Connect nodes at same level using constant extra space

Write a function to connect all the adjacent nodes at the same level in a binary tree. Structure of the given Binary Tree node is like following.

```
struct node {
  int data;
  struct node* left;
  struct node* right;
  struct node* nextRight;
}
```

Run on IDE

Initially, all the nextRight pointers point to garbage values. Your function should set these pointers to point next right for each node. You can use only constant extra space.

Example

```
Input Tree
       A
      / \
     B   C
    / \   \
   D   E   F

Output Tree
       A--->NULL
      / \
     B-->C-->NULL
    / \   \
   D-->E-->F-->NULL
```

We discussed two different approaches to do it in the previous post. The auxiliary space required in both of those approaches is not constant. Also, the method 2 discussed there only works for complete Binary Tree.

In this post, we will first modify the method 2 to make it work for all kind of trees. After that, we will remove recursion from this method so that the extra space becomes constant.

A Recursive Solution

In the method 2 of previous post, we traversed the nodes in pre order fashion. Instead of traversing in Pre Order fashion (root, left, right), if we traverse the nextRight node before the left and right children (root, nextRight, left), then we can make sure that all nodes at level i have the nextRight set, before the level i+1 nodes. Let us consider the following example (same example as previous post). The method 2 fails for right child of node 4. In this method, we make sure that all nodes at the 4's level (level 2) have nextRight set, before we try to set the nextRight of 9. So when we set the nextRight of 9, we search for a nonleaf node on right side of node 4 (getNextRight() does this for us).

```
            1            -------------- Level 0
          /    \
        2        3       -------------- Level 1
       / \      /  \
      4   5    6    7    -------------- Level 2
     / \           / \
    8   9        10   11 -------------- Level 3
```

C

```c
void connectRecur(struct node* p);
struct node *getNextRight(struct node *p);

// Sets the nextRight of root and calls connectRecur() for other nodes
void connect (struct node *p)
{
    // Set the nextRight for root
    p->nextRight = NULL;

    // Set the next right for rest of the nodes (other than root)
    connectRecur(p);
}

/* Set next right of all descendents of p. This function makes sure that
nextRight of nodes ar level i is set before level i+1 nodes. */
void connectRecur(struct node* p)
{
    // Base case
    if (!p)
       return;

    /* Before setting nextRight of left and right children, set nextRight
    of children of other nodes at same level (because we can access
    children of other nodes using p's nextRight only) */
    if (p->nextRight != NULL)
       connectRecur(p->nextRight);

    /* Set the nextRight pointer for p's left child */
    if (p->left)
    {
       if (p->right)
       {
            p->left->nextRight = p->right;
            p->right->nextRight = getNextRight(p);
        }
        else
            p->left->nextRight = getNextRight(p);

        /* Recursively call for next level nodes.  Note that we call only
        for left child. The call for left child will call for right child */
        connectRecur(p->left);
    }

    /* If left child is NULL then first node of next level will either be
      p->right or getNextRight(p) */
    else if (p->right)
    {
        p->right->nextRight = getNextRight(p);
        connectRecur(p->right);
    }
    else
       connectRecur(getNextRight(p));
}

/* This function returns the leftmost child of nodes at the same level as p.
   This function is used to getNExt right of p's right child
   If right child of p is NULL then this can also be used for the left child */
struct node *getNextRight(struct node *p)
{
    struct node *temp = p->nextRight;

    /* Traverse nodes at p's level and find and return
       the first node's first child */
    while(temp != NULL)
    {
        if(temp->left != NULL)
            return temp->left;
        if(temp->right != NULL)
            return temp->right;
        temp = temp->nextRight;
    }

    // If all the nodes at p's level are leaf nodes then return NULL
    return NULL;
}
```

Run on IDE

Java

```
// Recursive Java program to connect nodes at same level
// using constant extra space

// A binary tree node
class Node {

    int data;
    Node left, right, nextRight;

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

```
class BinaryTree {

    static Node root;

    /* Set next right of all descendents of p. This function makes sure that
     nextRight of nodes ar level i is set before level i+1 nodes. */
    void connectRecur(Node p) {

        // Base case
        if (p == null) {
            return;
        }

        /* Before setting nextRight of left and right children, set nextRight
         of children of other nodes at same level (because we can access
         children of other nodes using p's nextRight only) */
        if (p.nextRight != null) {
            connectRecur(p.nextRight);
        }

        /* Set the nextRight pointer for p's left child */
        if (p.left != null) {
            if (p.right != null) {
                p.left.nextRight = p.right;
                p.right.nextRight = getNextRight(p);
            } else {
                p.left.nextRight = getNextRight(p);
            }

            /* Recursively call for next level nodes.  Note that we call only
             for left child. The call for left child will call for right child */
            connectRecur(p.left);
        }

        /* If left child is NULL then first node of next level will either be
         p->right or getNextRight(p) */
        else if (p.right != null) {
            p.right.nextRight = getNextRight(p);
            connectRecur(p.right);
        } else {
            connectRecur(getNextRight(p));
        }
    }

    /* This function returns the leftmost child of nodes at the same level as p.
     This function is used to getNExt right of p's right child
     If right child of p is NULL then this can also be used for the left child */
    Node getNextRight(Node p) {
        Node temp = p.nextRight;

        /* Traverse nodes at p's level and find and return
         the first node's first child */
        while (temp != null) {
            if (temp.left != null) {
                return temp.left;
            }
            if (temp.right != null) {
                return temp.right;
            }
            temp = temp.nextRight;
        }

        // If all the nodes at p's level are leaf nodes then return NULL
        return null;
    }

    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.right.right = new Node(90);

        // Populates nextRight pointer in all nodes
        tree.connectRecur(root);

        // Let us check the values of nextRight pointers
        int a = root.nextRight != null ? root.nextRight.data : -1;
        int b = root.left.nextRight != null ? root.left.nextRight.data : -1;
        int c = root.right.nextRight != null ? root.right.nextRight.data : -1;
        int d = root.left.left.nextRight != null ? root.left.left.nextRight.data : -1;
        int e = root.right.right.nextRight != null ? root.right.right.nextRight.data : -1;

        // Now lets print the values
        System.out.println("Following are populated nextRight pointers in "
                + " the tree(-1 is printed if there is no nextRight)");
        System.out.println("nextRight of " + root.data + " is " + a);
        System.out.println("nextRight of " + root.left.data + " is " + b);
        System.out.println("nextRight of " + root.right.data + " is " + c);
        System.out.println("nextRight of " + root.left.left.data + " is " + d);
        System.out.println("nextRight of " + root.right.right.data + " is " + e);
    }
}
```

Run on IDE

An Iterative Solution

The recursive approach discussed above can be easily converted to iterative. In the iterative version, we use nested loop. The outer loop, goes through all the levels and the inner loop goes through all the nodes at every level. This solution uses constant space.

C

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

struct node
{
    int data;
    struct node *left;
    struct node *right;
    struct node *nextRight;
};

/* This function returns the leftmost child of nodes at the same level as p.
   This function is used to getNExt right of p's right child
   If right child of is NULL then this can also be sued for the left child */
```

```
struct node *getNextRight(struct node *p)
{
    struct node *temp = p->nextRight;

    /* Traverse nodes at p's level and find and return
       the first node's first child */
    while (temp != NULL)
    {
        if (temp->left != NULL)
            return temp->left;
        if (temp->right != NULL)
            return temp->right;
        temp = temp->nextRight;
    }

    // If all the nodes at p's level are leaf nodes then return NULL
    return NULL;
}

/* Sets nextRight of all nodes of a tree with root as p */
void connect(struct node* p)
{
    struct node *temp;

    if (!p)
      return;

    // Set nextRight for root
    p->nextRight = NULL;

    // set nextRight of all levels one by one
    while (p != NULL)
    {
        struct node *q = p;

        /* Connect all childrem nodes of p and children nodes of all other nodes
          at same level as p */
        while (q != NULL)
        {
            // Set the nextRight pointer for p's left child
            if (q->left)
            {
                // If q has right child, then right child is nextRight of
                // p and we also need to set nextRight of right child
                if (q->right)
                    q->left->nextRight = q->right;
                else
                    q->left->nextRight = getNextRight(q);
            }

            if (q->right)
                q->right->nextRight = getNextRight(q);

            // Set nextRight for other nodes in pre order fashion
            q = q->nextRight;
        }

        // start from the first node of next level
        if (p->left)
           p = p->left;
        else if (p->right)
           p = p->right;
        else
           p = getNextRight(p);
    }
}

/* UTILITY FUNCTIONS */
/* Helper function 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;
    node->nextRight = NULL;

    return(node);
}

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

    /* Constructed binary tree is
              10
            /   \
          8      2
        /         \
      3            90
    */
    struct node *root = newnode(10);
    root->left        = newnode(8);
    root->right       = newnode(2);
    root->left->left  = newnode(3);
    root->right->right       = newnode(90);

    // Populates nextRight pointer in all nodes
    connect(root);

    // Let us check the values of nextRight pointers
    printf("Following are populated nextRight pointers in the tree "
           "(-1 is printed if there is no nextRight) \n");
    printf("nextRight of %d is %d \n", root->data,
           root->nextRight? root->nextRight->data: -1);
    printf("nextRight of %d is %d \n", root->left->data,
           root->left->nextRight? root->left->nextRight->data: -1);
    printf("nextRight of %d is %d \n", root->right->data,
           root->right->nextRight? root->right->nextRight->data: -1);
    printf("nextRight of %d is %d \n", root->left->left->data,
           root->left->left->nextRight? root->left->left->nextRight->data: -1);
    printf("nextRight of %d is %d \n", root->right->right->data,
           root->right->right->nextRight? root->right->right->nextRight->data: -1);

    getchar();
    return 0;
}
```

Run on IDE

Java

```
// Iterative Java program to connect nodes at same level
// using constant extra space
// A binary tree node
class Node {

    int data;
    Node left, right, nextRight;

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

class BinaryTree {

    static Node root;

    /* This function returns the leftmost child of nodes at the same level as p.
     This function is used to getNExt right of p's right child
     If right child of is NULL then this can also be sued for the left child */
    Node getNextRight(Node p) {
        Node temp = p.nextRight;

        /* Traverse nodes at p's level and find and return
         the first node's first child */
        while (temp != null) {
            if (temp.left != null) {
                return temp.left;
            }
            if (temp.right != null) {
                return temp.right;
            }
            temp = temp.nextRight;
        }

        // If all the nodes at p's level are leaf nodes then return NULL
        return null;
    }

    /* Sets nextRight of all nodes of a tree with root as p */
    void connect(Node p) {
        Node temp = null;

        if (p == null) {
            return;
        }

        // Set nextRight for root
        p.nextRight = null;

        // set nextRight of all levels one by one
        while (p != null) {
            Node q = p;

            /* Connect all childrem nodes of p and children nodes of all other nodes
             at same level as p */
            while (q != null) {

                // Set the nextRight pointer for p's left child
                if (q.left != null) {

                    // If q has right child, then right child is nextRight of
                    // p and we also need to set nextRight of right child
                    if (q.right != null) {
                        q.left.nextRight = q.right;
                    } else {
                        q.left.nextRight = getNextRight(q);
                    }
                }

                if (q.right != null) {
                    q.right.nextRight = getNextRight(q);
                }

                // Set nextRight for other nodes in pre order fashion
                q = q.nextRight;
            }

            // start from the first node of next level
            if (p.left != null) {
                p = p.left;
            } else if (p.right != null) {
                p = p.right;
            } else {
                p = getNextRight(p);
            }
        }
    }

    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.right.right = new Node(90);

        // Populates nextRight pointer in all nodes
        tree.connect(root);

        // Let us check the values of nextRight pointers
        int a = root.nextRight != null ? root.nextRight.data : -1;
        int b = root.left.nextRight != null ? root.left.nextRight.data : -1;
        int c = root.right.nextRight != null ? root.right.nextRight.data : -1;
        int d = root.left.left.nextRight != null ? root.left.left.nextRight.data : -1;
        int e = root.right.right.nextRight != null ? root.right.right.nextRight.data : -1;

        // Now lets print the values
        System.out.println("Following are populated nextRight pointers in "
                + " the tree(-1 is printed if there is no nextRight)");
        System.out.println("nextRight of " + root.data + " is " + a);
        System.out.println("nextRight of " + root.left.data + " is " + b);
        System.out.println("nextRight of " + root.right.data + " is " + c);
        System.out.println("nextRight of " + root.left.left.data + " is " + d);
        System.out.println("nextRight of " + root.right.right.data + " is " + e);
    }
}
```

Run on IDE

Output:

```
Following are populated nextRight pointers in the tree (-1 is printed if
there is no nextRight)
nextRight of 10 is -1
nextRight of 8 is 2
nextRight of 2 is -1
nextRight of 3 is 90
nextRight of 90 is -1
```

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

118 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?

(Login to Rate and Mark)

4.2 Average Difficulty : **4.2/5.0**
Based on **19** vote(s)

☐ Add to TODO List
☐ Mark as DONE

Like Share 4 people like this.

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

118 Comments **GeeksforGeeks** 1 Login

Recommend 2 Share Sort by Newest

Join the discussion…

keyzer soje • 5 days ago
simple solution using queue.
<script src="http://ideone.com/e.js/ytI7YH" type="text/javascript"></script>
• Reply • Share ›

sai • 2 months ago
Can any one explain the question ?
I didn't understand the words 'using constant space' .
In the previous post ,using level order approach we can easily solve the problem right..!!
Why that problem is not using constant space ?
• Reply • Share ›

Anand Nahar sai • a month ago
Level order traversal needs a queue..so the space complexity there is O(n)...The 2nd method in previous post works only for Complete Binary Tree...In this post they have provided 2 methods without using any extra space.
• Reply • Share ›

Xploiter_coder Anand Nahar • 19 days ago
Are function calls not using extra space in recursive implementation ?
• Reply • Share ›

Vibhor Garg • 4 months ago
The code I have written also tackles all the cases very well.
Pls let me know if there is some case where it fails.
Pls find a cleaner code at http://code.geeksforgeeks.org/...
and let me know if there are issues with it .
I have used a BST to create any type of tree i want easily.
• Reply • Share ›

mukesh_p • 5 months ago
The above recursion is having problem your are traversing extreme right node multiple times hence time complexity can not be O(n)... Please find below code with some modification to above recursion approach. Instead of visiting left node first traverse all the right node and do not traverse peer node if its already been processed to avoid mulptile

right node and do not traverse peer node if its already been processed to avoid mulptile times traversing,.

void ConnectPeer(TNodePeer* root)
{

//base case
if(root == NULL)
return;

printf("%d\n",root->data);

if(!root->peer)
ConnectPeer(root->peer);

if(root->left)

see more

• Reply • Share ›

Koustav Chatterjee • 6 months ago

Java solution http://ideone.com/oiEuSO

Let me know if you find issues with it.

• Reply • Share ›

Somesh Suman • 6 months ago

we are going it to the rightmost node at current level using below
if (p->nextRight != NULL)
connectRecur(p->nextRight);

and after that we are calling below, so will not it process the right child first at all level and then left child pointer nextRight to right child.

connectRecur(p->left);
connectRecur(p->right);

Also, what is the time complexity of this also.

Please clear my confusion.

• Reply • Share ›

Somesh Suman • 6 months ago

Please correct me if I am wrong. I am not able to quite get the algo

Whole subtree of a node's right child is built first then the left child is taken and then again whole right subtree is called.

so basically all the nodes which are on the right side will be called multiple times?

• Reply • Share ›

Maddy • 7 months ago

In recursive solution,
At node p, connectRecur(p->left) and connectRecur(p->right) must be called

But if p->left exist, p->right will not be processed, creating the break in the required structure.

Please let me know If I am wrong.

• Reply • Share ›

मनोज पाटीदार Maddy • 6 months ago

in recursive solution there is no need of calling connectRecur for both left and right if the left one is exist, coz the rightNext of left will be right itself. and rightNext of right will be the getNextRight. getNextRight is there for it. in case if left doesnot exist then we go for connectRecur(right).

• Reply • Share ›

Mohit Gupta • 7 months ago

if (p->nextRight != NULL)

connectRecur(p->nextRight);

In above condition we are creating nextRight links for all nodes which are other than rooted node at same or greater level.Is it not having duplication.

I mean suppose i am on node 2 in that case, all the next Right pointers will be available for all the childrens of node 3 till the leaf of node 3 subrooted tree.

But when i again traverse for (node2->left) ie node 4 , there again this call for setting next Right pointers will happen.But some of the nodes will already be having next right pointers like node 7 with respect to the previous call, but will be created again.

Isn't there some duplication here .Please clear my doubt.

• Reply • Share ›

amirav Mohit Gupta • 6 months ago

Agree with you Mohit

• Reply • Share ›

codelearner • 7 months ago

simple using iterative level order approach https://ideone.com/6JKuNV

• Reply • Share ›

Arpit Quickgun Arora • 7 months ago

couldn't we just use a queue of the size equal to maximum width of the tree? That's the maximum we would ever need. Plus the time complexity won't be compromised, and will remain O(n).

• Reply • Share ›

Koustav Chatterjee • 7 months ago

if (p->nextRight != NULL)
connectRecur(p->nextRight);
What is the use of this ?

• Reply • Share ›

Maddy Koustav Chatterjee • 7 months ago

Hi Koustav,
Let me know if I am wrong.

He have said in comment that

/* Before setting nextRight of left and right children, set nextRight of children of other nodes at same level (because we can access children of other nodes using p's nextRight only) */

I Just wanted to know, why do you think this condition is not required or inappropriate..?

• Reply • Share ›

Billionaire • 7 months ago

See my Java solutions:

http://ideone.com/k1i1sH

1 • Reply • Share ›

Jayesh • 7 months ago

Java Implementation.

http://javabypatel.blogspot.in...

• Reply • Share ›

Shivam • 8 months ago

Doesn't recursion cause memory to be used? How is its space complexity O(1)?

• Reply • Share ›

Gaurav Arora • 8 months ago

For a recursive solution, we can first set the nextRight pointer of the current node's children and then first recur for the right child and then for the left child.

This would make sure that the nextRight pointers in the right sub-tree are already in their valid state when accessed by nodes in the left sub-tree.

• Reply • Share ›

Koustav Chatterjee Gaurav Arora • 7 months ago

Hey,

Can you explain a bit more ?

• Reply • Share ›

Gaurav Arora Koustav Chatterjee • 7 months ago

Hi,

It's like a simple pre-order traversal of the tree but in the order:
root
Right
Left.

Let's say I'm currently at node n which is at a depth of L.
Then all the nodes which are at level >=L and lie towards the right of n (but not in the subtree rooted at n) would have already been visited.
This means the nextRight pointer of these nodes are already in valid state.

So, for this node I can easily set the nextRight pointer of its children and then recur its right child and then the left child.

Tell me if you're still unclear or find some flaw here.

• Reply • Share ›

aashish • 9 months ago

else connectRecur(getNextRight(p));

what is the use of this statement when we have already calculated

connectRecur(getNextRight(p));

• Reply • Share ›

amirav aashish • 6 months ago

Did you get an answer to this aashish?
I have the same query, it looks like duplication.

• Reply • Share ›

Bibek Luitel • 9 months ago

implementation of using iterative .. O(n) time and without extra space used

```
void connect( struct node * root){
struct node * head= NULL, * prev= NULL;
root->nextRight= NULL;
if (root==NULL){ return; }
while(root){

if ( head== NULL && root->left){ head= root->left;}
if( head== NULL && root->right){ head= root->right;}
if(root->left && root->right){

root->left->nextRight= root->right;

if(prev){ prev->nextRight=root->left;}

prev= root->right;
}
else if (root->left){
```

see more

• Reply • Share ›

Swarnim Singhal • 9 months ago

An iterative implementation , extending level order traversal. However since a queue is used, space complexity no longer is O(1).

https://ideone.com/dGIQvB

• Reply • Share ›

shubham garg • 9 months ago

A simple implementation making use of earlier methods :p

```
#include<bits stdc++.h="">

using namespace std;

struct tree{

int data;

struct tree *left;

struct tree *right;

struct tree *rightptr;

};
```

struct tree* new_node(int x)

{

see more

• Reply • Share ›

Deepak Sharma • 9 months ago

iterative implementation.

http://ideone.com/IDQNrH

• Reply • Share ›

Anup Rai • 10 months ago

Tell me if I am wrong (or right)

In the recursive method

The code start at root.

now it goes to its left child then right child

parent will call the rec function for the left child.

the left child will call the rec func for its next right (the righ child of the root)

the whole subtree rooted at root right will be built first then the function will return to the root's left child.

Example:
tree is

______10
___/______\

see more

• Reply • Share ›

debugger • a year ago

else
connectRecur(getNextRight(p));

This is unnecessary and can simply be replaced by return;

• Reply • Share ›

Mr. Lazy ➦ debugger • 10 months ago

You are right.. its not needed in recursive method.

However the condition is required in iterative method.

1 ^ | ˅ • Reply • Share ›

Santosh Sarangi ➦ debugger • 10 months ago

Yes I agree with you.
Another problem with iterative code is " else if (p->right) " should be "else".
Otherwise it will lead to infinite loop.

^ | ˅ • Reply • Share ›

Ekta Goel ➦ debugger • a year ago

That is necessary to move on to the next level in cases like this
http://ideone.com/3uilM8

^ | ˅ • Reply • Share ›

Mr. Lazy ➦ Ekta Goel • 10 months ago

For Recursive Method: That's definitely unnecessary... it will move to 4 using connectRecur(root->nextRight) ... check it using paper pen.

However, the condition is required in iterative version.

1 ^ | ˅ • Reply • Share ›

debugger ➦ Ekta Goel • a year ago

In your example, 4 will be accessed from 3 by below code.
else
p->left->nextRight = getNextRight(p);

/* Recursively call for next level nodes. Note that we call only
for left child. The call for left child will call for right child */
connectRecur(p->left); //HERE

Pls check the code carefully.I am talking about removing
else
connectRecur(getNextRight(p));

You can refer http://ideone.com/KqRonU.
Line No. 63.
Let me know if its clear.

^ | ˅ • Reply • Share ›

Anupam ➦ debugger • a month ago

I guess then that may not be required in iterative solution also

^ | ˅ • Reply • Share ›

Ekta Goel ➦ debugger • 9 months ago

Yeah, got it!!

1 • Reply • Share ›

neelabhsingh • a year ago

what about this method:

http://ideone.com/zjKovP

• Reply • Share ›

Guest • a year ago

@GeeksForGeeks plz hyperlink : A Recursive Solution
In the method 2 of previous post previous post for shake of completion. thanks

• Reply • Share ›

paul • a year ago

Why not use a right to left, level order traversal and maintain a previous pointer ref which will be the nextRight of the node at the same level.
Much easier and cleaner.

• Reply • Share ›

Sumit Kesarwani • a year ago

T

• Reply • Share ›

Sumit Kesarwani • a year ago

Can any one please tell me .... In recursive , in else condition why they have given "connectRecur(getNextRight(p));"... ?..
thanks in advance

• Reply • Share ›

Guest → Sumit Kesarwani • a year ago

My similar solution:

```
void connectTrees(struct node *root)
{
............if ( root==NULL )
.................return;

.............connectTrees(root->nextRight);
.............if ( root->left )
.............{
.....................if ( root->right )
...........................root->left->nextRight = root->right;
......................else
```

............................root->left->nextRight = getNextRight (root->nextRight);
...............}

...............if (root->right)
..........................root->right->nextRight = getNextRight (root->nextRight);

...............connectTrees(root->left);
...............connectTrees(root->right);
}

Reply • Share ›

Nischay Guest • 5 months ago

I ,like this solution , but one doubt to clear. DO we really need to recur for the connectTrees(root->right).
Dont you think that It will get covered while traversing the connectTrees(root->nextRight) ????? Please explain

Reply • Share ›

Guest Nischay • 4 months ago

No, as an example for the processing for the below tree:
..............A
............/....\
..........B.....C
......../...\.......\
......D....E......F

(1) B->nextRight = C
(2) D->nextRight = E (via connectTrees(root->left))
(3) E->nextRight = F (via connectTrees(root->right))

Reply • Share ›

Sumit Kesarwani Guest • a year ago

Oh.. greate.. :)

Reply • Share ›

oO Sumit Kesarwani • a year ago

Seems unnecessary. When you are DOing it for a node say 'p', the first thing you do is DO it for its getNextRight. Now if this node 'p' does not have any children, you would DO for its getNextRight again - which would in turn repeat for every node of getNextRight which had already been done. Stupid is what this seems to me.

Reply • Share ›

rihansh • a year ago

http://ideone.com/CBVB9D

http://ideone.com/CRVR9D

Both methods are implemented at this link :D

Reply • Share ›

ryan • a year ago

@GeeksforGeeks in first method bottom right corner will be multiple time right connected?

Reply • Share ›

Load more comments