Prüfer-like codes for labeled trees

Article · January 2001

CITATIONS
21

READS
882

2 authors, including:

N. Deo
University of Central Florida
198 PUBLICATIONS **4,556** CITATIONS

SEE PROFILE

Prüfer-Like Codes for Labeled Trees

Narsingh Deo and Paulius Micikevicius
School of Electrical Engineering and Computer Science
University of Central Florida
Orlando, FL 32816-2362
E-mail: {deo, pmicikev} @cs.ucf.edu

Abstract

In 1918 Prüfer showed a one-to-one correspondence between n-node labeled trees and $(n - 2)$-tuples of node labels. The proof employed a tree code, computed by iteratively deleting the leaf with the smallest label and recording its neighbor. Since then other tree codes have been proposed, based on different node deletion sequences. These codes have different properties, interesting and useful in graph theory and computer science. In this paper we survey and classify these Prüfer-like codes as well as new codes based on the parameter values of the proposed generic tree-encoding algorithm and study their properties.

1. Introduction

Labeled trees are of interest in practical and theoretical areas of computer science. For example, Ethernet has one and only one path between every terminal device, and therefore is a tree. Labeling the nodes is necessary to uniquely identify each device in such network. Many graph algorithms require generating spanning trees of labeled graphs.

In 1918 Prüfer [14] showed a one-to-one correspondence between labeled trees of order n and $(n - 2)$-tuples of the node labels. In 1953 Neville [12] published three different methods for encoding labeled trees into $(n - 2)$-tuples of labels, the first one being equivalent to Prüfer's encoding. In 1963 Glicksman [7] gave a different proof of one-to-one correspondence between labeled trees and $(n - 2)$-tuples of nodes. However, his proof does not lead to an encoding. Recently, the authors have proposed yet another tree encoding method in [2]. Similar codes have been used for counting labeled q-trees and spanning trees of multipartite graphs [5, 6].

A number of properties for random labeled trees, such as the expected value of the diameter, radius, number of leaves, and the node-degree distribution, are easily determined by combinatorial arguments for tree codes [11]. Prüfer code is also utilized in tree and forest volumes of graphs [9].

Tree codes are also employed in genetic algorithms, where chromosomes in the population are represented as strings of integers. Genetic algorithms have

been used, for example, in heuristics for computing constrained minimum spanning trees – minimum-weight spanning trees satisfying an additional constraint, such as on the number of leaves, maximum degree, or diameter of the tree [4, 16]. Prüfer codes are also used to generate random trees and random connected graphs [10].

Prüfer-like tree codes are of interest in parallel computation as well. Parallel algorithms for computing Prüfer codes were proposed in [1, 8], an algorithm for constructing a tree from the Prüfer code was proposed in [16]. While the algorithms in [1, 8] are not work-optimal, a work-optimal parallel algorithm exists for encoding a tree using Neville's third method [3].

In this paper we classify the methods for encoding labeled trees based on the parameter values of a proposed generic tree-encoding algorithm. The parameter values result in the existing encoding algorithms, as well as one new encoding. Definitions and assumptions made in this paper are listed in Section 2. Existing tree-encoding methods are reviewed in Section 3. Construction of trees from the codes is described in Section 4. In Section 5 we propose the generic tree-encoding algorithm. Code classification is presented in Section 6.

2. Assumptions and Definitions

It is assumed that the nodes of a tree T are labeled with integers 1, …, n.

T: a tree on n nodes.
n: number of nodes in the tree.
$\deg_T(v)$: degree of node v in tree T.
r: root node of T (which is never deleted from the tree).
L: a list of nodes.
C: a tree code (an $(n-2)$-tuple of node labels)

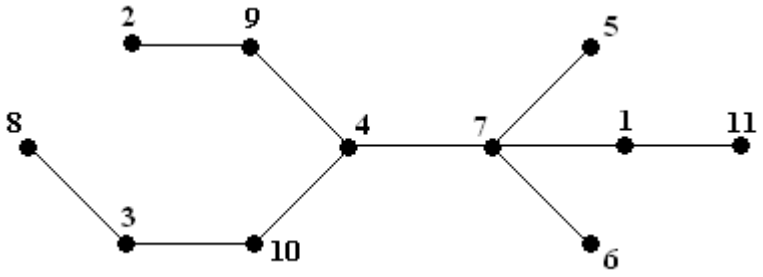

Figure 1. A tree on 11 nodes

3. Existing Methods for Encoding Labeled Trees

Prüfer's method [14] encodes a tree by iteratively deleting the leaf node with the smallest label and recording its neighbor, until only one edge remains. For example, the Prüfer code for the tree in Figure 1 is (9, 7, 7, 3, 10, 4, 4, 7, 1). An O(n)-time algorithm for constructing a tree from its Prüfer code has been proposed by Kilingsberg [13](p. 271). Kilingsberg's method can also be adopted to obtain the Prüfer code for a given tree of order n in O(n) time. One drawback of the Prüfer 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.

Neville [12] proposed three different methods for encoding labeled trees. In all three algorithms an arbitrary node is selected as the root node, which is never deleted from the tree. Modification of Neville's algorithms that does not require a root node was proposed by Moon [11]. When the node with the largest label is chosen as the root the first encoding proposed by Neville is equivalent to the Prüfer's method. Thus, Neville's first code for the tree in Figure 1 is (9, 7, 7, 3, 10, 4, 4, 7, 1).

Neville's second method proceeds in k stages, where k is the radius of the tree. In each stage the leaves of the current subtree are deleted in ascending order of the labels, and their neighbors are recorded. The process is repeated until only one edge remains (thus, if the radius of a tree is even, all but one leaf will be removed in the last stage). Employing Neville's second method the code for the tree in Figure 1 would be (9, 7, 7, 3, 1, 7, 10, 4, 4). This encoding scheme leads to a simple algorithm for computing the diameter of a tree directly from the code. However, there are no known O(n)-time algorithms for encoding or decoding trees using Neville's second method. The difficulties arise because leaf nodes must be sorted in each stage of the algorithm.

The third encoding due to Neville starts by deleting the leaf node with the smallest label and recording its neighbor, v. If v is a leaf in the resulting subtree, v is deleted next, otherwise a leaf with the smallest label is deleted. The process is repeated until one edge remains. With this method the code for the tree in Figure 1 would be (9, 4, 7, 7, 3, 10, 4, 7, 1). A tree can be encoded/decoded in O(n) time using Neville's third method.

Recently the authors proposed a new tree encoding scheme employing a stack and a queue [2]. All degree-one nodes are stored in a queue: nodes are deleted from the head of the queue while newly created leaves are appended at the tail of the queue (the leaves of the given tree are added to the queue in ascending order of their labels). Hence, the leaves of the given tree are deleted in ascending order of their labels, while the leaves of subsequent subtrees are deleted in the order in which they become leaves. Each time a node is deleted, the label of the node adjacent to it is appended to the code. The code for the tree in Figure 1 would be (9, 7, 7, 3, 1, 4, 10, 7, 4). The algorithm for constructing a tree from its code employs a stack. Encoding and decoding algorithms require O(n) time.

4. Constructing a Tree from the Code

Given a Prüfer code the tree can be constructed as follows. Let L be the set of all labels that do not appear in the code (we know that a code of length $(n - 2)$ encodes a tree on n nodes, therefore integer labels 1 through n are used). Iteratively consider each label in the code from left to right:

- let u be the current label in the code and v be the smallest label in L,
- add edge (u, v) to the tree,
- remove v from the set L,
- if this was the right-most occurrence of u in the code, add u to the set L.

Now, let u be the last label in the code and v be the remaining label in L. Add edge (u, v) to the tree.

Employing the above procedure one can reconstruct the tree in Figure 1 from Prüfer code (9, 7, 7, 3, 10, 4, 4, 7, 1). Notice that the smallest label in the set L is chosen during each iteration, which corresponds to deleting the leaf with the smallest label when encoding the tree. Tree-reconstruction algorithms for other encoding methods, described in Section 3, are derived similarly; the only difference is the order in which the labels are removed from the set L (the order is always the same as that in which the leaves are deleted when computing the code).

5. Generic Tree-Encoding Algorithm

We propose a generic tree-encoding algorithm, which is used in Section 6 to classify tree-codes. A root node (denoted by r) is never deleted from the tree by the encoding algorithm. Note that $r = n$ for Prüfer's and Neville's codes. We call a tree-encoding method *Prüfer-like* if it:

- iteratively deletes the non-root nodes of degree one, until only one edge remains,
- each time a node is deleted, the label of the node adjacent to it is appended to the code.

The encodings reviewed in Section 3 differ only by the order in which degree-one nodes are deleted. To formalize this order, the generic algorithm maintains L, the list of non-root nodes of degree one, where the node at the head of the list is always deleted next. Initially the list contains the leaves of the given tree. In order to correctly reconstruct the tree from the code, the initial list must be arranged in some deterministic order (in this paper we only consider ascending or descending order). The algorithm employs either a single or multiple lists. For simplicity of presentation we consider the algorithms employing single and multiple lists separately. A single-list generic algorithm for Prüfer-like encoding of a tree on n nodes is described in Figure 2.

```
1.  C ← ∅
2.  L ← {v | deg_T(v) = 1 and v ≠ r}
3.  for i ← 1 to (n – 2) do
4.      u ← the node at the head of L
5.      v ← the node adjacent to u
6.      append v to code C
7.      delete u from T
8.      delete u from L
9.      if deg_T(v) = 1 and v ≠ r then
10.         add v to L
```

Figure 2. Single-list generic algorithm for Prüfer-like encoding

The algorithm starts with an empty code C (line 1). The list L is initialized to contain all the leaves of the given tree on line 2. Nodes of degree one are iteratively deleted and their neighbors are recorded to the code in the loop on lines 3 through 10. More details on adding the nodes to list L on line 10, selecting the root node, ordering list L on line 2 are provided in Section 6.

A multiple-list generic tree-encoding algorithm is described in Figure 3 (the k^{th} list is denoted by $L(k)$).

```
1.  C ← ∅
2.  k ← 0
3.  L(k) ← {v | deg_T(v) = 1 and v ≠ r}
4.  L(k + 1) ← ∅
5.  for i ← 1 to (n – 2) do
6.      u ← the node at the head of L(k)
7.      v ← the node adjacent to u
8.      append v to code C
9.      delete u from T
10.     delete u from L(k)
11.     if L(k) = ∅ then
12.         k ← k + 1
13.         L(k + 1) ← ∅
14.     if deg_T(v) = 1 and v ≠ r then
15.         add v to L(k + 1)
```

Figure 3. Multiple-list generic algorithm for Prüfer-like encoding

The algorithm in Figure 3 is an extension of the algorithm in Figure 2. During each iteration the node at the head of list $L(k)$ is deleted from the tree (lines 6 and 9), while newly created leaves are added to list $L(k + 1)$ on line 15. If the last node in $L(k)$ is deleted on line 10, the variable k is incremented and a new empty list is

created (lines 12 and 13, respectively). Notice that list $L(k)$ contains only the nodes that became leaves when the nodes in $L(k-1)$ were deleted. Adding newly created leaves to the list, selecting the root node, and ordering the initial list $L(0)$ is discussed in the next section.

6. Classification of Prüfer-Like Codes

To obtain a complete specification for the generic algorithm described in Section 4, one must select the values for the following four parameters:

- number of lists,
- method for adding newly created leaves to the current list,
- method for sorting the initial list,
- root node r.

Encodings and resulting tree-codes are classified by the parameter values. A complete algorithm is designated by a string of one-letter abbreviations for the first three parameters and the label of the root node, in the following order: number of lists, method for adding leaves, sorting method, root node. There are two choices for the number of lists: one or multiple lists, abbreviated with letter O or M, respectively. Newly created leaves can be added to the current list in one of the three ways: at the head of the list, at the tail of the list, or inserted to maintain the sorted order, denoted by H, T, and S, respectively. Notice that adding leaves at the head of the list is equivalent to maintaining a stack data structure and adding at the tail of the list is equivalent to keeping a queue of degree-one nodes. Any one of the n nodes in the tree can be selected as the root node. The initial list can be sorted in either ascending or descending order of the labels, denoted by A and D, respectively. For example, algorithm OHA1 encodes a tree by employing a single list, adding the newly created leaves at the head of the list, sorting the nodes in the initial list in ascending order of their labels, and selecting node 1 as the root. There are $2 \cdot 3 \cdot 2 \cdot n = 12 \cdot n$ combinations of parameter choices for encoding a tree on n nodes. Not all of these $12n$ combinations lead to distinct tree codes, as will be shown in the example.

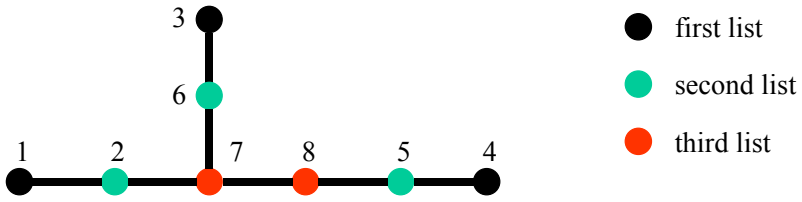

Figure 4. A tree on eight nodes

Consider the tree in Figure 4. The following 12 codes are obtained if node 8 is selected as the root:

OHA8:	(2, 7, 6, 7, 5, 8)	MHA8:	(2, 6, 5, 8, 7, 7)
OTA8:	(2, 6, 5, 7, 7, 8)	MTA8:	(2, 6, 5, 7, 7, 8)
OSA8:	(2, 7, 6, 5, 8, 7)	MSA8:	(2, 6, 5, 7, 8, 7)
OHD8:	(5, 8, 6, 7, 2, 7)	MHD8:	(5, 6, 2, 7, 7, 8)
OTD8:	(5, 6, 2, 8, 7, 7)	MTD8:	(5, 6, 2, 8, 7, 7)
OSD8:	(5, 8, 6, 7, 2, 7)	MSD8:	(5, 6, 2, 7, 8, 7)

Notice that OTA8 and MTA8 codes are equivalent, as are codes OTD8 and MTD8. This observation leads to the following theorem.

Theorem. Tree codes OTxy and MTxy are identical, where x and y are any values for the corresponding parameters.

Proof. It suffices to show that the algorithms OTxy and MTxy delete the same sequence of nodes. Note that the order in which the nodes are added to the lists is determined solely by node deletion sequence. The proof is by induction on the number of lists utilized by MTxy algorithm.

The initial list in OTxy algorithm and the list $L(0)$ in MTxy algorithm are identical since they both contain the leaves of the given tree, arranged in the same order. Thus, the base case holds true.

Now, let us assume that the sequence of k nodes deleted from list L by OTxy algorithm is identical to the sequence deleted from lists $L(0)$, $L(1)$, …, $L(i)$ by MTxy algorithm (all nodes in $L(i)$ have been deleted). The list $L(i + 1)$ contains only the nodes that became leaves when the nodes in $L(i)$ were deleted. List L, maintained by OTxy algorithm, contains the same sequence of nodes as $L(i + 1)$ since nodes are added at the tail of the list. Since the nodes are always deleted from the head of a list, the same sequence of $|L(i + 1)|$ nodes will be deleted next by the algorithms OTxy and MTxy. Thus, the induction step holds true. ■

The number of identical codes depends on the tree. For example, the codes MSA11 and MHA11 for the tree in Figure 1 are both (9, 7, 7, 3, 10, 4, 4, 7, 1). Prüfer's code and Neville's first code are equivalent to OSAn algorithm (single list, maintain sorted order, do not delete node n, the list is sorted in ascending order of the labels). Neville's second and third encodings correspond to MSAn and OHAn algorithms, respectively. Algorithm OTAc is equivalent to the encoding proposed in [2], where c is a center node of the tree. Encoding MHAn is a new method that to the best of our knowledge has not appeared in the literature.

7. Conclusions and Future Work

A broad classification scheme for Prüfer-like codes for labeled trees is proposed in this paper. The classification is based on the parameter values of the proposed generic tree-encoding algorithm. Equivalence of two pairs of encoding classes was also shown, leading to at most $10n$ distinct codes for a labeled tree on n nodes.

It would be interesting to determine how many distinct encodings exist for a given tree, and whether one tree-code can be converted into another without first reconstructing the original tree.

References

[1] H. Chen, Y. Wang. An efficient algorithm for generating Prüfer codes from labeled trees. *Theory of Computing Systems*, vol. 33, pp. 97-105, 2000.

[2] N. Deo, P. Micikevicius. A new encoding for labeled trees employing a stack and a queue. *Bulletin of the ICA*, to appear.

[3] N. Deo, P. Micikevicius. Parallel algorithms for computing Prüfer-like codes for labeled trees. *Technical Report CS-TR-01*, Department of Computer Science, University of Central Florida, Orlando, 2001.

[4] W. Edelson, M. L. Gargano. Feasible encodings for GA solutions of constrained minimal spanning tree problems. *Proceedings of GECCO-2000*, Las Vegas, Nevada, 2000.

[5] Ö. Eğecioğlu, L-P Shen. A bijective proof for the number of labeled q-trees. *Ars Combinatoria*, vol. 25B, pp. 3-30, 1988.

[6] Ö. Eğecioğlu, J. B. Remmel. A bijection for spanning trees of complete multipartite graphs. *Congressus Numerantium*, vol. 100, pp. 225-243, 1994.

[7] S. Glicksman. On the representation and enumeration of trees. *Proceedings of Cambridge Philosophical Society*, vol. 59, pp. 509-517, 1963.

[8] R. Greenlaw, R. Petreschi. Computing Prüfer codes efficiently in parallel. *Discrete Applied Mathematics*, vol. 102, pp. 205-222, 2000.

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

[10] V. Kumar, N. Deo, N. Kumar. Parallel generation of random trees and connected graphs. *Congressus Numerantium*, vol. 138, pp. 7-18, 1998.

[11] J. W. Moon. *Counting Labeled Trees*. William Clowes and Sons, London, 1970.

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

[13] A. Nijenhuis, H. S. Wilf. *Combinatorial Algorithms*. Academic Press, New York, 1978.

[14] H. Prüfer. Neuer Beweis eines Satzes über Permutationen. *Archiv für Mathematik und Physik*, vol. 27, pp. 142-144, 1918.

[15] Y. Wang, H. Chen, W. Liu. A parallel algorithm for constructing a labeled tree. *IEEE Transactions on Parallel and Distributed Systems*, vol. 8, no. 12, pp. 1236-1240, 1997.

[16] G. Zhou, M. Gen. A note on genetic algorithms for degree-constrained spanning tree problems. *Networks*, vol. 30, pp. 91-95, 1997.