ĐẠI HỌC QUỐC GIA TP. HCM

TRƯỜNG ĐẠI HỌC BÁCH KHOA

BÀI TẬP LỚN MÔN HỌC

NHẬP MÔN THỊ GIÁC MÁY TÍNH

ĐỀ TÀI

PHƯƠNG PHÁP NÉN ẢNH THEO CHUẨN JPEG

LỚP L02 - HK211

GVHD: LÊ THANH HẢI

NHÓM SINH VIÊN THỰC HIỆN

STT	MSSV	HỌ VÀ TÊN	ĐIỂM BTL
1	1910048	TRẦN HUY BẢO	
2	1910115	PHAN TRUNG ĐẠT	
3	1910132	NGUYỄN HỮU ĐÔ	
4	1911992	ĐỖ ANH TÀI	

TP. HỒ CHÍ MINH, NĂM 2021

MỤC LỤC

PHẦN MỞ ĐẦU

Hiện nay, công nghệ thông tin đang phát triển một cách nhanh chóng, liên tục đạt được những dấu mốc quan trọng trong lĩnh vực này. Với sự phát triển ấy thì một vài khái niệm, một vài quy chuẩn hay là những đối tượng tài nguyên mới cũng bắt đầu ra đời theo đó. Hơn nữa, đó cũng chính là sự thành công của con người trong công cuộc khai phá tri thức.

Dữ liệu đa phương tiện là một phần quan trọng của lĩnh vực công nghệ thông tin. Dữ liệu đa phương tiện bao gồm một hay nhiều kiểu dữ liệu phương tiện truyền thông chính như văn bản, hình ảnh, âm thanh, video, …

Trải qua tìm hiểu một thời gian, nhóm chúng em cảm thấy một đối tượng trong phạm vi của Dữ liệu đa phương tiện đã thực sự mang lại những lợi ích lớn cho người dùng trong việc lưu trữ, xử lí và truyền tải thông tin dạng hình ảnh trên mạng máy tính hay trên các thiết bị số khác, đó chính là *Phương pháp nén ảnh theo chuẩn JPEG* (Joint Photographic Experts Group). Vì vậy chúng em quyết định tìm hiểu, phân tích và đưa ra một cái nhìn mang tính cụ thể, rõ ràng hơn ở phương pháp này.

Trong quá trình thực hiện Bài tập lớn này, nhóm đã rất nỗ lực tìm hiểu, thực hành và đánh giá kết quả của chính mình. Tuy nhiên, những sơ xuất, thiếu xót là không thể tránh khỏi. Kính mong quý thầy cô tận tình chỉ bảo để giúp chúng em bổ sung và khắc phục những khuyết điểm còn tồn tại và tích lũy thêm kiến thức để chúng em tiếp tục thực hiện các đề tài sau này.

Cuối cùng, chúng em xin chân thành cảm ơn quý thầy cô trong bộ môn, đặc biệt là thầy **Lê Thanh Hải** đã tận tình chỉ bảo giúp đỡ chúng em hoàn thành nhiệm vụ của mình.

PHẦN NỘI DUNG

CHƯƠNG 1. TỔNG QUAN VỀ JPEG

1.1. Khái quát về JPEG

JPEG là viết tắt tên của nhóm nghiên cứu phát minh ra phương pháp này – Joint Photographic Experts Group và đã được quy chuẩn vào năm 1994 với tiêu chuẩn ISO/IEC 10918-1; được sử dụng để lưu trữ và truyền ảnh qua mạng internet (World Wide Web).

Hình 1.1 - Ảnh nén JPEG

Thông thường các ảnh màu hiện nay dùng 8-bit (1-byte) hay 256 màu thay cho từng mức cường độ của các màu đỏ, xanh lá cây và xanh da trời. Như thế mỗi điểm của ảnh cần 3-byte để lưu mã màu, và lượng byte một ảnh màu này chiếm gấp 24 lần ảnh trắng đen cùng cỡ. Với những ảnh này các phương pháp nén ảnh như IFF (Image File Format) theo phương pháp RLE (Run Length Encoding) không mang lại hiệu quả vì hệ số nén chỉ đạt tới 2:1 hay 3:1 (tất nhiên là kết quả nén theo phương pháp RLE phụ thuộc vào cụ thể từng loại ảnh, ví dụ như kết quả rất tốt với các loại ảnh ít đổi màu). Ưu điểm cao của phương pháp này là ảnh đã nén sau khi giải nén sẽ trùng khớp với ảnh ban đầu. Một số phương pháp nén khác không để mất thông tin như của Lempel – Ziv – Welch (LZW) có thể cho hệ số nén tới 6:1. Nhưng như thế vẫn chưa thật đáp ứng yêu cầu đòi hỏi thực tế.

Công nghệ nén ảnh theo chuẩn JPEG là một trong những công nghệ nén ảnh hiệu quả, cho phép làm việc với các ảnh có nhiều màu và kích cỡ lớn, tỉ lệ nén ảnh đạt được tới 80:1. Nhưng khuyết điểm của công nghệ này là phải chịu mất thông tin (ảnh sau khi giải nén khác với ảnh ban đầu), lượng thông tin mất mát tăng dần theo hệ số nén. Tuy nhiên, chúng ta có thể hoàn toàn kiểm soát sự mất mát này bằng cách hạn chế hệ số nén.

JPEG tiến hành sửa đổi thông tin ảnh khi nén sao cho ảnh mới gần giống như ảnh cũ khiến cho ta có thể không cảm thấy sự khác biệt rõ ràng. Như thế người dùng có thể cân nhắc giữa cái lợi của việc tiết kiệm bộ nhớ và mức độ mất thông tin của ảnh để chọn phương án thích hợp.

1.2. Ưu – nhược điểm của JPEG

Ưu điểm:

JPEG cho phép nén ảnh với tỉ số nén lên đến 80:1 hoặc cao hơn, hiển thị các hình ảnh đầy đủ màu, cho định dạng di động mà kích thước file lại nhỏ hơn. JPEG cũng được sử dụng rất nhiều trên mạng internet. Lợi ích chính của chúng là chúng có thể hiển thị các hình ảnh với màu chính xác, điều đó cho phép chúng được sử dụng tốt nhất cho các hình ảnh chụp và hình ảnh minh họa có số lượng màu lớn.

Hình 1.2 - JPEG vs PNG

Nhược điểm:

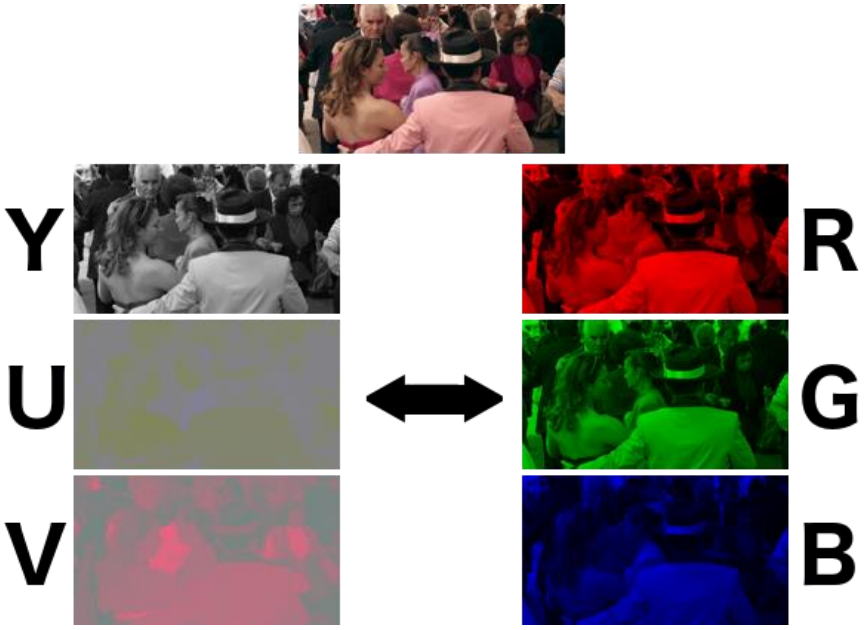
Nhược điểm chính của JPEG là chúng được nén bằng thuật toán Lossy (mất dữ liệu). Điều này có nghĩa rằng hình ảnh của bạn sẽ bị mất một số thông tin khi chuyển sang định dạng JPEG. Đường bao giữa các khối màu có thể xuất hiện nhiều điểm mờ và các vùng sẽ mất sự rõ nét, tỉ số nén càng cao thì sự mất mát thông tin trên ảnh JPEG càng lớn. Nói một cách khác, định dạng JPEG thực hiện bảo quản tất cả thông tin màu trong hình ảnh đó, tuy nhiên với các hình ảnh chất lượng màu cao như hình ảnh chụp thì điều này sẽ không hề hấn gì. Các ảnh JPEG không thể làm trong suốt hoặc chuyển động – trong trường hợp này bạn sẽ sử dụng định dạng GIF (hoặc định dạng PNG để tạo trong suốt).

1.3. Một vài tiêu chuẩn JPEG

1.3.1. JPEG 1992

Bài tập lớn này sẽ trình bày về phương pháp JPEG 1992. Đây là loại JPEG tiêu chuẩn. Phương pháp nén ảnh dựa trên nguyên lý sau:

Đầu tiên, ảnh màu trong không gian của ba màu RGB (Red Green Blue) được biến đổi về hệ YUV (hay YCbCr). Hệ YUV là kết quả nghiên cứu của các nhà sản xuất vô tuyến truyền hình hệ Pal, Secam và NTSC, nhận thấy tín hiệu video có thể phân ra 3 thành phần Y, U, V (cũng như phân theo màu chuẩn đỏ, xanh lá cây và xanh da trời). Và một điều thú vị là thị giác của con người rất nhạy cảm với thành phần Y và kém nhạy cảm với hai loại U và V. Phương pháp JPEG đã nắm bắt phát hiện này để tách những thông tin thừa của ảnh. Hệ thống nén thành phần Y của ảnh với mức độ ít hơn so với U và V.



Hình 1.3 - YUV vs RGB

Tiếp theo, biến đổi những vùng (block) thể hiện dùng biến đổi cosine rời rạc (thông thường là những vùng 8x8 pixel). Khi đó thông tin về 64-pixel ban đầu sẽ biến đổi thành ma trận có 64 hệ số thể hiện "thực trạng" các pixel. Điều quan trọng là ở đây hệ số đầu tiên có khả năng thể hiện "thực trạng" cao nhất, khả năng đó giảm rất nhanh với các hệ số khác. Nói cách khác thì lượng thông tin của 64-pixel tập trung chủ yếu ở một số hệ số ma trận theo biến đổi trên. Trong giai đoạn này có sự mất mát thông tin, bởi không có biến đổi ngược chính xác. Nhưng lượng thông tin bị mất này chưa đáng kể so với giai đoạn tiếp theo. Ma trận nhận được sau biến đổi cosine rời rạc được lược bớt sự khác nhau giữa các hệ số. Đây chính là lúc mất nhiều thông tin vì thuật toán sẽ vứt bỏ những

thay đổi nhỏ của các hệ số. Như thế khi giải nén ảnh đã nén, chúng ta sẽ có được những tham số khác của các pixel. Các biến đổi trên áp dụng cho thành phần U và V của ảnh với mức độ cao hơn so với Y (mất nhiều thông tin của U và V hơn). Sau đó thì áp dụng phương pháp mã hóa của Huffman: Phân tích dãy số, các phần tử lặp lại nhiều được mã hóa bằng ký hiệu ngắn (marker). Khi giải nén ảnh, ta chỉ việc làm lại các bước trên theo quá trình ngược lại cùng với các biến đổi ngược.

Vì phương pháp này thực hiện với các vùng ảnh (thông thường là 8x8 pixel) nên hay xuất hiện sự mất mát thông tin trên vùng biên của các vùng (block) này. Hiện nay con người đã giải quyết vấn đề này bằng cách làm trơn ảnh sau khi bung nén để che lấp sự khác biệt của biên giới giữa các block. Một hệ nén ảnh theo chuẩn JPEG cùng thuật toán làm trơn ảnh đã được công ty ASDG đưa ra trong hệ Art Department Professional.

1.3.2. LS – JPEG

LS – JPEG (Lossless JPEG) là một bổ sung cho JPEG vào năm 1993, bằng cách sử dụng một hệ thống dự báo được sắp xếp dựa trên ba điểm lân cận (upper, left và upper-left) và entropy mã hóa dựa trên các lỗi dự báo.

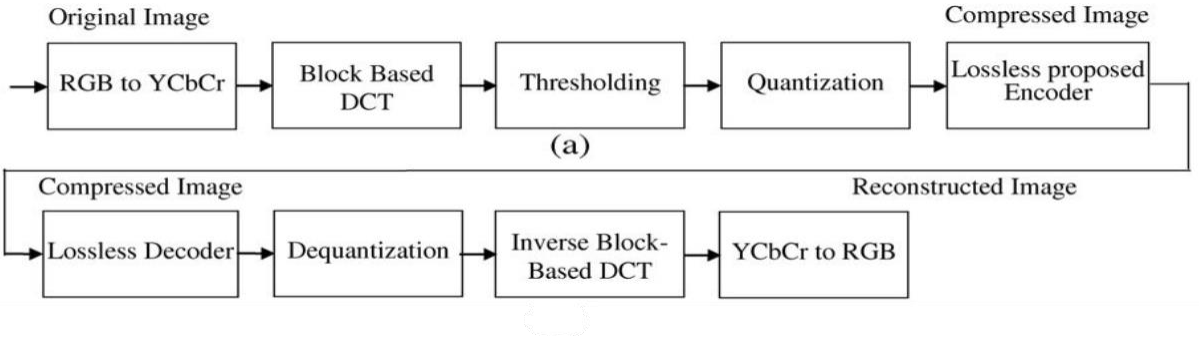
1.3.3. JPEG Search

Ngày nay, nhiều định dạng siêu dữ liệu khác nhau tồn tại để mô tả hình ảnh nhưng vẫn còn nhiều vấn đề trong khả năng tương tác. Trong bối cảnh đó, trọng tâm chính của JPEG Search là cung cấp một khả năng tương tác tốt hơn trong tìm kiếm hình ảnh.

1.3.4. JPEG XR

Là một định dạng hình ảnh cung cấp một số cải tiến so với JPEG như khả năng nén tốt hơn, chất lượng màu tốt hơn, giảm bớt vùng nén ảnh và nén không mất mát.

CHƯƠNG 2. KỸ THUẬT JPEG



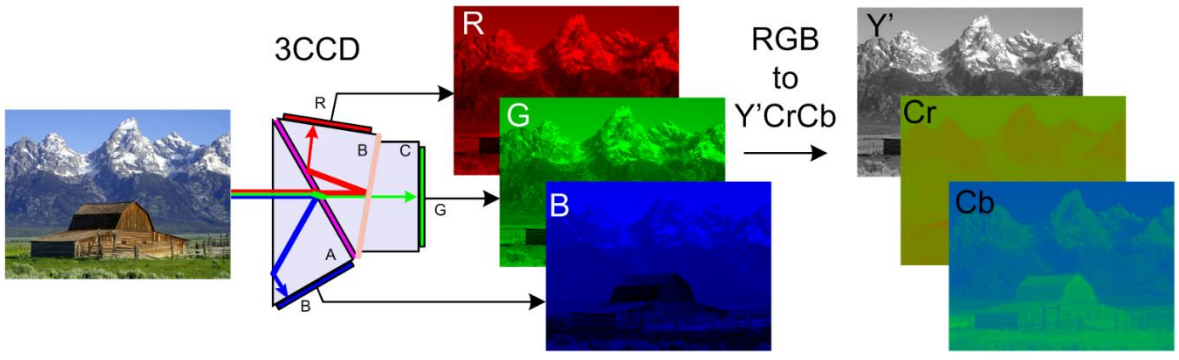
Hình 2.1 - Trình tự thực hiện JPEG

2.1. Quá trình nén ảnh

Các kỹ thuật nén ảnh hưởng tới việc giải quyết bài toán giảm khối lượng thông tin cần thiết để mô tả ảnh số. Nền tảng của quá trình nén là loại bỏ dư thừa có trong tín hiệu. Phương pháp nén hiệu quả nhất thường sử dụng các biến đổi toán học để biến ma trận các điểm ảnh trong không gian hai chiều sang một không gian hai chiều khác, nơi mức độ tương quan giữa các hệ số biến đổi mới nhỏ hơn. Như chúng ta biết, độ dư thừa trong tín hiệu ảnh số phụ thuộc vào mức độ tương quan giữa các điểm ảnh, độ tương quan lớn thì độ dư thừa cũng lớn.

2.1.1. Xử lý màu

Biến đổi ảnh từ không gian màu RGB sang YCbCr để tăng thành phần độ chói, giảm các thành phần màu sắc (do mắt người nhạy cảm với độ sáng hơn so với màu sắc).



Hình 2.2 - Chuyển kênh màu

Công thức biến đổi như sau:

$$Y' = 16 + \frac{65.738 \cdot {R'}_D}{256} + \frac{129.057 \cdot {G'}_D}{256} + \frac{25.064 \cdot {B'}_D}{256}$$

$$C_b = 128 - \frac{37.945 \cdot {R'}_D}{256} - \frac{74.494 \cdot {G'}_D}{256} + \frac{112.439 \cdot {B'}_D}{256}$$

$$C_r = 128 + \frac{112.439 \cdot {R'}_D}{256} - \frac{94.154 \cdot {G'}_D}{256} - \frac{18.285 \cdot {B'}_D}{256}$$

Thực chất quá trình này là không bắt buộc. Tuy nhiên các nhà nghiên cứu khuyến khích nên thực hiện bước này trước tiên để kết quả nén được cao hơn, tốt hơn.

2.1.2. Resize

Ảnh màu được chia thành các khối 8×8, khối màu là một đơn vị dữ liệu.

Mục đích chuyển thành các khối 8×8 để giảm thời gian tính toán, tăng khả năng chính xác khi tính toán. Do các điểm ảnh lân cận có độ tương quan cao, do đó phép biến đổi DCT cho từng khối nhỏ sẽ tập trung năng lượng vào một số ít các hệ biến đổi. Việc loại bỏ một số năng lượng thấp trong các khối chỉ tạo ra mất mát thông tin cục bộ giúp nâng cao chất lượng hình ảnh.

2.1.3. Quá trình DCT (Discrete Cosine Transform)

DCT là phép biến đổi Cosine rời rạc để chuyển tín hiệu từ miền thời gian hay không gian sang miền tần số. Đặc điểm của phép biến đổi này là tín hiệu ảnh trong miền không gian chuyển sang miền tần số thì các thành phần DC và các thành phần AC mang hầu hết các thông tin chứa trong ảnh gốc. Trong đó, DC là thành phần quan trọng nhất mang độ chói trung bình của ảnh, các thành phần AC chứa các thông tin về chi tiết của ảnh. Sau đó, khi qua tầng lượng tử hoá, các hệ số ít quan trọng sẽ bị loại bỏ bớt và chỉ giữ lại một số hệ số hệ số đầu tiên gọi là hệ số DCT. Vai trò chủ yếu của phương pháp DCT là giảm độ dư thừa dữ liệu trong pixel ở miền tần số cao. Bởi vì bất kì một giá trị pixel nào đó cũng có thể dự đoán từ các giá trị pixel lân cận của nó nên thông tin từ các pixel riêng lẻ là tương đối nhỏ. Để giảm độ dư thừa của các pixel đặc biệt là các pixel ở miền tần số cao thì phương pháp DCT là vô cùng thích hợp. Không những vậy, sau khi biến đổi DCT thì hàm giải tương quan giảm đi một cách đáng kể. Chính vì vậy mà hiệu suất nén đạt được tỉ số nén cao. Điều quan trọng là khối DCT đóng vai trò quan trọng trong quá

trình lượng tử hóa khi thiết kế hệ thống nén vì nó ảnh hưởng trực tiếp đến việc cho lại chất lượng ảnh khôi phục tốt hay xấu. Có thể nói chính điều đó đã làm lên một chuẩn JPEG được ứng dựng rộng rãi như vậy.

Mục đích:

Thứ nhất, chuyển từ miền không gian sang miền tần số. Tín hiệu sẽ tập trung ở miền tần số thấp.

Thứ hai, DCT làm giảm độ tương quan không gian block. Điều đó cho phép biểu diễn thích hợp ở miền DCT do các hệ số DCT có xu hướng có phần dư thừa ít hơn. Điều này có nghĩa là DCT gói một phần lớn năng lượng tín hiệu vào các thành phần có thành phần tần số tương ứng để lưu trữ hoặc truyền dẫn tạo 0 hoặc các giá trị thấp đối với các thành phần tần số cao, nhờ đặc tính của mắt người, các hệ số DCT có thể mã hóa phù hợp. Chỉ các hệ số DCT quan trọng nhất mới được mã hóa và truyền đi. DCT thuận kết hợp với ảnh đầu vào được mã hóa bằng các mẫu dài 8-bit. Nếu hệ số được lượng tử hóa bằng 11-bit thì nén sẽ tổn hao. Sau khi thực hiện DCT năng lượng thấp tập trung chủ yếu miền tần số thấp.

Quá trình DCT là một trong những công đoạn chính trong JPEG. Nhiệm vụ của nó là tập trung năng lượng vào một số các giá trị để giải tương quan tốt nhất, nhằm nâng cao tỉ số nén.

Vì phép biến đổi DCT làm việc với các giá trị pixel từ -128 đến 127 nên giá trị pixel của 3 kênh trong ảnh gốc $(0-255)$ cần trừ đi cho 128. Sau đó thực hiện biến đổi DCT hai chiều cho 3 kênh theo công thức:

$$F(u,v) = \frac{1}{8}\sum_{j=0}^{7}\sum_{k=0}^{7} f(j,k)\cos\frac{(2j+1)u\pi}{16}\cos\frac{(2k+1)v\pi}{16} \quad \text{nếu } u,v = 0$$

$$F(u,v) = \frac{1}{4}\sum_{j=0}^{7}\sum_{k=0}^{7} f(j,k)\cos\frac{(2j+1)u\pi}{16}\cos\frac{(2k+1)v\pi}{16} \quad \text{nếu } u,v \neq 0$$

trong đó, $F(u,v)$ – các hệ số của khối DCT 8×8

$f(j,k)$ – các mẫu gốc trong khối 8×8 pixel

Phép biến đổi này thực hiện cho mỗi khối 8×8 của ảnh. Để thực hiện, ta "đánh số" cho mỗi khối 8×8, mỗi khối block có một toạ độ nhất định, ta thực hiện quét lần lượt tới từng khối, trong mỗi khối đó ta tiến hành biến đổi DCT, cứ như thế cho đến toàn bộ ảnh. Vì có rất nhiều khối block trong một bức ảnh nên ta cần tìm cách chuyển các chỉ số cục bộ trong công thức sang giá trị toàn cục của một bức ảnh để truy xuất đến giá trị của các pixel mà không làm mất tính tổng quát của công thức.

Kết quả phép tính bằng 8 lần giá trị pixel trung bình trong khối. Do đó hệ số thứ nhất được gọi là hệ số DC. Các hệ số khác, dưới giá trị thành phần một chiều, biểu diễn các tần số cao hơn theo chiều dọc. Các hệ số ở về phía bên phải của thành phần một chiều biểu thị các tần số cao hon theo chiều ngang. Hệ số trên cùng ở cận phải (0,7) sẽ đặc trưng cho tín hiệu có tần số cao nhất theo phương nằm ngang của ma trận 8×8, và hệ số hàng cuối bên trái (7,0) sẽ đặc trưng cho tín hiệu có tần số cao nhất theo phương thẳng đứng. Còn các hệ số khác ứng với những phối hợp khác nhau của các tần số theo chiều dọc và chiều ngang.

Ta cần chú ý rằng bản thân biến đổi DCT không làm mất thông tin vì DCT là một biến đổi tuyến tính chuyển các giá trị của điểm ảnh từ miền không gian thành các hệ số trong miền tần số. Nếu biến đổi DCT thuận và nghịch được tính toán với độ chính xác tuyệt đối và nếu các hệ số DCT không phải qua bước lượng tử và mã hoá thì ảnh thu được sau biến đổi DCT ngược sẽ giống hệt ảnh gốc.

2.1.4. Lượng tử hoá

Quá trình lượng tử hoá được coi như việc chia các hệ số DCT cho bước nhảy lượng tử tương ứng, kết quả được làm tròn xuống số nguyên gần nhất.

Nhiệm vụ: Mã hoá ma trận đầu vào sau khi biến đổi DCT thành các giá trị mức xám đặc trưng cho cường độ sáng.

Công thức:

$$F_q(u,v) = round\left[\frac{F(u,v)}{Q(u,v)}\right]$$

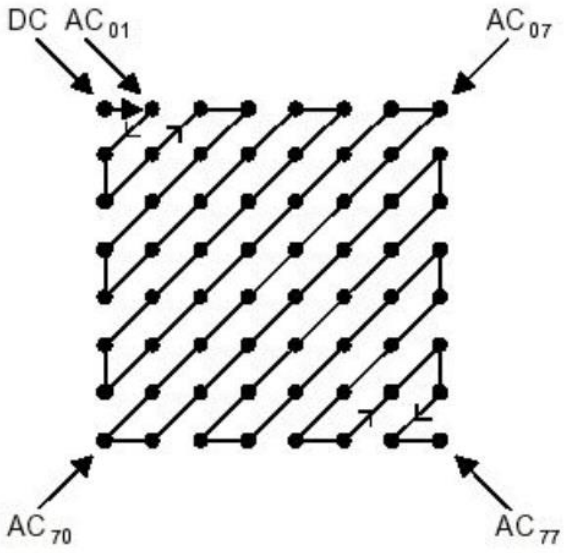
Bảng lượng tử $Q(u,v)$ thông dụng:

$$\begin{bmatrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\ 14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\ 24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\ 49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\ 72 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \end{bmatrix}$$

2.1.5. Quét zigzag

Mã hóa entropy được sử dụng để làm giảm sự dư thừa thống kê trong khối hệ số lượng tử hóa. Các dữ liệu hình ảnh ban đầu sẽ được biểu diễn bởi một chuỗi các từ mã nhị phân sau quá trình mã hóa. Mã hóa entropy dựa trên mã hóa Huffman được sử dụng trong hệ thống JPEG cho mã hóa hệ số AC. Trong hệ thống nén hình ảnh tích hợp của cơ sở ghép DPCM/DCT, tất cả các hệ số AC và DC được xử lý trong cùng một cách.

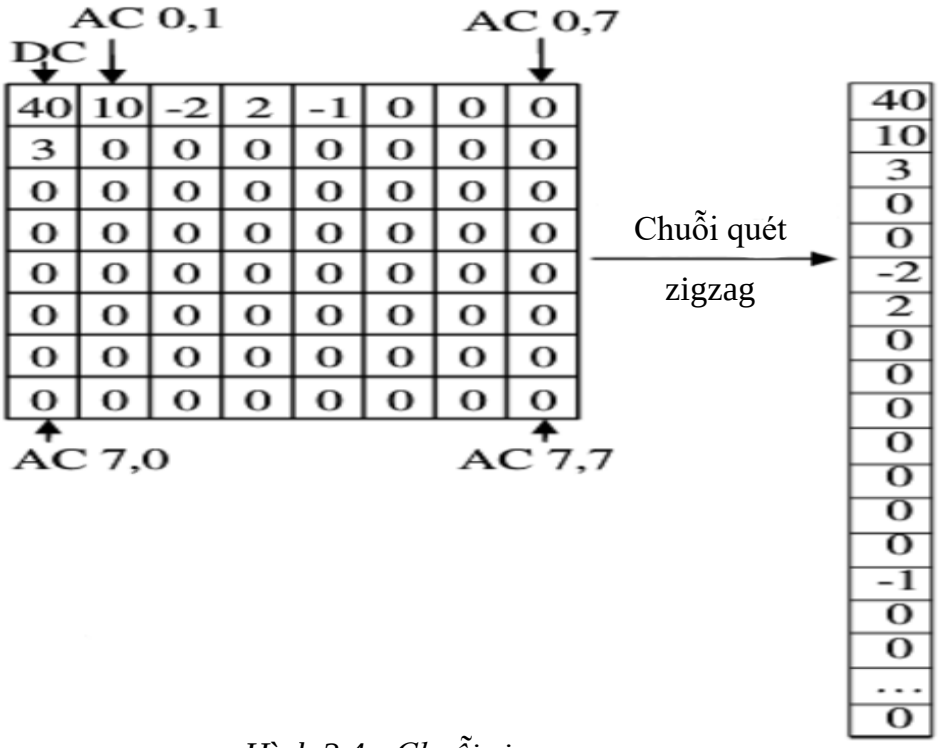
Để mã hóa entropy trong hệ thống JPEG các hệ số được lượng tử hóa bàng cách biến đổi các hệ số của mảng hai chiều thành chuỗi số một chiều bàng cách sử dụng quét zigzag. Trong trường hợp này yếu tố đầu tiên là quét hệ số DC, kết quả sẽ là một chuỗi có độ dài 64. Kết quả này được sắp xếp thành chuỗi hệ số tăng dần với tăng tần số không gian, tính chất của hệ số lượng tử có tính năng chặn các hài bậc cao hoặc không có ý nghĩa trong không gian và chỉ số quét zigzag sẽ làm tăng khả năng xuất hiện các hệ số có giá trị 0. Việc lập sơ đồ quét zigzag các hệ số lượng tử hóa với tập hợp các ký hiệu ở đầu vào cho bảng mã Huffman. Mỗi ký hiệu biểu diễn một giá trị khác 0 được tạo ra bàng cách kết hợp các hệ số có giá trị 0 thay bằng các chữ số 0 xuất hiện. Độ lớn của hệ số a khác 0.



Hình 2.3 - Quét zigzag

Ví dụ: Sau khi lượng tử hóa, chúng ta kết thúc với rất nhiều số 0 ở tần số cao. Ta có thể lưu trữ thông tin này hiệu quả hơn bằng cách sắp xếp lại các thông tin này theo thứ tự zigzag từ trên cùng bên trái đến dưới

cùng bên phải. Bằng cách này, ta có thể nhóm các số không lại với nhau và lưu trữ chúng dưới dạng dải ô như thế này:



Hình 2.4 - Chuỗi zigzag

2.1.6. Mã hoá Huffman (entropy)

Phương pháp mã hóa Huffman là phương pháp dựa vào mô hình thống kê. Người ta tính tần suất xuất hiện của các ký tự bằng cách duyệt tuần tự từ đầu tệp gốc đến cuối tệp. Việc xử lý ở đây tính theo bit. Trong phương pháp này các ký tự có tần suất cao một từ mã ngắn, các ký tự có tần suất thấp một từ mã dài. Như vậy với cách thức này ta đã làm giảm chiều dài trung bình của từ mã hoá bằng cách dùng chiều dài biến đổi tuy nhiên cũng có trường hợp bị mất một ít bit khi tần suất là rất thấp.

Thuật toán gồm hai giai đoạn:

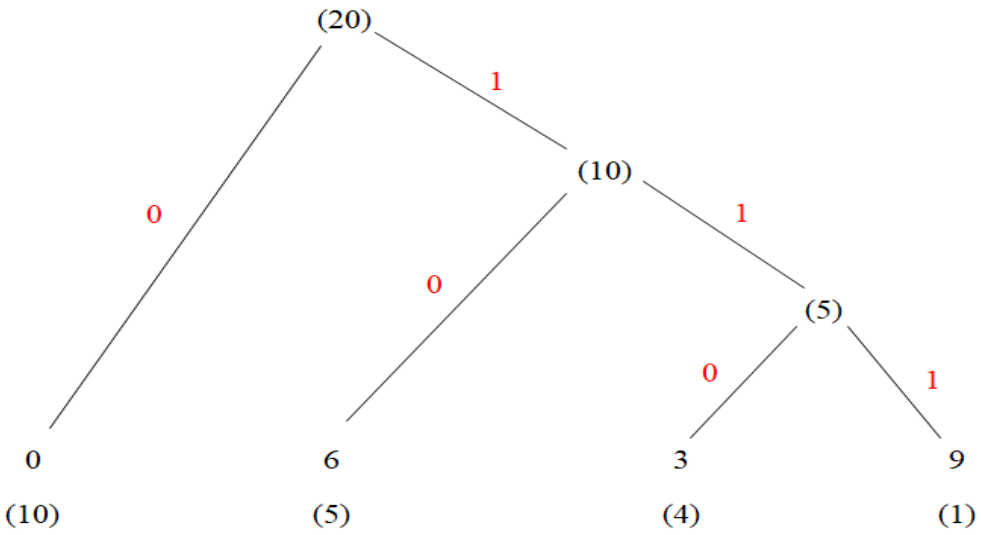
Giai đoạn đầu tiên, tính tần suất của các ký tự trong dữ liệu gốc: duyệt tệp gốc một cách tuần tự từ đầu đến cuối để xây dựng bảng mã. Tiếp sau đó là sắp xếp lại bảng mã theo thứ tự tần suất giảm dần.

Giai đoạn thứ hai, mã hóa: duyệt bảng tần suất từ cuối lên đầu để thực hiện ghép hai phần tử có tần suất xuất hiện thấp nhất thành một phần tử duy nhất. Phần tử này có tần

suất bằng tổng hai tần suất thành phần. Tiến hành cập nhật lại bảng và đương nhiên loại bỏ hai phần tử đã xét. Quá trình được lặp lại cho đến khi bảng chỉ có một phần tử. Quá trình này gọi là quá trình tạo cây mã Huffman vì việc tập hợp được tiến hành nhờ một cây nhị phân hai nhánh. Phần tử có tần suất thấp ở bên phải, phần tử kia ở bên trái. Với cách tạo cây này, tất cả các bit dữ liệu/ký tự là nút lá; các nút trong là các nút tổng hợp. Sau khi cây đã tạo xong, người ta tiến hành gán mã cho các nút lá. Việc mã hóa rất đơn giản: mỗi lần xuống bên phải ta thêm một bit "1" vào từ mã; mỗi lần xuống bên trái ta thêm một bit "0". Tất nhiên có thể làm ngược lại, chỉ có giá trên mã thay đổi còn tổng chiều dài là không đổi. Cũng chính do lý do này mà cây có tên gọi là cây mã Huffman như trên đã gọi. Quá trình giải mã diễn ra ngược lại với quá trình mã hoá.

Ví dụ: Chuỗi nguồn: 00000000006666693333 có kích thước $= 20 * 8 = 160$-bit

Xây dựng cây Huffman:



0: 0 6: 10 3: 110 9: 111

Sử dụng thuật toán Huffman lập nên cây Huffman như trên, ta mã hóa chuỗi nguồn và kết quả thu được sau khi mã hóa có kích thước $= 10 * 1 + 5 * 2 + 1 * 3 + 4 * 3 =$ 35-bit (vì giá trị 0 xuất hiện mười lần nhưng chỉ dùng 1-bit để thể hiện, giá trị 6 xuất hiện năm lần dùng 2-bit để thể hiện, giá trị 9 xuất hiện một lần dùng 3-bit và giá trị 3 xuất hiện bốn lần dùng 3-bit để thể hiện)

Vậy tỉ số nén = $160/35 = 3.57$

Trong phương pháp mã hoá Huffman, mã của ký tự là duy nhất và không mã nào là phần bắt đầu của mã trước. Vì vậy khi đọc theo từng bit từ đầu đến cuối tệp nén ta có thể duyệt cây mã cho đến một lá, tức là ký tự đà được giải mã. Việc giải mã chắc chắn phải sử dụng cây nhị phân giống như trong mã hoá. Để đọc, giải mã được yêu cầu phải sử dụng theo đúng tiêu chuẩn nhất định.

***Trong xử lý ảnh, nén ảnh JPEG lũy tiến**,* việc mã hóa sau khi quét zigzag được thực hiện theo 2 bước: mã hóa thành phần DC và mã hóa thành phần AC.

- *Mã hóa thành phần DC:*

Các hệ số DC là giá trị trung bình của các khối ảnh 8×8. Độ chói trung bình của các block ảnh gần nhau thường ít biến đổi, do đó trong chuẩn nén JPEG, các hệ số DC được mã hóa theo phương pháp DPCM. Để tăng hiệu suất nén, kết quả nhận được sau đó được mã hóa tiếp bằng mã Huffman. Hệ số DC của các block DCT được lần lượt đưa tới bộ DPCM. Thành phần sai số giữa hai hệ số DC liên tiếp sẽ được mã hóa trong bộ mã Huffman. Quá trình mã hóa Huffman được thực hiện cho thành phần DC như sau:

1 - dò tìm trong bảng phân loại để tìm "loại" của giá trị ΔDC ("loại" chính là chiều dài từ mã dùng để mã hóa thành phần ΔDC)

2 - dùng bảng mã Huffman cho thành phần DC để tìm ra từ mã cho "loại" ΔDC tìm ở bước 1

3 - mã hóa nhị phân giá trị ΔDC để có được từ mã cho thành phần DC.

Trên *Hình 2.5* và *Hình 2.6* là các bảng tra cần thiết để mã hóa thành phần DC.

Loại	Phạm vi hệ số	
NA	0	
1	-1	1
2	-3, -2	2, 3
3	-7,-6,-5,-4	4,5,6,7
4	-15,…,-8	8,…,15
5	-31,…,-16	16,…,31
6	-63,…,-32	32,…63
7	-126,…,-64	64,…127
8	-255,…,-128	128,…,255
9	-511,…,-256	256,…,511
10	-1023,…,-512	512,…,1023
11	-2047,…,-1024	1024,…,2047

Hình 2.5 - Bảng phân loại các hệ số AC và DC

Các hệ số DC sai lệch	Phân loại	Từ mã
-255,…,-128 ; 128,…,255	8	1111110
-126,…,-64 ; 64,…127	7	111110
-63,…,-32 ; 32,…,63	6	11110
-31,…,-16 ; 16,…,31	5	1110
-15,…,-8 ; 8,…,15	4	110
-7,-6,-5,-4 ; 4,5,6,7	3	101
-3, -2 ; 2, 3	2	01
-1 ; 1	1	00
0	0	100

Hình 2.6 - Bảng mã Huffman cho thành phần DC

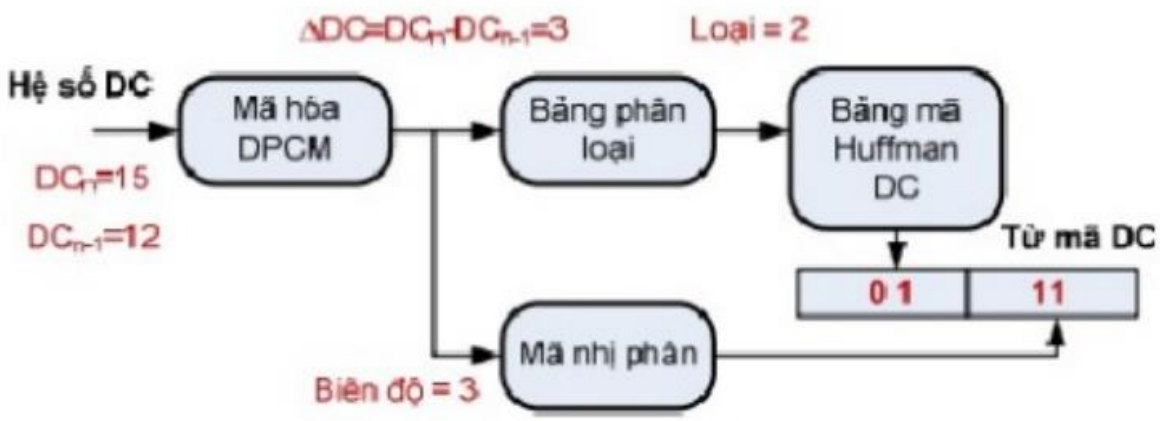
Ta có ví dụ sau: Thành phần DC trong block trên hình dưới đây có giá trị bằng 15

15	0	-1	0	0	0	0	0
-2	-1	0	0	0	0	0	0
-1	-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

Hình 2.7 - Block hệ số DCT cần được mã hóa

Ta giả sử thành phần DC của block trước đó có giá trị là 12, như vậy kết quả mã hóa DPCM sẽ có giá trị là 3, tra trên *Hình 2.5*, ta có giá trị 3 sẽ thuộc "loại" 2. Dựa vào bảng mã Huffman (*Hình 2.6*) ta có mã tương ứng với loại 2 là (01) → 2 chính là độ dài từ mã, giá trị = 3 được biểu diễn bằng chuỗi nhị phân là (11), như vậy thành phần DC sau khi mã hóa sẽ là: 0111

Sơ đồ bộ mã hóa thành phần DC cho ví dụ trên:



Sơ đồ bộ mã hóa DC

Mã hóa thành phần AC:

Chuỗi các hệ số thành phần AC được đưa vào bộ mã hóa RLC. Ở đầu ra ta nhận được các từ mã bao gồm hai thành phần:

(1) - Giá trị chạy là số lượng bit "0" liên tục đứng trước hệ số khác "0" đang được mã hóa.

(2) - biên độ các hệ số khác “0” nói trên.

Từ mã Huffman ứng với mỗi cặp giá trị trên được tìm ra trong bảng phân loại *Hình 2.5* và bảng mã Huffman cho thành phần AC (*Hình 2.8).* Từ mã AC bao gồm cho từ mã Huffman và giá trị biên độ (nhị phân) của hệ số AC.

Giá trị chạy	Loại	Độ dài mã	Từ mã
0	1	2	00
0	2	2	01
1	1	4	1100
1	2	6	111001
1	3	7	1111001
1	4	9	111110110
2	1	5	11011
2	2	8	11111000
4	1	6	111011
5	1	7	1111010
6	1	7	1111011
EOB		4	1010

Hình 2.8 - Bảng mã Huffman cho thành phần AC

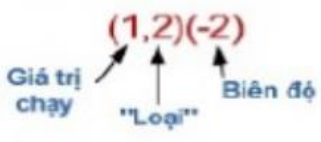
Ví dụ: Sau quá trình zigzag, từ block hệ số DCT như trên *Hình 2.7*, ta nhận được chuỗi các hệ số AC sau: 0, -2, -1, -1, -1, 0, 0, -1, 0, 0, …

Chuỗi bit nhận được từ bộ mã RLC là: (1,-2); (0,-1); (0,-1); (0,-1); (2,-1); (EOB).

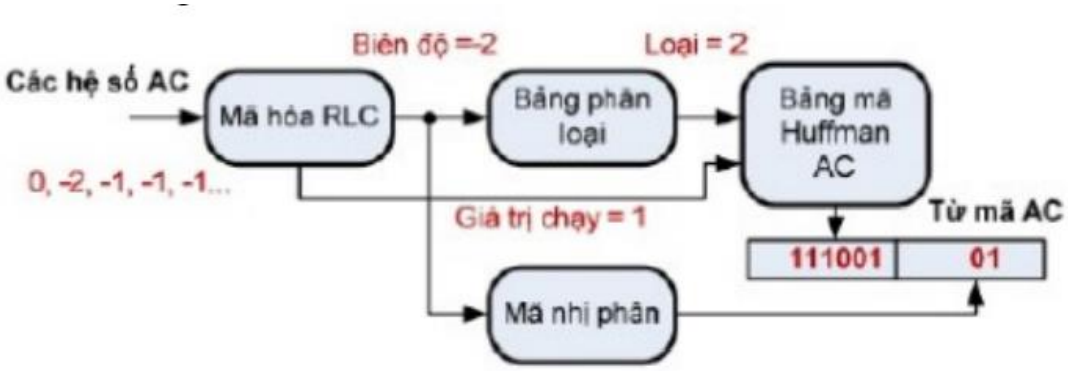
Sau hệ số khác “0” cuối cùng, chúng ta đặt từ mã đặc biệt để báo hiệu kết thúc khối, từ mã này có tên là EOB – End of Block.

Sử dụng bảng phân loại *Hình 2.5* chúng ta tìm được loại của biên độ các hệ số, tín hiệu được đưa vào mã Huffman có cấu trúc như sau:

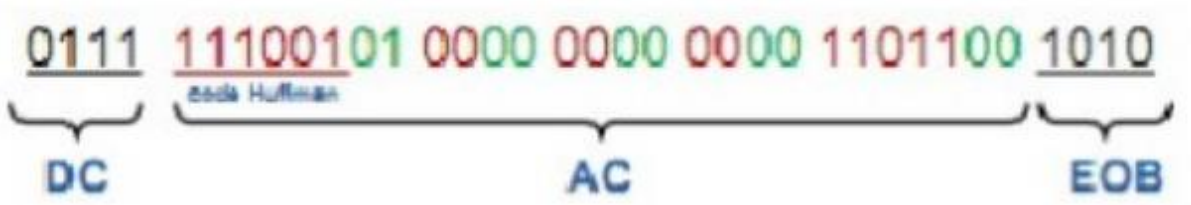
(1,2)(-2) ; (0,1)(-1) ; (0,1)(-1) ; (0,1)(-1) ; (2,1)(-1) ; (0,0). Ý nghĩa của các giá trị trong chuỗi được giải thích như sau:



Ta xét (1,2)(-2) : Với Giá trị chạy là 1 và "Loại" của Biên độ là 2, trang bảng *Hình 2.8* ta có được từ mã là: 111001. Biên độ là (-2) ta tra bảng *Hình 2.6* ta được từ mã là 01 → (1,2)(-2) được mã hóa thành 11100101. Ta có sơ đồ mã hóa thành phần AC:



Tương tự đối với giá trị còn lại trong chuỗi và sau đó tập hợp kết quả mã hóa các thành phần AC và DC ta thu được kết quả là chuỗi bit có dạng sau:



Có thể thấy rằng chỉ cần 35-bits để truyền đi block 64 điểm ảnh, như vậy hiệu quả nén của phương pháp JPEG trong trường hợp này là 0.5-bit/điểm ảnh.

2.2. Quá trình giải nén ảnh

Tất nhiên, ta sẽ giải nén ảnh bằng cách đi ngược lại trình tự, công thức của quá trình nén ảnh đã nêu ra ở trên.

2.2.1. Giải mã quá trình Huffman

Như đã đề cập, quá trình giải mã Huffman thực hiện ngược lại với quá trình mã hóa Huffman, người ta cũng phải dựa vào bảng mã tạo ra trong giai đoạn nén (bảng này được giữ lại trong cấu trúc của tệp nén cùng với dữ liệu nén). Do đó để giải mã chuỗi nhị phân

đã mã hóa, chúng ta cần có cây Huffman (cây nhị phân mã hóa chứa giá trị mã hóa của từng kí tự) hoặc bảng mã đã được lưu giữ lại. Sau đó, duyệt tuần tự qua chuỗi bit mã hóa theo các bước sau:

(1) Duyệt qua từng bit của chuỗi bit mã hóa từ nút gốc đến khi gặp nút lá.

(2) Nếu bit hiện tại là bit 0 thì chuyển qua nhánh trái.

(3) Nếu bit hiện tại là bit 1 thì chuyển qua nhánh phải.

(4) Khi gặp nút lá, ta thêm ký tự ở nút lá vào chuỗi kết quả. Sau đó trở lại nút gốc cho bit tiếp theo.

2.2.2. Quét zigzag ngược

Ta thực hiện quá trình quét zigzag ngược lại với chiều ban đầu: điểm đầu là điểm có toạ độ (7,7) và điểm cuối là điểm (0,0)

2.2.3. Giải mã quá trình lượng tử hoá

Là phép toán ngược của phép lượng tử hóa ở trên, công thức của quá trình nghịch lượng tử hóa:

$$F(u,v) = F_q(u,v).Q(u,v)$$

2.2.4. DCT nghịch

Tái tạo lại các giá trị mẫu $f(j,k)$ trên cơ sở các hệ số $F(u,v)$ theo công thức:

$$f(j,k) = \sum_{u=0}^{7}\sum_{v=0}^{7}\frac{C(u)C(v)}{4}F(u,v)\cos\frac{(2u+1)u\pi}{16}\cos\frac{(2v+1)v\pi}{16}$$

trong đó, $C(u), C(v) = \begin{cases} 1/\sqrt{2}, & u,v = 0 \\ 1, & u,v \neq 0 \end{cases}$

2.2.5. Xử lý màu

Thực hiện chuyển đổi ảnh từ YCbCr về lại ảnh RGB:

$$R'_D = \frac{298.082 \cdot Y'}{256} + \frac{408.583 \cdot C_r}{256} - 222.921$$

$$G'_D = \frac{298.082 \cdot Y'}{256} - \frac{100.291 \cdot C_b}{256} + \frac{208.120 \cdot C_r}{256} + 135.576$$

$$B'_D = \frac{298.082 \cdot Y'}{256} - \frac{516.412 \cdot C_b}{256} - 276.836$$

2.2.6. Tái tạo lại kích thước ban đầu

Quá trình này rất đơn giản, ta chỉ việc thực hiện ngược lại với quá trình Resize để ảnh trở về kích thước ban đầu. Đây chính là kết quả cuối cùng mà chúng ta thu được, cũng chính là bức ảnh mà người bên kia mạng Internet nhận được sau khi chúng ta gửi ảnh có sử dụng kĩ thuật JPEG.

CHƯƠNG 3. TIÊU CHUẨN ĐÁNH GIÁ CHẤT LƯỢNG HÌNH ẢNH

Quá trình nén ảnh thường đi đôi với việc ảnh nén bị biến dạng so với ảnh gốc. Vì vậy cần xác định tiêu chí và phương pháp đánh giá một cách khách quan lượng thông tin về ảnh đã bị mất đi sau khi nén. Có thể đánh giá mức độ sai số giữa hai ảnh thông qua mức sai lệch trung bình bình phương – RMS (Root Mean Square). Cho $f(x,y)$ là ảnh gốc, $f'(x,y)$ là ảnh khôi phục sau khi nén. Khác biệt tuyệt đối giữa hai ảnh là: $e(x,y) = f(x,y) - f'(x,y)$.

Sai số trung bình được tính theo công thức:

$$e_{RMS} = \left[\frac{1}{MN}\sum_{x=0}^{M-1}\sum_{y=0}^{N-1}(f(x,y) - f'(x,y))^2\right]^{\frac{1}{2}}$$

Thông thường, khi giá trị RMS thấp thì chất lượng ảnh nén sẽ càng tốt. Tuy nhiên trong một số trường hợp chất lượng hình ảnh nén không nhất thiết phải tỉ lệ thuận với giá trị RMS. Một phương pháp đánh giá chất lượng ảnh nén khác dựa trên tỉ lệ tín hiệu/nhiễu được tính theo công thức sau:

$$SNR = \left[\frac{\sum_{x=0}^{M-1}\sum_{y=0}^{N-1}(f'(x,y))^2}{\sum_{x=0}^{M-1}\sum_{y=0}^{N-1}(f(x,y) - f'(x,y))^2}\right]^{\frac{1}{2}}$$

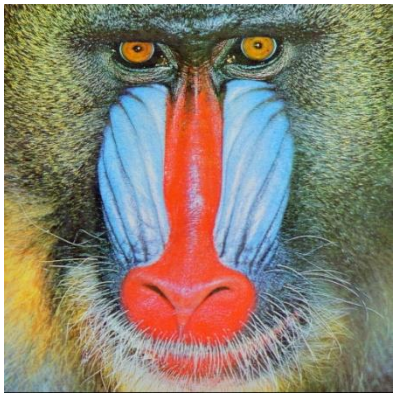
SNR (Single to Noise Ratio) – tỉ lệ tín hiệu/nhiễu

CHƯƠNG 4. THỰC HÀNH

Quá trình thực hành và thử nghiệm kết quả được chúng em viết trên ngôn ngữ lập trình C#.

Những ảnh nguồn mà nhóm sử dụng:





Hình 4.1 - Ảnh nguồn

4.1. Quá trình nén ảnh

4.1.1. Xử lý màu

Code cho thuật toán:

```
void RGB2YCbCr(Image<Ycc, byte> b, Image<Bgr, Byte> a)
    {
        int i, j;
        byte Y, Cb, Cr;      //Khai báo 3 biến để lưu giá trị 3 kênh màu
        for (i = 0; i < img.Height; i++)       //Vòng lặp quét toàn bộ ảnh
        {
            for (j = 0; j < img.Width; j++)
            {
                Y = (byte)(65.738 / 256 * a.Data[i, j, 2] + 129.057 / 256 * a.Data[i, j, 1]
+ 25.604 / 256 * a.Data[i, j, 0]);
                Y = (byte)(Y + 16);                 // Gán giá trị cho Y theo công thức
```

```
                Cb = (byte)(-37.945 / 256 * a.Data[i, j, 2] - 74.494 / 256 * a.Data[i, j, 1]
+ 112.439 / 256 * a.Data[i, j, 0]);
                Cb = (byte)(Cb + 128);          //Gán giá trị cho Cb theo công thức
                Cr = (byte)(112.439 / 256 * a.Data[i, j, 2] - 94.154 / 256 * a.Data[i, j, 1]
- 18.285 / 256 * a.Data[i, j, 0]);
                Cr = (byte)(Cr + 128);          //Gán giá trị cho Cr theo công thức
                b.Data[i, j, 0] = Y;    //Gán giá trị cho kênh 0,1,2 tương ứng là Y, Cr, Cb
                b.Data[i, j, 2] = Cb;
                b.Data[i, j, 1] = Cr;
            }
        }
    }
```

Lưu ý: Vì hàm ToBitmap() không hiển thị được ảnh kiểu Ycc, do đó muốn hiển thị kết quả với dạng ảnh Ycc ta cần qua một bước trung gian là gán các dữ liệu ba kênh màu sau khi chuyển đổi sang cho ba kênh R, G, B của một ảnh RGB khác, sau đó mới hiển thị.

Kết quả:

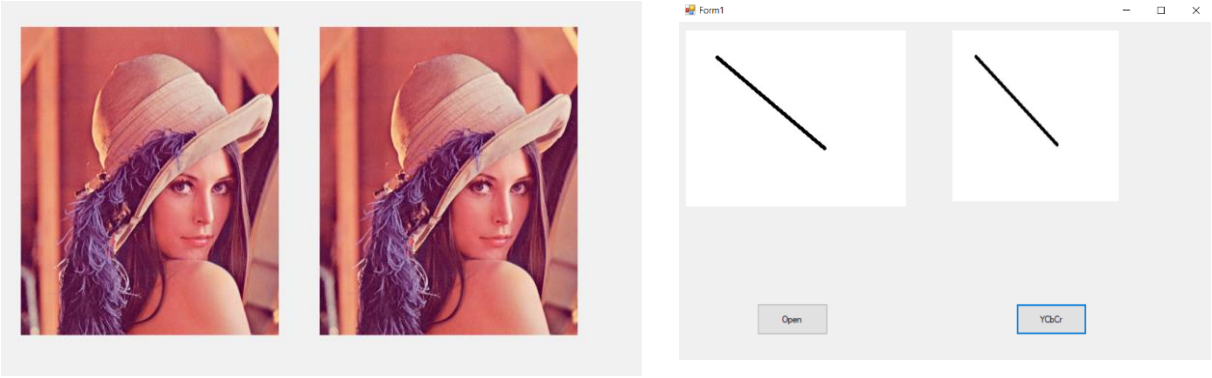
Hình 4.2 - RGB to YCbCr

4.1.2. Resize

Code cho thuật toán:

```
new Image<Ycc, byte> Size(Image<Ycc, byte> a)      //Hàm trả về dữ liệu kiểu ảnh Ycc
    {
        double k = img.Height / 8;       //k để lưu giá trị của số dòng block có thể chia được
        int newsize = 8 * (int)Math.Round(k);      //làm tròn giá trị và nhân lại cho 8
        a = a.Resize(newsize, newsize, Inter.Linear);      //hàm thay đổi size ảnh theo kích thước cho trước, lúc này ảnh sẽ có kích thước vuông
        return a;       //Trả về ảnh
    }
```

Kết quả:



Hình 4.3 - Chia thành các khối 8×8

4.1.3. DCT

Code cho thuật toán:

```
double DCT(double[,] f, int i, int j)     //i và j là tọa độ của các khối 8x8
    {
        double s = 0;
```

```
        for (int k = 0; k <= 7; k++)     //Quét ma trận 8x8 trong mỗi khối block
        {
            for (int l = 0; l <= 7; l++)
            {
                s += f[k + 8 * i - 8, l + 8 * j - 8] * Math.Cos((2 * k + 1) * (k + 8 * (i - 1)) * Math.PI / 16) * Math.Cos((2 * l + 1) * (l + 8 * (j - 1)) * Math.PI / 16);
  //các trị số k+8i-8 và l+8j-8 là các tọa độ chuyển đổi giữa các tọa độ cục bộ (0-7) sang các tọa độ toàn cục của ảnh, sau đó cộng dồn vào s
            }
        }
      return s;
   }
//Đoạn code sau này nằm ở đoạn chương trình chính, với giá trị pixel đầu tiên của các khối block 8x8 (có tọa độ (0,0)) thì chia s cho 8, 63 pixel còn lại thì chia cho 4 (theo công thức trên). Muốn vậy, ta khai báo thêm biến dem để sau mỗi ô pixel quét qua thì tăng biến lên 1, khi quét hết 64 ô pixel thì dem=64, khi đó sẽ sang block mới, khi đó gán lại dem = 0 để báo hiệu khi đó đang ở vị trí đầu tiên của 1 khối block mới
  int i,j,k,l;
  int dem=0;
  for (i = 1; i <= n; i++)
        {
            for (j = 1; j <= n; j++)
            {
                for (k = 8 * i - 8; k <= 8 * i - 1; k++)
                {
```

```
            for (l = 8 * j - 8; l <= 8 * j - 1; l++)
            {
               if (dem == 0)
               {
                  DCTY[k, l] = DCT(Y, i, j) / 8;
                  DCTCb[k, l] = DCT(Cb, i, j) / 8;
                  DCTCr[k, l] = DCT(Cr, i, j) / 8;
               }
               else
               {
                  DCTY[k, l] = DCT(Y, i, j) / 4;
                  DCTCb[k, l] = DCT(Cb, i, j) / 4;
                  DCTCr[k, l] = DCT(Cr, i, j) / 4;
               }
               dem += 1;
               if (dem == 64)
               {
                  dem = 0;
               }
            }
         }
      }   }
```

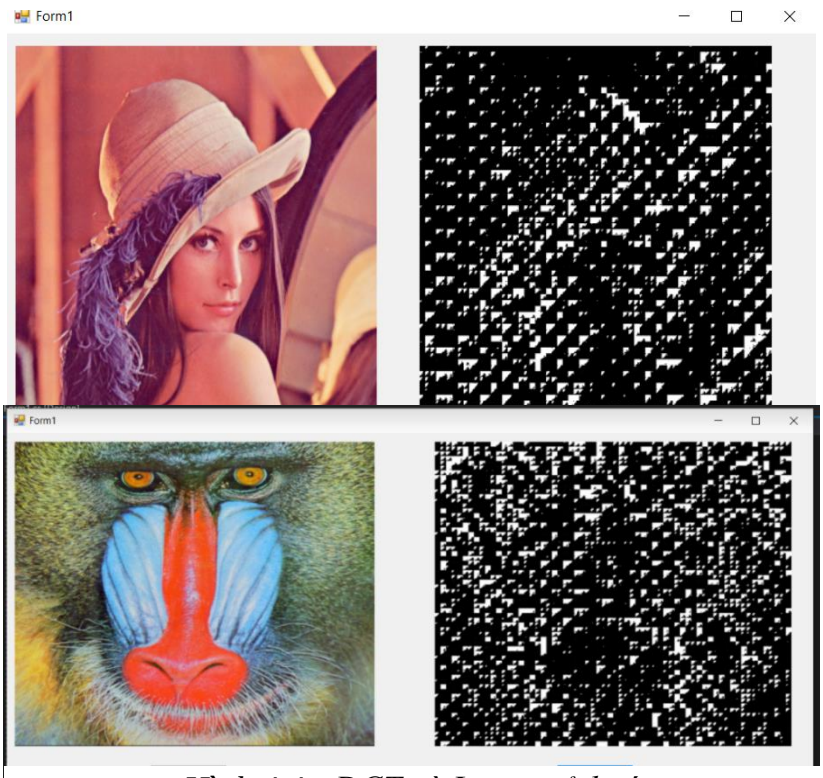
4.1.4. Lượng tử hoá

Code cho thuật toán:

```
int H = img2.Height;        //Chiều cao ảnh
int W = img2.Width;         //Chiều rộng ảnh
int n = H / 8;              //Số block 8x8 trên 1 cạnh
//Khởi tạo bảng lượng tử Q
double[,] Q =
        {
           {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}
        };
 //2 vòng for đầu tiên truy xuất đến toạ độ các khối block, 2 vòng for sau truy xuất đến các toạ độ pixel trong khối đó và thực hiện quá trình lượng tử hóa
            for (i = 1; i <= n; i++)
        {
           for (j = 1; j <= n; j++)
           {
              for (k = 8 * i - 8; k <= 8 * i - 1; k++)
{
                 for (l = 8 * j - 8; l <= 8 * j - 1; l++)
```

```
                {
//Thực hiện lượng tử hóa cho 3 kênh, hàm Round() để loại bỏ phần thập phân sau khi chia, các chỉ số trong Q[][] để truy xuất đến các tọa độ trong Q như phép DCT ở trên
                    DCTY[k, l] = Math.Round((DCTY[k, l] / Q[k + 8 - 8 * i, l + 8 - 8 * j]), 0);
                    DCTCb[k, l] = Math.Round((DCTCb[k, l] / Q[k + 8 - 8 * i, l + 8 - 8 * j]), 0);
                    DCTCr[k, l] = Math.Round((DCTCr[k, l] / Q[k + 8 - 8 * i, l + 8 - 8 * j]), 0);
                }
            }
        }
    }
```

Kết quả:



Hình 4.4 - DCT và Lượng tử hoá

4.1.5. Quét zigzag

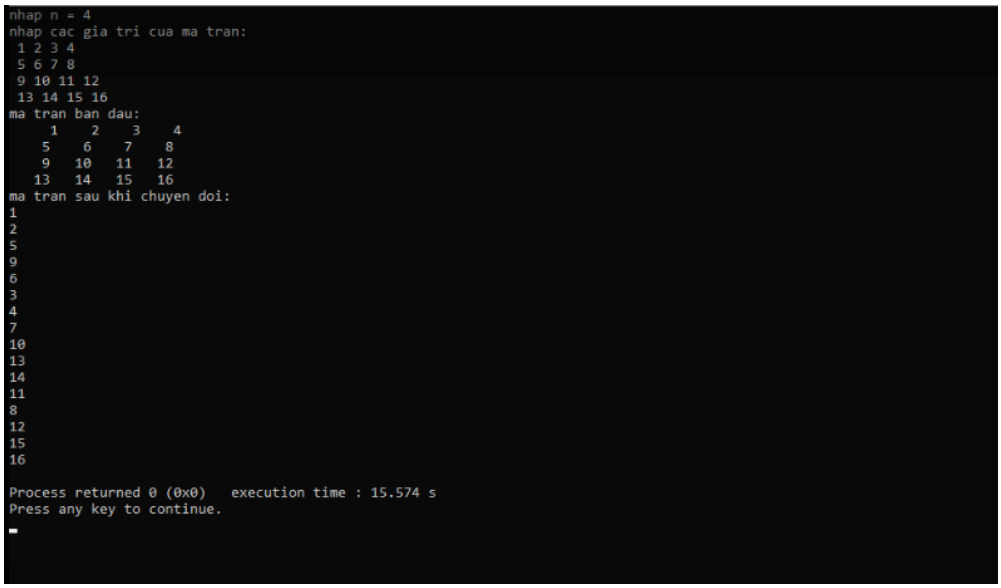
Code cho thuật toán:

```
void Zigzag(double[,] f, int n, double[] g)  // xuất các giá trị của ma trận 2 chiều f ra cùng một hàng theo hướng zigzag
    {
        int s = 0;   // s là biến đếm
        // ma trận vuông cấp n thì sẽ có 2*n - 1 đường chéo đánh số từ 0 đến 2n - 2
        // ta sẽ xuất các giá trị của ma trận theo hai hướng
        // đi xuống: khi chỉ số đường chéo lẻ
        // đi lên: khi chỉ số đường chéo chẵn
        // các phần tử thuộc cùng một đường chéo trên ma trận thì có tổng chỉ số hàng và cột bằng nhau và bằng chỉ số đường chéo
        for (int k = 0; k < (2 * n - 1); k++) // k là chỉ số đường chéo
        {
            if (k == 0)
            {
                g[s] = f[0, 0]; // trường hợp k = 0 thì gán giá trị của chỉ số đầu tiên ma trận cho phần tử đầu tiên của hàng
                s += 1;
            }
            else if ((k > 0) && (k < (2 * n - 2)))
            {
                if ((k % 2) == 1)                    // đi xuống khi k lẻ
                {
```

```
                for (int i = 0; i < n; i++)
                {
                    for (int j = n - 1; j >= 0; j--)
                    {
                        if ((i + j) == k)          // tổng chỉ số hàng và cột bằng chỉ số đường chéo
                        {
                            g[s] = f[i, j];
                            s += 1;
                        }
                    }
                }
            }
            else                                   // đi lên khi k chẵn
            {
                for (int i = n - 1; i >= 0; i--)
                {
                    for (int j = 0; j < n; j++)
                    {
                        if ((i + j) == k)          // tổng chỉ số hàng và cột bằng chỉ số đường chéo
                        {
                            g[s] = f[i, j];
                            s += 1;
```

```
                    }
                }
            }
        }
        else
        {
            g[s] = f[n - 1, n - 1];     // trường hợp k = 2*n - 2  thì gán giá trị của chỉ số cuối cùng  ma trận cho phần tử cuối cùng của hàng
        }
    }
}
```

Ví dụ về kết quả: Giả sử ta có ma trận sau, sau khi quét zigzag thì ta được kết quả:



Hình 4.5 - Quét zigzag thuận

4.1.6. Mã hoá Huffman

4.2. Quá trình giải nén ảnh

4.2.1. Giải mã Huffman

4.2.2. Quét zigzag ngược

Code cho thuật toán:

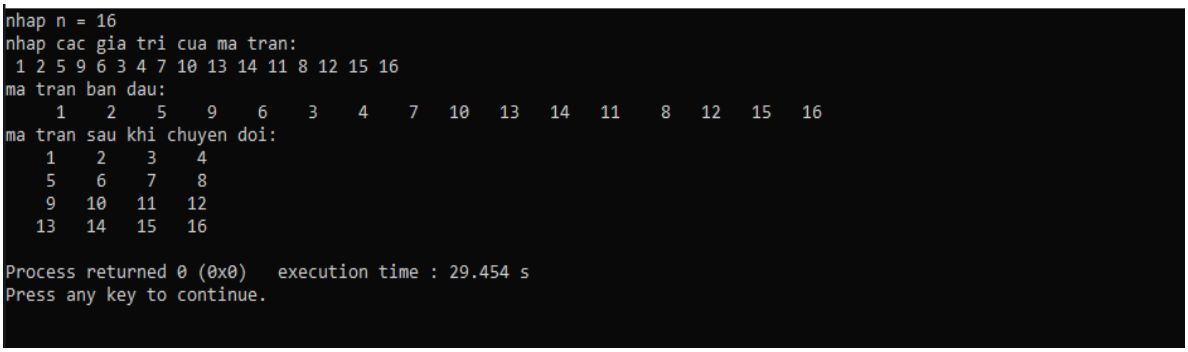
```
void InZigzag(double[] A, int N, double[,] f)     // chuyển các giá trị cùng một hàng qua ma trận 2 chiều f  theo hướng zigzag
    {                                  // các bước làm của InZigzag ngược lại so với Zigzag
        int s = 0;   // biến đếm
        int n;
        n = (byte)Math.Sqrt(N);    // n là cấp của ma trận 2 chiều f

        for (int k = 0; k < (2 * n - 1); k++) // k là chỉ số đường chéo
        {
            if (k == 0)
            {
                f[0, 0] = A[s];  // trường hợp k = 0 thì gán giá trị của chỉ số đầu tiên hàng cho phần tử đầu tiên của ma trận
                s += 1;
            }
            else if ((k > 0) && (k < (2 * n - 2)))
            {
                if ((k % 2) == 1)                 // đi xuống
                {
```

```
                for (int i = 0; i < n; i++)
                {
                    for (int j = n - 1; j >= 0; j--)
                    {
                        if ((i + j) == k)      // tổng chỉ số hàng và cột bằng chỉ số đường chéo
                        {
                            f[i, j] = A[s];
                            s += 1;
                        }
                    }
                }
            }
            else                            // đi lên
            {
                for (int i = n - 1; i >= 0; i--)
                {
                    for (int j = 0; j < n; j++)
                    {
                        if ((i + j) == k)      // tổng chỉ số hàng và cột bằng chỉ số đường chéo
                        {
                            f[i, j] = A[s];
                            s += 1;
```

```
                    }
                }
            }
        }
    }
    else
    {
        f[n - 1, n - 1] = A[s];          // trường hợp k = 2*n - 2  thì gán giá trị của chỉ số cuối cùng  hàng  cho phần tử cuối cùng của ma trận
    }
  }
}
```

Ví dụ về kết quả:



Hình 4.6 - Quét zigzag ngược

4.2.3. Nghịch lượng tử hoá

Code cho thuật toán:

```
//A, B và C là các ma trận kết quả ở 3 kênh ở bước Zigzag nghịch, tương tự như trên ta có đoạn code sau:
double[,] IDCTY = new double[H, W];     //Khai báo các ma trận của 3 kênh
```

```
double[,] IDCTCb = new double[H, W];
    double[,] IDCTCr = new double[H, W];
for (i = 1; i <= n; i++)
          {
             for (j = 1; j <= n; j++)
             {
                for (k = 8 * i - 8; k <= 8 * i - 1; k++)
                {
                   for (l = 8 * j - 8; l <= 8 * j - 1; l++)
                   {
                      // IDCTY[k, l] = Math.Round((A[k, l] * Q[k + 8 - 8 * i, l + 8 - 8 *
j]), 0);
                      // IDCTCb[k, l] = Math.Round((B[k, l] * Q[k + 8 - 8 * i, l + 8 - 8
* j]), 0);
                      // IDCTCr[k, l] = Math.Round((C[k, l] * Q[k + 8 - 8 * i, l + 8 - 8 *
j]), 0);
                   } } }   }
```

4.2.4. DCT nghịch

Code cho thuật toán:

```
double IDCT(double[,] f, int i, int j)
      {
         double s = 0;
         for (int k = 0; k <= 7; k++)        //Quét các giá trị khối 8 × 8
         {
```

```
            for (int l = 0; l <= 7; l++)
            {
               if ((k == 0) && (l ==0))      //vị trí đầu tiên của mỗi khối Block
               {
                  s += 1/8 * f[k + 8 * i - 8, l + 8 * j - 8] * Math.Cos((2 * k + 1) * (k + 8 * (i - 1)) * Math.PI / 16) * Math.Cos((2 * l + 1) * (l + 8 * (j - 1)) * Math.PI / 16);
               }
               else
               {
                  s += 1/4 * f[k + 8 * i - 8, l + 8 * j - 8] * Math.Cos((2 * k + 1) * (k + 8 * (i - 1)) * Math.PI / 16) * Math.Cos((2 * l + 1) * (l + 8 * (j - 1)) * Math.PI / 16);
               }
            }
         }
         return s;
      }
//Đoạn code ở chương trình chính, biến đổi DCT cho toàn bộ ảnh
double[,] IY = new double[H, W];
double[,] ICb = new double[H, W];
double[,] ICr = new double[H, W];
for (i = 1; i <= n; i++)
         {
            for (j = 1; j <= n; j++)
            {
```

```
            for (k = 8 * i - 8; k <= 8 * i - 1; k++)
            {
               for (l = 8 * j - 8; l <= 8 * j - 1; l++)
               {
                  IY[k, l] = IDCT(IDCTY, i, j);
                  ICb[k, l] = IDCT(IDCTCb, i, j);
                  ICr[k, l] = IDCT(IDCTCr, i, j);
               }
            }
         }
      }
```

4.2.5. Xử lý màu

Code cho thuật toán:

```
for (i = 0; i < H; i++)
      {
         for (j = 0; j < W; j++)
         {
            img2.Data[i, j, 0] = (byte)(IY[i, j] + 128);
            img2.Data[i, j, 2] = (byte)(ICb[i, j] + 128);
            img2.Data[i, j, 1] = (byte)(ICr[i, j] + 128);
         }
      }
void YCbCr2RGB(Image<Bgr, Byte> b, Image<Ycc, byte> a)
    {
```

```
        int i, j;
        byte B, G, R;     //khai báo 3 biến lưu giá trị 3 kênh R, G, B
        for (i = 0; i < img.Height; i++)
        {
           for (j = 0; j < img.Width; j++)
           {
      //Thực hiện tính toán theo công thức
              R = (byte)(298.082 / 256 * a.Data[i, j, 0] + 408.583 / 256 * a.Data[i, j, 1] - 222.291);
              G = (byte)(298.082 / 256 * a.Data[i, j, 0] - 100.291 / 256 * a.Data[i, j, 2] - 208.120 / 256 * a.Data[i, j, 1] + 135.576);
              B = (byte)(298.082 / 256 * a.Data[i, j, 0] + 516.412 / 256 * a.Data[i, j, 2] - 276.836);
     //Gán giá trị cho 3 kênh màu
              b.Data[i, j, 0] = B;
              b.Data[i, j, 1] = G;
              b.Data[i, j, 2] = R;
           }
        }
     }
```

Kết quả:

Hình 4.7 - Chuyển về lại kênh màu RGB

4.2.6. Tái tạo lại kích thước ban đầu

Code cho thuật toán:

```
new Image<Bgr, byte> Resize(Image<Bgr, byte> a)
    {
        a = a.Resize(img.Width, img.Height, Inter.Linear);
        return a;
    }
```

Kết quả:

Hình 4.8 - Tái tạo kích thước ban đầu

CHƯƠNG 5. KẾT LUẬN

Công nghệ nén ảnh JPEG là một phương pháp nén ảnh vô cùng hiệu quả. Trong sự phát triển của công nghệ thông tin và Internet thì JPEG là một công cụ rất hữu hiệu trong việc truyền thông tin và lưu trữ thông tin. Trong quá trình phát triển, JPEG ngày càng được cải tiến để khắc phục những nhược điểm của mình để trở thành định dạng ảnh phổ biến nhất trên thế giới. Tuy nhiên, vì JPEG là chuẩn không bảo toàn bởi thông tin về ảnh sẽ bị thay đổi khi nó bị mã hóa nhiều lần. Vì vậy mà các chuyên gia trên thế giới đã bắt đầu nghiên cứu các định dạng ảnh thừa kế những ưu điểm của JPEG mà hạn chế được sự mất mát thông tin. Và chúng ta cùng chờ đợi nhiều định dạng ảnh khác hiệu quả hơn ra đời trong tương lai gần.

Về hạn chế, nhóm chúng em rất lấy làm tiếc khi chưa hiện thực hoá được vấn đề mã hoá Huffman và giải mã quá trình đó. Mặc dù nhóm đã tiếp tục đào sâu thêm về vấn đề này và phát hiện rằng ngoài Phương pháp mã hoá Huffman, còn có các phương pháp mã hoá khác phù hợp cho việc nén ảnh JPEG, đó là Phương pháp mã hoá loạt dài RLE (Run Length Encoding), Phương pháp mã hoá Lemple-Ziv nhưng do sự hạn chế về hiểu biết nên nhóm đã không thực hiện được những phương pháp mã hoá trên. Kính mong quý thầy cô thông cảm về sự thiếu sót này.

Chúng em xin chân thành cảm ơn quý thầy cô.

DANH MỤC TÀI LIỆU THAM KHẢO

1. Jussi Lammi and Tapani Sarjakoski, *Compression of digital color images be the JPEG*, https://www.isprs.org/proceedings/XXIX/congress/part2/456_XXIX-part2.pdf.
2. PTS. Đinh Tiến Sơn, *Phương pháp nén ảnh theo chuẩn JPEG*, https://www.scribd.com/doc/52683135/Ph%C6%B0%C6%A1ng-phap-nen-%E1%BA%A3nh-theo-chu%E1%BA%A9n-JPEG.
3. Ricardo L. de Queiroz (12/12/1998), *Processing JPEG-compressed images and documents*, http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.454.7668&rep=rep1&type=pdf.
4. Suman Kunwar (12/12/2017), *Image compression algorithm and JPEG standard*, https://www.researchgate.net/publication/322240903_Image_Compression_Algorithm_and_JPEG_Standard.