CSSE 230 Day 13

Balanced Trees

Due this week

- Displayable due today, but "grace day until tomorrow 8 AM)
  - Lab assistants tonight in F217 (Doug 7-9, Brian 9-11)
- EditorTrees team preference survey due Wednesday at noon.
  - Teams of three.
  - I will try to avoid "performance mismatches", so survey asks for your overall course average.
  - Read item description on ANGEL for more details.
- WA5 due Thursday
  - Includes first "threaded" problem, so start early.
- Doublets Milestone 1 due Friday
  - Aim for earlier; Milestone 1 is considerably less than the halfway point of code for the project.

Today's Agenda

- Your questions (about anything)
- Doublets: what's it all about?
- Meet your Doublets partner
- Return exams and discuss a few of problems
- Another induction example
- The need for balanced trees
- Analysis of worst case for completely balanced trees
- (After the break) Analysis of worst case for height-balanced (AVL) trees
- AVL tree balance after insert.
- **This is a lot: Some of the AVL tree stuff may spill over into tomorrow**

Doublets: What's it all about?

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: ***oat***
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 form 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.

Doublets pairs, repositories: Section 1

csse230-201230-doublets-11,amesen,piliseal
csse230-201230-doublets-12,dingx,elswicwj,weirjm
csse230-201230-doublets-13,eubankct,sanderej
csse230-201230-doublets-14,goldthea,maglioms
csse230-201230-doublets-15,harbisjs,murphysw
csse230-201230-doublets-16,huangz,namdw
csse230-201230-doublets-17,jarvisnw,mcdonabj
csse230-201230-doublets-18,mccullwc,yuhasmj
csse230-201230-doublets-19,mehrinla,morrista
csse230-201230-doublets-20,millerns,koestedj
csse230-201230-doublets-21,newmansr,rudichza
csse230-201230-doublets-22,nuanests,shahdk
csse230-201230-doublets-23,paulbi,woolleld
csse230-201230-doublets-24,postcn,rujirasl
csse230-201230-doublets-25,semmeln,timaeudg

Meet your partner, exchange contact info, plan when you can meet again.

There will be in-class work time days 14 and 15.

Doublets pairs, repositories: Section 2

csse230-201230-doublets-26,bolivabd,memeriaj
csse230-201230-doublets-27,davelldf,iwemamj
csse230-201230-doublets-28,ewertbe,spryct
csse230-201230-doublets-29,faulknks,hopwoocp
csse230-201230-doublets-30,fendrirj,pohltm
csse230-201230-doublets-31,gartzkds,minardar
csse230-201230-doublets-32,haydr,lawrener
csse230-201230-doublets-33,modivr,qinz
csse230-201230-doublets-34,lius,weil
csse230-201230-doublets-35,mengx,stewarzt
csse230-201230-doublets-36,meyermc,yuhasem
csse230-201230-doublets-37,roetkefj,uphusar
csse230-201230-doublets-38,ruthat,tilleraj
csse230-201230-doublets-39,scroggd,watterlm
csse230-201230-doublets-40,taylorem,zhangz

Meet your partner, exchange contact info, plan when you can meet again.

There will be in-class work time days 14 and 15.

Exam question 2

2. (14 points) Give the big-theta worst-case running time for the most efficient algorithm for each problem

_n log n__ merge sort an array of n elements *Every* sort is Ω(n). Why?

___ n ____ sequential search of an array of n elements

___2^n___ solve towers of Hanoi for n disks

___ n __ insert a new node into a binary tree with n elements — Worst case is not a balanced tree

___ n ___ post-order traversal of a tree with n elements

___ n log n _ determine whether an array of n elements represents a set (i.e., has no duplicate elements) ↑ Merge sort (n log n), then look at adjacent elements (n)

__ n ___ find the maximum contiguous subsequence sum in an array of n numbers

We studied an O(n) algorithm in class, and it is in the textbook.

Exam questions 6

6. (8 points) Use the formal definition of O or Ω (the existence of constants n_0 and c) to prove *one* of the following statements. Both are true, but you are only asked to show one of them. No extra points for doing both.
   a. If $f(n) = n^2 - 7$ and $g(n) = n^2$, show that f(n) is Ω(g(n)).
   b. O is transitive. I.e. if f(n) is O(g(n)) and g(n) is O(h(n)), then f(n) is O(h(n)).

   Which part are you proving? (circle it) a b

   a. Example: $c = \frac{1}{2}$. Then we need $n^2 - 7 \geq \frac{1}{2} n^2$. This gives us $n^2 \geq 14$, which is true for $n \geq 4$. So $n_0=4$. (or any larger number).

   [c can be any number between 0 and 1, and n0 is calculated similarly for each]

   b. Since f(n) is O(g(n)) there are constants n_1 and c_1 such that $f(n) \leq c_1\, g(n)$ for all $n \geq n_1$. Similarly,
   Since g(n) is O(h(n)) there are constants n_2 and c_2 such that $g(n) \leq c_2\, h(n)$ for all $n \geq n_2$.

   Now let $n_0 = \max(n_1, n_2)$. If $n \geq n_0$, then $n \geq n_1$ and If $n \geq n_2$.

   Thus for all $n \geq n_0$, $f(n) \leq c_1\, g(n) \leq c_1 c_2\, h(n)$, so $c = c_1 c_2$ works.

Exam problem 7

```
public static boolean hasSpecial (List<Integer> c) {
    for(int i=0;  i<c.size();  i++ )
        for(int j = i+1; j< c. size();  j++)
            for(int k=0;  k<c.size(); k++)
                if(c.get(i) + c.get(j) == c.get(k))
                    return true;
    return false;
}
```

What is the worst-case big-theta running time when the list is an ArrayList?

The code that runs most often here is the test in the *if*. In an ArrayList, this test runs in constant time, so we get (in Maple notation)

sum(sum(sum(1, k = 0 .. n−1), j = i+1 .. n−1), i = 0 .. n−1);

the value is $\frac{1}{2} n^2(n-1)$, which is $\Theta(n^3)$.

b. (3) What is the worst-case running time when the list is a LinkedList?

The code that runs most often here is again the test in the *if*. In a linked list, this test runs in time proportional to i + j + k, so we get (in Maple notation)

sum(sum(sum(i + j + k, k = 0 .. n−1), j = i+1 .. n−1), i = 0 .. n−1);

the value is $\frac{3}{4} n^2(n^2 - 2n + 1)$, which is $\Theta(n^4)$.

c. (3) Suppose it takes 2 seconds (worst case) to run on a 1,000-item ArrayList. Approximately how long (worst case) will it take to run on a 3,000-item ArrayList?

Since the worst case growth rate is proportional to n^3, multiplying n by 3 multiples n^3 by 3^3, 2*27 = 54 seconds.

Programming : Use PQ to implement Queue

```
public class PQQueue<T> {

    private PriorityQueue<PQItem> pq;
    private static int sequence = 0;  // the priority of items in the PQ

    private class PQItem implements Comparable<PQItem> {
        T value;
        int sequenceNumber;

        public PQItem(T v, int s) {
            this.value = v;
            this.sequenceNumber = s;
        }

        @Override
        public int compareTo(PQItem other) {
            return this.sequenceNumber - other.sequenceNumber;
        }
    }

    public PQQueue() {
        this.pq = new PriorityQueue<PQItem>();
    }
    public void enqueue(T value) {
        this.pq.add(new PQItem(value, sequence++));
    }

    public T dequeue() throws NoSuchElementException {
        PQItem pqi = this.pq.poll();
        if (pqi == null)
            throw new NoSuchElementException("dequeue: empty queue");
        return pqi.value;
```

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

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

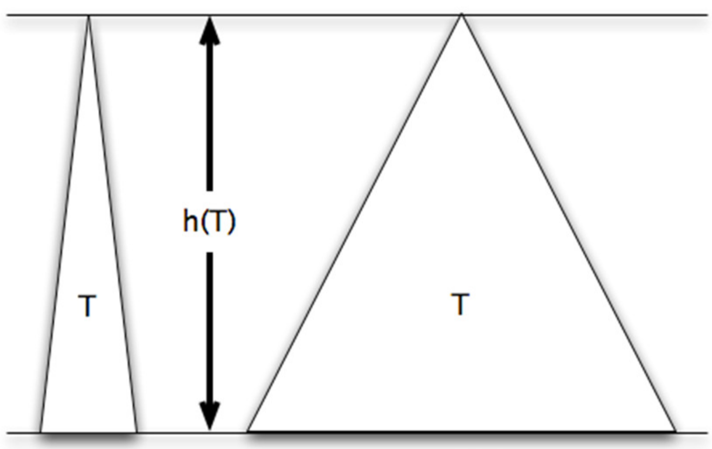
Recall: How to show that property P(n) is true for all $n \geq n_0$:

(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:

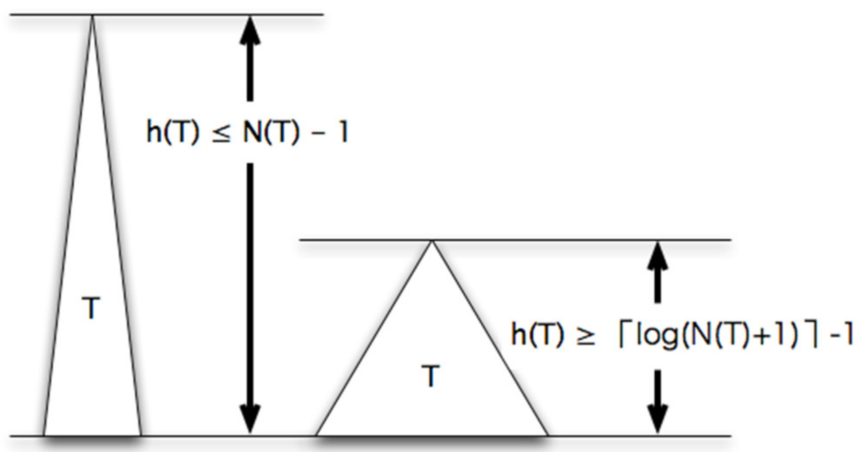
a. Write down the induction assumption for this specific problem
b. Write down what you need to show
c. Show it, using the induction assumption

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



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

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

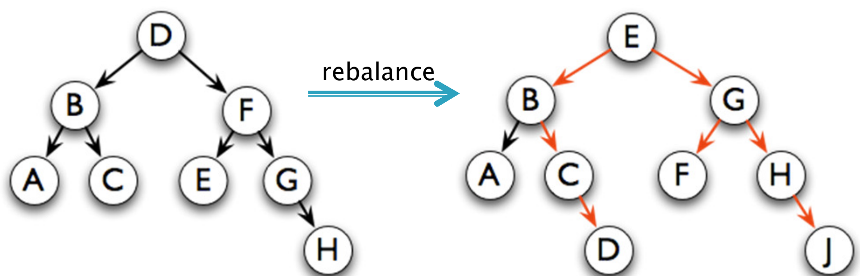
Q2

- 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

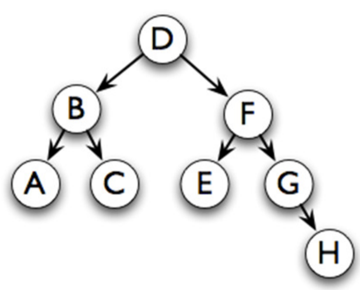
- Height of the tree can vary from log N to N
- Where would J go in this tree?
- What if we keep the tree perfectly balanced?
  - so height is always proportional to log N
- What does it take to balance that tree?
- Keeping completely balanced is too expensive:
  - O(N) to rebalance after insertion or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



Solution: Height Balanced Trees (less is more)

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

Q4



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.

Break

- And then exam discussion

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

Q 6-7

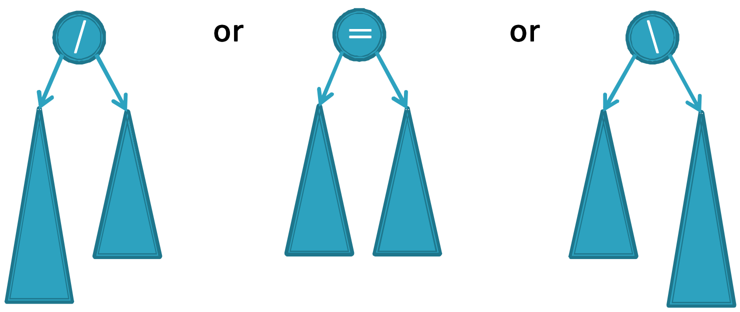
- Named for authors of original paper, Adelson-Velskii and Landis (1962).
- Max. height of an AVL tree with **N** nodes is: **H < 1.44 log (N+2) – 1.328 = O(log N)**

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"

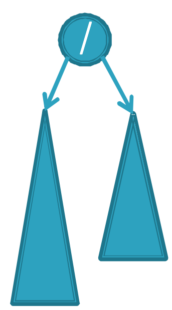


Different representations for / = \ :

- 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 this – how?
- Do appropriate rotation to balance the sub-tree rooted at this unbalanced node

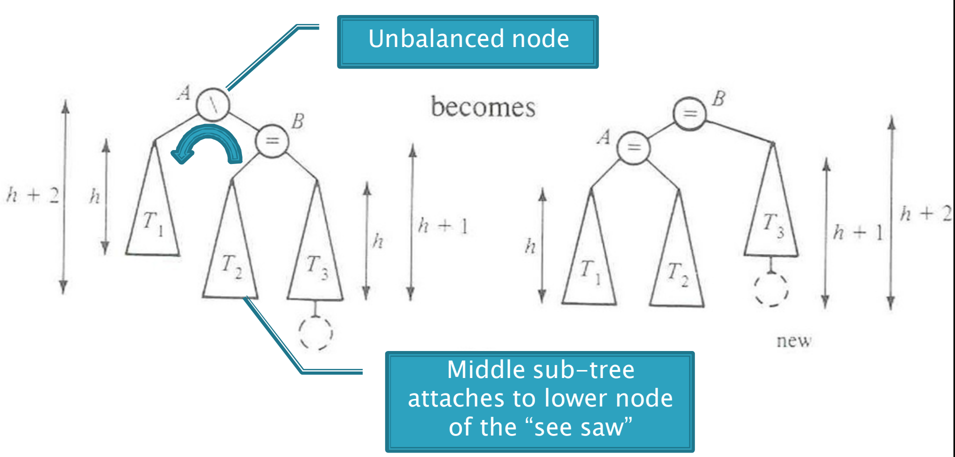


We rotate by pulling the "too tall" sub-tree up and pushing the "too short" sub-tree down

- 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

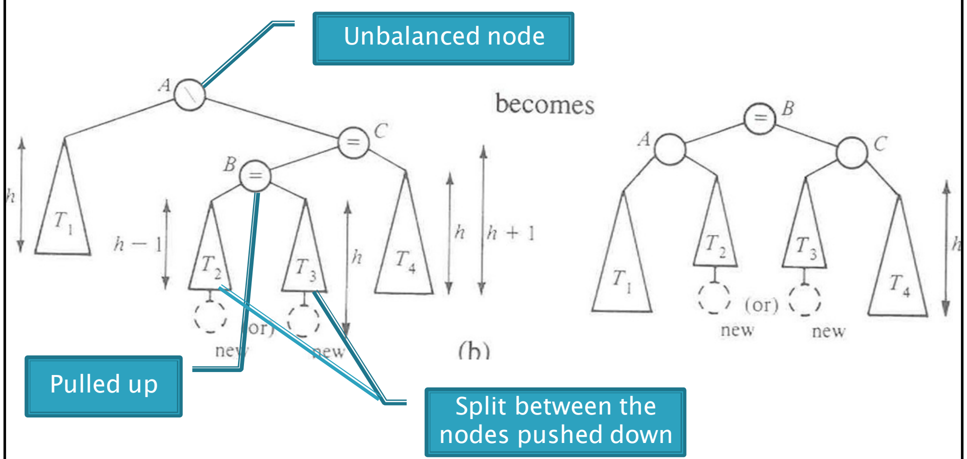
Q9-10



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

Double Left Rotation

Q11-12



Weiss calls this “right-left double rotation”

O(log N)?

Q 13-14

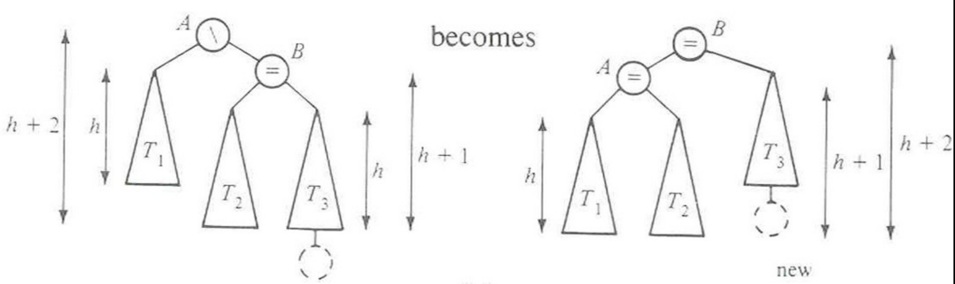
- Both kinds of rotation leave height the same as before the insertion!
- Is insertion plus rotation cost really O(log N)?

Which kind of rotation to do?

Depends on the first two links in the path from the node with the imbalance (A) down to the newly-inserted node.

First link (down from A)	**Second link** (down from A's child)	**Rotation type** (rotate "around A's position")
Left	Left	Single right
Left	Right	Double right
Right	Right	Single left
Right	Left	Double left

Your turn — work with a partner (if we don't run out of time) Q15-17



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

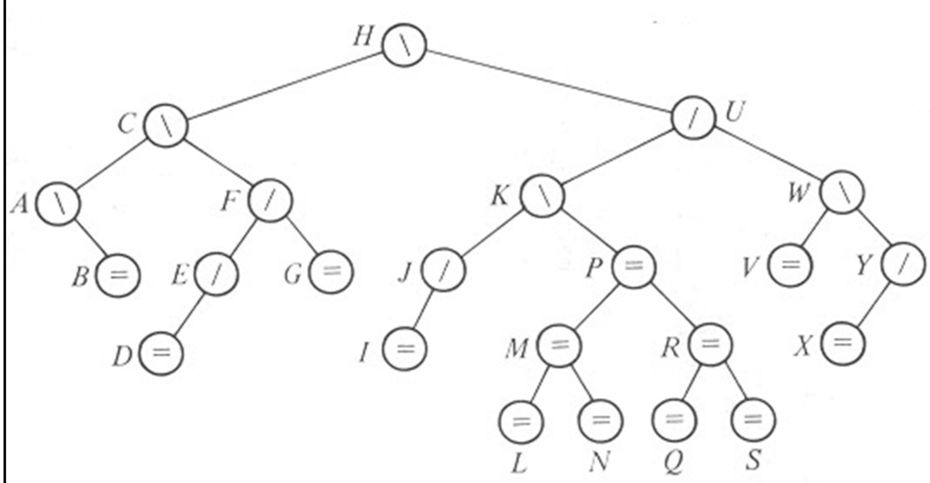
}
```
- Returns a reference to the new root of this subtree.
- Don't forget to set the balanceCode fields of the nodes.

Your turn — (after class?)

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


  }
  ```
- Returns a reference to the new root of this subtree.

A sample AVL tree



Insert **HA** into the tree, then **DA**, then **O**.
Delete **G** from the original tree, then **I**, **J**, **V**.
  ```

Your turn again (probably not until tomorrow)

- Start with an empty AVL tree.
- Add elements in the following order; do the appropriate rotations when needed.
  - 1 2 3 4 5 6 11 13 12 10 9 8 7
- How should we rebalance if each of the following sequences is deleted from the above tree?
  - (10 9 7 8) (13) (1 5)
  - For each of the three sequences, start with the original 13-element tree. E.g. when deleting 13, assume 10 9 8 7 are still in the tree.