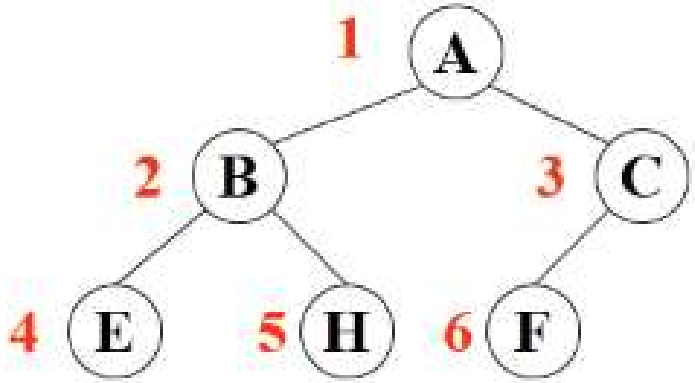
# 二叉树的存储结构

## 1.二叉树的顺序存储

完全二叉树的顺序存储：



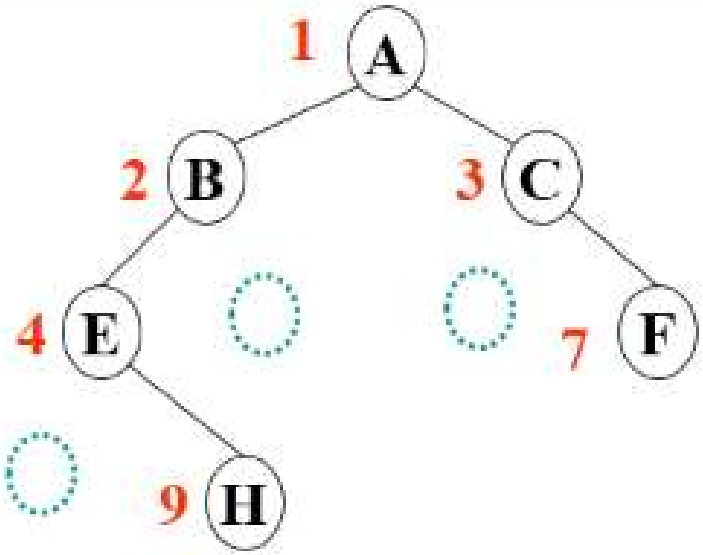

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| ST[ ] | | A | B | C | E | H | F |

根据性质5知： ST[i] 的双亲是ST[$\lfloor i/2 \rfloor$],

左子女是ST[2*i]，右子女是ST[2*i+1].

# 二叉树的存储结构

## 1.二叉树的顺序存储




| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| ST[ ] | | A | B | C | E | | | F | | H |

根据性质5知： ST[i] 的双亲是ST[$\lfloor i/2 \rfloor$]，左子女是ST[2*i],右子女是ST[2*i+1].

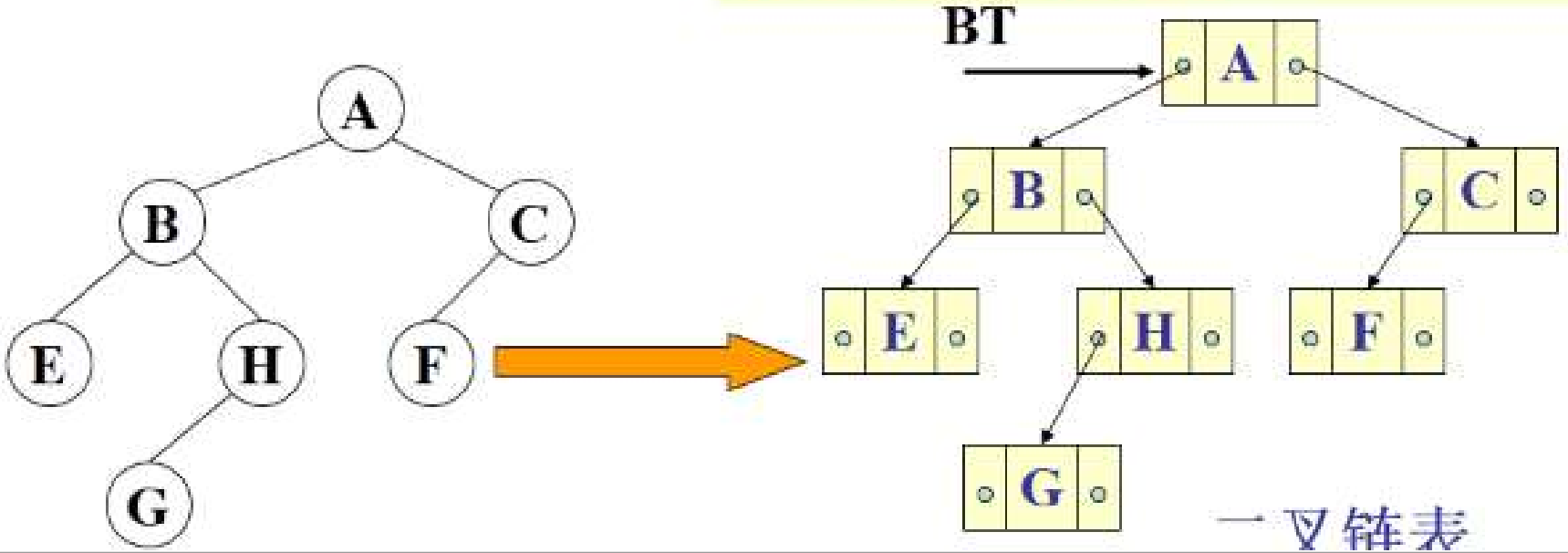
这样太浪费空间,适合完全二叉树

# 二叉树的存储结构

## 2.二叉树的链式存储结构(二叉链表)

**BiTNode:**

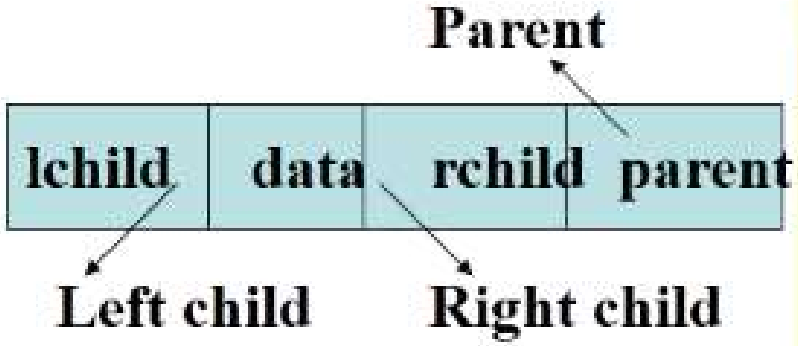

```
typedef struct   BiTNode{
    ElemType    data;
    struct   BiTNode   *lchild,* rchild;
}BiTNode, *BiTree;
```



# 二叉树的存储结构

## 2.二叉树的链式存储结构(三叉链表)

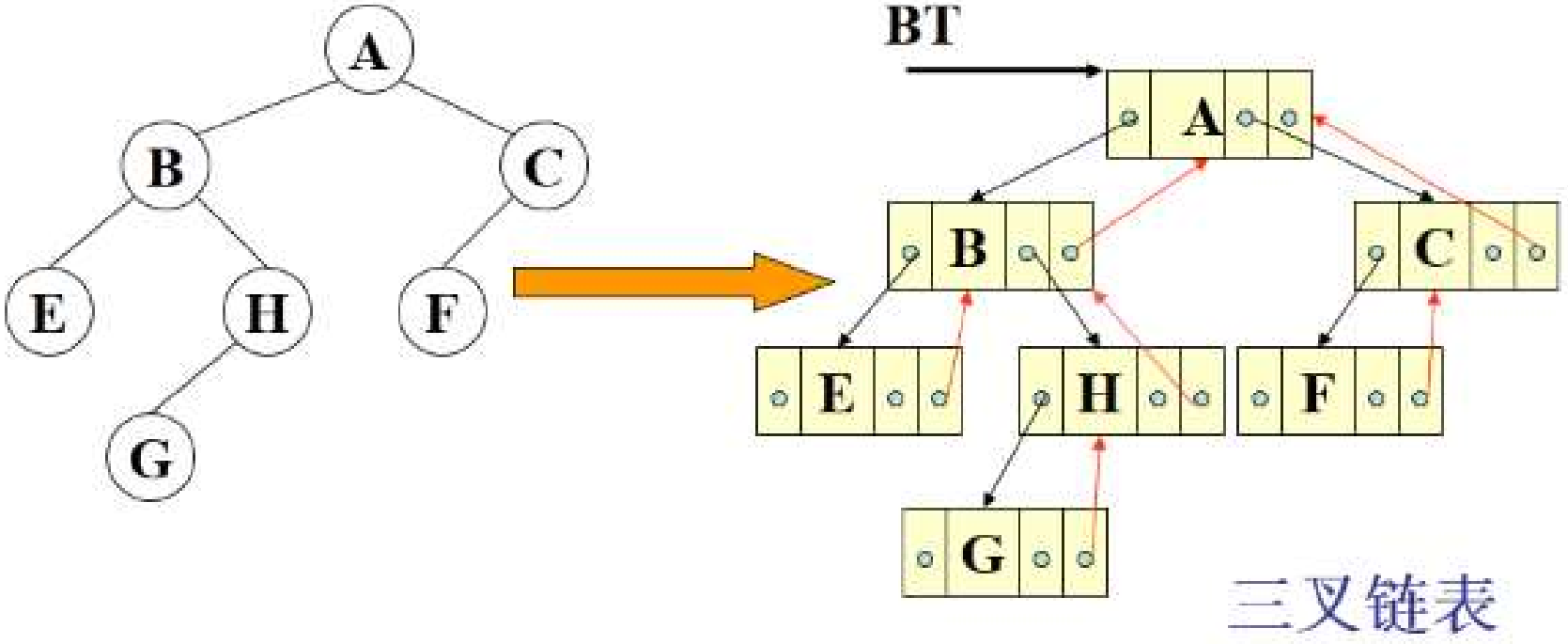
**BiTNode:**




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

# 二叉树的存储结构

## 2.二叉树的链式存储结构(三叉链表)



# 如何实现对二叉树数据元素的遍历

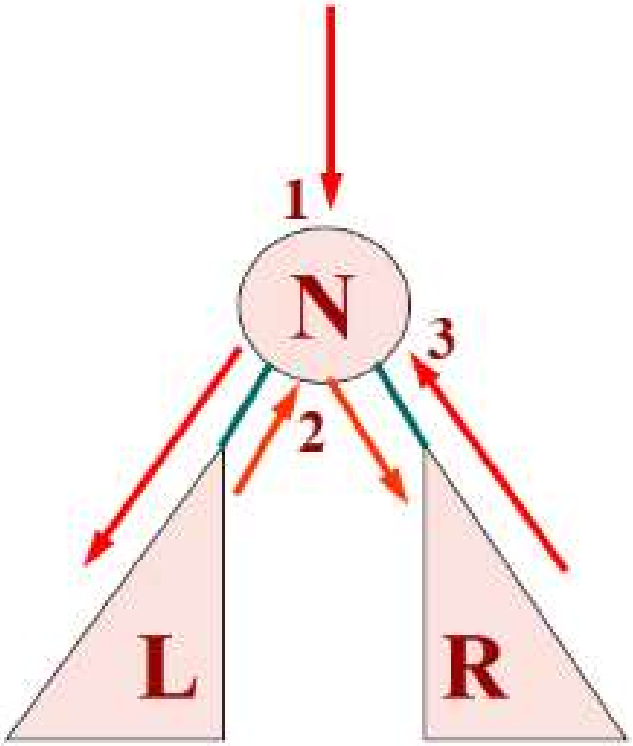
# 二叉树的遍历

遍历是指按照某种搜索路径访问二叉树中的所有结点，使得每个结点被访问一次且仅被访问一次。

“访问”的含义特别很广，如：输出结点的信息等。

因二叉树是非线性结构，每个结点可能有两个后继，则存在如何遍历即按什么样的搜索路径遍历的问题。

# 二叉树的遍历




**NLR** 前(先)序 遍历

**LNR** 中序遍历

**LRN** 后序遍历

~~NRL~~

~~RNL~~

~~RLN~~

# 二叉树的遍历

**算法思想6.1**

前序遍历：

若BT非空，则：

1.访问根结点；

2.前序遍历左子树；

3.前序遍历右子树；

**算法思想6.2**

中序遍历：

若BT非空，则：

1.中序遍历左子树；

2.访问根结点；

3.中序遍历右子树；

**算法思想6.3**

后序遍历：

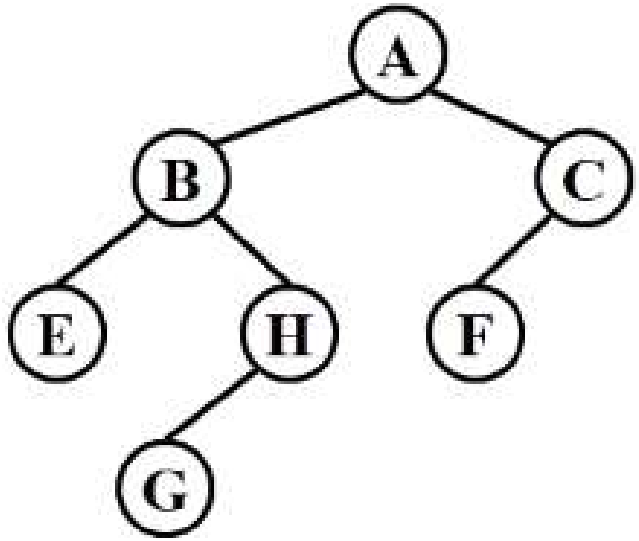
若BT非空，则：

1.后序遍历左子树；

2.后序遍历右子树；

3.访问结点；

# 二叉树的遍历




算法思想6.1

前序遍历：

若BT非空，则：

1.访问根结点；

2.前序遍历左子树；

3.前序遍历右子树；

前序遍历(NLR)序列： A B E H G C F

中序遍历(LNR)序列： E B G H A F C

后序遍历(LRN)序列： E G H B F C A