Target Sum Pair BST

1st Approach (Naïve)

1. First find the complement(Tar - node.data)
2. Use find function to find the comp.

2nd Approach

* In a arraylist add all nodes with inorder(increasing order)
* Then use 2 pointers to find the target sum.

3rd Approach

* In this instead of using recursive traversal , we will use iterative travelsal for left pointer

And reverse iterative traversal for right pointer ,and rest is just comparing left +right == target.

Space Complexity = h(height of stack or tree)

Time complexity = n(no. of nodes)

Public static class Itpair{

Node node;

Int state=0;

}

Public static void bestapproach(Node node,int tar){

Stack<ITPair> ls=new Stack<>();

Stack<ITPair> rs=new Stack<>();

Ls.push(newITPair(node,0);

rs.push(newITPair(node,0);

Node left = getNextFromNormalInorder(ls);

Node right=getNextFromReverseInorder(rs);

If(left.data+right.data<tar){

left = getNextFromNormalInorder(ls);

}else if(left.data+right.data>tar){

right=getNextFromReverseInorder(rs);

}else{

System.out.println(left.data+” “+right.data);

left = getNextFromNormalInorder(ls);

right=getNextFromReverseInorder(rs);

}

}

Public static Node getNextFromNormalInorder(Stack<ItPair> st){

While(st.size>0){

ITPair top = st.peek();

If(top.state==0){

If(top.node.left!=null){

St.push(new ITPair(top.node.left ,0));

}

Top.state++;

}

}else if(top.state==1){

If(top.node.right!=null){

St.push(new ITPair(top.node.right ,0));

}

Top.state++;

Return top.node;

}

Else{

St.pop();

}

Return null // code will never reach here ,but still!

}

Public static Node getNextFromNormalInorder(Stack<ItPair> st){

While(st.size>0){

ITPair top = st.peek();

If(top.state==0){

If(top.node.right!=null){

St.push(new ITPair(top.node.right ,0));

}

Top.state++;

}

}else if(top.state==1){

If(top.node.leftt!=null){

St.push(new ITPair(top.node.leftt ,0));

}

Top.state++;

Return top.node;

}

Else{

St.pop();

}

Return null // code will never reach here ,but still!

}

Complexity

Time comp.

N=no. of nodes

1st approach = n x h

2nd Approach = n

Space Comp.

1st Approach = height of tree(h)

2nd Approach = n