方探测情况甚至更糟：一旦表被填充超过一半，当表的大小不是素数时甚至在表被填充一半之前，就不能保证一次找到空的单元了。这是因为最多有表的一半可以用作解决冲突的备选位置。

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

图 5-13 在每次插入后，利用平方探测得到的散列表

我们现在就来证明，如果表有一半是空的，并且表的大小是素数，那么我们保证总能够插入一个新的元素。

定理 5.1 如果使用平方探测，且表的大小是素数，那么当表至少有一半是空的时候，总能够插入一个新的元素。

证明：

令表的大小 *TableSize* 是一个大于 3 的(奇)素数。我们证明，前⌈*TableSiaze*/2⌉个备选位置(包括初始位置 $h_0(x)$)是互异的。$h(x)+i^2(\text{mod } TableSize)$和 $h(x)+j^2(\text{mod } TableSize)$是这些位置中的两个，其中 $0 \leqslant i, j \leqslant \lfloor TableSiaze/2 \rfloor$。为推出矛盾，假设这两个位置相同，但 $i \neq j$，于是

$$\begin{aligned} h(x)+i^2 &= h(x)+j^2 && (\text{mod } TableSize) \\ i^2 &= j^2 && (\text{mod } TableSize) \\ i^2-j^2 &= 0 && (\text{mod } TableSize) \\ (i-j)(i+j) &= 0 && (\text{mod } TableSize) \end{aligned}$$

由于 *TableSize* 是素数，因此，要么$(i-j)$等于 0(mod *TableSize*)要么$(i+j)$等于 0(mod *TableSize*)。既然 i 和 j 是互异的，那么第一个选择是不可能的。但 $0 \leqslant i, j \leqslant \lfloor TableSiaze/2 \rfloor$，因此第二个选择也是不可能的。从而，前⌈*TableSiaze*/2⌉个备选位置是互异的。如果最多有⌊*TableSiaze*/2⌋个位置被使用，那么空单元总能够找到。 ■

即使表被填充的位置仅仅比一半多一个，那么插入都有可能失败(虽然这是非常难于见到的)。因此，把它记住很重要。另外，表的大小是素数也非常重要[○]。如果表的大小不是素数，则备选单元的个数可能会锐减。例如，若表的大小是 16，那么备选单元只能在距散列值 1，4 或 9 远处。

在探测散列表中标准的删除操作不能执行，因为相应的单元可能已经引起过冲突，元素绕过它存在了别处。例如，如果我们删除 89，那么实际上所有剩下的 contains 操作都将失败。因此，探测散列表需要懒惰删除，不过在这种情况下实际上并不存在所意味的懒惰。

实现探测散列表所需要的类架构如图 5-14 中所示。这里，我们不用链表数组，而是使用散列

○ 如果表的大小是形如 $4k+3$ 的素数，且使用的平方冲突解决方法为 $F(i)=\pm i^2$，那么整个表均可被探测到。其代价则是例程要略微复杂。