

فرادرس

فرادرس آنلاین
کلاس درس
www.faradars.org

طراحی الگوریتم

درس هفتم : الگوریتم های گراف

مدرب: فرشید شیراًفکن

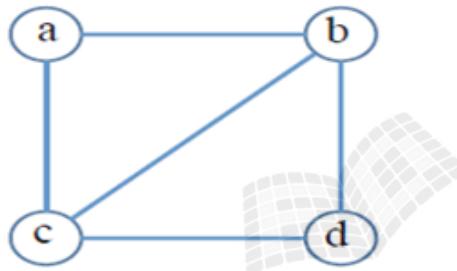
دانشجوی دکتری دانشگاه تهران

(کارشناسی و کارشناسی ارشد: کامپیوتر نرم افزار)

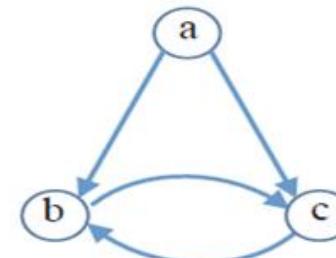
(دکتری: بیو انفورماتیک)

تعريف

- *Undirected*
- $V = \{a, b, c, d\}$
- $E = \{\{a, b\}, \{a, c\}, \{b, c\}, \{b, d\}, \{c, d\}\}$



- *Directed*
- $V = \{a, b, c\}$
- $E = \{(a, c), (a, b), (b, c), (c, b)\}$

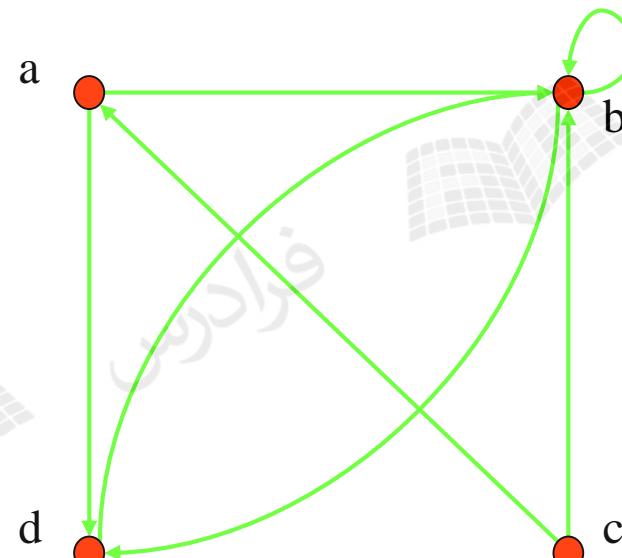


درجه گراف

$$\deg^-(a) = 1$$

$$\deg^+(a) = 2$$

$$\begin{aligned}\deg^-(d) &= 2 \\ \deg^+(d) &= 1\end{aligned}$$



$$\deg^-(b) = 4$$

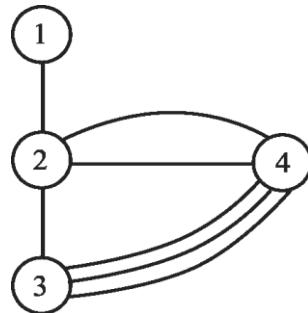
$$\deg^+(b) = 2$$

$$\deg^-(c) = 0$$

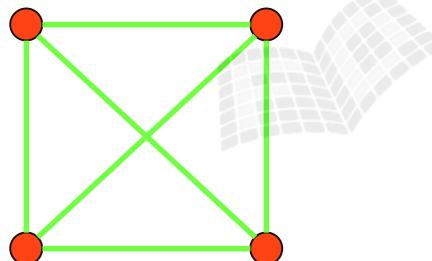
$$\deg^+(c) = 2$$

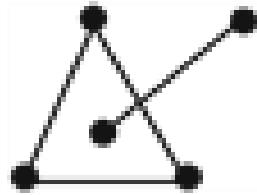
انواع گراف

گراف چند گانه : گرافی که در آن یالهای چندگانه (Multi Edge) مجاز باشد.



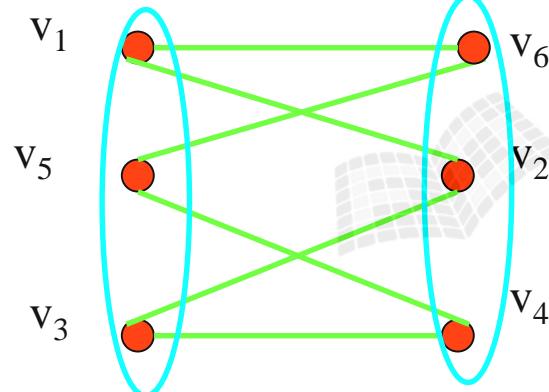
گراف کامل : گراف بدون جهتی که همه یالهای آن رسم شده باشد.





گراف همبند : گرافی که یک مسیر بین هر دو گره آن وجود داشته باشد.
یک گراف ناهمبند :

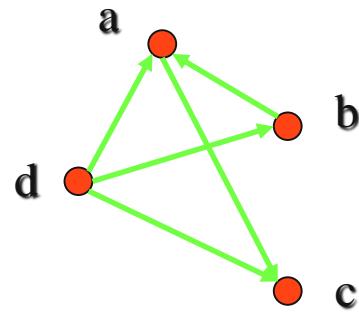
گراف همبند قوی : گراف جهت داری که برای هر زوج گره v,u هم یک مسیر از u به v و هم یک مسیر از v به u وجود داشته باشد.



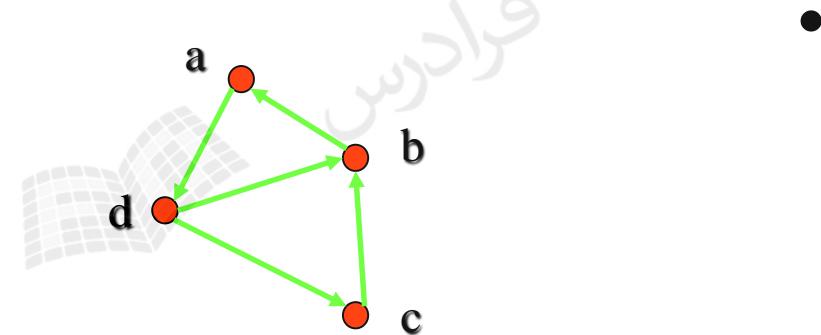
گراف دو قسمتی (Bipartite)

گرافی که نودهای آن قابل دسته بندی به دو مجموعه مستقل U و V هستند. هر یال در این گراف نودی از مجموعه U را به نودی در مجموعه V وصل می کند. این گراف را معمولاً به صورت $G=(V,U,E)$ نمایش می دهند.

Connectivity



Weakly connected
no path from b to d.



Strongly connected

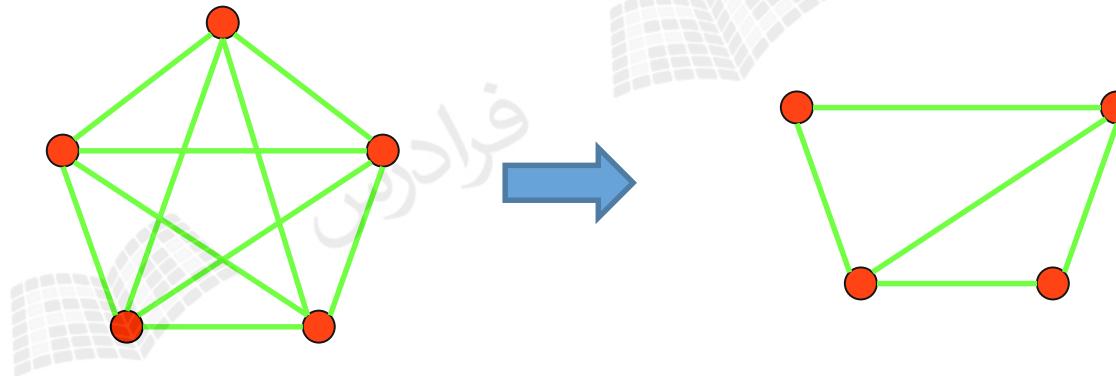
زیرگراف (subgraph)

یک زیر گراف از گراف $H=(W,F)$ ، یک گراف $G=(V,E)$ است که :

$$W \subseteq V$$

and

$$F \subseteq E$$

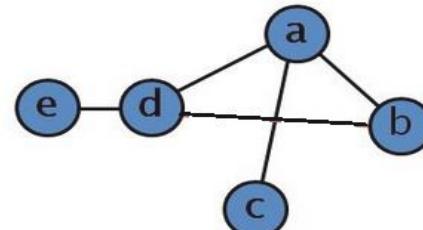


نمایش گراف

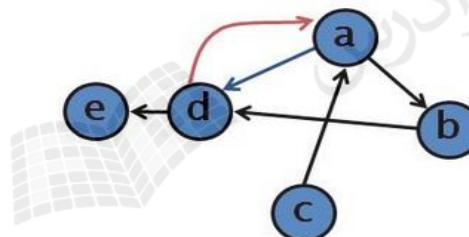
۱- ماتریس همجواری
(Adjacency matrices)

۲- لیست همجواری
(Adjacency List)

ماتریس هم جواری

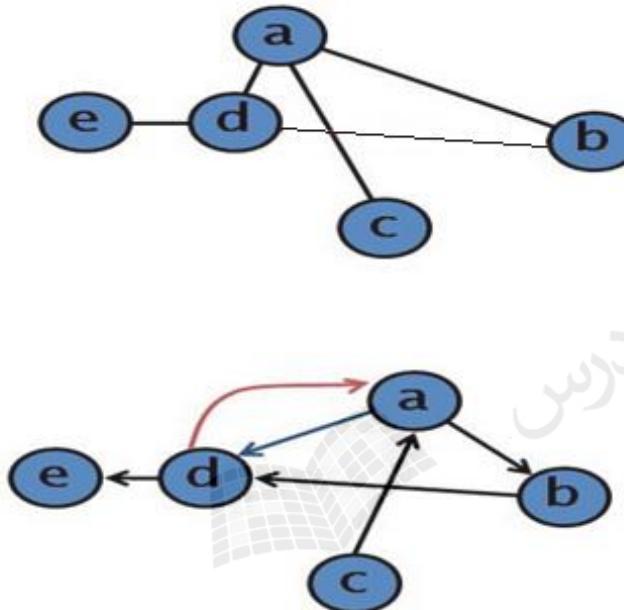


| | a | b | c | d | e |
|---|---|---|---|---|---|
| a | 0 | 1 | 1 | 1 | 0 |
| b | 1 | 0 | 0 | 1 | 0 |
| c | 1 | 0 | 0 | 0 | 0 |
| d | 1 | 1 | 0 | 0 | 1 |
| e | 0 | 0 | 0 | 1 | 0 |



| | a | b | c | d | e |
|---|---|---|---|---|---|
| a | 0 | 1 | 0 | 1 | 0 |
| b | 0 | 0 | 0 | 1 | 0 |
| c | 1 | 0 | 0 | 0 | 0 |
| d | 1 | 0 | 0 | 0 | 1 |
| e | 0 | 0 | 0 | 0 | 0 |

لیست همچواری

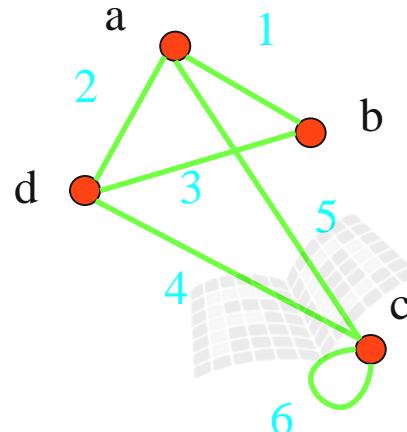


| | | | | | | | |
|---|---|---|---|---|---|---|---|
| a | - | b | - | d | - | c | ☒ |
| b | - | a | - | d | ☒ | | |
| c | - | a | ☒ | | | | |
| d | - | e | - | b | - | a | ☒ |
| e | - | d | ☒ | | | | |

| | | | | | | |
|---|---|---|---|---|---|--|
| a | - | b | - | d | ☒ | |
| b | - | d | ☒ | | | |
| c | - | a | ☒ | | | |
| d | - | a | - | e | - | |
| e | ☒ | | | | | |

ماتریس برخورد

این ماتریس برای یک گراف بدون جهت، یک ماتریس با $|V|$ سطر و $|E|$ ستون می‌باشد، چنانچه اگر لبه i راس j را تلaci کند، آنگاه درایه واقع در سطر i و ستون j برابر ۱ خواهد بود.



$$M = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 & 0 \end{bmatrix}$$

پیمایش گراف

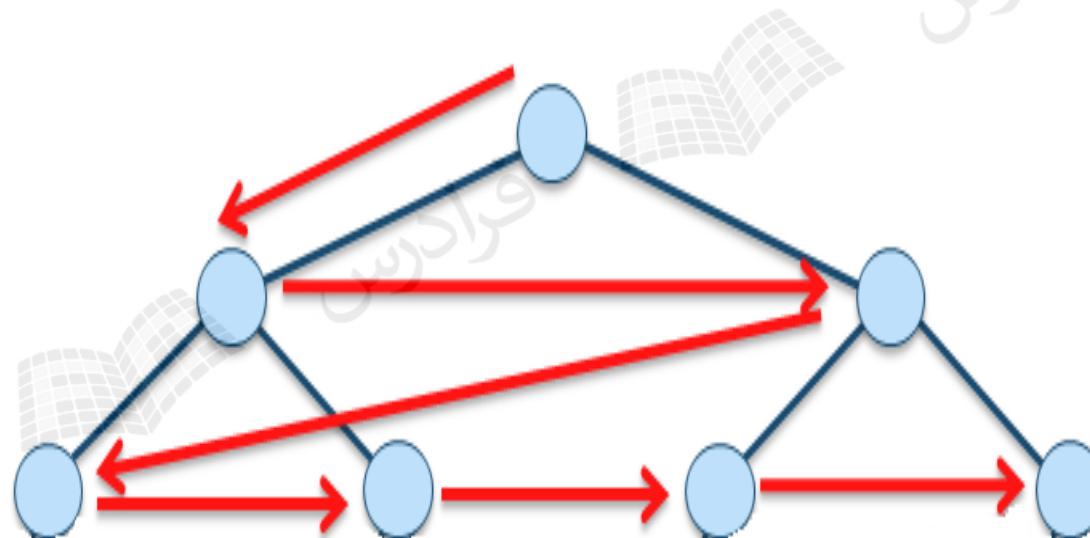
۱- سطحی (BFS : Breadth First Search)

۲- عمقی (DFS : Depth First Search)

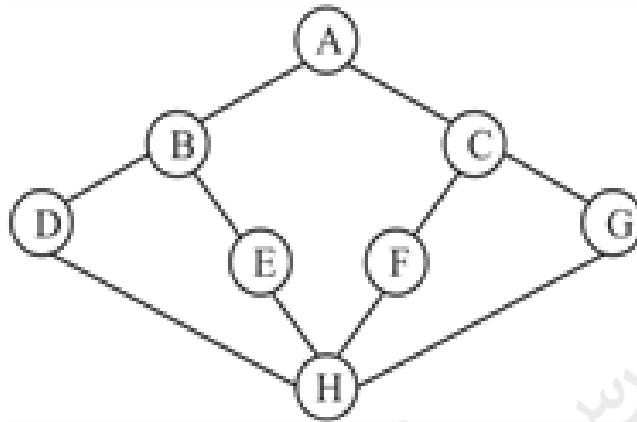
در پیمایش DFS از پشته و در پیمایش BFS از صف استفاده می‌شود.

پیمايش سطحی

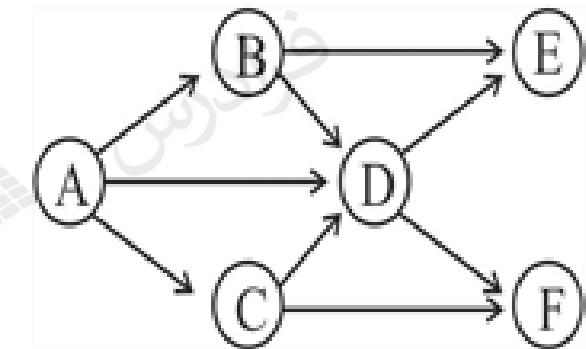
حال عنصر بعدی از صفت را حذف کرده و گره های مجاور آن را درج می کنیم. این عمل را ادامه داده تا همه گره ها پیمایش شوند.



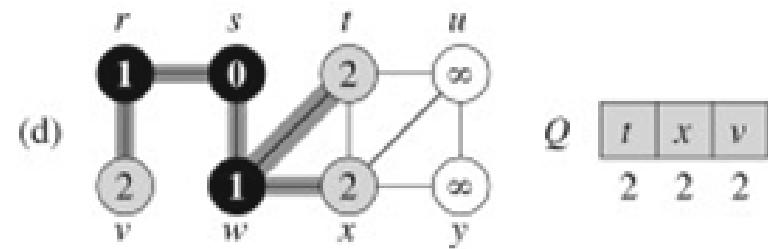
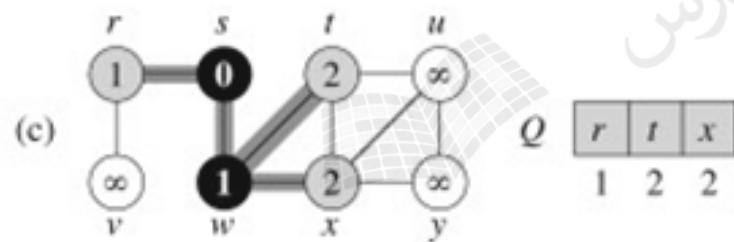
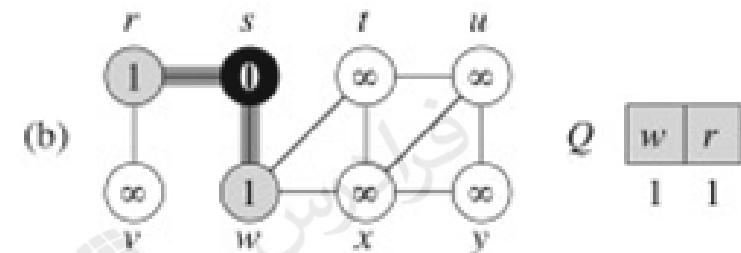
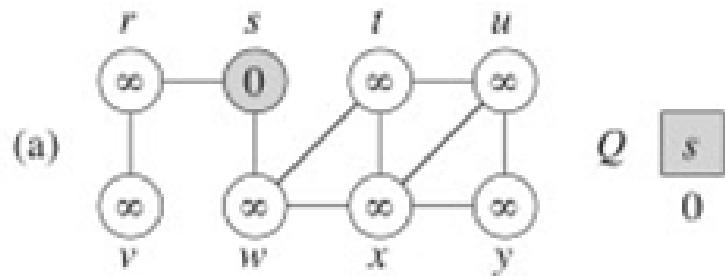
مثال

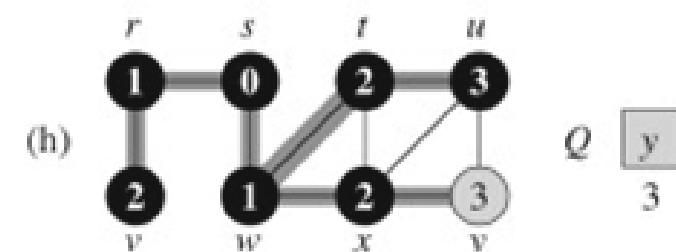
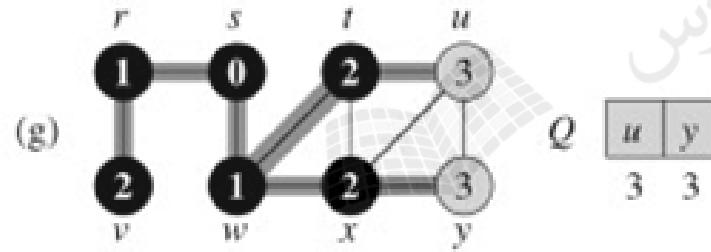
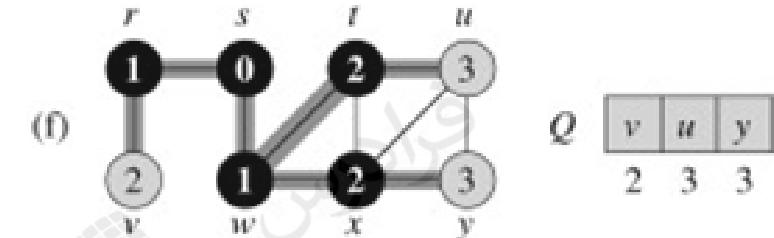
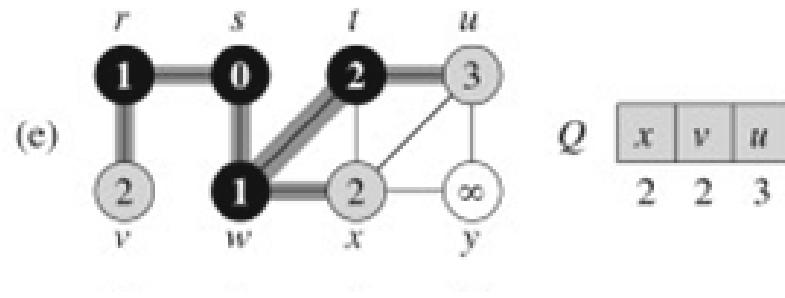


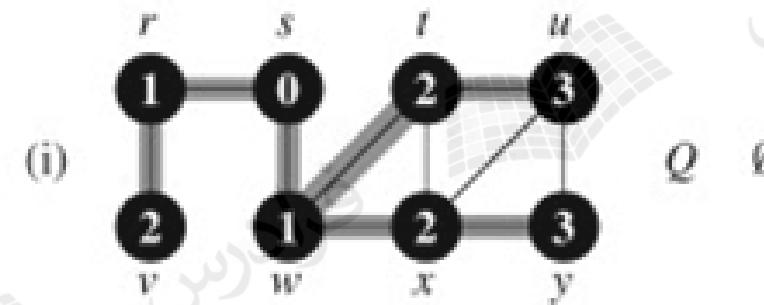
A B C D E F G H



A B C D E F







BFS

الگوريتم

$BFS(G, s)$

```

1  for each vertex  $u \in V[G] - \{s\}$ 
2       $color[u] = \text{WHITE}$ 
3       $d[u] = \infty$ 
4       $\pi[u] = \text{NIL}$ 
5   $color[s] = \text{GRAY}$ 
6   $d[s] = 0$ 
7   $\pi[s] = \text{NIL}$ 
8   $Q = \emptyset$ 
9  ENQUEUE( $Q, s$ )
10 while  $Q \neq \emptyset$ 
11      $u = \text{DEQUEUE}(Q)$ 
12     for each  $v \in Adj[u]$ 
13         if  $color[v] = \text{WHITE}$ 
14              $color[v] = \text{GRAY}$ 
15              $d[v] = d[u] + 1$ 
16              $\pi[v] = u$ 
17             ENQUEUE( $Q, v$ )
18      $color[u] = \text{BLACK}$ 

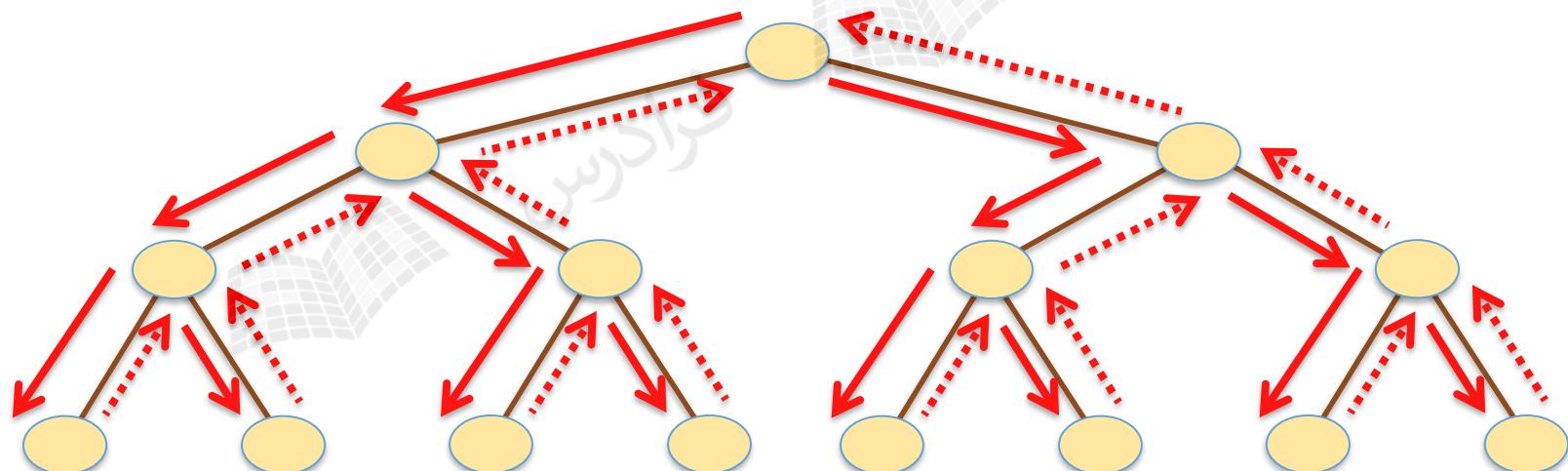
```

پیمایش عمقی

Depth-First Search (DFS)

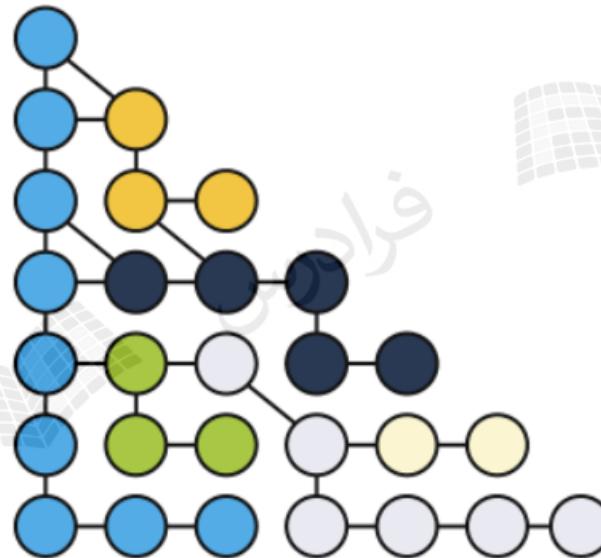
پیمایش عمقی

با شروع از یک گره، آن را ملاقات کرده و سپس کلیه گره‌های سمت چپ را تا آخرین عمق پیمایش می‌کنیم. اگر در پایین رفتن‌های متوالی، گره‌ای ملاقات شود که هیچ گره هم‌جواری نداشته باشد یا تمام گره‌های هم‌جوار آن ملاقات شده باشد، برگشت به یک سطح بالا انجام می‌گیرند و روند فوق تکرار می‌شود.



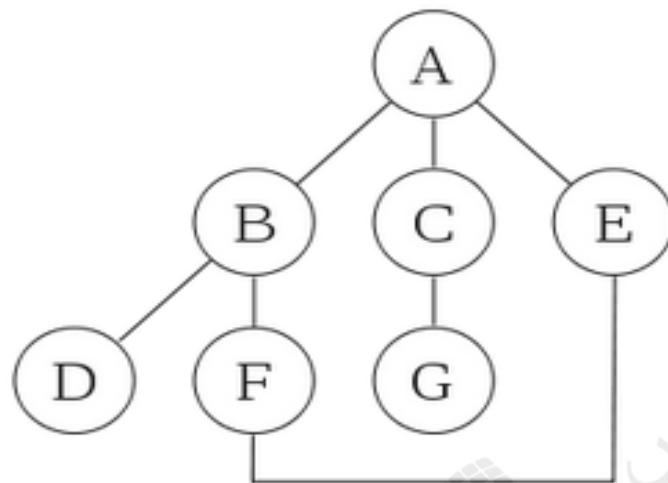
مثال

Depth-First Search

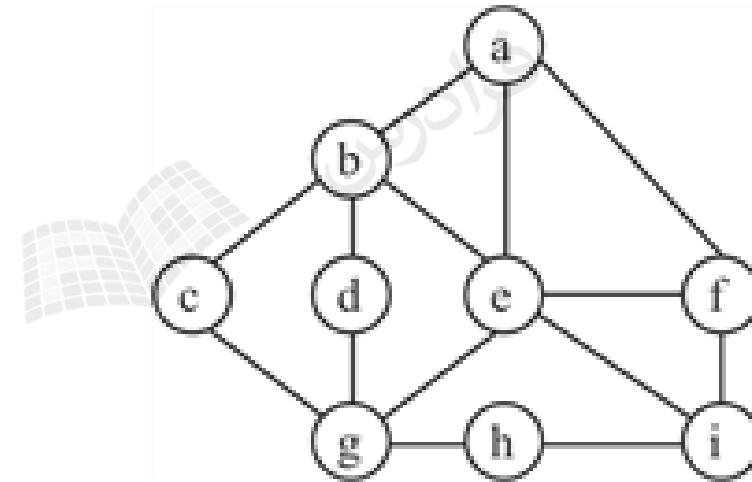


| Level |
|-------|
| 1 |
| 2 |
| 3 |
| 4 |
| 5 |
| 6 |

مثال

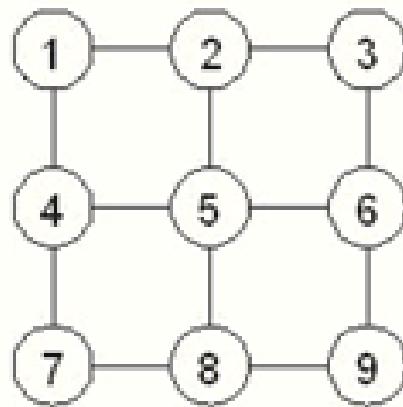


A B D F E C G

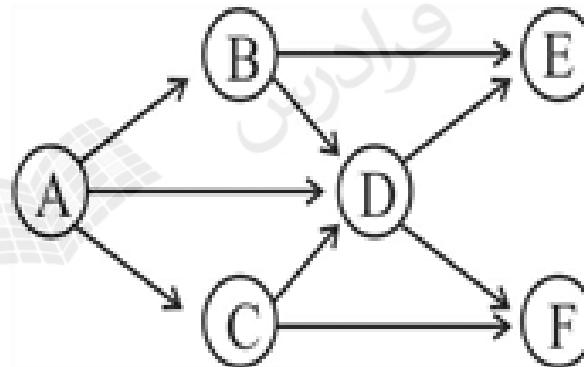


a b c g d e f i h

مثال



1 , 4 , 7 , 8 , 9 , 6 , 5 , 2 , 3

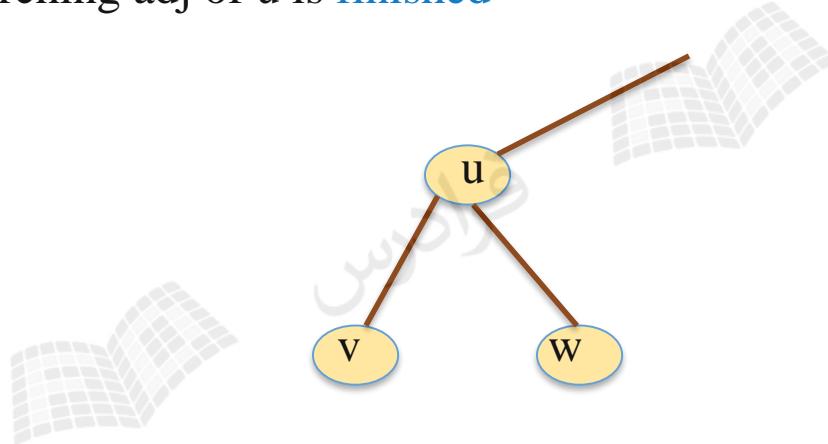


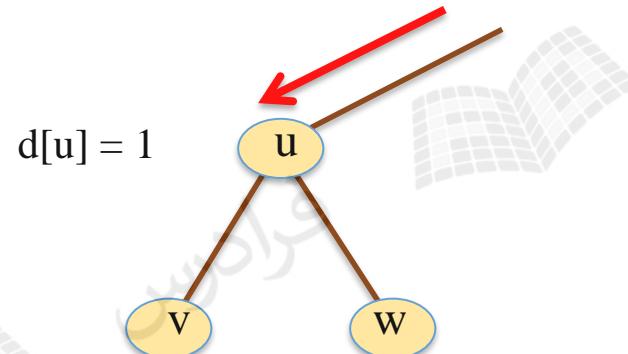
A B E D F C

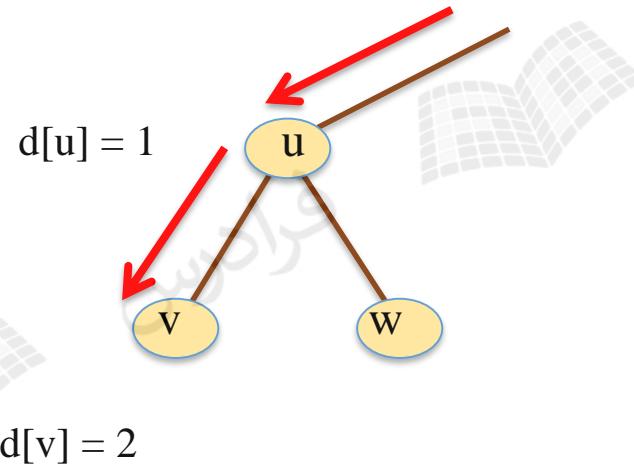
زمان کشف و زمان خاتمه

$d[u]$: when u is discovered

$f[u]$: when searching adj of u is finished

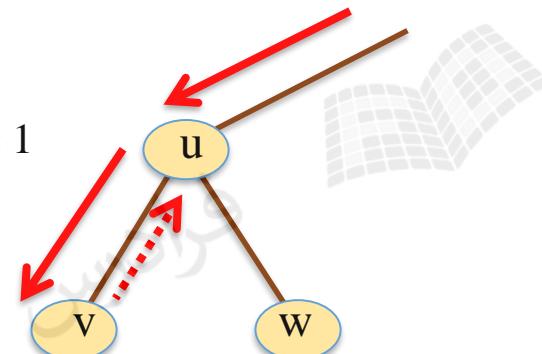






فصل

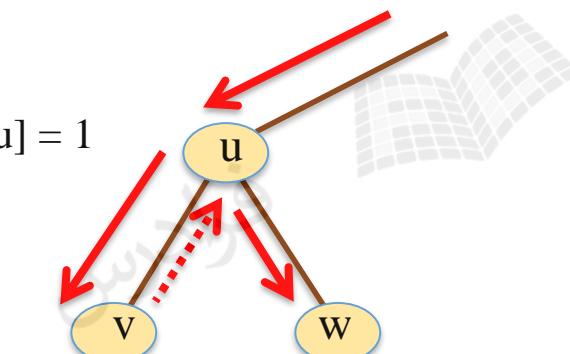
$$d[u] = 1$$



$$d[v] = 2$$

$$f[v] = 3$$

فرادرس

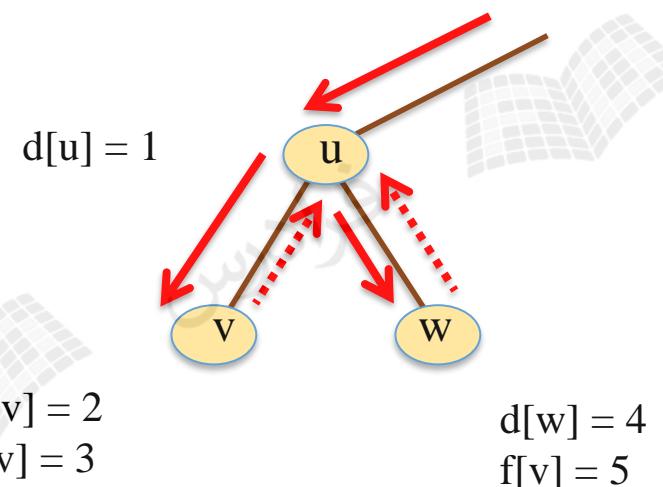


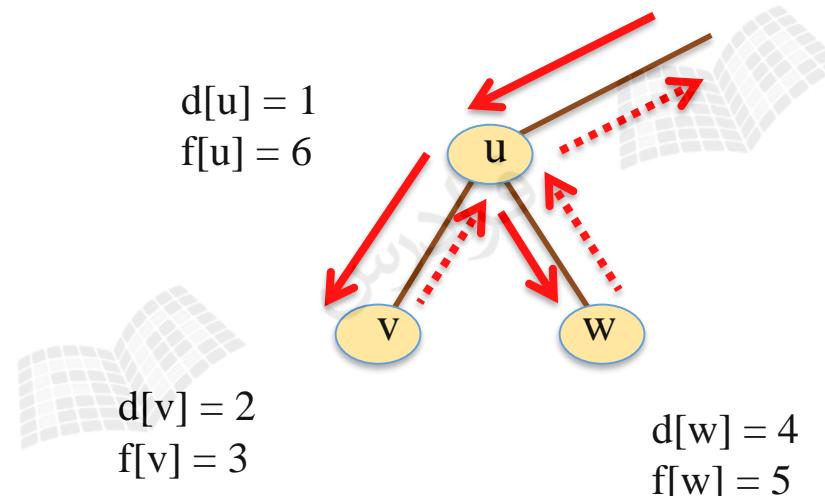
$$d[u] = 1$$

$$\begin{aligned} d[v] &= 2 \\ f[v] &= 3 \end{aligned}$$

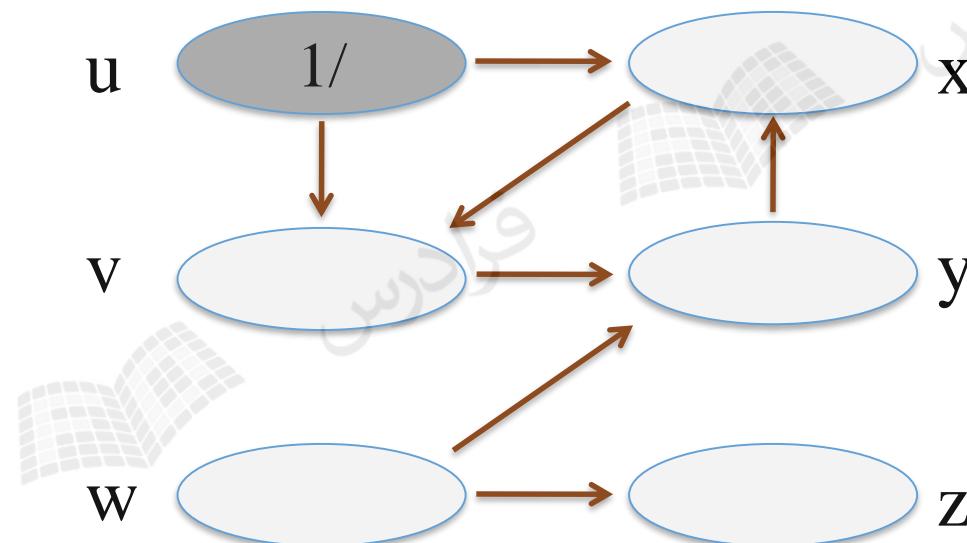
$$d[w] = 4$$

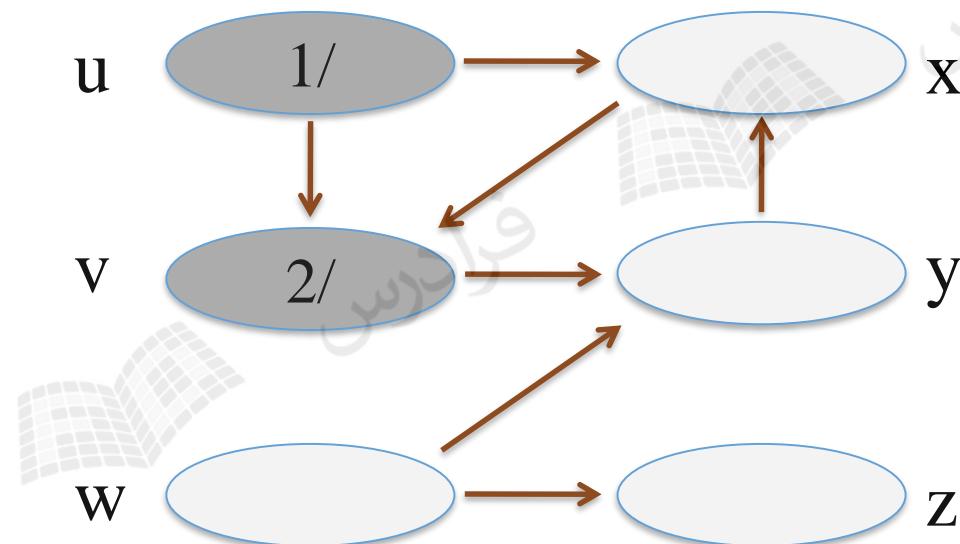
فصل

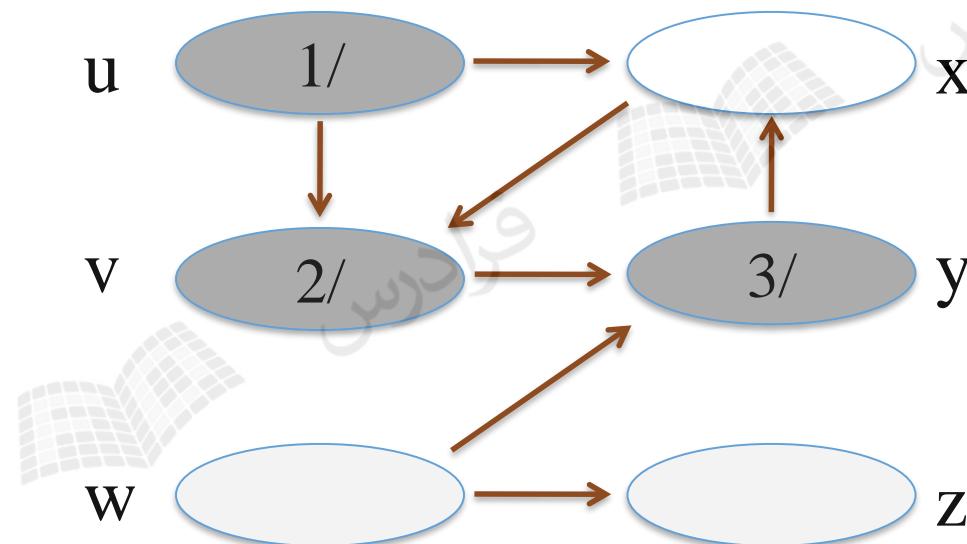


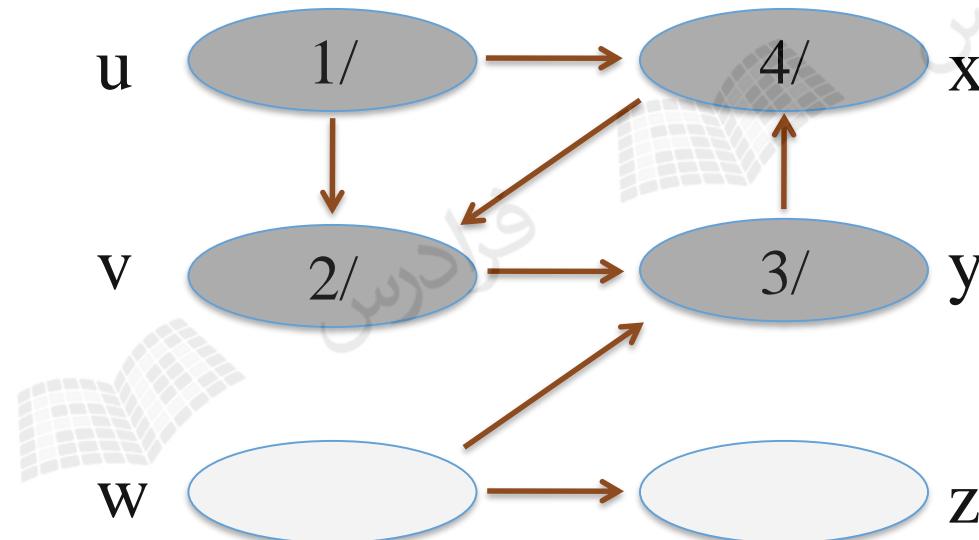


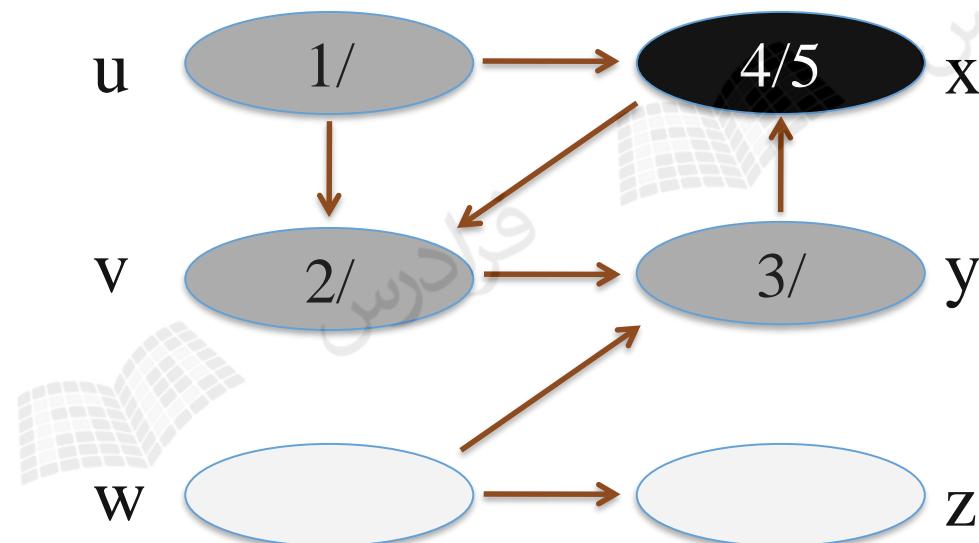
مثال

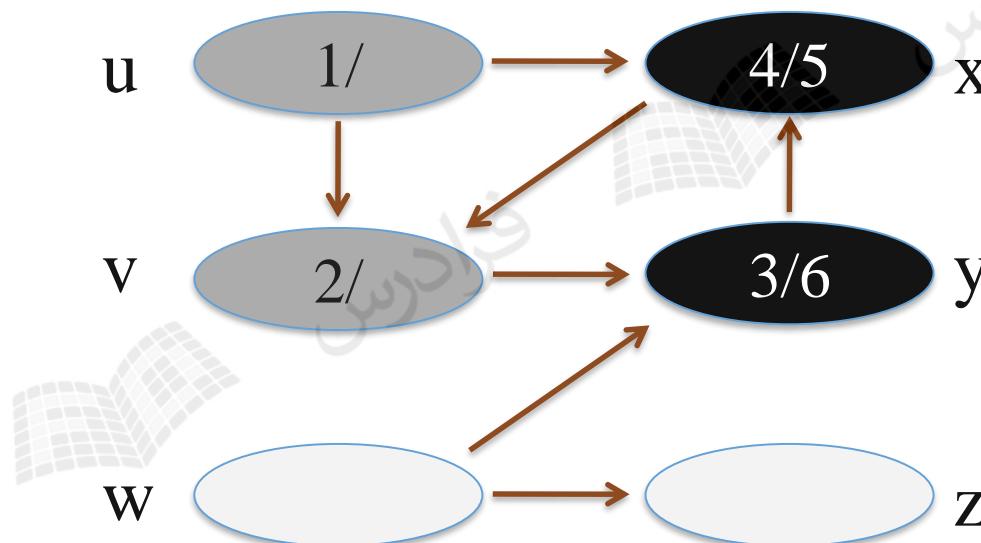


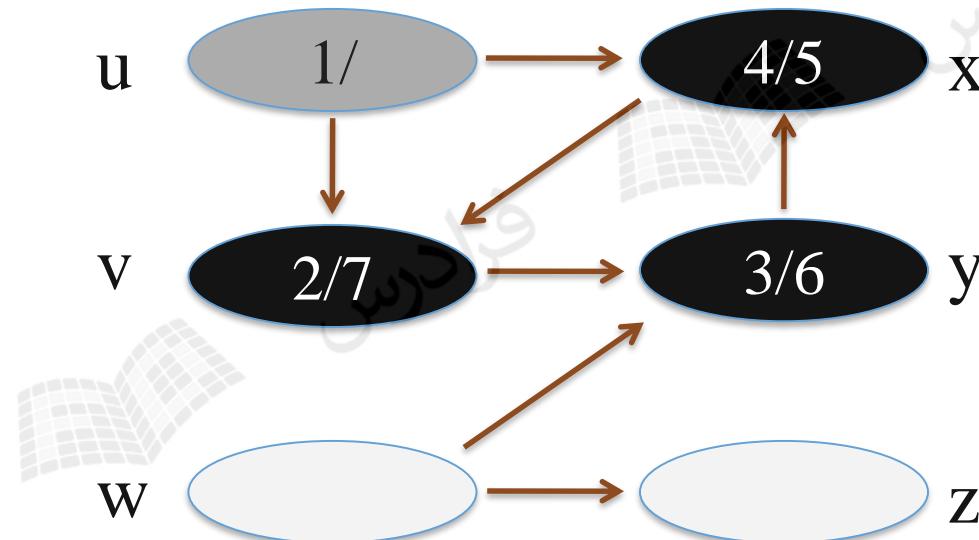


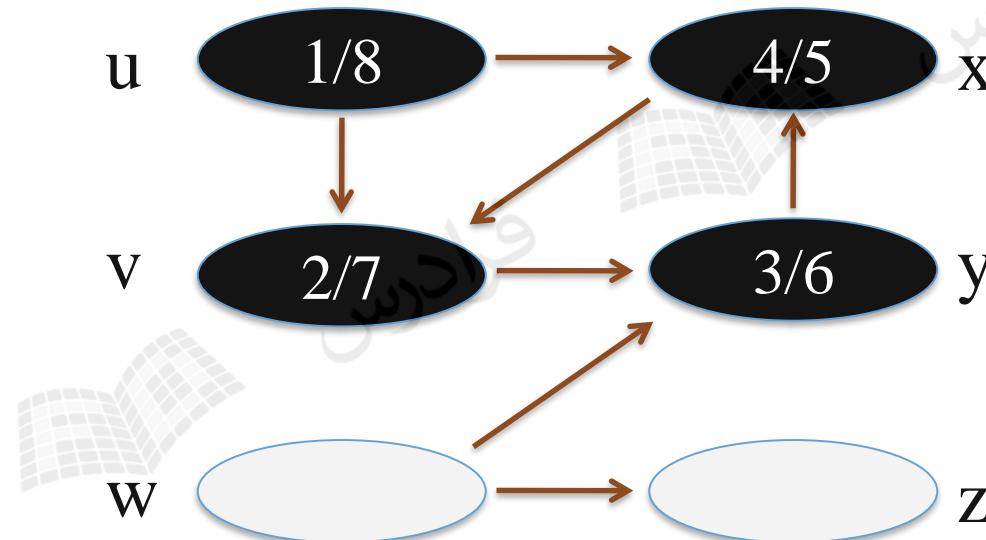


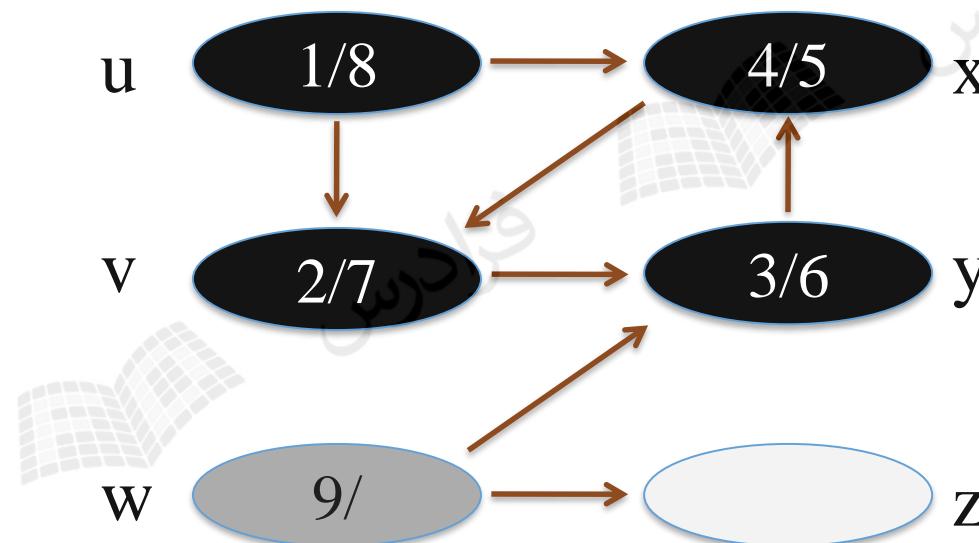


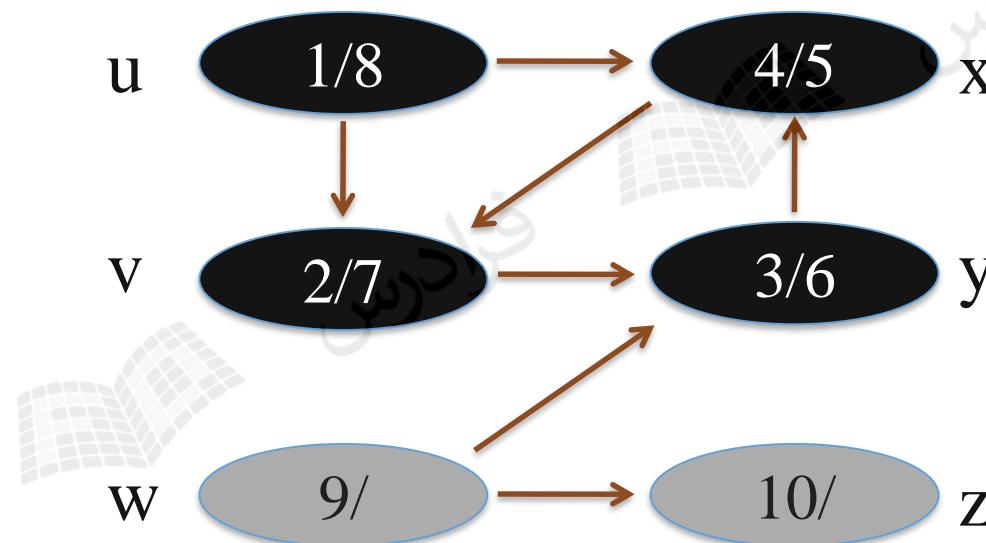


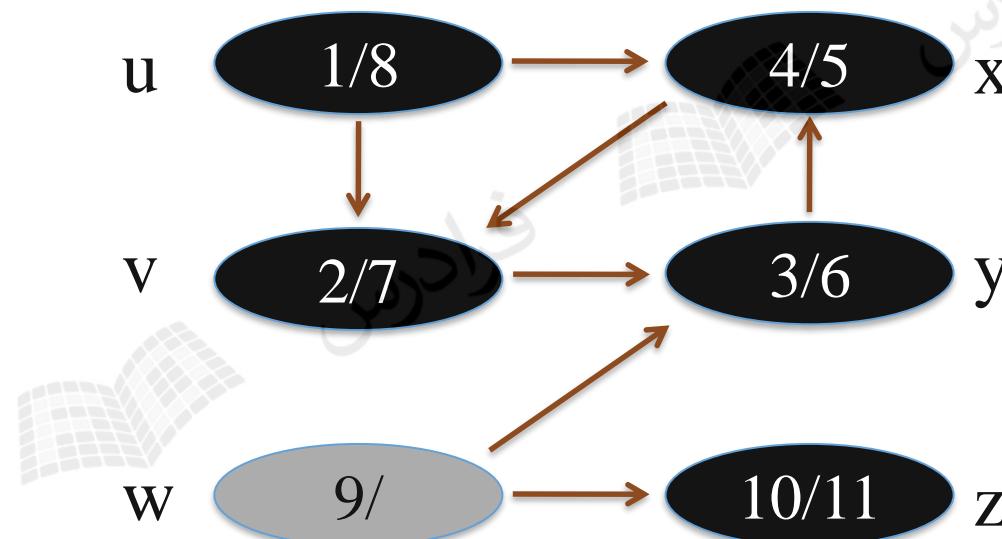


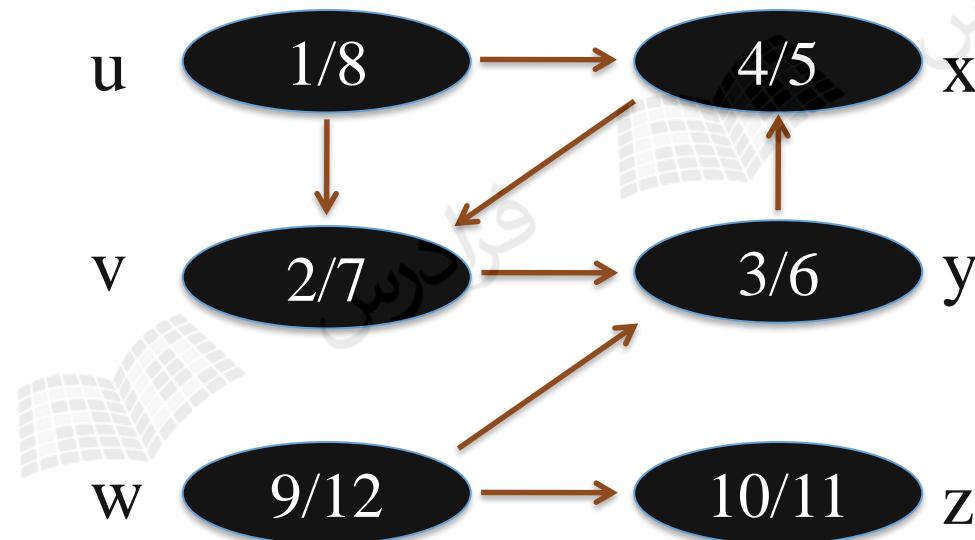


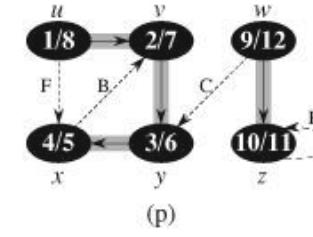
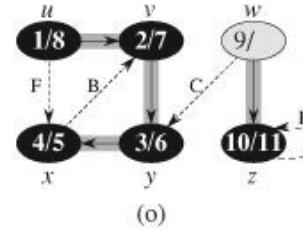
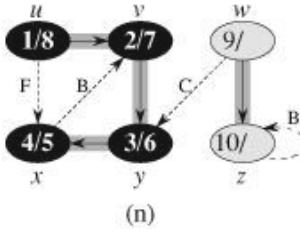
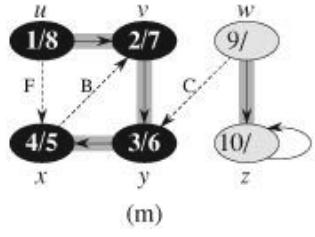
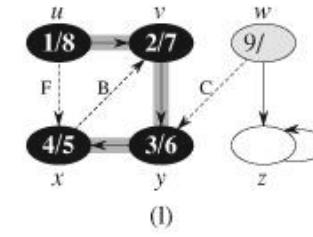
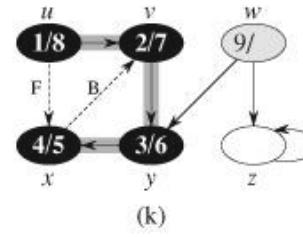
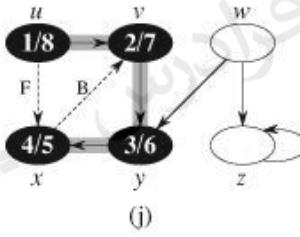
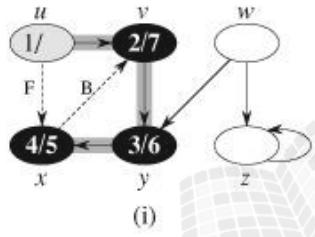
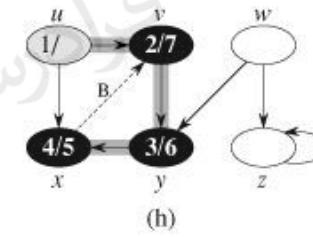
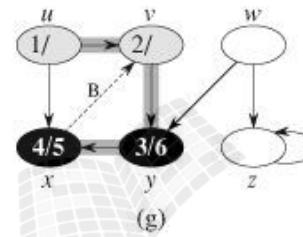
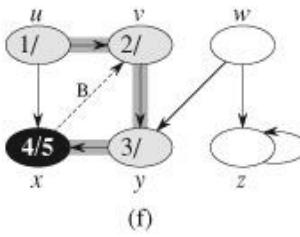
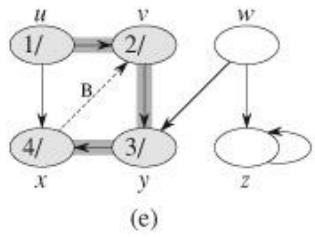
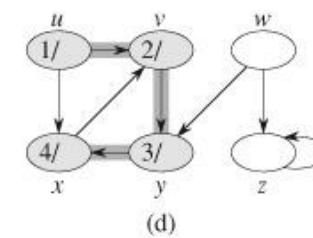
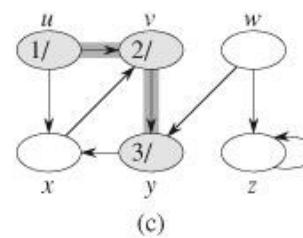
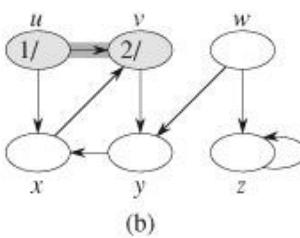
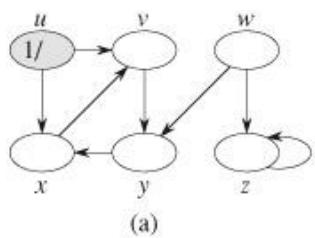












DFS(G)

```
{   for each u in V
    {   color[u]=W; pred[u]=NIL; }
time=0;
for each u in V
  if(color[u]==W) DFSVisit(u);
}
```

DFSVisit(u)

```
{   color[u]=G;
d[u]=++time;
for each v in adj[u]
  if(color[v]==W)
    {   pred[v]=u; DFSVisit(v);}
color[u]=B;
f[u]=++time;
}
```

طبقه بندی یال ها

فرادرس

فرادرس



طبقه بندی یال ها

یال ها را می توان به وسیله جستجوی اول عمق(DFS) به ۴ رده، طبقه بندی کرد.

۱- **درختی (tree edge)** : یالی که DFS آن را ملاقات می کند.

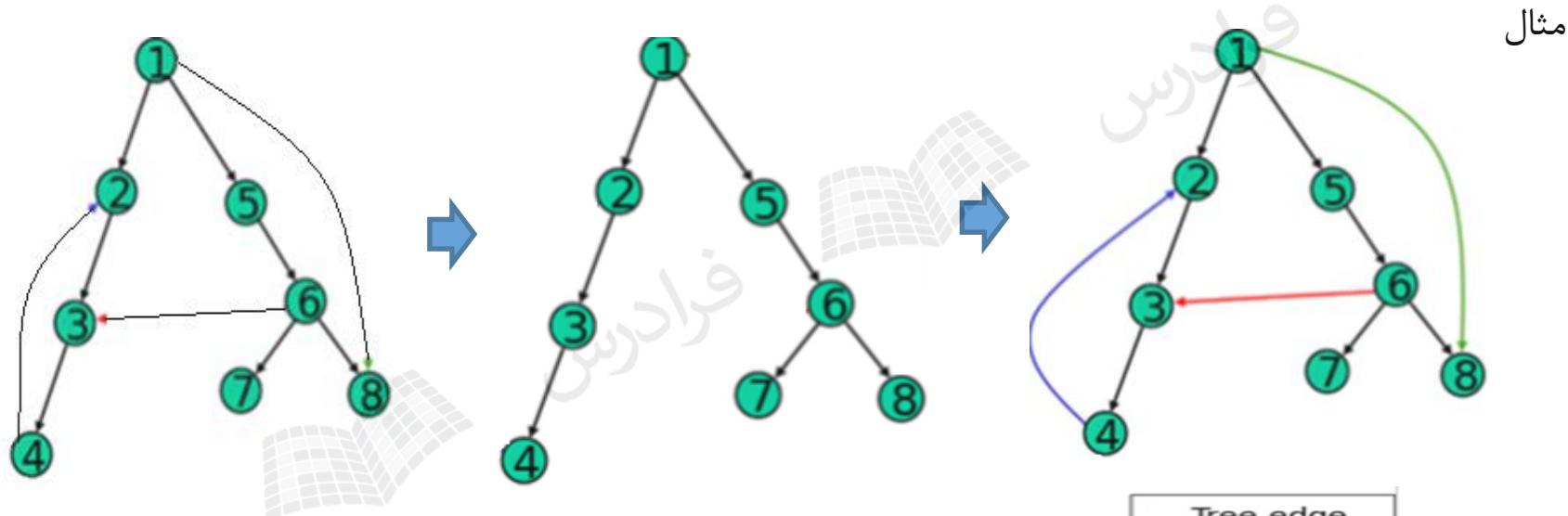
۲- **جلورو (forward edge)** : یال (a,b) که DFS آن را ملاقات نمی کند و در جنگل حاصل، b نواده a است.

۳- **برگشتی (back edge)** : یال (a,b) که DFS آن را ملاقات نمی کند و در جنگل حاصل، a نواده b است.

۴- **تقاطعی (cross edge)** : یال هایی که در ۳ دسته بندی بالا نباشند.

تذکر: به یال برگشتی، یال پس رو و به یال تقاطعی، یال دو طرفه نیز می گویند.

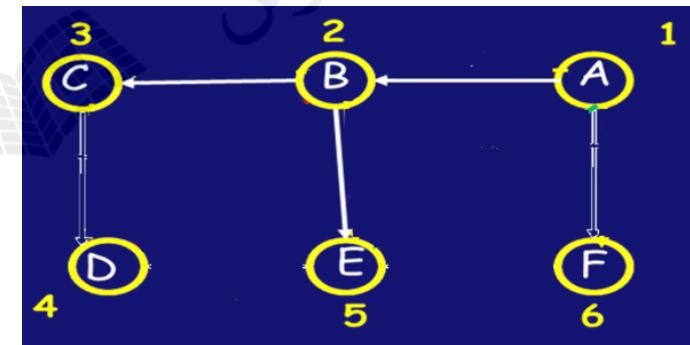
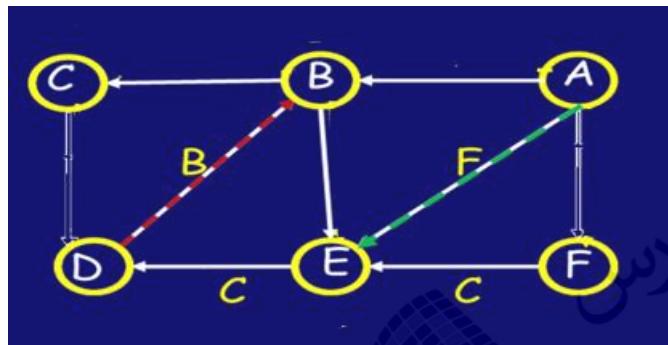
مثال



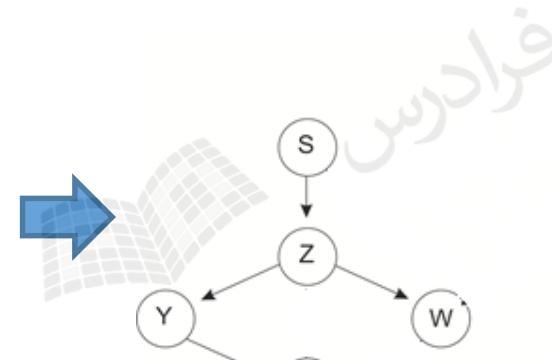
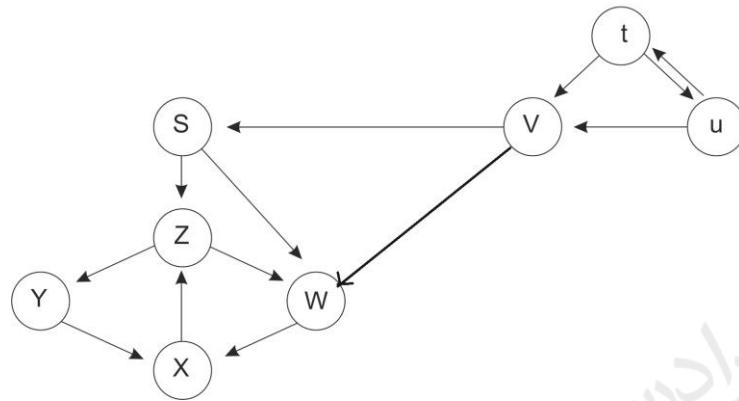
| |
|--------------|
| Tree edge |
| Back edge |
| Forward edge |
| Cross edge |

مثال

مشخص کردن نوع یالها بر اساس پیمایش عمقی



مثال

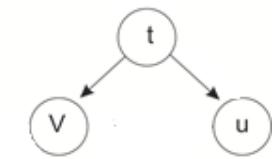


tree edge : (s,z) , (z,y) , (y,x) , (z,w) , (t,v) , (t,u)

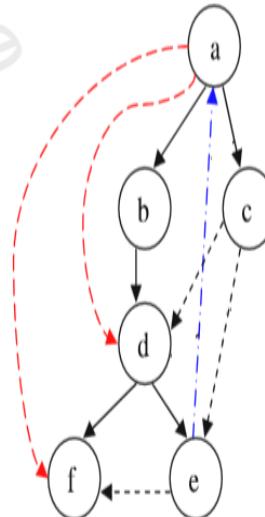
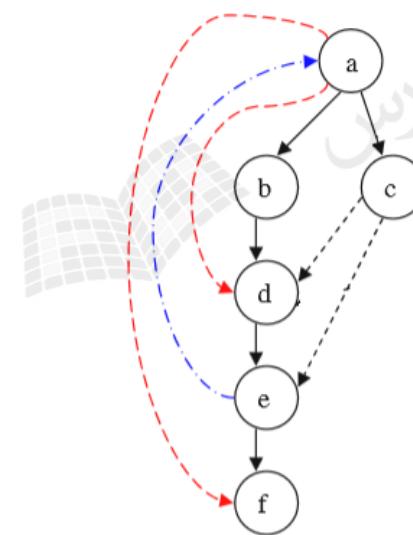
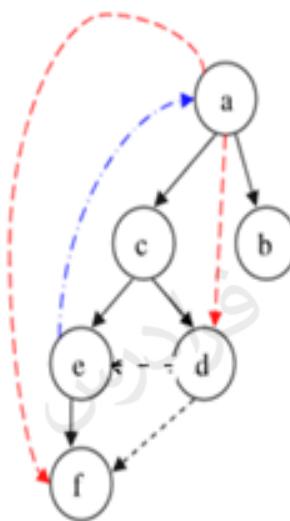
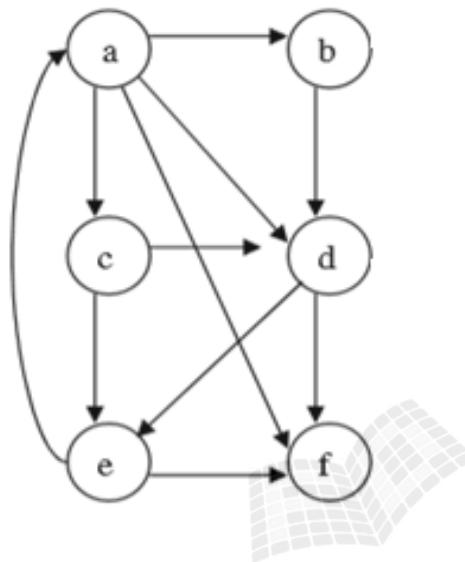
forward edge : (s,w)

back edge : (x,z) , (u,t)

cross edge : (w,x) , (v,w) , (v,s) , (u,v)



مثال



- Tree edge
- ↔ Cross edge
- Forward edge
- Back edge

نکات

در جستجوی اول عمق یک گراف جهت دار :

$$d_u < d_v < f_v < f_u$$

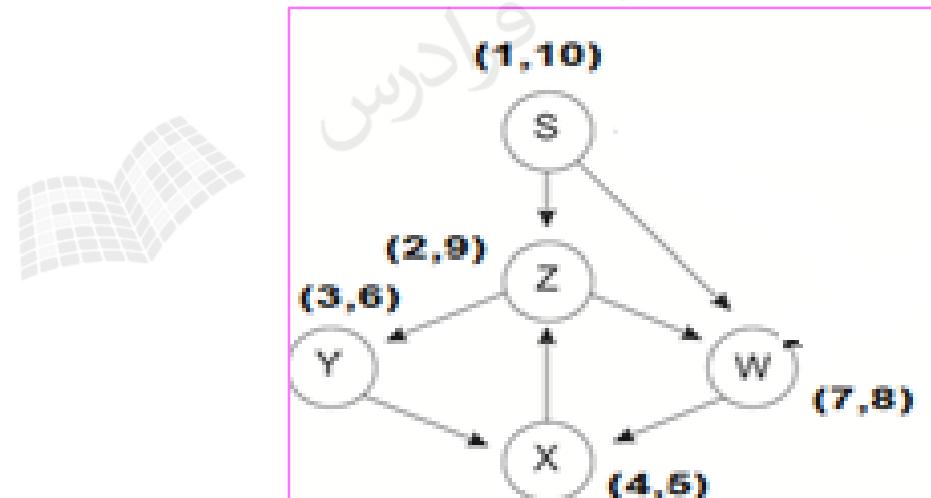
۱- یال (u,v) از نوع Forward یا Tree است اگر و فقط اگر :

$$d_v < d_u < f_u < f_v$$

۲- یال (u,v) از نوع Back است اگر و فقط اگر :

$$d_v < f_v < d_u < f_u$$

۳- یال (u,v) از نوع Cross است اگر و فقط اگر :

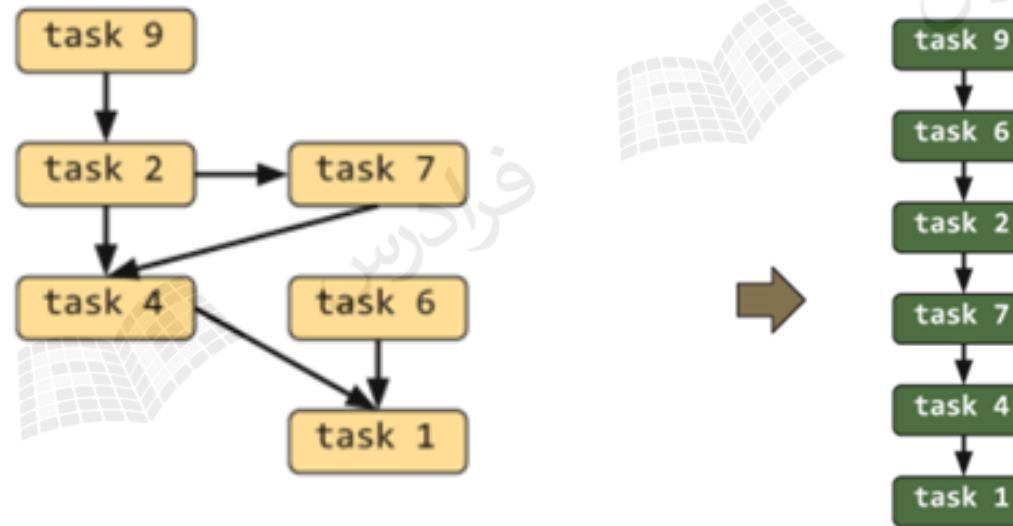


مرتب سازی توپولوژیکی

Topological Sort

مرتب سازی توپولوژیکی

A topological sort of a directed acyclic graph $G = (V,E)$ is a linear ordering of all its vertices such that if G contains an edge (u,v) , then u appears before v in the ordering.



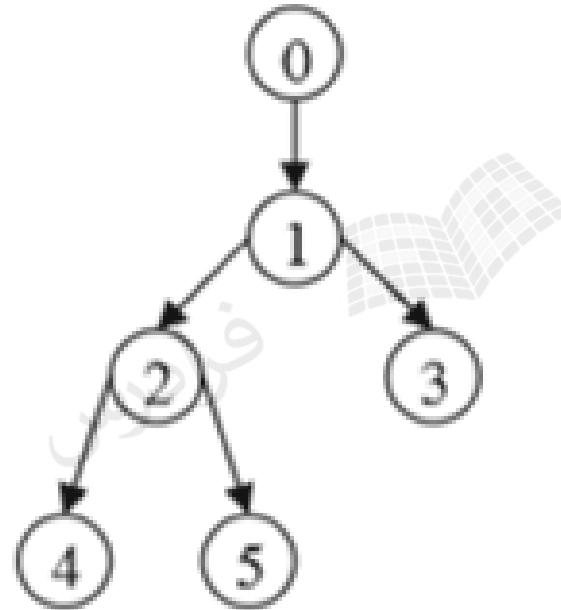
مرتب سازی توپولوژی فقط بر روی DAG قابل اجرا است : (Directed Acyclic Graph)

مثال

0 1 3 2 4 5

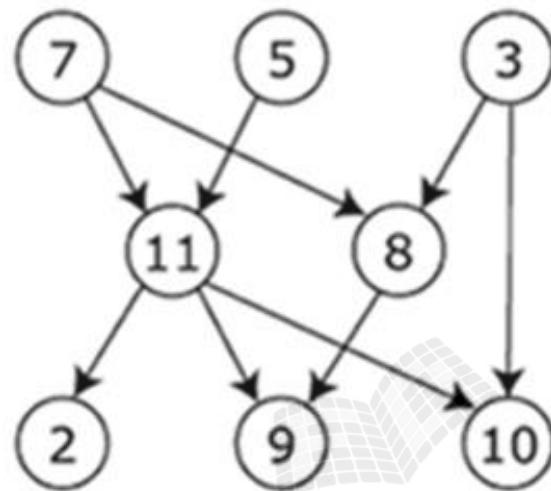
0 1 3 2 5 4

0 1 2 3 4 5



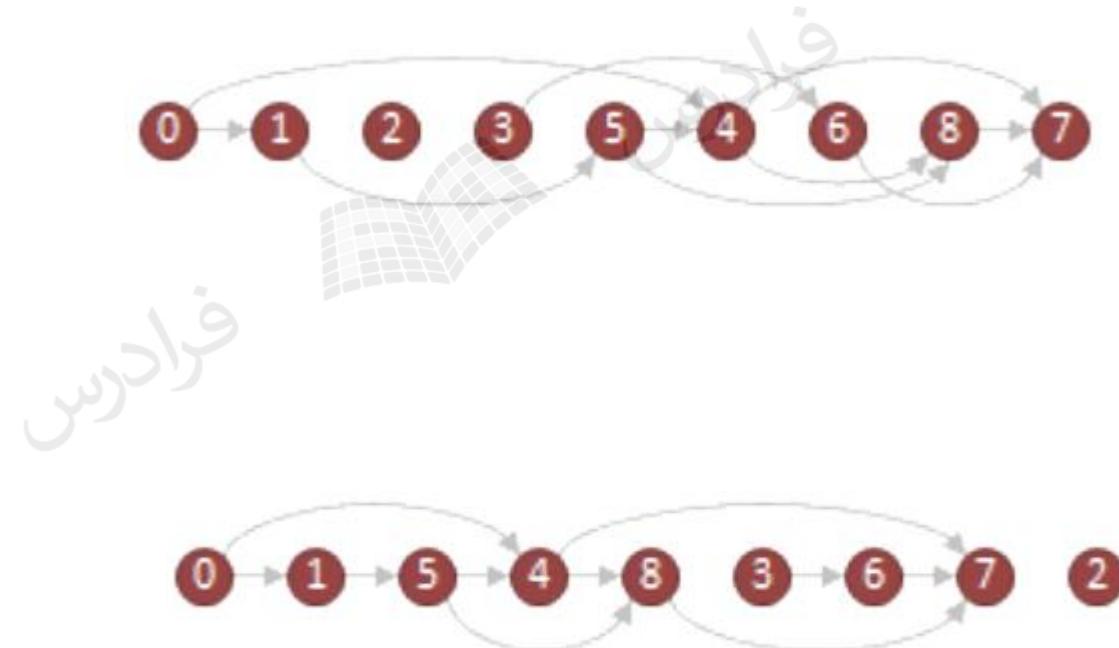
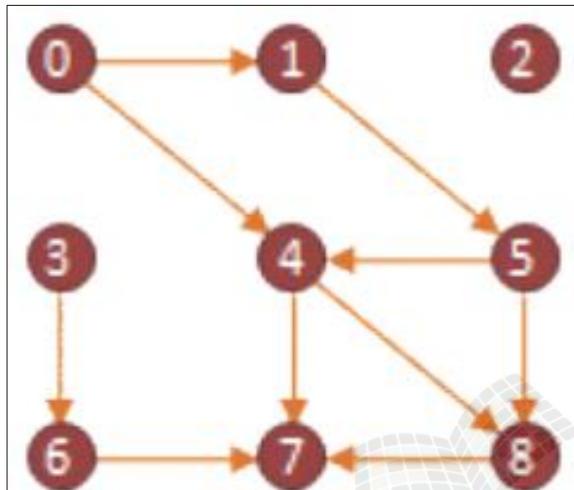
مثال

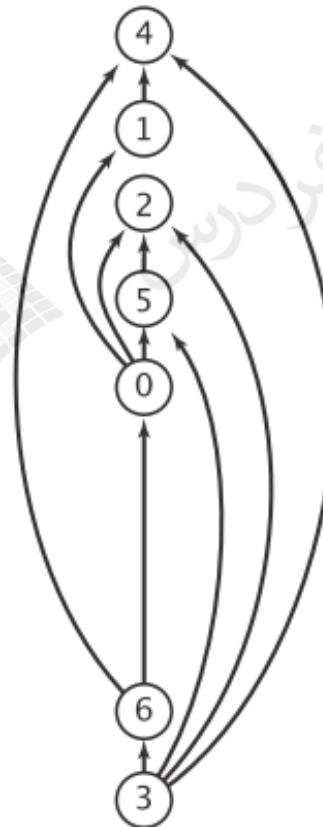
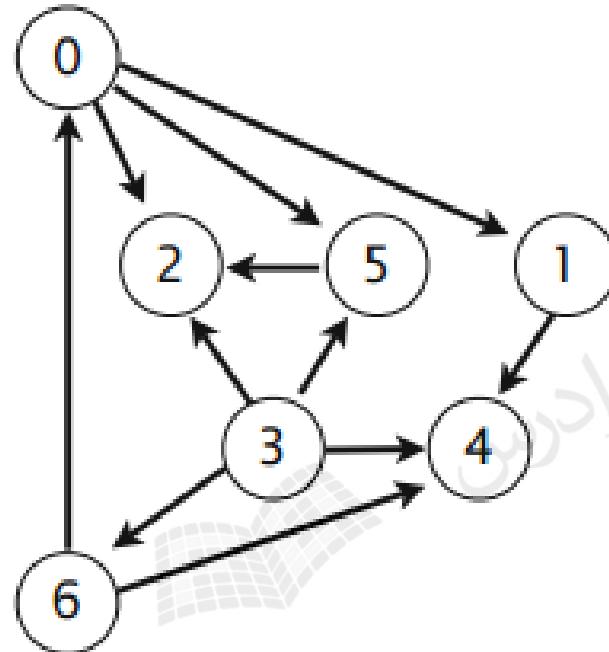
چند ترتیب توپولوژیکی :



- 7, 5, 3, 11, 8, 2, 9, 10
- 3, 5, 7, 8, 11, 2, 9, 10
- 5, 7, 3, 8, 11, 10, 9, 2
- 7, 5, 11, 3, 10, 8, 9, 2
- 7, 5, 11, 2, 3, 8, 9, 10
- 3, 7, 8, 5, 11, 10, 2, 9

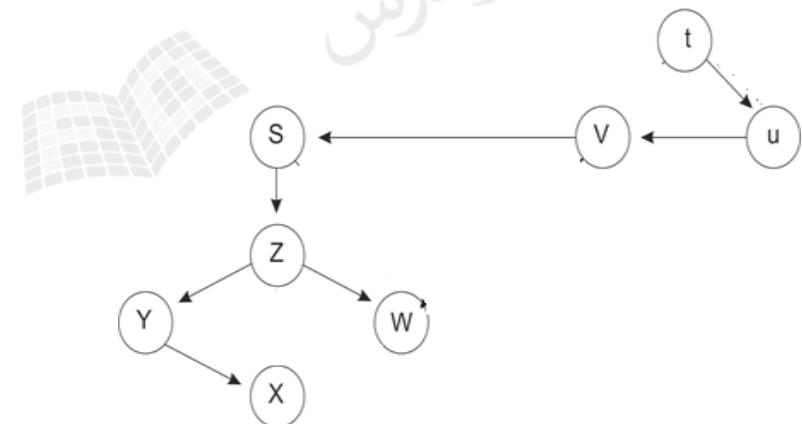
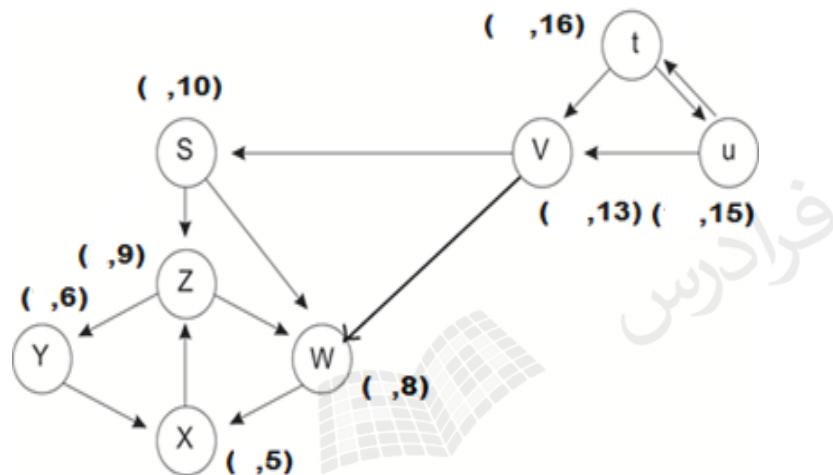
مثال





مرتب سازی توپولوژیکی (بر اساس زمان خاتمه)

گره ها را به ترتیب نزولی زمان **خاتمه** ملاقات می کنیم.



t u v s z w y x

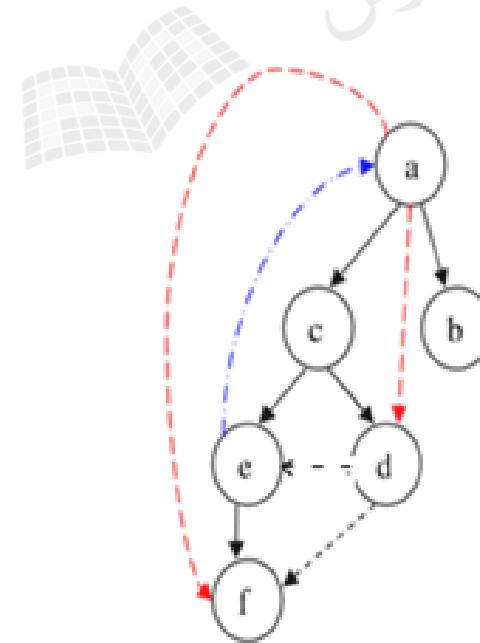
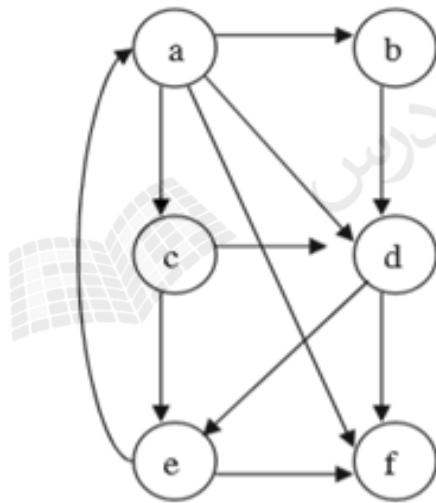
الگوریتم مرتب سازی توپولوژیکی

Topological-Sort(G)

- 1 call DFS(G) to compute **finishing times** $f[v]$ for each vertex v
- 2 as each vertex is finished, insert it onto the front of a linked list
- 3 return the linked list of vertices

لهم

A directed graph G is **acyclic** if and only if a depth-first search of G yields no back edges.



قضیه

TOPOLOGICAL-SORT(G) produces a topological sort of a DAG G .

اثبات

کافی است نشان دهیم به ازای هر یال (u, v) ، داریم : $f[v] < f[u]$ ،
یعنی v زودتر از u نوشته می شود.

وقتی dfs را اجرا می کنیم، u را زودتر از v دیده ایم و v نمی تواند خاکستری باشد. چون اگر خاکستری باشد ،
یال (u, v) از نوع برگشتی خواهد بود که طبق لم قبل ممکن نیست.
یعنی v یا سفید است و یا مشکی.

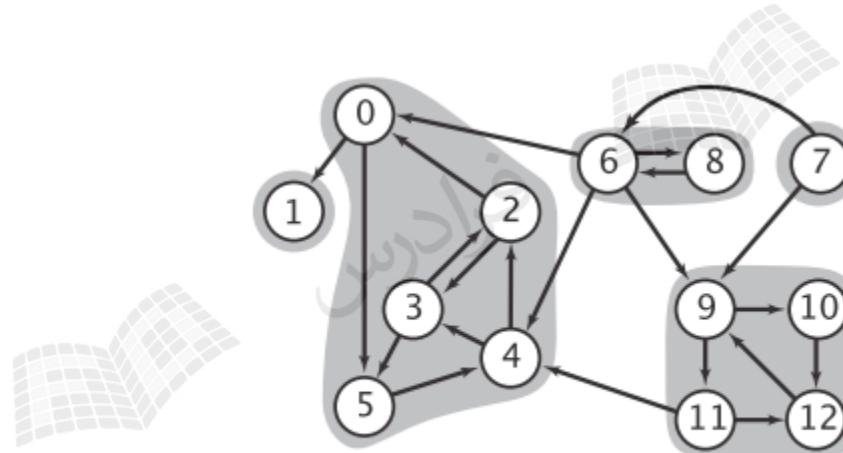
اگر v سفید باشد ، آنگاه نسلی از u است و در نتیجه $f[v] < f[u]$

اگر v مشکی باشد ، آنگاه قبلا از یک مسیر دیگر v پیمايش شده و زمان خاتمه آن set شده و $f[v] < f[u]$.

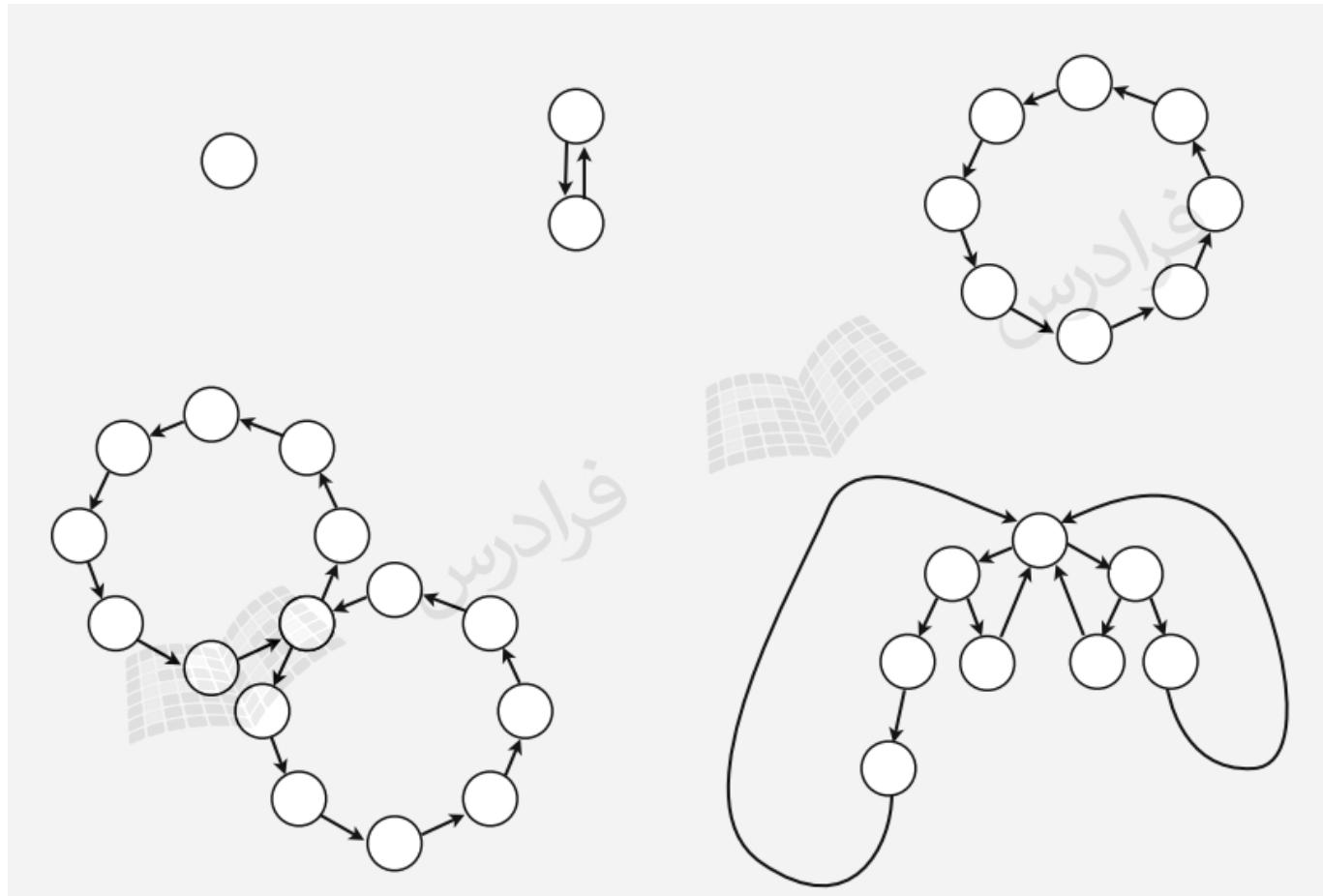
مولفه های همبند قوی

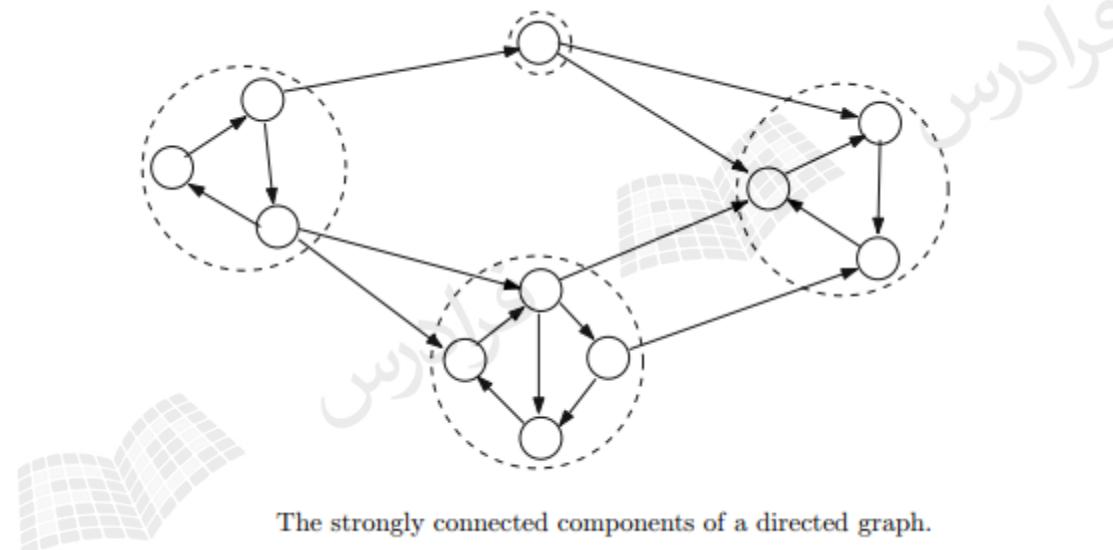
Strongly Connected Components(SCC)

A strongly connected component of a directed graph $G = (V, E)$ is a maximal set of vertices $C \subset V$ such that for every pair of vertices u and v in C , we have both $u \rightsquigarrow v$ and $v \rightsquigarrow u$.



5 strongly-connected components

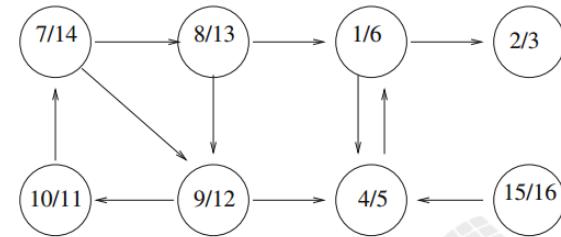
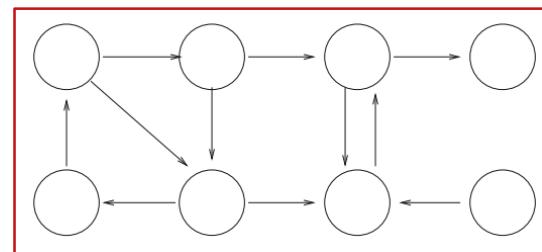




Strongly-Connected-Components(G)

- 1 call $\text{DFS}(G)$ to compute finishing times $f[u]$ for each vertex u
- 2 compute G^T
- 3 call $\text{DFS}(G^T)$, but in the main loop of DFS, consider the vertices
in order of decreasing $f[u]$ (as computed in line 1)
- 4 output the vertices of each tree in the depth-first forest formed in line 3 as a
separate strongly connected component

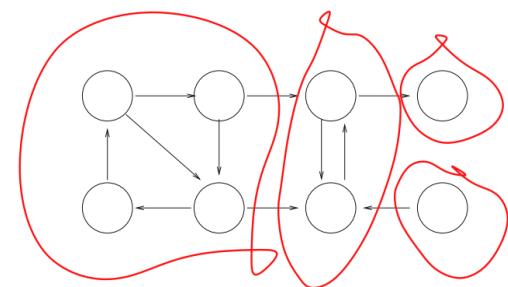
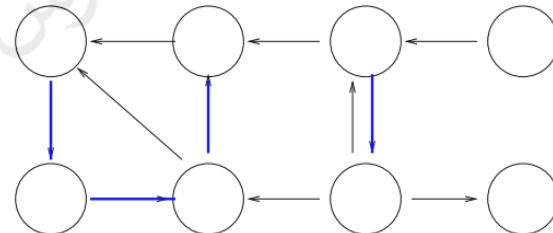
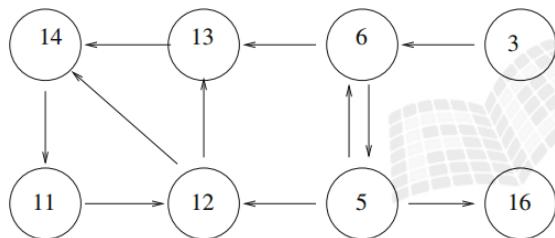
$$\Theta(V + E)$$

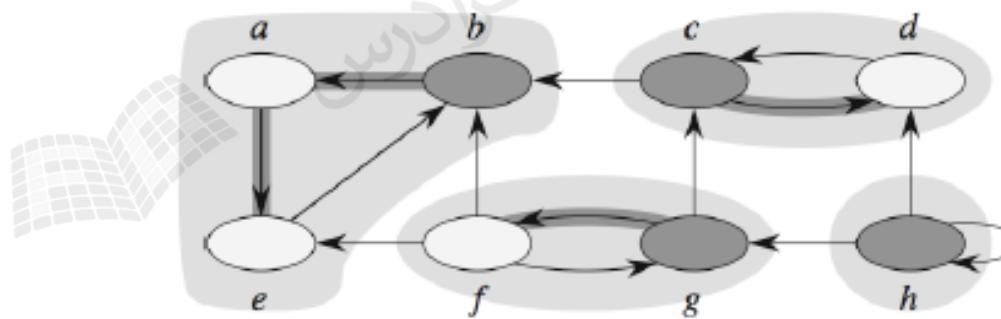
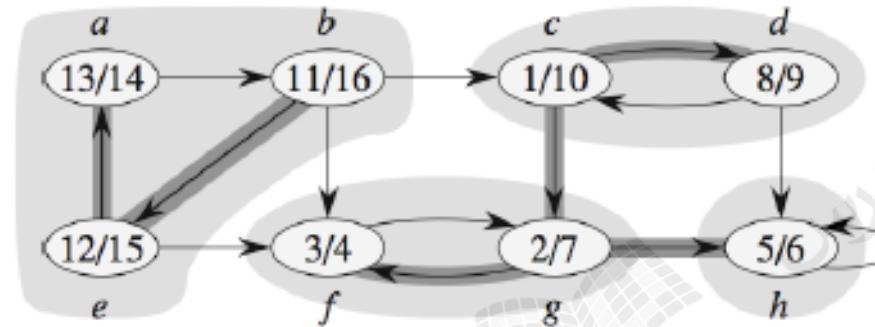


فراز

G^T

DFS in G^T





کوتاهترین مسیر

Shortest Paths

Single-Source All Destination :

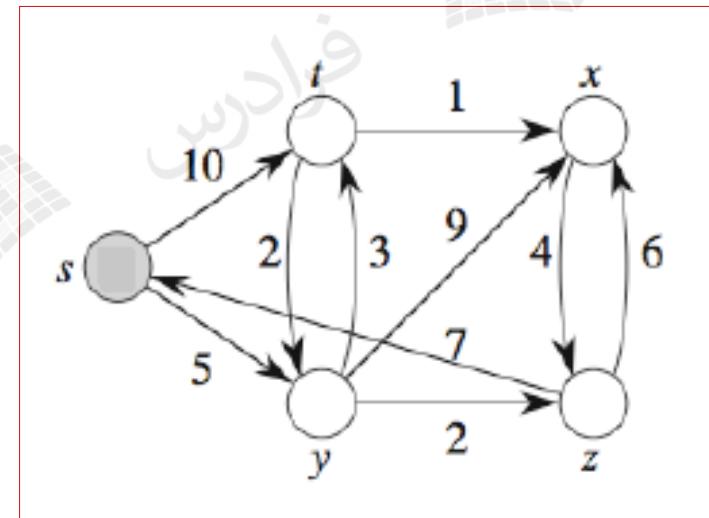
Dijkstra

Bellman-Ford

All-Pairs :

Matrix Multiplication

Floyd-Warshall



کوتاهترین مسیر

The weight of path $p = \langle v_0, v_1, \dots, v_k \rangle$

$$w(p) = \sum_{i=1}^k w(v_{i-1}, v_i).$$

shortest path weight from u to v

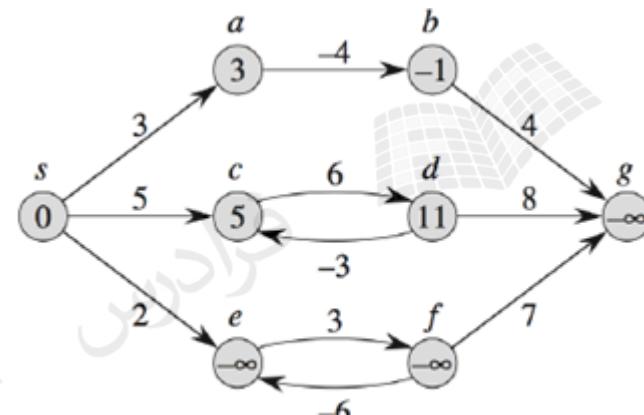
$$\delta(u, v) = \begin{cases} \min\{w(p) : u \xrightarrow{p} v\} & \text{if there is a path from } u \text{ to } v, \\ \infty & \text{otherwise.} \end{cases}$$

shortest path from vertex u to vertex v:

any path p with weight $w(p) = \delta(u, v).$

یال با وزن منفی

If $G = (V, E)$ contains no negative-weight cycles reachable from s , then for all v , the shortest path weight $\delta(s, v)$ remains **well defined**.



not well defined and we assume $\delta(s, v) = -\infty$.

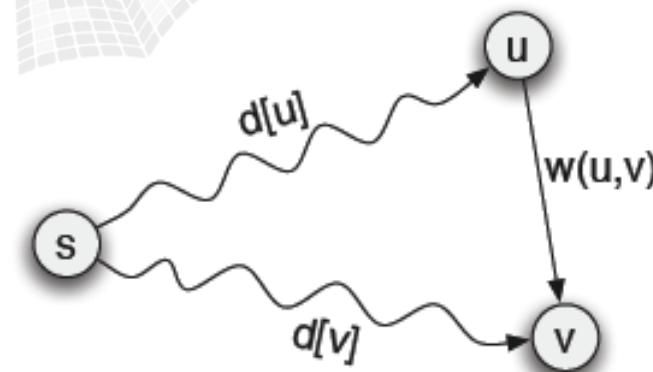
کوتاهترین مسیر شامل یک سیکل نمی تواند باشد.

Relaxation

The process of relaxing an edge (u, v) consists of testing whether we can improve the shortest path to v found so far by going through u and, if so, updating $d[v]$ and $\pi[v]$.

RELAX(u, v, w)

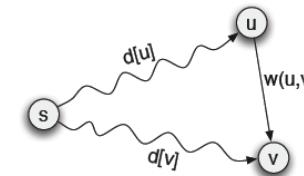
- 1 **if** $d[v] > d[u] + w(u, v)$
- 2 **then** $d[v] \leftarrow d[u] + w(u, v)$
- 3 $\pi[v] \leftarrow u$



خواص (property)

Triangle inequality:

$$\delta(s, v) \leq \delta(s, u) + w(u, v).$$



Upper-bound :

We always have $d[v] \geq \delta(s, v)$ for all vertices $v \in V$, and once $d[v]$ achieves the value $\delta(s, v)$, it never changes.

No-path :

If there is no path from s to v , then we always have $d[v] = \delta(s, v) = \infty$.

Convergence :

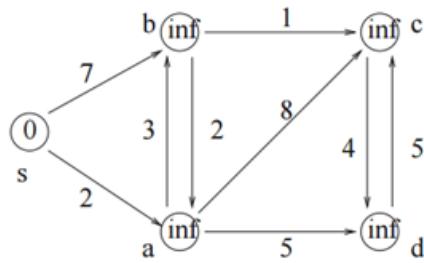


If $s \rightsquigarrow u \rightarrow v$ is a shortest path in G for some $u, v \in V$, and if $d[u] = \delta(s, u)$ at any time prior to relaxing edge (u, v) , then $d[v] = \delta(s, v)$ at all times afterward.

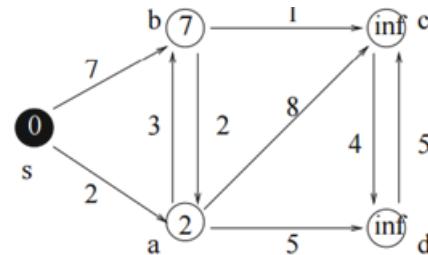
Path-relaxation :

If $p = \langle v_0, v_1, \dots, v_k \rangle$ is a shortest path from $s = v_0$ to v_k , and the edges of p are relaxed in the order $(v_0, v_1), (v_1, v_2), \dots, (v_{k-1}, v_k)$, then $d[v_k] = \delta(s, v_k)$.

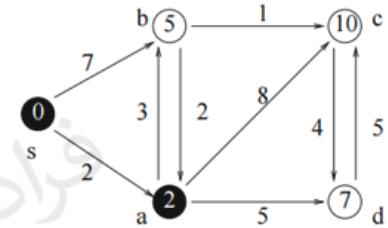
دایکسترا



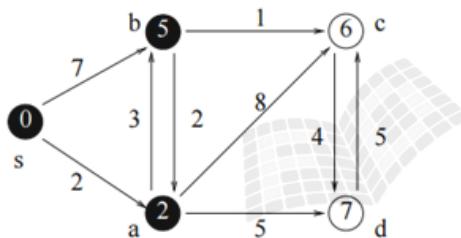
| v | s | a | b | c | d |
|--------|---|----------|----------|----------|----------|
| $d[v]$ | 0 | ∞ | ∞ | ∞ | ∞ |



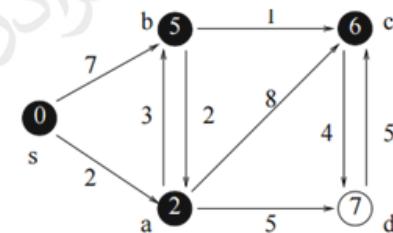
| v | a | b | c | d |
|--------|---|---|----------|----------|
| $d[v]$ | 2 | 7 | ∞ | ∞ |



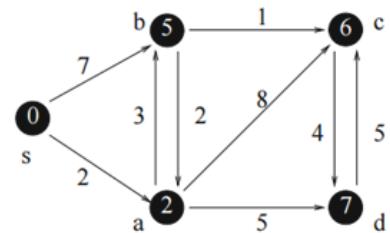
| v | b | c | d |
|--------|---|----|---|
| $d[v]$ | 5 | 10 | 7 |



| v | c | d |
|--------|---|---|
| $d[v]$ | 6 | 7 |



| v | d |
|--------|---|
| $d[v]$ | 7 |



Priority Queue: $Q = \emptyset$.

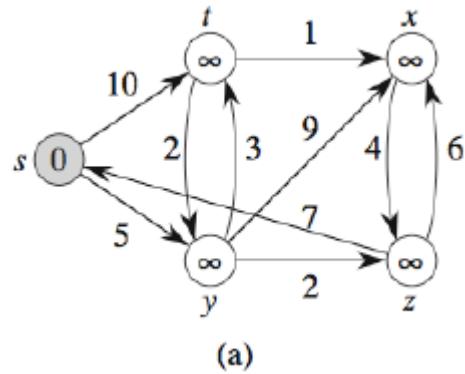
DIJKSTRA(G, w, s)

- 1 INITIALIZE-SINGLE-SOURCE(G, s)
- 2 $S \leftarrow \emptyset$
- 3 $Q \leftarrow V[G]$
- 4 **while** $Q \neq \emptyset$
 - 5 **do** $u \leftarrow \text{EXTRACT-MIN}(Q)$
 - 6 $S \leftarrow S \cup \{u\}$
 - 7 **for each vertex** $v \in \text{Adj}[u]$
 - 8 **do** $\text{RELAX}(u, v, w)$

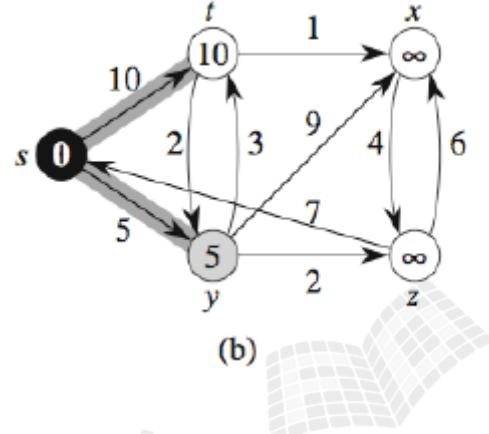
INITIALIZE-SINGLE-SOURCE(G, s)

- 1 **for each vertex** $v \in V[G]$
 - 2 **do** $d[v] \leftarrow \infty$
 - 3 $\pi[v] \leftarrow \text{NIL}$
- 4 $d[s] \leftarrow 0$

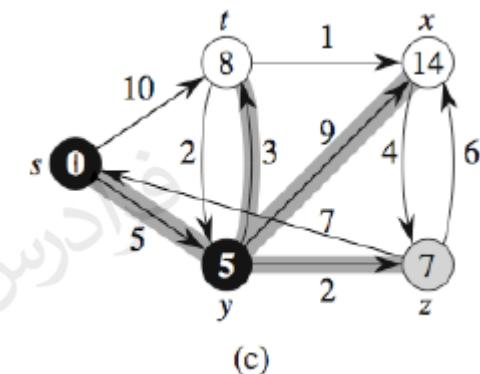
دایکسترا



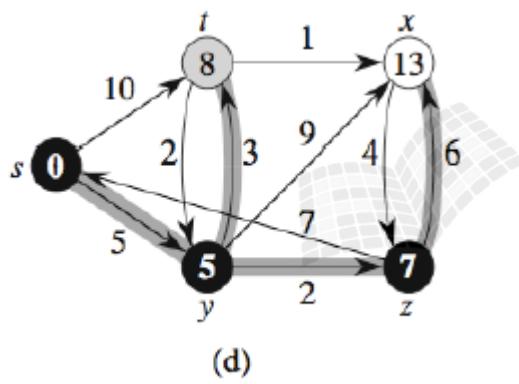
(a)



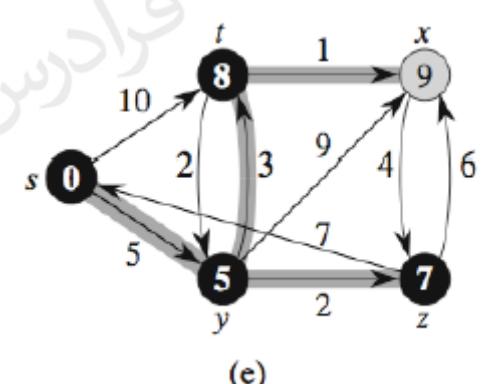
(b)



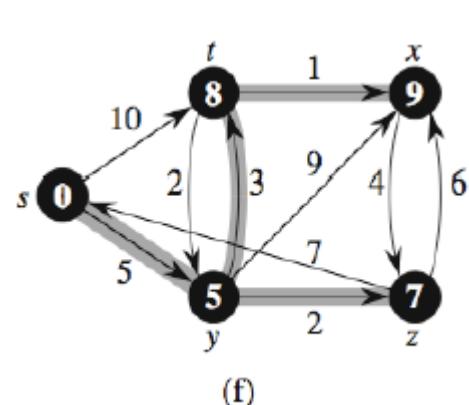
(c)



(d)



(e)



(f)

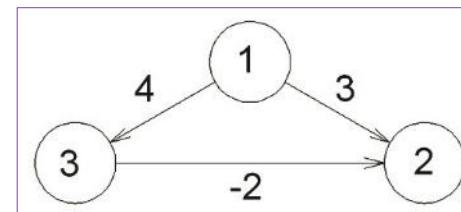
الگوریتم دایکسترا

(کوتاه ترین مسیر تک مبدأ)

الگوریتم **حریصانه** دایکسترا ، از مرتبه $\Theta(n^2)$ است.

الگوریتم دایکسترا همواره کوتاه ترین مسیر را ایجاد می کند.

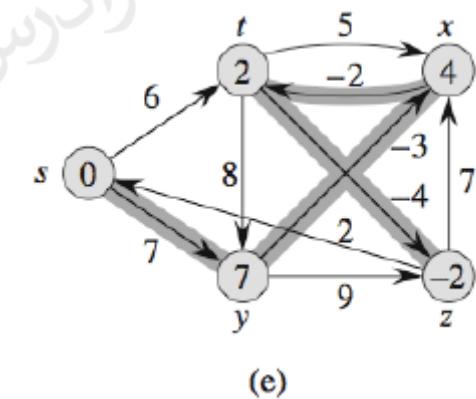
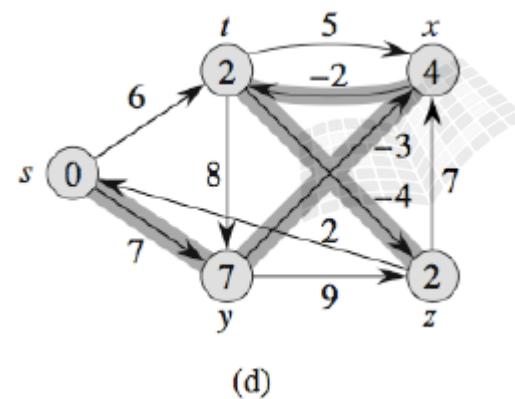
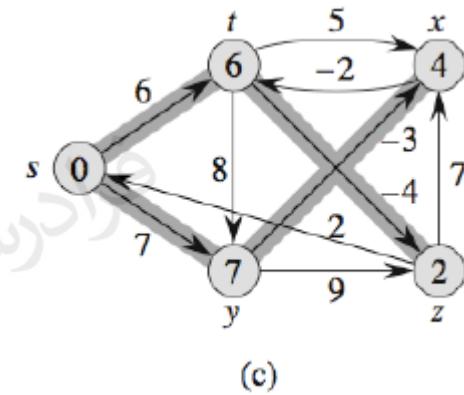
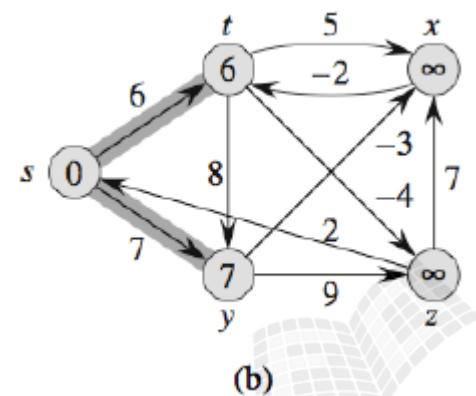
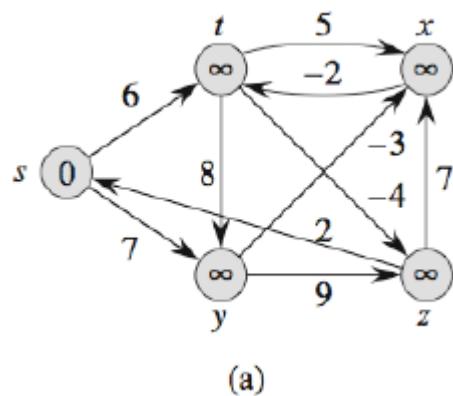
الگوریتم دایکسترا برای گرافی که دارای یال منفی است، همیشه درست کار نمی کند.



بلمن - فورد

Given a weighted directed graph $G = (V,E)$ with source s and weight function $w: E \mapsto R$, the Bellman-Ford algorithm returns a Boolean value indicating whether or not there is a negative-weight cycle that is reachable from the source. If there is such a cycle, the algorithm indicates that no solution exists. Otherwise, the algorithm produces the shortest paths and their weights.

بلمن - فورد



Bellman-Ford algorithm

BELLMAN-FORD(G, w, s)

```

1  INITIALIZE-SINGLE-SOURCE( $G, s$ )
2  for  $i \leftarrow 1$  to  $|V[G]| - 1$ 
3    do for each edge  $(u, v) \in E[G]$ 
4      do RELAX( $u, v, w$ )
5  for each edge  $(u, v) \in E[G]$ 
6    do if  $d[v] > d[u] + w(u, v)$ 
7      then return FALSE
8  return TRUE

```

INITIALIZE-SINGLE-SOURCE(G, s)

```

1  for each vertex  $v \in V[G]$ 
2    do  $d[v] \leftarrow \infty$ 
3     $\pi[v] \leftarrow \text{NIL}$ 
4   $d[s] \leftarrow 0$ 

```

The time complexity of BELLMAN-FORD algorithm is $O(|V| \times |E|)$.

الگوریتم فلوید

تعیین طول کوتاه ترین مسیرها بین هر جفت گره از یک گراف جهت دار.

مسئله کوتاه ترین مسیر یک مسئله بهینه سازی است و با استفاده از برنامه نویسی پویا، آن را حل می کنیم.

برای اینکار ماتریس $D[1..n][1..n]$ را در نظر گرفته و تمام خانه های آنرا مانند ماتریس همچواری در نظر می گیریم و به آن $D^{(0)}$ می گوییم.

سپس به کمک رابطه زیر، $D^{(1)}$ تا $D^{(n)}$ را محاسبه می کنیم. $D^{(n)}$ جواب است.

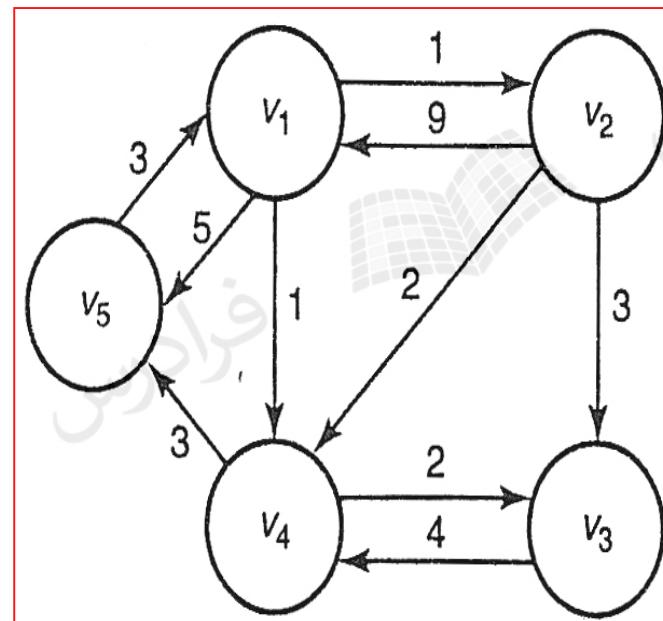
$$D^{(k)}[i][j] = \min(D^{(k-1)}[i][j], D^{(k-1)}[i][k] + D^{(k-1)}[k][j])$$

مثال

تعیین کوتاهترین مسیر بین هر جفت گره.

| 1 | 2 | 3 | 4 | 5 | |
|---|----------|----------|----------|----------|----------|
| 1 | 0 | 1 | ∞ | 1 | 5 |
| 2 | 9 | 0 | 3 | 2 | ∞ |
| 3 | ∞ | ∞ | 0 | 4 | ∞ |
| 4 | ∞ | ∞ | 2 | 0 | 3 |
| 5 | 3 | ∞ | ∞ | ∞ | 0 |

ماتریس همچواری



| 1 | 2 | 3 | 4 | 5 | |
|---|----|----|---|---|---|
| 1 | 0 | 1 | 3 | 1 | 4 |
| 2 | 8 | 0 | 3 | 2 | 5 |
| 3 | 10 | 11 | 0 | 4 | 7 |
| 4 | 6 | 7 | 2 | 0 | 3 |
| 5 | 3 | 4 | 6 | 4 | 0 |

جواب

مراحل پیدا کردن کوتاهترین مسیر بین گره ۲ و ۵ :

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

$$D^{(1)}[2][5] = \min(D^{(0)}[2][5], D^{(0)}[2][1] + D^{(0)}[1][5]) = \min(\infty, 9 + 5) = 14$$

$$D^{(2)}[2][5] = \min(D^{(1)}[2][5], D^{(1)}[2][2] + D^{(1)}[2][5]) = \min(14, 0 + 14) = 14$$

$$D^{(3)}[2][5] = \min(D^{(2)}[2][5], D^{(2)}[2][3] + D^{(2)}[3][5]) = \min(14, 3 + \infty) = 14$$

$$D^{(4)}[2][5] = \min(D^{(3)}[2][5], D^{(3)}[2][4] + D^{(3)}[4][5]) = \min(14, 2 + 3) = 5$$

$$D^{(5)}[2][5] = \min(D^{(4)}[2][5], D^{(4)}[2][5] + D^{(4)}[5][5]) = \min(5, 5 + 0) = 5$$

بنابراین طول کوتاه ترین مسیر از گره ۲ به ۵ برابر ۵ است. (از ۲ به ۴ رفته و سپس از ۴ به ۵ می رویم.)

روابط زیر برقرار است:

$$D^{(k)}[i][k] = D^{(k-1)}[i][k]$$

$$D^{(k)}[k][j] = D^{(k-1)}[k][j]$$

الگوریتم فلوید

```
floyd ( n , W[ ][ ] , D[ ][ ] , P[ ][ ] ){
    for( i=1; i<= n ; i++)
        for( j=1; j<= n ; j++)
            P[i][j]=0;
    D=W;
    for( k=1; k<= n ; k++)
        for( i=1; i<= n ; i++)
            for( j=1; j<=n ; j++)
                if ( D[i][k] + D[k][j] < D[i][j] )
{
                P[i][j]=k;
                D[i][j]=D[i][k]+D[k][j];
}
}
```

$$T(n) \in \Theta(n^3)$$

مثال

با توجه به ماتریس P ، مسیر بهینه برای رفتن از گره ۵ به گره ۳ را مشخص کنید.

| | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | 0 | 0 | 4 | 0 | 4 |
| 2 | 5 | 0 | 0 | 0 | 4 |
| 3 | 5 | 5 | 0 | 0 | 4 |
| 4 | 5 | 5 | 0 | 0 | 0 |
| 5 | 0 | 1 | 4 | 1 | 0 |

$5 \rightarrow \dots \rightarrow 4 \rightarrow \dots \rightarrow 3$

$5 \rightarrow \dots \rightarrow 1 \rightarrow \dots \rightarrow 4 \rightarrow \dots \rightarrow 3$

$5 \rightarrow 1 \rightarrow 4 \rightarrow 3$

مثال

در الگوریتم فلوید فرض کنیم که برای گرافی با ۴ گره ماتریس P به شکل زیر باشد. مسیر بهینه برای رفتن از گره ۱ به گره ۳ را مشخص کنید:

$$P = \begin{bmatrix} 0 & 0 & 4 & 2 \\ 4 & 0 & 4 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

$1 \rightarrow \dots \rightarrow 4 \rightarrow \dots \rightarrow 3$

$1 \rightarrow 2 \rightarrow 4 \rightarrow \dots \rightarrow 3$

$1 \rightarrow 2 \rightarrow 4 \rightarrow 3$

All-Pairs Shortest Paths

Matrix Multiplication

Matrix Multiplication

هزینه کوتاهترین مسیر از گره i به گره j که حداقل m یال دارد:

$m = 0$ مجاز نیستم یالی بین دو راس قرار دهم:

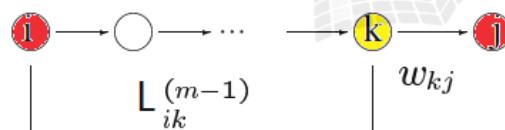
$m \geq 1$ مجاز هستم حداقل یک یال بین دو راس قرار دهم:

$$l_{ij}^{(0)} = \begin{cases} 0 & \text{if } i = j, \\ \infty & \text{if } i \neq j. \end{cases}$$

$$l_{ij}^{(m)} = \min \left(l_{ij}^{(m-1)}, \min_{1 \leq k \leq n} \{ l_{ik}^{(m-1)} + w_{kj} \} \right)$$

چون k می تواند برابر j شود، می توان قسمت اول را حذف کرد.

$$l_{ij}^{(m)} = \min_{1 \leq k \leq n} \{ l_{ik}^{(m-1)} + w_{kj} \}$$

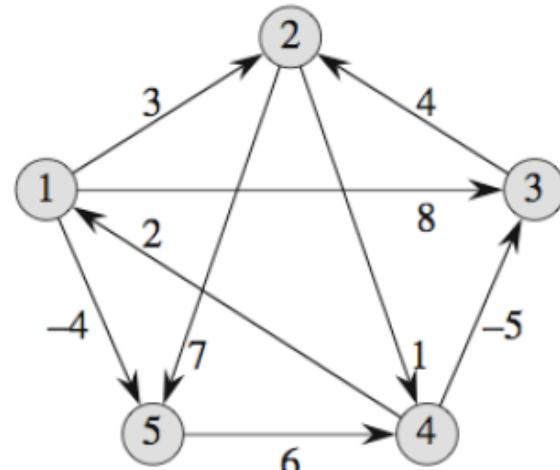


با حداقل $m-1$ یال از i به k رفته و سپس با آن یک یال از k به j می رویم.

$$\delta(i, j) = l_{ij}^{(n-1)} = l_{ij}^{(n)} = l_{ij}^{(n+1)} = \dots$$

کوتاهترین مسیر بین i و j حداقل شامل $n-1$ یال است.
همچنین هر مسیر طولانی تری، وزن کمتری ندارد.

Matrix Multiplication



$$L^{(1)} = \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \infty & -5 & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \quad L^{(2)} = \begin{pmatrix} 0 & 3 & 8 & \textcircled{2} & -4 \\ 3 & 0 & -4 & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & \infty & 1 & 6 & 0 \end{pmatrix}$$

پیدا کردن 2

$$\begin{pmatrix} 0 & 3 & 8 & \infty & -4 \end{pmatrix} \begin{pmatrix} \infty \\ 1 \\ \infty \\ 0 \\ 6 \end{pmatrix} \rightarrow \begin{pmatrix} \infty \\ 4 \\ \infty \\ \infty \\ 2 \end{pmatrix}$$

مشابه ضرب ماتریس ها است . کافی است به جای ضرب از جمع استفاده کرد و به جای جمع از \min .

$$L^{(3)} = \begin{pmatrix} 0 & 3 & -3 & 2 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 11 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix}$$

$$L^{(4)} = \begin{pmatrix} 0 & 1 & -3 & 2 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix}$$

$$\left\{ \begin{array}{l} \cdot \rightarrow + \\ + \rightarrow \min \end{array} \right.$$

Matrix Multiplication

SLOW-ALL-PAIRS-SHORTEST-PATHS(W)

```

1   $n \leftarrow \text{rows}[W]$ 
2   $L^{(1)} \leftarrow W$ 
3  for  $m \leftarrow 2$  to  $n - 1$ 
4      do  $L^{(m)} \leftarrow \text{EXTEND-SHORTEST-PATHS}(L^{(m-1)}, W)$ 
5  return  $L^{(n-1)}$ 

```

EXTEND-SHORTEST-PATHS(L, W)

```

1   $n \leftarrow \text{rows}[L]$ 
2  let  $L' = (l'_{ij})$  be an  $n \times n$  matrix
3  for  $i \leftarrow 1$  to  $n$ 
4      do for  $j \leftarrow 1$  to  $n$ 
5          do  $l'_{ij} \leftarrow \infty$ 
6              for  $k \leftarrow 1$  to  $n$ 
7                  do  $l'_{ij} \leftarrow \min(l'_{ij}, l_{ik} + w_{kj})$ 
8  return  $L'$ 

```

باید یک سری از ماتریس های L را محاسبه کرد.

$$\begin{aligned}
 L^{(1)} &= L^{(0)} \cdot W = W, \\
 L^{(2)} &= L^{(1)} \cdot W = W^2, \\
 L^{(3)} &= L^{(2)} \cdot W = W^3, \\
 &\vdots \\
 L^{(n-1)} &= L^{(n-2)} \cdot W = W^{n-1}
 \end{aligned}$$

پیچیدگی زمانی: $O(n^4)$

الگوریتم سریعتر

$$\begin{aligned}
 L^{(1)} &= W, \\
 L^{(2)} &= W^2 = W \cdot W, \\
 L^{(4)} &= W^4 = W^2 \cdot W^2 \\
 L^{(8)} &= W^8 = W^4 \cdot W^4, \\
 &\vdots \\
 L^{(2^{\lceil \lg(n-1) \rceil})} &= W^{2^{\lceil \lg(n-1) \rceil}} = W^{2^{\lceil \lg(n-1) \rceil}-1} \cdot W^{2^{\lceil \lg(n-1) \rceil}-1}
 \end{aligned}$$

FASTER-ALL-PAIRS-SHORTEST-PATHS(W)

```

1   $n \leftarrow \text{rows}[W]$ 
2   $L^{(1)} \leftarrow W$ 
3   $m \leftarrow 1$ 
4  while  $m < n - 1$ 
5      do  $L^{(2m)} \leftarrow \text{EXTEND-SHORTEST-PATHS}(L^{(m)}, L^{(m)})$ 
6       $m \leftarrow 2m$ 
7  return  $L^{(m)}$ 

```

پیچیدگی زمانی:

$O(n^3 \log n)$

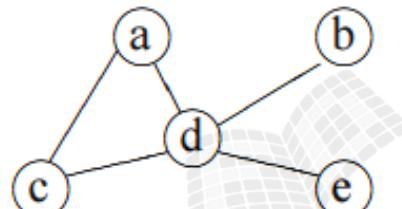
درخت پوشای کمینه

Minimum Spanning Trees(MST)

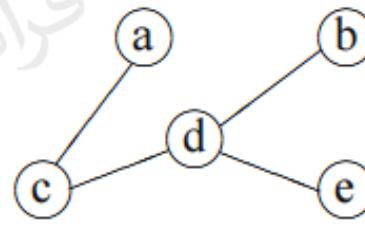
درخت پوشای

درخت پوشای(فراگیر) برای یک گراف:

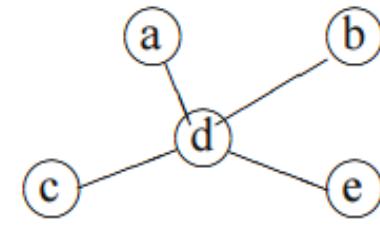
یک زیر گراف متصل است که حاوی همه رئوس موجود در گراف بوده و یک درخت باشد یعنی چرخه نداشته باشد.



Graph

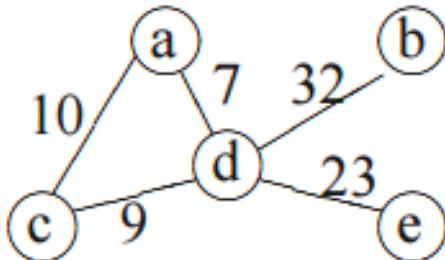


spanning tree 1

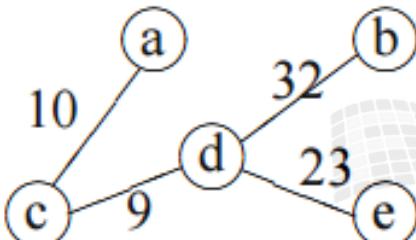
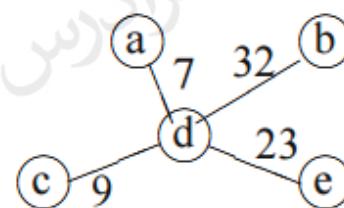


spanning tree 2

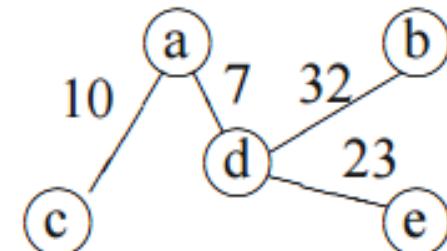
درخت پوشای کمینه



weighted graph

Tree 1. $w=74$ 

Tree 2, $w=71$
Minimum spanning tree

Tree 3, $w=72$

الگوریتم های تعیین درخت پوشای کمینه

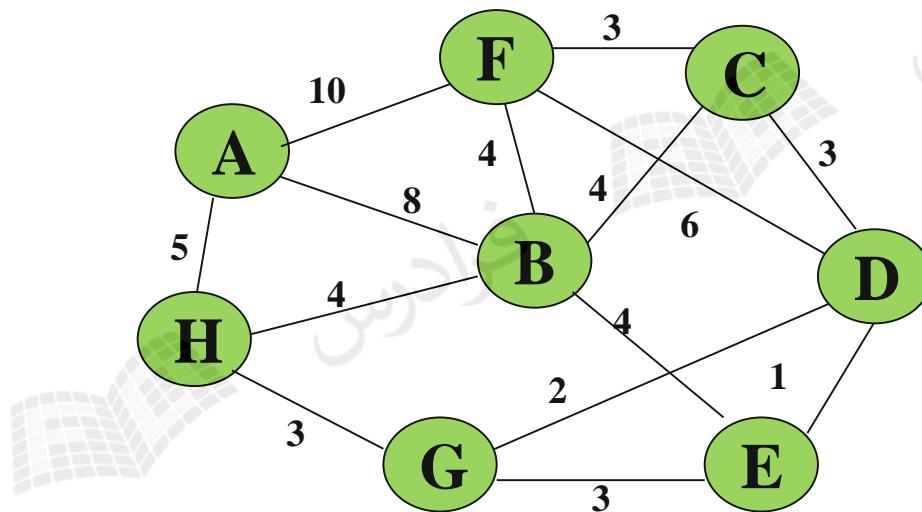
GENERIC-MST(G, w)

- 1 $A \leftarrow \emptyset$
- 2 **while** A does not form a spanning tree
- 3 **do** find an edge (u, v) that is safe for A
- 4 $A \leftarrow A \cup \{(u, v)\}$
- 5 **return** A

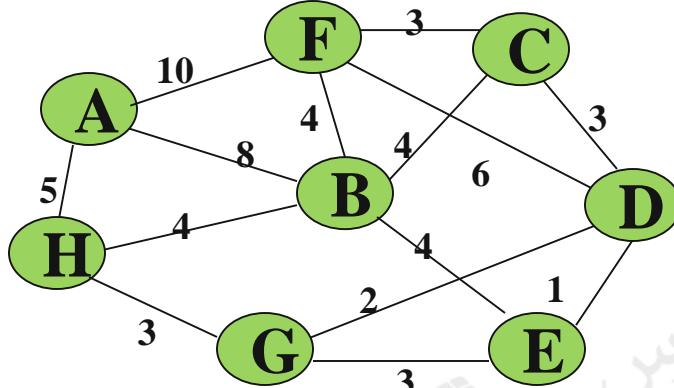
۱ - کروسکال (Kruskal)

۲ - پریم (Prim)

کروسکال

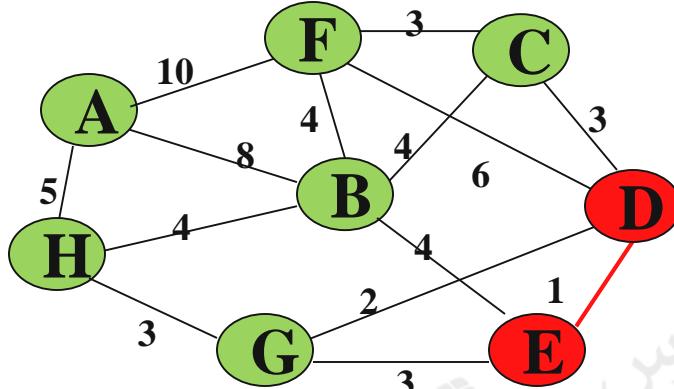


مرتب کردن صعودی یال ها



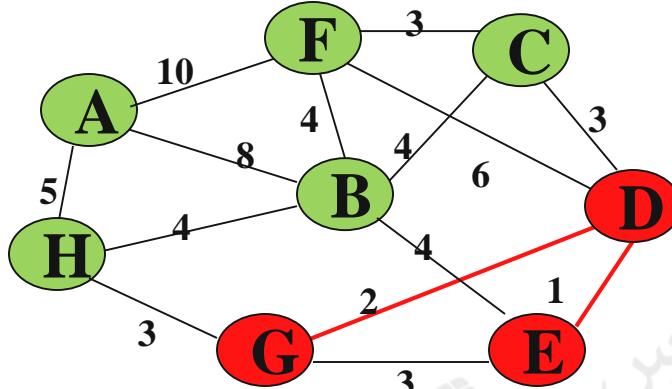
| <i>edge</i> | d_v | |
|-------------|-------|--|
| (D,E) | 1 | |
| (D,G) | 2 | |
| (E,G) | 3 | |
| (C,D) | 3 | |
| (G,H) | 3 | |
| (C,F) | 3 | |
| (B,C) | 4 | |

| <i>edge</i> | d_v | |
|-------------|-------|--|
| (B,E) | 4 | |
| (B,F) | 4 | |
| (B,H) | 4 | |
| (A,H) | 5 | |
| (D,F) | 6 | |
| (A,B) | 8 | |
| (A,F) | 10 | |



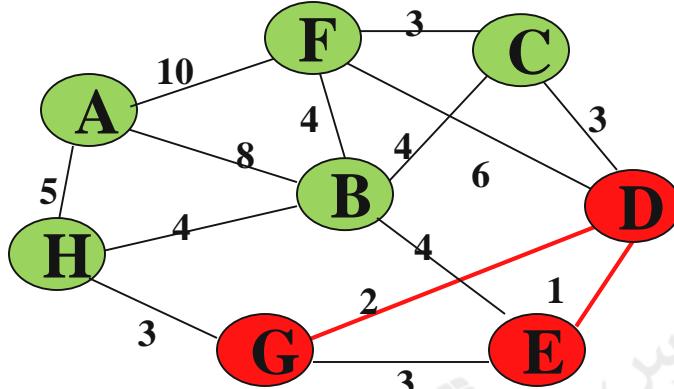
| <i>edge</i> | d_v | |
|-------------|-------|---|
| (D,E) | 1 | ✓ |
| (D,G) | 2 | |
| (E,G) | 3 | |
| (C,D) | 3 | |
| (G,H) | 3 | |
| (C,F) | 3 | |
| (B,C) | 4 | |

| <i>edge</i> | d_v | |
|-------------|-------|--|
| (B,E) | 4 | |
| (B,F) | 4 | |
| (B,H) | 4 | |
| (A,H) | 5 | |
| (D,F) | 6 | |
| (A,B) | 8 | |
| (A,F) | 10 | |



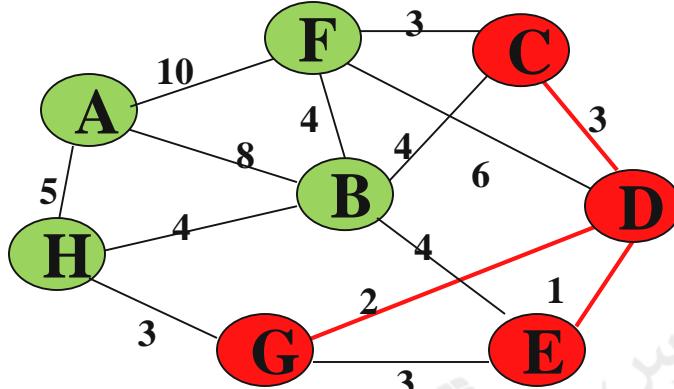
| <i>edge</i> | d_v | |
|-------------|-------|---|
| (D,E) | 1 | ✓ |
| (D,G) | 2 | ✓ |
| (E,G) | 3 | |
| (C,D) | 3 | |
| (G,H) | 3 | |
| (C,F) | 3 | |
| (B,C) | 4 | |

| <i>edge</i> | d_v | |
|-------------|-------|--|
| (B,E) | 4 | |
| (B,F) | 4 | |
| (B,H) | 4 | |
| (A,H) | 5 | |
| (D,F) | 6 | |
| (A,B) | 8 | |
| (A,F) | 10 | |



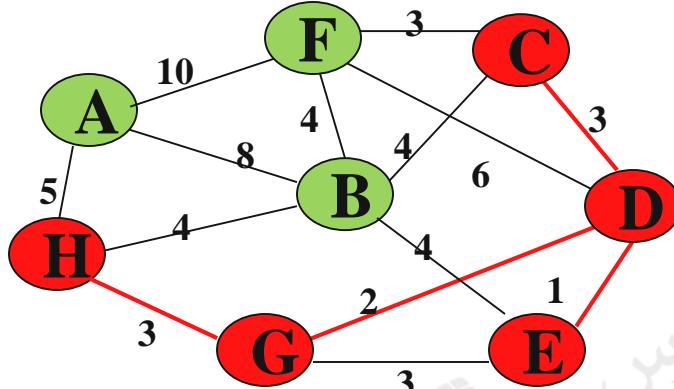
| <i>edge</i> | d_v | |
|-------------|-------|---|
| (D,E) | 1 | ✓ |
| (D,G) | 2 | ✓ |
| (E,G) | 3 | ✗ |
| (C,D) | 3 | |
| (G,H) | 3 | |
| (C,F) | 3 | |
| (B,C) | 4 | |

| <i>edge</i> | d_v | |
|-------------|-------|--|
| (B,E) | 4 | |
| (B,F) | 4 | |
| (B,H) | 4 | |
| (A,H) | 5 | |
| (D,F) | 6 | |
| (A,B) | 8 | |
| (A,F) | 10 | |



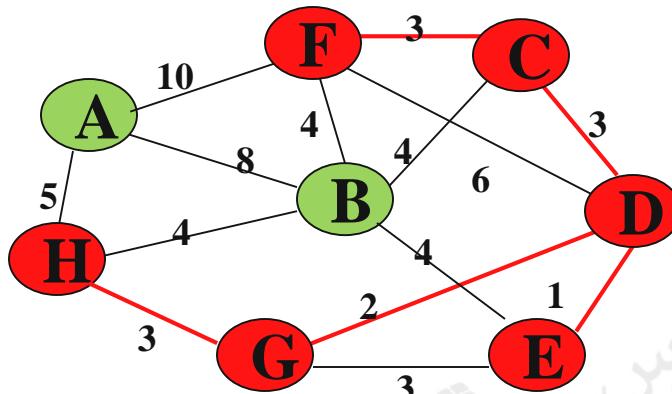
| <i>edge</i> | d_v | |
|-------------|-------|---|
| (D,E) | 1 | ✓ |
| (D,G) | 2 | ✓ |
| (E,G) | 3 | ✗ |
| (C,D) | 3 | ✓ |
| (G,H) | 3 | |
| (C,F) | 3 | |
| (B,C) | 4 | |

| <i>edge</i> | d_v | |
|-------------|-------|--|
| (B,E) | 4 | |
| (B,F) | 4 | |
| (B,H) | 4 | |
| (A,H) | 5 | |
| (D,F) | 6 | |
| (A,B) | 8 | |
| (A,F) | 10 | |



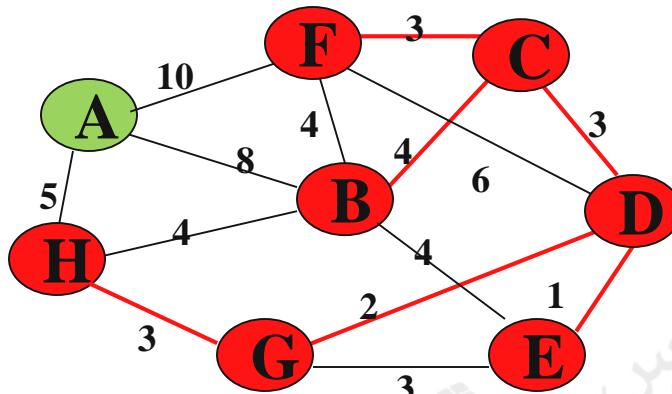
| <i>edge</i> | d_v | |
|-------------|-------|---|
| (D,E) | 1 | ✓ |
| (D,G) | 2 | ✓ |
| (E,G) | 3 | ✗ |
| (C,D) | 3 | ✓ |
| (G,H) | 3 | ✓ |
| (C,F) | 3 | |
| (B,C) | 4 | |

| <i>edge</i> | d_v | |
|-------------|-------|--|
| (B,E) | 4 | |
| (B,F) | 4 | |
| (B,H) | 4 | |
| (A,H) | 5 | |
| (D,F) | 6 | |
| (A,B) | 8 | |
| (A,F) | 10 | |



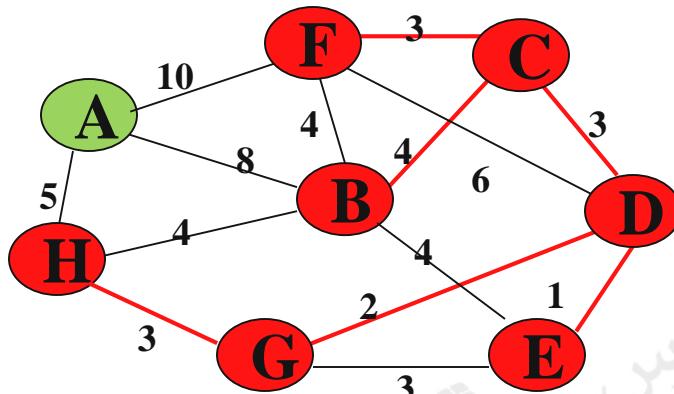
| <i>edge</i> | d_v | |
|-------------|-------|---|
| (D,E) | 1 | ✓ |
| (D,G) | 2 | ✓ |
| (E,G) | 3 | ✗ |
| (C,D) | 3 | ✓ |
| (G,H) | 3 | ✓ |
| (C,F) | 3 | ✓ |
| (B,C) | 4 | |

| <i>edge</i> | d_v | |
|-------------|-------|--|
| (B,E) | 4 | |
| (B,F) | 4 | |
| (B,H) | 4 | |
| (A,H) | 5 | |
| (D,F) | 6 | |
| (A,B) | 8 | |
| (A,F) | 10 | |



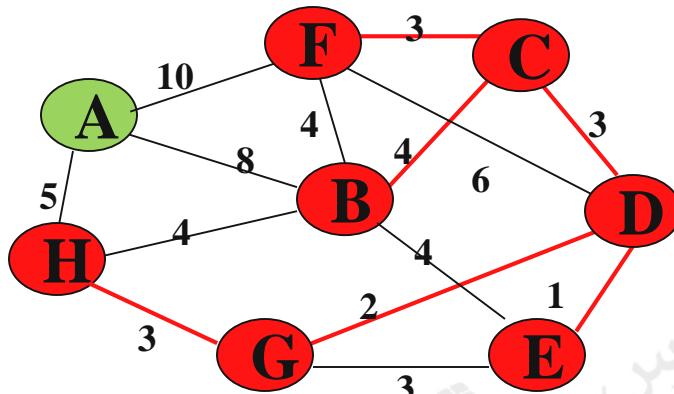
| <i>edge</i> | d_v | |
|-------------|-------|---|
| (D,E) | 1 | ✓ |
| (D,G) | 2 | ✓ |
| (E,G) | 3 | ✗ |
| (C,D) | 3 | ✓ |
| (G,H) | 3 | ✓ |
| (C,F) | 3 | ✓ |
| (B,C) | 4 | ✓ |

| <i>edge</i> | d_v | |
|-------------|-------|--|
| (B,E) | 4 | |
| (B,F) | 4 | |
| (B,H) | 4 | |
| (A,H) | 5 | |
| (D,F) | 6 | |
| (A,B) | 8 | |
| (A,F) | 10 | |



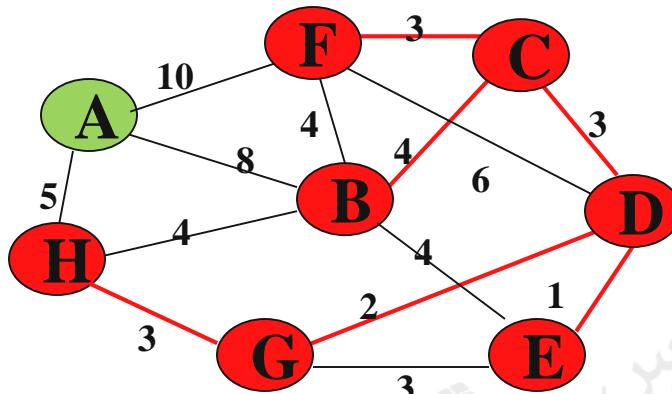
| <i>edge</i> | d_v | |
|-------------|-------|---|
| (D,E) | 1 | ✓ |
| (D,G) | 2 | ✓ |
| (E,G) | 3 | ✗ |
| (C,D) | 3 | ✓ |
| (G,H) | 3 | ✓ |
| (C,F) | 3 | ✓ |
| (B,C) | 4 | ✓ |

| <i>edge</i> | d_v | |
|-------------|-------|---|
| (B,E) | 4 | ✗ |
| (B,F) | 4 | |
| (B,H) | 4 | |
| (A,H) | 5 | |
| (D,F) | 6 | |
| (A,B) | 8 | |
| (A,F) | 10 | |



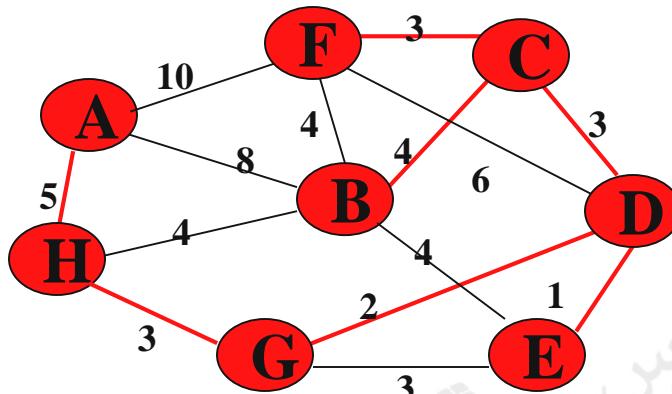
| <i>edge</i> | d_v | |
|-------------|-------|---|
| (D,E) | 1 | ✓ |
| (D,G) | 2 | ✓ |
| (E,G) | 3 | ✗ |
| (C,D) | 3 | ✓ |
| (G,H) | 3 | ✓ |
| (C,F) | 3 | ✓ |
| (B,C) | 4 | ✓ |

| <i>edge</i> | d_v | |
|-------------|-------|---|
| (B,E) | 4 | ✗ |
| (B,F) | 4 | ✗ |
| (B,H) | 4 | |
| (A,H) | 5 | |
| (D,F) | 6 | |
| (A,B) | 8 | |
| (A,F) | 10 | |



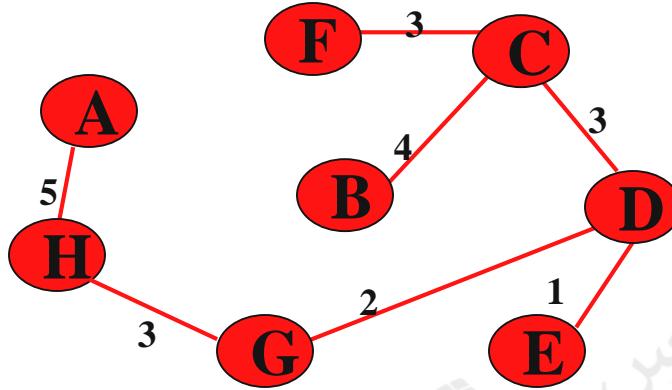
| <i>edge</i> | d_v | |
|-------------|-------|---|
| (D,E) | 1 | ✓ |
| (D,G) | 2 | ✓ |
| (E,G) | 3 | ✗ |
| (C,D) | 3 | ✓ |
| (G,H) | 3 | ✓ |
| (C,F) | 3 | ✓ |
| (B,C) | 4 | ✓ |

| <i>edge</i> | d_v | |
|-------------|-------|---|
| (B,E) | 4 | ✗ |
| (B,F) | 4 | ✗ |
| (B,H) | 4 | ✗ |
| (A,H) | 5 | |
| (D,F) | 6 | |
| (A,B) | 8 | |
| (A,F) | 10 | |



| <i>edge</i> | d_v | |
|-------------|-------|---|
| (D,E) | 1 | ✓ |
| (D,G) | 2 | ✓ |
| (E,G) | 3 | ✗ |
| (C,D) | 3 | ✓ |
| (G,H) | 3 | ✓ |
| (C,F) | 3 | ✓ |
| (B,C) | 4 | ✓ |

| <i>edge</i> | d_v | |
|-------------|-------|---|
| (B,E) | 4 | ✗ |
| (B,F) | 4 | ✗ |
| (B,H) | 4 | ✗ |
| (A,H) | 5 | ✓ |
| (D,F) | 6 | |
| (A,B) | 8 | |
| (A,F) | 10 | |



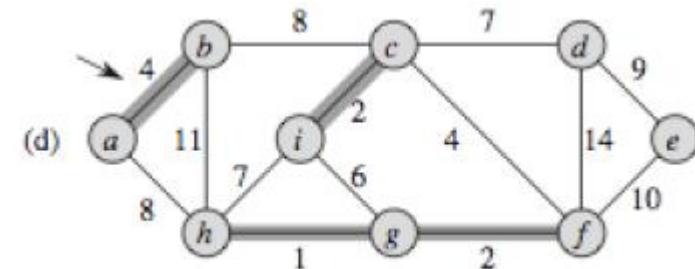
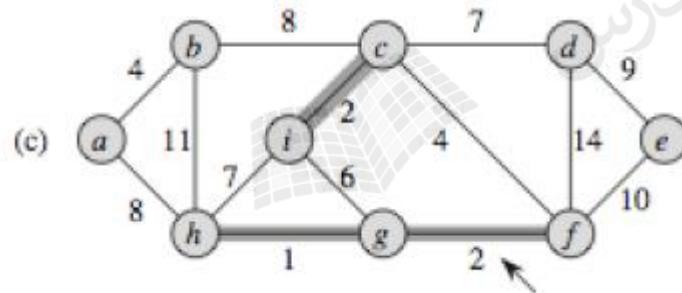
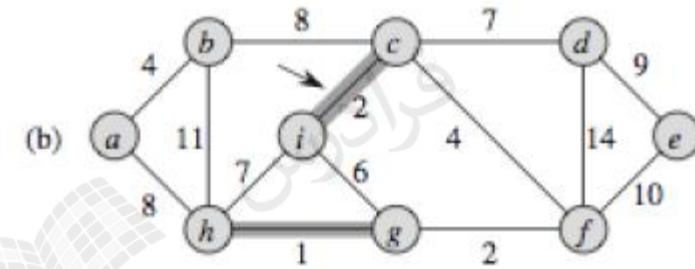
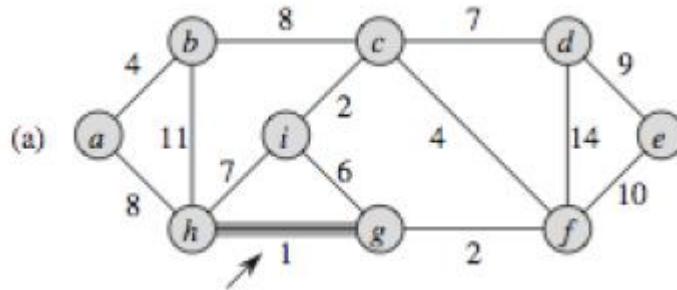
| <i>edge</i> | d_v | |
|-------------|-------|---|
| (D,E) | 1 | ✓ |
| (D,G) | 2 | ✓ |
| (E,G) | 3 | ✗ |
| (C,D) | 3 | ✓ |
| (G,H) | 3 | ✓ |
| (C,F) | 3 | ✓ |
| (B,C) | 4 | ✓ |

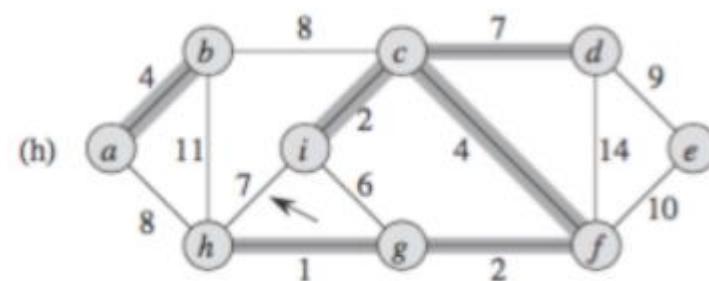
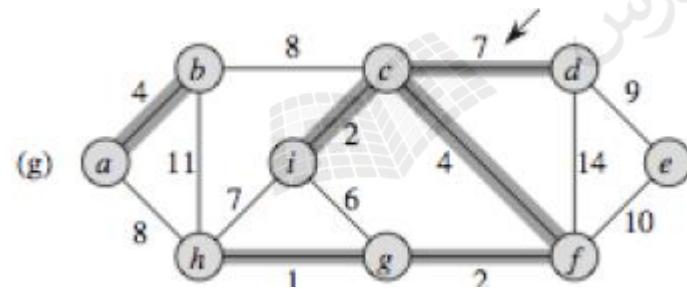
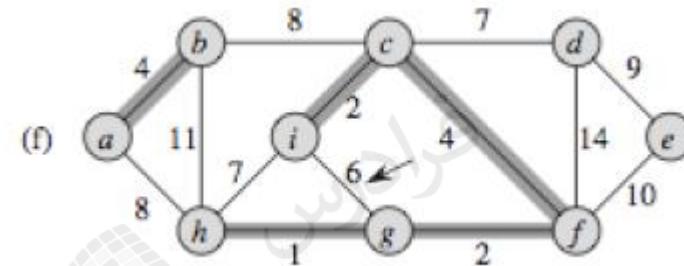
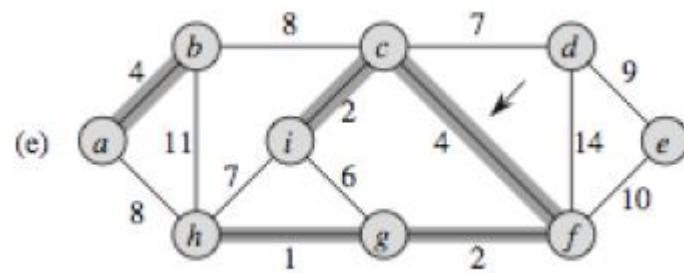
| <i>edge</i> | d_v | |
|-------------|-------|---|
| (B,E) | 4 | ✗ |
| (B,F) | 4 | ✗ |
| (B,H) | 4 | ✗ |
| (A,H) | 5 | ✓ |
| (D,F) | 6 | |
| (A,B) | 8 | |
| (A,F) | 10 | |

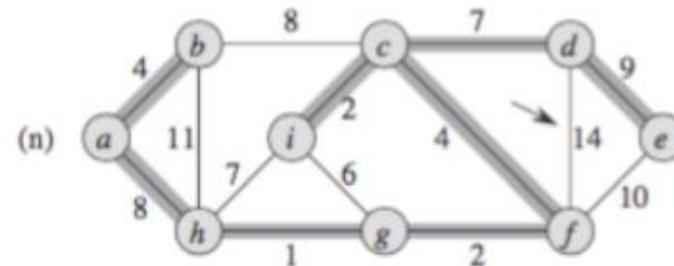
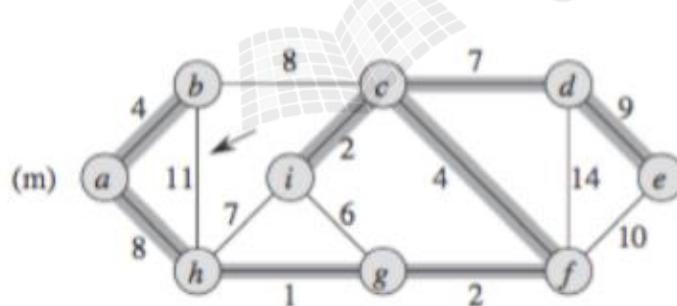
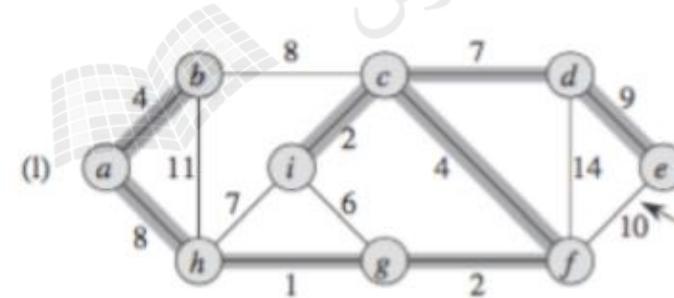
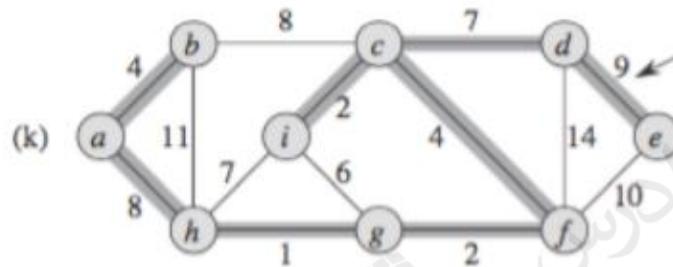
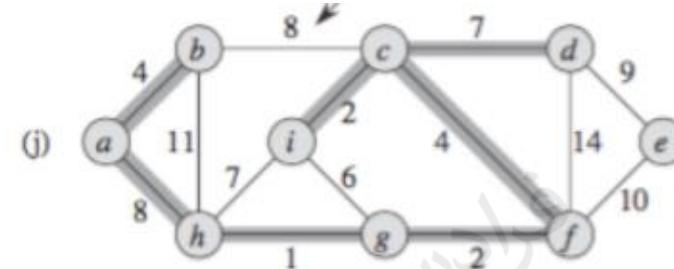
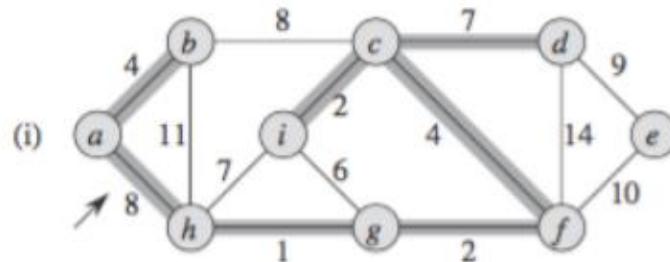
} not considered

Total Cost = 21

کروسکال







الگوریتم کروسکال

MST-Kruskal(G, w)

- 1 $A \leftarrow \emptyset$
- 2 for each vertex $v \in V[G]$
 - 3 do MAKE-SET(v)
 - 4 sort the edges of E into nondecreasing order by weight w
 - 5 for each edge $(u, v) \in E$, taken in nondecreasing order by weight
 - 6 do if FIND-SET(u) \neq FIND-SET(v)
 - 7 then $A \leftarrow A \cup \{(u, v)\}$
 - 8 UNION(u, v)
 - 9 return A

$O(E \lg E)$.

ساختمن داده مجموعه های جدا از هم

Disjoint set data structures

A disjoint-set data structure maintains a collection $\{S_1, \dots, S_k\}$ of disjoint sets.

Each set is identified by a “representative,” which is some member of the set.

- **MakeSet(x):** Create a new set containing x and only x . (x should not be in any other set, since the sets are disjoint.)
- **Union(x, y):** Create a new set that is the union of the sets containing x and y . Destroy the old sets.
- **FindSet(x):** Return a pointer to the representative of the set containing x .

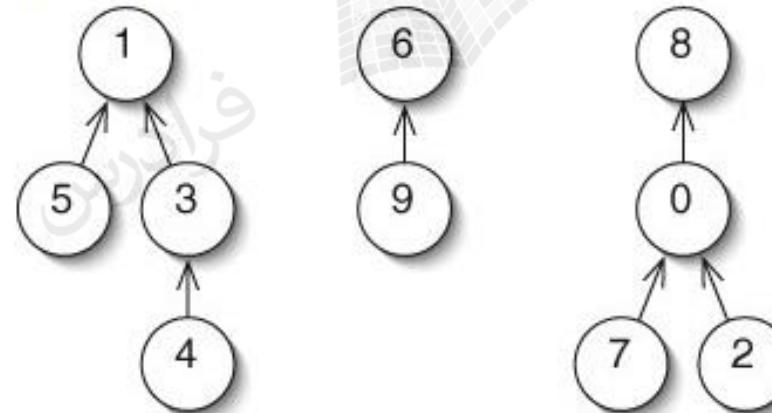
پیاده سازی با لیست پیوندی

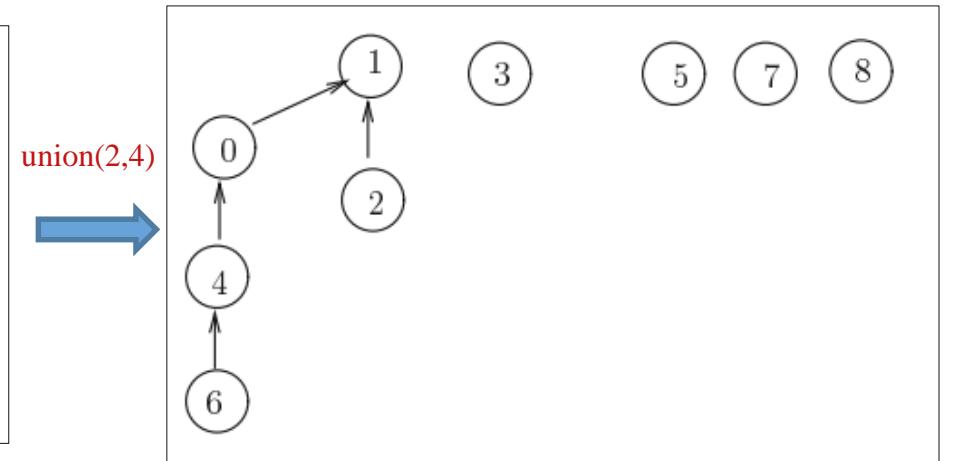
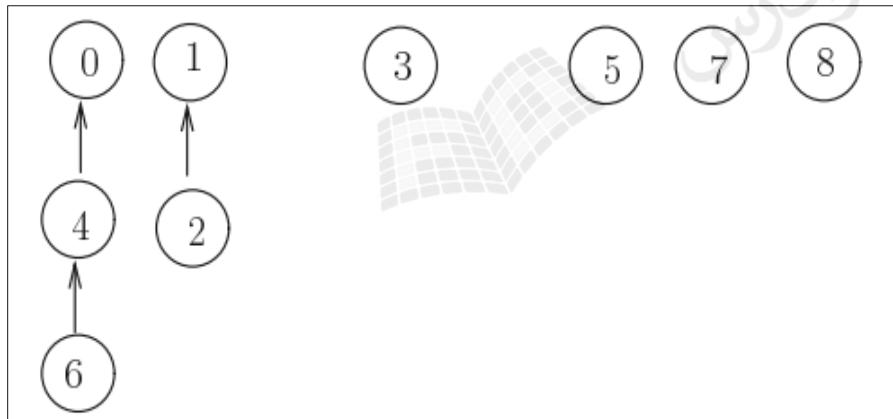
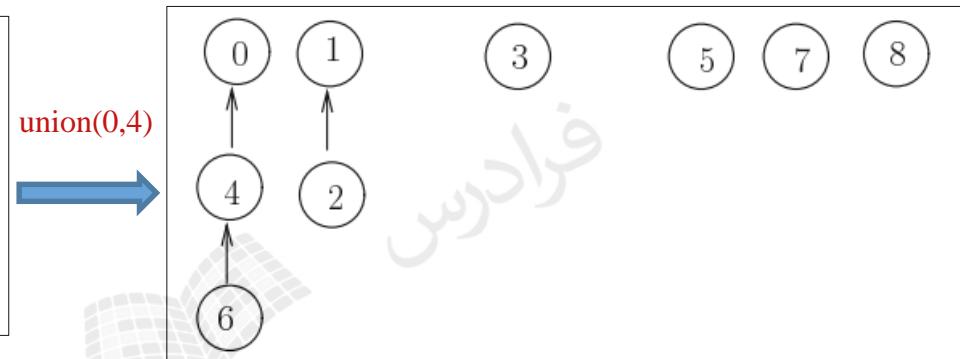
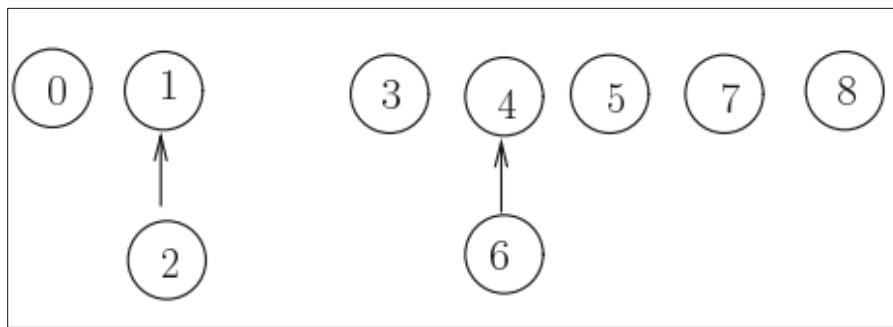
برای هر مجموعه، یک لیست پیوندی تشکیل دهیم و عنصر ابتدای هر لیست را به عنوان نماینده آن مجموعه انتخاب کینم.

هر مجموعه به وسیله یک درخت نشان داده می شود که هر گره در آن یک اشاره گر به پدرش دارد.

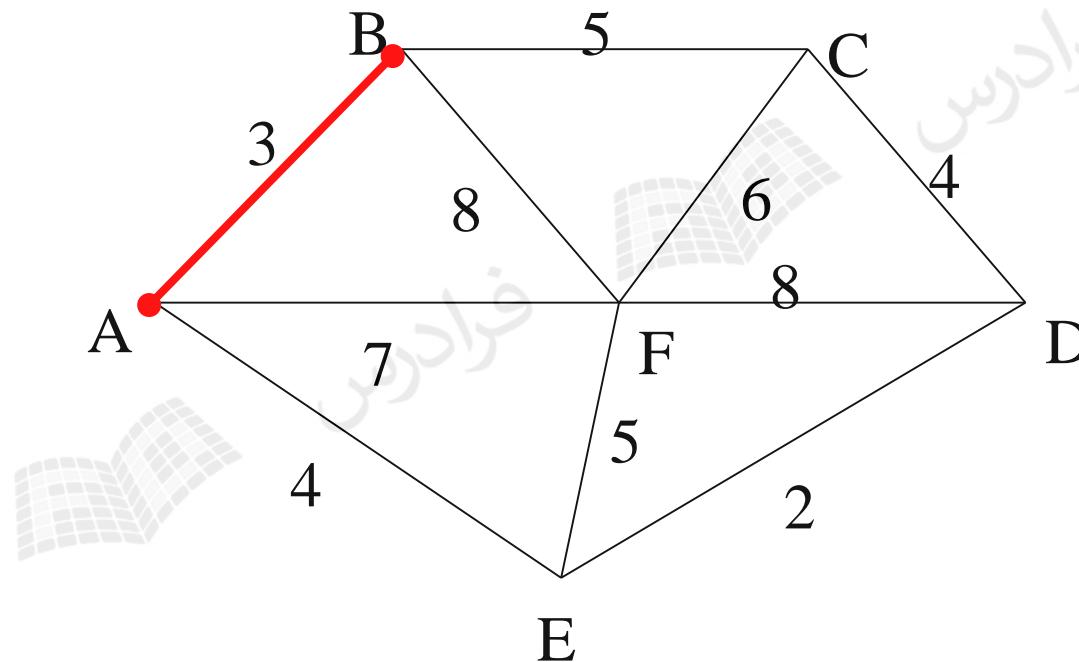
پیاده سازی به وسیله جنگل ها

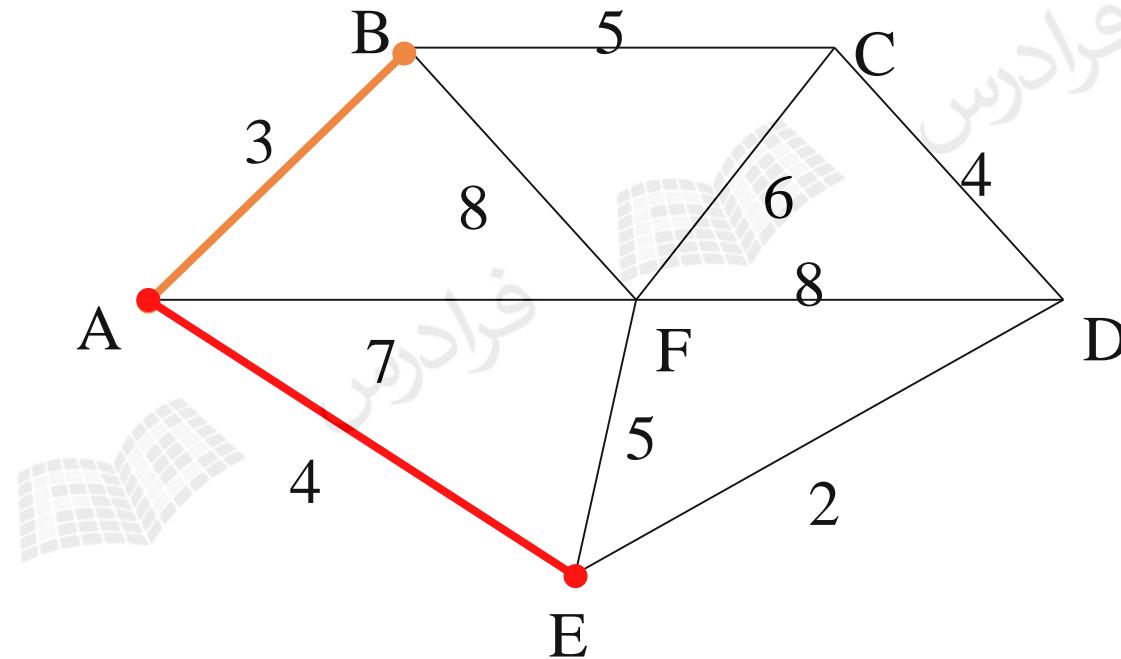
sets $\{1, 3, 4, 5\}$, $\{6, 9\}$, and $\{0, 2, 7, 8\}$.

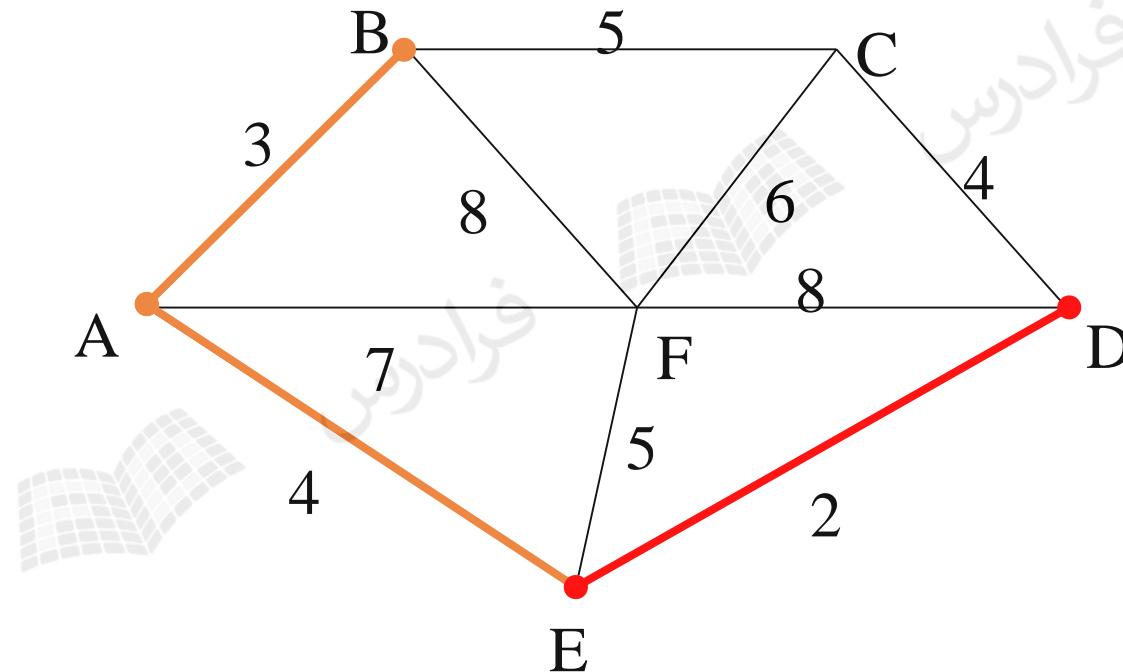


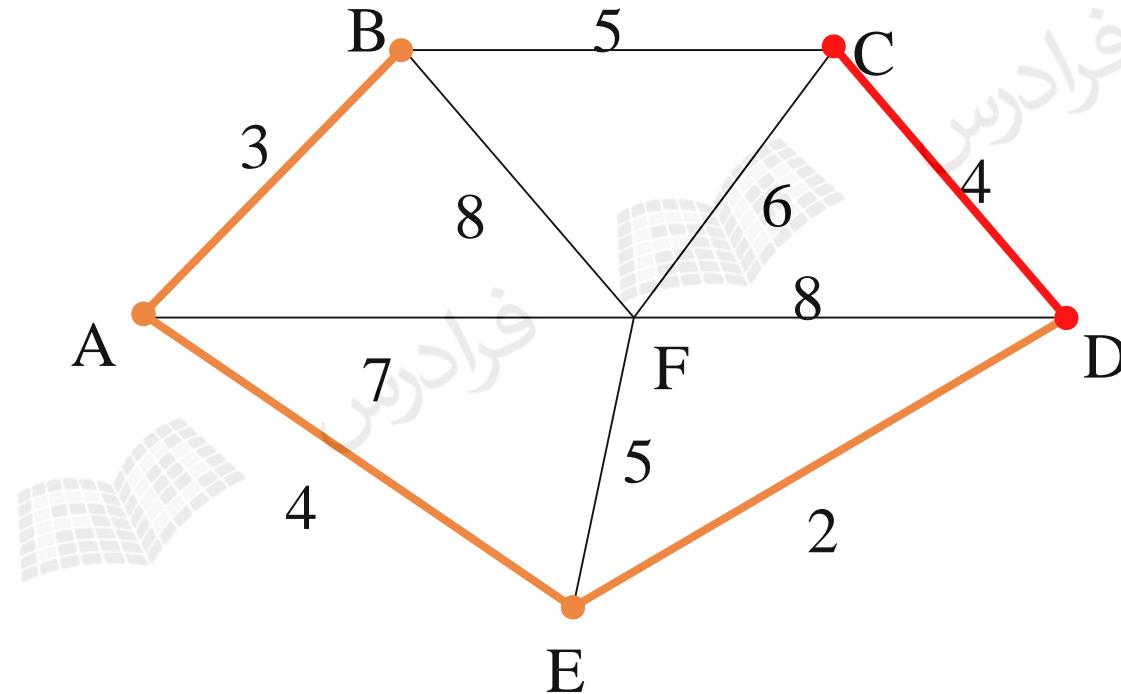


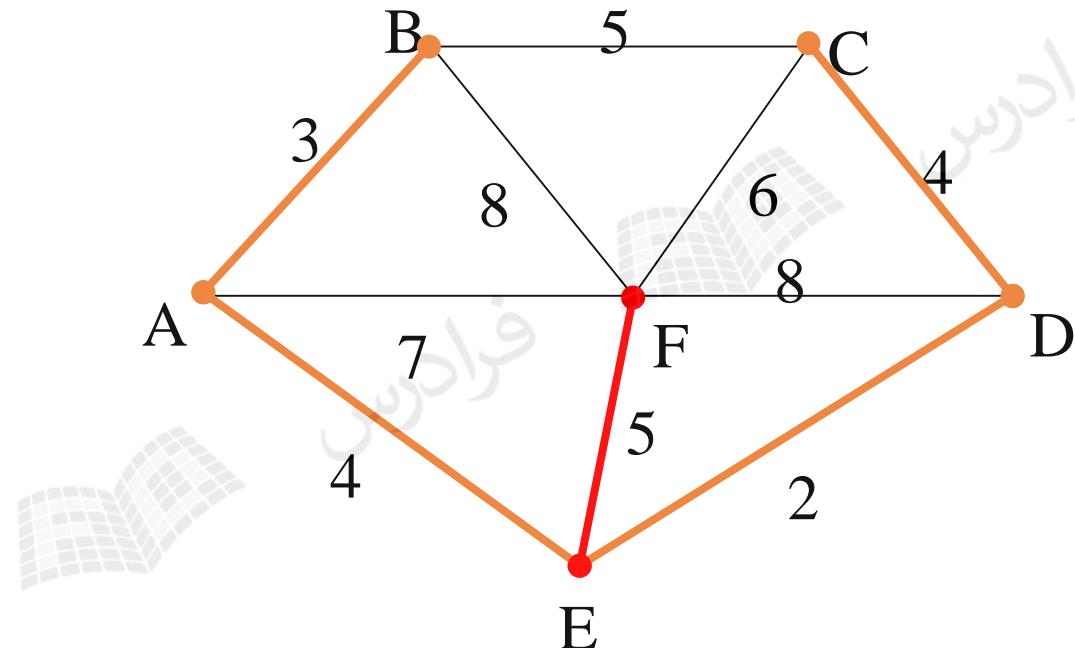
مثال (پریم)

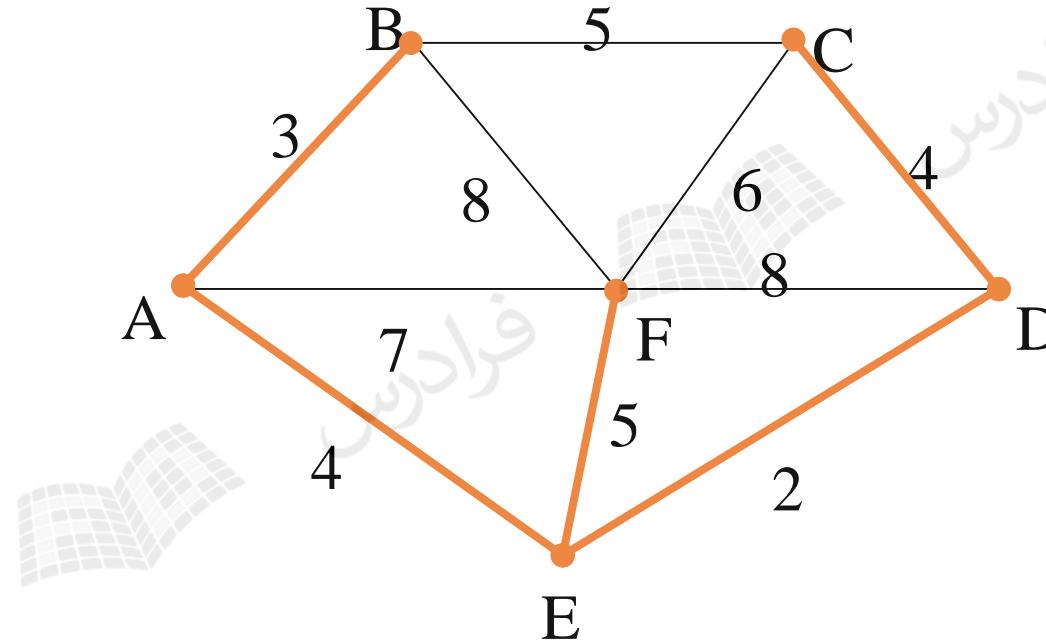






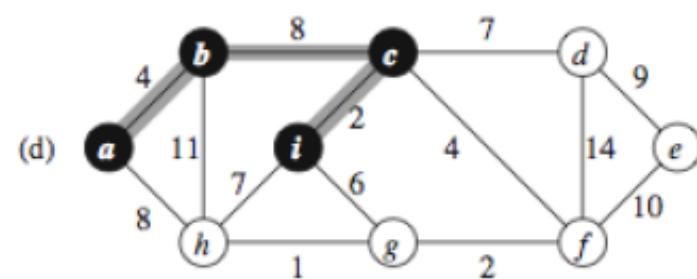
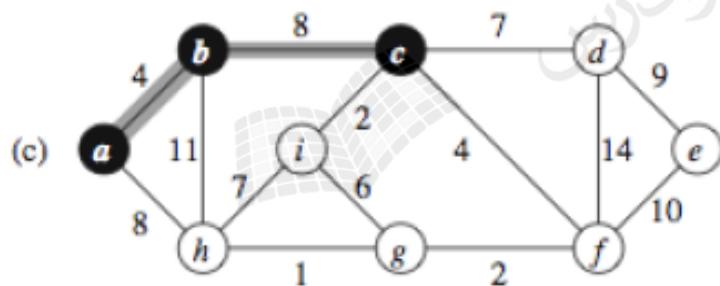
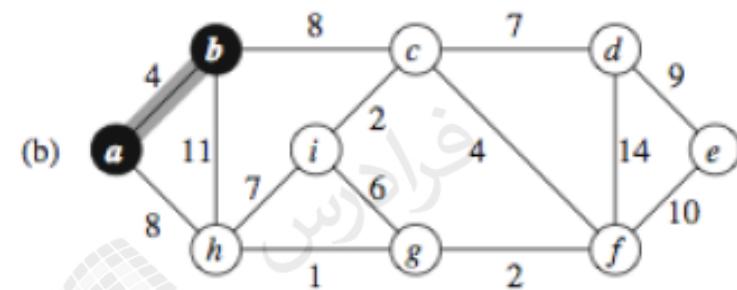
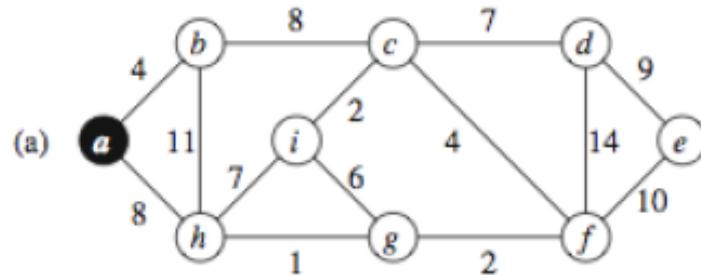


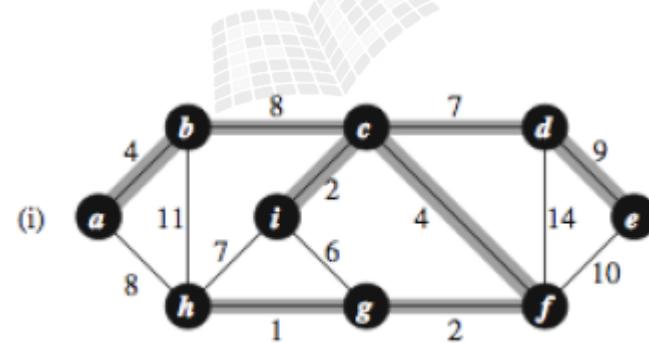
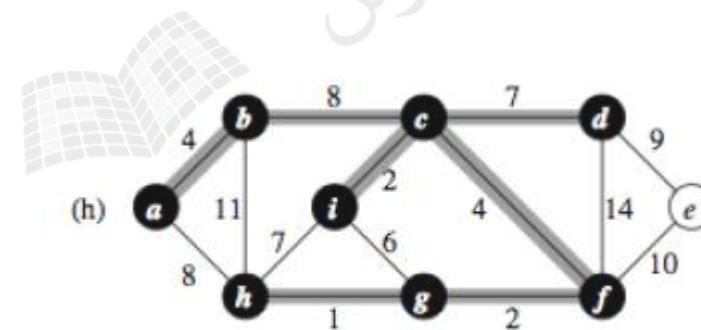
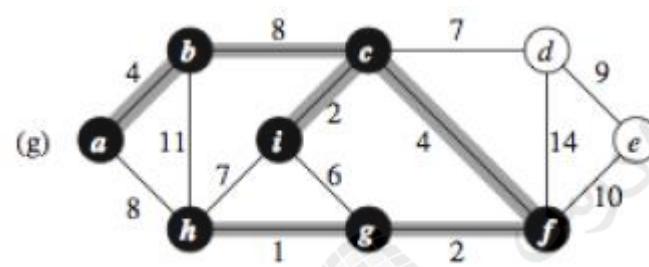
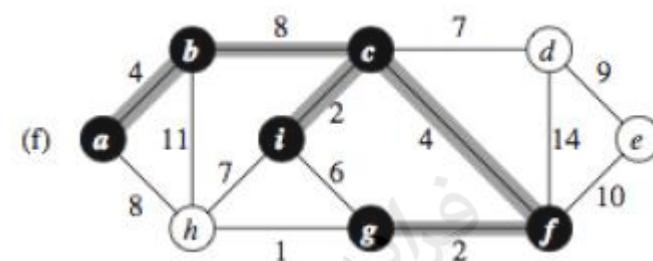
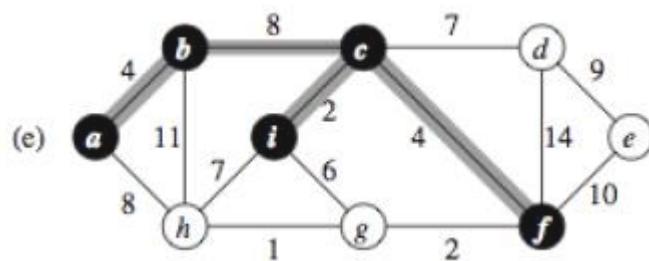




Total weight of tree: 18

پریم





پریم

MST-Prim(G, w, r)

```

1  for each  $u \in V[G]$ 
2      do  $key[u] \leftarrow \infty$ 
3           $\pi[u] \leftarrow \text{NIL}$ 
4   $key[r] \leftarrow 0$ 
5   $Q \leftarrow V[G]$ 
6  while  $Q \neq \emptyset$ 
7      do  $u \leftarrow \text{EXTRACT-MIN}(Q)$ 
8          for each  $v \in Adj[u]$ 
9              do if  $v \in Q$  and  $w(u, v) < key[v]$ 
10                 then  $\pi[v] \leftarrow u$ 
11                      $key[v] \leftarrow w(u, v)$ 

```

 $O(E \lg V)$



فروشنده دوره‌گرد

TSP : Traveling Salesman Problem

فروشنده دوره‌گرد

TSP : Traveling Salesman Problem

فروشنده‌ای می خواهد به یک سفر شامل چند شهر برود.

هر شهر توسط جاده‌ای به چند شهر دیگر متصل است.

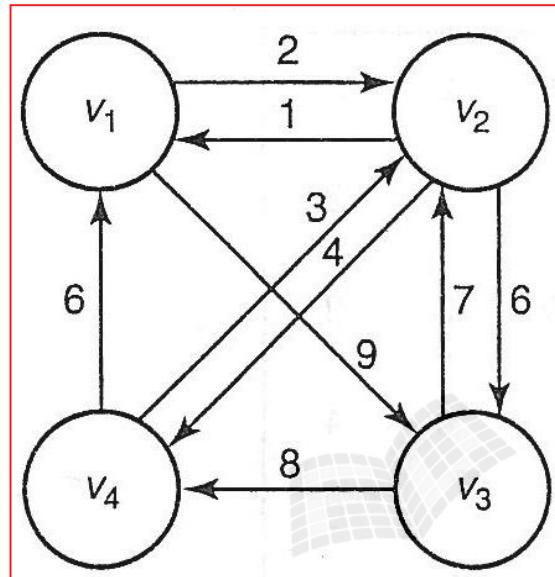
می خواهیم **کوتاه‌ترین** مسیر را با شروع از وطن فروشنده،

با یکبار عبور از همه شهرها و بازگشت به وطن تعیین کنیم.

مسئله فروشنده دوره گرد، یافتن یک **تور بهینه** در گرافی وزن دار و جهت دار است.

مثال

تعیین تور بهینه با شروع از $v1$.



سه تور
وجود دارد:

$$[v1, v2, v3, v4, v1] = 22$$

$$[v1, v3, v2, v4, v1] = 26$$

$$[v1, v3, v4, v2, v1] = 21$$

| | 1 | 2 | 3 | 4 |
|---|----------|---|----------|----------|
| 1 | 0 | 2 | 9 | ∞ |
| 2 | 1 | 0 | 6 | 4 |
| 3 | ∞ | 7 | 0 | 8 |
| 4 | 6 | 3 | ∞ | 0 |

ماتریس همچواری (W):

رابطه بازگشتی مسئله فروشنده دوره گرد

$$D[i][A] = \min_{j : v_j \in A} (W[i][j] + D[j][A - \{v_j\}])$$

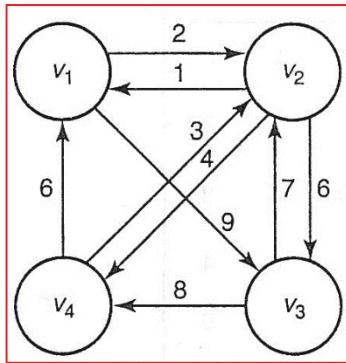
$$D[i][\phi] = W[i][1]$$

ماتریس D :

{φ} {v₂} {v₃} {v₄} {v₂,v₃} {v₂,v₄} {v₃,v₄} {v₂,v₃,v₄}

| 1 | | | | | | | جواب |
|---|--|--|--|--|--|--|------|
| 2 | | | | | | | |
| 3 | | | | | | | |
| 4 | | | | | | | |

تعیین ستون اول



$$D[i][\varphi] = W[i][1]$$

| | 1 | 2 | 3 | 4 |
|---|----------|---|----------|----------|
| 1 | 0 | 2 | 9 | ∞ |
| 2 | 1 | 0 | 6 | 4 |
| 3 | ∞ | 7 | 0 | 8 |
| 4 | 6 | 3 | ∞ | 0 |

$\{\varphi\}$ $\{v_2\}$ $\{v_3\}$ $\{v_4\}$ $\{v_2, v_3\}$ $\{v_2, v_4\}$ $\{v_3, v_4\}$ $\{v_2, v_3, v_4\}$

| | | | | | | | | |
|---|----------|--|--|--|--|--|--|--|
| 1 | | | | | | | | |
| 2 | 1 | | | | | | | |
| 3 | ∞ | | | | | | | |
| 4 | 6 | | | | | | | |

مجموعه های تک عنصری

$$D[i][A] = \min_{j: v_j \in A} (W[i][j] + D[j][A - \{v_j\}])$$

D[i][{v_j}] = W[i][j] + D[j][φ]

D[2][{v_3}] = W[2][3] + D[3][φ] = 6 + ∞ = ∞

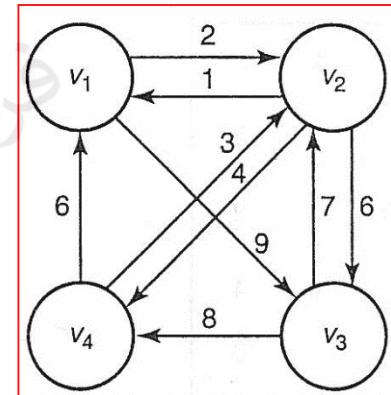
D[2][{v_4}] = W[2][4] + D[4][φ] = 4 + 6 = 10

D[3][{v_2}] = W[3][2] + D[2][φ] = 7 + 1 = 8

D[3][{v_4}] = W[3][4] + D[4][φ] = 8 + 6 = 14

D[4][{v_2}] = W[4][2] + D[2][φ] = 3 + 1 = 4

D[4][{v_3}] = W[4][3] + D[3][φ] = ∞ + ∞ = ∞



{φ} {v₂} {v₃} {v₄} {v₂,v₃} {v₂,v₄} {v₃,v₄} {v₂,v₃,v₄}

| | | | | | | | |
|---|---|---|---|----|--|--|--|
| 1 | | | | | | | |
| 2 | 1 | | ∞ | 10 | | | |
| 3 | ∞ | 8 | | 14 | | | |
| 4 | 6 | 4 | ∞ | | | | |

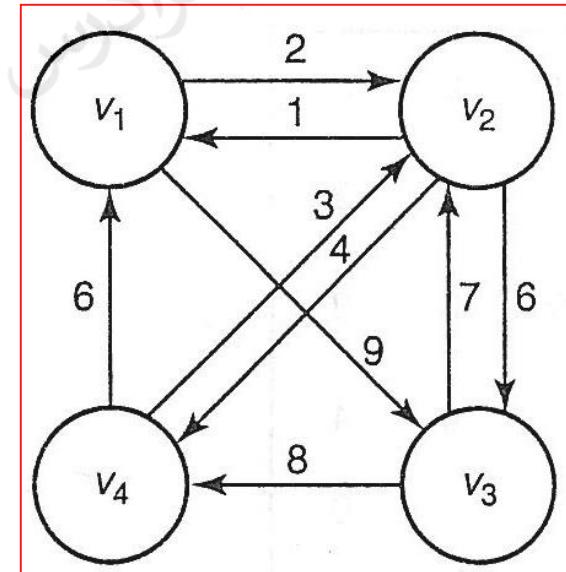
مجموعه های دو عنصری

$$D[i][A] = \min_{j: v_j \in A} (W[i][j] + D[j][A - \{v_j\}])$$

$$D[2][\{v_3, v_4\}] = \begin{cases} W[2][3] + D[3][\{v_4\}] = 6 + 14 = 20 \\ W[2][4] + D[4][\{v_3\}] = 4 + \infty = \infty \end{cases}$$

$$D[3][\{v_2, v_4\}] = \begin{cases} W[3][2] + D[2][\{v_4\}] = 7 + 10 = 17 \\ W[3][4] + D[4][\{v_2\}] = 8 + 4 = 12 \end{cases}$$

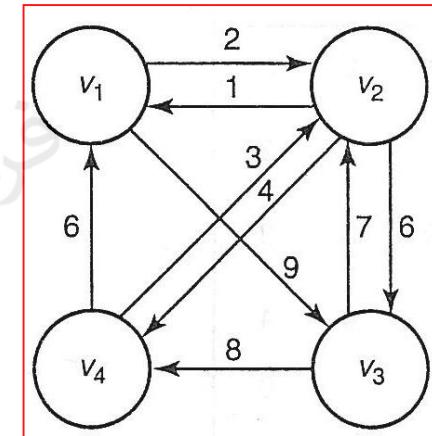
$$D[4][\{v_2, v_3\}] = \begin{cases} W[4][2] + D[2][\{v_3\}] = 3 + \infty = \infty \\ W[4][3] + D[3][\{v_2\}] = \infty + 8 = \infty \end{cases}$$



جواب نهایی

$$D[i][A] = \min_{j: v_j \in A} (W[i][j] + D[j][A - \{v_j\}])$$

$$D[1][\{v_2, v_3, v_4\}] = \begin{cases} W[1][2] + D[2][\{v_3, v_4\}] = 2 + 20 = 22 \\ W[1][3] + D[3][\{v_2, v_4\}] = 9 + 12 = 21 \\ W[1][4] + D[4][\{v_2, v_3\}] = \infty + \infty = \infty \end{cases}$$



{ \emptyset } { v_2 } { v_3 } { v_4 } { v_2, v_3 } { v_2, v_4 } { v_3, v_4 } { v_2, v_3, v_4 }

| | | | | | | | |
|---|----------|----------|----------|----|----------|----|----|
| 1 | | | | | | | 21 |
| 2 | 1 | ∞ | 10 | | | 20 | |
| 3 | ∞ | 8 | | 14 | | 12 | |
| 4 | 6 | 4 | ∞ | | ∞ | | |

ماتریس P

$\{\varnothing\}$ $\{v_2\}$ $\{v_3\}$ $\{v_4\}$ $\{v_2, v_3\}$ $\{v_2, v_4\}$ $\{v_3, v_4\}$ $\{v_2, v_3, v_4\}$

| | | | | | | | | |
|---|---|--|---|--|--|---|---|---|
| 1 | | | | | | | | 3 |
| 2 | | | 4 | | | | 3 | |
| 3 | 2 | | 4 | | | 4 | | |
| 4 | 2 | | | | | | | |

ابتدا به آخرین خانه پر شده نگاه می کنیم. چون مقدار آن ۳ است، پس ابتدا از v_1 به v_3 باید رفت. حال به $P[3][\{v_2, v_4\}]$ نگاه کنیم. چون برابر ۴ است، باید به v_4 رفت. در نهایت هم با توجه به اینکه $P[4][\{v_2\}]$ برابر ۲ است، باید به گره v_2 رفت.

بنابراین تور بهینه:

$[v_1, v_3, v_4, v_2, v_1]$

تحليل الگوریتم فروشنده دوره گرد

تعداد کل دفعاتی که عمل اصلی انجام می شود:

$$T(n) = \sum_{k=1}^{n-2} (n-1-k) \times k \times \binom{n-1}{k}$$

: تعداد عناصری که در هر ستون k عنصری باید محاسبه شود.

$$(n-1-k)$$

$$\binom{n-1}{k}$$

: تعداد ستون های k عنصری

: تعداد دفعات minimum گیری.

مثال

$$T(n) = \sum_{k=1}^{n-2} (n-1-k) \times k \times \binom{n-1}{k}$$

$$T(4) = (3-1) \times 1 \times \binom{3}{1} + (3-2) \times 2 \times \binom{3}{2} = 2 \times 1 \times 3 + 1 \times 2 \times 3 = 6 + 6 = 12$$

(3-1) : تعداد عناصری که در هر ستون تک عنصری باید پر شوند که برای هر یک از این عناصر باید 1 بار عمل اصلی (تعیین حداقل) انجام گیرد.

(3-2) : تعداد عناصری که در هر ستون دو عنصری باید پر شوند که برای هر یک از این عناصر باید 2 بار عمل اصلی (تعیین حداقل) انجام گیرد.

$\{\varphi\}$ $\{v_2\}$ $\{v_3\}$ $\{v_4\}$ $\{v_2, v_3\}$ $\{v_2, v_4\}$ $\{v_3, v_4\}$ $\{v_2, v_3, v_4\}$

| | | | | | | | | |
|---|---|----------|----------|----|----------|--|----|----|
| 1 | | | | | | | | 21 |
| 2 | 1 | ∞ | 10 | | | | 20 | |
| 3 | | 8 | | 14 | | | 12 | |
| 4 | 6 | 4 | ∞ | | ∞ | | | |

مرتبه

$$T(n) = \sum_{k=1}^{n-2} (n-1-k) \times k \times \binom{n-1}{k}$$

$$T(n) = (n-1)(n-2)2^{(n-3)} \in \Theta(n^2 2^n)$$

مثال

در مسئله فروشنده دوره گرد، با فرض این که پردازش دستور اصلی در الگوریتم، یک میکرو ثانیه زمان ببرد، برای $n=20$ به چند ثانیه زمان نیاز دارد؟

$$T(n) = (n - 1)(n - 2)2^{(n-3)}$$

$$T(20) = (20 - 1)(20 - 2)2^{(20-3)} \mu\text{s} = 45\text{s}$$

استفاده از این الگوریتم وقتی کارایی دارد که n کوچک باشد.

پیچیدگی حافظه ای

$$M(n) = 2 \times n2^{n-1} = n2^n \in \Theta(n2^n)$$

{φ} {v₂} {v₃} {v₄} {v₂,v₃} {v₂,v₄} {v₃,v₄} {v₂,v₃,v₄}

| | | | | | | | |
|---|--|--|--|--|--|--|--|
| 1 | | | | | | | |
| 2 | | | | | | | |
| 3 | | | | | | | |
| 4 | | | | | | | |

این اسلاید ها بر مبنای نکات مطرح شده در فرادرس
«آموزش طراحی الگوریتم»
تهیه شده است.

برای کسب اطلاعات بیشتر در مورد این آموزش به لینک زیر مراجعه نمایید
faradars.org/fvsft1092