Xiaodong WANG[1], Jun TIAN[2]

Fuzhou University (1), Fujian Medical University (2)

Efficient Tree Coding Algorithms

Abstract. *This paper studies the algorithms for coding and decoding second Neville's codes of a labeled tree. The algorithms for coding and decoding second Neville's codes of a labeled tree in the literatures require O(n log n) time usually. As stated in [1][2], no linear time algorithms for the second Neville's codes. In this paper we consider the second Neville's 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 interesting themselves.*

Streszczenie. *W artykule rozważano problem kodu Neville drugiego rzędu stosowanego do etykietowania elementów struktury typu drzewo. (**Skuteczny algorytm kodowania drzewa**)*

Keywords: Labeled trees, Neville codes, depth first search, optimal algorithms.
Słowa kluczowe: kodowane drzewo, kod Neville

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 [3] 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 have 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[8]. 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 [5][6][7][10].

A tree code is any bijection between the set of all free labeled trees on n nodes and strings of length n − 2. The oldest and most famous code is due to Prufer and always deletes the leaf with smallest label. One drawback of the Prufer code is its lack of structure for determining the diameter or the center(s) of the tree directly from the code without first constructing the tree. In 1953, Neville [9] presented three different codes, the first of which coincides with Prufer's one. The second Neville's code, before updating T , eliminates all the leaves ordered by increasing labels. The third Neville's code works by deleting chains.

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. The labeled tree has *n*-1 edges $e_0, e_1, \cdots, e_{n-2}$. The second Neville's code for the labeled tree T denoted as $c_0, c_1, \cdots, c_{n-3}$, $0 \le c_i \le n-1$, $0 \le i \le n-3$, can be defined as follows.

(1) Let X be the node set containing all current leaf nodes of T.

(2) Delete all the leaf nodes in X ordered by increasing labels. When a leaf is deleted, the label of its parent is added to the code.

Repeat (1) and (2) until all the *n*-2 code words $c_0, c_1, \cdots, c_{n-3}$ are produced.

The algorithms for coding and decoding second Neville's codes of a labeled tree in the literatures require $O(n \log n)$ time usually. As stated in [1][2],no linear time algorithms for the second Neville's codes. In this paper we consider the second Neville's code problem in a different angle and a more direct manner. We start from a naïve algorithm, and then improved it gradually and finally we obtain a very practical linear time algorithm.

A Linear Time Algorithm for Coding

The most time consuming step in Algorithm A is Step (2). The leaf elimination scheme of the second Neville's code implicitly defines a root for the labeled tree T. Actually, it is easy to see that the last element in the code is the root of the labeled tree T which is the center of the tree T. Therefore the second Neville's code corresponding to a rooted labeled tree. The diameter of a labeled tree T is defined to be the length of the longest path in T, denoted $k = diam(T)$. The center of the tree T is the middle node in the longest path of T. Let $V_0, V_1, \cdots, V_k$ be a longest path of T. If k is even, then node $V_{k/2}$ is the center of the tree T. If k is odd, then nodes $V_{k/2}$ and $V_{k/2+1}$ are the two centers of the tree T. For our algorithm to generate the second Neville's code of T, we must compute the center of T first. The center of the tree T can be found in $O(n)$ time with two Depth First Traversals (DFT).

The first DFT can be start from any node of T. The purpose of the first DFT is to compute the length of the longest path of T and an end node in this path.

The input parameters v and dep of the algorithm dfs are the current node visited and the depth of the recursion. The array f records the visited status of the nodes initializing with value -1 for each node. When visited the value of f[v] for the node v is the label of the deepest node of the subtree rooted at node v, thus is nonnegative. In the algorithm, the variable g is used to record the largest depths of the subtree rooted at node v. The variable h is used to record the length of the longest path which starts from node

v and in the direction opposite to the node f[v]. Therefore the value of g+h is the length of longest path passing through node v. In the algorithm, the global variable maxl is used to record the length of the current longest path, and another global variable maxv is used to record the end node in the current longest path. After the first DFT algorithm is terminated, the value of the variable maxl is the length of the longest path of T, and the value of the variable maxv is the label of the end node of the longest path.

The second DFT starts from node maxv, the end node of the longest path of T. The purpose of the second DFT is to compute the center of T.

Another global variable m represents the parity of the length of the longest path, that is m=maxk mod 2. After the second DFT algorithm is terminated, the value of the variable cen is the label of the cetre node of T.

Based on the discussion above, we can describe the algorithm to find the center of the tree T by twice Depth First Traversals as follows.

```
Algorithm Center:
1:   maxl ← 0
2:   for i ← 0 to n-1 do f[i]← -1
3:   dfs(0,0)
4:   m ← maxl mod 2
5:   for i ← 0 to n-1 do f[i]← -1
6:   dfs(maxv,0)
7:   return cen
```

In the algorithm above, the simple for loop needs $O(n)$ time. The two DFTs need $O(n)$ time. The total time for the algorithm is $O(n)$.

Once the center of the tree T is determined, the parent node array f and the degree array d for the tree T can be established by another Depth First Traversal or a Breadth First Traversal in $O(n)$ time, which allows checking and updating a node's degree in $O(1)$ time. By using a heap, the set containing current leaf nodes can be maintained in $O(n \log n)$ total time. The time complexity of the coding algorithm is therefore $O(n \log n)$. Based on the discussion above, we can improve the $O(n \log n)$ time algorithm.

In the second Neville's coding algorithm the leaves are deleted level by level. For example, in the rooted labeled tree (a) of figure 2, the nodes are divided into 4 levels. In the first level, the leaves 1,4,5,7,10 are deleted. In the second level, the nodes 0,2,8 and then in the third level, the nodes 6,9 are deleted. The levels for all nodes can be computed in $O(n)$ time with another Depth First Traversal of the tree T.

```
Algorithm Invoke:
invoke(v)
1:  for i ← 0 to n-1 do f[i]← -1, l[i] ← 0
2:  r← f[v]← 0
3:  dft(v)
```

Now we build a queue for every level. By scanning the level array l from left to right, we add the label of the parent node of the scanning node to the corresponding queue. After the scanning finished, the labels are output from the queue level by level. The resulting output is the second Neville's code of the tree T. The coding algorithm based on above discussion can be described as follows.

```
Coding Algorithm:
1: for x ← 0 to n-1 do
2:     y ← l[x], z ← f[x]
3:     ENQUEUE(q[y],z)
4: i ← 0
5: for x ← 0 to r do
6:    While not EMPTY(q[x]) do
7:      DEQUEUE(q[x],y)
8:      c[i] ← y, i ← i+1
```

In the coding algorithm above, the array q is a level queue array. When the node x is scanned, its level is l[x]. The parent label f[x] of the node x is added to the queue q[l[x]]. After the scanning finished, the labels are output in turn from the queue q[0],q[1],⋯,q[r] to the code array c.

Since every queue operation costs $O(1)$ time, the total time required by the algorithm is $O(n)$.

A Linear Time Algorithm for Decoding

Decoding is to build the tree T corresponding to the given second Neville's 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 $i \in \{0,1,\ldots,n-1\}$, 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. In the second Neville's coding algorithm the leaves are deleted level by level. For example, in the rooted labeled tree (a) of figure 2, the nodes are divided into 4 levels. In the first level, the leaves 1,4,5,7,10 are deleted. In the second level, the nodes 0,2,8 and then in the third level, the nodes 6,9 are deleted. In general cases, a linear scan of code array c can determine the level array l. The scanning algorithm can be described as follows.

```
Algorithm Level:
1:  i← k← x← y← p← r← 0
2:  for x ← 0 to n-1 do
3:    if d[x]=1 then l[x]← r, k← k+1
4:  while i<n-2 do
5:      r← r+1
6:      for j ← 0 to k-1 do
7:     d[c[i]]← d[c[i]]-1
8:     if d[c[i]]=1 then l[c[i]]← r, p← p+1
9:          i← i+1
10:     k← p, p← 0
```

In the algorithm above, the levels for all nodes are stored in array l. The global variable r is used to record the maximal level. The algorithm performs a linear scan of the code array c, the time required by the algorithm is clearly $O(n)$.

Now we build a queue for every level. By scanning the level array l from left to right, we add the label of the scanning node to the corresponding queue. After the scanning finished, the labels are output from the queue level by level and the tree edges are produced with code word c[i] accordingly. All output edges form the tree T. The decoding algorithm based on above discussion can be described as follows.

```
Decoding Algorithm:
1: for x ← 0 to n-1 do
2:     y ← l[x]
3:     ENQUEUE(q[y],x)
4: i ← y ← 0
5: for x ← 0 to r do
6:    While not EMPTY(q[x]) do
7:      DEQUEUE(q[x],y)
8:      e[i][0] ← y, e[i][1] ← c[i], i ← i+1
```

In the decoding algorithm above, the array q is a level queue array. When the node x is scanned, its level is l[x]. The node x is added to the queue q[l[x]]. After the scanning finished, the labels are output from the queue level by level and the tree edges are produced with code word c[i] accordingly. Since every queue operation costs $O(1)$ time, the total time required by the algorithm is $O(n)$.

The authors acknowledge the financial support of Science and Technology of Fengze under Grant No.2009FZ24 and 2010FZ02 and the Haixi Project of Fujian under Grant No.A099.

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, 339-348, 2004.

[2] S. Caminiti, I. Finocchi, and R. Petreschi. On Coding Labeled Trees. To appear on Theoretical Computer Science, 2006.

[3] A.Cayley, A theorem on trees. Quarterly Journal of Mathematics, 23, 376–378, 1889.

[4] H.C. Chen and Y.L.Wang, An efficient algorithm for generating Neville codes from labeled trees. Theory of Computing Systems, 33, 97–105, 2000.

[5] N. Deo and P.Micikevicius, Parallel algorithms for computing Neville-like codes of labeled trees. Computer Science Technical Report CS-TR-01-06, 2001.

[6]W. Edelson and M.L.Gargano, Feasible encodings for GA solutions of constrained minimal spanning tree problems. Proceedings of the Genetic and Evolutionary Computation Conference, 754-755, 2000.

[7] A. Kelmans, I.Pak and A.Postnikov, Tree and forest volumes of graphs. DIMACS Technical Report 2000-03, 2000.

[8]V.Kumar, N.Deo and N.Kumar, Parallel generation of random trees and connected graphs. Congressus Numerantium, 130, 7–18, 1998.

[9]E. H. Neville. The codifying of tree-structure. Proceedings of Cambridge Philosophical Society, vol. 49, 381-385, 1953.

[10]G. Zhou and M.Gen, A note on genetic algorithms for degree-constrained spanning tree problems. Networks, 30, 91–95, 1997.

***Authors**: prof. Xiaodong Wang, Quanzhou Normal University, Quanzhou 362000, P.R.China. E-mail: wangxd@fzu.edu.cn; prof. Jun Tian (Corresponding author), Fujian Medical University, PR China,Email: tianjunfjmu@126.com.*