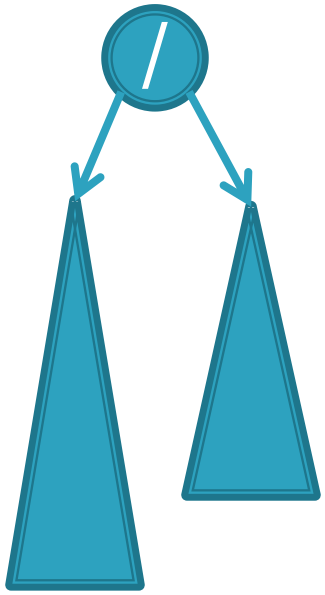


CSSE 230 Day 13

AVL trees and rotations

This week, you should be able to…
…perform rotations on height-balanced trees,
on paper and in code
… write a rotate() method
… search for the kth item in-order using rank

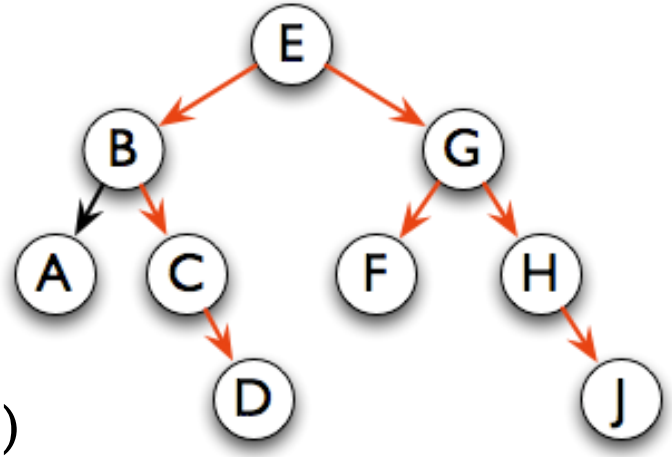
Announcements

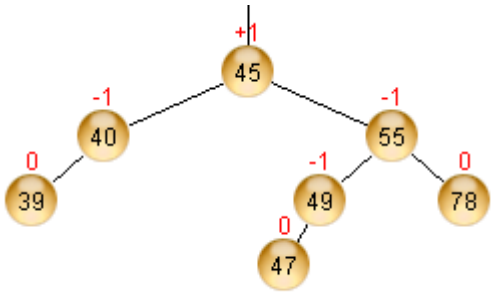
- See schedule page

Summary: for fast tree operations, we must keep the tree somewhat balanced in O(log n) time

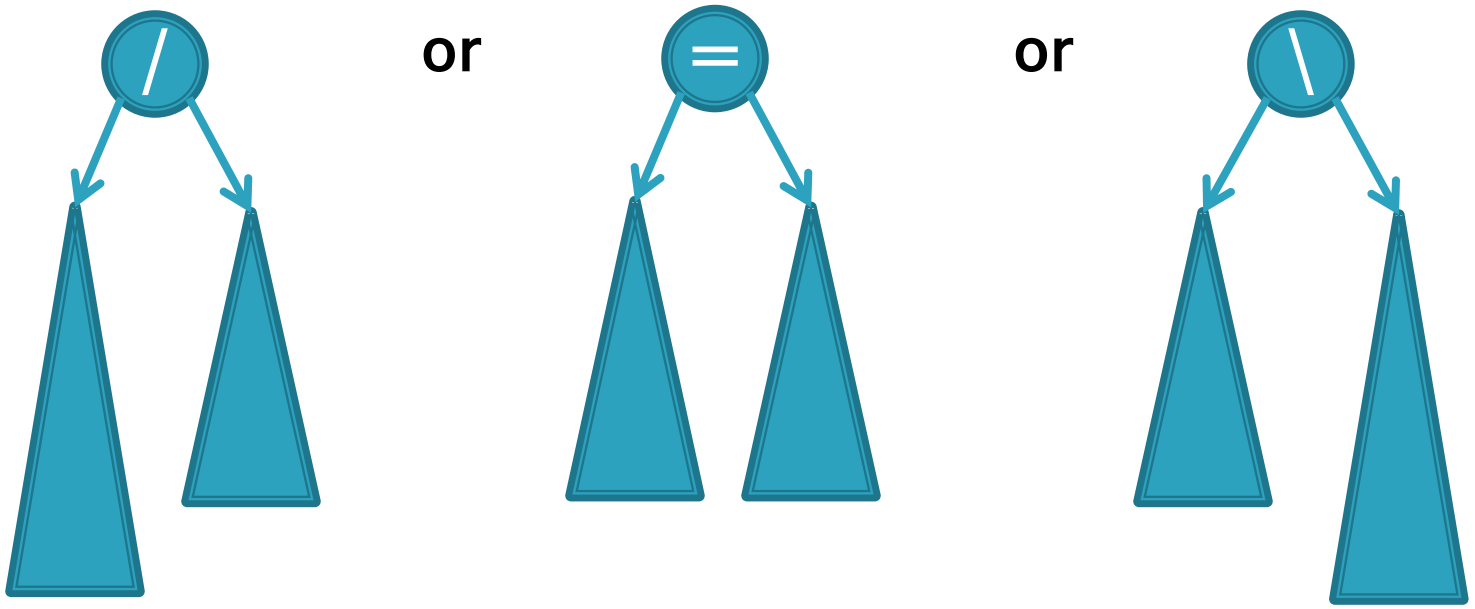
Q1

- Operations (insert, delete, search) are O(height)
- Tree height is O(log n) if perfectly balanced
  - But maintaining perfect balance is O(n)
- Height-balanced trees are still O(log n)
  - For T with height h, N(T) ≤ Fib(h+3) – 1
  - So H < 1.44 log (N+2) – 1.328 *
- AVL (**A**delson-**V**elskii and **L**andis) trees maintain height-balance using rotations
- Are rotations O(log n)? We'll see…





AVL nodes are just like BinaryNodes, but also have an extra "balance code"

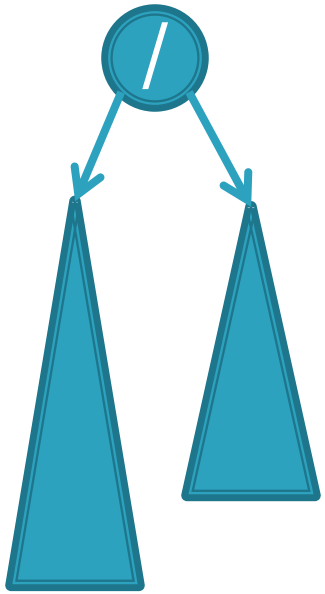


Different representations for / = \ :

- Just two bits in a low-level language
- Enum in a higher-level language

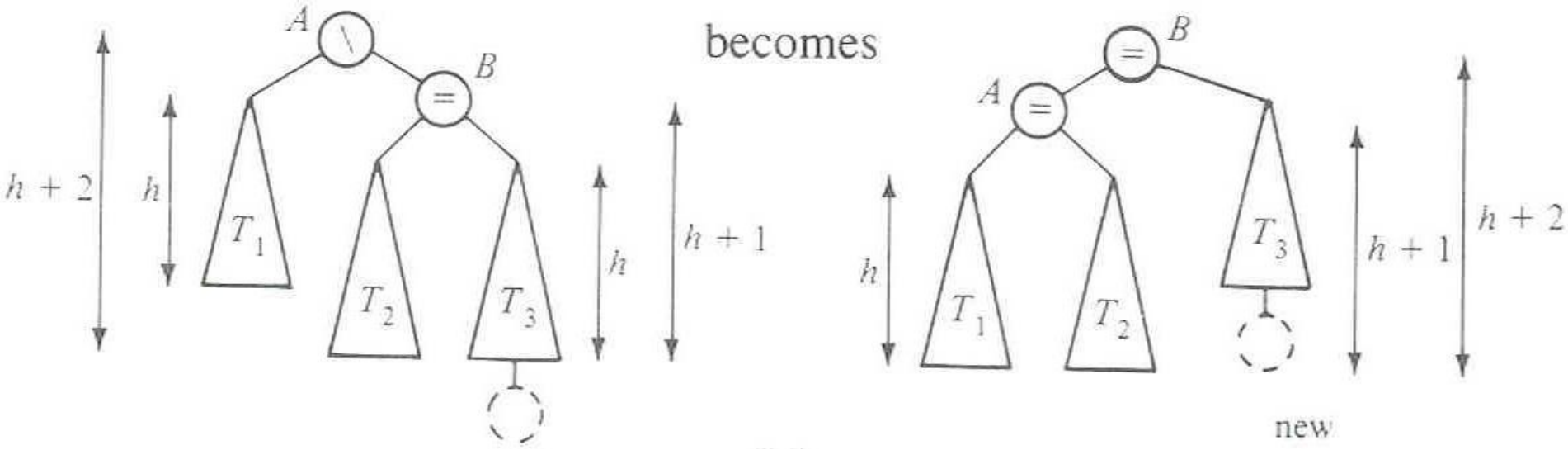
AVL Tree (Re)balancing Act

- Assume tree is height-balanced before insertion
- Insert as usual for a BST
- Move up from the newly inserted node to the lowest "unbalanced" node (if any)
  - Use the **balance code** to detect unbalance – how?
- Do an appropriate rotation to balance the sub-tree rooted at this unbalanced node



Four types of rotations are required to remove different cases of tree imbalances

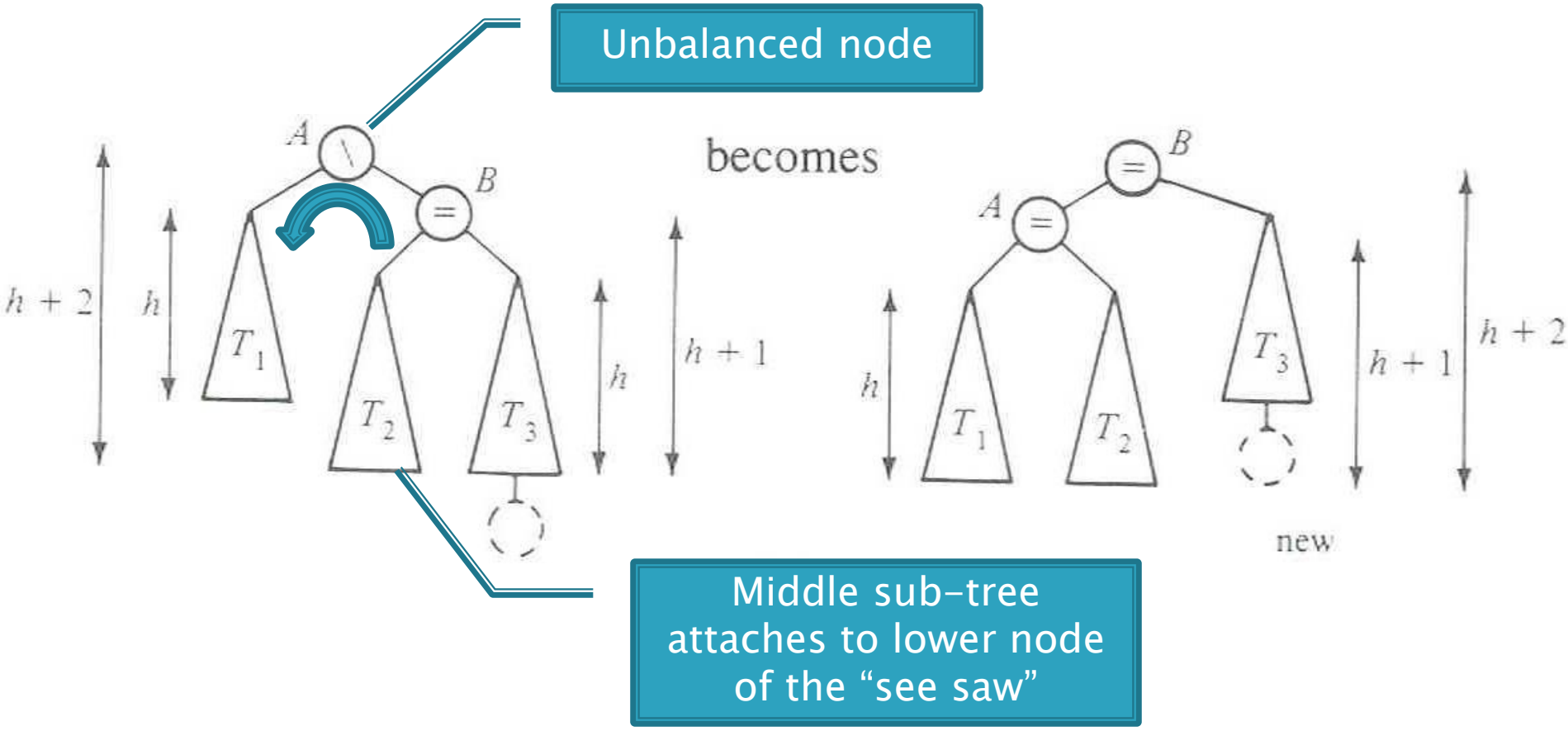
- For example, a *single left rotation*:



We rotate by pulling the “too tall” sub-tree up and pushing the “too short” sub-tree down

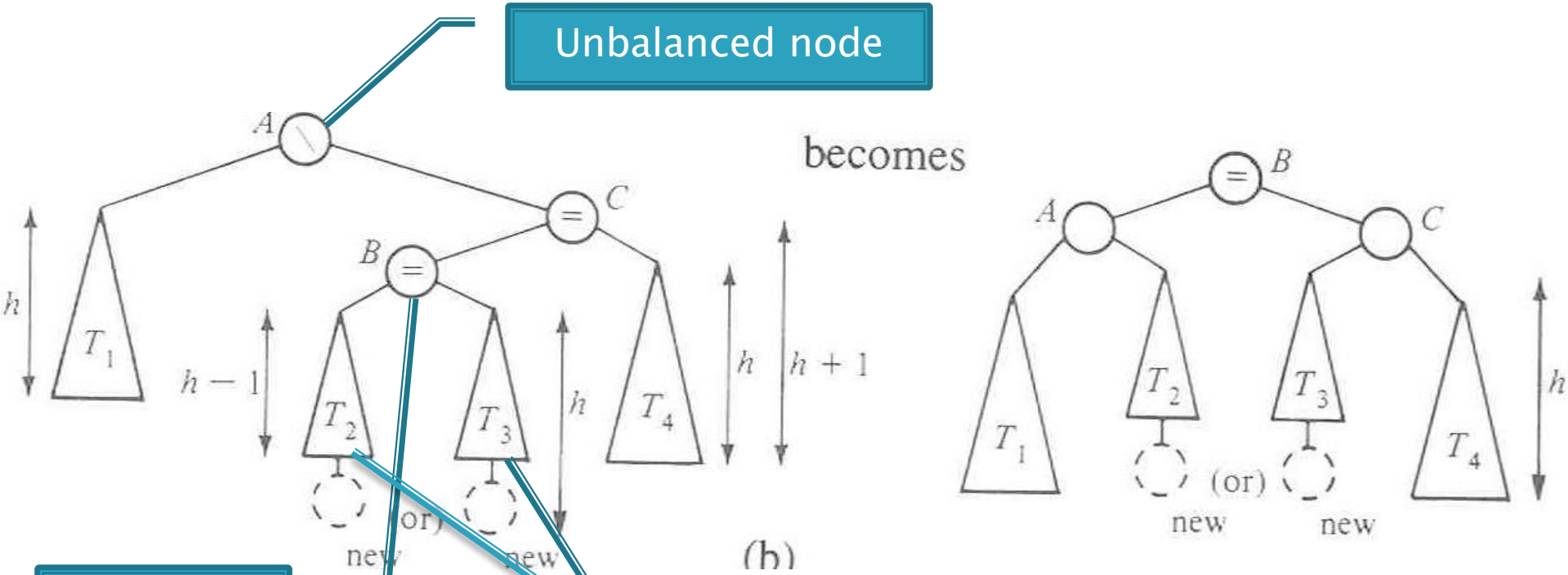
- Two basic cases
  - “See saw” case:
    - Too-tall sub-tree is on the outside
    - So tip the see saw so it’s level
  - “Suck in your gut” case:
    - Too-tall sub-tree is in the middle
    - Pull its root up a level

Single Left Rotation



Diagrams are from *Data Structures* by E.M. Reingold and W.J. Hansen

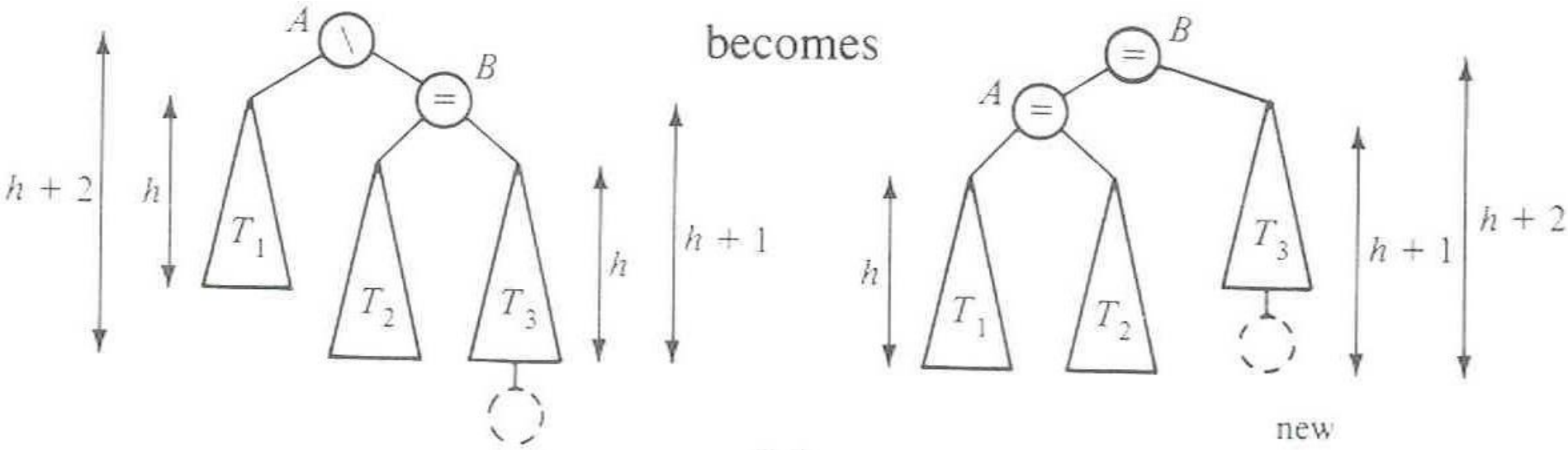
Double Left Rotation



Weiss calls this “right-left double rotation”

Your turn — work with a partner

Q6



- Write the method:
- ```
  static BalancedBinaryNode singleRotateLeft (
      BalancedBinaryNode parent,   /* A */
      BalancedBinaryNode child     /* B */  ) {

  }
  ```
- Returns a reference to the new root of this subtree.
- Don't forget to set the balanceCode fields of the nodes.
  ```

More practice— (sometime after class)

- Write the method:
- ```
  BalancedBinaryNode doubleRotateRight (
    BalancedBinaryNode parent,      /* A */
    BalancedBinaryNode child,       /* C */
    BalancedBinaryNode grandChild   /* B */ ) {


  }
  ```
- Returns a reference to the new root of this subtree.
- Rotation is mirror image of double rotation from an earlier slide
  ```

O(log N)?

- Both kinds of rotation leave height the same as before the insertion!

- Is insertion plus rotation cost really O(log N)?

Insertion/deletion in AVL Tree:	O(log n)
Find the imbalance point (if any):	O(log n)
Single or double rotation:	O(1)
in deletion case, may have to do O(log N) rotations	
Total work:	O(log n)

Term Project: EditorTrees

Height-balanced, but not AVL
Insertion/deletion by **index**, not by comparing elements

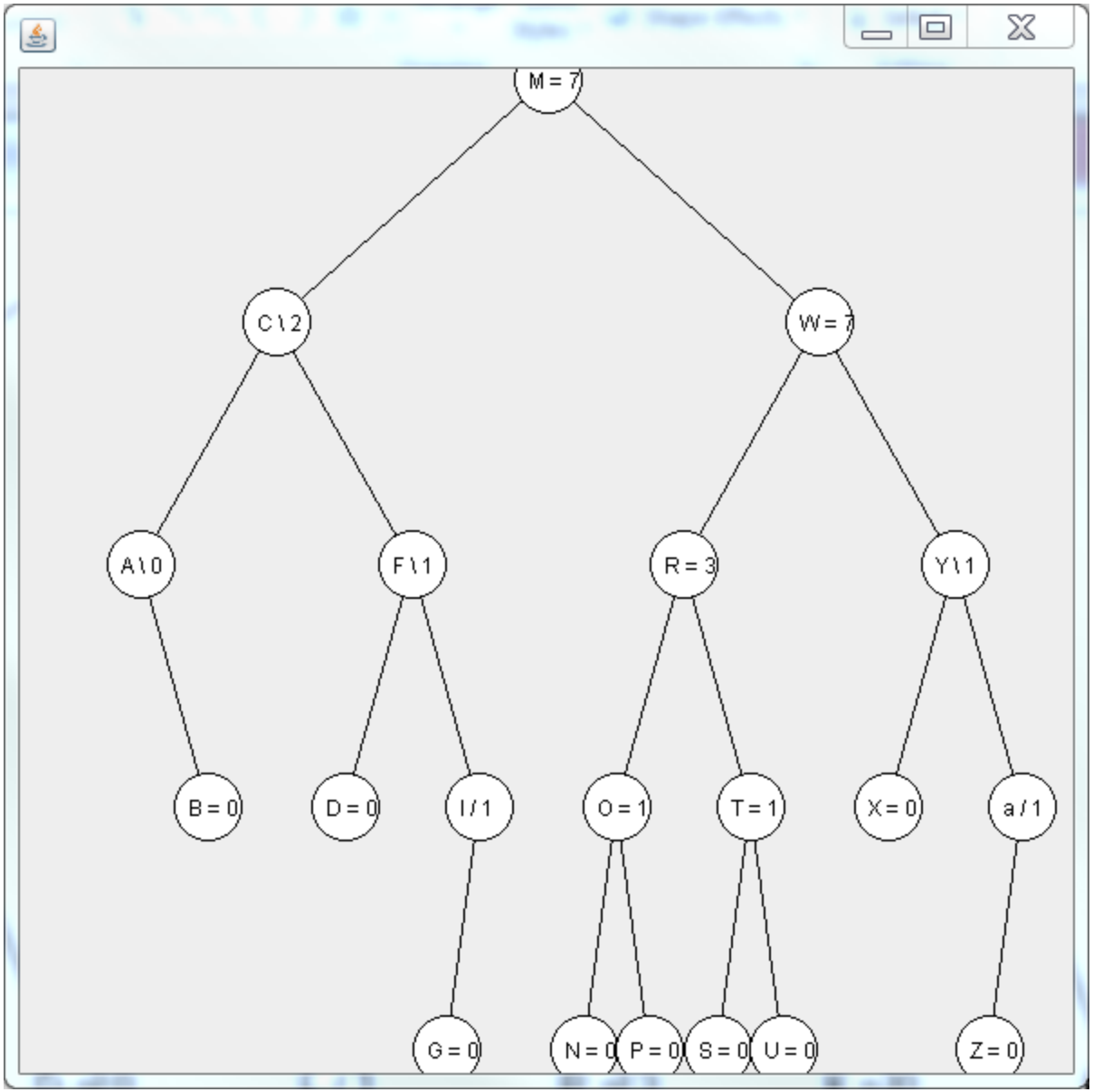
How do we find the k^{th} element?

We can find the kth element easily if we add a `rank` field to BinaryNode

- Gives the in-order position of this node within its own subtree
  - i.e., the size of its left subtree

0-based indexing

- How would we do `findK`$_{th}$?
- `Insert` and `delete` start similarly


M = 7
C \ 2
W = 7
A \ 0
F \ 1
R = 3
Y \ 1
B = 0
D = 0
I / 1
O = 1
T = 1
X = 0
a / 1
G = 0
N = 0
P = 0
S = 0
U = 0
Z = 0

Get with your EditorTrees team

Read the specification and check out the starting code

Milestone 1 due Tuesday