**数据结构实验报告**

姓名： 赵楠 学号：U201617007 班级：软工1601班

**实验六**

1. 问题描述

1、4.42 判定二叉树的同构，交换部分（或全部）节点的左右儿子后得到的树称为同构。请写出相应函数，判断两棵给定的树是否同构。

2、4.46 2-d树。使用两个关键词的二叉查找树。完成问题(b)和(c) 。

二、问题分析与算法设计

1、问题1

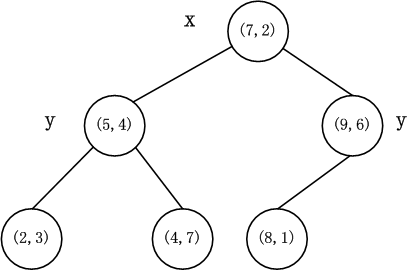
（1）本题要判断二叉树的同构性，不难想到可以通过每次比较first根节点以及second根节点的两个子节点之间的相互关系，如若first节点左节点与second节点的右节点相同，而first节点的右节点与second节点的左节点也相同，则将second节点的左右子节点交换，这样一直遍历整棵树，遇到类似情况则交换节点，即可完成所有判断。

（2）解法1需要改变second节点或者first节点，那我们能否不通过修改节点的关系来获取正确结果呢？答案是可以的，我们转一个弯，每次遇到解法1中示例情形时，我们可以不交换左右节点，而是通过递归的方式比较first节点左子树和second节点的右子树，比较first节点的右子树和second节点的左子树，即可达到相同的效果。

1. 问题2

b、问题2乍一看很难，后来发现我们只需要解决简化的2-d 树，而不是 完整的 k-d树，这里 2-d 树的解释很清楚，在树的偶数层比较key1(层数从0开始），树的奇数层比较key2，一直到正确的位置。

例如：



当（5， 4）插入时，5先与第0层的7比较，比7小，而（7， 2）此时没有子节点，故插入为（7, 2）的左子树，中间略过其他树的插入，对于（8, 1）的插入，首先8比7大，故再与（7,2）的右子树比较，而此时比较key2，1比6小，所以（8,1）插入（9,6）的左子树。

解释到这里，如何插入已经很明显了，只要用一个变量表示当前是奇数层还是偶数层，再按照顺序进行判断，找到适当的位置插入即可。

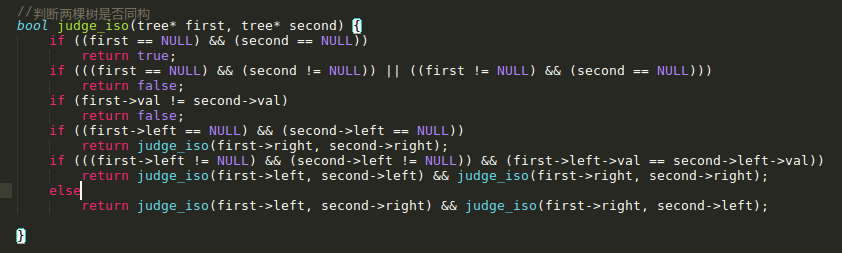
c、有了b问题的基础，c问题只需直接执行先序、后序后者中序中的一种遍历即可，这里选择先序遍历，遍历过程中发现满足条件即可输出。

1. 算法实现

代码见钉盘

1、问题1

关键函数代码如下：



解释：每次都只需判断first节点以及second节点的关系以及first节点的的左子树和second节点的左子树的关系即可，右子树则不必判断，因为右子树的判断包含在first节点以及second节点的关系中（下一轮递归中），故不必重复判断。

1. 问题2

关键函数代码如下：

// 2-d 树的插入

two\_d\_Tree\* insert(two\_d\_Tree\* root, two\_d\_Tree\* tree)

{

if (root == NULL)

return tree;

int odd\_even = 0; // 0 代表偶数层， 1代表奇数层

two\_d\_Tree\* temp = root;

while (temp != NULL)

{

// 偶数层，从第0层开始

if (odd\_even)

{

if (tree->key2 < temp->key2)

{

if (temp->left == NULL)

{

temp->left = tree;

break;

}

else

temp = temp->left;

}

else

{

if (temp->right == NULL)

{

temp->right = tree;

break;

}

else

temp = temp->right;

}

odd\_even = 0;

}

else

{

if (tree->key1 < temp->key1)

{

if (temp->left == NULL)

{

temp->left = tree;

break;

}

else

temp = temp->left;

}

else

{

if (temp->right == NULL)

{

temp->right = tree;

break;

}

else

temp = temp->right;

}

odd\_even = 1;

}

}

return root;

}

//高效打印满足 条件 low1 <= key1 <= high1, low2 <= key2 <= high2

void print\_keys(two\_d\_Tree\* root, int low1, int high1, int low2, int high2)

{

if (root != NULL)

{

if (root->key1 >= low1 && root->key1 <= high1 && root->key2 >= low2 && root->key2 <= high2)

printf("(%d, %d) ", root->key1, root->key2);

print\_keys(root->left, low1, high1, low2, high2);

print\_keys(root->right, low1, high1, low2, high2);

}

}

四、测试结果与分析

1. 问题1

测试用例：

First Second

1 1

2 3 3 2

4 5 7 7 5 4

6 8 8 6

输出结果：



Ok

1. 问题2

插入数组（按顺序插入）：

int test[10][2] = {{7, 2}, {5, 4}, {9, 6}, {2, 3}, {4, 7}, {8, 1}, {7, 7}, {1, 4}, {3, 5}, {8, 4}};

2-d树：

（7,2）

（5,4） （9,6）

（2,3） （4,7） （8,1） （7,7）

（1,4） （8,4）

（3,5）

问题c: low1 = 3, high1 = 7, low2 = 4, high2 = 7

打印结果：



ok