数据结构（C语言版）（第2版）

树和二叉树

遍历二叉树

主讲教师：汪红松

教 学 内 容 Contents

1 树和二叉树的定义

2 二叉树的性质和存储结构

3 遍历二叉树

4 线索二叉树

5 树和森林

6 哈夫曼树及其应用

一、遍历二叉树和线索二叉树

1.遍历定义

二叉树的遍历是指从根结点出发，按照某种次序访问二叉树中的所有结点，使得每个结点被访问一次且仅被访问一次。

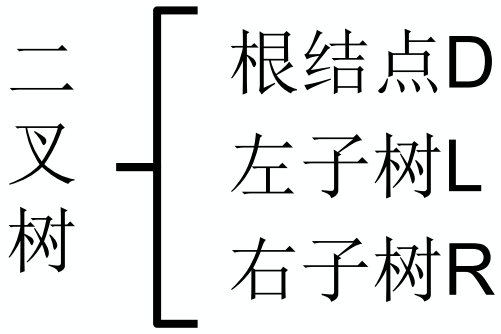
抽象操作，可以是对结点进行的各种处理，这里简化为输出结点的数据。

前序遍历
中序遍历
后序遍历
层序遍历

二叉树遍历操作的结果？ ⟹ 非线性结构线性化

一、遍历二叉树和线索二叉树

2. 二叉树的组成：



二叉树的遍历方式：

DLR、LDR、LRD、

DRL、RDL、RLD

如果限定先左后右，则二叉树遍历方式有三种：

前序：**DLR**

中序：**LDR**

后序：**LRD**

层序遍历：按二叉树的层序编号的次序访问各结点。

3. 二叉树的遍历操作

遍历的操作定义是递归的

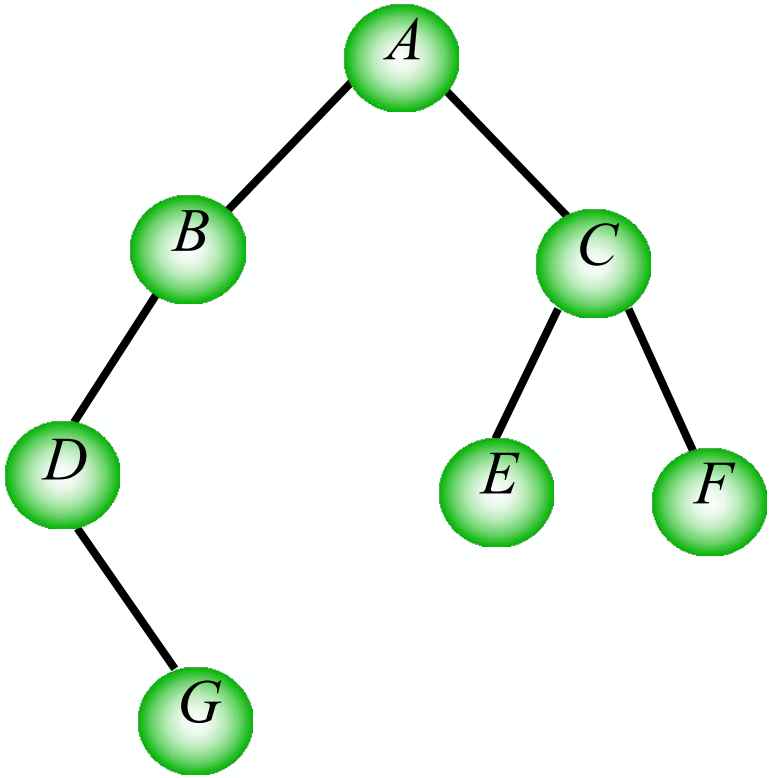
（1）前序（根）遍历

若二叉树为空，则空操作返回；
否则：

①访问根结点；

②前序遍历根结点的左子树；

③前序遍历根结点的右子树。



前序遍历序列：*A B D G C E F*

3. 二叉树的遍历操作

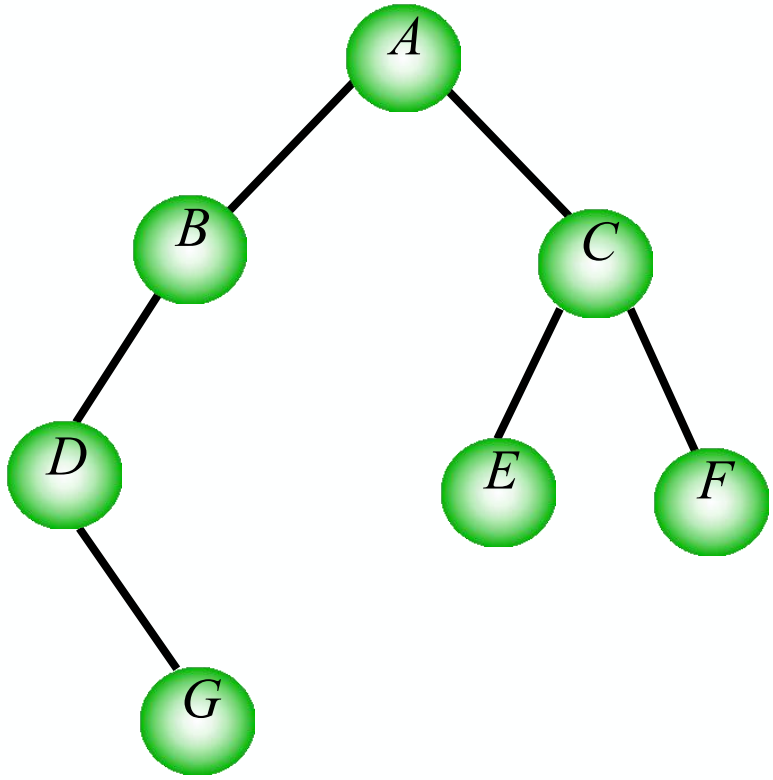
（2）中序（根）遍历

若二叉树为空，则空操作返回；
否则：

①中序遍历根结点的左子树；

②访问根结点；

③中序遍历根结点的右子树。



中序遍历序列：*D G B A E C F*

3. 二叉树的遍历操作

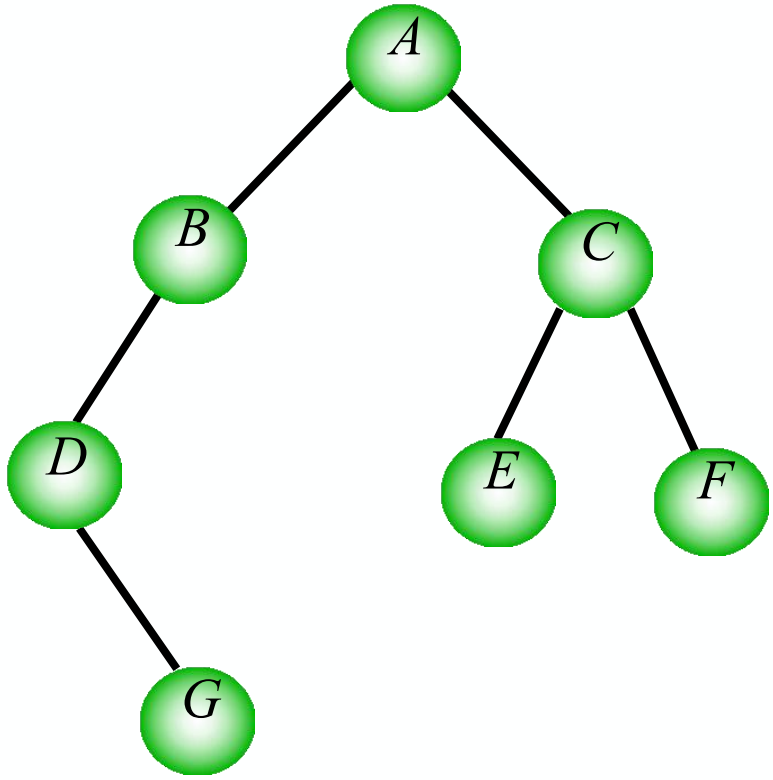
（3）后序（根）遍历

若二叉树为空，则空操作返回；
否则：

①后序遍历根结点的左子树；

②后序遍历根结点的右子树；

③访问根结点。



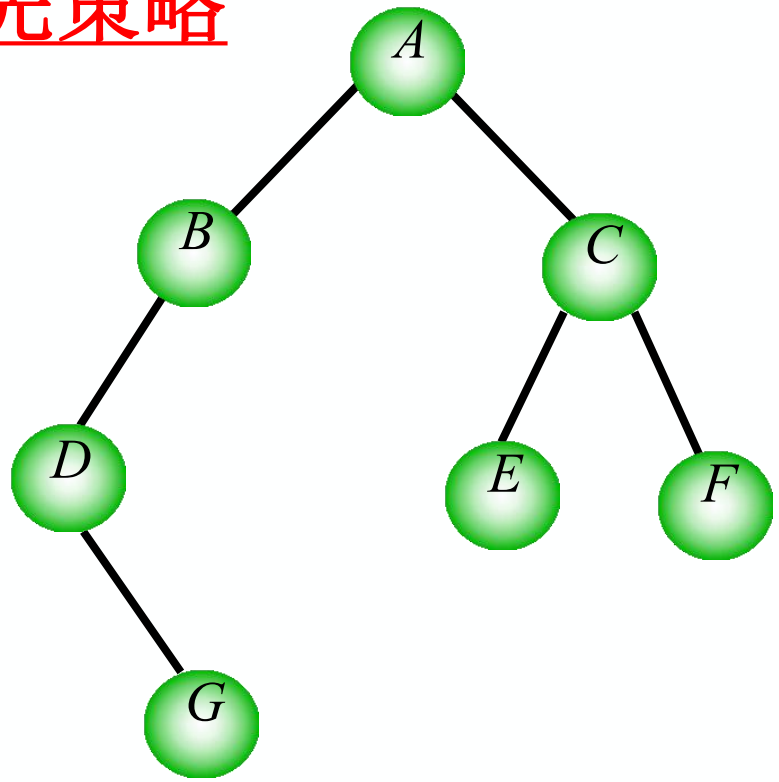
后序遍历序列：*G D B E F C A*

3. 二叉树的遍历操作

前序(中序、后序)遍历都是深度优先策略，层序遍历则是广度优先策略

（4）层序遍历

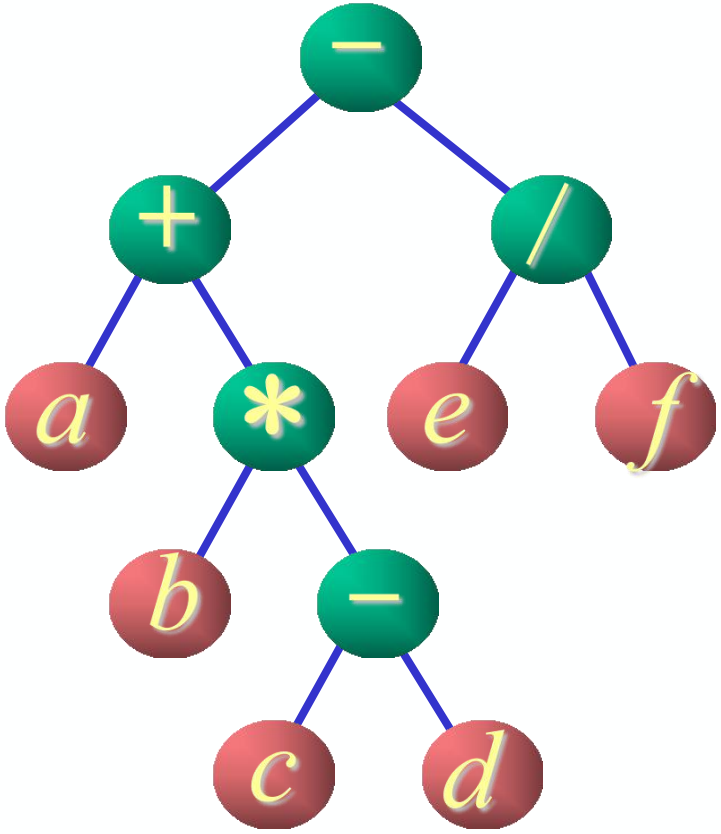
二叉树的层次遍历是指从二叉树的第一层（即根结点）开始，从上至下逐层遍历，在同一层中，则按从左到右的顺序对结点逐个访问。



层序遍历序列：*A B C D E F G*

二叉树遍历操作例子

表达式 a + b * (c - d) – e / f 对应的二叉树



前序遍历结果：- + a * b - c d / e f

中序遍历结果：a + b * c - d - e / f

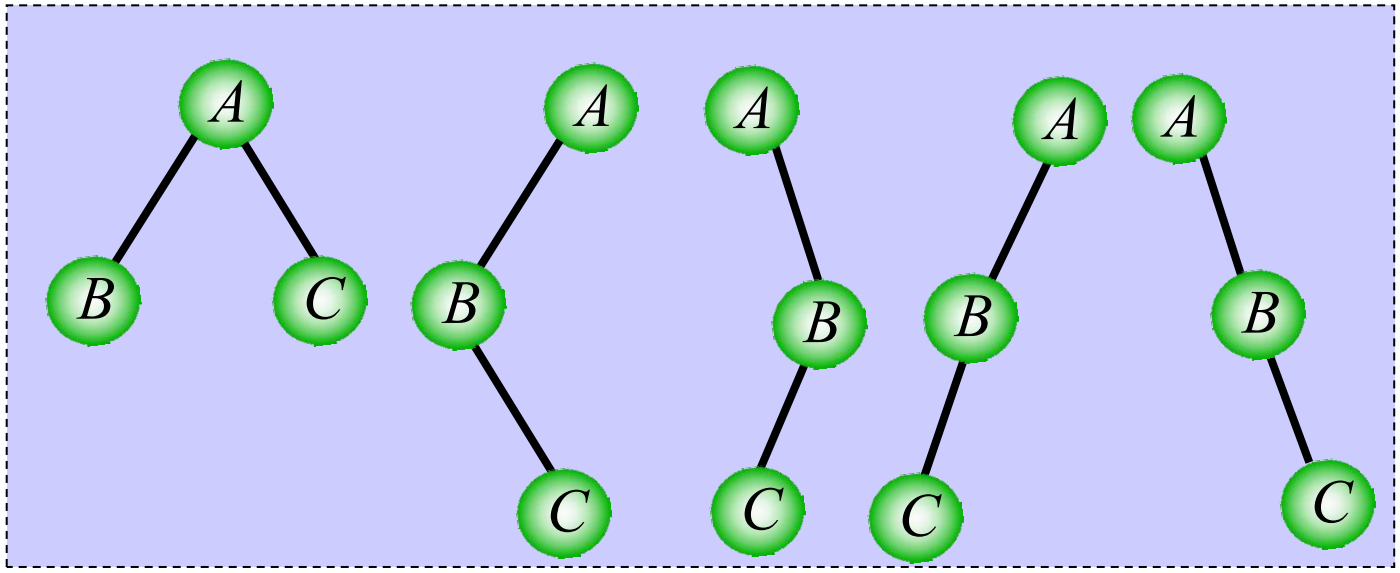
后序遍历结果：a b c d - * + e f / -

表达式的前序、中序和后序遍历为表达式前缀表示（波兰式）、中缀表示和后缀表示（逆波兰式）。

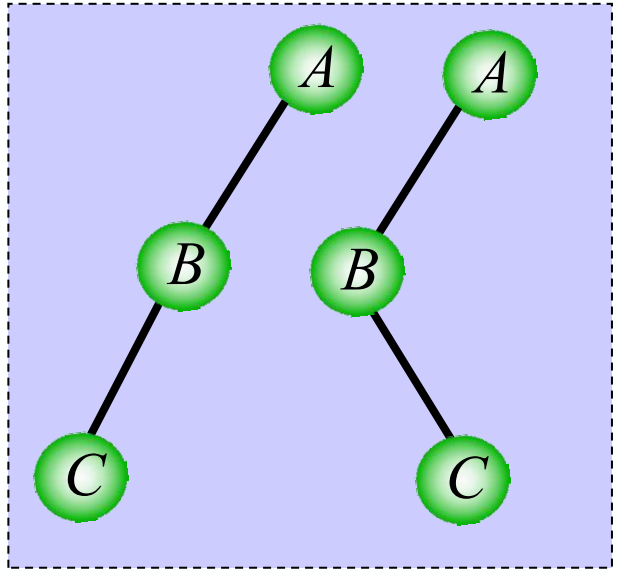
3. 二叉树的遍历操作

若已知一棵二叉树的前序（或中序，或后序，或层序）序列，能否唯一确定这棵二叉树呢？

例：已知前序序列为ABC，则可能的二叉树有5种。若其后序遍历序列为CBA，则下列二叉树都满足条件。



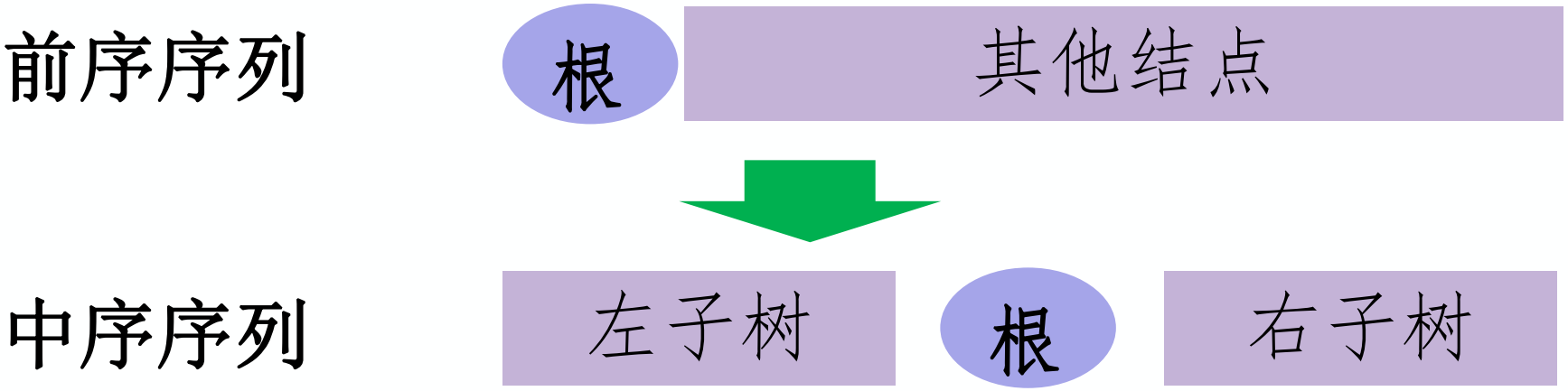
(a)



(b)

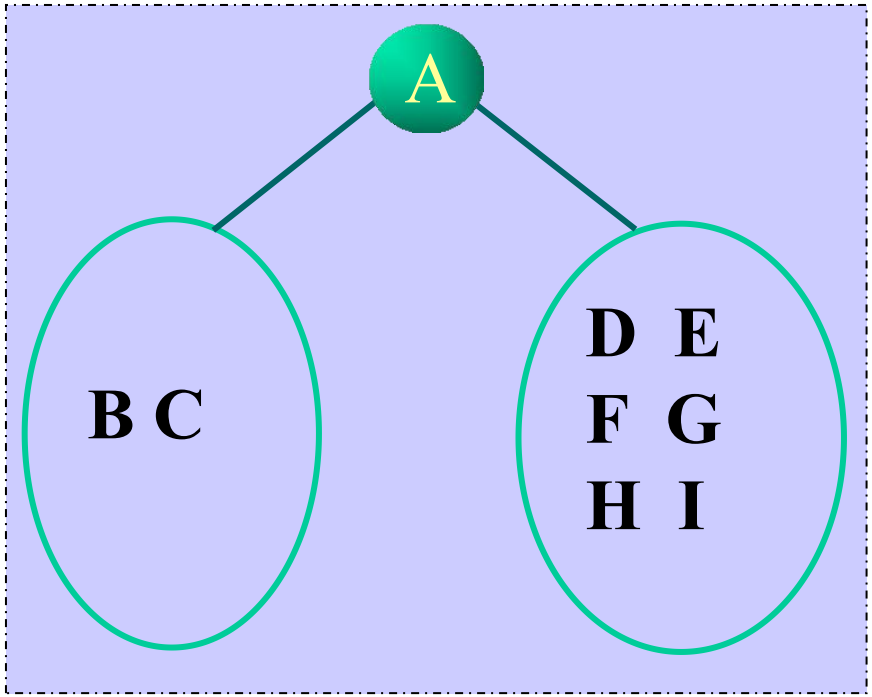
一、遍历二叉树和线索二叉树

若已知一棵二叉树的前序序列和中序序列，能否唯一确定这棵二叉树呢？怎样确定？



例如：已知一棵二叉树的前序遍历序列和中序遍历序列分别为**ABCDEFGHI** 和**BCAEDGHFI**，如何构造该二叉树呢?

一、遍历二叉树和线索二叉树



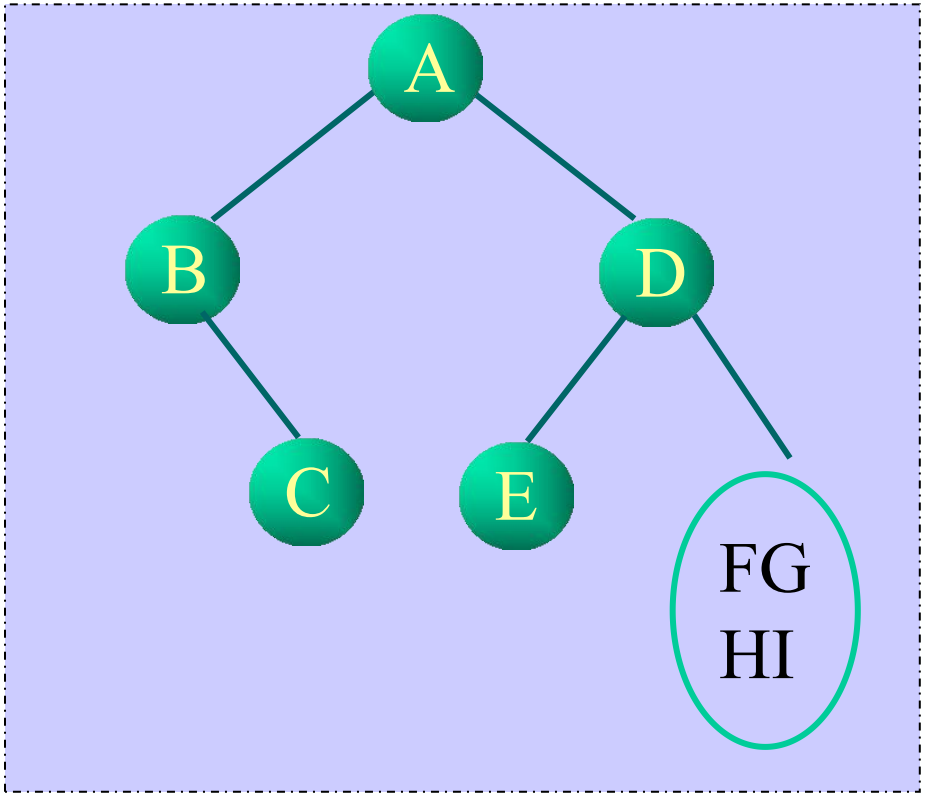
前序：A B C D E F G H I中序：B C A E D G H F I

前序： B C

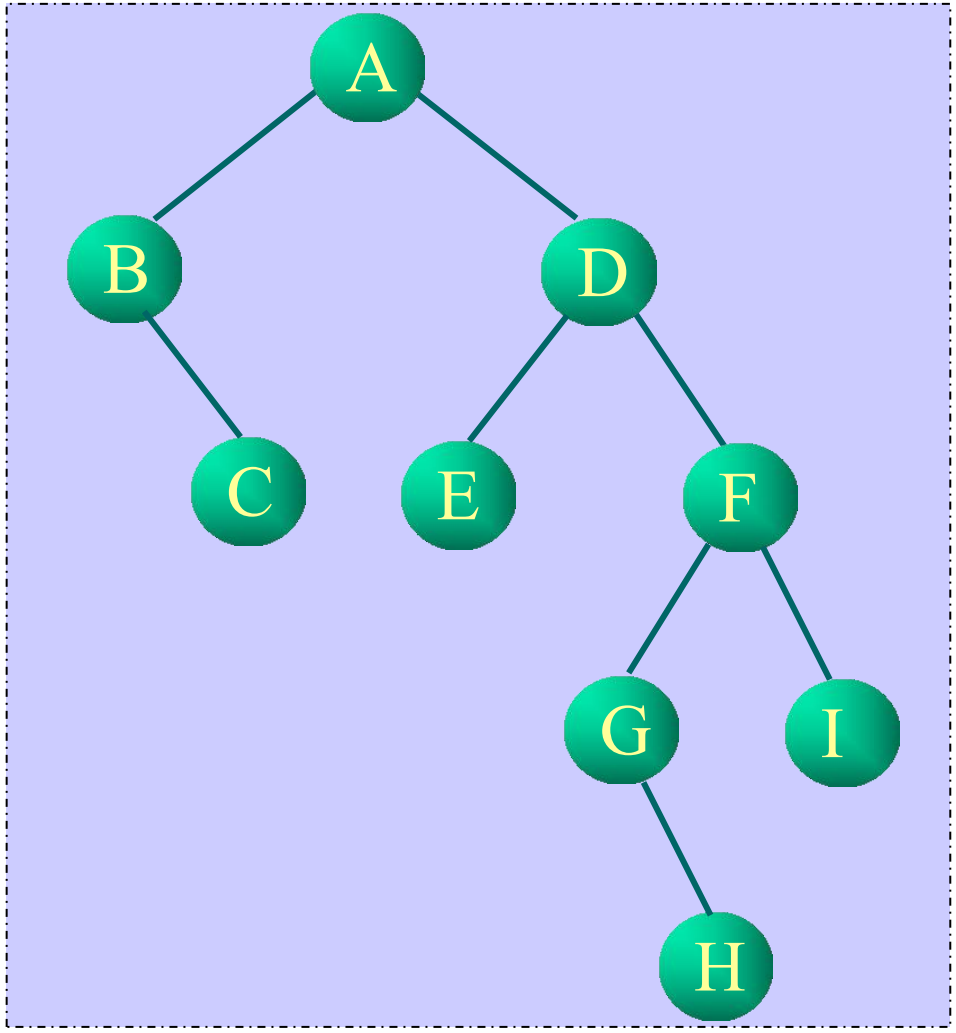
中序： B C

前序： D E F G H I

中序： E D G H F I

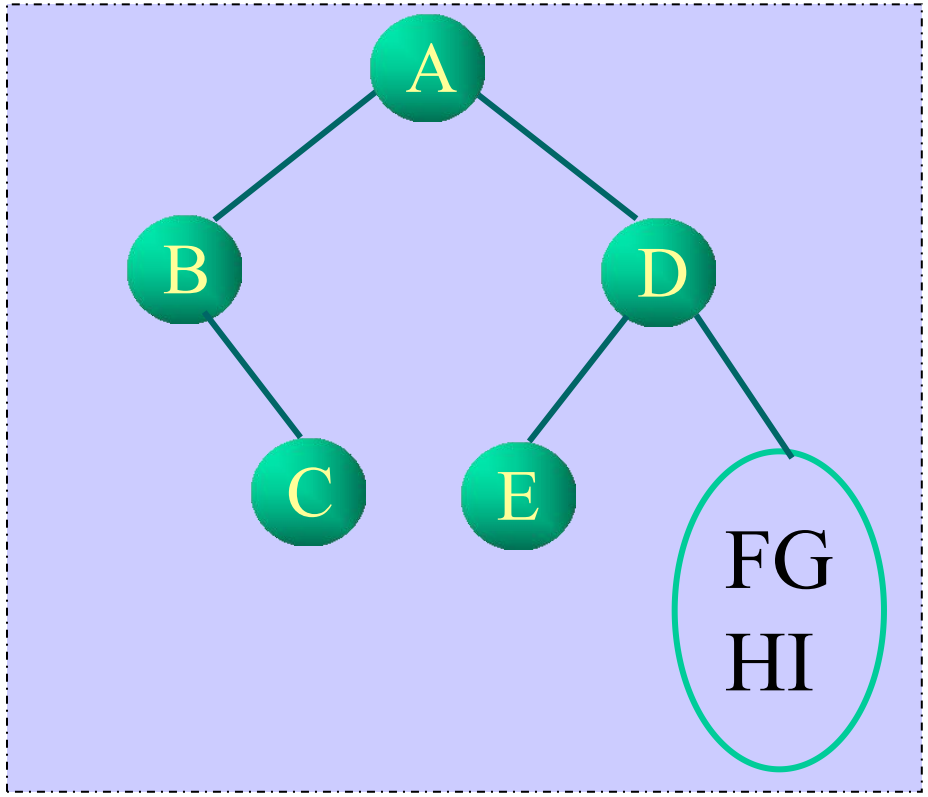


一、遍历二叉树和线索二叉树



前序：F G H I

中序：G H F I



前序：D E F G H I

中序：E D G H F I

二、遍历二叉树和线索二叉树

1. 根据遍历序列确定二叉树

已知一棵二叉树的前序序列和中序序列，构造该二叉树的过程如下：

（1）根据前序序列的第一个元素建立根结点；
（2）在中序序列中找到该元素，确定根结点的左右子树的中序序列；
（3）在前序序列中确定左右子树的前序序列；
（4）由左子树的前序序列和中序序列建立左子树；
（5）由右子树的前序序列和中序序列建立右子树。

由二叉树的前序序列和中序序列，或由其后序序列和中序序列均能唯一确定一棵二叉树。

二、遍历算法的分析

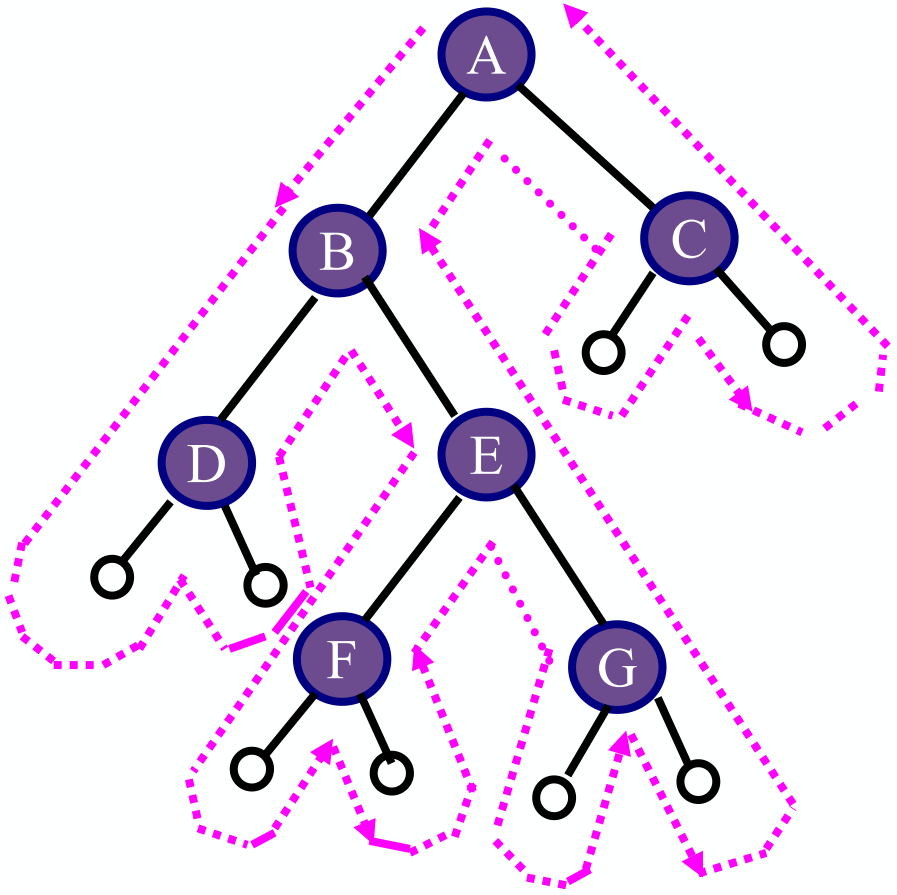
```
Status PreOrderTraverse(BiTree T){      //前序遍历
  if(T==NULL) return OK; //空二叉树
  else{
     cout<<T->data; //访问根结点
     PreOrderTraverse(T->lchild); //递归遍历左子树
     PreOrderTraverse(T->rchild); //递归遍历右子树    }
}
```

```
Status InOrderTraverse(BiTree T){   //中序遍历
  if(T==NULL) return OK; //空二叉树
  else{
     InOrderTraverse(T->lchild); //递归遍历左子树
     cout<<T->data; //访问根结点
     InOrderTraverse(T->rchild); //递归遍历右子树    }
}
```

```
Status PostOrderTraverse(BiTree T){   //后序遍历
  if(T==NULL) return OK; //空二叉树
  else{
     PostOrderTraverse(T->lchild); //递归遍历左子树
     PostOrderTraverse(T->rchild); //递归遍历右子树
     cout<<T->data; //访问根结点    }
}
```

二、遍历算法的分析

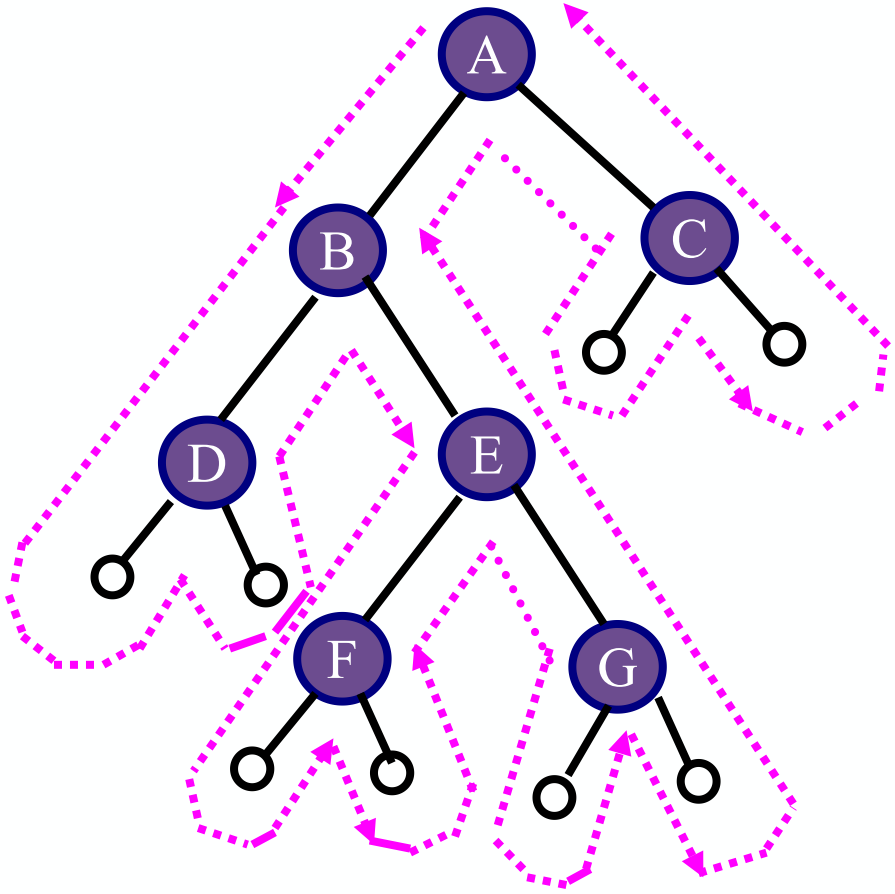
如果去掉输出语句，从递归的角度看，三种算法是完全相同的，或说这三种算法的访问路径是相同的，只是访问结点的时机不同。



从虚线的出发点到终点的路径上，每个结点经过3次。

第1次经过时访问 = 先序遍历
第2次经过时访问 = 中序遍历
第3次经过时访问 = 后序遍历

二、遍历算法的分析



时间效率:O(n)

//每个结点只访问一次

空间效率:O(n)

//栈占用的最大辅助空间

三、二叉树遍历算法的应用

1.按先序遍历序列建立二叉树的二叉链表

2.计算二叉树结点总数

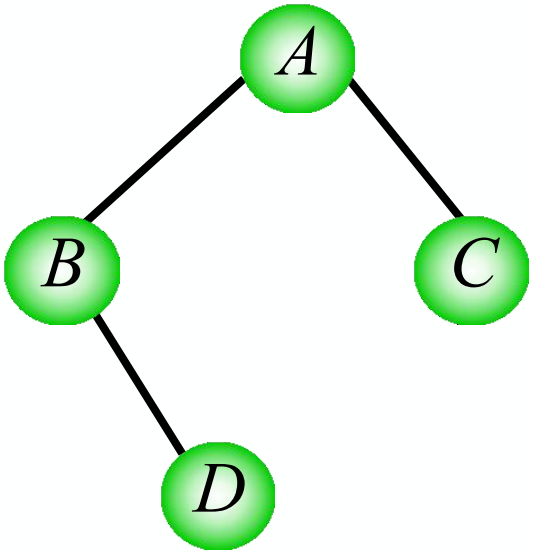
3.计算二叉树叶子结点总数

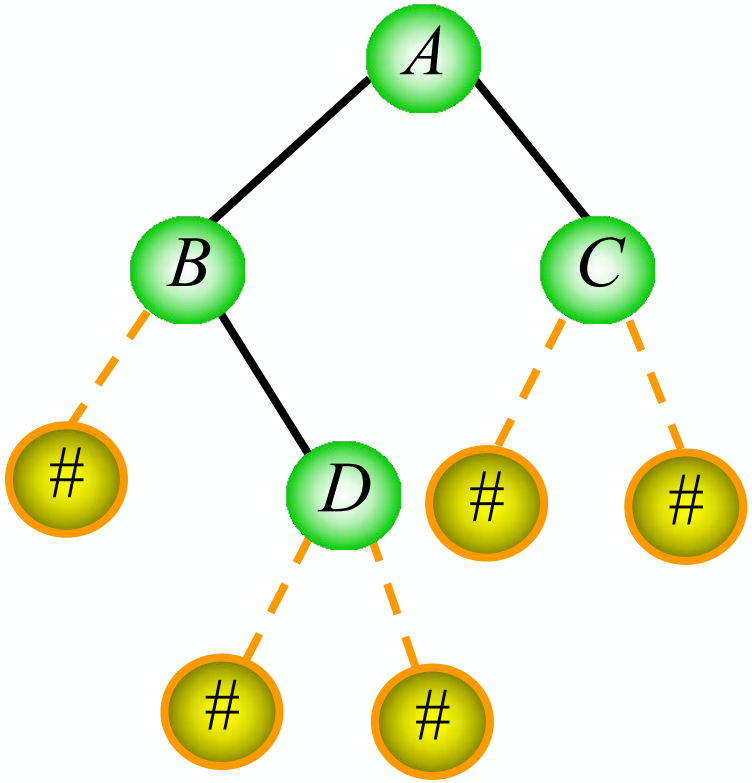
4.计算二叉树深度

三、二叉树遍历算法的应用

1. 构造函数——建立二叉树

为了建立一棵二叉树，将二叉树中每个结点的空指针引出一个虚结点，其值为一特定值如“#”，以标识其为空，把这样处理后的二叉树称为原二叉树的*扩展二叉树*。





扩展二叉树的前序遍历序列:*A B # D # # C # #*

三、二叉树遍历算法的应用

1. 构造函数——建立二叉树

设二叉树中的结点均为一个字符。假设扩展二叉树的前序遍历序列由键盘输入，root为指向根结点的指针，二叉链表的建立过程是：

首先输入根结点，若输入的是一个“#”字符，则表明该二叉树为空树，即root=NULL；否则输入的字符应该赋给root->data,，之后依次递归建立它的左子树和右子树 。

三、二叉树遍历算法的应用 ——二叉树的建立

```
void CreateBiTree(BiTree &T ) {
cin>>ch;
if (ch==’#’)   T=NULL;   //递归结束，建空树
else{
    T=new BiTNode;    T- > data=ch;    //生成根结点
    CreateBiTree(T- > lchild);  //递归创建左子树
    CreateBiTree(T- > rchild); //递归创建右子树
  }
}
```

三、二叉树遍历算法的应用

2.计算二叉树结点总数

- 如果是空树，则结点个数为0；
- 否则，结点个数为左子树的结点个数+右子树的结点个数再+1。

```
///计算二叉树结点总数
int NodeCount(BiTree T){
  if(T == NULL ) return 0;
  else return NodeCount(T->lchild)+NodeCount(T->rchild)+1;
}
```

三、二叉树遍历算法的应用

3.计算二叉树叶子结点总数

- 如果是空树，则叶子结点个数为0；
- 否则，为左子树的叶子结点个数+右子树的叶子结点个数

```
int LeadCount(BiTree T){
    if(T==NULL)          //如果是空树返回0
        return 0;
    if (T->lchild == NULL && T->rchild == NULL)
        return 1; //如果是叶子结点返回1
    else return LeafCount(T->lchild) + LeafCount(T->rchild) ;
}
```

三、二叉树遍历算法的应用

4.计算二叉树深度

- 如果是空树，则深度为0；
- 否则，递归计算左子树的深度记为m，递归计算右子树的深度记为n，二叉树的深度则为m与n的较大者加1。

```
int  Depth(BiNode *root)
{
    if (root == NULL) return 0;
    else {
        hl= Depth(root->lchild);
        hr= Depth(root ->rchild);
        return max(hl, hr)+1;
    }
}
```

小结

1. 遍历二叉树的算法步骤
2. 根据二叉树遍历序列确定二叉树的方法
3. 二叉树遍历算法的几个应用