LEARN DSA WITH C++

WEEK :: 09

LEARN DSA WITH C++

COURSE INSTRUCTOR BY
ROHIT NEGI

Check Profile

MADE BY-
PRADUM SINGHA

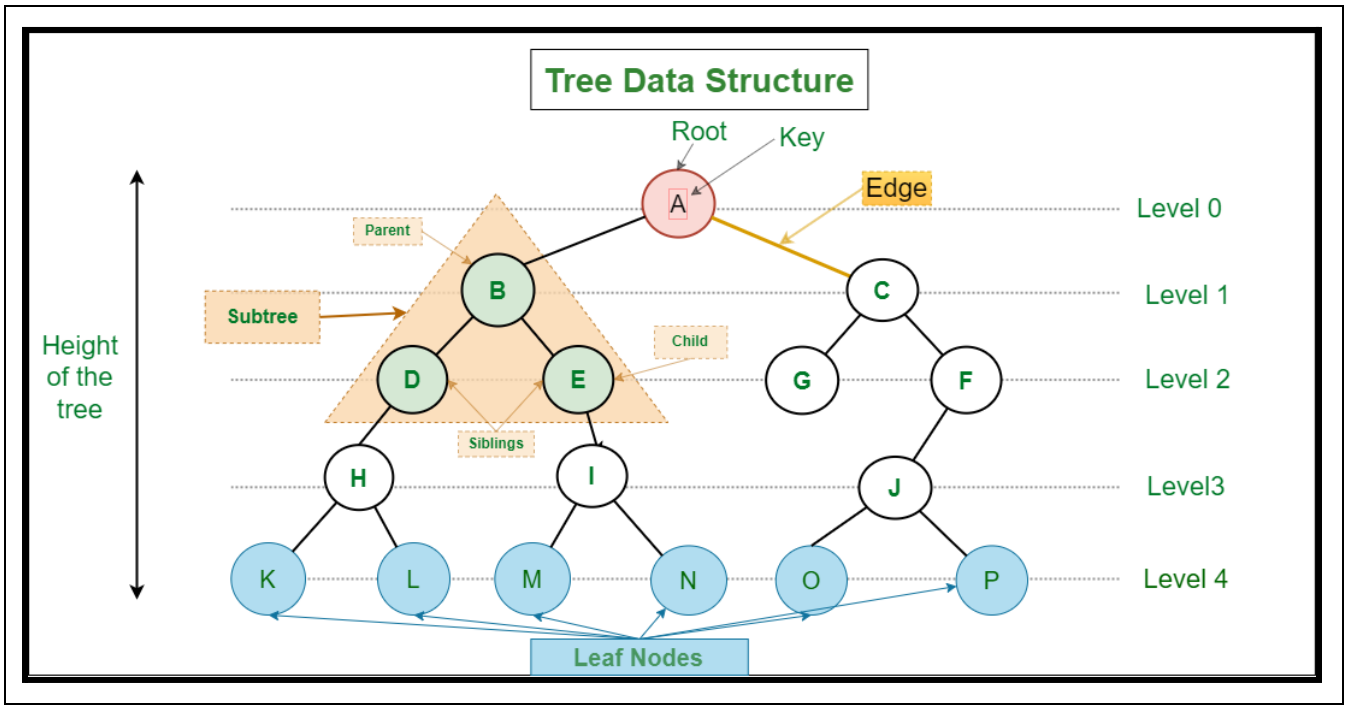
My profile

LEARN DSA WITH C++

WEEK :: 09 **DAY: 01** **DATE: 13-06-2023**

TREES

A tree is also one of the data structures that represent hierarchical data.

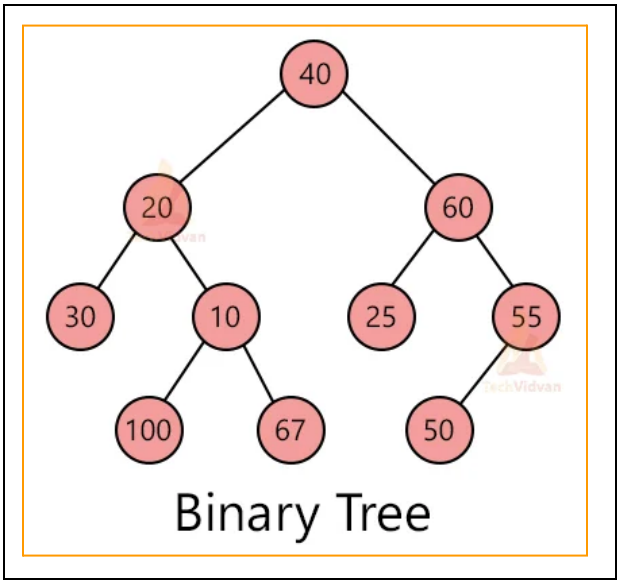


No. of Node : N

No of edges : N-1

Binary Tree

Max. node : 2 only



Create Node:

```
class Node
{
   public:
   int Dta;
   Node* left;
   Node* right;

   Node (int value)
   {
      Data =value;
      left = NULL;
      right = NULL;
    }
}
```

Build Tree :

```
Binary Tree( )
{
  int x;
  cin >>x;
  if(x==-1)
  return  NULL;
 Node* root = new Node(x);
 Binary Tree (root ->left)
 Binary Tree (root ->right)
 return root;
}
```

Print Value of tree::

Inorder Traversal	Post-order Traversal	Pre-order Traversal
L N R	L R N	N L R

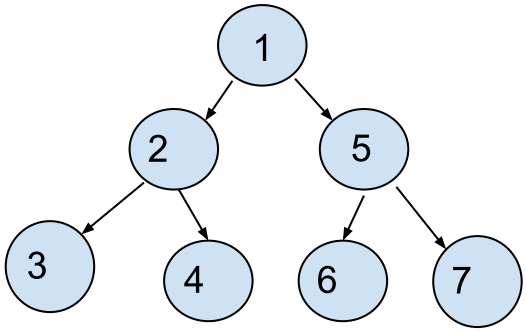
Create Binary Tree::

```
#include<iostream>
using namespace std;
class Node
{
    public:
```

```
    int data;
    Node *left;
    Node *right;

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

void Inorder(Node *root)
{
    if(!root)
    return;

    Inorder(root->left);
    cout<<root->data<<" ";
    Inorder(root->right);
    return;
}

void Postorder(Node *root)
{
    if(!root)
    return;

    Postorder(root->left);
    Postorder(root->right);
    cout<<root->data<<" ";
    return;
}

void Preorder(Node *root)
{
    if(!root)
    return;

    cout<<root->data<<" ";
    Preorder(root->left);
    Postorder(root->right);
    return;
}

Node* BinaryTree()
{
    int x;
    cout<<"Enter the value: ";
    cin>>x;
    if(x==-1)
    return NULL;
```

```
    Node *root = new Node(x);
    cout<<"Enter the left child of "<<x<<"\n";
    root->left = BinaryTree();
    cout<<"Enter the right child of "<<x<<"\n";
    root->right = BinaryTree();

    return root;
}
int main()
{
    Node *root = BinaryTree();
    cout<<"Inorder Traversal: ";
    Inorder(root);
    cout<<endl;
    cout<<"Postorder Traversal: ";
    Postorder(root);
    cout<<endl;
    cout<<"Preorder Traversal: ";
    Preorder(root);
    cout<<endl;

return 0;
};
```

Level Order Traversal ::



There are uses to solve Queue not Recursion.

```
queue <Node*>q;
q.push(root);
while(!q.empty())
{
   Node* temp = q.front();
        q.pop();
   cout<<temp->data)
   if(temp->left)
   q.push (temp ->left)
   if(temp->right)
```

```
    q.push(temp ->right)
}
```

Inorder Traversal << GeeksforGeek>>

```
class Solution {
  public:
  void InorderTraversal(Node *root, vector<int>&ans)
  {
      if(!root)
      return;

      InorderTraversal(root->left, ans);
      ans.push_back(root->data);
      InorderTraversal(root->right, ans);
  }
    // Function to return a list containing the inorder traversal of the tree.
    vector<int> inOrder(Node* root) {
        // Your code here
        vector<int>ans;
        InorderTraversal(root, ans);
        return ans;
    }
}
```

Count Leaves in Binary Tree << GeeksforGeek >>

```
int countLeaves(Node* root)
{
  // Your code here
  if(!root)
  return 0;

  if(!root-> left && !root -> right)
  return 1;
  return countLeaves(root->left) + countLeaves(root->right);
};
```

Size of Binary Tree <<GeeksforGeeks>>

```
int getSize(Node* root)
{
   // Your code here
   if(!root)
   return 0;

   return 1 + getSize(root->left)+getSize(root->right);
}
```

LEARN DSA WITH C++

WEEK :: 09 **DAY: 02** **DATE: 14-06-2023**

TREES TRAVERSAL

Determine if Two Trees are Identical << GeeksforGeeks>>

```
class Solution {
public:
    // Function to check if two trees are identical.
    bool isIdentical(Node* r1, Node* r2) {
        // Your Code here
        if (!r1 && !r2)
            return 1;

        if (!r1 && r2 || r1 && !r2)
            return 0;

        if (r1->data != r2->data)
            return 0;

        return isIdentical(r1->left, r2->left) && isIdentical(r1->right, r2->right);
    }
};
```

Diameter of a Binary Tree << GeeksforGeeks >>

```
class Solution {
  public:
  // height of the tree

  int find(Node *root, int &ans)
  {
      if(!root)
      return 0;

      int left = find(root->left, ans);
      int right =find(root->right, ans);

      // Diameter calculate
      ans = max(ans, 1+left + right);

      return 1+max(left,right);
  }
    // Function to return the diameter of a Binary Tree.
    int diameter(Node* root) {
        // Your code here
        int ans =0;
        find(root, ans);
        return ans;

    }
};
```

Check for Balanced Tree << GeeksforGeeks >>

```
class Solution {
public:
    int height(Node* root, bool& ans) {
        if (!root)
            return 0;

        int left = height(root->left, ans);
        int right = height(root->right, ans);

        if (abs(left - right) > 1)
            ans = false;

        return 1 + max(left, right);
    }

    // Function to check whether a binary tree is balanced or not.
    bool isBalanced(Node* root) {
        bool ans = true;
        height(root, ans);
        return ans;
    }
};
```

Left View of Binary Tree << GeeksforGeeks>>

```
vector<int> leftView(Node *root)
{
   // Your code here
   vector<int>ans;
   if(!root)
   return ans;

   queue<Node *>q;
   int size;
   q.push(root);

   while(!q.empty())
   {
       ans.push_back(q.front() ->data);
       size = q.size();
       while(size--)
       {
           Node *temp = q.front();
           q.pop();
           if(temp ->left)
           q.push(temp ->left);
           if(temp ->right)
           q.push(temp ->right);
       }
   }
   return ans;
}
```

LEARN DSA WITH C++

WEEK :: 09 **DAY**: 03 **DATE**: 15-06-2023

TREES TRAVERSAL

Nodes at Odd Levels << GeeksforGeeks >>

```
class Solution
{
public:

void PreOrder(Node *root, vector<Node *>&ans, int level)
{
    if(!root)
    return;

    if(level%2 == 1)
    ans.push_back(root);

    PreOrder(root ->left, ans, level +1);
    PreOrder(root ->right, ans, level +1);
};

    vector<Node *> nodesAtOddLevels(Node *root)
    {
        //code here
        vector<Node *>ans;
        int level = 1;
        PreOrder(root, ans, level);
        return ans;
    }
};
```

Sum Tree << GeeksforGeeks >>

```
class Solution {

    int sumTree(Node* root, bool& ans) {
        if (!root)
            return 0;

        // Identify leaf node
        if (!root->left && !root->right)
            return root->data;

        // left sum
        int left = sumTree(root->left, ans);
        int right = sumTree(root->right, ans);

        // right sum
        if (left + right != root->data)
            ans = false;
```

```
        return root->data + left + right;
    }
public:
    bool isSumTree(Node* root) {
        bool ans = true;
        sumTree(root, ans);
        return ans;
    }
};
```

Maximum difference between node and its ancestor << GeeksforGeeks >>

```
void Find(Node *root, int Anc_max, int &Diff)
 {
     if(!root)
     return;
     Diff = max(Diff, Anc_max-root->data);

     Anc_max = max(Anc_max, root->data);
     Find(root->left, Anc_max, Diff);
     Find(root->right, Anc_max, Diff);
 }
//Function to return the maximum difference between any node and its ancestor.
int maxDiff(Node* root)
{
    // Your code here
    int Diff =INT_MIN;
    Find(root-> left, root->data, Diff);
    Find(root ->right, root->data, Diff);
    return Diff;
}
```

Preorder traversal (Iterative) << GeeksforGeeks >>

```
class Solution{
    public:
    vector<int> preOrder(Node* root)
    {
        //code here
        vector<int>ans;
        stack<Node *>s;
        s.push(root);  // Root element in stack
        Node *temp;
        while(!s.empty())
        {
            temp = s.top();
            s.pop();
            if(temp->right)
            s.push(temp->right);
            if(temp->left)
            s.push(temp->left);
            ans.push_back(temp->data);
        };
        return ans;
    }
};
```

LEARN DSA WITH C++

WEEK :: 09 **DAY**: 04 **DATE**: 16-06-2023

TREES QUESTION PART I

Print Nodes having K leaves << GeeksforGeeks >>

```
class Solution{
  public:
    /*You are required to complete below method */
    int Find(Node *root, vector<int> &count, int k)
    {
        if(!root)
        return 0;

        //leaf node
        if(!root ->left && !root ->right)
        return 1;

        int left = Find(root->left, count, k);
        int right = Find(root->right, count, k);

        if(k==left+right)
        count.push_back(root ->data);

        return left + right;
    }

    vector<int> btWithKleaves(Node *ptr, int k)
    {
        //your code here.
        vector<int>count;
        Find(ptr, count, k);
        if(count.empty())
        count.push_back(-1);
        return count;
    }

};
```

Level order traversal in spiral form << GeeksforGeeks >>

```
vector<int> findSpiral(Node *root)
{
    //Your code here
    vector<int>ans;
    if(!root)
    return ans;
    queue<Node *>q;
    stack<Node *>s;
    q.push(root);
    bool dir = 0;
    Node *temp;
```

```
    while(!q.empty())
    {
        // Right to left
        if(dir ==0)
        {
            int size = q.size();
            while(!q.empty())
            {
                temp = q.front();
                q.pop();

                if(temp ->right)
                s.push(temp ->right);
                if(temp ->left)
                s.push(temp ->left);
                ans.push_back(temp ->data);
            }
            dir = 1;
        }
        else  // Left to Right
        {
             int size = q.size();
            while(!q.empty())
            {
                temp = q.front();
                q.pop();

                if(temp ->left)
                s.push(temp ->left);
                if(temp ->right)
                s.push(temp ->right);
                ans.push_back(temp ->data);
            }
            dir = 0;
        }
        while(!s.empty())
        {
            q.push(s.top());
            s.pop();
        }
    }

    return ans;
  }
```

Boundary Traversal of binary tree << GeeksforGeeks >>

```
class Solution {
public:

// Left Subtree Except leaf
void Left_sub(Node* root, vector<int>&ans)
{
    if(!root || !root -> left && !root -> right)
    return;

    ans.push_back(root->data);
```

```
    if(root ->left)
    Left_sub(root ->left, ans);
    else
    Left_sub(root -> right, ans);
    return;
}

// Leaf node
void Leaf_sub(Node *root, vector<int>&ans)
{
    if(!root)
    return;

    if(!root-> left && !root ->right)
    {
        ans.push_back(root ->data);
        return;
    }

    Leaf_sub(root ->left, ans);
    Leaf_sub(root ->right, ans);
}

// Reverse Right Subtree except leaf
void Right_sub(Node *root, vector<int>&ans)
{
    if(!root || !root ->left && !root -> right)
    return;

    if(root ->right)
    Right_sub(root ->right, ans);
    else
    Right_sub(root->left, ans);

    ans.push_back(root ->data);
}
    vector <int> boundary(Node *root)
    {
        //Your code here
        vector<int>ans;
        ans.push_back(root ->data);
        Left_sub(root-> left, ans);
        if(root ->left || root ->right)
        Leaf_sub(root, ans);
        Right_sub(root ->right, ans);
        return ans;
    }
};
```

Diagonal Traversal << InterviewBit >>

```
void pre(TreeNode *A, vector<vector<int>> &v, int left)
{ if(!A) return;
if(left ==v.size())
v.push_back({A ->val});
else
v[left].push_back(A->val);
pre(A->left, v, left +1);
pre(A->right, v, left);
}
vector<int> Solution::solve(TreeNode* A)
{
vector<vector<int>>v;
pre(A, v, 0);
vector<int>ans;
for(int i=0; i<v.size(); i++)
for(int j=0; j<v[i].size();
j++) ans.push_back(v[i][j]);
return ans;
```

LEARN DSA WITH C++

WEEK :: 09 **DAY**: 05 **DATE**: 19-06-2023

TREES QUESTION PART II

Maximum Path Sum between 2 Special Nodes << GeeksforGeeks >>

```
class Solution {
public:

    int maxSum(Node *root, int &sum)
    {
        // Null
        if(!root)
        return 0;

        // leaf Node
        if(!root ->left && !root ->right)
        return root -> data;

        int left = maxSum(root ->left, sum);
        int right = maxSum(root ->right, sum);

        // left right node exist
        if(root ->left && root ->right)
        {
            sum = max(sum, root->data + left +right);
            return root ->data + max(left, right);
        }
        else if(root ->left)
        {
            return root ->data + left;
        }
        else
        return root ->data + right;
    }

    int maxPathSum(Node* root)
    {
        // code here
        int sum = INT_MIN;
        int val = maxSum(root, sum);
        if(root->left && root ->right)
        return sum;
        return max(sum, val);
    }
};
```

Burning Tree << GeeksforGeeks >>

```
class Solution {
  public:
```

```
int Burn(Node *root, int target, int &timer)
{
    if(!root)
    return 0;

    //Burning Node
    if(root ->data == target)
    return -1;

    int left = Burn(root ->left, target, timer);
    int right = Burn(root ->right, target, timer);

    //left side Burnout
    if(left<0)
    {
        timer = max(timer, abs(left)+right);
        return left -1;
    }

    // Right side Burnout
    if(right<0)
    {
        timer = max(timer, abs(right)+left);
        return right-1;
    }

    // left and right both positive
    return max(left, right) +1;
}

void Find(Node *root, int target, Node *&temp)
{
    if(!root)
    return;

    if(root ->data == target)
    {
        temp = root;
        return;
    }

    Find(root ->left, target, temp);
    Find(root ->right, target, temp);
};

int Height(Node *root)
{
    if(!root)
    return 0;

    return 1+max(Height(root ->left), Height(root ->right));
};

int minTime(Node* root, int target)
{
    // Your code goes here
    int timer = 0;
    Burn(root, target, timer);
```

```
        Node *temp;
        Find(root, target, temp);
        int num = Height(temp)-1;

        return max(timer, num);
    }
};
```