大数据计算基础

第三章 面向大数据的数据结构

哈尔滨工业大学
刘显敏
liuxianmin@hit.edu.cn

本讲内容

Bloom Filter

预览

- 谷歌/淘宝是怎么做下面这些事情的
  - 取样
    - 比例取样
    - 固定size取样
  - 频度统计
    - 统计item发生的次数
  - 白名单过滤
  - 统计不同查询的个数
  - 评估用户访问的均匀性
  - 发现最热item
- 简单的数据统计问题，在大数据场合下，新的方法

流

- 数据以流的方式进入
  - 搜索引擎的查询请求
  - 微博更新
- 特点
  - 无穷
  - 非平稳
    - 流的到达速率取决于用户行为，系统无法控制
- 元素（Element）
  - Tuple

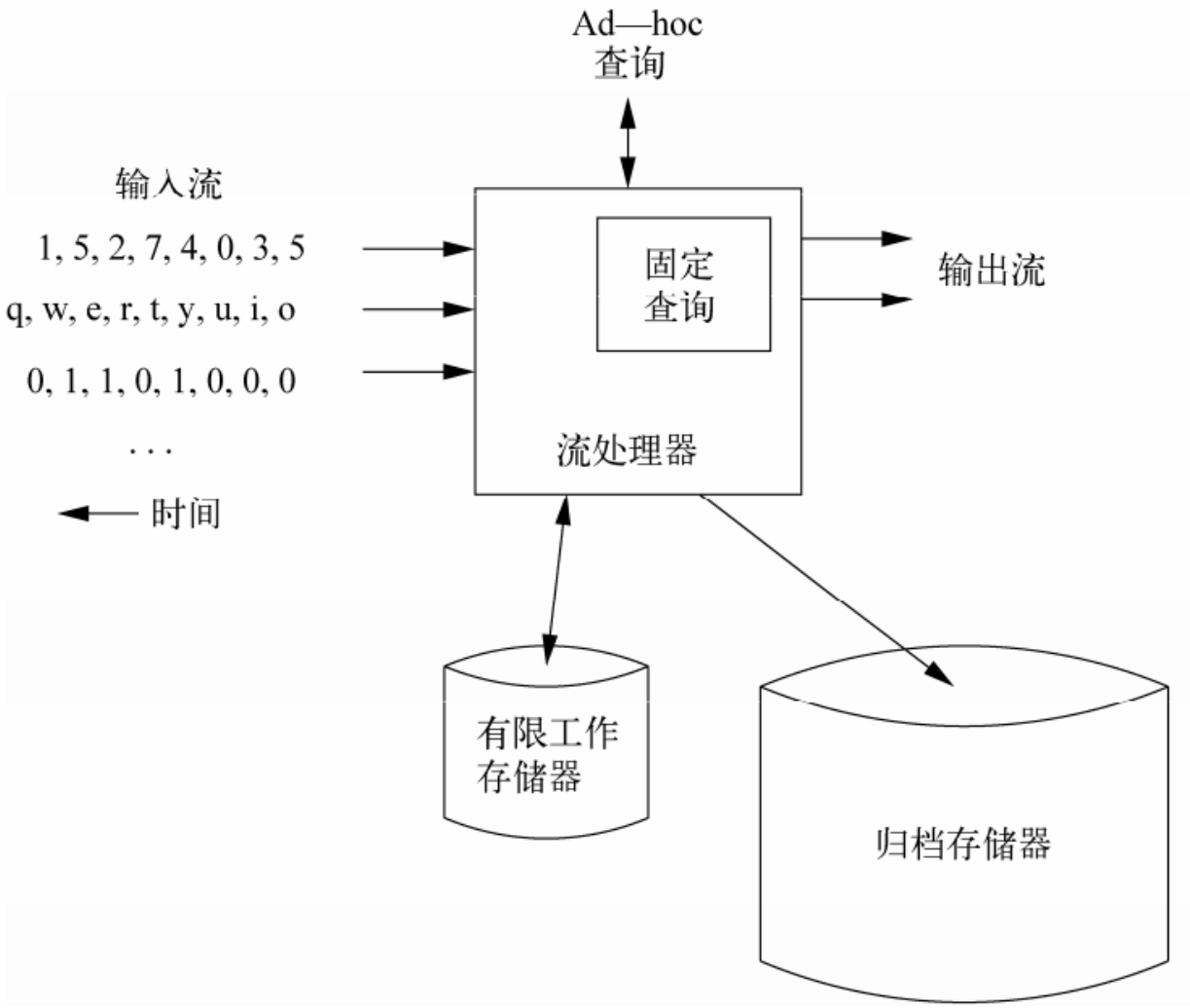
大数据下的系统限制

- 流源源不断地来
  - 要求实时处理
- 系统限制
  - 存储限制，不能存这么多
  - 存得多，处理量也大，处理能力限制
- NSA（美国棱镜门）
  - 存几个月
- 流处理
  - 有限存储情况下，怎么实时处理？
  - Online learning

模型

两种查询

1. 固定查询：
   - Standing query
   - 从不停止
   - 例：
     - 历史最高温度
   - 事先写好
2. Ad-hoc查询
   - 不全存，但还存一些内容
   - 根据这些存储内容应答



问题

- 取样：
  - 随机取样（Sampling）
  - 过滤（白名单）：选取特定属性的元素（Filtering）
- 计数（一定窗口内）
  - 有多少个不同的元素？（distinct elements)
  - 各元素的Popularity？
    - 特征：各阶矩
  - 谁最流行?

应用

- Google：
  - 查询流
  - 发现最流行的查询关键字
- Yahoo：
  - 发现最流行的页面
- 微博：
  - 发现最热的话题
  - 找人
- 传感器网络
- 电话记录
  - 美国，棱镜门
- 网络交换机
  - 流量统计，优化路由
  - 检测DDoS攻击

抽样

- 两种抽样
  - 固定比率抽样
    - 1 in 10
  - 固定Size抽样
    - 总是保持s个元素

固定比率抽样

- 应用场合
  - 搜索引擎，一个用户的搜索中，重复的有多少？
  - 存不了全部，可以存1/10
- 最明显的办法
  - 每来一个query
  - 生成一个随机整数：0…9
  - 如果是0，就存起来
  - 1/10的采样
  - 然后统计其中的用户重复搜索比例
- 对吗？

有问题

- 假设：一个用户所有搜索字符串中，x个查询了一次，d个查询了两次，没有其他查询。
- 重复查询占比：d/(x+d)
- 随机采样10%后，重复查询占比是怎样的？
  - 采样后，获得(x+2d)/10个查询，其中x/10个查询是属于x，肯定只出现一次
  - 针对d的2d/10个查询
    - d中任一查询，两次都被抽中的概率为1/10×1/10 = 1/100
    - 所以，平均有d/100个查询会被抽中两次，占2d/100个查询
    - 剩下2d/10 – 2d/100 = 18d/100次查询，也只出现一次。
- 结果 $\frac{\frac{d}{100}}{\frac{x}{10}+\frac{d}{100}+\frac{18d}{100}} = \frac{d}{10x+19d}$
- 不等于d/(x+d)。错误

正确方法：按用户采样

- 挑1/10的用户，观察它们的全部查询
- 采样方法
  - Hash(User ID) mod 10，把用户分到十个桶中
  - 选第一个桶的用户（hash后结果为0）
- 挑2/10的用户怎么办？
  - 选前面两个桶

固定Size抽样

- 总是保持s个元素
  - 这s个元素，是对过去所有元素的均匀取样
  - 即：过去所有元素，进入这s个元素的概率相同
- 直观方案：
  - 全存起来，然后从中随机挑s个
- 大数据下，因为存储空间的限制，不可行
- 流方案
  - 进来一个新元素时，
    - 新元素以概率p进入s
    - 原有的s个元素按一定的概率从s中剔除

新元素进入S的概率p

- 假设已到达n个元素，它们以s/n的概率被采样，组成s个元素的集合
- 新进来一个元素，一共到达了n+1个元素。
  - 这n+1元素，以相同概率进入s
    - 这个概率： s/(n+1)
  - 所以，这个新元素以s/(n+1)的概率进入s

• p = s/(n+1)

S中原元素的剔除策略

- 原来在s个元素集合中的元素，随机剔除一个
- 不被剔除的概率

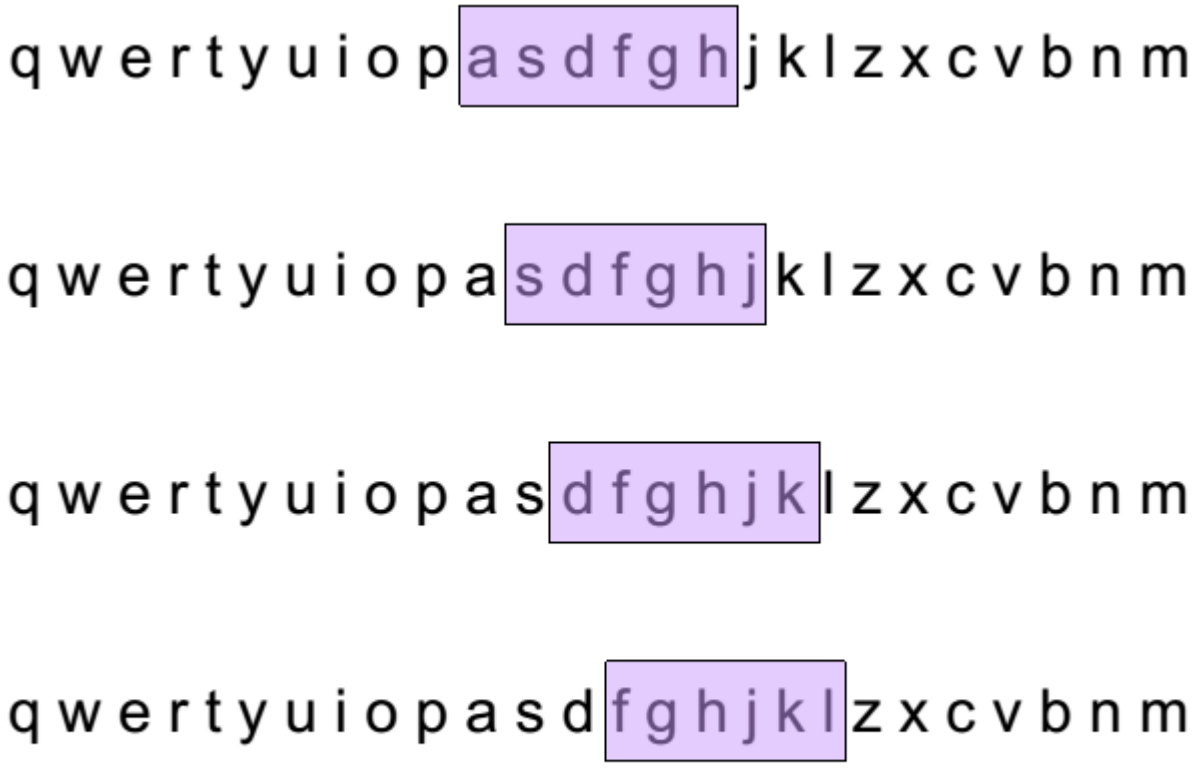
$$(1-\frac{s}{n+1})+(\frac{s}{n+1})(\frac{s-1}{s})=\frac{n}{n+1}$$

新元素不进s的概率　　新元素进s，但在s中不被剔除的概率

- 原先，这n个元素，是以s/n概率进入s的。
- 这一轮过后，任一元素留在s中的概率 $\frac{s}{n}\cdot\frac{n}{n+1}=\frac{s}{n+1}$
- 和新到元素的留下概率s/(n+1)相等
- 结果：所有n+1个元素，以s/(n+1)的概率留下

示例

- N = 6


q w e r t y u i o p a s d f g h j k l z x c v b n m
q w e r t y u i o p a s d f g h j k l z x c v b n m
q w e r t y u i o p a s d f g h j k l z x c v b n m
q w e r t y u i o p a s d f g h j k l z x c v b n m
Past
Future

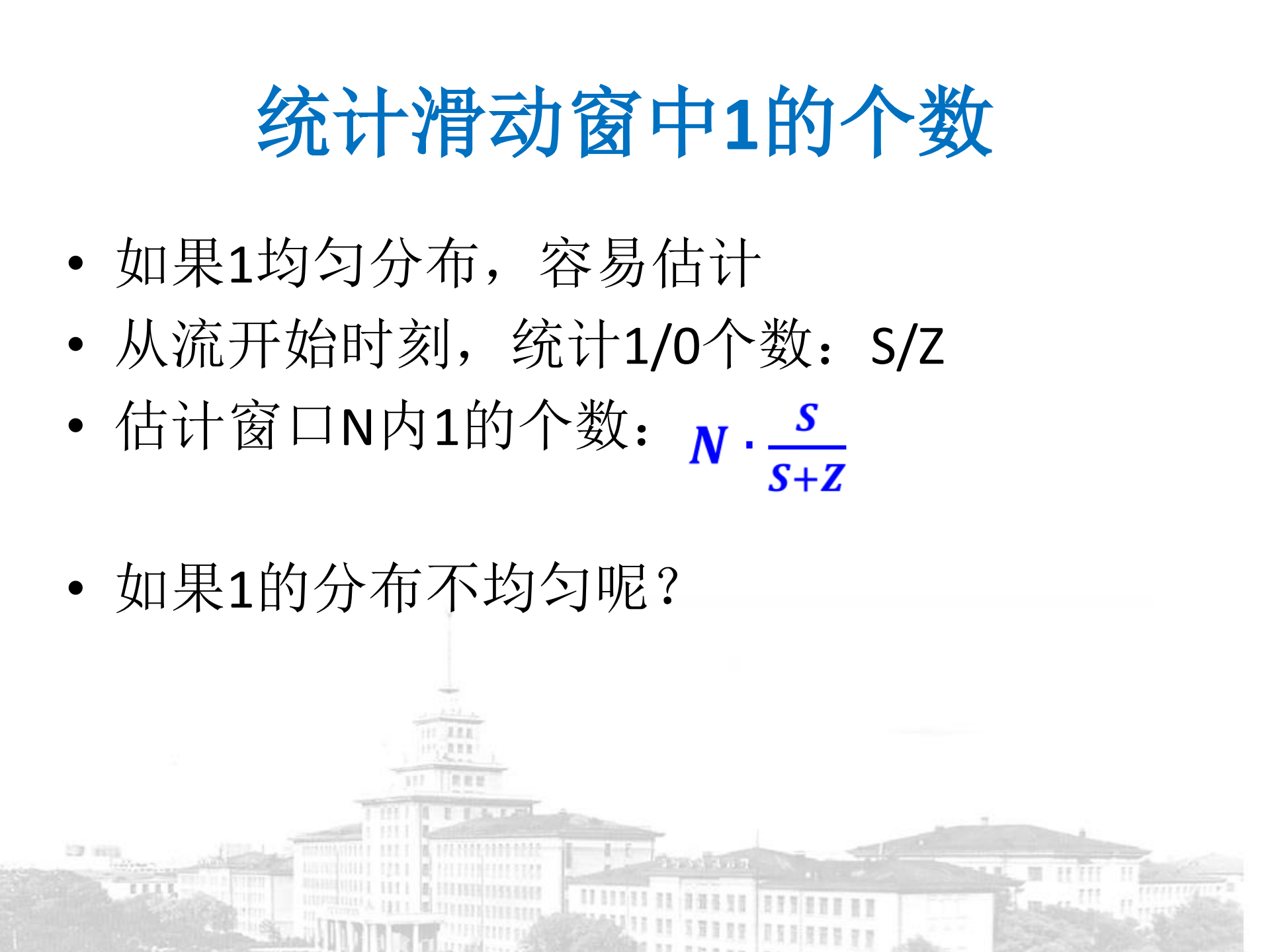
应用：统计滑动窗中1的个数

- 频率
- 简单方案
  - FIFO，窗口大小：N
  - 存起来
  - 然后统计
- 但是：N太大(Billion)/流太多(Billion)，存不下。怎么办？
  - 近似方案

统计滑动窗中1的个数

- 如果1均匀分布，容易估计
- 从流开始时刻，统计1/0个数：S/Z
- 估计窗口N内1的个数：$N \cdot \frac{S}{S+Z}$

- 如果1的分布不均匀呢？

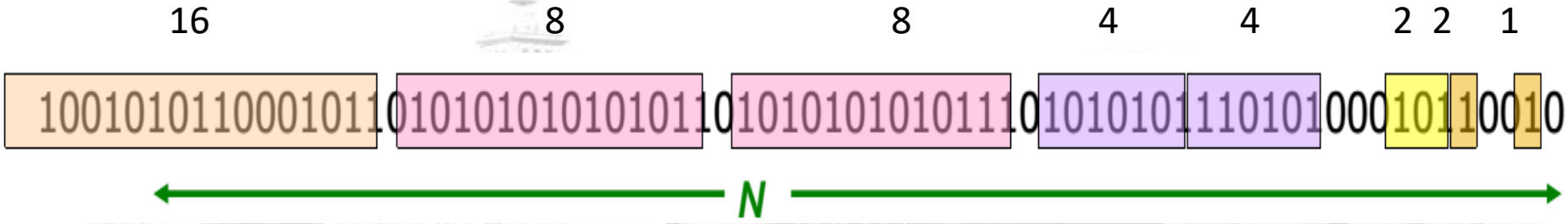


DGIM方法

- 每个流，存储 $O(\log^2 N)$ 比特
- 结果误差不超过正确结果的50%
  - 可以进一步减少

DGIM

- [Datar , Gionis, Indyk, Motwani]
- 指数窗口
- 每个窗口中包括 i 个1， i : 2的幂（指数增长）
- 同样i的窗口最多可以有两个
- 窗口不重叠，可以不连续（中间可以隔0）



DGIM需要的存储空间

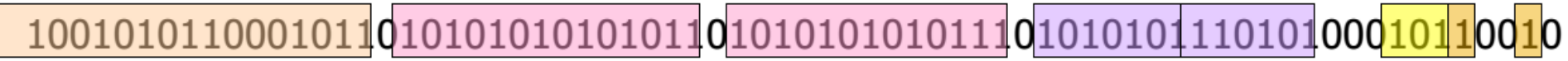
- 每个子窗（Bucket）有一个时标，记录结束时间
  - 取值范围 1 … N
  - 需要 $O(log_2 N)$ 比特存储空间
- 每个bucket记录自己包含的1的个数
  - 取值范围：1…logN
  - 需要 [O(log log *N*) bits] 存储空间

更新

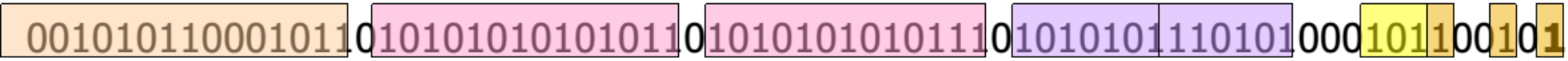
- 新元素到了
- 如果一个Bucket的end time已超过当前时刻 - N，drop它
- 如果新元素是0，什么也不做
- 如果是1
  - 创建一个Bucket， size = 1， end time = 当前时间
  - 如果有3个1，就合并前两个2。
  - 依次类推，如果有3个一样的小的，就合并前两个为一个大的。
    - 雪崩式前滚

示例

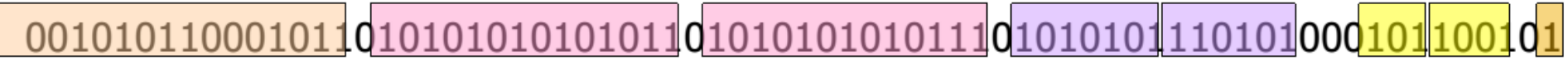
Current state of the stream:



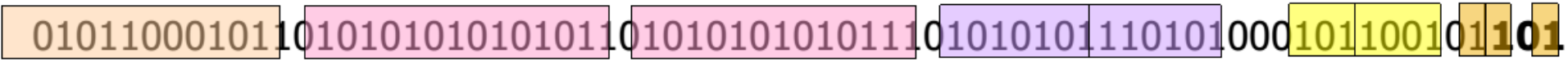
Bit of value 1 arrives



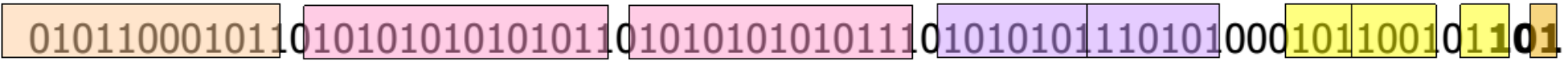
Two orange buckets get merged into a yellow bucket



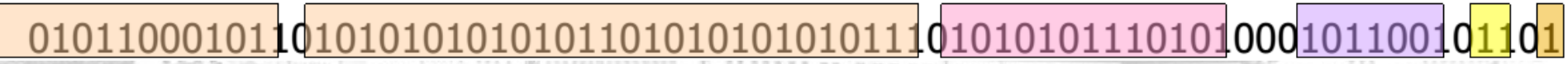
Bit 1 arrives, new orange bucket is created, then 0 comes, then 1:



Buckets get merged…

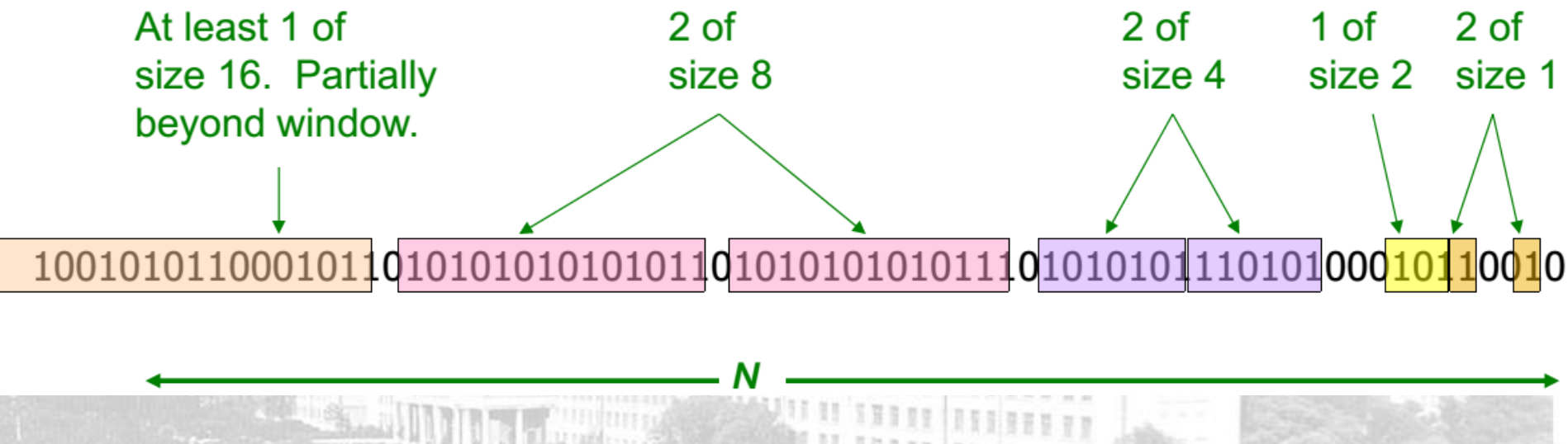


State of the buckets after merging



估计1的个数

- 除了最后一个bucket，把其他bucket的size相加
  - Size就是其中1的个数
- 再加上最后一个Bucket size的一半
  - 因为最后一个bucket，只是最后一位还在N里，



Error bound：50%

- 假设最后一个bucket的size：2^r
- 我们在统计中算了它的一半“1”，所以，最多产生2^{r-1}的错误
- 比它size小的bucket有2^{r-1}，2^{r-2}，2^{r-3}，…，1，每种至少有一个
- 所以，它们包含的“1”的个数至少为：$2^{r-1} + 2^{r-2} + 2^{r-3} + \ldots + 1 = 2^r - 1$.
- 最后一个bucket在窗口中至少还有1个“1”，所以，“1”的个数至少为2^r
- 所以，最大的错误率：$2^{r-1} / 2^r = 1/2 = 50\%$

扩展

- 同样size的bucket数目可以是r或r-1个。 r > 2
- 最大Size的bucket，可以有1,…,r个
- 错误的上界1/(r-1)
- 实践中，根据需要选择r

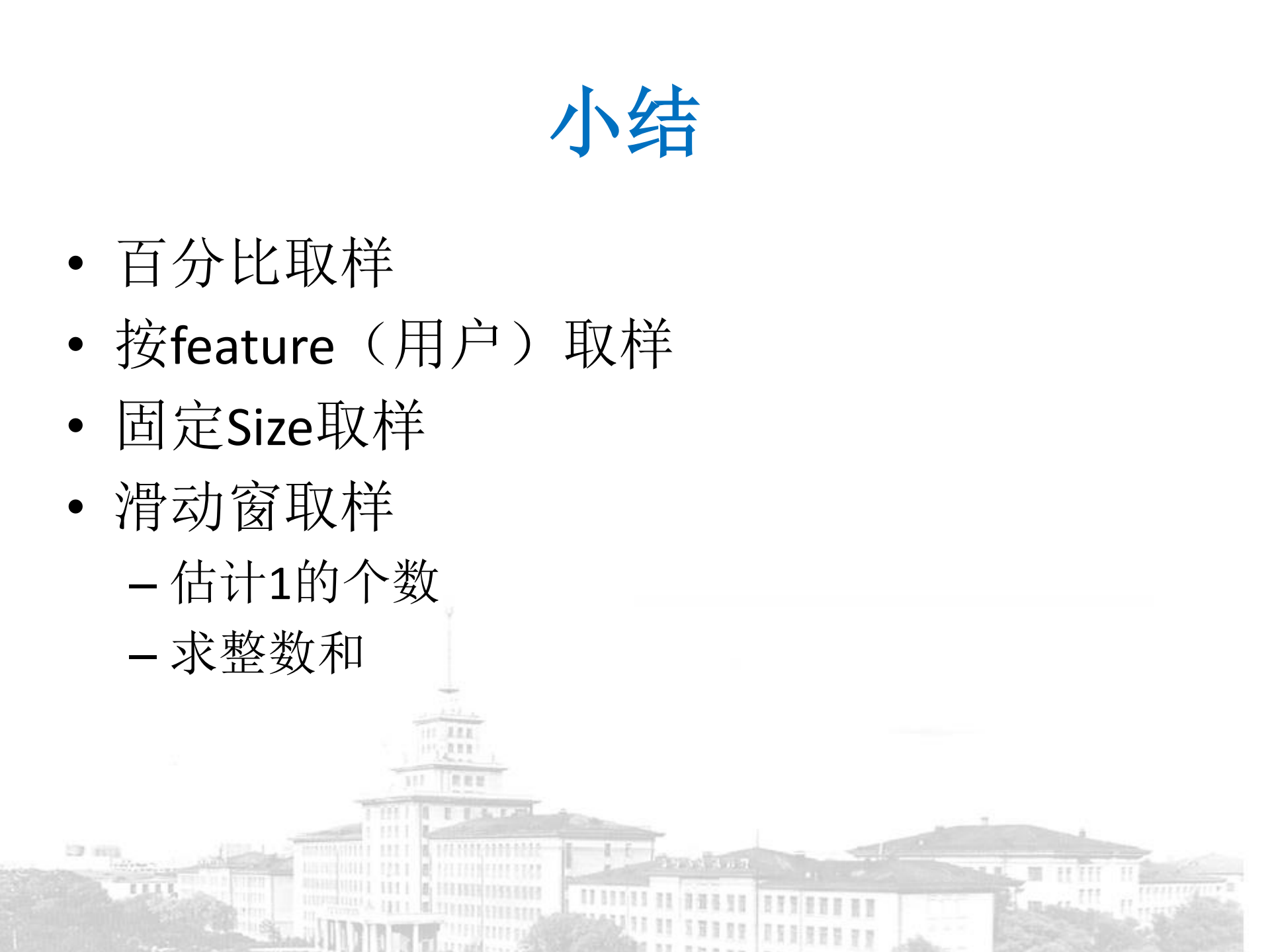
应用：窗口内整数的和

- 把整数的每一个bit作为一个stream
- 统计每一个stream的1的个数，C_i
- 求和：

$$\sum_{i=0}^{m-1} c_i 2^i$$

小结

- 百分比取样
- 按feature（用户）取样
- 固定Size取样
- 滑动窗取样
  - 估计1的个数
  - 求整数和



Bloom filter

- Bloom是一个人
- 从stream中选择符合特定条件的元素
- 例1：垃圾邮件检查
  - 白名单
- 例2：Google Alert
  - Pub-Sub系统，每个人可以设定订阅的关键词
- 明显的方法
  - 建立Hash表，查询，命中
- 大数据下，filter太多，数据太多，怎么办？
  - 包括10 billion 个白名单

初始化

- 白名单中包括s个允许的key值
  - s = 1 billion
- n个检查位，n >> s，初始化为0
- 把这s个白名字Hash到1,…,n上
  - 对应的bit位设1
- 最后，n中大约有s个“1”
  - 事实上小于s个，因为会重合。

到底有几个1？

- 一个白名字，被均匀地撒在n个比特上
  - 撒上概率：1/n
- 一个比特位，没有被撒上的概率
  - 被1个白名字错过的概率：$1 - 1/n$
  - 被所有s个白名字都错过的概率
    - $(1-1/n)^s = (1-1/n)^{n(s/n)}$
      - 近似等于 $e^{-s/n}$
- 所以，一个比特位，被撒上的概率
  - $1 - e^{-s/n}$
- 总共，$n(1 - e^{-s/n})$个比特位被撒上
  - 值为“1”

检查

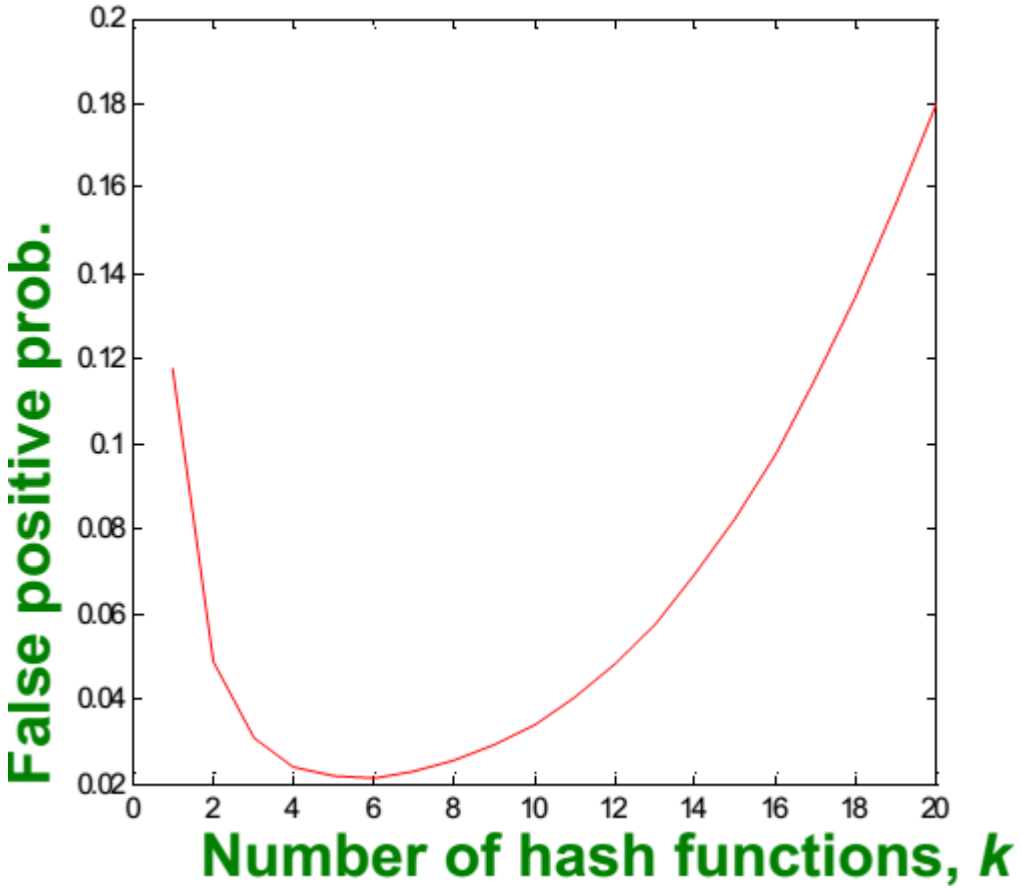
- 来了一个邮件，把发件人地址，hash一下，如果对应的比特位为0，肯定不在白名单里，Reject
- 不在白名单里，也会被均匀撒在n个比特位上
  - 如果那个比特位碰巧是“1”，就会pass
  - False positives - 假阳（FP）
    - Pass：Positive
- 和n中“1”的比例有关，
  - $n(1 - e^{-s/n})/n = 1 - e^{-s/n}$
- 所以，可以通过增加n，降低FP概率
  - $s = 10^9$, $n = 8 \times 10^9$，概率 $1 - e^{-1/8} = 0.1185 \sim 1/8 = s/n$

改进：多个hash函数

- 初始化
  - 对s中任一元素，用k个独立hash函数，分别撒k次
  - “1”的个数：
    - 类似前面，只是撒了ks次
    - $n(1 - e^{-ks/n})$
- 检查
  - 来一封信，用这k个hash检查，全部为“1”才行。
  - False positive率
    - 混过去一个hash函数，概率$(1 - e^{-ks/n})$
    - 混过去全部k个hash检查，概率$(1 - e^{-ks/n})^k$
- K=2， 概率 0.0493 ~ 1/20 << 1/8
  - 改进了性能

K的选择

- K不是越大越好
- 对这个例子，最优的在6的样子。

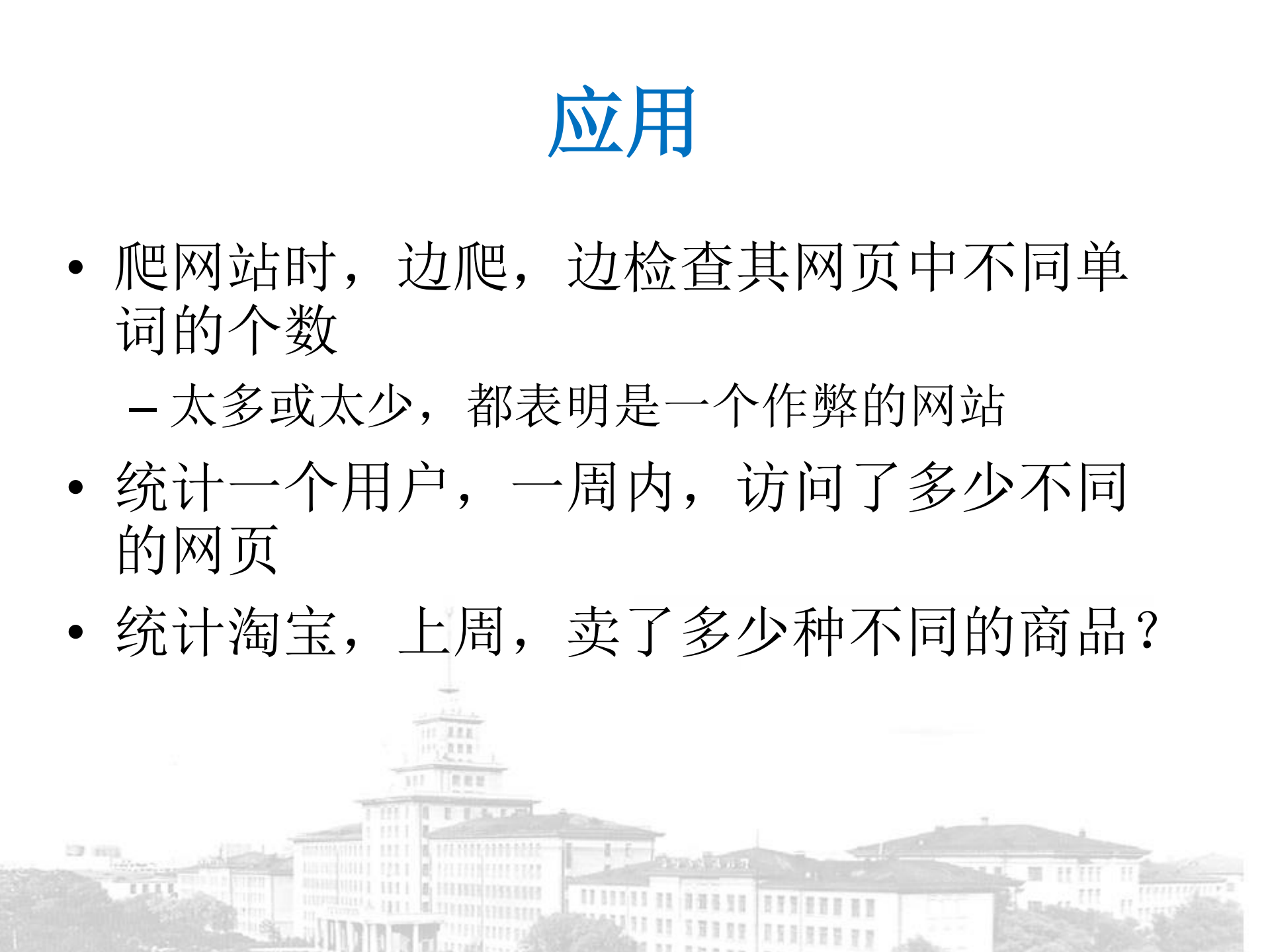


Bloom Filter总结

- 只会false positive
- 不会false negative
  - 错杀概率 = 0
- 适合预处理
  - 先筛选一些
- 适合硬件实现
- 适合并行
  - Map-reduce

应用

- 爬网站时，边爬，边检查其网页中不同单词的个数
  - 太多或太少，都表明是一个作弊的网站
- 统计一个用户，一周内，访问了多少不同的网页
- 统计淘宝，上周，卖了多少种不同的商品？



明显的方法

- 建立一个Distinct元素列表（hash表）
- 进来一个，和列表中已有的元素对照，如果不同，就加入
- 跟踪列表Size的变化

大数据情况下

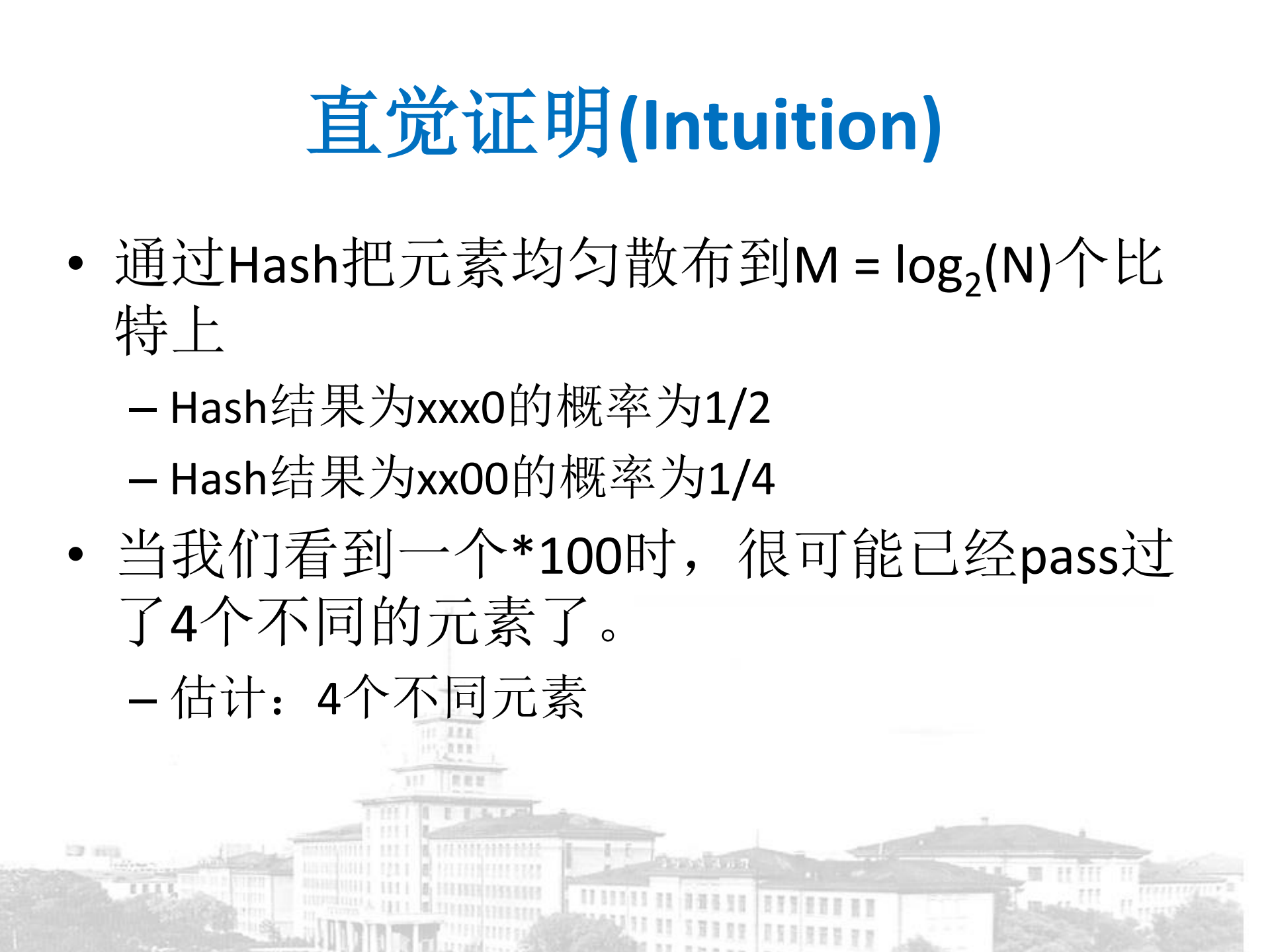
- 存不下
- 维护成本很高
- 需要
  - 减少存储要求
  - 减小计算复杂度
- Tradeoff：
  - 准确性 <> 实用性
- 估计

Flajolet-Martin方法

- 启发式算法
- 用Hash，把N个元素，映射到至少$\log_2(N)$比特上
- 检查映射的结果，看它们尾部连0的个数：r_i
  - 例：1100 -> 2
  - 1000 -> 3
- 找出最大的 r_i
  - 例：$R = \max\{2,3\} = 3$
- 估计不同元素个数为2^R
  - 例：$2^3 = 8$

直觉证明(Intuition)

- 通过Hash把元素均匀散布到M = $\log_2(N)$个比特上
  - Hash结果为xxx0的概率为1/2
  - Hash结果为xx00的概率为1/4
- 当我们看到一个*100时，很可能已经pass过了4个不同的元素了。
  - 估计：4个不同元素



更形式化的证明

- 一个元素，hash后，尾部连续r个0的概率
  - $(½)^r = 2^{-r}$
- m个不同元素hash后的m个结果，尾部都不“连续r个0”
  - 概率：$(1 - 2^{-r})^m = \left((1 - 2^{-r})^{2^r}\right)^{m2^{-r}}$

$$= e^{-m2^{-r}}$$

- 出现连续r个0的概率 $1 - e^{-m2^{-r}}$
  - $m >> 2^r$，概率为1，即总能得到连续r个0的结果。
  - $m << 2^r$，概率为0，即得不到连续r个0
- 所以，估计$m = 2^r$大致上是合理的。

实际应用

- 问题：
  - R加1，2^R就涨一倍。
  - E[2^R]无穷大
- 解决办法
  - 用多个hash函数，结果组合起来
- 组合办法
  - 平均：偶尔一个大值对结果的影响很大，不好
  - 中值：估计的结果总是2的幂次，取值不连续，也不好
- 解决方案：
  - 样本分组
  - 组内取平均
  - 组间取中值

不同元素数目扩展--矩估计

- *N*个到达的元素，统计各不同元素的流行度(Popularity)
  - 不同元素的集合A，
  - 各不同元素*i*出现的次数m_i（流行度）
- 流行度的K阶矩

$$\sum_{i \in A} (m_i)^k$$

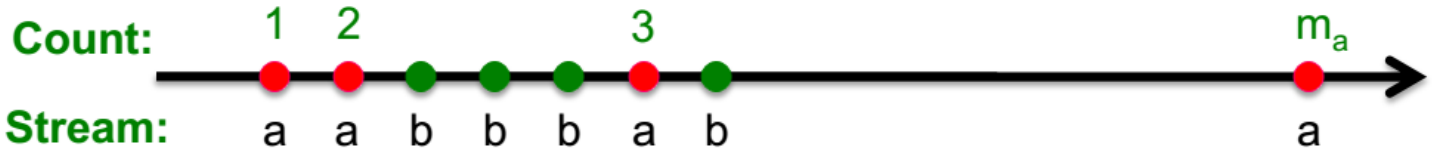
- 物理意义
  - k = 0，A的size，即不同元素的个数。 |A|
  - k = 1，N，stream长度，元素个数
  - k = 2，Surprise number（奇异数），Popularity分布均匀的度量

Surprise number（奇异数）

- Popularity分布的均匀程度的度量
- 例：
  - |A| = 11：11个视频
  - N = 100：100次用户观看
- 观看在视频上的分布
  - Case 1：10, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9
    - Surprise S = 910
    - 比较均匀
  - Case 2：90, 1, 1, 1, 1, 1, 1, 1 ,1, 1, 1
    - Surprise S = 8110
    - 更不均匀

AMS方法

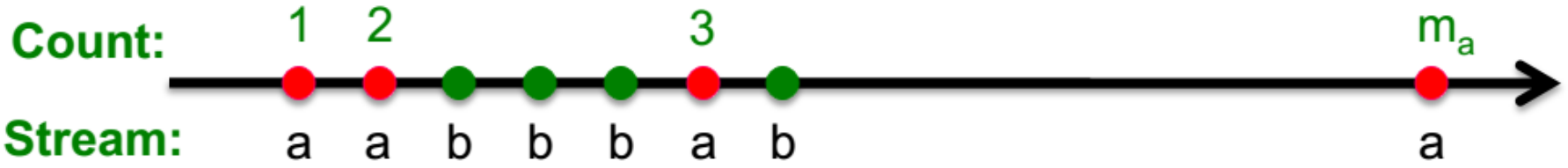
- Alon, Matias, and Szegedy
- 以k = 2为例
- 随机挑一个时刻，对stream采样
  - 采样获得的值存在X.el里
- 然后对后面进来的stream中这个值计数，直到stream结尾
  - 计数*c*存在X.val里
- 做多次，对最后的X.val，乘2，减1，乘n，然后求平均 $S = f(X) = n(2 \cdot c - 1)$



t = 1时采， X.el = a，结束时，有 X.val = m_a

t = 3时采， X.el = b，结束时，有 X.val = m_b

分析



- a
  - 如果在最后一个a采，结束时，有X.val = 1
  - 如果在倒数第二个a采，结束时，有X.val = 2
  - …
  - 如果在t = 1时采，结束时，有X.val = m_a
- 求这些 n(2*X.val - 1) 的均值

$$E[f(X)] = \frac{1}{n}\sum_{t=1}^{n} n(2c_t - 1) = \frac{1}{n}\sum_i n\,(1 + 3 + 5 + \cdots + 2m_i - 1)$$

- 因为 $\sum_{i=1}^{m_i}(2i-1) = 2\frac{m_i(m_i+1)}{2} - m_i = (m_i)^2$
- 所以 $E[f(X)] = \frac{1}{n}\sum_i n\,(m_i)^2 = \sum_i (m_i)^2 = S$
- 正是我们要的

推广

- 背后是什么？利用了
- 即 $\sum_{i=1}^{m_i}(2i-1) = 2\frac{m_i(m_i+1)}{2} - m_i = (m_i)^2$

  $-(i+1)^2 - i^2 = 2i - 1$
- 同理，求$(m_i)^3$，就对样本执行

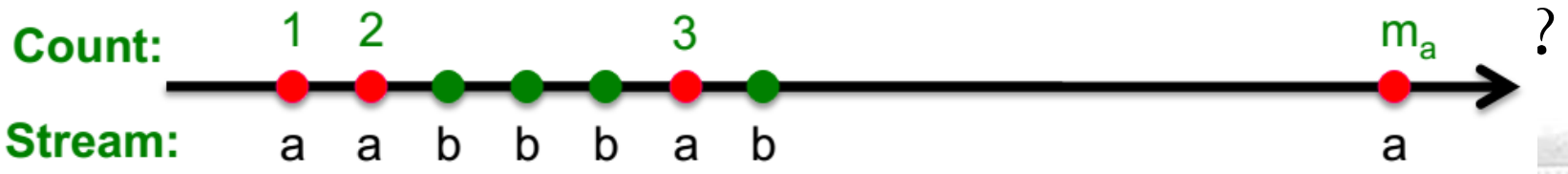
  $$c^3 - (c-1)^3 = 3c^2 - 3c + 1$$

  再做求和平均
- 推广，求$(m_i)^k$

$$n\,(c^k - (c-1)^k)$$

应用

- 根据memory大小，尽可能多随机取样，统计个数
- 对每个x.val (*c*) 求 $n\,(c^k - (c-1)^k)$
- 分组
  - 组内平均
  - 组间中值
- 局限



对Infinite Stream

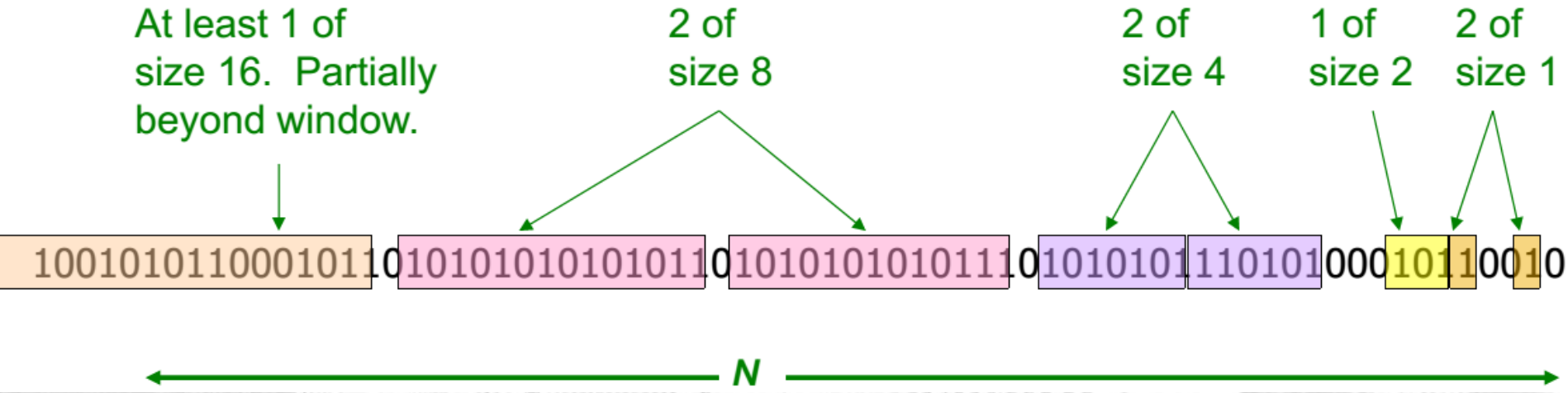
- 采用前面介绍过的固定Size采样办法
  - 采样Size：k
  - 当第n个元素到达时，以k/n的概率留下
- 在采样的*k*个样本中计算*c*
  - 近似得到一个对整个流的矩估计
  - k = 2，Surprise number（奇异数），Popularity 分布均匀的度量

发现流行

- 找出过去1个月内，被看次数超过1000的视频？

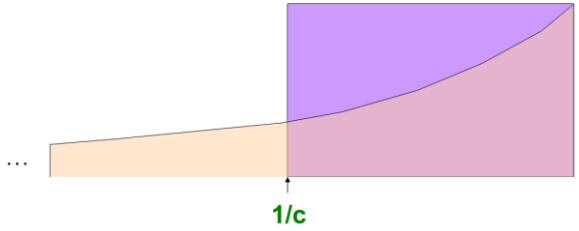
DGIM方法

- 对每个视频，建立一个1/0流，统计1的个数
- 然后挑出超出1000的视频
- 大数据下，太多视频，每个视频一个streaming不现实
  - Youtube，billions of videos



指数衰减窗方法（EDW）

- 启发式方法
  - 我们关心的是“现在”流行啥？
  - 过去的计数，让它们慢慢衰减
  - 热度 = $\sum_{i=1}^{t} a_i (1-c)^{t-i}$
    - a_i：计数
    - c：衰减系数，一般取10^{-6}, 10^{-9}
  - 权重和：$\sum_t (1-c)^t$ is 1/[1 – (1 – c)] = 1/c



- 等价于
  - 来一个新的a，把老热度乘1 - c（即衰减），然后加上这个新a
  - 实现起来非常方便

发现流行

- 实际中，为了减少存储，设一个阈值（如1/2），权重低于该阈值的，就不跟踪了
- 估计要跟踪多少个视频
- 任意时刻，所有视频热度的和
  - 来一个视频观看，以前所有视频观看带来的热度乘（1-c），再给对应视频的热度+1
  - 所有视频观看带来的热度的分布，也是一个等比级数，和为

$$\sum_t (1-c)^t \text{ is } 1/[1-(1-c)] = 1/c$$

- 因此，得分超过1/2的电影个数
  - 不会超过2/c
  - 否则，总分将超过1/c
- 所以，最多只需要跟踪2/c个视频的热度
- 省

扩展到一篮子（项集Itemsets）

- 如何用EDW对项集流行度进行跟踪呢？
- 来了一篮子元素B
  - 把所有已有的元素/项集的热度乘 1 – c （衰减）
  - 加新元素
  - 篮子里已有的元素和项集，热度+1
  - 热度 < 1/2的，扔掉
  - B中出现的新子集，如果它的所有子集都在B到来之前被跟踪着，就新增这个子集
    - 直觉：所有子集都热，那它也可能热
  - 例：
    - i,j都在集里了，那么，开始计数$\{i,j\}$
    - $\{i,j\}$ $\{i,k\}$ $\{k,j\}$都在集里了，那么，开始计数$\{i,j,k\}$

跟踪多少个？

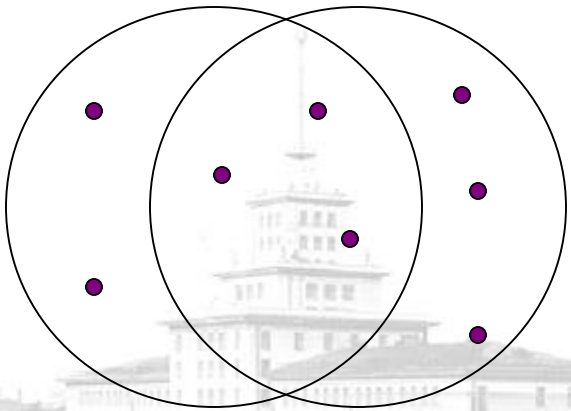
- 单item
  - < (2/c) ×篮子里的item数
- 子集
  - 和Load有关

本讲内容

MinHash & LSH

近邻搜索(Similarity Search)

- 相似度：通过计算交集的相对大小来获得集合之间的相似度，也称为Jaccard相似度
- Jaccard *Sim* (C_1, C_2) = $|C_1 \cap C_2| / |C_1 \cup C_2|$.



3 in intersection.
8 in union.
Jaccard similarity
= 3/8

应用：相似文档

- Jaccard相似度在如下问题取得较好效果：在大的语料库(web,新闻）中寻找文本内容相似的文档。这里主要指字面上的相似，而非语义上的相似
- 如果只需要检查两个文档是否严格相同，只需要逐字比较就可以。
- 很多应用里，两篇文档不是完全重复，只是大部分相同。

应用：相似文档

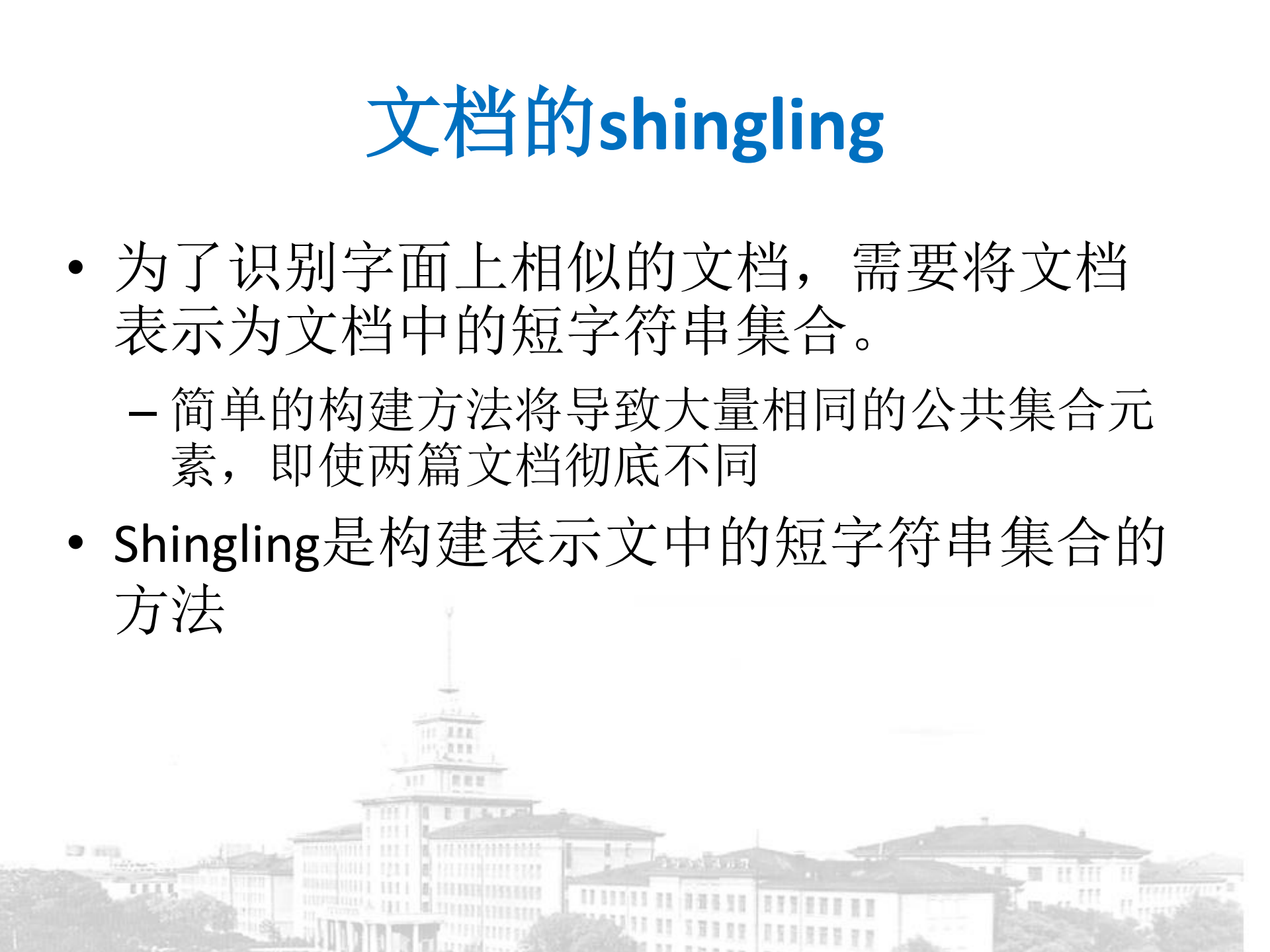
- 应用场景
  - 抄袭文档。抄袭者可能会从其它文档中将某些部分的文档据为己有，同时可能对某些词语或者原始文本中的次序进行改变。
  - 镜像页面。重要的web站点会在多个主机建立镜像页面。这些镜像的主要内容相似，但是也包括不同的内容（每个站点都指向其他站点而不指向自己）。
    - 搜索引擎需要过滤掉内容相同的镜像站点
  - 同源新闻稿。一个记者可能把一个新闻稿件投到多家报刊。每家报刊进行修改后刊发。Google new能够发现此类稿件，只显示一个版本。

应用：协同过滤

- 在协同过滤中，系统会向用户推荐相似用户所喜欢的那些项。
- 在线购物。
  - 两个用户兴趣相似，如果他们购买的商品集合有较高的Jaccard相似度。20%就很高了。
  - 两个商品相似，如果顾客集合有较高的Jaccard相似度
  - 可能需要一些辅助工作来发现相似。如两个顾客各自购买了大量科幻小说，但是这些科幻小说都不相同，通过相似度发现和聚类，把这些科幻小说归为一类，从而提高这两个顾客的相似度。

文档的shingling

- 为了识别字面上相似的文档，需要将文档表示为文档中的短字符串集合。
  - 简单的构建方法将导致大量相同的公共集合元素，即使两篇文档彻底不同
- Shingling是构建表示文中的短字符串集合的方法

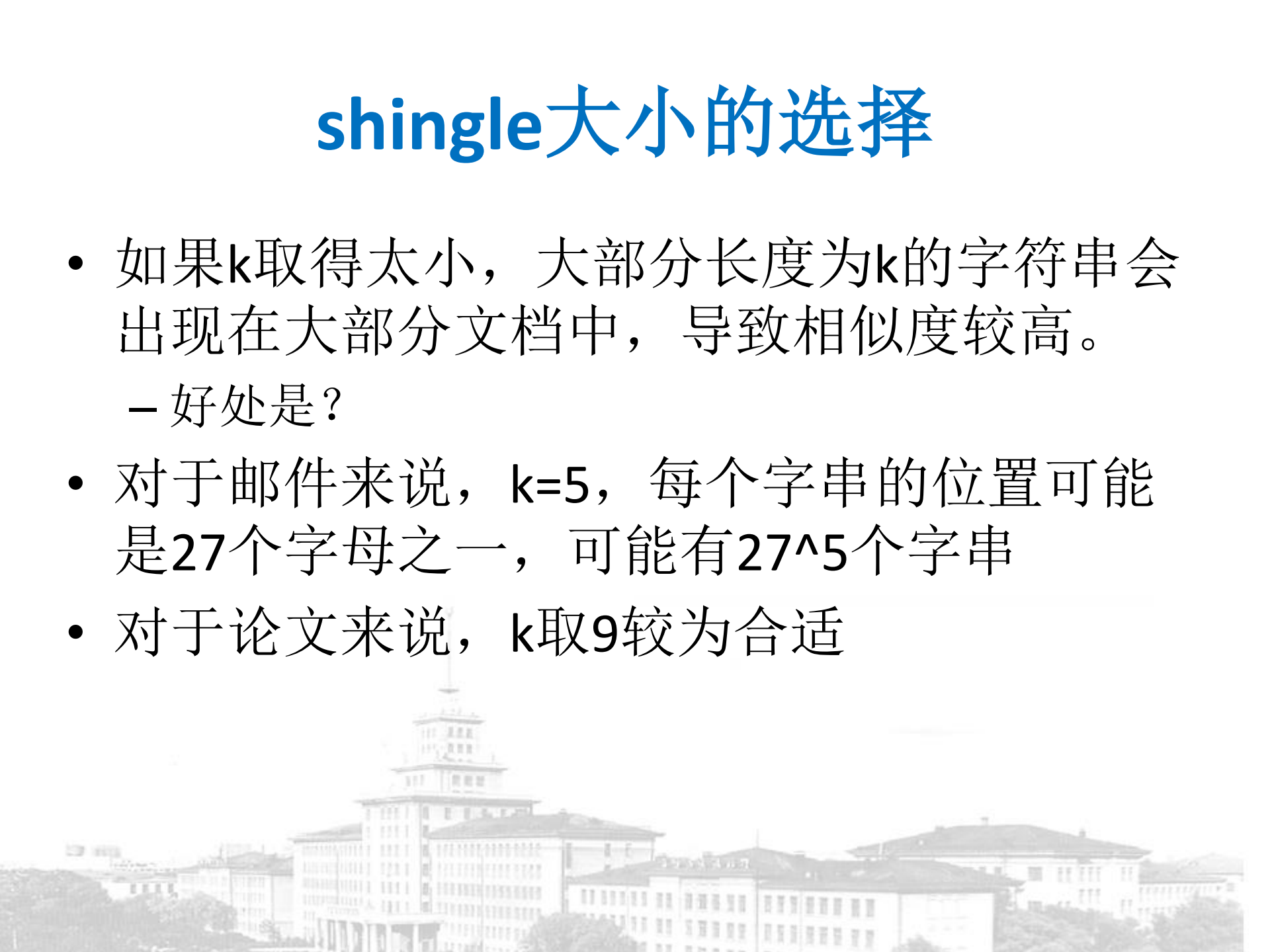


k-shingle

- 把一篇文档看成是一个字符串。文档的k-shingle就是文档中出现过的长度为k的字符串。
- 一篇文档表示为k-shingle的集合
  - 例3.3 假设文档为字符串abcdabd,k=2，则所有2-shingle组成的集合为{ab,bc,cd,da,bd}，
    - 注意ab在字符串里出现了两次，但是在集合里面只有一次

shingle大小的选择

- 如果k取得太小，大部分长度为k的字符串会出现在大部分文档中，导致相似度较高。
  - 好处是？
- 对于邮件来说，k=5，每个字串的位置可能是27个字母之一，可能有27^5个字串
- 对于论文来说，k取9较为合适



保持相似度的集合摘要

- Shingle 集合非常大。一个4 shingle 集合也是原始文件的4倍
- 想办法计算文件的签名（一个较小的文件），通过计算签名的相似性来推断文件之间的相似性。

最小哈希(MinHash)

- 矩阵的列表示各个集合，行表示所有可能的元素
- S1={a,d},s2={c},s3={b,d,e},s4={a,c,d}

元素	s1	s2	s3	s4	
A	1	0	0	1	
B	0	0	1	0	
C	0	1	0	1	
D	1	0	1	1	
e	0	0	1	0	

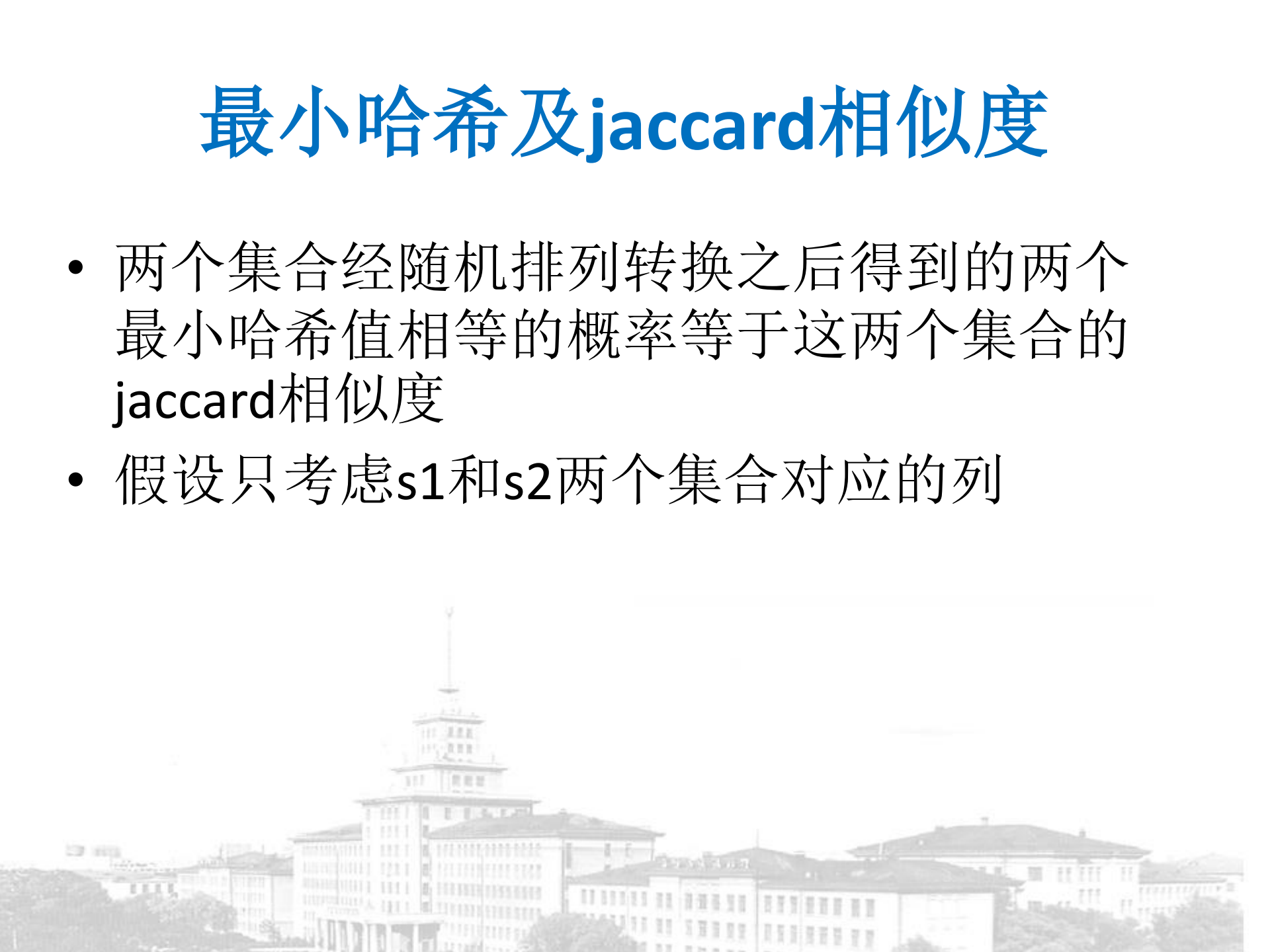
最小哈希(MinHash)

- 首先选择行的一个排列变换。
- 任意一列的最小哈希值是在该行排列顺序下第一个列值为1的行的行号。
- H(S1)=a,H(s2)=c,H(s3)=b,H(s4)=a

元素	S1	S2	s3	s4
B	0	0	1	0
E	0	0	1	0
A	1	0	0	1
D	1	0	1	1
c	0	1	0	1

最小哈希及jaccard相似度

- 两个集合经随机排列转换之后得到的两个最小哈希值相等的概率等于这两个集合的jaccard相似度
- 假设只考虑s1和s2两个集合对应的列



最小哈希及jaccard相似度

- 给定列C_1和C_2, 行可以如下划分:

	C_1	C_2
a	1	1
b	1	0
c	0	1
d	0	0

- 两列同时为0的行对于这两列的相似性没有贡献。
- 设x是两列都为1的行的数目
- y是其中一行为1的数目
- 注意$Sim\ (C_1, C_2) = x/(x+y)$.

最小哈希及jaccard相似度

- 两列的MinHash值是否相同，仅与x+y行的排列相对位置有关
- 考虑x+y行中排在最前位置的行I
- i取值为x+y中每一行的概率均是1/(x+y)
- 其中两行均为1的取值有x个
- 因此，两行均为1的概率是x/(x+y)
- H(s1)=H(s2)的概率是x/(x+y)

最小哈希签名

- 对于每一个行排列方式，每一个集合(文档)都有一个最小哈希值
- 如果有n个行排列方式（n一般是1百到数百），每个集合(文档)就能产生n个最小哈希值，把这些哈希值写成一个列向量。也称为哈希签名。
- 每个集合(文档)的列向量组合在一起，写成一个矩阵，称为签名矩阵。
- 文档的相似性就等于最小哈希值相等的概率

最小哈希签名的计算

- 操作矩阵较为困难，操作较小的签名矩阵是可以的
- 用签名矩阵记录行变换的结果，每一个位置只记录当前变换的最小的位置

1. Compute $h_1(r), h_2(r), \ldots, h_n(r)$.

2. For each column c do the following:

   (a) If c has 0 in row r, do nothing.

   (b) However, if c has 1 in row r, then for each $i = 1, 2, \ldots, n$ set $\text{SIG}(i, c)$ to the smaller of the current value of $\text{SIG}(i, c)$ and $h_i(r)$.

Row	S_1	S_2	S_3	S_4	$x+1 \mod 5$	$3x+1 \mod 5$
0	1	0	0	1	1	1
1	0	0	1	0	2	4
2	0	1	0	1	3	2
3	1	0	1	1	4	0
4	0	0	1	0	0	3

Figure 3.4: Hash functions computed for the matrix of Fig. 3.2

	S_1	S_2	S_3	S_4
h_1	∞	∞	∞	∞
h_2	∞	∞	∞	∞

	S_1	S_2	S_3	S_4
h_1	1	∞	∞	1
h_2	1	∞	∞	1

Row	S_1	S_2	S_3	S_4	$x+1 \mod 5$	$3x+1 \mod 5$
0	1	0	0	1	1	1
1	0	0	1	0	2	4
2	0	1	0	1	3	2
3	1	0	1	1	4	0
4	0	0	1	0	0	3

Figure 3.4: Hash functions computed for the matrix of Fig. 3.2

	S_1	S_2	S_3	S_4
h_1	1	∞	2	1
h_2	1	∞	4	1

	S_1	S_2	S_3	S_4
h_1	1	3	2	1
h_2	1	2	4	1

	S_1	S_2	S_3	S_4
h_1	1	3	2	1
h_2	0	2	0	0

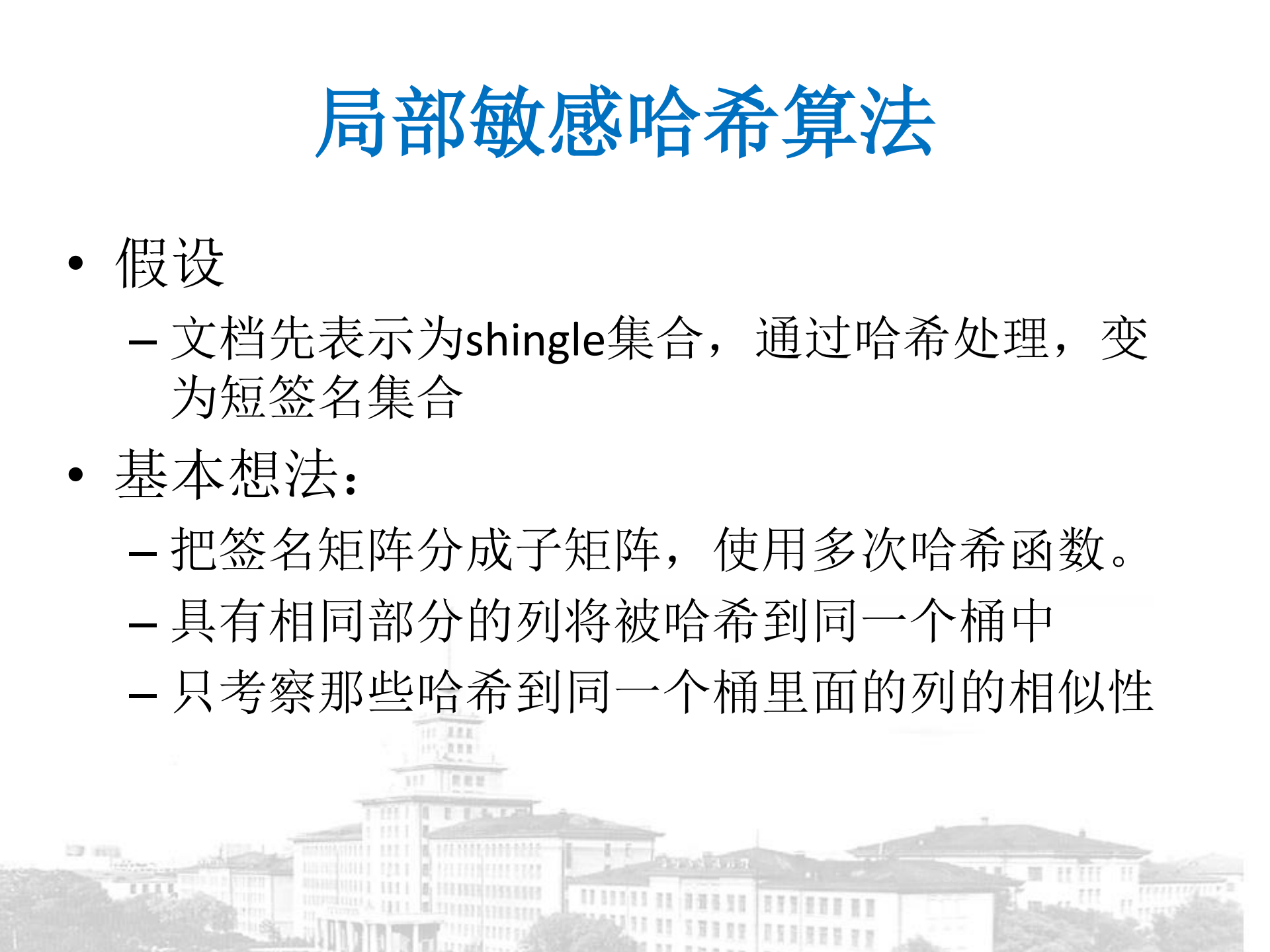
	S_1	S_2	S_3	S_4
h_1	1	3	0	1
h_2	0	2	0	0

局部敏感哈希算法

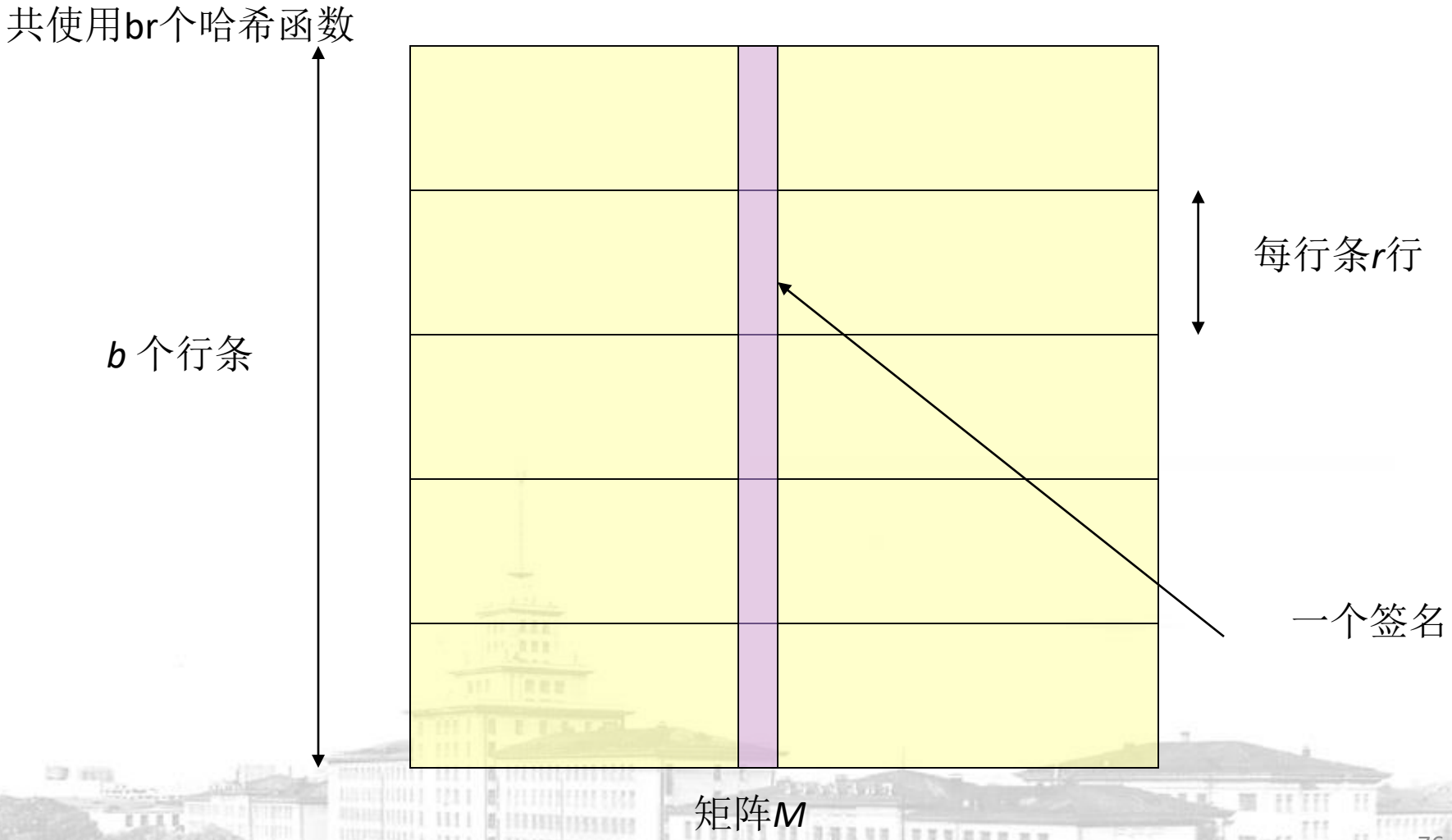
- Locality Sensitive Hashing(LSH)
- 即使可以使用最小哈希将大文档压缩成小的签名并同时保存任意文档对之间的相似度，寻找相似文档对仍然是不可能的
  - 文档的数目太大
- 实际上，只需要寻找那些相似性大于某个阈值的文档对，而不是全部文档对。这就是局部敏感哈希

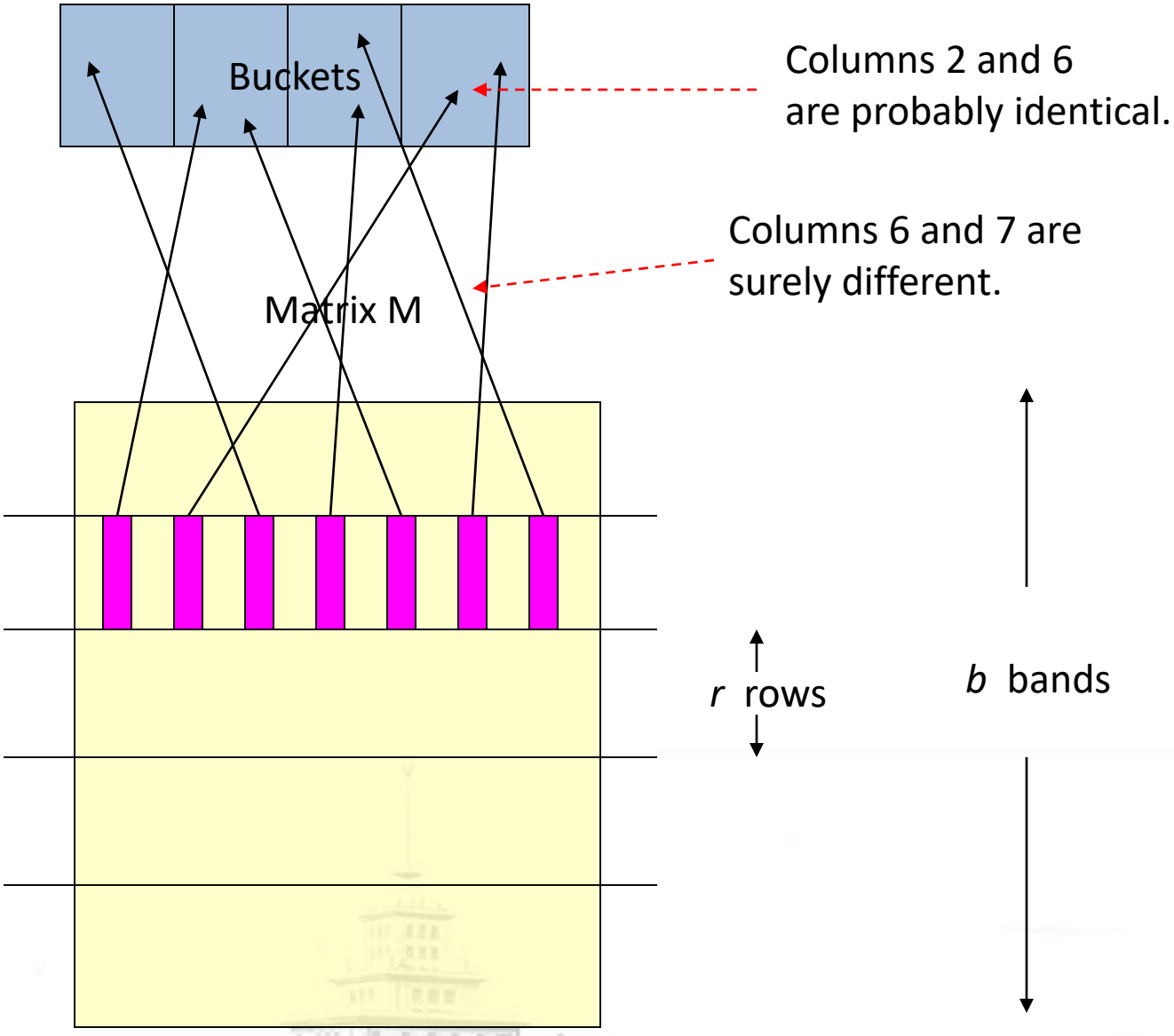
局部敏感哈希算法

- 假设
  - 文档先表示为shingle集合，通过哈希处理，变为短签名集合
- 基本想法：
  - 把签名矩阵分成子矩阵，使用多次哈希函数。
  - 具有相同部分的列将被哈希到同一个桶中
  - 只考察那些哈希到同一个桶里面的列的相似性

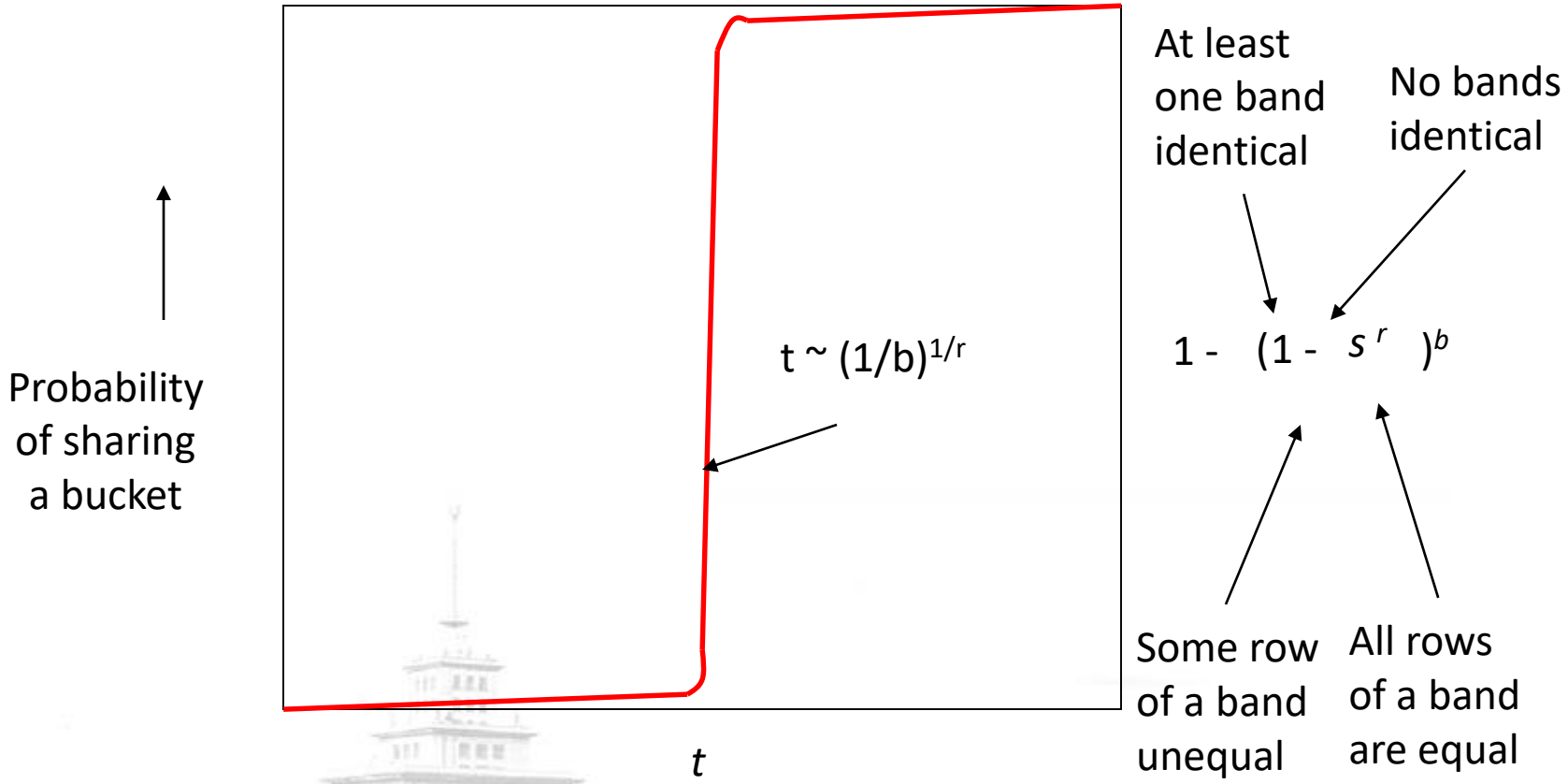


Banding for LSH




Buckets
Columns 2 and 6
are probably identical.
Columns 6 and 7 are
surely different.
Matrix M
r rows
b bands

What *b* Bands of *r* Rows Gives You



Example: $b = 20$; $r = 5$

s	$1-(1-s^r)^b$
.2	.006
.3	.047
.4	.186
.5	.470
.6	.802
.7	.975
.8	.9996

相似文档的LSH算法

- 利用LSH算法可以：
  - 找出具有相似签名的集合对，删除大部分不相似的集合对
  - 在内存里面检查这些候选的集合对

距离测度

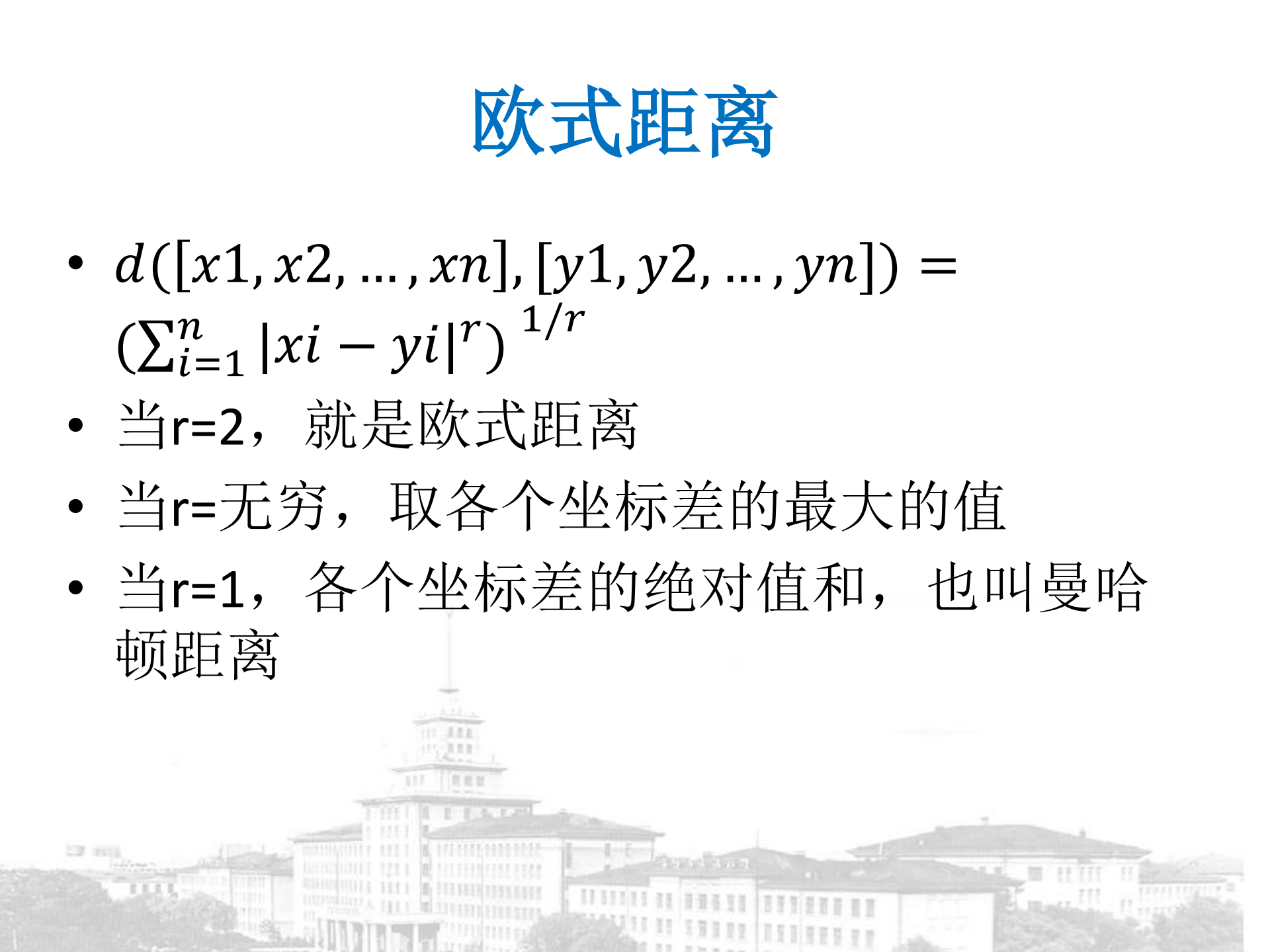
- 两个集合越相似，距离越近（越小）
  - Jaccard测度，两个集合越相似， Jaccard测度越大
  - Jaccard距离，则： 1-Jaccard测度
- 本节介绍其他的距离定义

距离测度的定义

- 距离的定义
  - 有一些点组成的集合，也称为空间
  - 在该空间定义一个函数d(x,y)，x,y是空间的点，该函数输出一个实数
  - D(x,y）>=0
  - D(x,y）=0 if and only if x=y
  - D(x,y)=D(y,x)
  - D(x,y)<=D(x,z)+D(z,y)（三角不等式）

欧式距离

- $d([x1, x2, \ldots, xn], [y1, y2, \ldots, yn]) = (\sum_{i=1}^{n} |xi - yi|^r)^{1/r}$
- 当r=2，就是欧式距离
- 当r=无穷，取各个坐标差的最大的值
- 当r=1，各个坐标差的绝对值和，也叫曼哈顿距离

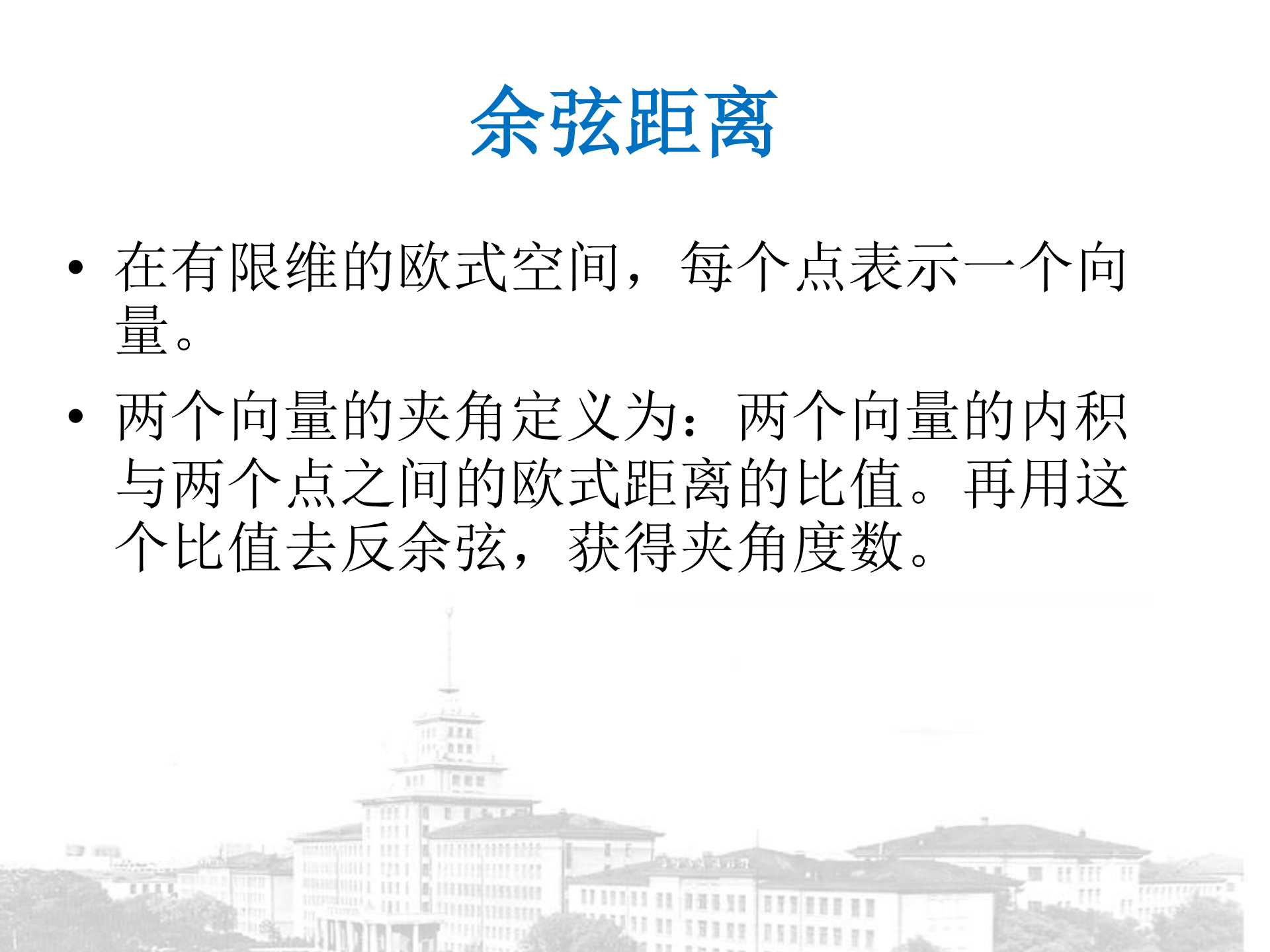


Jaccard距离

- D(x,y)=1-sim(x,y)

余弦距离

- 在有限维的欧式空间，每个点表示一个向量。
- 两个向量的夹角定义为：两个向量的内积与两个点之间的欧式距离的比值。再用这个比值去反余弦，获得夹角度数。



编辑距离

- 两个字符串的距离定义为把一个字符串变为另外一个字符串需要插入的单字符和删除的字符的最小数目
- 例如：x=abcde，y=acfdeg。把x变为y至少需要三步
  - 删除b
  - C后插入f
  - E后插入g

编辑距离

- X和y的最长公共子序列（LCS）：通过在x和y的某些位置进行删除，得到x,y的最长公共字符串。
- 编辑距离等于x的长度,y的长度的和减去两倍的LCS
- 如x=abcde, y=acfdeg。Lcs=acde
  - 编辑距离=5+6-2*4=3

海明距离

- 海明距离定义为两个向量中不同分量的个数。
- 例如：10101和11110的距离是3

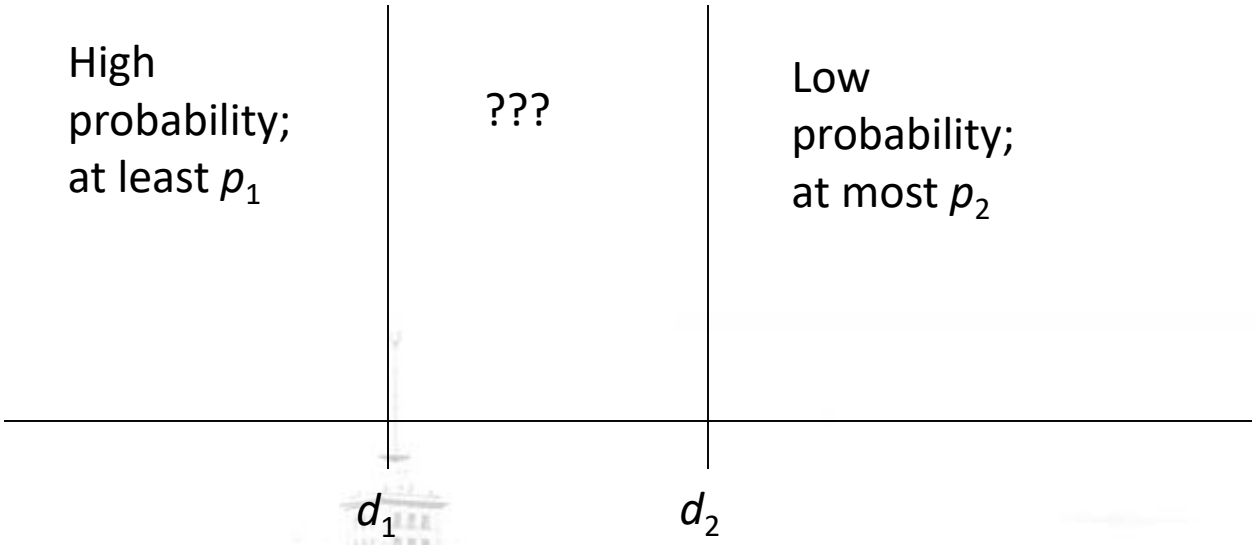
局部敏感函数理论:哈希函数簇

1. 一个“哈希”输入两个集合元素，输出结果是：这两个元素是否相同(做相似性检测）
   - Shorthand: h(x) = h(y) means “*h* says *x* and *y* are equal.”
2. 上述函数的集合称为哈希函数簇

局部敏感哈希函数簇

- 在集合S上面定义了距离空间d.
- 哈希函数簇 **H** 称为 (d_1,d_2,p_1,p_2)-*sensitive*，*如果对于S中的x和y*：令h从H中随机选择
  1. 如果$d(x,y) \le d_1$, $h(x) = h(y)$ 的概率至少是 p_1.
  2. 如果 $d(x,y) \ge d_2$, $h(x) = h(y)$ 的概率最多是 p_2.

LS 族: 说明


High probability; at least p_1
???
Low probability; at most p_2
d_1
d_2

Example

- 设 S = 集合, d = Jaccard距离, **H是所有行变换条件下的最小哈希函数构成的集合.**
- 那么 Prob[h(x)=h(y)] = 1-d(x,y).
  - 回忆最小哈希函数和 Jaccard相似性的关系的定理
- 对于Jaccard距离, 最小哈希函数簇给出了 $\left(d_1,d_2,(1-d_1),(1-d_2)\right)$-的函数簇.

局部敏感函数簇的变换

- 在签名矩阵使用的 “bands” 技术可以做通用化扩展.
- 目标: “S-曲线” 效果.
- AND构造类似 “rows in a band.”
- OR构造类似 “many bands.”

Hash函数的AND

- 给定一个函数簇 **H**, 构建函数簇 **H’， H’中的每个函数由H中的** *r* 个函数构成。
- 对于**H’** 中的$h = [h_1,\ldots,h_r]$, h(x)=h(y) 当且仅当 $h_i(x)=h_i(y)$ 对于所有 *i*.
- 定理： 如果 **H**是(d_1,d_2,p_1,p_2)-sensitive, 那么 **H’** 是 $(d_1,d_2,(p_1)^r,(p_2)^r)$-sensitive.
- 证明: 利用 h_i 独立的事实.

Hash函数的OR

- 给定一个函数簇 **H**, 构造函数簇包含**H**中*b*个函数的**H'**.
- 对于中**H'**的 $h = [h_1,\ldots,h_b]$, h(x)=h(y当且仅当 $h_i(x)=h_i(y)$ 对于某个 *i*.
- 定理: 如果**H**是(d_1,d_2,p_1,p_2)-sensitive, 那么**H'**是$(d_1,d_2,1-(1-p_1)^b,1-(1-p_2)^b)$-sensitive.

AND和OR构造的效果

- AND 使得所有的概率减少, 如果恰当的选择 r, 可以使低的概率趋于0而高的概率不变。
- OR 使所有的概率增加，如果恰当的选择 b , 可以使高的概率趋于1而低的概率不变。

AND-OR 复合

- 概率 p 变为 $1-(1-p^r)^b$.
  - 前面讲过的 “S-curve” .
- 例子: 考虑 **H** 和利用$r = 4$，AND构造的**H’**. 然后, 利用**H’**, 利用$b = 4$，OR构造**H’’**.

函数$1-(1-p^4)^4$的表

p	$1-(1-p^4)^4$
.2	.0064
.3	.0320
.4	.0985
.5	.2275
.6	.4260
.7	.6666
.8	.8785
.9	.9860

例: 将(.2,.8,.8,.2)-sensitive函数簇转化为(.2,.8,.8785,.0064)-sensitive函数簇.

OR-AND 复合

- 两个概率之一 p 转化成为 $(1-(1-p)^b)^r$.
  - 同样的S-curve, 垂直和水平镜像.
- Example: 基于 **H**, b = 4,OR构造 **H'**，然后, 从 **H'**, r = 4，AND构造**H''**.

函数$(1-(1-p)^4)^4$的表

p	$(1-(1-p)^4)^4$
.1	.0140
.2	.1215
.3	.3334
.4	.5740
.5	.7725
.6	.9015
.7	.9680
.8	.9936

例:将(.2,.8,.8,.2)-sensitive函数簇转化为(.2,.8,.9936,.1215)-函数簇.

如何设计banding过程？

- 预处理时间：O(nbr)
- 空间：O(nb)
- 查询时间：$O(b(r+dnp^r))$

总结