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;
}
```

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

```
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;
}
```

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
{
    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;
    }

    /* Driver program to test the 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.right.right = new Node(90);

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

        // Let us check the values of nextRight pointers
        int a = tree.root.nextRight != null ?
                         tree.root.nextRight.data : -1;
        int b = tree.root.left.nextRight != null ?
                         tree.root.left.nextRight.data : -1;
        int c = tree.root.right.nextRight != null ?
                           tree.root.right.nextRight.data : -1;
        int d = tree.root.left.left.nextRight != null ?
                        tree.root.left.left.nextRight.data : -1;
        int e = tree.root.right.right.nextRight != null ?
                        tree.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 " + tree.root.data + " is " + a);
        System.out.println("nextRight of " + tree.root.left.data + " is " + b);
        System.out.println("nextRight of " + tree.root.right.data + " is " + c);
        System.out.println("nextRight of " + tree.root.left.left.data +
                                                          " is " + d);
        System.out.println("nextRight of " + tree.root.right.right.data +
                                                          " is " + e);
    }
}

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

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;
}
```

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;
```

```
        left = right = nextRight = null;
    }
}

class BinaryTree
{
    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);
        }
    }

    /* Driver program to test above functions */
    public static void main(String args[])
    {
         /* Constructed binary tree is
                10
              /   \
             8      2
           /         \
         3            90
        */
        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(tree.root);

        // Let us check the values of nextRight pointers
        int a = tree.root.nextRight != null ?
                                    tree.root.nextRight.data : -1;
        int b = tree.root.left.nextRight != null ?
                                 tree.root.left.nextRight.data : -1;
        int c = tree.root.right.nextRight != null ?
                                tree.root.right.nextRight.data : -1;
        int d = tree.root.left.left.nextRight != null ?
                                  tree.root.left.left.nextRight.data : -1;
        int e = tree.root.right.right.nextRight != null ?
                                   tree.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 " + tree.root.data + " is " + a);
        System.out.println("nextRight of " + tree.root.left.data
                                                    + " is " + b);
        System.out.println("nextRight of " + tree.root.right.data +
                                                         " is " + c);
        System.out.println("nextRight of " + tree.root.left.left.data +
                                                           " is " + d);
        System.out.println("nextRight of " + tree.root.right.right.data +
                                                            " is " + e);
    }
}

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

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
```