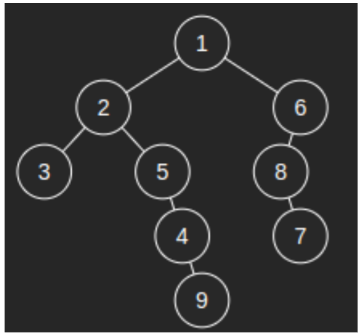
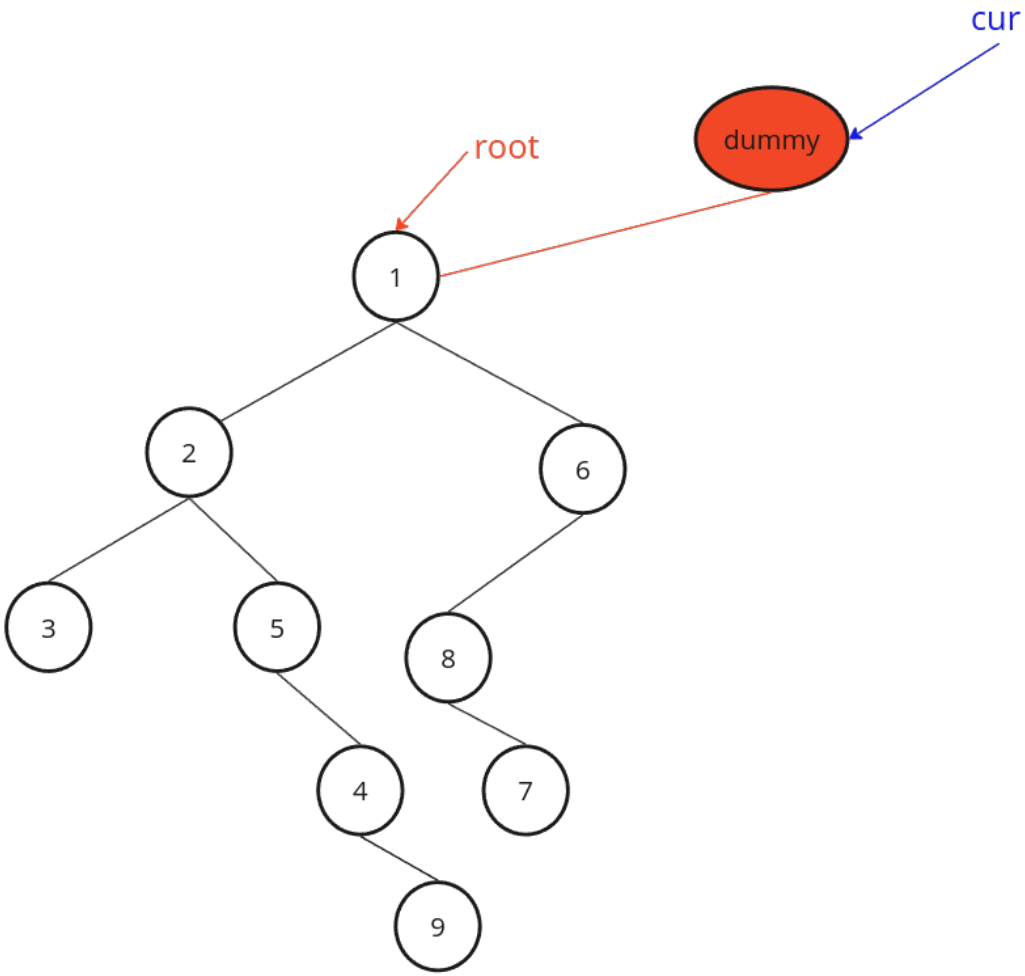
Post order traversal in a binary tree using Morris algorithm



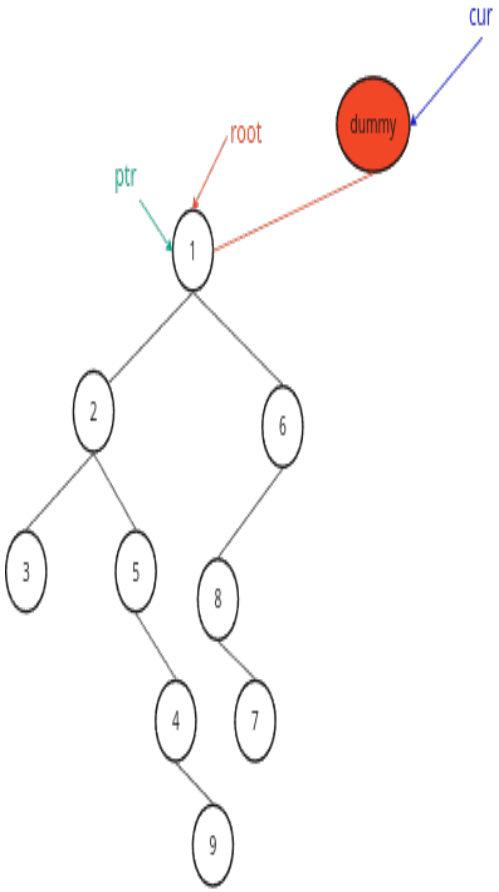
post order: [3,9,4,5,2,7,8,6,1]

Initiallization:

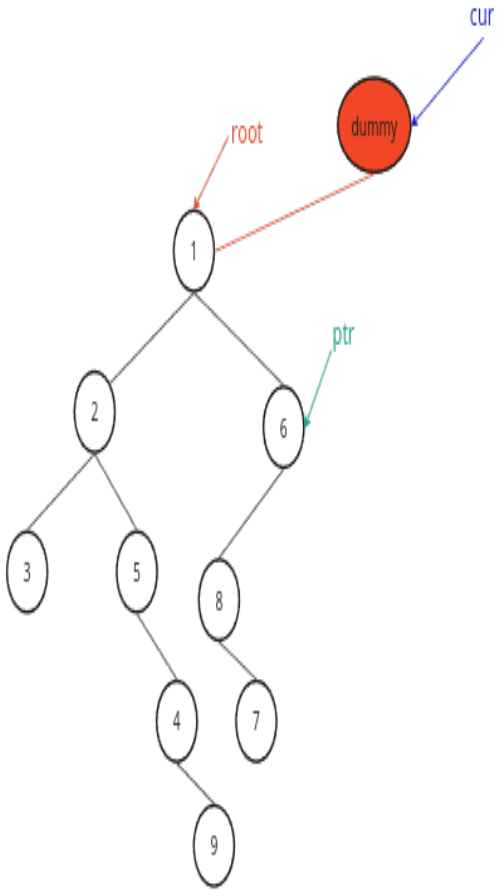



```
TreeNode* dummy = new TreeNode();
 dummy->left=root;
 TreeNode* cur=dummy;
```

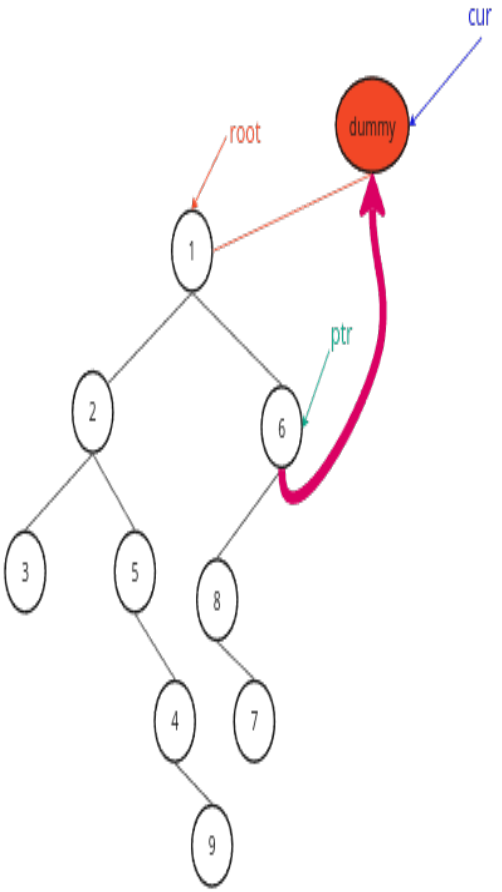
Iteration #1:



TreeNode* ptr=cur->left;

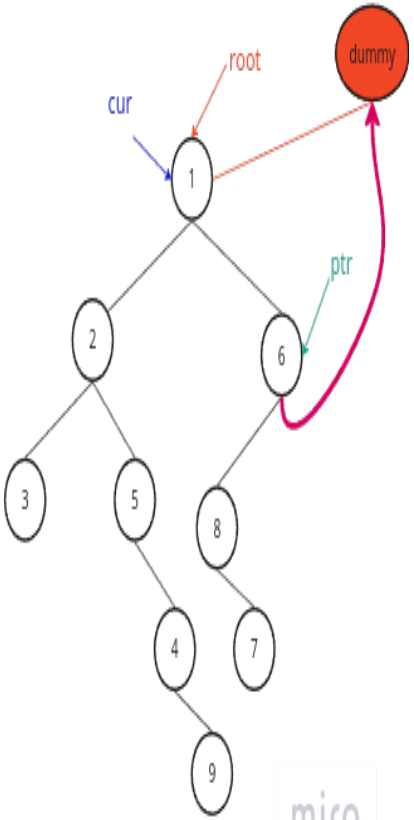


while(ptr->right && ptr->right!=cur) ptr = ptr->right;



No right child for ptr

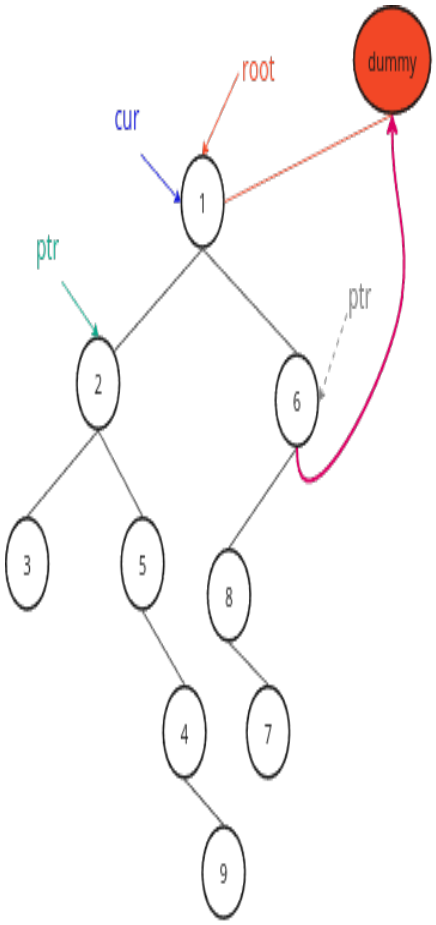
ptr's right will be the current node: ptr->right=cur



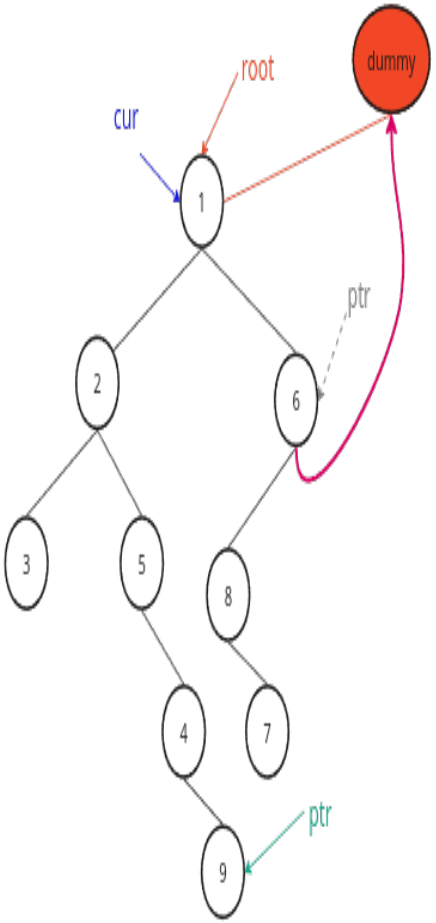
and current pointer goes to its left

cur=cur->left

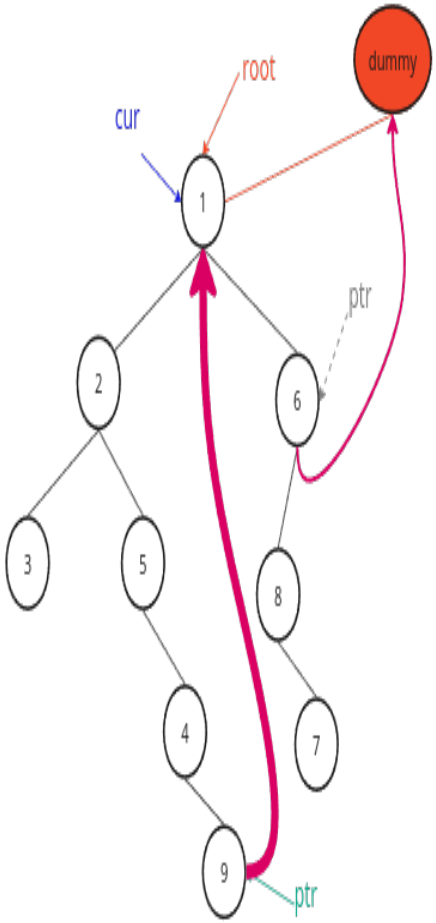
Iteration #2:



TreeNode* ptr=cur->left;

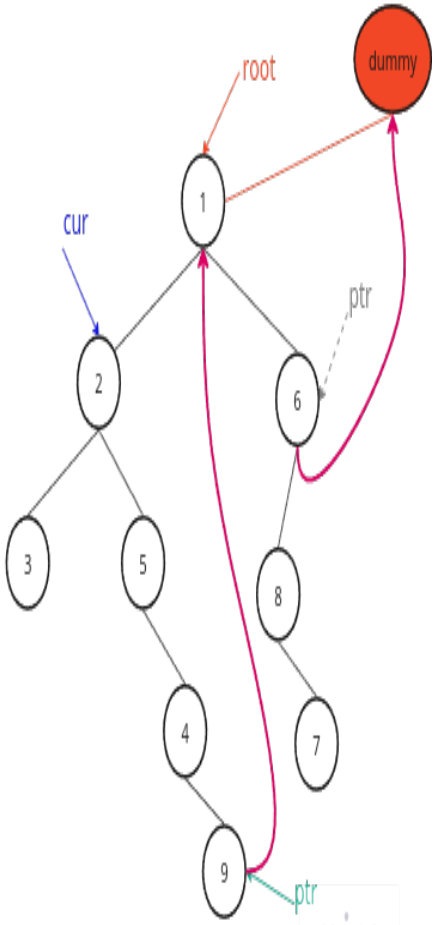


while(ptr->right && ptr->right!=cur) ptr = ptr->right;



No right child for ptr

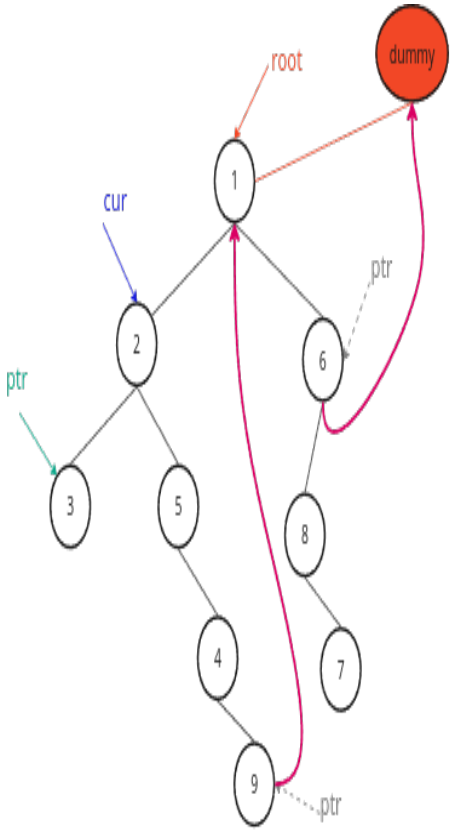
ptr's right will be the current node: ptr->right=cur



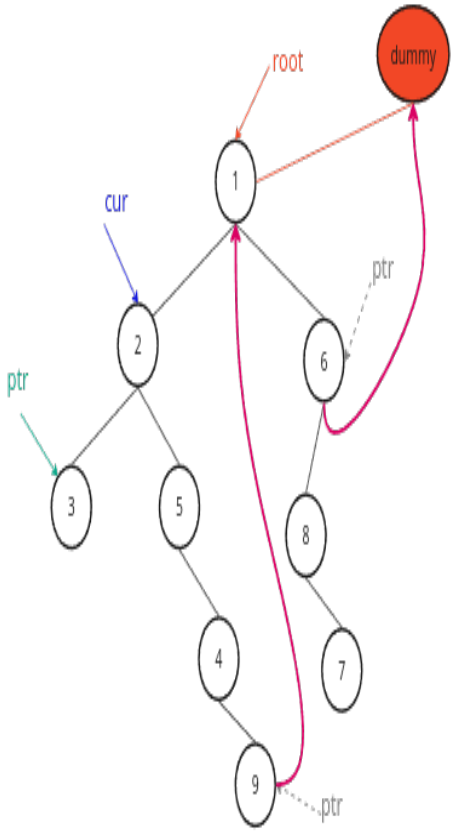
and current pointer goes to its left

cur=cur->left

Iteration #3:

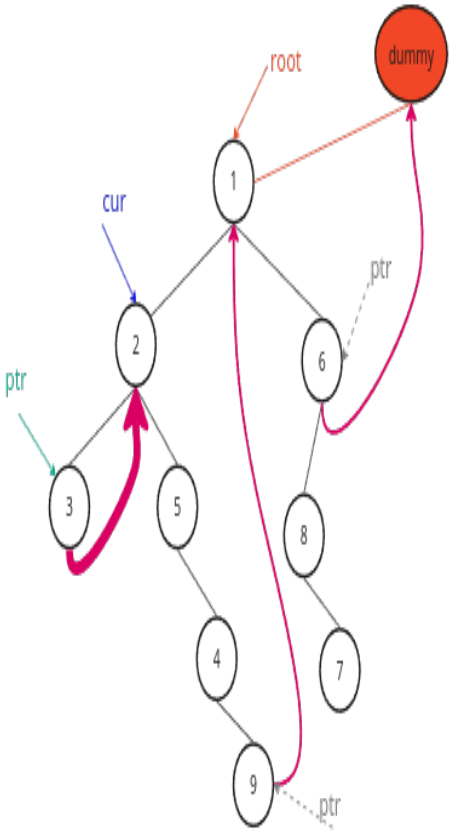


TreeNode* ptr=cur->left;



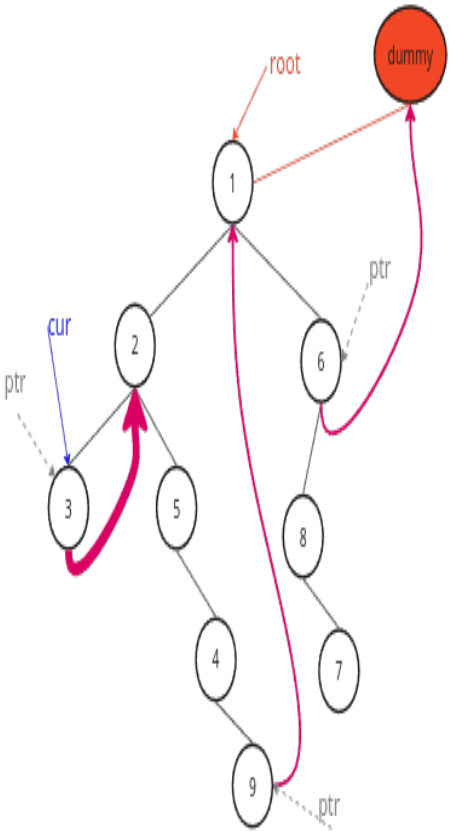
while(ptr->right && ptr->right!=cur) ptr = ptr->right;

no right child: while loop breaks , ptr remain in place



No right child for ptr

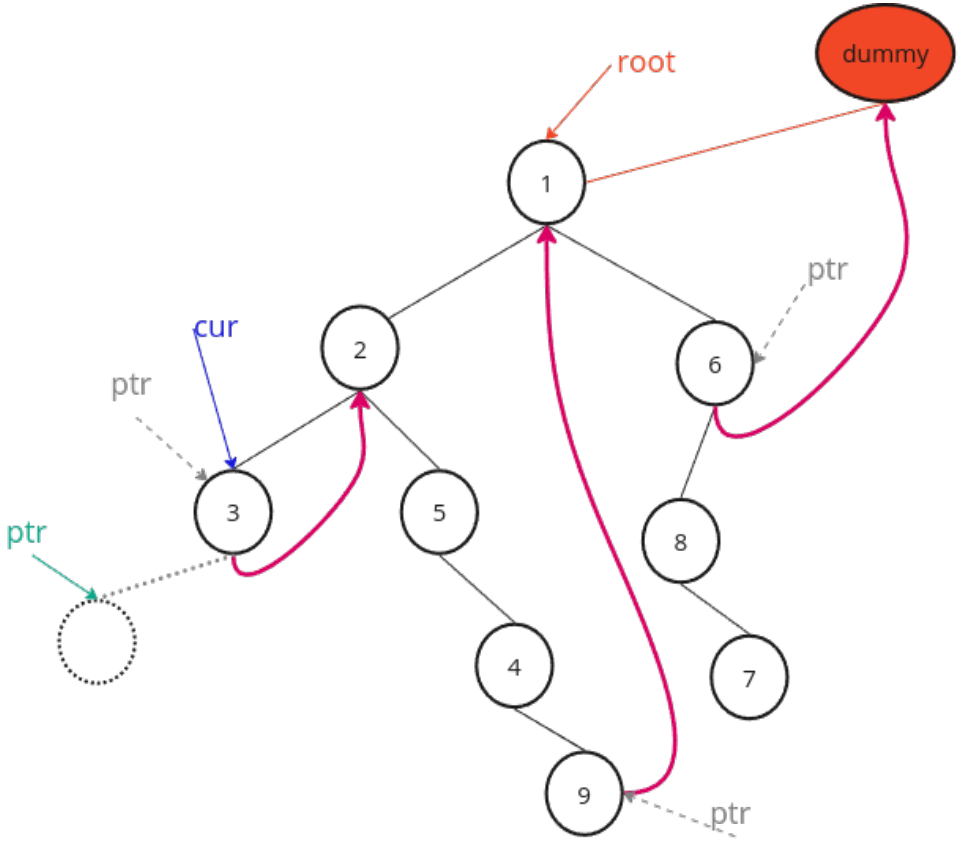
ptr's right will be the current node: ptr->right=cur



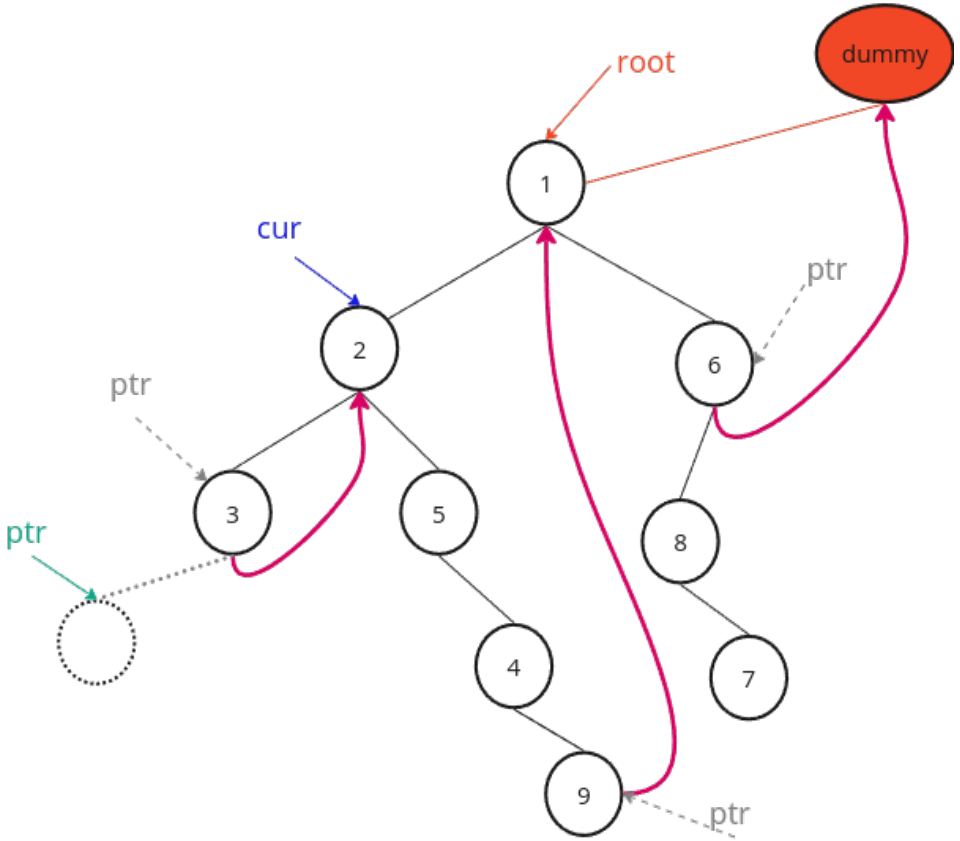
and current pointer goes to its left

cur=cur->left

Iteration #4:

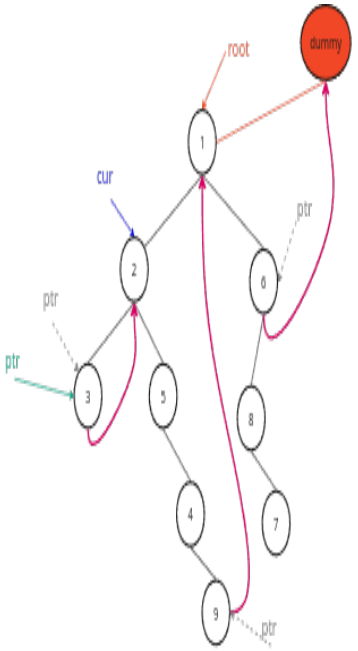


TreeNode* ptr=cur->left;
ptr points to null

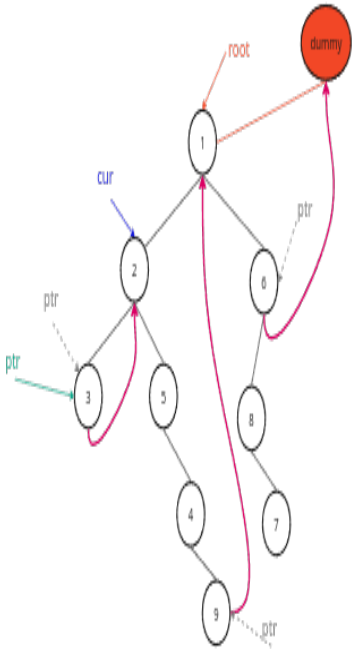


and current pointer goes to its right
cur=cur->right

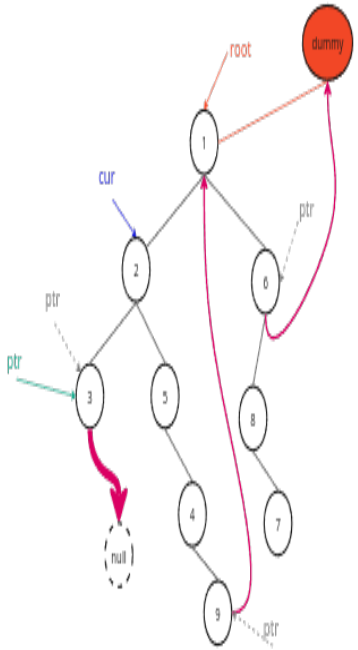
Iteration #5:



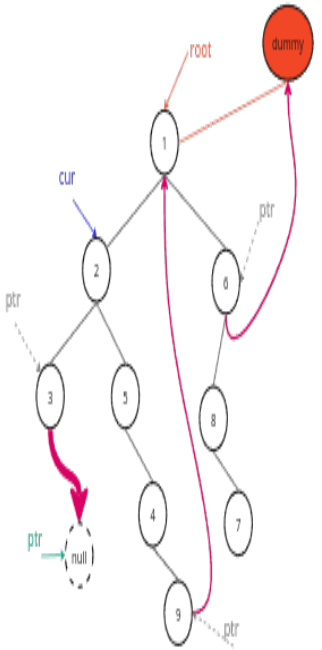
TreeNode* ptr=cur->left;



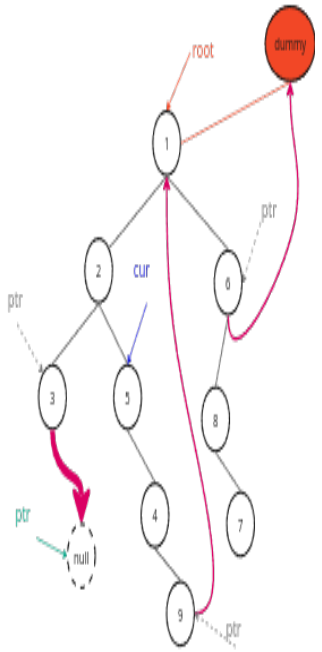
while(ptr->right && ptr->right!=cur) ptr = ptr->right;
ptr->right is equal to the current node: while loop breaks



ptr right's points to null
ptr points to current's node left

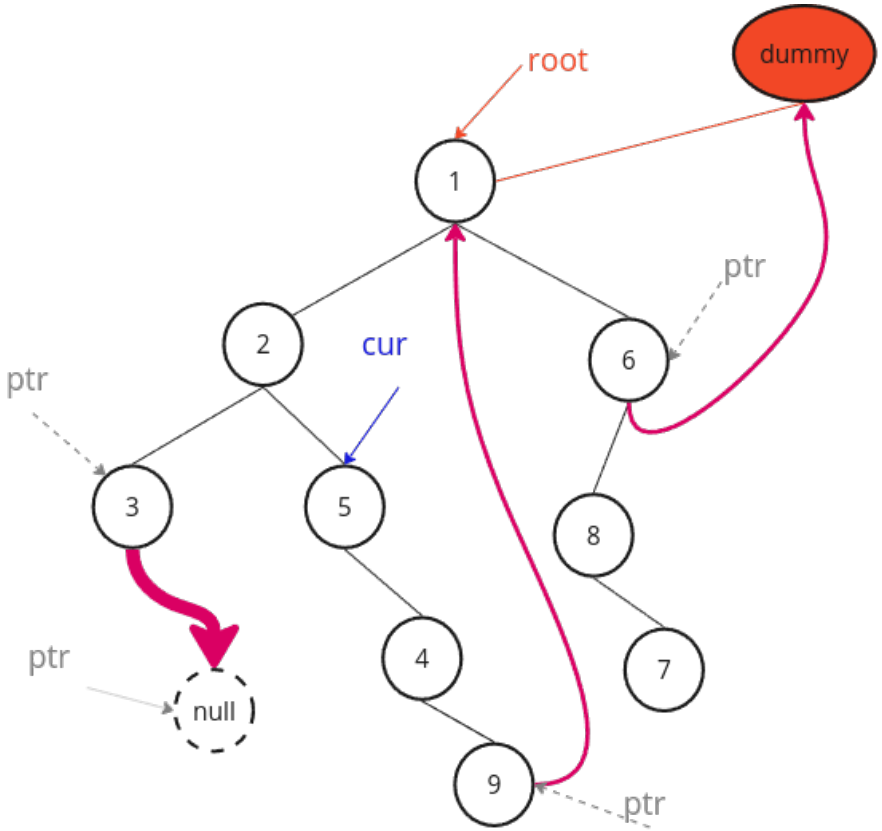


m=size of ans=0
while(ptr){
ans.push_back(ptr->val);
ptr=ptr->right;
}
ans={3}

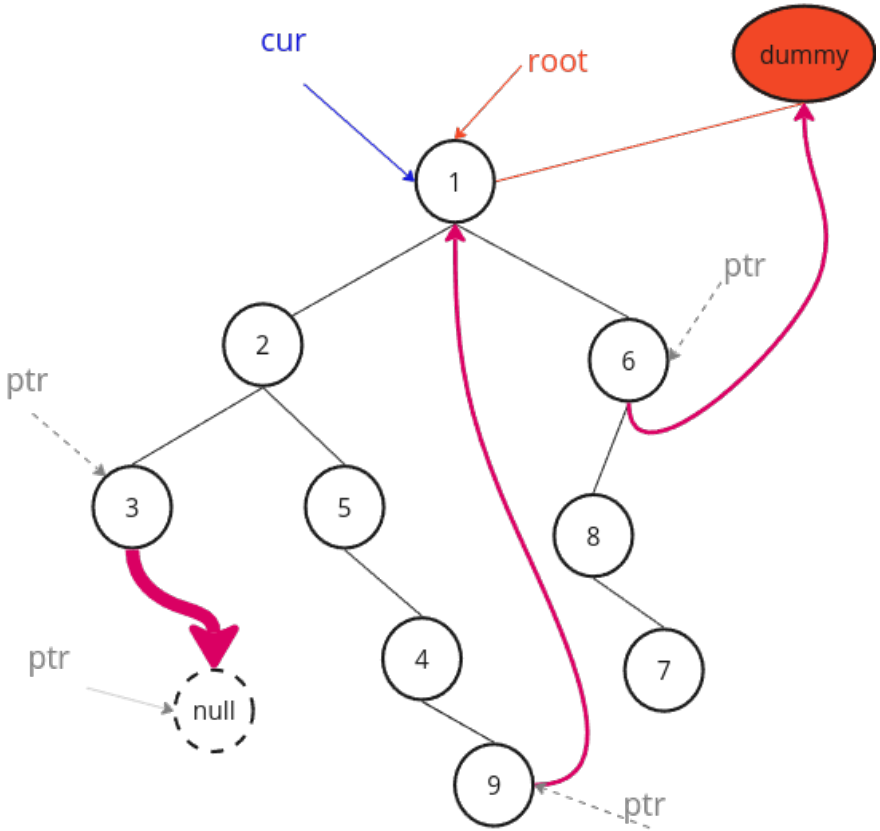


reverse(ans.begin()+0, ans.end());
ans={3}
cur=cur->right;

Iteration #6:

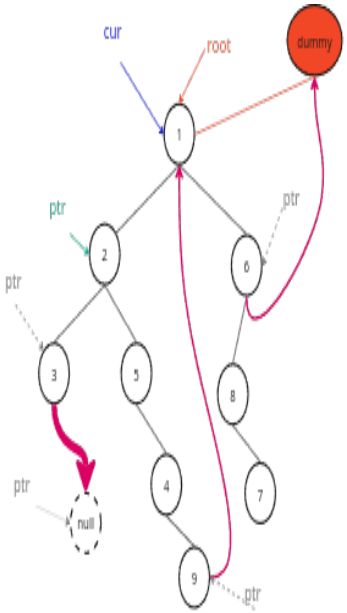


TreeNode* ptr=cur->left;
No lefts for the current nodes,

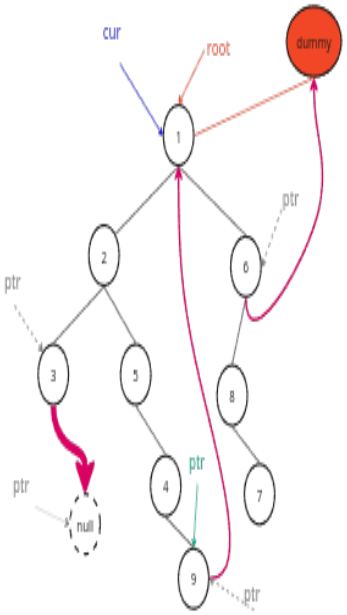


so cur will keep goin right

Iteration #7:

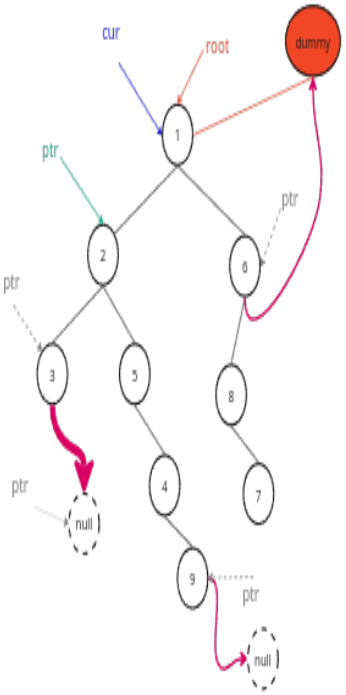


TreeNode* ptr=cur->left;



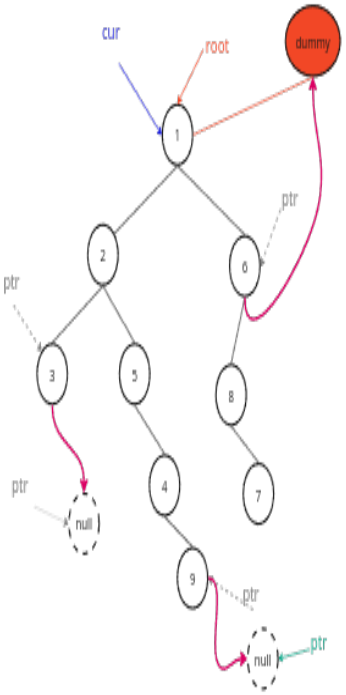
while(ptr->right && ptr->right!=cur) ptr = ptr->right;

ptr->right is equal to the current node: while loop breaks



ptr right's points to null

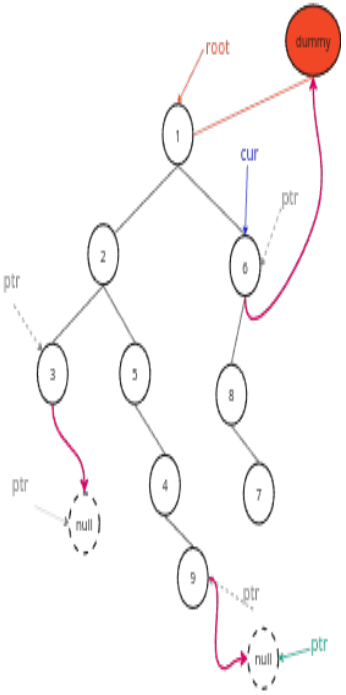
ptr points to current's node left



m=size of ans=1

```
while(ptr){
    ans.push_back(ptr->val);
    ptr=ptr->right;
}
```

ans={3,2,5,4,9}

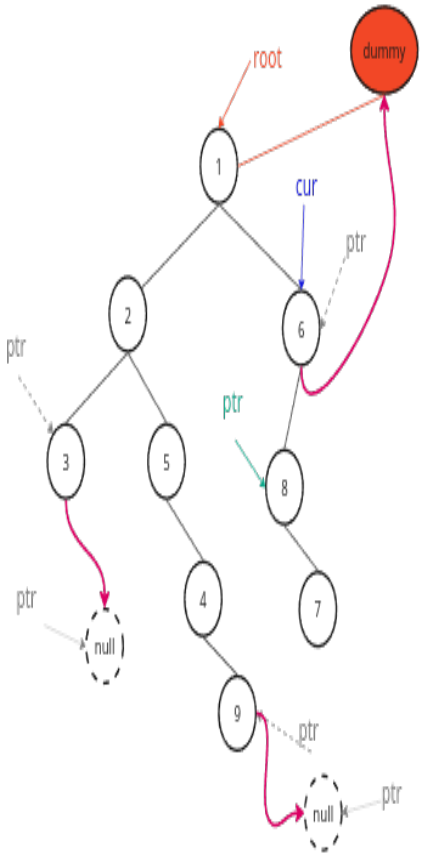


reverse(ans.begin()+1,ans.end());

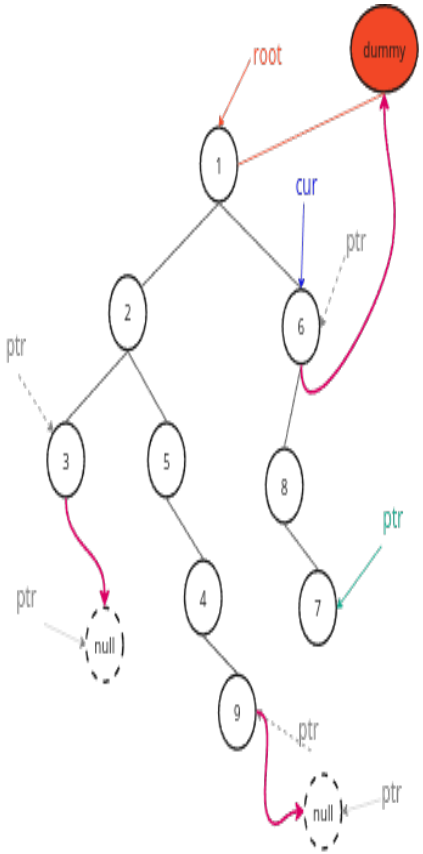
ans={3,9,4,5,2}

cur=cur->right;

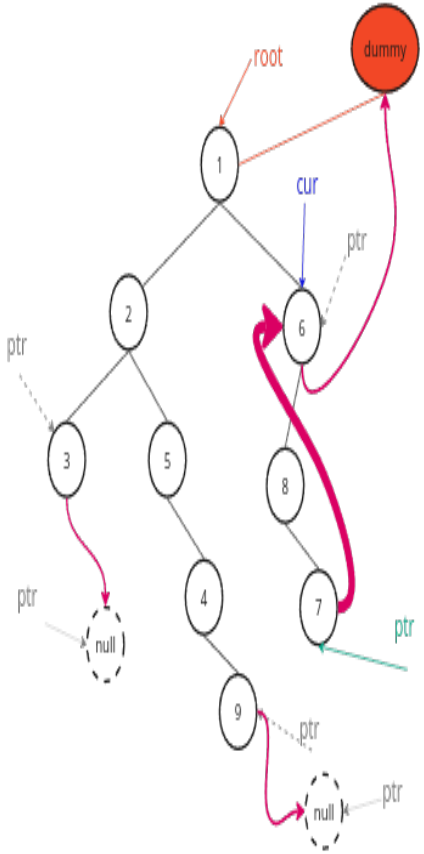
Iteration #8:



TreeNode* ptr=cur->left;

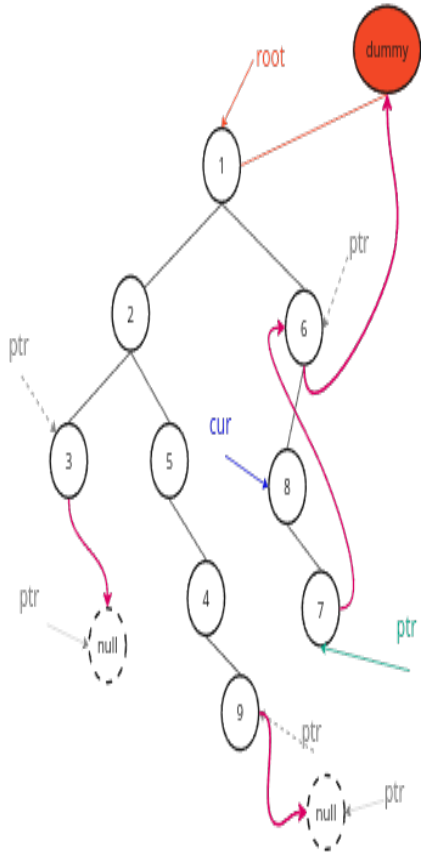


while(ptr->right && ptr->right!=cur) ptr = ptr->right;



No right child for ptr

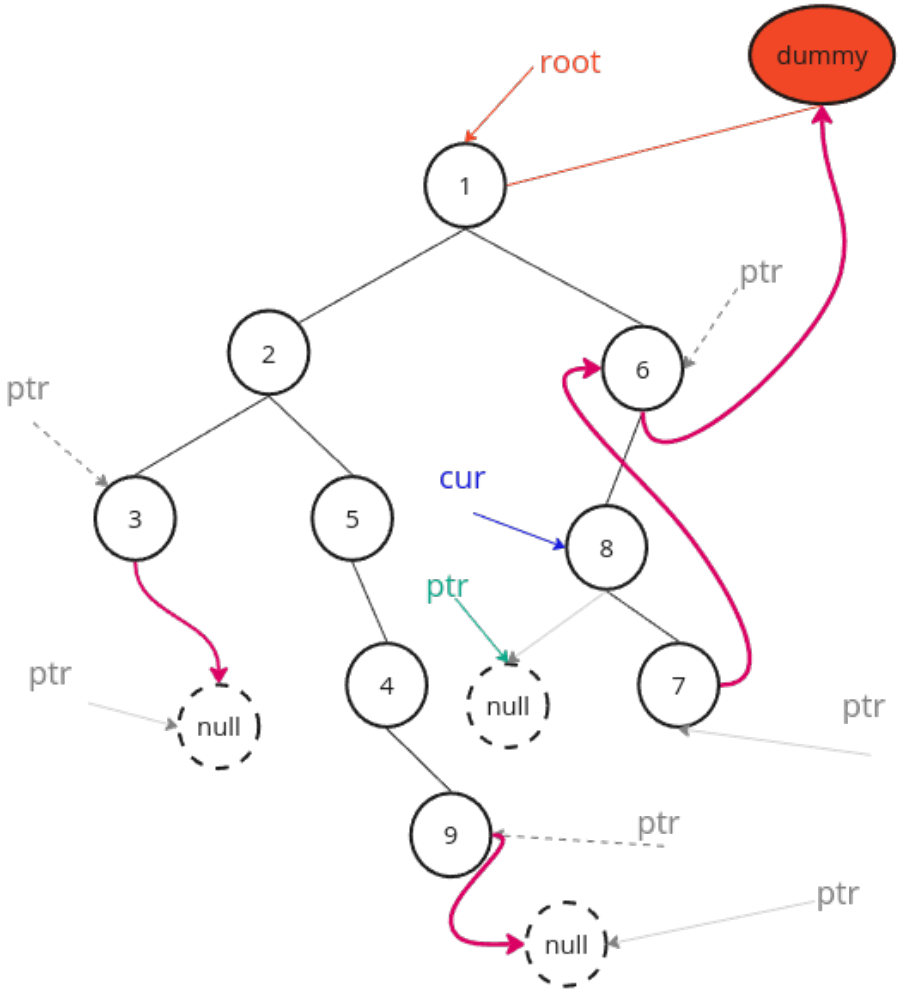
ptr's right will be the current node: ptr->right=cur



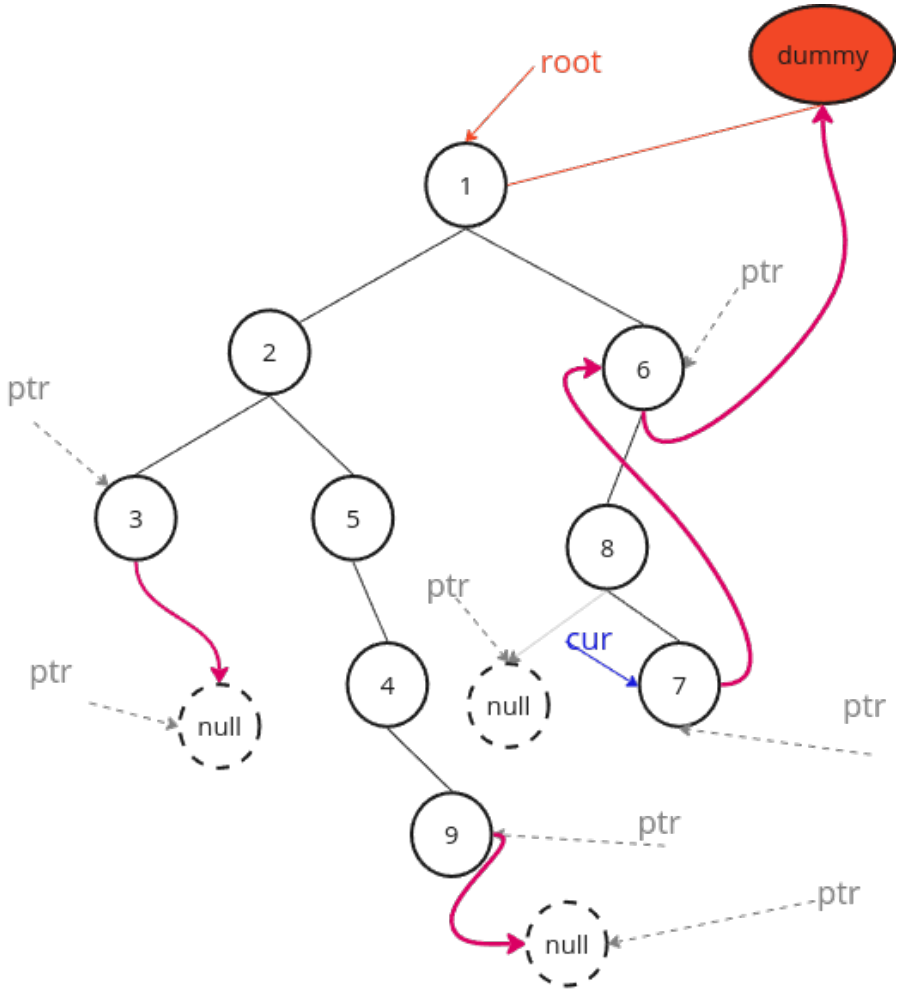
and current pointer goes to its left

cur=cur->left

Iteration #9:

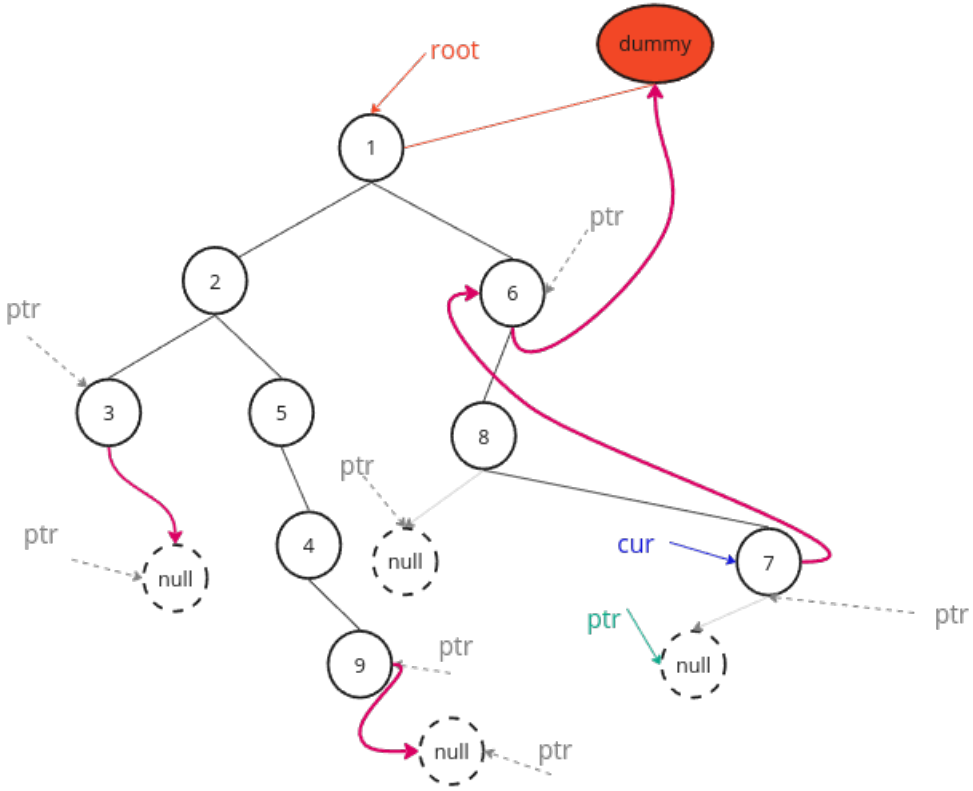


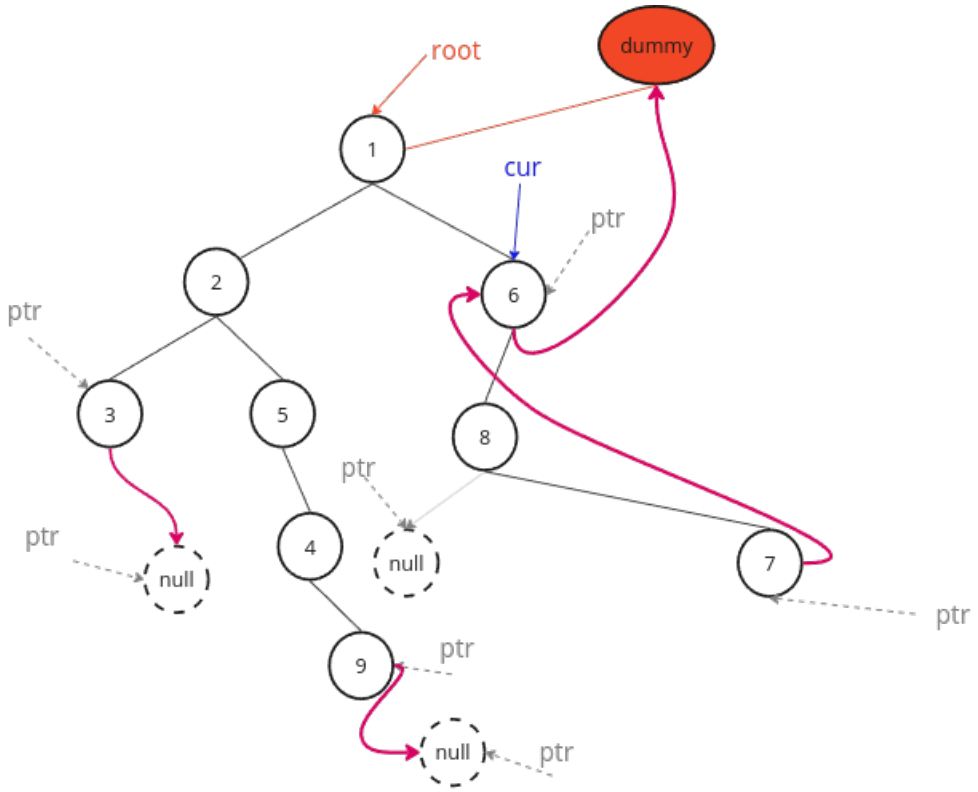
TreeNode* ptr=cur->left;
ptr points to null:



cur go right

Iteration #10:

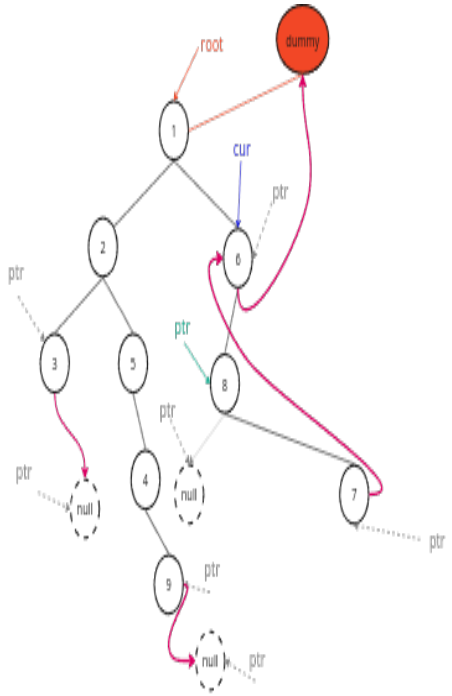




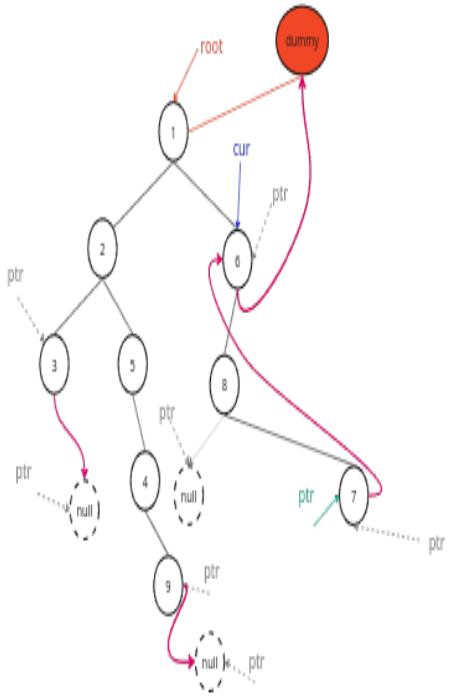
cur go right

TreeNode* ptr=cur->left;
ptr points to null:

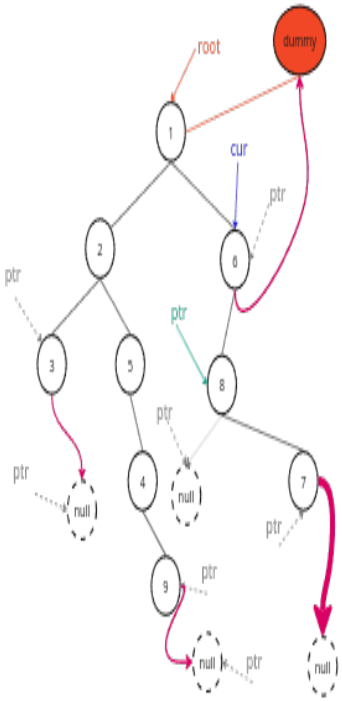
Iteration #11:



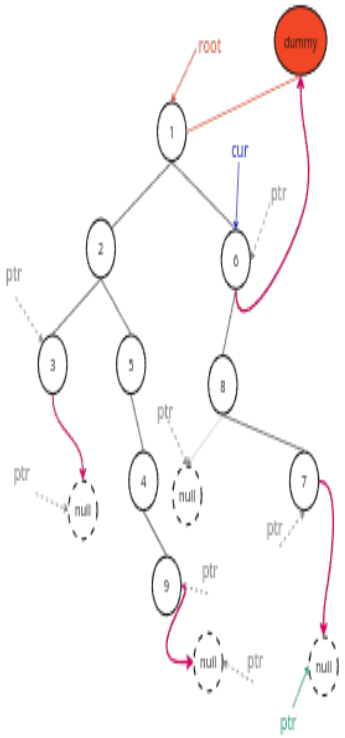
TreeNode* ptr=cur->left;



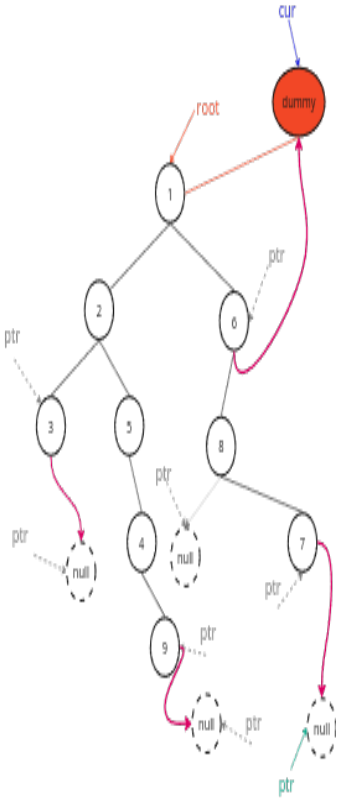
while(ptr->right && ptr->right!=cur) ptr = ptr->right;
ptr->right is equal to the current node: while loop breaks



ptr right's points to null
ptr points to current's node left

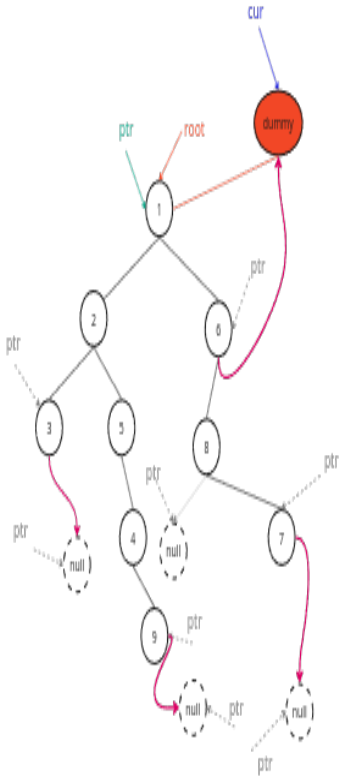


m=size of ans=5
while(ptr){
    ans.push_back(ptr->val);
    ptr=ptr->right;
}
ans={3,9,4,5,2,8,7}

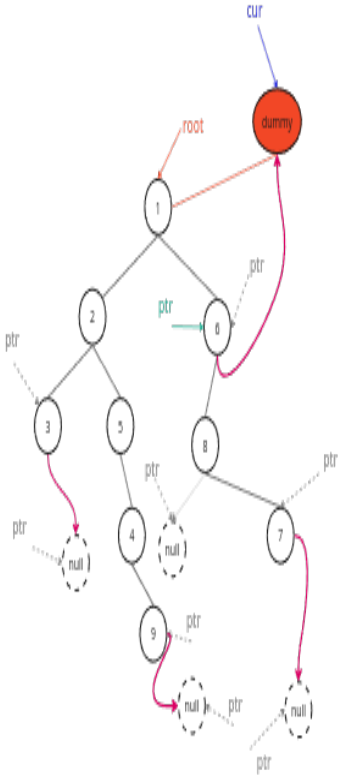


reverse(ans.begin()+3, ans.end());
ans={3,9,4,5,2,7,8}
cur=cur->right;

Iteration #12:

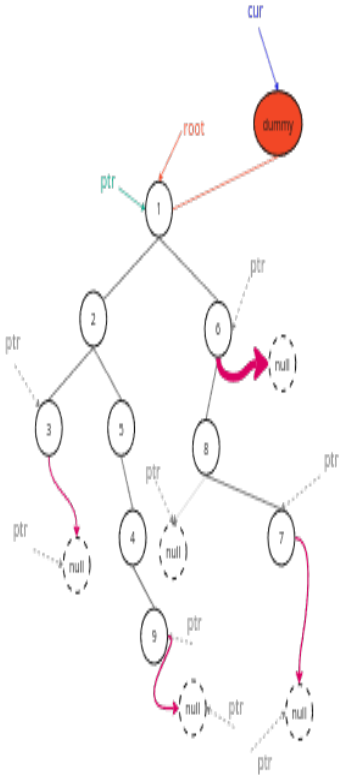


TreeNode* ptr=cur->left;



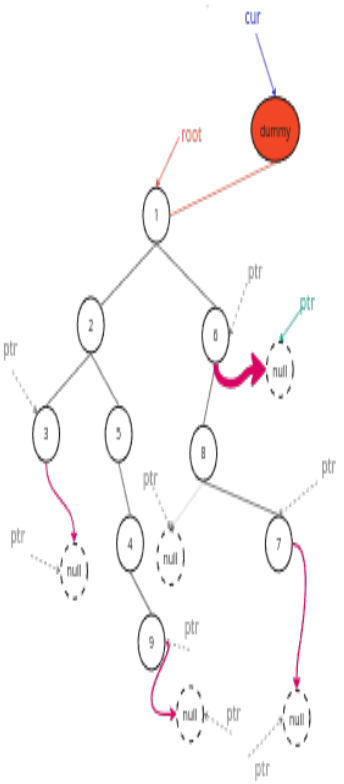
while(ptr->right && ptr->right!=cur) ptr = ptr->right;

ptr->right is equal to the current node: while loop breaks



ptr right's points to null

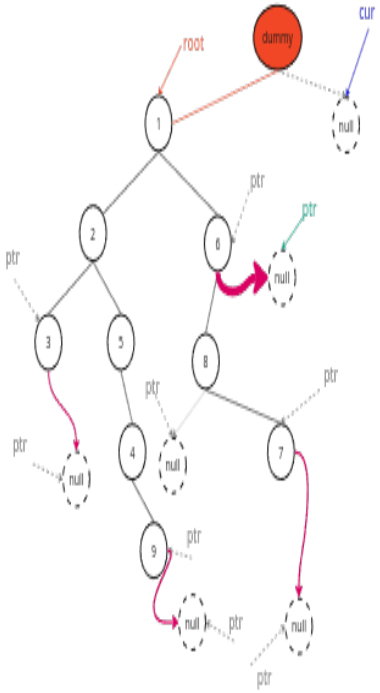
ptr points to current's node left



m=size of ans=7

```
while(ptr){
    ans.push_back(ptr->val);
    ptr=ptr->right;
}
```

ans={3,9,4,5,2,7,8,1,6}



reverse(ans.begin()+7, ans.end());

ans={3,9,4,5,2,7,8,6,1}

cur=cur->right;

cur points ti null: END