CS109B Notes for Lecture 4/10/95

Depth-First Search

- A method of exploring a directed graph and numbering the nodes.
- Many useful properties — stay tuned.

The DFS Algorithm

1. "unmark" all nodes.
2. Pick a start node v_0 and execute the recursive function $dfs(v_0)$.
3. $dfs(u)$ = for each successor v of u that is unmarked:

   a) Mark v.

   b) Call $dfs(v)$.

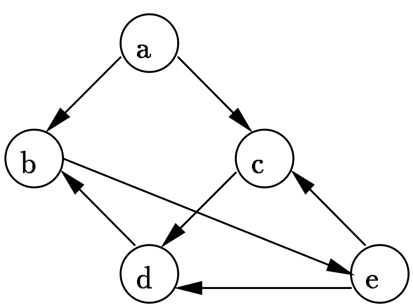
   (Do nothing if v was already marked.)

Depth-First Search Tree

If $dfs(v)$ is called by $dfs(u)$, then make $u \rightarrow v$ a *tree edge* with u the parent.

- Add children of a node in order, from the left.

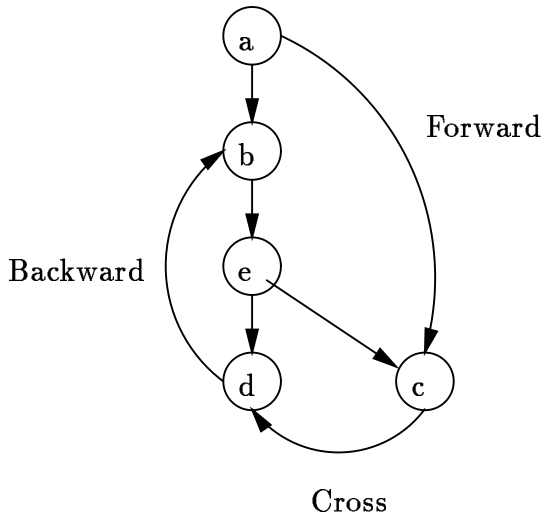
Example:



Other Arcs

The other arcs of the graph fall into 3 groups:

1. *Forward* arcs: ancestor-to-proper-descendant
2. *Backward* arcs: descendant-to-not-necessarily-proper-ancestor.
3. *Cross* arcs: right-to-left only.



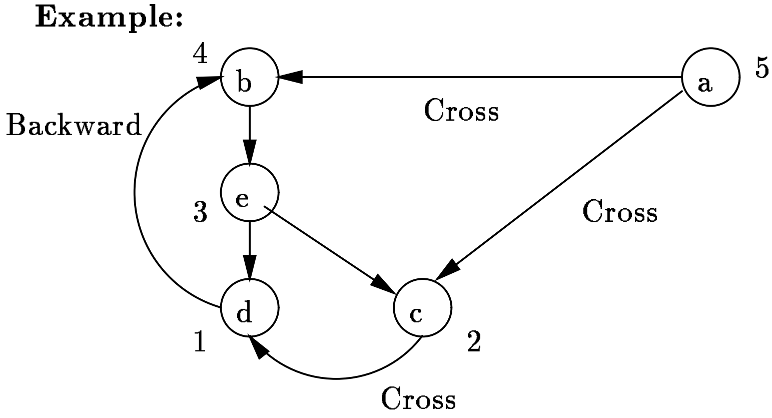
- ☐ left-to-right impossible — see FCS, pp. 488–489.

DFS Forest

If some nodes not included in first tree, start again from some unmarked node.

- Result is a sequence of trees, ordered left-to-right in order of creation = *depth-first search forest*.
- Note arcs between trees must go right-to-left.
  - ☐ These are considered cross arcs.

Example:



Postorder Numbering

We may number nodes in the order that *dfs* finishes on the node.

Example: Figure above shows postorder numbers for this DFSF.

Postorder Numbers and Arcs Types

If $u \to v$ is an arc, then the postorder number of u is the postorder number of v unless $u \to v$ is a backward arc.

- FCS, pp. 493–4 explains why.

Running Time

DFS takes time at each node u proportional to the number of successors of u, plus $O(1)$ in case there are no successors.

- Thus, total time is $O(n)$ for reaching each node, plus $O(m)$ for examining successors of all nodes.
  - □ Important trick: efforts at different nodes varies, but total is proportional to number of arcs. (Details: FCS, p. 491.)
  - □ Since $n \leq m$, total is $O(m)$, i.e., proportional to size of data.

Why Depth-First Search?

A number of important algorithms are based on depth-first search.

- Acyclicity and topological sorting (in class).
- Finding connected components (FCS, p. 499).
- More advanced, very efficient algorithms for:
  - □ *Planarity* testing: can a graph be drawn in the plane with no crossing edges? (important for integrated circuit layout, e.g.)
  - □ *Strong components*: equivalence classes in directed graph defined by uEv iff there are paths from u to v and back.
  - □ *Biconnected components*: equivalence classes in an undirected graph defined by uEv iff $u = v$ or u and v are on a common simple cycle. (important for "survivable"

networks = loss of an edge cannot disconnect nodes)

Testing For Cycles

A graph is *acyclic* if it has no cycles.

1. Create a DFSF.
2. Look at all arcs to see if they are backward. Easy: just see if the head $\geq$ tail.

- If a backward arc, surely a cycle.
- If no backward arc, then surely no cycle.
    - □ Proof: consider the postorder numbers of nodes on such a cycle. All arcs decrease the number, but the sum of changes around the cycle would have to be 0.

Topological Sorting

Given an acyclic graph, find a *topological* ordering of the nodes so that all arcs have their tail preceding their head.

- The reverse of postorder serves.
- The relation uRv iff there is a path from u to v is a partial order if the graph is acyclic. The topological sorting is a total order containing this partial order.

Class Problem

Given an acyclic graph and a *source* node s, find the length of the shortest path from s to each node it can reach.

- Start with a topological order of the nodes, and visit them in this order. Consider the successors v of each node u visited and deduce something about the shortest path to v from the already-known shortest path to u.
- Also: invent a similar algorithm to find the longest path from s to each node.