6.006- Introduction to Algorithms

Lecture 6

Prof. Nir Shavit
Lecture Overview

• Review: Binary Search Trees
• Importance of being balanced
• Balanced BSTs
  – AVL trees
    • definition
    • rotations, insert
  – Other balanced trees
Binary Search Trees (BSTs)

- Each node $x$ has:
  - $\text{key}[x]$
  - Pointers: left[$x$], right[$x$], p[$x$]

- Property: for any node $x$:
  - For all nodes $y$ in the left subtree of $x$: $\text{key}[y] \leq \text{key}[x]$
  - For all nodes $y$ in the right subtree of $x$: $\text{key}[y] \geq \text{key}[x]$

- Height: $3$
The importance of being balanced

for $n$ nodes:

$\text{Perfectly Balanced}$

$h = \Theta(\log n)$

$\text{Path}$

$h = \Theta(n)$
Balanced BST Strategy

- Augment every node with some property
- Define a local invariant on property
- Show (prove) that invariant guarantees $\Theta(\log n)$ height
- Design algorithms to maintain property and the invariant
AVL Trees: Definition
[Adelson-Velskii and Landis’62]

• **Property**: for every node, store its height ("augmentation")
  - Leaves have height 0
  - NIL has "height" -1

• **Invariant**: for every node x, the heights of its left child and right child differ by at most 1
AVL trees have height $\Theta(\log n)$

- Let $n_h$ be the minimum number of nodes of an AVL tree of height $h$
- We have $n_h \geq 1 + n_{h-1} + n_{h-2}$
  
  \[ \Rightarrow n_h > 2n_{h-2} \]
  
  \[ \Rightarrow n_h > 2^{h/2} \]
  
  \[ \Rightarrow h < 2 \log n_h \]

Better bounds:

$n_h \geq n_{h-1} + n_{h-2} = \text{Fib}_{h-2}$

Fibonacci series approximation gives $1.4 \log n$
Rotations maintain the inorder ordering of keys:

\[ a \in \alpha, \ b \in \beta, \ c \in \gamma \implies a \leq A \leq b \leq B \leq c. \]
Insertions/Deletions

- Insert new node $u$ as in the simple BST
  - Can create imbalance
- Work your way up the tree, restoring the balance
- Similar issue/solution when deleting a node
Balancing

- Let $x$ be the lowest “violating” node
- Assume the right child of $x$ is deeper than the left child of $x$ ($x$ is “right-heavy”)
- Scenarios:
  - Case 1: Right child $y$ of $x$ is right-heavy
  - Case 2: Right child $y$ of $x$ is balanced
  - Case 3: Right child $y$ of $x$ is left-heavy

In class we saw the symmetric left heavy case because it fits better with the examples
Case 1: $y$ is right-heavy

LEFT-ROTATE$(x)$
Case 2: \( y \) is balanced

\[
\text{LEFT-ROTATE}(x)
\]

Same as Case 1
Case 3: \( y \) is left-heavy

Need to do more …
Case 3: \( y \) is left-heavy

\[ \text{RIGHT-ROTATE}(y) \]
\[ \text{LEFT-ROTATE}(x) \]

And we are done!
Examples of insert/balancing

Insert(23)

$x = 29$: left-left case

Done

Insert(55)

$x = 65$: left-right case

Done
AVL Sort

- Insert $n$ items in AVL tree $\implies O(n \log n)$ time
- Traverse tree and output items $\implies O(n)$
- Total time for AVL Sort is $O(n \log n)$