6.3.2 堆序性质

让操作快速执行的性质是**堆序性质**(heap-order property)。由于我们想要快速找出最小元，因此最小元应该在根上。如果我们考虑任意子树也应该是一个堆，那么任意节点就应该小于它的所有后裔。

应用这个逻辑，我们得到堆序性质。在一个堆中，对于每一个节点 X，X 的父亲中的关键字小于(或等于)X 中的关键字，根节点除外(它没有父亲)[⊖]。在图 6-5 中左边的树是一个堆，而右边的树则不是(虚线表示堆有序性被破坏)。

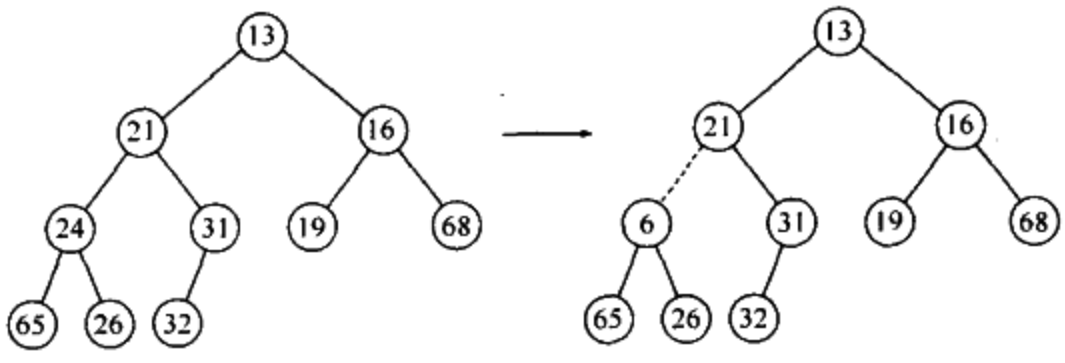

图 6-5 两棵完全树(只有左边的树是堆)

根据堆序性质，最小元总可以在根处找到。因此，我们以常数时间得到附加操作 findMin。

6.3.3 基本的堆操作

无论从概念上还是实际上考虑，执行这两个所要求的操作都是容易的。所有的工作都需要保证始终保持堆序性质。

insert(插入)

为将一个元素 X 插入到堆中，我们在下一个可用位置创建一个空穴，否则该堆将不是完全树。如果 X 可以放在该空穴中而并不破坏堆的序，那么插入完成。否则，我们把空穴的父节点上的元素移入该空穴中，这样，空穴就朝着根的方向上冒一步。继续该过程直到 X 能被放入空穴中为止。如图 6-6 所示，为了插入 14，我们在堆的下一个可用位置建立一个空穴。由于将 14 插入空穴破坏了堆序性质，因此将 31 移入该空穴。在图 6-7 中继续这种策略，直到找出置入 14 的正确位置。

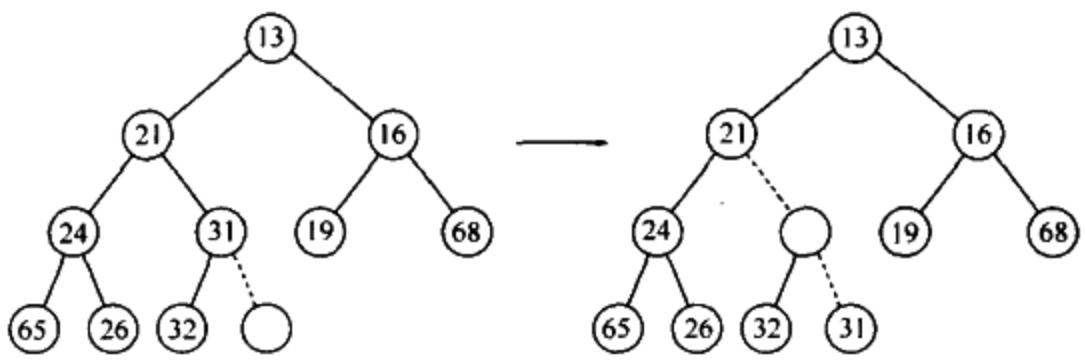

图 6-6 尝试插入 14：创建一个空穴，再将空穴上冒

这种一般的策略叫做**上滤**(percolate up)；新元素在堆中上滤直到找出正确的位置。使用图 6-8所示的代码很容易实现插入。

⊖ 类似地，我们可以声明一个(max)堆，它使我们通过改变堆序性质能够有效地找出和删除最大元。因此，优先队列可以用来找出最大元或最小元，但这需要提前决定。