A Simplified Correctness Proof for a Well-Known Algorithm Computing Strongly Connected Components

Ingo Wegener
FB Informatik, LS2, Univ. Dortmund,
44221 Dortmund, Germany
wegener@ls2.cs.uni-dortmund.de

Abstract

The computation of the strongly connected components of a directed graph is one of the fundamental algorithmic graph problems. Linear-time algorithms with simple implementations are known. Here a simplified correctness proof for one of these algorithms is presented.

Keywords: algorithms, depth first search, graph problems, strongly connected components.

1 Introduction

The computation of the strongly connected components of a directed graph is one of the fundamental algorithmic graph problems. Remember that two vertices v and w are strongly connected if there is a directed path from v to w (denoted by $v \rightarrow w$) and a directed path from w to v. A strongly connected component (SCC) is a maximal vertex set where all pairs of vertices are strongly connected. The SCCs are a partition of the vertex set. Tarjan[10] presented the first linear-time algorithm for the computation of all SCCs. His algorithm is based on a DFS traversal and the computation of the so-called low-link numbers. This resembles the computation of the biconnected components of undirected graphs. However, the correctness proof is not too simple (see also [1], [6], [8], and [5]). Variants of this approach are presented in [7] and [9]. A different algorithm is described in [2], [3], and [4]. Again, the presented proofs are not too simple. My experience is that students have difficulties with these SCC algorithms. Therefore, a much simpler and more illustrative correctness proof is presented. In Section 2, the algorithm from [2], [3], and [4] is described. In Section 3, we describe which information on SCCs can be derived from a DFS traversal. These considerations lead to the new correctness proof which uses an inductive argument on the number of SCCs.

2 The Algorithm

Here we present a well-known SCC algorithm ([2], [3], [4]).

Algorithm 1 $G = (V, E)$ is a directed graph given by adjacency lists.

1.) Perform a DFS traversal on G and compute numbers such that a vertex v whose DFS call is finished *later* than the DFS call for w gets a *smaller* number. We refer to these numbers as f-numbers, since they depend on the point of time when the corresponding DFS call is finished.

2.) Compute the reversed graph $G^* = (V, E^*)$ where $(v, w) \in E^*$ iff $(w, v) \in E$.

3.) Perform a DFS traversal on G^* where the vertex array (deciding at which vertex a new tree is started) is sorted according to the f-numbers.

This algorithm is easy to implement and it is obvious that the runtime is linear, namely $O(n + m)$. The claim is that the vertex sets of the DFS trees computed in Step 3 are the SCCs of G. This looks like a miracle.

3 The Correctness Proof

We start our investigations without considering Algorithm 1. We ask which information on the SCCs can be gained from a simple DFS traversal on G, see Figure 1 for an example. We use the convention that the DFS trees are constructed from left to right and that the same holds for the subtrees of a tree.

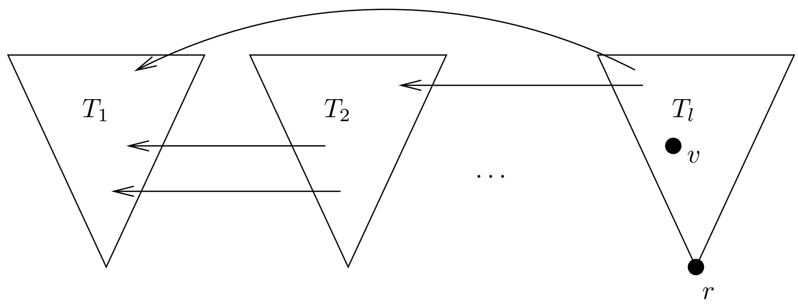

Figure 1: The result of a DFS traversal on G.

Because of our convention, edges between trees lead from right to left and there is no directed path from T_i to T_j, if $i < j$. Hence, each SCC is contained in one of the DFS trees.

How can we compute one special SCC? We choose the SCC C containing the root r of the last (rightmost) DFS tree T_l. The reasons for this choice are twofold. First, r is the only vertex of T_l where we know that $r \to v$ for all v in T_l. Hence, C is exactly the set of all vertices v in T_l such that $v \to r$. Moreover, the set of these vertices can be computed by computing all vertices v in T_l such that $r \to v$ after reversing all edges. Second, this also explains why we have chosen the last tree T_l. After reversing all edges the DFS traversal starting at r cannot reach vertices outside T_l. Hence, the DFS traversal starting in G^* at r computes the vertices of the SCC containing r as vertex set of the first DFS tree. Moreover, it is obvious that r is the vertex with the smallest f-number (Step 1 of Algorithm 1).

Do we know further properties of the SCC C containing r? If the vertex v of T_l (see Figure 1) belongs to C and $(r, v_0, \ldots, v_m = v)$ is the T_l-path from r to v, then $v_0, \ldots, v_{m-1}$ also belong to C, since this T_l-path and a path from v to r form a cycle. Hence, C is a connected part of T_l (see Figure 2). This implies that T_l without C consists of subtrees of T_l (rooted in Figure 2 at $v_1, \ldots, v_5$).

In order to use an inductive argument we like to argue that the resulting forest consisting of $T_1, \ldots, T_{l-1}$, and the subtrees of T_l is also the result of a DFS traversal on the subgraph of G on $V - C$. However, this depends on the initial numbering of the vertices (deciding where to start a new tree). If this numbering equals the DFS numbering obtained in Step 1 of Algorithm 1, we obtain the same DFS forest on V as before and we obtain the desired DFS forest on $V - C$. For our correctness proof we can assume that we start with

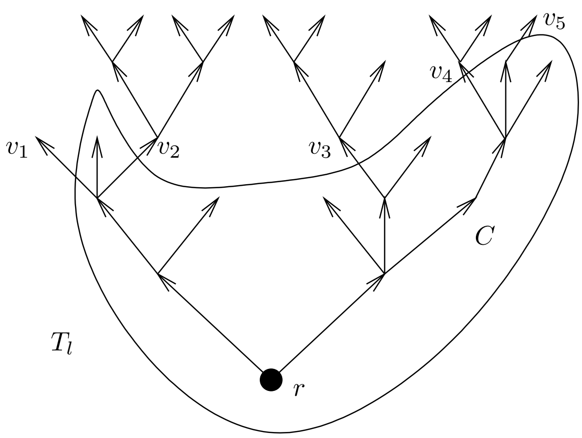

Figure 2: The DFS tree T_l with the SCC C containing r.

this DFS numbering, since this does not change the result of the algorithm. Surprisingly enough, this is the main observation.

The formal inductive argument is now easy. If G contains only one SCC, this SCC is computed correctly. If G contains k SCCs, then the algorithm computes first the SCC C containing the root of the last DFS tree. Then Step 3 of the algorithm works on the DFS forest obtained by eliminating C and implicitly on a subgraph of G with $k-1$ SCCs. The correctness of the algorithm on this subgraph follows from the induction hypothesis.

All correctness proofs not using induction have to argue about all SCCs and, therefore, much more carefully about the relations between the time intervals of the different DFS calls leading to more sophisticated arguments on the f-numbers.

We hope that this clear and illustrative correctness proof supports the teaching of the SCC algorithm described in Section 2.

References

[1] A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *The Design and Analysis of Computer Algorithms.* Addison-Wesley, 1974.

[2] A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *Data Structures and Algorithms.* Addison-Wesley, 1983.

[3] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms.* MIT Press, 1990.

[4] R. H. Güting. *Datenstrukturen und Algorithmen.* Teubner, 1992.

[5] J. van Leeuwen (Ed.). *Handbook of Theoretical Computer Science. Vol. A: Algorithms and Complexity.* MIT Press, 1990.

[6] K. Mehlhorn. *Data Structures and Algorithms. Vol. 2: Graph Algorithms and NP-Completeness.* Springer, 1984.

[7] T. Ottmann and P. Widmayer. *Algorithmen und Datenstrukturen.* Spektrum Akademischer Verlag, 1996.

[8] E. M. Reingold, J. Nievergelt, and N. Deo. *Combinatorial Algorithms: Theory and Practice.* Prentice-Hall, 1977.

[9] R. Sedgewick. *Algorithms.* Addison-Wesley, 1983.

[10] R. E. Tarjan. Depth-first search and linear graph algorithms. SIAM Journal on Computing 1, 146-160, 1972.