Bài 3: Nén ảnh JPEG

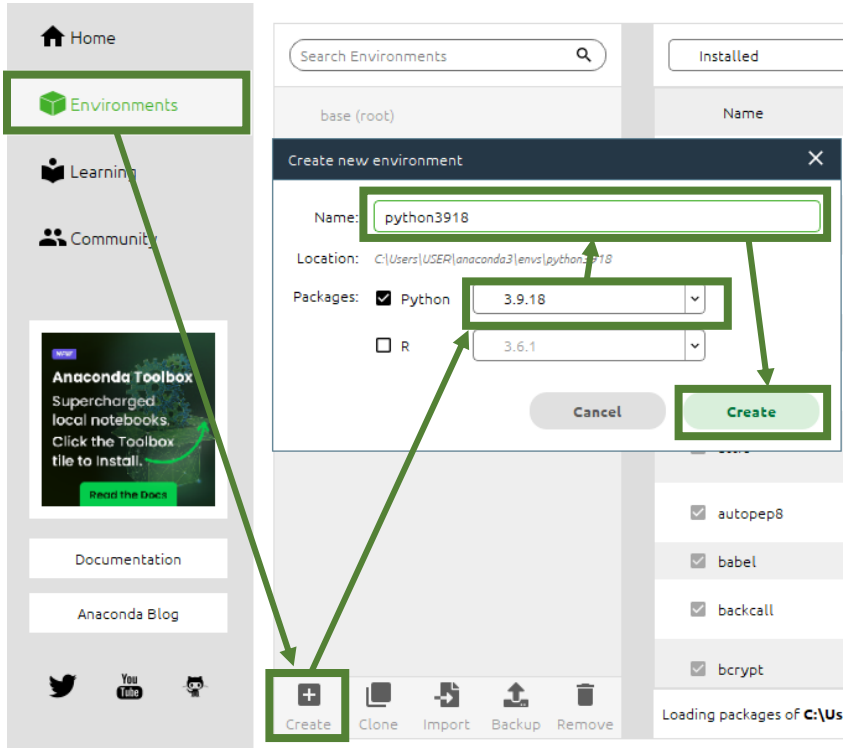
1. Mục tiêu

Tìm hiểu quy trình nén ảnh JPEG.

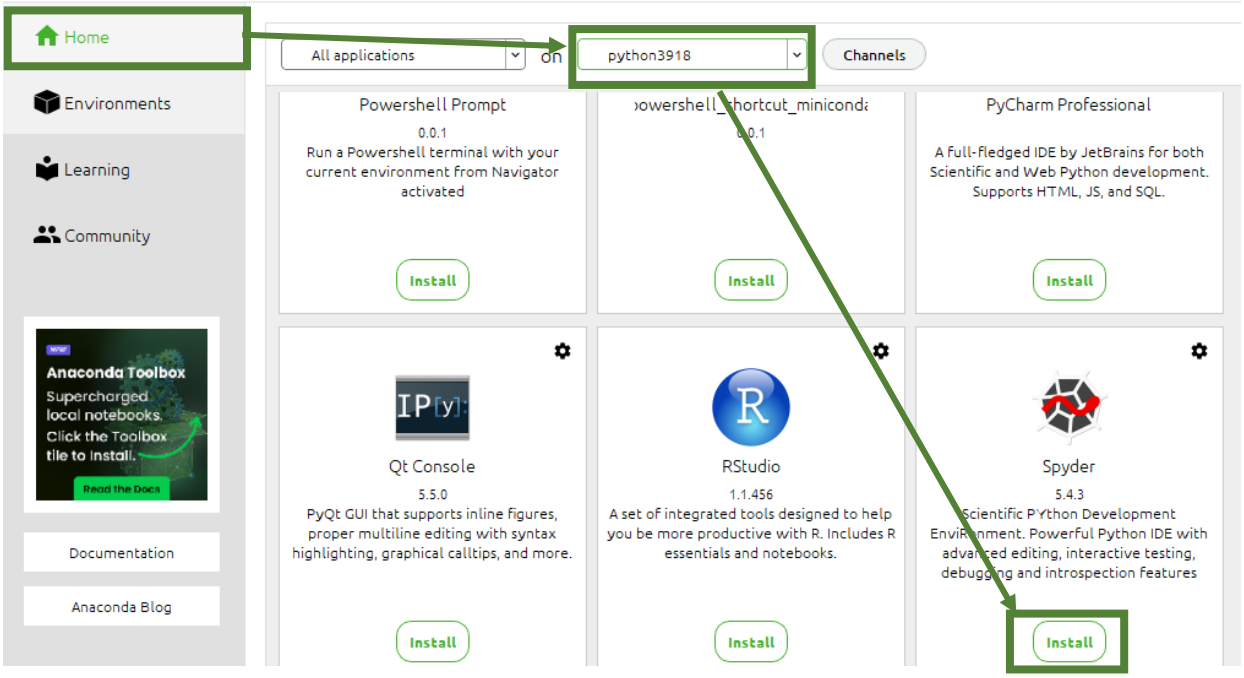
2. Cài đặt phần mềm

Cài đặt phần mềm:

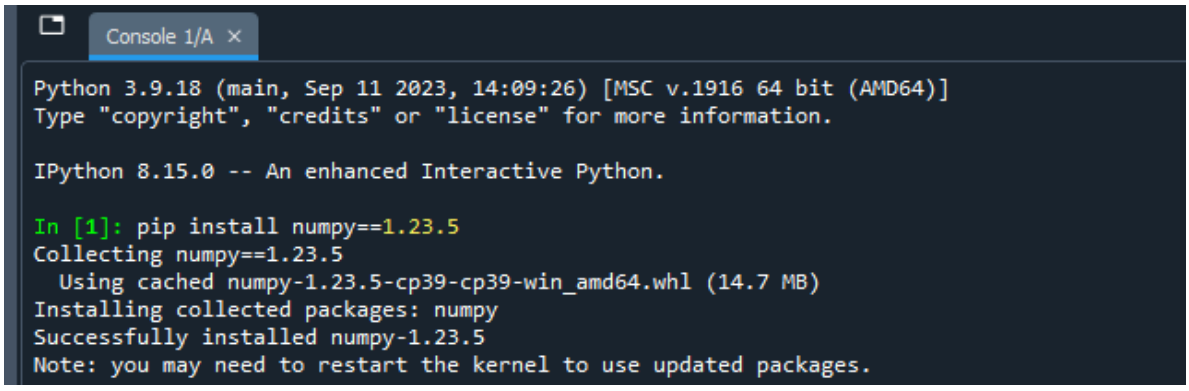
1. Mở phần mềm Anaconda Navigator
2. Chọn "Environments" → Chọn "Create" → Chọn Python 3.9.18 → Đặt tên ví dụ "python3918" cho dễ nhớ phiên bản → Chọn "Create".



3. Chọn "Home" → Chọn vào tên python vừa tạo → Chọn "Install" Spyder.



4. Launch Spyder.
5. Tại cửa sổ Console, cài đặt các thư viện sau:
   a. pip install numpy==1.23.5
   b. pip install pillow==9.2.0
   c. pip install scipy==1.9.1
   d. pip install bitstream==2.6.0.2
   e. pip install matplotlib

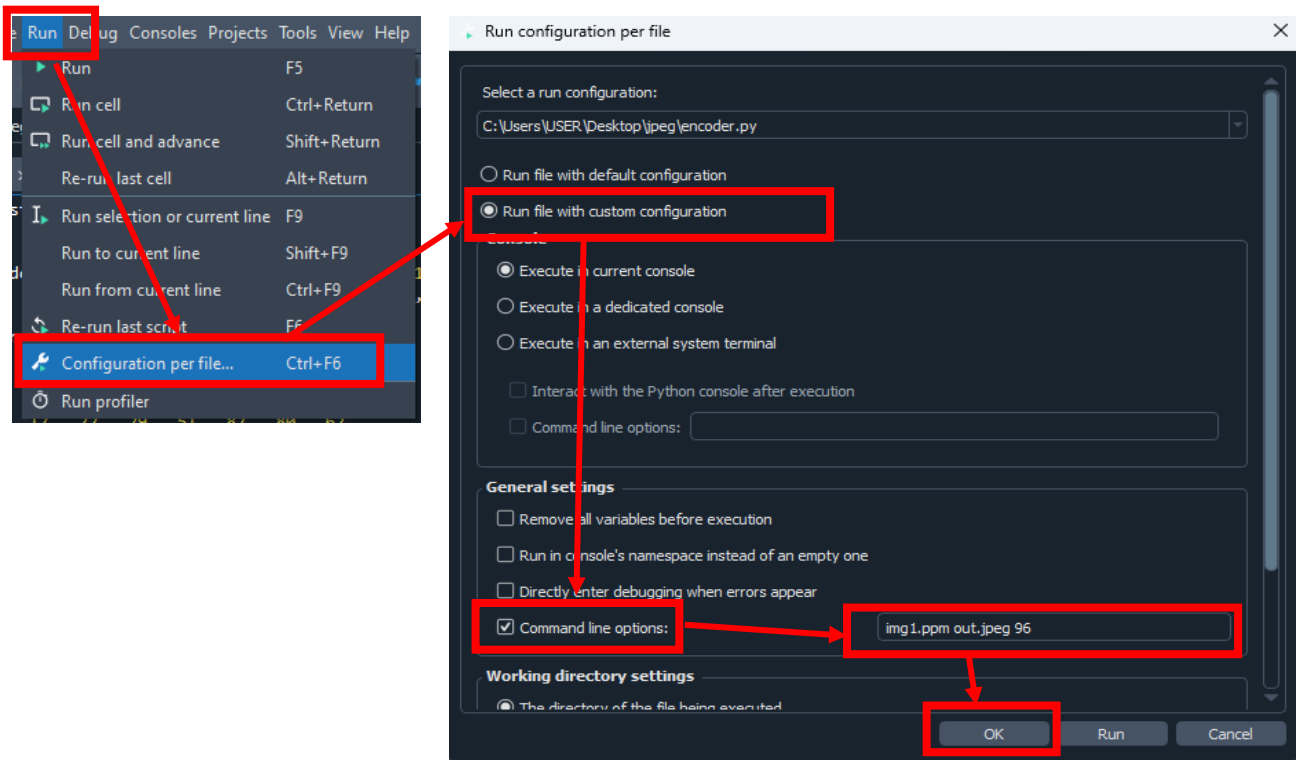


- **Source code demo JPEG:**

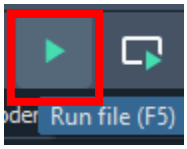
https://drive.google.com/drive/folders/1XwA8LK8uvBL--izNdprUWcSXRRrvjylU?usp=sharing

Chạy chương trình

1. Mở phần mềm Spyder.
2. Cấu hình tham số chọn "Run" → Chọn "Configuration per file…" → Chọn "Run file with custom configuration" → tích "Command line option" → gõ <ảnh đầu vào><ảnh đầu ra><chất lượng nén>. Ví dụ: ảnh đầu vào là img1.ppm (định dạng ảnh đầu vào là **ppm**), ảnh đầu ra có tên là out.jpeg (định dạng ảnh đầu ra là **jpeg/jpg**) và chất lượng 96 (trong phạm vi 0 →100), thì đối số truyền vào sẽ là "img1.ppm out.jpeg 96".



3. Chọn RunFile "encoder.py"

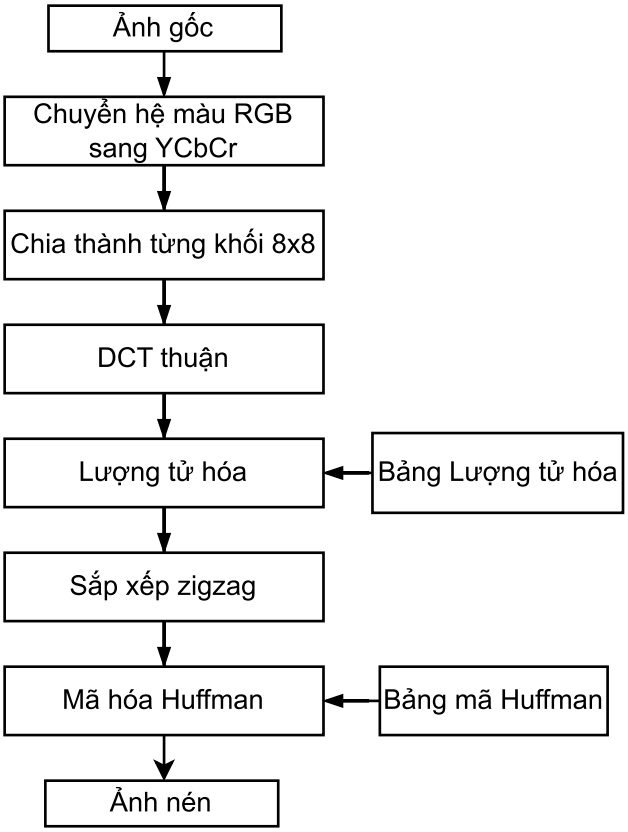


3. Chuẩn nén JPEG

Chuẩn nén JPEG (Joint Photographic Experts Group) là một phương pháp nén hình ảnh phổ biến được sử dụng rộng rãi để giảm kích thước của hình ảnh trong khi vẫn cố gắng duy trì mức độ chất lượng hình ảnh chấp nhận được. Chuẩn nén JPEG dựa trên việc loại bỏ thông tin không cần thiết và sử dụng biến đổi không gian màu (chủ yếu là chuyển đổi từ không gian màu RGB sang không gian màu YCbCr) để biểu diễn hình ảnh.

Các bước nén ảnh cơ bản:

1. Chuyển hệ màu RGB thành YcbCr.
2. Chia ảnh thành các khối 8x8.
3. Mỗi khối 8x8 được chuyển đổi bằng Biến đổi Cosine rời rạc (DCT).
4. Mỗi khối sau khi DCT được lượng tử hóa bằng bảng lượng tử (chia hệ số DCT cho hệ số lượng tử tương ứng).
5. Hệ số sau khi lượng tử được sắp xếp lại theo mô hình zigzag.
6. Mã hóa Huffman: Mã hóa những hệ số khác 0 trong quá trình quét zigzag.



1. Chuyển đổi ảnh RGB thành ảnh YCbCr

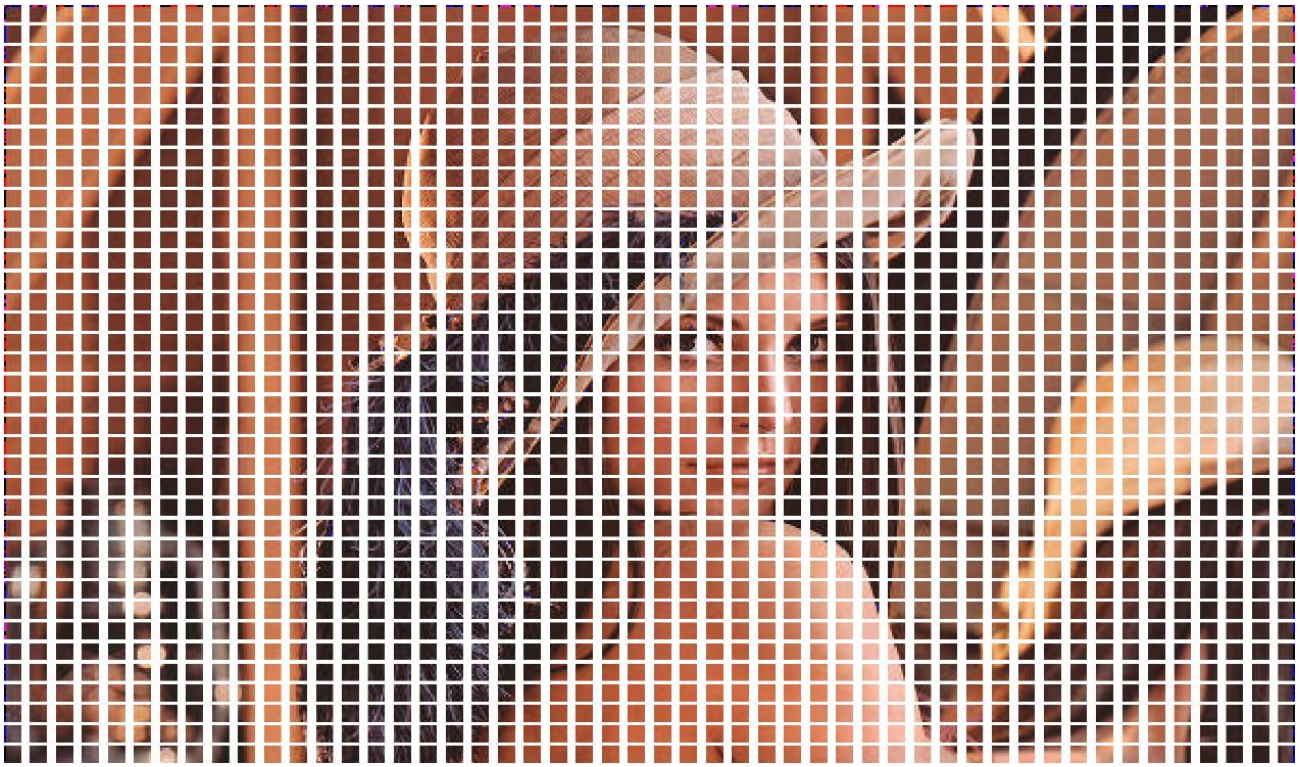
$$Y = 16 + \frac{65.481 * R}{255} + \frac{128.533 * G}{255} + \frac{24.966 * B}{255}$$

$$C_B = 128 - \frac{37.797 * R}{255} + \frac{74.203 * G}{255} + \frac{112.0 * B}{255}$$

$$C_R = 128 + \frac{112.0 * R}{255} + \frac{93.786 * G}{255} + \frac{18.214 * B}{255}$$

```
def rgb2ycbcr(im):
    cbcr = numpy.empty_like(im)
    r = im[:,:,0]
    g = im[:,:,1]
    b = im[:,:,2]
    # Y
    cbcr[:,:,0] = 16 +  65.481/255 * r + 128.553/255 * g + 24.966/255 * b
    # Cb
    cbcr[:,:,1] = 128 - 37.797/255 * r - 74.203/255 * g + 112.0/255 * b
    # Cr
    cbcr[:,:,2] =  128 +  112.0/255 * r - 93.786/255 * g - 18.214/255 * b
    return numpy.uint8(cbcr)
```

2. Chia ảnh thành các khối 8x8



3. DCT thuận trên tất cả khối 8x8

$$F(u,v) = \frac{\varepsilon_u \varepsilon_v}{4} \sum_{x=0}^{7} \sum_{y=0}^{7} f(x,y) * \cos\left(\frac{(2x+1)u\pi}{16}\right) * \cos\left(\frac{(2y+1)v\pi}{16}\right)$$

Trong đó, $\varepsilon_u = \begin{cases} \frac{1}{\sqrt{2}} \text{ nếu } u = 0 \\ 1 \text{ nếu } u > 0 \end{cases}$, $\varepsilon_v = \begin{cases} \frac{1}{\sqrt{2}} \text{ nếu } v = 0 \\ 1 \text{ nếu } v > 0 \end{cases}$

Code:

```
# use DCT transform form fft libary
Y_DCTMatrix = fftpack.dct(fftpack.dct(
    Y_matrix[i:i + 8, j:j + 8], norm='ortho').T, norm='ortho').T
Cb_DCTMatrix = fftpack.dct(fftpack.dct(
    Cb_matrix[i:i + 8, j:j + 8], norm='ortho').T, norm='ortho').T
Cr_DCTMatrix = fftpack.dct(fftpack.dct(
    Cr_matrix[i:i + 8, j:j + 8], norm='ortho').T, norm='ortho').T
```

Ví dụ:

```
block8x8:
 [[-98 -97 -97 -97 -98 -98 -97 -97]
 [-98 -97 -97 -97 -98 -98 -98 -97]
 [-98 -97 -97 -97 -98 -99 -98 -98]
 [-98 -97 -97 -98 -99 -99 -99 -99]
 [-98 -97 -97 -98 -99 -99 -99 -99]
 [-97 -96 -96 -97 -99 -99 -99 -99]
 [-96 -96 -96 -97 -98 -99 -99 -99]
 [-96 -95 -95 -96 -98 -99 -99 -99]]
DCT:
 [[-7.81500000e+02  5.85147893e+00 -1.91341716e-01 -3.47888151e+00
  -1.00000000e+00  1.02233374e-01 -4.61939766e-01 -3.83570239e-01]
 [-3.46759961e-01 -4.23375218e+00  2.45196320e-01  6.76495125e-02
   1.96423740e-01 -9.00970659e-02 -3.14171165e-01 -1.73619396e-01]
 [ 2.88372398e+00  1.01563641e-01 -7.32233047e-02 -2.07867403e-01
   4.61939766e-01  1.38892558e-01 -1.76776695e-01 -2.02022643e-02]
 [-2.93968901e-01 -4.94426208e-01  4.04291143e-01  1.80852469e-01
  -3.46759961e-01  1.33991229e-01  2.07867403e-01 -1.63320371e-01]
 [-5.00000000e-01  0.00000000e+00  0.00000000e+00  3.14018492e-16
  -5.00000000e-01  1.37383090e-16  0.00000000e+00 -7.85046229e-17]
 [-1.96423740e-01  1.63320371e-01  1.55076342e-01  3.83991229e-01
   6.89748448e-02  1.90467875e-01 -1.38892558e-01  3.59127183e-01]
 [ 4.29110718e-01  2.45196320e-01 -1.76776695e-01 -5.01836304e-01
  -1.91341716e-01  3.35316298e-01 -4.26776695e-01 -4.87725805e-02]
 [-6.89748448e-02 -4.23619396e-01 -4.87725805e-02  2.36543675e-01
   2.93968901e-01  6.76495125e-02 -2.97987381e-01  3.62431838e-01]]
```

4. Lượng tử hóa

$$Fq(u,v) = \left\lfloor \frac{F(u,v)}{Q(u,v)} + 0.5 \right\rfloor$$

Trong đó, F(u,v) là bảng hệ số sau khi biến đổi DCT thuận, Q(u,v) là bảng lượng tử.

Cách tính bảng lượng tử theo chất lượng nén (0 < Q < 100)

$$T_s[i] = \left\lfloor \frac{S * T[i] + 50}{100} \right\rfloor$$

Trong đó: T[i] là phần tử thứ I trong bảng lượng tử.

$$S = \begin{cases} \frac{5000}{Q} \; nếu \; Q < 50 \\ 200 - 2Q \; nếu \; Q \; \geq 50 \end{cases}$$

Bảng lượng tử tiêu chuẩn cho Luma

```
std_luminance_quant_tbl:
 [[ 16  11  10  16  24  40  51  61]
 [ 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]]
```

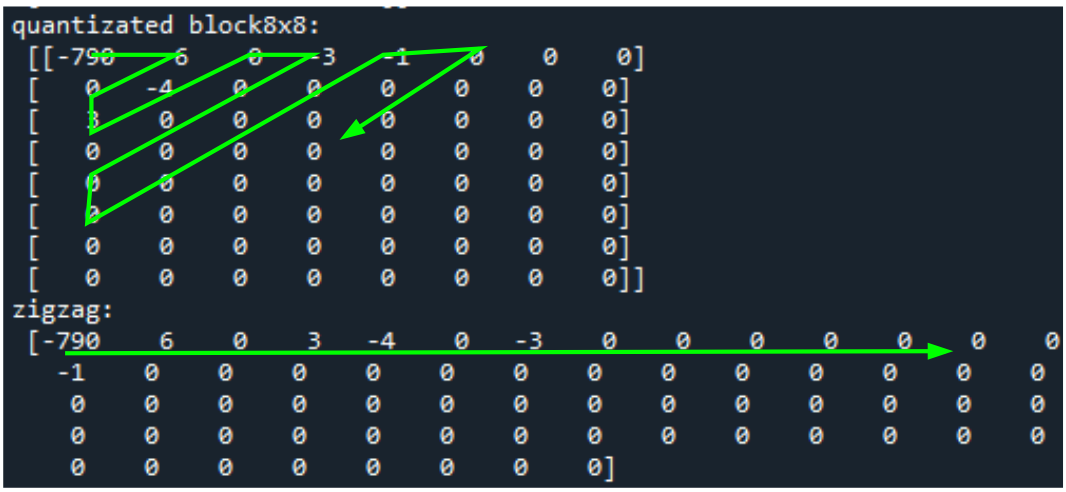
Bảng lượng tử chất lượng **96** cho Luma

```
luminanceQuantTbl:
 [[ 1  1  1  1  2  3  4  5]
 [ 1  1  1  2  2  5  5  4]
 [ 1  1  1  2  3  5  6  4]
 [ 1  1  2  2  4  7  6  5]
 [ 1  2  3  4  5  9  8  6]
 [ 2  3  4  5  6  8  9  7]
 [ 4  5  6  7  8 10 10  8]
 [ 6  7  8  8  9  8  8  8]]
```

Ví dụ:

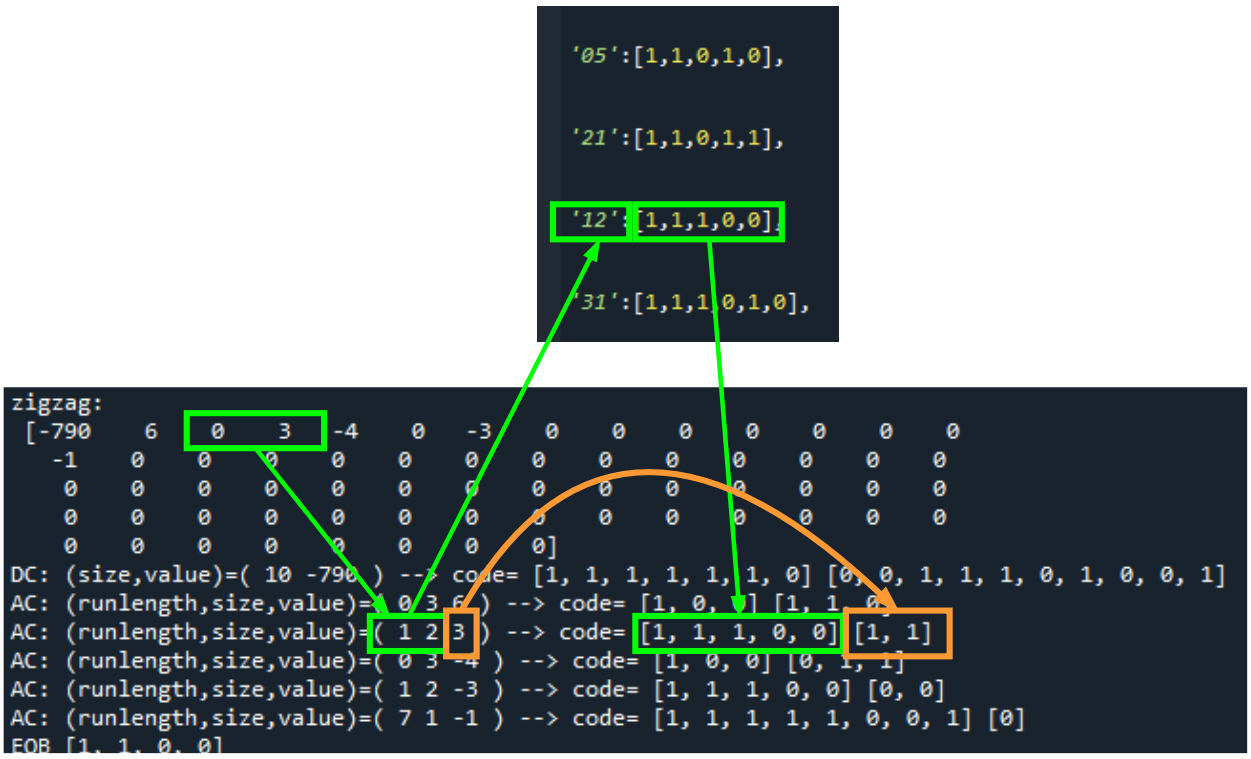
```
DCT:
 [[-7.89500000e+02  5.85147893e+00 -1.91341716e-01 -3.47888151e+00
  -1.00000000e+00  1.02233374e-01 -4.61939766e-01 -3.83570239e-01]
 [-3.46759961e-01 -4.23375218e+00  2.45196320e-01  6.76495125e-02
   1.96423740e-01 -9.00970659e-02 -3.14171165e-01 -1.73619396e-01]
 [ 2.88372398e+00  1.01563641e-01 -7.32233047e-02 -2.07867403e-01
   4.61939766e-01  1.38892558e-01 -1.76776695e-01 -2.02022643e-02]
 [-2.93968901e-01 -4.94426208e-01  4.04291143e-01  1.80852469e-01
  -3.46759961e-01  1.33991229e-01  2.07867403e-01 -1.63320371e-01]
 [-5.00000000e-01  0.00000000e+00  0.00000000e+00  3.14018492e-16
  -5.00000000e-01  1.37383090e-16  0.00000000e+00 -7.85046229e-17]
 [-1.96423740e-01  1.63320371e-01  1.55076342e-01  3.83991229e-01
   6.89748448e-02  1.90467875e-01 -1.38892558e-01  3.59127183e-01]
 [ 4.29110718e-01  2.45196320e-01 -1.76776695e-01 -5.01836304e-01
  -1.91341716e-01  3.35316298e-01 -4.26776695e-01 -4.87725805e-02]
 [-6.89748448e-02 -4.23619396e-01 -4.87725805e-02  2.36543675e-01
   2.93968901e-01  6.76495125e-02 -2.97987381e-01  3.62431838e-01]]
luminanceQuantTbl:
 [[ 1  1  1  1  2  3  4  5]
 [ 1  1  1  2  2  5  5  4]
 [ 1  1  1  2  3  5  6  4]
 [ 1  1  2  2  4  7  6  5]
 [ 1  2  3  4  5  9  8  6]
 [ 2  3  4  5  6  8  9  7]
 [ 4  5  6  7  8 10 10  8]
 [ 6  7  8  8  9  8  8  8]]
quantizated block8x8:
 [[-790    6    0   -3   -1    0    0    0]
 [   0   -4    0    0    0    0    0    0]
 [   3    0    0    0    0    0    0    0]
 [   0    0    0    0    0    0    0    0]
 [   0    0    0    0    0    0    0    0]
 [   0    0    0    0    0    0    0    0]
 [   0    0    0    0    0    0    0    0]
 [   0    0    0    0    0    0    0    0]]
```

5. Mô hình zigzag



6. Mã hóa Huffman

DC	Symbol-1 (SIZE)	Symbol-2 (AMPLITUDE)
AC	Symbol-1 (RUN-LENGTH, SIZE)	Symbol-2 (AMPLITUDE)


'05':[1,1,0,1,0],
'21':[1,1,0,1,1],
'12':[1,1,1,0,0],
'31':[1,1,1,0,1,0],
zigzag:
[-790 6 0 3 -4 0 -3 0 0 0 0 0 0 0
-1 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0]
DC: (size,value)=(10 -790) --> code= [1, 1, 1, 1, 1, 1, 0] [0, 0, 1, 1, 1, 0, 1, 0, 0, 1]
AC: (runlength,size,value)=(0 3 6) --> code= [1, 0, 0] [1, 1, 0]
AC: (runlength,size,value)=(1 2 3) --> code= [1, 1, 1, 0, 0] [1, 1]
AC: (runlength,size,value)=(0 3 -4) --> code= [1, 0, 0] [0, 1, 1]
AC: (runlength,size,value)=(1 2 -3) --> code= [1, 1, 1, 0, 0] [0, 0]
AC: (runlength,size,value)=(7 1 -1) --> code= [1, 1, 1, 1, 1, 0, 0, 1] [0]
EOB [1, 1, 0, 0]

Bài tập

1. Viết chương trình thực hiện DCT thay cho DCT từ thư viện fft.
2. Thuật toán tính nhanh DCT (Bin DCT)
   - Đọc hiểu và giải thích: sơ đồ thuật toán Bin DCT loại A, B, C (Hình 4,5,6 trong file binDCT1.pdf) và thuật toán chung của BinDCT (có hệ số scale) (hình 5 của file binDCT.pdf)
   - Viết code thực hiện BinDCT loại C thay cho DCT ở câu 1.
3. Viết chương trình thực hiện zigzag thay cho phương pháp đang dùng trong demo (không dùng hàm có sẵn).
4. Viết chương trình tính PSNR của ảnh sau khi nén (ảnh jpeg) và ảnh gốc với các chất lượng nén lần lượt là 95, 80, 50, 20 (Lập bảng so sánh PSNR khi sử dụng thuật toán BinDCT và DCT tự viết ở câu 1).

$$PSNR = 10 * log_{10}\left(\frac{MAX^2}{MSE}\right)$$

Trong đó,

$$MSE = \frac{1}{m * n}\sum_{i=0}^{m-1}\sum_{j=0}^{n-1}[I(i,j) - K(i,j)]^2$$

MAX là giá trị tối đa của một pixel = 255

Tài liệu tham khảo

[1] G. K. Wallace, "The JPEG still picture compression standard," in *IEEE Transactions on Consumer Electronics*, vol. 38, no. 1, pp. xviii-xxxiv, Feb. 1992, doi: 10.1109/30.125072. https://drive.google.com/drive/folders/1EfL5YNxazBHL5958FG39xsW04CH6BYA_?usp=drive_link

[2] Sayood, Khalid. *Introduction to data compression*. Morgan Kaufmann, 2017. https://drive.google.com/drive/folders/1EfL5YNxazBHL5958FG39xsW04CH6BYA_?usp=drive_link