# 错误检测与纠正

# Outline

- 导论
- 奇偶校验
- RAID原理
- ISBN检测
- 纠错编码

# 导论

- 当数据储存在硬盘或传送到网络上时，它们一般是不会发生改变的。
- 不过有时候一些故障也会导致数据被突然改变，比如电子干扰
- 我们应该避免这类事件的发生
- 那么计算机是否能够自动检测出数据的变化，甚至是自动修正错误呢？

# 导论

例如你向网上卖家发了一封email，确认你将以20元来购买他的一件商品。但是一阵电子干扰过后，20元变成了80元。卖家一定会欣然接受你的订单，有人居然愿意出4倍的价格来购买一件商品。但是你将不幸得蒙受损失！

# 导论

- 任何存储在计算机或传送在计算机之间的数据都是采用比特的形式表达的
- 存储或传输设备上发生的错误，很容易导致数据的突然变化
  - CD上的划痕或表明的小灰尘会把0变成1，或将1变成0
  - 硬盘存放数据的区域可能被意外地磁化
  - 网络的干扰和连续不畅也会导致比特被改变

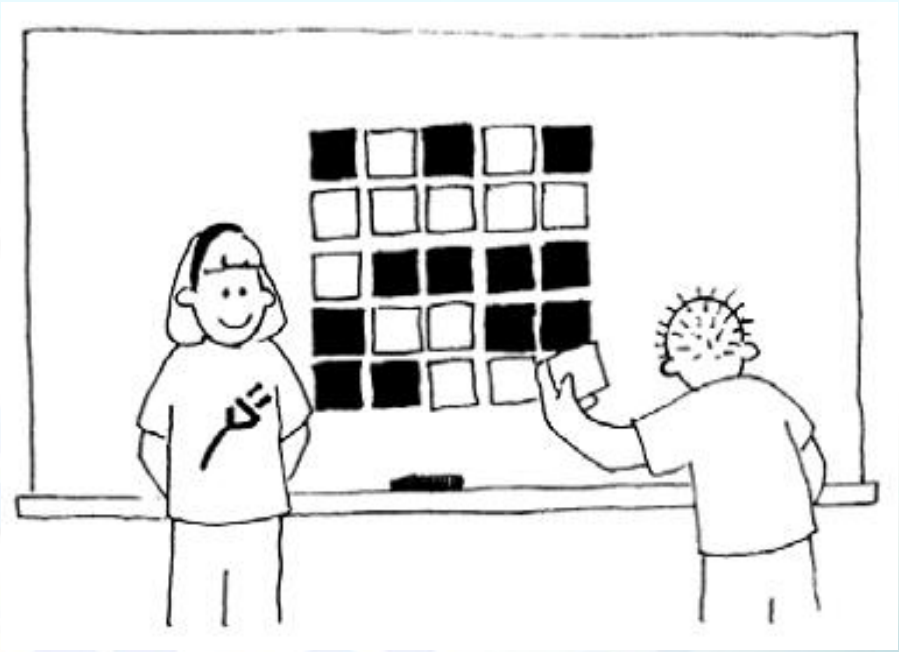
# 导论

那么我们要怎样才不用担心发生这些意外呢？

- 幸好科学家们发明了让计算机自动检测数据中的错误并自动修复的方法
- 接下来介绍其中的一个方法：奇偶校验
- 首先看一个小魔术

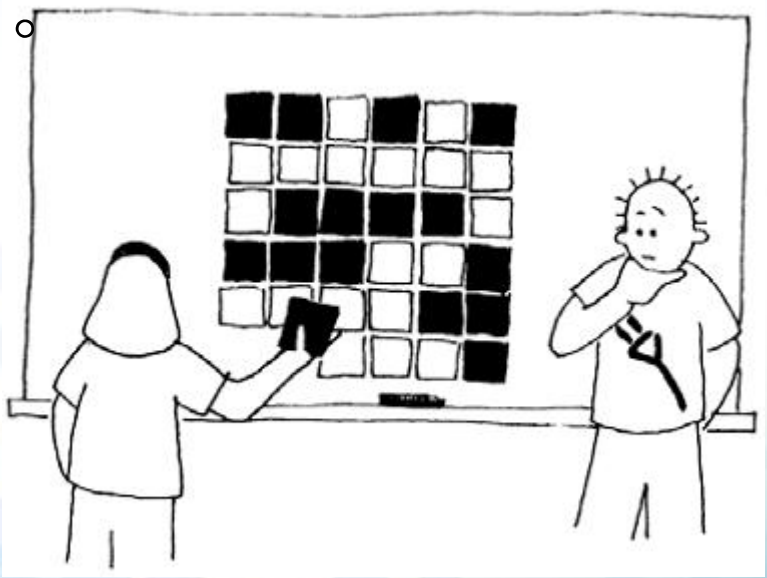
# 翻卡魔术

- 这个魔术需要36张卡片，保证每张卡片两面的和颜色不同
- 找一个同学任意摆放一个5x5的卡片方阵



# 翻卡魔术

- 为了增加难度你再在5x5的方阵周围再加入一圈卡片
- 你背过身去，由别人任意翻转一张卡片
- 你转过身来时，总能告诉他哪一张卡片是被翻过面的。



# 翻卡魔术

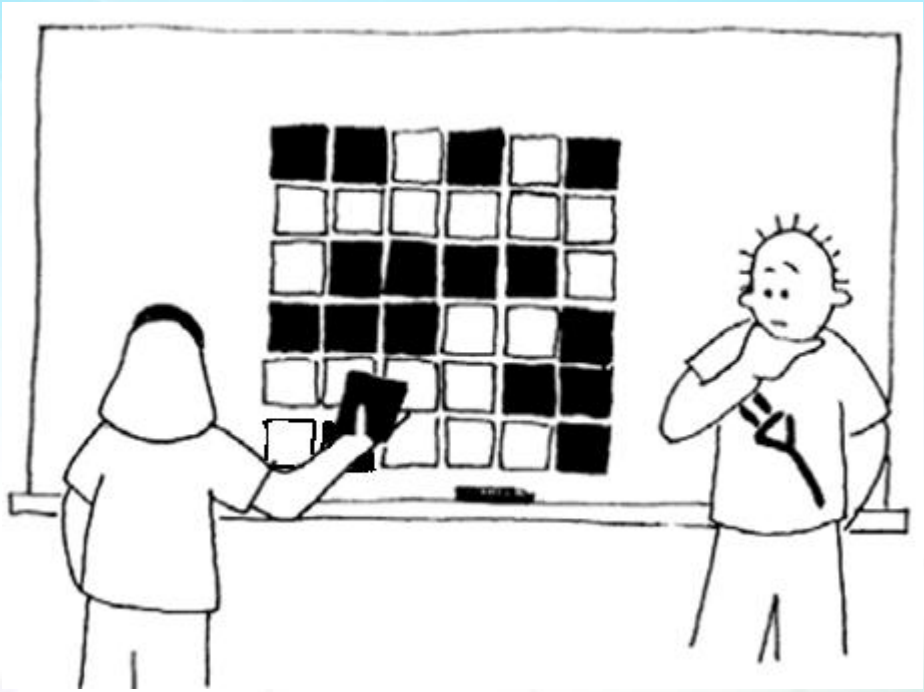
你能揭示这个魔术的秘密吗？难道是因为魔术师的记忆力超群吗？如果更多卡片呢？

# 翻卡魔术揭秘

- 其实诀窍就在于你增加的那几张卡片
- 实际上你是在原有卡片的最右侧增加一列、最底部增加一行来放新增加的卡片
- 当然你口头上说的是这样做是为了增加魔术的难度
- 下面我们将观察一下这些卡片到底有什么用的秘密！

# 翻卡魔术揭秘

- 每行有多少张白色卡片呢？
- 每列有多少张白色卡片呢？
- 这些数字有哪些规律呢？

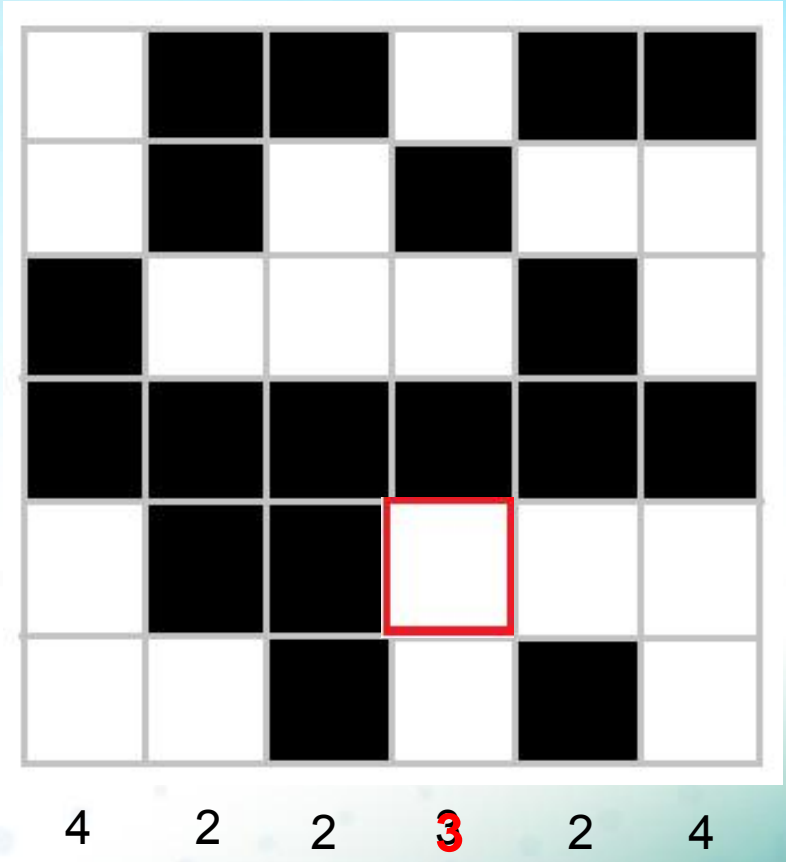


# 翻卡魔术揭秘

- 其实你已经意识到每行及每列白色卡片的数量都是偶数
- 每当男孩在一行中摆放了奇数张的白色卡片，女孩变会在行末增加一张白色卡片来保证改行白色卡片为偶数
- 如果男孩摆放时白色卡片的总数已经为偶数了，那么女孩则添加一张黑色卡片来保持白色卡片总数仍为偶数

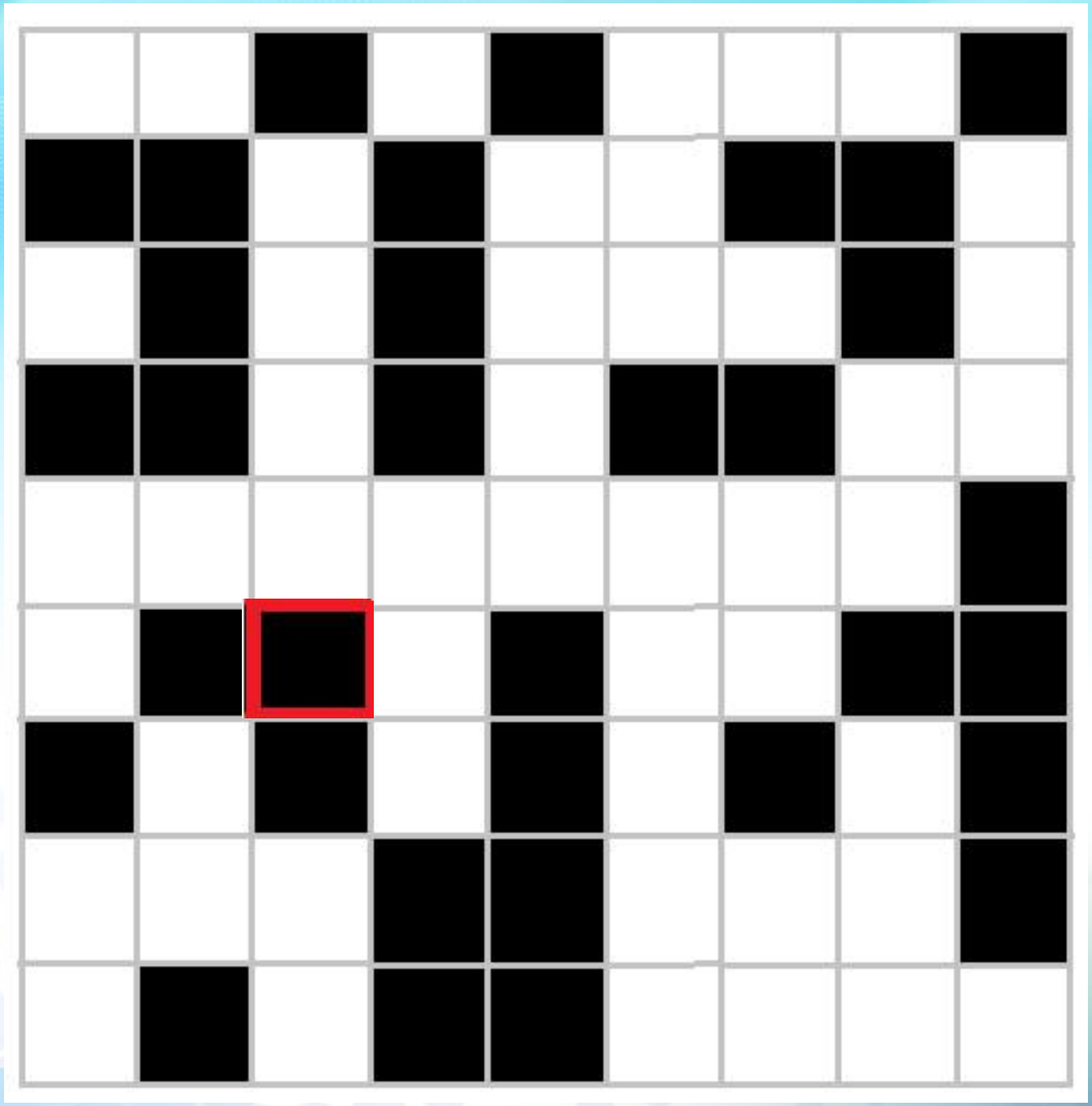
# 翻卡魔术

- 记下每行白色卡片的总数
- 哪一行白色卡片总数不是偶数了呢？
- 记下每列白色卡片的总数
- 那一列白色卡片总数不是偶数了呢？
- 现在你能判断出是那张卡片被翻动过了吗？



# 翻卡魔术

- 其实你已经发现被翻动过的卡片所在的行和列中白色卡片的总数都变成奇数了
- 要找到那张被翻动过的卡片，只需要看看哪一行哪一列拥有奇数的白色卡片
- 它们相交的地方就是你寻找的目标了
- 这个方法适用于任何成矩形排列的卡片组合，试试找出被翻动的卡片？



# 奇偶校验

- 这些被放置的卡片就好比计算机中的比特（0或1）
- 而0和1的组合代表了数字、字母或图像
- 那些新增的卡片我们称之为奇偶校验位
- 计算机就是通过在数据中添加奇偶校验位来保证数据不被随意修改

# 奇偶校验

- 奇偶校验(parity)一词的英文源于词根“pair”
- 偶校验(even parity)表示物件的总数量为偶数
- 奇校验(odd parity)表示它们不能被组成对
- 在上面的例子中我们采用的是偶校验来保证白色卡片的数量总能组成对

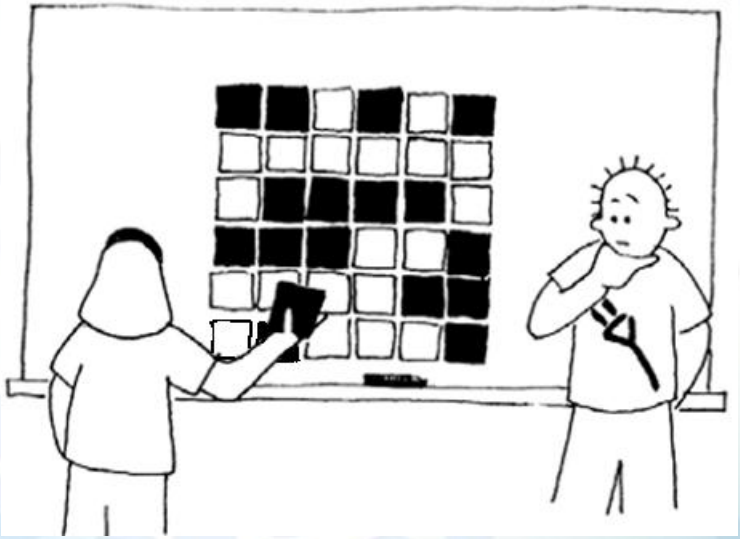
# 奇偶校验

利用类似奇偶校验的方法，可以保护计算机中几乎所有的数据。数据硬盘、CD、DVD、闪存、网络下载、电子邮件和网页都在数据中添加了你看不见的校验码。一旦系统中个别比特发生了错误，计算机就会在你不知情的情况下自动恢复原始数据。

# 奇偶校验

- 请注意图中最右下角的那张卡片，请问它总能保证上面一列和左边一行的正确吗？为什么？

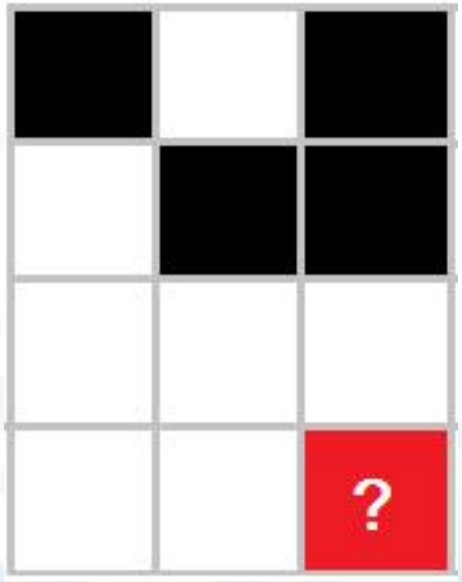
一定满足，因为验证码中的出现白色卡片，说明该行或该列是奇数，那么最后的行或者列中的白色卡片数量的奇偶性就是所有卡片中白色卡片和的奇偶性，是一致的。



# 奇偶校验

- 之前例子中，我们使用的是偶校验，那么使用奇校验是否可行呢？

这也是可行的，但是最右下角的那张卡片只有在行和列都是奇数或者都是偶数的时候才能满足，例如5x9或者4x6，而3x4是不行的



# 奇偶校验

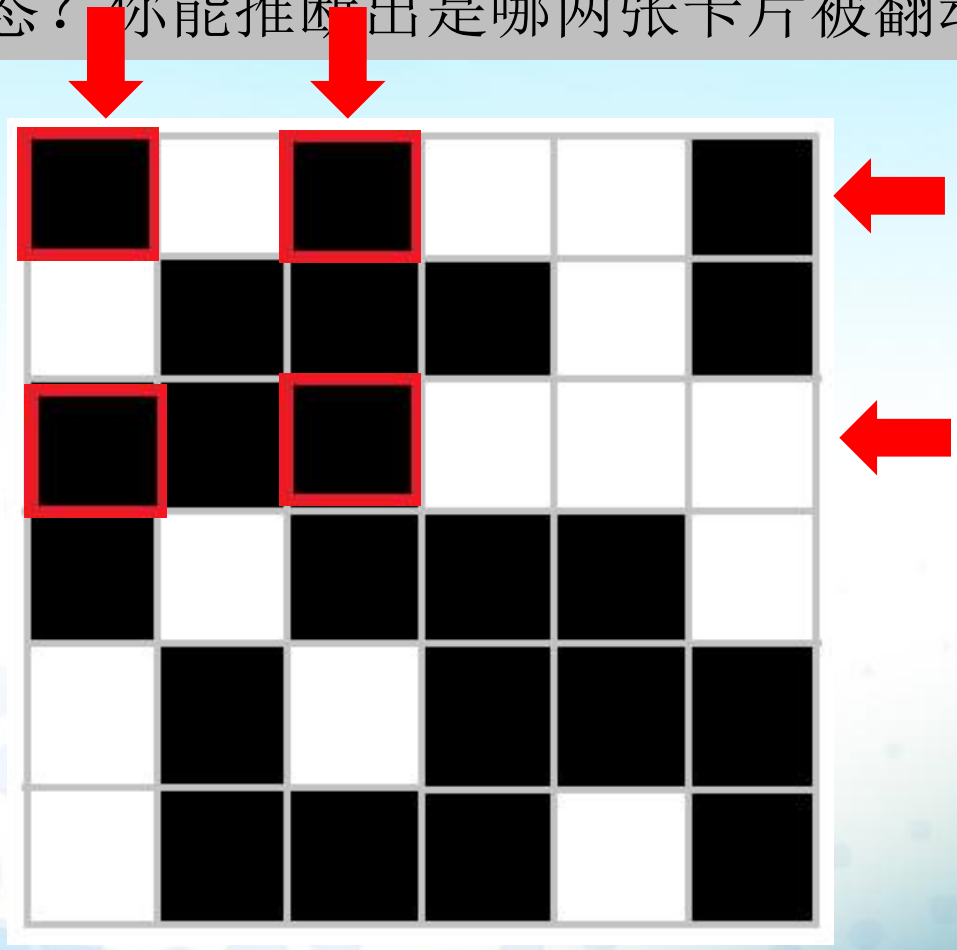
- 有时候仅须检测到错误的发生就足够了
  - 两台计算机正通过网络收发数据，如果接收方察觉数据在传输中被改变了，它只要让发送方再传送一次即可
- 然而有时候数据是无法再一次被发送的
  - 磁盘或闪存保存的数据，一旦因为磁化或过热导致磁盘上的数据被改变，除非计算机能够修正错误的部分，否则这个数据就永远地遗失了

# 奇偶校验

- 奇偶校验可以检测并修正一个错误
- 计算机中也会有不止一个比特发生错误的情况
- 那么在不只一个错误发生的情况下，奇偶校验是如何操作的？
- 当发生一系列错误时，什么情况下计算机能利用奇偶校验位来检测并修正错误？

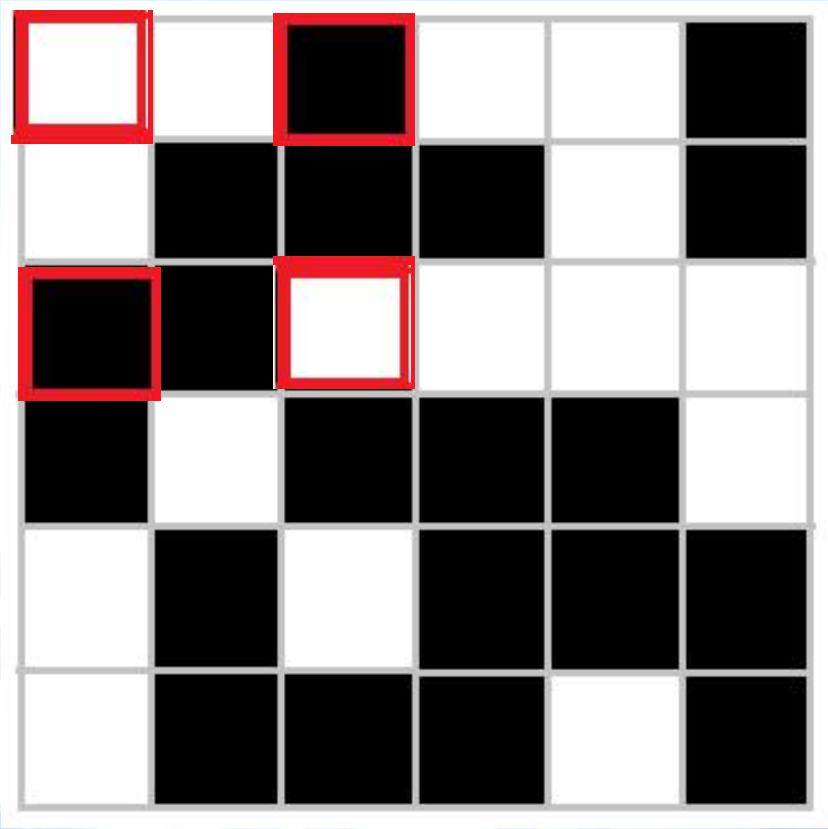
# 检测更多的错误

这里有两张卡片翻了过来，哪几行哪几列现在处于错误状态？你能推断出是哪两张卡片被翻动过了吗？



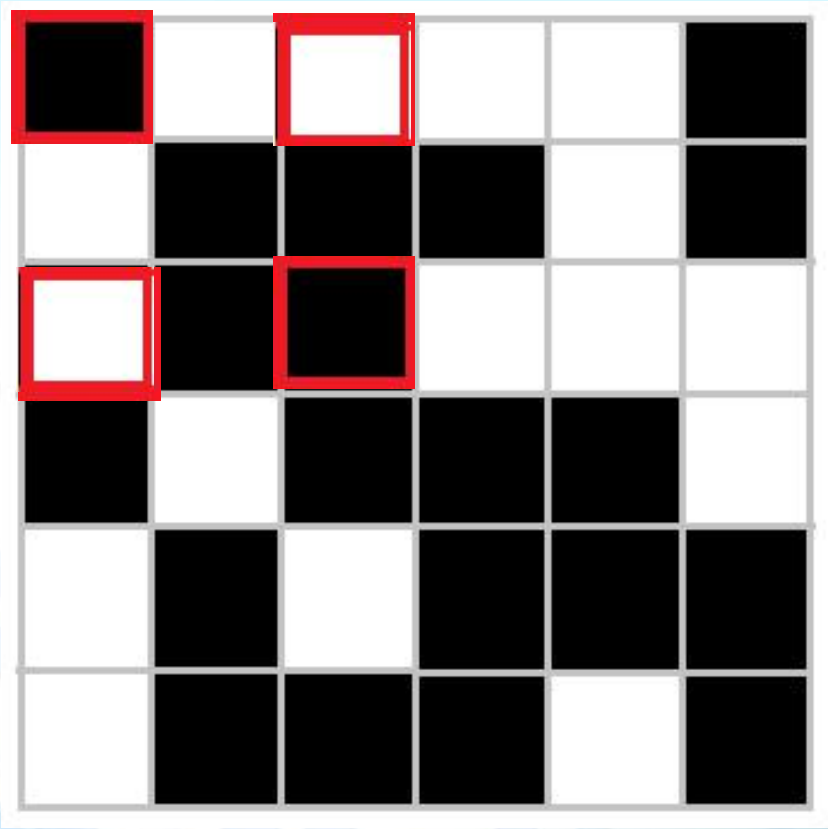
# 检测更多的错误

当两张卡片被翻过来后，在上面的例子中我们检测发生了错误，但是你能把它修正过来吗？



# 检测更多的错误

当两张卡片被翻过来后，在上面的例子中我们检测发生了错误，但是你能把它修正过来吗？

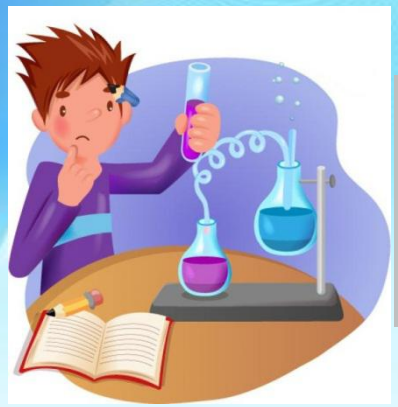


# 检测更多的错误

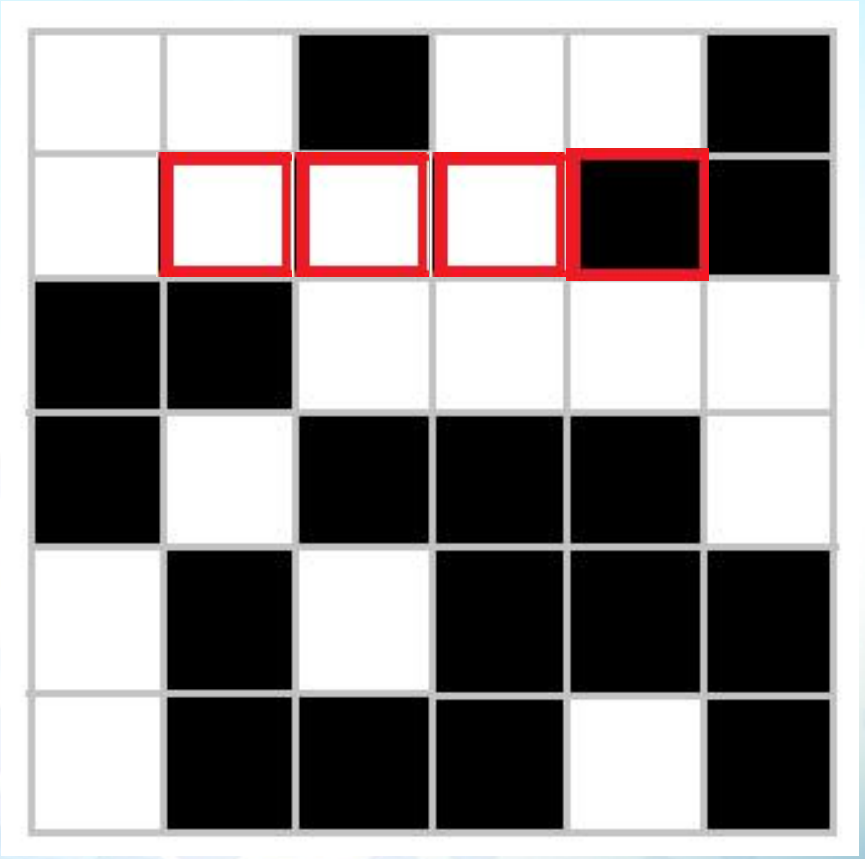
- 上面的例子中翻过来两张卡片后，可以检测到错误，但是却无法修正过来，因为有两种修改的可能

试着用几张卡片排列成满足奇偶校验原理的阵列（保证每行和每列的白色卡片均为偶数）。看看你能在翻动两张卡片后，保持每行每列白色卡片总数仍为偶数吗？

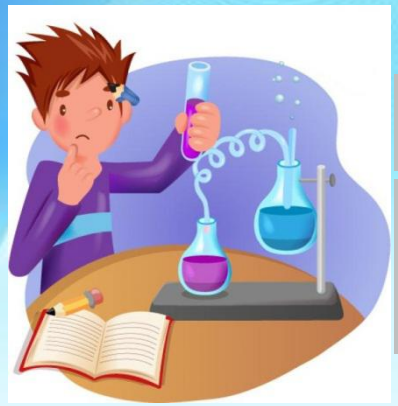
# 检测更多的错误



继续尝试一下，你能做到翻动3张卡片，但不被检测出来吗？最后如果我们在同一行中翻动4张卡片，能被检测出来吗？

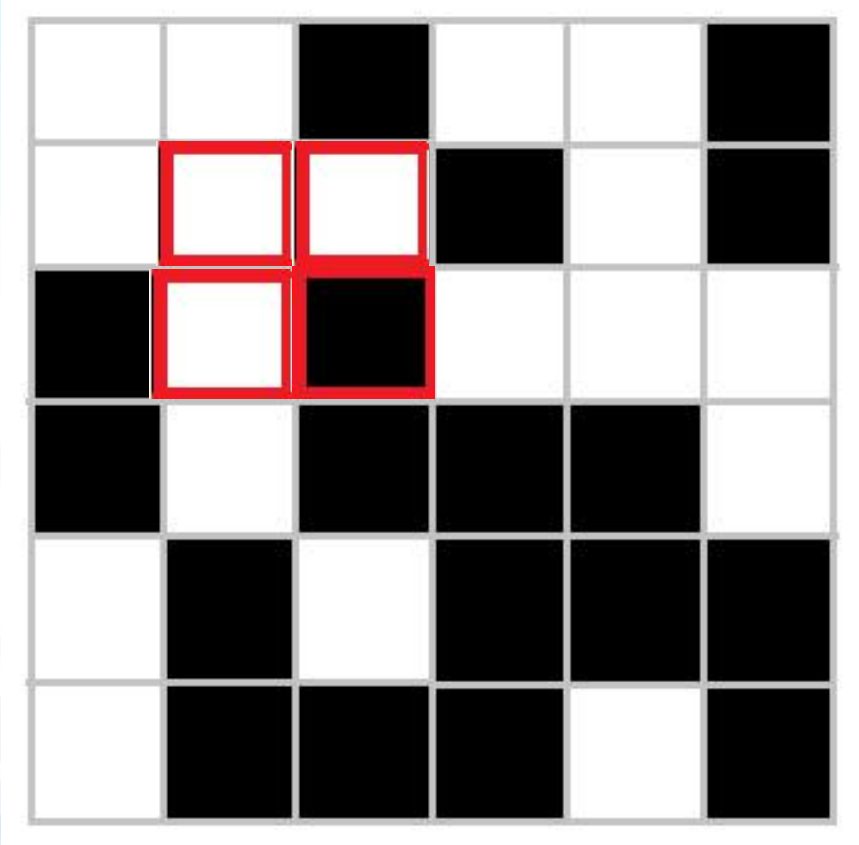


# 检测更多的错误



你有办法翻动4张卡片但并不被检测出来吗？

因此，如果发生了4个错误，计算机或许连一个错误都检测不出



# 奇偶校验

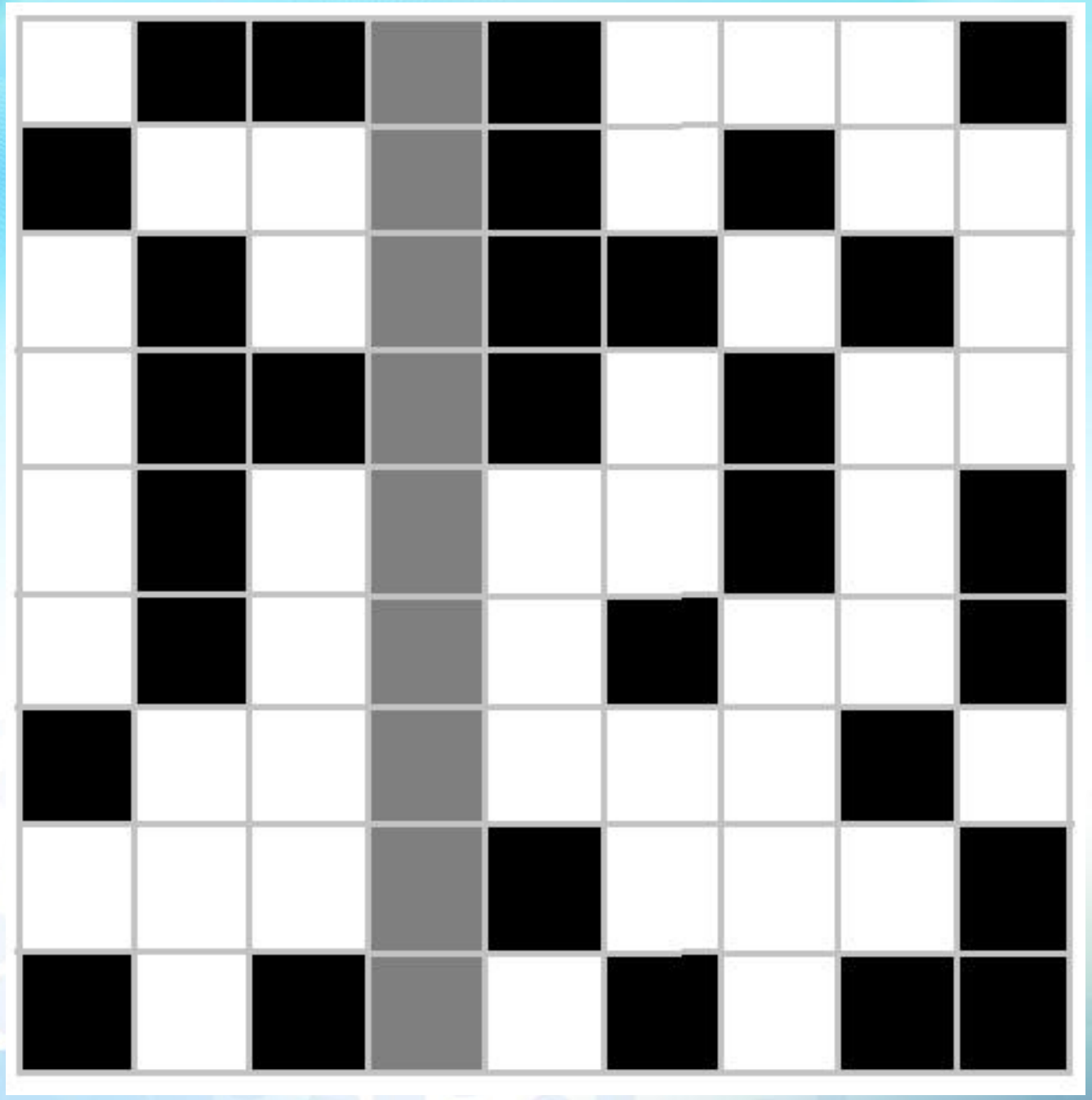
- 总结校验的特点

| 错误的数量 | 总能检测到 | 总能修正 |
| --- | --- | --- |
| 1 | YES | YES |
| 2 or 3 | YES | NO |
| 4 | NO | NO |

# RAID

- 当发生多个错误的时候，有一种特殊情况下错误可以被纠正
- 下一页显示了一个奇偶校验阵列（每行每列的白色卡片数均为偶数），但是它的第四列全部丢失

这种情况你能否将丢失的部分恢复过来吗？



# RAID

- RAID是独立冗余磁盘阵列
  - Redundant Array of Independent Disks
- RAID磁盘系统采用的就是这种纠错方式
  - 通过将数据分散储存在多块而不是一块硬盘中，来保证运行的高速性和稳定性
  - 额外附加的硬盘提高了硬盘速度和纠错性能

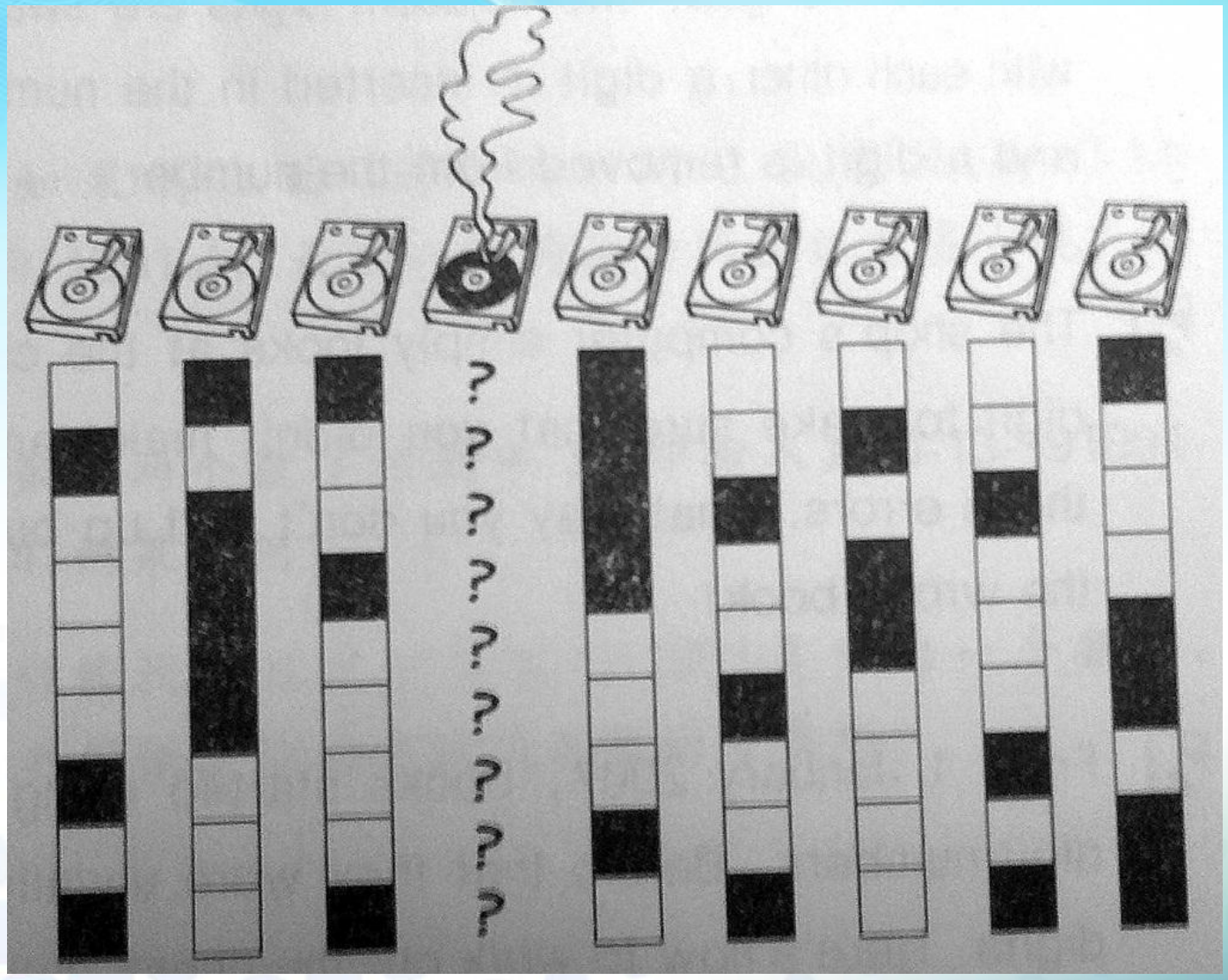
# RAID

- 奇偶校验系统的一个优化方案称为RAID5。
- 假设你需要用8个硬盘来储存大量数据，
  - 这时你可以将每个字节打散成8比特分别储存在多个硬盘上
  - 而不是将数据陆续填满每个硬盘。
- 这样的存储方式会让系统运行得更快，因为当计算机需要读取文件时，它只用分别同时向每块硬盘读取片段即可。

# RAID

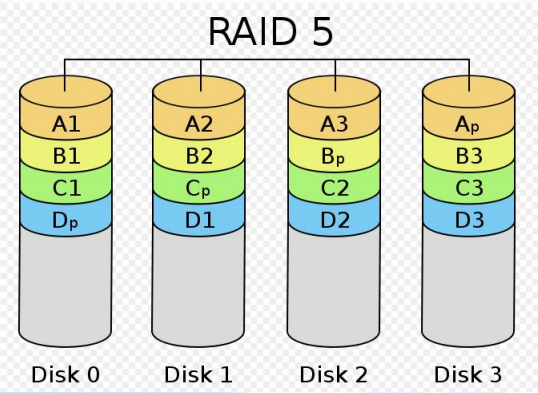
- 该方法也可用于提高纠错性能
- 再增加存有奇偶校验位的第9块硬盘
- 每列数据放在不同的硬盘上，如果其中一块硬盘损坏，即使损失全部数据，仍然能依靠奇偶校验来修复原始数据
- 只用算出遗失的比特使得9个硬盘上值为1的比特总保持偶数即可。

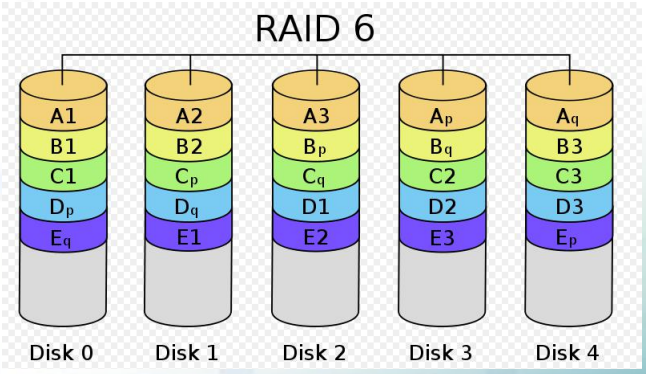
# RAID原理



# RAID的发展

- 前面介绍的只是RAID的基本原理
- 实际上的RAID要更加的复杂
- RAID也分为RAID0, RAID1, RAID2, …, RAID6, RAID7, RAID10/01等
- 但是实际应用多会使用RAID5或RAID6



# RAID磁盘系统

- 磁盘是数据库管理系统性能的瓶颈
  - 微处理器速度的提高为每年50%
  - 磁盘访问的速度的提高为每年10%
  - 数据传输的速度的提高为每年20%
- 磁盘阵列（Disk Array）
  - 通过数据条带（Data Striping）分布将多个磁盘变成一个整体
  - 若干磁盘组织在一起，通过并行提高速度
  - 通过冗余提高数据的可靠性
  - Redundant array of independent disk=RAID

# 数据条带

- 磁盘间的并行性提高了磁盘组数据读取的性能
- 数据条带
  - 数据被分成等长的分区，分布在多个盘上，
  - 每个分区的大小为一个条带单元(striping unit)

# 数据冗余

- 磁盘组可有效提高性能，但降低了可靠性
- 通过增加数据冗余，即check disk来提高数据可靠性
  - 冗余信息的存放位置
  - 如何计算冗余信息
    - 奇偶校验
    - Hamming code
      - 数据存放的位置： D1 D2 D3 D4 D5 D6 D7 D8
        海明码： C1 C2 C3 C4
        C1 = D1 + D2 + D4 + D5 + D7
        C2 = D1 + D3 + D4 + D6 + D7
        C3 = D2 + D3 + D4 + D8
        C4 = D5 + D6 + D7 + D8
    - Reed-Solomon Code

# RAID LEVEL

- RAID
  - 将磁盘分成组，每个组有若干数据盘和一些校验盘构成，校验盘的个数有RAID Level 决定
- LEVEL 0：Nonredundant
  - 没有冗余数据
  - 可靠性差
  - 写性能最优，读性能却不是最优
  - 空间利用率100%

# RAID LEVEL

- LEVEL 1：Mirrored
  - 最昂贵的一种方法，每个磁盘都由一个备份
  - 每次写两遍，同一块数据可并行读
  - 空间利用率５０％
- LEVEL ０＋１：Striping and Mirroring
  - 以Mirrored的方法为基础，增加数据分布的条带化
  - 各种性能与Level1相似

# RAID LEVEL

- LEVEL 2：Error-Correcting Code
  - Striping unit 是一位
  - 冗余模式为Hamming Code
  - Check disk的数量是数据盘的数量的对数
  - 读操作一次D块,D为数据盘的数量
  - 写操作遵循read-modify-write cycle
- LEVEL 3：Bit-Interleaved Parity
  - Striping unit 是一位
  - 冗余模式为奇偶校验
  - Check disk的数量是 1

# RAID LEVEL

- LEVEL 4：Block-Interleaved Parity
  - Striping unit 是一块
  - 冗余模式为奇偶校验
  - Check disk的数量是 1
  - 写操作遵循read-modify-write cycle，但只需读一块
- LEVEL 5： Block-Interleaved Distributed Parity
  - 基本想法与Level 4 相同
  - 冗余信息分布在不同的盘上，避免check disk成为瓶颈
  - 性能最好

# RAID LEVEL

- LEVEL 6：P+Q redundancy
  - 以Level 5为基础
  - 在大量盘的情况下，防止两个盘同时出错
  - 采用Reed-Solomon冗余模式，有2个Check disk
- Raid level的选择
  - Level0代价小
  - Level 0+1适用于小型系统或大量写的系统
  - Level 3适用于大规模连续读、写的系统，优于Level 2
  - Level 5各种情况性能均不错，优于Level 4
  - Level 6适用于高可靠性系统

# RAID的应用

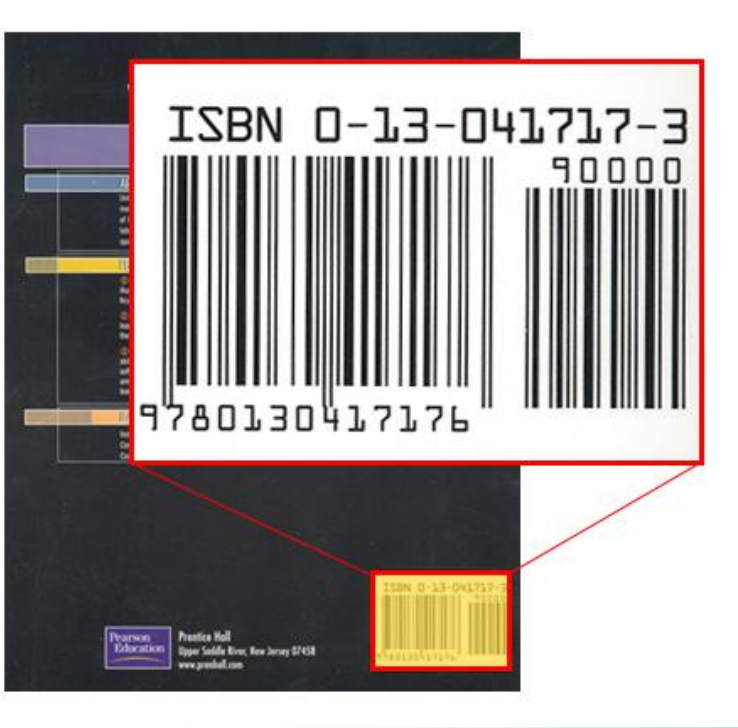
- 对于大型数据中心和重要的网站来说，RAID已成为提高运行速度和保证稳定性的廉价方案
- 比起购买8块性能稍稳定但价格不菲的硬盘来说，买9块便宜的硬盘和一些备用硬盘更加经济实用

# ISBN检测

- 每一本公开发行的书都会在封底编上一个10位或13位的编号即ISBN
- ISBN称为国际标准书号
  - International Standard Book Number
- ISBN也会用到一种检测技术
- 如果你用ISBN订购一本书，书店可以用其中的计算机校验码来检查你是否订错

# ISBN检测

将第一位数字乘以10，第二位乘以9，第三位乘以8，一次类推，一直到第九位乘以2。将他们的总和除以11，记下余数；再用11减掉这个余数就是ISBN的最后一位数字。有时候校验码的值为10，这种情况下我们用X代替。



# ISBN检测

- 下面我们验证 ISBN 0-13-911991-4

**ISBN 0-13-911991-4**

$$(0 \times 10) + (1 \times 9) + (3 \times 8) + (9 \times 7) + (1 \times 6) + (1 \times 5) + (9 \times 4) + (9 \times 3) + (1 \times 2) = 172$$

$$172 \div 11 = 15 \text{ with a remainder } 7$$

$$11 - 7 = 4$$

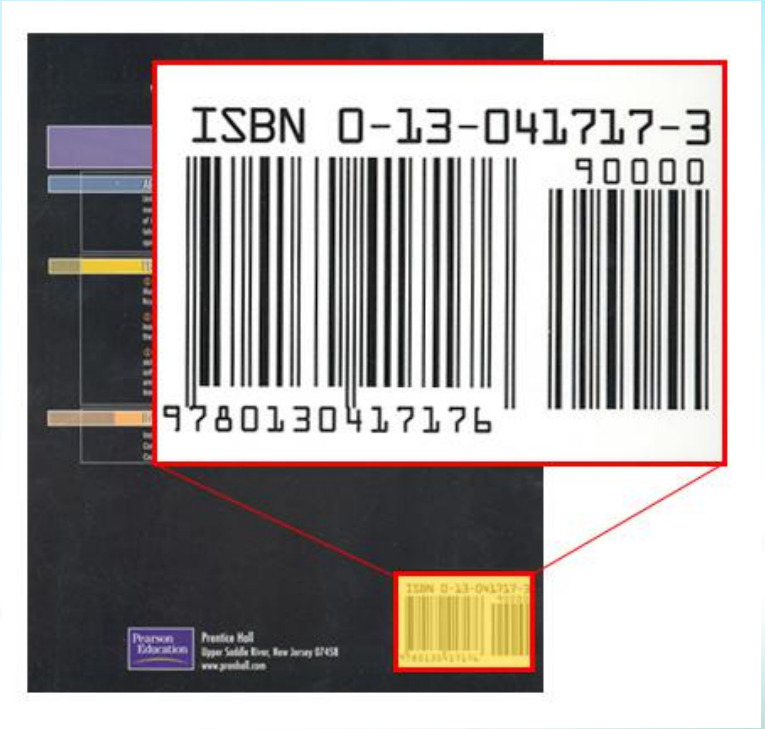
计算结果符合ISBN的最后一位吗？
如果最后一位不是4的话，我们便知道输入ISBN的时候一定发生了错误

# ISBN检测

- 如果将ISBN 0-13-911991-4中的3换成4，最后一位的校验码应该是多少？ 7
- 如果两个数字被颠倒了，即输入成0-13-191991-4，结果如何？ 校验码应该为1，错误
- 作业：ISBN 1-00-235045-X的校验码正确吗？
- 你能找出只改变0-13-911991-4中的一个数字，并保证[illegible]变的方法吗？ 这是不可能的，为什么？

# ISBN检测

当使用13位ISBN后，生成校验码的公式变简单了：只需将第1位乘以1，第2位乘以3，第3位乘以1，第4位乘以3，依此类推，直到第12位乘以3；然后将各位结果相加之后，取总和的末位数字（即除以10之后的余数）后再用10减（如果结果为10取0）即可。



# ISBN检测

- 下面我们验证 ISBN 978-897283571-4

ISBN 978-897283571-4

$$(9 \times 1) + (7 \times 3) + (8 \times 1) + (8 \times 3) + (9 \times 1) + (7 \times 3)$$
$$+ (2 \times 1) + (8 \times 3) + (3 \times 1) + (5 \times 3) + (7 \times 1) + (1 \times 3)$$
$$= 146$$

$146 \div 10 = 14$ with a remainder 6

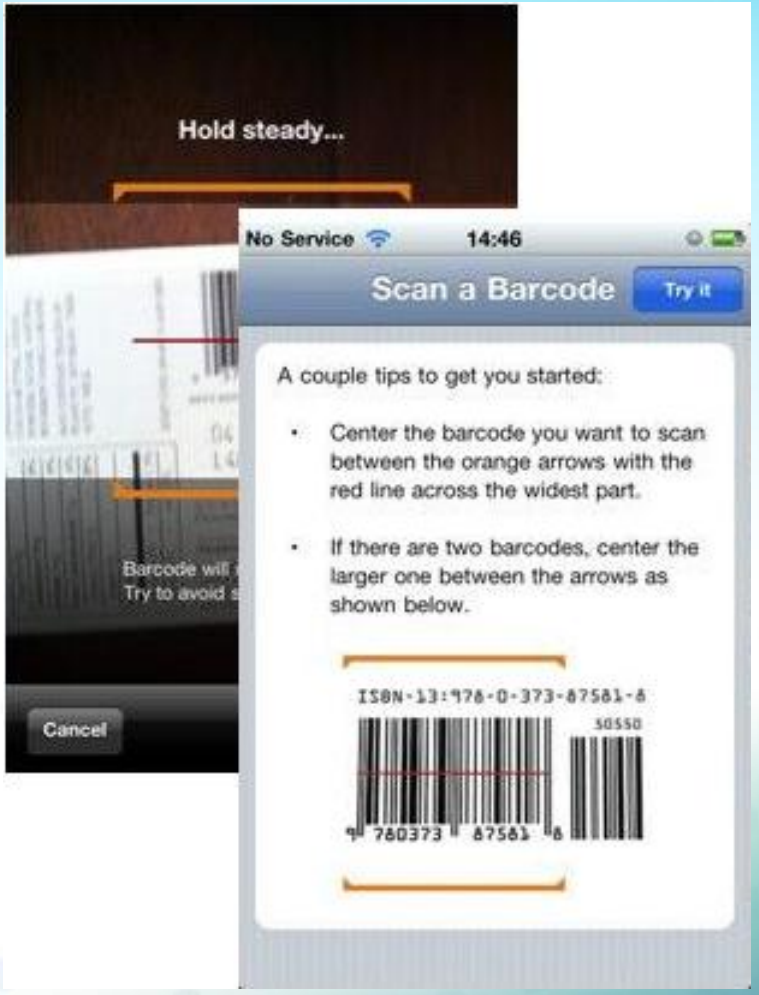
$10 - 6 = 4$

# ISBN检测

- 如果交换这个ISBN其中的两位数字（变成798-897283571-4），校验码可以发现这个错误吗？ 可以
- 如果交换数字7和2（变成978-892783571-4），校验码可以发现这个错误吗？
- 如果颠倒一个13位ISBN中连续三个数字，校验码可以发现这个错误吗？（比如颠倒978-892783571-4 的前三位数字得到879892783571-4） 不能

# ISBN检测

这样的校验码还被用于日常商品的条形码中，而且生成校验码的基本公式与前述相差无几。商品通过结账处的扫描枪时，扫描枪将校验读入的条形码校验值是否符合计算结果，如果不符合，扫描枪将鸣声报错，收银员便知道他们需要再扫一次条形码。



# 身份证编码

- 将身份证号码17位数分别乘以不同的系数。从第一位到第十七位的系数分别为：7 9 10 5 8 4 2 1 6 3 7 9 10 5 8 4 2；
- 将这17位数字和系数相乘的结果相加；
- 用加出来和除以11，看余数是多少？；
- 余数只可能有0 1 2 3 4 5 6 7 8 9 10这11个数字。其分别对应的最后一位身份证的号码为1 0 X 9 8 7 6 5 4 3 2；
  - 通过上面得知如果余数是2，就会在身份证的第18位数字上出现罗马数字的X。如果余数是10，身份证的最后一位号码就是2；
  - 例如：某男性的身份证号码是340524198001010O1X。我们要看看这个身份证是不是合法的身份证。

合格

# 纠错编码

- 奇偶校验及其应用RAID5都只能纠正一个错误
- ISBN只能发现错误，但是不能纠正错误
- 那么能否设计出一种编码既可以检测更多的错误，而且能够纠正更多的错误的呢？
- 有很多这样的编码，接下来介绍“纠错编码”

# 汉明距离

- 汉明距离
  - Hamming distance
  - 两个码字的对应比特取值不同的比特数称为这两个码字的汉明距离
  - 根据 R.W. Hamming的姓氏命名，由于在20世纪40年代继电器可靠性的缺乏上受到挫折，他开始引领纠错码的研究

# 汉明距离

- A和B的汉明距离是多少？

汉明距离是4

- B和C的汉明距离是多少？

汉明距离是3

- 任何两个符号之间的汉明距离至少是多少？

汉明距离至少是3

| 符号 | 代码 |
| --- | --- |
| A | 000000 |
| B | 001111 |
| C | 010011 |
| D | 011100 |
| E | 100110 |
| F | 101001 |
| G | 110101 |
| H | 111010 |

# 纠错编码

- 如果代码意外发生了一个错误，编码就会发现错误，因为它的结果不会是一个合法的模式
- 至少将每个代码改变3个位，它们才能变成另外一个合法的代码
- 同时可以指出原始的编码是什么，为什么？

因为修改过的代码和其原始形式的汉明距离是1，而和其他任何合法代码的汉明距离至少是2

# 纠错编码

- 例如我们接收到代码 **010100**，那么是否发生了错误，正确的代码最可能是什么？

| 符号 | 代码 | 接受到的代码 | 汉明距离 |
|---|---|---|---|
| A | 000000 | 0 **1** 0 **1** 0 0 | 2 |
| B | 001111 | 0 **1** **0** 1 **0** **0** | 4 |
| C | 010011 | 0 1 0 **1** **0** **0** | 3 |
| D | 011100 | 0 1 **0** 1 0 0 | **1** |
| E | 100110 | **0** **1** 0 1 **0** 0 | 3 |
| F | 101001 | **0** **1** **0** **1** 0 **0** | 5 |
| G | 110101 | **0** 1 0 1 0 **0** | 2 |
| H | 111010 | **0** 1 **0** **1** 0 0 | 4 |

# 纠错编码

如果只发生了一个错误，那么**010100**对应的正确的符号就是**D**

| 符号 | 代码 | 接受到的代码 | 汉明距离 |
|---|---|---|---|
| A | 000000 | 0 **1** 0 **1** 0 0 | 2 |
| B | 001111 | 0 **1** **0** 1 **0** **0** | 4 |
| C | 010011 | 0 1 0 **1** **0** **0** | 3 |
| D | 011100 | 0 1 **0** 1 0 0 | **1** |
| E | 100110 | **0** **1** 0 1 **0** 0 | 3 |
| F | 101001 | **0** **1** **0** **1** 0 **0** | 5 |
| G | 110101 | **0** 1 0 1 0 **0** | 2 |
| H | 111010 | **0** 1 **0** **1** 0 0 | 4 |

# 纠错编码

- 那么使用前面的编码，我们可以检测多少错误，修正多少错误呢？

因为代码之间的汉明距离至少是3，因此可以最多可检测到2个错误，修正1个错误。如果发生3个错误，就可能无法检测出来了。

# 纠错编码

- 如果我们能设计一种编码，每个代码和其他代码之间的汉明距离都至少是5，那么这个模式将发现多少错误，修正多少错误呢？

因为代码之间的汉明距离至少是5，因此可以最多可检测到4个错误，修正2个错误。如果发生5个错误，就可能无法检测出来了；发生3个错误就可能无法修正错误。

# 纠错编码

- 其实设计一种具有长汉明距离的编码不是一件简单的事
- 事实上，它是代数编码理论的数学分支的一部分，这个理论是线性代数和矩阵理论的子领域

# 纠错编码的应用

- 纠错技术被广泛用于增加计算设备的可靠性。
- 它们经常被用于高容量磁盘设备，以减少磁盘编码瑕疵而损坏数据的可能性
- 最初用于音频的CD格式与后来用于计算机数据存储的格式之间的主要差别就是纠错的程度

# 纠错编码的应用

- CD-DA格式包括的纠错特点使得错误率减少到2张CD只有1个错误，对视频录制足够
- 但是对于用CD向客户交付软件的公司，磁盘50%的瑕疵是无法忍受的
- 因此，人们将附加的纠错特征用于CD中来存储数据，使产生错误的可能性减少到20000个磁盘1个错误

# 纠错编码

- 除了前面介绍的编码方法之外还有众多的纠错编码
  - 线性分组码
  - 循环码
  - 卷积码
  - ……

# 作业

- ISBN 1-00-235045-X的校验码正确吗？
- 对于ISBN 978-897283571-4，如果交换数字7和2，变成978-892783571-4，校验码可以发现这个错误吗？