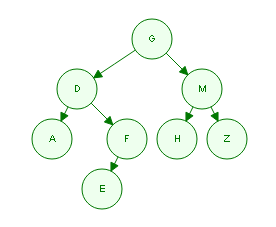
[二叉树系列(1)已知二叉树的中序遍历和前序遍历，如何求后序遍历](http://blog.csdn.net/feliciafay/article/details/6816871)

分类： [C族](http://blog.csdn.net/feliciafay/article/category/679348) [数据结构与算法DataStructure&Algorithm](http://blog.csdn.net/feliciafay/article/category/679350)2011-09-24 11:04 6191人阅读 [评论](http://blog.csdn.net/feliciafay/article/details/6816871#comments)(3) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/feliciafay/article/details/6816871#report)

[struct](http://www.csdn.net/tag/struct)[null](http://www.csdn.net/tag/null)

(昨晚HULU的笔试题之一，被鄙视的惨，面对向往已久的公司，交出苍白无力的答卷，心里像被剜了一刀。再多解释都是苍白。我鄙视我自己。现在开始舔舐伤口。)

假设有棵树，长下面这个样子，它的前序遍历，中序遍历，后续遍历都很容易知道。



PreOrder:         GDAFEMHZ

InOrder:            ADEFGHMZ

PostOrder:       AEFDHZMG

现在，假设仅仅知道前序和中序遍历，如何求后序遍历呢？比如，已知一棵树的前序遍历是”GDAFEMHZ”，而中序遍历是”ADEFGHMZ”应该如何求后续遍历?

第一步，root最简单，前序遍历的第一节点G就是root。

第二步，继续观察前序遍历GDAFEMHZ，除了知道G是root，剩下的节点必然是root的左右子树之外，没法找到更多信息了。

第三步，那就观察中序遍历ADEFGHMZ。其中root节点G左侧的ADEF必然是root的左子树，G右侧的HMZ必然是root的右子树。

第四步，观察左子树ADEF，左子树的中的根节点必然是大树的root的leftchild。在前序遍历中，大树的root的leftchild位于root之后，所以左子树的根节点为D。

第五步，同样的道理，root的右子树节点HMZ中的根节点也可以通过前序遍历求得。在前序遍历中，一定是先把root和root的所有左子树节点遍历完之后才会遍历右子树，并且遍历的右子树的第一个节点就是右子树的根节点。

如何知道哪里是前序遍历中的左子树和右子树的分界线呢？通过中序遍历去数节点的个数。

在上一次中序遍历中，root左侧是A、D、E、F，所以有4个节点位于root左侧。那么在前序遍历中，必然是第1个是G，第2到第5个由A、D、E、F过程，第6个就是root的右子树的根节点了，是M。

第六步，观察发现，上面的过程是递归的。先找到当前树的根节点，然后划分为左子树，右子树，然后进入左子树重复上面的过程，然后进入右子树重复上面的过程。最后就可以还原一棵树了。

第七步，其实，如果仅仅要求写后续遍历，甚至不要专门占用空间保存还原后的树。只需要稍微改动第六步，就能实现要求。仅需要把第六步的递归的过程改动为如下:

1 确定根,确定左子树，确定右子树。

2 在左子树中递归。

3 在右子树中递归。

4 打印当前根。

参考了一些网上的讨论，具体程序是:

**[cpp]** [view plaincopy](http://blog.csdn.net/feliciafay/article/details/6816871)

1. #include <iostream>
2. #include <fstream>
3. #include <string>
5. **struct** TreeNode
6. {
7. **struct** TreeNode\* left;
8. **struct** TreeNode\* right;
9. **char**  elem;
10. };

13. TreeNode\* BinaryTreeFromOrderings(**char**\* inorder, **char**\* preorder, **int** length)
14. {
15. **if**(length == 0)
16. {
17. **return** NULL;
18. }
19. TreeNode\* node = **new** TreeNode;//Noice that [new] should be written out.
20. node->elem = \*preorder;
21. **int** rootIndex = 0;
22. **for**(;rootIndex < length; rootIndex++)//a variation of the loop
23. {
24. **if**(inorder[rootIndex] == \*preorder)
25. **break**;
26. }
27. node->left = BinaryTreeFromOrderings(inorder, preorder +1, rootIndex);
28. node->right = BinaryTreeFromOrderings(inorder + rootIndex + 1, preorder + rootIndex + 1, length - (rootIndex + 1));
29. std::cout<<node->elem<<std::endl;
30. **return** node;
31. }
33. **int** main(**int** argc, **char**\*\* argv){
34. **char**\* pr="GDAFEMHZ";
35. **char**\* in="ADEFGHMZ"; BinaryTreeFromOrderings(in, pr, 8); printf("\n"); **return** 0;}

其实上面的代码写得不够简洁。题目只要求输出后续遍历，并没有要求建树。所以，不需要去计算出node->left与node->right，也不需要去return node。改进版本如下

**[cpp]** [view plaincopy](http://blog.csdn.net/feliciafay/article/details/6816871)

1. **struct** TreeNode
2. {
3. **struct** TreeNode\* left;
4. **struct** TreeNode\* right;
5. **char**  elem;
6. };
8. **void** BinaryTreeFromOrderings(**char**\* inorder, **char**\* preorder, **int** length)
9. {
10. **if**(length == 0)
11. {
12. //cout<<"invalid length";
13. **return**;
14. }
15. TreeNode\* node = **new** TreeNode;//Noice that [new] should be written out.
16. node->elem = \*preorder;
17. **int** rootIndex = 0;
18. **for**(;rootIndex < length; rootIndex++)
19. {
20. **if**(inorder[rootIndex] == \*preorder)
21. **break**;
22. }
23. //Left
24. BinaryTreeFromOrderings(inorder, preorder +1, rootIndex);
25. //Right
26. BinaryTreeFromOrderings(inorder + rootIndex + 1, preorder + rootIndex + 1, length - (rootIndex + 1));
27. cout<<node->elem<<endl;
28. **return**;
29. }

32. **int** main(**int** argc, **char**\* argv[])
33. {
34. printf("Hello World!\n");
35. **char**\* pr="GDAFEMHZ";
36. **char**\* in="ADEFGHMZ";
38. BinaryTreeFromOrderings(in, pr, 8);
40. printf("\n");
41. **return** 0;
42. }