We have three arrays of Inorder, Postorder and Preorder which consist the color of node i.e ‘R’ or ‘B’ . We know that in Preorder we first get root node, so in preorder array first position is root node. Simalarly in postorder root node is last, so in postorder array last position is root node.

But in inorder we cannot tell which position is root node,so the task is to find root node in inorder array .

Example

Input: Inorder: B R R B B B R R B R

Preorder: B R B R B R B B R R

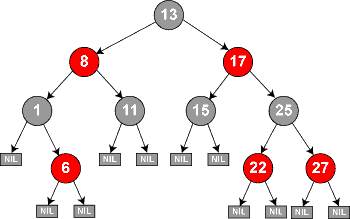
Postorder: R B B R B R R B R B

Output: Root Node is Present on 5th Position in Inorder Traversal.

In inorder traversal we first traverse the left sub-tree first then root and then right sub-tree, so we can use this concept to our advantage.

First loop in postorder and preorder array until you get equal no. of black node in both arrays and similarly red node. Suppose we get i (index) which sastify the condition .Then we traverse the inorder array upto i and count no. of black and no. of red , if these no. matches with the previous count from post order and preorder then we see that if i+1 is black because that is the root node else we continue. Also to addition we compare right child of original tree in post and pre order i.e i+2 element in preorder and size-2 element in postorder. If all the conditions satisfy we can tell that i+1 is the root node.

Example 1:



Step 1:

Inorder:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| B | R | R | B | B | B | R | R | B | R |

Preorder:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| B | R | B | R | B | R | B | B | R | R |

Postorder:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| R  i | B | B | R | B | R | R | B | R | B |

Step 2:

Inorder:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| B | R | R | B | B | B | R | R | B | R |

Preorder:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| B | R | B | R | B | R | B | B | R | R |

Postorder:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| R | B  i | B | R | B | R | R | B | R | B |

Step 3:

Inorder:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| B | R | R | B | B | B | R | R | B | R |

Preorder:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| B | R | B | R | B | R | B | B | R | R |

Postorder:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| R | B | B  i | R | B | R | R | B | R | B |

Step 4:

Inorder:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| B | R | R | B | B | B | R | R | B | R |

Preorder:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| B | R | B | R | B | R | B | B | R | R |

Postorder:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| R | B | B | R  i | B | R | R | B | R | B |

Step 5:

Inorder:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| B | R | R | B | B | B | R | R | B | R |

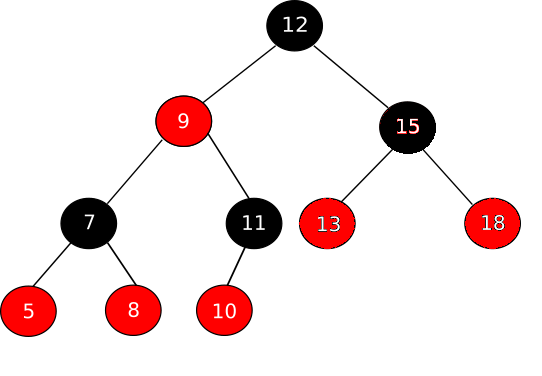
Preorder:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| B | R | B | R | B | R | B | B | R | R |

Postorder:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| R | B | B | R  i | B | R | R | B | R | B |

Example 2:



Inorder

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| R | B | R  i | R | R | B | B | R | B | R |

Preorder

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| B | R | B | R | R | B | R | B | R | R |

Postorder

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| R | R | B | R | B | R | R | R | B | B |

No. of black=1

No. of red=2

We can see that no. of black in all three array are equal and so is no. of red but i+1 is not node because i+1 is not black as root node is black. Also preorder[i+2] is not equal to postorder[size-2].

So we continue further.

Inorder

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| R | B | R | R | R | B  i | B | R | B | R |

Preorder

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| B | R | B | R | R | B | R | B | R | R |

Postorder

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| R | R | B | R | B | R | R | R | B | B |

No. of black=2

No. of red=4

Now we can see that no. of black in all three array are equal and so is no. of red and i+1 in black.

Also preorder[i+2] is equal to postorder[size-2], so i+1 is the root node position in inorder array;

Code:

import java.util.ArrayList;

import java.util.Scanner;

public class RBTree\_GFG {

static int rootNode(int n, char[] inorder, char[] preorder, char[] postorder){

ArrayList<Integer> arr = new ArrayList<>();

int root;

int post\_R = 0, post\_B = 0; //No. of black and red node in postorder

int pre\_R = 0, pre\_B = 0; //No. of black and red node in preorder

int in\_R = 0, in\_B = 0; //No. of black and red node in inorder

int p = 0;

for( int i=0; i<n-2; i++ ){

if(preorder[i+1] == postorder[i]){

if(postorder[i] == 'R'){

post\_R++;

pre\_R++;

}

else{

post\_B++;

pre\_B++;

}

}

else{

if(postorder[i] == 'B' && preorder[i+1] == 'R'){

post\_B++;

pre\_R++;

}

else{

post\_R++;

pre\_B++;

}

}

if(post\_B == pre\_B && post\_R == pre\_R && post\_B != 0){

for(int j=p; j<=i; j++){

if(inorder[j] == 'B')

in\_B++;

else

in\_R++;

}

p=i+1;

if(in\_B == post\_B && in\_R == post\_R && in\_B != 0){

if(inorder[i+1] == 'B' && preorder[i+2] == postorder[n-2]){

arr.add(i+2);

}

}

}

}

if( arr.size() == 1 )

root = arr.get(0);

else

root = arr.get(arr.size()-2);

return root;

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

int n;

System.out.print("Enter No of Nodes: ");

n = sc.nextInt(); //Take No of Node input

char[] inorder = new char[n];

char[] preorder = new char[n];

char[] postorder = new char[n];

System.out.println("Enter Inorder Traversal: ");

for(int i=0; i<n; i++)

inorder[i] = sc.next().charAt(0); //Take color code input in inorder

System.out.println("Enter Preorder Traversal: ");

for(int i=0; i<n; i++)

preorder[i] = sc.next().charAt(0); //Take color code input in inorder

System.out.println("Enter Postorder Traversal: ");

for(int i=0;i<n;i++)

postorder[i] = sc.next().charAt(0); //Take color code input in inorder

int root = rootNode(n, inorder, preorder, postorder);

System.out.println("Position of Root in Inorder is " + root);

}

}

Output 1:

Enter No of Nodes: 10

Enter Inorder Traversal:

B R R B B B R R B R

Enter Preorder Traversal:

B R B R B R B B R R

Enter Postorder Traversal:

R B B R B R R B R B

Position of Root in Inorder is 5

Output 2:

Enter No of Nodes: 10

Enter Inorder Traversal:

R B R R R B B R B R

Enter Preorder Traversal:

B R B R R B R B R R

Enter Postorder Traversal:

R R B R B R R R B B

Position of Root in Inorder is 7

<https://lwn.net/Articles/184495/>