BFS / DFS
Topological Sort

طراحی الگوریتم‌ها – جلسه هفدهم و هجدهم

Introduction to Algorithm

استاد: جوانمردی

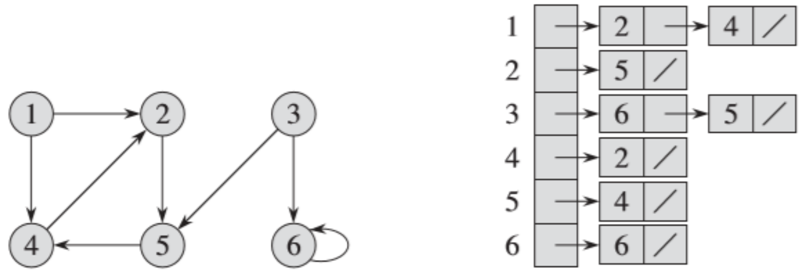
فصل ۲۲ کتاب – الگوریتم‌های پایه گراف

- ۲۲.۱: نمایش گراف
- ۲۲.۲: جستجوی اول سطح
- ۲۲.۳: جستجوی اول عمق
- ۲۲.۴: مرتب‌سازی توپولوژیکی

VI Graph Algorithms

گراف و نحوه نمایش آن

- یک گراف را معمولا به صورت دو مجموعه رئوس و یال‌ها نمایش می‌دهیم $G(V,E)$
- گراف جهت دار: گرافی است که یال ها جهت دارند و یال بصورت یک زوج مرتب بیان میشود



(a) (b)

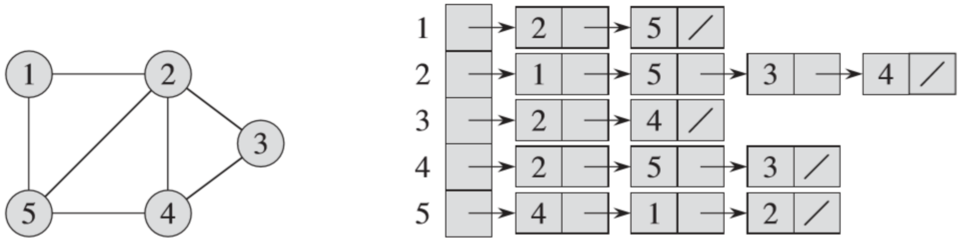
لیست همسایگی

	1	2	3	4	5	6
1	0	1	0	1	0	0
2	0	0	0	0	1	0
3	0	0	0	0	1	1
4	0	1	0	0	0	0
5	0	0	0	1	0	0
6	0	0	0	0	0	1

(c)

ماتریس همسایگی

- گراف بدون جهت: در صورت وجود یک رابطه بین دو گره حتما بر عکس آن نیز وجود دارد



(a) (b)

	1	2	3	4	5
1	0	1	0	0	1
2	1	0	1	1	1
3	0	1	0	1	0
4	0	1	1	0	1
5	1	1	0	1	0

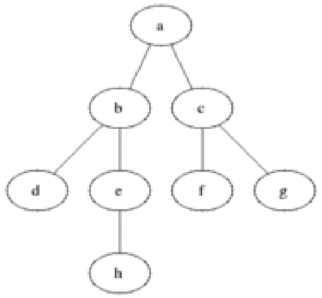
(c)

الگوریتم جستجوی اول سطح (BFS)

- الگوریتم‌های جستجوی گراف بصورت کلی:
  - از گره $source$ شروع و با پیمایش سیستماتیک یال‌های گراف کشف تمام گره های قابل دسترسی از گره منبع
- الگوریتم جستجوی اول سطح یا Breadth-first search
  - ورودی: یک گراف $G = (V,E)$ و گره $source$
  - خروجی: ۱. محاسبه فاصله بین گره‌های قابل دسترسی از گره s تا گره s (کمترین تعداد یال‌ها)
    ۲. درخت اول سطح (breadth-first tree) با ریشه s که شامل تمام گره‌های قابل دسترسی از s هستند

مسیر ساده از s به گره دلخواه v در درخت اول سطح در واقع کوتاه‌ترین مسیر از گره s به v در گراف G است

- علت نامگذاری:
  - در BFS همه گره‌های با فاصله K از گره $source$ حتما قبل از گره‌هایی با فاصله $K + 1$ مشاهده می‌شوند

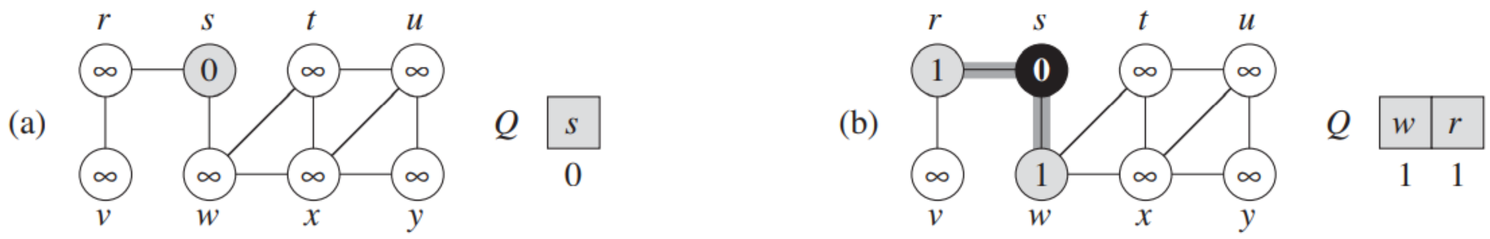


http://en.wikipedia.org/wiki/Image:Animated_BFS.gif

الگوریتم جستجوی اول سطح – نحوه عملکرد

- در روند اجرای BFS گره ها به سه رنگ سفید – طوسی – مشکی رنگ می‌شوند
- ابتدای کار رنگ تمام گره‌ها سفید می باشند و در ادامه شاید طوسی و سپس مشکی شوند
- هر گره وقتی برای اولین بار کشف می‌شود به رنگ غیر سفید در می‌آید
- به همین دلیل گره‌های طوسی و مشکی گره‌هایی هستند که کشف شده اند
- دلیل رنگ‌گذاری متفاوت طوسی و مشکی دستیابی به هدف سطح اول در روند جستجو می‌باشد

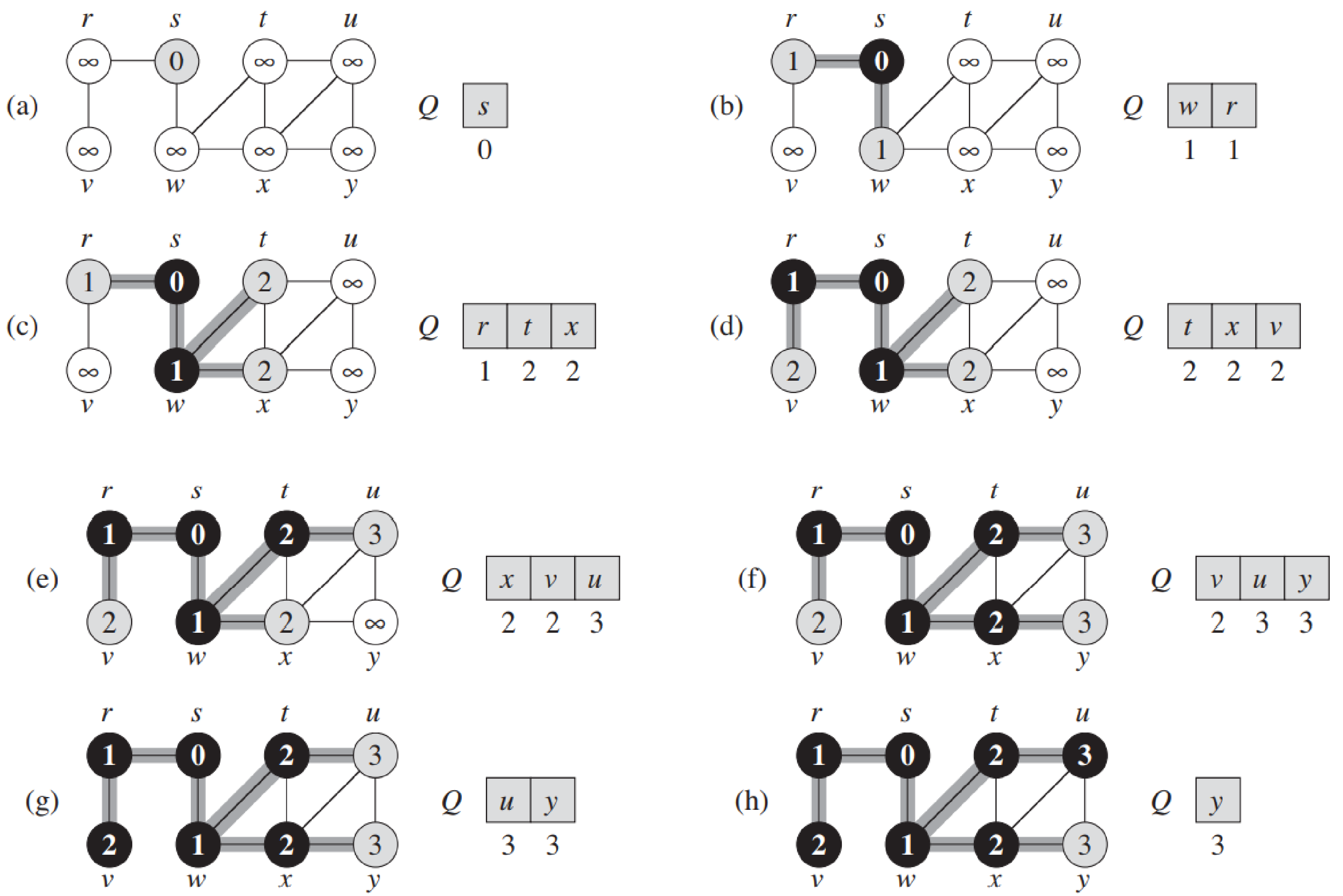
- استفاده از ساختمان داده صف (queue) در روند جستجو



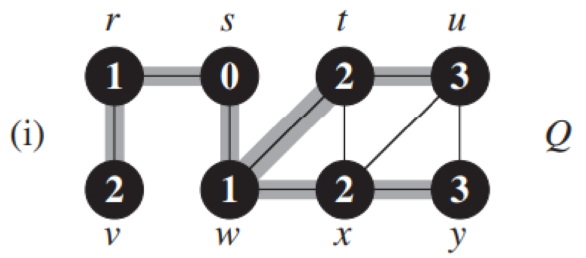
الگوریتم جستجوی اول سطح – شبه کد

```
BFS(G, s)
for each vertex u ∈ G.V − {s}
    u.color = WHITE
    u.d = ∞
    u.π = NIL
s.color = GRAY
s.d = 0
s.π = NIL
Q = ∅
ENQUEUE(Q, s)
while Q ≠ ∅
    u = DEQUEUE(Q)
    for each v ∈ G.Adj[u]
        if v.color == WHITE
            v.color = GRAY
            v.d = u.d + 1
            v.π = u
            ENQUEUE(Q, v)
    u.color = BLACK
```

الگوریتم جستجوی اول سطح – مثال




```
BFS(G, s)
    for each vertex u ∈ G.V − {s}
        u.color = WHITE
        u.d = ∞
        u.π = NIL
    s.color = GRAY
    s.d = 0
    s.π = NIL
    Q = ∅
    ENQUEUE(Q, s)
    while Q ≠ ∅
        u = DEQUEUE(Q)
        for each v ∈ G.Adj[u]
            if v.color == WHITE
                v.color = GRAY
                v.d = u.d + 1
                v.π = u
                ENQUEUE(Q, v)
        u.color = BLACK
```



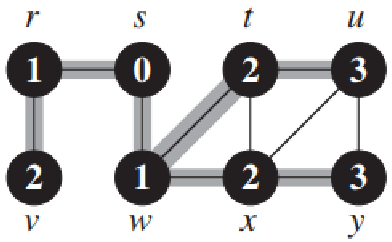
تحلیل زمانی BFS

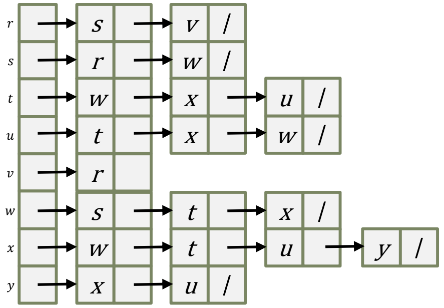
$$G = (V, E)$$

استفاده از روش تجمیعی برای محاسبه مرتبه زمانی الگوریتم

```
BFS(G, s)
for each vertex u ∈ G.V − {s}
    u.color = WHITE
    u.d = ∞
    u.π = NIL
s.color = GRAY
s.d = 0
s.π = NIL
Q = ∅
ENQUEUE(Q, s)
while Q ≠ ∅
    u = DEQUEUE(Q)
    for each v ∈ G.Adj[u]
        if v.color == WHITE
            v.color = GRAY
            v.d = u.d + 1
            v.π = u
            ENQUEUE(Q, v)
    u.color = BLACK
```

- Initialization overhead $O(V)$
- Enqueue and Dequeue happen only once for each node. – $O(V)$.
- Sum of the lengths of adjacency lists – $\theta(E)$ (for a directed graph)





Total runtime $O(V + E)$

BFS و کوتاه‌ترین مسیر

- فاصله کوتاه ترین مسیر از S به V را با $\delta(s, v)$ نشان می‌دهیم
- اثبات محاسبه فاصله کوتاه‌ترین مسیر در BFS در $v.d$: $\boxed{v.d = \delta(s, v)}$

Lemma 22.1
Let $G = (V, E)$ be a directed or undirected graph, and let $s \in V$ be an arbitrary vertex. Then, for any edge $(u, v) \in E$,

$$\delta(s, v) \leq \delta(s, u) + 1 .$$

Lemma 22.2
Let $G = (V, E)$ be a directed or undirected graph, and suppose that BFS is run on G from a given source vertex $s \in V$. Then upon termination, for each vertex $v \in V$, the value $v.d$ computed by BFS satisfies $v.d \geq \delta(s, v)$.

Lemma 22.3
Suppose that during the execution of BFS on a graph $G = (V, E)$, the queue Q contains the vertices $\langle v_1, v_2, \ldots, v_r \rangle$, where v_1 is the head of Q and v_r is the tail. Then, $v_r.d \leq v_1.d + 1$ and $v_i.d \leq v_{i+1}.d$ for $i = 1, 2, \ldots, r - 1$.

Corollary 22.4
Suppose that vertices v_i and v_j are enqueued during the execution of BFS, and that v_i is enqueued before v_j. Then $v_i.d \leq v_j.d$ at the time that v_j is enqueued.

Theorem 22.5 (Correctness of breadth-first search)
Let $G = (V, E)$ be a directed or undirected graph, and suppose that BFS is run on G from a given source vertex $s \in V$. Then, during its execution, BFS discovers every vertex $v \in V$ that is reachable from the source s, and upon termination, $v.d = \delta(s, v)$ for all $v \in V$. Moreover, for any vertex $v \neq s$ that is reachable from s, one of the shortest paths from s to v is a shortest path from s to $v.\pi$ followed by the edge $(v.\pi, v)$.

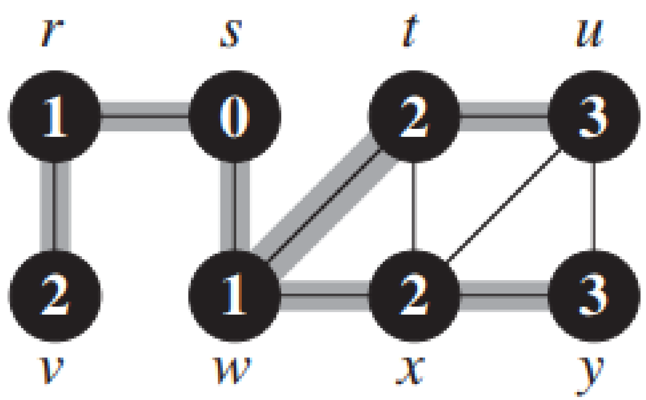
Breadth-first tree درخت اول سطح

$G = (V, E)$ with source s.

$G_\pi = (V_\pi, E_\pi)$

$V_\pi = \{v \in V : v.\pi \neq \text{NIL}\} \cup \{s\}$

$E_\pi = \{(v.\pi, v) : v \in V_\pi - \{s\}\}$

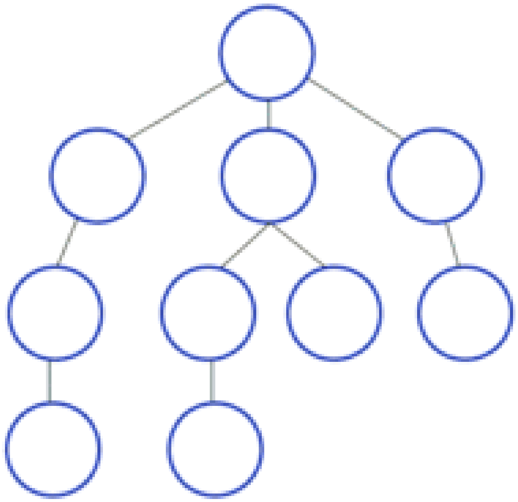


$$|E_\pi| = |V_\pi| - 1$$

```
PRINT-PATH(G, s, v)
if v == s
    print s
elseif v.π == NIL
    print "no path from" s "to" v "exists"
else PRINT-PATH(G, s, v.π)
    print v
```

الگوریتم جستجوی اول عمق (DFS)

- الگوریتم جستجوی اول عمق یا Depth-first search
  - مفهوم: جستجوی عمیق‌تر در گراف تا زمانی که امکان دارد (دقیقا عکس جستجوی اول سطح)
  - ورودی: یک گراف $G = (V,E)$ (نیازی به گره $source$ نداریم)
  - خروجی: ۱. زمان‌های کشف $(v.d)$ و پایان $(v.f)$ برای تمام گره‌ها
    ۲. جنگل اول عمق (depth-first forest)



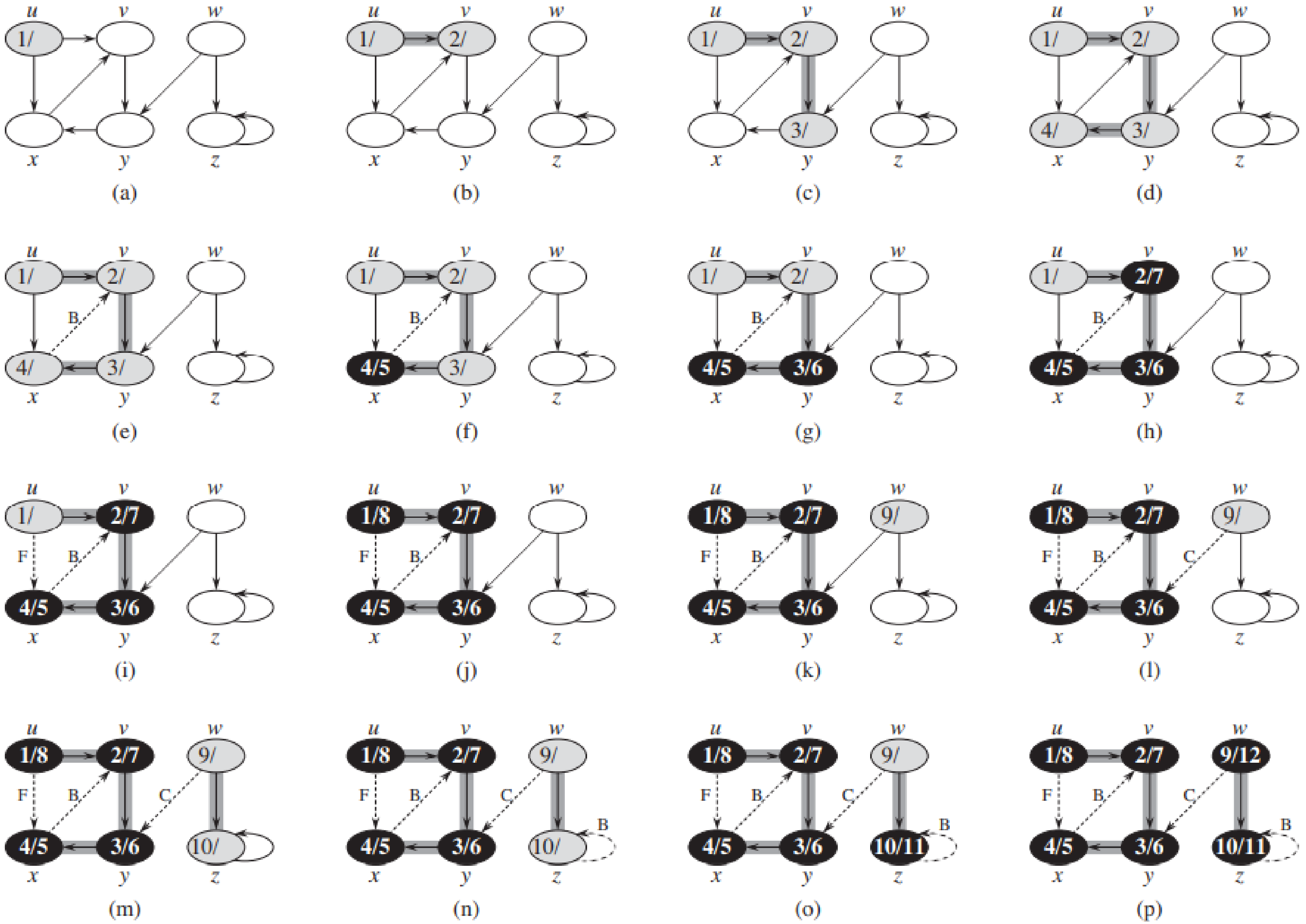
https://en.wikipedia.org/wiki/Depth-first_search

الگوریتم جستجوی اول عمق – شبه کد

```
DFS(G)
for each vertex u ∈ G.V
    u.color = WHITE
    u.π = NIL
time = 0
for each vertex u ∈ G.V
    if u.color == WHITE
        DFS-VISIT(G, u)
```

```
DFS-VISIT(G, u)
time = time + 1                  // white vertex u has just been discovered
u.d = time
u.color = GRAY
for each v ∈ G.Adj[u]            // explore edge (u, v)
    if v.color == WHITE
        v.π = u
        DFS-VISIT(G, v)
u.color = BLACK                  // blacken u; it is finished
time = time + 1
u.f = time
```

الگوریتم جستجوی اول عمق – مثال



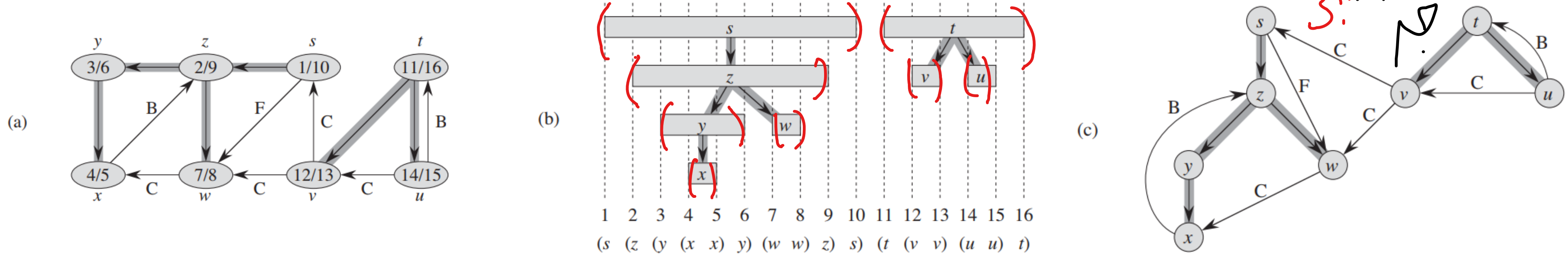

```
DFS(G)
for each vertex u ∈ G.V
    u.color = WHITE
    u.π = NIL
time = 0
for each vertex u ∈ G.V
    if u.color == WHITE
        DFS-VISIT(G, u)
```

```
DFS-VISIT(G, u)
time = time + 1
u.d = time
u.color = GRAY
for each v ∈ G.Adj[u]
    if v.color == WHITE
        v.π = u
        DFS-VISIT(G, v)
u.color = BLACK
time = time + 1
u.f = time
```

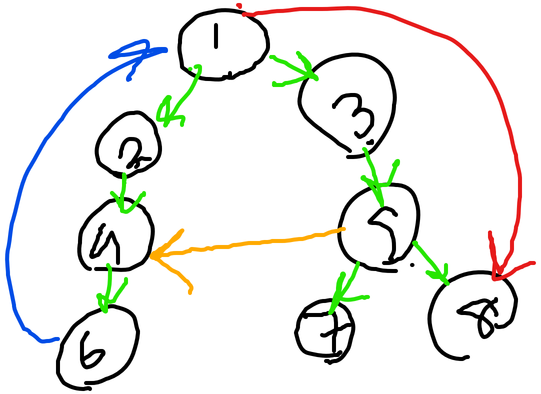
ویژگی‌های DFS

In any depth-first search of a (directed or undirected) graph $G = (V, E)$, for any two vertices u and v, exactly one of the following three conditions holds:

- the intervals $[u.d, u.f]$ and $[v.d, v.f]$ are entirely disjoint, and neither u nor v is a descendant of the other in the depth-first forest,
- the interval $[u.d, u.f]$ is contained entirely within the interval $[v.d, v.f]$, and u is a descendant of v in a depth-first tree, or
- the interval $[v.d, v.f]$ is contained entirely within the interval $[u.d, u.f]$, and v is a descendant of u in a depth-first tree.



دانشگاه صنعتی امیرکبیر
(پلی تکنیک تهران)



انواع یال‌ها در DFS

tree forward

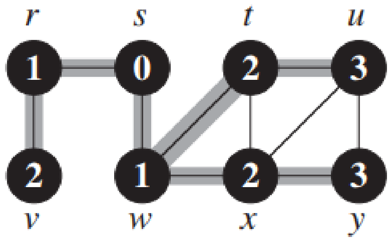
cross back

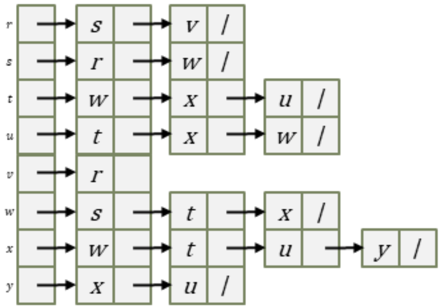
1. ***Tree edges*** are edges in the depth-first forest G_π. Edge (u, v) is a tree edge if v was first discovered by exploring edge (u, v).
2. ***Back edges*** are those edges (u, v) connecting a vertex u to an ancestor v in a depth-first tree. We consider self-loops, which may occur in directed graphs, to be back edges.
3. ***Forward edges*** are those nontree edges (u, v) connecting a vertex u to a descendant v in a depth-first tree.
4. ***Cross edges*** are all other edges. They can go between vertices in the same depth-first tree, as long as one vertex is not an ancestor of the other, or they can go between vertices in different depth-first trees.

تحلیل زمانی DFS

$G = (V, E)$

استفاده از روش تجمیعی برای محاسبه مرتبه زمانی الگوریتم





DFS(G)
```
for each vertex u ∈ G.V
    u.color = WHITE
    u.π = NIL
time = 0
for each vertex u ∈ G.V
    if u.color == WHITE
        DFS-VISIT(G, u)
```

DFS-VISIT(G, u)
```
time = time + 1
u.d = time
u.color = GRAY
for each v ∈ G.Adj[u]
    if v.color == WHITE
        v.π = u
        DFS-VISIT(G, v)
u.color = BLACK
time = time + 1
u.f = time
```

- مقدار دهی اولیه: $O(V)$
- تعداد فراخوان‌های DFS-VISIT: $O(V)$
- مجموع لیست همسایگی‌ها: $\Theta(E)$

BFS and DFS – possible applications

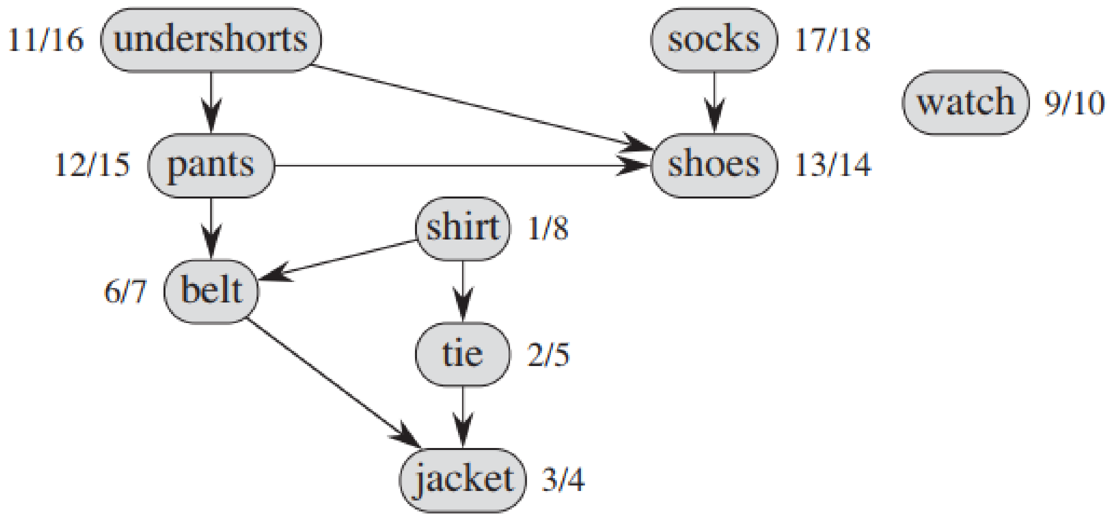
- Exploration algorithms in Artificial Intelligence
- Possible to use in routing / exploration wherever travel is involved. E.g.,
  - I want to explore all the nearest pizza places and want to go to the nearest one with only two intersections.
  - Find distance from my factory to every delivery center.
  - Most of the mapping software (GOOGLE maps, YAHOO(?) maps) should be using these algorithms.
  - Companies like Waste Management, UPS and FedEx?
- Applications of DFS
  - Topologically sorting a directed acyclic graph.
    - List the graph elements in such an order that all the nodes are listed before nodes to which they have outgoing edges.
  - Finding the strongly connected components of a directed graph.
    - List all the subgraphs of a strongly connected graph which themselves are strongly connected.

مرتب‌سازی توپولوژیکی یا Topological Sort

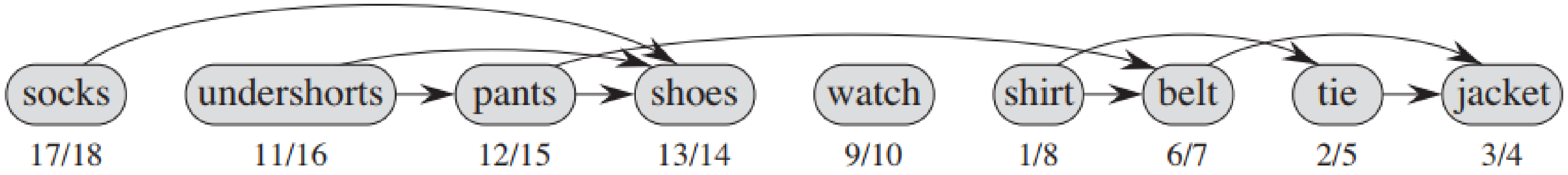
یک ترتیب خطی از همه رئوس گراف به‌طوری‌که هر گره قبل از همه گره‌هایی می‌آید که از آن به آن‌ها یال خارج شده‌است

directed acyclic graph, or a “dag”

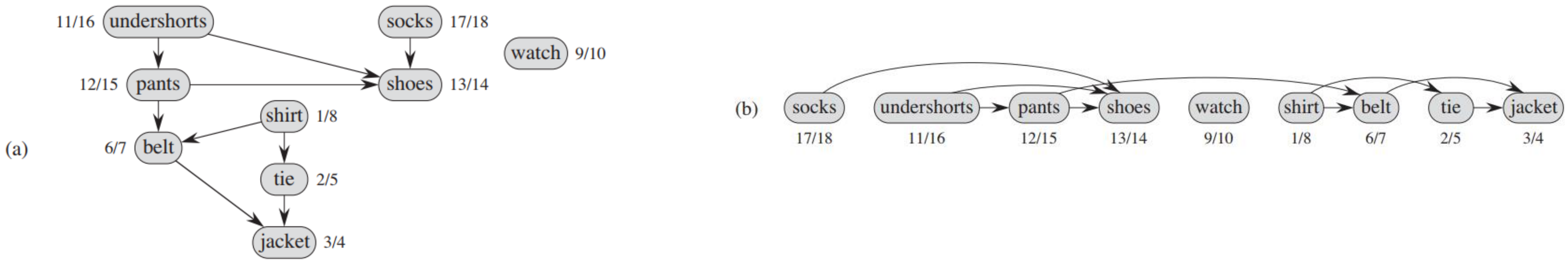
(a)



(b)



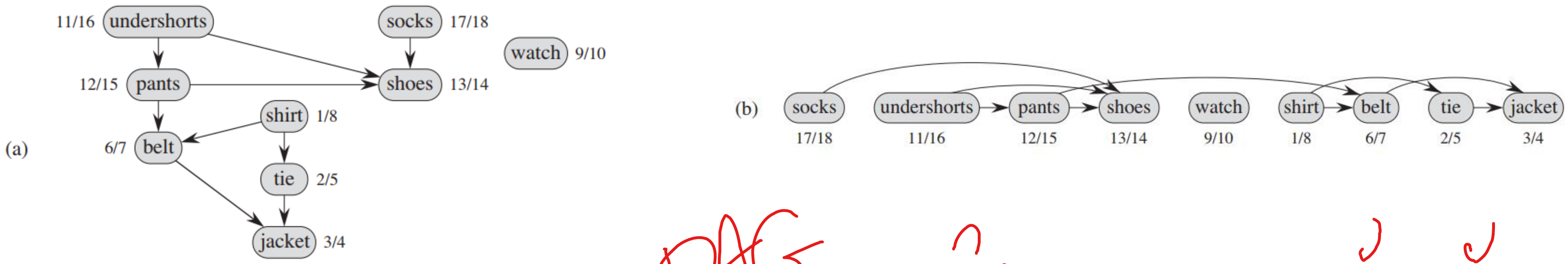
مرتب‌سازی توپولوژیکی یا Topological Sort



TOPOLOGICAL-SORT(G)

```
call DFS(G) to compute finishing times v.f for each vertex v
as each vertex is finished, insert it onto the front of a linked list
return the linked list of vertices
```

مرتب‌سازی توپولوژیکی یا Topological Sort

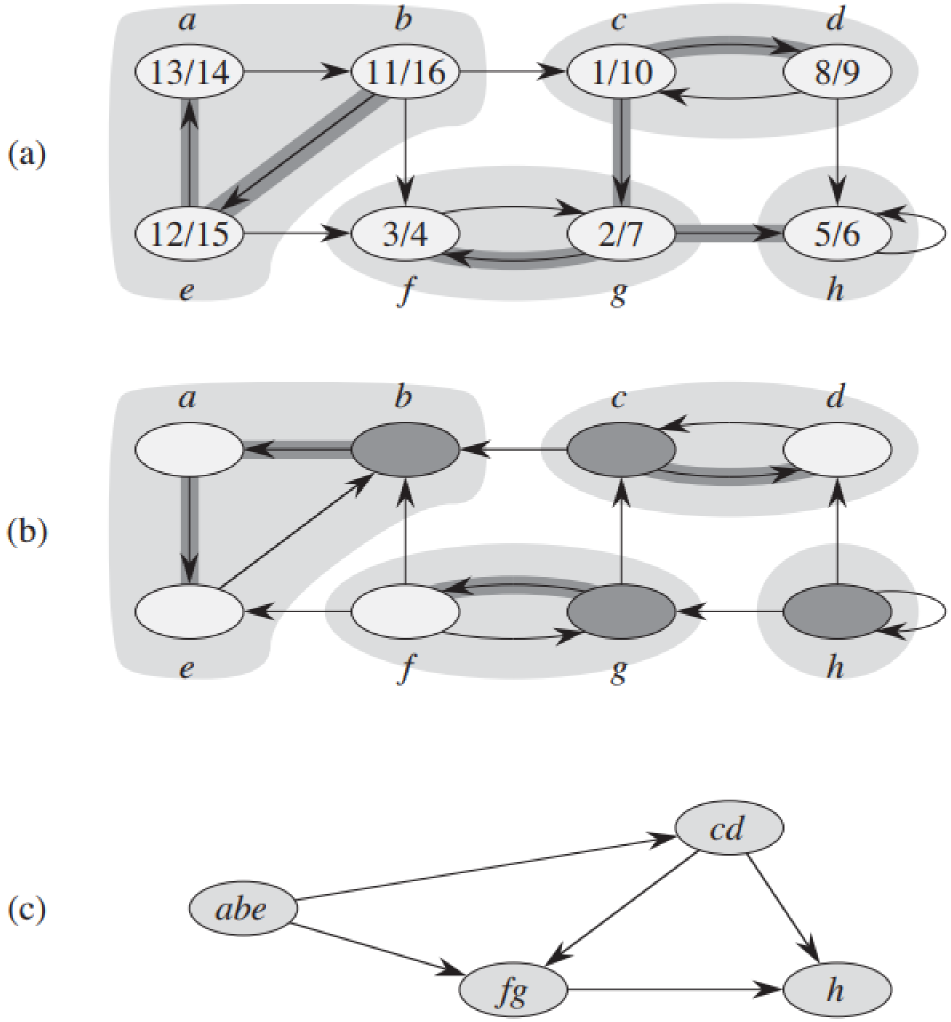


اگه توی گراف یال B نداشته باشیم ← DAG

TOPOLOGICAL-SORT(G)

call DFS(G) to compute finishing times $v.f$ for each vertex v
as each vertex is finished, insert it onto the front of a linked list
return the linked list of vertices

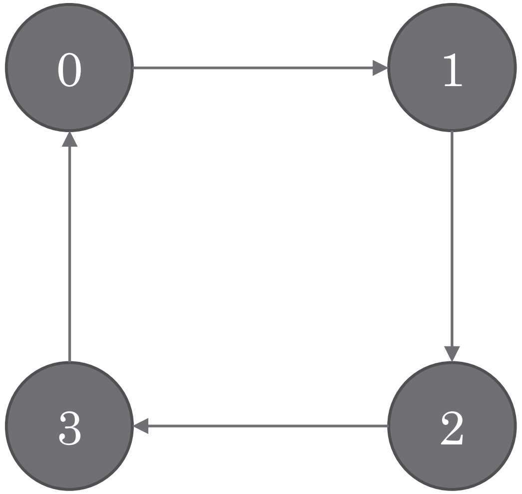
Strongly connected components اجزا با همبندی قوی



$G = (V, E)$

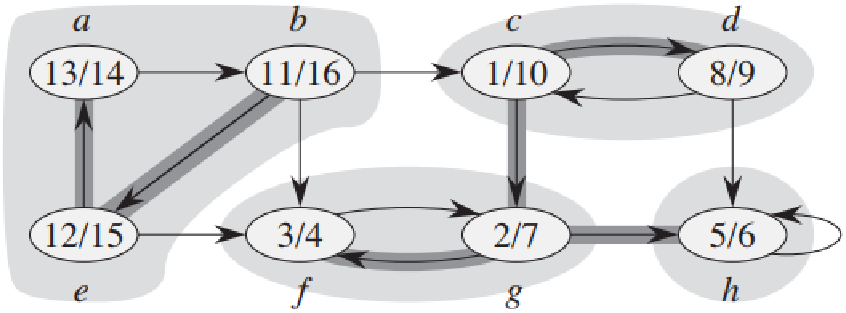
$C \subseteq V \qquad u \leadsto v \qquad v \leadsto u$

اجزا با همبندی قوی Strongly connected components



گراف قویا همبند

اجزا با همبندی قوی



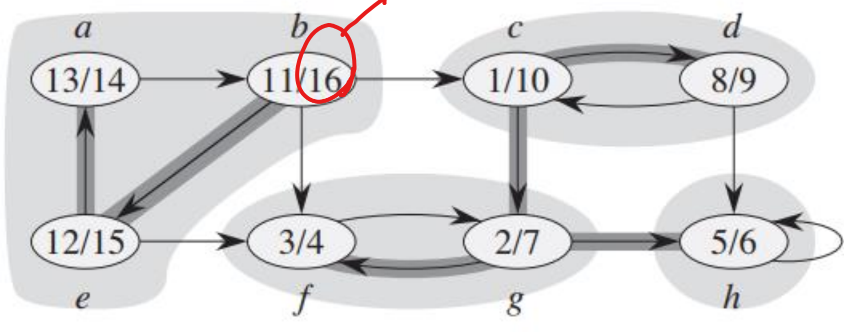
$G = (V, E)$

$C \subseteq V \qquad u \rightsquigarrow v \qquad v \rightsquigarrow u$

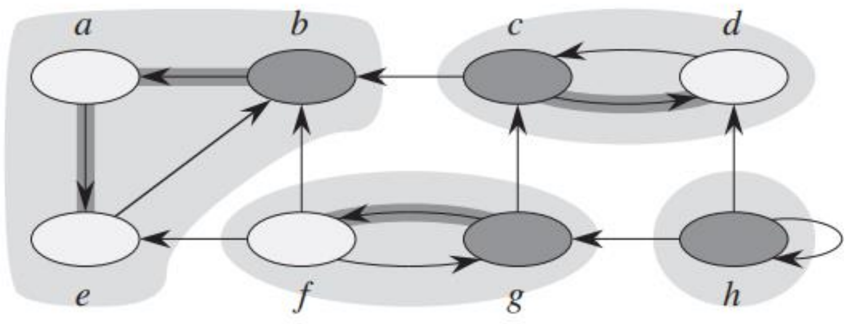
Strongly connected components اجزا با همبندی قوی

e max → اول

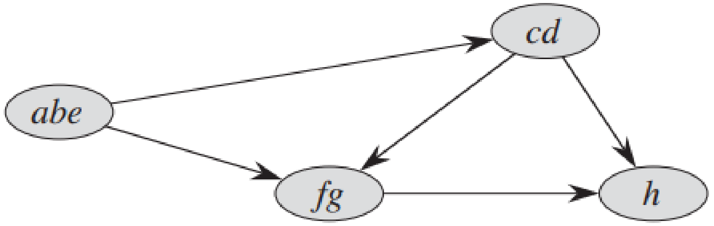
(a)



(b)



(c)



برعکس کردن یال‌ها

STRONGLY-CONNECTED-COMPONENTS(G)

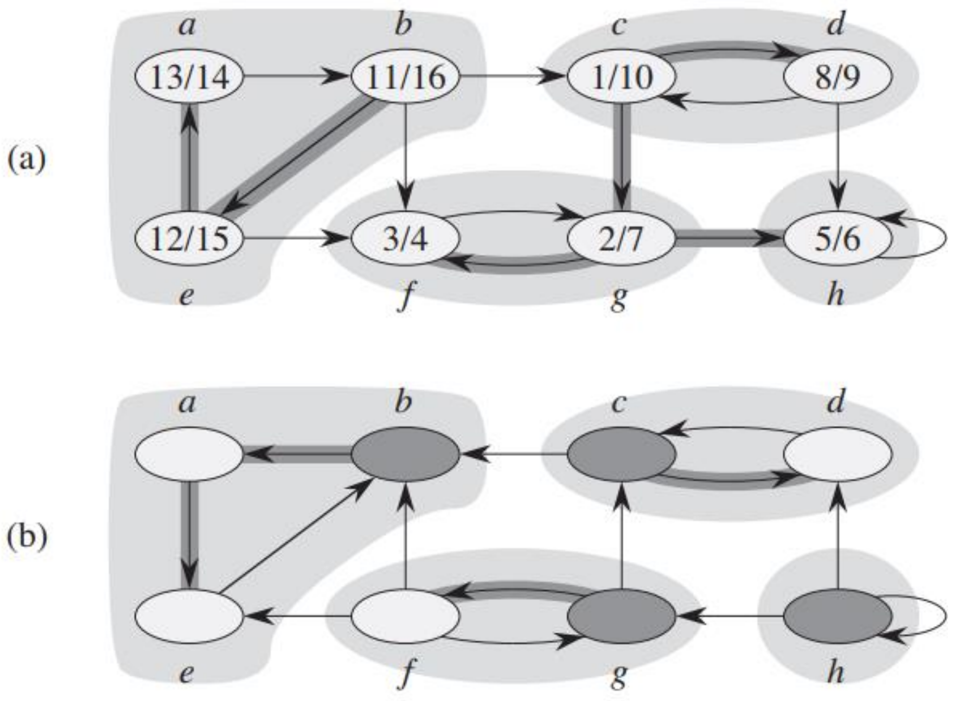
1 call DFS(G) to compute finishing times $u.f$ for each vertex u
2 compute G^{T}
3 call DFS(G^{T}), but in the main loop of DFS, consider the vertices in order of decreasing $u.f$ (as computed in line 1)
4 output the vertices of each tree in the depth-first forest formed in line 3 as a separate strongly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

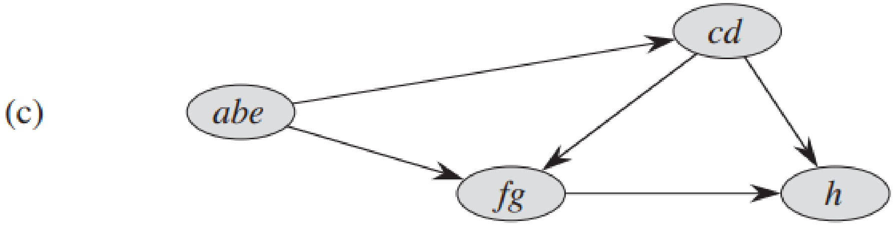
finish time از بیشترین

```
DFS(G)
  for each vertex u ∈ G.V
      u.color = WHITE
      u.π = NIL
  time = 0
  for each vertex u ∈ G.V
      if u.color == WHITE
          DFS-VISIT(G, u)

DFS-VISIT(G, u)
  time = time + 1
  u.d = time
  u.color = GRAY
  for each v ∈ G.Adj[u]
      if v.color == WHITE
          v.π = u
          DFS-VISIT(G, v)
  u.color = BLACK
  time = time + 1
  u.f = time
```

اجزا با همبندی قوی Strongly connected components





STRONGLY-CONNECTED-COMPONENTS(G)

1 call DFS(G) to compute finishing times $u.f$ for each vertex u
2 compute G^{T}
3 call DFS(G^{T}), but in the main loop of DFS, consider the vertices in order of decreasing $u.f$ (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

```
DFS(G)
for each vertex u ∈ G.V
    u.color = WHITE
    u.π = NIL
time = 0
for each vertex u ∈ G.V
    if u.color == WHITE
        DFS-VISIT(G, u)

DFS-VISIT(G, u)
time = time + 1
u.d = time
u.color = GRAY
for each v ∈ G.Adj[u]
    if v.color == WHITE
        v.π = u
        DFS-VISIT(G, v)
u.color = BLACK
time = time + 1
u.f = time
```