将会大大增加运行时间，因为我们不能再通过移一个二进制位来实现除法了。d-堆在理论上很有趣，因为存在许多算法，其插入次数比 deleteMin 的次数多得多(因此理论上的加速是可能的)。当优先队列太大而不能完全装入主存的时候，d-堆也是很有用的。在这种情况下，d-堆能够以与 B 树大致相同的方式发挥作用。最后，有证据显示，在实践中 4-堆可以胜过二叉堆。

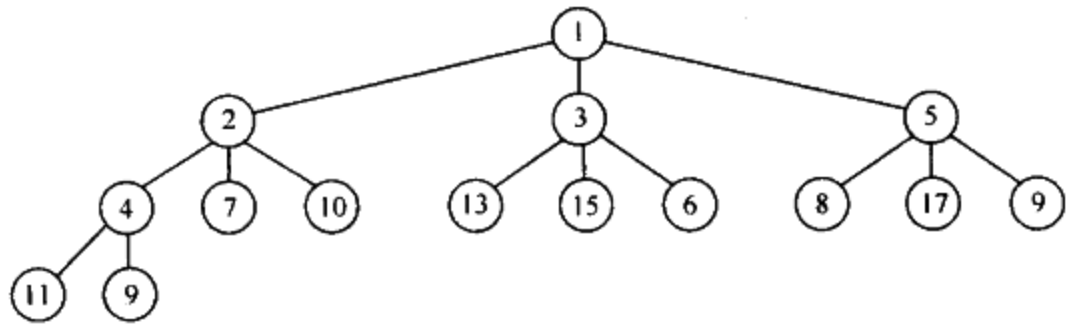

图 6-19 一个 d-堆

除不能实施 find 外，堆实现的最明显的缺点是：将两个堆合并成一个堆是困难的操作。这种附加的操作叫做合并(merge)。存在许多实现堆的方法使得一次 merge 操作的运行时间是 $O(\log N)$。现在我们就来讨论三种复杂程度不一的数据结构，它们都有效地支持 merge 操作。我们将把复杂的分析推迟到第 11 章讨论。

6.6 左式堆

设计一种堆结构像二叉堆那样有效地支持合并操作(即以 $o(N)$时间处理一个 merge)而且只使用一个数组似乎很困难。原因在于，合并似乎需要把一个数组拷贝到另一个数组中去，对于相同大小的堆这将花费时间 $\Theta(N)$。正因为如此，所有支持有效合并的高级数据结构都需要使用链式数据结构。实践中，我们预计这将可能使得所有其他操作变慢。

左式堆(leftist heap)像二叉堆那样也具有结构性和有序性。事实上，和所有使用的堆一样，左式堆具有相同的堆序性质，该性质我们已经看到过。不仅如此，左式堆也是二叉树。左式堆和二叉堆唯一的区别是：左式堆不是理想平衡的(perfectly balanced)，而实际上趋向于非常不平衡。

6.6.1 左式堆性质

我们把任一节点 X 的零路径长(null path length)npl(X)定义为从 X 到一个不具有两个儿子的节点的最短路径的长。因此，具有 0 个或一个儿子的节点的 npl 为 0，而 npl(null) = −1。在图 6-20 的树中，零路径长标记在树的节点内。

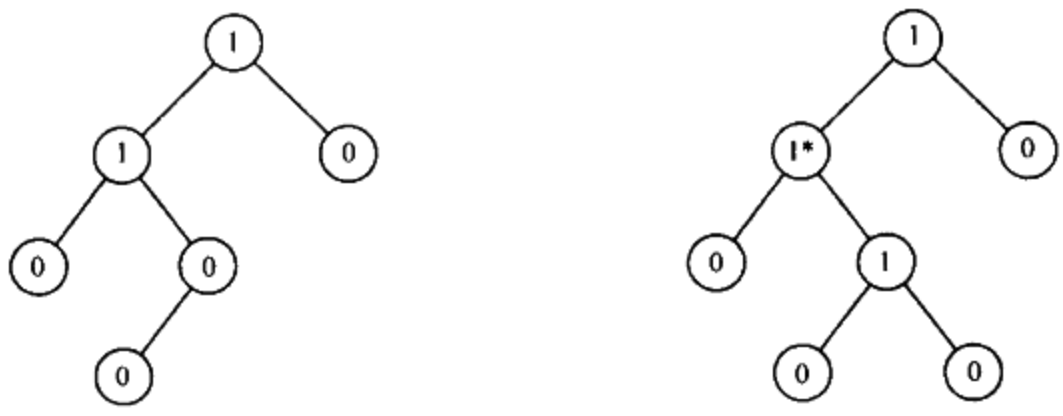

图 6-20 两棵树的零路径长；只有左边的树是左式的

注意，任一节点的零路径长比它的各个儿子节点的零路径长的最小值大 1。这个结论也适用少于两个儿子的节点，因为 null 的零路径长是 −1。