

0	6
1	15
2	
3	24
4	
5	
6	13

图 5-19 使用线性探测插入 13、15、6、24 的散列表

0	6
1	15
2	23
3	24
4	
5	
6	13

图 5-20 使用线性探测插入 23 后的散列表

0	
1	
2	
3	
4	
5	
6	6
7	23
8	24
9	
10	
11	
12	
13	13
14	
15	15
16	

图 5-21 在再散列之后的线性探测散列表

整个操作就叫做再散列 (rehashing)。显然这是一种开销非常大的操作；其运行时间为 $O(N)$ ，因为有 N 个元素要再散列而表的大小约为 $2N$ ，不过，由于不是经常发生，因此实际效果根本没有这么差。特别是在最后的再散列之前必然已经存在 $N/2$ 次 insert，因此添加到每个插入上的花费基本上是一个常数开销^①。如果这种数据结构是程序的一部分，那么其影响是不明显的。另一方面，如果再散列作为交互系统的一部分运行，那么其插入引起再散列的不幸用户将会感到速度减慢。

再散列可以用平方探测以多种方法实现。一种做法是只要表满到一半就再散列。另一种极端的方法是只有当插入失败时才再散列。第三种方法即途中 (middle-of-the-road) 策略：当散列表到达某一个装填因子时进行再散列。由于随着装填因子的增长散列表的性能确实下降，因此，以好的截止手段实现的第三种策略，可能是最好的策略。

对于分离链接散列表其再散列是类似的。图 5-22 显示再散列实现起来是简单的，并且还分离链接再散列提供一种实现方法。

```
1      /**
2      * Rehashing for quadratic probing hash table.
3      */
4      private void rehash( )
5      {
6          HashEntry<AnyType> [ ] oldArray = array;
```

图 5-22 对分离链接散列表和探测散列表的再散列

① 这就是为什么新表要做成老表两倍大的原因。