CMSC 132, Object-Oriented Programming II **Summer 2015**

Lecture 13

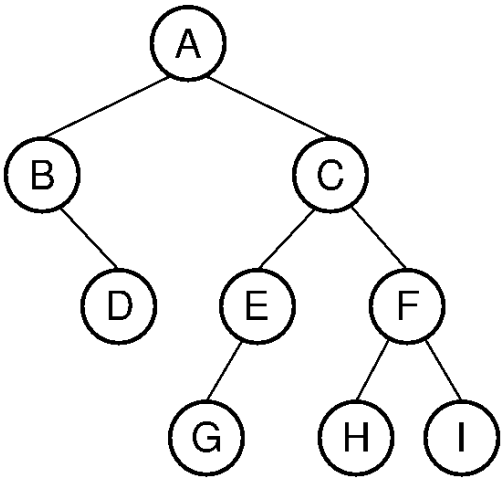
Lecturer: Anwar Mamat

Disclaimer: *These notes may be distributed outside this class only with the permission of the Instructor.*

13.1 Trees

13.1.1 Binary Tree

A binary tree is a tree data structure in which each node has at most two children, which are referred to as the left child and the right child.



- Tree Traversal: tree traversal refers to the process of visiting (examining and/or updating) each node in a tree data structure, exactly once, in a systematic way.
  - preOrder:root, left child, right child: A,B,D,C,E,G,G,H,I
  - inOrder: left child, root, right child : B,D,A,G,E,C,H,F,I
  - postOrder:left child, right child, root: D,B,G,E,H,I,F,C,A
  - levelOrder:A,B,C,D,E,F,G,G,I

Listing 1: Binary Tree

```
1 /**
2  *              2
3  *          /       \
4  *      7                5
5  *   /    \                 \
6  *  2      6                  9
```

```
7  *      /  \            /
8  *     5    11         4
9  *
10 */
11
12 package binarytree;
13
14 import java.util.LinkedList;
15 import java.util.Queue;
16 import java.util.Stack;
17 public class BinaryTree {
18
19     private Node root;              // root of binary tree
20     Stack<Node> pathStack;
21     BinaryTree(){
22         pathStack = new Stack();
23         create();
24     }
25     private class Node {
26         private Integer key;            // sorted by key
27         private Node left, right;  // left and right subtrees
28
29         public Node(Integer key) {
30             this.key = key;
31             left = null;
32             right = null;
33
34         }
35     }
36
37     private void create(){
38         root = new Node(2);
39         root.left = new Node(7);
40
41         root.left.left = new Node(2);
42         root.left.right = new Node(6);
43         root.left.right.left = new Node(5);
44         root.left.right.right = new Node(11);
45
46         root.right = new Node(5);
47         root.right.right = new Node(9);
48         root.right.right.left = new Node(4);
49
50     }
51     /*
52      * preOrder traverse the binary tree
53      */
54     public void preOrder(){
55         preOrderTraverseIterate(root);
56     }
57     /*
```

```
58          * Preorder traverse the binary tree starting from a node
59          */
60         public void preOrder(Node r){
61             if(r == null){return;}
62             System.out.print(r.key+",");
63             preOrder(r.left);
64             preOrder(r.right);
65         }
66
67         /*
68          * inOrder traverse the binary tree
69          */
70         public void inOrder(){
71             inOrderTraverseIterate(root);
72         }
73         /*
74          * inOrder traverse the binary tree starting from a node
75          */
76         public void inOrder(Node r){
77             if(r == null){return;}
78             inOrder(r.left);
79             System.out.print(r.key+",");
80             inOrder(r.right);
81         }
82
83         /*
84          * postOrder traverse the binary tree
85          */
86         public void postOrder(){
87             postOrder(root);
88         }
89         /*
90          * postOrder traverse the binary tree starting from a node
91          */
92         public void postOrder(Node r){
93             if(r == null){return;}
94             postOrder(r.left);
95             postOrder(r.right);
96             System.out.print(r.key+",");
97         }
98
99
100        private void preOrderTraverseIterate(Node r){
101            Stack<Node> S = new Stack();
102            Node current = root;
103            while(current != null || !S.empty()){
104                while(current != null){
105                    System.out.print(current.key +"->");
106                    S.push(current.right);
107                    current = current.left;
108                }
```

```
109             if(!S.empty()){
110                 current = S.pop();
111             }
112         }
113     }
114     private void inOrderTraverseIterate(Node r){
115     Stack<Node> S = new Stack();
116     Node current = root;
117     while(current != null  || !S.empty()){
118         if(current != null){
119            S.push(current);
120            current = current.left;
121         }else{
122             current = S.pop();
123             System.out.print(current.key+ "->");
124             current = current.right;
125         }
126     }
127 }
128
129     // is the symbol table empty?
130     public boolean isEmpty() {
131         return root == null;
132     }
133
134     // return number of nodes in the tree
135     public int size() {
136         return size(root);
137     }
138     // return the number of nodes in a subtree rooted at r
139     private int size(Node r){
140         if(r == null) return 0;
141         return 1 + size(r.left )+ size(r.right);
142     }
143
144
145     // height of this binary tree (one-node tree has height 0)
146     public int height() {
147         return height(root);
148     }
149     // height of a tree rooted at r
150     private int height(Node r) {
151         if (r == null) return -1;
152         return 1 + Math.max(height(r.left), height(r.right));
153     }
154
155     //return the number of leaves
156     public int countLeaves(){
157         return countLeaves(root);
158     }
159
```

```
160     //return the number of leaves of a rooted at r
161     public int countLeaves(Node r){
162         if(r == null) return 0;
163         if(r.left == null && r.right == null) return 1;
164         return countLeaves(r.left) + countLeaves(r.right) ;
165     }
166
167    //return diameter of the tree
168    public int diameter(){
169        return diameter(root);
170    }
171    //return diameter of the tree rooted at r
172    public int diameter(Node r){
173        if (r == null) return 0;
174        int lheight = height(r.left)+1;
175        int rheight = height(r.right)+1;
176        int ldiameter = diameter(r.left);
177        int rdiameter = diameter(r.right);
178        return Math.max(lheight + rheight + 1, Math.max(ldiameter,rdiameter));
179     }
180
181     public void levelOrder(){
182         if(root == null){ return; }
183         Queue<Node> q = new LinkedList();
184         q.offer(root);
185         while(!q.isEmpty()){
186             Node t = q.poll();
187             System.out.print(t.key+",");
188             if(t.left != null) q.offer(t.left);
189             if(t.right != null) q.offer(t.right);
190         }
191     }
192
193
194     public void path(Integer k){
195         path(root,k);
196     }
197     public void path(Node r, Integer k){
198         if(r == null) return;
199         pathStack.push(r);
200         if(r.key.equals(k)) {
201             System.out.println("\nPath to " + k +":" );
202             for(Node tt: pathStack){
203                 System.out.print(tt.key+"-->");
204             }
205             System.out.println("\n");
206             return;
207         }
208         path(r.left,k);
209         path(r.right,k);
210         pathStack.pop();
```

```
211     }
212
213
214
215     /**
216      * @param args the command line arguments
217      */
218     public static void main(String[] args) {
219         BinaryTree b = new BinaryTree();
220         System.out.println("preOrder traversal:");
221         b.preOrder();
222         System.out.println("\ninOrder traversal:");
223         b.inOrder();
224         System.out.println("\npostOrder traversal:");
225         b.postOrder();
226         System.out.println("\n");
227         System.out.println("Size:" + b.size());
228         System.out.println("Height:" + b.height());
229         System.out.println("Number of leaves:" + b.countLeaves());
230         System.out.println("Diameter:" + b.diameter());
231         System.out.println("Level Order:");
232         b.levelOrder();
233         b.path(12);
234     }
235 }
```