San Francisco State University
CSC 220: Data Structures
Dawson Zhou (zhoud@sfsu.edu)

Assignment 5: Binary Search Trees and Heaps

Overview

In this assignment, you'll apply what you've learned about binary search trees and heaps to answer various questions, some of which involve some code implementation.

Submission

Assignment 5 is due on **Friday, December 17, 2021** at **11:59 pm PT**. Late submissions will **not** be accepted for this assignment.

Submit the answers to the questions through the quiz link on iLearn.

Grading

100 pts total:

- Question 1: 25 pts
- Question 2: 25 pts
- Question 3: 10 pts
- Question 4: 20 pts
- Question 5: 20 pts

Detailed Requirements

Information for Question 1 and Question 2

Question 1 and Question 2 will use a modified binary search tree that can store multiple copies of any element. Every node in the tree will store the element itself, a count for how many "copies" of that element are stored, and the left and right child nodes. The **Node** class is defined as follows:

```java
public class Node {
    private String element;
    private int count;
    private Node left;
    private Node right;

    public Node(String element) {
        this.element = element;
        this.count = 1;
    }

    public String getElement() {
        return element;
    }

    public int getCount() {
        return count;
    }

    public void incrementCount() {
        count++;
    }

    public void decrementCount() {
        count--;
    }

    public Node getLeft() {
        return left;
    }

    public Node getRight() {
        return right;
    }

    public void setLeft(Node left) {
        this.left = left;
    }

    public void setRight(Node right) {
        this.right = right;
    }
}
```

This version of **Node** is very similar to the ones we've discussed in class, except that each node tracks how many occurrences of each element there are with the `count` variable. As you can see from the constructor, a new node is created with an initial count of one.

Question 1

Using the version of the binary search tree **Node** defined above, implement the **add** method:

```
public Node add(Node root, String element) {
    // TODO: Implement.
}
```

add should attempt to insert the new element into the binary search tree. If there is not already a **Node** containing `element`, then a new **Node** should be created and added to the tree. If there already is a **Node** containing `element`, then instead of creating a new **Node**, you should call **`incrementCount`** on the existing one.

Include the code in the body of the **add** method in your submitted answer on iLearn. As a hint, this implementation should look extremely similar to the one shown in class. The difference will be checking for nodes that already contain the newly inserted element.

Question 2

Using the version of the binary search tree **Node** defined above, implement the **`getCount`** method:

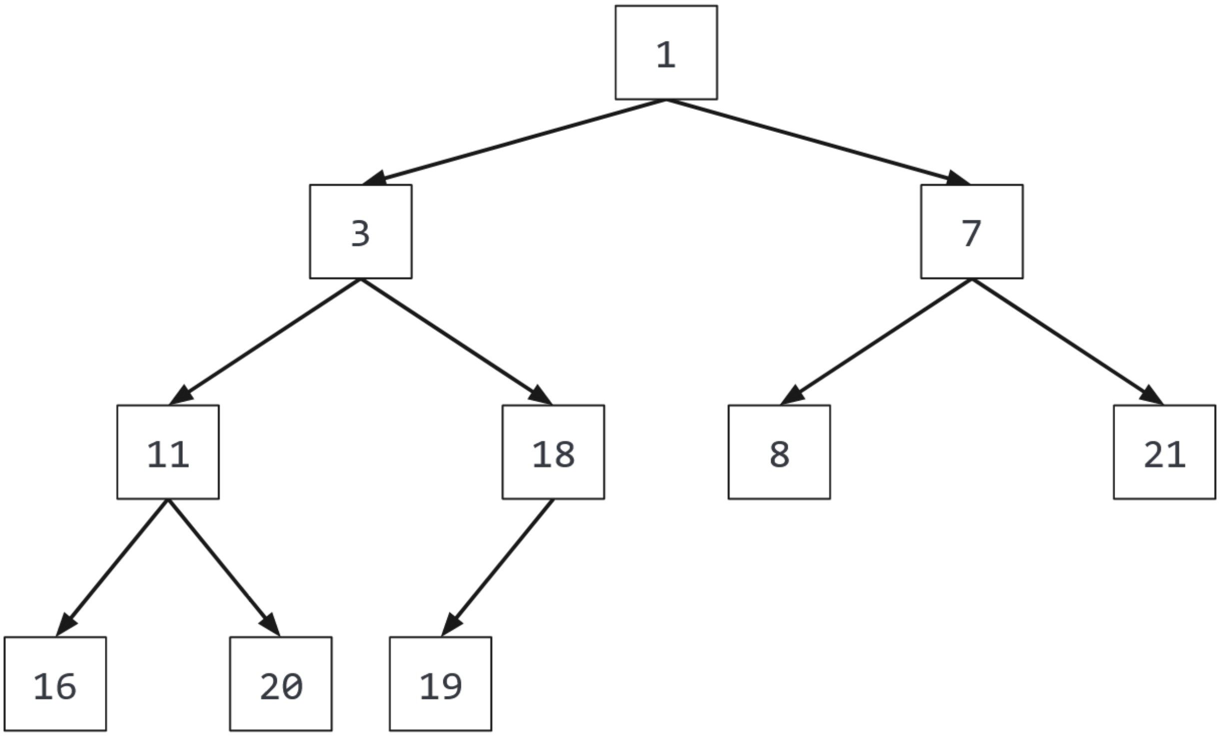
```
public int getCount(Node root, String element) {
    // TODO: Implement.
}
```

`getCount` should determine how many times `element` appears in the binary search tree. If there is a **Node** containing `element`, the stored count should be returned. If there is no **Node** containing element, `0` should be returned.

Include the code in the body of the **`getCount`** method in your submitted answer on iLearn. As a hint, this implementation should look extremely similar to the one shown in class. The difference will be checking the `count` variable stored in the **Node**.

Information for Question 3, Question 4, and Question 5

Question 3, Question 4, and Question 5 will refer to the following min heap:



Question 3

How would the min heap shown above be represented as an **`ArrayList`**? Please explain your reasoning. Assume that `-1` will be placed in the unused index `0` spot.

You can format your answer as follows: `[-1, a, b, c, d, e]` (replacing a, b, c, d, and e with actual values)

Question 4

Please explain in detail, step by step, how you would insert the value 2 into the min heap while keeping the structure as a valid min heap.

Question 5

Please explain in detail, step by step, the process of removing the smallest element from the min heap shown above (not including the 2 added in Question 4) while keeping the structure as a valid min heap.