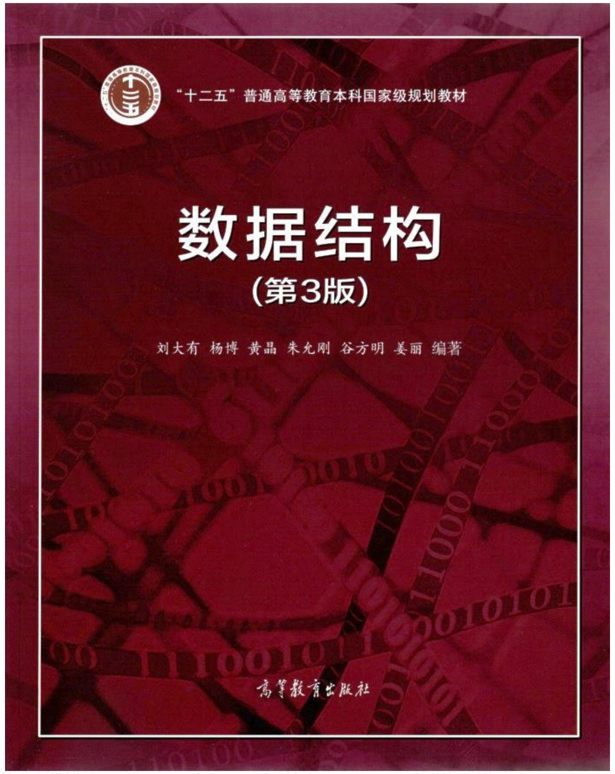
计算机学院王湘浩班
2024级



树的存储和操作

- 树/森林与二叉树的转换
- 树的存储结构
- 树的基本操作
- 树的序列化与反序列化

数据之法
结构之美
算法之道

Last updated on 2025.4

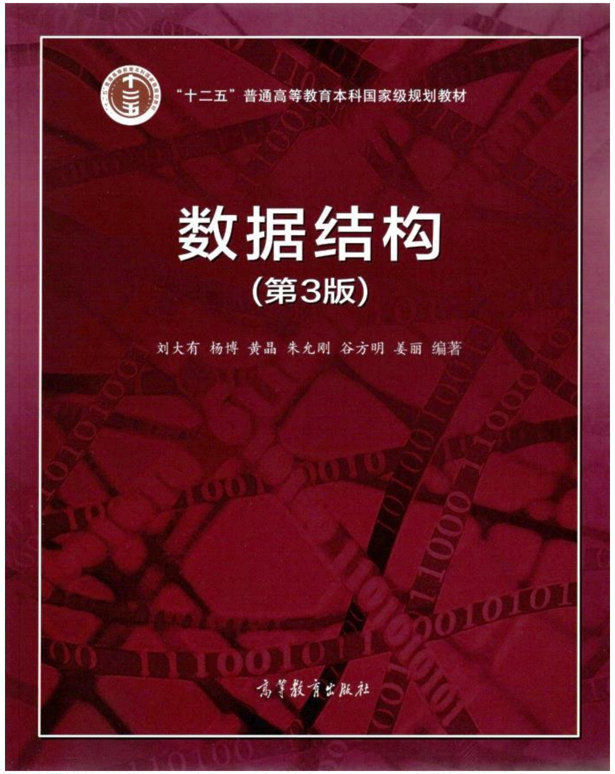
zhuyungang@jlu.edu.cn

建议大家争取在大学四年中积累编写10万行代码的经验。我们必须明白的是：好程序员是写出来的。

——李开复

原微软全球副总裁、微软亚洲研究院院长

原Google全球副总裁兼大中华区总裁



树的存储和操作

- **树/森林与二叉树的转换**
- 树的存储结构
- 树的基本操作
- 树的序列化与反序列化

数据之法
结构之美
算法之道

Last updated on 2025.4

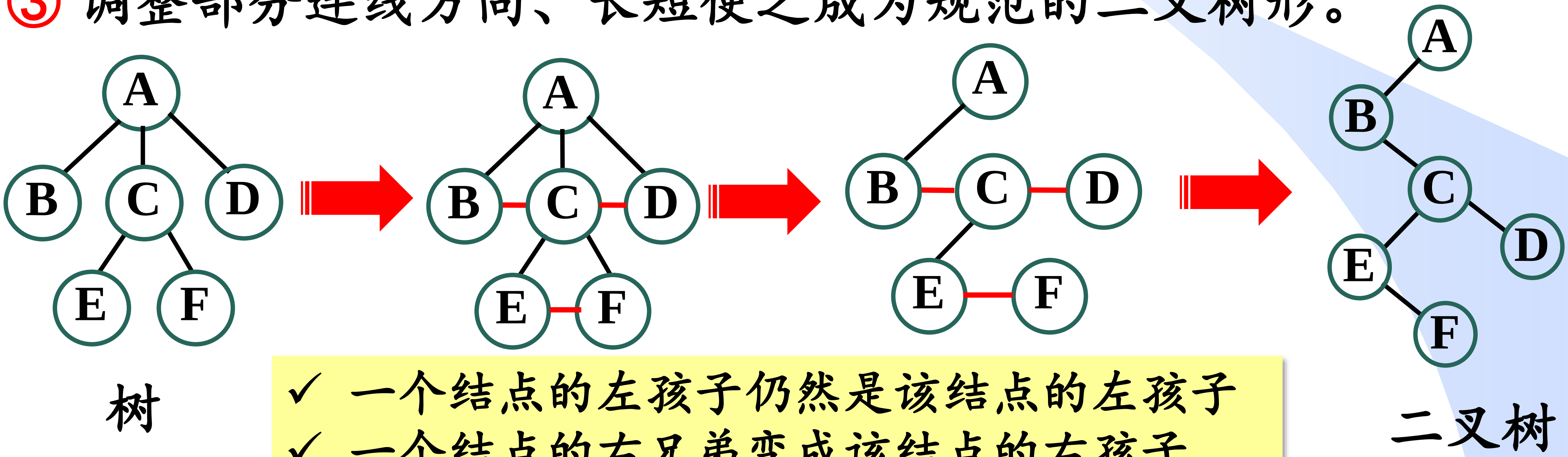
zhuyungang@jlu.edu.cn

（1） 树转换成二叉树

① 在所有兄弟结点间加一条连线；

② 对每个结点与其子结点的连线：保留与其左孩子的连线，去掉与其它孩子间的连线；

③ 调整部分连线方向、长短使之成为规范的二叉树形。

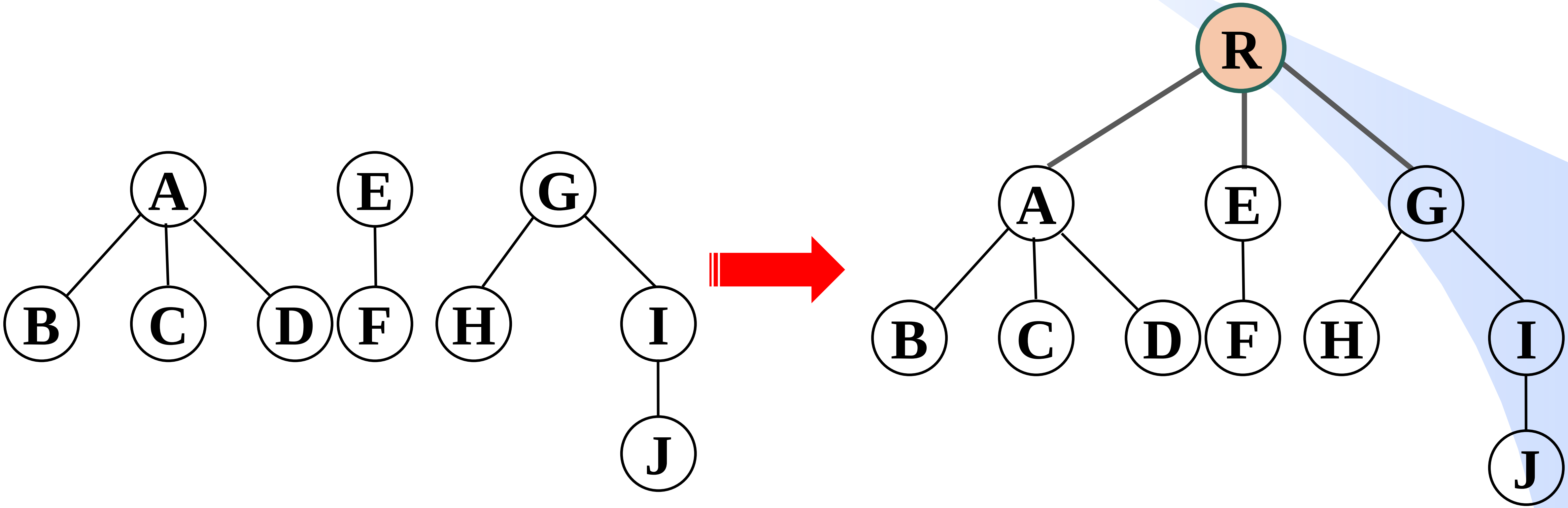


- ✓ 一个结点的左孩子仍然是该结点的左孩子
- ✓ 一个结点的右兄弟变成该结点的右孩子
- ✓ 转换后的二叉树的根结点无右孩子

（2）森林转换成二叉树（方法1）

A

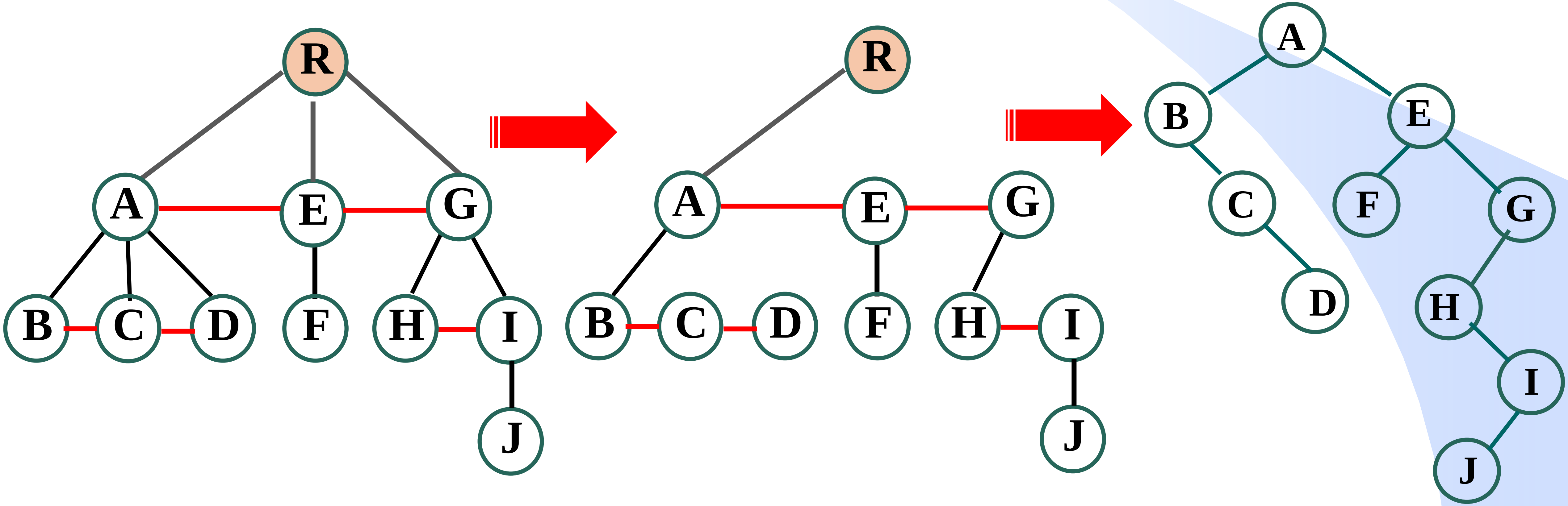
① 引入一个虚拟的根，将森林转变为树；
② 将树转换为二叉树。



（2）森林转换成二叉树（方法1）

A

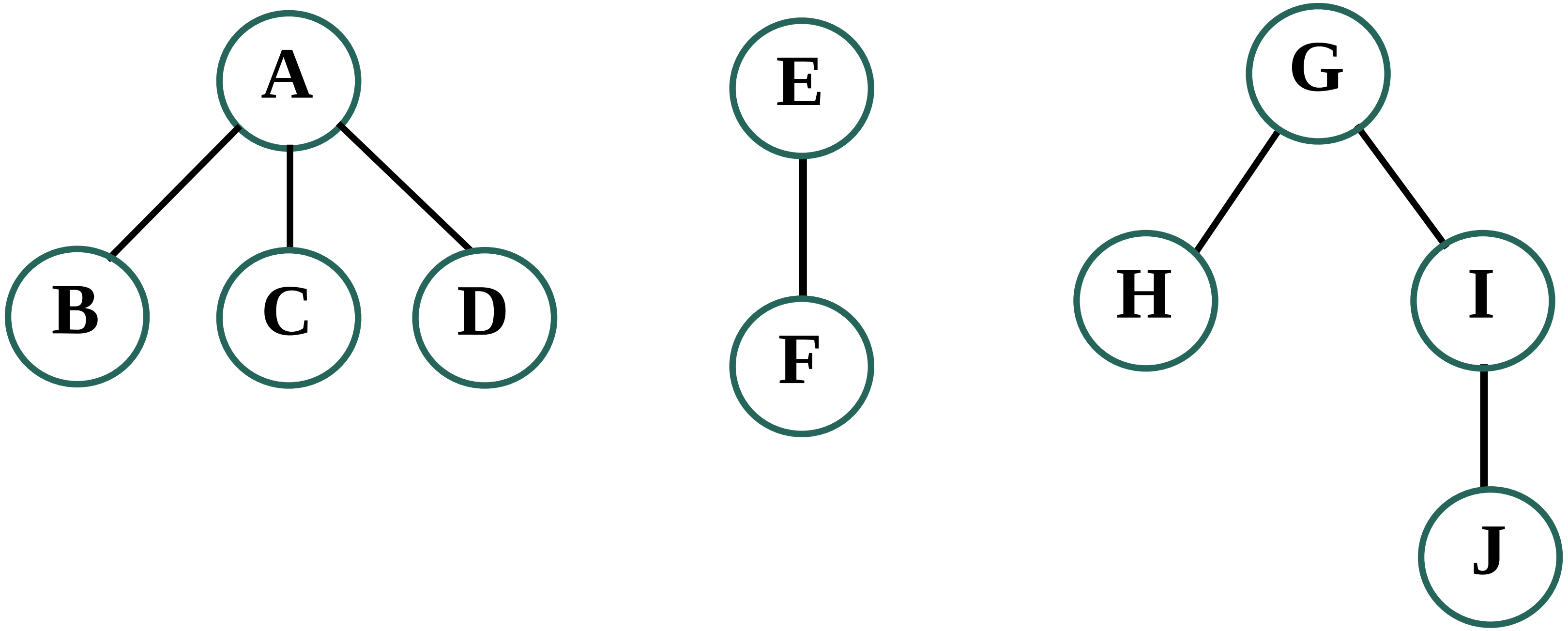
① 引入一个虚拟的根，将森林转变为树；
② 将树转换为二叉树。

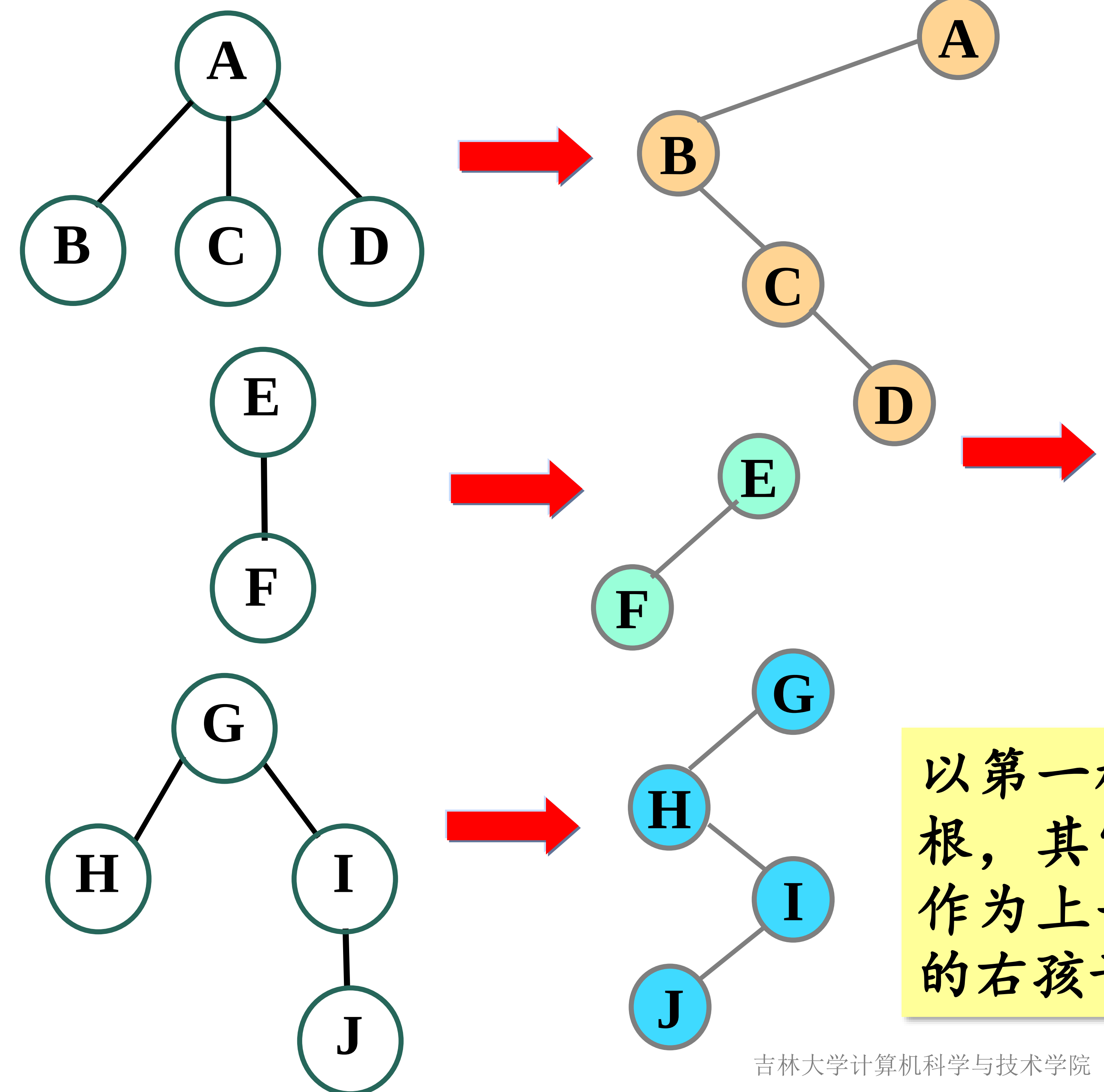


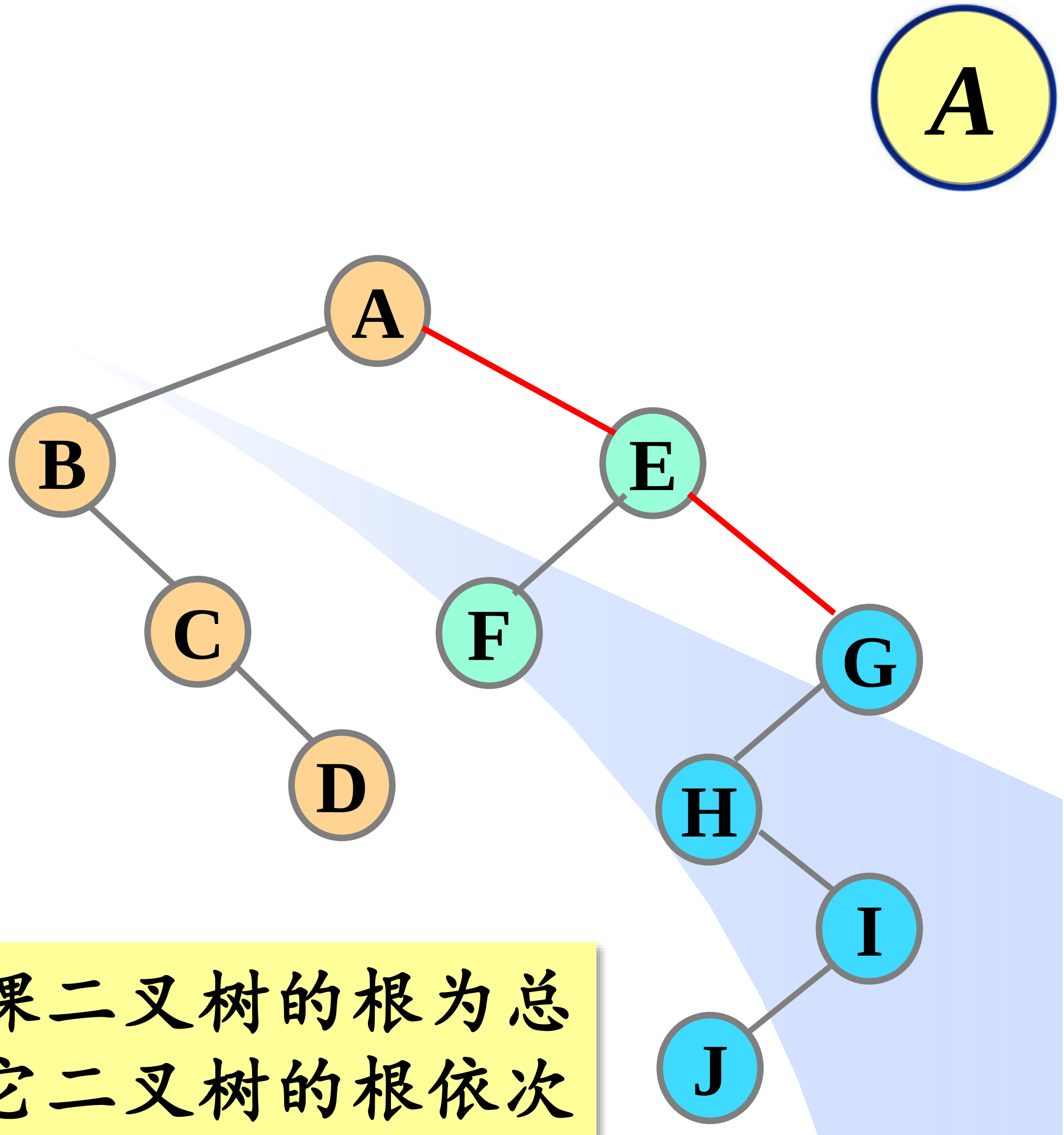
（2）森林转换成二叉树（方法2）

A

① 首先将每个树转换为二叉树；
② 将以第一个二叉树的根为总根，将其它二叉树的根依次作为上一个二叉树根结点的右孩子。







以第一棵二叉树的根为总根，其它二叉树的根依次作为上一棵二叉树根结点的右孩子

（3）二叉树转换成树

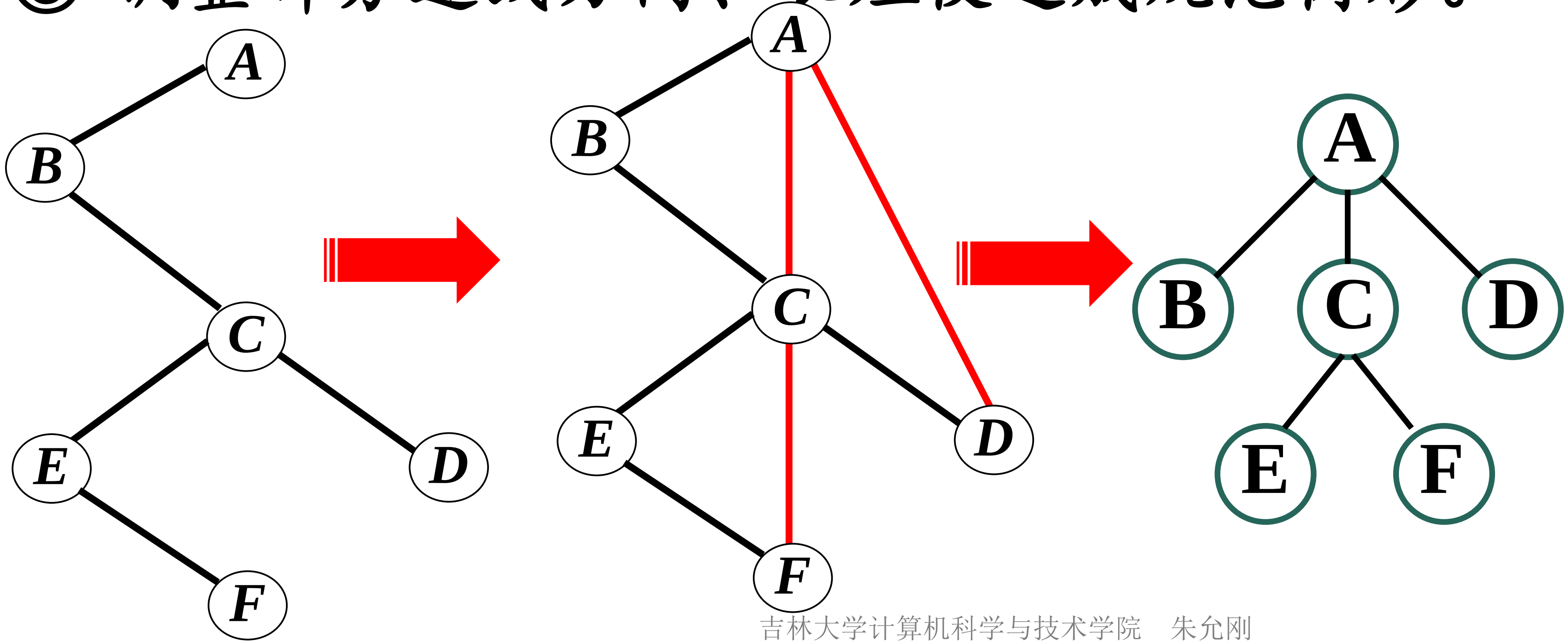
A

二叉树的右子树为空，则可转换为一棵树。

① 对每个结点，若该结点有左孩子，将左孩子的右孩子、右孩子的右孩子……与该结点用线连接起来；

② 去掉每个结点与其右孩子之间的连线；

③ 调整部分连线方向、长短使之成规范树形。



✓一个结点的左孩子仍是该结点的孩子

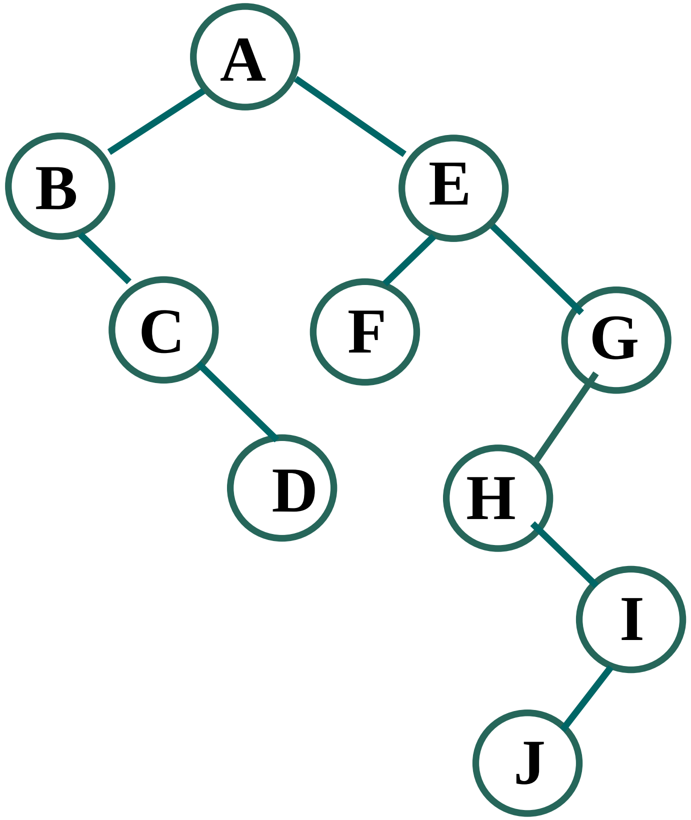
✓一个结点的右孩子变为该结点的右兄弟

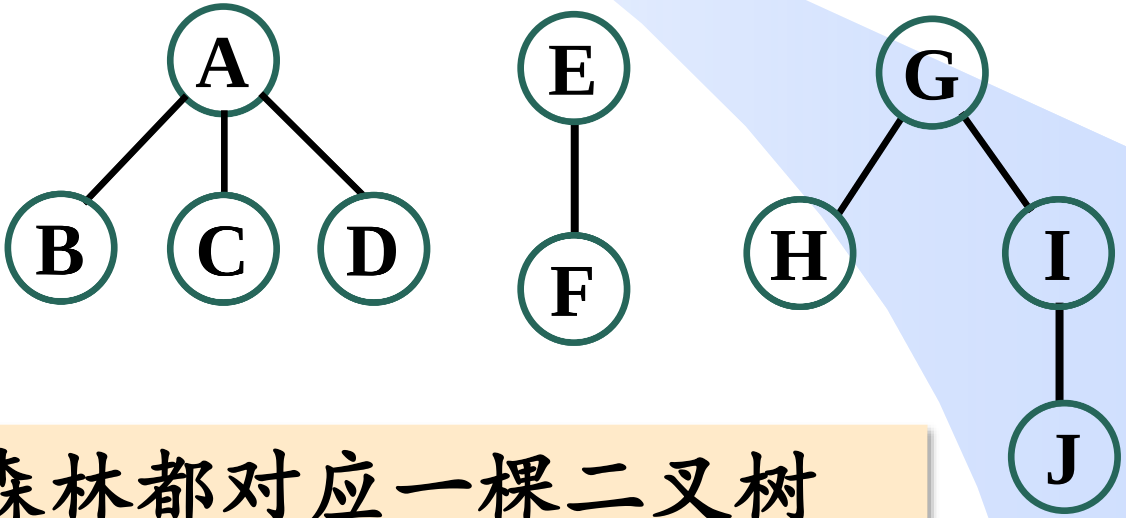
（4）二叉树转换成森林

二叉树的右子树不为空，则可转换为森林。

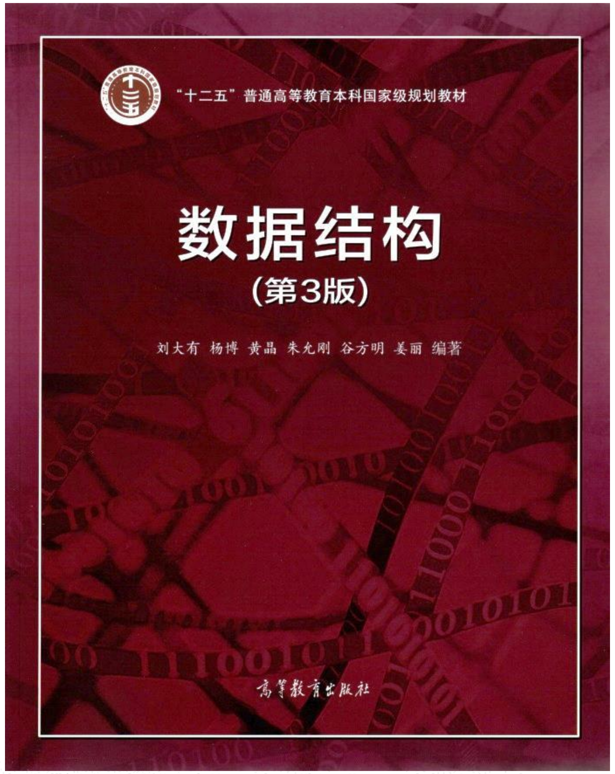
① 从根出发，断开其与右孩子的连线，得到多个二叉树；

② 将每个二叉树按以上方法转化为树。





- 任一树/森林都对应一棵二叉树
- 任一二叉树都对应唯一的树/森林



树的存储和操作

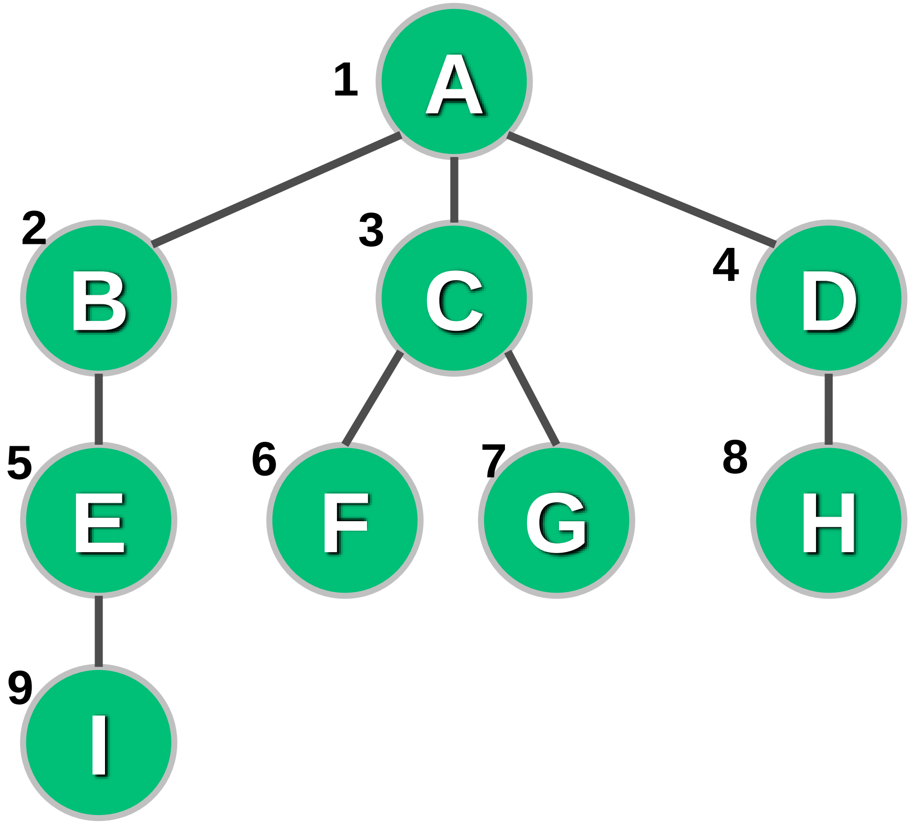
- 树与二叉树的转换
- **树的存储结构**
- 树的基本操作
- 树的序列化与反序列化

数据之法 结构之美 算法之道

Last updated on 2025.4

zhuyungang@jlu.edu.cn

顺序存储——双亲表示法

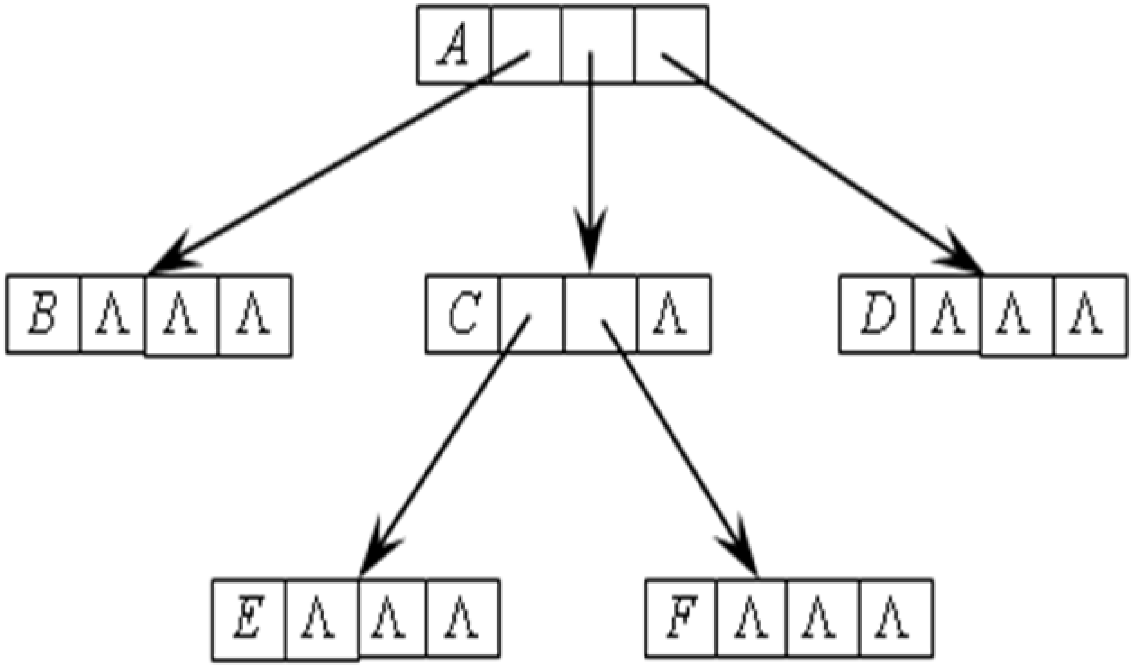


- ✓ 按某种遍历顺序对结点编号
- ✓ 编号为*i*的结点存在数组第*i*位
- ✓ 数组下标：结点编号
- ✓ *parent*[*i*]：结点*i*的父结点下标
- ✓ 便于涉及父结点的操作

data		A	B	C	D	E	F	G	H	I
parent		0	1	1	1	2	3	3	4	5
	0	1	2	3	4	5	6	7	8	9

链接存储——多叉链表（孩子链）

A



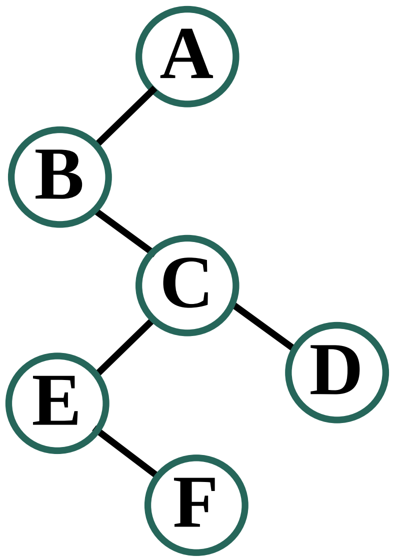
每个结点的指针数以整棵树中孩子最多的结点为准，大量指针为空，浪费空间

链接存储——左孩子-右兄弟（LCRS）链接结构

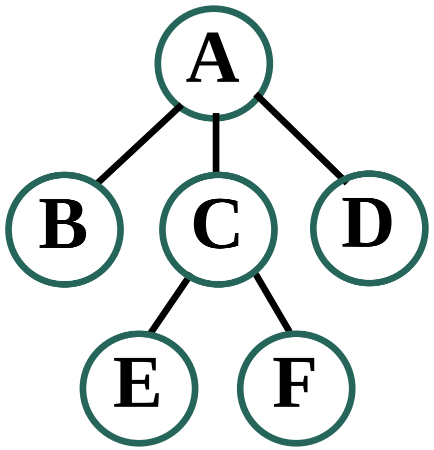
A

回顾：树与二叉树的转换

二叉树结点的左孩子 ⇔ 树结点的左孩子
二叉树结点的右孩子 ⇔ 树结点的右兄弟



二叉树



树

链接存储——左孩子-右兄弟（LCRS）链接结构

A

二叉树结点

Left	*Data*	*Right*

二叉树结点的左孩子 ⇕ 树结点的左孩子

二叉树结点的右孩子 ⇕ 树结点的右兄弟

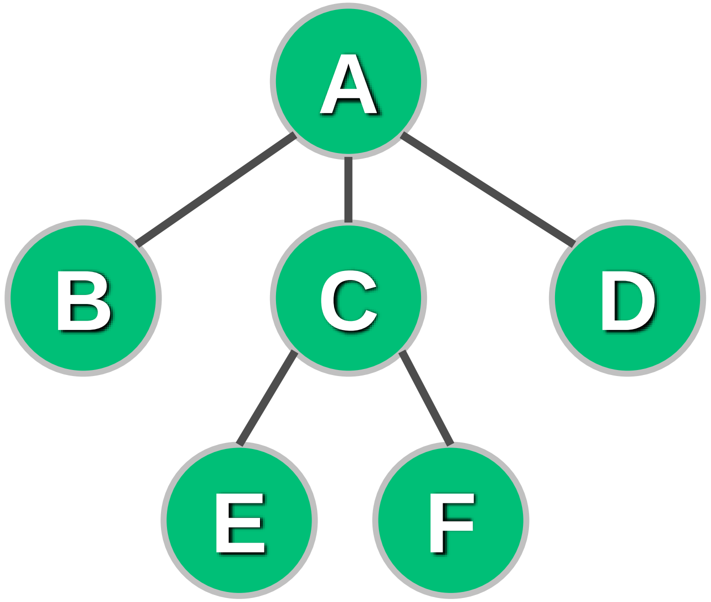
树结点

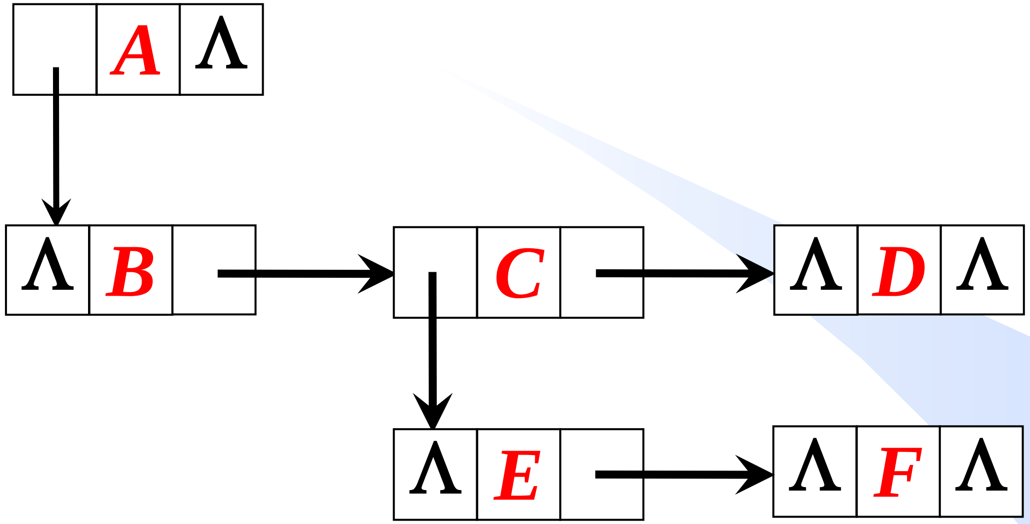
LeftChild	*Data*	*RightSibling*

✓ 与二叉树的链接存储结构相同

✓ 可利用二叉树的算法框架来实现树的操作

左孩子-右兄弟表示法示例

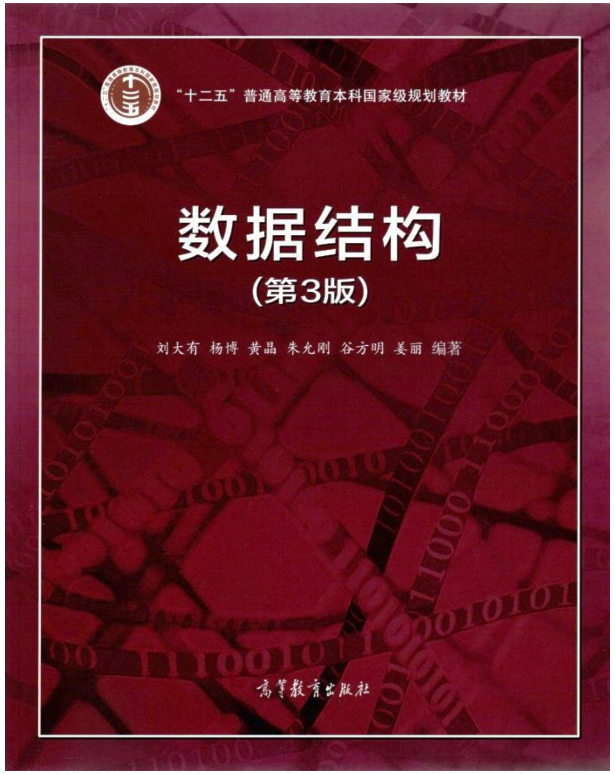




结点结构

LeftChild	*Data*	*RightSibling*

```
struct TreeNode{
   int data;
   TreeNode* LeftChild;
   TreeNode* RightSibling;
};
```



树的存储和操作

- 树与二叉树的转换
- 树的存储结构
- **树的基本操作**
- 树的序列化与反序列化

数据之法
结构之美
算法之道

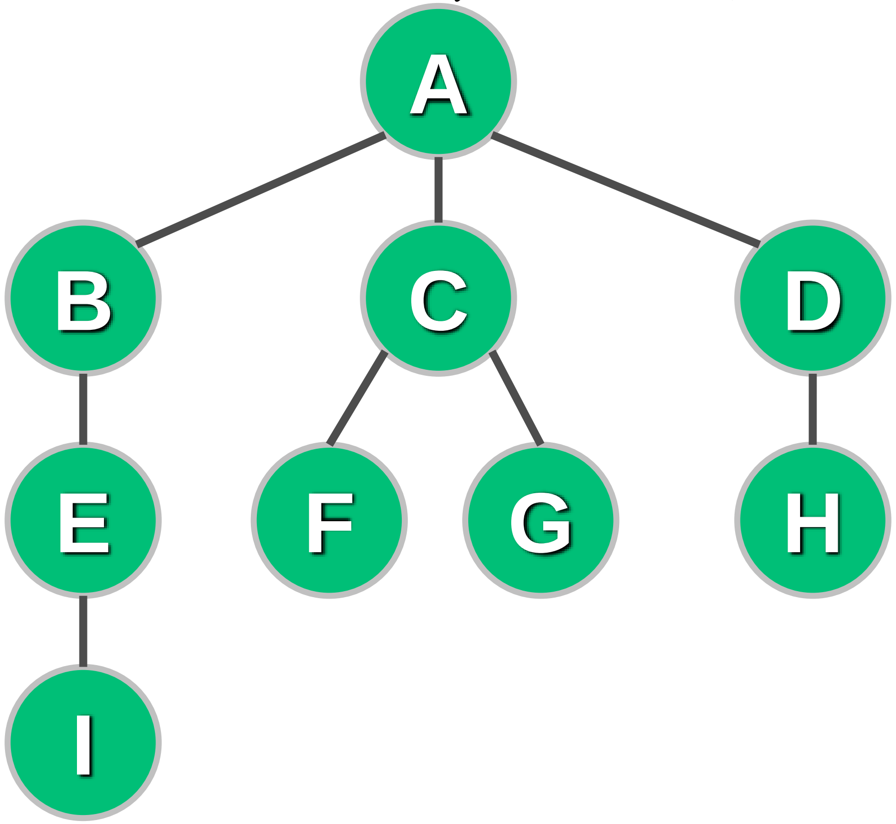
zhuyungang@jlu.edu.cn

树的先根遍历

A

（1）访问根结点

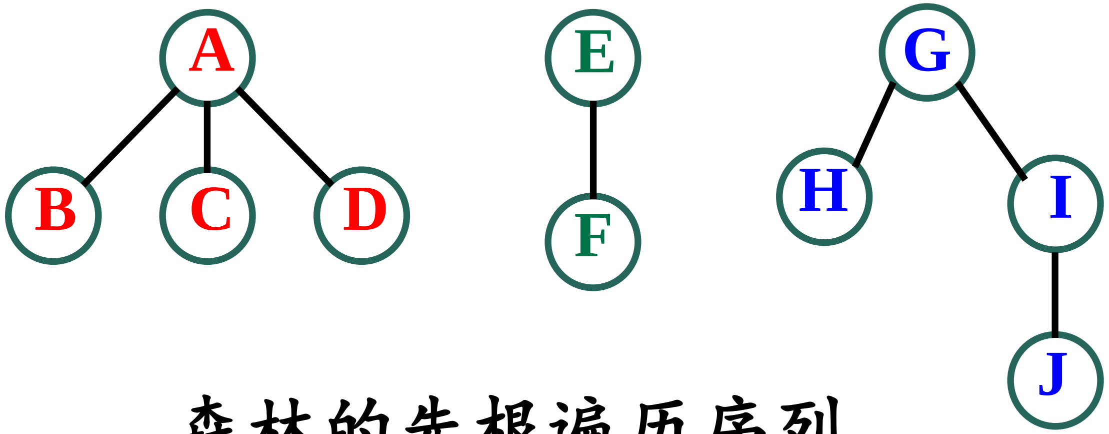
（2）从左到右依次先根次序遍历树的诸子树



先根序列　A B E I C F G D H

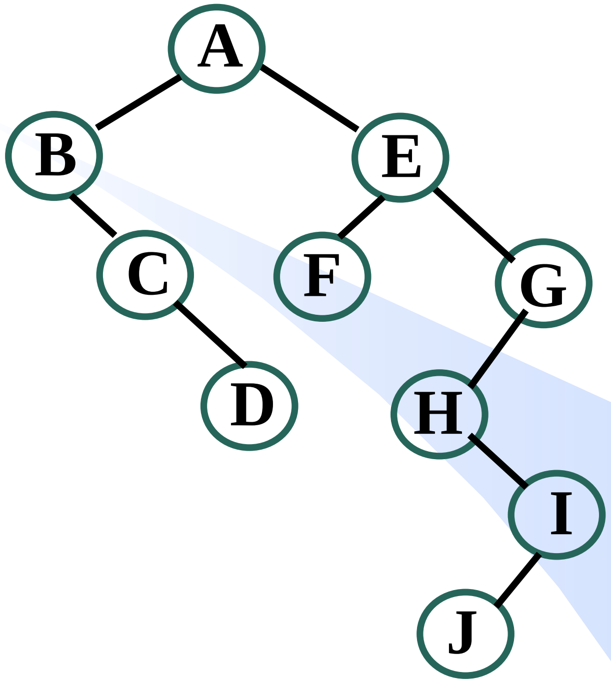
森林的先根遍历

① 访问森林中第一棵树的根结点；
② 先根遍历第一棵树中的诸子树；
③ 先根遍历其余的诸树。



森林的先根遍历序列

A B C D E F G H I J



二叉树的先根序列

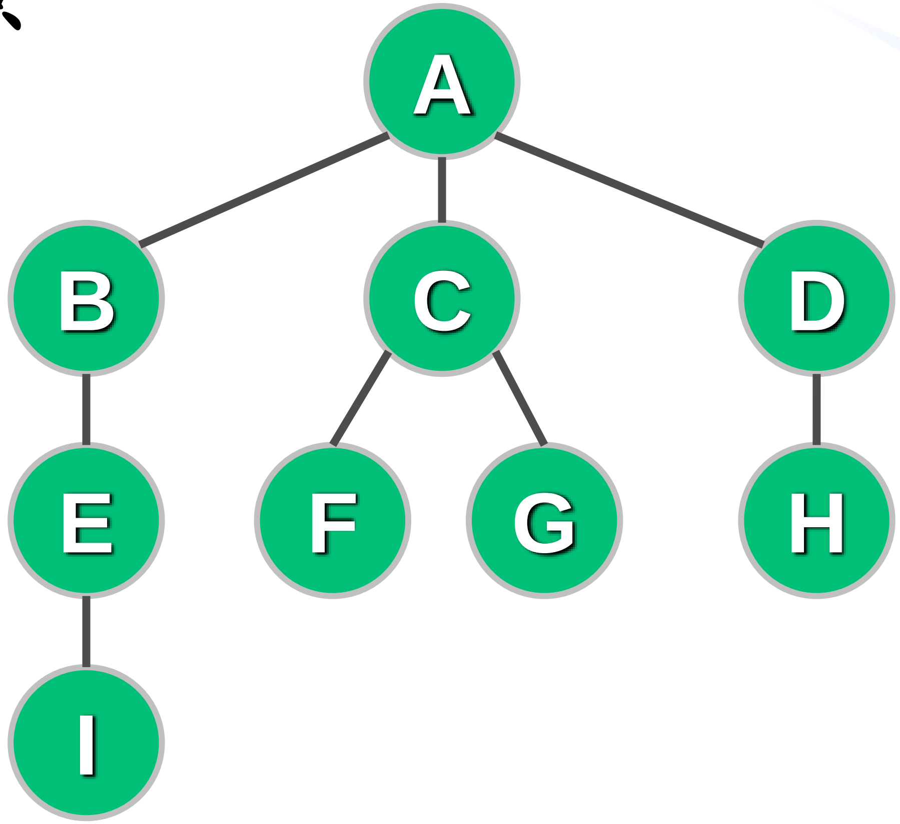
A B C D E F G H I J

森林的先根序列与它对应的二叉树的先根序列相等

树的后根遍历

A

（1）从左到右依次后根遍历根结点的诸子树
（2）访问根结点

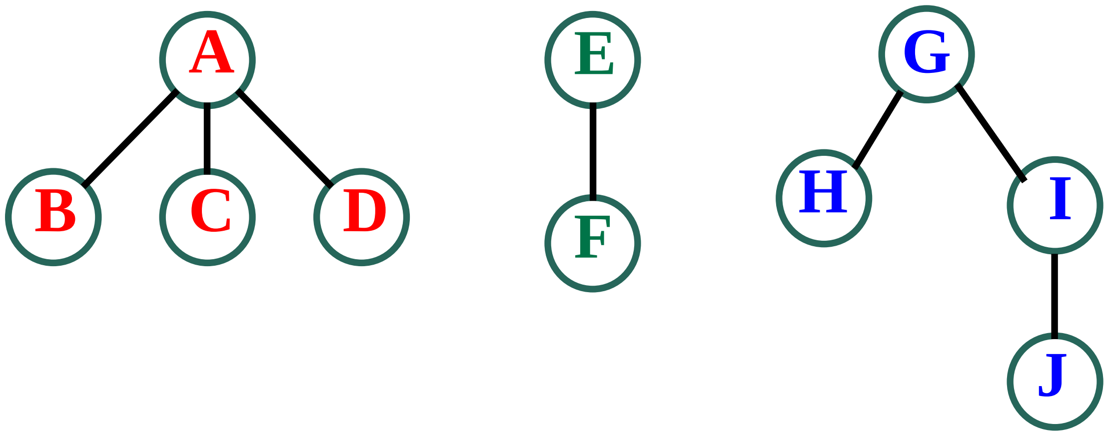


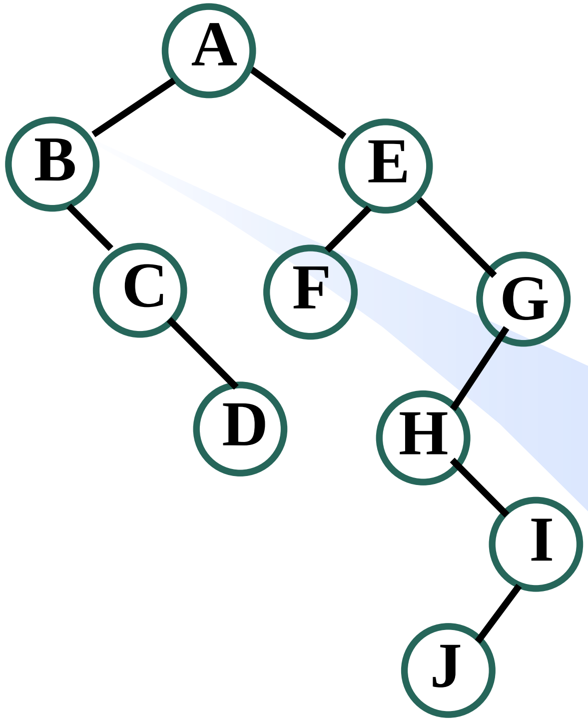
树有中根遍历么？

后根序列 IEBFGCHDA

森林的后根遍历

① 后根遍历第一棵树的诸子树；
② 访问森林中第一棵树的根结点；
③ 后序遍历其余的诸树。





课下思考

由树的先根序列和后根序列，能唯一确定一棵树么？

森林的后根遍历序列

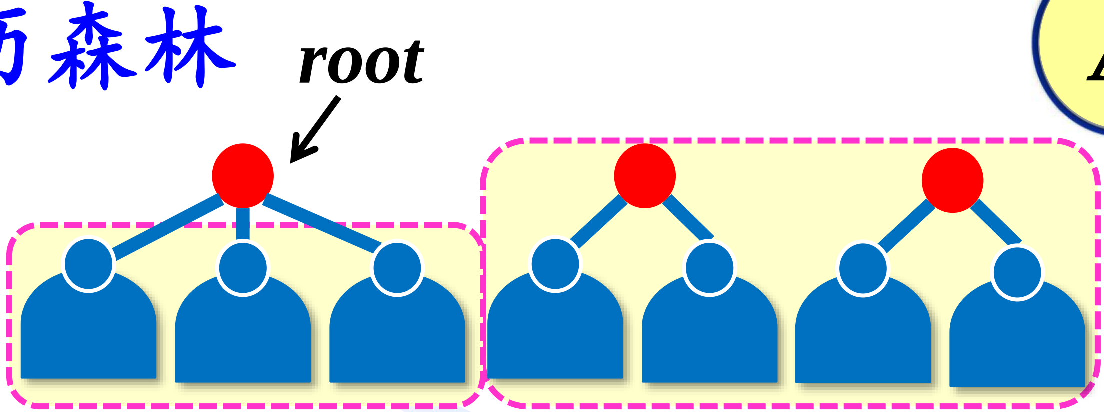
B C D A F E H J I G

二叉树的中根序列

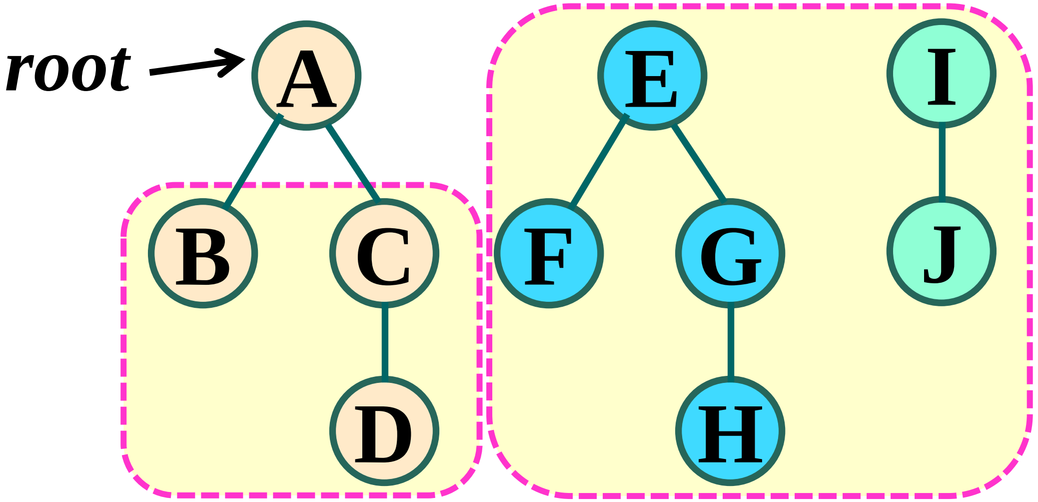
B C D A F E H J I G

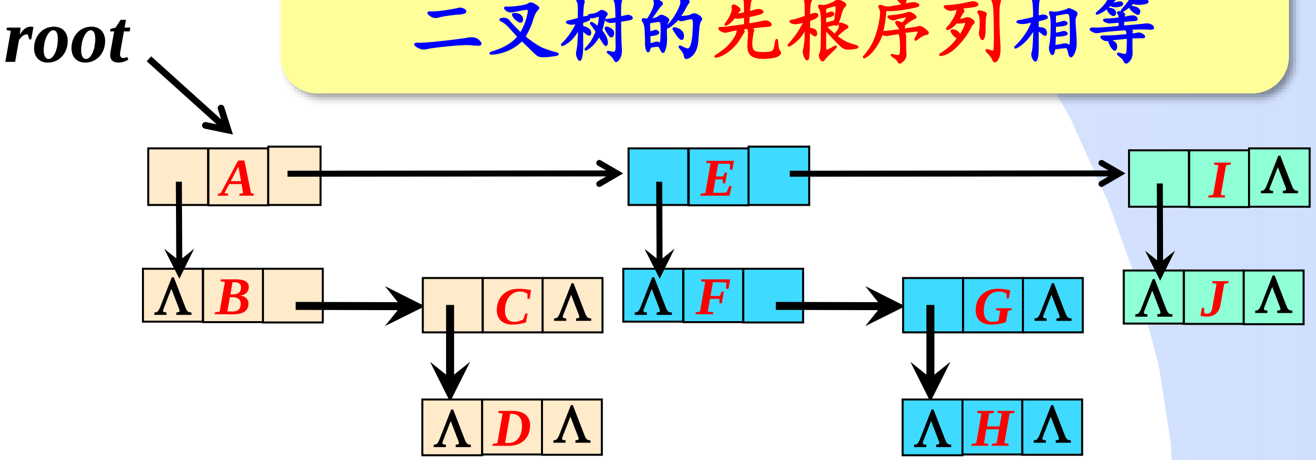
森林的后根序列与它对应的二叉树的中根序列相等

先根遍历森林



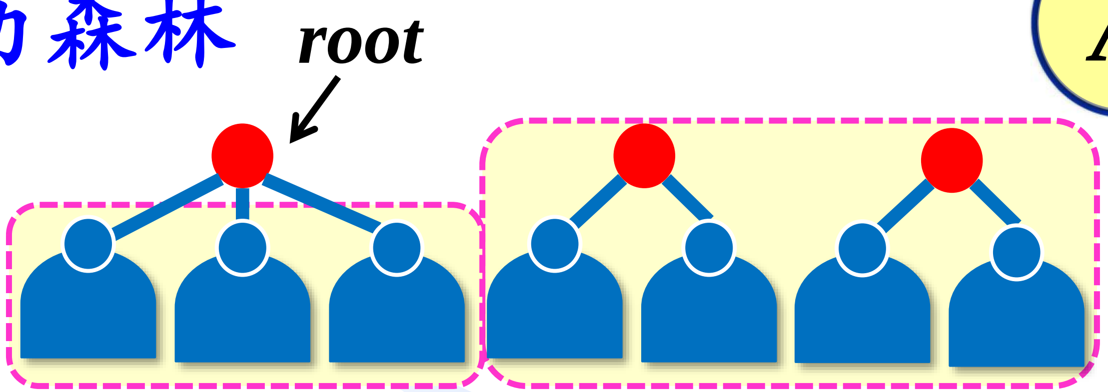

```
void PreOrder(TreeNode* root){
   if(root==NULL) return;
   visit(root->data); //访问root
   PreOrder(root->LeftChild);      //先根遍历root的各子树
   PreOrder(root->RightSibling); //先根遍历 以root的兄弟为根的树
}
```



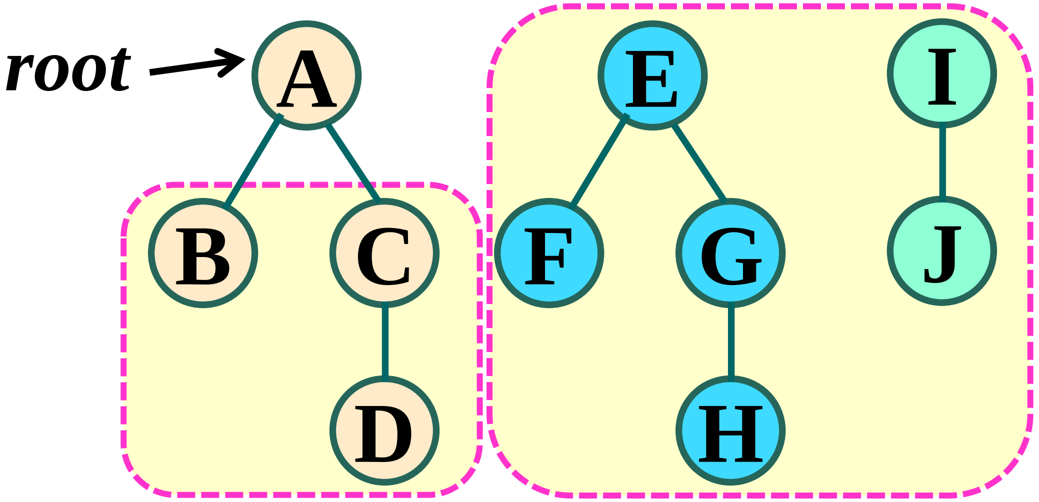


森林的先根序列与它对应的二叉树的先根序列相等

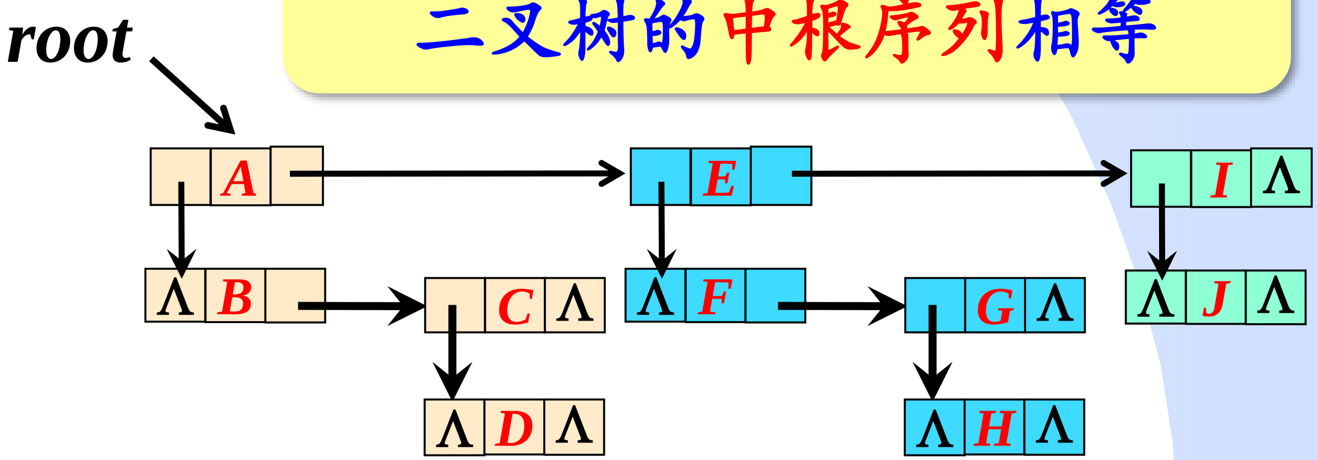
后根遍历森林




```
void PostOrder(TreeNode* root){
   if(root==NULL) return;
   PostOrder(root->LeftChild);    //后根遍历root的各子树
   visit(root->data); //访问root
   PostOrder(root->RightSibling); //后根遍历以root的兄弟为根的树
}
```



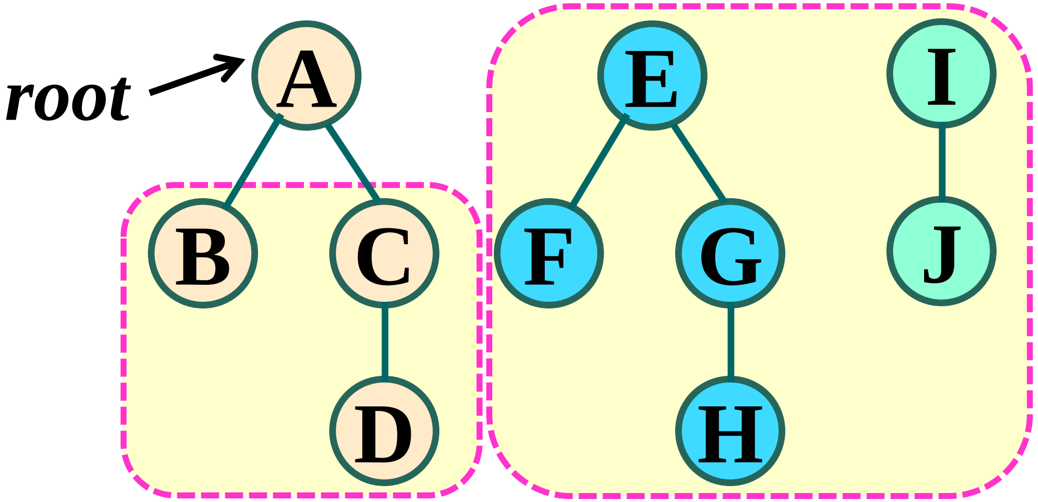
森林的后根序列与它对应的二叉树的中根序列相等

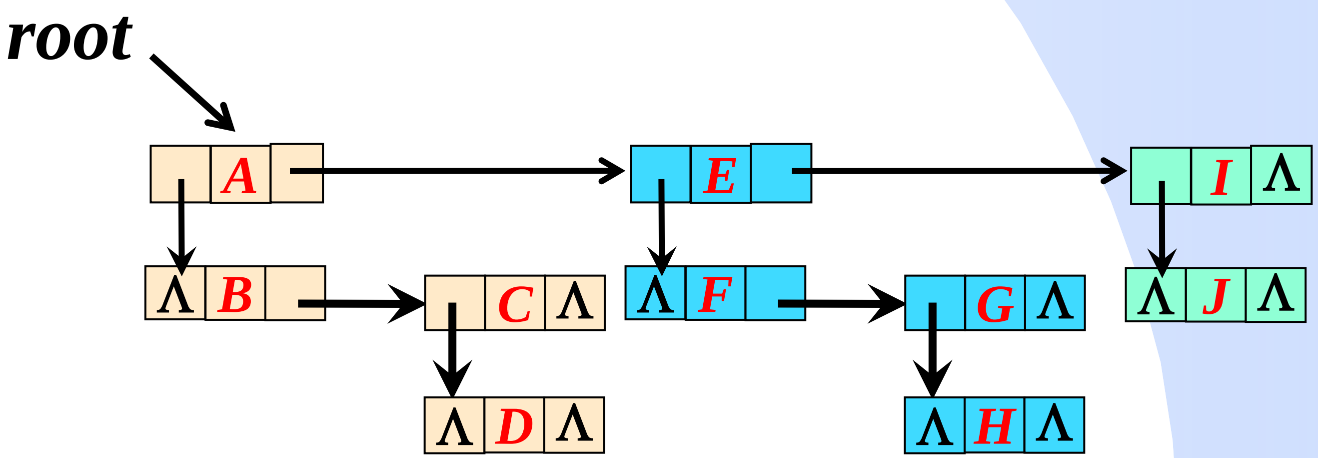


在以root为根的森林中找数据值等于K的结点

A

```
TreeNode* Search(TreeNode* root, int K){
    if(root==NULL) return NULL;        //树空，没找到
    if(root->data==K) return root;     //根即为所求
    TreeNode* ans= Search(root->LeftChild,K); //在root子树里找
    if(ans!=NULL) return ans;
    return Search(root->RightSibling,K);//在root兄弟为根的树里找
}
```

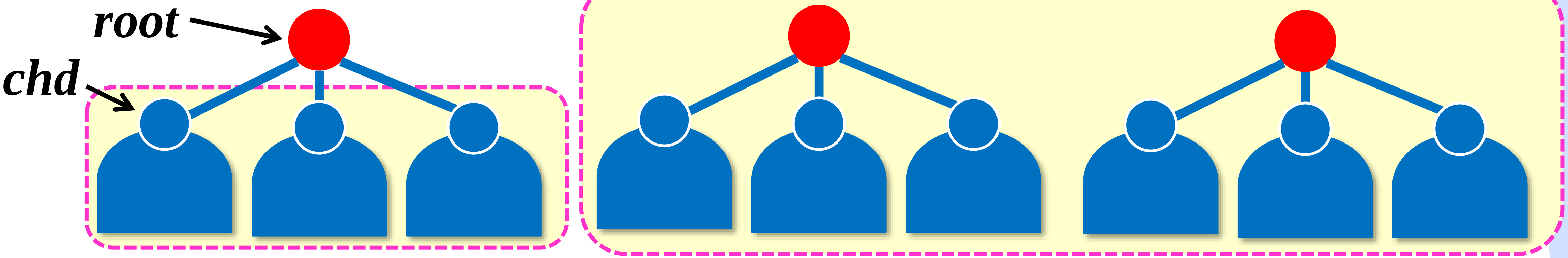




在以root为根的森林中找p的父结点

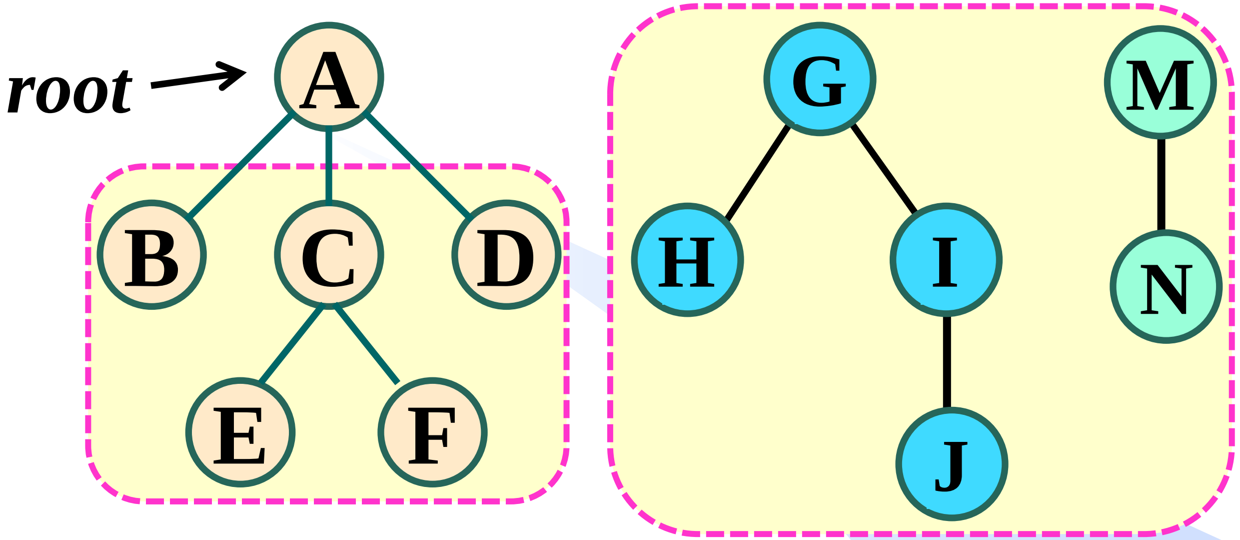
A

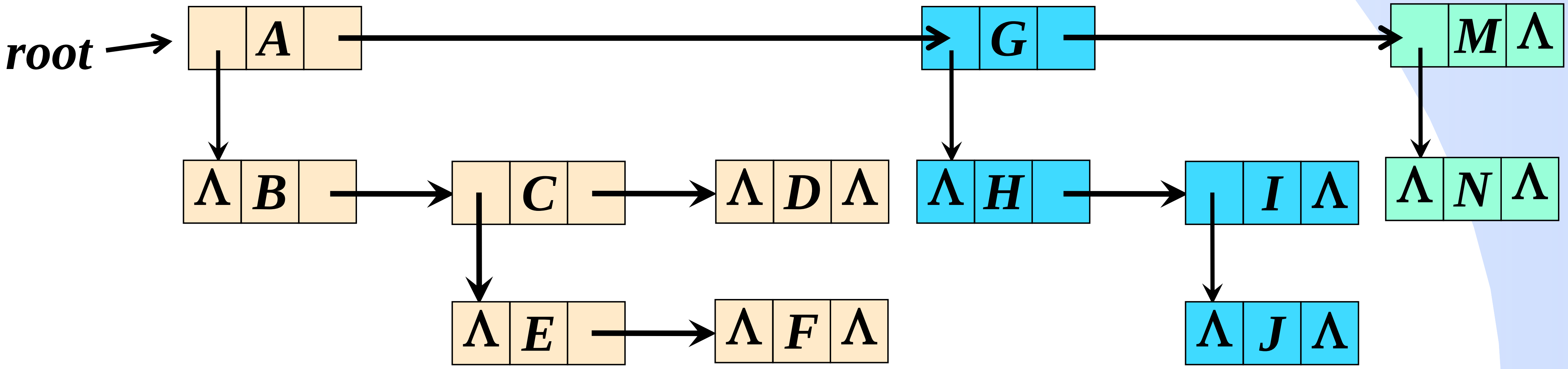
```
TreeNode* FindParent(TreeNode* root, TreeNode* p){
    if(root==NULL || p==NULL || p==root) return NULL;
    for(TreeNode* chd=root->LeftChild; chd; chd=chd->RightSibling)
        if(chd==p) return root; //root就是p的父结点
    TreeNode* ans=FindParent(root->LeftChild, p);//在root的子树里找
    if(ans!=NULL) return ans;
    return FindParent(root->RightSibling, p); //在其余树里找

}
```



删除整个森林

```
void Del(TreeNode* &root){
    //删除以root为根的森林
    if(root==NULL) return;
    Del(root->LeftChild);
    Del(root->RightSibling);
    delete root;
    root=NULL;
}
```

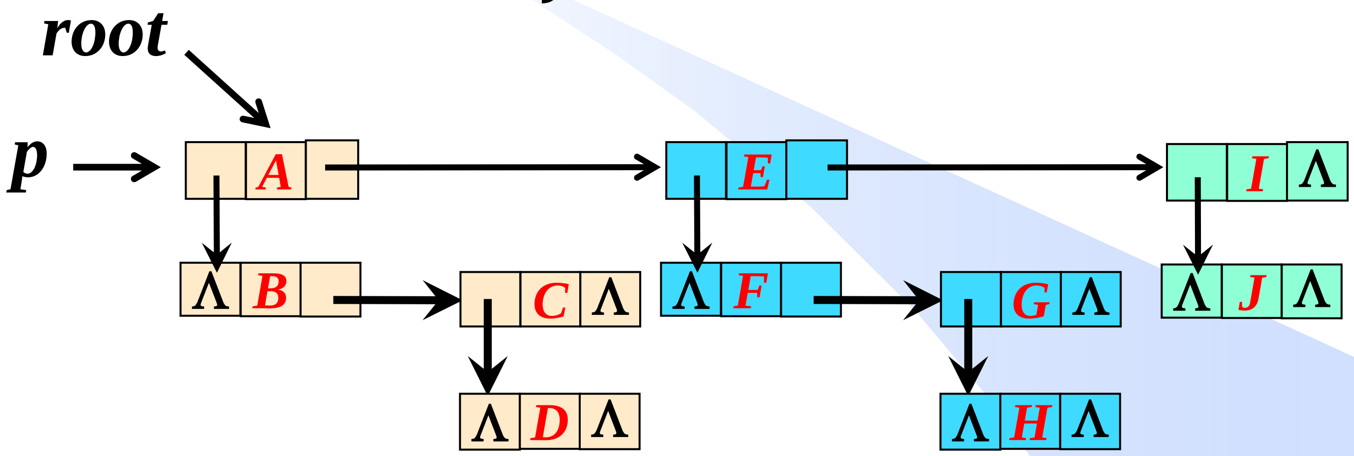


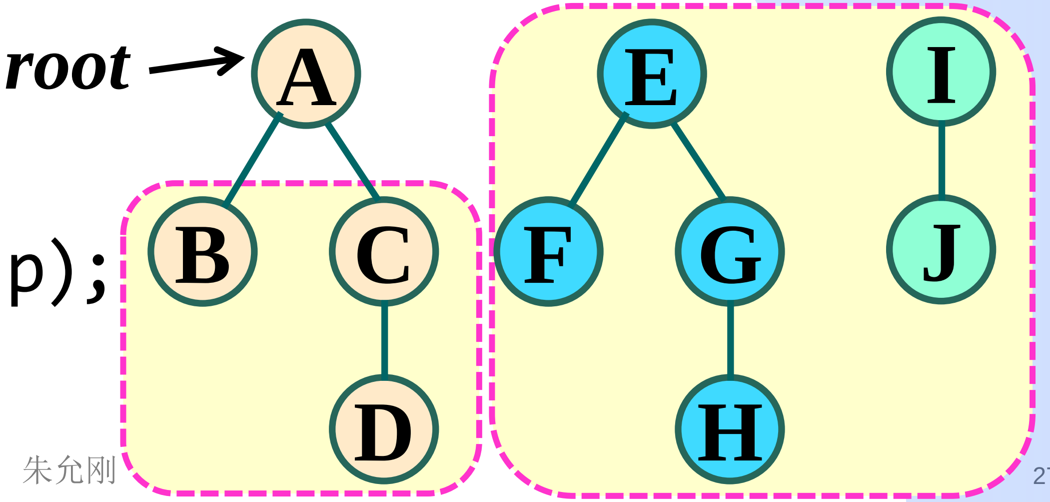


B

在森林中删除子树

```
bool DelSubTree(TreeNode *&root, TreeNode *p){
  //在以root为根的森林中删除以p为根的子树,删除成功返回真,否则返回假
  if(root==NULL || p==NULL) return false;
  if(p==root) {
     root=p->RightSibling;
     p->RightSibling=NULL;
     Del(p);
     return true;
  }
  if(DelSubTree(root->LeftChild, p))
     return true;
  return DelSubTree(root->RightSibling,p);
}
```



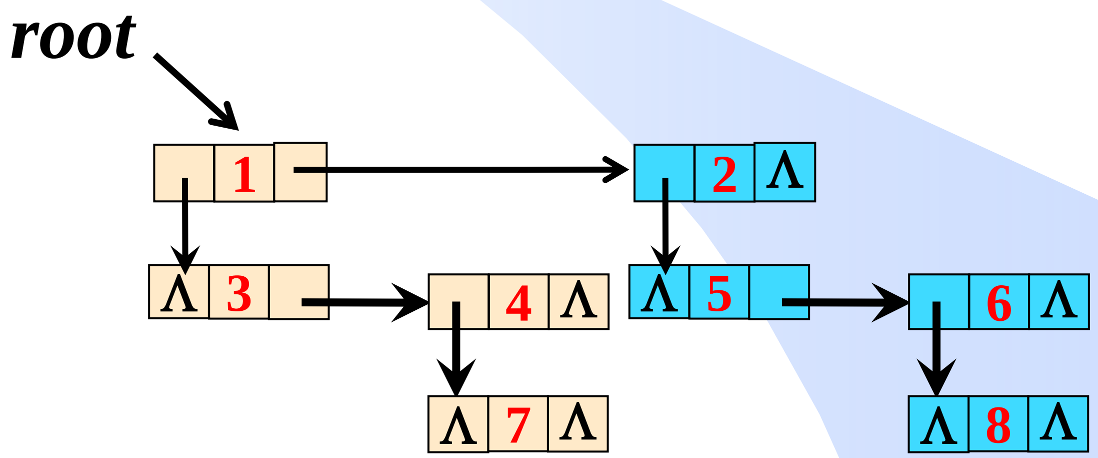


创建树/森林

A

已知一棵非空树结点的数据域为不等于0的整数，输入为一组用空格间隔的整数，表示带空指针信息的树先根序列，其中空指针信息用0表示，如1 3 0 4 7 0 0 0 2 5 0 6 8 0 0 0 0表示如下树。

```
TreeNode* CreateTree(){
    int k;
    scanf("%d", &k);
    if(k==0) return NULL;
    TreeNode *root = new TreeNode;
    root->data = k;
    root->LeftChild = CreateTree();
    root->RightSibling = CreateTree();
    return root;
}
```

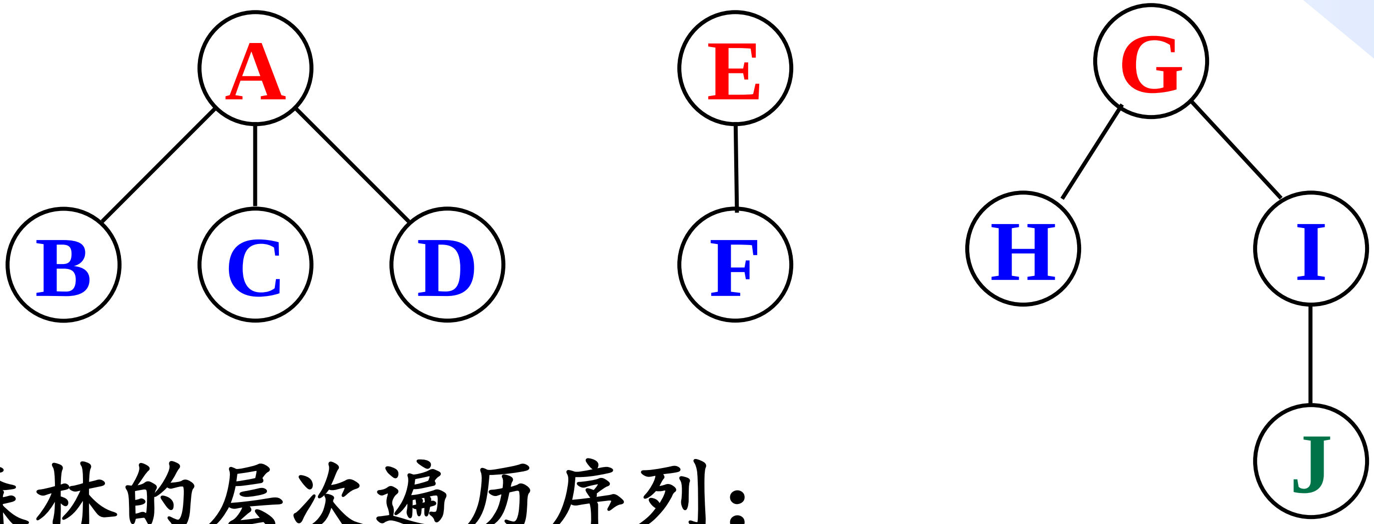


树和森林的层次遍历

B

从第0层到最后一层，且同层结点的从左到右访问。

[例]

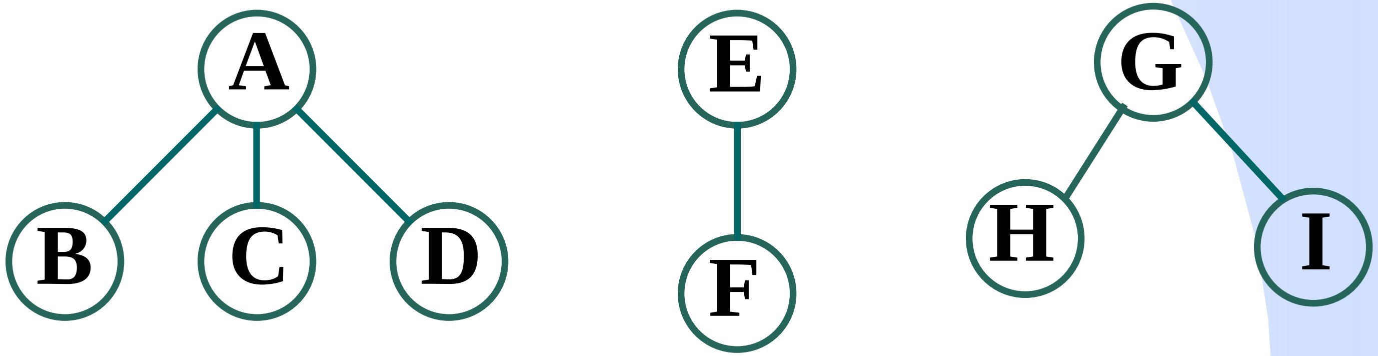


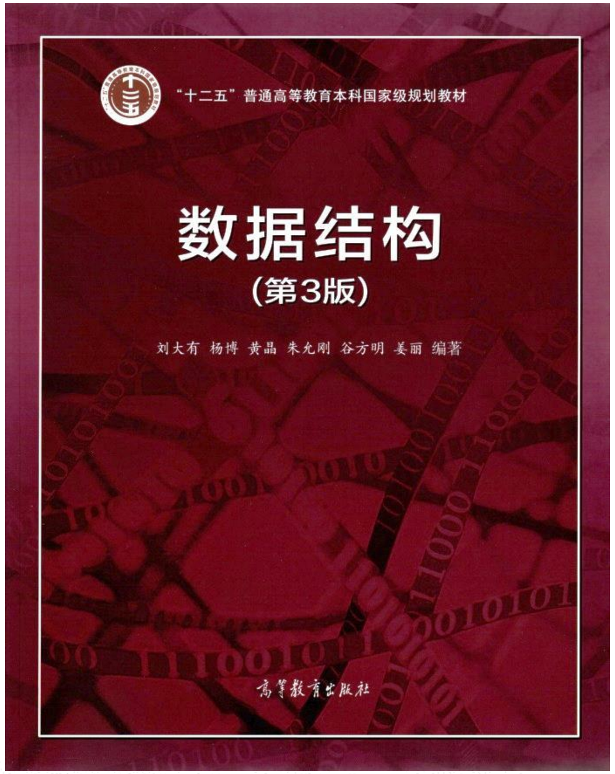
森林的层次遍历序列：

A E G B C D F H I J

层次遍历森林

```
void LevelOrder(TreeNode* root){
   Queue<TreeNode*> Q; TreeNode *p, *chd;
   for(p=root; p!=NULL; p=p->RightSibling)
      Q.enQueue(p);             //每棵树的根都入队
   while(!Q.empty()){
      p=Q.deQueue();            //出队一个结点p
      visit(p->data);           //访问p
      for(chd=p->LeftChild; chd!=NULL; chd=chd->RightSibling)
          Q.enQueue(chd);       //p的每个孩子都入队
   }
}
```





树的存储和操作

- 树与二叉树的转换
- 树的存储结构
- 树的基本操作
- **树的序列化与反序列化**

数据之法
结构之美
算法之道

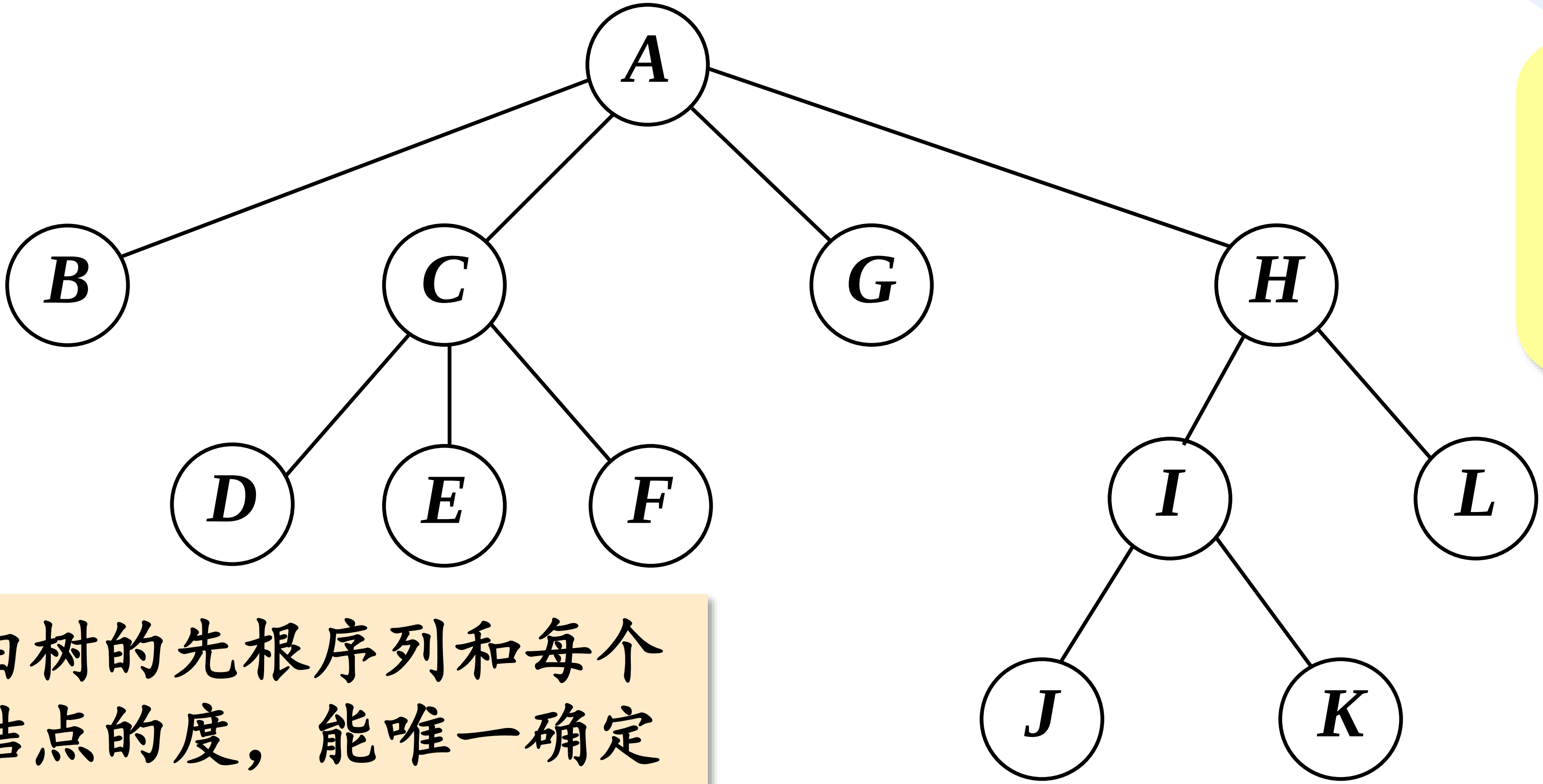
Last updated on 2025.4

zhuyungang@jlu.edu.cn

树的先根序列+结点度

B

先根序列	A	B	C	D	E	F	G	H	I	J	K	L
结点度	4	0	3	0	0	0	0	2	2	0	0	0



反序列化：从右往左扫描，若结点X的度为k，则选择离X最近的k棵子树作为X的子树

由树的先根序列和每个结点的度，能唯一确定树的结构。

树的先根序列+结点度

B

定理 已知一棵树的先根序列和每个结点的度，能唯一确定树的结构。

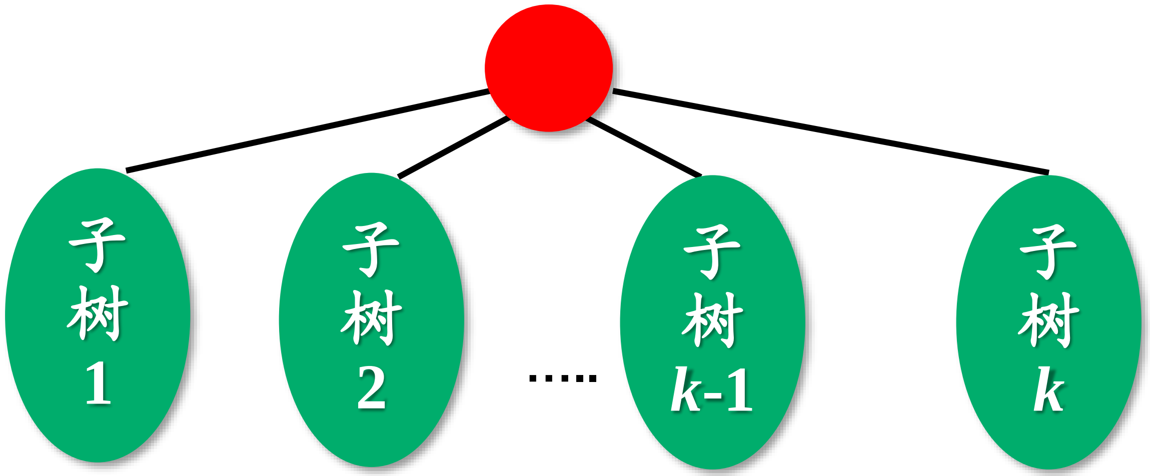
证明：用数学归纳法

1. 若树中只有一个结点，定理显然成立。

2. 假设树中结点个数小于$n(n\geq 2)$时定理成立。

3. 当树中有n个结点时，由树的先根序列可知，第一个结点是根结点，设该结点的度为k，$k\geq 1$，因此根结点有k个子树。每个子树的结点个数小于n，由归纳假设可知，每个子树可以唯一确定，从而整棵树的树形可以唯一确定。

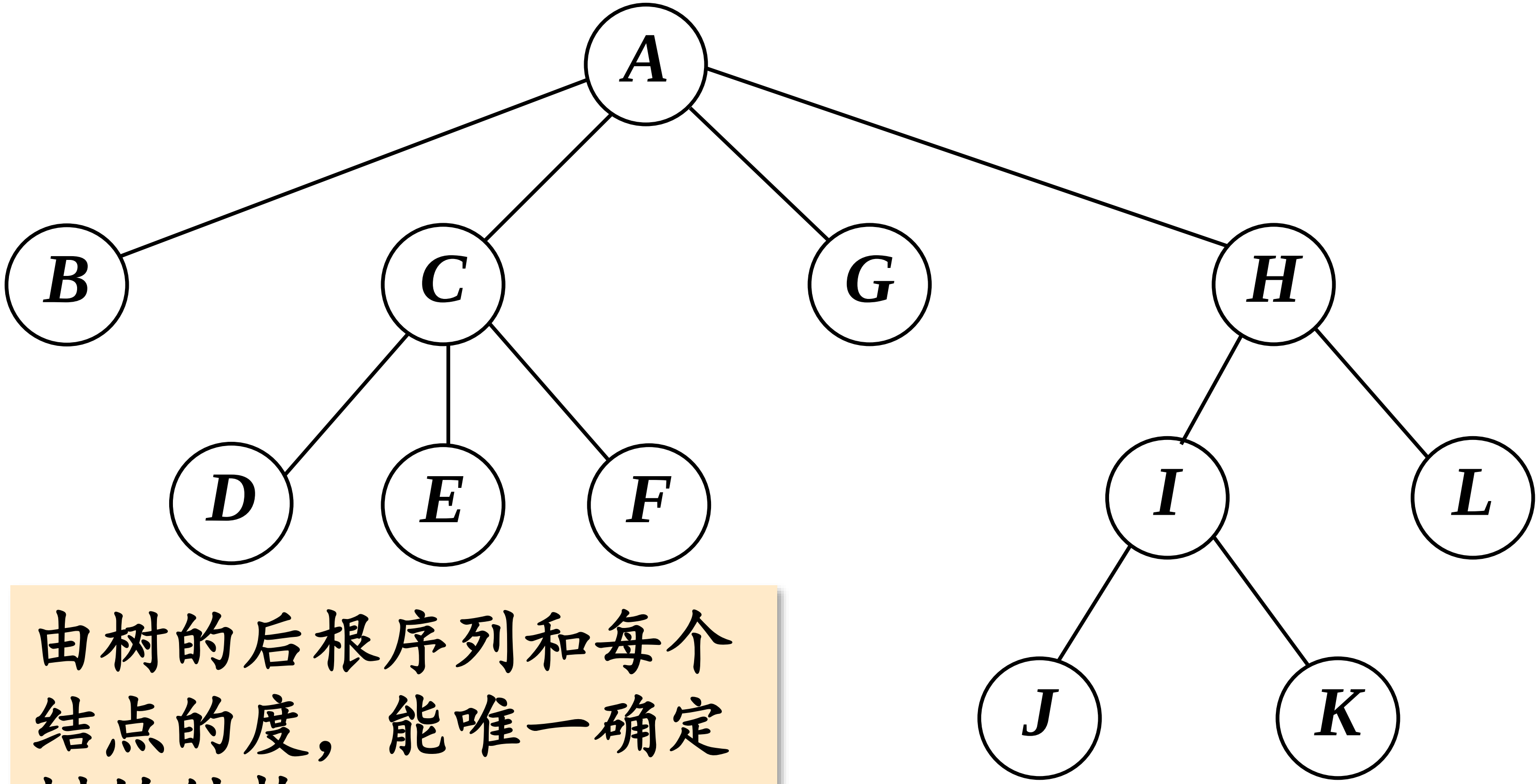
证毕。



树的后根序列+结点度

B

后根序列	B	D	E	F	C	G	J	K	I	L	H	A
结点度	0	0	0	0	3	0	0	0	2	0	2	4



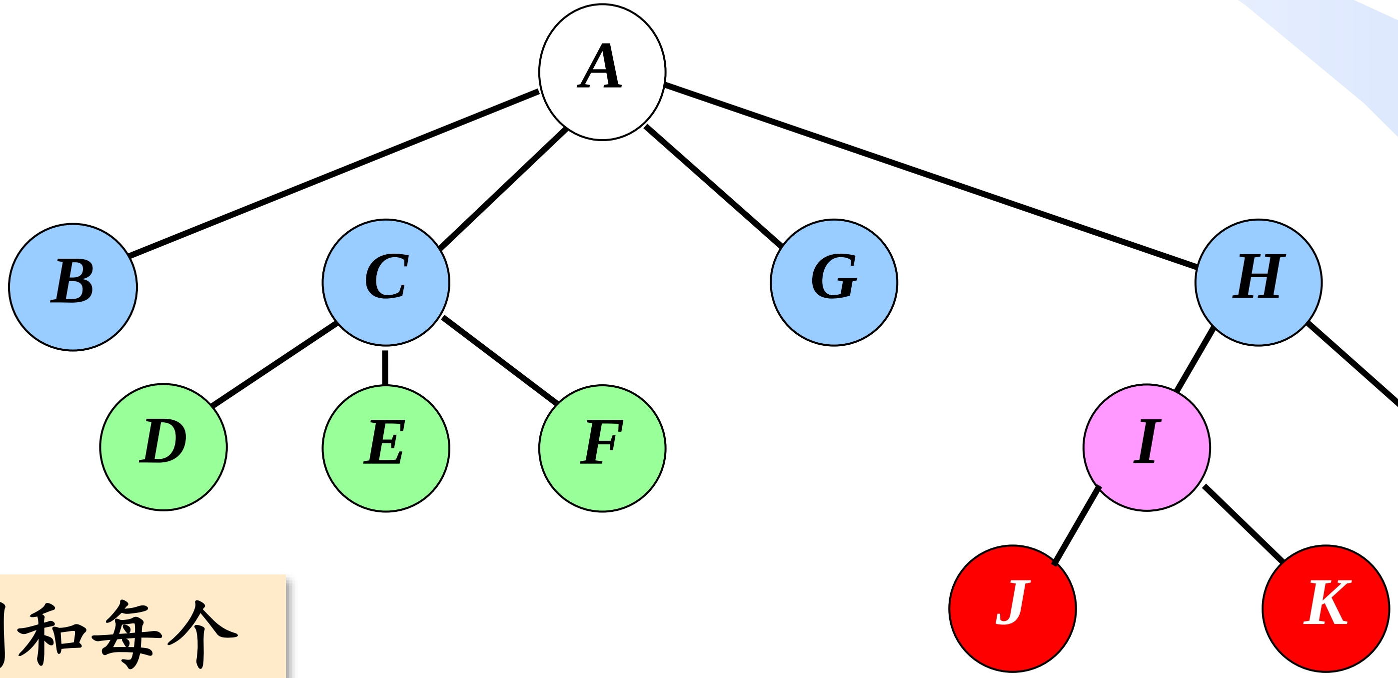
由树的后根序列和每个结点的度，能唯一确定树的结构。

反序列化：从左往右扫描，若结点X的度为k，则选择离X最近的k棵子树作为X的子树

树的层次序列+结点度

B

层次序列	A	B	C	G	H	D	E	F	I	L	J	K
结点度	4	0	3	0	2	0	0	0	2	0	0	0



由树的层次序列和每个结点的度，能唯一确定树的结构。