VISVESWARAYA TECHNOLOGICAL UNIVERSITY

BELGAUM

SRI JAYACHAMARAJENDRA COLLEGE OF ENGINEERING, MYSORE-570006

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Report on

IMPLEMENTATION OF LENGAUER-TARJAN ALGORITHM TO GENERATE DOMINATORS FOR NODES OF A RANDOM GRAPH

Guidance of

N.R. PRASHANTH

Professor

Department of CS&E, SJCE, Mysore.

Done By:

VIKRAM T.V.

4th Semester,

Computer Science and Engineering,

S.J.C.E, Mysore

Contents

1 Introduction

Prosser introduced the notion of dominance in a 1959 paper on the analysis of flow diagrams, defining it as follows:

We say box i dominates box j if every path (leading from input to output through the diagram) which passes through box j must also pass through box i. Thus box i dominates box j if box j is subordinate to box i in the program.

For SSA to help make a compiler faster, we must be able to compute the SSA form quickly. The algorithms for computing SSA from dominator tree are efficient. Iterative set-based algorithm to compute dominators are slow for worst cases, but the Langauer-Tarjan algorithm is much more efficient with a linear complexity.

2 Construction

The construction of the dominator tree for a random graph starts with depth-first ordering of the nodes. This is done by the depth first search algorithm. Using the order of traversal, the semidominators are computed for each node using the semidominator theorem. Finally, the dominator theorem uses the semidominators to calculate the immediate dominator for each of the nodes of a random graph.

2.1 Depth-First Spanning Trees

The DFS algorithm on a random graph gives the depth-first ordering of the nodes. Each node is assigned with a depth-first number called as *dfnum*. There can different possible depth-first orderings, but any one ordering can suffice. Also if there is a path from a to b following the spanning tree edges or $a = b$, then we say *a is an ancestor of b*. If a is an ancestor of b and $a \neq b$ then *a is a proper ancestor of b*.

Properties: If a is a proper ancestor of b, then *dfnum(a)* $<$ *dfnum(b)*.

If there is a path from a to b but a is not an ancestor of b, then *dfnum(a)* $>$ *dfnum(b)*, and this means the path includes some non-spanning edges of the tree.

Also, the left most branches are visited first and thus the dfnum's of left branches are less than the dfnum's of right branches. Therefore, knowing a path from a to b, we can test whether a is an ancestor of b by just comparing the dfnum's of a and b.

2.2 Semidominators

Consider a non-root node n in the random graph with its immediate dominator d, which lies between the root and n and d is an ancestor of n. Thus, *dfnum(d)* $<$ *dfnum(n)*. If x is some ancestor that does not dominate n, then there must be a path that departs from spanning-tree above x and rejoins below x. As the bypassing nodes have higher dfnum's, they are not ancestors of n. The bypassed path may rejoin spanning-tree path to n either at n or above n.

If the bypassing path rejoins above n then the immediate dominator of n is its parent. Let the bypassing path rejoin the spanning tree at n. There is a path that departs from the tree at the highest possible ancestor s to n and we call s as the *semidominator* of n. That is, s is the node of smallest *dfnum* having a path to n whose nodes are not ancestors of n. Usually, a node's semidominator is also its immediate dominator. But, there are cases, when s itself resides in one of the bypassing paths. In such cases, the dominator of s becomes the dominator of n. Such a path is shown in figure 1. Node y is a node between s and n, where the bypassing around s rejoins. If y is a smallest-numbered semidominator and *semi(y)* is a proper ancestor of s, then y's immediate dominator also immediately dominates n.

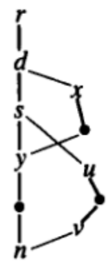

Figure 1: Semidominator

Semidominator Theorem: To find the semidominator of a node n, consider all predecessors v of n in the random graph.

- If v is a proper ancestor of n in the spanning tree, *dfnum(v)* $<$ *dfnum(n)*, then v is a candidate for *semi(n)*.
- If v is a non-ancestor of n, *dfnum(v)* $>$ *dfnum(n)*, then for each u that is an ancestor of v, or $u = v$, *semi(u)* is a candidate for *semi(n)*.

Of all these candidates, the one with the lowest *dfnum* is the semidominator of n.

2.3 Dominator Theorem

On the spanning-tree path below *semi(n)* and above or including n, let y be the node with the smallest-numbered semidominator (minimum *dfnum(semi(y))*). Then,

$$idom(n) = \begin{cases} semi(n) & \text{if } semi(y) = semi(n) \\ idom(y) & \text{if } semi(y) \neq semi(n) \end{cases}$$

3 Implementation

The semidominator theorem and dominator theorem have been implemented alongwith depth-first search algorithm. The dfs(parent, node) assigns the *dfnum* described by the count N. Also an array *vertex*[N] maps the dfnum back to the nodes. The array *parent[n]* maintains the parent of each node n. Each node is visited recursively and marked as visited using the *dfnum[n]* array. A psuedocode for *dfs* routine is given in algorithm 1.

Algorithm 1 Psuedocode for depth-first search routine

DFS (node p, node n)
if *dfnum[n]* $= 0$ **then**
    dfnum[n] $\leftarrow$ N
    vertex[n] $\leftarrow n$
    parent[n] $\leftarrow p$
    N $\leftarrow$ N $+ 1$
    for each successor w of n **do**
        DFS (n, w)
    end for
end if

Link (node$_1$, node$_2$) adds the edge $p \rightarrow n$ to the spanning forest that is implied by the *ancestor[n]* array. To find the lowest-numbered semidominator for node n other than the root, the function ancestorWithLowestSemi(node v) is implemented. It starts from the rightmost bottom node having the highest *dfnum*, moves up the tree till root, by checking if there exists a lower dfnum for *semi(v)* than the existing one and finally returns the node with lowest-numbered semidominator. Psuedocodes for Link and ancestorWithLowestSemi routines are given in algorithm 2.

Algorithm 2 Psuedocodes for *Link* and *ancestorWithLowestSemi* routines

/* add edge $p \rightarrow n$ to spanning forest implied by *ancestor* array */
Link (node p, node n)
ancestor[n] $\leftarrow p$

/* in a forest, find the nonroot ancestor of n that has the lowest-numbered semidominator
ancestorWithLowestSemi (node v)
$u \leftarrow v$
while *ancestor[v]* $\neq$ none **do**
    if *dfnum[semi[v]]* $<$ *dfnum[semi[u]]* **then**
        $u \leftarrow v$
    end if
    $v \leftarrow$ *ancestor[v]*
end while
return u

The function build_dominator_tree() implements the semidominator and dominator theorem. If the dominator of y is not known, then it is deferred until it is known by using *samedom[n]* array. Due to this deferring, forests of nodes are created. Later, with the second clause of dominator theorem, using the samedom array, the forests are linked and the dominators are calculated. A psuedocode for the building the dominator tree is given in algorithm 3.

Time Complexity: The routine AncestorWithLowestSemi(node v) starts from a node with highest *dfnum*

Algorithm 3 Psuedocode for build_dominator_tree routine

N $\leftarrow 0$
$\forall n.$*bucket[n]* $\leftarrow \{\}$
$\forall$*dfnum[n]* $\leftarrow 0$
semi[n] $\leftarrow$ *ancestor[n]* $\leftarrow$ none
idom[n] $\leftarrow$ *samedom[n]* $\leftarrow$ none
DFS (none, r)
for i $\leftarrow$ N - 1 **downto** 1 **do**
    $n \leftarrow$ *vertex[i]*
    $p \leftarrow$ *parent[n]*
    $s \leftarrow p$
    for each predecessor v of n **do**
        if *dfnum[v]* $\leq$ *dfnum[n]* **then**
            s' $\leftarrow v$
        else
            s' $\leftarrow$ *semi*[ancestorWithLowestSemi(v)]
        end if
        if *dfnum[s']* $<$ *dfnum[s]* **then**
            $s \leftarrow$ *s'*
        end if
    end for
    semi[n] $\leftarrow s$
    bucket[s] $\leftarrow$ *bucket[s]* $\cup \{n\}$
    Link (p, n)
    for each v in *bucket[p]* **do**
        $y \leftarrow$ ancestorWithLowestSemi (v)
        if *semi[y]* $=$ *semi[v]* **then**
            idom[v] $\leftarrow p$
        else
            samedom[v] $\leftarrow y$
        end if
    end for
    bucket[p] $\leftarrow \{\}$
end for
for $i \leftarrow 1$ **to** N - 1 **do**
    $n \leftarrow$ *vertex[i]*
    if *samedom[n]* $\neq$ none **then**
        idom[n] $\leftarrow$ *idom[samedom[n]]*
    end if
end for

(the rightmost bottom node) and runs through the entire graph upto the root taking a O(N) time, N being the total number of nodes in the graph. The dominator routine implements Langauer-Tarjan algorithm taking O(N) time, the linear time. Thus, the complexity of the entire algorithm is $O(N_2)$. If we can achieve a complexity of O(log N) with ancestorWithLowestSemi() by path compression techniques, then the complexity reduces to O(N log N).

4 Sample Run

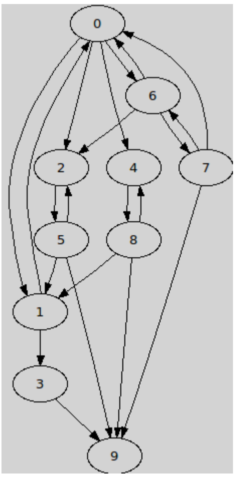

Figure 2: Some randomly generated graph

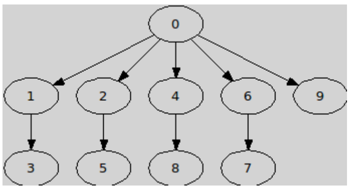

Figure 3: Dominator tree for figure 2

Figure 2 shows a randomly generated graph. After running through the Lenguer-Tarjan algorithm for dominator computation, a dominator tree shown in figure 3 was generated. The root node was considered to be at node 0.

References

[1] appel andrew w. appel, *modern compiler implementation in c*, Revised Edition, Cambridge University Press, 2008