ĐẠI HỌC QUỐC GIA TP. HỒ CHÍ MINH

TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN

UIT

TRƯỜNG ĐẠI HỌC
CÔNG NGHỆ THÔNG TIN

BÁO CÁO ĐỒ ÁN

ĐÁNH GIÁ CÁC THUẬT TOÁN NÉN HUFFMAN CODING, LZW VÀ RLE

Môn học: Tính toán đa phương tiện

Lớp: CS232.N21.KHCL

Giảng viên hướng dẫn: Th.S Đỗ Văn Tiến

Nhóm thực hiện: Lê Văn Khoa - 20521467

Hoàng Đình Hữu - 20521384

Nguyễn Nguyên Khôi - 21521009

TP. HỒ CHÍ MINH, 2023

MỤC LỤC

I. TỔNG QUAN VỀ ĐỀ TÀI

1.1. Giới thiệu

- Sự phát triển của công nghệ thông tin những năm gần đây dẫn đến sự bùng nổ về khối lượng dữ liệu được lưu trữ trên đĩa cứng và được gửi qua Internet. Ngày nay, với công nghệ phát triển thì việc lưu trữ dữ liệu lớn là điều có thể làm được. Tuy nhiên, với sự gia tăng nhanh chóng của dữ liệu thì các hệ thống trên không thể đáp ứng đủ nhu cầu của người dùng, cũng như việc gửi những bộ dữ liệu lớn qua Internet. Do đó nhu cầu "nén dữ liệu" được sinh ra.
- Nén dữ liệu tức là là việc chuyển định dạng thông tin sử dụng ít bit hơn cách thể hiện ở dữ liệu gốc.
- Trong đề tài này, chúng em sử dụng ba thuật toán nén là Huffman Coding, LZW, RLE để nén chuỗi không cách và so sánh kết quả mà ba thuật toán này đã làm được, từ đó xem xét ứng với mỗi chuỗi khác nhau thì thuật toán nào tối ưu hơn.

1.2. Mục tiêu và phương pháp

1.2.1. Mục tiêu

- Với mỗi trường hợp cụ thế, chúng em sẽ so sánh tỉ lệ nén và tốc độ nén của từng thuật toán và rút ra nhận xét chung về ba thuật toán trong từng trường hợp, giải thích sự hiệu quả của mỗi thuật toán về mặt nén dữ liệu và thới gian nén.

1.2.2. Phương pháp

- Tính toán số bit trước khi nén và sau khi nén của chuỗi ban đầu đưa vào, sau đó sẽ có được số bit tiết kiệm được và có được tỉ lệ nén ba thuật toán.
- Sử dụng thư viện time trong Python để tính toán thời gian nén (thời gian nén không chắc chắn đúng 100% vì tùy vào cấu hình máy tính cài đặt mà thời gian nén sẽ khác nhau, ở đây em sử dụng laptop cá nhân để thực hiện thao tác đo thời gian nên có thể con số này sẽ khác khi sử dụng môi trường thực thi khác).

II. CÁC THUẬT TOÁN NÉN

2.1. Thuật toán Run Length Encoding

2.1.1. Giới thiệu

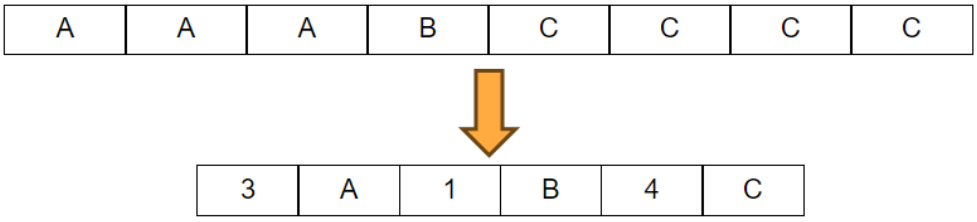
- Thuật toán nén Run-Length Encoding (RLE) là một phương pháp đơn giản để nén dữ liệu. Nó dựa trên việc lặp lại các giá trị liên tiếp trong chuỗi dữ liệu để thay thế chúng bằng một đại diện duy nhất và số lần lặp lại của chúng.

  Ví dụ: Ta có chuỗi: "AAABBBBCCCCCCD" (14 bytes)
  Sau khi nén bằng RLE: 3A4B6C1D (8 bytes)

2.1.2. Quá trình nén

Bước 1: Bắt đầu từ đầu dữ liệu, xem xét từng ký tự theo thứ tự
Bước 2: Xác định số lượng ký tự lặp lại liên tiếp, bắt đầu từ ký tự hiện tại. Đếm số lượng ký tự liên tiếp cho đến khi gặp một ký tự khác.
Bước 3: Ghi lại ký tự hiện tại và số lần lặp lại vào output.
Bước 4: Di chuyển đến ký tự tiếp theo và quay lại Bước 2 cho đến khi đi qua toàn bộ dữ liệu.

Ví dụ:



Giải thích:

- Bắt đầu từ trái sang phải ta xác định chuỗi lặp lại. Đầu tiên là ký tự A lặp lại 3 lần cho ra output là '3A'
- Qua ký tự khác là đến ký tự B lặp lại 1 lần cho ra output '1B'
- Qua tiếp ký tự C lặp lại 4 lần cho ra output '4C'. Tới đây thì đã duyệt qua toàn bộ dữ liệu, thuật toán kết thúc.
- Output: "3A1B4C"

2.1.3. Quá trình giải nén

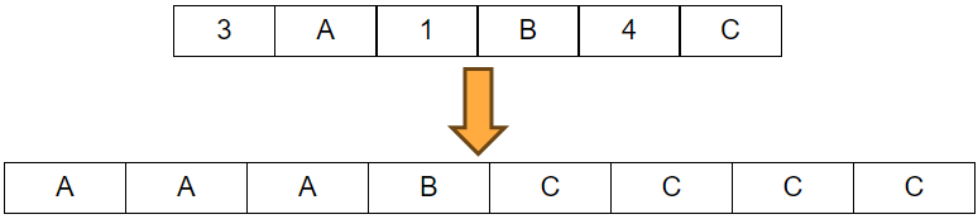
Bước 1: Bắt đầu từ đầu dữ liệu, xem xét từng ký tự theo thứ tự

Bước 2: Đọc ký tự hiện tại và xác định số lần lặp lại tương ứng.

Bước 3: Ghi lại ký tự hiện tại vào đầu đầu ra giải nén RLE, lặp lại nó theo số lần lặp lại đã xác định.

Bước 4: Di chuyển đến ký tự tiếp theo và quay lại Bước 2 cho đến khi đi qua toàn bộ dữ liệu

Ví dụ



Giải thích:

- Bắt đầu từ trái sang phải ta xác định số lần lặp của ký tự ‘A’ là 3, xuất ra output là ‘AAA’.
- Qua kí tự tiếp theo là ‘B’ với số lần lặp là 1, xuất ra output là ‘B’.
- Tiếp theo qua kí tự ‘C’ với số lần lặp là 4, xuất ra output là ‘CCCC’.
- Output: ‘AAABCCCC’.

2.1.4. Ưu điểm và nhược điểm của thuật toán RLE

Ưu điểm

- Hiệu quả đối với dữ liệu có tính chất lặp lại: RLE thường rất hiệu quả đối với dữ liệu có sự lặp lại nhiều. Với các chuỗi giá trị liên tiếp, RLE có thể giảm số lượng bit cần để biểu diễn dữ liệu một cách đáng kể.
- Tốn ít tài nguyên: RLE không đòi hỏi nhiều tài nguyên tính toán hoặc bộ nhớ để thực hiện quá trình nén và giải nén. Điều này làm cho nó hữu ích trong các hệ thống có tài nguyên hạn chế.

Nhược điểm

- Không hiệu quả đối với dữ liệu không có tính chất lặp lại: RLE không hiệu quả khi áp dụng cho dữ liệu không có sự lặp

lại. Trong trường hợp này, kích thước dữ liệu nén có thể lớn hơn hoặc không thay đổi so với dữ liệu gốc do cặp được tạo ra cho mỗi giá trị duy nhất.
Ví dụ: Ta có chuỗi: "AAABBCDE" (8 bytes)
Sau khi nén bằng RLE: 3A2B1C1D1E (10 bytes)

- Không đạt được mức nén cao: RLE không thể đạt được mức nén cao như các thuật toán nén phức tạp hơn như Huffman hay LZW. Nó chỉ đơn giản là thay thế các chuỗi giá trị lặp lại bằng một cặp giá trị, mà không tận dụng được các mẫu xuất hiện trong dữ liệu để tạo ra một mã hóa ngắn hơn.

2.2. Thuật toán LZW (Lempel-Ziv -Welch)

2.2.1. Giới thiệu

- **Lempel–Ziv–Welch** hay 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.
- LZW là thuật toán nén dữ liệu không mất thông tin, được sử dụng phổ biến trong việc giảm kích thước của các tệp tin văn bản. Thuật toán thường được sử dụng trong GIF và tùy chọn trong PDF và TFF. Ý tưởng chính của thuật toán này là mã hóa và giải mã chuỗi ký tự thông qua việc xây dựng từ điển.

2.2.2. Quá trình nén

- LZW hoạt động đựa trên một ý tưởng rất đơn giản là người mã hoá và người giải mã cùng xây dựng bảng mã hay từ điển. Từ điển hay bảng mã này không cần được lưu kèm với dữ liệu trong quá trình nén mà khi giải nén người giải nén sẽ xây dựng lại nó.
- Ý tưởng nén của thuật toán LZW:
    - **Bước 1:** Khởi tạo từ điển ban đầu chứa các ký tự đơn (thường sẽ dùng bộ mã ASCII với 256 ký tự).
    - **Bước 2**: Đọc vào một chuỗi ký tự và tiến hành nén chuỗi bằng cách tìm chuỗi con trong từ điển trùng với chuỗi được đọc vào.

        - Nếu chuỗi con đó không có trong từ điển thì thêm chuỗi con đó vào từ điển và đưa ra mã hóa của chuỗi ký tự trước đó.
        - 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.
    - **Bước 3**: Lưu lại thông tin về mã hóa vào tệp tin đích.
- Ví dụ: Mã hóa chuỗi kí tự 'XYWXYZ'

Trong ví dụ này, ta sẽ tạo từ điển và có sẵn mã của 256 kí tự theo bảng mã ASCII. như ý tưởng thuật toán là ta sẽ thêm những từ chưa có trong từ điển và tạo cho nó một mã hóa mới bắt đầu từ 256, và nếu ký tự xuất hiện trong từ điển thì ta trả về mã hóa của nó.

STT	Kết quả trả về		Từ điển	
	Mã	Kí tự	Mã hóa	Chuỗi
1	88	X	256	XY
2	89	Y	257	YW
3	87	W	258	WX
4	256	XY	259	XYZ
5	90	Z		

Giải thích:

1. X có trong từ điển, ta trả về mã hóa của nó là 88. Kết hợp X và ký tự tiếp theo, ta có XY chưa có trong từ điển, thêm vào từ điển và mã hóa của nó là 256.
2. Y có trong từ điển, ta trả về mã hóa của nó là 89. Kết hợp Y với ký tự tiếp theo, ta có YW chưa có trong từ điển, thêm vào từ điển và mã hóa của nó là 257.

3. W có trong từ điển, ta trả về mã hóa của nó là 87. Kết hợp W với ký tự tiếp theo, ta có WX chưa có trong từ điển, thêm vào từ điển và mã hóa của nó là 258.
4. XY có trong từ điển, ta trả về mã hóa của nó là 256. Kết hợp XY với ký tự tiếp theo, ta có XYZ chưa có trong từ điển, thêm vào từ điển và mã hóa của nó là 259.
5. Z có trong từ điển, ta trả về mã hóa của nó là 90. Z là ký tự cuối cùng nên ta kết thúc quá trình nén.

→ Kết quả sau khi nén chuỗi ký tự 'XYWXYZ' là [88,89,87,256,90].

2.2.3. Quá trình giải nén

- Cũng giống với quá trình nén, ta phải xây dựng từ điển và tiến hành tra từ điển để trả về ký tự với mã hóa tương ứng.
- Ý tưởng quá trình giải nén:
  - **Bước 1**: Khởi tạo từ điển chứa các ký tự đơn (ASCII).
  - **Bước 2**: Đọc một mã từ tệp tin nén và tiến hành tìm kiếm mã trong từ điển. Gồm 2 bước nhỏ:
    - Nếu mã không có trong từ điển thì thêm mã vào từ điển và đưa ra ký tự với mã trước đó.
    - Nếu mã đã có trong từ điển thì đưa ra ký tự tương ứng với mã hiện tại.
  - **Bước 3**: Lưu lại chuỗi ký tự tương ứng với mã vừa giải được.

- Ví dụ: Giải nén mã [67,68,256,69]

Giống với quá trình nén, quá trình giải nén cũng phải xây dựng từ điển và trong ví dụ này đã lấy mã hóa theo bảng mã ASCII nên từ điển ban đầu sẽ có 256 mã bắt đầu từ 0 đến 255. Mã mới hay kí tự mới sẽ được thêm vào và bắt đầu từ 256.

STT	Kết quả trả về		Từ điển	
	Mã	Kí tự	Mã hóa	Chuỗi
1	67	C	256	CD
2	68	D	257	DC
3	256	CD	258	CDE
4	69	E		

Giải thích:

1. Mã 67 có trong từ điển, ta trả về ký tự của mã đó là C, kết hợp mã hiện tại với mã tiếp theo, ta có CD chưa có trong từ điển, thêm vào từ điển và mã hóa của nó là 256.
2. Mã 68 có trong từ điển, ta trả về ký tự của mã đó là D, kết hợp mã hiện tại với mã tiếp theo, ta có DC chưa có trong từ điển, thêm vào từ điển và mã hóa của nó là 257.
3. Mã 256 có trong từ điển, ta trả về ký tự của mã đó là CD, kết hợp mã hiện tại với mã tiếp theo, ta có CDE chưa có trong từ điển, thêm vào từ điển và mã hóa của nó là 258.
4. Mã 69 có trong từ điển, ta trả về ký tự của mã đó là E. Mã 69 là mã cuối cùng nên ta kết thúc quá trình giải nén.

→ Kết quả sau khi giải nén mã [67,68,256,69] là CDCDE.

2.2.4. Ưu điểm và nhược điểm của thuật toán LZW

Ưu điểm

- Vì là thuật toán nén không mất dữ liệu nên LZW hoạt động khá tốt trên tệp dữ liệu chứa nhiều dữ liệu lặp lại thường gặp với các hình ảnh đơn sắc.
- Không yêu cầu dữ liệu đầu vào, việc nén và giải nén đều phải xây dựng từ điển hoặc bảng mã.
- Thuật toán LZW khá đơn giản và dễ hiểu nên dễ cài đặt nhanh.
- Tỷ lệ nén cao: LZW có thể đạt được tỷ lệ nén cao, đặc biệt là đối với dữ liệu chứa văn bản. Điều này có thể làm giảm kích thước tệp và yêu cầu lưu trữ.
- Giải nén nhanh: Thuật toán LZW có quá trình giải nén nhanh hơn so với một vài thuật toán khác nên nó trở thành một lựa chọn tốt cho các ứng dụng mà tốc độc giải nén là quan trọng.
- Nén động: LZW là thuật toán nén động, điều này nghĩa là LZW thích ứng với bộ dữ liệu được nén và tỷ lệ nén cao nếu bộ dữ liệu có nhiều dữ liệu lặp lại.

Nhược điểm

- Các tệp nén không có thông tin lặp lại có thể lớn, không phù hợp với mục đích nén.
- Không gian lưu trữ: LZW yêu cầu dung lượng bộ nhớ đáng kể để có thể duy trì từ điển nén cũng như từ điển giải nén. Đây có thể là hạn chế đối với các ứng dụng có tài nguyên bộ nhớ bị giới hạn.
- Tốc độ nén: thuật toán LZW có thể nén chậm hơn so với một vài kỹ thuật nén khác, đặc biệt là bộ dữ liệu lớn do cần phải cập nhật lại từ điển liên tục.
- Khả năng áp dụng hạn chế: LZW nén dữ liệu hiệu quả đối với các tệp dữ liệu là dạng văn bản nhưng có thể không hiệu quả đối với một vài loại dữ liệu khác như hình ảnh, video hoặc có yêu cầu nén khác.
- Về mặt cấp phép và bản quyền: LZW có nhiều vấn đề về mặt sáng chế, thuật toán này đã được cấp bằng vào năm 1980 và để sử dụng cần phải trả phí để được cấp phép. Điều này đã làm hạn chế việc áp dụng thuật toán vào trong một số ứng dụng.

2.3. Thuật toán Huffman Coding

2.3.1. Giới thiệu

- Huffman Coding là một thuật toán do David A. Huffman của MIT đưa ra vào năm 1952 để nén dữ liệu dạng văn bản, mã hóa các bytes trong tệp dữ liệu nguồn bằng biến nhị phân. Nó tạo mã độ dài biến thiên là một tập hợp các bits. Mặc dù là một thuật toán nén tương đối đơn giản, nhưng Huffman đủ mạnh để các biến thể của nó vẫn được sử dụng ngày nay trong mạng máy tính, máy fax, modem, HDTV và các lĩnh vực khác.
- Nguyên lý của phương pháp Huffman là mã hóa các bytes trong tệp dữ liệu nguồn bằng biến nhị phân. Nó tạo mã độ dài biến thiên là một tập hợp các bits. Đây là phương pháp nén kiểu thống kê, những ký tự xuất hiện nhiều hơn sẽ có mã ngắn hơn.
- Thông thường, dữ liệu văn bản được lưu trữ ở định dạng chuẩn 8 bit cho mỗi ký tự, sử dụng mã hóa ASCII ánh xạ mỗi ký tự thành một giá trị số nguyên nhị phân từ 0-255. Ý tưởng của mã hóa Huffman là từ bỏ yêu cầu cứng nhắc 8 bit cho mỗi ký tự và thay vào đó sử dụng mã hóa nhị phân có độ dài khác nhau cho

các ký tự khác nhau. Ưu điểm của việc này là nếu một ký tự xuất hiện thường xuyên trong tệp, chẳng hạn như chữ cái 'e' rất phổ biến, thì nó có thể được mã hóa ngắn hơn (tức là ít bit hơn), làm cho tệp tổng thể nhỏ hơn. Sự cân bằng là một số ký tự có thể cần phải sử dụng các mã hóa dài hơn 8 bit, nhưng điều này được dành riêng cho các ký tự xảy ra không thường xuyên, vì vậy, số dư phải trả thêm.

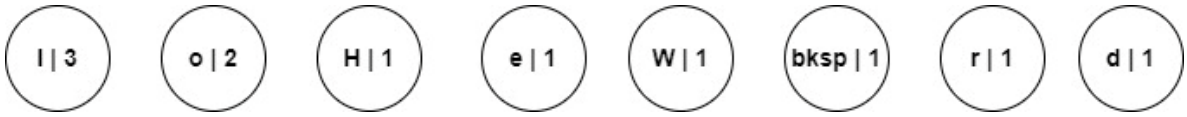
- Bảng dưới đây so sánh các giá trị ASCII của các ký tự khác nhau với các bảng mã Huffman có thể có cho một số văn bản tiếng Anh. Các ký tự thông thường như dấu cách và 'e' có mã hóa ngắn, trong khi các ký tự hiếm hơn (như 'z') có mã hóa dài hơn.

2.3.2. Quá trình nén

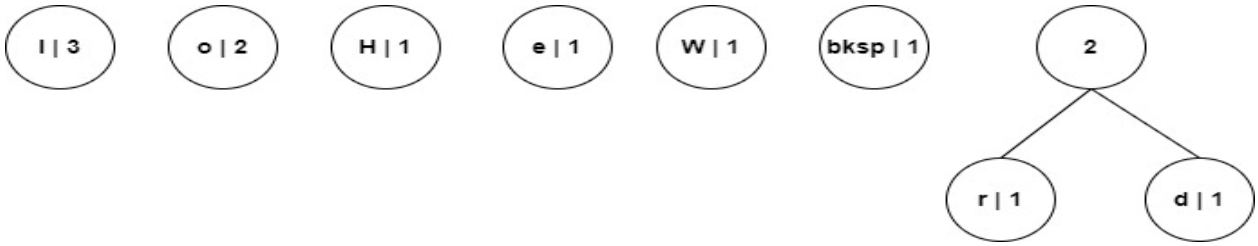
- Với ý tưởng trên, thuật toán nén Huffman coding gồm 3 bước:
  - Bước 1: Đếm tần suất xuất hiện của các phần tử trong chuỗi đầu vào.
  - Bước 2: Xây dựng cây Huffman (cây nhị phân mã hóa) với quy ước bên trái mã 0, bên phải mã 1.
  - Bước 3: Từ cây Huffman, ta có được các giá trị mã hóa. Lúc này, ta có thể xây dựng chuỗi mã hóa từ các giá trị này.
- Ở Bước 2, quá trình xây dựng cây Huffman gồm các bước sau:
  - 2.1. Tạo danh sách chứa các nút lá bao gồm phần tử đầu vào và trọng số nút là tần suất xuất hiện của nó.
  - 2.2. 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 với trọng số là tổng của 2 trọng số ở nút vừa lấy ra để tạo thành một cây.
  - 2.3. Đẩy cây mới vào lại danh sách.
  - 2.4. Lặp lại bước 2 và 3 cho đến khi danh sách chỉ còn 1 nút gốc duy nhất của cây. Đây chính là nút gốc của cây Huffman.

VD: chuỗi "Hello World"

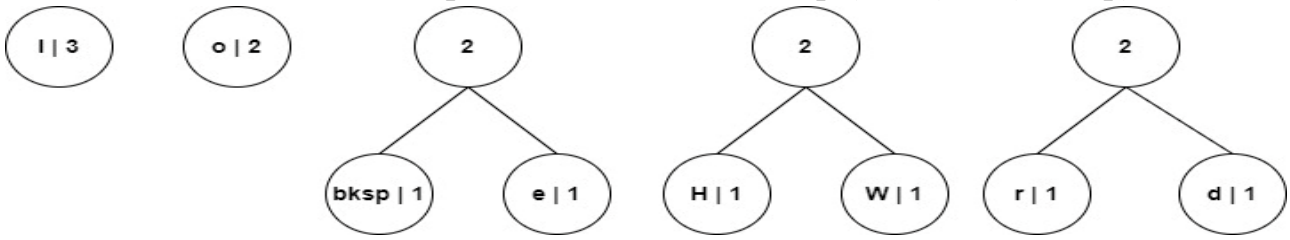
- Bước 1 và 2.1: Đếm tần suất xuất hiện các phần tử đầu vào và tạo danh sách các nút lá với trọng số là tần suất xuất hiện.



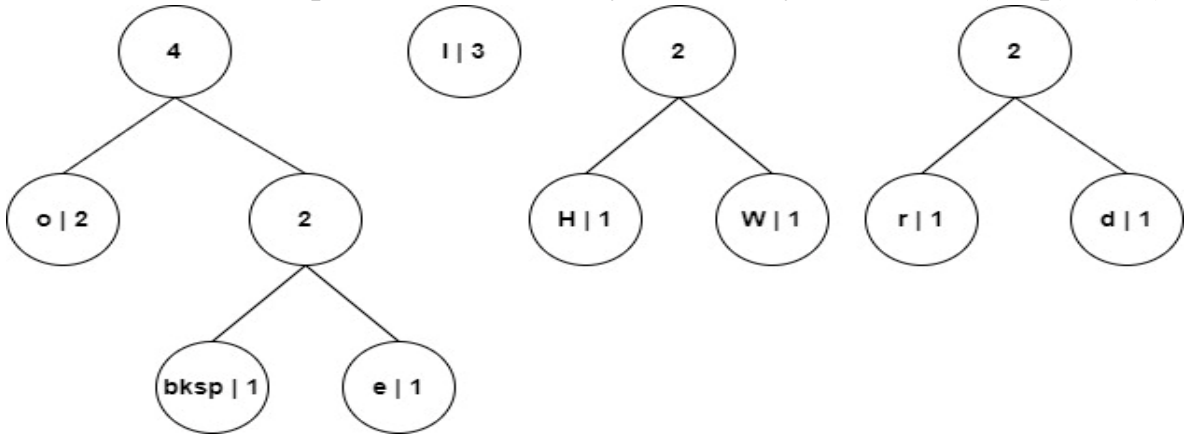
- Bước 2.2: Chọn 2 nút có trọng số thấp nhất, tạo nút gốc mới có trọng số bằng tổng 2 trọng số nút con. Sau đó gắn 2 nút con vào nút gốc và đẩy lại vào danh sách. Danh sách cần được biểu diễn đặc biệt để có thể lấy ra các nút trọng số nhỏ nhất một cách tối ưu nhất.



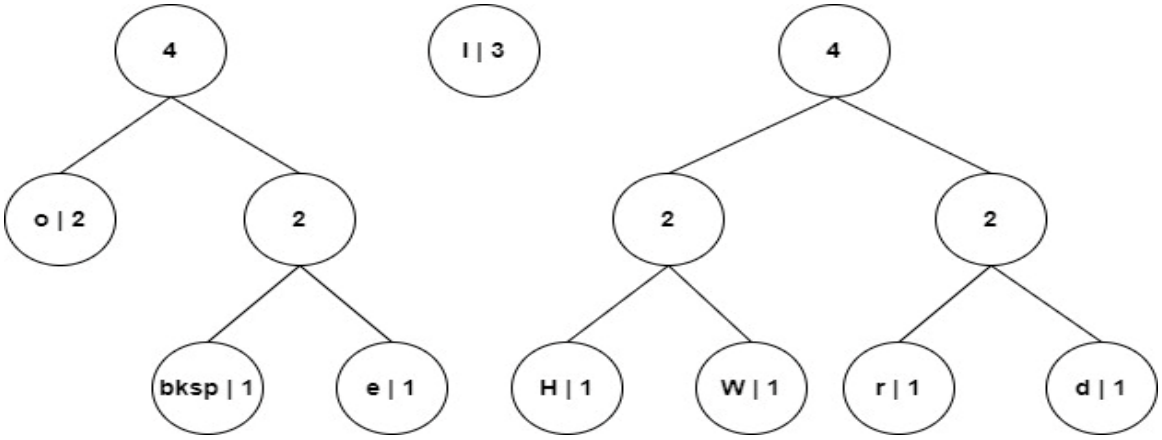
- Bước 2.3 và 2.4: Lặp lại bước 2.2 cho các cặp (H, W) và (e, bksp)



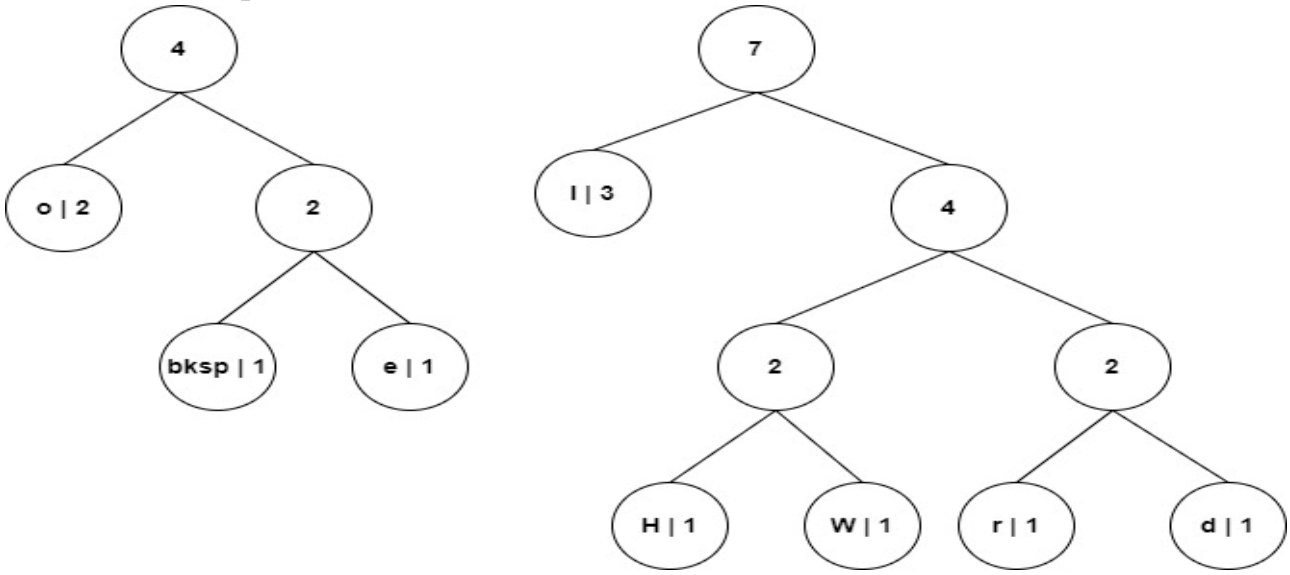
- Bước 2.5: Lặp lại bước 2.2 cho ký tự o và cây có 2 nút lá (bksp) và (e)



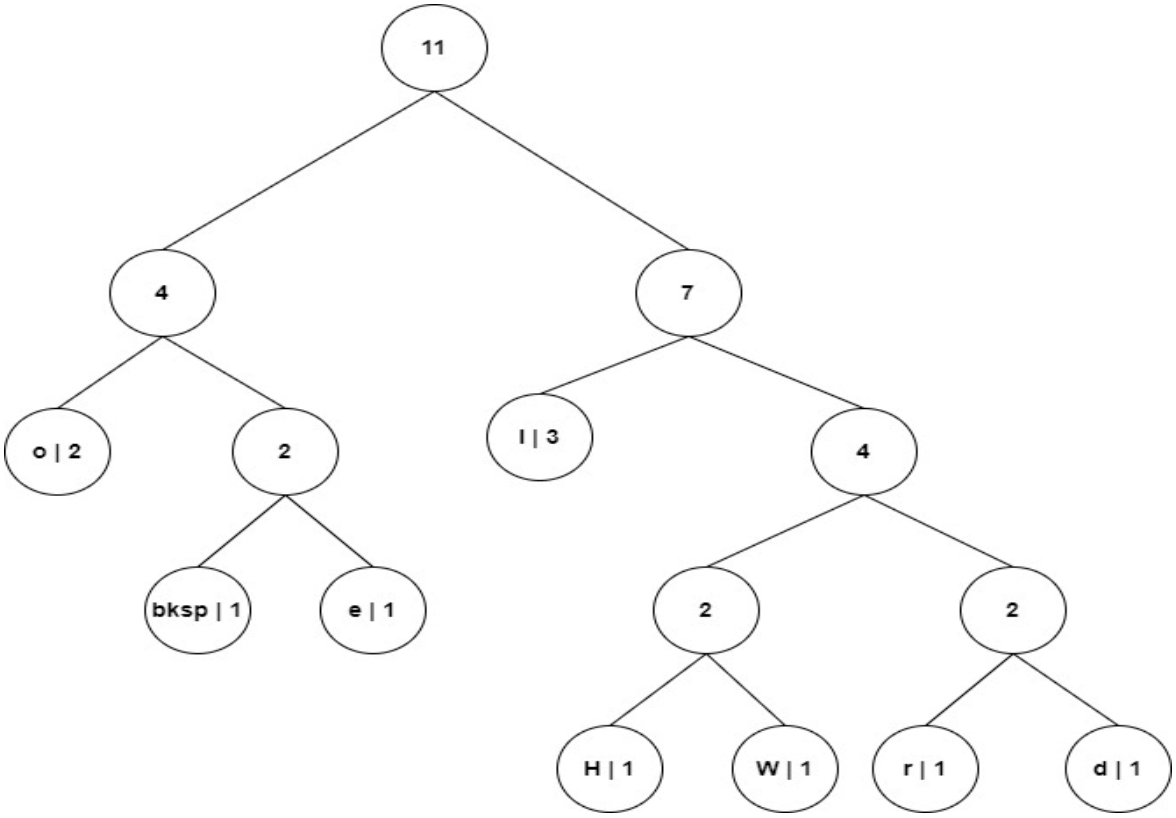
- Bước 2.6: Lặp lại bước 2.2 cho 2 cây còn lại có nút gốc mang trọng số 2



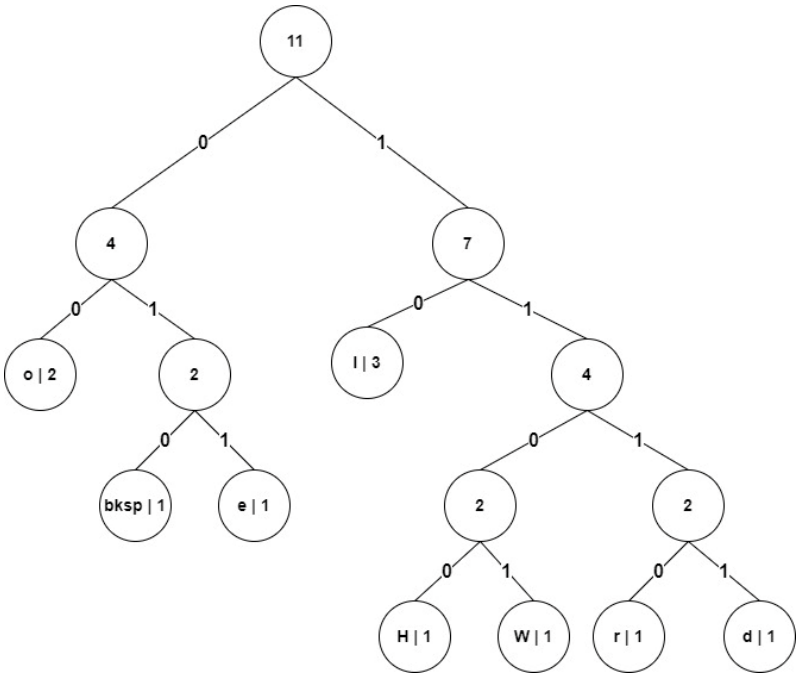
- Bước 2.7: Lặp lại bước 2.2 cho nút "l" và cây có nút gốc mang trọng số 4 ở bên phải



- Bước 2.8: Lặp lại bước 2.2 cho 2 cây còn lại trong danh sách



- Bước 2.9: Trong danh sách hiện tại chỉ còn một nút (trọng số 11) đó chính là nút gốc của cây Huffman
- Bước 3:



Ký tự	Chuỗi mã hóa
l	10
o	00
e	010
[bksp]	011
H	1100
W	1101
r	1110
d	1111

Chuỗi "**Hello World**" được mã hóa thành: 11000101010000111101001110101111

2.3.3. Quá trình giải nén

- Khởi tạo một chuỗi S rỗng. Ta đọc từng bit trong dãy mã hóa, đồng thời duyệt cây Huffman, bắt đầu từ nút gốc và dừng lại khi gặp một nút lá bất kỳ. Đẩy ký tự ứng với nút lá đó vào một chuỗi S và lặp lại quá trình trên cho các bit tiếp theo.

- Nút gốc, đọc 1 ➔qua nhánh phải ➔ nút 7, đọc 1➔ qua nhánh phải ➔ nút 4, đọc 0 ➔ qua nhánh trái ➔ nút 2, đọc 0 ➔ qua nhánh trái ➔ nút H, lá ➔ dừng ➔ chuỗi S= "H"

  Đánh dấu: ~~1100~~01110100001011010011101011111

- Nút gốc, đọc 0 ➔ qua nhánh trái ➔ nút 4, đọc 1 ➔ qua nhánh phải ➔ nút 2, đọc 1 ➔ qua nhánh phải ➔ nút e, lá ➔ dừng ➔ chuỗi S= "He"

  Đánh dấu: ~~1100011~~01010000111101001110101111

- Nút gốc, đọc 1 ➔ sang phải ➔ nút 7, đọc 0 ➔ sang trái ➔ nút l, lá ➔ dừng ➔ chuỗi S= "Hel"

  Đánh dấu: ~~11000111~~01000011110100111010111

- Nút gốc, đọc 1 ➔ sang phải ➔ nút 7, đọc 0 ➔ sang trái ➔ nút l, lá ➔ dừng ➔ chuỗi S= "Hell"

  Đánh dấu: ~~1100011101~~0000111101001110101111

- Tiếp tục quá trình trên cho đến bit cuối cùng trong dãy mã hóa. Kết quả cuối cùng của chuỗi S là S= “Hello World”

2.3.4. Ưu và nhược điểm của thuật toán Huffman

Ưu điểm:

- Nguyên lý hoạt động đơn giản, dễ hiểu
- Các ký tự khác nhau có thể được biểu diễn bằng các mã có độ dài khác nhau. Tính linh hoạt này cho phép mã hóa hiệu quả các ký hiệu có tần số khác nhau, tối ưu hóa việc nén.

Nhược điểm:

- Thuật toán Huffman chỉ thực hiện được khi biết được tần suất xuất hiện của các ký tự.
- Thuật toán Huffman đòi hỏi phải xây dựng cây nhị phân sẵn chứa các khả năng. Điều này đòi hỏi thời gian không ít do ta không biết trước kiểu dữ liệu sẽ được thực hiện nén.
- Quá trình giải nén phức tạp do chiều dài mã không được biết trước cho đến khi ký tự đầu tiên được tìm ra.

III. SO SÁNH TỈ LỆ NÉN VÀ TỐC ĐỘ NÉN CỦA 3 THUẬT TOÁN

3.1. Mô tả bộ test

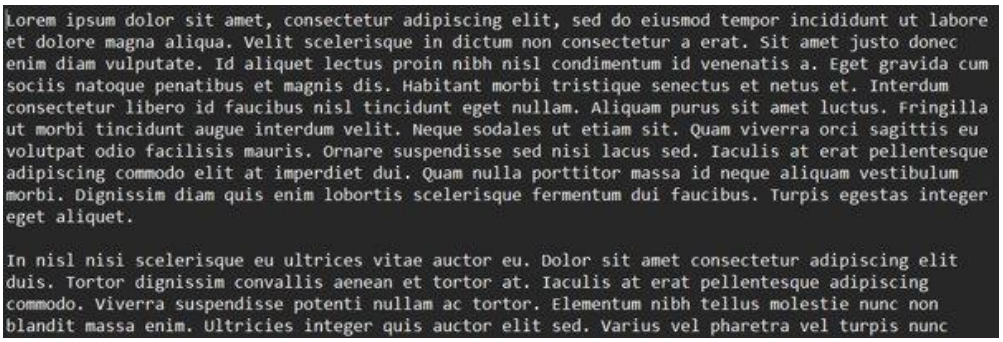
- Gồm hai bộ test

  + Test 1: Là một file chứa các chuỗi lặp lại

  Kích thước gốc: 15KB (~15.355 bytes)

  +Test 2: Là file text bao gồm nhiều kí tự khác nhau không lặp lại

  Kích thước gốc: 2.06MB (~2.167.737 bytes)



3.2. Phương pháp

- Tỉ lệ nén

$$Tỉ\ lệ\ nén = \frac{Kích\ thước\ sau\ khi\ nén}{Kích\ thước\ ban\ đầu} * 100$$

- Xác định thời gian nén và giải nén bằng cách dùng thư viện time.

```
start_time = time.time()
    // Thuật toán
end_time = time.time()
execution_time = end_time - start_time
```

- Sau khi có hai thông số tỉ lệ nén cũng như thời gian nén và giải nén ta sẽ tiến hành so sánh ba thuật toán dựa trên sự chênh lệch của chúng và đưa ra nhận xét.

3.3. Kết quả

- Đối với Test 1

Đánh giá	RLE	Huffman	LZW
Kích thước sau khi nén (bytes)	3705	7958.125	176
Tỉ lệ nén	0.24	0.5183	0.01146
Thời gian nén (s)	0.003	0.009	0.0
Thời gian giải nén (s)	0.0	0.029	0.0

- Đối với Test 2

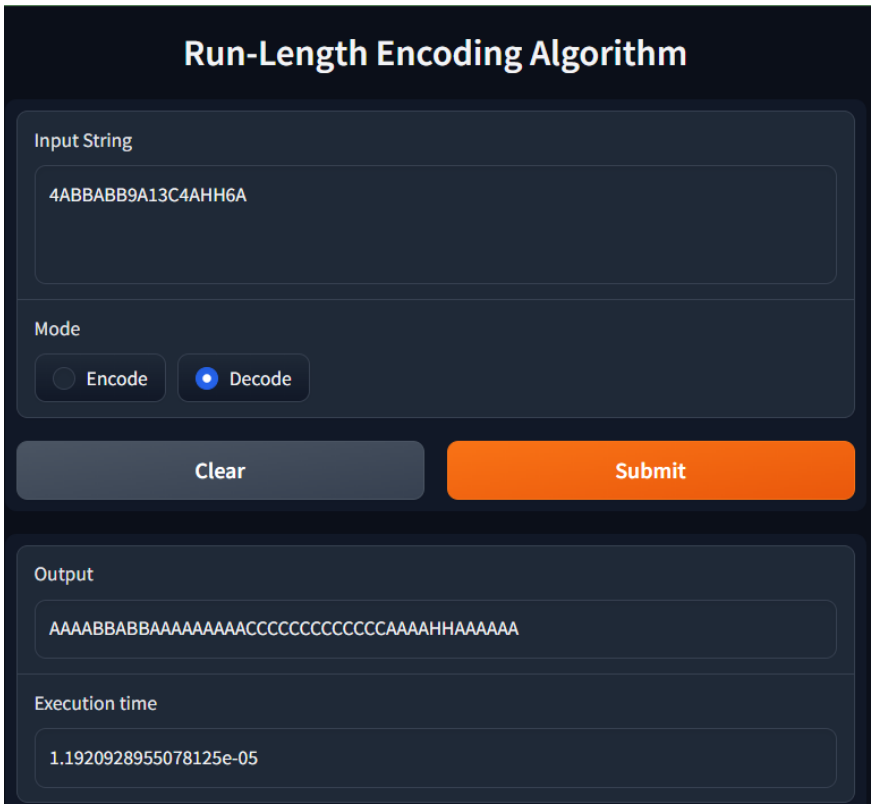
Đánh giá	RLE	Huffman	LZW
Kích thước sau khi nén (bytes)	2.173.443	1.145.519	1.119.173
Tỉ lệ nén	1.002	0.5284	0.5163
Thời gian nén (s)	1.053	1.066	1311.03
Thời gian giải nén (s)	0.72	3.771	0.35

3.4. DEMO

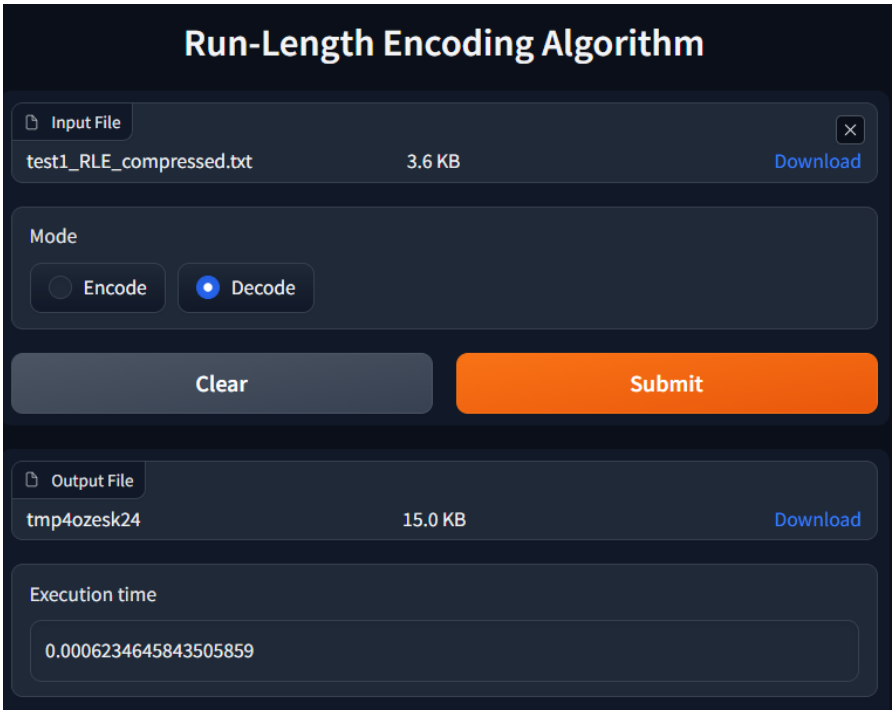
- Nhóm em sử dụng thư viện Gradio là một thư viện Python cho phép tạo các giao diện có thể tùy chỉnh cho các mô hình thuật toán. Gradio cung cấp một cách đơn giản để tạo và chia sẻ các thành phần giao diện cho người dùng tương tác.
- Bởi vì các demo này chỉ sử trên local nên phải chạy code '*Compression Algorithm.ipynb'* chúng em đã đính kèm để chạy được demo.
- Mỗi thuật toán sẽ demo 2 phần: Nhập chuỗi hoặc nhập file để Encode hoặc Decode. Sau khi thực thi thì sẽ cho ra chuỗi đầu ra và thời gian chạy tương ứng của mỗi thuật toán

- Ví dụ về giao diện cơ bản Encode và Decode của thuật toán RLE

Nhập vào một chuỗi để Encode hoặc Decode.

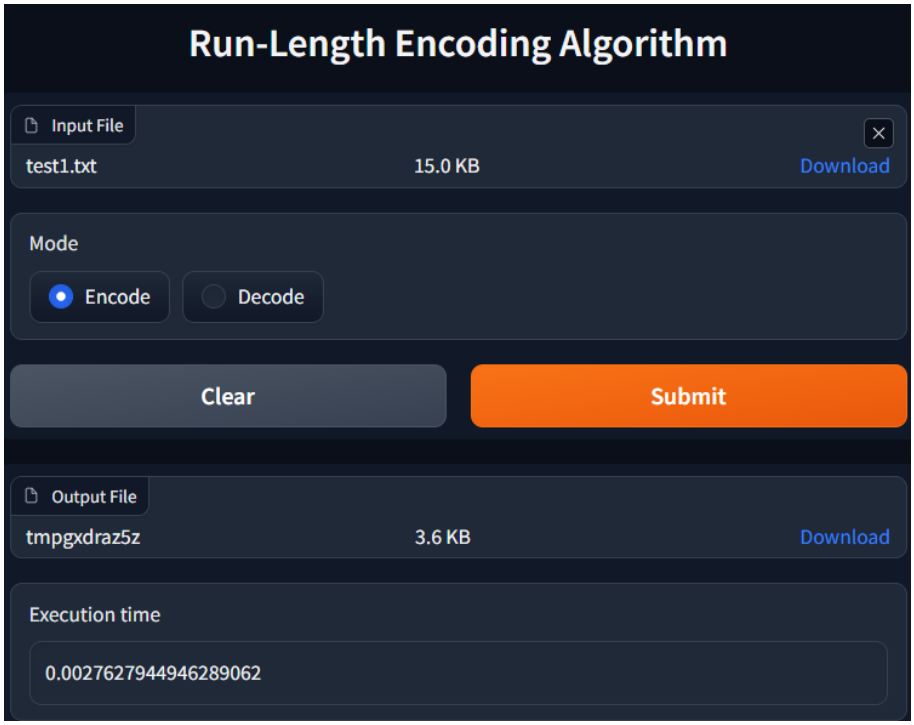


Nhập vào một chuỗi để Encode

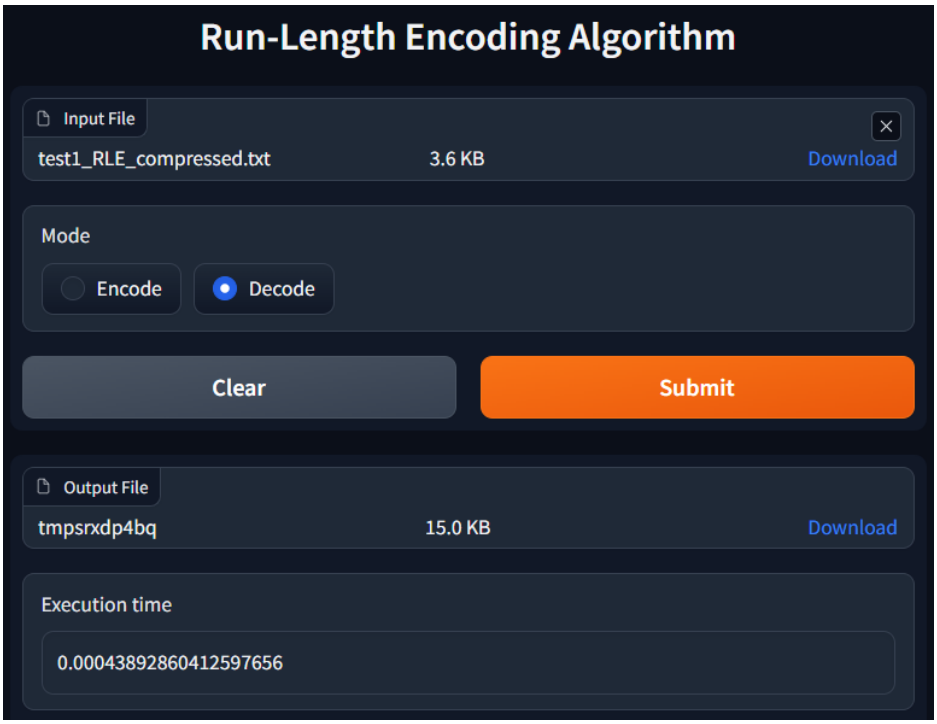


Nhập vào một chuỗi để Decode

Nhập vào một file có sẵn để Encode/ Decode



Nhập vào một file để Encode



Nhập vào một file để Decode

3.5. Nhận xét

- Sau khi thử nghiệm trên 2 tập test thì ta rút ra được một số nhận xét chung như sau:
    - Ở Test 1 thuật toán LZW nén khá tốt (15,355 bytes còn 176 bytes) cùng với thời gian nén và giải nén. Còn kém nhất là thuật toán Huffman Coding.
    - Ở Test 2 thì cả hai thuật toán Huffman Coding và LZW làm tốt việc nén dữ liệu. Tuy nhiên ở LZW thì thời gian nén khá cao (~21 phút) có lẽ là vì việc lập và tra từ điển làm cho thời gian nén file này quá cao. Còn thuật toán RLE thể hiện kém trong việc nén file Test 2, kích thước sau khi nén còn lớn hơn kích trước gốc.
    - Đối với file chứa các kí tự khác nhau, không lặp lại nhiều thì thuật toán nén RLE không thật sự hiệu quả, tạo ra dữ liệu nén có dung lượng lớn hơn dữ liệu gốc. Nó không thể nén các loại dữ liệu phức tạp.
    - Thuật toán LZW việc xây dựng và duy trì từ điển có thể tốn kém về mặt tài nguyên, đặc biệt là đối với các dữ liệu lớn. Đồng thời, quá trình decode yêu cầu sử dụng từ điển giống nhau gây nên độ phức tạp và tốn thời gian.
    - Thuật toán nén Huffman Coding cũng nén khá tốt cho các tập tin có tần suất xuất hiện của ký tự không đồng đều. Tuy nhiên Huffman không thế nén tốt cho các tập tin có số lượng ký tự quá lớn. Ngoài ra phải lưu trữ bảng tần suất hoặc cây Huffman cùng với dữ liệu gốc để decode, làm tăng kích thước lưu trữ hoặc truyền tải

IV. KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

4.1. Kết luận

Cả ba thuật toán đều hoạt động nhanh và hiệu quả, nhưng vẫn có sự chênh lệch về tỉ lệ nén và thời gian nén khi thực thi trên những bộ test khác nhau. Và hai test là chưa đủ để thấy rõ sự chênh lệch và ưu nhược điểm của từng thuật toán.

4.2. Hướng phát triển

Chúng em sẽ tìm thêm nhiều thuật toán nén khác và làm đa dạng thêm bộ test để có thể đưa ra nhận xét chính xác hơn cho từng thuật toán cũng như hiểu thêm về chúng.

V. PHÂN CHIA CÔNG VIỆC

	Phân công công việc	Huffman Coding	LZW	RLE	Làm slide	Viết báo cáo
Lê Văn Khoa	x			x	x	x
Hoàng Đình Hữu			x		x	x
Nguyễn Nguyên Khôi		x			x	x

VI. TÀI LIỆU THAM KHẢO

- LZW Compression Technique, GeeksforGeeks
- Lempel–Ziv–Welch, Wikipedia
- LZW Compression, TechTarget
- Run-Length Encoding, GeeksforGeeks
- Gradio interface, documentation
- Huffman Coding, Programiz
- Huffman Decoding, GeeksforGeeks