An $O(|V|^2)$ Algorithm for Single Connectedness

Samir Khuller *

Abstract

A directed graph $G = (V, E)$ is said to be *singly connected* if $u \leadsto v$ implies that there is at most one simple path from u to v for all vertices $u, v \in V$. In this paper we design an algorithm to test a graph for being singly connected that takes $O(|V|^2)$ steps.

1. Introduction

A directed graph $G = (V, E)$ is said to be *singly connected* if $u \leadsto v$ implies that there is at most one simple path from u to v for all distinct[1] vertices $u, v \in V$. This problem is mentioned in the textbook by Cormen, Leiserson and Rivest [1] on page 485. Personal communication with Tom Cormen indicates that one can design an algorithm that runs in $O(|E||V|)$ steps by doing a DFS from each vertex, and that this was the best available solution when the book was written. (If each DFS yields only tree and back edges then the graph is singly connected.) For sparse graphs this is a good solution, but for dense graphs this may be quite inefficient. We design an algorithm to test a graph for being singly connected that takes $O(|V|^2)$ steps. We leave open the question of designing an algorithm that takes only $O(|E| + |V|)$ steps.

2. Algorithm

We first compute the strongly connected components of the graph. These can be computed in linear time [1]. Once we compute the strong component graph[2] G^{SCC} we check to see if there are any multiple edges in this graph between two vertices (strong components). If there are multiple edges between two vertices, then this implies that there are two strongly connected components with multiple edges between them; it is easy to see that in this case the graph cannot be singly connected. We may now assume that G^{SCC} does not have any multiple edges.

We first explain the intuition behind the algorithm. If there is a pair of vertices a and b in G such that there are at least two simple paths between them, then there are two cases. Either the vertices are in the same strongly connected component or in different strongly connected components. We will first address the former case. Observe that the multiple paths are restricted to using vertices

*Department of Computer Science and Institute for Advanced Computer Studies, University of Maryland, College Park, MD 20742. Research supported by NSF Award CCR-9820965 and an NSF CAREER Award CCR-9501355. E-mail : `samir@cs.umd.edu`.

[1]The solution below can be modified to work without this requirement as well.

[2]This is the directed acyclic graph obtained contracting all strong components to single vertices

in the same strongly connected component as the vertices themselves. We will show that this case is easy to detect in $O(|E|)$ time. In fact a strongly connected graph that is also singly connected has a very simple structure as is shown by the following theorem.

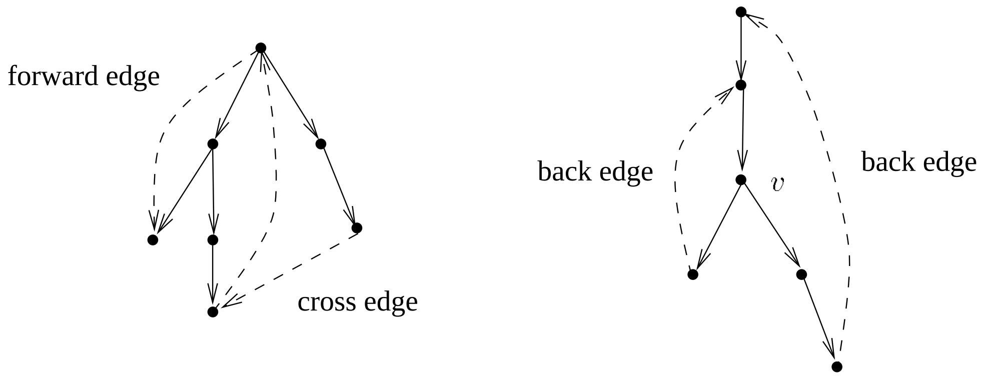

Figure 1: If a strongly connected graph has either a cross edge, or a forward edge or double back edges then it is not singly connected.

Theorem 2.1: *Let H be a strongly connected graph. H is not singly connected if and only if at least one of the following conditions holds. The DFS search in H either yields a cross edge, or a forward edge, or a vertex v such that from the subtree rooted at v, there are at least two back edges to proper ancestors of v.*

Proof: We first argue that if any of these conditions hold then there are two simple paths between some pair of vertices (see Figure 1). The DFS search has a single root, since all the vertices in H are reachable from this vertex. Any forward edge or cross edge will immediately yield a pair of vertices with multiple simple paths between them. If we have a vertex v with the property that there are at least two back edges out of the subtree rooted at v to proper ancestors of v, then again there is a pair of vertices with multiple simple paths between them.

We now need to argue that if none of the three conditions hold, then the graph is singly connected. Assume that the DFS search does not yield a forward edge, or a cross edge. Assume that the graph is not singly connected. In this case there is a pair of vertices a and b such that there are two simple paths from a to b. Let the last vertex on each path (before b) be c and d respectively (note that a could be c or d itself, and this only simplifies the proof). Since b can only have one incoming tree edge, either (d, b) is a back edge and (c, b) a tree edge[3] or both (c, b) and (d, b) are back edges.

We first deal with the case when (c, b) is a tree edge and (d, b) is a back edge. Let $T[b, d]$ be the path in the DFS tree from b to d with x as the first vertex other than b on this path (see Figure 2). Since a has a path to d avoiding b we claim that $LCA(a, d)$ is on $T[x, d]$. ($LCA(x, y)$ is the least common ancestor of x and y in a rooted tree.) To see this note that the LCA is an ancestor of d. If it is a proper ancestor of x then a cannot reach d avoiding b. Hence it must be on $T[x, d]$. This would in turn imply that a is in the subtree rooted at x. Since the subtree rooted at x has only

[3] We can name c and d appropriately.

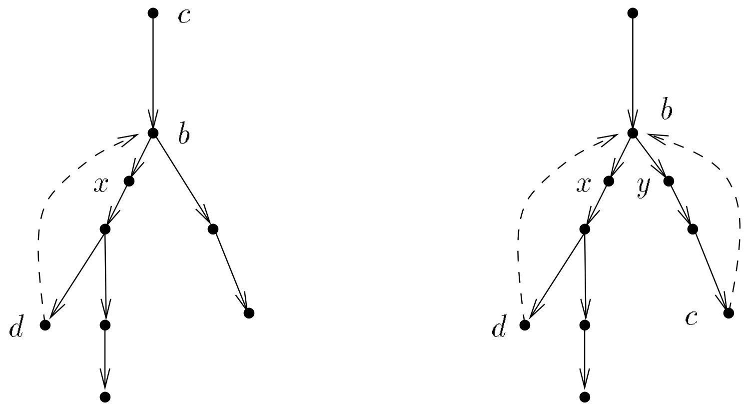

Figure 2: Cases when (c, b) is a tree or back edge

one back edge coming out of it (the back edge (d, b)), there is no path from a to c avoiding b which yields a contradiction.

In the second case when both (c, b) and (d, b) are back edges. Note that c and d are in the subtree rooted at b. We claim that $LCA(c, d) = b$, otherwise there is a vertex with two back edges going out of its subtree to a proper ancestor, namely b. Let $T[b, d]$ be the path in the DFS tree from b to d with x as the first vertex other than b on this path. Let $T[b, c]$ be the path in the DFS tree from b to c with y as the first vertex other than b on this path. Since a has a path to d avoiding b we claim that $LCA(a, d)$ is on $T[x, d]$. Similarly, we claim that since a has a path to c avoiding b we claim that $LCA(a, c)$ is on $T[y, d]$. This would imply that a is in the subtree rooted at x and the subtree rooted at y, but these two subtrees are disjoint, a contradiction. $\square$

It is very easy to design an $O(|E|)$ algorithm to check for these conditions in each strongly connected component of G. This concludes the case when a and b are in the same strong component.

We now deal with the case when they belong to different strong components. In the latter case observe that the graph G^{SCC} in fact has a pair of nodes with multiple paths between them. We will perform repeated depth first searches from *each* vertex in G^{SCC}. If we ever encounter a forward edge, or a cross edge then the graph is not singly connected. Since there are no back edges, each DFS will only take $O(|V|)$ steps since there are only $|V| - 1$ tree edges. Repeating this for all vertices will take $O(|V|^2)$ steps.

Acknowledgments

I would like to thank Koushik Kar for asking me this question while studying for his final exam. I would also like to thank Dave Mount, Tom Cormen and An Zhu for discussions.

References

[1] T. H. Cormen, C. E. Leiserson and R. L. Rivest, "Introduction to Algorithms", *MIT Press* (1989).