数据结构与算法
第 6 章 树

主讲：赵海燕
北京大学信息科学技术学院
“数据结构与算法”教学组

国家精品课“数据结构与算法”
http://www.jpk.pku.edu.cn/pkujpk/course/sjjg/

张铭，王腾蛟，赵海燕
高等教育出版社，2008. 6, “十一五”国家级规划教材

树的存储

树/树林的顺序存储

问题？

- 如何建立存储内容，怎么恢复成树林或二叉树（树的结构）？

存储方法

- 带右链的先根次序表示法
- 带双标记的先根次序表示法
- 带双标记的层次次序表示法
- 带度数的后根次序表示法

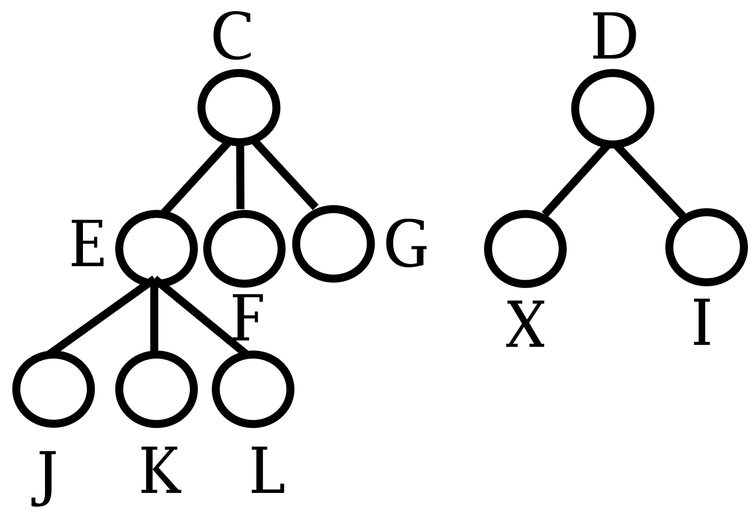
带右链的先根次序表示法

- 树的先根次序

  CEJKLFGDXI

  给出了树林的一部分信息：

  1. 以任何结点为根的子树的所有结点都直接在该结点之后；
  2. 每棵子树的所有结点都聚集在一起，中间不会插入任何别的结点；
  3. 任何一个分支结点后面跟的都是其第一个子结点

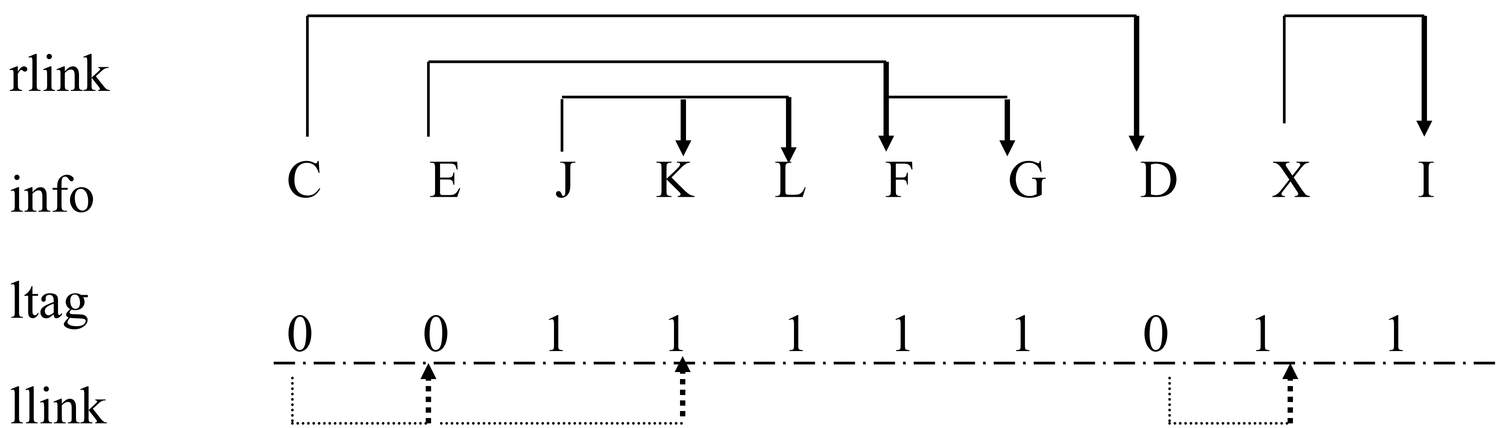


仅有这些信息不足以确定一个树（林）的结构，需要**附加别的**信息

带右链的先根次序表示法

- 每个结点除本身数据外，附加两个表示结构的字段
  - 静态指针 rlink 指向下一个兄弟结点/对应二叉树的右子
  - ltag有子结点（对应二叉树有左子）则为0, 没有则为1

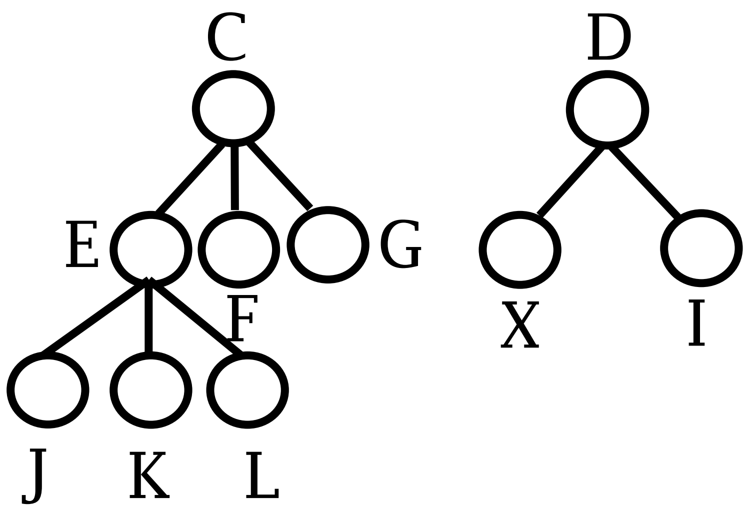
ltag	info	rlink

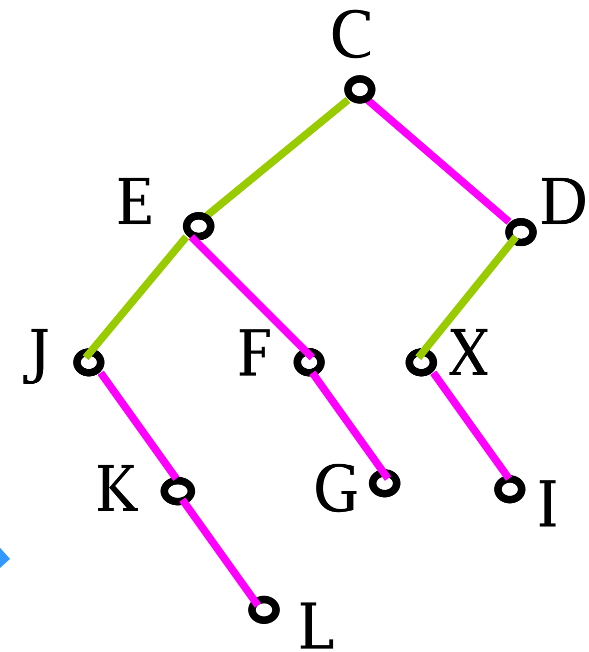


带右链的先根次序表示法

- 与llink-rlink法相比，用ltag代替llink，占用较少的存储单元，且不丢失信息，从结点的次序和 ltag 的值完全可推知llink
  - ltag为 0 的结点有左子结点，llink指向存储区中该结点顺序的下一个结点
  - ltag为 1 的结点没有左子结点，llink为空

带右链的先根次序表示法

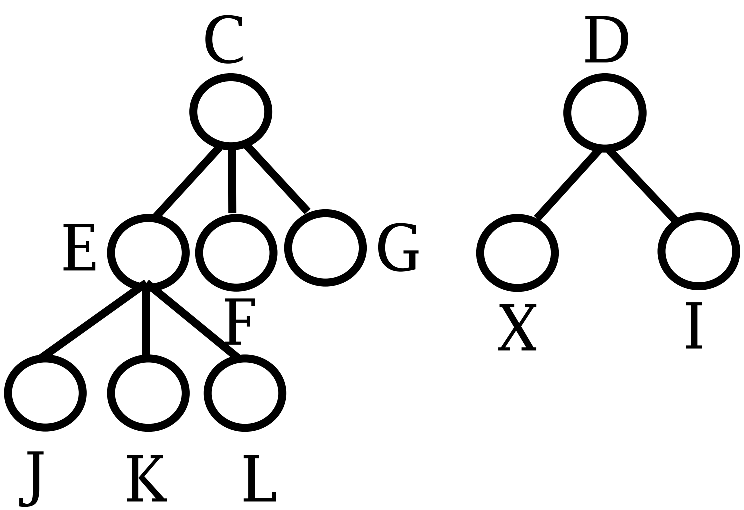


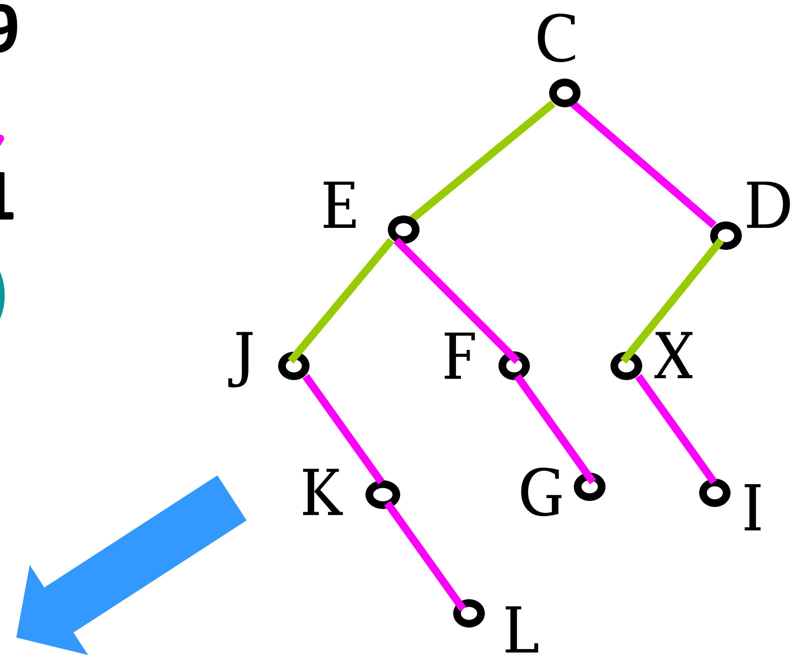


下标	0	1	2	3	4	5	6	7	8	9
rlink	7	5	3	4	-1	6	-1	-1	9	-1
info	C	E	J	K	L	F	G	D	X	I
ltag	0	0	1	1	1	1	1	0	1	1

带右链的先根次序表示法

下标	0	1	2	3	4	5	6	7	8	9
rlink	7	5	3	4	-1	6	-1	-1	9	-1
info	C	E	J	K	L	F	G	D	X	I
ltag	0	0	1	1	1	1	1	0	1	1



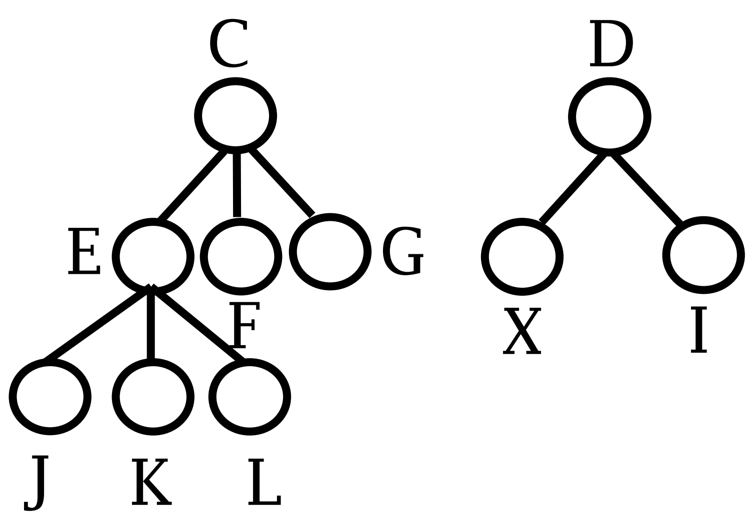


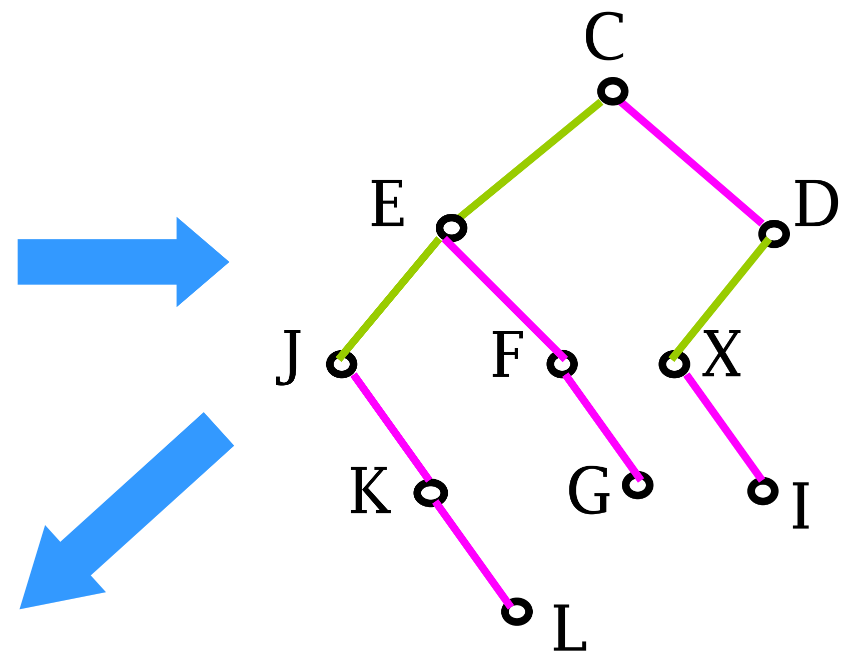
带双标记位的先根次序表示法

- rlink-ltag表示法中 ltag 仍嫌 冗余，因为
  - 有rlink指向的结点，则其序列前驱的 ltag必为 1 (why?)，否则为0
  - 事实上，**rlink并非必需**（信息冗余），代之以取值为 0/1 的rtag 就足以表示出整个森林的结构信息，以降低冗余度
- 带双标记位的先根次序表示法
  - 当结点没有下一个兄弟时 rtag 为1
  - 否则 rtag 为0

ltag	info	rtag

带双标记位的先根次序表示法





下标	0	1	2	3	4	5	6	7	8	9
rtag	0	0	0	0	1	0	1	1	0	1
info	C	E	J	K	L	F	G	D	X	I
ltag	0	0	1	1	1	1	1	0	1	1

带双标记位的先根次序表示法

- 由结点的排列次序和ltag、rtag的值可推知rlink的值
  - 结点 x 的rtag为 1 时，其 rlink显然应为空
  - 结点 x 的rtag为 0 时，其 rlink 应指向结点序列中排在结点 x 的子树结点后面的那个结点 y
- 如何确定这个结点 y 呢？

带双标记位的先根次序表示法

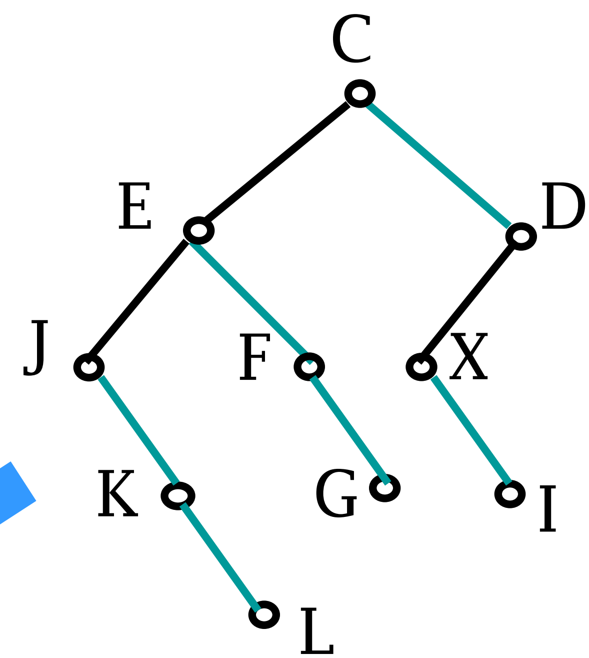
- 可观察到的事实:
  - 任何结点的子树结点在先根序列中都在该结点之后
  - 每棵子树的先根序列的最后一个结点必为叶结点，该结点的ltag 必为1
- 基于此，扫描先根序列并试图确定各结点的rlink的过程中
  - 当遇到一个 rtag=0 的结点 x 时，要继续往后扫描，在结点 x 的子树结点中查找 ltag=1 的、该子树的最后一个结点，序列中排在这个结点后面的那个结点就是结点 x 的 rlink 应该指向的结点 y

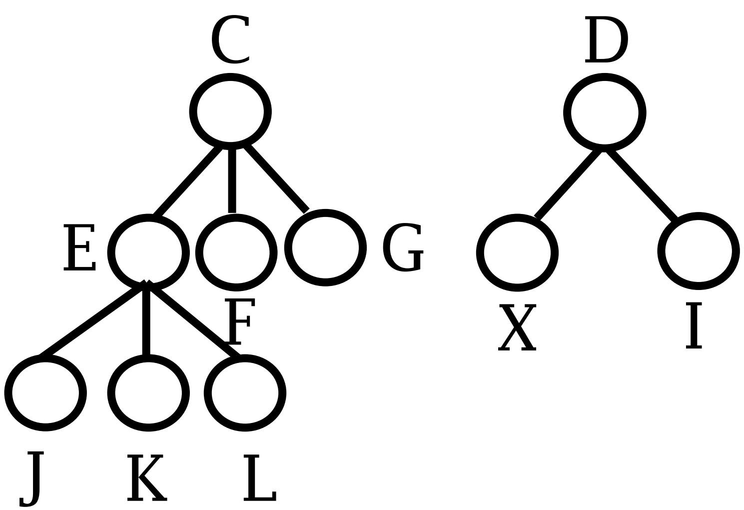
带双标记位的先根次序表示法

- （先根次序表示的）树结构是**嵌套**的，故，在扫描结点 x 的子树结点序列查找最后一个结点的过程中，有可能遇到一棵更小的子树的根结点 x’，其 rtag 也为 0
- 处理过程中用栈来记录 **待配置** rlink **的结点**
  - 扫描到一个rtag = 0 的结点就进栈
  - 扫描到一个ltag = 1 的结点就从栈顶弹出一个元素，并使其右指针指向当前 ltag=1 的元素的下一个元素

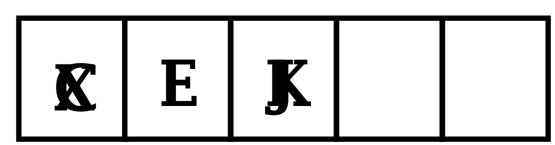
带双标记位的先根次序表示法

下标	0	1	2	3	4	5	6	7	8	9
rtag	0	0	0	0	1	0	1	1	0	1
info	C	E	J	K	L	F	G	D	X	I
ltag	0	0	1	1	1	1	1	0	1	1





stack



带双标记位的先根次序表示法

- 问题
  - 带双标记位的先根次序表示法虽然比带右链的先根次序表示法进一步节省了存储空间，但由于需要额外的处理来推导失去的信息
- 带右链的先根次序表示法采用更为广泛

带双标记位先根次序树的构造

```
template<class T>
class DualTagTreeNode  {                          // 双标记位先根次序树结点类
public:
    T info;                                       // 结点数据信息
    int ltag, rtag;                               // 左、右标记
    DualTagTreeNode();                            // 构造函数
    virtual ~DualTagTreeNode(); };

template <class T>
Tree<T>::Tree(DualTagTreeNode<T> *nodeArray, int count)  {
    // 利用带双标记位的先根次序表示构造左孩子右兄弟表示的树
    using std::stack;                             // 使用STL中的栈
    stack<TreeNode<T>* > aStack;
    TreeNode<T> *pointer = new TreeNode<T>;       // 准备建立根结点
    root = pointer;
```

带双标记位先根次序树的构造

```
        for (int i = 0; i < count-1; i++) {                    // 处理一个结点
            pointer->setValue(nodeArray[i].info);               // 结点赋值
            if (nodeArray[i].rtag == 0)                      // 若右标记为0则将结点压栈
                    aStack.push(pointer);
            else pointer->setSibling(NULL);                  // 右标记为1，则右兄弟指针为空
            TreeNode<T> *temppointer = new TreeNode<T>;      // 预先准备下一个
            if (nodeArray[i].ltag == 0)                      // 左标记为0，则设置子结点
                    pointer->setChild(temppointer);
            else {                                              // 若左标记为1
                    pointer->setChild(NULL);                    // 孩子指针设为空
                    pointer = aStack.top();                     // 取栈顶元素
                    aStack.pop();
                    pointer->setSibling(temppointer);           // 为栈顶设置一个兄弟结点
            }
            pointer = temppointer; }
            pointer->setValue(nodeArray[count-1].info);      // 处理最后一个结点
            pointer->setChild(NULL);  pointer->setSibling(NULL);
        }
}
```

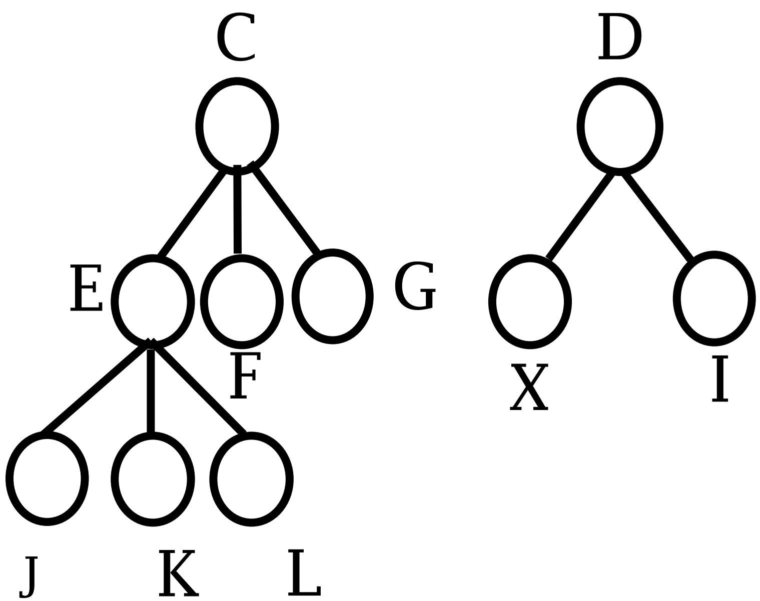
带双标记的层次次序表示法

- 在带双标记的层次次序表示中，结点按 **层次次序顺序** 存储在一片连续的存储单元中，每个结点除包括结点本身数据外，还 **附加两个表示结构**的信息字段

ltag	info	rtag

  - Info 结点的数据
  - 当结点没有左子时标记位 ltag 为1，否则为0
  - 当结点没有右兄时标记位 rtag 为1，否则为0

带双标记的层次次序表示法



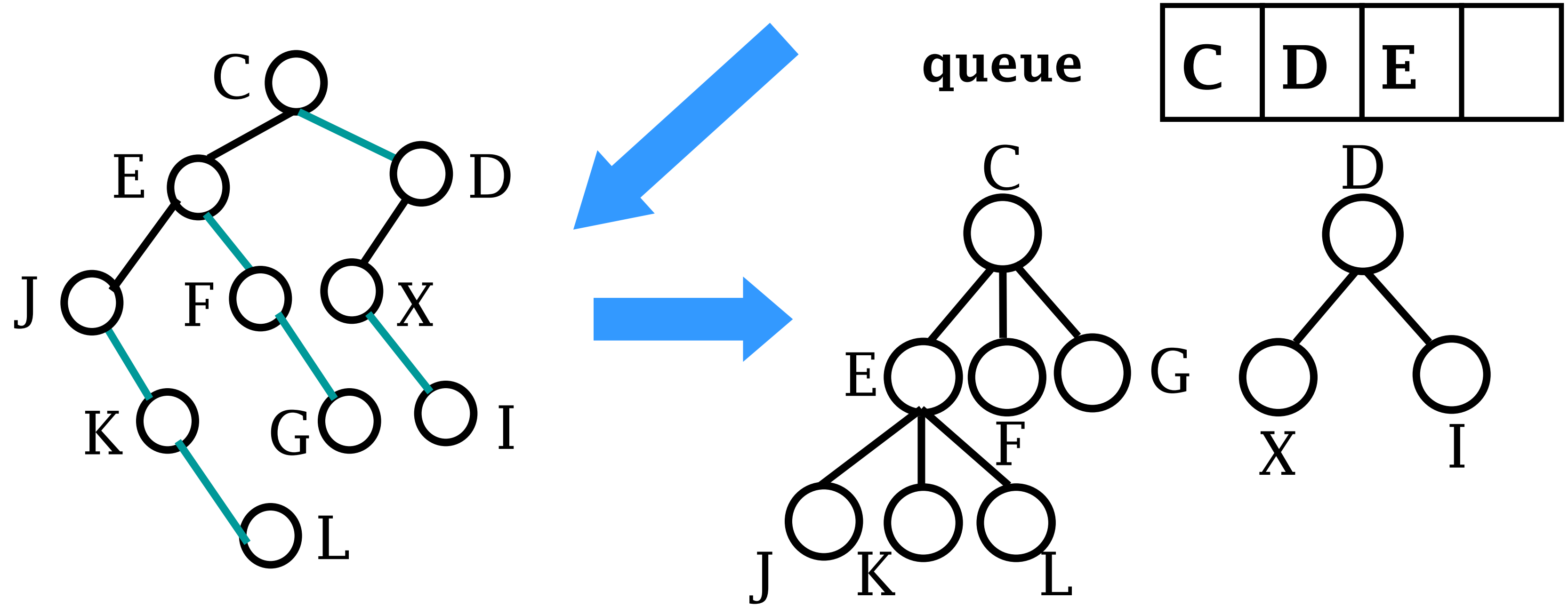
ltag	0	0	1	0	1	0	1	1	1	1
info	C	D	E	F	G	X	I	J	K	L
rtag	0	1	0	0	1	0	1	0	1	1

带双标记的层次次序表示法

- 层次序列的性质
  - 任何结点的子结点都排在该结点的所有兄弟结点后面
  - 任何结点的最后一个兄弟结点的rtag为1
- 由结点的层次序列以及ltag、rtag 两个标志位，可确定树的左子/右兄链表中结点的 llink 和 rlink 值
  - **rlink**：结点的rtag为1，则 rlink 为空；rtag 为0，则其 rlink 指向该结点紧邻的下一个结点
  - **llink**：若结点的ltag值为1，则置其 llink 为空；当结点 **x** 的ltag为0时，其llink应指向结点序列中排在 **x** 的兄弟结点链之后的那个结点 **y**，故 **x** 应进入队列
- 如何构造?
  - 队列

带双标记的层次次序表示法

ltag	0	0	0	1	1	1	1	1	1	1
info	C	D	E	F	G	X	I	J	K	L
rtag	0	1	0	0	1	0	1	0	0	1



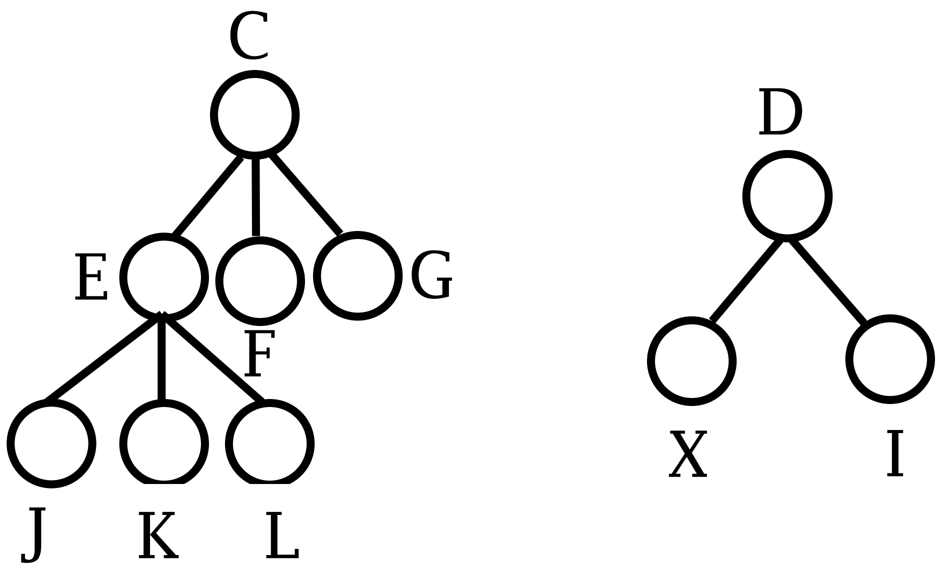
带度数的后根次序表示法

- 在带度数的后根序列表示中，结点按 后根次序顺序 存储在一片连续的存储单元中，结点的形式为

info	degree

  - info 结点的数据
  - degree 结点的度数

带度数的后根次序表示法



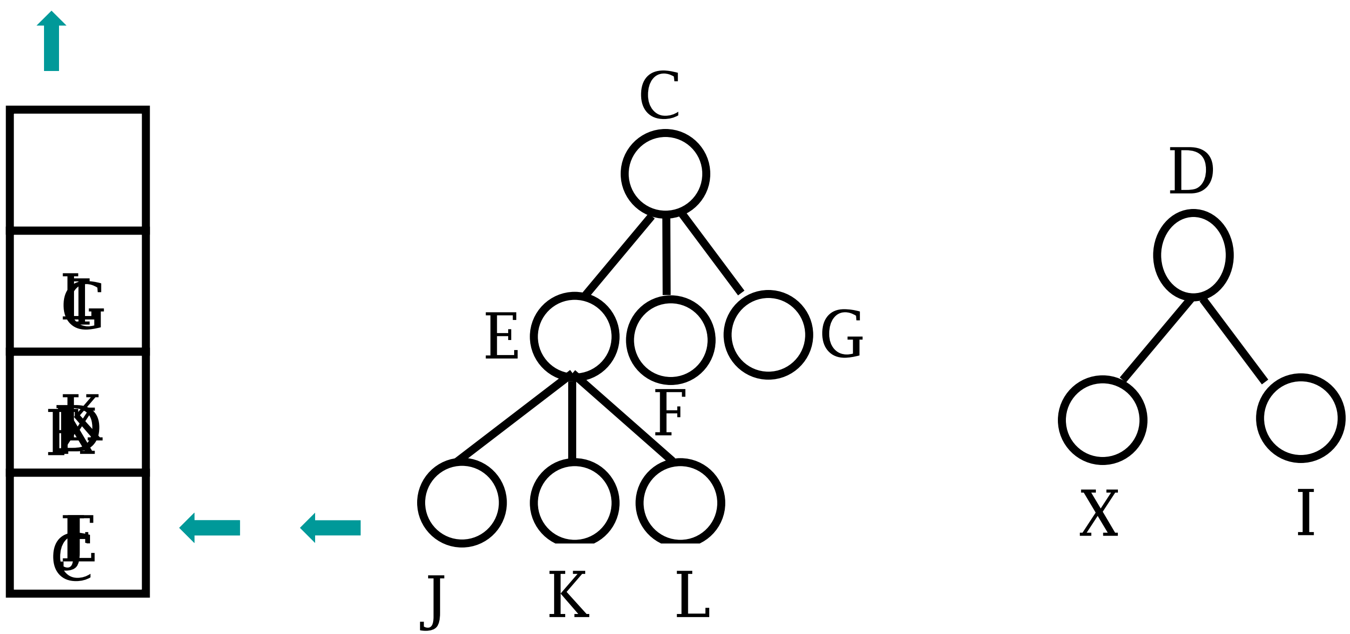
degree	0	0	0	3	0	0	3	0	0	2
info	J	K	L	E	F	G	C	X	I	D

带度数的后根次序表示法

- 虽不包含指针，仍能完全反映树的结构
  - 任何一棵子树的所有结点都聚集在一起，且该子树的根作为 最后一个结点
  - 若某结点的度数为m，则该结点有m个子结点：
    - 最右子树的根（最右子结点）就是其左边的结点（后序前驱）
    - 次最右子结点是以最右子女为根的子树在后根次序序列中的前驱
    - ……

带度数的后根次序表示法

degree	0	0	0	3	0	0	3	0	0	2
info	J	K	L	E	F	G	C	X	I	D



带度数的后根次序表示法

- 从指定的结点 x 开始，向左扫描，分别计算 **结点总数**、**度数总数**，满足

  结点总数 = 度数总数 + 1

  的结点 y，就是该结点的最右子树最后一个结点，array[y..x] 的内容就是以 x 为根的子树的全部结点

任何子树：
结点数 - 边数 = 1

- 可以按照这种办法嵌套地来恢复
  - 栈

带度数的后根次序表示法

B	K	C	A	H	E	J	F	G	D
0	0	1	2	0	1	0	1	0	3

- 例:

找F为根的子树在后根次序中的前驱，从F开始计数:

结点	结点总数	度数总数
F	1	1
J	2	1

满足；结点总数 = 度数总数 +1

故其前的结点 E 即 F为根的子树在后根次序中的前驱

树/树林的顺序存储讨论

- 冗余与效率的问题
- 基于一定的遍历次序
  - 若反复按某个次序遍历树结构，可采用相应的顺序存储方式
- 其他顺序存储?
  - 带度数的先根次序?
  - 带度数的层次次序?

树/树林的顺序存储讨论

- 二叉树的顺序存储?
  - 二叉树与森林对应，但语义不同
    - 带右链的二叉树前序
    - 带左链的二叉树层次次序

K叉树

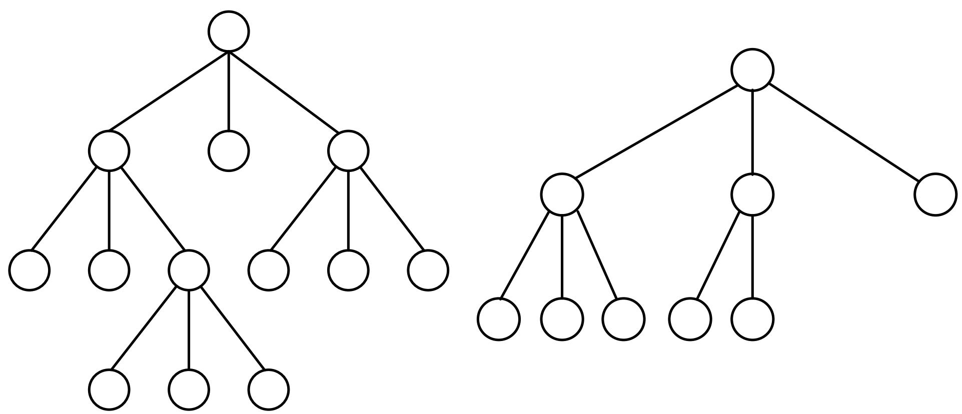
- K叉树T (K-ary Tree) 是有下列性质的有限结点集合：
  - 或为空；
  - 或是由一个根结点R 及 K棵互不相交的 K 叉树构成。亦即，其余结点被划分成T_0，T_1，…，T_{K-1}（$K \geq 1$）个子集，每个子集都是K叉树，使得$T = \{R，T_0，T_1，…，T_{K-1}\}$
- K叉树的各分支结点有 K个子结点
  - 特例：2叉树，PR四分树

K叉树

- 不同于树，K叉树的结点有K个子结点，子结点数目是固定的，因此相对来说容易实现
- 注意当K变大时，空指针的潜在数目会增加，并且叶结点与分支结点在所需空间大小上的差异也会更为显著
  - 当K增加时，对叶结点与分支结点采用不同实现的需要就变得更加迫切

K叉树

- 满 K 叉树和 完全 K 叉树 与满二叉树和完全二叉树是类似的
  - 可以把完全K叉树按编号顺序存储在一个数组中
- 二叉树的许多性质可以推广到K叉树
  - 结点数目、树高、及其之间的关系等等……
- 实际上大多数K叉树的应用采用的是完全K叉树或者满K叉树



（a）满3叉树　　（b）完全3叉树

补充内容

树计数

两类问题

- 树计数（Enumeration of Trees）
  - 具有n个结点、互不相似的树的数目问题
  - 树的相似定义
    - 若两棵有序树 T_1 和 T_2 相似，须满足下列条件之一：
      (1) 同为空树；
      (2) 非空时， T_1和T_2的所有子树分别依次对应相似

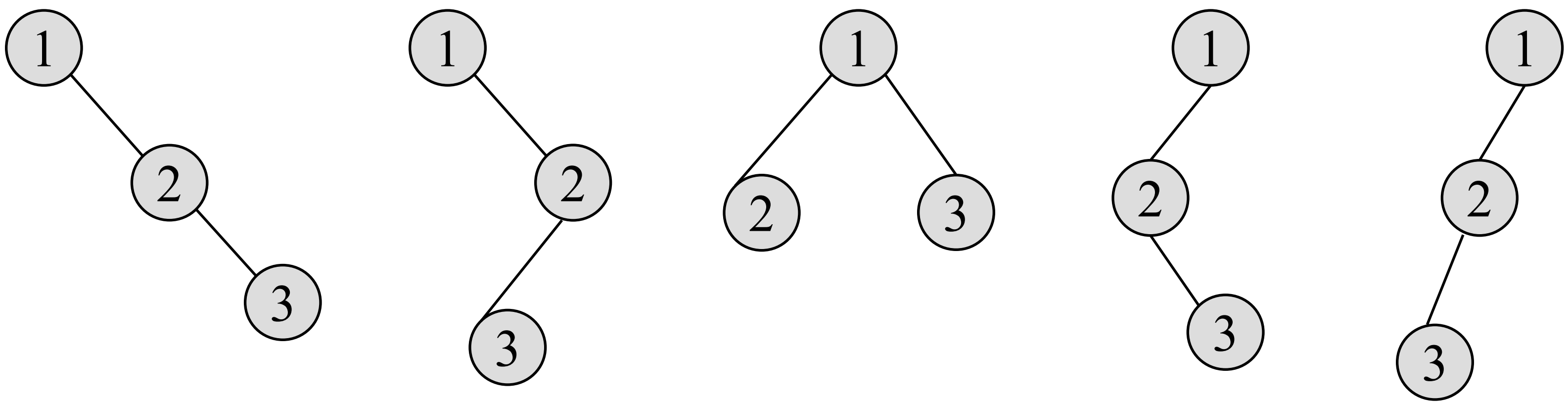
- 栈的出栈序列
  - 当有1，2，…， n共n个数顺序进栈，可以得到多少种不同的出栈序列呢？

栈的出栈序列

- 栈是一种后进先出的数据结构，当某一个输入序列顺序进栈时，可以得到不同的出栈序列
  - E.g.，1, 2, 3 三个数顺序进栈可以得到
    - 123,
    - 132,
    - 213,
    - 231,
    - 321

  五种不同出栈序列

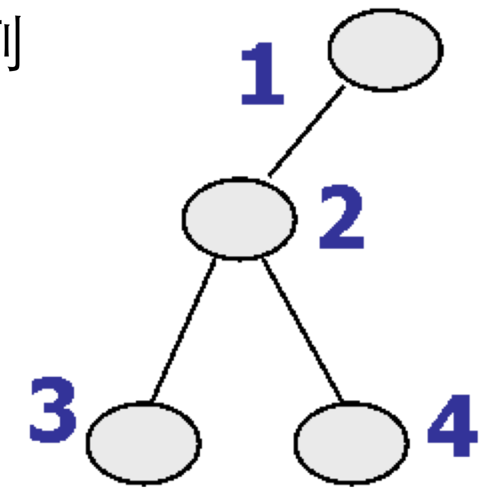
n=3时的二叉树计数



两者之间有什么样的关系？

树计数

- 对于数值为1,2,3,4,...n 的 进栈序列，构造二叉树
  - 例：S1S2S3X3X2S4X4X1 得到二叉树
    这棵二叉树的中序遍历结果即出栈序列



显然所有的合法序列数就是 n个结点，前序遍历为1234...n 的二叉树个数

二叉树计数

1. n个结点的不同形态二叉树的数目

- 前序、中序可以决定一棵二叉树
- 令前序序列为1, 2, ..., n的合法中序所构成的二叉树，中序周游的过程实际上是以1, 2, ..., n为顺序进栈、出栈的过程
- 1, 2, ..., n顺序进栈、出栈所得序列数目为*Catalan数*

$$b_n = c_{2n}^{n} - c_{2n}^{n-1} = \frac{1}{n+1} c_{2n}^{n}$$

方法1：二叉树递推方程

- 一棵具有n(n≥1)个结点的二叉树可看成由一个根结点、一个有i个结点的左子树和一棵有n-i-1个结点的右子树组成（0≤i≤n-1）
  - 左、右子树的不同组合决定整棵树的形态
  - n个结点的二叉树的个数 b(n) = b[i] * b[n-i-1]，可表示为如下的二项式公式如下：

$$b_0 = 1$$

$$b_n = \sum_{i=0}^{n-1} b_i b_{n-i-1}$$

解此递推方程，得

$$b_n = \frac{1}{n+1} C_{2n}^{n}$$

方法2：等概率匹配

- n个S，n个X的序列共有 (2n,n)种可能；
- 从左侧开始扫描这个序列，将S和X配对，其方法是每扫描到一个S或X，从它的右侧开始选取第一个与之匹配的X或S，删除这对SX或XS后再从最左侧开始下一个匹配。
  - 例如：SXSX和SSXX经扫描后都得到两个匹配：SX，SX。
- 对于一个长度为2n的序列，得到n对匹配。其中SX的个数可以为0，1，…，n共n+1种情况
- 根据栈的LIFO特性，只有SX个数为n的那些序列才是合法的。由于等概率分布，故合法序列个数为(2n,n)/(n+1)

方法3：Knuth

- n个S，n个X的序列共有 (2n,n)种可能
- 对于这样的序列从左到右进行扫描，遇到第一个X个数大于S个数的位置 j，把从1到j这个j个位置的所有X换成S、S换成X， 得到一个n+1个S、n-1个X组成的序列
  - 一个由n+1个S、n-1个X组成的序列，可与n-1个S、n+1个X所组成的不合法的序列一一对应
  - 这样的序列共有(2n, n+1)种可能
- 故合法序列个数为

(2n,n) - (2n,n+1) = (2n,n)/(n+1)

方法4：非降路径问题

- 经典组合数学问题
  - 合法序列个数相当于从（0,0）点到（n,n）点不穿过直线y=x且在该直线下侧的非降路径数
    - 沿X轴走一步表示压栈，沿Y轴走一步表示弹栈
    - 接触到直线 y=x 表示该时刻栈为空。可以证明这样的对应是双射
    - 由非降路径问题可知，从（0,0）点到（n,n）点不接触直线y=x且在该直线下侧的非降路径数为(2n,n)/2(2n-1)
  - 从点(0,0)到点 (n+1,n+1) 、除端点外不接触直线 y=x 的且在直线 y=x下面的非降路径数
    - 等于从点(1,0)到点(n+1,n)的可接触不可穿过 y=x-1且在直线 y=x-1下面的非降路径数
    - 等于从点(0,0)到点(n,n)的可接触不可穿过 y=x 且在直线 y=x下面的非降路径数
  - 故合法序列个数为 (2(n+1),n+1)/2(2n+1)

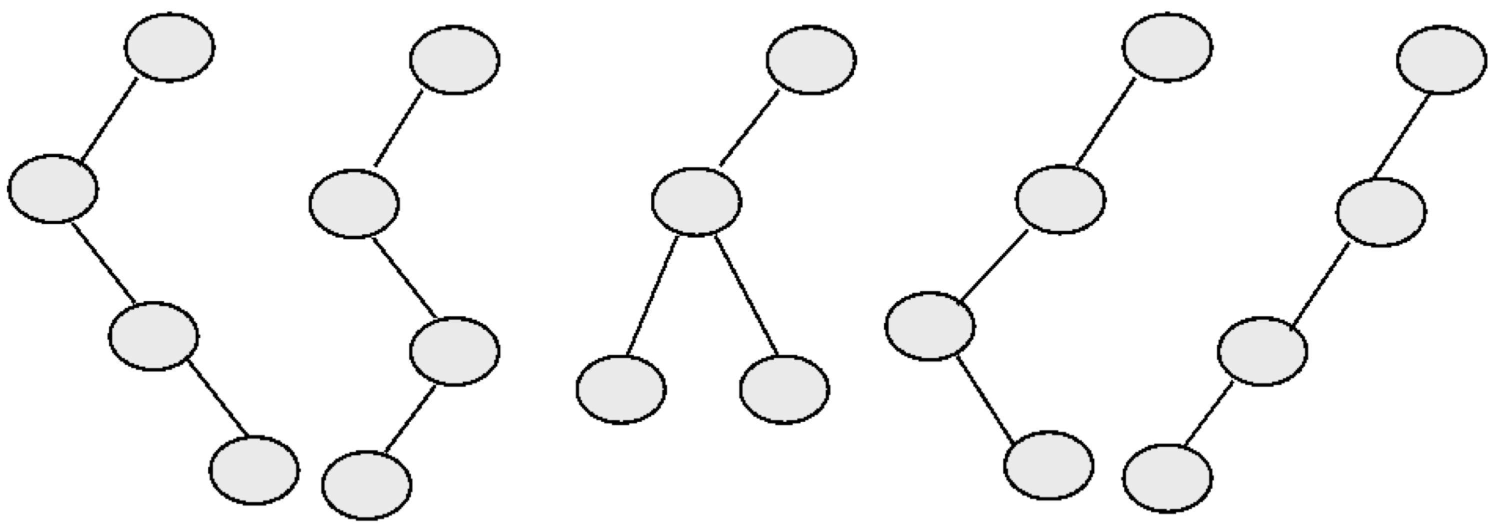
二叉树计数

2. n个结点的不同形态标号二叉树的数目

$$\frac{n!}{n+1}c_{2n}^{n}$$

树计数

具有 4 个结点、根的右子树为空的不同二叉树



树计数

1. n个结点的形态不同有序树的数目

具有n个结点互不相似的树的数目t_n与具有n-1个结点的互不相似的二叉树的数目b_{n-1}相同

- 由于一棵树可转换成一棵没有右子树的二叉树，反之亦然

$$t_n = b_{n-1} = \frac{1}{n} c_{2(n-1)}^{n-1}$$

树计数

2. n个结点的不同形态标号有序树的数目

$$\frac{n!}{n} c_{2(n-1)}^{n-1} = (n-1)! c_{2(n-1)}^{n-1}$$

总结

- 树和森林的概念
- 树与二叉树的联系、区别与转换
- 树的链式存储
  - 左子/右兄二叉链表
  - 父指针表示法
- 树的顺序存储
- K叉树
- 树计数

课堂练习 ☺

1. 设树T的度为4，若其中度为1，2，3和4的结点个数分别为4，2，1，1，那么树T中有多少叶结点?
2. 一棵具有n个分支结点的非空满K叉树，有多少个叶结点?
3. 若森林F对应的二叉树B中有m个结点，B的根结点 r 的右子树具有n个结点，那么森林F中第1棵树含有多少结点? ________

   A. m-n； B. m-n-1； C. n+1； D. 无法确定
4. 将有关二叉树的概念推广到三叉树，请计算一棵有251个结点的完全三叉树的高度。
5. 设树T在后根遍历时的结点排列及其相应的度数如下：

   后根遍历： B D E F C G J K I L H A

   度 数： 0 0 0 0 3 0 0 0 2 0 2 4

   请画出树 T 。