# TRACING BACK A BINARY TREE

**Dr. Mrs. Meenakshi M. Sagdeo**[1]

***ABSTRACT: Binary tree is a nonlinear data structure, mainly used to represent data containing a hierarchical relationship between elements. There are three standard ways of traversing a binary tree, namely, preorder, inorder and postorder. The problem usually faced is to find the original tree structure given either (a) Preorder and Inorder traversals or (b) Postorder and Inorder traversals.***

***Some authors (e.g. Lipschutz) have given a method to trace back the original tree. However, it is not suitable for solving the problem manually, especially when the total number of nodes is large.***

***Hence, in this paper, stack oriented algorithms are presented for solving the above problem and obtaining link representation of a binary tree.***

***The corresponding programs in 'C' language with output are also given.***

***Keywords: Binary tree, Inorder, Postorder and Preorder Traversals.***

## 1. INTRODUCTION

Binary tree is a nonlinear data structure, mainly used to represent data containing a hierarchical relationship between elements. There are three standard ways of traversing a binary tree namely, **preorder**, **inorder** and **postorder**. It should be noted that the left sub-tree of the root is always traversed before the right sub-tree and the root of the given tree is the first node appearing in preorder traversal and the last node appearing in postorder traversal. For convenience, we shall use positive integers as node identifiers although generally nodes are represented by alphabets.

Consider the tree in Fig.1. It has 11 nodes and its traversals are as follows:

PREORDER: 1 2 4 5 6 3 7 8 9 11 10
INORDER: 4 2 6 5 1 7 3 11 9 8 10
POSTORDER: 4 6 5 2 7 11 9 10 8 3 1

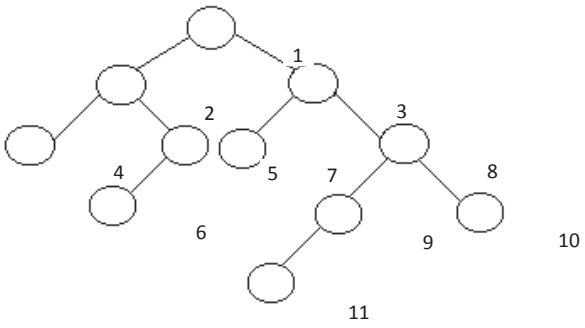


***Fig. 1 BINARY TREE***

The problem usually faced is to find the original tree structure given either

a) Preorder and inorder traversals or

b) Postorder and inorder traversals.

Some authors (e.g. Lipschutz) have given a method to trace back the original tree. However, it is not suitable for solving the problem manually, especially when the total number of nodes is large.

Hence, in this paper, we present *stack oriented* algorithms for solving the above problem and obtaining *link representation* of a binary tree if its inorder traversal is given along with either its preorder or postorder traversal.

## 2. ALGORITHMS

We now give two algorithms to trace back a binary tree. Algorithm 1 works when inorder traversal is given along with preorder traversal. Algorithm 2 works when inorder traversal is given along with postorder traversal. Both the algorithms are independent of the contents of the different nodes. Assume that the binary tree has *n* nodes and the sequences of nodes in inorder, preorder and postorder traversals are stored in arrays *in*, *pre* and *post* respectively. Both the algorithms use a special array named *posn* to store the positions of different nodes in the inorder traversal.

The definition of *posn* of a node and the way of constructing the array *posn* is as follows.

***Definition:***

Position of a particular node in array *pre* or *pos*t is defined as the serial number of the location in which it appears in inorder traversal i.e. in array *in*.

A numeric array *posn* is used to store these positions. Its elements present a **unique permutation** of n numbers which serves as a **link** between preorder or postorder traversals and inorder traversal.

***Example:***

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Sr. No.( *i* ): | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
| ( *in* ): | 4 | 2 | 6 | 5 | 1 | 7 | 3 | 11 | 9 | 8 | 10 |
| ( *pre* ): | 1 | 2 | 4 | 5 | 6 | 3 | 7 | 8 | 9 | 11 | 10 |
| ( *posn* ): | 5 | 2 | 1 | 4 | 3 | 7 | 6 | 10 | 9 | 8 | 11 |

(Here, *posn* gives positions where the nodes in *pre* appear in *in*.)

Arrays *l* and *r* are used to store the left and right successors of a node respectively. An array, *par* is used to store the parent-nodes of different nodes appearing in pre (or post) traversal. $p = 0$ is used as a sentinel to indicate that the root has been reached; where *p* gives the parent-node number. As will be clear from the program attached herewith, whenever some node is put as a left or right successor, its parent-node is thereby decided and accordingly the arrays *l*, *r* and *par* are filled.

### *2.1 Case 1:*

Given - Inorder and Preorder traversals.

**Algorithm 1:**

1. Input the total number of nodes in the binary tree (*n*).

2. Input the nodes in preorder and inorder traversals. i.e. input the elements of arrays *pre* and *in*.

3. Find the positions of elements of array *pre* in array *in*. (i.e. generate a new array *posn*)

4. Print that the root of the tree is *pre[1]*.

5. Consider the first node in the preorder traversal. i.e. Set *i = 1*.

6. If *posn[1]=1*, then the root has no left branch.
   Hence, set *l[i]=0*
   *r[i]=pre[i+1]*
   *par[i+1]=i*
   and go to step 9.

7. If *posn[1]=n* then the root has no right branch.

8. Main Logic:

   Compare the position of the node under consideration i.e. the 'current-node' (*posn[i]*) with the position of the 'next-node' (*posn[i+1]*), in an array *pre*.

   If the 'current-node' has its position beyond the position of the 'next-node', then it means that the 'next-node' is the left-successor of the 'current-node' and then go to step 9, else, the 'next-node' is the right-successor of some node on the branch containing the 'current-node'. The exact position of the 'next-node' in the tree is decided by putting on stack all the nodes having their positions before the position of 'next-node' on stack. For this we start with the 'current-node', then consider its parent-node and so on. In other words, we traverse backward from the 'current-node' on the branch to the root. Stop stacking the nodes when either position of the node under consideration is found to be beyond the position of the 'next-node' or the root has been reached. Then pop up one by one the nodes from the stack. Stop as soon as we

ISBN - 978-93-81583-55-5 

get the first node with no right-successor. Put the 'next-node' as the right-successor of this node obtained from the stack. An array *st* is used for this purpose.

9. Proceed further i.e. go to step 8 by considering the 'next-node' in *pre* i.e. increment *i*, while $i < n$.

10. When the $(n-1)^{th}$ node is compared with the $n^{th}$ node in an array *pre* and its position is decided in the tree, print the entire linked representation of tree.

***2.2 Case 2:***

Given - Inorder and Postorder traversals.

***Algorithm 2:***

1. Input the total number of nodes in the tree (*n*).

2. Input the nodes in postorder and inorder traversals. i.e. input the elements of arrays *post* and *in*.

3. Find the positions of elements of array *post* in array *in*. (i.e. generate a new array *posn*)

4. Print that the root of the tree is *post[n]*.

5. Consider the last node in the postorder traversal. i.e. Set $i = n$.

6. If *posn[n]=n*, then the root has no right branch.

   Hence, set r*[i]=0*
   *l[i]=post[i-1]*
   *par[i-1]= i*
   and go to step 9.

7. If *posn[n]= 1* then the root has no left branch.

8. Main Logic:

   Compare the position of the node under consideration i.e. the 'current-node' (*posn[i]*) with the position of the 'preceding-node' (*posn[i-1]*), in an array *post*.

   If the 'current-node' has its position before the position of the 'preceding-node', then it means that the 'preceding-node' is the right-successor of the 'current-node' and then go to step 9, else, the 'preceding-node' is the left-successor of some node on the branch containing the 'current-node'. The exact position of the 'preceding-node' in the tree is decided by putting on stack all the nodes having their positions beyond the position of the 'preceding-node'

on stack. For this we start with the 'current-node', then consider its parent-node and so on. That is, we traverse backward from the 'current-node' on the branch to the root. Stop stacking the nodes when either position of the node under consideration is found to be before the position of the 'preceding-node' or the root has been reached. An array *st* is used for this purpose.

Then pop up one by one the nodes from the stack. Stop as soon as we get the first node with no left-successor. Put the 'preceding-node' as the left-successor of this node obtained from the stack.

9. Proceed further i.e. go to step 8 by considering the 'preceding-node' in *post* i.e. decrement *i* and proceed to step 8, while $i > 1$.

10. When the second node is compared with the first node in an array *post* and its position is decided in the tree, print the entire linked representation of tree.

***2.3 Case 3:***

If only the postorder and preorder traversals are given, while the inorder traversal is not known then we can not uniquely determine the tree.

Consider the binary trees having their inorder traversals as follows:

(a) *in* : 1 2 3 4 5

(b) *in* : 2 3 4 5 1

We observe that their preorder and postorder traversals are as follows:

*pre* : 1 3 2 4 5

*post*: 2 5 4 3 1

Thus both trees, although distinct, have the same preorder and post order traversals.

***Remark:***

The programs for the above algorithms in 'C' language along with the outputs are given in the appendix. The 'time complexity' of both the programs is o ($n^2$). However, the 'storage space' required is increased, as a special array *posn* has to be maintained.

## 3. CONCLUSION

The algorithms proposed in this paper give us an easy method to solve the problem of tracing back a binary tree when its in-order traversal is given along with any one of its pre-order or post-order traversals.

ISBN - 978-93-81583-55-5

## *APPENDIX*

### *PROGRAM-1 FOR ALGORITHM 1*

### *(Given: PREORDER AND INORDER TRAVERSALS)*

```
#include<stdio.h>
#include<math.h>
main()
{
 /* Application of Algorithm 1 */
Int posn[101]={0},par[101],st[101],pre[101],in[101];
Int l[101]={0},r[101]={0};
int n,i,j,p,top;
clrscr();
printf("\n Enter total number of nodes in the binary tree");
scanf("%d",&n);
printf("\n Input node numbers in preorder and inorder traversals:\n");
for(i=1;i<=n;i++)
  scanf("%d %d",&pre[i],&in[i]);
printf("\n Sequence of nodes in Preorder Traversal:   ");
for(i=1;i<=n;i++)
  {printf("%d ",pre[i]);}
printf("\n Sequence of nodes in Inorder Traversal:   ");
for(i=1;i<=n;i++)
  {printf("%d ",in[i]);}

/* Finding positions of nodes in Preorder Traversal as they appear in
Inorder  Traversal ");*/
for(i=1;i<=n;i++)
  for(j=1;j<=n;j++)
                                        if(pre[i]==in[j])
                                          posn[i]=j;
printf("\n \n Positions are as follows:");
printf("\n");
for(i=1;i<=n;i++)
  {printf("%d ", posn[i]);}
printf("\n Root of the tree is %d ",pre[1]);
i=1;
par[i]=0;
if(posn[i]==1)
  { printf("\n Root has no left branch");
    l[i]=0;
    r[i]=pre[i+1];
    par[i+1]=i;
    i=i+1;
  }
else
  { if (posn[1]==n)
```

```
      {  printf("\n Root has no right branch");
        r[i]=0;
                                                }
              }
while(i<n)
{ if(posn[i]>posn[i+1])
              { l[i]=pre[i+1];
    par[i+1]=i;
                                                 i=i+1;
                                               }
 else
                                               { top=1;
                                                 st[top]=i;
                                                 p=par[i];
    while((p!=0) && (posn[p]<posn[i+1]))
                                                                                { top=top+1;
    st[top]=p;
    p=par[p];
   }
  while (r[st[top]]!=0)
   { top=top-1;}
    r[st[top]]=pre[i+1];
    par[i+1]=st[top];
    i=i+1;
   }
}
printf("\n node      l        r");
for(i=1;i<=n;i++)
  for(j=1;j<=n;j++)
  if(i==pre[j])
  printf("\n %d    %d    %d    ",i,l[j],r[j]);
  getch();
}
```

**OUTPUT OF PROGRAM-1 FOR ALGORITHM 1**

```
Enter total number of nodes in the binary tree
Input node numbers in preorder and inorder traversals:
Sequence of nodes in Preorder Traversal:1 2 4 5 6 3 7 8 9 11 10
Sequence of nodes in Inorder Traversal:   4 2 6 5 1 7 3 11 9 8 10
Positions are as follows:
5 2 1 4 3 7 6 10 9 8 11
Root of the tree is 1
         node    l     r
 1      2     3
 2      4     5
 3      7     8
```

| | | |
|---|---|---|
| 4 | 0 | 0 |
| 5 | 6 | 0 |
| 6 | 0 | 0 |
| 7 | 0 | 0 |
| 8 | 9 | 10 |
| 9 | 11 | 0 |
| 10 | 0 | 0 |
| 11 | 0 | 0 |

***PROGRAM-2 FOR ALGORITHM 2***
***(Given: POSTORDER AND INORDER TRAVERSALS)***

```
#include<stdio.h>
#include<math.h>
main()
{
/* Application of Algorithm 2  */
Int posn[101]={0},par[101],st[101],post[101],in[101];
Int l[101]={0},r[101]={0};
int n,i,j,p,top;
clrscr();
printf("\n Enter total number of nodes in the binary tree");
scanf("%d",&n);
printf("\n Input node numbers in postorder and inorder traversals:\n");
for(i=1;i<=n;i++)
scanf("%d %d",&post[i],&in[i]);
printf("\n Sequence of nodes in Postorder Traversal:   ");
for(i=1;i<=n;i++)
      {printf("%d ",post[i]);}
       printf("\n Sequence of nodes in Inorder Traversal:   ");
       for(i=1;i<=n;i++)
{printf("%d ",in[i]);

/* Finding positions of nodes in Postorder Traversal as they appear in
Inorder Traversal " */
for(i=1;i<=n;i++)
  for(j=1;j<=n;j++)
if(post[i]==in[j])
posn[i]=j;
printf("\n \n Positions are as follows:");
printf("\n");
for(i=1;i<=n;i++)
  {printf("%d ", posn[i]);}
printf("\n Root of the tree is %d ",post[n]);
i=n;
par[i]=0;
```

```
if(posn[n]==n)
  { printf("\n Root has no right branch");
    r[i]=0;
    l[i]=post[i-1];
    par[i-1]=i;
    i=i-1;
  }
else
  {if (posn[n]==1)
  {printf("\n Root has no left branch");
   l[i]=0;
  }
  }
while(i>1)
{
if(posn[i]<posn[i-1])
  {r[i]=post[i-1];
   par[i-1]=i;
   i=i-1;
  }
else
  { top=1;
    st[top]=i;
    p=par[i];
    while((p!=0) && (posn[p]>posn[i-1]))
                                          { top=top+1;
                                            st[top]=p;
                                            p=par[p];
                                          }
    while (l[st[top]]!=0)
                                          { top=top-1;}
                                            l[st[top]]=post[i-1];
                                            par[i-1]=st[top];
                                            i=i-1;
                                          }
}
printf("\n node     l       r");
for(i=1;i<=n;i++)
  for(j=1;j<=n;j++)
                                            if(i==post[j])
  printf("\n %d       %d       %d    ",i,l[j],r[j]);
  getch();
}
```

***OUTPUT OF PROGRAM-2 FOR ALGORITHM 2***

Enter total number of nodes in the binary tree
Input node numbers in postorder and inorder traversals:
Sequence of nodes in Postorder Traversal: 4 6 5 2 7 11 9 10 8 3 1
Sequence of nodes in Inorder Traversal: 4 2 6 5 1 7 3 11 9 8 10
Positions are as follows:
1 3 4 2 6 8 9 11 10 7 5
Root of the tree is 1

| node | l | r |
|---|---|---|
| 1 | 2 | 3 |
| 2 | 4 | 5 |
| 3 | 7 | 8 |
| 4 | 0 | 0 |
| 5 | 6 | 0 |
| 6 | 0 | 0 |
| 7 | 0 | 0 |
| 8 | 9 | 10 |
| 9 | 11 | 0 |
| 10 | 0 | 0 |
| 11 | 0 | 0 |

## 4. REFERENCE

1. Seymour Lipschutz, Data Structures (Schuam's Outline Series), McGraw Hill Publication

**********

[1]*Dr. Meenakshi M Sagdeo*
*Associate Professor of Statistics*
*Ismail Yusuf College*
*Jogeshwari (East)*
*Mumbai-4000060*
*Maharashtra State, India*
*msagdeo@hotmail.com*