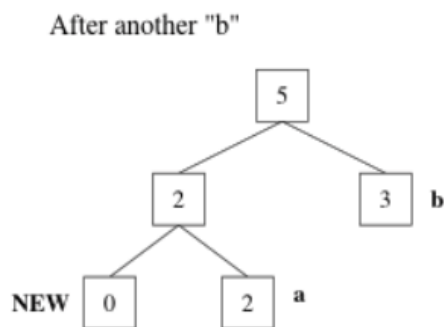


第一题

(a) Like any other adaptive compression algorithms, it is more dynamic, therefore offers better compression and works even when prior statistics of the data distribution is unavailable as it is in most multimedia applications. It also saves the overhead since no symbol table needs to be transmitted.

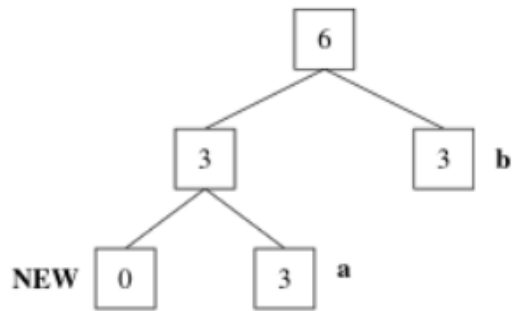
(b) (i) The additional letters received are "b(01) a(01) c(0010) c(101)". 01 matches b, we change the tree as (ii) describes. Then 01 matches a. The tree changes, but the coding don't change. Then 001 doesn't match any word, so there must be a new word, we insert c, then 0010 matches c. Then change the tree. The following 101 matches c.

(ii) The trees are as below.



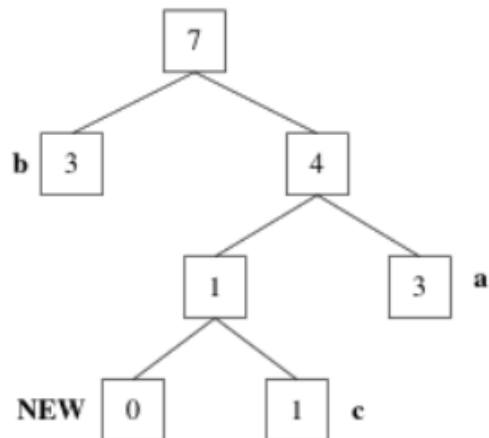
b is shown before, add b don't need to change the tree. b++

After another "a"



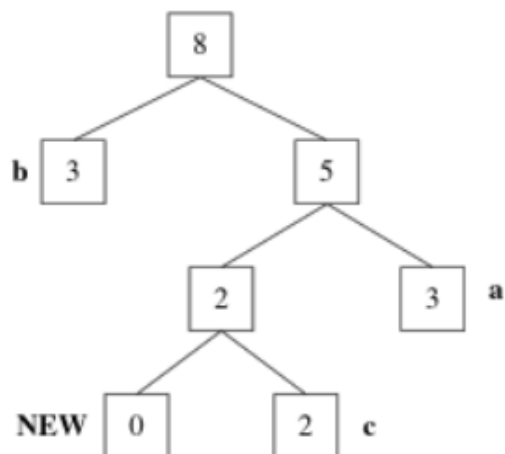
a is shown before, add a don't need to change the tree. a++

After "c"



c is new, add it to the child node of new, and using new subtree replace original node. The brother node of b' s number is 4, bigger than b, so change their place.

After another "c"



add c' s number and his father node' s number, nothing need change.

第二题

理论分析:

GIF will generally be better, compared to JPEG, because JPEG firstly keeps 24-bit color, which is not needed for a cartoon, due to it has only a few numbers. But the LZW compression will do a very good job in terms of compression ratio, since only a few values exist and their runs will build up in the dictionary. On the other hand, JPEG will treat the image as a complex image no matter if it is or not.

编程实现:

编码主文件: encode.m

```
% 颜色转换
R=double(src(:,:,1));
G=double(src(:,:,2));
B=double(src(:,:,3));
Y=0.299*R+0.5870*G+0.114*B;
Cb=-0.1687*R-0.3313*G+0.5*B+128;
Cr=0.5*R-0.4187*G-0.0813*B+128;
```

```

% DCT变换
N=8;
T=zeros(N);
for i=0:N-1
    if i==0
        c=sqrt(1/N);
    else
        c=sqrt(2/N);
    end
    for j=0:N-1
        T(i+1,j+1)=c*cos(pi*(j+0.5)*i/N);
    end
end
fun=@(block_struct) T*block_struct.data*T';
Y1=blockproc(Y,[8 8],fun,'PadPartialBlocks',true);
Cb1=blockproc(Cb,[8 8],fun,'PadPartialBlocks',true);
Cr1=blockproc(Cr,[8 8],fun,'PadPartialBlocks',true);

% 量化,根据标准量化表
fun1=@(block_struct) round(block_struct.data./a);
fun2=@(block_struct) round(block_struct.data./b);
Y2=blockproc(Y1,[8 8],fun1,'PadPartialBlocks',true);
Cb2=blockproc(Cb1,[8 8],fun2,'PadPartialBlocks',true);
Cr2=blockproc(Cr1,[8 8],fun2,'PadPartialBlocks',true);

```

计算 DPCM, 在 dpcm.m 中

```

[m,nx]=size(x);
if min(m,nx) > 1, error('x must be a vector'); end
if m==1 && nx > 1, x = x'; m=nx; end % ensure x is a column vector
[p,na]=size(a);
if p==1 && na > 1, a = a'; p=na; end % ensure a is a col. vector

% First p element of x must be sent unchanged as part of r after
% scalar quantization

r=round(x(1:p)); xtilde=r;
for t=p+1:m
    xhat(t)=a'*xtilde(t-1:-1:t-p);
    r(t)=round(x(t)-xhat(t));
    xtilde(t)=xhat(t)+r(t);
end

```

在 jdcenc.m 中对 DC 系数进行哈夫曼编码

对 AC 系数进行 Zig-Zag 扫描, 尽可能把 0 放在一起, 并标记最后的 0 块

```

acseq=[];
for i=1:mb
    for j=1:nb
        tmp(z)=x(8*(i-1)+1:8*i,8*(j-1)+1:8*j);
        % tmp is 1 by 64
        eobi=find(tmp~=0, 1, 'last' ); %end of block index
        % eob is labelled with 999
        acseq=[acseq tmp(2:eobi) 999];
    end
end
accf=jacenc(acseq);

```

在 jacenc.m 中对处理后的 AC 系数进行哈夫曼编码

得到最终的编码

```

% huffman for dc and ac coefficients
[dccof1, accof1]=huffman(Y2,mf,nf);
[dccof2, accof2]=huffman(Cb2,mf,nf);
[dccof3, accof3]=huffman(Cr2,mf,nf);

```

最后计算压缩率和像素深度

```

dccofLength=length(dccof1)+length(dccof2)+length(dccof3);
accofLength=length(accof1)+length(accof2)+length(accof3);

disp(['DC coefficient after Huffman coding has ' int2str(dccofLength) ' bits']);
disp(['AC coefficient after Huffman coding has ' int2str(accofLength) ' bits']);
OB=mf*nf*3*8;
CB=dccofLength+accofLength;
disp(['Original Bit: ' num2str(OB) ' bits']);
disp(['Compressed Bit: ' num2str(CB) ' bits']);
disp(['Compression Ratio ' num2str(OB/CB) ' : 1']);
disp(['Pixel Depth: ' num2str(CB/mf/nf) ' bits / pixel ']);

```

解码主文件: decode.m, 注意需要先运行 encode 获得编码

在 jacdec.m 和 jdcdec.m 中对编码后的 AC 系数和 DC 系数进行解码,在 reconstruct.m 中

还原为矩阵

```

acarr=jacdec(accof);
dcarr=jdcdec(dccof);

mb=mf/8; nb=nf/8; % Number of blocks

Eob=find(acarr==999);
kk=1;ind1=1;n=1;
for ii=1:mb
    for jj=1:nb
        ac=acarr(ind1:Eob(n)-1);
        ind1=Eob(n)+1;
        n=n+1;
        ri(8*(ii-1)+1:8*ii,8*(jj-1)+1:8*jj)=dezz([dcarr(kk) ac zeros(1,63-length(ac))]);
        kk=kk+1;
    end
end

```

还原量化和反 dct 变换

```

iFq=round(blkproc(ri,[8 8],'x.*P1',Q));
iFf=blkproc(iFq,[8 8],'idct2');

```

转换颜色空间并归一化到 0-255

```

[m, n,~]=size(src);
Y3=reconstruct(accof1,dccof1,mf,nf,a);
Cb3=reconstruct(accof2,dccof2,mf,nf,b);
Cr3=reconstruct(accof3,dccof3,mf,nf,b);

R1=Y3+1.402*(Cr3-128);
G1=Y3-0.34414*(Cb3-128)-0.71414*(Cr3-128);
B1=Y3+1.772*(Cb3-128);

img(:,:,1)=mat2gray(R1(1:m,1:n),[0 255]);
img(:,:,2)=mat2gray(G1(1:m,1:n),[0 255]);
img(:,:,3)=mat2gray(B1(1:m,1:n),[0 255]);

```

计算均方差 (MSE) 和峰值信噪比 (PSNR), 用于评价压缩后的图像和原图像的相似程度

```

% Calculate MSE , PSNR
MSE=mean(mean((im2uint8(img)-src).^2))
PSNR=10*log10(255^2/MSE)

```

实验结果:

```
>> encode
```

```
DC coefficient after Huffman coding has 143060 bits
```

```
AC coefficient after Huffman coding has 858661 bits
```

```
Original Bit: 17280000 bits
```

```
Compressed Bit: 1001721 bits
```

```
Compression Ratio 17.2503 : 1
```

```
Pixel Depth: 1.3913 bits / pixel
```

```
MSE(:, :, 1) =
```

```
11.6547
```

```
MSE(:, :, 2) =
```

```
10.1816
```

```
MSE(:, :, 3) =
```

```
13.9176
```

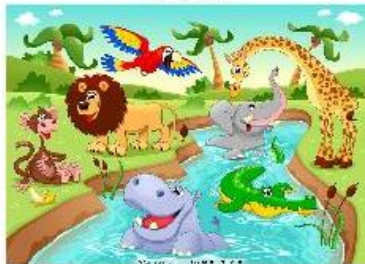
```
PSNR(:, :, 1) =
```

```
37.4658
```

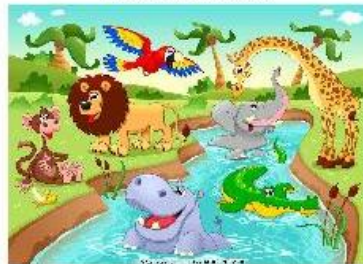
```
PSNR(:, :, 2) =
```

```
38.0526
```

Original



Reconstructed



```
>> encode
```

```
DC coefficient after Huffman coding has 154586 bits
```

```
AC coefficient after Huffman coding has 1459302 bits
```

```
Original Bit: 16908288 bits
```

```
Compressed Bit: 1613888 bits
```

```
Compression Ratio 10.4767 : 1
```

```
Pixel Depth: 2.2908 bits / pixel
```







```
>> decode
MSE(:, :, 1) =
    13.2265

MSE(:, :, 2) =
    12.2128

MSE(:, :, 3) =
    20.8959

PSNR(:, :, 1) =
    36.9163
```

 animal	2018/11/29 16:15	GIF 文件	601 KB
 animal	2018/11/1 13:37	JPG 文件	173 KB
 carton	2018/11/29 16:18	GIF 文件	408 KB
 carton	2018/11/1 13:35	JPG 文件	114 KB

可以看到 jpg 和 gif 压缩的视觉效果差不多，肉眼看不出区别，但是 jpg 的大小小了很多。