STRUKTURE PODATAKA
LETNJI SEMESTAR 2014/2015

GRAFOVI

OBILAZAK GRAFA
NAJKRAĆI PUT U GRAFU

Prof. Dr Leonid Stoimenov

Katedra za računarstvo
Elektronski fakultet u Nišu

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.

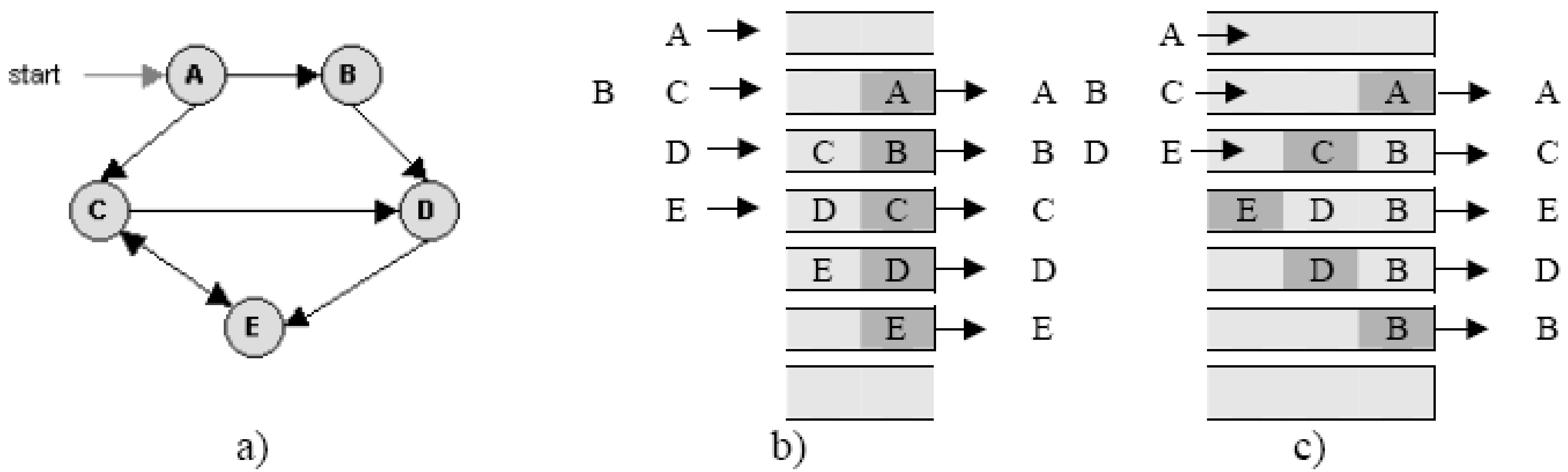
OBILAZAK PO ŠIRINI/DUBINI

BFS/DFS – razlika je u pomoćnoj strukturi !!

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

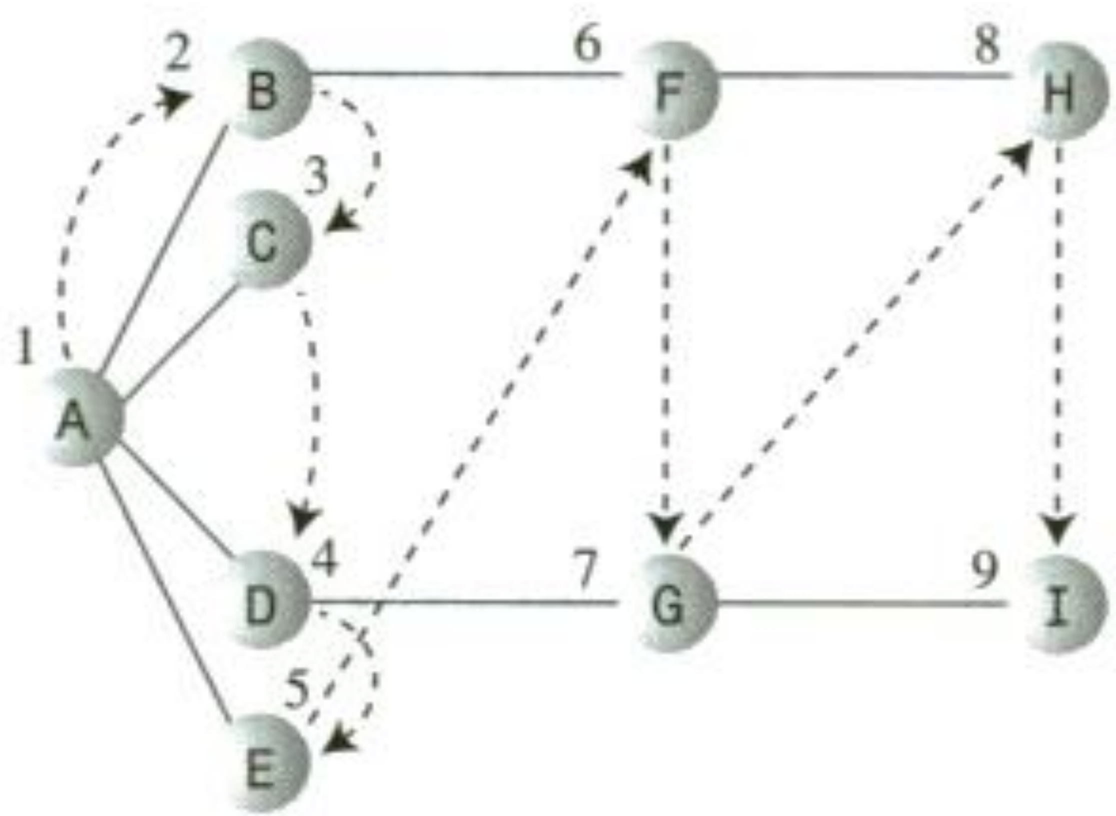
1. Postaviti sve čvorove u STATUS=1
2. Upisati prvi čvor u **RED** / **MAGACIN** i promeniti mu status na STATUS=2
3. Sve dok **RED** / **MAGACIN** ne bude prazan
   a) Uzeti čvor sa početka **REDa** / **MAGACINa**. Obraditi N u promeniti mu STATUS=3
   b) Dodati u **RED** / **MAGACIN** sve susede čvora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

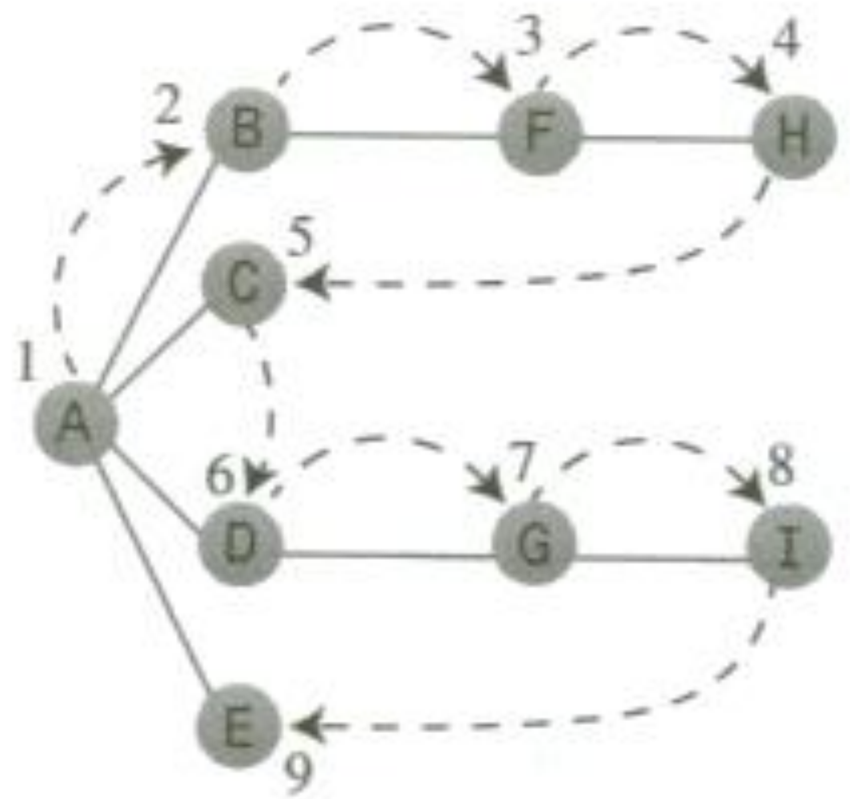
Redosled obilaska po BFS



DFS

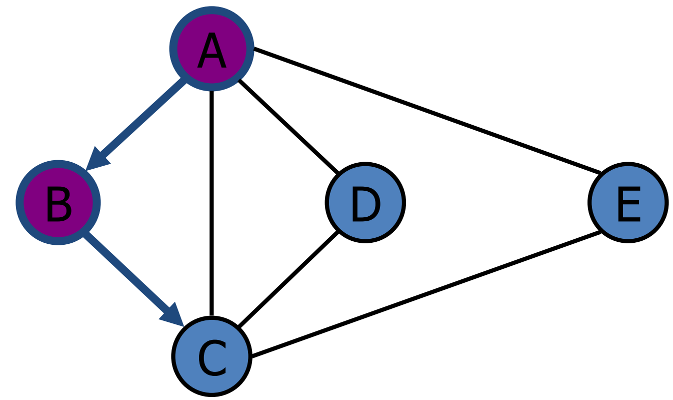
- DFS je generalna tehnika za obilazak grafa
- DFS obilazak
  - Obiđi sve čvorove i potege grafa G
  - Određuje da li je graf povezan
  - Određuje povezane komponente grafa G
  - Određuje *spanning forest* grafa G
- DFS za graf sa $\boldsymbol{n}$ čvorova i $\boldsymbol{m}$ potega zahteva $\boldsymbol{O}(\boldsymbol{n}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

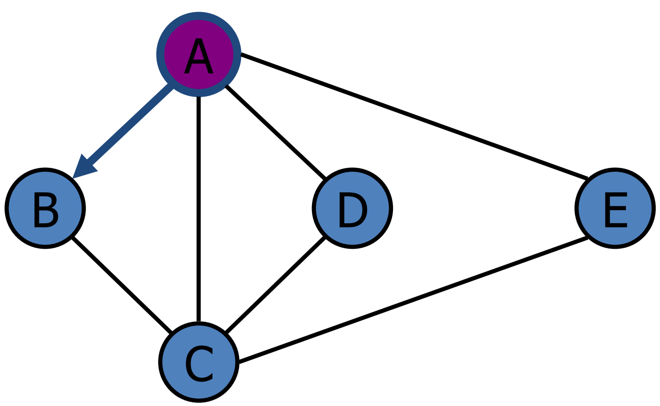
REDOSLED OBILASKA PO DFS

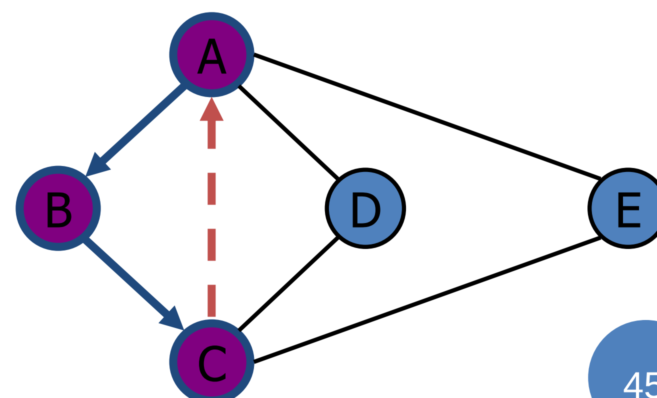


Primer za DFS

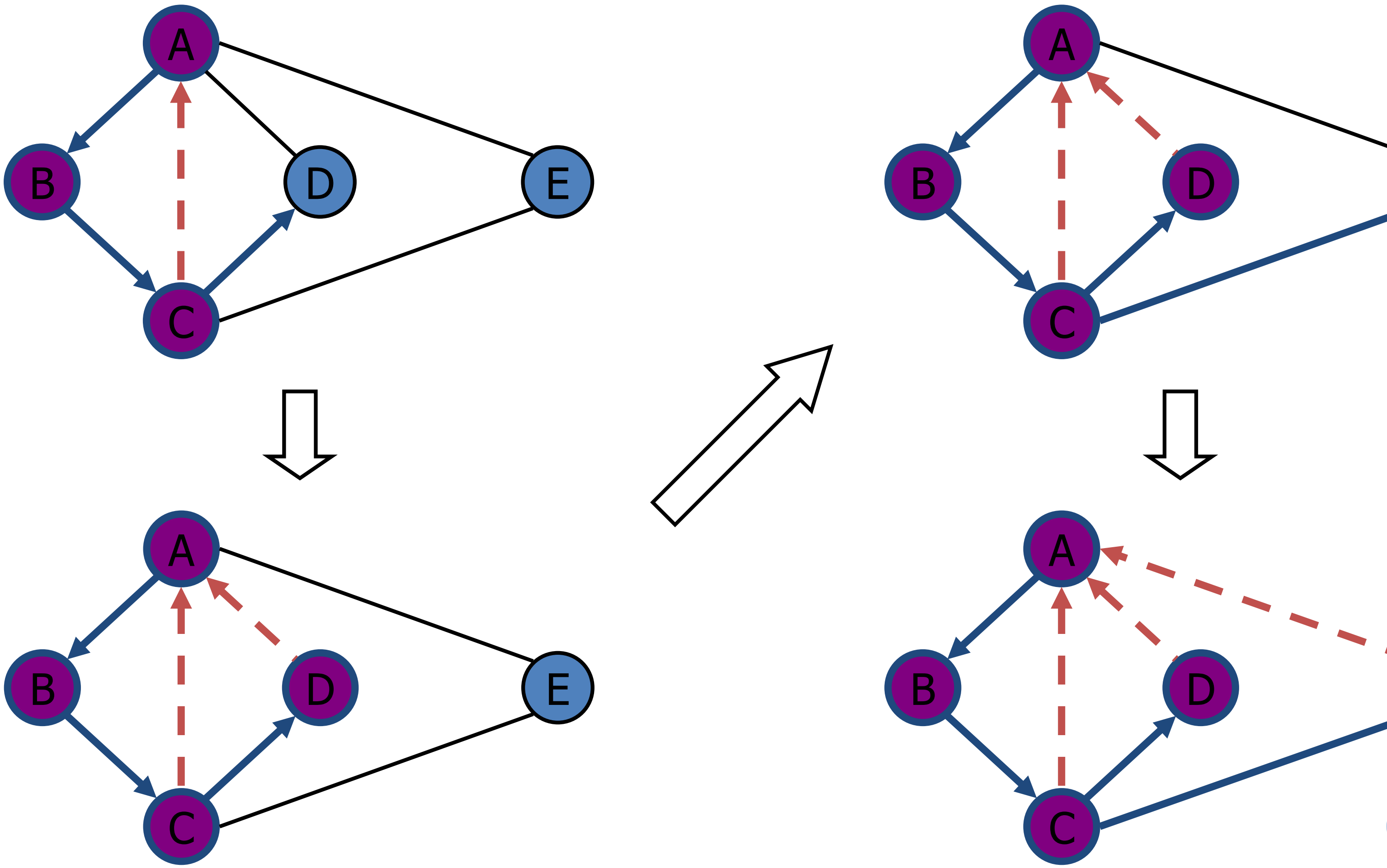
- A Čvor kome je SATUS=1
- A STATUS=3
- Neobrađeni poteg
- Poteg koji se obrađuje
- Povratni poteg



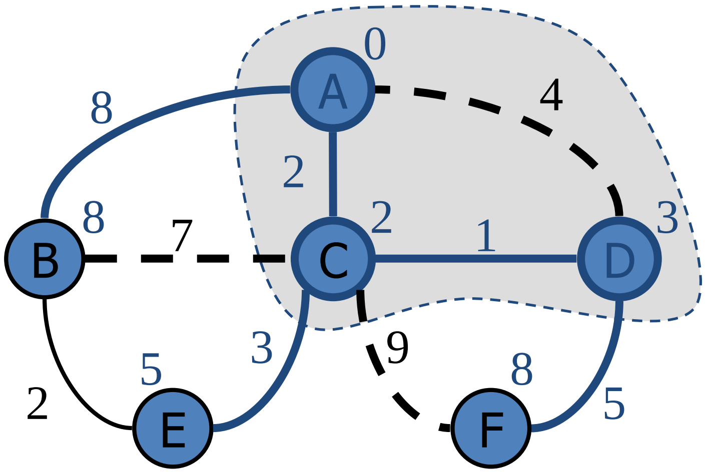




Primer za DFS (nast.)



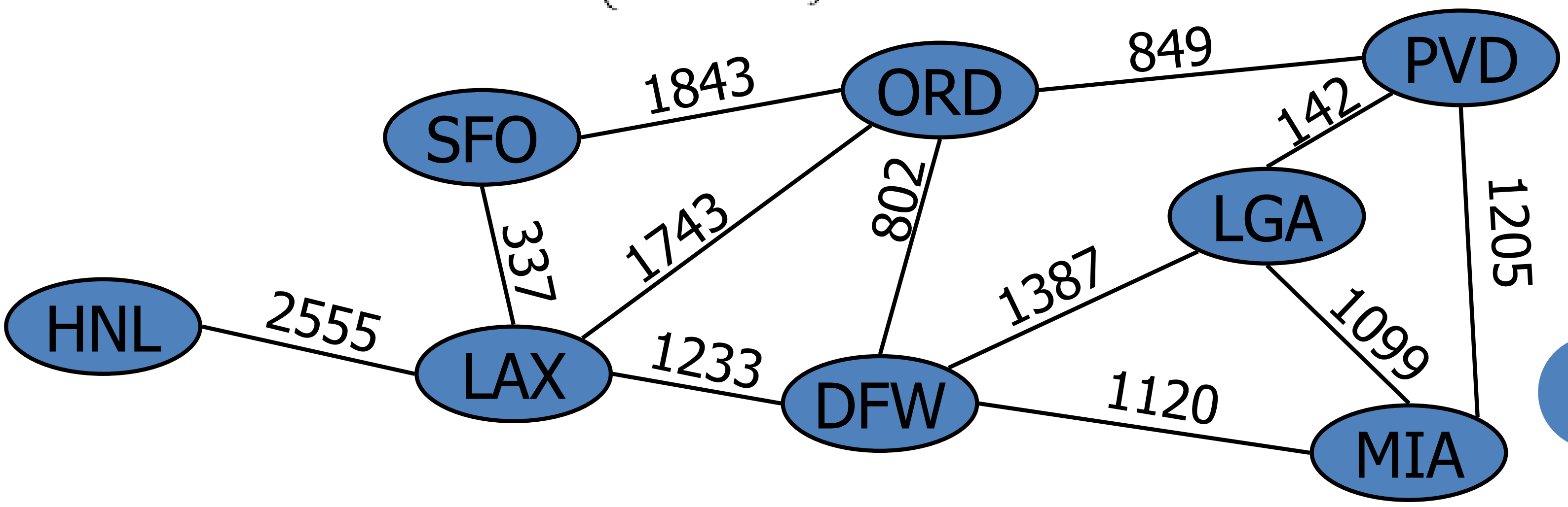
Najkraći put u grafu



PODSETNIK: TEŽINSKI GRAF

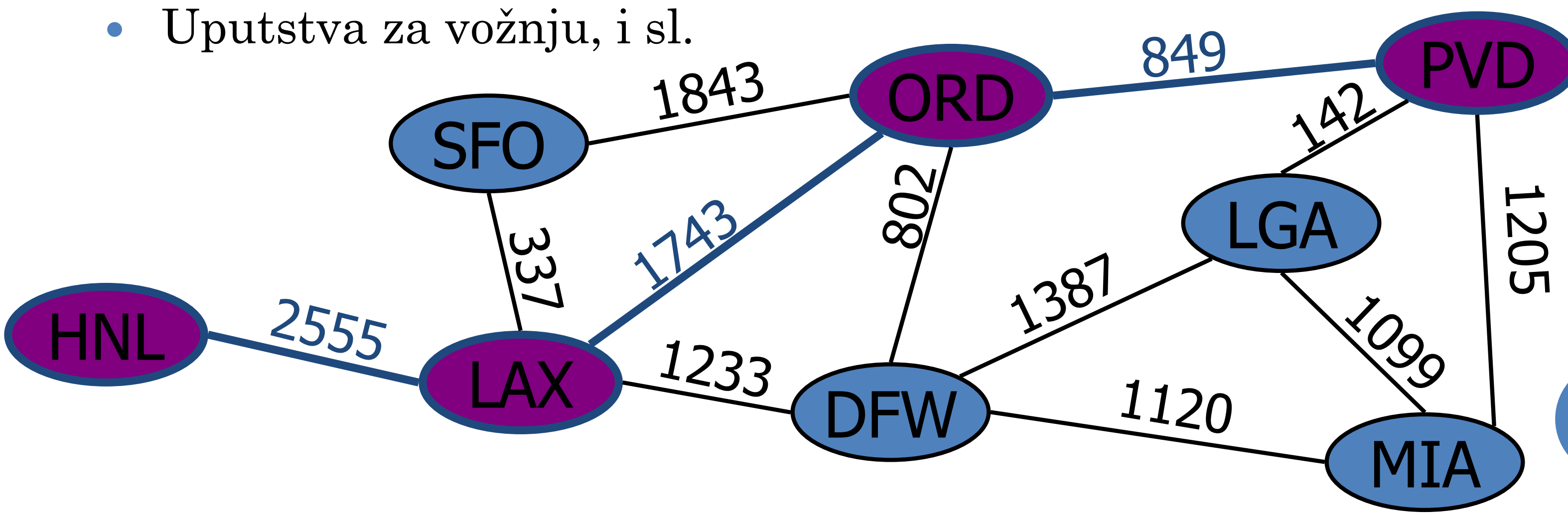
- U težinskom grafu svaki poteg ima dodeljen broj, koji definiše težinu potega
- Težina može biti rastojanje, cena, ili neka druga vrednost
- Primer:
  - U grafu koji predstavlja trase letova, težine potega su rastojanja između dva aerodroma
- Matrica težina: umesto 1 i 0, vrednosti su težine potega 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:

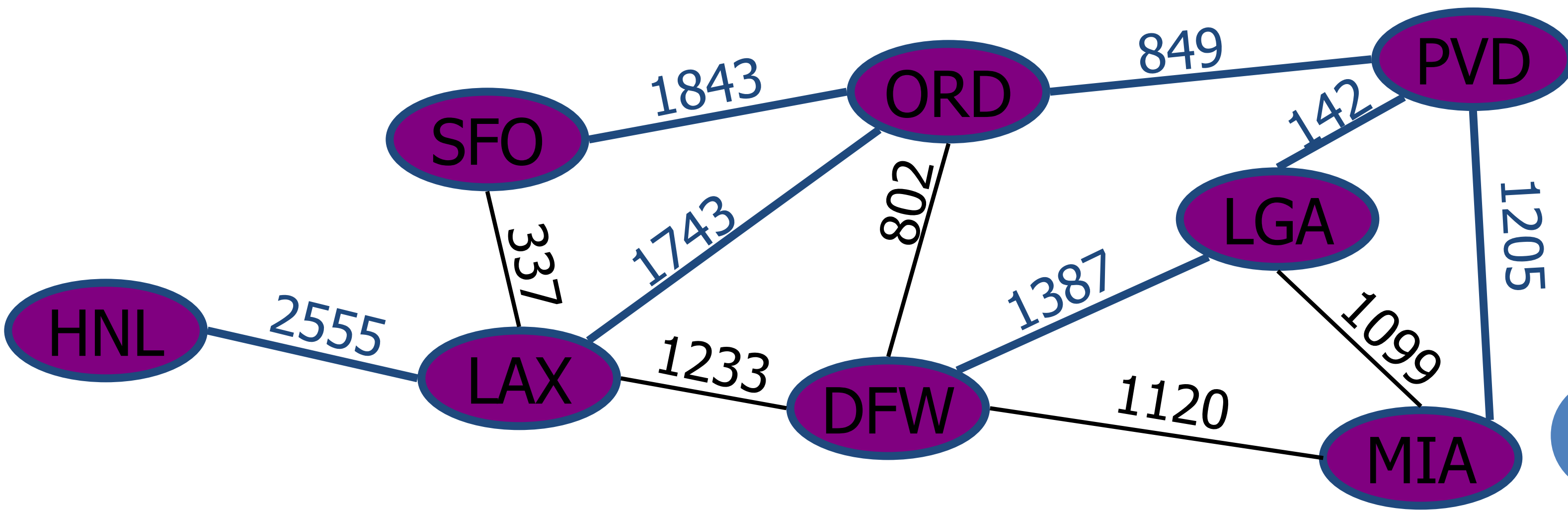
Put koji je deo najkraćeg puta je takođe najkraći put

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
- Grafimo matricu 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

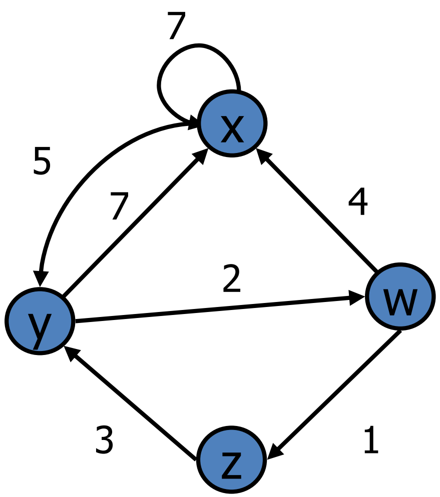
Modifikovani Warshall-ov algoritam – pseudokod

Algoritam G.3. Modifikovani Warshall-ov algoritam

ModifWarshall(W,m)

```
{// data je matrica težina W i broj čvorova m
 // algoritam generiše matricu Q
repeat for (i=1,m)      //inicijalizacija matrice Q0
    { repeat for (j=1,m)
        { if (W[i,j]=0)
            then Q[i,j]=MAX
            else Q[i,j]=W[i,j] }}
 repeat for k=1,m     //Azuriranje Q
    {repeat for i=1,m
       {repeat for j=1,m
          Q[i,j]=min(Q[i,j], (Q[i,k] + Q[k,j]) }}
return}
```

MODIFIKOVANI WARSHALL-OV ALGORITAM - PRIMER



$$Q_0 = W = \begin{array}{c|cccc|} & x & y & z & w \\ \hline x & 7 & 5 & \infty & \infty \\ y & 7 & \infty & \infty & 2 \\ z & \infty & 3 & \infty & \infty \\ w & 4 & \infty & 1 & \infty \end{array}$$

+

min

$$Q_1 = \begin{array}{c|cccc|} & x & y & z & w \\ \hline x & 7 & 5 & \infty & \infty \\ y & 7 & 12 & \infty & 2 \\ z & \infty & 3 & \infty & \infty \\ w & 4 & 9 & 1 & \infty \end{array}$$

$$Q_2 = \begin{array}{c|cccc|} & x & y & z & w \\ \hline x & 7 & 5 & \infty & 7 \\ y & 7 & 12 & \infty & 2 \\ z & 10 & 3 & \infty & 5 \\ w & 4 & 9 & 1 & 11 \end{array}$$

$$Q_3 = \begin{array}{c|cccc|} & x & y & z & w \\ \hline x & 7 & 5 & \infty & 7 \\ y & 7 & 12 & \infty & 2 \\ z & 10 & 3 & \infty & 5 \\ w & 4 & 4 & 1 & 6 \end{array}$$

$$Q = Q_4 = \begin{array}{c|cccc|} & x & y & z & w \\ \hline x & 7 & 5 & 8 & 7 \\ y & 7 & 11 & 3 & 2 \\ z & 9 & 3 & 6 & 5 \\ w & 4 & 4 & 1 & 6 \end{array}$$

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
- 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***

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

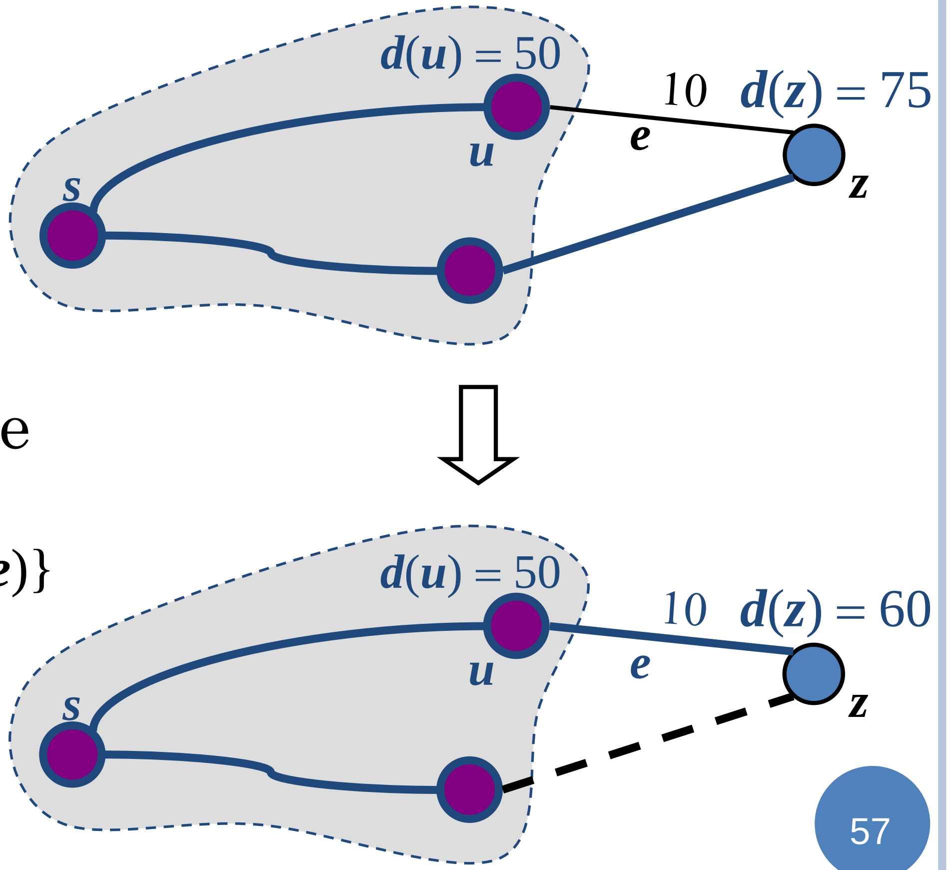
Algoritam G.5.
***DijkstraShortestPathsTree*(*G, s*)**

```
    …

  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)
```

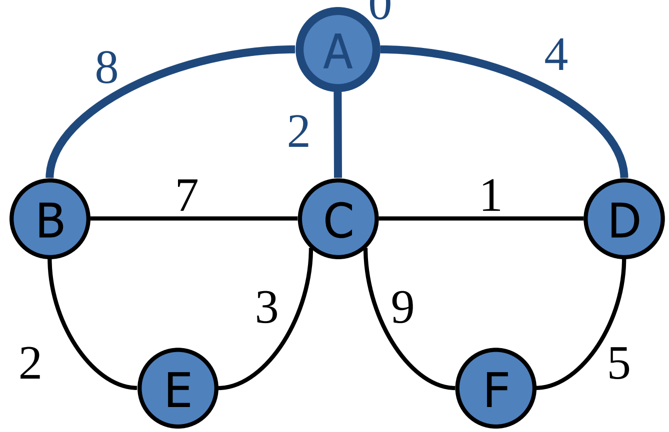
Relaksacija potega

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

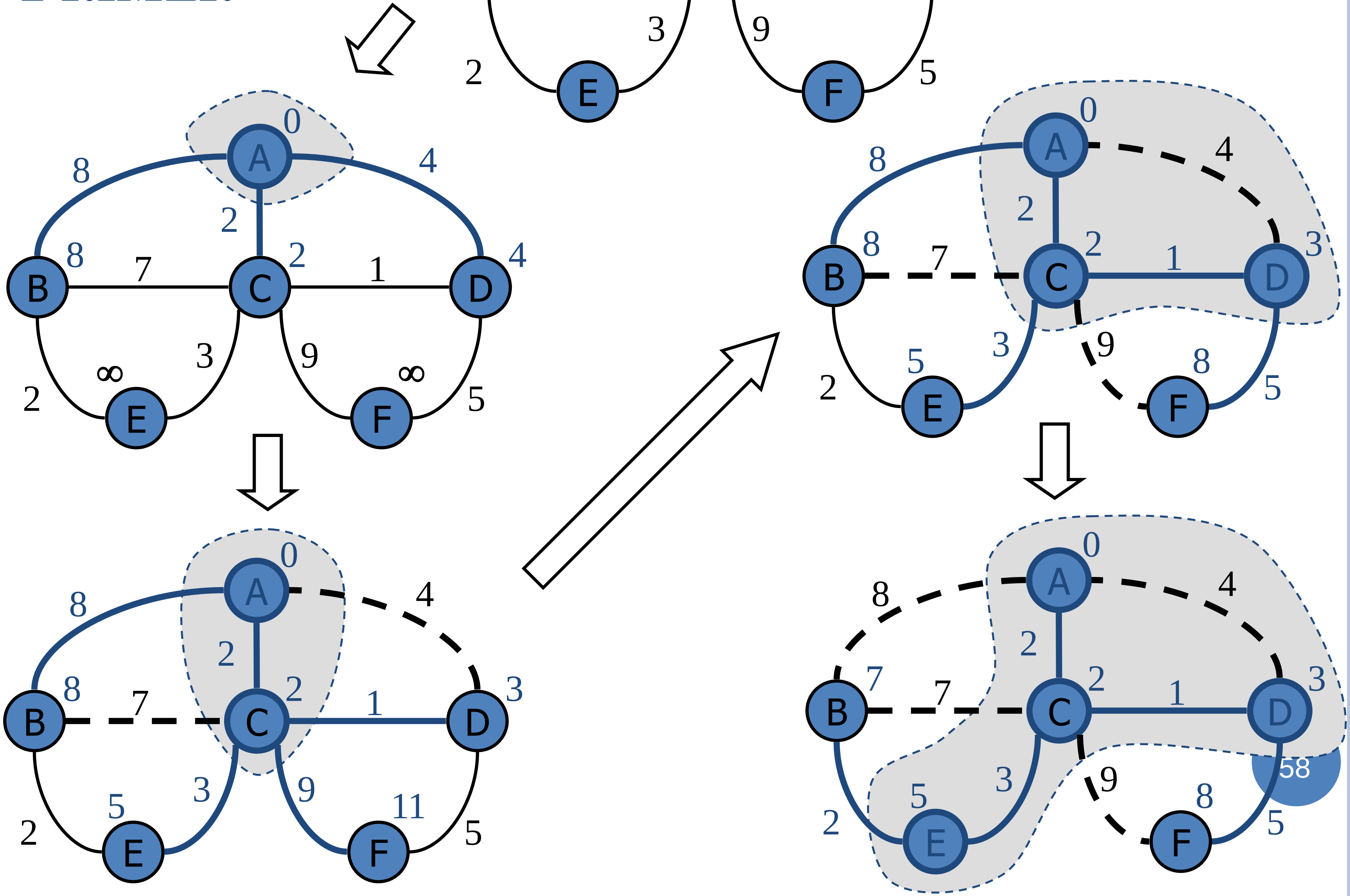
$$\boldsymbol{d}(\boldsymbol{z}) \leftarrow \min\{\boldsymbol{d}(\boldsymbol{z}),\boldsymbol{d}(\boldsymbol{u}) + \boldsymbol{weight}(\boldsymbol{e})\}$$



PRIMER

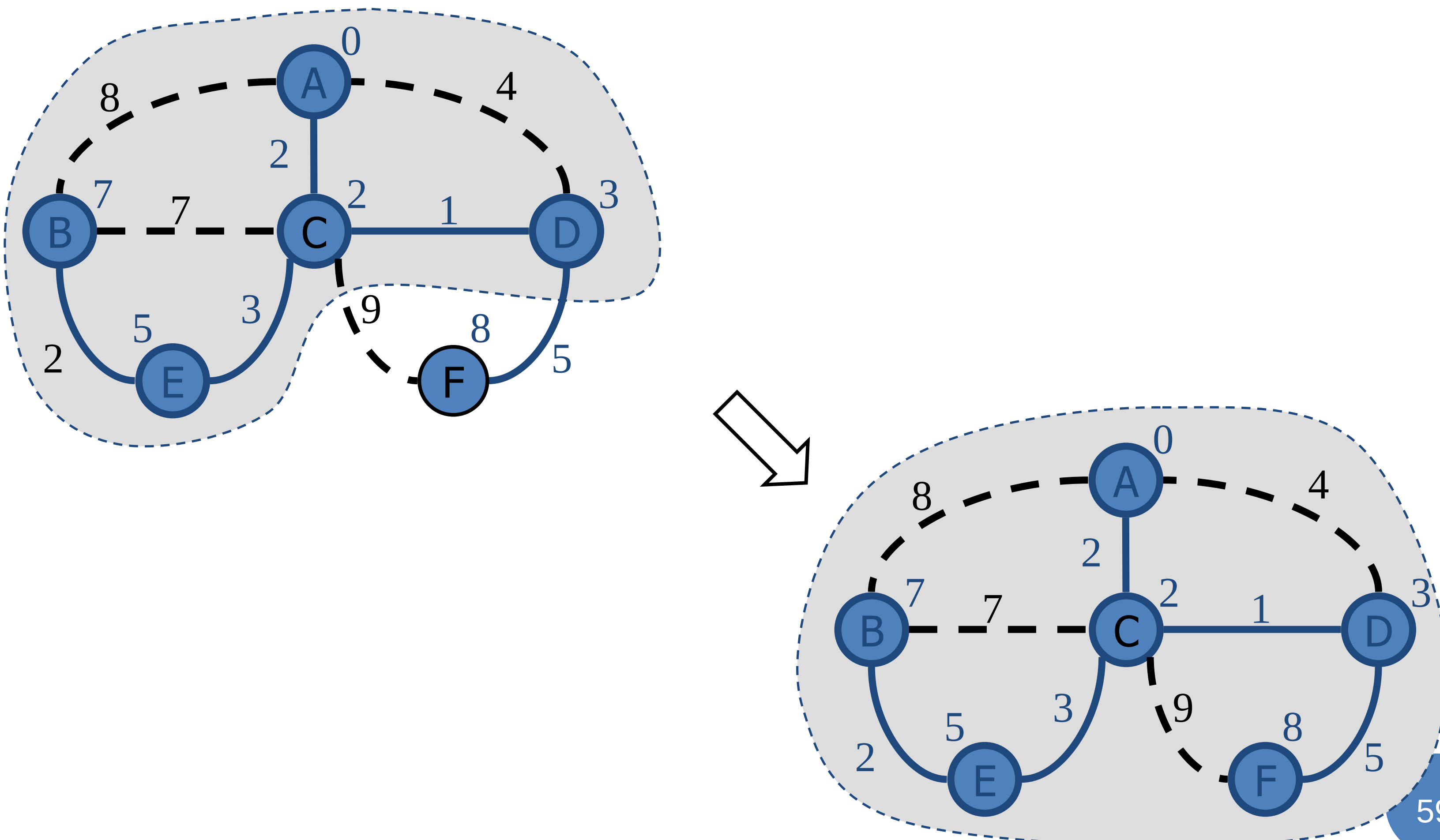


A B C D E F
0
8 4 2 7 1 3 9 2 5



A B C D E F
0 8 2 4 ∞ ∞
8 4 2 7 1 3 9 2 5
0 8 2 3 5 11
0 8 2 3 5 8
0 7 2 3 5 8

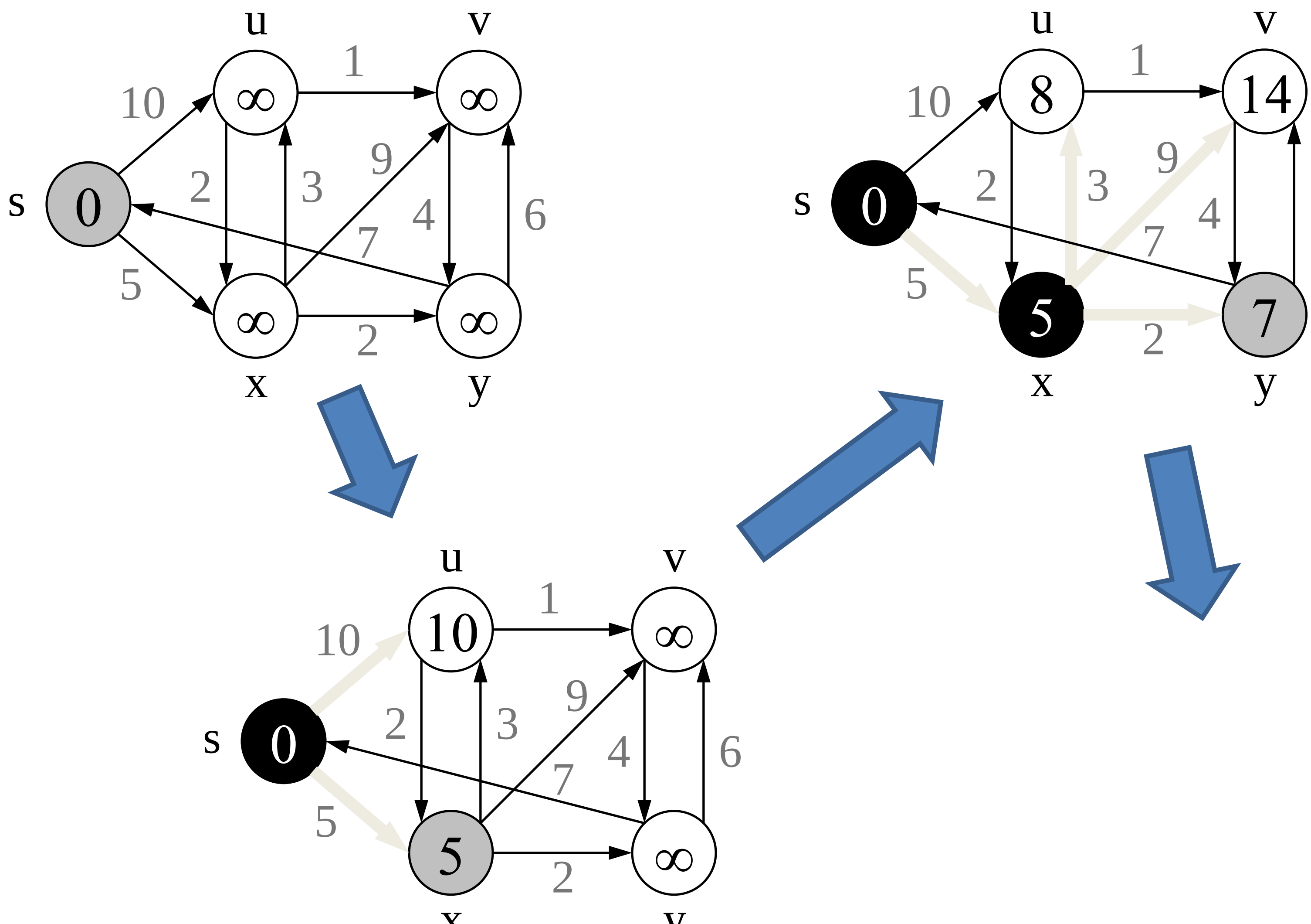
PRIMER (NAST.)



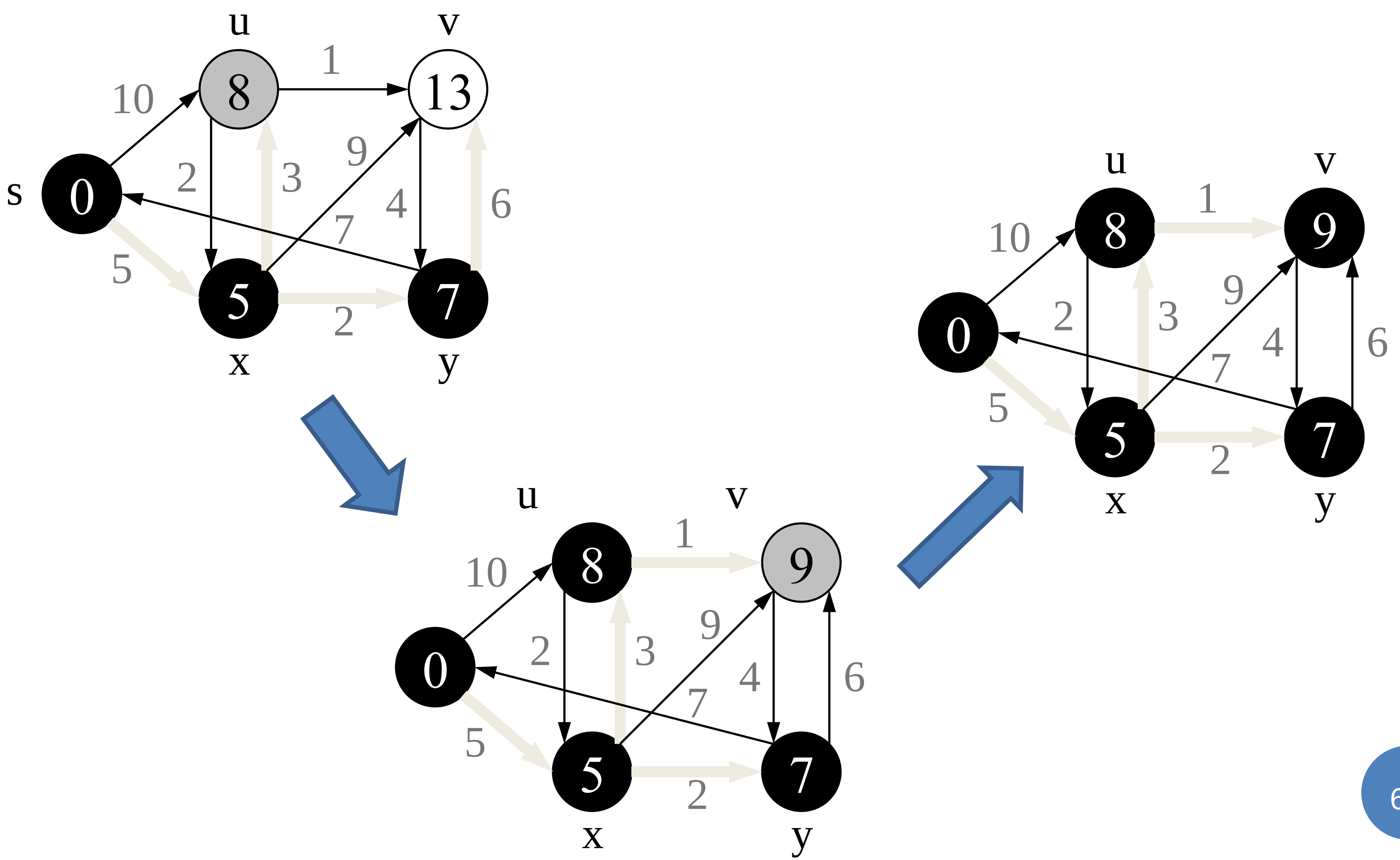
Dijkstra Algoritam - pregled

- 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.)

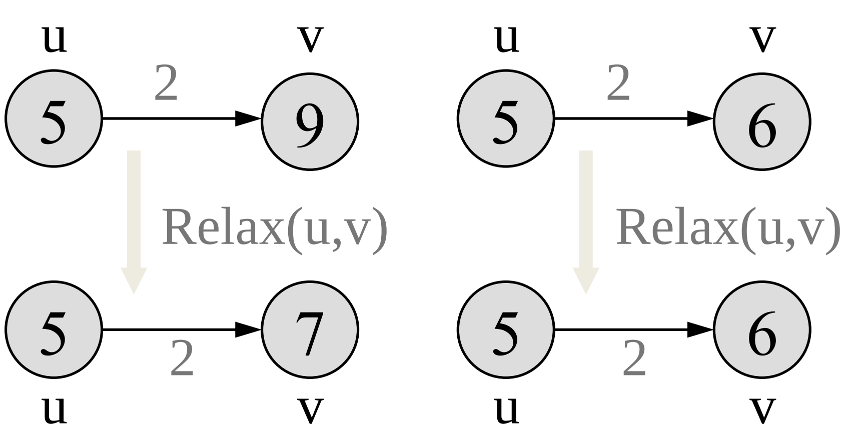


Bellman-Ford Algoritam

- 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



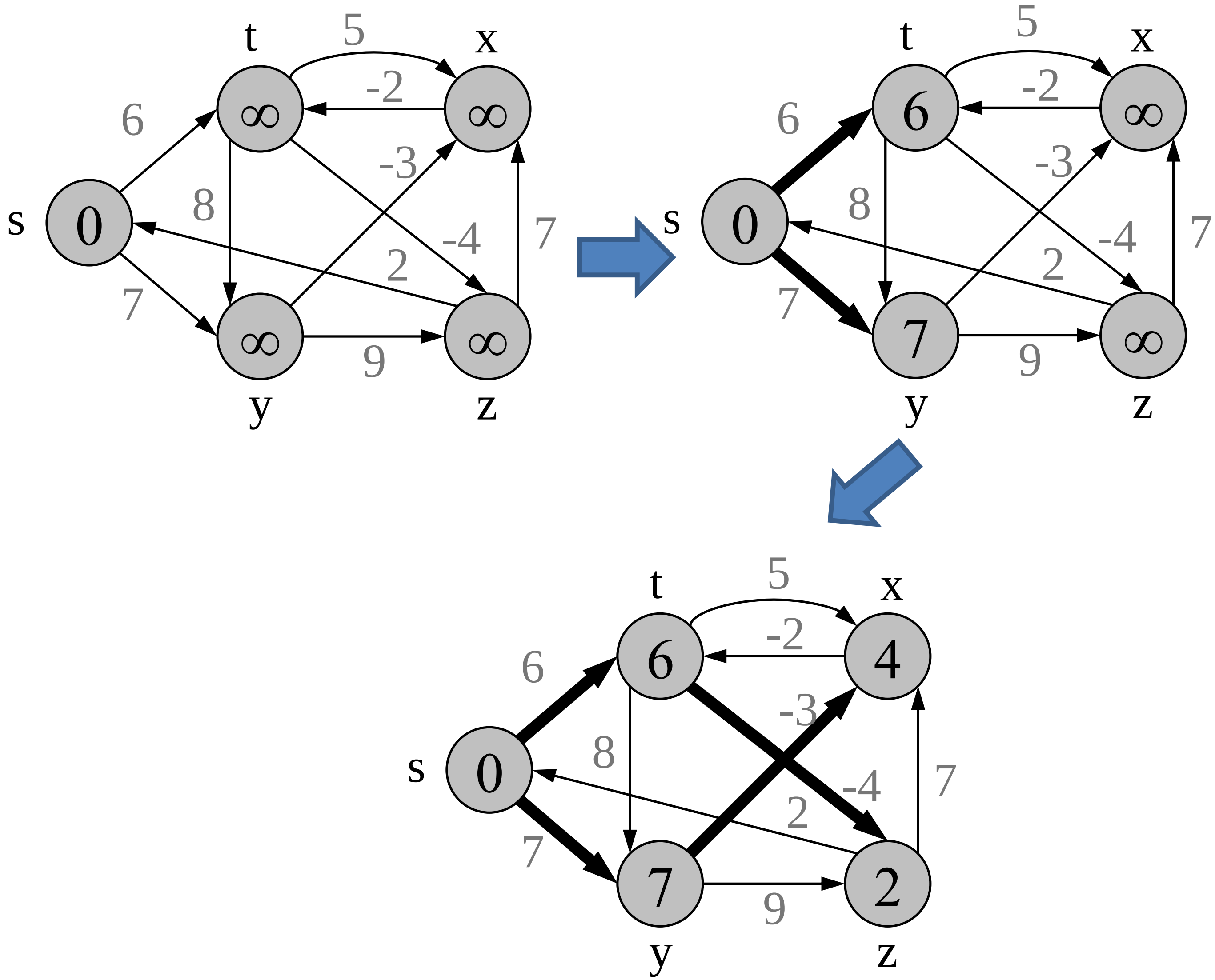

```
Relax (u,v,G)
if v.d() > u.d()+G.w(u,v) then
   v.setd(u.d()+G.w(u,v))
   v.setparent(u)
```

Bellman-Ford Algoritam

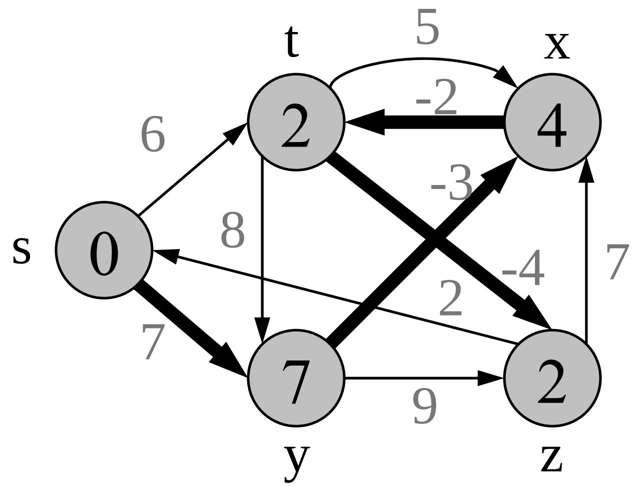
```
Bellman-Ford(G,s)
for each vertex u ∈ G.V()
    u.setd(∞)
    u.setparent(NIL)
s.setd(0)
for i ← 1 to |G.V()|-1 do
    for each edge (u,v) ∈ G.E() do
        Relax (u,v,G)
for each edge (u,v) ∈ G.E() do
    if v.d() > u.d() + G.w(u,v) then
        return false
return true
```

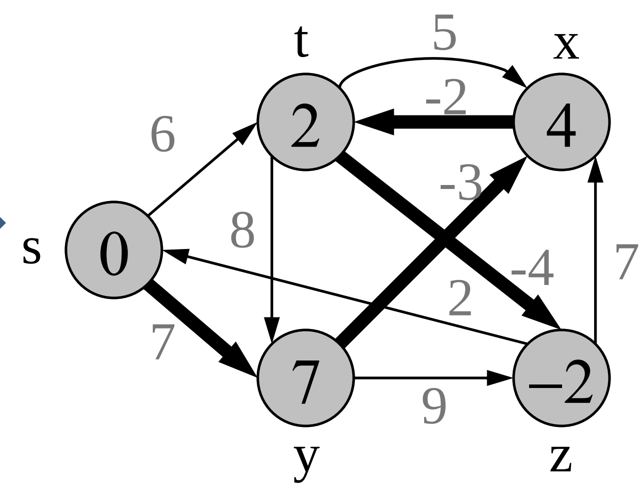

65

Bellman-Ford – Primer



Bellman-Ford Example





Pitanja, ideje, komentari