Trees

Ye Yang
Stevens Institute of Technology

Drawbacks of Lists

- So far, the ADT's we've examined have been linear
  - O(N) for simple operations
- Can we do better?
- Recall binary search: log N for find :-)
- But list must be sorted. N log N to sort :-(

Trees

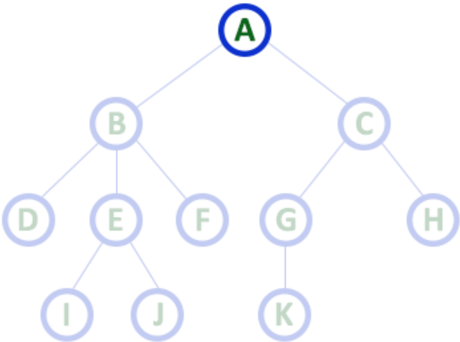
- Extension of Linked List structure:
  - Each node connects to multiple nodes
- Examples include file systems, Java class hierarchies

Tree Terminology

- Just like Lists, **Trees** are collections of **nodes**
- Conceptualize trees upside down (like family trees)
  - the top node is the **root**
  - nodes are connected by **edges**
  - edges define **parent** and **child** nodes
  - nodes which belong to same Parent are called as **siblings**
  - nodes with no children are called **leaves (a.k.a. external nodes)**
  - nodes with at least one child are called as **internal nodes**
  - **Degree, level, height, depth, path, subtree**

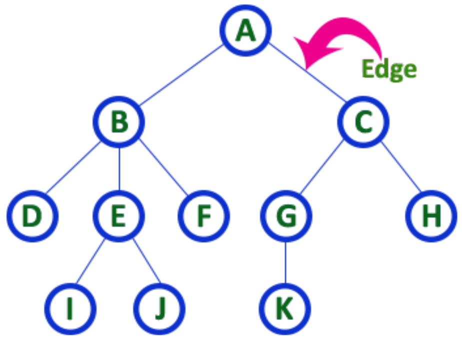
More Tree Terminology

- A **path** is a sequence of nodes such that the next node in the sequence is a child of the previous
- A node's **depth** is the length of the path from root
- the **height** of a tree is the maximum depth
- if a path exists between two nodes, one is an **ancestor** and the other is a **descendant**

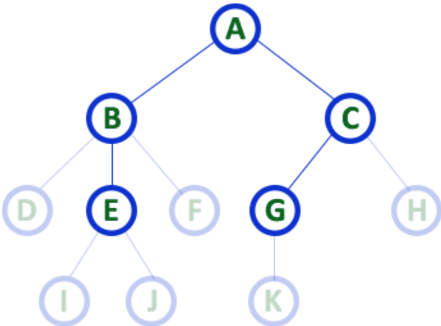


Here 'A' is the 'root' node

- In any tree the first node is called as ROOT node



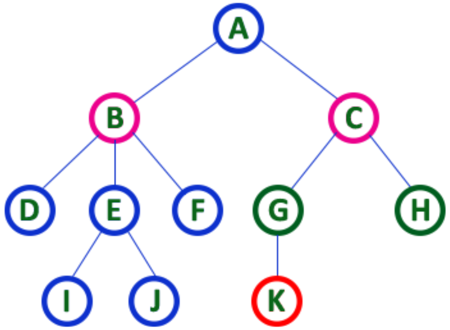
- In any tree, 'Edge' is a connecting link between two nodes.



Here A, B, C, E & G are Parent nodes

- In any tree the node which has child / children is called 'Parent'

- A node which is predecessor of any other node is called 'Parent'

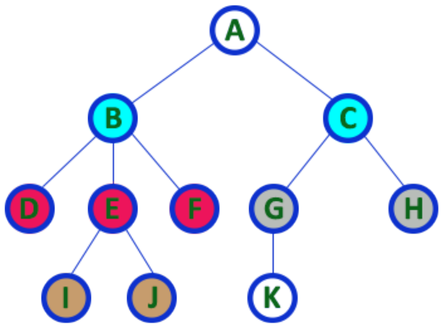


Here B & C are Children of A

Here G & H are Children of C

Here K is Child of G

- descendant of any node is called as CHILD Node



Here B & C are Siblings

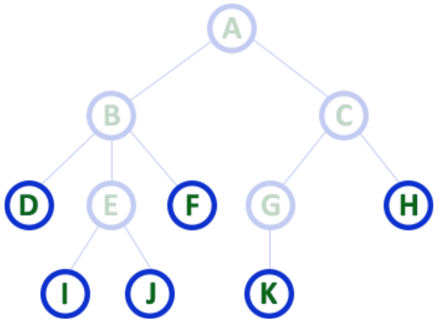
Here D E & F are Siblings

Here G & H are Siblings

Here I & J are Siblings

- In any tree the nodes which has same Parent are called 'Siblings'

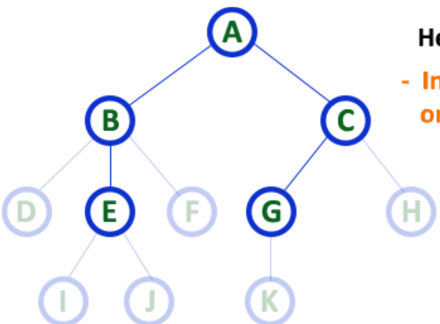
- The children of a Parent are called 'Siblings'



Here D, I, J, F, K & H are Leaf nodes

- In any tree the node which does not have children is called 'Leaf'

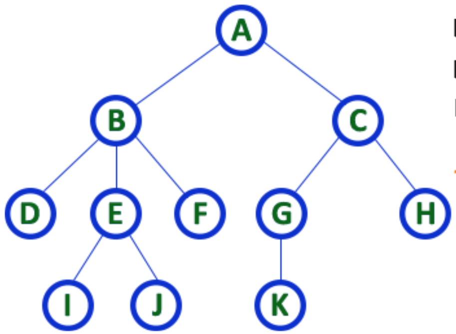
- A node without successors is called a 'leaf' node



Here A, B, C, E & G are Internal nodes

- In any tree the node which has atleast one child is called 'Internal' node

- Every non-leaf node is called as 'Internal' node

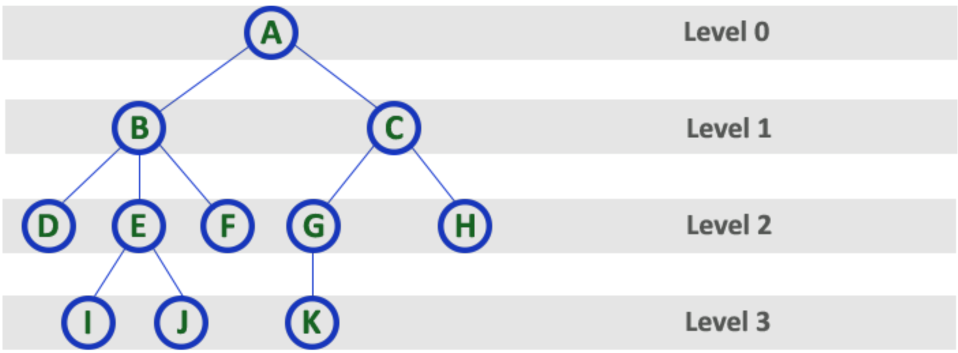


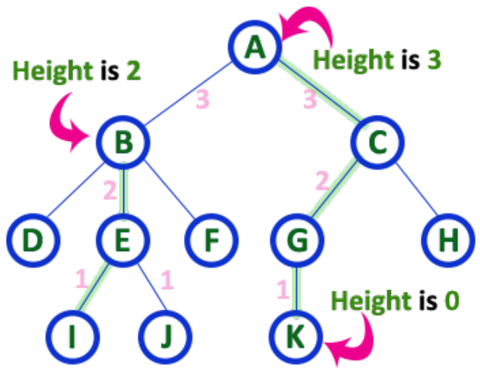
Here Degree of B is 3

Here Degree of A is 2

Here Degree of F is 0

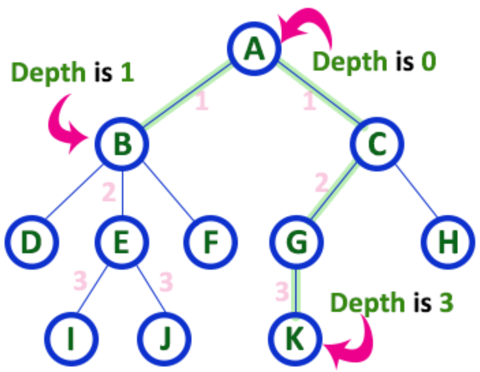
- In any tree, 'Degree' a node is total number of children it has.





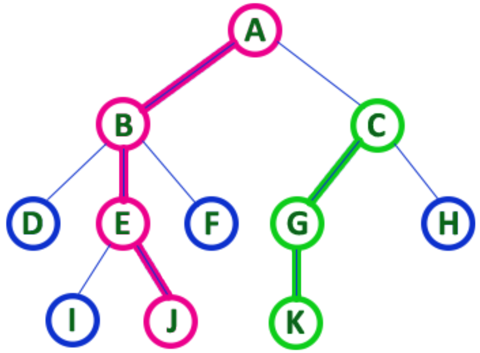
Here Height of tree is 3

- In any tree, ‘Height of Node’ is total number of Edges from leaf to that node in longest path.
- In any tree, ‘Height of Tree’ is the height of the root node.



Here Depth of tree is 3

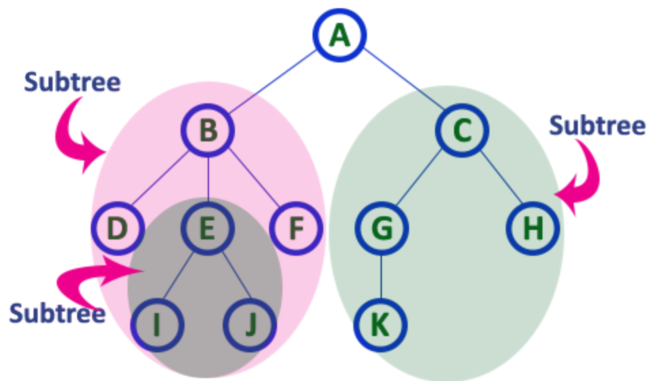
- In any tree, ‘Depth of Node’ is total number of Edges from root to that node.
- In any tree, ‘Depth of Tree’ is total number of edges from root to leaf in the longest path.



- In any tree, ‘Path’ is a sequence of nodes and edges between two nodes.

Here, ‘Path’ between A & J is
A - B - E - J

Here, ‘Path’ between C & K is
C - G - K



Tree Implementation

- Each node is part of a Linked List of siblings
- Additionally, each node stores a reference to its children

```
public class TreeNode {
    Object element;
    TreeNode firstChild;
    TreeNode nextSibling;
}
```

```
public class Tree {
    TreeNode    root;
    int size;
}
```

Tree Traversals

- A traversal is a manner of visiting each node in a tree once
- What you do when visiting any particular node depends on the application
  - Suppose we want to print all the nodes in a tree
- What order should we visit the nodes?
  - Preorder - read the parent before its children
  - Postorder - read the parent after its children

Preorder Traversal

- In a _preorder_ traversal, you visit each node before recursively visiting its children, which are visited from left to right. The root is visited first.

```
class TreeNode {
  public void preorder() {
    this.visit();
    if (firstChild != null) {
      firstChild.preorder();
    }
    if (nextSibling != null) {
      nextSibling.preorder();
    }
  }
}
```

```
         1
        / \
       /   \
      2     6
    / | \   / \
   3 4 5   7   8
```

Order of visits to node in a preorder traversal

Postorder Traversal

- In a postorder traversal, you visit each node's children (in left-to-right order) before the node itself.

```
public void postorder() {
  if (firstChild != null) {
    firstChild.postorder();
  }
  this.visit();
  if (nextSibling != null) {
    nextSibling.postorder();
  }
}
```

```
      8
     / \
    /   \
   4     7
  /|\   / \
 1 2 3 5   6
```

Order of visits to node in a postorder traversal

Binary Trees

- Nodes can only have two children:
  - left child and right child
- Simplifies implementation and logic

```
public class BinaryNode {
    Object element;
    BinaryNode left;
    BinaryNode right;
}
```

- Provides new inorder traversal

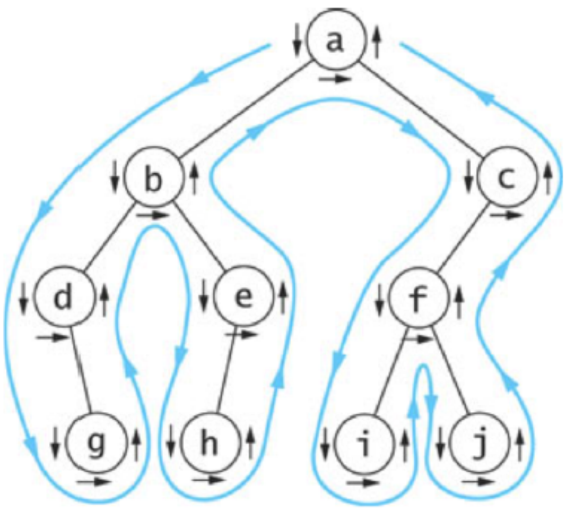
Inorder Traversal

- Read left child, then parent, then right child
- Essentially scans whole tree from left to right

```
inorder(node x)
    inorder(x.left)
    print(x)
    inorder(x.right)
```

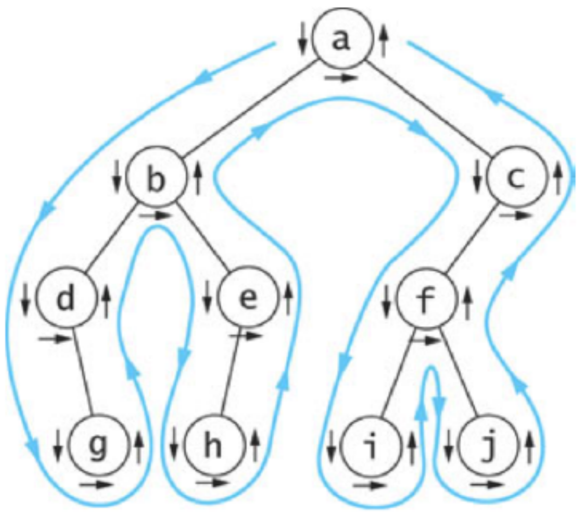
Preorder vs. Postorder

```
preorder(node x) {
    print(x);
    preorder(left);
    preorder(right);
}
```



a b d g e h c f i j

```
postorder(node x) {
    postorder(left);
    postorder(right);
    print(x);
}
```



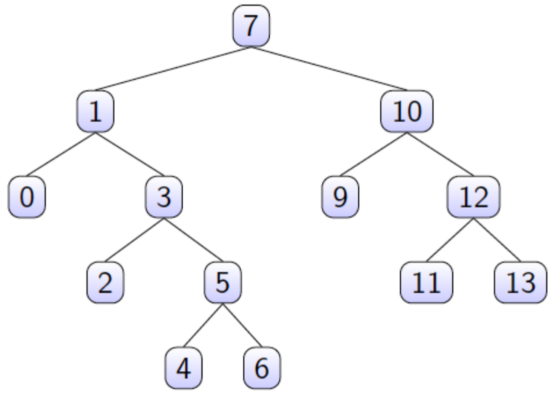
g d h e b i j f c a

Level Order Traversal

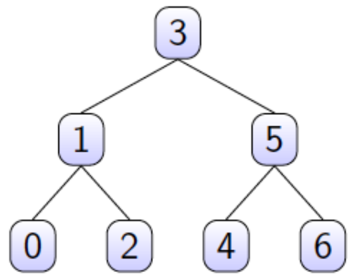
- In a level-order traversal, you visit the root (level-0), then all the level-1 nodes (from left to right), then all the level-2 nodes, etc.
- Previous example: “a b c d e f g h i j”
- O(n) time implementation
  - using a queue, initially containing only the root;
  - Then repeat the following steps:
    - Dequeue a node;
    - Visit it;
    - Enqueue its children (in order from left to right);
  - Continue until the queue is empty.

Binary Tree Properties

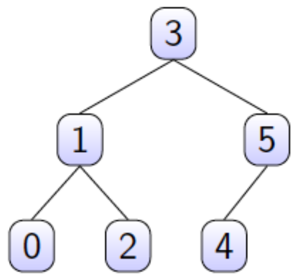
- A binary tree is **full** if each node has 2 or 0 children



- A binary tree is **perfect** if it is full and each leaf is at the same depth
  - A perfect tree of height h has 2^{h+1}-1 nodes



- A binary tree is complete if it is a perfect binary tree through level h - 1 with some extra leaf nodes at level n (the tree height), all toward the left
  - A complete tree of height, h, has between 2^h and 2^{h+1}-1 nodes.



Full Binary Tree Depth

- The number of nodes at depth d is 2^d
- Total in a tree of depth d is $\sum_{i=0}^{d} 2^i = 2^{d+1} - 1$
  - (series identity)

- A perfect binary tree has $N = 2^{d+1} - 1$ nodes
- Solving for d finds: $d = \log(N+1) - 1$

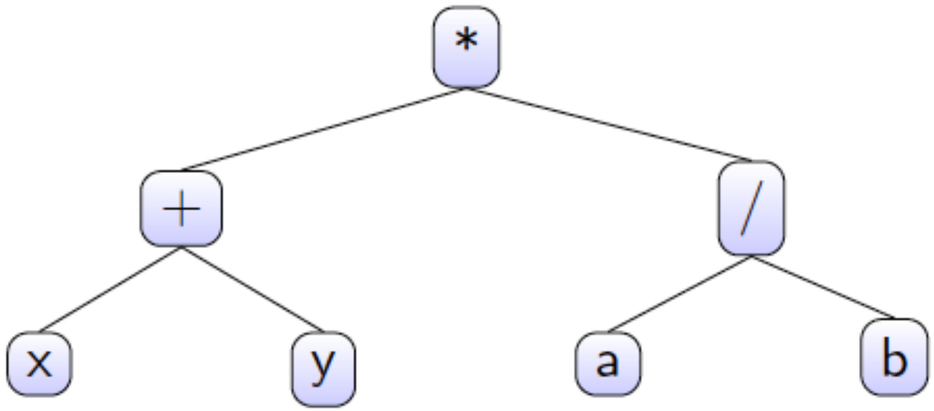
Binary Tree Questions

- What is the maximum height of a binary tree with n nodes? What is the minimum height?
- What is the minimum and maximum number of nodes in a binary tree of height h?
- What is the minimum number of nodes in a full tree of height h?

- Is a complete tree a full tree?
- Is perfect tree a full and complete tree?

Arithmetic Expression Trees

- Each node contains an operator or an operand
- Operands are stored in leaf nodes
- Parentheses are not stored in the tree because the tree structure dictates the order of operand evaluation
  - Operators in nodes at higher levels are evaluated after operators in nodes at lower levels
- Inorder traversal reads back infix notation
- Postorder traversal reads postfix notation
- Preorder traversal reads prefix notation



(x + y) * (a / b)

Decision Trees

- It is often useful to design decision trees
- Left/right child represents yes/no answers to questions