J. Software Engineering & Applications, **2009, 2: 111-115**
doi:10.4236/jsea.2009.22016 Published Online July 2009 (www.SciRP.org/journal/jsea)

An Optimal Algorithm for Prufer Codes*

Xiaodong Wang[1, 2], Lei Wang[3], Yingjie Wu[1]

[1]Department of Computer Science, Fuzhou University, Fuzhou, China; [2]Department of Computer Science, Quanzhou Normal University, Quanzhou, China; [3]College of Computing, Georgia Institute of Technology, Atlanta GA 30332, USA.
Email: wangxd@fzu.edu.cn

Received February 19th, 2009; revised February 23rd, 2009; accepted February 24th, 2009.

ABSTRACT

This paper studies the algorithms for coding and decoding Prufer codes of a labeled tree. The algorithms for coding and decoding Prufer codes of a labeled tree in the literatures require $O(nlogn)$ *time usually. Although there exist linear time algorithms for Prufer-like codes* [*1,2,3*], *the algorithms utilize the integer sorting algorithms. The special range of the integers to be sorted is utilized to obtain a linear time integer sorting algorithm. The Prufer code problem is reduced to integer sorting. In this paper we consider the Prufer code problem in a different angle and a more direct manner. We start from a naïve algorithm, then improved it gradually and finally we obtain a very practical linear time algorithm. The techniques we used in this paper are of interest in their own right.*

Keywords: *Design of Algorithm, Labeled Trees, Prufer Codes, Integer Sorting*

1. Introduction

Labeled trees are of interest in practical and theoretical areas of computer science. For example, Ethernet has a unique path between terminal devices, thus being a tree: labeling the tree nodes is necessary to uniquely identify each device in the network. An interesting alternative to the usual representations of tree data structures in computer memories is based on coding labeled trees by means of strings of node labels. This representation was first used in the proof of Cayley's theorem [4] to show a one-to-one correspondence between free labeled trees on n nodes and strings of length n-2. In addition to this purely mathematical use, string-based coding of trees has many practical applications. For instance, they make it possible to generate random uniformly distributed trees and random connected graphs: the generation of a random string followed by the use of a fast decoding algorithm is typically more efficient than random tree generation by the addition of edges, since in the latter case one must pay attention not to introduce cycles. In addition, tree codes are employed in genetic algorithms, where chromosomes in the population are represented as strings of integers, and in heuristics for computing minimum spanning trees with additional constraints, e.g., on the number of leaves or on the diameter of the tree itself. Not last, tree codes are used for data compression and for computing the tree and forest volumes of graphs.

Let T be a labeled tree whose nodes are numbered from 0 to n-1. For some vertex v in T, the degree of v, denoted by d[v], is the number of edges incident to v. If d[v]=1, then v is called a leaf. According to Prufer's proof, any sequence of n-2 numbers, each number in {0,1,…,n-1} can determine a unique labeled tree of n nodes. Such a number sequence is the Prufer code of a labeled tree. Algorithm A is the straightforward implementation of Prufer's proof.

Algorithm A

Input: A labeled tree *T* of n nodes as a list of *n*-1 edges.

Output: c, the Prufer code of *T*.

```
Method:
Step 1. B←{0,1,…,n-1}.
Step 2. For $i$ ← 0 to $n$-3 do
Step 2.1. $x$ ← min{k∈B: k is a leaf }.
Step 2.2. B←B-{x}.
Step 2.3. Remove $x$ and its incident edge ($x$, $y$) from T.
Step 2.4. $c$[i]←$y$.
Step 3. Return $c$.
End of Algorithm
```

For example, let the input labeled tree T be the graph depicted in Figure 1. After the Algorithm A terminates, c = [2,4,0,1,3,3] which is the Prufer code of T .

[*]Supported by Natural Science Foundation of China under Grant No. 60172017 and Natural Science Foundation of Fujian under Grant No. A0510008.

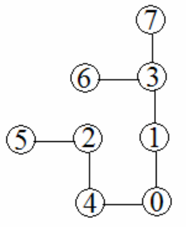

Figure 1. A labelled tree

The algorithms for coding and decoding Prufer codes of a labeled tree in the literatures require $O(nlogn)$ time usually. As stated in [3], although the problem of producing a Prufer code in linear time is an exercise in two books [5, 6], there exists no explicit publication of a solution. In [3] an $O(n)$ time algorithm for generating a Prufer code from a labeled tree was described, but the $O(n)$ time algorithm utilized the integer sorting algorithms. The special range of the integers to be sorted was utilized to obtain a linear time integer sorting algorithm. The Prufer code problem is reduced to integer sorting. In this paper we consider the Prufer code problem in a different angle and present a very practical linear time algorithm directly. The techniques we used in this paper are of interest in their own right.

2. A Linear Time Algorithm for Coding

The most time consuming step in Algorithm A is Step 2. The leaf elimination scheme of the Prufer code implicitly defines a root for the labeled tree T. Actually, it is easy to see that the last element in the code is n-1 which is the root of the labeled tree T. Given a list of n-1 edges, we can build the parent array f and the degree array d of the labeled tree T with only $O(n)$ preprocessing time, which allows checking and updating a node's degree in $O(1)$ time. With this preprocessing, the Step 2.3 of Algorithm A can be implemented in $O(1)$ time. Using a heap, the leaf with the smallest number can be found in $O(nlogn)$ time. Hence, Step 2 takes $O(nlogn)$ time totally. The time complexity of Algorithm A is also $O(nlogn)$.

We can improve the algorithms further. An insight into the problem is that for the heap operations in the Step 2.1 of Algorithm A, there is a special kind of nodes that they are deleted from the heap immediately after they are inserted into the heap. This kind of nodes can be treated easily without heap operation. The remained heap operation can be replaced by a linear scan of the degree array d. Therefore the total time to find x in the Step 2.1 is reduced to $O(n)$.

The linear time algorithm can be presented detailed as follows.

Coding Algorithm:

```
1: index ← x ← min{0≤k<n: d[k]=1}
2: for i ← 0 to n-3 do
3:     y ← f[x]
4:     c[i] ← y
5:     d[y] ← d[y]-1
6:     if y<index and d[y]=1 then x ← y
7:     else index ← x ← min{index<k<n: d[k]=1}
```

In the algorithm described above, d[v] is the degree of node v and f[v] is the parent node of the node v. The variable index is a cursor of degree array d. On line 6, when node y becomes a leaf and the label of y is less than the current cursor index, the node y must be the kind of nodes that are deleted from the heap immediately after they are inserted into the heap. In this case, the node y becomes the next node with minimal label. Otherwise, on line 7 the cursor index moves to the next leaf node in the degree array d. The computing time of line 1 and line 7 is $O(n)$, since the cursor index goes through the degree array d from left to right once. The remaining time is clearly $O(n)$, leading to $O(n)$ complexity for the entire algorithm.

3. A Linear Time Algorithm for Decoding

Decoding is to build the tree T corresponding to the given Prufer code c. As far as c is computed, each node label in it represents the parent of a leaf eliminated from T. Hence, in order to reconstruct T , it is sufficient to compute the ordered sequence of labels of the eliminated leaves, say s: for each i $\in$ {0,1,…,n-1}, the pair (c[i], s[i]) will thus be an edge in the tree.

We first observe that the leaves of T are exactly those nodes that do not appear in the code, as they are not parents of any node. Each internal node, say v, in general may appear in c more than once; each appearance corresponds to the elimination of one of its children, and therefore to decreasing the degree of v by 1. After the rightmost occurrence in the code, v is clearly a leaf and thus becomes a candidate for being eliminated. Therefore, the times of a node v appears in c is exactly the degree of v minus 1. A linear scan of code array c can determine the degree array d.

Using a heap, the leaf with the smallest number can be found in $O(nlogn)$ time, leading an $O(nlogn)$ time decoding algorithm. Like the coding algorithm, we can improve the algorithms further. An insight into the problem is also that for the heap operations of the decoding algorithm, there is a special kind of nodes that they are deleted from the heap immediately after they are inserted into the heap. This kind of nodes can be treated easily without heap operation. The remained heap operation can be replaced by a linear scan of the degree array d.

The linear time algorithm can be presented detailed as follows.

```
Decoding Algorithm:
1: index ← x ← min{0≤k<n: d[k]=1}
2: for i ← 0 to n-2 do
3:     y ← c[i]
4:     add edge (x,y) to T
5:     d[y] ← d[y]-1
6:     if y<index and d[y]=1 then x ← y
7:     else index ← x ← min{index<k<n: d[k]=1}
```

Like the coding algorithm, the time required by the decoding is also $O(n)$.

4. Examples

We use the labeled tree T depicted in Figure 1 as an example to demonstrate the algorithms for coding and decoding Prufer codes described above. The input of the labeled tree T is an edge list {0,1}{0,4}{1,3}{4,2}{3,6}{3,7}{2,5}. There are 8(n) nodes labeled 0,1,2,3,4,5,6,7 and 7(n-1) edges.

It is easy to see that the last element 7 is the root of the labeled tree T. For the given edge list, we can build the parent array f and the degree array d of the labeled tree T by a depth first search with only $O(n)$ time.

With this two arrays, the Step 2.3 of Algorithm A can be implemented in $O(1)$ time.

For our linear time coding algorithm, we first go through the degree array d to find an index such that index=min{0≤k<n: d[k]=1}. It is clear that the index equals 5 for the first time in our example as shown in Table 1.

Table 1. The parent array f and the degree array d of T

	i							
	0	1	2	3	4	5	6	7
f[i]	1	3	4	7	0	2	3	-1
d[i]	2	2	2	3	2	1	1	1
						↑ index		

Table 2. After edge {2,5} is deleted

	i							
	0	1	2	3	4	[5]	6	7
f[i]	1	3	4	7	0	2	3	-1
d[i]	2	2	1	3	2	1	1	1
						↑ index		

Table 3. After edge {4,2} is deleted

	i							
	0	1	[2]	3	4	[5]	6	7
f[i]	1	3	4	7	0	2	3	-1
d[i]	2	2	1	3	1	1	1	1
						↑ index		

Table 4. After the edges {0,4}, {0,1} and {1,3} are deleted

	i							
	[0]	[1]	[2]	3	[4]	[5]	6	7
f[i]	1	3	4	7	0	2	3	-1
d[i]	1	1	1	2	1	1	1	1
						↑ index		

Table 5. After the edge {3,6} is deleted

	i							
	[0]	[1]	[2]	3	[4]	[5]	[6]	7
f[i]	1	3	4	7	0	2	3	-1
d[i]	1	1	1	1	1	1	1	1
							↑ index	

f[index]=f[5]=2 is the first value of the Prufer code c. The edge {2,5} is deleted and d[2] is decreased by 1 on line 5 of the coding algorithm. The status of the tree T now becomes Table 2.

Follows the coding algorithm on line 6 the next node to be deleted is 2. The father node of 2 is node 4, the next value of the Prufer code c. The edge {4,2} is deleted and d[4] is decreased by 1 on line 5 of the coding algorithm. The status of the tree T now becomes Table 3.

Similarly, in the next three steps we obtain the next three values 0,1 and 3 of the Prufer code c. The edges {0,4}, {0,1} and {1,3} are deleted accordingly. The Prufer code we have obtained up to now is (2,4,0,1,3). The status of the tree T now becomes Table 4.

Look at the Table 4, the next node to be deleted is 6 which is determined on line 7 of our coding algorithm. We do not scan the degree array d from the beginning. We scan the degree array d from index (5) to right and the index is moved to 6. This is a key point to see the algorithm running in linear time.

The father node of 6 is node 3, the next value of the Prufer code c. The edge {3,6} is deleted and d[3] is decreased by 1 on line 5 of the coding algorithm. The status of the tree T now becomes Table 5.

Table 6. The initio status of the decoding algorithm

	i							
	0	1	2	3	4	5	6	7
c[i]	2	4	0	1	3	3	7	-1
d[i]	2	2	2	3	2	1	1	1
						↑ index		

Table 7. After edge {2,5} is added

	i							
	0	1	2	3	4	[5]	6	7
c[i]	2	4	0	1	3	3	7	-1
d[i]	2	2	1	3	2	1	1	1
						↑ index		

Table 8. After edge {2,4} is added

	i							
	0	1	[2]	3	4	[5]	6	7
c[i]	2	4	0	1	3	3	7	-1
d[i]	2	2	1	3	1	1	1	1
						↑ index		

We now have the n-2 values of the Prufer code (2,4,0,1,3,3). The algorithm terminates.

Using the same labeled tree T depicted in Figure 1 as an example we can demonstrate the algorithm for decoding Prufer codes as follows.

The input of the decoding algorithm is the Prufer code c of the tree T. In our example the Prufer code of the tree T is (2,4,0,1,3,3). We fist noted that the times of a node v appears in the Prufer code c is exactly the degree of v minus 1. A linear scan of code array c can determine the degree array d as shown in Table 6.

For our linear time decoding algorithm, we first go through the degree array d to find an index such that index=min{0≤k<n:d[k]=1}. It is clear that the index equals 5 for the first time in our example as shown in Table 6 which is the first leaf node deleted in the coding algorithm.

c[0]=2 is the other node label of the edge to be added. The edge {2,5} is added and d[2] is decreased by 1 on line 5 of the decoding algorithm. The status of the tree T now becomes Table 7.

Follows the decoding algorithm on line 6 the next node to be added is 2. c[1]=4 is the other node label of the edge to be added. The edge {2,4} is added and d[4] is decreased by 1 on line 5 of the decoding algorithm. The status of the tree T now becomes Table 8.

Similarly, in the next three steps we obtain the next three edges {0,4}, {0,1} and {1,3}. The status of the tree T now becomes Table 9.

Look at the Table 9. The next node to be added is 6 which is determined on line 7 of our decoding algorithm. We do not scan the degree array d from the beginning. We scan the degree array d from index (5) to right and the index is moved to 6. This is a key point to see the decoding algorithm running in linear time.

The other node label of the edge to be added is c[5]=3. The edge {3,6} is added and d[3] is decreased by 1 on line 5 of the decoding algorithm. The status of the tree T now becomes Table 10.

Follows the decoding algorithm on line 6 the next node to be added is 3. c[6]=7 is the other node label of the edge to be added. The edge {3,7} is added. We now have the n-1 edges of the tree T. The decoding algorithm terminates.

5. Conclusions

Prufer codes for labeled trees have many practical applications. The algorithms for coding and decoding Prufer codes of a labeled tree are of interest in practical and theoretical areas of computer science. The existing linear time algorithms for Prufer-like codes utilized the integer sorting algorithms. The special range of the integers to be sorted is utilized to obtain a linear time integer sorting algorithm. The optimal algorithms for coding and decoding Prufer codes presented in this paper are very practical linear time algorithms. The techniques used in these algorithms are of interest in their own right.

Table 9. After the edges {0,4}, {0,1} and {1,3} are added

	i							
	[0]	[1]	[2]	3	[4]	[5]	6	7
c[i]	2	4	0	1	3	3	7	-1
d[i]	1	1	1	2	1	1	1	1
						↑ index		

Table 10. After the edge {3,6} is added

	i							
	[0]	[1]	[2]	3	[4]	[5]	[6]	7
c[i]	2	4	0	1	3	3	7	-1
d[i]	1	1	1	1	1	1	1	1
						↑ index		

6. Acknowledgments

This work was supported by Natural Science Foundation of China under Grant No. 60172017 and Natural Science Foundation of Fujian under Grant No. A0510008.

REFERENCES

[1] S. Caminiti, I. Finocchi, and R. Petreschi, “A unified approach to coding labeled trees,” in Proceedings of the 6th Latin American Symposium on Theoretical Informatics (LATIN’04), LNCS 2976, pp. 339−348, 2004.

[2] S. Caminiti, I. Finocchi, and R. Petreschi, “On coding labeled trees,” To appear on Theoretical Computer Science, 2006.

[3] H. C. Chen and Y. L. Wang, “An efficient algorithm for generating Prufer codes from labeled trees,” Theory of Computing Systems, Vol. 33, pp. 97–105, 2000.

[4] A. Cayley, “A theorem on trees,” Quarterly Journal of Mathematics, Vol. 23, pp. 376–378, 1889.

[5] L. Devroye, “Non-uniform random variate generation,” Springer-Verlag, New York, Exercise 2, pp. 666, 1986.

[6] A. Nijenhuis and H. S. Wilf, “Combinatorial algorithms for computers and calculators,” Second Editon, Academic Press, New York, Exercise 46, pp. 293, 1978.