6.006	Lecture 4	Feb. 13, 2008

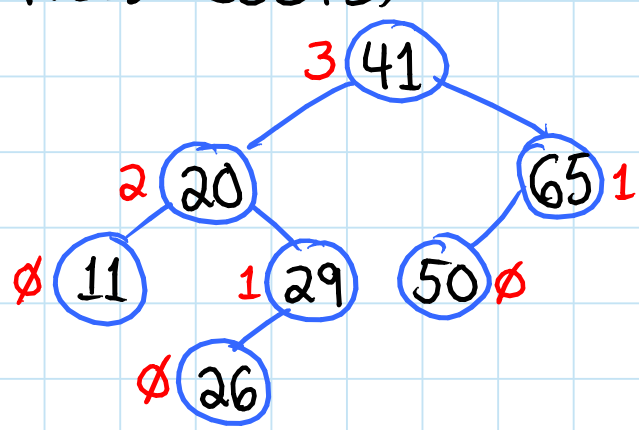
Outline: Balanced BSTs

- The importance of being balanced
- AVL trees
  - definition
  - balance
  - insert
- Other balanced trees
- Data structures in general

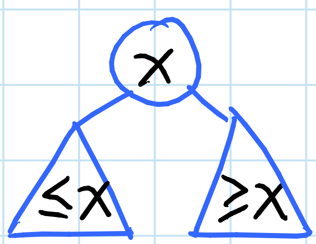
Reading: CLRS 13.1 & 13.2
(but different approach: red-black trees)

Recall: Binary Search Trees (BSTs)

- rooted binary tree
- each node has
  - key
  - left pointer
  - right pointer
  - parent pointer



- BST property:

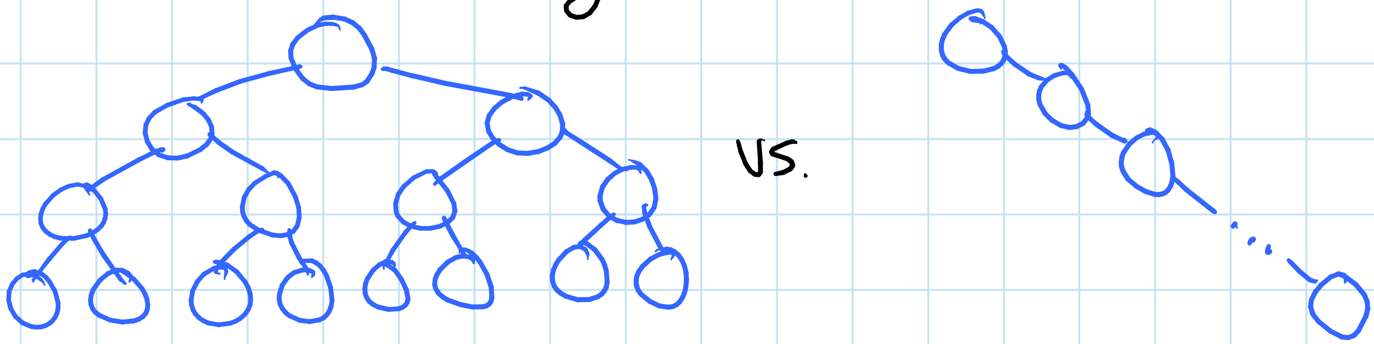


CLRS B.5

- height of node = length (# edges) of longest downward path to a leaf

The importance of being balanced:

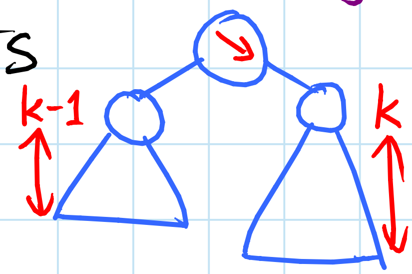
- BSTs support insert, min, delete, rank, etc. in $O(h)$ time, where h = height of tree (= height of root)
- h is between $\lg n$ and n:



perfectly balanced — path

- balanced BST maintains $h = O(\lg n)$
  $\Rightarrow$ all operations run in $O(\lg n)$ time

AVL trees: [Adel'son-Vel'skii & Landis 1962]



for every node, require heights of left & right children to differ by at most ± 1

- treat nil tree as height -1
- each node stores its height (DATA STRUCTURE AUGMENTATION) (like subtree size)
  (alternatively, can just store difference in heights)

Balance: worst when every node differs by 1

- let N_h = (min.) # nodes in height-h AVL tree

$\Rightarrow N_h = N_{h-1} + N_{h-2} + 1$
$\quad > 2 N_{h-2}$

$\Rightarrow N_h > 2^{h/2}$

$\Rightarrow h < \frac{1}{2} \lg h$

Alternatively: $N_h > F_h$ (nth Fibonacci number)

- in fact $N_h = F_{h+2} - 1$ (simple induction)
- $F_h = \varphi^h / \sqrt{5}$ rounded to nearest integer where $\varphi = \frac{1+\sqrt{5}}{2} \approx 1.618$ (golden ratio)

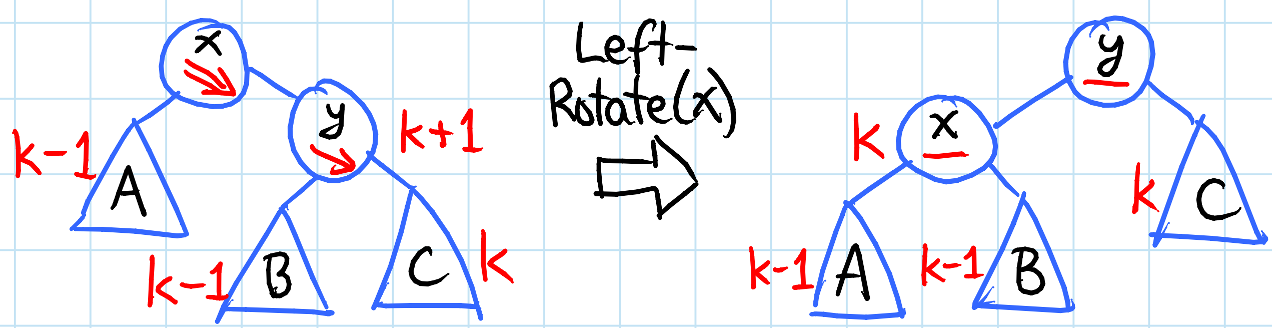
$\Rightarrow$ max. $h \approx \log_\varphi n \approx 1.440 \lg n$

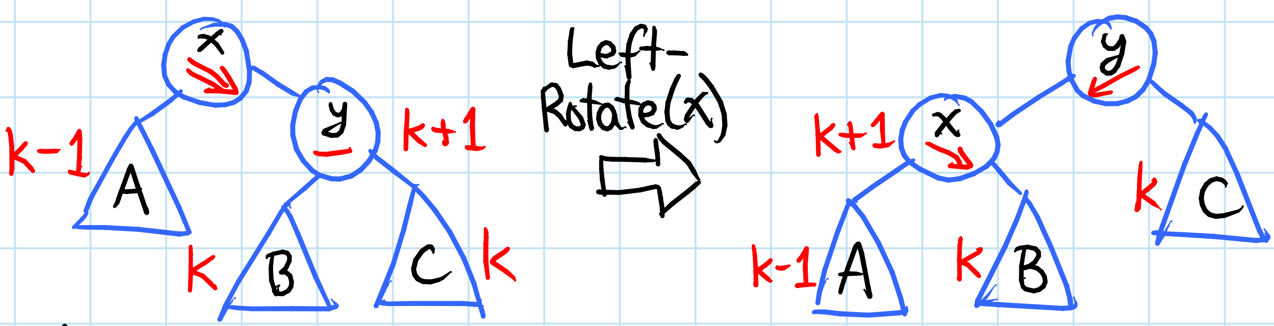
AVL insert:

1. insert as in simple BST
2. work your way up tree, restoring AVL property (and updating heights as you go)

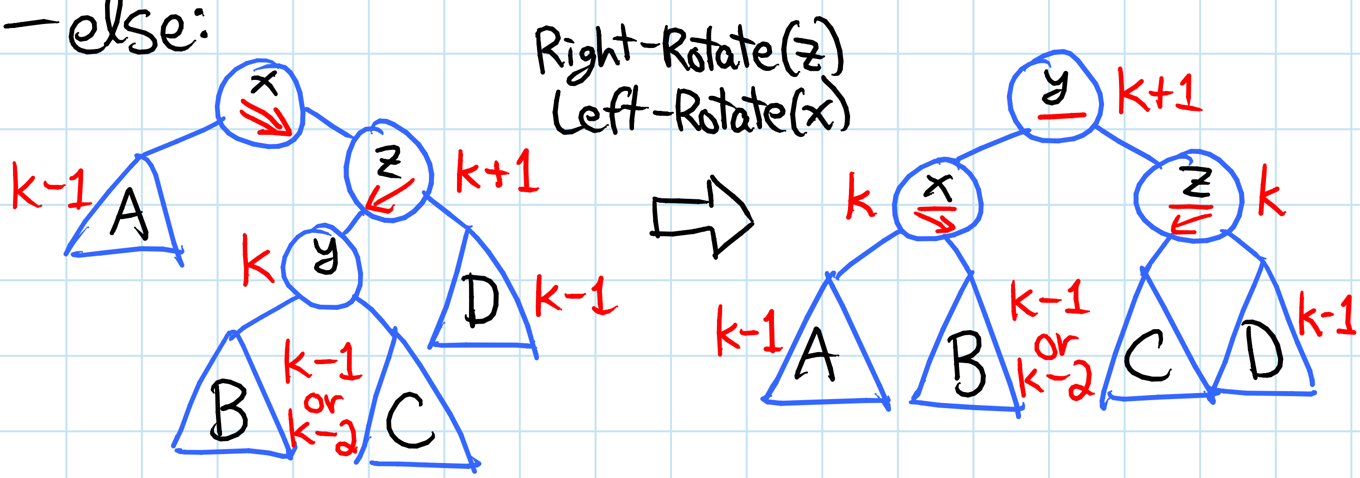
Each step:

- suppose x is lowest node violating AVL
- assume x is right-heavy (left case symmetric)
- if x's right child is right-heavy or balanced:





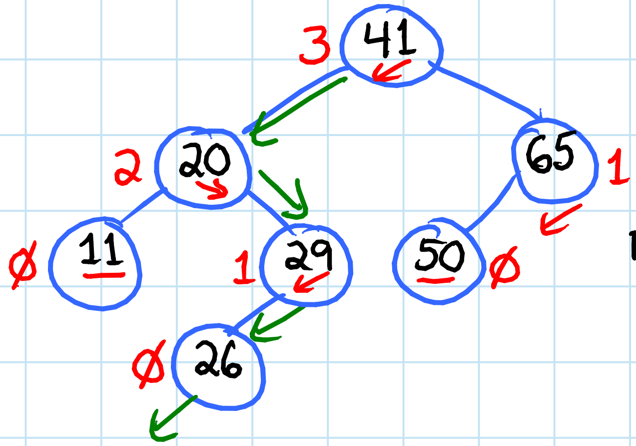
- else:



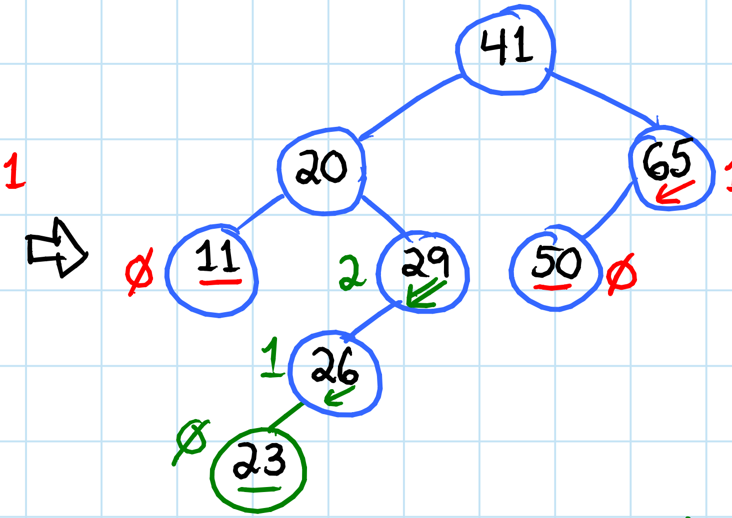
- then continue up to x's grandparent, greatgp, ...

Example:

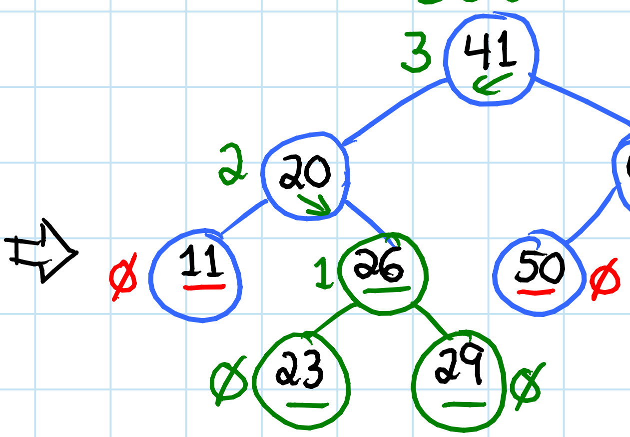
Insert(23)



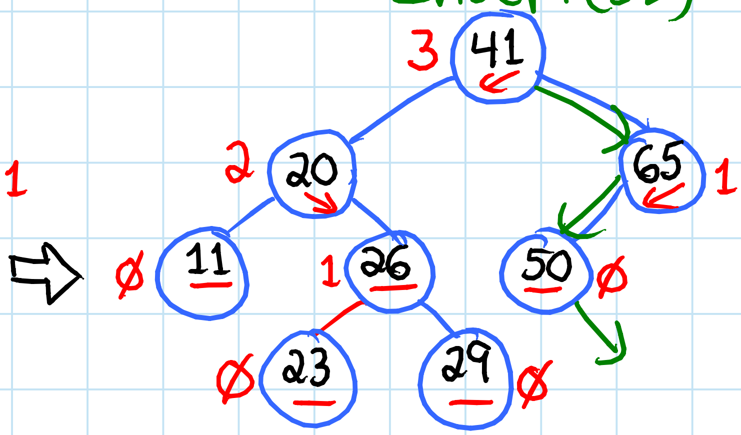
x=29: left-left case



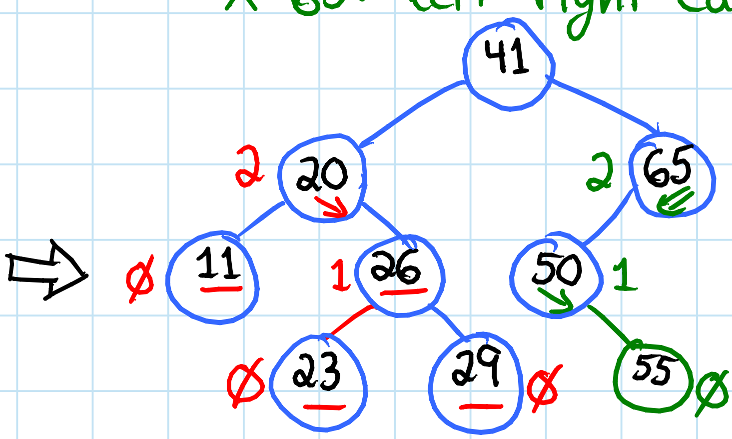
Done.



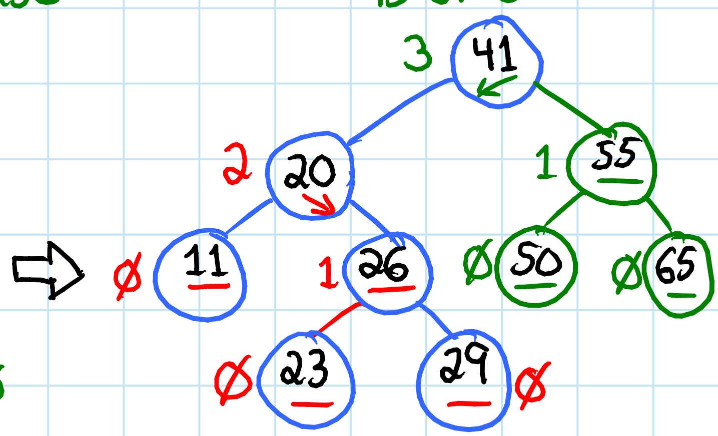
Insert(55)



x=65: left-right case



Done.



- in general may need several rotations before done with an Insert
- Delete(-min) harder but possible

Balanced search trees: there are many!

- AVL trees [Adel'son-Vels'kiĭ & Landis 1962]
- B-trees / 2-3-4 trees [Bayer & McCreight 1972] CLRS 18
- BB[α] trees [Nievergelt & Reingold 1973]
- red-black trees [CLRS ch. 13]
- Ⓐ – splay trees [Sleator & Tarjan 1985]
- Ⓡ – skip lists [Pugh 1989]
- Ⓐ – scapegoat trees [Galperin & Rivest 1993]
- Ⓡ – treaps [Seidel & Aragon 1996]

Ⓡ = use random numbers to make decisions fast with high probability

Ⓐ = "amortized": adding up costs for several operations ⇒ fast on average

Splay trees:

upon access (search or insert), move node to root by sequence of rotations and/or double-rotations (just like AVL trees)

- height can be linear!
- but still $O(\lg n)$ per operation "on average" (amortized)

(we'll see more about amortization in a couple of lectures)

Optimality:

- for BSTs, can't do better than $\Theta(\lg n)$ per search in worst case
- in some cases can do better, e.g.:
  - in-order traversal takes $\Theta(n)$ time for n elements
  - put more frequent items near root

CONJECTURE: splay trees are O(best BST) for every access pattern

- with fancier tricks, can achieve $O(\lg \lg u)$ performance for integers $1..u$
  [van Emde Boas: see 6.854 or 6.851]
  Advanced Data Structures

Big picture:

Abstract Data Type (ADT): interface spec.

e.g. PRIORITY QUEUE:

- Q = new-empty-queue()
- Q.insert(x)
- x = Q.deletemin()

vs. Data Structure (DS): algorithm for each op.

- many possible DSs for one ADT
  e.g. much later, "heap" priority queue