Sorting IV: AVL Trees and AVL Sort

Reading: CLRS 13.1-2, Chpt 14

Admin:
- Final Exam, Monday 16 May 2016, 1:30-4:30 PM, WS35
- Cookie Challenge, PS 2 Part B submitted 48 hrs early

---

Last Time: Binary Search Trees

**BST Property**

For any node \( x \):
- all nodes \( y \) in left subtree: \( \text{key}[y] \leq \text{key}[x] \)
- all nodes \( z \) in right subtree: \( \text{key}[z] \geq \text{key}[x] \)

Operations
- insert/delete, find-min/find-max, successor/predecessor, find all \( \Theta(h) \) for any particular case
- and \( \Theta(h) \) worst case

\[ \text{Inorder-Tree-Walk} \quad \Theta(n) \]

**BST Sort**

Create-BST-from-input
Inorder-Tree-Walk(root [Tree])

<table>
<thead>
<tr>
<th>last time</th>
<th>today</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \Theta(n^2) )</td>
<td>( \Theta(n \log n) )</td>
</tr>
<tr>
<td>( \Theta(n) )</td>
<td>( \Theta(n) )</td>
</tr>
<tr>
<td>( \Theta(n^2) )</td>
<td>( \Theta(n \log n) )</td>
</tr>
</tbody>
</table>
The "problem" with BSTs is the relationship between $h$ (height) and $n$ (# of nodes).

Worst case $h \approx n$

Improvement:
Somehow balance trees so $h = \Theta(\log n)$

Need to do this in a way that balance can be maintained in $O(\log n)$ time. → Then operations become $\Theta(\log n)$

Many possibilities — we will discuss AVL Trees as BSTs with extra condition

AVL Trees (Adelson-Velskii and Landis, 1962)

Invariant: For every node $x$, the heights of its left child and right child differ by at most 1.

$|h_L - h_R| \leq 1$
How to keep track of heights?

Augment the data structure for every node with its height

- Null has height -1
- Leaves have height 0
- Node has height equal to # of edges to "deepest descendant"

Prove: AVL trees have height $\Omega(\log n)$

"Most compact" tree: Complete binary tree of height $h$ has $n = 2^{h+1} - 1$, so $n \leq 2^{h+1} - 1$ and $h \geq \log_2(n+1) - 1$, so $h = \Omega(\log n)$

"Least compact" tree: $N_h = \text{minimum # nodes in AVL tree of height } h$

\[ N_h \geq 1 + N_{h-1} + N_{h-2} > 2N_{h-2} \]

$N_h > 2^{h/2} \rightarrow h < 2 \log_2(N_h)$

$\therefore h = \Theta(\log n)$
But how can the invariant be maintained? (in log n time)

First, note that changes to tree structure (insert/delete) only change heights of nodes on direct route between site of change and root.

Rotations Correct imbalanced trees

Right-Rotate (Y)

Left-Rotate (X)

Right-Rotate

- X rises and Y drops (reversing their parent-child relationship)
- X would have 3 children \( \Rightarrow \) \( \text{left}[Y] \leftarrow \text{Root}[T_2] \)
- Y would have 2 parents \( \Rightarrow \) \( p[x] \leftarrow \text{Previous p}[y] \)

Proper Key Order Maintained

\[ \text{keys}[T_1] \leq \text{key}[x] \leq \text{keys}[T_2] \leq \text{key}[y] \leq \text{keys}[T_3] \]
Re-balancing

- Let $x$ be lowest violating node - fix subtree and move up
- Assume $x$ is "right heavy" ($x$'s right child is deeper than left)
- Exist 3 cases
  1) $x$'s right child $y$ is right-heavy
  2) $y$ is balanced
  3) $y$ is left-heavy

Case 1

Case 2

same solution as case 1
Case 3: $y$ is left-heavy

![Tree Diagram]

- **Left-Rotate** ($X$)
  - The tree remains unbalanced.

- **Right-Rotate** ($y$)
  - The tree remains unbalanced.

- Further operations:
  - One of each
  - **Left-Rotate** ($X$)
Operations can be handled $\mathcal{O} (\log n)$ on AVL trees because balanced BSTs can be maintained $\mathcal{O} (\log n)$.

Insertion and Deletion can be carried out in the ordinary BST way, and imbalances created can then be corrected working up the tree toward the root.