Analysis of Algorithms	**Strongly Connected Components**
CS6820 Fall 2022	**Wednesday, August 24, 2022**

Here is a linear-time algorithm based on DFS for finding the strongly connected components (strong components) of a given directed graph $G = (V, E)$. This algorithm is due to Tarjan [1].

We start by constructing the DFS forest of G, calculating preorder numbers and a designation of each edge as a tree, back, forward, or cross edge. For $u, v \in V$, define $u \equiv v$ if u and v are in the same strong component; that is, if there is a directed cycle of G containing both u and v.

Let us say that u is *to the left* of v and v is *to the right* of u if neither u nor v is an ancestor of the other and u is visited before v in the depth-first search order. Relative to a given node u, the nodes can be partitioned into four disjoint regions:

(A) proper ancestors of u;[1]

(B) nodes to the left of u;

(C) nodes to the right of u;

(D) descendants of u.[1]

These are illustrated in Fig. 1. These regions are distinguished by the preorder and postorder numbers of their elements relative to u, as given by the following table:

v in region	pre(v) ? pre(u)	post(v) ? post(u)
(A)	$<$	$>$
(B)	$<$	$<$
(C)	$>$	$>$
(D)	$\geq$	$\leq$

Let $[u]$ denote the strong component of u. Let us call a node u *canonical* if it has the minimum preorder number among all elements of $[u]$.

Lemma 1 A node u is canonical iff all elements of $[u]$ lie in the subtree of the DFS forest rooted at u.

Proof. Suppose u is canonical. Looking at Fig. 1, $[u]$ does not intersect regions (A) or (B), as the elements of these regions all have smaller preorder numbers than u. Thus $[u]$ is a subset of the union of regions (C) and (D). But there is no cycle containing nodes of

[1]We consider "ancestor" and "descendant" to be reflexive relations; thus a node is both and ancestor and a descendant of itself. If we mean otherwise, we will say "proper ancestor" and "proper descendant."

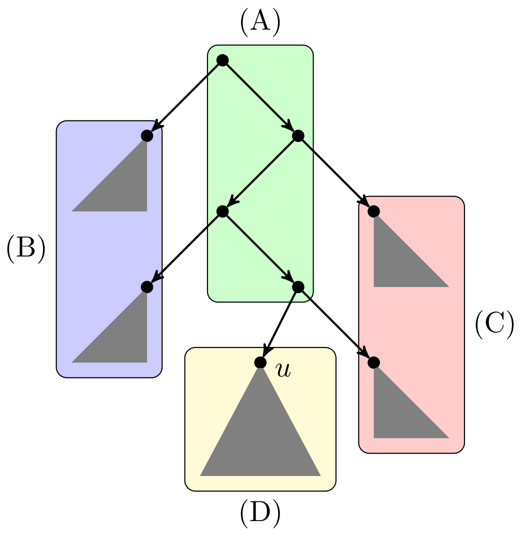

Figure 1: A DFS tree.

both (C) and (D), because there is no edge from (D) to (C): any such edge would have to increase both postorder and preorder numbers, but none of the four types of edges do this. The converse is obvious. □

Using this lemma, we can give a linear-time algorithm for finding the strong components. We traverse the DFS tree again in depth-first order, pruning strong components as we find them. To check whether a node u is a canonical element, we ask whether there is an edge exiting the region (D). We claim that if there is no such edge, then the subtree rooted at u is exactly $[u]$, and u is its canonical element; otherwise, u is not canonical. In the former case, we delete the subtree from the graph and all its incident edges.

To prove the claim, note that the checks are done and subtrees deleted in postorder. Thus at the time we do the check at u, all strong components whose canonical elements have smaller postorder numbers have already been deleted. These were subtrees of regions (B) and (D).

If there is an edge exiting the region (D), say to a node w, then it must be either a back edge to region (A) or cross edge to region (B). If the former, then u is not canonical, as w is in $[u]$ and is a proper ancestor of u. If the latter, then the canonical element of $[w]$ must lie in region (A); if it lay in (B), then $[w]$ would already have been deleted. In that case as well, $[u]$ contains a proper ancestor of u, thus u is not canonical. Otherwise, no edge from region (D) leaves region (D), thus u is canonical. Moreover, every element of (D) is in $[u]$, as all other strong components contained in (D) have already been deleted. □

The algorithm is linear time. Whether there exists an edge exiting the region (D) can be determined recursively during the DFS traversal by calculating the minimum preorder number accessible from some descendant of u via a single edge and comparing the preorder number to u. The algorithm as we have given it involves two DFS passes over the graph, the first to calculate the preorder and postorder numbers, and the second to find and prune the strong components ([1] does both in one pass). The pruning takes linear time in all, because each node and edge is deleted only once.

References

[1] R. E. Tarjan. Depth-first search and linear graph algorithms. *SIAM J. Computing*, 1(2):146–160, 1972.