ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA

ĐẠI SỐ TUYẾN TÍNH (MT1007)

Báo cáo bài tập lớn

Mã Hamming (7,4)

Cơ Chế và Ứng Dụng trong Xử Lý Lỗi

Giáo viên hướng dẫn: Ths.Nguyễn Hữu Hiệp
Sinh viên thực hiện: Nguyễn Ngọc Khắc Thành
Nguyễn Ngọc Loan Anh
Nguyễn Ngọc Như Ý
Nguyễn Ngọc Phúc Huy
Nguyễn Phan Hoàng Vinh
Nguyễn Phúc Nhân
Nguyễn Phúc Minh Tâm
Nguyễn Quang Bảo

THÀNH PHỐ HỒ CHÍ MINH, THÁNG 4 2024

Mục lục

1 Danh sách thành viên & Khối lượng công việc

STT	Họ và tên	MSSV	Nội dung thực hiện	Mức độ hoàn thành
1	Nguyễn Phúc Nhân	2312438	Viết code	100%
2	Nguyễn Phúc Minh Tâm	2313033	Viết báo cáo	100%
3	Nguyễn Ngọc Khắc Thành	2012051	Viết báo cáo	100%
4	Nguyễn Ngọc Loan Anh	2012604	Làm PowerPoint	100%
5	Nguyễn Ngọc Như Ý	2314037	Làm PowerPoint	100%
6	Nguyễn Ngọc Phúc Huy	2314048	Soạn lý thuyết	100%
7	Nguyễn Phan Hoàng Vinh	2313921	Soạn lý thuyết	100%
8	Nguyễn Quang Bảo	2314044	Soạn lý thuyết	100%

2 Yêu cầu

Như chúng ta đã biết, Mã Hamming là một chủ đề quan trọng trong lĩnh vực kỹ thuật số và truyền thông, đặc biệt là trong việc phát hiện và sửa lỗi. Mã Hamming giúp phát hiện và sửa chữa các lỗi này, đảm bảo được sự ổn định trong quá trình truyền tải, giảm thiểu sự sai lệch do sự đảo lộn các bit số nhị phân gây ra.

Trong các hệ thống kỹ thuật số, dữ liệu được truyền cho giao tiếp có thể bị hỏng do nhiễu bên ngoài và bất kỳ lỗi vật lý nào khác. Mã Hamming giúp phát hiện và sửa chữa các lỗi này, đảm bảo tính chính xác và an toàn của dữ liệu. Điều này đặc biệt quan trọng trong các lĩnh vực như truyền thông, lưu trữ dữ liệu, và hệ thống máy tính.

Ngoài ra, Mã Hamming là một phần quan trọng của các khóa học về kỹ thuật số, truyền thông và lý thuyết thông tin. Hiểu biết về mã Hamming giúp sinh viên nắm bắt được cách thức hoạt động của các hệ thống truyền thông hiện đại và phương pháp để đảm bảo tính chính xác của dữ liệu.

Do đó, trong nội dung môn học Đại Số Tuyến Tính, nhóm cần tiến hành mở rộng tìm hiểm thêm về mã Hamming cũng như áp dụng mã Hamming vào 1 bài toán kỹ thuật cụ thể, sau đây là tổng quan qua những vấn đề mà nhóm sẽ làm rõ trong bài báo cáo này:

1. Giới thiệu thành viên và phân rõ nhiệm vụ, trách nhiệm
2. Giới thiệu sơ lược về những yêu cầu đề tài cần đạt được
3. Giới thiệu về 1 số định nghĩa liên quan đến đề tài
4. Làm rõ cơ sở lý thuyết của Mã Hamming (7,4)
5. Xây dựng mô hình về mã Hamming bằng ngôn ngữ Python
6. Giới thiệu 1 số ứng dụng thực tế của Mã Hamming
7. Đưa ra lời tổng kết, đúc kết kinh nghiệm của nhóm

3 Giới thiệu

3.1 Tổng quan ngành khoa học mật mã

Khoa học mật mã, còn được gọi là mật mã học, là một lĩnh vực của khoa học máy tính và toán học tập trung vào việc bảo mật thông tin. Nó bao gồm việc nghiên cứu, thiết kế và ứng dụng các thuật toán và hệ thống để bảo vệ dữ liệu từ việc truy cập hoặc thay đổi không hợp pháp.

Hình 1: Sơ đồ giải thuật của công việc mã hóa

Mật mã học không chỉ bao gồm việc mã hóa thông tin (chuyển đổi thông tin từ dạng có thể đọc được thành dạng không thể đọc được mà không có khóa) và giải mã (chuyển đổi thông tin từ dạng không thể đọc được trở lại dạng có thể đọc được với khóa), mà còn bao gồm cả các lĩnh vực như chữ ký số, bảo mật mạng và bảo mật hệ thống.

3.2 Giới thiệu về mã Hamming

Mã Hamming là một trong những thuật toán mã hóa lỗi phổ biến nhất được sử dụng trong các hệ thống truyền thông và lưu trữ dữ liệu. Được đặt tên theo Richard Hamming, người phát minh ra nó, mã Hamming có khả năng phát hiện và sửa lỗi đơn trong mỗi khối dữ liệu.

Trong mã Hamming, một số bit được sử dụng để mã hóa thông tin, trong khi các bit khác được sử dụng để kiểm tra lỗi. Điều này cho phép hệ thống phát hiện và sửa lỗi tự động, giảm thiểu khả năng mất mát thông tin do lỗi truyền dẫn.

4 Cơ sở lý thuyết

4.1 Mã Hamming

Mã Hamming được phát minh bởi Richard W. Hamming vào những năm 1950 khi làm việc tại Bell Labs, một phần của công ty điện thoại ATT. Richard Hamming là một nhà toán học và kỹ sư điện tử nổi tiếng, và ông đã đóng góp rất nhiều cho lĩnh vực mã hóa và lý thuyết thông tin. Mã Hamming được phát triển với mục đích giảm thiểu các lỗi trong truyền thông số, đặc biệt là trong các hệ thống số có nhiễu, mà nhiễu có thể gây ra sự thay đổi không mong muốn trong dữ liệu. Mã Hamming cung cấp một phương pháp hiệu quả để phát hiện và thậm chí sửa chữa các lỗi như vậy.

Bằng cách sử dụng các kỹ thuật mã hóa đặc biệt, mã Hamming có thể phát hiện và sửa chữa một số lỗi nhất định trong dữ liệu mà không cần phải truy xuất đến dữ liệu gốc. Điều này giúp tăng cường tính tin cậy và độ tin cậy của các hệ thống số, đặc biệt là khi dữ liệu phải được truyền qua các kênh không đáng tin cậy như các kết nối mạng hoặc các ổ đĩa lưu trữ. Kể từ khi được phát hiện và thể hiện được sự tiện lợi, nó được sử dụng rộng rãi để phát hiện và sửa lỗi trong dãy bit dữ liệu. Nói ngắn gọn, mã Hamming cho ta một phương pháp để phát hiện và sửa lỗi được thể hiện qua các bước sau:
Mã hóa dữ liệu ban đầu (bằng cách thêm các thông tin đặc tả dữ liệu gốc) → Truyền thông tin đi → Nhận về dữ liệu và kiểm tra → Trả về dữ liệu gốc.

- Mã Hamming thường được thể hiện dưới dạng (n, k). Có nghĩa là dữ liệu đầu vào của chúng ta sẽ là một chuỗi gồm k bit dưới dạng nhị phân, dữ liệu này sẽ được mã hóa bằng cách thêm (n-k) bits tương ứng. Sau đó, n bits này sẽ được phân tích và so sánh để biết có sự sai sót nào xảy ra không.
- Mã Hamming có thể phát hiện một bit hoặc hai bit bị lỗi. Một số mã Hamming phổ biến là Hamming(11,7), Hamming(7,4), Hamming(8,7), Hamming(3,1). Từ khi được phát triển, Hamming đã thể hiện sự vượt trội hơn so với các mã kiểm tra trước đó như mã chẵn lẻ (chỉ có thể phát hiện lỗi một bit và không thể sửa lỗi), mã lặp (tốn nhiều không gian lưu trữ, hiệu quả sửa lỗi thấp).
- Mã Hamming thường được ứng dụng trong các lĩnh vực yêu cầu độ chính xác cao và độ tin cậy cao trong truyền và phân tích dữ liệu. Một số ngành sử dụng

mã Hamming: truyền thông và mạng, đọc ghi mã, giao tiếp không dây, truyền thông vệ tinh, bảo mật và mã hóa, ...

4.2 Mã Hamming(7,4)

- Mã Hamming(7,4) là một loại mã Hamming cụ thể được thiết kế để mã hóa dãy gồm 4 bit dữ liệu thành một dãy 7 bit, bao gồm 4 bit dữ liệu ban đầu và 3 bit kiểm tra. Thuật toán (7,4) của Hamming có thể sửa chữa bất cứ một bit lỗi nào, và phát hiện tất cả lỗi của 1 bit hoặc 2 bit gây ra. Điều này có nghĩa là đối với tất cả các phương tiện truyền thông không có chùm lỗi đột ngột xảy ra, mã Hamming(7,4) rất có hiệu quả (trừ phi phương tiện truyền thông có độ nhiễu rất cao thì nó mới có thể gây cho 2 bit trong số 7 bit truyền bị thay đổi).

- Cho các bit dữ liệu d_1, d_2, d_3, d_4

- Mã Hamming có thể xác định các bit chẵn lẻ p_1, p_2 và p_3 là:

$$p_1 = d_1 + d_2 + d_4$$

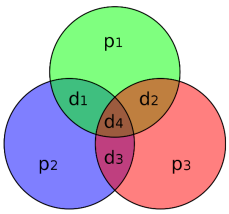
$$p_2 = d_1 + d_3 + d_4$$

$$p_3 = d_2 + d_3 + d_4$$

- Có một phương trình khác cho một bit chẵn lẻ có thể được sử dụng trong mã Hamming:

$$p_4 = d_1 + d_2 + d_3$$

- Mã Hamming hợp lệ có thể sử dụng bất kỳ 3 trong 4 định nghĩa bit chẵn lẻ ở trên và cũng có thể đặt các bit chẵn lẻ ở bất kỳ vị trí nào trong tập hợp 7 bit dữ liệu và chẵn lẻ. Ví dụ ta sử dụng tổ hợp p_1, p_2, p_3:



- Các bit chẵn lẻ p1,p2,p3 sẽ được tạo ra bằng cách tính chẵn lẽ của số lượng các số bit 1.
- Trong đó, ở mỗi vòng tròn, số lượng của bit 1 phải đảm bảo là một số chẵn.
- Từ đó, ta sẽ có một mã gồm 7 bit có dạng: $p_1p_2d_1p_3d_2d_3d_4$.

p_1	p_2	d_1	p_3	d_2	d_3	d_4

- Tuy nhiên ta cũng có thể đặt các bit chẵn lẻ ở các vị trí khác nhau, chúng sẽ đưa ra các kết quả khác nhau nhưng chúng vẫn là mã Hamming và được xem là tương đương.

4.3 Ma trận tạo mã

- Mã Hamming(7,4) sử dụng ma trận tạo mã để chuyển dữ liệu từ dãy bit dữ liệu 4 bit ban đầu để tạo thành dãy bit dữ liệu gồm 7 bit với 3 bit kiểm tra được thêm vào. Ta tiến hành tạo ma trận G bằng cách:

1. Biểu diễn từng bit dữ liệu bằng vectơ cột như sau:

$$d1 = \begin{bmatrix}1\\0\\0\\0\end{bmatrix} \; d2 = \begin{bmatrix}0\\1\\0\\0\end{bmatrix} \; d3 = \begin{bmatrix}0\\0\\1\\0\end{bmatrix} \; d4 = \begin{bmatrix}0\\0\\0\\1\end{bmatrix}$$

2. Biểu diễn mỗi bit chẵn lẻ bằng vectơ cột chứa số 1 trong hàng tương ứng với mỗi bit dữ liệu có trong phép tính và số 0 trong tất cả các hàng khác. Sử dụng các định nghĩa bit chẵn lẻ :

$$p_1 = \begin{bmatrix}1\\1\\0\\1\end{bmatrix} \; p_2 = \begin{bmatrix}1\\0\\1\\1\end{bmatrix} \; p_3 = \begin{bmatrix}0\\1\\1\\1\end{bmatrix}$$

3. Tạo ma trận chuyển đổi dữ liệu [G], bằng cách sắp xếp các vectơ cột từ các bước trước đó thành một ma trận 4×7 sao cho các cột được sắp xếp để khớp với các bit tương ứng của chúng trong một từ mã.

- Để tạo một ma trận chuyển đổi dữ liệu tạo ra các từ mã với các bit có thứ tự $p_1, p_2, p_3, d_1, d_2, d_3, d_4$ (với 3 bit chẵn lẻ và theo sau là 4 bit dữ liệu), hãy sử dụng các vectơ từ các bước trước và sắp xếp chúng thành các cột sau

$$\begin{bmatrix} p_1 & p_2 & p_3 & d_1 & d_2 & d_3 & d_4 \end{bmatrix}$$

- Kết quả ta được ma trận chuyển đổi dữ liệu 4×7:

$$\mathbf{G} = \begin{bmatrix} \mathbf{p1} & \mathbf{p2} & \mathbf{d1} & \mathbf{p3} & \mathbf{d2} & \mathbf{d3} & \mathbf{d4} \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{bmatrix}$$

- Việc sắp xếp các cột theo bất kỳ thứ tự nào khác sẽ chỉ thay đổi vị trí của các bit trong

4.4 Ma trận kiểm tra

– Mã Hamming(7,4) sử dụng ma trận kiểm tra để kiểm tra lỗi và sửa lỗi trong dãy mã Hamming.

– Ma trận kiểm tra H của vector dữ liệu d4×1 sẽ có dạng H3×7:

$$\mathbf{H} = \begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix}$$

– Trong đó từng hàng của H tương ứng với p3, p2, p1. Số 1 sẽ được điền vào các vị trí tương ứng với bit chẵn lẽ đó. Ví dụ p3 = d2 + d3 + d4 trong đó ở ô mã 7 bit, ta thấy vị trí của p3, d2, d3, d4 sẽ lần lượt là 4,5,6,7. Vậy ta được vector hàng p3 = $\begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}$

Tóm tắt phép toán số học modulo 2

– Là một phép toán trong đó các số được chia cho 2 và kết quả là phần dư sau khi chia. Trong modulo 2, chỉ có hai giá trị có thể có: 0 và 1. Khi một số chia cho 2 nếu phần dư là 0, thì số đó được coi là chẵn; nếu phần dư là 1, thì số đó được coi là lẻ.

– Phép toán số học modulo 2 thường được sử dụng trong lĩnh vực của mã hóa và toán học máy tính, đặc biệt là trong các thuật toán và phương pháp liên quan đến việc xử lý và lưu trữ dữ liệu số học. Trong lĩnh vực này, phép toán số học modulo 2 thường

được sử dụng để thực hiện các phép toán logic, kiểm tra tính đối xứng của dữ liệu, hoặc trong việc kiểm tra lỗi.

Các phép toán trong module 2

- **Phép cộng:**

  $0 + 0 = 0$

  $1 + 1 = 0$

  $0 + 1 = 1$

  $1 + 0 = 1$

- **Phép nhân:**

  $0 \times 0 = 0$

  $1 \times 0 = 0$

  $0 \times 1 = 0$

  $1 \times 1 = 1$

  Ví dụ:

  $$\begin{bmatrix} 1 & 0 \end{bmatrix} \times \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \times 1 + 0 \times 1 \end{bmatrix} = \begin{bmatrix} 1 + 0 \end{bmatrix} = \begin{bmatrix} 1 \end{bmatrix}$$

  $$\begin{bmatrix} 1 & 1 \end{bmatrix} = \begin{bmatrix} 1 \times 1 + 1 \times 1 \end{bmatrix} = \begin{bmatrix} 1 + 1 \end{bmatrix} = \begin{bmatrix} 0 \end{bmatrix}$$

4.5 Sử dụng ma trận để tạo và giải mã Hamming(7,4)

– Để truyền gửi dữ liệu, ta nhóm các bit dữ liệu mà mình muốn gửi thành một vectơ. Lấy ví dụ, nếu dữ liệu là "1011" thì vectơ của nó là:

$$\mathrm{p} = \begin{bmatrix} 1 & 0 & 1 & 1 \end{bmatrix}$$

– Giả sử ta muốn truyền gửi dữ liệu trên, ta nhân G và P thu được r, r chính là dữ liệu được mã hóa Hamming(7,4). Sau khi phép nhân hoàn thành, khác với cách giải thích ở phần trước (các bit chẵn lẻ nằm ở vị trí 2k), trật tự của các bit trong từ mã ở đây khác với cách bố trí đã nói (các bit dữ liệu nằm ở trên, các bit kiểm chẵn lẻ nằm ở dưới).

$$r = P.G = \begin{bmatrix} 1 & 0 & 1 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 1 & 1 \end{bmatrix}$$

- Khi muốn thu dữ liệu, ta sẽ nhận H với nghịch đảo của r để kiểm tra có lỗi hay không:

$$H.r' = \begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

  - Do phép nhân H và r' thu được một vectơ toàn số không nên máy có thể kết luận là không có lỗi xảy ra.

- Ví dụ trong trường hợp khác, ta giả sử là lỗi một bit đã xảy ra. Phép nhân H.r' cho chúng ta kết quả $\begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix}$ tương đương với cột thứ 3("011" tương đương với giá trị 3 trong số thập phân), do đó chúng ta biết rằng lỗi đã xảy ra ở vị trí thứ 3 trong hàng dữ liệu, khi đó chúng ta có thể sửa lỗi đó.

4.6 Sử dụng mã Haming bổ sung để kiểm tra trường hợp hai lỗi

- Mã Hamming là mã có thể phát hiện và sửa một lỗi đơn lẻ, tuy nhiên nó không thể phân biệt lỗi bit kép của một số từ mã với lỗi bit đơn của một từ mã khác. Do đó, một số lỗi bit kép sẽ được giải mã không chính xác do bị nhầm lẫn đó là lỗi bit đơn mà không bị phát hiện, trừ khi chúng ta không cố gắng tìm ra và sửa nó. Để khắc phục tình trạng nhầm lẫn này, mã Hamming có thể được mở rộng thêm một bit kiểm tra chẵn lẻ ở vị trí thứ 8 để kiểm tra trường hợp hai lỗi

  Ví dụ: Kiểm tra của đoạn bit: 1001010

  - Ta bổ sung 1 bit chẵn lẻ ở vị trí cuối cùng để nó có khả năng phát hiện hai lỗi trên 7 bit còn lại. Do 1001010 có 3 số 1 là số lẻ nên bit chẵn lẻ ở vị trí thứ 8 là 1, vì vậy ta được: 1001010 => 10010101

Vị trí	1	2	3	4	5	6	7	8
Mã nhận	1	0	0	1	0	1	0	1
Kiểm tra tính chẵn lẻ ở vị trí 1-3-5-7	1		0		0	1	0	$c_1 = 1$
Kiểm tra tính chẵn lẻ ở vị trí 2-3-6-7		0	0			1	0	$c_2 = 1$
Kiểm tra tính chẵn lẻ ở vị trí 4-5-6-7				1	0	1	0	$c_3 = 0$

*** Giải thích:**

- Sau khi nhận được đoạn mã 8 bit, ta bắt đầu kiểm tra tính chẵn lẻ ở vị trí 1-3-5-7 tương ứng 1000, do có 1 số 1 là số lẻ nên c1 = 1
- Tiếp tục kiểm tra tính chẵn lẻ ở vị trí 2-3-6-7 tương ứng 0010, do có 1 số 1 là số lẻ nên c2 = 1
- Tương tự, ta kiểm tra tính chẵn lẻ ở vị trí 4-5-6-7 tương ứng 1010, do có số số 1 là số chẵn là 2 nên c3 = 0

❑ Kiểm tra tính chẵn lẻ của toàn đoạn mã 10010101: 0 (do có số số 1 là 4 là số chẵn) => mã không có lỗi hoặc có hai lỗi

❑ C = 011 = 3 ≠ 0

→ Kết luận: Đoạn bit 1001010 là do lỗi hai bit

4.7 Các loại mã Hamming khác

- Với m bit chẵn lẽ, bao gồm từ 1 đến 2m- 1. Sau khi loại bỏ các bit chẵn lẽ, 2m- m -1 bit còn lại sẽ được sử dụng làm dữ liệu. với biến m, chúng ta có được những hamming sau:

Bit chẵn lẽ	Tổng số bit	Bit dữ liệu	Tên	Tốc độ
2	3	1	Hamming(3,1)	$\frac{1}{3} = 0.333$
3	7	4	Hamming(7,4)	$\frac{4}{7} = 0.571$
4	15	11	Hamming(15,11)	$\frac{11}{15} = 0.733$
5	31	26	Hamming(31,26)	$\frac{26}{31} = 0.839$
6	63	57	Hamming(63,57)	$\frac{57}{63} = 0.905$
7	127	120	Hamming(127,120)	$\frac{120}{127} = 0.945$
8	255	247	Hamming(255,247)	$\frac{247}{255} = 0.969$
9	511	502	Hamming(511,502)	$\frac{502}{511} = 0.982$
...	...	...	...	...
m	$n = 2^m - 1$	$k = 2^m - m - 1$	Hamming($2^m - 1$,$2^m - m - 1$)	$\frac{2^m - m - 1}{2^m - 1}$

5 Dùng ngôn ngữ lập trình Python để hiện thực sản phẩm về Hamming Code

5.1 Giới thiệu về Python

Python là một ngôn ngữ lập trình tạo ra bởi Guido van Rossum và được phát hành năm 1991

Python là một ngôn ngữ mạnh và được ứng dụng rộng rãi:

- Phát triển web/app (đặc biệt trong xử lý ngoại biên)
- Công nghệ phần mềm
- Khoa học dữ liệu và Trí tuệ nhân tạo
- Toán học và xử lý số liệu
- Thiết kế và quản lý hệ thống, hệ điều hành

Cú pháp của Python so với các ngôn ngữ khác:

- Python là ngôn ngữ bậc cao được thiết kế gần với ngôn ngữ tự nhiên giúp dễ dàng hiện thực các thuật toán
- Python sử dụng các dòng mới để hoàn thành một lệnh thay vì dùng dấu ';' hoặc '()'
- Python giúp cho lập trình viên có thể viết code ngắn hơn các ngôn ngữ khác

Làm sao để sử dụng Python

Thông thường, một vài loại PC và Mac đã có cài đặt sẵn Python, hoặc có thể tải tại đây. Để kiểm tra xem Python đã được cài đặt trên Window PC hay chưa, bạn có thể tìm kiếm trong Start hoặc gõ lệnh trong Command Line (cmd.exe):

```
python --version
```

Hàm trong Python dùng để định nghĩa một chức năng bằng một khối lệnh được biểu diễn như sau:

```
def <Ten_ham>(<Danh sach cac tham so>):
    Khoi lenh
```

Vòng lặp for trong Python cho phép thực hiện một khối mã nhiều lần

```
for <bien> in <danh sach hoac day so>:
    Khoi lenh
```

Ngoài ra Python còn hỗ trợ các kiểu dữ liệu mạnh (list, dictionary, tuple, int, float, string ...) và các câu lệnh hỗ trợ như (format(), chr(), append(), join(), transpose(), ...) và còn nhiều hơn nữa các hàm, các lệnh xử lý rất mạnh của Python (tại đây)

5.1.1 Thư viện NumPy xử lý toán học

NumPy là một thư viện toán học và mảng số trong Python (Numerical Python). NumPy hỗ trợ các phương thức, hàm hỗ trợ trong Đại số tuyến tính, biến đổi Fourier, và ma trận. Thư viện NumPy là một dự án mã nguồn mở được tạo ra năm 2005 bởi Travis Oliphant và bạn có thể dùng nó miễn phí.

Để cài đặt NumPy, bạn có thể sử dụng lệnh pip của Python trong Command Line (đối với Windows) và terminal (Đối với Macs và các hệ điều hành Unix):

```
pip install numpy
```

Sau khi cài đặt xong thư viện, bạn có thể đưa thư viện vào chương trình bằng từ khoá import:

```
import numpy
```

Có thể sử dụng numpy bằng tên thay thế np

```
import numpy as np
```

Ta tạo mảng số 1, 2 chiều hoặc nhiều chiều bằng NumPy:

```
# Python program for create arrays
import numpy as np

# Creating a array 1D
arr = np.array([1, 2, 3])
print("Array 1D: \n",arr)

# Creating a array 2D (Matrix)
```

```
arr = np.array([[1, 2, 3],
                [4, 5, 6]])
print("Array 2D: \n",arr)

# Creating an array from tuple
arr = np.array((1, 3, 2))
print("\nArray created using tuple:\n", arr)
```

Kết quả:

```
Array 1D:
 [1 2 3]
Array 2D:
 [[1 2 3]
 [4 5 6]]

Array created using tuple:
 [1 3 2]
```

Tìm hiểu thêm về NumPy tại đây

NumPy là một thư viện mạnh ứng dụng rộng rãi trong các lĩnh vực đặc biệt của Toán Ứng Dụng và Toán - tin như Khoa học Dữ liệu (Data Science), Học Máy (Machine Learning), Học Sâu (Deep Learning), Trí tuệ nhân tạo (Artificial Intelligence), Phân tích dữ liệu (Data Analysist), Thị giác máy tính (Computer Vision), ...

5.1.2 Thư viện Tkinter biểu diễn đồ hoạ

Tkinter được định nghĩa là Giao diện đồ hoạ người dùng trong Python (Python GUI), nó cung cấp các bộ công cụ (tools) và các thành phần đồ hoạ (widgets) để tạo ra ứng dụng máy tính với giao diện đồ hoạ. Thư viện này được cài đặt sẵn cùng với Python, nhằm giúp lập trình viên dễ dàng xây dựng đồ hoạ mà không cần thư viện bổ sung.

Những ứng dụng của Tkinter trong Python:

- **Tạo của sổ và hộp thoại:** Tkinter có thể được dùng để tạo của sổ hộp thoại. Thông qua đó có thể hiển thị thông tin, thu thập đầu vào hay trình bày tuỳ chọn cho người dùng

- **Xây dựng giao diện người dùng (GUI) cho máy tính:** Tkinter có thể tạo giao diện để dễ dàng tương tác bao gồm menu, các nút, ...
- Tạo các thành phần đồ hoạ (widget) tuỳ chỉnh: Tkinter có tích hợp các widget đa dạng, tuy nhiên bạn có thể tự định nghĩa các widget của riêng mình

Đây là một ví dụ để tạo một chương trình đồ hoạ:

```
from tkinter import *
root = Tk()
root.title("Hello World")
root.geometry('350x200')
root.mainloop()
```

Kết quả chương trình

Hình 2: Chương trình đồ hoạ Hello World

Tóm lại, Tkinter là một công cụ hữu ích để tạo nhiều giao diện người dùng đồ họa, bao gồm cửa sổ, hộp thoại và tiện ích tùy chỉnh. Nó đặc biệt phù hợp để xây dựng các ứng dụng máy tính để bàn và thêm GUI vào các chương trình dòng lệnh.

5.1.3 Module trong Python

Module có thể coi như là một thư viện mà ta tự định nghĩa, nó chưa các hàm mà ta muốn đưa vào ứng dụng của mình

Cách tạo module đơn giản là định nghĩa các hàm sau đó lưu tệp bằng đuôi .py

Ví dụ chúng ta tạo 1 file có tên mymodule.py:

```
def sayHi(name):
    print("Hello " + name)
```

Sau đó ở chương trình chính, ta gọi:

```
import mymodule
mymodule.sayHi("thay Hiep")
```

hoặc

```
from module import sayHi
sayHi("thay Hiep")
```

Như vậy, chương trình sẽ cho ra kết quả:

```
Hello thay Hiep
```

5.2 Hiện thực sản phẩm truyền và kiểm tra dữ liệu bằng Hamming Code

Sản phẩm truyền dữ liệu (ký tự) và kiểm tra được hiện thực bằng ngôn ngữ lập trình Python và bằng các 'nguyên liệu' đã giới thiệu ở phần 5.1.

Chương trình chính, các module và hình ảnh sẽ được chứa trong thư mục HammingCode, bao gồm:

- ***binary_exchange.py*** Định nghĩa các hàm chuyển dữ liệu từ ký tự về dạng nhị phân (dựa theo bảng mã ASCII)
- ***encode.py*** Chứa các hàm định nghĩa phép nhân ma trận trong không gian nhị phân, hàm truyền dữ liệu và hàm gây nhiễu
- ***decode.py*** Sẽ chauws các hàm thực hiện việc kiểm tra, sửa lỗi và chuyển dữ liệu từ dạng bit (nhị phân) về dạng ký tự
- ***Matrix_private.py*** chứa các ma trận tạo mã và kiểm tra như đã định nghĩa trong phần Cơ sở lý thuyết
- ***main.py*** Kết nối các hàm trong các module và hiện thực các thuật toán đồ hoạ để hoàn thành sản phẩm
- ***logo.png*** là Logo Trường Đại học Bách Khoa Thành phố Hồ Chí Minh dùng trong quá trình thiết lập giao diện

5.2.1 Định nghĩa ma trận tạo mã và ma trận kiểm tra

Như đã giới thiệu, ta định nghĩa trong module *Matrix_private.py* như sau:

```
import numpy as np
encode_matrix = np.array([[1, 1, 1, 0, 0, 0, 0],
                          [1, 0, 0, 1, 1, 0, 0],
                          [0, 1, 0, 1, 0, 1, 0],
                          [1, 1, 0, 1, 0, 0, 1]])
parity_matrix = np.array([[0, 0, 0, 1, 1, 1, 1],
                          [0, 1, 1, 0, 0, 1, 1],
                          [1, 0, 1, 0, 1, 0, 1]])
```

Để chuyển ký tự về dạng nhị phân, ta dùng:

- ***ord(char)*** Chuyển đổi một ký tự char thành mã Unicode tương ứng. Bên cạnh đó ta sử dụng lệnh ***format(<value>, '08b')*** để chuyển đổi số nguyên 'value' thành chuỗi nhị phân với độ dài là 8
- Để tách mã 8 bit thành các đoạn 4 bit, ta sẽ lưu mã 8 bit vào 1 danh sách có số phần tử là số các ký tự nhập vào, sau đó 'bẻ đôi' dãy 8 bit thành 2 dãy 4 bit và thêm vào 1 ma trận $(2n) \times 4$ bằng lệnh ***append()***

Trong file ***binary_ exchange.py***

```
def char_to_binary(char):
    return format(ord(char), '08b')

def bit_to_char(bits):
    return chr(int(bits, 2))

def convert_data_to_binary(data):
    matrix = []
    for char in data:
        binary = char_to_binary(char)
        first_half = binary[:4]
        second_half = binary[4:]
```

```
            first_half_int = [int(bit) for bit in first_half]
            second_half_int = [int(bit) for bit in second_half]
            matrix.append(first_half_int)
            matrix.append(second_half_int)
        return matrix
```

Sau khi chuyển dữ liệu về dạng bit (nhị phân), ta thực hiện bước truyền dữ liệu bằng ma trận tạo mã bằng cách lấy ma trận dữ liệu $(2n) \times 4$ nhân với ma trận tạo mã 4×7 sẽ tạo ra được ma trận truyền đi kích thước $(2n) \times 7$. Vì vậy, thuật toán chính ở đây là định nghĩa phép nhân trong không gian nhị phân, và hàm truyền dữ liệu sẽ gọi phép nhân ta định nghĩa.

Như giới thiệu trong cơ sở lý thuyết, ta định nghĩa phép nhân và phương thức truyền mã trong file ***encode.py***, ta dùng kỹ thuật 'throw exception' để bao quát các trường hợp khi cần tái sử dụng code. Hiện thực chương trình như sau:

```
import numpy as np

def multiply_matrices_binary(matrix1, matrix2):
    try:
        mat1 = np.array(matrix1)
        mat2 = np.array(matrix2)
        if mat1.shape[1] != mat2.shape[0]:
            raise ValueError("No match size.")
        mat2 = np.transpose(mat2)
        temp_row = []
        for m1 in mat1:
            temp_col = []
            for m2 in mat2:
                temp = np.dot(m1, m2)
                temp = temp % 2
                temp_col.append(int(temp))
            temp_row.append(temp_col)
        result = temp_row
        return result
    except ValueError as e:
```

```
            print(e)
            return None

    def transmission_data(data, matrix_trans):
        return multiply_matrices_binary(data, matrix_trans)
```

Bên cạnh đó, file ***encode.py*** sẽ chứa hàm gây nhiễu dữ liệu bằng cách thay đổi 1 bit trong 1 hàng ma trận truyền đi. Dữ liệu đầu vào là 1 chuỗi bit. Như vậy, mỗi ký tự sẽ bị sai 2 bit, dữ liệu đã bị gây nhiễu nặng nề.

```
    def create_jam(bit):
        for d in bit:
            ran_num = random.randint(0, len(d) - 1)
            d[ran_num] = 1 - d[ran_num]
```

Tiếp theo, chúng ta thiết kế các hàm để kiểm tra, sửa lỗi và chuyển về kiểu nhị phân trong file ***decode.py***

Như các thuật toán kiểm tra trong Cơ sở lý thuyết, ta thiết kế các hàm như sau:

- Hàm checking_data(data_check, matrix_check) Với đầu vào là ma trận truyền đi và ma trận kiểm tra. Kết quả trả về của hàm là mảng 1 chiều có $2n$ phần tử, mỗi phần tử là vị trí lỗi trong mỗi hàng của dữ liệu truyền đi, nếu không có thì sẽ là -1
- Hàm error_correction(data_error, pos) Với đầu vào là ma trận chứa lỗi và mảng chứa vị trí lỗi trong mỗi hàng của ma trận
- Hàm decode_list(trans) có đầu vào là ma trận đã được sửa lỗi và chuyển về 1 chuỗi bit có độ dài 8n với n là số ký tự đầu vào đã thu thập. Sau đó, chuỗi bit sẽ trả về các ký tự bằng hàm bit_to_char() trong file encode.py

Lưu ý: trước khi hiện thực các hàm, ta gọi các hàm cần thiết đã được định nghĩa từ các module trước

```
    import numpy as np
    from encode import transmission_data
    from binary_exchange import bit_to_char
```

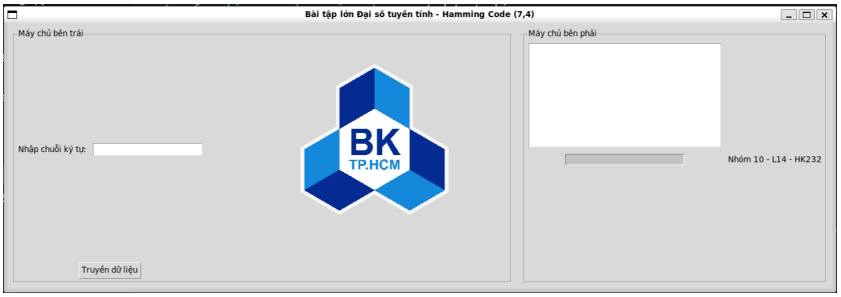
```
def checking_data(data_check, matrix_check):
    res = []
    has_false_bit = False
    check = transmission_data(matrix_check,\
                        np.transpose(data_check))
    check = np.transpose(check)
    for need_check in check:
        for i in range(matrix_check.shape[1]):
            if (need_check == matrix_check[:,i]).all():
                has_false_bit = True
                break
        if has_false_bit:
            res.append(i)
        else:
            res.append(-1)
    return res

def error_correction(data_error, pos):
    for i in range(len(pos)):
        data_error[i][pos[i]] = 1 - \
                        data_error[i][pos[i]]

def decode_list(trans):
    encoder = [[row[2], row[4], row[5], row[6]]\
                                for row in trans]
    bit_char = ''.join([''.join(map(str, lst)) \
                                for lst in encoder])
    bit_char = [bit_char[i:i+8] for i \
                    in range(0, len(bit_char), 8)]
    char_list = [bit_to_char(byte) \
                    for byte in bit_char]
    return ''.join(char_list)
```

Sau khi đã định nghĩa xong hết các hàm hiện thực thuật toán, ta sẽ hiện thực chương trình chính trong ***main.py***. Tuy nhiên, ở đây chúng ta sẽ không tập trung vào phần đồ hoạ mà chỉ tập trung vào hàm xử lý chính ***process_ data()***. Trong hàm này có sử dụng các phương thức xử lý xâu và các lệnh messagebox, errorbox trong tkinter để thông báo về các quá trình của chương trình đang diễn ra, trong báo cáo này sẽ thay bằng lệnh print() và in ra result là thông tin chương trình. Sau khi bấm vào nút (button) 'Truyền dữ liệu', hàm ***process_ data()*** sẽ được gọi và được hiện thực như sau:

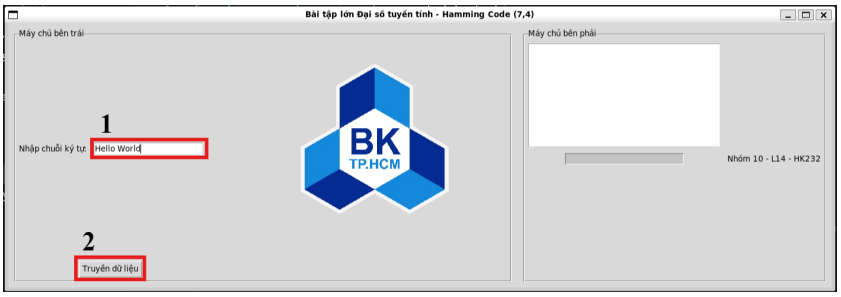
```
def process_data():
    data = data_var.get()     # Get data from user
    # Convert data to binary
    binary_data = convert_data_to_binary(data)

    result = ''.join([''.join(map(str, lst)) \
                                for lst in binary_data])
    result = result.ljust((len(result) + 7) // 8 * 8, '0')
    result = '\n'.join(result[i:i+8] for i \
                            in range(0, len(result), 8))
    print(result)
    # data * G (matrix for transmission)
    trans = transmission_data(binary_data, G)

    result = '\n'.join([' '.join(map(str, lst))\
                                        for lst in trans])
    print(result)

    create_jam(trans) # Cause interference

    # Progress bar when transmitting data
    for _ in range(20):
        progress_bar["value"] += 10
        root.update_idletasks()
        time.sleep(0.1)
```

```
    parity = checking_data(trans, H) # Check parity bits
    no_false = all(item == -1 for item in parity)
    if no_false:
        print(result)
    else:
        # Error correction
        print(trans)
        error_correction(trans, parity)
        result = '\n'.join([' '.join(map(str, lst))\
                                        for lst in trans])
        print(result)

    # Decode data
    result_string = decode_list(trans)

    # Show result
    result_text.delete(1.0, tk.END)
    result_text.insert(tk.END, result_string)
```

Như vậy, sản phẩm mô tả quá trình tạo mã, truyền dữ liệu, làm nhiễu, kiểm tra, sữa lỗi bằng Hamming Code đã được hiện thực trên ngôn ngữ lập trình Python. Sau đây là kết quả khi chạy chương trình

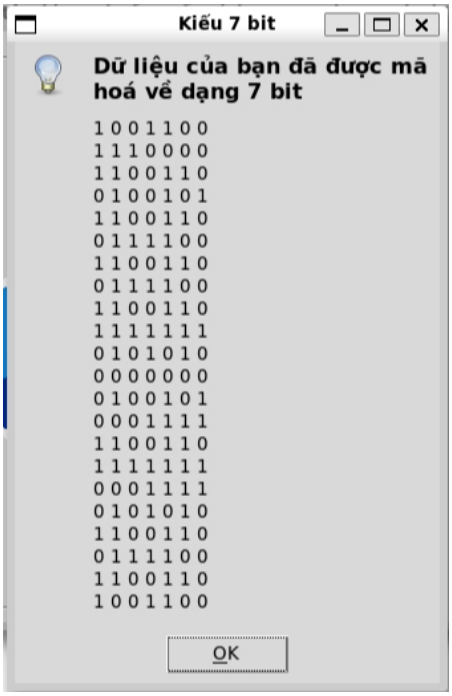


Hình 3: Màn hình khởi động của chương trình

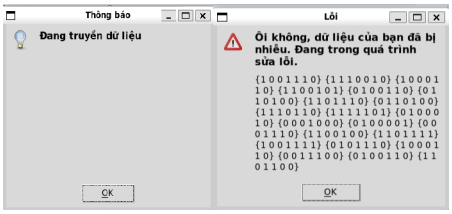


Hình 4: Nhập dữ liệu và truyền đi

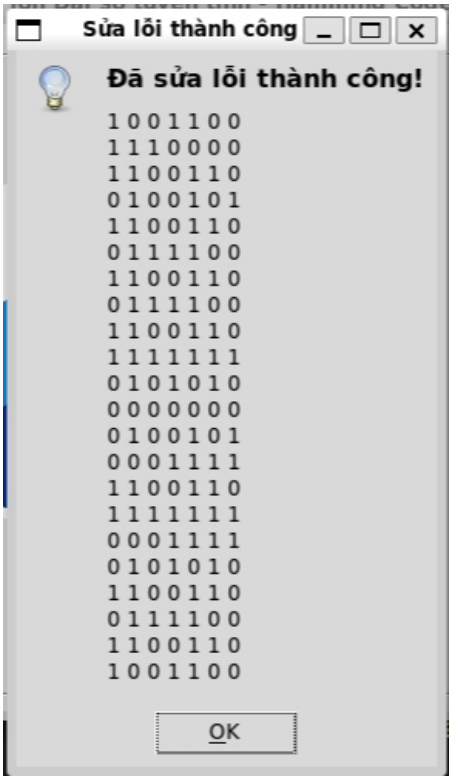
Hình 5: Chuyển dữ liệu ký tự sang bit



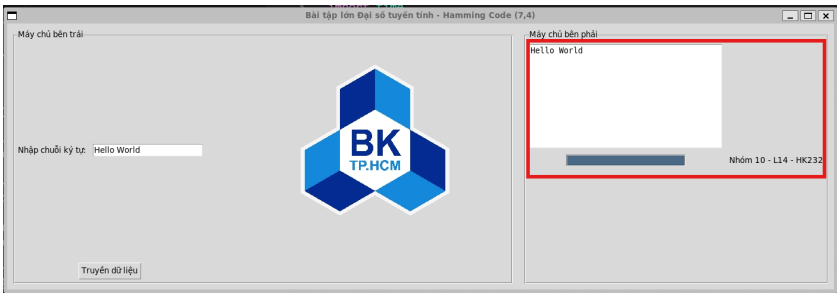
Hình 6: Mã truyền đi



Hình 7: Truyền đi và gây nhiễu



Hình 8: Sửa lỗi dữ liệu bị nhiễu



Hình 9: Truyền dữ liệu thành công, chương trình kết thúc

5.3 Toàn bộ chương trình

Quan sát cách thiết kế và xây dựng chương trình tại đây

Sau khi đã đưa mã nguồn về máy (từ github hoặc nơi khác) để chạy chương trình ta gõ lệnh trong cmd.exe hoặc Terminal:

```
python3 main.py
```

Hamming code là một phương pháp mạnh mẽ để bảo vệ tính toàn vẹn của dữ liệu trong truyền thông số. Triển khai nó trong Python không chỉ giúp hiểu rõ về cách nó hoạt động, mà còn cung cấp cho chúng ta các công cụ để áp dụng nó trong các ứng dụng thực tế.

6 Ứng dụng

1. **Truyền thông và Lưu trữ Dữ liệu:** Mã Hamming được sử dụng rộng rãi trong các ứng dụng truyền thông số như giao thức truyền dẫn, mã hóa kênh, và truyền dữ liệu qua mạng. Nó giúp phát hiện và sửa lỗi trong dữ liệu được truyền qua các kênh có nhiễu.

2. **Bộ nhớ và lưu trữ dữ liệu:** Trong các hệ thống lưu trữ dữ liệu như ổ đĩa cứng, bộ nhớ flash, hoặc RAM, mã Hamming được sử dụng để bảo vệ dữ liệu trước khi ghi vào và đọc từ các thiết bị lưu trữ. Điều này giúp giảm thiểu rủi ro mất dữ liệu do lỗi phần cứng.

3. **Điện tử và Máy tính:** Trong các hệ thống điện tử và máy tính, mã Hamming thường được sử dụng để kiểm tra tính toàn vẹn của dữ liệu, đảm bảo rằng các lỗi trong bộ nhớ hoặc dữ liệu không được phép xảy ra.

4. **Truyền thông không dây:** Trong các hệ thống truyền thông không dây như Wi-Fi, Bluetooth, hoặc các hệ thống di động, mã Hamming có thể được sử dụng để giảm thiểu lỗi trong quá trình truyền dẫn.

7 Kết luận

Qua bài tập lớn đề tài Mã Hamming (7,4) này, chúng ta được biết về mã Hamming, một phương pháp mã hóa thông tin truyền thông và lưu trữ dữ liệu. Mã Hamming giúp chúng ta có sự ổn định trong quá trình truyền tải, giảm thiểu sự sai lệch do sự đảo lộn các bit số nhị phân gây ra. Cụ thể hơn ở đề tài này, chúng ta biết được cách tiến hành mã hóa, kiểm tra và sửa lỗi của mã Hamming 7,4 bằng phương pháp sử dụng ma trận tạo mã và ma trận kiểm tra, cách để thực hiện một chương trình hoàn chỉnh mô tả quá trình kiểm tra và sửa lỗi bằng phương pháp Hamming (7,4). Thêm vào đó, ta được biết về cách mà mã Hamming phát hiện ra lỗi do sự sai lệch của 2 bit dữ liệu gây ra. Qua tìm hiểu và nghiên cứu, mã Hamming (7,4) vẫn còn những bất tiện như không thể xử lý nhiều hơn một lỗi và phát hiện nhiều hơn hai lỗi, không thích hợp để xử lý các dữ liệu có kích thước lớn. Tóm lại, mặc dù mã Hamming (7,4) vẫn tồn tại những nhược điểm, nó vẫn thể hiện được độ đáng tin cậy trong ứng dụng truyền thông và lưu trữ dữ liệu, và được sử dụng rộng rãi trong nhiều ứng dụng yêu cầu độ chính xác và tính toàn vẹn dữ liệu cao.

Tài liệu tham khảo

[1] Raymond Hill, (1993). "A First Course in Coding Theory", Clarendon Press, Oxrord, USA.

[2] Yehuda Lindell, “Introduction to Coding Theory”, Lecture Notes, Department of Computer Science Bar-Ilan University , Israel (2010).

[3] Tom Richardson, Rudiger Urbanke. “Modern Coding Theory”. Cambridge University Press (2008)

[4] Đặng Văn Chuyết, Nguyễn Tuấn Anh.(1998)“Giáo trình Cơ sở lý thuyết truyền tin”. NXB Giáo dục.

[5] "Thử thách toán học tháng 4 năm 2013 Mã sửa lỗi" (PDF), đội ngũ lãnh đạo tập đoàn swissQuant, tháng 4 năm 2013, đã lưu trữ (PDF) từ bản gốc vào ngày 12 tháng 9 năm 2017.

[6] Hamming, Richard Wesley (1950). "Mã phát hiện lỗi và sửa lỗi" (PDF) . Tạp chí kỹ thuật hệ thống Bell. Được lưu trữ (PDF) từ bản gốc vào ngày 2022-10-09.

[7] Mặt trăng, Todd K. (2005). Mã sửa lỗi . New Jersey : John Wiley và các con trai.