Hamming Code

And why every bit counts

Basic principles

And why knowing a little bit about it is so important

Wait...Who is Hamming?

You must be wondering... Who was Hamming, and why would you even care? Well, Richard Hamming was the genius behind the Hamming Code, a tool that makes sure the data you send or receive doesn’t get scrambled along the way. Imagine sending an important message, and a small glitch ruins everything. Hamming's method catches those tiny errors, fixing them before they cause any trouble. In a world where reliable data transmission is everything, Hamming’s work keeps things running smoothly, bit by bit!

After all, why is this code so important?

The Hamming Code is important because it was one of the first error correcting codes that could both detect and correct errors in data transmission. Imagine sending a message over a noisy channel, like a weak internet connection or a wireless network. If a single bit gets flipped due to interference, the entire message could be corrupted. The Hamming Code smartly adds redundancy to the data, allowing systems to pinpoint where an error occurred and fix it without needing to resend the data.

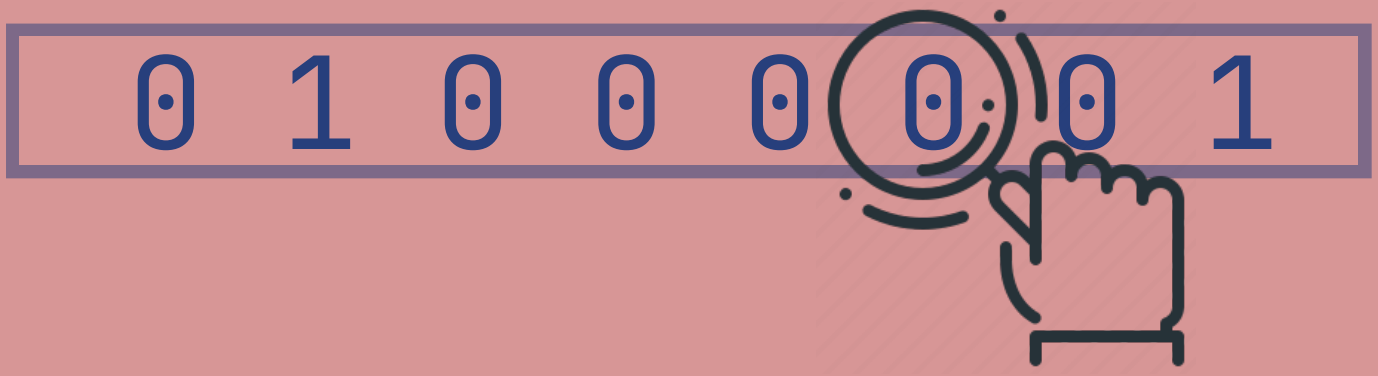
What is the binary system

Before talk about the Hamming code and these things, we need to talk abou the binary system. The binary code is a system of representing data using only two digits: **0** and **1**. It is the fundamental language of computers and digital systems because these two states correspond to the on and off states of electrical signals in electronic devices.

But what does this have to do with data transmission and the Hamming code?

How the data is transmited

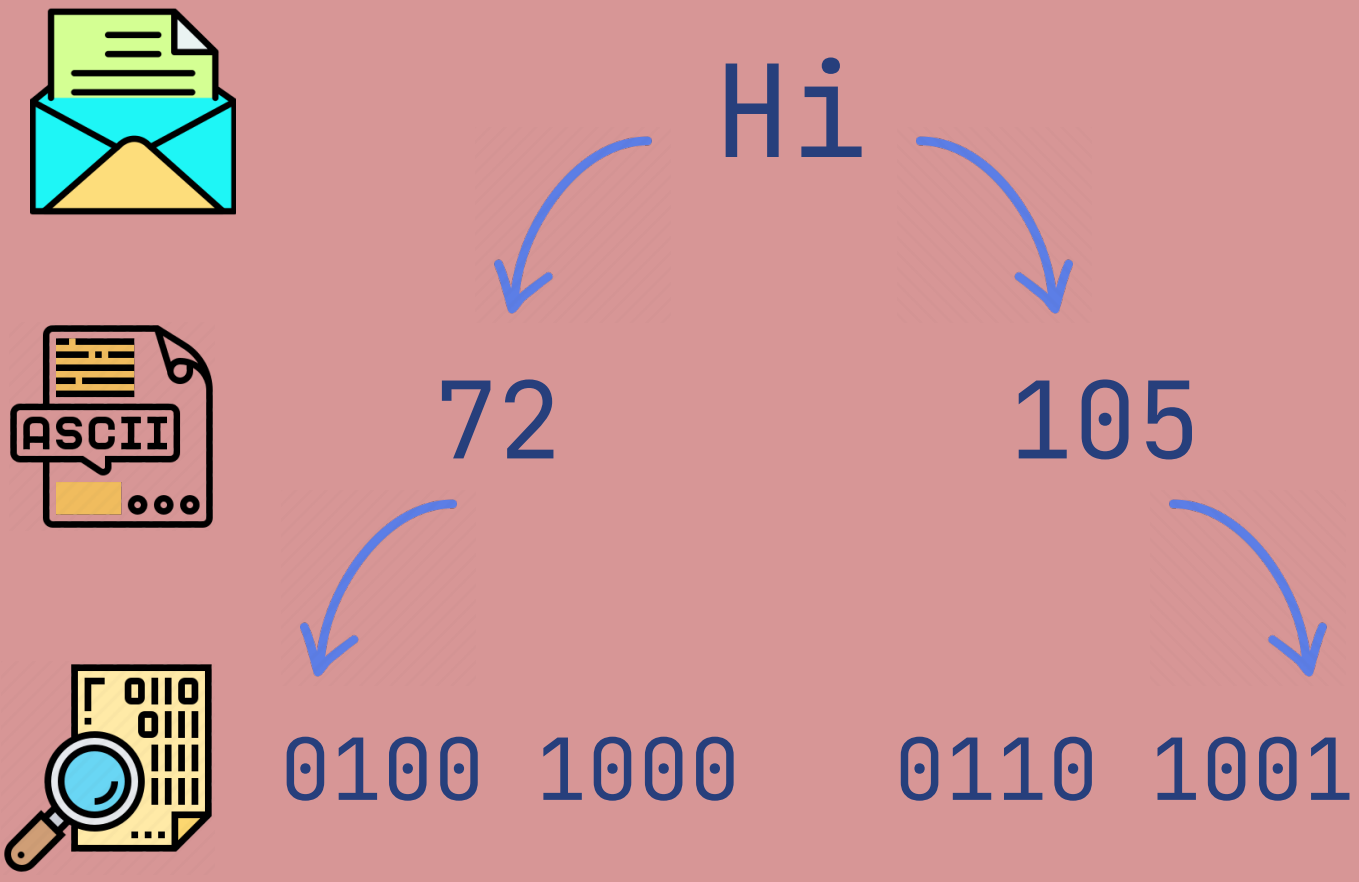
Messages transmitted over the internet are represented in binary. Let’s see it!



Every piece of data, whether it’s text, images, or video, is ultimately converted into binary code. For example, the letter "A" in ASCII (a common text encoding) is represented as 65 in decimal, which is 01000001 in binary.

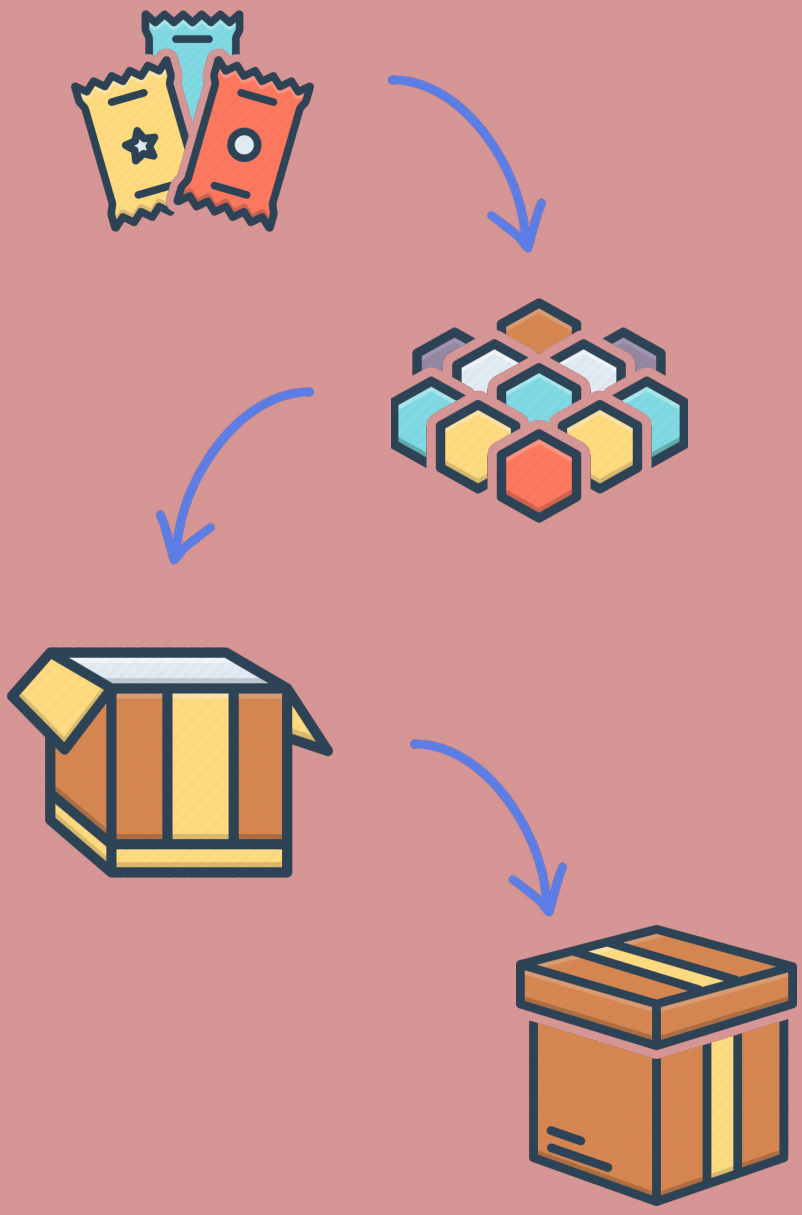
Encoding

When you send a message (like an email), the computer converts the text into binary using a specific encoding standard (such as ASCII or UTF-8). Each character is transformed into its corresponding binary value.



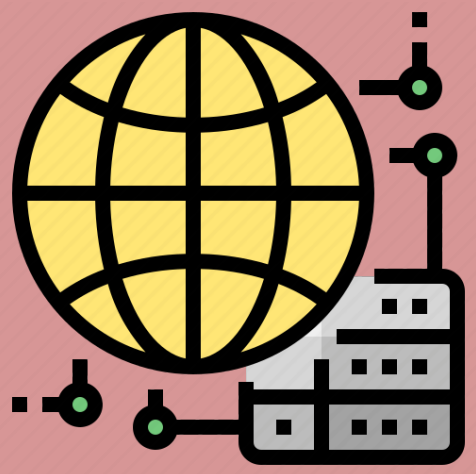
Packets

Once the data is in binary form, it’s split into smaller pieces called packets. Each packet contains a portion of the message along with metadata, including the sender's and recipient's IP addresses, to ensure it reaches the correct destination.



Transmission

These packets are transmitted over the internet through a series of routers and switches. They can take different paths to reach their destination, allowing for efficient use of the network. The packets travel as electrical signals over cables, as light signals in fiber optics, or as radio waves in wireless communication.



Destination and Reassemble

When the packets arrive at the destination, the receiving computer reassembles them into the original message. This process involves checking for any errors using methods like checksums or error-correcting codes (like Hamming Code) to ensure the data is accurate.

And finally, displaying the message

Finally, the binary data is converted back into a human-readable format. For example, the binary data that represents the original text is translated back into characters that you can read on your screen.

0100 1000 0110 1001

72 105

Hi

Now, the practice

One byte at a time!

Let’s codify

Like we just see, first of all, we choose the message we want to send. Today, we’ll send the word ‘**bit**’. Going to the ASCII table, we identify the decimal numbers that are equal to the letters we need, and its correspondent binary version.

b - 98 → 0110 0010
i - 105 → 0110 1001
t - 116 → 0111 0100

Now, the first Hamming step

After we have our message entirely in binary, we must organize it. This organization consist in position all the bits correctly in a vector, back to front, jumping the positions that are power of two*.

0110 0010 0110 1001 0111 0100

0	1	1	0	0	0	1	0	0	1	1	0	1		0	0	1	0	1	1	1		0	1	0		0		
29	28	27	26	25	24	23	22	21	20	19	18	17	**16**	15	14	13	12	11	10	9	**8**	7	6	5	**4**	3	**2**	**1**

* A power of two is a number that can be expressed as 2^n, where ***n*** is a non-negative integer. This means that powers of two are the result of multiplying the number 2 by itself ***n*** times. For example, in the case of our example:

$2^0 = 1$
$2^1 = 2$
$2^2 = 4$
$2^3 = 8$
$2^4 = 16$

Calculating parity

After we have put the binary numbers beautifully in order in the vector, we have to calculate the Number of Hamming, that will fill the spaces we just jumped over. And how we do that?

0	1	1	0	0	0	1	0	0	1	1	0	1		0	0	1	0	1	1	1		0	1	0		0		
29	**28**	**27**	26	25	24	**23**	22	21	**20**	**19**	18	**17**	**16**	15	14	**13**	12	**11**	**10**	**9**	**8**	7	**6**	5	**4**	3	**2**	**1**

The first step is identify what positions are occupied with the binary *'1'*, and convert the decimal number that correspond to this positions to binary.

6 - 110
9 - 1001
10 - 1010
11 - 1011
13 - 1101
17 - 10001
19 - 10011
20 - 10100
23 - 10111
27 - 11011
28 - 11100

Still calculating parity

Now, with the numbers in hand, we can see some of them have more or less digits. This is absolutely normal, some numbers need less bits to be represented. All we need to do is make them all equal in quantity of digits, adding a zero in the left positions, until fill all the spaces we need.

6 - 00110
9 - 01001
10 - 01010
11 - 01011
13 - 01101
17 - 10001
19 - 10011
20 - 10100
23 - 10111
27 - 11011
28 - 11100

Almost finishing

Now, with the numbers in position, we have to calculate the parity in fact. We'll work with the ***even parity,*** where if there are a number even of binarys '1', we would assume the parity is '0', and if the number of binarys '1' is odd, we would assume the parity is '1'. Let's try?

6	-	0011	0
9	-	0100	1
10	-	0101	0
11	-	0101	1
13	-	0110	1
17	-	1000	1
19	-	1001	1
20	-	1010	0
23	-	1011	1
27	-	1101	1
28	-	1110	0

We will analyze the columns of the matrix of numbers we just created, and count all the times that the digit '1' appears in each one, and write the result right under it. And in the way we just talked:

Even: 0

Odd: 1

7 times (seven is a odd number). So, parity is... 1

Getting Close

So, let’s calculate the parity to all the columns of our matrix? (You can call Neo if you need any help)

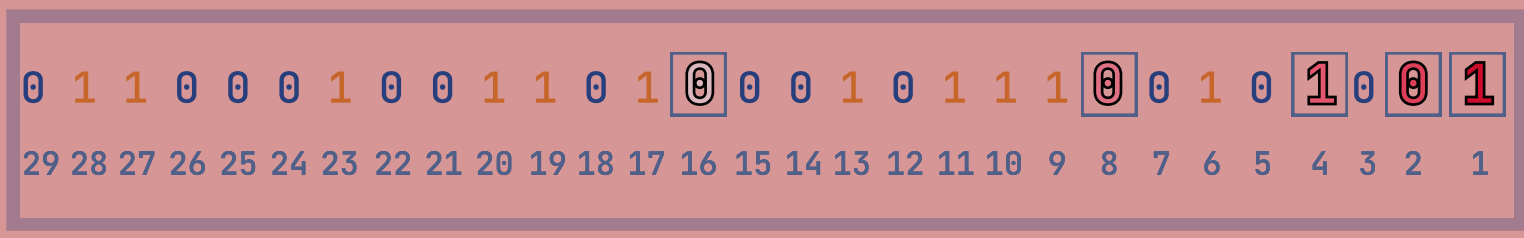
```
6  - 0 0 1 1 0
9  - 0 1 0 0 1
10 - 0 1 0 1 0
11 - 0 1 0 1 1
13 - 0 1 1 0 1
17 - 1 0 0 0 1
19 - 1 0 0 1 1
20 - 1 0 1 0 0
23 - 1 0 1 1 1
27 - 1 1 0 1 1
28 - 1 1 1 0 0
-------------
     0 0 1 0 1
```

That’s it! We just found the Hamming Number!

Let’s go!

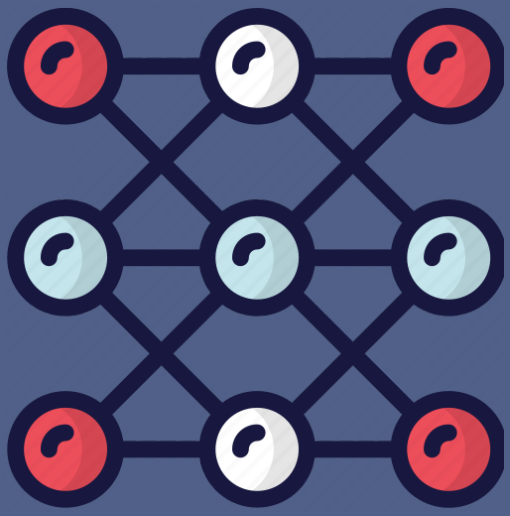
Now that we’ve found the Hamming Number, we need to positionate it in the vector with our binary message. We’ll fill the positions that we jumped while positioning the message, the positions that are power of two.

0 0 1 0 1



It’s just put the binary digits of the Hamming Number, in order, in the positions that are empty. And now... We’re ready to send our message through the internet!

The ins and outs



Mastering the maze

Receiving data

After receiving the message with the Hamming Number, the parity bits and all the thing, we can easily verify if something was alter in the way until arrive to destiny.

0	1	1	0	0	0	1	0	0	1	1	0	1	0	0	0	1	0	1	1	1	0	0	1	0	1	0	0	1
29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1

We just recalculte the Hamming Number, the same way we did before, putting all the positions that have the binary ‘1’ in a matrix with its binary representation, and calculating the parity of the columns, but now, considering the bits of the Hamming Number we calculate before, and did put in the power of two positions. Let’s do it?

Checking...

```
1  - 0 0 0 0 1
4  - 0 0 1 0 0
6  - 0 0 1 1 0
9  - 0 1 0 0 1
10 - 0 1 0 1 0
11 - 0 1 0 1 1
13 - 0 1 1 0 1
17 - 1 0 0 0 1
19 - 1 0 0 1 1
20 - 1 0 1 0 0
23 - 1 0 1 1 1
27 - 1 1 0 1 1
28 - 1 1 1 0 0
-------------
     0 0 0 0 0
```

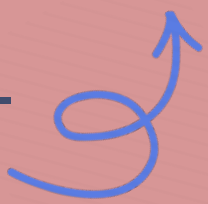
And take a look of what we have here... That's right! The parity number is '0' in all the bits. That means the transfer of the data occurred perfectly, without interferences.

What if something went wrong?

Let’s imagine that an error occured during the transmission, and one bit was affected, and changed it value. In our example, the bit in the position 13. Its value was ‘1’, so we was considering it in the matrix. If we supose its value changed to ‘0’, it wouldn’t be in our matrix, right? So, let’s remove it.

```
1  - 0 0 0 0 1
4  - 0 0 1 0 0
6  - 0 0 1 1 0
9  - 0 1 0 0 1
10 - 0 1 0 1 0
11 - 0 1 0 1 1
17 - 1 0 0 0 1
19 - 1 0 0 1 1
20 - 1 0 1 0 0
23 - 1 0 1 1 1
27 - 1 1 0 1 1
28 - 1 1 1 0 0
-------------
     0 1 1 0 1
```

Wait a minute... The parity is changed. What number is this?



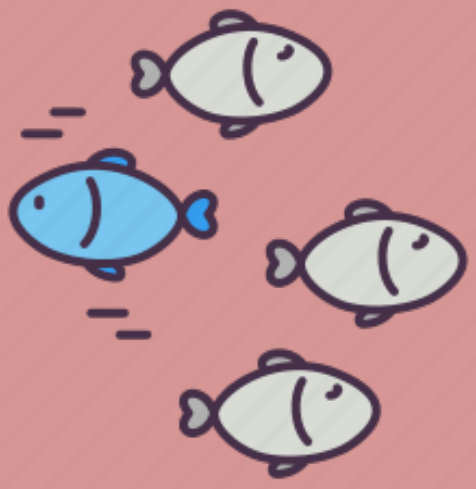
An error GPS!

Recapitulating... The parity now is **01101**, instead of all zero. And what this means? The more attent eyes maybe have already noticed. This binary correspond exactly to the number **‘13’** in decimal, wich is the position where the error occured. And knowing this information, we don’t need ask to the sender to send the message again. The binary system has only two possible values, and if it is **‘0’** and the Hamming Code said this is wrong... All is needed to do is invert this value to his opposite! Correcting the message, and so it can be decodified of binary to his decimals, and of decimals to the correspondent carachters!

0 1 1 0 1 → 13

Some attention points

The Hamming Code was projected to identify and correct just **one** error in one bit. If there are more bits that suffered some interfference, the Hamming Code may not be able to indicate the correct positions of this errors, or idenfity there are more than one error in the transmition. Hamming Code is usually used in association with another security protocols, like CRC (Cyclic Redundancy Check), and BCC (Block Check Character), to identify another possible alterations.



How to learn more

If you want to learn more about the Hamming Code, I can help you!

How to find help!

If you have any type of questions, or would like to talk a little bit more about the subjects you've found in this book, you can contact me!

nathalielaise@gmail.com

instagram.com/grey.wind_

github.com/NathaliePatzer

linkedin.com/in/nathaliepatzer

Thank you! ;)