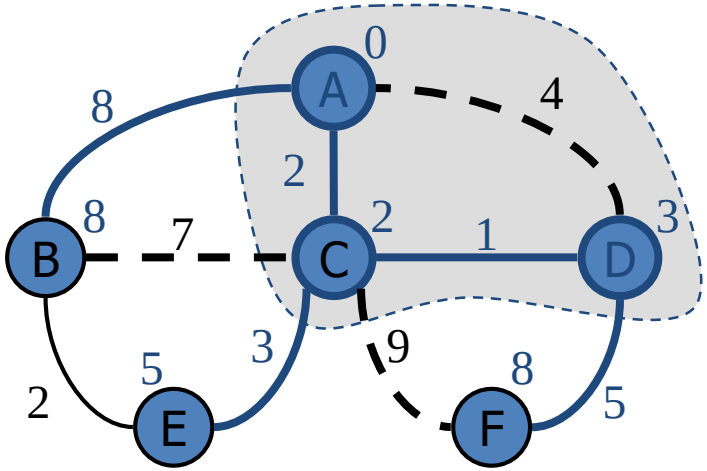




Primer za DFS (nast.)



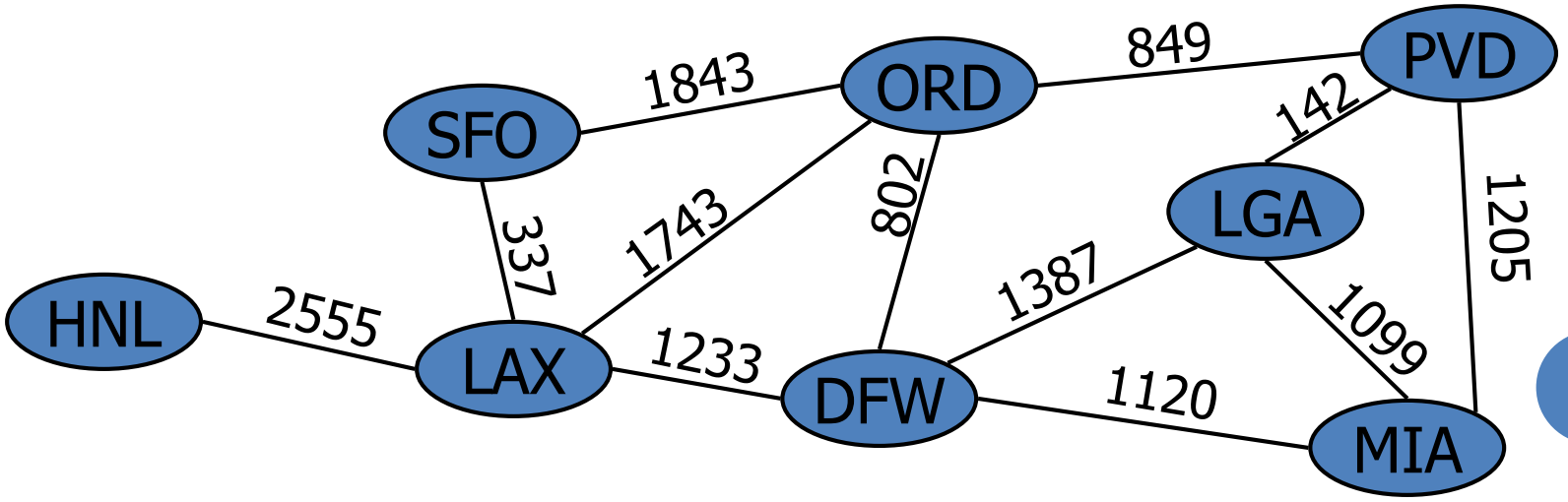
Najkraći put u grafu



Težinski graf

- **Matrica težina**: umesto 1 i 0, vrednosti su težine potega 0 ili ∞ (definisano kod implementacije)

$$W_{ij} = \begin{cases} t & (v_i, v_j) \in E \\ \infty & (v_i, v_j) \notin E \end{cases}$$



Problem najkraćeg puta

- Ako je dat težinski graf, i dva čvora ***u*** i ***v***, treba **naći put sa minimalnom ukupnom težinom** između ***u*** i ***v.***
  - Dužina puta je **zbir težina potega** koji su deo puta.
- Primer:
  - Najkraći put između aerodroma u Providence PVD i Honolulu HNL
- Primena
  - Rutiranje Internet paketa
  - Rezervacije letova
  - Uputstva za vožnju, i sl.

Najkraći put - osobine

Osobina 1:

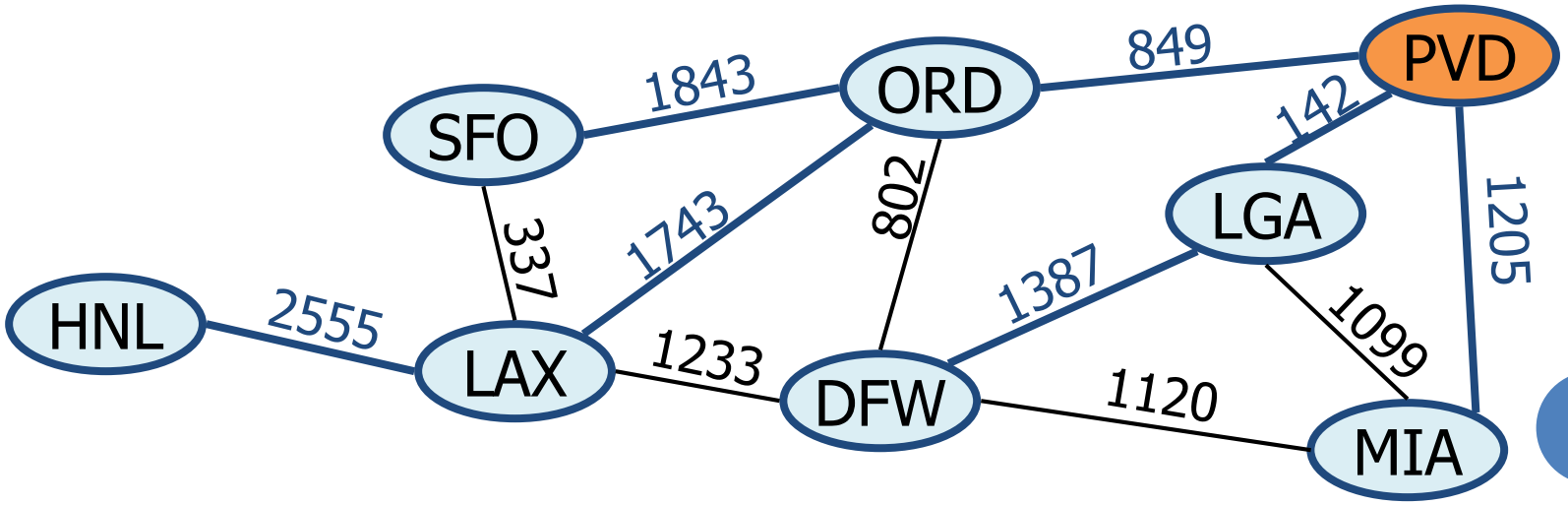
Put koji je deo najkraćeg puta je takođe najkraći put

Osobina 2:

Počev od startnog čvora, postoji **stablo najkraćih puteva** do svih ostalih čvorova

Primer:

Stablo najkraćih puteva od aerodroma u Providence-u PVD



Traženje najkraćeg puta u grafu – sekvencijalna reprezentacija

- Polazimo od matrice težina W
- Matrica P pokazuje da postoji put između dva čvora
- **Matrica** Q, čiji elementi sadrže dužinu najkraćeg puta
- Modifikovani Warshall-ov algoritam - za nalaženje matrice Q
- Definišemo matrice Q_0, Q_1, ... Q_m, tako da je matrica Q_k:

$$Q_k[i,j]=\min(Q_{k-1}[i,j], (Q_{k-1}[i,k] + Q_{k-1}[k,j])$$

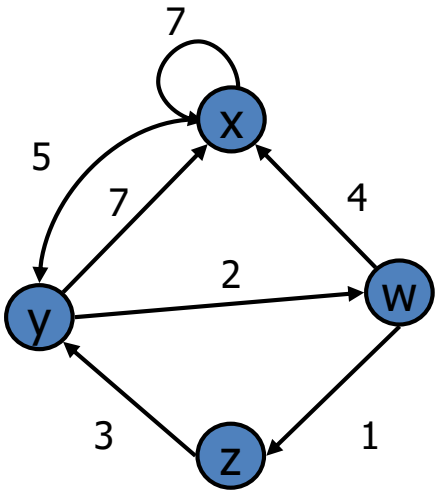
- Može se uočiti da je:
  - $Q_0=W$
  - $Q_m=Q$
- Ako je graf zadat matricom A, podrazumeva se da su težine za svaki poteg = 1, i primenjuje se isti algoritam

Modifikovani Warshall-ov algoritam – pseudokod

Algoritam G.3. Modifikovani Warshall-ov algoritam
ModifWarshall(W,m)

```
1.  {// data je matrica težina W i broj čvorova m
     // algoritam generiše matricu Q
2.  repeat for (i=1,m)        //inicijalizacija matrice Q0
3.      { repeat for (j=1,m)
4.          { if (W[i,j]=0)
5.                  then Q[i,j]=MAX
6.                  else Q[i,j]=W[i,j] }}
7.   repeat for k=1,m       //Azuriranje Q
8.      {repeat for i=1,m
9.         {repeat for j=1,m
10.         Q[i,j]=min(Q[i,j], (Q[i,k] + Q[k,j]) }}
11. return}
```

Modifikovani Warshall-ov algoritam - primer



$Q_0 = W =$

	x	y	z	w
x	7	5	∞	∞
y	7	∞	∞	2
z	∞	3	∞	∞
w	4	∞	1	∞

+ min

$Q_1 =$

	x	y	z	w
x	7	5	∞	∞
y	7	12	∞	2
z	∞	3	∞	∞
w	4	9	1	∞

$Q_2 =$

	x	y	z	w
x	7	5	∞	7
y	7	12	∞	2
z	10	3	∞	5
w	4	9	1	11

$Q_3 =$

	x	y	z	w
x	7	5	∞	7
y	7	12	∞	2
z	10	3	∞	5
w	4	4	1	6

$Q = Q_4 =$

	x	y	z	w
x	7	5	8	7
y	7	11	3	2
z	9	3	6	5
w	4	4	1	6

Najkraći put u grafu: Dijkstra-in algoritam – lančana reprezentacija grafa

- Rastojanje čvora ***v*** od čvora ***s*** je dužina najkraćeg puta između ***s*** i ***v***
- Dijkstra-in algoritam računa rastojanja za sve čvorove počev od startnog čvora ***s***
- Pretpostavke:
  - Graf je povezan
  - Težine potega su ne-negativne

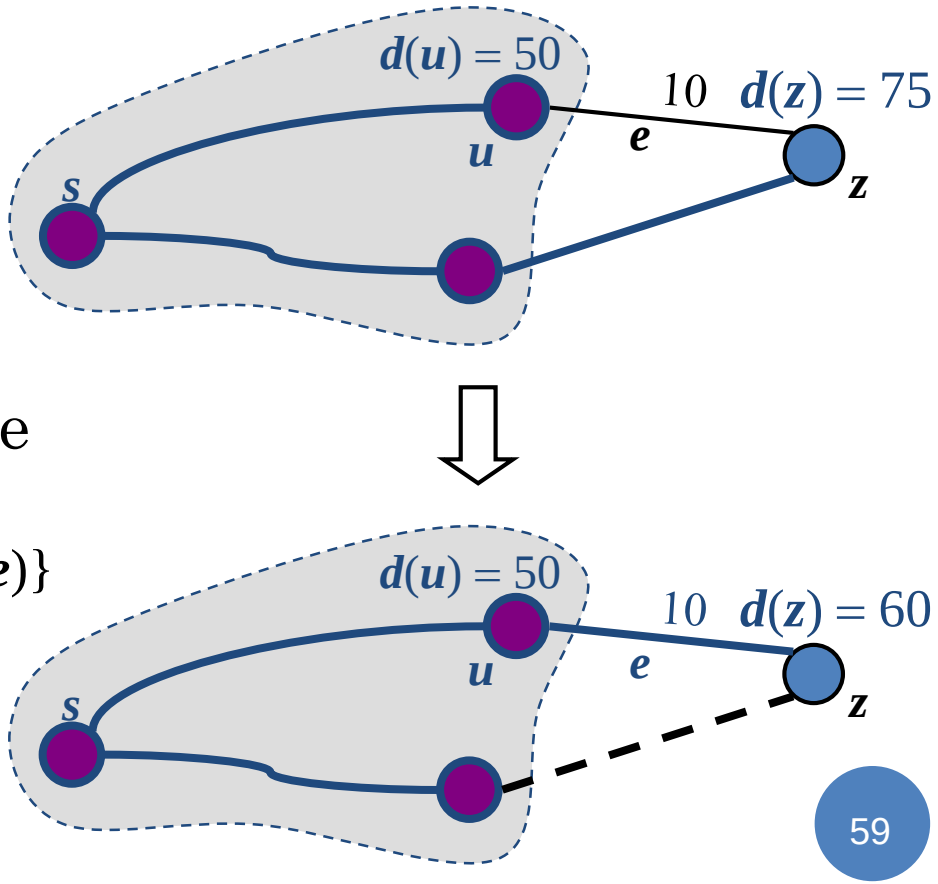
Najkraći put u grafu: Dijkstra-in algoritam – lančana reprezentacija grafa

- Algoritam počinje od startnog čvora obradom svih njegovih potega, tako što se formira "oblak"
- Za svaki čvor ***v*** pamti se vrednost/labela ***d*(*v*)** koja predstavlja rastojanje ***v*** od ***s*** u pod-grafu koji se sastoji od "oblaka" i povezanih čvorova
- U svakom koraku
  - Dodajemo"oblaku" čvor ***u,*** koji je izvan oblaka, i sa najmanjom vrednosti distance ***d*(*u*)**
  - Ažuriramo vrednosti distanci za sve čvorove susedne sa ***u***

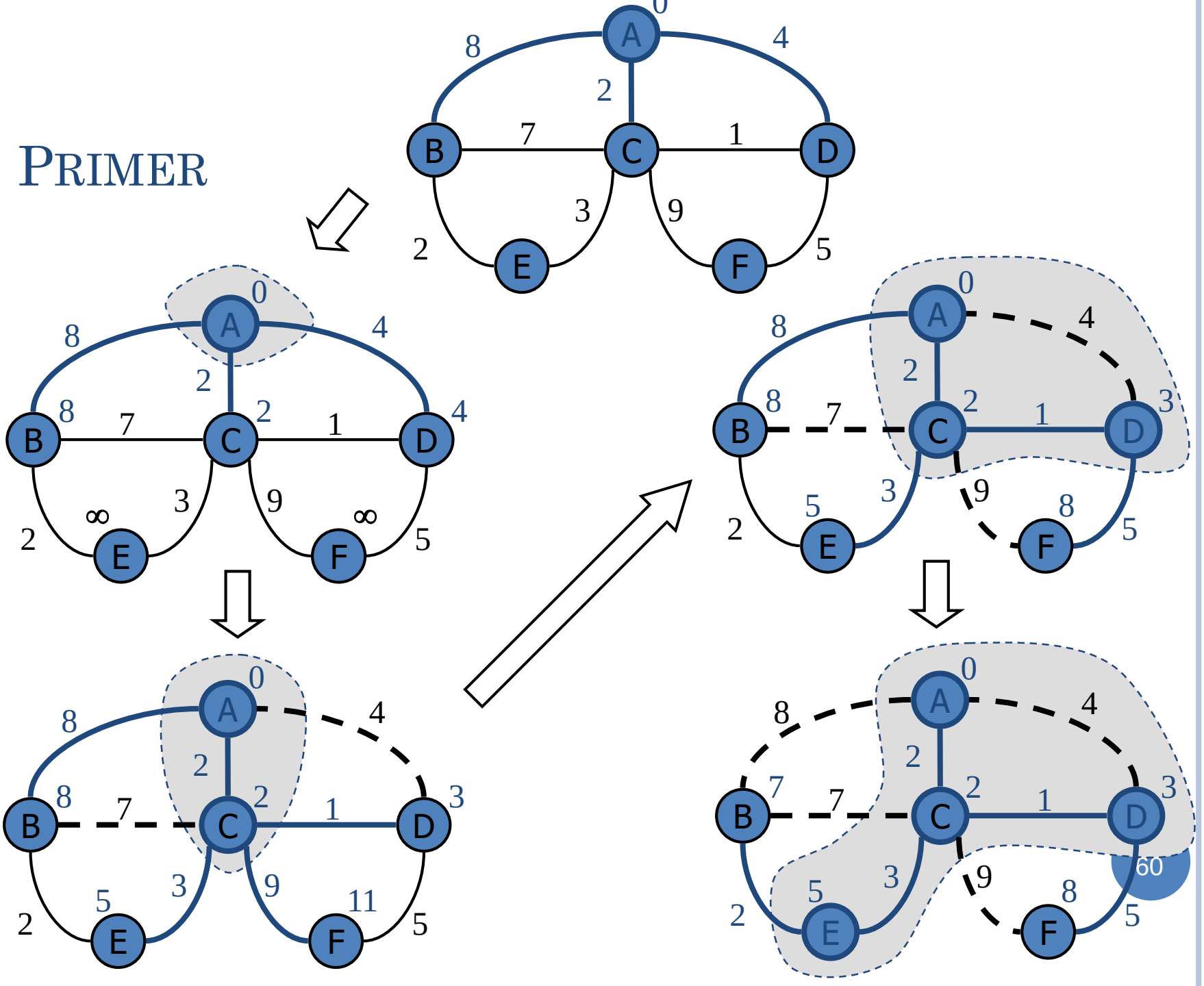
Relaksacija potega

- Posmatramo poteg ***e*** **=** **(*u*,*z*)** takav da je
  - ***u*** čvor koji je skoro dodat oblaku
  - ***z*** nije u oblaku
- Relaksacija potega ***e*** podrazumeva ažuriranje ***d*(*z*)** :

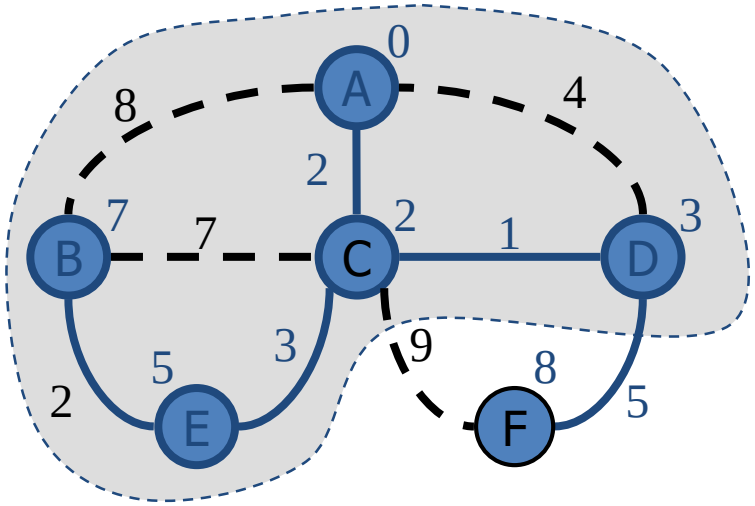
$$d(z) \leftarrow \min\{d(z), d(u) + weight(e)\}$$

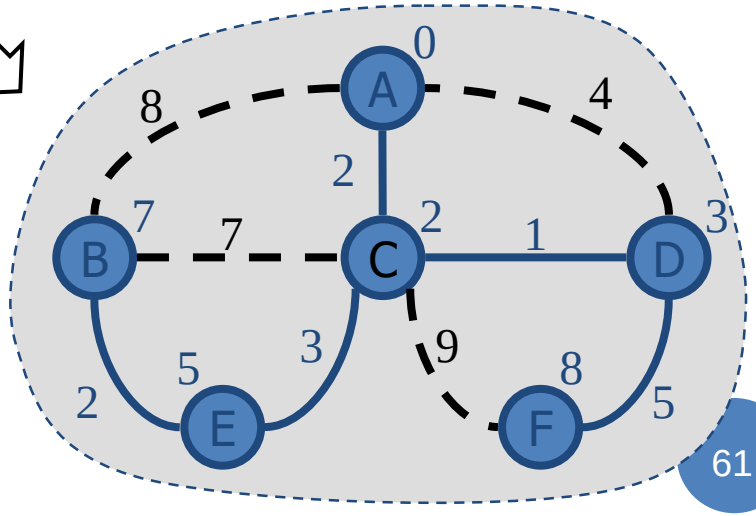


Primer



Primer (nast.)





Dijkstra-in Algoritam

- Red sa proritetom čuva čvorove koji su van oblaka
  - Ključ *Key*: distanca
  - Element: čvor
- Locator-based metoda
  - *insert(k,e)* vraća locator
  - *replaceKey(l,k)* menja vrednost ključa zadatog elementa
- Čuvamo dve labele za svaki čvor:
  - distanca (labela d(v))
  - lokator u redu sa prioritetom

Algoritam G.4. *DijkstraDistances(G, s)*

```
Q ← new heap-based priority queue
for all v ∈ G.vertices()
    if v = s
        setDistance(v, 0)
    else
        setDistance(v, ∞)
    l ← Q.insert(getDistance(v), v)
    setLocator(v,l)
while ¬Q.isEmpty()
    u ← Q.removeMin()
    for all e ∈ G.incidentEdges(u)
        { relax edge e }
        z ← G.opposite(u,e)
        r ← getDistance(u) + weight(e)
        if r < getDistance(z)
            setDistance(z,r)
            Q.replaceKey(getLocator(z),r)
```

Proširenje Dijkstra-inog algoritma – stablo najkraćih puteva

- Vraća stablo najkraćih puteva od startnog čvora do svih ostalih
- Pamtimo uz svaki čvor treću labelu:
  - Roditeljski poteg u stablu najkraćeg puta
- U procesu relaksacije potega ažuriramo i ovu labelu

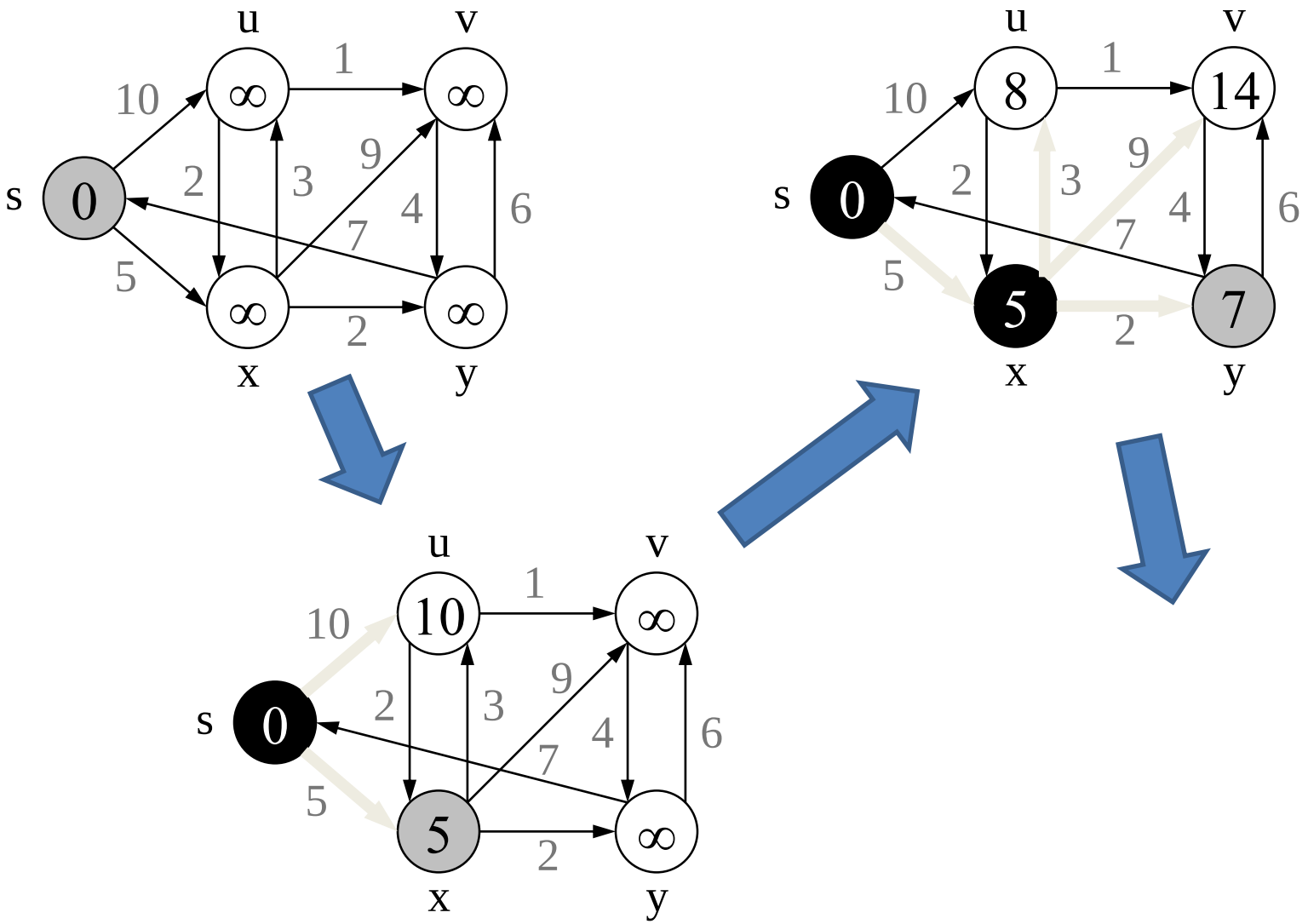
Algoritam G.5.
DijkstraShortestPathsTree(G, s)

```
…

for all v ∈ G.vertices()
  …
  setParent(v, ∅)
  …

  for all e ∈ G.incidentEdges(u)
    { relax edge e }
    z ← G.opposite(u,e)
    r ← getDistance(u) + weight(e)
    if r < getDistance(z)
      setDistance(z,r)
      setParent(z,e)
      Q.replaceKey(getLocator(z),r)
```

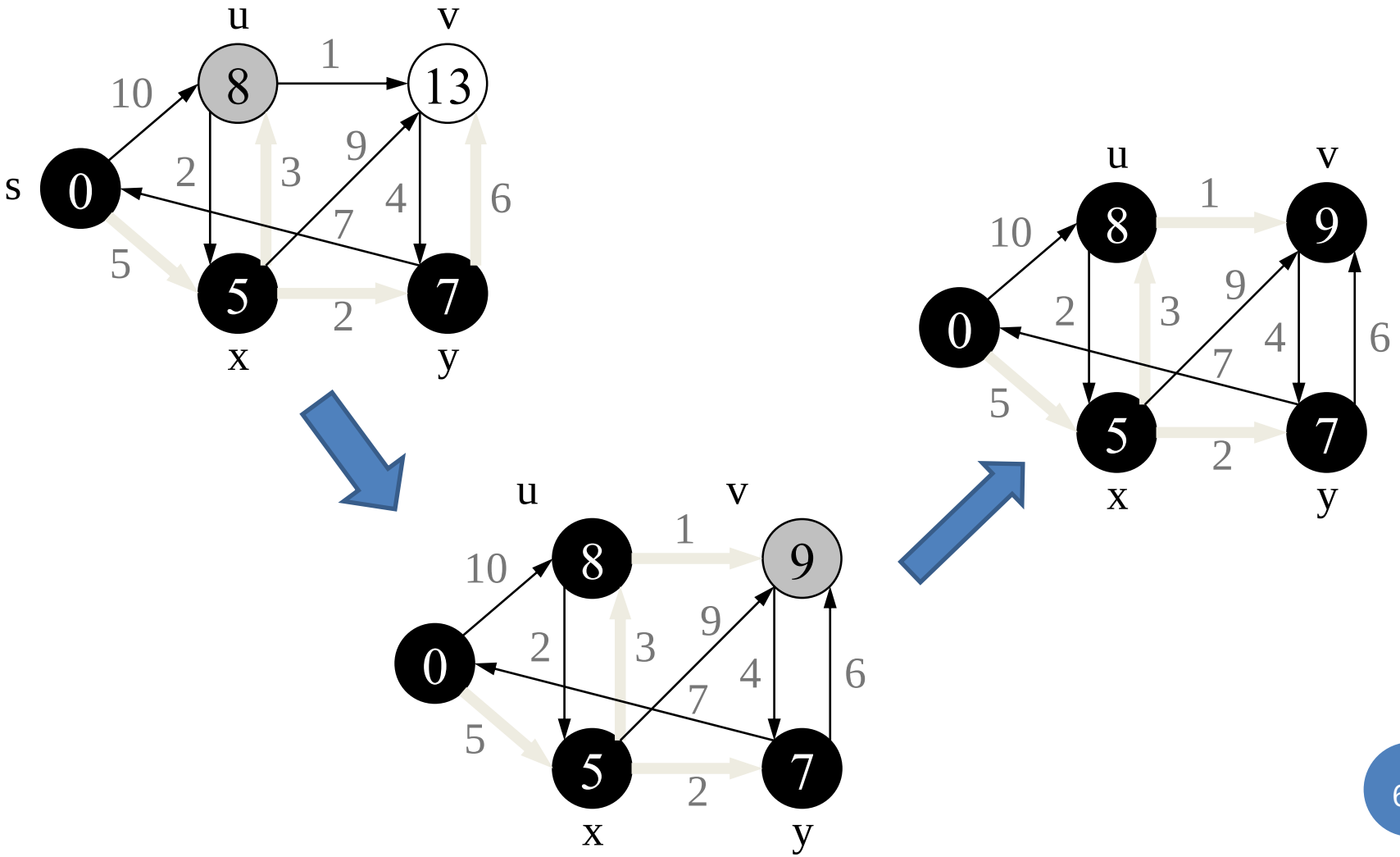
Dijkstra Algoritam - pregled

- Ne-negativne težine potega
- *Greedy* algoritam
- Odgovara BFS (breadth-first search) algoritmu (ako su sve težine = 1, može se jednostavno koristiti BFS)
- Koristi ADT *Q*, red sa prioritetom, prioritet/ključ je vrednost labele u čvoru (BFS koristi FIFO red)
- Osnovna ideja
  - „Oblak“ – skup S obrađenih čvorova/potega
  - U svakom koraku bira „najbliži" čvor *u*, dodaje ga *S*, i relaksira sve potege iz *u*

Dijkstra – primer 2



Dijkstra – primer 2 (nast.)

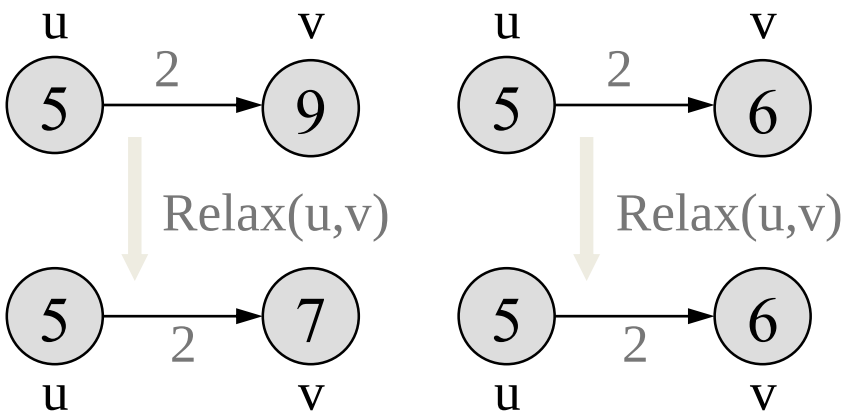


Dodatne informacije: Bellman-Ford Algoritam

- Dijkstra ne radi sa negativnim potezima:
  - Intuicija – ne možemo biti pohlepni (greedy) uz pretpostavku da će s dužine puta povećavati u narednim koracima obrade
- Bellman-Ford algoritam detektuje negativne cikluse (vraća *false*) ili vraća stablo najkraćih puteva

Relaksacija potega

- Za svali čvor v u grafu, čuvamo v.**d**(), procenjeni najkraći put, inicijalizovan na ∞ na početku
- Relaksacija potega (u,v) znači proveru da li možda možemo da nađemo kraći put do v preko čvora u

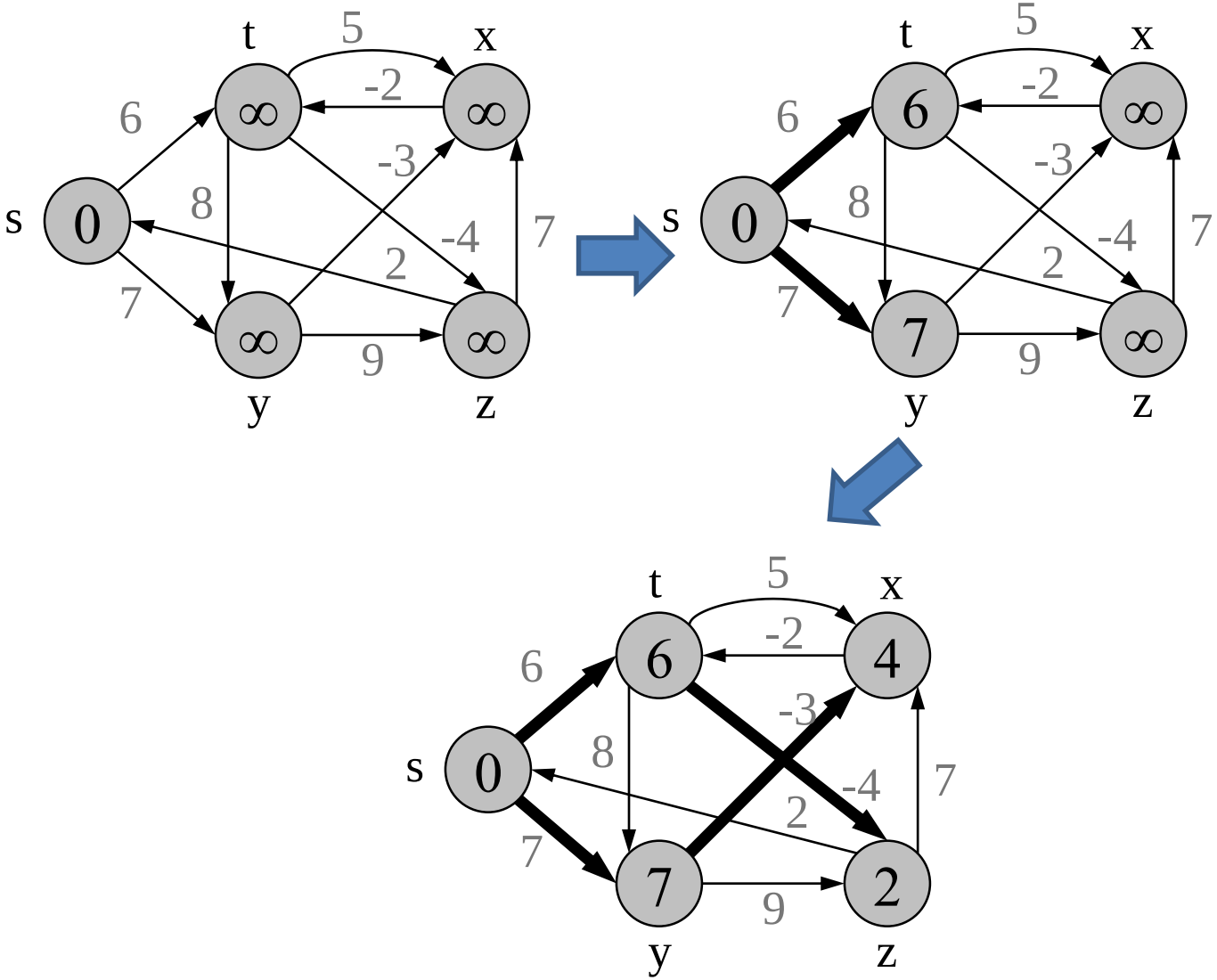



```
Relax (u,v,G)
if v.d() > u.d()+G.w(u,v) then
   v.setd(u.d()+G.w(u,v))
   v.setparent(u)
```

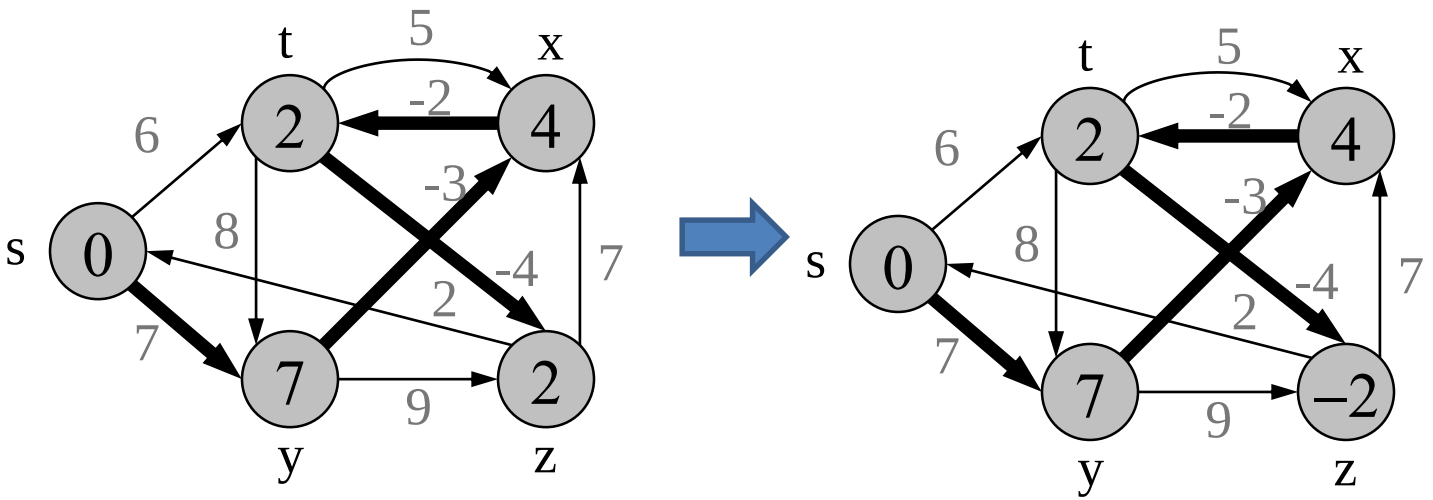
Bellman-Ford Algoritam

```
Bellman-Ford(G,s)
01 for each vertex u ∈ G.V()
02    u.setd(∞)
03    u.setparent(NIL)
04 s.setd(0)
05 for i ← 1 to |G.V()|-1 do
06     for each edge (u,v) ∈ G.E() do
07         Relax (u,v,G)
08 for each edge (u,v) ∈ G.E() do
09      if v.d() > u.d() + G.w(u,v) then
10         return false
11 return true
```

Bellman-Ford – Primer



Bellman-Ford Example



Pitanja, ideje, komentari