

**ĐẠI HỌC QUỐC GIA TP. HỒ CHÍ MINH**  
**TRƯỜNG ĐẠI HỌC BÁCH KHOA**  
**KHOA ĐIỆN – ĐIỆN TỬ**  
**BỘ MÔN VIỄN THÔNG**

-----o0o-----



**MÔN HỌC: XỬ LÝ ẢNH**

**BÁO CÁO ĐỀ TÀI**  
**TÌM HIỂU, KHẢO SÁT VÀ XÂY DỰNG ỨNG DỤNG**  
**THỬ NGHIỆM CỦA PHƯƠNG PHÁP NÉN TỪ ĐIỂN LZW**

**LỚP: A01-B**  
**NHÓM 9**  
**GVHD: TS. VÕ TUẤN KIỆT**

HỌ VÀ TÊN	MSSV
ĐINH PHƯỚC LỘC	1710178
NGUYỄN HOÀNG PHỤNG	1710246
KIỀU NGỌC ANH	1710464

**TP. HỒ CHÍ MINH, THÁNG 10 NĂM 2019**

## MỤC LỤC

<b>I.</b>	<b>TỔNG QUAN VỀ PHƯƠNG PHÁP LZW.....</b>	<b>3</b>
<b>II.</b>	<b>NGUYÊN TẮC CỦA PHƯƠNG PHÁP .....</b>	<b>3</b>
<b>III.</b>	<b>THUẬT TOÁN LZW.....</b>	<b>5</b>
1.	Quá trình nén .....	5
2.	Quá trình giải nén .....	7
3.	Trường hợp đặc biệt .....	8
4.	Đánh giá thuật toán LZW.....	8
<b>IV.</b>	<b>VÍ DỤ MINH HỌA.....</b>	<b>9</b>
<b>V.</b>	<b>VÍ DỤ LẬP TRÌNH TRÊN PYTHON:.....</b>	<b>11</b>
<b>VI.</b>	<b>ÁP DỤNG PHƯƠNG PHÁP NÉN TỪ ĐIỂN LZW VÀO MỘT ẢNH HAI CHIỀU ĐƠN GIẢN .....</b>	<b>13</b>
<b>VII.</b>	<b>NHẬN XÉT .....</b>	<b>15</b>
<b>VIII.</b>	<b>TÀI LIỆU THAM KHẢO .....</b>	<b>16</b>

## I. TỔNG QUAN VỀ PHƯƠNG PHÁP LZW

- Khái niệm nén từ điển được Jacob Lampel và Abraham Ziv đưa ra lần đầu tiên vào năm 1977. Sau đó phát triển thành một họ giải thuật nén từ điển LZ. Năm 1984 Terry Welch đã cải tiến thuật giải LZ thành một họ giải thuật mới hiệu quả hơn và đặt tên là LZW.
- Phương pháp LZW dựa trên việc xây dựng từ điển cho các “chuỗi ký tự” đã từng xuất hiện trong văn bản, những “chuỗi ký tự” xuất hiện sau đó sẽ được thay thế bằng mã của nó trong bảng từ điển.
- Giải thuật LZW được sử dụng cho tất cả các loại file nhị phân. Nó thường được dùng để nén các loại văn bản, ảnh đen trắng, ảnh màu ... và là chuẩn nén cho các dạng ảnh GIF, TIFF... Mức độ hiệu quả của LZW không phụ thuộc vào số bit màu của ảnh.

## II. NGUYÊN TẮC CỦA PHƯƠNG PHÁP

- Phương pháp LZW hoạt động theo nguyên tắc là tạo ra một từ điển động theo dữ liệu của file ảnh. Từ điển là tập hợp những cặp **Khoá** và **Nghĩa** của nó. Trong đó **Khoá** được sắp xếp theo thứ tự nhất định, **Nghĩa** là một chuỗi con trong dữ liệu ảnh.
- Từ điển được xây dựng đồng thời với quá trình đọc dữ liệu. Sự có mặt của một chuỗi con trong từ điển khẳng định rằng chuỗi đó đã từng xuất hiện trong phần dữ liệu đã đọc. Thuật toán liên tục tra cứu và cập nhật từ điển sau mỗi lần đọc một ký tự ở dữ liệu đầu vào.
- Do kích thước bộ nhớ không phải là vô hạn và để đảm bảo tốc độ tìm kiếm, người ta thường dùng từ điển với kích thước  $4096 (2^{12})$  phần tử.

- Cấu trúc từ điển có dạng như sau:

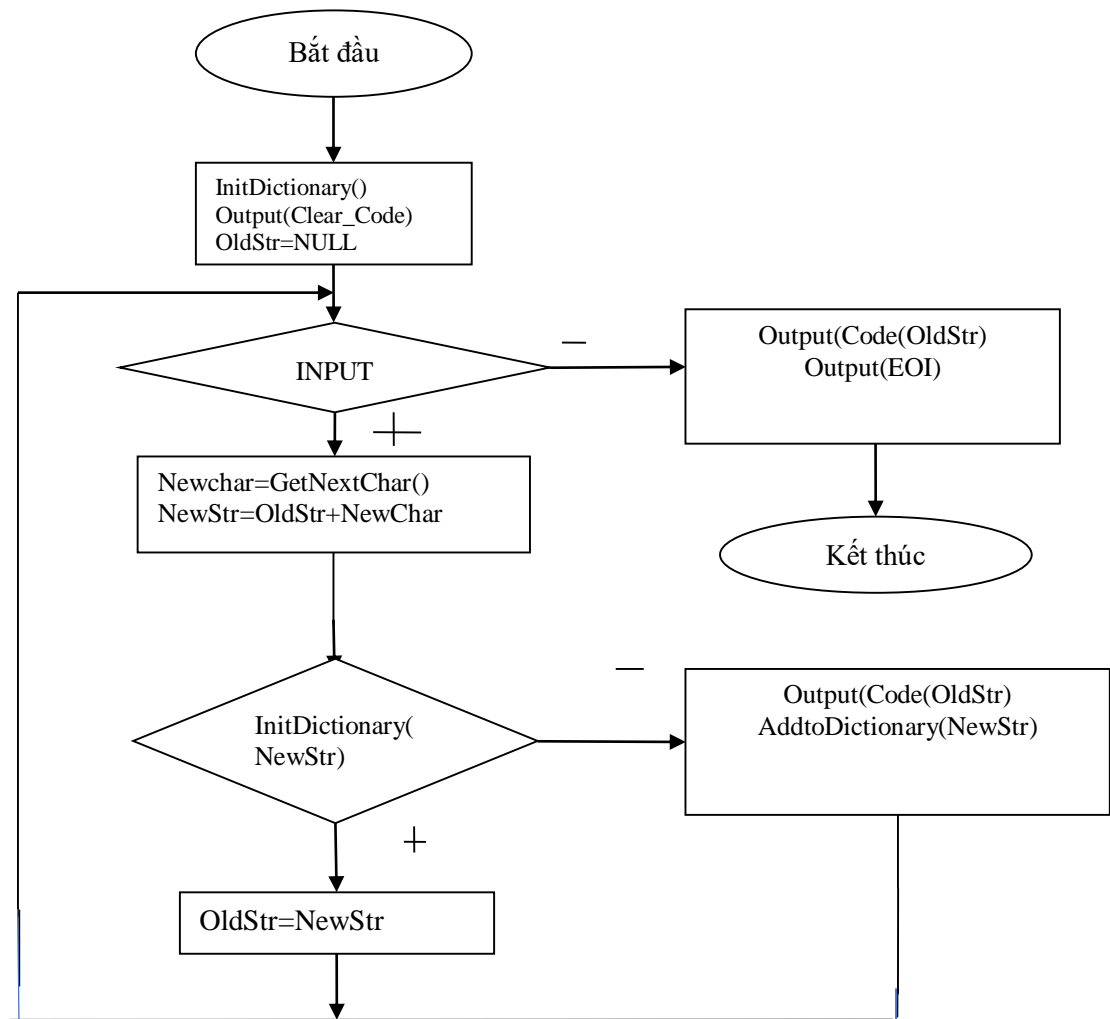
Khoá	Ý nghĩa	Ghi chú
0	0	
1	1	
...	...	
255	255	
256	256	<i>(Clear Code)</i>
257	257	<i>(End Of Information)</i>
258	Chuỗi	
259	Chuỗi	
...	...	
4095	Chuỗi	

- 256 từ mã đầu tiên có khoá từ 0 .. 255 chứa 256 ký tự của bảng mã ASCII).
- Từ mã thứ 256 chứa một mã đặc biệt là “mã xóa” (CC - Clear Code). Mục đích việc dùng mã xóa nhằm khắc phục tình trạng số mẫu lặp trong ảnh lớn hơn 4096. Khi đó một ảnh được quan niệm là nhiều mảnh ảnh, và từ điển là một bộ từ điển gồm nhiều từ điển con. Cứ hết một mảnh ảnh người ta lại gửi một mã xóa để báo hiệu kết thúc mảnh ảnh cũ, bắt đầu mảnh ảnh mới đồng thời khởi tạo lại từ điển cho mảnh ảnh mới.
- Từ mã thứ 257 chứa mã kết thúc thông tin (EOI - End Of Information). Một file ảnh có thể chứa nhiều ảnh (ví dụ ảnh GIF), khi đó mỗi ảnh sẽ được mã hóa riêng và mã EOI dùng để xác định điểm kết thúc thông tin của 1 ảnh.
- Các từ mã tiếp theo (từ 258 trở đi) chứa các mẫu lặp lại trong ảnh. Các từ mã này được sinh ra trong quá trình mã hoá.

### III. THUẬT TOÁN LZW

#### 1. Quá trình nén

Sơ đồ thuật toán:



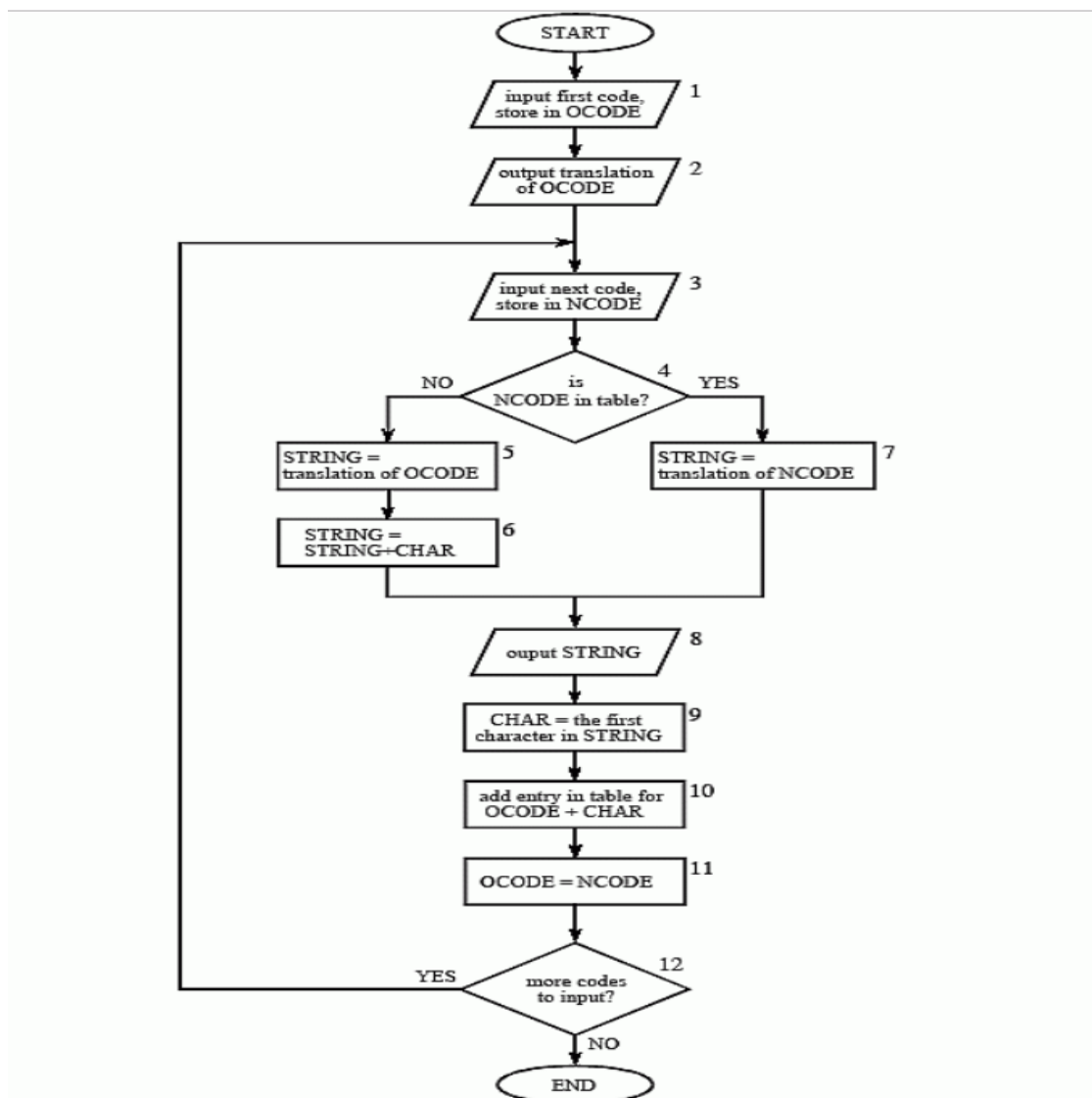
**Ý tưởng:** Nếu còn dữ liệu đầu vào thì tiếp tục đọc. Một chuỗi mới sẽ được tạo ra từ một chuỗi cũ (chuỗi này ban đầu rỗng, chuỗi này phải là chuỗi đã tồn tại trong từ điển) và kí tự vừa đọc vào, sau đó kiểm tra xem chuỗi mới đã có trong từ điển hay chưa. Mục đích của công việc này là hy vọng tìm được chuỗi có số ký tự lớn nhất đã tồn tại trong từ điển. Nếu tồn tại ta lại tiếp tục đọc một ký tự tiếp theo và lặp lại công việc. Nếu chưa có trong từ điển thì gửi chuỗi cũ ra ngoài và thêm chuỗi mới vào từ điển.

### **Giải thích ý nghĩa các hàm trong sơ đồ thuật toán:**

- Giá trị cờ INPUT=true khi vẫn còn dữ liệu đầu vào và ngược lại
- InitDictionary(): Hàm này có chức năng khởi tạo từ điển. Đặt giá trị cho 255 phần đầu tiên. Gán mã xóa(Clear Code) cho phần tử thứ 256 và mã kết thúc thông tin (End of Information) cho phần tử thứ 257. Xóa giá trị tất cả các phần tử còn lại
- Hàm Output(): gửi chuỗi bit ra file. Chuỗi bit này có độ dài là 9,10,11 hoặc 12 tùy thuộc vào vị trí trong từ điển của từ mã gửi ra. Các chuỗi bit này được nối tiếp vào với nhau
- Hàm GetNextChar(): Trả về một ký tự đầu vào. Hàm này cập nhật giá trị của cờ INPUT xác định xem còn dữ liệu đầu vào nữa hay không?
- Hàm AddtoDictionary() sẽ được gọi khi có một mẫu mới xuất hiện. Hàm này sẽ cập nhật mẫu này vào phần tử tiếp theo trong từ điển. Nếu từ điển đã đầy nó sẽ gửi ra từ mã xóa (Clear Code) và gọi đến hàm InitDictionary() để khởi tạo lại từ điển.
- Hàm Code(): Trả về giá trị tương ứng với chuỗi.

## 2. Quá trình giải nén

Giải thuật giải nén gần như tương tự với giải thuật nén. Với giải thuật giải nén, một từ mã ứng với một chuỗi sẽ được ghi ra tệp khi chuỗi ghép bởi chuỗi trên với ký tự vừa đọc chưa có trong từ điển. Người ta cũng cập nhật ngay vào từ điển từ mã ứng với chuỗi tạo bởi chuỗi cũ với ký tự vừa đọc. Ký tự này đồng thời là ký tự đầu tiên trong chuỗi ứng với từ mã sẽ được ghi ra tiếp theo. Đây là điểm mấu chốt cho phép xây dựng thuật toán giải nén.



### 3. Trường hợp đặc biệt

- Giải thuật LZW như trên có thể dẫn đến một tình huống là quá trình nén thì thực hiện được, còn quá trình giải nén thì “không” thực hiện được.
- Tình huống đó xảy ra như sau: giả sử  $c$  là một ký tự,  $S$  là một chuỗi (có độ dài lớn hơn 0); mã  $k$  của từ điển chứa giá trị là  $cS$ . Nếu chuỗi vào tiếp theo là  $cScSc$ , khi đọc đến  $cSc$  chương trình sẽ tạo mã  $k'$  chứa  $cSc$  (vì  $cS$  đã có mã là  $k$ ), theo thuật toán nén ký tự  $c$  được giữ lại và chương trình đọc tiếp xâu  $S$ , ký tự  $c$  để hình thành từ  $cSc$ , khi đó mã  $k'$  sẽ được dùng thay cho  $cSc$ .
- Trong chương trình giải nén,  $k'$  sẽ xuất hiện trước khi nó được định nghĩa. Rất may là từ mã vừa đọc trong trường hợp này bao giờ cũng có nội dung trùng với tổ hợp của từ mã cũ với ký tự đầu tiên của nó. Điều này giúp cho quá trình cài đặt chương trình khắc phục được trường hợp ngoại lệ một cách dễ dàng.

### 4. Đánh giá thuật toán LZW

Thuật toán LZW đặc biệt có hiệu quả khi sử dụng để nén file văn bản vì độ lặp lại của ký tự là lớn.

- Tỷ lệ nén:  $2 \div 5$
- Độ phức tạp: Đơn giản
- Tốc độ nén: Trung bình
- Ứng dụng: Áp dụng cho tất cả các file nhị phân. Thường dùng để nén các loại văn bản, ảnh đen trắng, ảnh màu, ảnh đa mức xám...và là chuẩn nén cho các định dạng ảnh GIF và TIFF. Mức độ hiệu quả của LZW không phụ thuộc vào số bit màu của ảnh.



#### IV. VÍ DỤ MINH HỌA

1. Cho chuỗi đầu vào là “ABCBCABCABCD” (Biết mã ASCII của A - 65; B - 66; C - 67; D - 68).

a) **Quá trình nén:** Từ điển được khởi tạo đến khoá 257, và tiếp tục được xây dựng trong quá trình mã hoá chuỗi vào như sau:

Vào	Ra	Thực hiện
A (65)		A đã có trong từ điển (TD) => đọc tiếp
B (66)	65	AB chưa có trong TD => Thêm vào TD mã 258 đại diện cho AB
C (67)	66	BC chưa có trong TD => Thêm vào TD mã 259 đại diện cho BC
B	67	CB chưa có trong TD => Thêm vào TD mã 260 đại diện cho CB
C		BC đã có trong từ điển (TD) => đọc tiếp
A	259 (BC)	BCA chưa có trong TD => Thêm vào TD mã 261 đại diện cho BCA
B		AB đã có trong TD => đọc tiếp
C	258 (AB)	ABC chưa có trong TD => Thêm vào TD mã 262 đại diện cho ABC
A	67	CA chưa có trong TD => Thêm vào TD mã 263 đại diện cho CA
B		AB đã có trong TD => đọc tiếp
C		ABC đã có trong TD => đọc tiếp
D	262 (ABC)	ABCD chưa có trong TD => Thêm vào TD mã 264 đại diện cho ABCD.
EOF	68	Kết thúc file

**Kết quả thu được chuỗi mã hóa:**

65 – 66 – 67- 259 – 258 – 67 – 262 – 68

b) **Quá trình giải nén:** Khởi tạo từ điển với 258 từ mã (từ 0-257) như quá trình mã hóa.

Ví dụ: Giải nén chuỗi 65 – 66 – 67- 259 – 258 – 67 – 262 – 68

Quá trình giải nén được thực hiện như sau:

Vào	Ra	Thực hiện
A (65)		A đã có trong từ điển (TD) => đọc tiếp
B (66)	A	AB chưa có trong TD => Thêm vào TD mã 258 đại diện cho AB (ghi ra A, giữ lại B)
C (67)	B	BC chưa có trong TD => Thêm vào TD mã 259 đại diện cho BC (ghi ra B, giữ lại C)
BC (259)	C	CB chưa có trong TD => Thêm vào TD mã 260 đại diện cho CB (ghi ra C giữ lại BC)
AB (258)	BC	BCA chưa có trong TD => Thêm vào TD mã 261 đại diện cho BCA (ghi ra BC, giữ lại AB)
C (67)	AB	ABC chưa có trong TD => Thêm vào TD mã 262 đại diện cho ABC (ghi ra AB, giữ lại C)
ABC (262)	C	CA chưa có trong TD => Thêm vào TD mã 263 đại diện cho CA (ghi ra C, giữ lại ABC)
D (68)	ABC	ABCD chưa có trong TD => Thêm vào TD mã 264 đại diện cho ABCD (ghi ra ABC, giữ lại D)
EOF	D	Kết thúc file - ghi ra D

**Kết quả thu được: “ABCBCABCABCD”**

2. Ví dụ cho trường hợp đặc biệt: Cho chuỗi “ABABCD**ABCABCA**E”.

a) Quá trình nén: Khởi tạo 258 từ mã cho bảng mã ASCII + CC + EOI

Vào	Ra	Thực hiện
A		A có => TT
B	A (65)	AB k => 258= AB, ghi A, giữ B
A	B (66)	BA k => 259= BA, ghi B, giữ A
B		AB có => TT
C	AB (258)	ABC k => 260=ABC, ghi AB, giữ C
D	C (67)	CD k => 261=CD, ghi C, giữ D
A	D (68)	DA k => 262=DA, ghi D, giữ A
B		AB có => TT
C		ABC có => TT
A	ABC (260)	ABCA k => 263=ABCA, ghi ABC, giữ A
B		AB có => TT
C		ABC có => TT
A		ABCA có => TT
E	263	ABCAE k => 264=ABCAE, ghi ABCA, giữ E
EOF	E (69)	Kết thúc

**Kết quả thu được:** 65 – 66 – 258 – 67 – 68 – 260 – 263

b) Quá trình giải nén:

Vào	Ra	Thực hiện
65 (A)		A có =>TT
66 (B)	A	AB k => 258=AB, ghi A, giữ B
258 (AB)	B	BA k => 259=BA, ghi B, giữ AB
67 (C)	AB	ABC k => 260=ABC, ghi AB, giữ C
68 (D)	C	CD k => 261=CD, ghi C, giữ D
260 (ABC)	D	DA k => 262=DA, ghi D, giữ ABC
263 (?)	ABCA	263 chưa có => 263= ABCA, ghi ABC, giữ ABCA
69 (E)	ABCA	ABCAE k => 264=ABCAE, ghi ABCA, giữ E
EOF	E	Kết thúc

**Kết quả thu được:** ABABCDABCABCAE.

## V. VÍ DỤ LẬP TRÌNH TRÊN PYTHON:

Code:

```
def compress(uncompressed):
    """Compress a string to a list of output symbols."""

    # Build the dictionary.
    dict_size = 256
    dictionary = {chr(i): i for i in range(dict_size)}

    w = ""
    result = []
    for c in uncompressed:
        wc = w + c
        if wc in dictionary:
            w = wc
        else:
            result.append(dictionary[w])
            # Add wc to the dictionary.
            dictionary[wc] = dict_size
            dict_size += 1
            w = c

    # Output the code for w.
    if w:
```

```

        result.append(dictionary[w])
    return result

def decompress(compressed):
    """Decompress a list of output ks to a string."""
    from io import StringIO

    # Build the dictionary.
    dict_size = 256
    dictionary = {i: chr(i) for i in range(dict_size)}

    # use StringIO, otherwise this becomes O(N^2)
    # due to string concatenation in a loop
    result = StringIO()
    w = chr(compressed.pop(0))
    result.write(w)
    for k in compressed:
        if k in dictionary:
            entry = dictionary[k]
        elif k == dict_size:
            entry = w + w[0]
        else:
            raise ValueError('Bad compressed k: %s' % k)
        result.write(entry)

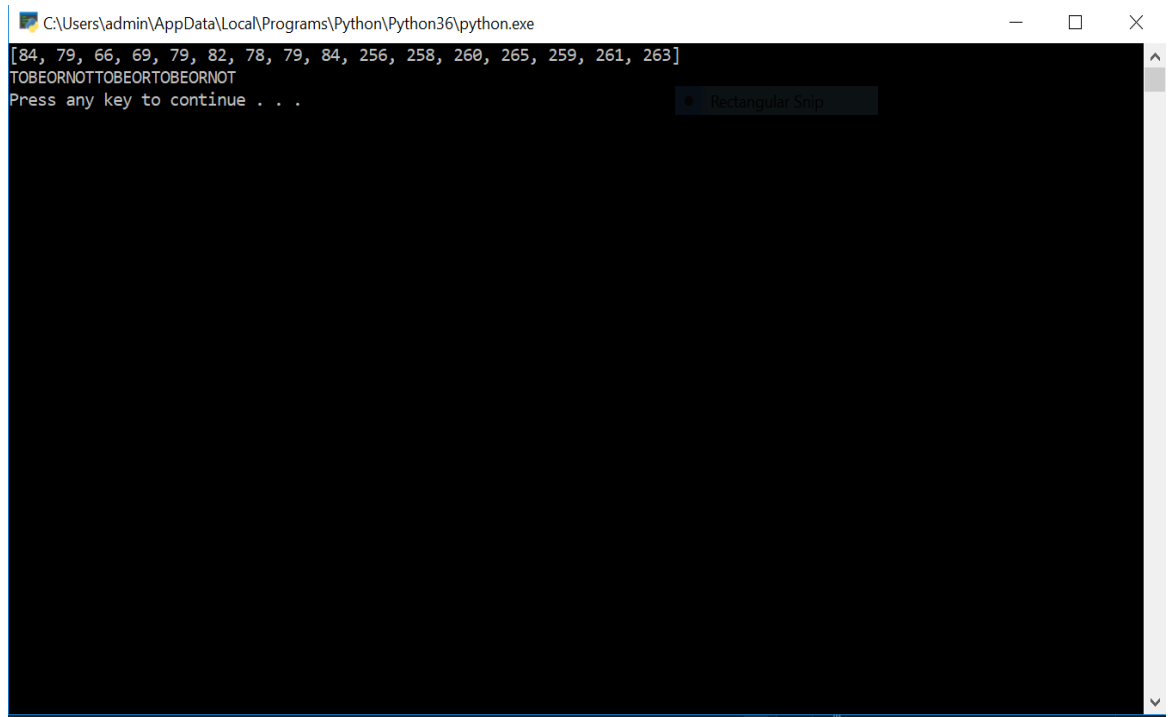
        # Add w+entry[0] to the dictionary.
        dictionary[dict_size] = w + entry[0]
        dict_size += 1

    w = entry
    return result.getvalue()

compressed = compress('TOBEORNOTTOBEORTOBEORNOT')
print (compressed)
decompressed = decompress(compressed)
print (decompressed)

```

KẾT QUẢ:



## VI. ÁP DỤNG PHƯƠNG PHÁP NÉN TỪ ĐIỂN LZW VÀO MỘT ẢNH HAI CHIỀU ĐƠN GIẢN

*ĐỀ BÀI: CHO ẢNH SAU (MỨC XÁM [0,255]):*

```
[ 25 66 83 75
 37 22 41 89
 15 22 55 67
 75 88 255 254]
```

a) Nén ảnh

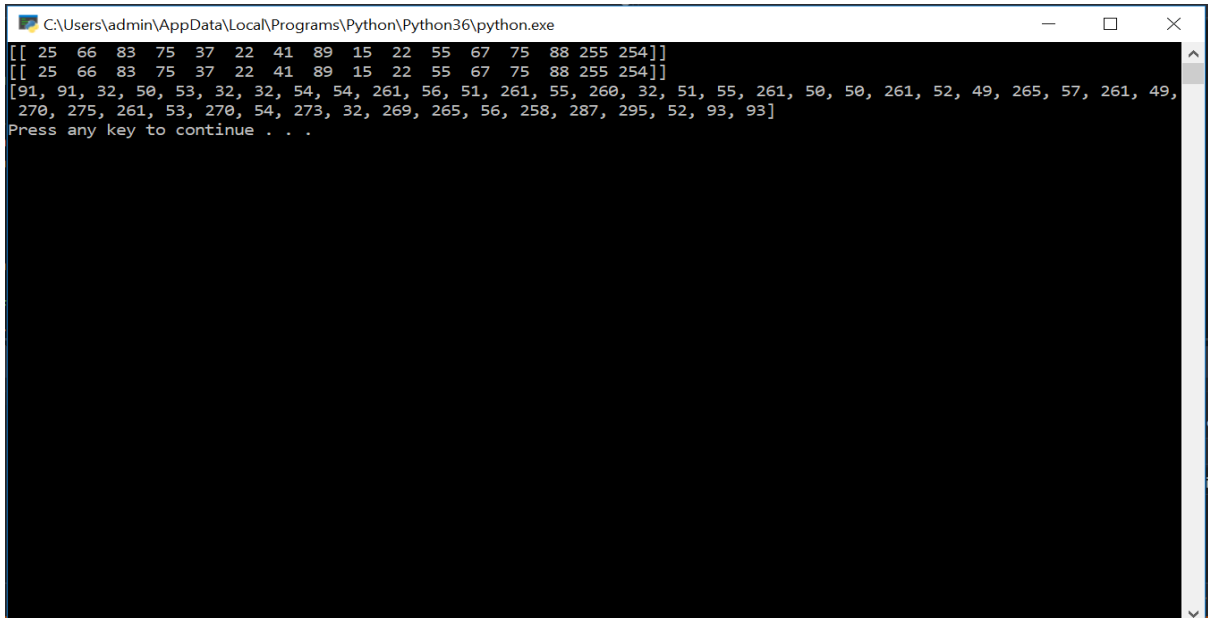
```
import numpy as np
a = np.array([[25,66,83,75],
              [37,22,41,89],
              [15,22,55,67],
              [75,88,255,254]]).reshape(1,16)
print(a)
```

```

b = np.array2string(a)
print(b)
#Using function "compressed" as above
compressed = compress(b)
print (compressed)

```

KẾT QUẢ:



```

C:\Users\admin\AppData\Local\Programs\Python\Python36\python.exe
[[ 25  66  83  75  37  22  41  89  15  22  55  67  75  88 255 254]]
[[ 25  66  83  75  37  22  41  89  15  22  55  67  75  88 255 254]]
[91, 91, 32, 50, 53, 32, 32, 54, 54, 261, 56, 51, 261, 55, 260, 32, 51, 55, 261, 50, 50, 261, 52, 49, 265, 57, 261, 49,
270, 275, 261, 53, 270, 54, 273, 32, 269, 265, 56, 258, 287, 295, 52, 93, 93]
Press any key to continue . . .

```

*GIẢI THÍCH:* Trong chuỗi b có xuất hiện các ký tự “ ” và [] nên độ dài chuỗi nén có dài hơn so với độ dài chuỗi gốc (46 ký tự so với 36 ký tự ở chuỗi gốc).

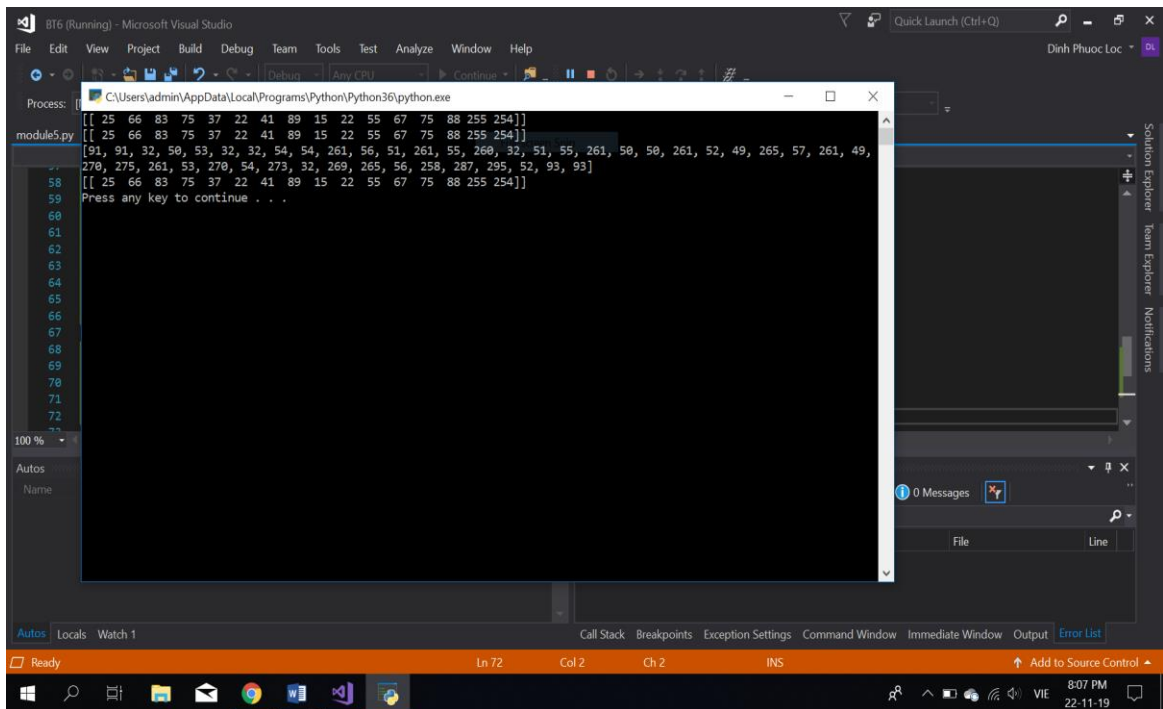
b) Giải nén

```

decompressed = decompress(compressed)
print (decompressed)

```

KẾT QUẢ:



```
Process: C:\Users\admin\AppData\Local\Programs\Python\Python36\python.exe
module5.py
[[ 25 66 83 75 37 22 41 89 15 22 55 67 75 88 255 254]]
[[ 25 66 83 75 37 22 41 89 15 22 55 67 75 88 255 254]]
[91, 91, 32, 50, 53, 32, 32, 54, 54, 261, 56, 51, 261, 55, 260, 32, 51, 55, 261, 50, 50, 261, 52, 49, 265, 57, 261, 49,
270, 275, 261, 53, 270, 54, 273, 32, 269, 265, 56, 258, 287, 295, 52, 93, 93]
[[ 25 66 83 75 37 22 41 89 15 22 55 67 75 88 255 254]]
Press any key to continue . . .
```

**KẾT LUẬN:** Giải nén thành công, chuỗi được trả về giá trị ban đầu.

## VII. NHẬN XÉT

- Thuật toán này sử dụng chuỗi nén có độ dài cố định là 12 bit để mã hóa cho 1 byte dữ liệu có độ dài là 8 bit. Như thế dung lượng tăng lên 4 bit. Điều này không thực sự hiệu quả, một số trường hợp file nén có dung lượng lớn hơn file ban đầu. Nói chung tỉ lệ nén sẽ thấp. Ta có thể dùng chuỗi bit có độ dài thay đổi để mã hóa cho 1 byte dữ liệu. Chuỗi bit này ban đầu có độ dài 9 bit và sẽ tăng dần lên 10, 11, và 12 bit. Điều này có thể làm tăng tỉ lệ nén, nhưng sẽ có khó khăn khi giải nén vì khi đọc dữ liệu đã được nén để giải nén thì ta phải xác định tại thời điểm đó ta phải đọc 9 bit hay 10 bit,...
- Đây là một trong các thuật toán nén thế hệ thứ nhất. Ngoài ra còn có các thuật toán nén thế hệ thứ hai, hiệu quả hơn nhưng đồng thời độ phức tạp cũng tăng lên rất nhiều.

## **VIII. TÀI LIỆU THAM KHẢO**

1. Digital Image Processing- Gonzales (Chương 8)
2. Wikipedia (keyword: Lempel-Ziv-Welch, LZW)
3. The Scientist and Engineer's Guide to Digital Signal Processing – Steven W.Smith (chapter 27).