COMP 3170 - Analysis of Algorithms & Data Structures

Shahin Kamali

Lecture 9 - Jan. 22, 2018

CLRS 12.2, 12.3, 13.2, read problem 13-3

University of Manitoba
Binary Search Trees (review)

Structure  A BST is either empty or contains a KVP, left child BST, and right child BST.

Ordering  Every key $k$ in $T.left$ is less than the root key.
Every key $k$ in $T.right$ is greater than the root key.
BST Search and Insert

\( \text{search}(k) \) Compare \( k \) to current node, stop if found, else recurse on subtree unless it’s empty

\( \text{insert}(k, v) \) Search for \( k \), then insert \((k, v)\) as new node

Example:
BST Delete

- If node is a leaf, just delete it.
- If node has one child, move child up
- Else, swap with **successor** or **predecessor** node and then delete
  - predecessor is the rightmost node on the left subtree
  - successor is the leftmost node on the right subtree
Binary Search Trees

- How to find max/min elements in a BST?
  - Just find the rightmost/leftmost node in $\Theta(h)$ time

- How can I print all keys in sorted order?
  - Do an in-order traversal of the tree in $\Theta(n)$ time
  - Can we do that in $o(n)$? No! We need to report an output of size $n$

- **BSTs maintain data in sorted order, which is useful for some queries (an advantage over hash tables which scatter data).**
Height of a BST

*search, insert, delete* all have cost \( \Theta(h) \), where 
\( h = \text{height of the tree} = \text{max. path length from root to leaf} \)

If \( n \) items are *inserted* one-at-a-time, how big is \( h \)?

- Worst-case: \( \Theta(n) \)
- Best-case: \( \Theta(\log n) \)
- Average-case: \( \Theta(\log n) \)
  (similar analysis to quick-sort1)
Balanced BSTs

- Perfectly balanced BSTs: all nodes except for the bottom 2 levels are full.
  - Too strict for efficient BST balancing.
- Weight balanced: at each internal node $i$, at least $cn_i$ nodes are in its left subtree and $cn_i$ in its right subtree, for some constant $c \in (0, 1/2]$, where $n_i$ denotes the number of descendants for node $i$.
- Height balanced: heights of left and right subtrees of each internal node differ by at most $k$, for some constant $k \geq 1$.
  - For AVL trees, $k = 1$.
  - We will assume $k = 1$ for the remainder of our discussion.
- Height $\Theta(\log n)$ where $n$ is the number of nodes in the tree.
- All balanced BSTs (with respect to any of above definitions) have height $\Theta(\log n)$
  - We see the proof for height-balanced BSTs in a minute.
Tree height

**Definition**

The **height** of a node $a$ is the length of the longest path between $a$ and any descendent of $a$

- as opposed to **depth** which is the length of the path between $a$ and the root.
- Height can be defined recursively as follows:

$$
\text{height}(a) = \begin{cases} 
-1, & a = \Phi \\
1 + \max\{\text{height}(a.\text{left}), \text{height}(a.\text{right})\} & a \neq \Phi
\end{cases}
$$

- For a height-balanced BST with $k = 1$, the balancing factor for any node is in $\{-1, 0, 1\}$. 

Bounds for the height of height-balanced BSTs

Theorem

For the height $h(n)$ of a height-balanced BST (with $k = 1$) on sufficiently large $n$ nodes we have $\log(n) - 1 < h(n) < 1.45 \log(n + 1)$

- This implies $h(n) \in \Theta(\log n)$.
- Let’s see the proof.
Lower Bound for the height of height-balanced BSTs

- We want to prove $\log(n) - 1 < h(n)$.
- The number of nodes in a binary search tree of height $h$ is at most:

$$n \leq 2^{h+1} - 1 \Rightarrow \log n \leq \log(2^{h+1} - 1) < \log(2^{h+1}) = h + 1$$

Hence, we have $\log n - 1 < h$. 
Upper Bound for the height of height-balanced BSTs

- We want to show $h(n) < 1.45 \log(n + 1)$.
  - Let $s(n)$ denote the minimum number of nodes in a height-balanced BST (with $k = 1$)
  - We have $s(0) = 1 \quad s(1) = 2 \quad s(2) = 4$

$$s(h) = \begin{cases} 
1 & h = 0 \\
2 & h = 1 \\
s(h - 1) + s(h - 2) + 1, & h \geq 2
\end{cases}$$

- We can say $s(h) > F(h)$ where $F(h)$ is the $h$’th Fibonacci number.
  - For large $n$, we have $F(h) \approx \frac{1}{\sqrt{5}} \left( \frac{1+\sqrt{5}}{2} \right)^{h+1} - 1$

We have $n > \frac{1}{\sqrt{5}} \left( \frac{1+\sqrt{5}}{2} \right)^{h+1} - 1 \rightarrow \sqrt{5}(n + 1) \geq \left( \frac{1+\sqrt{5}}{2} \right)^{h+1} \rightarrow 
\log(\sqrt{5}(n + 1)) \geq (h + 1) \log\left( \frac{1+\sqrt{5}}{2} \right) \rightarrow h < \frac{\log\sqrt{5} + \log(n+1)}{\log(1+\sqrt{5}) - 1} - 1
\rightarrow \frac{1}{\log(1+\sqrt{5}) - 1} \log(n + 1) + \frac{\log\sqrt{5}}{\log(1+\sqrt{5}) - 1} - 1 < 1.45 \log(n + 1)$
BST Single Rotation

- Height of a height-balanced BST on $n$ nodes is $\Theta(\log n)$

- A **self-balancing BST** maintains the height-balanced property after an insertion/deletion via **tree rotation**

![BST Rotation Diagram]

- Every rotation swaps parent-child relationship between two nodes (here between 2 and 4)
- Tree rotation preserves the BST key ordering property.
- Each rotation requires updating a few pointers in $O(1)$ time.
- original height: $\max(height(a) + 2; height(b) + 2; height(c) + 1)$
- new height: $\max(height(a) + 1; height(b) + 2; height(c) + 2)$