BÁO CÁO BÀI TẬP LAB CUỐI KỲ

Bộ môn : Thông tin số

Nhóm	: 12A
Thành viên	: Nguyễn Văn
	Lê Quang Hoàng
	Trần Đức Phát
	Tô Ngọc Hoan
	Nguyễn Ngọc Hưng
Lớp	: 21KTMT
Giảng viên hướng dẫn	: TS. Hoàng Lê Uyên Thục

Các bài Lab được thực hiện trên laptop Lenovo Ideapad S145. Với bộ xử lý Intel(R) Core(TM) i3-8130U CPU @ 2.20GHz, 2208 Mhz, 2 Lõi, 4 Bộ xử lý logical.

Lab 1

Bài 1A: Tìm hiểu ảnh hưởng của việc thay đổi tần số lấy mẫu đến chất lượng ảnh xám. Chụp ảnh cả nhóm, chuyển về ảnh xám và thực hiện các yêu cầu sau: Thay đổi các tần số lấy mẫu trên ảnh này. Quan sát ảnh, so sánh, đánh giá chất lượng và dung lượng ảnh, rút ra nhận xét.

```
%Hàm lấy mẫu
function [out] = sampling(gray_img, scale)
[col, row] = size(gray_img);
downSampledImg = gray_img(1:scale:end,1:scale:end);
for i=1:col
    for j=1:row
        tmp(i,j) =
downSampledImg(ceil(i/scale),ceil(j/scale));
    end
end
out = tmp;
end

% Thực hiện Lab1A
im = imread('img1.jpg');
gray_img = rgb2gray(im);
imwrite(gray_img,'gray_img.png');

figure;
subplot(121); imshow(im); title('Color img');
subplot(122); imshow(gray_img); title('Gray img');

G = dir('gray_img.png'); G.bytes

figure;
%Lấy mẫu mỗi 2 bits
im2bit = sampling(gray_img,2);
subplot(221); imshow(im2bit);
title('2 bits sample');
```

```
imwrite(im2bit,'samp2.png');
samp2 = dir('samp2.png'); samp2.bytes

%Lấy mẫu mỗi 4 bits
im4bit = sampling(gray_img,4);
subplot(222); imshow(im4bit);
title('4 bits sample')
imwrite(im4bit,'samp4.png');
samp4 = dir('samp4.png'); samp4.bytes

%Lấy mẫu mỗi 8 bits
im8bit = sampling(gray_img,8);
subplot(223); imshow(im8bit);
title('8 bits sample');
imwrite(im8bit,'samp8.png');
samp8 = dir('samp8.png'); samp8.bytes

%Lấy mẫu mỗi 16 bits
im16bit = sampling(gray_img,16);
subplot(224); imshow(im16bit);
title('16 bits sample');
imwrite(im16bit,'samp16.png');
samp16 = dir('samp16.png'); samp16.bytes
```

	Ảnh xám	2 bits	4 bits	8 bits	16 bits
Dung lượng	653.985 bytes	282.542 bytes	107.687 bytes	40.609 bytes	16.740 bytes

Ảnh gốc

Ảnh xám

Lấy mẫu mỗi 2 bits

Lấy mẫu mỗi 4 bits

Lấy mẫu mỗi 8 bits

Lấy mẫu mỗi 16 bits

Nhận xét :

So sánh, đánh giá: Với tần số lấy mẫu càng nhỏ thì chất lượng ảnh càng giảm, dung lượng ảnh cũng giảm hơn so với ảnh xám ban đầu.

Nhận xét: Tần số lấy mẫu nhỏ lại thì chi tiết ảnh cũng sẽ bị mất dần. Cần một tần số lấy mẫu tối ưu để có thể giảm dung lượng ảnh nhưng đồng thời chất lượng ảnh cũng không thay đổi quá nhiều so với ảnh xám gốc (cảm quan mắt thường khó phân biệt được).

>>Tần số lấy mẫu tối ưu: lấy mẫu với mỗi 2 bits với dung lượng sau lấy mẫu là **282542 bytes.**

Bài 1B: Tìm hiểu ảnh hưởng của việc thay đổi số mức lượng tử hóa đến chất lượng ảnh xám. Cho bức ảnh xám ở câu 1A. Thay đổi số mức lượng tử hóa trên ảnh này. Quan sát ảnh, so sánh, đánh giá chất lượng ảnh và dung lượng, rút ra nhận xét. Từ đó chọn ra số mức lượng tử hóa tối ưu.

```
% Hàm lượng tử hóa
function [out] = quantize(gray_img, scale)
gray_img = double(gray_img);
quantizeBit = double(2^(8 - scale));
out = uint8(floor(gray_img/quantizeBit)*quantizeBit);
end

% Thực hiện Lab1B
img = 'img1.jpg';
im = rgb2gray(imread(img));

figure;
subplot(241);
imshow(im); title('8-bit / gray img');

im7 = quantize(im,7);
im6 = quantize(im,6);
im5 = quantize(im,5);
im4 = quantize(im,4);
im3 = quantize(im,3);
im2 = quantize(im,2);
im1 = quantize(im,1);

subplot(242), imshow(im7), title('7-bit');
subplot(243), imshow(im6), title('6-bit');
subplot(244), imshow(im5), title('5-bit');
subplot(245), imshow(im4), title('4-bit');
subplot(246), imshow(im3), title('3-bit');
subplot(247), imshow(im2), title('2-bit');
subplot(248), imshow(im1), title('1-bit');
```

```
imwrite(im,'8bitgray.png'), q8 = dir('8bitgray.png');
q8.bytes
imwrite(im7,'quant7im.png'), q7 = dir('quant7im.png');
q7.bytes
imwrite(im6,'quant6im.png'), q6 = dir('quant6im.png');
q6.bytes
imwrite(im5,'quant5im.png'), q5 = dir('quant5im.png');
q5.bytes
imwrite(im4,'quant4im.png'), q4 = dir('quant4im.png');
q4.bytes
imwrite(im3,'quant3im.png'), q3 = dir('quant3im.png');
q3.bytes
imwrite(im2,'quant2im.png'), q2 = dir('quant2im.png');
q2.bytes
imwrite(im1,'quant1im.png'), q1 = dir('quant1im.png');
q1.bytes
```

	Xám gốc / 8-bit	7-bit	6-bit	5-bit
Dung lượng (bytes)	653.985	527.512	392.537	278.446
	4-bit	3-bit	2-bit	1-bit
Dung lượng (bytes)	188.749	123.056	70.872	30.784

So sánh, đánh giá: Với số bit lượng tử hóa càng nhỏ thì số màu xám trong ảnh càng giảm, dung lượng ảnh cũng giảm dần hơn so với ảnh gốc ban đầu.

Nhận xét: Mức lượng tử hóa nhỏ lại thì số mức xám ít đi và ảnh sẽ dần chuyển thành màu đen. Cần một mức lượng tử hóa tối ưu để có thể giảm dung lượng ảnh nhưng đồng thời chất lượng ảnh cũng không thay đổi quá nhiều so với ảnh gốc (mắt thường khó phân biệt được)

⇨ **Mức lượng tử hóa tối ưu:** 5-bit với dung lượng là 278.446 bytes

Lab 1C : Kết hợp bài 1A và 1B để xét ảnh hưởng của lấy mẫu và lượng tử hóa đến chất lượng ảnh màu.

Yêu cầu: Chọn ảnh màu chụp cả nhóm, thay đổi tần số lấy mẫu trên ảnh này. Quan sát ảnh, so sánh, đánh giá chất lượng ảnh và dung lượng ảnh, rút ra nhận xét.

Thay đổi số mức lượng tử hóa trên ảnh này. Quan sát ảnh, so sánh, đánh giá chất lượng ảnh và dung lượng ảnh, rút ra nhận xét.

Lưu ý : cần quan sát sự thay đổi màu sắc. Tần số lấy mẫu và số mức lượng tử hóa có ảnh hưởng như thế nào đến màu sắc?

```
im = imread('img1.jpg');

[imR, imG, imB] = divideLayer(im);

% Lấy mẫu
downscale = 2; % Hệ số điều chỉnh lấy mẫu
imdownR = sampling(imR, downscale);
imdownG = sampling(imG, downscale);
imdownB = sampling(imB, downscale);

figure;
subplot(121); imshow(im), title('original img');

% Lượng tử
quantBit = 5; % Hệ số điều chỉnh lượng tử
quantimR = quantize(imdownR,quantBit);
quantimG = quantize(imdownG,quantBit);
quantimB = quantize(imdownB,quantBit);

out(:,:,1) = quantimR;
out(:,:,2) = quantimG;
out(:,:,3) = quantimB;
```

```
subplot(122); imshow(out); title('after got sampled and
quantized');
```

	Ảnh gốc	Lấy mẫu 2 Lượng tự 5	Lấy mẫu 3 Lượng tử 5	Lấy mẫu 2 Lượng tử 4
Dung lượng	1.511.540 bytes	351.010 bytes	199.501 bytes	247.371 bytes

Ảnh màu gốc

Ảnh lấy mẫu 2, lượng tử 5

Ảnh lấy mẫu 3, lượng tử 5

Ảnh lấy mẫu 2, lượng tử 4

Chọn hệ số lấy mẫu là 2 bits và lượng tử hóa là 5-bit theo như Lab1A và Lab1B. Thì được ảnh như bên dưới ít có sự sai khác với ảnh gốc nhất.

Khi tăng hệ số lấy mẫu lên 3 bits và giữ nguyên mức lượng tử hóa 5-bit thì ảnh bắt đầu xuống hiện hiệu ứng răng cưa nhỏ tại các chi tiết góc cạnh và có độ tượng phản cao, ví dụ như tại vị trí gọng kính và tay.

Còn khi giữ nguyên hệ số lấy mẫu bằng 2 bits và mức lượng tử hóa xuống 4-bit thì các mảng màu trong ảnh bắt đầu phân vùng tại các vị trí sắc độ có sự thay đổi ít và chiếu sáng ko đồng đều, ví dụ như trên quần áo hoặc khuôn mặt.

⇨ Tần số lấy mẫu và lượng tử hóa tối ưu cho một tấm ảnh màu là 2-bit và 5 bits. Giúp giảm một tấm ảnh màu gốc từ 1.511.540 bytes xuống còn 351.010 bytes mà không làm thay đổi chất lượng của ảnh quá nhiều về mặt cảm quan khi nhìn. Giảm dung lượng của tấm ảnh xuống 4,3 lần.

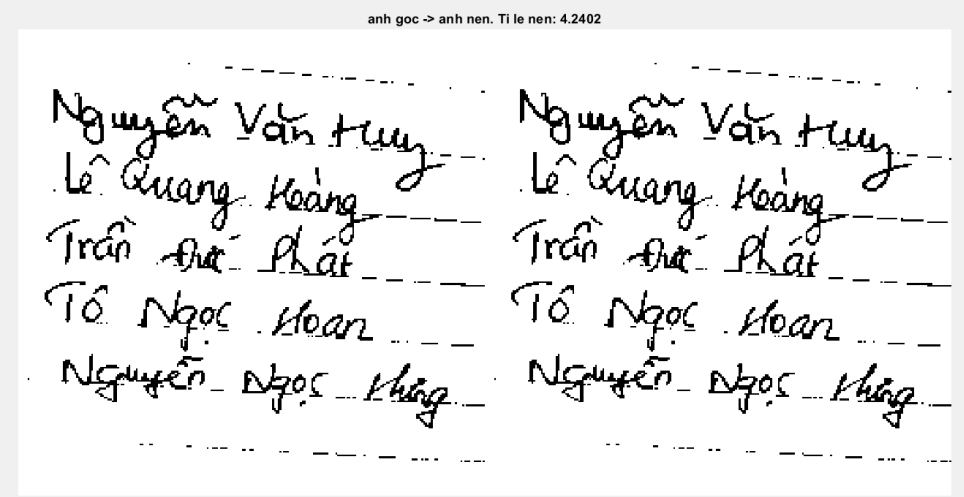
Lab 2

Code:

```
% đọc ảnh
i = imread('test1.jpg');
i = imresize(i,[250,250]); % kích thước ảnh -> 250x250
% chuyển ảnh nhị phân
i = rgb2gray(i);
i = imbinarize(i); %
% đếm và lưu runlength
pixels = i(:);
compressed_data = [];
current_pixel = pixels(1);
count = 1;
for k = 2:length(pixels)
    if pixels(k) == current_pixel
        count = count + 1;
    else
        compressed_data = [compressed_data; current_pixel,
count];
        current_pixel = pixels(k);
        count = 1;
    end
end
% m? hóa và gi?i m?
compressed_data = [compressed_data; current_pixel, count];
compressed_size = sum(cellfun(@(x) numel(dec2bin(x)),
num2cell(compressed_data(:,1))) + ...
cellfun(@(x) numel(dec2bin(x)),
num2cell(compressed_data(:,2))));
```

```
compression_ratio = 250 *250 / compressed_size;
decoded_pixels = [];
for k = 1:size(compressed_data, 1)
    decoded_pixels = [decoded_pixels;
    repmat(compressed_data(k,1), compressed_data(k,2),
1)];
end
% Tao lai ma tran anh tu du lieu giai nen
decoded_image = reshape(decoded_pixels, size(i));
figure
imshowpair(i,decoded_image,'montage');
title(['anh goc -> anh nen. Ti le nen:
',num2str(compression_ratio)]);
```

Kết quả:



Nhận xét, đánh giá:

+ Kết quả cho thấy ảnh gốc và ảnh sau khi giải nén gần như là giống nhau
+ Tỷ lệ nén nhỏ 4.2402
+ Ảnh sau khi giải nén vẫn giữ được những chi tiết vốn có của ảnh gốc

Lab 3

Bài 3A

Đoạn văn bản có tên thành viên trong nhóm:

```
inputString = 'HOAN HOANG HUNG HUY PHAT';
```

Văn bản được mã hoá	Văn bản bị lỗi
1 0 0 1 0 0 0	**0** 0 0 1 0 0 0

1 0 0 1 1 1 1	1 0 0 1 1 1 1
1 0 0 0 0 0 1	1 0 0 0 0 0 1
1 0 0 1 1 1 0	1 0 0 1 1 1 0
0 1 0 0 0 0 0	0 1 0 0 0 0 0
1 0 0 1 0 0 0	**0** 0 0 1 0 0 0
1 0 0 1 1 1 1	1 0 0 1 1 1 1
1 0 0 0 0 0 1	1 0 0 0 0 0 1
1 0 0 1 1 1 0	1 0 0 1 1 1 0
1 0 0 0 1 1 1	1 0 0 0 1 1 1
0 1 0 0 0 0 0	0 1 0 0 0 0 0
1 0 0 1 0 0 0	**0** 0 0 1 0 0 0
1 0 1 0 1 0 1	1 0 1 0 1 0 1
1 0 0 1 1 1 0	1 0 0 1 1 1 0
1 0 0 0 1 1 1	1 0 0 0 1 1 1
0 1 0 0 0 0 0	0 1 0 0 0 0 0
1 0 0 1 0 0 0	**0** 0 0 1 0 0 0
1 0 1 0 1 0 1	1 0 1 0 1 0 1
1 0 1 1 0 0 1	1 0 1 1 0 0 1
0 1 0 0 0 0 0	0 1 0 0 0 0 0
1 0 1 0 0 0 0	**0** 0 1 0 0 0 0
1 0 0 1 0 0 0	1 0 0 1 0 0 0
1 0 0 0 0 0 1	1 0 0 0 0 0 1
1 0 1 0 1 0 0	1 0 1 0 1 0 0
HOAN HOANG HUNG HUY PHAT	OAN OANG UNG UY HAT

Vị trí màu đỏ in đậm là vị trí sai

Khi mã hoá bằng mã ASCII thì văn bản bị sai không có khả năng phát hiện lỗi

Code:

```
% Ví dụ sử dụng:
inputString = 'HOAN HOANG HUNG HUY PHAT';
disp(inputString);
binaryMatrix = convertToBinaryMatrix(inputString,7);
%chuyển đổi kí tự chuyển đi thành ma trận
c=binaryMatrix;          %truyền đi với mã ascii 7 bit
disp(c);
c(1,1)=c(1,1)+1;
c(6,1)=c(6,1)+1;
c(12,1)=c(12,1)+1;
c(17,1)=c(17,1)+1;
```

```
c(21,1)=c(21,1)+1;
r=mod(c,2);
disp(r);
dulieu=matrixToString(r);
disp(dulieu);
```

3B

Mã khi phát	Mã khi thu
1 0 0 1 0 0 1	**0** 0 0 1 0 0 1
0 0 0 1 1 1 0	0 0 0 1 1 1 0
0 0 1 1 0 1 1	0 0 1 1 0 1 1
1 1 1 0 0 0 1	1 1 1 0 0 0 1
0 0 0 0 0 0 0	0 0 0 0 0 0 0
1 1 0 0 1 0 0	1 1 0 0 1 0 0
1 1 1 0 0 0 1	1 1 1 0 0 0 1
0 1 0 0 0 1 1	0 1 0 0 0 1 1
0 0 0 1 1 1 0	0 0 0 1 1 1 0
0 0 1 0 1 0 1	0 0 1 0 1 0 1
0 0 1 0 1 0 1	**1** 0 1 0 1 0 1
0 1 1 1 0 0 0	0 1 1 1 0 0 0
1 1 0 0 1 0 0	1 1 0 0 1 0 0
0 0 0 1 1 1 0	0 0 0 1 1 1 0
1 0 0 1 0 0 1	1 0 0 1 0 0 1
1 1 0 1 0 1 0	1 1 0 1 0 1 0
0 0 0 1 1 1 0	0 0 0 1 1 1 0
1 1 0 1 0 1 0	1 1 0 1 0 1 0
0 0 0 0 0 0 0	0 0 0 0 0 0 0
0 1 0 0 0 1 1	0 1 0 0 0 1 1
1 0 0 0 1 1 1	1 0 0 0 1 1 1
1 0 1 0 0 1 0	1 0 1 0 0 1 0
1 0 1 1 1 0 0	**0** 0 1 1 1 0 0
0 0 1 1 0 1 1	0 0 1 1 0 1 1
1 0 1 0 0 1 0	1 0 1 0 0 1 0
0 0 1 1 0 1 1	0 0 1 1 0 1 1
1 0 1 0 0 1 0	1 0 1 0 0 1 0
0 0 0 0 0 0 0	0 0 0 0 0 0 0
1 0 0 1 0 0 1	1 0 0 1 0 0 1
0 0 0 1 1 1 0	0 0 0 1 1 1 0
0 1 0 1 1 0 1	0 1 0 1 1 0 1
0 1 1 0 1 1 0	0 1 1 0 1 1 0
1 1 0 0 1 0 0	**0** 1 0 0 1 0 0

1 0 1 0 0 1 0 0 0 0 0 0 0 0 1 0 1 0 0 1 0 0 0 0 1 1 1 0 0 0 1 0 1 0 1 0 0 1 0 1 0 1 0 0 0 0 0 0 0 **1** 1 0 1 0 1 0 0 1 0 0 0 1 1	1 0 1 0 0 1 0 0 0 0 0 0 0 0 1 0 1 0 0 1 0 0 0 0 1 1 1 0 0 0 1 0 1 0 1 0 0 1 0 1 0 1 0 0 0 0 0 0 0 **0** 1 0 1 0 1 0 0 1 0 0 0 1 1
HOAN HOANG HUNG HUY PHAT	HOAN HOANG HUNG HUY PHAT

Khi ta sử dụng mã khối (4,7) thì tin khi nhận được chúng ta có thể phát hiện cũng như tự sửa lỗi mà không cần truyền lại.

```
H=[1 0 1 1 1 0 0
   1 1 0 1 0 1 0
   1 1 1 0 0 0 1];
G=[ 1 0 0 0 1 1 1
    0 1 0 0 0 1 1
    0 0 1 0 1 0 1
    0 0 0 1 1 1 0];
e={ [0 0 0 0 0 0 0]
    [1 0 0 0 0 0 0]
    [0 1 0 0 0 0 0]
    [0 0 1 0 0 0 0]
    [0 0 0 1 0 0 0]
    [0 0 0 0 1 0 0]
    [0 0 0 0 0 1 0]
    [0 0 0 0 0 0 1]};
mSyn=size(e,1);

%tạo mã syndrone từ ma trận lỗi vector
for i=1:mSyn
    syndrone{i}=e{i}*H';
end

% tạo ma trận syndrone
for i=1:size(syndrone,2)-1
    syns(i,:)=syndrone{1,i+1};
end
% Ví dụ sử dụng:
inputString = 'HOAN HOANG HUNG HUY PHAT';
```

```
binaryMatrix = convertToBinaryMatrix(inputString,4);
%chuyển đổi kí tự chuyển đi thành ma trận
disp(inputString);

% m là ma tin cần chuyển đi
m=binaryMatrix;
xm=size(m,1);
% c là tin phát
c=m(1:xm,:)*G;
c=mod(c,2);              %truyền đi với mã khối (4,7)
disp(c);
% r là tin thu, bị sai khác một số vị trí
c(1,1)=c(1,1)+1;
c(11,1)=c(11,1)+1;
c(23,1)=c(23,1)+1;
c(33,1)=c(33,1)+1;
c(41,1)=c(41,1)+1;

r=mod(c,2);
disp(r);
% kiểm tra và sửa lỗi
for i=1:size(r,1)
    s=r(i,:)*syns;
    s=mod(s,2);
    for j=1:size(e)
        if (s==syndrone{1,j})
            r(i,:)=r(i,:)+e{j,1};
            r(i,:)=mod(r(i,:),2);
            break
        end
    end
end

%loại bỏ CRC
dulieu=r(:,1:4);


% Hiển thị ma trận 4 cột
str=matrixToString(dulieu);
disp(str);
```

3C

Tăng lỗi lên 2 lỗi ở một kí tự:

```
c(1,1)=c(1,1)+1;
c(2,1)=c(2,1)+1;
c(11,1)=c(11,1)+1;
c(12,1)=c(12,1)+1;
c(23,1)=c(23,1)+1;
c(24,1)=c(24,1)+1;
c(33,1)=c(33,1)+1;
c(34,1)=c(34,1)+1;
c(41,1)=c(41,1)+1;
c(42,1)=c(42,1)+1;
```

Tin khi truyền	Tin khi nhận
HOAN HOANG HUNG HUY PHAT	HOAN HOANG HUNG HUY PHAT

Nhận xét: Tin sau khi nhận có thể phát hiện và sửa lỗi. Khi làm sai hai lỗi của một kí tự cách nhau 4 bit thì có thể sửa lỗi do cứ 4 bit sẽ được mã hoá thành 1 khối nên có thể phát hiện được lỗi sai

3D

Mã khối tuyến tính (7,11)

```
H=[0 0 1 1 1 0 1 1 0 0 0
   0 1 1 0 1 1 0 0 1 0 0
   1 0 1 0 0 1 1 0 0 1 0
   1 1 0 1 0 0 1 0 0 0 1];
G=[ 1 0 0 0 0 0 0 0 0 1 1
    0 1 0 0 0 0 0 0 1 0 1
    0 0 1 0 0 0 0 1 1 1 0
    0 0 0 1 0 0 0 1 0 0 1
    0 0 0 0 1 0 0 1 1 0 0
    0 0 0 0 0 1 0 0 1 1 0
    0 0 0 0 0 0 1 1 0 1 1];
e={ [0 0 0 0 0 0 0 0 0 0 0]
    [1 0 0 0 0 0 0 0 0 0 0]
    [0 1 0 0 0 0 0 0 0 0 0]
    [0 0 1 0 0 0 0 0 0 0 0]
    [0 0 0 1 0 0 0 0 0 0 0]
    [0 0 0 0 1 0 0 0 0 0 0]
    [0 0 0 0 0 1 0 0 0 0 0]
```

```
[0 0 0 0 0 0 1 0 0 0 0]
[0 0 0 0 0 0 0 1 0 0 0]
[0 0 0 0 0 0 0 0 1 0 0]
[0 0 0 0 0 0 0 0 0 1 0]
[0 0 0 0 0 0 0 0 0 0 1]
[0 1 0 0 0 0 0 0 0 1 0]
[1 0 0 1 0 0 0 0 0 0 0]
[1 0 1 0 0 0 0 0 0 0 0]
[1 0 0 0 1 0 0 0 0 0 0]};
```

Với ma trận H, ma trận sinh và vector lỗi như trên ta được giá trị syndrone tương ứng như sau:

Vector lỗi	Syndrone tương ứng
[0 0 0 0 0 0 0 0 0 0 0]	0 0 0 0
[1 0 0 0 0 0 0 0 0 0 0]	0 0 1 1
[0 1 0 0 0 0 0 0 0 0 0]	0 1 0 1
[0 0 1 0 0 0 0 0 0 0 0]	1 1 1 0
[0 0 0 1 0 0 0 0 0 0 0]	1 0 0 1
[0 0 0 0 1 0 0 0 0 0 0]	1 1 0 0
[0 0 0 0 0 1 0 0 0 0 0]	0 1 1 0
[0 0 0 0 0 0 1 0 0 0 0]	1 0 1 1
[0 0 0 0 0 0 0 1 0 0 0]	1 0 0 0
[0 0 0 0 0 0 0 0 1 0 0]	0 1 0 0
[0 0 0 0 0 0 0 0 0 1 0]	0 0 1 0
[0 0 0 0 0 0 0 0 0 0 1]	0 0 0 1
[0 1 0 0 0 0 0 0 0 1 0]	0 1 1 1
[1 0 0 1 0 0 0 0 0 0 0]	1 0 1 0
[1 0 1 0 0 0 0 0 0 0 0]	1 1 0 1
[1 0 0 0 1 0 0 0 0 0 0]	1 1 1 1

Khi cho một lỗi như bài 3A thì tin vẫn nhận và sửa lỗi thành công

Khi cho 2 lỗi như bài 3C thì tin vẫn nhận và sửa lỗi thành công, tuy nhiên nó chỉ là 1 trường hợp mà mà mã khối (7,11) có thể phát hiện và sửa. Tuy nhiên có rất nhiều trường hợp mà sai 2 lỗi mã khối (7,11) chỉ phát hiện nhưng không sửa được:

```
>> maKhoi711
HOAN HOANG HUNG HUY PHAT
HOAN HOANG HUNG HUY PHAT
```

Sửa được với lỗi số 1 và 5

```
>> maKhoi711
HOAN HOANG HUNG HUY PHAT
*OAN *OANG *UNG HUY 2HAT
```

Không sửa được lỗi số 1 và 2

```
HOAN HOANG HUNG HUY PHAT
dOAN dOANG dUNG dUY |HAT
```

Không sửa được lỗi số 2 và 4

```
>> maKhoi711
HOAN HOANG HUNG HUY PHAT
*OAN *OANG *UNG *UY 2HAT
```

Không sửa được lỗi số 1 và 6

Nhận xét: Mã khối (7,11) có khả năng phát hiện và sửa được lỗi đơn, với 2 lỗi thì mã chỉ có thể phát hiện và sửa được tối đa 4 lỗi dựa trên sự lựa chọn của người lập trình khi chọn lỗi

Mã khối (7,11) có tỉ lệ mã cao hơn (0.64) so với mã khối (4,7) (0.58). Tuy nhiên khi gặp các trường hợp các bit lỗi có vị trí gần nhau thì khả năng sửa lỗi của mã khối (4,7) cao hơn.