大数据问题的算法

//** 笔记基本根据上课的顺序完成，但是对一些问题做了注释和补充 **//

本部分讨论的是输入数据过大而无法完全存入随机存取内存的大数据问题。针对这个问题的一个模型是流模型（streaming），其中 n 个数据 $a_1, a_2, \cdots, a_n$ 依次到达。例如，a_i 可能是互联网路由器观察到的 IP 地址，目标是在不使用过多内存的情况下计算这些数据项的某些统计信息、属性或者汇总（远小于 n）。更具体地说，我们假设每个 a_i 是一个 b 位的量，这里 b 并不很大，例如，每个 a_i 可能是一个整数，满足 $1 \le a_i \le m$，其中 $m = 2^b$。目标是使用多项式复杂度的 b 和 $\log n$ 空间来产生所需的输出。

一个在流模型中很容易解决的问题是计算所有的 a_i 的总和。如果每个 a_i 是介于 1 和 $m = 2^b$ 之间的整数，那么所有的 a_i 的总和也是一个介于 1 和 mn 间的整数，因此维持总和所需的内存位数为 $O(b + \log n)$。一个更复杂的问题是统计输入的不同数字的数量。

处理流模型中一系列问题的一个自然方法是对输入进行即时的随机抽样。为了介绍即时抽样的基本概念，考虑从流 $a_1, a_2, \ldots, a_n$ 中选择一个索引 i，其概率与 a_i 的值成比例。当我们看到一个元素时，由于归一化常数依赖于所有元素（包括我们尚未看到的那些），所以我们不知道以何种概率选择它。然而，以下方法是可行的：

设 s 为目前为止看到的 a_i 的总和。保持 s 和一个以 $\frac{a_i}{s}$ 的概率选中的索引 i。初始时 $i = 1$ 且 $s = a_1$。在看到符号 $a_1, a_2, \ldots, a_j$ 后，s 将等于 $a_1 + a_2 + \cdots + a_j$，对于 $i \in \{1, \ldots, j\}$，选中的索引将是 i，其概率为 $\frac{a_i}{s}$。在看到 a_{j+1} 时，以 $\frac{a_{j+1}}{s+a_{j+1}}$ 的概率改变选中的索引为 $j+1$，否则以 $1 - \frac{a_{j+1}}{s+a_{j+1}}$ 的概率保留之前的索引。如果我们改变了索引到 $j+1$，显然它是以正确的概率被选中的。如果我们保留 i 作为我们的选择，那么它将以如下概率被选中：

$$\left(1 - \frac{a_{j+1}}{s + a_{j+1}}\right) \cdot \frac{a_i}{s} = \frac{s}{s + a_{j+1}} \cdot \frac{a_i}{s} = \frac{a_i}{s + a_{j+1}},$$

这是选择索引 i 的正确概率。最后，通过加上 a_{j+1} 来更新 s。这个问题出现在许多领域，如睡眠专家问题，在那里有一系列权重，我们希望根据其权重的比例挑选一个专家。这里的 a_i 就是权重，下标 i 表示专家。

数据流的频率矩

一类重要的问题涉及到数据流的频率矩。长度为 n 的数据流 $a_1, a_2, \ldots, a_n$ 由来自一个包含 m 个可能符号的字母表中的符号 a_i 组成，为了方便起见，我们将这些符号表示为 $\{1, 2, \ldots, m\}$。在本节中，n，m 和 a_i 将具有上述含义，而 s（代表符号）将表示 $\{1, 2, \ldots, m\}$ 的任意元素。符号 s 的频率 f_s 是指 s 在流中出现的次数。

对于非负整数 p，流的第 p 个频率矩定义为：

$$F_p = \sum_{s=1}^{m} f_s^p. \tag{1}$$

请注意，当 $p = 0$ 时，频率矩对应于流中不同符号的数量，这里我们采用约定 $0^0 = 0$。第一个频率矩即为 n，也就是字符串的长度。第二个频率矩 $F_2 = \sum_{s=1}^{m} f_s^2$ 对于计算流的方差是有用的，即平均频率

的平方差异：

$$\text{Variance} = \frac{1}{m}\sum_{s=1}^{m}(f_s - \frac{n}{m})^2 \tag{2}$$

$$= \frac{1}{m}\sum_{s=1}^{m}\left(f_s^2 - 2\frac{n}{m}f_s + \left(\frac{n}{m}\right)^2\right) \tag{3}$$

$$= \frac{1}{m}\sum_{s=1}^{m} f_s^2 - \frac{n^2}{m^2}. \tag{4}$$

随着 p 变得越来越大，$(\sum_{s=1}^{m} f_s^p)^{1/p}$ 趋向于最频繁元素的频率。

我们将很快描述基于抽样的算法来计算这些量以处理流式数据。首先，关于这些问题的动机的一些说明。最频繁项的大小和频率，或者更普遍地说，那些频率超过给定 n 比例的项，在许多应用中显然是重要的。如果这些项是网络上的数据包，带有源地址和/或目的地址，则高频项可以识别出占用大量带宽的用户。如果数据是由超市中的购买记录组成，那么高频项就是最畅销的商品。确定不同符号的数量是抽象地确定诸如账户数量、网页用户数量或信用卡持有者数量等问题。第二矩和方差在网络、数据库以及其他应用中也是有用的。路由器会生成大量的网络日志数据，这些数据可以记录所有通过它们的消息的源地址、目的地址以及包的数量。这种大规模数据无法轻易地按每个源-目的地址进行排序或汇总。但是，了解某些流行的源-目的地址对是否有大量流量是很重要的，其中方差是一个自然的度量标准。

考虑一个由 n 个元素组成的序列 $a_1, a_2, \ldots, a_n$，每个 a_i 是范围从 1 到 m 的整数，其中 n 和 m 都非常大。假设我们想要确定序列中不同 a_i 的数量。每个 a_i 可能代表从一系列信用卡交易中提取出的信用卡号，而我们希望确定有多少个不同的信用卡账户。请注意，这可以通过使用 $O(m)$ 空间来轻松完成，方法是仅存储一个位向量（bit-vector），记录哪些元素已经被看到以及哪些尚未被看到。同样，也可以通过存储所有已见过的不同元素列表以 $O(n \log m)$ 的空间复杂度来实现。然而，我们的目标是以 m 和 n 的对数级空间复杂度进行计算。首先，我们将证明使用精确的确定性算法是不可能的。任何能够准确确定不同元素数量的确定性算法在某些长度为 $O(m)$ 的输入序列上必须使用至少 m 位的内存。然后，我们将证明如何通过随机化和近似的方法解决这个问题。

确定性算法的内存下界 我们证明任何精确的确定性算法在某些长度为 $m+1$ 的序列上必须使用至少 m 位的内存。假设我们已经看到了 $a_1, \ldots, a_m$，并且对于所有这样的序列，我们的算法使用的内存少于 m 位。存在 $2^m - 1$ 个可能的 $\{1, 2, \ldots, m\}$ 的子集，该序列可能包含这些子集中的任意一个，但我们的算法的内存状态只有 2^{m-1} 种可能性。因此，必然存在两个不同的子集 S_1 和 S_2 导致相同的内存状态。如果 S_1 和 S_2 的大小不同，那么显然这意味着对于其中一个输入序列会有错误。另一方面，如果它们的大小相同，且下一个元素属于 S_1 但不属于 S_2，算法在这两种情况下会给出相同的答案，因此至少在一个序列上的答案必然是不正确的。

求不同元素数量的算法 为了打破上述下界，我们可以考虑估算不同元素的数量。我们的算法将使用随机化来产生一个数字，这个数字是在正确答案的一个常数因子范围内，并且有很小的失败概率。假设集合 S 中的不同元素是从 $\{1, \ldots, m\}$ 中均匀随机选择的。令 min 表示 S 中的最小元素。min 的期望值是多少？如果只有一个不同元素，那么它的期望值大约是 $m/2$。如果有两个不同元素，它们的期望

值大约是 $m/3$。更普遍地，对于一个随机集合 S，最小值的期望值大约是 $m/(|S|+1)$。解方程

$$\min = \frac{m}{|S|+1}$$

得到

$$|S| = \frac{m}{\min} - 1.$$

这表明可以在 $O(\log m)$ 空间内记录最小元素，并使用此公式来估计 $|S|$。

哈希函数

通常，集合 S 可能不是均匀随机选择的。如果 S 的元素是通过选择 $\{1, 2, \ldots, m\}$ 中的 $|S|$ 个最小元素获得的，上述技术将给出非常糟糕的答案。然而，我们可以通过哈希函数将我们的直觉转化为一种算法，该算法对于每一个序列都有很高的概率工作良好。具体来说，我们将使用一个哈希函数 h，其中

$$h : \{1, 2, \ldots, m\} \to \{0, 1, 2, \ldots, M-1\},$$

然后不是记录 S 中的最小元素 a_i，而是记录最小的哈希值。现在的问题是：我们需要哈希函数具备什么属性？由于我们需要存储 h，我们不能使用完全随机的映射，因为那需要太多的位。幸运的是，一个可以紧凑存储的成对独立哈希函数就足够了。

成对独立哈希函数 (2-universal pairwise independent hash functions) 不同的应用程序对随机性的需求量各不相同。对于一个向量 $x = (x_1, x_2, \ldots, x_d)$，其中每个 $x_i \in \{0, 1\}$，如果需要完全的随机性，那么应该从所有 2^d 个二进制向量中均匀随机选择 x。然而，如果我们只需要每个 x_i 等可能地是 0 或 1，我们可以从两个向量 $\{(0, 0, ..., 0), (1, 1, ..., 1)\}$ 中选择 x。此外，如果还需要每一对坐标 x_i 和 x_j 在统计上独立，我们需要从一个更大的集合中选择，该集合具有这样的性质：对于每一对 i 和 j，(x_i, x_j) 等可能地是 $(0, 0), (0, 1), (1, 0)$ 或 $(1, 1)$。

一个哈希函数族

$$H = \{h \mid h : \{1, 2, \ldots, m\} \to \{0, 1, 2, \ldots, M-1\}\}$$

被称为 2-universal 或成对独立的，如果对于所有的 x 和 y 在 $\{1, 2, \ldots, m\}$ 中且 $x \neq y$，$h(x)$ 和 $h(y)$ 各自等可能地是 $\{0, 1, 2, \ldots, M-1\}$ 的任意元素，并且它们在统计上是独立的。这意味着一个哈希函数族 H 是 2-通用的当且仅当对于所有的 x 和 y 在 $\{1, 2, \ldots, m\}$ 中，$x \neq y$，$h(x)$ 和 $h(y)$ 各自等可能地是 $\{0, 1, 2, \ldots, M-1\}$ 的任意元素，并且对于所有的 w 和 z，$\text{Prob}_{h \sim H}[h(x) = w \text{ and } h(y) = z] = \frac{1}{M^2}$.

现在我们给出一个 2-universal 哈希函数族的例子。设 M 是一个大于 m 的素数。对于每一个整数对 a 和 b 在范围 $[0, M-1]$ 内，定义一个哈希函数 $h_{ab}(x) = ax + b \pmod M$. 为了存储哈希函数 h_{ab}，只需存储两个整数 a 和 b。这只需要 $O(\log M)$ 的空间。要证明这个函数族是 2-universal 的，注意 $h(x) = w$ 和 $h(y) = z$ 当且仅当

$$\begin{pmatrix} x & 1 \\ y & 1 \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} w \\ z \end{pmatrix} \mod M.$$

如果 $x \neq y$，矩阵 $\begin{pmatrix} x & 1 \\ y & 1 \end{pmatrix}$ 在模 M 下是可逆的。因此，$\begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} x & 1 \\ y & 1 \end{pmatrix}^{-1} \begin{pmatrix} w \\ z \end{pmatrix} \mod M$, 并且对于每一个 $\begin{pmatrix} w \\ z \end{pmatrix}$ 存在一个唯一的 $\begin{pmatrix} a \\ b \end{pmatrix}$。因此，

$$\text{Prob}[h(x) = w \text{ and } h(y) = z] = \frac{1}{M^2},$$

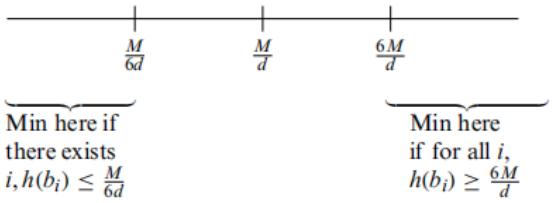

Figure 6.3: Location of the minimum in the distinct counting algorithm.

图 1: 6.3

所以 H 是 2-通用的。

不同元素计数算法的分析

设 $b_1, b_2, \ldots, b_d$ 是输入中出现的不同值。从 2-universal 哈希函数族 H 中选择 h。然后集合 $S = \{h(b_1), h(b_2), \ldots, h(b_d)\}$ 是来自集合 $\{0, 1, 2, \ldots, M-1\}$ 的 d 个随机且成对独立的值。我们现在证明 $\frac{M}{\min}$ 是输入中不同元素数量 d 的一个良好估计，其中 $\min = \min(S)$。

以至少 $\frac{2}{3} - \frac{d}{M}$ 的概率，不同元素数量的估计满足 $\frac{d}{6} \leq \frac{M}{\min} \leq 6d$，其中 min 是 S 中的最小元素。也就是说，$\frac{M}{6d} \leq \min \leq \frac{6M}{d}$.

证明. 我们要证明的两个不等式如图 6.3 所示。首先，我们证明 $\text{Prob}[\min \leq \frac{M}{6d}] \leq \frac{1}{6} + \frac{d}{M}$。这部分不需要成对独立性。$\text{Prob}[\min \leq \frac{M}{6d}] = \text{Prob}[\exists k, h(b_k) \leq \frac{M}{6d}] \leq \sum_{i=1}^{d} \text{Prob}[h(b_i) \leq \frac{M}{6d}] \leq d \cdot \frac{\lfloor \frac{M}{6d} \rfloor + 1}{M}$ $\leq d \cdot \left(\frac{1}{6d} + \frac{1}{M}\right) \leq \frac{1}{6} + \frac{d}{M}$.

接下来，我们证明 $\text{Prob}[\min \geq \frac{6d}{M}] \leq \frac{1}{6}$。这部分将使用成对独立性。首先，$\text{Prob}[\min \geq \frac{6d}{M}] = \text{Prob}[\forall k, h(b_k) \geq \frac{6d}{M}]$. 对于 $i = 1, 2, \ldots, d$，定义指示变量 $y_i = \begin{cases} 0 & \text{if } h(b_i) \geq \frac{6M}{d} \\ 1 & \text{otherwise} \end{cases}$ 并令 $y = \sum_{i=1}^{d} y_i$. 我们希望证明 $\text{Prob}(y = 0)$ 很小。现在 $\text{Prob}(y_i = 1) \geq \frac{6}{d}$，$\text{E}(y_i) \geq \frac{6}{d}$，和 $\text{E}(y) \geq 6$。对于成对独立的随机变量，它们和的方差是它们方差的和。因此 $\text{Var}(y) = d \cdot \text{Var}(y_1)$。进一步，由于 y_1 是 0 或 1，$\text{Var}(y_1) = \text{E}[(y_1 - \text{E}(y_1))^2] = \text{E}(y_1^2) - \text{E}^2(y_1) = \text{E}(y_1) - \text{E}^2(y_1) \leq \text{E}(y_1)$。因此 $\text{Var}(y) \leq \text{E}(y)$。根据切比雪夫不等式，$\text{Prob}[\min \geq \frac{6M}{d}] = \text{Prob}[\forall k, h(b_k) \geq \frac{6M}{d}] = \text{Prob}(y = 0) \leq \text{Prob}(|y - \text{E}(y)| \geq \text{E}(y))$ $\leq \frac{\text{Var}(y)}{\text{E}^2(y)} \leq \frac{1}{\text{E}(y)} \leq \frac{1}{6}$.

因为 $\frac{M}{\min} \geq 6d$ 的概率最多为 $\frac{1}{6} + \frac{d}{M}$，而 $\frac{M}{\min} \leq \frac{d}{6}$ 的概率最多为 $\frac{1}{6}$，所以 $\frac{d}{6} \leq \frac{M}{\min} \leq 6d$ 的概率至少为 $\frac{2}{3} - \frac{d}{M}$。 □

要计算流中给定元素出现的次数最多需要 $O(\log n)$ 的空间，其中 n 是流的长度。显然，对于任何在实际应用中出现的流长度，我们可以承受 $O(\log n)$ 的空间。因此，以下材料可能在实践中永远不会使用，但该技术是有趣的，并且可能会为如何解决其他问题提供洞察。

考虑一个由 0 和 1 组成的长度为 n 的字符串，我们希望计算 1 出现的次数。显然，使用 $O(\log n)$ 位内存可以跟踪 1 的确切数量。然而，这个数字可以用 $O(\log \log n)$ 位来近似。

设 m 是序列中 1 出现的次数。保持一个值 k 使得 2^k 大约等于出现次数 m。存储 k 只需要 $O(\log\log n)$ 位内存。算法如下：

- 初始化 $k=0$。
- 对于每次 1 的出现，以概率 $\frac{1}{2^k}$ 将 1 加到 k 上。
- 在字符串结束时，2^{k-1} 是 m 的估计值。

为了获得一枚以概率 $\frac{1}{2^k}$ 正面朝上的硬币，可以抛掷一枚公平的硬币（正面朝上的概率为 $\frac{1}{2}$）共 k 次，并报告如果这 k 次抛掷全部正面朝上，则认为结果为正面。

给定 k，平均来说，在 k 增加之前需要 2^k 个 1。因此，产生当前 k 值的 1 的期望数量是 $1+2+4+\cdots+2^{k-1}=2^k-1$。

首先考虑一个非常简单的问题：n 个人投票，有 m 个候选人 $\{1,2,\ldots,m\}$。我们想要确定是否有一个候选人获得了多数票，如果有，是谁。形式上，我们得到一个整数流 $a_1,a_2,\ldots,a_n$，每个 a_i 属于 $\{1,2,\ldots,m\}$，并想要确定是否存在某个 $s\in\{1,2,\ldots,m\}$ 出现超过 $\frac{n}{2}$ 次，如果是，那么是哪个 s。很容易看出，要在一次读取流数据的情况下用确定性算法精确地解决问题需要 $O(\min(n,m))$ 的空间。假设 n 是偶数，最后 $\frac{n}{2}$ 项是相同的。还假设在读取前 $\frac{n}{2}$ 项之后，有两个不同的元素集合导致我们的内存内容相同。在这种情况下，如果后一半的流完全由一个只在一个集合中而不在另一个集合中的元素组成，就会发生错误。如果 $\frac{n}{2}\geq m$，那么前 $\frac{n}{2}$ 个元素至少有 2^m-1 个可能的子集。如果 $\frac{n}{2}\leq m$，则有 $\binom{m}{\frac{n}{2}}$ 个子集。根据上述论点，内存的位数必须至少是子集数量的二进制对数，即 $O(\min(m,n))$。

令人惊讶的是，我们可以通过稍微削弱我们的目标来绕过上述下界。再次要求如果某些元素出现超过 $\frac{n}{2}$ 次，我们必须输出它。但现在，如果我们说如果没有元素出现超过 $\frac{n}{2}$ 次，我们的算法可以输出任何它想输出的东西，而不是要求它输出“没有”。也就是说，可能会有“假阳性”，但不会有“假阴性”。

多数元素算法

- 存储 a_1 并初始化计数器为 1。
- 对于后续的每个 a_i，如果 a_i 与当前存储的项相同，则将计数器加 1。如果不同，则在计数器非零的情况下将计数器减 1。如果计数器为零，则存储 a_i 并将计数器设置为 1。

分析此算法时，将减计数器步骤视为“消除”两个项目，一个新的项目和导致上次计数器增加的项目是方便的。很容易看到，如果存在多数元素 s，它必须在最后被存储。如果不是，那么每次 s 出现都被消除了；但是每次这样的消除也会导致另一个项目被消除。因此，如果多数项目在最后不被存储，那么必须有超过 n 个项目被消除，这是一个矛盾。

接下来，我们修改上述算法，不仅检测多数元素，还检测频率高于某个阈值的项目。更具体地说，下面的算法在加性误差 $O\left(\frac{n}{k+1}\right)$ 内找到每个元素 $\{1,2,\ldots,m\}$ 的频率（出现次数）。也就是说，对于每个符号 s，算法生成一个值 $\tilde{f}_s$ 位于 $\left[f_s-\frac{n}{k+1},f_s\right]$ 内，其中 f_s 是符号 s 在序列中真正出现的次数。它将通过保持 k 个计数器而不是仅一个计数器来做到这一点，使用 $O(k\log n+k\log m)$ 的空间。

频繁元素算法

- 维护一个正在计数的项目列表。初始时列表为空。
- 对于每个项目，如果它与列表上的某个项目相同，则将其计数器加 1。
- 如果它与列表上的所有项目都不同，则如果列表上的项目少于 k 个，添加该项目到列表并将计数器设置为 1。
- 如果列表上已经有 k 个项目，则将当前所有计数器减 1。如果某个元素的计数变为 0，则从列表中删除该元素。

定理：在频繁元素算法结束时，对于每个 $s \in \{1,2,\ldots,m\}$，如果它在列表上，则其计数器 $\tilde{f}_s$ 满足 $\tilde{f}_s \in \left[f_s - \frac{n}{k+1}, f_s\right]$。如果某个 s 没有出现在列表上，其计数器为 0，定理断言 $f_s \leq \frac{n}{k+1}$。

证明：$\tilde{f}_s \leq f_s$ 是直接的。为了显示 $\tilde{f}_s \geq f_s - \frac{n}{k+1}$，将每次减计数器步骤视为消除一些项目。当当前读取的 a_i 不在列表上并且已经有 k 个不同于它的符号在列表上时，项目被消除；在这种情况下，a_i 和其他 k 个不同的符号同时被消除。因此，每个 $s \in \{1,2,\ldots,m\}$ 的出现次数的消除实际上相当于 $k+1$ 个不同符号的消除。因此，任何符号的出现次数不能被消除超过 $\frac{n}{k+1}$ 次。很明显，如果一个项目没有被消除，那么它必须仍然在列表的末尾。这证明了定理。

流的二次矩

本节关注的是计算来自 $\{1,2,\ldots,m\}$ 符号流的二次矩。设 f_s 表示符号 s 在流中出现的次数，回忆一下，流的二次矩由下式给出：

$$F_2 = \sum_{s=1}^{m} f_s^2$$

为了计算二次矩，对于每个符号 s $(1 \leq s \leq m)$，独立地设置一个随机变量 x_s 为 ± 1，概率为 $\frac{1}{2}$。特别地，可以将 x_s 视作随机哈希函数 $h(s)$ 的输出，该函数的值域仅为两个桶 $\{-1,1\}$。目前，可以认为 h 是完全独立的哈希函数。通过每次符号 s 出现时向总和添加 x_s 来维护一个总和。在流结束时，总和将等于 $\sum_{s=1}^{m} x_s f_s$。该总和的期望值为零，这里的期望是针对 x_s 的 ± 1 值的选择。

$$E\left[\sum_{s=1}^{m} x_s f_s\right] = 0$$

虽然总和的期望值为零，但它的实际值是一个随机变量，总和平方的期望值由下式给出：

$$E\left[\left(\sum_{s=1}^{m} x_s f_s\right)^2\right] = E\left[\sum_{s=1}^{m} x_s^2 f_s^2\right] + 2E\left[\sum_{s \neq t} x_s x_t f_s f_t\right] = \sum_{s=1}^{m} f_s^2$$

最后的等式成立是因为 $E(x_s x_t) = E(x_s)E(x_t) = 0$ 对于 $s \neq t$，这里使用了随机变量的成对独立性。因此，

$$a = \left(\sum_{s=1}^{m} x_s f_s\right)^2$$

是对 $\sum_{s=1}^{m} f_s^2$ 的无偏估计量，因为它具有正确的期望。注意到此时我们可以使用马尔可夫不等式来说明 $P(a \geq 3\sum_{s=1}^{m} f_s^2) \leq \frac{1}{3}$，但我们想要更严格的保证。为此，考虑 a 的二次矩：

$$E(a^2) = E\left[\left(\sum_{s=1}^{m} x_s f_s\right)^4\right] = E\left[\sum_{1\leq s,t,u,v\leq m} x_s x_t x_u x_v f_s f_t f_u f_v\right]$$

最后一个等式是通过展开得到的。假设随机变量 x_s 是四次独立的，或者等价地说，它们是由四次独立的哈希函数产生的。那么，由于 x_s 在最后一个和中的独立性，如果 $s, u, t, s, u,$ 或 v 中的任何一个与其他不同，则该项的期望值为零。因此，我们只需要处理形式为 $x_s^2 x_t^2$（当 $t = s$ 时）和 x_s^4 的项。

每个上述和中的项有四个索引 s, t, u, v，并且有 $\binom{4}{2}$ 种方式选择两个索引使它们具有相同的 x 值。因此，

$$E(a^2) \leq \binom{4}{2} E\left[\sum_{s=1}^{m}\sum_{t=s+1}^{m} x_s^2 x_t^2 f_s^2 f_t^2\right] + E\left[\sum_{s=1}^{m} x_s^4 f_s^4\right] = 6\sum_{s=1}^{m}\sum_{t=s+1}^{m} f_s^2 f_t^2 + \sum_{s=1}^{m} f_s^4 \leq 3\left(\sum_{s=1}^{m} f_s^2\right)^2 = 3E^2(a).$$

因此，方差

$$Var(a) = E(a^2) - E^2(a) \leq 2E^2(a).$$

由于方差与期望的平方相当，重复这个过程多次并取平均值，可以以高概率获得高精度。

定理：使用 $r = \frac{2}{\epsilon^2\delta}$ 次独立的四次独立随机变量集估计 $a_1, \ldots, a_r$ 的平均值 x，则

$$P(|x - E(x)| > \epsilon E(x)) < \frac{Var(x)}{\epsilon^2 E^2(x)} \leq \delta.$$

证明：这个结论来自于取 r 次独立重复的平均值会将方差减少 r 倍，因此 $Var(x) \leq \delta\epsilon^2 E^2(x)$，然后应用切比雪夫不等式。

接下来需要展示我们可以用 $O(\log m)$ 的空间实现所需的四次独立随机变量。我们之前给出了构造成对独立哈希函数集的方法；现在我们需要四次独立，尽管只到 $\{-1, 1\}$ 的范围。以下是这样一个构造的例子。

纠错码、多项式插值和有限独立性

考虑生成一个随机的 m 维 ± 1 向量 x，使得任意四个坐标是相互独立的。这样的 m 维向量可以从仅有 $O(\log m)$ 个相互独立的位的真实随机 “种子” 生成。因此，我们只需要存储 $O(\log m)$ 位，并且可以在需要时生成任何 m 个坐标。对于任何 k，存在一个精确包含 2^k 个元素的有限域 F，每个元素可以

用 k 位表示，并且可以在 $O(k^2)$ 时间内执行域中的算术运算。这里，k 是 $\lceil \log_2 m \rceil$。关于多项式插值的一个基本事实是，一个最多三次的多项式可以通过其在任何域 F 上的四个点的值唯一确定。更具体地说，对于 F 中的任何四个不同点 a_1, a_2, a_3, a_4 和 F 中的任何四个可能不不同的值 b_1, b_2, b_3, b_4，存在一个唯一的最多三次的多项式 $f(x) = f_0 + f_1 x + f_2 x^2 + f_3 x^3$，使得 $f(a_i) = b_i$ $(1 \le i \le 4)$，计算是在 F 上进行的。

定义四次独立的伪随机 ± 1 向量 x 非常简单。从 F 中随机选择四个元素 f_0, f_1, f_2, f_3 并形成多项式 $f(s) = f_0 + f_1 s + f_2 s^2 + f_3 s^3$。这个多项式代表 x 如下：对于 $s = 1, 2, \ldots, m$，x_s 是 $f(s)$ 的 k 位表示的最高位。因此，m 维向量 x 仅需 $O(k)$ 位，其中 $k = \lceil \log_2 m \rceil$。

引理：上述定义的 x 具有四次独立性。

证明：假设 F 的元素用 ± 1 而不是传统的 0 和 1 表示。令 s, t, u, v 为 x 的任意四个坐标，而 $\alpha, \beta, \gamma, \delta$ 的值在 ± 1 中。恰好有 2^{k-1} 个 F 中的元素其最高位是 α，同样适用于 β, γ, δ。因此，恰好有 $2^{4(k-1)}$ 个 F 中的 4-元组 b_1, b_2, b_3, b_4 使得 b_1 的最高位是 α，b_2 的最高位是 β，b_3 的最高位是 γ，b_4 的最高位是 δ。对于每个这样的 b_1, b_2, b_3, b_4，存在一个唯一的多项式 f 使得 $f(s) = b_1, f(t) = b_2, f(u) = b_3, f(v) = b_4$。$x_s = \alpha, x_t = \beta, x_u = \gamma, x_v = \delta$ 的概率恰好为

$$\frac{2^{4(k-1)}}{\text{总的} f \text{ 的数量}} = \frac{2^{4(k-1)}}{2^{4k}} = \frac{1}{16}.$$

四次独立性由此得出，因为 $P(x_s = \alpha) = P(x_t = \beta) = P(x_u = \gamma) = P(x_v = \delta) = \frac{1}{2}$，因此

$$P(x_s = \alpha)P(x_t = \beta)P(x_u = \gamma)P(x_v = \delta) = P(x_s = \alpha, x_t = \beta, x_u = \gamma \text{ and } x_v = \delta).$$

引理描述了如何获得一个具有四次独立性的向量 x。然而，我们需要 $r = O\left(\frac{1}{\epsilon^2}\right)$ 个互不相关的向量。开始时选择 r 个独立的多项式。

为了用低空间复杂度实现算法，只需在内存中存储这些多项式。这需要每个多项式 $4k = O(\log m)$ 位，总共 $O(r \log m) = O\left(\frac{\log m}{\epsilon^2}\right)$ 位。当流中的符号 s 被读取时，计算每个多项式在 s 处的值以获得对应的 x_s 值，并更新运行总和。x_s 就是多项式在 s 处评估值的最高位。这种计算需要 $O(\log m)$ 时间。因此，我们反复从“种子”，即多项式的系数，计算 x_s。

多项式插值的思想也在其他上下文中使用。纠错码就是一个重要的例子。为了通过可能会引入噪声的通道传输 n 位，可以在传输中引入冗余，以便可以纠正一些通道错误。一种简单的方法是将要传输的 n 位视为最多 $n-1$ 次的多项式 $f(x)$ 的系数。现在在点 $1, 2, 3, \ldots, n+m$ 处传输 f 的值。在接收端，任何 n 个正确值都足以重构多项式和真实消息。因此，可以容忍多达 m 个错误。但是，即使错误数量最多为 m，也并不容易知道哪些值被破坏了。我们在这里不做详细说明。