PRE-IN ORDER BINARY TREE

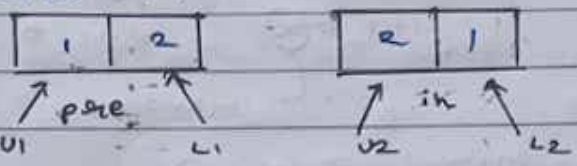
AIM :

To construct a unique binary tree using the given pre-order and in-order traversals.

ALGORITHM :

this is a recursive algorithm.

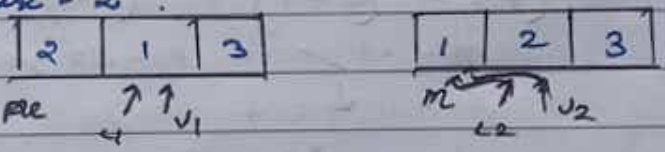
Base case - 1 :



when lower limit crosses the upper limit

this is an invalid case, so the value NULL should be returned.

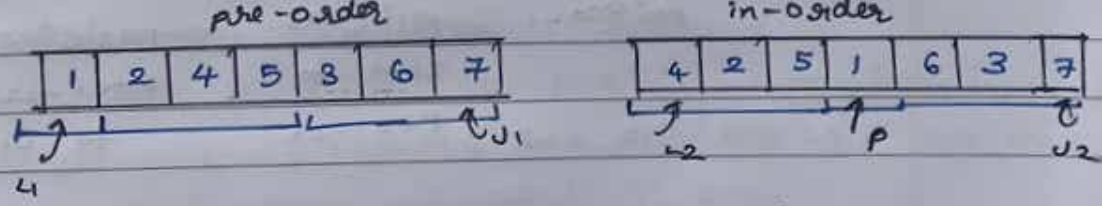
Base case - 2 :



when both limits equalise

this is a leaf node. so once root node is created, it can be directly returned without modifying its left or right child pointers.

Recursive cases :



* the root is present at the 4th position in the pre-order traversal.

* 'P' denotes the location of the root in the in-order traversal.

★ In-order traversal is depicted as

Left → positions L2 to P-1
Root → position P
Right → positions P+1 to U2

★ Pre-order traversal is depicted as

Root → position L1
Left → positions L1+1 to x
Right → positions x+1 to U1

x → unknown

★ calculation of x:

For the left subtree, no. of elements in pre-order traversal = in-order traversal

up - low + 1 = up - low + 1

$(x) - (L1+1) = (P-1) - (L2)$

$x - L1 - 1 = P - L2 - 1$

$x = P + L1 - L2$

so finally

★ Pre-order traversal:

Root → position L1
Left → positions L1+1 to P+L1-L2
Right → positions P+L1-L2+1 to U1

★ First P is calculated.
Then root → left is formed by recursion.
Then root → right is formed by recursion.
Then the formed root is returned.

★ Time complexity: worst case skewed tree, O(n2)
(n of elements)

CODE :

```
#include "bits/stdc++.h"
using namespace std;

// structure of a Binary Tree Node
struct Node
{
    int data;  // data
    Node * left, * right; // left & right children
    Node (int n)  // constructor
    {
        data = n;  // set data
        left = right = NULL; // leaf by default
    }
};

// search for an element in an array
int search ( int * A, int n, int l, int u)
{
    for (int i= l; i <= u; i++)
        if ( A[i] == n)  return i; // element found
    return -1;  // element not found
}

// construction Function
struct Node * form (int pre[], int in[],
                    int l1, int u1, int l2, int u2)
{
    if (l1 > u1)  return NULL;  // base-case 1

    struct Node * root = new Node (pre[l1]);
    if (l1 == u1)  return root;  // base-case 2
```

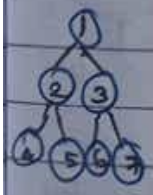
```
    int p = search( in, pre[l1], l2, u2); // locate
                        root in in-order traversal
    root -> left = form ( pre, in,
                l1+1, p+l1-l2, l2, p-1); // left
    root -> right = form ( pre, in,
                p+l1-l2+1, u1, p+1, u2); // right

    return root;
}

// Print a binary tree with NULL children also
void print ( struct Node * root)
{
    if (!root) cout << "N ";
    else
    {
        cout << root -> data << " "; // root
        print ( root -> left); // left
        print ( root -> right); // right
    }
}

int main()
{
    cout << "welcome to c++ Pre-In order Binary
        Tree Generator !" << endl << endl;

    int n; // number of elements
    cout << "Enter the number of elements in your
                                    tree : ";
    cin >> n;

    int pre[n]; // pre-order array
    cout << "Enter the pre-order traversal : ";
    for (int i=0; i<n; i++) cin >> pre[i];
```

```
int in[n];  // in-order array
cout << "Enter the in-order traversal : ";
for (int i=0; i<n; i++) cin >> in[i];

struct Node * root = form(pre, in,
                    0, n-1, 0, n-1); // whole array limits
cout << endl << " Binary Tree was formed
    successfully ! The tree is " << endl;
print(root); // print binary tree

cout << endl << endl << " Thank you for
    using C++ Pre-In Binary Tree
    Generator. Bye Bye ! ";
}
```

INPUT AND OUTPUT :

```
Welcome to C++ Pre-In order Binary Tree Generator!

Enter the number of elements in your tree : 7
Enter the pre-order traversal : 1 2 4 5 3 6 7
Enter the in-order traversal : 4 2 5 1 6 3 7

Binary Tree was formed successfully ! The tree is :
1 2 4 N N 5 N N 3 6 N N 7 N N

Thank you for using C++ Pre-In Binary Tree Generator
Bye Bye !
```

Sample Tree



RESULT :

Binary tree with given pre & in order traversals was successfully generated.

PRE-POST ORDER BINARY TREE

AIM :

To construct a binary tree using the given pre-order and post-order traversals.

ALGORITHM :

This is a recursive algorithm.

Base case-1 : $l1 > u1$

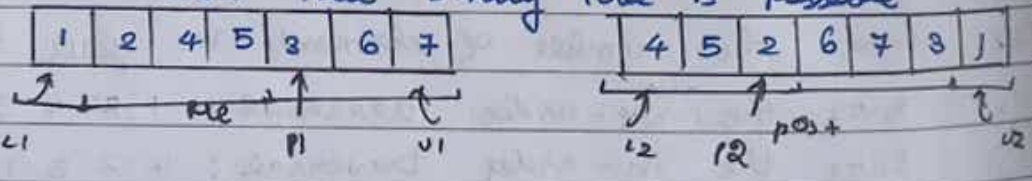
Invalid case → return NULL

Base case-2 : $l1 = u1$

Leaf node → form root and return it without modifying its children.

Recursive cases :

case 1 : when Full Binary Tree is possible



* The root of the tree is present @ the $l1^{th}$ position in the pre-order traversal and the $u2^{th}$ position in the post-order traversal.

* The immediate left child is present @ the $(l1+1)^{th}$ position in the pre-order traversal. This is then located in the post-order traversal and marked as p2.

* The immediate right child is present @ the $(u2-1)^{th}$ position in the post-order traversal

This is then located in the pre-order traversal and marked as P1.

* Pre-order traversal is depicted as

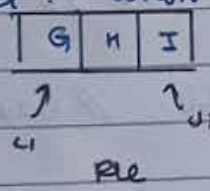
Root → position L1
Left → positions L1+1 to P1-1
Right → positions P1 to U1

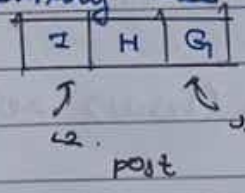
* Post-order traversal is depicted as

Left → positions L2 to P2
Right → positions P2+1 to U2-1
Root → position U2

* First P1 and P2 are calculated.
Then root → left is formed by recursion.
Then root → right is formed by recursion.
Then the formed tree is returned.

Case 2 : when Full Binary Tree is not possible





* Elements @ $(L1+1)^{th}$ and $(U2-1)^{th}$ positions are equal. This means both can't be right & left children, as there is only one node. So, we have to pick up either left or right.

* considering left [and thereby, right is NULL]

Pre : Root → L1, Left → L1+1 to U1

Post : Left → L2 to U2-1, Root → U2

classification into cases

check if pre[l1+1] and post[u2-1] are same.

If yes → only modify left (case 2)
Otherwise → modify left & right (case 3)
then return root of the formed tree.

Time complexity :: O(n) in worst case (skewed tree)

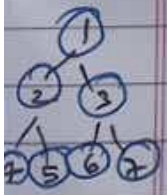
CODE :

```
#include "bits/stdc++.h"
using namespace std;

struct Node { /* same as previous program */ }
int search (char A, char n, int c, int u)
 { /* same as previous program */ }

struct Node * form (char pre[], char post[],
                    int l1, int u1, int l2, int u2)
{
    if (l1>u1) return NULL; // base case-1

    struct Node * root = new Node (pre[l1]);
    if (l1==u1) return root; // base case-2

    if (pre[l1+1] == post[u2-1]) // recursive case 2
        root → left = form (pre, post,
                            l1+1, u1, l2, u2-1);

    else // recursive case-1
    { int p1 = search (pre, post[u2-1], l1, u1);
      int p2 = search (post, pre[l1+1], l2, u2);
```

```
    root->left = form(pre, post,
                 d1+1, p1-1, d2+0, p2+0); // left
    root->right = form(pre, post,
                 p1, u1, p2+1, u2-1); // right
  }

  return root;
}

void print(struct Node * root)
{ /* same as previous program */ }

int main()
{ /* same with slight modifications:
     replacing 'in' with 'post' */ }
```

INPUT AND OUTPUT :

Sample tree




```
Welcome to C++ Pre-Post order Binary Tree Generator!

Enter the number of elements in your tree : 7
Enter the pre-order traversal : 1 2 4 5 3 6 7
Enter the post-order traversal : 4 5 2 6 7 3 1

Binary Tree was formed successfully. The tree is:
1 2 4 N N 5 N N 3 6 N N 7 N N

Thank you for using C++ Pre-Post Binary Tree Generator. Bye Bye!
```

RESULT!

Binary tree with given pre & in order traversals was successfully generated.