NOTES ON DISTANCE-BASED CODING METHODS FOR BINARY TREES

Juha Lehikoinen and Erkki Mäkinen

DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF TAMPERE

REPORT A-1994-2

UNIVERSITY OF TAMPERE
DEPARTMENT OF COMPUTER SCIENCE
SERIES OF PUBLICATIONS A
A-1994-2, FEBRUARY 1994

NOTES ON DISTANCE-BASED CODING METHODS FOR BINARY TREES

Juha Lehikoinen and Erkki Mäkinen

University of Tampere
Department of Computer Science
P.O. Box 607
FIN-33101 Tampere, Finland

ISBN 951-44-3498-6
ISSN 0783-6910

NOTES ON DISTANCE-BASED CODING METHODS FOR BINARY TREES

Juha Lehikoinen and Erkki Mäkinen#
University of Tampere, Department of Computer Science
P.O. Box 607, FIN-33101 Tampere, Finland

Abstract. This note introduces a new coding method for binary trees in which the code item corresponding to a node equals node's distance from the left arm of the tree according to the metrics defined by the tree. The new coding method is compared with previously known methods. Moreover, we establish a connection between distance-based methods and a recently introduced method by Johnsen.

CR Categories: E.1, F.2.2, G.2.1.

Key Words and Phrases: binary tree, coding, decoding.

1. Introduction

Various algorithms for creating and manipulating trees derive advantage from coding methods which replace the shape of trees by unique codewords. Numerous such coding methods are presented in the literature [3]. Most of these methods apply to binary trees. We shall also restrict ourselves to binary trees only.

An efficient coding method is obtained when the code item corresponding to a node equals the distance from the left arm of the tree. (*Left arm* is the path from the root to the least node in symmetric order.) Reading the code items in symmetric order then gives the codeword [2]. These codewords are referred to as *ld-codewords* (where *ld* stands for "left distance").

The ld-codeword of the tree shown in Figure 1 is (0,0,1,1,2,1,2). The code item corresponding to the least node in symmetric order is always 0. If x_i is a code item then the next code item x_{i+1} is from the interval $[0..x_i+1]$. This simple characterization implies very efficient generation of valid codewords [4].

The code item corresponding to a node equals the number of right children in the path from a node in the left arm to the node in question. Hence, the code item gives a kind of bird's-eye view distance from the left arm. A unique coding method is also obtained if this bird's-eye view is replaced by the actual distance from a node in the left arm counting both the left and right children.

To whom all correspondence should be addressed (em@cs.uta.fi).

The codeword so obtained is called the *ad-codeword*. The ad-codeword of the tree shown in Figure 1 is (0,0,3,2,3,1,2).

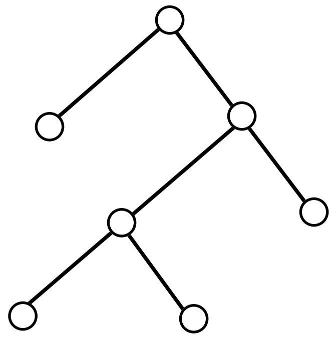

Figure 1. A sample tree.

This note discusses the properties of ad-codewords and other distance-based codewords. In chapter 2 we give algorithms for coding binary trees and decoding codewords. In chapter 3 we consider the codewords from a rotational point of view. Chapter 4 concerns the relationship between ad-codewords and so called d-codewords. Finally in chapter 5, we establish a connection between distance-based methods and a recently introduced coding method by Johnsen [1].

2. Coding and decoding

Given a binary tree, it is straightforward to find the corresponding ad-codeword. We simply traverse the tree in symmetric order and record the actual distances from the left arm.

Similarly, it is possible to decode a valid codeword back to the corresponding binary tree in linear time. This algorithm scans the codeword from left to right and performs the operations to be described below. If x_i is a code item then the corresponding node in the tree is denoted by $c(x_i)$.

For each code item $x_i = 0$, $c(x_i)$ becomes the root of the subtree so far constructed. The previous subtree becomes the left subtree of $c(x_i)$. The new root becomes the new *current node* from which the computation continues.

If the code item x_{i+1} is of the form $x_i + 1$, then $c(x_{i+1})$ will be the right child of the current node. The right child to be created is the new current node.

Third case appears when $x_{i+1} > x_i + 1$. Then $c(x_{i+1})$ is the least node in symmetric order in the right subtree of $c(x_i)$. In order to place $c(x_{i+1})$ to its correct position with respect to $c(x_i)$, we must insert additional nodes between $c(x_i)$ and $c(x_{i+1})$; $c(x_{i+1})$ is the new current node. The number of additional nodes to be inserted depends on the difference $x_{i+1} - x_i$. This situation is demonstrated in Figure 2.

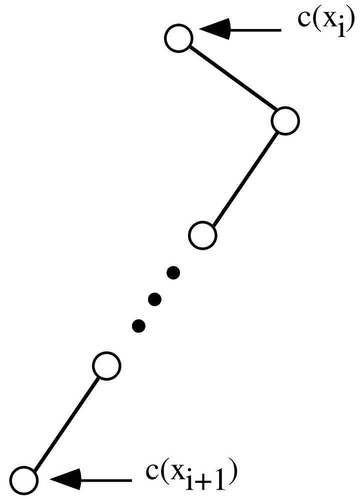

Figure 2. The case $x_{i+1} > x_i + 1$.

The additional nodes inserted correspond to code items not yet scanned. We need a pushdown store to which an element is pushed for each additional node. A pushdown store element contains a pointer to the corresponding node in the tree. The pushdown store elements added are so ordered that the element corresponding to the parent of $c(x_{i+1})$ is the new top element and the one corresponding to the right child of $c(x_i)$ is the lowest of the elements added.

The fourth case appears, when $0 < x_{i+1} < x_i$. We know that $c(x_{i+1})$ cannot be in the subtree rooted at $c(x_i)$. The top element of the pushdown store now contains a pointer to the new current node. The top element is popped from the pushdown store. Hence, in this case no node is inserted to the tree because $c(x_{i+1})$ is already present.

We have now considered all the possible cases. Indeed, $x_i = x_{i+1}$ is possible only when $x_i = x_{i+1} = 0$.

Consider now, as an example, the construction of the tree shown in Figure 1 from its codeword (0,0,3,2,3,1,2). The code items are denoted by $x_1, x_2, ..., x_7$. When x_1, x_2 and x_3 have been scanned the tree and the pushdown store are as shown in Figure 3.

Since $0 < x_4 < x_3$, we pop the top element from the pushdown store. The node pointed by the popped element ($c(x_4)$, the parent of $c(x_3)$) is the new current node. Next, we have $x_5 = x_4 + 1$, and $c(x_5)$ is the right child of $c(x_4)$. Since $0 < x_6 < x_5$, we pop the top element from the pushdown store, and the new current node $c(x_6)$ will be the parent of $c(x_4)$. The last code item x_7 is of the form $x_6 + 1$ and hence, $c(x_7)$ is the right child of $c(x_6)$. This completes the construction, and we have obtained the tree shown in Figure 1.

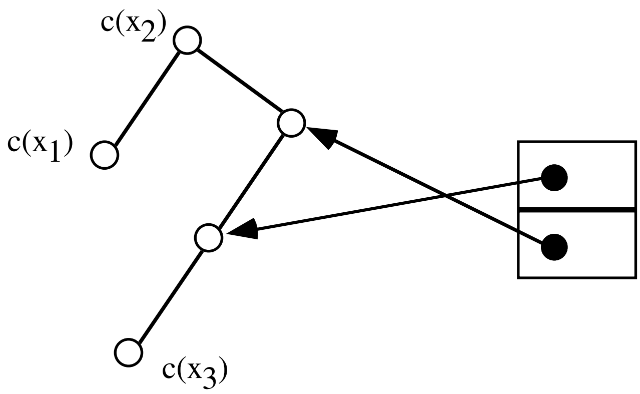

Figure 3. The sample construction after scanning x_1, x_2 and x_3.

Contrary to the case of valid ld-codewords, there is no simple characterization for valid ad-codewords.

3. A rotational interpretation

Ld-codewords were originally invented as a variation of Zerling's codewords [6] which in turn are related to rotations made in the binary tree in question. In Zerling's method the codeword corresponding to a tree on n nodes is obtained as follows. We make left rotations on the root of the tree as long as possible, i.e. until the greatest node in symmetric order is on the root. The code item related to the greatest node in symmetric order is the number of rotations done. The same procedure is then repeated in the left subtree; i.e. the greatest node is deleted and left rotations are done on the new root. This continues until the code item related to the second least node in symmetric order is set to have value 0 or 1 depending on whether we must rotate when there are two nodes left. The code item related to the least node is always 0.

Ld-codewords can be considered as a variation of Zerling's codewords by noting that we can rotate the edge joining the current greatest node in symmetric order and its parent instead the edge joining the current root and its right child. By performing the rotations as in the ld-method, the distance of the smaller nodes in symmetric order from the left arm is kept unchanged. This guarantees that the ld-codewords indeed have the geometric interpretation of giving the bird's-eye view distance from the left arm.

We can also give a rotational interpretation for the ad-codewords. Consider again the sample tree shown in Figure 1. Refer to the third node in symmetric order as x. By rotating three times the edge joining x and its parent moves x to the left arm. (We must perform two right rotations and one left rotation.) Figure 4 shows the tree after these rotations.

The distances of the greater nodes in symmetric order from the left list are kept unchanged. Hence, as in the ld-method, our new method gives the rotation distances of the nodes from the left list. However, in the present method we handle the nodes starting from the least node in symmetric order and allow both left and right rotations.

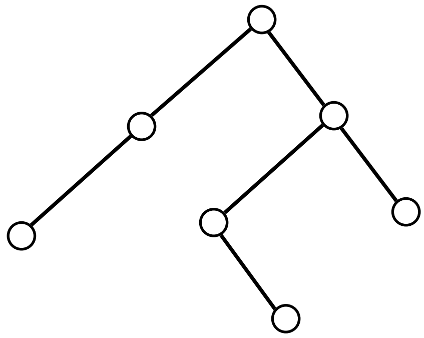

Figure 4. The sample tree of Figure 1 after certain rotations.

4. Ad-codewords vs. d-codewords

So far, we have considered the relationship between ad-codewords and ld-codewords. There is also another distance-based coding system closely related to ad-codewords. Namely, instead of counting the distances from the left arm we could as well count the distances from the root. When counting distances from the root it is more natural to use preorder than symmetric order. For the tree shown in Figure 1, this coding method gives the codeword (0,1,1,2,3,3,2). (Obviously, the leading 0 is redundant.) The codewords so obtained are referred to as *d-codewords*. Note that for the right subtree of the root, d-codewords and ad-codewords coincide (provided that the code items are recorded in the same order).

We can also record the code items in preorder when using ad-codewords. If so, we must count the distances from the right arm instead of the left arm. (*Right arm* is defined analogously to left arm.) After these changes the ad-codeword of the tree shown in Figure 1 is (0,1,0,1,2,2,0). Now 0 in the codeword corresponds to a node in the right arm. We can divide the original tree into subtrees by selecting the nodes in the right arm as roots. Each right subtree is empty and the left subtree is kept unchanged. Figure 5 show the result of this procedure in the case of the tree in Figure 1.

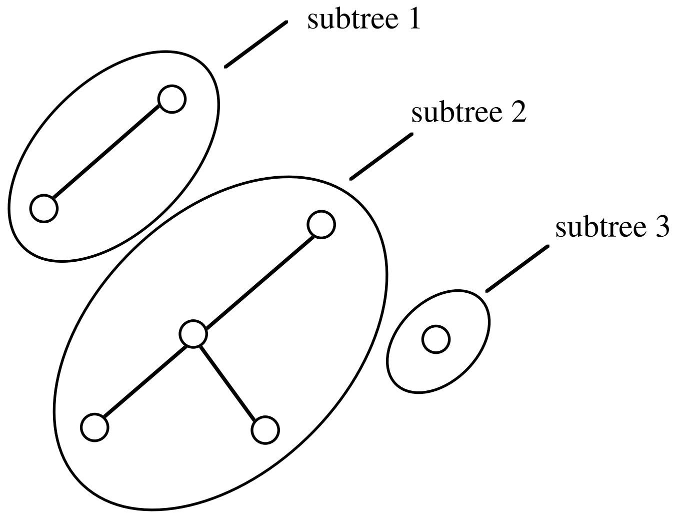

Figure 5. The result of the division procedure in the case of the tree in Figure 1.

The procedure divides the tree into subtrees; the corresponding subcodewords equals the d-codewords of the subtrees. In the above example we have subcodewords (0,1), (0,1,2,2), and (0). These are the d-codewords of the subtrees shown in Figure 5. A special case appears when the original tree has empty right subtree. In that case ad-codewords and d-codewords coincide.

5. Graftings are distance-based

Johnsen has shown [1] that a unique coding system for binary trees can be established by inserting the nodes into the tree in preorder and recording the positions to which the insertions are made. As an example consider the tree in Figure 6. There are three possibilities to insert the next node in preorder. These are numbered from left to right by 0, 1, and 2. So far, the insertions are done into the positions 0, 2, and 0. (Excluding the zero corresponding to the root.) The next insertion will augment the codeword to be (0,2,0,0), (0,2,0,1), or (0,2,0,2) depending on our choice for the place of the next insertion. The operation of inserting a node in preorder is called *grafting* in [1].

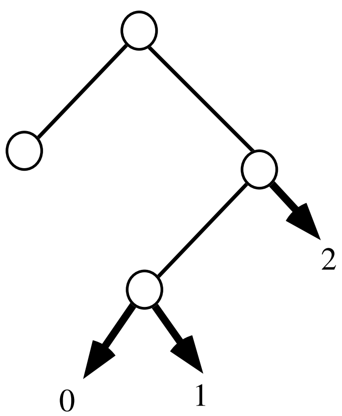

Figure 6. The three possibilities to insert the next node in preorder.

It is noted in [5] that the set of codewords according to Johnsen's method coincide with that of Zerling's method. We show that there exists even a closer resemblance between graftings and distance-based methods.

Instead of numbering the possible positions to insert the next node from left to right we can as well number them from right to left. The codeword corresponding to the tree shown in Figure 6 is now (1,0,1). (Again, the code item corresponding to the root is omitted.) The next insertion possibilities are as shown in Figure 7.

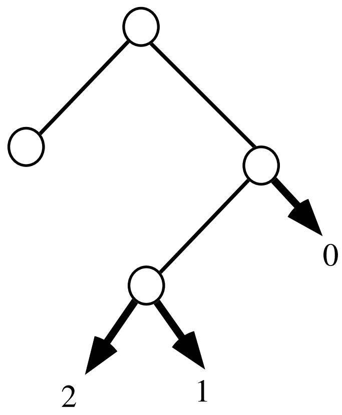

Figure 7. Renumbering the insertion possibilities.

Hence, after changing the numbering of the insertion possibilities, the code items obtained coincide with the bird's-eye view distances from the right arm.

References

[1] B. Johnsen, Generating binary trees with uniform probability. *BIT* **31** (1991), 15-31.

[2] E. Mäkinen, Left distance binary tree representations. *BIT* **27** (1987), 163-169.

[3] E. Mäkinen, A survey on binary tree codings. *Comput. J.* **34** (1991) 438-443.

[4] E. Mäkinen, Efficient generation of rotational-admissible codewords for binary trees. *Comput. J.* **34** (1991), 379.

[5] E. Mäkinen, A note on graftings, rotations, and distances in binary trees. *Bull. EATCS* **46** (1992), 146-148.

[6] D. Zerling, Generating binary trees using rotations. *J. ACM* **32** (1985), 694-701.