下滑一层(见图 6-10)。重复该过程，由于 31 大于 19，因此把 19 置入空穴，在更下一层上建立一个新的空穴。然后，由于 31 还是太大,因此再把 26 置入空穴，在底层又建立一个新的空穴。最后，我们得以将 31 置入空穴中(图 6-11)。这种一般的策略叫做**下滤**(percolate down)。在其实现例程中我们使用类似于在 insert 例程中用过的技巧来避免进行交换操作。

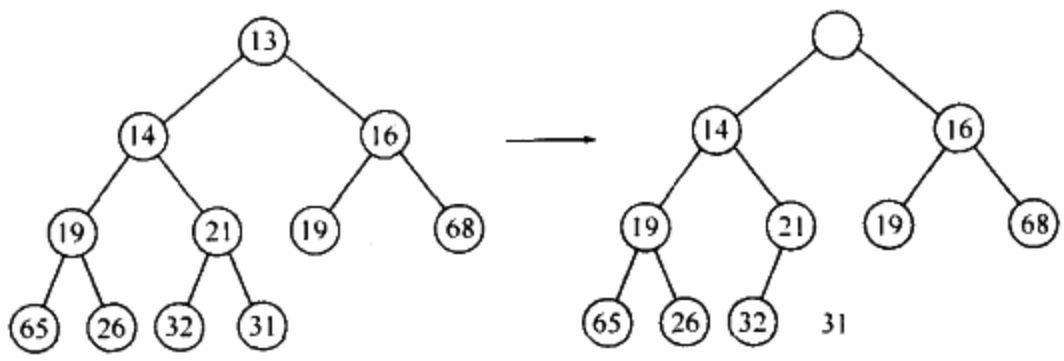

图 6-9 在根处建立空穴

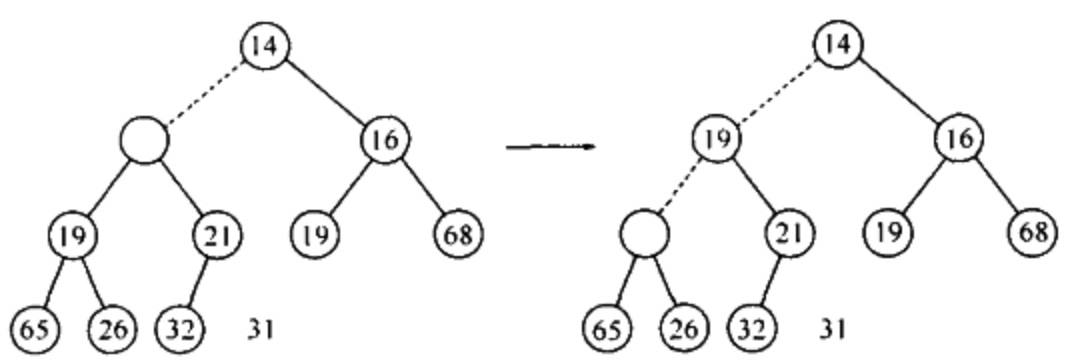

图 6-10 在 deleteMin 中的接下来的两步

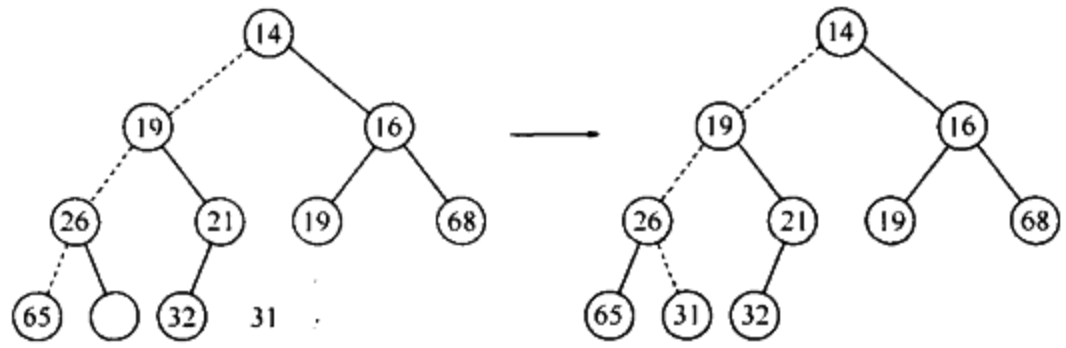

图 6-11 在 deleteMin 中的最后两步

在堆的实现中经常发生的错误是当堆中存在偶数个元素的时候，将遇到一个节点只有一个儿子的情况。我们必须保证节点不总有两个儿子的前提，因此这就涉及一个附加的测试。在图 6-12 描述的程序中，我们已在第 29 行进行了这种测试。一种极其巧妙的解决方法是始终保证算法把每一个节点都看成有两个儿子。为了实施这种解法，当堆的大小为偶数时在每个下滤开始处，可将其值大于堆中任何元素的标记放到堆的终端后面的位置上。我们必须在深思熟滤以后再这么做，而且必须插入一个是否确实使用这种技巧的评判。虽然这不再需要测试右儿子的存在性，但是还是需要测试何时到达底层，因为对每一片树叶算法将需要一个标记。

这种操作最坏情形运行时间为 $O(\log N)$。平均而言，被放到根处的元素几乎下滤到堆的底层(即它所来自的那层)，因此平均运行时间为 $O(\log N)$。

6.3.4 其他的堆操作

注意，虽然求最小值操作可以在常数时间完成，但是，按照求最小元设计的堆(也称做最小堆,(min)heap)在求最大元方面却无任何帮助。事实上，一个堆所蕴涵的序信息很少，因此，若不