CSE 326 Lecture 7: More on Search Trees

✦ Today’s Topics:
  ➤ Lazy Operations
  ➤ Run Time Analysis of Binary Search Tree Operations
  ➤ Balanced Search Trees
    ◆ AVL Trees and Rotations

✦ Covered in Chapter 4 of the text

From Last Time: Remove (Delete) Operation

✦ Removing a node containing X:
  1. Find the node containing X
  2. Replace it with:
     If it has no children, with NULL
     If it has 1 child, with that child
     If it has 2 children, with the node with
        the smallest value in its right subtree,
        (or largest value in left subtree)
  3. Recursively remove node used in 2 and 3

✦ Worst case: Recursion propagates all the way to a leaf node – time is
  \( O(\text{depth of tree}) \)
A “lazy” operation is one that puts off work as much as possible in the hope that a future operation will make the current operation unnecessary.

Lazy Deletion

- Idea: Mark node as deleted; no need to reorganize tree
- Skip marked nodes during Find or Insert
- Reorganize tree only when number of marked nodes exceeds a percentage of real nodes (e.g., 50%)
- Constant time penalty only due to marked nodes – depth increases only by a constant amount if 50% are marked undeleted nodes (N nodes max N/2 marked)

- Modify Insert to make use of marked nodes whenever possible e.g., when deleted value is re-inserted
- Can also use lazy deletion for Lists
Run Time Analysis of BST operations

✦ All BST operations (except MakeEmpty) are $O(d)$, where $d$ is the depth of the accessed node in the tree
  ➤ MakeEmpty takes $O(N)$ for a tree with $N$ nodes – frees all nodes

✦ We know: $\log N \leq d \leq N-1$ for a binary tree with $N$ nodes
  ➤ What is the best case tree? What is the worst case tree?

✦ Best Case Running Time of Insert/Remove/etc. = ?
✦ Worst Case Running Time = ?
✦ Average Case Running Time = ?

The best, the worst, and the average…

✦ For a binary tree with $N$ nodes, depth $d$ of any node satisfies: $\log N \leq d \leq N-1$
✦ So, best case running time of BST operations is $O(\log N)$
✦ Worst case running time is $O(N)$
✦ Average case running time = $O(\text{average value of } d) = O(\log N)$
  ➤ Can prove that average depth over all nodes = $O(\log N)$ if all insertion sequences equally likely.
  ➤ See Chap. 4 in textbook for proof
Can we do better?

- Worst case running time of BST operations is $O(N)$
- E.g. What happens when you Insert elements in ascending (or descending) order?
  - Insert 2, 4, 6, 8, 10, 12 into an empty BST
- **Problem:** Lack of “balance” – Tree becomes highly asymmetric
- **Idea:** Can we restore balance by re-arranging tree according to depths of left and right subtrees?
  - **Goal:** Get depth down from $O(N)$ to $O(\log N)$

Idea #1: Achieving the perfect balance…

- First try at balancing trees: Perfect balance
  - Re-arrange to get a complete tree after every operation
- **Recall:** A tree is complete if there are no “holes” when scanning from top to bottom, left to right
- **Problem:** Too expensive to re-arrange
  - E.g. Insert 2 in the example shown
- **Need a looser constraint…**
Idea #2: Leave it to the professionals…

- Many efficient algorithms exist for balancing trees in order to achieve faster running times for the BST operations
  - Adelson-Velskii and Landis (AVL) trees (1962)
  - Splay trees and other self-adjusting trees (1978)
  - B-trees and other multiway search trees (1972)

AVL Trees

- AVL trees are height-balanced binary search trees

- **Balance factor** of a node = height(left subtree) - height(right subtree)

- An AVL tree can only have balance factors of 1, 0, or -1 at every node
  - For every node, heights of left and right subtree differ by no more than 1
  - Height of an empty subtree = -1

- **Implementation**: Store current heights in each node
Which of these are AVL trees?

AVL Trees: Examples and Non-Examples

Balance factors

Not AVL

AVL
The good news about AVL Trees

✦ Can prove: Height of an AVL tree of $N$ nodes is always $O(\log N)$

✦ How? Can show:

- Height $h \leq 1.44 \log(N+2)-0.328$
- Prove using recurrence relation for minimum number of nodes $S(h)$ in an AVL tree of height $h$:
  
  $S(h) = S(h-1) + S(h-2) + 1$

- Use Fibonacci numbers to get bound on $S(h)$ bound on height $h$
- See textbook for details

The really good news about AVL Trees

✦ Can prove: Height of an AVL tree of $N$ nodes is always $O(\log N)$

✦ All operations (e.g. Find, Remove using lazy deletion, etc.) on an AVL tree are $O(\log N)$…

✦ …except Insert

- Why is Insert different?

R. Rao, CSE 326
The bad news about AVL Trees

![Diagram showing insertion of 3]

No longer an AVL tree (i.e. not balanced anymore)

Restoring Balance in (the life of) an AVL Tree

- **Problem**: Insert may cause balance factor to become 2 or –2 for some node on the path from insertion point to root node
- **Idea**: After Inserting the new node,  
  1. Back up to root **updating heights** along the access path  
  2. If Balance Factor = 2 or –2, adjust tree by **rotation** around **deepest** such node.
Rotating to restore Balance: A Simple Example

![AVL and Not AVL diagrams]

Next Class:
Rotating and Splaying for Fun and Profit

To Do:
Finish Reading Chapter 4