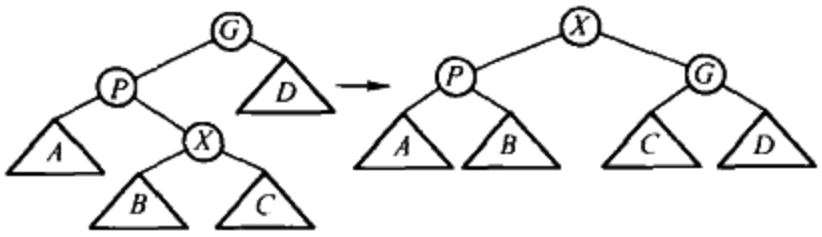


图 4-44　之字形(zig-zag)情形

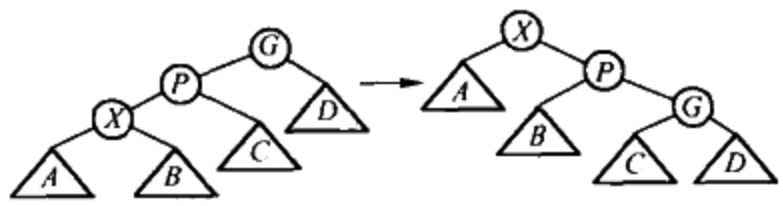
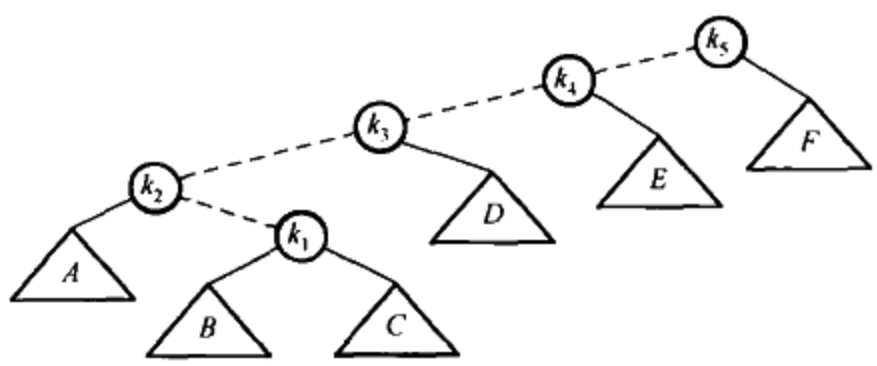
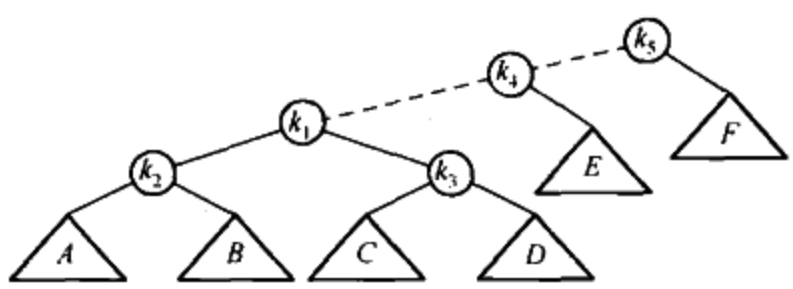


图 4-45　一字形(zig-zig)情形

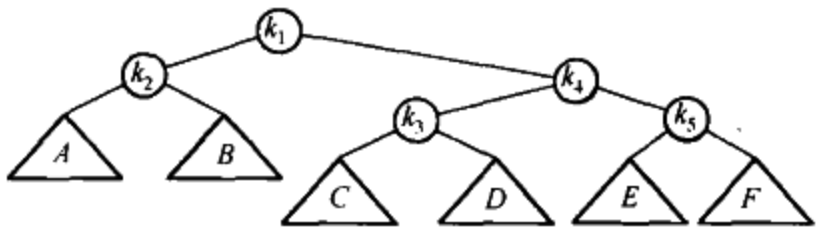
作为例子，考虑来自最后的例子中的树，对 k_1 执行一次 contains：



展开的第一步是在 k_1，显然是一个之字形，因此我们用 k_1、k_2 和 k_3 执行一次标准的 AVL 双旋转。得到如下的树。



在 k_1 的下一步展开是一个一字形，因此我们用 k_1、k_4 和 k_5 做一字形旋转，得到最后的树。



虽然从一些小例子很难看出来，但是展开操作不仅将访问的节点移动到根处，而且还把访问路径上的大部分节点的深度大致减少一半(某些浅的节点最多向下推后两层)。

为了看出展开与简单旋转的差别，再来考虑将 1,2,3,…,N 各项插入到初始空树中去的效果。如前所述可知共花费 $O(N)$时间，并产生与一些简单旋转结果相同的树。图 4-46 指出在项为 1 的节点展开的结果。区别在于，在对项为 1 的节点访问(花费 $N-1$ 个单元的时间)之后，对项为 2 的节点的访问只花费 $N/2$ 个时间单元而不是 $N-2$ 个时间单元；不存在像以前那么深层的节点。

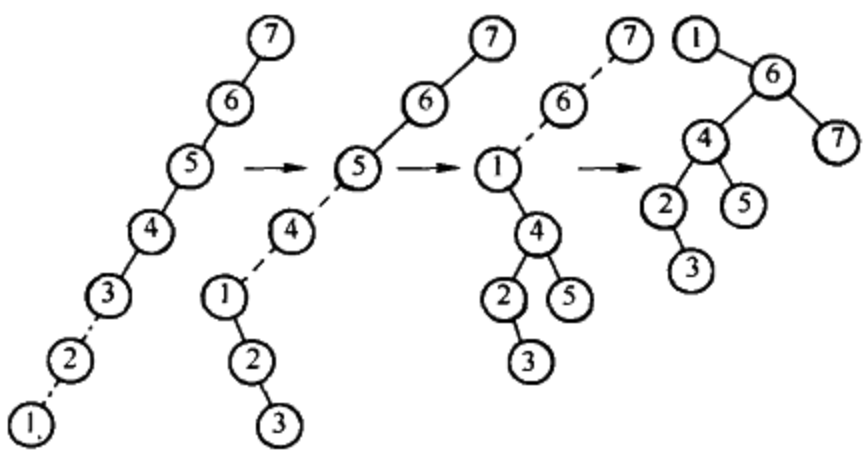


图 4-46　在节点 1 展开的结果