Binary Codes for Fast Determination of Ancestor-Descendant Relationship in Trees and Directed A-cyclic Graphs

Sanjeev Baskiyar, Ph.D. and N. Meghanthan
Department of Computer Science and Software Engineering
Auburn University, Auburn, AL 36849
e-mail: baskiyar@eng.auburn.edu

Abstract

This paper develops simple binary codes, called *Binary Ancestry (BA)* codes, for trees using which ancestor-descendant relationships among any two nodes of a tree can be determined without tree traversal. The *BA* coding technique assigns unique binary codes to each node of a tree. A procedure, *IsAncestor*, that uses *BA* codes to determine the relationships, yielded correct results in sample trees of different heights and widths. *IsAncestor* is of *O(1)* complexity versus *O(d)* required to determine the relationships via traversal of any tree of height *d*. *IsAncestor* may be used, either at compile-time or run-time, to determine superclass-subclass relationships, needed to perform method resolutions in single-inheritance object-oriented environments. The *BA* codes and *IsAncestor* have been extended for environments where multiple-inheritance is allowed.

Keywords: Tree, Binary codes, Ancestor-Descendant, Object-Oriented, Inheritance

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 height of the tree ($d \geq log_2n$, where *n* is the number of nodes in the tree). (Alternatively, one can also determine the relationships by pre-order, in-order or post-order traversal algorithms with complexity *O(n)*.) The need for a coding technique and an algorithm pair to quickly determine ancestor descendant relationships for use in object-oriented programming was identified in [2]. This paper develops a coding technique, called *Binary Ancestry* (*BA*) codes and an algorithm *IsAncestor* for determining ancestor descendant relationships. The *BA* coding technique assigns a binary code to each node of the tree. The algorithm *IsAncestor* is of *O(1)* complexity, it can be used to determine the relationship among classes, either at compile-time or at run-time in an object-oriented programming environment.

The organization of this paper is as follows. In Section 2, we survey past work on codes for analysis of trees. In Section 3, we present (a) *BA* codes and discuss its length and (b) the algorithm *IsAncestor* and its complexity, and its run on sample trees. In Section 4, we discuss applications of *BA* codes. In Section 5, we extend *BA* codes to handle multiple-inheritance. Section 6 has some concluding remarks.

2 Background

A method to compress *decimal* codes of tree nodes, while retaining the fast determination of relations (e.g. ancestor, descendant, sibling, etc.), appears in [1]. Nodes that do not have grandchildren are called *kernel* nodes. If *n(m)* represents the number of the *total (kernel)* nodes, a decimal code can be compressed with worst-case complexity of $O(n+m^2)$ and space complexity *O(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.

Gray codes represent numbers in the set of integers $0...2^{n-1}$ as binary strings of length *n* such that adjacent integers have representations which differ in only a single bit position. Xiang, et.al [8] proposed an algorithm to generate Gray codes for *s*-ary trees with *n* internal nodes ($n\geq 2$, $s>3$) in 2^{n-1} different ways. (The internal nodes [4] of a tree are nodes where branches sprout.) However, determining ancestor descendant relationship between any two nodes in a tree has not been addressed.

Gupta's [6] coding scheme codes a binary tree by labeling the left branches by *0*s and the right branches by *1*s in a pre-order traversal. 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 mapping scheme. In a variable-length encoding scheme such as Tunstall codes, certain code words can form prefixes of other codes. Such codes are called prefix codes. Jun et.al [7] established relationships between 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. A notion of divisibility and primality on *k*-ary trees was introduced in [2] and a relation between indecomposable prefix codes and prime trees was established. The indecomposable prefix codes have not been shown to determine ancestor-descendant relationships in a tree.

A Prufer code of a labeled free tree (a connected a-cyclic undirected graph) with *n* nodes is a sequence of length *n*-2. 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. Greenlaw and Petreschi [5], presented an optimal *O(logn)* algorithm on *n/logn* 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(logn)* 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 have not been used for fast determination of ancestor-descendant relationships in a tree.

3 Ancestry codes

We propose variable length binary codes whose code length depends 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 inherited from 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 *S(i)* be the number of siblings of any node n_i, the number of bits used for the suffix is $\lceil \log_2 S(i) \rceil$. The root node is assigned the code *0*. The suffix establishes uniqueness among siblings and the prefix among nodes other than siblings. Therefore, the code for any node is unique among siblings.

The length (in bits) of code x_i of any node, n_i, is

$$|x_i| = |x_{P(i)}| + \lceil \log_2 S(i) \rceil \text{ if } S(i) > 1$$

$$= |x_{P(i)}| + 1, \text{ if } S(i) = 1 \qquad (i)$$

where,

$n_{p(i)}$ is the parent of n_i

$S(i)$ is the number of siblings of n_i, including n_i

$|x_i| = 1$, if n_i is root

Maximum number of bits used to code any node of a tree, T, is $Q = max_{i \in L}\, x_i$, where L is the set of leaf nodes in T.

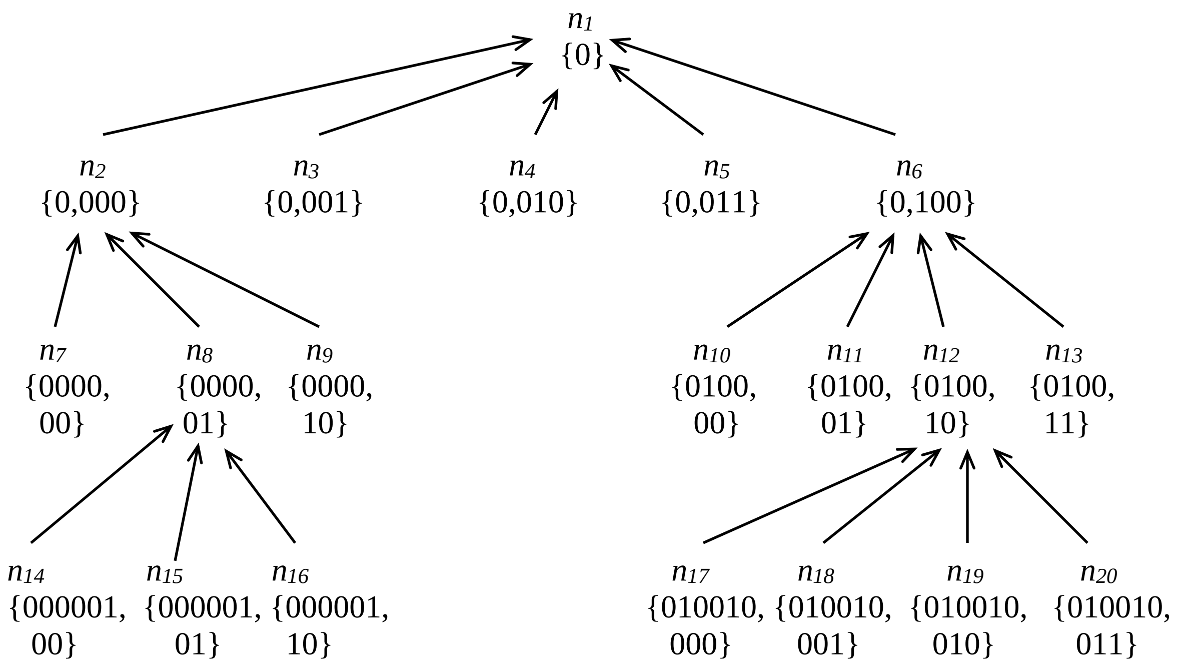

Figure 1. A tree, T_1, showing ancestry codes

Consider a full s-ary tree (each node having $s>1$ children). At depth $d = 0$, code length = 1. From equation *(i)* the code length of a node at depth $d=1$ is $1+ \lceil log_2 s \rceil$, at depth $d = 2$ is $1 + \lceil log_2 s \rceil + \lceil log_2 s \rceil = 1+2 \lceil log_2 s \rceil$. Let at depth, g, code-length = $1 + g \lceil log_2 s \rceil$. Using eqn (i) at depth $g + 1$, code-length = $1 + g \lceil log_2 s \rceil + \lceil log_2 s \rceil = 1 + (g+1) \lceil log_2 s \rceil$. By induction for a node at any depth d, code-length = $1+ d^* \lceil log_2 s \rceil$, for $s > 1$. *For* $s=1$, code length = $1+d$. Table 1 shows the code length required for different s-ary trees with different depths, d. We observe that the code length of a node scales linearly with the height of the tree and logarithmically with the number of siblings.

Height of tree, d	Children per node, s (s-ary tree)	No. of nodes in the tree, $s^{d+1} -1$, *for* $s >1$ *and* $1+d$ *for* $s =1$	Code length $1+d^* \lceil log_2 s \rceil$, *for* $s>1$ *and* $1 +d$ *for* $s =1$
1	*50*	*51*	*7*

25	*1 (chain)*	*26*	*26*
25	*2 (binary)*	2^{26}-1	*26*
50	*3*	3^{51}-1	*101*
50	*4*	4^{51}-1	*101*
25	*8*	8^{26}-1	*76*
25	*20*	20^{26}-1	*126*

Table 1. Code length required for different *s*-ary trees with different heights, *d*.

3.1 Implementation

We use *W* bits to code each node of any tree, where $W = kw$, *k* is the smallest integer such that $W \geq Q$ and *w* is the number of bits in a word of a computer. The most significant bits of codes whose length is shorter than *W*, are padded with *1*s to make the code exactly *W* bits. Let p_i represent the number of *1*s prefixed to the unextended code of length q_i, of n_i to make it exactly $W= p_i+q_i$ bits. Figure 2 shows the complete code for node n_{19} of T_1.

p_{19}							q_{19}								
1	1	1	1	1	1	1	0	1	0	0	1	0	0	1	0

Figure 2. Complete code x_{19} for node n_{19} tree T_1

We observe that, using double words, in *64*-bit computers, *BA* codes can code trees of up to 20^{26}-*1* nodes (*20*-ary, height *25*). Therefore, *IsAncestor*, discussed next, can determine ancestry employing very few memory accesses.

3.2 Algorithm *IsAncestor*

The algorithm, *IsAncestor*, outlined in Figure 3, determines whether a node n_i in a tree is a descendant of n_j or same as n_j (hereafter referred to as $n_i \subseteq n_j$). Let x_i represent the complete code of n_i consisting of *W* bits. Line *8* compares the p_jth to p_j+q_j-1th bits of x_j against the p_ith to p_i+q_j-1th bits of x_i. If the q_j bits compared are identical, $n_i \subseteq n_j$ otherwise not.

```
1    int IsAncestor(int $x_i$, int $x_j$){
2    //Inputs: The complete codes of nodes $n_i$ and $n_j$ of tree $T$.
3    //Outputs: Returns 1 if $n_i \subseteq n_j$, otherwise 0.
4    //Uses: Function Mid($p_i$,$q_i$, $n_i$) which returns $p_i$th to $p_i+q_i-1$th bits in $x_i$. See Appendix.
5        int    $p_i$, $p_j$, $q_j$;
6        Count preceding 1s, $p_j$ in $x_j$. Let $q_j$ = $/x_j/-p_j$.
7        Count preceding 1s, $p_i$ in $x_i$.
8        If ($q_j > x_i - p_i$) return 0. //code length of $n_i$ not shorter than $n_j$
9        If Mid ($p_i$,$q_j$, $n_i$) = = Mid($p_j$,$q_j$, $n_j$)
10           return 1
11       else
12           return 0
13   } //end
```

Figure 3. Procedure *IsAncestor*

Lines $6,7$ and 8 are computed in $O(1)$ time. Also, line 9, comparing $Mid(p_i,q_j,n_i)$ and $Mid(p_j,q_j,n_j)$ is executed in constant time, as per Appendix.

Here we consider sample runs of *IsAncestor*. For tree T_1 consider checking whether $n_{20} \subseteq n_6$ using *IsAncestor*. *Lines 6 and 7* give $p_{20} = 7$, $p_6 = 12$, $q_6 = |x_6| - p_6 = 4$. In line 9, since the p_{20}th to $p_{20}+q_6-1$th and p_6th to p_6+q_6-1th bits of x_{20} and x_6 are identical, $n_{20} \subseteq n_6$. Next, consider checking whether, $n_{15} \subseteq n_9$. Lines *6 and 7* of *IsAncestor* give $p_9 = 10$, $p_{15} = 8$, $q_9 = |x_9| - p_9 = 6$. In line 9, since p_9th to p_9+q_9-1th and p_{15}th to $p_{15}+q_9-1$th bits of x_9 and x_{15} are different, $n_{15} \not\subseteq n_9$.

4 Applications

The *BA* codes can be employed to determine whether a class is a subclass of another in any hierarchical classification. Below we discuss a few specific examples.

The *BA* codes can speed method resolution prior to binding calls in object-oriented programs. In object-oriented programming, in the absence of multiple-inheritance, the class-subclass relationships are in the form a tree. Polymorphism allows multiple definitions of the same method name whereas inheritance allows an object of a class to invoke methods in its parent classes. Therefore, to correctly resolve calls to methods, a check is run to determine whether the class of the object is a subclass of that of the invoked method. The check is performed at compile-time or run-time depending on whether the method resolution is static or dynamic. As an example, if the nodes n_i in tree T_1 represent classes, before executing a call $o_8.f()$, where method $f()$ is defined in n_2 and object o_8 is an instance of n_8, it must be tested whether $n_8 \subseteq n_2$. Languages that support dynamic bindings are Smalltalk, *C*++, Java and *C*#.

BA codes can be used in the area of data mining. In data mining, the properties of a set of training data are analyzed to develop a class model that is used to classify future incoming data.

BA codes can also be used in determining inheritance relationships in the hierarchical classification of species.

5 Support for multiple-inheritance

Languages such as Eiffel and *C*++ support multiple-inheritance and therefore programs developed in these languages may have a class structure in the form of a directed a-cyclic graph rather than a tree. Below we extend *BA* codes to find parent-child relationships in directed a-cyclic graphs. A node inheriting from more than one parent also inherits multiple codes. The prefix of each such code has is inherited from the corresponding parent and the suffix computed based upon the number of children the parent has. If a parent has multiple codes, the child inherits all the codes. Thus, the code of any node n_i with p parents consists of a set $X_i = \{x_{i1}, x_{i2}, \ldots, x_{ik}\}$ of k codes, when each parent has only one code. Table 2 lists the ancestry codes for the nodes in the DAG, G_1 of Figure 4, representing multiple-inheritance. The procedure *IsAncestor2* can be used to test ancestry.

$$\begin{tikzcd}
 & & n_1 & \\
 & n_2 \arrow[ur] & n_3 \arrow[u] & n_4 \arrow[ul] \\
n_5 \arrow[ur] \arrow[urr] & n_6 \arrow[u] \arrow[urr] & n_7 \arrow[u] \arrow[ur] & n_8 \arrow[u] \\
n_9 \arrow[u] & n_{10} \arrow[ul] \arrow[ur] & n_{11} \arrow[ul] \arrow[u] & n_{12} \arrow[ul] \arrow[u]
\end{tikzcd}$$

Figure 4. Directed a-cyclic graph, G_1

n_i	Prefix of $x_i \in X_i$	Suffix of $x_i \in Xi$
n_1	-	*0*
n_2	*0*	*00*
n_3	*0*	*01*
n_4	*0*	*10*
n_5	*000*	*0*
n_5	*001*	*0*
n_6	*000*	*1*
n_6	*010*	*00*
n_7	*001*	*1*
n_7	*010*	*01*
n_8	*010*	*10*
n_9	*0000*	*0*
n_9	*0010*	*0*
n_{10}	*0000*	*1*
n_{10}	*0010*	*1*
n_{10}	*0011*	*00*
n_{10}	*01001*	*00*
n_{11}	*0001*	*0*
n_{11}	*01000*	*0*
n_{11}	*0011*	*01*
n_{11}	*01001*	*01*
n_{12}	*0011*	*10*
n_{12}	*01001*	*10*
n_{12}	*01010*	*0*

Table 2. *Ancestry codes of G_1*

5.1 Code length

Let p and s represent the maximum in-degree and out-degree respectively of any node in a DAG. The depth, d, of any node is defined as one more than the maximum depth of its parents, with root(s) at $d=0$. The code of any node n_i with p parents consists of a set $X_i = \{x_{i1}, x_{i2}, \ldots, x_{ik}\}$ of k codes. By observation, the length of the code for any $x_{ik} \in X_i$ is similar to that of the case of a tree. Therefore, the code length of any node at depth d, is $k*(1+d*\lceil \log_2 s \rceil)$ for $s > 1$ (there is no multiple-inheritance for $s = 1$). Each node, at any depth $d > 0$, can inherit from p parents. In the worst case, at $d = 0, k = 1$, at $d = 1, k = p$; $d=2, k = p^2$. Now, if at $d = m, k = p^m$, then at $d = m + 1$, since a node can inherit from p parents, each of which in turn can inherit from p parents, each having p^m codes, $k = p^m * p = p^{m+1}$. Therefore, by induction, $k = p^d$ at any depth d of the DAG. Therefore, total length of all codes at any depth d, is $p^d*(1+d*\lceil \log_2 s \rceil)$ for $s>1, d>0$. However, for many applications where $p_{ave} << 2$, code length is reasonable.

5.2 **Algorithm** *IsAncestor2*

In a DAG, to test whether $n_i \subseteq n_j$, *IsAncestor2*, shown in Figure 3, can be used. *IsAncestor2* calls *IsAncestor* for all members of X_j, but it suffices to call it for only one element of X_i because if indeed $n_i \subseteq n_j$, n_i must have inherited a prefix of a code in the set X_j. In the worst case, the for-loop will run through all p^d elements of X_j (where p is the maximum in-degree of any node in the DAG) for a tree of height d. Hence, the worst case time complexity of *IsAncestor* is $O(p^d)$. However, for many applications, where $1 < p << 2$ and the directed a-cyclic task graph is deep, the actual run-times may compare favorably against determining the relationship by graph traversal.

```
int IsAncestor2(Set $X_i$, Set $X_j$)
// Returns 1 if $n_i \subseteq n_j$, otherwise 0.
{
    unsigned int $x_q$ ,$x_p$ //Set members
    Arbitrarily pick an element $x_p \in X_i$
    for all $x_q \in X_j$ {
        if (IsAncestor($x_p$, $x_q$)==0)
            return 1;  // $n_i \subseteq n_j$
    }
    return 0;  // $n_i \not\subset n_j$
}
```

Figure 3. Procedure *IsAncestor2*

Consider testing whether $n_{11} \subseteq n_4$ using *IsAncestor2*. The prefix part of the codes of n_{11} are, 0001, 01000, 0011 and 01001. Node n_4 has only one code, 010 of length $q_4 = 3$. Since, three bits of a prefix code of n_{11} (actually two!) matches the code of n_4, $n_{11} \subseteq n_4$. Next, consider testing whether $n_{12} \subseteq n_5$. The prefix part of codes of n_{12} are 0011, 01001 and 01010 and of n_5 are 0000 and 0010. Let line *6* of *IsAncestor2* select 0011 from the set of codes of n_{12}. Since the *4*-bits of any of the codes of n_5 do not match the selected code, $n_{12} \not\subset n_5$.

6 Conclusions

This paper has developed *BA* codes for trees using which ancestor-descendant relationships among any nodes of a tree can be determined in *O(1)* time versus *O(d)* for tree traversal, where *d* is the height of the tree. This technique may be used, either at compile-time or run-time, to determine superclass-subclass relationships, needed to perform method resolutions in single-inheritance object-oriented environments. The code length increases linearly with the depth of the tree and logarithmically with the number of siblings. The *BA* codes have been extended for environments where multiple-inheritance is allowed.

References

[1] J. Aoe, "Efficient algorithm of compressing decimal notations for tree structures," *Proceedings of the 13th International Computer Software & Applications Conference*, pp. 316-323, 1989.

[2] S. Baskiyar, "Architectural and scheduler support for object-oriented programs," *Ph.D. Thesis*, University of Minnesota, Minneapolis, 1993.

[3] C. M. Gabriella, D. Guaiana and S. Matanci, "Indecomposable prefix codes and prime trees," *Proceedings of the 3rd International Conference Developments in Language Theory*, pp. 135-145, 1997.

[4] G. H. Gonnet and S. A. Benner, "Probabilistic ancestral sequences and multiple alignments," *Informatik –Organic Chemistry*, 1998.

[5] R. Greenlaw. and R. Petreschi, "Computing Prufer codes efficiently in parallel," *Discrete Applied Mathematics*, v 102, n 3, pp. 205-222, 2000.

[6] D. K. Gupta, "Generation of binary trees from (0-1) codes," *International Journal of Computer Mathematics*, v 42, n 3-4, pp.157-162, 1992.

[7] Y. Jun, F. Fangwei and S. Shiyi, "On the Tunstall codes in source coding," *Acta Mathematicae Applicatae Sinica*, v 23, n 3, pp. 367-376, 2000.

[8] L. Xiang, K. Ushijima and T. Changjie, "Efficient loopless generation of Gray codes for K-ary trees," *Information Processing Letters*, v 76, n 4-6, pp. 169-174, 2000.

Appendix A

```
int Mid(int p, int q, unsigned n){
const int w=sizeof(int)*8; //register size (bits)
  unsigned mask = 1 << w-1;
  int prefix = 0;
  int k=p+q;
        q--;
        for (int i=0;i<w;i++){
                if (i>=p && i<k)
                        n & mask ? prefix+=pow(2,q--): q--;
                n <<=1;
        }//for
        return prefix;
}//Mid
```

BIODATA OF AUTHORS

Sanjeev Baskiyar received the PhD and MSEE degrees (major: Electrical (Computer) Engineering, minor: Computer Science) from the *University of Minnesota,* Minneapolis in 1993 and 1988 respectively and the BE (Electronics and Communications) degree from the *Indian Institute of Science,* Bangalore in 1984. He received the BS degree in Physics with honors and distinction in Mathematics in 1981. He was a recipient of the competitive National Merit Scholarship, the State-merit Scholarship (twice) and the Indian Institute of Science Scholarship. He ranked in the top quartile in JEE/IIT. He was nominated for the *Best Teaching Assistant Award of the Year* in the ECE dept. at the University of Minnesota, Minneapolis. He has taught courses in the areas of Real-time and Embedded Computing, Computer Systems Architecture, Operating Systems, Microprocessor Programming and Interfacing and VLSI Design. His research interests are in the areas of Computer Systems Architecture, Distributed Computing and Task Scheduling. His experience includes working as an Assistant Professor at Western Michigan University, as a Senior Software Engineer in the Unisys Corporation, as a Programmer in IBM's *ProjectWoksape,* as a Project Manager in U of M, St. Paul and as an Assistant Computer Engineer in Tata Engineering and Locomotive Company Ltd., India. He has published in the areas of Computer Systems Architecture and Task Scheduling on Multiprocessors *(web site—http://www.eng.auburn.edu/users/baskiyar).*

N. Meghanathan received the Bachelor of Engineering degree in Chemical Engineering from Anna University, India in 1998 and a MS degree in Computer Science from Auburn University, AL in 2002. He was a research assistant in the department of Chemical Engineering and later in the department of Computer Science and Software Engineering at Auburn University.