TRƯỜNG ĐẠI HỌC
CÔNG NGHỆ THÔNG TIN
KHOA: KHOA HỌC MÁY TÍNH

ĐỒ ÁN CUỐI KỲ

COMPRESSION VÀ DECOMPRESSION

Môn học: **TÍNH TOÁN ĐA PHƯƠNG TIỆN**

Mã môn học: **CS232.J11.KHTN**

Giáo viên lý thuyết: **NGÔ ĐỨC THÀNH**

Giáo viên hướng dẫn thực hành: **ĐỖ VĂN TIẾN**

Sinh viên thực hiện:

Cao Quốc Đạt **15520097**

Huỳnh Vĩ Hà **15520175**

Hoàng Yến **15521042**

CHƯƠNG 1: GIỚI THIỆU

Nén dữ liệu là quá trình mã hóa thông tin, dùng số bit ít hơn so với thông tin chưa được mã hóa. Hiểu một cách đơn giản, nén dữ liệu có nghĩa là thu nhỏ kích thước dung lượng của lữ liệu hiện tại.

Tác dụng của việc nén dữ liệu: Nén dữ liệu giúp việc sao chép, di chuyển cũng như truyền dữ liệu được dễ dàng và nhanh chóng, giúp tiết kiệm dung lượng lưu trữ trên các thiết bị và một số các lợi ích khác.

Phân loại: Các phương pháp nén dữ liệu được chia làm hai nhóm phương pháp chính: lossless compression và lossy compression

- **Lossless compression:**
  - Trong lossless compression, tính toàn vẹn của dữ liệu được bảo toàn. Dữ liệu ban đầu trước khi nén và dữ liệu sau khi giải nén là giống nhau hoàn toàn, không có bất cứ phần nào của dữ liệu bị mất trong quá trình nén và giải nén.
  - Phương pháp này được sử dụng cho các dữ liệu liên quan đến văn bản, con số, nhằm tránh các mất mát làm sai lệch các thông tin quan trọng.
- **Lossy compression:**
  - Lossy compression dựa trên hiện tượng mắt và tai người khó nhận ra những sự thay đổi nhỏ. Dữ liệu ban đầu trước khi nén và dữ liệu sai khi nén không giống nhau hoànt toàn.
  - Phương pháp compression phù hợp với việc nén dữ liệu media không cần độ chính xác cao như ảnh, video, audio…

Trong đồ án này, nhóm quyết định chỉ nén trên dữ liệu ảnh.

Các thuật toán nhóm chọn để tìm hiểu, xây dựng chương trình minh họa và đánh giá như sau:

- **Lossless: LZW và Huffman**
- **Lossy: K-means**

Chi tiết của từng thuật toán sẽ được trình bày trình chi tiết trong chương tiếp theo.

CHƯƠNG 2: CÁC THUẬT TOÁN

2.1 Thuật toán Lempel-Ziv-Welch (LZW)

2.1.1 Tìm hiểu về thuật toán

Ý tưởng cơ bản:

LZW là một thuật toán dictionary-based encoding.

LZW sử dụng codewords để thay thế các chuỗi ký tự thường xuyên xuất hiện cùng nhau trong dữ liệu. Chẳng hạn, codeword "1" tương ứng với chuỗi "aba". Như vậy, thay vì lưu cả chuỗi "aba", ta chỉ cần lưu codewords "1". Từ ý tưởng này, ta sẽ tạo nên một bảng dictionary để lưu các codewords và các chuỗi tương ứng với codeword đó.

Encode và decoder sẽ xây dựng một bảng dictionary giống nhau. Do đó, bảng dictionary này không cần phải lưu kèm theo dữ liệu sau khi encode.

Thuật toán:

Encode:

```
s = next input character
while not EOF {
      c = next input character
      if s + c exists in the dictionary
            s = s + c
      else {
            output the code for s
            add string s + c to the dictionary with a new code
            s = c
      }
}
output the code for s
```

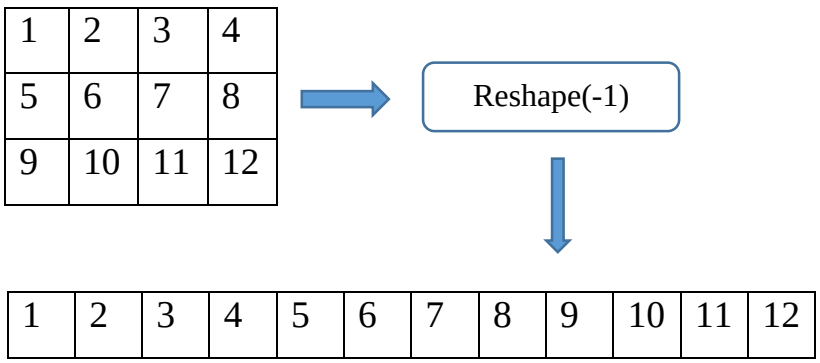
Decode:

```
s = NULL
while not EOF {
	k = next input code
	entry = dictionary entry for k
	if (entry == NULL)
		entry = s + s[0]
	output entry
	if (s != NULL)
		add string s + entry[0] to dictionary with a new code
	s = entry
}
```

2.1.2 Phân tích và xây dựng chương trình

Hàm lzw_encode:

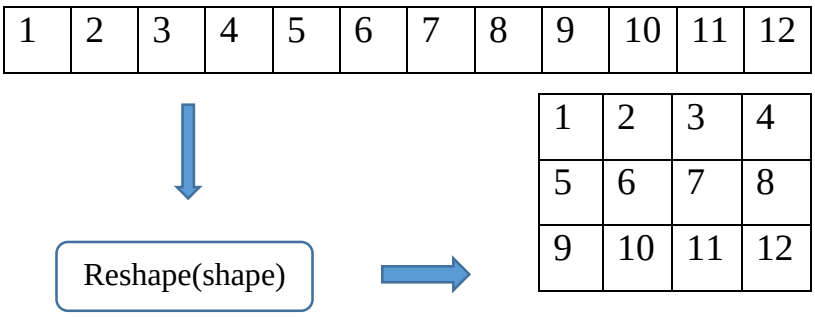
- **Input:** link của ảnh cần nén và link để lưu kết quả.
- **Output:** một chuỗi codewords.
- Load ảnh bằng lệnh cv2.imread
- Theo như thuật toán LZW được trình bày trong phần trên, input là một chuỗi. Tuy nhiên, ta đang xét đến việc nén dữ liệu ảnh, ảnh ở đây là dữ liệu có hai hoặc ba chiều, vì thế trước tiên ta reshape lại ma trận ảnh thành một chuỗi các giá trị.



- Tiếp theo, ta khởi tạo dictionary ban đầu với 256 cặp giá trị (string, code).
- Ta bắt đầu encode chuỗi giá trị của ảnh sau khi reshape bằng thuật toán LZW encode.
- Lưu kết quả output là một chuỗi codewords và kích thước của ảnh (shape) vào file pickle.

Hàm lzw_decode:

- **Input:** link lưu chuỗi codewords sau khi encode, kèm kích thước ảnh và link để lưu ảnh output sao khi được decode.
- **Output:** ảnh sau khi được decode.
- Load chuỗi codewords và kích thước ảnh (shape).
- Tiếp theo, ta khởi tạo dictionary ban đầu với 256 cặp giá trị (code, string) tương tự như dictionary ở quá trình encode.
- Ta bắt đầu decode chuỗi codewords dựa trên thuật toán LZW decode, trả về kết quả là một chuỗi các giá trị (chuỗi này có giá trị giống như chuỗi các giá trị của ảnh sau khi được reshape(-1)).
- Từ chuỗi kết quả này, ta reshape lại ảnh như kích thước ảnh ban đầu.

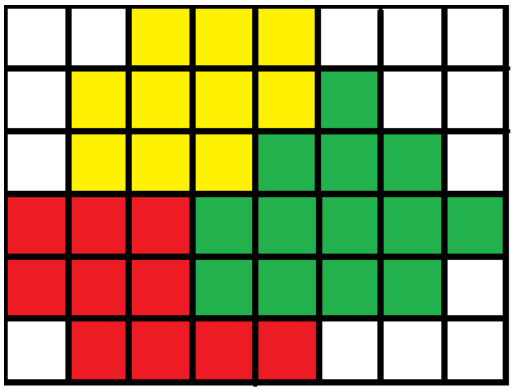


- Lưu ảnh output vào file bmp.

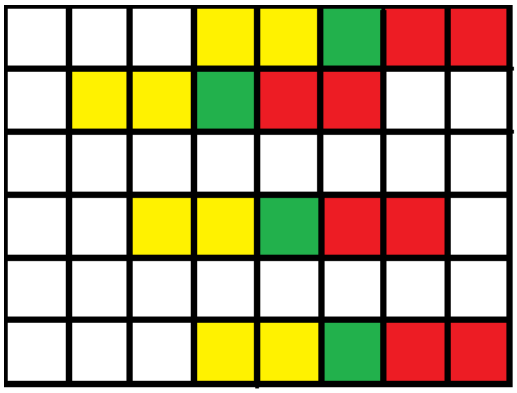
2.1.3 Đánh giá tính hiệu quả của thuật toán

- Do thuật toán LZW dựa trên ý tưởng đưa các chuỗi ký tự xuất hiện thường xuyên thành một codewords, nên khi thực hiện trên ảnh, thuật toán chỉ hiệu quả khi ảnh:

  + có nhiều chuỗi các pixel và mỗi pixel trong chuỗi đó **có cùng giá trị.**



  + có nhiều chuỗi các pixel **giống nhau** (về giá trị của mỗi pixel theo thứ tự của chúng trong chuỗi) xuất hiện trong ảnh.



- Từ nhận xét trên, ta có thể thấy thuật toán LZW chỉ hiệu quả trên một số ảnh nhất định, như ảnh có nhiều vùng pixel giống nhau (ví dụ như các ảnh từ phim hoạt hình), ảnh grayscale, ảnh nhị phân. Còn đối với ảnh ít có các vùng pixel giống nhau thì thuật toán LZW không hiệu quả.

- Một số ví dụ thuật toán hiệu quả:






Ảnh	Trước khi encode	Sau khi encode
1	11.2 KB	10.7 KB
2	1.54 MB	94.6 KB
3	1.68 MB	1.21 MB
4	1.26 MB	1.13 MB
5	4.82 MB	465 KB
6	1.71 MB	316 KB

- Một số ví dụ thuật toán **không** hiệu quả:



Ảnh	Trước khi encode	Sau khi encode
1	293 KB	459 KB
2	641 KB	0.98 MB
3	898 KB	1.09 MB

2.2 Thuật toán phân cụm K-means

2.2.1 Tìm hiểu về thuật toán

Tên phương pháp: Nén ảnh với thuật toán phân cụm K-means: Lượng tử hóa màu.

Ý tưởng: Dùng thuật toán phân cụm K-means để lượng tử hóa các màu có trong ảnh giúp nén ảnh mà không làm ảnh hưởng nhiều đến chất lượng của ảnh. Đây là một ứng dụng của thuật toán phân cụm K-means trong việc định lượng màu sắc có trong ảnh gốc.

Cho một ảnh màu với mỗi pixel có kích thước 3 bytes (RGB), trong đó mỗi màu có giá trị từ 0 đến 255. Như vậy, tổng số màu có thể được biểu thị là 256 * 256 * 256 màu. Trên thực tế, chúng ta chỉ có thể thấy được một vài màu trong một bức ảnh. Dưới đây là một ảnh 1280x720 pixel, chiếm 1,71 MB ở định dạng PNG. PNG là một kỹ thuật nén lossless cho ảnh. Mục tiêu là nén ảnh hơn nữa bằng cách sử dụng phương pháp lượng tử hóa màu, vì vậy đây là thuật toán nén lossy (lossy compression).

2.2.2 Phân tích và xây dựng chương trình

Thuật toán phân cụm K-means: Về cơ bản, đây là một thuật toán tối ưu hóa để tìm ‘k’ cụm trong tập hợp các điểm dữ liệu cho trước. Ban đầu, thuật toán sẽ chỉ định ngẫu nhiên các center cho k-cụm và sau đó trên cơ sở là các số liệu khoảng cách (ví dụ: khoảng cách euclide), giảm thiểu tổng khoảng cách bình phương của các điểm dữ liệu đến center. Các bước trong thuật toán phân cụm K-means:

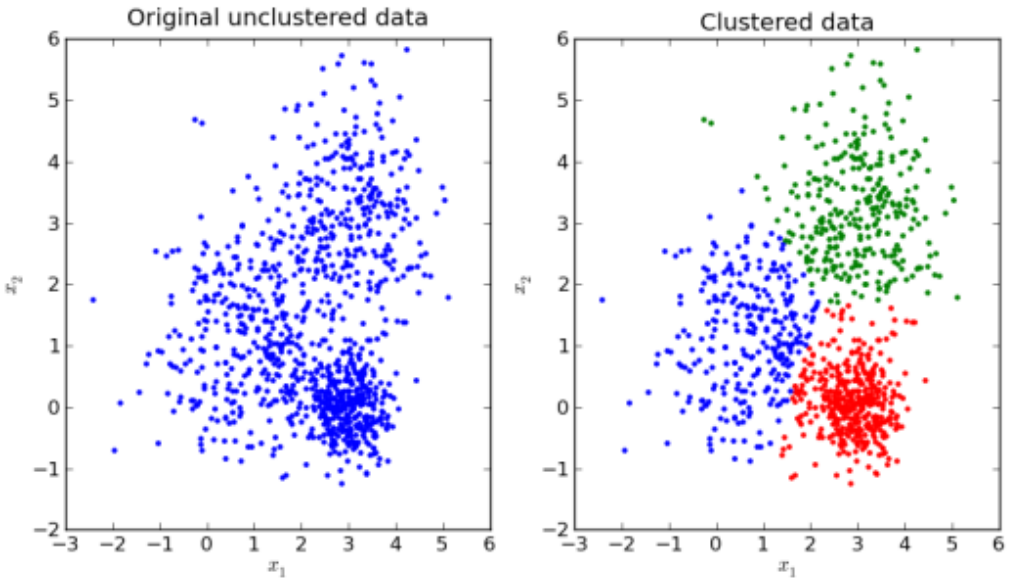
- Bước 1: Chọn ‘k’ điểm bất kỳ làm các center ban đầu.

- Bước 2: Phân mỗi điểm dữ liệu vào cụm có center gần nó nhất.

- Bước 3: Nếu việc gán dữ liệu vào từng cụm ở bước 2 không thay đổi so với vòng lặp trước nó thì ta dừng thuật toán.

- Bước 4: Cập nhật center cho từng cụm bằng cách lấy trung bình cộng của tất các các điểm dữ liệu đã được gán vào cụm đó sau bước 2.

- Bước 5: Quay lại bước 2.



Ví dụ minh họa thuật toán phân cụm K-means với k = 3

Với bài toán nén ảnh này, thuật toán phân cụm K-means sẽ nhóm các màu tương tự lại với nhau trong ‘k’ cụm (giả sử k = 128). Do đó, center của mỗi cụm là đại diện của các vector màu 3 chiều (RGB) thuộc các cụm tương ứng. ‘K’ center này sẽ thay thế cho tất cả các vector màu trong cụm của chúng, vì vậy chỉ giữ lại ‘k’ tổ hợp màu cho cả bức ảnh. Từ đó, chỉ cần giữ lại nhãn của từng pixel trong ảnh cho biết cụm mà pixel đó thuộc vào. Ngoài ra, giữ lại ‘k’ center như một bảng mã những màu còn lại trong ảnh nén.

Compression (Nén):

Chạy thuật toán K-means với k = 128 cho ảnh cần nén. Ảnh sau khi nén được lưu lại là mảng các nhãn cụm của từng pixel trong ảnh gốc. Codebook được lưu lại là mảng danh sách các center của các cụm (3-d RGB) có được sau khi chạy thuật toán K-means. Hai mảng trên được lưu với kiểu dữ liệu unsigned integer, có giá trị từ 0 đến 255, và giá trị của ‘k’ luôn nhỏ hơn 255.

```
from skimage import io
from sklearn.cluster import KMeans
import numpy as np

image = io.imread('tiger.png')
io.imshow(image)
io.show()

rows = image.shape[0]
cols = image.shape[1]

image = image.reshape(image.shape[0]*image.shape[1],3)
kmeans = KMeans(n_clusters = 128, n_init=10, max_iter=200)
kmeans.fit(image)

clusters = np.asarray(kmeans.cluster_centers_,dtype=np.uint8)
labels = np.asarray(kmeans.labels_,dtype=np.uint8 )
labels = labels.reshape(rows,cols);

np.save('codebook_tiger.npy',clusters)
io.imsave('compressed_tiger.png',labels);
```

Chúng ta có thể chọn một số “k” vừa đủ để thể hiện tốt màu sắc của một bức ảnh. Ở đây nhóm chọn k = 128. Điều này có nghĩa là tất cả các màu có trong ảnh gốc đã được lượng tử hóa lại thành 128 màu khác nhau. Ảnh sau khi được dựng lại từ ảnh đã nén (ảnh sau khi giải nén) sẽ có 128 màu này và phải trông tương tự như ảnh gốc.

Decompression (Giải nén):

Giải nén ảnh bằng cách gán các màu từ codebook đã lưu ở bước nén cho từng pixel tùy thuộc vào nhãn của chúng.

```
from skimage import io
import numpy as np

centers = np.load('codebook_tiger.npy')
c_image = io.imread('compressed_tiger.png')

image = np.zeros((c_image.shape[0],c_image.shape[1],3),dtype=np.uint8 )
for i in range(c_image.shape[0]):
    for j in range(c_image.shape[1]):
            image[i,j,:] = centers[c_image[i,j],:]
io.imsave('reconstructed_tiger.png',image);
io.imshow(image)
io.show()
```

Có thể thấy ảnh sau khi giải nén đã mất rất nhiều thông tin màu so với ảnh gốc nhưng nhìn chung vẫn không có nhiều sự khác biệt về mặt trực quan.

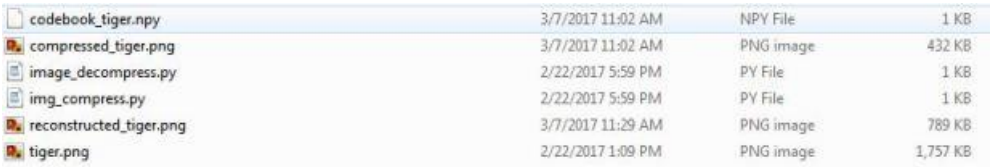
2.2.3 Đánh giá tính hiệu quả của thuật toán

- Đối với ảnh ở định dạng JPEG, nếu nén theo phương pháp này sẽ gặp lỗi vì vốn dĩ ảnh JPEG đã là lossy compression. Thuật toán nén của JPEG thay đổi các giá trị cường độ của mỗi pixel nên các pixel trong ảnh nén chứa nhãn có thể nhiều hơn ‘k’ và việc này sẽ dẫn đến lỗi.

- Vì thuật toán K-means tối ưu việc tìm các cụm trong tập dữ liệu đã cho nên thời gian thực thi sẽ tăng nếu kích thước ảnh tăng hoặc ‘k’ tăng.

- Có sự đánh đổi giữa thời gian thực thi và số lượng màu được thể hiện trong ảnh sau khi giải nén. Nếu ‘k’ lớn thì sau khi giải nén sẽ cho ra kết quả tốt hơn, ít mất mát dữ liệu hơn nhưng đồng thời sẽ mất nhiều thời gian để thực thi hơn.

- Kết quả thử nghiệm: Ảnh gốc (tiger.png) có kích thước là 1757 KB trong khi ảnh sau khi nén (compressed_tiger.png) và codebook chỉ có kích thước là 433 KB. Ảnh sau khi được dựng lại hay còn gọi là ảnh sau khi giải nén (reconstructed_tiger.png) cũng chiếm ít bộ nhớ hơn vì PNG chạy thuật toán nén riêng của nó. Vì hiện tại chỉ có 128 màu nên PNG có thể có tỷ lệ nén hơn 2.



codebook_tiger.npy	3/7/2017 11:02 AM	NPY File	1 KB
compressed_tiger.png	3/7/2017 11:02 AM	PNG image	432 KB
image_decompress.py	2/22/2017 5:59 PM	PY File	1 KB
img_compress.py	2/22/2017 5:59 PM	PY File	1 KB
reconstructed_tiger.png	3/7/2017 11:29 AM	PNG image	789 KB
tiger.png	2/22/2017 1:09 PM	PNG image	1,757 KB

- Có thể kết luận rằng thuật toán nén này chỉ được thực hiện bằng cách giảm số lượng màu có trong ảnh hay còn gọi là định lượng màu, chứ không giảm kích thước ảnh hay cường độ của pixel.

2.3 Thuật toán Huffman

2.3.1 Tìm hiểu về thuật toán Huffman Coding

Nén Huffman tĩnh là 1 dạng nén không mất dữ liệu (lossless data) được đề xuất bởi David A. Huffman dựa trên ý tưởng bảng tần suất các kí tự xuất hiện để xây dựng mã nhị phân cho các kí tự đó sao cho số bit là nhỏ nhất.

Ví dụ: ở ảnh grayscale kích thước 4x5 có 4 mức độ xám 0,1,2,3,4 với số lần xuất hiện tương ứng 3,6,7,2,3 . Mỗi điểm ảnh cần 8-bit để lưu trữ như trong bảng dưới đây:

Mức xám	Bit
0	00000000
1	00000001
2	00000010
3	00000011
4	00000100

Vậy cần dùng tổng cộng 8x2x5=160 bit

Nhưng khi nén và lưu ảnh dưới dãy bit sau khi áp dụng thuật toán Huffman, các mức xám được biểu diễn dưới dãy bit 0 và 1 sao cho mức xám có tần số suất hiện cao được biểu diễn dưới dạng dãy ít bit nhất như trong bảng dưới.

Mức xám	Bit – Huffman coding
0	110
1	10
2	0
3	1110
4	1111

Vậy cần tổng cộng: 9 bit cho mức xám 0, 12 bit cho mức xám 1, 7 bit cho mức xám 2, 4 bit cho mức xám 3, 12 bit cho mức xám 4, tổng cộng cần 44 bit lưu trữ.

2.3.2 Phân tích và xây dựng chương trình

- **File huffman.py**
  - Các hàm sử dụng:

    binaryTreePath(root):

    Input: root của cây nhị phân.

    Output: danh sách các mảng một chiều, mỗi mảng một chiều biểu diễn đường đi từ root đến lá và phần tử cuối cùng của mảng là giá trị độ xám.

  - Cấu trúc dữ liệu sử dụng:

    Node(): bao gồm các thuộc tính:

    Dense: mật độ mức xám trong ảnh.

    Grayvalue: giá trị độ xám.

    Isleaf: trả về True nếu đó là node lá và False nếu là không phải là node lá.

    Id: có giá trị 0 nếu nằm bên trái node cha hoặc 1 nếu nằm bên phải node cha.

    Left, Right: hai con trỏ dùng đề lưu vị trí các node con.

- **File de_huffman.py**
  - Các hàm sử dụng:

    Tree_builder(): xây dựng cây từ dictionary sau khi đã nén ảnh.

    binaryTreePath(root): hàm kiểm tra cây có được xây dựng thành công hay không.

    Find(): hàm truy suất đến tìm mức xám ở từng pixel của ảnh.

  - Cấu trúc dữ liệu sữ dụng:

    Node(): bao gồm các thuộc tính:

    Dense: mật độ mức xám trong ảnh.

    Grayvalue: giá trị độ xám.

Isleaf: trả về True nếu đó là node lá và False nếu là không phải là node lá.

Id: có giá trị 0 nếu nằm bên trái node cha hoặc 1 nếu nằm bên phải node cha.

Left, Right: hai con trỏ dùng đề lưu vị trí các node con.

2.3.3 Đánh giá tính hiệu quả của thuật toán

Thuật toán hiệu quả ở những bức ảnh có những vùng lớn có cùng một độ xám.

```
--------------------------------------
HUFFMAN COMPRESSION FOR GRAYSCALE IMAGE:
Total bits before compressing: 8000000
Total bits after compressing : 5274478
Compressed                   : 34.07  percent
Elodies-MacBook-Pro:doan elodiefloreari$
```

Ở những tấm ảnh có nhiều chi tiết, ít những mảng lớn có cùng độ xám, tính hiệu quả nén thấp.

```
----------------------------------------
HUFFMAN COMPRESSION FOR GRAYSCALE IMAGE:
Total bits before compressing: 7246464
Total bits after compressing : 6788808
Compressed                   : 6.32  percent
Elodies-MacBook-Pro:doan elodiefloreari$
```

BẢNG PHÂN CÔNG CÔNG VIỆC

Họ và tên	Công việc
Cao Quốc Đạt	Tìm hiểu, thực thi và đánh giá thuật toán lossless Huffman.
	Viết báo cáo phần thuật toán Huffman.
Hoàng Yến	Tìm hiểu, thực thi và đánh giá thuật toán lossy K-means.
	Viết báo cáo cho phần thuật toán K-means.
	Tổng hợp các source code và up lên github.
Huỳnh Vĩ Hà	Tìm hiểu, thực thi và đánh giá thuật toán lossless LZW.
	Viết báo cáo cho phần thuật toán LZW và phần giới thiệu.
	Tổng hợp báo cáo của các thành viên, căn chỉnh format.

TÀI LIỆU THAM KHẢO

II Multimedia Data Compression – Fundamentals of Multimedia (Ze-Nian Li and Mark S.Drew)