算法设计与分析第三次上机：哈夫曼编码压缩

学号：21009200158　姓名：游霄童

时间：2023/11/07

代码语言为Python

问题描述

编写用哈弗曼编码实现文件压缩，输出压缩率，并测试，编程语言不限。

霍夫曼编码使用变长编码表对源符号（如文件中的一个字母）进行编码，通过采用不等长的编码方式，将出现频率高的符号用相对短的比特串表示、出现频率低的符合以相对长的比特串表示，能够缩短表示完整源数据所需要的总比特长度，从而达到无损压缩数据的效果。

Python 3 代码如下：

```python
# -*- coding: utf-8 -*-
import sys
import os

sys.setrecursionlimit(1000000)  # 压缩大文件实时会出现超出递归深度，故修改限制


# 定义哈夫曼树的节点类
class node(object):

    def __init__(self, value=None, left=None, right=None, father=None):
        self.value = value
        self.left = left
        self.right = right
        self.father = father

    def build_father(left, right):
        n = node(value=left.value + right.value, left=left, right=right)
        left.father = right.father = n
        return n

    def encode(n):
        if n.father == None:
            return b''
        if n.father.left == n:
            return node.encode(n.father) + b'0'  # 左节点编号'0'
        else:
            return node.encode(n.father) + b'1'  # 右节点编号'1'
```

```
# 哈夫曼树构建
def build_tree(l):
    if len(l) == 1:
        return l
    sorts = sorted(l, key=lambda x: x.value, reverse=False)
    n = node.build_father(sorts[0], sorts[1])
    sorts.pop(0)
    sorts.pop(0)
    sorts.append(n)
    return build_tree(sorts)


def encode(echo):
    for x in node_dict.keys():
        ec_dict[x] = node.encode(node_dict[x])
        if echo == True:  # 输出编码表（用于调试）
            print(x)
            print(ec_dict[x])


def encodefile(inputfile):
    print("Starting encode...")
    f = open(inputfile, "rb")
    bytes_width = 1  # 每次读取的字节宽度
    i = 0

    f.seek(0, 2)
    count = f.tell() / bytes_width
    print(count)
    nodes = []  # 结点列表，用于构建哈夫曼树
    buff = [b''] * int(count)
    f.seek(0)

    # 计算字符频率,并将单个字符构建成单一节点
    while i < count:
        buff[i] = f.read(bytes_width)
        if count_dict.get(buff[i], -1) == -1:
            count_dict[buff[i]] = 0
        count_dict[buff[i]] = count_dict[buff[i]] + 1
        i = i + 1
    #print("Read OK")
    #print(count_dict)  # 输出权值字典,可注释掉
    for x in count_dict.keys():
        node_dict[x] = node(count_dict[x])
        nodes.append(node_dict[x])

    f.close()
    tree = build_tree(nodes)  # 哈夫曼树构建
    encode(False)  # 构建编码表
    #print("Encode OK")

    head = sorted(count_dict.items(), key=lambda x: x[1], reverse=True)  # 对所有根节点进行排序
    bit_width = 1
    #print("head:", head[0][1])  # 动态调整编码表的字节长度，优化文件头大小
    if head[0][1] > 255:
        bit_width = 2
        if head[0][1] > 65535:
            bit_width = 3
            if head[0][1] > 16777215:
                bit_width = 4
    #print("bit_width:", bit_width)
    i = 0
    raw = 0b1
    last = 0
    name = inputfile.split('.')
    o = open(name[0] + ".ys", 'wb')
    #print(o)
    name = inputfile.split('/')
    o.write((name[len(name) - 1] + '\n').encode(encoding="utf-8"))  # 写出原文件名
    o.write(int.to_bytes(len(ec_dict), 2, byteorder='big'))  # 写出结点数量
    o.write(int.to_bytes(bit_width, 1, byteorder='big'))  # 写出编码表字节宽度
    for x in ec_dict.keys():  # 编码文件头
        o.write(x)
        o.write(int.to_bytes(count_dict[x], bit_width, byteorder='big'))

    #print('head OK')
    while i < count:  # 开始压缩数据
        for x in ec_dict[buff[i]]:
            raw = raw << 1
            if x == 49:
                raw = raw | 1
            if raw.bit_length() == 9:
                raw = raw & (~(1 << 8))
                o.write(int.to_bytes(raw, 1, byteorder='big'))
                o.flush()
                raw = 0b1
                tem = int(i / len(buff) * 100)
                if tem > last:
                    #print("encode:", tem, '%')  # 输出压缩进度
                    last = tem
        i = i + 1

    if raw.bit_length() > 1:  # 处理文件尾部不足一个字节的数据
```

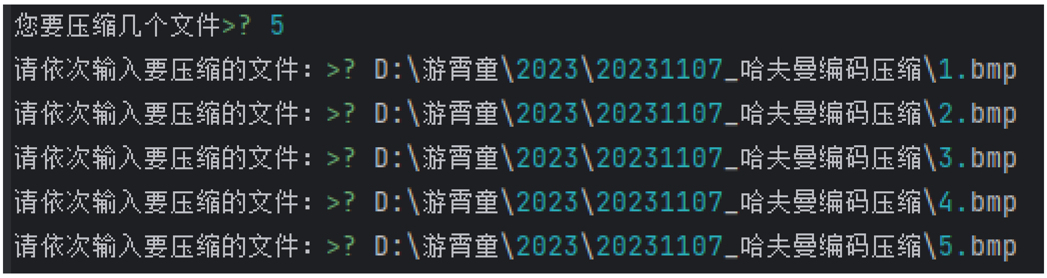
```
        raw = raw << (8 - (raw.bit_length() - 1))
        raw = raw & (~(1 << raw.bit_length() - 1))
        o.write(int.to_bytes(raw, 1, byteorder='big'))
    o.close()
    print(f'{inputfile} File encode successful.')
    file_size1 = os.path.getsize(inputfile)
    file_size2 = os.path.getsize(o.name)
    print(f'{inputfile}压缩率为：{file_size2/file_size1*100:.2f}%')
if __name__ == '__main__':

    # 数据初始化
    node_dict = {}  # 建立原始数据与编码节点的映射，便于稍后输出数据的编码
    count_dict = {}
    ec_dict = {}
    nodes = []
    inverse_dict = {}
    n=int(input('您要压缩几个文件'))
    filename=[]
    for i in range(n):
        filename.append(input("请依次输入要压缩的文件："))
    for i in range(n):
        encodefile(filename[i])
```

测试及结果

输入

- 5
- D:\游霄童\2023\20231107_哈夫曼编码压缩\1.bmp
- D:\游霄童\2023\20231107_哈夫曼编码压缩\2.bmp
- D:\游霄童\2023\20231107_哈夫曼编码压缩\3.bmp
- D:\游霄童\2023\20231107_哈夫曼编码压缩\4.bmp
- D:\游霄童\2023\20231107_哈夫曼编码压缩\5.bmp

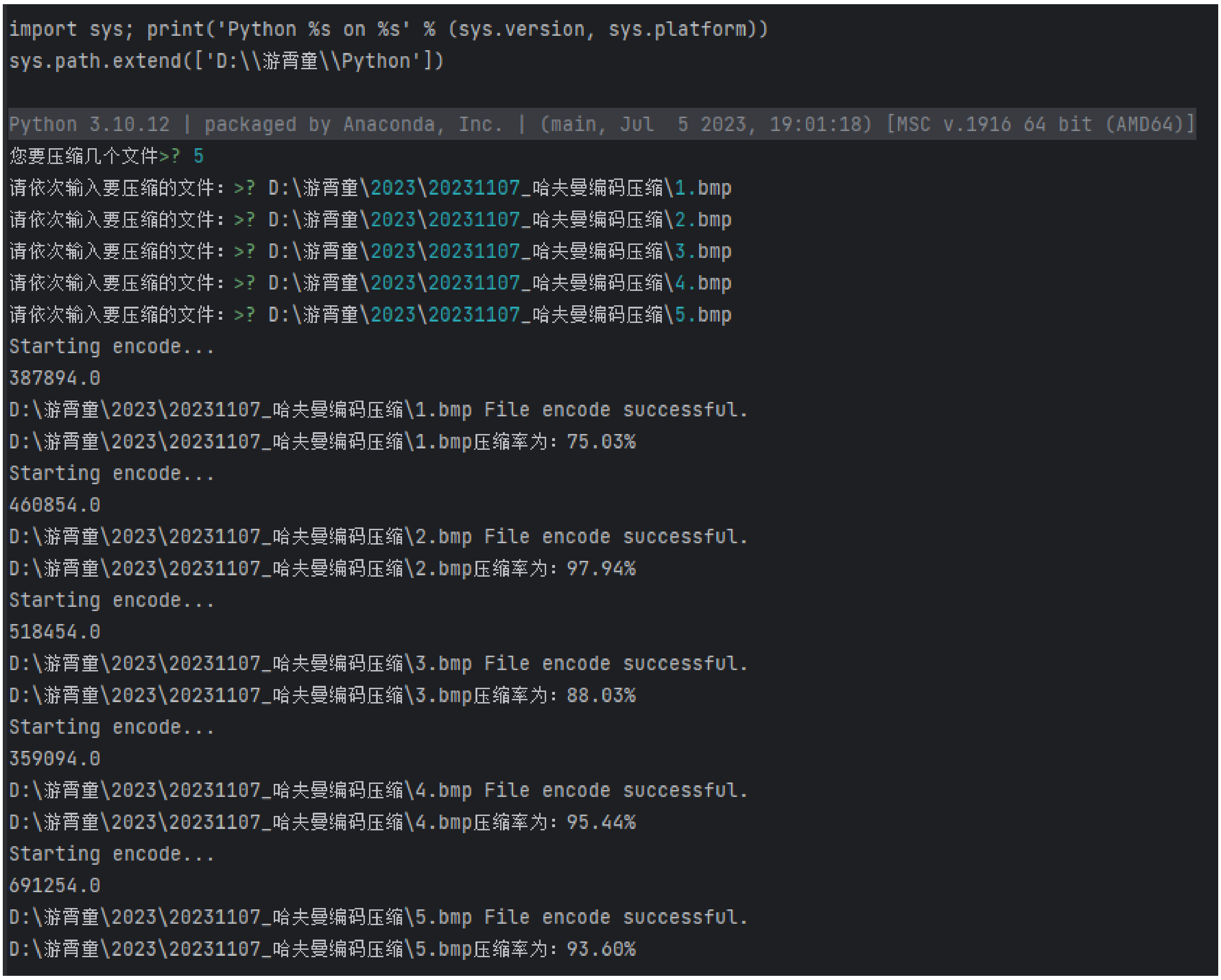


```
您要压缩几个文件>? 5
请依次输入要压缩的文件：>? D:\游霄童\2023\20231107_哈夫曼编码压缩\1.bmp
请依次输入要压缩的文件：>? D:\游霄童\2023\20231107_哈夫曼编码压缩\2.bmp
请依次输入要压缩的文件：>? D:\游霄童\2023\20231107_哈夫曼编码压缩\3.bmp
请依次输入要压缩的文件：>? D:\游霄童\2023\20231107_哈夫曼编码压缩\4.bmp
请依次输入要压缩的文件：>? D:\游霄童\2023\20231107_哈夫曼编码压缩\5.bmp
```

结果如下：

设定压缩率为

$$\text{Compression Ratio} = \frac{\text{Compressed File Size}}{\text{Original File Size}}$$



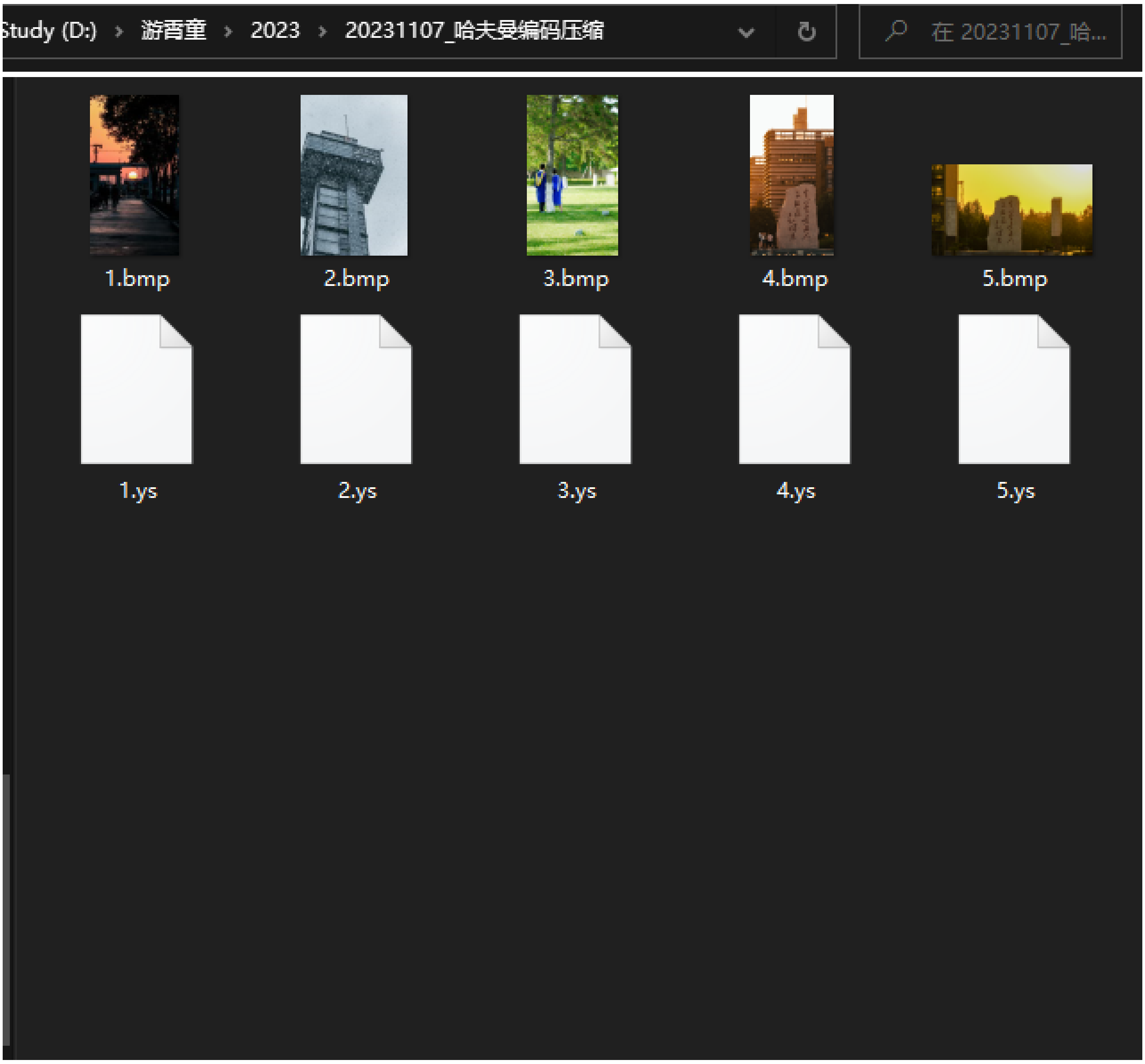
```
import sys; print('Python %s on %s' % (sys.version, sys.platform))
sys.path.extend(['D:\\游霄童\\Python'])

Python 3.10.12 | packaged by Anaconda, Inc. | (main, Jul  5 2023, 19:01:18) [MSC v.1916 64 bit (AMD64)]
您要压缩几个文件>? 5
请依次输入要压缩的文件：>? D:\游霄童\2023\20231107_哈夫曼编码压缩\1.bmp
请依次输入要压缩的文件：>? D:\游霄童\2023\20231107_哈夫曼编码压缩\2.bmp
请依次输入要压缩的文件：>? D:\游霄童\2023\20231107_哈夫曼编码压缩\3.bmp
请依次输入要压缩的文件：>? D:\游霄童\2023\20231107_哈夫曼编码压缩\4.bmp
请依次输入要压缩的文件：>? D:\游霄童\2023\20231107_哈夫曼编码压缩\5.bmp
Starting encode...
387894.0
D:\游霄童\2023\20231107_哈夫曼编码压缩\1.bmp File encode successful.
D:\游霄童\2023\20231107_哈夫曼编码压缩\1.bmp压缩率为：75.03%
Starting encode...
460854.0
D:\游霄童\2023\20231107_哈夫曼编码压缩\2.bmp File encode successful.
D:\游霄童\2023\20231107_哈夫曼编码压缩\2.bmp压缩率为：97.94%
Starting encode...
518454.0
D:\游霄童\2023\20231107_哈夫曼编码压缩\3.bmp File encode successful.
D:\游霄童\2023\20231107_哈夫曼编码压缩\3.bmp压缩率为：88.03%
Starting encode...
359094.0
D:\游霄童\2023\20231107_哈夫曼编码压缩\4.bmp File encode successful.
D:\游霄童\2023\20231107_哈夫曼编码压缩\4.bmp压缩率为：95.44%
Starting encode...
691254.0
D:\游霄童\2023\20231107_哈夫曼编码压缩\5.bmp File encode successful.
D:\游霄童\2023\20231107_哈夫曼编码压缩\5.bmp压缩率为：93.60%
```

那么压缩率以此为

$$75.03\%, 97.94\%, 88.03\%, 95.44\%, 93.60\%$$

结果如下



Study (D:) › 游香童 › 2023 › 20231107_哈夫曼编码压缩
在 20231107_哈...
1.bmp
2.bmp
3.bmp
4.bmp
5.bmp
1.ys
2.ys
3.ys
4.ys
5.ys