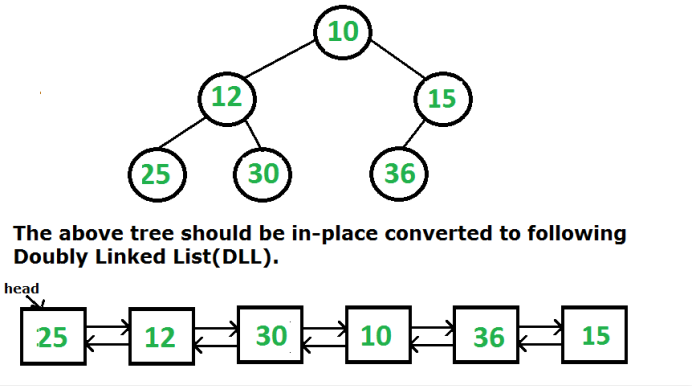
Convert a given Binary Tree to Doubly Linked List | Set 3

Given a Binary Tree (BT), convert it to a Doubly Linked List(DLL) In-Place. 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.



Following two different solutions have been discussed for this problem.

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

Convert a given Binary Tree to Doubly Linked List | Set 2

In this post, a third solution is discussed which seems to be the simplest of all. The idea is to do inorder traversal of the binary tree. While doing inorder traversal, keep track of the previously visited node in a variable say *prev*. For every visited node, make it next of *prev* and previous of this node as *prev*.

Thanks to rahul, wishall and all other readers for their useful comments on the above two posts.

Following is C++ implementation of this solution.

C++

```
// A C++ program for in-place conversion of Binary Tree to DLL
#include <iostream>
using namespace std;

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

// A simple recursive function to convert a given Binary tree to Doubly
// Linked List
// root --> Root of Binary Tree
// head --> Pointer to head node of created doubly linked list
void BinaryTree2DoubleLinkedList(node *root, node **head)
{
    // Base case
    if (root == NULL) return;

    // Initialize previously visited node as NULL. This is
    // static so that the same value is accessible in all recursive
    // calls
    static node* prev = NULL;

    // Recursively convert left subtree
    BinaryTree2DoubleLinkedList(root->left, head);
```

```
    // Now convert this node
    if (prev == NULL)
        *head = root;
    else
    {
        root->left = prev;
        prev->right = root;
    }
    prev = root;

    // Finally convert right subtree
    BinaryTree2DoubleLinkedList(root->right, head);
}

/* 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)
    {
        cout << 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 = NULL;
    BinaryTree2DoubleLinkedList(root, &head);

    // Print the converted list
    printList(head);

    return 0;
}
```

Java

```
// A Java program for in-place conversion of Binary Tree to DLL

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

    public Node(int data)
    {
        this.data = data;
        left = right = null;
```

```
    }
}

class BinaryTree
{
    Node root;

    // head --> Pointer to head node of created doubly linked list
    Node head;

    // Initialize previously visited node as NULL. This is
    // static so that the same value is accessible in all recursive
    // calls
    static Node prev = null;

    // A simple recursive function to convert a given Binary tree
    // to Doubly Linked List
    // root --> Root of Binary Tree
    void BinaryTree2DoubleLinkedList(Node root)
    {
        // Base case
        if (root == null)
            return;

        // Recursively convert left subtree
        BinaryTree2DoubleLinkedList(root.left);

        // Now convert this node
        if (prev == null)
            head = root;
        else
        {
            root.left = prev;
            prev.right = root;
        }
        prev = root;

        // Finally convert right subtree
        BinaryTree2DoubleLinkedList(root.right);
    }

    /* 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)
    {
        // Let us create the tree as shown in above diagram
        BinaryTree tree = new BinaryTree();
        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
        tree.BinaryTree2DoubleLinkedList(tree.root);

        // Print the converted List
        tree.printList(tree.head);

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

Output:

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

Note that use of static variables like above is not a recommended practice (we have used static for simplicity). Imagine a situation where same function is called for two or more trees, the old value of *prev* would be used in next call for a different tree. To avoid such problems, we can use double pointer or reference to a pointer.

Time Complexity: The above program does a simple inorder traversal, so time complexity is O(n) where n is the number of nodes in given binary tree.