Trees I

Definitions, Traversals,
Binary Trees

Announcement

- Final Exam
  - Wednesday, February 25, 2004
  - 8:00am – 10:00 am
  - 70-3435
- Please report all exam conflicts now!

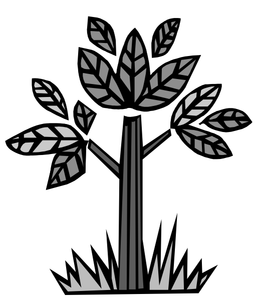
Project 2 Notes

- Writeup now on the Web
- Due Dates:
  - Minimum submission due Friday, Feb 6th
    - Entry.java & Document.java
    - Partial implementation of Directory.java provided
  - Final submission due Sunday, Feb 15th
    - A little more than a week after the minimum!
    - Complete Directory.java & VFSystem.java
    - Integration tests WILL be performed.

Questions

- On sorting, searching?
- Any other questions?

Trees



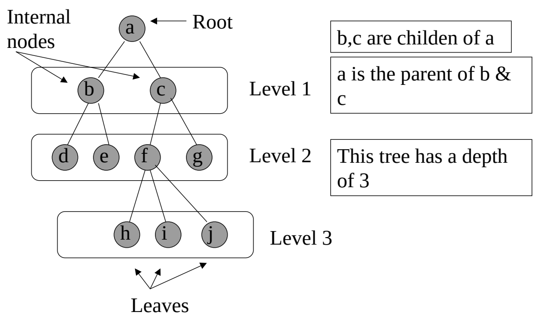
I think that I shall never see
A poem lovely as a tree

-- J. Kilmer

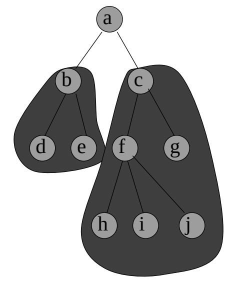
Trees

- In CS, we look at trees from the bottom up

Anatomy of a Tree

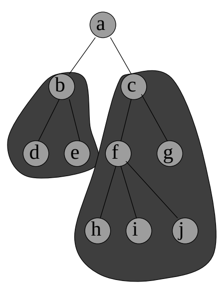


Anatomy of a Tree



- Subtree
  - All children of a node, can be considered the root of it's own tree
  - These are called subtrees
  - I smell recursion!

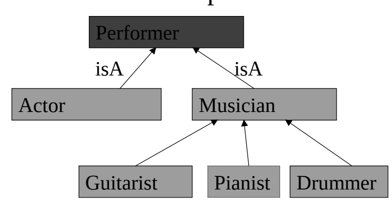
Anatomy of a Tree



- Balanced Tree
  - A tree is balanced if all subtrees at the same level have the same depths
  - This tree is not balanced since Node c's subtree has a depth of 2 but Node b's subtree has a depth of 1.

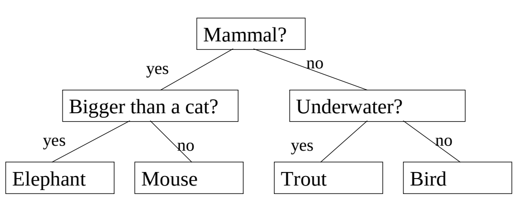
What are trees good for?

- Hierarchical relationships



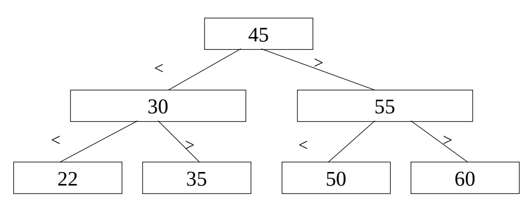
What are trees good for?

- Binary trees can be used to represent decision taxonomies



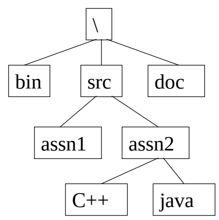
What are trees good for?

- Branching can also imply ordering of node data



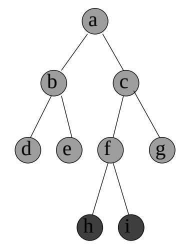
What are trees good for?

- Representing Hierarchical File Structures
  - Hmmm…



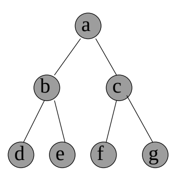
  - Questions?

Anatomy of a Binary Tree



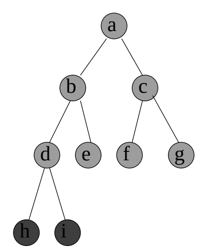
- Binary Tree
  - Each node has at most 2 children
  - Children of nodes in a binary tree are referred to as <u>left child</u> or <u>right child</u>
    - Node h is the left child of Node f
    - Node i is the right child of Node f

Anatomy of a Binary Tree



- Full Binary Tree
  - A binary tree is <u>full</u> if
    - All of it's leaf nodes are of the same depth
    - Each non-leaf node has 2 children

Anatomy of a Tree

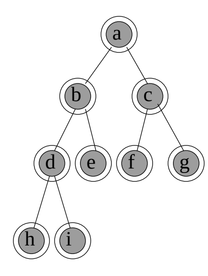


- Complete Binary Tree
  - A binary tree is <u>complete</u> if
    - Each level (except the deepest) must contain as many nodes as possible
    - At the deepest level, all nodes as as far left as possible

Traversing a Tree

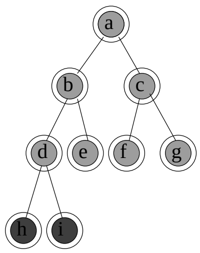
- A means to process all the nodes in a tree
  - A traversal starts at the root
  - Visits each node exactly once
    - "Processes" the data in a node when visited
  - Nodes can be visited in different orders
    - Breadth-first traversal
    - Depth-first traversal
      - Preorder
      - Inorder
      - Postorder

Anatomy of a Tree



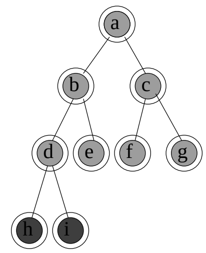
- Breadth-first
  - All the nodes at a given level are visited before the nodes at the next level
  - Example:
    - a,b,c,d,e,f,g,h,i

Anatomy of a Tree



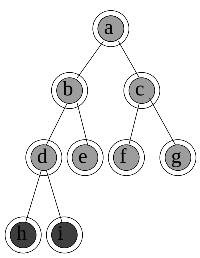
- Pre-order
  - At each node
    - The node is visited first
    - Pre-order traversal of the left subtree
    - Pre-order traversal of the right subtree
  - Example:
    - a,b,d,h,i,e,c,f,g

Anatomy of a Tree



- in-order
  - At each node
    - In-order traversal of the left subtree
    - The node is visited next
    - The In-order traversal of the right subtree
  - Example:
    - h,d,i,b,e,a,f,c,g

Anatomy of a Tree



- post-order
  - At each node
    - Post-order traversal of the left subtree
    - Post-order traversal of the right subtree
    - The node is visited next
  - Example:
    - h,i,d,e,b,f,g,c,a

Implementing a Binary Tree

- Define a binary tree node object
- Each node can be seen as the root of a Binary Tree.

A Binary Tree Node Class

- Class BTNode
  - Member variables
    - data – data stored within the node (Object)
    - leftChild – left subtree (BTNode)
    - rightChild – right subtree(BTNode)
    - parent – parent node (BTNode)
  - Methods
    - Constructors (for internal node, for leaf)
    - Get methods (getData, getLeft, getRight, getParent)
    - Set methods (setData, setLeft, setRight, setParent)
    - Traversal methods (inorder, preorder, postorder, visit)

BTNode

```
public class BTNode {
    protected Object data;
    protected BTNode leftChild;
    protected BTNode rightChild;
    protected BTNode parent;
```

What about a tree that isn't binary?

BTNode -- constructors

```
// Constructor for interior node
public BTNode (Object o, BTNode l, BTNode r)
{
     data = o;
     parent = null;
     setLeft (l);
     setRight(r);
}

// Constructor for a leaf
public BTNode (Object o)
{
     data = o;
     parent = null;
     setLeft (null);
     setRight (null);
}
```

BTNode – get Methods

```
// get the Data
public Object getData()
{
     return data;
}

// Get the left child
public BTNode getLeft ()
{
     return leftChild;
}

// Get the right child
public BTNode getRight ()
{
     return rightChild;
}

// Get the parent
public BTNode getParent ()
{
     return parent;
}
```

BTNode – set Methods

```
// set the Data
public void setData(Object o)
{
   data = o;
}

// Set the left child
public void setLeft (BTNode n)
{
   leftChild = n;
   if (n!= null)
      n.setParent (this);
}

// Set the right child
public void setRight (BTNode n)
{
   rightChild = n;
   if (n!= null)
      n.setParent (this);
}

// Set the parent
public void setParent (BTNode n)
{
    parent = n;
}
```

BTNode – traversal

- Visit
  - Default is to print
  - Assume will be overridden by subclasses

```
public void visit()
{
        System.out.println (data.toString());
}
```

BTNode – traversal

- Inorder
  - Process left child
  - Visit node
  - Process right child

```
public void inorder()
{
    leftChild.inorder();
    visit();
    rightChild.inorder();
}
```

BTNode – traversal

- But won’t this recursion go on forever?

BTNode – traversal

- Inorder
  - Process left child
  - Visit node
  - Process right child

Test

Stop = do nothing

```
public void inorder()
{
  if (leftChild != null) leftChild.inorder();
   visit();
   if (rightChild != null) rightChild.inorder();
}
```

Continue

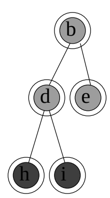
BTNode – traversal

- Pre-order, post-order

```
public void preorder()
     {
        visit();
        if (leftChild != null) leftChild.preorder();
        if (rightChild != null) rightChild.preorder();
     }


public void postorder()
     {
        if (leftChild != null) leftChild.postorder();
        if (rightChild != null) rightChild.postorder();
        visit();
     }
```

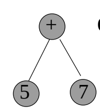
BTNode – let's build a tree

```
public static void main (String args[])
{
   // Level 2
   BTNode h = new BTNode ("h");
   BTNode i = new BTNode ("i");

   // Level 1
   BTNode d = new BTNode ("d", h, i);
   BTNode e = new BTNode ("e");

   // Root
   BTNode root = new BTNode ("b", d, e);

   // Do an inorder traversal
   root.inorder ();

}
```



What can we do with binary trees?

- Binary trees can be used to represent arithmetic expressions.
  - Interior nodes are operator (+, --, *, /)
  - Leaves are operands (numbers)
  - Example
    - 5 + 7



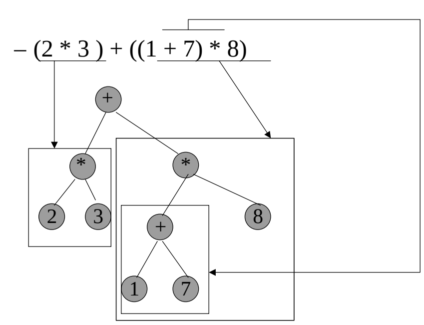
Operator

Operand

Expression trees

- The operand of one node, can itself be an expression
  - Children nodes can be roots of their own subtrees
- Example:
  - (2 * 3) + ((1 + 7) * 8)

Expression Trees



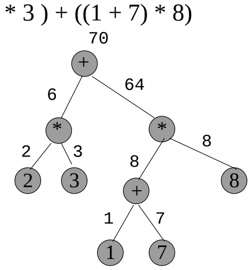
Expression trees

- Evaluating expression trees
  - Leaf nodes evaluate to the number that they represent
  - Interior nodes:
    1. Evaluate the left and right children
    2. Apply appropriate operation on results of Step 1

    What kind of traversal would this be?

Expression Trees

– (2 * 3) + ((1 + 7) * 8)



Expression trees

- Let’s implement this

```
public class ExpressionTreeNode extends BTNode
{
   public int eval();
}
```

Expression trees

```
public int eval()
{
    int left = 0;
    int right = 0;

    if (leftChild != null) left = leftChild.eval();
    if (rightChild != null) right = rightChild.eval();

    if (data.equals (“+”)) return left + right;
    else if (data.equals (“-”)) return left - right;
    else if (data.equals (“*”)) return left * right;
    else if (data.equals (“/”)) return left / right;
    else return Integer.parseInt ((String)data);
}
```

Expression trees

```
public int eval() throws NumberFormatException
{
    int left = 0;
    int right = 0;

    if (leftChild != null) left = leftChild.eval();
    if (rightChild != null) right = rightChild.eval();

    if (data.equals (“+”)) return left + right;
    else if (data.equals (“-”)) return left - right;
    else if (data.equals (“*”)) return left * right;
    else if (data.equals (“/”)) return left / right;
    else return Integer.parseInt ((String)data);
}
```

Summary

- Trees
- Binary Trees
  – Implementation
  – Example: Expression Trees

Binary Search Trees

- Branching can also imply ordering of node data