Computer Science 308-250B Homework #2
Due Monday February 9, 2004, 13:30

In all the problems,* <u>all calculations are done mod M</u> *where* M=3333373 *in order to limit the size of the integers involved. This way we avoid problems due to overflow.

As we have seen in class, the Fibonacci numbers are defined as follows:

$$f_n = \begin{cases} 1 & \text{if } n=1,2 \\ f_{n-1} + f_{n-2} & \text{if } n>2 \end{cases}$$

Thus the most natural way to write a program to compute one uses the next algorithm:

```
fibrec(n)
If (n<3) Then return 1
Else return fibrec (n-1)+ fibrec (n-2)
```

[15%] **•1)** Write a **Java** program that implements this algorithm. Find the time needed to compute $f_1,f_2,f_3,f_4,f_5,\ldots$ and plot a graph of your results. What is the largest n for which you can compute f_n within 1 second ?

No **Java** details provided.
The running time should be exponential. Indeed, the running time is $\Omega(f_n)$ as explained in class before.

An iterative way of computing the same values only keeps track of the last two values and use them to compute the new ones :

```
fibit(n)
f:=1; g:=1
For k:=2 to n do
        f:= f+g
        g:= f-g
Return f
```

[15%] **•2** Write a **Java** program that implements this algorithm. Find the time needed to compute $f_1, f_2, f_3, f_4, f_5, \ldots$ and plot a graph of your results. What is the largest n for which you can compute f_n within 1 second ?

No **Java** details provided.
The running time should be linear, i.e. O(n).

An alternate definition we have **NOT** seen in class for the Fibonacci numbers is:

$$f_n = \begin{cases} 1 & \textbf{if } n=1,2 \\ f^2_{n+\frac{1}{2}} + f^2_{n-\frac{1}{2}} & \textbf{if } n>2, \textbf{ n odd} \\ f^2_{\frac{n}{2}+1} - f^2_{\frac{n}{2}-1} & \textbf{if } n>2, \textbf{ n even} \end{cases}$$

[15%] **•3a)** Prove by mathematical induction that this new definition of f_n is correct.

basis case: n=1,2: $f_1 = f_2 = 1$

induction step: Let n>2 and assume the formulas are correct for all k, 0<k<n. Then

<u>notation: $F=f^2$</u>

n odd, (n-1 even, n-2 odd)

$$
\begin{aligned}
f_n &= f_{n-1}+f_{n-2} \\
&=_* [F_{(n-1)/2+1}-F_{(n-1)/2-1}]+[F_{(n-1)/2}+F_{(n-3)/2}] \\
&= F_{(n+1)/2}-F_{(n-3)/2}+F_{(n-1)/2}+F_{(n-3)/2} \\
&= F_{(n+1)/2}+F_{(n-1)/2}
\end{aligned}
$$

NOTE : "$=_*$" indicates use of the induction hypothesis. All other steps are simple algebraic manipulations.

n even, (n-1 odd, n-2 even)

$$
\begin{aligned}
f_n &= f_{n-1}+f_{n-2} \\
&=_* [F_{n/2}+F_{(n-2)/2}]+[F_{(n-2)/2+1}-F_{(n-2)/2-1}] \\
&= F_{n/2}+F_{n/2-1}+F_{n/2}-F_{n/2-2} \\
&=_* 2F_{n/2}+[f_{n/2}-f_{n/2-2}]^2-F_{n/2-2} \\
&= 2F_{n/2}+[F_{n/2}-2f_{n/2}f_{n/2-2}+F_{n/2-2}]-F_{n/2-2} \\
&= F_{n/2}+2F_{n/2}-2f_{n/2}f_{n/2-2} \\
&=_* f_{n/2}[f_{n/2}+2f_{n/2}-2f_{n/2-2}] \\
&= f_{n/2}[f_{n/2}+2f_{n/2-1}] \\
&= [f_{n/2+1}-f_{n/2-1}]\,[f_{n/2+1}+f_{n/2-1}] \\
&=_* F_{n/2+1}-F_{n/2-1}
\end{aligned}
$$

[15%] **•3b)** Write a **Java** program that implements this algorithm. Find the time needed to compute $f_1, f_2, f_3, f_4, f_5, \ldots$ and plot a graph of your results. What is the largest n for which you can compute f_n within 1 second ?

No **Java** details provided.
The running time should be linear, i.e. O(n).

[15%] **•4a)** Prove the following by mathematical induction for all **n>0**.

$$
\begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}^n = \begin{pmatrix} f_{n-1} & f_n \\ f_n & f_{n+1} \end{pmatrix}
$$

basis case: n=1,2 :

$$\begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}^1 = \begin{pmatrix} f_0 & f_1 \\ f_1 & f_2 \end{pmatrix}, \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}^2 = \begin{pmatrix} f_1 & f_2 \\ f_2 & f_3 \end{pmatrix}$$

induction step: Let n>2 and assume the formula is correct for n-1. Then

$$\begin{aligned}
\begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}^n &= \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}\begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}^{n-1} \\
&= * \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}\begin{pmatrix} f_{n-2} & f_{n-1} \\ f_{n-1} & f_n \end{pmatrix} \\
&= \begin{pmatrix} f_{n-1} & f_n \\ f_{n-2}+f_{n-1} & f_{n-1}+f_n \end{pmatrix} \\
&= \begin{pmatrix} f_{n-1} & f_n \\ f_n & f_{n+1} \end{pmatrix}
\end{aligned}$$

A natural way of computing this exponentiation is to use a method similar to the one we used in Homework #1 for numbers (below **I** stands for the 2 by 2 identity matrix):

```
ExpMOD(A,b)
If (b=0)
Then return I
Else  If (b mod 2)=0   Then return ExpMOD( A^2, b/2 )
                       Else return A( ExpMOD(A^2, (b-1)/2 ) )
```

[15%] **•4b)** Write a Java program that implements this algorithm. Find the time needed to compute $f_1, f_2, f_3, f_4, f_5, \ldots$ and plot a graph of your results. What is the largest n for which you can compute f_n within 1 second (give an estimate if n is too big…)?

No **Java** details provided.
The running time should be logarithmic, i.e. O(log n).

[10%] **•5)** If **M** is rather small (say M<1000) and you wish to compute the value $\mathbf{f_n \bmod M}$; find a <u>very</u> fast algorithm to compute this value, even for very large values of **n**.
hint: Show that there exists an integer R, $0<R<M^2$ such that $\mathbf{f_n \bmod M = f_{n \bmod R} \bmod M}$.

Since the value of $\mathbf{f_n \bmod M}$ is completely defined by $\mathbf{f_{n-1} \bmod M}$ and $\mathbf{f_{n-2} \bmod M}$ it is clear that whenever $\mathbf{f_{n-1} = f_{n+R-1} \bmod M}$ and $\mathbf{f_{n-2} = f_{n+R-2} \bmod M}$ then $\mathbf{f_n = f_{n+R} \bmod M}$. Therefore if we find a value **R** such that $\mathbf{f_{R+1} = 1\ (=f_1) \bmod M}$ and $\mathbf{f_{R+2} = 1\ (=f_2) \bmod M}$ then for any positive **n** we have $\mathbf{f_n = f_{n+R} \bmod M}$ and thus $\mathbf{f_n \bmod M = f_{n \bmod R} \bmod M}$.

All is left to prove is that there exists an integer $\mathbf{R,\ 0<R<M^2}$ such that $\mathbf{f_{R+1} = 1 \bmod M}$ and $\mathbf{f_{R+2} = 1 \bmod M}$. If we consider the sequence

$$\mathbf{f_1 \bmod M,\ f_2 \bmod M,\ f_3 \bmod M,\ \ldots,\ f_{M^2} \bmod M,\ f_{M^2+1} \bmod M,\ f_{M^2+2} \bmod M}$$

we enumerate $\mathbf{M^2+1}$ pairs $\mathbf{(f_i \bmod M,\ f_{i+1} \bmod M)}$ of consecutive values of the Fibonacci sequence. Now since only $\mathbf{M^2}$ such pairs **(x,y)** exist, there must exist an **i** and a **j** such that $\mathbf{(f_i \bmod M,\ f_{i+1} \bmod M) = (f_j \bmod M,\ f_{j+1} \bmod M)}$. Let $\mathbf{R=i-j<M^2}$. Notice that if we start with $\mathbf{(f_i \bmod M,\ f_{i+1} \bmod M) = (f_j \bmod M,\ f_{j+1} \bmod M)}$ then $\mathbf{f_{i-1} \bmod M} = \mathbf{f_{j-1} \bmod M}$ as well because $\mathbf{f_{n-1} = f_{n+1} - f_n}$. We obtain $\mathbf{(f_i \bmod M,\ f_{i+1} \bmod M) = (f_j \bmod M,\ f_{j+1} \bmod M)}$ which means that $\mathbf{(f_i \bmod M,\ f_{i+1} \bmod M) = (f_{i+R} \bmod M,\ f_{i+R+1} \bmod M)}$ which implies that $\mathbf{(f_{i-1} \bmod M,\ f_i \bmod M) = (f_{i+R-1} \bmod M,\ f_{i+R} \bmod M)}$ which implies that $\mathbf{(f_{i-2} \bmod M,\ f_{i-1} \bmod M) = (f_{i+R-2} \bmod M,\ f_{i+R-1} \bmod M)}$ which implies by induction that

$$\mathbf{(f_1 \bmod M,\ f_2 \bmod M) = (f_{R+1} \bmod M,\ f_{R+2} \bmod M).}$$

Pre-processing:

```
FindR(M)
f:=1; g:=2; R:=1
While NOT (f=1 and g=1) do
        f:= f+g mod M
        g:= f-g mod M
        R:=R+1
Return R
```

Given constant R:

```
FastFib(n)
Return(Fibit(n mod R))
```