6.4 遍历的应用

遍历是二叉树各种操作的基础，可以在遍历过程中对结点进行各种操作：

（1）求结点的双亲；

（2）求孩子结点；

（3）求结点的层次；

（4）遍历过程中生成结点，建立二叉树；

遍历二叉树的过程实质是把二叉树的结点进行线性排列的过程。

6.4 遍历的应用

6.4.1 遍历的基本应用

6.4.2 二叉树的遍历与存储结构的应用

6.4.3 二叉树的相似与等价

6.4 遍历的应用

6.4.1 遍历的基本应用

6.4.2 二叉树的遍历与存储结构的应用

6.4.3 二叉树的相似与等价

6.4.1 遍历的基本应用

• 二叉树的生成

递归建立二叉树

我们按先序递归遍历的思想来建立二叉树。

其建立思想如下：

（1）建立二叉树的根结点；

（2）先序建立二叉树的左子树；

（3）先序建立二叉树的右子树。

6.4.1 遍历的基本应用

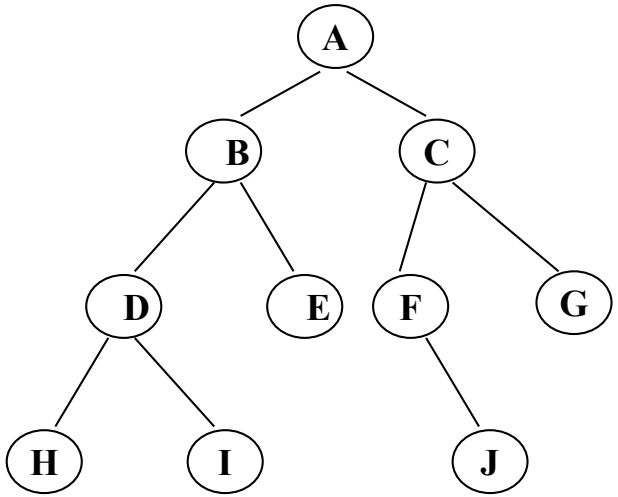
• 二叉树的生成

按先序序列建立二叉树的左右链结构。

如下图所示二叉树，输入：

ABDH##I##E##CF#J##G##

其中:#表示空




```
BTREE *CreateTree1()
{   BTREE *bt;
    char ch;
    fflush(stdin);
    scanf("%c",&ch);
    if(ch=='#')
              bt=Null;
    else
    { bt=New BNODE;
      if(!bt) exit(0);
      bt->data=ch;
      bt->lchild= CreateTree1();
      bt->rchild= CreateTree1();
     }
     return(bt);
}
```

6.4.1 遍历的基本应用

□ 求二叉树的叶子数。

算法思想：采用任何遍历方法，遍历时判断访问的结点是不是叶子，若是则叶子数加1。

```
int countleaf(BinTree  t, int num)
{ if(t!=NULL)
  {if((t->lch==NULL) &&(t- >rch)==NULL))
       num++;
   num=countleaf(t->lch,num);
   num=countleaf(t->rch,num);
  }
return num;
}
```

6.4.1 遍历的基本应用

□ 求二叉树的深度

算法思想：左右子树中深度更大的那棵树的深度+1。

求二叉树深度的递归算法(后序遍历)

```
int treedepth(BinTree  t)
 {int h,lh,rh;
  if(t==NULL)    h=0;
  else     { lh=treedepth(t->lch);
             rh=treedepth(t->rch);
             if(lh>=rh)  h=lh+1;
              else   h=rh+1; }//加上第一层的根节点
   return h;
 }
```

6.4 遍历的应用

6.4.1 遍历的基本应用

6.4.2 二叉树的遍历与存储结构的应用

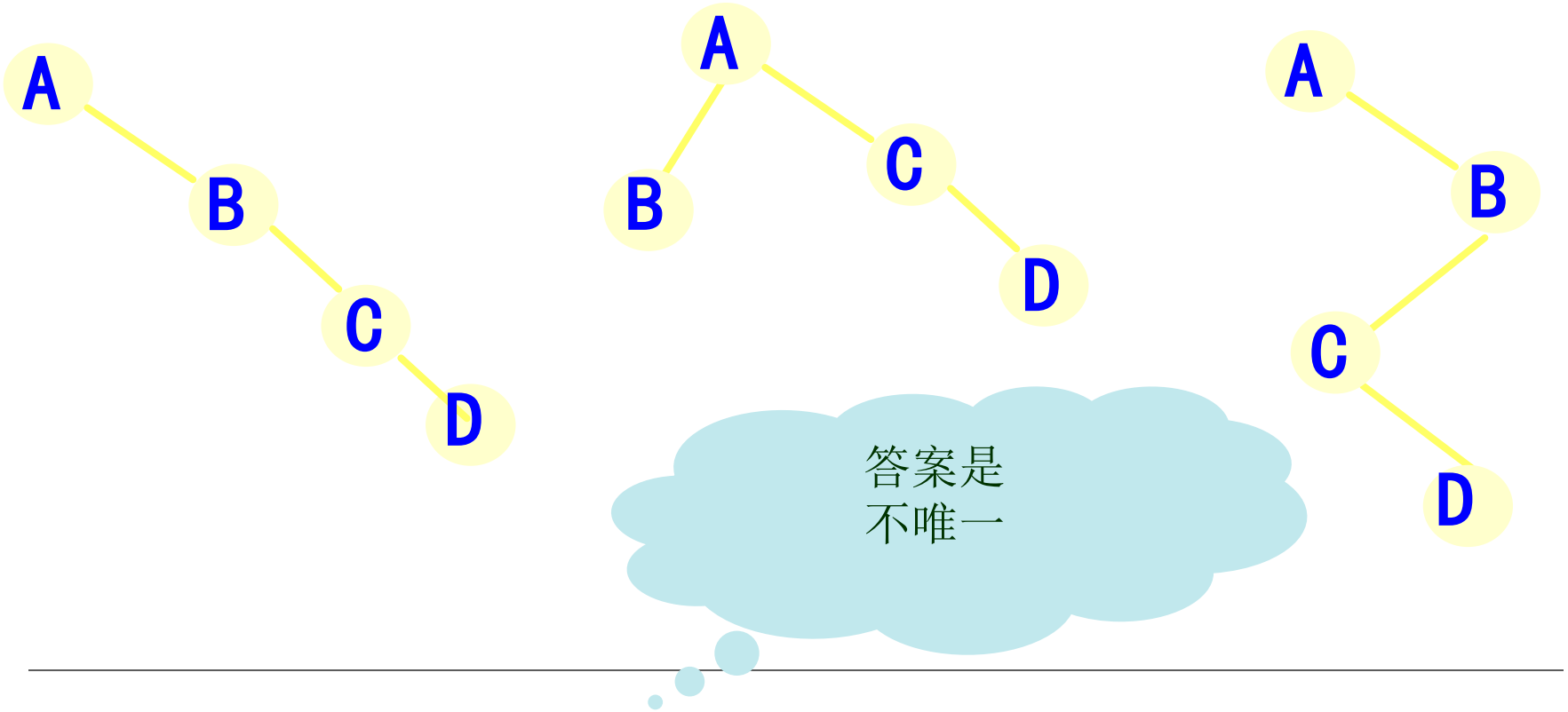
6.4.3 二叉树的相似与等价

6.4.2 二叉树的遍历与存储结构的应用

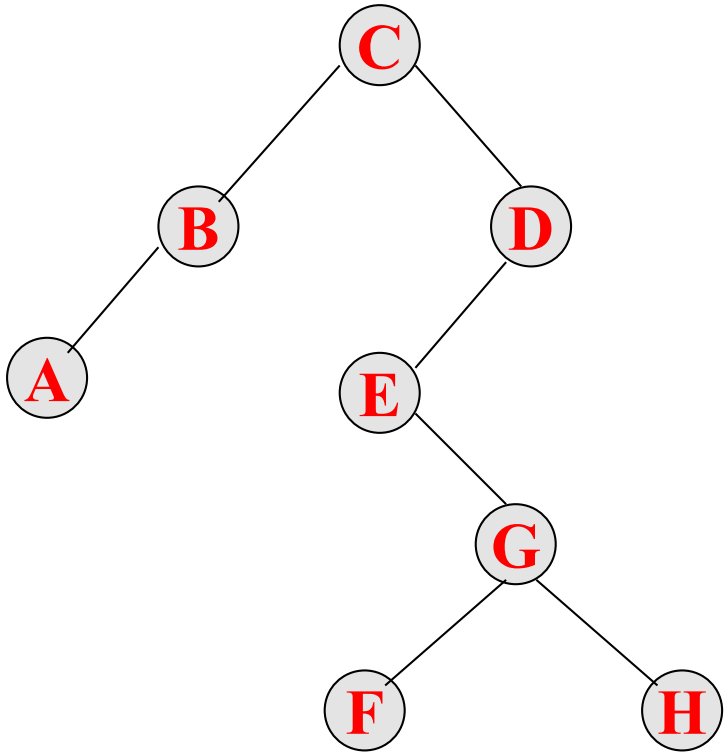
□二叉树的遍历与存储结构之间的转化

问题：给定一个遍历序列，能否唯一确定一棵二叉树？

例如：先序序列为ABCD，其二叉树的结构是什么？



例：二叉树中序序列为：ABCEFGHD，
后序序列为：ABFHGEDC。
请画出此二叉树。



已知：

①已知先序和中序？√

②已知中序和后序？√

③已知先序和后序？×

能否唯一还原二叉树？

6.4.2 二叉树的遍历与存储结构的应用

□ 构造二叉树

关键（1）确定二叉树的根结点；

（2）结点的左右次序。

给定某两种遍历序列能否唯一确定一棵二叉树？

给定中序和后序 ⟶ 唯一确定一棵二叉树

给定中序和前序 ⟶ 唯一确定一棵二叉树

给定先序和后序不能唯一确定一棵二叉树

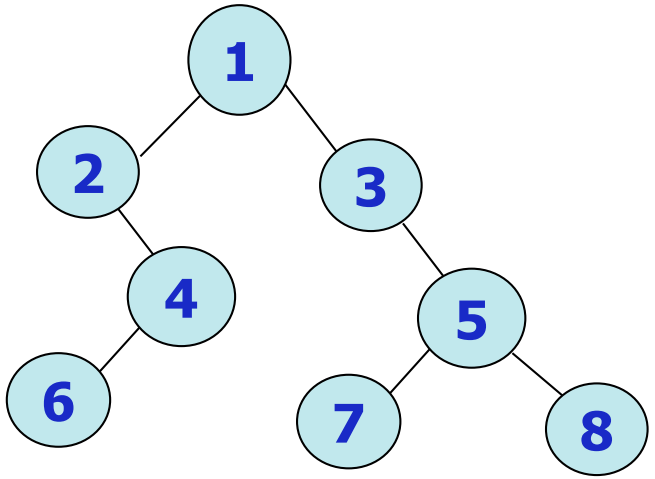
6.4.2 二叉树的遍历与存储结构的应用

□ 构造二叉树

例 给定二叉树先序和中序遍历序列，如何构造二叉树？

先序：1 2 4 6 3 5 7 8

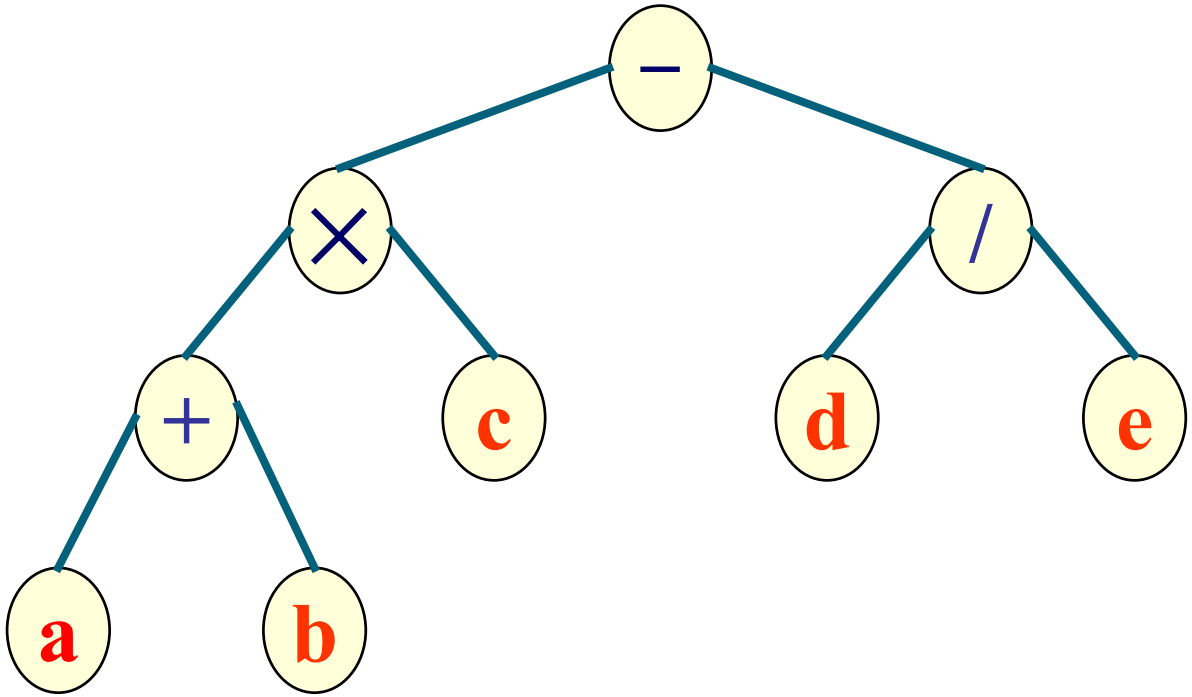
中序：2 6 4 1 3 7 5 8



6.5.3线索二叉树的遍历

应用举例

按给定的表达式建立相应的二叉树



对于二元运算符：

- 左右子树不空；
- 操作数为叶子结点；
- 运算符为分支结点；

6.5.3 线索二叉树的遍历

应用举例 **按给定后缀的表达式建立相应的二叉树**

a b + c * d e / -

后缀式的运算规则为

- 运算符在式中出现的顺序恰为表达式的运算顺序；
- 每个运算符和在它之前出现且紧靠它的两个操作数构成一个最小表达式。
- 操作数的顺序不变。

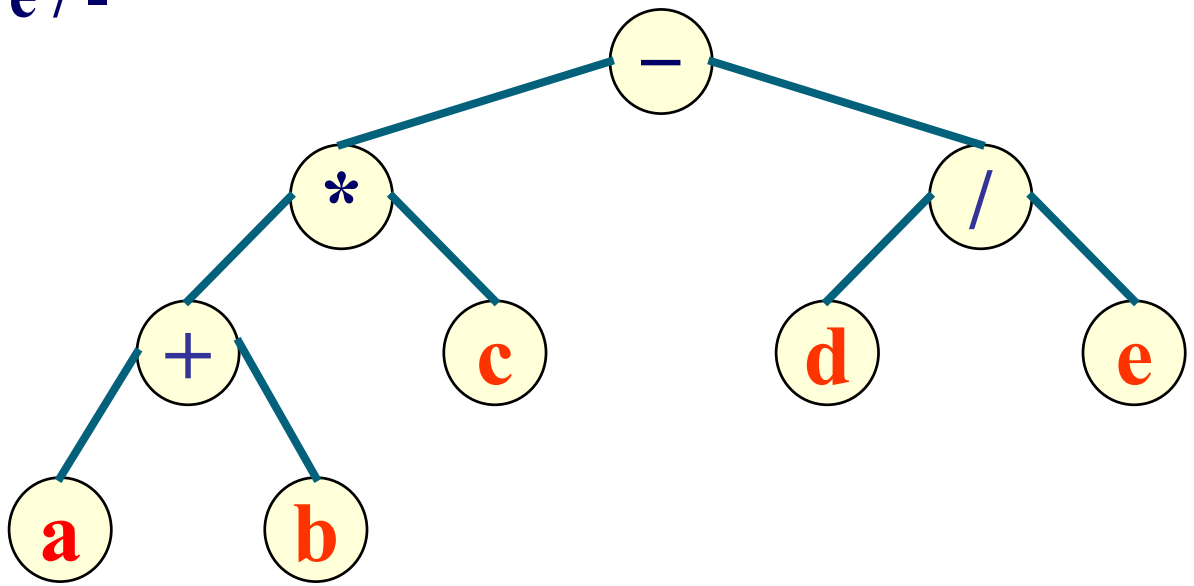
6.5.3 线索二叉树的遍历

应用举例

按给定后缀的表达式建立相应的二叉树

方法：从左到右扫描后缀表达式，遇到操作符则对前面的操作数建立二叉树，以此类推。

例如：后缀表达式a b + c * d e / -



6.5.3 线索二叉树的遍历

应用举例 **按给定前缀的表达式建立相应的二叉树**

已知表达式的前缀表示式 －×+ a b c / d e

前缀式的运算规则为：

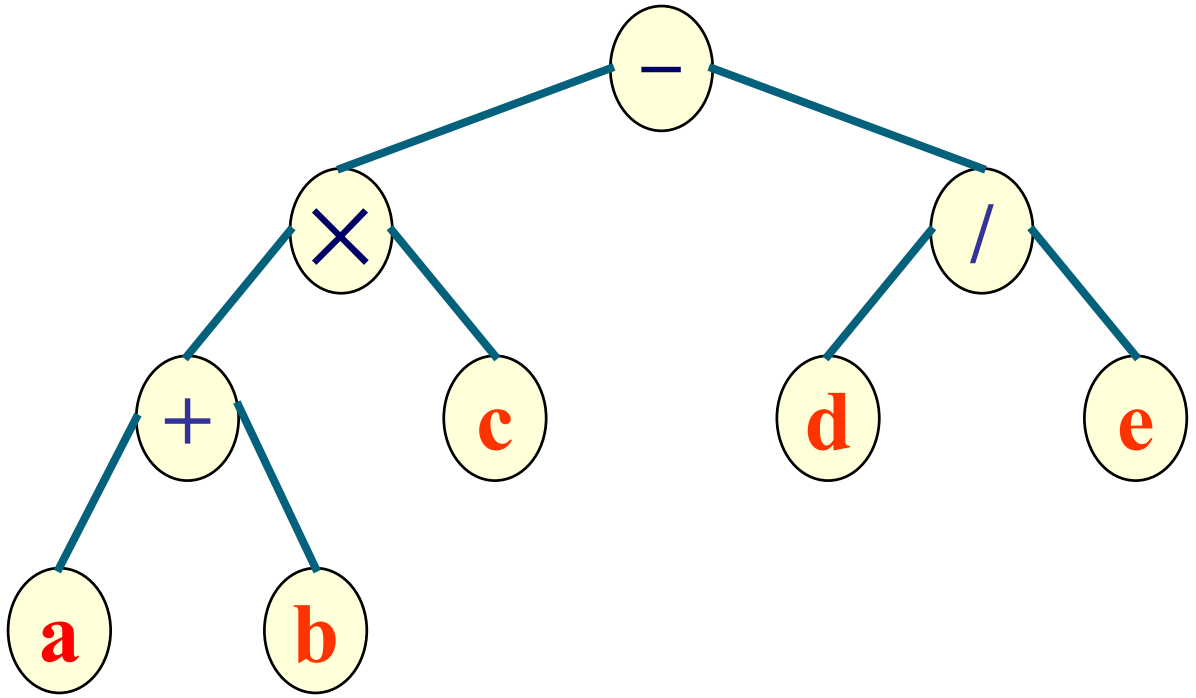
- 连续出现的两个操作数和在它们之前且紧靠它们的运算符构成一个最小表达式；
- 前缀式唯一确定了运算顺序；

6.5.3 线索二叉树的遍历

应用举例

按给定前缀的表达式建立相应的二叉树

已知表达式的前缀表示式 －×+ a b c / d e



6.4 遍历的应用

6.4.1 遍历的基本应用

6.4.2 二叉树的遍历与存储结构的应用

6.4.3 二叉树的相似与等价

6.4.2 二叉树的相似与等价

- 二叉树的相似与等价的含义

两株二叉树具有相同结构指：

“形状”相同

（1）它们都是空的；

（2）它们都是非空的，且左右子树分别具有相同结构。

- 定义具有相同结构的二叉树为**相似**二叉树。
- 相似且相应结点包含相同信息的二叉树称为**等价**二叉树。

6.4.2 二叉树的相似与等价

- 判断两株二叉树是否等价

```
int   EQUAL(BinTree t1 , BinTree t2 )
{  int  x ;
   x = 0 ;
   if ( ISEMPTY(t1) && ISEMPTY(t2) )//二叉树空
      x = 1 ;
   else if ( !ISEMPTY( t1 ) && ! ISEMPTY( t2) ) //二叉树不空
      if (  DATA( t1 ) == DATA( t2 )  )
         if ( EQUAL( LCHILD( t1 ) , LCHILD( t2 ) ) )
            x= EQUAL( RCHILD( t1) , RCHILD( t2) )
   return( x ) ;
} /* EQUAL */
```

6.4.2 二叉树的相似与等价

- 二叉树的复制

```
BinTree  COPY(BinTree oldtree )
{
   BinTree   temp ;
   if ( oldtree != NULL )
     {
        temp = new  Node ;
        temp -> lch = COPY( oldtree->lch ) ;
        temp -> rch = COPY( oldtree->rch ) ;
        temp -> data = oldtree->data ;
        return ( temp );
      }
   return ( NULL ) ;
}
```

6.4.2 二叉树的遍历与存储结构的应用

结论：

- “遍历”是二叉树各种操作的基础；
- 可以在遍历过程中对结点进行各种操作，
  - 对于一棵已知树可求结点的双亲；
  - 求结点的孩子结点；
  - 判定结点所在层次；
  - 树的深度；
  - 生成二叉树
  - 二叉树的复制

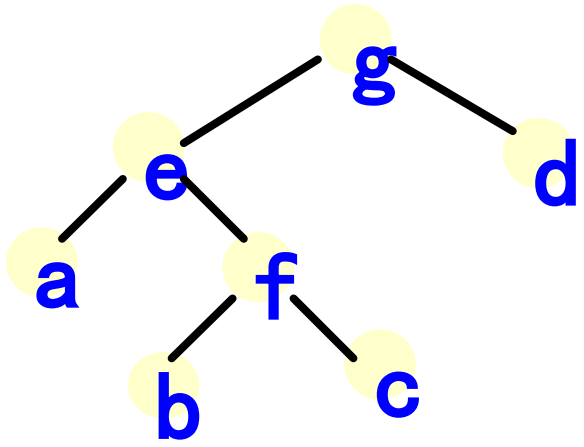
6.5 线索二叉树

6.5.1 线索二叉树的表示

6.5.2 二叉树的线索化

6.5.3 线索二叉树的遍历

二叉树的中序遍历序列



中序 a,e,b,f,c,g,d

1.能否从一个指定结点开始中序遍历?

2.如何找到指定结点p在中序遍历序列中的前驱?

3.如何找到p的中序后继?

二叉链表存储缺点：找前驱、后继很不方便；遍历 操作必须从根开始

n个结点的二叉树，有 n+1个空链域

6.5 线索二叉树

6.5.1 线索二叉树的表示

6.5.2 二叉树的线索化

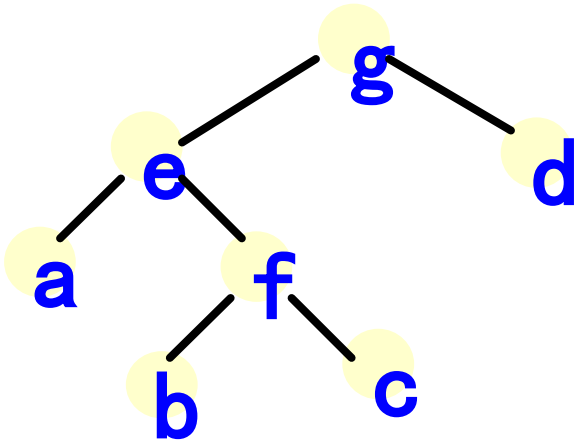
6.5.3 线索二叉树的遍历

6.5.1 线索二叉树的表示

• 线索二叉树的概念

➢考虑利用n+1个空链域来存放遍历后结点的前驱和后继信息，这就是线索二叉树构成的思想。

➢采用既可以指示其前驱又可以指示后继的双向链接结构的二叉树被称为线索二叉树。



中序 a,e,b,f,c,g,d

先序 g,e,a,f,b,c,d

后序 a,b,c,f,e,d,g

6.5.1 线索二叉树的表示

• 线索链表的结点结构

lch	ltag	data	rtag	rch

```
typedef struct BTnode
  {datatype data;
   struct BTnode *lch,*rch;
   int ltag,rtag;
  }BTnode, *threadbithptr;
```

其中：ltag，rtag为两个标志域

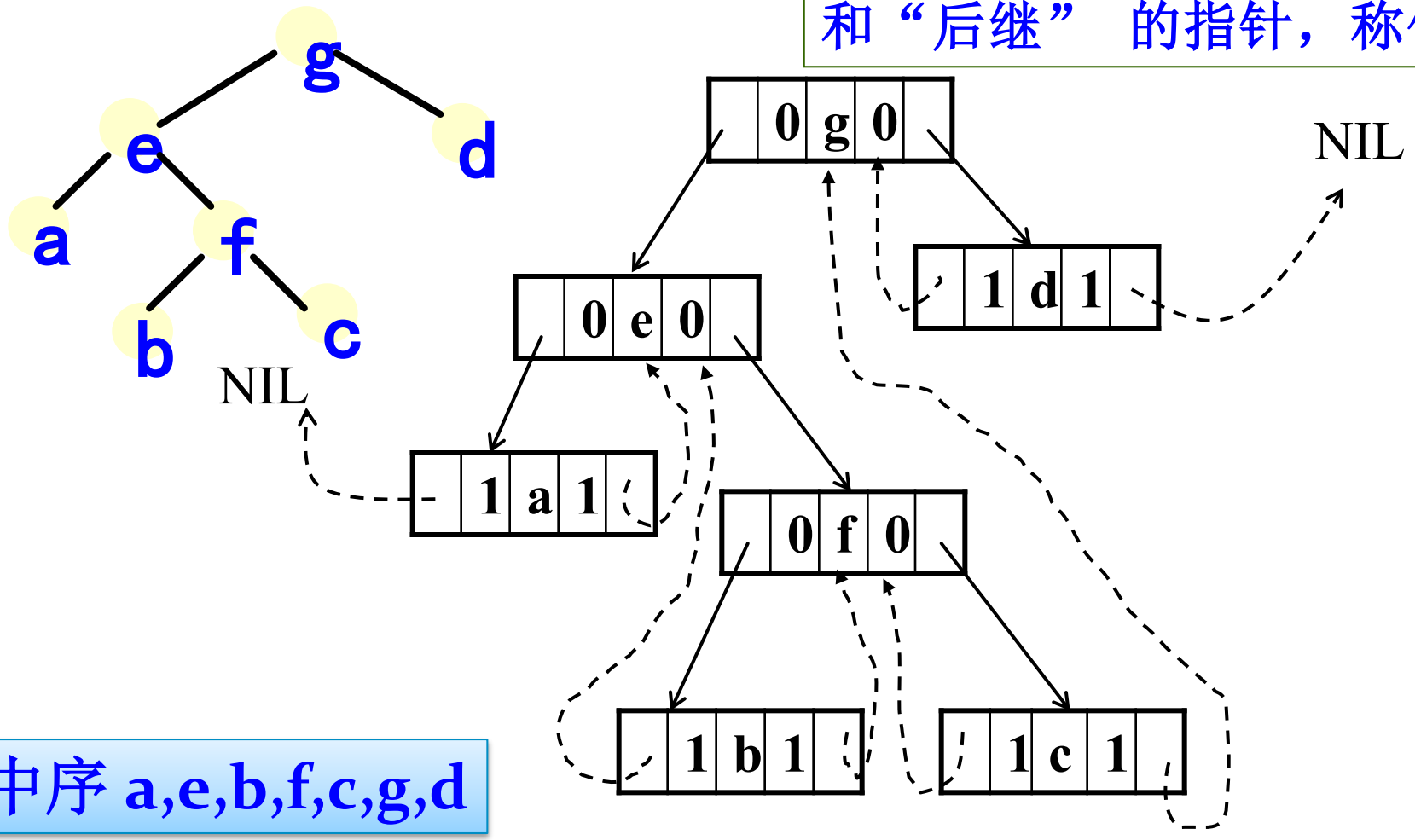
ltag= { 0 lch域指示结点的左孩子
      { 1 lch域指示结点的前驱

rtag= { 0 rch域指示结点的右孩子
      { 1 rch域指示结点的后继

6.5.1 线索二叉树的表示

- 二叉树存储结构--线索链表

遍历指向该线性序列中的“前驱”和“后继” 的指针，称作“线索”。



中序 a,e,b,f,c,g,d

线索链表

6.5.1 线索二叉树的表示

• 线索二叉树的相关概念

➢**线索链表：** 以上述结点结构构成的二叉链表作为二叉树的存储结构，叫线索链表。

➢**线索：** 指向前驱和后继的指针。

➢**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。

6.5 线索二叉树

6.5.1 线索二叉树的表示

6.5.2 二叉树的线索化

6.5.3 线索二叉树的遍历

6.5.3 线索二叉树的遍历

线索化的实质：

是将二叉链表中的空指针改为指向前驱或后继的线索，而前驱或后继信息只有在遍历时才能得到，因此线索化的过程即为在遍历过程中修改空指针的过程。

6.5.3 线索二叉树的遍历

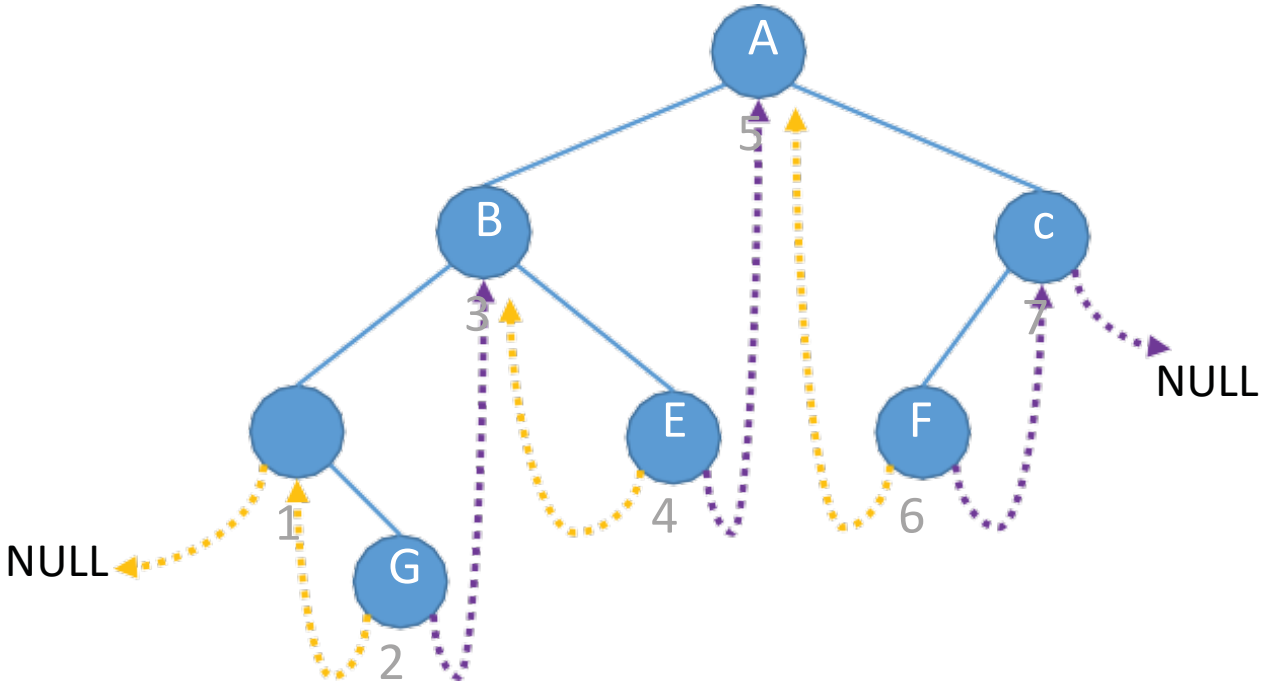
- 中序线索化二叉树

如何建立线索链表?

1. 在中序遍历过程中修改结点的左、右指针域
2. 保存当前访问结点的“前驱”和“后继”信息。
3. 遍历过程中，附设指针pre，pre指向刚访问过的结点。P指向当前访问结点。即pre是p的前驱

6.5.3 线索二叉树的遍历

- 中序线索化二叉树

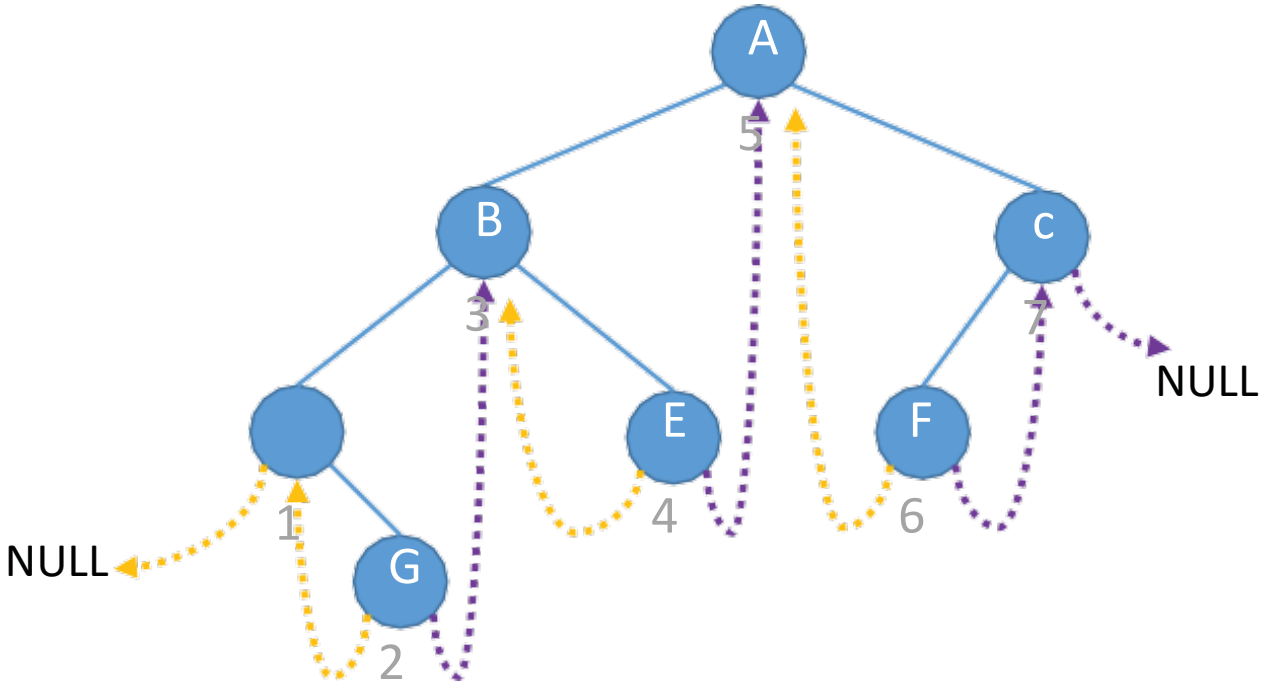



```
//全局变量 pre，指向当前访问结点的前驱
ThreadNode *pre=NULL;
```

```
//中序遍历二叉树，一边遍历一边线索化
void InThread(ThreadTree T){
    if(T!=NULL){
        InThread(T->lchild);     //中序遍历左子树
        visit(T);                //访问根节点
        InThread(T->rchild);     //中序遍历右子树
    }
}
void visit(ThreadNode *q) {
    if(q->lchild==NULL){//左子树为空，建立前驱线索
        q->lchild=pre;
        q->ltag=1;
    }
    if(pre!=NULL&&pre->rchild==NULL){
        pre->rchild=q;  //建立前驱结点的后继线索
        pre->rtag=1;
    }
    pre=q;
}
```

6.5.3 线索二叉树的遍历

- 中序线索化二叉树

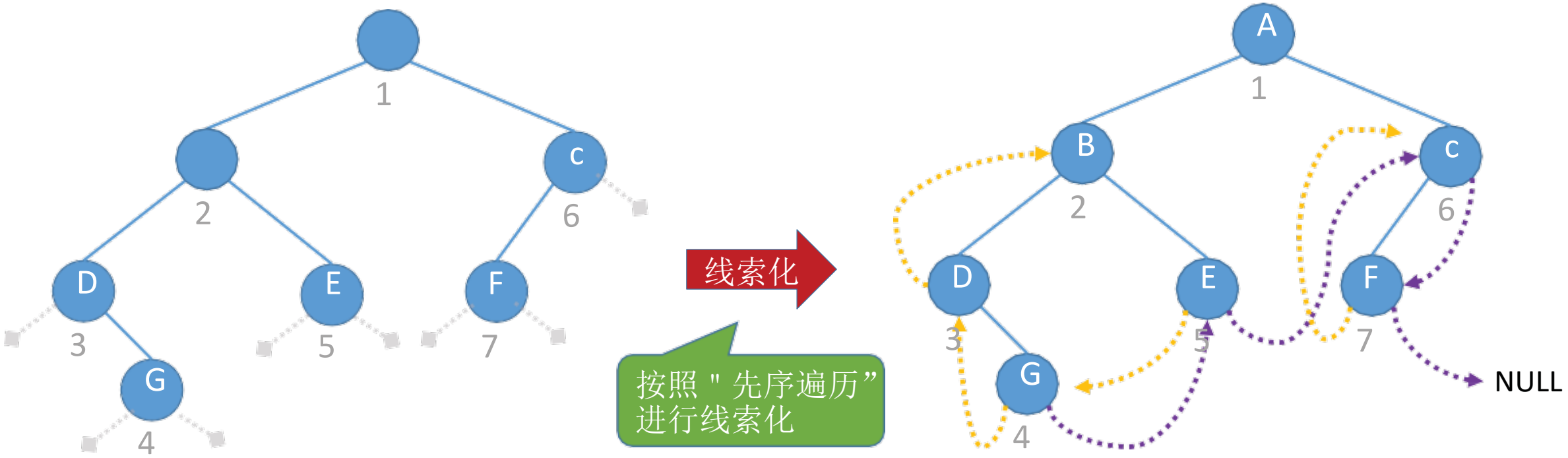



```
//全局变量 pre，指向当前访问结点的前驱
ThreadNode *pre=NULL;
```

```
//中序遍历二叉树，一边遍历一边线索化
void InThread(ThreadTree T){
    if(T!=NULL){
        InThread(T->lchild);     //中序遍历左子树
        visit(T);                //访问根节点
        InThread(T->rchild);     //中序遍历右子树
    }
}

//中序线索化二叉树T
void CreateInThread(ThreadTree T){
    pre=NULL;                    //pre初始为NULL
    if(T!=NULL){                 //非空二叉树才能线索化
        InThread(T);             //中序线索化二叉树
        if (pre->rchild==NULL)
            pre->rtag=1;         //处理遍历的最后一个结点
    }
}
```

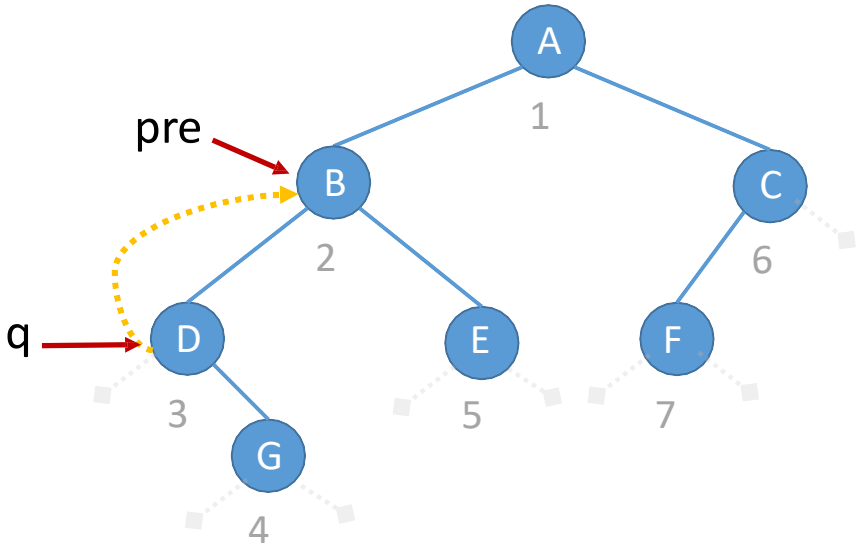
6.5.3 线索二叉树的遍历



先序遍历序列：ABDGECF

6.5.3 线索二叉树的遍历

- 先序线索化二叉树

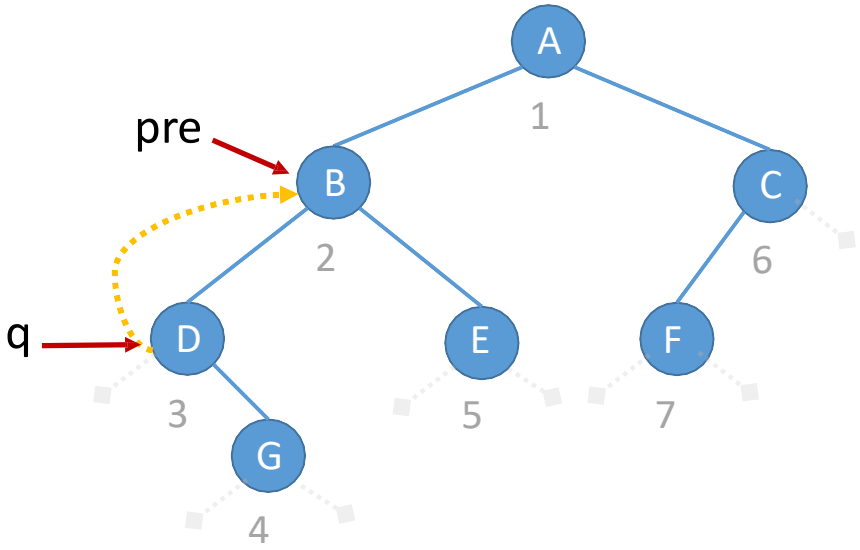



```
//全局变量 pre，指向当前访问结点的前驱
ThreadNode *pre=NULL;
```

```
//先序遍历二叉树，一边遍历一边线索化
void PreThread(ThreadTree T){
    if(T!=NULL){
        visit(T);          //先处理根节点
        PreThread(T->lchild);
        PreThread(T->rchild);
    }
}
void visit(ThreadNode *q) {
    if(q->lchild==NULL){//左子树为空，建立前驱线索
        q->lchild=pre;
        q->ltag=1;
    }
    if(pre!=NULL&&pre->rchild==NULL){
        pre->rchild=q;  //建立前驱结点的后继线索
        pre->rtag=1;
    }
    pre=q;
}
```

6.5.3 线索二叉树的遍历

- 先序线索化二叉树



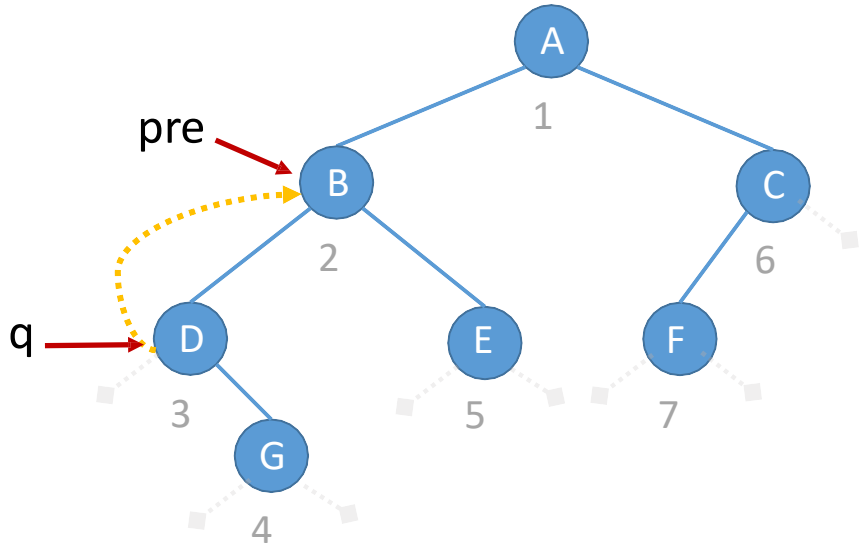

```
//全局变量 pre，指向当前访问结点的前驱
ThreadNode *pre=NULL;
```

```
//先序遍历二叉树，一边遍历一边线索化
void PreThread(ThreadTree T){
    if(T!=NULL){
        visit(T);          //先处理根节点
        PreThread(T->lchild);
        PreThread(T->rchild);
    }
}
```

```
//先序遍历二叉树，一边遍历一边线索化
void PreThread(ThreadTree T){
    if(T!=NULL){
        visit(T);          //先处理根节点
        if (T->ltag==0) //lchild不是前驱线索
            PreThread(T->lchild);
        PreThread(T->rchild);
    }
}
```

6.5.3 线索二叉树的遍历

• 先序线索化二叉树




```
//全局变量 pre，指向当前访问结点的前驱
ThreadNode *pre=NULL;
```

```
//先序遍历二叉树，一边遍历一边线索化
void PreThread(ThreadTree T){
    if(T!=NULL){
        visit(T);           //先处理根节点
        if (T->ltag==0) //lchild不是前驱线索
            PreThread(T->lchild);
        PreThread(T->rchild);
    }
}
//先序线索化二叉树T
void CreatePreThread(ThreadTree T){
    pre=NULL;                   //pre初始为NULL
    if(T!=NULL){                //非空二叉树才能线索化
        PreThread(T);           //先序线索化二叉树
        if (pre->rchild==NULL)
            pre->rtag=1;        //处理遍历的最后一个结点
    }
}
```

6.5 线索二叉树

- 小结：
  - 二叉树的线索化核心：
    - 中序、先序、后序遍历算法的改造
    - 需要一个指针pre记录当前访问结点的前驱结点
  - 易错点：
    - 最后一个结点的处理
    - 先序线索化中注意转圈问题

6.5 线索二叉树

6.5.1 线索二叉树的表示

6.5.2 二叉树的线索化

6.5.3 线索二叉树的遍历

6.5.2 二叉树的线索化

- 查找---在线索树中找结点---中序后继

在中序线索树中查找中序后继的方法

（1）当结点没有右子树时，即：

当p->rtag =1 时，p->rch 既为所求后继结点（线索）。

（2）当结点有右子树时，即：

当p->rtag = 0 时，p的后继结点为p的右子树的最左下结点。

6.5.2 二叉树的线索化

• 查找---在线索树中找结点---中序后继

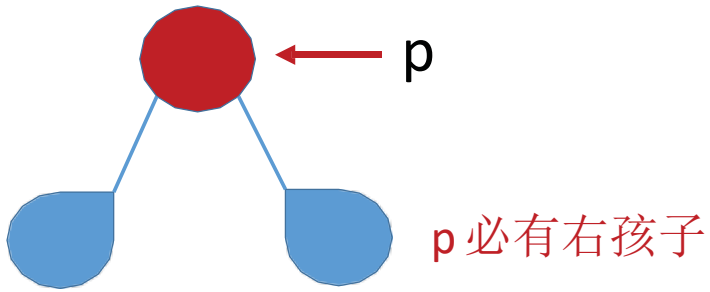
在中序线索树中查找中序后继的方法

（1）当结点没有右子树时，即：

当p->rtag =1 时，p->rch 既为所求后继结点（线索）。

（2）当结点有右子树时，即：

当p->rtag = 0 时，p的后继结点为p的右子树的最左下结点。




```
//找到以P为根的子树中，第一个被中序遍历的结点
ThreadNode *Firstnode(ThreadNode *p){
    //循环找到最左下结点(不一定是叶结点)
    while(p->ltag==0) p=p->lchild;
    return p;
}
```

6.5.2 二叉树的线索化

• 查找---在线索树中找结点---中序后继

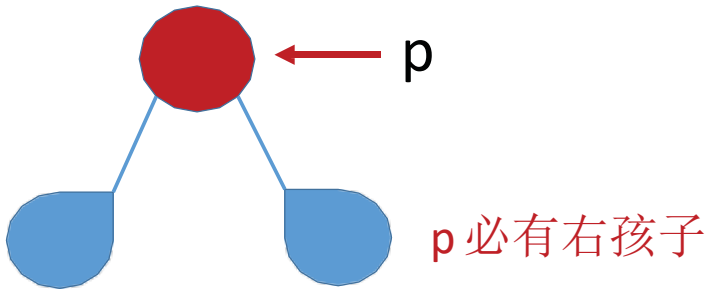
在中序线索树中查找中序后继的方法

（1）当结点没有右子树时，即：

当p->rtag =1 时，p->rch 既为所求后继结点（线索）。

（2）当结点有右子树时，即：

当p->rtag = 0 时，p的后继结点为p的右子树的最左下结点。




```
//在中序线索二叉树中找到结点p的后继结点
ThreadNode *Nextnode(ThreadNode *p){
    //右子树中最左下结点
    if(p->rtag==0) return Firstnode(p->rchild);
    else return p->rchild;    //rtag==1直接返回后继线索
}
```

6.5.2 二叉树的线索化

- 查找--在线索二叉树树中找结点--中序前驱

在线索二叉树中查找中序前驱的方法

（1）当结点没有左子树时，即：

当p->ltag =1 时，p->lch 既为所求前驱结点（线索）。

（2）当结点有左子树时，即：

当p->ltag = 0 时，p的前驱结点为p的左子树的最右下结点。

6.5.2 二叉树的线索化

• 查找--在线索二叉树树中找结点--中序前驱

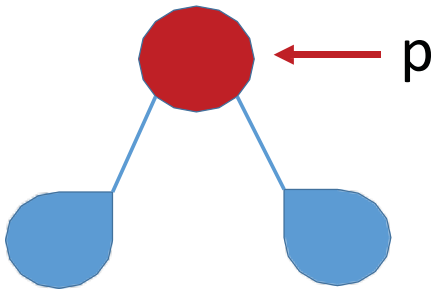
在线索二叉树中查找中序前驱的方法

（1）当结点没有左子树时，即：

当p->ltag =1 时，p->lch 既为所求前驱结点（线索）。

（2）当结点有左子树时，即：

当p->ltag = 0 时，p的前驱结点为p的左子树的最右下结点。



p 必有左孩子

```
//找到以P为根的子树中，最后一个被中序遍历的结点
ThreadNode *Lastnode(ThreadNode *p){
    //循环找到最右下结点(不一定是叶结点)
    while(p->rtag==0) p=p->rchild;
    return p;
}
```

6.5.2 二叉树的线索化

• 查找--在线索二叉树树中找结点--中序前驱

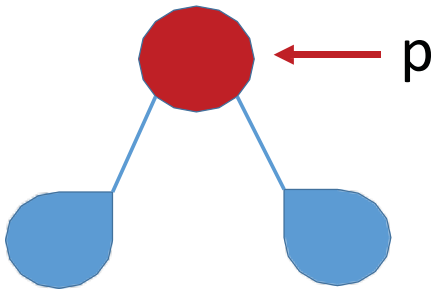
在线索二叉树中查找中序前驱的方法

（1）当结点没有左子树时，即：

当p->ltag =1 时，p->lch 既为所求前驱结点（线索）。

（2）当结点有左子树时，即：

当p->ltag = 0 时，p的前驱结点为p的左子树的最右下结点。



p 必有左孩子

```
//在中序线索二叉树中找到结点p的前驱结点
ThreadNode *Prenode(ThreadNode *p){
    //左子树中最右下结点
    if(p->ltag==0) return Lastnode(p->lchild);
    else return p->lchild;    //ltag==1直接返回前驱线索
}
```

6.5.2 二叉树的线索化

- 在中序线索二叉树树中进行中序遍历

```
//对中序线索二叉树进行中序遍历（利用线索实现的非递归算法）
void Inorder(ThreadNode *T){
    for(ThreadNode *p=Firstnode(T);p!=NULL; p=Nextnode(p))
        visit(p);
}

//对中序线索二叉树进行逆向中序遍历
void RevInorder(ThreadNode *T){
    for(ThreadNode *p=Lastnode(T);p!=NULL; p=Prenode(p))
        visit(p);
}
```

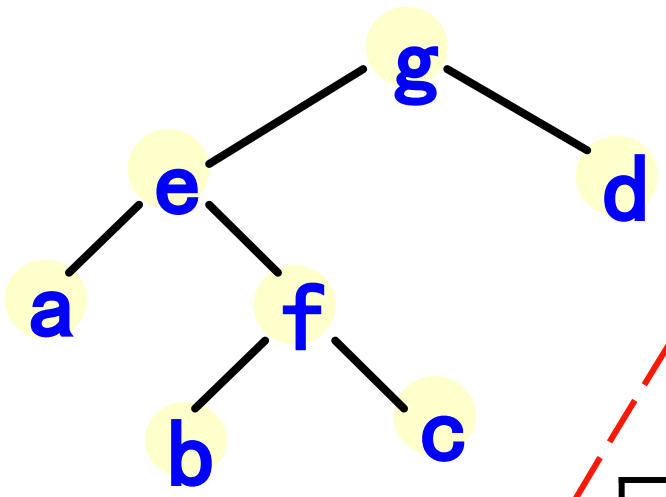
6.5.2 二叉树的线索化

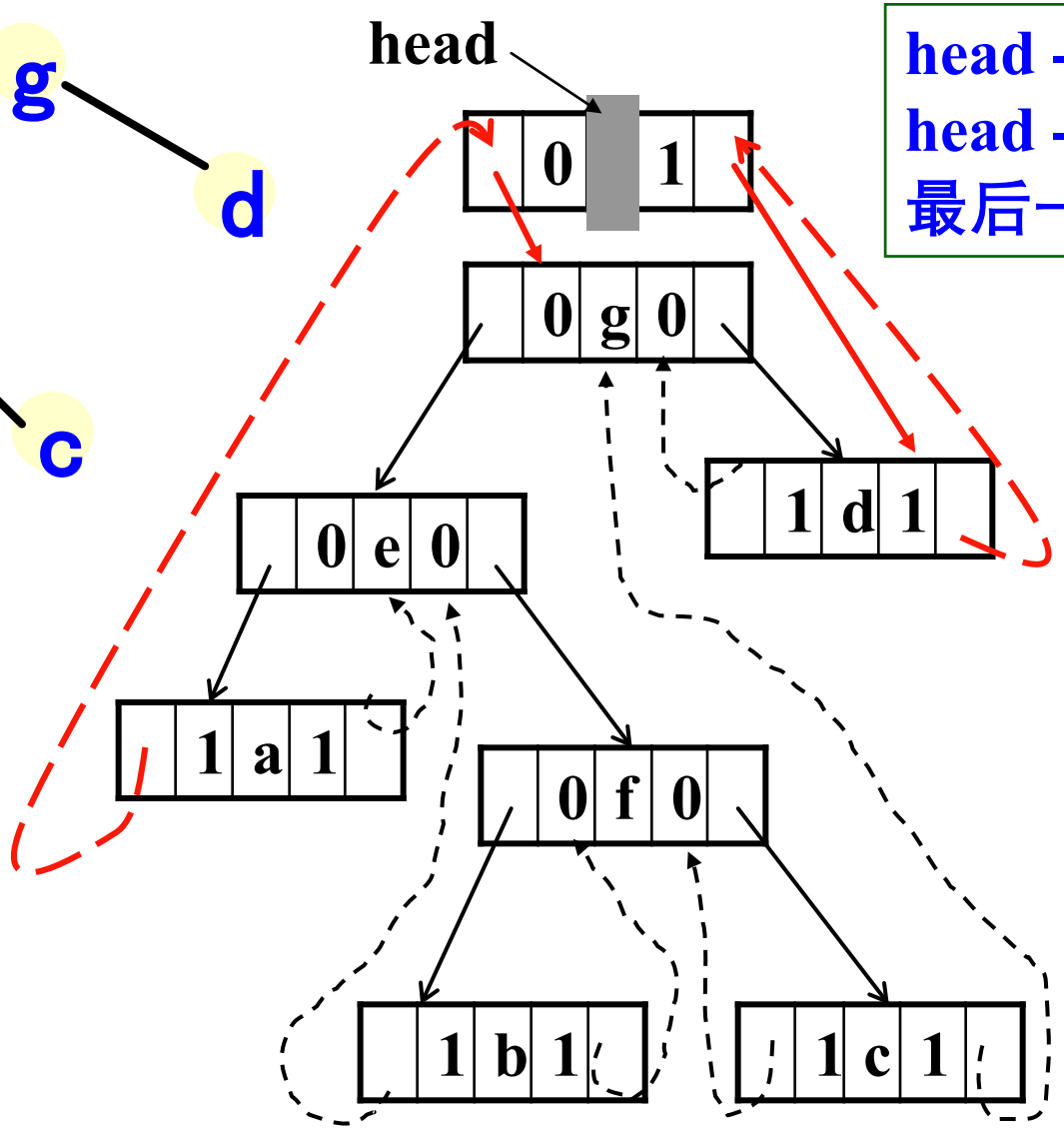
- 关于线索二叉链表的头结点
  - 跟线性链表类似，头结点是二叉树根结点的前驱
  - 如何设定头结点：

线索二叉树的头结点：

- head ->lch指向二叉树的根结点，head->ltag = 0 (非空树) / 1 (空树)
- head ->rch有两种设定方法：
  ① head ->rch = head， head->rtag = 0；
  ② head ->rch 指向遍历序列的最后一个结点， head->rtag = 1；
- 遍历序列的第一个结点的前驱线索和最后一个结点的后继线索均指向头结点。

- 关于线索二叉链表的头结点的设定方法





head ->lch指向根结点
head ->rch指向中序最后一个结点

中序遍历的第一个结点和最后一个结点都指向头结点

构成双向循环线索链表

中序序列 a,e,b,f,c,t,d

中序线索链表

6.5.3 线索二叉树的遍历

线索树的缺点：

- 先序线索二叉树无法找到任意结点的先序遍历前驱
  - 先序后继为左孩子（若无左孩子，则为右孩子）
- 后序线索二叉树无法找到任意结点的后序遍历后继
  - 后序前驱为右孩子（若无右孩子，则为左孩子）
- 在插入和删除时，除了修改指针外，还要相应地修改线索。

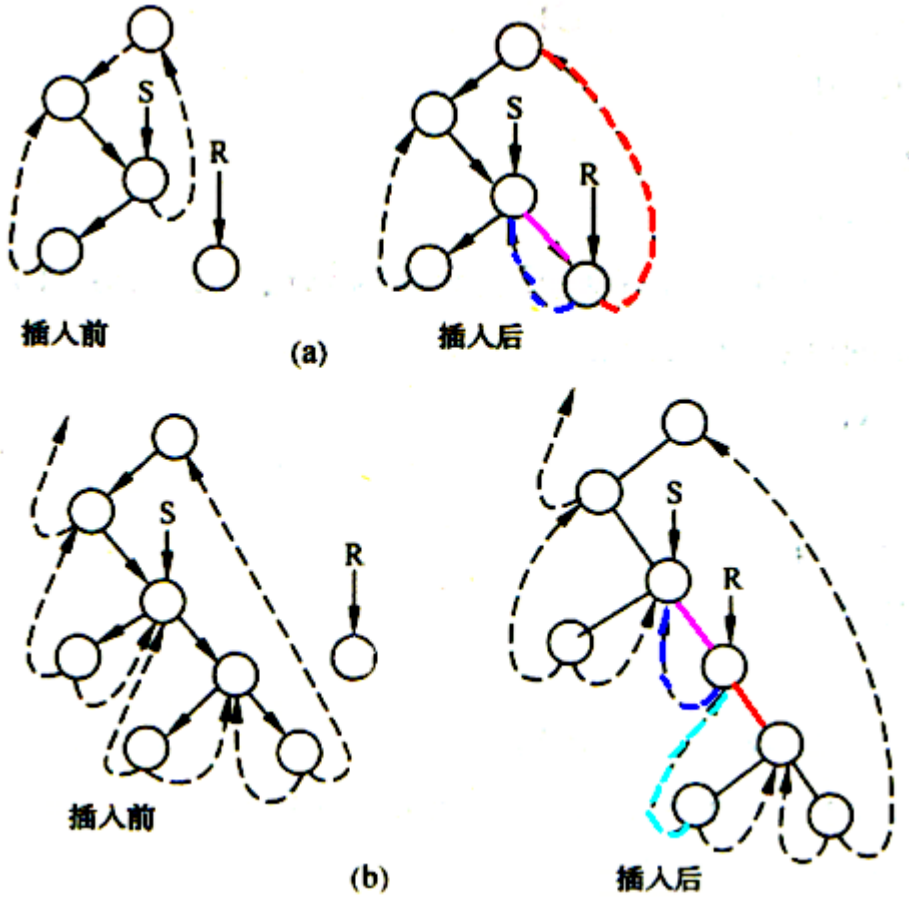
例：将结点 R 插入作为结点 S 的右孩子结点。

（1）若S的右子树为空，插入比较简单;

（2）若S的右子树非空，则 R 插入后 原来 S 的右子树作为 R 的右子树。

操作：

```
Void    RINSERT ( THTREE   S , THTREE   R )
   THTREE   W ;
    R->rchild = S->rchild ;
    R->rtag = S->rtag ;
    R->lchild = S ;
    R->ltag = 1 ;
    S->rchild = R ;
    S->rtag = 0 ;
    if ( R->rtag == 0 )
      {  w =Nextnode( R ) ;
          w->lchild = R ;  }
```



6.5.3 线索二叉树的遍历

思考题：

线索二叉树的左插入算法：将结点 R 插入作为结点 S 的左孩子