**数据结构实验报告**

姓名：郭宇瑶 学号：U201717031 班级：软工1704班

**实验六 判定二叉树同构及2-d树问题的求解**

一、实验描述

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) 。

2-d树类似于二叉树，不同之处在于在偶数层用Key1来分叉，奇数层用Key2来分叉。

（b）给定美国总统的名和姓，编写一个向一棵2-d树进行插入的历程。

(c)编写一个高效的过程，该过程打印同时满足Low1<=Key1<=High1和Low2<=Key2<=High2的树的所有记录。

二、实验设计

1.如果树T1通过交换其（某些）节点的左右儿子变成树T2，则称树T1与T2同构。给出一个多项式时间算法以决定是否两棵树是同构的。

输入格式:

输入给出2棵二叉树树的信息。共给定四个数组，各自通过先序遍历和中序遍历的结果构造两棵二叉树

输出格式:

如果两棵树是同构的，输出“Yes”，否则输出“No”。

二叉树的实现、构造、层序遍历用了实验五的代码

实现：（1）定义树的结构体：

typedef struct binTreeNode {

ElementType element;

struct binTreeNode \*left;

struct binTreeNode \*right;

} BinTreeNode;

（2）构造二叉树：利用实验五的pre\_and\_mid\_construct(char a[], char b[], int n)函数构造两个二叉树；

（3）判定是否同构：如果两棵树为空，则同构；

如果一棵树为空，另一棵树不为空，则不同构；

如果两棵树根节点的数据不同，则不同构 ；

如果两棵树左节点都为空，则比较右节点 ；

如果两棵树左节点不为空且数据相同，则比较左左，右右 ；

否则比较左右，右左。

1. 测试数据：例题的两个同构的树

2.2-d树

（1）定义二叉查找树结构体，与普通二叉树不同的是有两个元素属性：key1和key2

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*key1 : key1 分叉条件

\*key2 : key2 分叉条件

\*left : 左子树

\*right : 右子树

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

struct TreeNode {

ElementType \*Key1;

ElementType \*Key2;

Tree Left;

Tree Right; };

（2）实现2-d树的插入：相比于普通二叉树的插入，要对层数进行判断，从而用key进行比较、递归；经过插入历程生成一个二叉树

（3）实现打印同时满足Low1<=Key1<=High1和Low2<=Key2<=High2的树的所有记录：中序遍历，对左右子树分别遍历比较key的值。

（4）测试数据：4.46的数据

1. 实验实现过程

1.判断两棵树是否同构

#define ElementType char

#define YES 1

#define NO 0

typedef struct binTreeNode {

ElementType element;

struct binTreeNode \*left;

struct binTreeNode \*right;

} BinTreeNode;

//根据先序遍历和中序遍历数组构造二叉树

static binTreeNode \*pre\_and\_mid\_construct(char a[], char b[], int n) {

binTreeNode \*root;

int k, l;

/\* trivial case: tree is empty \*/

if (n == 0) return (NULL);

root = (binTreeNode \*)malloc(sizeof(binTreeNode));

root->element = a[0];

k = find\_root(b, n, a[0]);

if (k < 0) {

printf("node %c not found in in-order traversal\n", a[0]); return (NULL); }

root->left = pre\_and\_mid\_construct(&a[1], b, k);

root->right = pre\_and\_mid\_construct(&a[k + 1], &b[k + 1], n - k - 1);

return (root);

}

int isomorphic(binTreeNode \*tree1, binTreeNode \*tree2) {

//如果两棵树为空，则同构

if (tree1 == NULL && tree2 == NULL) { return YES; }

//如果一棵树为空，另一棵树不为空，则不同构

if ((tree1 == NULL && tree2 != NULL) || (tree1 != NULL && tree2 == NULL)) { return NO; }

//如果两棵树根节点的数据不同，则不同构

if (tree1->element != tree2->element) { return NO; }

//如果两棵树左节点都为空，则比较右节点

if (tree1->left == NULL && tree2->left == NULL) { return isomorphic(tree1->right, tree2->right); }

//如果两棵树左节点不为空且数据相同，则比较左左，右右

if ((tree1->left != NULL && tree2->left != NULL) && (tree1->element == tree2->element)) { return isomorphic(tree1->left, tree2->left) && isomorphic(tree1->right, tree2->right); }

//否则比较左右，右左

else { return isomorphic(tree1->left, tree2->right) && isomorphic(tree1->right, tree2->left); }

}

int main()

{

int i;

//binTreeNode \*Root1 = NULL,\* Root2 = NULL;

/\*createTree(Root1);

createTree(Root2);\*/

char a1[] = { 'A','B','D', 'E','F', 'C', 'G' ,'H' };

char b1[] = { 'D', 'B','F','E', 'A','H','G', 'C', };

int n1 = 8;

binTreeNode \*Root1 = pre\_and\_mid\_construct(a1, b1, n1);

LevelOrder(Root1);

printf("\n");

char a[] = { 'A','C', 'G','H','B','E', 'F','D' };

char b[] = { 'G','H','C','A', 'F','E', 'B' , 'D' };

int n = 8;

binTreeNode \*Root2 = pre\_and\_mid\_construct(a, b, n);

LevelOrder(Root2);

printf("\n");

if (isomorphic(Root1, Root2))

printf("Yes\n");

else

printf("No\n");

system("pause");

return 0;

}

2.2-d树的插入生成与查找打印

#include <stdio.h>

#include <stdlib.h>

#include<string.h>

#define ElementType char

typedef struct TreeNode\* Tree;

Tree Insert(ElementType \*key1, ElementType \*key2, Tree tree, int deep);

void PrintTree(Tree tree, ElementType \*key1L, ElementType \*key1R, ElementType \*key2L, ElementType \*key2R, int deep);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*key1 : key1 分叉条件

\*key2 : key2 分叉条件

\*left : 左子树

\*right : 右子树

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

struct TreeNode {

ElementType \*Key1;

ElementType \*Key2;

Tree Left;

Tree Right;

};

//历任美国总统的名和姓

//key1 ： 名

//key2 ： 姓

char\* key1[] = { "Harry","Dwight","John","George","Gerald","Lyndon","Richard","Bill","Jimmy","Ronald" };

char\* key2[] = { "Truman","Eisenhower","Kennedy","Bush","Ford","Johnson","Nixon","Clinton","Carter","Reagan" };

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*key1 ： key1 分叉条件

\*key2 ： key2 分叉条件

\*tree ： 2 - d 树数据类型

\*deep ： 树的深度

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

Tree Insert(ElementType \*key1, ElementType \*key2, Tree tree, int deep) {

if (tree == NULL) {

Tree thisNode = (Tree)malloc(sizeof(struct TreeNode));

thisNode->Left = NULL;

thisNode->Right = NULL;

thisNode->Key1 = key1;//给地址

thisNode->Key2 = key2;//给地址

return thisNode;

}

else if (deep % 2 == 1)//奇数层以KEY2比较

{

if (strcmp(key1, tree->Key1) < 0)//字符串比较

{

tree->Left = Insert(key1, key2, tree->Left, deep + 1);

}

else { tree->Right = Insert(key1, key2, tree->Right, deep + 1); }

}

else {

if (strcmp(key2, tree->Key2) < 0) { tree->Left = Insert(key1, key2, tree->Left, deep + 1); }

else { tree->Right = Insert(key1, key2, tree->Right, deep + 1); }

}

return tree;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*tree : 2 - d 树的基本数据类型

\*key1L : Low1

\*key1R : High1

\*key2L : Low2

\*key2R : High2

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void PrintTree(Tree tree, ElementType \*key1L, ElementType \*key1R, ElementType \*key2L, ElementType \*key2R, int deep) {

if (tree == NULL)

return;

char \*keyL, \*keyR;

char \*treeKey;

if (deep % 2 == 1)//下一层是偶数层，判断key1

{

treeKey = tree->Key1;

keyL = key1L;

keyR = key1R;

}

else {

treeKey = tree->Key2;

keyL = key2L;

keyR = key2R;

}

if (strcmp(treeKey, keyL) < 0)//不符合，递归左子树

{

PrintTree(tree->Left, key1L, key1R, key2L, key2R, deep + 1);

}

else if (strcmp(treeKey, keyR) > 0)//不符合,递归右子树

{

PrintTree(tree->Right, key1L, key1R, key2L, key2R, deep + 1);

}

else if ((strcmp(tree->Key1, key1L) >= 0) && (strcmp(tree->Key1, key1R) <= 0) && (strcmp(tree->Key2, key2L) >= 0) && (strcmp(tree->Key2, key2R) <= 0))

{

printf("\n%s %s", tree->Key1, tree->Key2);

PrintTree(tree->Left, key1L, key1R, key2L, key2R, deep + 1);

PrintTree(tree->Right, key1L, key1R, key2L, key2R, deep + 1);

}

}

void main() {

Tree tree = NULL;

for (int i = 0; i < 10; i++)

{

tree = Insert(key1[i], key2[i], tree, 1);

}

char \*Low1 = "Ca"; char \*High1 = "Kd"; char \*Low2 = "Dsas"; char \*High2 = "Zsd";

printf("People whose key1 between %s and %s, and key2 between %s and %s\n", Low1, High1, Low2, High2);

PrintTree(tree, Low1, High1, Low2, High2, 1);

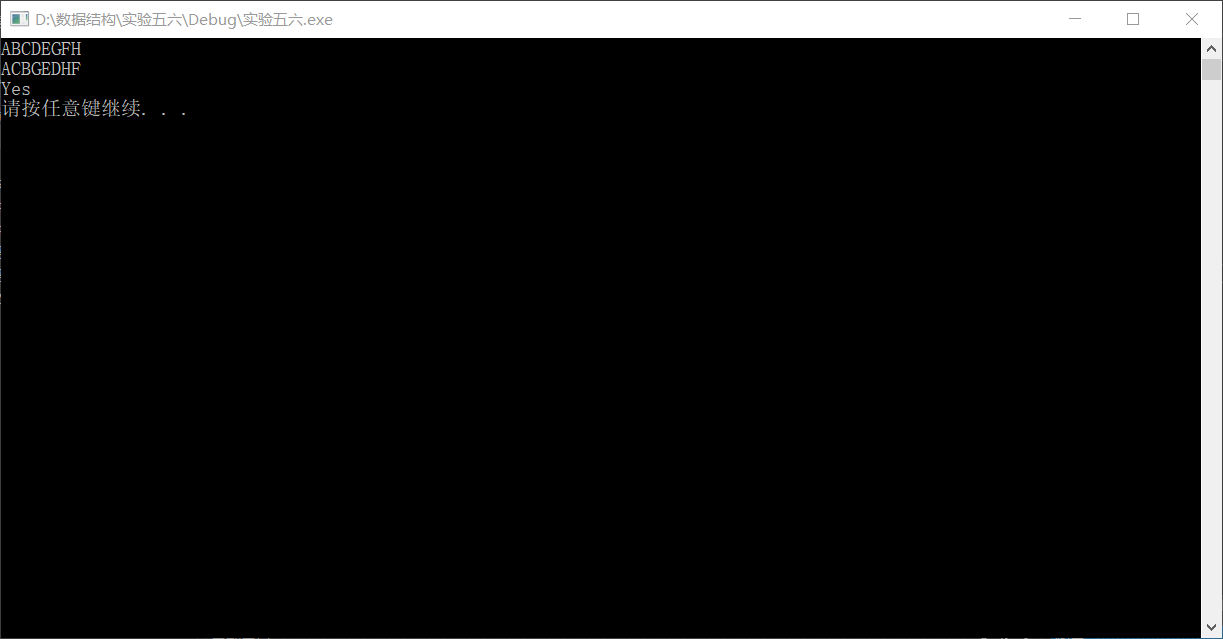
printf("\n");

system("pause");

}

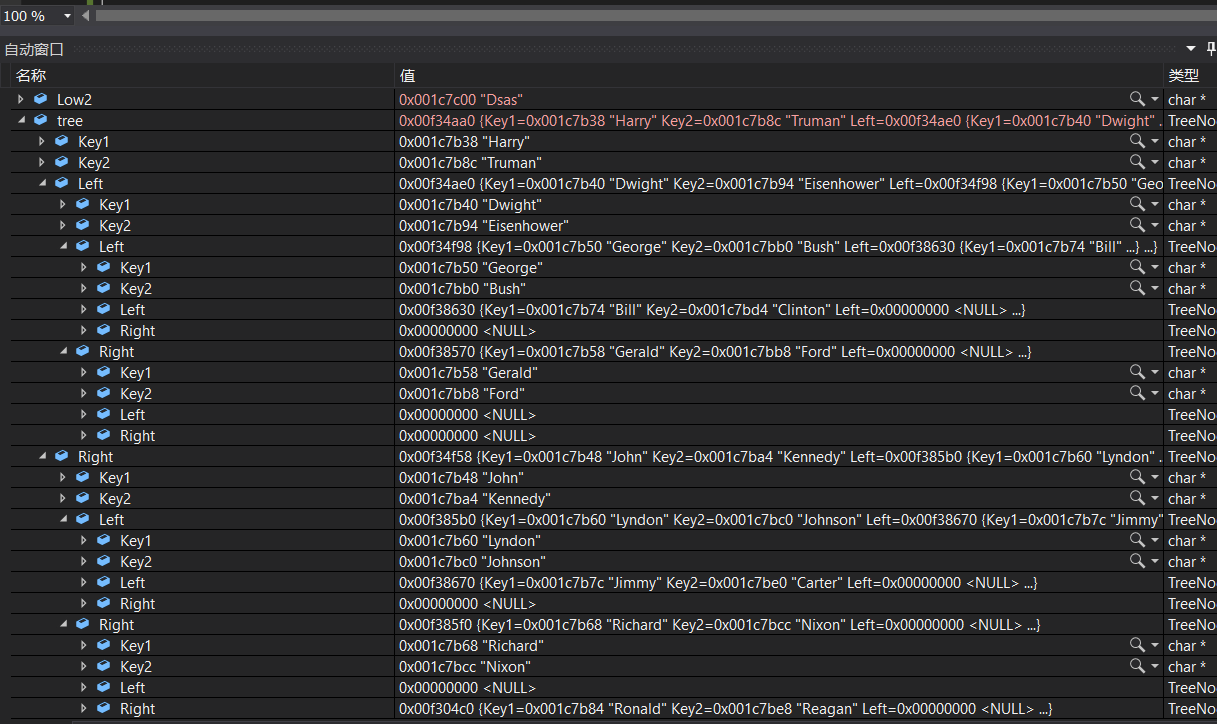
四、实验结果

1.判断给定树是否同构

.

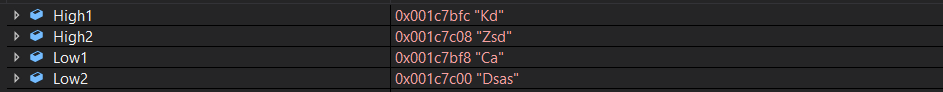
2.2-d树

（1）验证生成的2-d树结构

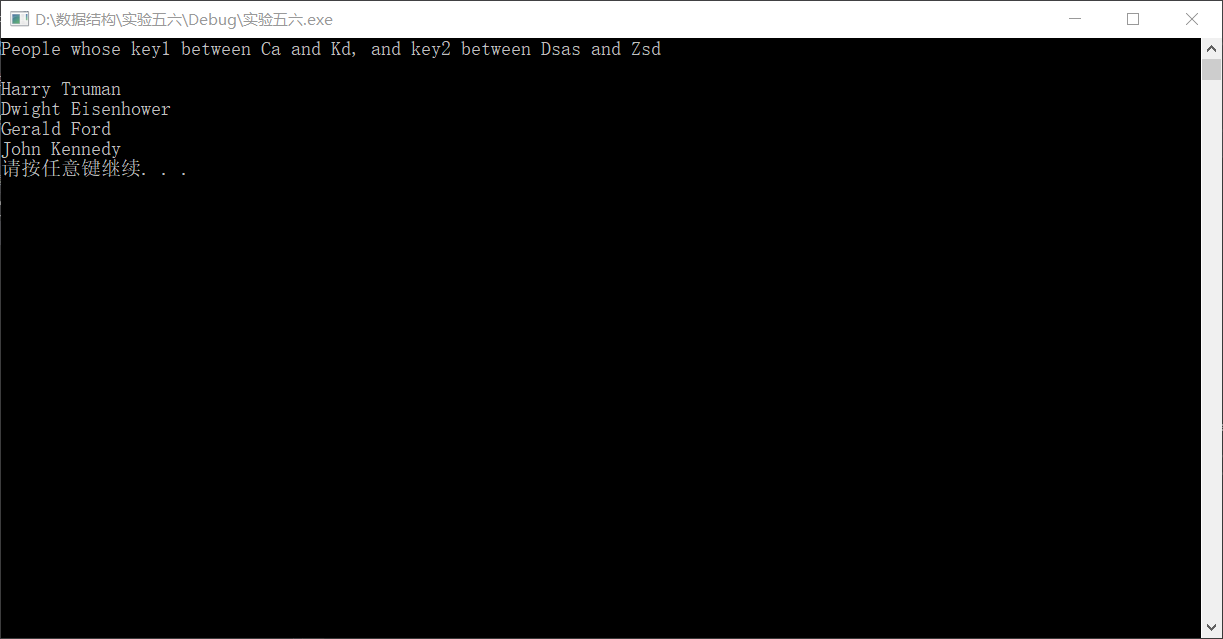


经验证生成的二叉查找树与例题给出的结果一致；

1. 打印满足Ca<=key1<=Kd,Dsas<=key2<=Zsd的节点



1. 打印结果



五、实验分析与结论

1.两个题目均正常实现了功能；

2.对于判断树的同构，普通算法需要递归的次数过多，效率低；改进：ahu算法

3.2-d树的操作与普通二叉树无太大差别，只是在插入、查找时要根据层数选择key值进行判断。

4.题1、2的时间复杂度均为O（logn）:递归次数与树的高度成正比。