Connect nodes at same level

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;
}
```

Initially, all the nextRight pointers point to garbage values. Your function should set these pointers to point next right for each node.

Example

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


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

Method 1 (Extend Level Order Traversal or BFS)

Consider the method 2 of Level Order Traversal. The method 2 can easily be extended to connect nodes of same level. We can augment queue entries to contain level of nodes also which is 0 for root, 1 for root's children and so on. So a queue node will now contain a pointer to a tree node and an integer level. When we enqueue a node, we make sure that correct level value for node is being set in queue. To set nextRight, for

every node N, we dequeue the next node from queue, if the level number of next node is same, we set the nextRight of N as address of the dequeued node, otherwise we set nextRight of N as NULL.

Time Complexity: O(n)

Method 2 (Extend Pre Order Traversal)

This approach works only for Complete Binary Trees. In this method we set nextRight in Pre Order fashion to make sure that the nextRight of parent is set before its children. When we are at node p, we set the nextRight of its left and right children. Since the tree is complete tree, nextRight of p's left child (p->left->nextRight) will always be p's right child, and nextRight of p's right child (p->right->nextRight) will always be left child of p's nextRight (if p is not the rightmost node at its level). If p is the rightmost node, then nextRight of p's right child will be NULL.

C

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

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

void connectRecur(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.
   Assumption:  p is a compete binary tree */
void connectRecur(struct node* p)
{
  // Base case
  if (!p)
    return;

  // Set the nextRight pointer for p's left child
```

```
  if (p->left)
    p->left->nextRight = p->right;

  // Set the nextRight pointer for p's right child
  // p->nextRight will be NULL if p is the right most child at its level
  if (p->right)
    p->right->nextRight = (p->nextRight)? p->nextRight->left: NULL;

  // Set nextRight for other nodes in pre order fashion
  connectRecur(p->left);
  connectRecur(p->right);
}

/* 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
  */
  struct node *root = newnode(10);
  root->left        = newnode(8);
  root->right       = newnode(2);
  root->left->left  = newnode(3);

  // 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);

  getchar();
  return 0;
}
```

Java

```
// Java program to connect nodes at same level using extended
// pre-order traversal

// 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;

    // Sets the nextRight of root and calls connectRecur() for other nodes
    void connect(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.
     Assumption:  p is a compete binary tree */
    void connectRecur(Node p) {

        // Base case
```

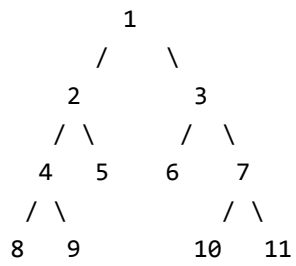
```
        if (p == null) {
            return;
        }

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

        // Set the nextRight pointer for p's right child
        // p->nextRight will be NULL if p is the right most child at its level
        if (p.right != null) {
            p.right.nextRight = (p.nextRight != null) ? p.nextRight.left : null;
        }

        // Set nextRight for other nodes in pre order fashion
        connectRecur(p.left);
        connectRecur(p.right);
    }

    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);

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

        // Let us check the values of nextRight pointers
        System.out.println("Following are populated nextRight pointers in the tree "
                + "(-1 is printed if there is no nextRight)");
        int a = root.nextRight != null ? root.nextRight.data : -1;
        System.out.println("nextRight of " + root.data + " is "
                + a);
        int b = root.left.nextRight != null ? root.left.nextRight.data : -1;
        System.out.println("nextRight of " + root.left.data + " is "
                + b);
        int c=root.right.nextRight != null ? root.right.nextRight.data : -1;
        System.out.println("nextRight of " + root.right.data + " is "
                + c);
        int d = root.left.left.nextRight != null ? root.left.left.nextRight.data : -1;
        System.out.println("nextRight of " + root.left.left.data + " is "
                + d);

    }
}

// This code has been contributed by Mayank Jaiswal
```

Thanks to Dhanya for suggesting this approach.

Time Complexity: O(n)

Why doesn't method 2 work for trees which are not Complete Binary Trees?

Let us consider following tree as an example. In Method 2, we set the nextRight pointer in pre order fashion. When we are at node 4, we set the nextRight of its children which are 8 and 9 (the nextRight of 4 is already set as node 5). nextRight of 8 will simply be set as 9, but nextRight of 9 will be set as NULL which is incorrect. We can't set the correct nextRight, because when we set nextRight of 9, we only have nextRight of node 4 and ancestors of node 4, we don't have nextRight of nodes in right subtree of root.



See next post for more solutions.

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

135 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)

3.1 Average Difficulty : **3.1/5.0**
Based on **20** vote(s)

Add to TODO List

Mark as DONE

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