**数据结构实验报告**

姓名： 杨家玺 学号：U201717007 班级：软工1703班

2018．12．17

**实验六 二叉树的同构、2-d树**

1. 实验描述
2. 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。
3. 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、实验环境

1. 开发环境： OS X

2. IDE： VSCode

3. 编译器： Clang 9.1.0 Apple LLVM

4. C标准： C11

三、问题分析

1. 判断两棵二叉树是否同构

树的同构：给定两棵树T1和T2。如果T1可以通过若干次左右孩子互换就变成T2，则我们称两棵树是“同构”的。

假设T1拥有儿子A和B，T2拥有儿子C和D，当T1与T2存储的值相同的情况下，T1与T2同构的情况有：

1. 本位：A同构于C，B同构于D
2. 交换：A同构于D，B同构于C

所以该问题可以划分为自相似的子问题，原问题的解由众多子问题转移而来。

原问题：T1和T2是否同构

问题分解：

1. 根处的值是否对应相等
2. 左右子树是否满足同构条件

平凡情况：

1. 两树根均为空 -> 同构
2. 一个为空一个不为空 -> 不同构
3. 两树根存储的值不相同 -> 不同构

基于这种递归的思想，可以很轻易地编写出代码。

2. 基于BST的2-d树

2-d树在偶数层用元素1分裂，奇数层用元素2分裂，树根为0层。考虑到层数为或，故利用逻辑取反符!进行奇偶层判断。同时，每个节点维护一个长度为2的数组，这样就能用0或1直接访问数组内容，不用针对奇偶层分情况编写代码。

对于元素的插入与范围查找，使用函数指针便可以轻易地处理数据类型不同的用例，提高了程序的扩展性与泛化性。

四、算法实现

1. 判断两棵二叉树是否同构

Bool Isomorphic(Tree t1, Tree t2)

{

if (t1 == NULL && t2 == NULL)

return True;

if (t1 != NULL && t2 != NULL && t1->Element == t2->Element)

return (Isomorphic(t1->Left, t2->Left) &&

Isomorphic(t1->Right, t2->Right)) ||

(Isomorphic(t1->Left, t2->Right) &&

Isomorphic(t1->Right, t2->Left));

return False;

}

char\* IsTreeIsomorphic(Tree t1, Tree t2)

{

return Isomorphic(t1, t2) ? "[True] Isomorphic" : "[False] Not Isomorphic";

}

2. 基于BST的2-d树

// 递归插入节点

static KDTree InsertRec(ItemType Item, KDTree T, int Level, Comparator cmp)

{

if (T == NULL)

{

T = CreateNode(NULL, Item, NULL);

}

else if (cmp(Item[Level], T->Data[Level]) <= 0)

T->Left = InsertRec(Item, T->Left, !Level, cmp);

else

T->Right = InsertRec(Item, T->Right, !Level, cmp);

return T;

}

// 插入节点(暴露的接口)

KDTree TreeInsert(ItemType Item, KDTree T, Comparator cmp)

{

return InsertRec(Item, T, 0, cmp);

}

// 递归寻找元素

static void FindRec(ItemType Low, ItemType High, KDTree T, int Level, Comparator cmp)

{

if (T != NULL)

{

if (cmp(Low[0], T->Data[0]) <= 0 && cmp(T->Data[0], High[0]) <= 0 &&

cmp(Low[1], T->Data[1]) <= 0 && cmp(T->Data[1], High[1]) <= 0)

{

PrintData(T->Data);

endl();

}

if (cmp(Low[Level], T->Data[Level]) <= 0)

FindRec(Low, High, T->Left, !Level, cmp);

if (cmp(High[Level], T->Data[Level]) >= 0)

FindRec(Low, High, T->Right, !Level, cmp);

}

}

// 寻找符合范围的元素(暴露的接口)

void FindAndPrint(ItemType Low, ItemType High, KDTree T, Comparator cmp)

{

printf("Query X where \t");

PrintData(Low);

printf("\t<= X <=\t");

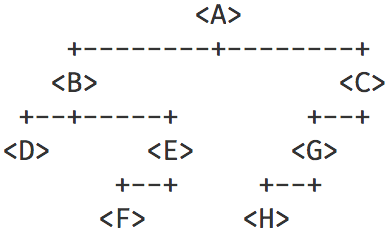
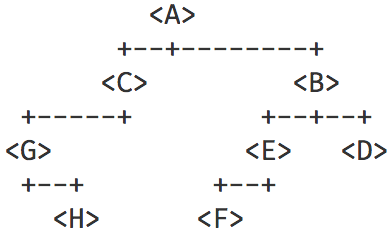
PrintData(High);

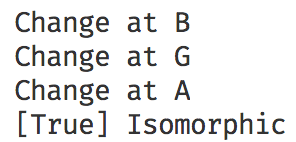
printf(" : \n");

FindRec(Low, High, T, 0, cmp);

}

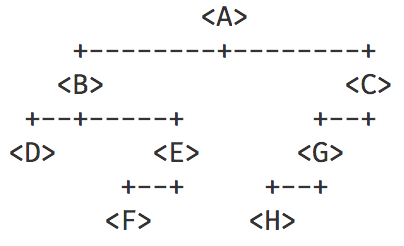
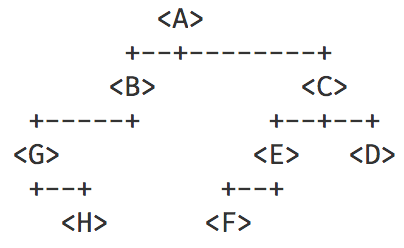
1. 实验结果与分析
2. 判断两棵二叉树是否同构
3. 同构

t1:  t2: 

运行结果：

分析：在依次交换节点B、G、A的子树后，两树完全相同，所以t1与t2同构

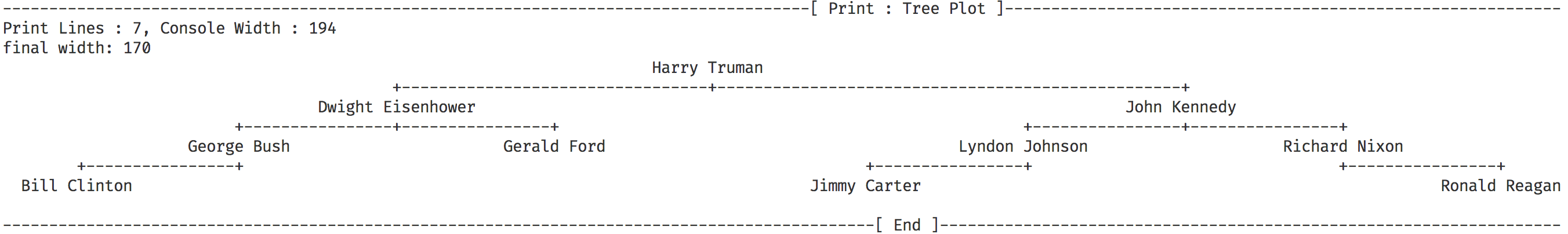
1. 不同构

t1:  t2: 

运行结果：

2. 基于BST的2-d树

(1) 样例:



(2) Query:

