Announcements

- Homework 2 is up. Due Feb. 23

- Problem 2 (Weiss 3.7), trimToSize() creates a new array of same size as list, copies each element.
Review

- Introduction to Trees
- Definitions
- Tree Traversal Algorithms
- Binary Trees
Today’s Plan

- Finish up examples of binary tree applications
- Binary Search Trees
  - Basic operations: insert, findMin/Max, contains
  - Delete
- Average depth analysis
The number of nodes at depth $d$ is $2^d$.

Total in a tree of depth $d$ is
\[ \sum_{i=0}^{d} 2^i = 2^{d+1} - 1 \]
(series identity)

A perfect binary tree has $N = 2^{d+1} - 1$ nodes.

Solving for $d$ finds $d = \log(N + 1) - 1$. 
Expression Trees

Expression Trees are yet another way to store mathematical expressions

\( ((x + y) \times z)/300 \)

Note that the main mathematical operators have 2 operands each

Inorder traversal reads back infix notation

Postorder traversal reads postfix notation
Decision Trees

- It is often useful to design decision trees
- Left/right child represents yes/no answers to questions

![Decision Tree Diagram]

- Hungry?
  - Do nothing
  - Enough money?
    - Chicken and Rice
    - Subsconscious
Search (Tree) ADT

- ADT that allows insertion, removal, and searching by **key**
  - A **key** is a value that can be compared
  - In Java, we use the **Comparable** interface
  - Comparison must obey transitive property
- Notice that the Search ADT doesn’t use any index
Binary Search Tree Property:
Keys in left subtree are less than root.
Keys in right subtree are greater than root.

BST property holds for all subtrees of a BST
Inserting into a BST

- \textbf{insert}(x) is public method
- privately, use \textbf{insert}(x, \text{root})

- \textbf{insert}(x, \text{Node } t)
  - if (t == \text{null}) return new \text{Node}(x)
  - if (x > t.key), then t.right = \text{insert}(x, \text{t.right})
  - if (x < t.key), then t.left = \text{insert}(x, \text{t.left})
  - return t
Searching a BST

- `findMin(t)`
  - if (t.left == null) return t.key
  - else return `findMin(t.left)`

- `contains(x, t)`
  - if (t == null) return false
  - if (x == t.key) return true
  - if (x > t.key), then return `contains(x, t.right)`
  - if (x < t.key), then return `contains(x, t.left)`
Deleting from a BST

- Removing a leaf is easy, removing a node with one child is also easy.
- Nodes with no grandchildren are easy.
- Nodes with both children and grandchildren need more thought.
- Why can’t we replace the removed node with either of its children?
A Removal Strategy

- First, find node to be removed, \( t \)
- Replace with the smallest node from the right subtree
  - \( a = \text{findMin}(t.\text{right}); \)
    - \( t.\text{key} = a.\text{key}; \)
- Then delete original smallest node in right subtree
  - \( \text{remove}(a.\text{key}, t.\text{right}) \)
Average Case Analysis

- All operations run in $O(d)$ time, but what is $d$?
  - Worst case $d = N$
  - Best case $d = \log(N+1)-1$
  - Average case?
Consider the **internal path length**: the sum of the depths of all nodes in a tree

Let $D(N)$ be the internal path length for some tree $T$ with $N$ nodes*

Suppose $i$ nodes are in the left subtree of $T$

Then $D(N) = D(i) + D(N - i - 1) + N - 1$
Average Case Analysis

- \( D(N) = D(i) + D(N - i - 1) + N - 1 \)
- Assume all insertion sequences are equally likely
- Subtree sizes only depend on the 1\(^{st}\) key inserted
- All subtree sizes equally likely
- Average of \( D(i) \) (and \( D(N-i-1) \)) is \( \frac{1}{N} \sum_{j=0}^{N-1} D(j) \)
Average Case Analysis

Average case $D(N)$ then becomes

$$D(N) = \frac{2}{N} \left[ \sum_{j=0}^{N-1} D(j) \right] + N - 1$$

This is a recurrence, which can be solved to show that $D(N) = O(N \log N)$

(page 272-273 in Weiss)

Then the average depth over all $N$ nodes is $O(\log N)$
Looking Forward

• How do we implement Search Trees that explicitly avoid worst case \( O(N) \) operations?

• What is the cost of avoiding worst case?
Reading

- Weiss Section 4.4: AVL Trees