lecture17: AVL Trees

Sean Massung

Largely based on slides by Cinda Heeren
CS 225 UIUC

9th July, 2013
Announcements

- mt2 tonight!
- mp5.1 extra credit due Friday (7/12)
An interesting tree

- Can you make a BST that looks like a zig zag?
- What is the insertion order?
BST algorithms depend on the height of the tree; the analysis should be in terms of the amount of data \(n\) in the tree. Therefore, we need a relationship between the height, \(h\), and \(n\). Reminder: \( \text{height}(T) \) is:

- \(-1\) if \(T = \{\}\),
- \(1 + \max(\text{height}(T_{\text{left}}), \text{height}(T_{\text{right}}))\) otherwise
Question 1

What is the maximum number of nodes in a tree of height $h$?

$$M(h) = \sum_{i=0}^{h} 2^i = 2^{h+1} - 1$$

We will prove this is correct with the following recurrence: $M(h) = 1 + 2M(h-1), M(0) = 1$. Why?

- **Base:** $M(0) = 2^{0+1} - 1 = 1 \ \checkmark$

- **Inductive step:** assume $M(h-1) = 2^h - 1$. Then
  
  $$M(h) = 1 + 2(2^h - 1) = 1 + 2^{h+1} - 2 = 2^{h+1} - 1 \ \checkmark$$
What is the least possible height of a tree of $n$ nodes?

We know from the previous slide $n \leq 2^{h+1} - 1$. That means:

- $n < 2^{h+1}$
- $\log n < h + 1$
- $\log n - 1 < h$
- So a $h$ is $O(\log n)$
Question III and IV

What is the minimum number of nodes in a tree of height $h$?

$n \geq h + 1$

What is the greatest possible height of a tree with $n$ nodes?

$h \leq n - 1$

Fun facts:

- The height of a BST depends on the order the data is inserted
- Average height is $O(\log n)$, but worst case is $O(n)$
The height balance of a tree is defined to be:
\[ b = \text{height}(T_{\text{left}}) - \text{height}(T_{\text{right}}) \]

So, a tree is height balanced if:
- \( T = \{\} \), or
- \( T = \{r, T_{\text{left}}, T_{\text{right}}\} \), \( b \leq 1 \) and \( T_{\text{left}}, T_{\text{right}} \) are height balanced.
**Balance check**

- \( b = \text{height}(T_{left}) - \text{height}(T_{right}) \)
- A tree is *height balanced* if:
  - \( T = \{\} \), or
  - \( T = \{r, T_{left}, T_{right}\}, b \leq 1 \) and \( T_{left, right} \) are height balanced
- Is the tree to the right balanced?
Can you describe a linear algorithm to calculate the height at each node?

Is there an easy way to know a node’s height after it is inserted?
Rotating trees: how does the balance change?

“stick”

If you had a TreeNode pointer to the node containing 2, what code makes this rotation?
Fixing balance with rotations

- We just witnessed a *left rotation* about the node containing 2
- There is an analogous *right rotation*
- Left and right rotations turn sticks into mountains
- Using the correct rotations can decrease the height of the tree!
- Note that there could have been other subtrees beneath these nodes
Left and right rotations

**Left Case**

- Node 5
- Node 4
- Node 3
- A, B, C, D

**Right Case**

- Node 3
- Node 4
- Node 5
- A, B, C, D

**Balanced**

- Node 4
- Node 3
- Node 5
- A, B, C, D
Double rotations

- How can we fix this weird-looking tree?
- Can you make it balanced with a single rotation?
- How about two rotations?
Double rotations

80-85-90: “boomerang”

80-85-90: “stick”

80-85-90: “mountain”
Left-right and Right-left rotations

Left Right Case

Right Left Case

Balanced
An efficient Dictionary implementation

- Goal: use rotations to keep a BST balanced
- How does this affect the running time?
- What is the running time of all the rotations?
- How do we know \textit{when} to rotate? Or, how do we know when we are imbalanced?
AVL Trees are the answer!

- AVL trees have a guaranteed $O(\log n)$ height
  - We know from earlier this is the least possible height of a tree with $n$ nodes
- They maintain this height with rotations after inserting and deleting
- Heights are stored in each node, making for efficient calculations
  - These stored heights need to be updated after inserting and deleting!

In fact, the AVL tree was the first self-balancing binary search tree to be invented. It is described in the 1962 paper “An algorithm for the organization of information” by Adelson-Velskii and Landis.