Extract Leaves of a Binary Tree in a Doubly Linked List

Given a Binary Tree, extract all leaves of it in a **D**oubly **L**inked **L**ist (DLL). Note that the DLL need to be created in-place. Assume that the node structure of DLL and Binary Tree is same, only the meaning of left and right pointers are different. In DLL, left means previous pointer and right means next pointer.

```
Let the following be input binary tree
        1
     /     \
    2       3
   / \       \
  4   5       6
 / \         / \
7   8       9   10


Output:
Doubly Linked List
7<->8<->5<->9<->10

Modified Tree:
        1
     /     \
    2       3
   /         \
  4           6
```

We strongly recommend you to minimize the browser and try this yourself first.

We need to traverse all leaves and connect them by changing their left and right pointers. We also need to remove them from Binary Tree by changing left or right pointers in parent nodes. There can be many ways to solve this. In the following implementation, we add leaves at the beginning of current linked list and update head of the list using pointer to head pointer. Since we insert at the beginning, we need to process leaves in reverse order. For reverse order, we first traverse the right subtree then the left subtree. We use return values to update left or right pointers in parent nodes.

C

```
// C program to extract leaves of a Binary Tree in a Doubly Linked List
#include <stdio.h>
#include <stdlib.h>

// Structure for tree and linked list
struct Node
{
    int data;
    struct Node *left, *right;
};

// Main function which extracts all leaves from given Binary Tree.
// The function returns new root of Binary Tree (Note that root may change
// if Binary Tree has only one node).  The function also sets *head_ref as
// head of doubly linked list.  left pointer of tree is used as prev in DLL
// and right pointer is used as next
struct Node* extractLeafList(struct Node *root, struct Node **head_ref)
{
   // Base cases
   if (root == NULL)  return NULL;

   if (root->left == NULL && root->right == NULL)
   {
       // This node is going to be added to doubly linked list
       // of leaves, set right pointer of this node as previous
```

```
        // of leaves, set right pointer of this node as previous
        // head of DLL. We don't need to set left pointer as left
        // is already NULL
        root->right = *head_ref;

        // Change left pointer of previous head
        if (*head_ref != NULL) (*head_ref)->left = root;

        // Change head of linked list
        *head_ref = root;

        return NULL;  // Return new root
    }

    // Recur for right and left subtrees
    root->right = extractLeafList(root->right, head_ref);
    root->left  = extractLeafList(root->left, head_ref);

    return root;
}

// Utility function for allocating node for Binary Tree.
struct Node* newNode(int data)
{
    struct Node* node = (struct Node*)malloc(sizeof(struct Node));
    node->data = data;
    node->left = node->right = NULL;
    return node;
}

// Utility function for printing tree in In-Order.
void print(struct Node *root)
{
    if (root != NULL)
    {
         print(root->left);
         printf("%d ",root->data);
         print(root->right);
    }
}

// Utility function for printing double linked list.
void printList(struct Node *head)
{
     while (head)
     {
         printf("%d ", head->data);
         head = head->right;
     }
}

// Driver program to test above function
int main()
{
     struct Node *head = NULL;
     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->right = newNode(6);
     root->left->left->left = newNode(7);
     root->left->left->right = newNode(8);
     root->right->right->left = newNode(9);
     root->right->right->right = newNode(10);

     printf("Inorder Trvaersal of given Tree is:\n");
     print(root);

     root = extractLeafList(root, &head);

     printf("\nExtracted Double Linked list is:\n");
     printList(head);
```

```
     printf("\nInorder traversal of modified tree is:\n");
     print(root);
     return 0;
}
```

Java

```
// Java program to extract leaf nodes from binary tree
// using double linked list

// A binay tree node
class Node
{
    int data;
    Node left, right;

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

public class BinaryTree
{
    Node root;
    Node head; // will point to head of DLL
    Node prev; // temporary pointer

    // The main fuction that links the list list to be traversed
    public Node extractLeafList(Node root)
    {
        if (root == null)
            return null;
        if (root.left == null && root.right == null)
        {
            if (head == null)
            {
                head = root;
                prev = root;
            }
            else
            {
                prev.right = root;
                root.left = prev;
                prev = root;
            }
            return null;
        }
        root.left = extractLeafList(root.left);
        root.right = extractLeafList(root.right);
        return root;
    }

    //Prints the DLL in both forward and reverse directions.
    public void printDLL(Node head)
    {
        Node last = null;
        while (head != null)
        {
            System.out.print(head.data + " ");
            last = head;
            head = head.right;
        }
    }

    void inorder(Node node)
    {
        if (node == null)
```

```
        if (node == null)
            return;
        inorder(node.left);
        System.out.print(node.data + " ");
        inorder(node.right);
    }

    // Driver program to test above functions
    public static void main(String args[])
    {
        BinaryTree tree = new BinaryTree();
        tree.root = new Node(1);
        tree.root.left = new Node(2);
        tree.root.right = new Node(3);

        tree.root.left.left = new Node(4);
        tree.root.left.right = new Node(5);
        tree.root.right.right = new Node(6);
        tree.root.left.left.left = new Node(7);
        tree.root.left.left.right = new Node(8);
        tree.root.right.right.left = new Node(9);
        tree.root.right.right.right = new Node(10);

        System.out.println("Inorder traversal of given tree is : ");
        tree.inorder(tree.root);
        tree.extractLeafList(tree.root);
        System.out.println("");
        System.out.println("Extracted double link list is : ");
        tree.printDLL(tree.head);
        System.out.println("");
        System.out.println("Inorder traversal of modified tree is : ");
        tree.inorder(tree.root);
    }
}

// This code has been contributed by Mayank Jaiswal(mayank_24)
```

Python

```
# Python program to extract leaf nodes from binary tree
# using double linked list

# A binary tree node
class Node:

    # Constructor to create a new node
    def __init__(self, data):
        self.data = data
        self.left = None
        self.right = None

# Main function which extracts all leaves from given Binary Tree.
# The function returns new root of Binary Tree (Note that
# root may change if Binary Tree has only one node).
# The function also sets *head_ref as head of doubly linked list.
# left pointer of tree is used as prev in DLL
# and right pointer is used as next
def extractLeafList(root):

    # Base Case
    if root is None:
        return None

    if root.left is None and root.right is None:
        # This node is going to be added to doubly linked
        # list of leaves, set pointer of this node as
        # previous head of DLL. We don't need to set left
        # pointer as left is already None
        root.right = extractLeafList.head
```

```
        # Change the left pointer of previous head
        if extractLeafList.head is not None:
            extractLeafList.head.left = root

        # Change head of linked list
        extractLeafList.head = root

        return None # Return new root

    # Recur for right and left subtrees
    root.right = extractLeafList(root.right)
    root.left = extractLeafList(root.left)

    return root

# Utility function for printing tree in InOrder
def printInorder(root):
    if root is not None:
        printInorder(root.left)
        print root.data,
        printInorder(root.right)


def printList(head):
    while(head):
        if head.data is not None:
            print head.data,
        head = head.right

# Driver program to test above function
extractLeafList.head = Node(None)
root = Node(1)
root.left = Node(2)
root.right = Node(3)
root.left.left = Node(4)
root.left.right = Node(5)
root.right.right = Node(6)
root.left.left.left = Node(7)
root.left.left.right = Node(8)
root.right.right.left = Node(9)
root.right.right.right = Node(10)

print "Inorder traversal of given tree is:"
printInorder(root)

root = extractLeafList(root)

print "\nExtract Double Linked List is:"
printList(extractLeafList.head)

print "\nInorder traversal of modified tree is:"
printInorder(root)
```

Output:

```
Inorder Trvaersal of given Tree is:
7 4 8 2 5 1 3 9 6 10
Extracted Double Linked list is:
7 8 5 9 10
Inorder traversal of modified tree is:
4 2 1 3 6
```

Time Complexity: O(n), the solution does a single traversal of given Binary Tree.