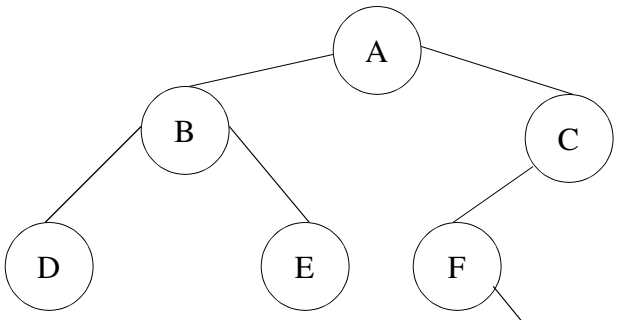
Introduction to Algorithms and Data Structures

Lecture 12 - “I think that I shall never see.. a data structure lovely as a” Binary Tree

What is a Binary Tree

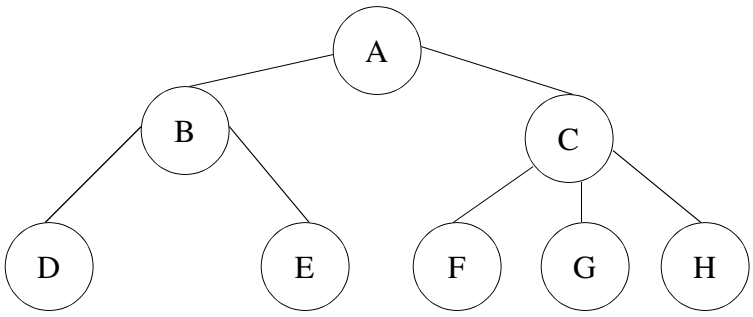
- A ***binary tree*** is a a collection of nodes that consists of the ***root*** and other subsets to the root points, which are called the ***left*** and ***right subtrees***.
- Every node on a binary tree can have up to two ***sons*** (roots of the two subtrees); any more sons and it becomes a general tree.

A Few Terms Regarding Binary Trees



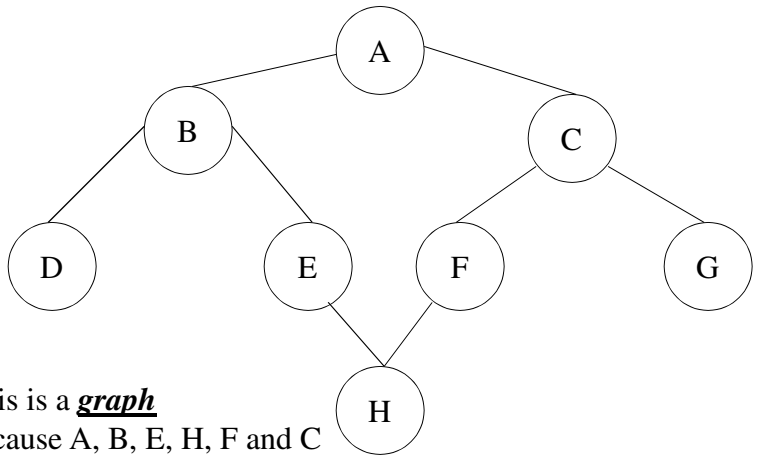
A is the root
B and C are A's son
A is B's father
B & C are the root of two subtrees
D, E, F and G are leaves

This is ***NOT*** A Binary Tree



This is a ***general tree*** because C has three sons

This is ***NOT*** A Binary Tree

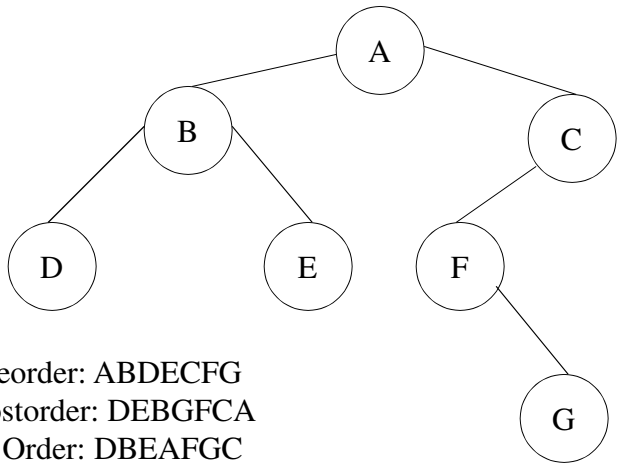


This is a ***graph***
because A, B, E, H, F and C
form a circuit

Tree Traversal

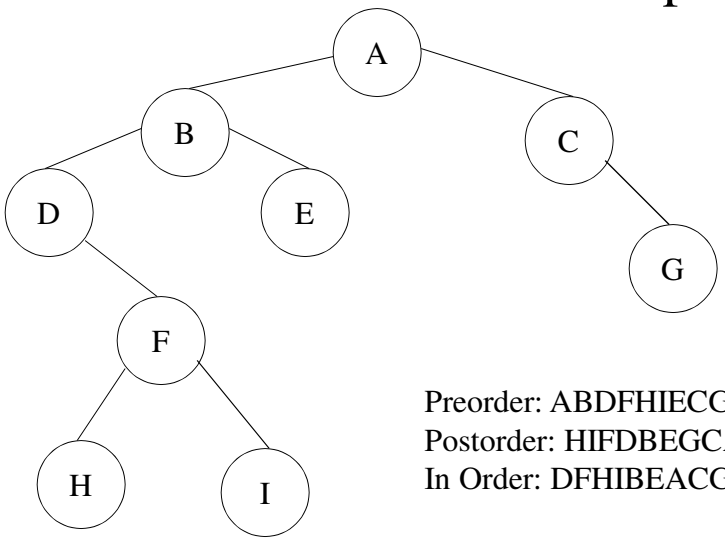
- There are three common ways to traverse a tree:
  - Preorder: Visit the root, traverse the left subtree (preorder) and then traverse the right subtree (preorder)
  - Postorder: Traverse the left subtree (postorder), traverse the right subtree (postorder) and then visit the root.
  - Inorder: Traverse the left subtree (in order), visit the root and the traverse the right subtree (in order).

Tree Traversals: An Example



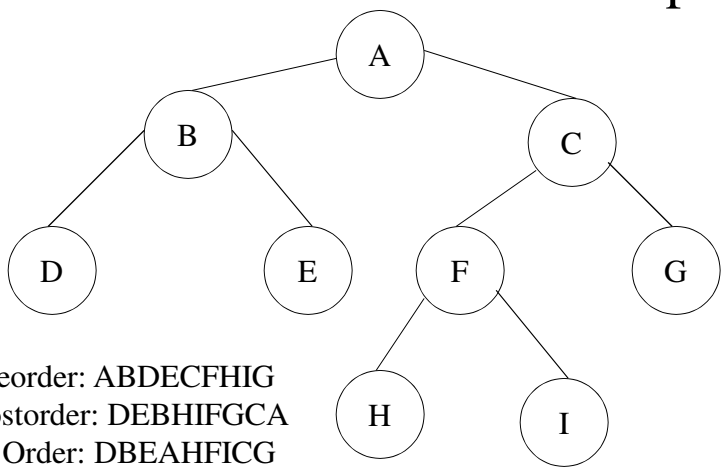
Preorder: ABDECFG
Postorder: DEBGFCA
In Order: DBEAFGC

Tree Traversals: An Example



Preorder: ABDFHIECG
Postorder: HIFDBEGCA
In Order: DFHIBEACG

Tree Traversals: An Example



Preorder: ABDECFHIG
Postorder: DEBHIFGCA
In Order: DBEAHFICG

Basic Implementation of a Binary Tree

- We can implement a binary in essentially the same way as a linked list, except that there are two nodes that comes next:

```
public class Node {
  private int data;
  private Node left;
  private Node right;
```

```
  public int getData()  {
    return data;
  }

  public Node getLeft()  {
    return left;
  }

    public Node getRight()  {
    return right;
  }

  public void setData(int x)  {
    data = x;
  }
```

```
  public void setLeft(Node p)  {
    left = p;
  }

  public void setRight(Node p)  {
    right = p;
  }


}
```

The `tree` Class

```
public class Tree {
    private Node root;

    // tree() - The default constructor - Starts
    //          the tree as empty

    public Tree() {
        root = null;
    }

    // Tree() - An initializing constructor that
    //          creates a node and places in it
    //          the initial value
```

```
    public Tree(int x) {
        root = new Node();
        root.setData(x);
        root.setLeft(null);
        root.setRight(null);
    }

    public Node getRoot() {
        return root;
    }
```

```
// newNode() -  Creates a new node with a
//              zero as data by default
public Node newNode() {
    Node   p;

    p = new Node();
    p.setData(0);
    p.setLeft(null);
    p.setRight(null);
    return(p);
}
```

```
// newNode() -  Creates a new node with the
//              parameter x as its value
public Node newNode(int x)  {
    Node   p;

    p = new Node();
    p.setData(x);
    p.setLeft(null);
    p.setRight(null);
    return(p);
}
```

```
    public void  travTree() {
        if (root != null)
          travSubtree(root);
        System.out.println();
    }

    public void  travSubtree(Node p) {
        if (p != null)  {
            travSubtree(p.getLeft());
            System.out.println(p.getData()
                               + "'\t'");
            travSubtree(p.getRight());
        }
    }
```

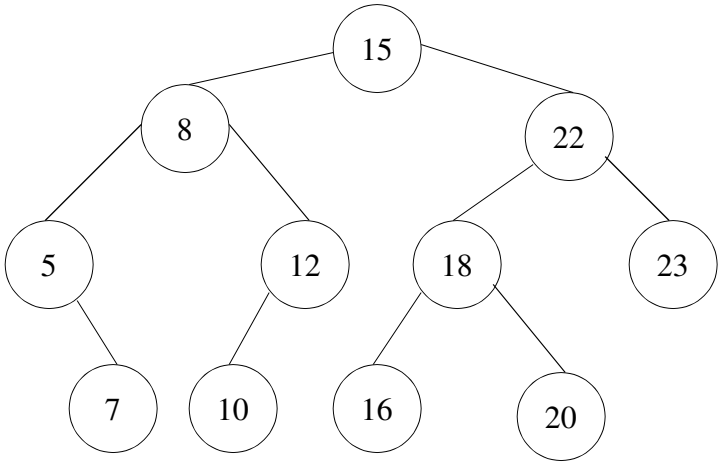
```
    // addLeft() -  Inserts a new node containing
    //              x as the left son of p
    public void  addLeft(Node p, int x) {
        Node   q = newNode(x);
        p.setLeft(q);
    }

    public void  addRight(Node p, int x) {
        Node   q = newNode(x);
        p.setRight(q);
    }
}
```

A Basic Search Tree

- We can construct a simple search tree if we add new nodes with value x on the tree using this strategy:
  - Every time x is less than the value in the node we move down to the left.
  - Every time x is greater than the value in the node we move down to the right.

A Sample Search Tree



```
// insert() - Insert value x in a new node to
//                          be inserted after p
public void   insert(int x)  {
    Node      p, q;
    boolean   found = false;

    p = root;
    q = null;

    while (p != null && !found) {
        q = p;
        if (p.getData() == x)
          found = true;
        else if (p.getData() > x)
          p = p.getLeft();
        else
          p = p.getRight();
    }
```

```
    if (found)
      error("Duplicate entry");

    if (q.getData() > x)
      addLeft(q, x);
    else
      addRight(q, x);

    q = newNode(x);
}
```

```
// isXThere() -     Is there a node on the
//                  list containing x?
public boolean      isXThere(int x)  {
    Node      p;
    boolean   found = false;

    p = root;
    while (p != null && !found) {

        if (p.getData() == x)
          found = true;
        else if (p.getData() > x)
          p = p.getLeft();
        else
          p = p.getRight();
    }
    return(found);
}
```

```
public void   error(String message) {
  System.out.println(message);
  System.exit(0);
}
```

```
// getNode() -      Get the pointer for the
//                  node containing x
public Node   getNode(int x) {
    Node      p, q;
    boolean   found = false;

    p = root;
    q = null;
    while (p != null && !found) {
        q = p;
        if (p.getData() == x)
          found = true;
        else if (p.getData() > x)
          p = p.getLeft();
        else
          p = p.getRight();
  }
```

```
        if (found)
          return(q);
        else
          return(null);
}
```

```
public class UseTree {
    public static void main(String[] args) {
        Tree      mytree = new Tree(8);
        mytree.addLeft(mytree.getRoot(), 6);
        mytree.addRight(mytree.getRoot(), 9);
        mytree.insert(4);
        mytree.insert(1);
        mytree.insert(12);

        if (mytree.isXThere(13))
          System.out.println("great");
        else System.out.println("not so great");
        mytree.travTree();
    }
}
```