个人资料

WalkingInTheWind

访问： 258788次
积分： 3179
等级：
排名： 第5442名

原创： 49篇 转载： 7篇
译文： 0篇 243条

文章搜索

文章分类

算法与数据结构 (30)
c/c++ (8)
VC/MFC (2)
操作系统 (3)
网络 (1)
ACM (5)
数据库 (1)
笔试与面试 (23)
LeetCode (0)

文章存档

2014年03月 (1)
2013年10月 (1)
2013年05月 (13)
2013年04月 (7)
2013年03月 (4)

展开

阅读排行

轻松搞定面试中的二叉树 (88474)
轻松搞定面试中的链表题 (21721)
2014年计算机求职总结-- (19988)
深入理解C/C++数组和指针 (9105)
用位运算实现两个整数的 (8821)

轻松搞定面试中的二叉树题目

分类： 算法与数据结构 笔试与面试 2012-08-29 21:24 88550人阅读 评论(53) 收藏 举报

二叉树 面试

树是一种比较重要的数据结构，尤其是二叉树。二叉树是一种特殊的树，在二叉树中每个节点最多有两个子节点，一般称为左子节点和右子节点（或左孩子和右孩子），并且二叉树的子树有左右之分，其次序不能任意颠倒。二叉树是递归定义的，因此，与二叉树有关的题目基本都可以用递归思想解决，当然有些题目非递归解法也应该掌握，如非递归遍历节点等等。本文努力对二叉树相关题目做一个较全的整理总结，希望对找工作的同学有所帮助。

二叉树节点定义如下：

```
struct BinaryTreeNode
{
    int m_nValue;
    BinaryTreeNode* m_pLeft;
    BinaryTreeNode* m_pRight;
};
```

相关链接：

轻松搞定面试中的链表题目

题目列表：

详细解答

1. 求二叉树中的节点个数

递归解法：

（1）如果二叉树为空，

（2）如果二叉树不为空， 数 = 左子树节点个数 + 右子树节点个数 + 1

参考代码如下：

二分查找，你真的掌握了 (7177)
2014年计算机求职总结-- (5154)
求能整除正整数a或b的正 (4707)
A*路径搜寻算法 (4139)
【谷歌面试题】找出二叉 (4067)

评论排行

2014年计算机求职总结-- (65)
轻松搞定面试中的二叉树 (53)
轻松搞定面试中的链表题 (33)
深入理解C/C++数组和指针 (29)
二分查找，你真的掌握了 (11)
经典面试题：链表的相交 (6)
2014年计算机求职总结-- (6)
函数调用约定 (5)
用位运算实现两个整数的 (4)
子数组的乘积 (4)

推荐文章

最新评论

轻松搞定面试中的二叉树题目
_LinDL: 学习了，赞下
二分查找，你真的掌握了吗？
wei__hai: 大赞
轻松搞定面试中的链表题目
DanW: 要获得前一个节点的指针，平均需要O(N)的时间复杂度
轻松搞定面试中的二叉树题目
caoyuanlang1: @pure_life:说的是非递归解法吗？同一条路径的也可以吧，怎么不行了，说清楚
轻松搞定面试中的链表题目
fengyang_cpp: 写的真不错
轻松搞定面试中的二叉树题目
lhf_115: 得到树高的函数在vs2010上运行不了
轻松搞定面试中的链表题目
baobao000fdfdsf: 提供求链表环的题目的思路：创建一种集合容器set，它能保证加入集合的数据和已存在的数据不同，否则报错...
轻松搞定面试中的链表题目
baobao000fdfdsf: 第2、3题提供两种思路：一、遍历放入栈容器中，再取出第K个即可。二、递归，层层返回，每次i++，但i...
hashtable简单实现
buxizhizhou530: 迅雷让手写hashtable？
2014年计算机求职总结--面试篇
xiaozhudefendou: 好厉害啊。

```cpp
int GetNodeNum(BinaryTreeNode * pRoot)
{
    if(pRoot == NULL) // 递归出口
        return 0;
    return GetNodeNum(pRoot->m_pLeft) + GetNodeNum(pRoot->m_pRight) + 1;
}
```

2. 求二叉树的深度

递归解法：

（1）如果二叉树为空，二叉树的深度为0

（2）如果二叉树不为空，二叉树的深度 = max(左子树深度， 右子树深度) + 1

参考代码如下：

```cpp
int GetDepth(BinaryTreeNode * pRoot)
{
    if(pRoot == NULL) // 递归出口
        return 0;
    int depthLeft = GetDepth(pRoot->m_pLeft);
    int depthRight = GetDepth(pRoot->m_pRight);
    return depthLeft > depthRight ? (depthLeft + 1) : (depthRight + 1);
}
```

3. 前序遍历，中序遍历，后序遍历

前序遍历递归解法：

（1）如果二叉树为空，空操作

（2）如果二叉树不为空，访问根节点，前序遍历左子树，前序遍历右子树

参考代码如下：

```cpp
void PreOrderTraverse(BinaryTreeNode * pRoot)
{
    if(pRoot == NULL)

    Visit(pRoot); // 访问根节点
    PreOrderTraverse(pRoot->m_pLeft); // 前序遍历左子树
    PreOrderTraverse(pRoot->m_pRight); // 前序遍历右子树
}
```

中序遍历递归解法

（1）如果二叉树为空，空操作。

（2）如果二叉树不为空，中序遍历左子树，访问根节点，中序遍历右子树

参考代码如下：

```cpp
void InOrderTraverse(BinaryTreeNode * pRoot)
{
    if(pRoot == NULL)
        return;
    InOrderTraverse(pRoot->m_pLeft); // 中序遍历左子树
    Visit(pRoot); // 访问根节点
    InOrderTraverse(pRoot->m_pRight); // 中序遍历右子树
}
```

后序遍历递归解法

（1）如果二叉树为空，空操作

（2）如果二叉树不为空，后序遍历左子树，后序遍历右子树，访问根节点

参考代码如下：

```cpp
void PostOrderTraverse(BinaryTreeNode * pRoot)
{
    if(pRoot == NULL)
        return;
    PostOrderTraverse(pRoot->m_pLeft); // 后序遍历左子树
    PostOrderTraverse(pRoot->m_pRight); // 后序遍历右子树
```

```
    Visit(pRoot); // 访问根节点
}
```

4.分层遍历二叉树（按层次从上往下，从左往右）

相当于广度优先搜索，使用队列实现。队列初始化，将根节点压入队列。当队列不为空，进行如下操作：弹出一个节点，访问，若左子节点或右子节点不为空，将其压入队列。

```
[cpp]

void LevelTraverse(BinaryTreeNode * pRoot)
{
    if(pRoot == NULL)
        return;
    queue<BinaryTreeNode *> q;
    q.push(pRoot);
    while(!q.empty())
    {
        BinaryTreeNode * pNode = q.front();
        q.pop();
        Visit(pNode); // 访问节点
        if(pNode->m_pLeft != NULL)
            q.push(pNode->m_pLeft);
        if(pNode->m_pRight != NULL)
            q.push(pNode->m_pRight);
    }
    return;
}
```

5. 将二叉查找树变为有序的双向链表

要求不能创建新节点，只调整指针。

递归解法：

（1）如果二叉树查找树为空，不需要转换，对应双向链表的第一个节点是NULL，最后一个节点是NULL

（2）如果二叉查找树不为空：

如果左子树为空，对应双向有序链表的第一个节点是根节点，左边不需要其他操作；

如果左子树不为空，转换左子树，二叉查找树对应双向有序链表的第一个节点就是左子树转换后双向有序链表的第一个节点，同时将根节点和左子树转换后的双向有序链 表的最后一个节点连接；

如果右子树为空，对应双向有序链表的最后一个节点是根节点，右边不需要其他操作；

如果右子树不为空，对应双向有序链表的最后一个节点就是右子树转换后双向有序链表的最后一个节点，同时将根节点和右子树转换后的双向有序链表的第一个节点连 接。

参考代码如下：

```
[cpp]

/*****************************************************************************
参数：
pRoot: 二叉查找树根节点指针
pFirstNode: 转换后双向有序链表的第一个节点指针
pLastNode: 转换后双向有序链表的最后一个节点指针
*****************************************************************************/
void Convert(BinaryTreeNode * pRoot,
             BinaryTreeNode * & pFirstNode, BinaryTreeNode * & pLastNode)
{
    BinaryTreeNode *pFirstLeft, *pLastLeft, * pFirstRight, *pLastRight;
    if(pRoot == NULL)
    {
        pFirstNode = NULL;
        pLastNode = NULL;
        return;
    }

    if(pRoot->m_pLeft == NULL)
    {
        // 如果左子树为空，对应双向有序链表的第一个节点是根节点
        pFirstNode = pRoot;
    }
    else
    {
        Convert(pRoot->m_pLeft, pFirstLeft, pLastLeft);
        // 二叉查找树对应双向有序链表的第一个节点就是左子树转换后双向有序链表的第一个节点
        pFirstNode = pFirstLeft;
        // 将根节点和左子树转换后的双向有序链表的最后一个节点连接
```

```
        pRoot->m_pLeft = pLastLeft;
        pLastLeft->m_pRight = pRoot;
    }

    if(pRoot->m_pRight == NULL)
    {
        // 对应双向有序链表的最后一个节点是根节点
        pLastNode = pRoot;
    }
    else
    {
        Convert(pRoot->m_pRight, pFirstRight, pLastRight);
        // 对应双向有序链表的最后一个节点就是右子树转换后双向有序链表的最后一个节点
        pLastNode = pLastRight;
        // 将根节点和右子树转换后的双向有序链表的第一个节点连接
        pRoot->m_pRight = pFirstRight;
        pFirstRight->m_pLeft = pRoot;
    }

    return;
}
```

6. 求二叉树第K层的节点个数

递归解法：

（1）如果二叉树为空或者k<1返回0

（2）如果二叉树不为空并且k==1，返回1

（3）如果二叉树不为空且k>1，返回左子树中k-1层的节点个数与右子树k-1层节点个数之和

参考代码如下：

```cpp
int GetNodeNumKthLevel(BinaryTreeNode * pRoot, int k)
{
    if(pRoot == NULL || k < 1)
        return 0;
    if(k == 1)
        return 1;
    int numLeft = GetNodeNumKthLevel(pRoot->m_pLeft, k-1); // 左子树中k-1层的节点个数
    int numRight = GetNodeNumKthLevel(pRoot->m_pRight, k-1); // 右子树中k-1层的节点个数
    return (numLeft + numRight);
}
```

7. 求二叉树中叶子节点的个数

递归解法：

（1）如果二叉树为空，返回0

（2）如果二叉树不为空且左右子树为空，返回1

（3）如果二叉树不为空，且左右子树不同时为空，返回左子树中叶子节点个数加上右子树中叶子节点个数

参考代码如下：

```cpp
int GetLeafNodeNum(BinaryTreeNode * pRoot)
{
    if(pRoot == NULL)
        return 0;
    if(pRoot->m_pLeft == NULL && pRoot->m_pRight == NULL)
        return 1;
    int numLeft = GetLeafNodeNum(pRoot->m_pLeft); // 左子树中叶节点的个数
    int numRight = GetLeafNodeNum(pRoot->m_pRight); // 右子树中叶节点的个数
    return (numLeft + numRight);
}
```

8. 判断两棵二叉树是否结构相同

不考虑数据内容。结构相同意味着对应的左子树和对应的右子树都结构相同。

递归解法：

（1）如果两棵二叉树都为空，返回真

（2）如果两棵二叉树一棵为空，另一棵不为空，返回假

（3）如果两棵二叉树都不为空，如果对应的左子树和右子树都同构返回真，其他返回假

参考代码如下：

```cpp
bool StructureCmp(BinaryTreeNode * pRoot1, BinaryTreeNode * pRoot2)
{
    if(pRoot1 == NULL && pRoot2 == NULL) // 都为空，返回真
        return true;
    else if(pRoot1 == NULL || pRoot2 == NULL) // 有一个为空，一个不为空，返回假
        return false;
    bool resultLeft = StructureCmp(pRoot1->m_pLeft, pRoot2->m_pLeft); // 比较对应左子树
    bool resultRight = StructureCmp(pRoot1->m_pRight, pRoot2->m_pRight); // 比较对应右子树
    return (resultLeft && resultRight);
}
```

9. 判断二叉树是不是平衡二叉树

递归解法：

（1）如果二叉树为空，返回真

（2）如果二叉树不为空，如果左子树和右子树都是AVL树并且左子树和右子树高度相差不大于1，返回真，其他返回假

参考代码：

```cpp
bool IsAVL(BinaryTreeNode * pRoot, int & height)
{
    if(pRoot == NULL) // 空树，返回真
    {
        height = 0;
        return true;
    }
    int heightLeft;
    bool resultLeft = IsAVL(pRoot->m_pLeft, heightLeft);
    int heightRight;
    bool resultRight = IsAVL(pRoot->m_pRight, heightRight);
    if(resultLeft && resultRight && abs(heightLeft - heightRight) <= 1) // 左子树和右子树都是AVL，并且高度相差不大于1，返回真
    {
        height = max(heightLeft, heightRight) + 1;
        return true;
    }
    else
    {
        height = max(heightLeft, heightRight) + 1;
        return false;
    }
}
```

10. 求二叉树的镜像

递归解法：

（1）如果二叉树为空，返回空

（2）如果二叉树不为空，求左子树和右子树的镜像，然后交换左子树和右子树

参考代码如下：

```cpp
BinaryTreeNode * Mirror(BinaryTreeNode * pRoot)
{
    if(pRoot == NULL) // 返回NULL
        return NULL;
    BinaryTreeNode * pLeft = Mirror(pRoot->m_pLeft); // 求左子树镜像
    BinaryTreeNode * pRight = Mirror(pRoot->m_pRight); // 求右子树镜像
        // 交换左子树和右子树
    pRoot->m_pLeft = pRight;
    pRoot->m_pRight = pLeft;
    return pRoot;
}
```

11. 求二叉树中两个节点的最低公共祖先节点

递归解法：

（1）如果两个节点分别在根节点的左子树和右子树，则返回根节点

（2）如果两个节点都在左子树，则递归处理左子树；如果两个节点都在右子树，则递归处理右子树

参考代码如下：

```cpp
bool FindNode(BinaryTreeNode * pRoot, BinaryTreeNode * pNode)
{
    if(pRoot == NULL || pNode == NULL)
        return false;

    if(pRoot == pNode)
        return true;

    bool found = FindNode(pRoot->m_pLeft, pNode);
    if(!found)
        found = FindNode(pRoot->m_pRight, pNode);

    return found;
}

BinaryTreeNode * GetLastCommonParent(BinaryTreeNode * pRoot,
                                     BinaryTreeNode * pNode1,
                                     BinaryTreeNode * pNode2)
{
    if(FindNode(pRoot->m_pLeft, pNode1))
    {
        if(FindNode(pRoot->m_pRight, pNode2))
            return pRoot;
        else
            return GetLastCommonParent(pRoot->m_pLeft, pNode1, pNode2);
    }
    else
    {
        if(FindNode(pRoot->m_pLeft, pNode2))
            return pRoot;
        else
            return GetLastCommonParent(pRoot->m_pRight, pNode1, pNode2);
    }
}
```

递归解法效率很低，有很多重复的遍历，下面看一下非递归解法。

非递归解法：

先求从根节点到两个节点的路径，然后再比较对应路径的节点就行，最后一个相同的节点也就是他们在二叉树中的最低公共祖先节点

参考代码如下：

```cpp
bool GetNodePath(BinaryTreeNode * pRoot, BinaryTreeNode * pNode,
                 list<BinaryTreeNode *> & path)
{
    if(pRoot == pNode)
    {
        path.push_back(pRoot);
        return true;
    }
    if(pRoot == NULL)
        return false;
    path.push_back(pRoot);
    bool found = false;
    found = GetNodePath(pRoot->m_pLeft, pNode, path);
    if(!found)
        found = GetNodePath(pRoot->m_pRight, pNode, path);
    if(!found)
        path.pop_back();
    return found;
}
BinaryTreeNode * GetLastCommonParent(BinaryTreeNode * pRoot, BinaryTreeNode * pNode1, BinaryTreeNode * pNod
{
    if(pRoot == NULL || pNode1 == NULL || pNode2 == NULL)
        return NULL;
    list<BinaryTreeNode*> path1;
    bool bResult1 = GetNodePath(pRoot, pNode1, path1);
    list<BinaryTreeNode*> path2;
    bool bResult2 = GetNodePath(pRoot, pNode2, path2);
    if(!bResult1 || !bResult2)
        return NULL;
    BinaryTreeNode * pLast = NULL;
    list<BinaryTreeNode*>::const_iterator iter1 = path1.begin();
    list<BinaryTreeNode*>::const_iterator iter2 = path2.begin();
```

```
    while(iter1 != path1.end() && iter2 != path2.end())
    {
        if(*iter1 == *iter2)
            pLast = *iter1;
        else
            break;
        iter1++;
        iter2++;
    }
    return pLast;
}
```

在上述算法的基础上稍加变化即可求二叉树中任意两个节点的距离了。

12. 求二叉树中节点的最大距离

即二叉树中相距最远的两个节点之间的距离。

递归解法：

（1）如果二叉树为空，返回0，同时记录左子树和右子树的深度，都为0

（2）如果二叉树不为空，最大距离要么是左子树中的最大距离，要么是右子树中的最大距离，要么是左子树节点中到根节点的最大距离+右子树节点中到根节点的最大距离，同时记录左子树和右子树节点中到根节点的最大距离。

参考代码如下：

```cpp
int GetMaxDistance(BinaryTreeNode * pRoot, int & maxLeft, int & maxRight)
{
    // maxLeft, 左子树中的节点距离根节点的最远距离
    // maxRight, 右子树中的节点距离根节点的最远距离
    if(pRoot == NULL)
    {
        maxLeft = 0;
        maxRight = 0;
        return 0;
    }
    int maxLL, maxLR, maxRL, maxRR;
    int maxDistLeft, maxDistRight;
    if(pRoot->m_pLeft != NULL)
    {
        maxDistLeft = GetMaxDistance(pRoot->m_pLeft, maxLL, maxLR);
        maxLeft = max(maxLL, maxLR) + 1;
    }
    else
    {
        maxDistLeft = 0;
        maxLeft = 0;
    }
    if(pRoot->m_pRight != NULL)
    {
        maxDistRight = GetMaxDistance(pRoot->m_pRight, maxRL, maxRR);
        maxRight = max(maxRL, maxRR) + 1;
    }
    else
    {
        maxDistRight = 0;
        maxRight = 0;
    }
    return max(max(maxDistLeft, maxDistRight), maxLeft+maxRight);
}
```

13. 由前序遍历序列和中序遍历序列重建二叉树

二叉树前序遍历序列中，第一个元素总是树的根节点的值。中序遍历序列中，左子树的节点的值位于根节点的值的左边，右子树的节点的值位

于根节点的值的右边。

递归解法：

（1）如果前序遍历为空或中序遍历为空或节点个数小于等于0，返回NULL。

（2）创建根节点。前序遍历的第一个数据就是根节点的数据，在中序遍历中找到根节点的位置，可分别得知左子树和右子树的前序和中序遍

历序列，重建左右子树。

```cpp
BinaryTreeNode * RebuildBinaryTree(int* pPreOrder, int* pInOrder, int nodeNum)
{
    if(pPreOrder == NULL || pInOrder == NULL || nodeNum <= 0)
        return NULL;
    BinaryTreeNode * pRoot = new BinaryTreeNode;
    // 前序遍历的第一个数据就是根节点数据
    pRoot->m_nValue = pPreOrder[0];
    pRoot->m_pLeft = NULL;
    pRoot->m_pRight = NULL;
    // 查找根节点在中序遍历中的位置，中序遍历中，根节点左边为左子树，右边为右子树
    int rootPositionInOrder = -1;
    for(int i = 0; i < nodeNum; i++)
        if(pInOrder[i] == pRoot->m_nValue)
        {
            rootPositionInOrder = i;
            break;
        }
    if(rootPositionInOrder == -1)
    {
        throw std::exception("Invalid input.");
    }
    // 重建左子树
    int nodeNumLeft = rootPositionInOrder;
    int * pPreOrderLeft = pPreOrder + 1;
    int * pInOrderLeft = pInOrder;
    pRoot->m_pLeft = RebuildBinaryTree(pPreOrderLeft, pInOrderLeft, nodeNumLeft);
    // 重建右子树
    int nodeNumRight = nodeNum - nodeNumLeft - 1;
    int * pPreOrderRight = pPreOrder + 1 + nodeNumLeft;
    int * pInOrderRight = pInOrder + nodeNumLeft + 1;
    pRoot->m_pRight = RebuildBinaryTree(pPreOrderRight, pInOrderRight, nodeNumRight);
    return pRoot;
}
```

同样，有中序遍历序列和后序遍历序列，类似的方法可重建二叉树，但前序遍历序列和后序遍历序列不同恢复一棵二叉树，证明略。

14.判断二叉树是不是完全二叉树

若设二叉树的深度为h，除第 h 层外，其它各层（1～h-1）的结点数都达到最大个数，第 h 层所有的结点都连续集中在最左边，这就是完全

二叉树。

有如下算法，按层次（从上到下，从左到右）遍历二叉树，当遇到一个节点的左子树为空时，则该节点右子树必须为空，且后面遍历的节点左

右子树都必须为空，否则不是完全二叉树。

```cpp
bool IsCompleteBinaryTree(BinaryTreeNode * pRoot)
{
    if(pRoot == NULL)
        return false;
    queue<BinaryTreeNode *> q;
    q.push(pRoot);
    bool mustHaveNoChild = false;
    bool result = true;
    while(!q.empty())
    {
        BinaryTreeNode * pNode = q.front();
        q.pop();
        if(mustHaveNoChild) // 已经出现了有空子树的节点了，后面出现的必须为叶节点（左右子树都为空）
        {
            if(pNode->m_pLeft != NULL || pNode->m_pRight != NULL)
            {
                result = false;
                break;
            }
        }
        else
        {
            if(pNode->m_pLeft != NULL && pNode->m_pRight != NULL)
            {
                q.push(pNode->m_pLeft);
                q.push(pNode->m_pRight);
            }
```

```
            else if(pNode->m_pLeft != NULL && pNode->m_pRight == NULL)
            {
                mustHaveNoChild = true;
                q.push(pNode->m_pLeft);
            }
            else if(pNode->m_pLeft == NULL && pNode->m_pRight != NULL)
            {
                result = false;
                break;
            }
            else
            {
                mustHaveNoChild = true;
            }
        }
    }
    return result;
}
```


上一篇 poj 1328 Radar Installation

下一篇 Reversing Microsoft Visual C++ Part I: Exception Handling

查看评论

34楼 _LinDL 2015-09-05 08:41发表

学习了，赞下

33楼 lhf_115 2015-06-03 19:57发表

得到树高的函数在vs2010上运行不了

32楼 zhl12358 2015-04-15 23:50发表

受教了

31楼 ttyyoung 2015-03-24 21:04发表

11非递归解法中

```cpp
if(pRoot == pNode)
    {
        path.push_back(pRoot);
        return true;
    }
    if(pRoot == NULL)
        return false;
```

两个判断应该调换一下位置

30楼 ttyyoung 2015-03-24 21:01发表

11中

```cpp
bool found = FindNode(pRoot->m_pLeft, pNode);
 if(!found)
found = FindNode(pRoot->m_pRight, pNode);
```

可以用bool found = FindNode(pRoot->m_pLeft, pNode)|| FindNode(pRoot->m_pRight, pNode); 代替，因为||的短路性

29楼 FengCoder 2015-03-15 10:06发表

博主 24楼说得有道理，平衡二叉树，首先它必须是一种二叉查找树，其次再看其平衡因子是否满足条件

28楼 ayangdanny11 2015-03-03 15:44发表

11题的递归算法中，当其中一个节点是另一个节点的祖先节点时，楼主所述将出现问题，应该将这一特殊情况考虑进去

27楼 GarryLin 2015-02-21 10:51发表

非常感谢博主整合!

26楼 kimi1911 2014-09-05 16:38发表

第11题，做的时候想到了一个 n + log n 的方法，层次遍历一次，利用辅助的数据结构，然后再进行一次有序的查找，就可以了。不知道楼主有没有更简单的算法可以交流一下。

25楼 蒲公英小帝 2014-09-04 07:37发表

很不错

24楼 WalkingInTheWind 2014-08-27 09:14发表

回复DHL1234567：你再想想吧，画个图看一下

23楼 inCR7dible 2014-08-22 16:19发表

大赞，必须收藏！最关键的是思路太赞了，确实学习了~~

22楼 ramboww 2014-08-20 20:38发表

这个平衡二叉树就是所谓的AVL树吗？那不是应该满足二叉查找树的要求么，但是“判断二叉树是不是平衡二叉树”这一题没有判断父子节点之间的大小关系？
还是说我理解错了？

Re: WalkingInTheWind 2014-08-20 20:43发表

回复ramboww：平衡二叉树和二叉查找树不是一个概念，AVL树是平衡二叉查找树，可以说是平衡二叉树和二叉查找树的结合。

Re: ramboww 2014-08-20 20:56发表

回复WalkingInTheWind：恩，好象是，概念不太清晰，eg:http://dongxicheng.org/structure/red-black-tree/.
不过也没啥大问题。

21楼 sccotjobs 2014-07-25 11:26发表

回复u012609067：我理解错了 不好意思

20楼 richardzrc 2014-07-21 23:14发表

LZ好奇，而且把计算height都统一到一个递归了，我绝对话写一个求height的递归函数：）

19楼 richardzrc 2014-07-21 23:11发表

判平衡二叉树为啥，一定要leftheight rightheight >0 才是平衡的，不一定吧

18楼 asz9255 2014-03-28 15:54发表

求两个节点的最大距离，我想了个更简单的方法。

```cpp
int getMaxDistance(TreeNode * pRoot){
  if (pRoot == NULL)
  {
    return 0;
  }

  return max(getTreeDepth(pRoot->left) + getTreeDepth(pRoot->right),
      getMaxDistance(pRoot->left), getMaxDistance(pRoot->right));
}
```

17楼 紫月蓝枫 2014-03-18 18:57发表

算法在于思想和实现，不要太注重语言的语法细节！这篇文章很棒！

Re: biruixing 2014-09-05 11:29发表

回复legendqiang：你再学几年再说

16楼 biruixing 2014-03-17 11:11发表

你在编译器上验证了吗，定义都是错误的，少了typedef编译是错误的，建议你运行成功了再发表，以免误导

Re: WalkingInTheWind 2014-03-17 11:16发表

回复biruixing：不好意思，我一般用c++，很少用纯c语言，c++中的struct是可以这样定义的。

15楼 John8169 2014-03-14 12:44发表

不错，不错，收藏了

14楼 Tangbzh 2014-02-07 12:13发表

递归，一直很迷糊，怎么破？

13楼 chenzhg33 2014-01-13 16:17发表

绝对收藏了，赞一个

12楼 zyc4262485789 2013-10-23 01:36发表

大赞楼主，好文啊

Re: WalkingInTheWind 2013-12-11 10:02发表

回复zyc4262485789：谢谢，这里总结的也不全，还有不少相关的好题目没有加进来，有时间再重新整理一遍。

11楼 yunlzz 2013-09-06 15:45发表

第五题，二叉查找有序的树转换为有序的双向链表，第45行代码pFirstRight->m_pRight = pRoot; 是不是应该改为：pFirstRight->m_pLeft = pRoot;

Re: WalkingInTheWind 2013-09-06 20:20发表

回复yunlzz：对，谢谢！

10楼 江南烟雨 2013-09-02 10:49发表

总结的不错！

9楼 yamaide 2013-08-10 18:09发表

第一题求叶子节点数代码有问题
得出的结果是正确的结果的两倍，因为把具有一个孩子的节点也当做叶子节点。
正确的代码应该需要判断是否是叶子节点
我的代码如下

```
int Leaf(SearchTree T)
{
if (T == NULL)
return 0;
if (T->Left == NULL && T->Right == NULL)//判断是否是叶子节点
return 1;
return Leaf(T->Left) + Leaf(T->Right);
}
```

PS:LZ很有毅力啊，写了这么多算法，佩服

Re: WalkingInTheWind 2013-08-10 22:56发表

回复yamaide：第一题吗？第一题是求节点个数，不是叶节点

Re: yamaide 2013-08-12 11:03发表

回复WalkingInTheWind：是我看错了==!

8楼 紫官樱草 2013-05-20 18:06发表

第四题好多看不多，用得太多自己定义的

7楼 紫官樱草 2013-05-20 17:30发表

great~

6楼 alzking 2013-05-10 17:24发表

总结得非常好～ Thx

5楼 heheluolei 2012-11-05 17:07发表

感谢楼主！受益匪浅！

4楼 石锅拌饭 2012-10-05 23:08发表

总结的很全，顶一个。

3楼 meiyouwanzimian 2012-10-03 17:10发表

第14题好像缺少一种判断，在一个节点左右子树都为空时mustHaveNoChild = true

Re: WalkingInTheWind 2012-10-03 21:58发表

回复meiyouwanzimian：是的，多谢指正

2楼 zxtgg 2012-09-13 09:21发表

不错~

1楼 BYD123 2012-08-30 14:12发表

文章不错，但是算法本身又用到了已有的数据结构。