作业2

第一题

（a）

传统的哈夫曼编码需要事先获得要编码的所有的字符的概率，并自底向上的构造哈夫曼树，为频率高的字符分配短编码。而自适应哈夫曼编码随着输入数据流的到达，动态地收集和更新符号的概率，然后符号会被赋予新的码字，这可以对不断变化的实时数据进行动态编码，弥补了哈夫曼编码的缺点。但由于单个丢失会损坏整个代码，因此它对传输错误更加敏感。

（b）

1. baNEWcc（bacc）

   最终的解码序列

码字	**01**	**01**	**00**	**10**	**101**
信号	b	a	NEW	c	c

   推导过程：

   1. 根据初始的哈夫曼树的解码序列为

码字	**1**	**01**	**10**	**11**	**NEW**
信号	a	b	c	d	00

   2. 接收到01010010101，其中01为b，加入输出队列，更新码字后为：

码字	01	1	10	11	NEW
信号	a	b	c	d	00

3. 剩余010010101，其中01为a，加入输出队列，更新码字后为：

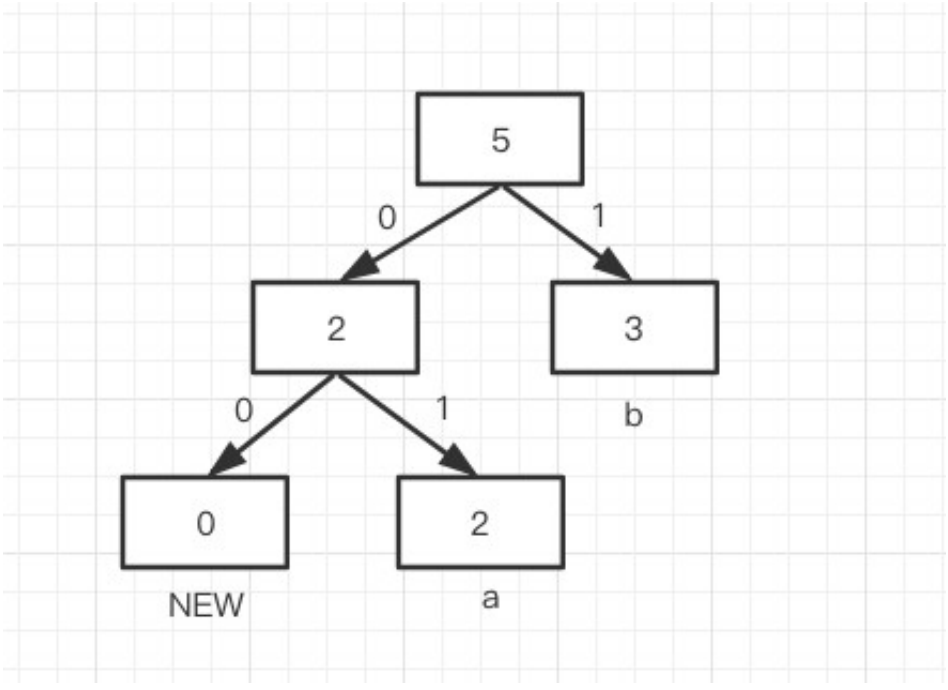
码字	01	1	10	11	NEW
信号	a	b	c	d	00

4. 剩余0010101，其中00为NEW，继续输入10为c，而不是取1为b因为不是新的符号，更新码字后为：

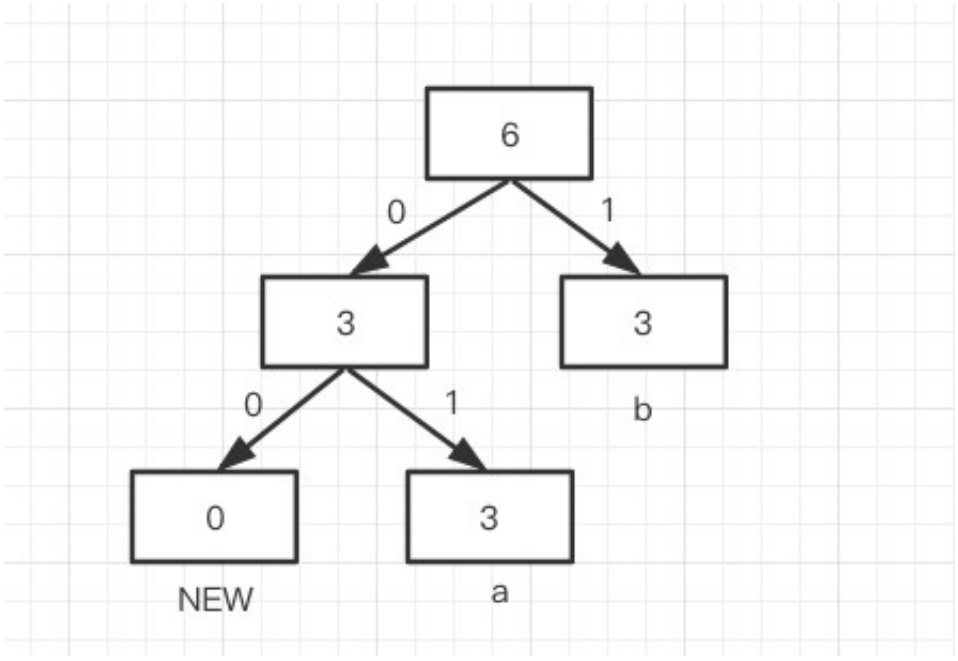
码字	11	0	101	11	NEW
信号	a	b	c	d	100

5. 最终剩余101，只有c满足要求，所以解码得到bacc。

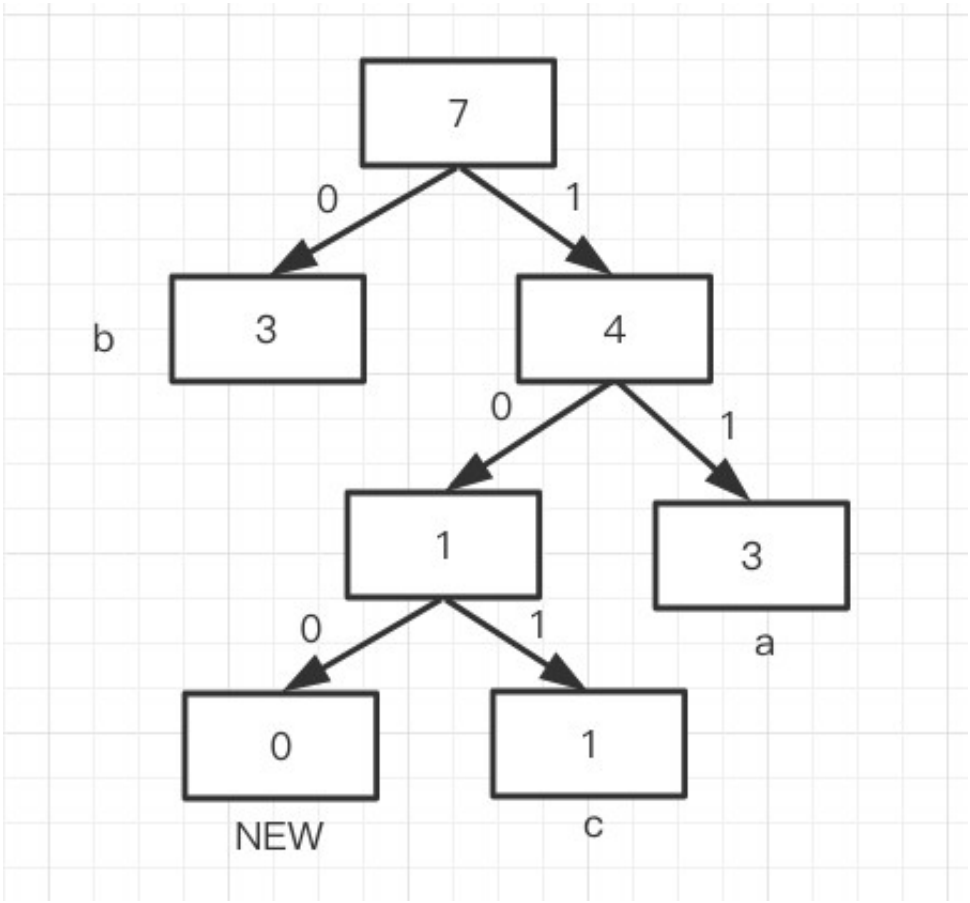
2. bacc的自适应哈夫曼树



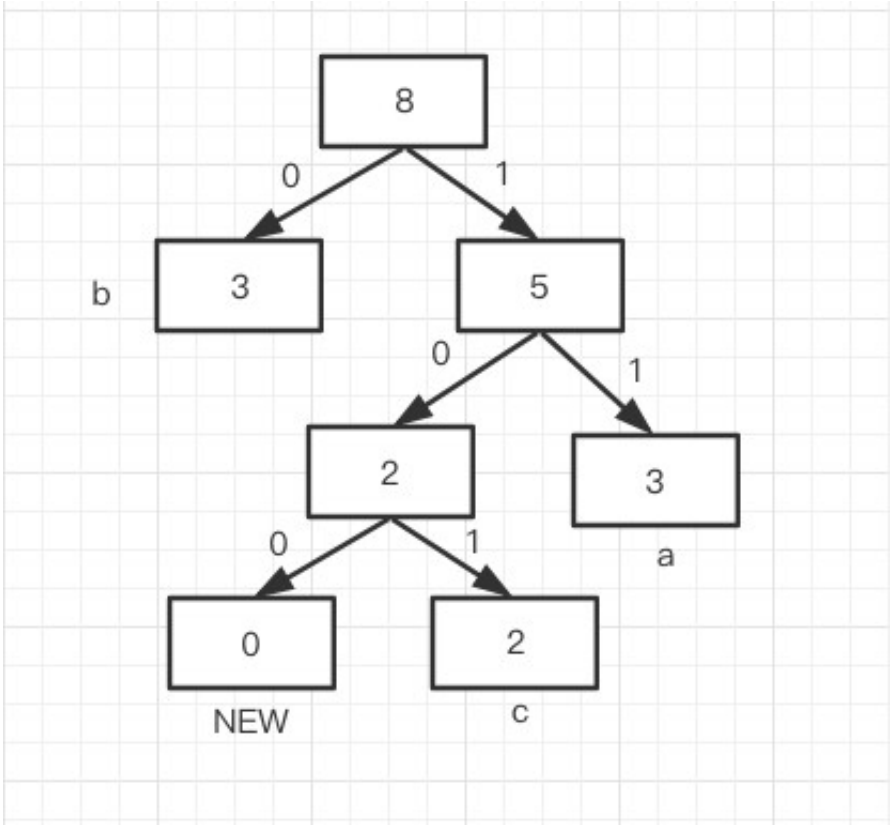
加入b后，不满足从左到右，从上到下的升序兄弟性质，与最远的a节点交换。



加入a后，满足性质不交换节点，更新节点频率值。



将新加入的c和NEW作为原来NEW节点的子节点，然后向上更新节点频率值，发现4小于3，然后交换。



最后加入c，满足性质不交换节点，只更新节点频率值。

第二题

1、理论原因分析

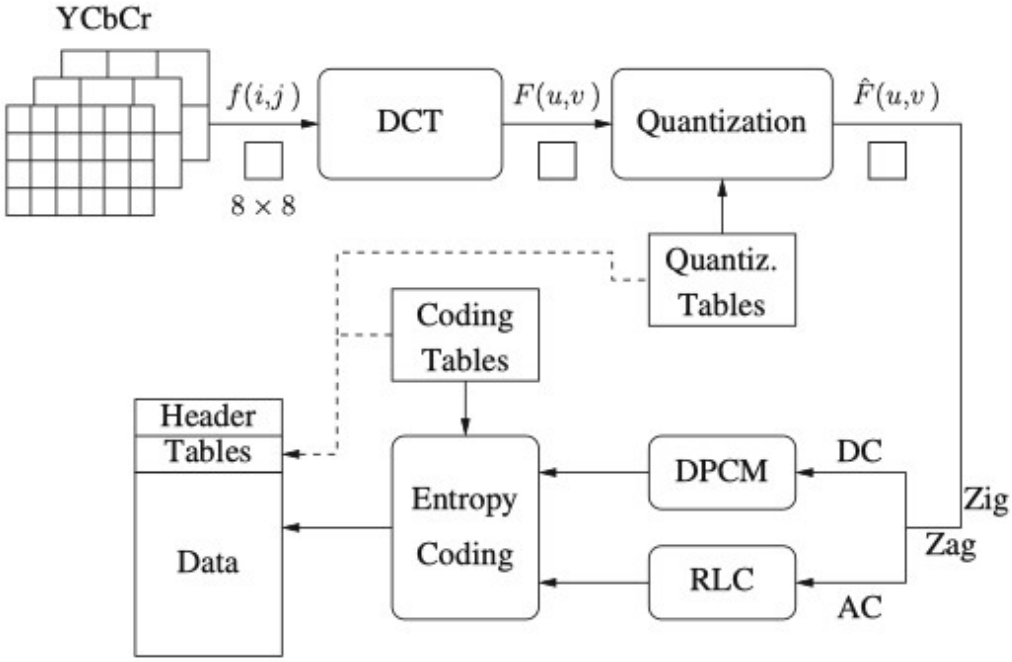
给出你做出选择的原因的解释，包括不同格式图像数据、不同格式图像的编码方法等对比。

1. 第一次作业的中值切分算法适用于GIF压缩格式，使用调色板的8位索引作为像素点的值，但会导致色彩的丰富大大降低，适用于颜色不是很丰富，变化不大的图片，如此次作业中的卡通图片，失真度主要在于色彩数目的减少。
2. 同样是有损压缩的JPEG算法，保存了RGB三个颜色通道的值，只是根据人眼的三个特性对图像内容进行了量化，并根据熵编码减少了信息编码的平均长度，使其更接近信息熵的下界，对自然图像的视觉失真度较小，适用于此次的动物照片，失真度主要在于高频信息的丢失。

JPEG中的DCT变化主要是为了减少高频信息，色度二次采样主要是针对人类对灰度的敏感度大于彩色的敏感度。

2、程序实现

JPEG 的压缩算法



编码器：

1. 把RGB转换成YCbCr颜色模型，并进行二次采样
2. 对8x8的图像块进行二维DCT变换
3. 根据亮度量化表和色度量化表进行量化
4. 进行熵编码的准备：ZigZag编序并对DC进行DCPM编码，对AC进行游长编码
5. 哈夫曼熵编码成二进制流

解码器：

1. 由于编码采用无损方式，根据建立好的哈夫曼表进行解码
2. 对DC和AC进行解码，根据ZigZag还原成8x8的图像块
3. 反量化
4. 逆DCT变换
5. 转换成RGB颜色空间

src/中的TestModule.ipynb负责测试各个模块的实现，pro2.ipynb负责jpeg压缩，具体见TestModule.html和pro2.html

编码器

1. 首先将图像的长和高转换为8的整数倍，因为之后需要进行8x8的块编码。

```
import argparse
import os
import math
import numpy as np
from scipy import fftpack
from PIL import Image
from matplotlib import pyplot as plt
from skimage import img_as_ubyte
import cv2
# from scipy.misc import imread,imsave
import imageio
```

把图像的长和高转换成8的倍数，并把RGB转换成YCbCr颜色模型

```
img1 = imageio.imread('动物照片.jpg')
img2 = imageio.imread('动物卡通图片.jpg')

# 块的大小
B=8
# 图片的大小
height1,width1=(np.array(img1.shape[:2])/B * B).astype(np.int32)
img1=img1[:height1,:width1]
height2,width2=(np.array(img2.shape[:2])/B * B).astype(np.int32)
img2=img2[:height2,:width2]
```

2. 把RGB转换成YCbCr颜色模型

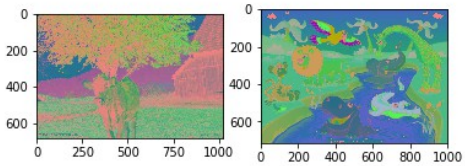
> Y'为颜色的亮度(luma)成分、而CB和CR则为蓝色和红色的浓度偏移量成份。
>
> 使用YCbCr是因为人眼对亮度变换的敏感度要比对色彩变换的敏感度高出很多，所以可以对它的CbCr通道进行接下来的降采样。

```
mat = np.array(
    [[ 0.299, 0.587, 0.144 ],
     [-0.168736, -0.331264, 0.5  ],
     [  0.5, -0.418688, -0.081312]])
mat_inv = np.linalg.inv(mat)
offset = np.array([0, 128, 128])

ycbcr1 = np.zeros(img1.shape)
for x in range(img1.shape[0]):
    for y in range(img1.shape[1]):
        ycbcr1[x, y, :] = np.round(np.dot(mat, img1[x, y, :]) + offset)

ycbcr2 = np.zeros(img2.shape)
for x in range(img2.shape[0]):
    for y in range(img2.shape[1]):
        ycbcr2[x, y, :] = np.round(np.dot(mat, img2[x, y, :]) + offset)

# 显示图片
plt.subplot(1,2,1)
plt.imshow(ycbcr1)
plt.subplot(1,2,2)
plt.imshow(ycbcr2)
plt.show()
imageio.imsave('../ycbcr1.jpg',ycbcr1)
imageio.imsave('../ycbcr2.jpg',ycbcr2)
```



3. 按照4:2:0方式二次降采样

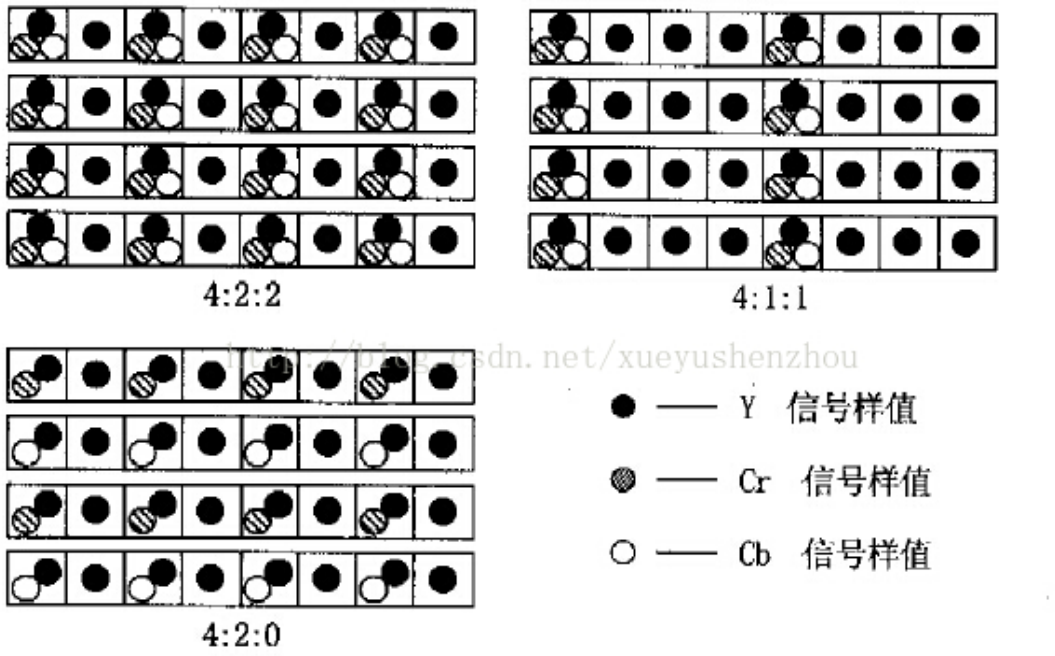


图1 3种取样格式的比较

> 使用2*2平滑滤波（取区域内的平均值）进行下采样。

```
# 每两个像素垂直采样一次
vertical_subsample=2
# 每两个像素水平采样一次
horizontal_subsample=2

# 2*2平滑滤波（取区域内的平均值）
crf1=cv2.boxFilter(ycbcr1[:,:,1],ddepth=-1,ksize=(2,2))
cbf1=cv2.boxFilter(ycbcr1[:,:,2],ddepth=-1,ksize=(2,2))
crf2=cv2.boxFilter(ycbcr2[:,:,1],ddepth=-1,ksize=(2,2))
cbf2=cv2.boxFilter(ycbcr2[:,:,2],ddepth=-1,ksize=(2,2))
# 每隔一行和一列采样，即2*2区域内采样一次
crsub1=crf1[::vertical_subsample,::horizontal_subsample]
cbsub1=cbf1[::vertical_subsample,::horizontal_subsample]
crsub2=crf2[::vertical_subsample,::horizontal_subsample]
cbsub2=cbf2[::vertical_subsample,::horizontal_subsample]
# 将三个通道下采样后的像素值存入列表
sub_img1=[ycbcr1[:,:,0],crsub1,cbsub1]
sub_img2=[ycbcr2[:,:,0],crsub2,cbsub2]

# 输出大小检验
print("图片1: \nY值采样点数量",ycbcr1[:,:,0].size)
print("Cr值采样点数量",crsub1.size)
print("Cb值采样点数量",cbsub1.size)
# print("\n")
print("图片2: \nY值采样点数量",ycbcr2[:,:,0].size)
print("Cr值采样点数量",crsub2.size)
print("Cb值采样点数量",cbsub2.size)
# print(imSub)
```

```
图片1:
Y值采样点数量 698368
Cr值采样点数量 174592
Cb值采样点数量 174592
图片2:
Y值采样点数量 715000
Cr值采样点数量 179000
Cb值采样点数量 179000
```

通过计算发现Y通道采样点的个数为Cb和Cr通道的4倍

4. 对8x8的图像块进行二维DCT变换

> DCT变换主要用于减少高频内容，比傅立叶变换好的地方就是基函数是实函数，可以利用两次一维DCT变换加速运算。

$$C(x) = \begin{cases} \frac{\sqrt{2}}{2} & x = 0 \\ 1 & x \neq 0 \end{cases}$$

$$F(u,v) = \frac{C(u)C(v)}{4} \sum_{i=0}^{7} \sum_{j=0}^{7} cos\frac{(2i+1)u\pi}{16} cos\frac{(2j+1)v\pi}{16} f(i,j)$$

```python
# 二维DCT变换
def DCT_Transform(block):
    # 使范围保持在-127~128
    block = block -128
    res = np.zeros(block.shape)
    C = lambda x: math.sqrt(2)/2 if x==0 else 1
    for u in range(8):
            for v in range(8):
                sum = 0
                for i in range(8):
                    for j in range(8):
                        sum+=math.cos(math.pi*(2*i+1)*u/16)*math.cos(math.pi*v*(2*j+1)/16)*(block[i][j])
                res[u,v] = round(sum *C(u)*C(v)/4)
    return res

# 书上例子用于测试
block = np.array([
                [200, 202, 189, 188, 189, 175, 175, 175],
                [200, 203, 198, 188, 189, 182, 178, 175],
                [203, 200, 200, 195, 200, 187, 185, 175],
                [200, 200, 200, 200, 197, 187, 187, 187],
                [200, 205, 200, 200, 195, 188, 187, 175],
                [200, 200, 200, 200, 200, 190, 187, 175],
                [205, 200, 199, 200, 191, 187, 187, 175],
                [210, 200, 200, 200, 188, 185, 187, 186]
            ])

print(DCT_Transform(block))
```

```
[[ 515.   65.  -12.    4.    1.    2.   -8.    5.]
 [ -16.    3.    2.    0.   -0.  -11.   -2.    3.]
 [ -12.    6.   11.   -1.    3.    0.    1.   -2.]
 [  -8.    3.   -4.    2.   -2.   -3.   -5.   -2.]
 [   0.   -2.    7.   -5.    4.   -0.   -1.   -4.]
 [   0.   -3.   -1.    0.    4.    1.   -1.    0.]
 [   3.   -2.   -3.    3.    3.   -1.   -1.    3.]
 [  -2.    5.   -2.    4.   -2.    2.   -3.    0.]]
```

5. Y根据亮度量化表，Cb和Cr根据色度量化表进行量化

> 有损变化，JPEG产生信息丢失的主要原因。

```
# 亮度和色度量化表
QY=np.array([[16,11,10,16,24,40,51,61],
             [12,12,14,19,26,48,60,55],
             [14,13,16,24,40,57,69,56],
             [14,17,22,29,51,87,80,62],
             [18,22,37,56,68,109,103,77],
             [24,35,55,64,81,104,113,92],
             [49,64,78,87,103,121,120,101],
             [72,92,95,98,112,100,103,99]])

QC=np.array([[17,18,24,47,99,99,99,99],
             [18,21,26,66,99,99,99,99],
             [24,26,56,99,99,99,99,99],
             [47,66,99,99,99,99,99,99],
             [99,99,99,99,99,99,99,99],
             [99,99,99,99,99,99,99,99],
             [99,99,99,99,99,99,99,99],
             [99,99,99,99,99,99,99,99]])

# 根据亮度量化表进行量化
print(np.round(DCT_Transform(block)/QY))
```

```
[[ 32.   6.  -1.   0.   0.   0.  -0.   0.]
 [ -1.   0.   0.   0.  -0.  -0.  -0.   0.]
 [ -1.   0.   1.  -0.   0.   0.   0.  -0.]
 [ -1.   0.  -0.   0.  -0.  -0.  -0.  -0.]
 [  0.  -0.   0.  -0.   0.  -0.  -0.  -0.]
 [  0.  -0.  -0.   0.   0.   0.  -0.   0.]
 [  0.  -0.  -0.   0.   0.  -0.  -0.   0.]
 [ -0.   0.  -0.   0.  -0.   0.  -0.   0.]]
```

6. ZigZag编序扫描

> 从低频内容向高频内容扫描，为RLC作准备。

```
def block_to_zigzag(block):
    return np.array([block[point] for point in zigzag_points(*block.shape)])

# 返回横纵坐标
def zigzag_points(rows, cols):
    # 方向选择
    UP, DOWN, RIGHT, LEFT, UP_RIGHT, DOWN_LEFT = range(6)

    # 移动坐标
    def move(direction, point):
        return {
            UP: lambda point: (point[0] - 1, point[1]),
            DOWN: lambda point: (point[0] + 1, point[1]),
            LEFT: lambda point: (point[0], point[1] - 1),
            RIGHT: lambda point: (point[0], point[1] + 1),
            UP_RIGHT: lambda point: move(UP, move(RIGHT, point)),
            DOWN_LEFT: lambda point: move(DOWN, move(LEFT, point))
        }[direction](point)

    # 判断是否在边界内
    def inbounds(point):
        return 0 <= point[0] < rows and 0 <= point[1] < cols

    # 左上角开始
    point = (0, 0)

    # True when moving up-right, False when moving down-left
    move_up = True
```

```
    for i in range(rows * cols):
        yield point
        if move_up:
            if inbounds(move(UP_RIGHT, point)):
                point = move(UP_RIGHT, point)
            else:
                move_up = False
                if inbounds(move(RIGHT, point)):
                    point = move(RIGHT, point)
                else:
                    point = move(DOWN, point)
        else:
            if inbounds(move(DOWN_LEFT, point)):
                point = move(DOWN_LEFT, point)
            else:
                move_up = True
                if inbounds(move(DOWN, point)):
                    point = move(DOWN, point)
                else:
                    point = move(RIGHT, point)

z_array = block_to_zigzag(quan_block)
print(z_array)
```

```
[ 32.   6.  -1.  -1.   0.  -1.   0.   0.   0.  -1.   0.   0.   1.   0.   0.
   0.  -0.  -0.  -0.  -0.   0.   0.  -0.   0.   0.   0.  -0.  -0.   0.  -0.
   0.  -0.  -0.  -0.  -0.  -0.   0.  -0.   0.   0.  -0.   0.   0.  -0.  -0.
  -0.   0.   0.  -0.   0.   0.   0.  -0.  -0.  -0.  -0.  -0.  -0.   0.  -0.
   0.   0.  -0.   0.]
```

7. AC系数的RLC游长编码

> symbol1：（runlength,size）
>
> Symbol2：（amplitude）

```
# 输入VALUE，得到SIZE
def int_size(number):
    str = ""
    if number==0:
        return 0
    if number < 0:
        number = abs(number)
        l = len(bin(number)) - 2
        str = bin(number)[-l:]
        str = reverse_str(str)
    else:
        l = len(bin(number)) - 2
        str = bin(number)[-l:]
    # return (l,str)
    return l

# 输入VALUE，得到幅值AMPLITUDE
def int_to_binstr(n):
    if n == 0:
        return ''

    binstr = bin(abs(n))[2:]

    # change every 0 to 1 and vice verse when n is negative
    return binstr if n > 0 else binstr_flip(binstr)
```

```python
# 返回 (RUNLENGTH, SIZE) 和 "AMPLIYTUDE"
def run_length_encode(arr):
    # determine where the sequence is ending prematurely
    last_nonzero = -1
    # print(arr)
    for i, elem in enumerate(arr):
        if elem != 0:
            last_nonzero = i
    # print(last_nonzero)

    # each symbols1 is a (RUNLENGTH, SIZE) tuple
    # each symbols2 is a AMPLITUDE string
    symbols1 = []
    symbols2 = []
    run_length = 0

    for i, elem in enumerate(arr):
        if i > last_nonzero:
            symbols1.append((0, 0))
            symbols2.append(0)
            break
        elif elem == 0 and run_length < 15:
            run_length += 1
        else:
            # size = bits_required(elem)
            symbols1.append((run_length, int_size(elem)))
            symbols2.append(int_to_binstr(elem))
            run_length = 0
    return symbols1,symbols2
```

```python
# 注意转换为32位整型
z_array = z_array.astype(np.int32)
# 注意从第二个下标开始进行游长编码
symbols1,symbols2 = run_length_encode(z_array[1:])
print("symbols1:(RUNLENGTH,SIZE)\n",symbols1)
print("symbols2:(AMPLITUDE)\n",symbols2)
```

```
symbols1:(RUNLENGTH,SIZE)
 [(0, 3), (0, 1), (0, 1), (1, 1), (3, 1), (2, 1), (0, 0)]
symbols2:(AMPLITUDE)
 ['110', '0', '0', '0', '0', '1', 0]
```

注意如果VALUE为0，AMPLITUDE为0，而不是字符串'0'，因为它代表-1。

8. DC系数的DCPM编码

> 根据DCT变换，每个块都会得到一个DC分量。对于每个颜色空间的DC分量，计算差值，使得过去的值可以用现在的值加上之前的累加和来表示。
>
> （size，amplitude）

```python
# 输入DC系数数组，输出SIZE和AMPLITUDE
def dcpm(dc_array):
    sizes = []
    amplitudes = []
    for i in range(dc_array.shape[0]):
        if i==0:
            sizes.append(int_size(dc_array[i]))
            amplitudes.append(int_to_binstr(dc_array[i]))
        else:
            sizes.append(int_size(dc_array[i]-dc_array[i-1]))
            amplitudes.append(int_to_binstr(dc_array[i]-dc_array[i-1]))
    return sizes,amplitudes

# 以书中例子作为测试
dc_arrays = np.array([150,155,149,152,144])
sizes,amplitudes = dcpm(dc_arrays)
print("SIZE:\n",sizes)
print("AMPLITUDE:\n",amplitudes)
```

```
SIZE:
 [8, 3, 3, 2, 4]
AMPLITUDE:
 ['10010110', '101', '001', '11', '0111']
```

9. 哈夫曼熵编码

   建立四个哈夫曼树，两个用于Y通道的DC和AC编码，两个用于CbCr通道的DC和AC编码

   其中对于DC系数，对SIZE进行哈夫曼编码，对于AC系数，对Symbol1（RUNLENGTH，SIZE）元组进行哈夫曼编码

   具体实现见huffmantree.py，或TestModule.ipynb

```
H_DC = HuffmanTree(sizes)
H_AC = HuffmanTree(symbols1)

print("DC SIZE: encode\n",H_DC.value_to_bitstring_table())
print("AC tuple:encode\n",H_AC.value_to_bitstring_table())
```

```
DC SIZE: encode
 {8: '00', 2: '01', 4: '10', 3: '11'}
AC tuple:encode
 {(3, 1): '00', (0, 3): '010', (1, 1): '011', (0, 1): '10', (0, 0): '110', (2, 1): '111'}
```

解码器

1. 读取二进制文件并进行哈夫曼解码，然后根据存储的AMPLITUDE解码DC和AC系数并还原成z型数组（此处省略见decode.py）
2. 将z型数组还原成8x8的块

```
# 解码器
# 具体二进制文件的读取和解码见decode.py
# 此处整体码字的解码，从得到z型数组开始

# 将z型数组还原成8x8的块
def zigzag_to_block(zigzag):
    # assuming that the width and the height of the block are equal
    rows = cols = int(math.sqrt(len(zigzag)))

    if rows * cols != len(zigzag):
        raise ValueError("length of zigzag should be a perfect square")

    block = np.empty((rows, cols), np.int32)

    for i, point in enumerate(zigzag_points(rows, cols)):
        block[point] = zigzag[i]

    return block

z_array_decode = zigzag_to_block(z_array)
print(z_array_decode)
```

```
[[32  6 -1  0  0  0  0  0]
 [-1  0  0  0  0  0  0  0]
 [-1  0  1  0  0  0  0  0]
 [-1  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0]]
```

3. 逆量化

```
# 逆量化，直接和量化表点乘
dequan_block = z_array_decode*QY
print(dequan_block)
```

```
[[512  66 -10   0   0   0   0   0]
 [-12   0   0   0   0   0   0   0]
 [-14   0  16   0   0   0   0   0]
 [-14   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0]]
```

4. 逆DCT变换

$$\widetilde{f}(i,j)=\sum_{u=0}^{7}\sum_{v=0}^{7}\frac{C(u)C(v)}{4}cos\frac{(2i+1)u\pi}{16}cos\frac{(2j+1)v\pi}{16}F(u,v)$$

```
def IDCT_TRANSFORM(block):
    res = np.zeros(block.shape)
    C = lambda x: math.sqrt(2)/2 if x==0 else 1
    for i in range(8):
            for j in range(8):
                sum = 0
                for u in range(8):
                    for v in range(8):
                        sum+=C(u)*C(v)/4*math.cos(math.pi*(2*i+1)*u/16)*math.cos(math.pi*v*(2*j+1)/16)*block[v][u]
                res[j][i] = round(sum)+128
    return res

idct_block = IDCT_TRANSFORM(dequan_block)
print(idct_block)
```

```
[[ 199.  196.  191.  186.  182.  178.  177.  176.]
 [ 201.  199.  196.  192.  188.  183.  180.  178.]
 [ 203.  203.  202.  200.  195.  189.  183.  180.]
 [ 202.  203.  204.  203.  198.  191.  183.  179.]
 [ 200.  201.  202.  201.  196.  189.  182.  177.]
 [ 200.  200.  199.  197.  192.  186.  181.  177.]
 [ 204.  202.  199.  195.  190.  186.  183.  181.]
 [ 207.  204.  200.  194.  190.  187.  185.  184.]]
```

5. 得到重构误差

```
# 计算重构误差
e = block - idct_block
print(e)
```

```
[[ 1.  6. -2.  2.  7. -3. -2. -1.]
 [-1.  4.  2. -4.  1. -1. -2. -3.]
 [ 0. -3. -2. -5.  5. -2.  2. -5.]
 [-2. -3. -4. -3. -1. -4.  4.  8.]
 [ 0.  4. -2. -1. -1. -1.  5. -2.]
 [ 0.  0.  1.  3.  8.  4.  6. -2.]
 [ 1. -2.  0.  5.  1.  1.  4. -6.]
 [ 3. -4.  0.  6. -2. -2.  2.  2.]]
```

3、结果对比

JPEG 和 GIF 图像格式的视觉效果和压缩效果比较（包括压缩率的比较和失真度的比较，各占10%）。

视觉效果

原始动物图像和动物卡通图像

压缩后的图像

具体见pro2中的

动物gif.gif	今天 下午5:51
动物jpg.jpg	2018年11月20日 下午11:28
卡通gif.gif	今天 下午5:52
卡通jpg.jpg	2018年11月20日 下午11:28

JPEG压缩并解码重构出来的图像（由于原图像的高度不为8的整数倍，故先进行padding后再进行压缩处理）

使用在线网站将jpg文件转换成gif文件

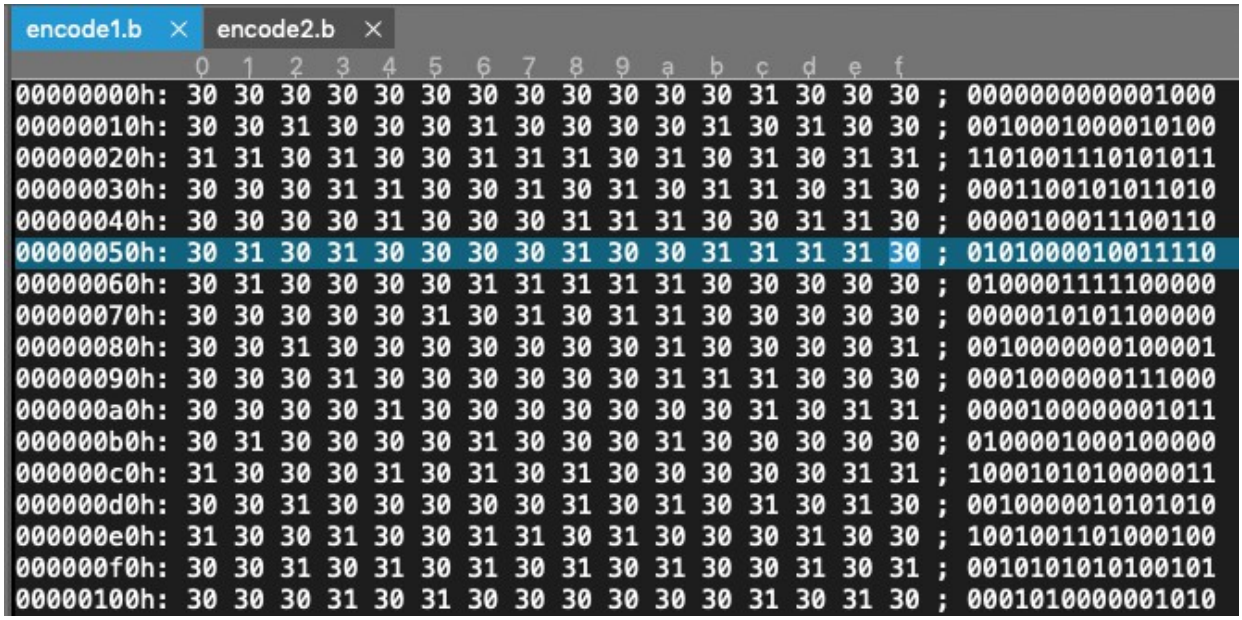
JPEG压缩并解码重构出来的图像（由于原图像的高度不为8的整数倍，故先进行padding后再进行压缩处理）

使用在线网站将jpg文件转换成gif文件

压缩率

在写二进制文件的时候对写入的位数进行计算，或者使用UltraEdit查看十六进制格式。

1. 动物图片编码后的位数为**1329860**，由于原图片为经过压缩后的jpg文件，对其使用RGB真彩色位图的计算方式，得到总位数为**17,160,000**，压缩率为**0.077497669**。



将jpg转换为gif的格式后的文件大小为**616KB**，压缩率为**0.2871794872**

2. 同理卡通图片原位数为**16,760,832**，编码后为**755615**，压缩率为**0.04508218924**

   动物卡通图片的gif大小为**417KB**，压缩率为**0.1990354655**

动物图像	jpeg压缩	gif
17,160,000 bits （2.145MB)	1329860 bits（166.2325KB)	616KB
压缩率	0.077497669	0.2871794872

动物卡通图像	jpeg压缩	gif
16,760,832 bits （2.095MB)	755615 bits（94.451KB)	417KB
压缩率	0.04508218924	0.1990354655

失真度

$$MSE = \sigma = \frac{1}{N}\sum_{n=1}^{N}(x_n - y_n)^2$$

$$SNR = 10log_{10}\frac{\sigma_x^2}{\sigma_d^2}$$

其中 σ_x 为原始数据均方，σ_d 为均方差 MSE。

```
# 失真度计算
# 适用于RGB图像，注意shape相同

# 均方差
def MSE(reCons,origin):
    res = 0
    for i in range(reCons.shape[0]):
        for j in range(reCons.shape[1]):
                res += sum((reCons[i,j] - origin[i,j]) ** 2)
    res = res / (reCons.shape[0]*reCons.shape[1])
    return res

# 信噪比
def SNR(reCons,origin):
    res = 0;
    for i in range(origin.shape[0]):
        for j in range(origin.shape[1]):
                res += sum(origin[i,j]** 2)
    res = res / (origin.shape[0]*origin.shape[1])
    return 10*math.log(res**2/MSE(reCons, origin)**2,10)
```

动物图片	JPEG	GIF
均方差	141.428259886	131.650367714
信噪比	6.844158124101839	7.466440552682235

动物卡通图片	JPEG	GIF
均方差	111.04381958	98.9218447552
信噪比	8.993319475662155	9.997363176949868

```
print("JPEG\n")
print("动物图片的均方差: ",MSE(rgb1,img1))
print("动物卡通图片的均方差: ",MSE(rgb2,img2))
print("动物图片的信噪比: ",SNR(rgb1,img1))
print("动物卡通图片的信噪比: ",SNR(rgb2,img2))

gif1 = imageio.imread('../动物gif.gif')
gif2 = imageio.imread('../卡通gif.gif')
gif1 = gif1[:,:,0:3]
gif2 = gif2[:,:,0:3]
print("GIF\n")
print("动物图片的均方差: ",MSE(gif1,img1))
print("动物卡通图片的均方差: ",MSE(gif2,img2))
print("动物图片的信噪比: ",SNR(gif1,img1))
print("动物卡通图片的信噪比: ",SNR(gif2,img2))
```

```
JPEG

动物图片的均方差:  141.428259886
动物卡通图片的均方差:  111.04381958
动物图片的信噪比:  6.844158124101839
动物卡通图片的信噪比:  8.993319475662155
GIF

动物图片的均方差:  131.650367714
动物卡通图片的均方差:  98.9218447552
动物图片的信噪比:  7.466440552682235
动物卡通图片的信噪比:  9.997363176949868
```