

Computer Science 61A

Faculty: Brian Harvey

Tutor: Christopher W. Fletcher

Email: chris.w.fletcher@gmail.com



Worksheet 5: Trees

Instructions: Provide a solution for each of the function specifications below. Ideally, each solution should be recursive. To test your functions for correctness, use the following tree:

```
;SAMPLE-TREE
```

```
(define sample-tree
  (make-tree 5 (list (make-tree 3 (list (make-tree 6 '())
                                       (make-tree 4 (list (make-tree 10
                                                         (list (make-tree 12 '()))))))))
              (make-tree 7 (list (make-tree 14 (list
                                               (make-tree 3 '()))))))))
```

- 1.) Write functions **depth**, **sum**, and **maxValue** for a tree data structure where each `make-tree` has a datum and children. Assume that datum is always a number and children is a list (forest) of `make-trees`. HINT: All three should be made up of two mutually recursive functions.

```
Example: > (depth sample-tree)
          5
          > (sum sample-tree)
          64
          > (maxValue sample-tree)
          14
```

- 2.) Write a version of **tree-map**, as described in lecture, that does not use scheme's `map` function.

```
Example: > (tree-map sample-tree (lambda (x) (* 2 x)))
          (10 (6 (12) (8 (20 (24)))) (14 (28 (6))))
```

- 3.) Write **tree-filter**, which performs scheme's `filter` function across all datum in a tree and returns a flattened list containing all remaining datum.

```
Example: > (tree-filter even? sample-tree)
          (6 4 10 12 14)
```

1.) **depth:**

```
(define (depth tree)
  (if (null? (children tree))
      1
      (+ 1 (depth-forest (children tree)))))
```

```
(define (depth-forest forest)
  (biggest (map depth forest)))
```

```
(define (biggest lon)
  (apply max lon))
```

sum:

```
(define (sum tree)
  (if (null? (children tree))
      (datum tree)
      (+ (datum tree) (sum-forest (children tree)))))
```

```
(define (sum-forest forest)
  (apply + (map sum forest)))
```

maxValue:

```
(define (maxValue tree)
  (if (null? (children tree))
      (datum tree)
      (max (datum tree)
           (maxValue-forest (children tree)))))
```

```
(define (maxValue-forest forest)
  (biggest (map maxValue forest)))
```

2.) **tree-map:**

```
(define (tree-map tree proc)
  (if (null? (children tree))
      (make-tree (proc (datum tree)) '())
      (make-tree (proc (datum tree))
                 (tree-map-forest
                  (children tree) proc))))
```

```
(define (tree-map-forest forest proc)
  (if (null? forest)
      '()
      (cons (tree-map (car forest) proc)
            (tree-map-forest (cdr forest) proc))))
```

3.) **tree-filter:**

```
(define (tree-filter proc tree)
  (let ((datum? (if (proc (datum tree))
                    (list (datum tree))
                    '())))
    (if (null? (children tree))
        datum?
        (append datum?
                  (apply append
                         (map (lambda (x) (tree-filter proc x))
                              (children tree)))))))
```