'''
程序说明：
***逻辑：先对bmp文件进行游程编码，再对游程编码后的文件进行哈夫曼编码
***优点：对于色块分布明显的图像，可以进一步提高压缩率
***缺点：对于色块分布不明显的图像，即色彩丰富而复杂的图像，无法有效提高压缩率
***改进： 1.对于哈夫曼树的构建过程，或可构建k叉哈夫曼树；
2.对于哈夫曼树的构建过程，或可采用最小堆；
'''
import sys

修改递归深度限制以便压缩大文件

sys.setrecursionlimit(1000000)
'''

1. 游程编码压缩
   '''
   def rle(inputfile, outputfile):
   f = open(inputfile, 'rb')
   w = open(outputfile, 'wb')
   last = None
   count = 0
   t = f.read(1)
   while t:
   if last is None:
   last = t
   count = 1
   else:
   if t == last and count < 255:
   count += 1
   else:
   w.write(int.to_bytes(count, 1, byteorder='big'))
   w.write(last)
   last = t
   count = 1
   t = f.read(1)
   w.write(int.to_bytes(count, 1, byteorder='big'))
   w.write(last)

f.close()
w.close()
'''

2. 游程编码解压
'''
def derle(inputfile, outputfile):
w = open(outputfile, 'wb')
f = open(inputfile, 'rb')
count = f.read(1)
byte = f.read(1)
while count and byte:
w.write(int.from_bytes(count, byteorder='big')*byte)
count = f.read(1)
byte = f.read(1)
w.close()
f.close()
'''

3. 定义哈夫曼树的节点类HuffNode
'''
class HuffNode(object):

1.初始化

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

2.父节点构建，并关联子节点

def buildfather(left, right):
n = HuffNode(value=left.value+right.value, left=left, right=right)
left.father = n
right.father = n
return n

3.左右节点编码

```
def encodenode(n):
if n.father == None:
return b''
if n.father.left == n:
return HuffNode.encodenode(n.father)+b'0'
else:
return HuffNode.encodenode(n.father)+b'1'
'''
```

4. 哈夫曼树的构建

```
'''
def buildtree(l):
```

1.若节点唯一则返回节点为哈夫曼树

```
if len(l) == 1:
return l
```

2.若节点不唯一则按规则进行二叉哈夫曼树构建,排序-构建-删除-排序

```
sorts = sorted(l, key=lambda x: x.value, reverse=False)
n = HuffNode.buildfather(sorts[0], sorts[1])
sorts.pop(0)
sorts.pop(0)
sorts.append(n)
return buildtree(sorts)
'''
```

5. 带权路径编码并输出编码表

```
'''
def encode(echo):
for x in node_dict.keys():
ec_dict[x] = HuffNode.encodenode(node_dict[x])
if echo == True:
```

```
print(x)
print(ec_dict[x]+'\n')
'''
```

6. 文件压缩

```
'''
def compressfile(inputfile):
print("开始压缩！")
```

0.进行游程编码压缩

```
print("开始游程编码！")
name0 = inputfile.split('.')
fileafterrle = name0[0]+'压缩.rle'
print(fileafterrle)
rle(inputfile,fileafterrle)
print("游程编码结束！")
print("开始哈夫曼编码！")
f = open(fileafterrle, 'rb')
```

1.初始化

```
readwidth = 1
i = 0
f.seek(0, 2)
count = f.tell()/readwidth
nodes = []
buff = [b'']*int(count)
f.seek(0)
```

2.计算频率并将单个字符构造成单一的哈夫曼节点类

```
for i in range(int(count)):
buff[i] = f.read(readwidth)
```

```
if count_dict.get(buff[i], -1) == -1:
count_dict[buff[i]] = 0
count_dict[buff[i]] = count_dict[buff[i]] + 1
print("读取结束！")
for x in count_dict.keys():
node_dict[x] = HuffNode(count_dict[x])
nodes.append(node_dict[x])
f.close()
```

3.构建哈夫曼树并进行编码

```
hufftree = buildtree(nodes)
encode(False)
print("编码完成！")
```

4.写入前准备，根据字符最高出现频率确定写入文件中所占空间，创建文件

```
head = sorted(count_dict.items(), key=lambda x: x[1], reverse=True)
writewidth = 1
if head[0][1] > 255:
writewidth = 2
if head[0][1] > 65535:
writewidth = 3
if head[0][1] > 16777215:
writewidth = 4
i = 0
raw = 0b1
last = 0
name = inputfile.split('.')
o = open(name[0]+'.wr', 'wb')
name = inputfile.split('/')
```

5.信息写入

```
o.write((name[len(name)-1] + '\n').encode(encoding="utf-8"))
o.write(int.to_bytes(len(ec_dict), 2, byteorder='big'))
o.write(int.to_bytes(writewidth, 1, byteorder='big'))
for x in ec_dict.keys():
o.write(x)
o.write(int.to_bytes(count_dict[x], writewidth, byteorder='big'))
```

6.数据压缩写入

```
for i in range(int(count)):
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("压缩进度：", tem, '%')
last = tem
i = i + 1
```

8.处理尾部数据

```
if raw.bit_length() > 1:
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("文件压缩完成！")
'''
7.文件解压
'''
```

```
def decompressfile(fileinput):
print("开始解压！")
```

1.哈夫曼编码解压缩初始化

```
print("开始哈夫曼解码！")
count = 0
count = 0
raw = 0
last = 0
f = open(fileinput,'rb')
f.seek(0,2)
eof = f.tell()
f.seek(0)
```

2.解压信息

```
name = fileinput.split('/')
outputfile = fileinput.replace(name[len(name)-1],f.readline().decode(encoding='utf-8'))
o = open(outputfile.replace('\n',''),'wb')
#print(outputfile.replace('\n',''))
count = int.from_bytes(f.read(2),byteorder = 'big')
readwidth = int.from_bytes(f.read(1),byteorder = 'big')
i = 0
de_dict = {}
for i in range(count):
key = f.read(1)
value = int.from_bytes(f.read(readwidth),byteorder='big')
de_dict[key] = value
for x in de_dict.keys():
node_dict[x] = HuffNode(de_dict[x])
nodes.append(node_dict[x])
```

3.重建哈夫曼树和编码表

```
hufftree =buildtree(nodes)
encode(False)
for x in ec_dict.keys():
inverse_dict[ec_dict[x]] = x
i = f.tell()
data = b''
```

4.解压数据写入文件

```
while i < eof:
raw = int.from_bytes(f.read(1),byteorder='big')
i = i+1
j = 8
while j>0:
if(raw>>(j-1))&1 == 1:
data = data+b'1'
raw = raw&(~(1<<(j-1)))
else:
data = data+b'0'
raw = raw&(~(1<<(j-1)))
if inverse_dict.get(data,0)!=0:
o.write(inverse_dict[data])
o.flush()
data = b''
j = j-1
tem = int(i/eof*100)
if tem>last:
print("解压进度：",tem,'%')
last = tem
raw = 0
f.close()
o.close()
```

5.进行游程编码解压，得到源文件

```
print("哈夫曼解码完成！")
print("开始游程解码！")
rleinput = outputfile.replace('\n','')
fileoutput = rleinput.split('.')
fileoutput = fileoutput[0]+'还原.bmp'
#print(rleinput,fileoutput)
derle(rleinput,fileoutput)
#derle(fileafterhuffman,fileoutput)
print("游程解码完成！")
print("文件解压完成！")
'''
8.主函数
'''
if name == 'main':
node_dict = {}
ec_dict = {}
count_dict = {}
nodes = []
inverse_dict = {}
if input("请输入要执行的操作\n1.压缩文件 2.解压文件\n") == '1':
# 1.批量压缩测试
#i = 1
#while i < 6:
#print("开始第"+str(i)+"张图片压缩！")
#compressfile('C:\Users\Jeremy\Desktop\'+str(i)+'.bmp','C:\Users\Jeremy\Desktop\reco\'+str(i)+'0.
wr')
#i = i+1
# 2.固定文件压缩测试
compressfile('C:\Users\Jeremy\Desktop\2.bmp')
# 3.自定义文件压缩测试
#compressfile(input("请输入要压缩的文件：\n"),input("请输入解压后文件：\n"))
else:
# 1.批量解压测试
#i = 1
#while i < 6:
#print("开始第"+str(i)+"张图片解压！")
#decompressfile('C:\Users\Jeremy\Desktop\'+str(i)+'.wr','C:\Users\Jeremy\Desktop\reco\'+str(i)+'.b
mp')
#i = i+1
```

```
# 2.固定文件解压测试
decompressfile('C:\Users\Jeremy\Desktop\2.wr')
# 3.自定义文件解压测试
#decompressfile(input("请输入要解压的文件：\n"),input("请输入游程编码压缩后文件存放位置：\n"))
```