等方返回该（键，值）对。任何对等方也将允许在数据库中插入新键－值对。这样一种分布式数据库被称为**分布式散列表**（Distributed Hash Table，DHT）。

在描述如何创建一个 DHT 的方法之前，我们首先描述在 P2P 文件共享环境中 DHT 服务的一个特定例子。在这种情况下，键是一个目录名，而值是具有该目录副本的对等方 IP 地址。因此，如果 Bob 和 Charlie 都具有一个最新 Linux 分发副本的话，则 DHT 数据库将包括下列两个键－值对：（Linux，IP_{Bob}）和（Linux，$IP_{Charlie}$）。更为具体地说，因为 DHT 数据库分布在一些对等方上，所以某些对等方如 Dave 将对键“Linux”负责，并且将具有相应的键－值对。现在假设 Alice 要获得 Linux 的一个副本。显然，她在能够开始下载之前，先需要知道哪个对等方拥有 Linux 的副本。为此，她用“Linux”作为键来查询 DHT。该 DHT 则确定对等方 Dave 负责键“Linux”。DHT 则联系对等方 Dave，从 Dave 处获得键－值对（Linux，IP_{Bob}）和（Linux，$IP_{Charlie}$），并将它们传送给 Alice。Alice 则能从 IP_{Bob} 或 $IP_{Charlie}$ 处下载最新 Linux 分发。

现在我们返回到为通用键－值对设计一个 DHT 这个一般性的问题上。构建 DHT 的一种幼稚的方法是跨越所有对等方随机地散布（键，值）对，让每个对等方维护一个所有参与对等方的列表。在该设计中，查询的对等方向所有其他对等方发送它的查询，并且包含与键匹配的（键，值）对的对等方能够用它们匹配的对进行响应。当然，这样的方法完全无扩展性，因为它将要求每个对等方不仅知道所有其他对等方（这样的对等方可能数以百万计！），而且更糟的是，要向所有对等方发送一个查询。

我们现在描述设计 DHT 的一种精确有效的方法。为此，我们现为每个对等方分配一个标识符，其中每个标识符是一个［0，2^n-1］范围内的整数，n 取某些固定的值。值得注意的是这样的标识符能够由一个 n 比特表示法来表示。我们也要求每个键是同一范围内的一个整数。敏锐的读者也许已经观察到刚才描述的键的例子（社会保险号和目录名）并非是整数。为在这些键范围外生成整数，我们将使用散列函数把每个键（如社会保险号）映射为［0，2^n-1］范围内的一个整数。散列函数是一种多对一的函数，使两个不同的输入能够具有相同的输出（相同的整数），但是具有相同输出的似然性极低。（不熟悉散列函数的读者可以看一下第 7 章，其中更为细致地讨论了散列函数。）假定系统中的所有对等方可以使用散列函数。因此，当我们提及“键”，指的是初始键的散列值。因此，假如初始键是“Led Zppelin IV”，在 DHT 中使用的键将是等于“Led Zppelin IV”散列值的整数。如你猜测的那样，这就是“散列”被用于术语“分布式散列表”中的理由。

现在我们来考虑在 DHT 中存储（键，值）对的问题。此时的中心问题是定义为对等方分配键的规则。给定每个对等方具有一个整数标识符，每个键也是一个位于相同范围的整数，一种当然的方法是为其标识符最邻近该键的对等方分配一个（键，值）对。为实现这样的方案，我们将需要定义“最邻近”的含义。为了方便起见，我们将最邻近对等方定义为键的*最邻近后继*。为了加深理解，我们来看一个特定的例子。假设 $n=4$，使所有对等方和键标识符都位于［0，15］范围内。进一步假定在该系统中有 8 个对等方，它们的标识符分别为 1、3、4、5、8、10、12 和 15。假定要在这 8 个对等方之一上存储（键，值）对（11，Johnny Wu）。但是放在哪个对等方上呢？使用我们定义的最邻近规则，因为对等方 12 是键 11 最邻近的后继，因此我们将（11，Johnny Wu）存储在对等方 12 上。［为了完成我们的“最邻近”定义，说明如下：如果该键恰好等于这些对等方标识符之一，我们在匹配的对等方中存储（键，值）对；并且如果该键大于所有对等方标识符，我们使用模