CSE 373 Lecture 7: More on Search Trees

- Today’s Topics:
  - Array Implementation of Trees
  - Lazy Deletion
  - Run Time Analysis of Binary Search Tree Operations
  - AVL Trees
  - Splay Trees
- Covered in Chapter 4 of the text

Array Implementation of Trees

- Used mostly for complete binary trees
  - A complete tree has no gaps when you scan the nodes left-to-right, top-to-bottom
- Idea: Use left-to-right scan to impose a linear order on the tree nodes
- Implementation:
  - Use a default value to indicate empty node

Pointer Implementation: Delete Operation

- Problem: When you delete a node, what do you replace it by?
- Solution:
  1. If it has no children, by NULL
  2. If it has 1 child, by that child
  3. If it has 2 children, by the node with the smallest value in its right subtree, (or largest value in left subtree)
  Recursively delete node being used in 2 and 3
- Worst case: Recursion propagates all the way to a leaf node – time is $O(\text{depth of tree})$

Lazy Deletion

- 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
- 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 due to marked nodes – depth increases only by a constant amount if 50% are marked undeleted nodes
- 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 Binary Search Trees

✦ All BST operations (except MakeEmpty) are $O(d)$, where $d$ is tree depth
  ➤ MakeEmpty takes $O(N)$ for a tree with $N$ nodes – frees all nodes
✦ From last time, 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?
✦ So, best case running time of BST operations is $O(\log N)$
  ➤ In fact, average case is also $O(\log N)$ – see text
✦ Worst case running time is $O(N)$
  ➤ E.g. What happens when you insert elements in ascending order?
  ➤ Problem: Lack of "balance": compare depths of left and right subtree

Balancing Trees

✦ Many algorithms exist for keeping trees balanced
  ➤ Adelson-Velskii and Landis (AVL) trees (1962)
  ➤ Splay trees and other self-adjusting trees (1978)
  ➤ B-trees and other multiway search trees (1972)
✦ First try at balancing trees: Perfect balance
  ➤ Want a complete tree after every operation
  ➤ Too expensive E.g. Insert 2
  ➤ Need a looser constraint…

AVL Trees

✦ AVL trees are height-balanced binary search trees
✦ Balance factor of a node = height(left subtree) - height(right subtree)
✦ An AVL tree has balance factor of 1, 0, or -1 at every node
  ➤ For every node, heights of left and right subtree differ by no more than 1
  ➤ Store current heights in each node
✦ Can prove: Height is $O(\log N)$
  ➤ All operations (e.g. Find) are $O(\log N)$ except Insert (assume lazy deletion)

Insert and Rotation in AVL Trees

✦ Insert operation may cause balance factor to become 2 or -2 for some node on the path from insertion point to root node
  ➤ After Insert, back up to root updating heights
  ➤ If difference = 2 or -2, adjust tree by rotation around deepest such node
  ➤ Example:

1. Insert 1
2. Rotate
Insertion: Another Example

Tree before insertion (BF = Balance Factor)

Insertion: Example 1 (Outside case)

Tree after insertion

Insertion: Example 2 (Inside case)

Tree after insertion

Insertions in AVL Trees

Let the node that needs rebalancing be \( \alpha \).

There are 4 cases:

Outside Cases (require single rotation):
1. Insertion into left subtree of left child of \( \alpha \).
2. Insertion into right subtree of right child of \( \alpha \).

Inside Cases (require double rotation):
3. Insertion into right subtree of left child of \( \alpha \).
4. Insertion into left subtree of right child of \( \alpha \).

The rebalancing is performed through four separate rotation algorithms – on board examples. See text for details.