Binary Codes for Fast Determination of Ancestor-Descendant Relationships in Trees

Natarajan Meghanathan  Sanjeev Baskiyar, Ph.D
Department of Computer Science and Software Engineering
Auburn University, Auburn, AL 36830
e-mail: baskiyar@eng.auburn.edu

Abstract

This paper proposes a new coding scheme and an algorithm to determine ancestor-descendant relationship among nodes in a tree without tree traversal. Each node in a tree is assigned a unique binary code. The algorithm and the binary coding approach were tested with trees of different heights and widths. The algorithm is of $O(1)$ complexity versus $O(d)$ for tree traversal, where $d$ is the depth of the tree. The algorithm can be used to determine, in $O(1)$ time, the superclass-subclass relationship either at compile-time or at run-time in an object-oriented (OO) programming environment.

1 Introduction

Ancestor-descendant relationship between any two nodes in a tree can be determined by traversing the tree from the descendant node to the root node while checking for the ancestor node. The complexity of such an algorithm is $O(d)$ where $d$ is the depth of the tree ($d \geq \log_2 n$, where $n$ is the number of nodes in the tree). Alternatively, one can also determine the relationship by preorder, in-order or post-order traversal algorithms with complexity $O(n)$. This
paper proposes a coding scheme, Binary Codes for Ancestor-Descendant (BCAD) relationships and an algorithm, Determination of Ancestor-Descendant (DAD) relationships to determine the ancestor-descendant relationship in any tree, by assigning a binary code to each node of the tree. The algorithm DAD can be used to determine, in $O(1)$ time, the superclass-subclass relationship, either at compile-time or at run-time in an object-oriented (OO) programming environment.

The organization of this paper is as follows. In Section 2, we outline the past work on different coding schemes used for analysis of trees. In Section 3, we present the BCAD scheme, in Section 4, we propose the algorithm DAD that uses the binary codes proposed in Section 3. We demonstrate a run of algorithm DAD on an example tree and discuss the complexity, storage requirements, advantages and drawbacks of this technique in Section 4. In Section 5, we present suggestions for future work.

2 Background

A decimal notation satisfies many mathematical properties and is a useful tool in the analysis of trees. Aoe and Jun-ichi [1] presented a practical method that compresses the decimal codes of tree nodes while maintaining the fast determination of relations (eg., ancestor, descendant, sibling, etc.). Nodes that do not have grandchildren have been called kernel nodes. If $n(m)$ represents the number of the total (kernel) nodes, a decimal code can be encoded in constant time, the worst-case time complexity of compressing the decimal codes is $O(n + m^2)$ and the size of the data structure is proportional to $m$. The method of determining relations between nodes using the compressed decimal codes has not been addressed although using hierarchical semantic language primitives such relations can be determined.
A Gray code represents each number in the sequence of integers $0...2^{n-1}$ as a binary string of length $n$ in an order such that adjacent integers have representations that differ in only one bit position. Xiang, et.al [6] proposed an algorithm to generate Gray codes for $k$-ary trees with $n$ internal nodes ($n \geq 2$ and $k > 3$) in $2^{n-1}$ different ways. (The internal nodes [3] of a tree represent the points of divergence where two different branches of evolution arise.) However, determining ancestor-descendant relationship between any two nodes in a tree has not been addressed.

Gupta’s [5] coding scheme codes a regular binary tree with $n$ nodes by labeling the left branches by 0s and the right branches by 1s and then traversing the branches in pre-order. Again, determining the inheritance relationship between any two nodes in a tree has not been addressed.

Tunstall codes, used extensively in data compression, are an example of variable-to-fixed length encoding scheme, that map a dictionary of variable length strings of source outputs to the set of code words of a given length. In a variable-length encoding scheme such as Tunstall codes, certain code words can form a prefix for other code words. Such codes are called prefix codes. Jun et.al [7] studied the properties of Tunstall codes and established relationships between the Tunstall codes and their extension numbers (each Tunstall code is pre-coded by recursively replacing the code with a prefix code and its extension character from the data dictionary). But, methods to apply Tunstall codes to determine ancestor-descendant relationships in a tree have not been discussed. Castelli et.al [2], introduced a notion of divisibility and primality on $k$-ary trees and found a relation between indecomposable prefix codes and prime trees, which are a type of combinatorial networks. The indecomposable prefix codes are not suitable for determining ancestor-descendant relationships in a tree.
A Prufer code of a labeled free tree (a free tree is a connected acyclic undirected graph) with \( n \) nodes is a sequence of length \( n-2 \) codes. The sequence is constructed as follows: for \( i \) ranging from 1 to \( n-2 \) the label of the neighbor of the smallest remaining leaf is inserted into the \( i^{th} \) position of the sequence and then the leaf is deleted. Prufer codes provide an alternative to the usual representation of trees. Greenlaw and Petreschi [4], presented an optimal \( O(\log n) \) time algorithm on \( n/\log n \) EREW-PRAM (Exclusive Read Exclusive Write – Parallel Random Access Memory) processors for determining the Prufer code of an \( n \)-node labeled chain and an \( O(\log n) \) time algorithm on \( n \) EREW-PRAM processors for constructing the Prufer code of an \( n \)-node labeled free tree. Prufer codes find extensive use in parallel algorithms but are not applicable in the context of determining ancestor-descendant relationships in a tree.

3 The BCAD scheme

We propose binary codes of variable length, which depend upon the position of the node in the tree. The code for any node consists of a prefix and a suffix part. The prefix part is the code for the node’s parent. Each sibling is assigned a unique suffix. For example, if there are three siblings, the suffixes assigned to them are 00, 01 and 10. If there are \( S \) siblings of a node, the number of bits used for the suffix is \( \lceil \log_2 S \rceil \). The root node is assigned the code 0. It is obvious that the code for each node will be unique in the above coding scheme.

The number of bits used to code any node, \( n_i \), is

\[
N(n_i) = N(P(n_i)) + \lceil \log_2 S(n_i) \rceil
\]  

(i)
where,

\[ P(n_i) \] is the parent of \( n_i \).

\[ S(n_i) \] is the number of siblings of \( n_i \).

\[ N(n_i) = 1, \] if \( n_i \) is the root node.

Let \( d \) represent the depth of a tree with the root node at \( d = 0 \). The number of bits used for any node \( n_i \) at a depth \( d \) is

\[
N_d = \max_{\forall n_i \in Z} \left[ N(n_i) \right]
\]

where \( Z \) is the set of all nodes at depth \( d \).

**Figure 1.** Example tree with codes to illustrate \( BCAD \)
Hence the total number of bits required to represent all nodes in a tree of height $h$ is

$$
\sum_{k=0}^{h-1} N_k \quad (iii)
$$

### 3.1 Implementation

Let $w$ be the number of bits in a word of a computer. Let $Q = \max_{n_i \in T} N(n_i)$ be the maximum number of bits needed to code any node of tree $T$ using BCAD. We use $W$ bits to code each node of $T$ where $W = kw; k$ is the smallest integer such that $W \geq Q$. For nodes whose code length is shorter than $W$, we prefix the code with $1$s to make them exactly $W$ bits. The total number of bits used to represent a tree of $n$ nodes is $nW$. Let $p_i$ represents the number of $1$s prefixed to the unextended code of length $q_i$, of $n_i$ to make it exactly $W$ bits. Let $\|n_i\|$ represents the total length of the complete code for $n_i$ (i.e., $W$ bits). Figure 2 shows the complete code for node $n_{19}$ of Figure 1.

\[\begin{array}{cccccccccccc}
1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\
\end{array}\]

\[p_{19}\]

\[q_{19}\]

$W = \|n_{19}\|

\textbf{Figure 2.} Complete code for node $n_{19}$ of Figure 1.
4 Algorithm DAD

The algorithm, DAD, to determine ancestor-descendant relationship using the BCAD codes is shown in Figure 3. It can determine whether a node $n_i$ in a tree is a descendant of node $n_j$ or same as $n_j$ (hereafter referred to as $n_i \subseteq n_j$). All the nodes in a tree are assigned a unique binary code. When two nodes $n_i$ and $n_j$ are tested for the relation $n_i \subseteq n_j$, the $p_j^{th}$ to $p_j+q_j-1^{th}$ bits of $n_j$ are compared against the $p_i^{th}$ to $p_i+q_i-1^{th}$ bits of $n_i$. If the $q_j$ bits in the codes of $n_i$ and $n_j$ compared in Step 3 are identical, $n_i$ is a descendant of $n_j$ otherwise not.

---

**Boolean DAD**

Inputs: The complete codes for nodes $n_i$ and $n_j$ of tree $T$.

1) Count number of preceding $1$s, $p_j$, in the code of $n_j$. Let $q_j = |n_j| - p_j$.

2) Count number of preceding $1$s, $p_i$, in the code of $n_i$.

3) If $MID(p_i, q_i, n_i) = MID(p_j, q_j, n_j)$

    return **TRUE**

else

    return **FALSE**

end DAD

The function $MID(p_i, q_i, n_i)$ returns $p_i^{th}$ to $p_i+q_i-1^{th}$ bits in the complete code for $n_i$. 

---
The values of $p_i$ and $p_j$ can be determined in $O(1)$ time. Also, Step 3 of the algorithm, comparing $MID(p_i,q_j, n_i)$ and $MID(p_j,q_j, n_j)$ can be executed at constant time. Therefore, the overall complexity of the algorithm is $O(1)$.

4.1 Example runs

We explain runs of the DAD algorithm. Consider testing whether node $n_{20}$ in Figure 1 is a descendant of node $n_6$. Steps 1 and 2 of the algorithm give $p_{20} = 7$ and $p_6 = 12$ and $q_6 = |n_6| - p_6 = 4$. Step 3 of the algorithm retrieves and compares the $p_{20}^{th}$ to $p_{20} + q_6 - 1^{th}$ and $p_6^{th}$ to $p_6 + q_6 - 1^{th}$ bits in the codes for nodes $n_{20}$ and $n_6$ respectively. Since the $q_6$ bits compared in this step are the same, we conclude that $n_{20}$ is a descendant of $n_6$. Next, consider testing whether node $n_{15}$ in Figure 1 is a descendant of node $n_9$. Steps 1 and 2 of the algorithm give $p_9 = 10$ and $p_{15} = 8$ and $q_9 = |n_9| - p_9 = 6$. Step 3 of the algorithm retrieves and compares the $p_9^{th}$ to $p_9 + q_9 - 1^{th}$ and $p_{15}^{th}$ to $p_{15} + q_9 - 1^{th}$ bits in the codes for nodes $n_9$ and $n_{15}$ respectively. Since the $q_9$ bits compared in this step are not the same, we conclude that $n_{15}$ is not a descendant of $n_9$.

4.2 Storage requirements

Consider a full $S$-ary tree (each node having $S$ children). The number of bits in the code for a node increases by $\lceil \log_2 S \rceil$ as one traverses from a level to the next lower level. Hence the number of bits required to represent a node at depth $d$ is $1 + d^* \lceil \log_2 S \rceil$ for $s > 1$ and is $1 + d$ for $s = 1$. For a full binary tree ($S = 2$) with $d = 3$, the maximum number of bits required to represent
any node is 4. For a chain \((S=1)\) with \(d = 6\), the maximum number of bits required to represent any node is 7. For a tree with \(d = 1\) and \(S = 7\), the maximum number of bits required to represent any node is 4.

<table>
<thead>
<tr>
<th>Depth of tree, (d)</th>
<th>Total number of nodes in the tree</th>
<th>Number of children per node, (S) (S-ary tree)</th>
<th>Maximum number of bits for any node (1+d\lceil \log_2 S \rceil)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>51</td>
<td>50</td>
<td>7</td>
</tr>
<tr>
<td>50</td>
<td>51</td>
<td>1</td>
<td>51</td>
</tr>
<tr>
<td>50</td>
<td>(2^{51}-1)</td>
<td>2</td>
<td>51</td>
</tr>
<tr>
<td>50</td>
<td>(3^{51}-1)</td>
<td>3</td>
<td>101</td>
</tr>
<tr>
<td>50</td>
<td>(4^{51}-1)</td>
<td>4</td>
<td>101</td>
</tr>
<tr>
<td>25</td>
<td>(8^{26}-1)</td>
<td>8</td>
<td>76</td>
</tr>
<tr>
<td>25</td>
<td>(20^{25}-1)</td>
<td>20</td>
<td>126</td>
</tr>
</tbody>
</table>

Table 1. Maximum number of bits for any node in an \(S\)-ary tree

Table 1 shows the maximum number of bits for any node for different values for an \(S\)-ary tree of depth \(d\). From the table, we observe that the complexity of the code size increases in \(O(\lceil \log_2 S \rceil)\). We can also observe that using double words in 64-bit computers, on trees with up to \(20^{26}-1\) nodes (20-ary, depth < 25), one can run algorithm DAD efficiently employing very few memory accesses.
4.3 Application in the Object-Oriented Paradigm

Inheritance, one of the key characteristics of the object-oriented (OO) paradigm, is a mechanism by which an object acquires the properties of another object. Objects of a subclass can invoke methods of their parent classes. Polymorphism is another feature of the OO paradigm that allows a single interface of methods to be used for a general set of actions. Compile-time and run-time checks are needed to ensure correct inheritance relationship between any two classes involved. The binary coding approach, in this paper, can be applied to determine, in $O(1)$ time, the superclass-subclass relationship in a class hierarchy tree, both at compile-time and at run-time for OO programs.

Languages such as C++ support multiple inheritance whereas Java and Smalltalk support single inheritance. The BCAD coding scheme is applicable only to single inheritance. Also, we observe that we round off $\log_2 S$ to get the number of bits required to code a descendant node where $S$ is the number of children of an ancestor node. This seems to be an unavoidable storage overhead.

5 Conclusions

We have proposed a binary coding scheme BCAD and an algorithm DAD to find ancestor-descendant relationship between any two nodes in a tree in $O(1)$ time. The algorithm DAD is of $O(1)$ complexity versus $O(d)$ for tree traversal, where $d$ is the depth of the tree, and can run with very few memory accesses for many reasonably sized trees. The algorithm can be used to determine either at compile-time or at run-time the superclass – subclass relationships in
an OO environment where single inheritance is employed. Future work should address developing similar codes where multiple inheritance is allowed.

References


