Adler-32

From Wikipedia, the free encyclopedia

Adler-32 is a checksum algorithm which was invented by Mark Adler in 1995,[1] and is a modification of the Fletcher checksum. Compared to a cyclic redundancy check of the same length, it trades reliability for speed (preferring the latter). Adler-32 is more reliable than Fletcher-16, and slightly less reliable than Fletcher-32.[2]

Contents [hide]

History [edit]

The Adler-32 checksum is part of the widely used zlib compression library, as both were developed by Mark Adler. A "rolling checksum" version of Adler-32 is used in the rsync utility.

The algorithm [edit]

An Adler-32 checksum is obtained by calculating two 16-bit checksums *A* and *B* and concatenating their bits into a 32-bit integer. *A* is the sum of all bytes in the stream plus one, and *B* is the sum of the individual values of *A* from each step.

At the beginning of an Adler-32 run, *A* is initialized to 1, *B* to 0. The sums are done modulo 65521 (the largest prime number smaller than 2^{16}). The bytes are stored in network order (big endian), *B* occupying the two most significant bytes.

The function may be expressed as

$$
\begin{aligned}
A &= 1 + D_1 + D_2 + \dots + D_n \pmod{65521} \\
B &= (1 + D_1) + (1 + D_1 + D_2) + \dots + (1 + D_1 + D_2 + \dots + D_n) \pmod{65521} \\
&= n \times D_1 + (n-1) \times D_2 + (n-2) \times D_3 + \dots + D_n + n \pmod{65521} \\
\\
\mathit{Adler\text{-}32}(D) &= B \times 65536 + A
\end{aligned}
$$

where *D* is the string of bytes for which the checksum is to be calculated, and *n* is the length of *D*.

Example [edit]

The Adler-32 sum of the ASCII string "`Wikipedia`" would be calculated as follows:

Character	ASCII code	A	B
(shown as base 10)			
W	87	1 + 87 = 88	0 + 88 = 88
i	105	88 + 105 = 193	88 + 193 = 281
k	107	193 + 107 = 300	281 + 300 = 581
i	105	300 + 105 = 405	581 + 405 = 986
p	112	405 + 112 = 517	986 + 517 = 1503
e	101	517 + 101 = 618	1503 + 618 = 2121

d	100		618 + 100 = 718		2121 + 718 = 2839
i	105		718 + 105 = 823		2839 + 823 = 3662
a	97		823 + 97 = 920		3662 + 920 = 4582

```
A = 920  =  398 hex  (base 16)
   B = 4582 = 11E6 hex
   Output = 4,582 × 65,536 + 920 = 300286872 = 11E60398 hex
```

The modulo operation had no effect in this example, since none of the values reached 65521.

Comparison with the Fletcher checksum [edit]

The first difference between the two algorithms is that Adler-32 sums are calculated modulo a prime number, whereas Fletcher sums are calculated modulo 2^4-1, 2^8-1, or $2^{16}-1$ (depending on the number of bits used), which are all composite numbers. Using a prime number makes it possible for Adler-32 to catch differences in certain combinations of bytes that Fletcher is unable to detect.

The second difference, which has the largest effect on the speed of the algorithm, is that the Adler sums are computed over 8-bit bytes rather than 16-bit words, resulting in twice the number of loop iterations. This results in the Adler-32 checksum taking between one-and-a-half to two times as long as Fletcher's checksum for 16-bit word aligned data. For byte-aligned data, Adler-32 is faster than a properly implemented Fletcher's checksum (e.g., one found in the Hierarchical Data Format).

Example implementation [edit]

In C, an inefficient but straightforward implementation is :

```c
const int MOD_ADLER = 65521;

uint32_t adler32(unsigned char *data, size_t len) /* where data is the location of
the data in physical memory and
                                                       len is the length of the data
in bytes */
{
    uint32_t a = 1, b = 0;
    size_t index;

    /* Process each byte of the data in order */
    for (index = 0; index < len; ++index)
    {
        a = (a + data[index]) % MOD_ADLER;
        b = (b + a) % MOD_ADLER;
    }

    return (b << 16) | a;
}
```

See the zlib source code for a more efficient implementation that requires a fetch and two additions per byte, with the modulo operations deferred with two remainders computed every several thousand bytes.

Advantages and disadvantages [edit]

- Like the standard CRC-32, the Adler-32 checksum can be forged easily and is therefore unsafe for protecting against *intentional* modification.
- It's faster than CRC-32 on many platforms.[3]
- Adler-32 has a weakness for short messages with few hundred bytes, because the checksums for these messages have a poor coverage of the 32 available bits.

Weakness [edit]

Adler-32 is weak for short messages because the sum *A* does not wrap. The maximum sum of a 128-byte message is 32640, which is below the value 65521 used by the modulo operation, meaning that roughly half of the output space is unused, and the distribution within the used part is nonuniform. An extended explanation can be found in RFC 3309, which mandates the use of CRC32C instead of Adler-32 for SCTP, the Stream

Control Transmission Protocol.[4] Adler-32 has also been shown to be weak for small incremental changes,[5] and also weak for strings generated from a common prefix and consecutive numbers (like auto-generated label names by typical code generators).[6]

See also [edit]

- List of hash functions

Notes [edit]

1. ^ First appearance of Adler-32 (see ChangeLog and adler32.c)
2. ^ Revisiting Fletcher and Adler Checksums
3. ^ Theresa C. Maxino, Philip J. Koopman (January 2009). "The Effectiveness of Checksums for Embedded Control Networks" (PDF). IEEE Transactions on Dependable and Secure Computing.
4. ^ RFC 3309
5. ^ http://cbloomrants.blogspot.com/2010/08/08-21-10-adler32.html
6. ^ http://www.strchr.com/hash_functions

External links [edit]

- RFC 1950 – specification, contains example C code
- ZLib – implements the Adler-32 checksum
- RFC 3309 – information about the short message weakness and related change to SCTP
- Catalogue of parametrised CRC algorithms

Categories: Checksum algorithms