个数可以有 1 的误差(例如，根和下一层可以存放在主存中，使得经过长时间运行后磁盘访问将只对第 3 层或更深层是需要的)。

剩下的问题是如何向 B 树添加项和从 B 树删除项；下面将概述所涉及的想法。注意，许多论题以前见到过。

我们首先考查插入。设想要把 57 插入到图 4-59 的 B 树中。沿树向下查找揭示出它不在树中。此时我们把它作为第 5 项添加到树叶中。注意我们可能要为此重新组织该树叶上的所有数据。然而，与磁盘访问相比(在这种情况下它还包含一次磁盘写)，这项操作的开销可以忽略不计。

当然，这是相对简单的，因为该树叶还没有被装满。设现在要插入 55。图 4-60 显示一个问题：55 想要插入其中的那片树叶已经满了。不过解法却不复杂：由于我们现在有 $L+1$ 项，因此把它们分成两片树叶，这两片树叶保证都有所需要的记录的最小个数。我们形成两片树叶，每叶 3 项。写这两片树叶需要 2 次磁盘访问，更新它们的父节点需要第 3 次磁盘访问。注意，在父节点中关键字和分支均发生了变化，但是这种变化是以容易计算的受控的方式处理的。最后得到的 B 树在图 4-61 中给出。虽然分裂节点是耗时的，因为它至少需要 2 次附加的磁盘写，但它相对很少发生。例如，如果 L 是 32，那么当节点被分裂时，具有 16 和 17 项的两片树叶分别被建立。对于有 17 项的那片树叶，我们可以再执行 15 次插入而不用另外的分裂。换句话说，对于每次分裂，大致存在 $L/2$ 次非分裂的插入。

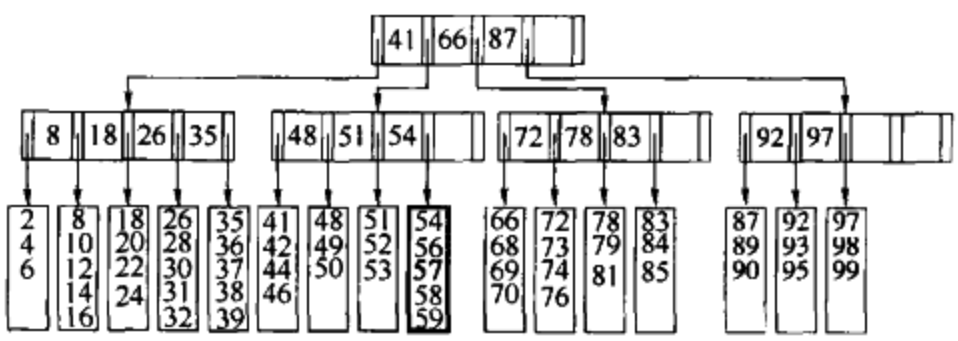

图 4-60 将 57 插入到图 4-59 的树中后的 B 树

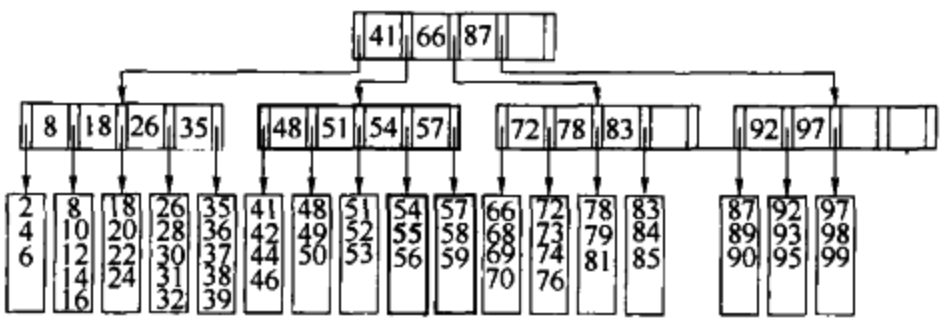

图 4-61 将 55 插入到图 4-60 的 B 树中引起分裂成两片树叶

前面例子中的节点分裂之所以行得通是因为其父节点的儿子个数尚未满员。可是，如果满员了又会怎样呢？例如，假设我们想要把 40 插入到图 4-61 的 B 树中。此时必须把包含关键字 35 到 39 而现在又要包含 40 的树叶分裂成 2 片树叶。但是这将使父节点有 6 个儿子，可是它只能有 5 个儿子。因此，解法就要分裂这个父节点。结果在图 4-62 中给出。当父节点被分裂时，必须更新那些关键字以及还有父节点的父亲的值，这样就招致额外的两次磁盘写(从而这次插入花费 5 次磁盘写)。然而，虽然由于有大量的情况要考虑而使得程序确实不那么简单，但是这些关键字还是以受控的方式变化。

正如这里的情形所示，当一个非叶节点分裂时，它的父节点得到了一个儿子。如果父节点的儿子个数已经达到规定的限度怎么办呢？在这种情况下，继续沿树向上分裂节点直到找到一个不需要再分裂的父节点，或者到达树根。如果分裂树根，那么我们就得到两个树根。显然这是不