Checksums

A checksum is a small value calculated from a sequence of data. It helps us detect accidental changes during storage or transmission.

The sender:

1. Calculates the checksum of the original data.
2. Sends both the data and the checksum.

The receiver:

1. Calculates the checksum of the received data.
2. Compares it with the received checksum.

If the two values differ, an error has been detected. If they are equal, the data may be correct, but this is **not guaranteed**: different data sequences can have the same checksum.

Unlike a Hamming code, these checksum methods do not identify or correct the damaged bit.

A simple checksum: sum modulo 256

Suppose that our data contains three bytes, written here as decimal numbers:

```
10, 20, 30
```

We add their values and keep the remainder after division by 256:

```
checksum = (10 + 20 + 30) mod 256 = 60
```

The `mod` operation gives the remainder after integer division. It keeps the checksum between 0 and 255, so the checksum fits into one byte.

For example:

```
(200 + 100) mod 256 = 300 mod 256 = 44
```

Detecting a changed byte

Suppose that 20 changes to 21 during transmission:

```
Original data: 10, 20, 30 → checksum = 60
Received data: 10, 21, 30 → checksum = 61
```

The checksums differ, so the receiver detects an error.

A limitation: order does not matter

What happens if two bytes exchange positions?

```
Original data: 10, 20, 30 → checksum = 60
Received data: 20, 10, 30 → checksum = 60
```

The checksum stays the same because changing the order does not change the sum.

Some changes can also cancel each other:

```
Original data: 10, 20, 30 → checksum = 60
Received data: 11, 19, 30 → checksum = 60
```

One value increased by 1 and another decreased by 1. This error remains undetected.

Adler-32: using two sums

Adler-32 uses two running sums, called **A** and **B**.

- **A** accumulates the byte values.
- **B** accumulates the successive values of A.

The second sum makes the result depend on the order of the bytes.

Initially:

```
A = 1
B = 0
```

For each byte, perform these operations in this order:

```
A = (A + byte) mod 65521
B = (B + A) mod 65521
```

Important: B uses the newly updated value of A.

The prime number 65521 is part of the algorithm. In our small examples, the sums remain below this number, so the modulo operation does not change them.

Example: calculating the checksum

Our data is:

```
10, 20, 30
```

Step	Current byte	Calculation of A	Calculation of B
Start	—	A = 1	B = 0
1	10	A = 1 + 10 = 11	B = 0 + 11 = 11
2	20	A = 11 + 20 = 31	B = 11 + 31 = 42
3	30	A = 31 + 30 = 61	B = 42 + 61 = 103

The final values are:

```
A = 61
B = 103
```

Adler-32 combines them into one 32-bit value:

```
checksum = B × 65536 + A
         = 103 × 65536 + 61
         = 6750269
```

B occupies the upper 16 bits, and A occupies the lower 16 bits.

Detecting a change in order

Now exchange the first two bytes:

```
20, 10, 30
```

Step	Current byte	A	B
Start	—	1	0
1	20	21	21
2	10	31	52
3	30	61	113

Compare the final results:

Data	Final A	Final B
10, 20, 30	61	103
20, 10, 30	61	113

A is unchanged, but **B is different**. Therefore, the Adler-32 checksums are different, and this error is detected.

Why? The first byte contributes to A earlier, so its effect is included in B more times than the effect of a later byte.

What can a checksum guarantee?

Adler-32 is used in the zlib compressed data format. It provides a practical way to detect accidental data corruption.

However:

- It does not detect every possible error.
- It does not locate or correct the damaged data.
- It does not protect against deliberate modification.

Different inputs that produce the same checksum are called a **collision**.

A matching checksum means that no error was detected. It does not prove that no error occurred.

Reference: RFC 1950 — ZLIB Compressed Data Format Specification

From:
https://edu.iit.uni-miskolc.hu/ - **Institute of Information Science - University of Miskolc**

Permanent link:
https://edu.iit.uni-miskolc.hu/tanszek:oktatas:techcomm:checksum

Last update: **2026/10/19 17:55**