6.2 一些简单的实现

有几种明显的方法可用于实现优先队列。我们可以使用一个简单链表在表头以 $O(1)$执行插入操作，并遍历该链表以删除最小元，这又需要 $O(N)$时间。另一种方法是始终让链表保持排序状态；这使得插入代价高昂($O(N)$)而 deleteMin 花费低廉($O(1)$)。基于 deleteMin 的操作从不多于插入操作的事实，前者恐怕是更好的想法。

另一种实现优先队列的方法是使用二叉查找树，它对这两种操作的平均运行时间都是 $O(\log N)$。尽管插入是随机的，而删除则不是，但这个结论还是成立的。记住我们删除的唯一元素是最小元。反复除去左子树中的节点似乎会损害树的平衡，使得右子树加重。然而，右子树是随机的。在最坏的情形下，即 deleteMin 将左子树删空的情形下，右子树拥有的元素最多也就是它应具有的两倍。这只是在期望的深度上加了一个小常数。注意，通过使用一棵平衡树，可以把这个界变成最坏情形的界；这将防止出现坏的插入序列。

使用查找树可能有些过分，因为它支持许许多多并不需要的操作。我们将要使用的基本的数据结构不需要链，它以最坏情形时间 $O(\log N)$支持上述两种操作。插入操作实际上将花费常数平均时间，若无删除操作的干扰，该结构的实现将以线性时间建立一个具有 N 项的优先队列。然后，我们将讨论如何实现优先队列以支持有效的合并。这个附加的操作似乎有些复杂，它显然需要使用链接的结构。

6.3 二叉堆

我们将要使用的这种工具叫做**二叉堆**(binary heap)，它的使用对于优先队列的实现相当普遍，以至于当堆(heap)这个词不加修饰地用在优先队列的上下文中时，一般都是指数据结构的这种实现。在本节，我们把二叉堆只叫做堆。像二叉查找树一样，堆也有两个性质，即结构性和堆序性。恰似 AVL 树，对堆的一次操作可能破坏这两个性质中的一个，因此，堆的操作必须到堆的所有性质都被满足时才能终止。事实上这并不难做到。

6.3.1 结构性质

堆是一棵被完全填满的二叉树，有可能的例外是在底层，底层上的元素从左到右填入。这样的树称为完全二叉树(complete binary tree)。图 6-2 给出了一个例子。

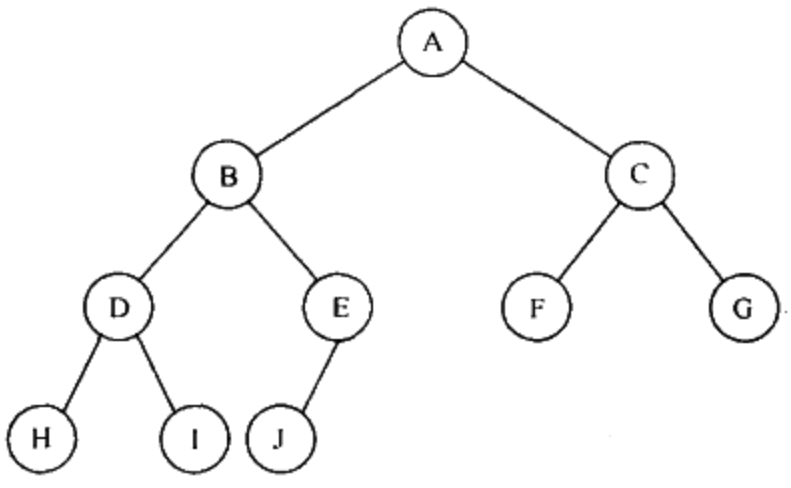

图 6-2 一棵完全二叉树

容易证明，一棵高为 h 的完全二叉树有 2^h 到 $2^{h+1}-1$ 个节点。这意味着完全二叉树的高是 $\lfloor \log N \rfloor$，显然它是 $O(\log N)$。

一个重要的观察发现，因为完全二叉树这么有规律，所以它可以用一个数组表示而不需要使