SE2852 Exam 1 Feedback

Dr. Yoder, Spring 2014, MSOE

Problem 1.a. Feedback, Page 2

① -1 Node missing from tree

② -1 Expression tree should have operators at non-leaf nodes. See Figure 2 for details.

③ -0.5 Root not clear (not at top of tree)

④ -1 Sub-trees should represent sub-expressions.

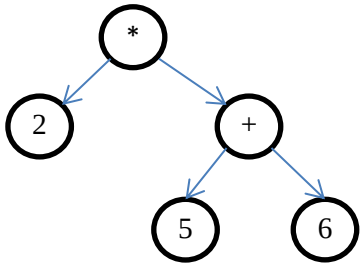

Figure 1: Tree answer for Problem 1a

Problem 2, Page 3

① -2 element does not remove – it throws an exception

② -1 element does not return -1 – it returns null. (Or return value for empty array not specified)

③ -0 peek and element don't copy the item, but only a reference to the item.

Problem 3, Page 3

① -2 Some mention of head, but doesn't describe why head is inefficient for an ArrayList

② -2 Some mention of Big-O, but no explanation of details

③ -10 It is not a problem that the index of first node keeps changing. Maintaining an index is just as efficient as maintaining a reference.

④ -2 The problem is not asking to ***use*** an ArrayList as a queue, but rather to ***implement*** a queue with an ArrayList as an instance variable.

⑤ -2 indexed lists ***are*** ordered lists.

⑥ -10 LinkedLists also grow in size – this is not a reason to avoid an array list or a queue.

⑦ -2 Unused methods do not take up any time. It's no problem to wrap a class and not use all of its methods. On the other hand, exposing the methods as part of the interface ***could*** cause trouble if someone used them when they shouldn't.

⑧ -2 Some mention that an ArrayList is more like a stack than a queue, but no mention why.

⑨ -5 Some mention of why LinkedList works, but the reason ArrayList doesn't work is totally wrong.

⑩ -5 Some mention that ArrayLists can grow – which is true, but not at all a problem.

Problem 4, Page 3

① -2 Circular queue uses an array, not a LinkedList

② -1 First element is not at start of queue, but rather where the first spot at the rear of the line was when it was added.

③ -2 Front element points to wrong element.

-1 but front elements clearly removed.

④ -1 array is wrong size and/or elements not put in at start of array.

⑤ -2 Queue treated like stack

⑥ -2 Not clear whether an array or LinkedList is used.

Problem 5.a., Page 4

If you made two -1.5 pt errors, there is a 0.5 discount for the second (so -2.5 total).

① -0.5 For full credit, implement in-order traversal.

② -1 State which sort of traversal you wrote. Is it pre- in- or post-order traversal?

③ -0.5 Use a variable like node.value rather than just node (or parent.value) when printing. This illustrates that you know where the node stores its data.

④ -0.5 Check for node == null explicitly.

⑤ -0.5 Minor errors such as syntax (missing {} on method) or missing a void on the parameter. Non-minor errors such as !(null==current), which won't compile in Java, but is a run-time exception in C/C++.

⑥ -1 Print left before right in all traversals – pre- in- or post-.

⑦ -1.5 If the Node given is null, simply do nothing. That's all you need to do to print an empty sub-tree (or not do, if that's how you prefer to look at it…)

⑧ -1.5 Don't attempt to process output. If you call yourself recursively, with a return type of "void", there is no result produced. In this case, we don't need a return value, since the recursive calls will print the rest of the tree.

⑨ -1.5 Follow references correctly. Need to print current, not parent node's value. Similarly, need to use `node.left` or `node.right`, and not just `node` since "`this`" is not pointing to "`node`."

⑩ -0 Using sout instead of System.out.println(…). **Yes, its OK**!

⑪ -0 **Missing the base call method.** It is important to be able to write this, but I did not explicitly request it.

⑫ -1.5 Implementing as

```
if(node.left != null) {
  postOrderTraversal(node.left)
  System.out.println(node.value);
}
if(node.right != null) {
  postOrderTraversal(node.right)
  System.out.println(node.value);
}
```

for Figure 2 will print out * + + *, a mixture of in- and post-order traversals without leaves.

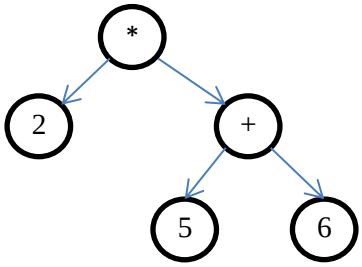

Figure 2: Tree answer for Problem 1a

Problem 5.a., Page 4

① -1 Simplify the O(n) expression. e.g. instead of O(2), write O(1). Instead of $O(n^2 + n)$, write O(n).

② If you leave out the recursive calls, it is O(1) because all that are left are a couple of conditions

-0.5 Correctly stating that the overall result is O(n)

Problem 6, Page 5

① -2 Complete and full (should not be full)

② -3 Full, but not complete (backwards!), and illustrates full clearly

③ -2 Not full, but not complete either.

④ -1 Diagram unclear – nodes or edge missing, some edges without nodes, others with…

Problem 7, Page 5

The key is to be able to distinguish between a null element and an empty queue.

Generally speaking, if there is some truth, but some incorrect statements (such as the -5 statements below), the incorrect statements are worth -1.

① -5 You don't want to use it just to get a stack trace. If you need a stack trace, you can use `new Throwable().getStackTrace();` at any time without needing to handle the exception. And you can detect that the queue is empty just by using the null.

② -2 A stack trace does not give you any more information about the location of null values than you could get by just checking if the output is null. But using a method that distinguishes between the end of the array and a null element does.

③ -5 Having a null or -1 value (see ⑧) returned will not cause problems later in the code – you can simply detect it with an if statement, and handle it faster than an exception.

④ -5 If the element is not null, then things are working well – we don't want to throw exceptions!

⑤ -2 Handling adding a "null" element correctly may be important, but there is no particular reason why ***adding*** a null element should fail, inherently. Removing a null element is where the problem comes in. (Granted, the non-exception methods actually ***do*** throw exceptions if you attempt to add a null element.)

⑥ -5 There are applications where it is legitimate to have null elements in a queue. For example, you may want to ignore one spot in the queue or handle it in a special way, perhaps in some sort of simulation where each spot in the queue corresponds to a time, and a null indicates that no events should be simulated at that time-step.

⑦ -5 The question is asking about why you would use the methods, not why you would want nulls in the queue.

⑧ -1 The methods do not return -1 – they return `null` if no element is provided.