CS 70 Spring 2005 — Discrete Mathematics for CS — Clancy/Wagner — HW 3

Due Thursday, February 10

Coverage: This assignment involves topics from the February 1 and 3 lectures and from sections 3.4 through 3.6 of Rosen.

Administrative reminders: We will accept only unformatted text files or PDF files for homework submission. Include your name, login name, section number, and partner list in your submission. Give the command `submit hw3` to submit your answers to this assignment.

Homework exercises:

1. **(14 pts.) Fibonacci numbers**
   The Fibonacci numbers are defined as follows:

$$\begin{aligned} F_0 &= 0 \\ F_1 &= 1 \\ F_n &= F_{n-2} + F_{n-1} \text{ for } n > 1 \end{aligned}$$

   (a) List the first ten Fibonacci numbers (F_0 through F_9).

   (b) Consider the following Scheme procedure that, given n, returns F_n.

```
(define (fib n)
  (cond
    ((= n 0) 0)
    ((= n 1) 1)
    (else (+ (fib (- n 1)) (fib (- n 2)))) ) )
```

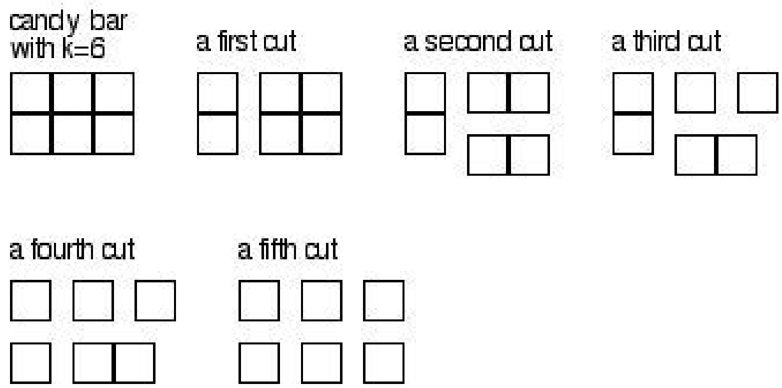
   Let T_n be the number of addition operations required in the computation of `(fib n)`. List the values T_0 through T_9.

   (c) State and prove a relationship between the T numbers and the F numbers.

   (d) Prove that $F_{n+1}F_{n-1} - F_n^2 = (-1)^n$.

2. (6 pts.) Chocolate chopping

Chocolate often comes in rectangular bars marked off into smaller squares. It is easy to break a larger rectangle into two smaller rectangles along any of the horizontal or vertical lines between the squares. The figure below shows an example.



Suppose I have a bar containing k squares and wish to break it down into its individual squares. Prove that *no matter which way I break it*, it will take exactly $k-1$ breaks to do this.

3. (17 pts.) A buggy search

A CS 70 student, attempting to write a guessing game, implements the following pseudocode.

print "Think of an integer between 1 and 100 (inclusive), and I will guess it.";
lowestPossible = 1;
highestPossible = 100;
while (we're not done)
    $mid = \lfloor (lowestPossible + highestPossible)/2 \rfloor$;
    print "Is it ", *mid*, "?";
    if user says yes, we're done
    otherwise
        print "Is it bigger than ", *mid*, "?";
        if user says yes, set *lowestPossible* to *mid*
        otherwise set *highestPossible* to *mid*;

(a) The CS 70 student is intending to maintain the following loop invariant: the number to be guessed is somewhere in the interval $[lowestPossible, highestPossible]$. Is this invariant maintained in each iteration of the loop? Explain why or why not.

(b) List all numbers between 1 and 100, inclusive, that the algorithm fails to guess, and describe what happens for each.

(c) Suppose we make two changes in the algorithm: first, initialize *highestPossible* to 101; second, update *lowestPossible* to $mid + 1$ if the user's value is bigger than *mid*. The modified code maintains a loop invariant that is stronger than the invariant described in part a. Describe the loop invariant in the modified code, and briefly explain your answer.

(d) Using your updated invariant, give an induction proof that the modified algorithm works correctly.

4. (8 pts.) Max-min computation

Given a set of N numeric values, we can find the maximum element using $N-1$ comparisons and then find the minimum element using $N-2$ more comparisons (skipping the maximum element), for a total of $2N-3$ comparisons. Consider the following algorithm:

if there are two elements in the set, then
    with one comparison determine the larger and the smaller,
        and store them in variables *larger* and *smaller*;
    return the pair $[larger, smaller]$;
otherwise
    divide the set of elements in half;
    make a recursive call to find the largest and smallest elements in the first half;
    make a recursive call to find the largest and smallest elements in the second half;
    set *larger* to the larger of the two largest elements (found with one comparison);
    set *smaller* is the smaller of the two smallest elements (found with one comparison);
    return the pair $[maximum, minimum]$;

(a) Prove that, for N a power of 2, that the above algorithm always requires less than $3N/2$ comparisons (ignoring whatever comparisons are necessary to divide the set in half).

(b) Identify the reason that this algorithm works faster than the straightforward method outlined above.

5. (15 pts.) Leaf analysis

A *rooted binary tree* either is empty, or consists of a *root* and zero, one, or two disjoint *subtrees* (which are also rooted binary trees). A *leaf* is a rooted binary tree with a root and no subtrees.

(a) Prove that, in a rooted binary tree with L leaves, that

$$\sum_{k=1}^{L} 2^{-d(k)} \leq 1$$

where $d(k)$ is the depth of leaf k. The depth of the root of a tree is 0, and the depth of any other node is 1 plus the depth of its parent.

(b) Completely describe all trees for which

$$\sum_{k=1}^{L} 2^{-d(k)} = 1$$

and prove your claim. This will involve two proofs, one that verifies that the trees you describe satisfy the equality, and one that shows that for all other trees, the given sum is less than 1.