CPSC 331 Spring 2022

Jaza Khan UCID 30119100

Assignment 3 - Proofs & Running Time Analysis

BST-MEDIAN ALGORITHM

Suppose that a binary search tree T has m nodes, all distinct, and of depth d, where m is an odd number. Design an efficient algorithm which runs in $O(d)$ time that finds the median-valued node of T (note that since the number of nodes m of the tree T is odd, there is exactly one median-valued node).

```
// Precondition: root is the root node of the binary search tree, nodesToMedian is the target node we are trying to reach, the number of nodes m in the tree is odd
// Postcondition: the node containing the median is found and returned

findBSTMedian(Node root, integer nodesToMedian) {
    if (root == null) {
        return
    }
    integer rightSize := root.right.size
    if (nodesToMedian > rightSize) {
        findBSTMedian(root.left, nodesToMedian - rightSize - 1)
    }
    else if (nodesToMedian < rightSize) {
        findBSTMedian(root.right, nodesToMedian)
    }
    else if (nodesToMedian == rightSize) {
        print("Median node: " + root)
    }
    else {
        print("Median node:  " + root)
    }
}
```

PROOF OF CORRECTNESS

An algorithm, for a given computational problem, is correct if the following property holds:

- If the algorithm is executed and the problem's precondition is satisfied, then the execution of the algorithm eventually ends, and the problem's postcondition is satisfied
    - No changes to the system state, that are not documented in the precondition and postcondition, should be caused by the execution of this algorithm if the precondition holds when the execution begins

This can be proven using the strong form of mathematical induction on d, where there base cases $d = 0$ and $d = 1$ will be considered.

Basis step

Suppose, first, that $d = 0$. Tracing the algorithm: this causes the body of the third `else-if` statement to execute, with the value of the only node returned as output — as required for this case as it would be the median.

Now suppose that $d = 1$. Since the number of nodes m has to be odd, this implies that the `size` of both the left and right subtree are 1. Tracing the algorithm: this will also cause the body of the third `else-if` statement to execute, returning the current `root` node as the median, which is still the `root` node of the entire tree.

Inductive step

Let k be an integer such that $k \geq 1$. It is necessary and sufficient to use the following *inductive hypothesis* to prove the following *inductive claim*:

Inductive hypothesis: Suppose that d is a nonnegative integer such that $0 < d \leq k$. If the algorithm `findBSTMedian()` is executed given a binary search tree with depth d as input then this execution of the algorithm eventually terminates, with the median of the binary search tree with depth d being returned.

Inductive claim: If the algorithm `findBSTMedian()` is executed given a binary search tree with depth $d = k + 1$ as input then this execution of the algorithm eventually terminates, with the median of the binary search tree with depth $k + 1$ is returned.

Proof

Suppose this algorithm is executed on a binary search tree with depth $d = k + 1$.

Lines 5-7 check if the `root` node is null. If it is not null, then:

Line 8 computes the `size` of the node immediately to the right of the current `root`'s node. The depth of this subtree is now $(k + 1) - 1$, or k.

It now follows by the *inductive hypothesis* that an execution of this algorithm with a binary search tree of depth k eventually terminates, with the median of this tree being returned.

One can see by inspection of the code that this does not change this binary search tree or have other undesired side-effects, so the algorithm `findBSTMedian()` can be considered **correct**.

RUNNING TIME ANALYSIS

We can analyze the runtime of this algorithm using the **uniform cost criterion** in order to form a function $f(d)$.

Suppose this algorithm is executed on a binary search tree with m nodes, all distinct, and of depth d, where m is an odd number.

Then:

- Line 5 costs 1 step, as it is a simple check to see if the `root` node is `null`
    - In the case that it is, line 6 costs 1 step as well
- Line 8 costs 1 step to set the `rightSize` variable using the provided `size` property
- Lines 9, 12 and 15 each cost 1 step, as they are `if` statements comparing the two sizes
- Line 10 contains a recursive call to the function which costs $2d$* steps *at worst*
- Line 13 contains a recursive call to the function which costs $2d$ steps *at worst*
- Lines 16 and 19 each cost 1 step, as they are simply returning the median node

$$findBSTMedian(\text{depth } d) = \begin{cases} 4 & \text{if } d = 0 \\ 4 & \text{if } d = 1 \\ 4d + 4 & \text{if } d \geq 2 \end{cases}$$

* I was able to arrive at this upper bound by tracing the algorithm for various types of binary search trees in depths $d = 0, 1, 2, 3$ and 4 and identying a pattern for the number of steps each recursive call took *at most*.

PROOF OF LINEAR RUNTIME

We will now prove that the runtime of the `findBSTMedian()` algorithm is in $O(d)$, **by application of the definition.**

By definition of $O(d)$, we are required to show that there exists a constant $c > 0$ and a constant $N_0 > 0$ such that $f(d) \leq cd$ for all $d \in \mathbb{R}$ and $d \geq D_0$.

Let $c = 15$ and $D_0 = 1$.

Let d be an *arbitrarily chosen element* in the range of f such that $d \geq D_0$. We must prove that the claim holds for this choice of d.

Then,

$4d + 4 \leq 4d + 4d = 15d$

since $1 \leq 1 \leq d$ whenever $d \geq 1$. Since d was arbitrarily chosen from $\mathbb{R}$, it follows that $4d + 4d \leq 15d = cd$ for all $d \in \mathbb{R}$ such that $d \geq 1 = D_0$. Since $c = 15$ and $D_0 = 1$ are constants, this establishes the claim that they are *existentially quantified*, as needed to conclude $4d + 4 \in O(d)$.

We have now proved a result for all d in the range of f such that $d \geq D_0$.

Another way of proving that this algorithm is in linear time is using a **limit test** for $O(g)$.

By definition of the theorem, if $\lim_{x \to +\infty} \frac{f(x)}{g(x)}$ exists and is a real constant — so that in particular, it is not
equal to $+\infty$ — then $f \in O(g)$.

Let $f(d) = 4d + 4$ and $g(d) = d$. Then,

$\lim_{d \to +\infty}(\frac{4d+4}{d})$

$= \lim_{d \to +\infty}(4 + \frac{4}{d})$

$= \lim_{d \to +\infty}\ (4) + \lim_{d \to +\infty}\ (\frac{4}{d})$

$= 4$

Since 4 is a real and non-negative constant, it now follows by the Limit Test for $O(g)$ that this algorithm is in $O(d)$, where d is the depth of the binary search tree.