

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

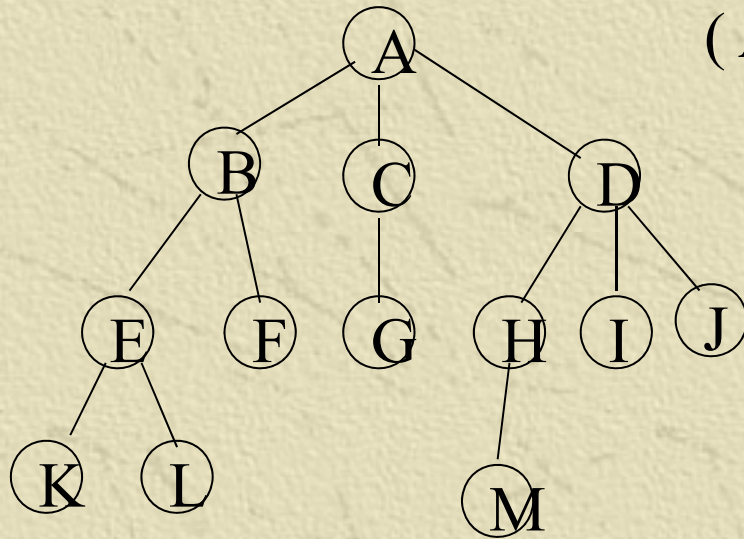
 树的定义:

是 n ($n \geq 0$)个结点的有限集合 T , 对于任意一棵非空树, 它满足:

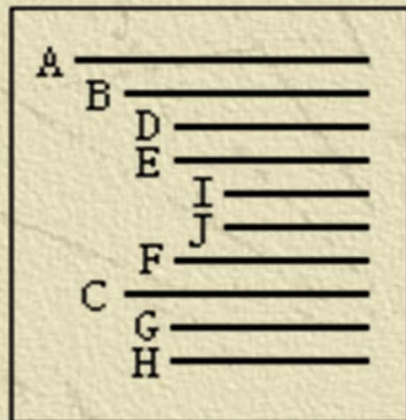
- (1) 有且仅有一个特定的称为根的结点;
- (2) 当 $n > 1$ 时, 其余结点可分为 m ($m > 0$)个互不相交的有限集 T_1, T_2, \dots, T_m , 其中每个集合本身又是一棵树, 称为根的子树。

显然: 上述树的定义是一个递归定义。

树的表示方法:



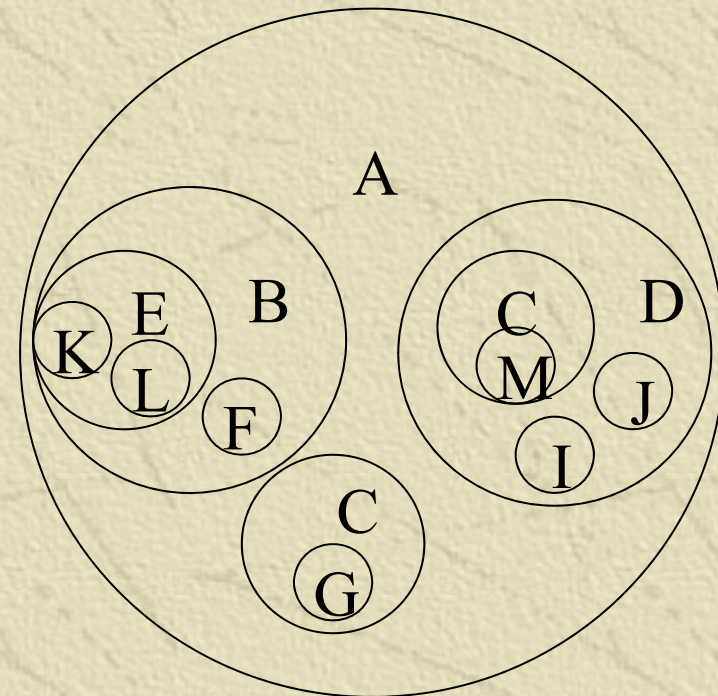
树形表示



凹入表

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

广义表



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**)

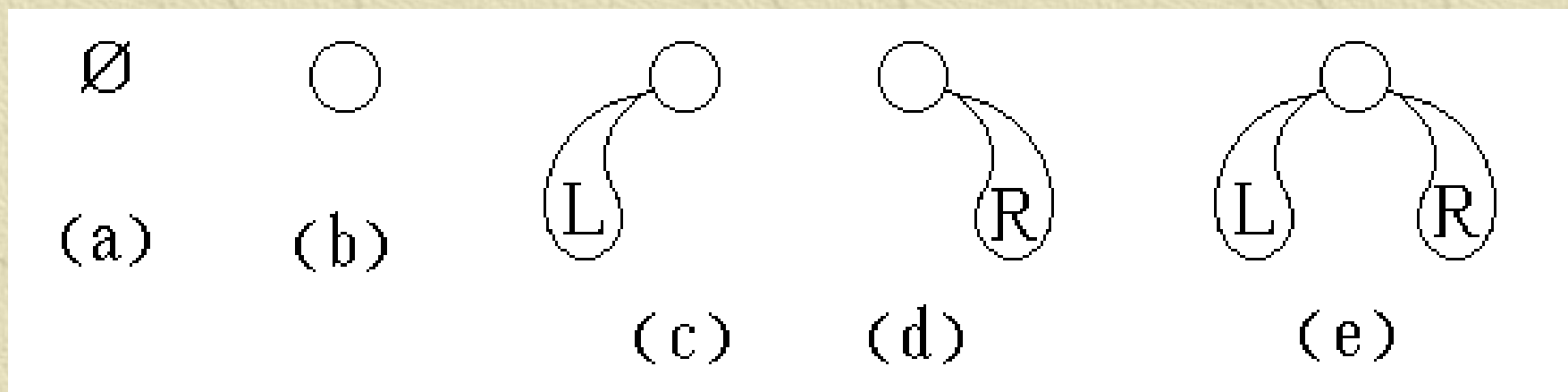
7.2 二叉树的概念与性质

二叉树的定义

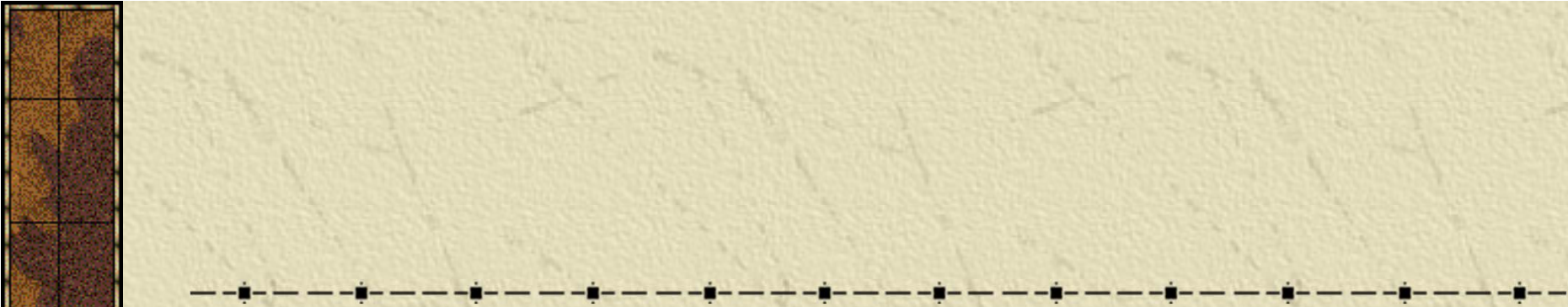
一棵二叉树是 n ($n \geq 0$)个结点的一个有限集合，

(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：

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

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：

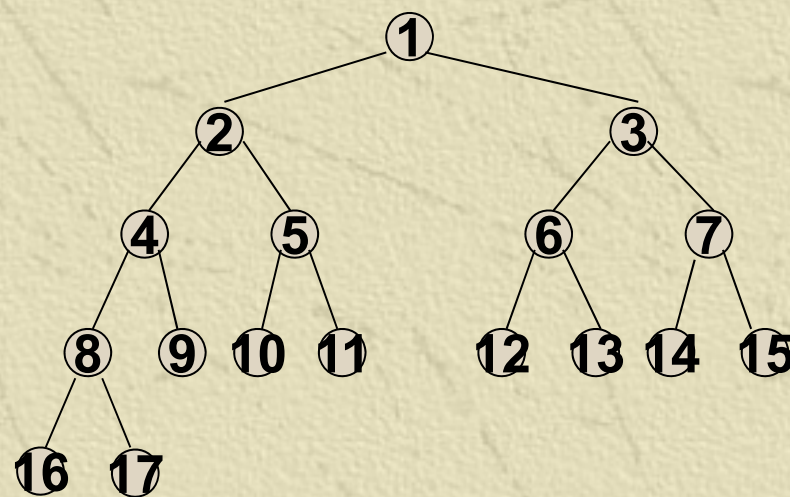
$$2^{k-1} - 1 < n \leq 2^k - 1, \text{ 即:}$$

$$2^{k-1} \leq n < 2^k, \text{ 于是有:}$$

$$k - 1 \leq \log_2 n < k$$

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

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 $1, 2, \dots, n-1, n$ ，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 ($1 \leq i \leq n$)。则有以下关系：



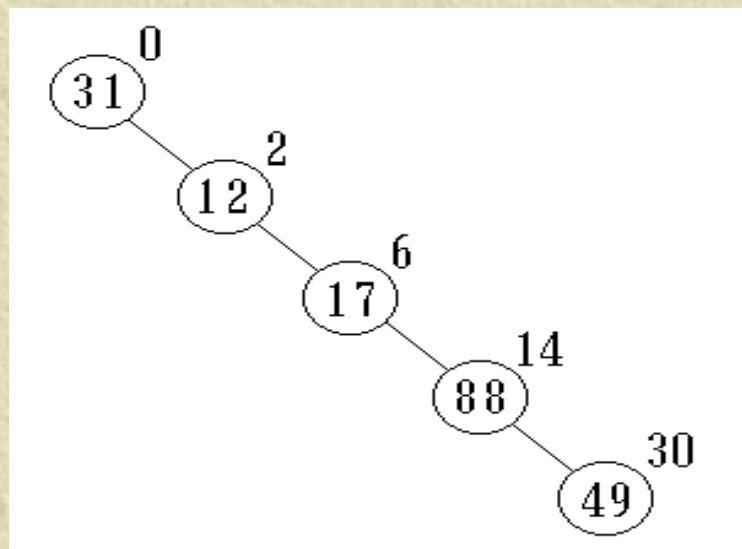
- 若 $i = 1$ ，则 i 无双亲
若 $i > 1$ ，则 i 的双亲为 $\lfloor i/2 \rfloor$
- 若 $2*i \leq n$ ，则 i 的左孩子为 $2*i$ ；否则， i 无左孩子，必定是叶结点，二叉树中 $i > \lfloor n/2 \rfloor$ 的结点必定是叶结点
若 $2*i+1 \leq n$ ，则 i 的右孩子为 $2*i+1$ ，否则， i 无右孩子
- 若 i 为奇数，且 i 不为1，则其左兄弟为 $i-1$ ，否则无左兄弟；
若 i 为偶数，且小于 n ，则其右兄弟为 $i+1$ ，否则无右兄弟
- i 所在层次为 $\lfloor \log_2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

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

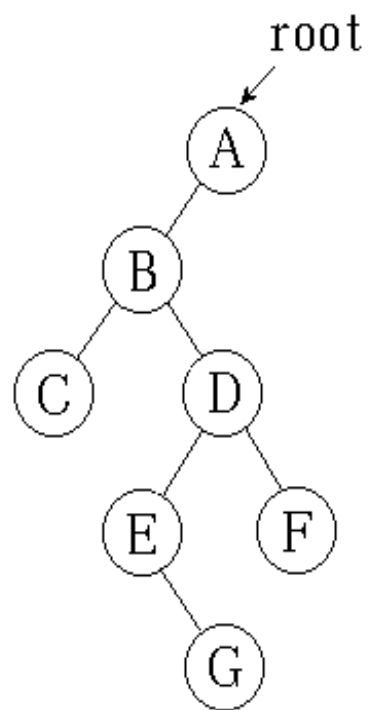
- 若要在树中经常插入和删除结点时，由于要大量移动结点，显然在这种情况下采用顺序方式并不可取。

二. 链式存储结构

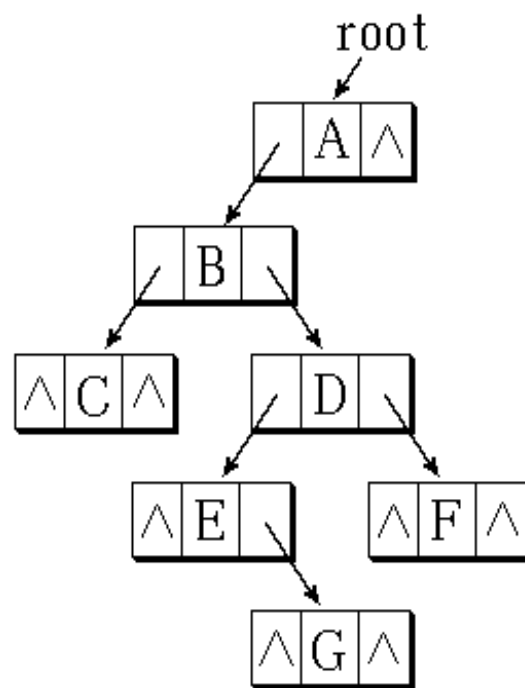
- ✧ 由于二叉树的每个结点最多有左、右两个孩子，因此在采用链式存储表示时，每个结点至少需要包含三个域：数据域和左、右指针域。

lchild	Data	rchild
--------	------	--------

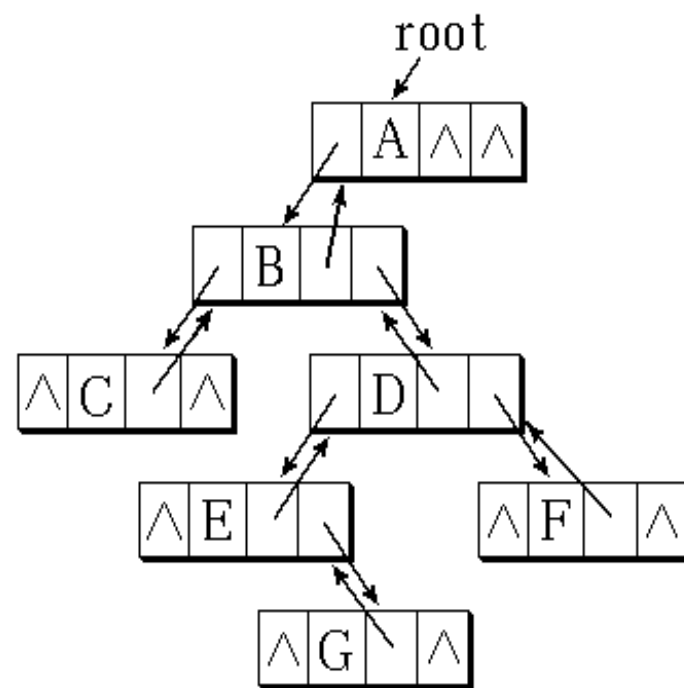
- ✧ 一个二叉树中所有这种形式的结点，再加上一个指向根结点的头指针，就构成了此二叉树的链式存储结构，称之为**二叉链表**。
- ✧ 如果想能够找到父结点，则可以增加一个指向父结点的指针域，则构成**三叉链表**。



(a) 二叉树



(b) 二叉链表



(c) 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

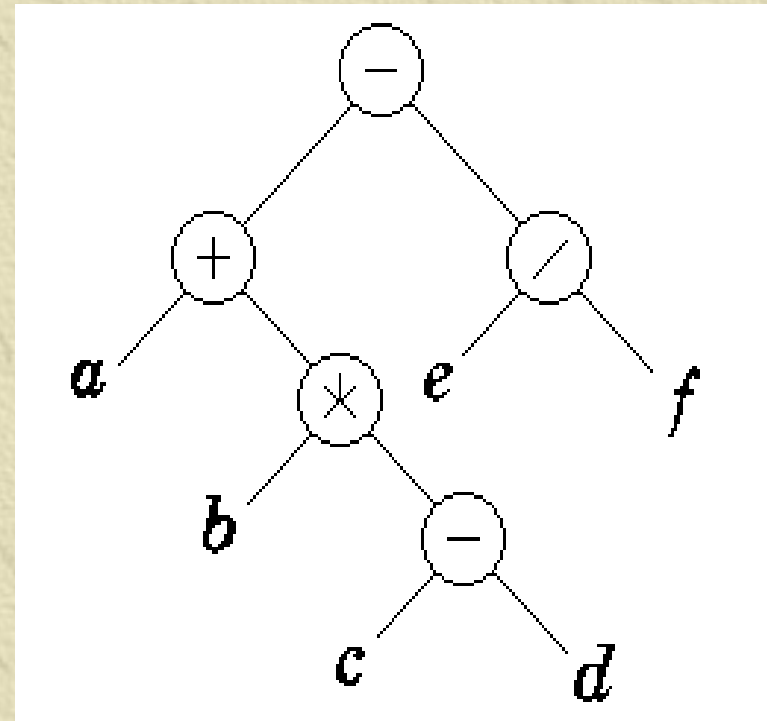
先序遍历

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：

$- + a * b - c d / e f$

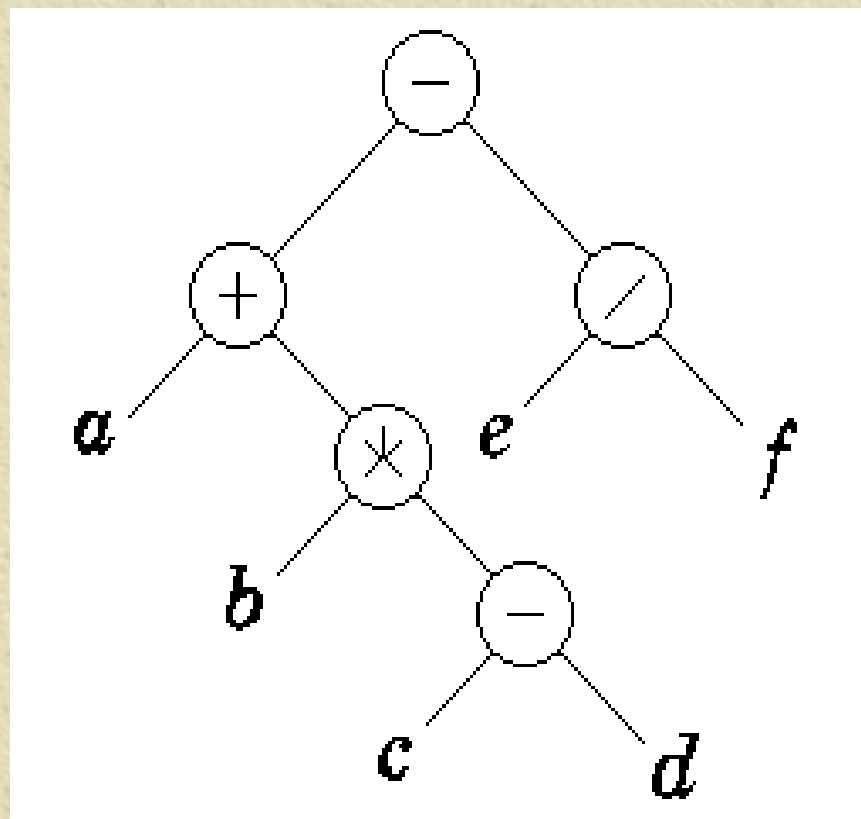


中序遍历

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

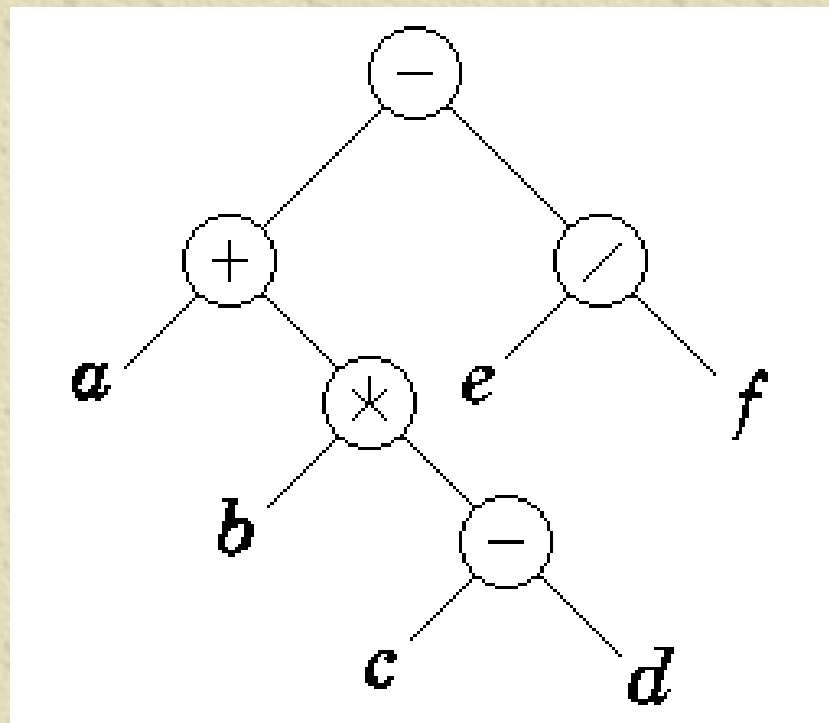
后序遍历

后序遍历二叉树算法的框架是

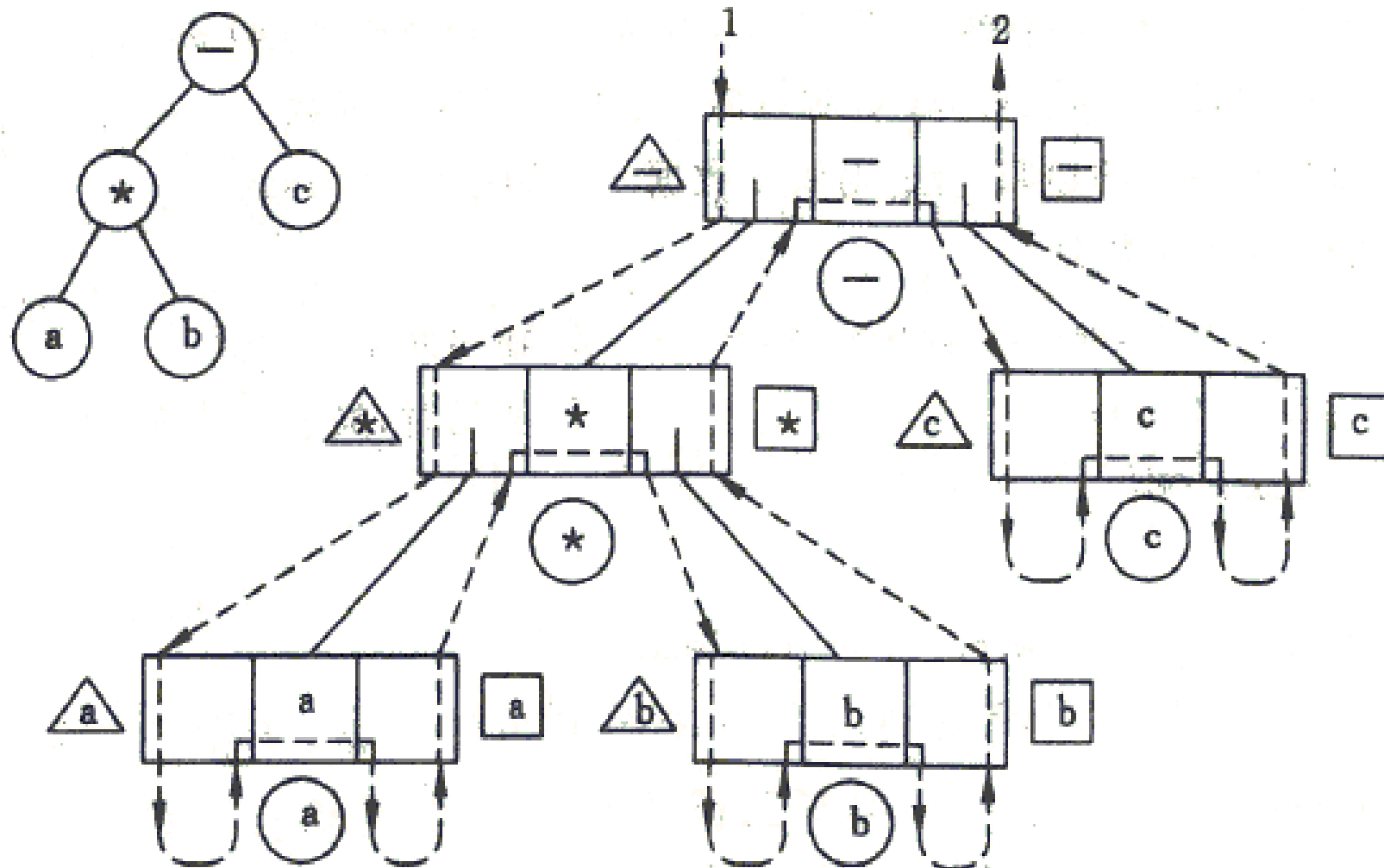
- 若二叉树为空，则空操作；
- 否则
 - 后序遍历左子树 (**L**)；
 - 后序遍历右子树 (**R**)；
 - 访问根结点 (**V**)。

遍历结果：

$a b c d - * + e f / -$



先、中、后序遍历的流程

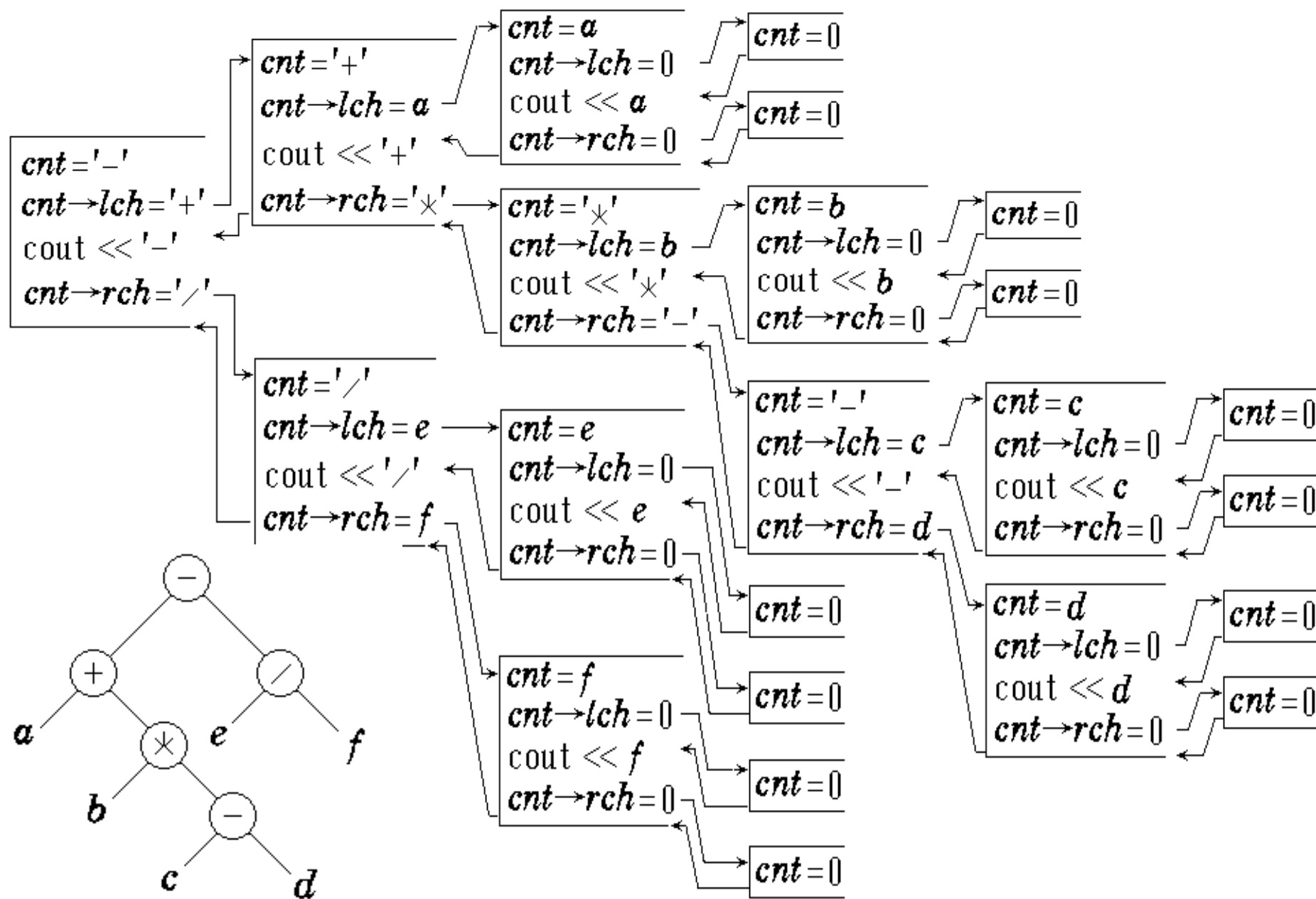


中序遍历的递归算法:

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

算法的
时间复杂度?

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

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？