

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

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)。

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 $2k$ 或 $2k + 1$ ，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();
        }

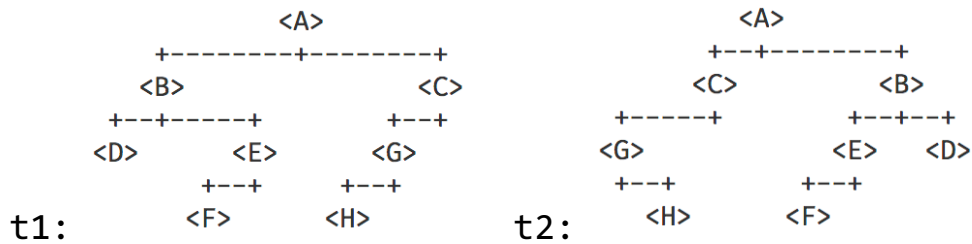
        if (cmp(Low[Level], T->Data[Level]) <= 0)
            FindRec(Low, High, T->Left, !Level, cmp);
        if (cmp(High[Level], T->Data[Level]) >= 0)
            FindRec(Low, High, T->Right, !Level, cmp);
    }
}

// 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);
}
```

五、实验结果与分析

1. 判断两棵二叉树是否同构

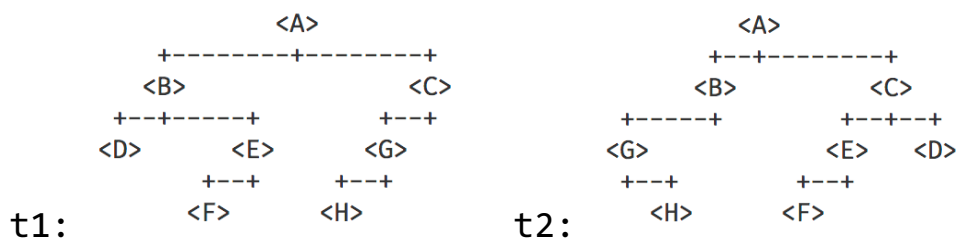
(1) 同构



Change at B
Change at G
Change at A
运行结果: [True] Isomorphic

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

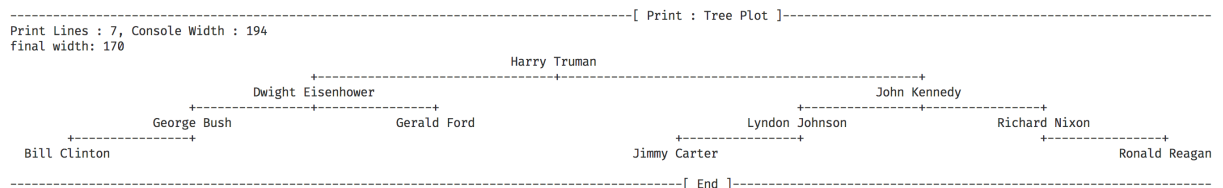
(2) 不同构



运行结果: [False] Not Isomorphic

2. 基于 BST 的 2-d 树

(1) 样例:



(2) Query:

