DISCUSSION OF SOME QUESTIONS ON PRELIM 2

Lecture 23

CS2110 – Spring 2015

About prelim 2 Mean: 76.3. Median: 78

The few programming problems (recursion) were not done so well, will discuss in a minute.

Pick up prelim in handback room 216 Gates. Regrade request? Fill out form. Please wait until you see solutions, which we expect to post on the Piazza tonight.

Please do not email asking for your course grade based on what you have done so far. We can't answer that now. Too many other things to do. Prelims are important. Most people do very well on assignments. Prelims show mastery of material

A8 available today

Due date: Tuesday, 5 May (last day of class)

We may allow them until 7-8 May, with very small penalty. But we don't guarantee that yet.

As soon as possible after A8 deadline, we complete grading, figure out tentative course grade, and make it available.

You choose to take final or not. Taking it may lower as well as raise grade (does not happen often).

Final optional: Sunday 17 May

Recursion

It was heartbreaking to see so many people not getting recursion problems correct. These were relatively easy. Let's try one last time to get across how to write recursive methods.

To do these problems well, you have to:

- Read specification
- Deal with base cases
- Use the recursive definition of the data structure
- Keep things simple –don't complicate
- Draw diagrams, pictures.

Changing a pattern of thinking requires *consciously* applying good strategies, principles

Reversing a BST

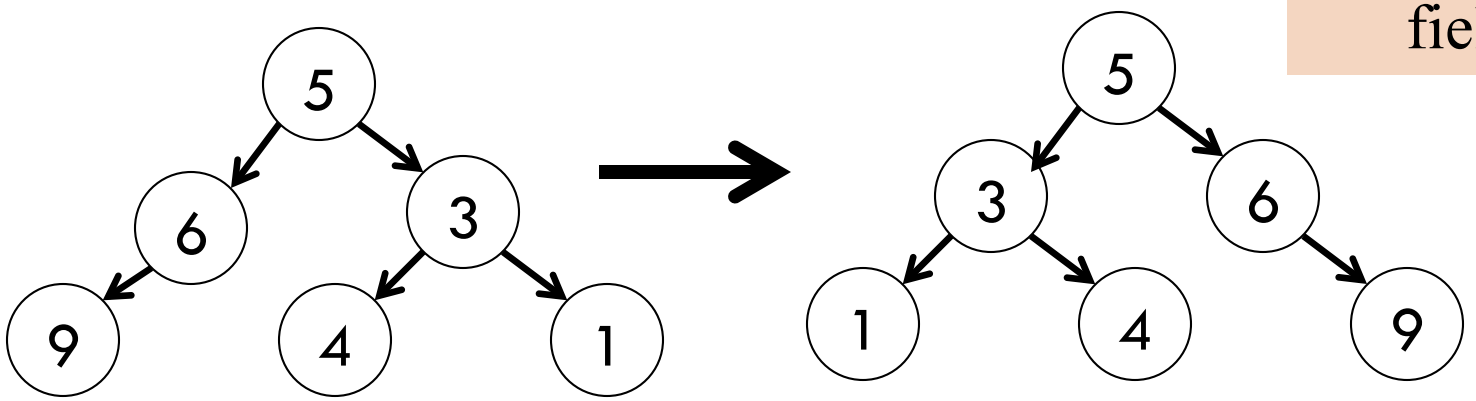
Consider trees whose nodes, of class Node, contain 3 fields:

value: the value at this node. Type is some class

left: left subtree (null if empty)

right: right subtree (null if empty)

Don't change field value



BST created using > instead of < for comparison, so the tree got put in kind of backward. E.g. inorder traversal of one BST should have been (1, 3, 4, 5, 6, 9) but was (9, 6, 5, 4, 3, 1).

Reverse a BST


```
/** Precondition: t is a binary search tree.
  * If t != null, change as many nodes of t as necessary
  * to reverse the inorder traversal of the tree.  */
public static void reverse(Node t) {
        if (t = null) return;
```

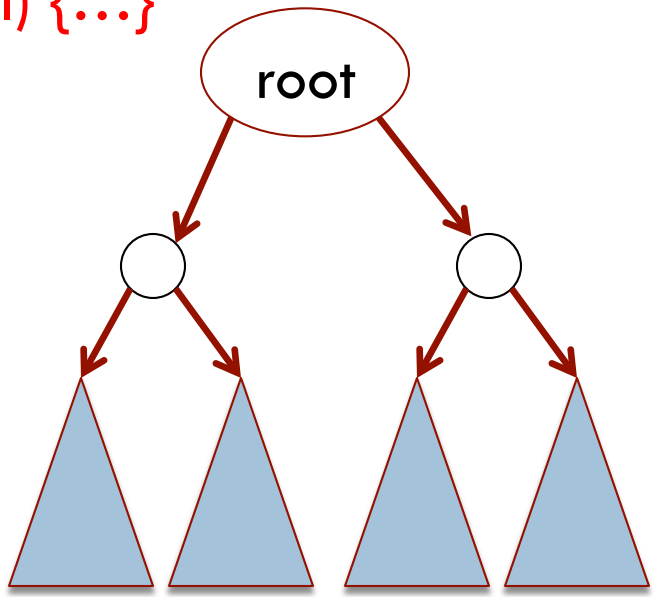
Estimate: over 25% people missed this base case. Why?

1. Did not read the specification.
2. Did not think about base case.

Reverse a BST


```
/** Precondition: t is a binary search tree.
  * If t != null, change as many nodes of t as necessary
   * to reverse the inorder traversal of the tree.  */
public static void reverse(Node t) {
        if (t = null) return;
        if (t.left == null  && t.right == null) {…}
        if (t.left.left …) {…}
```

Complicates matters.
Doesn't use the recursive definition of the data structure properly.



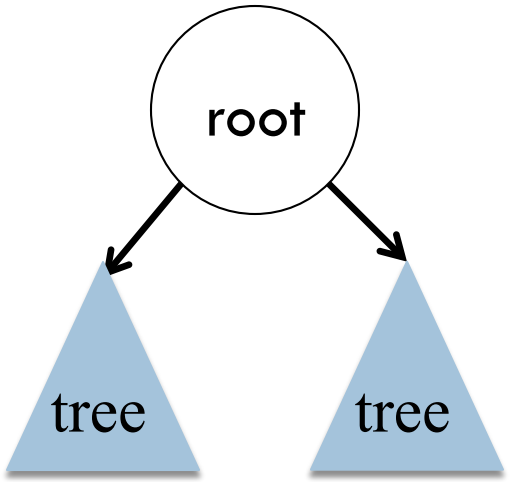
Reverse a BST

Writing a recursive tree method? Draw this tree!


```
/** Precondition: t is a binary search tree.
  * If t != null, change as many nodes of t as necessary
  * to reverse the inorder traversal of the tree.  */
public static void reverse(Node t) {
      if (t = null) return;
```

When viewing a tree as a recursive structure, like this, the code that is processing the root ---the whole tree--- should *not* look into the contents of the subtrees. They are just trees.

Tree is either null or

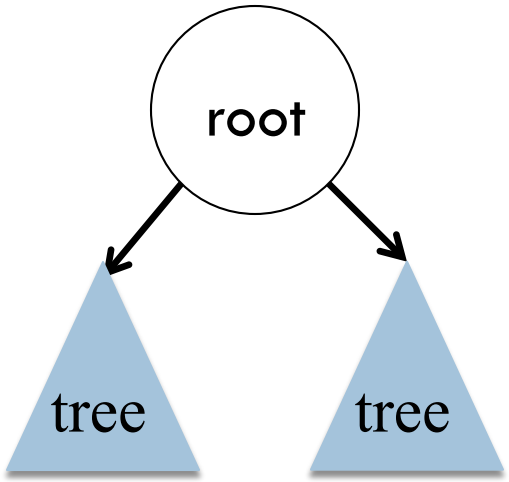


Reverse a BST


```
/** Precondition: t is a binary search tree.
   * If t != null, change as many nodes of t as necessary
   * to reverse the inorder traversal of the tree.  */
public static void reverse(Node t) {
        if (t = null) return;
        Swap t.left and t.right;
        reverse(t.left);
        reverse(t.right);
```

Writing a recursive tree method? Draw this tree!

Tree is either null or



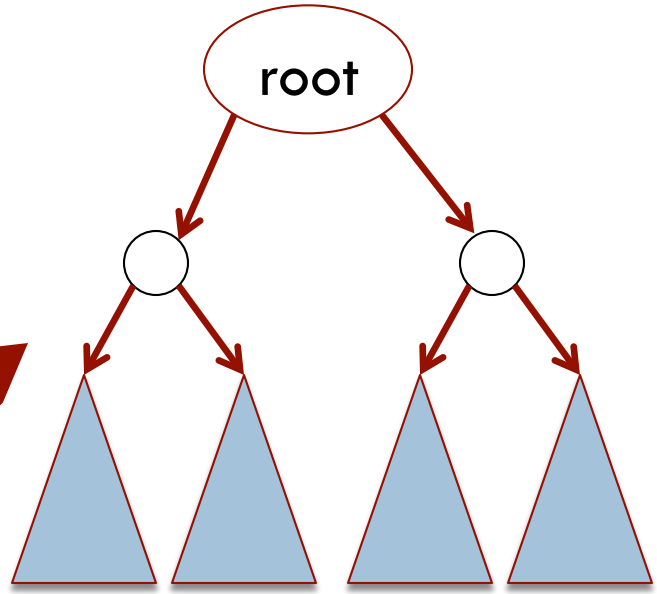
Tree equality


```
/** Return true iff tree t equals tree s –meaning they have the
    same shape and same values in their corresponding nodes. */
public static boolean equals(Node s, Node t) {
```

Complicated code that looks at
s.left.value, s.right.value, s.left.left, etc

Again, many people didn't handle the base case: At least one of s and t being null.

But then you are viewing the tree like this instead of in terms of the recursive definition of the tree.

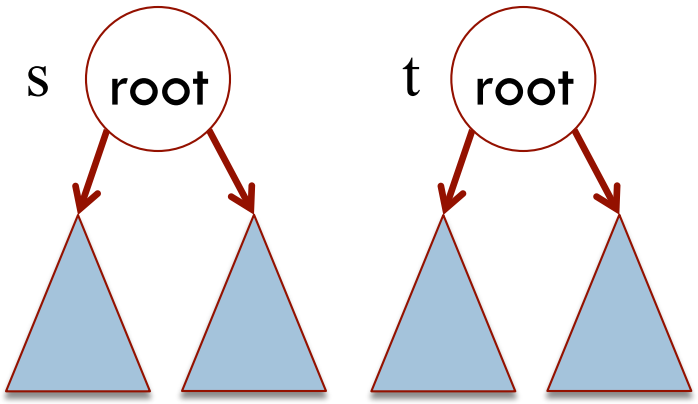


Tree equality


```
/** Return true iff tree t equals tree s –meaning they have the
    same shape and same values in their corresponding nodes. */
public static boolean equals(Node s, Node t) {
    if (s == null || t == null)
        return s == t;
    return s.value == t.value  &&
           equals(s.left, t.left) &&
           equals(s.right, t.right);
```

What is the base case?
At least one of s and t is null

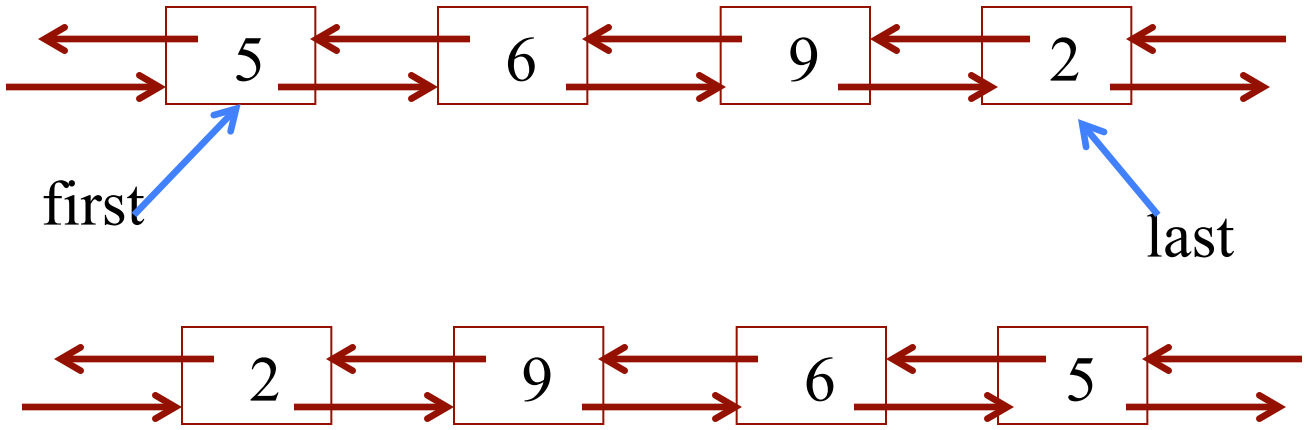
A tree is empty (null) or



Reverse values in a doubly linked list


```
/** Reverse the values in the nodes.
    * First and last point to first and last nodes to be reversed */
public static void rev(Node first, Node last)
```

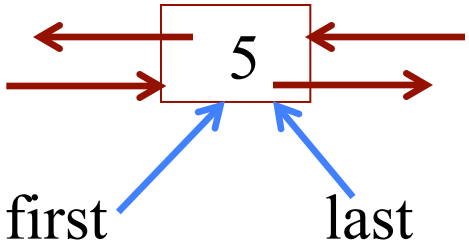
objects of class Node. Draw only what is necessary for understanding

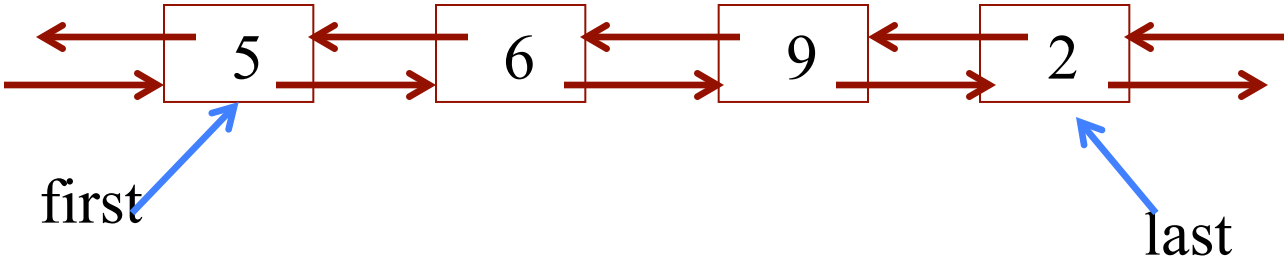


Reverse values in a doubly linked list


```
/** Reverse the values in the nodes.
    * First and last point to first and last nodes to be reversed */
public static void rev(Node first, Node last)
    if (first == last) return;
```

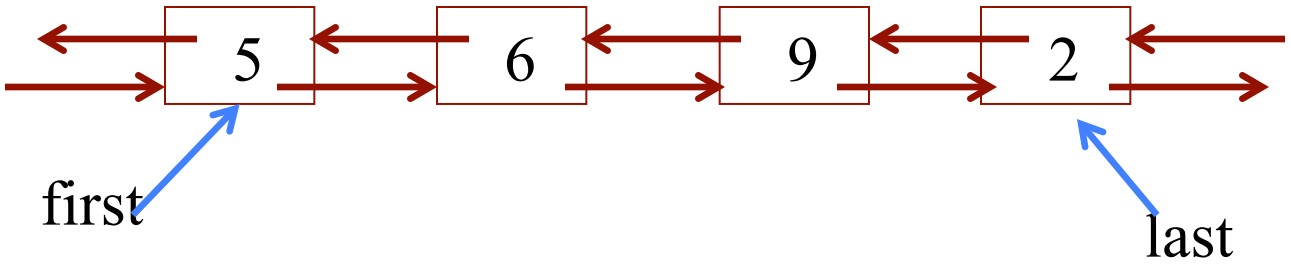
Base cases: (list is empty) and





Reverse values in a doubly linked list


```
/** Reverse the values in the nodes.
    * First and last point to first and last nodes to be reversed */
public static void rev(Node first, Node last)
    if (first == last) return;
    Swap first.value and last.value;
    if (first.next == last) return;
    rev(first.next, last.prev);
```



Be careful. If list has only two values, reversing is done

Recursion over a data structure

This is not a matter of learning facts. It is a matter of how you approach a new problem, how you think about it, develop a solution.

To do such problems well, you have to:

- Read specification
- Deal with base cases
- Use the recursive definition of the data structure
- Keep things simple –don't complicate
- Draw diagrams, pictures.

Changing a pattern of thinking requires *consciously* applying good strategies, principles