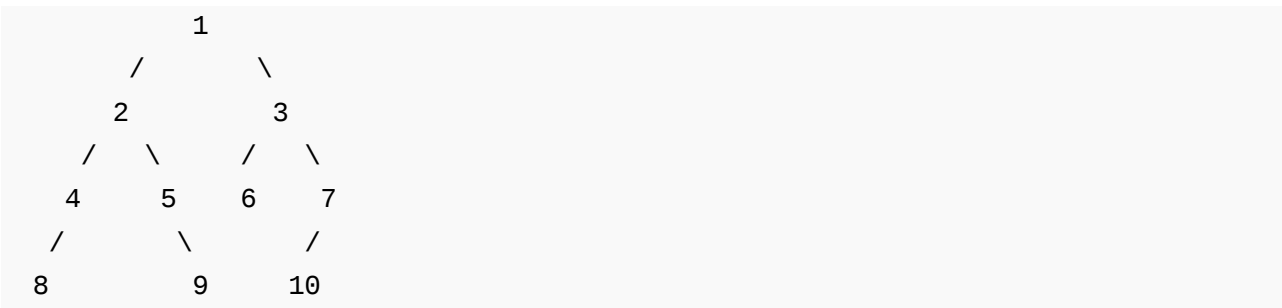


Home	Algorithms	DS	GATE	Interview Corner	Q&A	C	C++	Java	Books	Contribute	Ask a Q	About
Array	Bit Magic	C/C++	Articles	GFactS	Linked List	MCQ	Misc	Output	String	Tree	Graph	

Print ancestors of a given binary tree node without recursion

Given a Binary Tree and a key, write a function that prints all the ancestors of the key in the given binary tree.

For example, consider the following Binary Tree



Following are different input keys and their ancestors in the above tree

Input Key	List of Ancestors
1	
2	1
3	1
4	2 1
5	2 1
6	3 1
7	3 1
8	4 2 1

Google™ Custom Search



GeeksforGeeks



52,731 people like GeeksforGeeks.



Interview Experiences

Advanced Data Structures

Dynamic Programming

Greedy Algorithms

Backtracking

Pattern Searching

Divide & Conquer

Mathematical Algorithms

Recursion

Geometric Algorithms

```
9          5 2 1
10         7 3 1
```

Recursive solution for this problem is discussed [here](#).

It is clear that we need to use a stack based iterative traversal of the Binary Tree. The idea is to have all ancestors in stack when we reach the node with given key. Once we reach the key, all we have to do is, print contents of stack.

How to get all ancestors in stack when we reach the given node? We can traverse all nodes in Postorder way. If we take a closer look at the recursive postorder traversal, we can easily observe that, when recursive function is called for a node, the recursion call stack contains ancestors of the node. So idea is to do iterative Postorder traversal and stop the traversal when we reach the desired node.

Following is C implementation of the above approach.

```
// C program to print all ancestors of a given key
#include <stdio.h>
#include <stdlib.h>

// Maximum stack size
#define MAX_SIZE 100

// Structure for a tree node
struct Node
{
    int data;
    struct Node *left, *right;
};

// Structure for Stack
struct Stack
{
    int size;
    int top;
    struct Node* *array;
};

// A utility function to create a new tree node
struct Node* newNode(int data)
{
    struct Node* node = (struct Node*) malloc(sizeof(struct Node));
    node->data = data;
    node->left = node->right = NULL;
```

[Recursion/Algorithms](#)



Popular Posts

[All permutations of a given string](#)

[Memory Layout of C Programs](#)

[Understanding "extern" keyword in C](#)

[Median of two sorted arrays](#)

[Tree traversal without recursion and without stack!](#)

[Structure Member Alignment, Padding and Data Packing](#)

[Intersection point of two Linked Lists](#)

[Lowest Common Ancestor in a BST.](#)

[Check if a binary tree is BST or not](#)

[Sorted Linked List to Balanced BST](#)

```

    return node;
}

// A utility function to create a stack of given size
struct Stack* createStack(int size)
{
    struct Stack* stack = (struct Stack*) malloc(sizeof(struct Stack))
    stack->size = size;
    stack->top = -1;
    stack->array = (struct Node**) malloc(stack->size * sizeof(struct Node))
    return stack;
}

// BASIC OPERATIONS OF STACK
int isFull(struct Stack* stack)
{
    return stack->top - 1 == stack->size;
}
int isEmpty(struct Stack* stack)
{
    return stack->top == -1;
}
void push(struct Stack* stack, struct Node* node)
{
    if (isFull(stack))
        return;
    stack->array[++stack->top] = node;
}
struct Node* pop(struct Stack* stack)
{
    if (isEmpty(stack))
        return NULL;
    return stack->array[stack->top--];
}
struct Node* peek(struct Stack* stack)
{
    if (isEmpty(stack))
        return NULL;
    return stack->array[stack->top];
}

// Iterative Function to print all ancestors of a given key
void printAncestors(struct Node *root, int key)
{
    if (root == NULL) return;

    // Create a stack to hold ancestors

```



```
struct Stack* stack = createStack(MAX_SIZE);
```

```
// Traverse the complete tree in postorder way till we find the key
while (1)
{
    // Traverse the left side. While traversing, push the nodes in
    // the stack so that their right subtrees can be traversed later
    while (root && root->data != key)
    {
        push(stack, root);    // push current node
        root = root->left;    // move to next node
    }

    // If the node whose ancestors are to be printed is found,
    // then break the while loop.
    if (root && root->data == key)
        break;

    // Check if right sub-tree exists for the node at top
    // If not then pop that node because we don't need this
    // node any more.
    if (peek(stack)->right == NULL)
    {
        root = pop(stack);

        // If the popped node is right child of top, then remove it
        // as well. Left child of the top must have processed before
        // Consider the following tree for example and key = 3. If
        // remove the following loop, the program will go in an
        // infinite loop after reaching 5.
        //
        //      1
        //     / \
        //    2   3
        //     \
        //      4
        //     \
        //      5
        while (!isEmpty(stack) && peek(stack)->right == root)
            root = pop(stack);
    }

    // if stack is not empty then simply set the root as right child
    // of top and start traversing right sub-tree.
    root = isEmpty(stack)? NULL: peek(stack)->right;
}

// If stack is not empty, print contents of stack
```

695



Subscribe

Recent Comments

affiszerv Your example has two 4s on row 3, that's why it...

[Backtracking | Set 7 \(Sudoku\)](#) · 29 minutes ago

RVM Can someone please elaborate this Qs from above...

[Flipkart Interview | Set 6](#) · 49 minutes ago

Vishal Gupta I talked about as an Interviewer in general,...

[Software Engineering Lab, Samsung Interview | Set 2](#) · 49 minutes ago

@meya Working solution for question 2 of 4f2f round....

[Amazon Interview | Set 53 \(For SDE-1\)](#) · 1 hour ago

sandeep void rearrange(struct node *head) {...

Given a linked list, reverse alternate nodes and append at the end · 2 hours ago

Neha I think that is what it should return as, in...

[Find depth of the deepest odd level leaf node](#) · 2 hours ago

AdChoices

[► Binary Tree](#)

[► Java to C++](#)

```

// Here assumption is that the key is there in tree
while (!isEmpty(stack))
    printf("%d ", pop(stack)->data);
}

// Driver program to test above functions
int main()
{
    // Let us construct a binary tree
    struct Node* root = newNode(1);
    root->left = newNode(2);
    root->right = newNode(3);
    root->left->left = newNode(4);
    root->left->right = newNode(5);
    root->right->left = newNode(6);
    root->right->right = newNode(7);
    root->left->left->left = newNode(8);
    root->left->right->right = newNode(9);
    root->right->right->left = newNode(10);

    printf("Following are all keys and their ancestors\n");
    for (int key = 1; key <= 10; key++)
    {
        printf("%d: ", key);
        printAncestors(root, key);
        printf("\n");
    }

    getchar();
    return 0;
}

```


Output:

Following are all keys and their ancestors

```

1:
2: 1
3: 1
4: 2 1
5: 2 1
6: 3 1
7: 3 1
8: 4 2 1
9: 5 2 1


```

AdChoices 

[► Java Source Code](#)

[► Ancestors Tree](#)

[► Recursion](#)

AdChoices 

[► Tree Root](#)

[► 3 D Print](#)

[► We Print It](#)

Exercise

Note that the above solution assumes that the given key is present in the given Binary Tree. It may go in infinite loop if key is not present. Extend the above solution to work even when the key is not present in tree.

This article is contributed by **Chandra Prakash**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.



Related Tpoics:

- [Print a Binary Tree in Vertical Order | Set 2 \(Hashmap based Method\)](#)
- [Print Right View of a Binary Tree](#)
- [Red-Black Tree | Set 3 \(Delete\)](#)
- [Construct a tree from Inorder and Level order traversals](#)
- [Print all nodes at distance k from a given node](#)
- [Print a Binary Tree in Vertical Order | Set 1](#)
- [Interval Tree](#)
- [Check if a given Binary Tree is height balanced like a Red-Black Tree](#)



33



Tweet

4



1

Writing code in comment? Please use ideone.com and share the link here.

46 Comments**GeeksforGeeks**

Sort by Newest ▼



Join the discussion...

**gatorboy** · 4 months ago

This can be done in simple way using recursion.

```
PrintAncestors(Node root, int key){  
    if(root==null)  
        return false;  
    if(root.value == key)  
        return true;  
    if(PrintAncestors(root.left, key) || PrintAncestors(root.right, key)){  
        System.out.print(root.value)  
        return true;  
    }  
    return false;  
}
```

2 ^ | v · Reply · Share ›

**p** → **gatorboy** · 3 months ago

? ??

^ | v · Reply · Share ›

**zealfire** · 4 months ago

queue, but not remove top nodes while inserting their child node, then picking parent array by using formula for parent node which is $(i-1)/2$.

^ | v • Reply • Share ›

sriahsrha • 4 months ago

i think this works

1: do inorder traversal without recursion
2: once you find the target node
print the stack, traverse the stack from top to bottom and print only elements until you reach the root.

^ | v • Reply • Share ›

Yukang • 6 months ago

using two stacks, and code is very simple!

```
void PrintAncesters(struct Node* root, int target) {  
  
    if(root == NULL) return;  
  
    stack<struct node*> st;  
  
    stack<struct node*> out;  
  
    st.push(root);  
  
    bool found = false;  
  
    while(!st.empty()) {  
  
        struct Node* cur = st.top();  
  
        out.push(cur);
```


intuit->data == 1000000;

see more

^ | v • Reply • Share ›



sh • 6 months ago

One catch in this code though..if the element is not present in the tree, this co

^ | v • Reply • Share ›



rahul • 6 months ago

we can do it by preorder traversal also . am i right?

^ | v • Reply • Share ›



Suryabhan Singh • 7 months ago

```
void printanst(struct node *s)
{
    struct node *temp=s, *pre=NULL;
    queue<struct node*> q,parent;
    q.push(temp);
    while(!q.empty())
    {
        temp=q.front();
        q.pop();
        if(temp==s)
        {
            cout<<temp->data<<" no parent"<<endl; }="" else="" {="" c
        {
            q.push(temp->l);
            parent.push(temp);
        }
        if(temp->r)
```

[see more](#)

^ | v • Reply • Share ›



pavansrinivas • 7 months ago

Iterative Version using Queue in JAVA...

```
void allAncestors(int key){
    Node temp = root;
    Queue<object> q = new LinkedList<object>();
    String path = root.iData+"";
    q.add(temp);
    q.add(path);
    while(!q.isEmpty()){
        temp = (Node)q.remove();
        path = (String)q.remove();
        if(temp.leftChild!=null){
            String leftstr = path+"-"+temp.leftChild.iData;
            q.add(temp.leftChild);
            q.add(leftstr);
            if(temp.leftChild.iData==key){
                System.out.print(path);
                return;
            }
        }
    }
}
```

[see more](#)

^ | v • Reply • Share ›



grab • 7 months ago

```
while(1)
{
    while(root)
    {
        if(root == element to be found)
```

```

{flag =1;
return /// prnt the content of stack;}
push(root)
root = root ->left
}
if(empty(S)) break;
{
root=s.top();
while(! root->right)
{
pop();
root =top();
}
root=root->right;
}
}
if(flag ==0 )
{cout << element no found;}

```

^ | v • Reply • Share ›



Meraj Ahmed • 7 months ago

The followinig code solves the problem of infinite run and implements STACK:

```

#include <stdio.h>

#include <stdlib.h>

struct node

{

int val;

```

```
struct node *left,  
  
struct node *right;  
  
};
```

```
struct node_stack
```

```
{
```

[see more](#)

1 ^ | v • Reply • Share ›



din • 7 months ago

i still didnt get the algorithm ..pls help me...tnx....

^ | v • Reply • Share ›



atiq • 8 months ago

//It's same like Non recursive Postorder except key matching case
see my simple implementation

```
void NRPrintAncestors(Tree *Root,int key)  
{  
if(!Root)return;  
Stack *S=Create_Stack();  
Tree *temp=NULL;  
while(1)  
{  
while(Root->data!=key&&Root->left)  
{  
Push(&S,Root)  
Root=Root->left;  
}  
if(Root->data!=key&&Root->right)
```

```
Push(&S,Root)
Root=Root->data;
```

[see more](#)

^ | v • Reply • Share ›



atiq • 8 months ago

```
/* Paste your code here (You may delete these lines if not writing c)
void NRPrintAncestors(Tree *Root,int key)
{
    if(!Root)return;
    Stack *S=Create_Stack();
    Tree *temp=NULL;
    while(1)
    {
        while(Root->data!=key&&Root->left)
        {
            Push(&S, Root);
            Root=Root->left;
        }
        if(Root->data!=key&&Root->right)
        {
            Push(&S, Root);
            Root=Root->data;
        }
    }
}
```

[see more](#)

1 ^ | v • Reply • Share ›



Ayush Jain • 8 months ago

java code without recursion.

```
public List<Integer> ancestorOf(BinaryTreeNode btn, int node){  
List<Integer> ancestor = new ArrayList<Integer>();
```

```
while(btn!=null){  
if(btn.data==node && ancestor.size()==0){  
System.out.println("No ancestor of node in Binary Tree");  
break;  
}  
else if(node>btn.data && btn.rchild!=null){  
ancestor.add(btn.data);  
btn=btn.rchild;  
}  
else if(node<btn.data && btn.lchild!=null){  
ancestor.add(btn.data);  
btn=btn.lchild;  
}  
else if (node!=btn.data){  
System.out.println("Node does not exists in Binary Tree");  
ancestor=null;  
break;  
}  
else{  
break;  
}  
}  
return ancestor;  
}
```

^ | v • Reply • Share ›



Amit Bgl • 9 months ago

wow code :D

^ | v • Reply • Share ›



Tarun Kumar • 9 months ago

--writing sudo code.

vector<int> keys;

```

if(root->m_key == key){
{
std::cout<<key<<" ";
}
else
{
keys.push_back(root->m_key);
}

while(root->m_key!= key || root!= NULL).
{
if(root == NULL).
{.
cerr<<"No key/s found"<<std::endl;.
return;.
}.
}

```

[see more](#)

^ | v • Reply • Share ›



Rudra • 9 months ago

Implementation follo same logic described, only handel the condition when the

```

void TreeOperation::postOrderTraversalRecursive(TreeNode* root, int ele)
{
    TreeNodeStack stack(100);
    TreeNode* node = root;
    while(1){
        while(node && (node->data != ele)){
            stack.push(node);
            node = node->getLeftChild();
        }
        if(node != NULL && (node->data == ele)){
            break;
        }
    }
}

```

```
}
```

```
if(stack.peek()->rightChild == NULL){  
    node = stack.pop();  
    while( !stack.isEmpty() && (stack.peek()->rightChild != NULL)){  
        node = stack.pop();  
    }  
}
```

[see more](#)

^ | v • Reply • Share ›



Rudra • 9 months ago

The implementation follows the logic as described only handling the condition for a leaf node.

```
[sourcecode language="C++"]  
void TreeOperation::postOrderTraversalRecursive(TreeNode* root, int ele){  
    TreeNodeStack stack(100);  
    TreeNode* node = root;  
    while(1){  
        while(node && (node->data != ele)){  
            stack.push(node);  
            node = node->getLeftChild();  
        }  
        if(node != NULL && (node->data == ele)){  
            break;  
        }  
        if(stack.peek()->rightChild == NULL){  
            node = stack.pop();  
            while( !stack.isEmpty() && (stack.peek()->rightChild != NULL)){  
                node = stack.pop();  
            }  
        }  
    }  
}
```

[see more](#)

• Reply • Share ›



dark_night • 9 months ago

the given method is not working for the following case(key 3) :

```
1
2 3
4
5
6
root=1
root->left=2
root->right=3
root->left->left=4
root->left->left->right=5
root->left->left->right->right=6
```

^ | v • Reply • Share ›



Nidhi Gupta • 9 months ago

the code is not working for all the cases...consider this one.

```
struct Node* root = newNode(1);
```

```
root->left = newNode(2);
```

```
root->right = newNode(3);
```

```
root->left->left = newNode(4);
```

```
root->right->left = newNode(6);
```

```
root->right->right = newNode(7);
```

```
root->left->left->left = newNode(8);
```

```
root->left->right->right = newNode(9);
```

^ | v • Reply • Share ›



ubiquitous • 9 months ago

it can achieved iterative postorder traversal and when u find the expected node postorder traversal see the [http://en.wikipedia.org/wiki/T....](http://en.wikipedia.org/wiki/T...) If further assistance Thanks.

^ | v • Reply • Share ›



trying • 9 months ago

do a simple DFS and once you find the expected number just print the stack c

^ | v • Reply • Share ›



12rad • 9 months ago

It isn't a Binary Search tree... it's just a plain binary tree. So, you can't compar

^ | v • Reply • Share ›



sudhansu • 9 months ago

Here is my java code..

```
private boolean printAncestorsIteeratively(TreeNode root,int target)
    if(root==null)
        return false;
    TreeNode t;
    ArrayStack stack=new ArrayStack(20);
    while(root!=null && root.data!=target){
        if(target>=root.data){
            stack.push(root);
            root=root.right;
        }
        if(target<=root.data){
            stack.push(root);
            root=root.left;
        }
    }
}
```

```

        }
        if(root.data==target)
            break;
    }
    t=(TreeNode) stack.pop();
    while(t!=null){
        System.out.print(t.data);
        t=(TreeNode) stack.pop();
    }
    return true;
}

```

^ | v • Reply • Share ›



12rad → sudhansu • 9 months ago

Isn't a Binary Search tree.. it's just a plain binary tree. So, you can't cor

^ | v • Reply • Share ›



12rad → sudhansu • 9 months ago

It isn't a Binary Search tree... it's just a plain binary tree. So, you can't c

/* Paste your code here (You may **delete** these lines **if not** wri

^ | v • Reply • Share ›



Akshay Kumar • 9 months ago

It can also done by simple modification in iterative Inorder traversal :

- 1) Create an empty stack S.
- 2) Initialize current node as root.
- 3) Push the current node to S and set current = current->left until current is NULL
- 4) If current is NULL and stack is not empty then.

If stack top is equal to key, print the elements of stack.

a) Pop the top item from stack.

b) Print the popped item, set current = current->right.

c) Go to step 3.

2 ^ | v • Reply • Share ›



Akshay Jindal • 9 months ago

The following is a simpler implementation of the question. What we need to do then traverse till the root. Suppose we have x as the node whose ancestors are

```
y=x->parent;
while(y!=NULL)
{
    printf("%d ", y->data);
    x=y;
    y=y->parent;
}
```

^ | v • Reply • Share ›



vdraceil • 9 months ago

```
if (root && root->data == key)
    break;
```

The above can be replaced with

```
if (root) break;
```

^ | v • Reply • Share ›



Kumar Vikram · 10 months ago

The following is a simpler implementation of above problem using c++.

[sourcecode language="C++"]

```
#include<iostream>
```

```
#include<stack>
```

```
#include<stdlib.h>
```

```
using namespace std;
```

```
struct node
```

```
{
```

```
int data;
```

```
struct node *left,*right;
```

```
};
```

```
node * newNode(int num)
```

```
{
```

```
node *temp=(node *)malloc(sizeof(node));
```

```
temp->data=num;
```

```
temp->left=temp->right=NULL;
```

[see more](#)

^ | v · [Reply](#) · [Share](#) ›



GeeksforGeeks · 10 months ago

Please note that the problem is for Binary Tree, not Binary Search Tree.

^ | v · [Reply](#) · [Share](#) ›



Manisha Barnwal · 10 months ago

//It can be done in a much more simpler way..

/*Just print the ancestors while traversing the tree*/.

/*Assumption the key is present in the tree*/.

```
void print_ancestors(int key, node *root).
{

if(root==NULL).

return;.

if(root->data==key).

return;.

node *ptr;.

ptr=root;.

printf("nAncestors are :");.
```

[see more](#)

^ | v • Reply • Share ›



sijayaraman • 10 months ago

using stack

```
void ancestor(struct node* root, int key)
{
    stack<int> stack1;
    if(root->data == key)
    {
        cout<<"This is Root Node, No Ansces"<<endl; return;="" }="" while(1)="" {="" i
        NULL))
    {
        stack1.push(root->data);
        root = root->left;
```

```
-  
else if(root->data < key && (root->right != NULL))  
{  
    stack1.push(root->data);  
    root = root->right;  
}
```

[see more](#)

^ | v • Reply • Share ›



minoz • 10 months ago

1. Push the nodes in a stack while doing level order traversal until we get the s
2. Now check if the given node is left or right child of the top of stack. If yes, pri
top to the given node.
3. Pop the stack.
4. Repeat steps 2 & 3 until stack is empty.

Complexity: $O(n)$

Please comment if the approach has errors.

```
/* Paste your code here (You may delete these lines if not writing c
```

^ | v • Reply • Share ›



minoz → minoz • 10 months ago

In step 2, I meant, assign the node at the top to the given node.

^ | v • Reply • Share ›



minoz → minoz • 10 months ago

Got it. $O(n)$ extra space.

^ | v • Reply • Share ›



Geek86 • 10 months ago

Kindly validate my recursive solution.

[sourcecode language="java"]

```
import java.util.Stack;
```

```
public class BTUtil {
```

```
public static void getAncestors(BTNode root, int key, Stack<BTNode> nodes)
```

```
if ( isPresentInSubTree ( root.getLeft (), key, nodes )
```

```
|| isPresentInSubTree ( root.getRight (), key, nodes ) ) {
```

```
nodes.push ( root );
```

```
}
```

```
else {
```

```
if ( ! nodes.isEmpty () ) {
```

```
nodes.pop ();
```

```
}
```

```
}
```

[see more](#)

^ | v • Reply • Share ›



idexter → Geek86 • 10 months ago

You are calling isPresentInSubTree recursively and the first call to that
This employs recursive at some part in the solution. This won't help in
recursive elimination.

^ | v • Reply • Share ›



Geek86 → idexter • 10 months ago

Thank You for your reply. But one advantage I see here is, it eliminates
deepest ones.

^ | v · Reply · Share ›



Silent · 10 months ago

what is the problem in this code??

```
void printAncestors(struct tree *root,int x)
{
    struct stack st;
    st.top=-1;

    while(1)
    {
        while(root && root->info!=x)
        {
            push(&st,root);
            root=root->left;
        }

        if(root && root->info==x)
            break;

        root=topElement(&st);
```

see more

^ | v · Reply · Share ›



Sambasiva · 10 months ago

<http://effprog.wordpress.com/2013/07/01/print-ancestors-of-a-given-binary-tree/>

^ | v · Reply · Share ›



SELVAMANI · 10 months ago

The answer is very simple. **Use** a post-order iterative traversal using **while** pushing **or** popping the element from the stack verify whether the

... so, all the elements that are in the stack are its **parent**.

^ | v • Reply • Share ›



AMIT • 10 months ago

For exercise part

We should keep track of the root

Struct Node *root1=root;

inside the for loop whenever you find root on top of the stack you must break

Outside the loop check if required node is in the top?if not print error

^ | v • Reply • Share ›



AMIT → AMIT • 10 months ago

I mean root is on the top of the stack and its right is already visited

^ | v • Reply • Share ›



bateesh → AMIT • 10 months ago

Instead of using extra variable we can check for stack empty, as stack is empty means no ancestors for current node. It must be

^ | v • Reply • Share ›

 [Subscribe](#)

 [Add Disqus to your site](#)

@geeksforgeeks, **Some rights reserved**

Contact Us!

Powered by **WordPress** & **MooTools**, customized by geeksforgeeks team