数据结构实验六

姓名： 刘俊傲　　学号： **U201617047**　　班级： 软工**1603**

1. 判断两棵给定的树是否同构

1. 问题描述：

   > 如果树 T1 通过交换其（某些）节点的左右儿子变换成树 T2 ，则称树 T1 与树 T2 同构。

2. 问题分析与算法设计：

   > 1. 如果两棵树为空，则同构
   > 2. 如果一棵树为空，另一棵树不为空，则不同构
   > 3. 如果两棵树根节点的数据不同，则不同构
   > 4. 如果两棵树左节点都为空，则比较右节点
   > 5. 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

3. 算法实现：

```
//树同构的判定

#include<stdio.h>
#include<stdlib.h>

#define YES 1
#define NO 0

typedef struct DTree* tree;
typedef char ElementType;

void createTree(tree tmp);
int isomorphic(tree tree1, tree tree2);

/*************************
    *定义树的结构体
    *data : 树存储的数据
    *left : 树的左儿子
    *right : 树的右儿子
*************************/
struct DTree
{
    ElementType data;
    tree left;
    tree right;
};

/*************************
    *创建一个二叉树
    *tmp : 树的基本结构
*************************/
void createTree(tree tmp)
{
    ElementType ch;
    scanf("%c", &ch);

    if (ch == ' ')
    {
        tmp = NULL;
    }
    else
    {
        tmp = (tree)malloc(sizeof(DTree));
        if (tmp == NULL)
        {
            return;
        }
        tmp->data = ch;
        createTree(tmp->left);
        createTree(tmp->right);
    }
}
```

```
/*************************
    *判定两棵树是否为同构树
    *tree1 : 一棵待判定树
    *tree2 : 另一棵待判定树
*************************/
int isomorphic(tree tree1, tree 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->data != tree2->data)
    {
        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->data == tree2->data))
    {
        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);
    }
}
```

2. 2-d 树的插入与查找

1. 问题描述：

    1. 在偶数层使用 key1 来分叉，在奇数层使用 key2 来分叉
    2. 编写一个高效的打印，满足约束 Low1 <= key1 <= High1 和 Low2 <= key2 <= High2 的树的所有记录

2. 问题分析与算法实现

    1. 类比于二叉查找树，将在偶数层使用 key1 条件来分叉，在奇数层使用key2 条件来分叉
    2. 通过比较相应节点的关键值与 high ， low的关系来判断递归左子树还是右子树

3. 算法实现：

```
//2-d Tree

#include <stdio.h>
#include <stdlib.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","Re
agan" };

/*************************
    *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 = malloc(sizeof(struct TreeNode));
        thisNode->Left = NULL;
        thisNode->Right = NULL;

        thisNode->Key1 = key1;//给地址
        thisNode->Key2 = key2;//给地址


        return thisNode;
```

```
    }
    if (deep % 2 == 1)//下一层是偶数层，以key1为标准
    {
        if (key1 < tree->Key1)//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 (key2 < tree->Key2)
        {
            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 (treeKey < keyL)//不符合，递归左子树
```

```
    {
        PrintTree(tree->Left, key1L, key1R, key2L, key2R, deep + 1);
    }
    else if (treeKey > keyR)//不符合,递归右子树
    {
        PrintTree(tree->Right, key1L, key1R, key2L, key2R, deep + 1);
    }
    else if ((tree->Key1 >= key1L) && (tree->Key1 <= key1R) &&(tree->Key2 >= key2L) &&
(tree->Key2 <= key2R))
    {
        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", Low1,
High1, Low2, High2);
    PrintTree(tree, Low1, High1, Low2, High2, 1);
}
```