CS 310 – Advanced Data Structures and Algorithms

Tree

June 14, 2018
Tree is a fundamental data structure in computer science.

Almost all operating systems store files in trees or treelike structures.

A tree consists of a set of nodes and a set of directed edges that connect pairs of nodes.

One node is distinguished as the *root*.

Every node \( c \), except the root, is connected by an edge from exactly one other node \( p \). Node \( p \) is \( c \)'s *parent*, and \( c \) is one of \( p \)'s *children*.

A unique path traverses from the root to each node.

Node that has no children is called a *leaf*.
A tree with $N$ nodes must have $N - 1$ edges because every node except the parent has an incoming edge.

The depth of a node in a tree is the length of the path from the root to the node.
- The depth of the root is always 0
- The depth of any node is 1 more than the depth of its parent

The height of a node in a tree is the length of the path from the node to the deepest leaf.
- The height of a tree is the height of the root

Nodes with the same parent are called siblings.

If there is a path from node $u$ to node $v$, then $u$ is an ancestor of $v$ and $v$ is a descendant of $u$.

If $u \neq v$, then $u$ is a proper ancestor of $v$ and $v$ is a proper descendant of $u$. 
Textbook figure 18.1
A tree viewed recursively

Textbook figure 18.2
Hierarchical File System

Textbook figure 18.4
class Node {
    String name;
    Node parent;
    Node List<Node> children;
    public Node(String name, Node parent) {
        this.name = name;
        this.parent = parent;
        this.children = new ArrayList<Node>();
    }
}
public void bfs(Node root) {
    if (root == null) return;
    List<Node> queue = new ArrayList<>();
    queue.add(root);
    while (!queue.isEmpty()) {
        Node node = queue.remove(0);
        System.out.print(node.name + " ");
        for (Node n : node.children) {
            queue.add(n);
        }
    }
}
A binary tree is a tree in which no node can have more than two children.

Because there are only two children, we can name them left and right.

```java
public class TreeNode {
    int val;
    TreeNode left;
    TreeNode right;
    TreeNode(int x) { val = x; }
}
```
Tree Traversals

- Preorder (Root, Left, Right): 100, 19, 17, 2, 7, 3, 36, 25, 1
- Inorder (Left, Root, Right): 2, 17, 7, 19, 3, 100, 25, 36, 1
- Postorder (Left, Right, Root): 2, 7, 17, 3, 19, 25, 1, 36, 100
preorder(node)
  if (node = null)
    return
  visit(node)
  preorder(node.left)
  preorder(node.right)

iterativePreorder(node)
  if (node = null)
    return
  s ← empty stack
  s.push(node)
  while (not s.isEmpty())
    node ← s.pop()
    visit(node)
    // right child is pushed first so that left is processed first
    if (node.right ≠ null)
      s.push(node.right)
    if (node.left ≠ null)
      s.push(node.left)

From Wikipedia
**inorder(node)**

- if (node = null)
  - return
- inorder(node.left)
- visit(node)
- inorder(node.right)

**iterativeInorder(node)**

- s ← empty stack
- while (not s.isEmpty() or node ≠ null)
  - if (node ≠ null)
    - s.push(node)
    - node ← node.left
  - else
    - node ← s.pop()
    - visit(node)
    - node ← node.right

From Wikipedia
from Wikipedia

```
iteratePostorder(node)
    s ← empty stack
    lastNodeVisited ← null
    while (not s.isEmpty() or node ≠ null)
        if (node ≠ null)
            s.push(node)
            node ← node.left
        else
            peekNode ← s.peek()
            // if right child exists and traversing node
            // from left child, then move right
            if (peekNode.right ≠ null and lastNodeVisited ≠ peekNode.right)
                node ← peekNode.right
            else
                visit(peekNode)
                lastNodeVisited ← s.pop()
```

postorder(node)
    if (node = null) return
    postorder(node.left)
    postorder(node.right)
    visit(node)