我们将在第2节中详细讨论检测部分拷贝的大多数以前的技术都使用了以下思想。k-gram是长度为k的连续子字符串。将文档划分为k-gram，其中k是用户选择的参数。例如，图1（c）包含图1（b）中字符串的所有5-gram。请注意，文档中的k-gram几乎与字符数量相同，因为文档中的每个位置（除了最后的k-1位置）都标记k-gram的开始。现在对每个k-gram进行散列，并选择这些散列的一些子集作为文档的指纹。在所有实际方法中，指纹集是所有k-gram散列集的一小部分。指纹还包含位置信息，我们不会显示这些信息，这些信息描述文档以及该文档中指纹的来源位置。如果选择散列函数使得冲突的概率非常小，那么每当两个文档共享一个或多个指纹时，它们也极有可能共享一个k-gram。

为了提高效率，应该只保留散列的一个子集作为文档的指纹。一种流行的方法是选择所有0 mod p的散列，对于某些固定p。这种方法易于实现，并且只保留所有散列的1/p作为指纹（第2节）。文档相似度的有意义的度量也可以从文档之间共享的指纹数中得到。

这种方法的一个缺点是，它不能保证检测到文档之间的匹配：只有当散列为0 mod p时，才能检测到文档之间共享的k-gram。请考虑将文件的所有k-gram按顺序散列生成的散列序列。将连续选定指纹之间的距离称为它们之间的间距。如果选择0 mod p的指纹，两个指纹之间的最大间隔是无界的，并且没有检测到间隔内的任何匹配项。1在从Web获取的HTML数据上选择0 mod p的指纹的实验中，我们发现选择0 mod p的指纹之间的间隔可能相当大。事实上，有些段落比普通网页的大小要长得多，其中没有哈希值为0 mod p的合理值p（第5节）

在本文中，我们给出了第一个有效的算法，从一个哈希序列中选择指纹，保证至少有一部分足够长的匹配被检测到。将大小为w的窗口定义为文档中k-gram的w个连续散列（w是用户设置的参数）。通过从每个窗口中选择至少一个指纹，我们的算法限制了指纹之间的最大间隔。事实上，我们的算法保证在长度至少为w+k−1的任何共享子串中检测至少一个k-gram。

我们构造的核心是局部算法的思想（第4节），我们相信它捕获了任何文档指纹算法的基本属性，从而保证匹配被检测到。如果对于w个连续散列hi，…，hi+w−1的每个窗口，算法决定选择其中一个散列作为指纹，并且此选择仅取决于窗口的内容hi，…，hi+w−1，则算法是局部的。

请注意，由于算法在其中一个（可能有许多）重叠窗口中的操作，也可以选择此窗口中的一些其他哈希。

对于给定的输入分布，让一个指纹算法的密度是它选择作为指纹的哈希的期望比例。在第三节中，我们将介绍一种特殊的局部算法，即风选算法。我们分析了它在随机（独立均匀分布）数据上的性能。我们证明，在给定窗口大小w的情况下，密度渐近为2/（w+1）。我们还证明了局部算法密度的下界为1.5/（w+1）。因此，风选算法是在33%以内的最佳

我们还报告了两个winnowing实现的经验。第一个是一个纯粹的实验框架，用于比较实际性能和理论预测（第5节）。

我们对所有指纹算法的分析，包括筛选算法的分析，都是基于对k-grams进行散列生成的散列序列是独立且均匀随机的假设。在这样的随机数据上，我们系统的性能完全符合理论预测。然而，我们发现，在某些情况下，真实数据不能生成足够的随机散列序列

这类字符串的重要特征是它们很少有不同的k-gram，因此k-gram的散列也很少。对于单个字符的长字符串，只有一个k-gram（和散列）。简单的winnowing算法在这类字符串上选择的指纹比预测的多得多，但是对算法的简单修改会降低指纹的密度2。请注意，其他指纹选择方法也面临同样的问题。例如，选择0 mod p的哈希值意味着在字符串“11111111…”上，该字符串的所有哈希值或没有哈希值被选为指纹。

2一些其他的算法

3

在这一节中，我们描述并分析了从k-gram散列中选择指纹的winnowing算法。我们给出了筛选性能的上界，表示为必须选择的指纹数和保证检测到的最短匹配之间的折衷。

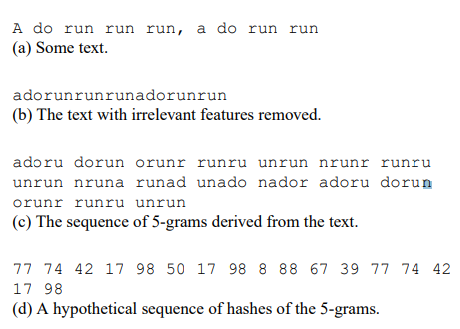
给定一组文档，我们希望它们之间的find子字符串匹配满足两个属性：

1如果存在至少等于保证阈值t的子串匹配，则检测到该匹配，并且

2我们没有检测到任何比噪声阈值k短的匹配。 ？？？

常数t和k≤t由用户选择。我们通过只考虑k-gram的散列来避免匹配噪声阈值以下的字符串。k越大，我们就越有信心文档之间的匹配不是巧合。另一方面，较大的k值也限制了对文档内容重新排序的敏感性，因为我们无法检测长度小于k的任何子字符串的重新定位。因此，选择k作为消除重合匹配的最小值非常重要（参见第5节）。

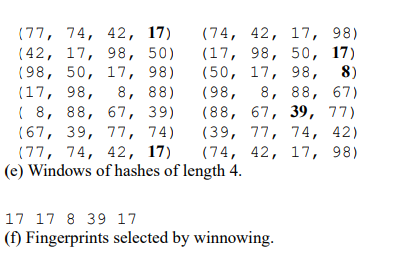
为了方便起见，图2（a）-（d）是从图1复制而来的，并显示了从一些示例文本派生的5克哈希序列。给定一系列散列h1…hn，如果n>t−k，则必须选择至少一个hi以保证检测到长度至少为t的所有匹配。这表明了以下简单方法。让窗口大小为w=t−k+1。考虑表示文档的散列h1h2…hn的序列。该序列中的每个位置1≤i≤n−w+1定义散列hi…hi+w−1的窗口。为了保持这种保证，必须从每个窗口中选择一个散列值作为文档的指纹。（这也足够了，见引理1）我们发现下面的策略在实践中效果很好。



定义1（风选）。在每个窗口中选择最小哈希值。如果存在多个具有最小值的哈希，请选择最右边的匹配项。现在将所有选定的哈希保存为文档的指纹

图2（e）给出了图2（d）中哈希序列的长度为4的窗口。选定的每个哈希都以粗体显示（但仅在第一个选择该哈希的窗口中显示一次）。选择最小散列的直觉是，一个窗口中的最小散列很可能保持相邻窗口中的最小散列，因为w个随机数的最小值很可能小于一个额外的随机数。

因此，许多重叠的窗口选择相同的散列，并且选择的指纹数远小于窗口数，同时仍然保持保证。图2（f）显示了示例中通过风选选择的指纹集。



在许多应用中，不仅要记录文档的指纹，而且还要记录指纹在文档中的位置。例如，我们需要位置信息来显示用户界面中匹配的子字符串。一个有效的风选实现还需要保留最近选择的指纹的位置。图2（f）显示了本例的[fingerprint，position]对集（第一个位置编号为0）。为了避免索引所有散列及其在文档k-gram的全局散列序列中的位置的符号复杂性，我们在呈现中抑制了对文档中k-gram位置的最显式引用。



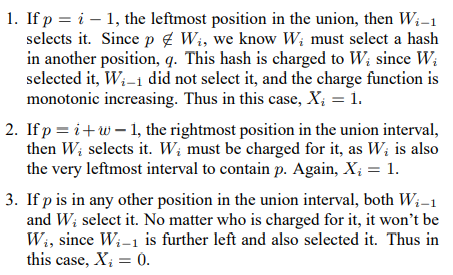
3.1期望密度

回想一下，指纹算法的密度是在给定随机输入的情况下，从所有计算的哈希值中选择的指纹的期望分数（第1节）。我们现在分析风选的密度，它给出了保证阈值和所需指纹数之间的权衡。

考虑函数C，它将每个选定指纹的位置映射到文档的所有窗口序列中选择它的第一个（最左边）窗口的位置。我们说，我们需要支付保存指纹到指定的窗口的消耗。函数是单调递增的，也就是说，如果p和q是两个选定指纹的位置，p<q，那么C（p）<C（q）。

进一步回顾一下，我们筛选的哈希序列是随机的。我们假设散列值的空间非常大，这样我们就可以安全地忽略这样一种可能性，即对于任何较小的窗口大小，最小值都有一个对应关系。我们在第5节中检验了这个假设的合理性。

考虑一个指标随机变量Xi，当第i个窗口被选择Xi置为1。考虑左侧Wi−1的相邻窗口。除了最左边和最右边的位置，这两个间隔重叠。它们的并集是一个长度为w+1的区间。考虑在该并集间隔中包含最小散列的位置p。任何包含p的窗口都会选择hp作为指纹。有三种情况：



前两种情况发生的概率为1/（w+1），因此Xi的期望值为2/（w+1）。回想一下，期望值之和就是期望值之和，即使随机变量不是独立的。预期的总间隔数，以及因此选择的指纹总数，就是这个值乘以文档长度。因此密度是



3.1.1与相同密度下的0 mod p比较

3.1.2 Comparison to 0 mod p with guarantee