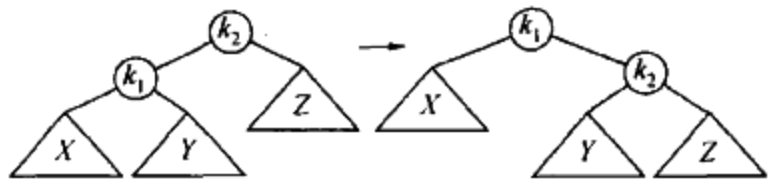

图 4-40 单旋转

```
1     /**
2      * Rotate binary tree node with left child.
3      * For AVL trees, this is a single rotation for case 1.
4      * Update heights, then return new root.
5      */
6     private AvlNode<AnyType> rotateWithLeftChild( AvlNode<AnyType> k2 )
7     {
8         AvlNode<AnyType> k1 = k2.left;
9         k2.left = k1.right;
10        k1.right = k2;
11        k2.height = Math.max( height( k2.left ), height( k2.right )) + 1;
12        k1.height = Math.max( height( k1.left ), k2.height ) + 1;
13        return k1;
14    }
```

图 4-41 执行单旋转的例程

我们要编写的最后一个方法将完成图 4-42 所描述的双旋转，其程序由图 4-43 表出。

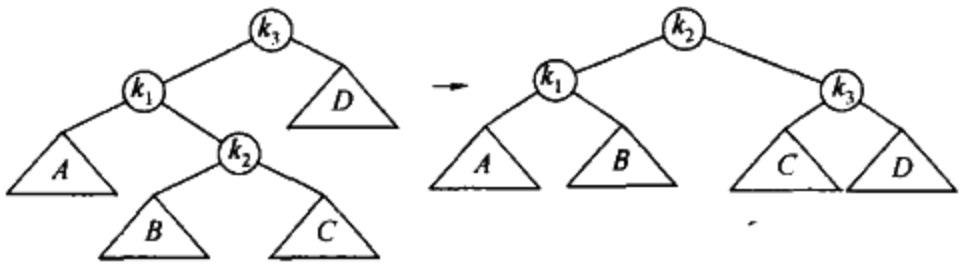

图 4-42 双旋转

```
1     /**
2      * Double rotate binary tree node: first left child
3      * with its right child; then node k3 with new left child.
4      * For AVL trees, this is a double rotation for case 2.
5      * Update heights, then return new root.
6      */
7     private AvlNode<AnyType> doubleWithLeftChild( AvlNode<AnyType> k3)
8     {
9         k3.left = rotateWithRightChild( k3.left );
10        return rotateWithLeftChild( k3 );
11    }
```

图 4-43 执行双旋转的例程

对 AVL 树的删除多少要比插入复杂，我们把它留作练习。如果删除操作相对较少，那么懒惰删除恐怕是最好的方式。

4.5 伸展树

现在我们描述一种相对简单的数据结构，叫做**伸展树**(splay tree)，它保证从空树开始连续 M