Código Hamming

É um código de detecção e correção de erros.

Detecção e correção de erros é muito utilizado na transmissão de informação em redes de computadores, por exemplo, mas também é utilizado em armazenamento de informação.

Então quando trabalhamos com armazenamento, é importante que tenhamos segurança naquela informação armazenada, ou seja, saber se algum bit foi trocado ou não, e a detecção de erros é importante. Porque quando falamos de transmissão de informação em uma rede de computadores, se algum erro for detectado, simplesmente pede-se para retransmitir aquela informação. Entretanto, quando falamos de armazenamento interno na nossa máquina, essa retransmissão não é possível.

A partir do momento em que a informação foi armazenada, é interessante detectar que um erro aconteceu no armazenamento e na hora que a informação foi recuperada, detecta-se esse erro e o corrige antes de utilizar aquela informação, já que é inviável pedir para retransmitir uma informação armazenada durante algum tempo.

Em canais de comunicação de dados é suficiente ter somente a habilidade de detectar erros. Já nos dispositivos de memória -- como já foi dito --, além da habilidade de detectar, é importante corrigir esses erros. Para isso, temos o "Código Hamming (CH)".

O CH é um código bem antigo e que não é mais utilizado hoje em dia. Atualmente o código que é utilizado chama-se "CRC".

O CH utiliza, para a detecção de erros, "Bits de redundância".

Bits de redundância nada mais são do que bits além da palavra que será armazenada, que são calculados, colocados e armazenados juntamente com a palavra. Por quê? Porque é através desses bits de redundância que é possível verificar posteriormente se houve algum erro nesse armazenamento.

Obrigatoriamente, teremos alguns bits para a palavra que será armazenada na memória, alguns bits de redundância, e o código final armazenado será a palavra original mais as redundâncias.

Fórmula:

Palavra de memória = m bits

Redundância = r bits

Código final = n bits = m + r

Exemplo: Uma palavra de 8 bits precisa de 4 bits de redundância para detectar e corrigir 1 erro. Significa que teremos o "Código final" de armazenamento de 12 bits. Desta forma, conseguimos detectar e corrigir o erro.

Como o Código Hamming funciona

Posição na palavra		
12	1100	dado8
11	1011	dado7
10	1010	dado6
9	1001	dado5
8	1000	redundância4
7	0111	dado4
6	0110	dado3
5	0101	dado2
4	0100	redundância3
3	0011	dado1
2	0010	redundância2
1	0001	redundância1

A palavra de 8 bits que usaremos no exemplo acima é a palavra "01001011". Significa que a informação inicial que será armazenada tem 8 bits.

> Obs.: A contagem sempre começa da direita para a esquerda. Cada bit é uma posição (informação).

Entretanto, além dessa informação (palavra), tem os bits de redundância que precisam ser calculados e inseridos juntos com a informação para poder, depois, verificar se houve algum tipo de erro.

Ao todo (no exemplo acima) temos 8 bits da palavra mais 4 bits de redundância, totalizando 12 bits no código final que será armazenado.

Por isso temos a tabela acima, que representa os 12 bits do código final.

Na segunda coluna da tabela, está o binário correspondente a cada uma das posições. Portanto, a posição "1" é "0001" -- o número "1" em binário é "0001"--, e assim por diante.

Na terceira coluna está a posição do código final onde serão colocados os bits de redundância (que ainda será calculado) e os bits de dados, que são as informações da palavra utilizada (os bits da informação inicial, no caso do exemplo, são todos os bits de "01001011").

Não será colocado os 8 bits da palavra e depois os 4 bits de redundância, ambos serão misturados / mesclados, ou seja, um pouco de dado e um pouco de redundância.

Como os bits de redundância ajudarão da detecção de erros?

Vamos pensar que escrevi um e-mail, e nesse e-mail estamos combinando que o código que utilizaremos é a contagem da quantidade de letras "a" que tenho após escrever o e-mail. Portanto, escrevi o meu e-mail inteiro, cheguei no final, contei quantas letras "a" tem e coloquei no final do e-mail.

Na hora que mando o e-mail para alguém, essa pessoa receberá o e-mail e o número de letras "a" que foram contadas nesse e-mail.

Portanto, quando a pessoa receber o e-mail, ela receberá ambas as informações, ou seja, o texto original mais a redundância.

> Obs.: "Redundância" é uma informação a mais, que no caso do exemplo, é a contagem de letras "a" no e-mail.

Quando a pessoa receber o e-mail, para saber se houve algum tipo de erro, ela fará o mesmo procedimento, ou seja, pegará o texto que recebeu e contar todas as letras "a" que tem nele. Após a contagem, ela verificará o valor que ela obteve e o valor que recebeu da minha contagem, e em seguida irá comparar esses valores. Se os valores forem iguais, não houve erro, caso contrário, algum erro aconteceu e tentaremos descobrir onde foi.

Na memória é a mesma coisa. Teremos a informação inicial (dado inicial / original) que precisa ser armazenado. Com essa informação inicial é calculada as redundâncias. Na hora que armazena, armazena a informação original mais as redundâncias. Na hora que esse dado for buscado / recuperado para ser utilizado em algum processamento, pega-se o dado original e recalcula a redundância, e em seguida compara a redundância que acabou de ser calculada com a redundância que foi armazenada.

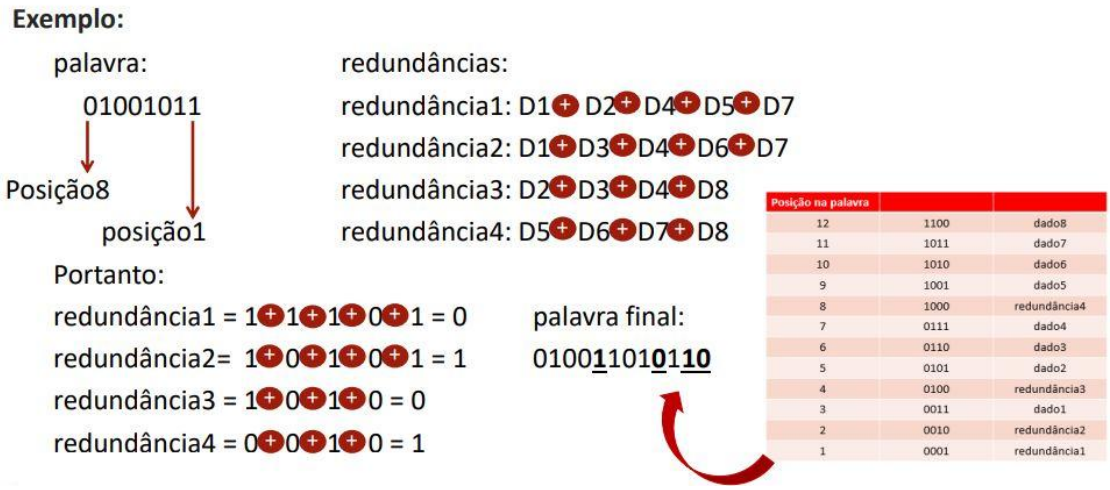
Como calcular as Redundâncias

Nas 12 posições, temos que perguntar onde iremos armazenar a redundância e onde iremos armazenar o dado original.

Observemos a segunda coluna da tabela:

Tudo que tiver apenas um bit '1', é redundância, o contrário, é dado.

> Ex: 0001, 0100, 1000...



Agora que sabemos onde devemos armazenar cada informação na hora de guardar as mesmas, devemos saber como calcular cada uma das redundâncias.

Utilizamos a tabela para saber onde devemos colocar o "dado" e a "redundância", mas também utilizaremos para saber exatamente como calcular as redundâncias, já que a redundância é calculada de acordo com a informação que será armazenada.

Calcular a "Redundância1 (R1)":

Se olharmos na tabela, a R1 tem o '1' na primeira posição (0001). Por isso, pegaremos todos os dados que tem '1' nessa mesma posição.

Descobrimos que tem '1' em "Dado1 (D1)", D2, D4, D5 e D7.

Repetimos esse mesmo processo para todas as redundâncias. Em todas as redundâncias devemos analisar em qual posição (dos bits) está o '1' e depois procurar todos os dados que têm '1' nessa mesma posição (dos bits).

Após analisarmos todas as redundâncias e descobrirmos seus dados respectivos, faremos um cálculo utilizando o "ou exclusivo" (XOR).

Para descobrirmos o valor de cada dado selecionado (D1, D2, etc.) devemos analisar a palavra original (dado original) e corresponder o dado à sua respectiva posição na palavra original.

> Ex: D1 é o bit que está na posição '1' do dado original. No exemplo acima o dado original é "01001011". Portanto, "D1=1".

Cálculo:

R1 = D1 ⊕ D2 ⊕ D4 ⊕ D5 ⊕ D7 ⇨ 1 ⊕ 1 ⊕ 1 ⊕ 0 ⊕ 1 = **0**

R2 = D1 ⊕ D3 ⊕ D4 ⊕ D6 ⊕ D7 ⇨ 1 ⊕ 0 ⊕ 1 ⊕ 0 ⊕ 1 = **1**

R3 = D2 ⊕ D3 ⊕ D4 ⊕ D8 ⇨ 1 ⊕ 0 ⊕ 1 ⊕ 0 = **0**

R4 = D5 ⊕ D6 ⊕ D7 ⊕ D8 ⇨ 0 ⊕ 0 ⊕ 1 ⊕ 0 = **1**

Após ter as redundâncias calculadas e o dado original, devemos montar o "código final" (palavra final). Devemos então procurar em qual posição guardaremos código final.

Se pegarmos a terceira coluna da tabela e substituirmos seus valores respectivos, temos que:

D8 = 0

D7 = 1

D6 = 0

D5 = 0

R4 = 1 (A redundância que descobrimos, aqui apenas colocamos seu valor)

D4 = 1

D3 = 0

D2 = 1

R3 = 0 (A redundância que descobrimos, aqui apenas colocamos seu valor)

D1 = 1

R2 = 1 (A redundância que descobrimos, aqui apenas colocamos seu valor)

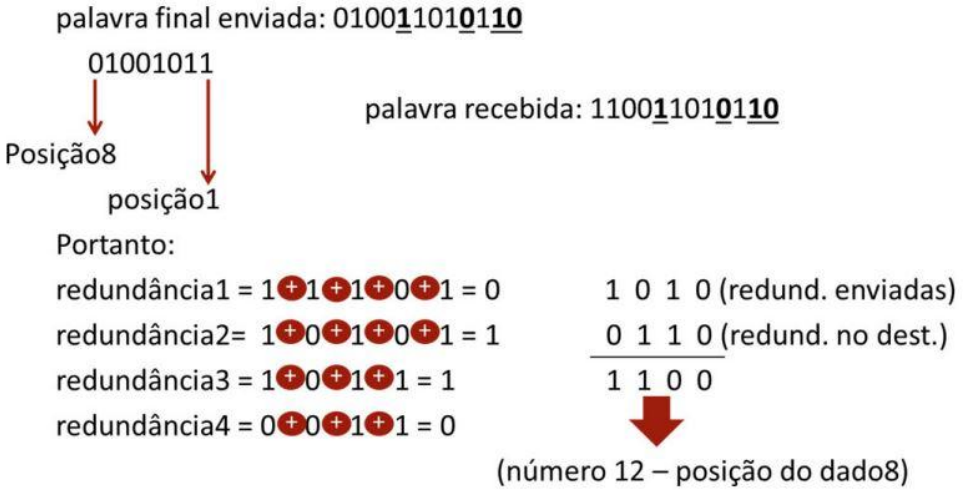
R1 = 0 (A redundância que descobrimos, aqui apenas colocamos seu valor)

Agora é só organizar esses valores de forma horizontal:

Resultado: 010011010110

Perceba que adicionamos as redundâncias junto com os dados, e assim chegamos no resultado final.

Exemplo:



Na hora de buscar a informação final (código final), que informação foi resgatada?

No exemplo, a informação recuperada (palavra recebida) foi "110011010110".

Para descobrirmos se está tudo certo, devemos refazer o cálculo de redundância (assim como no exemplo do e-mail) e comparar com a redundância anterior.

Para isso, devemos pegar a informação que recebemos e separar o que é "dado" e o que é "redundância".

R1 = 0

R2 = 1

D1 = 1

R3 = 0

D2 = 1

D3 = 0

D4 = 1

R4 = 1

D5 = 0

D6 = 0

D7 = 1

D8 = 1

Quem sabe a ordem, ou seja, o que é redundância e o que é dado, é o controlador de memória.

Para recuperar essa informação da memória, pega-se apenas os dados e recalcula as redundâncias.

Cálculo:

R1 ⇨ 1 ⊕ 1 ⊕ 1 ⊕ 0 ⊕ 1 = **0**

R2 ⇨ 1 ⊕ 0 ⊕ 1 ⊕ 0 ⊕ 1 = **1**

R3 ⇨ 1 ⊕ 0 ⊕ 1 ⊕ 1 = **1**

R4 ⇨ 0 ⊕ 0 ⊕ 1 ⊕ 1 = **0**

Chegamos aos valores novos de redundância.

Agora para conferirmos se elas batem, fazemos o seguinte cálculo:

```
1 0 1 0 (redund. enviadas)
0 1 1 0 (redund. no dest.)
---------
1 1 0 0
```

Esse cálculo é novamente um cálculo de "ou exclusivo" (XOR).

Descobrimos o resultado final, sendo este "1100".

"1100" corresponde ao número '12', e no número '12' tem o "Dado8".

Isso nos leva a concluir que existe um erro, e que o dado que foi trocado foi o "Dado8".

Se compararmos o dado armazenado com o dado resgatado, perceberemos que, de fato, a mudança está no "Dado8".

010011010110 (dado armazenado) ≠ **1**10011010110 (dado resgatado)

Com isso tudo, aprendemos que o "Código Hamming" verifica se houve um erro ou não, e qual é a posição do bit errado.

Caso descubra o erro, o controlador de memória irá trocar o bit errado pelo bit correto. No caso do exemplo, ele trocará o bit (na posição 8) por '0', já que ele está como '1'.