第6讲：数据流挖掘

武永卫
清华大学计算机系

数据流挖掘

- 数据以一个或多个流的方式到来，如果不对数据进行及时的处理或者存储，数据将会永远丢失；
- 数据到来的速度实在太快，以至于将全部数据存储到活动存储器（即传统数据库）起来再处理不太可能。
- 数据流处理的每个算法都在某种程度上包含流的汇总（Summarization）过程。
- 一种典型的数据流汇总的方法是只观察一个定长“窗口”；许多时候我们可能无法存下每个流的整个窗口；

数据流挖掘

本章主要内容：

1. 流数据模型
2. 流当中的数据抽样
3. 流过滤
4. 流中独立元素的数目统计
5. 矩估计
6. 窗口内的计数问题
7. 衰减窗口

流数据模型

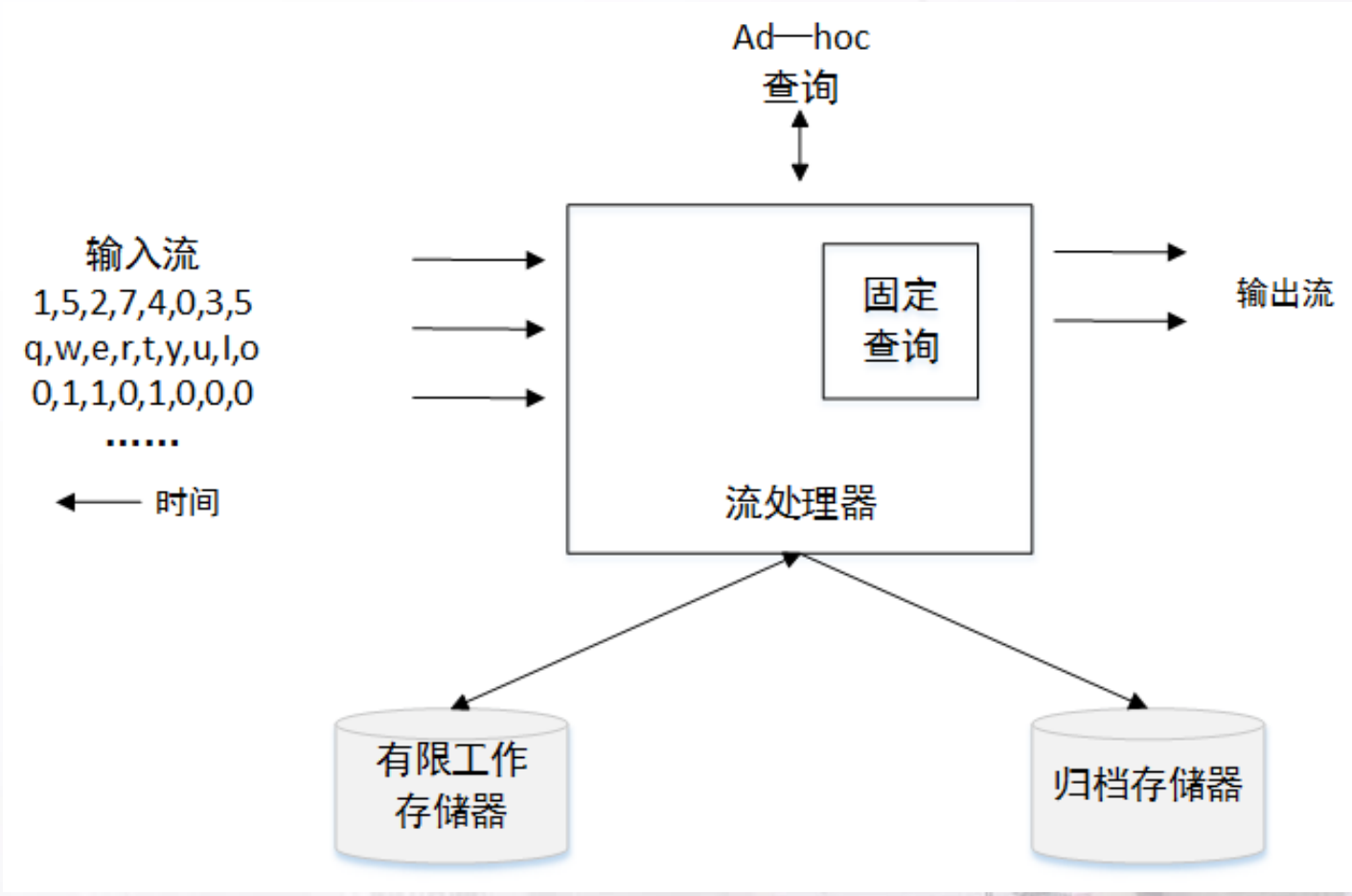
- 流数据特点
  - 数据以一个或多个流的方式到来，不同流的数据率或数据类型不必相同；
  - 一个流中的元素到达时间间隔不一定要满足均匀分布；
  - 流元素的到达速率不受系统的控制；
  - 如果不对数据进行及时的处理或者存储，数据将会永远丢失。

流数据模型

- 流数据源
  - 传感器数据
    - 大海上的温度传感器，每小时将海面温度传回基站；
    - 加上GPS，报告海平面的高度，每1/10秒传回一次；3.5M;
    - 每150平方英里部署一个传感器，海洋上100万个传感器，每天传回3.5TB数据
  - 图像数据
    - 卫星往往每天会给地球传回多个太字节（TB）的图像流数据；
    - 伦敦有600万太监控摄像机，每台产生一个图像流；
  - 互联网及Web流量
    - 交换机对数据包进行定位分发，更多功能：DOS攻击探测、根据拥塞信息重新进行包路由等；
    - Google每天收到几亿个查询；Yahoo十亿个点击；“咽喉痛”查询上升？某个链接点击率突然上升？

流数据模型

- 一个数据流管理系统



数据流模型

- 一个数据流管理系统
  - 流可以在大量归档存储器上，但是假设无法在其上进行应答查询。只有在特殊的情况下，才能在其上使用耗时的检索过程。
  - 流汇总数据或者部分流数据可以存在工作存储器（磁盘或者内存），该存储器可以用于查询应答处理，但是容量有限。
- 流处理中的若干问题
  - 通常情况下，获得问题的近似解比精确解要高效的多。
  - 一系列与哈希相关的技术被证明十分有用。

流查询

- 固定查询：
  - 海表面温度传感器：温度查过25度输出报警
  - 最近24次度数的平均值；
  - 当前的最高温度值；
  - 平均温度值；
- Ad-Hoc查询
  - 通过存储数据流的合适部分或者流概要信息
  - 在工作存储器上保存每个流的滑动窗口：最近的t个元素或者最近t个时间单位达到的所有元素；
  - 网站一个月的独立用户数：

```
SELECT COUNT(DISTINCT(NAME))
FROM Logins
WHERE time>=t
```

流当中的数据抽样

- 一个富于启发性的例子
  - 问题1：搜索引擎收到查询流，要求查询在过去一个月中，典型用户所提交的重复查询的比率是多少？并假设，我们只希望存储1/10的数据流。
  - 解决方法1：对每个搜索查询产生一个随机数（0~9），并当且仅当随机数为0时才存储该元组。这样，平均每个用户会有1/10的查询会被存储。（对吗？）
  - 问题2：查询用户提交的平均重复查询数目的比例？
  - 分析：上述的抽样机制会带来错误的结果。假定某个用户在过去一个月中有s个搜索查询只提交过一次，有d个搜索查询提交过2次，并且不存在其他超过2次的搜索查询。

流当中的数据抽样

- 一个富于启发性的例子
  - 分析：如果我们抽样的查询比例为1/10，那么在该用户的抽样查询中，提交过一次的查询数目达到我们所期望的s/10，而在出现2次的d个查询当中，只有d/100会在样本当中出现2次，该值等于d乘以该查询两次出现在1/10样本中的概率。于是，原来出现两次d个查询当中，在抽样样本中占据2d/10次，因此有2d/10-2d/100=18d/100个查询在样本中出现一次。
  - 结论：在所有搜索查询中重复搜索查询的比例的正确答案是d/(d+s)。但是，如果采用上述抽样的方法，我们得到的值为d/(10s+19d)。
  - (d/100) / (d/100+(s/10+18d/100))=d/(10s+19d)

流中的数据抽样

- 代表性样本的获取
  - 和很多其他有关典型用户统计信息的查询一样，上述例子不能从用户的搜索查询的抽样样本中得到正确答案。因此，必须要挑出1/10的用户并将他们的所有搜索查询放入样本中，而不考虑其他用户的搜索查询。
  - 解决方法：
    - 每当一个新的搜索查询到达流中时，我们会查找用户以判断其是否在已有样本中出现，如果出现，则将该搜索查询放入样本，否则丢弃。
    - 如果记录中没有出现此用户，那么产生0~9随机整数。如果随机数为0，那么就将该用户加入用户

流中的数据抽样

- 解决方法：
  - （接上页）列表，并将其标记为“in”。如果随机结果非0，那么也将该用户加入用户列表，但此时将它标记为“out”。只要能在内存中维护所有用户的列表以及它们的in/out决策表，那么上述方法就可行。
  - （改进）可以用哈希函数，将每个用户哈希到编号为0~9的10个桶的一个中去，如果哈希到桶0，那么就将该搜索查询放入样本，否则丢弃，这样避免了维护用户列表。
  - （进一步）可以得到任意用户比例的样本，比如a/b。只需哈希到0~b-1个桶中，取前a个。

流中的数据抽样

- 一般的抽样问题
  - 上述例子是一般抽样问题的一个典型代表。在一般的抽样问题中，流数据由一系列n字段元组构成。这些字段的一个子集称为关键词段。例如上例，字段为：user、query、time，其中只有user才是关键词段。当然可以使其他字段作为关键字，也可以是多个关键字组成关键词；
  - 假定抽样之后的样本规模为a/b，那么就可以将每个元组的键值哈希到b个桶中的一个，然后将哈希值小于a的元组放入样本中。

流中的数据抽样

- 样本规模的变化
  - 通常情况下，随着更多流数据进入系统，样本数目也随之增长，问题也就来了。
  - 如果存储空间大小预先设定好，那么选出的键值所占的比例必须要改变，它会随时间推移越来越低。
  - 哈希函数将键值映射到一个很大的取值范围0,1,…,B-1。我们还维护一个阈值t，它的初始值为B-1。任何时候，样本都由键值$h(k) \leq t$的元组构成。
  - 如果样本中存储的元组数目超过分配的空间大小，那么就将阈值降低为t-1，并将哪些键值K满足$h(k) = t$的元组去掉。为了提高效率，还可以将阈值降低的更多。
  - 如何去掉：可以维护一张哈希值-元组为倒排记录的倒排索引；

流过滤

流过滤

- 另一种常见的流处理方式是选择（selection）或者称为过滤（filtering）
- 只接受流当中满足某个准则的元组集合。被接受的元组会以流的方式传递给另一个过程，而其他元组被忽略。
- 如果选择的准则是基于元组的某个可计算属性（如第一个字段小于10），容易
- 选择准则包含集合元素的查找，困难；特别当集合达到无法在内存中存放时。

流过滤

一个例子

- 问题描述：假定集合S中包含了10亿个允许的邮件地址，这些地址都不是垃圾邮件地址，我们希望用这10亿个地址过滤垃圾邮件地址。
- 问题分析：流数据由邮件地址及邮件本身组成的二元组构成。由于典型的邮件地址为20或更多字节，则将S保存在内存当中是不合情理的（20G）。因此要么基于磁盘访问来确定是否让任何给定的流元素通过，要么设计一种方法，能够过滤掉大部分不想要的流元素，并且该方法所需的内存大小低于可用内存容量。

流过滤

一个例子

- 解决思路：假设我们有1GB的内存，在一种称为布隆过滤的技术当中，内存会被当成位数组使用，这样，1个字节代表有8位，所以1GB内存可以容纳80亿个位。可以设计一个哈希函数，将邮件地址映射到80亿个桶中。这时，我们将S中的每个元素映射到某位并将该位设置为1，而数组中所有其他的位仍为0。当一个流元素到达时，我们对其邮件地址进行哈希操作，如果该邮件地址哈希后对应位为1，那么就让邮件通过，但若对应的位为0，则丢弃。
- 问题讨论：不幸的是，可能会有一些垃圾邮件地址也会通过。

流过滤

- 有一些垃圾邮件地址也会通过
  - 邮件地址不在S中的流元素中有大约1/8会被哈希到位1，从而通过过滤。
  - 如何处理？
    - 大部分邮件都比垃圾邮件（有报道称80%的邮件都是垃圾邮件），因此剔除7/8的垃圾邮件得到的好处已经很大；
    - 检查通过过滤的那些邮件的邮件地址是否真正属于S，可通过二级存储来访问S本省解决；
    - 可以将多个过滤器串联起来使用，每个过滤器能够从当前数据邮件中过滤7/8的垃圾邮件；

流过滤

- 布隆过滤器
  - n个位组成的数组，每个位的初始值都为0。
  - 一系列哈希函数$h_1, h_2, \ldots, h_k$组成的集合。每个哈希函数将“键”值映射到上述的n个桶（对应于位数组中n个位）中。
  - m个键值组成的集合S。

布隆过滤器的目的是让所有键值在S中的流元素通过，而阻挡大部分键值不在S中的流元素。

（接下页）

流过滤

- 布隆过滤器

（接上页）位数组的所有位的初始值为0。对S中的每个键值K，利用每个哈希函数进行处理，对于一些哈希函数hi及S中的键值K，将每个$h_i(K)$对应的位置为1。

当键值为K的流元素到达时，检查所有的$h_1(K), h_2(K), \ldots, h_k(K)$对应的位是否全部为1，如果是，则允许流元素通过，如果有一位或多位为0，则认为K不可能在S中，于是拒绝该流元素通过。

布隆过滤方法的分析

- 如果某个元素的键值在S中出现，那么该元素肯定会通过Bloom Filter。我们必须了解如何基于位数组长度n、集合S的元素数目m及哈希函数的数目k来计算假阳（false positive）的概率。
- 使用飞镖模型模拟Bloom Filter。假设有y支飞镖和x个靶位。每支飞镖投中每个靶位的机会均等。那么飞镖投出后，预计将有多少个靶位至少会被投中一次？
  - 给定飞镖不能投中给定靶位的概率是(x-1)/x;
  - y支飞镖中全部都没有投中的给定靶位的概率是$(\frac{x-1}{x})^{y}$，该式子可以写成$(1-\frac{1}{x})^{x(\frac{y}{x})}$。
  - 当ε很小时，公式$(1-\varepsilon)^{1/\varepsilon}=1/e$近似成立，于是我们得出结论：y支飞镖全部都没命中给定靶位的概率约为$e^{-y/x}$。

流过滤

- 例子1：用上面公式计算位数组中真正的1的数目比例？
  - 我们可以将每一位看成是一个靶位，而集合S中的每个元素看成是一支飞镖。于是，某个给定位为1的概率也就是该靶位被一支或多支飞镖投中的概率。由于S中存在10亿元素，因此$y = 10^9$支飞镖，而位数组容量为80亿，因此$x = 8 \times 10^9$个靶位。所以，给定靶位未被击中的概率是$e^{-y/x} = e^{-1/8}$，而至少被投中一次的概率为$1 - e^{-1/8}$，约为0.1175。因此，一个哈希函数时，1的数目比例接近1/8

流过滤

- 推广到一般的情况
  - 此时集合S有m个元素，位数组容量为n，而哈希函数有k个。靶位的数目为x=n，飞镖的数目为y=km。因此投完所有飞镖后，某位仍然为0的概率是$e^{-km/n}$。我们的目标是使得0的比例很大，否则非S中的元素至少有一次哈希为0的概率就太小，从而出现太多的假正例。例如我们可以将哈希函数的数目k选为n/m或更小，那么0出现的概率至少为e^{-1}或37%。总之，假阳率的一个位为1的概率$(1-e^{-km/n})$的k次方，即$(1-e^{-km/n})^k$。

流过滤

- 例2：假设我们使用同样的S和同样的位数组，但是使用两个哈希函数，假阳率变为多少？
  - 这相当于往80亿个靶位上投20亿支飞镖，某个位为0的概率为$e^{-1/4}$。一个非S中的元素若要成为伪正例的话，那么就必须在两个哈希函数的作用下都映射为1，而该概率为$(1-e^{-1/4})^2 \approx 0.0493$。因此，增加一个哈希函数之后能够改进原有结果，假阳率从0.1175降到了0.0493。

流中独立元素的数目统计

- 独立元素计数问题
  - 描述：假定流元素取自某个全集。我们想知道流当中从头或某个已知的过去时刻开始所出现的不同的元素数目。
  - 问题实例：1.考虑某个Web网站对每个给定月份所看到的独立用户数目进行统计这一场景，其中全集由所有登陆集合组成。2. Google统计提交搜索的独立用户数目，全集是所有用户的IP地址。

流中独立元素的数目统计

- 独立元素计数问题
  - 解决方法一：内存中存储当前已有的所有流元素列表，可以利用哈希表或者搜索树等。缺点：当数目很大时无法存储。
  - 解决方法二：多个机器同时进行存储，每个机器处理一个或者几个流。
  - 解决方法三：将搜索结构的大部分存到一个二级存储器中，并对流元素进行分批处理。
  - 解决方法四：利用FM（Flajolet-Martin）算法进行估计。

流中独立元素的数目统计

- FM算法
  - 通过将全集中的的元素哈希到一个足够长的位串，就可以对独立元素个数进行估计。
    - 位串必须要足够长，以致哈希函数的可能结果数目要大于全集中的元素个数；（64位对于URL的哈希操作足够）
    - 选择多个哈希函数，对流中的每个元素进行哈希操作（相同元素上的哈希，结果也相同）
  - 如果流中的不同元素越多，那么我们看到的不同哈希值也会越多，同时也越能看到其中一个值变得“异常”，我们将使用的一个“异常”性质是该值后面会以多个0结束。

流中独立元素的数目统计

- FM算法
  - FM算法思想
    - 任何时候在流元素a上应用哈希函数h时，位串$h(a)$的尾部将以一些0结束，当然也可能没有0，尾部0的数目称为a和h的尾长。假设流当中目前所有已有元素a的最大尾长为R，那么我们将使用2^R来估计到目前为止流中所看到的独立元素的数目。
  - FM算法分析
    - 上述估计的直观意义：给定流元素a的哈希值h(a)末尾至少有r个0的概率

流中独立元素的数目统计

- FM算法
  - FM算法分析（接上页）
    - 为2^{-r}。假定流中有m个独立元素，那么任何元素的哈希值末尾都不满足至少有r个0的概率为$(1-2^{-r})^m$。迄今为止，这种类型的表达式我们应该不会陌生。上述表达式可以改写成$((1-2^{-r})^{2^r})^{m2^{-r}}$。当r相当大时，$(1-2^{-r})^{2^r} \approx 1/e$，因此任何元素的哈希值末尾都不满足至少有r个0的概率为$e^{-m2^{-r}}$。于是可以得出下列结论：
      - 如果m远大于2^r，那么发现一个尾部长度至少为r的概率接近1;
      - 如果m远小于2^r，那么发现一个尾部长度至少为r的概率接近0;
    - 基于以上结论，m的估计值2^R不可能过低或过高。

流中独立元素的数目统计

- 组合估计
  - 组合估计：将很多不同哈希函数下获得的独立元素个数m的估计值进行组合。我们假定每个哈希函数上得到不同的2^R的值。
  - 方法1：求这些值的平均数。
  - 方法1缺陷：对于每个流，R越大，2^R的值是翻倍增长，因此2^R的值越大，对平均值的贡献就越大，会过高的影响估计值。
  - 方法2：求这些值的中位数。中位数不会受到偶然极大的2^R值的影响。

流中独立元素的数目统计

- 组合估计
  - 方法2缺陷：估计值永远是2的幂，不论使用多少哈希函数，M的计算值都在两个2的幂之间，不可能得到非常近似的估计。
  - 方法3：将以上两种方法组合起来，首先将哈希函数分成小组，每个小组内取平均值，然后在所有平均值中取中位数。
  - 方法3分析：极大的2^R值会使得某些组的平均值很大，但是组间取中位数会将这种影响降低到几乎没有的地步。为了保证可以得到任何可能的平均值，每组大小至少是$\log_2 m$的一个小的倍数。

流中独立元素的数目统计

- 空间需求
  - 不需要将看到的元素保存起来；
  - 在内存保存的是每个哈希函数所对应过得一个整数，该整数记录当前哈希函数在已有流元素上得到的最大尾长
  - 如果只处理单个流，就可以使用几百万个哈希函数，远远多于近似估计所需要的数目；
  - 如果需要同时处理多个流，内存才会对与每个流关联的哈希函数数目有所限制；
  - 在实际应用中，每个流元素哈希值的计算时间对所用的哈希函数数目的限制更大

矩估计

- 矩计算
  - 将上述独立流元素技术推广到更一般的问题，包括不同流元素出现频率的分布的计算。
- 矩定义
  - 假定一个流由选自某个全集上的元素构成，并假定该全集中的所有元素都排好序，这样可以通过整数i来标记该序列中的第i个元素，假设该元素的出现次数为m_i，则流的k阶矩是所有i上的$(m_i)^k$之和。
  - 流的0阶矩的所有元素中不为0的元素m_i的数目（即$m_i > 0$，则加1，否则加0）。也就是说，0阶矩是流中的独立元素的个数。很显然，我们可以采用FM算法来估计流的0阶矩。

矩估计

- 矩定义
  - 流的一阶矩是所有元素m_i之和，即整个流的长度，因此，一阶矩的计算非常容易，只需要计算当前流所看到的元素个数即可。
  - 流的二阶矩是所有m_i的平方和，由于该数度量的是流中元素的分布的非均匀性，因此它有时也称为奇异数。例如：长度为100的流，其中不同的元素有11个。这些元素最均匀的分布为：其中10个元素出现9次，1个元素出现10次，奇异数为$10^2 + 10 \times 9^2 = 910$。另外一个极端是，1个元素出现90次，10个元素个出现1次，奇异数为$90^2 + 10 \times 1^2 = 8110$。
  - 在独立元素技术问题中，内存中保存的每个值都是单个哈希函数产生的最大尾长。我们将看到对二阶及多阶矩估计有用的其他形式的值。

矩估计

- 二阶矩估计的AMS算法
  - 没有足够的内存空间来计算流中所有元素的m_i，我们仍然可以在使用有限空间的情况下估计流的二阶矩，使用的空间越多，估计结果也越精确。
  - 二阶矩估计中，计算一定数量的变量，对于每个变量X，保存以下内容。
    - 全集当中的一个特定元素，记为$X.element$。
    - 一个整数，记为$X.value$，它是变量X的值。为确定该值，我们在流中均匀随机地选择1到n之间的一个位置。将$X.element$置为该位置上的元素，将$X.value$的初始值置为1。在流读取过程中，每再看到一个$X.element$时，就将其对应的$X.value$值加1。

矩估计

- 二阶矩估计的AMS算法
  - 实例：假定流为a,b,c,b,d,a,c,d,a,b,d,c,a,a,b,流的长度为n=15。由于a出现5次，b出现4次，c和d各出现3次，因此二阶矩应为$5^2 + 4^2 + 3^2 + 3^2 = 59$。假设维护3个变量$X_1$、$X_2$和$X_3$。另外，我们假设“随机”选出的3个变量分别为3、8和13，则可以基于这3个位置来定义各自对应的变量。
  - 当到达位置3时，对应的元素为c，因此$X_1.element = c$，此时$X_1.value = 1$。而位置4包含b，因此此时X_1的值不会变，位置5、6也不会变，位置7时，出现c，$X_1.value$ =2。

- 二阶矩估计的AMS算法
  - （接上页）位置8出现的是元素d，于是$X_2.element = d$，且$X_2.value = 1$。位置9和位置10分别出现a和b，因此X_1、X_2的值不会受到影响。位置11出现元素d，此时$X_2.value = 2$。位置12出又出现c，因此$X_1.value = 3$。位置13出现元素a，因此$X_3.element = a$，且$X_3.value = 1$。则在位置14处，$X_3.value = 2$，位置15处元素为b，不影响X_1、X_2和X_3。最终，$X_1.value = 3$，$X_2.value = X_3.value = 2$。

矩估计

- 二阶矩估计的AMS算法
  - 基于任意变量X，二阶矩的估计值为$n \times (2 \times X.value - 1)$。
  - 按上例，变量X_1得到的二阶矩的估计值为$15\times (2 \times 3 - 1) = 75$，变量$X_2$、$X_3$得到的二阶矩的估计值为$15\times (2 \times 2 - 1) = 45$。平均值为55，与正确结果59很接近。

矩估计

- AMS算法有效性原因
  - 证明上述方法构造出的任意变量的期望值都等于流的二阶矩。假设，$e(i)$表示流中第i个位置上的元素，而c(i)表示$e(i)$出现在位置$i, i+1, \ldots, n$上的次数。
  - 上例中：e(6)=a, c(6)=4, 位置1上的a对c(6)没有贡献
  - $X.value$的期望值为所有1到n上的位置i的$n \times (2 \times c(i) - 1)$值得平均值，即

$$E(X.value) = \frac{1}{n}\sum_{i=1}^{n} n \times (2 \times c(i) - 1)$$

  - 约去$1/n$ 和n，上式简化为

$$E(X.value) = \sum_{i=1}^{n} (2 \times c(i) - 1)$$

矩估计

- AMS算法有效性原因
  - 要理解上述公式，需要改变求和的次序。上式是对所有位置求和而并没有考虑不同元素，实际上可先考虑按元素分组求和，然后再对不同元素上的求和结果求和。
  - 例如，关注某个流中元素a的出现次数m_a，上式右边对应a出现的最后一次位置上的$2c(i)-1$的值为1，倒数第2次出现时为3，依次类推，直到第一次出现为$2m_a-1$。则

$$E(X.value)=\sum_{i=1}^{n}1+3+\cdots+(2m_a-1)\approx {m_a}^2$$

（和中项数推导）而这正是流的二阶矩定义

矩估计

- 更高阶矩的估计
  - 2阶矩估计公式

$(2 \times v - 1)$是v^2-$(v-1)^2$，所以：

$$n \times (2 \times v - 1) = n \times (v^2 - (v-1)^2)$$

  - 3阶矩估计公式

$v^3 - (v-1)^3 = (3v^2 - 3v + 1)$，所以

$$n \times (3v^2 - 3v + 1) = n \times (v^3 - (v-1)^3)$$

  - k阶矩估计公式

$$n \times (v^k - (v-1)^k)$$

矩估计

- 无限流的处理
  - 上面的矩估计都假定流的长度n是一个常数。
  - 实际应用中，n会随着时间推移不断增长。
  - 位置选择靠前，偏向于早期元素，位置选择靠后，早期元素太少，都会造成估计的可靠性不高。
  - 比较合理：在任何时候都尽可能保有足够多的变量，并在流增长时丢弃某些变量。而丢弃的变量会被新变量替代，但替代时必须要保证：
    - 在任何时候为变量选择位置时，选择某个任意位置的概率和选择任意其他位置的概率要相等。
  - 假设我们拥有存储s个变量，最早的s个位置中的每一个位置都会被选为s个变量中某一个变量的位置。

矩估计

- 无限流的处理
  - 假定流已有n个元素，任意某个特定位置都使某个变量（总共s个变量）位置的概率满足均匀分布，值为s/n；
  - 当第n+1个元素到达，选择上述位置的概率为$s/n+1$；如果该元素未被选上，那么s个变量仍然保持原来的位置，如果被选上，那么以等概率丢弃当前s个变量中的一个，并将该变量替换为一个新的变量，该变量元素的位置为n+1，值为1。

矩估计

- 当然：位置n+1被选中的概率为s/(n+1), 但是所有其他位置被选中的概率也是s/(n+1), 这个结论可以通过数学归纳法对n进行归纳而证明。此时，归纳假设是在n+1个流元素到达之前，选择一个位置的概率为s/n。
- 当第n+1个元素到达时，该元素位置没被选中的概率为1-s/(n+1), 这种情况下前面的n各位置上每个位置被选中的概率仍为s/n。但是第n+1个位置有s/(n+1)的概率被选中，一旦选中，那么前n个位置上每个位置被选中的概率减少为原来的(s-1)/s倍。综合考虑两种情况，选中前n个位置的每个位置的概率为：

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

- 可简化为：

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

- 进一步得到：

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

- 最后简化为：

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

- 因此：每个位置被选为变量位置的概率都是s/n。

矩估计

- 无限流的处理
  - 实际上解决了一个更一般的问题。它提供了一种从流中选出s个元素样本的方式，这种方式可以保证在任何时候，所有的流元素都有相等的概率被选为样本
  - 价值：在前述的流数据的抽样中选出键值属于某个随机选择子集的所有元组的例子，假定随着时间推移，和某个键关联的元组数目非常非常多。对于任一键K，当K的一个新元组到达时，我们可以采用本节的技术，将元组的数目限制为某个固定常数s。

窗口内的计数问题

- 问题描述
  - 假定有一个窗口大小为N的二进制流，我们希望在任何时候都能回答“对任意$k \leq N$，最近k位中有多少个1？”
  - 和前面一样，我们重点关注内存中无法容纳整个窗口的情况。
- 精确计数的开销
  - 假设：想要得到窗口大小为N的二进制流中最后任意k($k \leq N$)位中1的精确数目，那么可以断言必须要存储窗口中的所有N位，这是因为任何长度小于N位的表示方法都无法得到精确结果。

窗口内的计数问题

- 精确计数的开销
  - 证明：假设存在一个长度小于N的表示方法可以表示窗口中的所有N位。由于N位的不同组合序列有2^N个，但是此时表示的数目却小于2^N。于是必定存在两个不同的位串w和x，它们的表示却完全一样。由于$w \neq x$，所以至少必须有1位不相同。设w和x的最后k-1位相同，而倒数第k位却不相同。
  - 例子：如果$w = 0101$且$x = 1010$，则由于从右往左扫描的第一位就不用，因此$k = 1$。如果$w = 1001$且$x = 0101$，则从右往左扫描的第3位不同，因此此时$k = 3$。（接下页）

窗口内的计数问题

- 精确计数的开销
  - 假设表示窗口内容的数据中不论采用哪种位串方式来表示w和x，那么对于查询“最后k位当中有多少个1？”不管窗口是否包含w或x，查询应答算法都应该产生相同的答案，这是因为该算法只基于表示来进行。但是针对这两个位串的正确答案肯定不同。因此，我们前面给出的“存在长度小于N的表示方法”的假设不正确，也就是说必须至少使用N位来回答上述查询。

窗口内的计数问题

- 精确计数的开销
  - 实际上，即使仅仅回答查询“大小为N的整个窗口中包含多少个1？”，这个问题也需要N位。
  - 假设采用不到N位来表示窗口，于是可以采用上述方法来找到相应的w、x和k。如果在当前窗口后面加上任意的N-k位，则以w和x的右边k位为开始串的两个真实窗口的内容除了最左边一位之外，其他部分都相同。因此它们的1的数目并不相同。但是，由于w和x的表示是一样的，所以讲它们加载表示相同的窗口之后，得到的表示必须仍然相同。
  - 所以，对于上述查询，我们可以强迫出现在两种可能的窗口内容之一的情况下回答不正确。
  - 因此：上述查询也需要N位来表示窗口。

窗口内的计数问题

- DGIM（Datar-Gionis-Indyk-Motwani）算法简介
  - 该算法能够使用$O(log^2 N)$位来表示大小为N位的窗口，同时能保证窗口内1数目的估计错误率不高于50%。后面我们将介绍对该算法的改进，改进之后的算法能够在仍然只使用$O(log^2 N)$位的情况下，将估计的错误率降到任意大于0的小数ε之内（尽管当ε不断下降时，该复杂度会乘以某个不断增大的常数因子）。

窗口内的计数问题

- DGIM算法内容
  - 首先，流中每位都有一个时间戳(timestamp)，即它的到达位置。比如，第一位的时间戳为1，第二位为2，其余依次类推。由于只需要区分长度为N的窗口内的不同位置，因此可以将所有时间戳都对N取模，这样它们就可以通过$\log_2 N$位来表示。如果还存储了流中已看到的全部位数（即最近的时间戳）对N取模的结果，那么就可以基于该结果来确定其在窗口内的所在的时间戳。

窗口内的计数问题

- DGIM算法内容
  - 其次，我们将整个窗口划分成多个桶，每个桶中包含以下信息：
    - 桶最右部的时间戳（即最近的时间戳）；
    - 桶中1的数目，该数目必须是2的幂，我们将该数称为桶的大小。

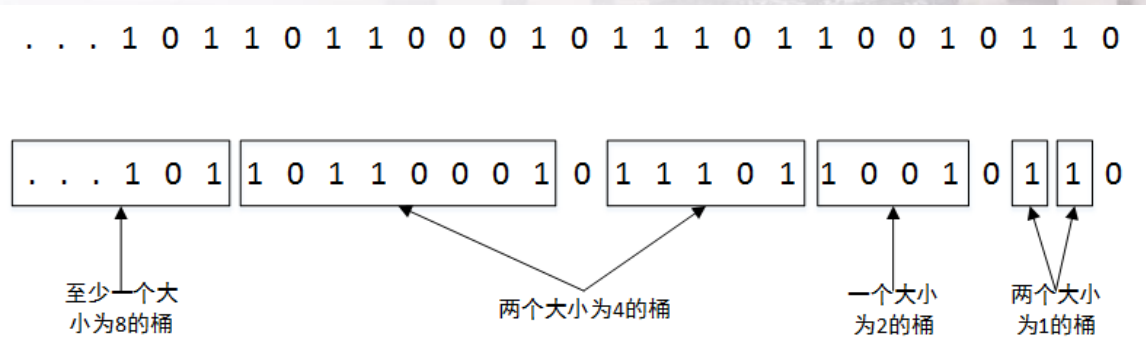
为了表示一个桶，需要$\log_2 N$位来表示其右部的时间戳（实际上是对N取模的结果）。为了表示桶中1的个数，只需要$\log_2 \log_2 N$位。其原因在于我们知道该数目i是2的幂，比如说2^j，因此只需要对j进行二进制编码来表示i即可。因为j最大为$\log_2 N$，所以对它进行表示需要$\log_2 \log_2 N$位。于是，采用$O(\log N)$位已经足够表示一个桶。

窗口内的计数问题

- DGIM算法内容
  - 当流采用上述桶表示方法是，必须遵循以下6条规则。
    - 桶最右部的位置上总是1；
    - 每个1的位置都在某个桶中；
    - 一个位置只能属于一个桶；
    - 桶的大小从最小一直变化到某个最大值，相同大小的桶只可能有一到两个；
    - 所有桶的大小必须都是2的幂；
    - 从右到左扫描（即从远到近扫描），桶的大小不会减小。

窗口内的计数问题

- DGIM算法内容
  - 下图给出了基于DGIM规则将二进制位流划分成多个桶的例子。我们看到最右部（最近）有两个大小1的桶，其左边有一个大小为2的桶。需要注意的是，这个桶覆盖了4个位置，但是其中1的个数为2,，继续往左扫描，可以见到两个大小为4的桶。再往左至少可以可到一个大小为8的桶。桶之间允许存在一些0。



窗口内的计数问题

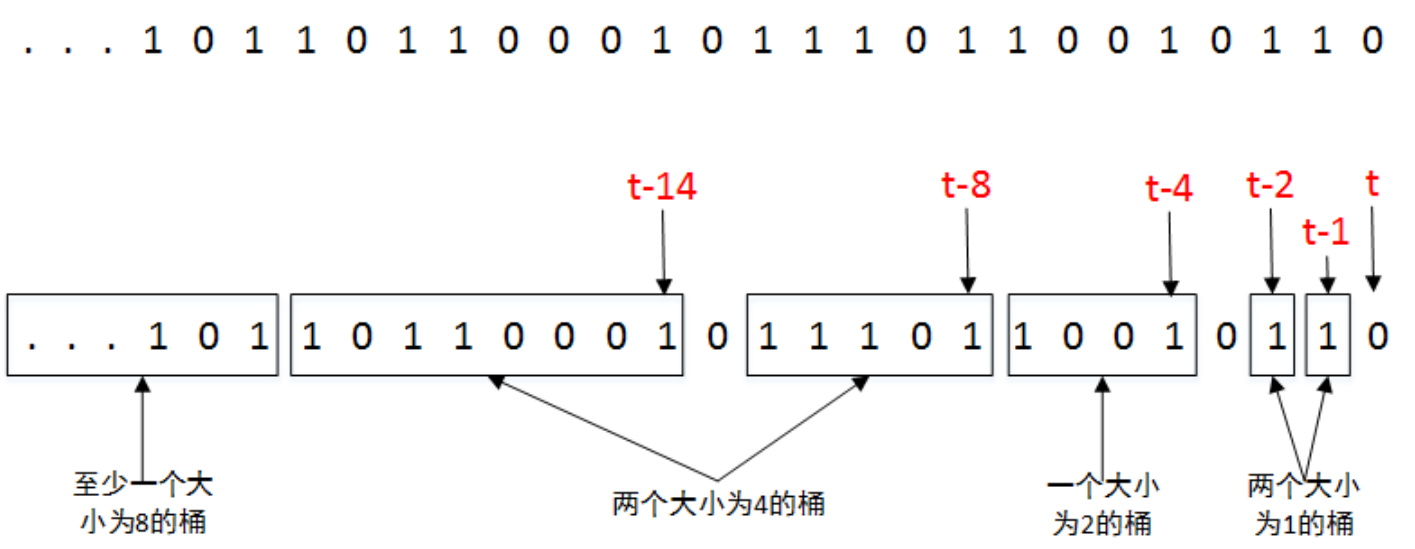
- DGIM算法的存储需求
  - 我们观察到每个桶可以采用$O(\log_2 N)$位来表示。如果窗口的长度为N，那么其中1的位数一定不会超过N。假定最大桶大小为2^j，那么j不可能超过$\log_2 N$，不然的话，该桶中1的个数会超过整个窗口内1的个数。因此，所有大小从$\log_2 N$到1的桶，都不会超过2个，而且不可能有更大的桶。
  - 桶的数目为k，$2^0 + 2^1 + \cdots + 2^k = 2^{k+1} - 1$，所以$O(\log_2 N)$。
  - 因此空间复杂度为: $O(\log_2 N)^2$。

窗口内的计数问题

- DGIM算法中的查询应答
  - 假定对于某个满足$1 \ll k \ll N$的整数k，需要回答的问题是“问题中最后k位中有多少个1？”。DGIM算法会寻找某个具有最低时间戳的桶b，它至少包含k个最近位中的一部分。最后的估计值为桶b右部（最近）所有桶的大小加上桶b的一半大小。
  - 例子：二进制流为1011011000101110110010110，k=10，利用DGIM算法进行估计？

窗口内的计数问题

- DGIM算法中的查询应答
  - 桶划分以及时间戳如下所示：



  - 最右边元素的时间戳为t，前四个桶的时间戳如上图所示。
  - 估计值为：$(1+1+2)+4/2=6$，真实值5

窗口内的计数问题

- DGIM算法中的查询应答
  - 估计错误率分析
    - 假定在上述查询应答的估计当中找到的桶b的大小为2^j，即它的一部分在查询的范围之内。接下来考虑估计值和真实值c之间的差异程度。这里存在两种情况，一种是估计值比c大，另一种是估计值小于c。
    - 情况1：估计值小于c。最坏情况下，桶b中的所有1实际上在查询范围内，这样估计值少算了桶b大小的一半，即2^{j-1}个1。但是这种情况下，估计值至少为2^j，c实际上至少为$2^{j+1}-1$，这是因为对于大小$2^{j-1}, 2^{j-2}, \ldots, 1$，都至少存在一个桶。因此得出结论，这种情况下估计值至少为c的50%。

窗口内的计数问题

- DGIM算法中的查询应答
  - 估计错误率分析
    - 情况2：估计值大于c。最坏情况下，桶b中只有最右边的一位在查询范围内，且对所有大小比桶b小的桶来说都仅只有一个桶。于是，$c = 1 + 2^{j-1} + 2^{j-2} + \cdots + 1 = 2^j$，而估计值为$2^{j-1} + 2^{j-1} + 2^{j-2} + \cdots + 1 = 2^j + 2^j - 1$。我们可以看到估计值最多比c大50%。

窗口内的计数问题

- DGIM条件的保持
  - 假定长度为N的窗口采用了合适的桶表示，该表示满足上面提到的DGIM条件。当新的位到达时，我们可能需要对桶进行修改，从而继续表示窗口并满足DGIM条件。
  - 首先，没当一个新的位到达时，检查最左边的（即最早的）桶，如果该桶的时间戳已经达到当前时间戳减去N，那么该桶所有1不再在窗口之内，因此将该桶丢弃。
  - 其次，考虑到来的新位是0还是1。如果是0，那么不需要任何修改，但是如果为1，就需

窗口内的计数问题

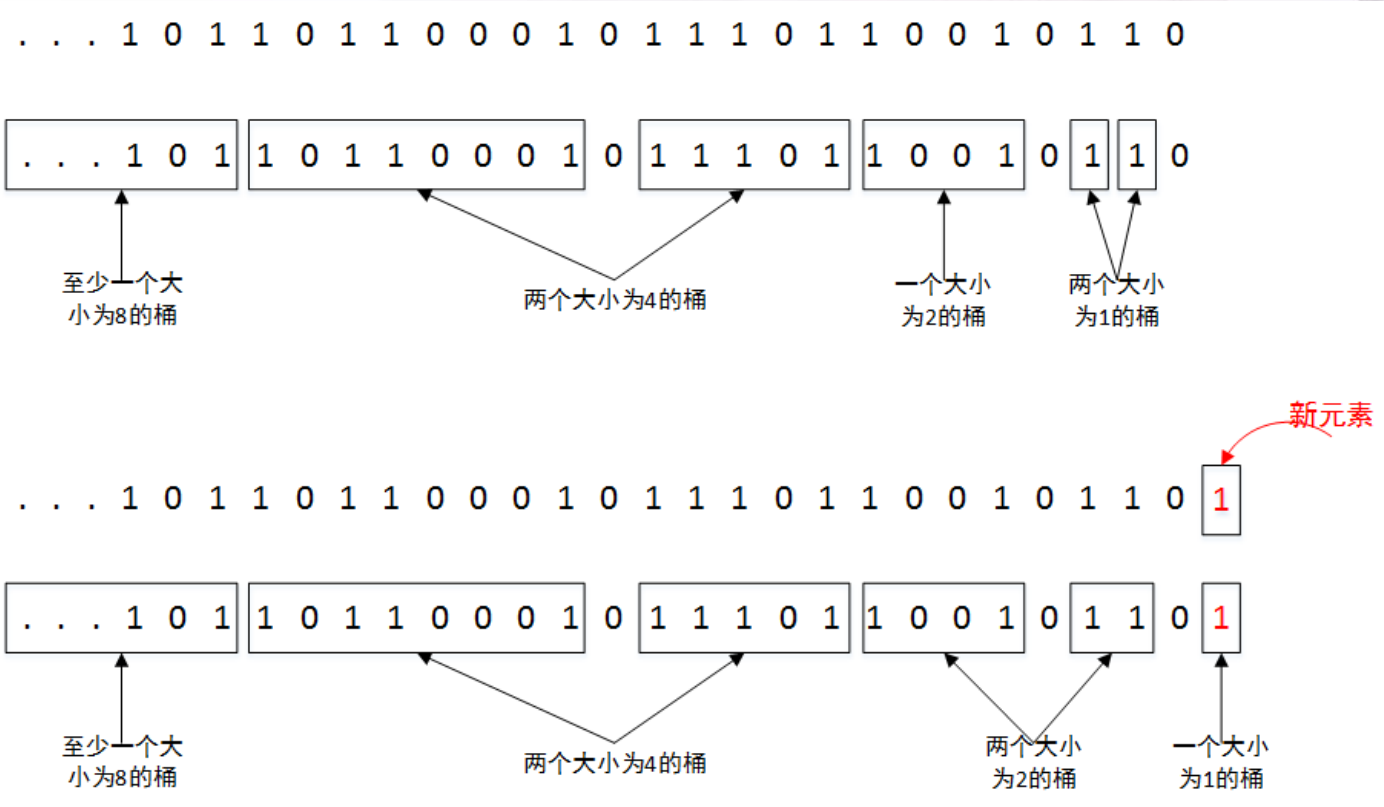
- DGIM条件的保持
  - （接上页）要做些改变。
  - 然后，基于当前时间戳建立一个新的大小为1的桶。如果仅有一个大小为1的桶，那么不需要做进一步的修改。但是，如果此时有3个大小为1的桶的话，其数目就多了1个。此时，可以通过将最左边（最早）的两个大小为1的桶进行合并来解决问题。
  - 最后，从右向左依次合并两个相同大小的连续桶，将它们替换为一个2倍大小的桶。新的桶的时间戳为被合并的最右边（时间上稍晚）那个桶的时间戳。

窗口内的计数问题

- DGIM条件的保持
  - 由于至多有$\log_2 N$个不同大小的桶，而上述两个相邻的具有相同大小的桶的合并过程只需要常数时间就可以完成。因此任意新位到达后的处理时间为$O(\log_2 N)$。

窗口内的计数问题

- DGIM条件的保持
  - 例子，当加入一个新的元素1时，桶的变化如下如所示。



窗口内的计数问题

- 降低错误率
  - 前面允许具有相同大小的桶的数目是1或者是2，与此不同，我们假定对于指数增长的桶大小1,2,4,…,具有相同大小的桶的数目是r-1或者r，其中r是一个大于2的整数。
  - 桶合并的规则本质上任然和上述的完全一样，如果大小为2^j的桶的数目为r+1，则将最左边的两个桶合并为一个大小为2^{j+1}的桶。合并之后可能导致大小为2^{j+1}的桶数目为r+1，如果这样的话，则继续将较大的桶合并。
  - 采用上面讲过的错误率推导过程，当最左边的桶b中仅有一个1在查询范围之内时，错误率相对最大，此时的查询结果被过高估计。假定桶b的大小为2^j,于是真实的查询结果值

窗口内的计数问题

- 降低错误率
  - （接上页）至少为$1+(r-1)\left(2^{j-1}+2^{j-2}+\right.$

窗口内的计数问题

- 窗口内计数问题的扩展
  - 一个很自然的问题是，DGIM算法是否可以推广？例如，对于整数流，是否可以估计流中大小为N的窗口内最近$k(1 \leq k \leq N)$个整数的和。
  - 当整数流既包含整数又包含负数时，不太可能利用DGIM算法来处理。假定有个流既包含大的正整数又包含绝对值很大的负整数，但是窗口内的所有整数之和非常接近于0。对于这些大整数而言，任意不太精确的估计结果都会严重影响整个求和的估计结果，因此错误率可能没有上限。

窗口内的计数问题

- 窗口内计数问题的扩展
  - 举例：假定和以往一样将上述整数流划分成多个桶，只不过这里的桶基于桶内整数和而不是1的位数来表示。如果只有一部分在查询范围之内是桶b，那么该桶的前半部分可能都是绝对值很大的负整数，而其后半部分都是绝对值很大的正整数，当然桶中的整数和接近于0。如果用b的一半估计，贡献值为0。但是桶b的实际贡献可能来自于其属于查询范围内的后半部分，具体的贡献值从0到所有的正整数之和都有可能。估计值和真实的差距很大，估计值没有意义。

窗口内的计数问题

- 举例（可以用DGIM算法）：假设整个流仅由1到2^m之间的正整数构成。此时，可以将每个整数的m位二进制看成一个单独的位流，然后使用DGIM方法来对处于所有位流上的每个位置（0,1,…,m-1）上的1的计数。假定所有位流上第i位上的1的个数为c_i（假定统计时按照整数的二进制从低位到高位的次序进行，初始值为0），那么所有整数的和为：

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

窗口内的计数问题

- 举例（可以用DGIM算法）：假设m=5，数据流到来的顺序为….2,1,4,8,16,2。那么统计过程如下。

	DGIM	DGIM	DGIM	DGIM	DGIM
.	.	.	.	.	.
.	.	.	.	.	.
.	.	.	.	.	.
2	0	0	0	1	0
1	0	0	0	0	1
4	0	0	1	0	0
8	0	1	0	0	0
16	1	0	0	0	0
2	0	0	0	1	0
	4	3	2	1	0

每个位置都运用DGIM算法估计出1的数目，分别为c_0, c_1, c_2, c_3, c_4，然后按照公式就可以算出估计值。

使用上面降低错误率的方法，即估计每个c_i的错误率不高于ε，那么最后真正求和的估计错误率也不高于ε。当所有c_i都被过高或者过低估计了相同的比率时，会出现最坏的情况。

衰减窗口

- 问题描述：前面我们都假设有一个滑动窗口容纳流的某个尾部元素，该窗口要么是由某个固定的N个元素组成，要么是由某个过去的时间点之后的元素组成。有时我们并不想将最近的元素和已过去一定时间的元素截然分开，而只是对最近的元素赋予更高的权重。
- 可以用衰减窗口的方法来处理。非常有用的应用：寻找近期的最常见元素。

衰减窗口

- 最常见元素问题
  - 假定有一个由全世界出售的电影票所构成的数据流，每个元素当中还包括电影的名称。我们希望能够从该数据流中总结出“当前”最流行的电影。虽然“当前”的概念有些含糊，但是从直观来说，我们希望降低诸如《星球大战4：新希望》之类电影的流行度，这是因为这些电影虽然卖出了很多票，但是大部分票都出售于几十年前。另一方面，一部近10周年每周都售出n张票的电影会比仅上周售出2n张票但是再往前一张票都没售出的电影的流行度要高。

衰减窗口

- 最常见元素问题
  - 一种解决方法是将每部电影想象成一个位流，如果当前的票属于这部电影，则该位置1，否则置0。选择整数N作为计算流行度时需要考虑的最近的电影票的数目。可以利用DGIM算法估计每部电影近期票房并按估计结果对电影排序。
  - 该技术对电影有效，因为电影的数目仅在几千范围内。当数目很大时，就会失败。并且这种方法只能产生近似的估计结果。

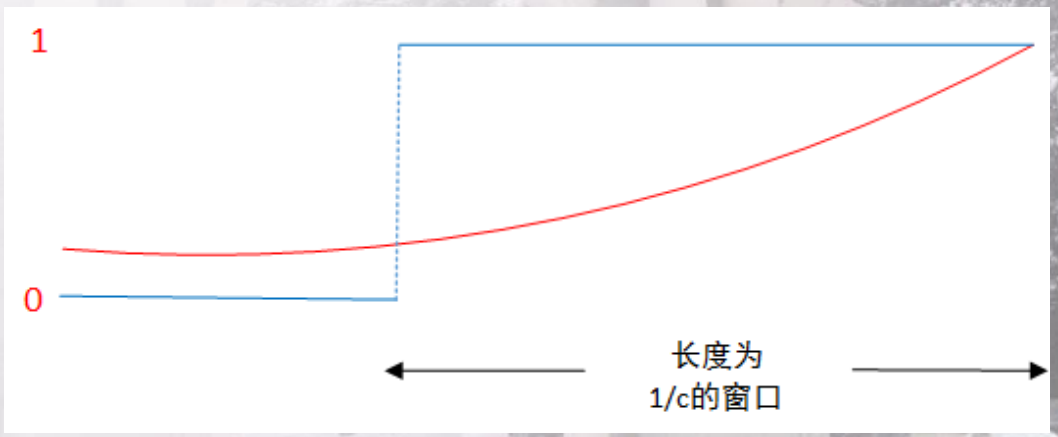
衰减窗口

- 衰减窗口的定义
  - 另一种方法是对问题进行重新定义，使得不再查询窗口内的1的数目。更确切地说，我们对流中已见的所有1计算一个平滑的累积值，其中采用的权值不断衰减，因此，元素在流中出现越早，其权值也越小。形式化地，令流当前的元素为$a_1, a_2, \ldots, a_t$,其中a_1是第一到达的元素，a_t是当前元素。令c为一个很小的常数，比如10^{-6}或10^{-9}。那么该流的指数衰减窗口定义为

$$\sum_{i=0}^{t-1} a_{t-i}(1-c)^i$$

衰减窗口

- 衰减窗口的定义
  - 上述定义的效果是，流中元素的权重取决于其离当前元素时间的远近，流中越早元素的权重越小。与此形成鲜明对照的是，对一个大小为1/c的固定大小的窗口内的元素进行加权求和时，会对最近1/c个元素都赋予权重1，而对所有更早的元素赋予权重0。这两种方法的区别如图



衰减窗口

- 衰减窗口的定义
  - 相对于定长滑动窗口来说，对指数衰减窗口中的求和结果进行调整要容易的多。对于滑动窗口而言，在每次新元素到达时必须要考虑它是否在窗口外。也就是说，我们必须要在求和结果之外保留元素的数目，或者使用诸如DGIM之类的近似模式。但是，在指数衰减窗口中，当新元素a_{t+1}到达时，所有需要做的仅仅是：
    - 将当前结果乘以1-c
    - 加上a_{t+1}。

窗口衰减

- 衰减窗口的定义
  - （接上页）这种处理方法有效的原因在于，当新元素到来时，原有元素相对于当前元素而言又远离了一个位置，因此其权重必须乘上1-c。另外，当前元素的权重为$(1-c)^0=1$，因此，直接加上a_{t+1}可以正确体现新元素的贡献。

窗口衰减

- 最流行元素的发现
  - 利用指数衰减窗口来解决电影票销售数据流中的最热门电影的发现问题。
  - 假设：c为10^{-9}，也就是说拥有能容纳最近10亿次购票记录的滑动窗口。对每部电影都想象有一个独立的位流来表示其购票记录，如果前面的电影票数据流中的位置对应当前电影，则该电影位流上相应位置置1，否则置0。该窗口中的所有1的衰减求和结果度量了电影的热门程度。

窗口衰减

- 最流行元素的发现—步骤
  - 当有新的电影票信息到达流，进行如下操作
    - 对当前保留得分的每部电影，将其得分乘上(1-c)
    - 假定新电影票对应的电影是M。如果当前M的得分存在的话，将M的得分加上1。如果M的得分不存在，那么为M建立一个初始为1的得分。
    - 任意一个得分如果小于阀值K（此例子中定义为1/2）的话，那么它们会被丢弃。这一点是为了保证任意时间保留得分的电影的数目是有限的。

衰减窗口

- 最流行元素的发现—阈值分析
  - 对于K=1/2分析。注意到所有得分的和为$(1-c)^0+(1-c)^1+\cdots+(1-c)^i \approx 1/c$。得分为1/2或者更高的电影数目不可能多于2/c，否则所有的得分之和会大于1/c。因此，任意时间进行计数的电影的数目的上限是2/c。当然，实际上任何电影票销售记录只会关注一小部分电影，因此真正计算的电影数目会远远小于2/c。

小节

- 流数据模型：该模型假定数据以某种速率到达处理引擎，该速率使得无法在当前可用内存中存放所有数据。流处理的一种策略是保留流的某个概要信息，使之足够回答关于数据的期望的查询。另一种方法是维持最近到达数据的一个滑动窗口。
- 流抽样：为创建能为某类查询所用的流样本，我们确定流中的关键属性集合。对任一到达流元素键值进行哈希处理，使用哈希值来确定包含该键值的全部元素（或者没有那个元素）会是抽样样本的一部分。

小节

- 布隆过滤器：该技术能够允许属于某个特定集合的流元素通过，而大部分其他元素被丢弃。我们使用一个大的位数组合多个哈希函数。给定集合上的元素哈希到桶（即整数中的每一位）中，这些位置上都会置1。为了检查某个流元素是否属于给定集合，我们使用每个哈希函数对它进行处理，只有所有哈希结果对应的位置为1时才能接受该元素。
- 独立元素技术：为了估计流中出现的不同元素的数目，可以将元素哈希成整数，这些整数可以解释为二进制整数。任意流元素的哈希值中最长的0序列长度作为2的幂得到的结果会作为独立元素数目的估计值。可以通过使用多个哈希函数并组合估计结果，首先组内取平均值，然后组间取中位数，最终可以得到可靠的估计结果。

小节

- 流的矩：流的k阶矩是流中至少出现一次的元素的出现次数的k次方之和。0阶矩是独立元素的个数，1阶矩是流的长度。
- 二阶矩估计：二阶矩或者说奇异数的估计有一个较好的方法。首先从流中随机选择一个位置，然后计算该位置开始往后的元素出现次数，最后将该数目乘2减1，并与流长度相乘。对于采用上述方法随机选出的多个变量，也可以采用独立元素估计中所用的组合方法来得到可靠的估计结果。
- 更高阶矩的估计：二阶矩的估计方法可以直接用于k阶矩的估计，唯一不同的是要将公式2x-1（其中x是从选择位置开始往后的元素出现的次数）替换为 $x^k-(x-1)^k$

小节

- 窗口内1的数目估计：可以将0/1二进制流窗口中的1分到多个桶中，从而估计出1的数目。其中，每个桶中1的数目是2的幂，相同大小的桶数目为1或2，且从右到左的桶的大小非减。如果只记录桶的位置和大小信息，就可以在消耗$O(\log_2^2 N)$空间的情况下表示大小为N的窗口内容。
- 有关1的数目的查询应答：如果想知道二进制流的最近k个元素中1的大概数目，可以寻找一个最早的桶B，它至少包含了查询范围的一部分（即包含最近的k个元素中的一部分），则最终的估计结果为B的一半大小加上所有后来的桶的大小。该估计值永远不会超出1的真实数目的50%。
- 1的数目的更精确估计：通过修改相同大小的桶数目所满足的限制条件，就可以更精确地估计1的数目，假定运行相同大小的桶的数目为r或者r-1，那么就可以保证估计值不会超出真实值的1/r。

小节

- 指数衰减窗口：不同于采用固定窗口大小，可以将窗口想象为所有到达的元素，但是对于t个时间单位之前到来的元素赋予的权重是 e^{-ct} （其中c是一个常数）。这样做十分容易就可以保留一个指数衰减窗口的概要。例如，当一个新元素到达时，只需要将当前的求和值乘以1-c再加上当前元素的值即可。
- 指数衰减窗口下的高频元素获取：可以将每个项都想象为由一个二进制位流表示的，其中0表示当前项不是给定时间到达的元素，1表示当前项是给定时间到达的元素。可以找出那些二进制流的和不低于1/2的元素。当心元素到达时，将当前积累的得分和乘以1-c后加上1，并删除所有和小于1/2的项。

参考文献

- 流管理相关
  - H.V. Jagadish, I.S. Mumick, and A. Silberschatz, “View maintenance issues for the chronicle data model,” Proc. ACM Symp. on Principles of Database Systems, pp. 113-124, 1995
  - B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom, “Models and issues in data stream systems,” Symposium on Principles of Database Systems, pp. 1-16 2002
  - M. Garofalakis, J. Gehrke, and R. Rastogi (editors), Data Stream Management, Springer, 2009.

参考文献

- 抽样技术
  - P.B. Gibbons, “Distinct sampling for highly-accurate answers to distinct values queries and event reports,” Intl. Conf. on Very Large Databases, pp. 541-550, 2001
- 布隆过滤器
  - B.H. Bloom, “Space/time trade-offs in hash coding with allowable errors,” Comm. ACM 13:7, pp. 422-426, 1970.
  - W.H. Kautz and R.C. Singleton, “Nonadaptive binary superimposed codes,” IEEE Transactions on Information Theory 10, pp. 363-377, 1964.

参考文献

- 独立元素计数算法
  - N. Alon, Y.Matias, and M. Szegedy, “ The space complexity of approximating frequency moments,” 28th ACM Symposium on Theory of Computiong, pp. 20-29, 1996.
  - P. Flajolet and G.N. Matrin, “Probabilistic counting for database applications,” 24th Symposium on Foundations of Computer Science, pp. 76-82, 1983
- 窗口内1的数目近似求解算法
  - M. Datar, A. Gionis, P. Indyk, and R. Motwani, “Maintaining stream statistics over sliding windows,” SIAM J. Computing 31, pp. 1794-1813, 2002.