AVL Trees

- Binary Tree that stays balanced!
- We’ve worked with TreeMap which we assume stays balanced

Rotation

- Rotation is a procedure that keeps balanced search trees balanced.
- There are two types -- a left rotation and a right rotation -- and each is the other’s reverse.
- The rotation procedure preserves the binary search tree invariant, i.e. after the rotation, the left subtree is less than or equal to the root and the right subtree is more than or equal to the root.

Important Dates

- There IS class next week!
- Project 3
  - You may have 1 or 2 partners.
  - NO EXCEPTIONS!
  - If you haven’t seen your partner’s code – it doesn’t exist!
  - Due Tuesday 8/11/2009 – 10pm
- Final Review
  - Sunday 8/09/2009 – 1-4pm in 306 Soda
- Final
  - Thursday 8/13/2009 – 5-8pm in 10 Evans

"Maximally-" Versus "Almost-" Balanced

Rotation

- Rotation is a procedure that keeps balanced search trees balanced.
- There are two types -- a left rotation and a right rotation -- and each is the other’s reverse.
- The rotation procedure preserves the binary search tree invariant, i.e. after the rotation, the left subtree is less than or equal to the root and the right subtree is more than or equal to the root.
Rotate Right
// this is D
Node temp = this.myLeft;  // temp = B
this.myLeft = temp.myRight; // D.left = C
temp.myRight = this;  // B.right = D

Rotate Left
// this is B
Node temp = this.myRight;  // temp = D
this.myRight = temp.myLeft; // B.right = C
temp.myLeft = this;  // D.left = B

Insertion / Rotation
• Assume A, C and E are each a subtree with height h
  – The leaves in C and E are one level lower than A’s leaves
• Add node F
• Now we have a height difference of 2
• Rotate LEFT!

Rotations
• Assume A, C, and E are each a subtree with height h
• Assume H is a subtree with height h+1
  – The leaves in C and E are one level lower than A’s and H’s leaves
• Add node F
• Now we have a height difference of 2
• Rotate Left THEN Right!
Stable Sort

- Sometimes there are equivalent keys in a data structure that is sorted.
- It usually doesn’t matter if you change the order of those equivalent keys.
- We call a sort “stable” if it doesn’t change the order of equivalent keys.

Easy Sorting Problems...

- Sort $n$ numbers in the range 0 to 9.
- What is the running time???
  - Problems like this can be solved in $O(n+\text{range})$.

Harder Sorting Problems...

Approach 1 (Bucket)

- Sort $n$ numbers in the range 0 to 99.
- Make some buckets and calculate what bucket they need to go into.
- Then sort each bucket.

Approach 2 (Radix)

- Sort $n$ numbers in the range 0 to 99.
- First sort by the one’s digit (this is counter intuitive).

Bucket Sort Pictures from Wikipedia

Some things are in the right order.
Radix Sort

- In our example I used a decimal digit
  - We could use binary, hexadecimal or r-binary digits
- The running time for our example: \( (n + b)*2 \)
  - We did two passes, each took \( n \) time.
  - We gathered up the buckets twice each took \( b \) time (where \( b \) is the number of buckets)
- What would be the running time if we had \( k \) possible decimal digits?
  \[ O\left((n + b)*k\right) \]

We can represent numbers with 1’s and 0’s

### Decimal

\[
\begin{align*}
3 & \quad 2 \quad 1 \quad 8 \\
10^3 & \quad 10^2 & \quad 10^1 & \quad 10^0 \\
3 \times 1000 & + 2 \times 100 & + 1 \times 10 & + 8 \times 1 \\
1 & \quad 0 & \quad 0 & \quad 1 \\
2^3 & \quad 2^2 & \quad 2^1 & \quad 2^0 \\
1 \times 8 & + 0 \times 4 & + 0 \times 2 & + 1 \times 1
\end{align*}
\]

Binary Numbers

<table>
<thead>
<tr>
<th>Decimal</th>
<th>Binary</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>0001</td>
<td>1</td>
<td>Typically we use 32 bit numbers</td>
</tr>
<tr>
<td>0010</td>
<td>2</td>
<td>32 digits where each digit is a 1 or a 0</td>
</tr>
<tr>
<td>0011</td>
<td>3</td>
<td>32 digits where each digit is a 1 or a 0</td>
</tr>
<tr>
<td>0100</td>
<td>4</td>
<td>32 digits where each digit is a 1 or a 0</td>
</tr>
<tr>
<td>0101</td>
<td>5</td>
<td>32 digits where each digit is a 1 or a 0</td>
</tr>
<tr>
<td>0110</td>
<td>6</td>
<td>32 digits where each digit is a 1 or a 0</td>
</tr>
<tr>
<td>0111</td>
<td>7</td>
<td>32 digits where each digit is a 1 or a 0</td>
</tr>
<tr>
<td>1000</td>
<td>8</td>
<td>32 digits where each digit is a 1 or a 0</td>
</tr>
</tbody>
</table>

Radix Sort with 32 bit numbers

- We can choose how many bits we want to sort at a time.
- 4 bits at a time (8 passes)
  - 0000 0000 0000 0000 0000 0000 0000 0000
- 8 bits at a time (4 passes)
  - 00000000 00000000 00000000 00000000
- Running Time of Radix Sort in terms of
  - \( n \): the number of items to sort
  - \( q \): the number of buckets
  - \( p \): the number of passes
  \[ O((n+q)*p) \]

Next Tuesday

- 2-3 Trees
- Btrees
- Tries