Website for CS5800

Strongly Connected Components Properties

When a directed graph G with vertex set V and edge set E is traversed via depth-first search, the vertexes V form a forest where the edges are the tree edges. Each vertex v in V is then the root of a subtree T_v in this forest. A useful property of trees is that two subtrees T_u and T_v in a forest are either disjoint or one is contained in the other.

In his paper on his strongly-connected component algorithm, Tarjan proved a number of useful properties about paths in G and how they can be related to the forest. These are shown here.

Property 1. Let u_1, u_2, ..., u_k be any vertexes of G such that their subtrees T_{u1}, T_{u2}, ..., T_{uk} are disjoint from one another. Suppose that p is a path in G all of whose vertexes are in these subtrees. If the first vertex of p is visited before the last vertex of p (i.e., the previsit times), then p is contained entirely within a single subtree.

Proof. Let (v,w) be one of the edges in the path p. By assumption, the vertexes of p are in the subtrees. Let T_{u_i} be the subtree containing v and let T_{u_j} be the one containing w. There are four possibilities for what kind of edge (v,w) is:

1. A tree edge. In this case w is in the same subtree as v since w is a child of v and v is a descendants (or equals) u_i. So T_{u_i} must be the same as T_{u_j}.
2. A forward edge. In this case w is in the same subtree as v since w is a descendant of v and v is a descendant (or equals) u_i. So T_{u_i} must be the same as T_{u_j} again.
3. A back edge. In this case w is an ancestor of v. So v is a descendant of w and w is a descendant (or equals) u_j. So once again T_{u_i} must be the same as T_{u_j}.
4. A cross edge. This is the only case in which the two subtrees could be different. Suppose that they are different. As we know from the properties of the previsit and postvisit times: $\text{pretime}(u_i) < \text{pretime}(v) < \text{posttime}(u_i)$ and $\text{pretime}(u_j) < \text{pretime}(w) < \text{posttime}(u_j)$. Since the subtrees T_{u_i} and T_{u_j} are disjoint, the times when they are being visited are disjoint intervals (i.e., all of the pre and postvisit times of one subtree are before all of the pre and postvisit times of the other). Since (v,w) is a cross edge, the pretime of v is later than the pretime of w. Therefore, $\text{pretime}(u_j) < \text{pretime}(w) < \text{posttime}(u_j) < \text{pretime}(u_i) < \text{pretime}(v) < \text{posttime}(u_i)$. In particular, $\text{pretime}(u_j) < \text{pretime}(u_i)$.

The four possibilities above tell us that for every edge (v,w), either v and w are in the same subtree or they are in two subtrees such that the root of the subtree containing w is visited before the one for v. Therefore, if we look at the pretimes of the roots of the subtrees of the vertexes in p, then as we proceed along the path, the pretimes must either stay the same or decrease. But we assumed that the first vertex of p was visited before the last vertex was visited. So all of the vertexes must be in the same subtree.

Property 2. Let p be a path in G. If the first vertex was visited of p before the final vertex of p, then there is a vertex v of p such that all of the vertexes of p are in the subtree T_v.

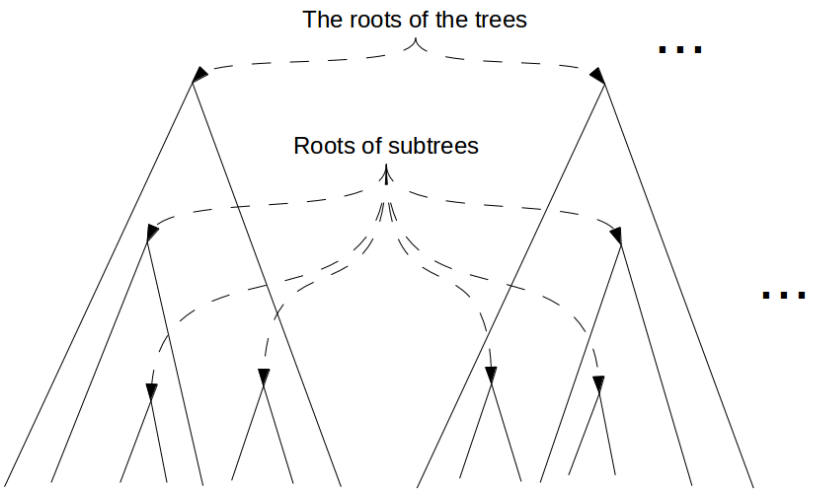
Proof. DFS produces a forest. Let u_1, u_2, ... u_k be the roots of the trees in this forest. Of course, since this is the entire forest, p is contained in it. So Property 1 applies. As an exercise, complete this proof.

Theorem. Let C be one of the strongly connected components of G. If v is the vertex of C that is the first one visited during the DFS (i.e., the pretime of v is the smallest of all vertexes in C), then C is contained in the subtree Tv.

Proof. Let u be any vertex of C other than v. By definition of a strongly connected component, there is a path p from v to u. Since v was the first vertex of C that was visited, the pretime of v is earlier than the pretime of u. So Property 2 applies. Therefore, there is a vertex w of C such that the entire path p is contained in T_w. In particular, v is in T_w. If v is the same as w, then the result has been shown. The other possibility is that v and w are different. Since v is in T_w, the vertex v is a descendant of w. This means that the pretime of w is earlier than the pretime of v. But v is the first vertex of C that was visited, so the pretime of v is earlier than the pretime of w. This is not possible, so the only possibility is that v is the same as w.

As consequence of this theorem, each SCC has a "root"; namely, the first vertex of the SCC that is visited during DFS. Note that the root in this sense is not intrinsic to the SCC but depends the order in which the adjacent vertexes are visited from each vertex. The following diagram shows how the SCCs are arranged within the directed graph after a DFS has been performed.

DFS constructs a forest whose edges (parent-child links) are the “tree edges.”
The following picture shows the trees in the forest.



The roots of the trees and subtrees are the roots of the SCCs. Each SCC consists of all vertexes that are in the subtree of the root and are not in any subtree of an SCC that is contained in the subtree of the root. The SCCs can be constructed during the DFS with a modified form of the union-find algorithm. In this case, the root of the SCC is a root of the disjoint set structure constructed by the union-find algorithm. In this algorithm, each SCC is removed from the graph when its construction has been completed. This is easily indicated by adding a `removed` Boolean variable to each node which is initially false and set to true when the SCC of the node is removed.

Initially, all vertexes are SCC roots. Whenever a back edge is encountered, the vertexes in the cycle are all made part of the same SCC by calling union on them. When a cross edge is encountered that links a vertex A to a vertex B that has not yet been removed, then there will be a cycle and the vertexes in the cycle should be added to the SCC of vertex B by calling union on them. Other cross edges can be ignored, and all forward edges can be ignored. During the postvisit to a vertex that is a root, the vertexes in the SCC of the root are all marked as removed.

This is, essentially, the original Tarjan algorithm. I have omitted some technical details

about how the SCCs are constructed. In particular, the Tarjan algorithm does not use a disjoint set structure for constructing the SCCs. Note that the original Tarjan algorithm uses only one DFS. The one in the textbook uses two DFSs.