Data Structures

Binary Search Tree

For a binary tree to be a binary search tree, the data of all the nodes in the left sub-tree of the root node should be $\leq$ the data of the root. The data of all the nodes in the right subtree of the root node should be $>$ the data of the root.

Example

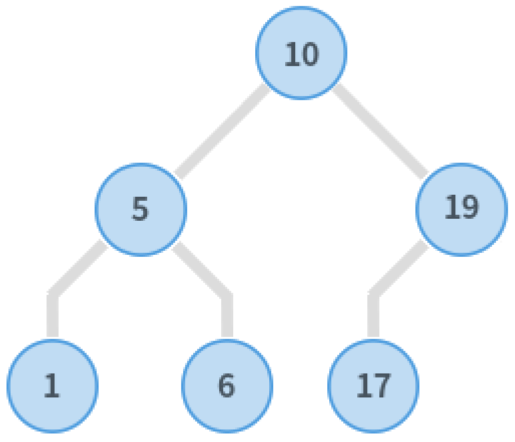


Fig. 1

In Fig. 1, consider the root node with data = 10.

- Data in the left subtree is: $[5, 1, 6]$
- All data elements are < 10
- Data in the right subtree is: $[19, 17]$
- All data elements are > 10

Also, considering the root node with $data = 5$, its children also satisfy the specified ordering. Similarly, the root node with $data = 19$ also satisfies this ordering. When recursive, all subtrees satisfy the left and right subtree ordering.

The tree is known as a Binary Search Tree or BST.

Traversing the tree

There are mainly *three* types of tree traversals.

Pre-order traversal

In this traversal technique the traversal order is root-left-right i.e.

- Process data of root node
- First, traverse left subtree completely
- Then, traverse right subtree

```
void perorder(struct node*root)
{
    if(root)
    {
        printf("%d ",root->data);    //Printf root->data
        preorder(root->left);    //Go to left subtree
        preorder(root->right);     //Go to right subtree
    }
}
```

Post-order traversal

In this traversal technique the traversal order is left-right-root.

- Process data of left subtree
- First, traverse right subtree
- Then, traverse root node

```
void postorder(struct node*root)
{
    if(root)
    {
        postorder(root->left);    //Go to left sub tree
        postorder(root->right);     //Go to right sub tree
        printf("%d ",root->data);    //Printf root->data
    }
}
```

In-order traversal

In in-order traversal, do the following:

- First process left subtree (before processing root node)
- Then, process current root node

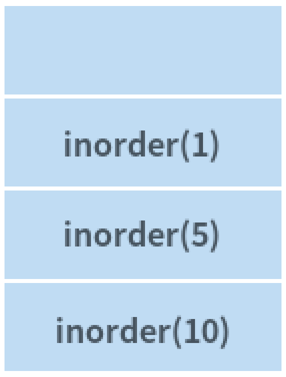
- Process right subtree

```
    void inorder(struct node*root)
    {
        if(root)
        {
            inorder(root->left);    //Go to left subtree
            printf("%d ",root->data);    //Printf root->data
            inorder(root->right);     //Go to right subtree
        }
    }
```

Consider the in-order traversal of a sample BST

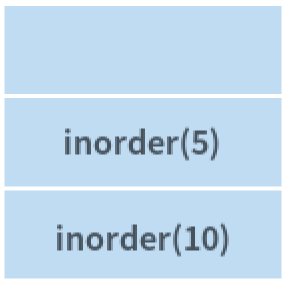
- The 'inorder( )' procedure is called with root equal to node with $data = 10$
- Since the node has a left subtree, 'inorder( )' is called with root equal to node with $data = 5$
- Again, the node has a left subtree, so 'inorder( )' is called with $root = 1$

The function call stack is as follows:



- Node with $data = 1$ does not have a left subtree. Hence, this node is processed.
- Node with $data = 1$ does not have a right subtree. Hence, nothing is done.
- `inorder(1)` gets completed and this function call is popped from the call stack.

The stack is as follows:

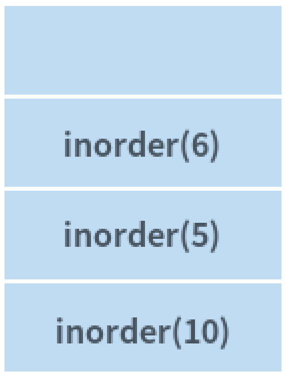


- Left subtree of node with $data = 5$ is completely processed. Hence, this node gets processed.
- Right subtree of this node with $data = 5$ is non-empty. Hence, the right subtree gets processed now. 'inorder(6)' is then called.

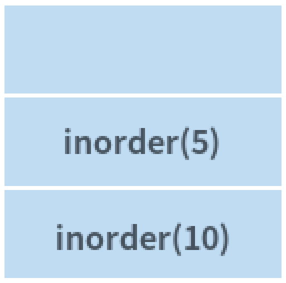
Note

'inorder(6)' is only equivalent to saying inorder(pointer to node with $data = 6$). The notation has been used for brevity.

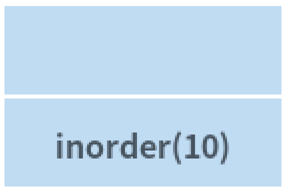
The function call stack is as follows:



Again, the node with $data = 6$ has no left subtree, Therefore, it can be processed and it also has no right subtree. 'inorder(6)' is then completed.



Both the left and right subtrees of node with $data = 5$ have been completely processed. Hence, `inorder(5)` is then completed.



- Now, node with $data = 10$ is processed
- Right subtree of this node gets processed in a similar way as described until step 10
- After right subtree of this node is completely processed, entire traversal of the BST is complete

The order in which BST in Fig. 1 is visited is: 1, 5, 6, 10, 17, 19. The in-order traversal of a BST gives a sorted ordering of the data elements that are present in the BST. This is an important property of a BST.

Insertion in BST

Consider the insertion of $data = 20$ in the BST.

Algorithm

Compare data of the root node and element to be inserted.

1. If the data of the root node is greater, and if a left subtree exists, then repeat step 1 with root = root of left subtree. Else, insert element as left child of current root.
2. If the data of the root node is greater, and if a right subtree exists, then repeat step 2 with root = root of right subtree. Else, insert element as right child of current root.

Implementation

```
    struct node* insert(struct node* root, int data)
    {
        if (root == NULL)    //If the tree is empty, return a new,single 
node
            return newNode(data);
        else
        {
            //Otherwise, recur down the tree
            if (data <= root->data)
                root->left  = insert(root->left, data);
            else
                root->right = insert(root->right, data);
            //return the (unchanged) root pointer
            return root;
        }
    }
```

Contributed by: Vaibhav Tulsyan

TEST YOUR UNDERSTANDING

Create BST

Create a Binary Search Tree from list A containing N elements. Insert elements in the same order as given. Print the pre-order traversal of the subtree with root node data equal to Q (inclusive of Q), separating each element by a space.

Input:
First line contains a single integer N – number of elements.
Second line contains N space-separated integers.

Third line contains a single integer Q – the element whose subtree is to be printed in pre-order form.

Output:
Print K space-separated integers – where K is the number of elements in the subtree of Q (inclusive of Q)

Constraints:

$1 \leq N \leq 10^3$
$-10^9 \leq A[i] \leq 10^9$

SAMPLE INPUT

```
4
2 1 3 4
3
```

SAMPLE OUTPUT

```
3
4
```

Enter your code or Upload your code as file. Save C (gcc 4.8.2)

```
#include <stdio.h>

int main()
{
    printf("Hello World!\n");
    return 0;
}

```

1:1

Provide custom input

COMPILE & TEST SUBMIT

Press Ctrl-space for autocomplete suggestions. POWERED BY code table

Need Help ?

In case you feel you are stuck with the problem, you can view our editorial.
Remember this is just to help you out, in order to learn you should try your best.

VIEW EDITORIAL

10 LIVE EVENTS

COMMENTS (7) SORT BY: Relevance

Join Discussion...

Cancel Post

LONE_WOLF Edited 3 months ago

please someone see the error in my code .i m not able to get it

```
#include<stdio.h>
#include<stdlib.h>
struct tree {
int data;
struct tree *left;
struct tree *right;
};
struct tree * maketree(long long int info)
{
struct tree *ps;
ps=(struct tree *)malloc(sizeof(struct tree ));
ps->data=info;
ps->right=NULL;
ps->left=NULL;
return ps;
}
void setleft(struct tree *ps1 ,long long int info)
{
if(ps1->left!=NULL)
{
printf("Invalisd insertiion");
}
else
{
ps1->left=maketree(info);
}
}
void setright(struct tree *ps1 ,long long int info)
{struct tree *ps;
if(ps1->right!=NULL)
{
```

```
printf("Invalisd insertiion");
}
else
{
ps1->right=maketree(info);
}
}
void intrav(struct tree *ps)
{
if(ps!=NULL)
{
printf("%lld",ps->data);
intrav(ps->left);
intrav(ps->right);
}
}
int main()
{
struct tree *ps,*q,*p,*ps1;
long long int A[1000],info;
int n,i,k;
i=0;
scanf("%d",&n);
while(i<n)
{
scanf("%lld ",&info);
A[i]=info;
i++;
}
printf("\n");
scanf("%d",&k);
i=0;
ps=maketree(A[0]);
i++;
if(A[0]==k)
{
ps1=ps;
}
while(i<n)
{
p=q=ps;
while(q!=NULL)
{
p=q;
if(p->data<A[i])
{
q=p->left;
}
else
{
q=p->right;
}
}
if(q->data<A[i])
{
setleft(p,A[i]);
}
else
{
setright(p,A[i]);
}
if(A[i]==k)
{
ps1=p;
}
}
intrav(ps1);return 0;
}
```

▲ 0 votes ● Reply ● Message ● Permalink

10

LIVE EVENTS

homputr 3 months ago

There is a typo in section "Pre-order traversal".

====

> "void perorder(.....)"

Should be:

"void preorder(....)"

====

▲ 0 votes ● Reply ● Message ● Permalink

kashish miglani 3 months ago

IF IN CASE AYONE NEEDS HELP

```
#include<iostream>
#include<cstring>
#include<stdio.h>
#include<iomanip>
#include<math.h>
#include<string>
#include<algorithm>
#include<stack>
#include<string>
#include<list>
#include<limits.h>
#include<cstdio>
#include<cstdlib>
#include<cstring>
#include<queue>
#include<deque>
#include<stack>
#include<cmath>
#include<numeric>
#include<map>
#include<vector>
#include<set>
#include<queue>
#define ll long long
#define str string
#define fast ios_base::sync_with_stdio(false),cin.tie(0),cout.tie(0);
#define vec vector<int>
#define long_vec vector<ll>
#define out cout<<
#define in cin>>
#define rep(a,b) for(int i=a;i<b;i++)
#define repl(a,b) for(ll i=a;i<b;i++)
#define nl cout<<endl;
#define sortv(v) sort(v.begin(),v.end())
#define ret0 return 0;
#define countr(v,a) (int)count(v.begin(),v.end(),a)
#define FILL(a,b) memset((a),(b),sizeof((a)))
#define start int main()
#define print(v) repl(0,v.size()){out v[i]<<" ";}
#define err(v) v.erase(v.begin(),v.end());
using namespace std;
struct tree
{
int data;
struct tree*l,*r;
};
```

```
typedef struct tree* tr;
tr getnode(int n)
{
tr new_node=(tr)malloc(sizeof(tree));
new_node->data=n;
new_node->l=new_node->r=NULL;
return new_node;
}
void insertt(tr root,tr node)
{
if(root)
{
if(root->data>node->data)
{
if(root->l==NULL)
root->l=node;
else
insertt(root->l, node);
}
else
{
if(root->r==NULL)
root->r=node;
else
insertt(root->r, node);
}
}
else
{
root=node;
}
}
void preorder(tr root)
{
if(root)
{
out root->data;nl
preorder(root->l);
preorder(root->r);
}
}
tr searcht(tr root,int n)
{
while(root!=NULL)
{
if(n<root->data)
root=root->l;
else if(n>root->data)
root=root->r;
else
{
return root;
}
}
return root;
}
start
{
int n;
in n;
int inp;
in inp;
tr root=getnode(inp);
for(int i=0;i<n-1;i++)
{
```

```
in inp;
insertt(root, getnode(inp));
}
int k;
in k;
tr e=searcht(root, k);
preorder(e);
}
```

0 votes ● Reply ● Message ● Permalink

10

LIVE EVENTS

Nhân Nguyễn Thành 3 months ago

You should change the parameter type of root variable in insert function to: struct node ** root or struct node *& root

0 votes ● Reply ● Message ● Permalink

patel.mdkumar a month ago

```
#include <iostream>
using namespace std;
typedef struct node{
int data;
struct node *left;
struct node *right;
}node;
node* makeNode(int data){
struct node *n = (node*)malloc(sizeof(node));
n->data = data;
n->left = NULL;
n->right = NULL;
return n;
}
node* insertNode(struct node *root,int data)
{
if(root == NULL)
{
return makeNode(data);
}
else{
if(data <= root->data)
root->left = insertNode(root->left,data);
else
root->right = insertNode(root->right,data);
return root;
}

}
void preorder(struct node* root)
{
if(root == NULL)
return;
cout<<root->data<<"\n";
preorder(root->left);
preorder(root->right);
}
node* search(struct node* root,int data)
{
if(root->data == data)
{
return root;
}
else
{
if(root->data >= data)
root = search(root->left,data);
else
root = search(root->right,data);
return root;
}
}
int main()
{
int total_input,input[10000];
cin>>total_input;
for(int i=0;i<total_input;i++)
{
cin>>input[i];
}
int element;
cin>>element;
struct node * root = makeNode(input[0]);
for(int i=1;i<total_input;i++)
{
root = insertNode(root,input[i]);
}
preorder(search(root,element));
return 0;
}
```

0 votes ● Reply ● Message ● Permalink

Nagaraj Gidde a month ago

My program works fine for sample test case. However result is not the same for the actual test case. Can somebody point out the error in my code. Thanks in advance.

```
#include <stdio.h>
typedef struct node node_t;
struct node{
long data;
node_t* left;
node_t* right;
};
node_t *root=NULL;
node_t* create_node(long data)
{
node_t *tmp_node=NULL;
tmp_node=malloc(sizeof(node_t));
tmp_node->data=data;
tmp_node->left=NULL;
tmp_node->right=NULL;
return tmp_node;
}
node_t* insert_node(node_t* root,long data)
{
if( root==NULL )
root=create_node(data);
else if( data<root->data )
root->left=insert_node(root->left,data);
else
root->right=insert_node(root->right,data);

return root;
}
void pre_order_traversal(node_t* root)
{
if(root==NULL){
return;
}
pre_order_traversal(root->left);
printf("%d\n",root->data);
pre_order_traversal(root->right);
}
node_t* find_node(node_t* root,long data)
{
if( root==NULL)
return NULL;
if( root->data==data )
return root;
if( data<root->data )
find_node(root->left,data);
else
find_node(root->right,data);
}
int main()
{
int n;
scanf("%d",&n);
int i=0;
for(i=0; i<n; i++)
{
long data;
scanf("%li",&data);
root=insert_node(root,data);
}
long pre_element;
scanf("%li",&pre_element);
printf("pre_element:%li\n",pre_element);
node_t *pre_order_node = find_node(root,pre_element);
pre_order_traversal(pre_order_node);
return 0;
}
```

▲ 0 votes ● Reply ● Message ● Permalink

Naragoni Sairam 16 days ago

```
#include <stdio.h>
#include<stdlib.h>
struct node{
long int data;
struct node* left;
struct node* right;
};
struct node* newNode(long int data)
{
struct node* temp = (struct node*)malloc(sizeof(struct node));
temp->data=data;
temp->left=NULL;
temp->right=NULL;
return temp;
}
struct node* insert(struct node* root,long int data)
{

if(root==NULL)
{
root=newNode(data);
return root;
}
else if(data<=root->data)
{
root->left=insert(root->left,data);
}
else if(data>root->data)
{
root->right=insert(root->right,data);
}
return root;
}
struct node* search(struct node* root,long int data)
{
if(root==NULL)
return NULL;
else if(root->data==data)
return root;
else if(data<=root->data)
search(root->left,data);
else if(data>root->data)
search(root->right,data);
}
void preorder(struct node* root)
{
if(root==NULL)
return;
printf("%ld\n",root->data);
preorder(root->left);
preorder(root->right);
}
int main()
{
struct node* root;
struct node* temp;
root = NULL;
int n,i;
long int x;
scanf("%d",&n);
for(i=0;i<n;i++)
{
scanf("%ld",&x);
root = insert(root,x);
}
long int q;
scanf("%ld",&q);
temp = search(root,q);
preorder(temp);
return 0;
}
```

▲ 0 votes ● Reply ● Message ● Permalink