

Checksums

Version 1.0 / October 2020

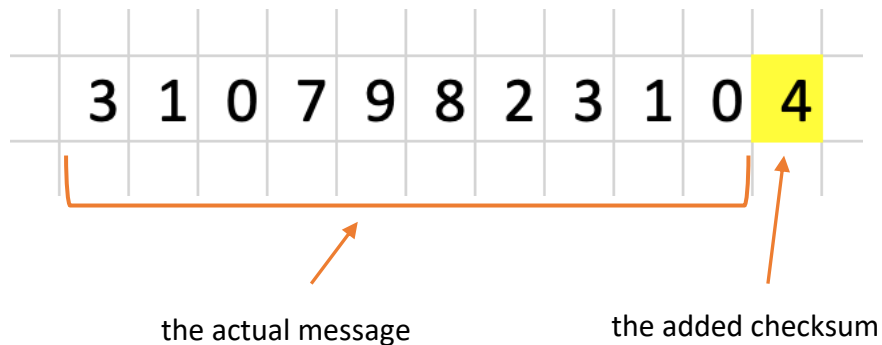
Doug Lichtman

Computers send enormous amounts of information over the Internet, and usually the information arrives fully intact. Sometimes, however, a power failure, a problem with computer memory, or some other issue causes data to be corrupted. Computers therefore need a strategy by which to detect rare, but problematic, errors. One such strategy is the use of a mathematical confirmation called checksums.

Here's a simple example.

Imagine that my computer wanted to send your computer the phone number 3-1-0-7-9-8-2-3-1-0. If my computer simply sent the numbers, your computer would have no way of knowing whether it received the message correctly. It would receive some numbers, and it would simply have to assume that the received message arrived intact.


But now imagine that my computer were to add one digit to the message. The new digit would be calculated by adding up the intended numbers, dividing by 10, and then reporting the remainder. Here, for example, the new digit would be a 4, because $3+1+0+7+9+8+2+3+1+0$ is equal to 34, and 34 divided by 10 gives a remainder of 4.



A computer receiving this message would now detect any single error in the received message. Imagine, for example, that the receiving computer received the message correctly, as shown below. The receiving computer would calculate the checksum, that calculation would match the received checksum, and the computer would know the message was accurate.

sent	3	1	0	7	9	8	2	3	1	0	4	
received	3	1	0	7	9	8	2	3	1	0	4	4


calculated by the receiving computer;
it matches!



But now consider any example where one digit of the messages was corrupted. The calculated checksum (shown in yellow) would not match the proposed checksum, and so the receiving computer will know that there was a mistake!

sent	3	1	0	7	9	8	2	3	1	0	4		
received	3	1	0	7	9	8	2	3	1	0	4	4	match
received	3	1	0	6	9	8	2	3	1	0	4	3	mismatch
received	3	1	0	7	9	8	9	3	1	0	4	1	mismatch
received	3	1	0	7	2	8	2	3	1	0	4	7	mismatch
received	3	1	0	7	9	8	2	3	8	0	4	1	mismatch
received	3	5	0	7	9	8	2	3	1	0	4	8	mismatch
received	3	1	7	7	9	8	2	3	1	0	4	1	mismatch

calculated by the receiving computer



HOMEWORK

Question 1.

The examples above all involve 10-digit numbers. Can this approach work on longer numbers? Is there any limit to how long the number can be?

Question 2.

The examples above consider only single mistakes. Would this same approach catch mistakes if two digits were corrupted? Always? Sometimes? Never?

Question 3.

Suppose I tell you that computer data is corrupted very rarely. For example, out of every million digits sent, imagine that only one digit will be wrong. Knowing that, would you suggest that I use the checksum approach to catch errors? What's good about this approach? What's risky?

Optional Investigation 1.

If you are interested in learning about other strategies that computers use to catch and correct errors in a data stream, this video summarizes several mind-blowing approaches. Be sure to check out the part about "checksum grids" at the end of the video.

<https://youtu.be/z684BB0M5CA>

Optional Investigation 2.

One of my favorite books proposes a strategy for detecting two errors, but I am skeptical that it actually works. This video summarizes the strategy and asks viewers to write Python code to test it. But we are mathematicians, so: can you use your knowledge of modular math to show either that this strategy will work, or that it definitely fails? If it fails, can you use your knowledge of modular math to fix it?

<https://youtu.be/qwIkTbX-klo>