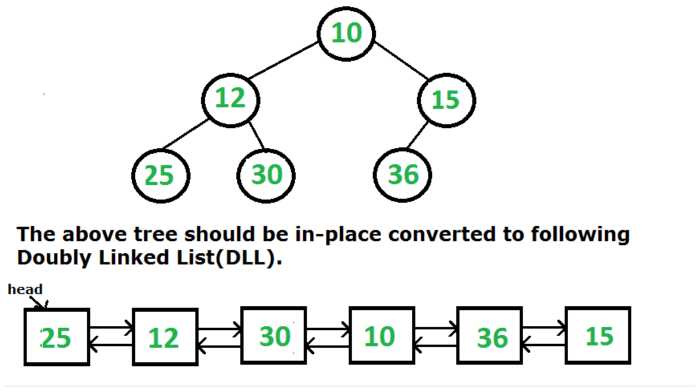
Convert a given Binary Tree to Doubly Linked List | Set 2

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

A solution to this problem is discussed in this post.
In this post, another simple and efficient solution is discussed. The solution discussed here has two simple steps.

1) ***Fix Left Pointers***: In this step, we change left pointers to point to previous nodes in DLL. The idea is simple, we do inorder traversal of tree. In inorder traversal, we keep track of previous visited node and change left pointer to the previous node. See *fixPrevPtr()* in below implementation.

2) ***Fix Right Pointers***: The above is intuitive and simple. How to change right pointers to point to next node in DLL? The idea is to use left pointers fixed in step 1. We start from the rightmost node in Binary Tree (BT). The rightmost node is the last node in DLL. Since left pointers are changed to point to previous node in DLL, we can linearly traverse the complete DLL using these pointers. The traversal would be from last to first node. While traversing the DLL, we keep track of the previously visited node and change the right pointer to the previous node. See *fixNextPtr()* in below implementation.

C

```
// A simple inorder traversal based program to convert a Binary Tree to DLL
#include<stdio.h>
#include<stdlib.h>

// A tree node
struct node
{
    int data;
    struct node *left, *right;
};

// A utility function to create a new tree node
struct node *newNode(int data)
{
    struct node *node = (struct node *)malloc(sizeof(struct node));
    node->data = data;
    node->left = node->right = NULL;
    return(node);
}

// Standard Inorder traversal of tree
void inorder(struct node *root)
{
    if (root != NULL)
    {
        inorder(root->left);
        printf("\t%d",root->data);
        printf("\t%d",root->data);
        inorder(root->right);
    }
}

// Changes left pointers to work as previous pointers in converted DLL
// The function simply does inorder traversal of Binary Tree and updates
// left pointer using previously visited node
void fixPrevPtr(struct node *root)
{
    static struct node *pre = NULL;

    if (root != NULL)
    {
        fixPrevPtr(root->left);
        root->left = pre;
        pre = root;
        fixPrevPtr(root->right);
    }
}

// Changes right pointers to work as next pointers in converted DLL
struct node *fixNextPtr(struct node *root)
{
    struct node *prev = NULL;

    // Find the right most node in BT or last node in DLL
    while (root && root->right != NULL)
        root = root->right;

    // Start from the rightmost node, traverse back using left pointers.
    // While traversing, change right pointer of nodes.
    while (root && root->left != NULL)
    {
        prev = root;
        root = root->left;
        root->right = prev;
    }

    // The leftmost node is head of linked list, return it
    return (root);
}

// The main function that converts BST to DLL and returns head of DLL
struct node *BTToDLL(struct node *root)
{
    // Set the previous pointer
    fixPrevPtr(root);

    // Set the next pointer and return head of DLL
    return fixNextPtr(root);
}

// Traverses the DLL from left tor right
void printList(struct node *root)
{
    while (root != NULL)
    {
        printf("\t%d", root->data);
        root = root->right;
    }
}

// Driver program to test above functions
int main(void)
{
    // Let us create the tree shown in above diagram
    struct 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);
    printf("\n\t\tInorder Tree Traversal\n\n");
    inorder(root);

    struct node *head = BTToDLL(root);

    printf("\n\n\t\tDLL Traversal\n\n");
    printList(head);
    return 0;
}
```

Python

```

# A simple inorder traversal based program to convert a
# Binary Tree to DLL

# A Binary Tree node
class Node:

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

# Standard Inorder traversal of tree
def inorder(root):

    if root is not None:
        inorder(root.left)
        print "\t%d" %(root.data),
        inorder(root.right)

# Changes left pointers to work as previous pointers
# in converted DLL
# The function simply does inorder traversal of
# Binary Tree and updates
# left pointer using previously visited node
def fixPrevPtr(root):
    if root is not None:
        fixPrevPtr(root.left)
        root.left = fixPrevPtr.pre
        fixPrevPtr.pre = root
        fixPrevPtr(root.right)

# Changes right pointers to work as nexr pointers in
# converted DLL
def fixNextPtr(root):

    prev = None
    # Find the right most node in BT or last node in DLL
    while(root and root.right != None):
        root = root.right

    # Start from the rightmost node, traverse back using
    # left pointers
    # While traversing, change right pointer of nodes
    while(root and root.left != None):
        prev = root
        root = root.left
        root.right = prev

    # The leftmost node is head of linked list, return it
    return root

# The main function that converts BST to DLL and returns
# head of DLL
def BTToDLL(root):

    # Set the previous pointer
    # Set the previous pointer
    fixPrevPtr(root)

    # Set the next pointer and return head of DLL
    return fixNextPtr(root)

# Traversses the DLL from left to right
def printList(root):
    while(root != None):
        print "\t%d" %(root.data),
        root = root.right

# Driver program to test above function
root = Node(10)
root.left = Node(12)
root.right = Node(15)
root.left.left = Node(25)
root.left.right = Node(30)
root.right.left = Node(36)

print "\n\t\t Inorder Tree Traversal\n"
inorder(root)

# Static variable pre for function fixPrevPtr
fixPrevPtr.pre = None
head = BTToDLL(root)

print "\n\n\t\tDLL Traversal\n"
printList(head)

# This code is contributed by Nikhil Kumar Singh(nickzuck_007)
```

Output:

```
                Inorder Tree Traversal

        25      12      30      10      36      15

                DLL Traversal

        25      12      30      10      36      15
```

Time Complexity: O(n) where n is the number of nodes in given Binary Tree. The solution simply does two traversals of all Binary Tree nodes.