第五章

1. 树的基本概念
2. 树的定义

树是n(n≥0)个节点的有限集。当n=0时，称为空树。在任意一棵非空树中应满足：①有且仅有一个特定的称为根的结点②当n>1时，根节点以外的节点可分为m(m>0)个互不相交的有限集，其中每个集合本身又是一棵树，并且称为根的子树。

1. 树的两个特点：①树的根节点没有前驱，除根节点外所有结点只有一个前驱；②树的所有结点可以有零个或多个后继；
2. 基本术语：

①根A到结点K的唯一路径上的任意结点，称为结点K的祖先，K是祖先结点的子孙结点；路径上最接近结点K的结点E称为K的双亲，而K为结点E的孩子；有相同双亲的结点称为兄弟；

②树中一个结点的孩子个数称为该**结点的度**，树中结点的最大度数称为**树的度**；

③度大于0的结点称为分支结点（非终端结点）：度为0的结点称为叶子结点（终端结点）。每个分支结点的分支数就是该结点的度；

④**结点的层次**从树根开始定义，根结点为第1层，它的子结点为第2层，以此类推；**结点的深度**是从根结点开始自顶向下逐层累加的；**结点的高度**是从叶结点开始自底向上逐层累加的；**树的高度**（或深度）是树中结点的最大层数；

⑤树中结点的各子树从左到右是有次序的，不能互换，称该树为**有序树**，否则称为无序树；

⑥树中两个结点之间的**路径**是由这两个结点之间所经过的结点序列构成的，而**路径长度**是路径上所经过的边的个数

1. 树的性质：

**①结点数等于总度数加1；**

**②度为的树第层最多有个结点（>1）；**

**③高度为的叉树至多有个结点；**

**④具有个结点的叉树的最小高度为；**

⑤度为m的树至少有一个节点度为m，m叉树最多只能有m个孩子，可以是空树；

⑥高度为h的m叉树至少有h个结点，高度为h，度为m的树至少有h+m-1个结点；

1. 二叉树的概念
2. 二叉树与度为2的有序树的区别：二叉树除了可以为空树外，虽然同为有序树，但二叉树无论是否有两个孩子都需要确定左右孩子，而度为2的树如果只有一个孩子无须区分左右；
3. 特殊的二叉树：

①**满二叉树**：一棵高度为h,且含有个结点的二叉树，即树中的每层都含有最多的结点；

②**完全二叉树**：n个结点的树，每个结点都与高度为h的满二又树中编号为1~n的结点一一对应；

③**二叉排序树**：左子树上所有结点的关键字均小于根结点的关键字，右子树上的所有结点的关键字均大于根结点的关键字，左子树和右子树又各是一棵二叉排序树；

④**平衡二叉树**：树上任一结点的左子树和右子树的深度之差不超过1；

1. 完全二叉树的性质

①结点的父母编号为；若,则结点为分支结点，否则为叶子结点；

②结点左孩子编号为，右孩子编号为；

③结点所在层次为;

④叶子结点只可能在层次最大的两层上出现；对于最大层次中的叶子结点，都依次排列在该层最左边的位置上；

⑤若有度为1的结点，则只可能有一个，且该结点只有左孩子而无右孩子；

⑥若n为奇数，则每个分支结点都有左孩子和右孩子；若n为偶数，则编号最大的分支结点只有左孩子，没有右孩子；

1. 二叉树的性质

**①非空二又树上的叶子结点数等于度为2的结点数加1,即(结点总数)；**

**②任意一棵树，若结点数量为n,则边的数量为n-1；**

**③非空二叉树上第k层上至多有个结点(k≥1)；**

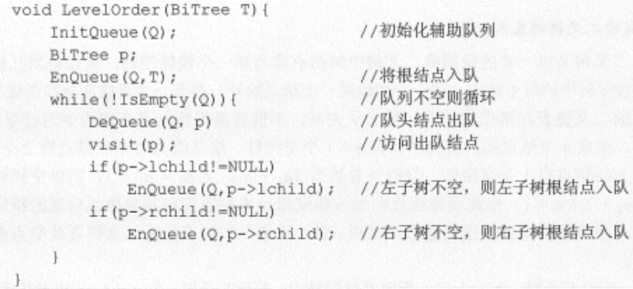
**④高度为h的二又树至多有个结点(h≥1)；**

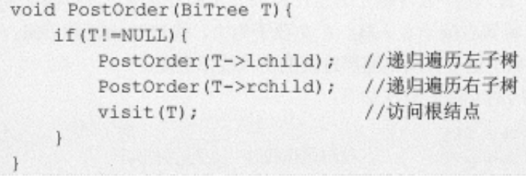
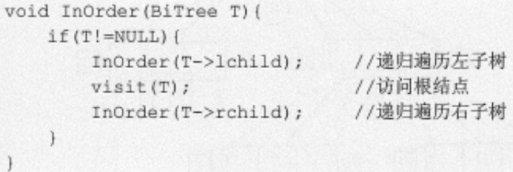
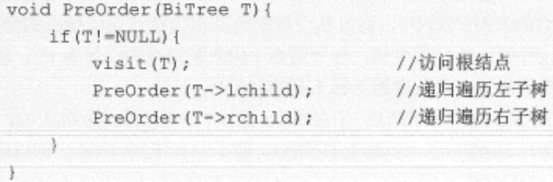
**⑤具有n个(n>0)结点的完全二又树的高度为或；**

1. 二叉树的存储结构

①顺序存储：将完全二叉树上编号为i的结点元素存储在一维数组下标为i-1的分量中（但建议从数组下标为1开始存储）；一般二叉树需要添加空结点，与完全二叉树对应，浪费空间；

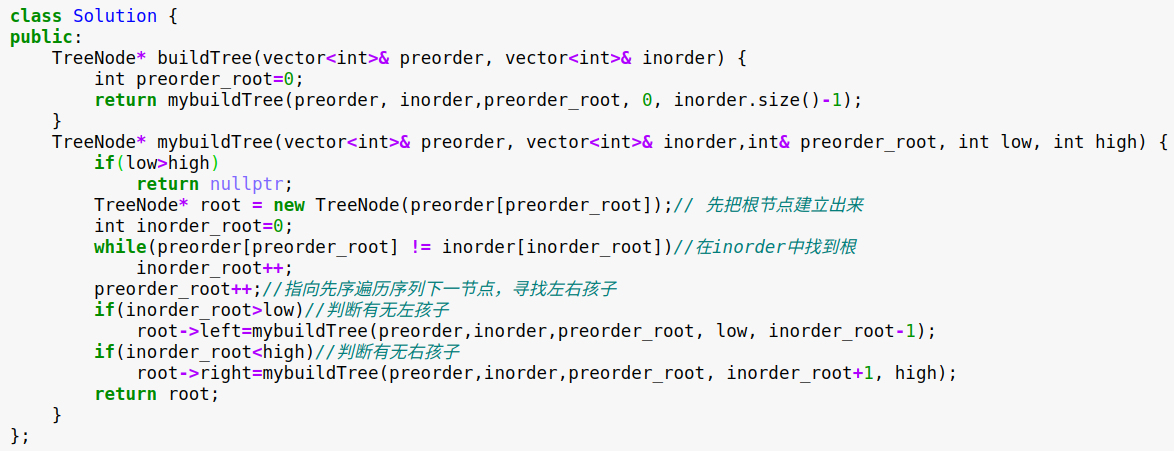
②链式存储：二叉链表至少包含三个域（数据域、左指针域、右指针域），增加指向父结点的指针后，变成三叉链表；在含有n个结点的二叉链表中，含有n+1个空链域；

1. 二叉树的遍历和线索二叉树
2. 遍历的概念：按照某种次序将所有结点访问一遍；
3. ①**先序遍历**（先根遍历）：若二叉树为空，则什么也不做；否则，访问根节点，先序遍历左子树，先序遍历右子树；②**中序遍历**（中根遍历）：若二叉树为空，则什么也不做；否则，中序遍历左子树，访问根节点，中序遍历右子树；③**后序遍历**（后根遍历）：若二叉树为空，则什么也不做；否则，后序遍历左子树，后序遍历右子树，访问根节点；④时间复杂度为；
4. 二叉树的层次遍历：①初始化一个辅助队列；②将根节点入队；③若队列非空，则将队头结点出队，访问该节点，并将其左右孩子插入队尾；④重复③直到队列为空；（时间复杂度为）



1. 由遍历序列构造二叉树：①前序+中序；②后序+中序；③层序+中序；④其中，前、后和层序用于定位根的位置，中序用于确定左右孩子；⑤算法思想：

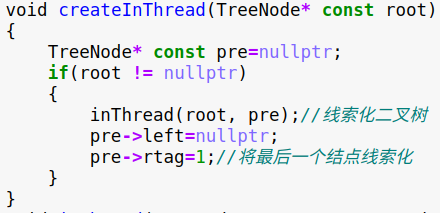
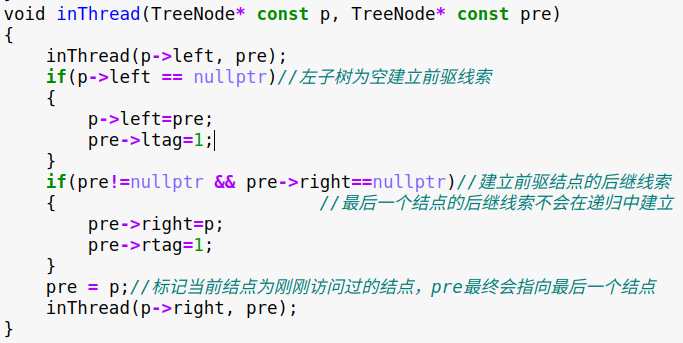
前序遍历的形式总是（即根节点总是前序遍历中的第一个节点）

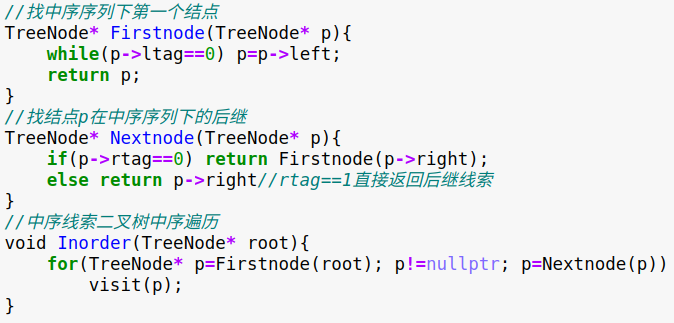
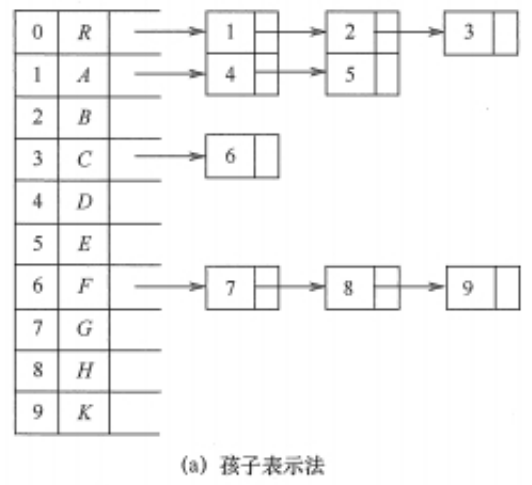
[ 根节点, [左子树的前序遍历结果], [右子树的前序遍历结果] ]

而中序遍历的形式总是

[ [左子树的中序遍历结果], 根节点, [右子树的中序遍历结果] ]

只要我们在中序遍历中定位到根节点，那么我们就可以分别知道左子树和右子树中的节点数目。由于同一颗子树的前序遍历和中序遍历的长度显然是相同的，因此我们就可以对应到前序遍历的结果中，对上述形式中的所有左右括号进行定位。



1. 线索二叉树：①在含有n个结点的二叉树中，有n+1个空指针（推）；②规定：若无左子树，令lchild指向其前驱结点，若无右子树，令rchild指向其后继结点，增加ltag和rtag标志指针域指向左右孩子还是前驱后继；
2. 二叉树线索化：①线索化的实质是（前中后序）遍历一次二叉树，设pre指向刚刚访问过的结点（初始为nullptr），指针p指向正在访问的结点，即pre指向p的前驱；②中序遍历过程中，检查p的左指针为空则指向pre，检查pre的右指针为空就指向p，pre最终指向中序遍历最后一个结点，需要处理pre的右孩子，因此是全局变量；③先序线索化过程中，先处理根节点，若没有左孩子会将其线索化，之后访问其左孩子，由于已经线索化因此会陷入循环，需要加ltag验证；
3. 中序线索二叉树的遍历：①后继一定是其右孩子的左孩子的左孩子……[ 左, 根, [[**左**, 根, 右], 根, 右] ]；②遍历时要先找到第一个结点，依次找后继，直至后继为空；③后继规律：若rtag为1则右链指向后继，否则遍历右子树找到最左下结点为其后继；④思考寻找前驱的方法和逆序遍历；
4. 先序线索二叉树的遍历：①rtag为0结点有左孩子时，后继就是其左孩子[ 根, [ **根**, 左, 右], 右]；②rtag为0结点没有左孩子，一定有右孩子，后继一定是其右孩子[ 根, 右] → [ 根, [ **根**, 左, 右]]；③先序线索二叉树（无线索时）找不到先序前驱，但时含有指向父结点指针的三叉链表可以，其前驱有四种情况：i若为左孩子则前驱为父结点，ii若为右孩子且父结点没有左孩子则前驱为父结点，iii若为右孩子且父结点有左孩子则前驱为父结点的左孩子优先向右下找否则向左直到叶子结点，iv若为根节点则无前驱；
5. 先序线索二叉树的遍历：后续线索二叉树（无线索时）找不到后序后继，但时含有指向父结点指针的三叉链表可以，其后继有四种情况：i若为右孩子则后继为父结点，ii若为左孩子且父结点没有右孩子则后继为父结点，iii若为左孩子且父结点有右孩子则后继为父结点的右孩子优先向左下找否则向右直到叶子结点，iv若为根节点则无后继；
6. 树、森林
7. 双亲表示法：①一数组存储每个结点，每个结点有一个伪指针表明父结点在数组中的位置；②优点：方便找父结点，缺点：找子结点要遍历；③树的顺序存储结构中，数组下标代表结点编号，内容表示结点之间关系，二叉树的顺序存储结构中，数组下标代表结点编号和结点之间关系；
8. 孩子表示法：①一数组存储每个结点，每个结点有一个指向链表的头指针，每个链表结点存储对应数组结点的孩子的位置以及指向下一个链表结点的指针；②优点：方便找子结点，缺点：找父结点要遍历n结点中的链表；
9. 孩子兄弟表示法
10. 树、森林与二叉树的转换
11. 树和森林的遍历
12. 树与二叉树的应用