

数据结构与算法分析 项目个人总结

项目名称: A SIMPLE HUFFMAN CODER

小组编号: 第 23 组

姓名: 施宇昂

学号: 2017141463210

提交报告时间: 2018 年 11 月 15 日

1. 个人信息:

姓名：施宇昂

学号：2017141463210

学院：软件学院

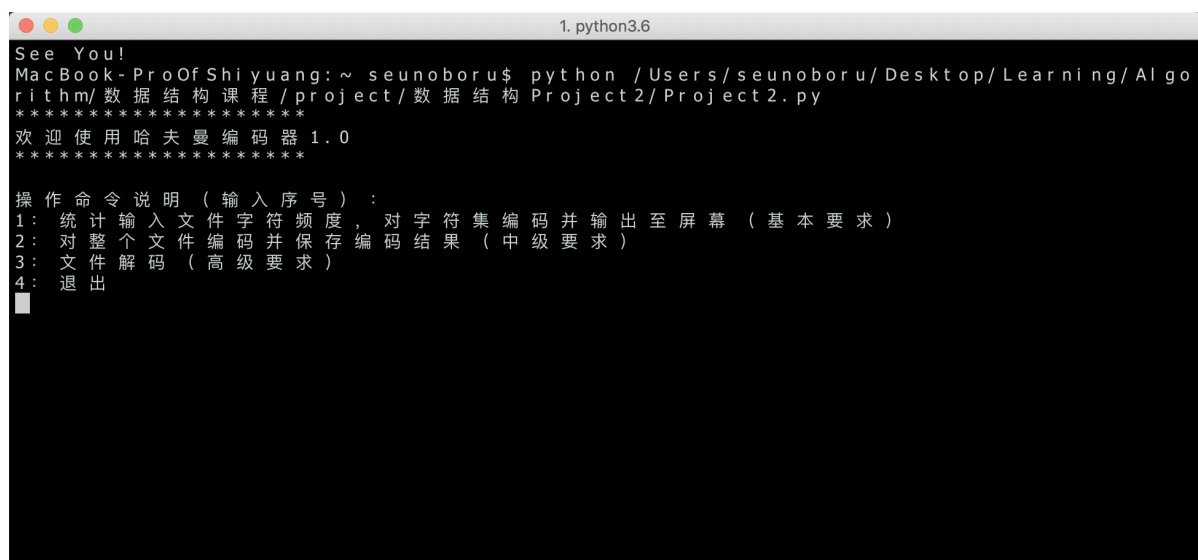
年级：2017 级

2. 个人分工

施宇昂完成了：

1. 控制台图形界面的设计与编写，具体样式如下：

图一是主界面，



```
1. python3.6
See You!
MacBook-ProOfShiyuang: ~ seunoboru$ python /Users/seunoboru/Desktop/Learning/Algorithm/数据结构课程/project/数据结构Project2/Project2.py
*****
欢迎使用哈夫曼编码器 1.0
*****

操作命令说明（输入序号）：
1：统计输入文件字符频度，对字符集编码并输出至屏幕（基本要求）
2：对整个文件编码并保存编码结果（中级要求）
3：文件解码（高级要求）
4：退出
█
```

图二是文件字频和字符集编码输出，



```
1. python3.6
L 3
F 3
R 3
8 3
J 2
I 2
4 2
+ 2
V 2
B 1
@ 1
/ 1
N 1
6 1
Z 1
= 1
W 1
J 1
G 1
5 1
*****
编码集为：
*****
r 0000
d 00010
p 000110
j 0001110000
i 0001110001
/ 00011100100
N 00011100101
B 00011100110
@ 00011100111
```

图三是对文件编码并保存编码结果,

```
1. python3.6
*****
欢迎使用哈夫曼编码器 1.0
*****

操作命令说明 (输入序号) :
1: 统计输入文件字符频度, 对字符集编码并输出至屏幕 (基本要求)
2: 对整个文件编码并保存编码结果 (中级要求)
3: 文件解码 (高级要求)
4: 退出
2
文件编码已完成
压缩率为: 37.05280946345585 %
是否继续? [Y/y][N/n] █
```

图四是文件解码,

```
1. python3.6
The dataset is organised as follows:
Calibration Data
S0: Training Data
contains sets background, city center, regular flow
S1: Person Count and Density Estimation
contains sets L1, L2, L3
S2: People Tracking
S3: Flow Analysis and Event Recognition

contains sets Event Recognition and Multiple Flow
Each subset contains several sequences and each sequence contains different views (4 up to 8).

Calibration Data

The calibration data (one file for each of the 8 cameras) can be found here. The ground plane is assumed to be the Z=0 plane. C++ code (available here) is provided to allow you to load and use the calibration parameters in your program (courtesy of project ETISEO). The provided calibration parameters were obtained using the freely available Tsai Camera Calibration Software by Reg Wilson. All spatial measurements are in metres.
The cameras used to film the datasets are:

Frames are compressed as JPEG image sequences. All sequences (except one) contain Views 001-004. A few sequences also contain Views 005-008. Please see below for more information.

Orientation
The cameras are installed at the locations shown below to cover an approximate area of 100m x 30m (the scale of the map is 20m):
解码已完成
是否继续? [Y/y][N/n] █
```

2. 从磁盘读入一个仅包括英文字母及标点符号的文本文件(如 f1.txt), 统计各字符的频度, 据此构建 Huffman 树, 对各字符进行编码, 并将字符集, 频度集及相应编码码字集输出在显示器或保存在另一个文本文件 (如 f1_code.txt) 中;

详细过程如下:

我首先调用 python 的 collections 内部库, 得到词频:

```
#得到词频
def getStatics(string):
    count = collections.Counter(string)
    return count
```

然后并以此构建 Huffman 树：

```
# 创建哈夫曼树
def creatHuffmanTree(nodeQ):
    while nodeQ.size != 1:
        node1 = nodeQ.popNode()
        node2 = nodeQ.popNode()
        r = TreeNode([None, node1.priority+node2.priority])
        r.leftChild = node1
        r.rightChild = node2
        nodeQ.addNode(r)
    return nodeQ.popNode()
```

然后由 Huffman 树得到 Huffman 编码：

```
# 由哈夫曼树得到哈夫曼编码表
def HuffmanCodeDic(head, x):
    global codeDic, codeList
    if head:
        HuffmanCodeDic(head.leftChild, x+'0')
        head.code += x
        if head.val:
            codeDic2[head.code] = head.val
            codeDic1[head.val] = head.code
        HuffmanCodeDic(head.rightChild, x+'1')
```

3. 对整个文件进行编码并将结果保存在一个二进制文件（如 f1_result.huf）中，同时计算压缩率；

压缩率的计算公式是：

压缩率 = 压缩文件大小 / 原文件大小

4. 根据 f1_code.txt 和 f1_result.huf 的内容，解码出原始的文本文件并另存到磁盘（如 f1_reconstruct.txt）。

我的思路是，首先读存有 Huffman 编码文件中的每一位（0 或者 1），每读一位就在编码集中进行查找，如果存在编码，就翻译为字符，如果不存在，就继续往下读一位。

具体代码如下：

```
# 字符串解码
def TransDecode(StringCode):
    global codeDic2
    code = ""
    ans = ""
    for ch in StringCode:
        code += ch
        if code in codeDic2:
            ans += codeDic2[code]
            code = ""
    return ans
```

```
#解码
def decode(filename):
    fr = open(filename)
    transdecode = fr.read()
    aa = TransDecode(transdecode)
    f = open('/Users/seunoboru/Desktop/Lea
    f.write(aa)
    f.close()
    fr.close()
    print(aa)
```

3. 个人工作中碰到的问题及解决思路

问题一：

如何把数字 0-1 用二进制的形式写入文件。

解决方案：

我用 Python 语言编写此项目。然而，Python 的高度封装，不像 C 和 C++ 能够接触底层，所以 Python 没有简单的、直接的方法，用二进制输出。

我查了相关资料，最后找到了一个将数字转换成二进制的库：“six”（不知道为什么要叫 six）。我只需调用函数 `six.int2byte()`，就能将数字以 byte 形式存入。

问题二：

有一个问题非常致命：现在所有的系统都是以 byte 为最小单位的，我们没办法以 bit 的形式把 0-1 一个一个地写进文件，我们只能一次将 8 个 bit 存入。问题就是，大部分字符的 Huffman 编码都不是 8 的倍数，字符的 Huffman 编码加起来也不是 8 的倍数，如果这样存入，系统会自动在缺位的地方补零。如此，在恢复文件的时候，最后补零的地方实际上会被误恢复成 Huffman 编码为“0”的字符。

解决方案：

我在写文件的时候，在文件顶端添加了一个信息：补零的个数。我只需记录补了几个零，然后再恢复文件的时候，先把后面补的零全删掉就可以了。举个例子，我现在的文本转换成 Huffman 编码，一共有 54 bit，那么我需要再补 2 个 bit 的 0。于是我在写文件的时候，就在文首先记录“000010”1 个 byte，然后再记录 Huffman 编码。

这样，我在解压的时候，先读第一个 byte，得到补了 2 个 0 的情况，假设我将补 0 个数存入 `zero_num` 变量，将文件所有内容读入 `data` 变量，我只要 `data = data[0:-zero_num]`，就能把补的 0 全部去除。

问题三：

计算字频，可以手写 for 循环，遍历整个文本得到，但是我想，万能的 Python 被江湖人称“只有你想不到，没有 Python 做不到”。统计字频作为自然语言处理（NLP）的基本步骤，Python 一定是有库支持的。一方面，用一行代码解决问题，代码简洁；另一方面，他人实现的库的效率一定比我们用循环的效率要高得多。

于是我就思考：能不能利用 Python 库实现字频？

解决方案：

我首先查了 NLTK 的库。NLTK 作为自然语言处理的万能库，肯定是有字频统计的。一

查，果然有。只需要调用 `freq = nltk.FreqDist(text)`，就能得到其中每一个字符的字频。其中 `freq` 是一个字典。

我又查了 Python 的内生库，发现了一个名为“collection”的库，也能统计字频，同样的，调用 `freq = collection.Counter(text)`，就能得到。其中 `freq` 同样是一个字频。

4. 个人总结

以前一直拿 Python 做科学计算和深度学习，没有做过一个稍微复杂一点的项目。这次选择 Python 作为项目语言，一开始是因为，我知道有库能够非常方便地实现字频统计。

其实也肯定有库能够实现 Huffman 编码，但是我没有去查，因为本项目的精髓就在于 Huffman 编码，实验的目的就是让我们熟悉 Huffman 树和 Huffman 编码，这块我都不自己写，那不是这个实验设计者的初衷。

在项目中遇到过很多问题，比如说 API 的各种调用方法、python 语言本身存在的特殊语法等等。因为这些问题不够具有代表性，所以我没有在第三部分展现。其实，在各种代码实现中，我发现了一个定律：在写代码的过程中，遇到的任何一个问题，只要花时间，一定能够解决。

我想起来在某处看到的某个老师说的话：“学习计算机专业的学生，需要培养一种价值观，即计算机的一切都是可以理解的。”

我想，计算机与其他自然科学相比，有一点天然优势：即使某些天赋异禀的科学家利用他们无双的天才创见堆砌了计算机科学的大楼，但他们都是人类，逃不开人类的思维方式和三观，他们创造的计算机理论或许晦涩，但都不可能逃离人类大脑的理解边界。在计算机中任何原理都是有理由的，我们是在创造理由，而不是像物理、化学、生物，在寻找理由。

即使有些问题你现阶段没有明白，但这些问题都必定是可以理解的，只是时间不够或缺少某些先修知识，所以先放一下，等有时间再学习它们。这种心态可以防止你滑入无助的陷阱。有时候你的潜意识会愚弄你，把很难做成的事情当成无法做成的事情。就像《肖申克的救赎》中表现的那样，Red 第一次看到 Andy 的锤子，他认定 Andy 肯定逃不出来。而实际上 Andy 用 19 年的时间把墙给凿开了，虽然其间各种艰辛，但是结局却很美好。

在这个项目中，我学到了：

1. 如何将数字转换成二进制，存入文件：

调用 six 库，使用 `six.int2byte()`

2. 如何解决系统自动补 0 的情况：

在压缩文本顶部记录补 0 信息，在解压的时候去除。

3. 耐心 debug ；

5. 使用 python 搭建 Huffman 树，实现 Huffman 编码。