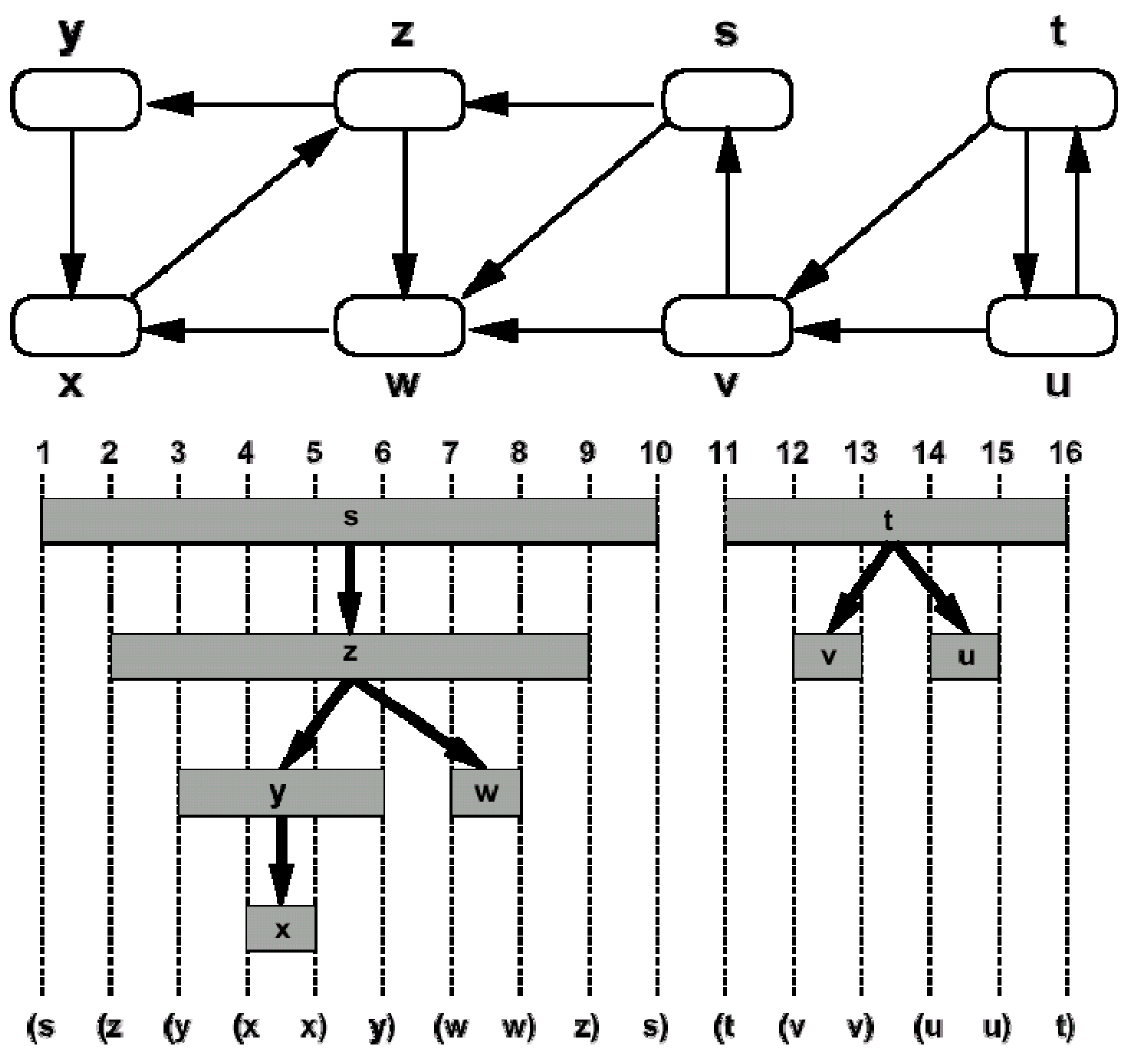
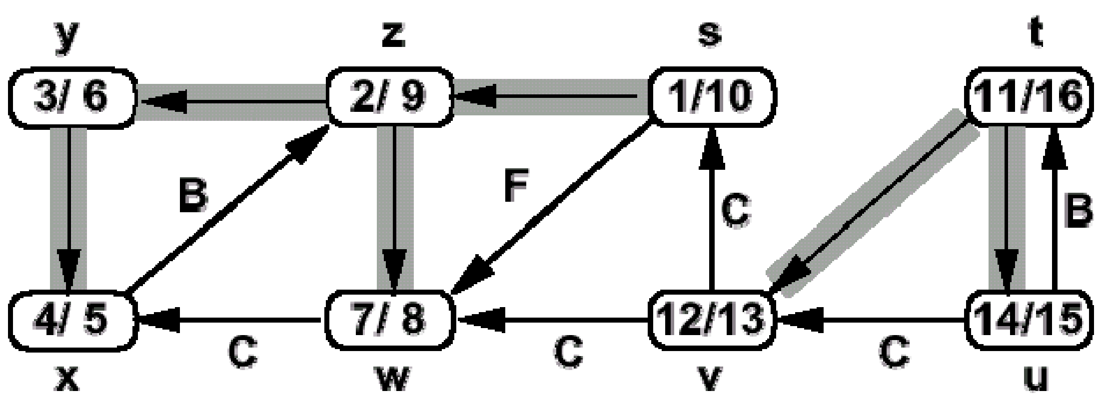
DFS Parenthesis Theorem (2)



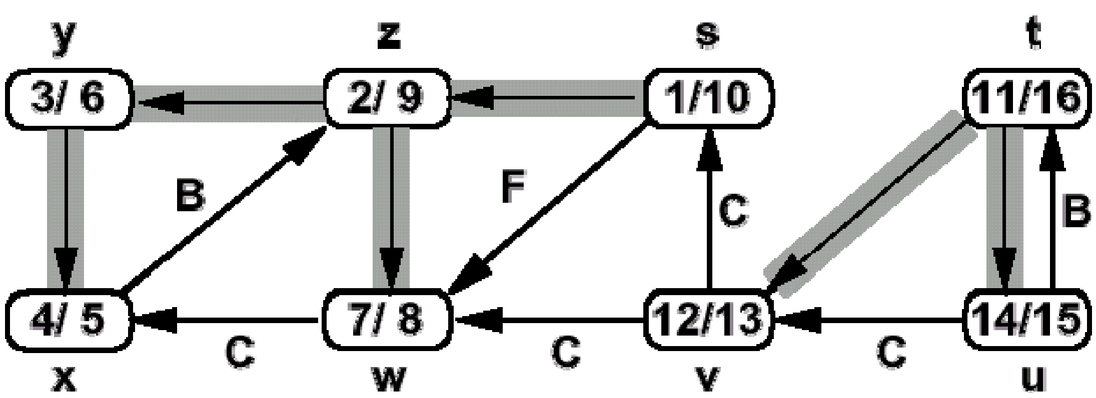
DFS Edge Classification

- Tree edge (gray to white)
  - encounter new vertices (white)
- Back edge (gray to gray)
  - from descendant to ancestor



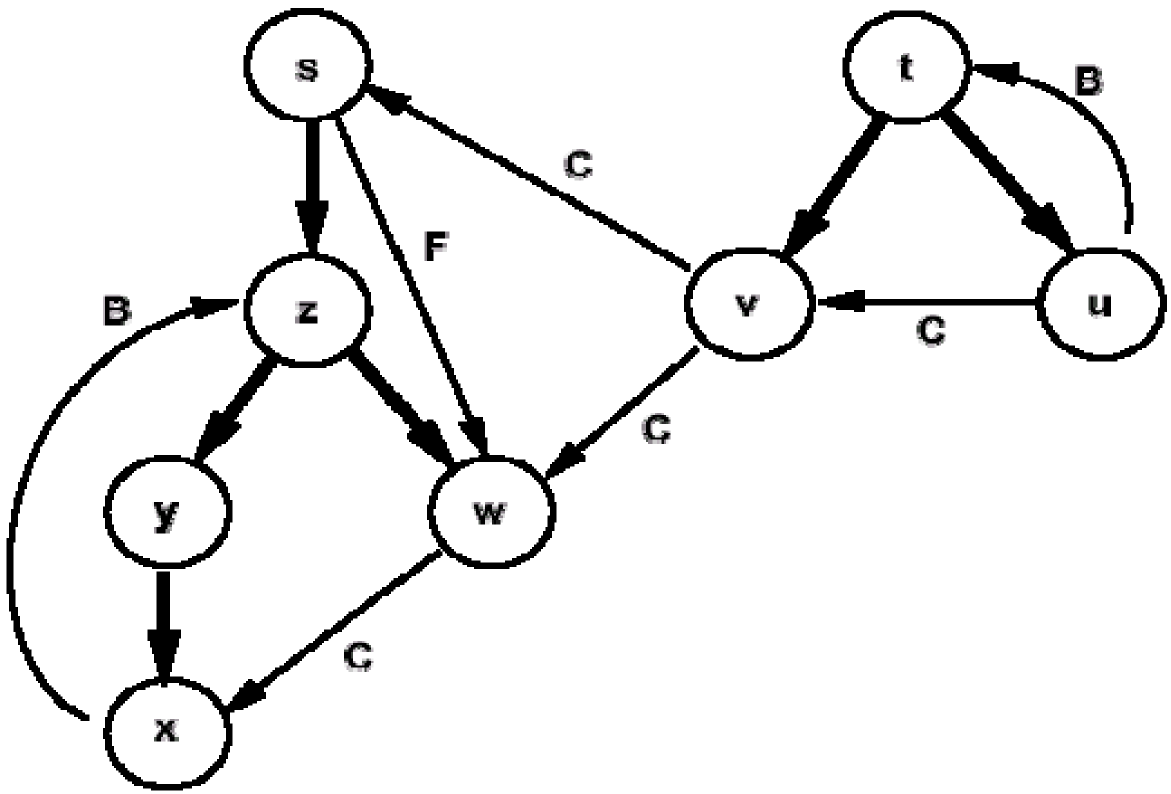
DFS Edge Classification (2)

- Forward edge (gray to black)
  - from ancestor to descendant
- Cross edge (gray to black)
  - remainder – between trees or subtrees



DFS Edge Classification (3)

- Tree and back edges are important
- Most algorithms do not distinguish between forward and cross edges

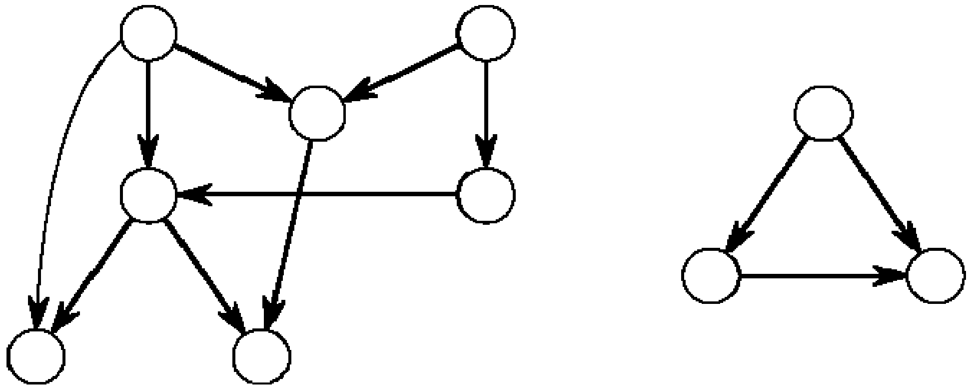


Next:

- Application of DFS: Topological Sort

Directed Acyclic Graphs

- A DAG is a directed graph with no cycles



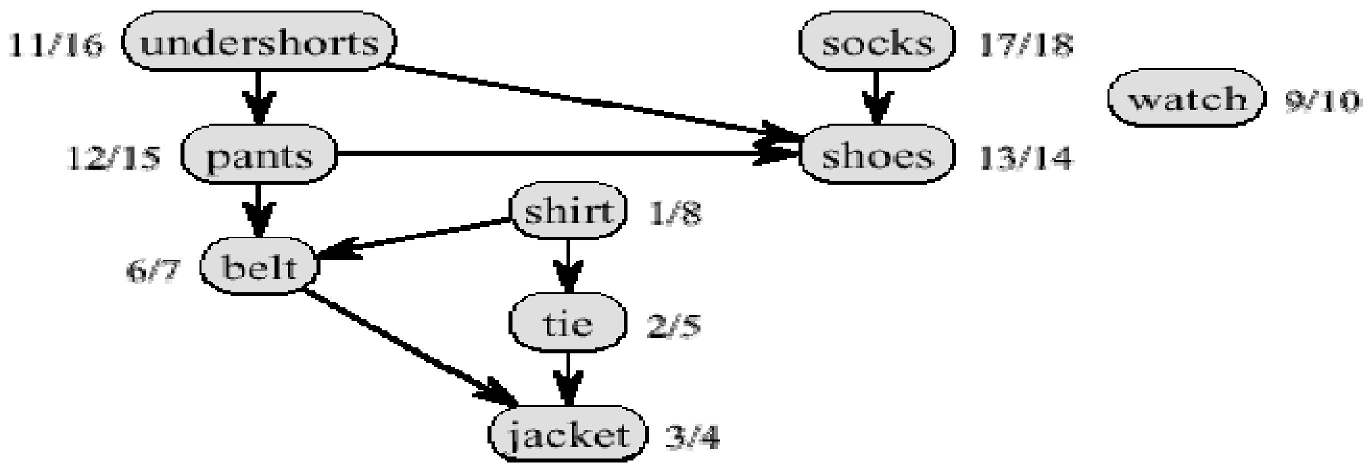
- Often used to indicate precedences among events, i.e., event *a* must happen before *b*
- An example would be a parallel code execution
- Total order can be introduced using **Topological Sorting**

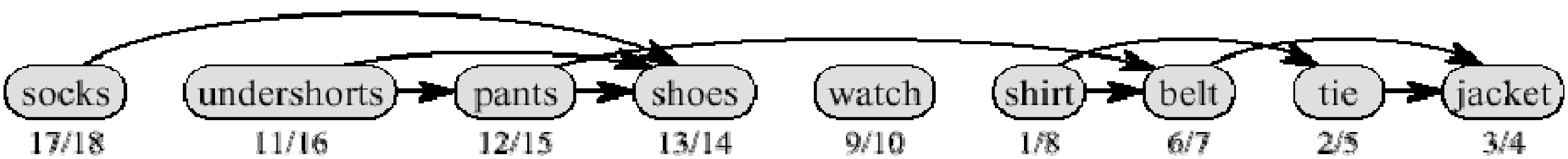
DAG Theorem

- A directed graph *G* is acyclic if and only if a DFS of *G* yields no back edges. Proof:
  - **suppose there is a back edge (*u*,*v*);** *v* is an ancestor of *u* in DFS forest. Thus, there is a path from *v* to *u* in G and (*u*,*v*) completes the cycle
  - **suppose there is a cycle *c*;** let *v* be the first vertex in *c* to be discovered and *u* is a predecessor of *v* in *c*.
    - Upon discovering *v* the whole cycle from *v* to *u* is white
    - We must visit all nodes reachable on this white path before return DFS-Visit(*v*), i.e., vertex *u* becomes a descendant of *v*
    - Thus, (*u*,*v*) is a back edge
- Thus, we can verify a DAG using DFS!

Topological Sort Example

- Precedence relations: an edge from *x* to *y* means one must be done with *x* before one can do *y*
- Intuition: can schedule task only when all of its subtasks have been scheduled





Topological Sort

- Sorting of a directed acyclic graph (DAG)
- A topological sort of a DAG is a linear ordering of all its vertices such that for any edge (u,v) in the DAG, u appears before v in the ordering
- The following algorithm topologically sorts a DAG

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

- The linked lists comprises a total ordering

Topological Sort

- Running time
  - depth-first search: $O(V+E)$ time
  - insert each of the $|V|$ vertices to the front of the linked list: $O(1)$ per insertion
- Thus the total running time is $O(V+E)$

Topological Sort Correctness

- Claim: for a DAG, an edge $(u,v) \in E \Rightarrow f[u] > f[v]$
- When (u,v) explored, u is gray. We can distinguish three cases
  - v = gray
    $\Rightarrow$ (u,v) = back edge (cycle, contradiction)
  - v = white
    $\Rightarrow$ v becomes descendant of u
    $\Rightarrow$ v will be finished before u
    $\Rightarrow$ $f[v] < f[u]$
  - v = black
    $\Rightarrow$ v is already finished
    $\Rightarrow$ $f[v] < f[u]$
- The definition of topological sort is satisfied