CS211, Lecture 18: Trees

Announcements:

- Partners?
- A4 and A6...

Reading:

- DS&SD: Chap 13.1-13.3.2; 13.3.4-13.4
- DS&A: Chap 9

Overview:

- foundation data structures continued!
- motivation for trees
- node and tree classes
- binary trees
- general trees
1. Motivation

- Dynamic vs. Static
  - arrays
  - lists
- Linear vs. Hierarchical
  - lists
  - trees
  - no cycles (graphs are later)
- Examples of Trees
  - directories, file systems
  - hierarchy of people
  - more? see Lecture 7
- Want more generic tree classes for ADTs
2. Tree Definitions (reminders)

Summarized From Lecture 7:

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Node, Point, Vertex</td>
<td>a state or position</td>
</tr>
<tr>
<td>Line, Edge</td>
<td>connection between nodes</td>
</tr>
<tr>
<td>Graph</td>
<td>set of vertices (also called nodes, points) and set of edges (also called lines, arcs) with each edge connecting a pair of vertices; nodes can connect to any other node</td>
</tr>
<tr>
<td>Path</td>
<td>sequence of edges in which each edge connects to another edge at each vertex</td>
</tr>
<tr>
<td>Tree</td>
<td>a graph in which every pair of vertices has a unique path</td>
</tr>
<tr>
<td>Rooted Tree</td>
<td>The &quot;top&quot; node has no “incoming” edges (no parent); all other nodes have a path to the root; a rooted tree is usually just called tree</td>
</tr>
<tr>
<td>Root</td>
<td>&quot;topmost&quot; node to which all other nodes have a path</td>
</tr>
<tr>
<td>Parent</td>
<td>node &quot;above&quot; another node, leading towards the root</td>
</tr>
<tr>
<td>Child</td>
<td>node &quot;below&quot; another node, leading away from root</td>
</tr>
<tr>
<td>Internal Node</td>
<td>node that has a parent and at least one child</td>
</tr>
<tr>
<td>Leaf</td>
<td>child node with no children</td>
</tr>
<tr>
<td>Sibling</td>
<td>node at the same level as another</td>
</tr>
<tr>
<td>Level, Depth</td>
<td>length of the path from the root to a node; root starts at level 0 (DS&amp;SD say 1)</td>
</tr>
<tr>
<td>Height</td>
<td>height of tree: longest path from root to leaf</td>
</tr>
<tr>
<td></td>
<td>height of node: longest path length from the node to a leaf</td>
</tr>
</tbody>
</table>
3. Some Design Decisions

3.1 “singly-linked” nodes (links for children only)

- parent links?
- sibling links?

3.2 overall design
3.3 Java structure: the gist

class TreeClass {
    private TreeNode root;

    // constructors
    // getters, setters
    // analysis of ENTIRE Tree
}

class TreeNode { // <-- make inner class of TreeClass?
    private Object item;
    private TreeNode n1;
    private TreeNode n2;

    // constructors
    // getters, setters
    // analys of THIS subtree
}

public class Driver {
    public static void main(String[] args) {

        // build tree (see t in 3.2)
        // add/del nodes & leaves
        // search/sort tree

    }
}
4. Tree Class

☐ Name:
  • flexible
  • usually BinaryTree, UnaryTree, NaryTree, ...

☐ Fields:
  • root: “top” of tree

☐ Methods:
  • building
  • printing
  • traversing
  • height
  • comparing
  • searching (next lecture: search structures)

☐ Interfaces?
  • see DS&SD LinkedSimpleTreeUOS and DS&A pg. 261 for examples
  • lots of disagreement in refs about what’s crucial
  • see Methods above for generally accepted operations
5. Node Classes

5.1 General Tree

“random” (dynamic) number of nodes:

```java
public class GeneralNode {
    private Object item; // stored value
    private LinkedList nodes; // list of nodes
    private int degree; // # of nodes
    // more stuff...
}
```
5.2 N-ary Tree

fixed number of nodes:

```java
public class NaryNode {
    private Object item; // stored value
    private NaryTree[] nodes; // array of nodes
    private int degree; // # of nodes
    // more stuff...
}
```
5.3 Binary Tree

2 nodes:

```java
public class BinaryNode {
    private Object item;  // stored value
    private BinaryNode left;  // left node
    private BinaryNode right;  // right node
    // more stuff...
}
```
6. Binary Tree Focus

6.1 Why?

- clarity of concepts (design, traversal, …)
- applications:
  - expression parsing (see A3)
  - decision trees
- search techniques (search structures)

6.2 general trees can be made from binary trees

Algorithm:

- convert general tree to level-order order tree
- for a node, left ← child, right ← sibling
7. BinaryNode Class

7.1 Basics

public class BinaryNode {

    private Object item;
    private BinaryNode left;
    private BinaryNode right;

    public BinaryNode() {
        this(null,null,null);
    }
    public BinaryNode(Object item) {
        this(item,null,null);
    }
    public BinaryNode(Object item, BinaryNode left,
                        BinaryNode right) {
        this.item=item;
        this.left=left;
        this.right=right;
    }

    public Object getItem() { return item; }
    public BinaryNode getLeft() { return left; }
    public BinaryNode getRight() { return right; }
    public void setItem(Object o) { item = o; }
    public void setLeft(BinaryNode n) { left = n; }
    public void setRight(BinaryNode n) { right = n; }

    // more methods
}

}
7.2 toString

- why: how should a binary tree appear?
- Algorithm:
  pre/post/in/level order?
  - stringify item, stringify children
  - stringification of children stringifies their children
- Code:

```java
// using pre-order for clarity:
public String toString() {
    return "" + _____ + "(" + _____ + "," + _______ + ")";
}
```
7.3 toTree

- why? like toString, help with debugging
- Algorithm:
  \[ \text{tree} \leftarrow \text{root} \text{ of subtree (or leaf if no more tree)} \]
  if left isn’t empty,
    \[ \text{tree} += \text{spacing} + \left. \text{left.toString}() \right] \]
  if right isn’t empty,
    \[ \text{tree} += \text{spacing} + \text{right.toString}() \]
- Code:

```java
// Start recursion:
public String toTree() {
    return toTree("|   ","|___");
}

// Build tree:
public String toTree(String blank,String spacing)
{
    String s = item.toString() + "\n";
    if (left != null)
        s += spacing +
            left.toTree(blank, blank+spacing);
    if (right != null)
        s += spacing +
            right.toTree(blank, blank+spacing);
    return s;
}
```
7.4 getHeight

- Why? help with analysis and search
- Algorithm (also recursive):
  height of leaf is 0
  height of children is
    \[ 1 + \max( \text{height} (\text{left}), \text{height} (\text{right})) \]
- Code:

```java
public int getHeight() {
    return getHeight(this);
}

private int getHeight(BinaryNode node) {
    // bottom of tree:
    if (node == null)
        return 0;

    // node to children adds 1 to height of node:
    else
        return 1 + Math.max(getHeight(node.left),
                             getHeight(node.right));
}
```
7.5 Saving For later/another time

- merging
- comparison
- `equals`
- traversing (using iterators for trees: needs stacks, queues)
- API: `TreeMap`, `TreeSet`
- parent links in tree
### 7.6 Demo

```java
class TestBinaryTree {
    public static void main(String[] args) {
        // I built this tree in Lecture 7:

        /*
         *      0
         *     1   2
         *   3   4   5  6
         *
         */

        System.out.println(t0);
        System.out.println(t0.toTree());
    }
}

/* Output:

0(1(3(null,null),4(null,null)),2(5(null,null),
6(null,null)))
0
  1
  |__3
  |   |__4
  |__2
    |__5
    |__6

*/
```
8. General Tree

- General (dynamic), N-ary (kind of static)
- showing general here

```java
import java.util.*;
public class GeneralNode {
   private Object item;
   private LinkedList nodeList;

   public GeneralNode() {
      nodeList = new LinkedList();
   }
   public GeneralNode(Object item) {
      this.item = item;
      if (nodeList == null)
         nodeList = new LinkedList();
   }
   public void addNode(GeneralNode node) {
      nodeList.add(node);
   }

   public String toString() {
      String s = item + "=";
      Iterator it = nodeList.iterator();
      if (it.hasNext()) s += "(";
      while(it.hasNext()) {
         s += it.next() + "",
      }
      return s;
   }
}
```
public String toTree() {
    return toTree("|   ", "|___");
}

public String toTree(String blank, String spacing) {
    String t = item.toString() + "\n";
    for (Iterator it = nodeList.iterator(); it.hasNext(); ) {
        GeneralNode next = (GeneralNode) it.next();
        t += spacing +
        (next.toTree(blank, blank+spacing));
    }
    return t;
}

} // Class GeneralNode

public class GeneralTree {

    private GeneralNode root;
    public GeneralTree(GeneralNode node) {
        root = node;
    }

    public String toString() {
        return root.toString();
    }

    public String toTree() {
        return root.toTree();
    }

}
public class TestGeneralTree {
    public static void main( String [ ] args ) {
        GeneralNode b0 = new GeneralNode("0");
        GeneralTree t0 = new GeneralTree(b0);
        GeneralNode b1 = new GeneralNode("1");
        GeneralNode b2 = new GeneralNode("2");
        GeneralNode b3 = new GeneralNode("3");
        GeneralNode b4 = new GeneralNode("4");
        GeneralNode b5 = new GeneralNode("5");
        GeneralNode b6 = new GeneralNode("6");
        GeneralNode b7 = new GeneralNode("7");
        GeneralNode b8 = new GeneralNode("8");
        GeneralNode b9 = new GeneralNode("9");

        b0.addNode(b1);
        b0.addNode(b2);
        b0.addNode(b3);
        b1.addNode(b4);
        b4.addNode(b5);
        b5.addNode(b6);
        b5.addNode(b7);
        b5.addNode(b8);
        b5.addNode(b9);

        System.out.println(t0);
        System.out.println(t0.toTree());
    }
}
9. Suggested Exercises

Write method(s) to get the depth of any node.

Why is there a need for the `merge` method in `BinaryTree`? That is, why shouldn’t we just provide the constructor? See 13.3.1 in DS&SD:

```java
public BinaryTree(Object item, BinaryTree t1,
                 BinaryTree t2) {
    root = new BinaryNode(item);
    root.setLeft(t1.root);
    root.setRight(t2.root);
}
```

Write method(s) to get the number of nodes for any subtree in a binary tree.

Write a program (or collection of methods) that converts a general tree to a binary tree.

Add `getHeight` and setters/getters to the general tree classes.