Fibonacci Numbers with Matrices

Rushi Shah

9-11-2015

Ahh, the Fibonacci numbers. What mathemetician doesn't love them?

Well, in Week 06 of CIS194, some interesting implementations were discussed. My favorite (that I never actually had encountered before), was in order to get the `n`'th number, you raise a two by two matrix to the `n`'th power.

Let's take a look at my implementation:

0.1 The Matrix

First off, you need to be able to represent matrices. I decided to use a tuple of tuples for the two by two matrix.

```
data Matrix =    Matrix ((Integer, Integer),
                         (Integer, Integer))
```

I also wanted to be able to print them nicely in the terminal, so I whipped up a quick show function. I could have derived it, but in my opinion, this makes it look slightly nicer (sorry, the function is a bit long, so the text wraps).

```
instance Show Matrix where
        show (Matrix ((a, b), (c, d))) =
                "[" ++ show a ++ ", " ++ show b ++ "]"
                "[" ++ show c ++ ", " ++ show d ++ "]"
```

And now let's instantiate a matrix!

```
m :: Matrix
m = Matrix ((1, 1), (1, 0))
```

To check that it works, let's print out the matrix in `ghci`:

```
> m
[1, 1]
[1, 0]
```

0.2 Multiplying Matrices

So that's great, but these matrices don't really do much. We need to be able to raise each matrix to a specific power, but who knows how to do that? I sure don't. With that being

said, I do know how to multiply two 2x2 matrices together! Let's define a function `(*)` that takes two 2x2 matrices and returns a matrix representing the multiplication of the two arguments. This multiplication function is a part of the `Num` typeclass, so in essence, we are making `Matrix` an instance of `Num`.

```
instance Num Matrix where
        (*) (Matrix ((a, b), (c, d))) (Matrix ((e, f), (g, h))) = Matrix (
                        ((a*e + b*g), (a*f + b*h)),
                        ((c*e + d*g), (c*f + d*h))
                )
```

You can raise any instance of `Num` to a power after defining the multiplication operator, so Haskell will take care of the rest.

0.3 Quick helper function

The last element of a matrix will represent the Fibonacci number you're looking for. So let's whip up a quick function to get that element.

```
l :: Matrix -> Integer
l (Matrix m) = (snd . snd) m
```

0.4 Finally, the Fibonacci Function!

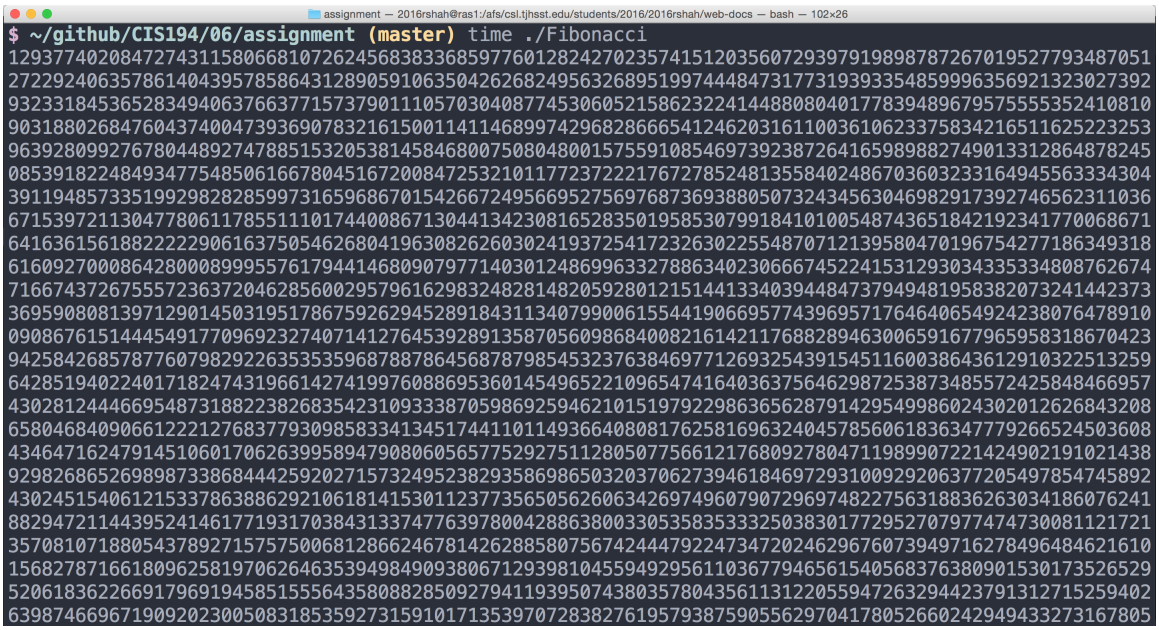
In CIS194, this is the fourth version of the function, so it is named `fib4`. Essentially, you take a number `n` and return the `n`th Fibonacci number by raising a 2x2 matrix to the `n`th power. Note that raising the matrix to the `0`th power won't work, so we'll use pattern-matching to account for that special case.

```
fib4 :: Integer -> Integer
fib4 0 = 0
fib4 n = l (f^n)
```

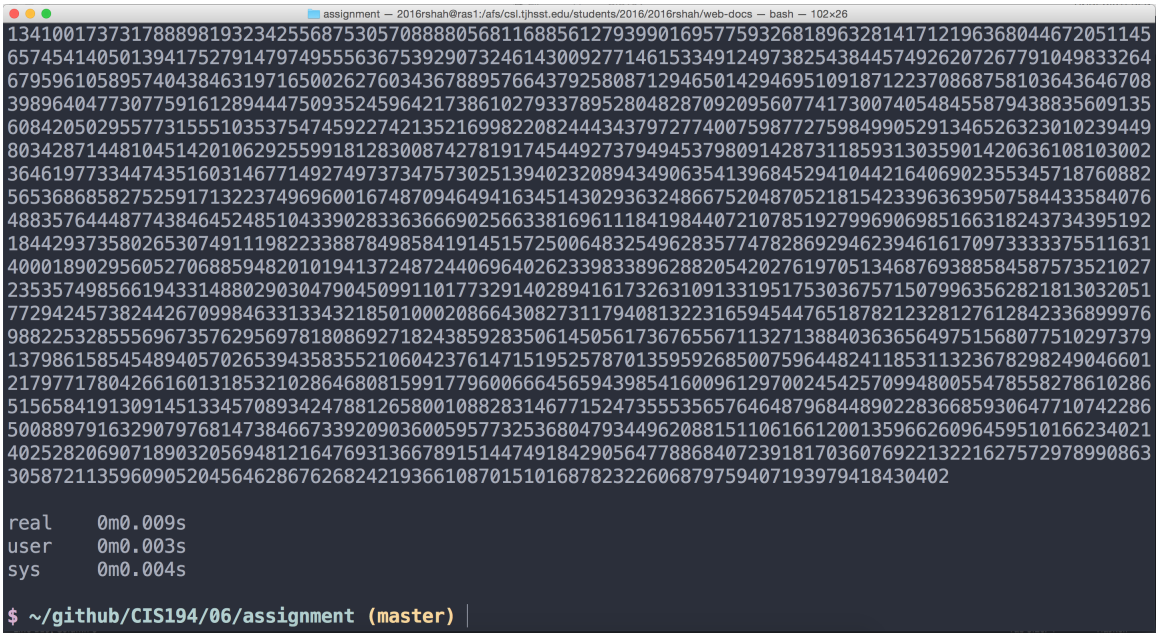
0.5 Conclusion

To conclude, let's try it out!

What's an insanely large Fibonacci number? Well my birthday is April 13th, 1998, so how about we calculate the `41398`th Fibonacci? That'll take a while, right? Wrong.



. . .



That's right, the answer is a 8652 digit number, and was calculated in about .009 seconds. If you want to see the answer, check out this .txt file.