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：

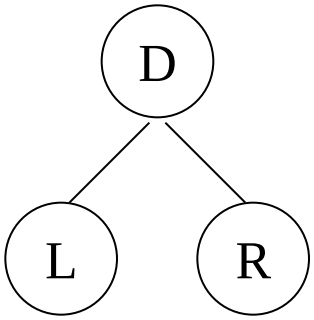
$$n_0 = n_2 + 1$$

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

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：左孩子 2i 右孩子2i+1

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：

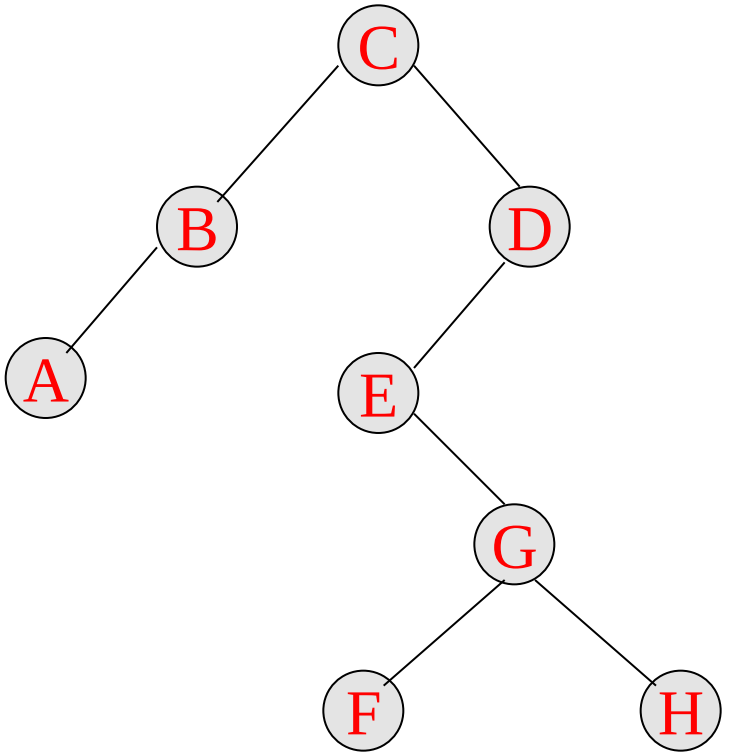
①先根顺序DLR， ②中根顺序LDR， ③后根顺序LRD

例题1：

二叉树中序序列为：**ABCEFGHD**，

后序序列为：**ABFHGEDC**

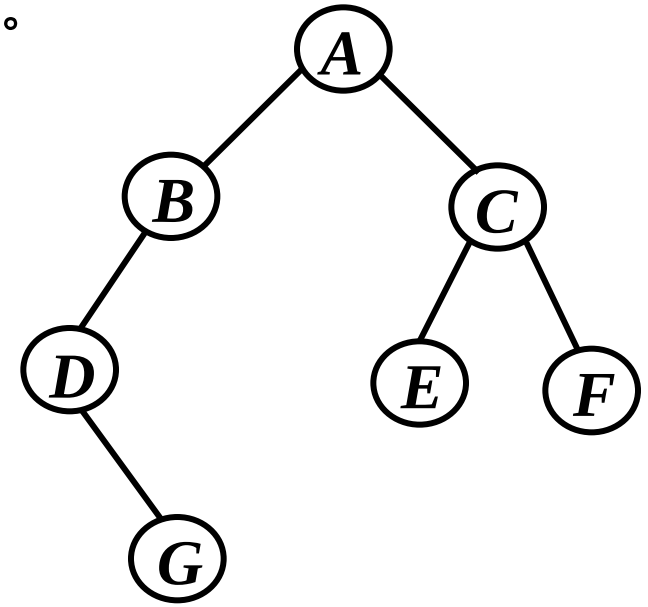
画出此二叉树



3.2 二叉树（Cont.）

二叉树遍历的定义

- 层序（次）遍历二叉树
  - 从二叉树的第一层（即根结点）开始，从上至下逐层遍历，在同一层中，则按从左到右的顺序对结点进行访问。
  - 所得到的线性序列分别称为层序序列。



- 层序遍历序列为： *A B C D E F G*

3.2 二叉树（Cont.）

二叉树的基本操作

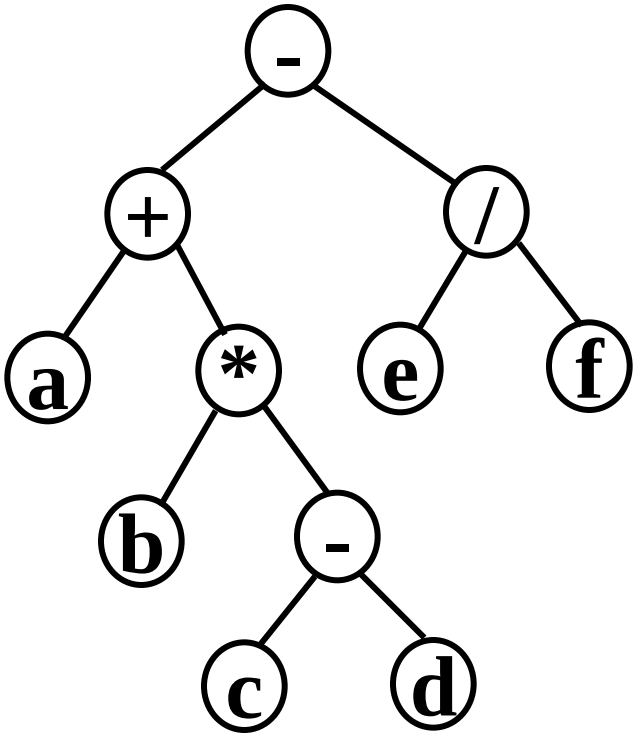
- ① **Empty (BT)：** 建立一株空的二元树。
- ② **IsEmpty (BT)：** 判断二元树是否为空,若是空则返回**TRUE**;否则返回**FALSE**。
- ③ **CreateBT (V, LT , RT)：** 建立一株新的二元树。这棵新二元树根结点的数据域为**V**,其作右子树分别为**LT**，**RT**。
- ④ **Lchild (BT)：** 返回二元树**BT**的左儿子。若无左儿子，则返回空。
- ⑤ **Rchild (BT)：** 返回二元树**BT**的右儿子。若无右儿子，则返回空。
- ⑥ **Data (BT)：** 返回二元树**BT**的根结点的数据域的值。

3.2 二叉树（Cont.）

利用二叉树的基本操作，写出前三种遍历算法的递归形式

- 先序遍历算法

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



a+b*(c-d)-e/f

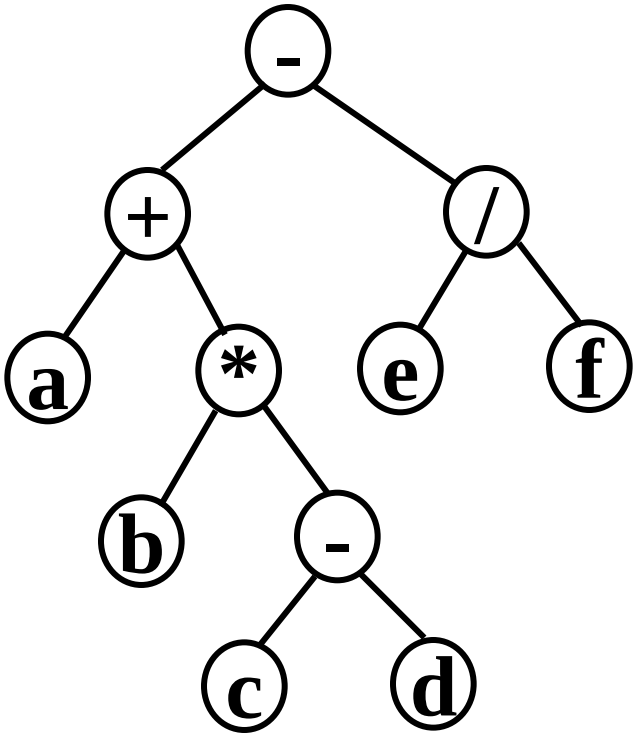
- 先序序列：- + a * b – c d / e f

3.2 二叉树（Cont.）

利用二叉树的基本操作，写出前三种遍历算法的递归形式

- 中序遍历算法

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



a+b*(c-d)-e/f

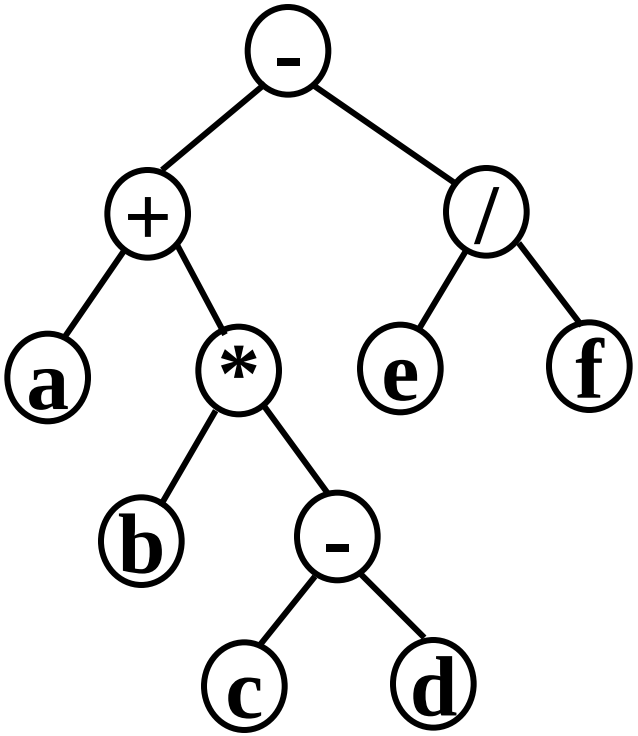
- 中序序列： a + b * c – d – e / f

3.2 二叉树（Cont.）

利用二叉树的基本操作，写出前三种遍历算法的递归形式

- 后序遍历算法

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



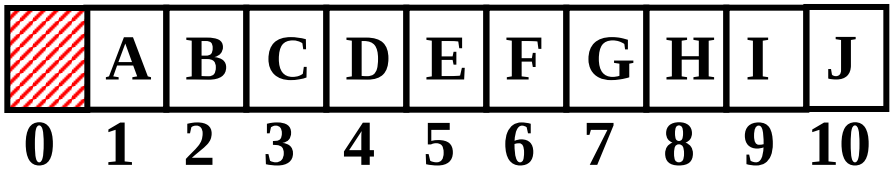
a+b*(c-d)-e/f

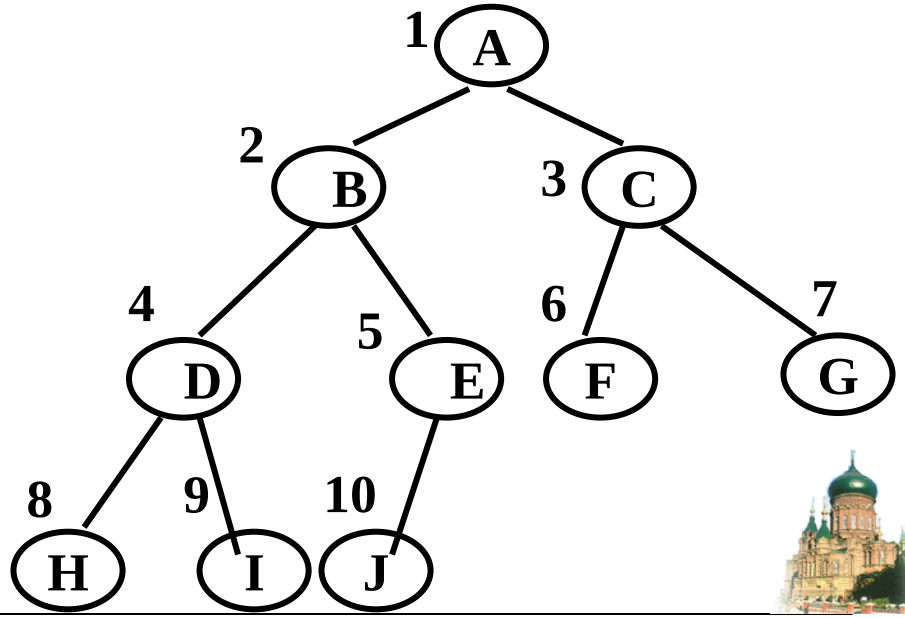
- 后序序列：a b c d - * + e f / -

3.2 二叉树（Cont.）

二叉树的存储结构

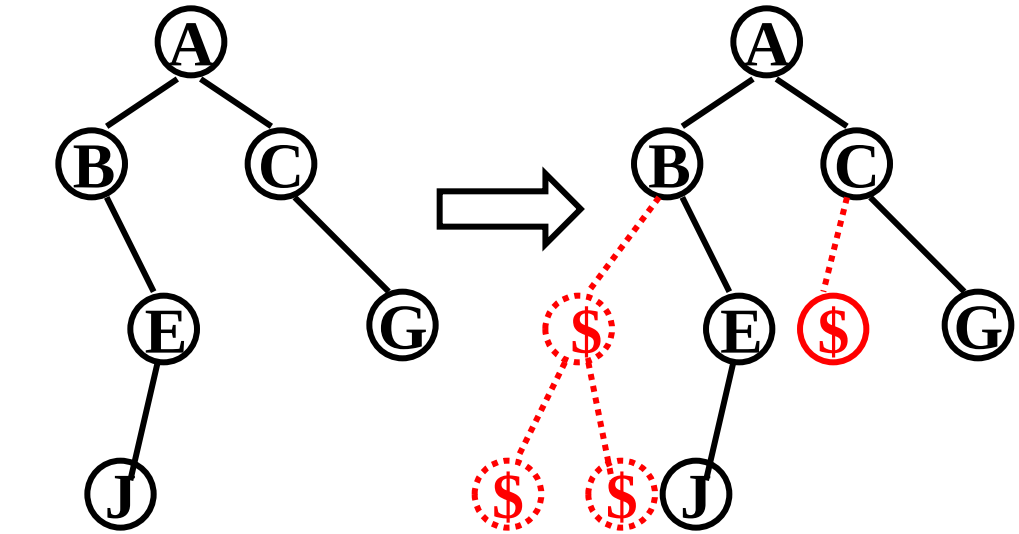
- 二叉树的顺序存储结构
  - 完全（满）二叉树---参见“完全二叉树的顺序存储结构”
    - 采用一维数组，按层序顺序依次存储二叉树的每一个结点。如下图所示：





3.2 二叉树（Cont.）

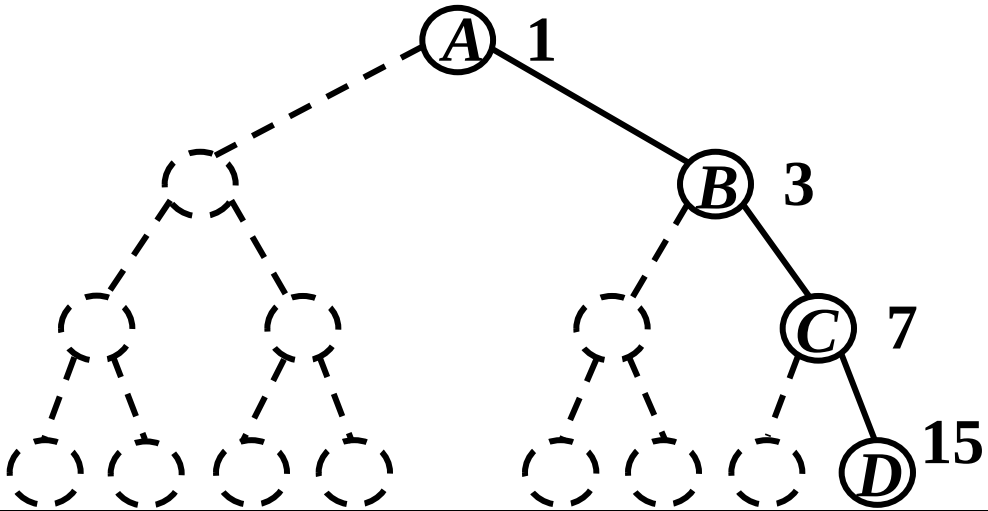
- 一般二叉树的顺序存储结构
  - 实现：按完全二叉树的结点层次编号，依次存放二叉树中的数据元素
  - 特点：结点间关系蕴含在其存储位置中



3.2 二叉树（Cont.）

一般二叉树的顺序存储结构

- 一棵斜树的顺序存储会怎样呢？
  - 高度为*k*的右斜树，*k*个结点需分配2^k-1个存储单元.
  - 需增加很多空结点，造成存储空间的浪费.



3.2 二叉树（Cont.）

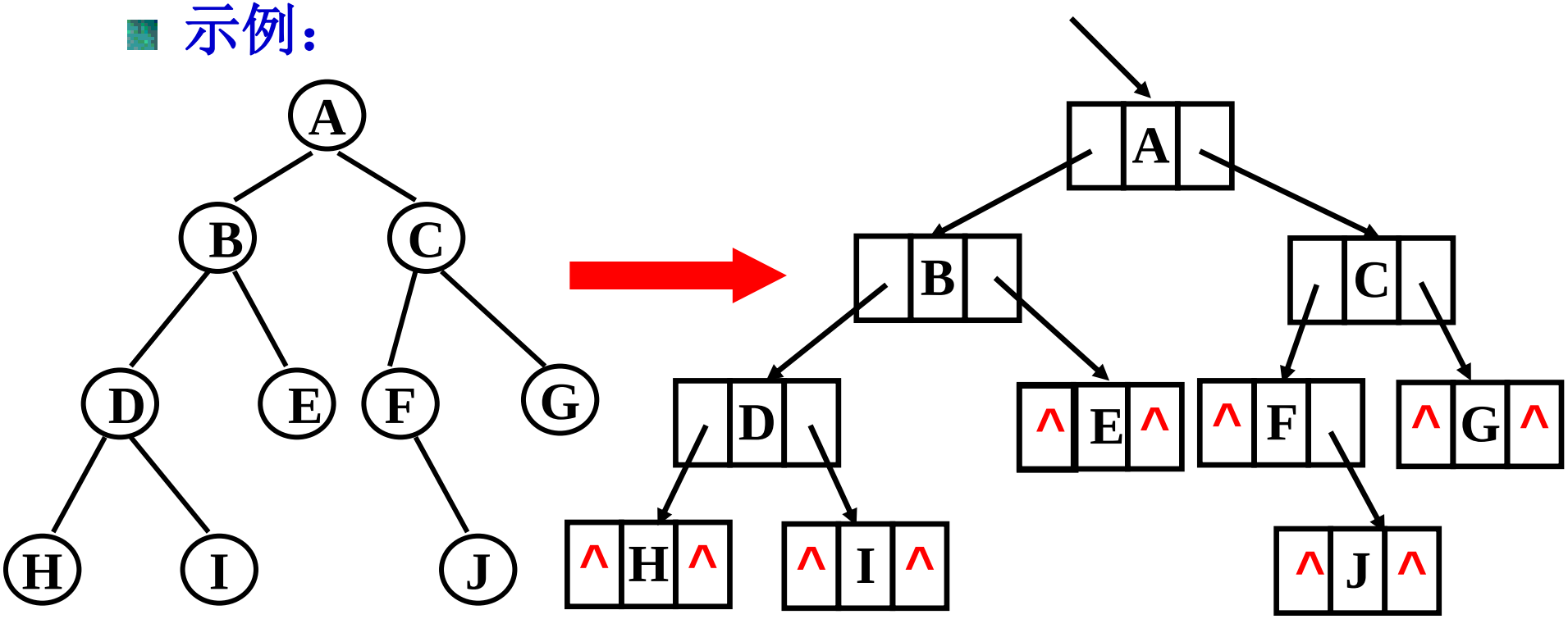
- 二叉树的左右链存储结构----动态二叉链表
  - 二叉树左右链表示：每个结点除了存放结点数据信息外，还设置两个指示左、右孩子的指针，如果该结点没有左或右孩子，则相应的指针为空。并用一个指向根结点的指针标识这个二叉树 。
  - 结点结构：

lchild	data	rchild

    - **data**：数据域，存放该结点的数据信息；
    - **lchild**：左指针域，存放指向左孩子的指针；
    - **rchild**：右指针域，存放指向右孩子的指针。

3.2 二叉树（Cont.）

- 二叉树的左右链存储结构----动态二叉链表
  - 示例：

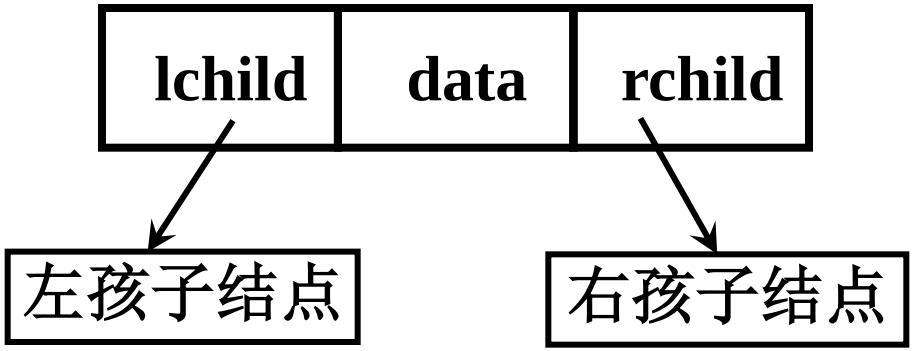


- 具有*n*个结点的二叉链表中，有多少个空指针？有多少指向孩子结点的指针？

3.2 二叉树（Cont.）

- 二叉树的左右链存储结构----动态二叉链表
  - 存储结构定义：

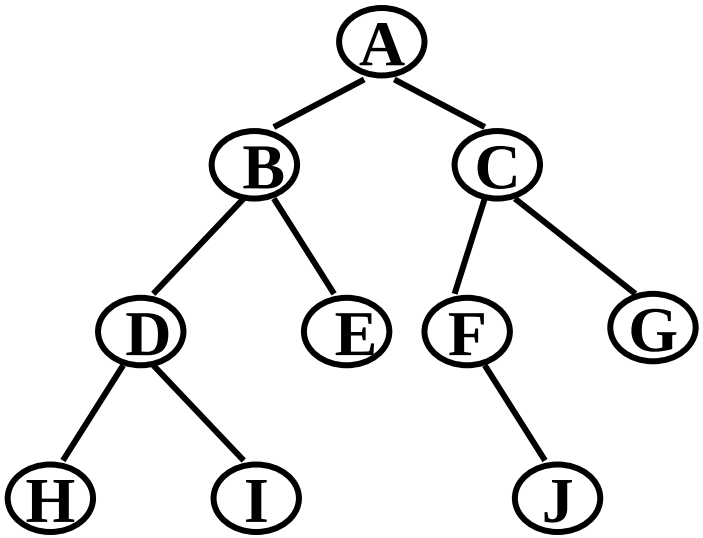
```
struct  node  {
    struct  node  *lchild ;
    struct  node  *rchild ;
          datatype  data ;
} ;
typedef  struct  node * Btree ;
```



3.2 二叉树（Cont.）

- 二叉树的左右链存储结构(二叉链表)的建立
  - 方法1：

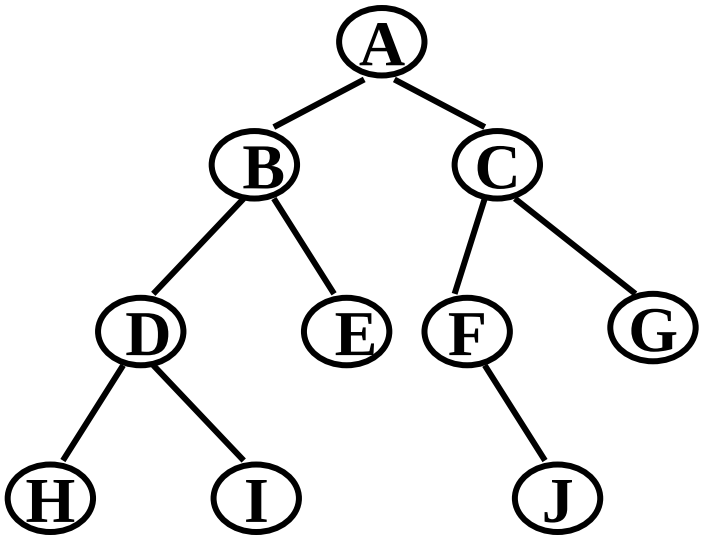
```
Btree CreateBT(datatype v, Btree ltree , Btree rtree )
{
    Btree  root ;
    root = new node ;
    root →data = v ;
    root →lchild = ltree ;
    root →rchild = rtree ;
    return  root ;
}
```



3.2 二叉树（Cont.）

- 方法2：按先序序列建立二叉树的左右链存储结构.
  - 如由图所示二叉树,输入:ABDH##I##E##CF#J##G##
    其中：#表示空

```
BTREE *CreateBT ()
{   BTREE *T;       char ch;
    scanf("%c",&ch);
    if(ch=='#')   T=NULL;
    else
    {          T=new node ;
              if(!T) exit(0);
               T->data=ch;
               T->lchild= CreateBT ();
               T->rchild= CreateBT ();
     }
    return(T);
}
```



3.2 二叉树（Cont.）

- 方法2：按先序序列建立二叉树的左右链存储结构.
  - 如由图所示二叉树,输入:ABDH##I##E##CF#J##G##
    其中：#表示空

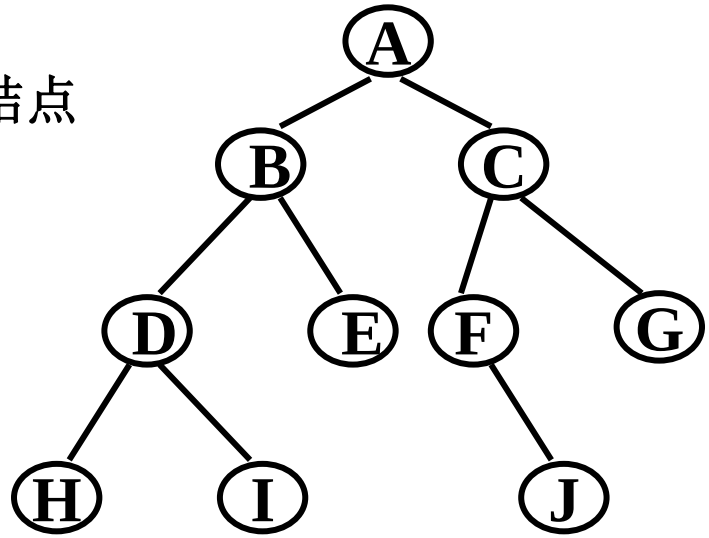
```
BTREE *CreateBT ()
{   BTREE *T;       char ch;
    scanf("%c",&ch);
    if(ch=='#')   T=NULL;
    else
    {          T=new node ;
              if(!T) exit(0);
               T->data=ch;
               T->lchild= CreateBT ();
               T->rchild= CreateBT ();
     }
    return(T);
}
```

```
void CreateBT(Btree & T)
{  cin >> ch ;
    if  ( ch == '#' ) T = NULL ;
    else{
          T →data = ch ;
          CreateBT ( T →lchild ) ;
          CreateBT ( T →rchild ) ;
    }
}
```

3.2 二叉树（Cont.）

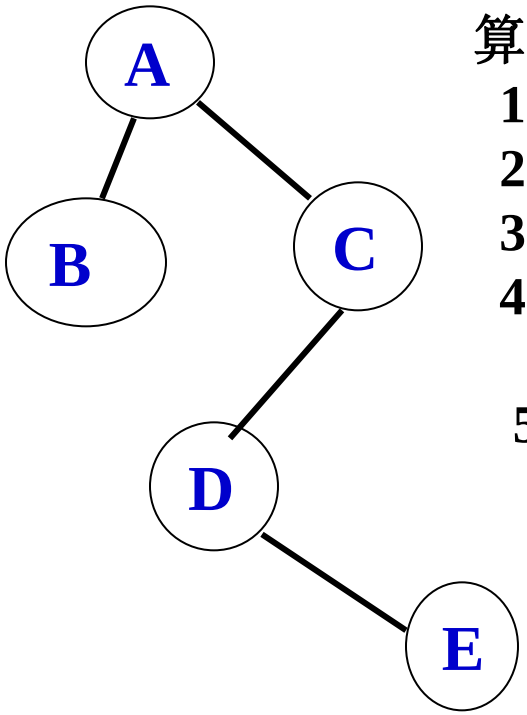
方法3：建立二叉树的左右链存储结构的非递归算法.

```
struct node *s[max];  // 辅助指针数组，存放二叉树结点指针
Btree CreateBT ( )
{  int i , j; datatype ch;
   struct node *bt, *p; // bt为根，p 用于建立结点
   cin >> i>>ch ;
   while ( i != 0&&ch != 0) {
        p =new node;        p → data=ch;
        p → lchild=NULL;  p → lchild=NULL;
        s[ i ]=p;
        if  ( i == 1 ) bt = p ;
        else {    j =i /2;   // 父结点的编号
                  if ( i %2==0 )   s[ j ]→lchild=p;  // i 是 j 的左儿子
                  else                s[ j ]→rchild=p; // i 是 j 的右儿子
       }cin >> i>>ch ;}
        return bt;
}
```



练习题：已知二叉树的逻辑结构如下，试写出建立二叉树的算法。

二叉树用三元组表示（data, parent, tag)，左图表示如下： (A,#,#),(B,A,L),(C,A,R),(D,C,L)
(E,D,R),(#,#,#)结束标志



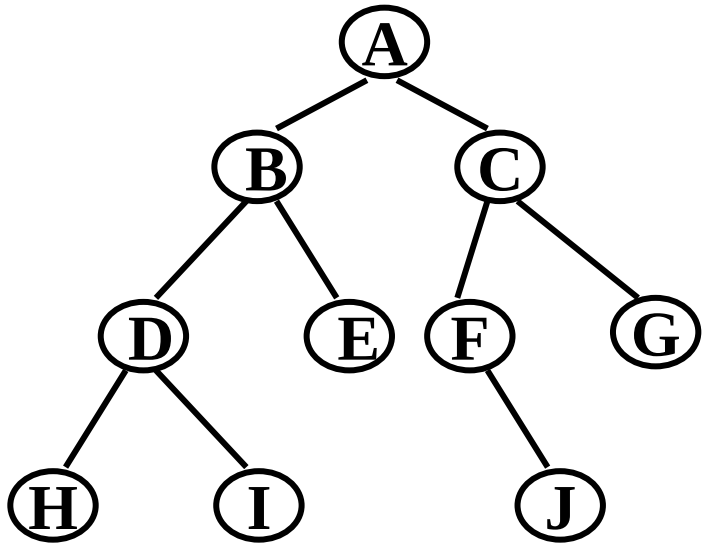
算法思想：

1、建立一个空树；
2、用一个队列存放输入的点；
3、每输入一个结点，建立并入队；
4、若其parent为‘#’，则为根，否则到队列中查找父结点，没找到出队；
5、根据读入的tag与双亲建立关系
重复3—5。

3.2 二叉树（Cont.）

- 二叉树左右链存储结构下的递归遍历算法.
  - 先序遍历

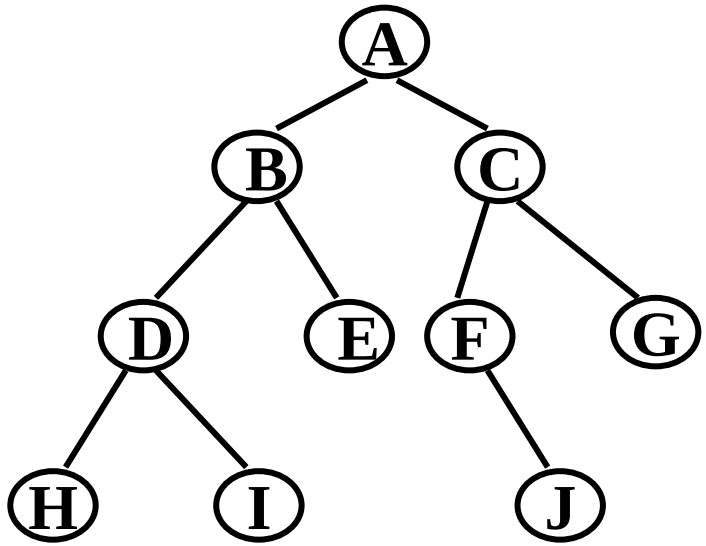
```
void  PreOrder (Btree BT )
{
    if ( BT != Null)
    {
        cout<< BT->data  ;
        PreOrder ( BT->lchild ) ;
        PreOrder ( BT->rchild ) ;
     }
}
```



3.2 二叉树（Cont.）

- 二叉树左右链存储结构下的递归遍历算法.
  - 中序遍历

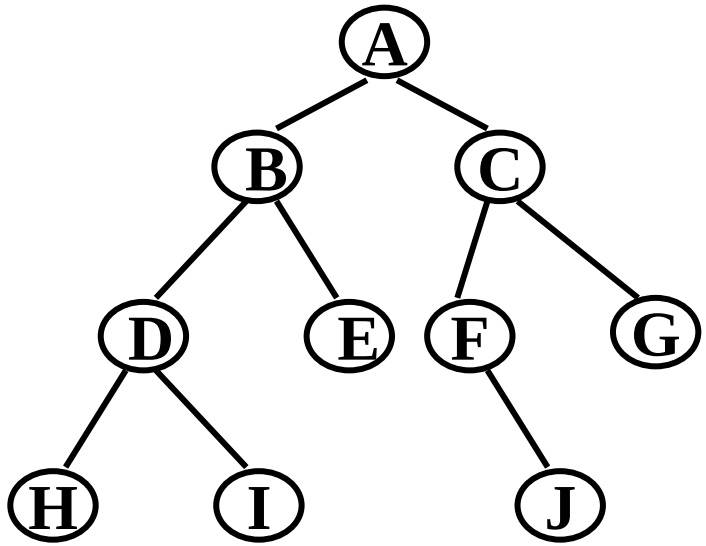
```
void  InOrder (Btree BT )
{
    if ( BT != Null)
    {
        InOrder ( BT->lchild ) ;
        cout<< BT->data  ;
        InOrder ( BT->rchild ) ;
     }
}
```



3.2 二叉树（Cont.）

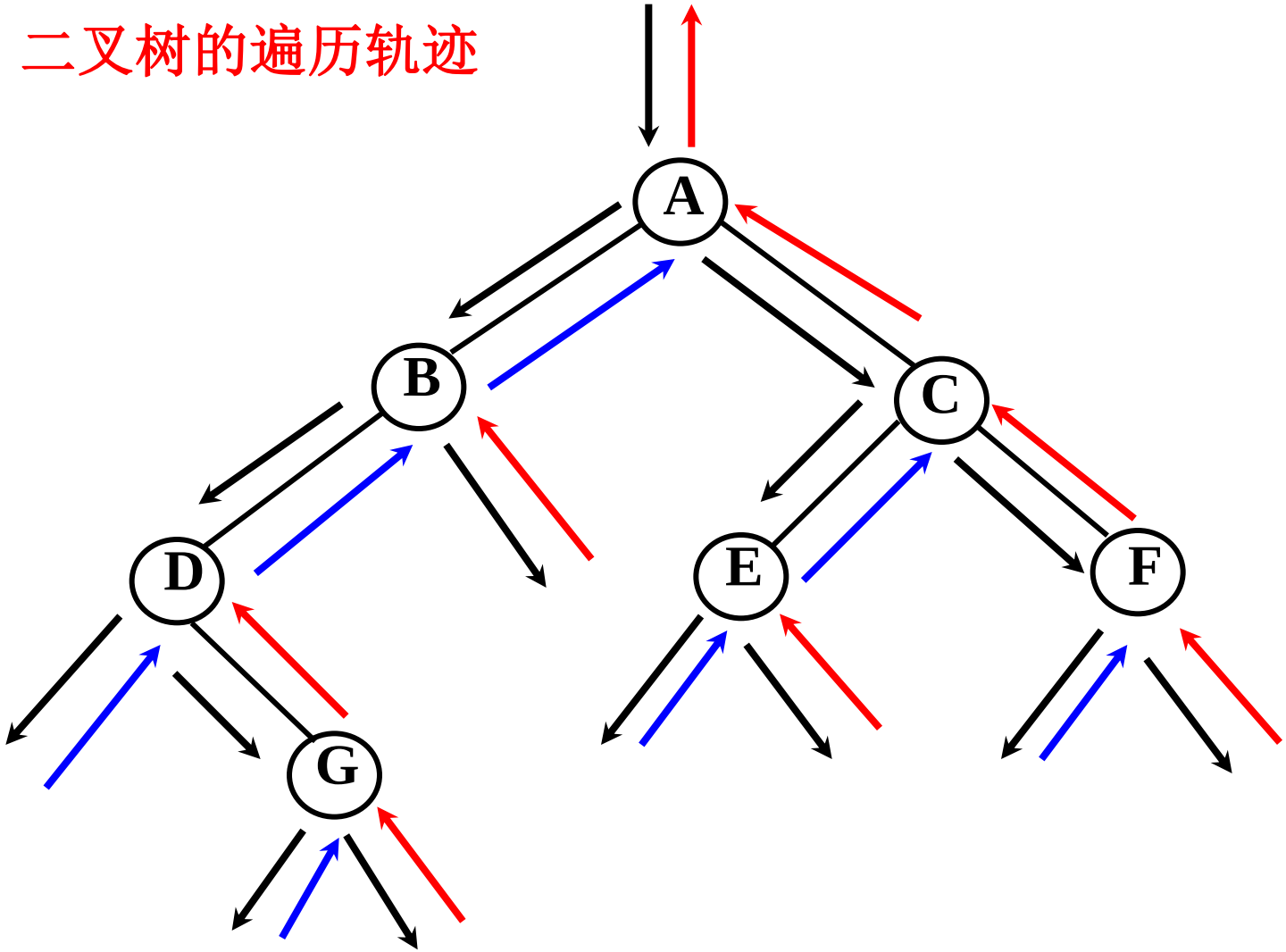
- 二叉树左右链存储结构下的递归遍历算法.
  - 后序遍历

```
void  PostOrder (Btree BT )
{
    if ( BT != Null)
    {
        PostOrder ( BT->lchild ) ;
        PostOrder ( BT->rchild ) ;
        cout<< BT->data  ;
     }
}
```

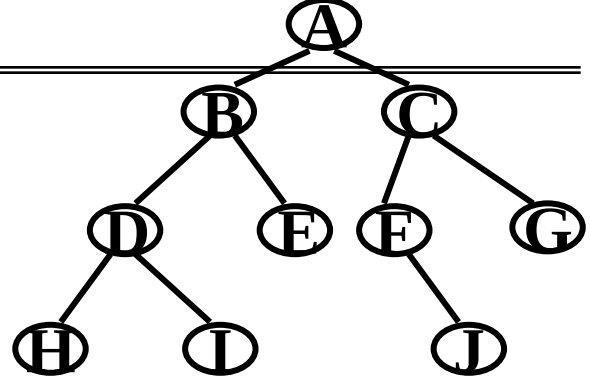


3.2 二叉树（Cont.）

- 二叉树左右链存储结构下的非递归遍历算法
  - 二叉树的遍历轨迹



3.2 二叉树（Cont.）



- 先序遍历非递归算法

1.栈s初始化；

2.循环直到root为空且栈s为空

2.1 当root不空时循环

2.1.1 输出root->data;

2.1.2 将指针root的值保存到栈中；

2.1.3 继续遍历root的左子树

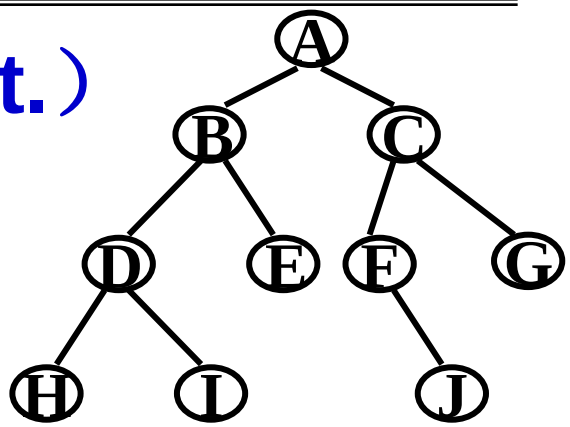
2.2 如果栈s不空，则

2.2.1 将栈顶元素弹出至root；

2.2.2 准备遍历root的右子树；}

```
void  PreOrder (Btree BT )
{
    if ( BT != Null)
    {
        cout<< BT->data  ;
        PreOrder ( BT->lchild ) ;
        PreOrder ( BT->rchild ) ;
    }
```

3.2 二叉树（Cont.）



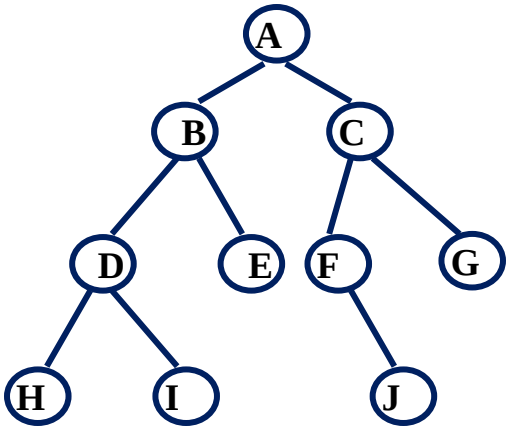
- 先序遍历非递归算法
- ---栈顶保存当前结点左子树

```
void PreOrder(Btree root)
{   top= -1;     //采用顺序栈，并假定不会发生上溢
    while (root!=Null | | top!= -1) {
        while (root!= Null) {
            cout<<root->data;
            s[++top]=root;
            root=root->lchild;
        }
        if (top!= -1) {
            root=s[top--];
            root=root->rchild;
        }
    }
}
```

```
Loop:
{
   if  (BT 非空)
      { 输出;
        进栈;
        左一步;}
   else
      { 退栈;
        右一步;}
};
```

先序遍历的非递归算法—栈顶保存当前结点的右子树

```
void PreOrder( Btree T )
 { Stack  S; MakeNull(S); //递归工作栈
   struct node* p = T;
   while ( p != Null ) {
       cout << p->data << endl;
       if ( p->rchild != Null )
          Push (S, p->rchild );
       if  ( p->lchild != Null )
           p = p->lchild;     //进左子树
       else {p=Top(S);Pop(S);}  //左子树空, 访问右子树 }}
```

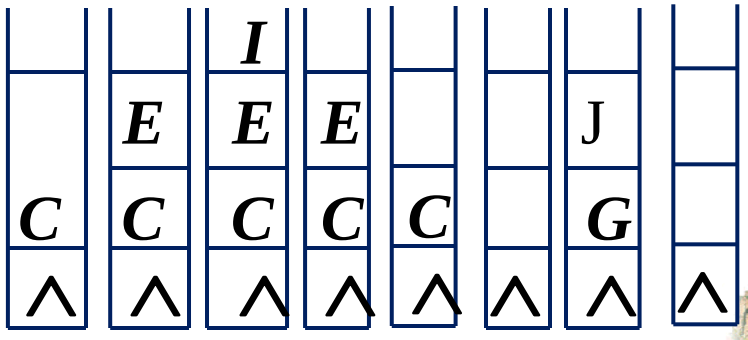


访问　A　B　D　H　I　E　C　F　J　G

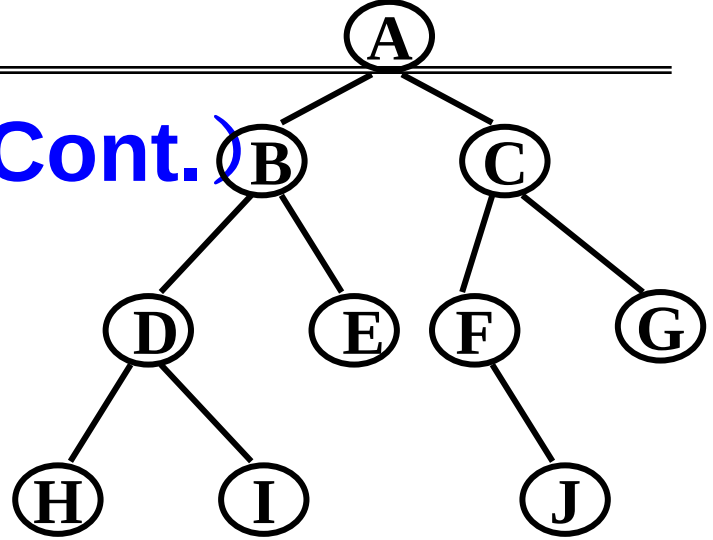
进栈　C　E　I　*　*　*　G　J

P向左前进　B　D　H　–　–　–　F

退栈　+　+　+　I　E　C　J　G



3.2 二叉树（Cont.）

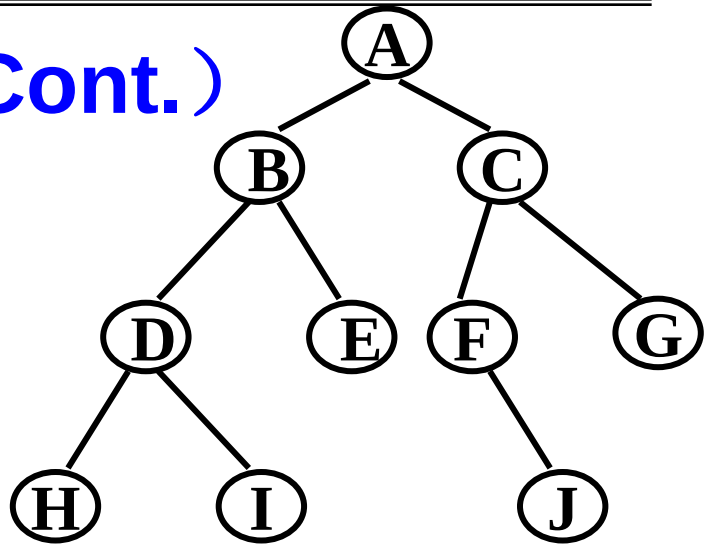


- 中序遍历非递归算法

1. 栈s初始化；
2. 循环直到root为空且栈s为空
   - 2.1 当root不空时循环
     - 2.1.1 将指针root的值保存到栈中；
       - 2.1.2 继续遍历root的左子树
   - 2.2 如果栈s不空，则
     - 2.2.1 将栈顶元素弹出至root；
       - 2.2.2 输出root->data;
     - 2.2.3 准备遍历root的右子树；

```
void  InOrder (Btree BT )
{
    if ( BT != Null)
    {
        InOrder ( BT->lchild ) ;
        cout<< BT->data  ;
        InOrder ( BT->rchild ) ;
    }
}
```

3.2 二叉树（Cont.）

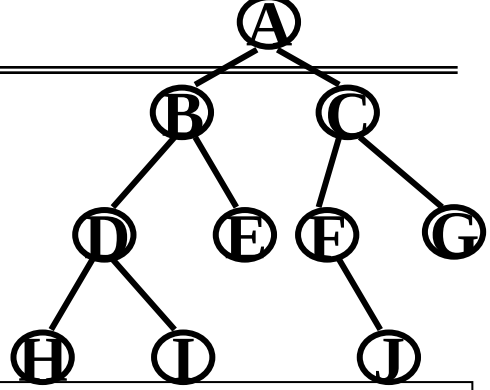


- 中序遍历非递归算法

```
void InOrder(Btree  root)
{   top= -1;
    while (root!=Null | | top!= -1) {
        while (root!= Null) {
            s[++top]=root;
            root=root->lchild;
        }
        if (top!= -1) {
            root=s[top--];
            cout<<root->data;
            root=root->rchild;
        }
    }
}
```

```
Loop:
{
   if  (BT 非空)
      { 进栈;
        左一步;}
   else
      {  退栈;
         输出;
         右一步;}
};
```

3.2 二叉树（Cont.）

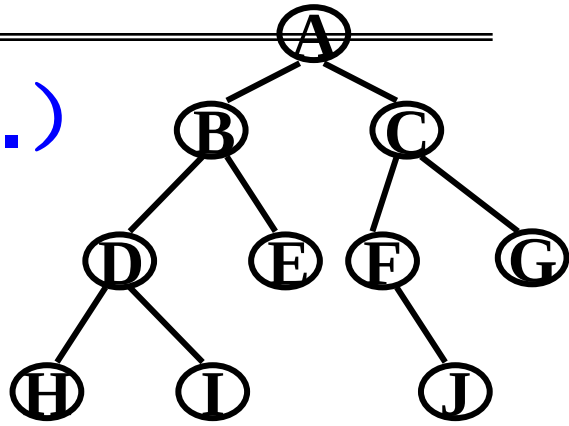


后序遍历非递归算法

1. 栈s初始化；
2. 循环直到root为空且栈s为空
   - 2.1 当root非空时循环
     - 2.1.1 将root连同标志flag=1入栈；
     - 2.1.2 继续遍历root的左子树；
   - 2.2 当栈s 非空且栈顶元素的标志为2 时，出栈并输出栈顶结点；
   - 2.3 若栈非空，将栈顶元素的标志改为2，准备遍历栈顶结点的右子树；

```
Loop:
{
   if (BT 非空)
      { 进栈;
        左一步;}
   else
      {  当栈顶指针
         所指结点的
         右子树不存
         在或已访问,
         退栈并访问;
         否则右一步; }
};
```

3.2 二叉树（Cont.）

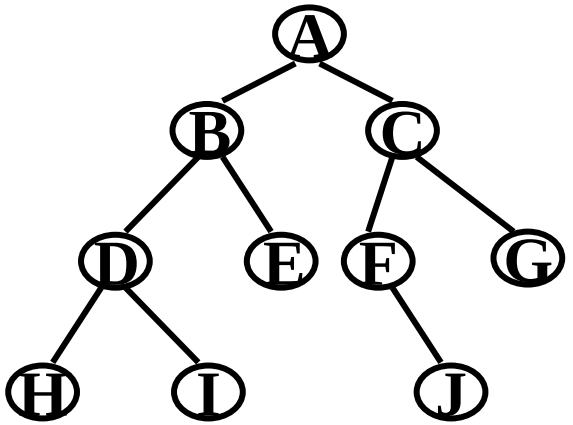


- 后序遍历非递归算法

```
void PostOrder(Btree root)
{   top= -1; //采用顺序栈，并假定栈不会发生上溢
    while (root!=Null | | top!= -1) {
        while (root!=Null) {
            top++;
            s[top].ptr=root;
            s[top].flag=1;
            root=root->lchild; }
     while (top!= -1 && s[top].flag==2)
           cout<<s[top--].ptr->data;
     if (top!= -1) {
           s[top].flag=2;
           root=s[top].ptr->rchild; }
     }
     }
```

```
Loop:
{
   if (BT 非空)
      { 进栈;
        左一步;}
   else
      {  当栈顶指针
         所指结点的
         右子树不存
         在或已访问,
         退栈并访问;
         否则右一步; }
};
```

```
void PostOrder (Btree t) (不设标志，设一变量）
{
  Btree p, pr; Stack s;
  MakeNull(s);
  p=t;
  while(p!=Null||!Empty(s))
  {
      while (p!=Null)
       { Push(p,s);
         pr=p->rc;p=p->lc;
         if(p==Null) p=pr;
        }
       p=Pop(s);visit(p->data);
      if(!Empty(s)&&Top(s)->lc==p)
           p=Top(s)->rc;
      else p=Null;
}}
```



练习题：在二叉树中增加两个域parent父结点，flag标志，写出不用栈进行后序遍历的非递归算法

flag用于区分在遍历过程中达到该点时的走向（初始时每个结点 的flag均为0）.

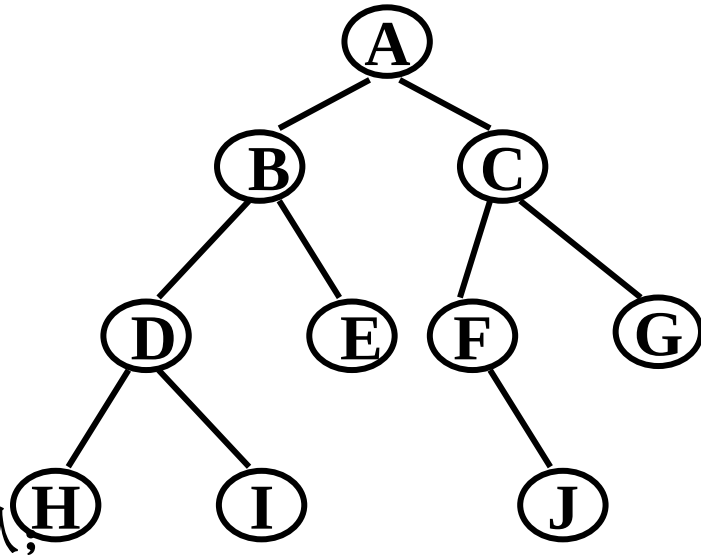
```
struct node {
        char data;
        node *lc,*rc,*parent;
        int flag;
       };
```

3.2 二叉树（Cont.）

层序遍历算法

- **基本思想：**按层次顺序遍历二叉树的原则是先被访问的结点的左、右儿子结点先被访问，因此，在遍历过程中需利用具有先进先出特性的队列结构
- **实现步骤：**

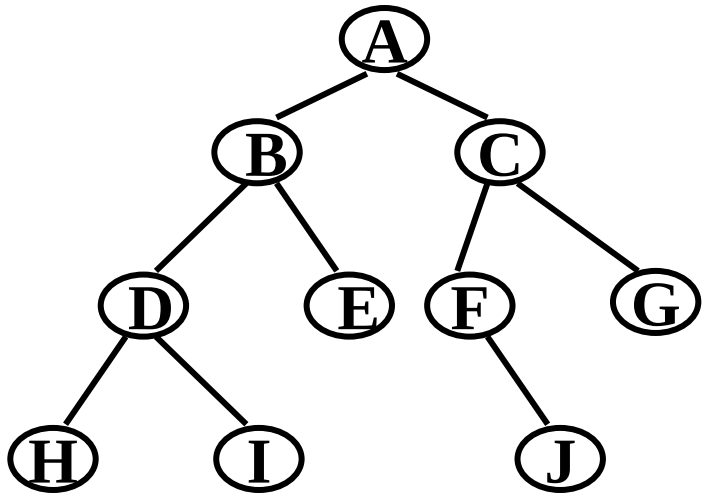
1. 队列Q初始化；
2. 如果二叉树非空，将根指针入队；
3. 循环直到队列Q为空
   - 3.1 q=队列Q的队头元素出队；
   - 3.2 访问结点q的数据域；
   - 3.3 若结点q存在左孩子，则将左孩子指针入队；
   - 3.4 若结点q存在右孩子，则将右孩子指针入队；



3.2 二叉树（Cont.）

- 层序遍历算法

```
void LeverOrder (Btree root)
{   front=rear=0;  //采用顺序队列，并假定不会发生上溢
    if (root==Null) return;
        Q[++rear]=root;
    while (front!=rear) {
        q=Q[++front];
        cout<<q->data;
        if (q->lchild!=Null)  Q[++rear]=q->lchild;
        if (q->rchild!=Null)  Q[++rear]=q->rchild;
    }
}
```



实验二　　树型结构及应用

- 1．编写建立二叉树的二叉链表存储结构（左右链表示）的程序，并以适当的形式显示和保存二叉树；
- 2．采用二叉树的二叉链表存储结构，编写程序实现二叉树的先序、中序和后序遍历的递归和非递归算法以及层序遍历算法，并以适当的形式显示和保存二叉树及其相应的遍历序列；
- 3．给定一个二叉树，**编写算法完成下列应用:（二选一）**
- **（1）判断其是否为完全二叉树；**
- **（2）求二叉树中任意两个结点的公共祖先。**