Depth First Search

Suprakash Datta

March 18, 2011

Graph algorithms - continued

Last week: All-pairs shortest paths.

This week: Depth-first search

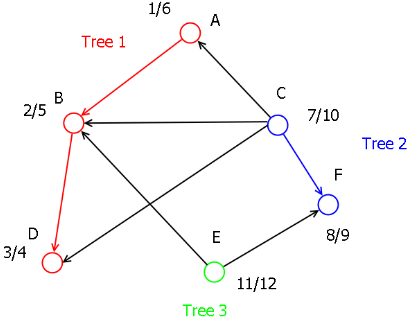
- Similar to breadth-first search
- Also used to explore graphs
- Can often use either; sometimes one is more useful than the other
- Same running time $\Theta(V + E)$, very similar implementation

The Depth-First Search algorithm

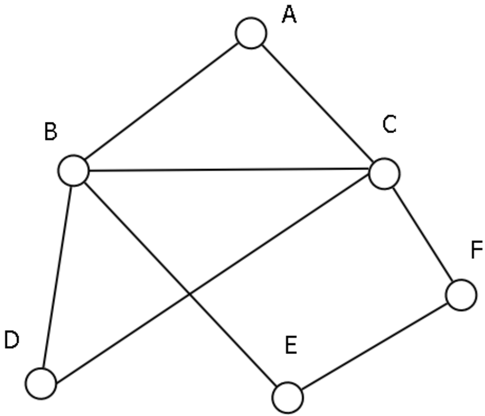
1. Start from some node s.
2. Instead of exploring all children on s before moving on (as in BFS), explore a unexplored child t of s, then to an unexplored child w of t and so on.
3. When no unexplored children exist, backtrack.
4. Initially all nodes are black, and time is zero. Thereafter time increments by 1 at each step.
5. Each node gets 2 timestamps one for when they enter the queue (turn red) and one for when they leave it (turn blue).
6. Edges can be classified by DFS – e.g. back edges go from red nodes to red nodes.
7. The above steps create a DFS tree.

DFS notes

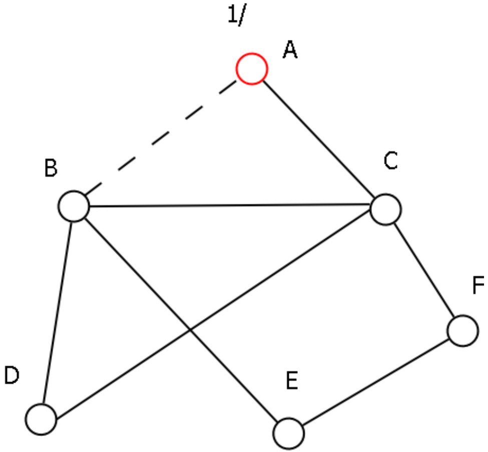
1. Only back edges today – back edges go from red nodes to red nodes, and indicate presence of a cycle
2. Will not use start times today
3. Might need mutiple DFS trees to completely a directed graph.



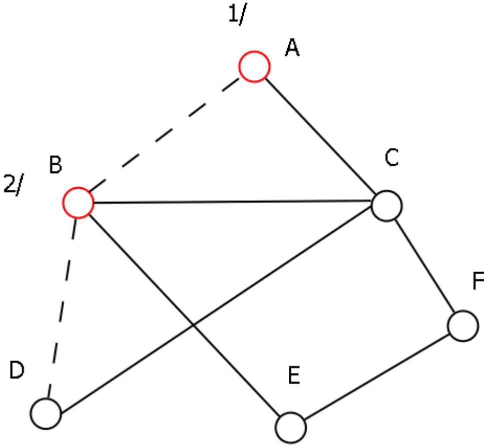
An example graph



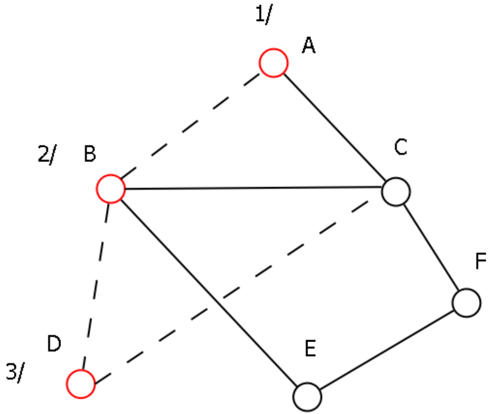
An example graph



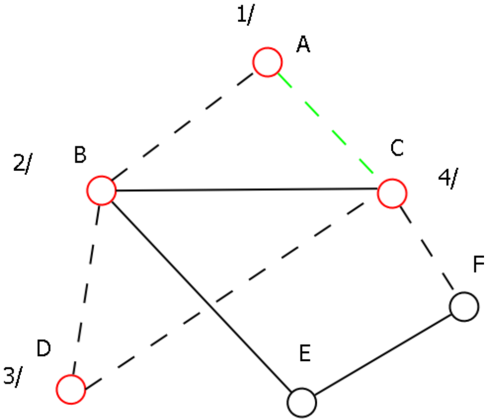
An example graph



An example graph

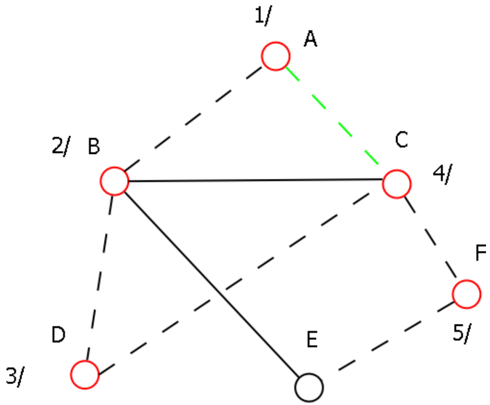


An example graph

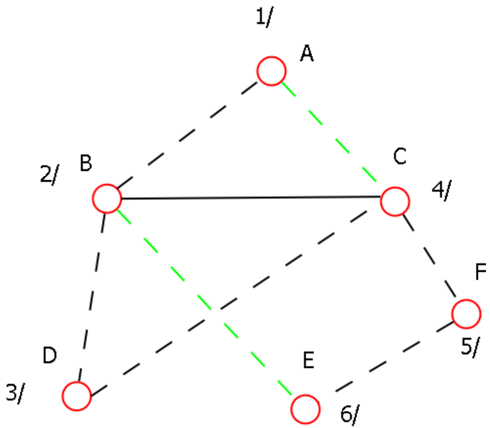


Back edge found (graph is cyclic)

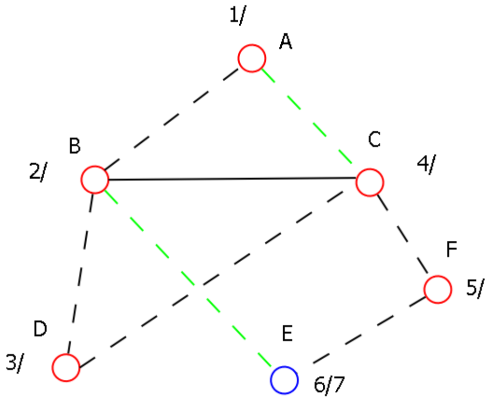
An example graph



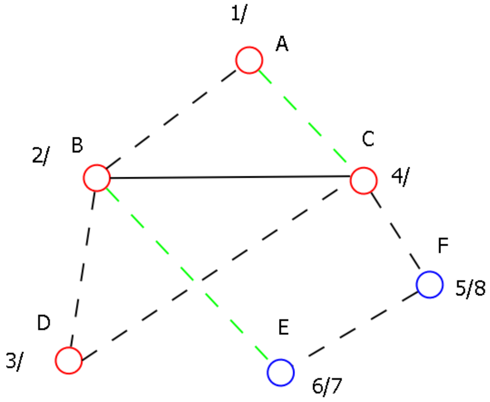
An example graph



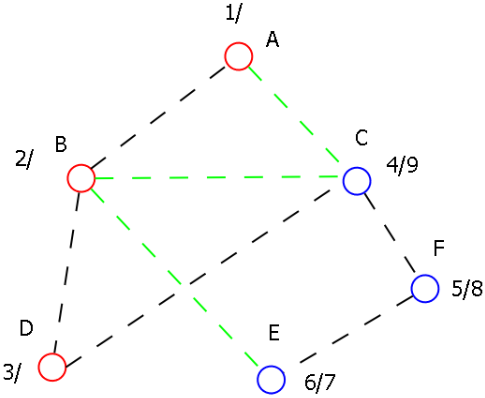
An example graph



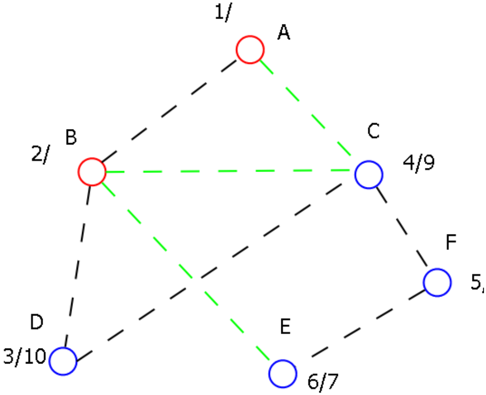
An example graph



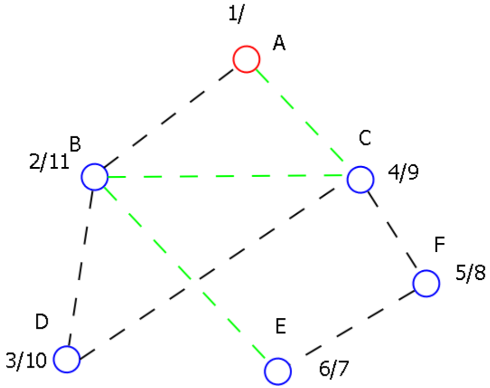
An example graph



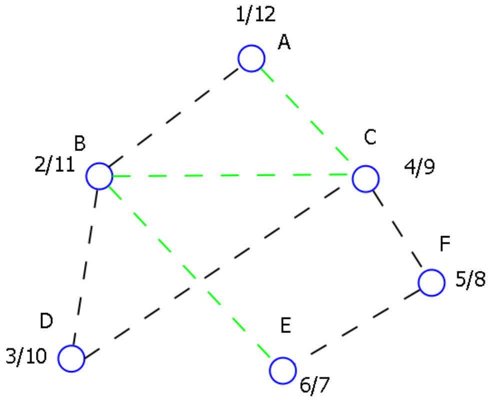
An example graph



An example graph



An example graph

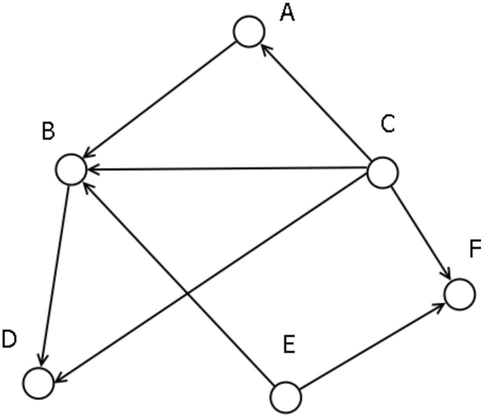


Java implementation

```
//Do a DFS starting from node s, 1 <= s <= n.
//list[i] - adjacency list of node i
//visited[], finish[]: color, finish times of nodes
Stack<Integer> stack = new Stack<Integer>();
boolean[] visited = new boolean[n + 1];
stack.push(s); // make s red
while (!stack.isEmpty())
{
   int u = stack.pop(); // make u blue
   finish[u] = ++time;
   if (!visited[u]) // u is black
   {
       visited[u] = true;
       // make black neighbours of u red
       stack.addAll(list[u]);
   }
}
```

Directed Acyclic Graphs

Directed graph with no directed cycles



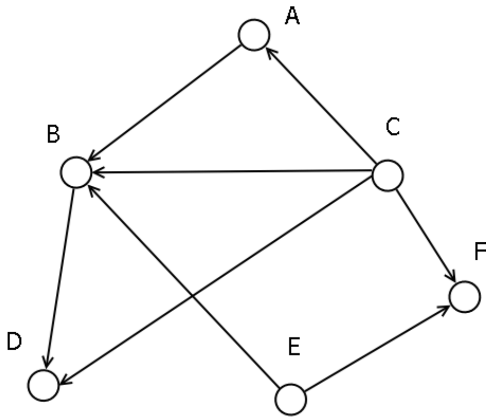
Topological Sort

Fact: A directed acyclic graph can be sorted topologically – i.e. nodes can be numbered so that all edges go from lower numbered nodes to higher number nodes.

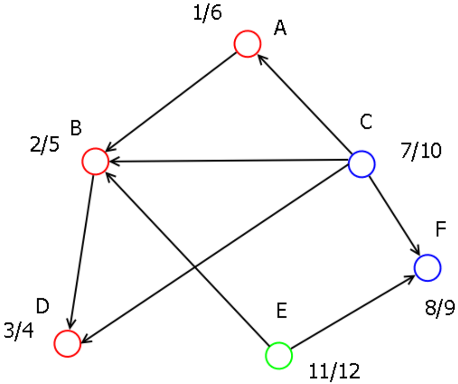
Fact: Nodes sorted by decreasing finish times of DFS are in topologically sorted order

Fact: DFS detects cycles (presence of back edges).

An example graph



An example graph



Topologically sorted order

E - C - F - A - B - D