Trees : Level 2
Lesson 3

Flatten binary tree to linked list

Given a binary tree, flatten it into linked list in-place. Usage of auxiliary data structure is not allowed. After flattening, left of each node should point to NULL and right should contain next node in preorder.

Example 1:

```
Input :
        1
      /   \
     2     5
    / \     \
   3   4     6

Output :
1 2 3 4 5 6

Explanation:
After flattening, the tree looks
like this
   1
    \
     2
      \
       3
        \
         4
          \
           5
            \
             6
```

Here, left of each node points to NULL and right contains the next node in preorder.The inorder traversal of this flattened tree is 1 2 3 4 5 6.

Example 2:

Input :

```
       1
      / \
     3   4
        /
       2
        \
         5
```

Output :
1 3 4 2 5

Explanation :
After flattening, the tree looks like this

```
    1
     \
      3
       \
        4
         \
          2
           \
            5
```

Here, left of each node points to NULL and right contains the next node in preorder.The inorder traversal of this flattened tree is 1 3 4 2 5.

Expected Time Complexity: O(n)
Expected Auxiliary Space: O(1)

Constraints :
1<=n<=10^5

```
void tree(TreeNode* root)
{
    if(root)
    {
        TreeNode *r=root->right;
        TreeNode* l=root->left;
        TreeNode* k=root;
        root->left=NULL;
        root->right=l;
        tree(l);
        while(k->right)
            k=k->right;
        k->right=r;
        tree(r);
    }
}
class Solution {
public:
    void flatten(TreeNode* root) {
        tree(root);
    }
```

Populate Inorder Successor for all nodes

Given a Binary Tree, write a function to populate the next pointer for all nodes. The next pointer for every node should be set to point to inorder successor.

Example 1:

```
Input:
       10
    /  \
   8    12
  /
  3
```

Output: 3->8 8->10 10->12 12->-1
Explanation: The inorder of the above tree is : 3 8 10 12. So the next pointer of node 3 is pointing to 8 , next pointer of 8 is pointing to 10 and so on.And next pointer of 12 is pointing to -1 as there is no inorder successor of 12.

Example 2:

```
Input:
       1
    /  \
   2    3
```
Output: 2->1 1->3 3->-1

Expected Time Complexity: O(N)

Expected Auxiliary Space: O(N)

Constraints:

1<=n<=10^5

1<=data of the node<=10^5

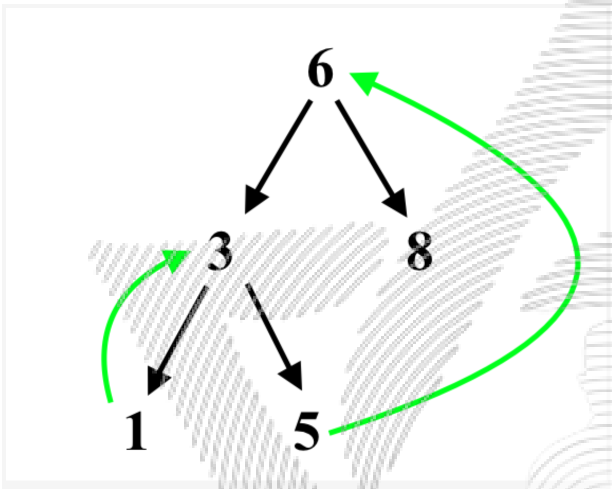
```
void fun(node*p , node**nextn)
{
   if(p==NULL)
      return;
   if(p)
   {
      fun(p->right,nextn);
      p->next=*nextn;
      *nextn=p;
      fun(p->left,nextn);
   }
}
void populateNext(struct node* p)
{
   node*nextn=NULL;
   fun(p,&nextn);

// Your code goes here

}
```

Clone a Binary Tree

Given a special binary tree having random pointers along with the usual left and right pointers. Clone the given tree.

Example 1:

Input:



Output: 1
Explanation: The tree was cloned successfully.

Note: The output is 1 if the tree is cloned successfully. Otherwise output is 0.

Expected Time Complexity: O(N).

Expected Auxiliary Space: O(N).

Constraints:

1 <=Number of nodes<= 100

1 <=Data of a node<= 1000

```
Node* rightandleft(Node*root,unordered_map<Node*,Node*>&um)
{
  if(root==NULL)
     return NULL;
  Node *clone = new Node(root->data);
  um[root]=clone;

  Node*leftn=rightandleft(root->left,um);
  clone->left=leftn;
  Node*rightn=rightandleft(root->right,um);
  clone->right=rightn;
  return clone;
}

void randomn(Node*root,Node*tree,unordered_map<Node*,Node*>&um)
{
  if(root==NULL || tree==NULL)
     return;
  root->random=um[tree->random];
  randomn(root->left,tree->left,um);
  randomn(root->right,tree->right,um);
}
Node* cloneTree(Node* tree)
{
  unordered_map<Node*,Node*>um;
  Node*root=rightandleft(tree,um);
  randomn(root,tree,um);
  return root;
 //Your code here
}
```

Leaves to DLL

Given a Binary Tree of size N, extract all its leaf nodes to form a Doubly Link List starting from the leftmost leaf. Modify the original tree to make the DLL thus removing the leaf nodes from the tree. Consider the left and right pointers of the tree to be the previous and next pointer of the DLL respectively.

Example 1:

Input :

```
     1
   /  \
  2    3
 / \  / \
4  5 6   7
```

Output:

Modified Tree :

```
     1
   /  \
  2    3
```

Doubly Link List :

4 <-> 5 <-> 6 <-> 7

Example 2:

Input :

```
     1
   /  \
  2    3
 / \
4  5
```

Output:

```
Modified Tree :
     1
    /
   2

Doubly Link List :
4 <-> 5 <-> 3
```

Note:

The generated output will contain the inorder traversal of the modified tree, the DLL from left to right and the DLL from right to left.

Expected Time Complexity: O(N)

Expected Auxiliary Space: O(height of tree)

Constraints:

1 ≤ N ≤ 10^4

```
Node* dll(Node*root , Node**head)
{
   if(root==NULL)
       return NULL;
   if(!root->left && !root->right)
   {
       root->right=*head;
       if(*head!=NULL)
           (*head)->left=root;
       *head=root;
       return NULL;
   }
   root->right=dll(root->right,head);
   root->left=dll(root->left,head);


   return root;
}
// return the head of the DLL and remove those node from the tree as well.
Node * convertToDLL(Node *root)
{
   Node*head=NULL;
   dll(root,&head);
   return head;
   // add code here.
}
```

Connect Nodes at Same Level

Given a binary tree, connect the nodes that are at the same level. You'll be given an additional **nextRight** pointer for the same.

Initially, all the **nextRight** pointers point to **garbage** values. **Your function** should set these pointers to point next right for each node.

```
        10                      10 ------> NULL

        / \                    /          \

        3   5           =>       3 ------> 5 --------> NULL

        / \     \                 /  \            \

  4   1   2           4 --> 1 -----> 2 -------> NULL
```

Example 1:

Input:

```
   3
  /  \
 1    2
```

Output:

```
3 1 2
1 3 2
```

Example 2:

Input:

```
     10
    /   \
  20   30
  /  \
 40  60
```

Output:

```
10 20 30 40 60
40 20 60 10 30
```

Expected Time Complexity: O(N).

Expected Auxiliary Space: O(N).

Constraints:

1 ≤ Number of nodes ≤ 105

0 ≤ Data of a node ≤ 105

```
 class Solution
 {
   public:
   //Function to connect nodes at same level.
   void connect(Node *root)
   {
      if(root==NULL)
         return;
      if(!root->left && !root->right)
      {
         root->nextRight=NULL;
      }
      queue<Node*>q;
      q.push(root);
      q.push(NULL);
      while(!q.empty())
      {
         Node*temp=q.front();
         q.pop();
         if(temp!=NULL)
         {
            temp->nextRight=q.front();
```

```
                if(temp->left)
                    q.push(temp->left);
                if(temp->right)
                    q.push(temp->right);
            }
            else if(!q.empty())
                q.push(NULL);
        }
        // Your Code Here
    }

};
```

Binary Tree to CDLL

Given a Binary Tree of **N** edges. The task is to convert this to a Circular Doubly Linked List(**CDLL**) in-place. The left and right pointers in nodes are to be used as previous and next pointers respectively in converted CDLL. The order of nodes in CDLL 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 CDLL.

Example 1:

Input:

```
    1
   /  \
  3    2
```

Output:

```
3 1 2
2 1 3
```

Example 2:

Input:

```
    10
   /   \
  20    30
 /  \
40  60
```

Output:

```
40 20 60 10 30
30 10 60 20 40
```

Constraints:

1 <= N <= 10^3

1 <= Data of a node <= 10^4

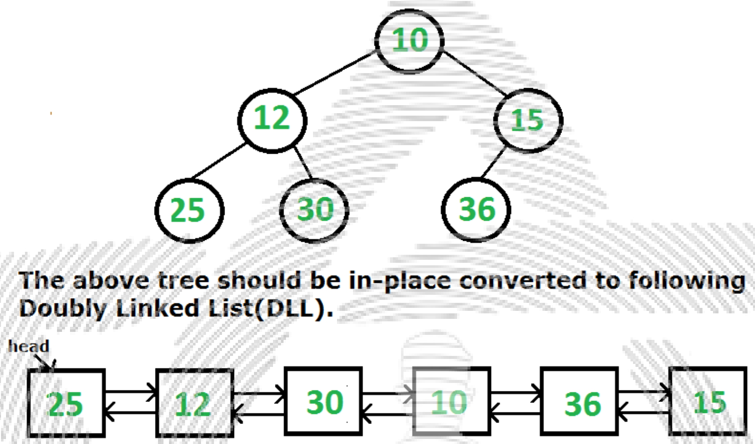
Expected time complexity: O(N)

Expected auxiliary space: O(h) , where h = height of tree

```
void inorder(Node *root, Node **head)
{
   if(root)
   {
      inorder(root->right, head);
      root->right=*head;
      if(*head)
         (*head)->left=root;
      *head=root;
      inorder(root->left, head);
   }
}

Node *bTreeToCList(Node *root)
{
   //add code here.
   Node *head=0;
   inorder(root ,&head);
   Node *r=head;
   while(r->right)
      r=r->right;
   r->right=head;
   head->left=r;
   return head;
}
```

Binary Tree to DLL

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 (leftmost node in BT) must be the head node of the DLL.



The above tree should be in-place converted to following Doubly Linked List(DLL).

Example 1:

Input:

```
   1
  / \
 3   2
```

Output:

DLL would be 3<=>1<=>2

Example 2:

Input:

```
       10
      /   \
   20   30
  /   \
 40   60
```

DLL would be 40<=>20<=>60<=>10<=>30.

Expected Time Complexity: O(N).

Expected Auxiliary Space: O(H).

Note: H is the height of the tree and this space is used implicitly for recursion stack.

Constraints:

1 <= Number of nodes <= 105

1 <= Data of a node <= 105

```
void Inorder(Node* (&head), Node *root)
{
   if(!root)
       return;
   Inorder(head,root->right);
   root->right=head;
   if(head)
       head->left=root;
   head=root;
   Inorder(head,root->left);
}
class Solution
{
   public:
    Node * bToDLL(Node *root)
   {
       // your code here
       Node* head=NULL;
       Inorder(head,root);
       return head;
   }
};
```