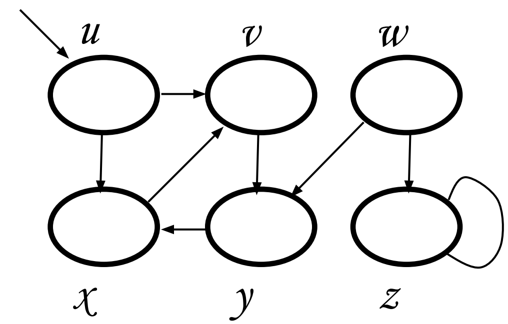
CSE
Design & Analysis of Algorithm

DFS (Revisited) & Topological Sort

DFS(V, E)

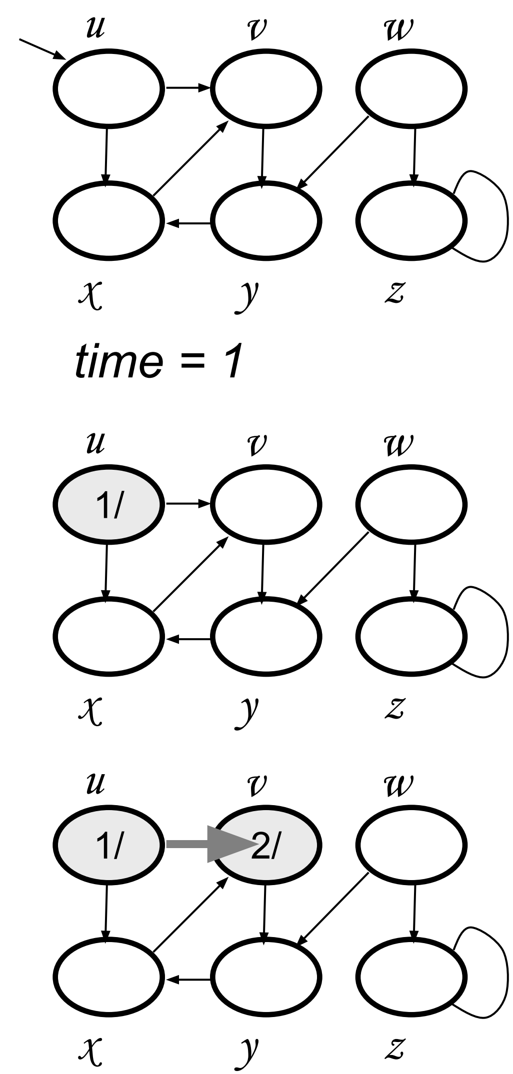



```
for each u ∈ V
      do color[u] ← WHITE
            prev[u] ← NIL
time ← 0
for each u ∈ V
      do if color[u] = WHITE
                  then DFS-VISIT(u)
```

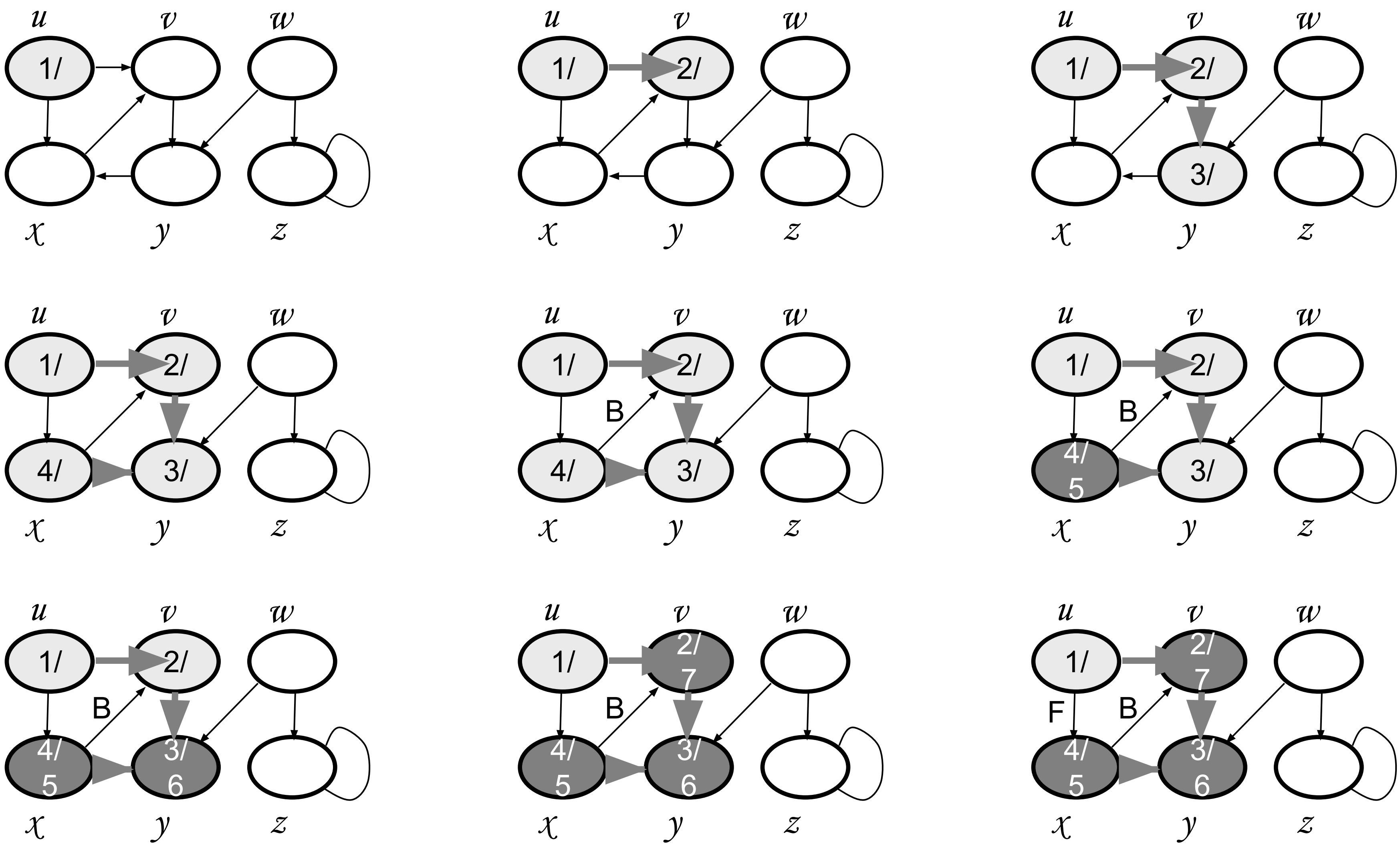
- Every time DFS-VISIT(u) is called, u becomes the root of a new tree in the depth-first forest

DFS-VISIT(u)

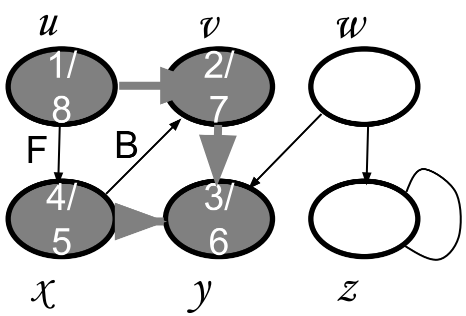
```
color[u] ← GRAY
time ← time+1
d[u] ← time
for each v ∈ Adj[u]
    do if color[v] = WHITE
        then prev[v] ← u
            DFS-VISIT(v)
color[u] ← BLACK
time ← time + 1
f[u] ← time
```

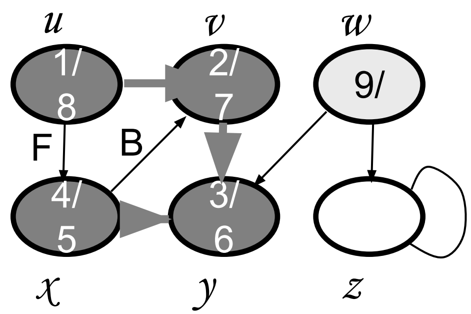


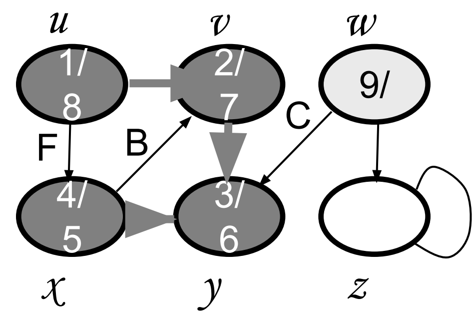
Example

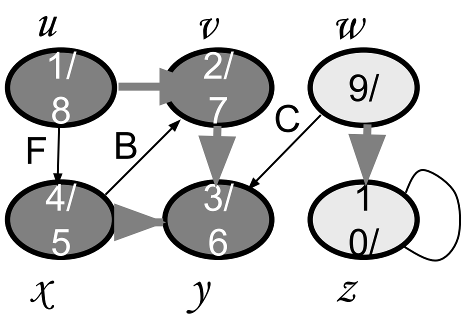


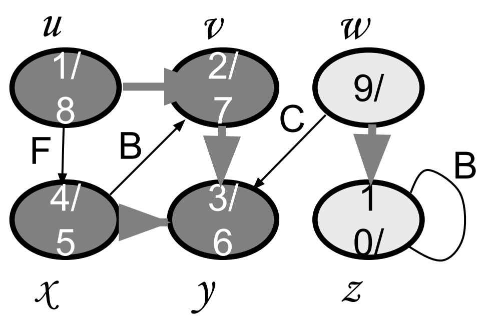
Example (cont.)

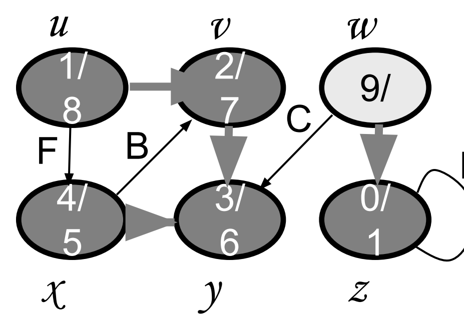


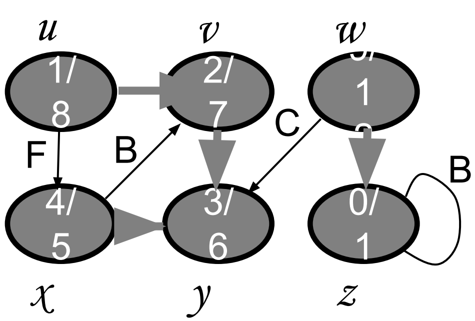










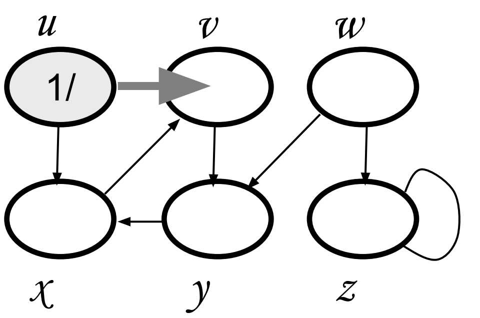


The results of DFS may depend on:

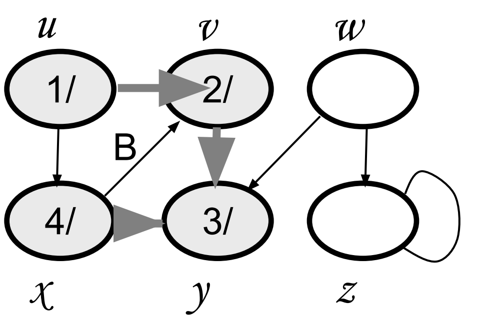
- The order in which nodes are explored in procedure DFS
- The order in which the neighbors of a vertex are visited in DFS-VISIT

Edge Classification

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

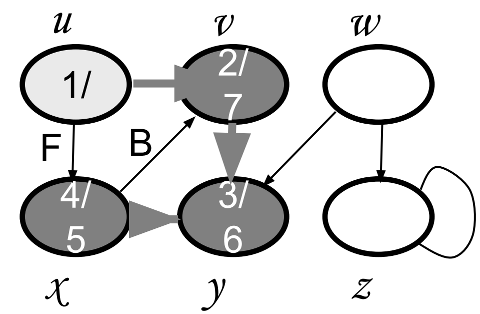


- **Back edge** (reaches a GRAY vertex):
  - (u, v), connecting a vertex u to an ancestor v in a depth first tree
  - Self loops (in directed graphs) are also back edges

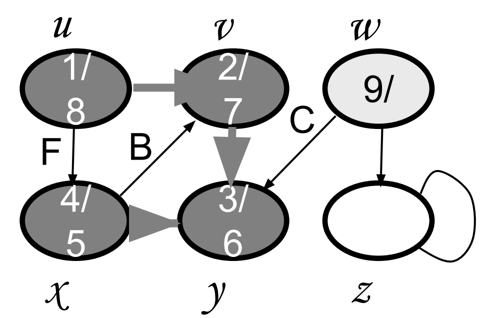


Edge Classification

- **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)

```
for each u ∈ V
        do color[u] ← WHITE
            π[u] ← NIL
time ← 0
for each u ∈ V
        do if color[u] = WHITE
                then DFS-VISIT(u)
```

$\Theta(V)$

$\Theta(V)$ – exclusive of time for DFS-VISIT

Analysis of DFS-VISIT(u)

```
color[u] ← GRAY
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
time ← time + 1
f[u] ← time
```

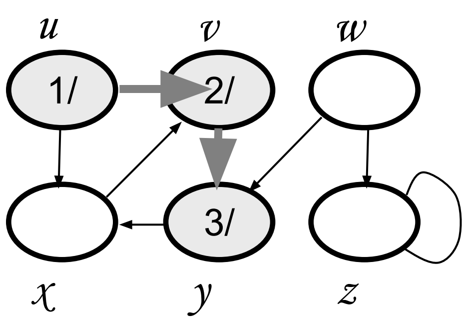
DFS-VISIT is called exactly once for each vertex

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



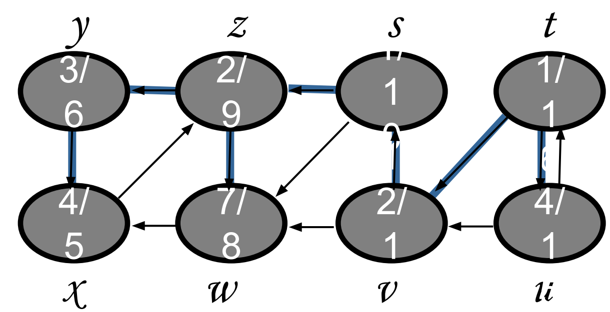
- Vertex v is a descendant of vertex u in the depth first forest ⇔ v is discovered during the time in which u is gray

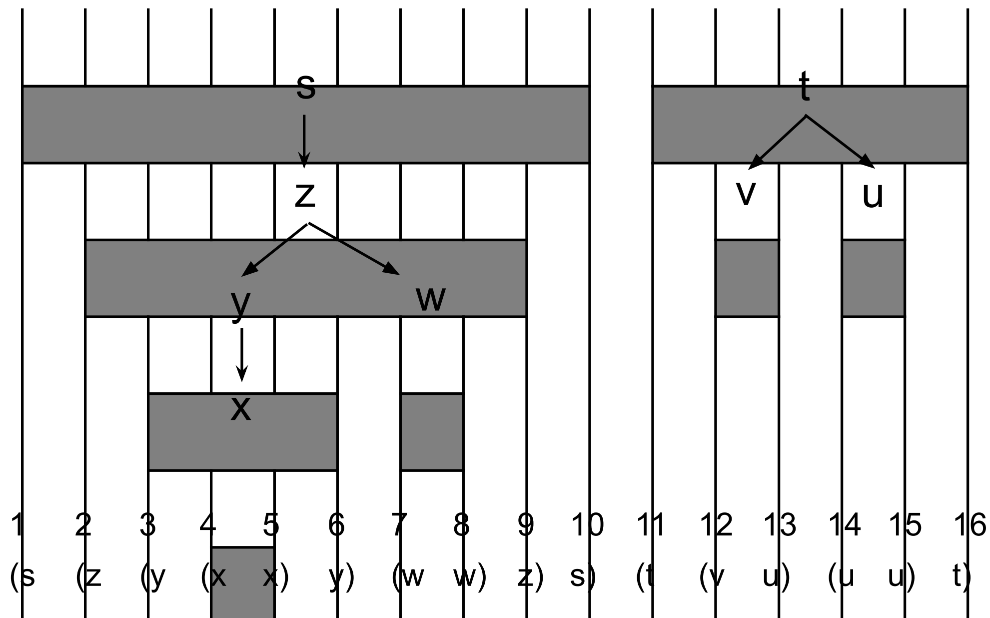
Who is descendent ?

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





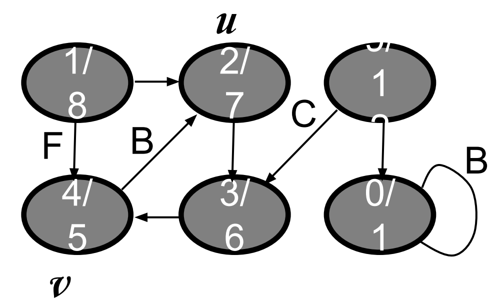
Well-formed expression: parenthesis are properly nested

Other Properties of DFS

Corollary

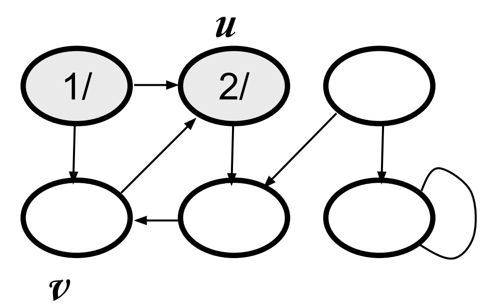
Vertex v is a proper descendant of u

⇔ 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.

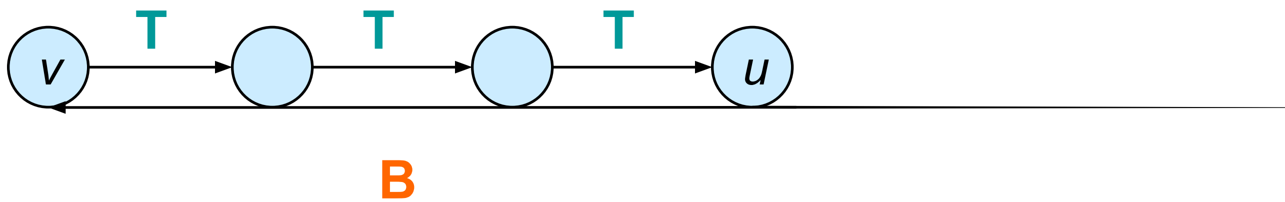


Directed Acyclic Graph

- 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.

mainly given a graph, we are trying to find node ordering

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

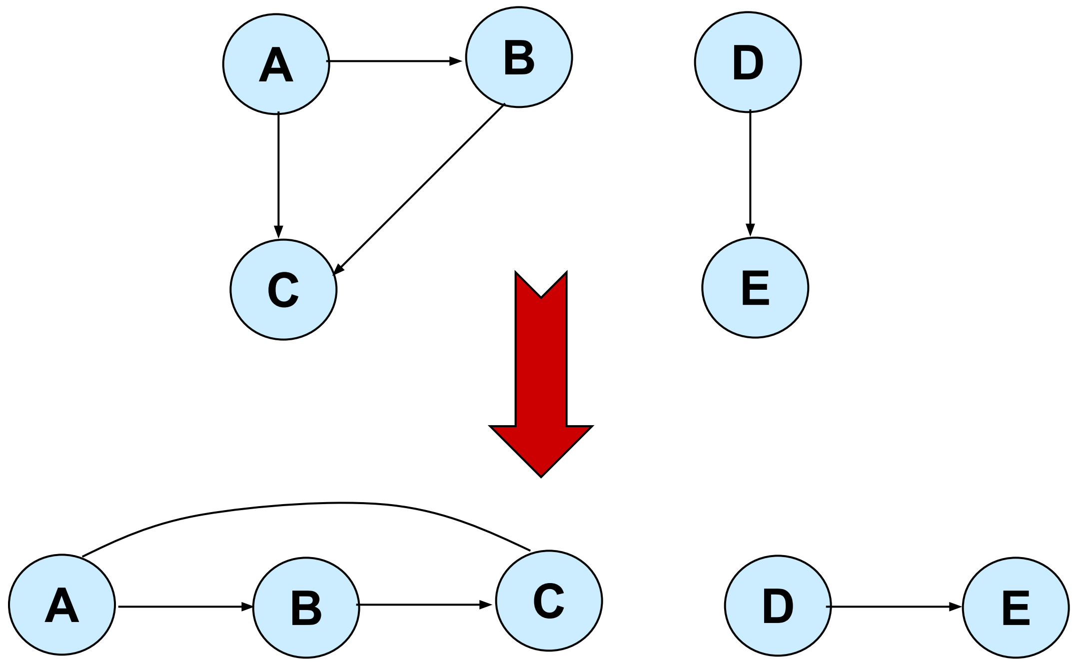
a before b
b before c
a before c

b before c
a before c
What about a and b?

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

Topological Sort

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



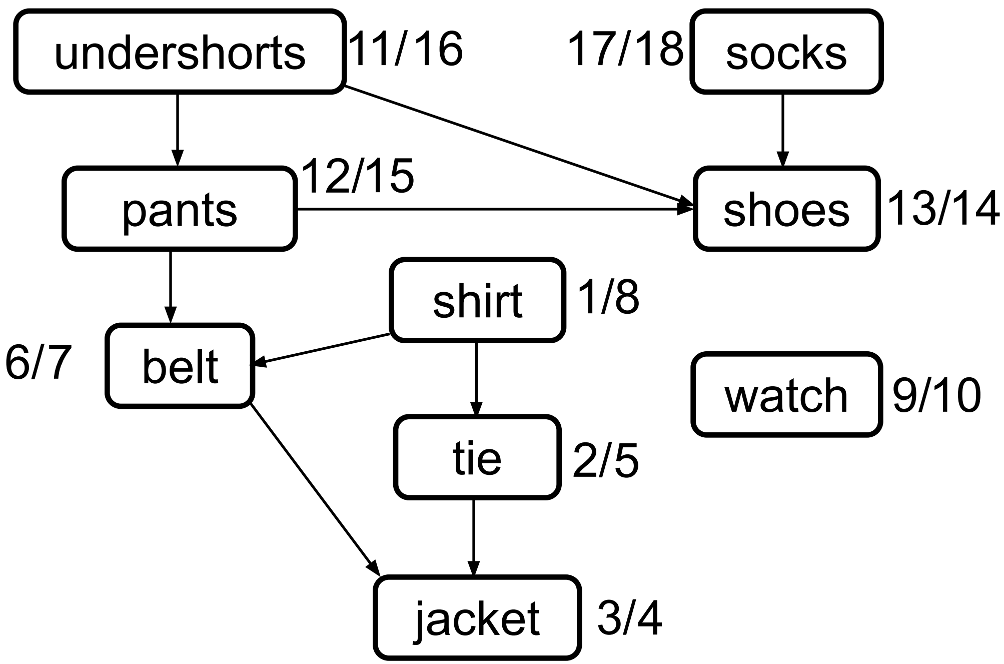
Think of original DAG as a **partial order**.

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

Topological Sort - Application

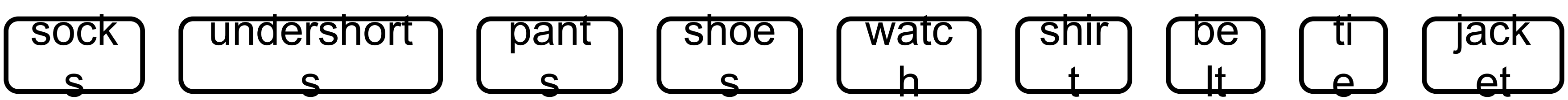
- Application 1
  - in scheduling a sequence of jobs.
  - The jobs are represented by vertices,
  - there is an edge from x to y if job x must be completed before job y can be done
    - (for example, washing machine must finish before we put the clothes to dry). Then, a topological sort gives an order in which to perform the jobs
- Application 2
  - In open credit system, how to take courses (in order) such that, prerequisite 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/vectors of vertices

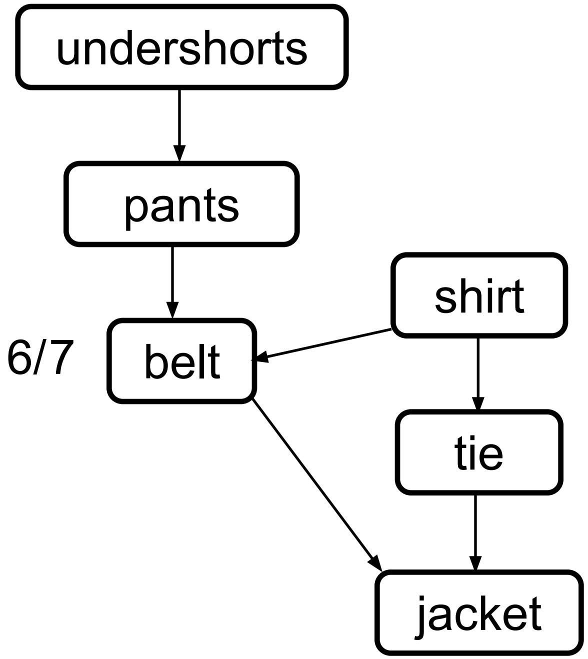


Running time: $\Theta(V + E)$

f[u] > f[v] means, u needs to be done early before v.

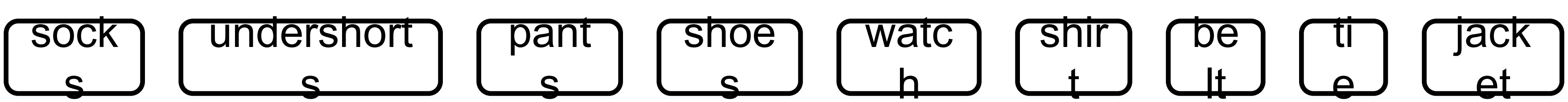
All the pre-dependencies are resolved to do a task

Topological Sort (Fig – Cormen)



Before wearing jacket, here we have two dependencies, wearing belt and tie.
When we simulate the algorithm, we will see that all the dependencies are resolved first,

- belt and tie are worn first before wearing jacket.



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

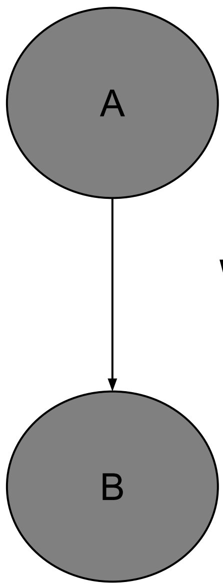
Topological sort using degree

- Let (A->B) denotes that, if we complete A, then we can complete B.
- Let’s make an adjacency list L, which stores this information (L, A: B)
- Now, from L calculate in degree, inD[ ] for each node (inD[A] =0, inD[B] = 1)

Topological sort using degree

- Start with the node u which has in degree 0 => this node has no dependencies, so we can complete it early (Here A).
- Now from this node, see where we can go, reach those nodes v, decrease the in degree of each such node, inD[v]--
- Now continue same process till each node has in degree 0.

Topological sort using degree

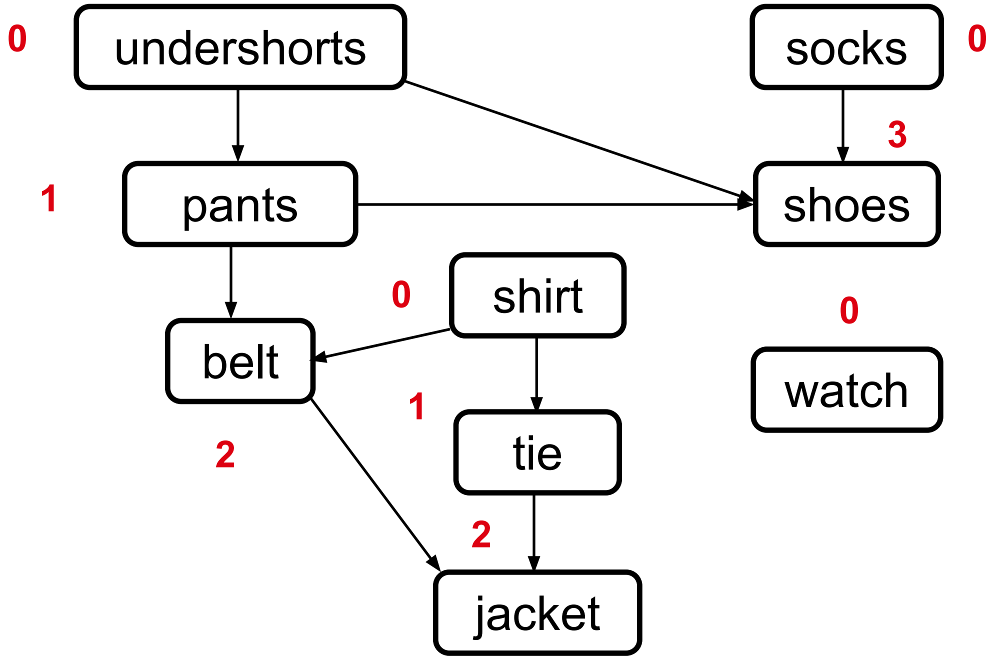


inD[A] = 0

We need to complete A first then B

inD[B] = 1

Topological sort using degree

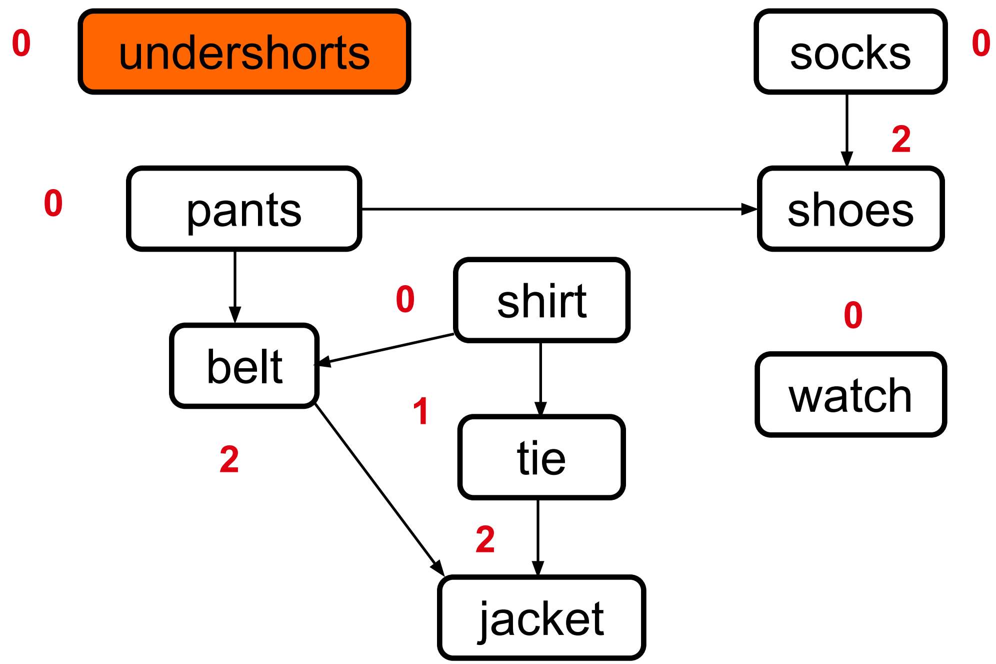


We can start with any of these

undershorts / socks / watch / shirt

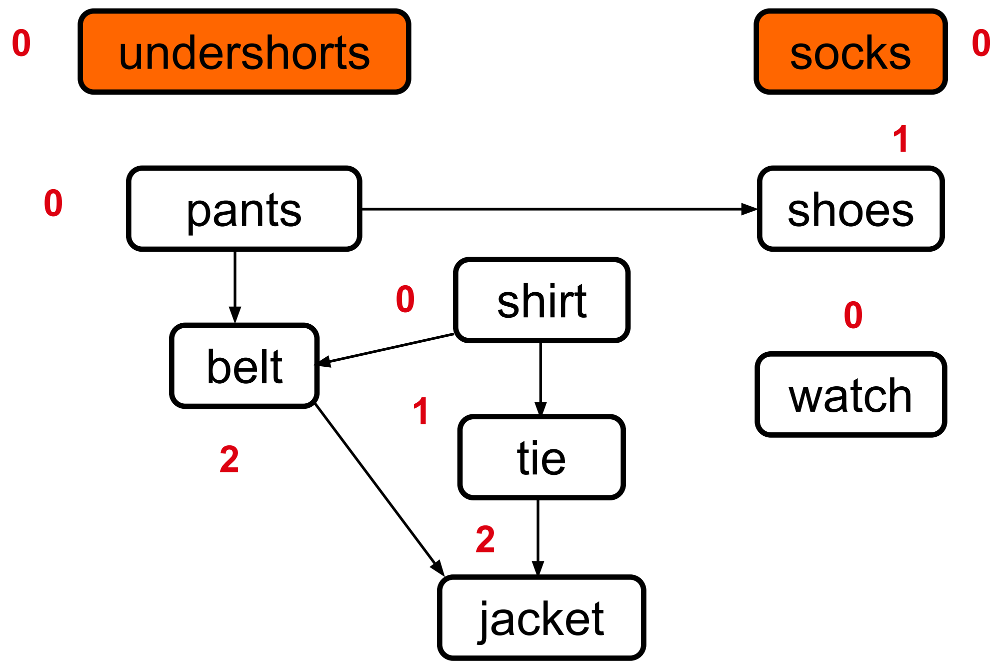
In degree of each node

Topological sort using degree



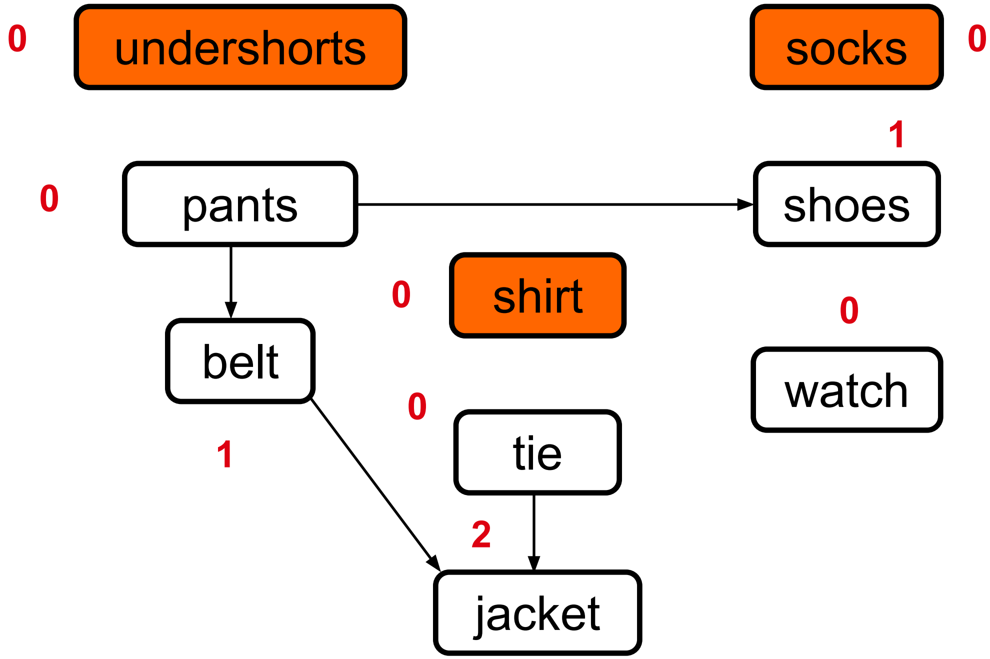
undershorts

Topological sort using degree



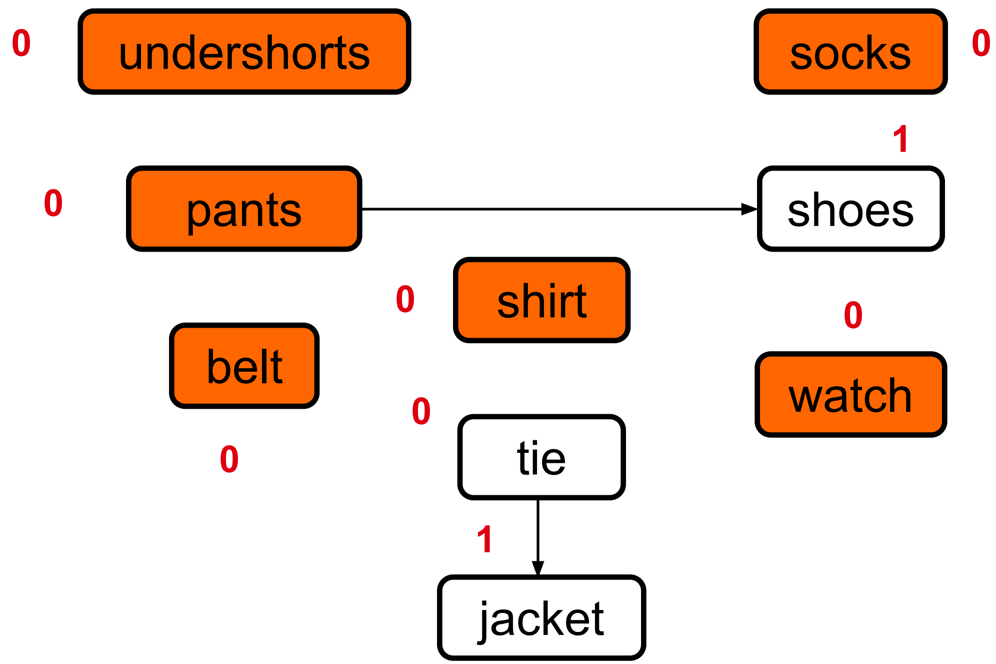
undershorts
socks

Topological sort using degree



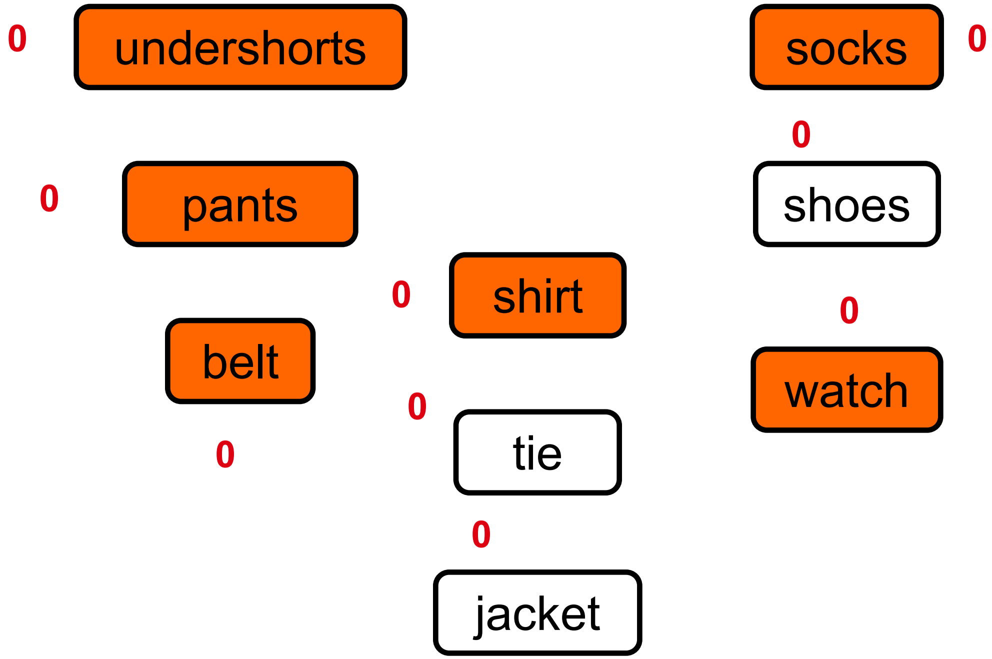
undershorts
socks
shirt

Topological sort using degree



undershorts
socks
shirt
watch
belt

Topological sort using degree



undershorts
socks
shirt
watch
belt
pants
tie
jacket
shoes

Topological sort using degree

- Calculate in degree (O(E)). We can find here, which node has indegree 0. Start with this node.
- In each iteration, some edges will be cut, thus we will visit maximum total number of edges O(E)
- Total complexity: O(E)