A transformation on ordered trees

B. Dasarathy* and Cheng Yang†‡

The paper introduces a new transformation on ordered trees which is used to compute a few average properties of ordered trees. The properties that are studied are: (*a*) the average path length of an ordered tree with *n* nodes, (*b*) the average number of terminal nodes of an ordered tree with *n* nodes and (*c*) its average height. An algorithm is also presented to effect this transformation on any ordered tree *T*. The algorithm is linear in the number of nodes in *T*.

(Received October 1978)

1. Introduction and preliminaries

Using a new transformation on ordered trees and Knuth's natural correspondence between an ordered tree and a binary tree (1968, pp. 332-333), a few combinatorial properties of ordered trees are explored. Given that an ordered tree has n nodes, the properties under consideration are: its average path length, its average number of terminal nodes and its average height.

The definitions given here are based on Knuth (1968). A binary tree is a finite set of nodes which is either empty or consists of a root and two disjoint binary trees called the *left subtree* and the *right subtree*. A tree T is a non-empty set of nodes such that

1. There is one specially designated node called the root of the tree, root (T), and
2. The remaining nodes (excluding the root) are partitioned into $m \geqslant 0$ disjoint sets $T_1, \ldots, T_m$, and each of these sets in turn is a tree. The trees $T_1, \ldots, T_m$ are called subtrees of the root.

If the relative order of the subtrees $T_1, \ldots, T_m$ in (2) of the definition is important, we say the tree is an *ordered tree*. For the sake of brevity, the term '*tree*' will be used for an ordered tree in subsequent discussion.

A *forest* is an ordered set of trees. Horowitz and Sahni (1976) have given the following transformation to represent a forest as a binary tree. If $T_1, \ldots, T_n$ is a forest of trees, then the binary tree corresponding to this forest, denoted by $B(T_1, \ldots, T_n)$,

(*a*) is empty if $n = 0$;

(*b*) has root equal to root (T_1); has left subtree equal to $B(T_{11}, T_{12}, \ldots, T_{1m})$ where $T_{11}, \ldots, T_{1m}$ are the subtrees of the root (T_1); and has right subtree $B(T_2, \ldots, T_n)$.

Conversely, it is easy to see that any binary tree corresponds to a unique forest of trees by reversing the process. This correspondence between a binary tree and a forest will be referred to as a 'natural correspondence' in this paper.

The *path* of a node v from the root (T) of a tree T is a sequence of nodes $v_0, v_1, v_2, \ldots, v_{k-1}, v_k$ such that v_0 = root (T), $v_k = v$ and each tree with v_{i+1} as the root is a subtree of the tree with root v_i, $i = 0, 1, \ldots, k-1$. The *path length* of a node v from the root of a tree is the number of nodes in the path of that node from the root minus 1. The *average path length* of a tree with n nodes is the sum of the path lengths of all nodes of all possible trees with n nodes divided by n and the number of trees. The *height* of a tree is defined to be the maximum of all the path lengths in the tree. The *average height* of a tree with n nodes is the average of heights of all possible trees with n nodes. The *degree* of a node is the number of subtrees of that node. A node of degree zero is called a *terminal node*. A *tree traversal* is a method to trace all the nodes of the tree.

In a similar manner, path, path length, average path length, height, average height, degree, terminal nodes and traversal can be defined for binary trees. The average path length and the average height of a binary tree with n nodes have been studied by Knuth (1968) (p. 590, p. 329). De Bruijn, Knuth and Rice (1972) have studied the average height of an ordered tree. The corresponding problem for an 'unordered tree' (free tree) has been solved by Renyi and Szekeres (1967).

The total path length of a tree, i.e. the sum of the path lengths of all nodes of a tree is a quantity of interest in many areas, such as information storage and retrieval and sorting and searching (Hibbard, 1962). The average number of terminal nodes is worth the investigation because the product of the average number of terminal nodes and the average height provides an upper bound on the (average) number of searches required if the information is stored in the terminal nodes alone.

2. A transformation on trees

A 'reflection-transformation' on a tree T will be defined using a reflection-transformation on a binary tree B. We obtain a *reflected binary tree* B^R from a binary tree B by interchanging for each node of B except its root the left and right subtrees of that node.

Using the natural correspondence between a tree and a binary tree as defined in Section 1 and the transformation on a binary tree given above, we define a *reflection-transformation* on a tree as follows:

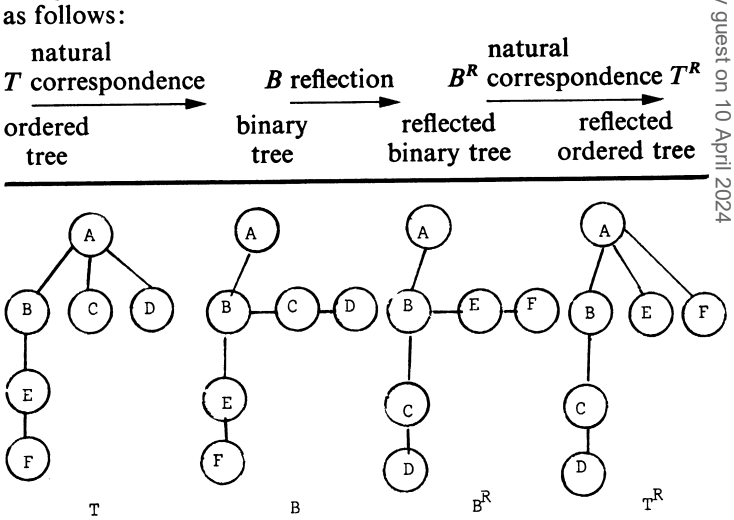
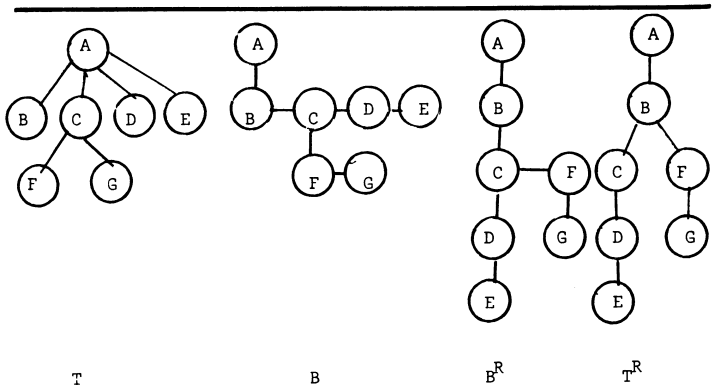


Fig. 1 Illustration of transformation from T to T^R

*Department of Mathematics, Computer Science and Statistics, University of South Carolina, Columbia, SC 29208, USA
†GTE Automatic Electric Laboratories, 11226 N. 23rd Avenue, Phoenix, Arizona, 85029, USA
‡The research reported here was conducted when the second author was a graduate student at the University of South Carolina



$P^n(T) + P^n(T^R) = 23 = P^n(B) + (n - 1)$

$P^n(T) = 1 + 1 + 1 + 1 + 2 + 2 + = 8$

$P^n(T^R) = 1 + 2 + 2 + 3 + 3 + 4 = 15$

$P^n(B) = 1 + 2 + 3 + 3 + 4 + 4 = 17$

$n = 7$

Fig. 2 Illustration of Theorem 1

We first use the natural correspondence to find a binary tree for the given ordered tree T, then we use the reflection-transformation to obtain B^R from B. We use the natural correspondence again to obtain T^R from B^R. The reflection-transformation thus defined produces a unique T^R for each T, and further maps T^R back to T. The example in **Fig. 1** illustrates the transformation.

3. Applications

We first compute the average path length of a tree as an application of our transformation. As noted in Section 1, the average path length of a tree with n nodes is the sum of the path lengths of all nodes of all possible trees with n nodes divided by n and the number of trees.

Theorem 1

Let T be a tree with n nodes, T^R be its reflected tree and B be the binary tree for T using the natural correspondence and let $P^n(T)$, $P^n(T^R)$ and $P^n(B)$ be the total path lengths of all nodes, of T, T^R and B respectively. Then

$$P^n(T) + P^n(T^R) = P^n(B) + n - 1 .$$

The statement of the theorem is made clear with the example in **Fig. 2.**

Proof

Let $P_v(B)$, $P_v(T)$, $P_v(T^R)$ stand for the path lengths from the root to a node v in B, T and T^R respectively.

Consider $P_v(B)$. In general, $P_v(B)$ consists of two kinds of edges, vertical and horizontal. It is easy to see by referring to Fig. 2 that the number of vertical edges in the path from the root to v in B is equal to the path length $P_v(T)$. Similarly the path length is equal to the number of vertical edges in the path $P_v(T^R)$ from the root to v in B^R. Since B^R is obtained from B by exchanging the left and the right subtrees of each node except the root, the number of vertical edges in the path from the root to v in B^R is equal to the number of horizontal edges in the path from the root to v in B plus 1. Thus

$$P_v(B) = P_v(T) + P_v(T^R) - 1 . \qquad (1)$$

Since $P^n(B) = \sum_{v \neq \text{root}} P_v(B)$, from (1), we obtain

$$P^n(B) = P^n(T) + P^n(T^R) - (n - 1), \text{ i.e.}$$

$$P^n(T) + P^n(T^R) = P^n(B) + (n - 1) .$$

Corollary 1

$P^n(T) + P^n(T^R) = P^{n-1}(B') + 2(n - 1)$, where B' is the $(n-1)$ node-binary tree obtained from B by deleting its root.

We now use the above corollary to compute the average path length of a tree with n nodes. The average path length of a tree is obtained as follows: we consider all possible n-node trees to be equally likely. For each tree, we compute the total path length of that tree. The average path length is the sum of the total path lengths, summed over all possible trees, divided by the product of the number of trees and n.

Theorem 2

The average path length of a tree with n nodes, considering all trees with n nodes to be equally likely, is

$$\approx 1/2 \sqrt{\pi n} - 1/4 \frac{\sqrt{\pi n}}{n} - 1/2 .$$

Proof

From corollary 1, we obtain

$$2 \sum_{T} P^n(T) = \sum_{B'} P^{n-1}(B') + 2(n - 1)^* b_{n-1} , \qquad (2)$$

where $\sum_T$ and $\sum_{B'}$ respectively stand for the summation over all possible trees with n nodes and all possible binary trees with $(n - 1)$ nodes and where b_{n-1} is the number of binary trees with $(n - 1)$ nodes.

But

$$\sum_{B'} P^{n-1}(B^1) = 4^{n-1} - (3n - 2)/n\binom{2n-2}{n-1} \text{ (Knuth, 1968, p. 590)} \qquad (3)$$

From (2) and (3), and using the known result

$$b_{n-1} = (1/n)\binom{2n-2}{n-1} \text{ (Knuth, 1968, p. 389)} \qquad (4)$$

we obtain

$$\sum P^n(T) = 1/2 \left[4^{n-1} - \binom{2n-2}{n-1}\right] .$$

Since the total number of trees with n nodes is also $\frac{1}{n}\binom{2n-2}{n-1}$ (Knuth, 1968, p. 389), the average path length of a tree with n nodes is then

$$(1/2)\left[4^{n-1}/\binom{2n-2}{n-1} - 1\right] . \qquad (5)$$

The proof of the theorem is concluded with the substitution of the stirling formula, $n ! \simeq n^n e^{-n}\sqrt{2\pi n}$, in (5).

Next, we compute the average number of terminal nodes in a tree with n nodes.

Theorem 3

Let T be a tree with n nodes, and let T^R be the tree obtained from T by the reflection-transformation. Then the number of terminal nodes in T and T^R, denoted by $E(T)$ and $E(T^R)$, satisfies

$$E(T) + E(T^R) = n . \qquad (6)$$

Proof

We use the natural correspondence to establish the following relation: a node in T is a terminal node if and only if under the natural correspondence the corresponding node in the binary tree B has no left son, and a node in T^R is a terminal node if and only if the corresponding node (except the root) in B has no right son.

Thus

$$E(T) + E(T^R) = 2^* d_0 + 1^* d_1 + 0^* d_2 - 1 ,$$

where d_i is the number of nodes in the binary tree B having degree i. Since B is a tree with n nodes, we have

$$\text{the number of edges} = 2^* d_2 + d_1 = n - 1, \qquad (7)$$

and

$$\text{the number of nodes} = d_0 + d_1 + d_2 = n . \qquad (8)$$

From (7) and (8) we obtain

$$d_1 + 2d_0 - 1 = n ,$$

and

$$E(T) + E(T^R) = 2d_0 + 1*d_1 + 0*d_2 - 1$$
$$= n .$$

From the above theorem, we state this useful and interesting corollary.

Corollary 2
The average number of terminal nodes of a tree with n nodes is $n/2$.

Proof
Let k be the number of trees with n nodes and let the k trees be denoted as $T_1, T_2, \ldots, T_k$. Summing (6) over all possible trees with n nodes, we obtain

$$\sum_{i=1}^{k} E(T_i) + \sum_{i=1}^{k} E(T_i^R) = \sum_{i=1}^{k} n . \quad (9)$$

Since $\sum E(T_i) = \sum E(T_i^R)$, (9) reduces to

$$2 \sum_{i=1}^{k} E(T_i) = k*n . \quad (10)$$

Assuming all trees to be equally likely, i.e. dividing (10) by k, the average number of terminal nodes in a tree with n nodes is then $n/2$.

The following corollary provides a sufficient condition for a tree with n nodes to have exactly $n/2$ terminal nodes.

Corollary 3
If the reflection-transformation of a tree T is the same as T, i.e. $T = T^R$, then it has exactly $n/2$ terminal nodes.

Proof
The proof is trivially obtained by substituting $T^R = T$ in (6).

As noted in Section 1, de Bruijn, Knuth and Rice (1972) have computed the average height of a tree with n nodes. Knuth (1968, p. 329) has also studied the average height of a binary tree using the generating-function technique. However, as an application of our reflection-transformation, we relate these two quantities in the form of an inequality.

The height of a tree is defined to be the maximum of the path lengths from the root to the terminal nodes of the tree. In a binary tree consisting of vertical and horizontal edges, the vertical (horizontal) height $hV(hH)$, is the maximum number of vertical (horizontal) edges among all paths from the root to the nodes.

It can be seen that the height $h^n(T)$ of a tree with n nodes is equal to $hV^{n-1}(B') + 1$, where B' is the binary tree obtained from B by deleting its root (see Fig. 2), i.e.

$$h^n(T) = hV^{n-1}(B') + 1 . \quad (11)$$

Similarly, we note

$$h^n(T^R) = hH^{n-1}(B') + 1 . \quad (12)$$

From (11) and (12), we obtain

$$h^n(T) + h^n(T^R) \geqslant h^{n-1}(B') + 1 . \quad (13)$$

Summing (13) over all trees with n nodes and all binary trees with $(n-1)$ nodes, we obtain

$$2 \sum_{T} h^n(T) \geqslant \sum_{B'} h^{n-1}(B') + b_{n-1} . \quad (14)$$

The inequality (14) yields the following theorem relating the heights of a tree and a binary tree.

Theorem 4
The average height of a binary tree with $(n-1)$ nodes is less than twice the average height of an ordered tree with n nodes.

4. A linear algorithm to obtain T^R

In this section we suggest an algorithm to obtain T^R from T. The algorithm is linear in the number of nodes in T. Since the reflection-transformation of T^R is T, the algorithm also maps T^R back to T. Such a linear algorithm will be very useful in an application where T^R might be preferred in place of T. Consider a situation where, for a given tree T, the height of T^R is much less than the height of T and the nodes of T have to be traversed often. Since the height of a tree determines the maximum stack size used in algorithms that traverse the tree (Knuth, 1968, pp. 317-318, p. 329), T^R may be a more appropriate choice for traversal than T.

List for node	*Father*	*Sons and links to Son's List*								
A	*A*	*B**	link to *B*'s list	*C*	link to *C*'s list	*D*	link to *D*'s list	*E*	link to *E*'s list	0
B	*B**	0								
C	*C*	*F*	link to *F*'s list	*G*	link to *G*'s list	0				
D	*D*	0								
E	*E*	0								
F	*F*	0								
G	*G*	0								

*This duplication of a node value in two places is not really needed.

The skeleton algorithm given below will produce a similar adjacency list for T^R.

Fig. 3 Adjacency list for T in Fig. 2

Our algorithm to obtain T^R from T is based on the following fact: in the reflected tree T^R, a node X has its sons $Y_1, Y_2, \ldots, Y_k$ if and only if Y_i is the 'first son' of Y_{i-1} in the original tree T for all $i \geqslant 2$ and if X is the root in T, then Y_1 is the first son of X and if X is not the root, then Y_1 is the 'next right brother'. The input to the algorithm is an adjacency list for T. For example, the tree T in Fig. 2 will have the representation shown in **Fig. 3.**

1. *Algorithm to obtain T^R from T*
Let $S(1), S(2), \ldots, S(n-1)$ be a $(n-1)$ dimensional vector, X and Y two temporary variables and K a pointer.

1. (*set the root of T^R and its first son*)
Set $X \leftarrow$(value at the) root of T, $Y \leftarrow$(value of the) first son of X and $K \leftarrow 0$ (if X does not have any son, go to step 3; the tree has only one node).

2. (*get all the remaining sons of X*)
$K \leftarrow K+1$, $S(k) \leftarrow Y$, $Y \leftarrow$first son of Y. Repeat this step until Y does not have any son, i.e. $Y\neg = 0$.

3. (*add the list for X to the adjacency list of T^R being formed*)
If $K\neg = 0$, then append the entry. '$X\ S(1)\ S(2) \ldots S(k)\ 0$' to the adjacency list of T^R; else the entry to be added is '$X\ 0$'.

4. (*get the node next to X in the adjacency list of T and set its first son*)
If the adjacency list of T is not exhausted, set $X \leftarrow$next non-zero element in the adjacency list, $Y \leftarrow$next brother of X and $K \leftarrow 0$; else stop.

5. If $Y\neg = 0$, then go to step 2; else go to step 3.

Next, we analyse the algorithm for its computational growth. For any tree, step 1 takes only a constant amount of time, since it is executed just once in setting up the root and its first son in

T^R. Step 2, in essence, computes all but the leftmost son of a node X. Each of these sons is obtained with one table look-up. The total number of sons cannot exceed n, since there are only that many nodes in the entire tree. Thus the growth of step 2 is $0(n)$. Step 3 merely sets up the sublist of sons for each node in the adjacency list of T^R. Hence step 3 also takes $0(n)$ time. In step 4, we find the first son of each node (except the root). The first son of a node in T^R is the next node of the node under consideration in the adjacency list of T. Hence the growth of step 4 is also linear. We determine in step 5 whether a node in T^R has a son other than the leftmost son found in step 4. This step is executed exactly $(n - 1)$ times. Thus the growth of the entire algorithm is $0(n)$.

It can easily be seen that the storage requirement of the algorithm is also linear in the number of nodes. In step 3 of the algorithm, the entry $X\ S(1)\ S(2) \ldots S(K)\ 0$ is added to the adjacency list of T^R. However, each $S(I)$ has to be accompanied by a link where the link should point to the beginning of the sublist where $S(I)$'s sons are stored. But this pointer to the sons of $S(I)$ will not be available when the sons of X are computed. The following modification to the algorithm will get around this problem; the growth of the algorithm will still be linear. In step 0 of the algorithm, as a preprocessing step, change the values of the nodes to 1, 2, . . . , N in the adjacency list of T. Store these values in a one dimensional array such that if a value v is changed to the integer J, then $S(J)$ contains v. This array will be used to obtain the original values of the nodes at the end of the algorithm. Let R be a $n \times 2$ array. The array entry $R(J, 1)$ can then be used to store the location of the link for the list of sons of node J in the adjacency list of T^R. The entry $R(J, 2)$ can be used to store the actual value of the link to the sons of J when it becomes available. Storing this information in R takes only a constant amount of time for each J. When we exit the algorithm in step 4, we can complete filling in the links in the adjacency list of T^R using R again. This can be done in linear time for all links.

5. Acknowledgements

We wish to express our appreciation to Professor Carter Bays of the University of South Carolina for his many helpful suggestions in improving the readability of this paper.

References

DE BRUIJN, N. G., KNUTH, D. E. and RICE, S. O. (1972). The average height of planted plane trees, *Graph theory and computing*, edited by R. C. Read, Academic Press, New York and London, pp. 15-22.

DEBRUIJN, N. G. and MORSELT, B. J. M. (1976). A note on plane trees, *Jour. of Comb. Theory*, Vol. 2, pp. 27-34.

HARARY, F., PRINS, G. and TUTTE, W. T. (1964). The number of plane trees, Neder Akad. Wetensch. Proc. Ser A67 (*Indag. Math.*, Vol. 26, pp. 319-329).

HIBBARD, T. N. (1962). Some combinatorial properties of certain trees with applications to searching and sorting, *JACM*, Vol. 9 No. 1, pp. 13-18.

HOROWITZ, E. and SAHNI, S. (1976). *Fundamentals of Data Structures*, Computer Science Press, Inc., 4566 Poe Avenue, Woodland Hills, Calif.

KLARNER, D. A. (1970). Correspondence between plane trees and binary sequences, *Jour. of Comb. Theory*, Vol. 9, pp. 401-411.

KNUTH, D. E. (1968). *The Art of Computer Programming*, Vol. 1: Fundamental Algorithms, Addison-Wesley, Reading, MA.

RENYI, A. and SZEKERES, G. (1967). On the height of trees, *Austral. J. Math.* Vol. 7, pp. 497-507.

Book reviews

Machine and Assembly Language Programming of the PDP-11 by Arthur Gill, 1979; 191 pages. (*Prentice-Hall*, £10·75)

This book is an expensive, slim volume which gives a very sound introduction to the PDP-11 machine organisation and programming. The author recommends that the reader be equipped with the manufacturer's processor, assembler and peripherals handbooks in order to undertake the exercises set in the text and also expects the reader to have encountered the basic concepts of algorithms, flowchart and stored program computer.

He commences with a strange first chapter which is a general review of number systems and various conversion algorithms—the reviewer would, however, admit its usefulness. It might have been better as an appendix referred to as necessary. The second chapter introduces the basics of PDP-11 machine organisation showing how the memory can be addressed by byte and by word (two bytes)—the even addresses. The general machine registers and peripheral registers are also introduced. The peripherals handled are limited to the console keyboard, console printer and line clock. The third chapter contrasts numbers and characters introducing integer, floating point and string representations. It is the reviewer's experience that students always have difficulty distinguishing coded representations from numeric value representations and the early emphasis on this in this text is to be applauded. The fourth chapter completes the groundwork for the rest of the text, dealing in full with the differing instruction forms and lengths and the various forms of addressing. The chapter includes several example coding sequences.

Having completed the machine code treatment, Chapter 5 introduces assembly language programming. Chapter 6 deals with stacks and subroutines introducing the system stack for the control of subroutine linkage and exit. The remaining chapters are: arithmetic operations, traps and interrupts, Assembler and Linkage Editor, Advance (Macro) Assembly techniques.

This is a well developed text with well developed examples. The author emphasises good programming style and dedicates an appendix to this. The book is to be recommended for anyone wishing to learn something of the PDP-11 class of computer.

A. H. WISE (Leicester)

Assembly Language Fundamentals by Rina Yarmish and J. Yarmish, 1978; 768 pages. (*Addison-Wesley*, £13·50)

This book sets out to teach Assembler for IBM/360 or /370 machines, under OS/VS and DOS/VS. The material is introduced gradually and is well explained, with many examples and copious exercises.

My only criticism is that it is rather too much of a good thing. The authors start from square one, assuming that the reader may not even know what a computer is like. Perhaps they would have served most of their readers by assuming at least some background in computing. Another economy could have been made by making the book more of a bridge to the manufacturer's manuals. As it is, the book cannot hope to cover everything, and the reader is in the end referred to the manuals.

If the aim of the book had been thus curtailed, it would have been just about as useful, though greatly reduced in bulk and price. But perhaps it is in the publishers' interest to produce expensive tomes?

A. COLIN DAY (London)