Data Structures

Tree: Terminology, Binary Tree, Binary Tree Traversal, Binary Search Tree, AVL tree, B Tree, Applications

Alka Jindal

Microsoft PowerPoint - [PowerPoint Slide Show - [7_TreeCLASS]]

What is a Tree?

- Similar to tree in general world
- Finite set of elements
- Non-linear data structure
- Used to represent hierarchical structures
- Eg. Syntax tree, Binary Search Tree, Animal Kingdom
- Application: Organization Charts, File system
- Tree can also be defined in itself as a node and list of child trees.

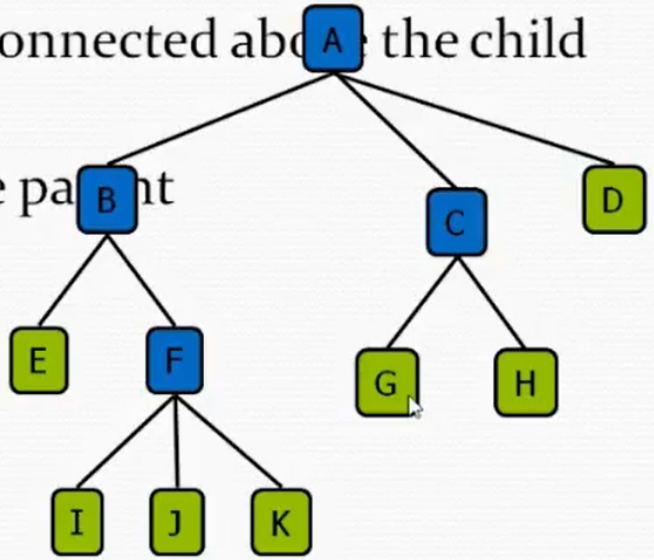
❖ **In Computer Science, Trees grow down!** ☺

Alka Jindal

Terminology

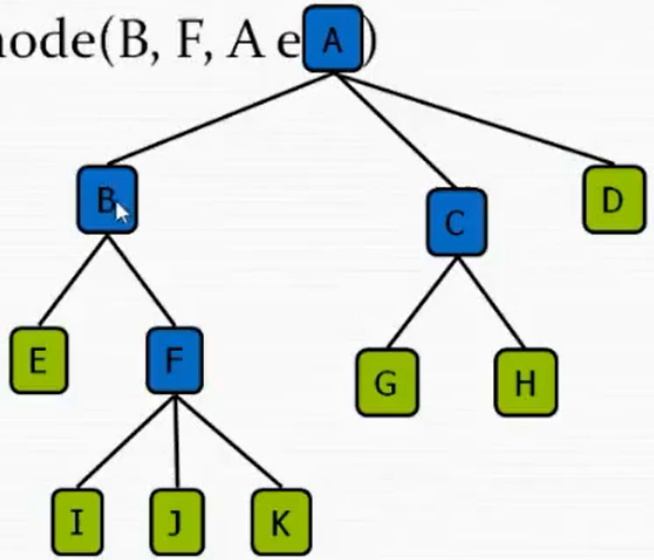
- **Node:** stores the key (all A, B, C… K)
- **Child node:** node directly connected below the parent node (B is child of A)
- **Parent node:** node directly connected above the child node (B is parent of E)
- **Siblings:** nodes sharing same parent

(E and F are siblings)



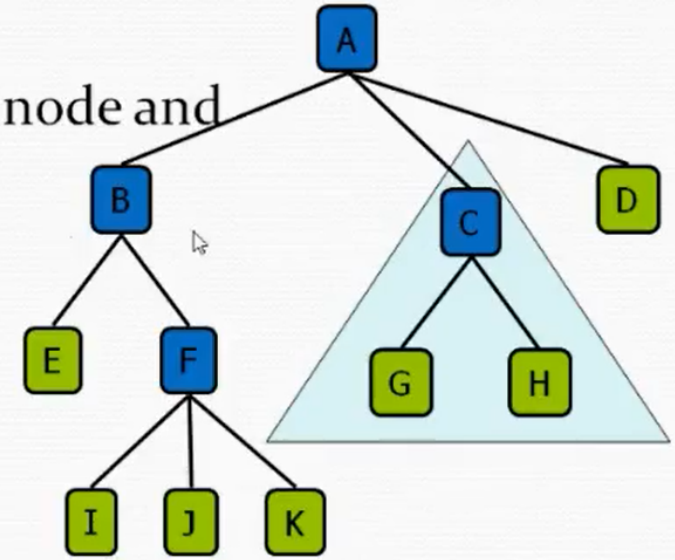
Terminology

- **Root**: top node in tree or node with no parent (A is root)
- **Leaf:** node with no child (E, I, J etc. are leaf node)
- **Interior node**: All non-leaf node(B, F, A e)



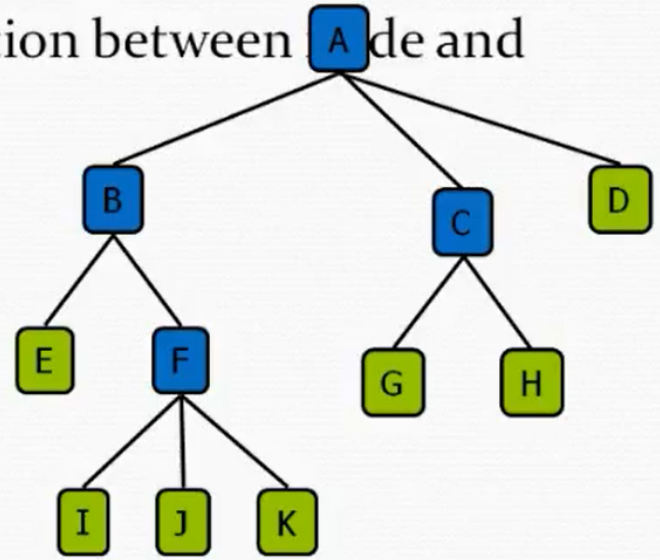
Terminology

- **Ancestors**: parent, parent of parent and so on. (F, B, A are ancestors of K)
- **Descendants**: child, child of child and so on. (E, F, I, J, K are descendants of B)
- **Subtree**: tree consisting of a node and its descendants



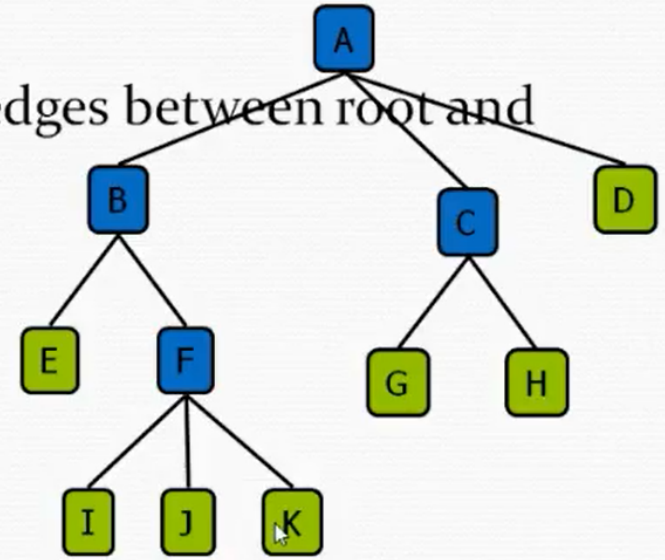
Terminology

- **Degree of a node**: number of its child (degree(A)=3, degree(B)=2)
- **Degree of tree:** maximum of degrees of all nodes
- **Level:** 1 + number of connection between node and root (level(F)=3)



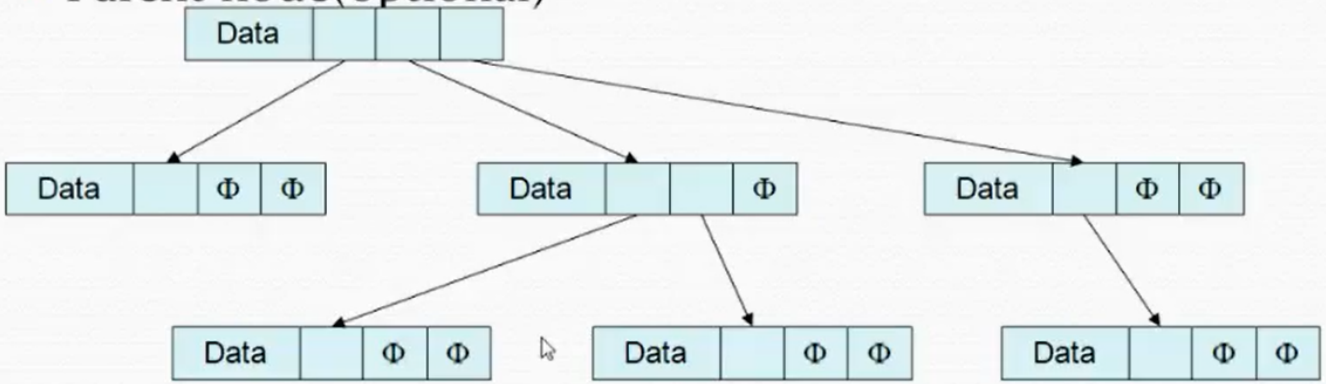
Terminology

- **Height of node:** number of edges between node and farthest leaf (Height(B)=2)
- **Height of tree:** height of root node (height of tree is 3)
- **Depth of node**: number of edges between root and node (depth of K is 3)



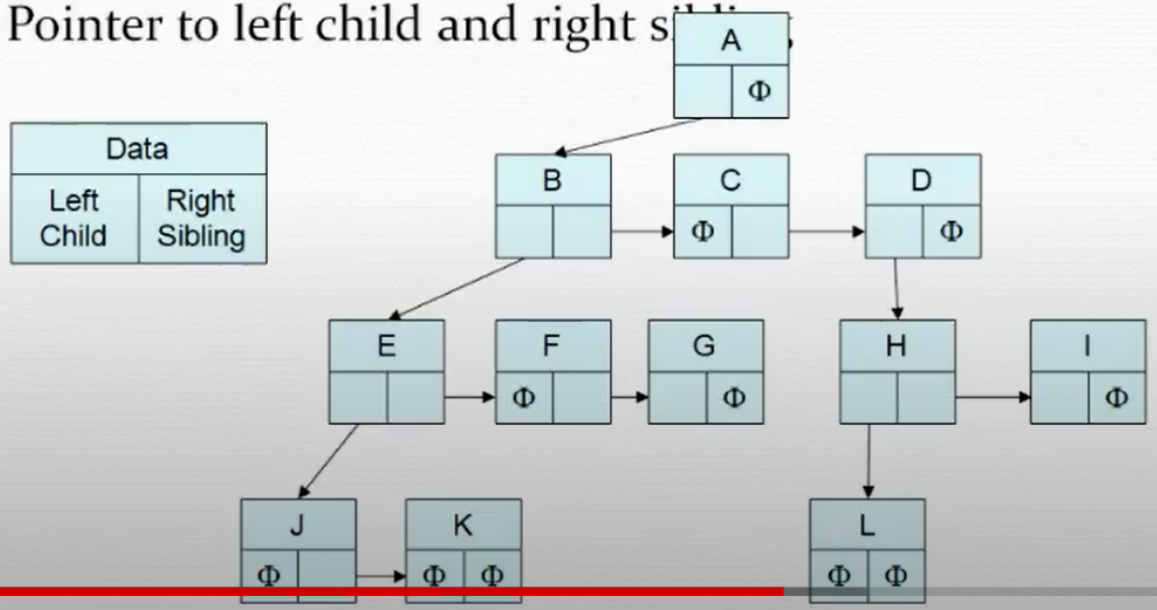
Tree Representation

- Every node contains:
  - Key/data
  - Children nodes
  - Parent node(optional)



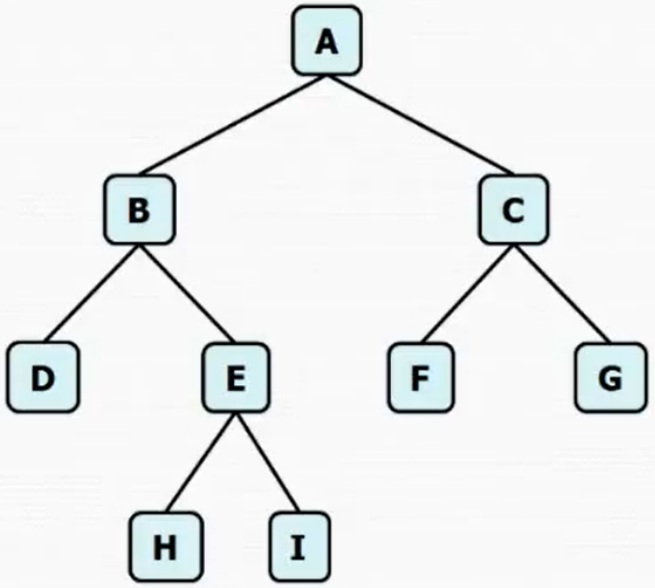
Left Child, Right Sibling Representation

- Every node contains
  - Key/data
  - Pointer to left child and right s



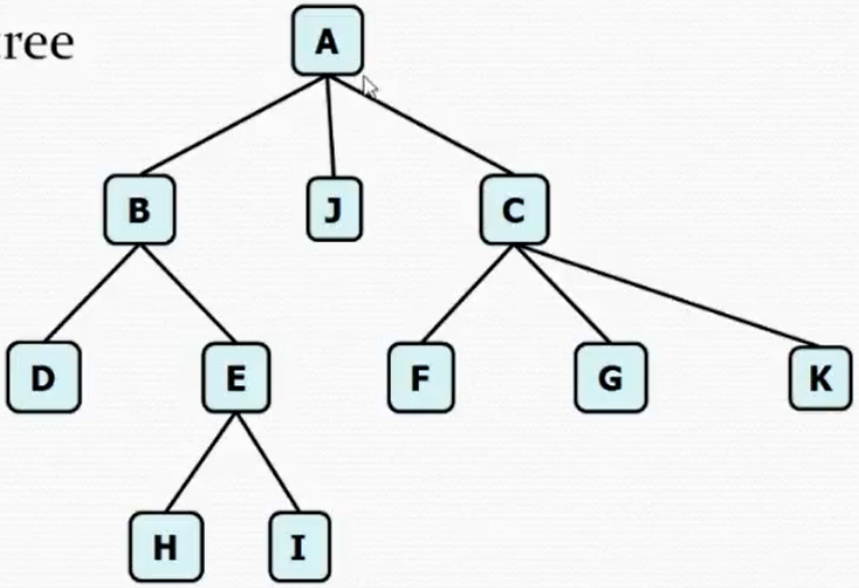
Binary Tree

- Each node has at most two children
- The children of a node are ordered pair (a left and a right child)
- Each node contains:
  - Key
  - Left
  - Right
  - Parent(optional)



K-ary Tree

- Each node has at most **K** children
- Binary tree is a special case with K=2
- Eg. 3-ary tree



Microsoft PowerPoint - [PowerPoint Slide Show - [7_TreeCLASS]]

- TRAVERSAL

A

Alka Jindal

Breadth First Traversal

- Traverse all the nodes at level ***i*** before progressing to level ***i+1*** starting from root node
- Add nodes in the queue as soon as their parent is visited.
- In each iteration, delete one element from queue and mark visited

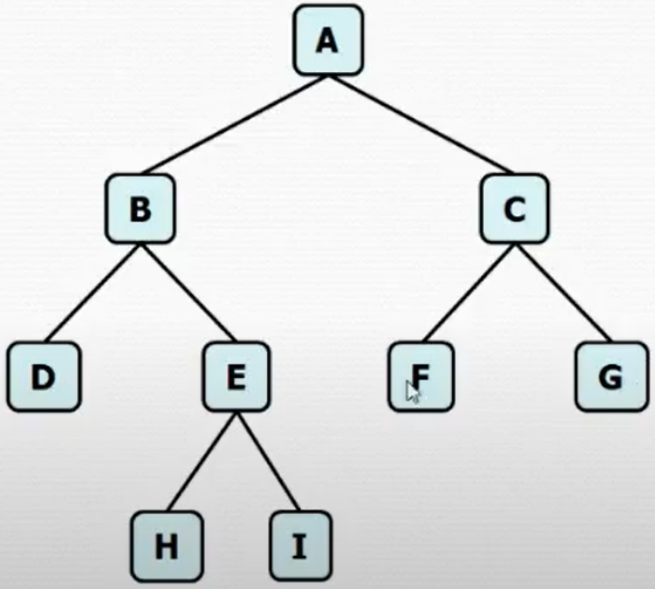
BFS Algorithm

```
BFS(Tree) {
        if (!isEmpty(Tree)) enqueue(Q, root);
        while (!isEmpty(Q)) {
                node = dequeue(Q);
                print(node->data);
                if (node->left != NULL) enqueue(Q,node->left);
                if (node->right != NULL) enqueue(Q,node->right);
        }
}
```

BFS Example

Output: A B C D E

Queue(Q): F, G, H, I



Depth First Search

- Travel All the nodes of one sub-tree of binary search before travelling other sub-tree
- DFS is recursively implemented on a tree to visit nodes
- **Note: Many of the tree algorithms are recursively implemented for the reason that tree itself is implemented recursively**
- DFS on binary tree can be implemented in 3 ways
  - PreOrder: Root-Left-Right
  - InOrder: Left-Root-Right
  - PostOrder: Left-Right-Root

Depth First Search

- Travel All the nodes of one sub-tree of binary search before travelling other sub-tree
- DFS is recursively implemented on a tree to visit nodes
- **Note: Many of the tree algorithms are recursively implemented for the reason that tree itself is implemented recursively**
- DFS on binary tree can be implemented in 3 ways
  - PreOrder: Root-Left-Right
  - InOrder: Left-Root-Right
  - PostOrder: Left-Right-Root

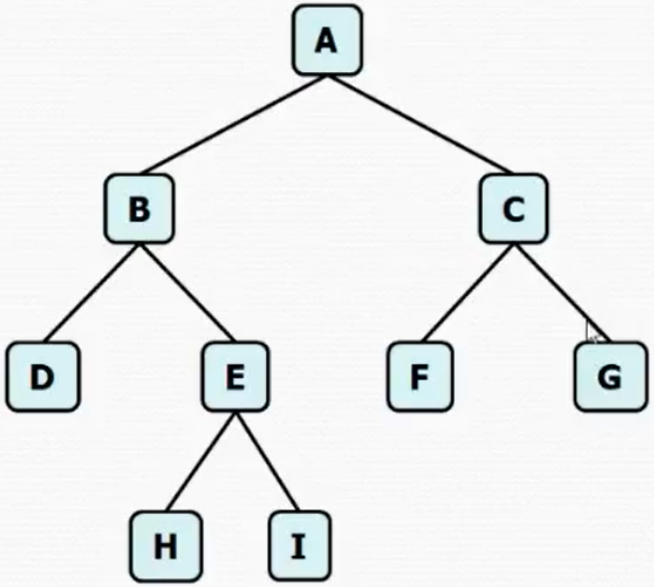
PreOrder Traversal

- Visit the root node first
- Visit left sub-tree in PreOrder
- Visit right sub-tree in PreOrder

```
PreOrderTraversal(Tree) {
        if (isEmpty(Tree)) return;
        else {
                print (tree->data);
                PreOrderTraversal(tree->left);
                PreOrderTraversal(tree->right);
        }
}
```

PreOrder Traversal: Example

Output: A B D E H I C F G



InOrder Traversal

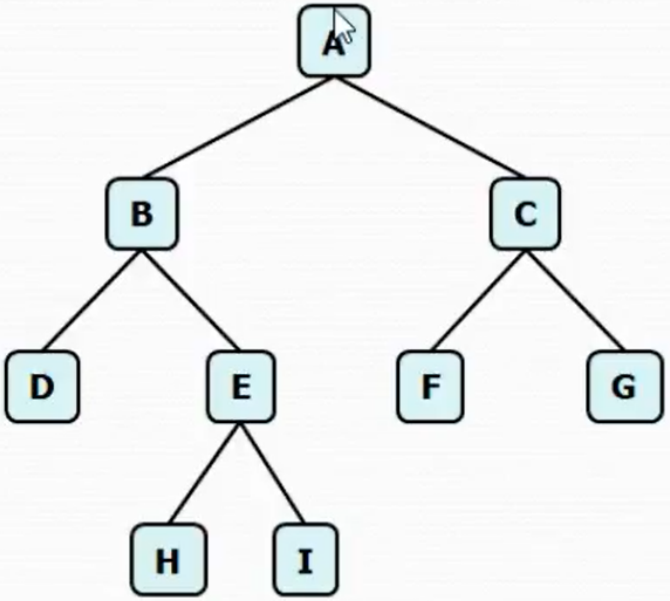
- Visit the left sub-tree in InOrder
- Visit root node
- Visit right sub-tree in InOrder

```
InOrderTraversal(Tree) {
        if (isEmpty(Tree)) return;
        else {
                InOrderTraversal(tree->left);
                print (tree->data);
                InOrderTraversal(tree->right);
        }
}
```

Microsoft PowerPoint - [PowerPoint Slide Show - [7_TreeCLASS]]

InOrder Traversal: Example

Output: D B H E I A F C G



A

Alka Jindal