Teorija Algoritama

Algoritmi nad grafovima

Algoritmi nad grafovima

- **Traženje najkraće putanje**;
- Minimalno razapinjujuće (*eng. spanning*) stablo;
- Maksimalni protok;
- Detekcija ciklusa;
- Redukcija grafa.

Problem traženja najkraće putanje

- U usmerenom grafu $G = (V, E)$, sa težinskom funkcijom $w: E \rightarrow R$, težina putanje se definiše kao suma težina grana na toj putanji. Najkraća putanja između čvorova u i v u grafu G podrazumeva putanju između tih čvorova sa najmanjom težinom.
- Varijacije problema:
  - Najkraće putanje od čvora u do svih ostalih čvorova (single source);
  - Najkraće putanje od svih čvorova do čvora u (single destination);
  - Najkraća putanja između čvora u i v (single pair);
  - Najkraće putanje za sve parove čvora u grafu (all pairs).

Bellman-Ford algoritam

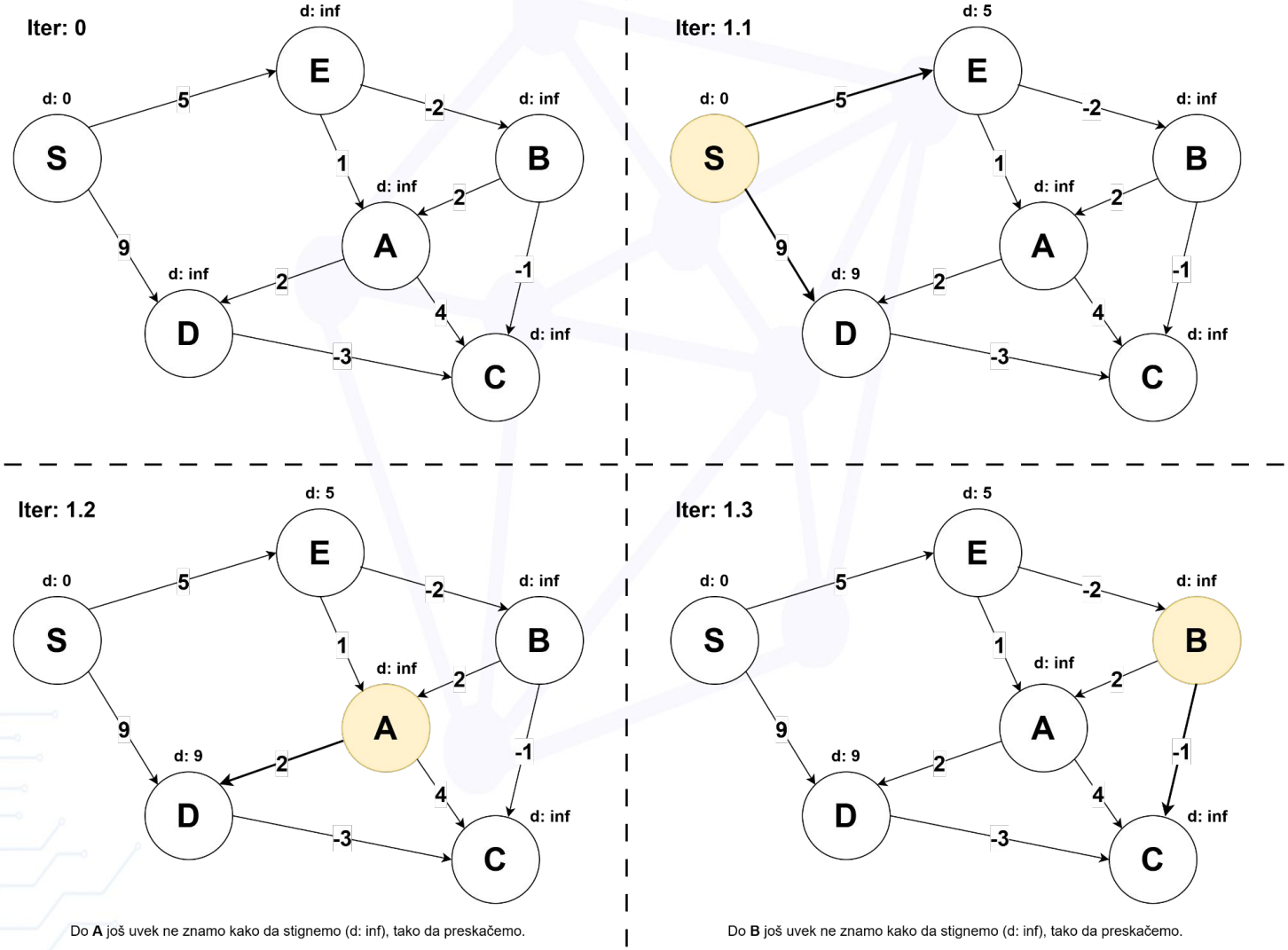
- Alfonso Shimbel (1955), Richard Bellman (1958), Lester Ford Jr. (1956), Edward F. Moore (1957)
- *Bellman-Ford* algoritam rešava single-source problem najkraće putanje u težinskom grafu u opštem slučaju (težine grana mogu biti negativne)
- *Bellman-Ford* algoritam otkriva i postojanje ciklusa sa negativnom težinom, što uzrokuje nepostojanje najkraće putanje - ukoliko postoji negativan ciklus to znači da je najkraća putanja -∞.
- Koristi se postupak relaksacije grana.
- Vremenska složenost: $\Theta(|E| |V|)$ gde je E skup ivica (grana), a V skup čvorova (temena).

Bellman-Ford algoritam - *pseudo-kod*

```
function initSingleSource(G, s)
    foreach u in G.V
        u.d = inf
        u.p = NIL
    s.d = 0

function relax(u, v, w)
    if v.d > u.d + w(u, v)
        v.d = u.d + w(u, v)
        v.p = u
```

```
function bellmanFord(G, w, s)
    initSingleSource(G, s)
    for i = 1 to |G.V| - 1
        foreach (u, v) in G.E
            relax(u, v, w)
    foreach (u, v) in G.E
        if v.d > u.d + w(u, v)
            return FALSE
    return TRUE
```

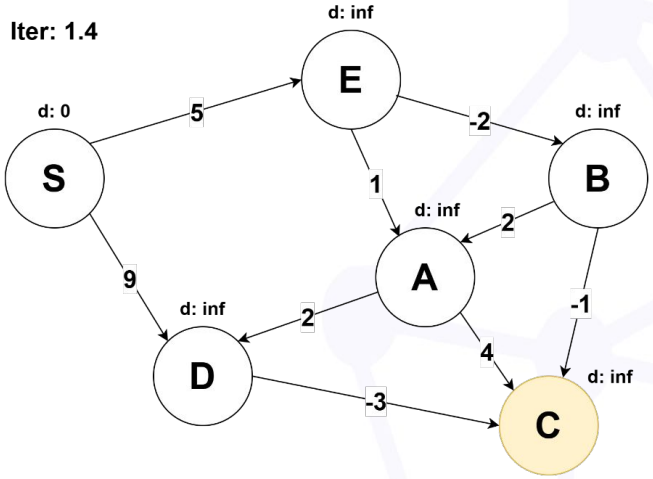
Bellman-Ford algoritam - primer



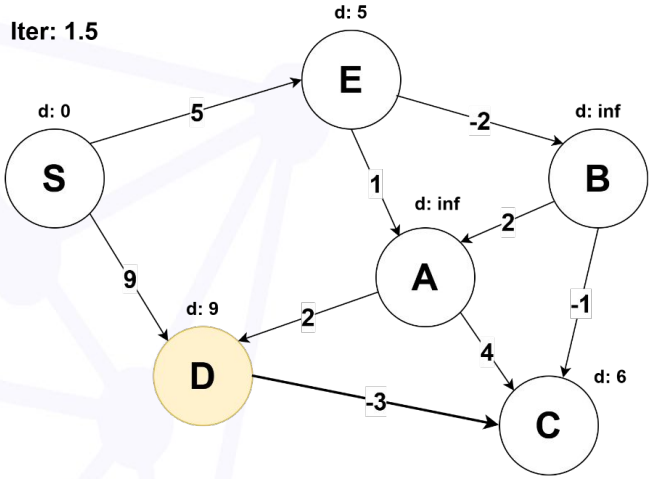
Do **A** još uvek ne znamo kako da stignemo (d: inf), tako da preskačemo.

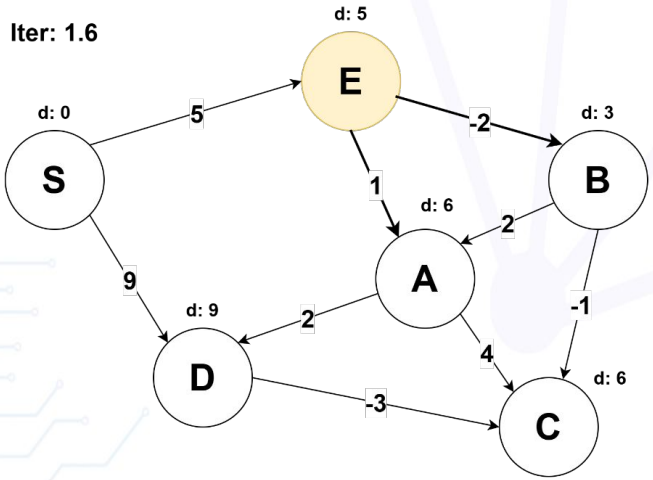
Do **B** još uvek ne znamo kako da stignemo (d: inf), tako da preskačemo.

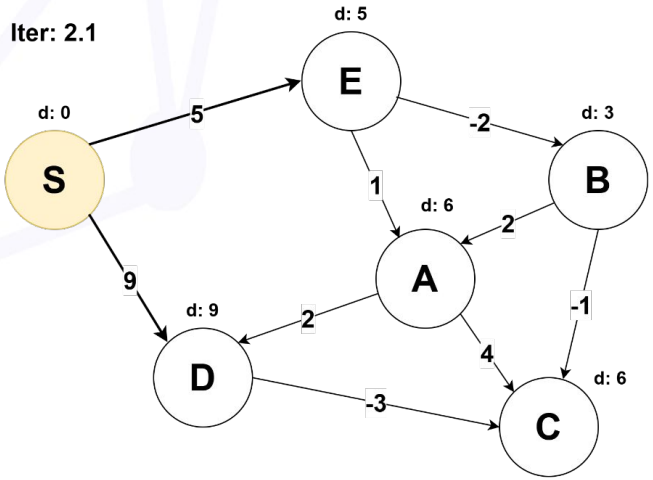
Bellman-Ford algoritam - primer



C nema izlaznih grana.

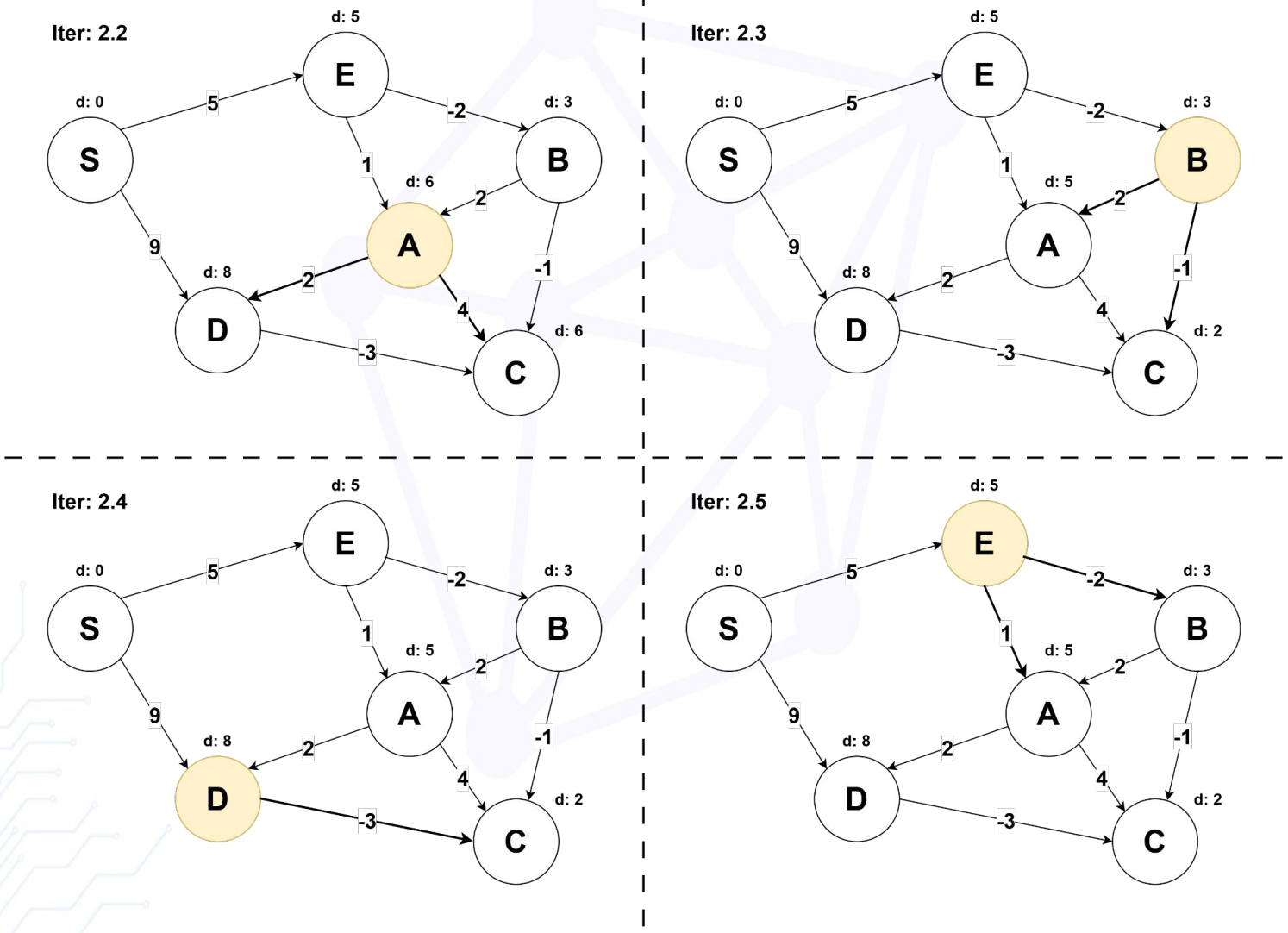




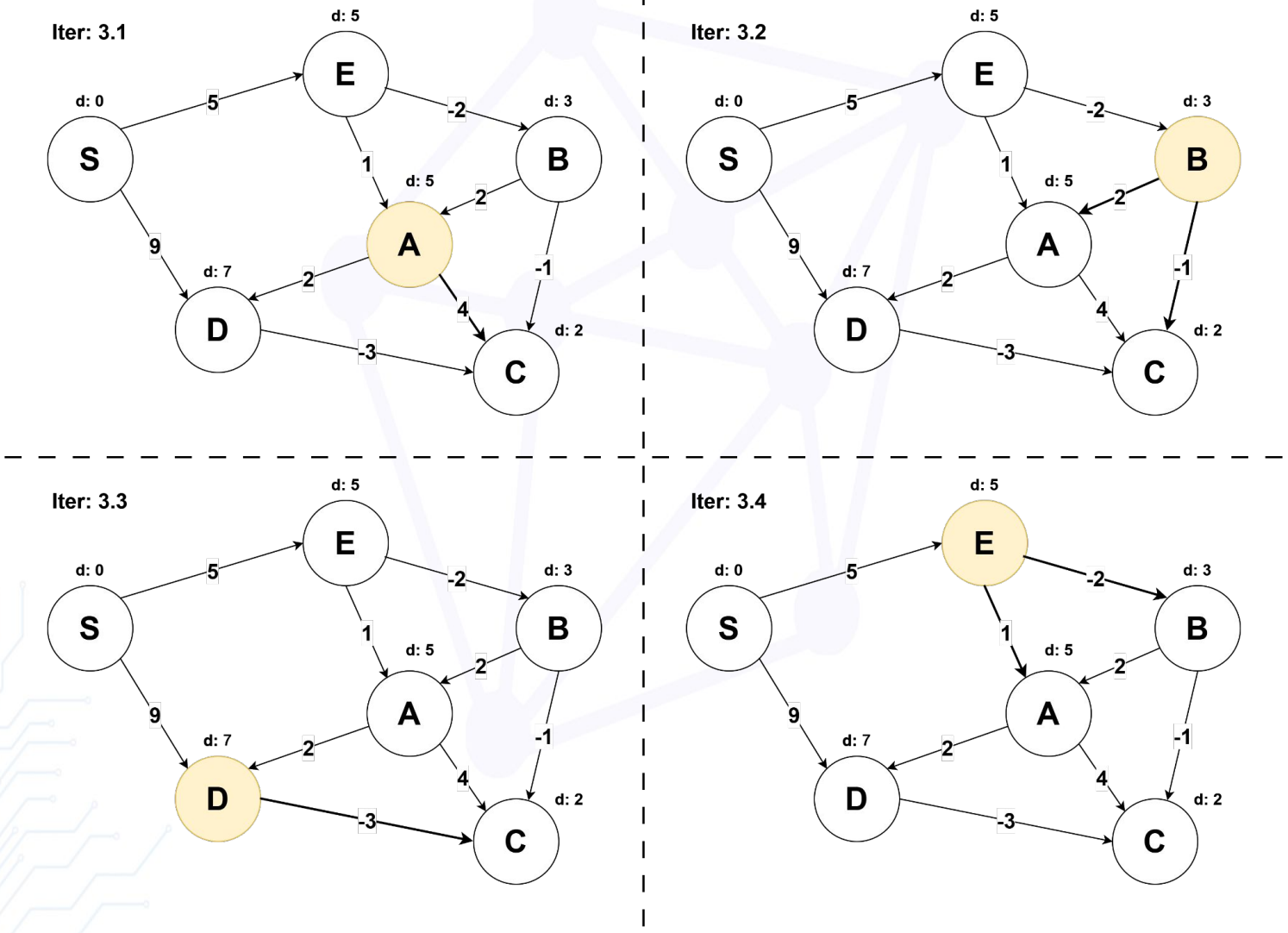


Početak 2. iteracije - za čvor S nema promena

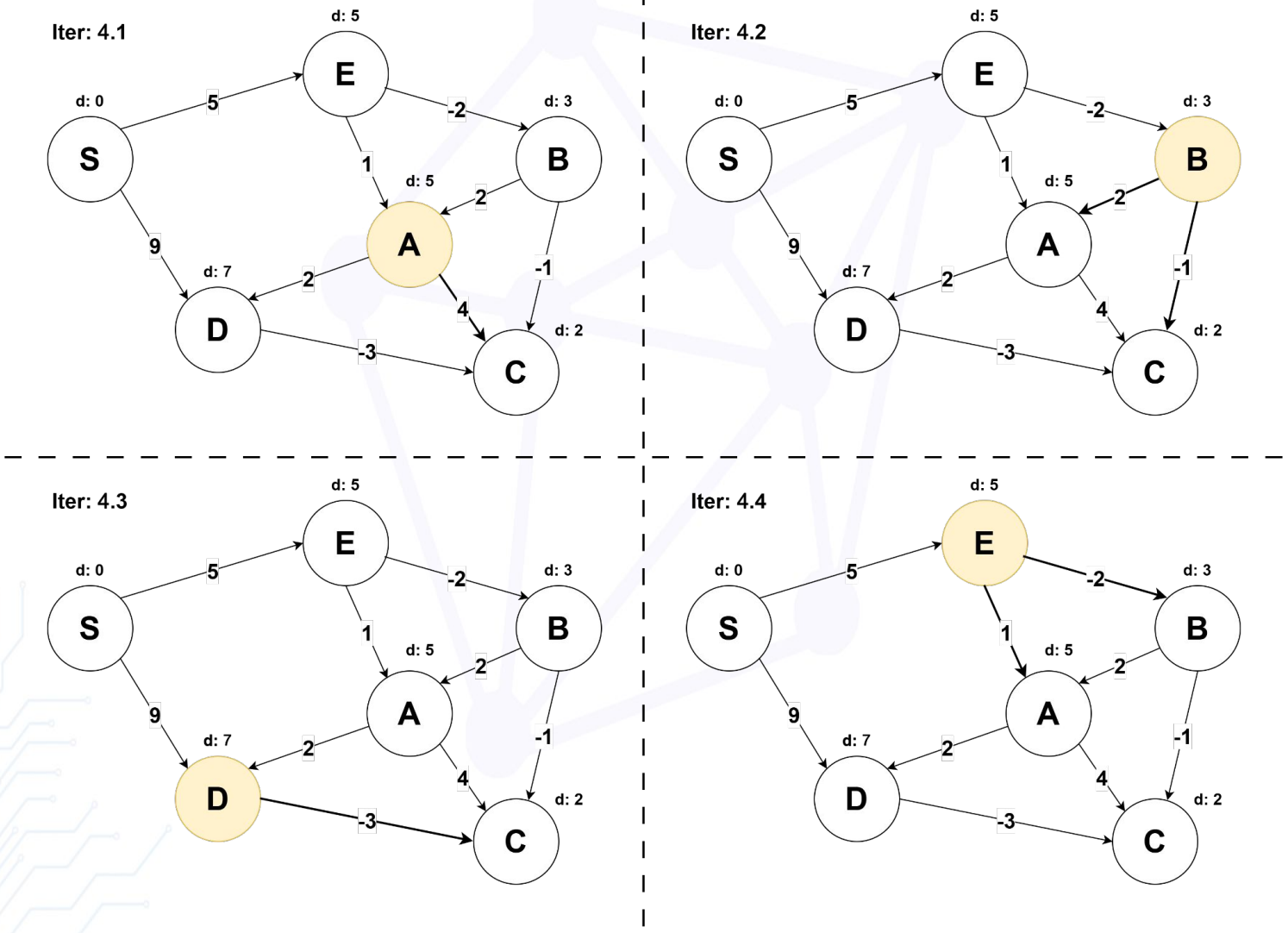
Bellman-Ford algoritam - primer



Bellman-Ford algoritam - primer



Bellman-Ford algoritam - primer

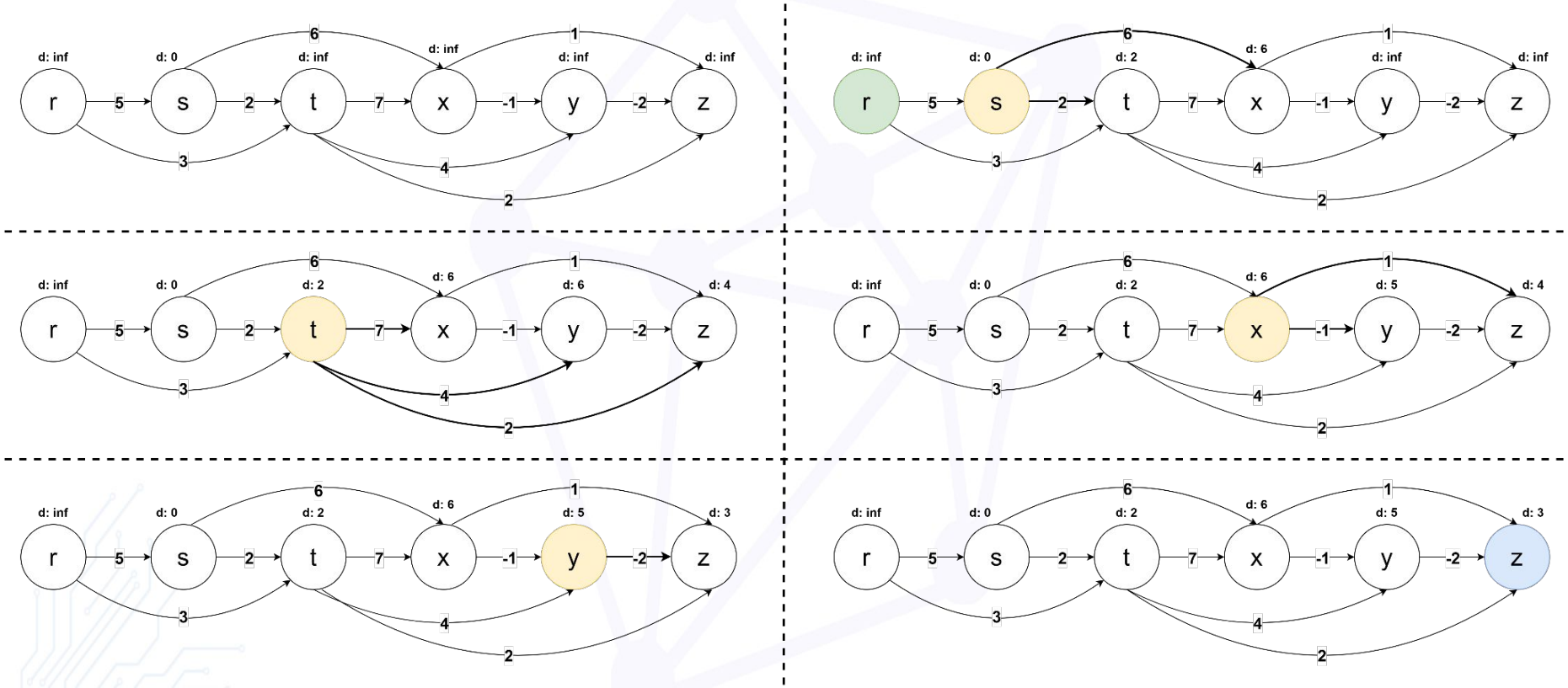


Najkraće putanje u DAG

- DAG (usmereni aciklični grafovi) - nemaju cikluse što olakšava traženje najkraće putanje.
- Topološki sort (sa prethodnih vežbi) pomaže pri odabiru kojem narednom čvoru treba relaksirati izlazne grane.

```
function dagShortestPaths(G, w, s)
    sortedList = topologicallySort(G)
    initSingleSource(G, s)
    foreach u in sortedList
        foreach v in G.adj[u]
            relax(u, v, w)
```

Najkraće putanje u DAG - primer



Dijkstrin algoritam

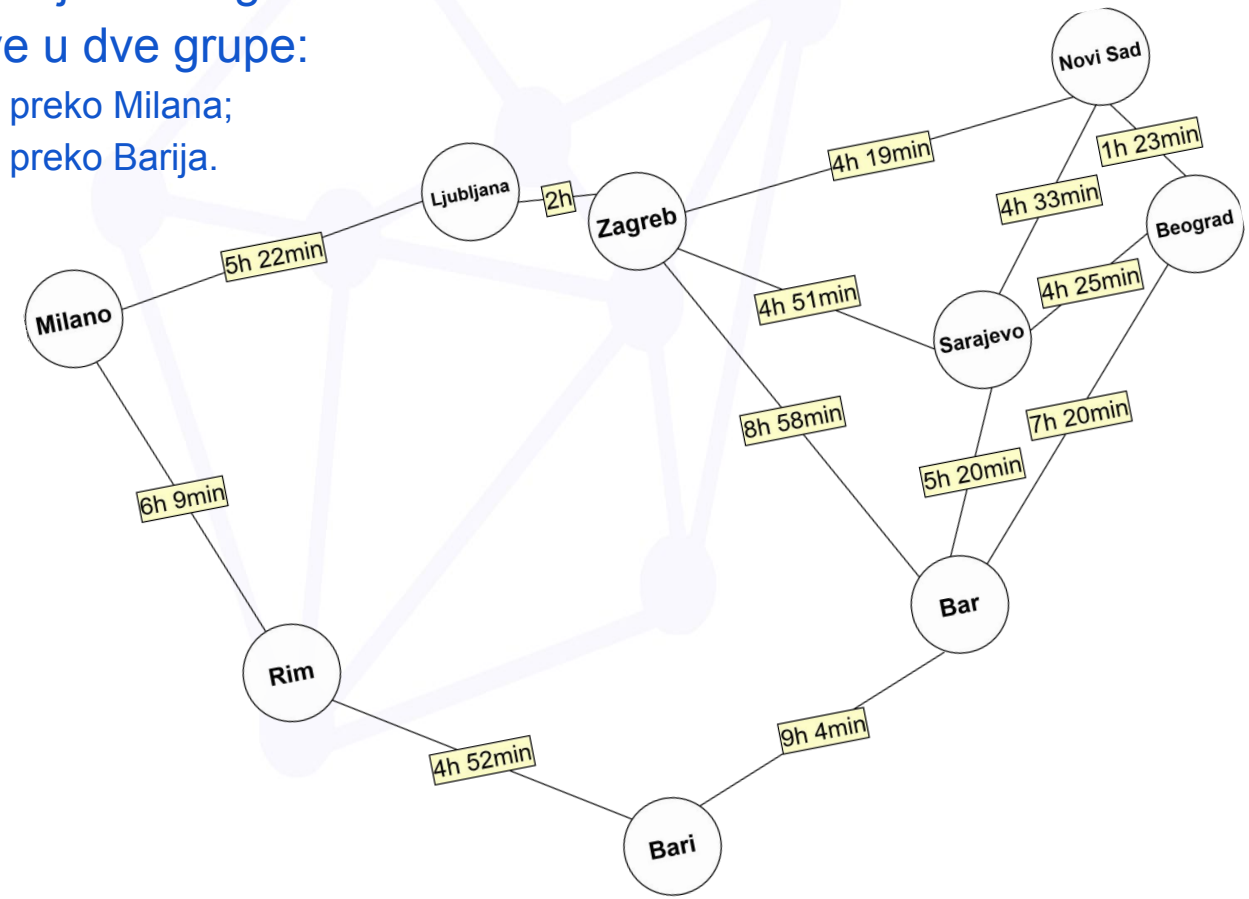
- Edsger W. Dijkstra (1956).
- Dijkstrin algoritam traži najkraće putanje u grafu pod pretpostavkom da su težine svih grana u grafu nenegativne.
- Algoritam održava skup čvorova ***S*** za koje je težina najkraće putanje već određena.
- Iterativno se određuje čvor iz ostalih ***V - S*** sa najmanjom procenom najkraće putanje i potom se dodaje u ***S*** uz relaksaciju njegovih izlaznih grana.
- Računska složenost: O(|V| * |V|), a korišćenjem *min-heap* kao prioritetnog reda: O(|E| + |V| log(|V|))

Dijkstra algoritam - *pseudo-kod*

```
function dijkstra(G, w, s)

    initSingleSource(G, s)
    Q = G.V

    while notEmpty(Q)
        u = extractMin(Q)
        foreach v in G.adj[u]
            relax(u, v, w)
```

Dijkstra algoritam - vežba

- Odrediti najkraće putanje od postojećih (na grafu) evropskih gradova do Rima, koristeći Dijkstrin algoritam.
- Podeliti gradove u dve grupe:
  - Najbrži put je preko Milana;
  - Najbrži put je preko Barija.



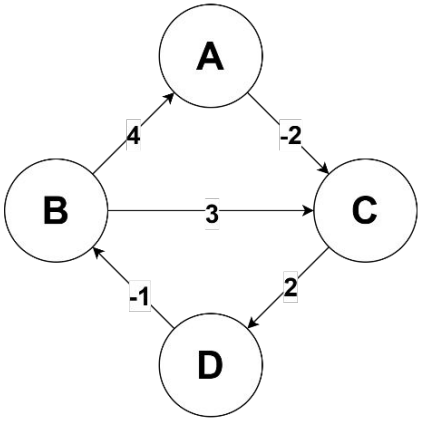
Floyd-Warshall algoritam

- Robert Floyd (1962), Stephen Warshall (1962), Bernard Roy (1959).
- Rešava *all-pairs* problem traženja najkraćih putanja u usmerenom težinskom grafu bez negativnih ciklusa.
- Koristi matricu za čuvanje svih najkraćih putanja u grafu.
- Može se modifikovati da vrši detekciju negativnih ciklusa, kao i da pamti prethodnike u najkraćim putanjama.
- Vremenska složenost $\Theta(|V| * |V| * |V|)$.

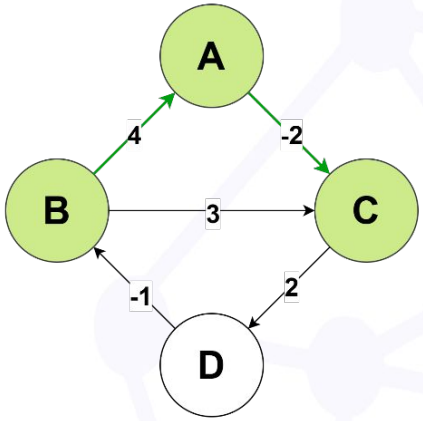
Floyd-Warshall algoritam - *pseudo-kod*

```
function floydWarshall(G)

    foreach (u, v) in G.E
        dist[u][v] = w(u,v)

    foreach v in G.V
        dist[v][v] = 0

    foreach k = 0 to |V|
        foreach i = 0 to |V|
            foreach j = 0 to |V|
                if dist[i][j] > dist[i][k] + dist[k][j]
                    dist[i][j] = dist[i][k] + dist[k][j]
```

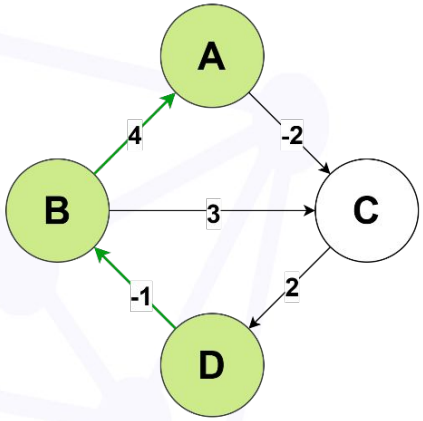
Floyd-Warshall algoritam - primer



INIT		j			
		A	B	C	D
i	A	0	∞	-2	∞
	B	4	0	3	∞
	C	∞	∞	0	2
	D	∞	-1	∞	0

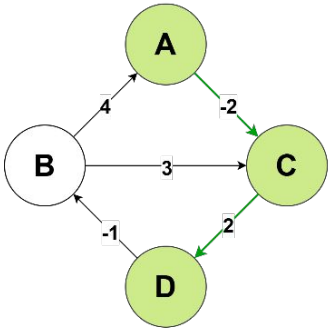


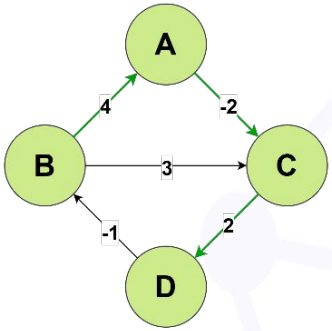
k = 0 (A)		j			
		A	B	C	D
i	A	0	∞	-2	∞
	B	4	0	2	∞
	C	∞	∞	0	2
	D	∞	-1	∞	0



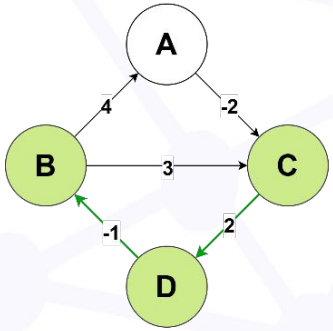
k = 1 (B)		j			
		A	B	C	D
i	A	0	∞	-2	∞
	B	4	0	2	∞
	C	∞	∞	0	2
	D	3	-1	1	0

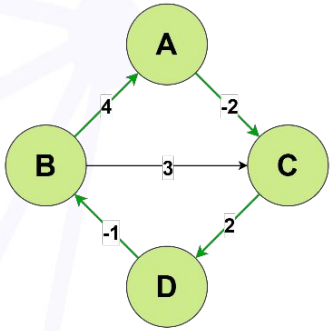
Floyd-Warshall algoritam - primer





k = 2 (C)		j			
		A	B	C	D
i	A	0	∞	-2	0
	B	4	0	2	4
	C	∞	∞	0	2
	D	3	-1	1	0





k = 3 (D)		j			
		A	B	C	D
i	A	0	-1	-2	0
	B	4	0	2	4
	C	5	1	0	2
	D	3	-1	1	0