Iterative Method for Recreating a Binary Tree from its Traversals

Nitin Arora
Assistant Professor
Dept. of Computer Sci. & Engg.
Uttarakhand Technical University
Women Institute of Technology (WIT), Dehradun

Pradeep Kumar Kaushik
Assistant Professor
Dept. of Computer Sci. & Engg.
Uttarakhand Technical University
Shivalik College of Engineering (SCE), Dehradun

Satendra Kumar
Assistant Professor
Dept. of Computer Sci. & Engg.
Uttarakhand Technical University
Seemant Institute of Technology (SIT), Pithoragarh

ABSTRACT
Many reconstruction algorithms for binary tree have been discussed in this paper. A particular focus of this paper is on *"A new Non-Recursive Algorithm for Reconstructing a Binary Tree from its Traversals".* The computation time required for executing the reconstruction algorithm are *O(N)* and space complexity is *O(NlogN)* where N is the number of nodes in the binary tree. This algorithm works well for most of the input cases, but it has some drawbacks. There are some sequences of pre-order and in-order for which no legitimate tree can be constructed but the algorithm didn't take these cases into consideration and constructed a wrong tree for these cases.
In this paper, we have proposed a solution to the problem in the previous algorithm and designed an algorithm which is the modified version of the previous algorithm for generating a correct binary tree. The new modified algorithm is implemented in C language and tested in **GCC Compiler** in **Linux**, for all types of input cases. The New modified algorithm works well for all types of input cases. We have calculated the best case time complexity of modified algorithm and show that a correct tree can be reported in ***O*(*N*)** time in best case and ***O*(*NlogN*)** space where N is the number of nodes in the tree. We have discussed some applications of the new modified algorithm in Huffman Coding, compiler design, text processing and searching algorithms.

Key words
Non-recursive; tree traversals; binary tree.

1. INTRODUCTION
A *tree* is a fundamental structure in computer science. Almost all operating systems store files in trees or tree like structures. It is well known that given the in-order traverse of a binary tree, along with one of its pre-order or post-order traversals, the original binary tree can be uniquely identified. It is not difficult to write a recursive algorithm to reconstruct the binary tree. Most text books and reference books present the recursive [1, 2, 3, 4, 5] and non-recursive algorithms [6, 7, 8, 9, 10] for traversing a binary tree in in-order, post-order and pre-order.
Many iterative methods [11, 12, 13, 14, 15] for reconstructing a binary tree from its traversals have been proposed till date for which computation time required is $O(N)$ where N is the number of nodes in the tree.
A binary tree is different recursively as either empty or consists of a root, a left tree and a right tree. The left and right trees may themselves be empty, thus a node with one child could have a left or right child.

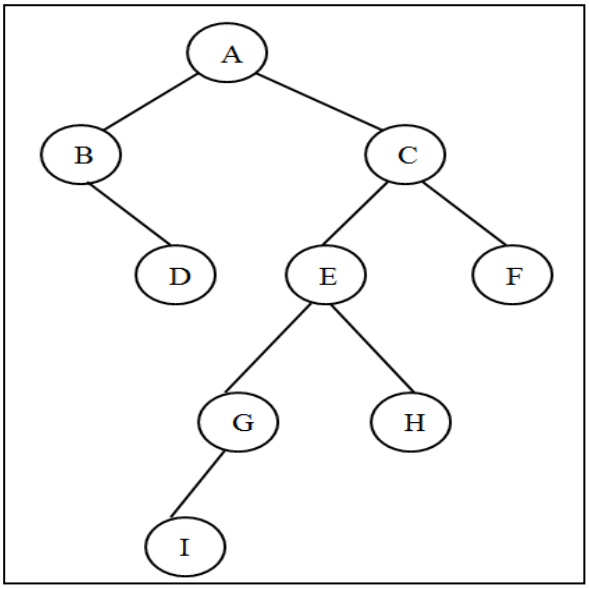

Figure 1: Binary Tree containing 9 nodes

Commonly there are three traversing methods: in-order, pre-order and post-order traversal. In an in-order traversal, first the left child is processed recursively, and then process the current node followed by the right child. The output of an in-order traversal algorithm of the binary tree shown in figure 1 is: B, D, A, I, G, E, H, C, F and that of the pre-order is: A, B, D, C, E, G, I. H, F.

It is well known that given the in-order traverse of a binary tree, along with one of its pre-order or post-order traversals, the original binary tree can be uniquely identified. It is not difficult to write a recursive algorithm to reconstruct the binary tree. Most text books and reference books present the recursive and non-recursive algorithms for traversing a binary tree in in-order, post-order and pre-order. The computation time required is $O(N^2)$ where N is the number of nodes in the tree. Many iterative methods for reconstructing a binary tree has been proposed till date for which computation time required is $O(N)$. In this paper one of the proposed algorithms has been examined. The proposed algorithm works well for most of the input cases, but it has a drawback. There are some sequences of pre-order and in-order traversals for which no legitimate tree can be constructed but the proposed algorithm didn't take this case into consideration and constructed a wrong tree for these cases. We have modified this algorithm so that if for an in-order and pre-order sequence a correct tree

cannot be drawn then it is reported in $O(N)$ best case time and $O(NlogN)$ space.
This paper is organized as follows: Section 2 includes the related reconstruction algorithm; Section 3 introduces our modified algorithm and its complexity analysis, performance and result comparison in Section 4 and conclusion and future aspects with applications of our algorithm are discussed in section 5.

2. BACKGROUND AND RELATED WORK

In computer science, a data structure is a particular way of storing and organizing data in a computer so that it can be used efficiently. Different kinds of data structures are suited to different kinds of applications, and some are highly specialized to specific tasks. For example, B-trees are particularly well-suited for implementation of databases, while compiler implementations usually use hash tables to look up identifiers [2, 16].
Data structures are used in almost every program or software system. Data structures provide a means to manage huge amounts of data efficiently, such as large databases and internet indexing services. Usually, efficient data structures are a key to designing efficient algorithms. Some formal design methods and programming languages emphasize data structures, rather than algorithms, as the key organizing factor in software design [2, 16].

2.1 Related reconstruction algorithm

It is well known that given the in-order traverse of a binary tree, along with one of its pre-order or post-order traversals, the original binary tree can be uniquely identified. Most textbooks and reference books present the traversal algorithm as a recursive algorithm [1, 2, 3, 4, 5, 8, 10]. It is not difficult to write a recursive algorithm to reconstruct the binary tree. The computation time required is $O(N^2)$ where N is the number of nodes in the tree.

2.1.1 Recursive Algorithm

If we know the sequences of nodes obtained through in-order/pre-order/post-order traversal it may not be feasible to reconstruct the binary tree. This is because two different binary trees may yield same sequence of nodes when traversed using post-order traversal. Similarly in-order or pre-order traversal of different binary trees may yield the same sequence of nodes. However, a unique binary tree can be constructed if the result of in-order and pre-order traversal are available. The recursive algorithm for reconstruct a binary tree from its traversals is as follows:

In-order traversal: 4, 7, 2, 8, 5, 1, 6, 9, 3
Pre-order traversal: 1, 2, 4, 7, 5, 8, 3, 6, 9

The first value in the pre-order traversal gives us the root of the binary tree. So the node with data 1 becomes the root of the binary tree. In in-order traversal, initially the left sub-tree is traversed then the root node and then the right sub-tree. So the data before 1 in the in-order list (i.e. 4, 7, 2, 8, 5) forms the left sub-tree and the data after 1 is the in-order list (i.e. 6, 9, 3) forms the right sub-tree. Figure 2(a) shows the structure of the tree after separating the tree in left and right sub-trees.

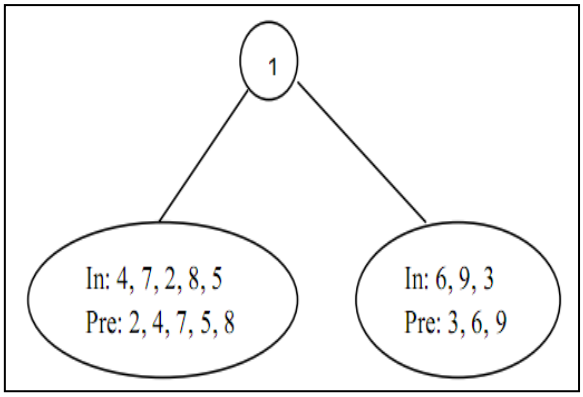

Figure 2(a): Reconstruction of a binary tree.

The next data in the pre-order list is 2 so the root node of the left sub-tree is 2. Hence the data before 2 in the in-order list (i.e. 4, 7) forms the left sub-tree of the node that contains a value 2. The data that comes to the right of 2 in the in-order list (i.e. 8, 5) forms the right sub-tree of the node with value 2. Figure 2(b) shows structure of tree after expanding the left and right sub-tree of the node that contains a value 2.

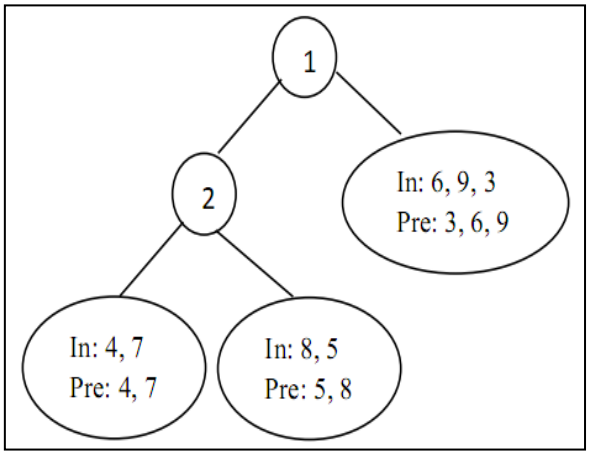

Figure 2(b): Reconstruction of a binary tree (contd.)

Now the next data in pre-order list is 4, so the root node of the left sub-tree of the node that contains a value 2 is 4. The data before 4 in the in-order list forms the left sub-tree of the node that contains a value 4. But as there is no data present before 4 in in-order list, the left sub-tree of the node with value 4 is empty. The data that comes to the right of 4 in the in-order list (i.e. 7) forms the right sub-tree of the node that contains a value 4. Figure 2(c) shows structure of tree after expanding the left and right sub-tree of the node that contains a value 4.

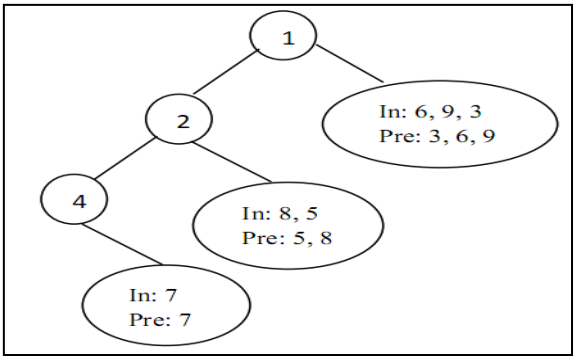

Figure 2(c): Reconstruction of a binary tree (contd.)

We are now left with only one value 7 in both the pre-order and in-order form we simply represent it with a node as shown in figure 2(d).

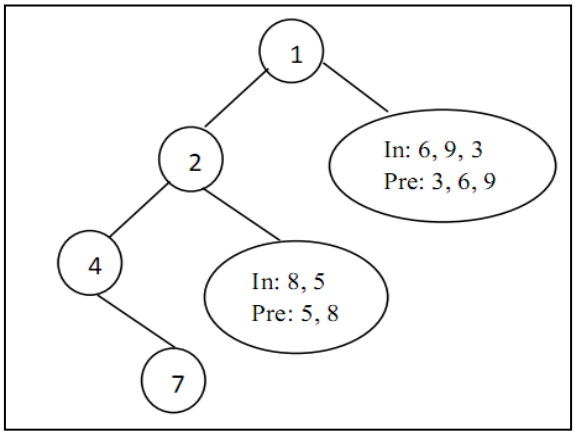

Figure 2(d): Reconstruction of a binary tree (contd.)

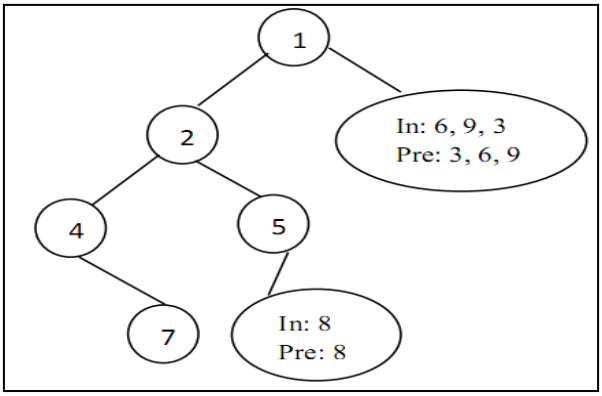

Figure 2(e): Reconstruction of a binary tree (contd.)

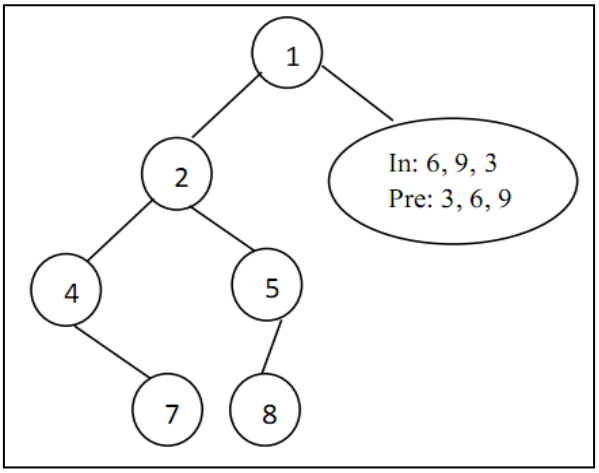

Figure 2(f): Reconstruction of a binary tree (contd.)

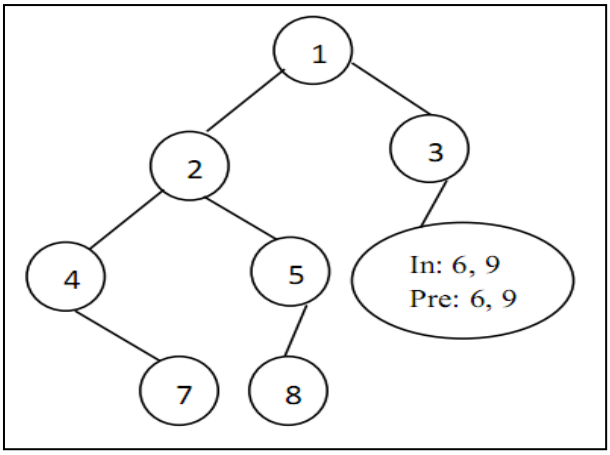

Figure 2(g): Reconstruction of a binary tree (contd.)

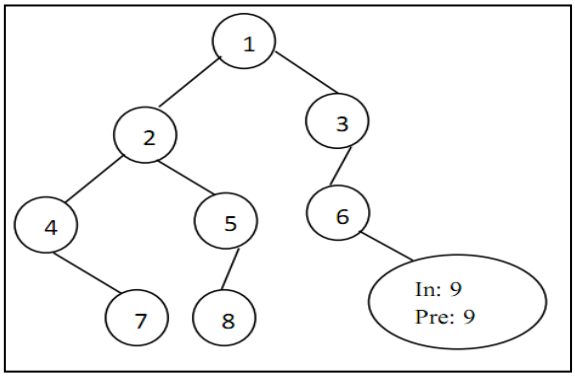

Figure 2(h): Reconstruction of a binary tree (contd.)

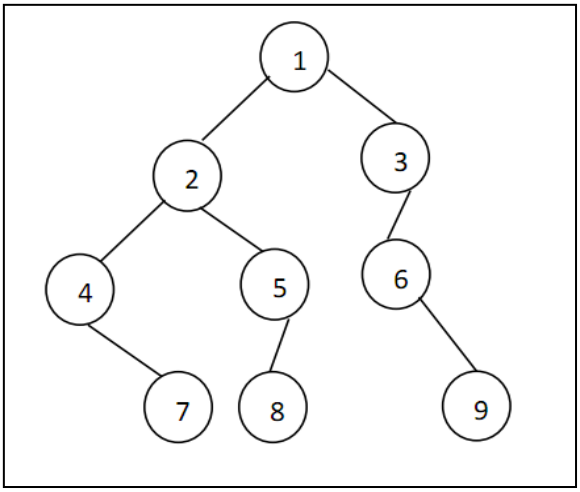

Figure 2(i): Reconstruction of a binary tree (contd.)

In the same way one by one all the data are picked from the pre-order list and are placed and their respective sub-trees are constructed and the whole tree is constructed. Figure 2(e) to 2(i) shows each step of construction of nodes one by one.

2.1.2 Parallel Algorithms

V. Kamakoti and **C. Pandu Rangan** [17] presents an optimal algorithm for reconstructing a binary tree. They presents a parallel EREW PRAM algorithm using $O(n/\log\log n)$ processors to reduce the reconstruction process to parallel merging. Their algorithms runs in $O(\log n)$ time.

Stephan Olariu and **Michael Overstreet** [18] present reconstructing algorithm with the best parallel merging algorithms, their algorithm can be implemented in $O(\log\log n)$ time using O(n/log log n) processors on CREW PRAM or in O(log n) time using O(n/log n) processors in EREW PRAM.

2.1.3 Non-Recursive Algorithms

H. A. Burgdorff [12] presented a non-recursive algorithm for reconstructing a binary tree from its in-order and pre-order sequences (in short i-p sequences) and their algorithm takes $O(N^2)$ computation time.

G. H. Chen [13] has also proposed a non-recursive algorithm for reconstructing a binary tree from its traversals from its i-p sequence array of time complexity $O(N)$ and inefficient space.

In the algorithm proposed by G H Chen [13] the non-recursive algorithm for reconstructing the original binary tree from its in-order and pre-order traversal is done in two stages [13]. First, the i-p sequence is constructed from the in-order and pre-order traversals. Then the original binary tree is reconstructed from the i-p sequence array. Let I[1...n] and P[1…n] be the two sequences that represents the in-order and pre-order traversal respectively of the given binary tree with n nodes. Also, let IP[1…N] represent the corresponding i-p sequence. In fact $IP[i] = I^{-1}[P[i]]$, $1 \le i \le N$.

G H Chen has proposed a non-recursive algorithm by proving following two lemmas [13].

Lemma 1: Let P[i] be a non-leaf node and $s\ (s > 0)$ be the number of its descendants. Then, $IP[i] > IP[j]$ for $i < j < k$ and $IP[i] < IP[j]$ for $k \le j \le i+s$ iff $P[i+1],\ldots P[K-1]$ are the left descendants of P[i] and P[k],…P[i+s] are the right descendants of P[i]. Moreover, P[$i + 1$] is the left child of P[i] and P[k] is the right child of P[i].

According to Lemma 1, the binary tree can be reconstructed by sequentially examining array P. For each non leaf node P[i], if s is known, then left and right child can find (if they exist) as stated by lemma 1. Let S[i], $1 \le i \le n$, denote the cardinality of the set $\{ j \mid j < i \text{ and } IP[j] < IP[i] \}$. S[i] is the number of nodes that precede P[i] in both the in-order and the pre-order sequences. Arrays S will be used in the reconstruction algorithm and can be computed according to the following lemma, with S[1] = 0 initially.

Lemma 2: Let P[j] be the parent of P[i]. If P[i] is the left child of P[j], then S[i] = S[j]. If P[i] is the right child of P[j], then S[i] = IP[j].

Let [L1[i]……..L2[i]] be the range of array P where the left descendants of P[i] are located and [R1[i]….R2[i]] be the range of array P where the right descendants of P[i] are located. If L1[i] > L2[i] (R1[i] > R2[i]), then P[i] has no left (right) descendants. Initially, L1[1]=2, L2[i]=IP[1], and R2[1]=n. Also, R1[1]=L2[1] + 1, if L1[1] ≤ L2[i], R1[1]=2 otherwise. Let P[j] (j<1) be the parent of P[i], L1[i], L2[i], R1[i] and R2[i] are computed as follows:

L1[i] = i+1;
L2[i] = I+IP[i]-S[i]-1;
R1[i] = **if**(L1[i] ≤ L2[i]) **then**
L2[i]+1
else
i + 1
R2 [i] = **if**(P [i] is the right child of P[j]) **then**
R2[j]
else
L2[j]

This algorithm is meant for only binary trees implemented using arrays.

The computations of L1[i], R1[i], and R2[i] are straightforward; the computation of L2[i] is based on the following lemma [13].

Lemma 3: The number of left descendants of P[i] equals IP[i]-S[i]-1.

The nodes that precede P[i] in the in-order sequence fall into two classes. The left descendants of P[i] belong to the first class. The others belong to the second class. Each node in the first class follows P[i] in the pre-order sequence; each node in the second class precedes P[i] in the pre-order sequence. Thus, the number of left descendants of P[i] equals the number of nodes preceding P[i] in the in-order sequence minus the number of nodes preceding P[i] in both the in-order and the pre-order sequences, which is IP[i]-S[i]-1.

In the following, **G. H. Chen** presents the reconstructing algorithm.

L1[1]:=2; L2[1]:=IP[1];
R1[1]:=**if** L1≤L2[1] **then** L2[1] +1 **else** 2;
R2[1]:= N; S[1]:=0;
if L1[1]≤L2[1]
then P[L1[1]] is the left child of P[1];
if R1[1]≤R2[1]
then P[R1[1]] is the right child of P[1];
for *i:= 2 to N* **do**
begin {Assume P[j] is the parent of P[i]
S[i]:= **if** P[i] is the left child of P[j] **then** S[j] **else** IP[j];
L1[i]:= i+1;
L2[i]:= i+IP[i] –S[i] -1;
R1[i]:= if L1[i]≤L2[i] then L2[i] + 1 else i+1;
R2[i]:= if P[i] is the right child of P[j] then R2[j] else L2[j];
if L1[i]≤L2[i] then P[L1[i]] is the left child of P[i];
if L1[i]≤L2[i] then P[L1[i]] is the left child of P[i];
if R1[i]≤R2[i] then P[R1[i]] is the right child of P[i];
end

The computation time and space required for executing the reconstructing algorithm are *O(N)*.

Vinu V Das [11] presented a non-recursive algorithm for reconstructing a binary tree from its in-order and pre-order traversals and their algorithm takes *O(N)* computation time and space complexity is *O(NlogN).*

The proposed non-recursive algorithm by **Vinu V Das** [11] is to reconstruct the binary tree, implemented using linked list, from its in-order and pre-order traversals [11]. Let P[1….N] and I[1….N] be the pre-order and in-order traversal of the given binary tree with N nodes. From this sequence a binary tree will be reconstructed using linked list and root node

address will be returned for any further use. This non-recursive algorithm works on the following lemma [11]:

Lemma: Let P[j] be the parent of P[i], (where j < i) then $L_{p[j]}$ be the corresponding index in the in-order sequence of node P[j] and $L_{p[i]}$ be the corresponding index in the in-order sequence of node P[i]. If $L_{p[j]}$ is greater than $L_{p[i]}$ then P[i] is the left child of P[j] otherwise it is the right child. The binary tree can be reconstructed by exploiting the following three properties of the above lemma [11].

1. First node in the pre-order sequence will be the root of the binary tree.
2. The node(s) which comes to the left of root node in the in-order sequence will be the nodes in the left sub tree and node(s) in the right are the nodes in the right sub tree of the reconstructed root node.
3. Apply the second property recursively to the left and right sub tree of the root node to obtain the complete binary tree.

For example in Figure 3, the first node of the pre-order sequence is A, which is a root node of the reconstructed tree. The nodes B and D are the only two nodes to the left of A in in-order sequence, and they are the nodes in the left sub tree of the root node A.

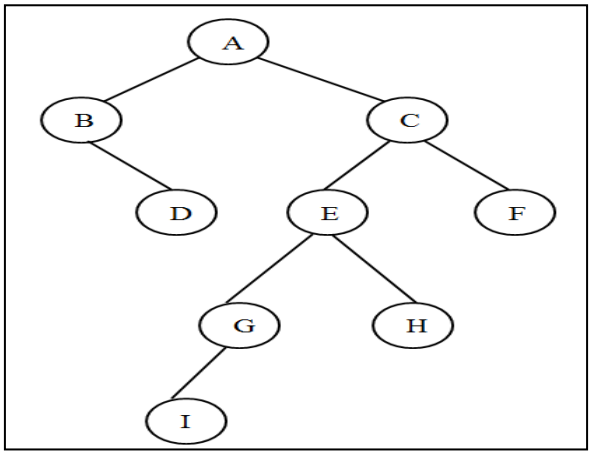

Figure 3: Binary Tree

The new Non-Recursive Algorithm for Reconstructing a Binary Tree from its Traversals presented by Vinu V Das is based on the following [11]:

The left and right are the two variables and any of them will be set to one when a new node needs to be placed to the left or right of the present node. If left = 1 and right = 0 then the new node will be left child of the present node. Let LL be a linear linked list where all visited nodes are stored pre-order and unvisited nodes in in-order sequence. Whenever a node is visited in in-order it will be removed from the LL. PresentNode is a pointer variable that holds the address of the present node.
Following is the reconstruction algorithm [11]:

```
Left = 0; Right = 0; count1=0; count2=0; data=P[0]; root=NULL;

for(count1=0;count1<n;count1++)
{
        Create a NewNode
        NewNode->info = data
        NewNode->lChild = NULL
        NewNode->rChild = NULL
        if(root == NULL)
        root = NewNode
                while (I[count2] is present in the LL)
                {
                        PresentNode= address of I[count2];
                        remove the I[count2] from the LL;
                        right = 1;Left=0;count2++;
                }
        if( data != I[count2] )
        {
                Add the "data" into the linear list LL
                If( Left == 1)
                Place the NewNode as a left child of PresentNode
                else if(Right == 1)
                Place the NewNode as a right child of PresentNode
                Left=1; Right=0;
        }
        else
        {
                If( Left == 1)
                Place the NewNode as a left child of PresentNode
                else if(Right == 1)
                Place the NewNode as a right child of PresentNode
                Left=0; Right=1; count2++;
        }
        data = P[count1];
        PresentNode=NewNode;
}
```

The computation time required for executing the reconstruction algorithm are *O(N)* and space complexity is *O(NlogN)* [11].

3. OUR MODIFIED NON-RECURSIVE ALGORITHM

The algorithm proposed by Vinu V Das [11] is implemented in C language and checked for some different input sequences. The algorithm works correctly for some sequences but on the other side it generated a wrong binary tree for some other input sequences. We made some modification in the algorithm and proposed the solution. The proposed solution is based on the following checks:

1. First construct the tree and if the height of the tree is less than *N*, where *N* is the number of nodes then the tree is always correct.
2. Otherwise, if the height of the tree is equal to n, then there is a chance that the constructed tree may be wrong.
3. In this case where is the height of the tree is equal to the number of nodes, the tree is traversed in an in-order fashion and it is compared with the given in-order sequence.
4. If the sequence matches then the tree formed is correct.
5. Otherwise, the in-order and pre-order sequences cannot combine to form a legitimate tree.

The left and right are the two variables and any of them will be set to one when a new node needs to be placed to the left or right of the present node. If left = 1 and right = 0 then the new node will be left child of the present node. Let LL be a linear linked list where all visited nodes are stored pre-order and

unvisited nodes in in-order sequence. Whenever a node is visited in in-order it will be removed from the LL. PresentNode is a pointer variable that holds the address of the present node.

An extra variable *m* is used in our algorithm which keeps a count of the number of nodes having two children.

Following is the modified Non-Recursive algorithm for generating a tree from its in-order and pre-order traversals:

```
/*Variable Declarations*/
left=0; right=0; count1=0; count2=0; data=P[0]; root=NULL;
m=0; /* New variable in our program which keeps a count of
the number of nodes having two children */
for ( count1 = 0  ; count1<n  ;  count1++  )
{
        /* create a new node*/
        newnode->info = data;
        newnode->lchild = NULL;
        newnode->rchild = NULL;
        If (root == NULL)
                root=newnode;
        /*Intially the linked list is empty*/
        while ( I[count2] is present in the linked list )
        {
                 presentnode = address of I[count2];
                remove the I[count2] from the linked list;
                right=1;left=0;count2++;
                /*Here a small modification is done*/
                m++;
        }
        If( data != I[count2] )
        {
                Add the data into the linked list;
                If (left == 1)
                Place the newnode as left child of
presentnode
                If( right == 1)
                Place the newnode as right child of
presentnode
                left=1;right=0;
        }
        else
        {
                 If( left == 1)
                Place the newnode as a left child of
presentnode
                elseif( right == 1)
                Place the newnode as right child of
presentnode
                left=0;right=1;count2++;
        }
        data=P[count1+1];
        presentnode = newnode;
} /*for loop end*/

If (m is greater than zero or check if the in-order traversal of
the tree formed matches with the given in-order sequence)
{
 Print the tree;
}
else
{
 Print that a legitimate tree cannot be constructed with the
sequences;
}
```

The best case time complexity of our modified algorithm is *O(N)* where *N* is the number of nodes.

3.1 Complexity analysis

By the performance analysis of a program, we mean the amount of computer memory and time needed to run a program.

On analysis of the algorithm we find out the recurrence relation that can be formed is

$$T(n) = T(n-1) + \log n \qquad (3.1)$$

We can solve this recurrence relation by iterative method as follows:

$$T(n) = T(n-1) + \log n$$

On adding the subsequent terms of (3.1), we get

$$T(n) = T(n-1) + \log(n)$$
$$T(n-1) = T(n-2) + \log(n-1)$$
$$T(n-2) = T(n-3) + \log(n-2)$$
… … …
… … …
$$T(2) = T(1) + \log(2)$$

Summing up all these terms we get

$$T(n) = T(1) + \log(n) + \log(n-1) + \log(n-2) \ldots\ldots\ldots \log(2)$$

$$T(n) = T(1) + \log(n) \text{ x } (n-1) \text{ x } (n-2) \text{ x } \ldots \text{ x } (2)$$

$$T(n) = T(1) + \log n! \qquad (3.2)$$

From the Sterling approximation [1, 19]

$$n! = \sqrt{2\pi n}\left(\frac{n}{e}\right)^n \left(1 + \theta\left(\frac{1}{n}\right)\right) \qquad (3.3)$$

Taking *log* at base e on both the sides of (3.3)

$$\log n! \geq n \log n - n \log e \qquad (3.4)$$

From (3.2) and (3.4) we get,

$$T(n) = T(1) + \theta(n \log n)$$

So the time complexity of the algorithm is of order of $\theta(n \log n)$.

For the best case scenario the loop runs for n times only, it does not have to search the link list for a parent address. Hence the time complexity will be n.

$$T(n) = T(n-1) + 1$$
$$T(n-1) = T(n-2) + 1$$
$$T(n-2) = T(n-3) + 1$$
… … …
… … …
$$T(2) = T(1) + 1$$

$$T(n) = T(1) + n$$

Hence $T(n)$ is of the order of $O(n)$.

4. PERFORMANCE ANALYSIS AND RESULTS COMPARISON

The new Non-Recursive Algorithm for Reconstructing a Binary Tree from its Traversals presented by Vinu V Das [11] is implemented in C language and test for some input in-order and pre-order sequences. The algorithm works well for some input cases but it has a drawback. This algorithm generates wrong binary tree for some another input sequences. We have modified this algorithm and generate a new Modified non-recursive reconstructing algorithm. The new modified algorithm is implemented in C language and test for some input in-order and pre-order sequences. Our new modified algorithm works well for all types of input cases.

4.1 Performance comparison

Both the algorithms presented by Vinu V Das [11] and our new modified Non-Recursive Algorithm for Reconstructing a Binary Tree from its Traversals is implemented in C language and test for different input cases.

4.1.1 Sample Input and Output

1. Supplied input sequence with total 9 nodes
Pre-order traversal: 1, 2, 4, 8, 9, 5, 3, 6, 7
In-order traversal: 8, 4, 9, 2, 5, 1, 6, 3, 7
Both the algorithm generated the same output tree shown in the figure 4

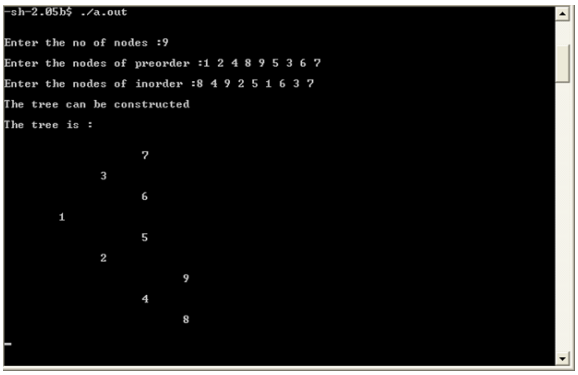

Fig. 4: The generated tree for input sequence Pre-order: 1, 2, 4, 8, 9, 5, 3, 6, 7 and In-order: 8, 4, 9, 2, 5, 1, 6, 3, 7

The tree generated in figure 4 for the input sequence Pre-order traversal: 1, 2, 4, 8, 9, 5, 3, 6, 7 and In-order traversal: 8, 4, 9, 2, 5, 1, 6, 3, 7 is the correct binary tree because we can generate the same pre-order and in-order traversals as supplied.

2. For the input sequence
Pre-order traversal: 1, 2, 4, 8, 9, 5, 3, 6, 7
In-order traversal: 7, 3, 6, 1, 5, 2, 9, 4, 8
The output tree generated by the algorithm presented by Vinu V Das [11] is shown in figure 5 and the output generated by our algorithm is shown in figure 4.3.

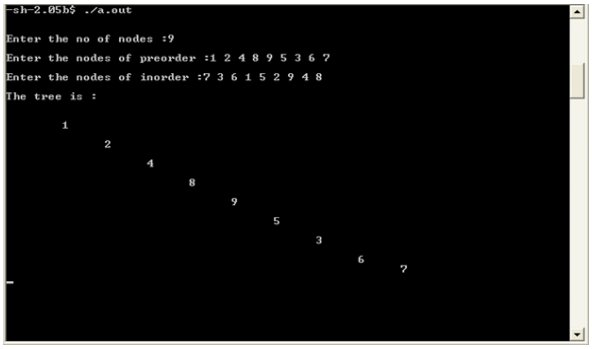

Fig. 5: The generated tree for input sequence Pre-order: 1, 2, 4, 8, 9, 5, 3, 6, 7 and In-order: 7, 3, 6, 1, 5, 2, 9, 4, 8 by the algorithm presented by Vinu V Das

The tree generated in figure 5 for the input sequence Pre-order traversal: 1, 2, 4, 8, 9, 5, 3, 6, 7 and In-order traversal: 7, 3, 6, 1, 5, 2, 9, 4, 8 is an incorrect binary tree because we cannot generate the same pre-order and in-order traversals as supplied.

Fig. 6: The generated output for input sequence Pre-order: 1, 2, 4, 8, 9, 5, 3, 6, 7 and In-order: 7, 3, 6, 1, 5, 2, 9, 4, 8 by our new modified the algorithm

The output message that the tree cannot be constructed in figure 6 for the input sequence Pre-order traversal: 1, 2, 4, 8, 9, 5, 3, 6, 7 and In-order traversal: 7, 3, 6, 1, 5, 2, 9, 4, 8 is generated.

Some other inputs supplied to our modified algorithm.

3. For the input

Pre-order Traversal: 1, 2, 3, 4, 5, 6
In-order Traversal: 1, 2, 3, 4, 5, 6
The correct generated binary tree is shown in figure 7.

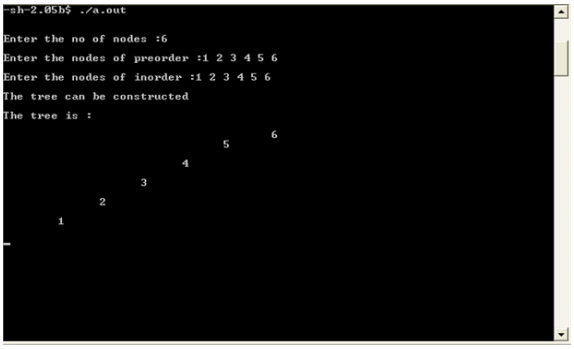

Fig. 7: The generated output for input sequence Pre-order: 1, 2, 3, 4, 5, 6 and In-order: 1, 2, 3, 4, 5, 6 by our new modified the algorithm

4. For the input sequence

Pre-order Traversal: 1, 2, 3, 4, 5, 6, 7
In-order Traversal: 7, 6, 5, 4, 3, 2, 1

The correct generated binary tree is shown in figure 8.

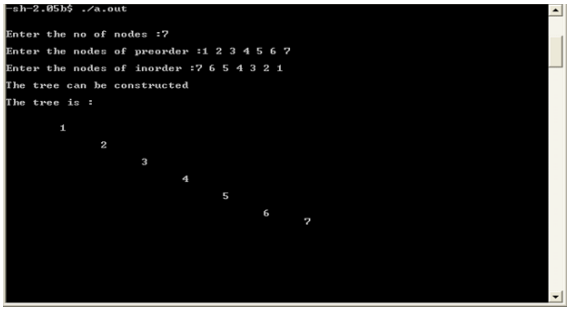

Fig. 8: The generated output for input sequence Pre-order: 1, 2, 3, 4, 5, 6, 7 and In-order: 7, 6, 5, 4, 3, 2, 1 by our new modified the algorithm

5. CONCLUSION AND FUTURE WORK

A tree is a fundamental structure in computer science. Almost all operating systems store files in trees or tree like structures. It is well known that given the in-order traverse of a binary tree, along with one of its pre-order or post-order traversals, the original binary tree can be uniquely identified. It is not difficult to write a recursive algorithm to reconstruct the binary tree. Most text books and reference books present the recursive and non-recursive algorithms for traversing a binary tree in in-order, post-order and pre-order. The computation time required is $O(N^2)$ where N is the number of nodes in the tree. Many iterative methods for reconstructing a binary tree has been proposed till date for which computation time required is $O(N)$. The Non- Recursive algorithm presented by Vinu V Das [11] is considered in this paper. This algorithm works well for most of the input in-order and pre-order traversals, but generates wrong binary tree for some another in-order and pre-order sequences. In this paper a new modified Non Recursive algorithm has been proposed which works well for all the input cases.

Also in the future better algorithm can be found out than the above one which would further reduce the complexity of the algorithm. The algorithm can be implemented using suitable simulation tools.

5.1 Applications

The tree is fundamental structure in computer science. The applications of new proposed algorithm in general are:

1. The optimality with the simplicity of the algorithm makes this method an alternative to other methods for temporary storage of a tree in secondary memory. Almost all operating systems store files in trees or tree like structures.
2. It may for instance be used on tree structures that are expensive to construct, such as optimum search trees and multidimensional trees.
3. Trees are also used in compiler design, text processing and searching algorithms.
4. In Huffman Coding instead of sending the whole tree in transmission, redundant sequences of in-order and pre-order are send from the sender to the receiver. If a wrong tree is accidently formed then the whole process of compression may go wrong. The modified algorithm prevents any wrong tree from forming.

6. REFERENCES

[1] Vinu V Das, "*Principles of Data Structures Using C and C++*", New Age International Publishers, Reading, Mass., 2005.

[2] M. Weiss, "*Data Structures & Problem Solving Using Java"*, 2nd ed., Addison Wesley, 2002.

[3] D. E. Knuth, "*The Art of Computer Programming"*, Vol. 3 (2nd ed.): Sorting and Searching, Addison Wesley, 1998.

[4] R. Sedgewick, "*Algorithms in Java"*, 3d edition, Addison Wesley, 2003.

[5] D. E. Kunth, "*The Art of Computer Programming*", Vol. 1: Fundamental Algorithm, Addison-Wesley, Reading, Mass., 1973.

[6] Mark Allen Weiss, "*Data Structures and Algorithm Analysis in C*", Vol. 3 (2nd ed.), Addison-Wesley, 1997.

[7] Yedidyah Langsam, Moshe J. Augenstein, Aaron M. Tanenbaum: "*Data Structures using C and C++*", 2nd Ed., Prentice-Hall India, July 2002.

[8] Sartaj Sahni: "*Data Structures, Algorithms and Applications in JAVA*", 2nd Ed., University Press.

[9] A. Anderson and S. Carlsson: "*Construction of a tree from its traversals in optimal time and space"*, Information Processing Letters, 34:21-25, 1990.

[10] N. Gabrani and P. Shankar: "*A note on the reconstruction of a binary tree from its traversals*", Information Processing Letters, 42:117-119, 1992.

[11] Vinu V Das, "*A new Non-Recursive Algorithm for Reconstructing a Binary Tree from its Traversals*", IEEE Comm., pp. 261-263, 2010.

[12] H. A. Burgdorff, S. Jojodia, F.N. Springsteel, and Y.Zalcstein, "*Alternative Methods for the Reconstruction of Tree from their Traversals*", BIT, Vol. 27, No. 2, p. 134, 1987.

[13] G. H. Chen, M. S. Yu, and L.T. Liu: "*Non-recursive Algorithms for Reconstructing a Binary Tree from its Traversals*", IEEE Comm., Vol. 88, pp. 490-492, 1988.

[14] C. C. Lee, D. T. Lee, C. K. Wong: "*Generating Binary Trees of bounded height*", Acta Inf., 23, 529-544, 1986.

[15] Seymour Lipschutz: "*Theory and problem of Data Structures*", International Edition, McGRAW-HILL, 1986.

[16] Glenn W. Rowe: "*Introduction to Data Structure and Algorithms with C++*", PHI, ISBN: 81-203-1277-5.

[17] Robert Sedgewick: "*Algorithms in C++*", Ed. 3rd, 2001, ISBN: 81-7808-249-7.

[18] C. P. Kruskal: "*Searching, merging, and sorting in parallel computation*", IEEE Transactions on Computers, 32:924-946, 1983.

[19] Lindstrom, G. Scanning: "*List structures without stacks or tag bits*". Inform. Proc. Letters 2, 1973, pp. 47-51.

[20] Arora N., Tamta V. and Kumar S: "*Modified Non-Recursive Algorithm for Reconstructing a Binary Tree from its Traversals*", IJCA, volume 43-No.10, April 2012.