二叉树

**1、二叉树是n(n>=0)个结点的有限集合**。该集合或者为空集(空二叉树)，或者由一个根结点和两棵互不相交的、分别称为根结点的左子树和右子树的二叉树组成。

**2、二叉树的特点**

每个结点最多有两棵子树，所以二叉树中不存在度大于2的结点。二叉树中每个节点都是一个对象，每个数据节点都有三个指针，分别指向父母、左孩子和右孩子的指针。每个节点都是通过指针相互连接的。相连指针的关系都是父子关系。

**3、二叉树节点的定义**

struct BinaryTreeNode{

int m\_nValue;

BinaryTreeNode\* m\_pLeft;

BinaryTreeNode\* m\_pRight;

}

**4、二叉树的五种基本形态**

空二叉树

只要一个根结点

根结点只有一个左子树

根结点只有一个右子树

根结点既有左子树又有右子树

**5、特殊二叉树**

满二叉树：在一棵二叉树中，如果所有分支结点都存在左子树和右子树，并且左右叶子都在同一层，即满二叉树【深度为k且含有2k-1个结点的二叉树。】

完全二叉树：最后一层左边是满的，右边可能满也可能不满，然后其余层都是满的。满二叉树也是完全二叉树。

完全二叉树的特点：叶子结点只能出现在最下面两层；最下层的叶子一定集中在左部连续位置；倒数第二层，若有叶子结点，一定都在右部连续位置；如果结点度为1，则该结点只有左孩子；同样结点树的二叉树，完全二叉树的深度最小。

完全二叉树的算法实现：

bool is\_complete(tree \*root){

queue q;

tree \*ptr;

// 进行广度优先遍历（层次遍历），并把NULL节点也放入队列

q.push(root);

while((ptr = q.pop()) != NULL)){

q.push(ptr->left);

q.push(ptr->right);

}

// 判断是否还有未被访问到的节点

while(!q.is\_empty()){

ptr = q.pop();

// 有未访问到的非NULL节点，则树存在空洞，为非完全二叉树

if(ptr != null){

return false;

}

}

return true;

}

**6、二叉树的性质**

性质一：在二叉树的第i层上至多有2^(i-1)个结点(i>=1)

性质二：深度为k的二叉树至多有2^(k) -1个结点(k>=1)

**7、二叉树的存储结构**

二叉树的顺序存储结构就是一堆数组存储二叉树的各个结点，并且结点的存储位置能体现结点之间的逻辑关系。

二叉链表(链式存储结构)就是一个数据域data和两个指针lchild、 rchild

**8、二叉树遍历**

（1）前序遍历

首先访问根结点，然后遍历左子树，最后遍历右子树。即 根-左-右

function preOrder(node){

if(node != null){

data.push(node);

preOrder(node.left);

preOrder(node.right);

}

}

（2）中序遍历

首先遍历左子树，然后访问根结点，最后遍历右子树。即 左-根-右

function inOrder(node){

if(node != null){

inOrder(node.left);

data.push(node);

inOrder(node.right);

}

}

（3）后序遍历

首先遍历左子树，然后遍历右子树，最后访问根结点。即 左-右-根

function postOrder(node){

if(node != null){

postOrder(node.left);

postOrder(node.right);

data.push(node);

}

}

**9、实现二叉查找树**

二叉查找树BST由节点组成，定义Node节点对象如下：

function Node(data,left,right){

this.data =data;

this.left = left;

this.right =right;

this.show =show;

function show(){

return this.data;

}

}

**10、查找最大值和最小值**

（1）查找最小值

二叉查找树，因为较小的值在左子节点上，在BST上查找最小值，只需遍历左子树，直至找到最后一个节点。

function getMin(){

var current = this.root;

while(current.left != null){

current = current.left;

}

return current.data;

}

（2）查找最大值

二叉查找树，只需遍历右子树，直至最后一个节点，该节点保存的值就是最大值。

function getMax(){

var current = this.root;

if(current.right != null){

current = current.right;

}

return current.data;

}

**二叉树的题目列表**

[**1. 求二叉树中的节点个数**](http://blog.csdn.net/luckyxiaoqiang/article/details/7518888#topic1)  
[**2. 求二叉树的深度**](http://blog.csdn.net/luckyxiaoqiang/article/details/7518888#topic2)[**3. 前序遍历，中序遍历，后序遍历**](http://blog.csdn.net/luckyxiaoqiang/article/details/7518888#topic3)[**4.分层遍历二叉树（按层次从上往下，从左往右）**](http://blog.csdn.net/luckyxiaoqiang/article/details/7518888#topic4)[**5. 将二叉查找树变为有序的双向链表**](http://blog.csdn.net/luckyxiaoqiang/article/details/7518888#topic5)[**6. 求二叉树第K层的节点个数**](http://blog.csdn.net/luckyxiaoqiang/article/details/7518888#topic6)[**7. 求二叉树中叶子节点的个数**](http://blog.csdn.net/luckyxiaoqiang/article/details/7518888#topic7)[**8. 判断两棵二叉树是否结构相同**](http://blog.csdn.net/luckyxiaoqiang/article/details/7518888#topic8)[**9. 判断二叉树是不是平衡二叉树**](http://blog.csdn.net/luckyxiaoqiang/article/details/7518888#topic9)[**10. 求二叉树的镜像**](http://blog.csdn.net/luckyxiaoqiang/article/details/7518888#topic10)[**11. 求二叉树中两个节点的最低公共祖先节点**](http://blog.csdn.net/luckyxiaoqiang/article/details/7518888#topic11)[**12. 求二叉树中节点的最大距离**](http://blog.csdn.net/luckyxiaoqiang/article/details/7518888#topic12)[**13. 由前序遍历序列和中序遍历序列重建二叉树**](http://blog.csdn.net/luckyxiaoqiang/article/details/7518888#topic13)[**14.判断二叉树是不是完全二叉树**](http://blog.csdn.net/luckyxiaoqiang/article/details/7518888#topic14)