Generating Coupon Codes: Saving space by using a simple (and insecure) hashing technique

Ben Rosenberg (bar94@cornell.edu)

July 11, 2022

1 Abstract

Have you ever shopped online?[1] What about shopping in real life?[2] If either one of these applies to you, you may be somewhat familiar with the "coupon", a thing[3] which is used at the time of purchase to reduce the amount purchased by either some fraction (for our less consumption-inclined readers, this is generally a fraction of the form $n\%$, where $0 < n < 100$ such that the application of a coupon for $n\%$-off gives a new price of $P - \frac{n}{100}P$) or a fixed amount (e.g., of the form $¤n$ wherein ¤ denotes the relevant currency sign for the purchase).

There are many types of these coupons: some apply to specific items; others to an overall purchase; still others, to multiple specific items, perhaps linking (e.g.) a specific bottle of hair product x to body wash product y. This article aims mainly to reduce the space taken up by these inevitably vast numbers of coupons, specifically those meant for use within the realm of online shopping.[4] We implement a variety of simple mathematic techniques to turn the validation of coupons from a storage-intensive task into a more computationally intensive one, with only severe drawbacks for security.

2 Introduction

To put it simply, we want a way to generate coupon codes (alphanumeric strings, generally of length no less than 6 characters[5]) which works within these constraints:

- The codes should be unique.
- The codes should not be easily guessable (e.g., they should not have a pattern that is easy to guess, like $000, 001, \dots$).
- The codes should be easily checked without the use of a table to store them.

One method of making sure that things[6] are correct is to use a technique known as hashing, in which you hash something. For the sake of example, in this article we use a simple hash, but this can be changed as necessary in the future.

3 Previous work

https://420couponcodes.com

4 Implementation

Consider some coupon code $C = c_1c_2c_3 \dots c_n$, wherein $c_1, \dots, c_n$ are the characters which compose C. The simple hash we will use in this article is the hash which sums up the numerical representation of each of these characters, where we define the numer-

[1] Author's note: You probably have.

[2] Author's note: You have probably done this too.

[3] This is the scholarly-agreed-upon verbiage.

[4] There is probably room for improvement in the more physical, IRL- (in-real-life) shopping coupons as well, but due to their already negligible size as slips of paper, and the fact that they can decompose into other matter, their buildup is not perceived as being quite as harmful to the general public as is the buildup of their electronically-stored counterparts.

[5] 6 is a cool number.

[6] Again, the scholarly-agreed-upon term.

ical representation to be a mapping

$$\eta(c) = \begin{cases} \text{`A'} & 0 \\ \text{`B'} & 1 \\ \vdots & \vdots \\ \text{`Z'} & 25 \\ \text{`a'} & 26 \\ \vdots & \vdots \\ \text{`z'} & 51 \\ \text{`0'} & 52 \\ \vdots & \vdots \\ \text{`9'} & 61 \end{cases}$$

Astute readers might recognize this to be a perversion of the ASCII character code table to our inhumane uses.

An example of the use of this hashing function might be the string "JJJJJJJJJJ", or `'J'` $\times$ `10` for those who prefer speaking in Snake.[7] Since 'J' comes out to 10 under our complex hash given above (the proof is left as an exercise for the reader), this string hashes to $10 + \cdots + 10 = 10 \times 10 = 10^2 = 100$.

Now this is well and good, but how do we ensure that all strings we generate are equal under the hash? Is there even a way?[8]

Yes! This is a classic application of the balls and bins/stars and bars/balls and dividers/stars and dividers/stars and stripes/stripes and wipes/balls and wipes/wipe your balls problem, which is commonly taught in combinatorics. Say for instance that you have some 10 balls and want to put them into 3 bins. Note that you can divide these balls into groups using a grand total of $3 - 1$, also known as 2, dividers, like so:

$$\bullet \mid \bullet\bullet\bullet \mid \bullet\bullet\bullet\bullet\bullet\bullet$$

In this example, we have divided up the balls into a group of 1, a group of 3, and a group of 6. To check our work, we might add our numbers back together:

$$1 + 3 + 6 = 10$$

Wow, this works! Amazing! Notice anything? Yeah, of course you notice something. We can use this in our thing! Wow!

Basically, we can split whatever number we want to add to into groups of whatever by splitting it up into $\bullet$ and then splitting those $\bullet$ into $\cdots$ delineated by $|$. Simple. So we can just choose where to put these dividers and the number of balls in each stripe or whatever corresponds to the character code we use. Epic.

4.1 How do we automate this? I don't like to do work and this looks like work ew

The solution to your question is that we use math. More specifically, we just generate $n-1$ random numbers (recall that $\|C\| = n$) which will correspond to the locations at which we place our dividers. We can then calculate the contents of the code from that, like so (where the numbers generated are given by $\beta_1, \beta_2, \ldots, \beta_{n-1}$, and Ⱥ is the magic number to which we desire our η-map of C to add):

$$C = \eta^{-1}(\beta_1)\eta^{-1}(\beta_2 - \beta_1)\ldots\eta^{-1}(\beta_{n-1} - \beta_{n-2})\eta^{-1}(\text{Ⱥ} - \beta_{n-1}) \tag{1}$$

So now, we can generate a coupon code using the following steps:

1. Generate $n - 1$ random numbers $x_1, \ldots, x_{n-1}$ such that $0 \leq x_i \leq 61$
2. Convert this set of numbers into a coupon string satisfying the hash function using (1)

And we can then check that it works by:

1. Converting these characters back into numbers using η
2. Adding them up using your fingers[9]
3. Checking if the result is the equal to Ⱥ

5 Evaluation

Of course, this method is hilariously weak to actual attacks of any kind. In fact, if anyone figures out that you used this to cut down on memory, you will more likely be cut down from your job. Any one consumer finding out that this method is being used could spell the end for your business as it would mean that the coupon method is just broken – finding collisions with this hash function is easier than getting a paper accepted by SIGBOVIK.

That said, it does kind of work for its intended purpose, which is cutting down on space usage. If you had a lot of codes to generate, you now don't have to store them anymore, as you can check whether they are valid in real time!

[7] Fans of 1-indexing arrays might prefer `'J' ^ 10`, but Snake is more popular.

[8] Read on to find out!

[9] For people without fingers, Google Assistant can do addition using speech to text.

6 Future work

6.1 Avoiding suspicion

We present several possible methods to avoid consumer suspicion:

1. Choose varying values for ~~A~~.
    - For example, say your code is supposed to expire at the end of the month. Then, make ~~A~~ depend on the month during which the code is being used (e.g., year + month)
2. If it is possible that others can view the source code of your checking algorithm (e.g. if, for some unfathomable reason, you make the algorithm execute client-side), make sure you minify or obscurify the code so that it becomes unreadable.[10] There are actually two bonus features to this: one, you save even *more* space, and two, your boss will have a harder time firing you when looking at your code (this is important).

6.2 Oops, this doesn't really save that much space

Lastly, and most unfortunately, is the matter of preventing the multiple-time use of one-use coupons (without which extensions such as Honey might accumulate a database of coupons and notice the trend). The only way to really prevent the use of the same key multiple times is to keep track of what has and hasn't been used so far... which means saving keys rather than simply calculating them in real time and stuff.

This kind of defeats the entire purpose of our system. The best way to deal with this is likely to just turn these codes into base-62 numbers and store them in a hash table, but that wouldn't be fun. The author recommends the use of a trie-like data structure in which there are 62 children of each node (besides leaves), corresponding to addig a character to the end of a continuing word (e.g., traversing the trie starting at the root with the path 'a' $\rightarrow$ 'b' $\rightarrow$ 'c' would give the string "abc"). This might marginally cut down on repetition in the first few characters used but, of course, in the short run will be much worse than a simple set. Note also that in order to gain benefits of any kind the trie should be built in real time, rather than building an entire trie wherein existence is defined via some node attribute rather than a node's existence.

7 Conclusion

Don't use this method. ~~This article was sponsored by Honey. Honey is a free browser add on available on Google, Opera, Firefox, Safari, if it's a browser it has Honey. Honey automatically saves you money when you checkout on sites Like Amazon. Papa John's. Kohl's. Wherever you shop it's a good chance that Honey can save you money. All you have to do when you're checking out at these major sites click that little orange button and it will scan the entire internet And find discount codes for you. It takes two clicks to install Honey. Now anytime you checkout Honey will scan the entire internet and find coupon codes for you. If there is a coupon code they will find it, and if there's not a coupon code you can rest assured that you are getting the best price possible and there literally is not one available on the internet. If you install Honey right now you can save like 50 to 100 dollars on your Christmas shopping, doing nothing. There's literally no reason not to install Honey, it takes two clicks, 10 million people use it, 100,000 5 star reviews, unless you hate money you should install Honey. If you want to install it just go to joinhoney.com/mrbeast, that's joinhoney.com/mrbeast.~~

8 References

1. Hash codes: https://420couponcodes.com
2. Honey: Honey
3. ~~A~~: https://monopoly.hasbro.com/en-us

[10] That is to say, *more* unreadable than it already probably is.