数据结构实验报告

姓名： 杨家玺 学号：U201717007 班级：软工 1703 班

2018. 12. 16

实验五 通过遍历重建二叉树、基于队列的二叉树层序遍历

一、 实验描述

1. 给出一棵二叉树的先序(或后序)遍历结果，以及中序遍历结果，如何构造这棵树?假定遍历结果以数组方式输入，请写出相应函数，判断是否存在生成同样遍历结果的树，如果存在，构造这棵树。
2. 二叉树的层序遍历。使用队列作为辅助存储，按树的结点的深度，从根开始依次访问所有结点。

二、实验环境

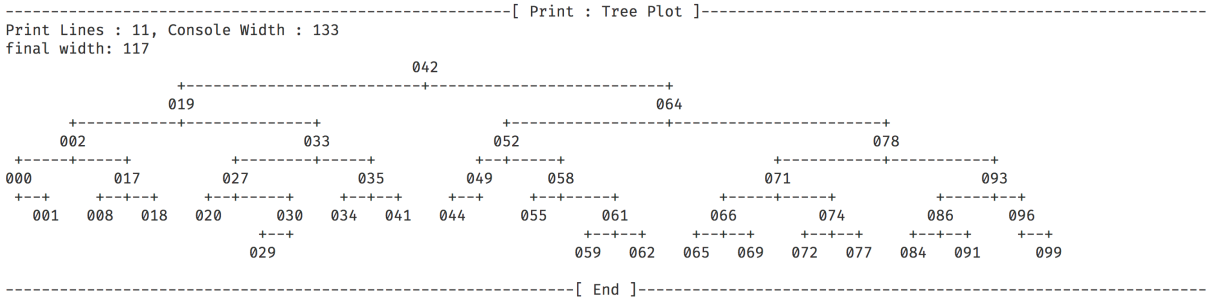
1. 开发环境： OS X
2. IDE： VSCode
3. 编译器： Clang 9.1.0 Apple LLVM
4. C 标准： C11

三、实验之前

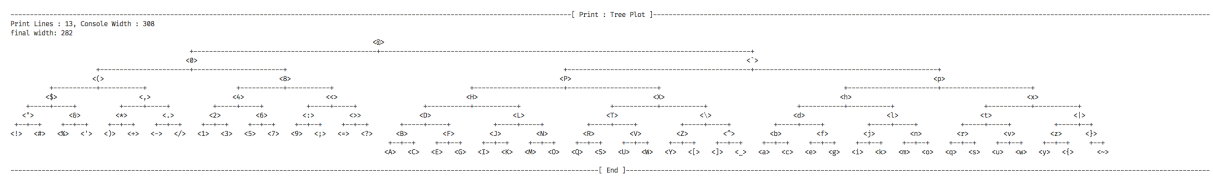
考虑到形象直观地展示树结构，我编写了一个在控制台打印输出二叉树的函数 `TreePlot(Tree T, char **format, int width)`，以便更好地展示实验结果。

效果展示：

1. 随机构建一棵50个元素的 Avl 树，节点$0 \leq x \leq 100$



2. 顺序插入 ASCII 码在33~126范围内的所有字符



三、问题分析

1. 通过遍历重建二叉树

对于四种遍历（先序、中序、后序、层序），只要给出的两个遍历中含有

中序遍历，则可唯一还原成一棵二叉树（如果遍历正确）。

大致思路：通过另外一种遍历确定根节点，从而把中序遍历划分为左子树遍历串+树根+右子树遍历串，然后对两个子树的遍历串使用同样的方法进行构建。最后将左子树，树根和右子树归入一个新建的节点，这便是算法的框架。

2．二叉树的层序遍历

首先将树根节点入队，然后开始进行基于队列的广度优先搜索：

0 ：如果队列不为空：

(1) 队首元素出队，赋值给临时变量 Tmp

(2) 输出 Tmp 所指节点存储的值

(3) 将 Tmp 的非空左右子树分别入队

(4) 返回 0

四、算法实现

1．通过遍历重建二叉树

```
//通过中序遍历和先序遍历恢复树
Tree InPreBuild(const char* InOrder, const char* PreOrder)
{
    char Root = PreOrder[0];
    int RootIdx = FindChar(InOrder, Root);
    int Left_n = RootIdx;
    int Right_n = strlen(InOrder) - RootIdx - 1;
    Tree LeftTree = NULL;
    Tree RightTree = NULL;
    if (Left_n > 0)
    {
        char* Left_In = StrCopy(InOrder, Left_n);
        char* Left_Pre = StrCopy(&PreOrder[1], Left_n);
        if (strlen(Left_In) > 1)
            LeftTree = InPreBuild(Left_In, Left_Pre);
        else
            LeftTree = CreateNode(NULL, Left_Pre[0], NULL);
        free(Left_In);
        free(Left_Pre);
    }
    if (Right_n > 0)
    {
        char* Right_In = StrCopy(&InOrder[RootIdx + 1], Right_n);
        char* Right_Pre = StrCopy(&PreOrder[RootIdx + 1], Right_n);
        if (strlen(Right_In) > 1)
            RightTree = InPreBuild(Right_In, Right_Pre);
        else
            RightTree = CreateNode(NULL, Right_Pre[0], NULL);
        free(Right_In);
        free(Right_Pre);
    }
    return CreateNode(LeftTree, Root, RightTree);
}
//通过中序遍历和后序遍历恢复树
Tree InPostBuild(const char* InOrder, const char* PostOrder)
{
    char Root = PostOrder[strlen(PostOrder) - 1];
    int RootIdx = FindChar(InOrder, Root);
    int Left_n = RootIdx;
```

```
    int Right_n = strlen(InOrder) - RootIdx - 1;
    Tree LeftTree = NULL;
    Tree RightTree = NULL;
    if (Left_n > 0)
    {
        char* Left_In = StrCopy(InOrder, Left_n);
        char* Left_Post = StrCopy(PostOrder, Left_n);
        if (strlen(Left_In) > 1)
            LeftTree = InPostBuild(Left_In, Left_Post);
        else
            LeftTree = CreateNode(NULL, Left_Post[0], NULL);
        free(Left_In);
        free(Left_Post);
    }
    if (Right_n > 0)
    {
        char* Right_In = StrCopy(&InOrder[RootIdx + 1], Right_n);
        char* Right_Post = StrCopy(&PostOrder[RootIdx], Right_n);
        if (strlen(Right_In) > 1)
            RightTree = InPostBuild(Right_In, Right_Post);
        else
            RightTree = CreateNode(NULL, Right_Post[0], NULL);
        free(Right_In);
        free(Right_Post);
    }
    return CreateNode(LeftTree, Root, RightTree);
}

// 根据 InOrder 提取 LevelOrder
char* FixLevel(const char* LevelOrder, const char* InOrder)
{
    int size = strlen(InOrder);
    char* res = malloc(sizeof(char) * (size + 1));
    int cnt = 0;
    int LO_len = strlen(LevelOrder);
    for (int i = 0; i < LO_len; i++)
    {
        if (FindChar(InOrder, LevelOrder[i]) != -1)
        {
            res[cnt++] = LevelOrder[i];
        }
    }
    res[cnt] = '\0';
    return res;
}
//通过中序遍历和层序遍历恢复树
Tree InLevelBuild(const char* InOrder, const char* LevelOrder)
{
    char Root = LevelOrder[0];
    int RootIdx = FindChar(InOrder, Root);
    int Left_n = RootIdx;
    int Right_n = strlen(InOrder) - RootIdx - 1;
    Tree LeftTree = NULL;
    Tree RightTree = NULL;
    if (Left_n > 0)
    {
        char* Left_In = StrCopy(InOrder, Left_n);
        char* Left_Level = FixLevel(LevelOrder, Left_In);
        if (strlen(Left_In) > 1)
        {
            LeftTree = InLevelBuild(Left_In, Left_Level);
        }
        else
            LeftTree = CreateNode(NULL, Left_Level[0], NULL);
        free(Left_In);
        free(Left_Level);
    }
    if (Right_n > 0)
    {
        char* Right_In = StrCopy(&InOrder[RootIdx + 1], Right_n);
        char* Right_Level = FixLevel(LevelOrder, Right_In);
        if (strlen(Right_In) > 1)
        {
            RightTree = InLevelBuild(Right_In, Right_Level);
        }
        else
            RightTree = CreateNode(NULL, Right_Level[0], NULL);
        free(Right_In);
        free(Right_Level);
    }
    return CreateNode(LeftTree, Root, RightTree);
}
```

2. 二叉树的层序遍历

```
// format : 输出格式
void LevelOrderTreeWalk(Tree tree, const char* format)
{
    Queue queue = InitQueue(NULL);
    InsertLast(tree, queue);
    TPosition Tmp = NULL;
    while (!IsQueueEmpty(queue))
    {
```

```
        Tmp = (TPosition)(RemoveFirst(queue));
        printf(format, Tmp->Element);
        if (Tmp->Left != NULL)
            InsertLast(Tmp->Left, queue);
        if (Tmp->Right != NULL)
            InsertLast(Tmp->Right, queue);
    }
    printf("\n");
    DeleteQueue(queue);
}
```

五、 实验结果与分析

1. 通过遍历重建二叉树

中序：ABCDEFGHIJKLM

(1) 先序：JDBACFEHGIKLM

(2) 后序：ACBEGIHFDMLKJ

(3) 层序：JDKBFLACEHMGI

运行结果均为：

```
-----------------------------------------[ Print : Tree Plot ]-----------------------------------------
Print Lines : 9, Console Width : 109
final width: 39
                          <J>
           +--------------+--+
          <D>                <K>
     +-----+-----+            +--+
    <B>         <F>             <L>
  +--+--+     +--+-----+         +--+
 <A>   <C>   <E>      <H>          <M>
                   +--+--+
                  <G>   <I>

-----------------------------------------------------[ End ]-----------------------------------------------------
```

2. 二叉树的层序遍历

树：向一棵 Avl 树依次插入字母 A~Z

运行结果：

```
-----------------------------------------------------[ Print : Tree Plot ]-----------------------------------------------------
Print Lines : 9, Console Width : 133
final width: 78
                                        <P>
                    +-------------------+-----------+
                   <H>                             <T>
          +--------+--------+                +-----+-----+
         <D>                 <L>             <R>         <X>
     +---+---+          +---+---+        +--+--+      +---+--+
    <B>     <F>        <J>     <N>      <Q>   <S>    <V>     <Y>
  +--+--+  +--+--+  +--+--+  +--+--+                +--+--+  +--+
 <A>   <C> <E>   <G> <I>   <K> <M>   <O>            <U>   <W>     <Z>

-----------------------------------------------------------[ End ]-----------------------------------------------------------

-----------------------------------------------------[ Level Order Traversal ]-----------------------------------------------------
 P  H  T  D  L  R  X  B  F  J  N  Q  S  V  Y  A  C  E  G  I  K  M  O  U  W  Z
-----------------------------------------------------------[ End ]-----------------------------------------------------------
```