CSE 326: Data Structures
Topic #7: Don’t Sweat It - Splay It

Luke McDowell
Summer Quarter 2003

AVL Trees: Are They Worth It?

Advantages
- Rotations are cool!

Disadvantages
- Wouldn’t want to meet one in a dark alley at night

Splay What?
- Blind adjusting version of AVL trees
  - Why worry about balances? Just rotate anyway!
- Amortized time for all operations is O(log n)
- Worst case time is O(n)
  - But guaranteed to happen rarely
- Insert/Find always rotates node to the root!

Analogy: AVL is to Splay trees as….

Idea…
You’re forced to make a really deep access:

Since you’re down there anyway, fix up a lot of deep nodes!

Details…
1. Find or insert a node \( n \)
2. Splay \( n \) to the root using: zig-zag, zig-zig, or plain ol’ zig
3. Helps the new root (\( n \)) and many others!

Zig-Zag*

*Just like an… Which nodes improve depth?
Is this an AVL zig-zig? How to implement?

Why does this help?

Special Case: Zig

Final result from splaying to root:
Relative depth of p, Y, Z?
Relative depth of everyone else?

Why not drop zig-zig and just zig all the way?

Splaying Example

Find(6)

Still Splaying 6

Almost There, Stay on Target

Splay Again

Find(4)
Example Splayed Out

Why Splaying Helps

- If a node $n$ on the access path is at depth $d$ before the splay, it’s at about depth $d/2$ after the splay
  - Exceptions are the root, the child of the root (and descendants), and the node splayed
- Overall, nodes which are below nodes on the access path tend to move closer to the root
- Splaying gets amortized $O(\log n)$ performance.

Splay Operations: Find

- Find the node in normal BST manner
- Splay the node to the root

Splay Operations: Insert

- Insert the node in normal BST manner
- Splay the node to the root

Splay Operations: Remove

Now what?

Join

- Join(L, R): given two trees such that L < R, merge them
- Splay on the maximum element in L then attach R

Does this work to join any two trees?
Nifty Splay Operation: Splitting

- Split(T, x) creates two BSTs L and R:
  - all elements of T are in either L or R (T = L ∪ R)
  - all elements in L are < x
  - all elements in R are > x
  - L and R share no elements (L ∩ R = ∅)

How do we split a splay tree?

Pssstt: Another Way to Insert

Interesting note: split-and-insert was the original algorithm. But insert-and-splay has better constants

Splay Tree Summary

- All operations are in amortized O(log n) time
- Splaying can be done top-down; better because:
  - only one pass
  - no recursion or parent pointers necessary
- Splay trees are very effective search trees
  - Relatively simple
  - No extra fields required
  - Excellent locality properties: frequently accessed keys are cheap to find

Coming Up

- Really big search trees
- Hashing
- More on HW #3’s mysterious benefactor…

To Do

- Finish Chapter 4, start Chapter 5
- Continue HW 3 – part B due Tuesday, 11 p.m.