Fibonacci Numbers with Matrices

Rushi Shah

11 September 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:

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.

```
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]
```

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.

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
```

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)
```

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.

```
assignment — 2016rshah@ras1:/afs/csl.tjhsst.edu/students/2016/2016rshah/web-docs — bash — 102×26
$ ~/github/CIS194/06/assignment (master) time ./Fibonacci
129377402084727431158066810726245683833685977601282427023574151203560729397919898787267019527793487051
272292406357861404395785864312890591063504262682495632689519974448473177319393354859996356921323027392
932331845365283494063766377157379011105703040877453060521586232241448808040177839489679575555352410810
903188026847604374004739369078321615001141146899742968286665412462031611003610623375834216511625223253
963928099276780448927478851532053814584680075080480015755910854697392387264165989882749013312864878245
085391822484934775485061667804516720084725321011772372221767278524813558402486703603233164945563334304
391194857335199298282859973165968670154266724956695275697687369388050732434563046982917392746562311036
671539721130477806117855111017440086713044134230816528350195853079918410100548743651842192341770068671
641636156188222229061637505462680419630826260302419372541723263022554870712139580470196754277186349318
616092700086428000899955761794414680907977140301248699633278863402306667452241531293034335334808762674
716674372675557236372046285600295796162983248281482059280121514413340394484737949481958382073241442373
369590808139712901450319517867592629452891843113407990061554419066957743969571764640654924238076478910
090867615144454917709692327407141276453928913587056098684008216142117688289463006591677965958318670423
942584268578776079829226353535968788786456878798545323763846977126932543915451160038643612910322513259
642851940224017182474319661427419976088695360145496522109654741640363756462987253873485572425848466957
430281244466954873188223826835423109333870598692594621015197922986365628791429549986024302012626843208
658046840906612221276837793098583341345174411011493664080817625816963240457856061836347779266524503608
434647162479145106017062639958947908060565775292751128050775661217680927804711989907221424902191021438
929826865269898733868444259202715732495238293586986503203706273946184697293100929206377205497854745892
430245154061215337863886292106181415301123773565056260634269749607907296974822756318836263034186076241
882947211443952414617719317038431337477639780042886380033053583533325038301772952707977474730081121721
357081071880543789271575750068128662467814262885807567424447922473472024629676073949716278496484621610
156827871661809625819706264635394984909380671293981045594929561103677946561540568376380901530173526529
520618362266917969194585155564358088285092794119395074380357804356113122055947263294423791312715259402
639874669671909202300508318535927315910171353970728382761957938759055629704178052660242949433273167805
```

. . .

```
assignment — 2016rshah@ras1:/afs/csl.tjhsst.edu/students/2016/2016rshah/web-docs — bash — 102×26
134100173731788898193234255687530570888805681168856127939901695775932681896328141712196368044672051145
657454140501394175279147974955563675392907324614300927714615334912497382543844574926207267791049833264
679596105895740438463197165002627603436788957664379258087129465014294695109187122370868758103643646708
398964047730775916128944475093524596421738610279337895280482870920956077417300740548455879438835609135
608420502955773155510353754745922742135216998220824443437972774007598772759849905291346526323010239449
803428714481045142010629255991812830087427819174544927379494537980914287311859313035901420636108103002
364619773344743516031467714927497373475730251394023208943490635413968452941044216406902355345718760882
565368685827525917132237496960016748709464941634514302936324866752048705218154233963639507584433584076
488357644487743846452485104339028336366690256633816961118419844072107851927996906985166318243734395192
184429373580265307491119822338878498584191451572500648325496283577478286929462394616170973333375511631
400018902956052706885948201019413724872440696402623398338962882054202761970513468769388584587573521027
235357498566194331488029030479045099110177329140289416173263109133195175303675715079963562821813032051
772942457382442670998463313343218501000208664308273117940813223165945447651878212328127612842336899976
988225328555696735762956978180869271824385928350614505617367655671132713884036365649751568077510297379
137986158545489405702653943583552106042376147151952578701359592685007596448241185311323678298249046601
217977178042661601318532102864680815991779600666456594398541600961297002454257099480055478558278610286
515658419130914513345708934247881265800108828314677152473555356576464879684489022836685930647710742286
500889791632907976814738466733920903600595773253680479344962088151106166120013596626096459510166234021
402528206907189032056948121647693136678915144749184290564778868407239181703607692213221627572978990863
305872113596090520456462867626824219366108701510168782322606879759407193979418430402

real	0m0.009s
user	0m0.003s
sys	0m0.004s

$ ~/github/CIS194/06/assignment (master)
```

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.