Test 2: CompSci 100

Owen Astrachan

November 11, 2009

Name: ____________________________________________

NetID/Login: ________________

Honor code acknowledgment (signature) ____________________________________________

<table>
<thead>
<tr>
<th>value</th>
<th>grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>Problem 1</td>
<td>14 pts.</td>
</tr>
<tr>
<td>Problem 2</td>
<td>16 pts.</td>
</tr>
<tr>
<td>Problem 3</td>
<td>10 pts.</td>
</tr>
<tr>
<td>Problem 4</td>
<td>25 pts.</td>
</tr>
<tr>
<td>TOTAL:</td>
<td>65 pts.</td>
</tr>
</tbody>
</table>

This test has 9 pages, be sure your test has them all. Do NOT spend too much time on one question — remember that this class lasts 75 minutes. The last page is blank, you can use it if you need more space.

In writing code you do not need to worry about specifying the proper import statements. Don’t worry about getting method names exactly right. Assume that all libraries and packages we’ve discussed are imported in any code you write.

Unless indicated otherwise, the TreeNode class for this test is on the left. Some common recurrences and their solutions are on the right.

    public class TreeNode {
        String info;
        TreeNode left;
        TreeNode right;
        TreeNode(String val, TreeNode lptr, TreeNode rptr) {
            info = val;
            left = lptr;
            right = rptr;
        }
    }

<table>
<thead>
<tr>
<th>label</th>
<th>recurrence</th>
<th>solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>$T(n) = T(n/2) + O(1)$</td>
<td>$O(\log n)$</td>
</tr>
<tr>
<td>B</td>
<td>$T(n) = T(n/2) + O(n)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>C</td>
<td>$T(n) = 2T(n/2) + O(1)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>D</td>
<td>$T(n) = 2T(n/2) + O(n)$</td>
<td>$O(n \log n)$</td>
</tr>
<tr>
<td>E</td>
<td>$T(n) = T(n-1) + O(1)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>F</td>
<td>$T(n) = T(n-1) + O(n)$</td>
<td>$O(n^2)$</td>
</tr>
<tr>
<td>G</td>
<td>$T(n) = 2T(n-1) + O(1)$</td>
<td>$O(2^n)$</td>
</tr>
</tbody>
</table>
PROBLEM 1:  (Analysis (14 points))

Part A (4 points)
The public method \texttt{largest} below returns the largest value in an array by calling a private, helper method as shown. Write a recurrence for the helper method in terms of \(N\), the number of elements in the array between \texttt{first} and \texttt{last} inclusive. Then use that recurrence to give the big-O complexity of the helper method.

\begin{verbatim}
public int largest(int[] a){
    return largest(a,0,a.length-1);
}

private int largest(int[] a, int first, int last){
    if (first > last) return Integer.MIN_VALUE;
    if (first == last) return a[first];
    int mid = (first+last)/2;
    int before = largest(a,first,mid);
    int after = largest(a,mid+1,last);
    return Math.max(before,after);
}
\end{verbatim}

Part B (10 points)
In class we discussed the methods \texttt{height} and \texttt{isBalanced} below. Method \texttt{isBalanced} returns true if its tree parameter is height-balanced. Write recurrences for each method and the big-O complexity for each method. Express the recurrences in terms of \(N\), the number of nodes in the tree that is passed to the methods. Assume that trees are roughly balanced in writing the recurrences. \textbf{You should write two recurrences and two big-O expressions, one for each method.}

\begin{verbatim}
public int height(TreeNode root) {
    if (root == null) return 0;
    return 1 + Math.max(height(root.left), height(root.right) );
}

public boolean isBalanced(TreeNode root){
    if (root == null) return true;
    return isBalanced(root.left) &&
    isBalanced(root.right) &&
    Math.abs(height(root.left) - height(root.right)) <= 1;
}
\end{verbatim}
The following strings of parentheses/braces/brackets are balanced in that each left-symbol is matched by the corresponding right-symbol.

" ( ) ( ) ( ) "
" ( [ { [ ] } ] ) "
" [ [ ( ) } { { ) ] ] ]" 

The following strings are not balanced for the reason given on each line.

" ( ( ] " the right-] doesn't match the left-(
" [ ( ( ) ) " the left-[ isn't matched by a corresponding right-
" [ { } ] ) ] " the right-)'s do not match any symbol

The method `isBalanced` below returns true for all strings shown as balanced above as well as for every properly-balanced string. The method stores a number corresponding to each left-symbol on a stack and matches the top element when a right-symbol is found as the string of symbols is processed one character at-a-time. The method returns false for the first un-balanced string above, but returns true for the second one and throws an exception for the third one. Questions appear after the code.

```java
public class Balancer {

    private static String LEFTS = "{{";
    private static String RIGHTS = "}}";

    public boolean isBalanced(String s){
        Stack<Integer> st = new Stack<Integer>();
        for(int k=0; k < s.length(); k++){
            char ch = s.charAt(k);
            if (Character.isWhitespace(ch)) continue;

            int leftIndex = LEFTS.indexOf(ch);
            if (leftIndex >= 0){
                st.push(leftIndex);
            } else {
                // add code for C

                int rightIndex = RIGHTS.indexOf(ch);
                if (st.peek() != rightIndex) return false;
                st.pop();
            }
        }
        return true; // replace for B
    }
}
```
Part A (4 points)
Explain how the strings LEFT and RIGHT combined with the use of the method `indexOf` and a stack result in all balanced strings being recognized correctly by `isBalanced`. In other words, describe how the method works at a high-level. Extra credit for appropriate use of the word `map` in your explanation.

Part B (4 points)
The method `isBalanced` returns true for the string "[ ( ( ) )" although the string isn’t balanced because the first bracket isn’t matched. Replace the last `return` statement so that correctly-balanced strings are still identified, but the unbalanced string "[ ( ( ) )" is identified as unbalanced as are other strings with at least one unmatched symbol.

Part C (4 points)
The string " [ { } ] ) ) " is not correctly identified as unbalanced because the code throws an empty-stack exception. Insert a statement in the place identified in the code so that this string is properly identified as unbalanced and no exception is thrown.

Part D (4 points)
Show in code or describe in words how to add the symbols < and > so that the strings below are correctly identified as balanced, e.g., so that in addition to parentheses, braces, and brackets the less-than and greater-than symbols can be used in left-right symbol expressions.

"( [ < > ] ) "
" < < < > > > ( ) { }"
PROBLEM 3:  (Splice and Dice (10 points))

In this problem doubly-linked lists are implemented using the class DNode below. A doubly-linked list with three nodes is shown, each node points to both the previous and the next node in the doubly-linked list, though the first node’s prev field is null as shown.

```java
public class DNode {
    String info;
    DNode next;
    DNode prev;
    DNode(String s, DNode before, DNode after){
        info = s;
        next = after;
        prev = before;
    }
}
```

```java
public void print(DNode list){
    while (list != null) {
        System.out.println(list.info);
        list = list.next;
    }
}
```

```
"cherry"  "fig"  "mango"
```

**Part A (2 points)**

The method addFront adds a new node to the front of a doubly-linked list.

```java
public DNode addFront(DNode list, String s){
    list = new DNode(s,null,list);
    return list;
}
```

For example, the call `first = addFront(first, "apple")`; will add a node to the front of list diagrammed above so that the list has four nodes and will be printed as (apple, cherry, fig, mango) using the `print` method above. However, the new first node is not linked back to by the node after it, e.g., the cherry node’s prev field does not point back to apple after the call to `addFront`.

Write one or two lines of code after the call to `new` and before the `return` statement so that the prev field of the previously-first-now-second node is correctly assigned to point back to the new first node.
Part B (8 points)
Write the method `splice` that inserts a new node containing parameter `str` after every node of the list passed to `splice`. For example, the call `splice(first, "egg")` should result in a list (cherry, egg, fig, egg, mango, egg) if passed the list diagrammed above. All nodes in the list should have `prev` and `next` fields initialized appropriately. You may assume `list` is not null.

```java
/**
 * Add/splice a new node containing str after every node in list.
 * The list is modified since next/prev fields are changed in the
 * process of splicing in new nodes.
 * @param list is the original list
 * @param str is the value spliced after every node of list
 */
public void splice(DNode list, String str) {
```
```
PROBLEM 4: (Tree Bees (25 points))

This question uses the tree below as the basis for the different parts. This tree is a search tree.

**Part A (3 points)**
Add nodes containing the values *justin, owen, and lilly*, in that order, to the tree above so that it remains a search tree. Draw the nodes in the tree diagram above.

**Part B (3 points)**
The first sixteen Catalan numbers are listed below.

```
1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786, 208012, 742900, 2674440, 9694845.
```

How many different binary search trees are there with the same number of nodes as the tree diagrammed above?

**Part B (4 points)**
Write the post-order traversal of the four nodes of the tree rooted at *patrick*

**Part C (5 points)**
There are five nodes in the tree that have exactly two children. Complete method `countTwo` below so that it returns the number of nodes in a tree that have exactly two children:

```java
public int countTwo(TreeNode root) {

}
```
Part D (10 points)
Complete the method rangeCount below so that it returns the number of nodes in the tree rooted at root whose values are between (and including) the parameters first and last. The method you write should examine/traverse the minimal number of nodes of the tree in determining this count. For example, the statement below should store 3 in result and visit the shaded nodes in the tree on the previous page when root points to the "nathan" node. You get half credit for any correct code, but full credit for visiting the minimal number of nodes.

    int result = rangeCount(root, "max", "riley");

Write the method below

    /**
     * Return count of number of nodes containing values in [first..last],
     * assume first <= last, i.e., first.compareTo(last) <= 0.
     */
    public int rangeCount(TreeNode root, String first, String last) {

(nothing on this page)