

CS61A Notes – Week 5: Trees, deep lists (solutions)

Fake Plastic Trees

QUESTIONS

1. Write (`square-tree tree`), which returns the same tree structure, but with every element squared. Don't use "map"!

```
(define (square-tree tree)
  (make-tree (square (datum tree))
             (square-forest (children tree))))
(define (square-forest forest)
  (if (null? forest)
      '()
      (cons (square-tree (car forest)) (square-forest (cdr forest)))))
```

2. Write (`max-of-tree tree`) that does the obvious thing. The tree has at least one element.

```
(define (max-of-tree tree)
  (if (null? (children tree))
      (datum tree)
      (max (datum tree) (max-of-forest (children tree)))))
(define (max-of-forest forest)
  (if (null? (cdr forest))
      (max-of-tree (car forest))
      (max (max-of-tree (car forest)) (max-of-forest (cdr forest)))))
```

3. Write (`listify-tree tree`) that turns the tree into a list in any order.

```
(define (listify-tree tree)
  (cons (datum tree) (listify-forest (children tree))))
(define (listify-forest forest)
  (if (null? forest)
      '()
      (append (listify-tree (car forest)) (listify-forest (cdr forest)))))
```

4. A maximum heap is a tree whose children's data are all less-than-or-equal-to the root's datum. Of course, its children are all maximum heaps as well. Write (`max-heap? tree`) that checks if a given tree is a maximum heap.

```
(define (max-heap? tree)
  (and (= (datum tree) (max-of-tree tree))
        (max-heaps? (children tree))))
(define (max-heaps? forest)
  (cond ((null? forest) #t)
        (else (and (max-heap? (car forest))
                    (max-heaps? (cdr forest))))))
```

QUESTIONS

1. **Jimmy the Smartass was told to write `(valid-bst? bst)` that checks whether a tree satisfies the binary-search-tree property – elements in left subtree are smaller than datum, and elements in right subtree are larger than datum. He came up with this:**

```
(define (valid-bst? bst)
  (cond ((null? bst) #t)
        (else
         (and (or (null? (left-branch bst))
                  (and (< (datum (left-branch bst)) (datum bst))
                       (valid-bst? (left-branch bst))))
              (or (null? (right-branch bst))
                  (and (> (datum (right-branch bst)) (datum bst))
                       (valid-bst? (right-branch bst))))))))))
```

Why will Jimmy never succeed in life? Give an example that would fool his pitiful procedure.

Checking if the `bst` property is true for your immediate children's labels does not guarantee that the property holds for the whole subtree. For example, this tree would fool `valid-bst?`:

```
      10
     /  \
    5    18
     \   / \
      1  30
```

The 1 violates the `bst` property (1 is not larger than 10), but Alex's algorithm will merely check that 1 is smaller than 18, and move on.

Can you do better?

2. **Write `(sum-of bst)` that takes in a binary search tree, and returns the sum of all the data in the tree.**

```
(define (sum-of bst)
  (cond ((null? bst) 0)
        (else (+ (datum bst)
                  (sum-of (left-branch bst))
                  (sum-of (right-branch bst))))))
```

3. **Write `(max-of bst)` that takes in a binary search tree, and returns the maximum datum in the tree. The tree has at least one element. (Hint: This should be easy.)**

```
(define (max-of bst)
  (cond ((null? (right-branch bst)) (datum bst))
        (else (max-of (right-branch bst)))))
```

4. **Write `(listify bst)` that converts elements of the given `bst` into a list. The list should be in **NON-DECREASING ORDER!****

```
(define (listify bst)
  (cond ((null? bst) '())
        (else (append (listify (left-branch bst))
                       (list (datum bst))
                       (listify (right-branch bst))))))
```

5. Write (`remove-leaves bst`) that takes in a `bst` and returns the `bst` with all the leaves removed.

```
(define (remove-leaves bst)
  (cond ((null? bst) '())
        ((leaf? bst) '())
        (else (make-tree (datum bst)
                          (remove-leaves (left-branch bst))
                          (remove-leaves (right-branch bst))))))
```

6. Write (`height-of tree`) that takes in a `tree` and returns the `height` – the length of the longest path from the root to a leaf.

```
(define (height-of tree)
  (cond ((leaf? tree) 0)
        (else (+ 1 (max (height-of (left-branch tree))
                        (height-of (right-branch tree))))))
```

7. (HARD!) Write (`width-of tree`) that takes in a `tree` and returns the `width` – the length of the longest path from one leaf to another leaf.

```
(define (width-of tree)
  (cond ((or (null? tree) (leaf? tree)) 0)
        (else (max (+ 2 (height-of (left-branch tree))
                    (height-of (right-branch tree)))
                  (width-of (left-branch tree))
                  (width-of (right-branch tree))))))
```

Deep Lists

QUESTION

Write a procedure (`file-list file-entry`) that, given a file entry for a directory, returns a list of names of the non-directory files anywhere in the corresponding directory tree. For example, given the file entry above, `file-list` should return the list (`proj2.scm proj2-2.scm readme transcript proj2-copy.scm`), although not necessarily in that order.

```
(define (file-list file-entry)
  (if (eq? (car file-entry) 'FILE)
      (list (cadr file-entry)
            (file-list-of-files (cddr file-entry))))

(define (file-list-of-files file-entries)
  (if (null? file-entries)
      '()
      (append (file-list (car file-entries))
              (file-list-of-files (cdr file-entries)))))
```