27计算高度

时间复杂度O(n)

int height (binaryTreeNode<E>\* rootin) {

if (rootin == nullptr) {

return 0;

}

int lheight = height (rootin->leftChild);

int rheight = height (rootin->rightChild);

return lheight > rheight ? ++lheight : ++rheight;

}

28计算个数

时间复杂度O(n)

int getSizeByLevelOrder () {// Level-order traversal.

arrayQueue<binaryTreeNode<E>\*> q;

binaryTreeNode<E>\* t = root;

int size = 0;

while (t != nullptr) {

size++;

if (t->left != nullptr)

q.push (t->left);

if (t->right != nullptr)

q.push (t->right);

if (q.empty ()) {

return size;

} else {

t = q.front ();

q.pop ();

}

}

}

29.计算最大宽度

外层while循环层数次，内层while代表每层元素数次，一共n次，时间复杂度0(n)

int getMaxWidth () {

//如果树为空，则宽度为0

if (root == nullptr) {return 0;}

int result = 0;

queue<binaryTreeNode<E>\*> q;

q.push (root);

while (!q.empty ()) {

int count = q.size ();//当前层的宽度

result = max (count, result);//获取最大宽度

while (count--) {//获取下一层元素

binaryTreeNode<E>\* temp = q.front ();

q.pop ();

if (temp->left != nullptr) {

q.push (temp->left);

}

if (temp->right != nullptr) {

q.push (temp->right);

}

}

}

return result;

}

33.能确定

template<class E>

binaryTreeNode<E>\* getTreeByPreAndIn (binaryTreeNode<E>\* preorder, binaryTreeNode<E>\* inorder,int length) {

//递归结束条件

if (length == 0) {

return nullptr;

}

binaryTreeNode<E>\* root = new binaryTreeNode<E>(preorder[0]);//当前根节点

//找到中序遍历根节点的位置

int index;

for (index = 0; inorder[index] != preorder[0]; index++) {}

//构建左右子树

root->left = getTreeByPreAndIn (preorder + 1, inorder, index);

root->right = getTreeByPreAndIn (preorder + 1 + index, inorder + 1 + index, length - 1 - index);

return root;//返回当前根节点

}

对每个节点都要构建一次，因此时间复杂度为O(n)