06.11.2020-PRACTICE SESSION CODES

NUMBER OF ROOT TO LEAF PATHS:

```
void count(Node* root,map<int,int> &mp,int pathLen)
{
    if(!root)   return ;
    if(!root->right && !root->left)
    {
        mp[pathLen]++;
        return ;
    }
    count(root->left,mp,pathLen+1);
    count(root->right,mp,pathLen+1);
}
void pathCounts(Node *root)
{
    map<int,int> mp;
    count(root,mp,1);
    for(auto i=mp.begin();i!=mp.end();i++)
        cout<<i->first<<" "<<i->second<<" $";
}
```

CONSTRUCT TREE FROM LINKED LIST:

```
void convert(Node *head, TreeNode *&root)
{
    queue<TreeNode *> q;
    root = new TreeNode(head->data);
    Node *temp = head->next;
    q.push(root);
    while (temp)
    {
        TreeNode *cTNode = q.front();
        q.pop();
        cTNode->left = new TreeNode(temp->data);
        q.push(cTNode->left);
        if (!(temp = temp->next))
            break;
        cTNode->right = new TreeNode(temp->data);
        q.push(cTNode->right);
        temp = temp->next;
    }
}
```

COUNT NUMBER OF BST NODES IN A GIVEN RANGE:

```
int getCountOfNode(Node *root, int l, int h)
{
    if (!root)
        return 0;
    if (root->data >= l && root->data <= h)
        return 1 + (getCountOfNode(root->left, l, h) + getCountOfNode(root-
>right, l, h));
    if (root->data > h)
        return getCountOfNode(root->left, l, h);
    if (root->data < l)
        return getCountOfNode(root->right, l, h);
}
```

FOLDABLE BINARY TREE:

```
#include <bits/stdc++.h>
using namespace std;

struct Node
{
    int data;
    Node *left;
    Node *right;

    Node(int val)
    {
        data = val;
        left = right = NULL;
    }
};

Node *buildTree(string str)
{
    if (str.length() == 0 || str[0] == 'N')
        return NULL;
    vector<string> ip;
    istringstream iss(str);
    for (string str; iss >> str;)
        ip.push_back(str);
    Node *root = new Node(stoi(ip[0]));
    queue<Node *> queue;
    queue.push(root);
    int i = 1;
    while (!queue.empty() && i < ip.size())
    {
        Node *currNode = queue.front();
```

```
        queue.pop();
        string currVal = ip[i];
        if (currVal != "N")
        {
            currNode->left = new Node(stoi(currVal));
            queue.push(currNode->left);
        }
        i++;
        if (i >= ip.size())
            break;
        currVal = ip[i];
        if (currVal != "N")
        {
            currNode->right = new Node(stoi(currVal));
            queue.push(currNode->right);
        }
        i++;
    }

    return root;
}

bool isSame(Node *l, Node *r)
{
    if (!l && !r)
        return true;
    return (l && r) && isSame(l->right, r->left) && isSame(l->left, r->right);
}
bool IsFoldable(Node *root)
{
    if (!root)
        return true;
    return isSame(root->left, root->right);
}

int main()
{
    string treeString;
    getline(cin, treeString);
    Node *root = buildTree(treeString);
    if (IsFoldable(root))
    {
        cout << "Yes\n";
    }
    else
    {
       cout << "No\n";
    }
    return 0;
}
```

SUBTREES WITH GIVEN SUM:

```
#include <bits/stdc++.h>
using namespace std;

struct Node
{
    int data;
    Node *left;
    Node *right;
};

Node *newNode(int val)
{
    Node *temp = new Node;
    temp->data = val;
    temp->left = NULL;
    temp->right = NULL;

    return temp;
}

Node *buildTree(string str)
{
    if (str.length() == 0 || str[0] == 'N')
        return NULL;
    vector<string> ip;
    istringstream iss(str);
    for (string str; iss >> str;)
        ip.push_back(str);
    Node *root = newNode(stoi(ip[0]));
    queue<Node *> queue;
    queue.push(root);
    int i = 1;
    while (!queue.empty() && i < ip.size())
    {
        Node *currNode = queue.front();
        queue.pop();
        string currVal = ip[i];
        if (currVal != "N")
        {
            currNode->left = newNode(stoi(currVal));
            queue.push(currNode->left);
        }
        i++;
        if (i >= ip.size())
            break;
        currVal = ip[i];
        if (currVal != "N")
        {
            currNode->right = newNode(stoi(currVal));
```

```
            queue.push(currNode->right);
        }
        i++;
    }

    return root;
}

int subtreeSum(Node *root, int &count, int X)
{
    if (!root)
        return 0;
    int sum = root->data + subtreeSum(root->left, count, X) + subtreeSum(root-
>right, count, X);
    if (sum == X)
        count++;
    return sum;
}
int countSubtreesWithSumX(Node *root, int X)
{
    int cnt = 0;
    subtreeSum(root, cnt, X);
    return cnt;
}

int main()
{
    int t;
    cin >> t;
    getchar();
    while (t--)
    {
        string s;
        getline(cin, s);
        Node *root = buildTree(s);

        int x;
        cin >> x;
        getchar();
        cout << countSubtreesWithSumX(root, x) << endl;
    }
    return 0;
}
```

LEAF NODES TO DLL:

```
#include <bits/stdc++.h>
using namespace std;

struct Node
{
    int data;
    Node *left;
    Node *right;

    Node(int val)
    {
        data = val;
        left = right = NULL;
    }
};

Node *buildTree(string str)
{
    if (str.length() == 0 || str[0] == 'N')
        return NULL;
    vector<string> ip;
    istringstream iss(str);
    for (string str; iss >> str;)
        ip.push_back(str);
    Node *root = new Node(stoi(ip[0]));
    queue<Node *> queue;
    queue.push(root);
    int i = 1;
    while (!queue.empty() && i < ip.size())
    {
        Node *currNode = queue.front();
        queue.pop();
        string currVal = ip[i];
        if (currVal != "N")
        {
            currNode->left = new Node(stoi(currVal));
            queue.push(currNode->left);
        }
        i++;
        if (i >= ip.size())
            break;
        currVal = ip[i];
        if (currVal != "N")
        {
            currNode->right = new Node(stoi(currVal));
            queue.push(currNode->right);
        }
        i++;
    }
```

```
    return root;
}

void inOrder(Node *root)
{
    if (!root)
        return;
    inOrder(root->left);
    cout << root->data << " ";
    inOrder(root->right);
}

Node *convertToDLL(Node *root)
{
    if (!root)
        return NULL;
    Node *head = NULL, *walk;
    queue<Node *> q;
    q.push(root);
    while (!q.empty())
    {
        Node *temp = q.front();
        q.pop();
        if (!temp->left && !temp->right)
        {
            if (!head)
            {
                head = new Node(temp->data);
                walk = head;
            }
            else
            {
                walk->right = new Node(temp->data);
                walk->right->left = walk;
                walk = walk->right;
            }
            continue;
        }
        if (temp->left)
        {
            q.push(temp->left);
            if (!temp->left->left && !temp->left->right)
            {
                temp->left = NULL;
            }
        }

        if (temp->right)
        {
            q.push(temp->right);
```

```
            if (!temp->right->left && !temp->right->right)
            {
                temp->right = NULL;
            }
        }
    }
    return head;
}
int main()
{
    int tc;
    cin >> tc;
    while (tc--)
    {
        string treeString = "1 2 3 4";
        //getline(cin, treeString);
        Node *root = buildTree(treeString);
        Node *head = convertToDLL(root);
        inOrder(root);
        cout << "\n";
        Node *curr = head, *last = head;
        while (curr)
        {
            cout << curr->data << " ";
            last = curr;
            curr = curr->right;
        }
        cout << "\n";
        curr = last;
        while (curr)
        {
            cout << curr->data << " ";
            curr = curr->left;
        }
        cout << "\n";
    }
    return 0;
}
```