4.16 重做二叉查找树类以实现懒惰删除。仔细注意这将影响所有的例程。特别具有挑战性的是 findMin 和 findMax，它们现在必须递归地完成。

*4.17 证明，随机二叉查找树的深度(最深的节点的深度)平均为 $O(\log N)$。

4.18 *a. 给出高度为 h 的 AVL 树的节点的最少个数的精确表达式。

b. 高度为 15 的 AVL 树中节点的最小个数是多少?

4.19 指出将 2，1，4，5，9，3，6，7 插入到初始空 AVL 树后的结果。

*4.20 依次将关键字 $1,2,\cdots,2^k-1$ 插入到一棵初始空 AVL 树中。证明所得到的树是理想平衡(perfectly balanced)的。

4.21 写出实现 AVL 单旋转和双旋转的其余的过程。

4.22 设计一个线性时间算法，该算法检验 AVL 树中的高度信息是否被正确保留并且平衡性质是否成立。

4.23 写出向 AVL 树进行插入的非递归方法。

*4.24 如何能够在 AVL 树中实现(非懒惰)删除?

4.25 a. 为了存储一棵 N－节点的 AVL 树中一个节点的高度，每个节点需要多少比特(bit)?

b. 使 8－比特高度计数器溢出的最小 AVL 树是什么?

4.26 写出执行双旋转的方法，其效率要超过做两个单旋转。

4.27 指出依序访问图 4-71 的伸展树中的关键字 3，9，1，5 后的结果。

4.28 指出在前一道练习所得到的伸展树中删除具有关键字 6 的元素后的结果。

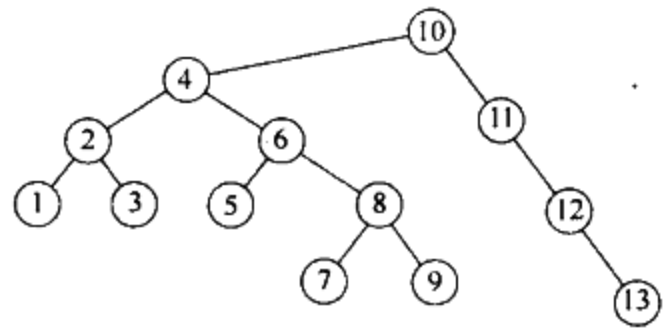

图 4-71 练习 4.27 中的树

4.29 a. 证明如果按顺序访问伸展树中的所有节点，则所得到的树由一连串的左儿子组成。

**b. 证明如果按顺序访问伸展树中的所有节点，则总的访问时间是 $O(N)$，与初始树无关。

4.30 编写一个程序对伸展树执行随机操作。计算所执行的总的旋转次数。与 AVL 树和非平衡二叉查找树相比，其运行时间如何?

4.31 编写一些高效率的方法，只使用对二叉树的根的引用 T，并计算：

a. T 中节点的个数。

b. T 中树叶的片数。

c. T 中满节点的个数。

4.32 设计一个递归的线性算法，该算法测试一棵二叉树是否在每一个节点都满足查找树的序的性质。

4.33 编写一个递归方法，该方法使用对树 T 的根节点的引用而返回从 T 删除所有树叶所得到的树的根节点的引用。

4.34 写出生成一棵 N－节点随机二叉查找树的方法，该树具有从 1 直到 N 的不同的关键字。你所编写的例程的运行时间是多少?

4.35 写出生成具有最少节点高度为 h 的 AVL 树的方法，该方法的运行时间是多少?

4.36 编写一个方法，使它生成一棵具有关键字从 1 直到 $2^{h+1}-1$ 且高为 h 的理想平衡二叉查找树(perfectly balanced binary search tree)。该方法运行时间是多少?