CS 218 : Design and Analysis of Algorithms

Lecture 2: Binary Search Trees

Lecturer: *Sundar Vishwanathan*
Computer Science & Engineering Indian Institute of Technology, Bombay

1 Balanced Binary Search Trees

A binary search tree is a binary tree with values at the nodes arranged such that the values at the nodes in the right subtree of a node is at least the value of the node and the values at nodes in the left subtree of the node is at most the value of the node.

It supports three operations:`(Insert, Delete and Find)`. A balanced binary search tree performs all three operations in $O(\log n)$ time where n is the total number of nodes in the tree. Recall that the *height* is the maximum number of edges on a root-leaf path. If you have analysed the operations from the last lecture you know that the time is proportional to the height of the tree.

In order to implement these operations `(Insert, Delete and Find)` in $O(\log n)$ time, the height of the tree should be at most $O(\log n)$. If we insert the elements in increasing order, we get a path, and the height of the tree becomes $\Omega(n)$. Hence during insert/delete, we should rearrange the tree and maintain balance. However we cannot afford to spend too much time doing this reordering. In order to keep track of balance it is natural to store the heights of the subtrees rooted at each node. While perfect balance will not be possible, to keep the tree balanced we will insist that

$$| \text{ height of left subtree} - \text{height of right subtree } | \leq 1 (\star) \tag{1}$$

This property is local to the node and hence easy to check. Before we see how to perform operations while maintaining this local condition, we see how this local condition ensures that the tree remains almost balanced, that is the height of the resultant tree is $O(\log n)$. It was important that this local check is enough to keep the height of the tree (a global quantity) $O(\log n)$.

Suppose a binary tree satisfies the local property and is of height h. What is the minimum number of nodes it should possess? Note that this is what we are looking for. Why?

Let $f(h)$ be the minimum number of nodes in a tree of height h which satisfies $(\star)$. Then it is easy to see that

$$\begin{aligned} f(h) &\geq f(h-1) + f(h-2) + 1 \\ f(1) &= 1 \\ f(2) &= 2 \end{aligned}$$

We give a simple proof of an $\Omega(c^m)$ bound for some constant c. Assuming that f is monotone increasing with h, $f(h) \geq 2 \cdot f(h-2)$.
Unravelling this k times we get $f(h) \geq 2^k \cdot f(h-2k)$. When $h - 2k = 2$ we have $k = \frac{h}{2}$. Therefore, by induction $f(h) \geq 2^{\frac{h}{2}}$. The base case is $f(2) = 2$ which is satisfied. Therefore, the longest path (root to leaf) in this tree is at most $2 \cdot \log n$.
Exercise. Find the best c you can which satisfies the above inequalities.

1.1 Operations

1.1.1 FIND

Find is done as described earlier. If the height is $2 \cdot \log n$, the time taken is $O(\log n)$.

1.1.2 INSERT

Note that inserts happen only at a new leaf. Also an insertion may create an imbalance only at the vertices in the path from the new leaf to the root and nowhere else.

The local operation which are used to maintain balance after insertion and deletion is rotation. This operation moves the nodes around a node while keeping the tree a binary search tree. It changes the height of a constant number of nodes close to x. The key is to use these rotations judiciously to keep the balance. Review this part. Also review how delete is performed. We will not need the exact way this is done, but will need the final data structure. The key take home funda is that a local property is maintained at each node that gives rise to a global inequality (on the height of the tree.) This is the key design principle used in many data structures.