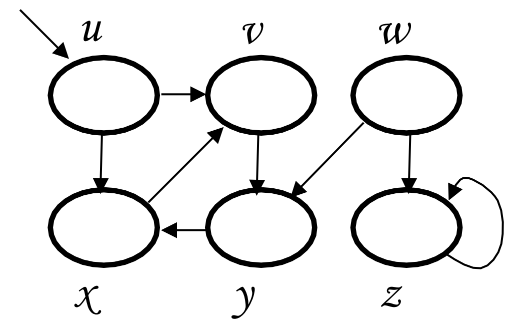
CSE 201
Data Structure and Algorithm

Lecture 3

DFS (Revisited) & Topological Sort

DFS(V, E)

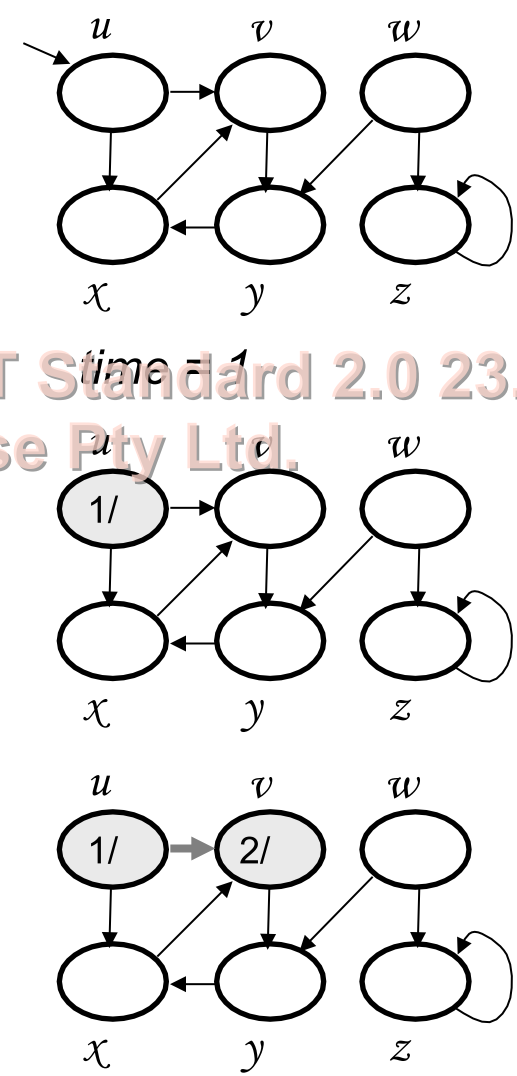
1. **for** each u ∈ V
2. **do** color[u] ← WHITE
3. prev[u] ← NIL
4. time ← 0
5. **for** each u ∈ V
6. **do if** color[u] = WHITE
7. **then** DFS-VISIT(u)



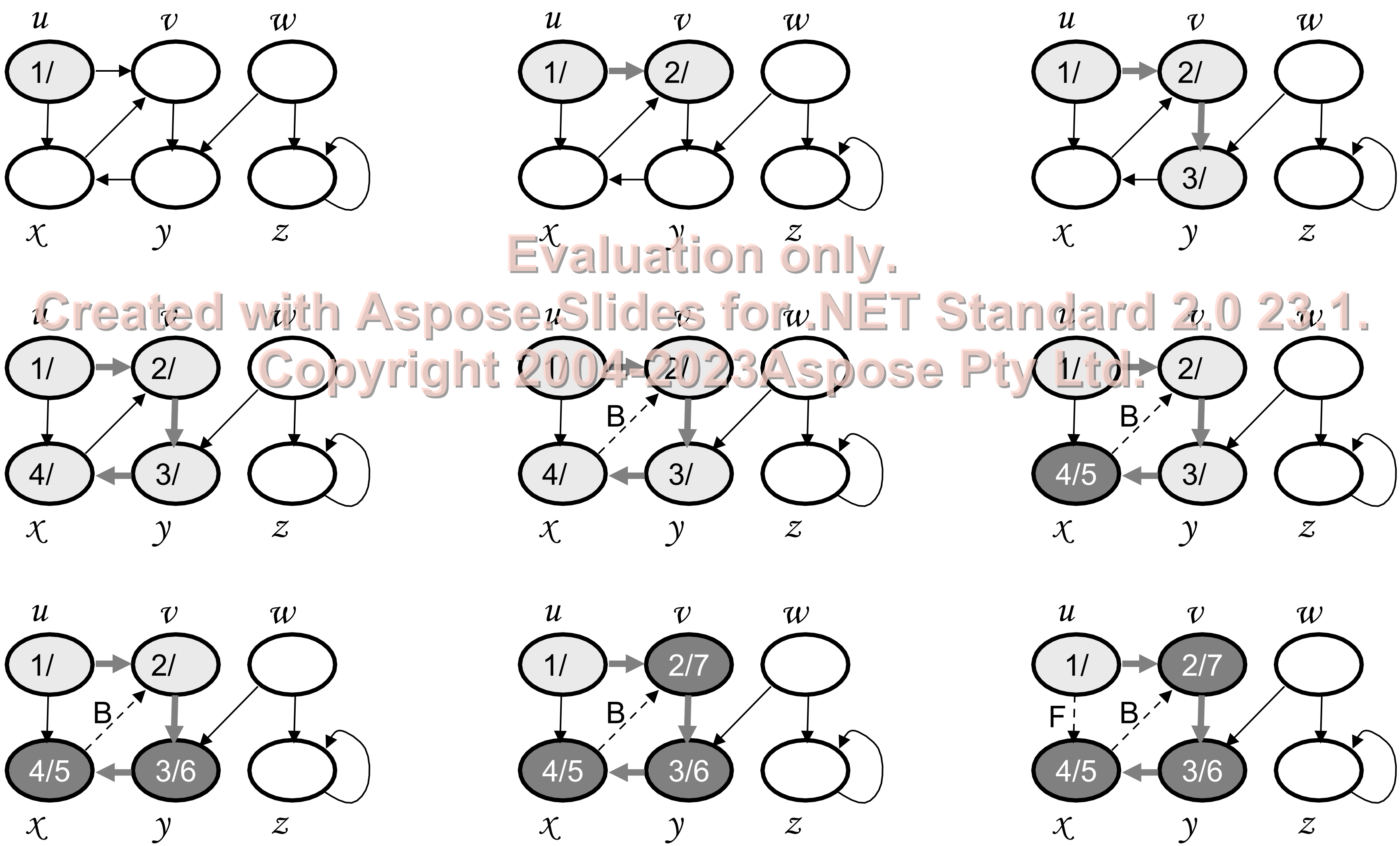
- Every time DFS-VISIT(u) is called, u becomes the root of a new tree in the depth-first forest

DFS-VISIT(u)

```
color[u] ← GRAY
time ← time+1
d[u] ← time
for each v ∈ Adj[u]
        do if color[v] = WHITE
                then prev[v] ← u
                        DFS-VISIT(v)
color[u] ← BLACK
time ← time + 1
f[u] ← time
```

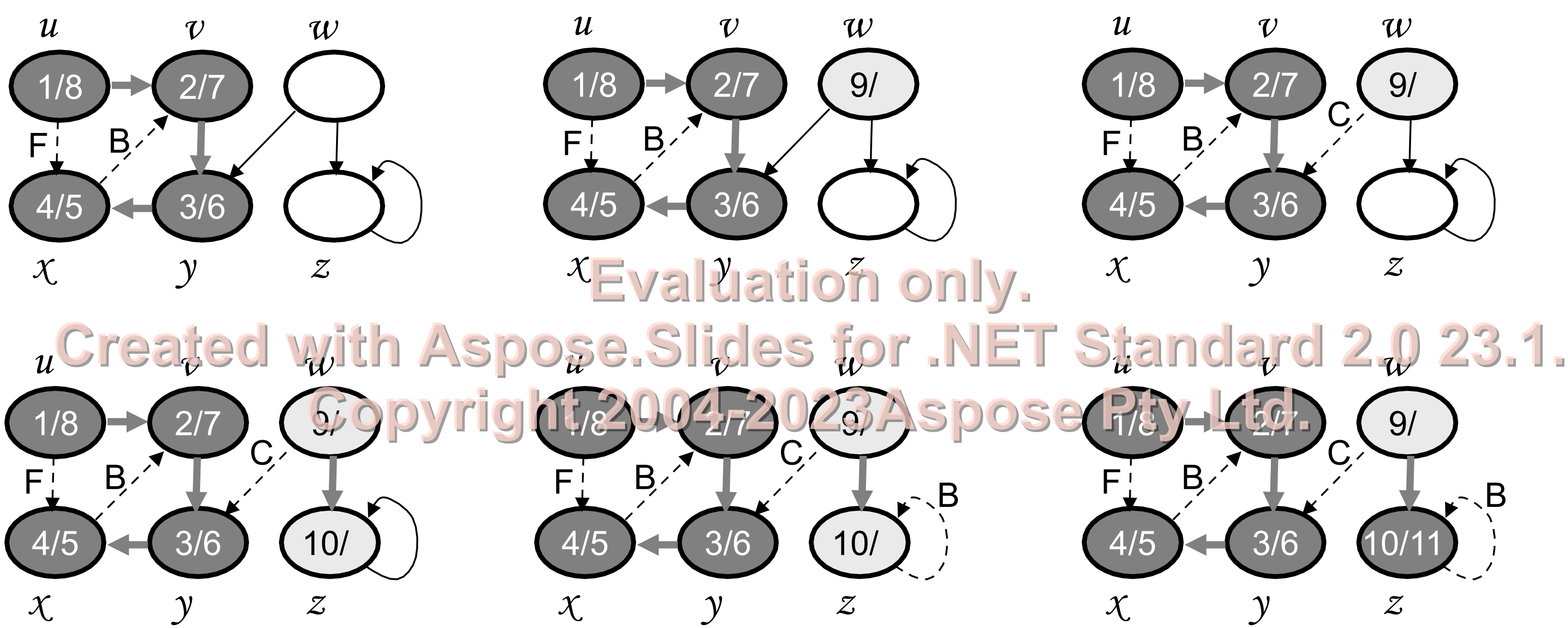


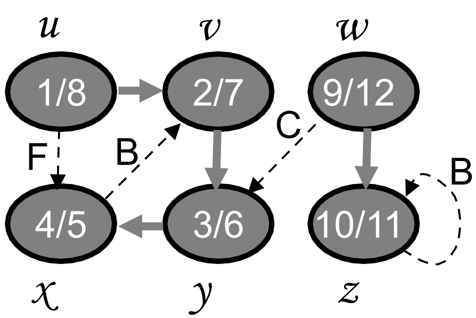
Example



u v w
1/
x y z
u v w
1/ 2/
x y z
u v w
1/ 2/
3/
x y z
u v w
1/ 2/
4/ 3/
x y z
u v w
1/ 2/
B
4/ 3/
x y z
u v w
1/ 2/
B
4/5 3/
x y z
u v w
1/ 2/
B
4/5 3/6
x y z
u v w
1/ 2/7
B
4/5 3/6
x y z
u v w
1/ 2/7
F B
4/5 3/6
x y z

Example (cont.)



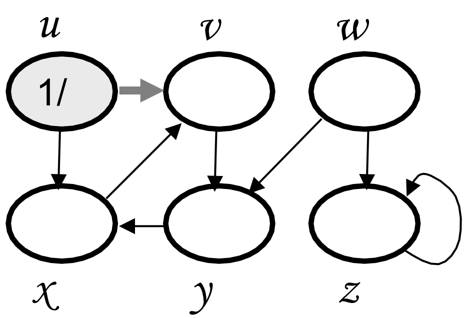


The results of DFS may depend on:

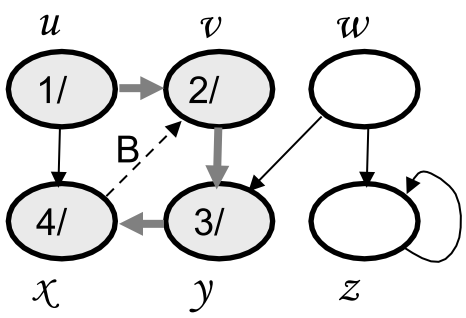
- The order in which nodes are explored in procedure DFS
- The order in which the neighbors of a vertex are visited in DFS-VISIT

Edge Classification

- **Tree edge** (reaches a WHITE vertex):
  - (u, v) is a tree edge if v was first discovered by exploring edge (u, v)

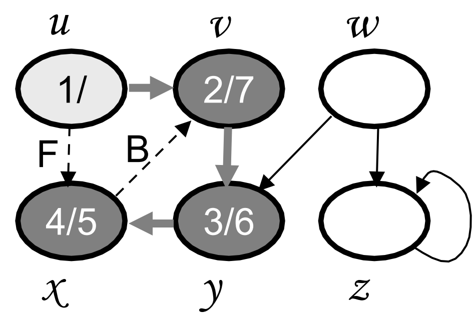


- **Back edge** (reaches a GRAY vertex):
  - (u, v), connecting a vertex u to an ancestor v in a depth first tree
  - Self loops (in directed graphs) are also back edges

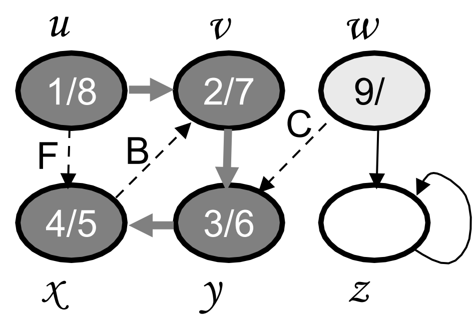


Edge Classification

- **Forward edge** (reaches a BLACK vertex & d[u] < d[v]):
  - Non-tree edges (u, v) that connect a vertex u to a descendant v in a depth first tree



- **Cross edge** (reaches a BLACK vertex & d[u] > d[v]):
  - Can go between vertices in same depth-first tree (as long as there is no ancestor / descendant relation) or between different depth-first trees



Analysis of DFS(V, E)

1. **for** each u ∈ V
2. **do** color[u] ← WHITE
3. π[u] ← NIL
4. time ← 0
5. **for** each u ∈ V
6. **do if** color[u] = WHITE
7. **then** DFS-VISIT(u)

Θ(V)

Θ(V) – exclusive of time for DFS-VISIT

Analysis of DFS-VISIT(u)

1. color[u] $\leftarrow$ GRAY
2. time $\leftarrow$ time+1
3. d[u] $\leftarrow$ time
4. **for** each v $\in$ *Adj*[u]
5. **do if** color[v] = WHITE
6. **then** π[v] $\leftarrow$ u
7. DFS-VISIT(v)
8. color[u] $\leftarrow$ BLACK
9. time $\leftarrow$ time + 1
10. f[u] $\leftarrow$ time

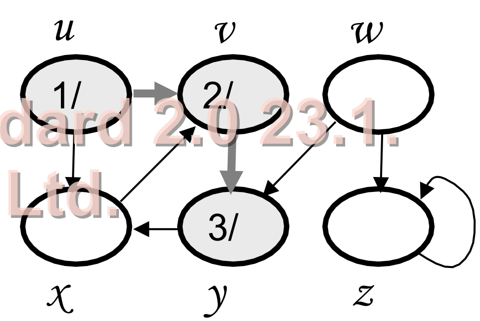
DFS-VISIT is called exactly once for each vertex

Each loop takes |Adj[v]|

Total: $\underbrace{\Sigma_{v \in V} |Adj[v]|}_{\Theta(E)} + \Theta(V) = \Theta(V + E)$

Properties of DFS

- u = prev[v] ⇔ DFS-VISIT(v) was called during a search of u's adjacency list

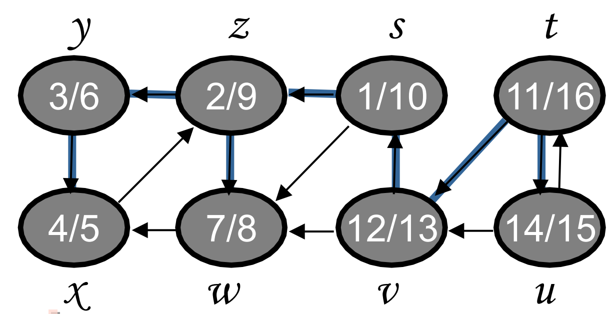


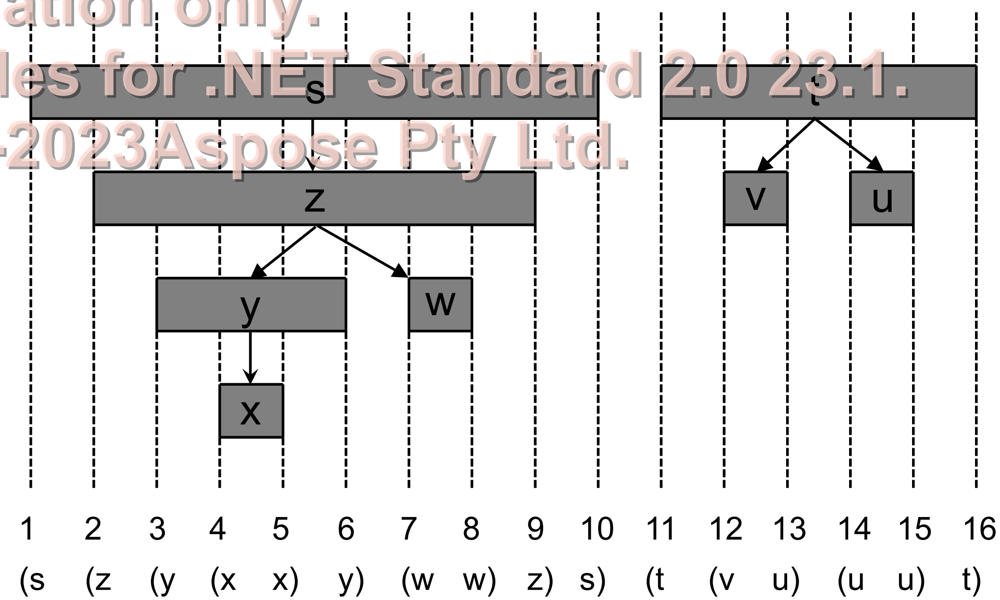
- Vertex v is a descendant of vertex u in the depth first forest ⇔ v is discovered during the time in which u is gray

Parenthesis Theorem

In any DFS of a graph G, for all u, v, exactly one of the following holds:

1. [d[u], f[u]] and [d[v], f[v]] are disjoint, and neither of u and v is a descendant of the other
2. [d[v], f[v]] is entirely within [d[u], f[u]] and v is a descendant of u
3. [d[u], f[u]] is entirely within [d[v], f[v]] and u is a descendant of v





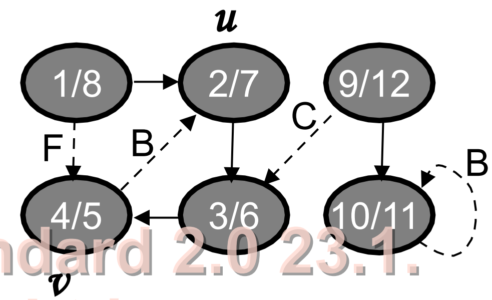
Well-formed expression: parenthesis are properly nested

Other Properties of DFS

Corollary

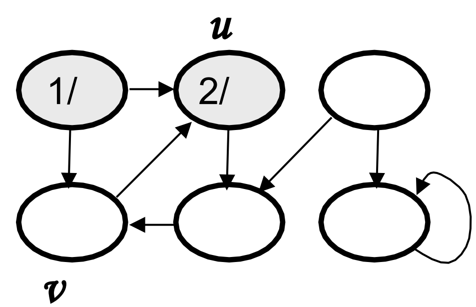
Vertex v is a proper descendant of u

$\Leftrightarrow$ d[u] < d[v] < f[v] < f[u]



Theorem (White-path Theorem)

In a depth-first forest of a graph G, vertex v is a descendant of u if and only if at time d[u], there is a path u ⇨ v consisting of only white vertices.

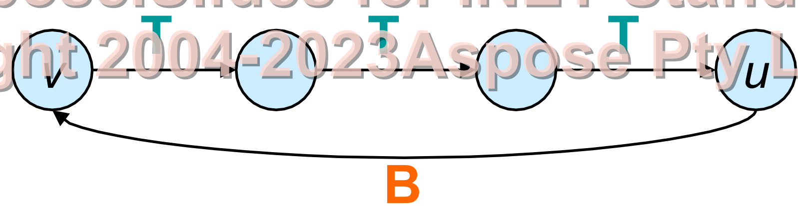


Directed Acyclic Graph

- DAG – Directed graph with no cycles.
- Good for modeling processes and structures that have a **partial order:**
  - $a > b$ and $b > c \Rightarrow a > c$.
  - But may have a and b such that neither $a > b$ nor $b > a$.
- Can always make a **total order** (either $a > b$ or $b > a$ for all $a \neq b$) from a partial order.

Characterizing a DAG

Lemma 22.11
A directed graph *G* is acyclic iff a DFS of G yields no back edges.



Topological Sort

Topological sort of a directed acyclic graph G = (V, E): a linear order of vertices such that if there exists an edge (u, v), then u appears before v in the ordering.

- Directed acyclic graphs (DAGs)
  - Used to represent precedence of events or processes that have a **partial order**

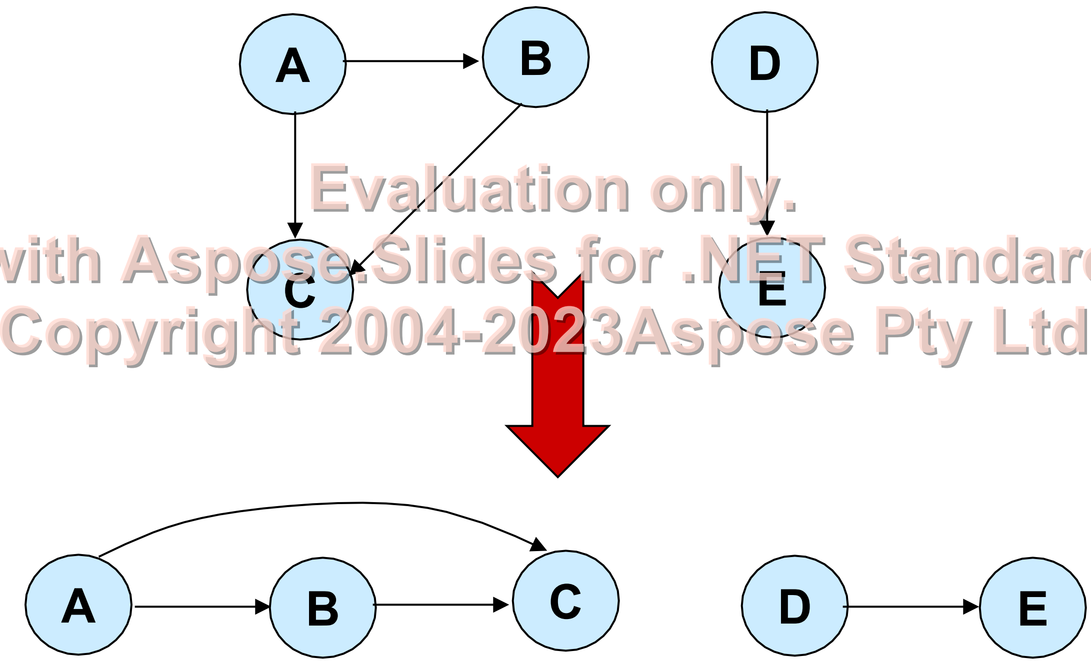
$\left.\begin{array}{l}\text{a before b}\\ \text{b before c}\end{array}\right\}$ a before c

$\left.\begin{array}{l}\text{b before c}\\ \text{a before c}\end{array}\right\}$ What about a and b?

Topological sort helps us establish a **total order**

Topological Sort

Want to “sort” a directed acyclic graph (DAG).



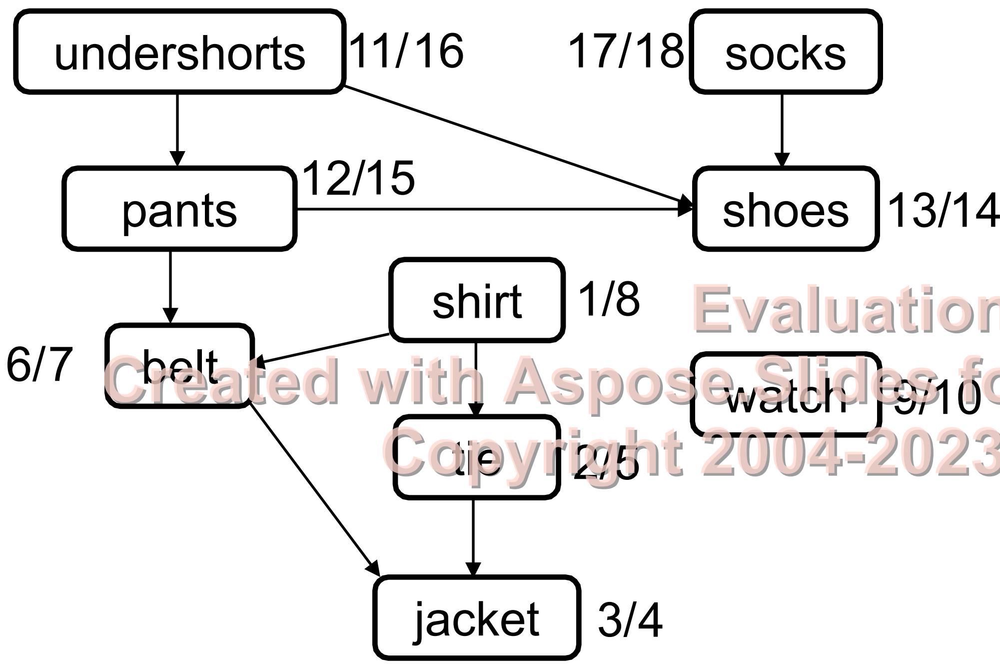
Think of original DAG as a **partial order**.

Want a **total order** that extends this partial order.

Topological Sort - Application

- Application 1
  - in scheduling a sequence of jobs.
  - The jobs are represented by vertices,
  - there is an edge from *x* to *y* if job *x* must be completed before job *y* can be done
    - (for example, washing machine must finish before we put the clothes to dry). Then, a topological sort gives an order in which to perform the jobs
- Application 2
  - In open credit system, how to take courses (in order) such that, pre-requisite of courses will not create any problem

Topological Sort (Fig – Cormen)



TOPOLOGICAL-SORT(V, E)

1. Call DFS(V, E) to compute finishing times f[v] for each vertex v
2. When each vertex is finished, insert it onto the front of a linked list
3. Return the linked list of vertices

socks | undershorts | pants | shoes | watch | shirt | belt | tie | jacket

Running time: $\Theta(V + E)$

Readings

- Cormen - Chapter 22
- Exercise:
  - 22.4-2 : Number of paths (important)
  - 22.4-3 : cycle (important and we have already solved it)
  - 22.4-5 : Topological sort using degree