# CS 412 – Week 07

## Depth-First Search, Topological Sort, and SCCs

Shah Jamal Alam

# Depth-first Search (DFS)

- Searches "deep" in the graph, whenever possible.

- Each **vertex** $\boldsymbol{u}$ is initially **white**; **grayed** when it is discovered; and **blackened** when it is finished.

- Uses two **timestamps**: each vertex $\boldsymbol{u}$ has two timestamps
  - The first timestamp $\boldsymbol{u}._{\boldsymbol{d}}$ records when it is first discovered and grayed.
  - The second timestamp $\boldsymbol{u}._{\boldsymbol{f}}$ records when the search finishes examining $\boldsymbol{u}$*'s* adjacency list and blackens $\boldsymbol{u}$*.*

- **Timestamps** are between $\mathbf{1}$ and $\mathbf{2}|\boldsymbol{V}|$ (why?)

  For every vertex $\boldsymbol{u}$, $\boldsymbol{u}._{\boldsymbol{d}} < \boldsymbol{u}_{.\boldsymbol{f}}$

# Depth-first Search (DFS)

**Input:** $G = (V, E)$, directed or undirected. No source vertex given!

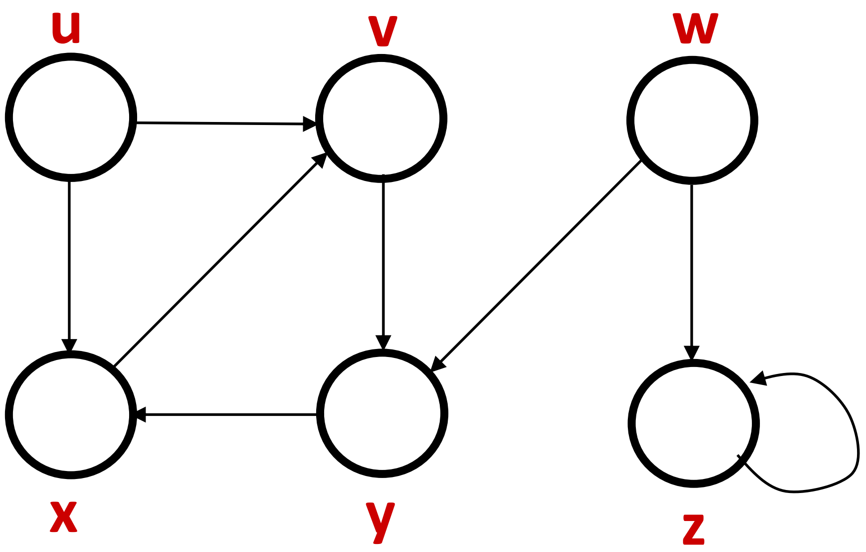
**Output:**

- **Two timestamps** on each vertex.
- $\pi[v]$: **predecessor of $\boldsymbol{v} = \boldsymbol{u}$**, such that $\boldsymbol{v}$ was discovered during the scan of $\boldsymbol{u}$'s adjacency list.

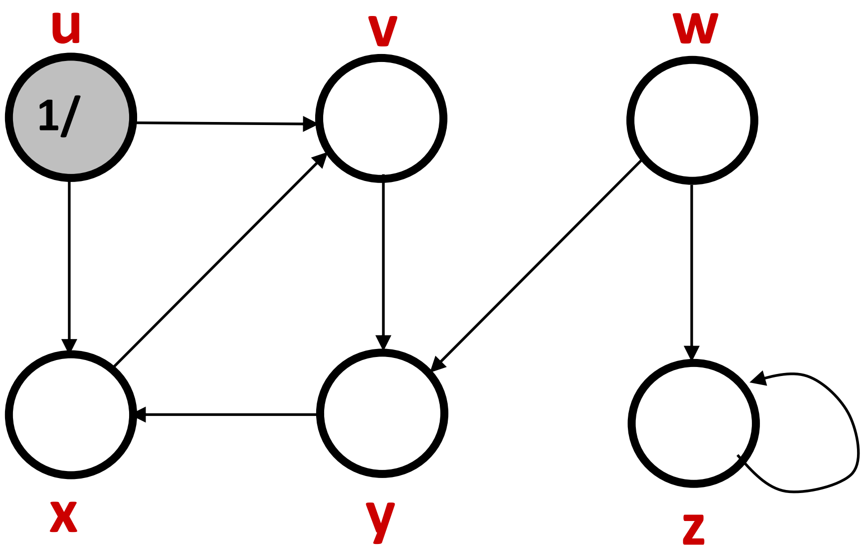
- Uses the same coloring scheme for vertices as BFS.

Source: TBA.

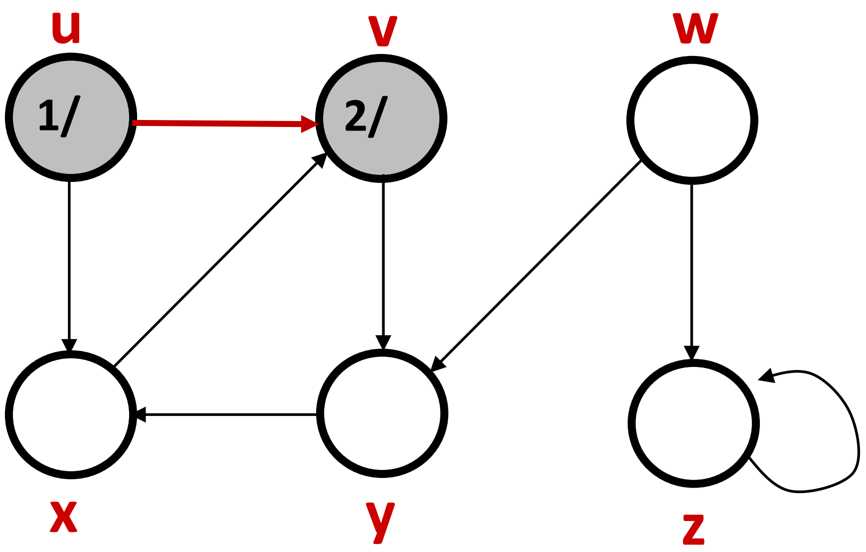
# **Depth-first Search (DFS):** an example



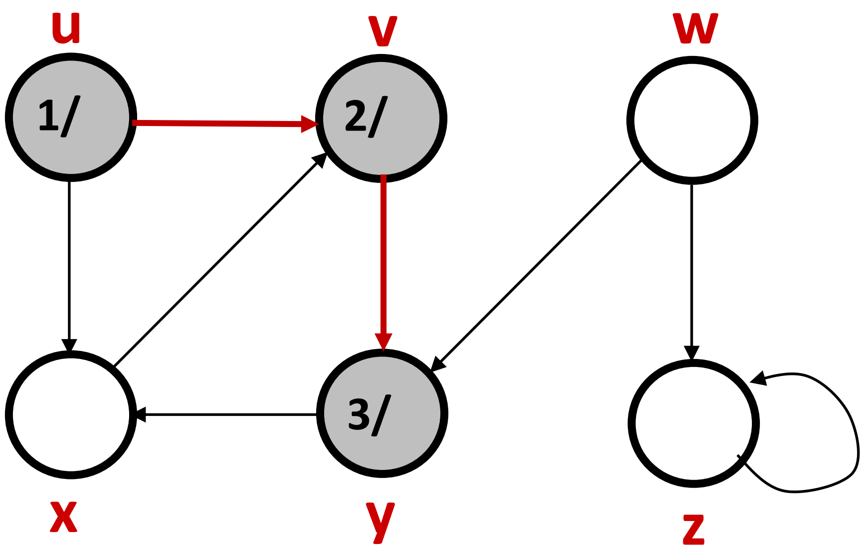
# **Depth-first Search (DFS):** an example



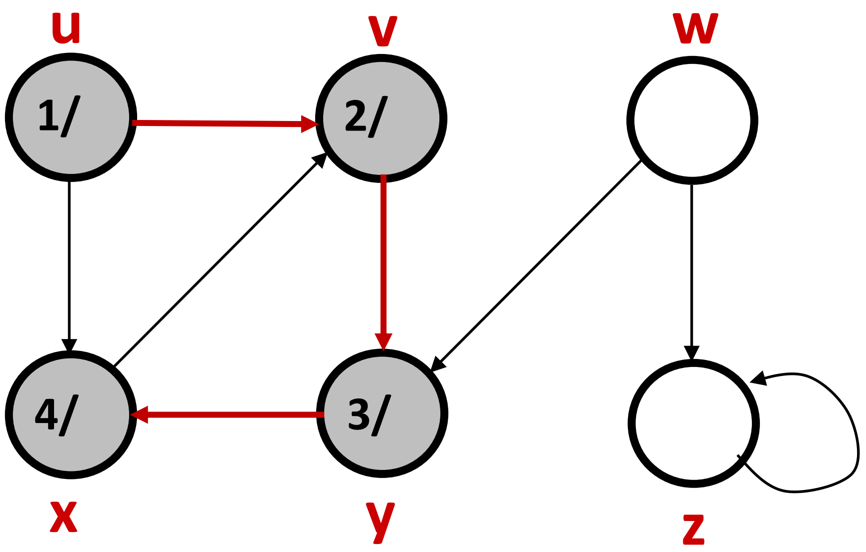
# **Depth-first Search (DFS):** an example



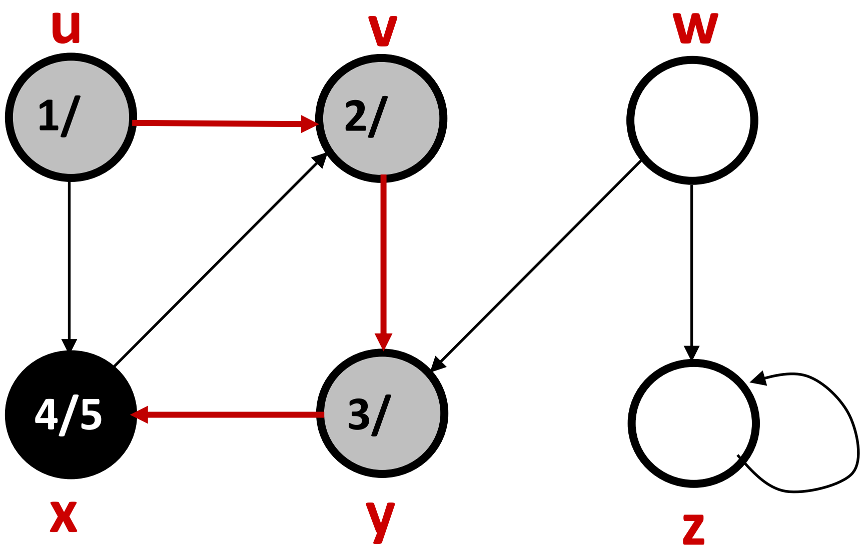
# **Depth-first Search (DFS):** an example



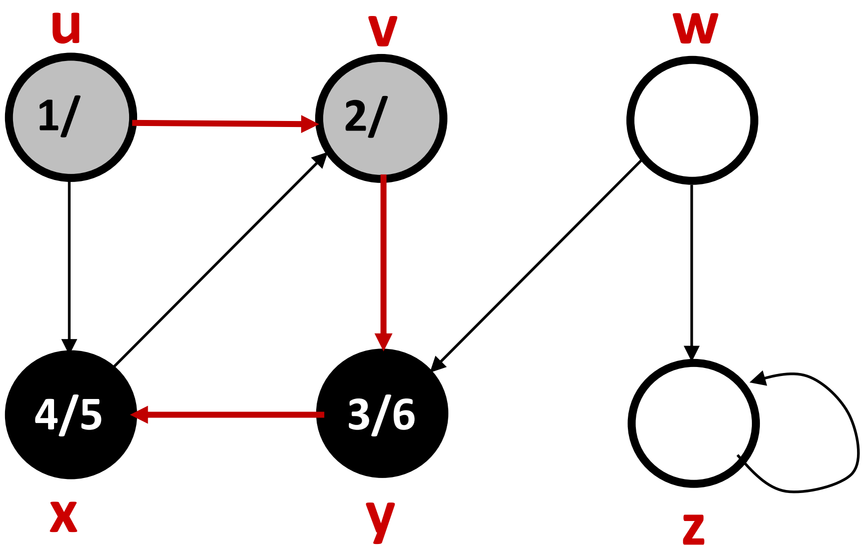
# **Depth-first Search (DFS):** an example



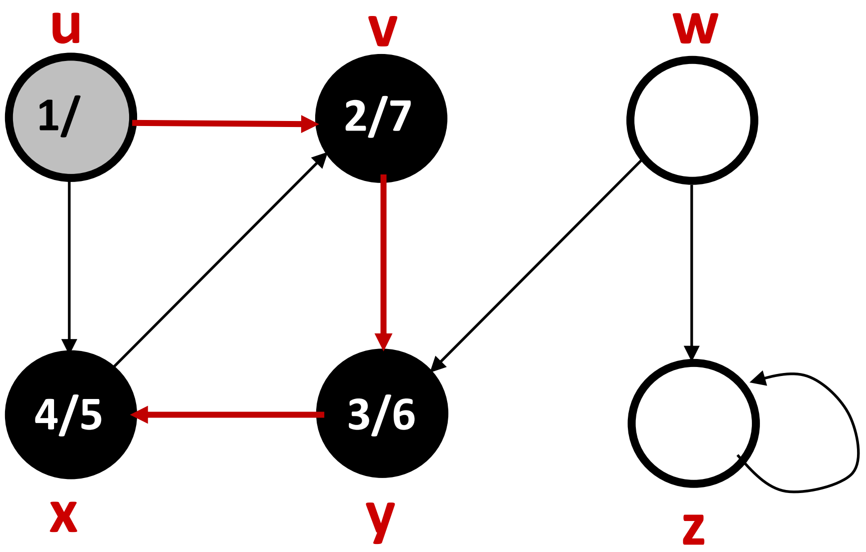
# **Depth-first Search (DFS):** an example



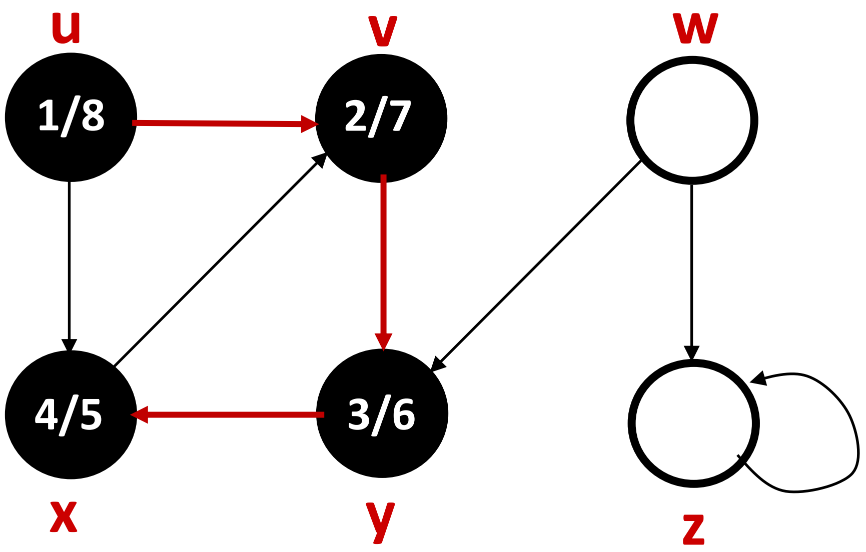
# **Depth-first Search (DFS):** an example



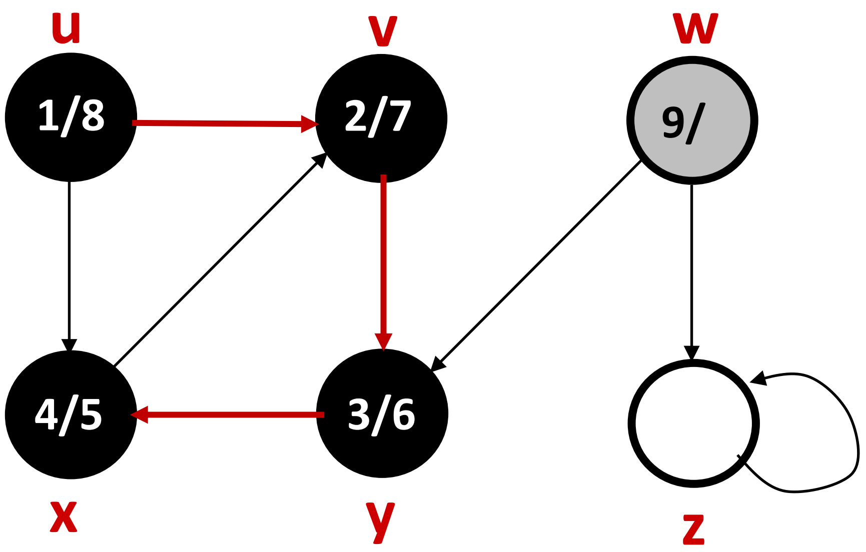
# **Depth-first Search (DFS):** an example



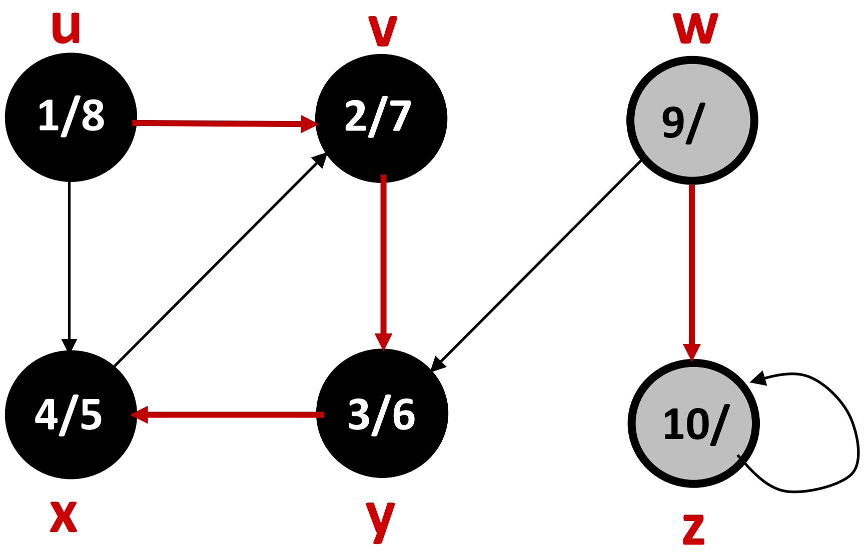
# **Depth-first Search (DFS):** an example



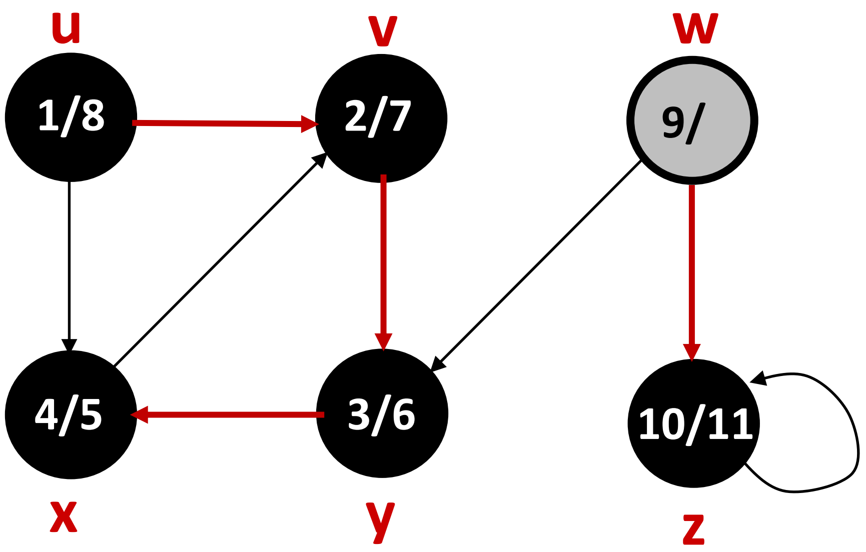
# **Depth-first Search (DFS):** an example



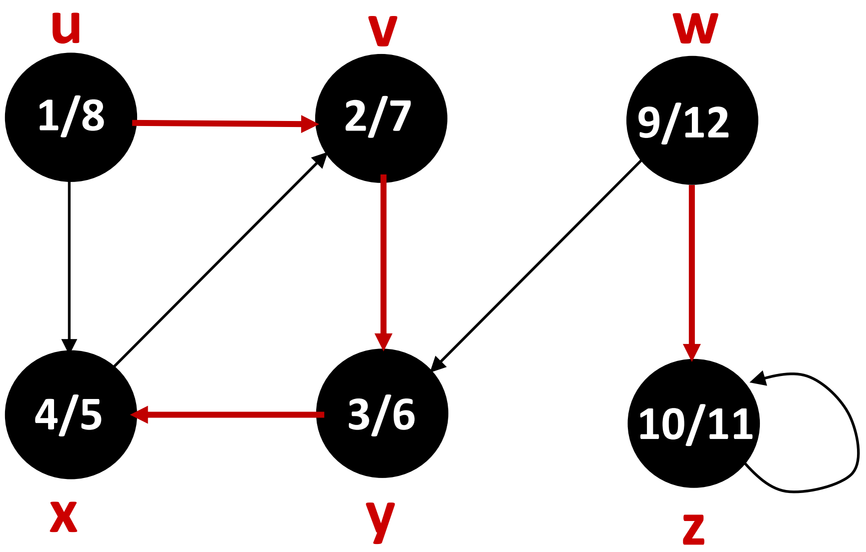
# **Depth-first Search (DFS):** an example



# **Depth-first Search (DFS):** an example



# **Depth-first Search (DFS):** an example



# Pseudocode

**DFS(*G*)**

```
for each vertex u ∈ V[G]
    do color[u] ← white
        π[u] ← NIL
time ← 0
for each vertex u ∈ V[G]
    do if color[u] = white
        then DFS-Visit(u)
```

**Uses a global timestamp *time*.**

DFS-Visit(*u*)

```
color[u] ← GRAY   ∇ White vertex u has been
discovered
time ← time + 1
 d[u] ← time
 for each v ∈ Adj[u]
     do if color[v] = WHITE
         then π[v] ← u
             DFS-Visit(v)
 color[u] ← BLACK    ∇ Blacken u;  it is finished.
 f[u] ← time ← time + 1
```

https://www.cs.unc.edu/~plaisted/

# Analysis of DFS

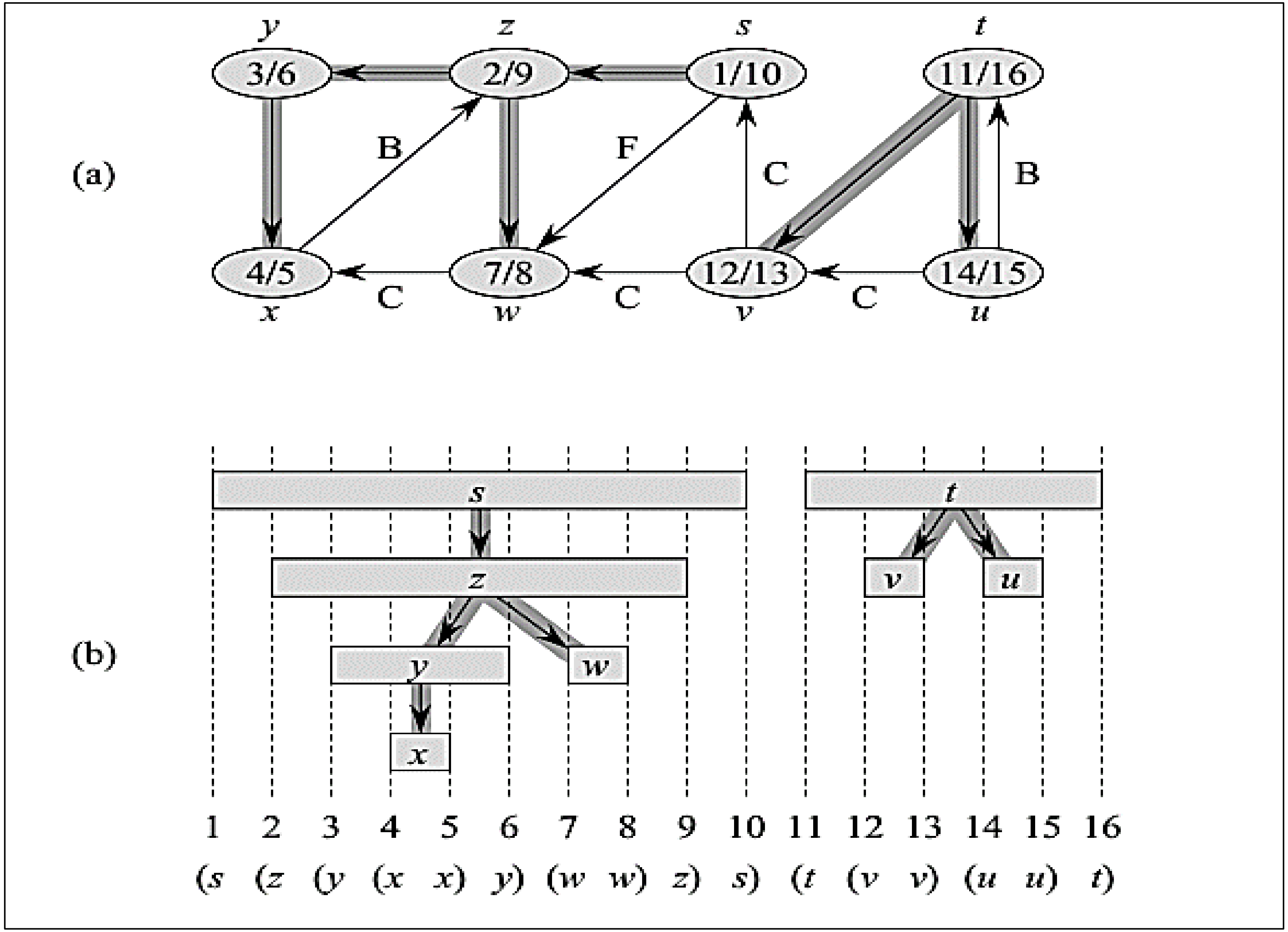
- Loops on lines 1-2 & 5-7 take $\Theta(|V|)$ time, excluding time to execute **DFS-Visit**.

- **DFS-Visit** is called once for each white vertex $\boldsymbol{v} \in \boldsymbol{V}$ when it is painted gray the first time. Lines 3-6 of **DFS-Visit** is executed $|\boldsymbol{Adj}[\boldsymbol{v}]|$ times. The total cost of executing **DFS-Visit** is $\sum_{\boldsymbol{v} \in \boldsymbol{V}} |\boldsymbol{Adj}[\boldsymbol{v}]| = \Theta(|\boldsymbol{E}|)$.

- **Total running time of DFS is $\Theta(|\boldsymbol{V}| + |\boldsymbol{E}|)$**

https://www.cs.unc.edu/~plaisted/

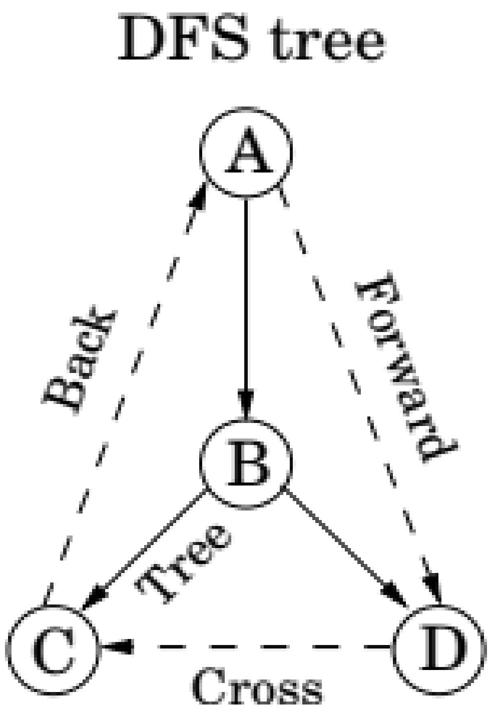
# Parenthesis structure property of DFS




Source: Dasgupta *et al.*

We can define four edge types in terms of the depth-first forest $G_\pi$ produced by a depth-first search on $G$:

1. ***Tree edges*** are edges in the depth-first forest $G_\pi$. 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.



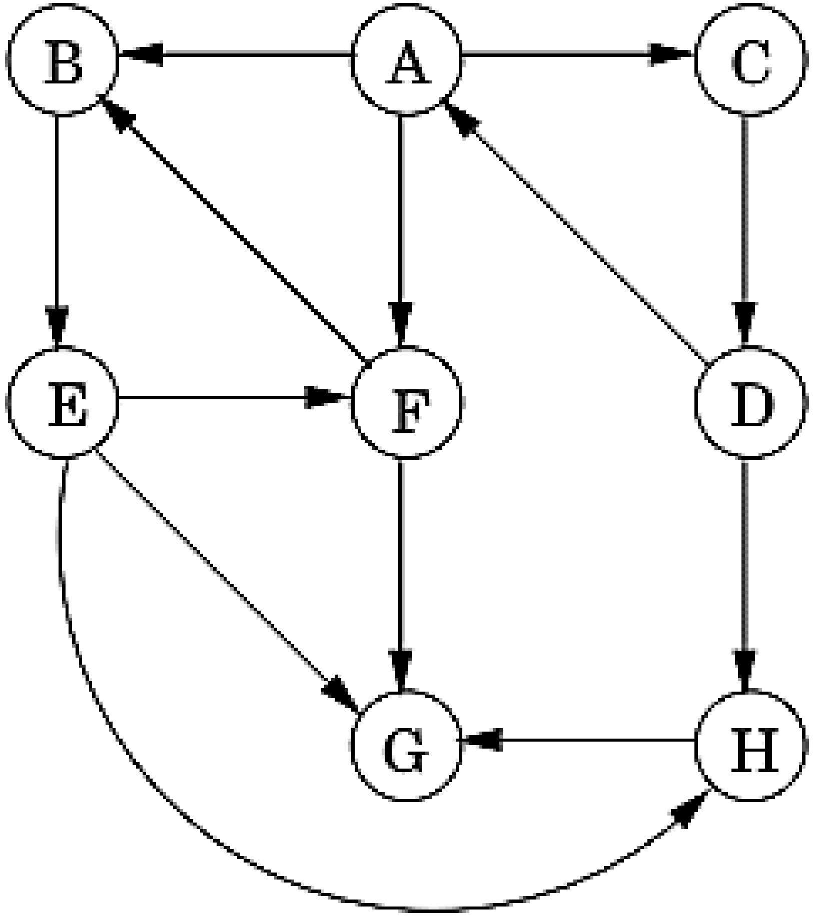

Show that edge $(u, v)$ is

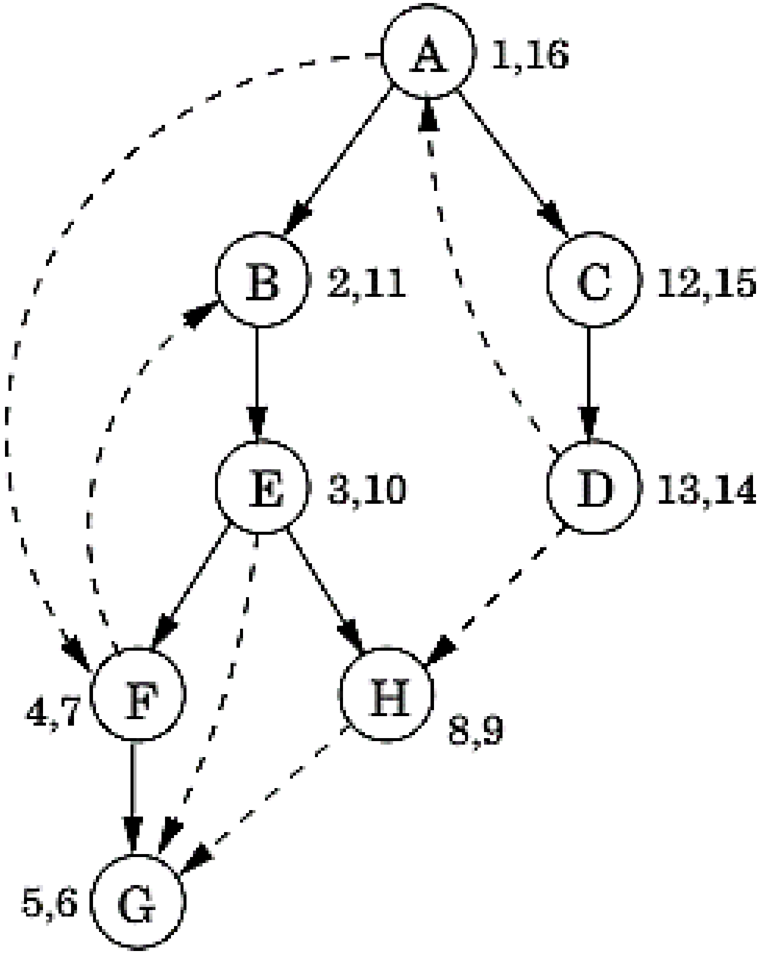
***a.*** a tree edge or forward edge if and only if $u.d < v.d < v.f < u.f$,

***b.*** a back edge if and only if $v.d \le u.d < u.f \le v.f$, and

***c.*** a cross edge if and only if $v.d < v.f < u.d < u.f$.

Source: Dasgupta *et al.*



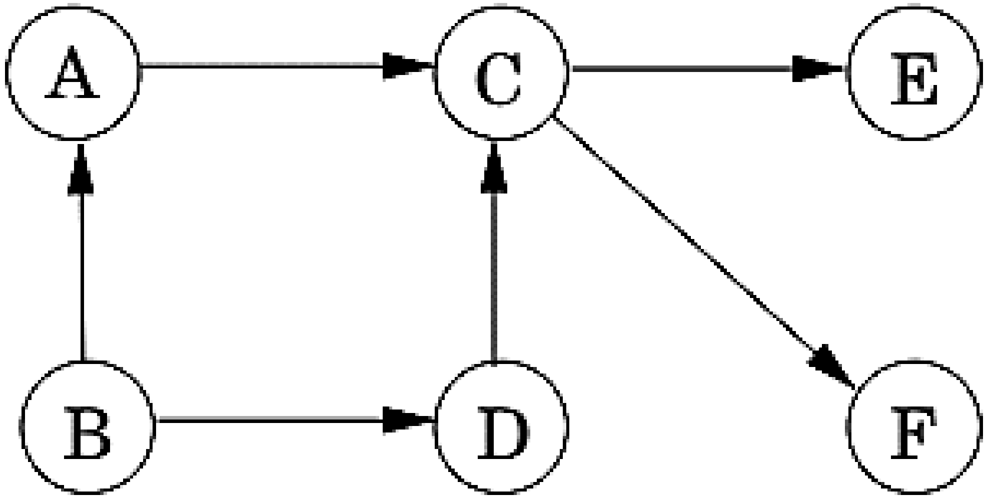




| `pre`/`post` *ordering for* $(u, v)$ | *Edge type* |
| --- | --- |
| $[_u \; [_v \; ]_v \; ]_u$ | Tree/forward |
| $[_v \; [_u \; ]_u \; ]_v$ | Back |
| $[_v \; ]_v \; [_u \; ]_u$ | Cross |

Source: Dasgupta *et al.*

# Directed Acyclic Graph (DAG)

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




**Property** *In a dag, every edge leads to a vertex with a lower* `post` *number.*

**Every *dag* can be linearized!**

Source: Dasgupta *et al.*

$S_1$ $a := 0$

$S_2$ $b := 1$

$S_3$ $c := a + 1$

$S_4$ $d := b + a$

$S_5$ $e := d + 1$

$S_6$ $e := c + d$

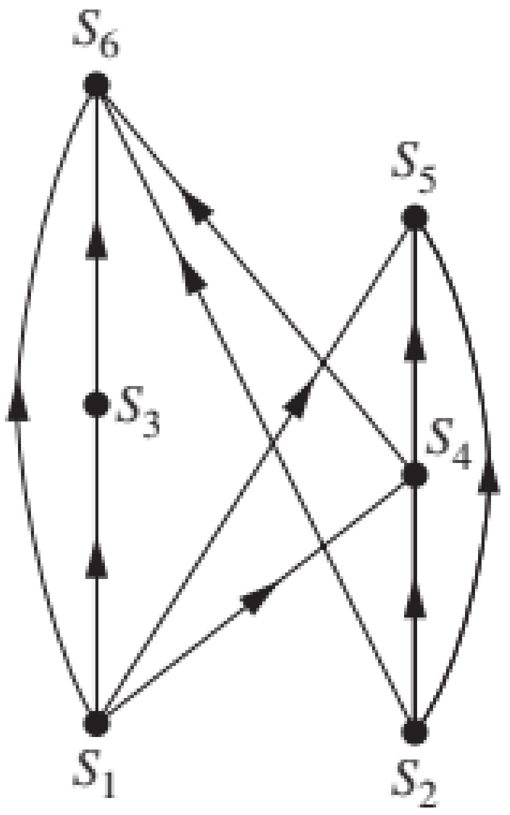


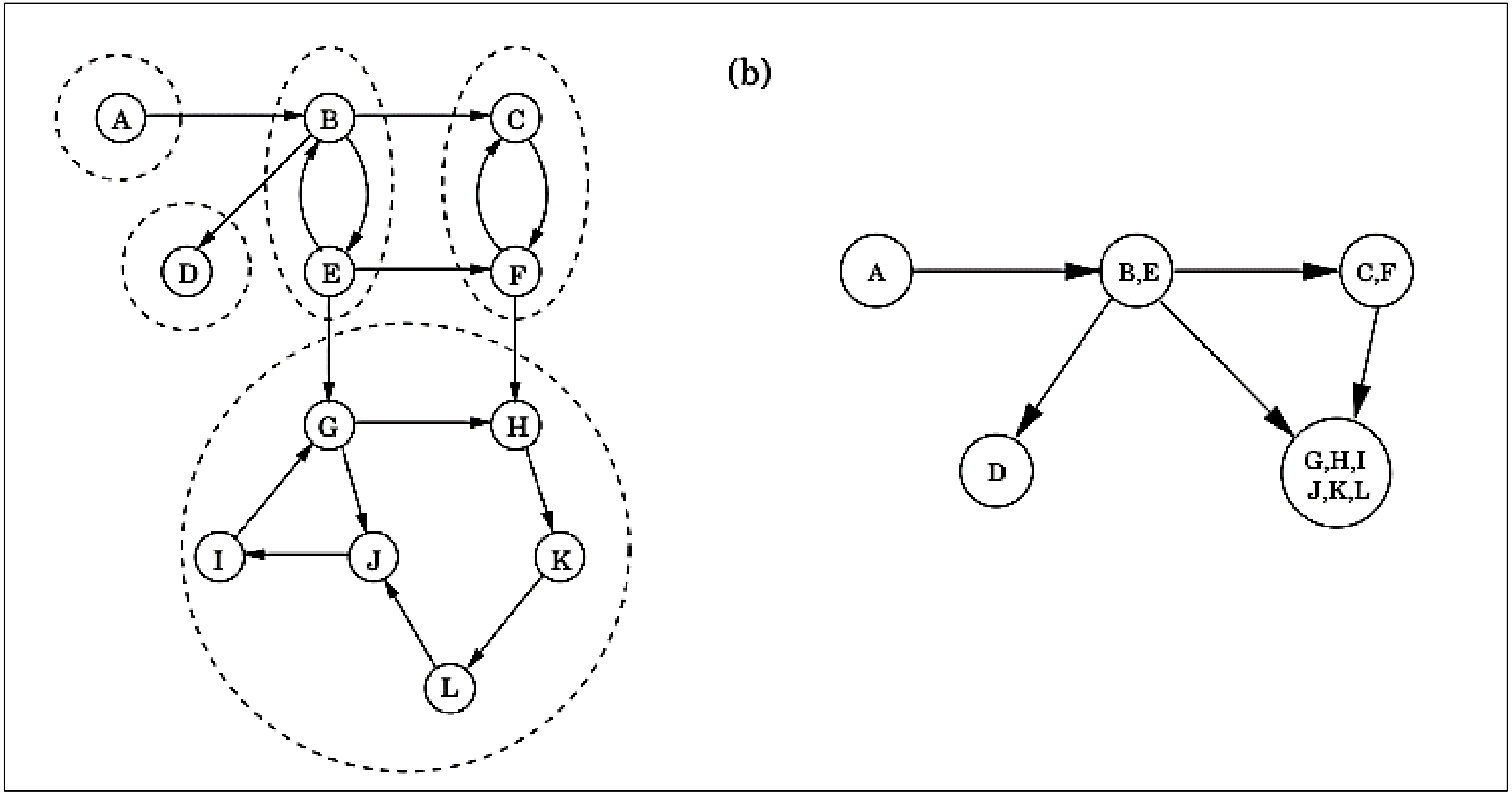

**FIGURE 10** **A Precedence Graph.**

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
```

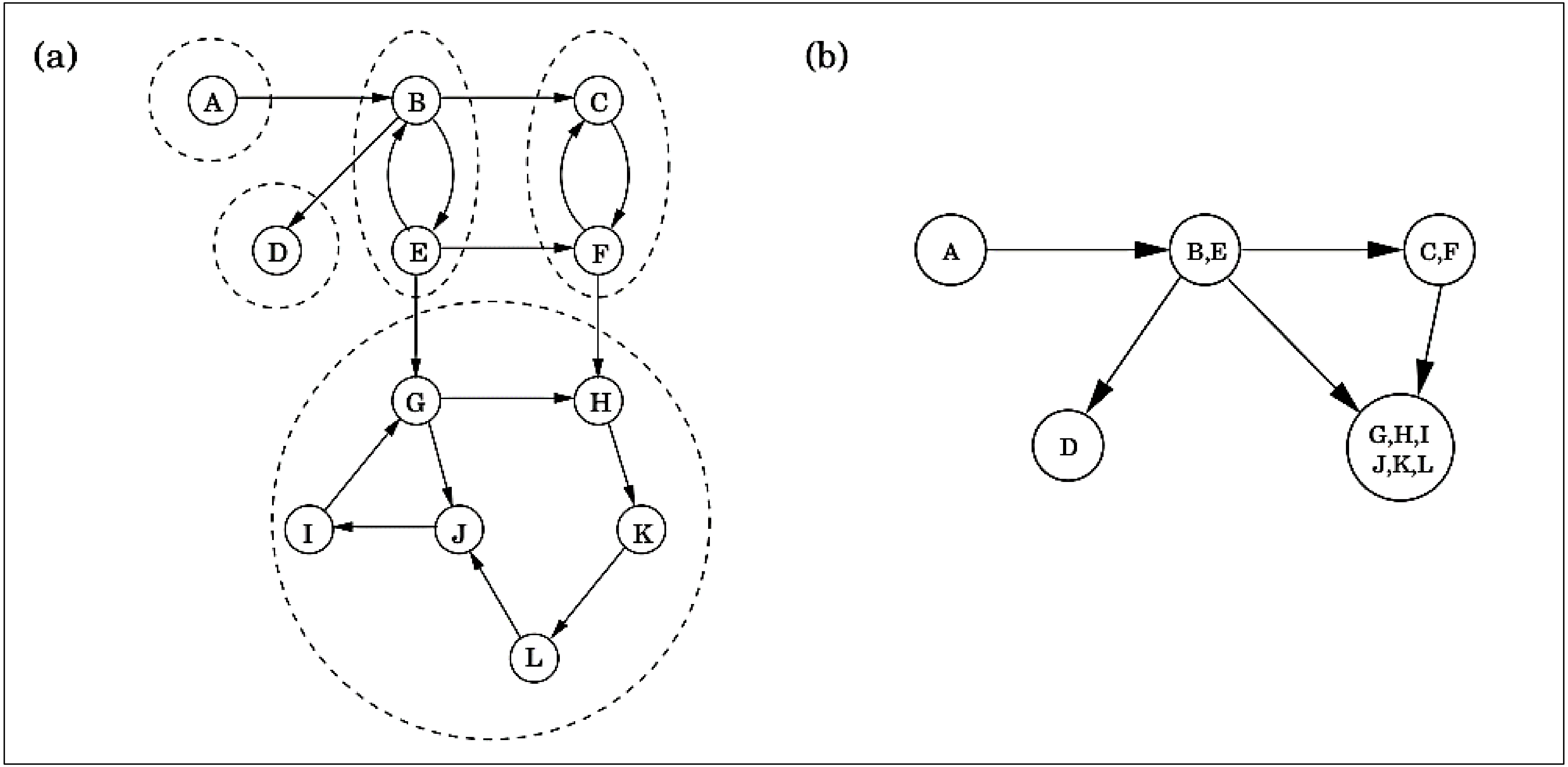
Source: Rosen

**Property** *Every directed graph is a dag of its strongly connected components.*




Source: Dasgupta *et al.*

**Property 1** *If the* `explore` *subroutine is started at node $u$, then it will terminate precisely when all nodes reachable from $u$ have been visited.*




Source: Dasgupta *et al.*

# The basic idea

- Run DFS on a node $\boldsymbol{u}$ in the given directed graph $G = (V, E)$.
- If we're lucky, then the node $\boldsymbol{u}$ belongs to the ***sink*** SCC in $G$.

**Challenge:**

- How do we find a node that belongs to the ***sink*** SCC in $G$?
- How do we continue once all nodes in the ***sink*** SCC are discovered?
- How do we find all SCCs in $O(n + m)$?

**Trick:**

- Finding a node in the ***source*** SCC is *easy.*
- To pick a node that belong in the *source* SCC, run DFS. The node in the highest finishing time (post-time) is in a source SCC.

Source: Dasgupta *et al.*

# Finding SCCs in a directed graph using DFS

1. Run DFS on $G = (V, E)$, calculating $u_{.d}$ and $u_{.f}$ for all nodes $u$ in $V$.
2. Run DFS on $G_{rev}$, picking nodes in *decreasing* finishing time (calculated from Step 1).

   [Every tree in DFS of $G_{rev}$ will be a SCC in $G$; recall the Parenthesis property].
3. Repeat Step 2.

Source: Dasgupta *et al.*