Journal of Algorithms and Computation

journal homepage: http://jac.ut.ac.ir

A novel algorithm to determine the leaf (leaves) of a binary tree from its preorder and postorder traversals

N. Aghaieabiane*[1], H. Koppelaar†[2] and P. Nasehpour‡[3]

[1]Department of Engineering, School of Computer Science, New Jersey Institute of Technology, Newark, New Jersey, the USA.
[2]Faculty of Electrical Engineering, Mathematics and Computer Science, Delft University of Technology, Delft, The Netherlands.
[3]Department of Engineering Science, Golpayegan University of Technology, Golpayegan, Iran.

ABSTRACT

Binary trees are essential structures in Computer Science. The leaf (leaves) of a binary tree is one of the most significant aspects of it. In this study, we prove that the order of a leaf (leaves) of a binary tree is the same in the main tree traversals; preorder, inorder, and postorder. Then, we prove that given the preorder and postorder traversals of a binary tree, the leaf (leaves) of a binary tree can be determined. We present the algorithm BT-LEAF, a novel one, to detect the leaf (leaves) of a binary tree from its preorder and postorder traversals in quadratic time and linear space.

Keyword: Binary tree, Proper binary tree, Preorder traversal, Inorder traversal, Postorder traversal, Time complexity, Space complexity

ARTICLE INFO

Article history:
Received 30, June 2017
Received in revised form 18, November 2017
Accepted 30 November 2017
Available online 01, December 2017

AMS subject Classification: 05C78.

*niloofaraghaie@ut.ac.ir, na396@njit.edu
†Koppelaar.Henk@gmail.com
‡Corresponding author: nasehpour@gut.ac.ir, nasehpour@gmail.com

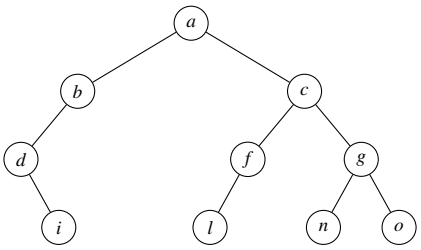

Figure 1: A binary trees. Preorder traversal: *a, b, d, i, c, f, l, g, n, o*, inorder traversal: *d, i, b, a, l, f, c, n, g, o*, and postorder traversal: *i, d, b, l, f, n, o, g, c, a*. The leaves are colored red. The order of the leaves are the same in all the three traversals.

1 Introduction

Binary trees [8, 12, 13] are fundamental data structures in computer science [3]. Leaves of a binary tree, in turn, are one of the most important aspects of a binary tree. As an example, the leaves of a decision tree exhibit the content [1, 10] for such classification. Tree traversals, also, are among the most significant aspects and uses of binary trees. Several algorithms have been proposed to reconstruct a binary tree from its traversals, for instance, [4, 6, 7, 14, 11, 5]. There are three main tree traversals: inorder, preorder, and postorder, which are together called tree walks as well [8].

In this study, we explore traversals of binary trees, to present a novel algorithm BT-LEAF combining two of these three traversal methods to detect the leaf (leaves) of a binary tree. In fact, we use its preorder and postorder traversals.

In section Order of the leaf (leaves) of a binary tree in the tree traversals, we show that there is a relationship between leaves of a binary tree in their traversals. In section Proposing algorithm, we present the BT-LEAF algorithm to determine the leaf of a binary tree using its preorder and postorder traversals. In section Results, the time and space complexity of the algorithm BT-LEAF is presented. Finally, in section Conclusion, our conclusion is given.

2 Order of the leaf (leaves) of a binary tree in the tree traversals

In this section, we show the relationship between the order of the leaf (leaves) of a binary tree in its traversals. We first ask, if there is any relationship between the order of the leaf (leaves) of a binary tree in its traversal. As it is seen from the Figure 1, the leaves of the binary tree appear in the same order. In other words, firstly the node with value *i*, secondly the node with value *l*, then the node with value *n*, and finally the node with value *o* appears in the three traversals. In fact, the order of leaves of a binary tree in the tree traversals is the same. Theorem 1 reflects this property.

Theorem 1. *If* x, y, *and* z *are the leaves of a binary tree such that* x *and* z *are the leftmost and rightmost, respectively, and* y *is in the middle, then the order of the leaves in preorder, inorder,*

and postorder traversals, in general, is ...,x,...,y,...,z,....

Proof. Without loss of generality, let x, y,and z be the left child of the nodes u, v, and w, respectively.

- According to preorder traversal, since the nodes x and z are the leftmost and rightmost leaves of a binary tree, in general, we have ...,u,x,...,v,y,...,u,z,..., that can be considered as ...,x,...,y,...,z,....
- According to inorder traversal, since the nodes x and z are the leftmost and rightmost leaves of a binary tree, in general, we have ...,x,u,...,y,v,...,z,w, which can be considered as ...,x,...,y,...,z,....
- According to postorder traversal, since the nodes x and z are the leftmost and rightmost leaves of a binary tree and the node y is in the middle, in general, we have ...,x,...,u,...,y,...,v,...,z,...,w,..., which can be considered as ...,x,...,y, ..., z.

Thus the nodes x, y,and z which are leaves of the a binary tree will be visited with the same order in the three tree traversals. □

It should be noted that although in the Theorem 1, we assume that tree has three leaves, this can be to generalized to less or more than three leaves.

3 Proposing algorithm

In this section, we present the BT-Leaf algorithm. As we proved in Order of the leaf (leaves) of a binary tree in the tree traversals, in three traversals the order of the leaves are the same. Here, we propose BT-Leaf algorithm to determine the leaf (leaves) of a binary tree using its preorder and postorder traversals.
The algorithm works by considering each two consecutive elements in preorder traversal, say x, y. It, then, looks for the order of x and y in postorder traversal. If the element y appears after the element x, then it marks the element x as a leaf node, otherwise x is not a leaf. In effect, Theorem 2 reflects this property.

Theorem 2. *If the elements* x, y *are two consecutive elements in preorder traversal and the element* y *appears after the element* x*, then the node* x *is a leaf.*

Proof. On the contrary, let x be no leaf. We have the two following cases:

- The element x is a node with only one child. We have the two following cases:
  1. The element y is the child of x. Without loss of generality we let y is its left child. According to preorder traversal, in general, we have ...,x,y,..., whereas based on postorder traversal, in general we have ...,y,x,.... As in postorder traversal, the element y appears before the element x, it is contradiction.

2. We let child of the element x be the element z and the element y is placed elsewhere. According to preorder traversal, in general, we have either $\ldots,x,z,\ldots,y,\ldots$, or $\ldots,y,\ldots,x,z,\ldots$. As the element y does not appear immediately after the element x or the element y appears before the element x, it is a contradiction.

- The element x is a node which has exactly two children. We have the three following cases:

1. The element x has the left and right children y and z respectively. According to preorder traversal, in general, we have $\ldots,x,y,\ldots,w,\ldots$,and based on postorder traversal, in general, we have $\ldots,y,\ldots,w,\ldots,x,\ldots$. As the element y appears before the element x, it is a contradiction.
2. The element x has the left and right children z and y respectively. According to preorder traversal, in general, we have $\ldots,x,z,\ldots,y,\ldots$. As two elements x and y are not two consecutive elements in preorder, it is a contradiction.
3. The element x has the left and right children u and v respectively, and the element y is placed elsewhere. According to preorder, in general, we have either $\ldots,y,\ldots,x,u,\ldots,v,\ldots$
or $\ldots,x,u,\ldots,v,\ldots,y,\ldots$. As in the former, the element y appears before the element x and in the later, there are at least two elements between the elements x and y, it is a contradiction.

Thus, if the elements x and y are two consecutive elements in preorder traversal, and in postorder traversal the element y appears after the element x, then the element x is a leaf. □

From Theorem 2, all the leaf elements can be determined except the last element in the preorder traversal. Based on Property 1, it is evident that the last element of preorder traversal is always a leaf. So, the last element of preorder traversal must be a leaf node.

Property 1. *The last element of the preorder is a leaf.*

3.1 BT-Leaf algorithm

Here, we present the pseudo code of BT-Leaf using the pseudo code style defined in [8]. First, without loss of generality, we let all the values be distinct. The two arrays $Pre[1..n]$ and $Pos[1..n]$ denote preorder and postorder traversals, respectively. The indices i and j denote position of the elements in preorder and postorder traversals, respectively. In each iteration, the algorithm considers two consecutive elements in preorder traversal using the index i, then it checks the order of these two consecutive elements in postorder traversal using the index j. In each iteration, the $flag$ defines the order of two consecutive elements considered in preorder, in postorder traversal. For two consecutive elements in preorder traversal, say x,y, if the $flag = 0$ means the element x appears after the element y in postorder traversal, otherwise, the element x appears before the element x in postorder traversal. So, when the $flag = 1$, the element x is a leaf.

Algorithm 1 Determining a leaf (leaves) of a binary tree from its preorder and postorder traversals.

BT-Leaf(Pre, Pos)

```
1  i = 2
2  j = 1
3  k = 1
4  flag = 0
5  while i ≤ Pre.length − 1
6      flag = 0
7      while temp_1 ≠ Pos[j] and j ≤ Pos.length
8          j = j + 1
9      while temp_2 ≠ Pos[j] and j ≤ Pos.length
10         j = j + 1
11     if j ≤ Pos.length
12         flag = 1
13     if flag == 0
14         i = i + 1
15     else leaves[k] = i
16         k = k + 1
17         i = i + 1
18     j = 1
19     flag = 0
20 leave[k] = Pre.length
```

Algorithm 1 shows the BT-Leaf. This algorithm gives two arrays as a preorder and postorder traversals, Pre and Pos in order. Since the first element in preorder traversal is the root, the algorithm starts considering each two consecutive elements in preorder from the immediate element after the root element, so the initial value of i is 2. In each iteration of the **while** loop in line 5, two consecutive elements are considered, where $Pre[i]$ and $Pre[i+1]$ indicate these two elements. The **while** loop in line 7 searches for the $Pre[i]$ in postorder traversal. The **while** loop in line 9 searches the $Pre[i+1]$ in postorder traversal. If the $Pre[i+1]$ do not appear after $Pre[i]$ in postorder traversal, then it should occur before this element. So, after finding the element $Pre[i]$, the algorithm only needs to check the elements after this element in postorder traversal. In other words, By finding the $Pre[i]$ and searching the elements after this element in postorder traversal, the algorithm can define the order of $Pre[i]$ and $Pre[i+1]$ in postorder traversal. If the element $Pre[i]$ appears before the element $Pre[i]$, then the algorithm initiates the $flag$ to 1, which means the element $Pre[i]$ is a leaf. At the end of each iteration of the **while** loop in line 5, the algorithm checks the $flag$. If the $flag$ is equal to 1, then it initiates the array $leave$ at index i as a defined leaf element.

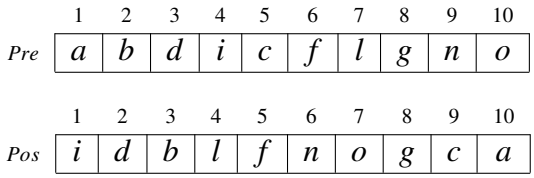

Figure 2: The initial step. *Pre* and *Pos* denote preorder and postorder traversals.

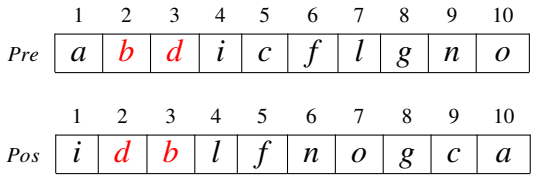

Figure 3: The first iteration. The two consecutive elements b and d are considered in *Pre*. The reverse order of these two elements is found in *Post*.

3.2 Example of BT-Leaf algorithm

In this section, we exemplify how the algorithm BT-Leaf works. Each figure shows one iteration of the **while** loop in line 5 and selected elements are colored red. Once again, consider the tree illustrated in Figure 1.

Figure 2 shows the initial step of the algorithm. *Pre* and *Pos* arrays denote preorder and postorder traversal. Figure 3 shows the first iteration of the **while** loop in line 5. The two consecutive elements of b and d are considered in preorder traversal, the reverse order of these two elements is found in postorder traversal, so the element b is not a leaf. Figure 4 shows the second iteration of the **while** loop in line 5. The two consecutive elements of d and i are considered in preorder traversal, the reverse order of these two elements is found in postorder traversal, so the element d is not a leaf. Figure 5 shows the third iteration of the **while** loop in line 5. The two consecutive elements of i and c are considered in preorder traversal, the same order of these two elements is found in postorder traversal, so the element i is a leaf. Figure 6 shows the next iteration. The two consecutive elements of c and f are considered in preorder traversal, the reverse order of these two elements is found in postorder traversal, so the element c is not a leaf. Figure 7 shows the next iteration. The two consecutive elements of f and l are considered

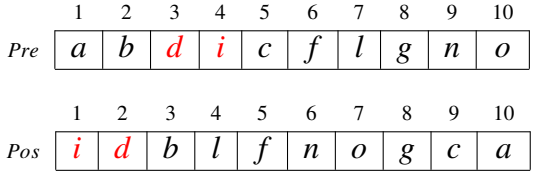

Figure 4: The second iteration. The two consecutive elements d and i are considered in *Pre*. The reverse order of these two elements is found in *Pos*.

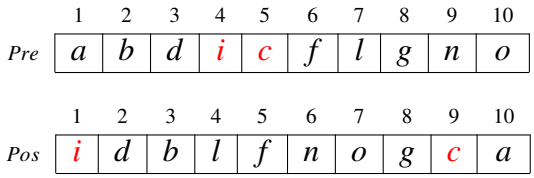

Figure 5: The third iteration. The two consecutive elements i and c are considered in Pre. The same order of these two elements is found in Pos.

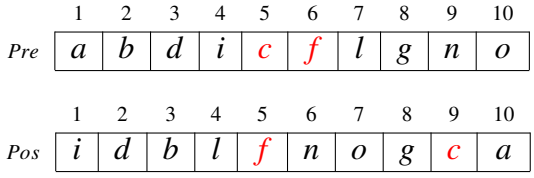

Figure 6: The next iteration. The two consecutive elements c and f are considered in Pre. The reverse order of these two elements is found in Pos.

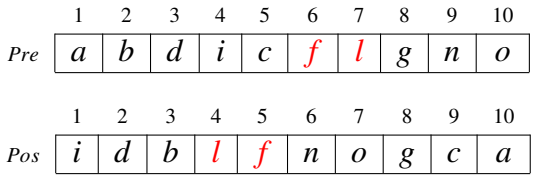

Figure 7: The next iteration. The two consecutive elements f and l are considered in Pre. The reverse order of these two elements is found in Pos.

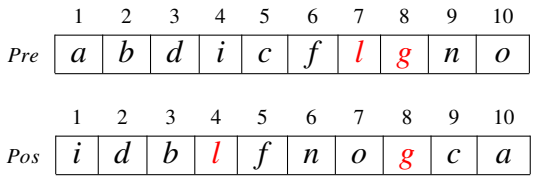

Figure 8: The next iteration. The two consecutive elements l and g are considered in Pre. The same order of these two elements is found in Pos.

Figure 9: The next iteration. The two consecutive elements g and n are considered in Pre. The reverse order of these two elements is found in Pos.

Figure 10: The last iteration. The two consecutive elements n and o are considered in Pre. The same order of these two elements is found in Pos.

in preorder traversal, the reverse order of these two elements is found in postorder traversal, so the element f is not a leaf. Figure 8 shows the next iteration. The two consecutive elements of l and g are considered in preorder traversal, the same order of these two elements is found in postorder traversal, so the element l is a leaf. Figure 9 shows the next iteration. The two consecutive elements of g and n are considered in preorder traversal, the reverse order of these two elements is found in postorder traversal, so the element g is not a leaf. Figure 10 shows the last iteration. The two consecutive elements of n and o are considered in preorder traversal, the same order of these two elements is found in postorder traversal, so the element n is a leaf. According to Property 1, the last element is preorder traversal (i.e o) is considered as a leaf.
In brief, Figure 2 to Figure 10 show how the BT-Leaf works on the binary tree illustrated in Figure 1.

4 Results

In this section, we explore the time and space complexity of the BT-Leaf separately.

4.1 Time complexity of BT-Leaf

The time complexity of the algorithm BT-Leaf depends on the three **while** loops in lines 5, 7, and 9. The **while** loops in lines 7 and 9 run for at most n time (i.e. $O(n)$), where n is the number of elements in one of the traversals. The **while** loop in line 5 runs for $n-1$ times. So, if $T(n)$ denotes the time complexity of BT-Leaf, the total complexity of the algorithm is equal to the times which are taken by the two **while** loops in line 7 and 11, as well as the **while** loops in line 5. We thus have

$$\begin{aligned} T(n) &= (n-1)\cdot O(n) \\ &= O(n^2) \end{aligned} \tag{1}$$

where n is the number of elements in of the traversals.

4.2 Space complexity of BT-Leaf

The space complexity of the algorithm BT-Leaf depends on the spaces which are taken by *Pre*, *Pos*, and *leaves* arrays. Each of the arrays of *Pre* and *Pos* take exactly n space, where n is the number of elements in one of the traversals. Hence, the space taken by these two arrays is equal to $n+n=2n$ which is $\theta(2n)=\theta(n)$. The array *leaves* takes $\theta(m)$, where m is the number of leaves. Therefore, if $S(n)$ denote space complexity, where n is the number of elements in of the traversal, the total space complexity is equal to

$$\begin{aligned} S(n) &= \theta(n)+m \\ &= \theta(n). \end{aligned} \tag{2}$$

It is evident that the number of leaves is always less than the total elements in one of the traversals.
Thus, from Equation 1 and Equation 2 the total time and space complexity of the BT-Leaf are equal to $T(n)=\mathrm{O}(n^2)$ and $S(n)=\theta(n)$, respectively.

5 Conclusion

Binary trees [8, 12, 13] are fundamental data structures in computer science [3]. Leaves of a binary tree, in turn, are one of the most important aspects of a binary tree. As an example, the leaves of a decision tree exhibit the content [1, 10] for such classification. Tree traversals, also, are among the most significant aspects and uses of binary trees. Several algorithms have been proposed to reconstruct a binary tree from its traversals, for instance, [4, 6, 7, 14, 11, 5, 2, 9, 3]. Here, we have proved that the order of a leaf (leaves) of a binary tree is the same in the three main tree traversals, Theorem 1. We have, then, showed that, in preorder and postorder traversals of a binary tree, if the node x and y are as two consecutive elements in preorder, and the element x appears after the element y in postorder traversal, then x is a leaf. Otherwise, it is not a leaf, Theorem 2. Regarding Theorem 2, the algorithm BT-Leaf can determine the leaf (leaves) of a binary tree in quadratic time and linear space.

References

[1] E. Alpaydin. *Introduction to Machine Learning*. The MIT Press, 2nd edition, 2010.

[2] A. Andersson and S. Carlsson. Construction of a tree from its traversals in optimal time and space. *Information Processing Letters*, 34(1):21–25, 1990. doi:10.1016/0020-0190(90)90224-L.

[3] N. Arora, P. K. Kaushik, and S. Kumar. Iterative method for recreating a binary tree from its traversals. *International Journal of Computer Applications*, 57(11):6–13, November 2012. doi:10.5120/9156-2056.

[4] H. Burgdorff, S. Jajodia, F. N. Springsteel, and Y. Zalcstein. Alternative methods for the reconstruction of trees from their traversals. *BIT Numerical Mathematics*, 27(2):133–140, 1987. doi:10.1007/BF01934177.

[5] R. Cameron, B. Bhattacharya, and E. Merks. Efficient reconstruction of binary trees from their traversals. *Applied Mathematics Letters*, 2(1):79–82, 1989. doi:10.1016/0893-9659(89)90122-5.

[6] G. Chen, M. Yu, and L. Liu. Nonrecursive algorithms for reconstructing a binary tree from its traversals. In *Computer Software and Applications Conference, 1988. COMPSAC 88. Proceedings., Twelfth International*, pages 490–492. IEEE, 1988. doi:10.1109/CMPSAC.1988.17225.

[7] G.-H. Chen, M. Yu, and L.-T. Liu. Two algorithms for constructing a binary tree from its traversals. *Information processing letters*, 28(6):297–299, 1988. doi:10.1016/0020-0190(88)90177-9.

[8] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press, 3rd edition, 2009.

[9] V. V. Das. A new non-recursive algorithm for reconstructing a binary tree from its traversals. In *Advances in Recent Technologies in Communication and Computing (ARTCom), 2010 International Conference on*, pages 261–263. IEEE, 2010. doi:10.1109/ARTCom.2010.88.

[10] L. I. Kuncheva. *Combining Pattern Classifiers: Methods and Algorithms*. Wiley-Interscience, 2004.

[11] E. Mäkinen. Constructing a binary tree from its traversals. *BIT Numerical Mathematics*, 29(3):572–575, 1989. doi:10.1007/BF02219241.

[12] D. R. Mazur. *Combinatorics: A Guided Tour*. MAA Textbooks. MAA, 2010.

[13] S. S. Skiena. *The Algorithm Design Manual*. Springer Publishing Company, Incorporated, 2nd edition, 2008.

[14] W. Slough and K. Efe. Efficient algorithms for tree reconstruction. *BIT Numerical Mathematics*, 29(2):361–363, 1989. doi:10.1007/BF01952690.