



2 Mar 2017 | 4 min. (811 words)

A general construction for rolling hash functions

What is a rolling hash function?

A hash function is a function $h : S^\times \rightarrow F$ with S, F being some finite sets.

A rolling hash function is really a set of functions (h, u) , where u allows retroactively updated a symbol



$$h(\dots a \dots) \mapsto h(\dots a' \dots)$$



To put it more formally, a rolling hash function has an associated function $u : F \times S^2 \times \mathbb{N} \rightarrow F$, satisfying

$$u(n, a, a', h(\underbrace{\dots a \dots}_n)) = h(\underbrace{\dots a' \dots}_n)$$

An example

One of my favorite examples of a rolling hash function is the Rabin-Karp rolling hash.

Essentially, you pick some prime p and do following operation (over \mathbb{Z}_n):

$$h(\{c_n\}) = c_1 p^{k-1} + c_2 p^{k-2} + \dots + c_k$$

You might be able to figure out how you can construct u .

$$u(n, x, x', H) = H + (x' - x)p^{k-n}$$

So why isn't this a pretty good choice? Well, it's

1. Slow. Doing the exponentiation can be quite expensive.
2. It's relatively poor quality. This can be shown by looking at the behavior of the bits: Multiplication never affects lower bits, so it's avalanche effect is very weak.

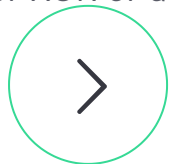
It has a really nice property though, you can use an arbitrary substring of the input and the substring's hash and replace it in $O(1)$, whereas most other rolling hash functions requires $O(n)$.

A general-purpose construction

So, is there a general way we can come up with these?

Well, what if we had some family of permutations, $\sigma_n : F \rightarrow F$?

Assume our F is an abelian group with some operation $+$ (could be addition or XOR or a multiplication).



Then, we can construct the hash function

$$h(\{x_n\}) = \sum_n \sigma_n(x_n)$$

Now, we can easily construct u :

$$u(n, x, x', H) = H - \sigma_n(x_n) + \sigma_n(x'_n)$$

XOR special case

As programmers, we love XOR, because it is so simple, and even better: Every element is its own inverse, under XOR.

Namely, under XOR, u would look like

$$u(n, x, x', H) = H \oplus \sigma_n(x_n) \oplus \sigma_n(x'_n)$$

Rabin-Karp as a special case

The interesting thing is that we can see Rabin-Karp as a special case, namely the family of permutations,

$$\sigma_n(x) \equiv xp^n \pmod{m}$$

The reason this is a permutation is because p is odd, hence p^n is odd, and must therefore have a multiplicative inverse in $\mathbb{Z}/m\mathbb{Z}$.

Now, why does p have to be a prime? Well, Every permutation must be distinct, $f(x) \equiv p^x \pmod{m}$ is a permutation itself (which can be shown relatively easily through basic group theory).

Statistical properties and qualities

- * Flipping a single bit will change the output, no matter what: If $x \neq x'$, $\sigma(x) \neq \sigma(x')$, because σ is a permutation.



as perfect collision property: Pick some n -bit sequence, s . The number of sequences colliding with s is independent of the choice of s (all equivalence classes have equal size).



Reduction to the permutation family

A lot of properties of the function are directly inherited from the quality of the permutation family. In fact, it can be shown that if the permutation family is a family of random oracles, the function is a perfect PRF.

Similarly, if the permutations are uniformly distributed over some input, the constructed function will be as well.

Almost all of the statistical properties, I can think of, has this kind of reductive property allowing us to prove it on the constructed property.

A good family of permutations

This is a really hard question. Analyzing a single permutation is easy, but analyzing a family of permutations can be pretty hard. Why? Because you need to show their independence.

If one permutation had some dependence on another, the hash function could have poor quality, even if the permutations are pseudorandom, when studied individually.

Parallelization

I'm the author of [SeaHash](#), and a big part of the design of SeaHash was to parallelize it.

And I could, pretty well. But with its design, there will always be a limit to this parallelization. In case of SeaHash, this limit is 4 (as there are 4 lanes). However, one could imagine hardware where such parallelization ideally should be say 32.

This construction allows for exactly this, without changing the specification. The function is adaptive: The implementation can choose whatever number of parallel lanes to hash in.

This can be done by simply breaking the input up in k strings, and hashing each individually, starting with n being the offset of the string.

This is a fairly nice property, as it also allows combination of threaded parallelization and ILP without any constant overhead. Say I'm hashing 4 TB of data, then I could spawn 4 threads (depending on your hardware) and still exploit the 4 CPU pipelines, while not hurting the performance of hashing only a few bytes.

Follow me on [Twitter](#) or [Github](#).

cryptography hash-functions

🐦 f g+

ALSO ON TICKI.GITHUB.IO

Skip Lists: Done Right · Ticki's blog

4 years ago · 13 comments

Skip lists are a wonderful data structure, but it is hard to get\right. This blog ...

An Atomic Hash Table · Ticki's blog

3 years ago · 2 comments

A design for a high-performance atomic and linearizable hash table.

A Critique of Rust's `std::collections` · ...

4 years ago · 3 comments

This is an in-depth critique of the bad parts of Rust's collection primitives.

You Are Doing I

3 years ago

This blog various i secure l

2 Comments

ticki.github.io

Disqus' Privacy Policy

Sandeep

Recommend


Tweet

Share

Sort by Best




Join the discussion...

- 

chmduquesne · 2 years ago

is in fact not obvious to me that seahash is a rolling hash. How should I update th
when the first byte of the window is removed?

^ | v · Reply · Share ›
- 

chmduquesne · 2 years ago

The rolling hash you used as an example of "simple" rolling hash is not a rabin fingerprint, but
it is the rolling hash that one usually use to explain the rabin-karp string search algorithm. The
rabin fingerprint is also a rolling hash, but it is the result of the division of the data, interpreted
as a polynom over GF(2), by another polynom, choosen to be irreducible over GF(2).

^ | v · Reply · Share ›

Subscribe

Add Disqus to your siteAdd DisqusAdd

Do Not Sell My Data