CSE1729 - Introduction to Programming | March 22, 2017

Laboratory Assignment 8 Solutions

Objectives

- Work with trees
- Use Binary Search Trees

Activities

Trees

1. Write a SCHEME function named `tree-depth` which takes a tree node, n, as a parameter and returns the depth of the tree rooted at n. For the purpose of this exercise, the depth of a tree rooted at n is one plus the maximum of the depths of its two children. Recall the following code from the lecture slides:

```
(define (make-tree value left right)
  (list value left right))

(define (value T) (car T))
(define (right T) (caddr T))
(define (left T)  (cadr T))

(define (insert x T)
  (cond ((null? T) (make-tree x '() '()))
        ((eq? x (value T)) T)
        ((< x (value T)) (make-tree (value T)
                                    (insert x (left T))
                                    (right T)))
        ((> x (value T)) (make-tree (value T)
                                    (left T)
                                    (insert x (right T))))))
```

Solution:

```
(define (tree-depth T)
  (if (null? T) 0
      (let ((d-left (tree-depth (left T)))
            (d-right (tree-depth (right T))))
        (if (> d-left d-right)
            (+ 1 d-left)
            (+ 1 d-right)))))
```

2. Define a SCHEME function named `occurences-in-tree` which given a binary tree and a value, returns the number of occurrences of the value in the tree.

Solution:

```
(define (occurences-in-tree T val)
   (if (null? T) 0
         (if (= (value T) val)
             (+ 1
                (occurences-in-tree (left T) val)
                (occurences-in-tree (right T) val))

             (+ (occurences-in-tree (left T) val)
                (occurences-in-tree (right T) val)))))
```

3. Define a SCHEME function named `count-one-child` which returns the number of internal nodes of a binary tree which have exactly one child.

Solution:

```
(define (count-one-child tree)
  (let ((left-child (left tree))
        (right-child (right tree)))
    (cond ((and (null? left-child)
                (null? right-child)) 0)
          ((and (not (null? left-child))
                (not (null? right-child)))
           (+ (count-one-child left-child)
              (count-one-child right-child)))
          ((null? left-child)
           (+ 1 (count-one-child right-child)))
          (else (+ 1 (count-one-child left-child))))))
```

Binary Search Trees

4. Define a SCHEME function named `(invert-bst T)` which, given a Binary Search Tree (BST) *inverts* that BST. That is, the function returns a binary tree in which has the property that all values at nodes in the left subtree are greater than the value at the root node and all values at nodes in the right subtree are less than the value at the root node.

Solution:

```
(define (invert-bst T)
  (if (null? T)
      (list)
      (make-tree (value T) (invert-bst (right T))
                           (invert-bst (left T)))))
```

5. Write a SCHEME function which, given a binary search tree T and an integer z, returns the number of integers in the tree that are greater than or equal to z. For example, given the tree below and the number 5, your function should return 4, since there are 4 numbers in the tree that are greater than or equal to 5 (specifically, 5, 8, 11, and 21).

   Your solution should exploit the fact that the tree is a binary search tree to avoid considering certain subtrees.

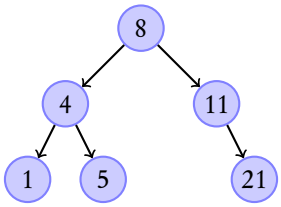

Figure 1: A binary search tree with 6 nodes, 4 of which are 5 or larger.

Solution:

```
(define (num-gteq-z z T)
  (cond ((null? T) 0)
        ((< (value T) z)
         (num-gteq-z z (right T))) ;all nodes in left
                                   ;subtree are < value here
        (else (let ((num-gteq-left (num-gteq-z z (left T)))
                    (num-gteq-right (num-gteq-z z (right T))))
                    (+ 1 num-gteq-left num-gteq-right)))))
```