**信息论与编码课程设计报告**

设计题目： 信源熵与哈夫曼编码

专业班级 电信1403班

学 号 311408000728

学生姓名 张鹏举

指导教师 成凌飞

教师评分

2017年　5月　24日

**目录**

[一、设计任务与要求 2](#_Toc483425049)

[二、设计思路 2](#_Toc483425050)

[1、总体架构设计 2](#_Toc483425051)

[2、程序编写思路 2](#_Toc483425052)

[三、设计流程图 2](#_Toc483425053)

[四、程序运行及结果 2](#_Toc483425054)

[五、心得体会 2](#_Toc483425055)

[参考文献 2](#_Toc483425056)

[附录：源程序 2](#_Toc483425057)

# 

# 一、设计任务与要求

通过本次课程设计的练习，使学生进一步巩固信源熵、信源编码的基本原理，掌握具体的编码方法，熟悉编程软件的使用，培养学生自主设计、编程调试的开发能力，同时提高学生的实践创新能力。

1、统计信源熵

要求：统计任意文本文件中各字符（区分大小写）数量，计算字符概率，并计算信源熵。

2、哈夫曼编码

要求：任意输入消息概率，利用哈夫曼编码方法进行编码，并计算信源熵和编码效率。

# 二、设计思路

## 1、总体架构设计

本次课程设计选择的两个题目，原因是它们之间有一定的关联性和挑战性，设计完成后将会产生一个系统，将两个题目联系到一起。该系统将任意读取一篇只包含有英文的Word文档，将文档中的字符个数按照英文字母表进行统计（区分大小写），并计算对应字符的概率以及信源熵，接下来对统计处理后的概率进行哈夫曼编码，分别计算出各个字符对应哈夫曼编码的码字，最后计算出来平均码长和编码效率。

在实际操作过程中，使用Python进行代码的编写，以及数据的计算处理，将数据计算后，并写入到EXCEL表格当中，用于数据的保存以及再处理。在得到各个符号的概率后，在程序中自动对数据进行处理，使得字符按照字母表顺序进行排列，并绘制字符概率的柱状图和饼状图。

具体的设计步骤如下：

1、寻找任意一篇全英文Word文档，进行字符的个数进行统计。

2、计算出每个字母出现的次数（区分大小写），并计算符号概率。

3、通过信源熵公式计算信源熵并绘制符号概率的柱状图和饼状图。

4、使用哈夫曼编码对各个符号进行编码。

5、计算编码效率并将所需数据数据写入到Excel表格中。

## 2、程序编写思路

程序分为三个文件，依次为Main.py、Comput.py、Huffman.py。第一个文件为主文件，将整个工程结合到一起，读入数据、处理数据并导出数据。第二个文件为计算和处理相关数据。第三个文件主要是进行Huffman编码使用的。这样，程序之间实现了模块化，使得整个工程的架构更加的清晰，程序编写的思路也更加明了。

在数据计算和处理中，用到了以下公式。

信源熵的计算：

平均码长的计算：

编码效率的计算：

其中为符号概率，为每个码字的长度。

在进行哈夫曼编码时，使用了面向对象的编程思想，在课堂上我们得知哈夫曼在进行编码时实质上是在进行二叉树的绘制，而且哈弗曼树其实就是最优二叉树，所以我们可以将树中的每个结点当做一个对象，在这个对象中，它既有父结点，也有左子结点和右子结点。首先我们要在程序中在逻辑上表达这个二叉树，

我们首先要将得到的概率按照从大到小进行排列，每个概率都相当于一个节点。然后进行以下步骤：

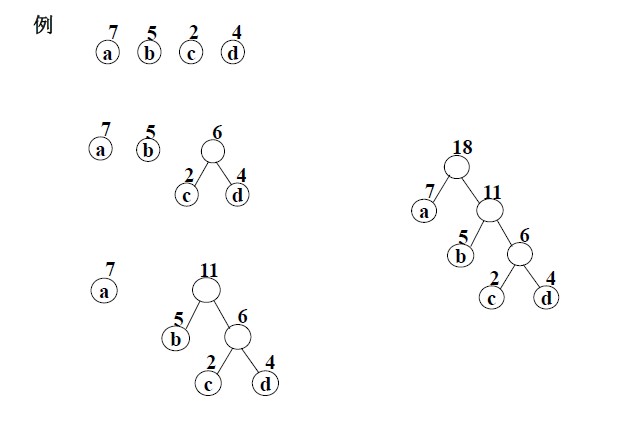
1、首先将寻找两个最小概率进行合并，合并两个结点后成为一个结点。

2、我们将合并后的结点当做原来两结点的父结点，三个结点组成的二叉树的路径赋值0和1。

3、循环上面的两个步骤，直到整个集合中只有一个结点。

4、自底向上得到哈弗曼树，且每个父结点到子结点路径值都为0或1，并且根结点到叶结点路径唯一。

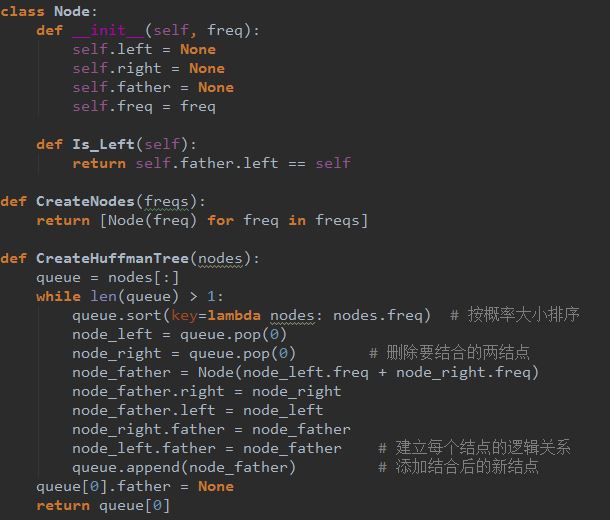
这样我们就构建了一个哈夫曼树，且每个树根结点和树叶结点之间路径唯一且平均路径最短。



接下来，我们要寻找每个概率对应的码字，其实码字就是根结点和叶结点之间路径上的0和1的值。码字是从根结点开始一直读到叶结点。

一般在我们人的思维中，从上向下读取比较简单，但是对于计算机而言，越是重复性的工作，在它看来就越简单，既然每个结点视为一个对象，那么在这个对象中，可以写一个判断自身为左子结点或右子结点的方法。然后对每个结点进行判断，从底层向上判断，若是左子结点那么路径值为0，相反若是右结点那么路径值为1。

于是这样我们就得到了各个字符及与之对应的哈夫曼编码的码字。



# 三、设计流程图

开始

全英Word文档

计算字符个数及其

出现概率

根据信源熵公式

计算信源熵

绘制字符概率的柱状图

将字符按照概率从大到小

进行排列

使用哈夫曼对概率进行编码，使得码字和字符对应起来

统计码字长度并计算平均码长和信源熵

再处理或保存数据

向Excel输出对应数据

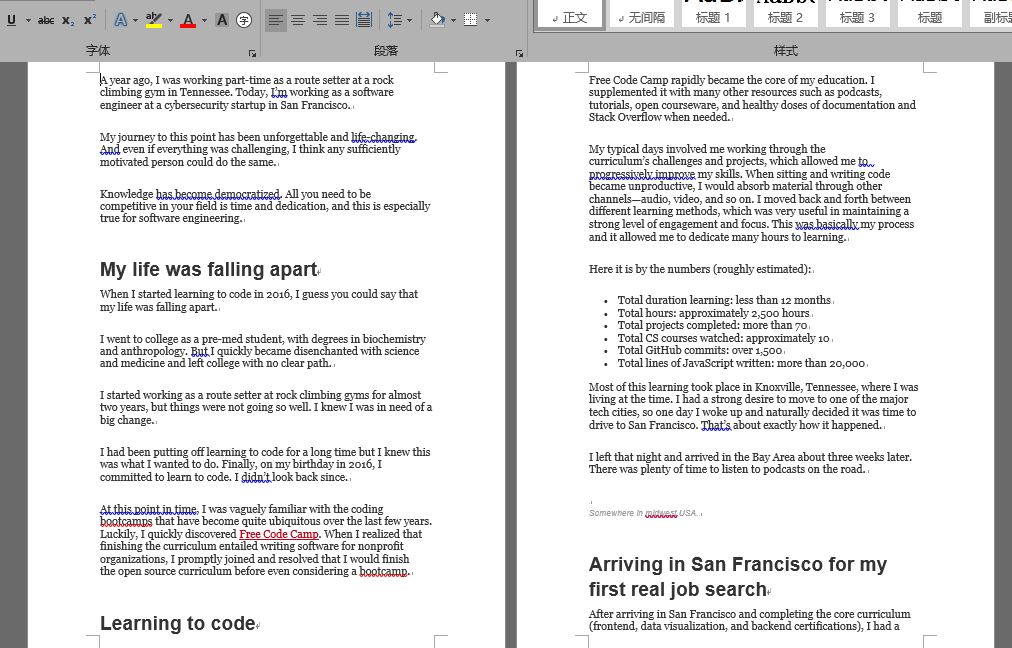
并保存

结束

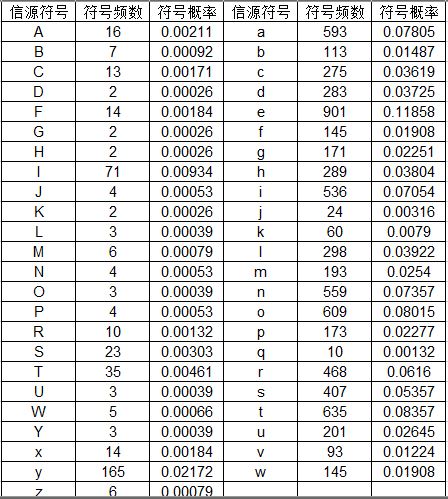
# 四、程序运行及结果

本程序运行后，会产生一个EXCEL的表格，里面会有符号、个数、符号概率、码字、码长、平均码长、信源熵、编码效率等实测数据，同时会自动绘制符号概率的柱状图和饼状图。使得对处理后的数据有更好的了解和把控。

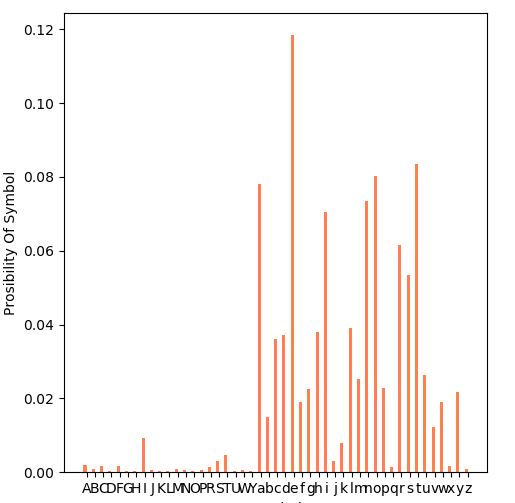
首先是待读取的WORD文档，由于文档内容过多，故下面为部分图。



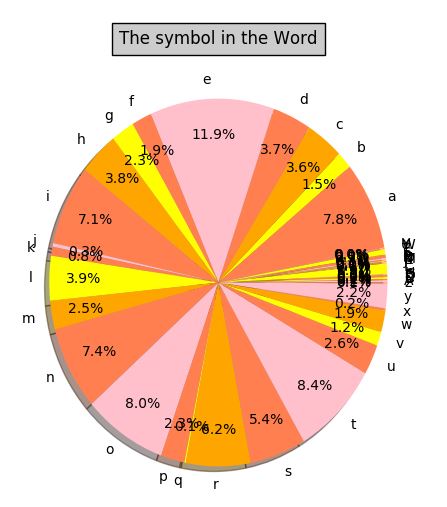
对内容进行读取后并计算得到以下数据：



然后根据表中数据绘制柱状图和饼状图。但是由于符号类型过多，故饼状图观察时出现字符重叠的现象，使得观察方式不太明显。柱状图相比饼状图清晰的多。



符号概率柱状图



符号概率的饼状图

# 五、心得体会

这次的课程设计是自己比较感兴趣的，由于之前自己对数据处理方面很感兴趣，从去年3月份就开始自学Python语言，在大数据的今天，我相信这门语言一定会有自己的一席之地，然后学习完了信息论之后，发现数学与数据计算变得越发重要，一些数据处理的思想是不可获取的。当老师布置完了这个任务之后，我第一想到的就是自己一直在学的Python语言，结果也是和我预料的一样，在数据处理方面Python还是有自己的很大优势。由于它是一门面向对象的编程语言，所以在处理哈夫曼编码时，可以很轻松的描绘出他的树状图结构以及其中的逻辑关系。而且在数据计算和导入、导出也十分的方便。

通过这次的课程设计，自己更加深入的了解了哈弗曼树的结构以及二叉树的相关算法。同时也对自己所需的课本内容进行了回顾，重新复习了信源熵、码字、信源编码和编码效率等相关概念。并把课本上的理论知识和实际运用相结合到一起。使得自己对理论知识有了更深的了解。同时也让自己更加熟练的运用Python去解决实际编程问题。让自己学到的东西转化为实际的结果，既对自己有了更加的推动力，有让自己有了很大的信心。

总之，非常感谢成老师在这半年信息论课程对我的指导，还有这次课程设计的机会，这次的课程设计让我受益匪浅。在以后的学习过程中，更要扎扎实实，脚踏实地的去学习每一样东西，不管他们是简单还是复杂。

# 参考文献

[1]曹雪虹，张宗橙.《信息论与编码(第二版)》[M]. 清华大学出版社，2006.

[2]傅祖芸.《信息论基础》.北京：电子工业出版社，1989

[3]王育民，梁传甲.《信息与编码理论》.西安：西安电子科技大学，1986

# 附录：源程序

**文件一：Main.py**

**from Huffman import \***

**from encode import \***

**import xlwt**

**if \_\_name\_\_ == '\_\_main\_\_':**

**ii = 0**

**HX = 0 # 信源熵**

**code\_AveLength = 0 # 平均码长**

**probability = {}**

**chars\_freqs = {}**

**title = ['信源符号', '符号频数', '符号概率', '码长', '码字', '平均码长', '符号熵', '编码效率']**

**book = xlwt.Workbook(encoding='utf-8', style\_compression=0)**

**sheet = book.add\_sheet('infor', cell\_overwrite\_ok=True) # 创建excel工作空间**

**for index in range(len(title)):**

**sheet.write(0, index, title[index]) # 写入excel标题**

**content = ReadDocx('test.docx') # 读入docx的内容**

**chars\_freqs, probability = CalcProbability(content) # 统计字符个数和概率**

**PaintPie(probability)**

**print(chars\_freqs) # 这样数据就可以按照字母顺序输出**

**# Huffman Encoding Process**

**nodes = CreateNodes([chars\_freqs[i] for i in chars\_freqs.keys()])**

**root = CreateHuffmanTree(nodes)**

**codes = HuffmanEncoding(nodes, root)**

**f = lambda y: abs(y\*(math.log2(y))) # 构造函数-Pi \* log2(Pi)**

**for tt in probability.keys():**

**HX += f(probability[tt]) # 计算信源熵**

**code\_AveLength += probability[tt]\*len(codes[ii]) # 计算平均码长**

**ii += 1**

**efficiency = HX/code\_AveLength # 计算Huffman编码效率**

**print(probability)**

**print('Huffman Encoding Efficiency is %.2f' % efficiency)**

**ii = 0**

**for item in chars\_freqs.keys():**

**print('Character:%s freq:%-2d encoding: %s' % (item, chars\_freqs[item], codes[ii]))**

**ii += 1 # 输出Huffman编码后码字**

**ii = 1 # 数据从第一行，第零列开始**

**index = 0**

**for tt in probability.keys():**

**sheet.write(ii, index, tt)**

**sheet.write(ii, index+1, chars\_freqs[tt])**

**sheet.write(ii, index+2, probability[tt])**

**sheet.write(ii, index+3, len(codes[ii-1]))**

**sheet.write(ii, index+4, codes[ii-1])**

**ii += 1 # 将数据按照字符顺序输出**

**sheet.write(1, 5, code\_AveLength) # 输出平均码长**

**sheet.write(1, 6, HX) # 输出信源熵**

**sheet.write(1, 7, round(efficiency, 2)) # 输出编码效率**

**book.save(r'D:\User\lenovo\Desktop\information\encode\test.xls')# 保存文件**

**文件二：Huffman.py**

**import math**

**class Node:**

**def \_\_init\_\_(self, freq):**

**self.left = None**

**self.right = None**

**self.father = None**

**self.freq = freq**

**def Is\_Left(self):**

**return self.father.left == self**

**def CreateNodes(freqs):**

**return [Node(freq) for freq in freqs]**

**def CreateHuffmanTree(nodes):**

**queue = nodes[:]**

**while len(queue) > 1:**

**queue.sort(key=lambda nodes: nodes.freq) # 按概率大小排序**

**node\_left = queue.pop(0)**

**node\_right = queue.pop(0) # 删除要结合的两结点**

**node\_father = Node(node\_left.freq + node\_right.freq)**

**node\_father.right = node\_right**

**node\_father.left = node\_left**

**node\_right.father = node\_father**

**node\_left.father = node\_father # 建立每个结点的逻辑关系**

**queue.append(node\_father) # 添加结合后的新结点**

**queue[0].father = None**

**return queue[0]**

**def HuffmanEncoding(nodes, root):**

**codes = [''] \* len(nodes)**

**for i in range(len(nodes)):**

**temp\_node = nodes[i]**

**while temp\_node != root: # 从叶结点遍历到根结点**

**if temp\_node.Is\_Left(): # 判断是否为左儿子**

**codes[i] = '0' + codes[i]**

**else:**

**codes[i] = '1' + codes[i]**

**temp\_node = temp\_node.father**

**return codes**

**文件三：Comput.py**

**import docx**

**import matplotlib.pyplot as plt**

**import numpy as np**

**def ReadDocx(docName):**

**FullText = []**

**doc = docx.Document(docName)**

**paras = doc.paragraphs # Word文档的每个段落**

**for p in paras:**

**FullText.append(p.text)**

**return ''.join(FullText)**

**def CalcProbability(content):**

**Result = {}**

**probability = {}**

**chars\_freqs = {} # 排序后字符字典**

**posba = {} # 排序后概率字典**

**for i in content:**

**if i.isalpha():**

**if i in Result.keys():**

**Result[i] += 1**

**else:**

**Result[i] = 1**

**total = sum(Result.values())**

**for j in Result.keys():**

**probability[j] = round(Result[j] / total, 5)**

**temp\_list1 = sorted(Result.items(), key=lambda d: d[0])**

**for ss in temp\_list1:**

**chars\_freqs[ss[0]] = ss[1]**

**temp\_list2 = sorted(probability.items(), key=lambda d: d[0])**

**for ss in temp\_list2:**

**posba[ss[0]] = ss[1] # 按照字母大小进行排序**

**return chars\_freqs, posba # 输出频数和概率**