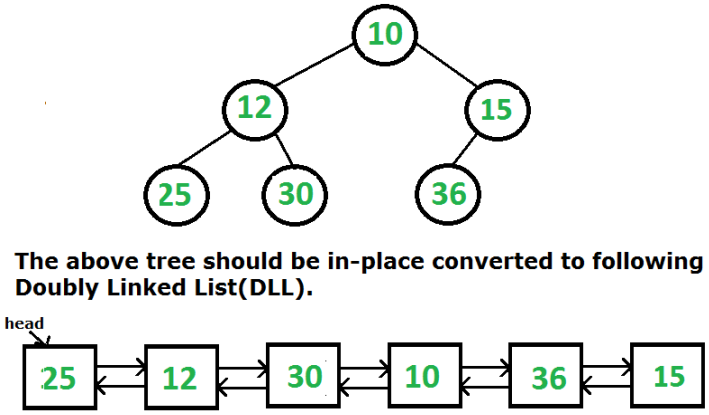
Convert a given Binary Tree to Doubly Linked List | Set 1

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.



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
class Node {

    int data;
    Node left, right;

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

```
}

class BinaryTree {

    static 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(root);

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

Output:

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

This article is compiled by **Ashish Mangla** and reviewed by GeeksforGeeks team. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

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

114 Comments Category: Trees

Related Posts:

- Check if removing an edge can divide a Binary Tree in two halves
- Check sum of Covered and Uncovered nodes of Binary Tree
- Lowest Common Ancestor in a Binary Tree | Set 2 (Using Parent Pointer)
- Construct a Binary Search Tree from given postorder
- BFS vs DFS for Binary Tree
- Maximum difference between node and its ancestor in Binary Tree
- Inorder Non-threaded Binary Tree Traversal without Recursion or Stack
- Check if leaf traversal of two Binary Trees is same?

(Login to Rate and Mark)

3.4 Average Difficulty : **3.4/5.0**
Based on **11** vote(s)

Add to TODO List
Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

prashant jha

a nice exapmle of divide and conquer algorithm
recursively create dll of left and right subtree and combine it with node
and return proper inorder or preorder successor
T(N)=2T(N/2)+0(N)
complexity=nlogn

```
tnode* create_dll(tnode* root,int &cnt)
{
if(!root)
return NULL;
if((!root->lchild)&&(!root->rchild))
return root;
tnode* k1=create_dll(root->lchild,cnt);
tnode* k2=create_dll(root->rchild,cnt);
if(k1)
{
root->lchild=k1;
k1->rchild=root;
}
if(k2)
{
root->rchild=k2;
```

```
k2->lchild=root;
}
if(cnt%2==0)
{
tnode* p=root;
while(p->rchild!=NULL)
{
p=p->rchild;
}
cnt++;
return p;
}
else
{
tnode* p=root;
while(p->lchild!=NULL)
{
p=p->lchild;
}
cnt++;
return p;
}
}
```

danny
I think this question can simply be done by doing Inorder traversal of the binary tree and forming DLL…
Please go through the function which I implemented and Let me know if there is some mistake or Improvement required…

```
void binarytreetoDLL(struct tree *root,struct tree **head)
{
static struct tree *prev=NULL;
struct tree *p=root;
if(p==NULL)
return;
else
{
convert(p->left,head);
p->left=prev;
if((*head)==NULL)
(*head)=p;
else
{ prev->right=p; }
prev=p;
convert(p->right,head);
}
}
```

The head parameter is for the head of DLL and is passed NULL initially….
ganeshprabhu1994
I dont find any need for having a base case in bintree2listutil.
Ravi
Try to tun it without base, it would crash.
ganeshprabhu1994
There is no way that node parameter will get the value of null then why would it crash?
http://opensourceforgeeks.blogspot.in/ Aniket Thakur
Java Code with output —> http://opensourceforgeeks.blogspot.in/2014/01/convert-given-binary-tree-to-doubly.html
wishall

```
//easy version
//C
//in main
struct tnode** prev=NULL;
call BST2DLL(root,prev,&root);
//implementation
void convertBST2DLL(struct tnode* root,struct tnode** prev,struct tnode** head){
if(!root)return ;
convertBST2DLL(root->left,prev,head);//call for left subtree
root->left=(*prev);//set left child to inorder predecessor
if(*prev)
(*prev)->right=root;//set right child of inorder predecessor to root
else
*head=root;//change the root,,this is only executed for the leftmost leaf
*prev=root;//update prev to its inorder successor
convertBST2DLL(root->right,prev,head);//call for right subtree
}
```

GeeksforGeeks
Thanks for suggesting this simple approach. We have published this as a separate post. Please see
http://www.geeksforgeeks.org/convert-given-binary-tree-doubly-linked-list-set-3/
wishall

wgpshashank
If you carefully examine the DLL , you will see all nodes which are in left subtree of root are on left side of root node in DLL and nodes of right subtree are on right side of root node in DLL , same apply for recursively.
So in-order Traversal is the solution , as we traverse the tree in-order, we can modify a node's left pointer to point to its predecessor. We also have to modify the predecessor's right pointer to point to the current node to maintain the doubly linked list behavior.
Here is the pseudo code

```
// We Will Keep Three Variable
```

```
//root: Current tree node
//pre: this pointer should have the address of in-order predecessor of root
//head: this denoted head of final link list
treeToDoublyList(Node root,Node pre, Node head)
{
if (!root) return;
treeToDoublyList(root->left, prev, head);//call left subtree
// current node's left points to previous node
root->left = pre;
if (pre)
pre->right = root;
// previous node's right points to current node
else
head = root;
// if previous is NULL that current node is head
//For right-subtree/parent, current node is in-order predecessor
pre = root;
treeToDoublyList(root->right, pree, head);
}
```

do correct me if anything wrong here ?

samsammy

Java Implementation without extra space-

```
package abc;
public class BinaryTreeToDLLNew {
Node previous;
boolean first=false;
Node head;
public static void main(String[] args) {
BinaryTreeToDLLNew obj= new BinaryTreeToDLLNew();
obj.dummyStart();
}
public void dummyStart()
{
Node root = new Node(2);
root.left = new Node(1);
root.right = new Node(3);
root.left.left=new Node(4);
root.left.right=new Node(5);
root.left.right.left=new Node(6);
root.right.left=new Node(7);
root.right.left.right=new Node(8);
root.right.left.right.left=new Node(9);
fixPtr(root);
traverseLeft(head);
traverseRight(head);
}
public void traverseRight(Node node)
{
while(node!=null)
{
System.out.print(node.data+" ");
node=node.right;
}
System.out.println("");
}
public void traverseLeft(Node node)
{
while(node.right!=null)
node=node.right;
while(node!=null)
{
System.out.print(node.data+" ");
node=node.left;
}
System.out.println("");
}
public void fixPtr(Node node)
{
if(node!=null)
{
fixPtr(node.left);
if(first==false)
{
head=node;
first=true;
previous=node;
}
else
{
previous.right=node;
node.left=previous;
previous=node;
}
```

```
fixPtr(node.right);
}
}
}
```

Mohit Sehgal

I dont think that above solution will work. You are not finding the inOrder successor and predecessor for leaf nodes correctly.

Psycho

We can also store the address of nodes in Inorder traversal in the order of inorder in an array, then scan array and change left pointer to previous element
and right pointer to next element of array. Is it a correct solution?

Jasprit

```
void convert(struct node *head, struct node **start, node **tail)
{
if (head == NULL)
return;
convert(head->left, start, tail);
head->left = *tail; // prev
if(*tail == NULL)
{
*start = head;
}
else
{
(*tail)->right = head; // next
}
*tail = head;
convert(head->right, start, tail);
}
struct node *BTToDLL(struct node *root)
{
struct node *start = NULL, *tail = NULL;
convert(root, &start, &tail);
return start;
}
```

CODED

It is as simple as this

```
node* treeToBst(node* root)
{
static node *pre = NULL;
node *head = NULL;
if(!root)
{
return NULL;
}
if(NULL == (head = treeToBst(root->left)))
{
head = root;
}
if(pre)
{
pre->right = root;
root->left = pre;
}
pre = root;
treeToBst(root->right);
return head;
}
```

Subrahmanyan Sankaran

```
// TreeToDLL.cpp : Defines the entry point for the console application.
#include "stdafx.h"
#include
using namespace std;
struct Node
{
int data;
Node *left;
Node *right;
};
class Tree
{
static int l;
static int flip;
static int leafdepth;
static Node * prev;
static Node * previous;
static int count;
public:
Node *head;
int maxdepth;
Tree()
{
head = NULL;
```

```
maxdepth = 0;
}
void Inorder(Node *node)
{
if(node)
{
Inorder(node->left);
std::cout<<" [ "<data <right);
}
}
void TreetoDLL(Node *node)
{
if(node)
{
TreetoDLL(node->left);
if(previous)
{
previous->right = node;
node->left = previous;
}
if(count == 0)
{
head = node;
count = count + 1;
}
previous = node;
TreetoDLL(node->right);
}
}
void PrintDLL()
{
Node *tmp = head;
while(tmp)
{
std::cout<data;
tmp = tmp->right;
}
}
void InsertNode(Node *node,int data)
{
Node * tmp = node;
if(head == NULL)
{
head = new Node;
head->data = data;
head->left = NULL;
head->right = NULL;
return;
}
else if(tmp)
{
prev = tmp;
if(data > tmp->data)
{
l=0;
tmp = tmp -> right;
InsertNode(tmp,data);
}
else
{
l=1;
InsertNode(tmp->left,data);
}
}
if(l == 1)
{
prev->left = new Node;
prev->left->data = data;
prev->left->left = NULL;
prev->left->right = NULL;
l = -1;
return;
}
else if(l == 0)
{
prev->right = new Node;
prev->right->data = data;
prev->right->left = NULL;
prev->right->right = NULL;
l = -1;
return;
}
}
```

```
};
int Tree::l =0;
int Tree::flip =0;
int Tree::leafdepth =-1;
int Tree::count = 0;
Node * Tree::prev = NULL;
Node * Tree::previous = NULL;
int main(int argc, char* argv[])
{
Tree t;
t.InsertNode(t.head,10);
t.InsertNode(t.head,7);
t.InsertNode(t.head,13);
t.InsertNode(t.head,5);
t.InsertNode(t.head,9);
t.InsertNode(t.head,12);
t.InsertNode(t.head,15);
t.InsertNode(t.head,3);
t.InsertNode(t.head,6);
t.InsertNode(t.head,8);
t.InsertNode(t.head,11);
t.InsertNode(t.head,1);
t.InsertNode(t.head,4);
t.InsertNode(t.head,2);
t.TreetoDLL(t.head);
t.PrintDLL();
return 0;
}
```

smuralimohan
All of the trouble can be circumvented by using the code at: http://cslibrary.stanford.edu/109/TreeListRecursion.html
and then in the final step, convert the circular doubly-linked list to a non-circular doubly-linked list.
syashi

```
/* Paste your code here (You may delete these lines if not writing code) */
```

/*so in this we convert BT to DLL first we set previous pointer.. all the left pointers are used to point to previous node as in inorder traversal!
it does smthing like inorder traversal n sets left pointer to previous node
A
/ \
B C
/ \ / \
D E F so inorder travesal is D B E A F C.. n D will be the head!!*/

```
#include
#include
struct node
{
struct node *right;
struct node *left;
int info;
};
struct node *newNode(int info)
{
struct node *temp=(struct node *)malloc(sizeof(struct node));
temp->info=info;
temp->right=temp->left=NULL;
return temp;
}
void inorder(struct node *root)
{
if(root!=NULL)
{
inorder(root->left);
printf("%d\t",root->info);
inorder(root->right);
}
}
void setprevious(struct node *root)
{
static struct node *previous=NULL;
if(root!=NULL)
{
setprevious(root->left);
root->left=previous;
previous=root;
setprevious(root->right);
}
}
struct node *setnext(struct node *root)
{
static struct node *next=NULL;
while(root && root->right!=NULL)
{
```

```
root=root->right; /*right isiliye coz right wala would be the last element both in the list as well as the inorder traversal*/
}
while(root!=NULL && root->left!=NULL)
{
next=root;
root=root->left;
root->right=next;
}
return(root);
}
struct node *BT2DLL(struct node *root)
{
setprevious(root);
return setnext(root);
}
void printlist(struct node *root)
{
while(root!=NULL)
{
printf("%d\t",root->info);
root=root->right;
}
}
main()
{
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);
root->right->right=newNode(70);
root->left->left->left=newNode(50);
root->left->left->right=newNode(40);
printf("\nInorder traversal\n");
inorder(root);
struct node *head=BT2DLL(root);
printf("\nDLL traversal\n");
printlist(head);
getch();
}
```

CODED
You need to find a way to return the head of the DLL
vikas

```
// i think this will work
struct node* bst_dll(struct node * root)
{
if (!root)
return NULL;
static struct node * head=NULL;
static struct node * tail=NULL;
bst_dll(root->left);
if(!head)
{
head=root;
tail=root;
}
else
{
tail->right=root;
tail=root;
}
bst_dll(root->right);
return head;
}
```

KK
I simply can not understand why we can not do this with inorder.

```
/* Paste your code here (You may delete these lines if not writing code) */
void treeToList(node* root,node*& prev, node*& head)
{
        if(root)
        {
                treeToList(root->left,prev,head);
                if(prev)
                        prev->right = root;
                else
                        head = root;
                root->left= prev;
                prev = root;
                treeToList(root->right,prev,head);
        }
}
```

In the calling function do prev->next = nullptr; (It is a not a circular double linked list)
Otherwise, add pointers between head and prev
Zohreh
The original solution does not solve the problem in O(n). It is of order O(nlogn) in average as it needs to traverse all levels.
// There might be some errors in the code

```
void BTree::TreeToDLL(BTree* n, BTree* &head, BTree* &tail) {
        BTree* head1;
        BTree* head2;
        BTree* tail1;
        BTree* tail2;

        if (n == NULL) {

                return;
        }

        head1 = head;
        tail1 = tail;
        head2 = head;
        tail2 = tail;

        if (n->left == NULL) {
                 head1 = n;
                 n->left = NULL;
        } else {
                TreeToDLL(n->left, head1, tail1);
                tail1->right = n;
                n->left = tail1;
        }

        if (n->right == NULL) {
                tail2 = n;
                n->right = NULL;
        } else {
                TreeToDLL(n->right, head2, tail2);
                n->right = head2;
                head2->left = n;
        }

        head = head1;
        tail = tail2;
}
```

Vikrant
What is the use of converting a binary tree into doubly Linked List ?
code_jazz

```
/************** headers ***************/
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>


/************ data types *****************/
struct node
{
    int data;
    struct node *left, *right;
};
struct LL{
    int data;
    struct LL *prev,*next;
};


/************** prototypes ***************/
int isNull(struct node *);
struct node * place_data(int,struct node *);
void convert(struct node *);
void extract(struct node *);
void connect(int,struct LL *);
void showlist(struct LL *);




/**********
function places data at appropriate place - just for tree construction
***********/
```

```
struct node * place_data(int data_rec,struct node *root){

    if( isNull(root)    ){
        root            =   (struct node *)malloc(sizeof(struct node));
        root->data      =   data_rec    ;
        root->left    =   NULL        ;
        root->right    =   NULL        ;
        return  root;
        }   /**** end of root allocation ****/
    else{

            if(root->data   >   data_rec){  //send left
                root->left    =   place_data(data_rec,root->left);
                }
            else if(root->data  <   data_rec){//send right
                root->right    =   place_data(data_rec,root->right);
                }
            else{
                printf("\ndata cannot be equal to the previous data/key\nprogram is exiting..");
                exit(1);    }

           return root;
        }   /**** end of else and place option ****/
}

/********
simply helper function
*********/
int isNull(struct node *ptr){
    if(ptr  ==  NULL)
        return 1;
        return 0;
}



/**** globals for extract function ****/
int flip = 0;
struct LL *mainNode;
/*********/

/**********
Extracts data and places it in the dll
***********/
void extract(struct node *temp){
    int data = temp->data;
    struct LL * temp1;
    if(flip == 0){      /******* <====== indicates that the DLL has yet to be created ***/

        temp1 = (struct LL *)malloc(sizeof(struct LL));
        temp1->data = data;
        temp1->next = NULL;
        temp1->prev = NULL;
        mainNode=temp1;
        flip=1;     /***** flip it ******/
    }

    else if(flip = 1){ /****** <======= indicates that the root node of dll
                                        has already been created *******/
        connect(data,mainNode);/***** so simply connect it now ********/
    }

} //void extract ends


/***********
called from main - subdivieds the work with inorder traversal
************/
void convert(struct node *root){
    if(root){
        convert(root->left);    /*** <=== inorder part *****/
        extract(root);
        convert(root->right);   /*** <=== inorder part *****/
    }
    else return;
}//void convert ends



/************
simply connects two nodes of a DLL
*************/
void connect(int data,struct LL * temp){
```

```
    struct LL * temp1 = (struct LL *)malloc(sizeof(struct LL ));
    struct LL * behind;
    while(temp!=NULL){
        behind = temp;
        temp=temp->next;
    }
    temp1->data = data;
    temp1->next = NULL;
    temp1->prev = behind;
    behind->next = temp1;


}//void connect ends




/***************
function to show list contents
***************/
void showlist(struct LL * root){
    struct LL * last;
    printf("\nShowing foreward : ");

    while(root!=NULL){
        printf("%d\t",root->data);
        last=root;
        root=root->next;
    }

    printf("\nShowing reverse : ");
    while(last!=NULL){
        printf("%d\t",last->data);
        last=last->prev;
    }
}




/************************ MAIN ************************/
int main()
{
    // Let us construct the tree given in the above diagram
    int arr[8]={20,8,4,12,10,14,22,25};
    int i;

    struct node *root=NULL;
    /************* tree placemnt *************/
    for(i=0;i<8;i++){
        //printf("%d\n",i);
        root    =   place_data(arr[i],root);
        }
    /*****************************************/
    convert(root);
    printf("\nshowing list : ");
    showlist(mainNode);
    return 0;
}//main ends
```

Anuj Prajapati
we are not allowed to create a new list,
we have to convert the given tree itself into doubly linked list..
abhishek08aug
Intelligent 😀
mukesh2009mit

```
/*Converting a Binary Search Tree to Doubley Linked List */
/*TIME COMPLEXITY=O(N)
STACK SPACE=O(N)*/

#include<stdio.h>
#include<conio.h>
struct node
{
int x;
struct node *lc;
struct node *rc;
};
```

```
void mklist(struct node *p,struct node** lm,struct node** rm)
{
struct node* lm1,*lm2,*rm1,*rm2;
lm1=lm2=rm1=rm2=NULL;
if(p->lc==NULL&&p->rc==NULL)
{
*lm=*rm=p; return;
}
if(p->lc!=NULL)
{
mklist(p->lc,&lm1,&rm1);
}
if(p->rc!=NULL)
{
mklist(p->rc,&lm2,&rm2);
}
p->lc=rm1;
if(rm1!=NULL) rm1->rc=p;
p->rc=lm2;
if(lm2!=NULL) lm2->lc=p;
if(lm1==NULL)
        *lm=p;
else
        *lm=lm1;
if(rm2==NULL)
        *rm=p;
else
       *rm=rm2;
}
void main()
{
clrscr();
struct node a1,a2,a3,a4,a5,a6,a7;
a1.x=1;
a2.x=2;
a3.x=3;
a4.x=4;
a5.x=5;
a6.x=6;
a7.x=7;
a1.lc=a1.rc=NULL;
a2.lc=&a1;a2.rc=NULL;
a3.lc=&a2;a3.rc=&a4;
a4.lc=a4.rc=NULL;
a5.lc=&a3;a5.rc=&a7;
a6.lc=a6.rc=NULL;
a7.lc=&a6,a7.rc=NULL;
struct node* lm,*rm;
lm=rm=NULL;
mklist(&a5,&lm,&rm);
while(lm!=NULL)
{
printf(" %d ",lm->x);
lm=lm->rc;
}
getch();
}
```

Code1101

```
class Tree {
    Node root;

    Tree(Node root) {
        this.root = root;
    }

    public void postOrder() {
        postOrder(root);
    }

    private void postOrder(Node node) {
        if(node == null) return;

        postOrder(node.rightChild);
        System.out.println(node.data);
        postOrder(node.leftChild);
    }

    public Node treeToDLL(Node node) {
        if (node == null) return null;

        Node data = new Node(null, null, node.data);
```

```
        if (node.leftChild != null) {
            Node leftChild = treeToDLL(node.leftChild);
            data.leftChild = leftChild;
            leftChild.rightChild = data;
        }

        if (node.rightChild != null) {
            Node rightChild = treeToDLL(node.rightChild);
            data.rightChild = rightChild;
            rightChild.leftChild = data;
        }
        if (data.rightChild != null) {
            return data.rightChild;
        } else if(data.leftChild != null) {
            return data.leftChild;
        } else return data;

    }
}

class Node {
    Node leftChild;
    Node rightChild;
    int data;

    Node(Node leftChild, Node rightChild, int data) {
        this.leftChild = leftChild;
        this.rightChild = rightChild;
        this.data = data;
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (!(o instanceof Node)) return false;

        Node node = (Node) o;

        if (data != node.data) return false;
        if (leftChild != null ? !leftChild.equals(node.leftChild) : node.leftChild != null) return false;
        if (rightChild != null ? !rightChild.equals(node.rightChild) : node.rightChild != null) return false;

        return true;
    }

    @Override
    public int hashCode() {
        int result = leftChild != null ? leftChild.hashCode() : 0;
        result = 31 * result + (rightChild != null ? rightChild.hashCode() : 0);
        result = 31 * result + data;
        return result;
    }
}
```

Ravikesh

```
void inorder(treeNode* root,treeNode* prev)/*initially prev will be NULL*/
{
treeNode* headDll = NULL;
if(root)
{
inorder(root->left,prev);
if(prev == NULL)
headDll = root;
else
{
prev->right=root;
root->left = prev;
}
prev = root;
inorder(root->right,prev);
}
}
```

Point to be noted that headDll prev will be NULL automatically and similarly last element right will be NULL/

naren596

reverse postorder traversal gives successor for next level recursion. I guess this will do the job in O(N)

```
public class ConvertBTtoDLL
{
node head=null;
public node convert_helper(node root)
{
convert(root);
return head;
```

```
}
public node convert(node root)
{
if(root!=null)
{
convert(root.right);
root.right=head;
if(head!=null) head.left=root;
head=root;
convert(root.left);
}
return root;
}
}
```

Praveen
+1
Praveen

```
package Btree;

import java.util.LinkedList;
import java.util.Queue;

public class BtreeFromDoublyLinked {

        public Bnode ConvertToDoubly(Bnode root){
                Queue<Bnode> q = new LinkedList<Bnode>();
                if(root==null)
                return null;
                q.add(root);
                Bnode start = root;
                Bnode lbnode=null;
                Bnode current=null;
                while(q.peek()!=null){
                        if(q.peek().left!=null){
                                q.add(q.peek().left);
                        }
                        if(q.peek().right!=null){
                                q.add(q.peek().right);
                        }
                        lbnode= current;
                        current = q.poll();
                        current.left=lbnode;
                        current.right = q.peek();
                }
        return start;
        }

        public void printList(Bnode head){
                while(head !=null){
                        System.out.println(head.data+" - ");
                        head = head.right;
                }
        }

        public static void main(String[] args) {
                Bnode b = new Bnode(1);
                b.left=new Bnode(2);
                b.right=new Bnode(3);
                b.left.left=new Bnode(4);
                b.left.right=new Bnode(5);
                b.right.left=new Bnode(6);
                BtreeFromDoublyLinked btol = new BtreeFromDoublyLinked();
                Bnode head = btol.ConvertToDoubly(b);
                btol.printList(head);
                }
}
```

http://effprog.blogspot.com Sambasiva

```
Node * Bst2Dbl(Node *t, Node **prev) {

        if (t == NULL) {
                return NULL;
        }

        Node *root = Bst2Dbl(t->left, prev);

        if (*prev) {
                (*prev)->right = t;
                t->left = *prev;
```

```
        }

        *prev = t;

        Bst2Dbl(t->right, prev);

        return root ? root : t;
}
```

Ramesh.Mxian
We can do it with O(N) easily. We will process the Nodes in inorder traversal as follows.
1. If left subtree exists then Create a DLL for the left subtree and return the head and tail of that linked list.
2. If right subtree exists then Create a DLL for the right subtree and return the head and tail of that DLL.
3. Insert the current node between tail of the DLL from left subtree and head of the right subtree
4. Return new head as the head from left subtree and tail as tail from right subtree.
We can make use of the same Node structure to return head and tail of DLL by using left as head and right as tail.
Following is the procedure
Node CreateDLL(Node node){
Node Result, LDLL,RDLL.
// If there is no left subtree then current node will be head of the resulting DLL
// If there is no right subtree then current node will be tail of the resulting DLL
Result.right = node
Result.left = node;
if(node.left != null){
LDLL= CreateDLL(node.left)
node.left = LDLL.right
LDLL.right.right = node
Result.left = LDLL.left
}
if(node.right != null){
RDLL = CreateDLL(node.right)
node.right = RDLL.left
RDLL.left.left = node
Result.right = RDLL.right
}
return Result.
}
mohitk
True.
had the same in my design, but just returning the head…
However, can also use the principle as given above by Ramesh.

```
public Node<T> tree2DLL(Node<T> root) {
                if (root == null) return null;
                return tree2DLLHelper(root, root, root);
        }

        /** Params: All are initially equal to node.
         *  Returns: Head of the new DLL to which the binary tree is transformed. */
        private Node<T> tree2DLLHelper(Node<T> node, Node<T> head, Node<T> tail) {
                // Left-subtree
                if (node.left != null) {
                        Node<T> headL, tailL;
                        headL = tailL = node.left;

                        // Update head of the combined DLL
                        head = tree2DLLHelper(node.left, headL, tailL);

                        // Connect the left DLL and the node
                        node.left = tailL;
                        tailL.right = node;
                }

                // Right-subtree
                if (node.right != null) {
                        Node<T> headR, tailR;
                        headR = tailR = node.right;

                        // Connect the node and right DLL
                        node.right = tree2DLLHelper(node.right, headR, tailR);
                        headR.left = node;

                        // Update the tail of the combined DLL
                        tail = tailR;
                }
                return head;
        }
```

sri

```
void modify_to_DLL(node*p, node*&prev,node*&head)
{
if(!p)return ;
modify_to_DLL(p->left,prev,head);
```

```
p->left=prev;
if(prev)
prev->right=p;
else
head=p;
node* right=p->right;
head->left=p;
p->right=head;
prev=p;//updating the prev node
modify_to_DLL(right,prev,head);
}
node* prev=new node;
node* head=new node;
prev=head=NULL;
modify_to_DLL(root,prev,head)

/* Paste your code here (You may delete these lines if not writing code) */
```

surabhiremix
Can we go the either way.i mean from doubly linked list to binary tree?

```
/* Paste your code here (You may delete these lines if not writing code) */
```

GeeksforGeeks
Please see following post:
http://www.geeksforgeeks.org/in-place-conversion-of-sorted-dll-to-balanced-bst/
Max
Segfault in bintree2list when root is null.
GeeksforGeeks
@Max: Thanks for pointing this out. We have added a condition in bintree2list() to handle NULL.
Himanshu Shivnani
i don't understand why there was segmentation fault when root is NULL because earlier we were returning NULL when root is NULL in function bintree2list(). Because in main function, we are passing 'head' to printList(), which checks for NULL before accessing 'data'.

```
/* Paste your code here (You may delete these lines if not writing code) */
```

Purushotham
Above suggested solution is of O(NlogN) time complexity. We can achieve the same in O(N) with a successor node having it as a class variable. Below is the code for this approach:

```
public class Tree2DLL {
public Node root;
private Node successor = null;
private class Node {
public int val;
public Node right;
public Node left;
}
public Node TreetoDLL(Node root){
if (root.left == null && root.right == null)
return null;
if (root.right != null)
TreetoDLL(root.right);
if(successor != null) successor.left = root;
root.right = successor;
successor = root;
if (root.left != null)
TreetoDLL(root.left);
return successor;
}
}

/* Paste your code here (You may delete these lines if not writing code) */
```