**树的计数、二叉树的恢复实现（笔试面试考的比较多）**

**一、树的计数**

**定义：**二叉树的计数问题就是讨论具有n个结点，两两之间互不相似的二叉树的个数。

（树的计数记住结论即可。P154上方）

**二、二叉树的恢复**

**定义：**二叉树的恢复是可以根据二叉树结点前序序列、中序序列和后序序列，恢复出惟一性二叉树。

**恢复二叉树的方法实现：（下面列出两种，唯独根据前序和后序不行，因为这样会出现相同的序列结构不同的二叉树形态，即不具有惟一性）**

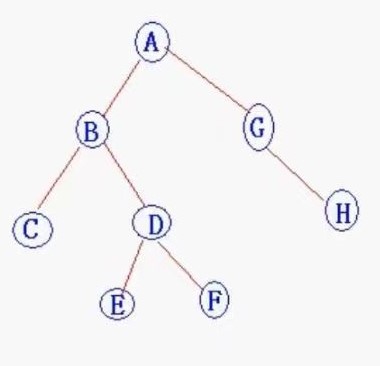
**一、//根据前序和中序序列**

由前序序列可以得到第一个数据就是根结点A，紧接着在中序中找到A的位置，并将中序序列一分为二，左边为A的左子树，右边为A的右子树。

紧接着轮到了B，看B在中序中的位置，发现在A的左边，则以A的左子树出现。

接着到C，发现C在A的左边，说明是A的左树，同时C也在B的左边，说明C是B的左树，依此类推，得到的二叉树如图所示。

VLR = "ABCDEFGH"; （前序） VLR = "CBEDFAGH";（中序）

****

void CreatBinTree(BinTreeNode\*& t, const char\* VLR, const char\* LVR, int n) {

if (n == 0) {

t = nullptr;

}

else {

int k = 0;

while (VLR[0] != LVR[k]) { //查找根结点

k++;

}

t = (BinTreeNode\*)malloc(sizeof(BinTreeNode));

assert(t != NULL);

t->data = LVR[k];

CreatBinTree(t->leftChild, VLR + 1, LVR, k);

**//VLR + 1？对于先序来说，由于是“根左右”的顺序，因此当前结点的左树在先序序列里一定是在当前结点的后面，因此字符指针指向当前节点的后面一个开始遍历。**

**//LVR ？对于中序来说，A的左边全是左树，则从第一个开始就行**

**//k相当于是左树里的元素个数**

CreatBinTree(t->rightChild, VLR +k+1, LVR+k+1, n-k-1);

**//VLR +k+1这一步如何确定？（先通过中序找到右树，然后在先序里确定第一个出现右树元素的位置）**

**//LVR+k+1？在中序里找右树的位置就行**

}

}

void CreatBinTree(BinTree& bt, const char\* VLR, const char\* LVR, int n) {

CreatBinTree(bt.root, VLR, LVR, n);

}

1. **//根据中序和后序序列**

**由于二叉树的后序序列最后一个为根结点，因此我们可以倒着查找。接着在中序里找到A结点的位置，并将中序序列一分为二，左边为A的左子树，右边为A的右子树。因为是倒着查找（**LRV**）所以我们在递归的时候可以进行变通，先建立右子树，再建立左子树。**

LVR = "CBEDFAGH"; LRV = "CEFDBHGA";

void CreatBinTree2(BinTreeNode\*& t, const char\* LVR, const char\* LRV, int n) {

if (n == 0) {

t = nullptr;

}

else {

int k = 0;

while (LRV[n-1] != LVR[k]) { //查找根结点

k++;

}

t = (BinTreeNode\*)malloc(sizeof(BinTreeNode));

assert(t != NULL);

t->data = LVR[k];

CreatBinTree2(t->rightChild, LVR + k + 1, LRV + k, n - k - 1);

//LVR+k+1? 在中序里找到右子树开始的位置即可

//LRV+k? 由中序确定右子树里的元素，然后在后序里找到第一个右树元素出现的位置即可

CreatBinTree2(t->leftChild, LVR, LRV, k);

//LVR？左树元素在中序里就是从第一个开始

//LRV？后续的左子树也是从第一个元素开始

}

}

void CreatBinTree2(BinTree& bt, const char\* LVR, const char\* LRV, int n) {

CreatBinTree2(bt.root, LVR,LRV, n);

}