```
%% LOSSY COMPRESSION-DECOMPRESSION USNIG DISCRETE COSINE TRANSFORM TECHNIQUE.
function[]=dct1(filename,n,m)
% "filename" is the string of characters including Image name and its
% extension.
% "n" denotes the number of bits per pixel.
% "m" denotes the number of most significant bits (MSB) of DCT Coefficients.

% Matrix Intializations.
N=8;                        % Block size for which DCT is Computed.
M=8;
I=imread(filename);         % Reading the input image file and storing intensity values in 2-D matrix I.
I_dim=size(I);              % Finding the dimensions of the image file.
I_Trsfrm.block=zeros(N,M);  % Initialising the DCT Coefficients Structure Matrix "I_Trsfrm" with the↙
required dimensions.

Norm_Mat=[16 11 10 16 24 40 51 61       % Normalization matrix (8 X 8) used to Normalize the DCT Matrix.
          12 12 14 19 26 58 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];

save('LenaInitial.txt','I');

%% PART-1: COMPRESSION TECHNIQUE.

% Computing the Quantized & Normalized Discrete Cosine Transform.
```

```
% Y(k,l)=(2/root(NM))*c(k)*c(l)*sigma(i=0:N-1)sigma(j=0:M-1)y(i,j)cos(pi(2i+1)k/(2N))cos(pi(2j+1)l/(2M))
% where c(u)=1/root(2) if u=0
%            = 1       if u>0

for a=1:I_dim(1)/N
    for b=1:I_dim(2)/M
        for k=1:N
            for l=1:M
                prod=0;
                for i=1:N
                    for j=1:M
                        prod=prod+double(I(N*(a-1)+i,M*(b-1)+j))*cos(pi*(k-1)*(2*i-1)/(2*N))*cos(pi*(l-1)↙
*(2*j-1)/(2*M));
                    end
                end
                if k==1
                    prod=prod*sqrt(1/N);
                else
                    prod=prod*sqrt(2/N);
                end
                if l==1
                    prod=prod*sqrt(1/M);
                else
                    prod=prod*sqrt(2/M);
                end
                I_Trsfrm(a,b).block(k,l)=prod;
            end
        end
        % Normalizing the DCT Matrix and Quantizing the resulting values.
```

```
        I_Trsfrm(a,b).block=round(I_Trsfrm(a,b).block./Norm_Mat);
    end
end

% zig-zag coding of the each 8 X 8 Block.
for a=1:I_dim(1)/N
    for b=1:I_dim(2)/M
        I_zigzag(a,b).block=zeros(1,0);
        freq_sum=2:(N+M);
        counter=1;
        for i=1:length(freq_sum)
            if i<=((length(freq_sum)+1)/2)
                if rem(i,2)~=0
                    x_indices=counter:freq_sum(i)-counter;
                else
                    x_indices=freq_sum(i)-counter:-1:counter;
                end
                    index_len=length(x_indices);
                    y_indices=x_indices(index_len:-1:1); % Creating reverse of the array as "y_indices".
                    for p=1:index_len
                        if I_Trsfrm(a,b).block(x_indices(p),y_indices(p))<0
                            bin_eq=dec2bin(bitxor(2^n-1,abs(I_Trsfrm(a,b).block(x_indices(p),y_indices↙
(p)))),n);
                        else
                            bin_eq=dec2bin(I_Trsfrm(a,b).block(x_indices(p),y_indices(p)),n);
                        end
                        I_zigzag(a,b).block=[I_zigzag(a,b).block,bin_eq(1:m)];
                    end
            else
```

```
                counter=counter+1;
                if rem(i,2)~=0
                    x_indices=counter:freq_sum(i)-counter;
                else
                    x_indices=freq_sum(i)-counter:-1:counter;
                end
                    index_len=length(x_indices);
                    y_indices=x_indices(index_len:-1:1); % Creating reverse of the array as "y_indices".
                    for p=1:index_len
                        if I_Trsfrm(a,b).block(x_indices(p),y_indices(p))<0
                            bin_eq=dec2bin(bitxor(2^n-1,abs(I_Trsfrm(a,b).block(x_indices(p),y_indices↙
(p)))),n);
                        else
                            bin_eq=dec2bin(I_Trsfrm(a,b).block(x_indices(p),y_indices(p)),n);
                        end
                        I_zigzag(a,b).block=[I_zigzag(a,b).block,bin_eq(1:m)];
                    end
            end
        end
    end
end

% Clearing unused variables from Memory space
clear I_Trsfrm prod;
clear x_indices y_indices counter;

% Run-Length Encoding the resulting code.
for a=1:I_dim(1)/N
    for b=1:I_dim(2)/M
```

```
% Computing the Count values for the corresponding symbols and
% savin them in "I_run" structure.
count=0;
run=zeros(1,0);
sym=I_zigzag(a,b).block(1);
j=1;
block_len=length(I_zigzag(a,b).block);
for i=1:block_len
    if I_zigzag(a,b).block(i)==sym
        count=count+1;
    else
        run.count(j)=count;
        run.sym(j)=sym;
        j=j+1;
        sym=I_zigzag(a,b).block(i);
        count=1;
    end
    if i==block_len
        run.count(j)=count;
        run.sym(j)=sym;
    end
end

% Computing the codelength needed for the count values.
dim=length(run.count);  % calculates number of symbols being encoded.
maxvalue=max(run.count);  % finds the maximum count value in the count array of run structure.
codelength=log2(maxvalue)+1;
codelength=floor(codelength);
```

```
        % Encoding the count values along with their symbols.
        I_runcode(a,b).code=zeros(1,0);
        for i=1:dim
            I_runcode(a,b).code=[I_runcode(a,b).code,dec2bin(run.count(i),codelength),run.sym(i)];
        end
    end
end
% Saving the Compressed Code to Disk.
save ('LenaCompressed.txt','I_runcode');

% Clearing unused variables from Memory Space.
clear I_zigzag run;

%% PART-2: DECOMPRESSION TECHNIQUE.

% Run-Length Decoding of the compressed image.
for a=1:I_dim(1)/N
    for b=1:I_dim(2)/M
        enc_str=I_runcode(a,b).code;

        % Computing the length of the encoded string.
        enc_len=length(enc_str);

        % Since Max. Count is unknown at the receiver, Number of bits used for each
        % count value is unknown and hence cannot be decoded directly. Number of bits
        % used for each count can be found out by trial and error method for all
        % the possible lengths => factors of encoded string length.
```

```
        % Computing the non-trivial factors of the "enc_len" (length of encoded
        % string) i.e., factors other than 1 & itself.
        factors_mat=zeros(1,0);
        if enc_len<=(n+1)
            realfact=enc_len;
        else
            for i=2:enc_len-2       % "enc_len-1" is always not a divisor of "enc_len".
                if(rem(enc_len,i)==0)
                    factors_mat=[factors_mat,i];
                end
            end

            % Trial and Error Method to Find the Exact count value.
            for i=1:length(factors_mat)
                flagcntr=0;
                temp_dim=enc_len/factors_mat(i);
                for j=1:temp_dim
                    if strcmp(enc_str(1+(j-1)*factors_mat(i):j*factors_mat(i)),dec2bin(0,factors_mat(i)))↙
==0
                        if j==1
                            flagcntr=flagcntr+1;
                        else
                            if enc_str((j-1)*factors_mat(i))~=enc_str(j*factors_mat(i))
                                flagcntr=flagcntr+1;
                            else
                                break;
                            end
                        end
                    else
```

```
                    break;
                end
            end
            if flagcntr==temp_dim
                realfact=factors_mat(i);
                break;
            end
        end
    end

    % Clearing unused variables from Memory space
    clear factors_mat flagcntr j

    % Finding out the count values of corresponding symbols in the encoded
    % string and then decoding it accordingly.
    dec_str=zeros(1,0);
    temp_dim=enc_len/realfact;
    for i=1:temp_dim
        count_str=enc_str(1+(i-1)*realfact:(i*realfact)-1);
        countval=bin2dec(count_str);
        for j=1:countval
            dec_str=[dec_str,enc_str(i*realfact)];
        end
    end
    I_runcode(a,b).code=dec_str;
  end
end

% Clearing unused variables from Memory space
```

```
clear enc_str dec_str temp_dim realfact enc_len
clear countval count_str

% Reconstructing the 8 X 8 blocks in Zig-Zag fashion.
I_rec_Trnsfm.block=zeros(N,M);
for a=1:I_dim(1)/N
    for b=1:I_dim(2)/M
        bpp=length(I_runcode(a,b).code)/(N*M);  % "bpp" is the bits-per-pixel in reconstruction of image.
        bpp_diff=n-bpp;
        freq_sum=2:(N+M);
        counter=1;
        c_indx=1;
        for i=1:length(freq_sum)
            if i<=((length(freq_sum)+1)/2)
                if rem(i,2)~=0
                    x_indices=counter:freq_sum(i)-counter;
                else
                    x_indices=freq_sum(i)-counter:-1:counter;
                end
                    index_len=length(x_indices);
                    y_indices=x_indices(index_len:-1:1); % Creating reverse of the array as "y_indices".
                    for p=1:index_len
                        decm_eq=bin2dec([I_runcode(a,b).code(1+m*(c_indx-1):m*c_indx),dec2bin(0,↙
bpp_diff)]);
                        if decm_eq>(2^(n-1))-1
                            decm_eq=decm_eq-(2^n-1);
                        end
                        I_rec_Trnsfm(a,b).block(x_indices(p),y_indices(p))=decm_eq;
                       c_indx=c_indx+1;
```

```
                    end
            else
                counter=counter+1;
                if rem(i,2)~=0
                    x_indices=counter:freq_sum(i)-counter;
                else
                    x_indices=freq_sum(i)-counter:-1:counter;
                end
                    index_len=length(x_indices);
                    y_indices=x_indices(index_len:-1:1); % Creating reverse of the array as "y_indices".
                    for p=1:index_len
                        decm_eq=bin2dec([I_runcode(a,b).code(1+m*(c_indx-1):m*c_indx),dec2bin(0,↙
bpp_diff)]);
                        if decm_eq>(2^(n-1))-1
                            decm_eq=decm_eq-(2^n-1);
                        end
                        I_rec_Trnsfm(a,b).block(x_indices(p),y_indices(p))=decm_eq;
                        c_indx=c_indx+1;
                    end
            end
        end
    end
end

% Clearing unused variables from Memory space
clear I_runcode x_indices y_indices
clear c_indx freq_sum

% Denormalizing the Reconstructed Tranform matrix using the same
```

```
% Normalization matrix.
for a=1:I_dim(1)/N
    for b=1:I_dim(2)/M
        I_rec_Trnsfm(a,b).block=(I_rec_Trnsfm(a,b).block).*Norm_Mat;
    end
end


% Inverse-Discrete Cosine Transform on the reconstructed Matrix.
% y(i,j)=(2/root(NM))*sigma(i=0:N-1)sigma(j=0:M-1) Y(k,l)c(k)*c(l)*cos(pi(2i+1)k/(2N))cos(pi(2j+1)l/(2M))
% where c(u)=1/root(2) if u=0
%            = 1       if u>0
for a=1:I_dim(1)/N
    for b=1:I_dim(2)/M
        for i=1:N
            for j=1:M
                prod=0;
                for k=1:N
                    for l=1:M
                        if k==1
                           temp=double(sqrt(1/2)*I_rec_Trnsfm(a,b).block(k,l))*cos(pi*(k-1)*(2*i-1)/↙
(2*N))*cos(pi*(l-1)*(2*j-1)/(2*M));
                        else
                            temp=double(I_rec_Trnsfm(a,b).block(k,l))*cos(pi*(k-1)*(2*i-1)/(2*N))*cos(pi*↙
(l-1)*(2*j-1)/(2*M));
                        end
                        if l==1
                            temp=temp*sqrt(1/2);
                        end
```

```
                        prod=prod+temp;
                    end
                end
                prod=prod*(2/sqrt(M*N));
                I_rec((a-1)*N+i,(b-1)*M+j)=prod;
            end
        end
    end
end

% Clearing unused variables from Memory Space.
clear I_rec_Trnsfm

% Displaying the Reconstructed Image.
diff=im2double(I)*255-I_rec;
diff=diff/max(max(diff));
diff=im2uint8(diff);
I_rec=I_rec/max(max(I_rec));
I_rec=im2uint8(I_rec);

figure, imshow(I_rec);
figure, imhist(I_rec);
figure, imhist(diff);
```