Overview

- Milestone #2
- Binary Trees

Milestone #2

- You'll need to extend the ordered container to support two new operations: `findNext` and `findPrev`.

```
public interface IOrderedContainer {

    /**
     * Returns the (key,value) with the next larger key
     * from that specified, regardless of whether the
     * specified key is itself in the container.  If
     * there isn't a (key,value) with a larger key,
     *  returns null.
     */
    public KeyValuePair findNext(IOrdered key);

    ...
```

Milestone #2

- If your implementation of `IOrderedContainer` uses a different name for the `KeyValuePair` class, keep that name.

- When are these new operations used? In the following steps...

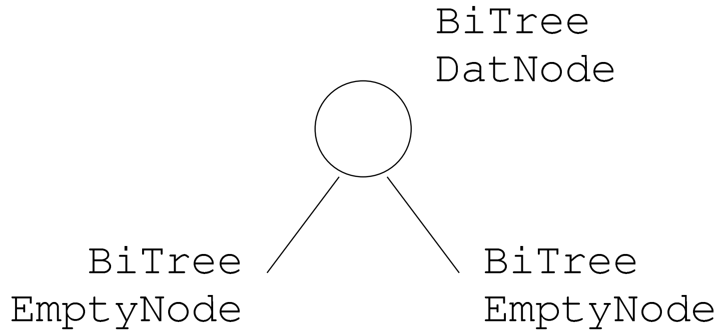
6. Insert their sum into the Ordered Container.
7. Compute the two new gaps for this number.

Milestone #2

- Coping with negative numbers
  - Use each number's absolute value as the key and maintain its (signed) value in the corresponding object.
  - Thus, if you discover that you're inserting a duplicate key into the ordered container, it's actually one of three cases:
    1. `-number` and `-number`
    2. `-number` and `number`
    3. `number` and `number`

Binary Trees

- Removing the root of an otherwise empty tree.



```
tree.remRoot();  -> _rootNode.remRoot(this);

                 -> _leftTree.remParent(_rightTree, parent);

                 -> _rootNode.remParent(sis, dad, this);

                 -> aunt.remOurParent(grandparent);

                 -> _rootNode.remOurParent(dad, this);

                 -> grandparent.setRootNode(EmptyNode.Singleton);
```

Binary Trees

- The following program creates and prints a simple binary tree.

```
import binaryTree.*;

class Test {
  public static void main(String args[])
  {
    BiTree tree = new BiTree();

    tree.insertRoot("I'm the root!");
    tree.setLeftSubTree(new BiTree());
    tree.getLeftSubTree().insertRoot("I'm the left child!");
    tree.setRightSubTree(new BiTree());
    tree.getRightSubTree().insertRoot("I'm the right child!");
    tree.execute(binaryTree.visitor.VerticalPrinter.Singleton,
                 null);
  }
}
```

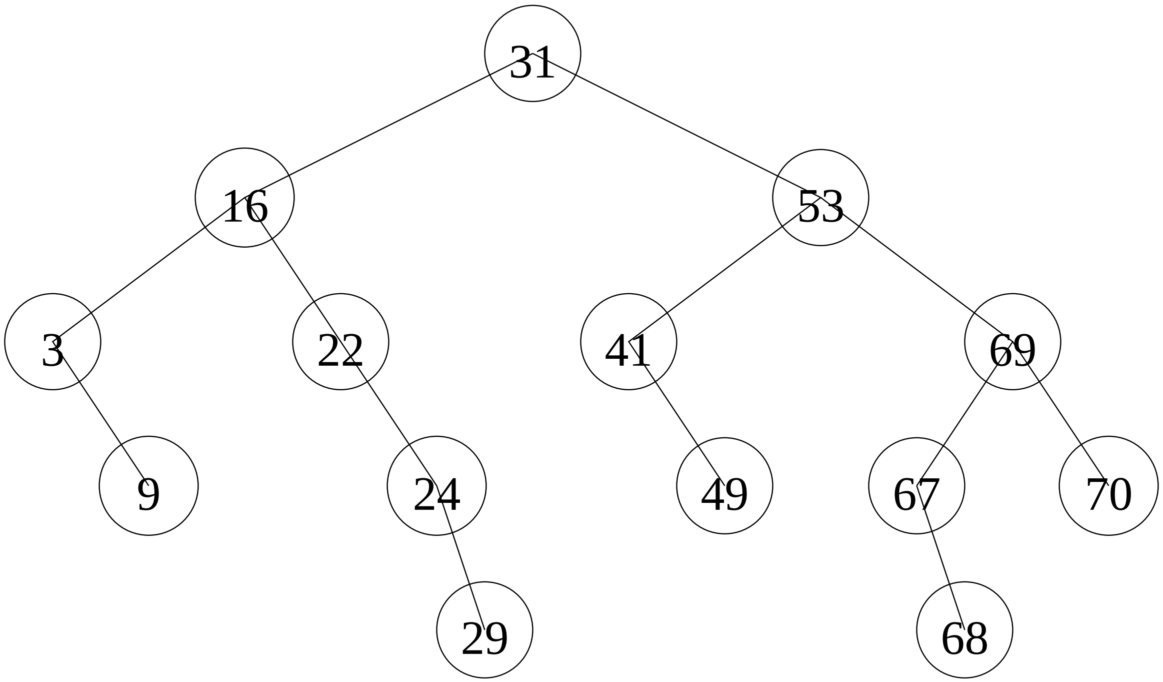
Binary Trees

- The printout looks like:

```
I’m the root!
 I’m the left child!
  []
  []
 I’m the right child!
  []
  []
```

Binary Search Trees

- In a binary search tree, each node's key is greater than its left child's key and less than its right child's key.


31
16
53
3
22
41
69
9
24
49
67
70
29
68