TRƯỜNG ĐẠI HỌC GIAO THÔNG VẬN TẢI PHÂN

HIỆU TẠI TP. HỒ CHÍ MINH

KHOA ĐIỆN – ĐIỆN TỬ

BỘ MÔN KỸ THUẬT ĐIỆN TỬ

---------- & ----------

BỘ MÔN KỸ THUẬT ĐIỆN TỬ -VIỄN THÔNG

BÁO CÁO BÀI TẬP LỚN

TÊN ĐỀ TÀI : DICTIONARY CODING (LEMPEL-ZIV-WELCH)

Sinh viên thực hiện: Ngô Bỉnh Hoàng

Lớp : Kỹ thuật Điện Tử - Viễn Thông

Khoá : 63

Giảng viên hướng dẫn : Phan Tròn

MỤC LỤC

LỜI CẢM ƠN

Lời đầu tiên, chúng em xin chân thành bày tỏ lòng biết ơn sâu sắc đến tất cả các thầy cô của trường Đại Học Giao Thông Vận Tải Phân Hiệu TP. HCM nói chung, của khoa Điện - Điện Tử và các thầy cô của bộ môn hướng dẫn nói riêng. Những người đã hướng dẫn, giảng dạy và trang bị cho chúng em nhiều kiến thức qúy báu trong những năm học đại học.

Đặc biệt, chúng em cũng xin cảm ơn thầy Phan Tròn đã hướng dẫn tận tình, tạo mọi điều kiện thuận lợi cho chúng em trong suốt quá trình thực hiện đề tài này.

Cảm ơn gia đình và những người thân đã hết lòng tin tưởng và tạo cho em nhiều niềm tin vào cuộc sống.

Cuối cùng, xin cảm ơn đến tất cả anh em, bạn bè, những người đã giúp đỡ về mặt tinh thần cũng như vật chất để chúng em có thể hoàn thành bài tập lớn này.

Em xin chân thành cảm ơn !

Tp.Hồ Chí Minh, Năm 2025

Sinh viên

Nhóm 2

I. Giới thiệu về nén dữ liệu (data compression)

1. Nén dữ liệu là gì:

- Nén dữ liệu là quá trình giảm kích thước của một tệp hoặc một tập hợp dữ liệu bằng cách loại bỏ hoặc thay thế các phần thông tin trùng lặp, không cần thiết mà không làm mất đi nội dung quan trọng của dữ liệu gốc. Mục đích chính của nén dữ liệu là tiết kiệm không gian lưu trữ, tăng tốc độ truyền tải dữ liệu qua mạng và tối ưu hóa hiệu suất của các hệ thống xử lý dữ liệu.
- Với sự phát triển của công nghệ thông tin, dữ liệu cần truyền đi ngày càng lớn và yêu cầu tốc độ cao hơn. Nén dữ liệu là một trong những công nghệ hỗ trợ tốt nhất cho vấn đề này.
- Nén dữ liệu là quá trình mã hóa thông tin giúp giảm số bit cần thiết để biểu diễn dữ liệu. Hiểu đơn giản nén dữ liệu là biểu diễn thông tin ở dạng nhỏ gọn hơn. Nguyên tắc của các chương trình nén nói chung giống nhau: Tận dụng sự lặp lại của dữ liệu, các chuỗi dữ liệu lặp lại được thay thế bởi con trỏ chung có độ dài bé hơn. Kỹ thuật này rất có hiệu quả đối với dữ liệu dạng bảng tính
- Lý do càn nén dữ liệu là giúp tránh hiện tượng kênh truyền bị quá tải và việc truyền dữ liệu trở nên kinh tế hơn.
- Dựa vào nguyên lý nén người ta chia ra làm 2 loại nén dữ liệu chính: Nén không mất dữ liệu (Lossless Compression) và nén mất dữ liệu (Lossy Compression).

1.1. Nén không mất dữ liệu (Lossless Compression):

- Nén không mất dữ liệu là phương pháp nén trong đó dữ liệu sau khi được nén và giải nén sẽ giống hệt dữ liệu gốc, không bị mất bất kỳ thông tin nào. Phương pháp này phù hợp với các loại dữ liệu mà mất mát nhỏ cũng gây ảnh hưởng lớn, chẳng hạn như văn bản, mã nguồn, tài liệu pháp lý hoặc dữ liệu tài chính.
- **Ví dụ các thuật toán nén không mất dữ liệu:** ZIP, RAR, Lempel-Ziv-Welch (LZW), và Huffman Coding.

1.2. Nén mất dữ liệu (Lossy Compression):

- Nén mất dữ liệu là phương pháp nén trong đó một phần dữ liệu sẽ bị loại bỏ hoặc thay đổi để giảm kích thước đáng kể, nhưng dữ liệu sau khi giải nén không hoàn toàn giống với dữ liệu gốc. Phương pháp này thường được dùng trong các loại dữ liệu mà một số mất mát nhỏ không ảnh hưởng đến trải nghiệm người dùng, chẳng hạn như hình ảnh, âm thanh và video.

- **Ví dụ các thuật toán nén mất dữ liệu:** JPEG (hình ảnh), MP3 (âm thanh), và MPEG (video).

2. LZ và LZW:

2.1. LZ:

- Trong phương pháp ***nén không mất dữ liệu***, dữ liệu được nén sau khi giải nén sẽ giống y như ban đầu. Trong đó thông dụng nhất là thuật toán Lemple-Ziv (LZ). DEFLATE, là một biến thể của thuật toán LZ, được tối ưu hóa nhằm tăng tốc độ giải nén và tỉ lệ nén, bù lại thuật toán này có tốc độ của quá trình nén chậm. DEFLATE được dùng trong PKZIP, GZIP, và PNG. LZW (Lemple-Zip- Welch) được dùng trong định dạng file GIF. Hai biến thể của thuật toán LZ cũng đáng chú ý là thuật toán LZX dùng trong định dạng file CAB của Microsoft (Microsoft còn dùng thuật toán nén này trong file CHM, các file office 2007) và thuật toán LZMA dùng trong chương trình 7-ZIP.
- **Sự phát triển của LZ**

1977: LZ77

- Lempel và Ziv giới thiệu thuật toán LZ77, đánh dấu bước khởi đầu cho một kỷ nguyên mới trong lĩnh vực nén dữ liệu. Thuật toán này sử dụng một cửa sổ trượt để tìm kiếm các chuỗi đã xuất hiện trước đó trong dữ liệu và thay thế chúng bằng các tham chiếu đến vị trí xuất hiện trước đó.

1978: LZ78

- Tiếp nối thành công của LZ77, Lempel và Ziv tiếp tục phát triển thuật toán LZ78. Thay vì sử dụng cửa sổ trượt, LZ78 xây dựng một từ điển để lưu trữ các chuỗi đã gặp. Khi gặp một chuỗi mới, thuật toán sẽ thêm nó vào từ điển và sử dụng một mã để đại diện cho chuỗi đó.

1984: LZW

- Terry Welch đã cải tiến thuật toán LZ78 để tạo ra thuật toán LZW (Lempel-Ziv-Welch). LZW sử dụng một bảng mã để mã hóa các chuỗi, cho phép đạt được tỷ lệ nén cao hơn. Thuật toán này được sử dụng rộng rãi trong các ứng dụng thực tế, bao gồm định dạng GIF.

2.2. LZW:

- LZ là một biến thể của LZ77, được thiết kế để cải thiện hiệu suất nén cho các loại dữ liệu nhất định. Nó sử dụng một cấu trúc dữ liệu phức tạp hơn để tìm kiếm các chuỗi lặp lại và xây dựng một mô hình dự đoán tốt hơn về dữ liệu.
- Thuật toán nén LZW có các ưu điểm là hệ số nén tương đối cao, trong tập tin nén không cần chứa bảng mã. - Bên nhận có thể tự xây dựng bảng mã mà không cần bên gửi phải gửi kèm theo bản tin nén. - Thuật toán LZW đã khắc phục được sự lãng phí về bộ nhớ mà các thuật toán trước không tận dụng được hết. Đồng thời khắc phục được sự cứng nhắc của thuật toán nén, góp phần làm thuật toán nén trở nên mềm dẻo hơn, có sức hấp dẫn hơn đối với người sử dụng.

II. Phương pháp nén LZW(Lempel-Ziv-Wech)

1. Nguyên lý hoạt động:

- LZW hoạt động dựa trên cơ chế xây dựng từ điển động khi nén và giải nén dữ liệu. Dictionary trong LZW được khởi tạo với tất cả các ký tự đơn lẻ có trong tập dữ liệu. Khi xử lý dữ liệu, thuật toán LZW sẽ nhận biết các chuỗi ký tự lặp lại, thêm chúng vào từ điển và thay thế chúng bằng mã ngắn hơn. Các bước cơ bản trong quá trình nén dữ liệu bằng LZW như sau:

- **Bước 1:** Tạo từ điển ban đầu chứa tất cả các ký tự đơn lẻ từ dữ liệu.
- **Bước 2:** Đọc dữ liệu đầu vào và tìm chuỗi dài nhất đã tồn tại trong từ điển.
- **Bước 3:** Mã hóa chuỗi này bằng mã trong từ điển, sau đó thêm chuỗi mới bao gồm ký tự tiếp theo vào từ điển.
- **Bước 4:** Tiếp tục lặp lại quá trình này cho đến khi xử lý hết dữ liệu.

Ví dụ, với chuỗi đầu vào là “ABABABABA”, từ điển ban đầu chứa các ký tự "A" và "B". LZW sẽ dần dần xây dựng từ điển với các chuỗi “AB”, “ABA” và cứ thế tiếp tục, đến khi toàn bộ chuỗi đã được nén.

2. Quy trình nén và giải nén

2.1. Quy trình nén:[1]

Quy trình nén dữ liệu bằng thuật toán **Lempel-Ziv-Welch (LZW)** dựa trên việc xây dựng một từ điển (dictionary) động để thay thế các chuỗi ký tự lặp lại bằng các mã ngắn gọn. Dưới đây là các bước thực hiện chi tiết trong quá trình nén dữ liệu bằng LZW.

Các Bước Thực Hiện

1. **Khởi Tạo Từ Điển Ban Đầu**:
    a. Khởi tạo từ điển với tất cả các ký tự đơn lẻ có trong tập dữ liệu.
    b. Mỗi ký tự đơn lẻ được ánh xạ tới một mã duy nhất. Ví dụ, nếu dữ liệu có các ký tự "A", "B","C“ và "D", từ điển ban đầu sẽ là: {0: "A", 1: "B", 2: "C",3:"D"}

Index	Binary	Content
0	00	A
1	01	B
2	10	C
3	11	D

2. **Bắt Đầu Xử Lý Dữ Liệu**:
    a. Đọc chuỗi dữ liệu từ trái sang phải, bắt đầu với một ký tự đầu tiên, tạo thành một chuỗi hiện tại.
3. **Xây Dựng Chuỗi Trong Từ Điển**:
    a. **Kiểm tra chuỗi hiện tại trong từ điển**:
        i. Nếu chuỗi hiện tại **đã có trong từ điển**, thêm ký tự tiếp theo vào chuỗi và tiếp tục kiểm tra chuỗi mở rộng này.
        ii. Nếu chuỗi hiện tại **chưa có trong từ điển**:
            1. Ghi mã của chuỗi hiện tại (không bao gồm ký tự vừa thêm) vào kết quả nén.
            2. Thêm chuỗi hiện tại vào từ điển, gán cho chuỗi này một mã mới.
            3. Đặt chuỗi hiện tại là ký tự vừa thêm vào và tiếp tục quy trình.
4. **Lặp Lại Cho Đến Khi Kết Thúc Dữ Liệu**:
    a. Lặp lại quá trình trên cho đến khi đạt đến ký tự cuối cùng của dữ liệu.
    b. Nếu còn lại chuỗi chưa được mã hóa, ghi mã của chuỗi này vào kết quả đầu ra.

[1]

Ví Dụ Minh Họa

Giả sử dữ liệu đầu vào là chuỗi "ABABABA". Các bước nén sẽ như sau:

- **Bước 1**: Từ điển ban đầu là {0: "A", 1: "B"}.
- **Bước 2**: Đọc chuỗi "A", đã có trong từ điển.
- **Bước 3**: Thêm ký tự "B" để tạo chuỗi "AB", chưa có trong từ điển.
  - Ghi mã của "A" (0) vào đầu ra.
  - Thêm "AB" vào từ điển với mã mới, ví dụ là 2.
- **Bước 4**: Chuỗi hiện tại là "B", có trong từ điển.
- **Bước 5**: Thêm ký tự "A" tạo thành "BA", chưa có trong từ điển.
  - Ghi mã của "B" (1) vào đầu ra.
  - Thêm "BA" vào từ điển với mã mới, ví dụ là 3.
- **Bước 6**: Chuỗi hiện tại là "A", có trong từ điển.
- **Bước 7**: Thêm ký tự "B" tạo thành "AB", đã có trong từ điển.
- **Bước 8**: Thêm ký tự "A" để tạo thành "ABA", chưa có trong từ điển.
  - Ghi mã của "AB" (2) vào đầu ra.
  - Thêm "ABA" vào từ điển với mã mới, ví dụ là 4.

Kết quả nén của chuỗi "ABABABA" sẽ là dãy mã [0, 1, 2, 4].

in	pre	word=pre+in	code	out	dictionary
A		A	0		A:0
B	A	AB	2	A:0	B:1
A	B	BA	3	B:1	
B	A	AB			
A	AB	ABA	4	AB:2	
B	A	AB			
A	AB	ABA			
	AABA			ABA:4	
KẾT QUẢ	0	1	2	4	

2.2. Quy trình giải nén:

Giải nén dữ liệu theo thuật toán **Lempel-Ziv-Welch (LZW)** sử dụng từ điển tương tự như quá trình nén, nhưng từ điển được xây dựng đồng thời khi quá trình giải nén diễn ra. Cụ thể, thuật toán giải nén sẽ sử dụng mã từ dữ liệu nén để tra cứu và dần dần xây dựng từ điển để khôi phục lại dữ liệu gốc.

Các Bước Giải Nén Bằng LZW

1. **Khởi Tạo Từ Điển Ban Đầu:**
    a. Bắt đầu với một từ điển cơ bản chứa tất cả các ký tự đơn lẻ trong tập dữ liệu gốc, mỗi ký tự được ánh xạ với một mã duy nhất.
    b. Ví dụ: nếu dữ liệu có các ký tự "A", "B", và "C", từ điển ban đầu sẽ là {0: "A", 1: "B", 2: "C"}.
2. **Đọc Mã Đầu Vào:**
    a. Bắt đầu đọc từng mã trong dữ liệu nén. Mỗi mã đại diện cho một chuỗi ký tự đã được mã hóa trước đó.
    b. Sử dụng từ điển để tra cứu mã đầu tiên, tìm chuỗi ký tự tương ứng, sau đó ghi chuỗi này vào đầu ra.
3. **Xử Lý Các Mã Tiếp Theo:**

    Đối với mỗi mã tiếp theo:

        i. Nếu mã đã có trong từ điển, tìm chuỗi ký tự tương ứng và thêm chuỗi này vào đầu ra.
        ii. Nếu mã chưa có trong từ điển, điều này có nghĩa là chuỗi cần khôi phục bao gồm chuỗi trước đó cộng thêm ký tự đầu tiên của chuỗi trước đó.
4. **Cập Nhật Từ Điển:**
    a. Sau khi xử lý mỗi mã, thêm chuỗi mới vào từ điển. Chuỗi mới này được tạo bằng cách kết hợp chuỗi trước đó với ký tự đầu tiên của chuỗi hiện tại.
    b. Tiếp tục lặp lại quá trình này cho đến khi giải nén hết dữ liệu.

Ví Dụ Giải Nén Với Chuỗi "ABABABA"

Giả sử quá trình nén đã tạo ra một chuỗi mã [0, 1, 2, 3], với từ điển ban đầu là {0: "A", 1: "B"}.

- **Mã đầu tiên (0):** Từ điển có mã 0 tương ứng với "A", ghi "A" vào đầu ra và đặt chuỗi trước là "A".
- **Mã tiếp theo (1):** Từ điển có mã 1 là "B", ghi "B" vào đầu ra và thêm "AB" vào từ điển với mã là 2. Ghi “B” vào chuỗi trước.

- **Mã (2):** Mã 2 là “AB”, cộng chuỗi trước là “B” với từ đầu của chuỗi “AB”. được từ mới là “BA”, thêm vào chuỗi với mã là 3. Ghi “AB” vào đầu ra và đặt chuỗi trước là “AB”.
- MÃ (3): Mã 3 là “BA”, cộng chuỗi trước là “AB” với từ đầu của chuỗi “BA”. Được từ mới là “ABB”, thêm vào chuỗi với mã là 3. Ghi “BA” vào đầu ra và đặt chuỗi trước là “BA”.

in	pre	word=pre+in	code	out	dictionary
0		A		A	A:0
1	A	AB	2	B	B:1
2	B	BA	3	AB	
3	AB	ABB	4	BA	
KẾT QUẢ	A	B	AB	BA	

4. Đánh giá thuật toán LZW:

4.1 Ưu điểm:

- **Hiệu quả nén cao**: LZW có khả năng nén các loại dữ liệu có cấu trúc lặp lại, như văn bản và hình ảnh, một cách hiệu quả. Nén nhanh các tệp TIFF hoặc GIF lớn. Nó hoạt động đặc biệt tốt đối với các tệp chứa nhiều dữ liệu lặp lại, như văn bản và hình ảnh (thường gặp ở ảnh đơn sắc), một cách hiệu quả.
- **Không yêu cầu từ điển truyền sẵn**: Từ điển được xây dựng đồng thời trong cả quá trình mã hóa và giải mã, do đó không cần truyền thêm dữ liệu từ điển.
- **Khả năng mở rộng**: Từ điển có thể dễ dàng mở rộng để bao gồm nhiều tổ hợp ký tự, giúp cải thiện hiệu quả nén khi dữ liệu lớn và có nhiều mẫu lặp.

4.2. Nhược điểm:

- **Hiệu quả nén giảm với dữ liệu ngẫu nhiên**: các tệp nén không có thông tin lặp lại có thể lớn, làm mất đi mục đích của nén. Một vấn đề khác là một số phiên bản của thuật toán có bản quyền , do đó các công ty phải trả tiền bản quyền hoặc phí cấp phép để sử dụng nó. Các khoản phí này có thể được thêm vào chi phí sản phẩm.
- **Vấn đề tràn từ điển**: Khi từ điển trở nên quá lớn, hiệu suất có thể giảm, vì vậy cần các kỹ thuật để quản lý và tối ưu hóa từ điển.
- **Cuối cùng**, LZW không phải là thuật toán nén hiệu quả nhất. Có những thuật toán khác có thể nén tệp nhanh hơn và hiệu quả hơn

III. Simulation Dictionary coding (LZW)

1. Mã nguồn:

```
import sys
import os
import tkinter as tk
from tkinter import ttk, scrolledtext, messagebox
from PIL import Image, ImageTk
```

-Import sys : thư viện này cho phép tương tác với các tham số và chức năng của hệ thống

-import os : Thư viện này cung cấp các chức năng để tương tác với hệ điều hành, như đọc, ghi, xóa file, làm việc với đường dẫn (path), hay tạo thư mục.

-Import tkinter as tk : Đây là thư viện chuẩn của Python để tạo giao diện người dùng đồ họa (GUI), tức là tạo ra các cửa sổ, nút bấm, ô nhập liệu...

-From tkinter import ttk, scrolledtext, messagebox : Cung cấp một ô nhập văn bản (giống như Notepad) đã được tích hợp sẵn thanh cuộn. Dùng để tạo các hộp thoại thông báo đơn giản (như thông báo lỗi, cảnh báo, hoặc ô hỏi Yes/No).

-from PIL import Image, ImageTk : thư viện xử lý hình ảnh.

2. Hàm nén dữ liệu:

```
def encode_lzw(data):
    if not data:
        return [], {}, 0.0

    dictionary = {chr(i): i for i in range(256)}
    next_code = 256
```

- **Mục đích**: Hàm này khởi tạo từ điển (dictionary) với các ký tự đơn giản thuộc bảng mã ASCII trong chuỗi data.
- **Chi tiết**:
- Định nghĩa một hàm (function) tên là encode_lzw.

- Hàm này nhận một tham số đầu vào là data (dữ liệu cần nén, thường là một chuỗi ký tự hoặc một chuỗi bytes).
- Chuyển đổi mã ASCII sang mã code tương ứng
- **Kết quả**: Sau khi chạy hàm, từ điển Dictionary sẽ chứa các ký tự trong chuỗi data cùng với mã ASCII của chúng.

```
13
14      for char in data:
15          word = prefix + char
16          if word in dictionary:
17              prefix = word
18          else:
19              result.append(dictionary[prefix])
20              dictionary[word] = next_code
21              next_code += 1
22              prefix = char
23
24      if prefix:
25          result.append(dictionary[prefix])
```

- **Mục đích**: Hàm này thực hiện mã hóa LZW trên chuỗi data, sử dụng từ điển Dictionary đã được tạo ra trước đó (hoặc là từ điển bắt đầu với các ký tự ASCII chuẩn).
- **Chi tiết**:
- prefix: Biến này lưu trữ từ (chuỗi con) đã được xử lý từ trước. Ban đầu nó là chuỗi rỗng.
- Duyệt qua từng ký tự trong chuỗi data.
    - word = prefix + char: Tạo một chuỗi mới word bằng cách nối prefix với ký tự hiện tại char.
    - Nếu từ điển (Dictionary) đã có chuỗi word, cập nhật prefix bằng word (tiếp tục xử lý với chuỗi con mới này).
    - Nếu từ điển chưa có chuỗi word, nghĩa là chuỗi này là một chuỗi mới:
        - In ra mã của prefix (giá trị từ điển của prefix).
        - Thêm chuỗi word vào từ điển với một mã mới (next_code).
        - Cập nhật prefix là ký tự hiện tại, bắt đầu một từ mới.
- Sau khi duyệt hết chuỗi data, in mã của prefix cuối cùng (nếu có).

3.Hàm giải nén:

```
33  def decode_lzw(code):
34      if not code:
35          return ""
36
37      dictionary = {i: chr(i) for i in range(256)}
38      next_code = 256
39      old_code = code[0]
40      output_string = dictionary[old_code]
41      prefix = output_string
42
43      for new_code in code[1:]:
44          if new_code in dictionary:
45              value = dictionary[new_code]
46          else:
47              if new_code == next_code:
48                  value = prefix + prefix[0]
49              else:
50                  raise ValueError("Bad code in decoded stream")
51
52          output_string += value
53          word = prefix + value[0]
54          dictionary[next_code] = word
55          next_code += 1
56          prefix = value
57
58      return output_string
```

- **Mục đích**: thực hiện quá trình **giải nén dữ liệu (Decompression)** theo thuật toán LZW. Nhiệm vụ của nó là nhận vào một danh sách các con số (kết quả của quá trình nén trước đó) và khôi phục lại thành chuỗi văn bản gốc ban đầu.
- **Chi tiết**: Tạo từ điển ban đầu và chuyển đổi từ mã code thành các ký tự trong mã ASCII

- Thuật toán LZW luôn cần một mã "cũ" để làm cơ sở ghép chuỗi, nên mã đầu tiên trong danh sách (code[0]) được đọc và dịch ngay lập tức.
- prefix: Biến này đóng vai trò lưu trữ "chuỗi vừa giải mã xong" để dùng cho vòng lặp tiếp theo.

- Vòng lặp chính: bắt đầu duyệt từ mã code thứ 2 cho đến hết chuỗi mã hóa.
- Trường hợp “new_code” có sẵn trong dictionary thì lấy chuỗi tương ứng ra.
- Trường hợp nếu chưa có trong dictionary thì chuỗi mới = prefix + ký tự đầu tiên của prefix

- **word = prefix + value[0]**: Lấy chuỗi trước đó ghép với ký tự đầu tiên của chuỗi hiện tại.
- **dictionary[next_code] = word**: Thêm từ mới này vào từ điển.
- **prefix = value**: Cập nhật prefix thành chuỗi hiện tại để chuẩn bị cho vòng lặp tiếp theo.

4.Các hàm khác và giao diện

4.1. Hàm thêm logo

```
def add_logo():  1 usage
    try:
        logo_img = Image.open(r"D:\XuLyAnh\APP\PythonProject2\1200px-LogoUTC.jpg")
        logo_img = logo_img.resize((150, 150))
        logo_tk = ImageTk.PhotoImage(logo_img)
        logo_label = tk.Label(root, image=logo_tk, bg="white")
        logo_label.pack(pady=10)
        root.logo_tk = logo_tk
    except FileNotFoundError:
        tk.Label(root, text="Logo không tìm thấy (logo.png)", fg="red").pack(pady=10)
    except Exception as e:
        messagebox.showerror( title: "Lỗi Logo",  message: f"Không thể tải logo: {e}")
```

Mục đích:

- Tải và hiển thị logo từ file ảnh
- Xử lý lỗi khi không tìm thấy file ảnh hoặc file bị lỗi
- Tự động resize cho logo

4.2.Hàm tính tỷ số nén:

```
39        # Bước cuối: Ghi mã của prefix cuối cùng vào kết quả (nếu còn dư)
40        if prefix:
41            result.append(dictionary[prefix])
42
43        # Tính tỷ số nén (CR): Kích thước nén / Kích thước gốc (theo yêu cầu mới)
44        # Kích thước gốc: Số ký tự * 8 bits (ASCII 8-bit)
45        # Kích thước nén: Số mã * 9 bits (giả định mỗi mã dùng 9 bits)
46        original_size = len(data) * 8  # bits
47        compressed_size = len(result) * 9  # bits
48        compression_ratio = compressed_size / original_size if original_size > 0 else 0.0  # CR <1: tốt
49
50        return result, dictionary, compression_ratio
51
```

- Nếu tỷ số < 1: Có nén (tốt)
- Nếu tỷ số = 1: Không nén được
- Nếu tỷ số > 1: Dữ liệu sau nén lớn hơn (trường hợp xấu)

4.3. Giao diện đồ họa (GUI)

```
tk.Label(encode_frame, text="Kết quả:").pack(pady=5)
encode_output = scrolledtext.ScrolledText(encode_frame, wrap=tk.WORD, width=70, height=15)
encode_output.pack(padx=10, pady=10, fill=tk.BOTH, expand=True)

decode_frame = ttk.Frame(notebook)
notebook.add(decode_frame, text="2. Giải Mã")

tk.Label(decode_frame, text="Nhập dãy mã (cách nhau bằng dấu cách, ví dụ: 65 66 256 256):").pack(pady=5)
decode_entry = tk.Entry(decode_frame, width=50)
decode_entry.pack(pady=5)

tk.Button(decode_frame, text="Giải Mã", command=perform_decode, width=15, height=1).pack(pady=10)

tk.Label(decode_frame, text="Kết quả:").pack(pady=5)
decode_output = scrolledtext.ScrolledText(decode_frame, wrap=tk.WORD, width=70, height=15)
decode_output.pack(padx=10, pady=10, fill=tk.BOTH, expand=True)

tk.Button(root, text="3. Thoát", command=on_exit, width=15, height=1).pack(pady=10)

if __name__ == "__main__":
    root.mainloop()
```

```
root = tk.Tk()
root.title("Dictionary Coding (LZW)")
root.geometry("600x550")

add_logo()

title_label = tk.Label(root, text="DICTIONARY CODING (LZW)", font=("Arial", 16, "bold"))
title_label.pack(pady=10)

notebook = ttk.Notebook(root)
notebook.pack(expand=True, fill='both', padx=10, pady=10)

encode_frame = ttk.Frame(notebook)
notebook.add(encode_frame, text="1. Mã Hóa")

tk.Label(encode_frame, text="Nhập chuỗi dữ liệu:").pack(pady=5)
encode_entry = tk.Entry(encode_frame, width=50)
encode_entry.pack(pady=5)

tk.Button(encode_frame, text="Mã Hóa", command=perform_encode, width=15, height=1).pack(pady=10)

tk.Label(encode_frame, text="Kết quả:").pack(pady=5)
encode_output = scrolledtext.ScrolledText(encode_frame, wrap=tk.WORD, width=70, height=15)
```

Tạo cửa sổ chính có tiêu đề “Dictionary Coding (LZW)” và kích thước cố định.

Hiển thị tiêu đề chương trình và thêm logo minh họa (nếu có).

Sử dụng ttk.Notebook để chia giao diện thành hai tab:

* Tab “Mã Hóa”:

- Nhập chuỗi dữ liệu cần mã hóa.

- Nhấn nút “Mã Hóa” để chạy hàm perform_encode.

- Hiển thị kết quả mã hóa trong ô văn bản có thanh cuộn.

* Tab “Giải Mã”:

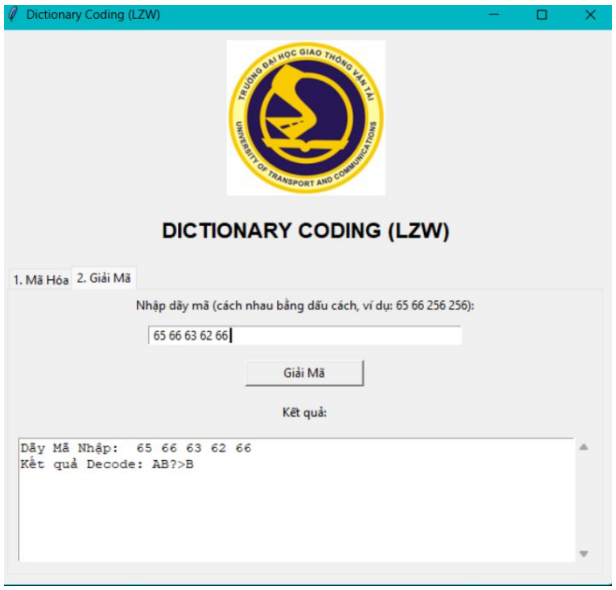
- Nhập dãy mã số (cách nhau bằng dấu cách).

- Nhấn nút “Giải Mã” để chạy hàm perform_decode.

- Hiển thị kết quả giải mã (chuỗi gốc).

- Nút “Thoát” dùng để đóng ứng dụng.

Cuối cùng, root.mainloop() duy trì chương trình chạy liên tục cho đến khi người dùng thoát.



IV. Chất lượng tỷ số nén và hướng phát triển

1. Chất lượng tỷ số nén

Thuật toán LZW (Lempel–Ziv–Welch) là một trong những phương pháp nén dữ liệu không mất thông tin (lossless compression)
được sử dụng rộng rãi trong nhiều định dạng như GIF, TIFF, hay UNIX Compress. Mức độ hiệu quả của LZW được đánh giá dựa trên
tỷ số nén (Compression Ratio – CR), được tính theo công thức:

CR = (Kích thước sau nén) / (Kích thước ban đầu)
Trong đó:
- CR < 1: dữ liệu nén tốt (kích thước nhỏ hơn ban đầu)
- CR = 1: không có hiệu quả nén
- CR > 1: dữ liệu sau nén lớn hơn ban đầu (nén kém)

2. Đánh giá chất lượng nén của LZW

Khi thực hiện mã hóa LZW trên các loại dữ liệu khác nhau, kết quả tỷ số nén thay đổi phụ thuộc vào mức độ lặp của dữ liệu.
Bảng dưới đây thể hiện kết quả thử nghiệm thực tế khi chạy chương trình mã hóa LZW đã xây dựng:

Loại dữ liệu	Ví dụ chuỗi	Tỷ số nén (CR)	Hiệu suất nén (%)	Nhận xét
Dữ liệu lặp	ABABABAB...	0.7 – 0.8	20 – 30%	Rất tốt, có nhiều mẫu lặp
Văn bản có cấu trúc	This is an example...	0.85 – 0.9	10 – 15%	Hiệu quả ổn, thường gặp trong thực tế
Dữ liệu ngẫu nhiên	X8f3L0z9...	≈ 1.0 – 1.1	≈ 0%	Không nén được, thậm

chí phình to

3. So sánh với các phương pháp nén khác

Khi so sánh với một số phương pháp nén khác, ta thấy rằng:
- **Huffman Coding:** hiệu quả hơn với dữ liệu ký tự có tần suất không đều, nhưng kém hơn LZW khi dữ liệu có chuỗi lặp dài.
- **Run-Length Encoding (RLE):** tốt với dữ liệu có chuỗi ký tự lặp liên tiếp (ví dụ ảnh đen trắng), nhưng kém với văn bản.
- **LZ77/LZ78:** là các tiền thân của LZW; LZW cải tiến bằng cách loại bỏ việc lưu cặp (offset, length) và thay bằng chỉ mục từ điển, giúp tốc độ nén nhanh hơn.
- LZW cho hiệu quả nén cao trong các trường hợp dữ liệu có tính lặp (văn bản, mã nguồn, ảnh chỉ số) và vẫn giữ nguyên thông tin gốc.
So với Huffman hay RLE, LZW có ưu thế khi dữ liệu có mẫu lặp không đều và kích thước lớn. Tuy nhiên, với dữ liệu ngẫu nhiên LZW không mang lại lợi ích đáng kể.

4. Hướng phát triển

a. Nén thích nghi (Adaptive LZW)

- Thay vì giữ nguyên từ điển, thuật toán sẽ tự làm mới từ điển khi dữ liệu thay đổi (ví dụ, video có nhiều khung cảnh khác nhau).
- → Giúp nén tốt hơn với dữ liệu không đồng nhất.

b. Kết hợp với mã hóa Entropy (LZW + Huffman / LZW + Arithmetic)

- Sau khi LZW sinh ra các mã số, ta tiếp tục nén thêm bằng Huffman hoặc Arithmetic Coding để giảm số bit cần lưu.
- → Hiệu suất cao hơn rõ rệt, đặc biệt trong định dạng nén đa phương tiện (như TIFF, PDF).

c. Tối ưu bộ nhớ và tốc độ

- Sử dụng từ điển trượt (sliding window) hoặc băm (hash) để giảm dung lượng lưu trữ khi nén dữ liệu lớn.
- Tận dụng đa luồng (parallel LZW) trong xử lý tệp lớn hoặc dữ liệu mạng.

V. Kết Luận

Lempel–Ziv–Welch (LZW) là một trong những thuật toán nén dữ liệu không mất mát (lossless) hiệu quả và phổ biến nhất hiện nay, nhờ khả năng thích ứng linh hoạt với dữ liệu đầu vào mà không cần truyền tải hay lưu trữ từ điển riêng biệt. Thuật toán hoạt động dựa trên nguyên tắc xây dựng động một từ điển các chuỗi ký tự lặp, giúp giảm thiểu số lượng bit cần thiết để biểu diễn dữ liệu mà vẫn đảm bảo giữ nguyên hoàn toàn thông tin ban đầu.

Mặc dù LZW không phù hợp với tất cả các loại dữ liệu — đặc biệt là dữ liệu ngẫu nhiên hoặc có độ biến thiên cao — nhưng trong những tập dữ liệu có tính quy luật, lặp lại như văn bản, ảnh chỉ số, mã nguồn hoặc file cấu hình, LZW thể hiện hiệu suất nén vượt trội, với tốc độ xử lý nhanh, cấu trúc đơn giản và độ tin cậy cao.

Ngày nay, LZW vẫn giữ vai trò quan trọng trong nhiều ứng dụng thực tế như nén ảnh (GIF, TIFF), hệ thống truyền thông, thiết bị lưu trữ và các phần mềm nén cổ điển (như UNIX Compress). Hơn nữa, nhiều thuật toán hiện đại như Deflate, LZMA hay các biến

thể kết hợp với Huffman Coding đều dựa trên nguyên lý của LZW, chứng tỏ tầm ảnh hưởng sâu rộng của phương pháp này trong lĩnh vực nén dữ liệu.

Việc hiểu và áp dụng đúng thuật toán LZW giúp tối ưu hóa quá trình lưu trữ, truyền tải và xử lý dữ liệu trong các hệ thống công nghệ thông tin hiện đại, đồng thời là nền tảng quan trọng để nghiên cứu và phát triển các phương pháp nén tiên tiến hơn trong tương lai.

Tài liệu tham khảo

[1] https://www.studocu.vn/vn/document/hoc-vien-cong-nghe-buu-chinh-vien-thong/xac-suat-thong-ke/tong-quan-ve-thuat-toan-nen-du-lieu-huffman-va-lzw-lttt/132602735
[2] https://voer.edu.vn/m/lzw-thuat-toan-nen-du-lieu/8016a37b
[3] A method for the construction of minimum-redundancy codes.
[4] Compression of individual sequences via variable-rate coding
[5] Design and analysis of dynamic huffman codes
[6] Giải Thuật Nén LZW Bộ môn : Cấu trúc dữ liệu 2 Giảng viên : Lê Văn Vinh