A simple and optimal algorithm for finding immediate dominators in reducible graphs

Stephen Alstrup* Peter W. Lauridsen*

Abstract

We present two simple algorithm for finding immediate dominator in reducible graphs with n nodes and m edges: A $O(n + m)$ RAM algorithm and a $O(n + m \log \log n)$ pointer machine algorithm.

1 Introduction

Algorithms for finding dominator trees for control flow graphs are described in [5, 7, 8]. Dominator trees are used in control flow analysis [1, 4]. In [5] a linear time algorithm is given. This algorithm is complicated and to our knowledge no experimental results using this algorithm have been published. This is the motivation for presenting two simpler algorithms, one of which runs on a pointer machine [10]. The algorithms presented in this paper have previously been described by the authors of this paper and also independently and simultaneously in [9]. But at that time the important results from [2, 3], were not applied, so the contribution of this paper is only a compilation.

2 Notation

A control flow graph $CFG(V, E, s)$ is a directed graph with a start node s, from which all nodes in V is reachable through the edges E. Node x dominates y if and only if all paths from s to y pass through x. The dominance relation is reflexive and transitive, and can be represented by a tree, called the dominator tree. If y is the parent of x in the dominator tree, then y immediately dominates x, denoted as $idom(x) = y$.

3 Algorithm

1) Given a $CFG(V, E, s)$ we first compute a depth first tree with root s and all back-edges are removed, reducing the graph to $CFG(V, E's)$. This has no effect on the dominance relation as Tarjan has observed [5].
2) The graph $CGF(V, E', s)$ is acyclic and can therefore be topologically sorted [6] ensuring that if $(v, w) \in E'$ then v has a lower topological number than w.
3) Now the dominator tree T can be constructed dynamically. Set s to the root of the dominator tree T and process the nodes from $V \backslash \{s\}$ in increasing topological order as follows. Notice that the part of T, which at any point has been build is used for determening *idom* for the rest of the nodes.

*E-mail:(stephen,waern)@diku.dk. Department of Computer Science, University of Copenhagen. February 9, 1996.

- Let $W = \{y|(y,x) \in E'\}$ be the set of predecessors of x in $CFG(V, E', s)$ and let A be the set of nodes, where each node in A is an ancestor in T to all nodes in W. The node $idom(x)$ is the node in A with the largest depth in T. Hence $idom(x)$ can be computed by repeatedly deleting two arbitrary nodes v and w from W and inserting the nearest common ancestor (nca) of these nodes into the set W until the set contains only one node.

- After computing $idom(x)$ the edge $(x, idom(x))$ is added to T.

The only unspecified part of the algorithm is the computation of nca in a tree T which grows under the addition of leaves. In [3] a RAM algorithm is given which processes nca and addition of leaves in $O(1)$ time per operation and in [2] a pointer machine algorithm is given which processes addition of leaves in $O(1)$ and nca in $O(\log\log n)$ time.

Theorem 1 *The dominator tree for a reducible control flow graph with* n *nodes and* m *edges can be determined in* $O(n+m)$ *on a RAM and* $O(n + m\log\log n)$ *on a pointer machine.*

Proof. Step 1 and 2 in the algorithm have complexity $O(n+m)$. In step 3 each node is visited and each edge can result in a query about nca in T, so at most m nca-query is performed, which establishes the complexity. □

References

[1] A.V. Aho, J.E. Hopcroft, and J.D. Ullman. On finding lowest common ancestor in trees. In *Annual ACM Symposium on the theory of computing (STOC)*, volume 5, pages 115–132, 1973.

[2] S. Alstrup. Optimal algorithms for finding nearest common ancestor in dynamic trees. Technical Report 95-30, Department of Computer Science, University of Copenhagen, 1995.

[3] H.N. Gabow. Data structure for weighted matching and nearest common ancestors with linking. In *Annual ACM-SIAM Symposium on discrete algorithms (SODA)*, volume 1, pages 434–443, 1990.

[4] R. Gupta. Generalized dominators and post-dominator. In *Ann. ACM Symp. on Principles of Programming Languages*, volume 19, pages 246–257, 1992.

[5] D. Harel. A linear time algorithm for finding dominator in flow graphs and related problems. In *Proc. 17th Annual ACM symposium on theory of computing*, pages 185–194, 1985.

[6] D.E. Knuth. *The art of programming*, volume 1. Addison-Wesley, 1968.

[7] T. Lengauer and R.E. Tarjan. A fast algorithm for finding dominators in a flowgraph. *ACM Trans. Programming Languages Systems*, 1:121–141, 1979.

[8] P.W. Purdom and E.F. Moore. Immediate predominators in a directed graph. *Comm. ACM*, 15(8):777–778, 1972.

[9] G. Ramalingam and Thomas Reps. An incremental algorithm for maintaining the dominator tree of a reducible flowgraph. In *popl21*, pages 287–298, 1994.

[10] R.E. Tarjan. A class of algorithms which require nonlinear time to maintain disjoint sets. *Journal of computer and system sciences*, 18(2):110–127, 1979.