GROUP B Kaustubh Shrikant Kabra

SE Comp Shift-1

PRACTICAL NO.1 Teams Sr.No.-20

Erp Roll Number-34

**Aim:**

A book consists of chapters , chapters consist of sections and sections consist of sub sections . Construct a tree and print the nodes . Find the time and space complexity of the program .

**Objective:**

To learn the basics of tree data structure in C++ and apply it .

**Pre-requisites:**

Basic C++,structure and classes in c++, Linked List

**Theory:**

**Trees**:

Unlike Arrays, Linked Lists, Stack and queues, which are linear data structures, trees are hierarchical data structures.

Trees are non-linear hierarchical data structures. A tree is a collection of nodes connected to each other by means of “edges” which are either directed or undirected. One of the nodes is designated as “Root node” and the remaining nodes are called child nodes or the leaf nodes of the root node.In general, each node can have as many children but only one parent node.

### **Tree with its various parts.**

* **Root node:** This is the topmost node in the tree hierarchy. In the above diagram, Node A is the root node. Note that the root node doesn’t have any parent.
* **Leaf node:** It is the Bottom most node in a tree hierarchy. Leaf nodes are the nodes that do not have any child nodes. They are also known as external nodes. Nodes E, F, G, H and C in the above tree are all leaf nodes.
* **Subtree:** Subtree represents various descendants of a node when the root is not null. A tree usually consists of a root node and one or more subtrees. In the above diagram, (B-E, B-F) and (D-G, D-H) are subtrees.
* **Parent node:** Any node except the root node that has a child node and an edge upward towards the parent.
* **Ancestor Node:** It is any predecessor node on a path from the root to that node. Note that the root does not have any ancestors. In the above diagram, A and B are the ancestors of E.
* **Key:** It represents the value of a node.
* **Level:** Represents the generation of a node. A root node is always at level 1. Child nodes of the root are at level 2, grandchildren of the root are at level 3 and so on. In general, each node is at a level higher than its parent.
* **Path:** The path is a sequence of consecutive edges. In the above diagram, the path to E is A=>B->E.
* **Degree:** Degree of a node indicates the number of children that a node has. In the above diagram, the degree of B and D is 2 each whereas the degree of C is 0.

**Tree types in c++:**

* General tree
* Forest
* Binary tree
* Binary Search tree
* Expression tree

**Binary Tree:**

A Binary tree is a widely used tree data structure. When each node of a tree has at most two child nodes then the tree is called a Binary tree.

**So a typical binary tree will have the following components:**

* A left subtree
* A root node
* A right subtree

**Binary Tree Types:**

* Full Binary Tree
* Complete Binary tree
* Perfect Binary tree
* Balanced Binary Tree

**Program:**

#include<iostream>

#include<stdio.h>

#include<queue>

using namespace std;

class Tree

{

typedef struct node

{

char data[10];

struct node \*left;

struct node \* right;

}btree;

public:

btree \*New,\*root;

Tree();

void create();

void insert(btree \*root,btree \*New);

void display();

};

Tree::Tree()

{

root=NULL;

}

void Tree::create()

{

New=new btree;

New->left=New->right=NULL;

cout<<"\n\tEnter the Data: ";

cin>>New->data;

if(root==NULL)

{

root=New;

}

else

{

insert(root,New);

}

}

void Tree::insert(btree \*root,btree \*New)

{

char ans;

cout<<"\n\t"<<New->data<<" Want to Insert at "<<root->data<<" at Left(L) OR Right(R)";

cin>>ans;

if(ans=='L'||ans=='l')

{

if(root->left==NULL)

root->left=New;

else

insert(root->left,New);

}

else

{

if(root->right==NULL)

root->right=New;

else

insert(root->right,New);

}

}

void Tree::display()

{

int i=1;

if(root==NULL)

{

cout<<"\n NULL Tree";

return;

}

queue<btree \*> q;

q.push(root);

cout<<"\n\tLevelwise(BFS) Traversal\n";

while(q.empty()==false)

{

btree \*node=q.front();

if(i==1)

cout<<node->data<<"\n";

if(i==2)

cout<<node->data<<"\t";

if(i==3)

cout<<node->data<<"\n";

if(i==4||i==5||i==6||i==7)

cout<<node->data<<"\t";

i++;

q.pop();

if(node->left!=NULL)

q.push(node->left);

if(node->right!=NULL)

q.push(node->right);

}

}

int main()

{

Tree tr;

int i=0;

do

{

if(i==0)

{

cout<<"\n\tEnter Chapter Name";

tr.create();

i++;

}

if(i==1||i==2)

{

cout<<"\n\tEnter Section Name";

tr.create();

i++;

}

if(i==3||i==4||i==5||i==6)

{

cout<<"\n\tEnter Sub-Section Name";

tr.create();

i++;

}

if(i==7)

{

cout<<"\n tree is:";

tr.display();

break;

}

}while(1);

}

**Output:**

Enter Chapter Name

Enter the Data: Chapter-1

Enter Section Name

Enter the Data: Section1.1

Section1.1 Want to Insert at Chapter-1 at Left(L) OR Right(R)L

Enter Section Name

Enter the Data: Section1.2

Section1.2 Want to Insert at Chapter-1 at Left(L) OR Right(R)R

Enter Sub-Section Name

Enter the Data: SubSection1.1.1

SubSection1.1.1 Want to Insert at Chapter-1 at Left(L) OR Right(R)L

SubSection1.1.1 Want to Insert at Section1.1 at Left(L) OR Right(R)L

Enter Sub-Section Name

Enter the Data: SubSection1.1.2

SubSection1.1.2 Want to Insert at Chapter-1 at Left(L) OR Right(R)L

SubSection1.1.2 Want to Insert at Section1.1 at Left(L) OR Right(R)R

Enter Sub-Section Name

Enter the Data: SubSection1.2.1

SubSection1.2.1 Want to Insert at Chapter-1 at Left(L) OR Right(R)R

SubSection1.2.1 Want to Insert at Section1.2 at Left(L) OR Right(R)L

Enter Sub-Section Name

Enter the Data: SubSection1.2.2

SubSection1.2.2 Want to Insert at Chapter-1 at Left(L) OR Right(R)R

SubSection1.2.2 Want to Insert at Section1.2 at Left(L) OR Right(R)R

tree is:

Levelwise(BFS) Traversal

Chapter-1

Section1.1 Section1.2

SubSection1.1.1 SubSection1.1.2 SubSection1.2.1 SubSection1.2.2

**Complexity:**

**Time Complexity:**

1. Creating node: O(1)
2. Inserting Node : O(1)
3. Displaying Tree : O(n) {n=number of node tree have}

**Space Complexity:**

Space complexity:O(h) {h= height of tree}

O(n) (in worst case )

**Outcome:**

Hence , we are able to implement Tree data structure in C++.

**Conclusion:**

Hence, Trees are a non-linear hierarchical data structure that is used in many applications in the software field.Unlike linear data structures that have only one way to traverse the list, we can traverse trees in a variety of ways.