This is a closed book test. You may not refer to any resources.

This is a 50 minute test.

Please write your answers in ink. Pencil answers will be marked, but will not be re-marked under any circumstances.

The test will be marked out of 50.

<table>
<thead>
<tr>
<th>Question</th>
<th>Marks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Question 1</td>
<td>/12</td>
</tr>
<tr>
<td>Question 2</td>
<td>/12</td>
</tr>
<tr>
<td>Question 3</td>
<td>/10</td>
</tr>
<tr>
<td>Question 4</td>
<td>/6</td>
</tr>
<tr>
<td>Question 5</td>
<td>/10</td>
</tr>
<tr>
<td><strong>TOTAL</strong></td>
<td><strong>/50</strong></td>
</tr>
</tbody>
</table>

“Red, white, black, brown or yellow, rich or poor, we all have the blues.”

B. B. King
Question 1 (12 marks)

(a) What is the big-O complexity of finding the largest value in a set of $n$ values stored in a Red-Black tree? Explain your answer.

Solution: The complexity is in $O(\log n)$. Explanation: we find the largest value in the Red-Black tree by repeatedly going to the right until we can’t go any further. The number of steps is directly proportional to the number of levels in the tree, which we know is $\leq 2 \times \log n$.

Marking:

Students are not required to give details of the algorithm used to find the largest value.

Correct complexity and an explanation that refers to the guaranteed $O(\log n)$ height of the tree: 6/6

Correct complexity and an incorrect explanation: 4/6

Correct complexity and no explanation: 3/6

Incorrect complexity (O(n) etc.) and an explanation referring to the height of the tree: 3/6

Incorrect complexity and an incorrect explanation: 2/6

Incorrect complexity and no explanation: 1/6
(b) What is the big-O complexity of finding the second-largest value in a set of $n$ values stored in a Red-Black tree? Explain your answer.

Solution: The complexity is in $O(\log n)$. Explanation: if the largest value has a left child, the second largest value in the set is the largest value in the subtree rooted at the left child of the largest value. If the largest value has no left child, the second-largest value is the parent of the largest value. Either way, the most work we will do visit one vertex on each level ... and as in part (a), we know the number of levels is $\leq 2 \times \log n$

Marking: Same as for part (a)
Question 2 (12 marks)

Write an algorithm (in pseudocode or any of Java, Python, or C) to compute the average depth of vertices in a Red-Black tree with \( n \) stored values where \( n \geq 0 \). Your algorithm must run in \( O(n) \) time.

You may assume that the root has depth 1.

Be sure to handle all cases.

Solution:

To compute the average depth we need to know the number of values in the tree. It’s ok to assume that this is given, but it’s easy to compute it.

// Let root be a pointer to the root of the tree.

def findSize(current):
    if (current.isLeaf):
        return 0
    else:
        return 1 + findSize(current.left) + findSize(current.right)

def findTotalDepth(current,d):
    if (current.isLeaf):
        return 0
    else:
        return d + findTotalDepth(current.left,d+1) + findTotalDepth(current.right,d+1)

n = findSize(root)
if n == 0:
    print “Tree is empty: Average depth = 0”
else:
    totalDepth = findTotalDepth(root,1)
    print “Average depth = “,totalDepth/n
Marking for Question 2:

This question should have been a piece of cake since it is a simple variation of part of the homework assignment.

Students may include the leaves in the calculation. This is ok – the base cases would be “current == null” instead of “current.isLeaf”.

As noted, it is ok if the students assume that the number of values in the tree is already known..

The algorithm shown is the most natural way to do it, but other solutions are probably possible.

Students were not asked to prove the O(n) complexity of their solutions.

Algorithm that correctly computes the result in O(n) time: 12/12

Algorithm that correctly computes the result but not in O(n) time: 7/12

Algorithm that does not correctly compute the result, but runs in O(n) time: 5/12

Algorithm that does not correctly compute the result, and does not run in O(n) time: 3/12

Deduct 2 marks from an algorithm which correctly computes the result for non-empty trees but fails to deal properly with the case where n = 0
Question 3 (10 marks)

Prove that in a properly coloured Red-Black tree, the sibling of a leaf cannot be an internal Black vertex.

Solution:

Suppose $T$ is a properly coloured Red-Black tree in which there is a leaf $x$ whose sibling is an internal Black vertex. The parent of these two vertices will see exactly 1 Black vertex going down to the leaf $x$. But going down the other side it will see the Black sibling of the leaf and eventually it will reach another leaf, which is also Black. Therefore the parent will see 1 Black vertex going down one side, and at least 2 Black vertices going down the other side. This violates the colouring rules for Red-Black trees so $T$ is not properly coloured – CONTRADICTION.

Therefore a properly coloured Red-Black tree cannot contain a leaf whose sibling is a Black internal vertex.

Marking:

For an argument that shows that such a situation would lead to a violation of the colouring rules: 10/10

For an argument that shows understanding of the colouring rules but does not show that this situation would violate them: 5/10

For an argument that shows very weak understanding of Red-Black trees: 2/10
Question 4 (6 marks)

In a Red-Black tree, the leaves are never used to contain values from the set being stored in the tree. What is the purpose of having these empty leaves in the tree?

Solution:

Having the empty leaves in the tree simplifies the coding – every vertex that contains a value is guaranteed to have two children so we never have to test for the children being null before we check their colour. As a concrete example, when we are dealing with a Red-Red situation and we want to check the colour of the parent’s sibling, we don’t have to write

```java
if ((sibling != null) && (sibling.colour == "Red")): ...
```

because we already know the sibling exists and has a colour. So we can just write

```java
if (sibling.colour == "Red"): ...
```

Marking:

For this or a similar answer: 6/6

For an answer that gives a different but valid reason for having the leaves (note: “because that is how Red-Black trees are defined” is not a valid reason): 6/6

For an answer that shows a good understanding of the colouring rules for Red-Black trees but doesn’t answer the question: 3/6

For an answer that doesn’t answer the question, and doesn’t show understanding of the colouring rules: 1/6
Question 5 (10 marks)

Suppose we have a hash table with $m = 50$ addresses and we resolve collisions using quadratic probing with $c_1 = c_2 = 2$

Thus

$$h(k, i) = (h'(k) + 2i + 2i^2) \mod 50$$

Does this seem like a good combination of $m$, $c_1$ and $c_2$? Why or why not?

Suggestion: consider some probing sequences.

Solution:

$$2i + 2i^2 = 2(i + i^2)$$ which is always even – in fact it is always a multiple of 4. Thus all the offsets from $h'(k)$ will be even, so all addresses that are an odd number of spaces away from $h'(k)$ will not appear in the probe sequence.

Examples:

Suppose $h'(k) = 0$. The probing sequence is 0, 4, 12, 24, 40, 10, 34 … All the addresses in this probing sequence are even, and this will be true whenever $h'(k)$ is even.

Similarly, suppose $h'(k) = 1$. The probing sequence is 1, 5, 13, 25, 41, 11, 35 … This probing sequence contains only odd addresses, and this will be true whenever $h'(k)$ is odd.

Thus every probing sequence contains at most half of the addresses in T. This is bad!

Therefore this is not a good combination of $m$, $c_1$ and $c_2$.
Marking:

Students do not have to write out the probing sequences – the suggestion was intended to guide them to the observation that all the offsets from h′(k) are even. Students may realize this without writing out the sequences.

A solution that shows that the probing sequences miss at least half of all addresses: 10/10

A solution that explores some probing sequences and observes that they consist of only even addresses or only odd addresses, but does not show that this is always the case: 7/10

A solution that makes a general claim such as “m should be prime” or “m should be odd” or “c₁ and c₂ should not be equal” without explanation: 5/10

A solution that does not show an understanding of quadratic probing: 2/10