哈夫曼树压缩-Python

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

import sys
sys.setrecursionlimit(1000000)     #python默认的递归深度有限，约900多，压缩文件时会超过，故引用sys修改最大递归深度

# 代码中出现的列表及字典的详细解释
# bytes_list: 存放读入的字节
# count_dict: key 不重复的字节 val 出现的次数
# node_dict: key 不重复的字节 val 对应的结点
# bytes_dict: key 不重复的字节 val 对应的编码
# nodes : 保存结点并构建树，构建完后只剩下根结点

#哈夫曼树结点的类定义
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 #分别定义左右结点，父结点，和结点的权重

#构建哈夫曼树
def creat_tree(nodes_list):
    nodes_list.sort(key=lambda x: x.value)       #将结点列表进行升序排序
    if len(nodes_list)==1:
        return nodes_list[0]         #只有一个结点时，返回根结点

father_node=node(nodes_list[0].value+nodes_list[1].value,nodes_list[0],nodes_list[1]) #创建最小的两个权重结点的父节点
    nodes_list[0].father=nodes_list[1].father=father_node
    nodes_list.pop(0)
    nodes_list.pop(0)
    nodes_list.insert(0,father_node)    #删除最小的两个结点并加入父结点
    return creat_tree(nodes_list)

def node_encode(node1):              #对叶子结点进行编码
```

```
    if node1.father==None:
        return b''
    if node1.father.left==node1:
        return node_encode(node1.father)+b'0'
    else:
        return node_encode(node1.father)+b'1'

def file_encode(input_file):
    print('打开文件并读取中...\n')
    with open(input_file,'rb') as f:
        f.seek(0, 2)        #读取文件的总长度，seek(0,2)移到文件末尾，
tell()指出当前位置，并且用seek(0)重新回到起点
        size=f.tell()
        f.seek(0)
        bytes_list=[0]*size  #创建一个长度为size的列表，存放读入的字节

        i=0
        while i<size:
            bytes_list[i]=f.read(1)  #每次读取一个符号
            i+=1

    print('统计各字节出现频率中...\n')
    count_dict = {}  # 使用一个字典统计一下出现次数
    for x in bytes_list:
        if x not in count_dict.keys():
            count_dict[x] = 0
        count_dict[x] += 1

    node_dict={}   #使用一个字典将count_list中的值node化,其中key为对应的字
符，值为字符对应的结点
    for x in count_dict.keys():
        node_dict[x]=node(count_dict[x])  #结点的权重即count_dict[x]

    print('生成哈夫曼编码中...\n')
    nodes=[]       #使用一个列表来保存结点并由此构建哈夫曼树
    for x in node_dict.keys():
        nodes.append(node_dict[x])

    root=creat_tree(nodes)   #构建哈夫曼树

    #对叶子结点编码，输出字节与哈夫曼码的字典
    bytes_dict={}
    for x in node_dict.keys():
```

```
        bytes_dict[x]=node_encode(node_dict[x])

    # print(bytes_dict)
    #开始压缩文件
    print('开始压缩文件...\n')
    path_list=input_file.split('.')
    name=input_file.split('/')[-1]
    with open(path_list[0]+'.cc','wb') as object:
        #首先将文件的原名写入
        object.write((name+'\n').encode(encoding='UTF-8'))
        #写入结点数量，占位2个字节
        n=len(count_dict.keys())
        object.write(int.to_bytes(n ,2 ,byteorder = 'big'))

        #先计算最大频率所占的字节数
        times=0
        for x in count_dict.keys():
            if times<count_dict[x]:
                times=count_dict[x]
        width=1
        if times>255:
            width=2
            if times>65535:
                width=3
                if times>16777215:
                    width=4
        # 写入width
        object.write(int.to_bytes(width,1,byteorder='big'))


        # 写入结点以及对应频率
        for x in count_dict.keys():
            object.write(x)
            object.write(int.to_bytes(count_dict[x], width,
byteorder='big'))

        #写入数据，注意每次要凑一个字节
        code=b''      #用来存放编译出来的代码
        for x in bytes_list:
            code+=bytes_dict[x]
            out=0
            while len(code)>=8:
                for s in range(8):
```

```
                    out = out << 1
                    if code[s] == 49:    #ASCII码中1为49
                        out = out | 1
                object.write(int.to_bytes(out,1,byteorder='big'))
                out=0
                code=code[8:]
        #处理可能不足一个字节的数据
        object.write(int.to_bytes(len(code), 1, byteorder='big')) #
写入最后一节数据长度
        print(len(code))
        print(code)
        out=0
        for i in range(len(code)):
            out = out << 1
            if code[i] == 49:
                out = out | 1
        object.write(int.to_bytes(out,1,byteorder='big'))
        print('压缩完成! ')



#文件的解压缩
#解压缩前，重温下压缩文件.cc的内容：第一行为原文件名
#第二行的开始两个字节纪录结点数量n，然后一个字节纪录频率的位宽width，后面纪录每个
字节与其频率
#之后全部是数据内容
def file_decode(input_file):
    print('开始解压缩文件')
    with open(input_file,'rb') as f_in:

        f_in.seek(0,2)
        length=f_in.tell() #读出文件的总长度
        f_in.seek(0)

        path_list = input_file.split('.')
        name = f_in.readline().decode(encoding="UTF-8").split('/')
[-1].replace('\n','')
        name = name.split('.')[-1]                    #读出文件名
        with open(path_list[0]+'.'+name,'wb') as f_out:
            n=int.from_bytes(f_in.read(2), byteorder = 'big')     #
读出结点数量
            width=int.from_bytes(f_in.read(1), byteorder = 'big') #
读出位宽
```

```
            count_dict={}
            i=0
            while i<n:
                dict_key=f_in.read(1)

dict_value=int.from_bytes(f_in.read(width),byteorder='big')
                count_dict[dict_key]=dict_value
                i+=1



            print('生成反向字典中...')
            #以下过程与编码时的构建过程相同
            node_dict = {}  # 使用一个字典将count_list中的值node化,其中
key为对应的字符，值为字符对应的结点
            for x in count_dict.keys():
                node_dict[x] = node(count_dict[x])  # 结点的权重即
count_dict[x]



            nodes = []  # 使用一个列表来保存结点并由此构建哈夫曼树
            for x in node_dict.keys():
                nodes.append(node_dict[x])

            root = creat_tree(nodes)  # 构建哈夫曼树

            # 对叶子结点编码，输出字节与哈夫曼码的字典
            bytes_dict = {}
            for x in node_dict.keys():
                bytes_dict[x] = node_encode(node_dict[x])

            # print(bytes_dict)
            # 生成反向字典,key为编码，value为对应的字节
            diff_dict={}
            for x in bytes_dict.keys():
                diff_dict[bytes_dict[x]]=x



            print('解码中...')
            # 解码时不停读取单个数字，遍历二叉树，直到找到叶子结点
            out=b''
            i=f_in.tell()
            node_now = root
            result = b''
```

```
            while i < length-2:
                i+=1
                temp=int.from_bytes(f_in.read(1),byteorder='big')
                for mm in range(8):         #将数据转换成b'01'形式
                    if temp&1 == 1:
                        out=b'1'+out
                    else:
                        out=b'0'+out
                    temp=temp>>1

                while out:        #遍历哈夫曼树
                    if out[0]==49:
                        node_now=node_now.right
                        result = result+b'1'
                    if out[0]==48:
                        node_now=node_now.left
                        result = result+b'0'
                    out=out[1:]
                    if node_now.left==None and
node_now.right==None:
                        f_out.write(diff_dict[result])
                        result=b''
                        node_now=root

            # 处理最后可能不满8位的数据
            last_length = int.from_bytes(f_in.read(1),
byteorder='big')
            print(last_length)
            temp= int.from_bytes(f_in.read(1), byteorder='big')
            print(temp)
            for mm in range(last_length):  # 将数据转换成b'01'形式
                if temp & 1 == 1:
                    out = b'1' + out
                else:
                    out = b'0' + out
                temp = temp >> 1
            print(out)
            while out:  # 遍历哈夫曼树
                if out[0] == 49:
                    node_now = node_now.right
                    result = result + b'1'
                if out[0] == 48:
                    node_now = node_now.left
```

```
                    result = result + b'0'
                out = out[1:]
                if node_now.left == None and node_now.right ==
None:
                    f_out.write(diff_dict[result])
                    print(result)
                    result = b''
                    node_now = root
    print('解压成功! ')

# 本体调用本函数时运行的内容
if __name__ == '__main__':
    de=int(input('请输入您需要进行的操作（1为压缩，2为解压）：'))
    if de==1:
        in_file=input('请输入您需要压缩的文件路径：')
        file_encode(in_file)
    if de==2:
        in_file=input('请输入您需要解压的文件路径：')
        file_decode(in_file)
```

利用哈夫曼编码进行压缩和解压，具体的算法解释之后会更新。
参考资料：

(152条消息) Python中使用哈夫曼算法实现文件的压缩与解压缩*LeafCC*的博客-*CSDN*博客
python哈夫曼树压缩

python-霍夫曼编码实现压缩和解压缩（二）_来自比邻星的博客-CSDN博客