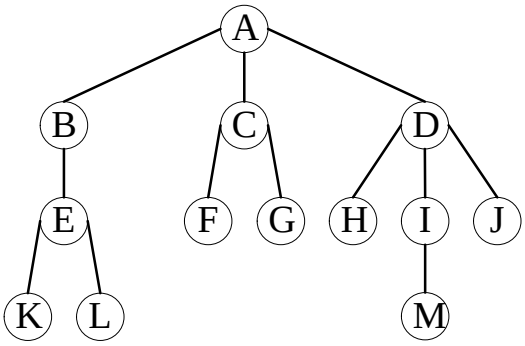
第 5 章 树与二叉树

5.1 树类型

1. 概念

树、结点、边、空树、子树、根、叶子（终端结点）、非叶子（非终端结点、分支结点）、双亲、孩子、祖先、后代（子孙）、兄弟、规模（树的结点个数）、结点的度（一个结点的孩子个数（或非空子树棵数）称为该结点的度）、树的度（规定空树的度为 0）、结点的层数（有两套体系）、树的高度（深度）（有两套体系。规定空树的高度为 0（-1））、有序树、无序树、森林（果园）



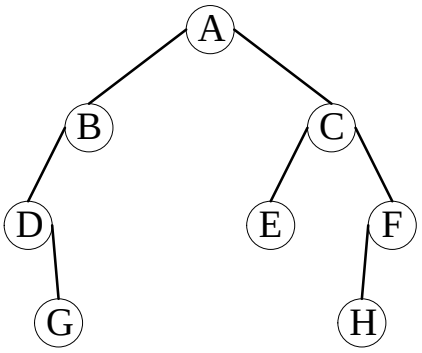
2. 树形结构

在非空树中，根没有直接前驱（双亲），其他每个结点都有且只有一个直接前驱，叶子没有直接后继（孩子），其他每个结点都有一个或多个直接后继。

5.2 二叉树

1. 概念

二叉树（二叉树的每个结点都有两棵子（二叉）树，左子树和右子树，子树可以为空。二叉树的每个结点最多有两个孩子，左孩子和右孩子。若二叉树的某个结点有一个孩子，则该孩子要明确表明是左孩子还是右孩子。若树的某个结点有一个孩子，则该孩子一定是第一个孩子。所以，二叉树不是树的特例）、左子树、右子树、左孩子、右孩子、空二叉树、满二叉树（每一层结点都是满的二叉树）、完全二叉树（除最后一层外，其余每一层结点都是满的，若最后一层结点不是满的，则这些结点都集中在左边的二叉树）、严格二叉树（结点要么没有孩子要么有两个孩子的非空二叉树）、单分支（二叉）树



2. 基本形态

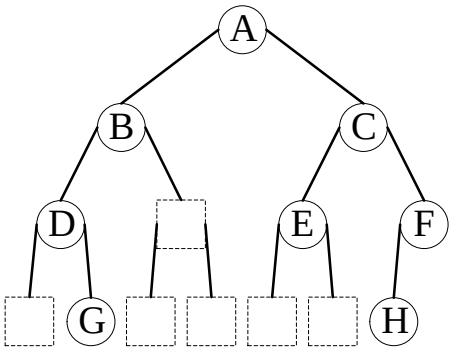
p.150 图 5.4

3. 性质

pp.151-153 性质 1-5

4. 顺序实现

根存储在 bt[0]中，bt[i]中存储的结点的左、右孩子（如果有的话）分别存储在 bt[2i+1],bt[2i+2]中。



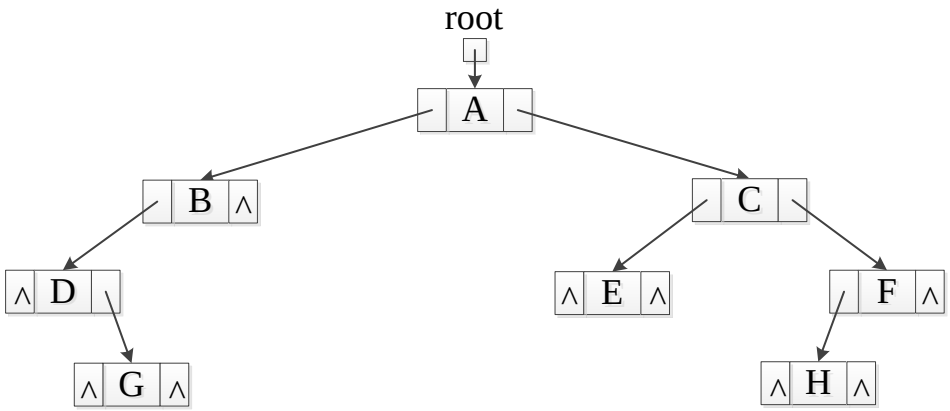
5. 链式实现

（1）二叉链表

①示意图

a.空

b.非空



②结点类定义

p.154

③二叉链表类定义

pp.155-156

（2）三叉链表

（3）线索链表

5.3 二叉树的遍历

1. 遍历及其算法

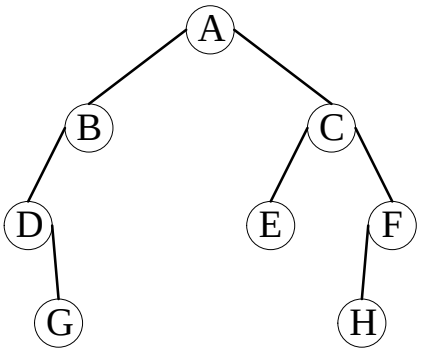
（1）遍历

①先序遍历（前序遍历）：根左右

②中序遍历（对称序遍历）：左根右

③后序遍历：左右根

④层序遍历：一层一层从上往下，每一层从左往右



先序序列：　　根→ABDGCEFH←最右边的叶子

中序序列：最左边的结点→DGBAECHF←最右边的结点

后序序列：最左边的叶子→GDBEHFCA←根

层序序列：　　根→ABCDEFGH

（2）先序、中序和后序遍历的递归算法

p.158 算法 5.1-5.3

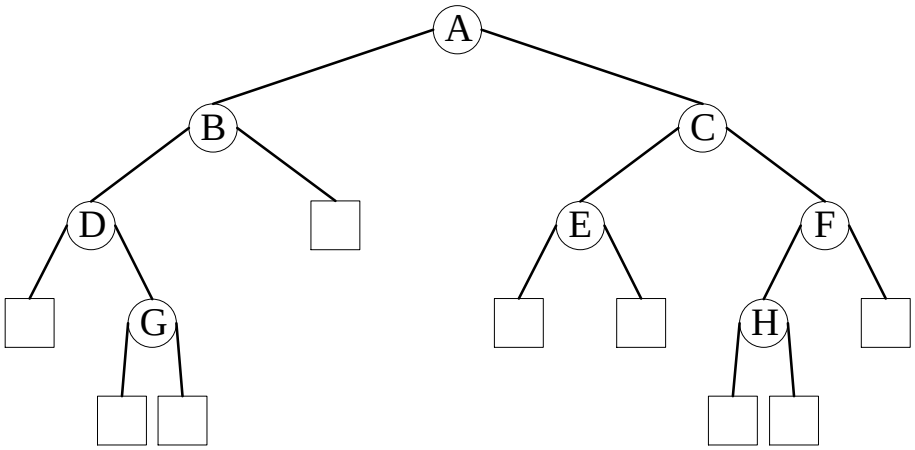
（3）先序、中序和后序遍历的非递归算法及层序遍历的算法

pp.159-162 算法 5.4-5.7

2. 遍历的应用

3. 构造二叉树

定义 1. 将空二叉树用扩展结点代替，将非空二叉树的结点的空子树都用扩展结点代替，得到与二叉树对应的扩展二叉树。



定理 1. 由与一棵二叉树对应的扩展二叉树的先序序列可以唯一确定该二叉树。

定理 2. 由与一棵二叉树对应的扩展二叉树的后序序列可以唯一确定该二叉树。

定理 3. 由与一棵二叉树对应的扩展二叉树的层序序列可以唯一确定该二叉树。

例 1. 已知一棵二叉树的先序和中序序列分别为 ABCDEFG 和 CBEDAFG。画出该二叉树。

定理 4. 由一棵二叉树的先序和中序序列可以唯一确定该二叉树。

定理 5. 由一棵二叉树的中序和后序序列可以唯一确定该二叉树。

定理 6. 由一棵二叉树的中序和层序序列可以唯一确定该二叉树。

定理 7. 由一棵严格二叉树的先序和后序序列可以唯一确定该严格二叉树。

定理 8. 由一棵严格二叉树的先序和层序序列可以唯一确定该严格二叉树。

定理 9. 由一棵严格二叉树的后序和层序序列可以唯一确定该严格二叉树。

5.4 哈夫曼树及哈夫曼编码

1. 哈夫曼树

（1）概念

路径长度、树的路径长度、结点的带权路径长度、树的带权路径长度、哈夫曼树（最优二叉树）

（2）哈夫曼算法

p.175

（3）说明

在合并两棵根的权值最小的二叉树时，将哪棵作为新结点的左子树，哪棵作为新结点的右子树是无所谓的。通常将根的权值最小的那棵作为新结点的左子树，根的权值次最小的那棵作为新结点的右子树，如果这两棵的根的权值一样的最小，则将先存在的那棵作为新结点的左子树，后存在的那棵作为新结点的右子树。

2. 哈夫曼编码

（1）概念

前缀编码、哈夫曼编码

（2）说明

对于结点下面的两条边，只要让其中一条表示字符’0’，另一条表示字符’1’即可，至于让哪条表示字符’0’，哪条表示字符’1’是无所谓的。通常让左下方那条边表示字符’0’，右下方那条边表示字符’1’。

哈夫曼编码不是唯一的，但都是最优的。

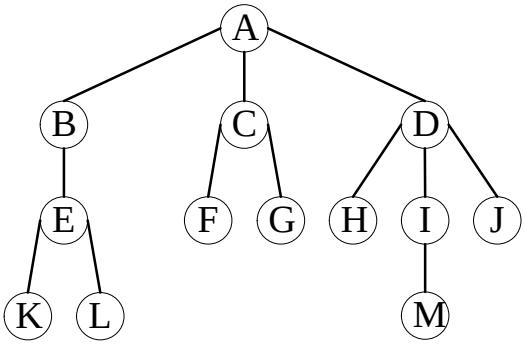
5.5 树与森林

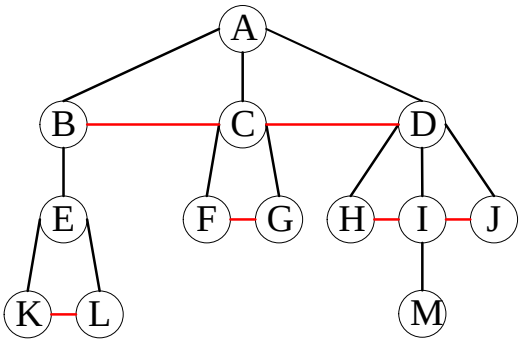
1. 树转化成二叉树

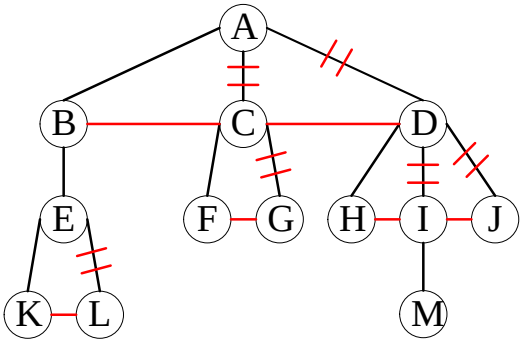
（1）相邻的兄弟之间加上一条边。

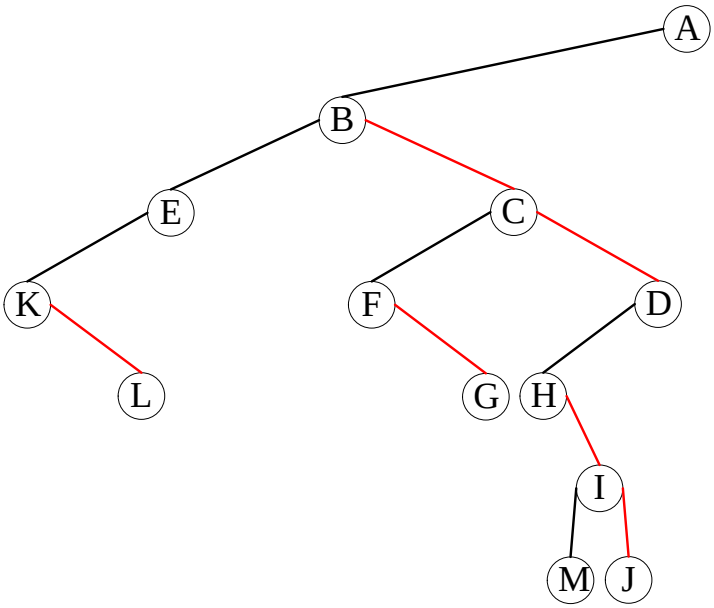
（2）对每个非叶子，只保留它与第一个孩子之间的边，去除它与其他孩子之间的边。

（3）以根为轴心，顺时针转动约 45° 。









树	二叉树
第一个孩子	左孩子
下一个兄弟	右孩子

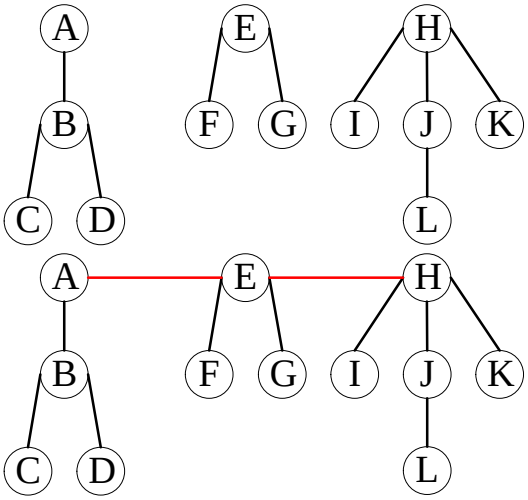
与非空树对应的二叉树的右子树一定为空。

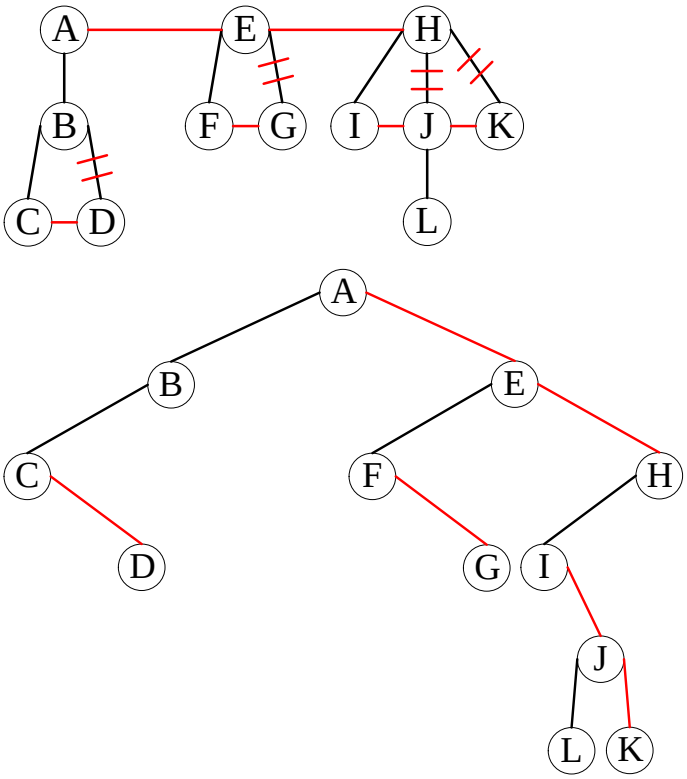
2. 森林转化成二叉树

（1）相邻的树的根之间加上一条边。

（2）每棵树转化成二叉树。

（3）以第一棵树的根为轴心，顺时针转动约 45° 。





森林与二叉树一一对应。

3. 树的链式实现

（1）双亲表示法

双亲链表（静态实现）

（2）孩子表示法

①多重链表

若结点同构，则浪费空间；若结点异构，则难以实现。

②孩子链表

（3）双亲孩子表示法

双亲孩子链表

（4）孩子兄弟表示法

孩子兄弟链表（即与树对应的二叉树的二叉链表）

4. 森林的链式实现

孩子兄弟链表（即与森林对应的二叉树的二叉链表）

5. 树的遍历

目前有两种不完全相同的定义。

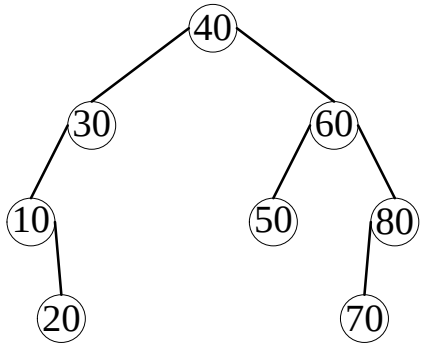
6. 森林的遍历

目前有两种不完全相同的定义。

5.6 二叉排序树（二叉查找树）（pp.292-299）

1. 概念

二叉排序树



中序遍历二叉排序树得到按关键字递增有序排列的记录序列。

定理 10. 由一棵二叉排序树的先序序列可以唯一确定该二叉排序树。

定理 11. 由一棵二叉排序树的后序序列可以唯一确定该二叉排序树。

定理 12. 由一棵二叉排序树的层序序列可以唯一确定该二叉排序树。

2. 操作

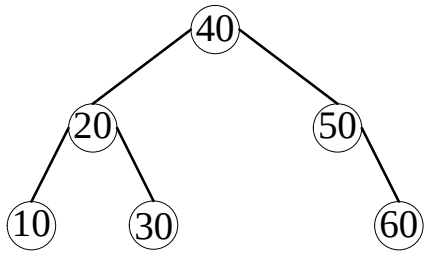
（1）查找

在二叉排序树中查找时，无需遍历二叉排序树。

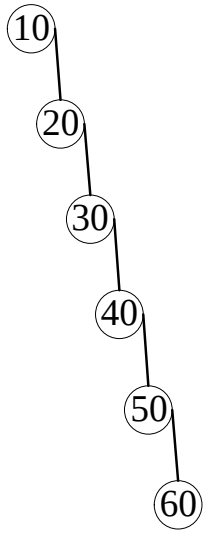
在等查找概率情况下查找成功时的平均查找长度=（Σ结点的层数）/规模，此处根的层数为 1。

（2）插入

例 2. 将关键字为 40,20,50,10,30,60 的记录依次插入一棵初始为空的二叉排序树。

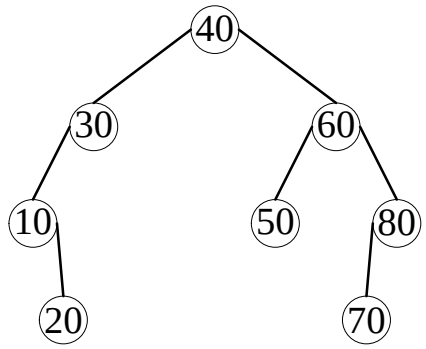


例 3. 将关键字为 10,20,30,40,50,60 的记录依次插入一棵初始为空的二叉排序树。



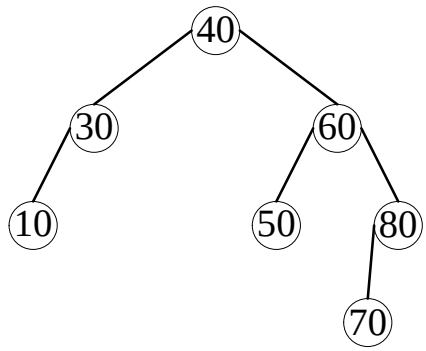
记录的插入次序对二叉排序树的形态有影响。

（3）删除



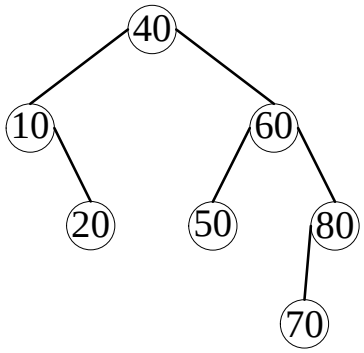
①删除叶子

删除 20。



②删除有一个孩子的结点

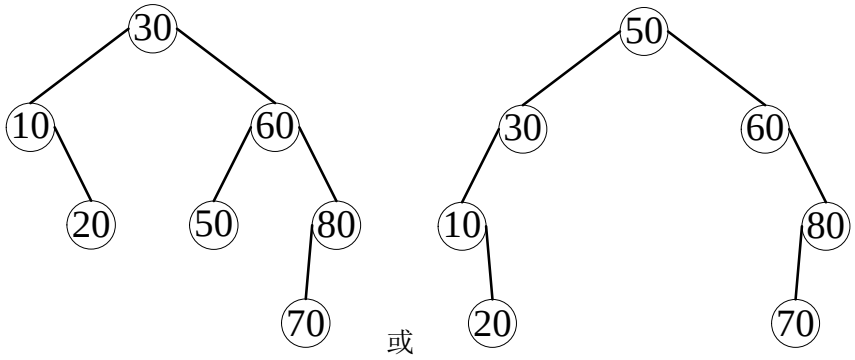
删除 30。



③删除有两个孩子的结点

用待删结点的中序前驱或中序后继取代待删结点。

删除 40。



5.7 AVL 树（pp.299-302）

1. 概念

AVL 树（第一种平衡二叉树，平衡二叉树是一个大类，简称平衡树，其中最有名的还有红黑树）、平衡因子（平衡度）

2. 插入

左单旋、右单旋、先左后右双旋、先右后左双旋

例 4. 将关键字为 Jan，Feb，Mar，Apr，May，June，July，Aug，Sep，Oct，Nov，Dec 的记录依次插入一棵初始为空的 AVL 树。

5.8 堆（pp.255-258）

1. 概念

将一个序列看成一棵完全二叉树的层序序列，若非叶子都小于或等于孩子，则称为小顶堆（小根堆、最小堆），若非叶子都大于或等于孩子，则称为大顶堆（大根堆、最大堆）。

0 1 2 3 4 5 6 7 8

（10,20,30,40,50,60,70,80,90）

（10,40,20,50,70,30,90,80,60）

（10,50,20,70,60,40,30,80,90）

2. 特点

小顶堆中的堆顶元素（第一个元素）一定是最小的，大顶堆中的堆顶元素一定是最大的。

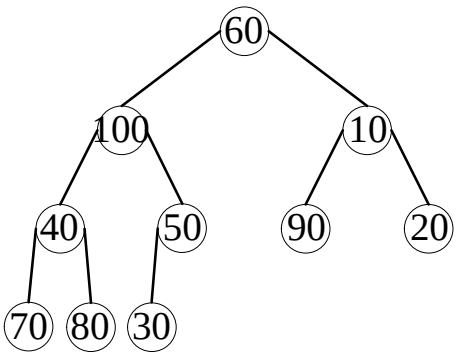
有序序列一定是堆，但堆不一定是有序序列。

3. 建堆

（1）逆层序逐个与后代比较

例 5. 将（60,100,10,40,50,90,20,70,80,30）调整为小顶堆。

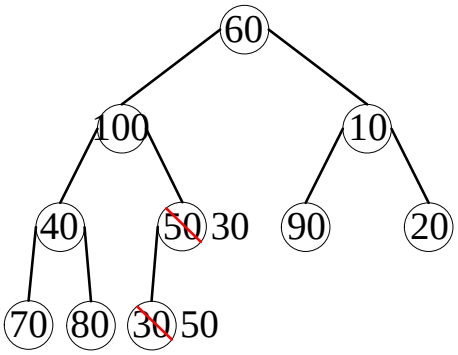
用 60,100,10,40,50,90,20,70,80,30 按层序构造一棵完全二叉树，得到：



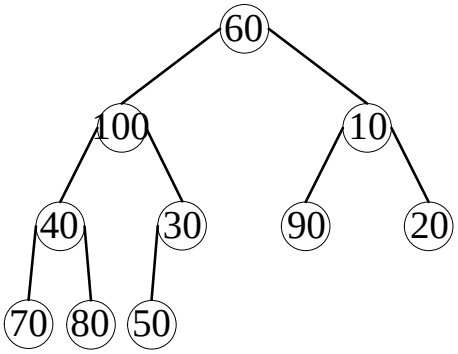
逆层序依次考察各结点。

先依次跳过叶子 30,80,70,20,90。

轮到 50，将 50 和 30 交换：



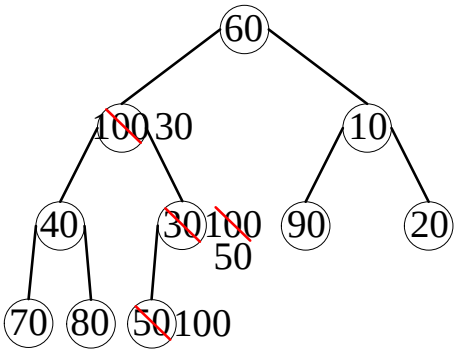
得到：



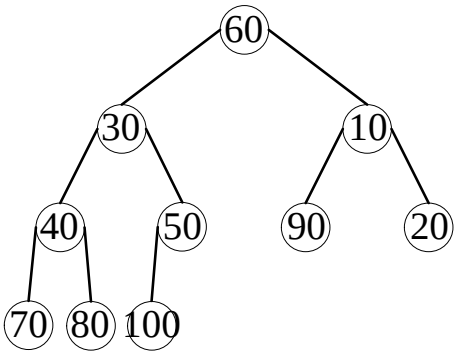
轮到 40，不变。

轮到 10，不变。

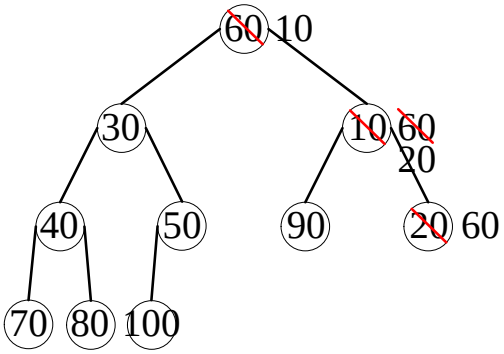
轮到 100，将 100 和 30 交换，再将 100 和 50 交换：



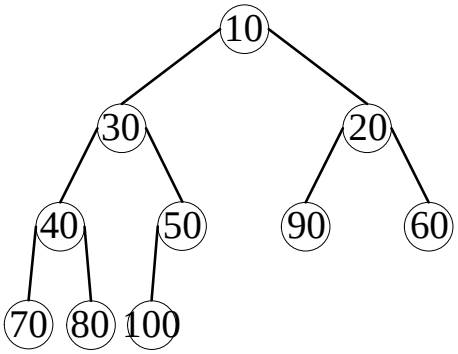
得到：



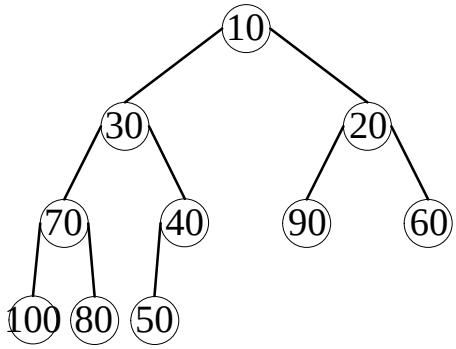
轮到 60，将 60 和 10 交换，再将 60 和 20 交换：



得到：



（2）层序逐个与祖先比较


10
30
20
70
40
90
60
100
80
50