**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**

1. 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

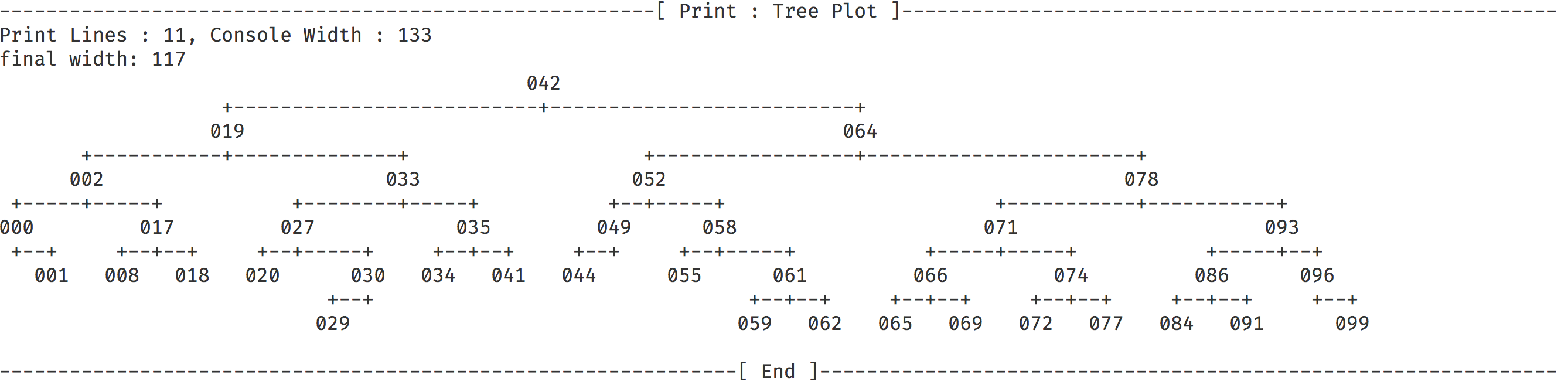
4. C标准： C11

三、实验之前

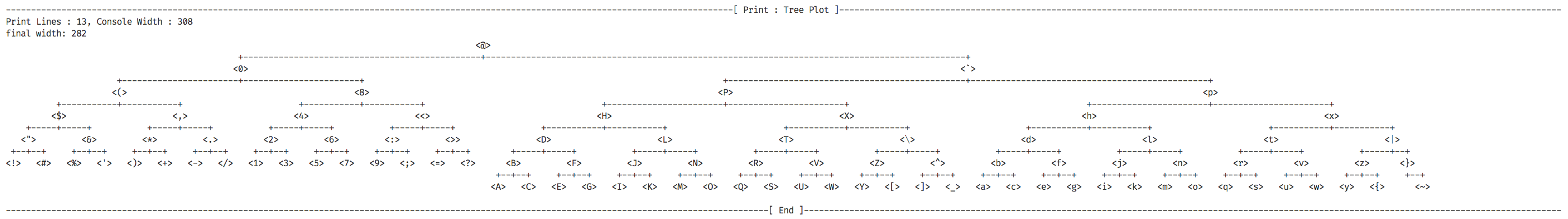
考虑到形象直观地展示树结构，我编写了一个在控制台打印输出二叉树的函数TreePlot(Tree T, char \*\*format, int width)，以便更好地展示实验结果。

效果展示：

1. 随机构建一棵个元素的Avl树，节点



1. 顺序插入ASCII码在范围内的所有字符



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);

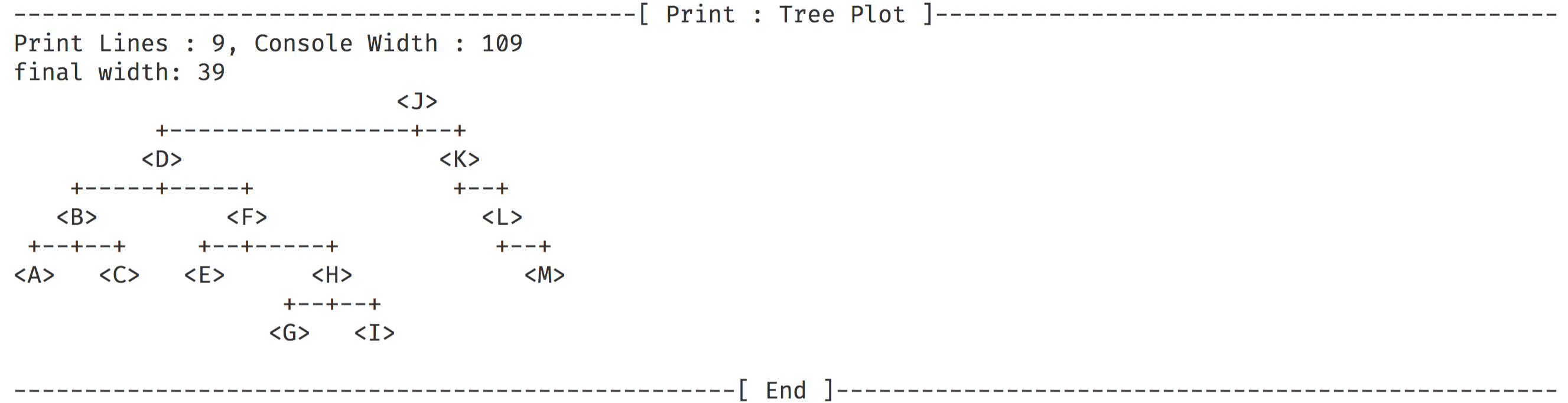
}

1. 实验结果与分析
2. 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：



1. 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：

