Discussion #32

Properties and Applications of Depth-First Search Trees
Topics

• Depth-first search trees and forests
• Tree edges; forward, backward, and cross edges
• Post order numbers
• Applications
  – Cycles
  – Topological sorting
  – Reachability
  – Connected components
DFS Trees and Forests

Convention: Roots in decreasing order; other nodes in increasing order

Note: $O(m)$ to create (check each edge once)
Add all edges — but make them dashed if a marked node is encountered.

Tree edge if \( y \) is a child of \( x \) in DFS forest.
Forward edge if \( y \) is a descendent of \( x \), but not a child.
Backward edge if \( y \) is an ancestor of \( x \) or if \( x = y \).
Cross edge if \( y \) is not \( x \) and is neither a descendent nor an ancestor of \( x \).
Observations about Edge Classification

- Requires $O(m)$ to create — each edge considered only once
- Only go between trees in a forest with cross edges
- If we go left to right in building the DFS tree, then all cross edges go from right to left. (For suppose not, then the edge must be a tree edge.)

Must not have been visited! All visited nodes are to the left (cross), above or same (backward), or below (forward).
Postorder Numbers

• Given a DFS tree, do a postorder traversal of the tree edges and number the nodes as they are visited.

• Can be added as the tree is built, so $O(m)$ (assuming $m \gg n$)
Observations about Postorder Numbers

• For each edge $x \rightarrow y$, where $P(x)$ and $P(y)$ are the postorder numbers for $x$ and $y$, if $x \rightarrow y$, then
  1. for a tree edge, $P(x) > P(y)$
  2. for a forward edge, $P(x) > P(y)$
  3. for a backward edge, $P(x) \leq P(y)$
  4. for a cross edge, $P(x) > P(y)$

• There are many interesting applications of all these ideas — all of which take $O(m)$ time, assuming $m \gg n$.  

Discussion #32
Cycles

• A directed graph $G$ is cyclic iff a DFS-tree of $G$ has a backward edge.

• This implies cycle detection can be done in $O(m)$, $O(n^2)$ in the worst case for a dense, nearly complete graph.

• Proof (sketch)
  – (if, i.e. If a DFS-tree of $G$ has a backward edge, $G$ is cyclic.)
    • Let $x \rightarrow y$ be the backward edge.
    • Then $<y, ..., x, y>$ is a cycle.
  – (only if, i.e. If $G$ is cyclic, a DFS-tree of $G$ has a backward edge.)
    • Let the cycle be $<n_1, n_2, ..., n_k, n_1>$.
    • If $k=1$, $<n_1, n_1>$ is a backward edge.
    • If $k>1$, then there is a backward edge. For suppose not, then $P(n_1) > P(n_2) > ... > P(n_k) > P(n_1)$ and thus $P(n_1) > P(n_1)$ — a contradiction.
Topological Sorting
(for acyclic directed graphs)

• Recall that partial orders and Hasse diagrams are acyclic directed graphs.

• A topological sort of a partial ordering $R$ is an imposed total ordering over the elements such that $x$ precedes $y$ if $xRy$.

• Algorithm: List the nodes in reverse postorder (i.e. push nodes on a stack whenever we assign a postorder number; then pop all of them.)
Topological Sort Example

Alphabetical:

Reverse Alphabetical:

Notes: (1) All edges go to the right. (2) The nodes are correctly and totally ordered. (3) More than one ordering is possible.
Topological Sort: Observations

• The algorithm is correct.
  Proof (sketch): The tail of $x \rightarrow y$ must precede the head because $P(x) > P(y)$, for if not then $P(x) \leq P(y)$ and we have a backward edge — a contradiction since the graph is acyclic.

• The algorithm takes $O(m)$ time to build the DFS forest and push the elements onto the stack and takes $O(n)$ time to pop the elements. Thus, the algorithm runs in $O(m)$ time assuming $m \gg n$.

• Why not do regular sort this way?
  – $O(m)$ is $O(n^2)$ in the worst case.
  – We can sort in $O(n \log n)$ time.

• Could we do topological sort using a regular sort?
  – If not, why not?
    • Some nodes not comparable — can let them be equal.
    • If we only have the edges, some nodes that should be comparable through transitivity are not comparable — can do transitive closure to make comparable.
  – If so, should we? (no, why not?)
Reachability

- To test if we can reach \( y \) from \( x \), build a DFS tree starting at \( x \); \( y \) is reachable iff \( y \) is in the tree rooted at \( x \).
- Proof: clear.
- \( O(m), (O(n^2) \text{ in the worst case}) \).
Connected Components for Undirected Graphs

• Replace each edge with two directed edges and then run the DFS forest algorithm.
• \( O(m) + O(m) = O(m) \)
• We can prove: \( x \) and \( y \) are in the same connected component of \( G \) iff \( x \) and \( y \) are in the same DFS tree. (homework)