WIKIPEDIA

Rolling hash

A **rolling hash** (also known as recursive hashing or rolling checksum) is a hash function where the input is hashed in a window that moves through the input.

A few hash functions allow a rolling hash to be computed very quickly—the new hash value is rapidly calculated given only the old hash value, the old value removed from the window, and the new value added to the window—similar to the way a moving average function can be computed much more quickly than other low-pass filters.

One of the main applications is the Rabin–Karp string search algorithm, which uses the rolling hash described below. Another popular application is the rsync program, which uses a checksum based on Mark Adler's adler-32 as its rolling hash. Low Bandwidth Network Filesystem (LBFS) uses a Rabin fingerprint as its rolling hash.

At best, rolling hash values are pairwise independent[1] or strongly universal. They cannot be 3-wise independent, for example.

Contents

Polynomial rolling hash

The Rabin–Karp string search algorithm is often explained using a very simple rolling hash function that only uses multiplications and additions:

$$H = c_1 a^{k-1} + c_2 a^{k-2} + c_3 a^{k-3} + \ldots + c_k a^0,$$

where a is a constant, and $c_1, \ldots, c_k$ are the input characters (but this function is not a Rabin fingerprint, see below).

In order to avoid manipulating huge H values, all math is done modulo n. The choice of a and n is critical to get good hashing; see linear congruential generator for more discussion.

Removing and adding characters simply involves adding or subtracting the first or last term. Shifting all characters by one position to the left requires multiplying the entire sum H by a. Shifting all characters by one position to the right requires dividing the entire sum H by a. Note that in modulo arithmetic, a can be chosen to have a multiplicative inverse a^{-1} by which H can be multiplied to get the result of the division without actually performing a division.

Rabin fingerprint

The Rabin fingerprint is another hash, which also interprets the input as a polynomial, but over the Galois field GF(2). Instead of seeing the input as a polynom of bytes, it is seen as a polynom of bits, and all arithmetic is done in GF(2) (similarly to CRC32). The hash is the result of the division of that polynom by an irreducible polynom over GF(2). It is possible to update a Rabin fingerprint using only the entering and the leaving byte, making it effectively a rolling hash.

Because it shares the same author as the Rabin–Karp string search algorithm, which is often explained with another, simpler rolling hash, and because this simpler rolling hash is also a polynomial, both rolling hashes are often mistaken for each other. The backup software restic (https://restic.net/) uses a Rabin fingerprint for splitting files, with blob size varying between 512 bytes and 8MiB.[2]

Cyclic polynomial

Hashing by cyclic polynomial[3]—sometimes called Buzhash—is also simple, but it has the benefit of avoiding multiplications, using barrel shifts instead. It is a form of tabulation hashing: it presumes that there is some hash function h from characters to integers in the interval $[0, 2^L)$. This hash function might be simply an array or a hash table mapping characters to random integers. Let the function s be a cyclic binary rotation (or circular shift): it rotates the bits by 1 to the left, pushing the latest bit in the first position. E.g., $s(10011) = 00111$. Let $\oplus$ be the bitwise exclusive or. The hash values are defined as

$$H = s^{k-1}(h(c_1)) \oplus s^{k-2}(h(c_2)) \oplus \ldots \oplus s(h(c_{k-1})) \oplus h(c_k),$$

where the multiplications by powers of two can be implemented by binary shifts. The result is a number in $[0, 2^L)$.

Computing the hash values in a rolling fashion is done as follows. Let H be the previous hash value. Rotate H once: $H \leftarrow s(H)$. If c_1 is the character to be removed, rotate it k times: $s^k(h(c_1))$. Then simply set

$$H \leftarrow s(H) \oplus s^k(h(c_1)) \oplus h(c_{k+1}),$$

where c_{k+1} is the new character.

Hashing by cyclic polynomials is strongly universal or pairwise independent: simply keep the first $L - k + 1$ bits. That is, take the result H and dismiss any $k - 1$ consecutive bits.[1] In practice, this can be achieved by an integer division $H \rightarrow H \div 2^{k-1}$.

The backup software Attic uses a Buzhash algorithm with a customizable chunk size range for splitting file streams.[4]

Content-based slicing using a rolling hash

One of the interesting use cases of the rolling hash function is that it can create dynamic, content-based chunks of a stream or file. This is especially useful when it is required to send only the changed chunks of a large file over a network and a simple byte addition at the front of the file would cause all the fixed size windows to become updated, while in reality, only the first "chunk" has been modified.

The simplest approach to calculate the dynamic chunks is to calculate the rolling hash and if it matches a pattern (like the lower *N* bits are all zeroes) then it's a chunk boundary. This approach will ensure that any change in the file will only affect its current and possibly the next chunk, but nothing else.

When the boundaries are known, the chunks need to be compared by their hash values to detect which one was modified and needs transfer across the network.[5]

Content-based slicing using moving sum

Several programs, including gzip (with the `--rsyncable` option) and rsyncrypto, do content-based slicing based on an this specific (unweighted) moving sum:[6]

$$S(n) = \sum_{i=n-8195}^{n} c_i,$$

$$H(n) = S(n) \mod 4096,$$

where

- $S(n)$ is the sum of 8196 consecutive bytes ending with byte n (requires 21 bits of storage),
- c_i is byte i of the file,
- $H(n)$ is a "hash value" consisting of the bottom 12 bits of $S(n)$.

Shifting the window by one byte simply involves adding the new character to the sum and subtracting the oldest character (no longer in the window) from the sum.

For every n where $H(n) == 0$, these programs cut the file between n and $n+1$. This approach will ensure that any change in the file will only affect its current and possibly the next chunk, but no other chunk.

Computational complexity

All rolling hash functions are linear in the number of characters, but their complexity with respect to the length of the window (k) varies. Rabin–Karp rolling hash requires the multiplications of two k-bit numbers, integer multiplication is in $O(k \log k 2^{O(\log^* k)})$.[7] Hashing ngrams by cyclic polynomials can be done in linear time.[1]

Software

- rollinghashcpp (https://github.com/lemire/rollinghashcpp) is a free–software C++ implementation of several rolling hash functions
- rollinghashjava (https://github.com/lemire/rollinghashjava) is an Apache–licensed Java implementation of rolling hash functions

See also

- MinHash
- w–shingling

External links

- MIT 6.006: Introduction to Algorithms 2011– Lecture Notes – Rolling Hash (http://courses.csail.mit.edu/6.006/spring11/rec/rec06.pdf)

Footnotes

1. Daniel Lemire, Owen Kaser: Recursive *n*–gram hashing is pairwise independent, at best, Computer Speech & Language 24 (4), pages 698—710, 2010. arXiv:0705.4676.
2. "References — restic 0.9.0 documentation" (https://restic.readthedocs.io/en/stable/100_references.html#backups–and–deduplication). *restic.readthedocs.io*. Retrieved 2018–05–24.

3. Jonathan D. Cohen, Recursive Hashing Functions for *n*-Grams, ACM Trans. Inf. Syst. 15 (3), 1997.
4. "Data structures and file formats — Borg – Deduplicating Archiver 1.1.5 documentation" (https://borgbackup.readthedocs.io/en/stable/internals/data-structures.html#chunker-details). *borgbackup.readthedocs.io*. Retrieved 2018-05-24.
5. Horvath, Adam (October 24, 2012). "Rabin Karp rolling hash – dynamic sized chunks based on hashed content" (http://blog.teamleadnet.com/2012/10/rabin-karp-rolling-hash-dynamic-sized.html).
6. "Rsyncrypto Algorithm" (http://rsyncrypto.lingnu.com/index.php/Algorithm).
7. M. Fürer, Faster integer multiplication, in: STOC '07, 2007, pp. 57—66.
