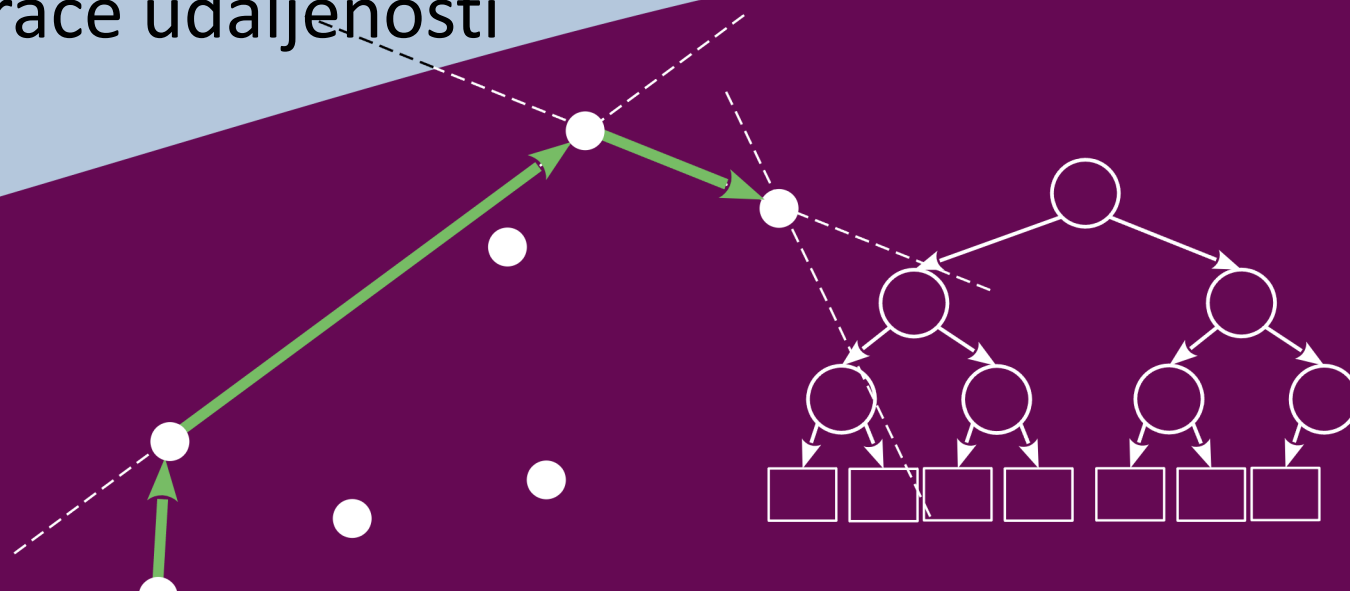
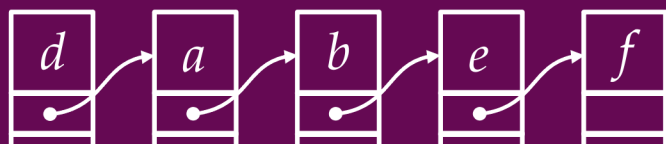
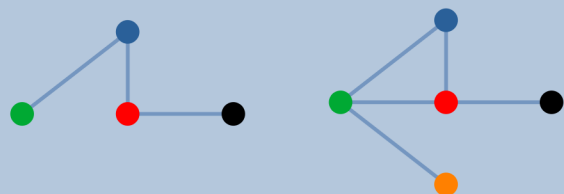


Napredni algoritmi i strukture podataka

Tjedan 9: Algoritmi nad grafovima
Teorija grafova, Najkraće udaljenosti



Osnove teorije grafova

- **Graf** (graph) je uređeni par nepraznog konačnog skupa vrhova V (vertex) i skupa (moguće praznog) bridova E (edge)

$$G = (V, E)$$

- **Brid** (edge) je element produkta skupa V

$$E \subseteq V \times V, e_k = (v_i, v_j) \in E$$

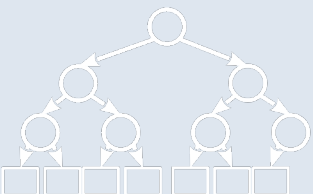
- Svaki brid je uređeni par vrhova $e_k = (v_i, v_j)$
- Brid se kraće može označiti samo $v_i v_j$ ili jednostavno samo e_k , pri čemu je $e_k = v_i v_j$

Alternativna literatura:

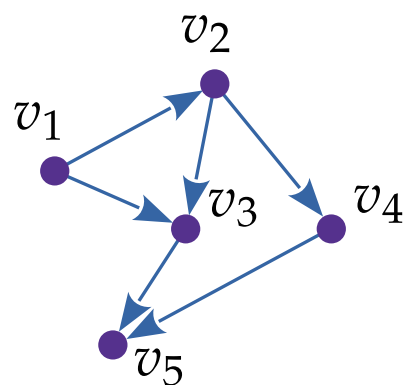
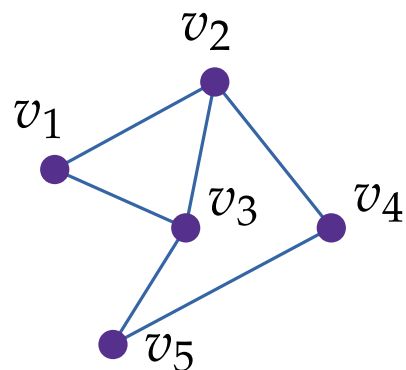
- Thomas H, C., Charles E, L., Ronald L, R., & Clifford, S. (2016). Introduction to Algorithms., Chapter VI

Osnove teorije grafova

- **Red** (order) **grafa** je broj vrhova u njemu ili $|V|$; često pojednostavljeno samo V
 - **Veličina** (size) **grafa** je broj bridova koje graf sadrži $|E|$; pojednostavljeno samo E
 - **Petlja** (loop) je brid koji počinje i završava u istom vrhu $e_k = v_i v_i$
 - **Stupanj** (degree) **vrha** je broj bridova koji su priležeći (incident) tom vrhu (spojeni s njim)
 - Stupanj vrha v označavat ćemo s $\deg(v)$
 - Povratne petlje se u $\deg(v)$ ubrajaju dva puta
- $$\forall v \in V \exists E' \subseteq E \forall (v_i, v_j) \in E': v_i = v \vee v_j = v \Rightarrow \deg(v) = |E'|$$



Teorija grafova – vrste grafova

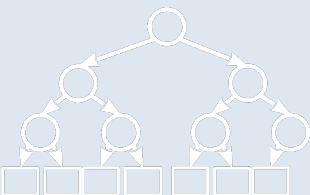


- **Neusmjereni** (undirected) **graf**

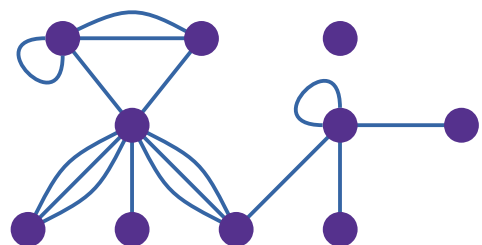
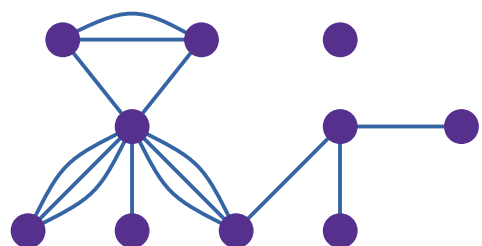
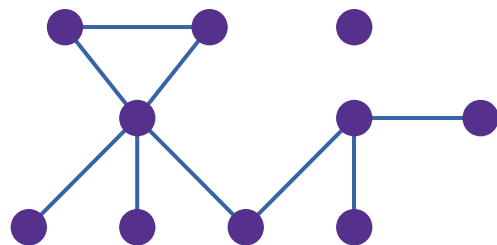
- Za svaki brid $v_i v_j$ vrijedi $v_i v_j = v_j v_i$
- Vrhovi se smatraju susjednima (adjacent) ako je $v_i v_j \in E$
- Takav je brid priležeći (incident) vrhovima v_i i v_j

- **Usmjereni** (directed) **graf**

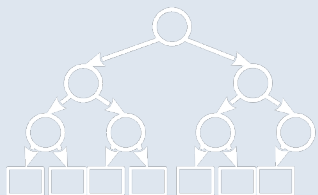
- Za brid $v_i v_j$ ne mora vrijediti $v_i v_j = v_j v_i$
- Vrh v_j se smatra susjedom vrhu v_i samo ako postoji $v_i v_j \in E$
- Brid $v_i v_j$ se naziva izlaznim bridom (outedge) vrha v_i i ulaznim bridom (inedge) vrha v_j



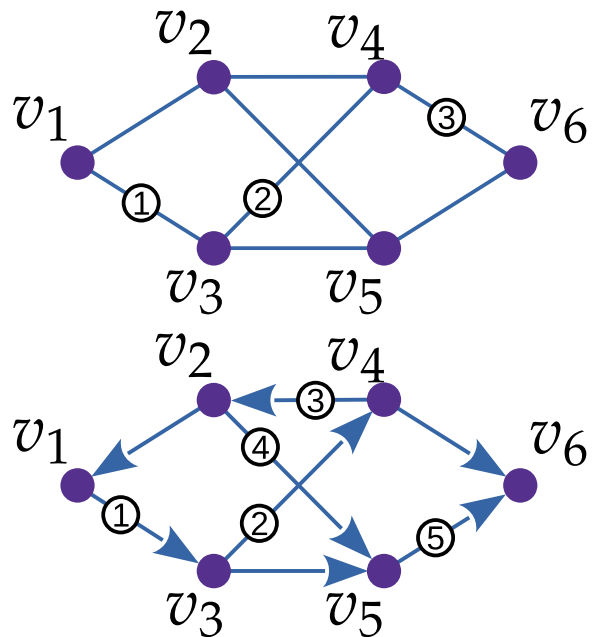
Teorija grafova – vrste grafova



- **Jednostavni graf** (simple graph) je graf (obično neusmjereni) koji između svaka dva vrha ima najviše jedan brid i u kojem nema petlji
- **Multigraf** (multigraph) je graf koji može imati više od jednog brida između dva vrha
- **Pseudograf** (pseudograph) je multigraf u kojem mogu postojati povratne petlje



Teorija grafova – obilazak, putanja, ...



- **Obilazak ili šetnja** (walk) od vrha v_1 do vrha v_n je naizmjenični niz vrhova i bridova

$v_1, e_2, v_2, e_3, \dots, e_n, v_n$

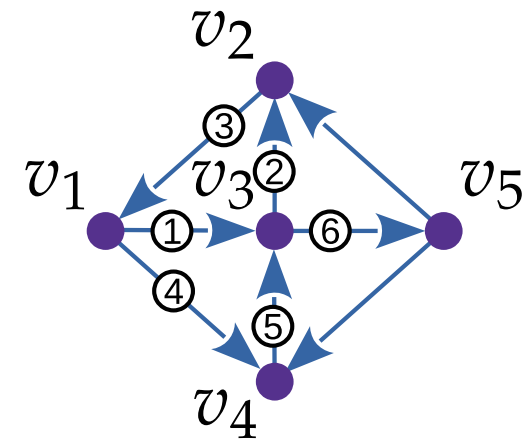
- Kraće se označava v_1, v_2, \dots, v_n ili samo $v_1 v_2 \dots v_n$

- **Putanja** (trail) je obilazak u kojem su svi bridovi različiti (znači svakim bridom se prolazi samo jednom, ali vrhovi se mogu ponavljati)

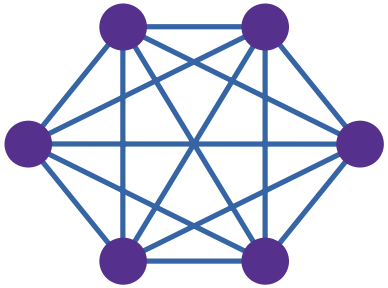
- **Put ili staza** (path) je putanja (trail) u kojoj su svi vrhovi različiti (dakle ne mogu se ponavljati)

- **Krug** (circuit) je putanja (trail) u kojoj je $v_1 = v_n$

- **Ciklus** (cycle) je staza (path) na kojoj je $v_1 = v_n$



Teorija grafova

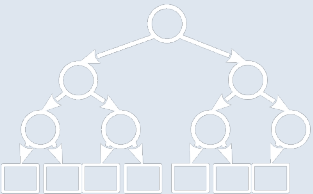


- **Potpuni** (complete) **graf** je graf u kojem je između svaka dva vrha točno jedan brid

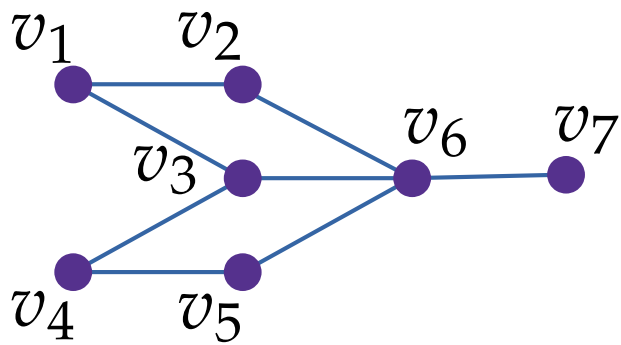
- Odnosi se samo na neusmjerene grafove
- Potpuni graf n -tog reda (n vrhova) označava se s K_n
- Broj bridova u potpunom grafu = broj dvočlanih podskupova skupa vrhova V

$$E(K_n) = \binom{V}{2} = \frac{V!}{(V-2)! \cdot 2!} = \frac{V(V-1)}{2} = O(V^2)$$

- Neka je $V' \subset V$ i $E' \subset E$, tada je $G' = (V', E')$ **podgraf** (subgraph) grafa $G = (V, E)$
 - Ako imamo $V' \subset V$, tada je $G[V']$ inducirani podgraf (induced subgraph) grafa G , koji sadrži sve bridove iz skupa E , a koji spajaju vrhove iz skupa V'



Povezanost grafa – neusmjereni grafovi



- Definiramo binarnu relaciju W^G takvu da obilaskom kroz graf G povezuje vrhove u i v . Ako su vrhovi u i v povezani vrijedi

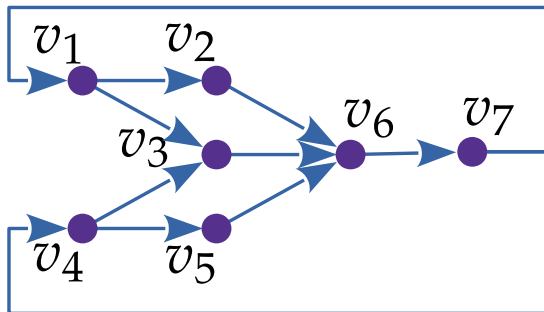
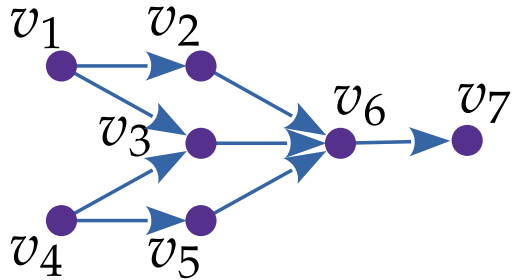
$$uW^G v$$

- Povezani neusmjereni graf** $G = (V, E)$ - neusmjereni graf je povezan (connected) samo ako za sve parove vrhova vrijedi

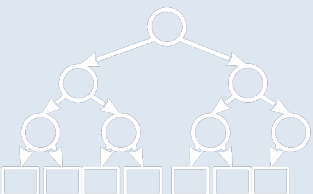
$$\forall u, v \in V: uW^G v \wedge vW^G u$$

- što znači da je vrh v **dohvatljiv** (reachable) od vrha u
- ali i obratno, što je u skladnosti sa definicijom neusmjerenog grafa

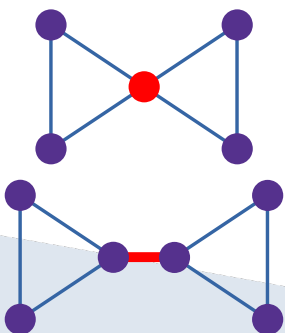
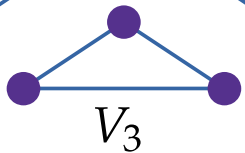
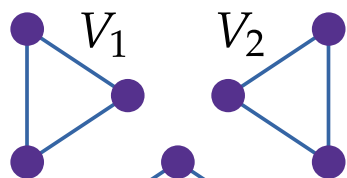
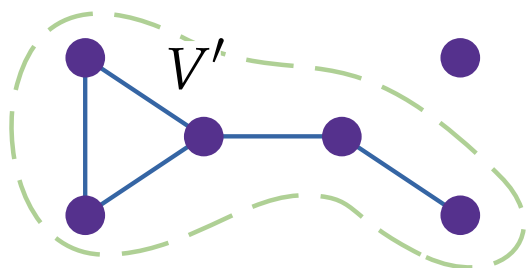
Povezanost grafa – usmjereni grafovi



- **Slabo povezani usmjereni graf G** (weakly connected) – je usmjereni graf čiji je neusmjereni ekvivalent povezan prema definiciji na prethodnoj prikaznici
- **Snažno povezani usmjereni graf G** (strongly connected) – je usmjereni graf kod kojeg su svi parovi vrhova međusobno dohvatljivi
$$\forall u, v \in V: uW^G v \wedge vW^G u$$
 - Primijetimo da je definicija za snažno povezani usmjereni graf identična definiciji za povezan neusmjereni graf
 - Razlika je i više nego očita s obzirom da se kod usmjerenog grafa stvaraju krugovi i ciklusi da bi on bio snažno povezan



Povezanost grafa



- **Nepovezani graf G** (disconnected graph)

- Definiramo podskup vrhova $V' \subset V$ takav da je inducirani podgraf $G[V']$ povezan i ne postoji obilazak između vrhova iz skupa V' i ostatka vrhova $V \setminus V'$

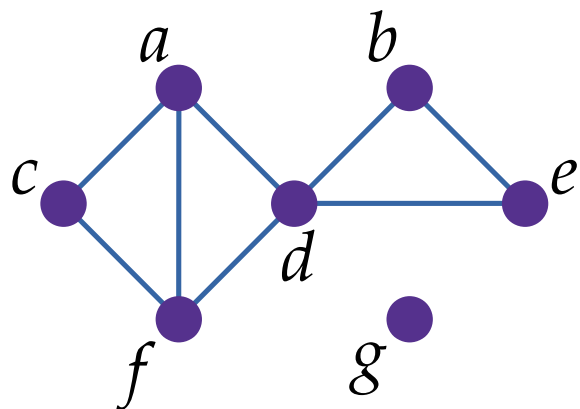
$$\nexists u \in V' \nexists v \in V \setminus V' : u W^G v \vee v W^G u$$

- Takav se graf smatra nepovezanim

- **Prijelomna točka** (articulation point) – je vrh čijim uklanjanjem graf postaje nepovezan

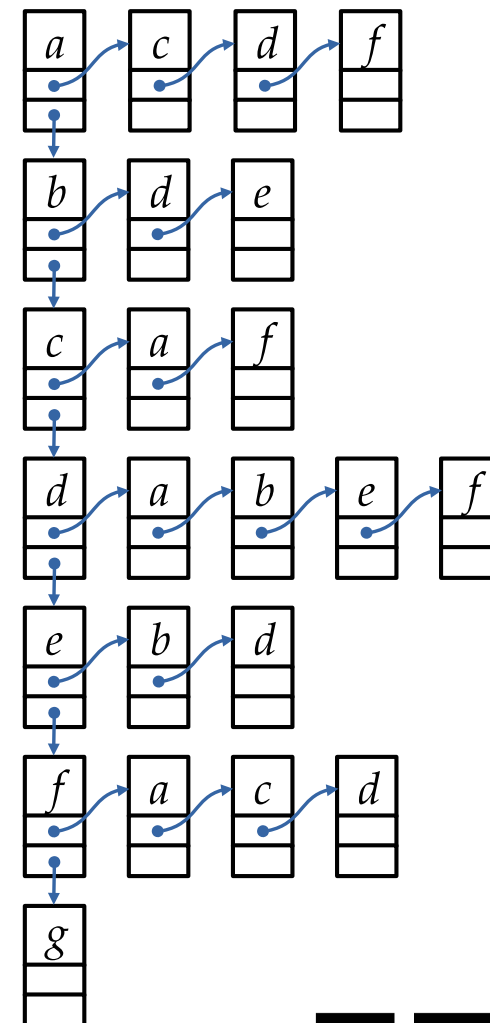
- **Most** (bridge) – je brid čijim uklanjanjem graf postaje nepovezan.

Pohrana grafa

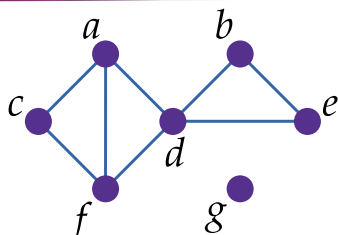


<i>a</i>	<i>c</i>	<i>d</i>	<i>f</i>	
<i>b</i>	<i>d</i>	<i>e</i>		
<i>c</i>	<i>a</i>	<i>f</i>		
<i>d</i>	<i>a</i>	<i>b</i>	<i>e</i>	<i>f</i>
<i>e</i>	<i>b</i>	<i>d</i>		
<i>f</i>	<i>a</i>	<i>c</i>	<i>d</i>	
<i>g</i>				

- Moguća je pohrana pomoću listi i pomoću matrica
- Prednost listi je u pristupu svim susjedima nekog vrha jer je potrebno $\deg(v)$ koraka u odnosu na $|V|$ za matrice
- Prednost matrica je u pojedinačnim intervencijama (dodavanje ili uklanjanje brida) zbog bržeg pristupa i složenosti $O(1)$ prilikom održavanja
- Primjer u obliku tablice (star representation)(lijevo) i u obliku dvodimenzionalne povezane liste (desno)



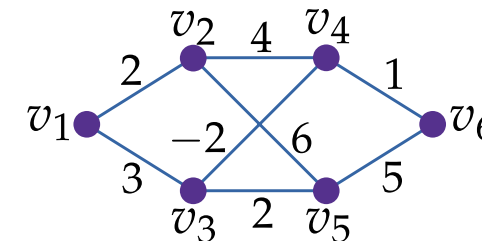
Pohrana grafa



$$\mathbf{A} = \begin{matrix} & \begin{matrix} a & b & c & d & e & f & g \end{matrix} \\ \begin{matrix} a \\ b \\ c \\ d \\ e \\ f \\ g \end{matrix} & \begin{bmatrix} 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{matrix}$$

$$\mathbf{B} = \begin{matrix} & \begin{matrix} ac & ad & af & bd & be & cf & de & df \end{matrix} \\ \begin{matrix} a \\ b \\ c \\ d \\ e \\ f \\ g \end{matrix} & \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{matrix}$$

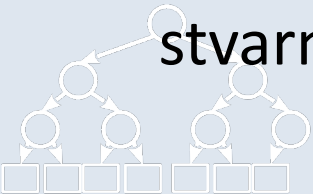
- Primjer u obliku matrice susjedstva (adjacency matrix)(lijevo-sredina)
 - Simetrična za neusmjerene grafove
- U obliku matrice incidencije (incidency matrix)(lijevo-dolje)
- U obliku težinske matrice susjedstva (weighted adjacency matrix)(desno)



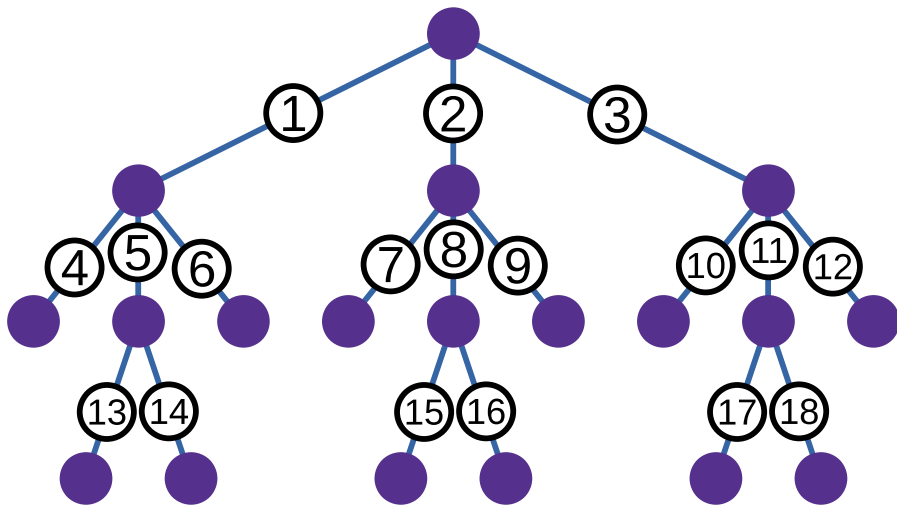
$$\mathbf{W} = \begin{matrix} & \begin{matrix} v_1 & v_2 & v_3 & v_4 & v_5 & v_6 \end{matrix} \\ \begin{matrix} v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \\ v_6 \end{matrix} & \begin{bmatrix} 0 & 2 & 3 & 0 & 0 & 0 \\ 2 & 0 & 0 & 4 & 6 & 0 \\ 3 & 0 & 0 & -2 & 2 & 0 \\ 0 & 4 & -2 & 0 & 0 & 1 \\ 0 & 6 & 2 & 0 & 0 & 5 \\ 0 & 0 & 0 & 1 & 5 & 0 \end{bmatrix} \end{matrix}$$

Obilazak grafa (graph traversal)

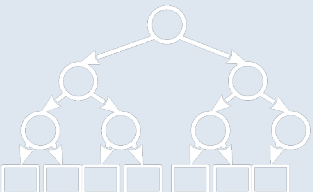
- Algoritmi za obilazak **stabla** nisu zadovoljavajući za općenite grafove jer:
 - Graf može imati cikluse pa bi se algoritam za stablo mogao naći u beskonačnoj petlji
 - Graf može imati odvojene i nepovezane vrhove, pa riskiramo da ne pronađemo sve povezane particije vrhova $P(V)$
- Dva najpoznatija algoritma za obilazak grafa su:
 - Obilazak (prvo) u širinu (*Breadth First Search*; BFS)
 - Obilazak (prvo) u dubinu (*Depth First Search*; DFS)
- Osim u jednostavnim primjenama (npr. obilazak grafa, detekcija ciklusa, provjera povezanosti pojedinih vrhova), DFS i BFS nisu međusobno zamjenjivi zbog bitne logičke različitosti
- Oba su temelj za brojne složenije algoritme i teorijski su podjednako brzi, ali u stvarnosti je DFS nešto sporiji jer je rekurzivan



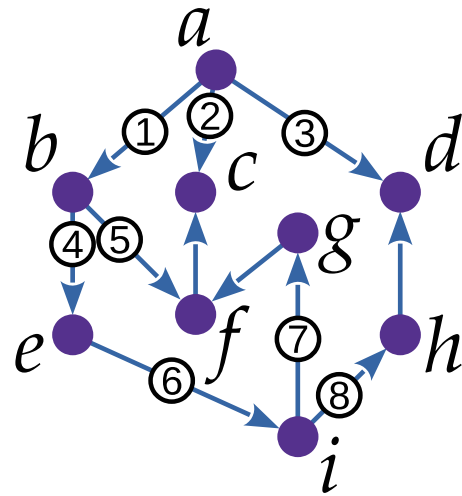
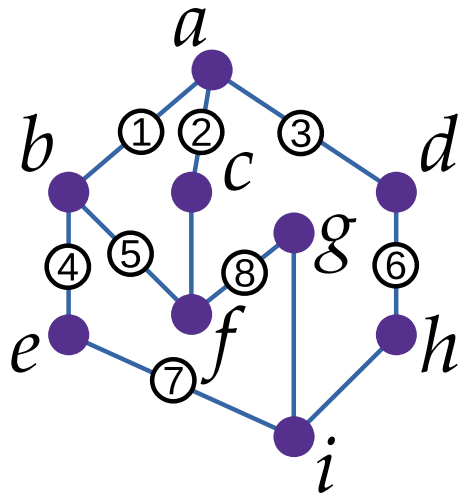
Obilazak grafa (prvo) u širinu - BFS



- Koncept BFS algoritma
 - Inicijalno u listi imamo samo korijenski vrh – lista je prirodna podatkovna struktura za BFS algoritam
 - Uzimamo prvi vrh iz liste kao trenutni vrh n
 - Prvo obilazimo SVE susjedne vrhove trenutnog vrha n
 - Kako obilazimo susjedne vrhove, stavljamo ih u listu
 - Nakon što smo obišli sve susjedne vrhove, uzimamo sljedeći vrh iz liste kao trenutni...
 - Taj postupak ponavljamo do dok nismo obišli sve vrhove u grafu
- Ukoliko imamo nepovezane podgrafove, postupak se ponavlja dok ima neobiđenih vrhova



Obilazak grafa (prvo) u širinu - BFS

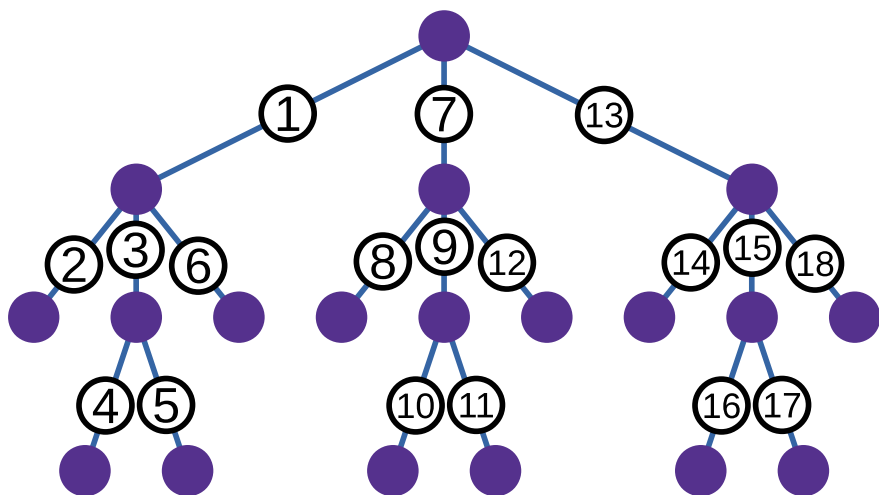


Step	u	queue Q
0	a	a
1	a	bcd
2	b	$cdef$
3	c	def
4	d	efh
5	e	fhi
6	f	hig
7	h	ig
8	i	g
9	g	

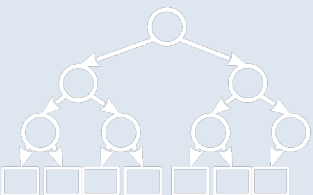
```

procedure BFS( $G$ )
  initialize all vertices in  $G$  as not visited
   $Q \leftarrow$  empty queue
  while there is an unvisited vertex  $u_0$  in  $G$  do
    mark  $u_0$  as visited
     $Q.enqueue(u_0)$ 
    while  $Q$  is not empty do
       $u \leftarrow Q.dequeue$ 
      process  $u$ 
      for  $v$  in adjacent vertices of  $u$  do
        if  $v$  is not visited then
          mark  $v$  as visited
           $Q.enqueue(v)$ 
    
```

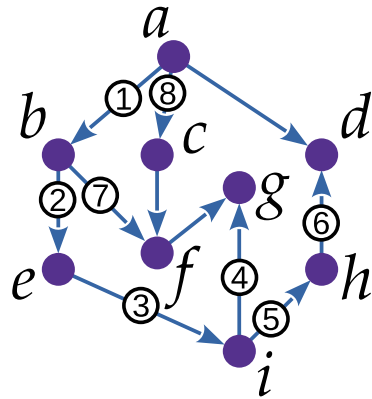
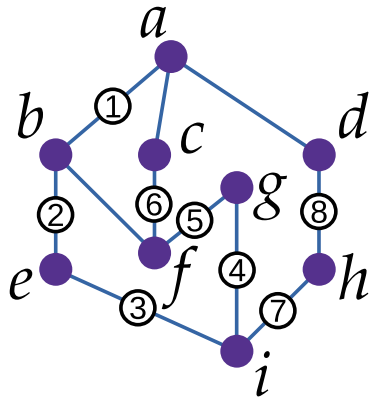
Obilazak grafa (prvo) u dubinu - DFS



- Koncept DFS algoritma - rekurzivno
 - Uzimamo prvi neobiđeni vrh
 - Rekurzivno se spuštamo na prvog susjeda
 - Rekurzija se ponavlja, sve do dok ne dođemo do lista, zatim se vraćamo natrag do prvog vrha na kojem imamo još neobiđenih susjeda, čim opet započinjemo rekurzivno spuštanje do listova
 - Stog je prirodna podatkovna struktura za DFS algoritam
- Ukoliko imamo nepovezane podgrafove, postupak se ponavlja dok ima neobiđenih vrhova



Obilazak grafa (prvo) u dubinu - DFS



Step	<i>u</i>	visited	stack <i>S</i>
0			<i>a</i>
1	<i>a</i>	No	<i>bcd</i>
2	<i>b</i>	No	<i>efcd</i>
3	<i>e</i>	No	<i>ifcd</i>
4	<i>i</i>	No	<i>ghfcd</i>
5	<i>g</i>	No	<i>fhfcd</i>
6	<i>f</i>	No	<i>chfcd</i>
7	<i>c</i>	No	<i>hfgcd</i>
8	<i>h</i>	No	<i>dfcd</i>
9	<i>d</i>	No	<i>fcd</i>
10	<i>f</i>	Yes	<i>cd</i>
11	<i>c</i>	Yes	<i>d</i>
12	<i>d</i>	Yes	

```

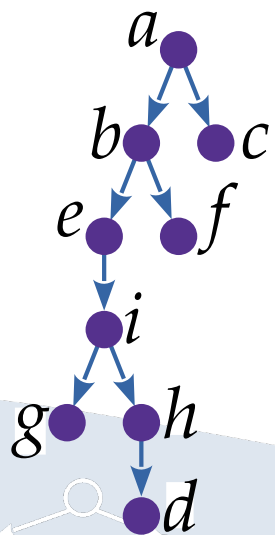
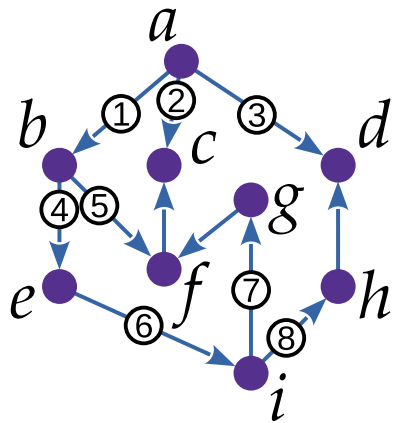
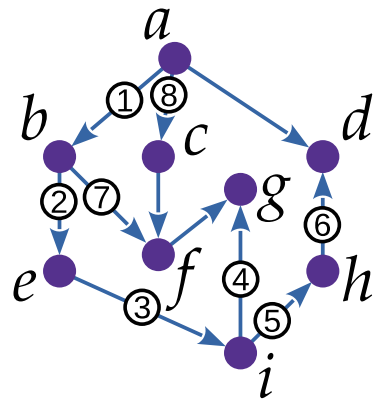
procedure DFS(G)
  initialize all vertices in G as not visited
  S ← empty stack
  while there is an unvisited vertex u0 in G do
    S.push(u0)
    while S is not empty do
      u ← S.pop
      if u is not visited then
        mark u as visited
        process u
        for v in reversed list of adjacent vertices of u do
          if v is not visited then
            S.push(v)
    
```

```

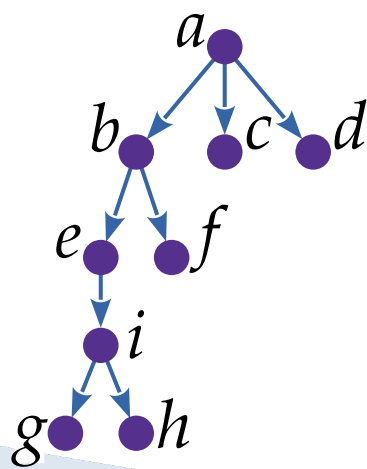
procedure DFS(G)
  initialize all vertices in G as not visited
  while there is an unvisited vertex u0 in G do
    DFS_recursive(G, u0)

procedure DFS_recursive(G, u)
  mark u as visited
  process u
  for v in adjacent vertices of u do
    if v is not visited then
      DFS_recursive(G, v)
    
```

Razapinjujuće stablo (spanning tree)



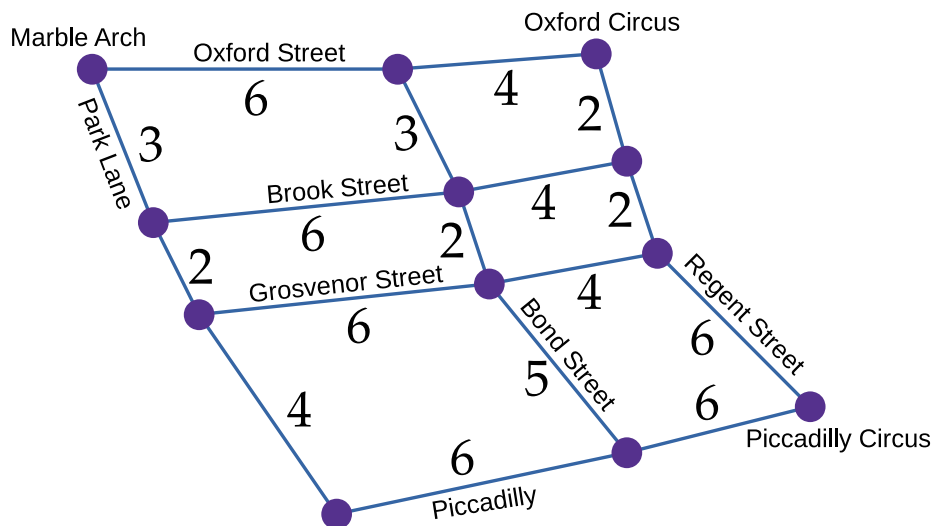
DFS



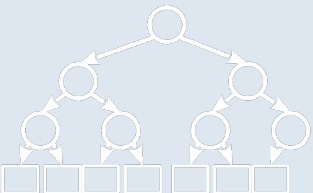
BFS

- Rezultanta putanja oba algoritma za obilazak grafa (BFS i DFS) čini stablo u kojem su svi vrhovi grafa. Takvo se stablo naziva **razapinjujuće stablo** (spanning tree).
 - Bilježimo samo bridove koji predstavljaju **napredovanje** BFS i DFS algoritama prema neobiđenim vrhovima – **unaprjedni bridovi** (forward edges)
 - Ne bilježimo bridove kojima se algoritmi vraćaju u vrhove koji su već obiđeni – **povratni bridovi** (back edges)

Najkraći putevi u grafu (shortest paths)

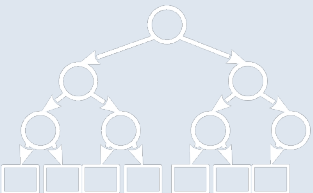


- Temeljni algoritmi za brojne primjene: transport, komunikacije, distribucijske energetske mreže, projektiranje integriranih elektroničkih sklopova i drugo...
- Bridovi, kao apstrakcija (model) poveznica između dvaju čimbenika nekog sustava, dobivaju “težine”; oznaka $w(u, v)$
- Važan je i pojam međuvrh; to je vrh na putu između neka dva vrha u grafu, dakle ni polazni ni završni
- Labela (label) = udaljenost vrha od nekog referentnog vrha (točke)
 - Najčešće se pohranjuje kao članska varijabla strukture ‘vrh’
- Dvije osnovne grupe algoritama:
 - Algoritmi koji pronalaze najkraći put između dva određena vrha
 - Algoritmi koji pronalaze najkraće puteve između svih vrhova u grafu (all-to-all)

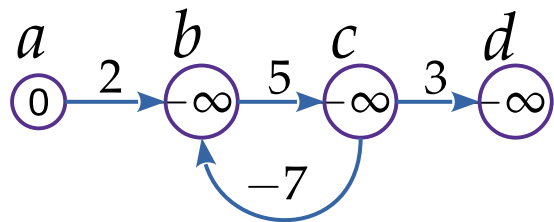
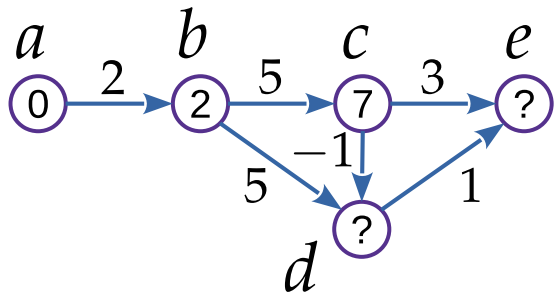


Najkraći putevi u grafu

- Odabir algoritma za traženje najkraćeg puta ovisi o težinama bridova u grafu; dvije su osnovne skupine tih algoritama:
 - **Label-setting** algoritmi – jednom upisana labela se više ne mijenja
 - Tako upisana labela se više ne provjerava
 - Funkcioniraju za grafove s isključivo pozitivnim težinama bridova – Dijkstrin algoritam
 - **Label-correcting** algoritmi – sve labele se mogu mijenjati sve do završetka postupka (konvergencije)
 - Funkcioniraju za grafove s bilo kakvim težinama bridova – Bellman-Ford algoritam



Najkraći putevi u grafu – negativne težine



Iteration	$d(b)$	$d(c)$
1	2	7
2	0	5
3	-2	3
4	-4	1
5	-6	-1
...
∞	$-\infty$	$-\infty$

- Negativna težina predstavlja problem za **label-setting** algoritme – uzmimo primjer na slici
 - Ako smo do vrha d došli obilaskom abd , tada je njegova labela $d(d) = 7$
 - Ako smo do vrha d došli obilaskom $abcd$, tada je njegova labela $d(d) = 6$
- Drugi problem predstavlja negativni ciklus, koji predstavlja problem i za **label-correcting** algoritme
 - Algoritam se uplete u beskonačnu petlju ažuriranja labela
 - U trenutku kada bi algoritam trebao konvergirati, on ne konvergira nego nastavlja ažurirati labela
 - Takav problem je nerješiv

Dijkstrin algoritam

- Spada u algoritme koji računaju najkraću udaljenost između dva vrha v_1 i v_n
- Label-setting algoritam
 - Jednom ažurirana labela vrha se više ne mijenja, osim kroz drugu putanju
 - Nije u mogućnosti raditi s negativnim težinama bridova
- Za neku putanju

$$p = v_1 v_2 \dots v_n$$

- može se primijeniti koncept aditivnosti udaljenosti

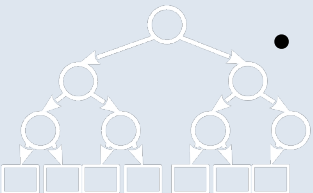
$$d(v_1, v_n) = d(v_1, v_i) + d(v_i, v_n)$$

- Ako je putanja p ujedno i najkraća, tada vrijedi

$$d_{min}(v_1, v_n) = d_{min}(v_1, v_i) + d_{min}(v_i, v_n)$$

- Gledajući unatrag, ako svaki vrh na najkraćem putu “zna” (barem) svojeg neposrednog prethodnika, može se rekonstruirati cijeli put do polaznog vrha.

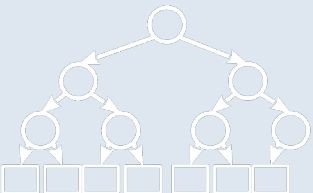
- Ideja je računati $d(v_1, v_n) = d(v_1, v_{j-1}) + d(v_{j-1}, v_j)$, od od $j = n$ do $j = 2$



Dijkstrin algoritam

```
procedure DIJKSTRA( $G, source, destination$ )  
  for each vertex  $v$  in  $V(G)$  do  
     $d(v) \leftarrow \infty$   
     $predecessor(v) \leftarrow null$   
   $d(source) \leftarrow 0$   
   $work \leftarrow V(G)$   
  while  $work$  is not empty do  
     $u \leftarrow$  take a vertex from  $work$  having minimal  $d(u)$   
    if  $u = destination$  then  
      return  
    for vertices  $v$  adjacent to  $u$  and in  $work$  do  
       $temp \leftarrow d(u) + w(uv)$   
      if  $temp < d(v)$  then  
         $d(v) \leftarrow temp$   
         $predecessor(v) \leftarrow u$ 
```

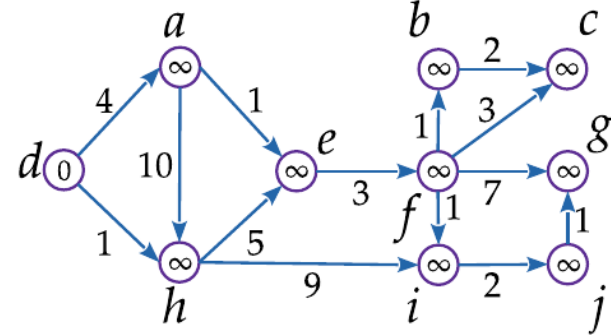
- Inicijaliziramo labele svih vrhova osim početnoga na ∞ , te prethodnike na *null*. Početni vrh inicijaliziramo na 0.
- Krenemo od početnog vrha, te računamo i ažuriramo udaljenosti prema svim susjedima
- Prilikom obilaska
 - Ako je udaljenost susjeda manja od trenutne, tada ažuriramo labelu i tom susjedu stavljamo novog prethodnika
- Obilazak se temelji na BFS algoritmu koji je prioritiziran udaljenošću – manje udaljeni čvorovi idu prvi
- Završavamo u završnom vrhu



Dijkstrin algoritam - primjer

Iteration 0: The input graph G and initialization

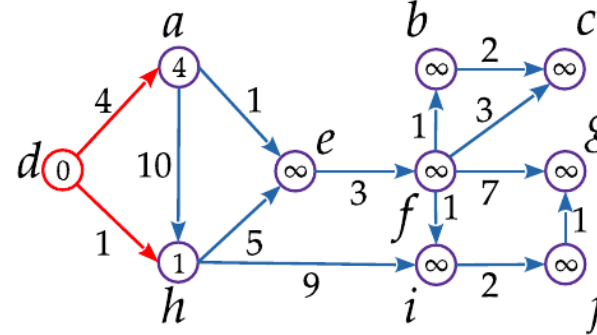
$work = \{a, b, c, d, e, f, g, h, i, j\}$



Iteration 1: $u = d$.

$work = \{a, b, c, e, f, g, h, i, j\}$

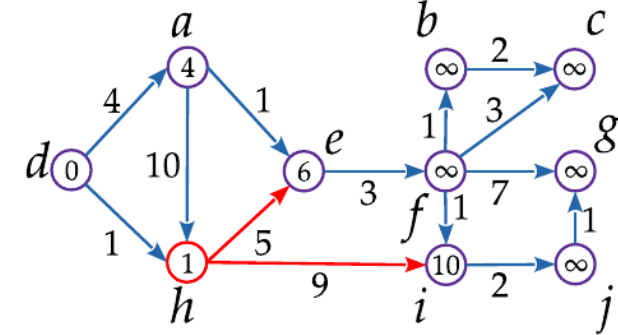
$d(a) = 4, d(h) = 1$



Iteration 2: $u = h$.

$work = \{a, b, c, e, f, g, i, j\}$

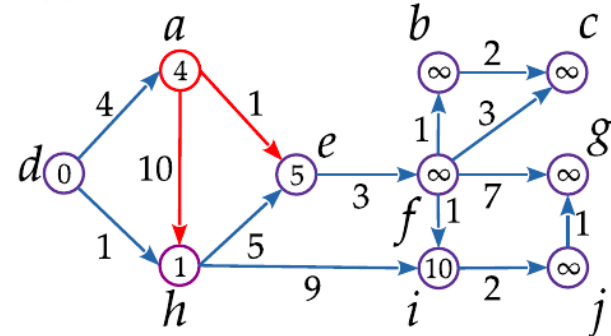
$d(e) = 6, d(i) = 10$



Iteration 3: $u = a$

$work = \{b, c, e, f, g, i, j\}$

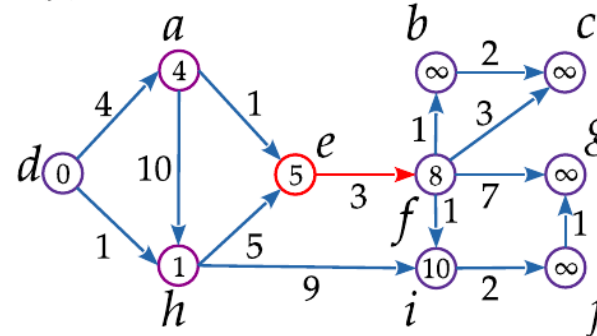
$d(e) = 5$



Iteration 4: $u = e$.

$work = \{b, c, f, g, i, j\}$

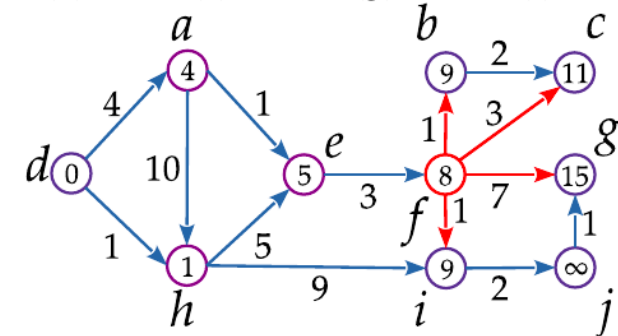
$d(f) = 8$



Iteration 5: $u = f$.

$work = \{b, c, g, i, j\}$

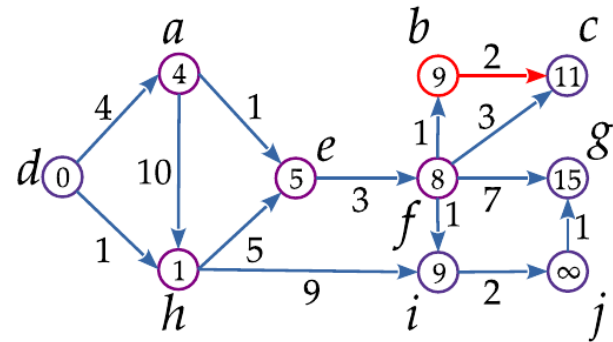
$d(b) = 9, d(c) = 11, d(g) = 15, d(i) = 9$



Dijkstrin algoritam - primjer

Iteration 6: $u = b$

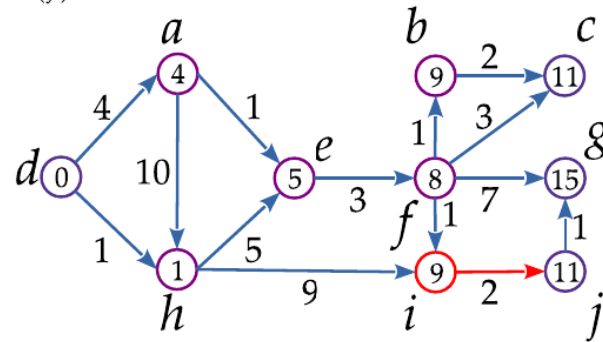
$work = \{c, g, i, j\}$



Iteration 7: $u = i$

$work = \{c, g, j\}$

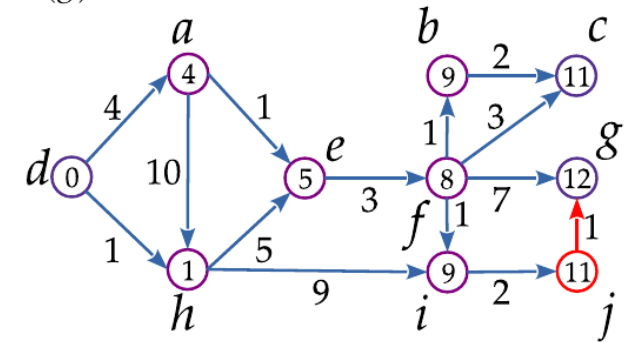
$d(j) = 11$



Iteration 8: $u = j$

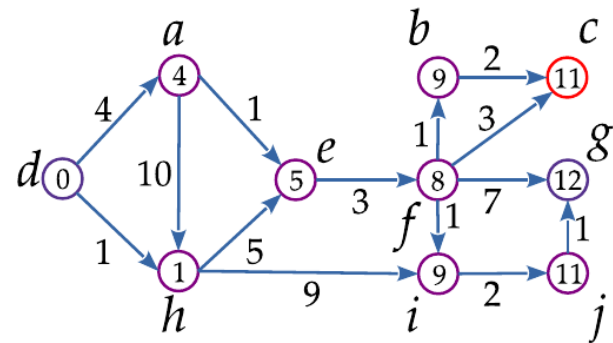
$work = \{c, g\}$

$d(g) = 12$



Iteration 9: $u = c$

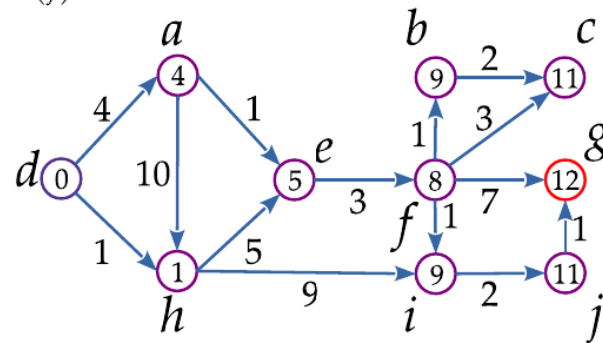
$work = \{g\}$



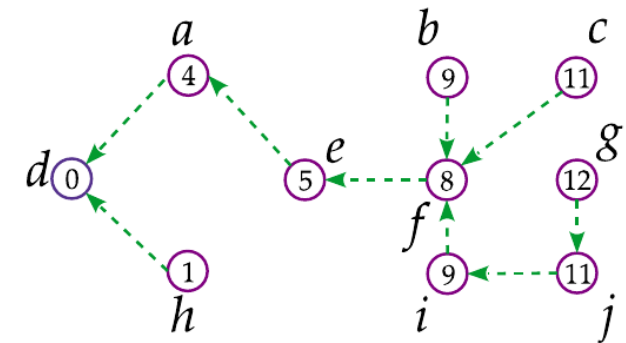
Iteration 10: $u = g$

$work = \{\}$

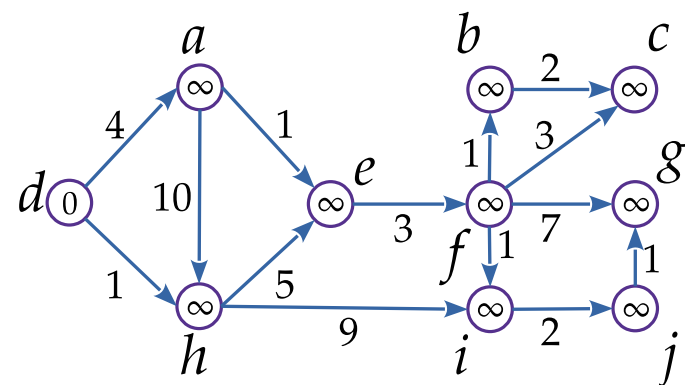
$d(j) = 11$



Final predecessors in the graph G.



Dijkstrin algoritam - primjer

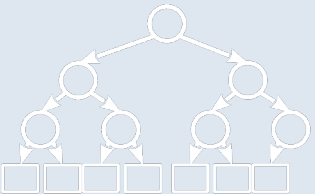


$g \rightarrow j \rightarrow i \rightarrow f \rightarrow e \rightarrow a \rightarrow d$

iteration	0	1	2	3	4	5	6	7	8	9	10
current vertex		<i>d</i>	<i>h</i>	<i>a</i>	<i>e</i>	<i>f</i>	<i>b</i>	<i>i</i>	<i>j</i>	<i>c</i>	<i>g</i>
<i>a</i>	∞	4/ <i>d</i>									
<i>b</i>	∞	∞	∞	∞	∞	9/ <i>f</i>					
<i>c</i>	∞	∞	∞	∞	∞	11/ <i>f</i>					
<i>d</i>	0										
<i>e</i>	∞	∞	6/ <i>h</i>	5/ <i>a</i>							
<i>f</i>	∞	∞	∞	∞	8/ <i>e</i>						
<i>g</i>	∞	∞	∞	∞	∞	15/ <i>f</i>	15/ <i>f</i>	15/ <i>f</i>	12/ <i>j</i>		
<i>h</i>	∞	1/ <i>d</i>									
<i>i</i>	∞	∞	10/ <i>h</i>	10/ <i>h</i>	10/ <i>h</i>	9/ <i>f</i>					
<i>j</i>	∞	∞	∞	∞	∞	∞	∞	11/ <i>i</i>			

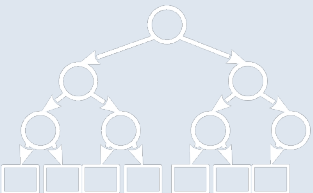
Dijkstrin algoritam - zaključak

- Kompleksnost algoritma ako se najbliži vrh iz *work* liste određuje sekvencijalno – $O(V^2)$
- Ukoliko se *work* lista implementira kao Fibonacci-eva gomila (Fibonacci heap), tada kompleksnost pada na $O(E + V \log_2 V)$

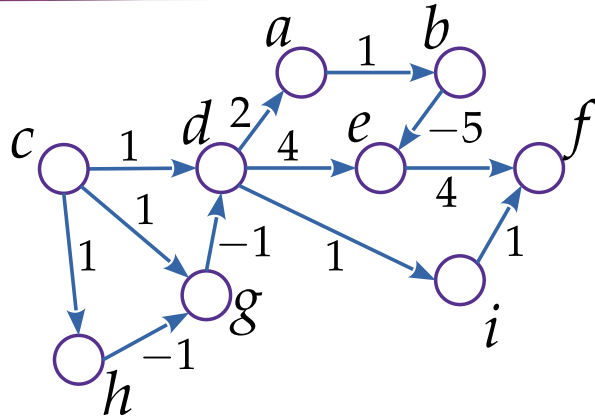


Bellman-Ford algoritam

- Spada u algoritme koji računaju najkraću udaljenost između početnog vrha i svih ostalih vrhova
- Label-correcting algoritam
 - Sve labele se ažuriraju do trenutka konvergencije – do dok više nema daljnjeg ažuriranja labela
 - Može raditi s negativnim težinama u grafu
 - Nije u mogućnosti raditi s negativnim ciklusima
- Sporiji od Dijkstrinovog algoritma
- Radi s bridovima
 - Provjerava sve bridove u grafu i po njima ažurira udaljenosti vrhova
 - Konvergencija je postavljena na:
 - Dok više nema ažuriranja labela na vrhovima
 - Programski limit je $|V| - 1$ iteracija kroz sve bridove grafa



Bellman-Ford algoritam



```
procedure BELLMAN-FORD( $G, source$ )
```

```
  for each vertex  $v$  in  $V(G)$  do
```

```
     $d(v) \leftarrow \infty$ 
```

```
    predecessor( $v$ )  $\leftarrow null$ 
```

```
   $d(source) \leftarrow 0$ 
```

```
  loop  $|V(G)| - 1$  times
```

```
    for each edge  $uv \in E(G)$  do
```

```
      if  $d(u) + w(uv) < d(v)$  then
```

```
         $d(v) \leftarrow d(u) + w(uv)$ 
```

```
        predecessor( $v$ )  $\leftarrow u$ 
```

```
  for each edge  $uv \in E(G)$  do
```

```
    if  $d(u) + w(uv) < d(v)$  then
```

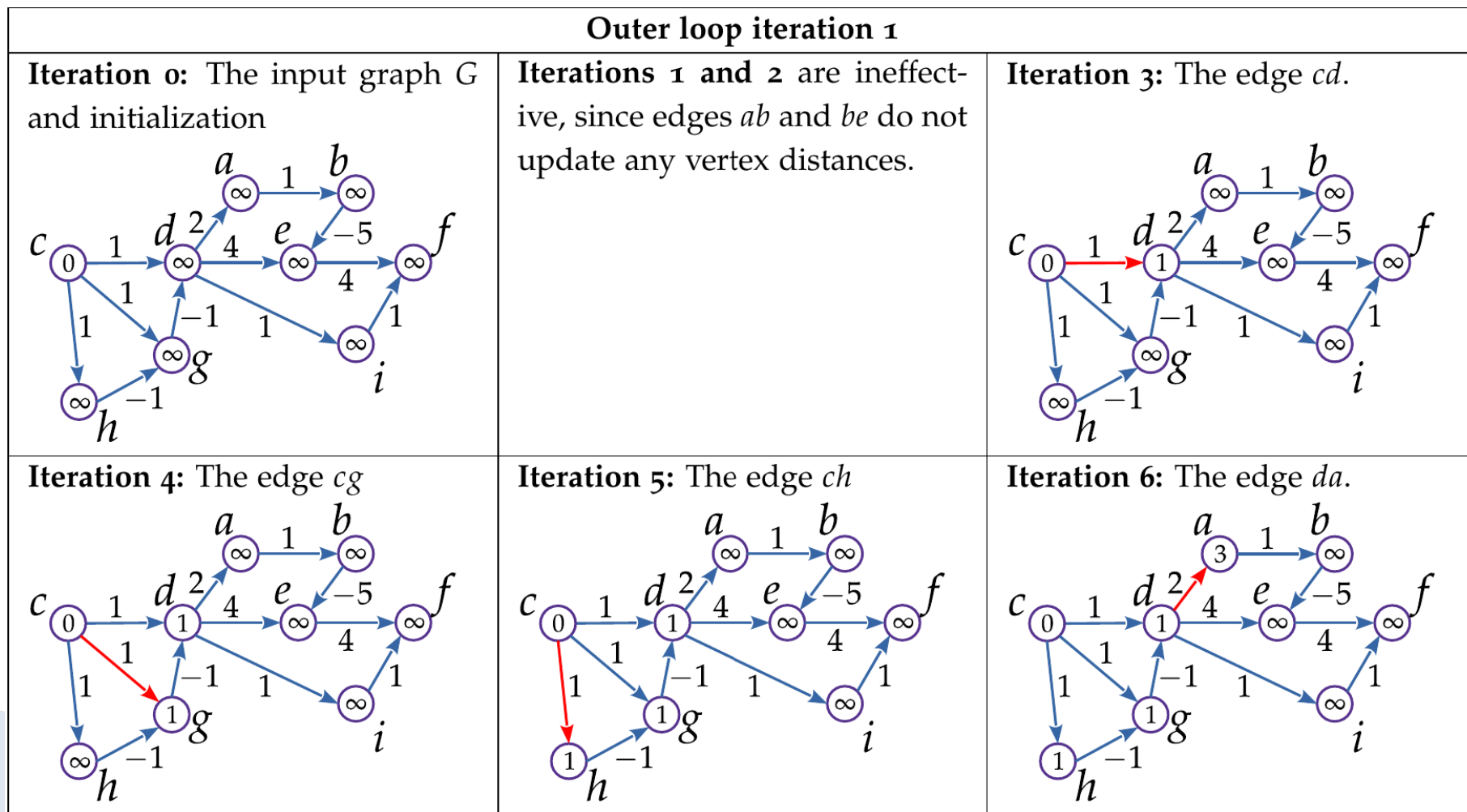
```
      raise exception 'negative cycle has been detected'
```

- Sortiramo listu bridova grafa na neki način, na primjer
 $E(G) = \{ab, be, cd, cg, ch, da, de, di, ef, gd, hg, if\}$
- Prolazimo kroz bridove u $E(G)$. Za brid uv ažuriramo udaljenost vrha v ako imamo

$$d(u) + w(uv) < d(v)$$

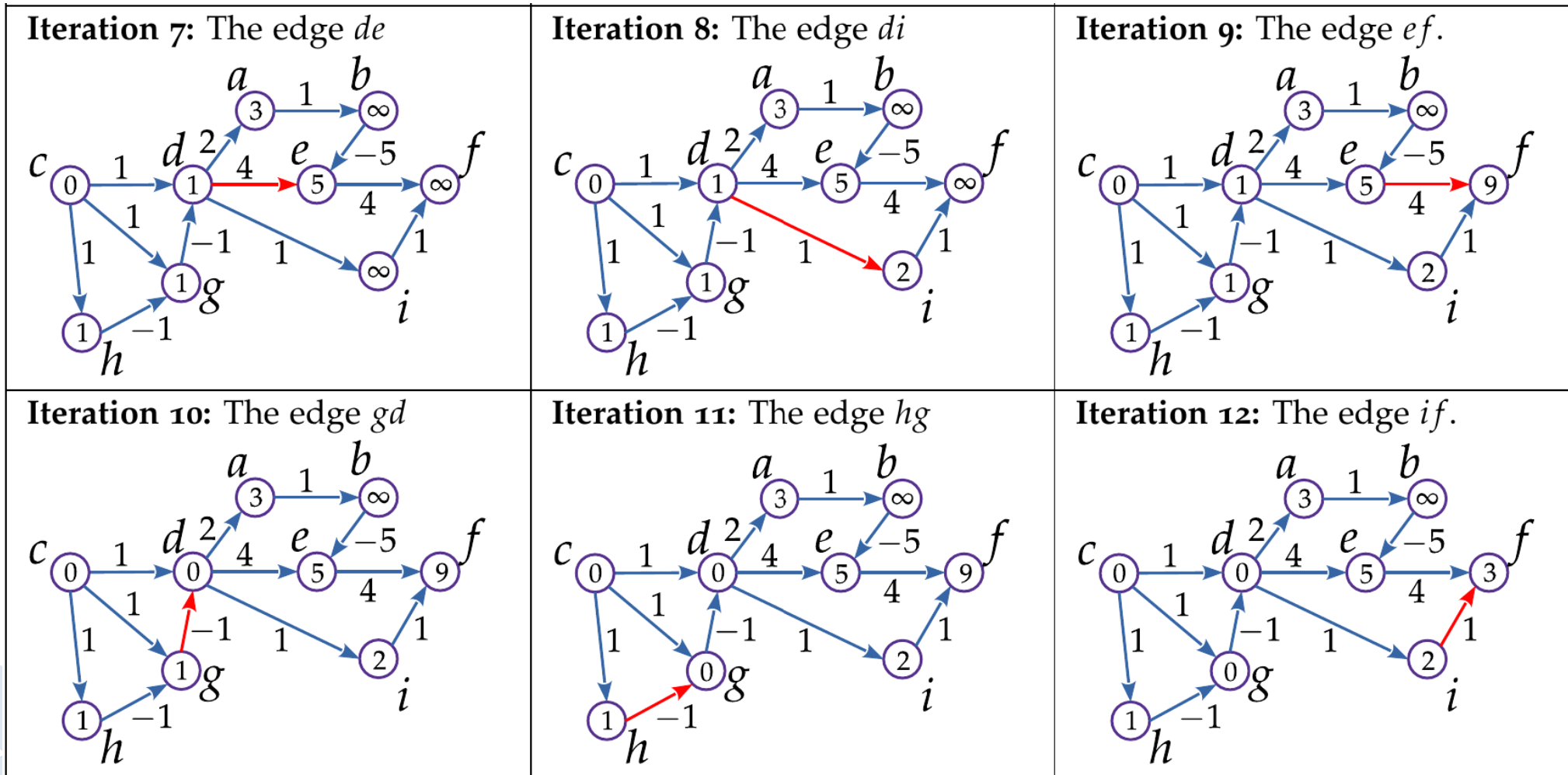
- To ponavljamo maksimalno $|V| - 1$ puta
- Na kraju prođemo još jednom kroz sve bridove, pa ako još uvijek imamo vrh čiju labelu možemo ažurirati – negativni ciklus

Bellman-Ford algoritam - primjer



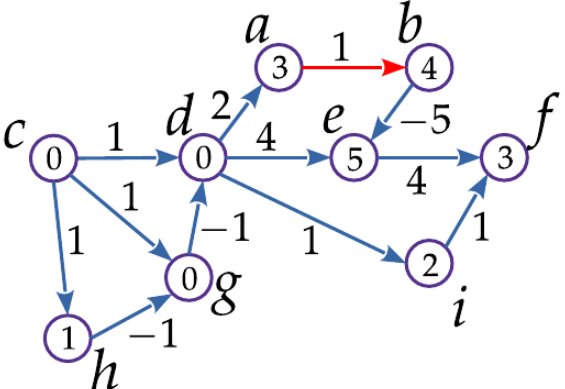
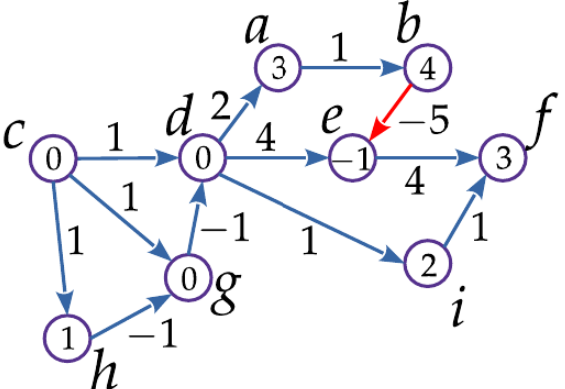
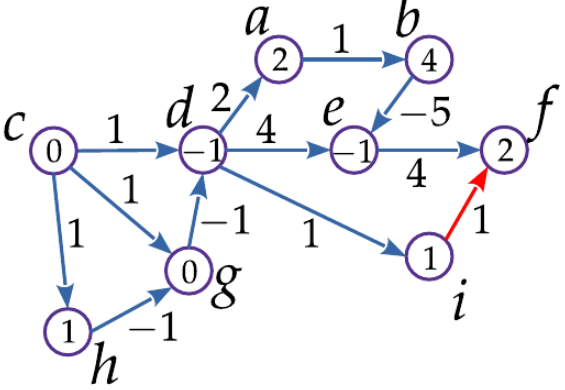
$$E(G) = \{ab, be, cd, cg, ch, da, de, di, ef, gd, hg, if\}$$

Bellman-Ford algoritam - primjer



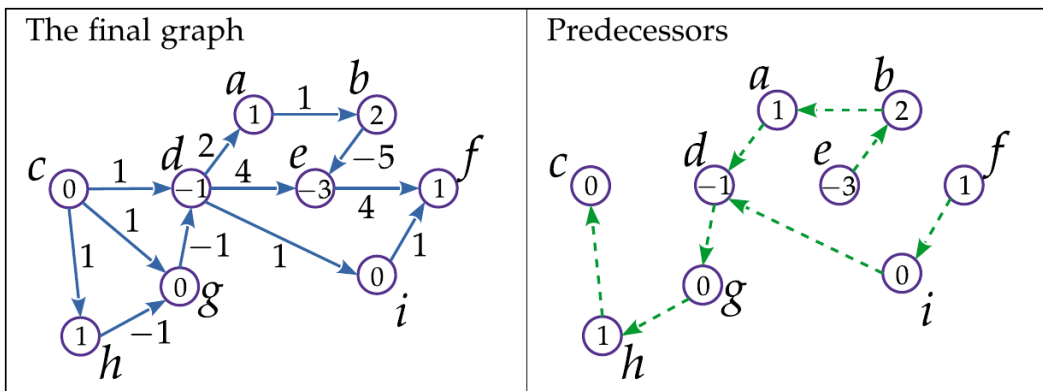
$$E(G) = \{ab, be, cd, cg, ch, da, de, di, ef, gd, hg, if\}$$

Bellman-Ford algoritam - primjer

Outer loop iteration 2		
<p>Iteration 1: The edge ab</p> 	<p>Iteration 2: The edge be</p> 	<p>We skip a number of the inner edge iterations here.</p>
<p>Iteration 12: The edge if</p> 		

$$E(G) = \{ab, be, cd, cg, ch, da, de, di, ef, gd, hg, if\}$$

Bellman-Ford algoritam - primjer



	Iteration					
	0	1		2	3	4
a	∞	3		2	1	
b	∞			4	3	2
c	0					
d	∞	1	0	-1	-1	
e	∞	5		-1	-2	-3
f	∞	9	3	2	1	
g	∞	1	0			
h	∞	1				
i	∞	2		1	0	

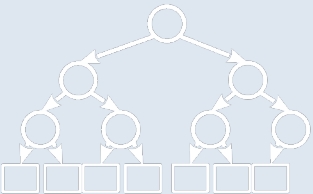
- Nakon $|V| - 1$ iteracija dobivamo konačni rezultat
 - Desni graf predstavlja prethodnike, a time i najkraće putanje od c do svih ostalih vrhova
- Primjer rješavanja kroz tablicu
 - Pratimo $E(G)$ i ažuriramo udaljenosti u koloni
- Kompleksnost Bellman-Ford algoritma je $O(E * V)$. Vanjska petlja iterira kroz vrhove, dok unutarnja iterira kroz bridove

$E(G) = \{ab, be, cd, cg, ch, da, de, di, ef, gd, hg, if\}$

Bellman-Ford algoritam – brža inačica

```
procedure BELLMAN-FORD( $G, source$ )  
  for each vertex  $v$  in  $V(G)$  do  
     $d(v) \leftarrow \infty$   
     $predecessor(v) \leftarrow null$   
  
   $d(source) \leftarrow 0$   
   $Q \leftarrow$  empty queue  
   $Q.enqueue(source)$   
  while  $Q$  is not empty do  
     $u \leftarrow Q.dequeue$   
    for  $v$  in adjacent vertices of  $u$  do  
      if  $d(u) + w(uv) < d(v)$  then  
         $d(v) \leftarrow d(u) + w(uv)$   
         $predecessor(v) \leftarrow u$   
        if  $v$  not in  $Q$  then  
           $Q.enqueue(v)$ 
```

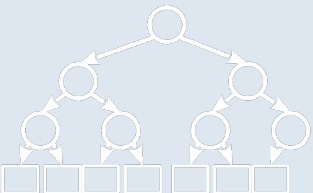
- Osnovna razlika je u tome što ne obrađujemo sve bridove
- U listu se stavlja početni vrh, a zatim samo vrhovi (susjedi) čija se labela promijenila
- Što znači da se obrađuju samo podgrafovi gdje će potencijalno doći do neke promjene labele – može se desiti ako imamo krugove i cikluse u grafu
- Kompleksnost je i dalje $O(E * V)$



Warshall-Floyd-Ingerman algoritam

- Spada u algoritme koji računaju najkraću udaljenost između svih vrhova grafa (all-to-all)
- Label-correcting algoritam
 - Sve labele se ažuriraju do kraja rada algoritma
 - Može raditi s negativnim težinama u grafu
 - Nije u mogućnosti raditi s negativnim ciklusima
- Radi s matricama udaljenosti (distance matrix)
- Zamislamo neki skup vrhova $V = \{a, b, c, d, e\}$
- Mapiramo vrhove iz V tako da ih označimo rednim brojevima

$$\forall x \in V: v_i = x, 1 \leq i \leq |V|$$
$$v_1 = a, v_2 = b, v_3 = c, v_4 = d, v_5 = e$$

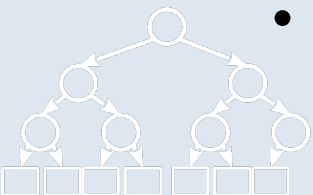


Warshall-Floyd-Ingerman algoritam

- Matrica udaljenosti za prethodni skup vrhova V izgleda kao

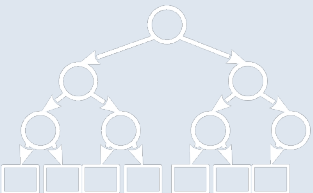
$$\mathbf{D} = \begin{array}{c} v_1 = a \\ v_2 = b \\ v_3 = c \\ v_4 = d \\ v_5 = e \end{array} \begin{array}{ccccc} v_1 = a & v_2 = b & v_3 = c & v_4 = d & v_5 = e \\ \left[\begin{array}{ccccc} d_{11} & d_{12} & d_{13} & d_{14} & d_{15} \\ d_{21} & d_{22} & d_{23} & d_{24} & d_{25} \\ d_{31} & d_{32} & d_{33} & d_{34} & d_{35} \\ d_{41} & d_{42} & d_{43} & d_{44} & d_{45} \\ d_{51} & d_{52} & d_{53} & d_{54} & d_{55} \end{array} \right] \end{array}$$

- udaljenost između vrhova v_i i v_j označava se kao d_{ij}
- za očekivati je da će se naći barem jedna putanja kroz graf između vrhova v_i i v_j - ovo nije nužno ako graf nije povezan, ali pretpostavimo da jest
- treba odabrati onu putanju koja je najkraća ili $d_{\min}(v_i, v_j)$



Warshall-Floyd-Ingerman algoritam

- Ako imamo novu putanju koja prolazi međuvrhom v_k
$$p = v_1 \dots v_k \dots v_n$$
 - smatra se da je putanja p kraća ako vrijedi $d(v_1, v_k) + d(v_k, v_n) < d(v_1, v_n)$
- WFI algoritam iterira po vrhovima grafa postavljajući ih kao međuvrh v_k
 - na taj se način testira da li je taj međuvrh v_k na minimalnoj putanji između v_i i v_j
 - s obzirom da WFI algoritam izračunava udaljenosti između svih vrhova, imamo tri petlje s kojima iteriramo po vrhovima grafa, odabirući u svakoj od njih v_i , v_j i v_k



Warshall-Floyd-Ingerman algoritam

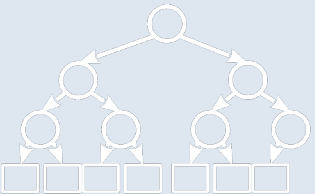
- Udaljenosti se računaju kao

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

- kada gledamo međuvrh v_k , udaljenost je minimum između udaljenosti d_{ij} izračunate za međuvrh v_{k-1} i udaljenosti putanje koja prolazi kroz međuvrh v_k
- Putanje (prethodnici) spremaju se u matricu putanja kao

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

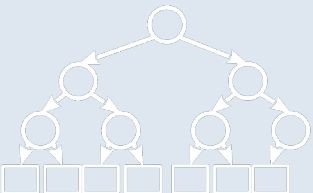
- što znači da ako smo ažurirali udaljenost d_{ij} tada moramo staviti i da je prethodnik vrha v_j međuvrh v_k



Warshall-Floyd-Ingerman algoritam

- Na početku imamo inicijalnu matricu udaljenosti, koja je $D^0 = W$, te sadrži udaljenosti samo direktno povezanih vrhova, izračun ostalih udaljenosti je stvar WFI algoritma
- Inicijalna matrica putanja definira se iz težinske matrice susjedstva W na način

$$\pi_{ij}^0 = \begin{cases} null & , i = j \text{ or } w_{ij} = 0 \\ i & , i \neq j \text{ and } w_{ij} \neq 0 \end{cases}$$



Warshall-Floyd-Ingerman algoritam

procedure WFI(W)

Create initial distance matrix $D = D^0$ from W

Create initial path matrix $\Pi = \Pi^0$ from W

for k from 1 to $|V|$ **do**

for i from 1 to $|V|$ **do**

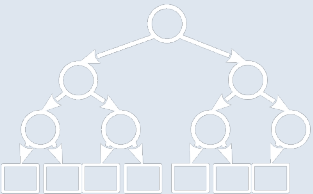
for j from 1 to $|V|$ **do**

if $D[i, k] + D[k, j] < D[i, j]$ **then**

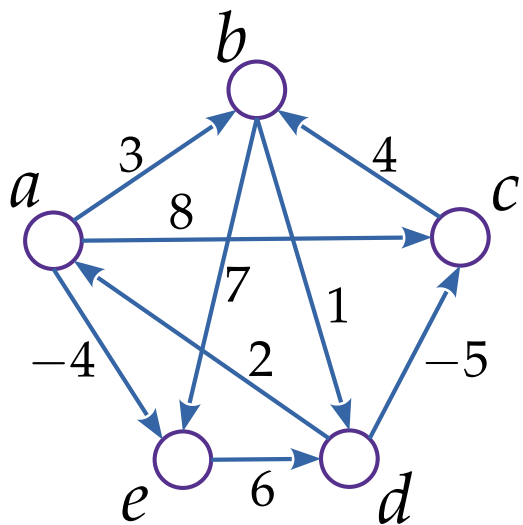
$D[i, j] = D[i, k] + D[k, j]$

$\Pi[i, j] = \Pi[k, j]$

- Algoritam je jednostavan, ima tri petlje kojima iteriramo po vrhovima v_i , v_j i v_k
- Time je i kompleksnost algoritma nešto visoka $O(V^3)$



WFI algoritam - primjer

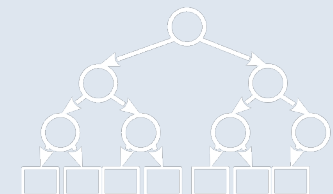


$$\mathbf{D}^0 = \begin{matrix} & \begin{matrix} a & b & c & d & e \end{matrix} \\ \begin{matrix} a \\ b \\ c \\ d \\ e \end{matrix} & \begin{bmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \infty & -5 & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{bmatrix} \end{matrix}, \mathbf{\Pi}^0 =$$

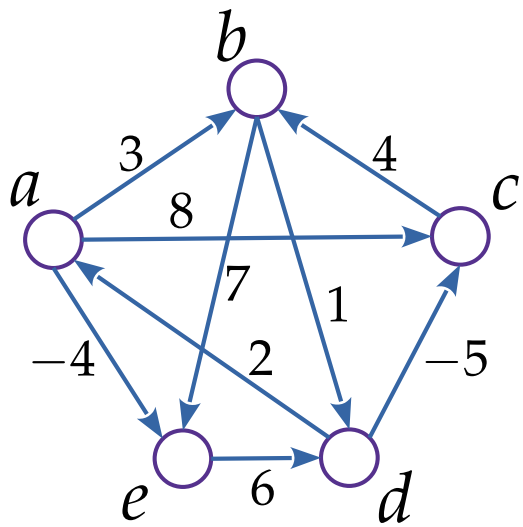
$$\begin{matrix} & \begin{matrix} a & b & c & d & e \end{matrix} \\ \begin{matrix} a \\ b \\ c \\ d \\ e \end{matrix} & \begin{bmatrix} \text{null} & 1 & 1 & \text{null} & 1 \\ \text{null} & \text{null} & \text{null} & 2 & 2 \\ \text{null} & 3 & \text{null} & \text{null} & \text{null} \\ 4 & \text{null} & 4 & \text{null} & \text{null} \\ \text{null} & \text{null} & \text{null} & 5 & \text{null} \end{bmatrix} \end{matrix}$$

$$\mathbf{D}^1 = \begin{matrix} & \begin{matrix} a & b & c & d & e \end{matrix} \\ \begin{matrix} a \\ b \\ c \\ d \\ e \end{matrix} & \begin{bmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & 5(\infty) & -5 & 0 & -2(\infty) \\ \infty & \infty & \infty & 6 & 0 \end{bmatrix} \end{matrix}, \mathbf{\Pi}^1 =$$

$$\begin{matrix} & \begin{matrix} a & b & c & d & e \end{matrix} \\ \begin{matrix} a \\ b \\ c \\ d \\ e \end{matrix} & \begin{bmatrix} \text{null} & 1 & 1 & \text{null} & 1 \\ \text{null} & \text{null} & \text{null} & 2 & 2 \\ \text{null} & 3 & \text{null} & \text{null} & \text{null} \\ 4 & 1(\text{null}) & 4 & \text{null} & 1(\text{null}) \\ \text{null} & \text{null} & \text{null} & 5 & \text{null} \end{bmatrix} \end{matrix}$$

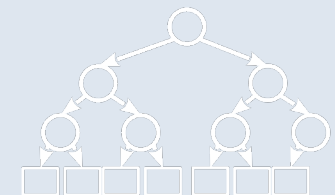


WFI algoritam - primjer

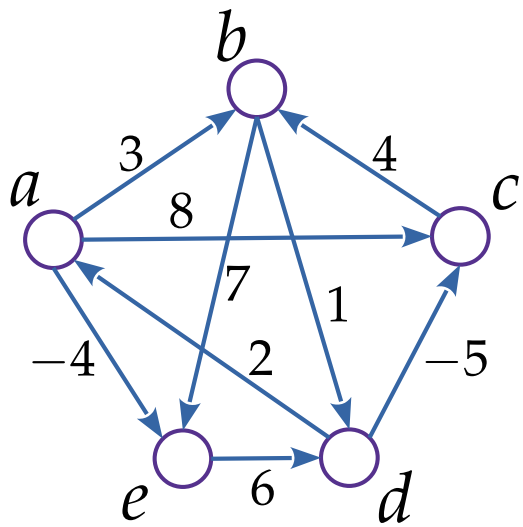


$$D^2 = \begin{matrix} & \begin{matrix} a & b & c & d & e \end{matrix} \\ \begin{matrix} a \\ b \\ c \\ d \\ e \end{matrix} & \begin{bmatrix} 0 & 3 & 8 & 4(\infty) & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & 5(\infty) & 11(\infty) \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{bmatrix} \end{matrix}, \Pi^2 = \begin{matrix} & \begin{matrix} a & b & c & d & e \end{matrix} \\ \begin{matrix} a \\ b \\ c \\ d \\ e \end{matrix} & \begin{bmatrix} \text{null} & 1 & 1 & 2(\text{null}) & 1 \\ \text{null} & \text{null} & \text{null} & 2 & 2 \\ \text{null} & 3 & \text{null} & 2(\text{null}) & 2(\text{null}) \\ 4 & 1 & 4 & \text{null} & 1 \\ \text{null} & \text{null} & \text{null} & 5 & \text{null} \end{bmatrix} \end{matrix}$$

$$D^3 = \begin{matrix} & \begin{matrix} a & b & c & d & e \end{matrix} \\ \begin{matrix} a \\ b \\ c \\ d \\ e \end{matrix} & \begin{bmatrix} 0 & 3 & 8 & 4 & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & -1(5) & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{bmatrix} \end{matrix}, \Pi^3 = \begin{matrix} & \begin{matrix} a & b & c & d & e \end{matrix} \\ \begin{matrix} a \\ b \\ c \\ d \\ e \end{matrix} & \begin{bmatrix} \text{null} & 1 & 1 & 2 & 1 \\ \text{null} & \text{null} & \text{null} & 2 & 2 \\ \text{null} & 3 & \text{null} & 2 & 2 \\ 4 & 3(1) & 4 & \text{null} & 1 \\ \text{null} & \text{null} & \text{null} & 5 & \text{null} \end{bmatrix} \end{matrix}$$

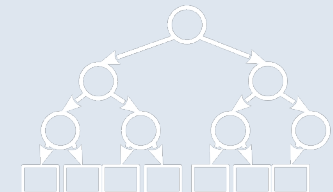


WFI algoritam - primjer



$$\mathbf{D}^4 = \begin{matrix} & \begin{matrix} a & b & c & d & e \end{matrix} \\ \begin{matrix} a \\ b \\ c \\ d \\ e \end{matrix} & \begin{bmatrix} 0 & 3 & -1(8) & 4 & -4 \\ 3(\infty) & 0 & -4(\infty) & 1 & -1(7) \\ 7(\infty) & 4 & 0 & 5 & 3(11) \\ 2 & -1 & -5 & 0 & -2 \\ 8(\infty) & 5(\infty) & 1(\infty) & 6 & 0 \end{bmatrix} \end{matrix}, \mathbf{\Pi}^4 = \begin{matrix} & \begin{matrix} a & b & c & d & e \end{matrix} \\ \begin{matrix} a \\ b \\ c \\ d \\ e \end{matrix} & \begin{bmatrix} \text{null} & 1 & 4(1) & 2 & 1 \\ 4(\text{null}) & \text{null} & 4(\text{null}) & 2 & 1(2) \\ 4(\text{null}) & 3 & \text{null} & 2 & 1(2) \\ 4 & 3 & 4 & \text{null} & 1 \\ 4(\text{null}) & 3(\text{null}) & 4(\text{null}) & 5 & \text{null} \end{bmatrix} \end{matrix}$$

$$\mathbf{D}^5 = \begin{matrix} & \begin{matrix} a & b & c & d & e \end{matrix} \\ \begin{matrix} a \\ b \\ c \\ d \\ e \end{matrix} & \begin{bmatrix} 0 & 1(3) & -3(-1) & 2(4) & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{bmatrix} \end{matrix}, \mathbf{\Pi}^5 = \begin{matrix} & \begin{matrix} a & b & c & d & e \end{matrix} \\ \begin{matrix} a \\ b \\ c \\ d \\ e \end{matrix} & \begin{bmatrix} \text{null} & 3(1) & 4(4) & 5(2) & 1 \\ 4 & \text{null} & 4 & 2 & 1 \\ 4 & 3 & \text{null} & 2 & 1 \\ 4 & 3 & 4 & \text{null} & 1 \\ 4 & 3 & 4 & 5 & \text{null} \end{bmatrix} \end{matrix}$$

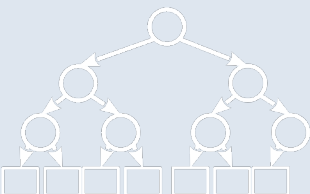


WFI algoritam - primjer

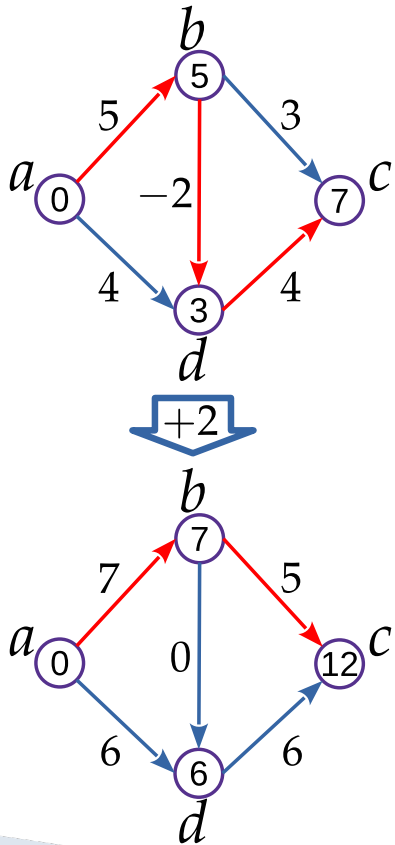
k	vertex
$\pi_{35}^5 = 1$	$v_5 = e$
$\pi_{31}^5 = 4$	$v_1 = a$
$\pi_{34}^5 = 2$	$v_4 = d$
$\pi_{32}^5 = 3$	$v_2 = b$
$\pi_{33}^5 = \text{null}$	$v_3 = c$

$$\Pi^5 = \begin{matrix} & \begin{matrix} a & b & c & d & e \end{matrix} \\ \begin{matrix} a \\ b \\ c \\ d \\ e \end{matrix} & \begin{bmatrix} \text{null} & 3(1) & 4(4) & 5(2) & 1 \\ 4 & \text{null} & 4 & 2 & 1 \\ 4 & 3 & \text{null} & 2 & 1 \\ 4 & 3 & 4 & \text{null} & 1 \\ 4 & 3 & 4 & 5 & \text{null} \end{bmatrix} \end{matrix}$$

- Gledamo udaljenost i najkraću putanju između $i = 3 = c$ i $j = 5 = e$
- Udaljenost očitamo direktno iz D^5 , to jest $d_{35}^5 = 3$
- Čitanje matrice putanja se interpretira kao
 - Ako je v_i početni vrh, a v_j završni, tada imamo putanju $v_i v_{k-n} \dots v_{k-1} v_k v_j$
 - U matrici Π^5 imamo $\pi_{ij}^5 = k$, pa zatim $\pi_{ik}^5 = k - 1$, pa $\pi_{i(k-1)}^5 = k - 1$, sve do $\pi_{i(k-n)}^5 = i$
 - π_{ii}^5 će po definiciji biti *null* i tu završavamo
- Za putanju idemo unatrag kroz matricu od π_{35}^5 , gdje je $i = 3$, a $j = 5$, pa je tako
 - $j = 5$ i to je vrh e
 - $k = \pi_{ij}^5 = \pi_{35}^5 = 1$ i to je vrh a
 - $k - 1 = \pi_{ik}^5 = \pi_{31}^5 = 4$ i to je vrh d
 - $k - 2 = \pi_{i(k-1)}^5 = \pi_{34}^5 = 2$ i to je vrh b
 - $k - 3 = \pi_{i(k-2)}^5 = \pi_{32}^5 = 3 = i$ i to je vrh c
 - $\pi_{ii}^5 = \pi_{33}^5 = \text{null}$ i tu završavamo.



Transformacija težina



- Žarko želimo koristiti Dijkstrin algoritam, ali nam smetaju negativne težine na bridovima
- Da li je moguće nekako transformirati graf da transformacijom uklonimo negativne težine?
- Naivni pokušaj bio bi identičnom translacijom prema najnegativnijoj težini brida
- Najkraći put od *a* do *c* u originalnom grafu je *abdc* s udaljenošću 7
- Dodavanjem težine 2 na sve bridove to se remeti i sada je najkraća putanja *abc* s udaljenošću 12

Transformacija težine

- Ono što znamo sa prethodnih prikaznica je

$$d(v) \leq d(u) + w(uv)$$

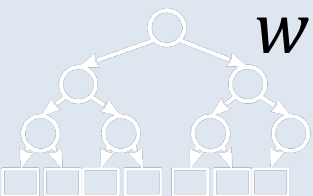
- znači udaljenost vrha v nije veća od udaljenosti vrha u uvećana za težinu brida $w(uv)$
- zapišemo li to drukčije dobivamo da je

$$0 \leq d(u) + w(uv) - d(v) = w'(uv)$$

- Ako prethodni izraz upotrijebimo na putanji $p = v_1 v_2 \dots v_k$ dobivamo udaljenost

$$L(v_1, v_k)' = \sum_{i=1}^{k-1} w'(v_i v_{i+1}) =$$

$$w(v_1 v_2) + d(v_1) - d(v_2) + \dots + w(v_{k-1} v_k) + d(v_{k-1}) - d(v_k)$$



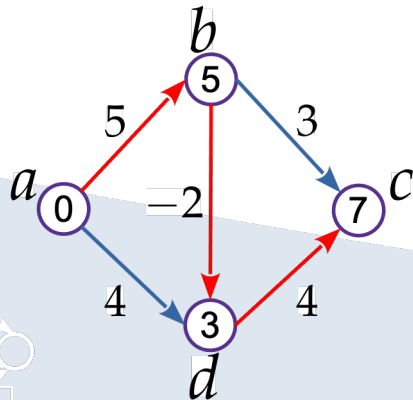
Transformacija težine

- Također vrijedi

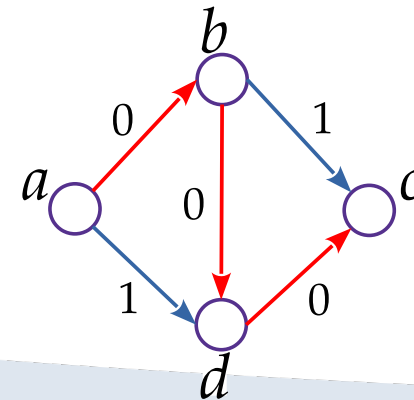
$$L'(v_1, v_k) = \left(\sum_{i=1}^{k-1} w(v_i v_{i+1}) \right) + d(v_1) - d(v_k) = L(v_1 v_k) + d(v_1) - d(v_k)$$

- Transformacija je bijektivna pa vrijedi i

$$L(v_1 v_k) = L'(v_1 v_k) + d(v_k) - d(v_1)$$



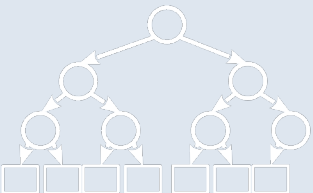
v_i	v_j	$w(v_i v_j)$	$d(v_i)$	$d(v_j)$	$w'(v_i v_j)$
a	b	5	0	5	$5+0-5=0$
a	d	4	0	3	$4+0-3=1$
b	d	-2	5	3	$-2+5-3=0$
b	c	3	5	7	$3+5-7=1$
d	c	4	3	7	$4+3-7=0$



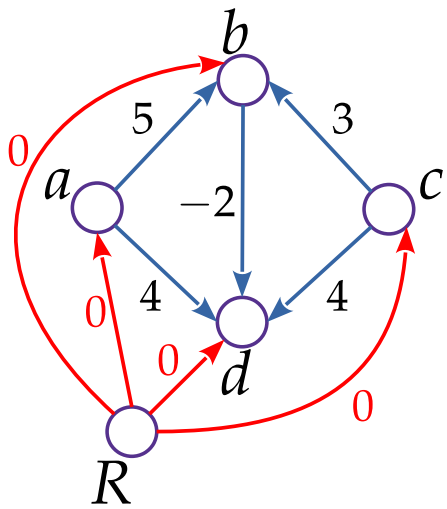
Transformacija težine

- 1: Using a *label-correcting* algorithm, determine the shortest distance to every vertex in the input graph G from an arbitrary reference vertex v_r . All vertices in the input graph G must be reachable from the reference vertex v_r .
- 2: Transform the input graph G into the non-negative weighted graph G' using the bijective transformation given in (3.51).
- 3: **for** $\forall v_i, v_k \in V(G')$ **do**
- 4: Find the shortest path between source and destination vertices v_i and v_k in the transformed graph G' using a *label-setting* algorithm. The length of this path is $L'(v_i v_k)$.
- 5: Use the inverse transformation in (3.51) to get the original length of the shortest path as $L(v_i v_k) = L'(v_i v_k) + d(v_k) - d(v_i)$.

- Da bismo odradili transformaciju težine, prvo trebamo Bellman-Ford algoritmom odrediti udaljenost svih vrhova od jednog određenog vrha
- Zatim odradimo transformaciju
- Nakon toga korištenjem Dijkstrinog algoritma odredimo udaljenost između svih parova vrhova u grafu, čime dobivamo *all-to-all* udaljenosti



Transformacija težine



- Ukoliko imamo nepovezani graf, tada je nemoguće izračunati udaljenost svih vrhova od jednog referentnog vrha
- Tada dodajemo umjetni vrh (recimo R) koji bridovima povezujemo sa svim ostalim vrhovima grafa
 - Težine tih umjetno dodanih bridova su 0
- Sad možemo odrediti udaljenost svih vrhova od umjetnog vrha R i na temelju toga napraviti transformaciju

v	$d(v)$
a	0
b	0
c	0
d	-2

v_i	v_j	$w(v_i v_j)$	$d(v_i)$	$d(v_j)$	$w'(v_i v_j)$
a	b	5	0	0	$5+0-0=5$
a	d	4	0	-2	$4+0+2=6$
b	d	-2	0	-2	$-2+0+2=0$
c	b	3	0	0	$3+0-0=3$
c	d	4	0	-2	$4+0+2=6$

