CISC 320 Introduction to Algorithms Fall 2005

Lecture 13

Topological Sort and Strongly Connected components

Topological sort

Directed Acyclic Graph (DAG)

Topological order of a dag G = (V, E) is
- a linear ordering of all vertices
- If G contains an edge (u,v), then u appears before v in the ordering

TopologicalSort(G)

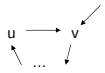
1. Call DFS(G) to compute finishing times f[u] for each vertex u.
2. As soon as each vertex is finished, insert it onto the front of a linked list
3. Return the linked list of vertices.

Topological sort

Theorem: A directed graph G is acyclic *iff* a DFS of G yields no back edges.

Proof:

- If DFS(G) yields a back edge (u,v), then there is a cycle in G. (why?)
- Suppose G has a cycle c. Edge (u,v) is part of c, and v is the first vertex on c discovered by DFS(G). All other vertices on c form a white path from v to u. By White path theorem, u is a descendant of v and edge (u,v) becomes a back edge. (Because DFS-visit(v) won't return to v until all reachable vertices are reached. When it reaches u, (u,v) is a back edge.)



Topological sort

Running time:
- DFS part: O(V+E)
- Insert each of the |V| vertices onto the front of the linked list: O(V)

Total time: O(V+E)

Correctness:

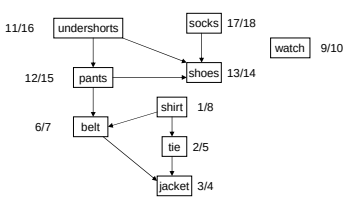
If G is a dag, then for any edge (u,v) $\in$ E $\Rightarrow$ f[u] > f[v]

when (u,v) is explored, u is GRAY.

1. v is also GRAY. Then (u, v) is a back edge. This means G is not a dag.
2. v is WHITE. Then v is a descendant of u. Therefore v is finished before u, namely f[v] < f[u].
3. v is BLACK. Then v is already finished, i.e., f[v] < f[u].

Topological sort

Example



Strongly connected component

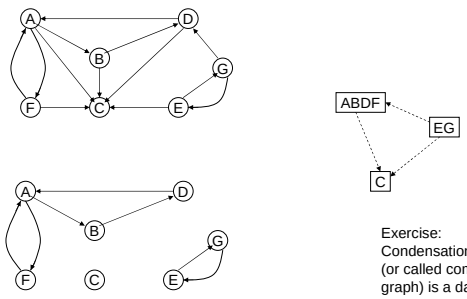
Definition: A scc of a digraph G = (V,E) is a maximal set of vertices U ($\subseteq$V) such that every pair of vertices are reachable from each other in G (why not just in U?).

Q: is it possible that u and v are in a scc and there are edges (u,x) and (x,v), but x is not in the same scc?

SCC(G)

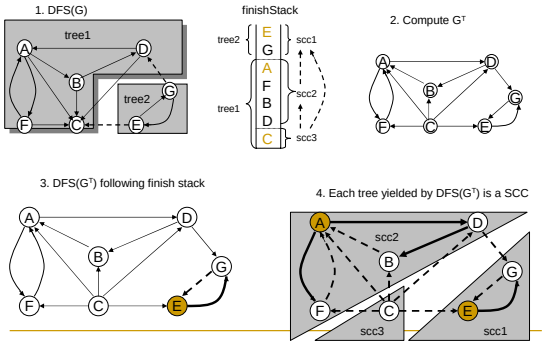
1. Call DFS(G) to compute finishing times f[u] for each vertex u; push u onto finishStack when it is finished.
2. Compute G^T
3. Call DFS(G^T), but in the main loop of DFS, consider the vertices in order of decreasing f[u] (as computed in step 1), that is, take vertices from the finishStack.
4. Output vertices of each tree (in the depth-first forest of step 3) as a separate SCC

Strongly connected component



Exercise:
Condensation graph (or called component graph) is a dag.

Strongly connected component



Strongly connected component

Running time of SCC(G):

- DFS(G): O(V+E)
- Compute G^T: O(V+E) (Exercise)
- Call DFS(G^T): O(V+E)

Total time: O(V+E)

Space usage: O(V+E) using adjacent list representation.

Correctness:

1. each DF tree can contain one or more SCCs, never contains partial SCC. (DFS push a topological ordering of condensation graph of G onto the finishStack)
2. In DFS(G^T), each DF tree can contain only one SCC, because there is no edge to go to the next SCC. (all edges in condensation graph of G are reversed)

DFS on undirected graphs

Similar to DFS on directed graphs
- No cross edges. Why?

Exercise: Design an algorithm to determine whether or not a given undirected graph contains a cycle. Your algorithm should run in O(V) time, independent of |E|.