第一题：

8.（a）赫夫曼算法需要有关信息源的先验统计知识，而这样的信息很难获得，特别是多媒体类应用，数据在到达之前是未知的，所以无法得到这些统计数据。而且统计数据符号表的传输依然是一笔很大的开销。

（b）i. 除去中间的 00 是 NEW 之外，传输的字符是 bacc，树的变化在 ii。

①第一次传的时候 01 是 b，之后树变换为图 1；

②之后传输的是 01，由图 1 可以看出 01 此时为 a，树变换为图 2；

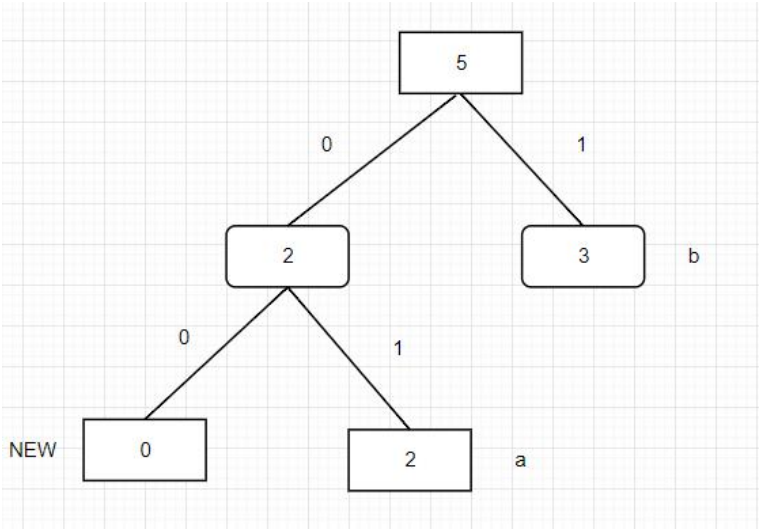
③之后传输 00，也就是一个新字符，新字符为 10 查找初始的编码为 c，此时树变换为图 3-2；

④之后传输 101，由图 3-2 可以看出 101 此时为 c，树变换为图 4。

综上传输的字符为 bacc。

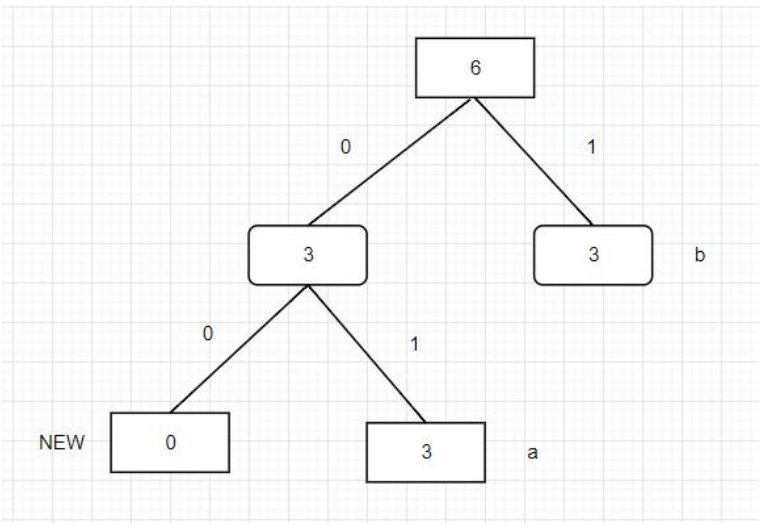
ii.

传第一个 01（b）之后：



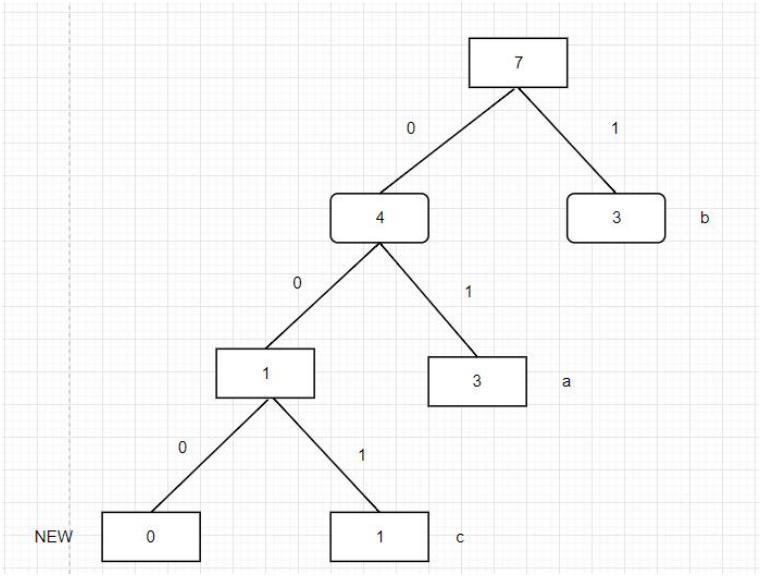
（图 1）

第二个 01（a）之后：

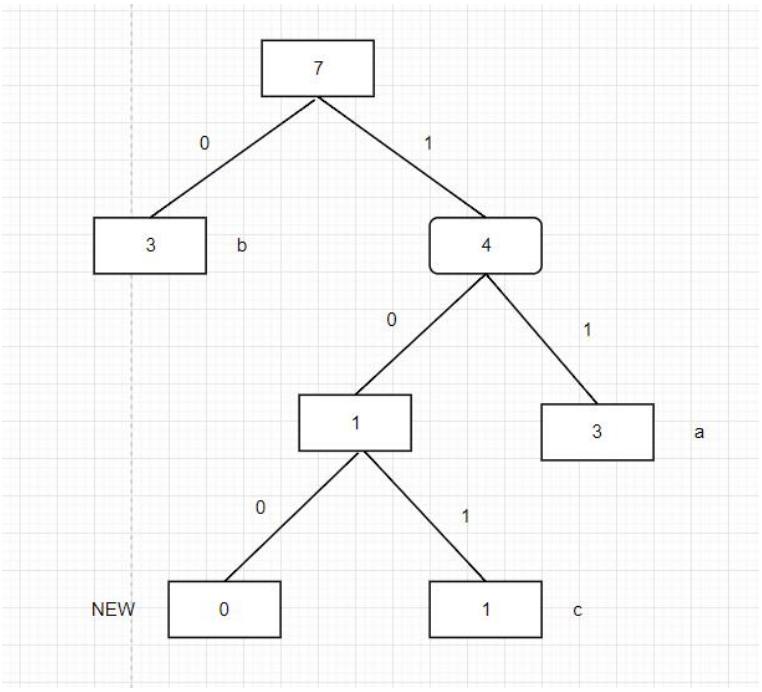


（图 2）

第三个 00，第四个 10（c）之后：

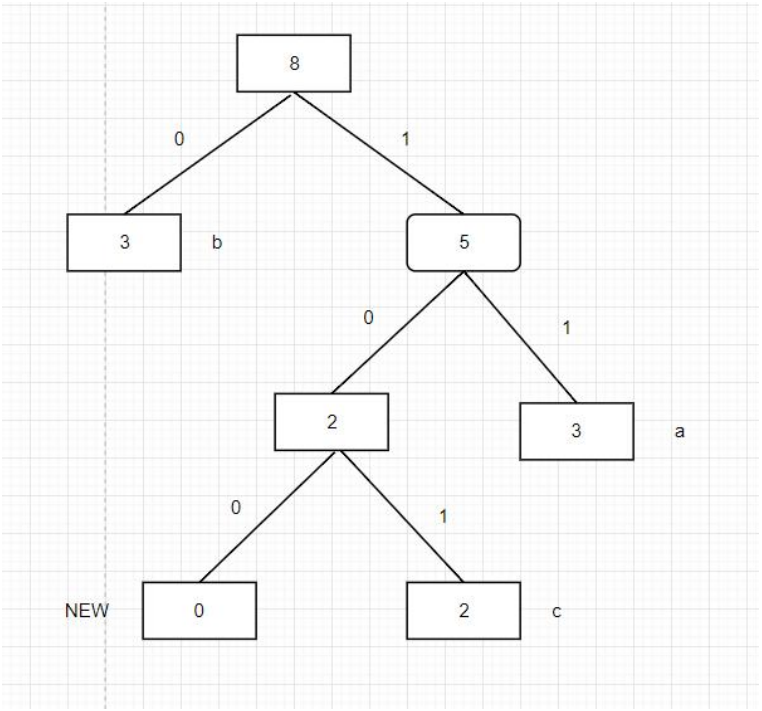


（图 3-1）



（图 3-2）

第五个 101（c）之后：



（图 4）

第二题：

理论原因分析：

GIF 压缩是采用无损压缩技术，图像不多于 256 色则既可以减少文件的大小又可以保

持质量，普遍用于图标按钮等只需要少量颜色的图像，如黑白图。主要由 87a 和 87b 两个版本。

JPEG 压缩是有损压缩技术，通过对图像进行色彩空间转换，取样，DCT，量化，熵编码的过程来对图像进行压缩，不适用与线条绘图和其他文字和图标的图形。

GIF 压缩是将图片转换成 256 色进行传输，256 色中每个颜色都有索引值，然后传输方传输索引值（如果索引表没有统一，那么连同索引表一起传输）。Jpeg 是将每个 8*8 压缩并编码成大小不一的数据流，然后将所有数据流传输，接收方接收数据流并根据 huffman 表和量化表进行还原（如果双方表没有统一，也需要连同图片一起传输）。

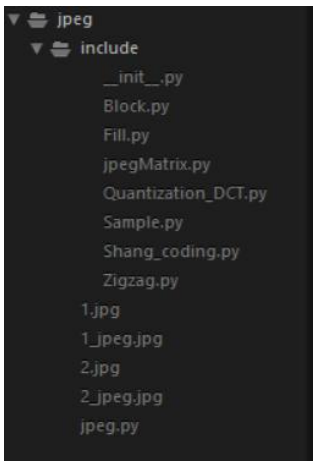
所以总的来说，GIF 适用于颜色数量少、图像细节不明显的图像，因此本题中更适合卡通图片的压缩；而 JPEG 压缩的话，对于颜色多细节多的图像效果会比 GIF 好得多，细节多的图像在 JPEG 处理之后，压缩比比较高，更适用于本题中的动物图片。

程序实现部分（JPEG）：

程序实现语言：python

版本：py2.7

程序文件概括：



主函数为 jpeg.py，include 文件夹中是编码和解码的各个函数（Fill 填充、Sample

采样、Block 取块、Quantization_DCT 进行 DCT 并量化、Zigzag 排序、Shang_coding 进行 DC 的 DPCM 编码和 AC 的游长编码，并转化为熵编码）。

运行：jpeg.py 目录下直接运行即可，压缩文件与 py 文件在同个目录下，生成的图片文件存储为？_jpeg.jpg，如下，其中 DCT 和 IDCT 因为没有使用内置的 dot 方法，导致耗时约 20s，请耐心等待：

```
C:\Users\ASUS\Desktop\jpeg>python jpeg.py
Please input the picture name: 2
Reading image 2.jpg ...
Fill...
Block...
Quantization and DCT...
Zigzag sort...
DC and AC coding...
shang coding...
De Shang coding...
De_DC_and_AC_coding...
De_Zigzag...
De_Quantization_DCT...
De_Block...
De fill...
waiting new pic...

C:\Users\ASUS\Desktop\jpeg>
```

①颜色转换（jpeg.py 中）

根据公式进行颜色的转换，RGB 转 YUV 如下：

```
31  ····#由每个像素点的RGB生成每个像素点的YUV,将RGB表转换成YUV三个matrix
32  ····for·i·in·range(0,·size[0]):
33  ········for·j·in·range(0,·size[1]):
34  ············pixel·=·image.getpixel((i,·j))
35  ············Y_matrix.append(0.299*pixel[0]·+·0.587*pixel[1]·+·0.114*pixel[2])
36  ············U_matrix.append(-0.1687*pixel[0]·-·0.3313*pixel[1]·+·0.5*pixel[2]·+·128)
37  ············V_matrix.append(0.5*pixel[0]·-·0.419*pixel[1]·-·0.081*pixel[2]+128)
```

YUV 转 RGB 如下：

```
148 ····for·i·in·range(0,·len(Y_matrix)):
149 ········R.append(Y_matrix[i]·+·1.402*(V_matrix[i]·-·128))
150 ········G.append(Y_matrix[i]·-·0.34414*(U_matrix[i]·-·128)·-·0.71414*(V_matrix[i]·-·128))
151 ········B.append(Y_matrix[i]·+·1.772*(U_matrix[i]·-·128))
```

②填充（Fill.py）

由于图片的长宽都需要满足是 16 的倍数（采样长宽会缩小 1/2 和取块长宽会缩小 1/8），所以需要将图片填充至 16 的倍数。

填充的函数如下，newsize 最终是填充结束的大小，size 是原来的大小。填充的像素经测试，不会影响原来的图片，所以填充 0 即可。

```
def fill(matrix, size,newsize):
    newsize0 = size[0]
    newsize1 = size[1]
    fill_size1 = size[1]%16
    if fill_size1 != 0:
        newsize1 = size[1]+16-fill_size1

    fill_size0 = size[0]%16
    if fill_size0 != 0:
        newsize0 = size[0]+16-fill_size0
    newsize[0] = newsize0
    newsize[1] = newsize1
    new_matrix = []
    for i in range(0, newsize1):
        for j in range(0, newsize0):
            if i < size[1] and j < size[0]:
                new_matrix.append(matrix[i*size[0] + j])
            # elif i < size[1] and j >= size[0]:
            #     new_matrix.append(matrix[(i+1)*size[0]-1])
            # elif i >= size[1] and j < size[0]:
            #     new_matrix.append(matrix[(size[1]-2)*(size[0]-1) + j])
            # elif i >= size[1] and j >= size[0]:
            #     new_matrix.append(matrix[(size[1]-1) * (size[0] - 1)])
            else:
                new_matrix.append(0)
    return new_matrix
```

返回时，要进行截取，使图片变回原来的大小：

```
def De_fill(matrix, size):
    newsize0 = size[0]
    newsize1 = size[1]
    fill_size1 = size[1]%16
    if fill_size1 != 0:
        newsize1 = size[1]+16-fill_size1

    fill_size0 = size[0]%16
    if fill_size0 != 0:
        newsize0 = size[0]+16-fill_size0
    new_matrix = []
    # print newsize0,newsize1
    # print size
    for i in range(0, newsize1):
        for j in range(0, newsize0):
            if i < size[1] and j < size[0]:
                # if j == size[0]-1:
                #     print j,matrix[i*newsize0 +j]
                # if j == size[0]-2:
                #     print j,matrix[i*newsize0 +j]
                new_matrix.append(matrix[i*newsize0 + j])
    return new_matrix
```

③采样（Sample.py）

采样是对 YUV 三个表中的 UV 进行的，采样操作比较简单，本次使用的采样是比较容易实现的 4:1:1，也就是每一个 2*2 方块中，只去左上角的值。

```python
# coding=utf-8
def Sample(matrix, size):
    temp_matrix = []
    for Y in range(0, size[1] / 2):
        for X in range(0, size[0] / 2):
            start_point = Y * 2 * size[0] + X * 2
            temp_matrix.append(matrix[start_point])
    return temp_matrix

def De_Sample(matrix, size):
    temp_matrix = []
    for Y in range(0, size[1]):
        for i in range(0, 2):
            for X in range(0, size[0]):
                temp_matrix.append(matrix[X + Y*size[0]])
                temp_matrix.append(matrix[X + Y*size[0]])
    return temp_matrix
```

④取块（Block.jpg）

JPEG 标准中，使用的是 8*8 块的处理，所以把 YUV 截成若干 8*8 矩阵。

```python
MN = 8;

def Block(matrix, sizeX,sizeY):
    little_matrix = []
    #start_point是每个8*8矩阵的第一个点
    for Y in range(0, sizeY / MN):
        for X in range(0, sizeX / MN):
            start_point = Y * sizeX * 8 + X * 8
            #初始化中间矩阵
            mid_matrix = []
            #生成中间矩阵
            for i in range(0, MN):
                for j in range(0, MN):
                    mid_matrix.append(matrix[start_point + i*sizeX + j])
            #压栈
            little_matrix.append(mid_matrix)
    return little_matrix

def De_Block(little_matrix, size):
    matrix = []
    for Y in range(0, size[1] / MN):
        for count1 in range(0, MN):
            for X in range(0, size[0] / MN):
                for count2 in range(0, MN):
                    if X+Y*(size[0]/MN) >= len(little_matrix):
                        print "?0"
                    elif count2+count1*MN >= len(little_matrix[X+Y*(size[0]/MN)]):
                        print "?1 %d %d %d %d %d %d"%(X,Y,count1, count2,count2+count1*MN, len(little_matrix[X+Y*(size[0]/MN)]))
                    matrix.append(little_matrix[X+Y*(size[0]/MN)][count2+count1*MN])
                    # matrix.append(count2 + X*MN + count1*MN*(size[0]/MN) + Y*MN*(size[0]/MN)*MN)
    return matrix
```

⑤DCT 和量化（Quantization_DCT.py）

DCT 函数和 IDCT 函数，C（）函数求出 DCT 中的系数。

```python
def C(u, MN):
    return math.sqrt(1.0 / MN) if u == 0 else math.sqrt(2.0 / MN)

def DCT(matrix, DCT_matrix, MN):
    for u in range(0, MN):
        for v in range(0,MN):
            mid = 0
            for i in range(0 ,MN):
                for j in range(0, MN):
                    mid += math.cos((2*i+1)*u*math.pi/(2*MN)) * math.cos((2*j+1)*v*math.pi/(2*MN)) * matrix[i*MN+j]
            DCT_matrix.append(int(round(C(u,MN)*C(v,MN)*mid)))
```

```
def oneIDCT(array, i):
    mid = 0
    for u in range(0, MN):
        mid += math.cos((2*i+1) * u * math.pi / 16) * array[u] * C(u,MN)
    return mid

def De_DCT(DCT_matrix):
    temp_matrix = list(range(64))
    temp_matrix2 = list(range(64))
    #列IDCT求出temp_matrix
    for i in range(0, MN):
        array = []
        for j in range(0, MN):
            array.append(DCT_matrix[i + j*MN])
        for u in range(0, MN):
            temp_matrix[u*MN + i] = oneIDCT(array, u)
    #行IDCT求出temp_matrix2
    for i in range(0, MN):
        array = []
        for j in range(0, MN):
            array.append(temp_matrix[i*MN + j])
        for u in range(0, MN):
            temp_matrix2[i*MN + u] = (oneIDCT(array, u))
    return temp_matrix2
```

JPEG 中重要的一环，亮度和色度量化表在 jpgeMatrix.py 中。

量化函数：

```
#量化函数
def Luminance_Quantization(DCT_matrix, Quantization_DCT_matrix, MN):
    for i in range(0, MN*MN):
        Quantization_DCT_matrix.append(int(round(1.0 * DCT_matrix[i] / Luminance_Quantization_Matrix[i])))
def Chroma_Quantization(DCT_matrix, Quantization_DCT_matrix, MN):
    for i in range(0, MN*MN):
        Quantization_DCT_matrix.append(int(round(1.0 * DCT_matrix[i] / Chroma_Quantization_Matrix[i])))
```

反量化函数：

```
def De_Luminance_Quantization(DCT_matrix, Quantization_DCT_matrix):
    for i in range(0, MN*MN):
        DCT_matrix.append(Quantization_DCT_matrix[i] * Luminance_Quantization_Matrix[i])
def De_Chroma_Quantization(DCT_matrix, Quantization_DCT_matrix):
    for i in range(0, MN*MN):
        DCT_matrix.append(Quantization_DCT_matrix[i] * Chroma_Quantization_Matrix[i])

```

⑥Zigzag 排序

Zigzag 排序没有直接使用初始化一个 Zigzag 排序，因为 Zigzag 的排序有规律可寻。Index 有四个方向：右上、右、下、左下。除了右上和左下，每次变换方向为右或者下，则下一步是右上或者左下。可以视为一个在 8*8 范围内的旋转（右上--右--下，左下--下--右）。

```
def Zigzag(matrix, ZIGZAG_trans_matrix):
    Zigzag_matrix = []
    i = 0
    j = 0
    #move_flag为0代表ij向右上移动，1代表向左下移动
    move_flag = 0
    Zigzag_matrix.append(matrix[0])
    for count in range(0, MN*MN-1):
        #求出i和j，i为横轴，j为纵轴
        if move_flag == 0:
            if In_Range(i+1, j-1):
                i = i+1
                j = j-1
            elif In_Range(i+1, j):
                i = i+1
                move_flag = 1
            elif In_Range(i, j+1):
                j = j+1
                move_flag = 1
        elif move_flag == 1:
            if In_Range(i-1, j+1):
                i = i-1
                j = j+1
            elif In_Range(i, j+1):
                j = j+1
                move_flag = 0
            elif In_Range(i+1, j):
                i = i+1
                move_flag = 0
        if len(ZIGZAG_trans_matrix)!= 64:
            ZIGZAG_trans_matrix.append(j*MN + i)
        Zigzag_matrix.append(matrix[j*MN + i])
    return Zigzag_matrix
```

运行完存储 Zigzag trans 矩阵，方便逆 Zigzag 运算。

```
44  def De_Zigzag(Zigzag_little_matrix,ZIGZAG_trans_matrix):
45      mid_matrix = []
46      if len(Zigzag_little_matrix)!= 64:
47          print len(Zigzag_little_matrix)
48      for j in range(0, len(Zigzag_little_matrix)):
49          for k in range(0, len(ZIGZAG_trans_matrix)):
50              if ZIGZAG_trans_matrix[k] == j:
51                  # print j
52                  mid_matrix.append(Zigzag_little_matrix[k])
53      if len(mid_matrix) != 64:
54          print "error"
55          print mid_matrix
56      return mid_matrix
```

⑦编码

编码的过程包括：将上面的 Zigzag 矩阵转换成（count，number）的游长矩阵形式（可以也把 DC 变换成这种形式，便于一起存储），然后将 number 再次变换成 VLI 形式（位数和 code 的格式），加上前面的 count 最终是（count，位数，code）的格式。

接下来将 count 和位数的组合根据默认的 huffman 表（亮度或者色度的 huffman 表）变换成 0101 的形式。

⑧统计

最后统计上述编码结束后的 01 的 bit 数，与原来图像的 RGB 总位数进行比较，算出压缩

率。

结果对比：

原图和效果图在最后部分，效果图出现略微椒盐化，测试为量化函数有误差。图 1（动物图片）的压缩率在 12 左右，动画图片在 9 左右。失真度发现动物图片的失真度小于动画图片。

GIF 处理之后相比于 JPEG 更加模糊，失真度更高。GIF 的压缩率大概在 2 左右。

总的来说，GIF 的压缩效果和压缩率均不如 JPEG，但是 GIF 的速度更快，可以运用在一些对图片细节要求不高的图片中（如动图），而且 GIF 的另外一个优势就是传输的时候可以从低清晰度开始传输，而 JPEG 是一行一行的传输，传输结束后才能看到整个图形，所以 GIF 对于传输速度慢的情况下更有优势。

原图 1：

效果图 1：


全景网 www.quanjing.com
pm0106-1816hx

原图 2：


图行天下 photophoto.cn No.20160709018370179098

效果图 2：


图行天下 photophoto.cn No.20160709018370179098