Tree Practice Problems

Chen Ruichao <linuxer.sheep.0x@gmail.com>

1 More Amoeba

Remember the `Amoeba` class? Let's play with it a bit more. (Feel free to make a copy and do the practice in the copy, just in case you mess up your lab.)

1.1 Warming Up

1. Write a method `int numLeaf()` that counts the number of leaf nodes in an `AmoebaFamily`.
2. Write a method `int numInternalNodes()` that counts the number of internal nodes in an `AmoebaFamily`[1].

1.2 Two Descendants

Write a method `int twoDescendants()` that counts the number of amoeba(e) that have exactly two children[2].

1.3 Longest Path

1. Write a method `int longestAnchoredPathLength()` that finds the longest path in the tree that pass through the root node, and return the number of nodes on that path. Of course, your path cannot include any node more than once.
2. Challenge: Write a method `int longestPathLength()` that finds the longest path in the tree and return the number of nodes on that path. Of course, your path cannot include any node more than once. (Friendly reminder: the longest path may or may not pass through the root node.)

For those with burning curiosity:

1. Analyze the runtime of your `longestPathLength`. Could you give the runtime in Big O notation? Could you give the runtime in Big Θ notation?

[1] An internal node is a node that's not a leaf.
[2] Recall that a child is a node directly below another node.

2. Challenge[3]: There's a way to find the longest path in $O(N \log N)$ time, where N is the number of nodes in the tree. Can you figure it out?

2 Trees, just trees

```
// I hate trees without dummy nodes...
class BinaryTree {
    class Node {
        int v;
        Node l, r;  // children

        Node(int v, Node l, Node r) {
            this.v = v;
            this.l = l;
            this.r = r;
        }
        Node(int v) { this(v, null, null); }
    }

    Node root;

    BinaryTree() {
        root = null;
    }

    // ...
}

class Tree {
    class Node {
        int v;
        ArrayList<Node> ch; // children

        Node(int v) {
            this.v = v;
            this.ch = new ArrayList<Node>();
        }

        void addChild(Node n) {
            ch.add(n);
        }
    }

    Node root;
```

[3] Don't worry if you can't solve this. This is far beyond the scope of CS 61BL.

```
    Tree() {
        root = null;
    }

    // ...
}
```

- Write a method `boolean contains(int v)` for `Tree` that checks whether v appears in the tree at least once. Also write a version for `BinaryTree`.
- Write a method `boolean isBST()` for `BinaryTree` that checks whether the binary tree is a BST.
- Write a method `boolean hasDuplicate()` for `Tree` that checks whether the tree contains any value more than once. (Hint: you'll probably find a `HashSet` handy)
- Lots of tree algorithms only works on binary trees, so in algorithm research, we often want to convert an aribitrary tree to a binary tree. One commond approach is the left-children-right-sibling method, which creates a binary tree in which every node's left child is the node's first children in the original tree, and every node's right child is the node's next sibling in the original tree. For example, if we transform the tree

  ```
      2
    / | \
   5  3  4
  / \ |
  6 1 7
  ```

  to a left-children-right-sibling representation, we'll get

  ```
        2
       /
      5
    /   \
   6     3
    \   / \
     1 7   4
  ```

  Write a method `BinaryTree toBinary()` for `Tree` that converts the tree to a binary tree using the approach we described above.

3 Applications!

3.1 The Upset Manager

(Original problem: 郁闷的出纳员 from NOI 2004, China)

OIER ltd is a large software company with tens of thousands of employees. As a manager, my job is to keep track of the salaries of each employee. What really upsets me is that our boss is really random. He frequently adjusts salaries by increasing/decreasing the salary for each employee by a certain amount. For example, if two employees A and B had \$1000/month and \$1500/month correspondingly, and salaries increase by \$8000/month, they'll have \$9000 and \$9500 per month, correspondingly.

Our employees are really uncomfortable with the frequent changes, especially when everybody's salary decreases. When an employee's salary gets lower than the amount guaranteed by the contract, he/she will be very angry and leave the company immediately. By the way, every employee has the same minimum salary limit on their contract.

Whenever a new employee comes, I need to create an entry in our financial database; and when somebody leaves, I need to delete the corresponding entry. The boss often asks me about the salaries, um, in a weird manner. Instead of asking about the salary of a certain employee, he asks what the k-th highest salary entry is. Sorting out the entries by hand is really tedious, and I guess you already know why I hired you: I want you to write a program that keeps track of the salary information for me.

```
// You are required to use a BST.  (Can you see why it's faster?)

class SalaryDatabase {
    // your instance variables here
    //...

    /**
     * Creates an empty database (i.e.: there's no employee in the company yet).
     */
    SalaryDatabase(int minimumSalary) {
        /* your code here */
    }

    /**
     * Adds an entry to the database, with initialSalary as the initial value.
     */
    void addEmployee(int initialSalary) {
        /* your code here */
    }

    /**
```

```
     * Increases every entry by delta.
     */
    void increaseSalary(int delta) {
        /* your code here */
    }

    /**
     * Decreases every entry by delta.  Don't forget that some employees may
     * leave and you need to update the data to reflect this.
     */
    void decreaseSalary(int delta) {
        /* your code here */
    }

    /**
     * Returns the kth highest salary entry.
     */
    int kthHighestSalary(int k) {
        /* your code here */
    }
}
```