Extra topics, Week 10

Overview

- Prelim.
  - Admin.
  - About the quiz.
- Q & A
  - Let’s watch Sam work through the homework assignment.

Admin

- Upcoming extra credit
  - Talk at 2:15, Science 3821
  - Talk at 4:30, Science 3821
  - And much much more.

What might be on the quiz?

Nothing. No quiz.

Background: Recursion over trees

- Generally has to be direct recursion; passing along a "so-far" doesn’t work. Gather info from subtrees and combine them.
- One of two structures

First structure

```
(define tree-proc
  (lambda (tree)
     (if (pair? tree)
         (combine (tree-proc (car tree))
                  (tree-proc (cdr tree)))
         base-case)))
```

Second structure

```
(define tree-proc
  (lambda (tree)
     (if (pair? tree)
         (let ((left-result (tree-proc (car tree)))
               (right-result (tree-proc (cdr tree))))
           (combine left-result right-result))
         base-case)))
```

Two standard procedures: height and size

- height: How far from the root (top) to the farthest leaf (bottom)
- size: How many things are in the tree, counting both pairs and leaves
  - Irrelvant but interesting: What's the relationshp between the number of pairs in a tree and the number of leaves?
    - There is always one more leaf than pair

Analysis

- height: base case?
  - If there's no pair, the height is 0
- height: recursive case?
  - Take the greater height of the two subtrees, add 1

  (define height (lambda (tree) (if (pair? tree) (+ 1 (max (height (car tree)) (height (cdr tree)))) 0)))

subvector->tree

Review:

- Given a vector of colors, turn part of it into a tree.

Analysis

- Base case: When the start and end index are the same, we can stop
  - We can then grab the color at that index.
- Recursive case:
  - Divide the range in half
  - Recurse on each half
  - Four values to compute: lower bound of left half upper bound of left half lower bound of right half upper bound of right half
- Code

  (define subvector->tree (lambda (colors lower-index upper-index) (if (= lower-index upper-index) (vector-ref colors lower-index) (let* ([left-lower lower-index]
  [left-upper (floor (/ (+ lower-index upper-index) 2))] [right-lower (+ left-upper 1)] [right-upper upper-index]) (cons (subvector->tree colors left-lower left-upper) (subvector->tree colors right-lower right-upper))))))

5c

```
(define height-balanced
  (lambda (tree)
     (if (pair? tree)
         (let ((left-result (height-balanced (car tree)))
               (right-result (height-balanced (cdr tree))))
           (cond
             [(not left-result) #f]
             [(not right-result) #f]
             [(not (<= (abs (- left-result right-result)) 1)) #f]
             [else (+ 1 left-result right-result)])))))
         0)))
```

More elegant with ands and ors.

Samuel A. Rebelsky, rebelsky@grinnell.edu