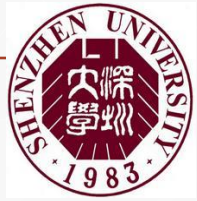
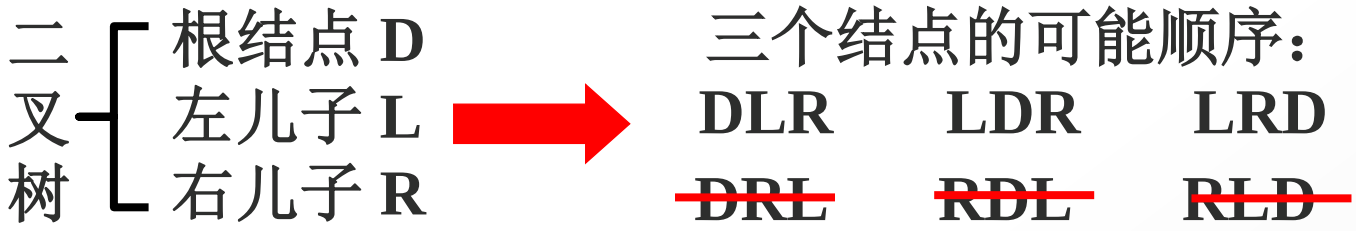
第五章 树与二叉树（二）



遍历二叉树

- 根据某种策略，按照一定的次序访问二叉树中的每一个结点，使每个结点被访问一次且只被访问一次。

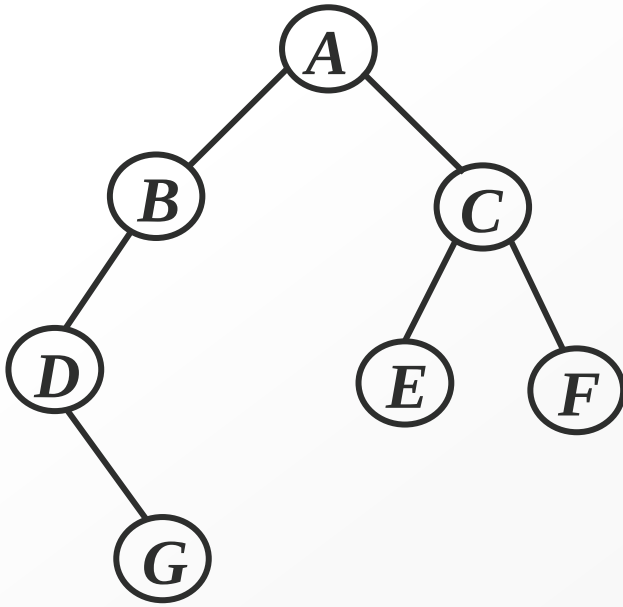


- 遍历结果是二叉树结点的线性序列。非线性结构线性化。
- 策略：左孩子结点一定要在右孩子结点之前访问
- 次序：先序（根）遍历、中序（根）遍历、后序（根）遍历
- 访问：抽象操作，可以是对结点进行的各种处理，这里简化为输出结点的数据。

先序遍历二叉树

- 若二叉树为空，则返回；否则，

  I. 访问根结点；
  II. 先序遍历根结点的左子树；
  III. 先序遍历根结点的右子树；



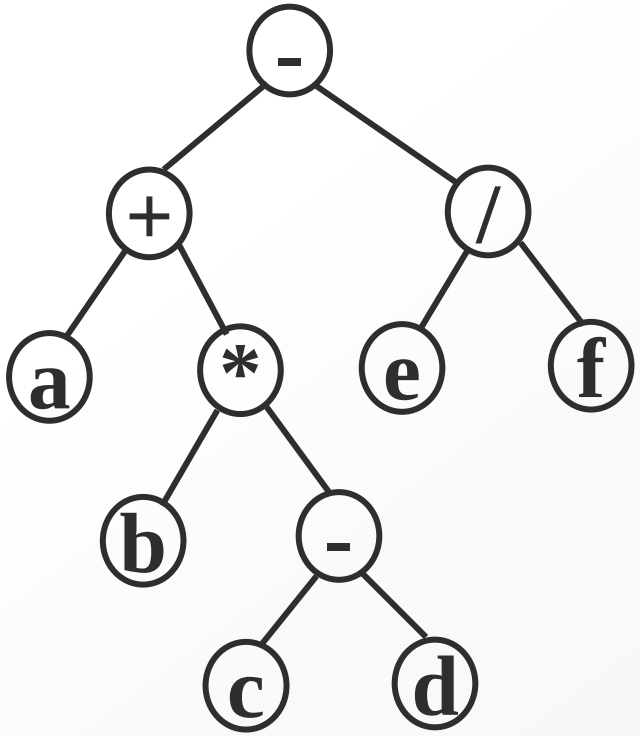
- 所得到的线性序列称为先序序列。

- 先序遍历序列为：*A B D G C E F*

先序遍历二叉树（cont.）

- 递归遍历算法–先序遍历

```
void  PreOrder (BiTree BT )
{
    if ( BT != NULL)
    {
        cout<< BT->data  ;
        PreOrder ( BT->lchild ) ;
        PreOrder ( BT->rchild ) ;
     }
}
```



a+b*(c-d)-e/f

✓ 先序序列：- + a * b – c d / e f

先序遍历二叉树（cont.）

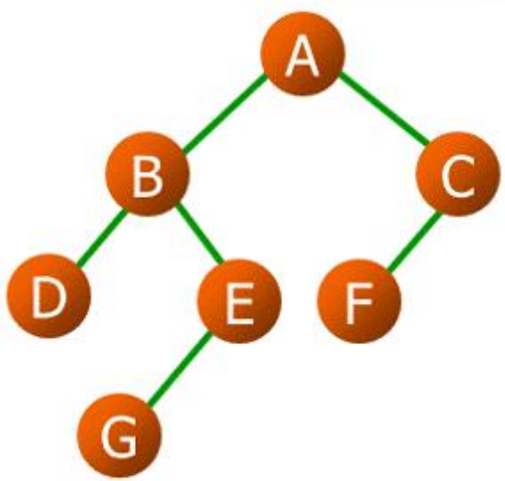
[特性]

- 先序遍历中，第一个输出节点必为根节点
- 先序遍历序列，由

  根 + 左子树先序遍历序列 +

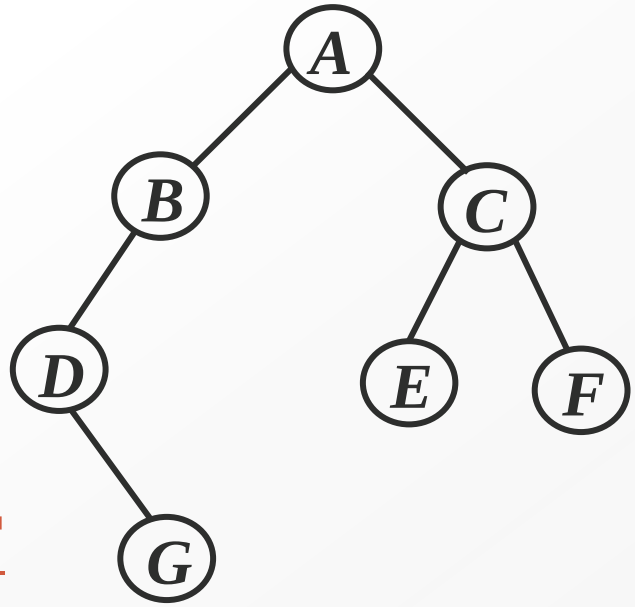
  右子树先序遍历序列

  组成
- 输出结果：**ABDEGCF**



中序遍历二叉树

- 若二叉树为空，则返回；否则，
  - I. 中序遍历根结点的左子树；
  - II. 访问根结点；
  - III. 中序遍历根结点的右子树；



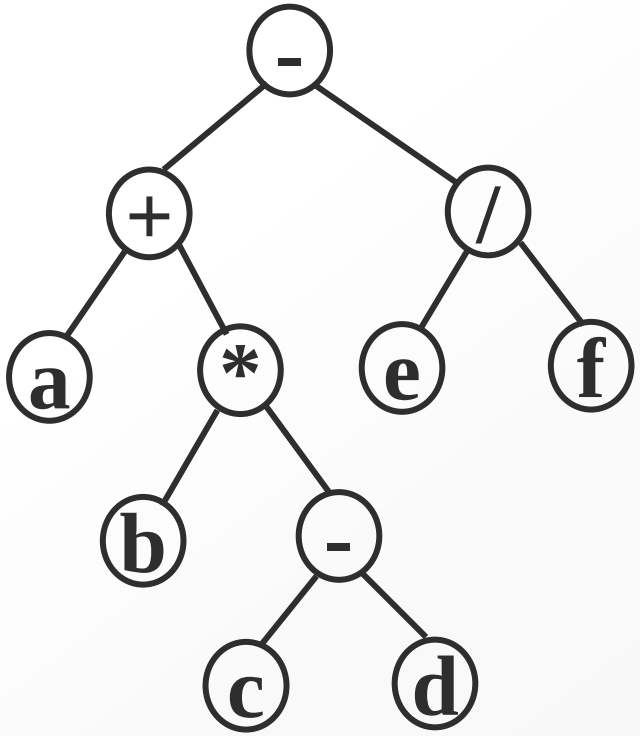
- 所得到的线性序列称为中序序列。

- 中序遍历序列为： D G B A E C F

中序遍历二叉树（cont.）

- 递归遍历算法–中序遍历

```
void  InOrder (BiTree BT )
{
    if ( BT != NULL)
    {
        InOrder ( BT->lchild ) ;
        cout<< BT->data  ;
        InOrder ( BT->rchild ) ;
     }
}
```



a+b*(c-d)-e/f

✓ 中序序列： a + b * c – d – e / f

中序遍历二叉树（cont.）

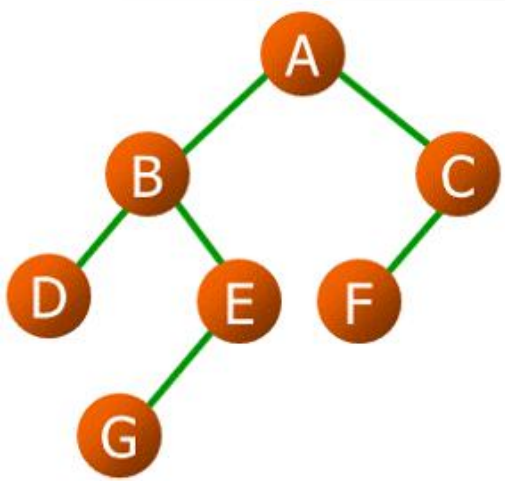
[特性]

- 中序遍历中，先于根节点输出的节点为左子树L的节点，后于根节点输出的节点为右子树R的节点
- 中序遍历序列，由

  左子树中序遍历序列 + 根 +

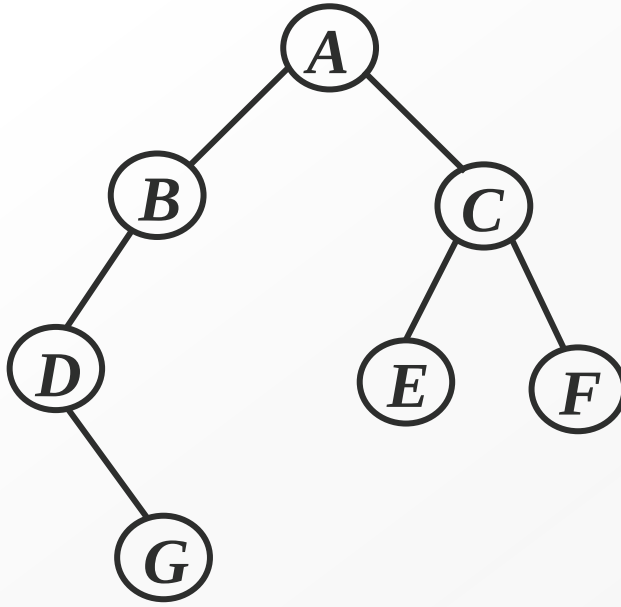
  右子树中序遍历序列

  组成
- 输出结果： **DBGEAFC**



后序遍历二叉树

- 若二叉树为空，则返回；否则，
  - I. 后序遍历根结点的左子树；
  - II. 后序遍历根结点的右子树；
  - III. 访问根结点；



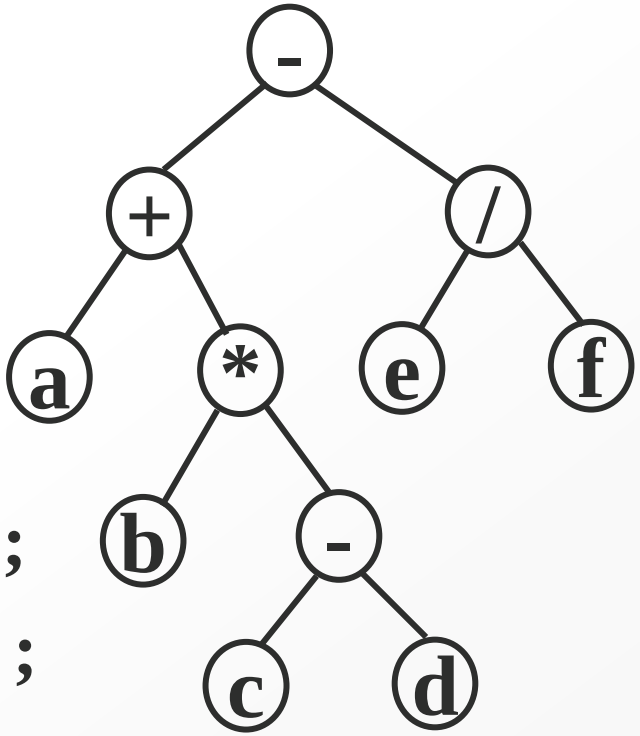
- 所得到的线性序列称为后序序列。

- 后序遍历序列为： ***G D B E F C A***

后序遍历二叉树（cont.）

- 递归遍历算法–后序遍历

```
void  PostOrder (BiTree BT )
{
    if ( BT != NULL)
    {
        PostOrder ( BT->lchild ) ;
        PostOrder ( BT->rchild ) ;
        cout<< BT->data  ;
     }
}
```



a+b*(c-d)-e/f

✓ 后序序列：a b c d - * + e f / -

后序遍历二叉树（cont.）

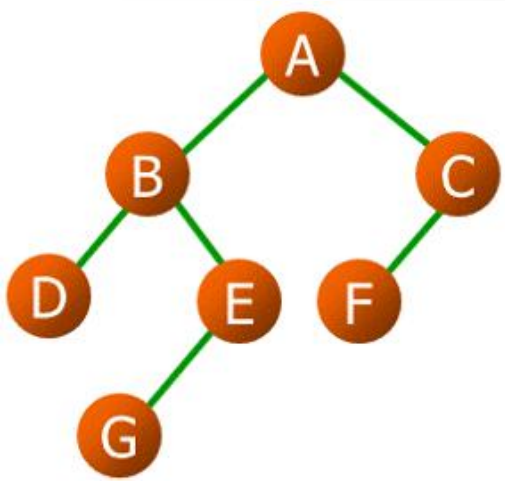
[特性]

- 后序遍历中，最后一个输出节点必为根节点
- 后序遍历序列中，由

  左子树后序遍历序列 +

  右子树后序遍历序列 + 根

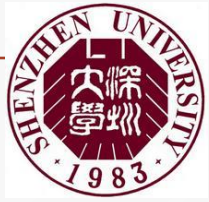
  组成
- 输出结果： **DGEBFCA**



根据先、中序遍历求序列二叉树

- 如果已知一颗二叉树的先序遍历和中序遍历序列，则可以唯一确定这棵二叉树
- 算法：

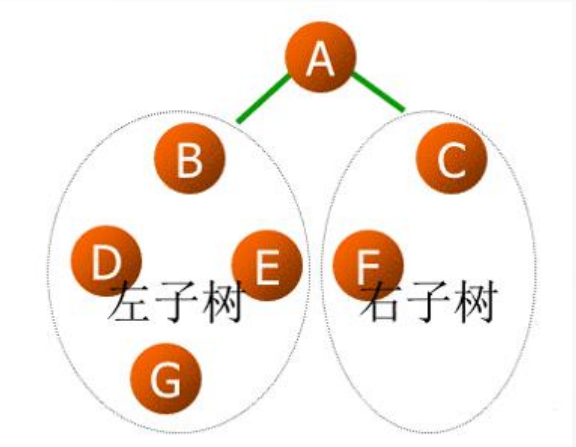
1. 在先序遍历序列中，第一个节点为根节点D，之后跟左子树先序遍历序列 + 右子树先序遍历序列
2. 在中序遍历序列中，根节点D左边的节点归左子树L，根节点D右边的节点归右子树R
3. 对每个子树反复使用1,2两步，直到确定二叉树



根据先、中序遍历求序列二叉树（cont.）

- 已知一棵二叉树的先序遍历序列为：ABDEGCF，中序遍历序列为：DBGEAFC，请画出这棵二叉树

- 根据先序遍历序列，可知根节点为A；再根据中序遍历序列可知，左子树由DBGE组成，右子树由FC组成



根据先、中序遍历求序列二叉树（cont.）

- 从整棵二叉树的先序遍历序列**ABDEGCF**可知，左子树的先序遍历序列为**BDEG**，右子树的先序遍历为**CF**

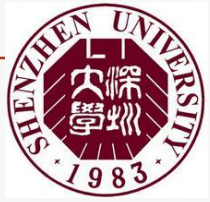
- 从整棵二叉树的中序遍历序列**DBGEAFC**可知，左子树的中序遍历为**DBGE**，右子树的中序遍历为**FC**

- 对左、右子树分别进行（上一页的）处理

根据后、中序遍历求序列二叉树

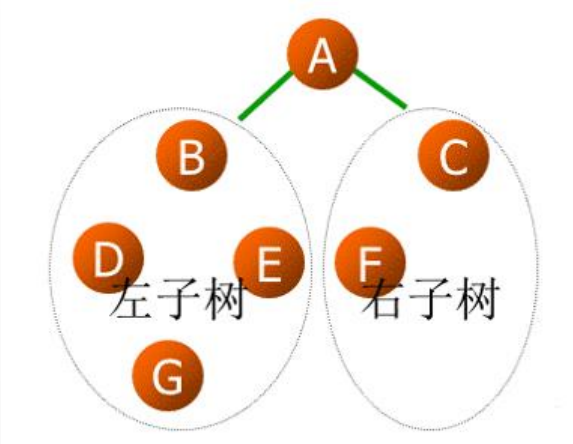
- 如果已知一棵二叉树的后序遍历和中序遍历，则可以唯一确定这棵二叉树
- 算法：

1. 在后序遍历序列中，最后一个节点为根节点D，前为左子树后序遍历序列 + 右子树后序遍历序列
2. 在中序遍历序列中，根节点D左边的节点归为左子树L，根节点D右边的节点归为右子树R
3. 对每个子树反复使用1,2两步，直到确定二叉树



根据后、中序遍历求序列二叉树（cont.）

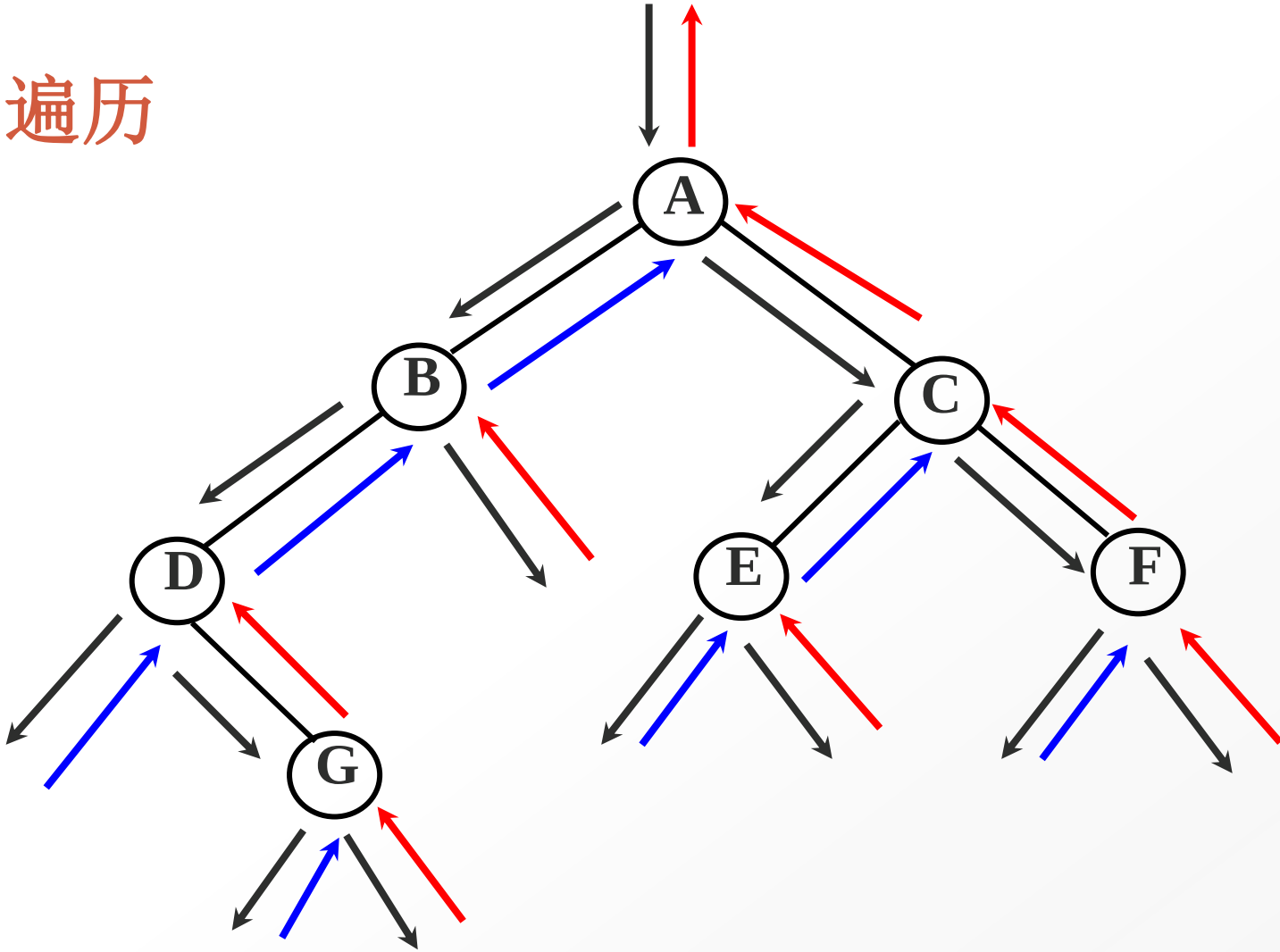
- 已知一棵二叉树的后序遍历序列为：DGEBFCA，中序遍历序列为：DBGEAFC，请画出这棵二叉树
- 根据后序遍历序列，可知根节点为A；再根据中序遍历序列可知，左子树由DBGE组成，右子树由FC组成



根据后、中序遍历求序列二叉树（cont.）

- 从整棵二叉树的后序遍历序列DGEBFCA可知，左子树的后序遍历序列为DGEB，右子树的后序遍历为FC
- 从整棵二叉树的中序遍历序列DBGEAFC可知，左子树的中序遍历为DBGE，右子树的中序遍历为FC
- 对左、右子树分别进行（上一页的）处理

二叉树的遍历

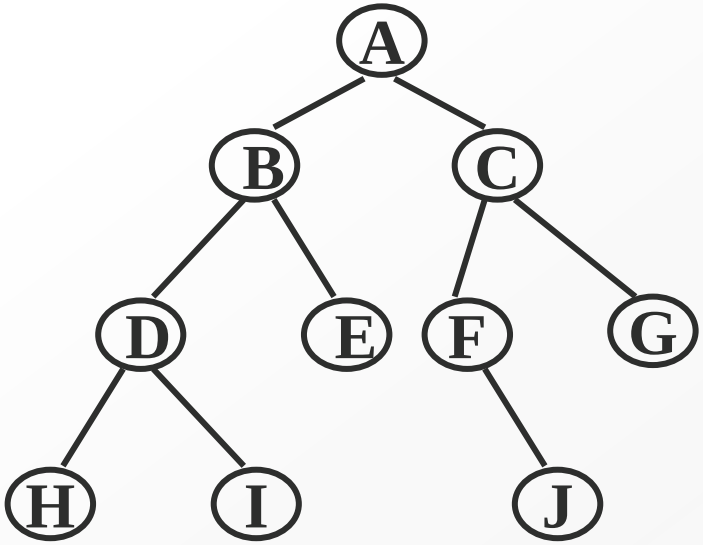


- 先序、中序和后序遍历过程中经过结点的路线一样，只是访问各结点的时机不同。

二叉树的非递归遍历（cont.）

- **中序遍历的非递归算法**

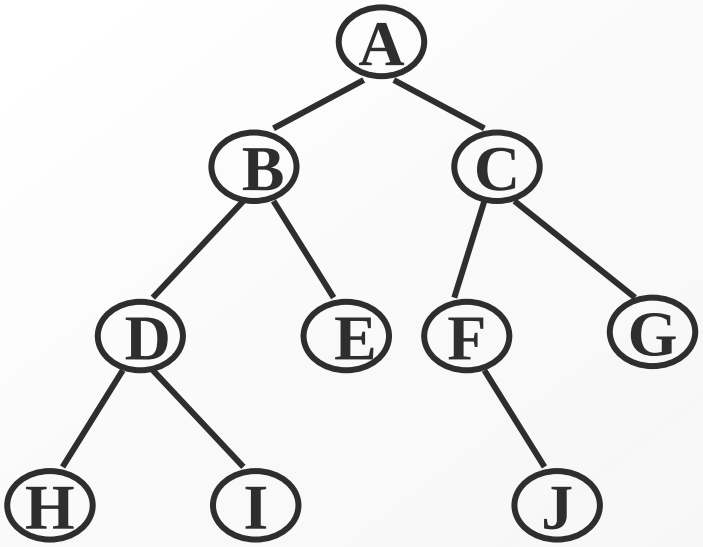
1. 初始化栈s;
2. 循环直到root为空且栈s为空
   2.1 当root不空时循环
      2.1.1 将指针root的值保存到栈中;
      2.1.2 继续遍历root的左子树
   2.2 如果栈s不空，则
      2.2.1 将栈顶元素弹出至root;
         2.2.2 输出root->data;
      2.2.3 准备遍历root的右子树;



二叉树的非递归遍历（cont.）

- 中序遍历的非递归算法

```
void InOrder(BTREE  root)
{   top= -1;     //采用顺序栈，并假定不会发生上溢
    while (root!=NULL || top!= -1) {
        while (root!= NULL) {
            s[++top]=root;
            root=root->lchild;
        }
        if (top!= -1) {
            root=s[top--];
            cout<<root->data;
            root=root->rchild;
        }
    }
}
```

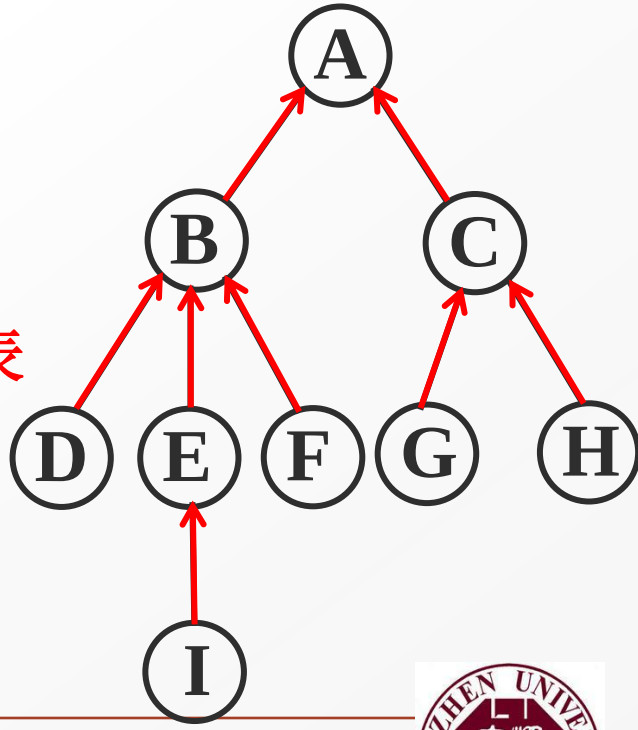


树的存储结构

1. 双亲表示法

- 每个结点（根结点除外）都只有唯一的双亲结点
- 因此，可以把各个结点按层序存储在一维数组中，同时记录其唯一双亲结点在数组中的下标。
- 结点结构定义

```
struct node {
  T data;    //数据域
  int parent;    //指针域，双亲的下标
};   //双亲表示实质上是一个静态链表
```



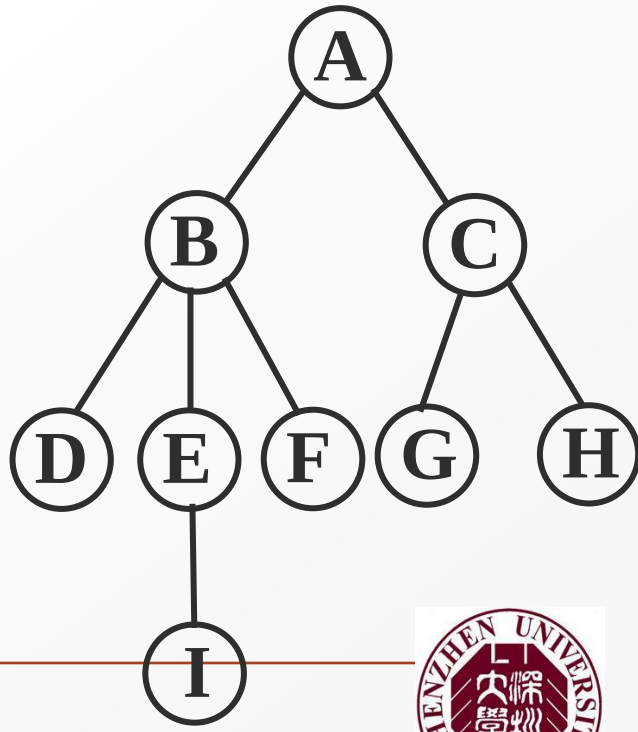
	1	2	3	4	5	6	7	8	9
data	A	B	C	D	E	F	G	H	I
parent	0	1	1	2	2	2	3	3	5

树的存储结构（cont.）

1. 双亲表示法

- ✓ 存储特点：
  - I. 每个结点均保存父结点所在的数组单元下标
  - II. 兄弟结点的编号连续。
- ✓ 如何查找双亲结点和祖先？
- ✓ 如何查找孩子结点？
- ✓ 如何查找兄弟结点？

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



树的存储结构（cont.）

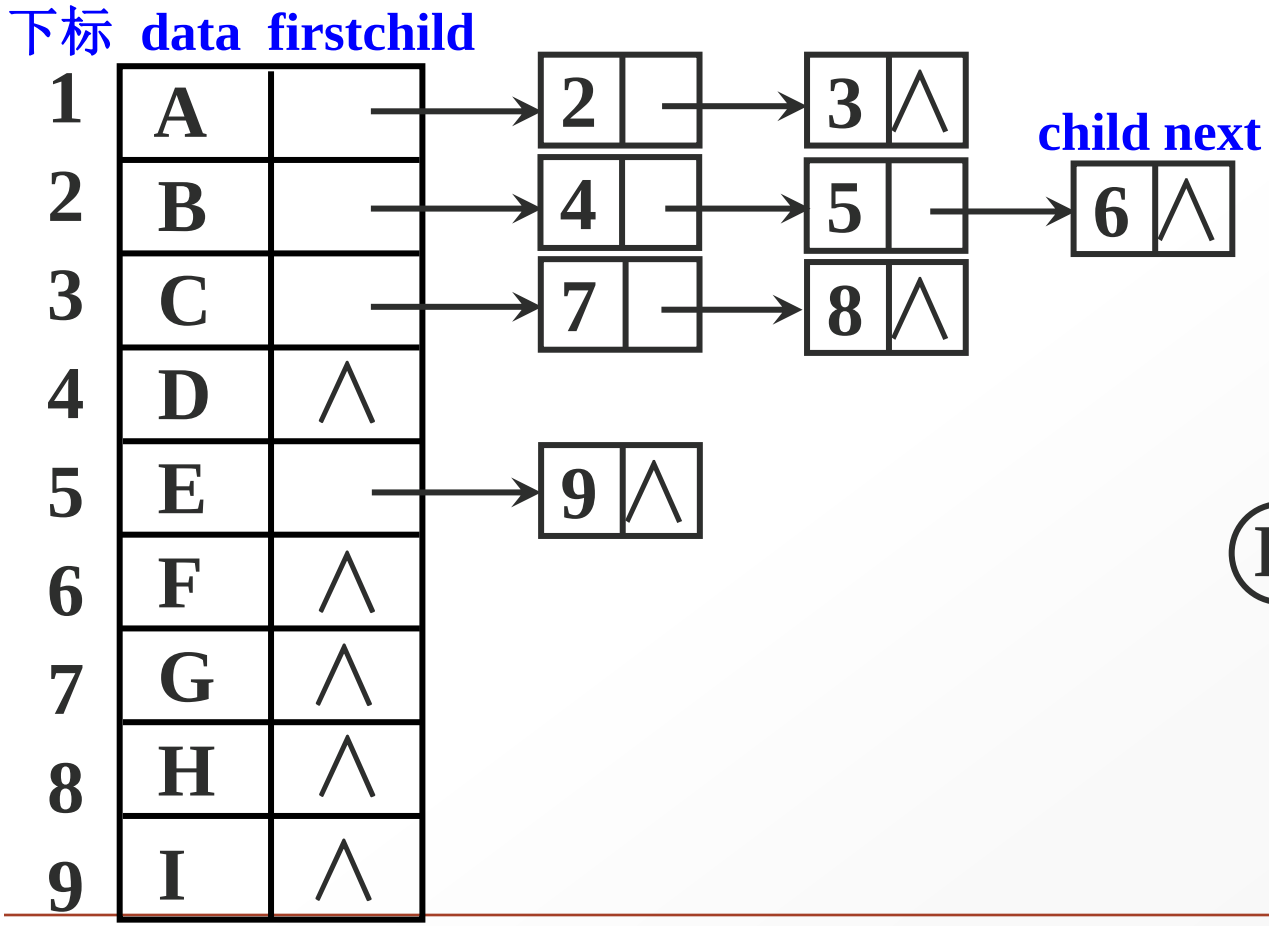
2. 孩子链表表示法

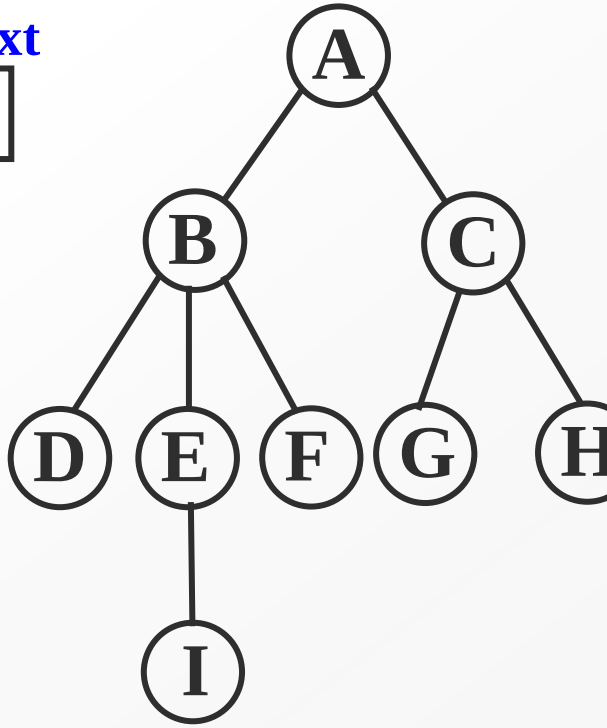
- ✓ 把每个结点的孩子排列起来，看成是一个线性表，且以单链表存储，则 n 个结点共有 n 个孩子链表。

- ✓ 再把每个单链表的头指针，组织成一个线性表，为了便于查找，采用顺序存储结构。

树的存储结构（cont.）

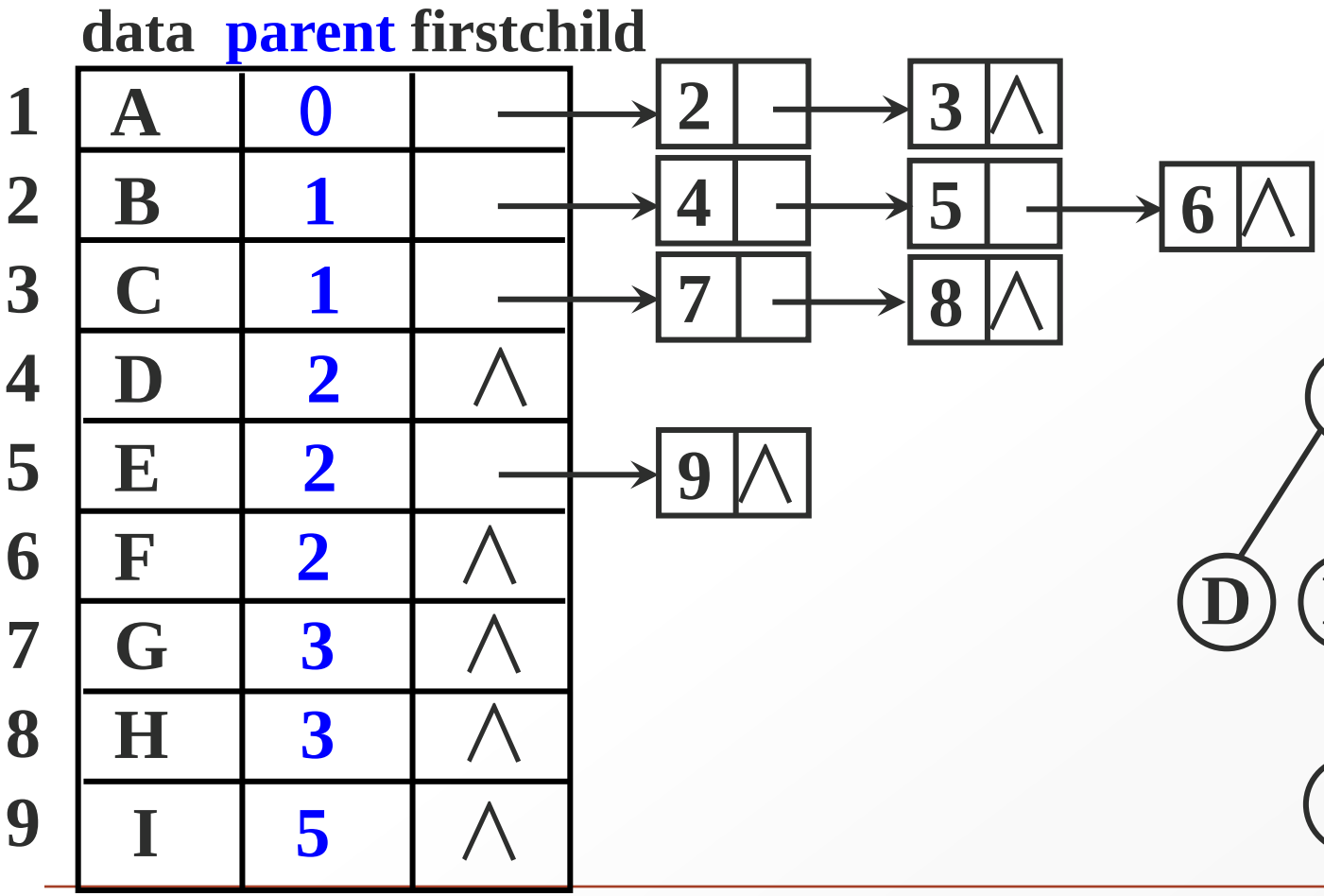
2. 孩子链表表示法

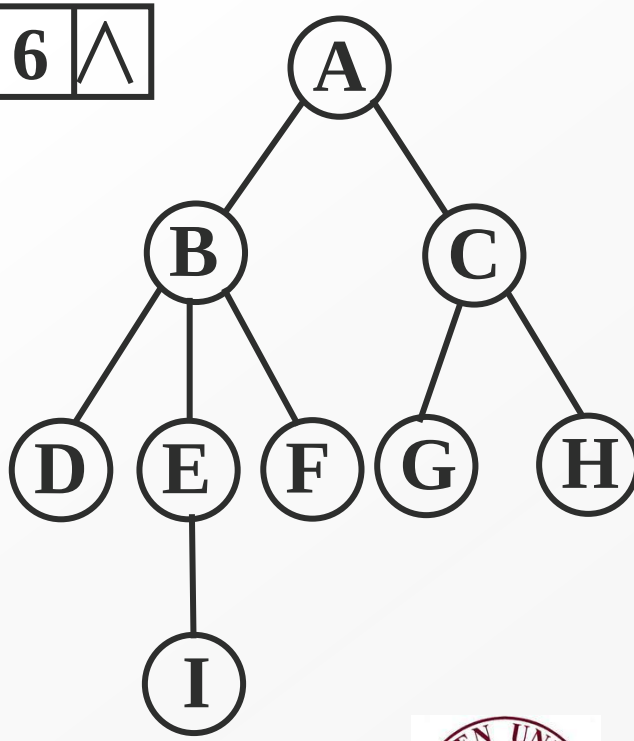




树的存储结构（cont.）

3. 双亲孩子表示法





树的存储结构（cont.）

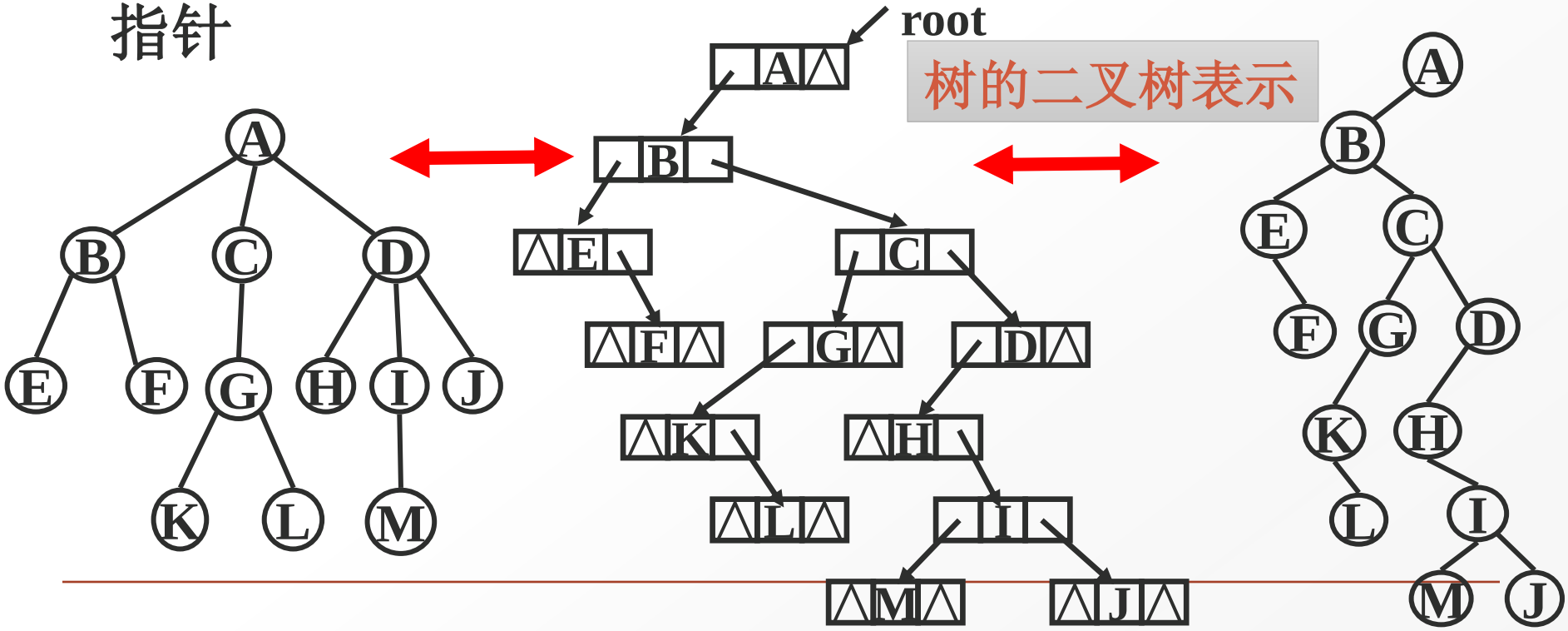
结点结构：

firstchild	data	rightsib

4. 孩子兄弟表示法

采用二叉链表（（左）孩子—（右）兄弟链表表示）

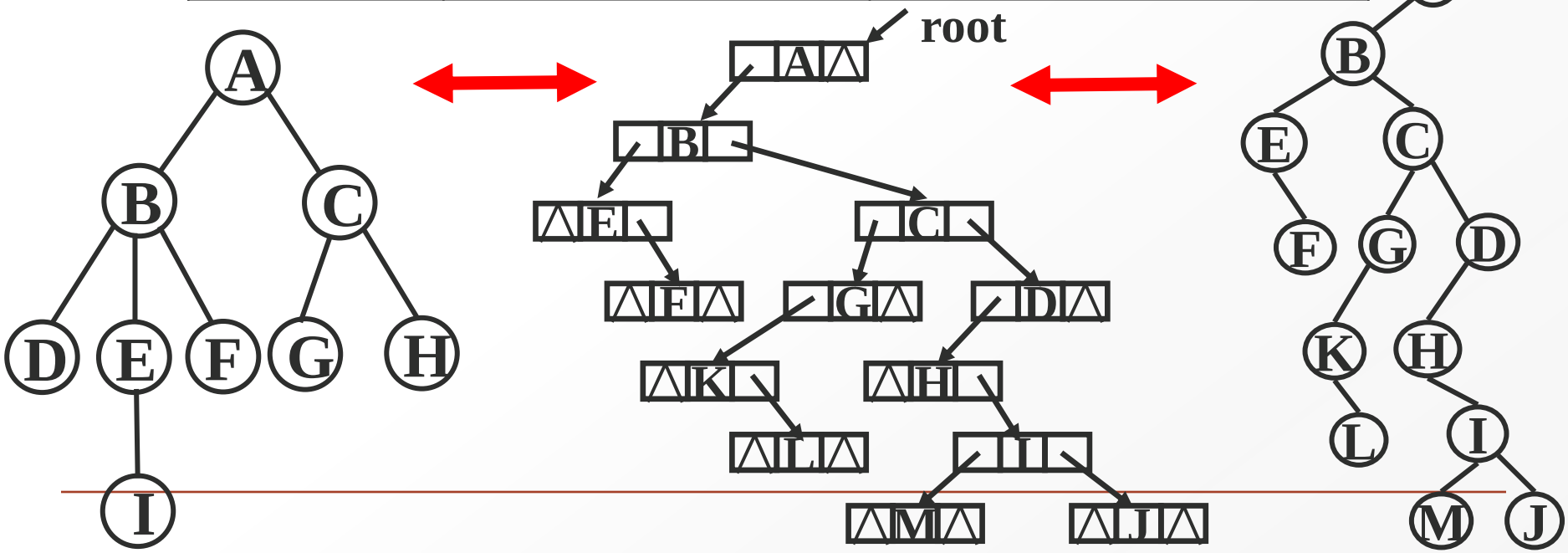
- 结点的右兄弟是唯一的
- 设置两个分别指向该结点的第一个孩子和右兄弟的指针



森林(树)与二叉树的对应关系

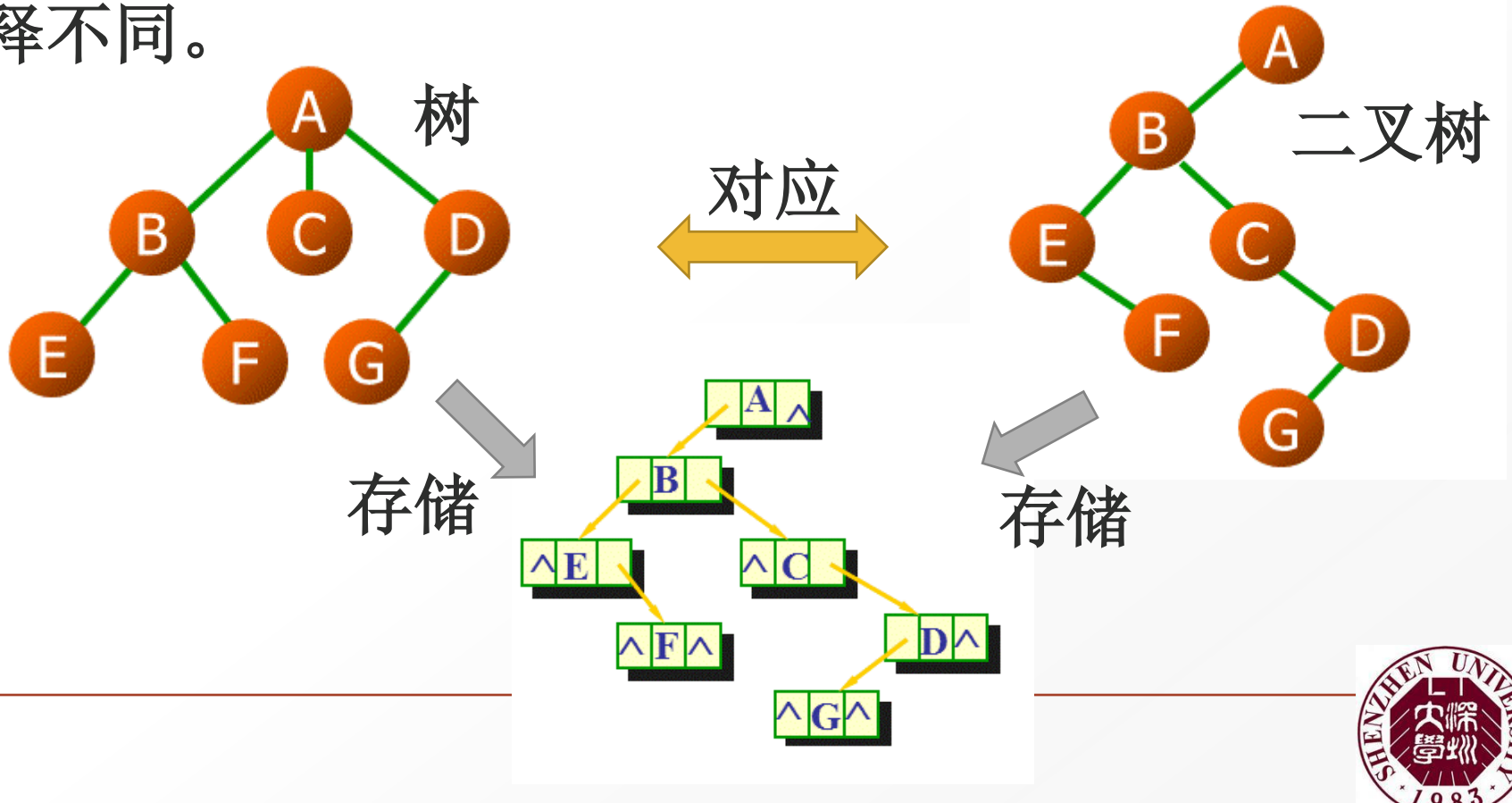
- 由于树与二叉树都可以采用二叉链表作存储结构，以二叉链表作为媒介可以导出树与二叉树的对应关系。

	树		二叉树
结点关系	兄弟关系	⟷	双亲和右孩子
	双亲和长子	⟷	双亲和左孩子



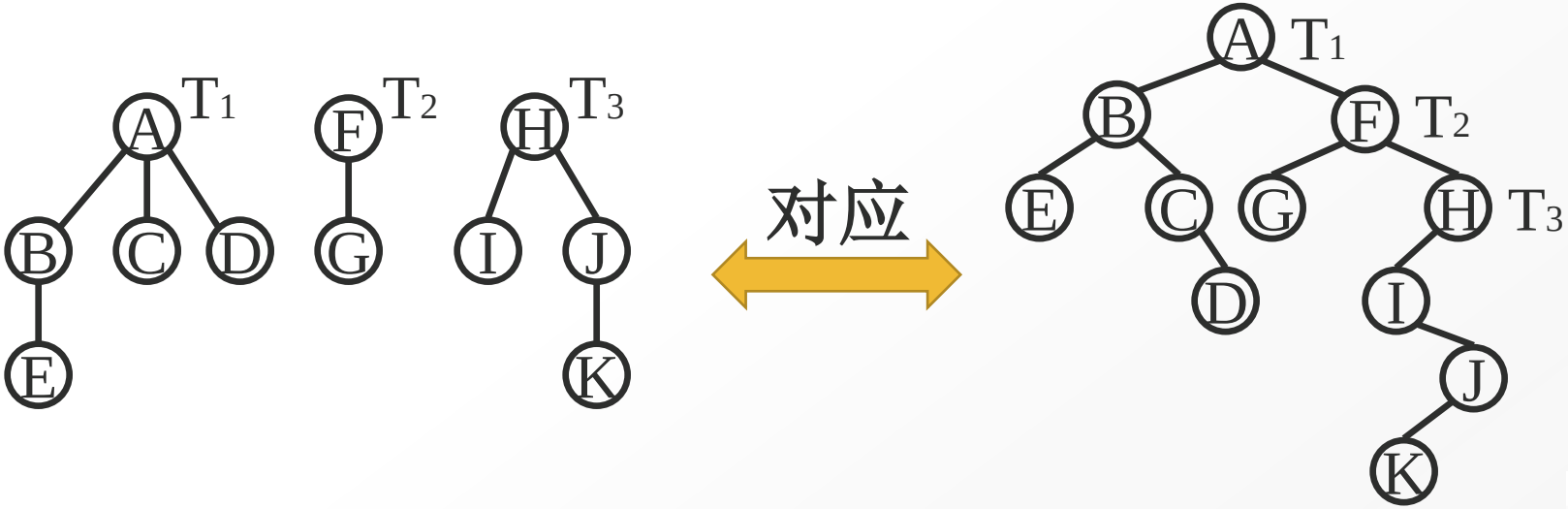
森林(树)与二叉树的对应关系（cont.）

- 任意给定一棵树，可以找到一个唯一的二叉树（没有右子树）与之对应。
- 从物理结构来看，它们的二叉链表是相同的，只是解释不同。



森林(树)与二叉树的对应关系（cont.）

- 任何一个森林对应唯一的一株二叉树
  - 第一株树的根对应二叉树的根；
  - 第一株树的所有子树森林对应二叉树的左子树；
  - 其余子树森林对应二叉树的右子树；



森林(树)与二叉树的对应关系（cont.)

1. 连线：

- 把每株树的各兄弟结点连起来；
- 把各株树的根结点连起来（视为兄弟）

2. 抹线：

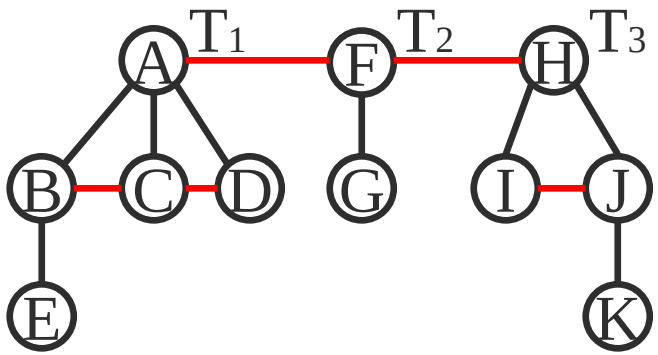
- 对于每个结点，只保留与其最左儿子的连线，抹去该结点与其它结点之间的连线

3. 旋转：

- 按顺时针旋转45度角（左链竖画，右链横画）

森林(树)与二叉树的对应关系（cont.）

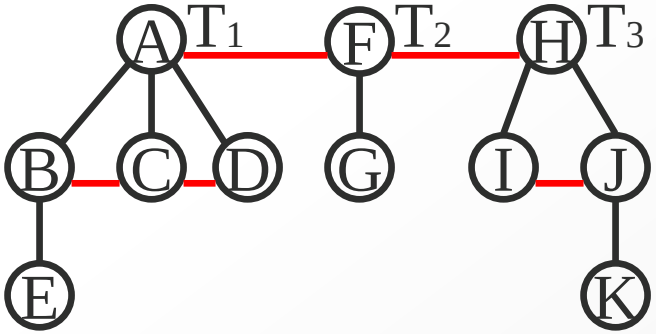
1. 连线：



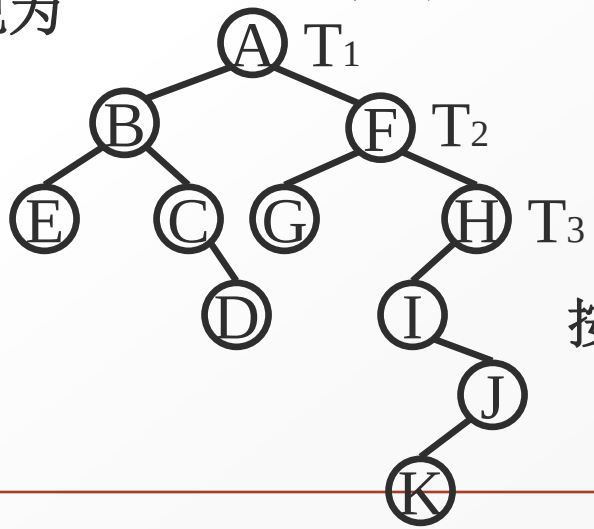
① 把每株树的各兄弟结点连起来

② 把各株树的根结点连起来（视为兄弟）

2. 抹线：



对于每个结点，只保留与其最左儿子的连线，抹去其他连线



3. 旋转：

按顺时针旋转45度角