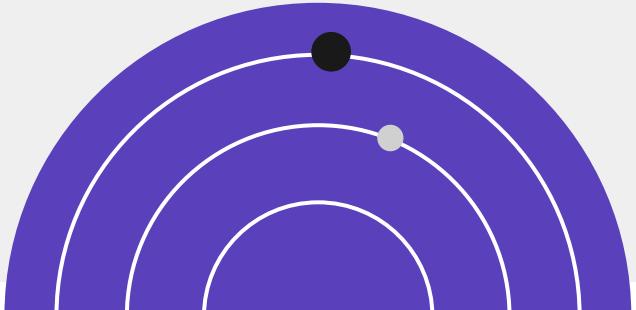
TEXT AND IMAGE COMPRESSION USING HUFFMAN, LZW AND SHANNON FANO

Giảng viên: Đỗ Văn Tiến
Thành viên:
20522127- Trần Vĩnh Tuân
20521826- Lữ Thị Thúy Quỳnh
20521990- Bùi Văn Thuận

Nội dung

01

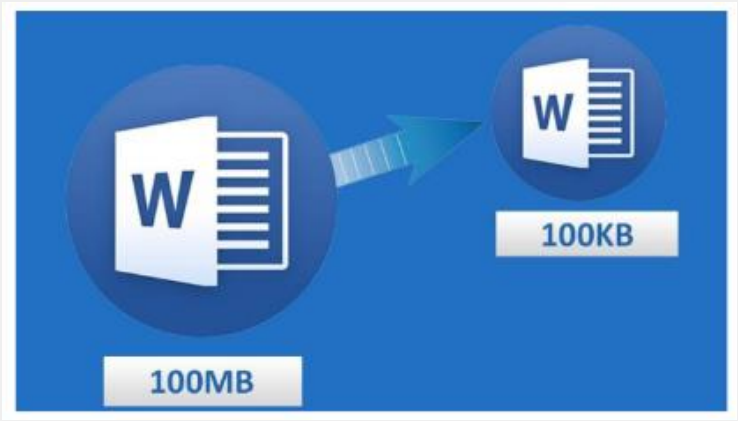
GIỚI THIỆU BÀI TOÁN



Giới thiệu bài toán

Nén dữ liệu tức là là việc chuyển định dạng thông tin sử dụng ít bit hơn cách thể hiện ở dữ liệu gốc.

Giới thiệu bài toán

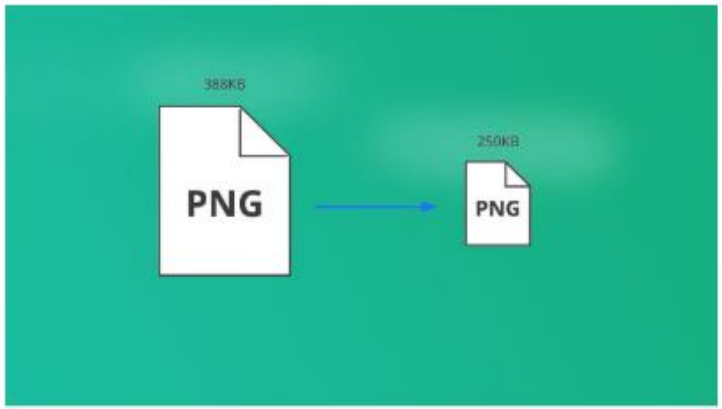


Nén tệp văn bản

Input: Một tập tin văn bản có định dạng là *.txt.
Output: Một file *.txt chứa thông tin đã mã hóa của tệp văn bản đầu vào với kích thước file nhỏ hơn.

Nén hình ảnh

Input: Một ảnh màu có định dạng là *.jpg.
Output: Một file *.txt chứa thông tin đã mã hóa của ảnh đầu vào với kích thước dữ liệu nhỏ hơn.

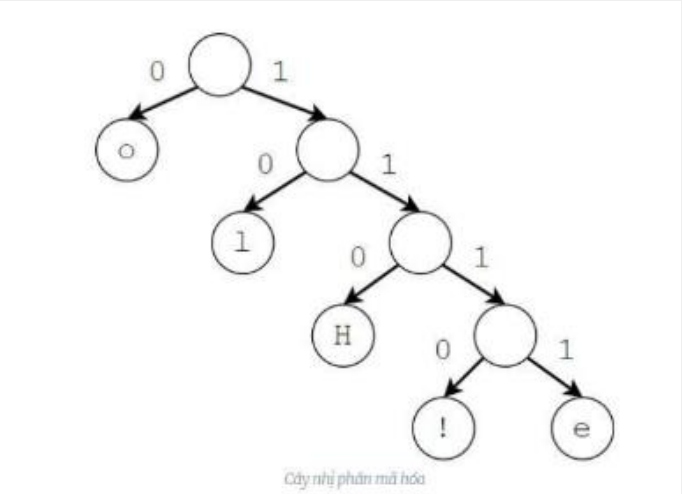


02

HUFFMAN

Giới thiệu

- Mã hóa Huffman (David A. Huffman)là một thuật toán mã hóa dùng để nén dữ liệu.
- Dựa trên bảng tần suất xuất hiện các kí tự cần mã hóa để xây dựng một bộ mã nhị phân cho các kí tự đó sao cho dung lượng (số bit) sau khi mã hóa là nhỏ nhất.



Cây nhị phân mã hóa

Quá trình nén

Với ý tưởng trên, thuật toán Huffman coding gồm 3 bước:

B1: Đếm tần suất xuất hiện của các phần tử trong chuỗi đầu vào.

B2: Xây dựng cây Huffman(Cây nhị phân mã hóa)

- **B2.1.** Tạo danh sách chứa các nút lá bao gồm phần tử đầu vào và trọng số nút là tần số xuất hiện của nó.
- **B2.2.** Từ danh sách này, lấy ra 2 phần tử có tần suất xuất hiện ít nhất. Sau đó gắn 2 nút vừa lấy ra vào một nút gốc mới vô trọng số là tổng của 2 trọng số ở nút vừa lấy ra để tạo thành một cây.
- **B2.3.** Đẩy cây mới vào lại danh sách.
- **B2.4.** Lặp lại bước 2 và 3 cho đến khi danh sách chỉ còn 1 nút gốc duy nhất của cây.
- **B2.5.** Nút còn lại chính là nút gốc của cây Huffman

B3: Từ cây Huffman, ta có các giá trị mã hóa. Lúc này, ta có thể xây dựng chuỗi mã hóa từ các giá trị này.

VD1: Nén ký tự:
“ABABBCBBDEEEABABBAEEDDCCABBBCDEEDCBCCCCDBBBCAAA”

Ký tự	Tần suất
A	9
B	15
C	10
D	6
E	7

Ký tự	Tần suất
B	15
C	10
A	9
E	**7**
D	**6**

Ký tự	Tần suất
B	15
DE	**13**
C	**10**
A	**9**

Ký tự	Tần suất
AC	**19**
B	**15**
DE	**13**

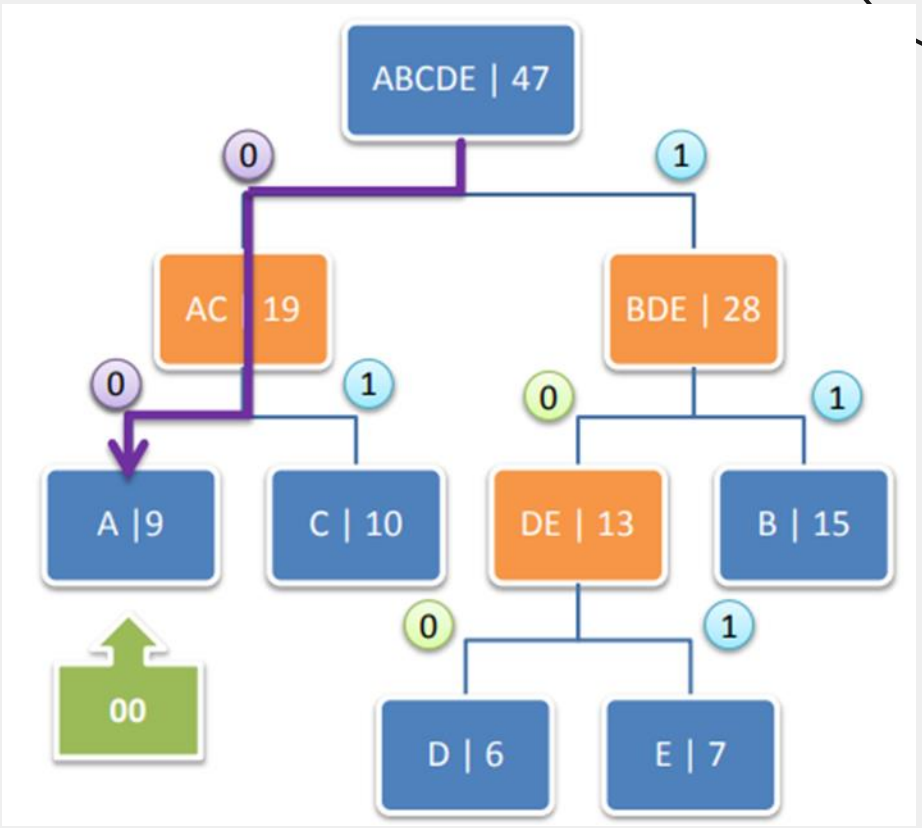
Ký tự	Tần suất
BDE	**28**
AC	**19**

Ký tự	Tần suất
ABCDE	**47**

Ký tự	Mã Huffman
A	00
B	11
C	01
D	100
E	101

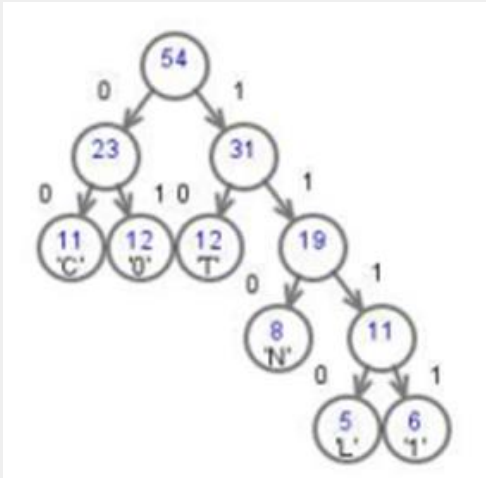
Chuỗi nén được thành :

- F=”00110011110111111001011011010
  01100111100101101100100010100111
  11101100101101100011101010101100
  11111101000000”
- Tiết kiệm: 8*47 - (2*9 + 2*15 + 2*10 + 3*6 + 3*7) = 376 - 107 = 269 bit



Quá trình giải nén

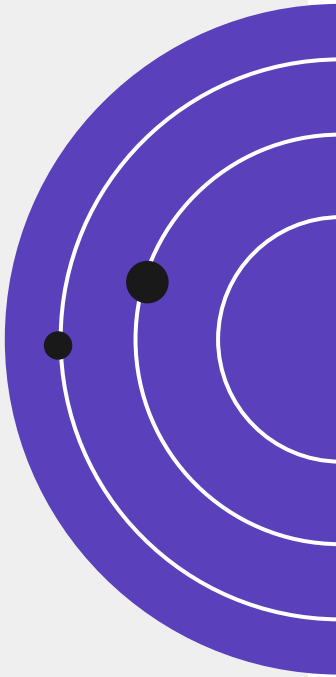
Các bước thực hiện:
Bước 1: Xây dựng lại cây Huffman từ thông tin giải mã đã lưu.
Bước 2: Duyệt file, đọc lần lượt từng bit trong file nén và duyệt cây
Bước 3: Xuất ký tự tương ứng khi duyệt hết nút lá.
Bước 4: Thực hiện B2, B3 cho đến khi duyệt hết file.
Bước 5: Xuất file đã giải nén.



Input:00110101011110111111111010011100000110110101010111101010101000000001110111011110000000101010101111111101010111010110110110010101011011010
Output:CNTT10110CLCCNNTTT10000CCCCLLLLCCCTTTT11000 NTNNN000TNT

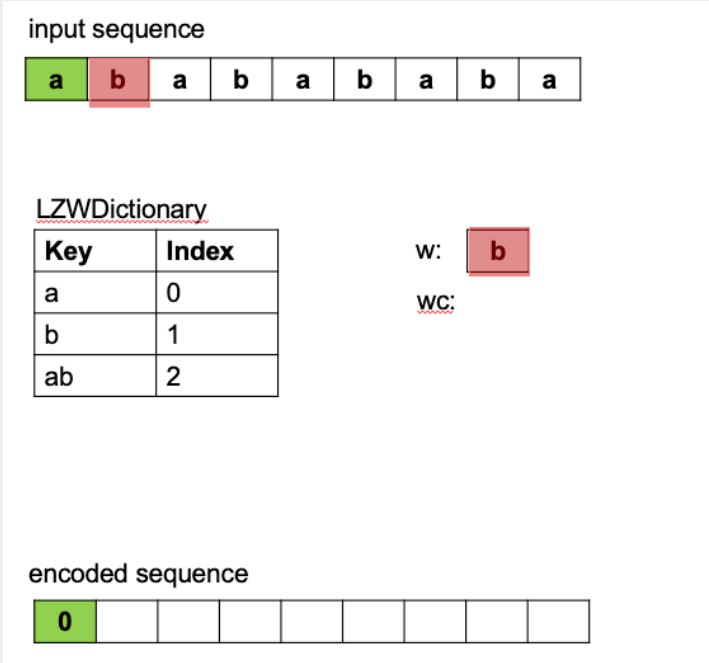
03 LZW

2.1 Quá trình nén
2.2 Quá trình giải nén
2.3 Ưu điểm và nhược điểm



Khái niệm

Lempel – Ziv – Welch (LZW) là một thuật toán nén dữ liệu không mất dữ liệu phổ quát được tạo ra bởi Abraham Lempel , Jacob Ziv và Terry Welch . Nó được Welch công bố vào năm 1984 như là một cách triển khai cải tiến của thuật toán LZ78 do Lempel và Ziv xuất bản vào năm 1978



2.1 Quá trình nén

- **Nguyên tắc hoạt động**
- Một xâu kí tự là một tập hợp từ hai kí tự trở lên.
- Nhớ tất cả các xâu kí tự đã gặp và gán cho nó một mã hóa riêng.
- Nếu lần sau gặp lại xâu kí tự đó, xâu kí tự sẽ được thay thế bằng mã hóa của nó.
- Tạo một mảng rất lớn dùng để lưu giữ các xâu kí tự đã gặp.(từ điển)
- Khi các byte dữ liệu cần nén được đem đến, chúng liền được giữ lại trong một bộ đệm chứa và đem so sánh với các chuỗi đã có trong "từ điển"
- Nếu chuỗi dữ liệu trong bộ đệm chứa không có trong "từ điển" thì nó được bổ sung thêm vào "từ điển" và chỉ số của chuỗi ở trong "từ điển" chính là mã hóa của chuỗi.
- Nếu có rồi thì mã hóa của chuỗi được đem ra thay cho chuỗi ở dòng dữ liệu ra.

2.1 Quá trình nén

Các bước trong quá trình nén:

1. Khởi tạo từ điển ban đầu chứa các ký tự đơn (nếu là mã ASCII thì ban đầu có 256 ký tự).
2. Đọc vào một chuỗi ký tự từ tệp tin ban đầu.
3. Tiến hành nén chuỗi ký tự đó bằng cách tìm chuỗi con trong từ điển trùng với chuỗi được đọc vào.
4. Nếu chuỗi con đó chưa có trong từ điển, thì thêm chuỗi con đó vào từ điển và đưa ra mã hóa của chuỗi ký tự trước đó.
5. Nếu chuỗi con đó đã có trong từ điển, thì sử dụng mã hóa đã có của chuỗi con đó để nén chuỗi ký tự trước đó.
6. Lưu các thông tin về mã hóa vào tệp tin đích.

2.1 Quá trình nén

- Ý tưởng code:

Khởi tạo bảng với các chuỗi ký tự đơn
P = ký tự đầu vào đầu tiên
WHILE không kết thúc luồng đầu vào
    C = ký tự đầu vào tiếp theo
    NẾU P + C nằm trong bảng chuỗi
        P = P + C
    ELSE
        xuất mã cho P
        thêm P + C vào bảng chuỗi
        P = C
END WHILE

```
def encoding(file_path):
    with open(file_path, 'r') as f:
        s1 = f.read()
    table = defaultdict(int)
    for i in range(256):
        ch = chr(i)
        table[ch] = i

    p = s1[0]
    code = 256
    output_code = []
    for i in range(len(s1)):
        if i != len(s1) - 1:
            c = s1[i + 1]
        if p + c in table:
            p = p + c
        else:
            output_code.append(table[p])
            #print(p, table[p])
            table[p + c] = code
            code += 1
            p = c
        c = ""
    output_code.append(table[p])
    #print(p, table[p])

    return output_code
```

2.1 Quá trình nén

VD: Mã hóa dãy kí tự sau : BABAABAAA

1. **BA** is not in the Dictionary; insert **BA**, output the code for its prefix: **code(B)**
2. **AB** is not in the Dictionary; insert **AB**, output the code for its prefix: **code(A)**
3. **BA** is in the Dictionary.
   BAA is not in Dictionary; insert **BAA**, output the code for its prefix: **code(BA)**
4. **AB** is in the Dictionary.
   ABA is not in the Dictionary; insert **ABA**, output the code for its prefix: **code(AB)**
5. **AA** is not in the Dictionary; insert **AA**, output the code for its prefix: **code(A)**
6. **AA** is in the Dictionary and it is the last pattern; output its code: **code(AA)**

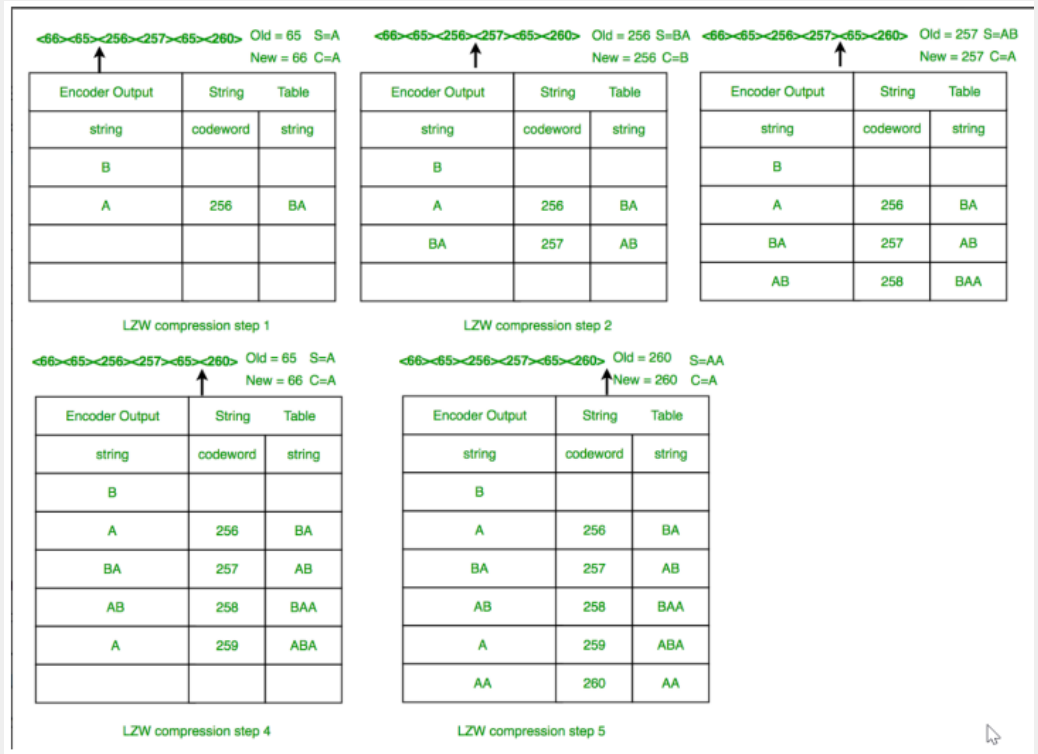
output	Dictionary	
	Code Word	String
66	256	BA
65	257	AB
256	258	BAA
257	259	ABA
65	260	AA
260		

The compressed message is: **<66><65><256><257><65><260>**

2.2 Quá trình giải nén

Bộ giải nén LZW tạo cùng một bảng chuỗi trong quá trình giải nén. Nó bắt đầu với 256 mục nhập bảng đầu tiên được khởi tạo thành các ký tự đơn.

Bảng chuỗi được cập nhật cho từng ký tự trong luồng đầu vào, ngoại trừ ký tự đầu tiên. Giải mã được thực hiện bằng cách đọc mã và dịch chúng thông qua bảng mã đang được xây dựng.



2.2 Quá trình giải nén

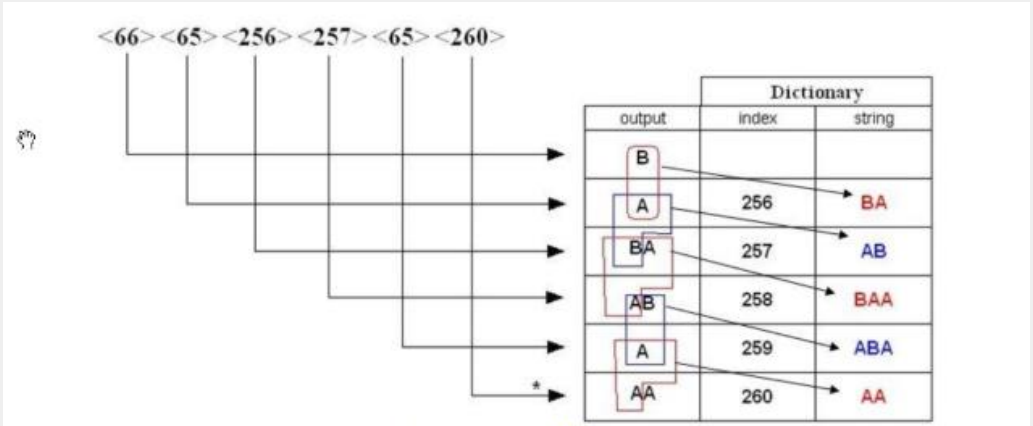
```
def decoding(op, file_path):
    # Khởi tạo từ điển ban đầu với các giá trị mặc định là chuỗi rỗng
    table = defaultdict(str)
    for i in range(256):
        ch = chr(i)
        table[i] = ch
    # Khởi tạo các biến ban đầu
    old = op[0]  # Giá trị đầu tiên trong op
    s = table[old]  # Giá trị tương ứng với old trong từ điển
    c = s[0]  # Ký tự đầu tiên của s
    count = 256  # output code cho những giá trị không có trong từ điển

    # Mở tệp tin để ghi dữ liệu giải nén
    with open(file_path, 'w') as f:
        # Ghi giá trị ban đầu của s vào tệp tin
        f.write(s)
        # Lặp qua các phần tử trong op (trừ phần tử đầu tiên)
        for i in range(len(op) - 1):
            n = op[i + 1]  # Giá trị hiện tại trong op

            # Nếu giá trị n không có trong từ điển
            if n not in table:
                s = table[old]
                s += c
            else:
                s = table[n]
            # Ghi giá trị s vào tệp tin
            f.write(s)

            # Cập nhật ký tự c
            c = s[0]
            # Thêm một cặp khóa-giá trị mới vào từ điển
            table[count] = table[old] + c
            count += 1
            # Cập nhật giá trị old
            old = n
```

Ý tưởng code

Khởi tạo từ điển ban đầu với các kí tự đơn trong ascii
Old= giá trị trong đầu tiên của mã output
S= kí tự tương ứng với mã output old
C= kí tự đầu tiên của chuỗi s
Mở file để ghi dữ liệu
Ghi giá trị đầu tiên vào
Thực hiện vòng lặp qua từng mã trong output code:
n = mã tiếp theo trong mảng
Nếu n không có trong từ điển:
S= mã của kí tự trước đó hay old
S=ký tự trước đó old+ ký tự đầu tiên của old
Nếu n có trong từ điển:
S= kí tự của mã n
Ghi s vào file
Cập nhật ký tự c và thêm cặp khóa mới vào từ điển
Tăng mã count
Cập nhật old = mã vào n đang xét
Lặp lại cho hết vòng lặp

2.2 Quá trình giải nén

- Ví dụ: giải nén cho <66><65><256><257><65><260>



1. **66** is in Dictionary; output **string(66)** i.e. **B**
2. **65** is in Dictionary; output **string(65)** i.e. **A**, insert BA
3. **256** is in Dictionary; output **string(256)** i.e. **BA**, insert **AB**
4. **257** is in Dictionary; output **string(257)** i.e. **AB**, insert **BAA**
5. **65** is in Dictionary; output **string(65)** i.e. **A**, insert **ABA**
6. **260** is **not** in Dictionary; output
   previous output + previous output first character: AA, insert AA

2.3 Ưu điểm và nhược điểm

Ưu điểm của LZW:

- **Hiệu suất nén tốt với chuỗi lặp lại**: LZW thường có hiệu suất nén tốt, đặc biệt là đối với các loại dữ liệu có sự lặp lại nhiều, như văn bản, tệp tin hình ảnh và âm thanh. Nó có thể nén dữ liệu thành các mã ngắn hơn, giúp giảm kích thước tệp tin.
- **Không mất dữ liệu**: LZW là một thuật toán nén không mất thông tin(lossless), có nghĩa là nén và giải nén dữ liệu không làm mất bất kỳ thông tin nào. Khi giải nén, dữ liệu ban đầu sẽ được khôi phục chính xác như ban đầu.
- **Đơn giản và dễ hiện thực**: Thuật toán LZW có cấu trúc đơn giản và dễ hiện thực. Nó chỉ cần một bảng từ điển(dictionary) để lưu trữ các chuỗi con đã xuất hiện trước đó trong quá trình nén và giải nén.

2.3 Ưu điểm và nhược điểm

Nhược điểm của LZW:

- **Tốn không gian lưu trữ**: Một nhược điểm của LZW là nó yêu cầu một bảng tra cứu lớn để lưu trữ các chuỗi con đã xuất hiện trước đó trong quá trình nén và giải nén.
- **Không tối ưu với dữ liệu ngẫu nhiên**: LZW thường hoạt động tốt trên các dữ liệu có sự lặp lại nhiều. Tuy nhiên, đối với các tệp tin có tính ngẫu nhiên cao, LZW có thể không có hiệu suất nén tốt như mong đợi và thậm chí có thể tăng kích thước tệp tin sau quá trình nén.

04

Shannon - Fano

Nội dung

Giới thiệu

Shannon và Fano là gì ?

Shannon

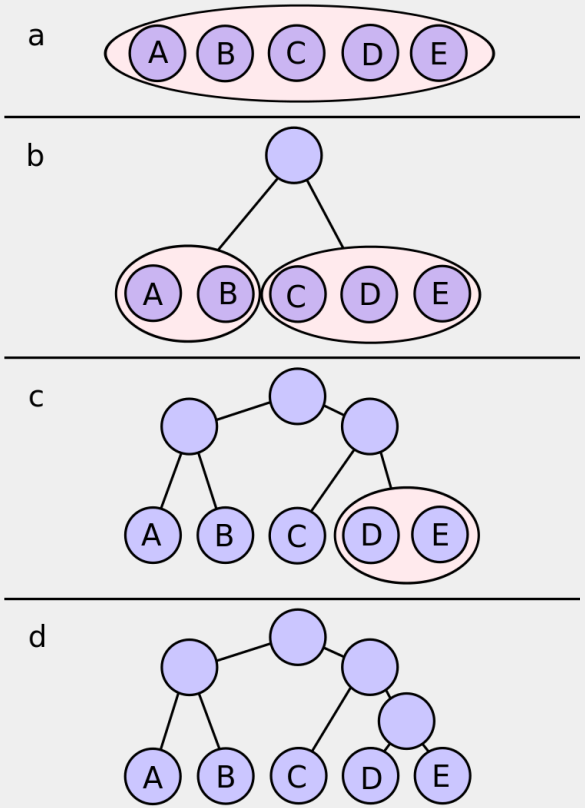
Mô tả thuật toán Shannon

Fano

Mô tả thuật toán Fano

Giới thiệu:

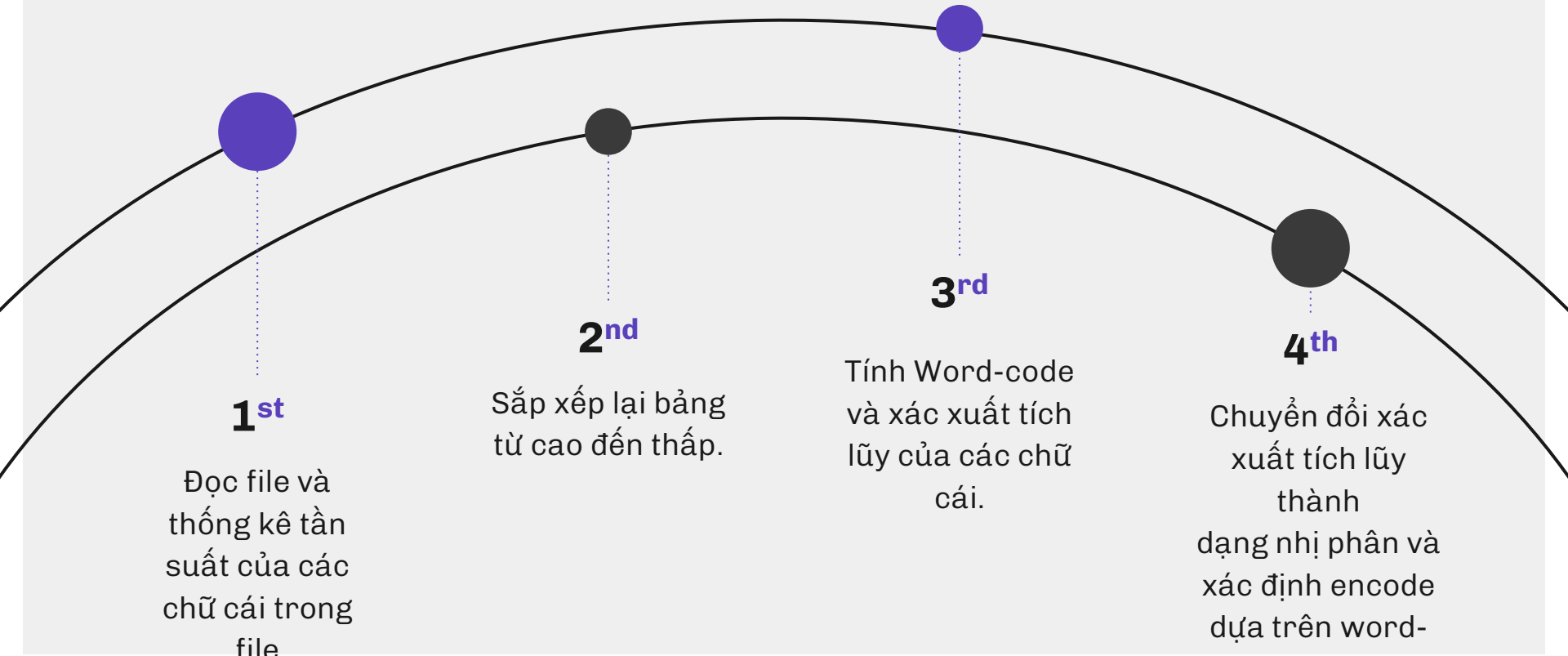
Thuật toán Shannon Fano là một kỹ thuật mã hóa entropy để nén dữ liệu đa phương tiện không mất dữ liệu. Được đặt theo tên của Claude Shannon và Robert Fano, nó gán một mã cho mỗi biểu tượng dựa trên xác suất xuất hiện của chúng. Đây là sơ đồ mã hóa có độ dài thay đổi, nghĩa là các mã được gán cho các ký hiệu sẽ có độ dài khác nhau



Shannon Algorithm

Phương pháp của Shannon chọn mã tiền tố trong đó ký hiệu nguồn được cung cấp độ dài wordcode = -log2(pi). Một cách phổ biến để chọn wordcode là sử dụng khai triển nhị phân của xác suất tích lũy.

Các bước thực hiện thuật toán:



1st

Đọc file và thống kê tần suất của các chữ cái trong file

2nd

Sắp xếp lại bảng từ cao đến thấp.

3rd

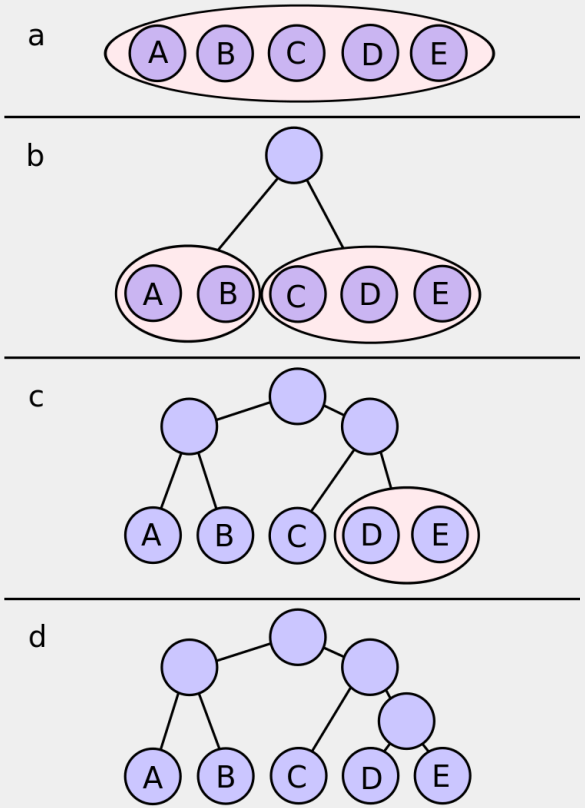
Tính Word-code và xác xuất tích lũy của các chữ cái.

4th

Chuyển đổi xác xuất tích lũy thành dạng nhị phân và xác định encode dựa trên word-code

Ví dụ:

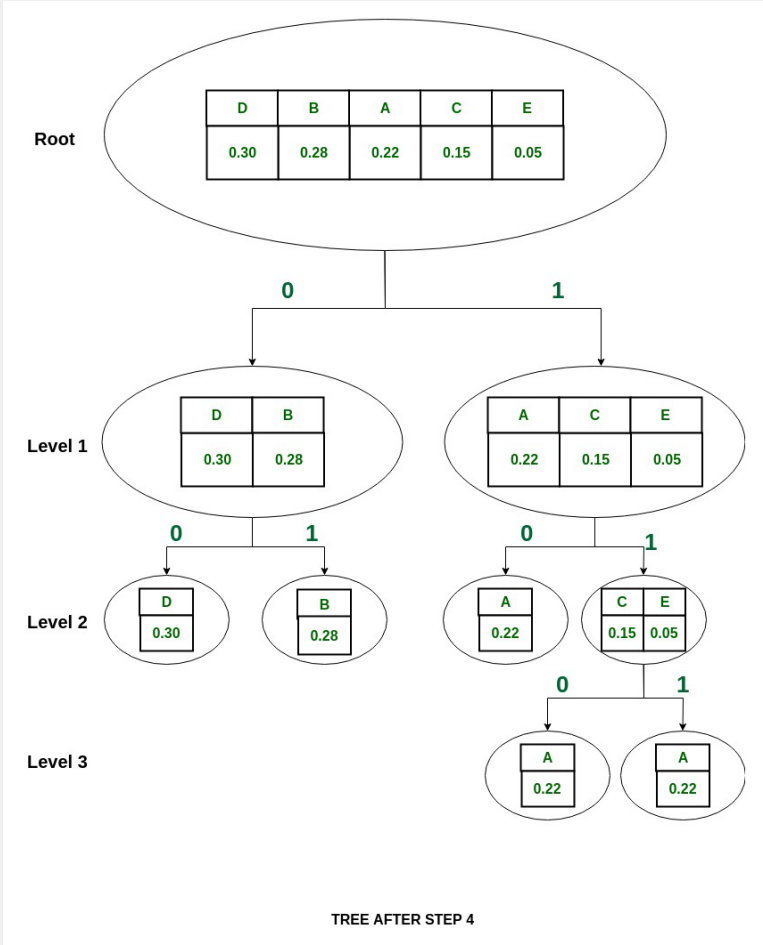
Symbol	A	B	C	D	E
Probabilities	0.385	0.179	0.154	0.154	0.128
Cumulative probabilities	0.000	0.385	0.564	0.718	0.872
...in binary	0.00000	0.01100	0.10010	0.10110	0.11011
Word lengths $\lceil -\log_2 p_i \rceil$	2	3	3	3	3
Codewords	00	011	100	101	110



Fano

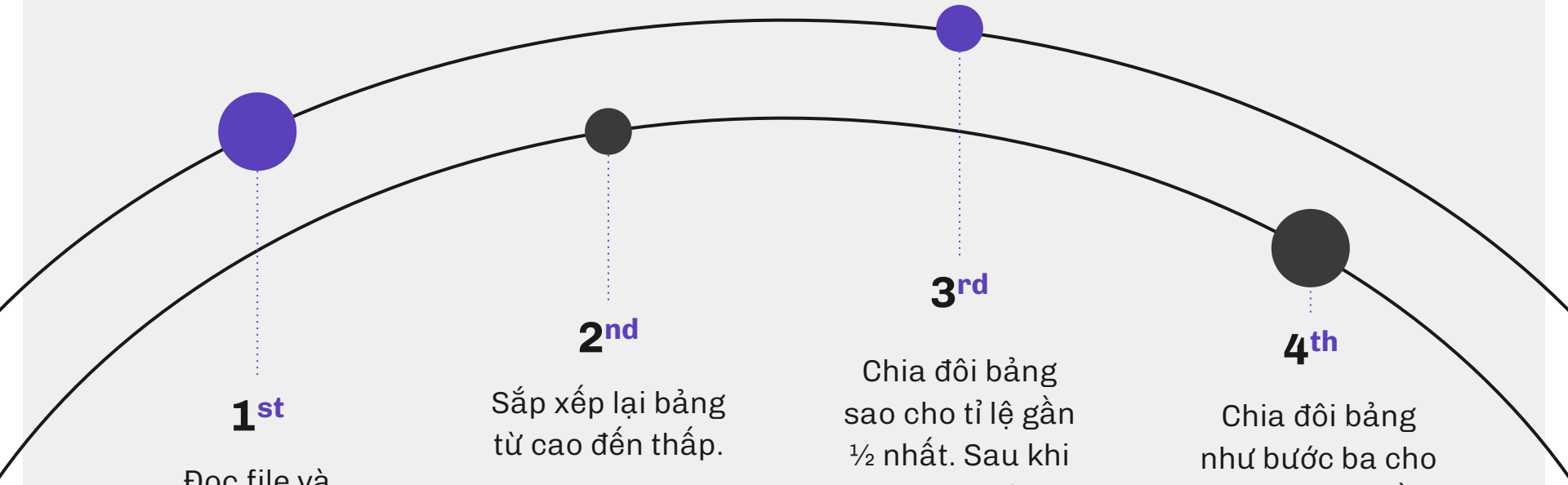
Phương pháp của Fano chia các ký hiệu nguồn thành hai bộ ("0" và "1") với xác suất càng gần 1/2 càng tốt. Sau đó, các tập hợp đó lại được chia thành hai, và cứ tiếp tục như vậy, cho đến khi mỗi tập hợp chỉ chứa một ký hiệu. Từ mã cho biểu tượng đó là chuỗi "0" và "1" ghi lại một nửa số lần chia mà nó rơi vào.

Ví dụ:



TREE AFTER STEP 4

Các bước thực hiện thuật toán:



1st

Đọc file và thống kê tần suất của các chữ cái trong file

2nd

Sắp xếp lại bảng từ cao đến thấp.

3rd

Chia đôi bảng sao cho tỉ lệ gần ½ nhất. Sau khi được 2 nhóm,

4th

Chia đôi bảng như bước ba cho tới khi các phần từ trong bảng nằm trong các nhóm phù hợp

Ưu điểm và hạn chế

Ưu điểm

- Đối với Shannon ta không cần xây dựng toàn bộ mã, mà chỉ cần lấy mã cho thẻ tương ứng với một trình tự nhất định.
- Tốc độ thực thi nhanh.
- Dễ thực hiện bằng phương pháp đệ quy.

Hạn chế

- Có thể có 2 mã khác nhau cho một kí tự tùy thuộc vào cách chúng ta xây dựng cây.
- Không đảm bảo mã tối ưu.
- Mất 2 lần duyệt file khi nén.
- Phải lưu trữ thông tin giải mã vào file nén.

05

SO SÁNH VÀ ĐÁNH GIÁ

Phương pháp đánh giá

Tỉ số nén (compression ratio) là một đánh giá về hiệu quả của việc nén dữ liệu. Nó đo lường sự giảm kích thước của dữ liệu nén so với kích thước gốc của dữ liệu.

Tỉ số nén được tính bằng cách chia kích thước dữ liệu gốc cho kích thước dữ liệu sau khi nén. Công thức tỉ số nén được biểu diễn như sau:

Tỉ số nén = Kích thước dữ liệu gốc / Kích thước dữ liệu sau nén

Phương pháp đánh giá

Hiệu suất nén (compression performance) là một khái niệm để đánh giá hiệu quả của quá trình nén dữ liệu.

Nó đo lường khả năng của thuật toán nén dữ liệu trong việc giảm kích thước của dữ liệu và tốn bao nhiêu thời gian và tài nguyên để thực hiện quá trình nén.

Ở đây ta tính dựa trên tỉ số nén

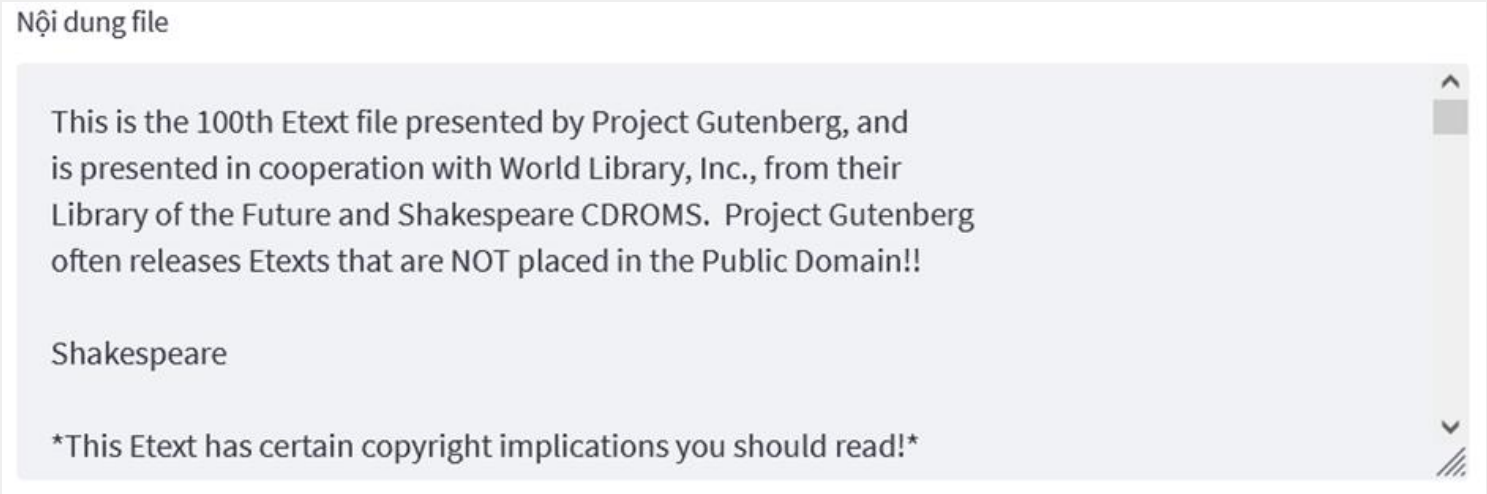
$$\text{Hiệu suất nén} = 1 - \frac{1}{\text{tỉ số nén}}$$

Dữ liệu

Test 1 là một chuỗi và lặp lại. Có kích thước là **5 kb**

Dữ liệu

Test 2 là một file văn bản chứa nội dung lớn **5.3 MB**: [*Link*]



Dữ liệu

Test 3 là một file lớn có dung lượng **6.3 MB** : [*link*]

Nội dung file

The Project Gutenberg EBook of The Adventures of Sherlock Holmes
by Sir Arthur Conan Doyle
(#15 in our series by Sir Arthur Conan Doyle)

Copyright laws are changing all over the world. Be sure to check the
copyright laws for your country before downloading or redistributing
this or any other Project Gutenberg eBook.

Kết quả trên text

File		Data ban đầu (bytes)	Data sau khi nén (bytes)	Hiệu suất nén	Tỉ số nén	Thời gian nén (s)	Thời gian giải nén
Test 1	Huffman	5.016	1674	66.63%	2.99642	0.00742	0.01785
	LZW	**5.016**	**972**	**80.62%**	**5.16049**	**0.002**	**0.00349**
	Shannon	5.016	1881	62.5%	2.66667	0.00719	0.02297
	Fanon	5.016	1672	66.67%	3	0.00471	0.02009
Test 2	**Huffman**	**5.582.653**	**3157249**	**43.45 %**	**1.7682**	**43.62**	**46.49**
	LZW	5.582.653	3302528	40.84 %	1.69042	2.48	1.65
	Shannon	5.582.653	3548557	36.44 %	1.57322	42.85	13.08
	Fanon	5.582.653	3224635	42.24 %	1.73125	29.28	12.65
Test 3	**Huffman**	**6.616.755**	**3682826**	**44.34 %**	**1.79665**	**71.82**	**60.23**
	LZW	6.616.755	3971008	39.99 %	1.66627	3.259	1.35
	Shannon	6.616.755	4119948	37.73 %	1.60603	52.06	10.34
	Fanon	6.616.755	3755034	43.25 %	1.7621	40.64	8.66

Nhận xét trên text

Với **test 1** là một file có kích thước nhỏ có nhiều chuỗi lặp lại

- LZW có hiệu suất nén cao nhất và thời gian xử lý nhanh nhất
- Những thuật toán còn lại cũng đem lại hiệu suất nén khá ổn trên 60%.
- Do file này có kích thước khá nhỏ nên thời gian giải nén và nén rất nhanh.

Với **test 2** là một file ebook có kích thước lớn (5.3MB)

- Xét về hiệu suất nén huffman>fano>lzw>shannon
- Xét về thời gian nén và giải nén lzw>fano>shannon>huffman
- Hiệu suất nén của các thuật toán đều không cao < 45%
- LZW có thời gian nén và giải nén nhanh hơn rất nhiều so với các thuật toán còn lại
-

Với **test 3** là tổng hợp các bài thơ của Shakespeare có kích thước lớn(6.3 MB)

- Ta có đánh giá tương tự như test 2

Dữ liệu ảnh

	Kích thước gốc (bytes)	Kích thước sau khi chuyển đổi thành mảng (bits)	Size
1.jpg	156,708	30,345,511	640 x 427
2.jpg	178,027	54,202,176	880 x 587
3.jpg	594,137	233,660,700	1920 x 1080

Kết quả trên ảnh

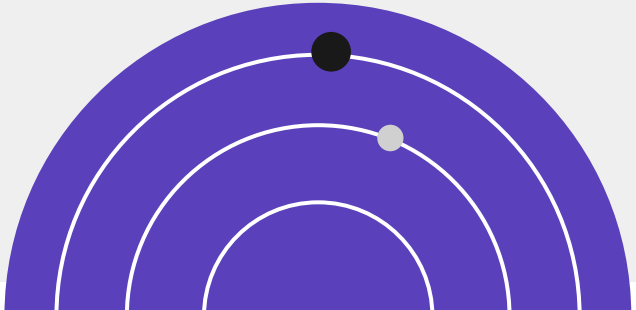
File		Data ban đầu (bit)	Data sau khi nén (bit)	Hiệu suất nén	Tỉ số nén	Thời gian nén (s)	Thời gian giải nén (s)
Test 1	H	30 345 511	15 185 294	49.96 %	1.99835	9.96352	13.33859
	LZW	**30 345 511**	**11 597 376**	**61.78 %**	**2.61658**	**2.58432**	**1.57215**
	S	30 345 511	17 060 800	43.78 %	1.77867	10.86339	7.80134
	F	30 345 511	16 451 238	45.79 %	1.84457	13.95744	44.40579
Test 2	H	54 202 176	27 097 969	50.01 %	2.00023	27.72809	35.3615
	LZW	**54 202 176**	**20 029 079**	**63.05 %**	**2.70617**	**4.84693**	**3.5622**
	S	54 202 176	30 437 678	43.84 %	1.78076	31.28466	14.37404
	F	54 202 176	29 245 750	46.04 %	1.85334	25.3036	78.29486
Test 3	H	233 660 700	117 637 336	49.65 %	1.98628	490.84862	476.38434
	LZW	**233 660 700**	**66 068 058**	**71.72 %**	**3.53667**	**21.33664**	**41.10317**
	S	233 660 700	131 803 767	43.59 %	1.77279	581.03776	100.06862
	F	233 660 700	120 495 742	48.43 %	1.93916	109.38912	364.66621

Nhận xét trên ảnh

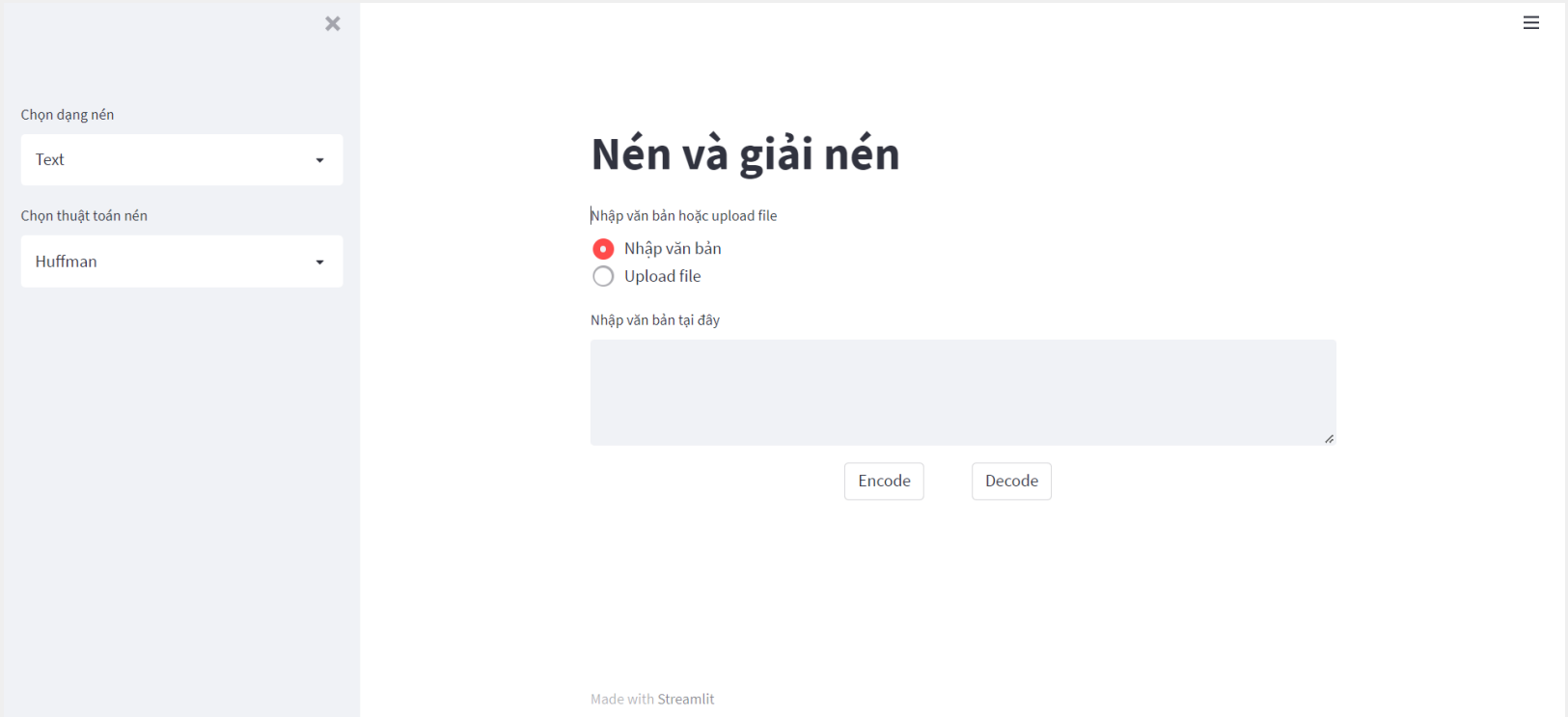
- Ta có thể nhận thấy LZW tỏ ra vượt trội so với Huffman, Shannon-Fano trên bộ dữ liệu mà nhóm đã chuẩn bị.
- LZW không chỉ có tỷ số nén cao hơn mà còn có thời gian nén lẫn thời gian giải nén ngắn hơn Huffman, Shannon-Fano.
- Ngoài ra, trong quá trình thực nghiệm, phương pháp Huffman cho kết quả tốt hơn phương pháp Shannon, Fano về hiệu suất nén.

06

DEMO



DEMO


Chọn dạng nén
Text
Chọn thuật toán nén
Huffman
Nén và giải nén
Nhập văn bản hoặc upload file
Nhập văn bản
Upload file
Nhập văn bản tại đây
Encode
Decode
Made with Streamlit

Bảng phân công công việc

MSSV	Tên	Phần công việc	Mức độ hoàn thành
20521826	Lữ Thị Thúy Quỳnh	Làm slide, báo cáo, code và demo phần LZW	100%
20521990	Bùi Văn Thuận	Làm slide, báo cáo, code và demo phần Huffman	100%
20522127	Trần Vĩnh Tuân	Làm slide, báo cáo, code và demo phần Shannon Fano	100%

Thanks

CREDITS: This presentation template was created by **Slidesgo**, and includes icons by **Flaticon**, and infographics & images by **Freepik**