Worksheet Fibonacci - Part III

Oliver Ray

Week 7

This worksheet continues the Fibonacci example. It asks you to write a very fast implementation by considering an alternative formulation that returns pairs (a, b) of consecutive Fibonacci numbers; and it invites you to find an upper bound on the number of arithmetic operations it performs.

Define the n'th pair of Fibonacci numbers as follows for all $n \geq 0$:

$$h(n) = \begin{cases} (0,1) & \text{if } n = 0 \\ (2ab - a^2, a^2 + b^2) \text{ where } (a,b) = h(n/2) & \text{if } n \text{ is even} \\ (b, a+b) \text{ where } (a,b) = h(n-1) & \text{if } n \text{ is odd} \end{cases}$$

1. Translate the above definition into a recursive function with prototype `Pair h(int n)` by defining a structured data type `Pair` that consists of a pair of `double`'s.

2. (Optional) Prove (by induction) that $h(n) = (f(n), f(n+1))$ for all $n \geq 0$ where $f(n)$ is the standard definition of Fibonacci numbers given two weeks ago. **Hint:** For the induction step use the odd/even characterisation of Fibonacci numbers given last week.

3. Write a function with prototype `double x(int n)` that returns the total number of *arithmetic* operations performed by `h` on an input `n`. **Note:** You are only asked to consider arithmetic operations (+,-,*,/).

4. Find the largest number in the Fibonacci series that can be computed by `h` (assuming that a `double` is represented by 8 bytes) and estimate how long the function actually takes to do so.

5. (Optional) Prove that $2 + 10\, log_2\, n$ is an upper bound on the number of arithmetic operations performed by `h`.

ANSWERS

1. ```
   typedef struct {double a,b;} Pair;

   Pair h(int n) {
       if (n==0) {
           return (Pair) {0,1};
       } else if ((n&1)==0) {
           Pair q=h(n/2);
           double a=q.a, b=q.b;
           return (Pair) {2*a*b-a*a,a*a+b*b};
       } else {
           Pair q=h(n-1);
           double a=q.a, b=q.b;
           return (Pair) {b,a+b};
       }
   }
   ```

2. Exercise for the reader.

3. ```
   double x(int n) {
       if (n==0) {
           return 0;
       } else if ((n&1)==0) {
           return 8+x(n/2); /* 5mul, 1add, 1sub, 1div */
       } else {
           return 2+x(n-1); /* 1add, 1sub */
       }
   }
   ```

4. The 1476'th Fibonacci number is the largest computed (when n=1475) before an overflow. On my laptop this was computed in 1 microsecond.

5. For any number n with $k+1$ bits, the worst case performance occurs when all the bits are set to 1 so that $n = 2^k - 1$. In this case `h` performs $8(k)+2(k+1)+0(1) = 2+10k$ arithmetic operations where $k = \lfloor log_2 n \rfloor$ and where 8, 2 and 0 are the number of operations performed in the even, odd and zero cases, respectively. Since $2 + 10 \lfloor log_2 n \rfloor$ is an upper bound on the number of operations, so must $2 + 10 \, log_2 n$ also be.