# 树微课Part1

许

# 基础知识-什么是树

- 树是n(n>=0)个节点的有限集
- 在任意一个非空树中：
  - 有且仅有一个根
  - 当n>1时，其余节点可以分为M个互不相交的有限集T1 T2 T3..Tm，其中每一个集合本身又是一棵树，并称为根的子树
- 树的结构定义是一种递归定义
- n=0 空树
- n=1 只有根

# 基础知识-基本定义

- 根 Root
- 结点 Node
- 度 Degree　　结点的儿子个数
- 叶子 Leaf　　度为0的结点
- 层次 Level　　根结点层次为1
- 深度 Depth　　结点的最大层次
- 森林 Forest　　m(m>=0)颗不会相交的树的集合

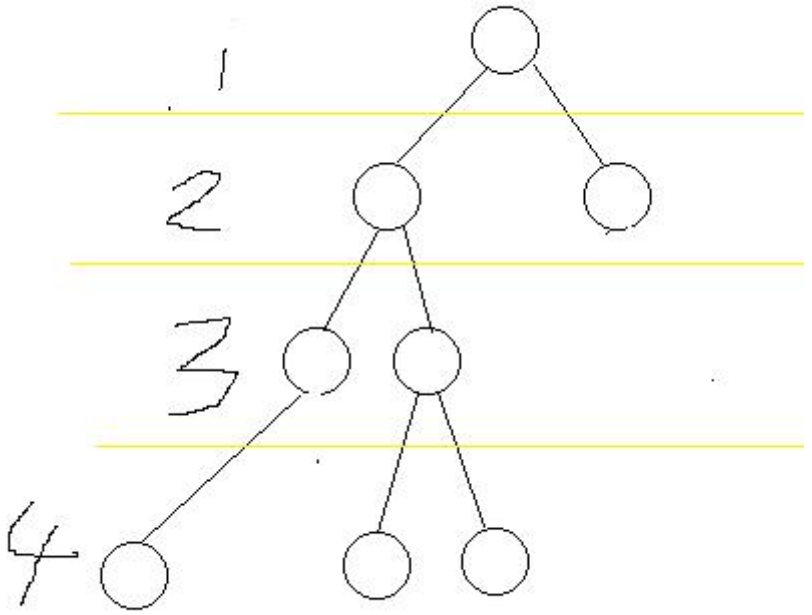
# 二叉树-定义

- 树
- 结点度<=2
- 儿子有序 左儿子 右儿子

```
//Definition for a binary tree node.
struct TreeNode {
    int val;
    TreeNode *left;
    TreeNode *right;
    TreeNode(int x) : val(x), left(NULL),
                                right(NULL) {}
}
```

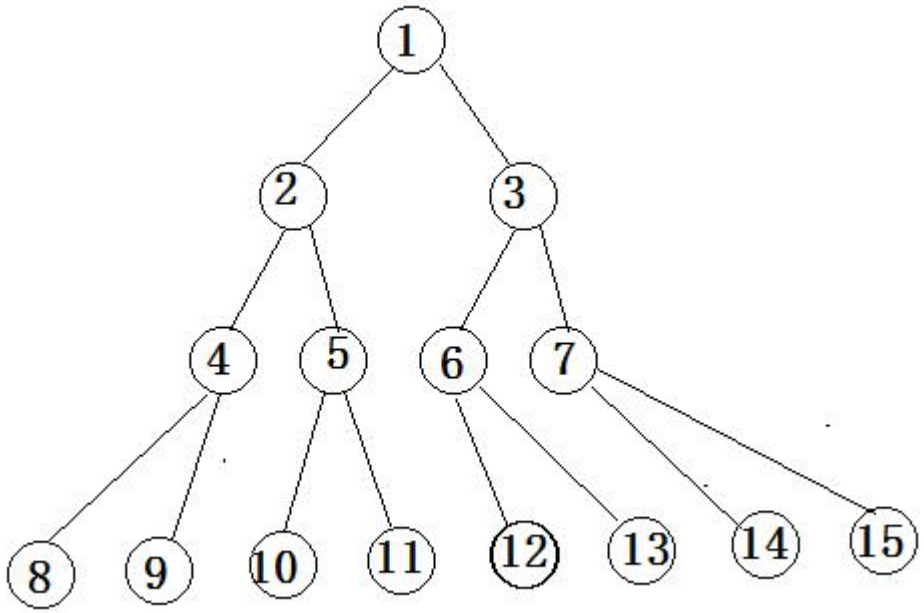
# 二叉树-基本性质

- 第level层的结点个数最多为： $2^{level-1}(level >= 1)$
- 深度为h的二叉树最多结点个数： $2^h - 1$



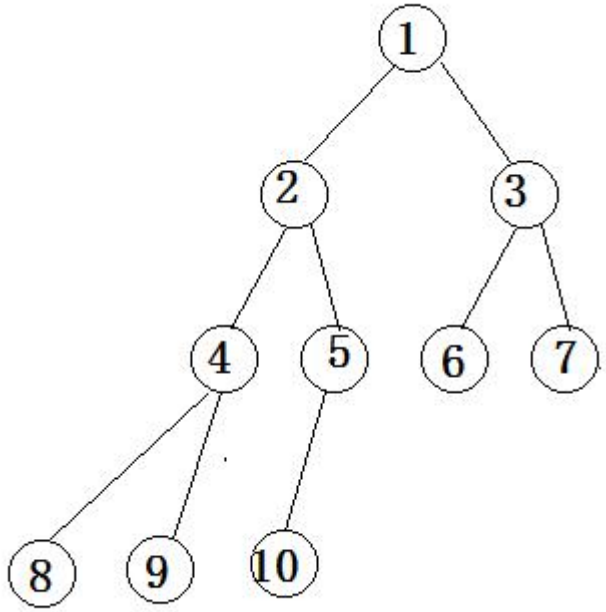
# 二叉树-满二叉树

- 结点个数= $2^h - 1$
- 结点层次= $\lfloor \log_2 i \rfloor + 1$
- 父结点
  - i=1 根 无父节点
  - i<>1 父节点= $\lfloor i/2 \rfloor$



# 二叉树-完全二叉树

- 具有满二叉树大部分性质
- 仅有最后一层缺失部分结点
- 满二叉树属于完全二叉树
- 可以用数组表示
- 不需要记录父结点与子节点



# 二叉树-树的简单问题

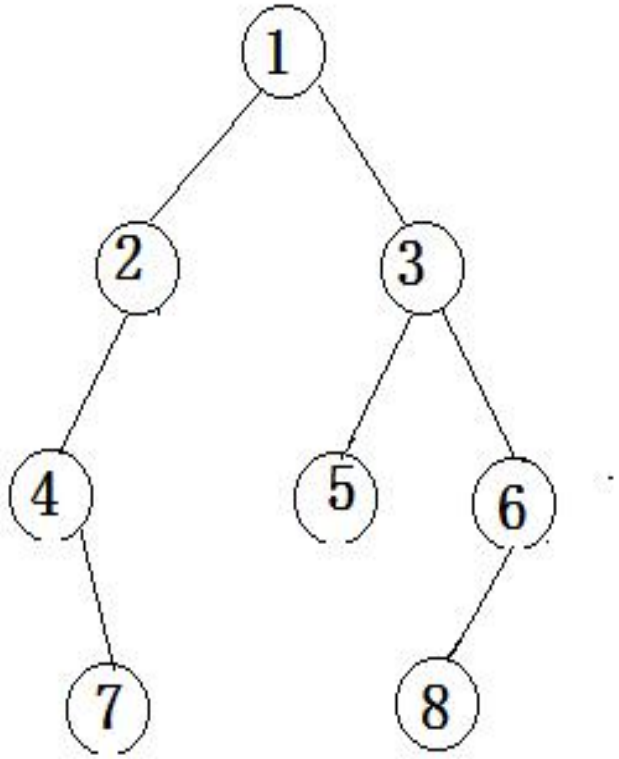
- TYPE FUNCTION(TreeNode * r)
- {
  - if( r==NUL L ) return ***;
  - ****
  - if(r->left!=NULL) FUNCTION(r->left);
  - if(r->right!=NULL) FUNCTION(r->right);
  - ****
- }

# 二叉树-简单问题示例

- LeetCode 100 SameTree 判断2个树是否相同
- LeetCode 101 Symmetric Tree 判断1个树是否左右映射
- LeetCode 236 求2个结点的LCA
- LeetCode 111 Minimum Depth of Binary Tree 最小高度叶子
- LeetCode 112. Path Sum 树的深度遍历

# 二叉树-树的遍历

- 前序遍历　　{1,2,4,7,3,5,6,8} LeetCode144
- 中序遍历　　{4,7,2,1,5,3,8,6} LeetCode94
- 后序遍历　　{7,4,2,5,8,6,3,1}



# 二叉树-前序遍历

```
void PreOrder(TreeNode * r)
{
    if (r != NULL)
    {
        cout << r->val << " ";
        PreOrder(r->left);
        PreOrder(r->right);
    }
}
```

# 二叉树-前序遍历非递归

- 使用栈来辅助
- 右儿子先进入
- 左儿子后进入

```
void PreOrder2(TreeNode * r)
{
    stack<TreeNode *> s;
    s.push(r);
    while (!s.empty())
    {
        r = s.top();
        s.pop();
        if (r != NULL)
        {
            cout << r->val << " ";
            s.push(r->right);
            s.push(r->left);
        }
    }
}
```

# 二叉树-中序遍历非递归

- 有左儿子 则自己近栈，R=左儿子
- 栈顶被弹出 访问当前节点 R=右儿子

```
void InOrder2(TreeNode * r)
{
    stack<TreeNode *> s;
    while (!s.empty() || r!=NULL )
    {
        while (r != NULL)
        {
            s.push(r);
            r = r->left;
        }
        if (!s.empty())
        {
            r = s.top();
            cout << r->val << " ";
            s.pop();
            r = r->right;
        }
    }
}
```

# 二叉树-后序遍历非递归

- 结点会出现在栈顶2次
- 有左儿子 则自己近栈，R=左儿子
- 第一次出现 r=右儿子
- 第二次出现 访问结点自己
- visit(left) * visit(right) * 访问自己

```
struct StackNode {
    TreeNode * r;
    bool isFirst;
};
```

```
void PostOrder2(TreeNode * r)
{
    stack<StackNode> s;
    StackNode snode;
    while (!s.empty() || r != NULL)
    {
        while (r != NULL)
        {
            snode.isFirst = true;
            snode.r = r;
            s.push(snode);
            r = r->left;
        }
        if (!s.empty())
        {
            snode = s.top();
            s.pop();

            if (snode.isFirst)
            {
                snode.isFirst = false;
                s.push(snode);
                r = snode.r->right;
            }
            else
            {
                cout << snode.r->val << " ";
                r = NULL;
            }
        }
    }
}
```

# 二叉树的遍历

- 递归-》非递归
- 模拟栈
- 注意状态
  - if(r==NULL)
  - 0 visit(left)　　visit(left)
  - 1 print　　visit(right)
  - 2 visit(right)　　print

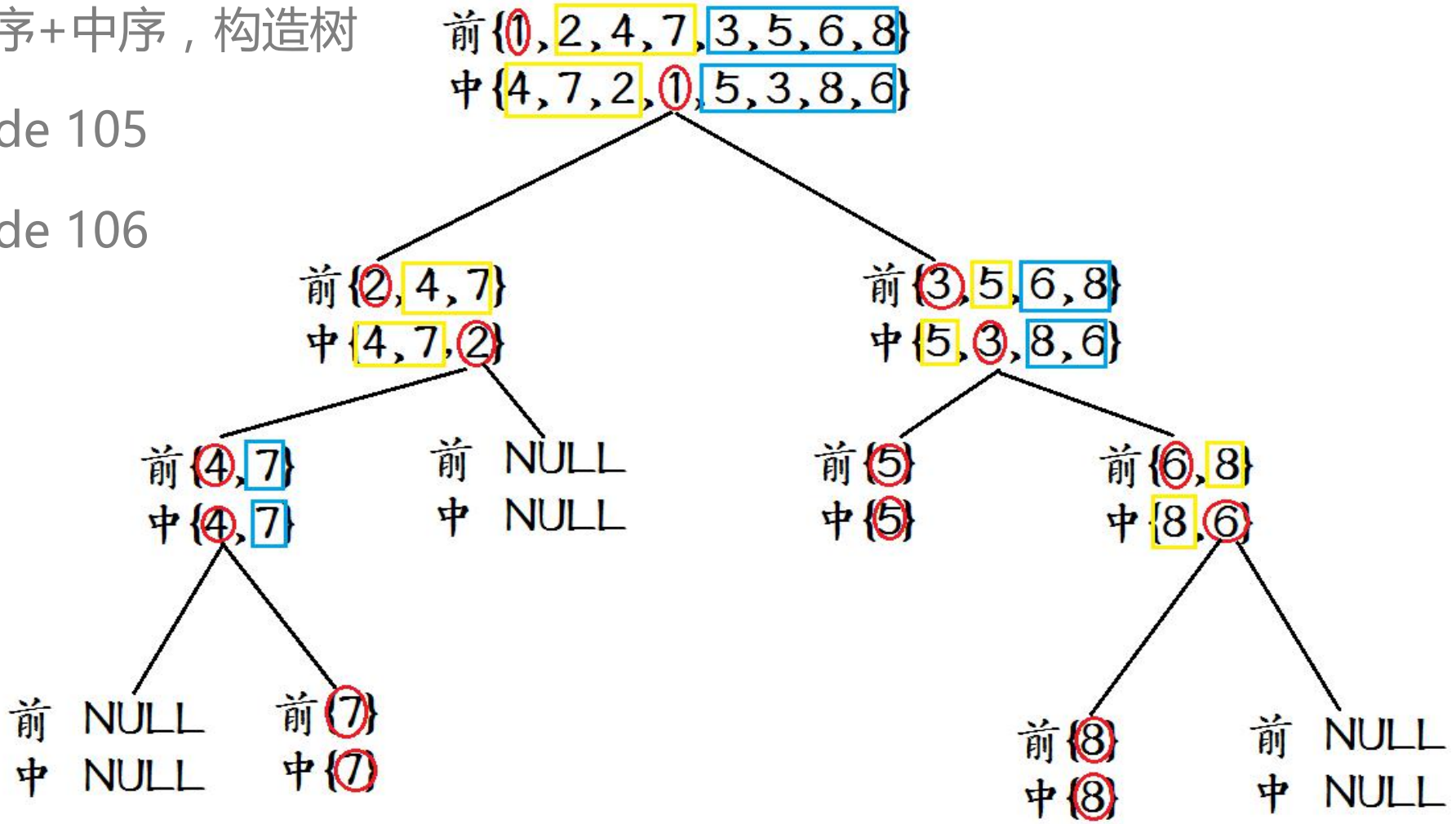
```
void InOrder3(TreeNode * r)
{
    StackNode1 snode;
    snode.r = r;
    snode.state = 0;
    stack<StackNode1> s;
    while (snode.r!=NULL || !s.empty())
    {
        while (!s.empty()&&(snode.r == NULL || snode.state>=3 ))
        {
            snode = s.top();
            s.pop();
            snode.state++;
        }
        if (snode.r == NULL || snode.state >= 3)
        {
            break;
        }
        switch ( snode.state )
        {
            case 0:
                s.push(snode);
                snode.r = snode.r->left;
                break;
            case 1:
                cout << snode.r->val << " ";
                snode.state++;
                break;
            case 2:
                s.push(snode);
                snode.r = snode.r->right;
                snode.state = 0;
                break;
            default:
                break;
        }
    }
```

# 二叉树的遍历

- LeetCode 144 前序
- LeetCode  94 中序
- LeetCode 145 后续 非递归

# 二叉树-树的遍历

- 已知前序+中序，构造树
- LeetCode 105
- LeetCode 106



# 二叉查找树

- r->left->val 小于 r->val
- r->right->val 大于 r->val
- 在树中查询是否存在val==x的node的代价是O(h)
- h是树的深度
- h最坏情况下等于n 所以需要平衡

# 二叉查找树

- TreeNode * TreeSearch(TreeNode * r ,int target){
  - if( r==NULL) return NULL;
  - if(r->val==target){
    - return r;
  - }
  - if(r->val>target){
    - return TreeSearch(r->left,target);
  - }else{
    - return TreeSearch(r->right,target)
  - }
- }

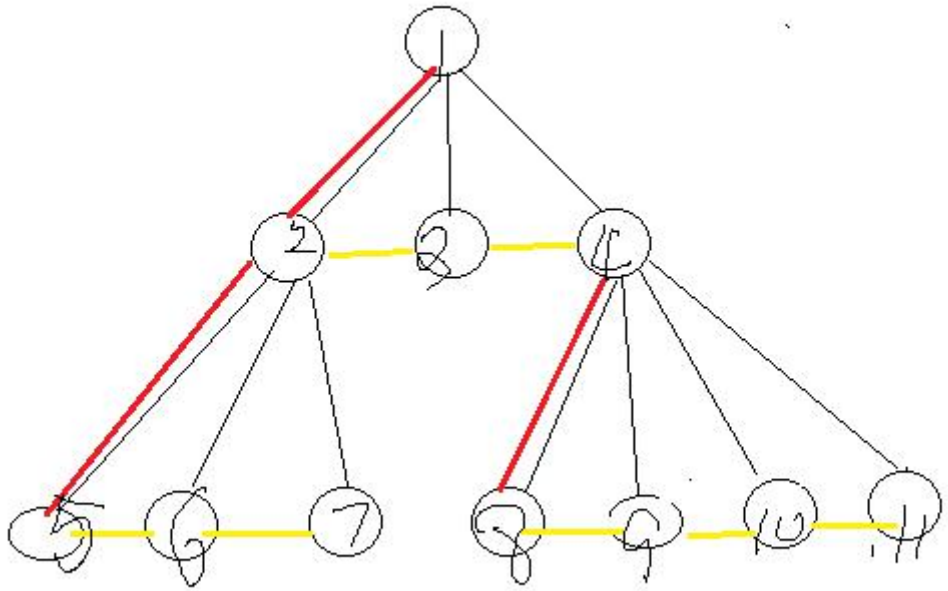
# 二叉查找树

- LeetCode 230 Kth Smallest Element in a BST 二叉树中的第k大数
- 数组查第K大 [...x..] O(n) 二分查一半
- 左子树结点个数 c c2 c+c2+1
- k<=c 在子问题已经解决
- k-c==1 根
- k-c>1 在右子树求子问题k=k-c-1;
- LeetCode 108 Convert Sorted Array to Binary Search Tree 有序数组->二叉树

# 左儿子右兄弟表示法

- 一颗非二叉树的存储方式
- 可以转换为左儿子右兄弟表示法

```
struct TreeNode
{           int val;
            TreeNode * firstchild;
            TreeNode * next;
}
```



```
struct TreeNode
{    int val;      int childst;    int childnum;   }       vector<TreeNode*> childlist;
```

childlist[ r->childst+i -1]  第 i 个儿子

# 总结

- 树的问题是典型的递归问题，可以转换成解决子树的问题
- 注意空树，单结点树，子节点为NULL的边界情况
- 简单问题: 注意细节，Show Me The Code
- 用递归理清思路
- 数据量较大时 使用非递归写法

- 微博：小小喵78
- Q&A