A LINEAR TIME AND SPACE ALGORITHM FOR FINDING ISOMORPHIC SUBTREES OF A BINARY TREE

ERKKI MÄKINEN
University of Tampere, Department of Computer Science,
P.O. Box 607, SF-33101 Tampere, Finland

Abstract.

This paper shows that it is possible to find all isomorphic subtrees of a binary tree in linear time and space.

C.R. categories: E.1, F.2.2, G.2.2, I.2.8.

Key words and Phrases: binary tree, isomorphic subtree.

1. Introduction.

The (binary) tree isomorphism problem is to find all isomorphic subtrees within a given binary tree. Efficient algorithms for this problem are needed e.g. when drawing [5] or compressing [2] trees.

Supowit and Reingold [5] proposed an algorithm which employs a ranking algorithm for binary trees. Knott's ranking algorithm [3] was used in [5], but there are several other ranking algorithms as well. Supowit and Reingold's algorithm traverses the tree in postorder and determines the rank of the subtree rooted in the current node by using the ranks of its subtrees. Isomorphic subtrees naturally have the same rank. The resulting ranks must be sorted in order to find isomorphic subtrees. Hence, the time complexity of this algorithm is $O(n \log n)$ where $n$ is the number of nodes in the tree. However, algorithms using ranks are impractical for large trees as noticed in [4]: the rank of a tree with 5000 nodes may require over 2000 digits in decimal notation.

A linear time algorithm for the tree isomorphism problem was presented by Jacobson [2]. Like the Supowit-Reingold algorithm, Jacobson's algorithm traverses the tree in postorder. Suppose the tree being traversed has $n$ nodes. Jacobson's algorithm assigns each subtree an integer label between 1 and $n$ such that isomorphic subtrees have the same label. The algorithm uses an auxiliary $n \times n$-
table $T$ in which the information concerning traversed nodes is stored. If the left and right subtrees of a node have the labels $x$ and $y$, respectively, then the position $T(x, y)$ is examined. If it contains zero, we know that the current node roots a subtree shape not yet encountered during the traversal. In this case, the current node will have the first free integer label which is also stored in $T(x, y)$. Otherwise, the position $T(x, y)$ contains the label used for subtrees isomorphic to the current one.

This paper shows that the tree isomorphism problem can be solved in linear time and space. The algorithm uses a special tree traversal.

2. The algorithm.

We suppose that trees have auxiliary pointers from children to their parents. These pointers can be added during a single tree traversal.

Our algorithm fully utilizes the following simple observation: If the subtree rooted at a node $a$ is isomorphic to the one rooted at $b$, then the left (resp. right) subtree of $a$ is isomorphic to the left (resp. right) subtree of $b$. As a consequence, if the subtree rooted at $b$ is isomorphic to no other subtree within the tree, then it is unnecessary to study the parents of $b$. We must traverse the tree so that all subtrees of equal height are handled in succession. (The height of a binary tree is the number of edges in the longest path from the root to a leaf.) This special traversal enables us to find isomorphic subtrees by using bucket sort [1].

Each node has two auxiliary bits ($\text{leftbit}$ and $\text{rightbit}$) to indicate whether the subtrees of the node are found to be isomorphic to some other subtrees in the tree. We have to consider only nodes whose both bits are set. We maintain a list $L$ of such nodes. Knowing that all leaves are isomorphic to each other, we initially label all leaves by 1 and place them in $L$. $L$ is traversed and for each node $a$ in $L$ the following operations are performed:

1. if $a$ is the left (resp. right) child of its parent $b$, we set $\text{leftbit}$ (resp. $\text{rightbit}$) in $b$.
2. if $b$'s both bits are set, $b$ is placed in the new list of set nodes.

Operations 1 and 2 above determine our traversal order. We can terminate the algorithm when the new list of set nodes contains less than two nodes. Naturally, the actual isomorphism test must also be done during the traversal.

Fig. 1. The tree to the left is supposed to have label 2 and the one to the right label 4.