Fall 2001 **LISP Warmup** CMSC 420

Hanan Samet

This problem reflects the philosophy that one learns fastest and best by doing. In this problem, you will be asked to run a given LISP program and observe and describe its behavior.

For this exercise, we will need two LISP data structures, `tree1` (a binary tree), and `tree2` (a general tree):

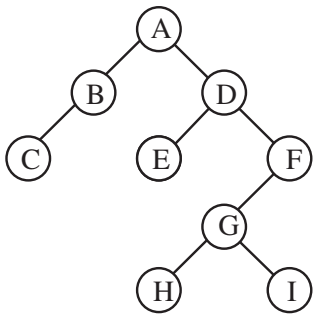

Figure 1: `tree1`

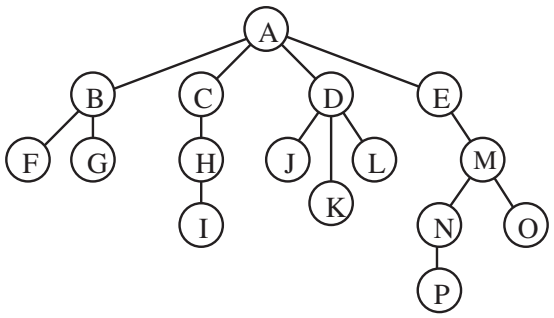

Figure 2: `tree2`

We will represent these two trees in LISP parenthesized notation. To begin this exercise type `lisp` into the command shell. At this point, you will be talking to LISP.

Next, set up this exercise by typing the following five s-expressions:

Define `tree1`:

```
(setq tree1 '(a (b (c nil nil) nil) (d (e nil nil)
             (f (g (h nil nil) (i nil nil)) nil)))
)
```

Define `tree2`:

```
(setq tree2 '(a (b (f) (g)) (c (h (i))) (d (j) (k) (l))
             (e (m (n (p))(o))))
)
```

Define function `searchb`:

```
(defun searchb (tree node)
    (cond ((null tree) nil)
          (t (or (eq (car tree) node)
                 (searchb (cadr tree) node)
                 (searchb (caddr tree) node)))))
```

Define function `search`:

```
(defun search (tree node)
    (cond ((null tree) nil)
          (t (or (eq node (car tree))
                 (searchsons (cdr tree) node)))))
```

Define functions `searchsons`:

```
(defun searchsons (sonlist node)
    (cond ((null sonlist) nil)
          ((search (car sonlist) node) t)
          (t (searchsons (cdr sonlist) node))))
```

Next, turn on the function trace mechanism for the tree functions so that you may observe the sequence of function calls which are about to occur, as well as the function values:

```
(trace searchb search searchsons)
(pp searchb search searchsons tree1 tree2)
```

You are now ready to run `searchb` on `tree1` and `search` on `tree2`. Type:

```
(searchb tree1 'g)
```

and pause to observe the trace output. Then do the same for:

```
(searchb tree1 'x)
(search tree2 'k)
(search tree2 'x)
```

Finally, either exit LISP by typing:

```
(exit)
```

or stick around and play with LISP a while. If you ever get yourself into a state you don't understand, type:

```
^D (control D)
```

until you get to the top level and then exit to the system.

Now for the fun part: the actual problem questions. You may write your answers on the output listing you have obtained. Turn in both the listing and your written answers.

1. Describe the notation used to represent binary tree `tree1` in LISP parenthesized notation. Write the binary tree in Figure 3 in this notation.
2. Describe the notation used to represent the general tree `tree2` in LISP. Write the tree in Figure 4 in this notation.
3. What does `searchb` do? Identify the recursion relations and the termination test.
4. What does `search` do? Identify the recursion relations and the termination test.
5. What does `searchsons` do? At entry, what is `sonlist`?

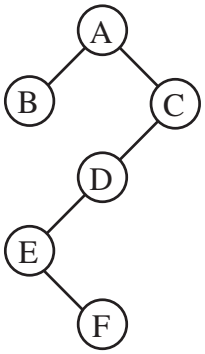

Figure 3: A binary tree

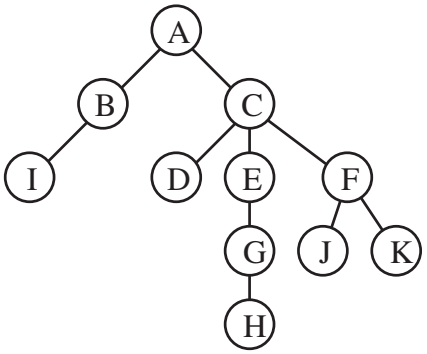

Figure 4: A general tree

6. Why is it necessary to have two distinct recursive functions, `search` and `searchsons`, to process general tree `tree2`, while only one recursive function, `searchb`, is required to process `tree1`? Draw a simple picture illustrating who calls who in the processing of `tree2`.

7. `searchb`, `search` and `searchsons` are really predicates in that they return a logical value, either `t` or `nil`. Suppose we want `searchb` to return a pointer to some part of the binary tree where it now returns only `t`. What would the appropriate value to return be? Write a modified `searchb`, `searchb1`, which would return such a pointer. Run this function if you have time and show its output for some example.