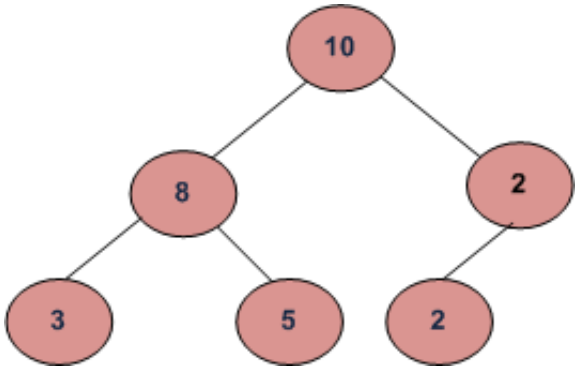
Given a binary tree, print all root-to-leaf paths

For the below example tree, all root-to-leaf paths are:

10 –> 8 –> 3

10 –> 8 –> 5

10 –> 2 –> 2



Algorithm:

Use a path array path[] to store current root to leaf path. Traverse from root to all leaves in top-down fashion. While traversing, store data of all nodes in current path in array path[]. When we reach a leaf node, print the path array.

C

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

/* A binary tree node has data, pointer to left child
   and a pointer to right child */
struct node
{
   int data;
   struct node* left;
   struct node* right;
};

/* Prototypes for funtions needed in printPaths() */
void printPathsRecur(struct node* node, int path[], int pathLen);
```

```
void printArray(int ints[], int len);

/*Given a binary tree, print out all of its root-to-leaf
 paths, one per line. Uses a recursive helper to do the work.*/
void printPaths(struct node* node)
{
  int path[1000];
  printPathsRecur(node, path, 0);
}

/* Recursive helper function -- given a node, and an array containing
 the path from the root node up to but not including this node,
 print out all the root-leaf paths.*/
void printPathsRecur(struct node* node, int path[], int pathLen)
{
  if (node==NULL)
    return;

  /* append this node to the path array */
  path[pathLen] = node->data;
  pathLen++;

  /* it's a leaf, so print the path that led to here  */
  if (node->left==NULL && node->right==NULL)
  {
    printArray(path, pathLen);
  }
  else
  {
    /* otherwise try both subtrees */
    printPathsRecur(node->left, path, pathLen);
    printPathsRecur(node->right, path, pathLen);
  }
}


/* UTILITY FUNCTIONS */
/* Utility that prints out an array on a line. */
void printArray(int ints[], int len)
{
  int i;
  for (i=0; i<len; i++)
  {
    printf("%d ", ints[i]);
  }
  printf("\n");
}

/* utility that allocates a new node with the
   given data and NULL left and right pointers. */
struct node* newnode(int data)
{
  struct node* node = (struct node*)
                      malloc(sizeof(struct node));
  node->data = data;
  node->left = NULL;
  node->right = NULL;

  return(node);
}

/* Driver program to test above functions*/
int main()
{
```

```
  /* Constructed binary tree is
            10
          /   \
        8      2
      /  \    /
    3     5  2
  */
  struct node *root = newnode(10);
  root->left        = newnode(8);
  root->right       = newnode(2);
  root->left->left  = newnode(3);
  root->left->right = newnode(5);
  root->right->left = newnode(2);

  printPaths(root);

  getchar();
  return 0;
}
```

Run on IDE

Java

```
// Java program to print all the node to leaf path

// A binary tree node
class Node {

    int data;
    Node left, right;

    Node(int item) {
        data = item;
        left = right = null;
    }
}

class BinaryTree {

    static Node root;

    /*Given a binary tree, print out all of its root-to-leaf
     paths, one per line. Uses a recursive helper to do the work.*/
    void printPaths(Node node) {
        int path[] = new int[1000];
        printPathsRecur(node, path, 0);
    }

    /* Recursive helper function -- given a node, and an array containing
     the path from the root node up to but not including this node,
     print out all the root-leaf paths.*/
    void printPathsRecur(Node node, int path[], int pathLen) {
        if (node == null) {
            return;
        }

        /* append this node to the path array */
        path[pathLen] = node.data;
        pathLen++;

        /* it's a leaf, so print the path that led to here  */
```

```
        if (node.left == null && node.right == null) {
            printArray(path, pathLen);
        } else {

            /* otherwise try both subtrees */
            printPathsRecur(node.left, path, pathLen);
            printPathsRecur(node.right, path, pathLen);
        }
    }

    /* Utility function that prints out an array on a line. */
    void printArray(int ints[], int len) {
        int i;
        for (i = 0; i < len; i++) {
            System.out.print(ints[i] + " ");
        }
        System.out.println("");
    }

    // driver program to test above functions
    public static void main(String args[]) {
        BinaryTree tree = new BinaryTree();
        tree.root = new Node(10);
        tree.root.left = new Node(8);
        tree.root.right = new Node(2);
        tree.root.left.left = new Node(3);
        tree.root.left.right = new Node(5);
        tree.root.right.left = new Node(2);
        tree.printPaths(root);
    }
}
```

Run on IDE

Time Complexity: O(n)

References:

http://cslibrary.stanford.edu/110/BinaryTrees.html

Please write comments if you find any bug in above codes/algorithms, or find other ways to solve the same problem.

68 Comments Category: Trees

Related Posts:

- Check if removing an edge can divide a Binary Tree in two halves
- Check sum of Covered and Uncovered nodes of Binary Tree
- Lowest Common Ancestor in a Binary Tree | Set 2 (Using Parent Pointer)
- Construct a Binary Search Tree from given postorder
- BFS vs DFS for Binary Tree
- Maximum difference between node and its ancestor in Binary Tree
- Inorder Non-threaded Binary Tree Traversal without Recursion or Stack
- Check if leaf traversal of two Binary Trees is same?

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

Stinger • 18 days ago

I got this question at my facebook interview

• Reply • Share ›

ravi ➚ Stinger • 18 days ago

How did you get a call from facebook

• Reply • Share ›

rockeblaze • 3 months ago

how is this complexity O(n)?

• Reply • Share ›

Shambhavi Shinde ➚ rockeblaze • 2 months ago

'cos the function printPathRecur is being called once only for every node.

• Reply • Share ›

Vinay Kairam • 5 months ago

is it possible without recursion ?

• Reply • Share ›

Tarun Grover ➚ Vinay Kairam • 4 months ago

Non Recursive way to solve this problem

```
public static void rootToLeafPathWithOutRecursion(Node p, int []path) {

StackIml<node> s=new StackIml<node>();

int length=0;

while(p!=null){

p.flag=0;

s.push(p);

path[length++]=p.data;

p=p.left;

}

while(ls.isEmpty()){
```

see more

• Reply • Share ›

nikhilnvj • 6 months ago

Simplest Solution in Python. Just 8 lines of code.

http://ideone.com/AMYk1U

• Reply • Share ›

Flower girl • 7 months ago

after printArray where the control goes??i mean from where the root path starts again in the prgm..

• Reply • Share ›

sunandan ➚ Flower girl • 2 months ago

Imagine it this way. printArray is only called when the node is the leaf node. So the function stack for leaf node need not proceed further. Thus, the current function stack (leaf) is removed, and control goes back to the parent's function stack. Actually, this is the basis of recursion. Parent -> Child . When child is over, control goes back to parent.

• Reply • Share ›

kolo • 9 months ago

```
void printPathsRecur(struct node* node, int path[], int pathLen)
{
if (node==NULL)
return;

/* append this node to the path array */
path[pathLen] = node->data;
pathLen++;

/* it's a leaf, so print the path that led to here */
if (node->left==NULL && node->right==NULL)
{
printArray(path, pathLen);
}
else
{
/* otherwise try both subtrees */
printPathsRecur(node->left, path, pathLen);
```

see more

• Reply • Share ›

Bewkoof_coder ➚ kolo • 9 months ago

that is for the single child nodes,at that time (node==NULL) will play role in returning,else if its a leaf node it will be printed.

• Reply • Share ›

kolo ➚ Bewkoof_coder • 9 months ago

thank u sir

• Reply • Share ›

Vineeth Reddy • a year ago

Some steps author needs to optimise(syntactical steps):
/* append this node to the path array */
path[pathLen] = node->data;
pathLen++;

path[pathLen++] = node->data; //use this instead

And we can skip else:
else
{
/* otherwise try both subtrees */
printPathsRecur(node->left, path, pathLen);
printPathsRecur(node->right, path, pathLen);
}

printPathsRecur(node->left, path, pathLen);
printPathsRecur(node->right, path, pathLen);

is sufficient.

• Reply • Share ›

watch my replay ➚ Vineeth Reddy • 7 months ago

These are actually some horrible advice. Maybe we should name printPathsRecur to pPR to "optimise syntactical steps"?

1. Both ways are correct, but the original code is more clear.

2. You would add two more functions calls, and two comparison with null for each leaf node every time when you could save all those operations by just adding "else","{", "}".

• Reply • Share ›

thevagabond85 • a year ago

Possible duplicate http://www.geeksforgeeks.org/g...

• Reply • Share ›

thevagabond85 • a year ago

Duplicate post . http://www.geeksforgeeks.org/g...

• Reply • Share ›

Aashish • a year ago
Time complexity is O(n^2)...
since no. of leaves can be upto O(n).. and for each leaf, we are taking O(n) time for printing.
3 • Reply • Share ›

sachin soni • a year ago
@geeksforgeek,
array size must be equal to or less than the height of tree,
int height= height(root);
int array[height]={0}; // always initialize contains
• Reply • Share ›

Tapan Anand • 2 years ago
Shouldn't the complexity of this code be O(nlogn). Cosider the case of a complete binary tree.
1 • Reply • Share ›

Musaddique Hossain ➦ Tapan Anand • 2 years ago
I also have the same doubt. Recursion will run for Log(n) times. And there is a loop to print the array. So it should be O(nLogn).
• Reply • Share ›

help ➦ Musaddique Hossain • a year ago
remove lower order terms for complexity

O(n) + O(logn) = O(n)
• Reply • Share ›

Gaurav Nara • 2 years ago
@GeeksforGeeks i don't why is your code so big?? you just need to use a string and that's it..
check this out .. http://ideone.com/V9xoJ3
1 • Reply • Share ›

vpr • 2 years ago
Hi GeeksforGeeks, please comment on following Pseudo Code for non recursive version i came up with.

```
void Path(struct node *root)
{
if(root==NULL) return;
S=Create-stack();
```

```
Push(S,root);
while(!IsEmpty(S))
{
while(root!=NULL)
{ root=root->left;
if(root) Push(S,root);
}
root=top(S);
root=root->right;
if(root==NULL) //a leaf is encountered
{ Print the stack in reverse order;
Pop(S);
root=top(S);
}
Push(S,root);
}
}
```

• Reply • Share ›

Mahda • 2 years ago

Hi geeks, what must I do if I want to add the averages from each path?

• Reply • Share ›

prashant jha • 2 years ago

use recursion and vector for path here is my code
http://ideone.com/idIpnx

• Reply • Share ›

Sabitaa Bhabhi • 2 years ago

why we are not using dynamic array

```
void printPaths(struct node* node)
{
int path[]={0};
printPathsRecur(node, path, 0);
}
```

• Reply • Share ›

Babajj • 2 years ago

Just use a stack having a print function.

-Push current node is not null
-Print stack if its a leaf node
-Recurse left

-Recurse right
-Pop stack
1 • Reply • Share ›

Aashutosh Swarnakar ➚ Babajj • 6 months ago
Thanks alot man.. excellent recursion..
• Reply • Share ›

shravan • 2 years ago
Each of the path[] array may contain O(h) elements, where h is the height of the tree. so process these elements only the algorithm takes O(n*h) time.
how we can say the algorithm runs in O(n)?
:-(
• Reply • Share ›

Jonathan ➚ shravan • 2 years ago
Each node is traversed only once throughout the tree.

O(n*h) would occur only when nodes traversed multiple times.
• Reply • Share ›

Vivek • 2 years ago
hi geeksforgeeks .. why waste so much space?
we know that the length of any path wont be greater than the height of the tree. so we can initialize the array "path" with length = height of tree instead of any predetermined value.
here's the optimized solution below
8 • Reply • Share ›

Klaus ➚ Vivek • 2 years ago
Nice Suggestion
• Reply • Share ›

Vivek • 2 years ago

```
#include <stdio.h>

#include <stdlib.h>

struct node

{
```

```
        int data;

        struct node *left;

        struct node *right;

};

void root_leaf(struct node *root, int *arr, int e)
```

see more

3 • Reply • Share ›

govind ➦ Vivek • 2 years ago

Vivek, in the root_leaf function, you have not written
"if(root==NULL) return;"
how is just "if(root){..}" enough? I guess else return must be present?
please explain.

• Reply • Share ›

kumar • 2 years ago

Same question has been asked on FB's interview, with constraint that use iterative rather recursive.

• Reply • Share ›

Rahul Mahale • 3 years ago

void PrintArr(int *arr,int len)
{
static int pathNo=0;
int i;

printf("\nPath %d",++pathNo);

for(i=0;i<len;i++) {="" printf("="" %d="" ",arr[i]);="" }="" return;="" }="" void="" printr2lpaths(bst="" *root,int="" patharr[],int="" pathlen)="" {="" if(root="=NULL)" return;="" patharr[pathlen]="root-">data;
pathLen++;

if(root->left==NULL && root->right==NULL)
{
PrintArr(pathArr,pathLen);
return;
}
else
{

```
PrintR2LPaths(root->left,pathArr,pathLen);
PrintR2LPaths(root->right,pathArr,pathLen);
}
}
```

• Reply • Share ›

Rahul Mahale • 3 years ago

```
void PrintArr(int *arr,int len)
{
        static int pathNo=0;
        int i;

        printf("\nPath %d",++pathNo);

        for(i=0;i<len;i++)
        {
                printf(" %d ",arr[i]);
        }

        return;
}

void PrintR2LPaths(BST *root,int pathArr[],int pathLen)
{
```

see more

1 • Reply • Share ›

shek8034 • 3 years ago

Simple and working code

```
 /* void printPaths(node* root, int len)
{
  static int path[1000];
  if(root == NULL)
        return;
  path[len++] = root->data;
  if (root->left==NULL && root->right==NULL)
  {
    int i;
        for(i=0;i<len;i++)
                printf("%d ",path[i]);
        printf("\n");
```

```
  }
  else
  {
    /* otherwise try both subtrees */
    printPaths(root->left,len);
    printPaths(root->right,len);
  }
} */
```

• Reply • Share ›

Nikhil Agrawal • 3 years ago

Iterative version for printing all possible paths from root to leafs:

```
 public static void printAllPathToLeafNonRecursive(Node root)
{
    if (root == null)
    {
        return;
    }

    Queue<Object> q = new LinkedList<Object>();
    q.add(root);
    q.add(root.value + "");

    while(!q.isEmpty()){

        Node head = (Node) q.remove();
        String headPath = (String) q.remove();
```

see more

3 • Reply • Share ›

Raghu ➚ Nikhil Agrawal • a year ago

excellent non-iterative solution Nikhil!

• Reply • Share ›

pavansrinivas ➚ Nikhil Agrawal • 2 years ago

nyc soln...

• Reply • Share ›

yo-gi • 4 years ago

void printAllPaths(struct node* root, int *arr, int len)

{
if(root == NULL)
return;

arr[len++] = root->data;

if((root->left == NULL) && (root->right==NULL))
{
int i=0;
for(i=0; i left, arr, len);
printAllPaths(root->right, arr, len);
}

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

• Reply • Share ›

hemant • 4 years ago

Correct me if I am wrong.

Assume a BST and insert 10,5,18,15,12
Now consider it to be a BT and run the logic given in the original post.
I guess it will give a crash because it will try to access the right of node with data 15 which is NULL
Hence a slight modification required.

Guys, please share your comments

int Roofleaf[6]= {0};// instead of 6 we can take height of the tree

void print_array(int size)
{
int i;
if(size == -1)
return;

for(i=0; idata;

see more

• Reply • Share ›

hemant ➚ hemant • 4 years ago

Not again.....I wonder why i always end up pasting the half code though i paste it completely...

anyways my point was we should add conditions like

if(root->left)
printf_root_to_leaf(root->left,i+1)

if(root->right)
printf_root_to_leaf(root->right,i+1)

• Reply • Share ›

GeeksforGeeks hemant • 4 years ago

@hemant: Please paste your code between sourcecode tags. Replace the following lines with your code and keep the sourcecode tags.

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

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

• Reply • Share ›

hemant GeeksforGeeks • 4 years ago

```
 void print_array(int size)
{
    int i;
        for(i=0; i<=size; i++)
     printf("%d ",Roofleaf[i]);

     printf("\n");
}

void printf_root_to_leaf(struct node* root, int i)
{
   Roofleaf[i]= root->data;

    if((root->left == NULL) && (root->right == NULL))
    {
      print_array(i);
      return;
    }
```

see more

• Reply • Share ›

hemant hemant • 4 years ago

I wonder why i always end up pasting the half code though i paste it completely...

anyways my point was we should add conditions like

anyways my point was we should add conditions like

if(root->left)
printf_root_to_leaf(root->left,i+1)

if(root->right)
printf_root_to_leaf(root->right,i+1)

• Reply • Share ›

hemant • 4 years ago

Assume a BST and insert 10,5,18,15,12
Now consider it to be a BT and run the logic given in the original post.
I guess it will give a crash because it will try to access the right of node with data 15 which is NULL
Hence a slight modification required.

Guys, please share your comments

int Roofleaf[6]= {0};// instead of 6 we can take height of the tree

void print_array(int size)
{
int i;
if(size == -1)
return;

for(i=0; idata;

if((root->left == NULL) && (root->right == NULL))

see more

• Reply • Share ›

krishna • 4 years ago

Instead of fixing the path size to const, its better if we take height of the tree as the length of the array.

1 • Reply • Share ›

FireFox • 4 years ago

```
void printAllpaths(Node root, int[] a, int i){
       If(node == null)
               return;
      A[i] = node.data;
      printAllpaths(root.left, a,i+1);
      printAllpaths(root.right,a,i+1);
      printTheArray(a,i);
```

```
}
```

• Reply • Share ›

Load more comments