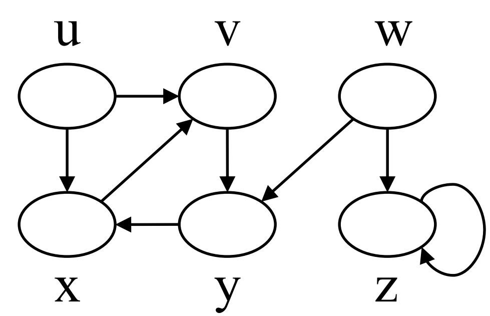
Depth-First Search in Directed Graphs

DFS Algorithm

Initialize all vertices UNMARKED.

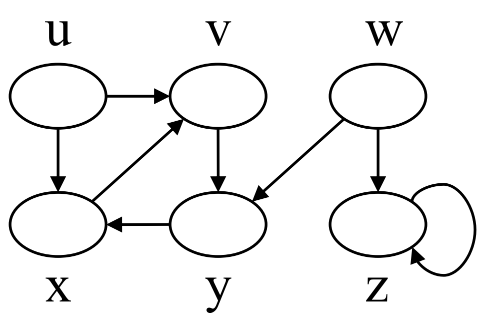
```
For u = 1 … n
    if u is UNMARKED
        DFS(u)

DFS(u) {

    MARK u

    for all out-neighbours v of u
       if v is UNMARKED
            DFS(v);

}
```



DFS Algorithm: DFS Tree

Initialize all vertices UNMARKED.

```
For u = 1 … n
    if u is UNMARKED
        DFS(u)

DFS(u) {

    MARK u

    for all out-neighbours v of u
       if v is UNMARKED
              DFS(v); parent(v)=u;


}
```



DFS Algorithm: stack view

Initialize all vertices UNMARKED.

For u = 1 … n
    if u is UNMARKED
        DFS(u)

DFS(u) { // u is pushed on stack

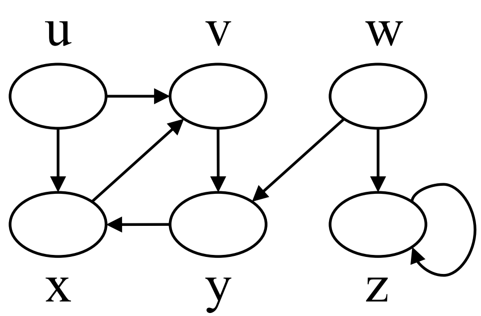
    MARK u

    for all out-neighbours v of u
      if v is UNMARKED
            DFS(v); parent(v)=u;

      // u is popped from the stack

}

Recursive calls can be simulated by a stack



DFS Algorithm: stack view

Initialize all vertices UNMARKED.

For u = 1 … n
    if u is UNMARKED
        DFS(u)

DFS(u) { // u is pushed on stack

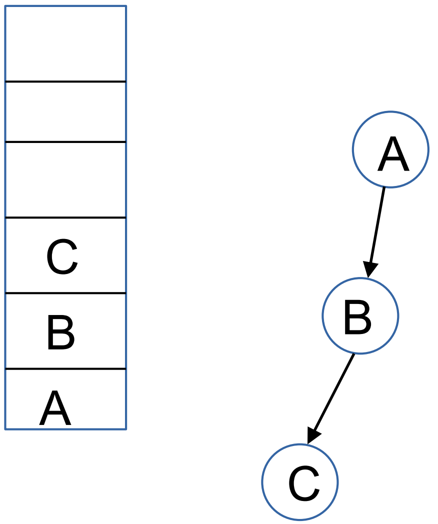
    MARK u

    for all out-neighbours v of u
      if v is UNMARKED
            DFS(v); parent(v)=u;

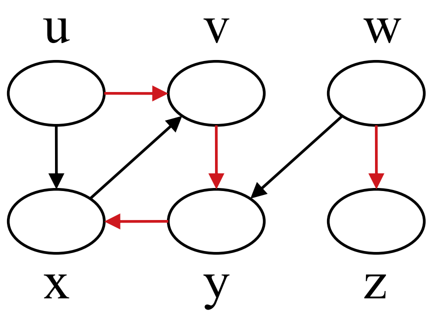
      // u is popped from the stack

}

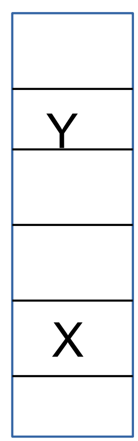
Recursive calls can be simulated by a stack

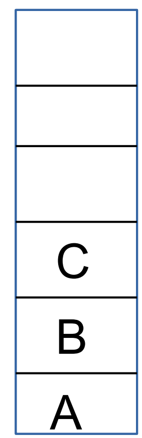


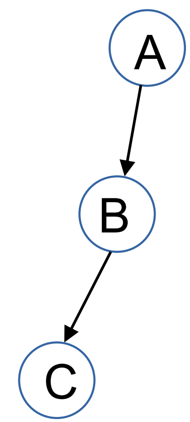
DFS Algorithm: stack view



A node X is an ancestor of Y in The DFS tree iff the stack looks as follows at some point of time:





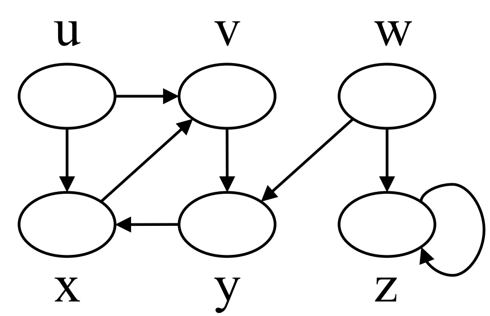


DFS Algorithm: timers

Initialize all vertices UNMARKED.

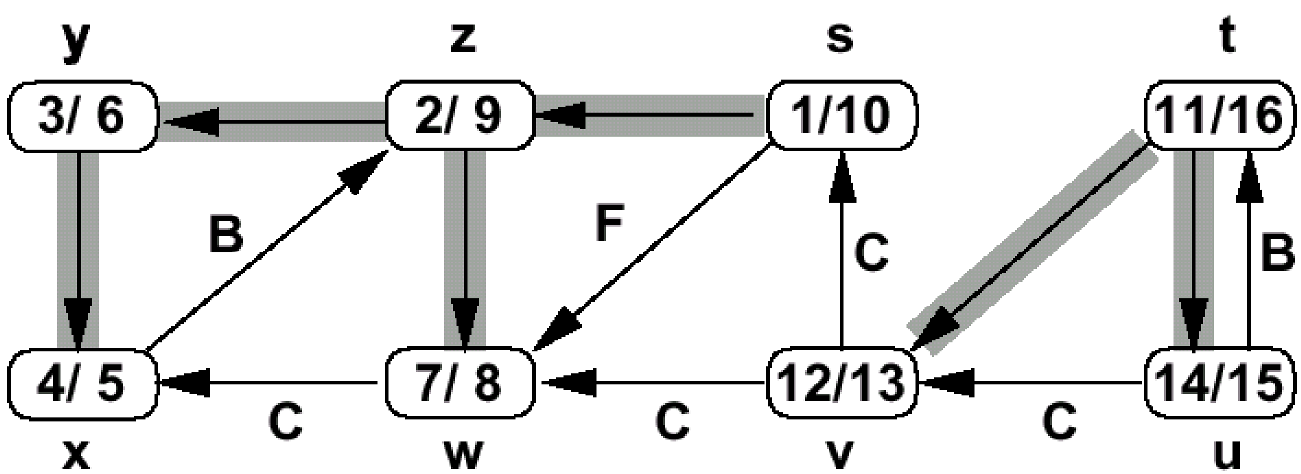
```
For u = 1 … n
    if u is UNMARKED
        DFS(u)

DFS(u) { // u is pushed on stack
    d[u] = timer++ (discovery time)
    MARK u

    for all out-neighbours v of u
      if v is UNMARKED
            DFS(v); parent(v)=u;

      // u is popped from the stack
    f[u] = timer++ (finish time)
}
```

Maintain a timer

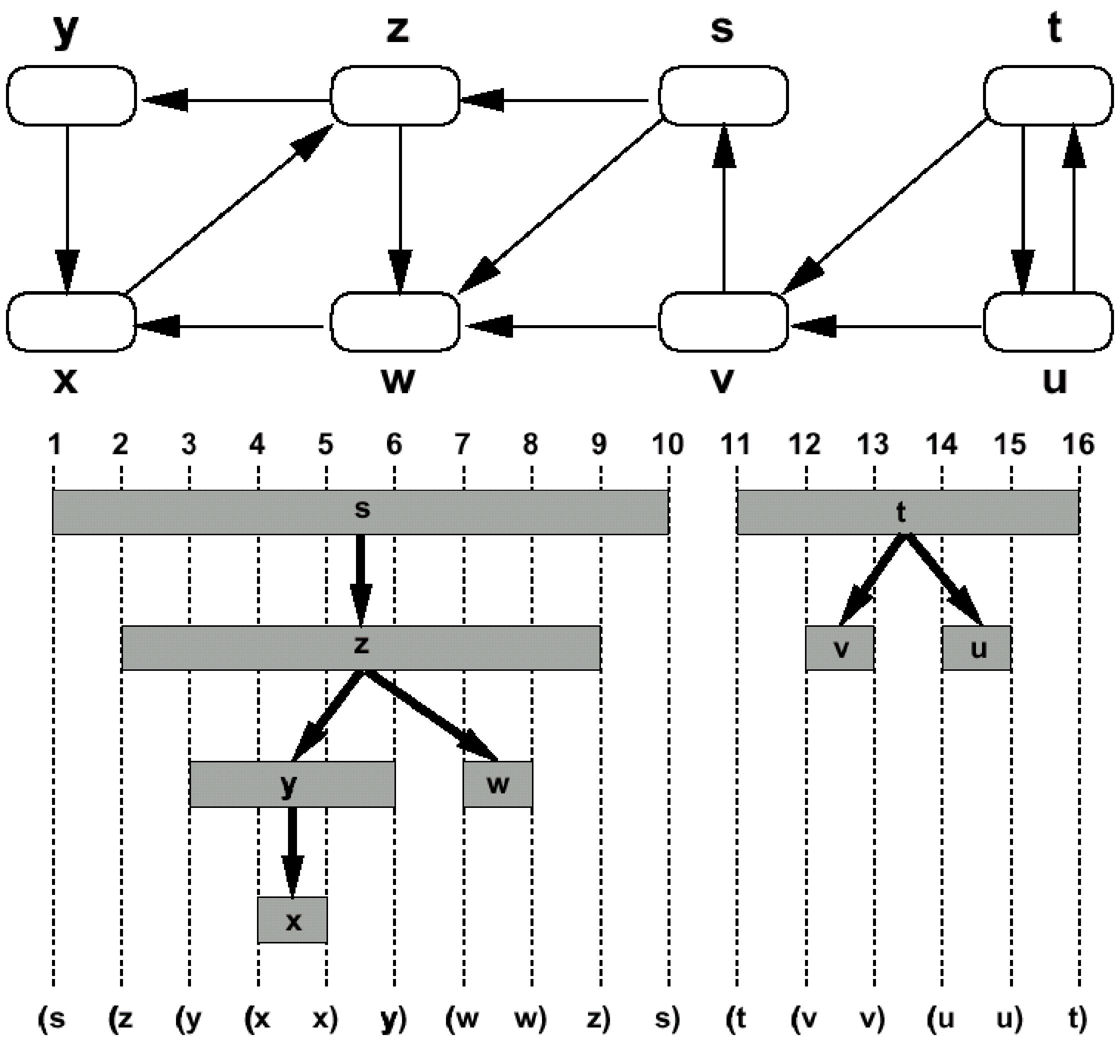


DFS Timestamping

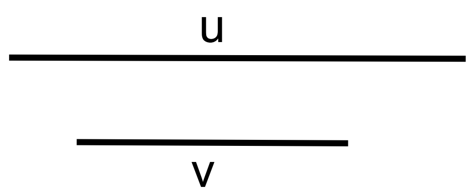
Discovery and finish times have parenthesis structure



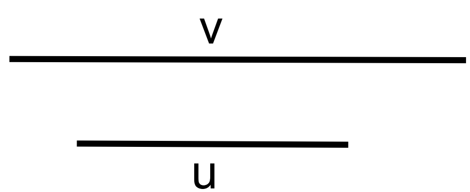
DFS Parenthesis Theorem



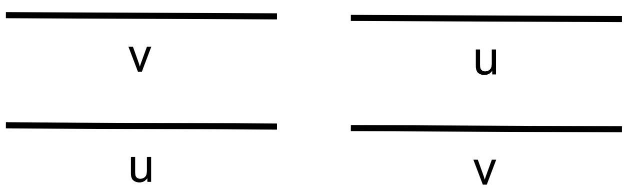
DFS Edge Classification



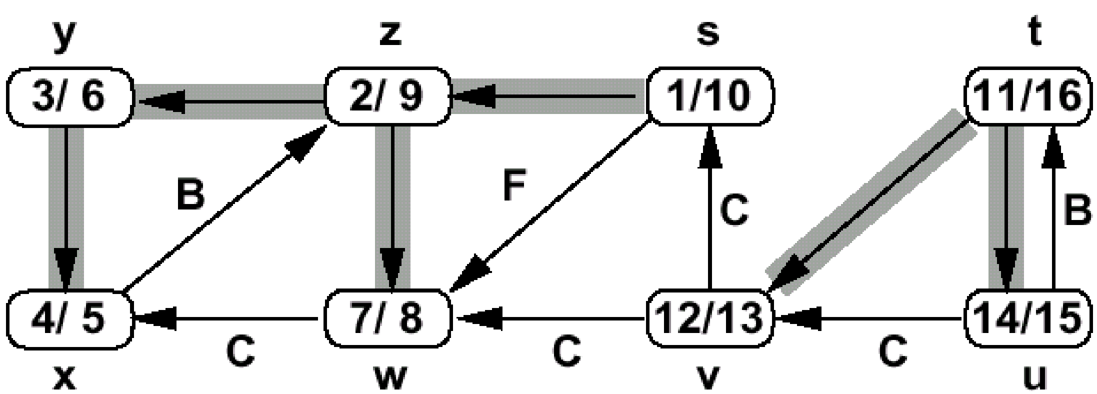
u is ancestor of v
(Forward or tree edge)



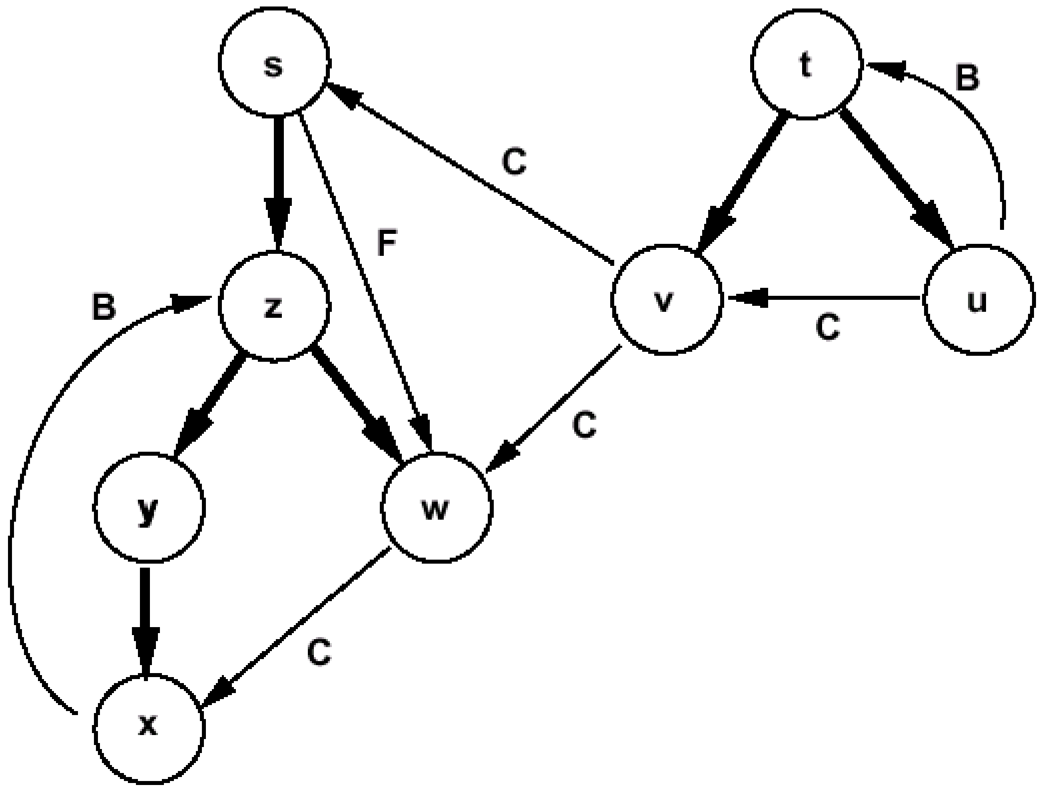
v is ancestor of u
(Back edge)



Cross edge

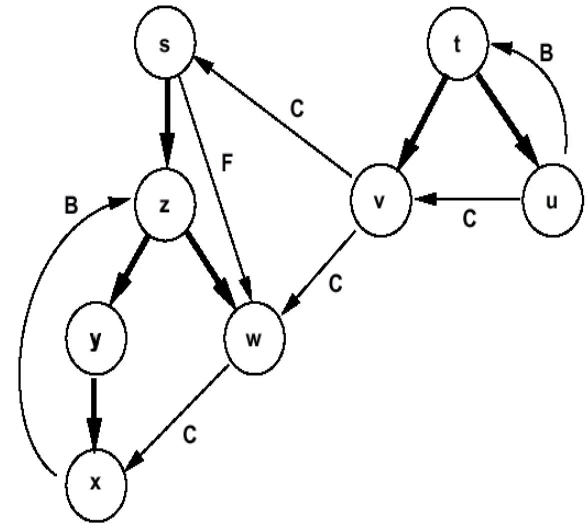


DFS Edge Classification



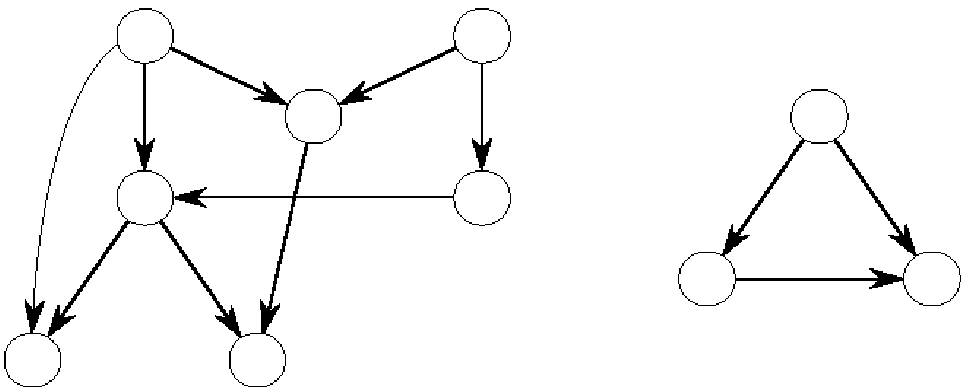
Application: Cycle Detection

There is a cycle iff there is a back edge



Directed Acyclic Graphs

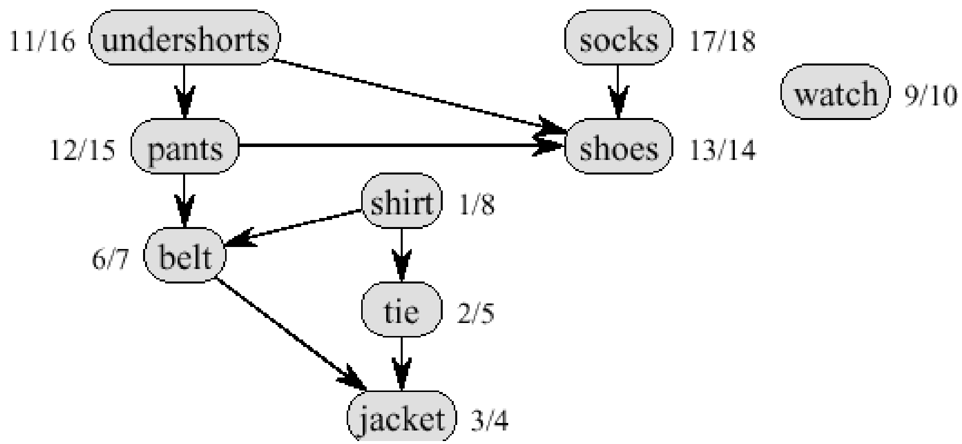
- A DAG is a directed graph with no cycles

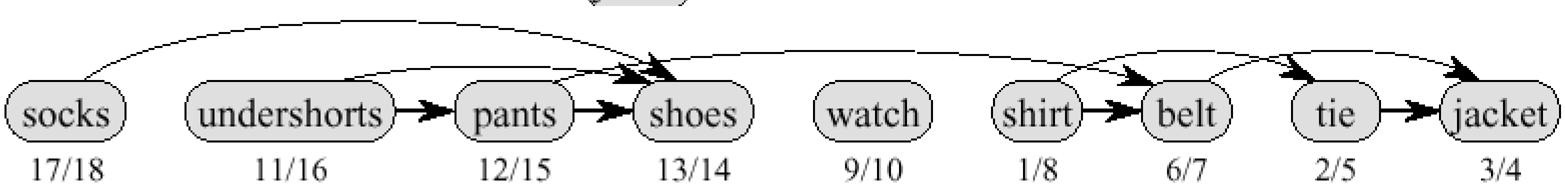


- Often used to indicate precedences among events, i.e., event *a* must happen before *b*
- An example would be a parallel code execution
- Total order can be introduced using **Topological Sorting**

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 (decreasing order of f[v] values)

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