CSE 373: Self-Adjusting Trees
("Cookbook Data Structures")

Chapter 4
(and Section 12.2)

Motivation

Most Binary Search Tree Operations are O(d)

- d can range from O(log n) to O(n)
- generally, d is O(log n)
- statistically, d is O(log n) on average
- but, for "common" insertion orders, d can be O(n)
e.g., inserting sorted lists in order
A Solution

**Self-Adjusting Binary Search Trees:** BST’s that automatically rearrange themselves to keep operations $O(\log n)$
- AVL Trees
- Splay Trees
- Red-Black Trees

AVL Trees

**The idea:** A balance condition is placed on the tree. Whenever an `insert()` breaks the condition, we rearrange the tree to fix it.

What should the balance condition be?
AVL Tree Strategy

Balance Condition: Every node’s left and right subtrees must have a height difference of no more than one

Rotations

Rotation: a simple way of rearranging a tree without breaking the binary search property
Two Cases of Bad Inserts

Given:

Case I

Case II

Fixing Case I

rotate
Trying to Fix Case II

Fixing Case II
AVL Tree Summary

- Keep every node’s subtrees “almost balanced”
- When insertions break the “almost balanced” condition, use rotations to fix things up
- Use lazy deletion to keep things simple
- All operations are $O(\log n)$
- Implementation Cost:
  - must store depth of each node’s child subtrees
  - must implement 4 cases for bad insertion (2 × L,R)

Splay Trees

- Every time a node is accessed, rotate it to the top of the tree no matter what
- Over time, trees tend to get shallower since all accessed nodes get rotated towards the top
- Result: Although any one operation may require $O(n)$ time, a series of $k$ operations is guaranteed to be $O(k \log n)$ – amortized analysis
- Benefits:
  - no need to store depths of nodes’ subtrees
Red-Black Trees

**Red-Black Trees:** Binary Search Trees with the following properties:
- every node is colored either red or black
- the root is always black
- if a node is red, its children must be black
- every path from a node to a NULL pointer must contain the same number of black nodes
Example Red-Black Tree

Intuitively...
- every path from root to leaf has same number of black nodes
- though they may have different # red nodes, alternate at worst
- Thus, worst-case $d = 2\log(n) = O(\log n)$

Inserting into Red-Black Trees

```c
Insert (T, 0);
Insert (T, 3);
Insert (T, 22);
Insert (T, 23);
Insert (T, 24);
```