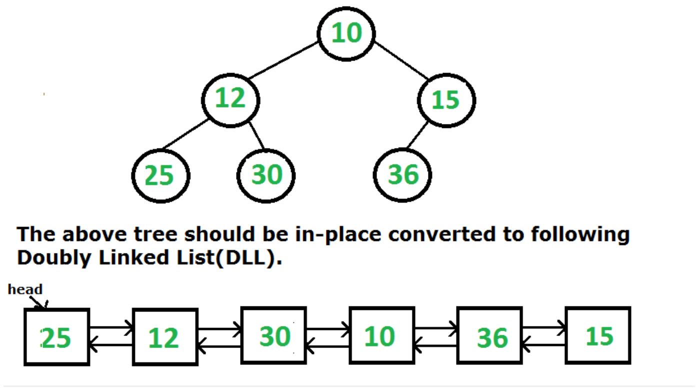
Convert a given Binary Tree to Doubly Linked List | Set 1

Given a Binary Tree (Bt), convert it to a Doubly Linked List(DLL). The left and right pointers in nodes are to be used as previous and next pointers respectively in converted DLL. The order of nodes in DLL must be same as Inorder of the given Binary Tree. The first node of Inorder traversal (left most node in BT) must be head node of the DLL.


10
12
15
25
30
36
The above tree should be in-place converted to following Doubly Linked List(DLL).
head
25
12
30
10
36
15

I came across this interview during one of my interviews. A similar problem is discussed in this post. The problem here is simpler as we don't need to create circular DLL, but a simple DLL. The idea behind its solution is quite simple and straight.

1. If left subtree exists, process the left subtree
.....**1.a)** Recursively convert the left subtree to DLL.
.....**1.b)** Then find inorder predecessor of root in left subtree (inorder predecessor is rightmost node in left subtree).
.....**1.c)** Make inorder predecessor as previous of root and root as next of inorder predecessor.
2. If right subtree exists, process the right subtree (Below 3 steps are similar to left subtree).
.....**2.a)** Recursively convert the right subtree to DLL.
.....**2.b)** Then find inorder successor of root in right subtree (inorder successor is leftmost node in right subtree).
.....**2.c)** Make inorder successor as next of root and root as previous of inorder successor.
3. Find the leftmost node and return it (the leftmost node is always head of converted DLL).

Below is the source code for above algorithm.

C

```
// A C++ program for in-place conversion of Binary Tree to DLL
#include <stdio.h>

/* A binary tree node has data, and left and right pointers */
struct node
{
    int data;
    node* left;
    node* right;
};

/* This is the core function to convert Tree to list. This function follows
  steps 1 and 2 of the above algorithm */
node* bintree2listUtil(node* root)
{
    // Base case
    if (root == NULL)
        return root;

    // Convert the left subtree and link to root
    if (root->left != NULL)
    {
        // Convert the left subtree
        node* left = bintree2listUtil(root->left);
        // Find inorder predecessor. After this loop, left
        // will point to the inorder predecessor
        for (; left->right!=NULL; left=left->right);

        // Make root as next of the predecessor
        left->right = root;

        // Make predecssor as previous of root
        root->left = left;
    }

    // Convert the right subtree and link to root
    if (root->right!=NULL)
    {
        // Convert the right subtree
        node* right = bintree2listUtil(root->right);

        // Find inorder successor. After this loop, right
        // will point to the inorder successor
        for (; right->left!=NULL; right = right->left);

        // Make root as previous of successor
        right->left = root;

        // Make successor as next of root
        root->right = right;
    }

    return root;
}

// The main function that first calls bintree2listUtil(), then follows step 3
//  of the above algorithm
node* bintree2list(node *root)
{
    // Base case
    if (root == NULL)
        return root;

    // Convert to DLL using bintree2listUtil()
    root = bintree2listUtil(root);

    // bintree2listUtil() returns root node of the converted
    // DLL.  We need pointer to the leftmost node which is
    // head of the constructed DLL, so move to the leftmost node
    while (root->left != NULL)
        root = root->left;

    return (root);
}

/* Helper function that allocates a new node with the
   given data and NULL left and right pointers. */
node* newNode(int data)
{
    node* new_node = new node;
    new_node->data = data;
    new_node->left = new_node->right = NULL;
    return (new_node);
}

/* Function to print nodes in a given doubly linked list */
void printList(node *node)
{
    while (node!=NULL)
    {
        printf("%d ", node->data);
        node = node->right;
    }
}

/* Driver program to test above functions*/
```

```
int main()
{
    // Let us create the tree shown in above diagram
    node *root        = newNode(10);
    root->left        = newNode(12);
    root->right       = newNode(15);
    root->left->left  = newNode(25);
    root->left->right = newNode(30);
    root->right->left = newNode(36);

    // Convert to DLL
    node *head = bintree2list(root);

    // Print the converted list
    printList(head);

    return 0;
}
```

Java

```
// Java program to convert binary tree to double linked list

/* A binary tree node has data, and left and right pointers */
class Node
{
    int data;
    Node left, right;

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

class BinaryTree
{
    Node root;
    /* This is the core function to convert Tree to list. This function
       follows steps 1 and 2 of the above algorithm */

    Node bintree2listUtil(Node node)
    {
        // Base case
        if (node == null)
            return node;

        // Convert the left subtree and link to root
        if (node.left != null)
        {
            // Convert the left subtree
            Node left = bintree2listUtil(node.left);

            // Find inorder predecessor. After this loop, left
            // will point to the inorder predecessor
            for (; left.right != null; left = left.right);

            // Make root as next of the predecessor
            left.right = node;

            // Make predecssor as previous of root
            node.left = left;
        }

        // Convert the right subtree and link to root
        if (node.right != null)
        {
            // Convert the right subtree
            Node right = bintree2listUtil(node.right);
            // Find inorder successor. After this loop, right
            // will point to the inorder successor
            for (; right.left != null; right = right.left);

            // Make root as previous of successor
            right.left = node;

            // Make successor as next of root
            node.right = right;
        }

        return node;
    }

    // The main function that first calls bintree2listUtil(), then follows
    // step 3 of the above algorithm

    Node bintree2list(Node node)
    {
        // Base case
        if (node == null)
            return node;

        // Convert to DLL using bintree2listUtil()
        node = bintree2listUtil(node);

        // bintree2listUtil() returns root node of the converted
        // DLL.  We need pointer to the leftmost node which is
        // head of the constructed DLL, so move to the leftmost node
        while (node.left != null)
            node = node.left;

        return node;
    }

    /* Function to print nodes in a given doubly linked list */
    void printList(Node node)
    {
        while (node != null)
        {
            System.out.print(node.data + " ");
            node = node.right;
        }
    }

    /* Driver program to test above functions*/
    public static void main(String[] args)
    {
        BinaryTree tree = new BinaryTree();

        // Let us create the tree shown in above diagram
        tree.root = new Node(10);
        tree.root.left = new Node(12);
        tree.root.right = new Node(15);
        tree.root.left.left = new Node(25);
        tree.root.left.right = new Node(30);
        tree.root.right.left = new Node(36);

        // Convert to DLL
        Node head = tree.bintree2list(tree.root);

        // Print the converted list
        tree.printList(head);
    }
}
```

Output:

```
25 12 30 10 36 15
```

You may also like to see Convert a given Binary Tree to Doubly Linked List | Set 2 for another simple and efficient solution.