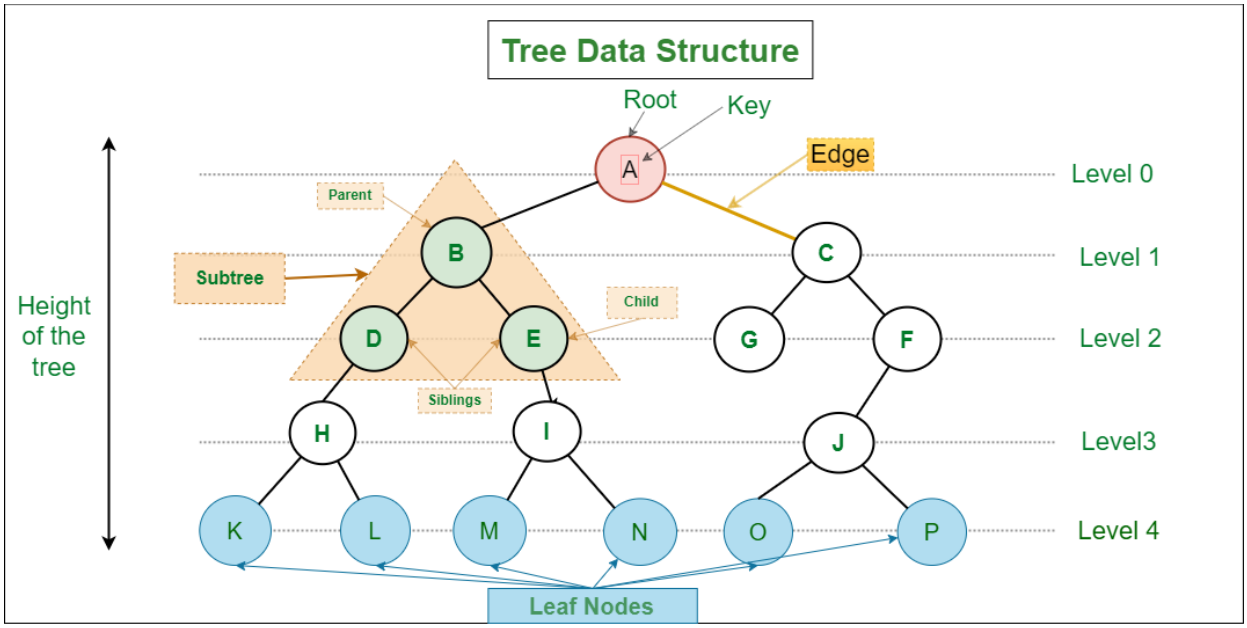
Lecture 11: Tree Traversals

Date: 09/26/2023

Polls: https://vevox.app/#/m/129412484

Tree Introduction



Reference: https://www.geeksforgeeks.org/introduction-to-tree-data-structure-and-algorithm-tutorials/

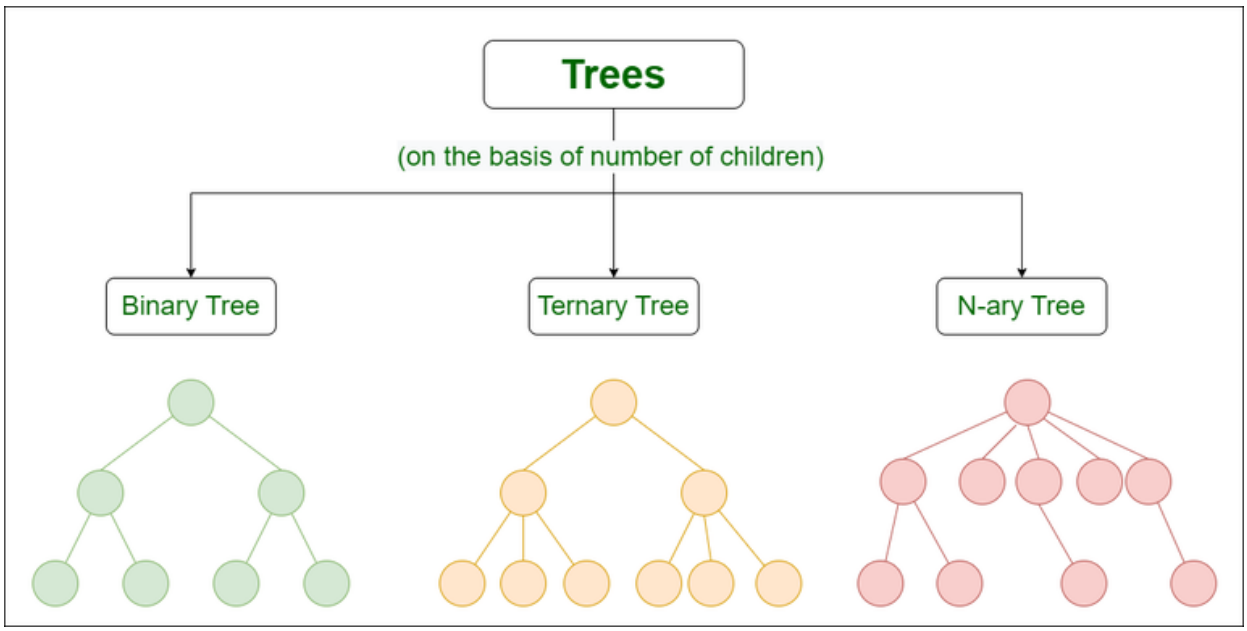
Terminologies

- **Parent Node**: The node which is a predecessor of a node is called the parent node of that node. {B} is the parent node of {D, E}.
- **Child Node**: The node which is the immediate successor of a node is called the child node of that node. Examples: {D, E} are the child nodes of {B}.
- **Root Node**: The topmost node of a tree or the node which does not have any parent node is called the root node. {A} is the root

node of the tree. A non-empty tree must contain exactly one root node and exactly one path from the root to all other nodes of the tree.

- **Leaf Node or External Node**: The nodes which do not have any child nodes are called leaf nodes. {K, L, M, N, O, P} are the leaf nodes of the tree.
- **Ancestor of a Node**: Any predecessor nodes on the path of the root to that node are called Ancestors of that node. {A,B} are the ancestor nodes of the node {E}
- **Descendant**: Any successor node on the path from the leaf node to that node. {E,I} are the descendants of the node {B}.
- **Sibling**: Children of the same parent node are called siblings. {D,E} are called siblings.
- **Level of a node**: The count of edges on the path from the root node to that node. The root node has level 0.
- **Internal node**: A node with at least one child is called Internal Node.
- **Neighbor**: Parent or child nodes of that node are called neighbors of that node.
- **Subtree**: Any node of the tree along with its descendant.

Types



Reference: https://www.geeksforgeeks.org/introduction-to-tree-data-structure-and-algorithm-tutorials/

- **Binary Tree**: Less than or equal to Two Childrens for each node
- **Ternary Tree**: Less than or equal to Three Childrens for each node
- **N-ary Tree**: Each node can have at max n childrens

⇒ We will be focusing on Binary Trees for this lecture.

Implementation

1_tree.cpp

```cpp
class Node {
public:
    int data;
    Node* left;
    Node* right;

    Node(int val){
        data = val;
    }
};

int main(){
    Node* root = new Node(10);
    Node* leftNode = new Node(15);
    Node* rightNode = new Node(20);

    root -> left = leftNode;
    root -> right = rightNode;

    cout << "root: " << root->data << "\n";
    cout << "\t->left: " << root->left->data << "\n";
    cout << "\t->right: " << root->right->data << "\n";

    return 0;
}
```

Tree Traversal

Traversing a tree means visiting and outputting the value of each node in a particular order.
As a tree is not a linear data structure like array / stack / queue etc. There can be multiple ways to traverse a tree.

Main Categories:

- Depth First Search
- Breadth First Search

Depth First Search

Depth First Search (DFS) traverses a tree Depth Wise.
⇒ For Binary Tree, there are three major DFS algorithms,

- InOrder
- PreOrder
- PostOrder

InOrder

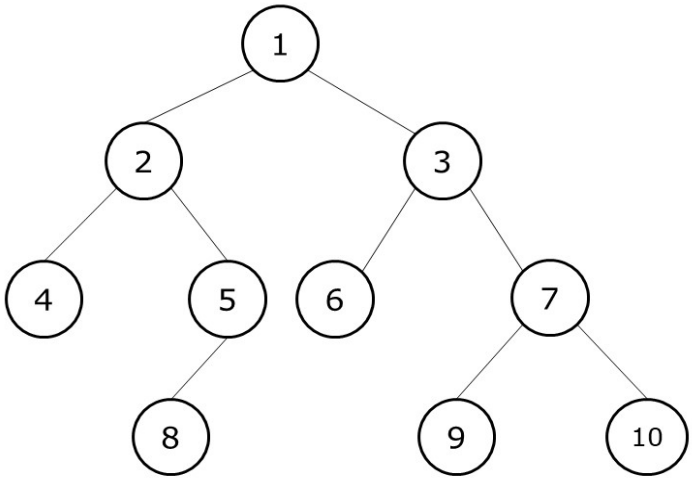
Traversal: Left -> **Root** -> Right

2_inorder.cpp

```
void inorder(Node* root){
    if(root == nullptr) return;

    inorder(root->left);
    cout << root -> data << "\n";
    inorder(root->right);
}
```

Example:



Inorder Traversal:
[left,root,right]

4	2	8	5	1	6	3	9	7	10

Reference: https://takeuforward.org/data-structure/inorder-traversal-of-binary-tree/

PreOrder

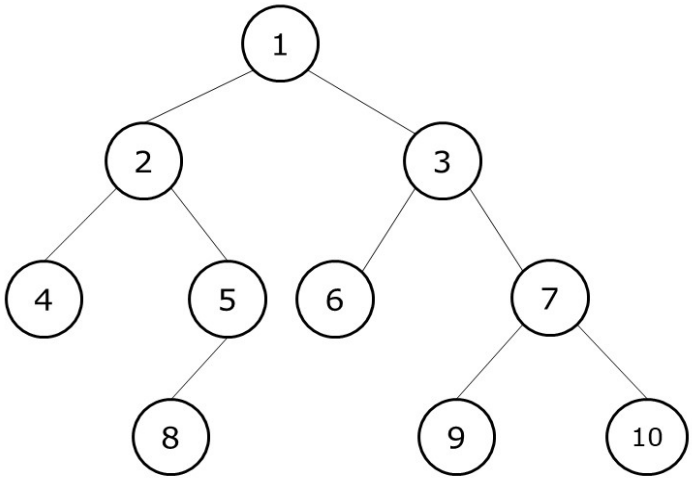
Traversal: **Root** -> Left -> Right

3_preorder.cpp

```
void preorder(Node* root){
    if(root == nullptr) return;

    cout << root -> data << "\n";
    preorder(root->left);
    preorder(root->right);
}
```

Example:



Preorder Traversal:
[root, left, right]

1	2	4	5	8	3	6	7	9	10

Reference: https://takeuforward.org/data-structure/preorder-traversal-of-binary-tree/

PostOrder

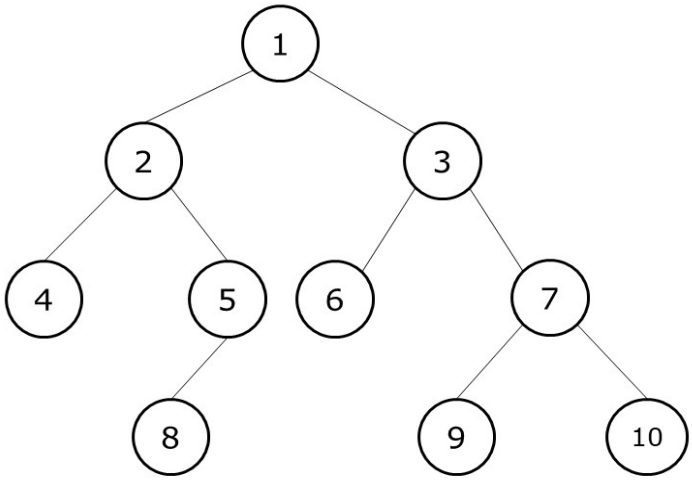
Traversal: Left -> Right -> **Root**

4_postorder.cpp

```
void postorder(Node* root){
    if(root == nullptr) return;

    postorder(root->left);
    postorder(root->right);
    cout << root -> data << "\n";
}
```

Example:



Postorder Traversal:
[left, right, root]

4	8	5	2	6	9	10	7	3	1

Reference: https://takeuforward.org/data-structure/post-order-traversal-of-binary-tree/

Breadth First Search

Breadth First Search (BFS) traverses a tree Breadth Wise.

5_bfs.cpp

```
void bfs(Node* root){
    // current level
    vector<Node*> level;
    level.push_back(root);

    // iterate until we have exhausted all nodes
    while(level.size()!=0){
        // to store next level nodes that will be
        // traversed in next iteration
        vector<Node*> newLevel;

        // for each node in current level, add all its
        // non-null child to nextLevel
        for(Node* node: level){

            // printing out current node data
```

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

            // add if non-null
            if(node->left)
            newLevel.push_back(node->left);

            // add if non-null
            if(node->right)
            newLevel.push_back(node->right);
        }

        // building next level
        level.clear();
        for(Node* node: newLevel)
        level.push_back(node);
    }
}
```

Example:

Traversal Usage

- **Pre-order**: Used to create a copy of a tree. For example, if you want to create a replica of a tree, put the nodes in an array with a pre-order traversal. Then perform an Insert operation on a new tree for each value in the array. You will end up with a copy of your original tree.

- **In-order**: Used to get the values of the nodes in non-decreasing order in a BST.

- **Post-order**: Used to delete a tree from leaf to root

- **Breadth First Search**: Find shortest path from source to target