可接受的，但我们可以建立一个新的根，这个根以分裂得到的两个树根作为它的两个儿子。这就是为什么准许树根可以最少有两个儿子的特权的原因。这也是B树增加高度的唯一方式。不用说，一路向上分裂直到根的情况是一种特别少见的异常事件，因为一棵具有4层的树意味着在整个插入序列中已经被分裂了3次(假设没有删除发生)。事实上，任何非叶节点的分裂也是相当少见的。

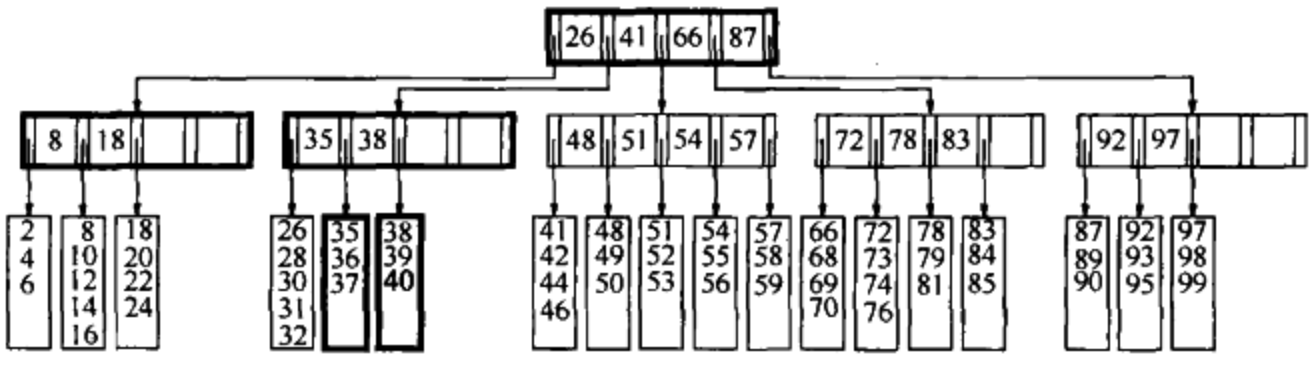

图4-62　把40插入到图4-61的B树中引起树叶被分裂成两片然后又造成父节点的分裂

还有其他一些方法处理儿子过多的情况。一种方法是在相邻节点有空间时把一个儿子交给该邻节点领养。例如，为了把29插入到图4-62的B树中，可以把32移到下一片树叶而腾出一个空间。这种方法要求对父节点进行修改，因为有些关键字受到了影响。然而，它趋向于使得节点更满，从而在长时间运行中节省空间。

我们可以通过查找要删除的项并在找到后删除它来执行删除操作。问题在于，如果被删元所在的树叶的数据项数已经是最小值，那么删除后它的项数就低于最小值了。我们可以通过在邻节点本身没有达到最小值时领养一个邻项来矫正这种状况。如果相邻结点已经达到最小值，那么可以与该相邻节点联合以形成一片满叶。可是，这意味着其父节点失去一个儿子。如果失去儿子的结果又引起父节点的儿子数低于最小值，那么我们使用相同的策略继续进行。这个过程可以一直上行到根。根不可能只有一个儿子(要是允许根有一个儿子那可就愚蠢了)。如果这个领养过程的结果使得根只剩下一个儿子，那么删除该根并让它的这个儿子作为树的新根。这是B树降低高度的唯一的方式。例如，假设我们想要从图4-62的B树中删除99。由于那片树叶只有两项而它的邻居已经是最小值3项了，因此我们把这些项合并成有5项的一片新的树叶。结果，它们的父节点只有两个儿子了。这时该父节点可以从它的邻节点领养，因为邻节点有4个儿子。领养的结果使得双方都有3个儿子，结果如图4-63所示。

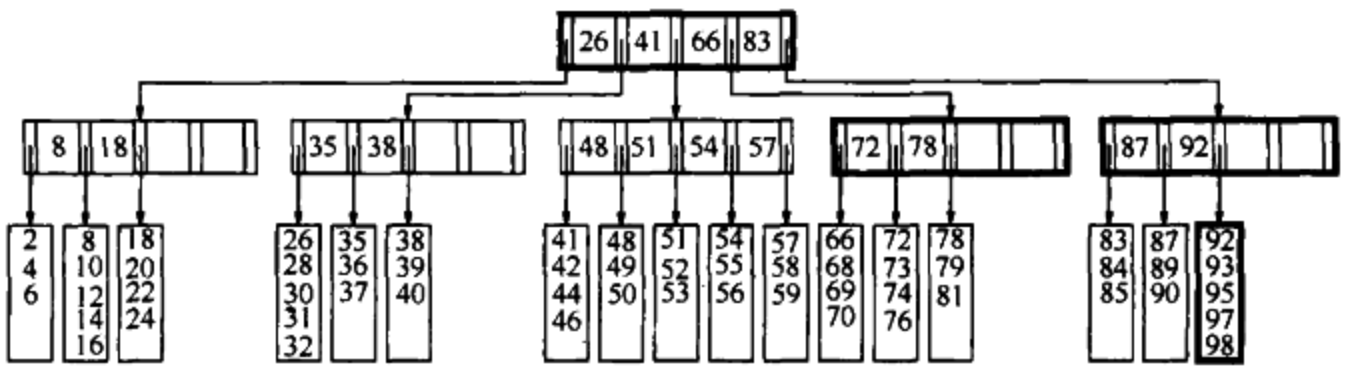

图4-63　在从图4-62的B树中删除99后的B树

4.8　标准库中的集合与映射

在第3章中讨论过的List容器即ArrayList和LinkedList用于查找效率很低。因此，Collections API提供了两个附加容器Set和Map，它们对诸如插入、删除和查找等基本操作提供有效的实现。