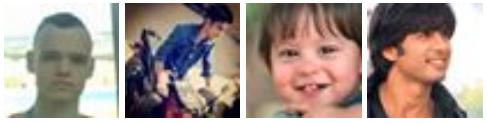
Extract Leaves of a Binary Tree in a Doubly Linked List

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
```

Interview Experiences
Advanced Data Structures
Dynamic Programming
Greedy Algorithms
Backtracking
Pattern Searching
Divide & Conquer
Mathematical Algorithms
Recursion
Geometric Algorithms

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 program to extract leaves of a Binary Tree in a Doubly Linked Lis
#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 ch
// if Binary Tree has only one node).  The function also sets *head_re
// head of doubly linked list.  left pointer of tree is used as prev i
// 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
       // 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;
```


Popular Posts

```
        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;
}
```

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.

This article is contributed by **Chandra Prakash**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Recent Comments

affiszerv Your example has two 4s on row 3, that's why it...

Backtracking | Set 7 (Sudoku) · 27 minutes ago

RVM Can someone please elaborate this Qs from above...

Flipkart Interview | Set 6 · 47 minutes ago

Vishal Gupta I talked about as an Interviewer in general,...

Software Engineering Lab, Samsung Interview | Set 2 · 47 minutes ago

@meya Working solution for question 2 of 4f2f round....

Amazon Interview | Set 53 (For SDE-1) · 1 hour ago

sandeep void rearrange(struct node *head) {...

Given a linked list, reverse alternate nodes and append at the end · 2 hours ago

Neha I think that is what it should return as, in...

Find depth of the deepest odd level leaf node · 2 hours ago

Related Tpoics:

- Print a Binary Tree in Vertical Order | Set 2 (Hashmap based Method)
- Print Right View of a Binary Tree
- Red-Black Tree | Set 3 (Delete)
- Construct a tree from Inorder and Level order traversals
- Print all nodes at distance k from a given node
- Print a Binary Tree in Vertical Order | Set 1
- Interval Tree
- Check if a given Binary Tree is height balanced like a Red-Black Tree

Writing code in comment? Please use **ideone.com** and share the link here.