第7章 树和二叉树

7.1 树的概念和性质
7.2 二叉树的概念与性质
7.3 二叉树的存储结构
7.4 二叉树的遍历
7.5 二叉树的其他操作算法
7.6 线索二叉树
7.7 树的存储结构与算法
7.8 Huffman树与Huffman编码
7.9 等价类问题

树和森林的概念

📖 树的定义：

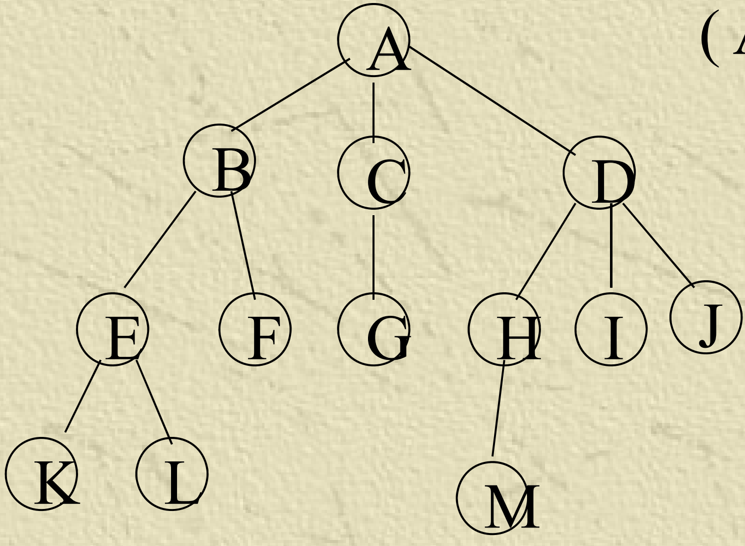
是n（n≥0)个结点的有限集合T，对于任意一棵非空树，它满足：

（1）有且仅有一个特定的称为根的结点；

（2）当n＞1时，其余结点可分为m(m＞0)个互不相交的有限集T_1，T_2，….，T_m，其中每个集合本身又是一棵树，称为根的子树。

显然：上述<u>树的定义是一个递归定义。</u>

树的表示方法:

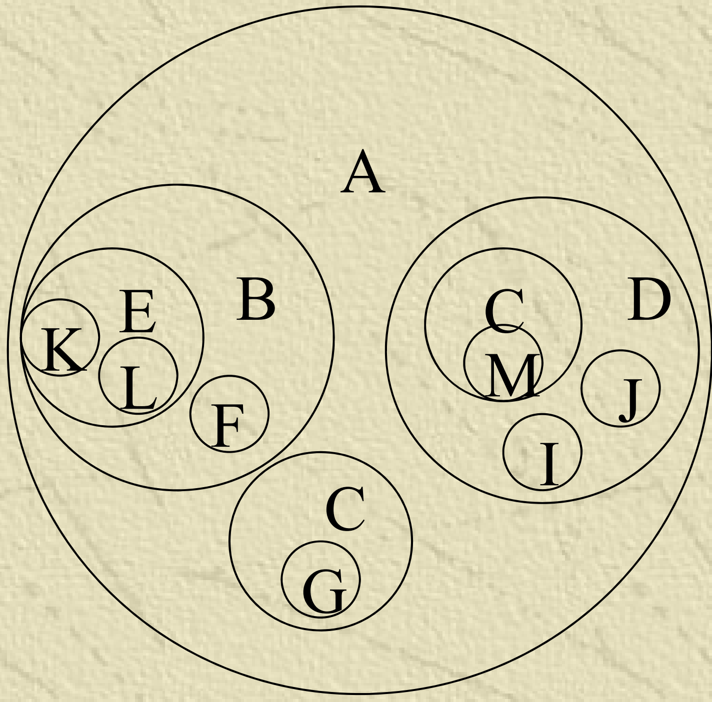


树形表示

(A(B(E(K,L), F), C(G), D(H(M), I, J))

广义表

凹入表



文氏图

树的基本术语：

• 结点(node)	
• 结点的度(degree)	结点的子树个数
• 分支(branch)结点	度不为0的结点
• 叶(leaf)结点	度为0的结点
• 孩子(child)结点	某结点子树的根结点
• 双亲(parent)结点	某个结点是其子树之根的双亲
• 兄弟(sibling)结点	具有同一双亲的所有结点
• 祖先(ancestor)结点	若树中结点k到k_s存在一条路径，则称k是k_s的祖先
• 子孙(descendant)结点	若树中结点k到k_s存在一条路径，则称k_s是k的子孙
• 结点所处层次(level)	根结点的层数为1，其余结点的层数为双亲结点的层数加1
• 树的高度(depth)	树中结点的最大层数
• 树的度(degree)	树中结点度数的最大值

- 有序树　　　　子树的次序不能互换
- 无序树　　　　子树的次序可以互换
- 森林　　　　　互不相交的树的集合

树的基本操作

1、初始化 (InitTree)

2、建立树 (CreateTree)

3、求指定结点的双亲结点 (Parent)

4、求指定结点的左孩子结点 (LeftChild)

5、求指定结点的右兄弟结点 (RightSibling)

6、将一棵树插入到另一树的指定结点下作为它的子树 (InsertChild)

7、删除指定结点的某一子树 (DeleteChild)

8、树的遍历 (TraverseTree)

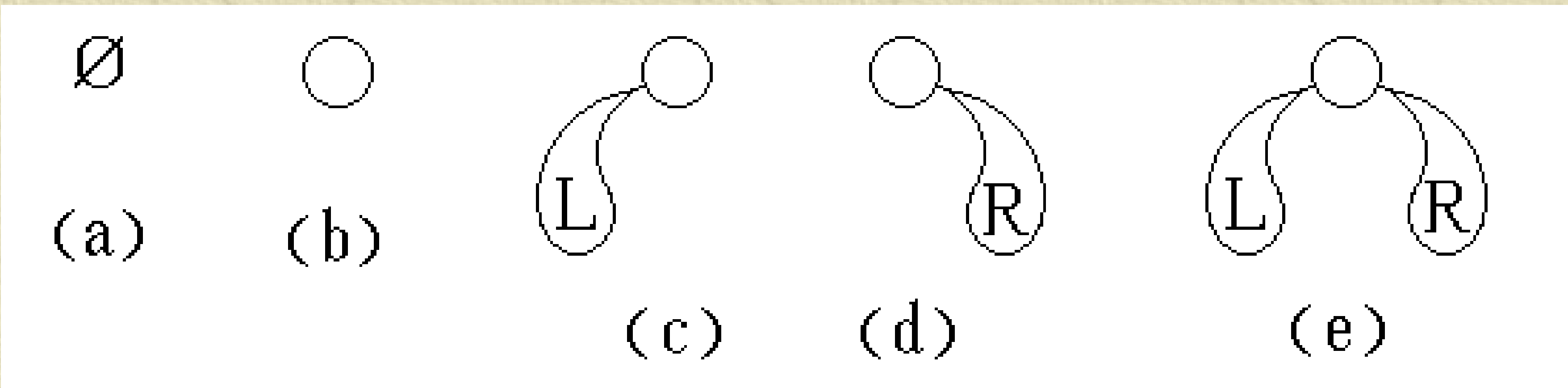
7.2 二叉树的概念与性质

二叉树的定义

一棵二叉树是**n** (**n** ≥0)个结点的一个有限集合，

(1) 该集合或者为空，

(2) 或者是由一个根结点加上两棵分别称为左子树和右子树的、互不相交的二叉树组成。



二叉树的五种不同形态

问题：

试分别画出具有3个结点的树和3个结点的二叉树的所有不同形态。

二叉树的性质

性质1 若二叉树的层次从1开始, 则在二叉树的第 i 层最多有 2^{i-1} 个结点。($i \geq 1$)

证明:

$i = 1$ 时，有$2^{i-1} = 2^0 = 1$，*成立*

假定: $i = k$ 时性质成立;

当 $i = k+1$ 时，第k+1层的结点至多是第k层结点的两倍，即总的结点个数至多为$2 \times 2^{k-1} = 2^k$

故命题成立

性质2 高度为k的二叉树最多有 2^k-1个结点。($k \geq 1$)

证明：仅当每一层都含有最大结点数时，二叉树的结点数最多，利用性质1可得二叉树的结点数至多为：

$$2^0 + 2^1 + 2^2 + 2^3 + \ldots + 2^{k-1} = 2^k-1$$

性质3 对任何一棵二叉树，如果其叶结点个数为n_0，度为2的非叶结点个数为 n_2，则有

$$n_0=n_2+1$$

证明：

1、结点总数为度为0的结点加上度为1的结点再加上度 为2的结点：

$$n = n_0 + n_1 + n_2$$

2、另一方面，二叉树中一度结点有一个孩子，二度结点有二个孩子，根结点不是任何结点的孩子，因此，结点总数为：

$$n = n_1 + 2n_2 + 1$$

3、两式相减，得到：

$$n_0 = n_2 + 1$$

定义1 满二叉树(Full Binary Tree)

一棵深度为k 且有2^k-1个结点的二叉树。

满二叉树的特点：每一层都取最大结点数

定义2 完全二叉树(Complete Binary Tree)

高度为k，有n个结点的二叉树是一棵完全二叉树，当且仅当其每个结点都与高度为k的满二叉树中层次编号1--n相对应。

🕮完全二叉树的特点---

（1）除最后一层外，每一层都取最大结点数，最后一层结点都有集中在该层最左边的若干位置。

（2）叶子结点只可能在层次最大的两层出现。

（3）对任一结点，若其右分支下的子孙的最大层次为L，则其左分支下的子孙的最大层次为L或L+1。

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

证明：

设深度为k，根据二叉树性质二知：

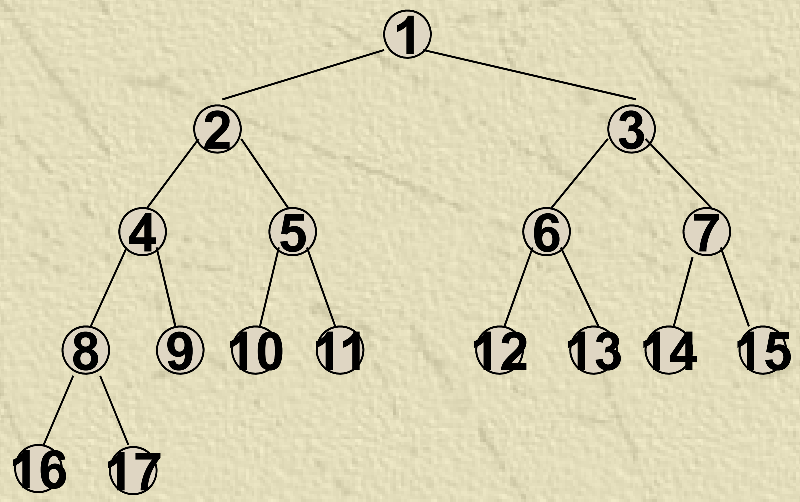
$2^{k-1}-1 < n \le 2^k-1$ ，即：

$2^{k-1} \le n < 2^k$，于是有：

$$k-1 \le \log_2 n \langle k$$

$$\because k\text{为整数},\therefore \text{取}k = \lfloor \log_2 n \rfloor + 1$$

性质5 如果将一棵有*n*个结点的完全二叉树自顶向下，同一层自左向右连续给结点编号1, 2, …, *n*-1,*n*，然后按此结点编号将树中各结点顺序地存放于一个一维数组中, 并简称编号为*i*的结点为结点*i* ($1 \le i \le n$)。则有以下关系：



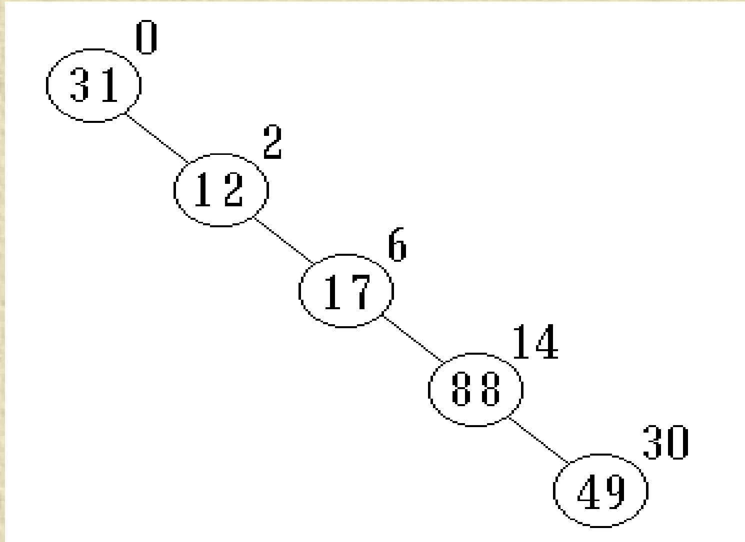
- 若*i* == 1, 则 *i* 无双亲
  若*i* > 1, 则 *i* 的双亲为$\lfloor i/2 \rfloor$
- 若2*i <= *n*, 则 *i* 的左孩子为2*i；否则，*i*无左孩子，必定是叶结点，二叉树中*i*>$\lfloor n/2 \rfloor$的结点必定是叶结点
  若2*i+1 <= *n*, 则 *i* 的右孩子为2*i+1，否则，i 无右孩子
- 若 *i* 为奇数, 且*i*不为1，则其左兄弟为*i*-1，否则无左兄弟；
  若 *i* 为偶数, 且小于 *n*，则其右兄弟为*i*+1，否则无右兄弟
- *i* 所在层次为$\lfloor \log_2 i \rfloor + 1$

7.3 二叉树的存储

一. 顺序存储结构

- 指用一组连续的存储单元存储二叉树的结点数据。
- 要求：必须把二叉树中的所有结点，按照一定的次序排成为一个线性序列，结点在这个序列中的相互位置能反映出结点之间的逻辑关系。
- 在结点的线性序列中，如何反映结点之间的逻辑关系（分支关系）？
  - 对于完全二叉树和满二叉树，结点的层次序列足以反映整个二叉树的结构。
  - 对于一般二叉树，则需要通过添加虚结点将其扩充为完全二叉树。

- 由于一般二叉树必须仿照完全二叉树那样存储，可能会浪费很多存储空间，单支树就是一个极端情况。



单支树

- 若要在树中经常插入和删除结点时，由于要大量移动结点，显然在这种情况下采用顺序方式并不可取。

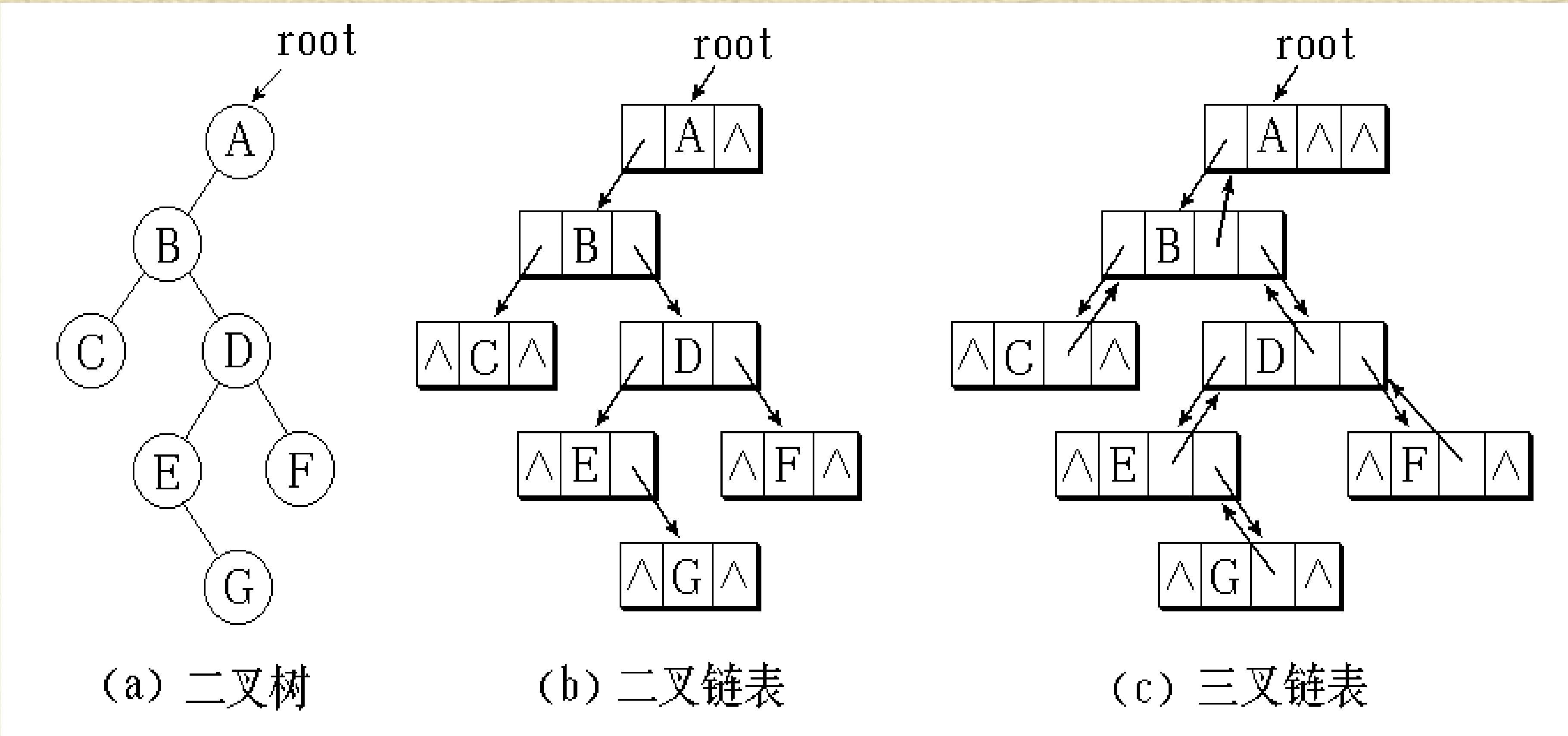
二. 链式存储结构

- 由于二叉树的每个结点最多有左、右两个孩子，因此在采用链式存储表示时，每个结点至少需要包含三个域：数据域和左、右指针域。

lchild	**Data**	**rchild**

- 一个二叉树中所有这种形式的结点，再加上一个指向根结点的头指针，就构成了此二叉树的链式存储结构，称之为二叉链表。

- 如果想能够找到父结点，则可以增加一个指向父结点的指针域，则构成三叉链表。



二叉树链表表示的示例

二叉链表结构定义：

```
template <class T>
struct BiNode
{   T data;        // 结点数据
    BiNode<T> *lchild;     // 左孩子的指针
    BiNode<T> *rchild;    // 右孩子的指针
};
```

二叉树的类定义

```
template <class T>
class BiTree{
   BiNode<T>* root;   // 根指针
 public:
   BiTree() { root=NULL; }
   BiTree(vector<T> &pre);
   BiTree<T>::BiTree(const BiTree<T> &tree);
   ~BiTree();
   void PreOrder();
   void InOrder();
   void PostOrder();
   void LevelOrder();
   int Height();
   BiNode<T> *Search(T e);
   BiNode<T> *SearchParent(BiNode<T>*child);
};
```

6.3 二叉树的遍历

- 遍历二叉树

  按某条搜索路径访问树中每一个结点，使得每个结点均被访问一次，且仅被访问一次。

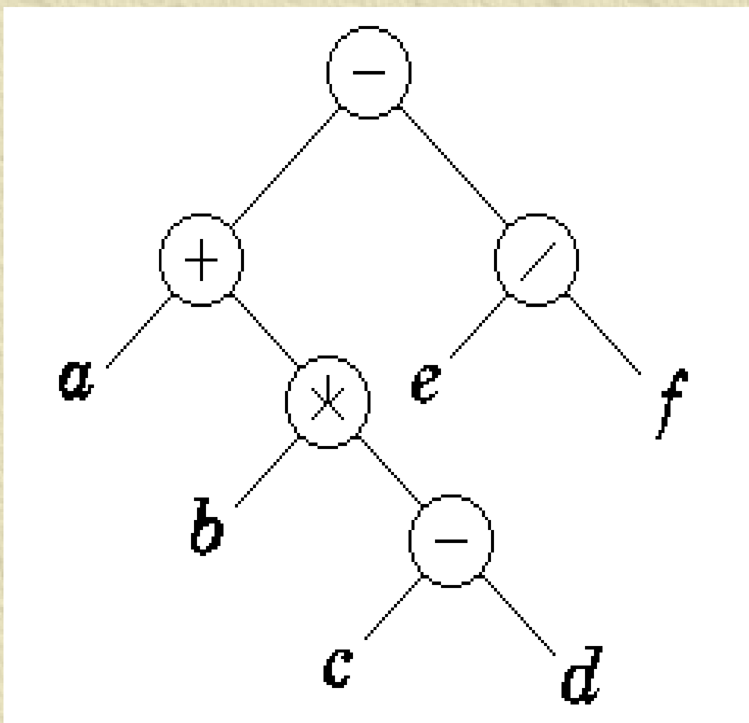
- 六种访问次序（**N**--访问根，**L**--遍历左子树，**R**--遍历右子树）：
  - **NLR**、**LNR**、**LRN**、**NRL**、**RNL**、**RLN**

- 若限定按先左后右的次序遍历，则有如下三种遍历次序：
  - 先序遍历：**NLR**
  - 中序遍历：**LNR**
  - 后序遍历：**LRN**

先序遍历

先序遍历二叉树算法的框架是

- 若二叉树为空，则空操作；
- 否则
  - – 访问根结点 **(V)**；
  - – 先序遍历左子树 **(L)**；
  - – 先序遍历右子树 **(R)**。



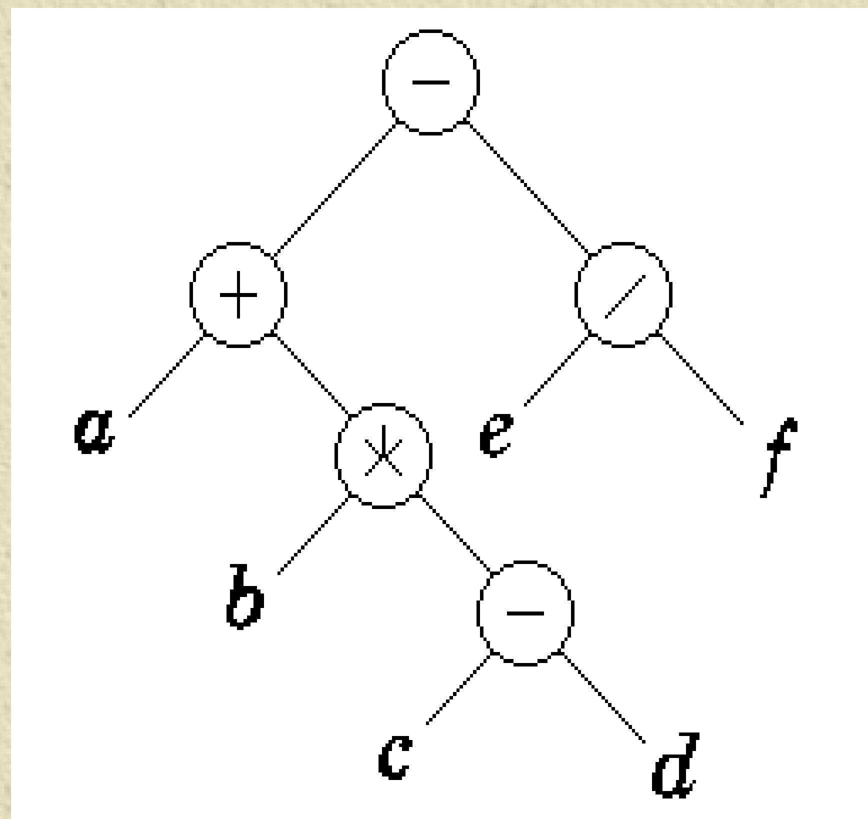
遍历结果:

− + a * b − c d / e f

中序遍历

中序遍历二叉树算法的框架是：

- 若二叉树为空，则空操作；
- 否则
  - 中序遍历左子树 (L)；
  - 访问根结点 (V)；
  - 中序遍历右子树 (R)。



遍历结果 **a + b * c - d - e / f**

表达式语法树

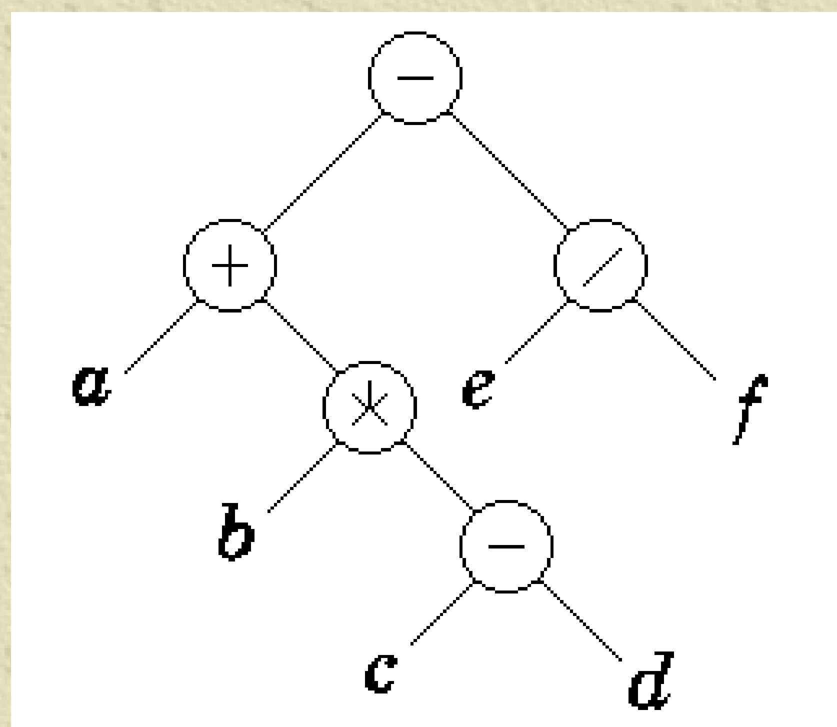
后序遍历

后序遍历二叉树算法的框架是

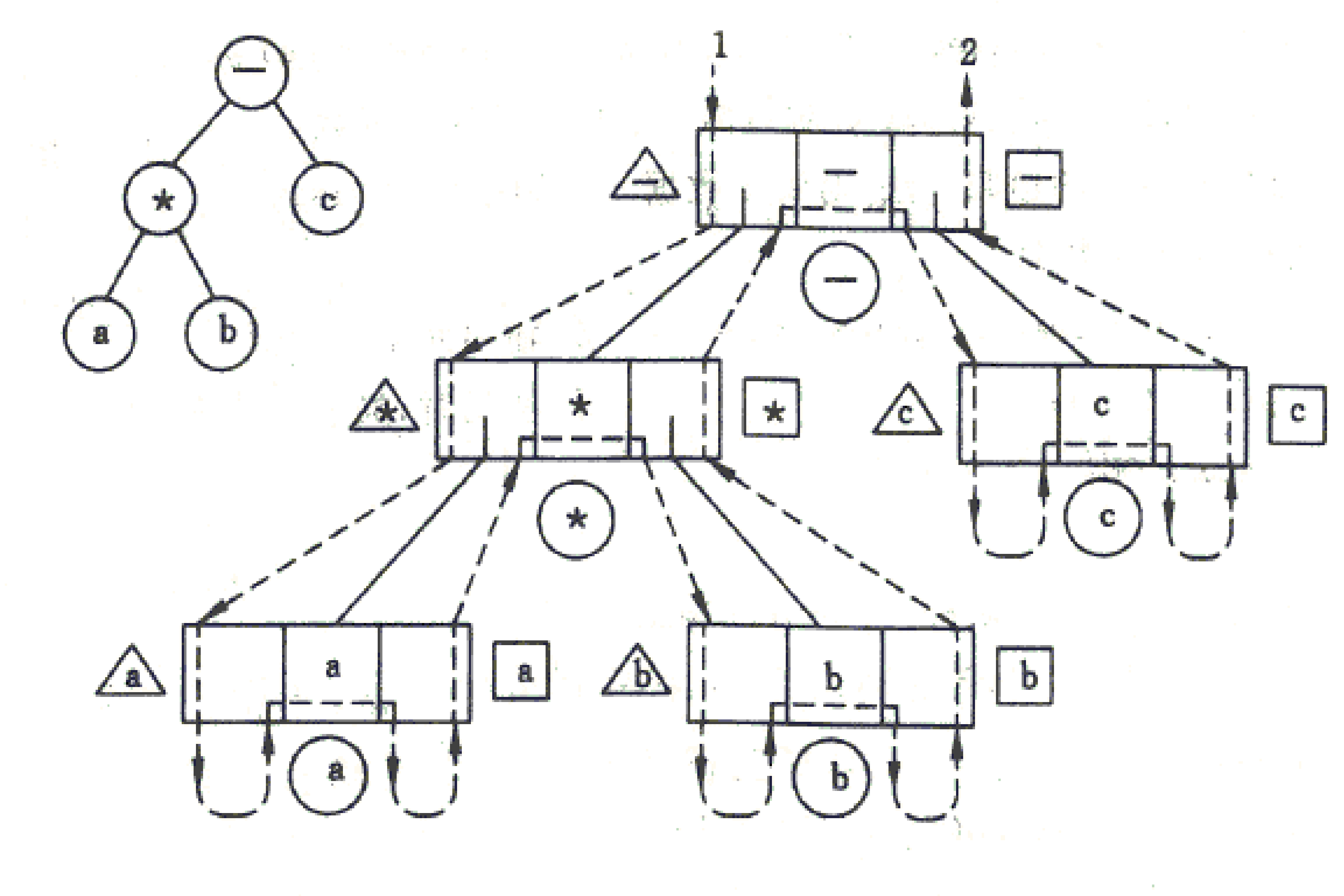
- 若二叉树为空，则空操作；
- 否则
  - 后序遍历左子树 (L)；
  - 后序遍历右子树 (R)；
  - 访问根结点 (V)。

遍历结果:

a b c d − * + e f / −



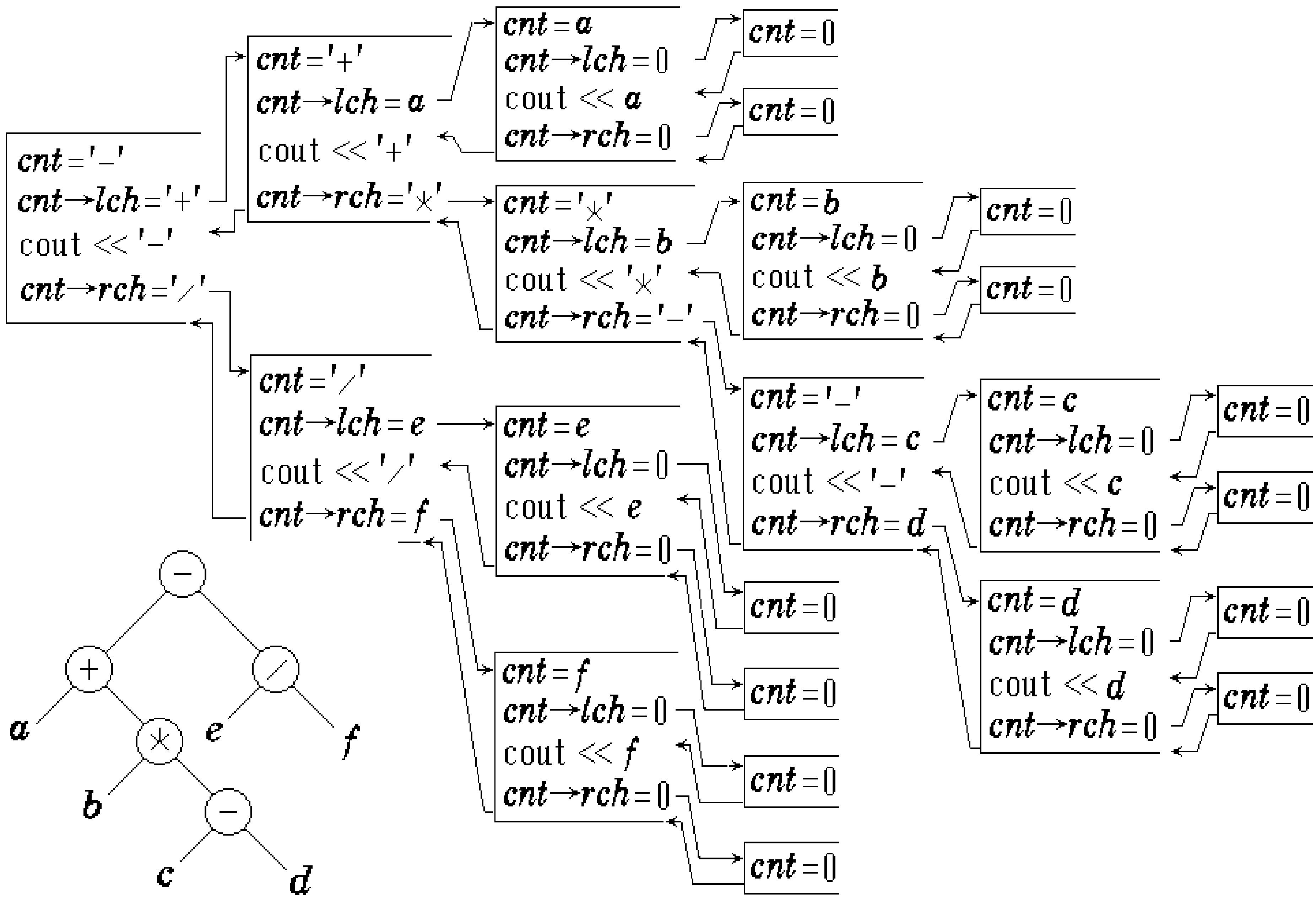
先、中、后序遍历的流程



中序遍历的递归算法：

```
template <class T>
void BiTree<T>::InOrder(BiNode<T> *p)
{   if(p==NULL) return;
    InOrder(p->lchild);
    cout << p->data;
    InOrder(p->rchild);
}
template <class T>
void BiTree<T>::InOrder()
{  InOrder(root);  }
```

算法的
时间复杂度？



中序遍历二叉树的递归过程图解

思考题

如何实现二叉树中序遍历的非递归算法？

中序遍历的非递归算法

```
template <class T>
void BiTree<T>::InOrder(BiNode<T> *t)
{   SeqStack S;    S.Push(t);
    while( !S.Empty() ){
        p=S.Top();
        while( p ){
            S.Push(p->lchild); p=p->lchild;    //向左走到尽头
        }
        p=S.Pop();    //空指针退栈
        if( !S.Empty() ){
            p=S.Pop();   cout<<p->data;   //访问结点
            S.Push(p->rchild)     //进入右子树
        }
    }
}
```

先序遍历的递归算法：

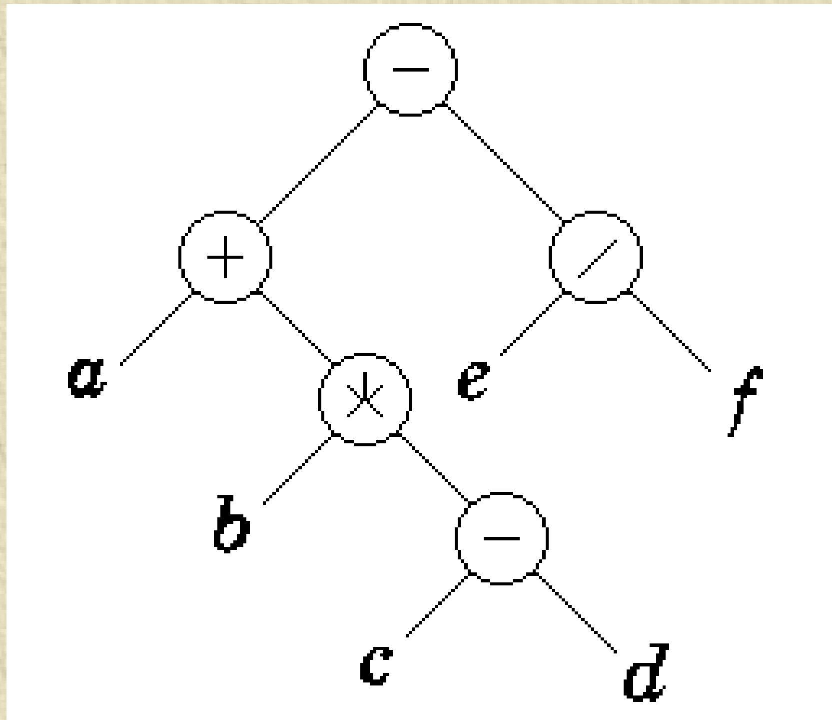
```
template <class T>
void BiTree<T>::PreOrder(BiNode<T> *p)
{   if(p==NULL) return;
    cout << p->data;
    PreOrder(p->lchild);
    PreOrder (p->rchild);
}

template <class T>
void BiTree<T>::PreOrder()
{  PreOrder(root)  }
```

层序遍历

层序遍历二叉树算法的框架是

- 若二叉树为空，则空操作；
- 将根结点入队
- 如队列不空，循环：
  - 做出队操作，队头元素作为当前结点；
  - 将当前结点的左右孩子入队
- 最后，出队序列就是层序遍历序列。



遍历结果:

− + / a * e f b − c d

二叉树的层序遍历算法

```
template <class T>
void BiTree<T>::LevelOrder() {
    Queue<BiNode<T> *> Q;   // Q为指针队列
    if( !root ) return;
    Q.EnQueue(root);
    while( !Q.Empty() )
     {   BiNode<T> *p= Q.Dequeue();
          cout<<p->data;
          if(p->lchild)  Q.EnQueue(p->lchild);
          if(p->rchild)  Q. EnQueue(p->rchild);
    }
}
```

7.4.3 二叉树的构造和析构算法

- 二叉树的建立
  - 目标：
    - 给定一棵二叉树的结点的值的序列，建立该二叉树对应的二叉链表。
- 若给定一棵二叉树的结点的先序序列，能建立一棵对应的二叉树吗？
  - 在序列中增加空指针标记

1、由单个遍历序列构造二叉树

```
template <class T>
BiNode<T> *BiTree<T>::CreateByPre()
{   e=getchar();
     if(e=='*')  return NULL;
    p=new BiNode<T>;
    p->data=e;
    p->lchild=CreateByPre();
    p->rchild=CreateByPre();
    return p;
}
```

1、由单个遍历序列构造二叉树

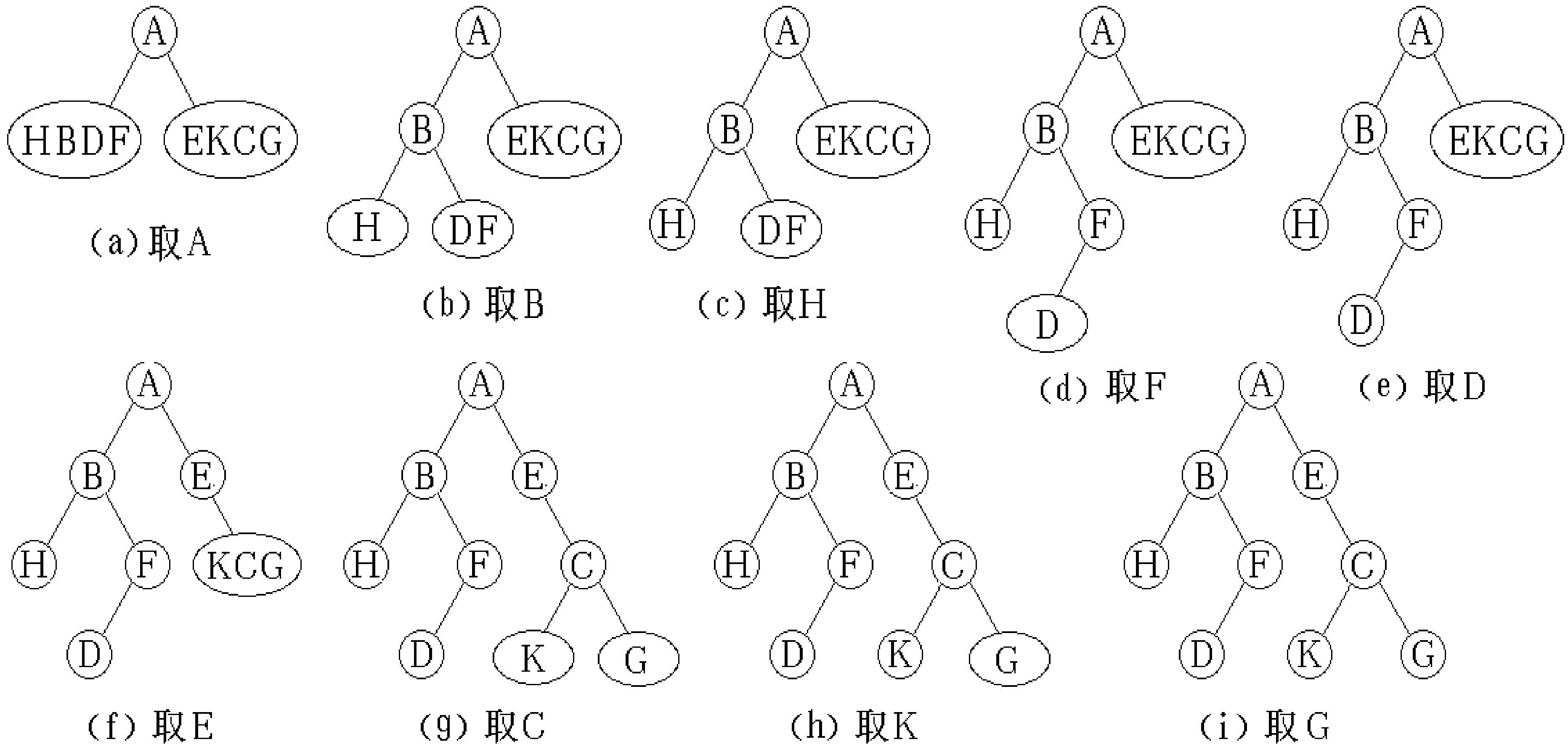
```
template <class T>
BiNode<T> *BiTree<T>::CreateByPre(vector<T> &pre,int &i)
{   e=pre[i];   i++;                // 提取当前数据
    if(e=='*')  return NULL;
   p=new BiNode<T>;  p->data=e;
   p->lchild=CreateByPre(pre, i);
   p->rchild=CreateByPre(pre, i);
   return p;
}
 template <class T>
 BiTree<T>::BiTree(vector<T> &pre) {
     i=0;
     root=CreateByPre(pre, i);
 }
```

思考题

问：由添加空指针标记的单个中序或后序遍历序列是否可构造相应的二叉树？

2、由二个遍历序列构造二叉树

已知： 先序序列 **{ ABHFDECKG }**,
中序序列 **{ HBDFAEKCG }**,
试构造相应的二叉树。



性质：一棵二叉树的先序序列和中序序列可以唯一的确定这棵二叉树.

用归纳法证明：

1、当 $n = 1$时，结论显然成立；

2、假定当 $n <= k$ 时，结论成立；

3、当 $n = k + 1$ 时，假定先序序列和中序序列分别为：

$$\{a_1, \ldots, a_{k+1}\} \text{ 和 } \{b_1, \ldots, b_{k+1}\}$$

如中序序列中与先序序列中的a_1相同的元素:为b_j :

- ✓ j = 1时，二叉树无左子树，由 $\{a_2, \ldots, a_{k+1}\}$ 和 $\{b_2, \ldots, b_{k+1}\}$ 可以唯一的确定二叉树的右子树；
- ✓ j = k+1时，二叉树无右子树，由 $\{a_2, \ldots, a_{k+1}\}$ 和 $\{b_1, \ldots, b_k\}$ 可以唯一的确定二叉树的左子树；
- ✓ 如2<= j <=k，则：
  - 子序列 $\{a_2, \ldots, a_j\}$ 和$\{b_1, \ldots, b_{j-1}\}$唯一地确定二叉树的左子树；
  - 子序列$\{a_{j+1}, \ldots, a_{k+1}\}$ 和 $\{b_{j+1}, \ldots, b_{k+1}\}$唯一地确定二叉树的右子树.

参数如何设置？

```
template <class T>
BiNode<T>* BiTree<T>::CreateByPreMid( vector<T> &pre,
                         vector<T> &mid, int ipre, int imid, int n )
{
   if(n==0) return NULL;
   p = new BiNode<T>;
   p->data = pre[ipre];
   for(i=0; i<n; i++)
       if( pre[ipre]==mid[imid+i] )  break;
   p->lchild = CreateByPreMid(pre, mid, ipre+1, imid, i);
   p->rchild = CreateByPreMid(pre, mid,
                              ipre+i+1, imid+i+1, n-i-1);
   return p;
}
```

3、拷贝构造函数

```
template <class T>
BiNode<T> * BiTree<T>::Copy(BiNode<T> *p)
{
    if( p==NULL )  return NULL;
    newp=new BiNode<T>;
    newp->data=p->data;
    newp->lchild= Copy(p->lchild);
    newp->rchild= Copy(p->rchild);
    return newp;
}
template <class T>
BiTree<T>::BiTree(const BiTree<T> &tree)
{  root=Copy(tree.root); }
```

4、析构函数

```
template <class T>
void BiTree<T>::Free(BiNode<T> *p)
{
    if(p==NULL)  return;
    Free( p->lchild );
    Free( p->rchild );
    delete p;        // 释放根结点
}
template <class T>     // 析构函数
BiTree<T>::~BiTree()
{ if(root)  Free(root); }
```

例：设计一个算法，将完全二叉树的顺序存储结构转换为二叉链表结构。

```
BiNode<T>*  turn( char A[], int n, int i ) {
    if( n<1 || i>n) return NULL;
    p=new BiNode<T>;
    p->data = A[i];
    if( 2*i <= n ) p->lchild=turn(A, n , 2*i);
    else  p->lchild=NULL;
    if( 2*i+1 <= n ) p->rchild=turn( A, n , 2*i+1);
    else  p->rchild=NULL;
    return p;
}
```

方法二 (非递归):

```
BiNode<T>* turn( char A[], int n )
{
    BiNode<T>* p=new BiNode<T>* [n+1];
    for( i=1; i<=n; i++ ){
        p[i] = new BiNode<T>;
        p[i]->data = A[i];
    }
    for( i=1; i<=n; i++ ){
        if(2*i <= n)  p[i]->lchild= p[2*i ];
        else p[i]->lchild= NULL;
        if(2*i+1 <= n) p[i]->rchild = p[2*i+1 ];
        else p[i]->rchild= NULL;
    }
    return p[1];
}
```

7.5 二叉树的其他操作算法

- 计算二叉树的结点数
- 计算二叉树的高度
- 根据关键值查找结点
- 查找结点的父结点

例1：计算二叉树结点数的算法

```
template <class T>
int BiTree<T>::Count(BiNode<T> *p)
{
    if( p==NULL )   return 0;
    left= Count(p->lchild);
    right=Count(p->rchild);
    return 1+left+right;
}
```

方法二: 计算二叉树结点数的算法

```
template <class T>
void BiTree<T>::Count(BiNode<T> *p, int &num)
{
    if( p==NULL )   return;
    num++;
    left= Count(p->lchild);
    right=Count(p->rchild);
}
```

例2：求二叉树的高度

```
template <class T>
int BiTree<T>::Height( BiNode<T> *t )
{
   if(t==NULL)  return 0;
   left =Height( t->lchild );
   right=Height( t->rchild );
   if(left>right) return left+1;
   return right+1;
}
```

方法二：二叉树的高度为树中的结点的层次最大值

参数如何设置？

```
template <class T>
void BiTree<T>::Height( BiNode<T> *t， int level,
                                  int& depth )
    if (t){
        if( level>depth) depth=level;
        Height(t->lchild, level+1, depth);
        Height(t->rchild, level+1, depth);
    }
}
```

例3：在二叉树中查找具有给定值的结点

```
template <class T>
BiNode<T> *BiTree<T>::Search(BiNode<T> *t, T e)
{
    if ( t==NULL) return NULL;
    if ( t->data == e ) return t;
    p= Search(t->lchild,e);
     if( p ) return p;
    return  Search( t->rchild,e) ;
}
```

例4：查找指定结点的父结点

```
template <class T>
BiNode<T> *BiTree<T>::SearchParent(BiNode<T> *t,
                                   BiNode<T>*child)
{
   if( t==NULL || child==NULL) return NULL;
   if(t->lchild==child || t->rchild==child)  return t;
   p= SearchParent(t->lchild, child);
   if( p ) return p;
   return SearchParent(t->rchild, child) );
}
```

练习题：设计一个算法，在二叉树中查找关键值为key的结点的父结点。

```
template <class T>
BiNode<T>* SearchParent(BiNode<T> * t ,  T key)
{
    if( t == NULL)  return  NULL;
    if( t->lchild && t->lchild->data==key ) return  t;
    if( t->rchild && t->rchild->data==key) return  t;
    p= SearchParent(t->lchild, key);
    if( p ) return p;
    return  SearchParent( t->rchild, key) ;
}
```

7.6 线索二叉树

- 二叉链表结构的局限性：
  - 对于某个结点只能找到其左右孩子，而不能直接得到该结点在某种遍历序列中的前趋或后继结点。
  - 要想得到该信息只能通过遍历的动态过程才行。
  - 怎样保存遍历过程中得到的信息呢？
- 解决方法：
  - 可利用二叉链表结点结构中的空指针域，在空指针域中存放结点在某种遍历次序下的前驱和后继结点信息，这种附加的指针称为“线索”。
  - 为避免混淆，需改变结点结构，即增加两个标志域。

ltype	lchild	data	rchild	rtype

ltype	lchild	data	rchild	rtype

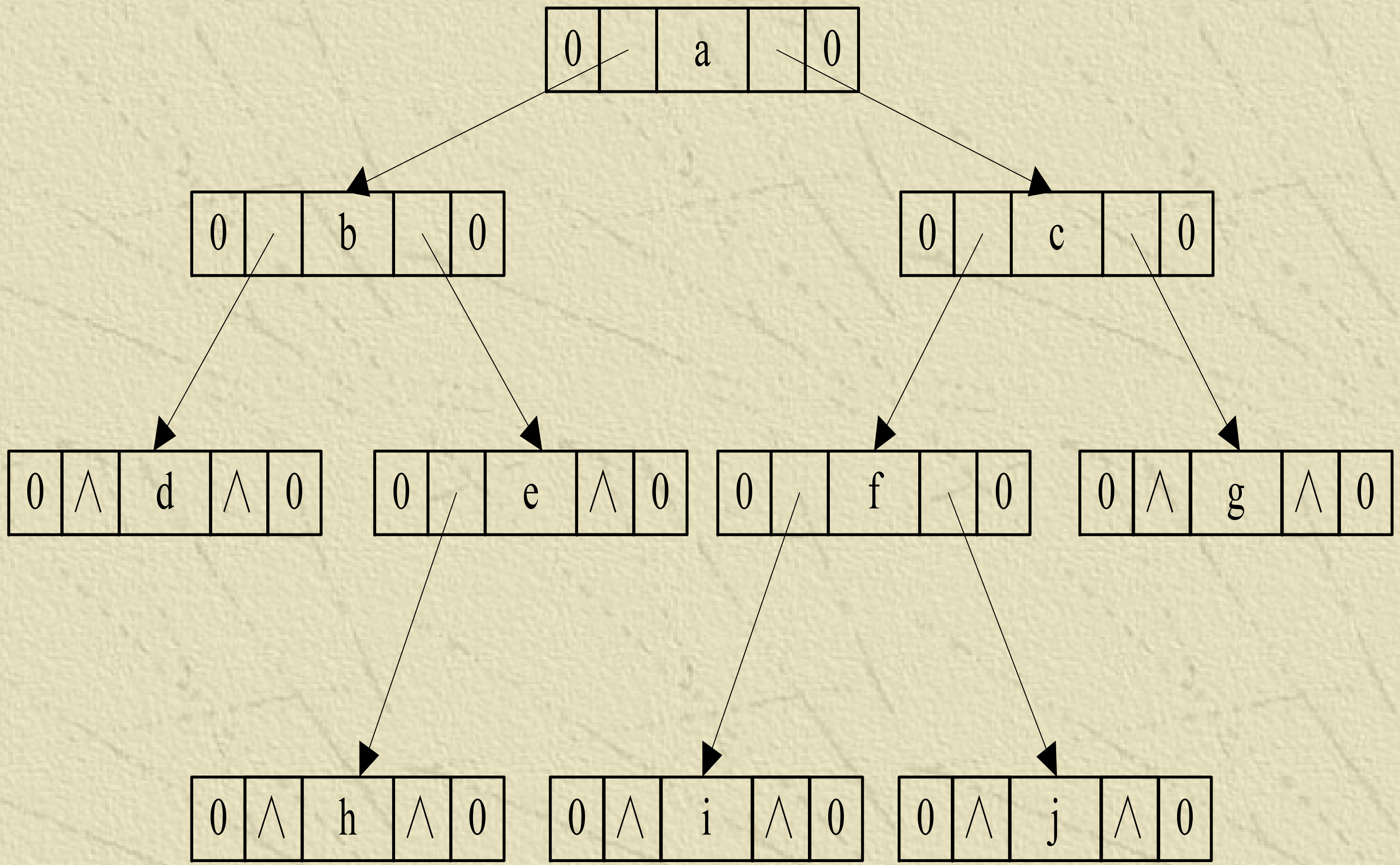
```
enum BiThrNodePointType{LINK, THREAD} ;
template <class T>
struct BiThrNode{
  BiThrNodePointType ltype, rtype;
  T data;
  BiThrNode<T> *lchild,*rchild;
};
```

标志位为0，表示指针指向孩子结点，

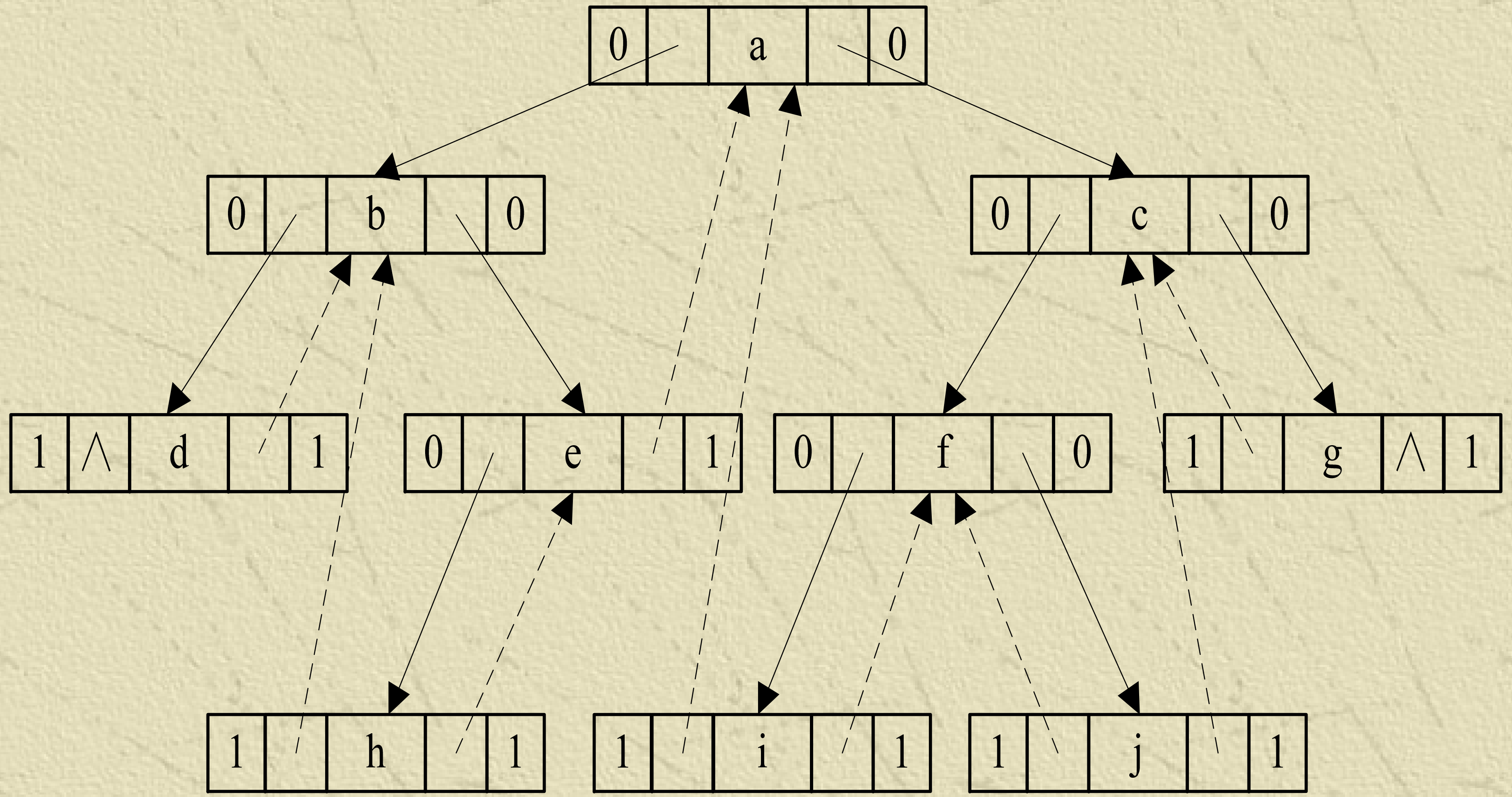
标志位为1，表示指针为线索。

有关概念：

- 以上面结构所构成的二叉链表作为二叉树的存储结构，叫做线索链表。指向结点前驱或后继的指针叫做**线索**。
- 加上线索的二叉树叫**线索二叉树**。
- 线索化：对二叉树以某种次序遍历使其变为线索二叉树的过程叫做**线索化**。



0 a 0
0 b 0
0 c 0
0 ∧ d ∧ 0
0 e ∧ 0
0 f 0
0 ∧ g ∧ 0
0 ∧ h ∧ 0
0 ∧ i ∧ 0
0 ∧ j ∧ 0



中序线索二叉树

中序线索二叉树类：

```
template <class T>
class InBiThrTree
{
    BiThrNode<T> *root;
 public:
    BinThrTree();
    ~BinThrTree();
    void InThreaded();    // 中序线索化
    ………
};
```

线索化的实现：

- 如何实现线索化？
  - 只要按该某种次序（先序、中序、后序）遍历二叉树，在遍历过程中，用线索取代空指针。
- 如何确立结点之间的前趋与后继关系？
  - 若指针**p**指向当前正在访问的结点，可另外附设一个指针**pre**，并始终保持指针**pre**指向当前访问的、指针**p**所指结点的前驱。

// 中序线索化算法：

```
template <class T>
void InBiThrTree<T>::InThreaded( BiThrNode<T> *p,
                                  BiThrNode<T> * &pre){
    if( p ){
        InThreaded(p->lchild, pre);      // 递归左子树线索化
        if( !p->lchild ) {               // 没有左孩子
            p->ltype=THREAD ;            // 前驱线索
            p->lchild=pre;               // 左孩子指针指向前驱
        }
        if( pre && !pre->rchild ){       // 前驱没有右孩子
            pre->rtype=THREAD ;          // 后继线索
            pre->rchild=p;               // 前驱右孩子指针指向后继
        }
        pre = p;                         // 保持pre指向p的前驱
        InThreaded(p->rchild, pre);  // 递归右子树线索化
    }
}
```

```
template <class T>
void InBiThrTree<T>::InThreaded()
{  if(root==NULL)  return;
   pre = NULL;      // 遍历序列中首结点的前驱为NULL
   InThreaded(root, pre);
   pre->rtype=THREAD;
   pre->rchild=NULL;  // 中序序列尾结点的后继为NULL
}
```

问题:

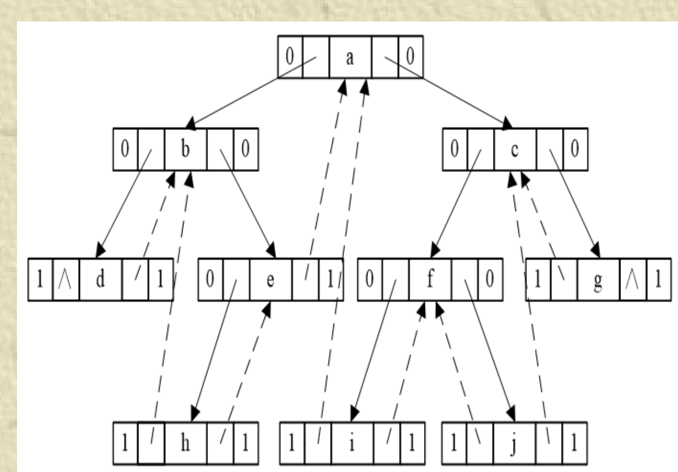
- 设计一个算法, 判断一棵给定的二叉树的中序遍历结点序列是否为非递减有序.

```
template <class T>
bool  JudgeOrder (BiNode<T> *p, BiNode<T> *&pre)
{
    if( !p ) return true;
    if( JudgeOrder (p->lchild, pre) ){
        if( pre && pre->data > p->data ) return false;
        pre = p;
        return  JudgeOrder (p->rchild, pre);
     }
     else  return false;
}
```

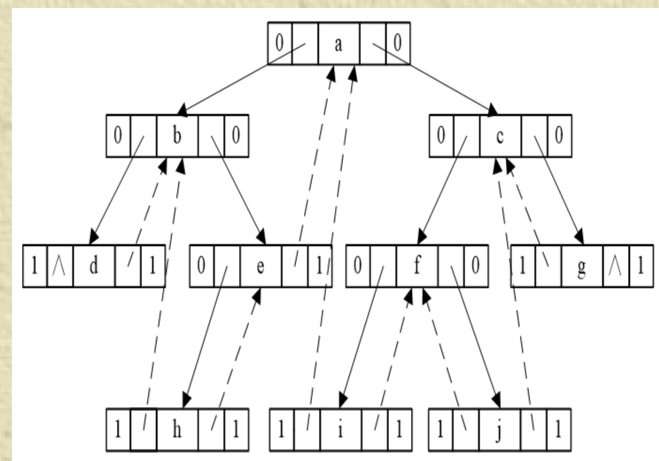
中序线索二叉树中，查找指定结点*p的中序后继结点

1、若 *p 的右子树为空，则 p->rchild 为右线索，直接指向 *p 的中序后继结点。

2、若 *p 的右子树非空，则 *p 的中序后继必是其右子树中第一个遍历到的结点，也就是从*p 的右孩子开始，沿左指针链往下查找，直到找到一个没有左孩子的结点为止。



对中序线索二叉树进行中序遍历的算法：

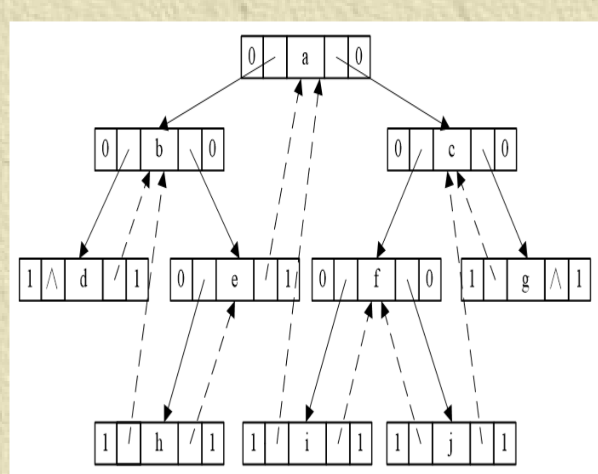


```
template <class T>
void InBiThrTree<T>::Travese() {
    p=root;
    if( !p ) return;
    while( p ){
          while( p->ltype==Link ) p = p->lchild;
          cout<< p-data;
          while( p->rtype == THREAD && p->rchild){
                 p = p->rchild; cout<< p->data;
          }
          p = p->rchild;
    }
}
```

中序线索二叉树中，查找指定结点*p的中序前驱结点

1、若 *p 的左子树为空，则 p->lchild 为左线索，直接指向 *p 的中序前驱结点。

2、若 *p 的左子树非空，则从 *p 的左孩子出发，沿右指针链往下查找，直到找到一个没有右孩子的结点为止。



后序线索二叉树中，查找指定结点*p的后序后继结点

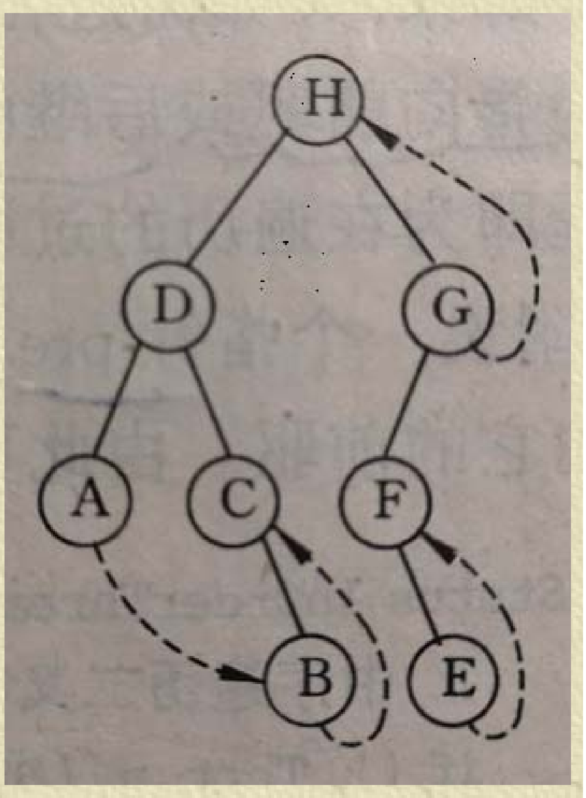
分四种情况：

1、若 *p是根，则其后继为空；

2、若 *p是其双亲的右孩子，则*p的后继就是其双亲结点；

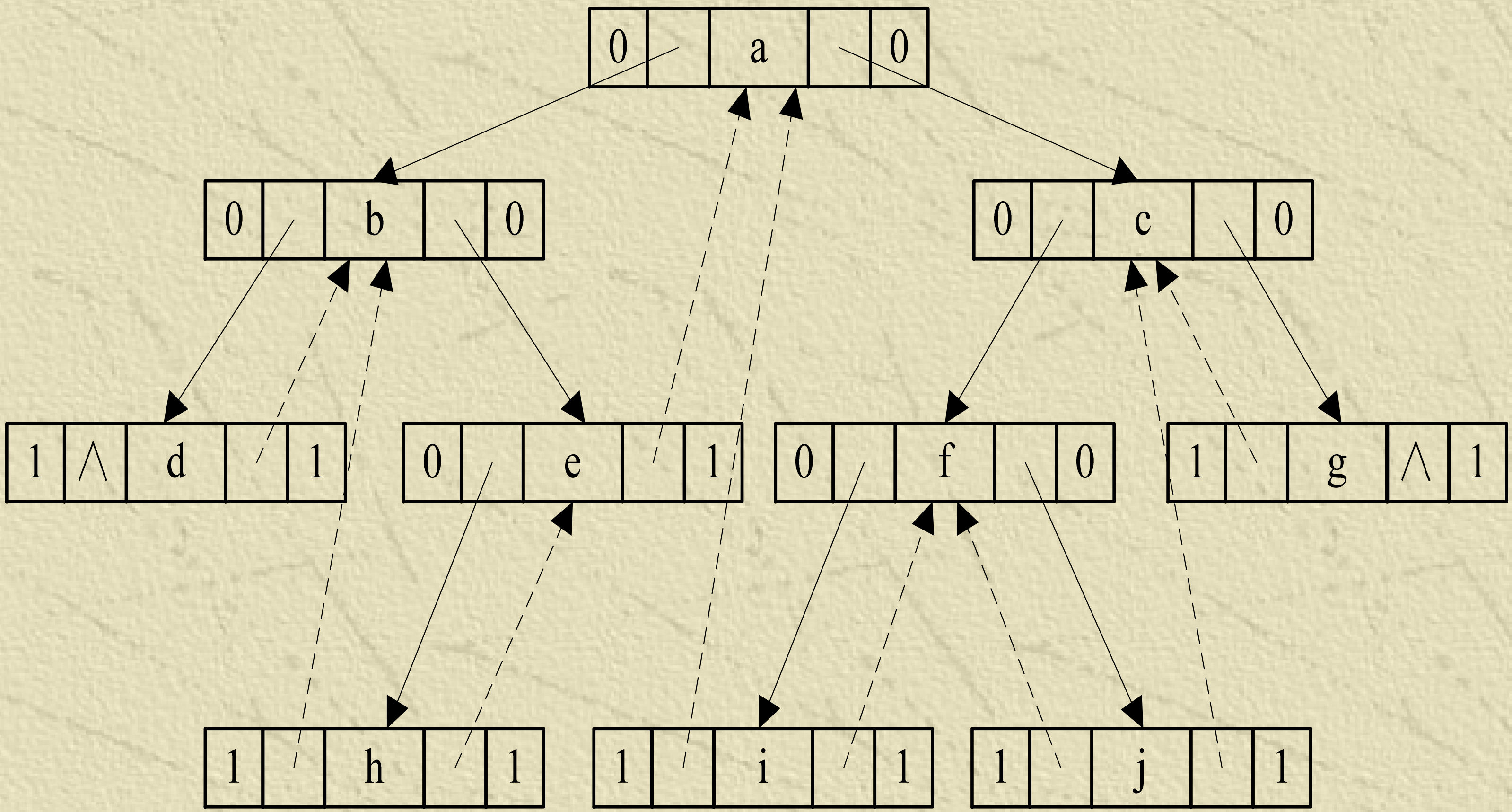
3、若 *p是其双亲的左孩子，但*p无右兄弟时，则*p的后继就是其双亲结点；

4、若 *p是其双亲的左孩子，且有右兄弟时，则*p的后继是其双亲的右子树中第一个后序遍历到的结点，它是该子树的“最左下的叶结点”。

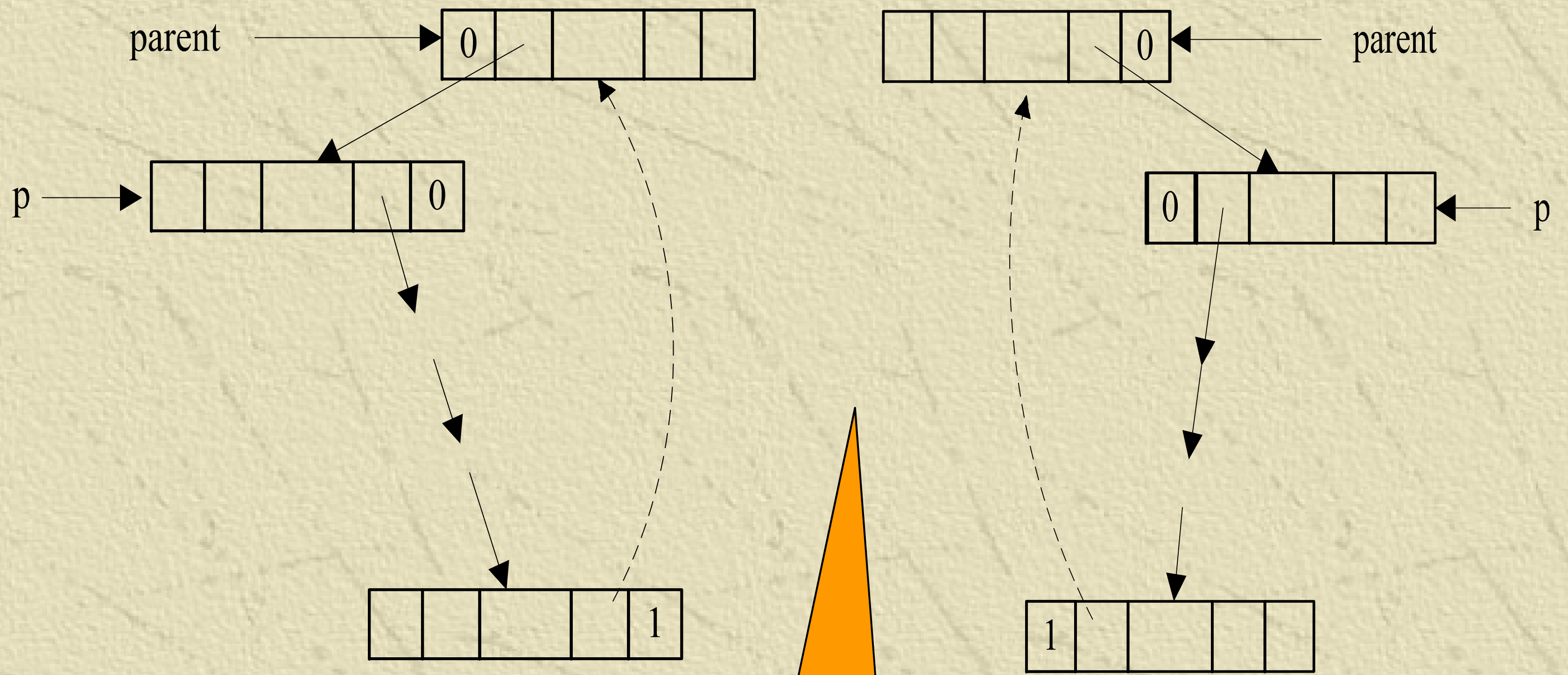


未必能找到！

4、求父结点的算法



*p和*parent存在如下两种情形：



试探法

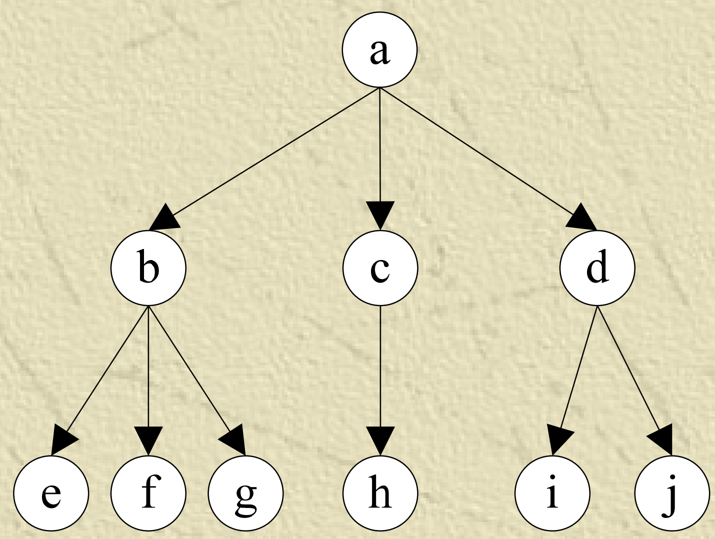
```
template <class T>
BiThrNode<T> *InBiThrTree<T>::GetParent(BiThrNode<T> *p)
{    if( !p || p==root ) return NULL;
     for( parent=p; parent->rtype==LINK; )
          parent=parent->rchild;
     parent=parent->rchild;    // parent是*p的最右下方结点的后继指针
     if( parent && parent->lchild==p) // 猜测*p是否是左孩子
          return parent;
     for( parent=p; parent->ltype==LINK; )
           parent=parent->lchild;
     parent=parent->lchild; // parent是*p的最左下方结点的前驱指针
     return parent;         // parent一定是*p的父指针
}
```

7.7 树的存储结构与算法

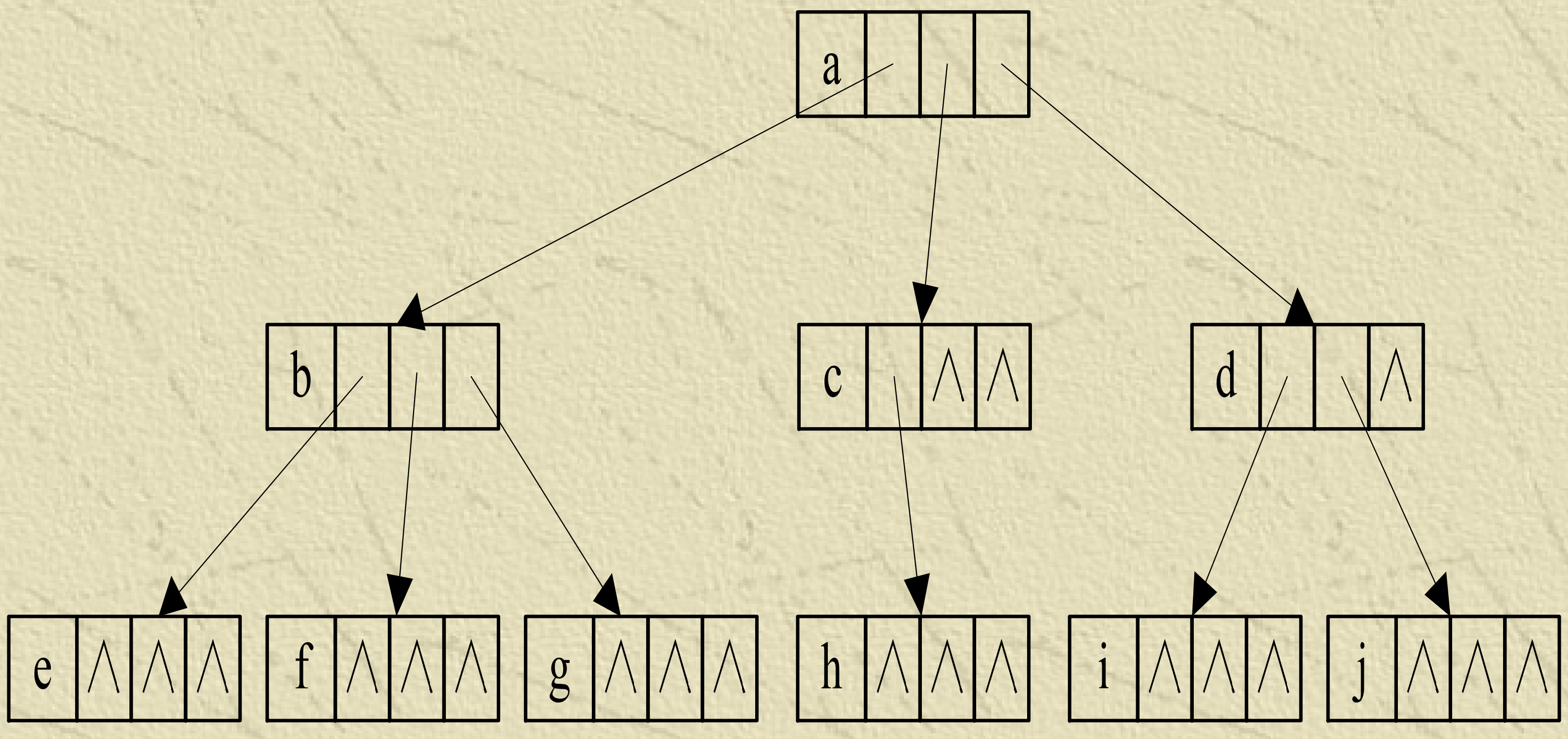
- 多叉链表表示法
- 广义表表示
- 孩子表示法
- 双亲表示法
- 双亲-孩子链表示法
- 二叉链表表示法

1. 多叉链表表示法

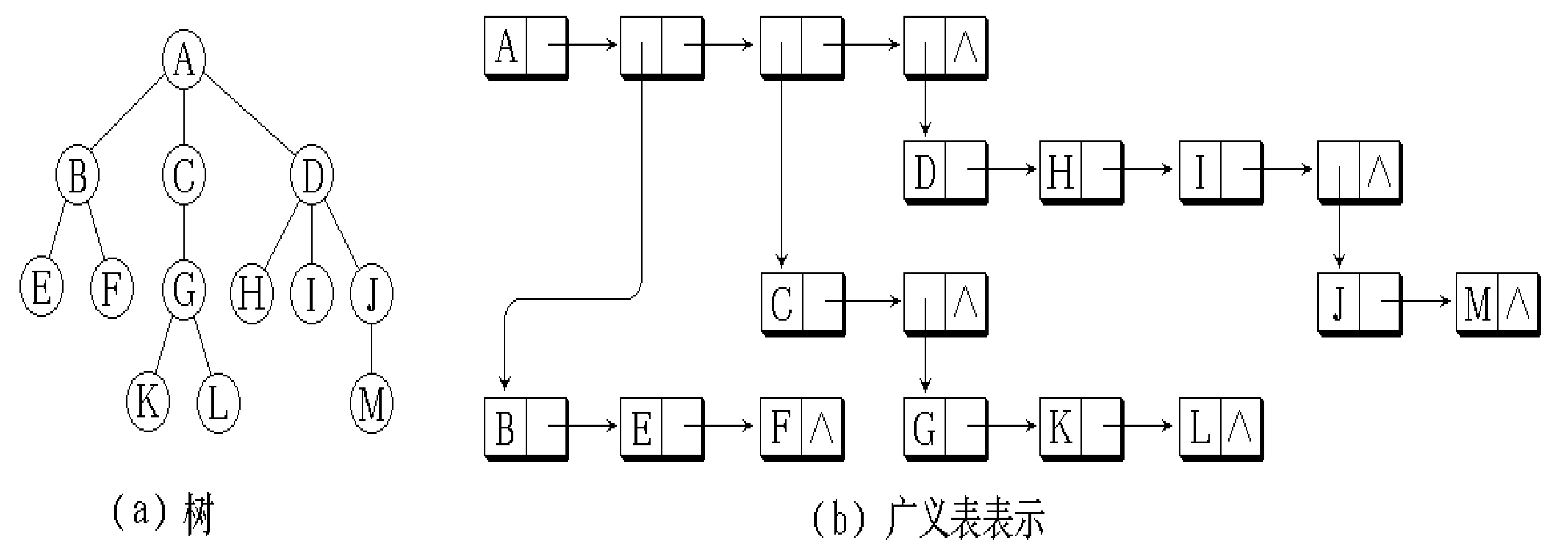
- 若树的度为*K*，则在结点结构中设置*K*个孩子指针域，使所有结点同构。



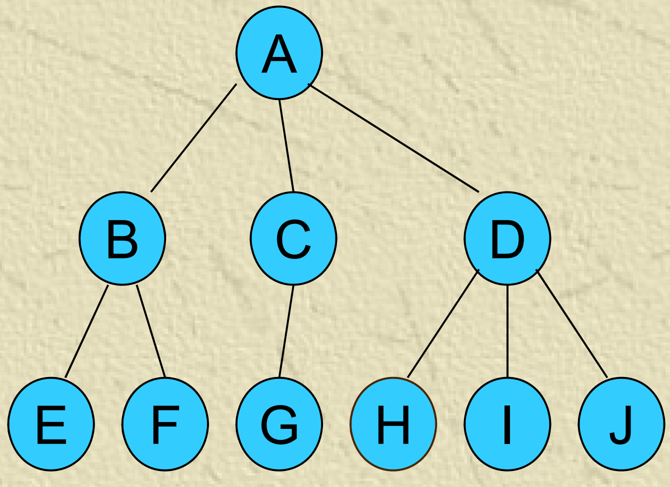
树的多叉链表表示

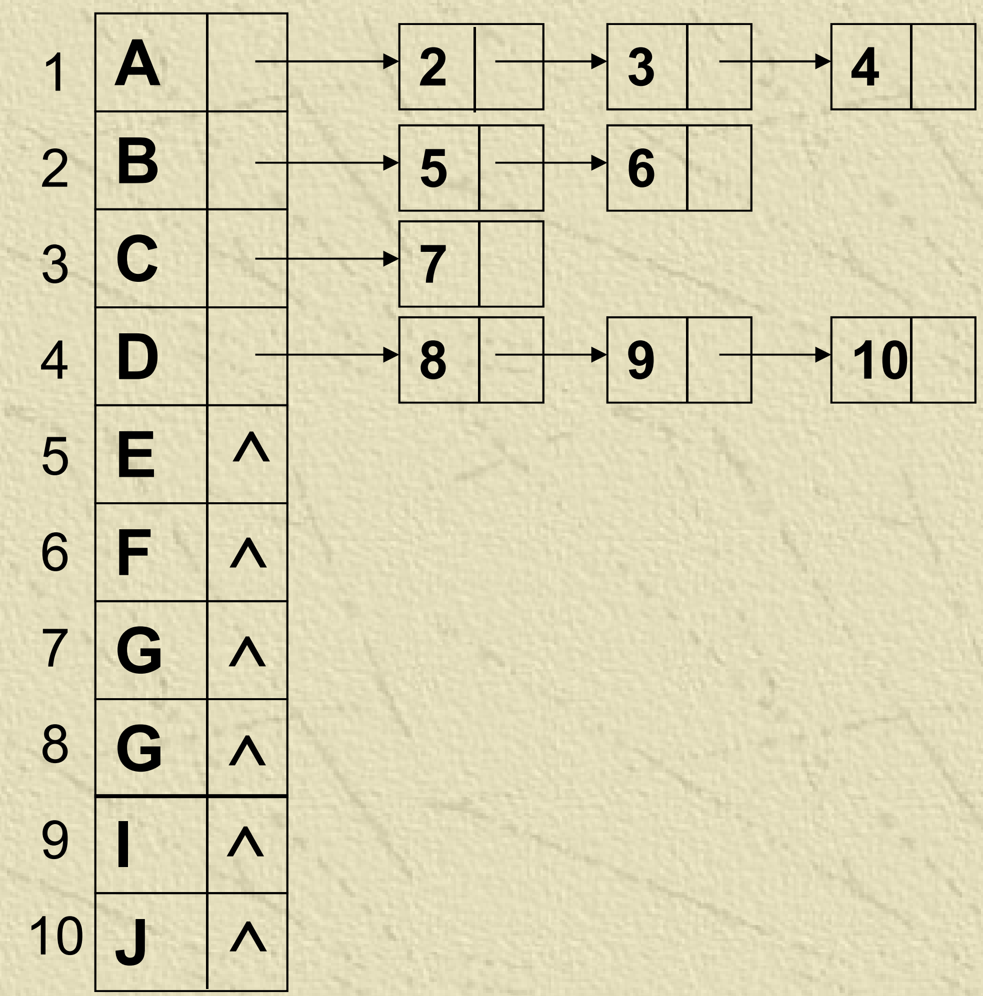


2. 树的广义表表示

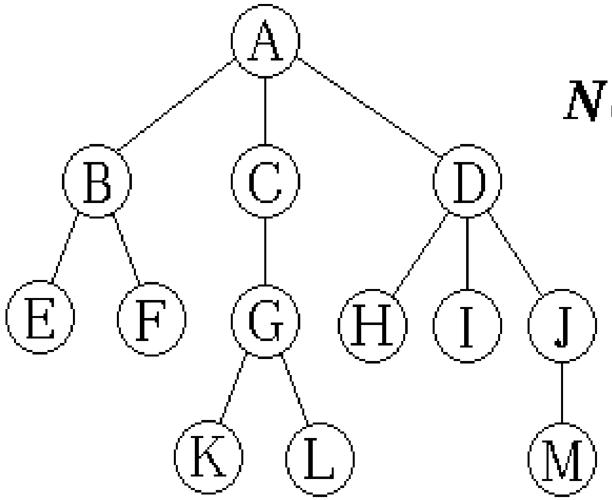


3. 孩子表示法：





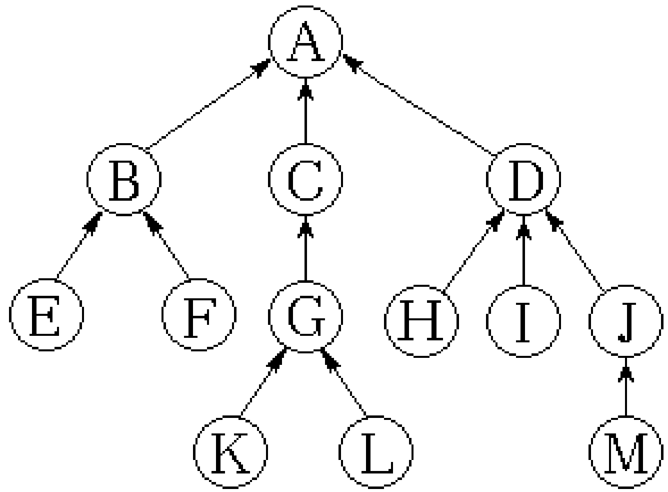
4. 双亲表示法：



（a）树

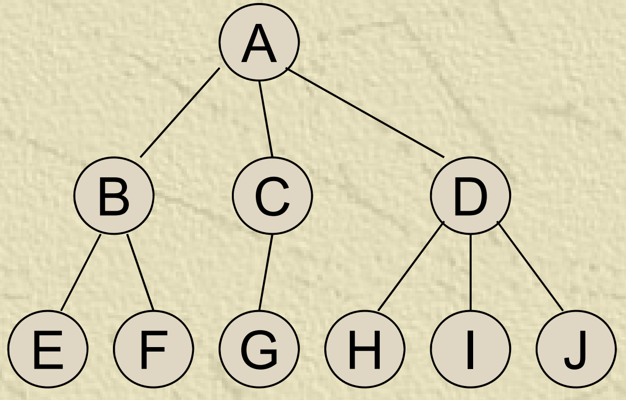
NodeList	1	2	3	4	5	6	7	8	9	10	11	12	13
data	A	B	E	F	C	G	K	L	D	H	I	J	M
parent	0	1	2	2	1	5	6	6	1	9	9	9	12

（b）双亲表示数组

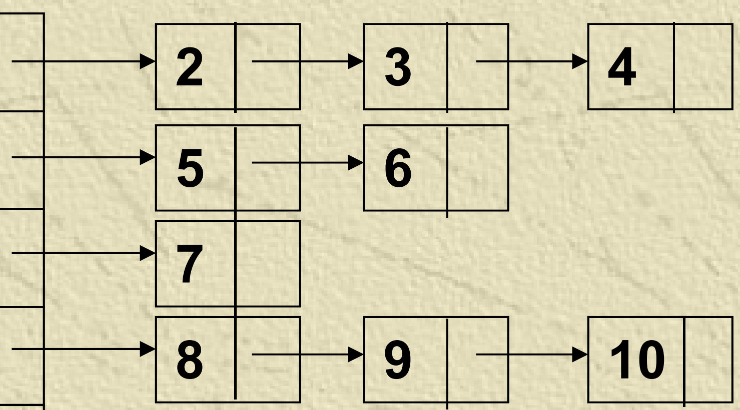


（c）双亲表示图解

5. 双亲-孩子链表示：

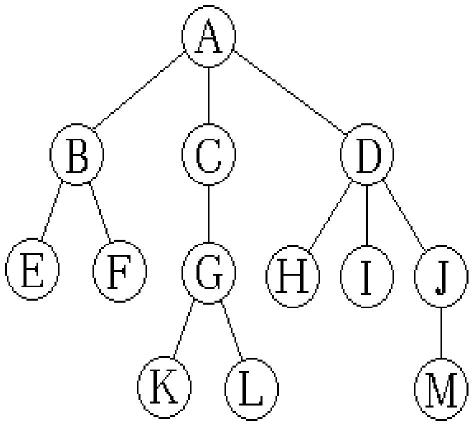


1	A	0	
2	B	1	
3	C	1	
4	D	1	
5	E	2	^
6	F	2	^
7	G	3	^
8	G	4	^
9	I	4	^
10	J	4	^



6. 树的二叉链表存储表示法（孩子兄弟表示法）

data	firstChild	nextSibling

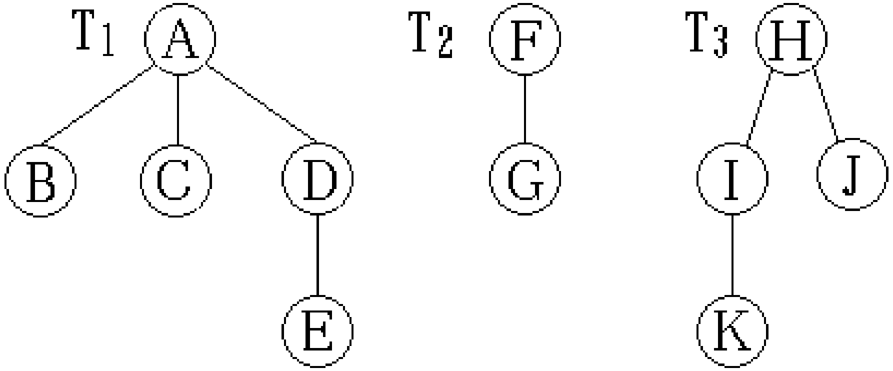


（a）树

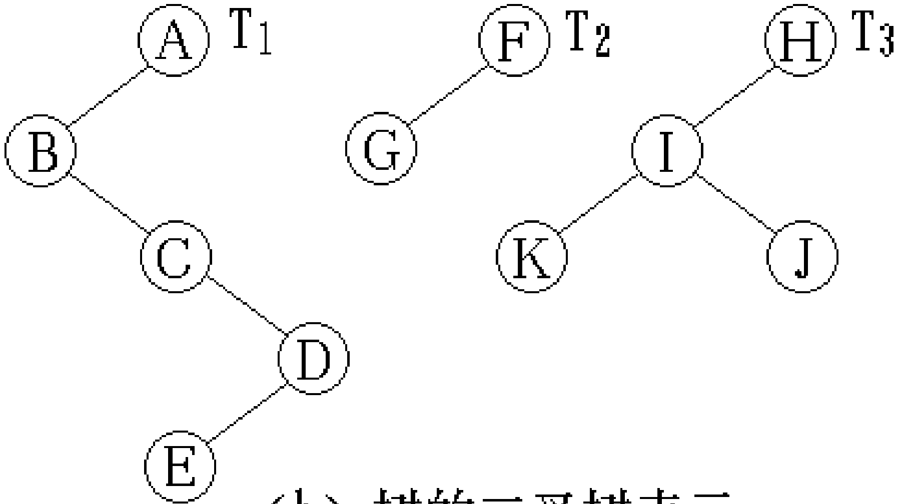
森林与二叉树的转换

- 树、森林与二叉树之间有一个自然的一一对应关系。
- 转换方法：
  - 树 → 二叉树
    - 兄弟结点间加连接（虚线）
    - 让每个结点只与最左孩子保持联系，与其余孩子的关系去掉
  - 森林 → 二叉树
    - 森林 → 树 → 二叉树 （添加一个虚根结点）
    - 去掉虚根结点

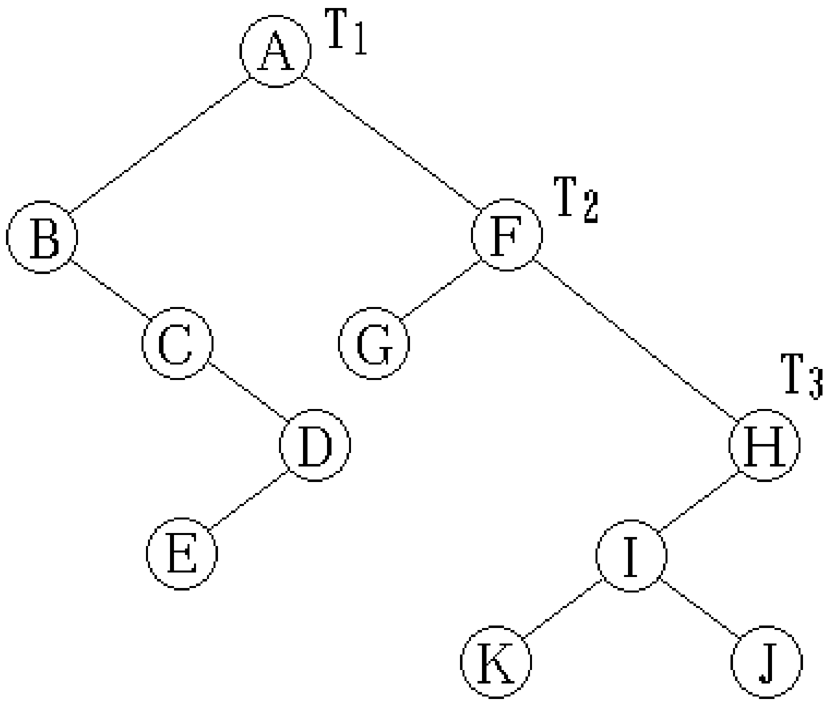
森林与二叉树的转换



（a） 3棵树的森林



（b）树的二叉树表示



（c）森林的二叉树表示

森林与二叉树的对应关系