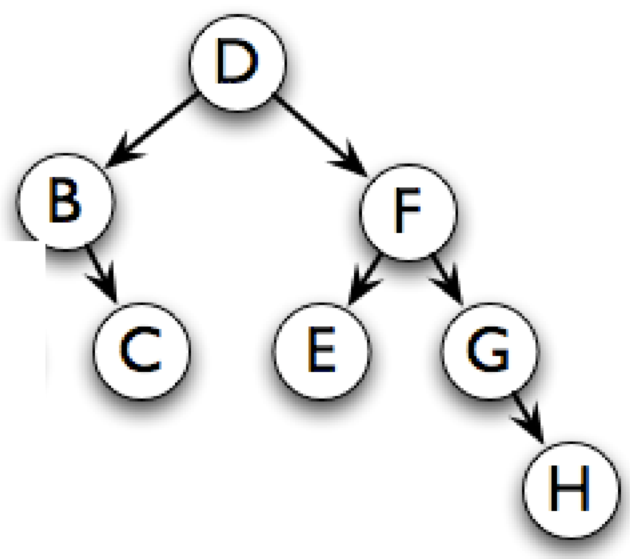
CSSE 230 Day 13

Height-Balanced Trees



Announcements

- Doublets Milestone 1 due next Tuesday night
- Exams redux now and Tuesday

Today's Agenda (a lot of it may spill over into Monday)

- Exam 1 review?
- Doublets: what's it all about?
- Finding k-th smallest in BST
- Meet your Doublets partner
- Another induction example
- Recap: The need for balanced trees
- Analysis of worst case for height-balanced (AVL) trees

Doublets: What's it all about?

Answers will vary from these!

Welcome to Doublets, a game of "verbal torture."
Enter starting word: ***flour***
Enter ending word: ***bread***
Enter chain manager (s: stack, q: queue, x: exit): ***s***
Chain: [flour, floor, flood, blood, bloom, gloom, groom, broom, brood, broad, bread]
Length: 11
Candidates: 16
Max size: 6
Enter starting word: ***wet***
Enter ending word: ***dry***
Enter chain manager (s: stack, q: queue, x: exit): ***q***
Chain: [wet, set, sat, say, day, dry]
Length: 6
Candidates: 82651
Max size: 847047
Enter starting word: ***whe***
Enter ending word: ***rye***
The word "oat" is not valid. Please try again.
Enter starting word: ***owner***
Enter ending word: ***bribe***
Enter chain manager (s: stack, q: queue, x: exit): ***s***
No doublet chain exists from owner to bribe.
Enter starting word: ***C***
Enter chain manager (s: stack, q: queue, x: exit): ***x***
Goodbye!

A **Link** is the collection of all words that can be reached from a given word in one step. I.e. all words that can be made from the given word by substituting a single letter.

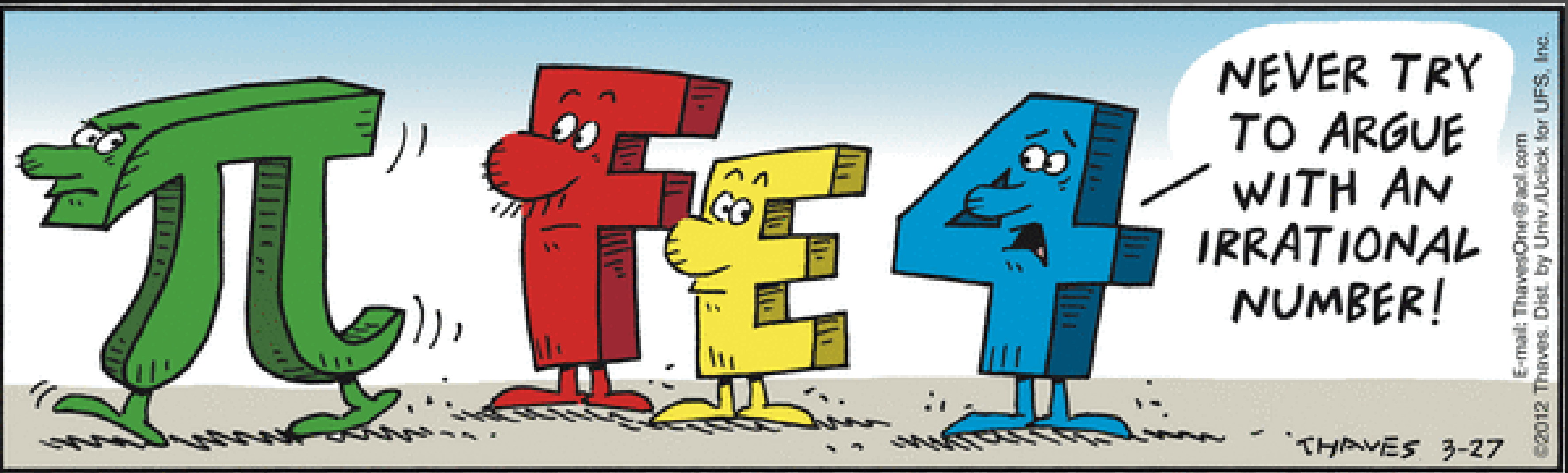
A **Chain** is a sequence of words (no duplicates) such that each word can be made from the one before it by a single letter substitution.

A **ChainManager** stores a collection of chains, and tries to extend one at a time, with a goal of extending to the ending word.

StackChainManager: depth-first search
QueueChainManager: breadth-first search
PriorityQueueChainManager: First extend the chain that ends with a word that is closest to the ending word.

BST with Rank

- Explore the concept
  How do Find and Insert work?



BSTs are an efficient way to represent ordered lists

- What's the performance of
  - insertion? O(h(T))
  - deletion? O(h(T))
  - find?   O(h(T))
  - iteration? O(n) to iterate through all

- What about finding the k^{th} smallest element?

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

Old Q6

- 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

Another induction example (we'll use this result) Q1

- Recall our definition of the Fibonacci numbers:
  - $F_0 = 0$, $F_1 = 1$, $F_{n+2} = F_{n+1} + F_n$
- An exercise from the textbook

7.8 Prove by induction the formula

$$F_N = \frac{1}{\sqrt{5}}\left(\left(\frac{(1+\sqrt{5})}{2}\right)^N - \left(\frac{1-\sqrt{5}}{2}\right)^N\right)$$

Recall: How to show that property P(n) is true for all $n \geq n_0$:

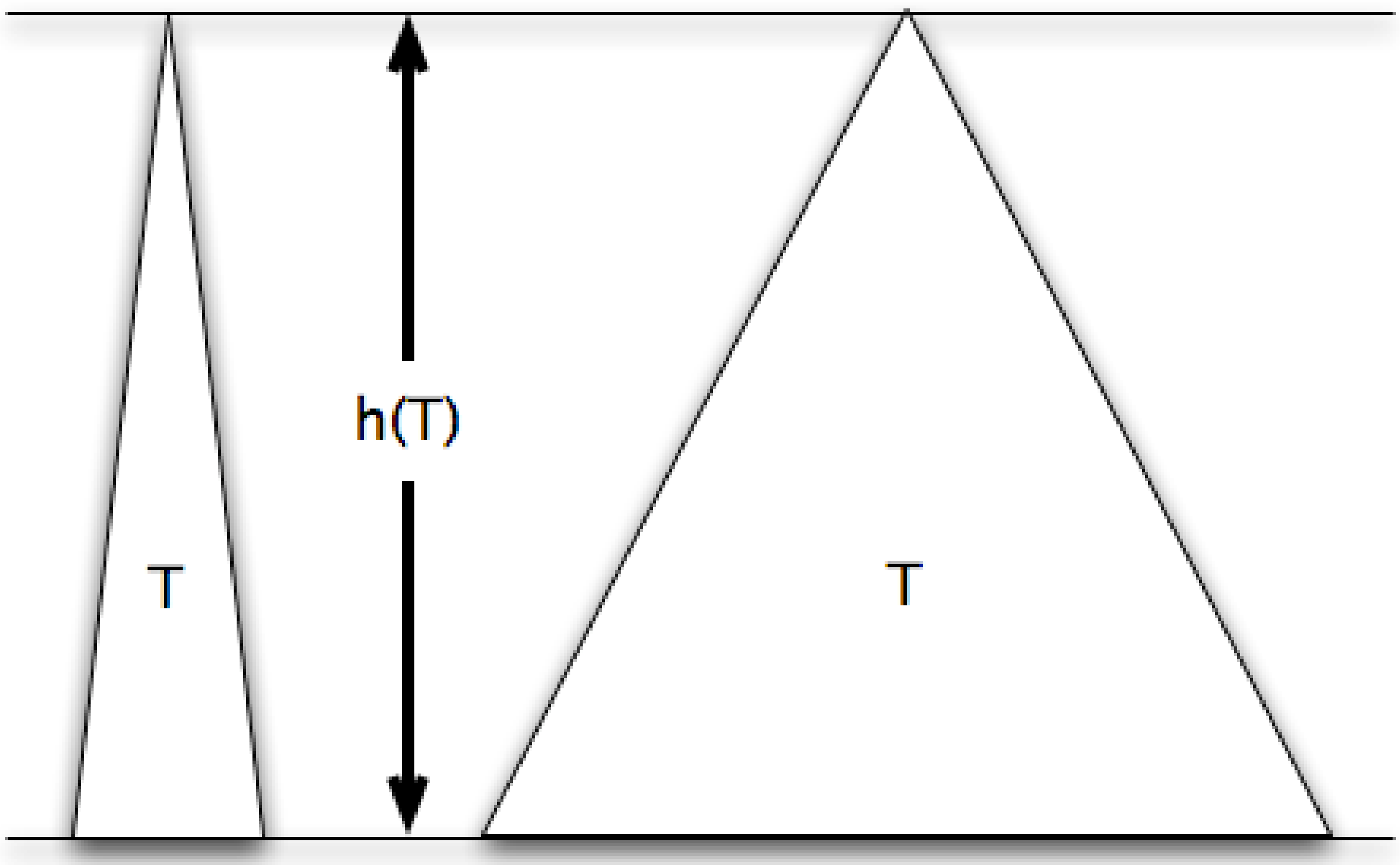
(1) Show the base case(s) directly

(2) Show that if P(j) is true for all j with $n_0 \leq j < k$, then P(k) is true also

Details of step 2:

a. Write down the induction assumption for this specific problem
b. Write down what you need to show
c. Show it, using the induction assumption

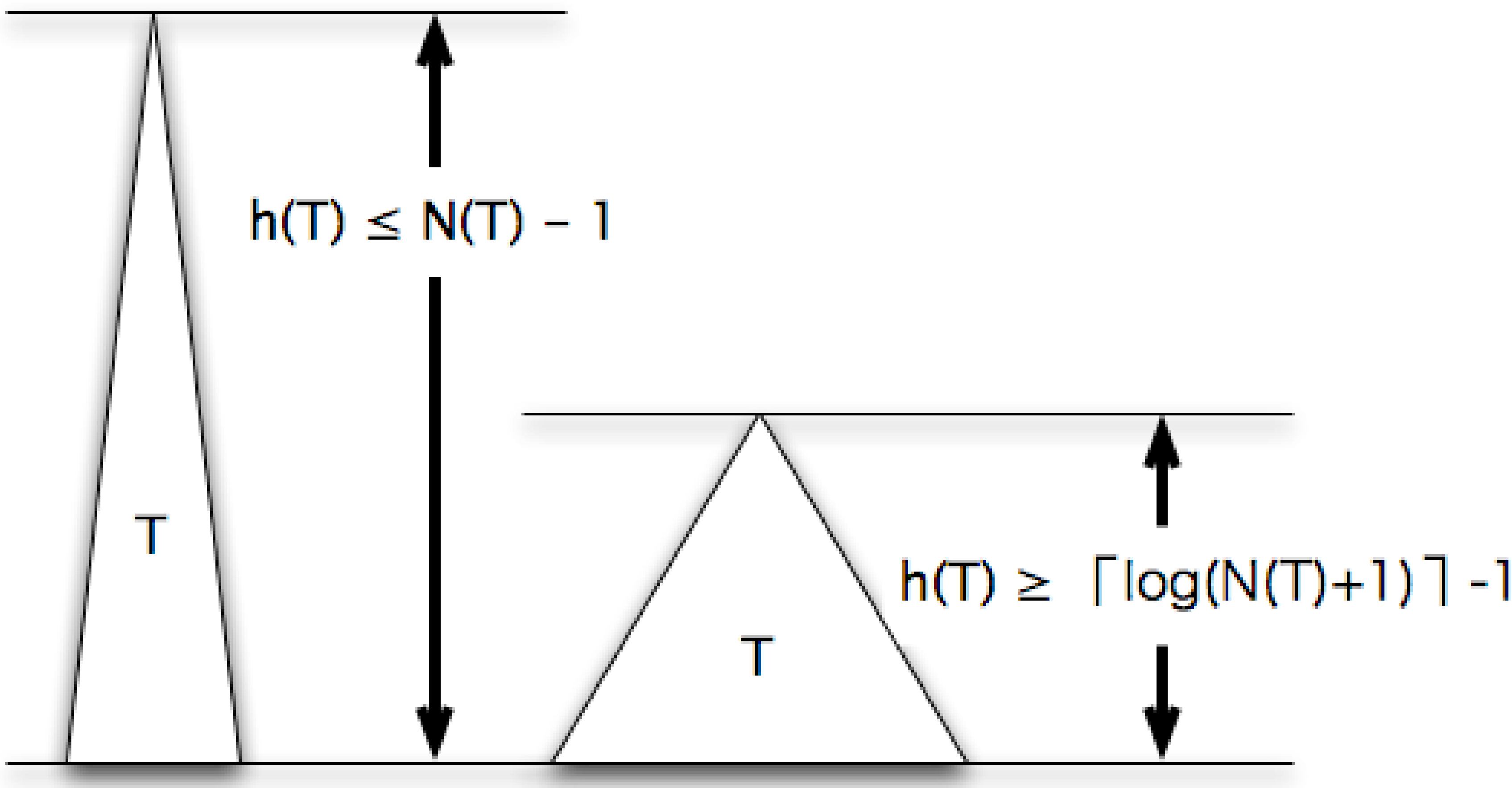
Review: The number of nodes in a tree with height h(T) is bounded



$N(T) \geq h(T) + 1$

$N(T) \leq 2^{h(T)+1} - 1$

Review: Therefore the height of a tree with N(T) nodes is also bounded



We want to keep trees balanced so that the run run time of BST algorithms is minimized

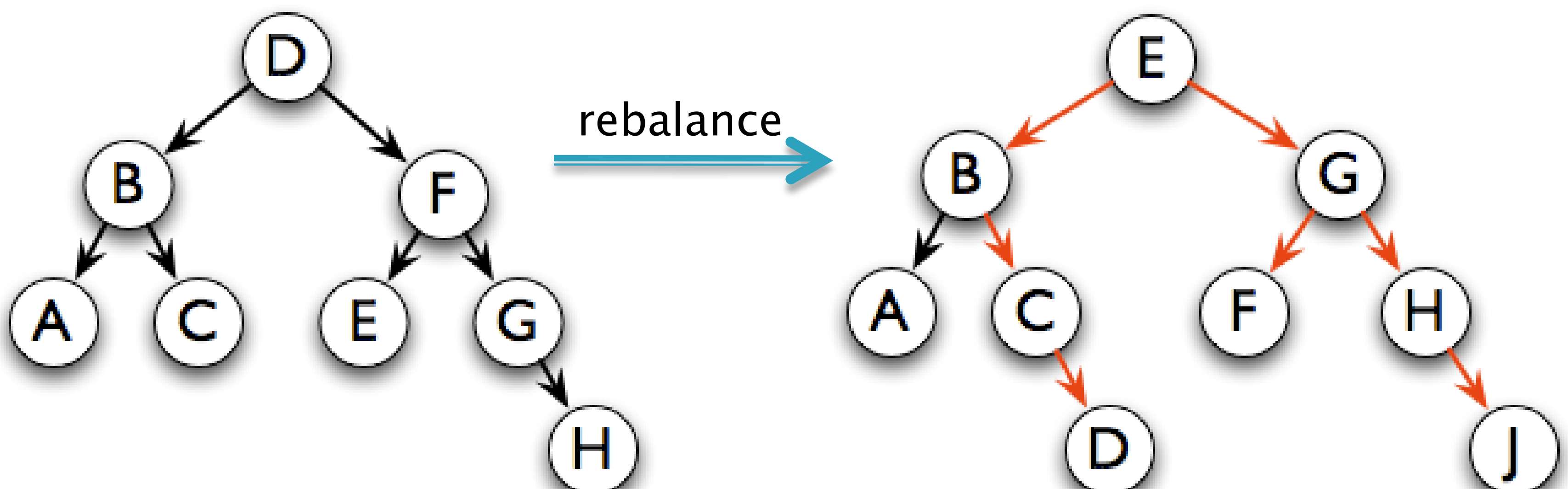
Q2

- BST algorithms are O(h(T))
- Minimum value of h(T) is $\lceil \log(N(T)+1) \rceil - 1$
- Can we rearrange the tree after an insertion to guarantee that h(T) is always minimized?

But keeping complete balance is too expensive!

Q3

- Height of the tree can vary from log N to N
- Where would J go in this tree?
- What if we keep the tree perfectly balanced?
  - so height is always proportional to log N
- What does it take to balance that tree?
- Keeping completely balanced is too expensive:
  - O(N) to rebalance after insertion or deletion

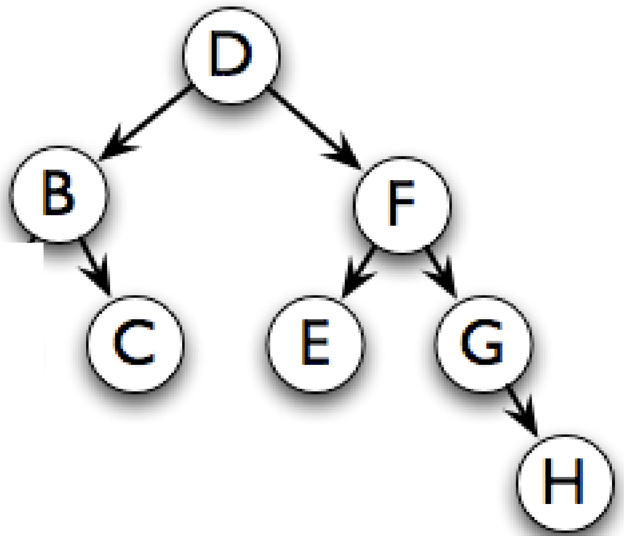


Solution: Height Balanced Trees (less is more)

Height-Balanced Trees have subtrees whose heights differ by at most 1

Q4

Still height-balanced?



More precisely , a binary tree **T** is height balanced if

T is empty, or if

| **height**(T_L) – **height**(T_R) | ≤ 1, and

T_L and T_R are both height balanced.

What is the tallest height-balanced tree with N nodes?

Q5

Is it taller than a completely balanced tree?

- Consider the dual concept: find the minimum number of nodes for height h.

A binary search tree **T** is height balanced if

T is empty, or if

| **height(** T_L **)** – **height(** T_R **)** | ≤ **1**, and

T_L and T_R are both height balanced.

An AVL tree is a height-balanced BST that maintains balance using "rotations"

- Named for authors of original paper, Adelson-Velskii and Landis (1962).
- Max. height of an AVL tree with **N** nodes is: **$H < 1.44 \log(N+2) - 1.328 = O(\log N)$**

Our goal is to rebalance an AVL tree after insert/delete in O(log n) time

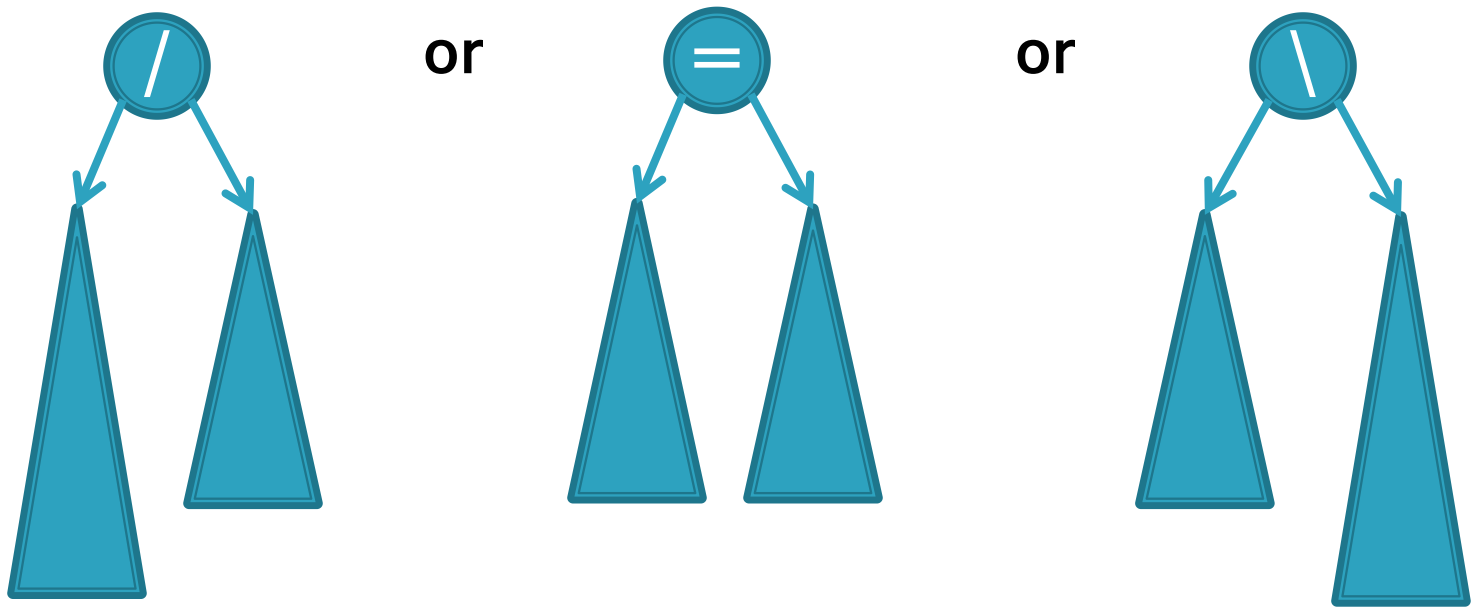
Q8

- Why?
- Worst cases for BST operations are **O(h(T))**
  - find, insert, and delete
- **h(T)** can vary from **O(log N)** to **O(N)**

- Height of a height-balanced tree is **O(log N)**

- So if we can rebalance after insert or delete in **O(log N)**, then **all** operations are **O(log N)**

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

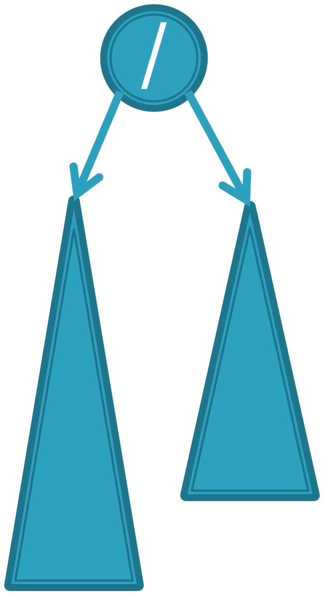


Different representations for / = \ :

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

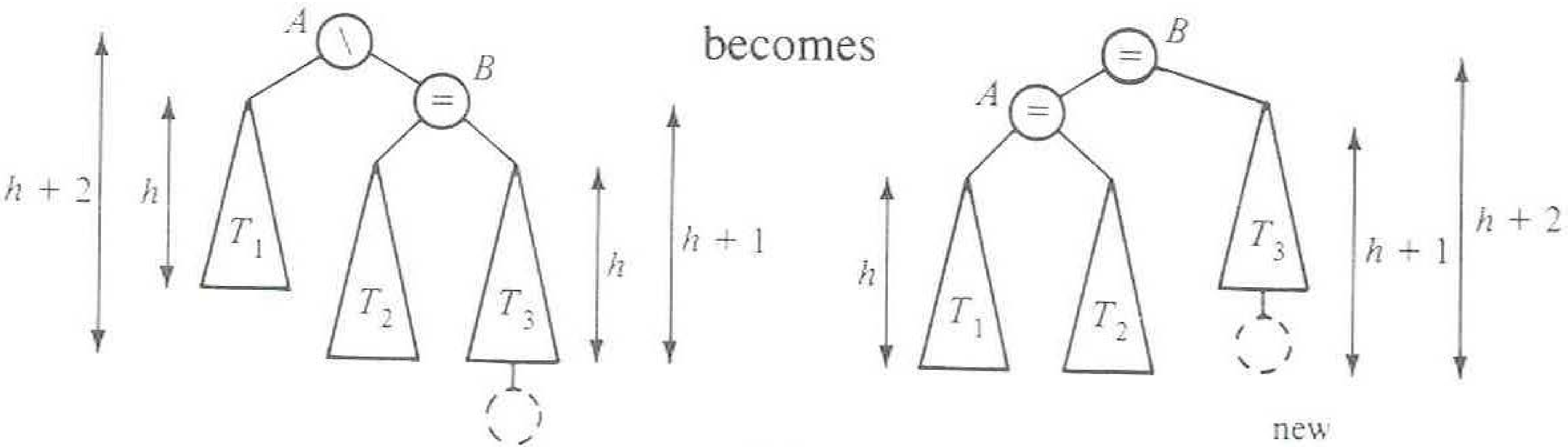
AVL Tree (Re)balancing Act

- 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 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'll pick up here next class…