数据结构与算法
Data Structures and Algorithms

第三部分 树

数据结构考查内容

一、线性表
（一）线性表的基本概念
（二）线性表的实现
（三）线性表的应用

二、栈、队列和数组
（一）栈和队列的基本概念
（二）栈和队列的顺序存储结构
（三）栈和队列的链式存储结构
（四）多维数组的存储
（五）特殊矩阵的压缩存储
（六）栈、队列和数组的应用

三、树与二叉树
（一）树的基本概念
（二）二叉树
（三）树、森林
（四）树与二叉树的应用

四、图
（一）图的基本概念
（二）图的存储及基本操作
（三）图的遍历
（四）图的基本应用

数据结构

五、查找
（一）查找的基本概念
（二）顺序查找法
（三）分块查找法
（四）折半查找法
（五）B树及其基本操作，B+树的基本概念
（六）散列（Hash）表
（七）字符串模式匹配
（八）查找算法分析及应用

六、排序
（一）排序的基本概念
（二）插入排序
（三）起泡排序
（四）简单选择排序
（五）希尔排序
（六）快速排序
（七）堆排序
（八）二路归并排序
（九）基数排序
（十）外部排序

算 法

教学要求

- 了解树型结构结点之间的层次关系；
- 掌握树和二叉树的定义及其相关的术语；
- 重点掌握二叉树的结构、性质，存储表示和四种遍历算法；
- 掌握二叉树线索化的实质及线索化的过程；
- 了解树的结构性质、存储表示方法和遍历算法；
- 掌握森林(树)与二叉树的对应关系和相互转换方法。

考纲内容

（一）树的基本概念

（二）二叉树

二叉树的定义及其主要特征；二叉树的顺序存储结构和链式存储结构；二叉树的遍历：线索二叉树的基本概念和构造

（三）树和森林

树的存储结构；森林和二叉树的转换；树和森林的遍历

（四）树和二叉树的应用

二叉排序树；平衡二叉树；哈夫曼树和哈夫曼编码

线性表：元素之间的线性关系
树：元素之间的层次关系

主要内容

3.1 基本术语

1、树的定义

【定义一】

（1）一个结点x组成的集{x}是一棵树(Tree)，这个结点x也是这棵树的根；

（2）假设x是一个结点，D_1，D_2，…，D_k是 k 棵互不相交的树，我们可以构造一棵新树：令x为根，并有k条边由x指向树D_1，D_2，…，D_k。这些边也叫做分支，D_1，D_2，…，D_k称作根x的树之子树（SubTree）。

【定义二】树是n(≥0)个结点的有限集。在任意一棵非空树中：

（1）有且仅有特定的称为根（Root）的结点；

（2）当n>1时，其余结点可分为k（>0）个互不相交的有限集D_1，D_2，…，D_k，其中每一个集合本身又是一棵树，并且称为根的子树（ SubTree ）。

【定义三】

数学模型

T = （D，R）

D：具有相同类型的数据元素的集合。

R：若 D 为空集，则称为空树；

若 D 仅含一个数据元素，则 R 为空集，否则 R = { H }，H 是如下的二元关系：

（1）在 D 中存在唯一的称为根的数据元素 root，它在关系 H 下无前驱；

（2）若 D - { root } ≠ $\not\subset$，则存在D-{ root }的一个划分D_1，D_2，…，D_k（k > 0），对任意 j ≠ l（1 ≤ j，l ≤ k）有$D_j \cap D_l = \not\subset$，对任意的 i (1 ≤ i ≤ k), 唯一存在数据元素 $x_i \in D_i$，有 < root , x_i > ∈ H;

（3）对应于 D - { root }的划分，H - { < root , x_1 > , … , <root,x_k>}有唯一的一个划分H_1，H_2，…，H_k (k>0), 对任意j≠l（1≤j，l≤k）有$H_j \cap H_l \neq \not\subset$, 且对任意的 i (1 ≤ i ≤ k), H_i是D_i上的二元关系，（D_i，{H_i}）是一棵符合本定义的树，称为根root的子树。

三个定义的共同点：

1、相同类型的元素构成的集合；

2、特定的结点---根；

3、除了根之外，组成 k 个划分，且互不相交；

4、每一个划分又是一棵树---递归；

几点说明：

① 递归定义，但不会产生循环定义；

② 构造性定义便于树型结构的建立；

③ 一株树的每个结点都是这株树的某株子树的根。

④ 树的根结点没有前驱结点，除根节点外每一个结点都有唯一前驱结点；

⑤ 树中所有结点可以有零个或多个后继结点。

线性结构

第一个数据元素
（无前驱）

最后一个数据元素
（无后继）

其它数据元素
（一个前驱、一个后继）

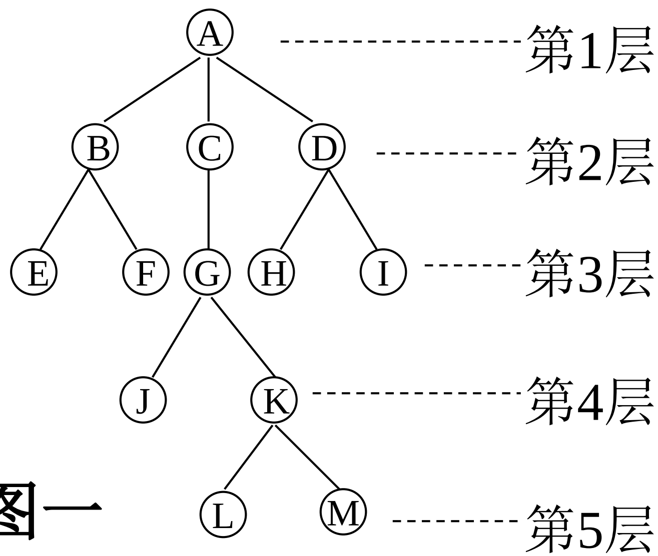
非线性结构—树

根结点
（无前驱）

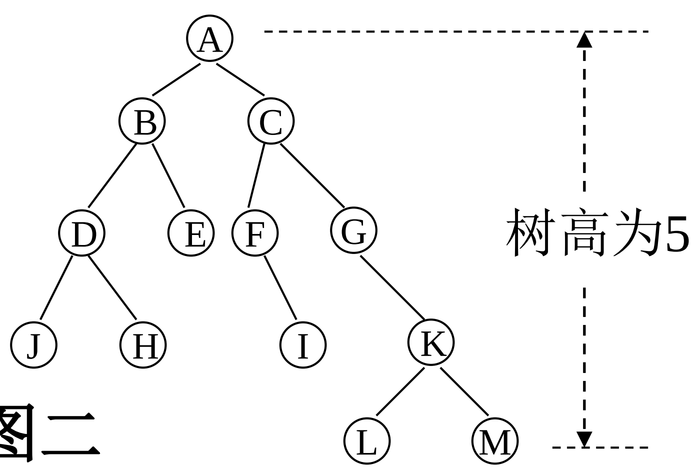
多个叶子结点
（无后继）

其它数据元素
（一个前驱、**多个**后继）

2、常用术语



图一



图二

结点
- 度
- 叶（终端结点）
- 非终端结点

分支　　路长

父亲　　双亲

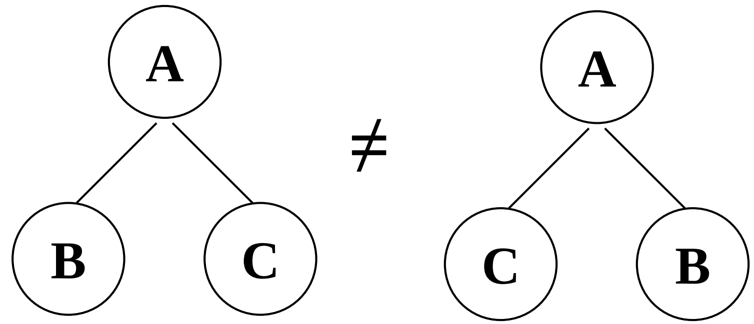
儿子　　兄弟

子孙　　祖先

层　　结点的高

树的高（深度）

有序树 & 无序树



森林forest：

是 n ≥ 0 棵互不相交的树的集合。

3.2 二叉树(BinaryTree)

3.2.1 二叉树的定义和基本性质

1、二叉树的定义

【定义一】 二叉树是有限个结点的集合，这个集合或者是空集，或者是由一个根结点和两棵互不相交的二叉树组成，其中一棵叫根的做左子树，另一棵叫做根的右子树。

【定义二】 BinaryTree = (D , R)

D：指数据对象，是由相同类型的数据元素组成的集合。

R：为数据元素间的关系：

若D＝ $\not\subset$, 则R= $\not\subset$ ， 称Binary tree 为空树；

若D $\neq$ $\not\subset$, 则R={H}，H是如下二元关系：

(1)在D中存在唯一的称为根的数据元素 root，它在关系H下无前驱；

(2)若D-{ root } $\neq \not\subset$ ， 则存在D-{root}={D_l，D_r}， 且$D_l \cap D_r$= $\not\subset$ ；

(3)若$D_l \neq \not\subset$ ， 则D_l中存在唯一的元素x_l，< root , x_l >∈H，且存在D_l上的关系H_l∈H；若$D_r \neq \not\subset$ ，则D_r中存在唯一的元素x_r，< root , x_r >∈H，且存在D_r上的关系H_r∈H；

H={< root , x_l>， <root,x_r>，H_l，H_r}；

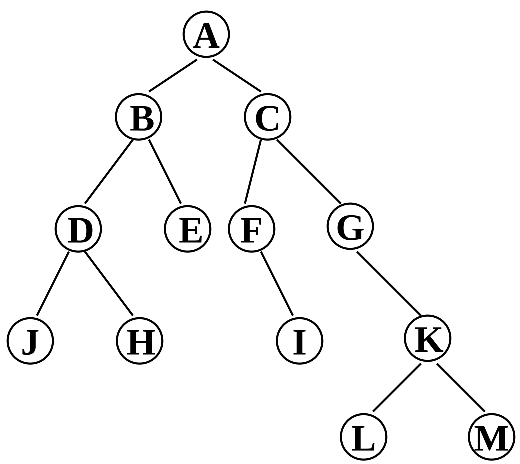
(4)（D_l，{H_l}）是符合本定义的二叉树，称为根的左子树；（D_r，{H_r}）是符合本定义的二叉树，称为根的右子树。

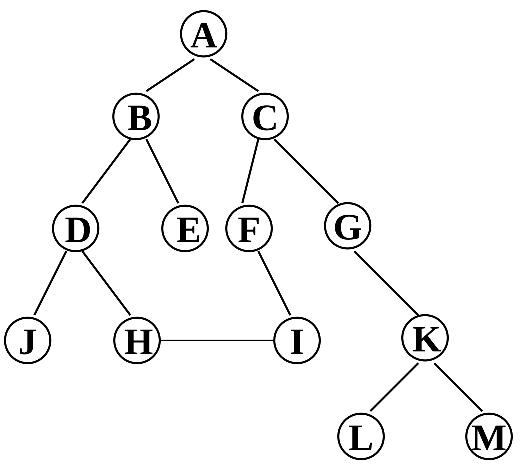
k <= 2
分左、右
二叉树是有序的

与树的定义对比：

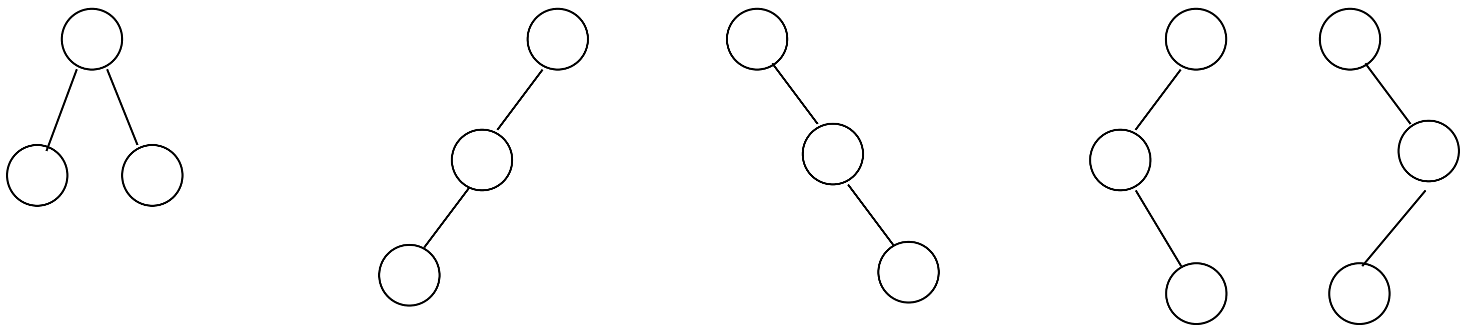
1、相同类型的元素构成的集合；
2、特定的结点---根；
3、除了根之外，组成k个划分，且互不相交；
4、每个结点至多有两棵子树；
5、每一个划分又是一棵二叉树---递归。

分析图（1）和图（2）的区别：





问题：具有三个结点的树和二叉树各有多少棵？



2、二叉树的性质

性质1： 在二叉树中第 i 层的结点数最多为2^{i-1}（$i \geq 1$）。

性质2： 高度为k的二叉树其结点总数最多为2^k-1（ $k \geq 1$）。

性质3： 对任意的非空二叉树 T，如果叶结点的个数为 n_0，而其度为 2 的结点数为 n_2，则：

$$n_0 = n_2 + 1。$$

【定义】 深度为k且有2^k-1个结点的二叉树称为满二叉树。

层序编号：对满二叉树的结点进行连续编号。从根结点开始，从上而下，自左至右。

【定义】 深度为 k 的，有n个结点的二叉树，当且仅当其每个结点都与深度为 k 的满二叉树中编号从 1 至 n 的结点一一对应，称之为完全二叉树。

二叉树的性质(续):

性质4 具有 n 个结点的完全二叉树的深度为 $\lfloor \log_2 n \rfloor + 1$。

证明：假设深度为 k，则根据性质 2 和完全二叉树的定义有

$$2^{k-1}-1 < n \leqslant 2^k - 1 \quad \text{或} \quad 2^{k-1} \leqslant n < 2^k$$

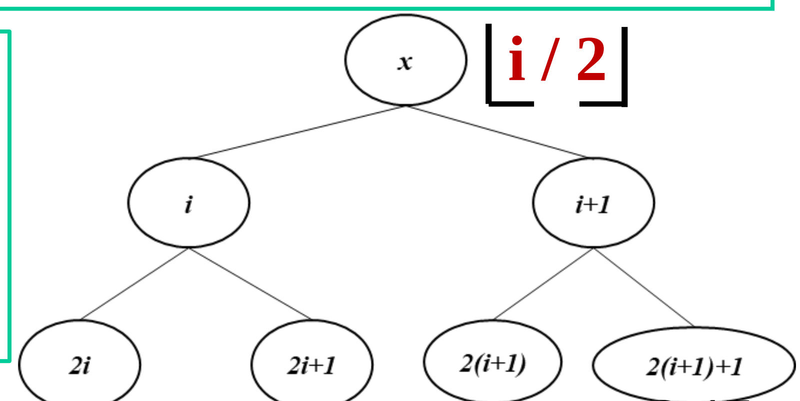
于是 $k-1 \leqslant \log_2 n < k$，因为 k 是整数，所以 $k = \lfloor \log_2 n \rfloor + 1$。

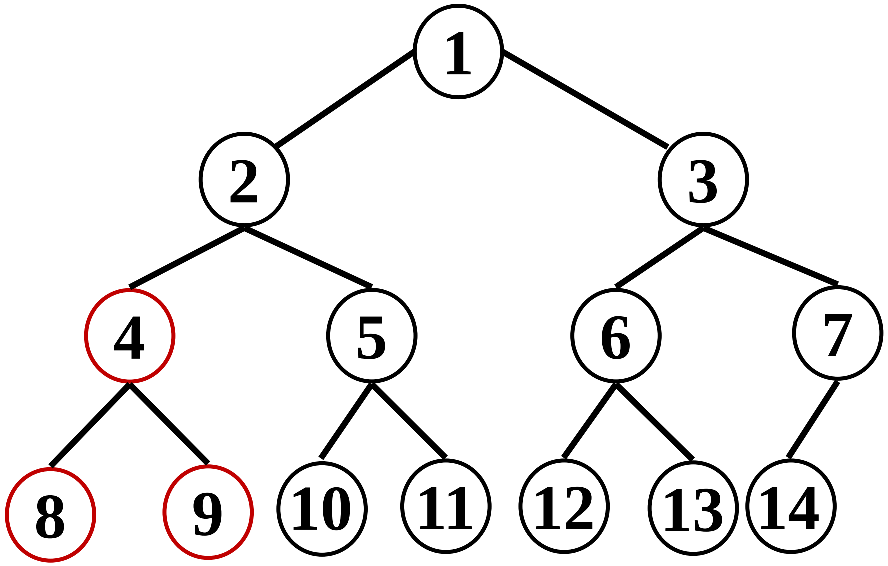
性质5 如果对一棵有 n 个结点的完全二叉树的结点按层序编号，则对任一结点 i 有：

(1) 如果 i = 1，则结点 i 是二叉树的根，无双亲；如果 i > 1，则其双亲结点是 $\lfloor i/2 \rfloor$；

(2) 如果 2 i > n，则结点 i 无左孩子结点，否则其左孩子结点是 2 i ；

(3) 如果 2 i + 1 > n，则结点 i 无右孩子结点，否则其右孩子结点是 2 i + 1。





编号：i = 4;
双亲为 $\lfloor i/2 \rfloor$ = 2;
左子树根为 2i = 8;
右子树根为2i+1=9;

i = 8, n = 14;
2i > n
无左子树

i = 7, n = 14;
2i+1 > n
无右子树

通过性质5把非线性结构转化成了线性结构

例题 1 如果二叉树的高度是h，并且只有度为0和2的结点，则此类二叉树中包含的结点至少得有（ ）个。

A. h B. 2h-1 C. 2h+1 D. h+1

思路：第一层（根）一个剩下h-1层，每层两个2(h-1)+1=2h-1.

例题 2 已知一棵完全二叉树的第6层(设根为第1层)有8个叶子结点，则该完全二叉树的结点个数最多是（ ）

A. 39 B. 52 C. 111 D. 119

思路：第6层有叶子，那么至多6或7层，所以第7层至少少了16个。

例题 3 若一棵完全二叉树有768个结点，则该二叉树中叶子结点的个数是（ ）

A. 257 B. 258 C. 384 D. 385

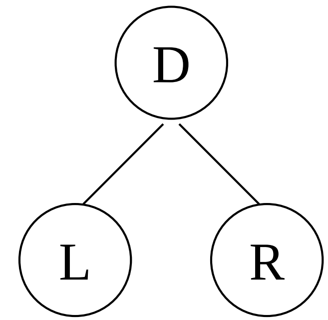
思路：最后一个结点的根是768/2向下取整 = 384，剩下的都是叶子结点 768-384=384。

例题 4 设一棵非空完全二叉树T的所有叶子结点均在同一层，且每个非叶子结点都有2子结点。如T有k个叶子结点，T结点总数为( )

A. 2k-1 B. 2k C. k^2 D. 2^k-1

思路：满二叉树，2^{h-1}=k; 2^h-1？

3、二叉树的遍历



【遍历】根据原则，按照一定的顺序访问二叉树中的每一个结点，使每个结点只能被访问一次。

根（D）、左孩子（L）和右孩子（R）三个结点可能出现的顺序有：

① DLR ② ~~DRL~~

③ LDR ④ LRD

⑤ ~~RLD~~ ⑥ ~~RDL~~

【原则】左孩子结点一定要在右孩子结点之前被访问。

要讨论的三种操作分别为：

①先根顺序DLR， ②中根顺序LDR， ③后根顺序LRD

①先根顺序遍历二叉树：

若二叉树非空则：
{ 访问根结点；
先根顺序遍历左子树；
先根顺序遍历右子树；
}；

②中根顺序遍历二叉树：

若二叉树非空则：
{ 中根顺序遍历左子树；
访问根结点；
中根顺序遍历右子树；
}；

③后根顺序遍历二叉树：

若二叉树非空则：
{ 后根顺序遍历左子树；
后根顺序遍历右子树；
访问根结点；
}；

3.2.2 抽象数据型二叉树

ADT操作：
① Empty (BT) ;
② IsEmpty (BT) ;
③ CreateBT (V, LT , RT) ;
④ Lchild (BT) ;
⑤ Rchild (BT) ;
⑥ Data (BT) ;

【例3-1】写一个递归函数，按先根顺序列出二叉树中每个结点的Data域之值。

```
Void  PreOrder ( BT )
BTREE  BT ;
{   if ( ! IsEmpty ( BT ) )
      {  visit ( Data ( BT ) ) ;
         PreOrder ( Lchild ( BT ) ) ;
         PreOrder ( Rchild ( BT ) ) ;
      }
}
```

抽象数据类型二叉树的定义如下：

ADT BinaryTree {

数据对象 D:D 是具有相同特性的数据元素的集合。

数据关系 R:

若 D = Φ,则 R = Φ,称 BinaryTree 为空二叉树；

若 D≠Φ,则 R = {H},H 是如下二元关系：

(1) 在 D 中存在惟一的称为根的数据元素 root,它在关系 H 下无前驱；

(2) 若 D - {root}≠Φ,则存在 D - {root} = $\{D_l, D_r\}$,且 $D_l \cap D_r = \Phi$;

(3) 若 $D_l \neq \Phi$,则 D_l 中存在惟一的元素 x_l,$<root, x_l> \in H$,且存在 D_l 上的关系 $H_l \subset H$；若 $D_r \neq \Phi$,则 D_r 中存在惟一的元素 x_r,$<root, x_r> \in H$,且存在 D_r 上的关系 $H_r \subset H$；$H = \{<root, x_l>, <root, x_r>, H_l, H_r\}$；

(4) $(D_l, \{H_l\})$是一棵符合本定义的二叉树,称为根的左子树,$(D_r, \{H_r\})$是一棵符合本定义的二叉树,称为根的右子树。

基本操作 P:

InitBiTree(&T);

操作结果:构造空二叉树 T。

DestroyBiTree(&T);

初始条件:二叉树 T 存在。

操作结果:销毁二叉树 T。

CreateBiTree(&T, definition);

初始条件：definition 给出二叉树 T 的定义。

操作结果：按 definition 构造二叉树 T。

ClearBiTree(&T)；

初始条件：二叉树 T 存在。

操作结果：将二叉树 T 清为空树。

BiTreeEmpty(T)；

初始条件：二叉树 T 存在。

操作结果：若 T 为空二叉树，则返回 TRUE，否则 FALSE。

BiTreeDepth(T)；

初始条件：二叉树 T 存在。

操作结果：返回 T 的深度。

Root(T)；

初始条件：二叉树 T 存在。

操作结果：返回 T 的根。

Value(T, e)；

初始条件：二叉树 T 存在，e 是 T 中某个结点。

操作结果：返回 e 的值。

Assign(T, &e, value)；

初始条件：二叉树 T 存在，e 是 T 中某个结点。

操作结果：结点 e 赋值为 value。

Parent(T, e);

初始条件:二叉树 T 存在,e 是 T 中某个结点。

操作结果:若 e 是 T 的非根结点,则返回它的双亲,否则返回"空"。

LeftChild(T, e);

初始条件:二叉树 T 存在,e 是 T 中某个结点。

操作结果:返回 e 的左孩子。若 e 无左孩子,则返回"空"。

RightChild(T, e);

初始条件:二叉树 T 存在,e 是 T 中某个结点。

操作结果:返回 e 的右孩子。若 e 无右孩子,则返回"空"。

LeftSibling(T, e);

初始条件:二叉树 T 存在,e 是 T 中某个结点。

操作结果:返回 e 的左兄弟。若 e 是 T 的左孩子或无左兄弟,则返回"空"。

RightSibling(T, e);

初始条件:二叉树 T 存在,e 是 T 中某个结点。

操作结果:返回 e 的右兄弟。若 e 是 T 的右孩子或无右兄弟,则返回"空"。

InsertChild(T, p, LR, c);

初始条件:二叉树 T 存在,p 指向 T 中某个结点,LR 为 0 或 1,非空二叉树 c 与 T 不相交且右子树为空。

操作结果:根据 LR 为 0 或 1,插入 c 为 T 中 p 所指结点的左或右子树。p 所指结点的原有左或右子树则成为 c 的右子树。

DeleteChild(T, p, LR);

初始条件:二叉树 T 存在,p 指向 T 中某个结点,LR 为 0 或 1。

操作结果：根据 LR 为 0 或 1，删除 T 中 p 所指结点的左或右子树。

PreOrderTraverse(T, Visit());

初始条件：二叉树 T 存在，Visit 是对结点操作的应用函数。

操作结果：先序遍历 T，对每个结点调用函数 Visit 一次且仅一次。一旦 visit()失败，则操作失败。

InOrderTraverse(T, Visit());

初始条件：二叉树 T 存在，Visit 是对结点操作的应用函数。

操作结果：中序遍历 T，对每个结点调用函数 Visit 一次且仅一次。一旦 visit()失败，则操作失败。

PostOrderTraverse(T, Visit());

初始条件：二叉树 T 存在，Visit 是对结点操作的应用函数。

操作结果：后序遍历 T，对每个结点调用函数 Visit 一次且仅一次。一旦 visit()失败，则操作失败。

LevelOrderTraverse(T, Visit()); ?

初始条件：二叉树 T 存在，Visit 是对结点操作的应用函数。

操作结果：层序遍历 T，对每个结点调用函数 Visit 一次且仅一次。一旦 visit()失败，则操作失败。

}ADT BinaryTree

【例3-2】写一个递归函数，按中根顺序列出二叉树中每个结点的Data域之值。

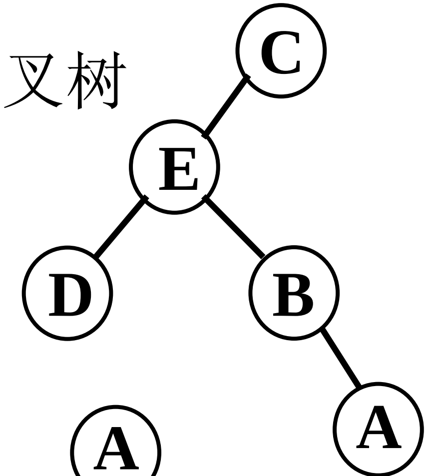
```
Void  InOrder ( BT )
BTREE  BT ;
{   if ( ! IsEmpty ( BT ) )
    {  InOrder ( Lchild ( BT ) ) ;
       visit ( Data ( BT ) ) ;
       InOrder ( Rchild ( BT ) ) ;
    }
}
```

```
Void  PostOrder ( BT )
BTREE  BT ;
{   if ( ! IsEmpty ( BT ) )
    {  PostOrder ( Lchild ( BT ) ) ;
       PostOrder ( Rchild ( BT ) ) ;
       visit ( Data ( BT ) ) ;
    }
}
```

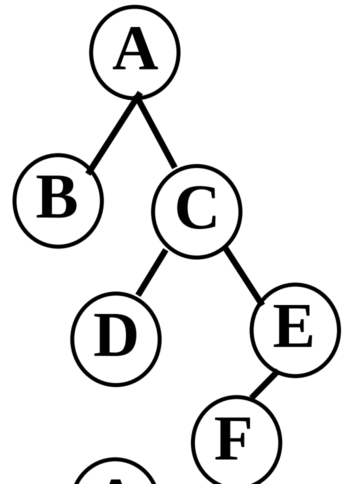
【例3-3】写一个递归函数，按后根顺序列出二叉树中每个结点的Data域之值。

例题 请画出下列给定遍历方式的二叉树

中序遍历序列: DEBAC
后序遍历序列: DABEC



先序遍历序列: ABCDEF
中序遍历序列: BADCFE



先序遍历序列: ABCDEF
中序遍历序列: CBAEDF