

CSE 326: Data Structures

Splay Trees

Brian Curless
Spring 2008

1

Announcements (4/25/08)

- No homework assigned this week
- Midterm next Friday
 - Closed notes, book
 - It will cover everything through today, maybe part of Monday's lecture.
 - It will not be a test of your Java knowledge.
 - It will not have hard proofs.
 - I will say more next week.
- Reading for this lecture is in Weiss, Chapter 4.

2

Pros and Cons of AVL Trees

Arguments for AVL trees:

1. Search is $O(\log n)$ since AVL trees are *always well balanced*.
2. The height balancing adds no more than a constant factor to the speed of insertion and deletion; thus both are $O(\log n)$.

Arguments against using AVL trees:

1. Must store and track height info.
2. May be fine to have $O(n)$ for a single operation if amortized cost can be $O(\log n)$ (e.g., Splay trees).
3. Very large searches are done in database systems and need efficient strategies based on disk access costs (e.g., B-trees).

3

Splay Trees

- Blind adjusting version of AVL trees
 - Why worry about balances? Just rotate anyway!
- Worst case time for an operation is $O(n)$
- Amortized time per operation is $O(\log n)$

SAT/GRE Analogy question:

AVL tree : Splay tree :: Leftist heap : *Skew heap*

4

Webster's Dictionary Definition

Main Entry:

³splay

Function:

adjective

Date:

1548

1 : turned outward <*splay knees*>

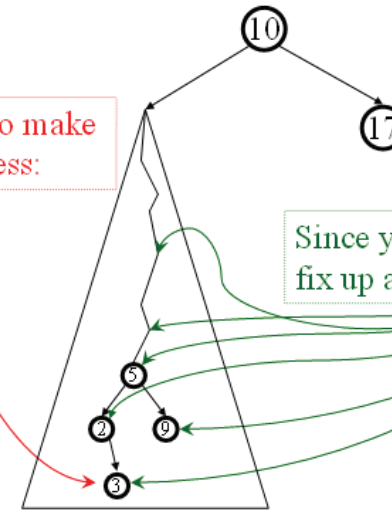
2 : awkward, ungainly

5

The Splay Tree Idea

If you're forced to make a really deep access:

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



6

Find/Insert in Splay Trees

1. Find or insert a node k
2. **Percolate k to the root with rotations.**

Instead of “percolate k to the root with rotations” we will often just say “*splay k to the root*”.

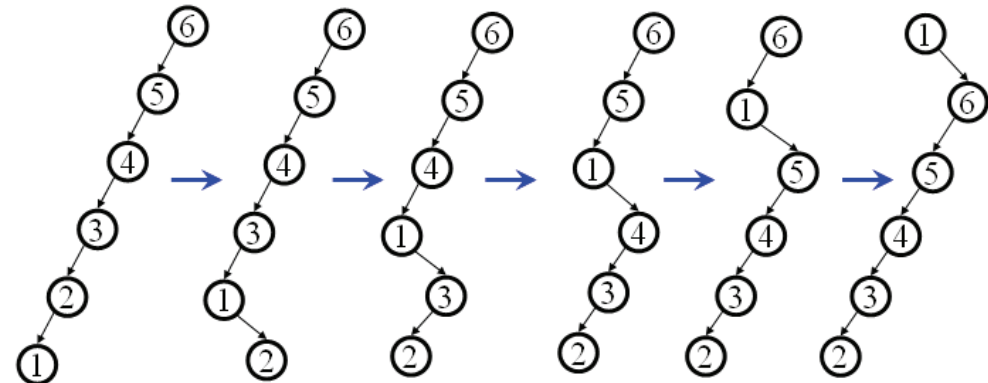
Why could this be good??

1. Helps the new root, k
 - o Great if k is accessed again soon
2. And helps many others on the path to k !
 - o Great if others on the path are accessed soon

7

Do it all with AVL single rotations?

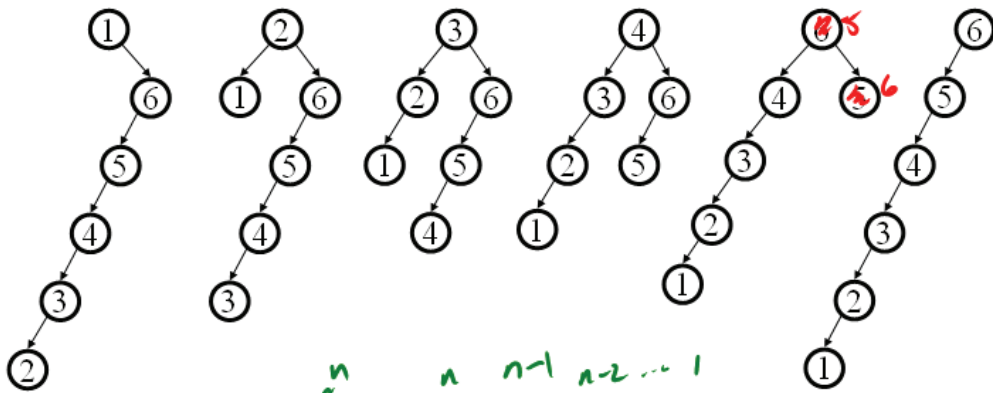
Consider the ordered “list tree” at left. Now do **find(1)** and splay it to the root with only AVL single rotations:



8

Do it all with AVL single rotations?

find(1) find(2) find(3) find(4) find(5) find(6)



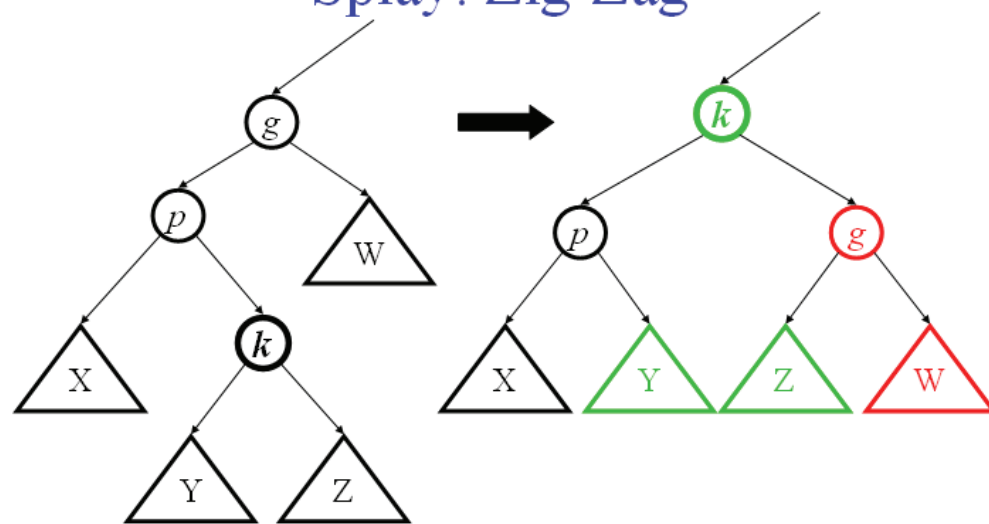
Cost of sequence: find(1), find(2), ..., find(n)?

$$\sum_{i=1}^n i \in O(n^2)$$

Single rotations can help, but they are not enough...

9

Splay: Zig-Zag

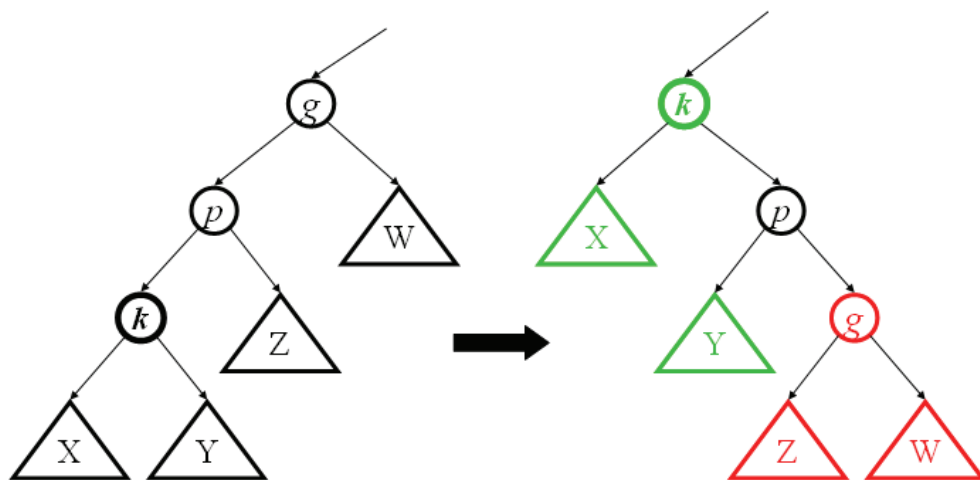


Relationship to AVL rotations?

AVL double rotation

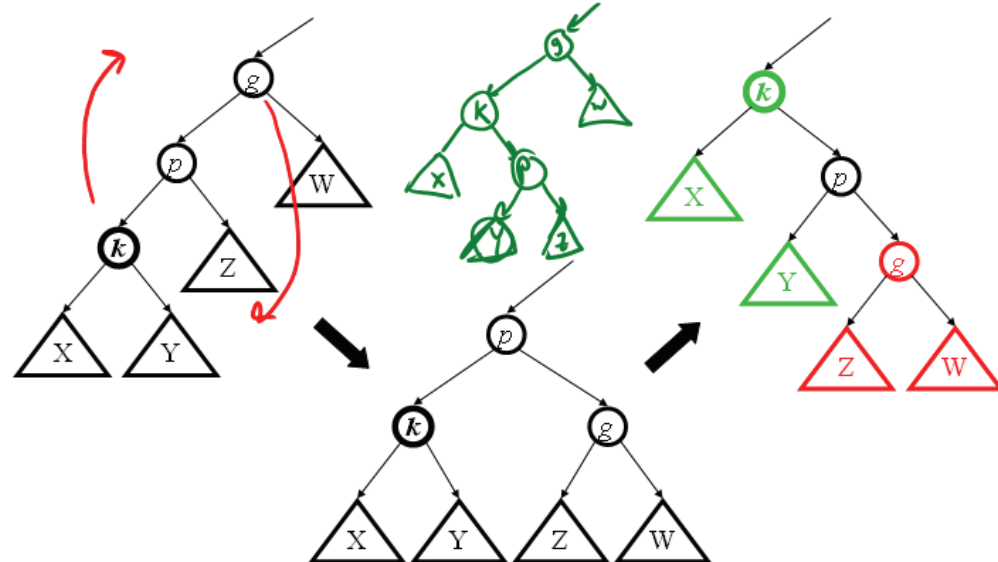
10

Splay: Zig-Zig



11

Splay: Zig-Zig

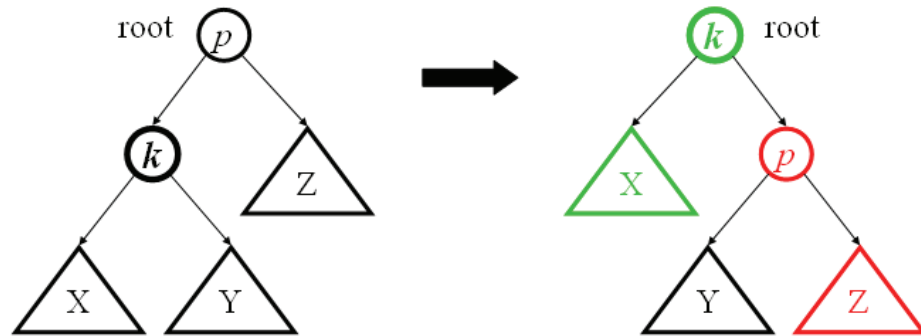


Relationship to AVL rotations?

2 single rotations (p,g), (k,p)

12

Special Case for Root: Zig

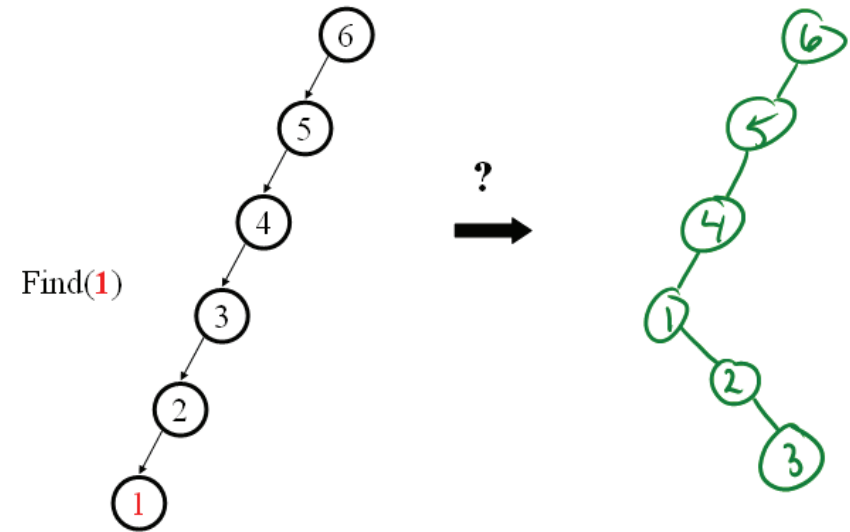


Relationship to AVL rotations?

AVL Single rotation

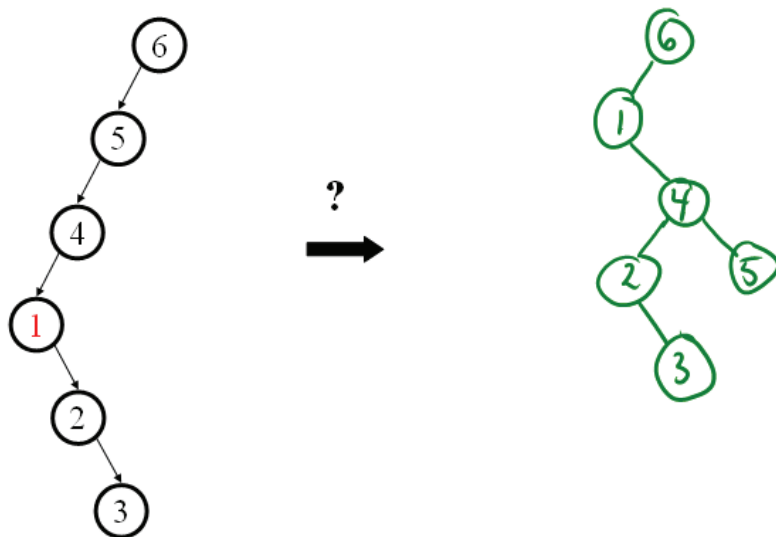
13

Splaying Example: Find(1)



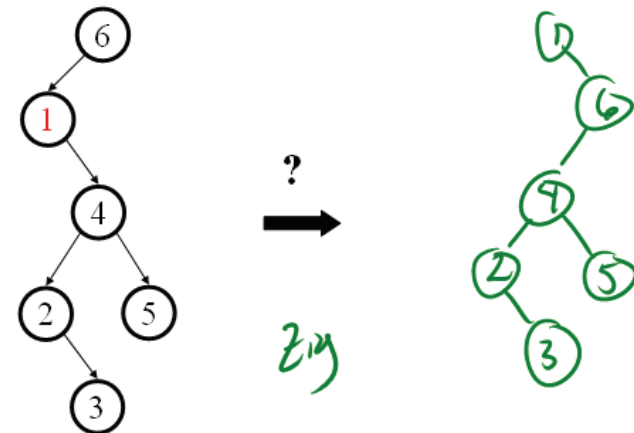
14

Find(1) continued...



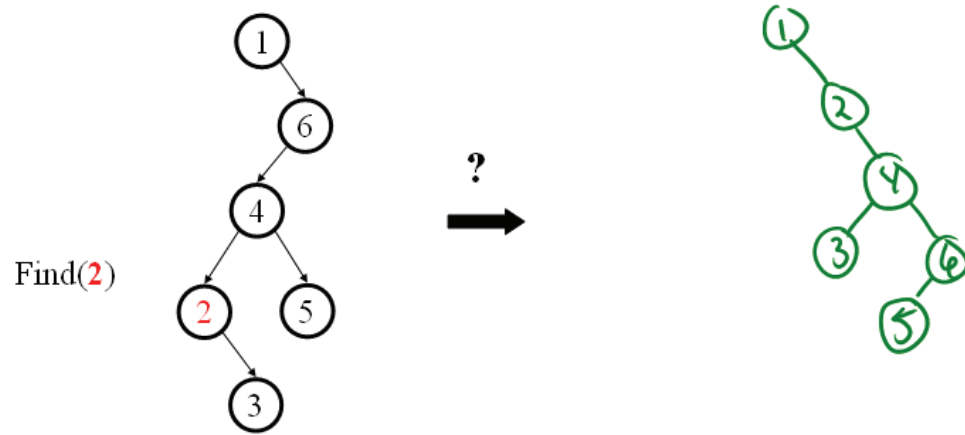
15

Find(1) finished



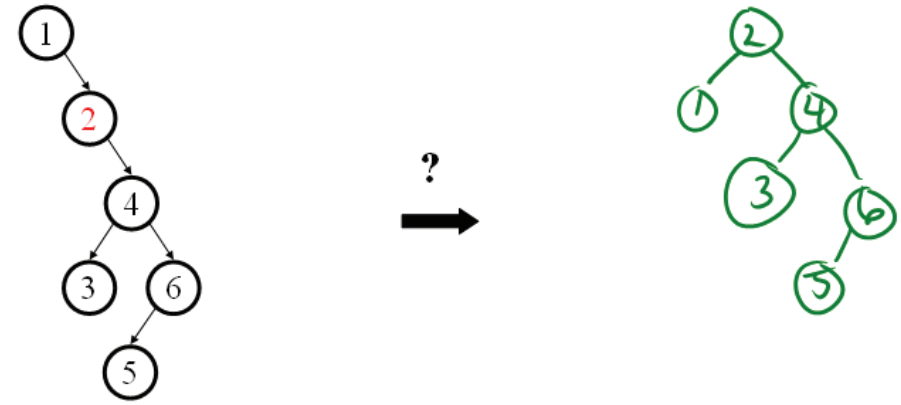
16

Another Splay: Find(2)



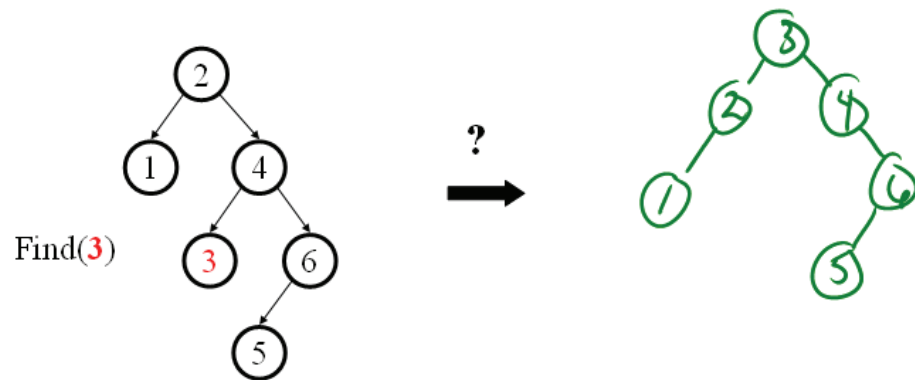
17

Find(2) finished



18

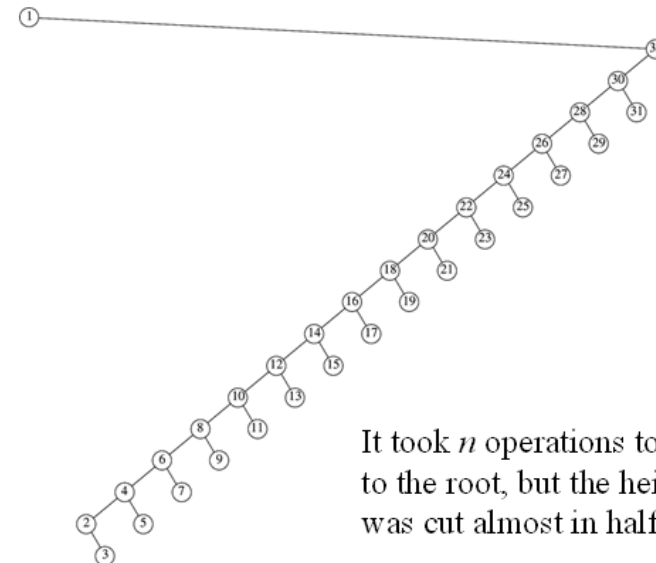
Another Splay: Find(3)



19

A bigger, badder example

Consider a “list tree” from 1-32. Doing find(1) gives:

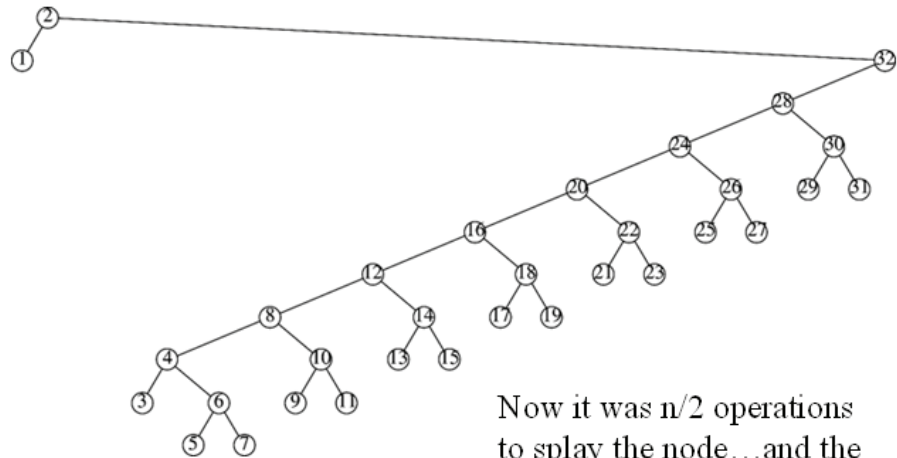


It took n operations to bring that node to the root, but the height of the tree was cut almost in half!

20

A bigger, badder example

Now do `find(2)`:

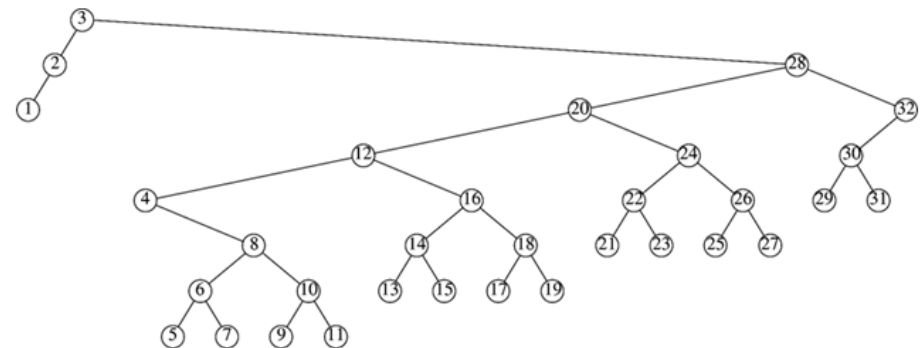


Now it was $n/2$ operations to splay the node...and the tree height is cut in $\sim$ half again!

21

A bigger, badder example

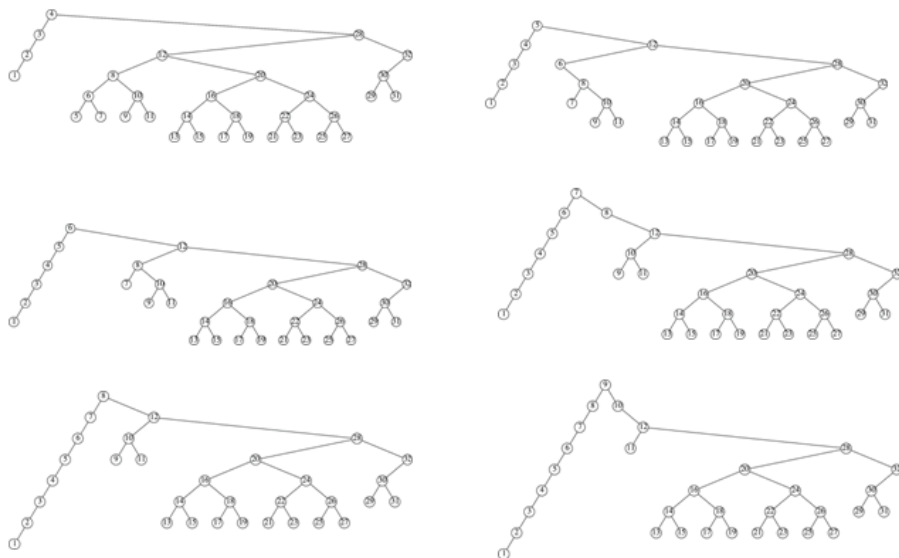
Now do `find(3)`:



The height is cut in $\sim$ half again.

22

A bigger, badder example



23

Why Splaying Helps

- If a node x on the access path is at depth d before the splay, it's typically at about depth $d/2$ after the splay
- Overall, nodes which are low on the access path tend to move closer to the root
- Splaying gets amortized $O(\log n)$ performance.

24

Practical Benefit of Splaying

- No heights to maintain, no imbalance to check for
 - Less storage per node, easier to code
- Data accessed once, is often soon accessed again
 - Splaying does implicit *caching* by bringing it to the root

25

Splay Operations: Find, Insert

Find(x):

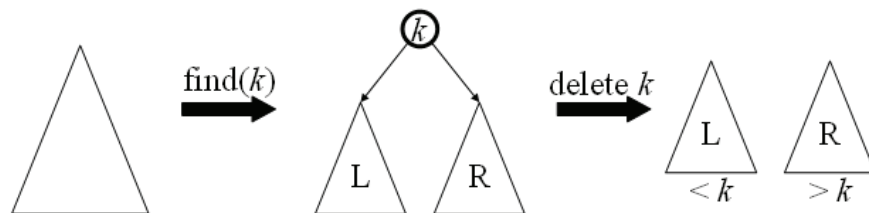
- Find the node in normal BST manner
- Splay the node to the root
 - if node not found, splay what would have been its parent

Insert(x):

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

26

Splay Operations: Remove



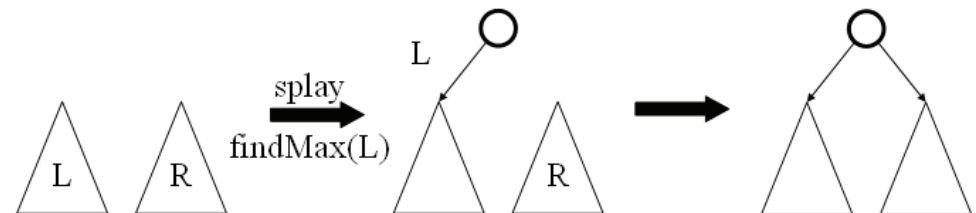
Now what?

27

Join

Join(L, R):

given two trees such that (stuff in L) $<$ (stuff in R), merge them:

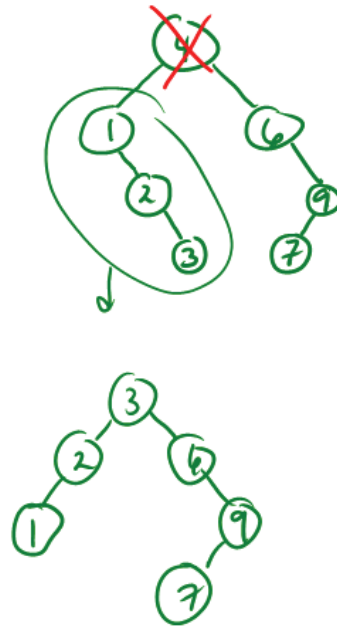
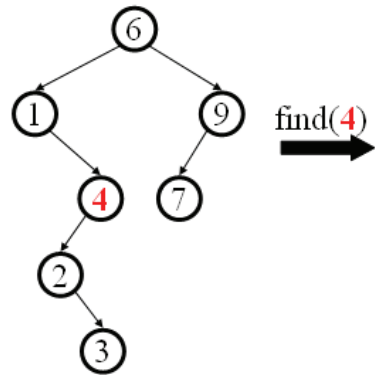


Splay on the maximum element in L , then attach R

28

Delete Example

Delete(4)



29

Splay Tree Summary

All operations are in amortized $O(\log n)$ time

Splaying can be done top-down; this may be better because:

- only one pass
- no recursion or parent pointers necessary
- (we didn't cover top-down in class)

Splay trees are *very* effective search trees

- Relatively simple
- No extra fields required
- Excellent *locality* properties in the following sense:
frequently accessed keys are cheap to find

30