**Bài tập lớn môn Mã hóa dữ liệu đa phương tiện**

**Đề tài 07**

# Triển khai thử nghiệm

## Môi trường

* Ngôn ngữ sử dụng: Python
* Thư viện: OpenCV, NumPy, bitarray
* Source Code: <https://github.com/VuXuanBac/BaselineJPEG/>

## Biểu diễn dữ liệu

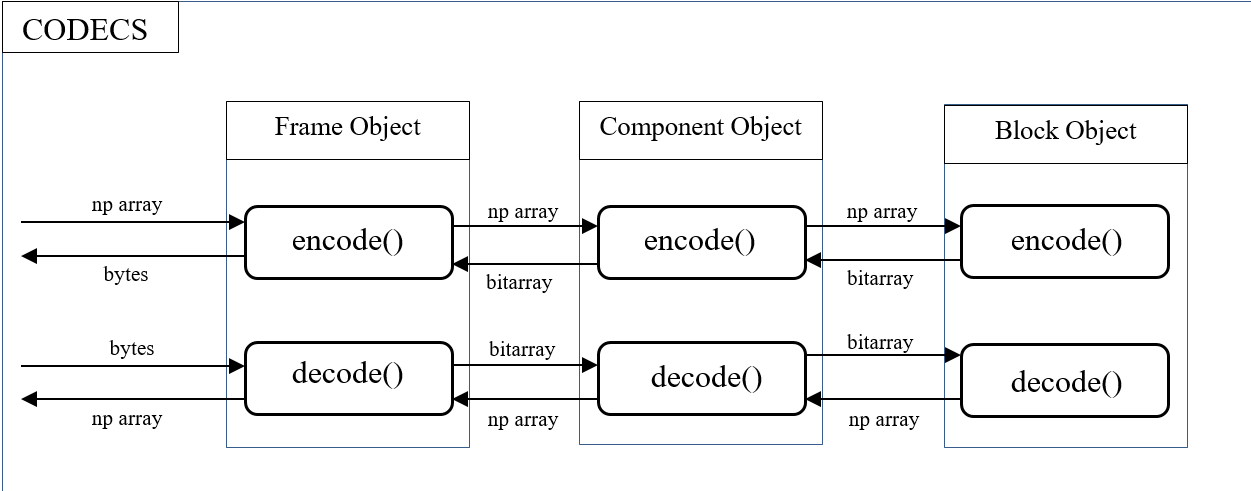
Chương trình có thể xử lý với hình ảnh màu theo hệ BGR hoặc ảnh greyscale. Trên thực tế, xử lý hình ảnh greyscale hoàn toàn tương tự, và thậm chí đơn giản hơn (tương đương một thành phần Y), do đó, trong phần triển khai thử nghiệm, nhóm em chỉ đề cập đến hình ảnh màu hệ BGR.

Dữ liệu biểu diễn hình ảnh được biểu diễn dưới dạng một ma trận 3 chiều (3D NumPy array) kích thước , trong đó là kích thước hình ảnh (chiều cao, chiều rộng) theo đơn vị pixels và là số thành phần tín hiệu (component) trong hệ màu biểu diễn hình ảnh, với hệ BGR hay YCbCr thì .

Dữ liệu mã hóa được biểu diễn dưới dạng Python ***bytes*** object (một tập hợp trong đó mỗi phần tử dữ liệu biểu diễn một byte nhị phân).

## Cấu trúc chương trình

Chương trình được tổ chức dưới dạng các modules chứa các ***lớp*** và ***hàm chức năng***, trong đó có 3 lớp sau là quan trọng với mục tiêu của đề tài, và luồng xử lý dữ liệu trên ba lớp này diễn ra như sau:



* ***Frame***: Bộ mã hóa/giải mã ở mức hình ảnh, với một số methods sau
* *encode()*: Thực hiện *chuyển đổi hệ màu*, *tiền xử lý (preencode)* *trên từng component*, *mã hóa (encode) từng component* và *ghép kết quả mã hóa* thành dòng bytes.
* *decode()*: Thực hiện *giải mã dòng bytes* thành ma trận dữ liệu biểu diễn các components, *hậu xử lý (postdecode) trên từng component* và *chuyển đổi hệ màu* về BGR.
* Ngoài ra, class còn có một số methods khác dùng để thiết lập cấu hình cho bộ mã hóa/giải mã như hệ số chất lượng, lấy mẫu sắc tố,...
* ***Component***: Bộ mã hóa/giải mã ở mức thành phần tín hiệu màu (component), với một số methods sau
* *encode()*: Thực hiện mã hóa ma trận dữ liệu biểu diễn một component thành dòng bits
* *decode()*: Thực hiện giải mã dòng bits thành ma trận dữ liệu biểu diễn một component.
* *preencode()*: Thực hiện *Downsampling* theo chiến lược lấy mẫu sắc tố, *Padding* (mở rộng kích thước để phù hợp với việc chia khối) và *Level Shift* (chuyển dải biểu diễn) trên ma trận dữ liệu ***trước mã hóa encode()***.
* *postdecode()*: Thực hiện *Level Shift*, *Cropping* (Loại bỏ phần padding trong quá trình mã hóa) và *Interpolation* (nội suy để khôi phục kích thước ban đầu) trên ma trận dữ liệu ***sau giải mã decode()***.
* ***Block***: Bộ mã hóa/giải mã ở mức khối dữ liệu 8 x 8, với 2 methods:
* *encode()*: Thực hiện FDCT, Quantization và mã hóa Huffman.
* *decode()*: Thực hiện giải mã Huffman, Dequantization và IDCT.

## Luồng hoạt động của người dùng

Để thực hiện mã hóa và giải mã hình ảnh:

* Trước hết, tạo một ***Frame*** object và thiết lập các cấu hình cho bộ mã hóa/giải mã, đơn giản nhất là thiết lập hệ số chất lượng.
* Để *mã hóa*, người dùng cần ma trận biểu diễn dữ liệu hình ảnh theo hệ BGR với kích thước như mô tả trên. Có thể sử dụng thư viện OpenCV để đọc một tệp hình ảnh và biểu diễn dữ liệu hình ảnh vừa đọc dưới dạng ma trận này. (Với hình ảnh greyscale, ma trận biểu diễn chỉ có 2 chiều ). Đưa ma trận biểu diễn này vào method *encode()* của ***Frame*** Object, ta được một dòng bytes là kết quả mã hóa.
* Để *giải mã*, người dùng cần (1) dòng bytes mã hóa và (2) kích thước mong đợi của hình ảnh khôi phục (do trong quá trình mã hóa có các điểm dữ liệu được thêm vào). Đưa các tham số này vào method *decode()* của ***Frame*** Object, ta được một ma trận dữ liệu biểu diễn hình ảnh theo hệ BGR, ta có thể sử dụng OpenCV để hiển thị hình ảnh kết quả.
* Để thực hiện *đánh giá hiệu quả* của việc mã hóa và giải mã (đánh giá tỉ số nén và độ tổn hao), người dùng tạo một ***CompressorAnalysis*** object từ module *compare* với 3 đối số cho hàm khởi tạo là (1) ma trận dữ liệu biểu diễn hình ảnh trước mã hóa (hình ảnh gốc), (2) ma trận dữ liệu biểu diễn hình ảnh sau giải mã (hình ảnh khôi phục) và (3) kích thước của dòng bytes mã hóa. Thực hiện *print()* object vừa tạo, màn hình console hiển thị các thông số đánh giá. Thực hiện method *show()* trên object vừa tạo, màn hình sẽ hiển thị hình ảnh gốc và hình ảnh khôi phục.

## Chi tiết khối chức năng

Sơ đồ tổng quan của Transform Coding có 3 khối chức năng, trong chương trình này, nhóm em triển khai như sau:

* ***Phép biến đổi***: Phép biến đổi sử dụng là DCT 2 chiều trên khối dữ liệu 8 x 8, sử dụng hàm từ thư viện OpenCV.
* ***Lượng tử hóa***: Nếu gọi là giá trị ở vị trí của ma trận khối dữ liệu 8 x 8 sau biến đổi FDCT, là hệ số lượng tử hóa ở vị trí của ma trận lượng tử hóa 8 x 8, là hệ số chất lượng xác định trước, cài đặt sẵn (sẽ được trình bày cụ thể ở dưới), thì giá trị ở vị trí của ma trận khối dữ liệu 8 x 8 sau lượng tử hóa là . Ở giai đoạn giải mã, giá trị ở vị trí của ma trận khối dữ liệu sau giải lượng tử hóa được xác định là .
* ***Mã hóa***: Ma trận khối dữ liệu 8 x 8 sau lượng tử hóa sẽ thực hiện mã hóa Huffman riêng cho thành phần DC và AC. Cụ thể, ma trận 2 chiều sẽ chuyển về mảng 1 chiều theo quy tắc ZigZag. Thành phần DC (vị trí 0) sẽ sử dụng mã hóa dự đoán, giá trị được mã hóa Huffman là , với là hệ số DC của ma trận khối hiện tại, là hệ số DC của ma trận khối ngay trước khối hiện tại (trong cùng component) ***trong quá trình duyệt*** (sẽ được trình bày cụ thể ở dưới) - khởi tạo là 0 với ma trận đầu tiên. Chi tiết: Phần F.1.2.1.1 trong [1]. Các thành phần AC (vị trí từ 1 đến 63) được mã hóa RLC. Chi tiết: Phần F.1.2.2.1 trong [1].

Dưới đây là một số *chi tiết kĩ thuật* trong chương trình của nhóm em:

1. Bảng lượng tử hóa cho thành phần Luma (Y) và Chroma (Cb, Cr) nhóm em sử dụng từ phần K.1 trong [1] (Table K.1 và Table K.2), lưu trực tiếp dưới dạng ma trận 8 x 8 trong mã nguồn.
2. Về mã hóa Huffman, nhóm em sử dụng 4 bảng từ điển: 2 bảng cho thành phần Luma (DC và AC) và 2 bảng cho thành phần Chroma (DC và AC). Các bảng này được xây dựng lại từ biểu diễn nén của chúng (là dòng byte đặc tả cho bảng, thường được đi cùng với dữ liệu hình ảnh). Các biểu diễn nén này lấy từ phần K.3.3 trong [1]. Thuật toán xây dựng bảng được tham khảo từ phần C.2 của [1].
3. Các thuật toán mã hóa và giải mã Huffman, nhóm em triển khai từ các sơ đồ trong phần F.1.2 của [1] (Figure F.2 và F.3), F.2.2 của [1] (Figure F.12 → F.18) và phần C.2 của [1] (Figure C.1 → C.3).

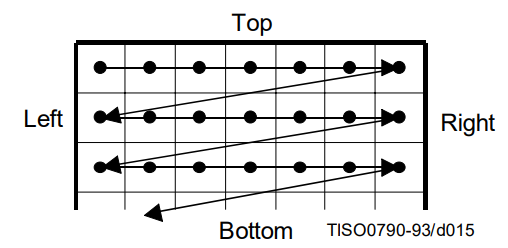
Để triển khai chức năng giải mã như các thuật giải trên, nhóm em định nghĩa lớp ***StateStream*** chứa dòng bits, có khả năng ghi nhớ vị trí và đọc một hoặc nhiều bits.

1. Về hàm trong quá trình lượng tử hóa và giải lượng tử hóa, nhóm em tham khảo từ mã nguồn của libjpeg [2], cụ thể là phần [Quantization](https://github.com/thorfdbg/libjpeg/blob/76133b78f65e716c334e1955dc9a30f0d59d2297/marker/quantization.cpp) (dòng 274-277). Hàm này thực hiện tính hệ số cho hệ số lượng tử hóa từ hệ số . Hệ số càng lớn thì tỉ số nén càng cao (càng nhiều giá trị sau lượng tử hóa về 0). Cụ thể, nếu , , nếu , , tức là, , hệ số này sẽ biến đổi nhanh trong đoạn và biến đổi chậm hơn ở đoạn còn lại, từ , .

Khi giá trị nằm ngoài đoạn , ta sẽ không sử dụng giá trị , thay vào đó sẽ được thay thế thành , tức là chỉ thực hiện cắt phần thập phân cho trong giai đoạn mã hóa và sử dụng trực tiếp trong giai đoạn giải mã.

1. Nói riêng về ***quá trình duyệt*** đề cập ở bước mã hóa Huffman trên. Phần này liên quan đến phần 4.8 trong [1], về quản lý đa thành phần (*multiple-component control*). Trong chương trình, nhóm em triển khai cả hai chức năng "non-interleaved" và "interleaved", tuy nhiên nhìn chung sự khác biệt về tỉ số nén và độ tổn hao không nhiều, và liên quan chủ yếu đến kĩ thuật nên sẽ không trình bày cụ thể mà chỉ quan tâm đến ***quá trình duyệt***, nhằm xác định giá trị .

Mặc định chế độ sử dụng là "non-interleaved", tương ứng với ***quá trình duyệt*** trong từng component là như sau:



Từ đây có thể xác định giá trị cho . Các giá trị sẽ được lưu lại ngay trong từng ***Component*** object.

Trong mã nguồn, lớp ***BlockExtend*** thực hiện nhiệm vụ lấy từng khối dữ liệu 8 x 8 trong ma trận dữ liệu của một component để đem ra cho quá trình mã hóa, và thực hiện ghép lại các khối dữ liệu 8 x 8 sau giải mã thành ma trận dữ liệu của một component, theo quy tắc trên.

## Độ mở của chương trình

Chương trình cho phép thiết lập các cấu hình sau cho quá trình mã hóa và giải mã JPEG, các cấu hình này được thiết lập qua ***Frame*** object, thông qua các method *set\_\_()*:

1. *set\_quality()*: Thiết lập hệ số chất lượng , một giá trị nguyên. Mặc định: **50**.
2. *set\_sampling\_factor()*: Thiết lập hệ số lấy mẫu phụ sắc tố, một giá trị nguyên, hoặc xâu kí tự. Mặc định: ***420*** (tương đương ***"4:2:0"***)
3. *set\_interpolation()*: Thiết lập chế độ làm mịn/giảm nhiễu cho quá trình khôi phục hình ảnh. Tham khảo các giá trị trong OpenCV (cv2.INTER\_\_\_). Mặc định: ***"linear"*** (Linear Interpolation).
4. *set\_quantization\_tables()*: Thiết lập các bảng lượng tử hóa cơ sở cho các thành phần tín hiệu (component). Mặc định: ***Sử dụng bảng từ mã nguồn***.
5. *set\_huffman\_tables()*: Thiết lập các bảng mã hóa Huffman cho các thành phần tín hiệu (component). Mặc định: ***Sử dụng bảng từ mã nguồn***.

Ngoài ra, khi thực hiện mã hóa *encode()* hoặc giải mã *decode()*, người dùng có thể lựa chọn chế độ *"non-interleaved"* hoặc *"interleaved"* cho đối số của methods, mặc định là *"non-interleaved"*.

Trong chương trình Test, nhóm em chỉ thực hiện thay đổi hệ số chất lượng để đạt mục tiêu của đề tài, các cấu hình khác sử dụng mặc định.

## Đánh giá và so sánh

### Thử nghiệm chương trình

Chương trình yêu cầu cài đặt môi trường như sau trước khi thực hiện kiểm thử:

* Cài đặt Python 3 (<https://www.python.org/downloads/>), tích hợp tải sẵn ***pip***.
* Cài đặt OpenCV: *pip install opencv-python==4.6.0.66*.
* Cài đặt NumPy: (Tích hợp tải cùng với OpenCV) *pip install numpy==1.24.0*.
* Cài đặt bitarray: *pip install bitarray==2.6.2*

Để thực hiện kiểm thử chương trình với một tệp hình ảnh, ta chạy dòng lệnh sau:

python test.py path/to/image quality

trong đó, path/to/image là đường dẫn đến tệp hình ảnh (định dạng bất kì), quality là một giá trị nguyên xác định hệ số chất lượng.

Chương trình sẽ thực hiện mã hóa ma trận biểu diễn hình ảnh gốc về dòng bytes và thực hiện giải nén dòng bytes đó về ma trận biểu diễn hình ảnh khôi phục, sau đó tính toán và hiển thị các chỉ số đánh giá: ***Độ tổn hao*** (PSNR) và ***Tỉ số nén***, cũng như hiển thị hình ảnh gốc và hình ảnh khôi phục lên màn hình.

***Độ tổn hao*** được tính toán trên so sánh giữa ma trận dữ liệu biểu diễn hình ảnh gốc và ma trận dữ liệu biểu diễn hình ảnh khôi phục.

***Tỉ số nén*** được tính toán theo tỉ lệ giữa số bytes cần thiết để lưu ma trận dữ liệu biểu diễn hình ảnh gốc (có thể tính nhanh với ma trận biểu diễn kích thước là , do mỗi phần tử trong ma trận sử dụng 1 byte) và kích thước của dòng bytes mã hóa.

Để đạt được yêu cầu về độ tổn hao và tỉ số nén, ta thực hiện thay đổi giá trị dần đến khi độ tổn hao hiển thị ít nhất là 30dB. Xu hướng chung là khi tăng thì độ tổn hao tăng và tỉ số nén giảm, và ngược lại.

### So sánh với công cụ khác

Chương trình cũng có phần thử nghiệm so sánh với công cụ khác, cụ thể là hàm *imwrite()* của thư viện OpenCV. Phần thử nghiệm này sẽ thực hiện đọc một tệp hình ảnh, sau đó lưu hình ảnh vào một tệp mới với định dạng ***.jpg***, sử dụng chế độ nén Baseline JPEG. Ta cũng có thể điều chỉnh hệ số chất lượng cho hàm. ***Độ tổn hao*** lúc này được tính dựa trên so sánh giữa ma trận dữ liệu biểu diễn hình ảnh ban đầu và ma trận dữ liệu biểu diễn hình ảnh khôi phục từ tệp vừa lưu. ***Tỉ số nén*** cũng được tính bằng tỉ lệ giữa số bytes cần thiết để lưu ma trận biểu diễn hình ảnh ban đầu và kích thước của *dòng bytes mã hóa* trong tệp vừa lưu (thực hiện đọc tệp và tách riêng các bytes nằm trong phần ECS-***entropy coded segment***, đồng thời loại bỏ các *stuff bytes*, như phần F.1.2.3 trong [1]).

Chương trình thử nghiệm với OpenCV có thể chạy với dòng lệnh sau:

python test\_cv2.py path/to/image quality

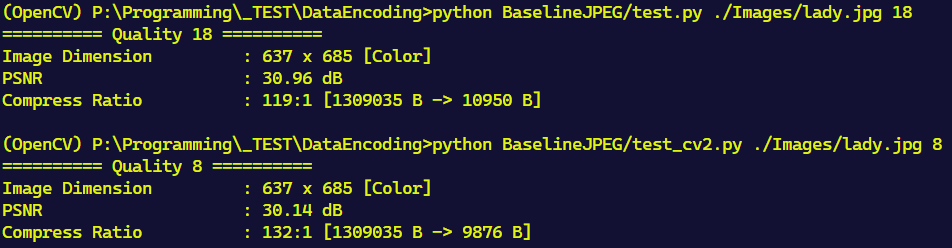
### Kết quả

So sánh với chương trình của nhóm em, kết quả nhìn chung là hàm nén của thư viện OpenCV thực hiện ***nhanh hơn*** và với cùng độ tổn hao thì ***tỉ số nén có lớn hơn***, song không đáng kể.

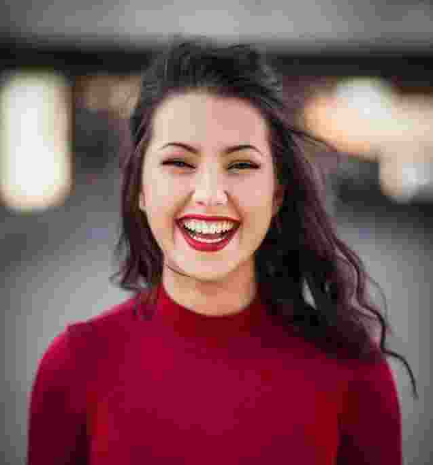
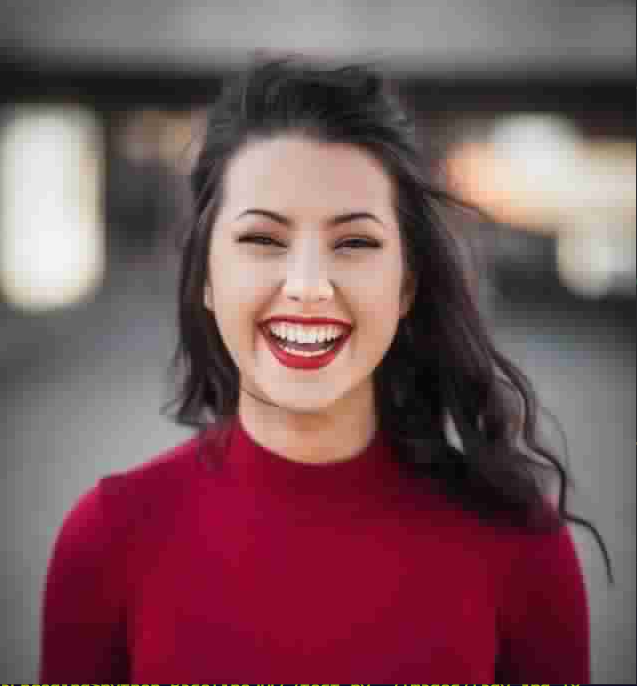
Cụ thể, với hình ảnh màu *lady.jpg* kích thước 637 x 685, kết quả đánh giá như sau:

* ***Thông số đánh giá***: Do cách đánh giá khác nhau về hệ số nên kết quả so sánh được đánh giá dựa trên tương đồng về độ tổn hao giữa hai thử nghiệm.

Trong thử nghiệm này, chương trình của nhóm em chạy với hệ số chất lượng 18 là có thể cho độ tổn hao lớn hơn 30dB, tỉ số nén khá tốt 119:1. Với thử nghiệm hàm nén của OpenCV, với độ tổn hao xấp xỉ thì có tỉ số nén cao hơn 132:1, kích thước dòng bytes mã hóa là ít hơn khoảng 1 KB.



* ***Hình ảnh***: Dưới đây là ba hình ảnh, theo thứ tự là hình ảnh gốc, hình ảnh khôi phục từ chương trình của nhóm chúng em và hình ảnh khôi phục từ thử nghiệm OpenCV. Về trực quan thì hình ảnh khôi phục của nhóm em có ***mịn*** hơn hình ảnh khôi phục từ OpenCV (do có xử lý nội suy trong quá trình khôi phục).



# Tài liệu tham khảo

|  |  |
| --- | --- |
| [1] | CCITT, "T.81 – DIGITAL COMPRESSION AND CODING OF CONTINUOUS-TONE STILL IMAGES – REQUIREMENTS AND GUIDELINES," 09 1992. [Online]. Available: https://www.w3.org/Graphics/JPEG/itu-t81.pdf. |
| [2] | "Libjpeg," [Online]. Available: https://github.com/thorfdbg/libjpeg/. |