CS211, Lecture 18 Trees

I think that I shall never see
A poem as lovely as a tree
A tree whose hungry mouth is prest
Against the earth’s sweet flowing breast
A tree that looks to God all day,
And lifts her leafy arms to pray;
A tree that may in summer wear
A nest of robins in her hair;
Upon whose bosom snow has lain;
Who intimately lives with rain.
Poems are made by fools like me,
But only God can make a tree.

Joyce Kilmer

Readings: Weiss, chapter 18, sections 18.1--18.3.

About assignment A5
Functional programming

Write an implementation of linked list using only recursion — no assignment statement or loops.
Implement sets using it — again without assignment or loops.

Of course, when using the implementation, you can use assignments.

```java
public class RList {
    public Object value;
    public RList next;
}
```

Append one list to another

```java
/** = l1 with l2 appended to it */
public static RList append(RList l1, RList l2) {
    if (l1 == null)  return l2;
    return new RList(l1.value,         append(l1.next, l2)                                 );
}
```

Overview of trees

**Tree**: recursive data structure.

Definition 1:
A tree is a
(1) a value (the root value)

**root**

**parent**

**children**

(2) one or more other trees, called its children.

Each of the circles is called a **node** of the tree. Definition does not allow for empty trees (trees with 0 nodes).

**Binary tree**

**Binary tree**: tree in which each node can have at most two children.

Redefinition of binary tree to allow empty tree

A binary tree is either
(1) Ø (the empty binary tree)

or
(2) a root node (with a value),
    a left binary tree,
    a right binary tree
• Edge A→B: A parent of B.
  B is child of A.
• Generalization of parent and child:
  ancestor and descendant
  – root and A are ancestors of B
• Leaf node: node with no descendants
  (or empty descendents)
• Depth of node: length of path
  from root to that node
  – depth(A) = 1  depth(B) = 2
• Height of node: length of longest
  path from node to leaf
  – height(A) = 1  height(B) = 0
• Height of tree = height of root
  – in example, height of tree = 2

/\ Terminology

/\ Class for binary tree nodes

/** An instance is a nonempty binary tree */
public class TreeNode {
    private Object datum;
    private TreeNode left;
    private TreeNode right;
/** Constructor: a one-node tree with root value ob */
    public TreeNode(Object ob) {
        datum = ob;
    }
/** Constructor: tree with root value ob and left and
right subtrees l and r */
    public TreeNode(Object ob, TreeNode l, TreeNode r) {
        datum = ob;
        left = l;
        right = r;
    }
/**getter and setter methods for all three fields**
}
empty tree is given by
value null

/\ Class for general trees

public class GTreeNode {
    private Object datum;
    private GTreeNode left;
    private GTreeNode sibling;
appropriate constructors and
getter and setter methods
}

• Parent node points directly
  only to its leftmost child.
• Leftmost child has pointer to
  next sibling, which points
to next sibling etc.

/\ One application of trees

• Most languages (natural and computer) have a recursive,
hierarchical structure.
• This structure is implicit in ordinary textual representation.
• Recursive structure can be made explicit by representing
sentences in the language as trees: abstract syntax trees
(AST’s)
• AST’s are easier to optimize, generate code from, etc. than
textual representation.
• Converting textual representations to AST: job of parser

/\ Writing a parser for the language

/** Token Scan.getToken() is first token of a sentence for E.
Parse it, giving error mess. if there are mistakes. After the parse,
Scan.getToken should be the symbol following the parsed E. */
public static void parseE() {
parseT();
    while (Scan.getToken() is + or - ) {
                        Scan.scan();
parse(T);
    }
}
In-order and post-order walks

- **In-order walk**: infix
  - process left sub-tree
  - process root
  - process right sub-tree

- **Post-order walk**: postfix
  - process left sub-tree
  - process right sub-tree
  - process root

Tree search

- Analog of linear search in lists: given tree and an object, find out if object is stored in tree.
- Trivial to write recursively; much harder to write iteratively.

Recursion on trees

- Recursive methods can be written to operate on trees in the obvious way.
- In most problems
  - base case: empty tree
  - sometimes base case is leaf node
  - recursive case: solve problem on left and right subtrees, and then put solutions together to compute solution for tree

Walks of tree

A walk of a tree processes each node of the tree in some order.

Example on last slide showed pre-order walk of tree:
- process root
- process left sub-tree
- process right sub-tree

Intuition: think of prefix representation of expressions

Some useful routines

- **isLeaf(TreeNode t)**
  ```java
  public static boolean isLeaf(TreeNode t) {
    return (t == null) & & t.getLeft() == null & & t.getRight() == null;
  }
  ```

- **height(TreeNode t)**
  ```java
  public static int height(TreeNode t) {
    if (t == null) return -1; // height is undefined for empty tree
    if (isLeaf(t)) return 0;
    return 1 + Math.max(height(t.getLeft()), height(t.getRight()));
  }
  ```

- **nNodes(TreeNode t)**
  ```java
  public static int nNodes(TreeNode t) {
    if (t == null) return 0;
    return t + nNodes(t.getLeft()) + nNodes(t.getRight());
  }
  ```
• Generate textual representation from AST (Abstract Syntax Tree)

/** = textual representation of AST t, with each binary operation parenthesized */

public static String flatten(TreeNode t) {
    if (t == null) return "";
    if (isLeaf(t)) return t.getDatum();
    return "(" + flatten(t.getLeft()) + t.getDatum() + flatten(t.getRight()) + ")" ;
}

Note. The code above does not deal with unary operators. Fix it yourself so that it does. A unary prefix operator will have a right subtree but no left subtree (i.e. an empty left subtree).

Useful facts about binary trees

• Maximum number of nodes at depth $d = 2^d$
• If height of tree is $h$,
  – minimum number of nodes it can have = $h+1$
  – maximum number of nodes it can have is $= 2^h + 2^{h-1} + \ldots + 2^1 + 1$
• Full binary tree of height $h$:
  – all levels of tree up to depth $h$ are completely filled.

Tree with header element

• As with lists, some prefer to have an explicit class Tree, an instance of which contains a pointer to the root of the tree.
• With this design, methods that operate on trees can be made into instance methods in this class, and the root of the tree does not have to be passed in explicitly to method.
• Feel free to use whatever works for you.

Tree with parent pointers

• In some applications, it is useful to have trees in which nodes other than root have references to their parents.
• Tree analog of doubly-linked lists.

class TreeWithPPNode {
    private Object datum;
    private TreeWithPPNode left, right, parent;
    …..appropriate constructors and getter and setter methods…
}

Summary

• Tree is a recursive data structure built from class TreeNode.
  – special case: binary tree
• Binary tree cells have both a left and a right “successor”
  – called children rather than successors
  – similarly, parent rather than predecessor
  – generalization of parent and child to ancestors and descendents
• Trees are useful for exposing the recursive structure of natural language programs and computer programs.
• File system on your hard drive is a tree.
• Table of contents of a book is a tree.