左式堆性质是：对于堆中的每一个节点 X，左儿子的零路径长至少与右儿子的零路径长相等。图 6-20 中只有一棵树，即左边的那棵树满足该性质。这个性质实际上超出了它确保树不平衡的要求，因为它显然偏重于使树向左增加深度。确实有可能存在由左节点形成的长路径构成的树(而且实际上更便于合并操作)——因此，我们就有了名称左式堆(leftist heap)。

因为左式堆趋向于加深左路径，所以右路径应该短。事实上，沿左式堆右侧的右路径确实是该堆中最短的路径。否则，就会存在过某个节点 X 的一条路径通过它的左儿子，此时 X 就破坏了左式堆的性质。

定理 6.2 在右路径上有 r 个节点的左式树必然至少有 $2^r - 1$ 个节点。

证明：

用数学归纳法证明。如果 $r = 1$，则必然至少存在一个树节点。其次，设定理对 1、2、...、r 个节点成立。考虑在右路径上有 $r+1$ 个节点的左式树。此时，根具有在右路径上含 r 个节点的右子树，以及在右路径上至少含 r 个节点的左子树(否则它就不是左式树)。对这两棵子树应用归纳假设，得知在每棵子树上最少有 $2^r - 1$ 个节点，再加上根节点，于是在该树上至少有 $2^{r+1} - 1$ 个节点，定理得证。 ■

从这个定理立刻得到，N 个节点的左式树有一条右路径最多含有$\lfloor \log(N+1) \rfloor$个节点。对左式堆操作的一般思路是将所有的工作放到右路径上进行，它保证树深度短。唯一的棘手部分在于，对右路径的 insert 和 merge 可能会破坏左式堆性质。事实上，恢复该性质是非常容易的。

6.6.2 左式堆操作

对左式堆的基本操作是合并。注意，插入只是合并的特殊情形，因为我们可以把插入看成是单节点堆与一个大的堆的 merge。首先，我们给出一个简单的递归解法，然后介绍如何能够非递归地执行该解法。我们的输入是两个左式堆 H_1 和 H_2，见图 6-21。读者应该验证，这些堆确实是左式堆。注意，最小的元素在根处。除数据、左引用和右引用所用空间外，每个节点还要有一个指示零路径长的项。

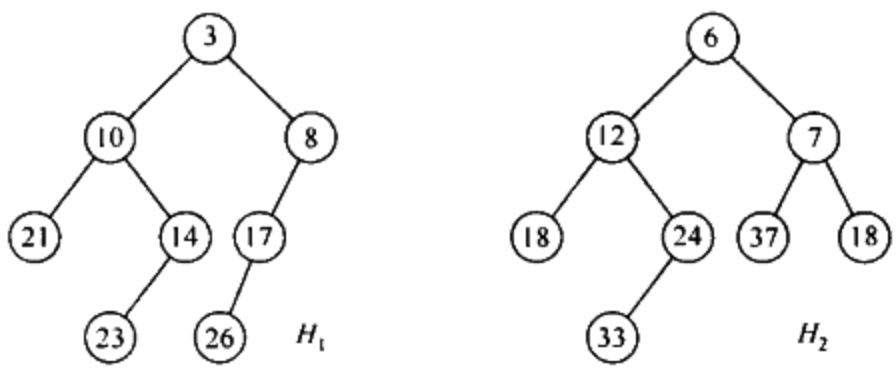

图 6-21 两个左式堆 H_1 和 H_2

如果这两个堆中有一个堆是空的，那么我们可以返回另外一个堆。否则，合并这两个堆，比较它们的根。首先，我们递归地将具有大的根值的堆与具有小的根值的堆的右子堆合并。在本例中，我们递归地将 H_2 与 H_1 的根在 8 处的右子堆合并，得到图 6-22 中的堆。

由于这棵树是递归形成的，而我们尚未对算法描述完毕，因此，现在还不能说明该堆是如何得到的。不过，有理由假设，最后的结果是一个左式堆，因为它是通过递归的步骤得到的。这很像归纳法证明中的归纳假设。既然我们能够处理基准情形(发生在一棵树是空的时候)，当然可以假设，只要能够完成合并那么递归步骤就是成立的；这是递归法则 3，我们在第一章中讨论过。现在，我们让这个新的堆成为 H_1 的根的右儿子(见图 6-23)。