Trees

Dr Tom Ridge

2018 Q1

Trees

Trees

- What are trees?
  - e.g. see wikipedia: Tree structure http://en.wikipedia.org/wiki/Tree_structure; Tree (data structure) http://en.wikipedia.org/wiki/Tree_(data_structure); Tree traversal http://en.wikipedia.org/wiki/Tree_traversal
- What are examples of trees?
  - some examples of trees: management structure, inheritance hierarchy, filesystem, abstract expressions (see following slides)

Trees, common words

- some common words: node, root node, parent, child, children, ancestors, descendants, subtree, branching factor, binary tree, decorated, inner/interior node (non-leaf node), leaf node (or just leaf)

Simple examples: arithmetic expressions

- 1+2
- 1+(2+3)
- 1+((2+3)+4)
- 1+2*3+4

Representing a tree

We assume either left and right are null, or both are not null; if both are null, we have a leaf, otherwise we have a (non-leaf) node

```
// bsh must be imported!
public class MyNode {

  public MyNode left = null;
  public Object obj = null;
  public MyNode right = null;

  public MyNode(MyNode l, Object o, MyNode r) {
    left = l; obj = o; right = r;
  }

  public String toString() {
    return "MyNode("+left+","+obj+","+right+")";
  }

}
```

Basic functions to build a tree

```
MyNode node(MyNode l, Object o, MyNode r) {
  return new MyNode(l,o,r);
}

MyNode leaf(Object o) {
  return new MyNode(null,o,null);
}
```

Basic functions to break down a tree

```
boolean isLeaf(MyNode n) {
  return (n.left == null) && (n.right == null);
}

boolean isNode(MyNode n) {
  return !(isLeaf(n));
}

// if isNode(n), can access n.left and n.right
```

Writing some simple expressions

```
expr12 = node(leaf(1),"+",leaf(2));

expr16 = node(leaf(1),"+",leaf(6));

// what about the other examples?
```

Writing some code to process trees - count leaves

Note the recursion in the following:

```
int count_leaves(MyNode n) {
  if(isLeaf(n)) return 1;
  return count_leaves(n.left) + count_leaves(n.right);
}
```

Forming a list of all the leaves - leaves

```
// get leaves of a tree as a list
  List leaves(MyNode n) {
    if(isLeaf(n)) return cons(n.obj,nil());
    List ls = leaves(n.left);
    List rs = leaves(n.right);
    return append(ls,rs);
  }
```

Evaluating arithmetic expressions

Suppose `t = node(leaf(1),"+",leaf(6))`. We want a function which takes the tree and "evaluates" it to give the result 7.

Evaluating arithmetic expressions - leaf

Leaf case:

```
int eval(MyNode n) {
  if(isLeaf(n)) return (int)(n.obj);
  ...
```

Evaluating arithmetic expressions - node

```
int eval(MyNode n) {
  if(isLeaf(n)) return (int)(n.obj);
  if(n.obj.equals("+"))
    return eval(n.left) + eval(n.right);
  else
    return -1; // never reached - we assume nodes are "+"
}
```

Evaluating arithmetic expressions - method invocations

```
eval(leaf(3));

eval(expr16);

expr = node(expr16,"+",expr12);

eval(expr);
```

Challenge - tree contains

Complete the following function which returns true if the tree `t` contains the object `o`, and false otherwise.
Note: you can assume any tree you are given is not null (this is a standard assumption in Java). But the left and right attributes may be null!
Note: you can also assume the object at a node is not null.

```
boolean tcontains(MyNode t, Object o) {
   ...
}
```

Solution - tree contains

```
boolean tcontains(MyNode t, Object o) {
  if (t.obj.equals(o)) return true;
  if(isLeaf(t)) return false;
  boolean in_left = tcontains(t.left,o);
  boolean in_right = tcontains(t.right,o);
  return (in_left || in_right);
}
```

Challenge - nodes at level

Complete the following function which returns the nodes at a given level in a tree. The root is at level 0, and the children of the root are at level 1, and so on.

```
List nodes_at_level(MyNode t, int l) {
  ...
}
```

Solution - nodes at level

```
List nodes_at_level(MyNode t, int l) {
  if(l==0) return cons(t.obj,nil());
  if(isLeaf(t)) return nil();
  List left_nodes = nodes_at_level(t.left,l-1);
  List right_nodes = nodes_at_level(t.right,l-1);
  return append(left_nodes,right_nodes);
}
```

Challenge - nodes less than

Complete the following function which takes a tree (with all nodes decorated by an integer) and returns those nodes less than a given value k.

```
List nodes_less_than(MyNode t, int k) {
  ...
}
```

Solution - nodes less than

```
List nodes_less_than(MyNode t, int k) {
  List this_node;
  if(((Integer)t.obj) < k)
    this_node=cons(t.obj,nil());
  else
    this_node=nil();
  if(isLeaf(t)) return this_node;
  List left_nodes = nodes_less_than(t.left,k);
  List right_nodes = nodes_less_than(t.right,k);
  return append(left_nodes,append(this_node,right_nodes));
}
```

Tree traversal

Tree traversal

- The next few slides introduce 3 ways to traverse a tree, visiting all nodes
- These are a favourite topic of exam question

Tree traversal

- Look at the example tree from http://en.wikipedia.org/wiki/Tree_traversal
- Some rules:
  - you visit the left subtree before the right subtree
- The thing that isn't clear: given a parent p, and a left subtree l and a right subtree r, l has to come before r, but where does p come?
- Options: plr or lpr or lr p
- These have names
  - plr - parent **before** l and r - preorder
  - lpr - parent **inbetween** l and r - inorder
  - lrp - parent **after** l and r - postorder

Tree traversal, examples

- Inorder, preorder and postorder on the example tree

Tree traversal, preorder, code

```
void preorder(MyNode n) {
  if(isLeaf(n)) { System.out.println(n.obj); return; }
  else {
    System.out.println(n.obj);
    preorder(n.left);
    preorder(n.right);
  }
}
```

Challenge - tree traversal, inorder, and postorder

Solution - inorder

```
void inorder(MyNode n) {
  if(isLeaf(n)) { System.out.println(n.obj); return; }
  else {
    inorder(n.left);
    System.out.println(n.obj); // note this has changed!
    inorder(n.right);
  }
}
```