小于 *TableSize* 的素数。如果我们选择 $R=7$，则图 5-18 显示插入与前面相同的一些关键字的结果。

	Empty Table	After 89	After 18	After 49	After 58	After 69
0						69
1						
2						
3					58	58
4						
5						
6				49	49	49
7						
8			18	18	18	18
9		89	89	89	89	89

图 5-18　使用双散列方法在每次插入后的散列表

第一个冲突发生在 49 被插入的时候。$hash_2(49)=7-0=7$，故 49 被插入到位置 6。$hash_2(58)=7-2=5$，于是 58 被插入到位置 3。最后，69 产生冲突，从而被插入到距离为 $hash_2(69)=7-6=1$ 远的地方。如果我们试图将 60 插入到位置 0 处，那么就会产生一个冲突。由于 $hash_2(60)=7-4=3$，因此我们尝试位置 3、6、9，然后是 2，直到找出一个空的单元。一般是有可能发现某个坏情形的，不过这里没有太多这样的情形。

前面已经提到，上面的散列表实例的大小不是素数。我们这么做是为了计算散列函数时方便，但是，有必要了解在使用双散列时为什么保证表的大小为素数是很重要的。如果想要把 23 插入到表中，那么它就会与 58 发生冲突。由于 $hash_2(23)=7-2=5$，且该表大小是 10，因此我们实际上只有一个备选位置，而这个位置已经被使用了。因此，如果表的大小不是素数，那么备选单元就有可能提前用完。然而，如果双散列正确实现，则模拟表明，预期的探测次数几乎和随机冲突解决方法的情形相同。这使得双散列理论上很有吸引力。不过，平方探测不需要使用第二个散列函数，从而在实践中使用可能更简单并且更快，特别对于像串这样的关键字，它们的散列函数计算起来相当耗时。

5.5　再散列

对于使用平方探测的开放定址散列法，如果散列表填得太满，那么操作的运行时间将开始消耗过长，且插入操作可能失败。这可能发生在有太多的移动和插入混合的场合。此时，一种解决方法是建立另外一个大约两倍大的表(而且使用一个相关的新散列函数)，扫描整个原始散列表，计算每个(未删除的)元素的新散列值并将其插入到新表中。

例如，设将元素 13、15、24 和 6 插入到大小为 7 的线性探测散列表中。散列函数是 $h(x)=x \bmod 7$。设使用线性探测方法解决冲突问题。插入结果得到的散列表如图 5-19 所示。

如果将 23 插入表中，那么图 5-20 中插入后的表将有超过 70% 的单元是满的。因为散列表太满，所以我们建立一个新的表。该表大小所以为 17，是因为 17 是原表大小两倍后的第一个素数。新的散列函数为 $h(x)=x \bmod 17$。扫描原来的表，并将元素 6、15、23、24 和 13 插入到新表中。最后得到的表见图 5-21。