多媒体技术第二次作业

第一题

(a) What are the advantages of Adaptive Huffman Coding compared to the original Huffman Coding algorithm?

(b) Assume that Adaptive Huffman Coding is used to code an information source S with a vocabulary of four letters (a, b, c, d). Before any transmission, the initial coding is a = 00, b = 01, c = 10, d = 11. As in the example illustrated in Fig. 7.8, a special symbol NEW will be sent before any letter if it is to be sent the first time.
Figure 7.18 is the Adaptive Huffman tree after sending letters **aabb**. After that, the additional bitstream received by the decoder for the next few letters is 01010010101.

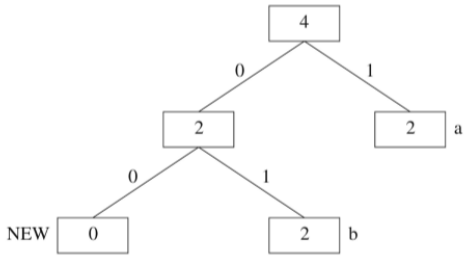


Fig. 7.18 Adaptive Huffman tree

i. What are the additional letters received?
ii. Draw the adaptive Huffman trees after each of the additional letters is received.

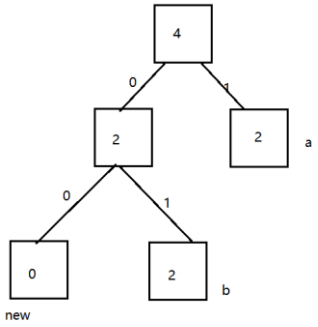
解:

(a) 哈夫曼编码需要有关信息源的先验统计知识，有时很难获取这样的知识，比如直播或流式的音频和视频中，数据在未到达之前是未知的，这种情况下传统哈夫曼编码难以使用。即使能够获得这些统计数字，符号表的传输也会占据很大的开销。

自适应哈夫曼编码作为一种自适应压缩算法，统计的数据随着数据流的到达而动态的收集和更新，各个符号的概率也是基于目前位置收到的数据而非先验知识的。所以这种方法很适合内容和统计数字快速变化的多媒体应用，即**待传输数据概率分布的先验统计知识未知，在这种情况下也能够完成压缩工作**，这是原本的哈夫曼编码做不到的。并且**由于符号表在接收方动态生成，该方法还能够节省符号表传输的开销**，只不过接收方需要在本地做更多工作，但是这相比增大网络传输任务带来的开销要小很多。

(b) 此题先分析解码过程，然后再回答问题:

收到 aabb 字符之后，接收方哈夫曼树为

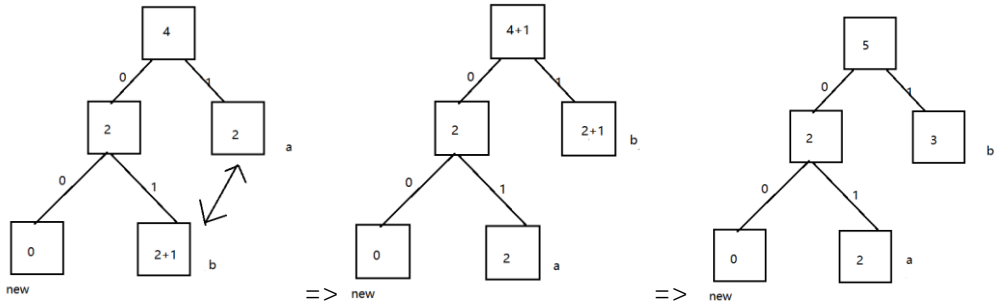


接收方的码表如下：

New	a	b	c	d
00	1	01	10	11

（步骤一）收到串“01010010101”之后将读头 01，解码为字符 b，并输出，输入串读头前进两个位变为“010010101”；

然后接收方更新哈夫曼树为

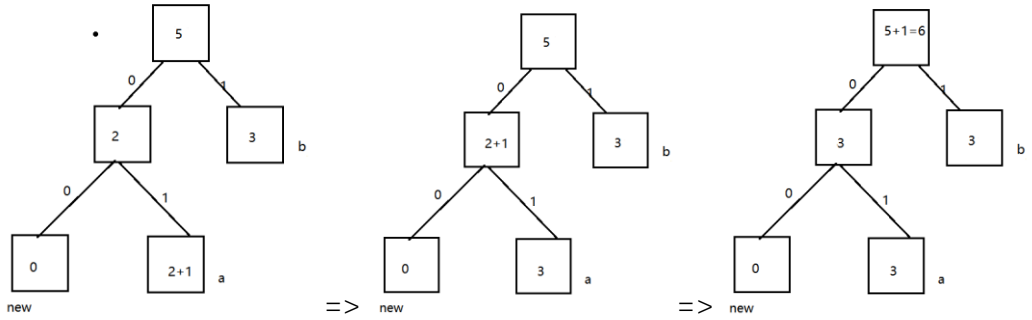


更新码表为：

New	a	b	c	d
00	01	1	10	11

（步骤二）分析剩余串“010010101”之后将读头 01，解码为字符 a，并输出，输入串读头前进两个位变为“0010101”；

然后更新哈夫曼树为：



码表不需要更新，因为树的结构未变为，沿用上一步码表：

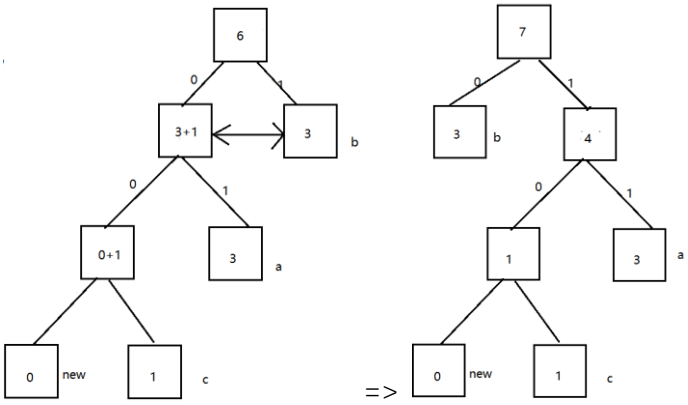
New	a	b	c	d
00	01	1	10	11

（步骤三）分析剩余串“0010101”之后将读头 00，解码为 new，不是 a/b/c/d，略去即可，输入串读头前进两个位变为“10101”；

哈夫曼树和码表都不需要更新，沿用之前的即可；

（步骤四）分析剩余串“10101”之后将读头 10，解码为字符 c，并输出，输入串读头前进两个位变为“101”；

然后更新哈夫曼树为：

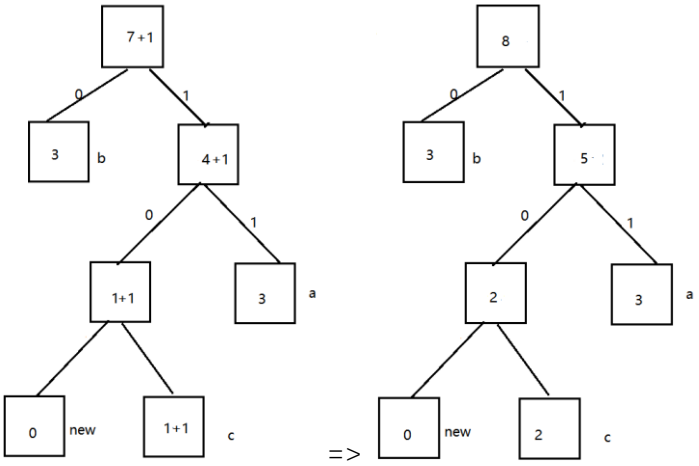


更新码表为：

New	a	b	c	d
00	11	0	101	11

（步骤五）分析剩余串“101”之后将读头 101，解码为字符 c，并输出，输入串读头前进三个位变为空串；

然后更新哈夫曼树为：



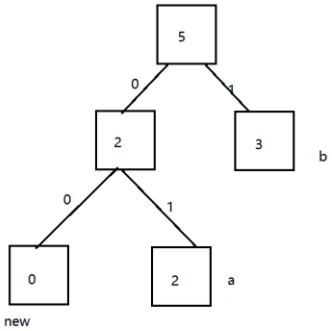
码表无需更新，因为树的结构没变；

至此，解码暂停等待后续到来的编码位流。

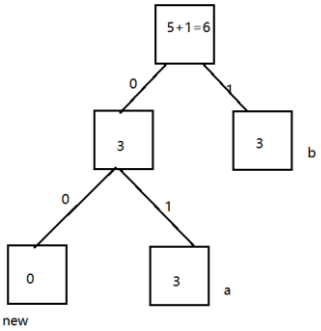
i． 通过上面的分析我们知道，后续接收到的几个字母为 b(01)、a(01)、c(00 10)、c(101)

ii．通过上面的分析，我们知道：

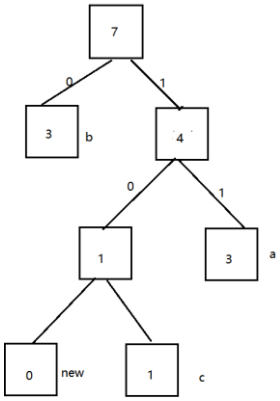
接收到字母 b 之后的哈夫曼树：



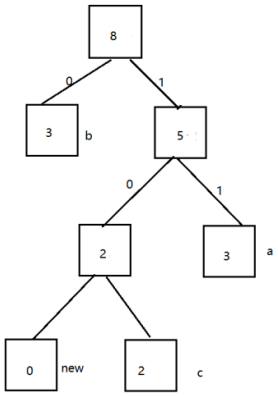
接收到字母 a 之后的哈夫曼树：



接收到前一个 c 之后的哈夫曼树:



接收到后一个 c 之后的哈夫曼树:



第二题

You are given a computer cartoon picture and a photograph. If you have a choice of using either JPEG compression or GIF, which compression would you apply for these two images? Justify your answer.

对于卡通图片，选择使用 GIF 压缩方法进行压缩会更好;
对于照片，选择使用 JPEG 压缩方法进行压缩会更好;

理论原因分析：
GIF 标准仅适用于 **8 位彩色图像**，因为这能够生成可接受的颜色，它适用于具有少量独特色彩的图像。GIF 图像采用 **LZW 压缩技术**，它的真实光栅数据本身在存储前使用自适应的、基于字典的压缩技术（LZW）进行压缩。
JPEG 图像则适用于 **24 位彩色图像**，JPEG 压缩方法**使用 DCT 变换进行编码**，(1) 利用图片内容空间变化频率缓慢的特点对图片进行压缩、(2) 利用人类对高频分量损失的感知能力低于低频分量损失的感知能力对图片高频分量进行大幅压缩、(3) 利用人类对彩色敏感度不如黑白压缩图片的彩色信息。

因此对于**卡通图像**，它仅具有少量种类的颜色，所以适合使用 GIF 技术进行压缩；并且由于颜色种类比较少，从而产生并加入字典的编码会很少，然后这些编码被反复利用，使得 LZW 压缩技术变得很高效；**使用 8 位表示颜色，再结合高效的 LZW 编码，使得大幅压缩图片存储空间的同时并不会使图片在视觉上产生很大偏差**，综合上面的原因，GIF 压缩方法很适合卡通图片。但是如果我们使用 JPEG 压缩方法压缩卡通图片，卡通图片将被当作复杂图片来对待，使用 24 位色彩来表示卡通图片，但是实际上卡通图片用到的色彩远远不到 255*255*255 种，但每一种颜色仍然用 3*8=24 位表示（本来可以用 8 位表示），所以大量的存储空间被浪费，这是不合理的。
对于**照片**，它具有较多的颜色种类，所以不适合用 GIF 压缩（该方法只使用 8 位彩色图像，将导致照片颜色信息的大量损失），而使用 JPEG 方法则非常高效——**JPEG 技术使用 DCT 变换进行编码，该编码方法充分利用前面说到的彩色图片三大特性进行压缩，对于彩色图片压缩非常高效，同时又使得图片的视觉感受差别不大（失真度较小）**，因此照片适合用 JPEG 方法进行压缩。

程序实现：

自己使用 matlab 语言进行编码，实现 JPEG 压缩（gif 压缩使用现有软件），完整程序代码分为了 JPEG.m、EncodeAndDecode.m、ACcode.m、DCcode.m、strm.m 五个文件，一并打包在作业中，其中

JPEG.m 是整个压缩并解码过程的主函数，输入图片名称然后进行压缩和解码，底下分为**颜色转换、二度采样、编码和解码、颜色反转换**等 8 个过程；

EncodeAndDecode.m 文件中是编码和解码函数，对应于 JPEG.m 的编码和解码模块，输入量化后的像素矩阵进行编码和解码并返回解码结果，其中分为 **DCT 变换、量化、系数编码和解码、反量化、反 DCT 变换**模块，DC 系数编码首先进行 DPCM 编码（预测函数是$\hat{f}_n = \tilde{f}_{n-1}$）然后进行熵编码，AC 系数编码首先进行游长编码然后进行熵编码，然后串联起来，解码的时候反过来，先做熵解码，然后 DC 系数做 DPCM 解码，AC 系数做游长编码解码；

DCcode.m 是编码函数，输入已经进行了 DPCM 编码的 DC 系数，输出熵编码后的 DC 系数，具体操作是先将 DC 系数做 DPCM 编码，然后将编码后 DC 系数分为 size 和 amplitude 两部分，其中 size 表示系数绝对值占的位数，amplitude 是系数绝对值的二进制码（系数绝对值是正数取原码，是负数则按位取反），然后对 size 进行哈夫曼编码，编码后串联整个 amplitude 部分（不编码）；

ACcode.m 是编码函数，输入已经进行了游长编码的 AC 系数，输出熵编码后的 AC 系数，具体操作是先对 AC 系数做游长编码，成为一系列（runLength，value）系数对，然后将编码后系数对分为 symbol1：（runLength，size）和 symbol2：（amplitude）两部分，其中 runLength 表示 AC 系数前面 0 的个数，size 是该非零 AC 系数绝对值的占位数，amplitude 是系数绝对值的二进制码（系数绝对值是正数取原码，是负数则按位取反），然后对 symbol1 进行哈夫曼编码，编码后串联整个 symbol2 部分（不编码）；由于 runLength 和 size 各占 4 位所以 0 串超过 15，需要用特殊扩展编码（15，0）表示，另外结尾的 0 串无论长度，均用（0，0）表示；

Strm.m 只是一个工具函数，用来比较两个字符串的大小；

Calculate.m 是一个计算失真度的函数，输入原图片和重现图片，打印失真度；

下面是模块内容的简要展示：

填充部分（长和宽填充为 8 的倍数，最后剪切回来）：

```
%% 读取图像并填充图像长/宽为8的倍数
img = imread(name);
[r0,c0,~] = size(img);
imgTemp = zeros(floor((r0-1)/8+1)*8,floor((c0-1)/8+1)*8,3);
imgTemp(1:r0,1:c0,1:3) = img(1:r0,1:c0,1:3);
img = imgTemp;
[r,c,color] = size(img);
```

RGB 转 YUV 部分：

```
%% YUV颜色转换
R = img(:,:,1);
G = img(:,:,2);
B = img(:,:,3);

Y = 0.299*R + 0.587*G + 0.114*B;
U = -0.147*R- 0.289*G + 0.436*B;
V = 0.615*R - 0.515*G - 0.100*B;
% YUV = cat(3, Y, U, V);
% figure; imshow(YUV);
```

色度二次采样部分（使用 4：2：0 方案）：

```
%% 色度二次采样，使用4：2：0方案：
sampleU = zeros(r/2, c/2);
sampleV = zeros(r/2, c/2);
for i = 1:r/2
    for j = 1:c/2
        sampleU(i,j) = U(i*2-1,j*2-1);
        sampleV(i,j) = V(i*2-1,j*2-1);
    end
end
```

```
%% 下面的操作分成三个分量分别做压缩和解码操作操作，然后返回解码后的分量
Y = EncodeAndDecode(Y,1);
sampleU = EncodeAndDecode(sampleU,2);
sampleV = EncodeAndDecode(sampleV,2);
```

具体压缩和解码过程：（包括 DCT 变换、量化、系数编码和解码、反量化、反 DCT 变换）

分块（8*8 分块）过程：

```
%% 对图像进行分块
[r0,c0] = size(imgBefore);
tempImgBefore = zeros(floor((r0-1)/8+1)*8, floor((c0-1)/8+1)*8);
tempImgBefore(1:r0,1:c0) = imgBefore(1:r0,1:c0);
imgBefore = tempImgBefore;
[r,c] = size(imgBefore);
extendR = r/8;
extendC = c/8;
Blocks = zeros(extendR*extendC*8, 8);
for i = 1:extendR
    for j = 1:extendC
        Blocks((i-1)*extendC*8+(j-1)*8+1:(i-1)*extendC*8+j*8,:) = imgBefore((i-1)*8+1:(i-1)*8+8, (j-1)*8+1:(j-1)*8+8);
    end
end
```

DCT 变化过程：

```
%% 对图像进行DCT变换
for i = 1:extendR*extendC
    Blocks((i-1)*8+1:(i-1)*8+8,:) = dct2(Blocks((i-1)*8+1:(i-1)*8+8,:));
end
```

量化过程：（下面的 table1 是亮度量化表，table2 是色度量化表）

```
if (type == 1)
    for i = 1:extendR*extendC
        Blocks((i-1)*8+1:(i-1)*8+8,:) = round(Blocks((i-1)*8+1:(i-1)*8+8,:)./table1);
    end
else
    for i = 1:extendR*extendC
        Blocks((i-1)*8+1:(i-1)*8+8,:) = round(Blocks((i-1)*8+1:(i-1)*8+8,:)./table2);
    end
end
```

系数编码和解码过程过于复杂，所以只展示部分代码：

系数编码：

```
% 进行z编序
Zcode = zeros(extendR*extendC, 64);
order = [
    1 9 2 3 10 17 25 18 11 4 5  12 19 26 33 41 34 27 20 1
];
for i = 1:extendR*extendC
    block = reshape(Blocks((i-1)*8+1:(i-1)*8+8,:),1,64);
    for j = 1:64
        Zcode(i,j) = block(order(j));
    end
end
```

```
% DC系数编码，先做DPCM编码（这个能够一次得
DCcoes = zeros(1,extendR*extendC);
DCcoes(1) = Zcode(1,1);
for i = 2:extendR*extendC
    DCcoes(i) = Zcode(i,1) - Zcode(i-1,1);
end
```

AC 系数游长编码:

```
%% 游长编码
RLC = zeros(1,8); j = 1; len = 0;
for i = 1:63
    if (ACcoes(i) == 0 && len < 15)
        len = len+1;
    else
        RLC(j) = len;
        RLC(j+1) = ACcoes(i);
        j = j+2;
        len = 0;
    end
end
if (ACcoes(63) == 0)
    RLC(j) = 0;
    RLC(j+1) = 0;
end

while (length(RLC) > 2)
    if (RLC(length(RLC)) == 0 && RLC(length(RLC)-2) == 0)
        RLC = RLC(1:length(RLC)-2);
        RLC(length(RLC)-1) = 0;
    else
        break;
    end
end
```

```
% DCcode函数：DC系数转成中间形式（size,amplitude）序列，然后对size做哈夫曼编码
% ACcode函数：AC系数先做游长编码，再转为symbol1（runlength,size）和symbol2(amplitude)，然后对symbol1做哈夫曼编码
jpegCode = [];
for i = 1:extendR*extendC
    jpegCode = [jpegCode DCcode(DCcoes(i),type) ACcode(Zcode(i,2:64),type)];
end
```

关于具体的 DCcode、ACcode 对 DPCM 编码后 DC 系数和游长编码后的 AC 系数编码过程见 DCcode.m、ACcode.m； 下面给出一小部分代码:

```
%% DCcode函数：DC系数转成中间形式（size,amplitude）序列，然后对size做哈夫曼编码
function code = DCcode(DCcoe, type)
% 亮度DC系数的哈夫曼编码表
T1 = {'00' '010' '011' '100' '101' '110' '1110' '11110' '111110' '1111110' '11111110' '111111110'};
% 色度DC系数的哈夫曼编码表
T2 = {'00' '01' '10' '110' '1110' '11110' '111110' '1111110' '11111110' '111111110' '1111111110' '11111111110'};
DCcoeAmplitude = abs(DCcoe);
DCcoeSize = 0;
while (DCcoeAmplitude/(2^DCcoeSize)>=1)
    DCcoeSize= DCcoeSize+1;
end
if (type == 1)
    S1 = char(T1(DCcoeSize+1));
else
    S1 = char(T2(DCcoeSize+1));
end

if (DCcoe == 0)
```

(节选自 DCcode.m)

```
%% 转化symbol1和symbol2:
code = '';
for i = 1:2:length(RLC)
    ACrunlength = RLC(i);
    ACcoeAmplitude = abs(RLC(i+1));
    ACcoeSize = 0;
    while (ACcoeAmplitude/(2^ACcoeSize) >= 1)
        ACcoeSize = ACcoeSize+1;
    end
    % 对Symbol1做哈夫曼编码
    if (type == 1)
        if (ACrunlength == 0 || ACrunlength == 15)
            S1 = char(T1(ACrunlength+1, ACcoeSize+1));
        else
            S1 = char(T1(ACrunlength+1, ACcoeSize));
        end
    else
        if (ACrunlength == 0 || ACrunlength == 15)
            S1 = char(T2(ACrunlength+1, ACcoeSize+1));
        else
            S1 = char(T2(ACrunlength+1, ACcoeSize));
```

（节选自 ACcode.m）

解码过程：（详情见代码，这里只给出一点点，因为太长了）

DC 系数解码：

```
% decode函数：将哈夫曼编码转化为矩阵信息
index = 1;
if (type == 1)
    for i = 1:extendR*extendC
        %首先匹配DC系数
        for j = 1:12
            DCcoeSizeDecode = char(T10(j));
            if (strcmpi(DCcoeSizeDecode,jpegCode(index:index+length(DCcoeSizeDecode)-1)))
                index = index+length(DCcoeSizeDecode);
                if (j==1)
                    if (i == 1)
                        ZcodeAfter(i, 1) = 0;%求出DC系数为0
                    else
                        ZcodeAfter(i, 1) = ZcodeAfter(i-1, 1);
```

AC 系数解码

```
%然后匹配AC系数
num = 2;
while (num <= 64)
    if (length(jpegCode) < index+15)
        ACcodeSymbol1 = jpegCode(index:length(jpegCode));
    else
        ACcodeSymbol1 = jpegCode(index:index+15);
    end
    start = 15;
    stop = 176;
    mid = floor((start+stop)/2);
    while (start <= stop)
        if (length(char(T11(mid))) > length(ACcodeSymbol1))
            mycmp = char(T11(mid));
            mycmp = mycmp(1:length(ACcodeSymbol1));
```

DC 系数和 AC 系数解码后按照 Z 编序还原：

```
% zig-zag还原
BlocksAfter = zeros(extendR*extendC*8, 8);
for i = 1:extendR*extendC
    blockAfter = zeros(1,64);
    for j = 1:64
        blockAfter(order(j)) = ZcodeAfter(i,j);
    end
    BlocksAfter((i-1)*8+1:(i-1)*8++8,:) = reshape(blockAfter,8,8);
end
Blocks = BlocksAfter;
```

反量化过程：

```
Blocks = BlocksAfter;

%% 对图像进行反量化
if (type == 1)
    for i = 1:extendR*extendC
        Blocks((i-1)*8+1:(i-1)*8+8,:) = Blocks((i-1)*8+1:(i-1)*8+8,:).*table1;
    end
else
    for i = 1:extendR*extendC
        Blocks((i-1)*8+1:(i-1)*8+8,:) = Blocks((i-1)*8+1:(i-1)*8+8,:).*table2;
    end
end
```

反 DCT 变化过程

```
%% 对图像进行逆DCT变换
for i = 1:extendR*extendC
    Blocks((i-1)*8+1:(i-1)*8+8,:) = idct2(Blocks((i-1)*8+1:(i-1)*8+8,:));
end
```

分块合并：

```
%% 对图像进行块合并
imgAfterDecode = zeros(r,c);
for i = 1:extendR
    for j = 1:extendC
        imgAfterDecode((i-1)*8+1:(i-1)*8+8, (j-1)*8+1:(j-1)*8+8) = Blocks((i-1)*extendC*8+(j-1)*8+1:(i-1)*extendC*8+j*8,:);
    end
end
imgAfterDecode = imgAfterDecode(1:r0,1:c0);
```

YUV 转 RGB 部分

```
%% 解码后YUV颜色转RGB
for i = 1:r
    for j = 1:c
        U(i,j) = sampleU(floor((i-1)/2+1),floor((j-1)/2+1));
        V(i,j) = sampleV(floor((i-1)/2+1),floor((j-1)/2+1));
    end
end
reConStr = zeros(r,c,color);
reConStr(:,:,1) = Y + 1.14 * V;
reConStr(:,:,2) = Y - 0.39 * U - 0.58 * V;
reConStr(:,:,3) = Y + 2.03 * U;
reConStr = uint8(reConStr);
figure; imshow(reConStr(1:r0,1:c0,1:3));
%imwrite(reConStr(1:r0,1:c0,1:3), 'task2_2.jpg');
```

上述是大部分代码内容，详情见各个代码文件

结果对比：（先给图，再给压缩率、失真度计算）

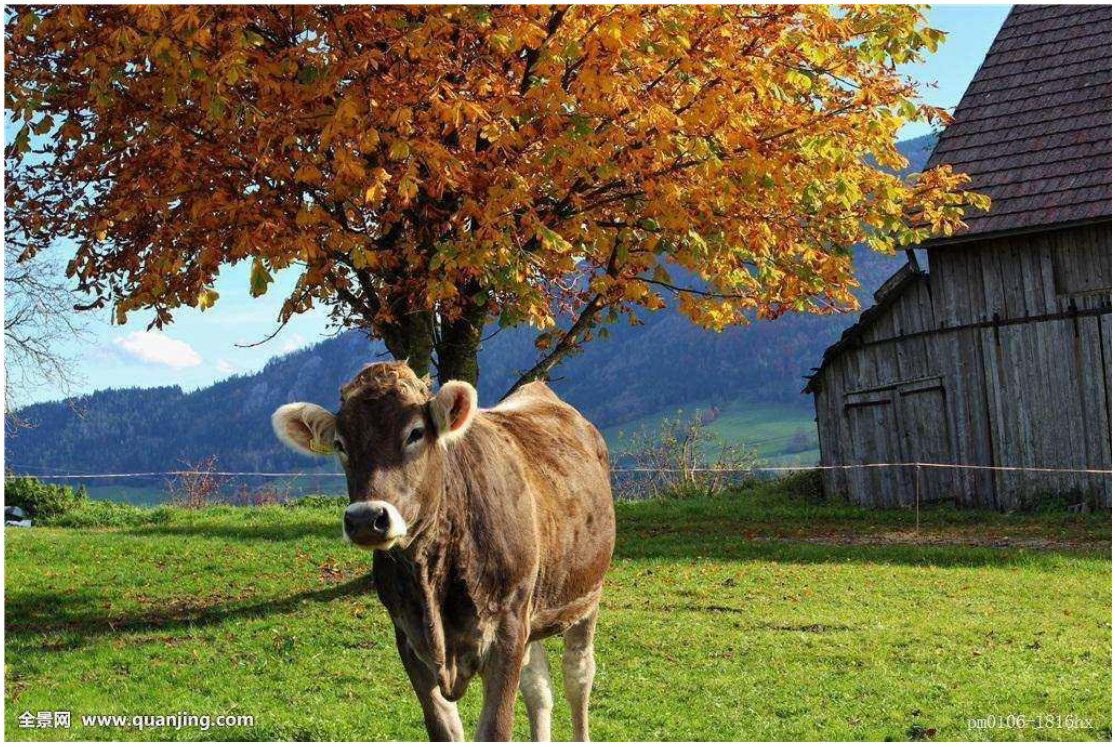
动物卡通图片.jpg 原图：

JPEG 压缩后：

Gif 压缩后:

动物照片.jpg 的原图:

JPEG 压缩后：



Gif 压缩后：

对于压缩率计算，采用公式$\frac{B_0}{B_1}$，此处B_0是压缩前图片的大小，B_1是压缩后图像数据大小（并不是解码后图片大小，而是压缩后的数据大小），GIF 压缩率由压缩软件指定，JPEG 计算压缩率代码如下：

```
img = imread(name);
[r0,c0,~] = size(img);
disp("原图像大小按照bit计算为：");
```

```
% DCcode函数：DC系数转成中间形式（size,amplitude）序列，然后对size做哈夫曼编码
% ACcode函数：AC系数先做游长编码，再转为symbol1（runlength,size）和symbol2(amp
jpegCode = [];
for i = 1:extendR*extendC
    jpegCode = [jpegCode DCcode(DCcoes(i),type) ACcode(Zcode(i,2:64),type)];
end
disp("该通道压缩率后bit数为：");
disp(length(jpegCode));
```

所以 JPEG 图像压缩方法的压缩率通过程序计算结果为：

对动物卡通图片.jpg 的压缩率：

```
>> JPEG('动物卡通图片.jpg')
原图像大小按照bit计算为：
    17160000

该通道压缩率后bit数为：
      669307

该通道压缩率后bit数为：
       46738

该通道压缩率后bit数为：
       75845

>> disp("压缩比计算为：")
压缩比计算为：
>> disp(17160000/(669307+46738+75845))
   21.6697
```

对动物照片.jpg 的压缩率：

```
>> JPEG('动物照片.jpg')
原图像大小按照bit计算为：
    16760832

该通道压缩率后bit数为：
     1264526

该通道压缩率后bit数为：
       62728

该通道压缩率后bit数为：
       59309

>> disp("压缩比计算为：")
压缩比计算为：
>> disp(16760832/(1264526+62728+59309))
   12.0880
```

PNG 图像压缩方法的压缩率计算如下，为

对动物卡通图片.jpg 的压缩率：

对动物照片.jpg 的压缩率：

上面指出动物卡通图片.gif 大小 256kb，动物照片.gif 大小 532kb

而原图动物卡通图片.jpg、动物照片.jpg 的大小在前面已经计算过了分别为 17160000、16760832，于是作如下计算：

```
>> disp("动物卡通图片压缩比计算为：")
动物卡通图片压缩比计算为：
>> disp(17160000/262979/8)
    8.1565
```

```
>> disp("动物照片压缩比计算为：")
动物照片压缩比计算为：
>> disp(16760832/544931/8)
    3.8447
```

对于失真度，采用峰值信噪比进行度量，即 PSNR=$10\,log_{10}\frac{x_{peak}^2}{\sigma_d^2}$，

其中x_{peak}是峰值信号，也就是图片像素点的颜色最大值，对于此题给出的图片，它们是 24 位彩色图像，对于每个颜色通道（r/g/b 通道）使用 0-255 的字节值表示亮度，所以x_{peak}的值为 255，

σ_d^2为输入数据序列和重现数据序列的均方差，这里的输入数据是原始图片（全部像素点），输出数据是压缩并解码后的图片，对于灰度图片，$\sigma_d^2=\frac{1}{mn}\sum_{i=1}^{m}\sum_{j=1}^{n}(s_1(i,j)-s_0(i,j))^2$，

其中 m、n 为图片长、宽，$s_0(i,j)$表示原图片在(i,j)这一像素点处的亮度值，$s_1(i,j)$表示压缩并解码后图片在(i,j)像素点处的亮度值，对于此题给出的图片，它们是彩色图像，所以均方差有两种计算方法，第一种是将彩色图转为灰度图再使用灰度图均方差计算方法，第二种是计算 R、G、B 各分量的均方差然后去平均值，这里我采用第一种计算方法，计算代码在 calculate.m 中：

```
%% 采用峰值信噪比进行度量，即PSNR=20 〖log〗_10 〖(x_peak)/(σ_d )〗
function calculate(picture1,picture2)
img1 = imread(picture1);
img2 = imread(picture2);
[~,~,c] = size(img2);
if (c == 1)
    %说明是gif
    [img2,map] = imread(picture2);
    img2 = ind2rgb(img2,map);
end
%x_peak采用256
x_peak = 256.0;
%输入数据和重现数据序列的方差计算：
%首先做灰度图片转换
ray1 = rgb2gray(img1);
ray2 = rgb2gray(img2);
[r,c] = size(ray1);
%计算均方差
myVariance = 0;
for i = 1:r
    for j = 1:c
        myVariance = myVariance + double((ray1(i,j)-ray2(i,j))^2);
    end
end
StandardDeviation = myVariance/(r*c);
disp("失真度为：");
disp(20*log(x_peak/StandardDeviation)/log(10));
```

然后通过程序计算得出 JPEG 图像压缩方法的 PSNR 为：

动物卡通图片.jpg 压缩后失真度：

```
>> calculate('动物卡通图片.jpg','task2_2.jpg')
失真度为：
   27.5427
```

动物照片.jpg 压缩后失真度：

```
>> calculate('动物照片.jpg','task2_1.jpg')
失真度为：
   26.8241
```

GIF 图像压缩方法的 PSNR 为

动物卡通图片.jpg 压缩后失真度：

```
>> calculate('动物卡通图片.jpg','动物卡通图片.gif')
失真度为：
    0.0480
```

动物照片.jpg 压缩后失真度：

```
>> calculate('动物照片.jpg','动物照片.gif')
失真度为：
    0.1839
```

计算结果 结合 图片效果分析如下：
在卡通图片的对比中（JPEG 压缩、GIF 压缩、原图），gif 压缩的两种压缩方法带来的视觉损失都不大（但是看起来好像 JPEG 压缩过后图片有些模糊，所以 gif 压缩后视觉效果可能好一些），但是显然 GIF 压缩方法压缩效果更好（拥有更大的压缩比）；
在照片的对比中（JPEG 压缩、GIF 压缩、原图），gif 压缩方法带来严重的视觉效果失真（树林覆盖的山区、天空部分都有很明显的失真）。而 JPEG 压缩方法得到的图片视觉效果则更好，虽然理所当然需要更大的空间（压缩比更小），压缩效果差一些；
JPEG 压缩能够带来更好视觉效果（PSNR 更大，失真越少，因此视觉效果更好），但是同时付出代价更高，即压缩比不如 GIF 压缩，所以很适用于照片压缩（因为照片相比卡通图片更需要避免视觉失真），对于卡通图片压缩就有点浪费存储空间了；
GIF 压缩能够带来更好的压缩比（特别对于卡通图片），但是视觉效果损失更大，在上面的卡通图片经过 gif 压缩之后视觉效果损失不是特别明显，但是对于照片，在颜色变化平缓的地方表现出明显的失真，所以 GIF 更适合卡通图片的压缩，对于照片则不适合（虽然压缩率很大，但是视觉效果损失太严重）；
综上所述，实验结果和理论分析基本一致，实验成功。