## 关于++i和i++http://blog.csdn.net/david8631/article/details/2707914

## 第二讲 线性结构

##### 2.1 线性表及其实现

1. **什么是线性表？**

线性表(Linear List)：由同类型数据元素构成有序序列的线性结构，表中元素个数称为线性表的长度，线性表没有元素时，称为空表表起始位置称表头，表结束位置称表尾。

相关方法：

1、List MakeEmpty()：初始化一个空线性表L；

2、ElementType FindKth( int K, List L )：根据位序K，返回相应元素 ；

3、int Find( ElementType X, List L )：在线性表L中查找X的第一次出现位置；

4、void Insert( ElementType X, int i, List L)：在位序i前插入一个新元素X；

5、void Delete( int i, List L )：删除指定位序i的元素；

6、int Length( List L )：返回线性表L的长度n。

1. **线性表的顺序存储方式**

利用数组实现

typedef struct{

ElementType Data[MAXSIZE];

int Last;

} List;

List L, \*PtrL;

访问下标为 i 的元素：L.Data[i] 或 PtrL->Data[i]线性表的长度：L.Last+1 或 PtrL->Last+1关于C/C++中声明struct的不同：http://www.cnblogs.com/qyaizs/articles/2039101.html

在C中声明struct一定是要用typedef，如上所述，而C++中可以直接使用struct：

Struct student{ typedef struct student{ int a;}stu2//sut2是一个结

Int a; 结构体类型

}stu1;//stu1是一个变量.

1. 初始化：

List \*MakeEmpty(){

List \*PtrL;

PtrL = (List \*)malloc( sizeof(List) );

PtrL->Last = -1;//Ptrl->Last是线性表的最后一个元素的下标。

return PtrL;

}

1. 查找

int Find( ElementType X, List \*PtrL )

{ int i = 0;

while( i <= PtrL->Last && PtrL->Data[i]!= X )

i++;

if (i > PtrL->Last) return -1; /\* 如果没找到，返回-1 \*/

else return i; /\* 找到后返回的是存储位置 \*/

}

1. 插入 在第i个位置插入一个值为X新元素，需要先把地i个位置之后的元素向后移动一个位置，必须倒序移动。然后再插入

void Insert( ElementType X, int i, List \*PtrL )

{ int j;

if ( PtrL->Last == MAXSIZE-1 ){ /\* 表空间已满，不能插入\*/

printf(＂表满＂);

return;

}

if ( i < 1 || i > PtrL->Last+2) { /\*检查插入位置的合法性\*/

printf(＂位置不合法＂);

return;

}

for ( j = PtrL->Last; j >= i-1; j-- )

PtrL->Data[j+1] = PtrL->Data[j]; /\*将 ai～ an倒序向后移动\*/

PtrL->Data[i-1] = X; /\*新元素插入\*/

PtrL->Last++; /\*Last仍指向最后元素\*/

return;

}

1. 删除表的第i个位置上的元素。也就是下标为i-1的相应元素。需要将i-1后面的元素都依次向前移动一个位置。

void Delete( int i, List \*PtrL )

{ int j;

if( i < 1 || i > PtrL->Last+1 ) { /\*检查空表及删除位置的合法性\*/

printf (“不存在第%d个元素”, i );

return ;

}

for ( j = i; j <= PtrL->Last; j++ )

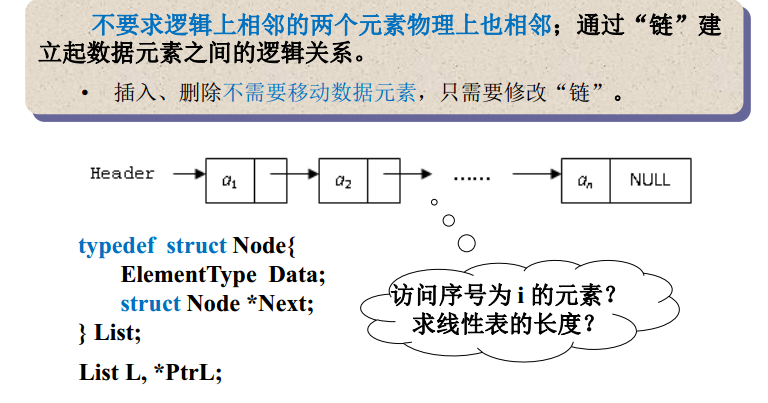
PtrL->Data[j-1] = PtrL->Data[j]; /\*将 ai+1～ an顺序向前移动\*/

PtrL->Last--; /\*Last仍指向最后元素\*/

return;

}

1. **线性表的链式存储实现**
2. 用单向链表实现：



每一个Node节点，存储的是数据域Data和指针域\*Next，是指向下一个Node节点的指针。假设p是指向第i个数据元素的指针，则p->Next 是指向第i+1个数据元素的指针，也就是如果p->Data = Ai,则p->Next->Data = Ai+1;

1. 求表长：

int Length ( List \*PtrL )

{ List \*p = PtrL; /\* p指向表的第一个结点\*/

int j = 0;

while ( p ) {

p = p->Next;

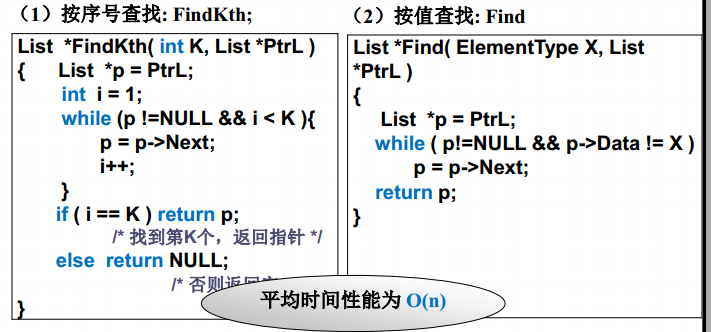
j++; /\* 当前p指向的是第 j 个结点\*/

}

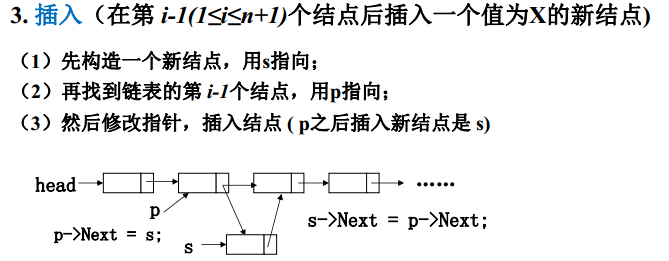
return j;

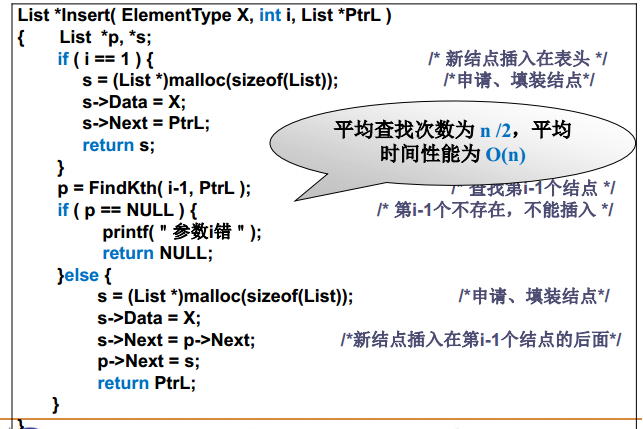
}

1. 查找

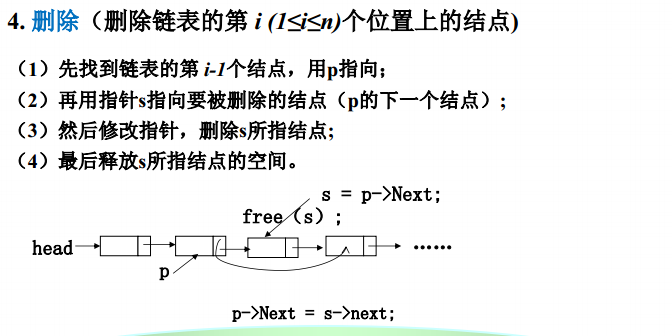


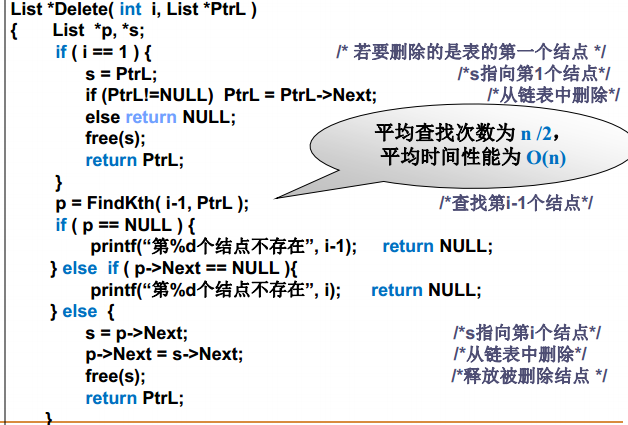
1. 插入





1. 删除操作





1. 循环链表

类似于双向链表，特点是表中的最后一个节点的指针指向头结点。

1. 双向链表

为了克服单向链表的劣势，就是如果执行NextElem则时间为O(1),如果执行查找前一个元素PriorElem则为O(n).有两个指针域，一个指向后面，另一个指向前面。数据存储结构如下：

Typedef struct DulNode{

ElemType Data;

Struct DulNode \*Prior;

Struct DulNode \*Next;

}DulNode, \*DuLinkList

但是双向链表的插入和删除操作与单向链表有很大的不同

##### 2.2 堆栈

1. 什么是堆栈，基本定义?

堆栈（Stack）：具有一定操作约束的线性表只在一端（栈顶，Top）做 插入、删除

插入数据：入栈（Push）;删除数据:出栈（Pop）;后入先出：Last In First Out（LIFO）

类型名称: 堆栈（Stack）

数据对象集：一个有0个或多个元素的有穷线性表。

操作集：长度为MaxSize的堆栈S属于 Stack，堆栈元素item属于 ElementType

1、Stack CreateStack( int MaxSize )： 生成空堆栈，其最大长度为MaxSize；

2、int IsFull( Stack S, int MaxSize )：判断堆栈S是否已满；

3、void Push( Stack S, ElementType item )：将元素item压入堆栈；

4、int IsEmpty ( Stack S )：判断堆栈S是否为空；

5、ElementType Pop( Stack S )：删除并返回栈顶元素

栈的顺序存储结构通常由一个一维数组和一个记录栈顶元素位置的变量组成。

#define MaxSize <最大个数>

typedef struct{

ElementType Data[MaxSize];

int Top;

}Stack;

1. 堆栈的顺序存储-入栈操作

void Push(Stack \*Ptrs,ElementType item){

if(Ptrs->Top == MaxSize-1){

printf(“full”); return;

}

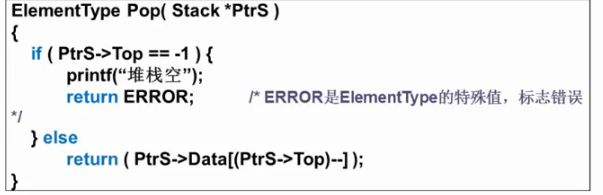
else{

Ptrs->Data[++(Ptrs->Top)] =item;

}

}

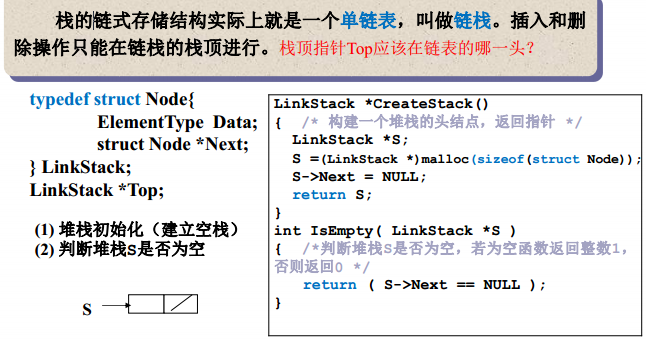
1. 堆栈的顺序-出栈操作

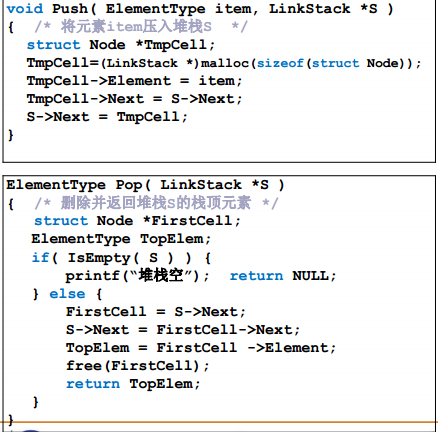


注意：入栈和出栈的时候++和--的位置，入栈：先把Top加一；而出栈的时候要先取出栈顶

元素然后Top再减一。

1. 堆栈的链式存储 入栈和出栈





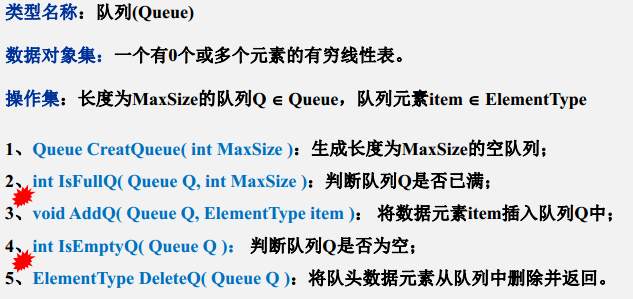
##### 2.2 队列

1. **什么是队列？**

队列(Queue)：具有一定操作约束的线性表插入和删除操作：只能在一端插入，而在另一端删除。

数据插入：入队列（AddQ） 数据删除：出队列（DeleteQ）先来先服务先进先出FIFO

队列的抽象数据模型：



1. **队列的顺序存储实现**

队列的顺序存储结构通常是由一个一维数组和一个记录队列头元素位置的变量front和一个记录队列尾元素的位置变量rear组成。

#define MaxSize <储存数据元素的最大个数>

typedef struct {

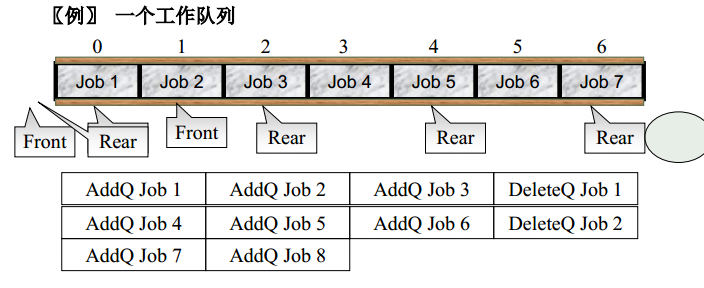
ElementType Data[ MaxSize ];

int rear;

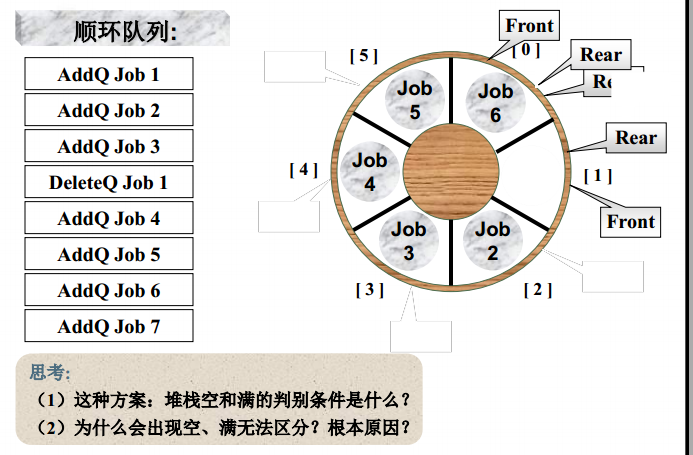
int front;

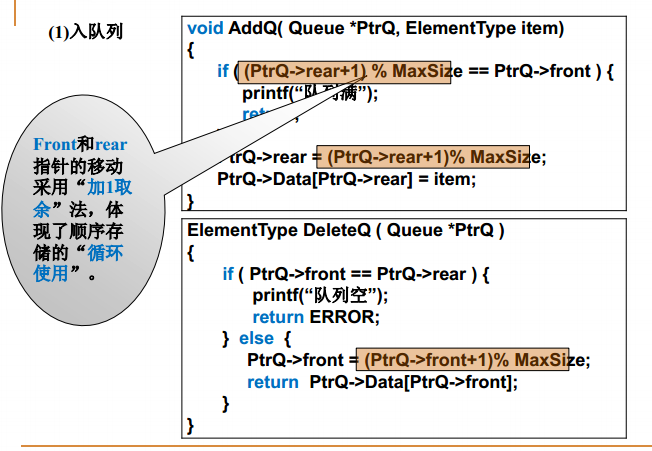
} Queue;

一个工作队列：（front是指的第一个元素的前一个位置，而rear是指的就是最后一个元素）不断的入队和出队的过程中，会出现一个问题是当数组满的时候而前面的元素还是空的，所以这个时候就出现了循环队列。

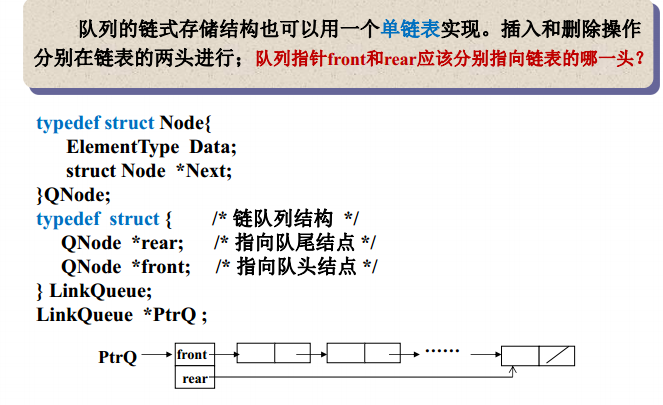


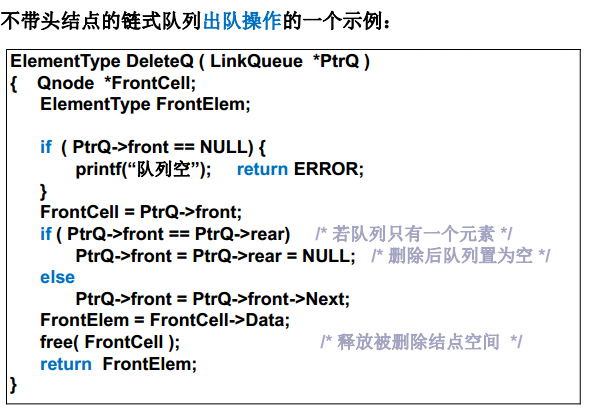
注意循环队列有一个是判断为满的条件。不能根据rear = front ，因为当对列是空的时候rear也是等于front的， 用到的方法是不把数组存满，而是仅仅存取n-1个元素。判断满的条件是通过用当前rear值对MAXSIZE取余，如果是0则满，不是O则还是当前值，程序如下图所示：





1. **队列的链式存储**





## 第三讲 树（上）

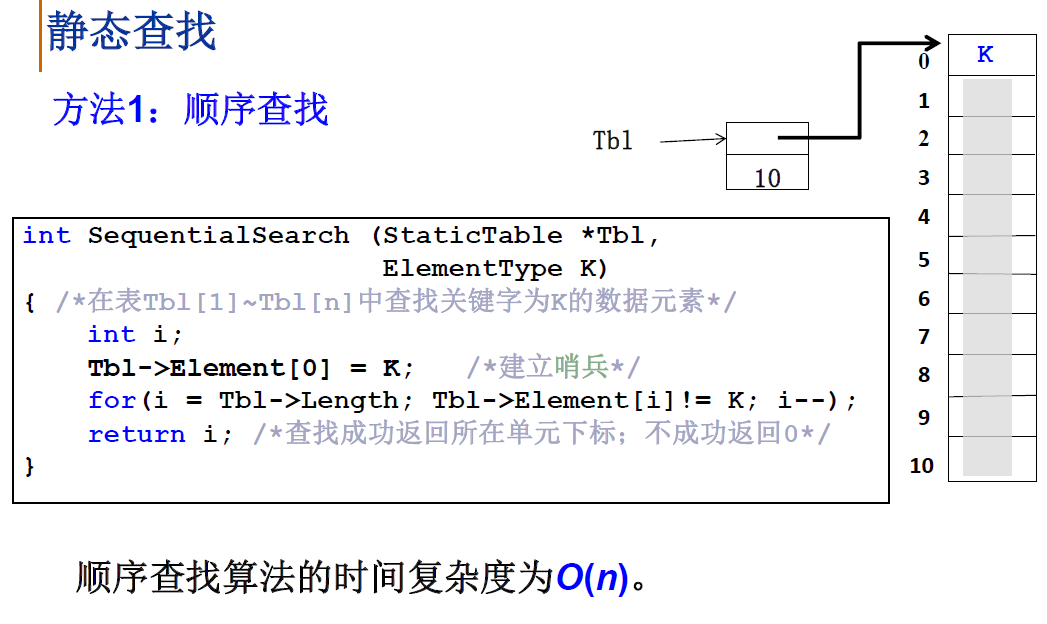
##### 3.1 树与树的表示

1. **什么是树**

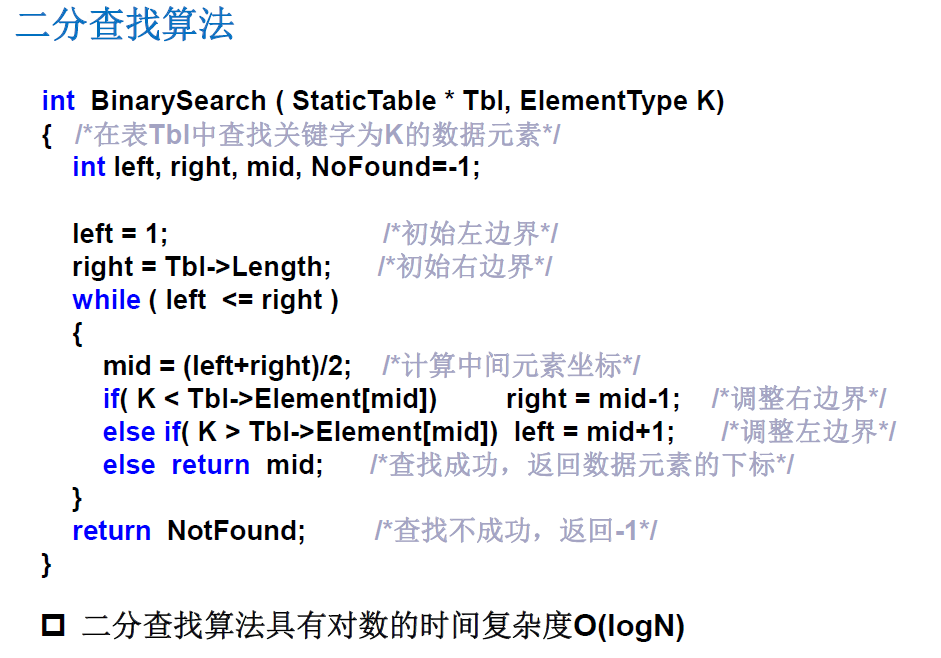
**分层次组织在管理上具有更高的效率**

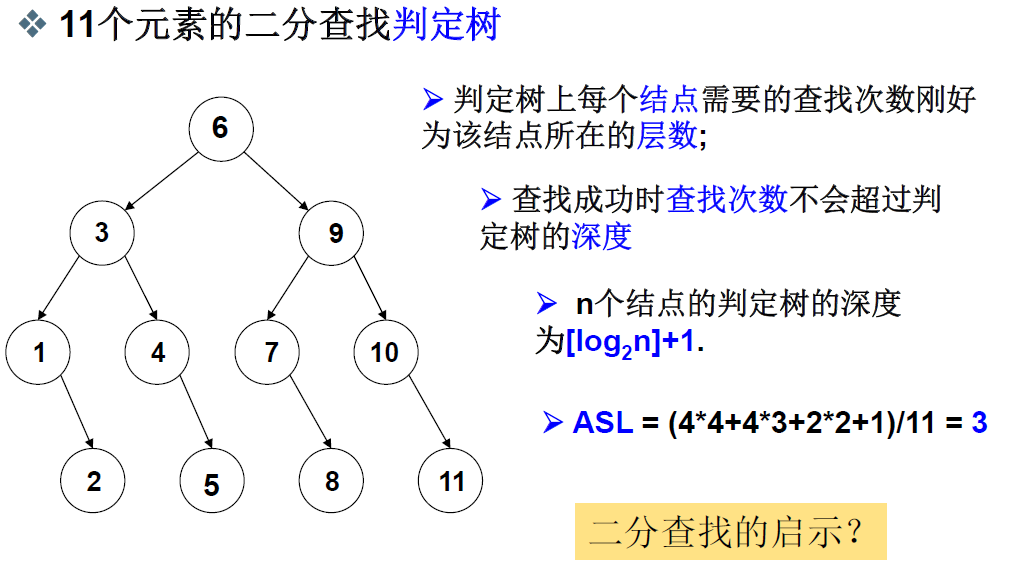
1. **静态查找**

* **方法一：顺序查找 通过建立哨兵，值和要查找的值相等的时候，return根据返回的i 来判断是否是所要查找的K**



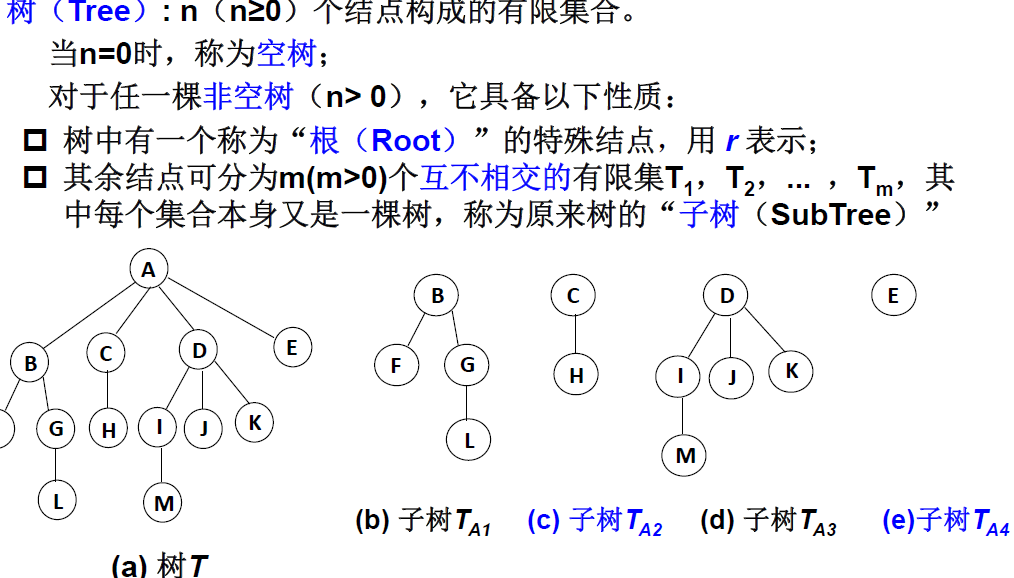
* **方法二：二分法查找,二分法查找必须要查找的数据放到一个有序数组，并且数组中的内容是排序好的。**





只要按照这种有序结构然后进行查找，效率就很高

1. **动态查找**



**树的特点：**

子树是不相交的； 除了根结点外，每个结点有且仅有一个父结点； 一棵N个结点的树有N-1条边。

**树的术语：**

1.结点的度（Degree）：结点的子树个数

2.树的度：树的所有结点中最大的度数

3.叶结点（Leaf）：度为0的结点

4.父结点（Parent）：有子树的结点是其子树的根结点的父结点

5.子结点（Child）：若A结点是B结点的父结点，则称B结点是A结点的子结点；子结点也称孩子结点。

6.兄弟结点（Sibling）：具有同一父结点的各结点彼此是兄弟结点。

7.路径和路径长度：从结点n1到nk的路径为一个结点序列n1 , n2 ,… , nk , ni是 ni+1的父结点。路径所包含边的个数为路径的长度。

9. 祖先结点(Ancestor)：沿树根到某一结点路径上的所有结点都是这个结点的祖先结点。

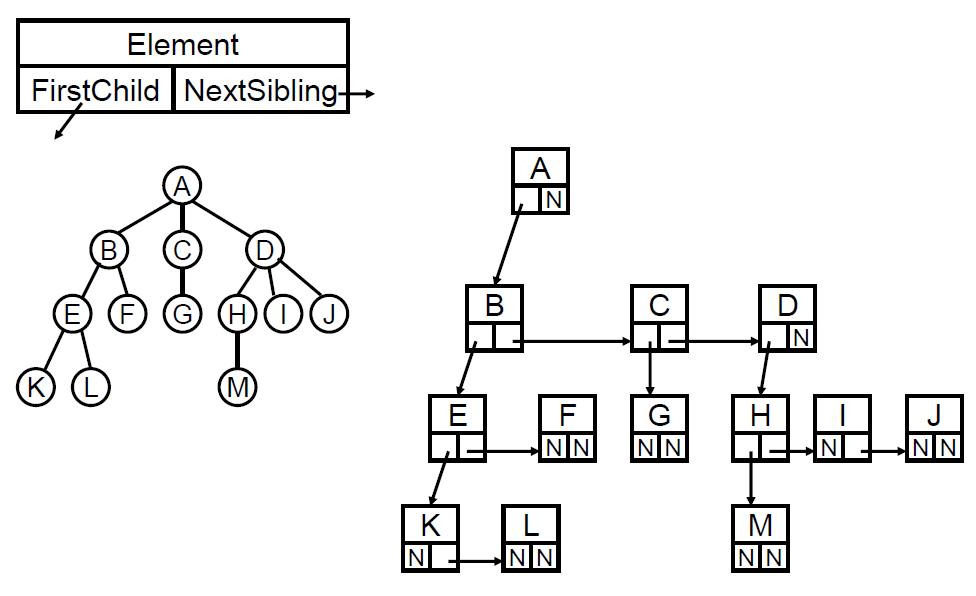
10. 子孙结点(Descendant)：某一结点的子树中的所有结点是这个结点的子孙。

11. 结点的层次（Level）：规定根结点在1层，其它任一结点的层数是其父结点的层数加1。

12. 树的深度（Depth）：树中所有结点中的最大层次是这棵树的深度。

**树的表示：**

儿子兄弟表示法：



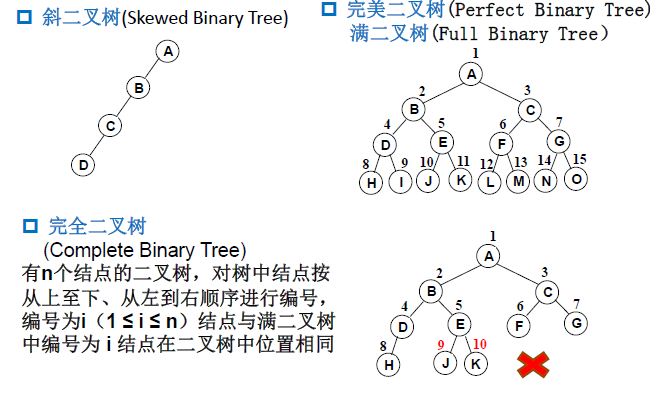
旋转45度后也就是二叉树。

一棵度为 m的树有n个节点。若每个节点直接用m个链指向相应的儿子，则表示这个树所需要的总空间是n\*(m+1) (假定每个链以及表示节点的数据域都是一个单位空间)。当采用儿子/兄弟（First Child/Next Sibling）表示法时，所需的总空间是：3n

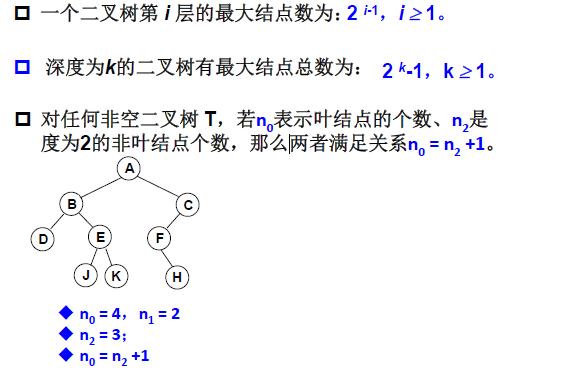
##### 3.2 二叉树及其存储结构

1. **二叉树基本概念**

**特殊二叉树：注意完全二叉树和完美二叉树的区别**

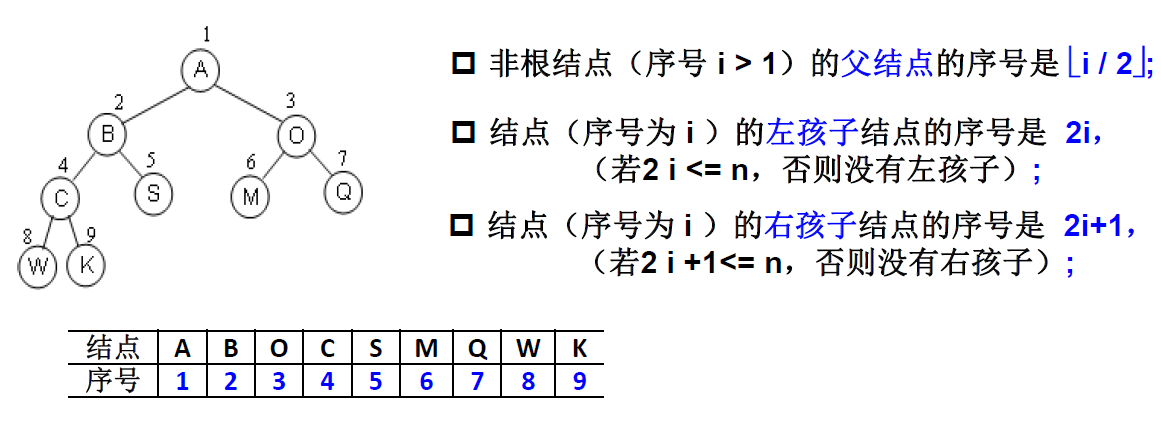


**二叉树的几个重要性质：**

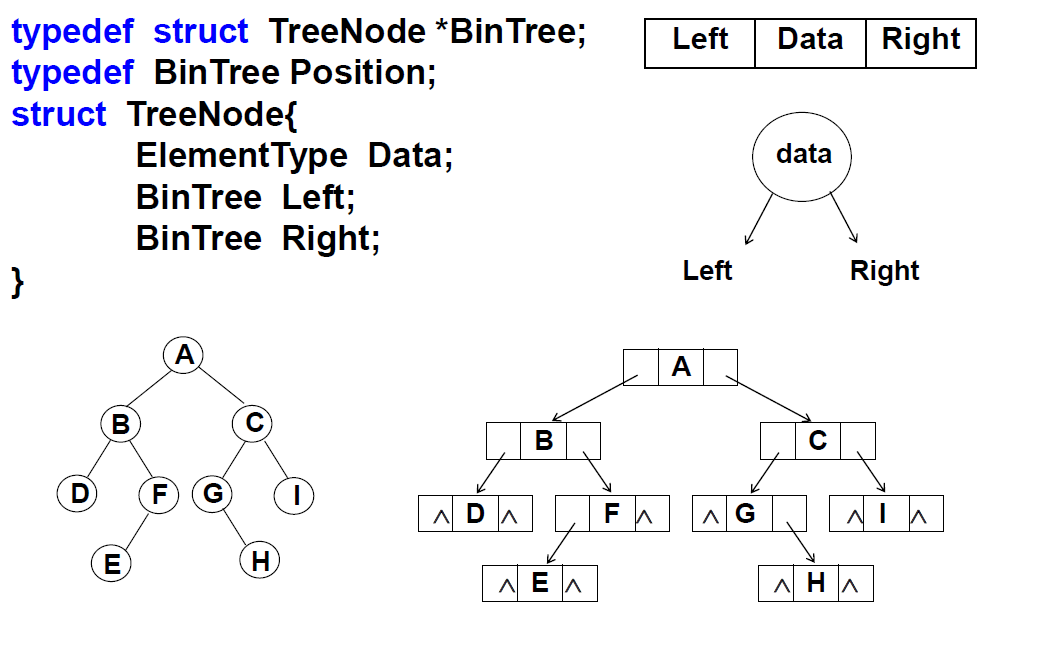


1. **二叉树的存储结构**

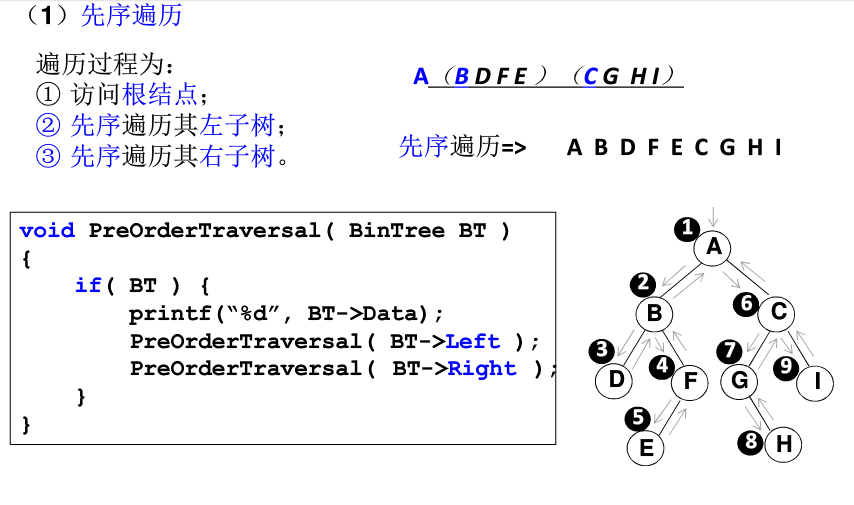
* **顺序存储结构**



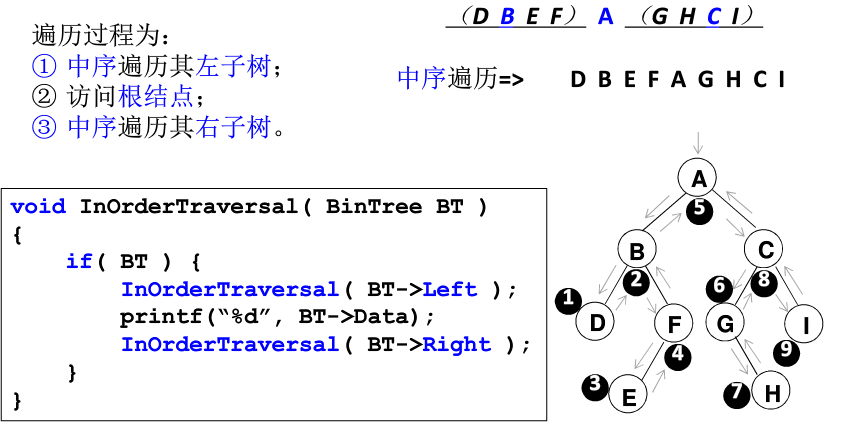
* **链表存储**



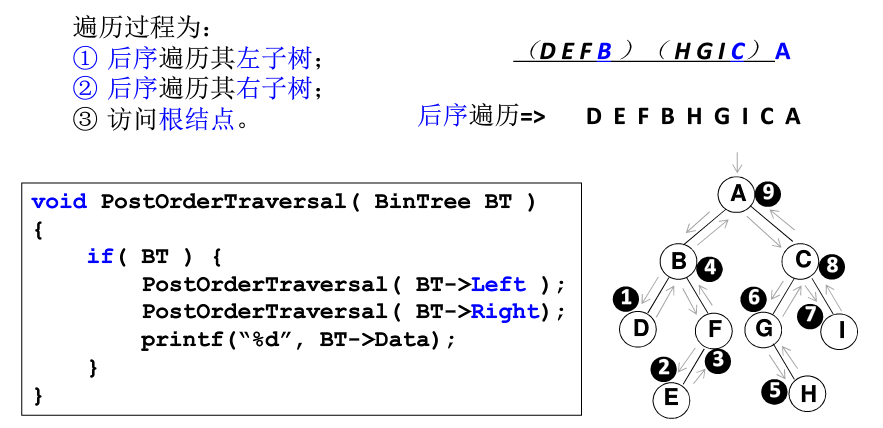
1. **二叉树的遍历**
2. **先序遍历**



1. **中序遍历**



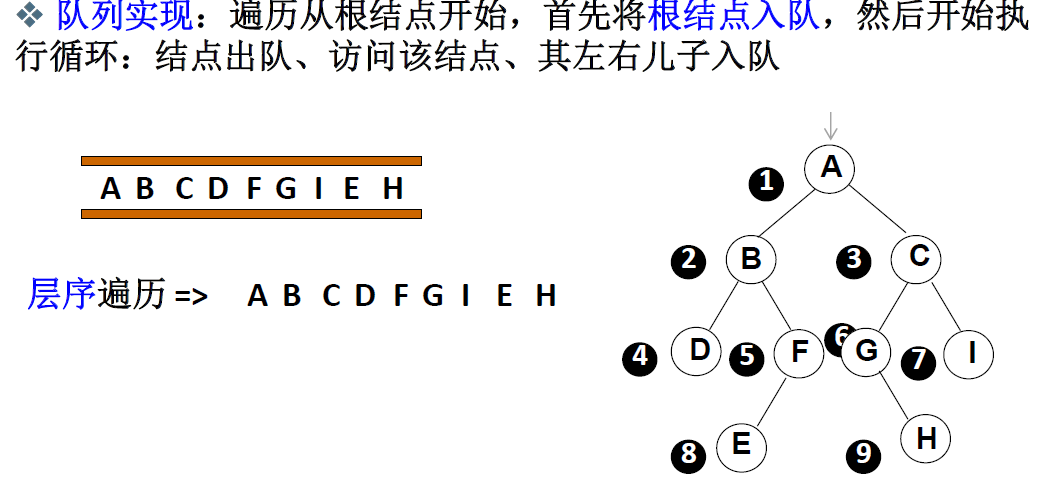
1. **后序遍历**

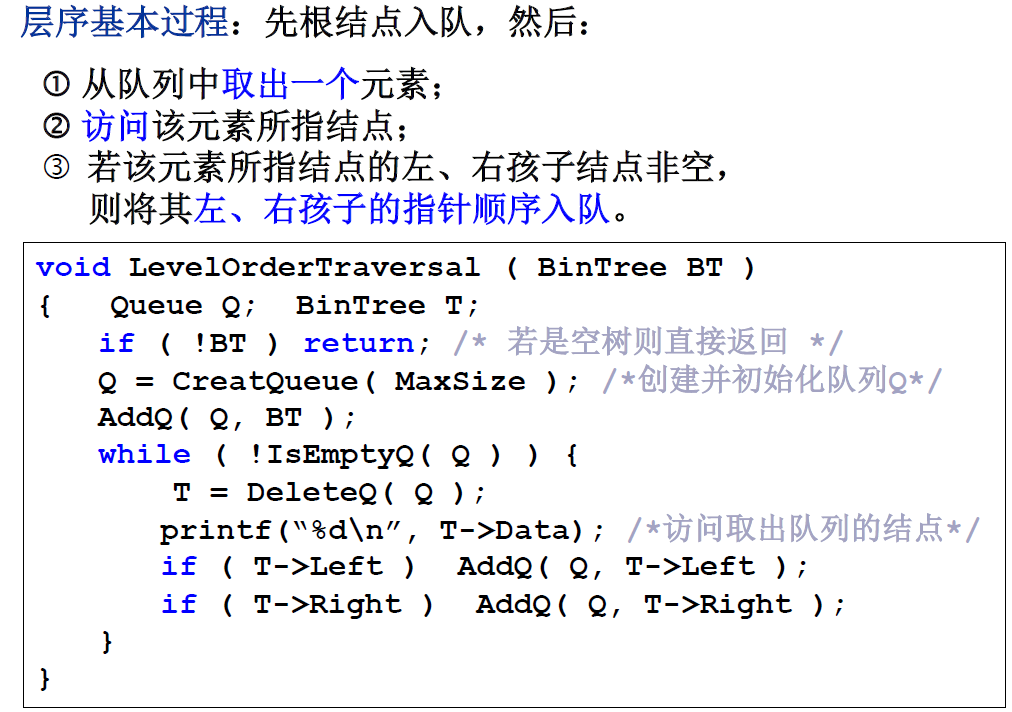


1. **层序遍历**

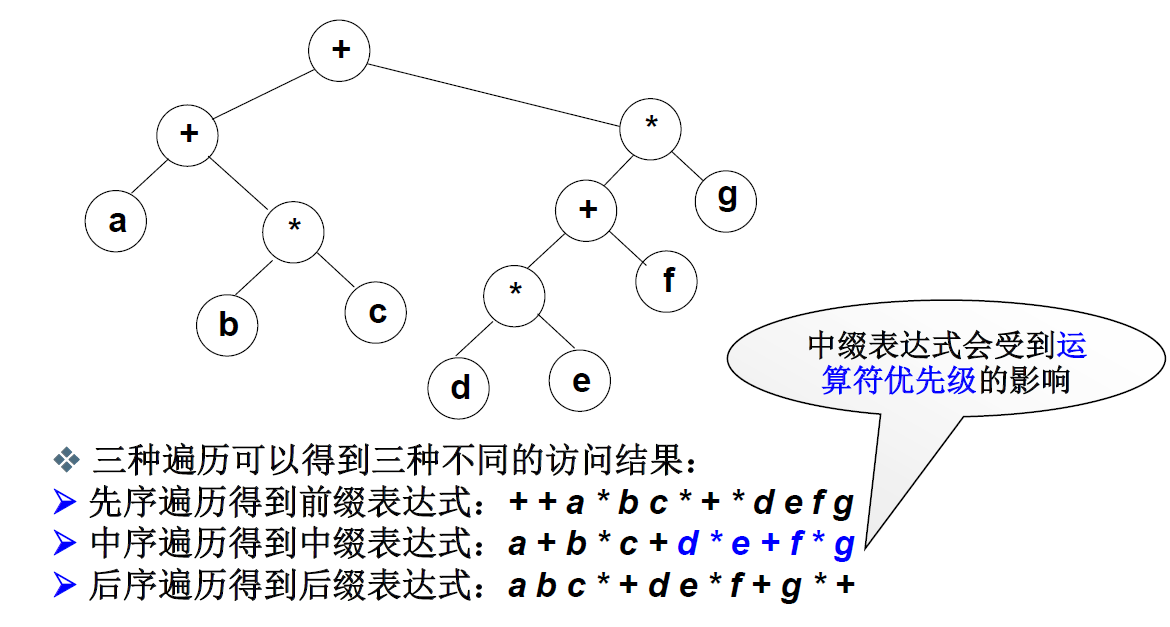
二叉树遍历的核心问题：二维结构的线性化 ；

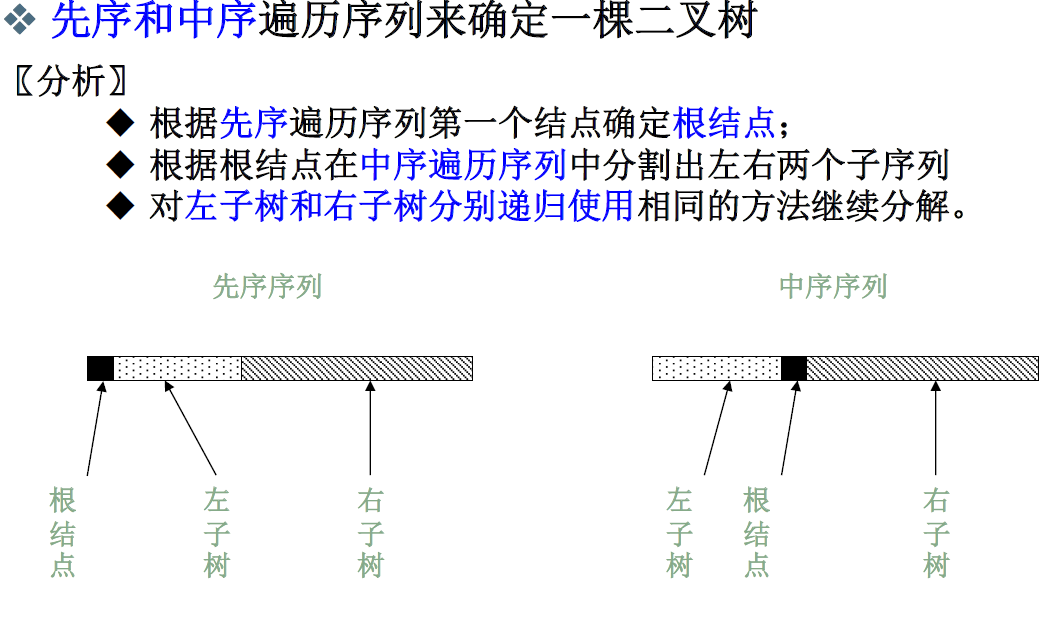
从结点访问其左、右儿子结点 访问左儿子后，右儿子结点怎么办？ 需要一个存储结构保存暂时不访问的结点 存储结构：堆栈、队列





遍历二叉树的应用：必须用先序+中序 或者后序+中序才可以确定一个二叉树

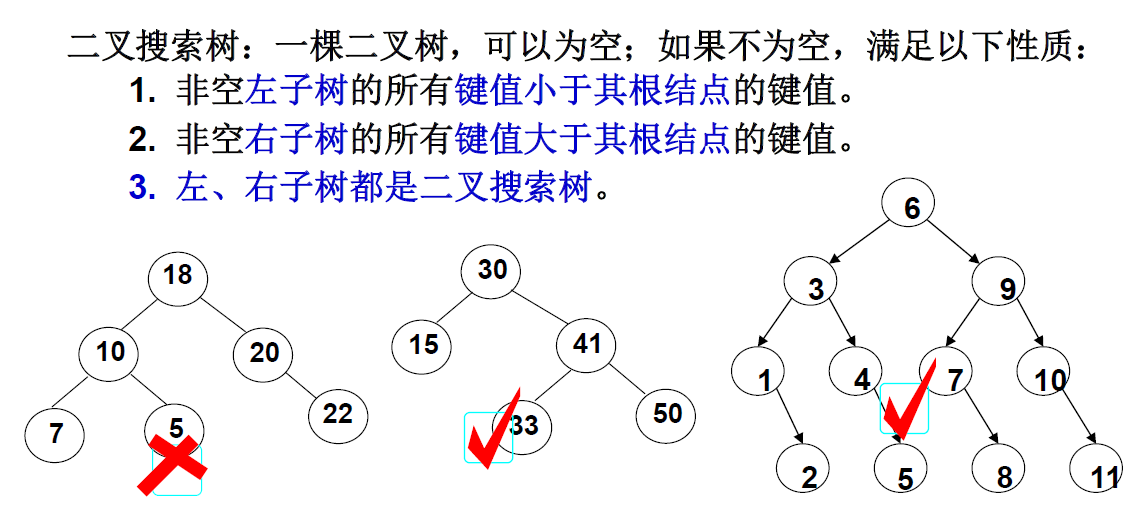




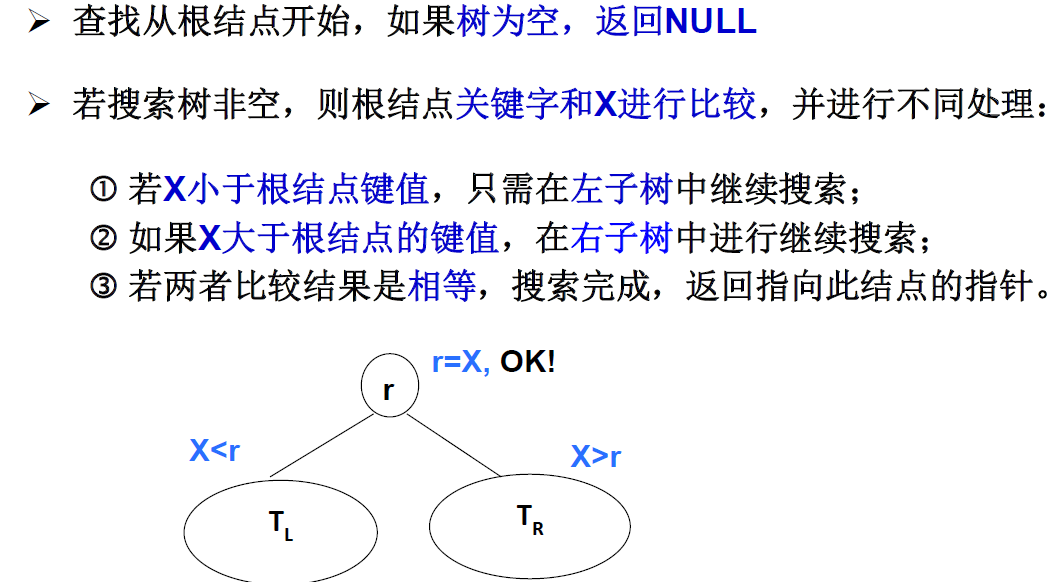
## 第四讲 树（中）

##### 4.1 二叉搜索树

1. **定义**



1. CRUD操作
2. **搜索**

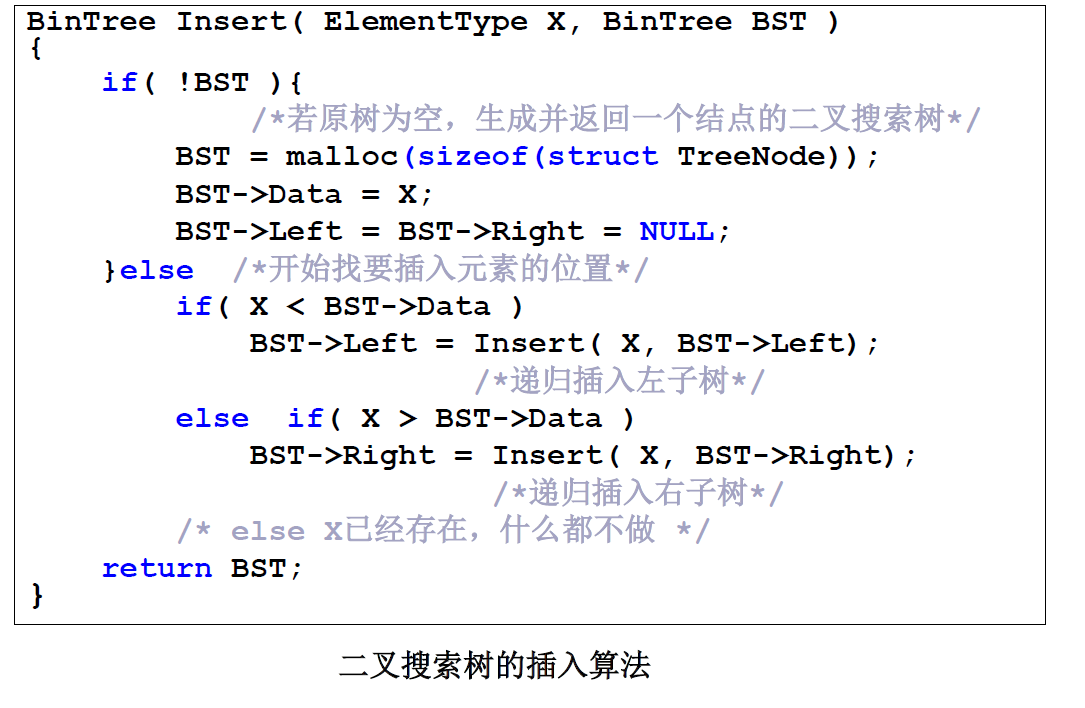


**最大和最小元素：**

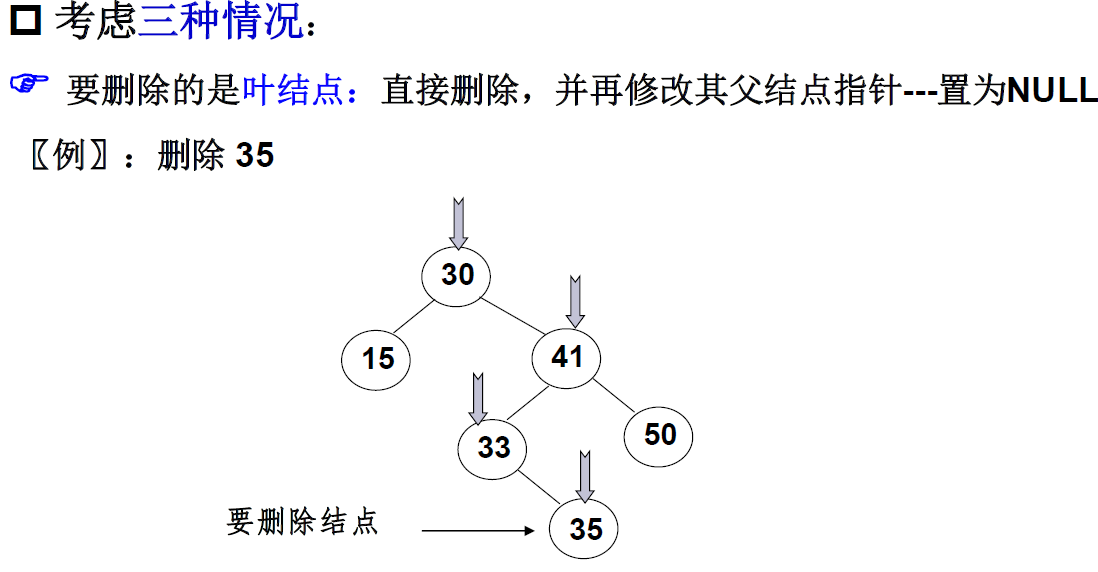
最大元素一定是在树的最右分枝的端结点上

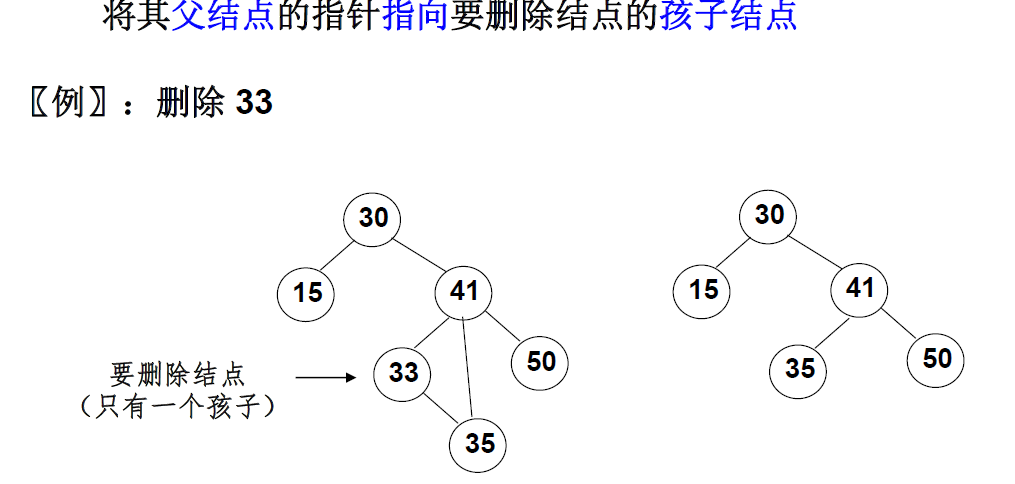
最小元素一定是在树的最左分枝的端结点上

1. **插入**

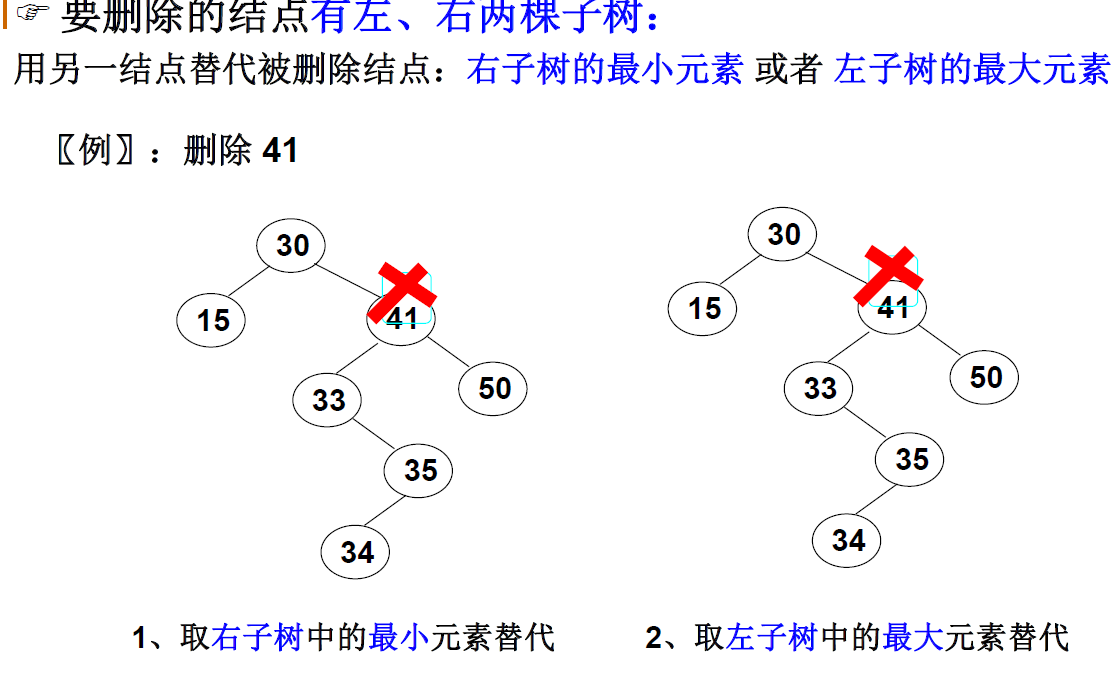


1. **删除 分三种情况，叶节点，只有一个儿子节点，两个儿子节点**





**下图：右子树的最小元素和左子树的最大元素，一定不是有两个子节点的节点**



若一搜索树（查找树）是一个有n个结点的完全二叉树，则该树的最大值一定在叶结点上 错误x

若一搜索树（查找树）是一个有n个结点的完全二叉树，则该树的最小值一定在叶结点上 正确

本讲义参照：

<http://mooc.study.163.com/learn/ZJU-1000033001?tid=1000044001#/learn/content?type=detail&id=1000112006&cid=1000102041>