讲座 3

第1部分今天：基本纠错码理论

第二部分今天：实际的汉明ECC示例

基本纠错码理论

（纠错码）：用于检测和（可能）纠正单词中的错误

有时我们可以检测并纠正错误；有时我们可以检测但无法纠正；有时两者都不能。

示例：你的朋友给你发短信说“ wouse ”。这是一个错误；wouse不在字典中。“ wouse ”本应是哪个词：souse、mouse、house、worse……我们不能一定纠正所有我们能检测到的错误。错误检测通常相当简单。错误纠正则更难。一些拼写错误无法被检测到：你的朋友发短信告诉你他们想去俱乐部（他们本想说“ 酒吧 ”）。由于club也有意义，这个错误可能无法被检测到。更糟糕的情况是：一些拼写错误比其他错误更糟。你给女朋友发短信“ 你愿意嫁给我吗？ ”而她本想打“ 是 ”，但却错误地打成了“ 不是 ”。

定义：　　请提供要翻译的源文本。：我们想要保护和存储的原始数据字。
请提供要翻译的源文本。

请提供要翻译的源文本。数据字，附加了额外的位。额外的位称为奇偶校验位或
请提供要翻译的源文本。冗余位，这些额外的位使我们能够检测/纠正错误。

汉明码是一种官方的错误检测和纠正协议。

首先，让我们创建我们自己的示例代码。

请提供要翻译的源文本。

代码 A:代码是一组我们一致认为有效的词（就像字典）。例如：如果你和我想创建一个只有两个词的代码，我们可以选择词0和1。这是一个非常高效的代码，但实际上这是一个糟糕的代码：如果发生某些错误，词0被损坏并更改为1，我们无法检测到，因为1也是一个有效的词。解决这个问题的一种方法是创建更长的词。我将使用代码A，新的词汇由两个我们都认为有效的词组成：

00000000

11111111

这些是代码 A 中唯一的两个有效词。

这8位字中的其他254种位组合是无效的。

假设您从RAM中检索到单词00000001。

这个词显然是错误的。错误检测非常简单：这不是00000000，也不是11111111。

错误更正更困难：00000001 应该是什么词？

请提供要翻译的源文本。

可能是 00000000 这是一个猜测，这是一个很好的猜测，通常可以正确地假设 00000001 应该是 00000000

00000000 和 11111111 是比 0 和 1 更好的代码，用于错误检测和纠正。因为后者的代码无法检测错误（因此无法纠正）。

** 我们已经看到了这种权衡：清晰度与效率。为了获得更清晰的词语，我们需要更长的词语，并且它们之间有很多差异。

假设你从RAM中检索到单词00001111。这是个麻烦。

修正拼写错误：

hib ?????????????

住院治疗 住院治疗

请提供要翻译的源文本。

代码 B: 请提供要翻译的源文本。

请提供要翻译的源文本。 B，具有 8 位字和 254 个单词的词汇表：

请提供要翻译的源文本。 00000000 和 11111111 是唯一两个无效的词。其他 254 个八位字的词是
请提供要翻译的源文本。

假设你从RAM中检索到单词01110100。它是一个有效的单词。不过，这是否存在错误？这个单词是有效的，但可能不正确（即它可能不是实际存储/意图的单词）。

请提供要翻译的源文本。

代码A和B的总结：代码A易于检测错误，并且可以纠正许多错误，但在空间上效率低下（一个比特的字也足以用于一个双字代码）。代码B几乎无法检测大多数错误（因此它们将无法被纠正）。它在空间上非常高效（256个可能的8位字中有254个是可用的）。代码B没有提供清晰性。代码B糟糕透了。

请提供要翻译的源文本。

问题：哪种代码（A或B）更适合检测和纠正错误？为什么？

答案：A，因为它可以纠正很多错误：无法纠正00001111；无法纠正像存储00000000然后被损坏为11111110这样的大错误。它可以纠正所有小错误，例如存储00000000然后被损坏为00100000或被损坏为01001000。

请提供要翻译的源文本。

请提供要翻译的源文本。

问题：

00000000

11111111

这些是代码 A 中唯一的两个有效词。

什么是比 A 更好的用于检测和纠正错误的代码？举个例子。解释一下。

0000000000000000　　请提供要翻译的源文本。（16位0）

1111111111111111

我们不希望这些词彼此混淆，因此我们使代码更长，词语之间的差异更大。

为什么这比A更好？ 0000000000001111 现在（可能）是可纠正的；但代码A无法修复4位错误

请提供要翻译的源文本。

9位字比8位字更好；奇数位数使得50/50字无法存在，例如00001111在奇数位数下是不可能的。

更高的位差异数量（9而不是8）使我们能够检测和纠正更多。因此，具有9位字的000000000和111111111的代码将比代码A更好。

例如，考虑代码 A：检索 00000111　　请提供要翻译的源文本。

这要么是3位错误（应该是00000000），要么是5位错误（11111111）

但我们将假设它应该是 00000000。我们无法用 A 修正 4 位错误。我们可以用 9 位码修正 4 位错误：000001111

请提供要翻译的源文本。

问题：代码 A 能纠正 1 位错误吗？为什么/为什么不？解释一下。可以。对于 1 位错误，单词看起来 7/8 像一个单词，只有 1/8th 像另一个单词。因此，我们会假设它应该是 7/8 的单词：例如，存储 000 00000 并检索 00000001：我们可以很容易地将其修正为 00000000，这样我们就是正确的。

问题：代码A能纠正2位错误吗？为什么/为什么不？解释一下。可以。类似的推理。

问题：代码 A 能纠正 3 位错误吗？为什么/为什么不？解释一下。可以。类似的推理。

问题：代码A能纠正4位错误吗？为什么/为什么不？解释。不能。这个词看起来正好一半像一个词，另一半像另一个词。没有办法纠正这个。00001111

问题：代码 A 能纠正 5 位错误吗？为什么/为什么不？不可以！这个有 5 个错误的单词看起来更像错误的单词，而不是正确的单词。我们将假设它应该是那个错误的单词，做出错误的假设，并且没有纠正它。我们让事情变得更糟。如果这个单词应该是 00000000，而你得到了 00011111，你会认为它应该是 11111111（但如果实际上发生了 5 个错误，我们就是错的）。

问题：代码 A 能纠正 6 位错误吗？为什么/为什么不？解释。与 5 位错误的推理没有相似之处，但我们现在更加自信（但我们是错的）。 00000000 -> 00111111

问题：代码 A 能纠正 7 位错误吗？为什么/为什么不？解释。不能。例子：存储 11111111；在错误后检索 00000001。我们会自信但错误地将其更改为 00000000。

问题：代码 A 能纠正 8 位错误吗？为什么/为什么不？解释。不能。这个词现在是有效的（但错误的、非预期的词）：例如，存储 11111111 但在错误后检索：00000000。这不仅是不可纠正的，而且我们甚至不会尝试去纠正它……这是不可检测的。有效但不正确/非预期的词。

代码 A 的最坏情况：

- 4位错误 - 8位错误 - 5/6/7位错误

请提供要翻译的源文本。

00000000

11111111

这些是代码 A 中唯一的两个有效词。

请提供要翻译的源文本。

代码 A 与其最接近的单词之间有 8 位的差异。这被称为汉明距离 (HD)。一个具有 8 位 HD 的代码可以纠正最多三位错误。为什么不能纠正 4 位错误？因为它正好处于中间，看起来像两个不同单词的中间状态：例如 00001111。为什么一个具有 8 位汉明距离的代码不能纠正一个 5 位错误的单词：00000000 → 00011111？因为错误的单词看起来更接近一个错误的单词，并且会被纠正为那个错误的单词。为什么一个具有 8 位汉明距离的代码不能纠正一个 6 位错误的单词？现在它看起来更像那个错误的单词。为什么一个具有 8 位汉明距离的代码不能纠正一个 7 位错误的单词？现在它看起来更像那个错误的单词。为什么一个具有 8 位汉明距离的代码不能纠正一个 8 位错误的单词？它就是错误的单词（有效，但不是预期的单词），因此甚至没有被检测为错误：00000000 → 11111111。

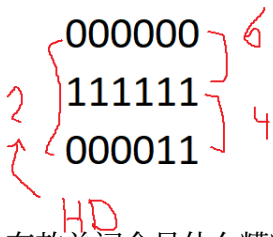
定义：汉明距离：在编码中，任何两个有效字之间不同的最小位数。 请提供要翻

一个具有 h 位汉明距离的代码可以检测最多少于 h 位的错误（任何不是 h 位的错误）。

一个具有 h 位汉明距离的代码可以纠正多达少于 h/2 位的错误（不等于 h/2）。

问题：以下（新）代码的汉明距离是多少：有效单词是：

000000
111111
000011



答案：2 位 问题：代码 A 的第三个有效单词会是什么糟糕的？ 答案：00000000 11111111 00000001 糟糕：代码 A 的 HD 是 8 位，现在是 1 位 ** 我们希望有效代码字之间有较大的间隔 *

* 我们希望有一个大的汉明距离

请提供要翻译的源文本。

请提供要翻译的源文本。

问题：代码 A 的第三个有效单词会是什么？ 答案： 00000000　 11111111　 00001111 我们希望单词尽可能不同；HD 现在是 4 位 (01010101 同样有效)

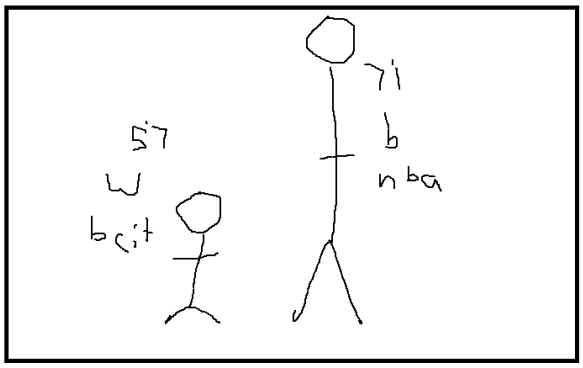
两个词越不同，你混淆它们的可能性就越小。

例如　 BCIT　 你的　 // 非常不同；好；不会混淆这些　 鸭子　 ****　 // 经常与另一个词混淆

请提供要翻译的源文本。

我们希望有效单词之间有较大的间隔

police lineup: we want to detect and correct mistakes



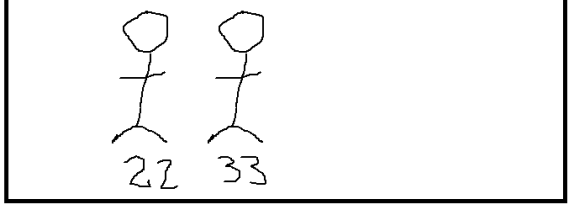
the Hamming distance between Jason and Shaq is large

(i.e. lots of differences for a police lineup)

witness: 7'3, east indian

bad witness: 6'4 chinese

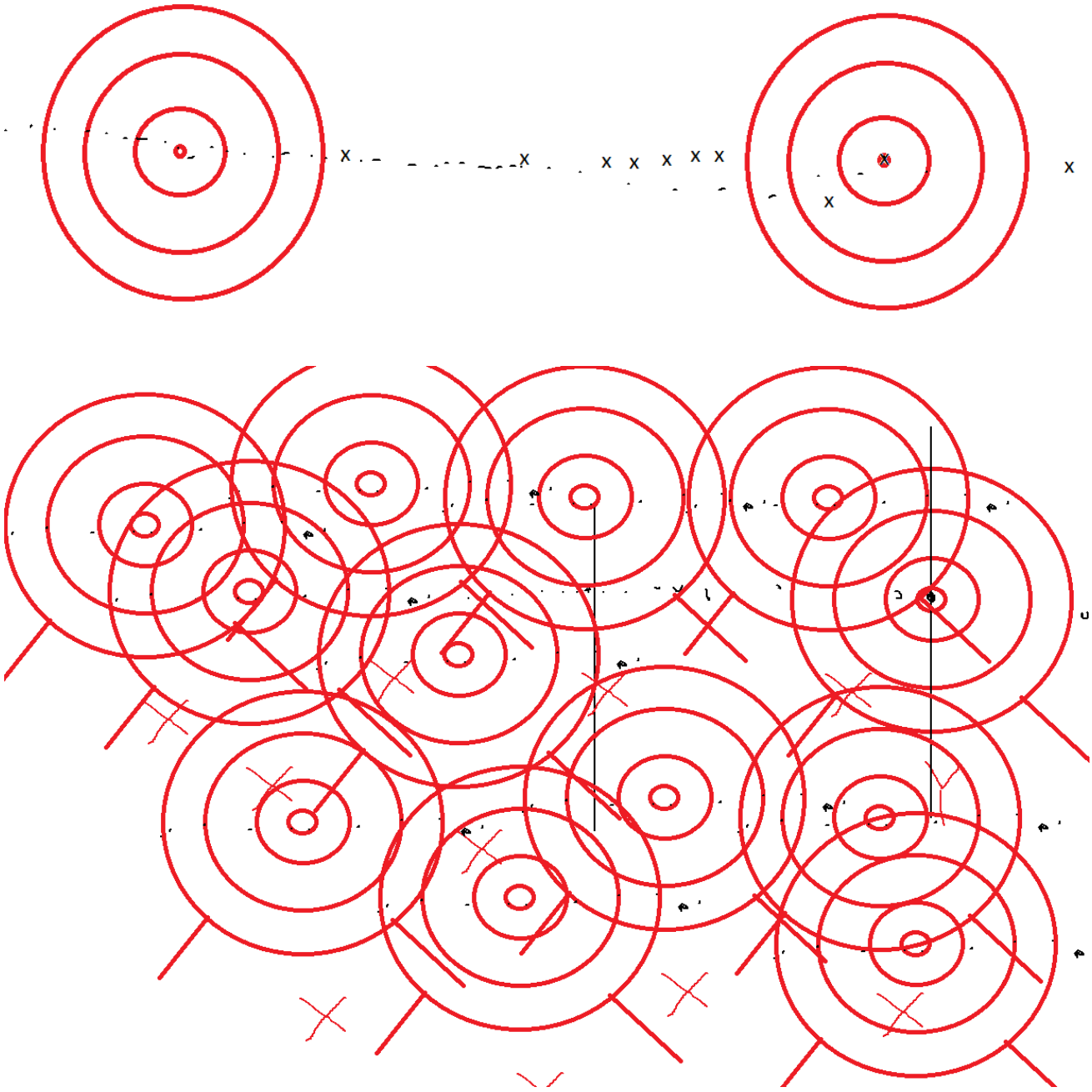
bad police lineup (small Hamming distance):



请提供要翻

请提供要翻译的源文本。

We want large gaps between the targets when we are trying to guess which target the person was shooting at



请提供要翻译的源文本。

请提供要翻译的源文本。

HD	ED (# bit errors detectable)	EC (# bit errors correctable)
8	Up to 7 bits (this is code A)	Up to 3
256	1 to 255	Less than 128
1	Code B: 0	0
H	Less than H	Less than H/2
21	20 (better than code A)	Less than 10.5 (i.e. 10 or fewer)
8000	Anything but 8000	Less than 4000
17	Up to 16 bits	8 or fewer
10	Less than 10	Less than 5

审查：想象一下你我正在交换信息。我们同意使用某些词汇（例如，英语在字典中对词汇的约定）。让我们在我们的代码中使用两个词：

000000

111111

您只能发送六个零或六个一。　　请提供要翻译的源文本。

如果我发送给你000000并且出现错误，而你收到100000，你会怎么做？这绝对不是一个正确的词，因为它不是000000也不是111111。它可能应该是000000。这个课程是关于错误检测和错误纠正的。你的计算机猜测（即使用概率）它不是111111。

# of errors	The word I sent	What you received	Detect error(s)?	Correct error(s)?
6	000000	111111	No, a wrong (but valid!) word was received	No, wouldn't even try to correct since it looks good
5	000000	111110	Yes: it's not 000000 and not 111111	NO. You think you could, but you'd be wrong; it looks like 111111 but in this case, with 5 errors, we would be wrong
4	000000	111100	Yes	No, same as above
3	000000	111000	Yes	No, it looks half like each of two words. We have no clue what to do
2	000000	110000	Yes	Yes, it looks more like 000000 and in this case we would be right
1	000000	100000	Yes	Yes, see the paragraph above

请提供要翻译的源文本。

000000和111111之间的距离是6位。一个编码的汉明距离是最小的 _____请提供要翻译的源文本。任意两个单词之间的位差数量。 ___

汉明距离越大，我们可以检测和纠正的错误就越多。

想象一下如果我们有这两个词：

00000000000000000000000000000000

11111111111111111111111111111111

32个零是一个词；32个一是另一个词

汉明距离是32位！我们可以检测最多31个错误。我们可以纠正最多15位错误。无法修复16个错误：0000000000000000111111111111111111是不可修复的。无法修复17到31个错误：因为我们将“纠正”成错误的单词（它更接近于这个单词）。无法修复32个错误：它现在看起来完全像错误的单词（这是另一个有效的单词，但不是我们所意图的）。

如果我们有这段代码：

00000000000000000000000000000000

11111111111111111111111111111111

然后有了　选择一个第三个词来达成一致，一个好的词会　ld be:

00000000000000001111111111111111

…因为它与其他词尽可能不同，产生了新的汉明距离为16位。
就像这样也很好：01010101010101010101010101010101 等等……

一个可怕的第三个词要添加到我们的代码中将是：

00000000000000000000000000000001

…或任何其他与前两个词过于接近的内容。这会破坏整个代码。HD 现在是 1 位。

请提供要翻译的源文本。

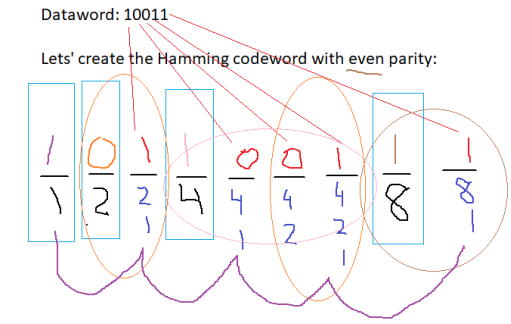
Hamming Distance (bits)	# of errors detectable	# of errors correctable
6	5 or fewer	2 or fewer
32	Up to 31	Up to 15
1	0	0
H	Less than H	Less than H/2
43	Less than 43	Up to 21

请提供要翻译的源文本。

实际汉明码：错误检测、纠正和测量

数据字：我们想要存储或发送的原始数据 码字：数据字加上额外的位。额外的位称为奇偶校验位，它们使我们能够检测/纠正错误 数据字：10011

让我们为这个数据字创建汉明码字。我们可以使用偶数或奇数校验。让我们选择偶数校验。



the codeword bit positions that are powers of 2 are RESERVED:

bits 1, 2, 4, 8, 16, 32, etc...

...reserved for parity bits, calculated by an algorithm

The other bit positions (3, 5, 6, 7, 9, etc...) contain the dataword bits

in even parity, the parity bits (1, 2, 4, 8, etc...) values make their sum EVEN

each parity bit checks all the bits with its number

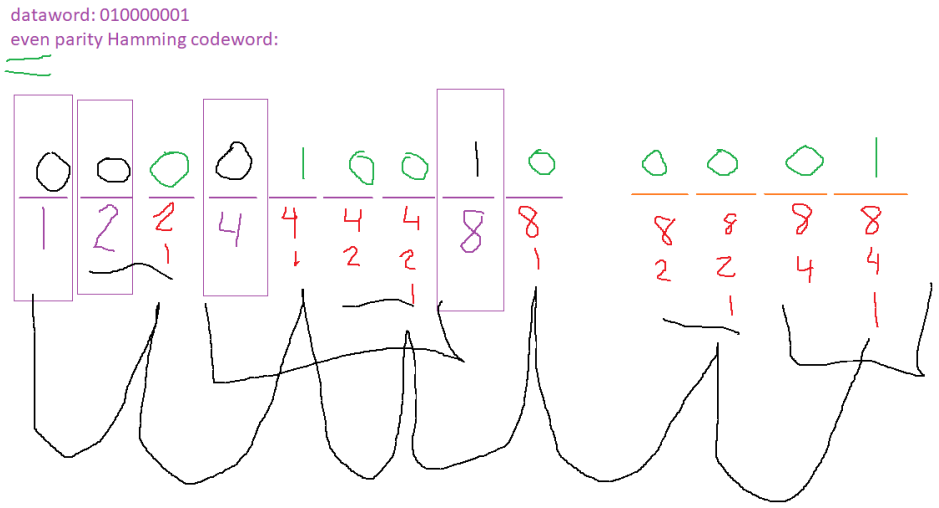
Therefore in this case we must store NINE bits in order to store a FIVE-bit word (with the ability to detect and correct errors).

in order to store the dataword 10011 with even parity we must store the Hamming codeword 101100111

请提供要翻

问题：为数据字 010000001 创建具有偶数奇偶校验的汉明码字

答案：



to store 010000001 as an even-parity Hamming codeword, we must store 0000100100001

ANSWER

请提供要翻

相同的问题，但现在我们想要一个奇偶校验码字。答案：翻转位 1-2-4-8：1101100000001

请提供要翻译的源文本。

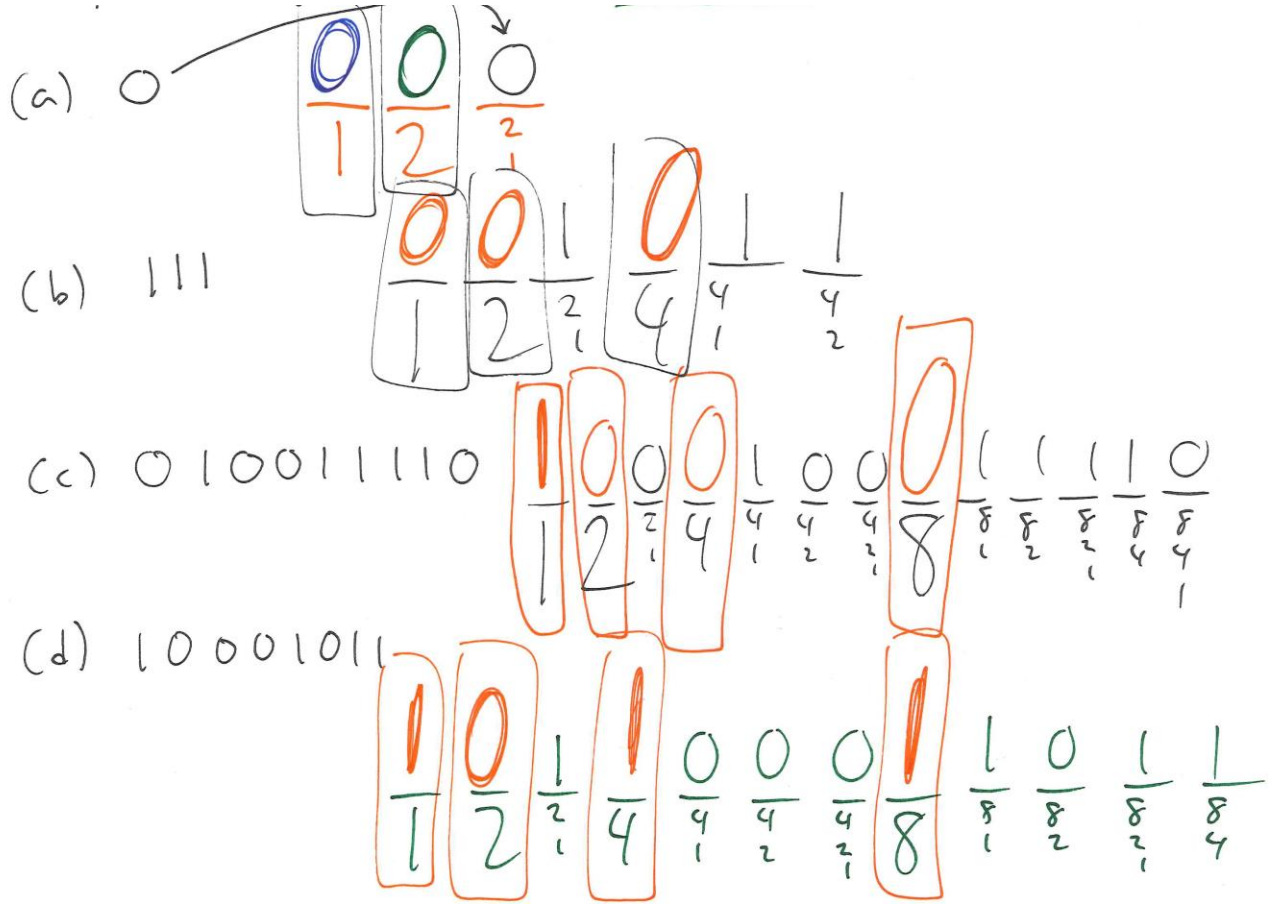
Create Hamming codewords with even parity for the following data words:

a) 0

b) 111

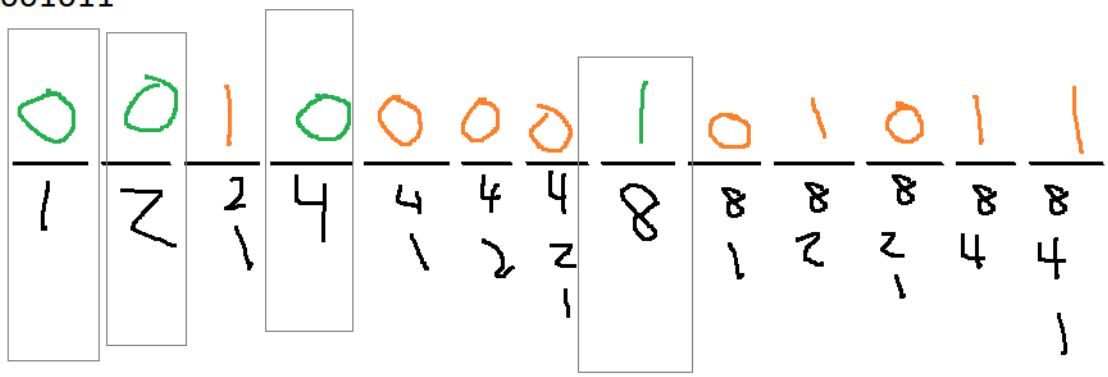
c) 010011110

d) 100001011



请提供要翻

d) 100001011



对于最后一个问题，对于奇数奇偶校验，只需翻转奇偶校验位：1111000001011

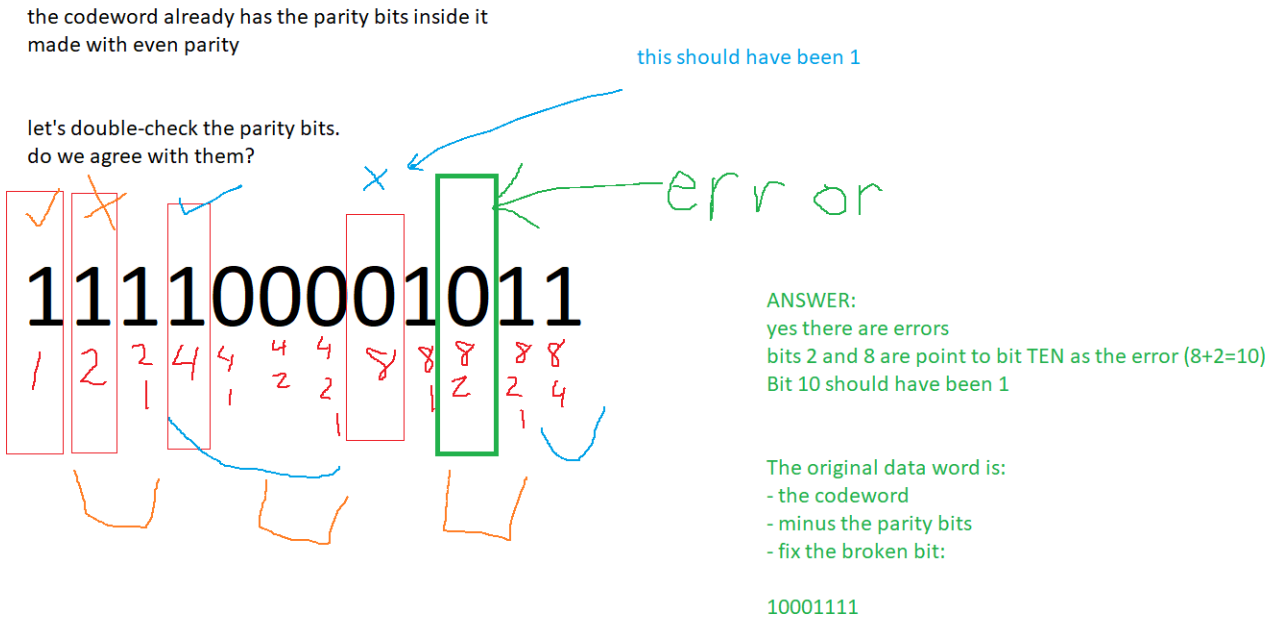
请提供要翻译的源文本。

请提供要翻译的源文本。

以下的汉明码字是使用偶校验创建的。它是否包含错误？在哪里？原始数据字应该是什么？

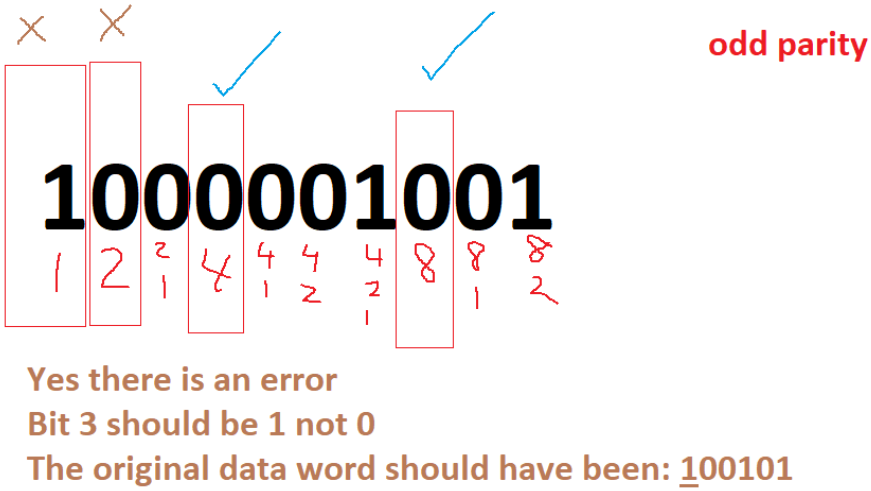
111100001011

请提供要翻译的源文本。



以下的汉明码字是使用奇校验生成的。它是否包含错误？在哪里？原始数据字应该是什么？

1000001001



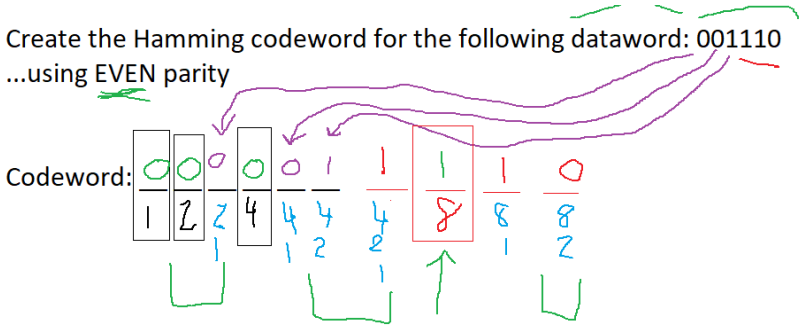
请提供要翻译的源文本。

问题：如果只有第2位是错误的呢？ 答案：那么
第2位就是错误的。就这样。

请提供要翻译的源文本。

五月12日：

Create the Hamming codeword for the following dataword: 001110
...using EVEN parity



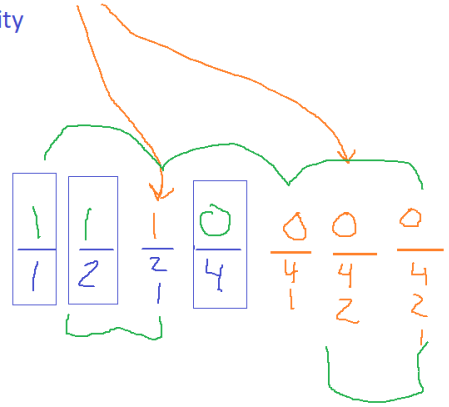
bits that are numbered as powers of 2:
1, 2, 4, 8, 16, ...

are called parity bits
(aka check bits, redundant bits, or Hamming bits)

To store the data word 001110 with even parity, we need to actually store the Hamming codeword 0000011110

请提供要翻译的源文本。

Create the Hamming codeword for the dataword 1000 with EVEN parity



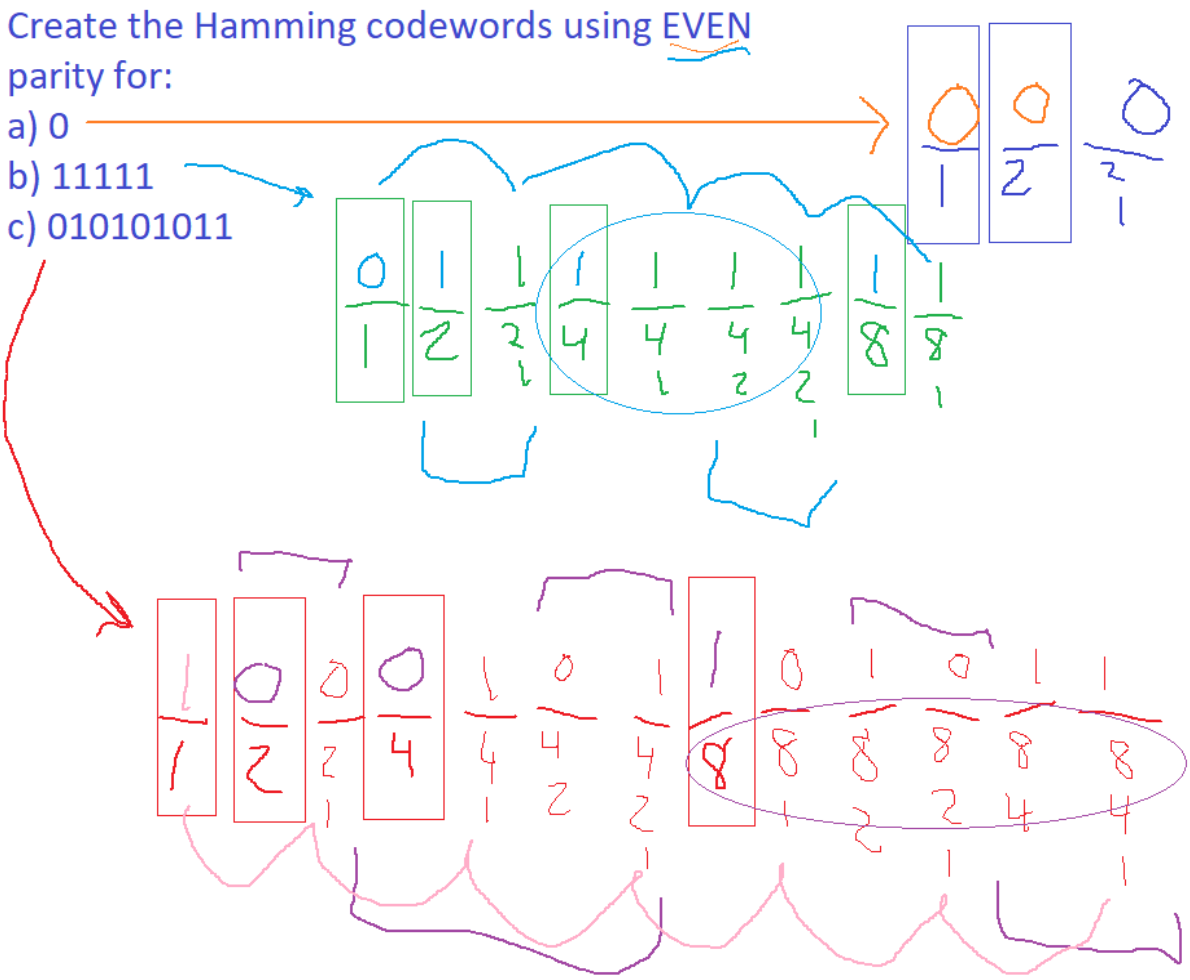
codeword: 1110000

What would the codeword be for ODD parity?

Just flip the parity bits:

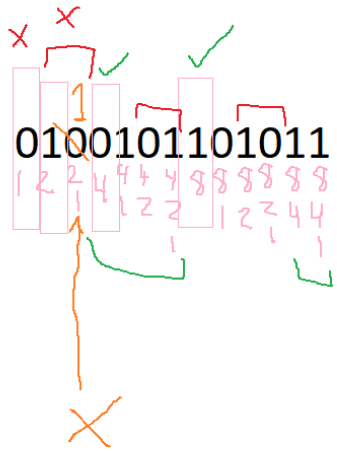
0011000

请提供要翻译的源文本。


Create the Hamming codewords using EVEN parity for:
a) 0
b) 11111
c) 010101011

请提供要翻

The following Hamming codeword was created using EVEN parity. Are there errors? If so, where? What was the original dataword supposed to have been?



double checking:
should bits 1,2,4,8 have the values 0101?

Yes there are errors.
Bits 1 and 2 show that bit 1+2= 3, bit 3 is wrong

Bit 3 should be 1

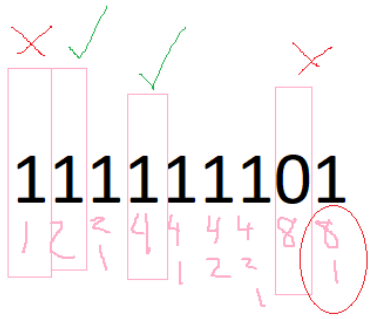
To recover the original dataword:
remove the parity bits
flip bit 3

110101011 was the original intended dataword

请提供要翻

新问题：以下汉明码字是使用偶校验生成的。是否存在错误？在哪里？原始数据字应该是什么？请展示你的计算过程。清楚地标识校验位：

111111101



Yes there are errors
Bit 1+8 = 9; bit 9 should be 0 instead of 1
The original dataword is the codeword, minus the parity bits, and flip bit 9:

11110

请提供要翻

后续问题：如果前一个问题使用奇数校验会怎样？

答案：是的。位 2+4 = 6；位 6 是错误。位 6 应该是 0 而不是 1。翻转位 6，去掉位 1、2、4、8，得到原始预期的数据字：11011

问题：如果唯一“看起来不对”的位是位 2 呢？

答案：那么只有第2位是错误的！

请提供要翻译的源文本。