CPSC 319

Data Structures, Algorithms and Their Application

Department of Computer Science

Created by:

Fahim Anzum

Winter 2020

Course Information

- **Course website:**

All course materials, such as lecture slides, important dates, assignments and exercises can be found on the course website, which is located at http://pages.cpsc.ucalgary.ca/~hudsonj/CPSC319W20/

- **For Assignment and Grading:**

D2L will be used for submitting assignments and reporting grades.

https://d2l.ucalgary.ca/d2l/home

- **Tutorial Contents:**

Will be uploaded in the following site:

https://sites.google.com/view/fahimanzum/courses

UNIVERSITY OF CALGARY

Contents

- Comparable interface in java
- Ordered tree, Binary tree
- Tree traversals
  - Pre-order traversal
  - Post-order traversal
  - In-order traversal
  - Level-order traversal
- Binary tree representation in java
- Tree traversals in java
- Recursive search in binary trees using java
- Practice

UNIVERSITY OF CALGARY

Comparable Interface in Java

- Comparable interface is used to sort objects using data members of the class
- A comparable object is capable of comparing itself with another object
- The class itself must implement the java.lang.Comparable interface to compare its instances

UNIVERSITY OF CALGARY

Comparable Interface in Java

- Consider a Movie class that has members like, rating, name, year
- Suppose we wish to sort a list of Movies based on year of release
- We can implement the Comparable interface with the Movie class, and we override the method compareTo() of Comparable interface

Example

- Implementing Comparable interface with the Movie class
- Overriding the compareTo() method of Comparable interface

```
public class Movie implements Comparable <Movie>{

    private double rating;
    private String name;
    private int year;

    // Constructor
    public Movie (String name, double rt, int yr) {
        this.name = name;
        this.rating = rt;
        this.year = yr;
    }


    // Sort movies by year
    public int compareTo(Movie m) {
        return this.year - m.year;
    }

    // Getter methods to access private data
    public double getRating() {
        return rating;
    }

    public String getName() {
        return name;
    }

    public int getYear() {
        return year;
    }

}
```

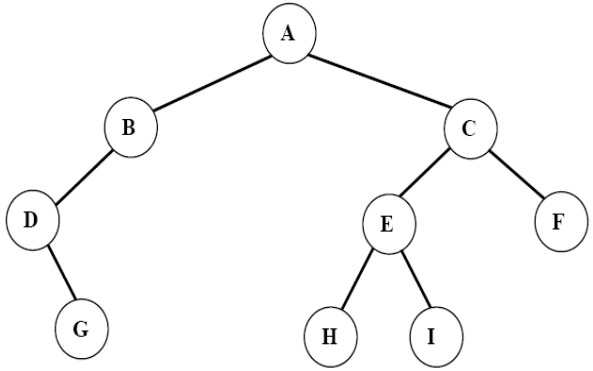
Example

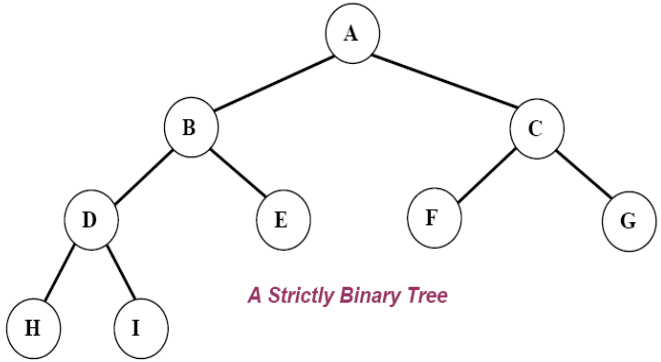
- Main class
- Creating a list of Movies having members like Name, Rating and Year of release
- Using Collections.sort() method to sort the movies based the criteria given in the Movie class

```
public class Main {

    public static void main(String[] args) {
        // TODO Auto-generated method stub

        ArrayList <Movie> list = new ArrayList <Movie> ();

        list.add(new Movie("Force Awakens", 8.3, 2015));
        list.add(new Movie("Star Wars", 8.7, 1977));
        list.add(new Movie("Empire Strikes Back", 8.8, 1980));
        list.add(new Movie("Return of the Jedi", 8.4, 1983));
        list.add(new Movie("Forrest Gump", 8.8, 1994));

        // Collections.sort() sorts the elements
        // present in the specified list of collection
        // in ascending order
        Collections.sort(list);

        System.out.println("Movies after sorting: ");

        for(Movie movie:list) {
            System.out.println(movie.getName() + " " + movie.getRating() + " " + movie.getYear());
        }

    }

}
```

Ordered Trees

- A tree is ordered if there is a linear ordering defined for each child of each node.
- A binary tree is an ordered tree in which every node has at most two children.
- If each node of a tree has either zero or two children, the tree is called a proper (strictly) binary tree.





A Strictly Binary Tree

UNIVERSITY OF CALGARY

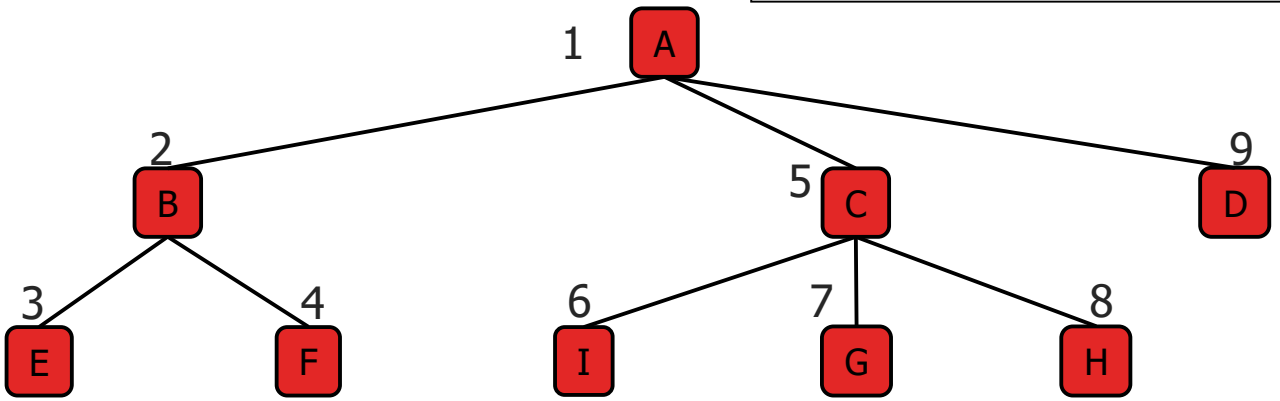
Traversal of Trees

- A traversal of a tree T is a systematic way of visiting all the nodes of T
- Traversing a tree involves visiting the root and traversing its subtrees
- There are the following traversal methods:
  - Preorder Traversal
  - Postorder Traversal
  - Inorder Traversal (of a binary tree)

Preorder Traversal

- In a preorder traversal, a node is visited before its descendants
- If a tree is ordered, then the subtrees are traversed according to the order of the children

Algorithm *preOrder*(*v*)
    visit(*v*)
    for each child *w* of *v*
        preorder (*w*)

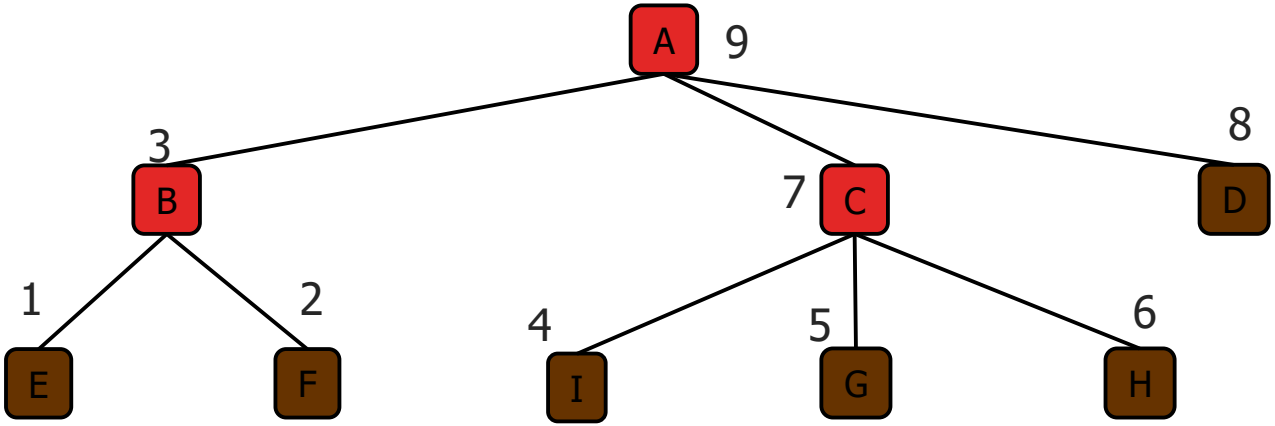


Preorder: *ABEFCIGHD*

Postorder Traversal

- In a postorder traversal, a node is visited after its descendants

Algorithm *postOrder*(*v*)
  for each child *w* of *v*
    postOrder (*w*)
  visit(*v*)

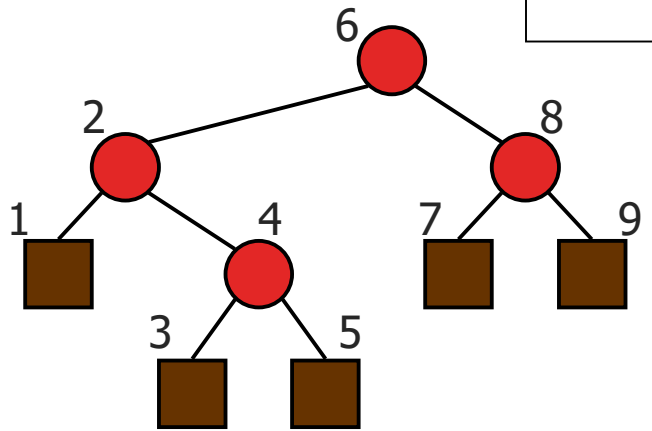


Postorder: ***EFBIGHCDA***

Inorder Traversal

- In an inorder traversal a node is visited after its left subtree and before its right subtree

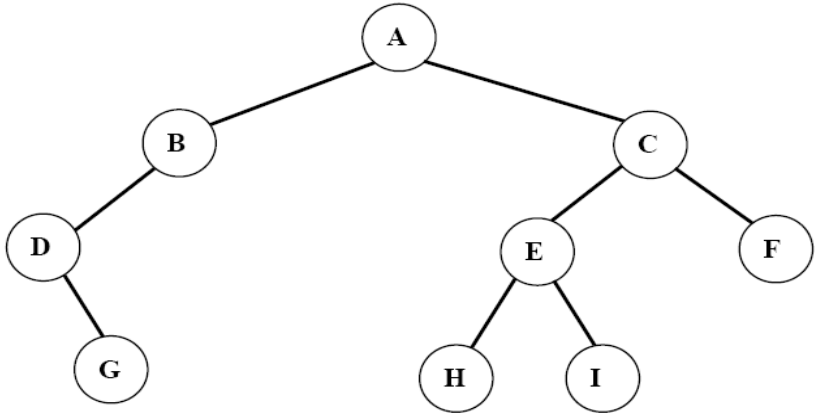
```
Algorithm inOrder(v)
    if isInternal (v)
        inOrder (leftChild (v))
    visit(v)
    if isInternal (v)
        inOrder (rightChild (v))
```



Inorder Traversal

Traversing a binary tree in *inorder*

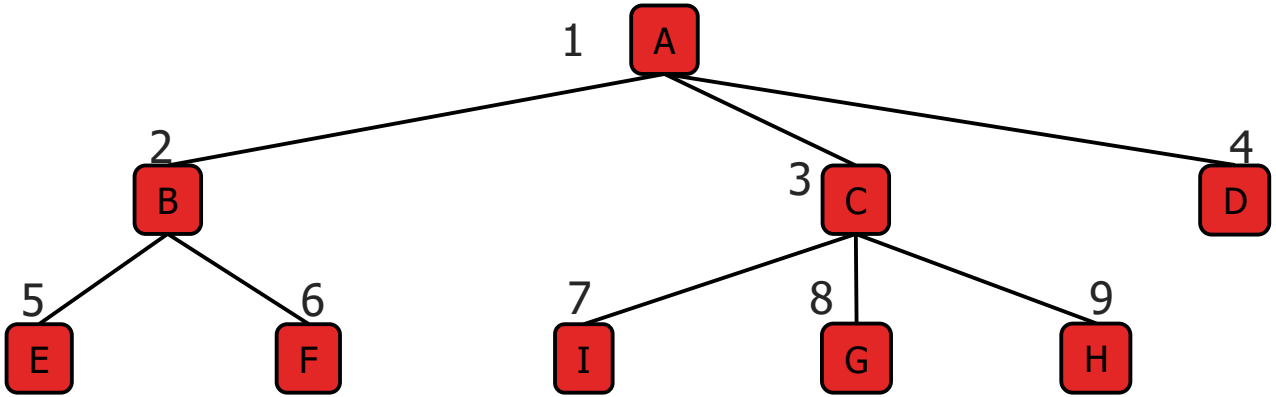
1. Traverse the ***left subtree*** in inorder.
2. Visit the ***root***.
3. Traverse the ***right subtree*** in inorder.



Inorder*: DGBAHEICF*

Level Order Traversal

- In a level order traversal, every node on a level is visited before going to a lower level



Level order*: ABCDEFIGH*

Binary Tree Representation in Java

```java
/* Class containing left and right child of current
node and key value*/
class Node
{
    int key;
    Node left, right;

    public Node(int item)
    {
        key = item;
        left = right = null;
    }
}
```

Binary Tree Representation in Java

```
// A Java program to introduce Binary Tree
class BinaryTree
{
    // Root of Binary Tree
    Node root;
    // Constructors
    BinaryTree(int key)
    {
        root = new Node(key);
    }
    BinaryTree()
    {
        root = null;
    }
    public static void main(String[] args)
    {
        BinaryTree tree = new BinaryTree();

        /*create root*/
        tree.root = new Node(1);

        /* following is the tree after above statement

            1
            / \
        null null    */
```

Binary Tree Representation in Java

```
        tree.root.left = new Node(2);
        tree.root.right = new Node(3);

        /* 2 and 3 become left and right children of 1
            1
            / \
            2    3
        / \ / \
        null null null null */


        tree.root.left.left = new Node(4);
        /* 4 becomes left child of 2
                    1
                /    \
            2         3
            / \  / \
            4 null null null
        / \
        null null
        */
    }
}
```

Tree Traversals

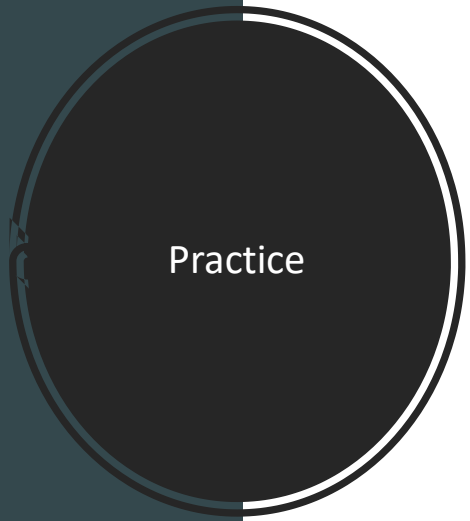
```
/* Given a binary tree, print its nodes according to the
"bottom-up" post-order traversal. */
void printPostorder(Node node)
{
    if (node == null)
        return;

    // first recur on left subtree
    printPostorder(node.left);

    // then recur on right subtree
    printPostorder(node.right);

    // now deal with the node
    System.out.print(node.key + " ");
}

/* Given a binary tree, print its nodes in in-order*/
void printInorder(Node node)
{
    if (node == null)
        return;

    /* first recur on left child */
    printInorder(node.left);

    /* then print the data of node */
    System.out.print(node.key + " ");

    /* now recur on right child */
    printInorder(node.right);
}
```

Tree Traversals

```java
/* Given a binary tree, print its nodes in pre-order*/
void printPreorder(Node node)
{
    if (node == null)
        return;

    /* first print data of node */
    System.out.print(node.key + " ");

    /* then recur on left subtree */
    printPreorder(node.left);

    /* now recur on right subtree */
    printPreorder(node.right);
}

// Wrappers over above recursive functions
void printPostorder() {  printPostorder(root); }
void printInorder() {    printInorder(root); }
void printPreorder() {   printPreorder(root); }

// Driver method
public static void main(String[] args)
{
    BinaryTree tree = new BinaryTree();
    tree.root = new Node(1);
    tree.root.left = new Node(2);
    tree.root.right = new Node(3);
    tree.root.left.left = new Node(4);
    tree.root.left.right = new Node(5);

    System.out.println("Preorder traversal of binary tree is "); tree.printPreorder();
    System.out.println("\nInorder traversal of binary tree is "); tree.printInorder();
    System.out.println("\nPostorder traversal of binary tree is "); tree.printPostorder();
}
```

Recursive Search in Tree

```java
// Do any of the tree traversal and check
// if the given element is present
public static boolean isPresent(Node root, int x) {

    if(root != null) {

        // check if current node has the element we are
        // looking for
        if(root.key == x) {
            return true;
        }

        else {
            // check the subtrees
            return isPresent(root.left, x) || isPresent(root.right, x);
        }
    }

    return false;
}
```



- Write a java program that takes user input as follows:
  10
  1 2 3 4 5 6 7 8 9 10
- Here the first line of the input represents the number of nodes in the binary tree
- The second line represents the nodes of the binary tree
- Write a function to construct the binary tree
- Write a function that finds the Maximum in binary tree. For the above input, the output should be 10
- Write a function that finds the Minimum the binary tree. For the above input, the output should be 1

UNIVERSITY OF CALGARY

Thank You

Fahim Anzum

Email: fahim.anzum@ucalgary.ca