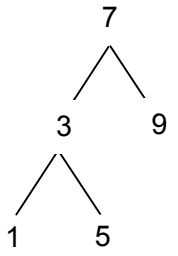
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
Department of Electrical Engineering and Computer Science
6.001 Structure and Interpretation of Computer Programs
Spring, 2007
Recitation 13, April 4

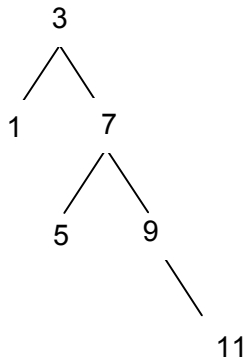
Binary Tree Problems

Dr. Kimberle Koile

Ordered Sets of Elements

A *binary tree* is a tree in which each node is represented by an *entry* and a *link* to each of two other (possibly empty) nodes; each node is itself a (sub)tree. (See Section 2.3.3 in textbook.) The "left" link points to elements smaller than the one at the node, and the "right" link to elements greater than the one at the node. Two trees that represent the set {1, 3, 5, 7, 9, 11}. To check whether an element x is in a set, we compare x with the entry at the top. If it is less than that element, we only have to search the left subtree; if it is greater, we only have to search the right subtree.



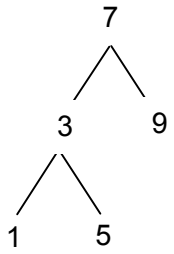


1. A binary tree is said to be *balanced* if each of the subtrees pointed to by left and right links at each node have approximately the same number of elements, and are therefore about half the size of the original. If a balanced binary tree is of size n, how many nodes might we have to search to find a number x? What does this imply about how the number of steps will grow as n grows?

Since the tree is halved at each step, the number of steps to search a tree of size n grows Θ log n.

2. Assume that we have the following representation for a binary tree and that the entries are numbers:

```
(define (make-node entry left right)
  (list entry left right))
(define (entry node)
  (car node))
(define (left-branch node)
 (cadr node))
(define (right-branch node)
  (caddr node))
(define (empty-tree?  tree)
   (null?  tree))
(define empty-tree '())
```




```
(7
 (3
    (1  ()  ())
    (5  ()  ()))
 (9  ()  ()))
```

Write a procedure element-of-tree? that searches a binary tree to find an element using the strategy above.

```
(define (element-of-tree? x tree)
    (cond ((empty-tree?  tree)  #f  )
          ((= x (entry tree))  #t )
          ((< x (entry tree)
            (element-of-tree?  x  (left-branch tree))
            )
          (else
            (element-of-tree?  x  (right-branch tree)))

      ))
```

3. Write a procedure insert-element that inserts a new entry into a binary tree. (Be sure to take care of the case in which the initial tree is empty.) Hint: The procedure will contain a cond that looks very similar to the one in question 2.

```
(define (insert x tree)
  (cond ((null? tree) (make-tree x empty-tree empty-tree))
      ((< x (entry tree))
        (make-tree (entry tree)
              (insert x (left-branch tree))
              (right-branch tree)))
      ((> x (entry tree))
        (make-tree (entry tree)
              (left-branch tree)
              (insert x (right-branch tree))))))
```

4. Write a procedure min-tree-element that finds the smallest element in a binary tree.

```
(define (min-tree-elt tree)
  (cond ((null?  tree)  #f)
        ((null?  (left-branch tree))
              (entry tree))
        (else  (min-tree-elt  (left-branch tree)))))
```

5. Write a procedure max-tree-element that finds the largest element in a binary tree.

```
(define (min-tree-elt tree)
  (cond ((null?  tree)  #f)
        ((null?  (right-branch tree))
              (entry tree))
        (else  (min-tree-elt  (right-branch tree)))))
```

6. What changes would you have to make for the above representation and procedures to work for a binary tree whose elements were not numbers?

write less-than and greater-than procedures for the element type