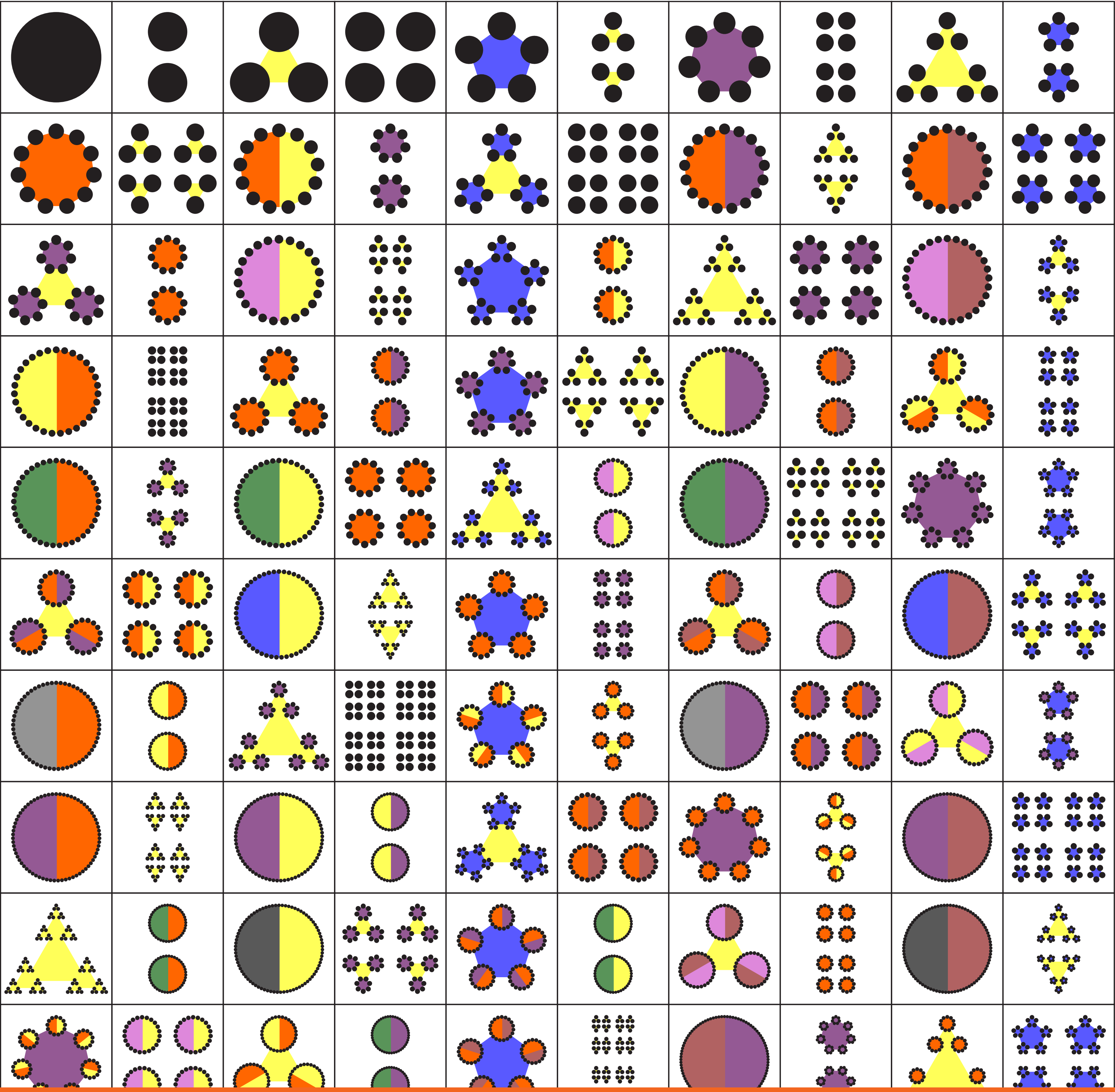
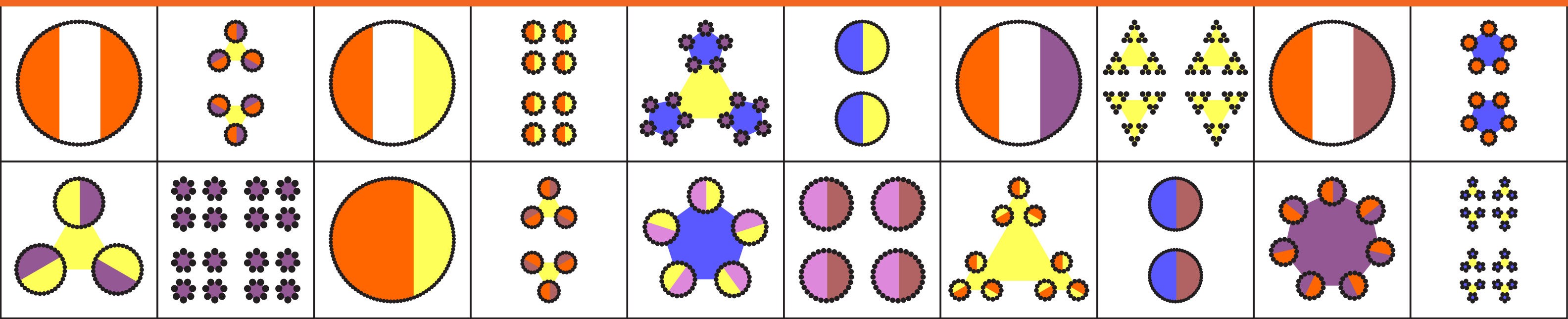
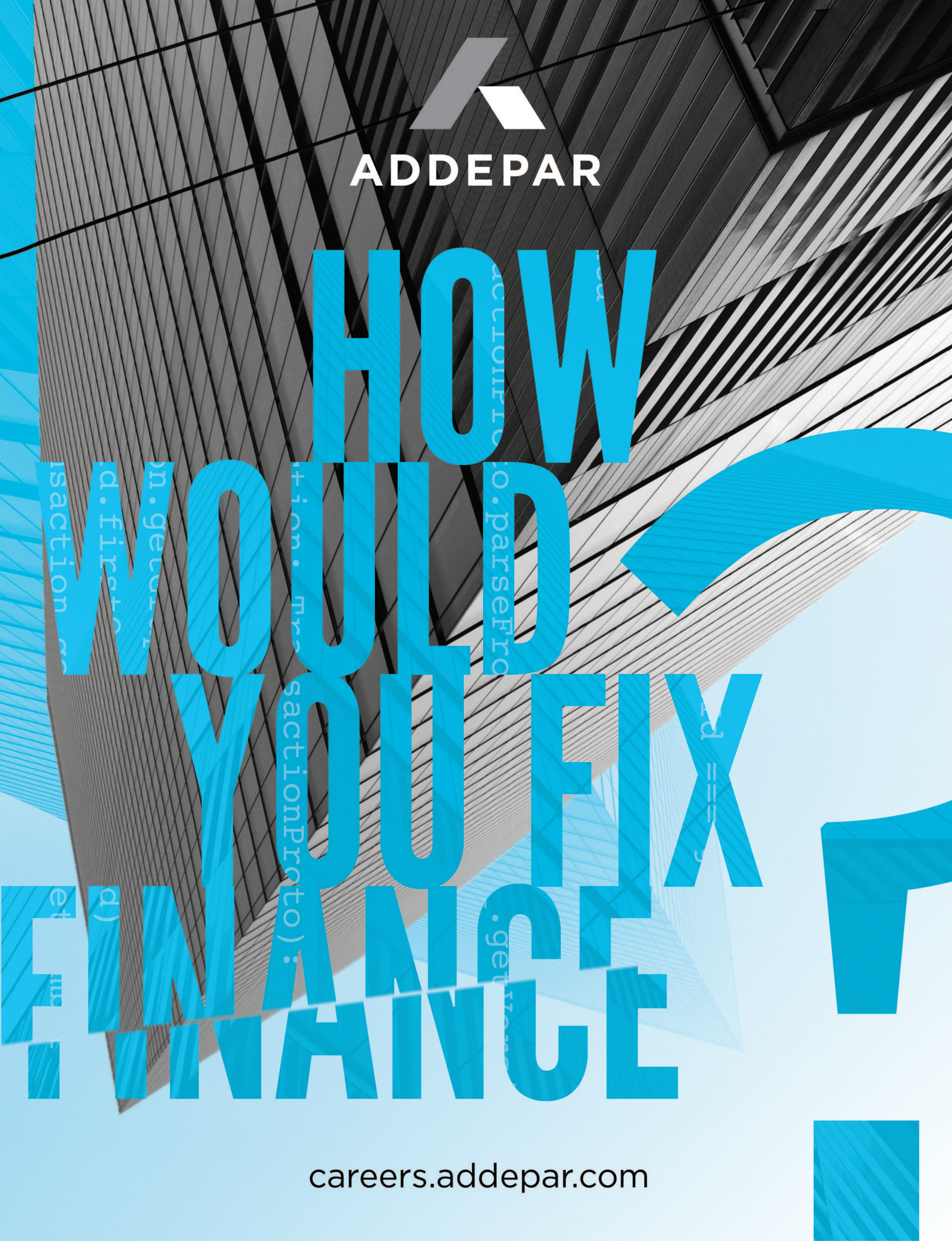


HACKERMONTHLY

Issue 31 December 2012



ADDEPAR



careers.addepar.com

Now you can hack on DuckDuckGo

DuckDuckHack

Create instant answer plugins for DuckDuckGo

duckduckhack.com

Curator
Lim Cheng Soon

Contributors
Brent Yorgey
Jesse Spaulding
David Hauser
Zachary Voase
Senthil Arivudainambi
Nick Johnson
Wesley Darlington
Pablo Rivera
Joel Gascoigne
Geoffroy Tremblay

Proofreaders
Emily Griffin
Sigmarie Soto

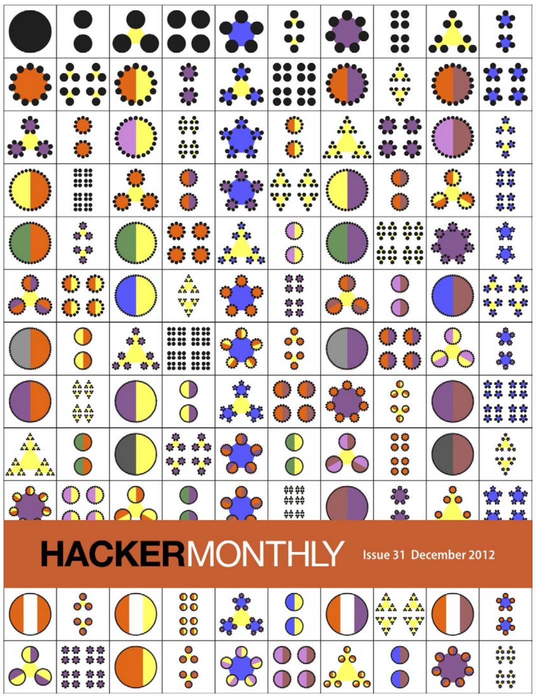
Printer
MagCloud

HACKER MONTHLY is the print magazine version of Hacker News — *news.ycombinator.com*, a social news website wildly popular among programmers and startup founders. The submission guidelines state that content can be "anything that gratifies one's intellectual curiosity." Every month, we select from the top voted articles on Hacker News and print them in magazine format. For more, visit *hackermonthly.com*

Advertising
ads@hackermonthly.com

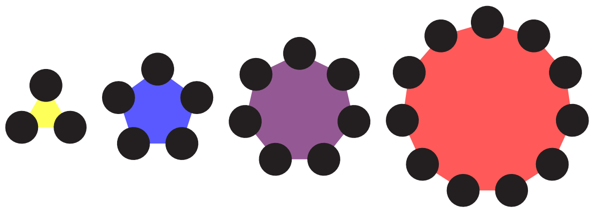
Contact
contact@hackermonthly.com

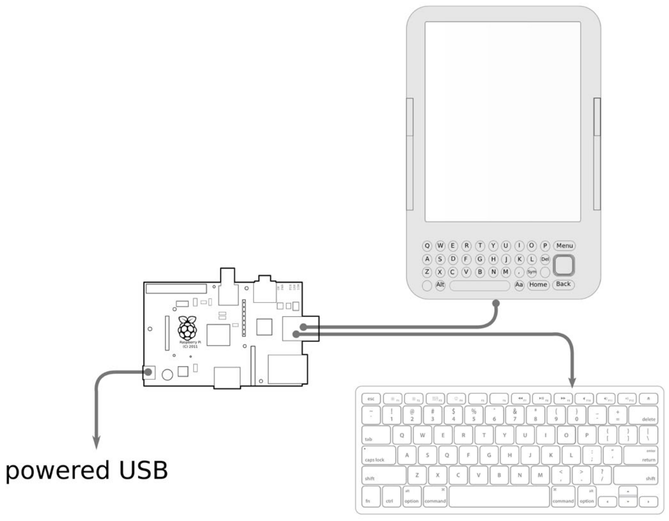
Published by
Netizens Media
46, Taylor Road,
11600 Penang,
Malaysia.



Cover "Factorization Diagrams" by Brent Yorgey

Contents



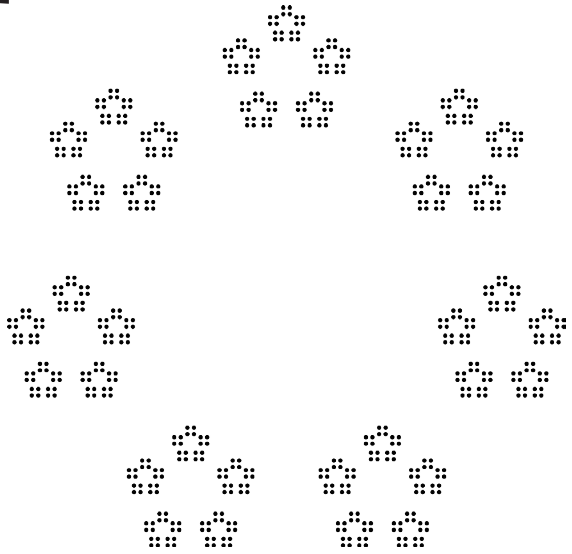


For links to Hacker News dicussions, visit *hackermonthly.com/issue-31*

Factorization Diagrams

By BRENT YORGEY

IN AN IDLE moment a while ago I wrote a program to generate "factorization diagrams." Here's 700:



It's easy to see, just by looking at the arrangement of dots, that there are `7 x 5 x 5 x 2 x 2` in total.

Here's how I did it. First, a few imports: a function to do factorization of integers and a library to draw pictures [projects.haskell.org/diagrams].

```
module Factorization where

import Math.NumberTheory.Primes.Factorisation
(factorise)

import Diagrams.Prelude
import Diagrams.Backend.Cairo.CmdLine

type Picture = Diagram Cairo R2
```

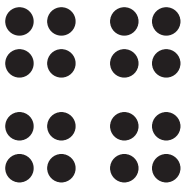
The `primeLayout` function takes an integer `n` (assumed to be a prime number) and some sort of picture, and symmetrically arranges `n` copies of the picture.

```
primeLayout :: Integer -> Picture -> Picture
```

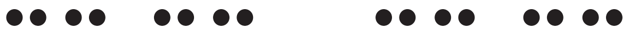
There is a special case for 2: if the picture is wider than tall, then we put the two copies one above the other; otherwise, we put them next to each other. In both cases we also add some space in between the copies (equal to half the height or width, respectively).

```
primeLayout 2 d
  | width d > height d
      = d === strutY (height d / 2) === d
  | otherwise
      = d ||| strutX (width d / 2)  ||| d
```

This means that when there are multiple factors of two and we call `primeLayout` repeatedly, we end up with things like:



If we always put the two copies next to each other, we would get:

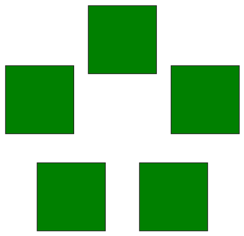


which is much clunkier and harder to understand at a glance.

For other primes, we create a regular polygon of the appropriate size (using some trig I worked out on a napkin; don't ask me to explain it) and position copies of the picture at the polygon's vertices.

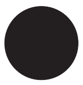
```
primeLayout p d = decoratePath pts (repeat d)
  where pts = polygon with
      { polyType = PolyRegular (fromIntegral p) r
       , polyOrient = OrientH
      }
    w = max (width d) (height d)
    r = w * c / sin (tau / (2 * fromIntegral p))
    c = 0.75
```

For example, here's `primeLayout` 5 applied to a green square:



Now, given a list of prime factors, we recursively generate an entire picture. First, if the list of prime factors is empty, that represents the number 1, so we just draw a black dot.

```
factorDiagram' :: [Integer] -> Diagram Cairo R2
factorDiagram' []     = circle 1 # fc black
```



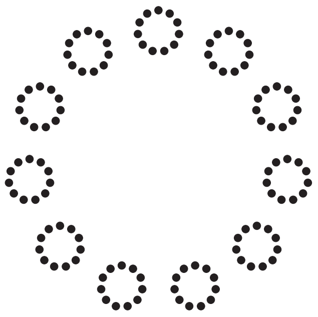
Otherwise, if the first prime is called `p` and the rest are `ps`, we recursively generate a picture from the rest of the primes `ps`, and then lay out `p` copies of that picture using the `primeLayout` function.

```
factorDiagram' (p:ps) = primeLayout p
       (factorDiagram' ps) # centerXY
```

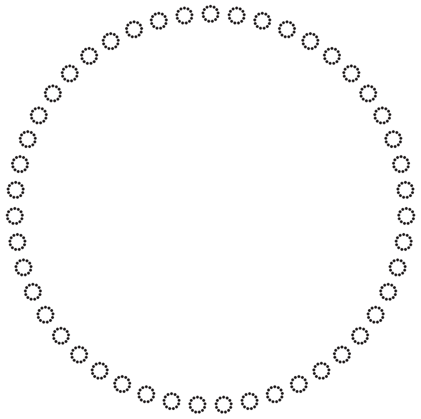
Finally, to turn a number into its factorization diagram, we factorize it, normalize the returned factorization into a list of primes, reverse it so the bigger primes come first, and call `factorDiagram'`.

```
factorDiagram :: Integer -> Diagram Cairo R2
factorDiagram = factorDiagram'
             . reverse
             . concatMap (uncurry $ flip replicate)
             . factorise
```

And voila! Of course, this really only works well for numbers with prime factors drawn from the set `{2, 3, 5, 7}` (and perhaps `11`). For example, here's 121:

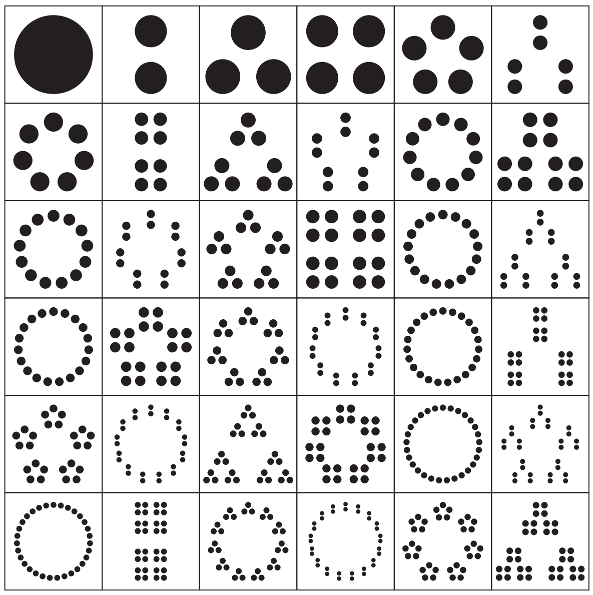


Are there 11 dots in those circles? 13? I can't really tell at a glance. And here's 611:

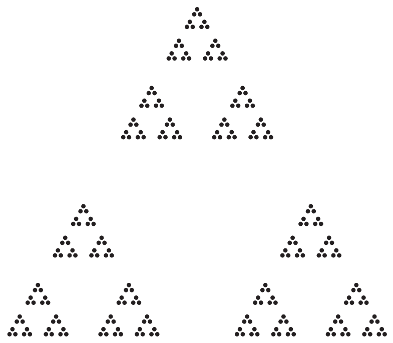


Uhh… well, at least it's pretty!

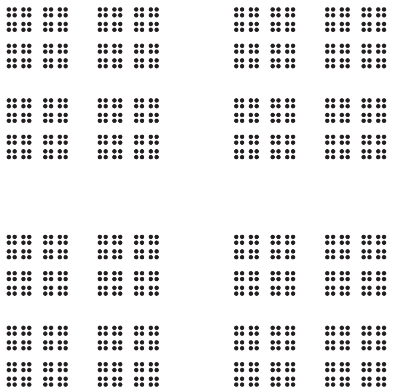
Here are the factorization diagrams for all the integers from 1 to 36:



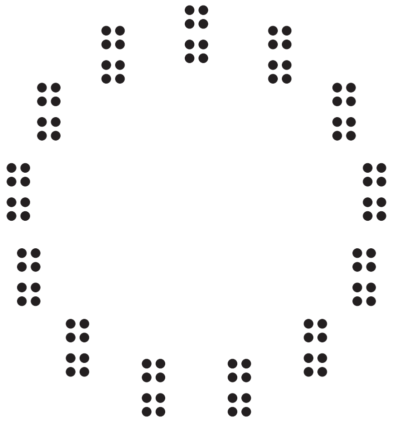
Powers of three are especially fun, since their factorization diagrams are Sierpinski triangles [hn.my/stri]! For example, here's `3⁵ = 243`:



Powers of two are also fun. Here's `2¹⁰ = 1024`:



One last one: 104.



Improved Diagrams

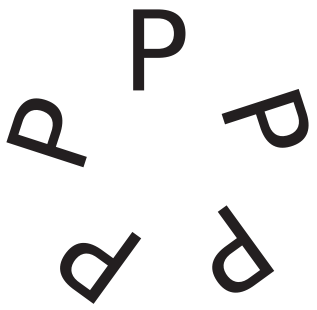
Now, on to the improved diagrams! Quite a few people suggested improvements, some of which I've adopted here.

```
 module Factorization2 where

 import Math.NumberTheory.Primes.Factorisation
(factorise)
 import Diagrams.Prelude
 import Diagrams.Backend.Cairo.CmdLine
 import Data.List.Split  ( chunksOf   )
 import Data.Char        ( digitToInt )

 type Picture = Diagram Cairo R2

 primeLayout :: Integer -> Picture -> Picture
```

Layout for 2 is mostly the same as before, except I've moved the diagrams a bit closer together, and added some extra calls to things like `reflectY` and `centerX` as preparation for the first improvement…

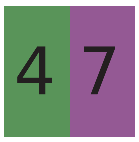
```
primeLayout 2 d
   | width d >= height d = (d === strutY (height
d / 3) === d # reflectY) # centerY
   | otherwise
       = (d ||| strutX (width d / 3)  ||| d)
         # centerX
```

…which is to lay out rotated copies of a subdiagram at the vertices of a *p*-sided polygon, as suggested by Mark Lentczner. This makes the diagrams much more symmetric and aesthetically pleasing.



```
primeLayout p d = (mconcat $
                  iterateN (fromIntegral p)
                    (rotateBy (1/fromIntegral p))
                    (d # translateY r)
                  )
```

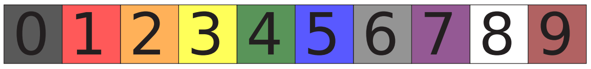
The next improvement is to add color (as suggested by Sjoerd Visscher). Instead of coloring the dots (which might look nice but doesn't really help visually identify the factorization), we color in the area inside each prime polygon, using a different color scheme for each prime. Actually, there's only a different color for each digit, and multi-digit primes are shown using vertical bars of color.



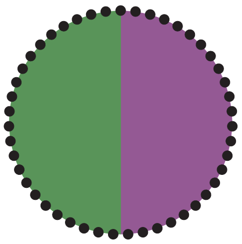
```
                  <>
                  colorBars p poly
where poly = polygon with
    { polyType = PolyRegular (fromIntegral p) r
      , polyOrient = OrientH
    }
  w   = max (width d) (height d)
  r   = w * c / sin (tau / (2 * fromIntegral p))
  c   = 0.75
```

And here are the colors I chose. They're based very loosely on the color code for resistors. I lightened them all up a bit (by blending with white), which I think looks nice.

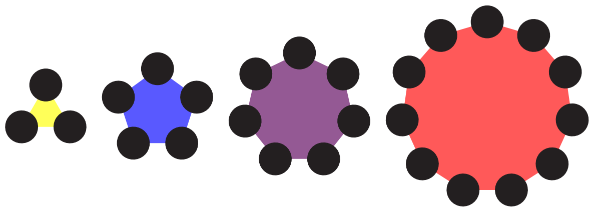
```
colors = map (blend 0.1 white)
[black,red,orange,yellow,green,blue,gray,purple,
white,brown]
```



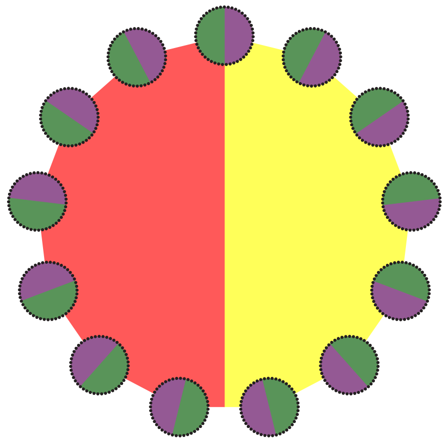
For example, here's the diagram for 47:



The odd primes up through each get their own color:



And here's the diagram for `611`:



Referring to the table of colors (and it's not hard to memorize), we can see that `611 = 13 x 47`.

For completeness, here's the implementation of `colorBars`:

```
colorBars p poly | p <= 11 = stroke poly
       # fc (colors!!(fromIntegral p `mod` 10))
       # lw 0
colorBars p poly = bars # clipBy poly
  where
    barColors = map ((colors!!) . digitToInt)
      (show p)
    barW = width poly / fromIntegral
      (length barColors)
    barH = height poly
    bars = (hcat $ map (\c -> rect barW barH
      # fc c # lc c) barColors) # centerX
```

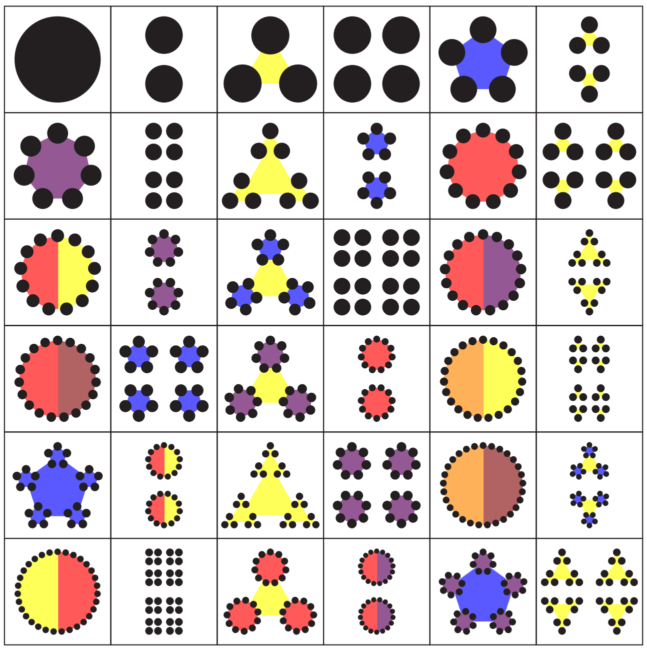
The code to actually generate complete factorization diagrams is almost the same as before, except I don't reverse the factors anymore. I think drawing the factors from largest to smallest actually gives more pleasing (and intelligible) diagrams.

```
factorDiagram' :: [Integer] -> Picture
factorDiagram' []     = circle 1 # fc black
factorDiagram' (p:ps) = primeLayout p
  (factorDiagram' ps)

factorDiagram :: Integer -> Picture
factorDiagram = centerXY
              . factorDiagram'
              . concatMap
                (uncurry $ flip replicate)
              . factorise
```

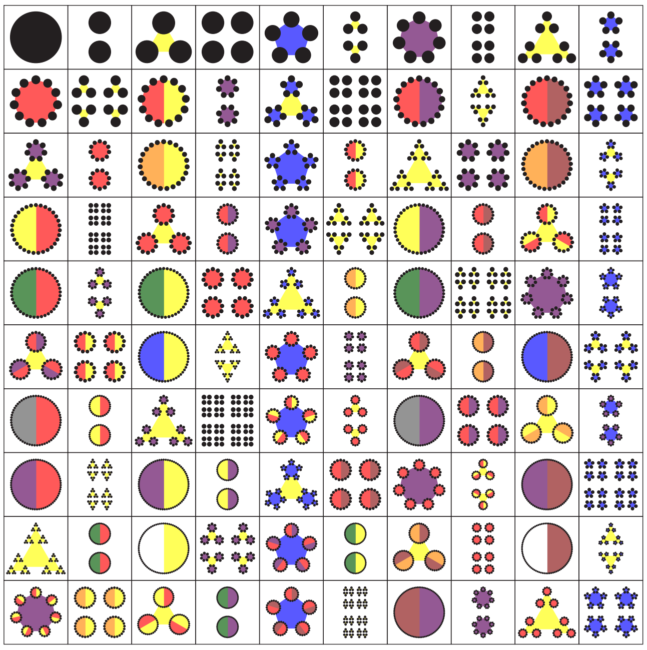
To show it off, here's some code for generating a table of factorization diagrams from *1* to *n*:

```
fd n = factorDiagram n # scaleUToY 0.8 <> square 1
table n = vcat . map hcat . (map . map) fd
        . chunksOf (fromIntegral n) $ [1..n*n]
```

Here's 1 to 36 as before:



And here's 1 to 100:



Sweet! ■

Brent Yorgey is a PhD student studying programming languages at the University of Pennsylvania in Philadelphia. He enjoys creating beauty and explaining mind-bending things to the uninitiated.

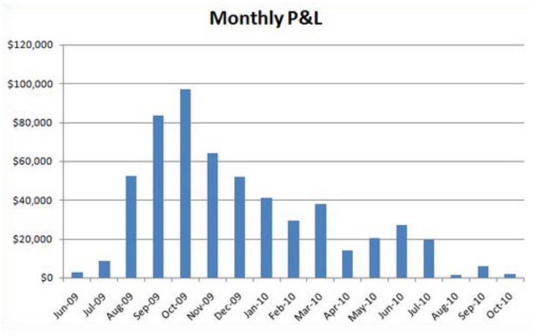
How I Made $500K with Machine Learning and HFT

By JESSE SPAULDING

THIS ARTICLE WILL detail what I did to make approximately $500K from high frequency trading from 2009 to 2010. Since I was trading completely independently and am no longer running my program, I'm happy to tell all. My trading was mostly in Russel 2000 and DAX futures contracts.

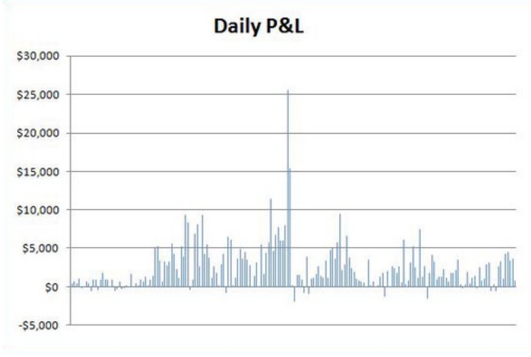
The key to my success, I believe, was not because of a sophisticated financial equation but rather the overall algorithm design that tied together many simple components and used machine learning to optimize for maximum profitability. You won't need to know any sophisticated terminology here because when I set up my program it was all based on intuition.

First, I just want to demonstrate that my success was not simply the result of luck. My program made 1000-4000 trades per day (half long, half short) and never got into positions of more than a few contracts at a time. This meant the random luck from any one particular trade averaged out pretty fast. The result was I never lost more than $2K in one day and never had a losing month:



(These figures are after paying commissions.)

Below is also a chart to give you a sense of the daily variation. Note that this excludes the last 7 months because I lost my motivation to enter them as the figures stopped going up.



My trading background

Prior to setting up my automated trading program, I had 2 years of experience as a "manual" day trader. This was back in 2001 — it was the early days of electronic trading and there were opportunities for "scalpers" to make good money. I can only describe what I was doing as akin to playing a video game/gambling with a supposed edge. Being successful meant being fast, being disciplined, and having good intuitive pattern recognition abilities. I was able to make around $250K, pay off my student loans, and have money left over. Win!

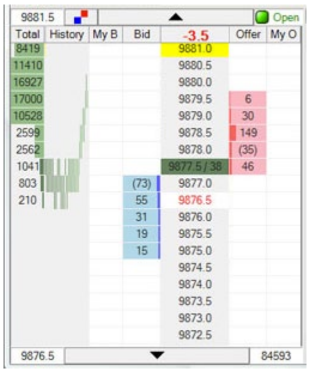
Over the next five years, I launched two startups, picking up some programming skills along the way. It wasn't until late 2008 that I got back into trading. With money running low from the sale of my first startup, trading offered hopes of some quick cash while I figured out my next move.

A trading API

In 2008 I was "manually" day trading futures using software called T4. I wanted some customized order entry hotkeys, so after discovering T4 had an API, I took on the challenge of learning C# (the programming language required to use the API) and went ahead and built myself some hotkeys.

After getting my feet wet with the API, I soon had bigger aspirations: I wanted to teach the computer to trade for me. The API provided both a stream of market data and an easy way to send orders to the exchange — all I had to do was create the logic in the middle.

Below is a screenshot of a T4 trading window. What was cool is that when I got my program working I was able to watch the computer trade on this exact same interface. Watching real orders popping in and out (by themselves with my real money) was both thrilling and scary.



9881.5 | Open

Total	History	My B	Bid	-3.5	Offer	My O
8419				9881.0		
11410				9880.5		
16927				9880.0		
17000				9879.5	6	
10528				9879.0	30	
2599				9878.5	149	
2562				9878.0	(35)	
1041				9877.5 / 38	46	
803			(73)	9877.0		
210			55	9876.5		
			31	9876.0		
			19	9875.5		
			15	9875.0		
				9874.5		
				9874.0		
				9873.5		
				9873.0		
				9872.5		

9876.5 | 84593

The design of my algorithm

From the outset, my goal was to setup a system such that I could be reasonably confident I'd make money before ever making any live trades. To accomplish this, I needed to build a trading simulation framework that would — as accurately as possible — simulate live trading.

While trading in live mode required processing market updates streamed through the API, simulation mode required reading market updates from a data file. To collect this data I setup the first version of my program to simply connect to the API and record market updates with timestamps. I ended up using 4 weeks' worth of recent market data to train and test my system.

With a basic framework in place I still had the task of figuring out how to make a profitable trading system. As it turns out my algorithm would break down into two distinct components, which I'll explore in turn:

- Predicting price movements; and
- Making profitable trades

Predicting price movements

Perhaps an obvious component of any trading system is being able to predict where prices will move, and mine was no exception. I defined the current price as the average of the inside bid and inside offer, and I set the goal of prediction to where the price would be in the next 10 seconds. My algorithm would need to come up with this prediction moment-by-moment throughout the trading day.

Creating & optimizing indicators

I created a handful of indicators that proved to have a meaningful ability to predict short-term price movements. Each indicator produced a number that was either positive or negative. An indicator was useful if more often than not a positive number corresponded with the market going up and a negative number corresponded with the market going down.

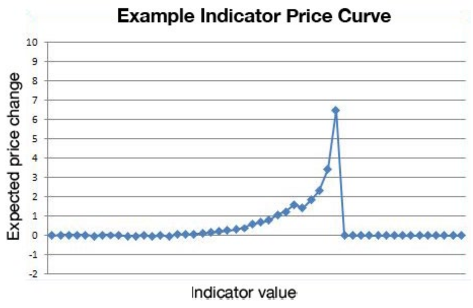
My system allowed me to quickly determine how much predictive ability any indicator had, so I was able to experiment with a lot of different indicators to see what worked. Many of the indicators had variables in the formulas that produced them, and I was able to find the optimal values for those variables by doing side by side comparisons of results achieved with varying values.

The indicators that were most useful were all relatively simple and were based on recent events in the market I was trading in as well as the markets of correlated securities.

Making exact price move predictions

Having indicators that simply predicted an up or down price movement wasn't enough. I needed to know exactly how much price movement was predicted by each possible value of each indicator. I needed a formula that would convert an indicator value to a price prediction.

To accomplish this, I tracked predicted price moves in 50 buckets that depended on the range the indicator value fell in. This produced unique predictions for each bucket that I was then able to graph in Excel. As you can see, the expected price change increases as the indicator value increases.



Based on a graph such as this I was able to make a formula to fit the curve. In the beginning I did this "curve fitting" manually, but I soon wrote up some code to automate this process.

Note that not all the indicator curves had the same shape. Also note the buckets were logarithmically distributed so as to spread the data points out evenly. Finally note that negative indicator values (and their corresponding downward price predictions) were flipped and combined with the positive values. (My algorithm treated up and down exactly the same.)

Combining indicators for a single prediction

An important thing to consider was that each indicator was not entirely independent. I couldn't simply just add up all the predictions each indicator made individually. The key was to figure out the additional predictive value each indicator had beyond what was already predicted. This wasn't too hard to implement, but it did mean that if I was "curve fitting" multiple indicators at the same time I had to be careful; changing one would affect the predictions of another.

In order to "curve fit" all of the indicators at the same time I setup the optimizer to step only 30% of the way towards the new prediction curves with each pass. With this 30% jump I found that the prediction curves would stabilize within a few passes.

With each indicator now giving us its additional price prediction, I could simply add them up to produce a single prediction of where the market would be in 10 seconds.

Why predicting prices is not enough

You might think that with this edge on the market I was golden, but you need to keep in mind that the market is made up of bids and offers — it's not just one market price. Success in high frequency trading comes down to getting good prices and it's not that easy.

The following factors make creating a profitable system difficult:

- With each trade I had to pay commissions to both my broker and the exchange.
- The spread (difference between highest bid and lowest offer) meant that if I were to simply buy and sell randomly I'd be losing a ton of money.
- Most of the market volume was comprised of other bots that would only execute a trade with me if they thought they had some statistical edge.
- Seeing an offer did not guarantee that I could buy it. By the time my buy order got to the exchange it was very possible that the offer would have been cancelled.
- As a small market player there was no way I could compete on speed alone.

Building a full trading simulation

I had a framework that allowed me to backtest and optimize indicators, but I had to go beyond this. I needed a framework that would allow me to backtest and optimize a full trading system — one where I was sending orders and getting in positions. In this case, I'd be optimizing for total P&L and to some extent average P&L per trade.

This would be trickier and in some ways impossible to model exactly, but I did as best as I could. Here are some of the issues I had to deal with:

- When an order was sent to the market in simulation I had to model the lag time. The fact that my system saw an offer did not mean that it could buy it straight away. The system would send the order, wait approximately 20 milliseconds, and then consider it an executed trade only if the offer was still there. This was inexact because the real lag time was inconsistent and unreported.
- When I placed bids or offers I had to look at the trade execution stream (provided by the API) and use that to gauge when my order would have gotten executed against. To do this right, I had to track the position of my order in the queue. (It's a first-in first-out system.) Again, I couldn't do this perfectly, but I made a best approximation.

To refine my order execution simulation, I took my log files from live trading through the API and compared them to log files produced by simulated trading from the exact same time period. I was able to get my simulation to the point that it was pretty accurate. For the parts that were impossible to model exactly, I made sure to at least produce outcomes that were statistically similar (in the metrics I thought were important).

Making profitable trades

With an order simulation model in place I could now send orders in simulation mode and see a simulated P&L. But how would my system know when and where to buy and sell?

The price move predictions were a starting point but not the whole story. What I did was create a scoring system for each of 5 price levels on the bid and offer. These included one level above the inside bid (for a buy order) and one level below the inside offer (for a sell order).

If the score at any given price level was above a certain threshold that would mean my system should have an active bid/offer there — if it was below the threshold, then any active orders should be cancelled. Based on this, it was not uncommon that my system would flash a bid in the market then immediately cancel it. (I tried to minimize this, as it's annoying as heck to anyone looking at the screen with human eyes — including me.)

The price level scores were calculated based on the following factors:

- The price move prediction (that we discussed earlier).
- The price level in question. (Inner levels meant greater price move predictions were required.)
- The number of contracts in front of my order in the queue. (Less was better.)
- The number of contracts behind my order in the queue. (More was better.)

Essentially these factors served to identify "safe" places to bid/offer. The price move prediction alone was not adequate because it did not account for the fact that when placing a bid I was not automatically filled - I only got filled if someone sold to me there. The reality was that the mere fact of someone selling to me at a certain price changed the statistical odds of the trade.

The variables used in this step were all subject to optimization. This was done in the exact same way as I optimized variables in the price move indicators except in this case I was optimizing for bottom-line P&L.

What my program ignored

When trading as humans we often have powerful emotions and biases that can lead to less than optimal decisions. Clearly I did not want to codify these biases. Here are some factors my system ignored:

- The price that a position was entered — In a trading office it's pretty common to hear conversation about the price at which someone is long or short, as if that should affect their future decision making. While this has some validity as part of a risk reduction strategy, it really has no bearing on the future course of events in the market. Therefore, my program completely ignored this information. It's the same concept as ignoring sunk costs.
- Going short vs. exiting a long position — Typically a trader would have different criteria that determines where to sell a long position versus where to go short. However, from my algorithm's perspective there was no reason to make a distinction. If my algorithm expected a downward move, selling was a good idea regardless if it was currently long, short, or flat.
- A "doubling up" strategy — This is a common strategy where traders will buy more stock in the event that their original trade goes against them. This results in your average purchase price being lower, and it means when (or if) the stock turns around, you'll be set to make your money back in no time. In my opinion, this is really a horrible strategy unless you're Warren Buffet. You're tricked into thinking you are doing well because most of your trades will be winners. The problem is that when you lose, you lose big. The other effect is it makes it hard to judge if you actually have an edge on the market or are just getting lucky. Being able to monitor and confirm that my program did in fact have an edge was an important goal.

Risk management

Since my algorithm made decisions the same way regardless of where it entered a trade or if it was currently long or short, it did occasionally sit in (and take) some large losing trades (in addition to some large winning trades). But, you shouldn't think there wasn't any risk management.

To manage risk I enforced a maximum position size of 2 contracts at a time, occasionally bumped up on high-volume days. I also had a maximum daily loss limit to safeguard against any unexpected market conditions or a bug in my software. These limits were enforced in my code but also in the backend through my broker. As it happened, I never encountered any significant problems.

Running the algorithm

From the moment I started working on my program, it took me about 6 months before I got it to the point of profitability and ran it live. Although to be fair, a significant amount of time was learning a new programming language. As I worked to improve the program, I saw increased profits for the next four months.

Each week I would retrain my system based on the previous 4 weeks' worth of data. I found this struck the right balance between capturing recent market behavioral trends and ensuring my algorithm had enough data to establish meaningful patterns. As the training began taking more and more time, I split it out so that it could be performed by 8 virtual machines using Amazon EC2. The results were then coalesced on my local machine.

The high point of my trading was October 2009 when I made almost $100K. After this I continued to spend the next four months trying to improve my program despite decreased profit each month. Unfortunately, by this point I guess I'd implemented all my best ideas because nothing I tried seemed to help much.

With the frustration of not being able to make improvements and not having a sense of growth, I began thinking about a new direction. I emailed 6 different high frequency trading firms to see if they'd be interested in purchasing my software and hiring me to work for them. Nobody replied. I had some new startup ideas I wanted to work on, so I never followed up.

Afterwords

I posted this on Hacker News and it has gotten a lot of attention. I just want to say that I do not advocate anyone trying to do something like this by themselves now. You would need a team of really smart people with a range of experiences to have any hope of competing. Even when I was doing this, I believe it was very rare for individuals to achieve success (though I had heard of others). ■

Jesse Spaulding is a serial entrepreneur with a background that includes day trading and automated algorithmic trading. He has founded, managed, and sold a comparison shopping site in Australia and since then worked on several startups that haven't panned out. His current project is CourseTalk, which can be considered a Yelp for MOOCS (Massive Open Online Courses).

25 Entrepreneurs Tell What They Wish They'd Known before Founding Their First Startup

By DAVID HAUSER

HINDSIGHT IS 20/20. When you look back on any project or endeavor, you get a better idea of what was important and what wasn't.

The same is true with startups. After working on a business for a year or two or more, you have a better idea about what was worth worrying about and what wasn't as big of a deal.

Since entrepreneurs are the most qualified to give other entrepreneurs advice about starting a business, I decided to ask twenty-five entrepreneurs about the number one thing they wish they'd known before founding their first startup. Below is a collection of this advice. It's invaluable whether you've recently started a business or you're looking to start one.

DAN MARTELL

Founder of Clarity

That you're not supposed to know how to do anything right, and that's okay.

It wasn't till I sold Spheric and started working on Flowtown that I realized that you didn't need to know how to do anything in the beginning— you just needed to get good at finding the right answers quickly. If you focused on learning, getting the right advice, in near real time, then you could take on any challenge. It's quite liberating once you realize that.

NEIL PATEL

Co-founder of Crazy Egg

I wish I knew how to price test. When we first released the product we based pricing off of what we wanted to charge, versus optimizing price to achieve maximum revenue and profitability.

At one point in time our customer base requested a lower pricing option. We did it because their was a high demand for it. Although it increased the total number of signups, it decreased our overall revenue. If we knew about price testing during that time, we wouldn't have made this big mistake.

NICK FRANCIS

Co-founder & CEO of Help Scout

I wish that I knew how difficult it is to acquire a customer, get them to pay for your product, and believe it's as magical as you think it is.

Most startup founders count on customer acquisition as a foregone conclusion, yet it's the number one thing that keeps them up at night for the first 2-3 years if not longer. Every part of that process is deeply challenging for a company. It also doesn't happen quickly.

A few tips on how to navigate early stage customer acquisition challenges:

- Talk to every person in your target market that will speak to you. Know their needs better than they know them. Your most valuable insights will come from talking to customers daily.
- Marketing to potential customers is a series of experiments. Before you start, define what success/failure looks like. When the experiment is over, rinse and repeat.
- Surround yourself with team members and advisers that will hold you accountable to the business' metrics and finances. The success/failure of the business depends on these people. You must trust them completely because you don't have time to look over their shoulder.

ALLAN BRANCH

Co-founder of LessAccounting

Wow, the number one thing…

My business partner, Steve Bristol, and I really used to put in major hours the first years of the company. We were working 80+ hours a week. After working ourselves to a point of being burned out, we realized that if we put in 40 x 2 hours the company didn't move forward 2x faster. In fact those extra 40 hour were less productive than the first 40 hours. The reality is you'll never be "done" with your work, you'll never finish all the tasks, build all the features and have the perfect design. At the end of the day, around 4 pm, we close our laptops and go home. Never forget work is here to make your personal life fruitful.

Also, I no longer care how famous I become, I don't care about being filthy rich or being on the cover of magazines. I care more about making our customers and employees happy. The only people I care about being famous to are my children and wife. I do, even at the age of 32, still strive to make my parents proud of me. I've let go of the burden of trying to focus the company to a $500 million company; I'm happy being the co-founder of an unknown software company.

Misc Tips

- Only hire people you'd want to hang out with during personal time.
- The first 10 hires set the tone for the whole company.
- Don't hire people that are getting a salary bump up by working with you.
- Don't wear white pants after labor day.

LEO WIDRICH

Co-founder of Buffer

The number one thing I wish I knew is that the people around you affect your success more than you would ever imagine.

Focusing on who you spend time with on a day to day basis, working with doers instead of talkers can make or break the progress of your business and, more importantly, self-improvement. Be selective who you choose. Jim Rohn put it best:

> *"You are the average of the five people you spend the most time with."*

RENEE WARREN

Co-founder of Onboardly

I wish I would have better known the value of my time. A "10 minute chat," which always leads to a much longer chat, was so easy to say yes to. It took me years to finally start saying NO to things that would take me away from what really needed my attention. No to meetings. No to interviews, and no to extra projects (for extra money.) When I implemented my daily to-do lists my whole day/week/month changed. I would only accept opportunities if they could come after my to-do's were completed.

Part of this realization came from a quote my grandfather once told me, "If you are not 10 minutes early, you are 10 minutes late." He meant this for many reasons: showing up to meetings, flights, phone calls, the gym, and so much more.

So, that's it: time is the most valuable thing you have. Make sure you invest it wisely.

POORNIMA VIJAYASHANKAR

Founder & CEO of BizeeBee

I knew that it takes time to build a product, but I also wish I had known that it takes time for users to adopt a product. While there may be early adopters who can get wedded to your product, mainstream adoption takes a lot of time and effort. Mainstream adoption requires people who aren't early adopters, those who are more reluctant to change, to discover your product, understand the value proposition, be willing to try it out, then actually use it and pay for it, and finally develop enough of a following to want to tell other people about it. This cycle takes a while because it requires a product to be solid, for a user to develop a relationship with your company and your product, and then finally develop enough attachment to want to talk about it with others.

While marketing efforts can plant the seed, a lot of time needs to pass where the product is out in the market, in order for mainstream adoption to take place. Giving things time is hard for a founder to process, because as a founder you want to think you are in control, and can make things happen, but sometimes you just have to be patient and wait!

JASON TRAFF

Founder of Leaky

Coming from MIT, I got lots of really good advice about starting a company: the importance of vesting, team chemistry, and building a good product. I wish that I had known more about the emotional roller coaster of startup life. Often when startups are portrayed in movies or TV shows, it's a bunch of twenty-somethings playing foosball all day and partying all night. What they rarely show are the lows that accompany those highs.

Never in my life have I been rejected as frequently or as vehemently as I have for Leaky. After all of the countless rejections, the scrapping to make payroll, and the cease-and-desist letters from insurance companies, what I learned was that you need fortitude to look past the temporary highs and lows to know that no gain or setback is ever permanent — otherwise, it would never be possible to get out of bed in the morning.

RAMI ESSAID

Co-founder & CEO of Distil

The one thing I wish I knew before founding my first startup would have been how to set clear and measurable goals. The problem with any startup is that there are a million unknowns. As you go through the journey of creating your company, you try and answer as many of those questions as possible and once enough are answered, you know you have actually created something. Along the way, it is easy to get lost. To make sure you don't, you need to be able to set clear goals and measure the success of your actions. If you see something isn't working, it is imperative that you recognize it as soon as possible and fix the issue or change course quickly. Goals and metrics are the only way to do so.

MIKE ARSENAULT

Co-founder of Rejoiner

Don't guess at price. So much is dictated by the way you price your product and many first-time Founders default to what they "think" customers are willing to pay for it (myself included). Focus on the value you are creating for your customers, not on what it costs you to deliver your product or service.

It also turns out that there are entire methodologies designed to help you extract the ideal pricing structure from your target market (Google "von westendorp"). Equally as important is finding out what product features your customers find most valuable.

By combining "willingness to pay" data with your customers' most desired features, you'll have a grounded approach for uncovering the pricing structure that attracts the right customers and drives the most profits for your business.

ETHAN BLOCH

Co-founder of Flowtown

Never take advice from anyone who hasn't done or isn't doing what you want to accomplish.

KAPIL KALE

Co-founder of GiftRocket

Ideas are great, but it is extremely important to think backwards from distribution. Ask yourself, who would use this, and how would they hear about it?

A lot of times, that will uncover the critical features you need to build into a product to make it useful enough for a user to tell their friends about it.

BO LU

Co-founder & CEO of FutureAdvisor

Look backwards in time.

The things that first-time entrepreneurs spend the first few days of their life as a founder worrying about usually don't matter. When I talk to new founders today, I generally get questions about how to structure the company legally, whether they should leave work now or wait for a bonus to appear, et cetera. Experience shows that these things do not make or break companies.

Instead, I wish first time founders would spend the critical first few days of their life as a founder thinking about their customers and what those customers need. Understanding that intimately better than the next guy will make or break your company. The best way to do this is to look backwards in time. Pretend it is four years from today and you have a successful company, and ask yourself: is the question I am agonizing over right now likely to be the thing I will agonize over four years from now? The answer is usually no.

DHARMESH SHAH

Founder & CTO of HubSpot

It's important to pick a big, growing market where you have some distinct advantage. And, to ensure that you control your own destiny and are not overly dependent on others.

BLAKE WILLIAMS

Co-founder of Keepsy

Seek out the most critical opinions of your plan that you can find. The natural tendency for a first-time entrepreneur is to fall in love with an idea and then look for friends and colleagues to support it. After all, who wants to have a fledgling idea crushed by naysayers? But these are exactly the types of folks you should be looking for.

Have them shred your plan and designs from top to bottom. If you find yourself agreeing with them and having doubts, then your plan (and possibly you) may not have the mettle to make it. But if you are able to defend it with conviction, repeatedly, then you probably have both the moxie to last through the long, tough grind you're facing, as well as a plan that just might work.

RICK PERREAULT

Co-founder & CEO of Unbounce

I wish I knew how important accurate metrics would become and that we could more easily support our reporting needs by preparing on day one. Over the last two years I've heard time and time again, "we can't track that easily because our app [insert issue here]." Had we decided early on what key metrics we would need to track and built the app to support our needs, we would have likely saved ourselves a world of pain.

WALTER CHEN

Co-founder & CEO of iDoneThis

I wish I'd known that running a startup team is a lot like parenting. You check up on them, you wonder what they're doing and you worry about them Skype-ing while driving. Often, you have to yell "Everybody calm down!" On some days, you have to remind them to buckle down and get their work done before dinner. On other days, you have to entertain them, so you take them to see movies and drive them to a go-karting arenas.

As a startup founder, you want to help your team identify their strengths on the job and support them. You want them to make mistakes and learn from them, instead of shying away from them. You don't dictate, you ask, "What do YOU think?" You're sensitive to the ebb and flow in their moods, you know when they're discouraged or frustrated. You get frustrated yourself, but you express it to them constructively. Above all, like any parent, you want them to be happy. Ok, AND successful. Because I'm an Asian parent.

OTTO HILSKA

Founder & CEO of Flowdock

It's well known that "premature optimization is the root of all evil," but somehow I failed to recognize that when we spent a lot of time playing with different databases before we even had any customers. In the beginning it's ok to validate your assumptions with a half-baked product.

ASH RUST

Co-founder of SendHub

Focus is more important than you can ever imagine.

ANDREW ANGUS

Founder & CEO of Switch Video

I wish I knew how big the opportunity was so I could have better planned to take advantage of it. Now that my company is older and more structured it is great to be able to focus on strategy, rather then just reacting all the time. At the beginning if I had stepped back and developed and funded a better plan I could have saved myself making a ton of mistakes. That being said these mistakes are what my education is built on and learning from them is why I think I will continue to be successful. (I would like to have made a few less mistakes though.)

ALEX SCHIFF

Co-founder & CEO of Fetchnotes

I wish someone had taught me earlier that you should be optimizing for speed and not cost. Everything in startups is about speed and your ability to move quickly. We were bootstrapped for a long time before raising a round, so we didn't have much of a choice but to be super frugal, but I do wish that I learned that lesson earlier since it makes all the difference. If it takes an hour of your time to hack something together to save $20/month, then it's not worth it.

GAUTAM GUPTA

Co-founder of NatureBox

Starting NatureBox has been an amazing experience for me and I will be forever grateful for having the opportunity to start this company. I think it's important for founders to know that when starting a company, they are about to embark on an emotional roller coaster ride. Managing your emotional state will become so hard but so important. When you hit a low point, remind yourself that it is just a bump in the road.

You can loose so much time worrying about things that don't even matter. You'll get good and bad news all the time and you'll feel like your life depends on the success of your company but keep your head down and execute.

ARI TULLA

Co-founder & CEO of BetterDoctor

Build a public working prototype as quickly as possible and then iterate furiously.

Our plan at BetterDoctor was to build the first MVP product in two months and release it publicly. We got this done, but it was so light on the viability side that we could only release it in closed beta. Closed beta meant very few users and little real world feedback.

In the end it took over six months before we finally launched the first beta product, which was still very much an MVP. Now after a year we have released BetterDoctor search service nationwide and have a stable platform to build upon. Today we can release new features in couple of days and test them with real users immediately.

A year is a long time, and if there is any way to get the product into consumer hands sooner, you should try to do it.

TRI TRAN

Co-founder & CEO of Munchery

There will be a lot of ups and downs. When you feel down, stay calm and know that things will get better. When you feel up, enjoy the moment but save some of that for a rainy day.

I've been fortunate to have worked at multiple startups to know roughly what to expect. Reading Hacker News regularly gave me a good head start. Most every top rated advice you read there will come into play.

Be prepared that founding your first startup is likely the hardest thing you have ever done in your life to date. It's not at all glamorous. Seek full support from your spouse (if you have one), and seek out co-founder(s) that you can fully trust and work well with. Ultimately, I cannot imagine a better professional experience than founding your very own company!

ELIZABETH YIN

Co-founder & CEO of LaunchBit

The first company I started, a social shopping application, was a complete disaster. We built out what we thought was an awesome tool, but nobody wanted it. We wasted about $20k and about 1.5 years. From that experience, I realized that as much as there is a shortage of tech talent, the hard part isn't the technology — it's getting user demand.

If I were to have done things differently, I would've tested the market with little hacks before building out a product. I would've created landing pages to capture contact information of potential users and would've talked with them beforehand. I would've generated fake buttons that led nowhere or to a "coming soon" message to measure demand. In short, I wish I'd known to build as little as possible to test the market before building a product. ■

David Hauser is the co-founder at Grasshopper Group and Chargify. Follow him on Twitter at @dh

MEET MANDRILL

By MailChimp

Mandrill is a new way to send transactional, triggered, and personalized emails.
It's also the world's largest species of monkey.

MANDRILL.COM

Probabilistic M2M Relationships Using Bloom Filters

By ZACHARY VOASE

HERE'S AN IDEA that's been kicking around inside my head recently.

A standard M2M relationship, as represented in SQL, looks like this:

```
CREATE TABLE movie (
  id SERIAL PRIMARY KEY,
  title VARCHAR(255)
);

CREATE TABLE person (
  id SERIAL PRIMARY KEY,
  name VARCHAR(255)
);

CREATE TABLE movies_people (
  movie_id INTEGER REFERENCES movie,
  person_id INTEGER REFERENCES person
);
```

To find the people for a given movie (including the details of the movie itself):

```
SELECT *
FROM
  movie
  INNER JOIN movie_people ON (movie.id = movie_
people.movie_id)
  INNER JOIN person ON (movie_people.person_id =
person.id)
WHERE movie.id = MOVIE_ID;
```

Finding the movies for a given person just involves changing the `WHERE` predicate to filter for `person.id` instead.

Using a junction table for a sparse or small data set (where there are not many associations between movies and people) gives acceptable space and time consumption properties. But for denser association matrices (which may grow over time), the upper bound on the size of the junction table is *`O(n(movies) * n(people))`*, and the upper bound on the time taken to join all three tables will be the square of that. So what optimizations and trade-offs can be made in such a situation?

Well, we can use a Bloom filter on each side of the M2M relationship and do away with the junction table altogether. Here's what the SQL (for Postgres) looks like:

```
CREATE TABLE movie (
  id SERIAL PRIMARY KEY,
  title VARCHAR(255) UNIQUE,
  person_filter BIT(PERSON_FILTER_LENGTH),
  hash BIT(MOVIE_FILTER_LENGTH)
);

CREATE TABLE person (
  id SERIAL PRIMARY KEY,
  name VARCHAR(255),
  movie_filter BIT(MOVIE_FILTER_LENGTH),
  hash BIT(PERSON_FILTER_LENGTH)
);
```

I haven't calibrated these filters yet, so I've yet to decide how long to make each one. I'm also doing something different compared to the normal explanation of a Bloom filter. Typically each element is expressed as the set of results of *k* hash functions, each mapping to an index in a bit array of length *m*. I prefer to think of a single hash function with an *m*-bit output and a popcount guaranteed to be less than or equal to *k*. This is effectively identical, but it helps you think of the filters themselves in a different way: as a union of a set of hash outputs. All of a sudden, these filters seem less daunting — they're just fancy bit arrays. That's why `length(person.hash) = length(movie.person_filter)`, and vice versa.

Picking a hash

According to Kirsch and Mitzenmacher [hn.my/rsa], you can implement *k* hash functions using only two, with no increase in the false positive probability. Here's a Python example:

```
import pyhash
import bitstring

murmur = pyhash.murmur3_32()
def bloom_hash(string, k, m):
    """Hash a string for a bloom filter with
given `m` and `k`."""
    hash1 = murmur(string)
    hash2 = murmur(string, seed=hash1)
    output = bitstring.BitArray(length=m)
    for i in xrange(k):
        index = (hash1 + (i * hash2)) % m
        output[index] = True
    return output
```

I'm generating a bit array here so it can be simply OR'd with an existing Bloom filter to add the given element to the set.

Testing on example data

To test my system out, I'll use the community-generated MovieLens [hn.my/lens] database.

Cleaning the data

Download and unzip the 1M dataset, with ~6000 users, ~4000 movies and 1 million ratings:

```
$ ls
README  movies.dat    ratings.dat   users.dat
$ wc -l *.dat
    3883 movies.dat
 1000209 ratings.dat
    6040 users.dat
 1010132 total
```

The field separators in these files are `::`, but I want to convert them to tabs, so they play better with standard GNU userspace tools:

```
$ sed -i -e 's/::/\t/g' *.dat
```

Because we're treating set membership as binary, I'll use a high-pass filter for ratings — that is, I'll only consider higher-than-average ratings.

```
# Compute the average (the rating is the third
# column of ratings.dat)
$ awk '{ sum += $3 } END { print sum/NR }' \
  ratings.dat
3.58156
# Ratings are integral, so we just keep ratings
# of 4 or 5.
$ awk '$3 > 3 { print }' ratings.dat > \
  good-ratings.dat
```

How many ratings now?

```
$ wc -l good-ratings.dat
575281
```

Picking filter sizes

Given that we have 3,883 movies, 6,040 users, and 575,281 ratings, we can estimate the average number of elements in `movie.person_filter` to be 148, and for `person.movie_filter`, 95. The optimal size for a filter is given by the following formula:

$$m = -\frac{n \ln p}{(\ln 2)^2}$$

Choosing a false positive probability of 0.5% (0.005), that gives us a `movie.person_filter` of 1,632 bits, and a `person.movie_filter` of 1,048 bits. So our schema now looks like this (with some minor modifications):

```
CREATE TABLE movie (
  id INTEGER PRIMARY KEY,
  title VARCHAR(255) UNIQUE NOT NULL,
  person_filter BIT(1632) DEFAULT 
0::BIT(1632),
  hash BIT(1048) NOT NULL
);

CREATE TABLE person (
  id INTEGER PRIMARY KEY,
  name VARCHAR(255) UNIQUE NOT NULL,
  movie_filter BIT(1048) DEFAULT 
0::BIT(1048),
  hash BIT(1632) NOT NULL
);
```

These may seem large, but we're only adding 335 bytes for each movie and person. Our *k* value can also be calculated as follows:

$$p = 2^{-k}$$
$$\therefore k = -\log_2 p$$

Yielding a *k* of around 8 for both filters (since we decided our *p* in advance).

Loading the data: movies and people

The next step is to load the raw data for movies and people (but not yet ratings) into the database. Assuming the `CREATE TABLE` statements have already been issued separately:

```
from collections import namedtuple
import csv

import psycopg2

# Classes for handling the TSV input.

_User = namedtuple('_User', 'id gender age occupation 
                   zipcode')
class User(_User):

    @property
    def name(self):
        return '%s:%s:%s' % (self.id, self.age,
                             self.zipcode)

    @property
    def hash(self):
        return bloom_hash(self.name, 8, 1632).bin

_Movie = namedtuple('_Movie', 'id title genres')
class Movie(_Movie):

    @property
    def hash(self):
        return bloom_hash(self.title.encode('utf-8'), 8,
                          1048).bin


# This should be run from the directory containing
# `users.dat` and `movies.dat`
conn = psycopg2.connect('host=localhost dbname=movielens')

with conn.cursor() as cur:
    cur.execute('BEGIN')

    with open('users.dat') as users_file:
        users = csv.reader(users_file, delimiter='\t')
        for user in users:
            # The input is encoded as ISO-8859-1, and
            # unfortunately Python's csv lib doesn't handle
            # Unicode text well, so we have to decode it
            # after reading it.
            user = User(*[s.decode('iso-8859-1')
                           for s in user])
            cur.execute('''INSERT INTO person (id, name, hash)
                           VALUES (%s, %s, %s)''',
                        (int(user.id), user.name, user.hash))

    with open('movies.dat') as movies_file:
        movies = csv.reader(movies_file, delimiter='\t')
        for movie in movies:
            movie = Movie(*[s.decode('iso-8859-1') for s in movie])
            cur.execute('''INSERT INTO movie (id, title, hash)
                           VALUES (%s, %s, %s)''',
                        (int(movie.id), movie.title, movie.hash))

    cur.execute('COMMIT')
```

Loading the data: ratings

For the purpose of comparison, I'm going to load the data using both Bloom filters and a standard junction table. Create that table:

```
CREATE TABLE movie_person (
  movie_id INTEGER REFERENCES movie (id),
  person_id INTEGER REFERENCES person (id)
);
```

Now load in the ratings data for both the junction table and the Bloom filters:

```
with closing(conn.cursor()) as cur:
    cur.execute('BEGIN')
    with open('good-ratings.dat') as ratings_file:
        ratings = csv.reader(ratings_file, delimiter='\t')
        for rating in ratings:
            cur.execute('''INSERT INTO movie_person (movie_id, person_id)
                           VALUES (%s, %s)''',
                        (int(rating[1]), int(rating[0])))
    cur.execute('''UPDATE movie
                   SET person_filter = (
                       SELECT bit_or(person.hash)
                       FROM person, movie_person
                       WHERE person.id = movie_person.person_id AND
                             movie_person.movie_id = movie.id);''')
    cur.execute('''UPDATE person
                   SET movie_filter = (
                       SELECT bit_or(movie.hash)
                       FROM movie, movie_person
                       WHERE person.id = movie_person.person_id AND
                             movie_person.movie_id = movie.id);''')
    cur.execute('COMMIT')
```

This may take a few minutes.

Checking the performance

To query the movies for a given user (and vice versa) in the *traditional* way:

```
CREATE VIEW movies_for_people_junction AS
SELECT movie_person.person_id,
       movie.id AS movie_id,
       movie.title AS title
FROM movie, movie_person
WHERE movie.id = movie_person.movie_id;
```

And in the new, *Bloom filtered* way:

```
CREATE VIEW movies_for_people_bloom AS
SELECT person.id AS person_id,
       movie.id AS movie_id,
       movie.title AS title
FROM person, movie
WHERE (person.hash & movie.person_filter) =
person.hash;
```

Checking the query performance for the junction-based query:

```
EXPLAIN ANALYZE SELECT * FROM movies_for_people_
junction WHERE person_id = 160;
```

The result:

```
Hash Join  (cost=282.37..10401.08 rows=97
width=33) (actual time=7.440..64.843 rows=9
loops=1)
  Hash Cond: (movie_person.movie_id = movie.id)
  ->  Seq Scan on movie_person
(cost=0.00..10117.01 rows=97 width=8) (actual
time=2.540..59.933 rows=9 loops=1)
        Filter: (person_id = 160)
  ->  Hash  (cost=233.83..233.83 rows=3883
width=29) (actual time=4.884..4.884 rows=3883
loops=1)
        Buckets: 1024  Batches: 1  Memory Usage:
233kB
        ->  Seq Scan on movie
(cost=0.00..233.83 rows=3883 width=29) (actual
time=0.010..2.610 rows=3883 loops=1)
Total runtime: 64.887 ms
```

And for the Bloom query:

```
EXPLAIN ANALYZE SELECT * FROM movies_for_people_
bloom WHERE person_id = 160;
```

The result:

```
Nested Loop  (cost=4.26..300.35 rows=1 width=33)
(actual time=0.033..2.546 rows=430 loops=1)
  Join Filter: ((person.hash & movie.person_
filter) = person.hash)
  ->  Bitmap Heap Scan on person
(cost=4.26..8.27 rows=1 width=216) (actual
time=0.013..0.013 rows=1 loops=1)
        Recheck Cond: (id = 160)
        ->  Bitmap Index Scan on person_id_
idx  (cost=0.00..4.26 rows=1 width=0) (actual
time=0.009..0.009 rows=1 loops=1)
              Index Cond: (id = 160)
  ->  Seq Scan on movie  (cost=0.00..233.83
rows=3883 width=241) (actual time=0.014..0.785
rows=3883 loops=1)
Total runtime: 2.589 ms
```

Much better! I'm pretty sure there are still places where both the junction table and the Bloom table could be optimized, but this serves as a great demonstration of how a typically inefficient query can be sped up by just using a garden-variety probabilistic data structure, and sacrificing a minimal amount of accuracy. ■

Zack is a freelance Python and UNIX hacker based in London. He's currently working on kickstarting the London Big-O Meetup.

Reprinted with permission of the original author.
First appeared in *hn.my/m2mbloom* (zacharyvoase.com)

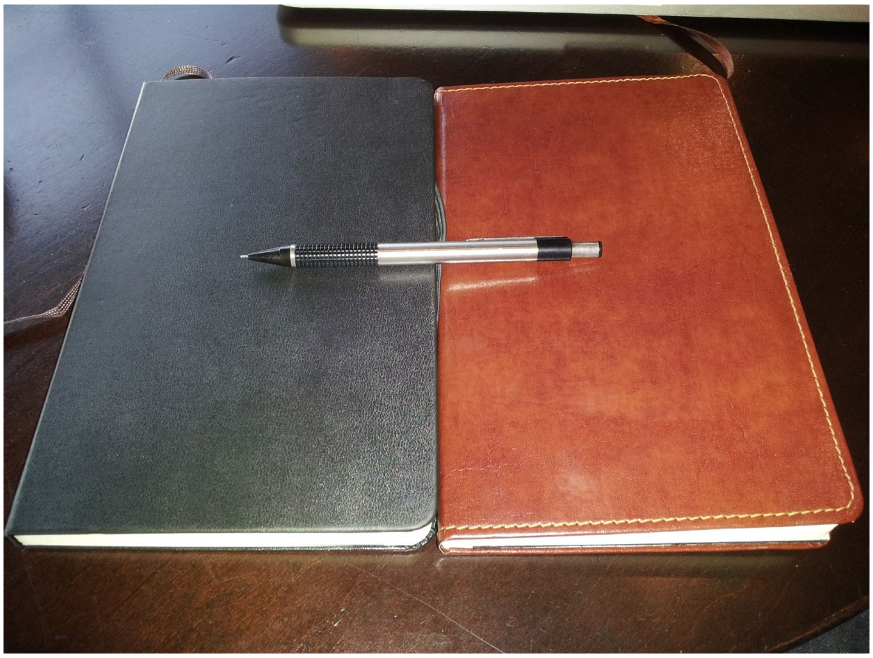
Keep a Programming Journal

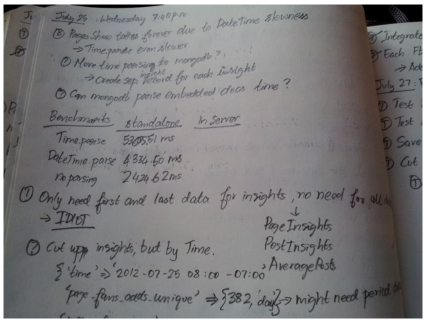
By SENTHIL ARIVUDAINAMBI

One of my favorite past times is to look at the notebooks of famous scientists. Da Vinci's notebook [hn.my/davincii] is well known, but there plenty [hn.my/edison] of others [hn.my/einstein]. Worshipping Da Vinci like no other, I bought a Think/Create/Record journal, used it mostly to jot down random thoughts and take notes. This was great in the beginning, but the conformity of lines drove me nuts. Only moleskines made blank notebooks, so I had to buy one.

At the same time, I started a freelance project. The project itself is irrelevant, but suffice it to say it was very complex and spanned several months. It seemed like a perfect opportunity to use the moleskine. Looking back, all my entries fell under few categories:

- To-do
- Question
- Thought
- Bug
- Feature





July 25 : Wednesday 7:40 p.m.

Ⓑ Pages.Show takes forever due to DateTime slowness
→ Time.parse even slower

Ⓠ Move time parsing to mongodb ?
→ Create sep. insight record for each insight

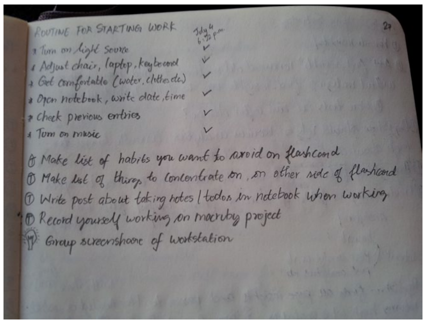
Ⓠ Can mongodb parse embedded docs time ?

Benchmarks	standalone	In Server
Time.parse	53~~0~~05.51 ms	
DateTime.parse	4334.50 ms	
no parsing	242462ms	

Ⓣ Only need first and last data for insights, no need for all
→ IDIOT

Ⓠ Cut up insights, but by Time.
{ 'time' => '2012-07-25 08:00 -07:00'
'page_fans_adds_unique' => {382, 'day'} → might need period

PageInsights
PostInsights
AveragePosts



ROUTINE FOR STARTING WORK — July 4, 6:22 p.m. — 27

* Turn on light source ✓
* Adjust chair, laptop, keyboard ✓
* Get comfortable (water, clothes etc) ✓
* Open notebook, write date, time ✓
* Check previous entries ✓
* Turn on music ✓

Ⓕ Make list of habits you want to avoid on flashcard
Ⓣ Make list of things to concentrate on, on other side of flashcard
Ⓣ Write post about taking notes / todos in notebook when working
Ⓕ Record yourself working on macruby project
Group screenshare of workstation

Clearly there isn't any technological reason you couldn't use Github issues or Pivotal or Jira. I tried those, but none of them caught on. The real value for me was, oddly, not looking back at the entries, but writing them down on paper, especially the question or thought. I'd write it down, research online, test it, and write down the results. It gets tedious at times, but at the end of the day it's a real pleasure to look back and see how far I've come.

If you decide to start your own journal, here are few tips:

- Keep a list of contents
- Number each page
- Note date and time before each entry
- Write everything

I've been doing this experiment for the past 4 months and it's been very helpful. Some days I'd feel lazy or be excited to write stuff down, but inevitably the regret train hits me the next day. Nowadays I open the notebook and write down date and time first thing in the morning. ■

Senthil Arivudainambi is a Ruby developer current doing Hacker School Fall 2012 batch in New York.

Reprinted with permission of the original author. First appeared in *hn.my/journal* (getriver.com)

Photographs by Senthil Arivudainambi.

Homomorphic Hashing

By NICK JOHNSON

In the last Damn Cool Algorithms article (Hacker Monthly issue #21), we learned about Fountain Codes [hn.my/fountain], a clever probabilistic algorithm that allows you break a large file up into a virtually infinite number of small chunks, such that you can collect any subset of those chunks — as long as you collect a few more than the volume of the original file — and be able to reconstruct the original file. This is a very cool construction, but as we observed last time, it has one major flaw when it comes to use in situations with untrusted users, such as peer to peer networks: there doesn't seem to be a practical way to verify if a peer is sending you valid blocks until you decode the file, which happens very near the end — far too late to detect and punish abuse.

It's here that homomorphic hashes come to our rescue. A homomorphic hash is a construction that's simple in principle: a hash function such that you can compute the hash of a composite block from the hashes of the individual blocks. With a construction like this, we could distribute a list of individual hashes to users, and they could use those to verify incoming blocks as they arrive, solving our problem.

Homomorphic hashing is described in the paper "On-the-fly verification of rateless erasure codes for efficient content distribution" by Krohn et al [hn.my/hpaper]. It's a clever construction, but rather difficult to understand at first, so we'll start with a strawman construction of a possible homomorphic hash, then improve upon it until it resembles the one in the paper — at which point you will hopefully have a better idea as to how it works. We'll also discuss the shortcomings and issues of the final hash, as well as how the authors propose to resolve them.

Before we continue, a small disclaimer is needed: I'm a computer scientist, not a mathematician, and my discrete math knowledge is far rustier than I'd like. This paper stretches the boundaries of my understanding, and describing the full theoretical underpinnings of it is something I'm likely to make a hash of. So my goal here is to provide a basic explanation of the principles, sufficient for an intuition of how the construction works, and leave the rest for further exploration by the interested reader.

A Homomorphic Hash That Isn't

We can construct a very simple candidate for a homomorphic hash by using one very simple mathematical identity: the observation that $g^{x0} * g^{x1} = g^{x0 + x1}$. So, for instance, $2^3 * 2^2 = 2^5$. We can make use of this by the following procedure:

1. Pick a random number g
2. For each element x in our message, take g^x. This is the hash of the given element.

Using the identity above, we can see that if we sum several message blocks together, we can compute their hash by multiplying the hashes of the individual blocks, and get the same result as if we "hash" the sum. Unfortunately, this construction has a couple of obvious issues:

- Our "hash" really isn't — the hashes are way longer than the message elements themselves!
- Any attacker can compute the original message block by taking the logarithm of the hash for that block. If we had a real hash with collisions, a similar procedure would let them generate a collision easily.

A Better Hash With Modular Arithmetic

Fortunately, there's a way we can fix both problems in one shot: by using modular arithmetic. Modular arithmetic keeps our numbers bounded, which solves our first problem, while also making our attacker's life more difficult: finding a preimage [hn.my/preimage] for one of our hashes now requires solving the discrete log problem [hn.my/discrete], a major unsolved problem in mathematics, and the foundation for several cryptosystems.

Here, unfortunately, is where the theory starts to get a little more complicated — and I start to get a little more vague. Bear with me.

First, we need to pick a modulus for adding blocks together. We'll call it `q`. For the purposes of this example, let's say we want to add numbers between 0 and 255, so let's pick the smallest prime greater than 255, which is 257.

We'll also need another modulus under which to perform exponentiation and multiplication. We'll call thisp. For reasons relating to Fermat's Little Theorem [hn.my/fermat], this also needs to be a prime, and further, needs to be chosen such that `p - 1` is a multiple of `q` (written `q | (p - 1)`, or equivalently, `p % q == 1`). For the purposes of this example, we'll choose 1543, which is `257 * 6 + 1`.

Using a finite field also puts some constraints on the number, g, that we use for the base of the exponent. Briefly, it has to be "of order q", meaning that g^q mod p must equal 1. For our example, we'll use 47, since 47^{257} `% 1543 == 1`.

So now we can reformulate our hash to work like this: to hash a message block, we compute g^b `mod p` — in our example, 47^b `mod 1543` — where b is the message block. To combine hashes, we simply multiply them `mod p`, and to combine message blocks, we add them `mod q`.

Let's try it out. Suppose our message is the sequence `[72, 101, 108, 108, 111]` — that's "Hello" in ASCII. We can compute the hash of the first number as 47^{72} `mod 1543`, which is 883. Following the same procedure for the other elements gives us our list of hashes: `[883, 958, 81, 81, 313]`.

We can now see how the properties of the hash play out. The sum of all the elements of the message is 500, which is `243 mod 257`. The hash of 243 is 47^{243} `mod 1543`, or 376. And the product of our hashes is `883 * 958 * 81 * 81 * 313 mod 1543` — also 376! Feel free to try this for yourself with other messages and other subsets. They'll always match, as you would expect.

A Practical Hash

Of course, our improved hash still has a couple of issues:

- The domain of our input values is small enough that an attacker could simply try them all out to find collisions. And the domain of our output values is small enough the attacker could attempt to find discrete logarithms by brute force, too.
- Although our hashes are shorter than they were without modular arithmetic, they're still longer than the input.

The first of these is fairly straightforward to resolve: we can simply pick larger primes for p and q. If we choose ones that are sufficiently large, both enumerating all inputs and brute force logarithm finding will become impractical.

The second problem is a little trickier, but not hugely so; we just have to reorganize our message a bit. Instead of breaking the message down into elements between 0 and q, and treating each of those as a block, we can break the message into arrays of elements between 0 and q. For instance, suppose we have a message that is 1024 bytes long. Instead of breaking it down into 1024 blocks of 1 byte each, let's break it down into, say, 64 blocks of 16 bytes. We then modify our hashing scheme a little bit to accommodate this:

Instead of picking a single random number as the base of our exponent, g, we pick 16 of them, g_0 - g_{16}. To hash a block, we take each number g_i and raise it to the power of the corresponding sub-block. The resulting output is the same length as when we were hashing only a single block per hash, but we're taking 16 elements as input instead of a single one. When adding blocks together, we add all the corresponding sub-blocks individually. All the properties we had earlier still hold. Better, we've given ourselves another tunable parameter: the number of sub blocks per block. This will be invaluable in getting the right tradeoff between security, granularity of blocks, and protocol overhead.

Practical Applications

What we've arrived at now is pretty much the construction described in the paper, and hopefully you can see how it would be applied to a system utilizing fountain codes. Simply pick two primes of about the right size (the paper recommends 257 bits for q and 1024 bits for p), figure out how big you want each block to be (and hence how many sub-blocks per block) and figure out a way for everyone to agree on the random numbers for g (such as by using a random number generator with a well defined seed value).

The construction we have now, although useful, is still not perfect, and has a couple more issues we should address. First of these is one you may have noticed yourself already: our input values pack neatly into bytes — integers between 0 and 255 in our example — but after summing them in a finite field, the domain has grown, and we can no longer pack them back into the same number of bits. There are two solutions to this: the tidy one and the ugly one.

The tidy one is what you'd expect: since each value has grown by one bit, chop off the leading bit and transmit it along with the rest of the block. This allows you to transmit your block reasonably sanely and with minimal expansion in size, but is a bit messy to implement and seems (at least to me) inelegant.

The ugly solution is this: pick the smallest prime number larger than your chosen power of 2 for q, and simply ignore or discard overflows. At first glance this seems like a terrible solution, but consider: the smallest prime larger than 2^{256} is 2^{256} `+ 297`. The chance that a random number in that range is larger than 2^{256} is approximately 1 in `3.9 * 1074`, or approximately one in 2^{247}. This is way smaller than the probability of, say, two randomly generated texts having the same SHA-1 hash.

Thus, I think there's a reasonable argument for picking a prime using that method, then simply ignoring the possibility of overflows. Or, if you want to be paranoid, you can check for them, and throw out any encoded blocks that cause overflows. There won't be many of them, to say the least.

Performance And How To Improve It

Another thing you may be wondering about this scheme is just how well it performs. Unfortunately, the short answer is "not well." Using the example parameters in the paper, for each sub-block we're raising a 1024 bit number to the power of a 257 bit number; even on modern hardware this is not fast. We're doing this for every 256 bits of the file, so to hash an entire 1 gigabyte file, for instance, we have to compute over 33 million exponentiations. This is an algorithm that promises to really put the assumption that it's always worth spending CPU to save bandwidth to the test.

The paper offers two solutions to this problem; one for the content creator and one for the distributors.

For the content creator, the authors demonstrate that there is a way to generate the random constants g, used as the bases of the exponents using a secret value. With this secret value, the content creator can generate the hashes for their files much more quickly than without it. However, anyone with the secret value can also trivially generate hash collisions, so in such a scheme, the publisher must be careful not to disclose the value to anyone, and only distribute the computed constants g_i. Further, the set of constants themselves isn't small. With the example parameters, a full set of constants weighs in at about the size of 4 data blocks. Thus, you need a good way to distribute the per-publisher constants in addition to the data itself. Anyone interested in this scheme should consult section C of the paper, titled "Per-Publisher Homomorphic Hashing."

For distributors, the authors offer a probabilistic check that works on batches of blocks, described in section D, "Computational Efficiency Improvements". Another easier to understand variant is this: instead of verifying blocks individually as they arrive, accumulate blocks in a batch. When you have enough blocks, sum them all together, and calculate an expected hash by taking the product of the expected hashes of the individual blocks. Compute the composite block's hash. If it verifies, all the individual blocks are valid! If it doesn't, divide and conquer: split your batch in half and check each, winnowing out valid blocks until you're left with any invalid ones.

The nice thing about either of these procedures is that they allow you to trade off verification work with your vulnerability window. You can even dedicate a certain amount of CPU time to verification, and simply batch up incoming blocks until the current computation finishes, ensuring you're always verifying the last batch as you receive the next.

Conclusion

Homomorphic hashing provides a neat solution to the problem of verifying data from untrusted peers when using a fountain coding system, but it's not without its own drawbacks. It's complicated to implement and computationally expensive to compute, and requires careful tuning of the parameters to minimize the volume of the hash data without compromising security. Used correctly in conjunction with fountain codes, however, homomorphic hashing could be used to create an impressively fast and efficient content distribution network. ■

Nick Johnson is a Developer Programs Engineer for Google App Engine. He regularly blogs about interesting computer science topics at his blog [blog.notdot.net], and when he's not saving the world there he can be found on Twitter (@nicksdjohnson) or Stack Overflow helping folks out. He likes long walks on the beach, beautiful sunsets, and Dijkstra's pathfinding algorithm.

Thoughts on Being a Programmer

By WESLEY DARLINGTON

- Don't be an asshole.
- Simple code is hard to write.
- Exquisitely simple code is exquisitely hard to write.
- Just because it's easy to understand doesn't mean it was easy to write.
- In fact, the easier it is to understand, the harder it probably was to write.
- There are many ways to do something.
- The first way you think of is highly unlikely to be the best way.
- Anyway, there probably is no best way - just lots of ways that are differently good.
- There's always plenty of room for improvement - in your code, in your abilities, in you.
- If you think you're as good as you're ever going to be, you're probably right.
- "One-line changes" are never "one-line changes."
- Learn to desire success more than you fear failure.
- You're only old when you can no longer learn new tricks.
- Always back up before tidying up.
- RTFM.
- Err vicariously.
- Sometimes it's OK to be a bit of an asshole, but don't make a habit of it. ■

The author, a pretty mediocre programmer, is currently between startups, working on monitoring large networks for an unnamed search engine company in Mountain View.

Teaching My 10 Year Old Niece How To Program

By PABLO RIVERA

THE FOLLOWING CONVERSATION took place some months ago.

Niece: *Uncle, will you teach me how to use a computer?*
Me: Yes. What is it that you want to do?
Niece: *I don't know. What can it do?*
Me: It can do a lot of things. Do you have anything in mind?
Niece: *No, but I want to learn how to use it.*

A normal person would have told her computers are for nerds, and not to waste her time with one. As a hacker, I saw that she is really into computers and not only wants to learn how to use it (referring to the operating system), but how to make useful stuff with it (programming).

I devised a quick plan. I would focus on showing her the basics behind a webpage, and then introduce some JavaScript to demonstrate how the computer follows the instructions we give it. In her first day she learned simple HTML markup. It was so exciting to be able to write her name and for it to show on the computer's screen. It was such a rush that she spent more than one hour writing markup.

Fast forward some weeks, and I sat down with her to review what she had learned. Not surprisingly, she still remembered about 80% of the HTML stuff. We opened up the same HTML file she had been working with, and I started telling her a bit more about the computer. I said:

> *"Niece, you have to realize something if you want to learn how to program. Computers are stupid and dumb. They take everything you tell them and do it. Never asking questions. That is why programming them is so simple. You just tell the computer what to do."*

Her eyes lit up. She wanted to test my words. I proceeded to write some simple JavaScript code using the `document.write()` method. She would assign values to a set of variables, and then have JS write the values to her webpage. How did she do? She wrote the example ten times without stopping. When she refreshed the page, some parts were not working. Her first bugs! Amazingly, she started debugging the little programs by making sure they all were typed correctly. I have to say that made me very proud. After some minutes of debugging, she realized that some semi-colons were missing. Some quick work on her part and the scripts were working as intended.

About a month passed, and I sat down with her again. This time to show her how to do simple math with JS. It was very productive because it allowed her to see the value of learning math, something her math teachers have failed to do. The multiplication tables had given her some grief. We focused on them. A quick and short program taught her how it all worked. Here is the code.

```
var base = 5; // table for number 5.
var num = 4;
// multiply the base number by this one.

var total = base * num
// total displays the results from multiplying
// base and num.

document.write(total);
//display the results from the operation.
```

It was as if someone had lifted a huge burden. She now understood multiplication and had seen how it worked without any complicated math syntax. This motivated her to play with the little program for a long while. Going through every number and saying out loud how much X by Y was. She would then confirm the results with the program.

Two weeks ago she brought her netbook from home with the intention of putting every one of the examples we had done on it. She wanted to code in her spare time. Unbelievable!

I went ahead and did something better. Went to *python.org* and installed the Python programming language on her computer. She was surprised when I told her Python was free. Actually, she didn't believe me at first. So we booted up python IDLE, I showed her the difference between the interpreter console and a blank program page. She quickly got it, and we moved on to programming Python.

I did not explain much, and just jumped into writing code for her to practice with. It was something along the lines of:

```
def get_name():
  name = str(raw_input("What is your name?\n"))
  return name
def show_name(name):
  print name

my_name = get_name()
show_name(my_name)
```

Results? She understood what the functions did without much hassle. She even went as far as understanding that a function is for encapsulating functionality. But I wasn't done. I added the following to the code:

```
for i in range(10):
    if name == "her name":
        print "Hello ", name
    else:
        print "Hello stranger"
```

The `for` loop was tough for her. Because she didn't get why a computer would do the same thing over and over. I quickly wrote some code to count from one to ten, and that did the trick. She then understood that computers have this thing called a stack that uses pointers to do stuff to the data. It was already a bit late, and she had to go home. I gave her the assignment of writing the Python program twenty times. She did not complain.

Did she complete her assignment? I haven't asked, but I'm pretty sure she did. I'm also sure that she will have modified the source code to do other stuff.

People tend to complain about how there are not enough women in software. It's true, there are not enough of them. That is why I'm doing my part to add one more to the ranks. Teach the kids in your family to program. You don't know if they will be the next Ritchie. ■

Pablo Rivera is a freelance software engineer.He is also the founder of Nuuton and Protocademy. He started hacking on a C64.

Things I Do To Be Consistently Happy

By JOEL GASCOIGNE

NOW THAT IT'S almost two years since I first had the idea for Buffer [bufferapp.com], and since the year and a half before that when I worked on my previous startup, I've started to notice a few patterns amongst the ups and downs that come with building a startup.

One of the most important things I've learned during this time is that I perform best when I'm happy. It really does change everything. If I'm happy then I'm more productive when hacking code, I'm better at answering support, and I find it easier to stay focused.

I've found that there are a few key habits which, for me, act as great rituals for enabling me to be consistently happy. They also act as anchor activities to bring my happiness level back up quickly whenever I have a period where I'm not feeling 100%. So here are 6 things I do:

1 Wake up early

One of the things I love about running my own startup is that I have complete freedom to experiment with my daily routine.

Through experimentation, I've found that waking up early every day makes me feel most invigorated and happy. It gives me a great start to the day, and this almost always leads to a great rest of the day. Over time, I've found I crave that "early morning" feeling, a time I can do some great work and be super focused. Gretchen Rubin from The Happiness Project mentioned something similar a recent article:

"I get up at 6:00 a.m. every day, even on weekends and vacation, because I love it."

Waking up early every day requires discipline, especially about what time I sleep. Right now, I have a sleep ritual of disengaging from the day at 9:30pm and sleeping at 10pm. I now love all aspects of this ritual and with it in place I awake at 6am feeling fresh.

2 Exercise daily

"We found that people who are more physically active have more pleasant-activated feelings than people who are less active."
– Amanda Hyde

In the last three years, I've gone from dabbling with exercise to making it something I do every weekday without fail. At first I had no idea what to do at the gym, so I asked my brother, who's a personal trainer. I then went a few times with a good friend, and soon I was hooked.

Over time, I developed this into a daily ritual so strong that I feel a pull towards it, and by doing it consistently I feel fantastic and can more easily take on other challenges. I recently discovered that exercise is a keystone habit which paves the way for growth in all other areas. I've also found that it helps me to get high quality sleep each night.

3 Have a habit of disengagement

"The richest, happiest, and most productive lives are characterized by the ability to fully engage in the challenge at hand, but also to disengage periodically and seek renewal." – Loehr and Schwarz, The Power of Full Engagement

As I mentioned earlier, a key way I am able to wake up at 6am is through my ritual of disengaging in the evening. I go for a walk at 9:30pm, along a route which I've done many times before. Since the route is already decided and is the same every time, I am simply walking and doing nothing else. This prompts reflection and relaxation.

Various thoughts enter and leave my mind during the walk, and I've found this to be very healthy. Sometimes I think about the great things I enjoyed that day. Other times I will realize a change I should make in order to be happier day to day. I also feel calm and relaxed by the time I return from my walk, and I can therefore go straight to bed and fall asleep sooner than if I been engaged in my work and had closed my laptop only a few minutes earlier.

4 Regularly help others

One of my most fascinating discoveries about myself so far this year, is how happy it makes me to help others. For some time I had been consistently meeting founders to help them with their startups without realizing that it was making me so happy. Then when I read Happiness: A Guide to Developing Life's Most Important Skill by Matthieu Ricard, I connected the dots of when I was happy and the activity I was doing: helping others.

I read Ricard's section on the link between altruism and happiness and everything clicked. Since then, I've been consistently helping many startup founders and it's brought me much happiness through both the challenge of finding ways to help each person, and the feeling that comes when I help the other person discover ways to make faster progress with their current challenges.

5 Learn new skills

"Being in the moment, focusing completely on a single task, and finding a sense of calm and happiness in your work. Flow is exactly that." – Leo Babauta

One thing I've found during my time working on Buffer, is that a key reason I've been happy for most of that time is that I've consistently had new challenges to take on. It may seem odd that new challenges can equate to happiness, but it is the times when I've slipped into a few weeks of working on something I already know well, that have led me to feel less happy than I want to be.

I think a key part of why learning new skills can bring happiness, is that you need to concentrate in order to make progress. The "flow" state has been found to trigger happiness. In addition, when learning something new you are able to learn a lot in a short space of time due to a steep learning curve. For example, in the last two weeks I've started learning Android development from scratch and I've personally found incredible the amount I know now compared to nothing two weeks ago.

6 Have multiple ways to "win" each day

Since the above activities are habitual, many days of the week I actually accomplish all of them. If I succeed with all five, I have a truly amazing day and feel fantastic. I have goals for Buffer, and I have goals in my weights routine, too. In addition, I try to schedule one or two meetings or Skype calls to help people each day. I do this based on learning from around a year ago through an interview Tim Ferriss had with Matt from 37signals. I've mentioned it before on my blog, but it's so good that I want to repeat it:

"If your entire ego and identity is vested in your startup, where there are certainly factors outside of your control, you can get into a depressive funk that affects your ability to function. So, you should also, let's say, join a rock climbing gym. Try to improve your time in the mile. Something like that. I recommend at least one physical activity. Then even if everything goes south — you have some horrible divorce agreement with your co-founder — if you had a good week and set a personal record in the gym or on the track or wherever, that can still be a good week."

So if I start my morning with a gym routine, work on Buffer during the day and help two people during lunch, I have 4 chances to have a great day. It almost always works. ■

Joel is the founder of Buffer, a smarter way to share. Joel started Buffer on the side whilst working full-time, and within 4 months it was profitable and he quit his work. Joel blogs regularly about startups, life, learning and happiness at *joel.is* and loves to be in touch on Twitter at @joelgascoigne

Reprinted with permission of the original author. First appeared in *hn.my/happy* (joel.is)

KindleBerry Pi

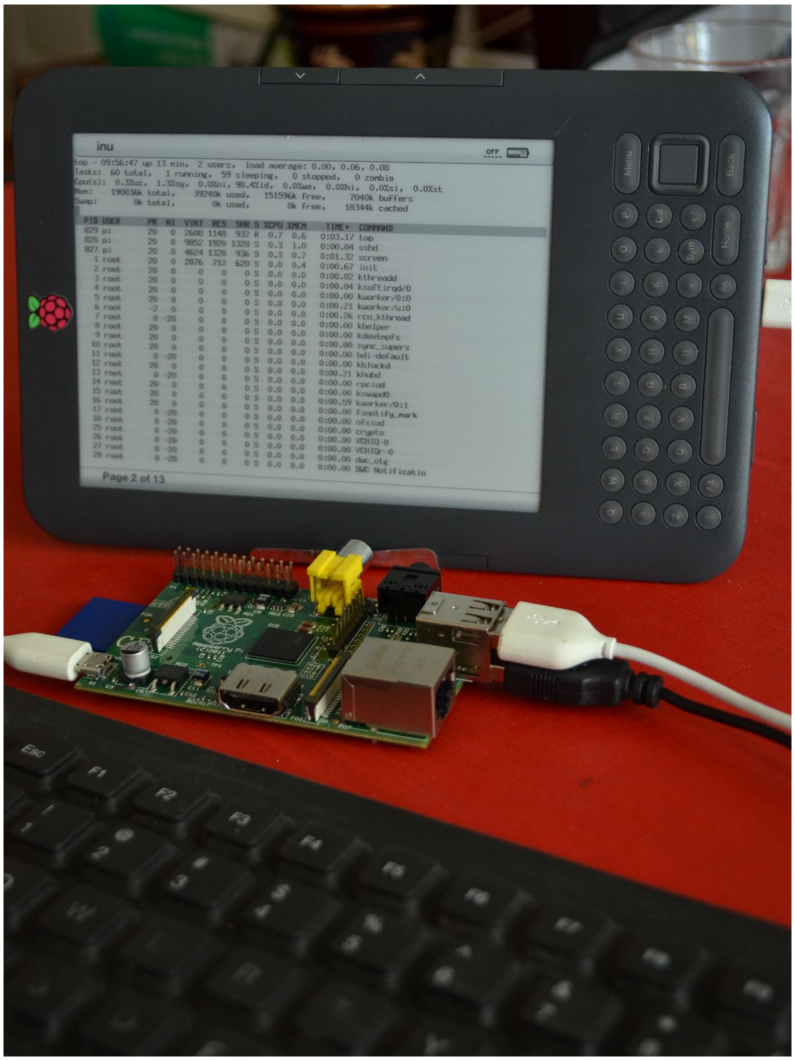
By GEOFFROY TREMBLAY

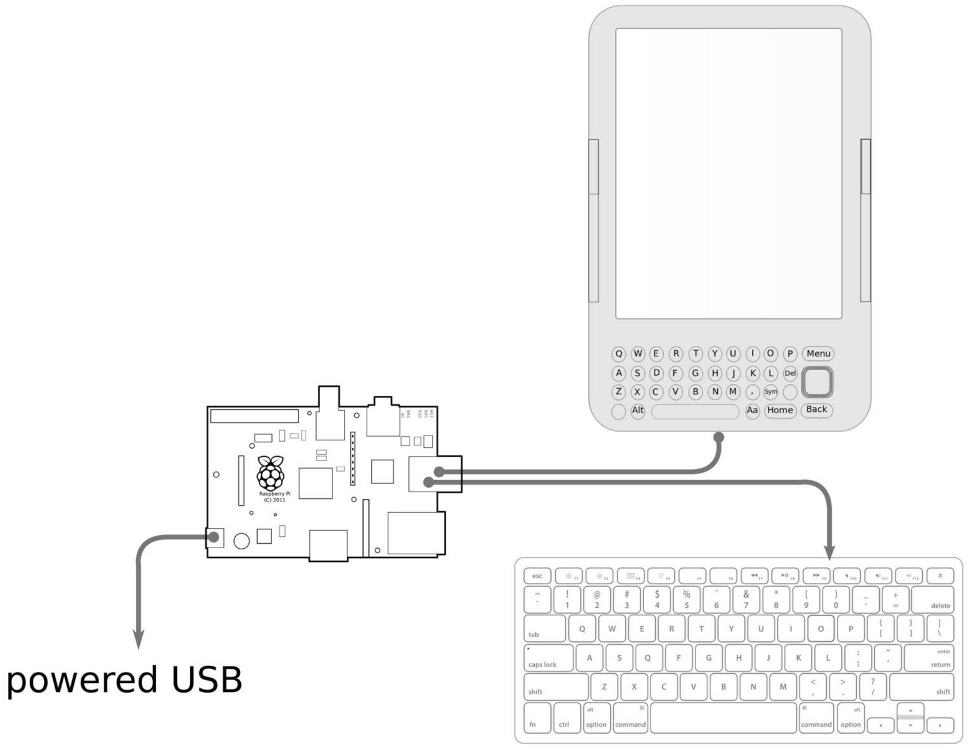
We left our little studio in the Kootenays last July, traveling throughout Europe to explore new media, spiritual centers, art, design and open source initiatives. I decided to go really minimal on the computer gear stuff, so I only packed my Kindle, a camera, an Android phone and, of course, my Raspberry Pi [raspberrypi.org]!

The Raspberry Pi, although a beautiful project and quite an electronic feat, can be a bit limiting as a main production machine, but I convinced myself I could use it as my main traveling computer.

The plan was to use a Kindle as a screen and connect it to the processing power of the Raspberry Pi while using an external keyboard to work comfortably. Since connecting an external keyboard to the Kindle seemed impossible at that point, I needed to use the Raspberry Pi as the "hub." The tinkering started and the KindleBerry Pi was soon born.

Although I ended up buying a laptop while traveling, all that dreaming and tinkering ended up working nicely, creating a really portable development platform. At the least it has become a proof of concept that could be used for other similar projects. So here is how you can create your very own KindleBerry Pi!


The KindleBerry Pi!



Voila, I can login into the Raspberry Pi with no problem, using the great display of the Kindle but sadly also using its limiting keyboard.

The main challenge now is to use the keyboard connected to the Raspberry Pi instead of the Kindle's. This is where the magic of GNU Screen comes into play! GNU Screen is a terminal multiplexer. If you don't know what a terminal is, well, I am not sure why you are reading this article in the first place, but let's say "GNU Screen is a terminal on steroid."

One of the nice functions of GNU Screen is that you can be multiple users on the same "screen" session. For instance, let's say you want to monitor what people do when they connect to your computer through ssh, or if you want to… well… screencast in a terminal environment (whatever enjoyment that would give you). Anyhow, I am not sure why there is a multiuser mode, but it is that ability that makes the KindleBerry Pi possible.

So what happens is that by using the keyboard connected to the Raspberry Pi, you can login into the Raspberry Pi with the Kindle and then share the same "screen" session so that you can use the keyboard connected on the Raspberry Pi. You will still need to use the Kindle keyboard to create that first connection, but once you're connected, you can use your mail keyboard.

Although GNU Screen comes to save the day, automating the whole process takes a few more steps.

Let's get started

What you will need to do this hack:

- One Kindle 3 (or two, if you end up breaking the first one)
- One Raspberry Pi
- Two micro USB to USB cables (one for power and one to connect the Kindle to the Raspberry Pi)
- One keyboard connected to the Raspberry pi
- Optional: A kindle stand (you can use an old audio tape box)
- Optional: A USB hub since the KindleBerry Pi has both ports in use when assembled

Hacking the Kindle

DISCLAIMER: You can brick (render unusable) your Kindle doing so. These are just pointers and I take no responsibility whatever you do with your kindle, or your life…

The first part — connecting the Kindle to the Raspberry Pi — is simple enough:

- Jail break the Kindle. [hn.my/jbk]
- Install a terminal emulator like this one. [hn.my/tek]
- Install UsbNetwork and make sure it is enabled. [hn.my/usbn]
- Connect the devices through USB,
- Do a quick `ifconfig usb0 192.168.2.1`.

USB network for the Raspberry Pi

First, we want to be able to use UsbNetworking when connecting the Kindle. When the Kindle is on usbNetworking, it assigns the IP 192.168.2.2 to its USB port. We then need the Raspberry Pi to automatically assign its USB port the IP 192.168.2.1. To do so, the first step is to add the following to your `/etc/network/interfaces`:

```
allow-hotplug usb0
mapping hotplug
script grep
map usb0
iface usb0 inet static
address 192.168.2.1
netmask 255.255.255.0
broadcast 192.168.2.255
up iptables -I INPUT 1 -s 192.168.2.1 -j ACCEPT
```

Automatic login on the Raspberry Pi

Now we need the Raspberry Pi to 1) make sure one user can login automatically and 2) initiate a screen multiuser session at boot time. We will be using the same user for the login at boot time and login with the Kindle.

For automatic login on debian (which is one of the main builds of the Raspberry Pi) you have to:

```
vim /etc/inittab
```

(or using any other editor) and comment out:

```
 #1:234:respawn:/sbin/getty 3840 tty1
```

and then add:

```
1:2345:respawn:/bin/login -f YOUR_USER_NAME
tty1/dev/tty1 2< & 1
```

Some readers have mentioned that using the following code worked better for them:

```
1:2345:respawn/sbin/agetty -a YOUR_USER_NAME -8
-s 38500 tty1 linux
```

That should do the trick. Now let's make sure GNU Screen starts automatically when the Raspberry Pi starts and whenever you login from the Kindle.

Bashrc

We now have to make sure the user that is automatically logged in will start a screen session. We also have to make sure that when you login with the Kindle, you don't start another screen session but instead join the already started screen session. Mileage might vary depending on your system here, so experiment with the code, but most of it should be in the `.bash_profile`. There is probably many other ways to go about it (use the bashrc and bash_profile, have more than one user, etc…), but this is one of the solutions I came up with.

Here is my `.bash_profile`:

```
if [ -z "$&;STY" ];  then
    exec screen -xR
fi
```

So once it's all in place, you should be able to fire up your KindleBerry Pi. Once the boot sequence is done, you can connect the Kindle in UsbNetwork mode through USB, moving into your shell and ssh into the Raspberry Pi. From there you should be able to use the keyboard connected to the Raspberry Pi and see the result on the Kindle! For some reason, sometimes I have to fiddle with the screen session, killing the extra one and then connecting to the main one.

You probably can work some simple password-less ssh with authentication key to save some time and add some cute scripting to simplify the whole connection process, but once you are connected you can then start using the Raspberry Pi keyboard.

It was really fun to create the KindleBerry Pi, and I even used that setup for few weeks. But I quickly realized that if I wanted to do anything productive, it was better to get myself a computer. My skill in command line and programming might have been too low to completely move to a shell-only lifestyle. So sadly the KindleBerry Pi at this point is only a proof of concept and could probably be really neat in an "end of the world" scenario! There still might be some minimal hardcore coder out there who would enjoy such a platform! ■

Geoffroy is a Web Designer and Yogi, who loves to tinker with opensource software and hardware. Living now in Berlin with his wife Melina, they are busy launching studio ponnuki, a joint creative project offering web design, online publication as well as yoga, massage and healing dance event!

stripe

Accept payments online.

MEMSET® HOSTING

Rent your IT infrastructure from Memset and discover the incredible benefits of cloud computing.

MINISERVER™ CLOUD COMPUTE

From £0.015p/hour
to 4 x 2.9 GHz Xeon cores
31 GBytes RAM
2.5TB RAID(1) disk

MEMSTORE™ CLOUD STORAGE

£0.07p/GByte/month or less
99.999999% object durability
99.995% availability guarantee
RESTful API, FTP/SFTP and CDN Service

SCAN THE CODE FOR MORE INFORMATION

MEMSET® HOSTING

CarbonNeutral® hosting

THE BRITISH ASSESSMENT BUREAU ISO9001 UKAS MANAGEMENT SYSTEMS 4307
ISO 9001: Quality

THE BRITISH ASSESSMENT BUREAU ISO14001 UKAS MANAGEMENT SYSTEMS 4307
ISO 14001: Environmental

THE BRITISH ASSESSMENT BUREAU ISO27001 THE BRITISH ASSESSMENT BUREAU
ISO 27001: Security

PC PRO Excellence AWARDS 2011 WINNER BEST WEB HOST

Best Managed Service WINNER The ISPAs 2011 www.ispaawards.org.uk

Find out more about us at
www.memset.com

or chat to our sales team on
0800 634 9270.