# 6.4二叉树的存储结构

## 顺序存储

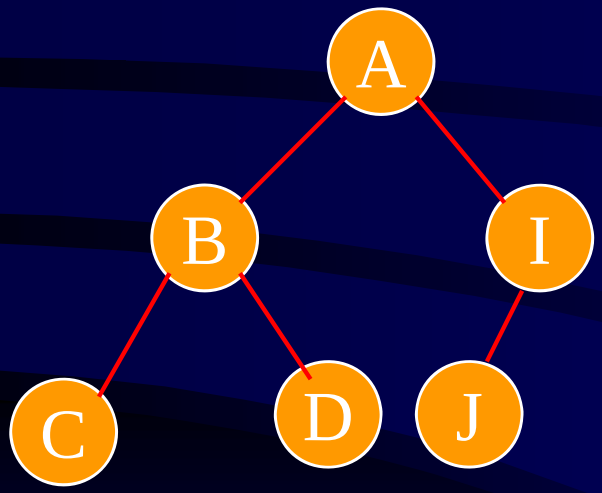
具体做法：

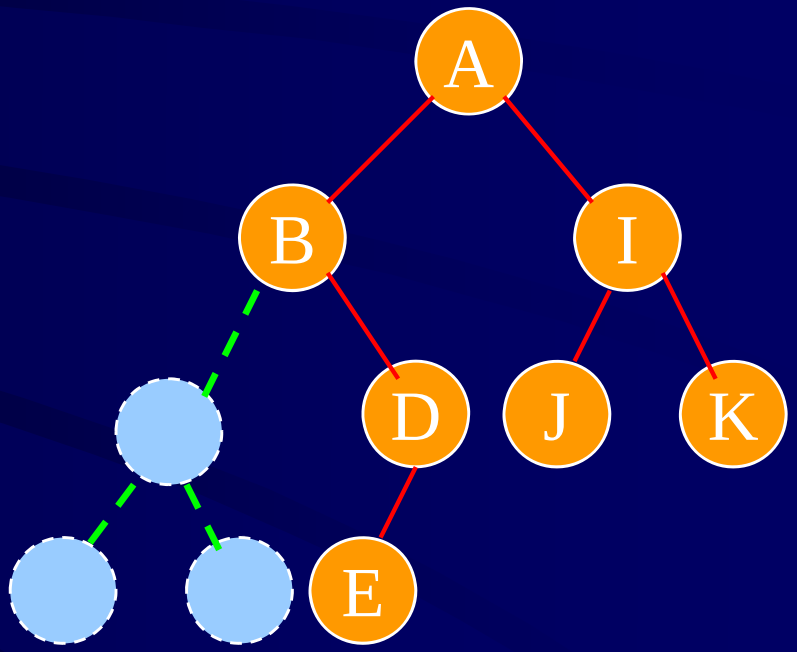
利用完全二叉树的性质，元素之间的关系隐含在元素的编号中，所以采用顺序存储后，完全二叉树中结点的存储位置为其编号所在的位置。对于一般二叉树，若采用顺序存储，须先增加虚结点以补成虚完全二叉树，对此虚完全二叉树的存储对应该二叉树的顺序存储。

```
//----------二叉树的顺序存储表示-----------
#define MAX_TREE_SIZE          100
typedef   TElemType        SqBiTree[MAX_TREE_SIZE ];
                                        //0号单元存放根结点
SqBiTree    bt;
```

# 6.4二叉树的存储结构（续）

## 顺序存储



# 6.4二叉树的存储结构（续）

## 二叉链表存储

| 结点 | data | lchild | rchild |
|---|---|---|---|

```
//----------二叉树的二叉链表存储表示-----------
typedef  struct  BiTNode{
          TElemType        data;      //数据域
          struct BiTNode   *lchild,  *rchild;
} BiTNode,    *BiTree;
```

# 6.4二叉树的存储结构（续）

## 三叉链表存储

| 结点 | data | lchild | rchild | parent |
|---|---|---|---|---|

```
//----------二叉树的三叉链表存储表示-----------
typedef  struct  BiTNode{
         TElemType        data;      //数据域
         struct BiTNode   *lchild,  *rchild,   *parent;
} BiTNode,    *BiTree;
```

# 6.4二叉树的存储结构（续）

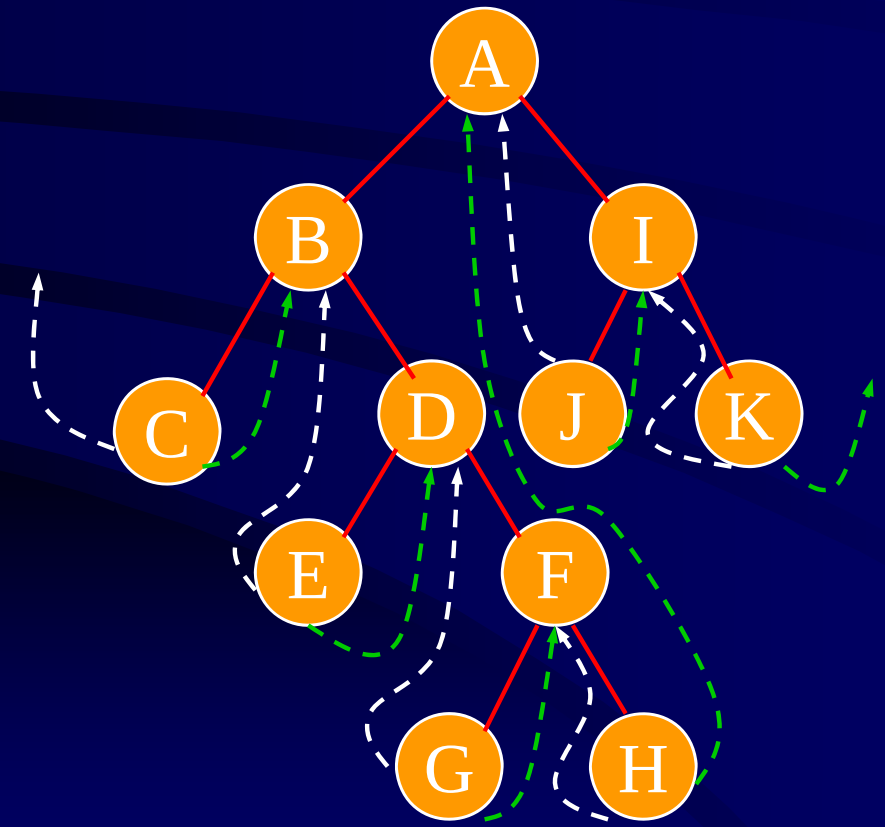
## （二叉）线索链表存储

| 结点 | data | lchild | ltag | rchild | rtag |
|---|---|---|---|---|---|

```
//----------二叉树的二叉线索链表存储表示-----------
typedef  enum {Link,Thread}  PointerTag;   //Link = =0; Thread = = 1
typedef  struct  BiThrNode{
            TElemType        data;      //数据域
            struct BiThrNode              *lchild,  *rchild;   //左、右孩子指针
            PointerTag          LTag,   Rtag;                  //左、右标志
} BiThrNode,    *BiThrTree;
```

# 6.4二叉树的存储结构（续）

## （二叉）线索链表存储



# 6.5二叉树的遍历及线索化

DS/SS

## 遍历二叉树

遍历(Traversing)：按照某条路径访问每个结点,使得每个结点均被访问一次，而且仅被访问一次。

- 在二叉树中查找具有某种特征的结点
- 需要对结点进行访问（处理、输出等）操作
- 二叉树上许多复杂操作的基础是对其遍历

- 遍历本质：结点之间非线性关系线性化的过程
- 两种思路：

*只要保证每个结点都被访问到一次，

*遍历过程中易于实现关于结点的复杂操

层次遍历法

递归遍历法：

先序、中序、后序

# 6.5二叉树的遍历及线索化（续）

DS/SS

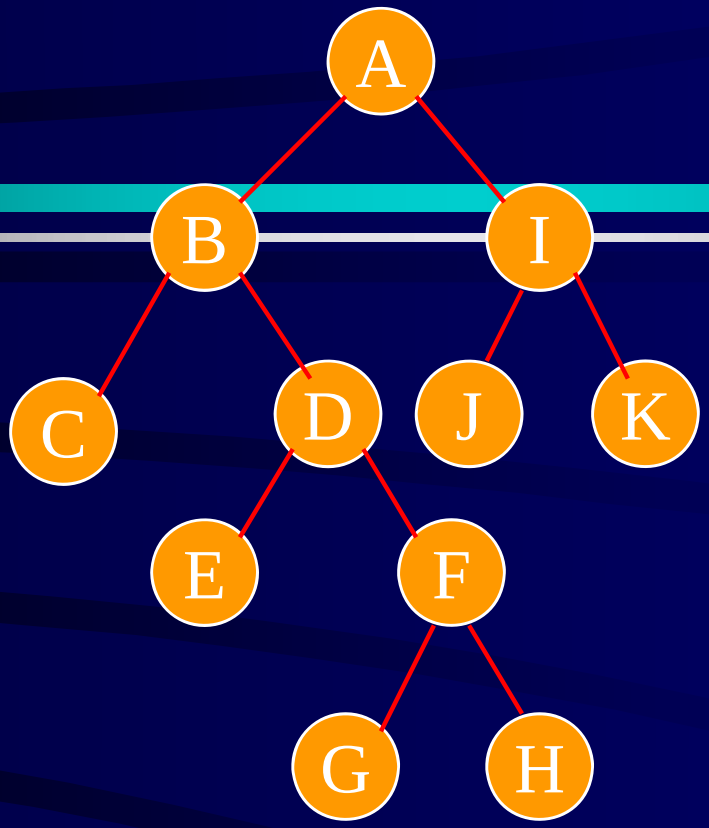
## 遍历二叉树的方法

- 层次遍历(LevelOrderTraverse)：从上到下，自左至右按照层次遍历。
- 先序遍历(PreOrderTraverse)：DLR;

  若二叉树为空，则空操作（递归基础）；否则：访问根结点；先序遍历左子树；先序遍历右子树。
- 中序遍历(InOrderTraverse)：LDR;

  若二叉树为空，则空操作（递归基础）；否则：中序遍历左子树；访问根结点；中序遍历右子树。
- 后序遍历(PostOrderTraverse)：LRD;

  若二叉树为空，则空操作（递归基础）；否则：后序遍历左子树；中序遍历右子树；访问根结点。




- 层次遍历： A,B,I,C,D,J,K,E,F,G,H
- 先序遍历： A,B,C,D,E,F,G,H,I,J,K
- 中序遍历： C,B,E,D,G,F,H,A,J,I,K
- 后序遍历： C,E,G,H F,D,B,J,K,I,A

# 6.5二叉树的遍历及线索化（续）

DS/SS

## 中序遍历的算法

```
Status InOrderTraverse(BiTree  T, Status (* Visit) (TElemType  e)) {
  //中序遍历二叉树T的递归算法，对每个数据元素调用函数Visit
   if(T) {
        if(InOrderTraverse(T->lchild,Visit))
            if(Visit(T->data)){
                if(InOrderTraverse(T->rchild,Visit))          return OK;
            } else   return ERROR;   //当访问失败时，出错
   } else            return OK;      //一次递归访问终止
} InOrderTraverse  // 算法时间复杂度：O(n)
```

# 6.5二叉树的遍历及线索化（续）

DS/SS

```
Status InOrderTraverse1(BiTree  T, Status (* Visit) (TElemType  e)) {
  //中序遍历二叉树T的非递归算法，对每个数据元素调用函数Visit
  InitStack(S);   Push(S,T);        //根指针入栈
  While (!StackEmpty(S))  {
      while  (GetTop(S,p)  &&p)  Push (S,p->lchild);   //向左走到尽头
      Pop(S,p);              //空指针退栈
      if (! StackEmpty(S)) {       //访问结点，向右一步
          Pop(S,p);       if (!Visit(p->data))  return ERROR  //访问出错
          Push(S,p->rchild);  } //if
  } //while
  return OK;       } InOrderTraverse    // 算法时间复杂度：O(n)
```

# 6.5二叉树的遍历及线索化（续）

DS/SS

```
Status InOrderTraverse2(BiTree  T, Status (* Visit) (TElemType  e)) {
  //中序遍历二叉树T的非递归算法，对每个数据元素调用函数Visit
  InitStack(S);    p = T;
  While ( p || !StackEmpty(S))  {
        if (p)  { Push (S,p);  p = p->lchild; }   //根指针入栈,遍历左子树
        else {                  //根指针退栈，访问根结点，遍历右子树
            Pop(S,p);       if (!Visit(p->data))  return ERROR  //访问出错
            p = p->rchild);  } //else
  } //while
  return OK;       } InOrderTraverse    // 算法时间复杂度：O(n)
```

# 6.5二叉树的遍历及线索化（续）

DS/SS

## 二叉树遍历的典型应用

二叉树表示表达式的递归定义：

若表达式为数或简单变量，则相应二叉树中仅有一个根结点，其数据域存放该表达式信息。

若表达式=（第一操作数）（运算符）（第二操作数），则相应的二叉树中以左子树表示第一操作数；右子树表示第二操作数；根结点的数据域存放运算符（若为一元运算符，则左子树为空）。操作数本身又为表达式。

前缀表示（波兰式）

-+a*b-cd/ef

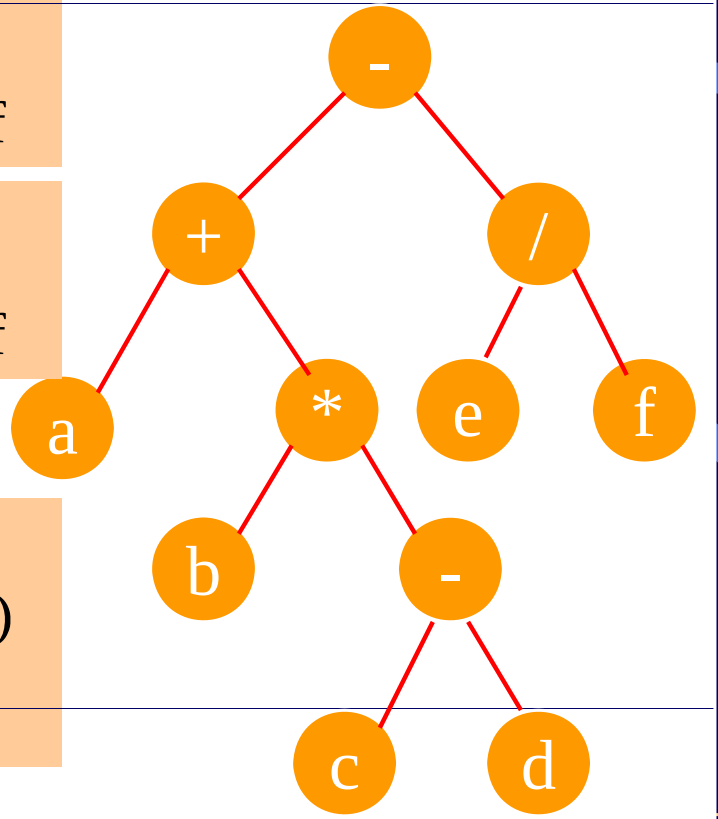
中缀表示

a+b*c-d-e/f

后缀表示（逆波兰式）

abcd-*+ef/-

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



# 6.5二叉树的遍历及线索化（续）

DS/SS

## 二叉树遍历的典型应用

已知二叉树的前序遍历序列和中序遍历序列，或者已知二叉树的后序遍历序列和中序遍历序列，可以唯一确定这棵二叉树。

InOrder:

C B E D G F H A J I K

PostOrder:

C E G H F D B J K I A

Step1:判定根，由PostOrder知根为A；

Step2:判定左子树元素集合，由InOrder知为{C B E D G F H }

Step3:判定右子树元素集合，由InOrder知为{J I K}

Step4:分别对左、右子树元素集合按照中序和后序序列递归进行Step1---Step3，直到元素集合为空。