-



**《信息论与编码理论》**

课程大作业

**题目：** Huffman编码的Java实现

**专业：** 信息工程专业

**姓名：** 韩璐

**学号：** 16010520041

目录

[一、题目 3](#_Toc7280982)

[二、原理 3](#_Toc7280983)

[2.1概述： 3](#_Toc7280984)

[2.2 原理 3](#_Toc7280985)

[2.3 构造huffman编码 4](#_Toc7280986)

[三、代码及分析 5](#_Toc7280987)

[3.1程序说明： 5](#_Toc7280988)

[3.2 编程思想 5](#_Toc7280989)

[3.3 流程图 5](#_Toc7280990)

[3.4 node，二叉树的构建 5](#_Toc7280991)

[3.5 主要代码子模块及分析 7](#_Toc7280992)

[四、仿真结果 10](#_Toc7280993)

[4.1 仿真结果 10](#_Toc7280994)

[4.2 结果分析 10](#_Toc7280995)

[五、心得体会 11](#_Toc7280996)

[六、附录：文件目录 11](#_Toc7280997)

# 一、题目

实现一个给定离散随机变量集的Huffman编码。

# 二、原理

### 2.1概述：

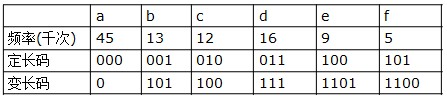
1952年, David A. Huffman提出了一个不同的算法，这个算法可以为任何的可能性提供出一个理想的树。香农-范诺编码（Shanno-Fano）是从树的根节点到叶子节点所进行的的编码，哈夫曼编码算法却是从相反的方向，暨从叶子节点到根节点的方向编码的。

为每个符号建立一个叶子节点，并加上其相应的发生频率

哈夫曼编码是广泛地用于数据文件压缩的十分有效的编码方法。其压缩率通常在20%～90%之间。哈夫曼编码算法用字符在文件中出现的频率表来建立一个用0，1串表示各字符的最优表示方式。

### 2.2 原理

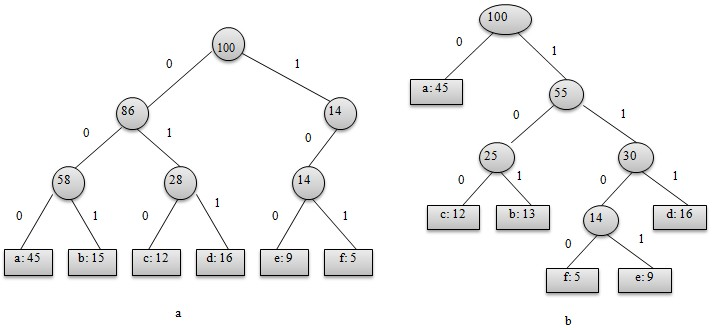
一个包含100,000个字符的文件，各字符出现频率不同，如下表所示。



有多种方式表示文件中的信息，若用0,1码表示字符的方法，即每个字符用唯一的一个0,1串表示。若采用定长编码表示，则需要3位表示一个字符，整个文件编码需要300,000位；若采用变长编码表示，给频率高的字符较短的编码；频率低的字符较长的编码，达到整体编码减少的目的，则整个文件编码需要（45×1+13×3+12×3+16×3+9×4+5×4）×1000=224,000位，由此可见，变长码比定长码方案好，总码长减小约25%。

前缀码：对每一个字符规定一个0,1串作为其代码，并要求任一字符的代码都不是其他字符代码的前缀。这种编码称为前缀码。编码的前缀性质可以使译码方法非常简单；例如001011101可以唯一的分解为0,0,101,1101，因而其译码为aabe。

译码过程需要方便的取出编码的前缀，因此需要表示前缀码的合适的数据结构。为此，可以用二叉树作为前缀码的数据结构：树叶表示给定字符；从树根到树叶的路径当作该字符的前缀码；代码中每一位的0或1分别作为指示某节点到左儿子或右儿子的“路标”。



从上图可以看出，表示最优前缀码的二叉树总是一棵完全二叉树，即树中任意节点都有2个儿子。图a表示定长编码方案不是最优的，其编码的二叉树不是一棵完全二叉树。在一般情况下，若C是编码字符集，表示其最优前缀码的二叉树中恰有|C|个叶子。每个叶子对应于字符集中的一个字符，该二叉树有|C|-1个内部节点。

给定编码字符集C及频率分布f,即C中任一字符c以频率f(c)在数据文件中出现。C的一个前缀码编码方案对应于一棵二叉树T。字符c在树T中的深度记为dT(c)。dT(c)也是字符c的前缀码长。使平均码长达到最小的前缀码编码方案称为C的最优前缀码。

### 2.3 构造huffman编码

当有一个以上的节点存在时，进行下列循环:

1.把这些节点作为带权值的二叉树的根节点，左右子树为空

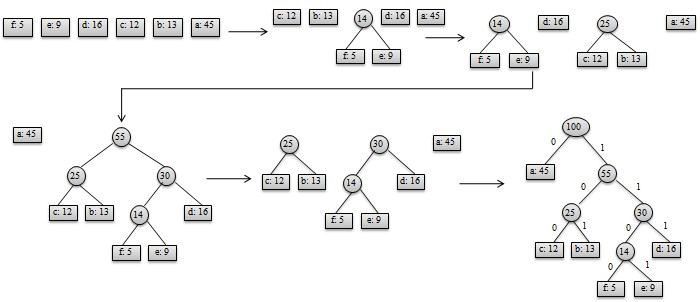
2.选择两棵根结点权值最小的树作为左右子树构造一棵新的二叉树，且至新的二叉树的根结点的权值为其左右子树上根结点的权值之和。

3.把权值最小的两个根节点移除

4.将新的二叉树加入队列中.

最后剩下的节点暨为根节点，此时二叉树已经完成。

构造过程如图所示：



# 三、代码及分析

### 3.1程序说明：

为了方便操作，本程序通过读取工程根目录下huffman.txt中包含的字符，统计各个字符出现的频数，然后在控制台输出相对应字符的频数以及编码。并存储到encodinghuffman.txt文件下。

### 3.2 编程思想

首先构建一个readString函数从文本读取所有字符传递给source这个变量，通过init函数对source这个字符串变量通过map存储键值对的方式进行统计各个出现的字符以及出现的权重，因为map是以键值对形式存储，且一个字符只能存储一次。统计完之后c代表字符，count代表c出现的频数。再将c赋给node.charData。count赋给node.key。相当于建了好多node，都存储着字符以及频数。

有了node就要构建二叉树。tress是一个java.util. PriorityQueue,是一个具有优先级的队列。根据重写comparable接口按出现权重进行顺序排列。while循环是每循环一次，从trees中删除已添加root（根）的node。（开始合并，根据key排序的最想的两个node合并为一个新的node。新的node是两个key的之和）。最后合并的trees的size只剩下一个，停止循环。

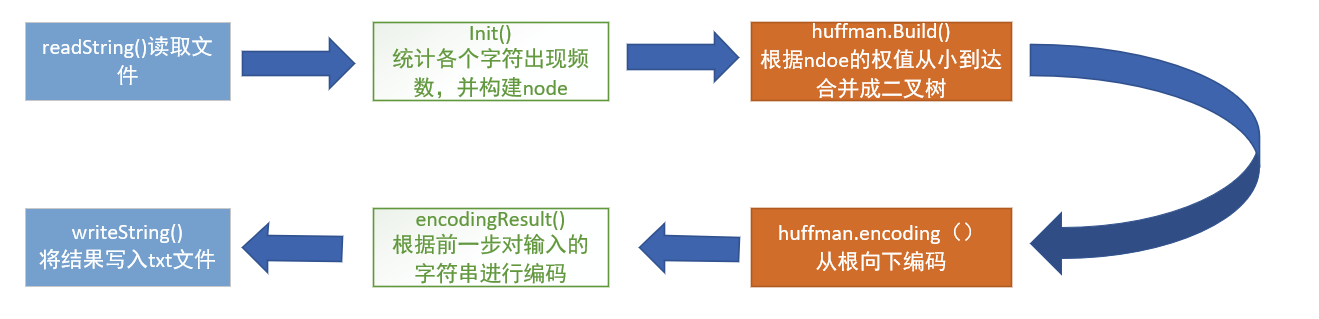
编码，encoding函数，读取每个node，从根开始，左孩子编0，右孩子编1.

displayEncodeing，输出所有叶子的键值对。k代表符号，v是编码。

整体字符编码输出，encodingResult,开始从输入的source字符每个字符开始读取，根据键值对开始编码，输出到StringBuilder中。

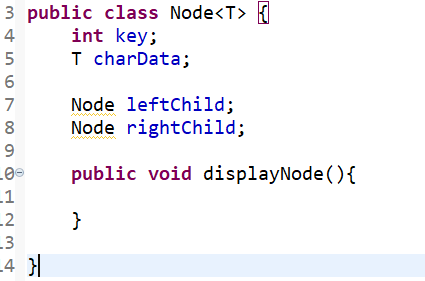
输出writeString函数，将得到的结果encodingResult存储到encodinghuffman.txt中。

### 3.3 流程图



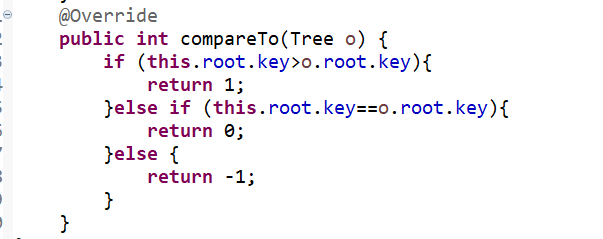
### 3.4 node，二叉树的构建

Node:作为一个节点，需要有相应的权重和字符，key表示权重，charData表示这个node对应的字符。节点需要有两个孩子，为leftChild和rightChild.,其中这两个孩子类型也是Node

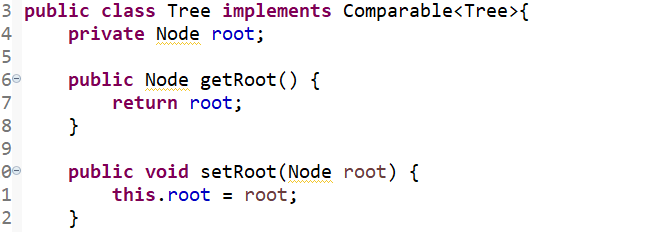


树：根据huffman树需要最小两个node依次合并。重新排然后再合并。所以需要构建一个有“顺序”的队列。这就是PriorityQueue，需要重写compareTo的方法让它从小到大排序。

compareTo()是接口java.lang.Comparable<T>中的方法，当某个类需要进行比较时，该类需要实现Comparable<T>接口，并且必须重写public int compareTo(T t)方法。compareTo()方法返回值为int类型，就是比较两个值，如：x.compareTo(y)。如果前者大于后者，返回1，前者等于后者则返回0，前者小于后者则返回-1。这样就可以根据权重key进行排序。

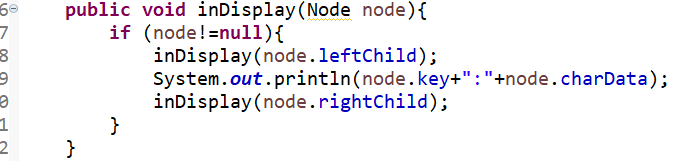


树有个属性root。需要有得到该属性的方法getRoot()和赋值的方法setRoot().root代表这个node的父亲。



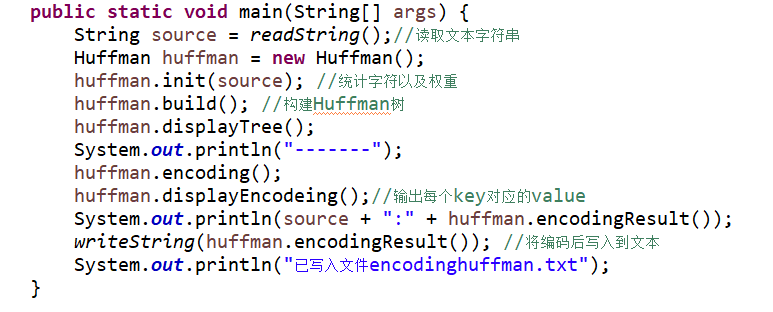
输出对应树的相应编码的方法：

如果node节点不为null。那么就有两个孩子，一直递归，从根到叶，打印相应的权重key和代表的字符charData

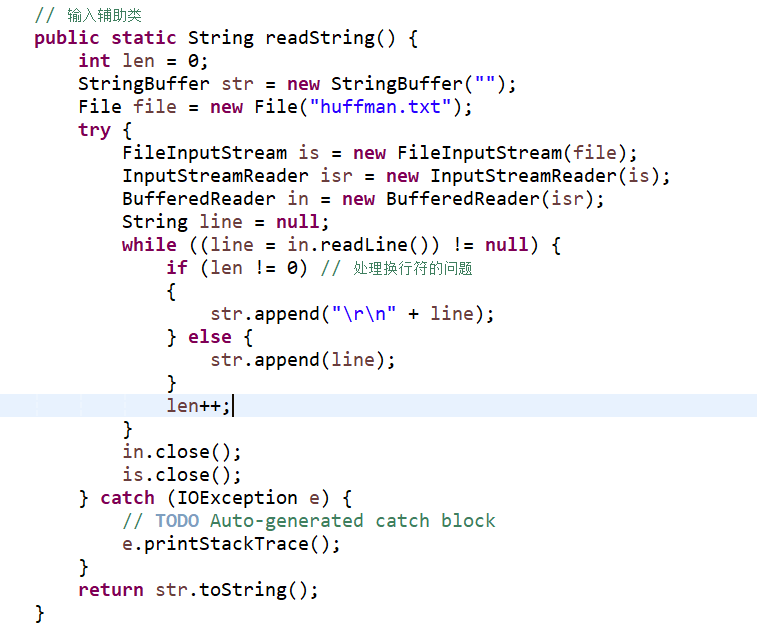


### 3.5 主要代码子模块及分析

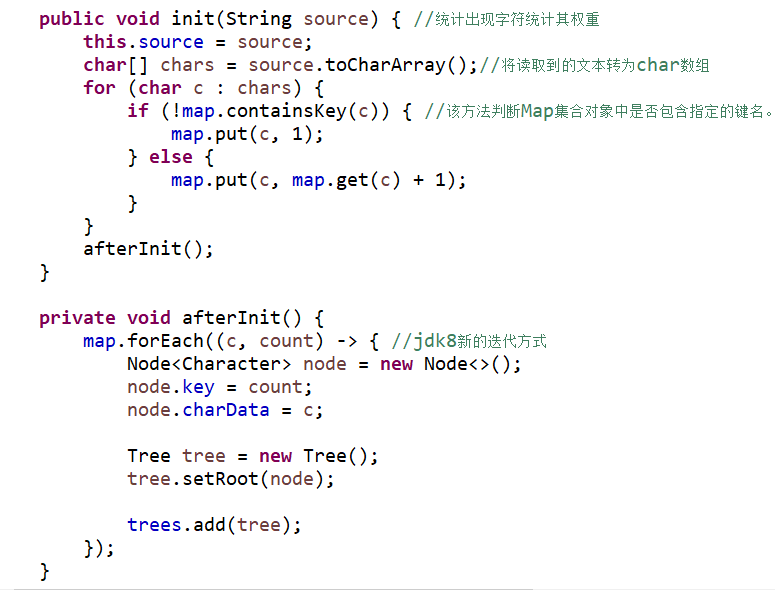
Main函数，可见跟流程图基本一致。



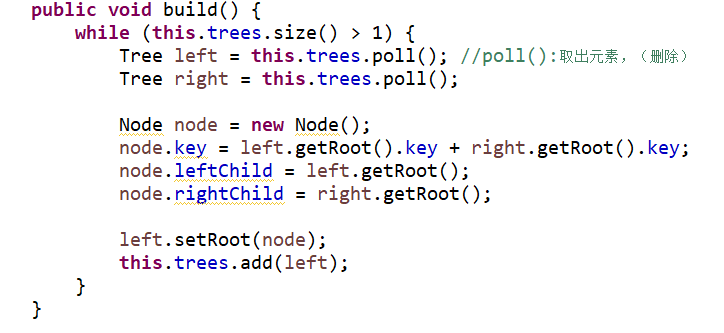
输入辅助类，将文本的huffman.txt 中的字符串读取到String中



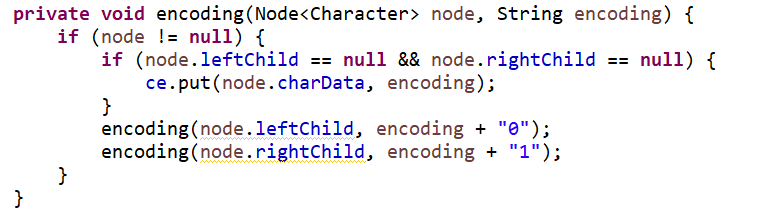
将读取进来的字符串进行分割成char数组，并存入，遍历chars进行字符和其频数的统计，如果字符没有出现过，就在map插入这个字符，（map是以键值对的方式存储数据），并将其值置为1，如果在map出现了，那么就把值+1.达到统计频数的作用。afterInit()的作用是将所有在map中出现的键值对，构建成node的形式。



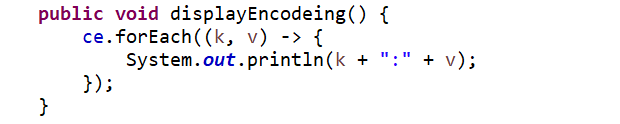
构建Huffman树，利用有优先级的队列根据node的key（频数）排列，然后开始从最小读取两个node，读取后删掉。构建成一个新的node存储进去，其key是原本的两个node的之和。这样经过遍历，最后只剩下一个node。trees.size()=1.停止循环。



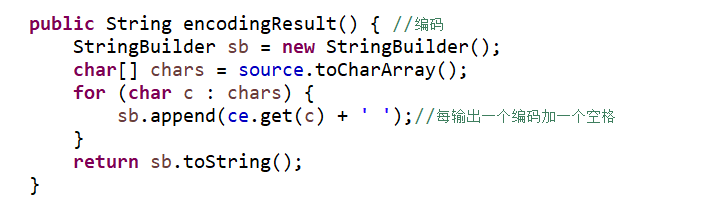
进行Huffman编码，从根部往下开始编码。leftChild置0，rightChild置1.



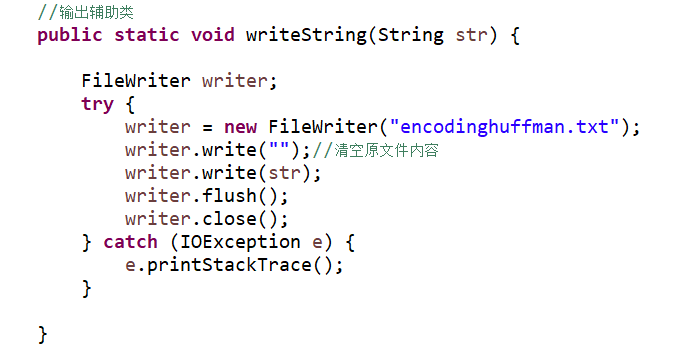
输出每个键值对，遍历所有key和value，输出



输出整个的编码。再次读取source，分成一个个char的数组，开始遍历。利用StringBuilder构建输出的字符串，每次读取一个char，就对照key,value进行编码，将编后的值舔加在sb的后边。



输出辅助类，将一个String写入到文件中。



# 四、仿真结果

### 4.1 仿真结果

文本内字符串：

“ABBBBBBBBBBBBBBBBCCCCCCCCCCCCCCCCCCCCDDDDDDDDDDDDDDDDDEEEEEEEEEFFFFF”

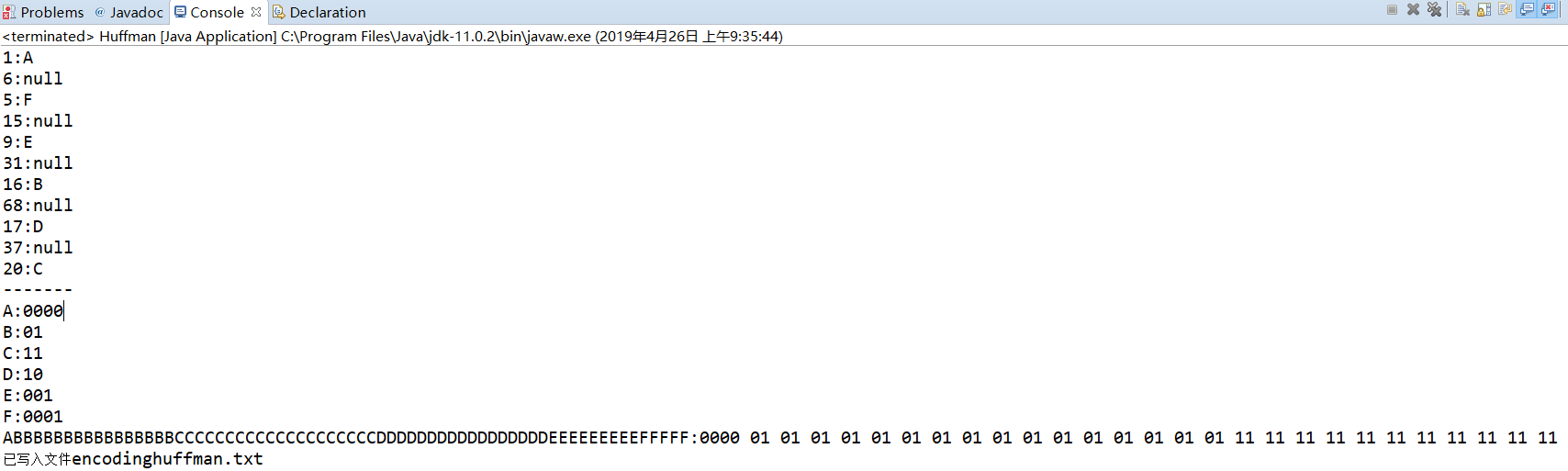
输出：

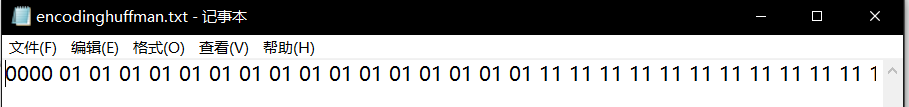
输出各个字符对应的编码，前边的数字是出现频数，后边是字符。

并已存入txt文件

01 01 01 01 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 001 001 001 001 001 001 001 001 001 0001 0001 0001 0001 0001

控制台首先输出了每个字符对应的权重，（前边的数字是权重），null的表示是由两个孩子合并而来的节点。然后分割线下边输出了每个字符对应的编码，最下是整体的编码，并存入到了文件中。



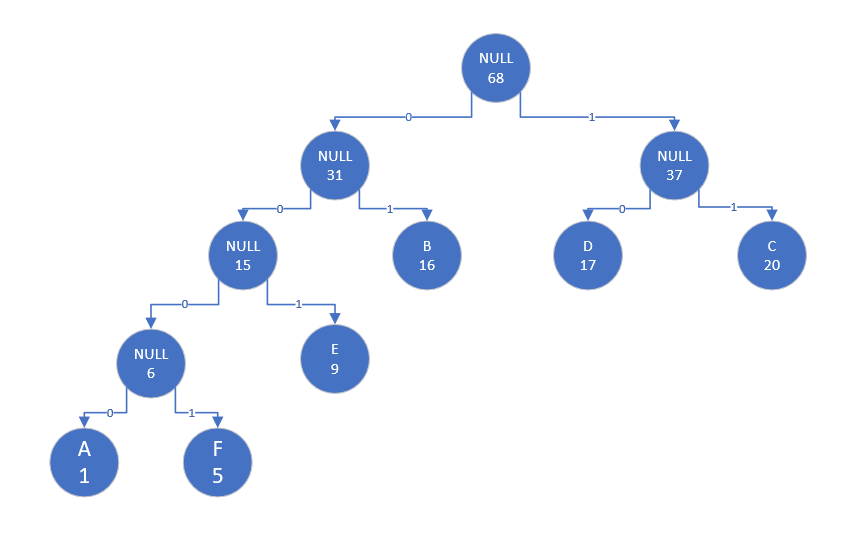


### 4.2 结果分析

相对应的二叉树编码：

根据node的key从小到大排序，依次从小到大合并构建二叉树。编码时二叉树左边置0，右边置1. 程序编出的码与自己画图编的一致，如下图。可见程序正确。完成老师给定的目标。输出了各个node的权重和字符，以及每个字符对应的编码，和总的编码。并存储到txt中。

下图为此程序输入的huffman树，上方为字符，下边数字是权重。中间连的线。左边是0，右边是1.



# 五、心得体会

通过这次动手编码。更深刻的了解了Huffman编码的过程，以及更熟悉了编程的思想以及操作，锻炼了自己的动手能力。也思考了如何将老师给出的题目进行更好的操作，更易于别人使用。所以采用了读取文本流。

回顾此次完成大作业的点点滴滴，我们有太多的收获，过程是痛苦的，结果是收获的，这就是我这个月最大的感受，我们是在发现问题和解决问题中不断进步。虽然在这次设计遇到了不少的困难和问题，但在同伴的共同努力，辛苦的去钻研，去学习，最终都克服了这些问难，使问题得到了解决。

# 六、附录：文件目录

.git 是以上传到GitHub创建的本地记录，我的GitHub：<https://github.com/karlhl/code.Huffman>

.bin 存放本地class文件

src 存放java源代码，我是用eclipse编写的。

huffman.txt存放要编码的字符

encodinghuffman.txt 存放编码字符

huffman.vsdx 是visio创建的流程图 插入在本报告点3.3

二叉树.vsdx 是visio创建的本文构建的二叉树 插入在本报告4.2