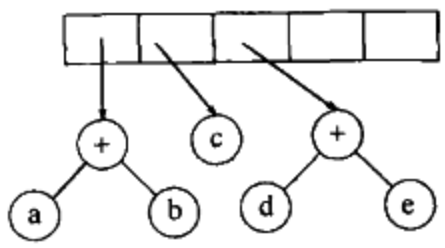
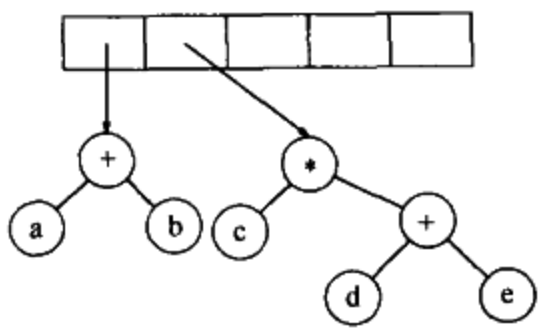
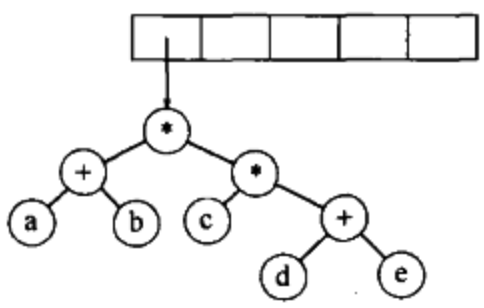
接下来读入‘+’号，因此两棵树合并。



继续进行，读入‘*’号，因此，我们弹出两棵树并形成一棵新的树，‘*’号是它的根。



最后，读入最后一个符号，两棵树合并，而最后的树被留在栈中。



4.3 查找树 ADT——二叉查找树

二叉树的一个重要的应用是它们在查找中的使用。假设树中的每个节点存储一项数据。在我们的例子中，虽然任意复杂的项在 Java 中都容易处理，但为简单起见还是假设它们是整数。还将假设所有的项都是互异的，以后再处理有重复元的情况。

使二叉树成为二叉查找树的性质是，对于树中的每个节点 X，它的左子树中所有项的值小于 X 中的项，而它的右子树中所有项的值大于 X 中的项。注意，这意味着该树所有的元素可以用某种一致的方式排序。在图 4-15 中，左边的树是二叉查找树，但右边的树则不是。右边的树在其项是 6 的节点(该节点正好是根节点)的左子树中，有一个节点的项是 7。

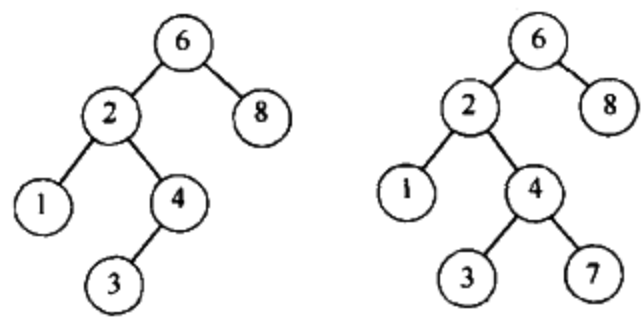


图 4-15 两棵二叉树(只有左边的树是查找树)

现在给出通常对二叉查找树进行的操作的简要描述。注意，由于树的递归定义，通常是递归地编写这些操作的例程。因为二叉查找树的平均深度是 $O(\log N)$，所以一般不必担心栈空间被用尽。

二叉查找树要求所有的项都能够排序。要写出一个一般的类，我们需要提供一个 interface (接口)来表示这个性质。这个接口就是 Comparable，第 1 章曾经描述过。该接口告诉我们，树中的两项总可以使用 compareTo 方法进行比较。由此，我们能够确定所有其他可能的关系。特别是我们不使用 equals 方法，而是根据两项相等当且仅当 compareTo 方法返回 0 来判断相等。另一种方法是使用一个函数对象，将在 4.3.1 节中描述。图 4-16 还指出，BinaryNode 类象链表类中的节