MASSACHVSETTS INSTITVTE OF TECHNOLOGY
Department of Electrical Engineering and Computer Science
6.001—Structure and Interpretation of Computer Programs
Fall 2007

Recitation 9 Solutions
Trees

Binary Trees

A binary search tree is a recursively defined data structure which allows for fast searches: lookups take $\Theta(\log n)$ time.

In order to support such searches, an invariant on each tree node holds: Each (nonempty) node has a value, and at most two child trees, with the requirement that any value reachable down the left subtree is smaller than the root value, and any value reachable down the right subtree is larger.

```
;a tree is either an empty tree, or a tree-node (defined below)
(define the-empty-tree null)
(define empty-tree? null?)
(define tree? list?)

(define (make-tree-node value left-subtree right-subtree)
  (list value left-subtree right-subtree))

;selector
(define (node-value node)
  (car node))

(define (node-left node)
  (cadr node))

(define (node-right node)
  (caddr node))
```

Problems

1. Complete the definition for `tree-lookup`, which returns true if the value is present in the tree.

```
(define (tree-lookup val tree)
  (cond ((empty-tree? tree) #f)
        ((= val (node-value tree)) #t)
        ((< val (node-value tree))
         (tree-lookup val (node-left tree)))
        (else
         (tree-lookup val (node-right tree)))))
```

2. Fill in the definition for `tree-insert`, which takes in a tree and a val and returns a new tree with the value added.

```
(define (tree-insert val tree)
  (cond ((empty-tree? tree)
         (make-tree-node val
                         the-empty-tree
                         the-empty-tree))
        ((< val (node-value tree))
         (make-tree-node (node-value tree)
                         (tree-insert val (node-left tree))
                         (node-right tree)))
        (else
         (make-tree-node (node-value tree)
                         (node-left tree)
                         (tree-insert val (node-right tree))))))
```

3. Consider the tree that results from evaluating the following

   `(tree-insert 1 (tree-insert 2` ... `(tree-insert` n `the-empty-tree)`

   What is the running time of calling `tree-lookup` on such a tree?

   $\Theta(n)$. This tree is entirely unbalanced – there are never any left children, so it is equivalent to a list.

4. Write a procedure, `build-balanced-tree`, that takes a list of sorted elements, and returns a balanced binary tree of those elements, ie one in which `tree-lookup` will run in $\Theta(\log n)$ time. Your solution (constructing the tree) may be slower than $\Theta(n)$ time, so long as lookups are fast.

   You may use the provided functions if you wish:

```
;return the last k elements of l
(define (list-tail l k)
  (if (zero? k)
      l
      (list-tail (cdr l) (- k 1))))

;return a list of the first k elements of l
(define (list-head l k)
  (if (zero? k)
      '()
      (cons (car l) (list-head (cdr l) (- k 1)))))

;lst must be sorted in increasing order
(define (build-balanced-tree lst)
  (let ((a (length lst)))
    (cond ((= a 0) the-empty-tree)
          ((= a 1) (tree-insert (car lst) the-empty-tree))
          (else
           (let ((pivot (quotient a 2)))
             (make-tree-node
              (list-ref lst (- pivot 1))
              (build-balanced-tree
               (list-head lst (- pivot 1)))
              (build-balanced-tree
               (list-tail lst pivot))))))))
```

5. Challenge: How would you construct a balanced binary tree starting with an unsorted list? Sorting it first and then passing the result to `build-balanced-tree` as defined above is not the fastest answer.

   The general approach is to require, at each node, that the depth of each child branch differs by at most one. On inserts, an operation called a pivot may be required to replace the root of the tree with a value from the taller subtree. For more discussion, searh for "AVL Tree" on google or wikipedia.