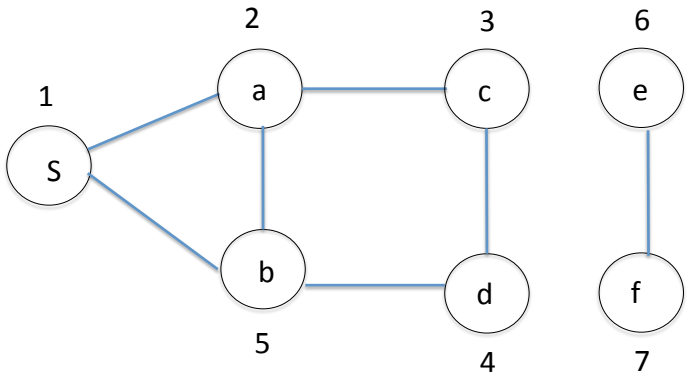
Graphs part 2

Depth First Search (DFS)

Search deeply first… Explores edges out of most recently discovered vertex so long as there are still unexplored edges leaving it… and then backtracks (like searching in a maze). Run time as in BFS will be linear in number of vertices and edges O(n+m)

Applications: topological sort in directed acyclic graphs (sequences of events where one comes before another; later); strongly connected components (later).

Simple example of DFS ordering showing main idea and distinguishing from bfs:



Note the "forest" (not just a single tree).

One way to implement: Stack instead of queue. Last in first out. But often implemented recursively.

Marking nodes in algorithm: Colors are the same:

White: Vertices are initially white;

Gray: First time vertex encountered, make gray

Black: Only when all its adjacent neighbors discovered, turn to black.

We also mark a node with a number "time stamp" both when it is first discovered (turns gray) and when we have gone through all its neighbors (black). The discovery time is u.d and the finish time is u.f (u.f > u.d).

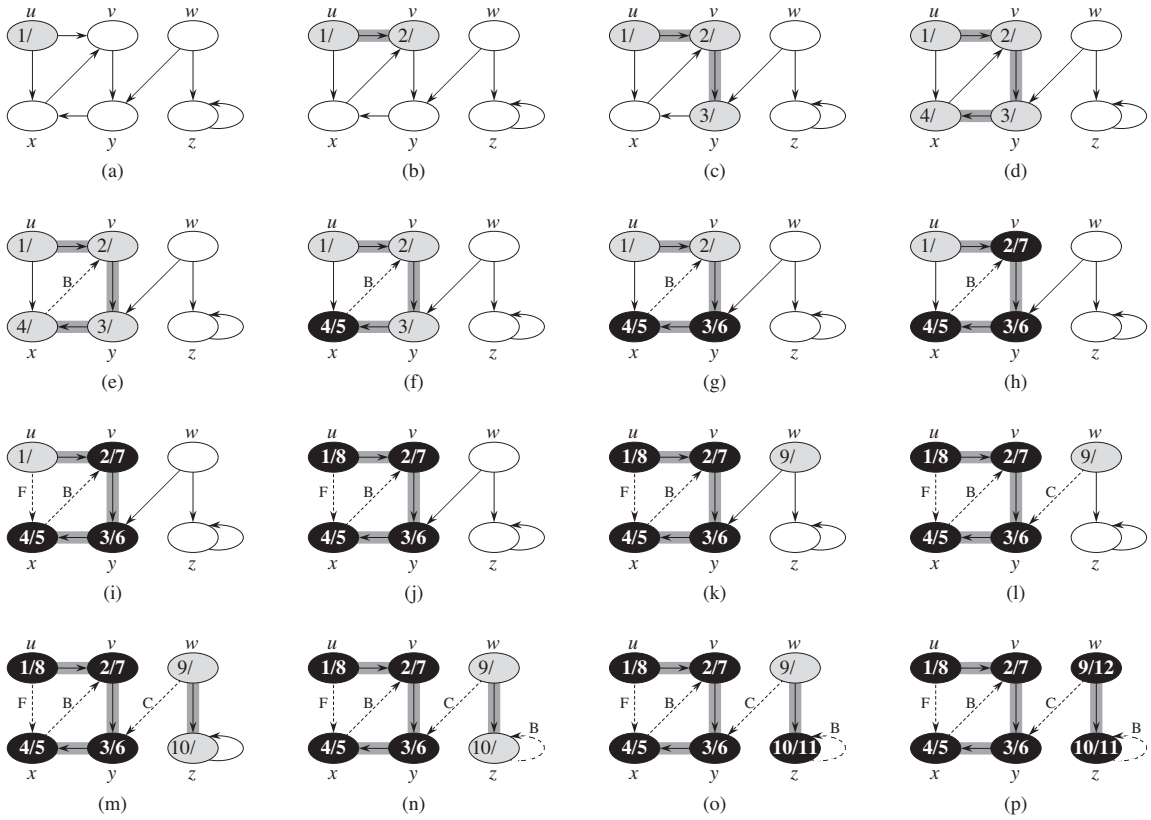
Main algorithm with recursion: We'll have a function DFS that initializes, and then calls DFS-Visit, which is a recursive function and does the depth first search.

```
DFS(G)    // G is a graph
- initialize all vertices to white
- set all time stamps to 0
- for each vertex u in G.V
    if u.color == white
        DFS-Visit(G,u)
```

Recursion:

```
DFS-Visit(G,u)
1. time = time + 1       // white vertex u has just been discovered
2. u.d = time             // start time
3. u.color = gray        // we've discovered u
4. For each v in adjacency list of u
5.        if v.color == white
6.             v.parent = u
7.             DFS-Visit(G, v)
8. u.color = black      // we finished going through all of u's neighbors
9. time = time + 1
10. u.f = time           // finish time
```

Example of DFS:



Run time DFS: O(n + m) with n number vertices and m number edges. Procedure DFS-Visit is called exactly once for each vertex (only if white and then changed). During the DFS-Visit all adjacent nodes of each vertex are used, so overall includes all edges.

Some properties DFS:

1. Every vertex v that gets discovered between start of u and end of u, is a child of u in the DFS tree. The v will also finish before u does (see path from u to v to y to x).

This can be seen in recursion: DFS-Visit(u) called first; and then calls DFS-visit(v), which finishes first (last in first out, as in a stack)

2. There is never an edge from a black to a white node (because if a node is black, we already visited all its neighbors).

3. White path theorem: vertex v is a descendant of vertex u if and only if when the search discovers u, there is a path from u to v consisting entirely of white vertices.

(example, look at panel a in example, nodes u and y, when u is discovered the path to y is all white). Proof: induction (we won't show).

4. Disjoint trees, contained trees, and parenthesis theorem: Consider vertices u and v. Either:

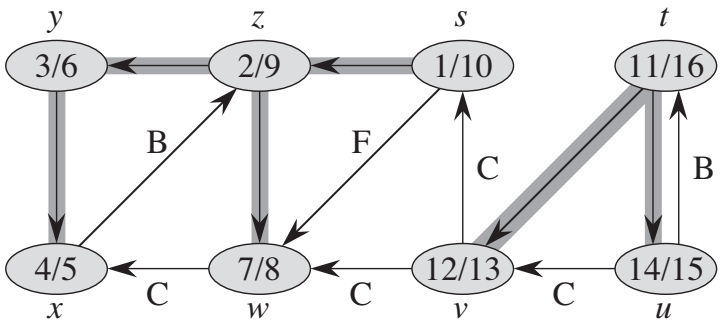
a. [u.d u.f] and [v.d v.f] are entirely disjoint. Neither u or v is a descendant of the other.

b. Or [u.d u.f] is contained entirely within [v.d v.f], and u is a descendant of v (which means u was discovered after v)

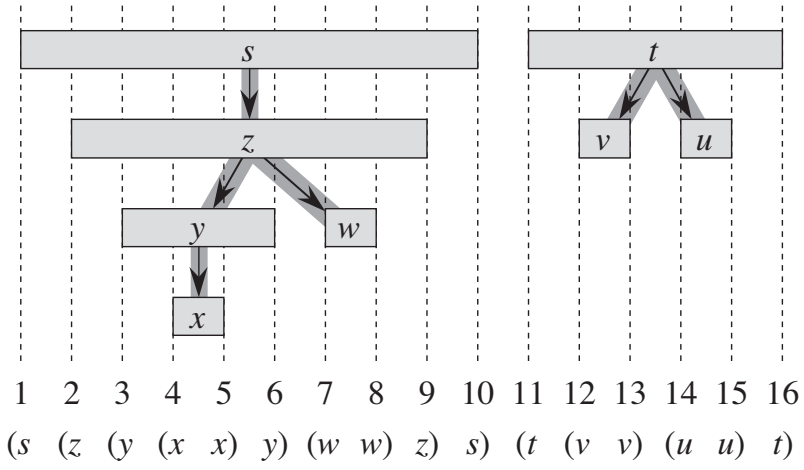
c. Or vice versa; [v.d v.f] is contained entirely within [u.d u.f], and v is a descendant of u (which means v was discovered after u)

We represent start time of u with "(u", and end time with "u)" …

(a)



(b)



5. Classification of edges: edge (u,v).

We did not discuss this in class. I am keeping it in the notes for completeness, but not required.

Tree edges: v was discovered from u, during the procedure. If this is the case, then when we were at u and checked out its adjacency neighbor v, v was white.

Back edges: connects u to ancestor v. This happens when we are in u and checking adjacency neighbor v, but v is already gray (so v is like a grandfather). (This also includes self-loops.)

Forward edges: connects u to a descendant v (but not part of the tree; not direct child that was discovered). This happens during the procedure if we are in u and check out adjacency neighbor v, and v is black.

Cross edges: everything else. Can be between trees or in the same tree. Here also v is black.

Note: Some ambiguity since in undirected graph, edge between u and v could also mean edge between v and u: can classify based on which is encountered first in procedure, or which is first in list of edge types.

Note: in undirected graph, every edge either a tree edge or back edge.