

软件技术基础

第九讲

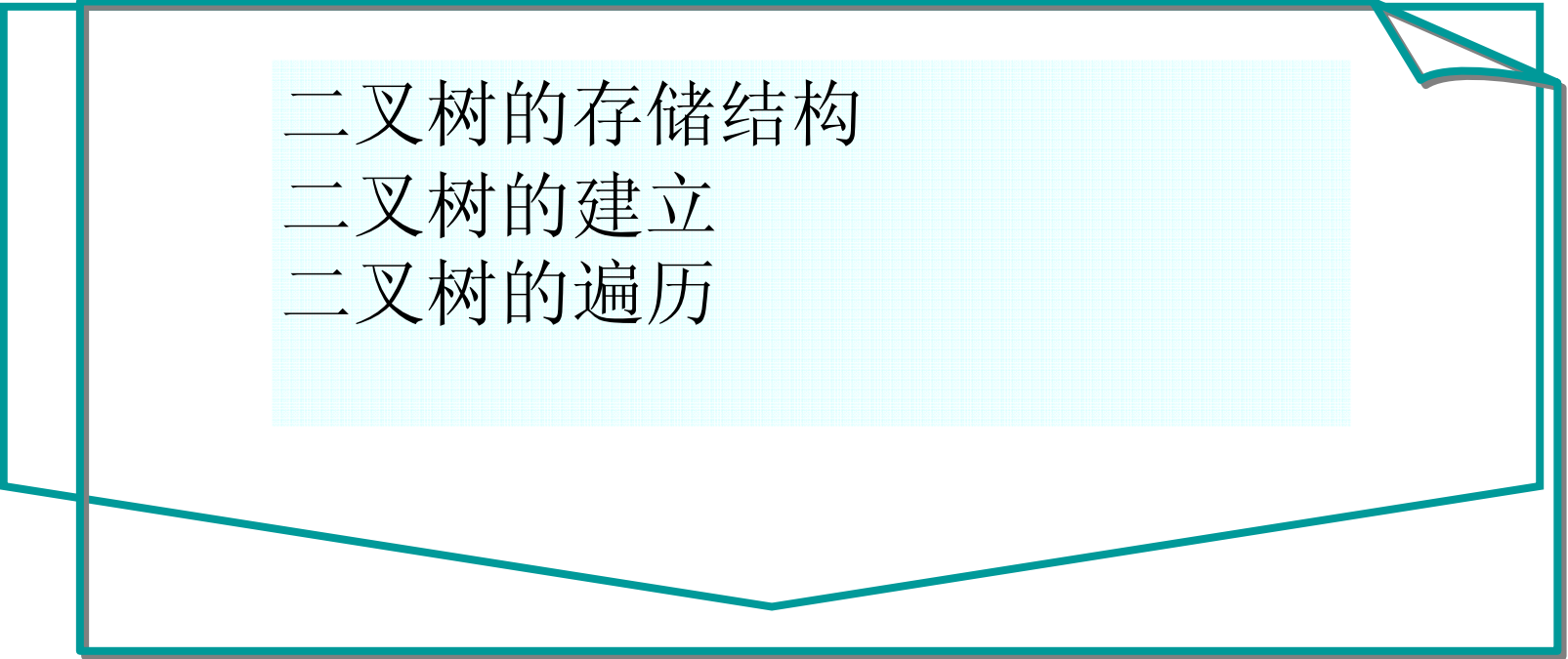


上讲主要内容

- 树的基本概念
- 二叉树的基本概念

树

本章主要内容



- 二叉树的存储结构
- 二叉树的建立
- 二叉树的遍历

二叉树的存储结构

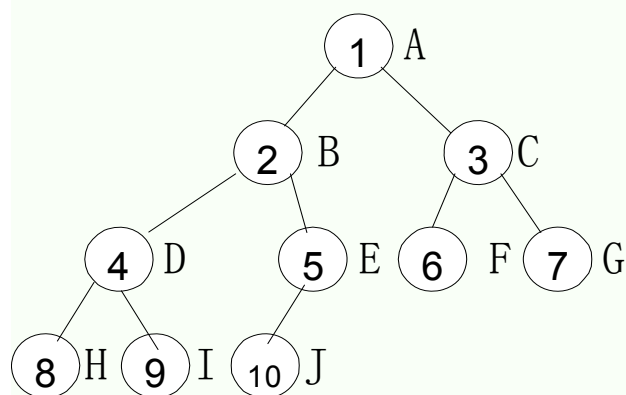
二叉树的存储结构

- ▶ 顺序存储结构
- ▶ 链式存储结构

顺序存储二叉树时，首先必须对树形结构的结点进行某种方式的**线性化**，使之成为一个**线性序列**，然后存储。为此必须必须借助于**完全二叉树及其编号的性质**。因此首先讨论完全二叉树的顺序存储。

完全二叉树的顺序存储结构

在一棵完全二叉树中，按照从根结点起，自上而下，从左至右的方式对结点进行顺序编号，便可得到一个反映结点之间关系的线性序列。



完全二叉树的结点编号

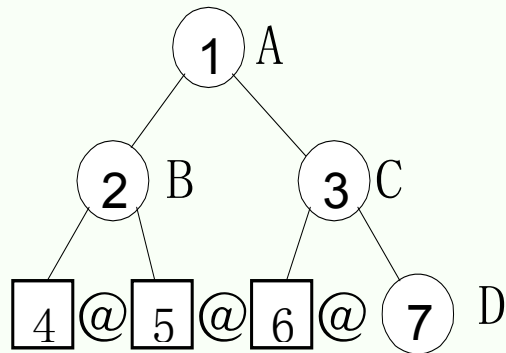
左图的完全二叉树的顺序存储

编 号	0	1	2	3	4	5	6	7	8	9	10
结点值		A	B	C	D	E	F	G	H	I	J

一般二叉树的顺序存储

按完全二叉树的方式存储一般二叉树

- 将二叉树映射为完全二叉树（通过虚结点）；
- 用完全二叉树的方式存储。



一般二叉树的结点编号

表 10.2 一般二叉树的顺序存储

编 号	0	1	2	3	4	5	6	7
结点值		A	B	C	@	@	@	D

二叉树的链式存储

含有两个指针域来分别指向左孩子指针域(**lchild**)和右孩子指针域(**rchild**), 以及结点数据域(**data**), 故二叉树的链式存储结构也称为二叉链表。

```
typedef struct BiTNode {  
    TelemType data;  
    struct BiTNode  
    *lchild,*rchild;  
} BiTNode,*BiTree;
```

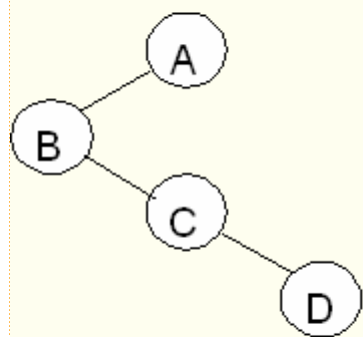
其中**root**是指向根结点的头指针, 当二叉树为空时, 则**root=NULL**。若结点某个孩子不存在时, 则相应的指针为空。

二叉树的链式存储

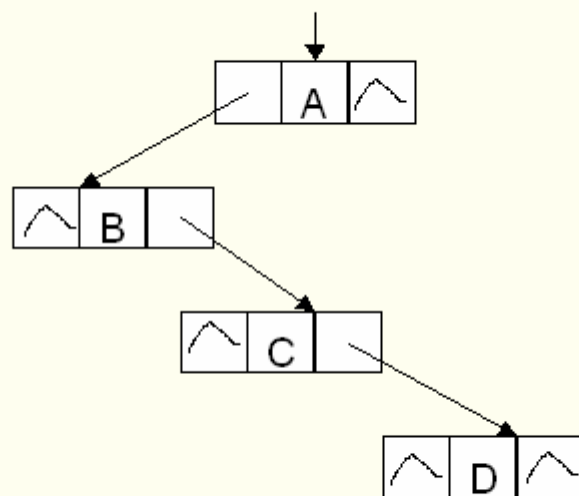
三叉链表 二叉链表中，要寻找某结点的双亲是困难的，故增加一个指向其双亲的指针域**parent**。

```
typedef int datatype;
typedef struct
{
    datatype data;
    struct node *lchild, *rchild, *parent;
} bitree;
bitree *root;
```

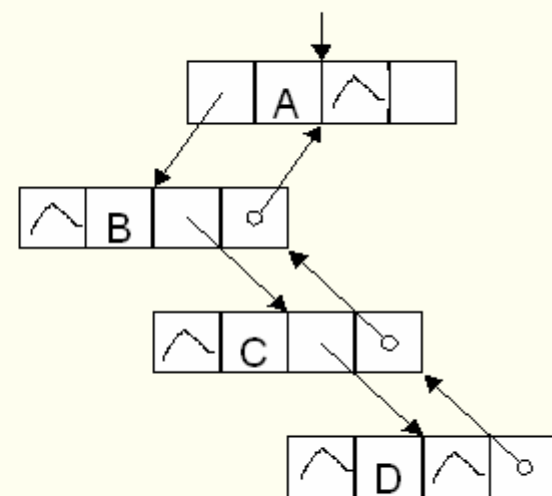

二叉树的链式存储实例



(a) 二叉树



(b) 二叉链表



(c) 三叉链表

二叉树的遍历

[定义]

二叉树的遍历是指按某种搜索路线来巡访二叉树中的每一个结点，使每个结点被且仅被访问一次。

[遍历的结果]

产生一个关于结点的线性序列。

[两种遍历方式]

- ✓ 深度优先遍历
- ✓ 广度优先遍历

二叉树的深度优先遍历

则有六种不同的二叉树深度优先遍历方案：

L->遍历左子树
D->访问根结点
R->遍历右子树

DLR
LDR
LRD
DRL
RDL
RLD

若限定按先左后右进行遍历

DLR(先(前)序(根)遍历)
LDR(中序(根)遍历)
LRD(后序(根)遍历)。

先序遍历算法

先序遍历算法的遍历过程是：

若二叉树非空，执行以下操作：

- ✓访问根结点；
- ✓先序遍历左子树；
- ✓先序遍历右子树；

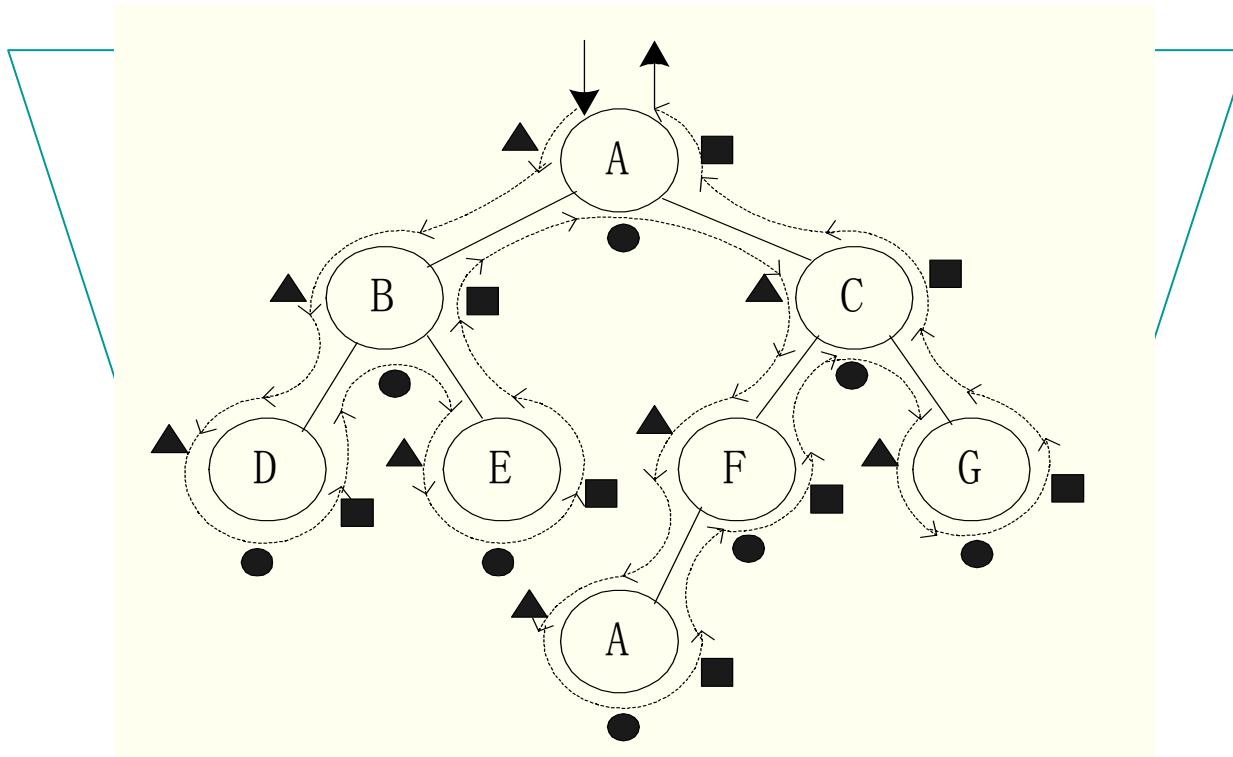
```
void preorder(biTree *T)
```

```
{ ① if (T!=NULL)
  { ② printf (" %c ", T->data);
    ③ preorder (T->lchild);
    ④ preorder (T->rchild);
  }
  ⑤ return;
}
```

```
/* 先序遍历二叉树，p指向  
   二叉树的根结点 */  
/* 二叉树p非空 */  
/* 访问p所结点 */  
/* 先序遍历左子树 */  
/* 先序遍历右子树 */  
  
/* 返回 */
```

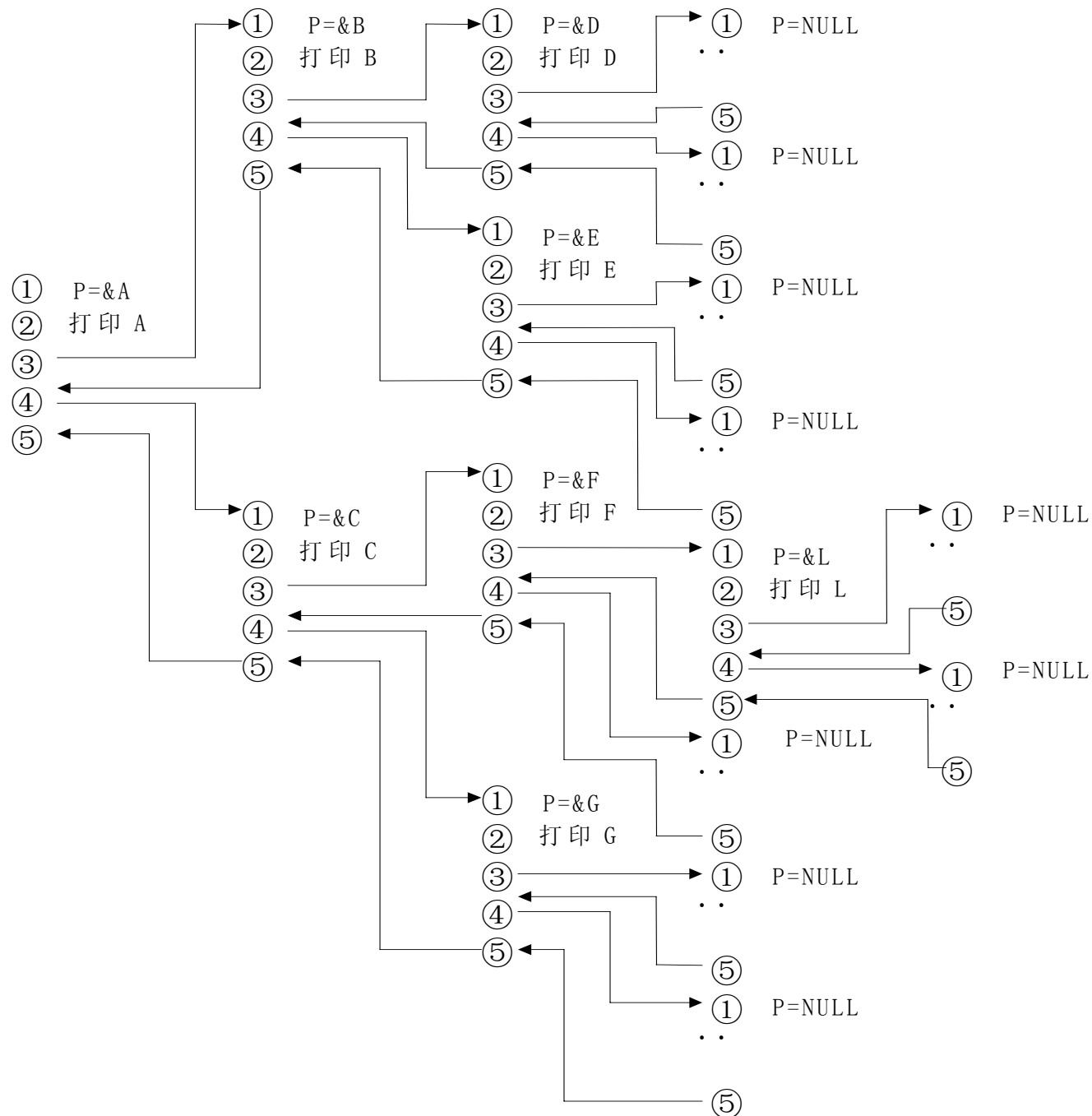
先序遍历算法执行过程

先序遍历算法执行过程（如图三角表示）



先序遍历结果序列为**A, B, D, E, C, F, L, G**

先序遍历算法执行过程



中序遍历算法

中序遍历算法的遍历过程是：

若二叉树非空，执行以下操作：

- ✓中序遍历左子树；
- ✓访问根结点；
- ✓中序遍历右子树；

```
void inorder(biTree *T)
```

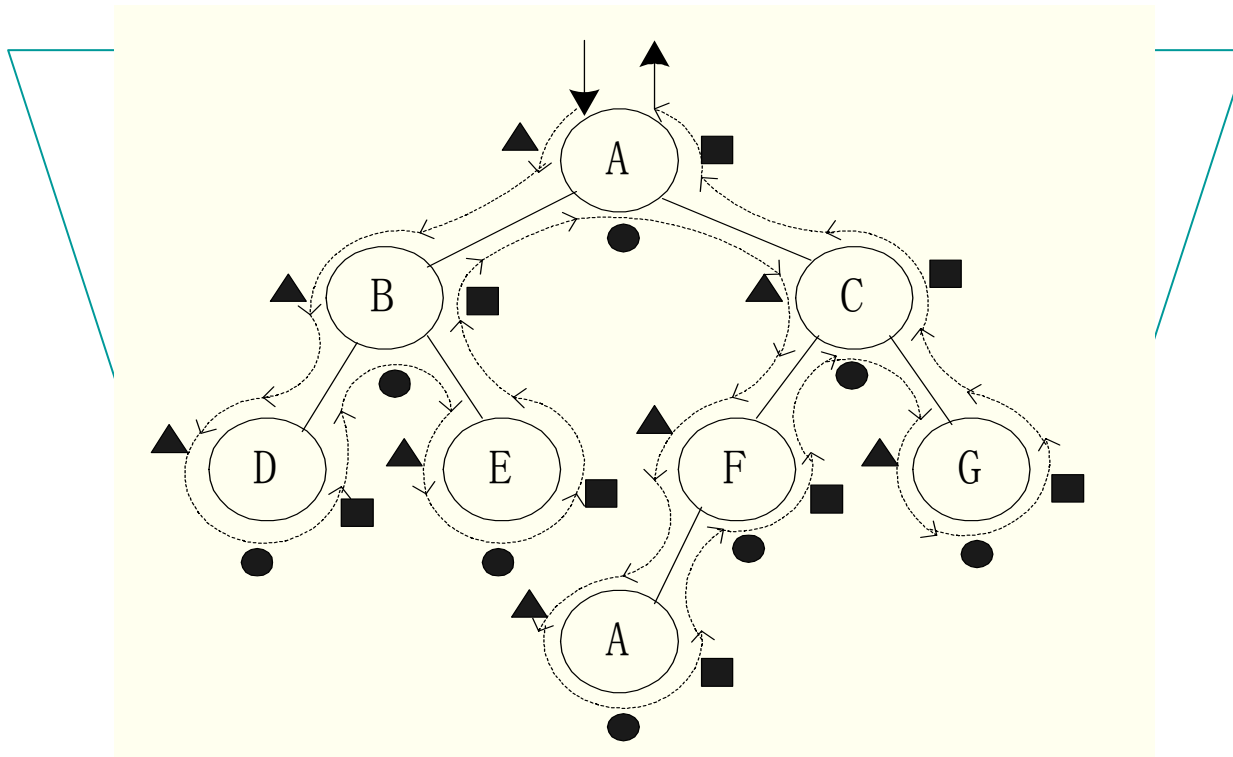
```
{ ① if (T!=NULL)
  { ② inorder (T→lchild);
    ③ printf (" %c ", T→data);
    ④ inorder (T→rchild);
  }
  ⑤ return;
}
```

```
/* 中序遍历二叉树，p指向二叉树的根结点 */
/* 二叉树p非空 */
/* 中序遍历左子树 */
/* 访问p所结点 */
/* 中序遍历右子树 */

/* 返回 */
```

中序遍历算法执行过程

先序遍历算法执行过程（如图原点表示）



中序遍历结果序列为**D, B, C, A, L, F, C, G**

后序遍历算法

后序遍历算法的遍历过程是：

若二叉树非空，执行以下操作：

- ✓中序遍历左子树；
- ✓访问根结点；
- ✓中序遍历右子树；

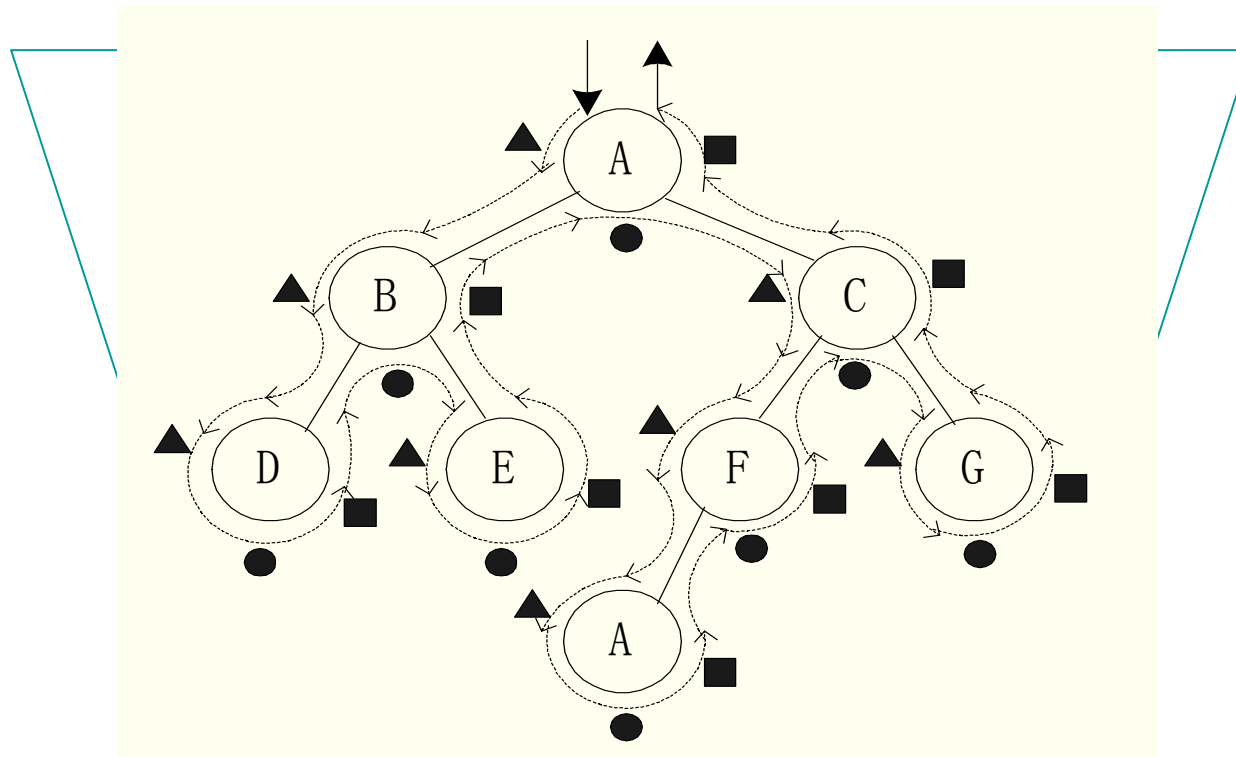
```
void postorder(bitree *T)
{ ① if (T!=NULL)
  { ② postorder (T→lchild);
    ③ postorder (T→rchild);
    ④ printf (" %c ", T→data);
  }
  ⑤ return;
}
```

```
/* 后序遍历二叉树，p指向二叉树的根结点 */
/* 二叉树p非空 */
/* 后序遍历左子树 */
/* 后序遍历右子树 */
/* 访问p所结点 */

/* 返回 */
```

后序遍历算法执行过程

先序遍历算法执行过程（如图方框表示）



中序遍历结果序列为**D, E, B, L, F, G, C, A**

先序建立二叉树的递归算法

```
Status CreateBiTree(BiTree &T)  
{ char ch;    scanf("%c",&ch);  
    if (ch==' ') T=NULL;  
    else { if (!(T=(BiTNode*)malloc(sizeof(BiTNode)))) exit(OVERFLOW);  
        T->data = ch;  
        CreateBiTree(T->lchild);  
        CreateBiTree(T->rchild);  
    }  
    return OK;  
}
```

二叉树的显示输出算法一

```
void PrintTreep (bitree*p)
{
    if(p!=NULL)
    {
        cout<<p->data;
        if(p->lchild!=NULL||p->rchild!=NULL)
        {
            cout<<"(";
            preorder(p->lchild);
            if(p->rchild!=NULL)cout<<",";
            preorder(p->rchild);
            cout<<")";
        }
    }
}
```

二叉树的显示输出算法二

```
void PrintBiTree(BiTree T,int n)
{
    int i; char ch=' ';
    if (T) {
        PrintBiTree(T->rchild,n+1);
        for (i=1;i<=n;++i) {printf("%5c",ch);}
        printf("%c\n", T->data);
        PrintBiTree(T->lchild,n+1);
    }
}
```

二叉树的广度优先遍历

二叉树的**广度优先遍历**又称为**按层次遍历**，这种遍历方式是先遍历二叉树的第一层结点，然后遍历第二层结点，……最后遍历最下层的结点。而对每一层的遍历是按**从左至右**的方式进行。

二叉树的广度优先遍历

[基本思想]

按照广度优先遍历方式，在上层中先被访问的结点，它的下层孩子也必然先被访问，因此在这种遍历算法的实现时，需要使用一个队列。在遍历进行之前先把二叉树的根结点的存储地址入队，然后依次从队列中出队结点的存储地址，每出一个结点的存储地址则对该结点进行访问，然后依次将该结点的左孩子和右孩子的存储地址入队，如此反复，直到队空为止。

二叉树的深度优先遍历的非递归算法基本思想

[基本思想]

- 当p所指的结点**非空**时，将该结点的存储地址**进栈**，然后再将p指向该结点的**左孩子结点**；
- 当p所指的结点为**空**时，从栈顶**退出栈**顶元素送p，并**访问**该结点，然后再将p指向该结点的**右孩子结点**；
- 如此反复，直到**p为空**并且栈顶指针**top = -1**为止。

二叉树的深度优先遍历

顺序栈的定义如下：

```
typedef BiTNode* SElemType;
typedef struct{
    SElemType *base;
    SElemType *top;
    int stacksize;
}SqStack;
```

先序遍历的非递归算法

```
void preorder(BiTree T)
{
    SqStack S;    BiTree P=T;
    InitStack(S); Push(S,NULL);
    while (P)
    {
        printf("%c",P->data);
        if (P->rchild)
            Push(S,P->rchild);
        if (P->lchild)
            P=P->lchild;
        else Pop(S,P);
    }
}
```

小结

二叉树的建立

二叉树的存储

二叉树的遍历

顺序

链式

作业

P143

23; 25