First, a Random Question
1. The average depth of a node in randomly-built binary search tree on \( n \) nodes is \( O(\log n) \).
   ➢ we showed this in class
2. The average height of a randomly-built binary search tree on \( n \) nodes is \( O(\log n) \).
   ➢ a stronger statement, still true
3. The average height of a binary tree with \( n \) nodes is \( \Theta(\sqrt{n}) \)

   Why is this not contradictory?

Are All Binary Trees Equally Likely to be Built?

➢ How many ways are there to sequence the numbers 1, 2, 3?

➢ Which of the binary trees can be built in more than one way?

Pros and Cons of AVL Trees

Arguments for AVL trees:
1. Search is \( O(\log N) \) since AVL trees are always balanced.
2. The height balancing adds no more than a constant factor to the speed of insertion.

Arguments against using AVL trees:
1. Difficult to program & debug; more space for height info.
2. Asymptotically faster but usually slower in practice!

More Treelike Data Structures

➢ Today: Splay Trees
   • Fast both in amortized analysis and in practice
     • Are used in the kernel of NT for keep track of process information!
     • Invented by Sleator and Tarjan (1985)
   • Good “locality”
   • Details:
     • Weiss 4.5 (basic splay trees)
     • 11.5 (amortized analysis)
     • 12.1 (better “top down” implementation)
➢ Coming up: B-Trees

Splay Trees

“Blind” rebalancing – no height info kept
➢ amortized time for all operations is \( O(\log n) \)
➢ worst case time is \( O(n) \)
➢ insert/find always rotates node to the root!
   • Good locality – most common keys move high in tree
You're forced to make a really deep access:

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

Zig-Zag

Helped
Unchanged
Hurt

This is just a double rotation

Why Splaying Helps

- Node n and its children are always helped (raised)
- Except for final zig, nodes that are hurt by a zig-zag or zig-zig are later helped by a rotation higher up the tree!

Result:
- shallow (zig) nodes may increase depth by one or two
- helped nodes may decrease depth by a large amount
- If a node n on the access path is at depth d before the splay, it's at about depth \( \frac{d}{2} \) after the splay
- Exceptions are the root, the child of the root, and the node splayed

Locality

- “Locality” – if an item is accessed, it is likely to be accessed again soon
  - Why?

- Assume \( m \geq n \) access in a tree of size \( n \)
  - Total amortized time \( O(m \log n) \)
  - \( O(\log n) \) per access on average

- Suppose only \( k \) distinct items are accessed in the \( m \) accesses.
  - Time is \( O(m \log k + n \log n) \)
  - What would an AVL tree do?
Splaying Example

Almost There, Stay on Target

Example Splayed Out

Still Splaying 6

Splay Again

Splay Operations: Insert

- To insert, could do an ordinary BST insert
  - but would not fix up tree
  - A BST insert followed by a find (splay)?
- Better idea: do the splay before the insert!
- How?
Splay Operations: Insert
➢ To insert, could do an ordinary BST insert
  ▪ but would not fix up tree
  ▪ A BST insert followed by a find (splay)?
➢ Better idea: do the splay before the insert!
➢ How?
➢ Split(\(T, x\)) creates two BSTs \(L\) and \(R\):
  ▪ all elements of \(T\) are in either \(L\) or \(R\) (\(T = L \cup R\))
  ▪ all elements in \(L\) are \(\leq x\)
  ▪ all elements in \(R\) are \(\geq x\)
  ▪ \(L\) and \(R\) share no elements (\(L \cap R = \emptyset\))
  
  Then how do we do the insert?

Splitting in Splay Trees
➢ How can we split? (SPOILERS below ^L)
  ▪ We have the splay operation.
  ▪ We can find \(x\) or the parent of where \(x\) should be.
  ▪ We can splay it to the root.
  ▪ Now, what’s true about the left subtree of the root?
  ▪ And the right?

Splay Operations: Delete
➢ Find \(x\)
➢ Delete \(x\)
➢ Join \(L, R\): given two trees such that \(L < R\), merge them

Join
➢ Join(\(L, R\)): given two trees such that \(L < R\), merge them
  ▪ Splay on the maximum element in \(L\) then attach \(R\)
**Delete Completed**

- **T** → **find(x)** → **delete x** → **L** → **R**
- **Joint(L,R)** → **T - x**

**Delete Example**

- **Delete(4)** → **find(4)** → **delete(4)** → **find max** → **For Wednesday**

- **Read 4.7**
- **You should be well on your way to completing assignment 3**