A unified approach to coding labeled trees

Saverio Caminiti[1], Irene Finocchi[2], and Rossella Petreschi[1]

[1] DSI, Università degli Studi di Roma "La Sapienza"
Via Salaria 113, 00198 Roma, Italy
{caminiti, petreschi}@dsi.uniroma1.it
[2] DISP, Università degli Studi di Roma "Tor Vergata"
Via del Politecnico 1, 00133 Roma, Italy
finocchi@disp.uniroma2.it

Abstract. We consider the problem of coding labeled trees by means of strings of node labels and we present a unified approach based on a reduction of both coding and decoding to integer (radix) sorting. Applying this approach to four well-known codes introduced by Prüfer [18], Neville [17], and Deo and Micikevicius [5], we close some open problems. With respect to coding, our general sequential algorithm requires optimal linear time, thus solving the problem of optimally computing the second code presented by Neville. The algorithm can be parallelized on the EREW PRAM model, so as to work in $O(\log n)$ time using $O(n)$ or $O(n\sqrt{\log n})$ operations, depending on the code.
With respect to decoding, the problem of finding an optimal sequential algorithm for the second Neville code was also open, and our general scheme solves it. Furthermore, in a parallel setting our scheme yields the first efficient decoding algorithms for the codes in [5] and [17].

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 [2, 18] 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 codings of trees have many practical applications. For instance, they make it possible to generate random uniformly distributed trees and random connected graphs [14]: 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 [7, 8, 20]. Not last, tree codes are used for data compression [19] and for computing the tree and forest volumes of graphs [13].

Tree codes. We now survey the main tree codes known in the literature, referring the interested reader to the taxonomy in [6] for further details. We assume to deal with a labeled n-node rooted tree T whose nodes have distinct labels from $[1, n]$. All the codes that we discuss are obtained by progressively *updating* the tree through the *deletion of leaves*: when a leaf is eliminated, the label of its parent is added to the code.

The oldest and most famous code is due to Prüfer [18] and always deletes the leaf with smallest label. In 1953, Neville [17] presented three different codes, the first of which coincides with Prüfer'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. We call *pending chain* a path $u_1, \ldots, u_k$ of the tree such that the starting point u_1 is a leaf, and, for each $i \in [1, k-1]$, the elimination of u_i makes u_{i+1} a leaf: the code works by iteratively eliminating the pending chain with the smallest starting point. Quite recently, Deo and Micikevicius [5] suggested the following coding approach: at the first iteration, eliminate all tree leaves as in the second Neville's code, then delete the remaining nodes in the order in which they assume degree 1. For brevity, we will denote the codes introduced above with `PR`, `N2`, `N3`, and `DM`, respectively.

All these codes have length $n-1$ and the last element is the root of the tree. If the tree is unrooted, the elimination scheme implicitly defines a root for it. Actually, it is easy to see that the last element in the code is: a) the maximum node label (i.e., n) for Prüfer's code; b) the maximum label of a center of the tree for the second Neville's code; c) the label of the maximum leaf for the third Neville's code; d) the label of any tree center for Deo and Micikevicius's code. In cases a), c), and d), the value of the last element can be univocally determined from the code and thus the code length can be reduced to $n-2$. We remark that codes `PR` and `DM` have been originally presented for free trees, while Neville's codes have been generalized for free trees by Moon [16].

Related work. A linear time algorithm for computing Prüfer codes is presented in [3]. The algorithm can be easily adapted to the third Neville's code. Deo and Micikevicius give a linear time algorithm for code `DM` based on a quite different approach. As stated in [6], no $O(n)$ time algorithm for the second Neville's code was known so far: sorting the leaves before each tree update yields indeed an $O(n \log n)$ bound. An optimal parallel algorithm for computing Prüfer codes, which improves over a previous result due to Greenlaw and Petreschi [10], is given in [9]. A few simple changes make the algorithm work also for code `N3`. Efficient, but not optimal, parallel algorithms for codes `N2` and `DM` are presented in [7]. A simple – non optimal – scheme for constructing a tree T from a Prüfer code is presented in [6]. The scheme can be promptly generalized for building T starting from any of the other codes: this implies decoding in linear time codes `N3` and `DM`, and in $O(n \log n)$ time codes `PR` and `N2`. An $O(n)$ time decoding algorithm for `PR` is described in [9]. In a parallel setting, Wang, Chen, and Liu [19] propose an $O(\log n)$ time decoding algorithm for Prüfer codes using $O(n)$ processors on

the EREW PRAM computational model. At the best of our knowledge, parallel decoding algorithms for the other codes were not known in the literature until this paper.

Our contribution. We show that both coding and decoding can be reduced to integer (radix) sorting. Based on this reduction, we present a unified approach that works for all the codes introduced so far and can be applied both in a sequential and in a parallel setting. The coding scheme is based on the definition of pairs associated to the nodes of T according to criteria dependent on the specific code: the coding problem is then reduced to the problem of sorting these pairs in lexicographic order. The decoding scheme is based on the computation of the rightmost occurrence of each label in the code: this is also reduced to integer radix sorting.

Concerning coding, our general sequential algorithm requires optimal linear time for all the presented codes; in particular, it solves the problem of computing the second Neville code in time $O(n)$, which was still open [6]. The algorithm can be parallelized, and its parallel version either matches or improves by a factor $O(\sqrt{\log n})$ the performances of the best ad-hoc approaches known so far. Concerning decoding, we design the first parallel algorithm for codes `N2`, `N3`, and `DM`, working on the EREW PRAM model in $O(\log n)$ time and $O(n\sqrt{\log n})$ operations (with respect to `PR`, our algorithm matches the performances of the best previous result). Our parallel results both for coding and for decoding are summarized in the following table:

	Coding		Decoding	
	before	this paper	before	this paper
`PR`	$O(n)$ [9]	$O(n)$	$O(n \log n)$ [19]	$O(n \log n)$
`N2`	$O(n \log n)$ [7]	$O(n\sqrt{\log n})$	open	$O(n\sqrt{\log n})$
`N3`	$O(n)$ [7, 9]	$O(n)$	open	$O(n\sqrt{\log n})$
`DM`	$O(n \log n)$ [7]	$O(n\sqrt{\log n})$	open	$O(n\sqrt{\log n})$

where costs are expressed in terms of number of operations. We remark that the problem of finding an optimal sequential decoding algorithm for code `N2` was also open, and our general scheme solves it optimally. Hence, we show that labeled trees can be coded and decoded in linear sequential time independently of the specific code. Due to lack of space, we omit many details in this extended abstract.

2 A unified coding algorithm

Many sequential and parallel coding algorithms have been presented in the literature [3, 5, 6, 9, 10, 19], but all of them strongly depend on the properties of the code which has to be computed and thus are very different from each other. In this section we show a unified approach that works for all the codes introduced in Section 1 and can be used both in a sequential and in a parallel setting.

Table 1. Pair (x_v, y_v) associated to node v for different codes.

	PR	N2	N3	DM
x_v	$\mu(v)$	$l(v)$	$\lambda(v)$	$l(v)$
y_v	$d(\mu(v), v)$	v	$d(\lambda(v), v)$	$\gamma(v)$

Namely, we associate each tree node with a pair of integer numbers and we sort nodes using such pairs as keys. The obtained ordering corresponds to the order in which nodes are removed from the tree and can be thus used to compute the code. In the rest of this section we show how different pair choices yield Prüfer, Neville, and Deo and Micikevicius codes, respectively. We then present a linear time sequential coding algorithm and its parallelization on the EREW PRAM model. The parallel algorithm works in $O(\log n)$ time and requires either $O(n)$ or $O(n\sqrt{\log n})$ operations, depending on the code.

Coding by sorting pairs. Let T be a rooted labeled n-node tree. If T is not rooted, we choose a root r as in points a) – d) in Section 1. Let u, v be any two nodes of tree T. Let us call: T_v, the subtree of T rooted at v; $d(u, v)$, the distance between any two nodes u and v ($d(v, v) = 0$); $l(v)$, the level of a node v, i.e., the maximum distance of v from a leaf in T_v; $\mu(v)$, the maximum label among nodes in T_v; $\lambda(v)$, the maximum label among leaves in T_v; $\gamma(v)$, the maximum label among the leaves in T_v at maximum distance from v; (x_v, y_v), a pair associated to node v according to the specific code as shown in Table 1; P, the set of pairs (x_v, y_v). The following lemma establishes a correspondence between the set P of pairs and the order in which nodes are removed from the tree. Due to lack of space, we defer the proof to the extended version of this paper.

Lemma 1. *For each code, the lexicographic ordering of the pairs (x_v, y_v) in set P corresponds to the order in which nodes are removed from tree T according to the code definition.*

Before describing the sequential and parallel algorithms, note that it is easy to sort the pairs (x_v, y_v) used in the coding scheme. Indeed, independently of the code, each element in such pairs is in the range $[1, n]$. A radix-sort like approach [4] is thus sufficient to sort them according to y_v, first, and x_v, later. In Figure 1 the pairs relative to the four codes are presented. The tree used in the example is the same in the four cases and is rooted according to points a) – d) in Section 1. Bold arcs in the trees related to codes `PR` and `N3` indicate chains [3] and pending chains, respectively; dashed lines in the trees related to codes `N2` and `DM` separate nodes at different levels. In each figure the string representing the generated code is also shown.

Sequential algorithm. Using the pairs defined in Table 1, an optimal sequential coding algorithm is now straightforward:

[3] According to the definition of Prüfer's code, when the node $\mu(v)$ is chosen for removal, the only remaining subtree of T_v consists of a *chain* from $\mu(v)$ to v.

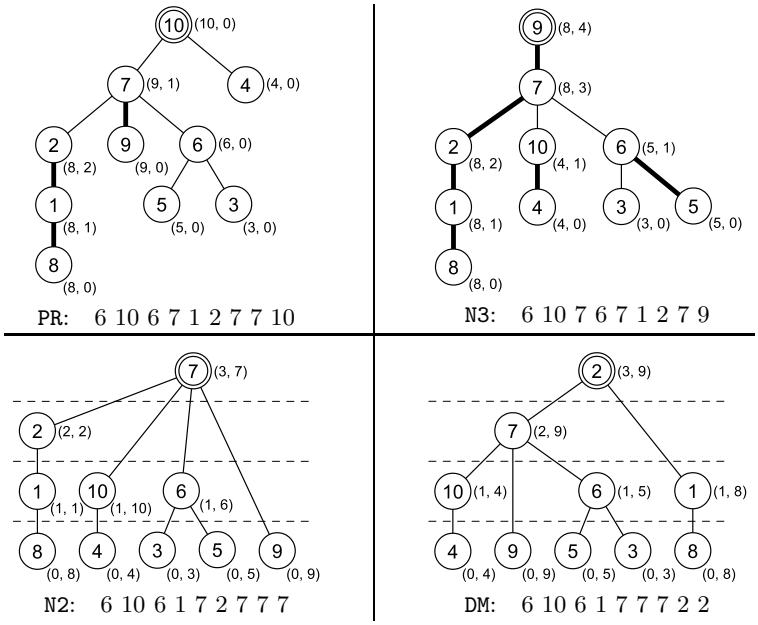

Fig. 1. Pair associated to each tree node as specified in Table 1.

UNIFIED CODING ALGORITHM:
1. **for each** `node` v`, compute the pair` (x_v, y_v)
2. `sort the tree nodes according to pairs` (x_v, y_v)
3. **for** $i = 1$ **to** $n - 1$ **do**
4. `  let` v `be the` i`-th node in the ordering`
5. `  append` $parent(v)$ `to the code`

The UNIFIED CODING ALGORITHM clearly requires linear time: the set of pairs can be easily computed in $O(n)$ time using a post-order visit of the tree, and two bucket-sorts can be used to implement step 2. Hence we have the following theorem:

Theorem 1. *Let T be a n-node tree and let the pair (x_v, y_v) associated to each node v of T be defined as in Table 1. The* UNIFIED CODING ALGORITHM *computes codes* `PR`*,* `N2`*,* `N3`*, and* `DM` *in $O(n)$ worst-case running time.*

Parallel algorithm. We now show how to parallelize each step of the sequential algorithm presented above. We work in the simplest PRAM model with exclusive read and write operations (EREW [12]). The Euler tour technique makes it possible to root the tree at the node r specified in Section 1 in $O(\log n)$ time with cost $O(n)$ [12]. The node r can be easily identified; in particular, when r is the center of the tree, we refer to the approach described in [15]. The pairs given

in Table 1 can be computed in $O(\log n)$ time with cost $O(n)$ using standard techniques, such as Euler tour, rake, and list ranking [12].

Step 3 can be trivially implemented in $O(1)$ time with cost $O(n)$. The sorting in step 2 is thus the most expensive operation. We follow a radix-sort like approach and use the stable integer-sorting algorithm presented in [11] as a subroutine. This requires $O(\log n)$ time, linear space and $O(n\sqrt{\log n})$ cost on an EREW PRAM with $O(\log n)$ word length. Under the hypothesis that the machine word length is $O(\log^2 n)$, the cost of sorting can be reduced to $O(n)$ [11], and so does the cost of our coding algorithm.

We remark that our algorithm solves within a unified framework the problem of computing four different tree codes. In addition, with respect to codes `N2` and `DM`, it improves of an $O(\sqrt{\log n})$ factor over the best approaches known in the literature [7]. Unfortunately, as far as we have described it, it does not match the perfomances of the optimal algorithms available for codes `PR` [9] and `N3` [7]. However, in these cases we can further reduce the cost of our algorithm to $O(n)$ by using an ad-hoc sorting procedure that benefits from the partition into chains.

Let us consider Prüfer codes first. As observed in [10], the final node ordering can be obtained by sorting chains among each other and nodes within each chain. In our framework, the chain ordering is given by the value $\mu(v)$, and the position of each node within its chain by the distance $d(\mu(v), v)$. Instead of using a black-box integer sorting procedure, we exploit the fact that we can compute optimally the size of each chain, i.e., the number of nodes with the same $\mu(v)$. A prefix sum computation gives, for each chain head, the number of nodes in the preceding chains, i.e., its final position. At last, the position of the remaining nodes is univocally determined summing up the position of the chain head $\mu(v)$ with the value $d(\mu(v), v)$. Similar considerations can be applied to the third Neville's code. The following theorem summarizes our results on parallel coding:

Theorem 2. *Let T be a n-node tree and let the pair (x_v, y_v) associated to each node v of T be defined as in Table 1. On the EREW PRAM model, the* UNIFIED CODING ALGORITHM *computes codes* `PR` *and* `N3` *optimally, i.e., in $O(\log n)$ time with cost $O(n)$, and codes* `N2` *and* `DM` *in $O(\log n)$ time with cost $O(n\sqrt{\log n})$.*

3 Decoding algorithms

In this section we present sequential and parallel algorithms for decoding, i.e., for building the tree T corresponding to a given 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 $i \in [1, n-1]$, the pair (C_i, S_i) will thus be an arc in the tree. Before describing the algorithms, we argue that computing the rightmost occurrence of a node in the code is very useful for decoding, and we show how to obtain such an information both in a sequential and in a parallel setting.

Decoding by rightmost occurrence computation. 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. More formally:

$$\forall v \neq r, \quad \exists \; unique \; j > rightmost(v, C) \;\; \text{such that} \;\; S_j = v$$

where r is the tree root (i.e., the last element in C) and $rightmost(v, C)$ denotes the index of the rightmost occurrence of node v in C. We assume that $rightmost(v, C) = 0$ if v is a leaf of T.

It is easy to compute the rightmost occurrence of each node sequentially by simply scanning code C. In parallel, we can reduce the rightmost occurrence computation problem to a pair sorting problem. Namely, we sort in increasing order the pairs (C_i, i), for $i \in [1, n-1]$. Let us now consider the sub-sequences of pairs with the same first element C_i: the second element of the last pair in each sub-sequence is the index of the rightmost occurrence of node C_i in the code. Since each pair value is an integer in $[1, n]$, we can use twice the stable integer-sorting algorithm of [11]: this requires $O(\log n)$ time and $O(n\sqrt{\log n})$ cost in the EREW PRAM model. Then, each processor p_i in parallel compares the first element of the i-th pair in the sorted sequence to the first element of the $(i+1)$-th pair, deciding if this is the end of a sub-sequence or not. This requires additional $O(1)$ time and linear cost with exclusive read and write operations.

A unified decoding algorithm. We now describe a decoding algorithm for codes `N3`, `PR`, and `DM` that works on the rightmost occurrences and can be used both in a sequential and in a parallel setting. First, for each code, we show how the position of a node in the sequence S that we want to construct can be expressed as a function of *rightmost*.

Third Neville code. By definition of code `N3`, each internal node v is eliminated as soon as it becomes a leaf. Thus, the position of v in sequence S is exactly $rightmost(v, C) + 1$. The entries of S which are still free after positioning all the internal nodes are occupied by the leaves of T in increasing order.

Prüfer code. Differently from the third Neville's code, in code `PR` an internal node v is eliminated as soon as it becomes a leaf if and only if there is no leaf with label smaller than v. In order to test this condition, following [19], we introduce the number of nodes with label smaller than v that become leaves before v: $prev(v, C) = |\{u : \; u < v \text{ and } rightmost(u, C) < rightmost(v, C)\}|$. Thus, the position of v in sequence S is $rightmost(v, C) + 1$ if and only if $rightmost(v, \; C) \geq prev(v, C)$. All the other nodes are assigned to the remaining entries of S by increasing label order.

Deo and Micikevicius code. All the leaves of T, sorted by increasing labels, are at the beginning of sequence S. Then, all the internal nodes appear in the order in which they become leaves, i.e., sorted by increasing *rightmost*. It is possible to get a closed formula giving the position of each node. For

each $i \in [1, n-1]$, let $\rho(i)$ be 1 if i is the rightmost occurrence of node C_i, and 0 otherwise. Let $\sigma(i) = \sum_{j \leq i} \rho(j)$. The position of an internal node v is exactly $|leaves(T)| + \sigma(rightmost(v, C))$.

Our unified decoding algorithm is as follows:

DECODING ALGORITHM:
1. **for each** `node` v `compute` $rightmost(v, C)$
2. **for each** `node` v `except for the root` **do**
3. **if** ($test(v) =$ `true`) **then** $S[position(v)] \leftarrow v$
4. `let` L `be the list of nodes not yet assigned in increasing order`
5. `let` P `be the set of positions of` S `which are still empty`
6. **for each** $i = 1$ **to** `|P|` **do**
7. $S[P[i]] \leftarrow L[i]$

where $test(v)$ and $position(v)$ are specified in Table 2. With respect to Prüfer's code, the algorithm is essentially the same as the one described in [19], and we refer to [19, 9] for a detailed parallel analysis. As observed in Section 1, a linear sequential decoding algorithm for Prüfer codes is presented in [9], while the straightforward sequential implementation of our algorithm would require $O(n \log n)$ time. This can be easily reduced to $O(n)$ time by adapting the DECODING ALGORITHM in such a way that the *prev* computation can be avoided. With respect to codes `N3` and `DM`, the DECODING ALGORITHM runs in linear time.

In parallel, $\sigma(i)$ can be computed for each i using a prefix sum operation [12]. In order to get set L in step 4, we can mark each node not yet assigned to S and obtain its rank in L by computing prefix sums. Similarly for set P. Hence, the most expensive step is the rightmost computation, which requires integer sorting. This implies the following result:

Theorem 3. *Let C be a string of $n-1$ integers in $[1, n]$. Let $\mathcal{C}$ be the set of codes* `PR`*,* `N3`*, and* `DM`*. For each $i \in [1, n-1]$, let* `test(C[i])` *and* `position(C[i])` *be defined as in Table 2. For each code in $\mathcal{C}$, the* DECODING ALGORITHM *computes the tree corresponding to string C in $O(n)$ sequential time. Decoding on the EREW PRAM model requires $O(\log n)$ time with cost $O(n \log n)$ for code* `PR` *and $O(\log n)$ time with cost $O(n\sqrt{\log n})$ for codes* `N3` *and* `DM`*.*

Second Neville code. Differently from the other codes, in code `N2` the rightmost occurrence of each node in C gives only partial information about sequence S. Thus, we treat `N2` separately in this section. We first observe that if all nodes were assigned with a level, an ordering with respect to pairs $(l(v), v)$ would give sequence S, and thus the tree. We refer to Section 2 for details on the correctness of this approach. We now show how to compute $l(v)$.

Let x be the number of leaves of T, which have level 1 and rightmost occurrence 0. Consider the first x elements of code C, say $C[1], \ldots, C[x]$. For each i, $1 \leq i \leq x$, such that i is the rightmost occurrence of $C[i]$, we know that node $C[i]$ has level 2. The same reasoning can be applied to get level-3 nodes from level-2 nodes, and so on. With respect to the running time, a sequential scan

Table 2. Condition on node v that is checked in the DECODING ALGORITHM and position of v as a function of $rightmost(v, C)$.

	$test(v)$	$position(v)$
N3	true	$rightmost(v, C) + 1$
PR	$rightmost(v, C) \geq prev(v, C)$	$rightmost(v, C) + 1$
DM	true	$\|leaves(T)\| + \sigma(rightmost(v, C))$

of code C is sufficient to compute the level of each node in linear time. Integer sorting does the rest. Unfortunately, this approach is inherently sequential and thus inefficient in parallel.

Before describing our parallel approach, we note that the procedure for level computation described above can be applied also for code `DM`. Indeed, let T' be the tree obtained interpreting C as the code by Deo and Micikevicius and let S' be the corresponding sequence: although T and T' are different, they have the same nodes at the same levels and, both in S and S', nodes at level $i+1$ appear after nodes at level i, but are differently permuted within the level. In view of these considerations, we are able to solve our problem in parallel, using T' to get our missing level information. Namely, first we build tree T' using the DECODING ALGORITHM, then we compute node levels applying the Euler tour technique, and finally we obtain sequence S (corresponding to tree T) by sorting the pairs $(l(v), v)$. It is to remark that the Euler tour technique requires a particular data structure [12] that can be built as described in [10]. The bottleneck of this procedure is sorting of pairs of integers in $[1, n]$, and thus we can use the parallel integer sorting presented in [11]. We can summarize the results concerning code `N2` as follows:

Theorem 4. *Let C be a string of $n-1$ integers in $[1, n]$. The tree corresponding to C according to code* `N2` *can be computed in $O(n)$ sequential time and in $O(\log n)$ time with cost $O(n\sqrt{\log n})$ on the EREW PRAM model.*

4 Conclusions and open problems

We have presented a unified approach for coding labeled trees by means of strings of node labels and have applied it to four well-known codes: `PR` [18], `N2` [17], `N3` [17], and `DM` [5]. The coding scheme is based on the definition of pairs associated to the nodes of the tree according to some criteria dependent on the specific code. The coding problem is reduced to the problem of sorting these pairs in lexicographic order. The decoding scheme is based on the computation of the rightmost occurrence of each label in the code: this is also reduced to radix sorting. We have applied these approaches both in a sequential and in a parallel setting. We have completely closed the sequential coding and decoding problem, showing that both operations in all the four codes can be done in linear time. In the parallel setting, further work is still needed in order to improve all the

non optimal coding and decoding algorithms. We remark that any improvement on the computation of integer sorting would yield better results for our parallel algorithms.

References

1. Atallah, M.J., Cole, R., and Goodrich, M.T.: Cascading divide-and-conquer: a technique for designing parallel algorithms. *SIAM Journal of Computing*, 18(3), pp. 499–532, 1989.
2. Cayley, A.: A theorem on trees. *Quarterly Journal of Mathematics*, 23, pp. 376–378, 1889.
3. Chen, H.C. and Wang, Y.L.: An efficient algorithm for generating Prüfer codes from labelled trees. *Theory of Computing Systems*, 33, pp. 97–105, 2000.
4. T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein: *Introduction to algorithms*. McGraw-Hill, 2001.
5. Deo, N. and Micikevicius, P.: A new encoding for labeled trees employing a stack and a queue. *Bulletin of the Institute of Combinatorics and its Applications*, 34, pp. 77–85, 2002.
6. Deo, N. and Micikevicius, P.: Prüfer-like codes for labeled trees. *Congressus Numerantium*, 151, pp. 65–73, 2001.
7. Deo, N. and Micikevicius, P.: Parallel algorithms for computing Prüfer-like codes of labeled trees. *Computer Science Technical Report* CS-TR-01-06, 2001.
8. Edelson, W. and Gargano, M.L.: Feasible encodings for GA solutions of constrained minimal spanning tree problems. *Proceedings of the Genetic and Evolutionary Computation Conference*, Morgan Kaufmann Publishers, page 754, 2000.
9. Greenlaw, R., Halldorsson, M.M. and Petreschi, R.: On computing Prüfer codes and their corresponding trees optimally. *Proceedings Journees de l'Informatique Messine, Graph algorithms*, 2000.
10. Greenlaw, R. and Petreschi, R.: Computing Prüfer codes efficiently in parallel. *Discrete Applied Mathematics*, 102, pp. 205-222, 2000.
11. Han, Y. and Shen, X.: Parallel integer sorting is more efficient than parallel comparison sorting on exclusive write PRAMS. *SIAM Journal of Computing*, 31(6), pp. 1852–1878, 2002.
12. JáJá, J.: *An Introduction to parallel algorithms*. Addison Wesley, 1992.
13. Kelmans, A., Pak, I., and Postnikov, A.: Tree and forest volumes of graphs. *DIMACS Technical Report* 2000-03, 2000.
14. Kumar, V., Deo, N., and Kumar, N.: Parallel generation of random trees and connected graphs. *Congressus Numerantium*, 130, pp. 7–18, 1998.
15. Lo, W.T., and Peng, S.: The optimal location of a structured facility in a tree network. *Journal of Parallel Algorithms and Applications*, 2, pp. 43–60, 1994.
16. Moon, J.W.: *Counting labeled trees*. William Clowes and Sons, London, 1970.
17. Neville, E.H.: The codifying of tree structures. *Proceedings of Cambridge Philosophical Society*, 49, pp. 381–385, 1953.
18. Prüfer, H.: Neuer Beweis eines Satzes über Permutationen. *Archiv für Mathematik und Physik*, 27, pp. 142–144, 1918.
19. Wang, Y.L., Chen, H.C. and Liu, W.K.: A parallel algorithm for constructing a labeled tree. *IEEE Transactions on Parallel and Distributed Systems*, 8(12), pp. 1236–1240, 1997.
20. Zhou, G. and Gen, M.: A note on genetic algorithms for degree-constrained spanning tree problems. *Networks*, 30, pp. 91–95, 1997.