CMSC 10500-1: Homework 3

(due on Friday July 2nd)

The magical rabbit colony

Long ago, there was a colony of magical rabbits. These rabbits would reproduce at an abominable rate. Each rabbit would take a month to mature. After the initial month, the rabbit produces one offspring every month. Also these magical rabbits never die. The colony started with one lone rabbit.

Let F_n denote the number of rabbits in the colony after n months.

1. **(2 pts)** Calculate F_1, F_2, F_3, F_4 and F_5.

2. **(3 pts)** Prove that for $n > 2$, $F_n = F_{n-1} + F_{n-2}$.

3. **(5 pts)** Write a simple scheme function `fib` which takes an input a number n and returns the value of F_n. Evaluate `(fib 30)` and notice how long it takes to calculate `(fib 30)`. If you are really adventurous you may try calculating `(fib 50)`.

4. **(5 pts)** Observe that the number of recursive calls needed to calculate `(fib 30)` is `(fib 30)`. Write another scheme function `fib2` to calculate F_n so that the number of recursive calls to calculate `(fib2 30)` is only 30, and evaluate `(log (fib2 1476))`.

   Hint: calculate the pair (F_{n+1}, F_n) *from* (F_n, F_{n-1}).

   Comments: `fib2` takes about n recursive calls to calculate F_n. One can actually write a `fib3` which requires less than $4 \log_2 n$ recursive calls to calculate F_n, using another clever trick.

Solution

1. $F_1 = 1, F_2 = 2, F_3 = 3, F_4 = 5, F_5 = 8$.

2. The number of rabbits at the end of the n'th month equals the number of rabbits at the end of the $(n-1)$'st month + the number of rabbits born during the n'th month. The parent of each rabbit born during the n'th month must have been born on or before the $(n-2)$'nd month. Conversely every rabbit living at the end of the $(n-2)$'nd month would have given birth to a rabbit during the n'th month. Thus the number of rabbits born during the n'th month equals the number of rabbits living at the end of the $(n-2)$'nd month. Hence $F_n = F_{n-1} + F_{n-2}$.

3. The obvious solution works here.

```
(define (fib n)
   (cond
      [(<= n 2) n]
      [else (+ (fib (- n 1)) (fib (- n 2)))]
   )
)
```

4. The previous solution calculates F_n by calculating F_{n-1} and F_{n-2} independently and adding them up. Computation of F_{n-1} in turn involves calculation F_{n-2}. Thus F_{n-2} is calculated twice, F_{n-3} thrice, F_{n-4} five times and so on... (F_{n-k} is calculated about F_k times). To speed up the process, we make use of the following observation: It is very easy to calculate F_n and F_{n-1} if we know the value of F_{n-1} and F_{n-2}. Consider the function $h(x, y) = (y, x + y)$ and observe that

$$h(F_{n-1}, F_n) = (F_n, F_n + F_{n-1}) = (F_n, F_{n+1})$$

Thus applying h to a pair of adjacent fibonacci numbers (thats what the F_n's are called), gives the next pair of adjacent fibonacci numbers. This is captured by the following scheme function.

```
;; Two adjacent fibonacci numbers
;; low < high
(define-struct fibpair (low high))

;; next-fib: fibpair -> fibpair
;;   given a fibpair produces the next fibpair
(define (next-fib fp)
    (make-fibpair (fibpair-high fp)
                  (+ (fibpair-high fp) (fibpair-low fp))
    )
)
```

Now in order to compute F_n, we just need to start with the pair (F_1, F_2) and apply the function `next-fib` $n - 1$ times. This is accomplished by the function below

```
;; applynextfib : number fibpair -> fibpair
;;   apply the function nextfib num number of times
(define (applynextfib num fp)
    (cond
        [(= n 0) fp)
```

```
            [else (applynextfib (- num 1) (next-fib fp))]
        )
    )
```

Applying `next-fib` $n-1$ times to (F_1, F_2) gives the pair (F_n, F_{n+1}). It only remains to extract F_n, as done by the scheme function below:

```
;; fib2: number -> number
;;   returns the n'th fibonacci number
(define (fib2 n)
    (fibpair-low (applynextfib (- n 1) (make-fibpair 1 2)))
)
```

This completes the solution. Another way to rewrite the same solution is as given below. Here the pair of number old and curr represent the low and high parts of fibpair.

```
(define (fibfast n old curr)
  (cond
    [(= n 0) curr]
    [else (fibfast (- n 1) curr (+ old curr))]
  )
)

(define (fib3 n)
  (cond
    [(< n 3) n]
    [else (fibfast (- n 2) 1 2)]
  )
)
```

Statistics

Let $a_1, \ldots, a_n$ be a sequence of numbers. The mean (denoted μ) and the Standard Deviation (denoted σ) are defined through the following formulae:

$$\mu := \frac{(a_1 + a_2 + \ldots + a_n)}{n}$$

$$\sigma := \sqrt{\frac{(a_1 - \mu)^2 + (a_2 - \mu)^2 + \ldots + (a_n - \mu)^2}{n}}$$

5. **(10 pts)** Write scheme function `mean` and `stddev` which consume lists of numbers and produce the mean and standard deviation of the input list respectively.

Solution

5. We need to write functions to calculate the length of the list, another to sum up all the numbers in the list, and a third one to calculate the sum of squares of the numbers minus a given number. Note how similar the function definitions will be if we write them!

   Instead of writing three different functions we write one function which does all three! The trick is to find what is common in all the three cases. What we need is to sum up $(x - base)^k$, where $base$ and k are fixed numbers and x varies through the list. Remember that $k = 0$ will give us 1 for each element of the list. Thus calculating the number of elements in the list. Putting $base = 0$ and $k = 1$ gives the sum of the numbers in the list. From these two we can calculate the mean. Finally putting $k = 2$, $base = mean$, we can calculate the standard deviation.

```
;; sum-power-deviation: list-of-numbers number number -> number
;;     calculate SUM ( (x - base)^k )
(define (sum-power-deviation lon base k)
  (cond
    [(empty? lon) 0]
    [else (+ (expt (- (first lon) base) k)
             (sum-power-deviation (rest lon) base k))]
  )
)

(define (count lon)
   (sum-power-deviation lon 0 0))

(define (mean lon)
   (cond
      [(empty? lon) 0]
      [else (/ (sum-power-deviation lon 0 1) (count lon))]
    )
)

(define (stddev lon)
  (cond
    [(empty? lon) 0]
    [else (sqrt (/ (sum-power-deviation lon (mean lon) 2) (count lon)))]
  )
)
```