CPSC 331

David Ng

Winter 2016

Question 1: The loop invariant for the iterative search algorithm for a binary search tree is presented below. The following properties are satisfied at the beginning of each execution of the loop body:

- $curr \neq null$
- $v = null$
- T and key have not been changed

We will prove that the algorithm is partially correct by induction on the loop invariant.

Base Case

- Before the first execution of the loop body, we have assigned $v = null$, so $v = null$ is satisfied before any iteration of the loop body
- We have assigned $curr = T.root$, and when loop test passes, this implies that $curr \neq null$.
- T and key have not been changed

Inductive Hypothesis
Assume that the loop body is executed $k \geq 0$ times, and the loop invariant is satisfied at the beginning of the kth execution. So, at the end of the kth execution:

- T and key have not been changed
- v has been unchanged, or is now $v = curr.value$
- $curr$ has been unchanged, or it is now $curr.left$ or $curr.right$.

*Note that only one of v or $curr$ could have changed, since the algorithm specifies cases. Now, if there is a $k+1$st iteration, then the loop test must have passed after the end of the kth execution (meaning that $curr \neq null$ and $v = null$), implying that immediately before the $k+1$st iteration,

- $curr \neq null$
- $v = null$
- T and key have not been changed

Thus, the loop invariant holds. We note then, that the search algorithm is partially correct, since the preconditions imply that the loop invariant will be satisfied. We now apply the loop invariant to prove that the algorithm's postconditions hold at the end of the loop.

The fact that T and key have not changed follows directly from the loop invariant. From the termination condition, we have either $curr = null$ or $v \neq null$ (not both).

- Case 1: $curr = null$. It then follows that $v = null$, so a KeyNotFoundException is thrown since key is not in T.
- Case 2: $v \neq null$. This means that v is returned by the if statement, since key is a key in T, so the associated value, v is returned.

We also realize that this algorithm eventually terminates. This is because for any binary search tree of finite size, a specified key will either be in the tree, or not be in the tree.

First, we consider the case that the key is in the tree. After each iteration of the while loop, the depth of our search has increased by 1. Since the key exists at a certain depth and our comparisons with the other keys lead us to take the path (left or right) towards that key, it will eventually be the case that $key = curr.key$ and since $v \neq null$, the loop terminates.

Now, we consider the case that the key is not in the tree. Since this tree is of a finite size, we will at most have to search through the entire height of the tree making appropriate comparisons. Because the key is not in the tree, we will eventually reach a leaf of the tree, and continue to search its nonexistent children for the key. At this point, $curr = null$, so the loop terminates as well.

Alternatively, we may consider a loop variant if we define it based on the height of the node at which key is located (or where it would be if it was in to be inserted into the tree). If we denote the height of the key as h, then we can define the function $f(h, d) = h - d$ where d denotes the current depth of the current node. We note that this is an integer valued function (since h and d are both integers). Furthermore, $f(h, d) = h - d$ decreases after every iteration of the loop because d increases which h remains constant. Lastly, $f(h, d) \leq 0$ when $d \geq h$, and the while loop terminates when $d = h$.

Question 2: Pseudocode for an iterative algorithm for insertion into a binary search tree is given below. To insert a new value v into the binary search tree T, we add a node w where $w.key = v$, $w.left = null$, and $w.right = null$.

```
Insertion(T,z)
y = null
x = T.root
while x ≠ null do
    y = x
    if z.key == x.key then
        throw KeyFoundException
```

```
        else if z.key < x.key then
            x = x.left
        else
            x = x.right
    z.parent = y
    if y == null then
        T.root = z
    else if z.key < y.key then
        y.left = z
    else
        y.right = z
```

Question 3: Pseudocode for an iterative algorithm for an inorder traversal of a binary search tree.

```
InorderTraversal(T)
s = newStack()
x = T.root
while !s.isEmpty() or x ≠ null do
    if x ≠ null then
        s.push(x)
        x = x.left
    else
        x = s.pop()
        visit(x)          //print out next value
        x = x.right
```