Announcements (1/23/09)

- HW #2 due now
- HW #3 out today, due at beginning of class next Friday.
- Project 2A due next Wed. night.
- Read Chapter 4

CSE 326: Data Structures
Binary Search Trees

Steve Seitz
Winter 2009

ADTs Seen So Far

- Stack
  - Push
  - Pop
- Queue
  - Enqueue
  - Dequeue
- Priority Queue
  - Insert
  - DeleteMin

Then there is decreaseKey...

The Dictionary ADT

- Data:
  - a set of (key, value) pairs

- Operations:
  - Insert (key, value)
  - Find (key)
  - Remove (key)

The Dictionary ADT is also called the “Map ADT”
A Modest Few Uses

- Sets
- Dictionaries
- Networks: Router tables
- Operating systems: Page tables
- Compilers: Symbol tables

Probably the most widely used ADT!

Implementations

<table>
<thead>
<tr>
<th></th>
<th>insert</th>
<th>find</th>
<th>delete</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unsorted Linked-list</td>
<td>$O(1)$</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>Unsorted array</td>
<td>$O(1)$</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>Sorted array</td>
<td>$O(\log n)$</td>
<td>$O(\log n)$</td>
<td>$O(n)$</td>
</tr>
</tbody>
</table>

Binary Trees

- Binary tree is
  - a root
  - left subtree (maybe empty)
  - right subtree (maybe empty)

- Representation:

<table>
<thead>
<tr>
<th>Data</th>
<th>left pointer</th>
<th>right pointer</th>
</tr>
</thead>
</table>

Binary Tree: Representation

```
  A
 /   \
B     C
   /   \
  D     E
 /   \
G     F
 /   \
H     I
   /   \
  J

  A
 /   \
B     C
   /   \
  D     E
 /   \
F
```
Tree Traversals

A *traversal* is an order for visiting all the nodes of a tree.

Three types:
- **Pre-order**: Root, left subtree, right subtree
- **In-order**: Left subtree, root, right subtree
- **Post-order**: Left subtree, right subtree, root

\[
(2 \times 4) + 5
\]

\[
\begin{array}{c}
+ \\
\times \quad 5 \\
2 \quad 4
\end{array}
\]

\[
((2 \times 4) + 5)
\]

(an expression tree)

Inorder Traversal

```c
void traverse(BNode t) {
    if (t != NULL) {
        traverse(t.left);
        process t.element;
        traverse(t.right);
    }
}
```

Binary Tree: Special Cases

- **Complete Tree**
- **Perfect Tree**
- **“List” Tree**

Binary Tree: Some Numbers...

Recall: height of a tree = longest path from root to leaf.

For binary tree of height $h$:
- max # of leaves: $2^h$
- max # of nodes: $2^{h+1} - 1$
- min # of leaves: 1
- min # of nodes: $h+1$
Binary Tree: Some Numbers…

Recall: depth of a node = path length from node to root.

Consider the space of all possible binary trees of N nodes.

• Sum up the depths of every node in that forest and divide by the number of nodes.
• This is the average depth over all nodes over all binary trees of size N. How big is it?

\[ \Theta(\sqrt{N}) \]

What would the average depth be for a well-balanced tree?

Binary Search Tree Data Structure

• Structural property
  – each node has \( \leq 2 \) children
  – result:
    • storage is small
    • operations are simple

• Order property
  – all keys in left subtree smaller than root’s key
  – all keys in right subtree larger than root’s key
  – result: easy to find any given key

Example and Counter-Example

Find in BST, Recursive

```java
Node Find(Object key, Node root) {
    if (root == NULL) {
        return NULL;
    
    if (key < root.key) {
        return Find(key, root.left);
    } else if (key > root.key) {
        return Find(key, root.right);
    } else {
        return root;
    }
}
```
Find in BST, Iterative

Node Find(Object key, Node root) {
    while (root != NULL &&
        root.key != key) {
        if (key < root.key)
            root = root.left;
        else
            root = root.right;
    }
    return root;
}

Bonus: FindMin/FindMax

- Find minimum
- Find maximum

Insert in BST

Integrate Insert(13) Insert(8) Insert(31)

Insertions happen only at the leaves — easy!

BuildTree for BST

- Suppose keys 1, 2, 3, 4, 5, 6, 7, 8, 9 are inserted into an initially empty BST.
  
  If inserted in given order, what is the tree? What big-O runtime for this kind of sorted input?
  
  If inserted in reverse order, what is the tree? What big-O runtime for this kind of sorted input?
BuildTree for BST

- Suppose keys 1, 2, 3, 4, 5, 6, 7, 8, 9 are inserted into an initially empty BST.

\[5, 3, 7, 2, 1, 8, 1, 0, 6\]

- If inserted median first, then left median, right median, etc., what is the tree? What is the big-O runtime for this kind of sorted input?

\[
\Theta(n \log n)
\]

Deletion in BST

Why might deletion be harder than insertion?

Deletion

- Removing an item disrupts the tree structure.
- Basic idea: find the node that is to be removed. Then "fix" the tree so that it is still a binary search tree.
- Three cases:
  - node has no children (leaf node)
  - node has one child
  - node has two children

Deletion – The Leaf Case

Delete(17)
Deletion – The One Child Case

Delete(15)

Deletion – The Two Child Case

Delete(5)

What can we replace 5 with?

Deletion – The Two Child Case

Idea: Replace the deleted node with a value guaranteed to be between the two child subtrees

Options:
- \textit{succ} from right subtree: \texttt{findMin(t.right)}
- \textit{pred} from left subtree: \texttt{findMax(t.left)}

Now delete the original node containing \textit{succ} or \textit{pred}
- Leaf or one child case – easy!

Finally…

7 replaces 5

Original node containing 7 gets deleted
Balanced BST

Observations

- BST: the shallower the better!
- For a BST with $n$ nodes
  - Average depth (averaged over all possible insertion orderings) is $O(\log n)$
  - Worst case maximum depth is $O(n)$
- Simple cases such as insert(1, 2, 3, ..., n) lead to the worst case scenario

Solution: Require a Balance Condition that

1. ensures depth is $O(\log n)$ — strong enough!
2. is easy to maintain — not too strong!