**LỜI MỞ ĐẦU**

Hiện nay, nhu cầu trao đổi, truyền và lưu trữ dữ liệu đóng một vai trò quan trọng trong nhiều lĩnh vực như: Bảo hiểm, ngân hàng, y tế… Việc đảm bảo tính an toàn, gọn nhẹ của dữ liệu luôn là mục tiêu hướng tới của những nhà phát triển và cung cấp dịch vụ công nghệ thông tin. Từ trước tới nay, đã có rất nhiều những phương pháp dùng để mã hóa dữ liệu, mỗi một phương pháp đều đem đến những hiệu quả nhất định.

Được sự hướng dẫn tận tình của thầy Phạm Văn Ất, em đã mạnh dạn tìm hiểu thuật toán **Mã hóa số học** để làm đề tài thực tập tốt nghiệp. Trong quá trình thực hiện đề tài, em đã tìm hiểu thêm được nhiều về mã hóa dữ liệu, tuy nhiên cũng gặp không ít khó khăn vì đây là một lĩnh vực khó và rộng lớn. Do đó em không tránh khỏi những thiếu sót. Em hi vọng các thầy giúp đỡ em để đề tài được hoàn chỉnh hơn. Em xin cám ơn.

Sinh viên

Đinh Văn Hải

**MỤC LỤC**

Tiêu đề Trang

LỜI MỞ ĐẦU 1

1. **GIỚI THIỆU ĐỀ TÀI** 3
2. **Tổng quan về mã hóa dữ liệu** 3
3. **Ứng dụng của mã hóa dữ liệu** 3
4. **Một số phương pháp và thuật toán mã hóa dữ liệu** 4
   1. Mã cổ điển 4
      1. Mã Ceasar 5
      2. Mã Vigenere 5
      3. Phương pháp thám mã Kasiski 5
   2. Thuật toán mã hóa MD5 6
   3. Mã RSA 6
   4. Mã hóa số học 6
5. **PHƯƠNG PHÁP MÃ HÓA SỐ HỌC** 7
6. **Chương 1: Giới thiệu các nguyên lý** 7
   1. Nguyên lý chứa trong 7
   2. Nguyên lý trung gian 7
   3. Nguyên lý chuyển sang chiều khác 7
   4. Nguyên lý rẻ thay cho đắt 8
7. **Chương 2: Áp dụng trong việc nén dữ liệu** 9
   1. Giới thiệu 9
   2. Phương pháp mã hóa số học 12
      1. Tag (thẻ) 12
         1. Phát sinh thẻ 12
         2. Khoảng chứa thẻ 14
         3. Giải mã thẻ 18
      2. Sinh mã nhị phân 20
         1. Bài toán 20
         2. Các ví dụ 20
      3. Mã hóa với số nguyên 25
         1. Nhập đề 25
         2. Thuật toán Encode 25
         3. Thuật toán Decode 29
8. **CÀI ĐẶT THUẬT TOÁN VÀ CHƯƠNG TRÌNH DEMO** 32
9. **Cài đặt các thuật toán** 32
10. **Chương trình demo** 36

TÀI LIỆU THAM KHẢO 39

1. **GIỚI THIỆU ĐỀ TÀI**
2. **Tổng quan về mã hóa dữ liệu:**

**Mã hóa dữ liệu**

Mã hóa dữ liệu (Encryption Data) là một quá trình mà trong đó các dữ liệu rõ được chuyển thành bản mã để nó không thể đọc được. Tổng quát hơn được gọi là sự “mã hóa”, quá trình này có thể được thực hiện bằng nhiều cách khác nhau, và với mức độ thành công khác nhau. Một số phương pháp mã hóa dữ liệu tốt có thể phải mất vài thế kỷ để giải mã, trong khi một số lại có thể giải mã được chỉ trong giây phút hoặc thậm chí bởi những người chỉ có kỹ năng cơ bản. Trong thời đại kỹ thuật số, con người phụ thuộc rất nhiều vào mã hóa dữ liệu trong những thứ cơ bản thường ngày. Có thể bạn đã nhận hoặc gửi dữ liệu đã được mã hóa vào một thời điểm nào đó trong ngày, ngay cả khi bạn không trực tiếp mã hóa hay giải mã các dữ liệu đó.

**Chuẩn mã hóa dữ liệu**

DES - viết tắt của Data Encryption Standard, hay Tiêu chuẩn Mã hóa Dữ liệu) là một phương pháp [mật mã hóa](http://vi.wikipedia.org/wiki/M%C3%A3_h%C3%B3a) được [FIPS](http://vi.wikipedia.org/w/index.php?title=FIPS&action=edit&redlink=1) (Tiêu chuẩn Xử lý Thông tin Liên bang Hoa Kỳ) chọn làm chuẩn chính thức vào năm [1976](http://vi.wikipedia.org/wiki/1976). Sau đó chuẩn này được sử dụng rộng rãi trên phạm vi thế giới. Ngay từ đầu, thuật toán của nó đã gây ra rất nhiều tranh cãi, do nó bao gồm các thành phần thiết kế mật, [độ dài khóa](http://vi.wikipedia.org/wiki/%C4%90%E1%BB%99_l%E1%BB%9Bn_kh%C3%B3a) tương đối ngắn, và các nghi ngờ về [cửa sau](http://vi.wikipedia.org/w/index.php?title=C%E1%BB%ADa_sau&action=edit&redlink=1) để [Cơ quan An ninh quốc gia Hoa Kỳ](http://vi.wikipedia.org/w/index.php?title=C%C6%A1_quan_An_ninh_qu%E1%BB%91c_gia_Hoa_K%E1%BB%B3&action=edit&redlink=1) (NSA) có thể bẻ khóa. Do đó, DES đã được giới nghiên cứu xem xét rất kỹ lưỡng, việc này đã thúc đẩy hiểu biết hiện đại về [mật mã khối](http://vi.wikipedia.org/w/index.php?title=M%E1%BA%ADt_m%C3%A3_kh%E1%BB%91i&action=edit&redlink=1) (block cipher) và các phương pháp [thám mã](http://vi.wikipedia.org/w/index.php?title=Th%C3%A1m_m%C3%A3&action=edit&redlink=1) tương ứng.

Hiện nay DES được xem là không đủ an toàn cho nhiều ứng dụng. Nguyên nhân chủ yếu là độ dài 56 [bit](http://vi.wikipedia.org/wiki/Bit) của [khóa](http://vi.wikipedia.org/wiki/Kh%C3%B3a_%28m%E1%BA%ADt_m%C3%A3%29) là quá nhỏ. Khóa DES đã từng bị phá trong vòng chưa đầy 24 giờ. Đã có rất nhiều kết quả phân tích cho thấy những điểm yếu về mặt lý thuyết của mã hóa có thể dẫn đến phá khóa, tuy chúng không khả thi trong thực tiễn. Thuật toán được tin tưởng là an toàn trong thực tiễn có dạng [Triple DES](http://vi.wikipedia.org/w/index.php?title=Triple_DES&action=edit&redlink=1) (thực hiện DES ba lần), mặc dù trên lý thuyết phương pháp này vẫn có thể bị phá. Gần đây DES đã được thay thế bằng [AES](http://vi.wikipedia.org/wiki/AES_%28m%C3%A3_h%C3%B3a%29) (Advanced Encryption Standard, hay Tiêu chuẩn Mã hóa Tiên tiến).

1. **Ứng dụng của mã hóa dữ liệu**

Một quan tâm khác của lý thuyết mã hóa là việc thiết kế các mã giúp vào việc [đồng bộ hóa](http://vi.wikipedia.org/w/index.php?title=%C4%90%E1%BB%93ng_b%E1%BB%99_h%C3%B3a&action=edit&redlink=1) (synchronization). Một mã có thể được thiết kế để phát hiện [dịch pha sóng điện](http://vi.wikipedia.org/w/index.php?title=D%E1%BB%8Bch_pha_s%C3%B3ng_%C4%91i%E1%BB%87n&action=edit&redlink=1) (phase shift) và sửa lỗi, đồng thời cho phép nhiều tín hiệu (multiple signals) được truyền gửi trên cùng một kênh. Có một hạng loại của những mã mà chúng ta hằng ngày thường gặp trong các điện thoại di động của mình. Chúng được gọi là mã [Đa truy nhập phân chia theo mã](http://vi.wikipedia.org/w/index.php?title=%C4%90a_truy_nh%E1%BA%ADp_ph%C3%A2n_chia_theo_m%C3%A3&action=edit&redlink=1) (Code Division Multiple Access - viết tắt là CDMA). Chi tiết về chúng thuộc phạm vi bên ngoài cuộc đàm luận ở đây, song nói một cách ngắn gọn, mỗi cái điện thoại được gắn cho một mã ký (codeword) lấy ra từ một hạng đặc biệt (special class) (ngành Đại Số). Trong khi truyền thông, mã ký được dùng để xáo trộn các bit đại biểu cho thông điệp âm thanh (voice message). Tại máy thu, một quy trình giải xáo trộn ("descrambling") được tiến hành để giải mã (decipher) thông điệp. Đặc tính của hạng mã ký này cho phép nhiều người dùng (với các mã khác nhau) sử dụng cùng một kênh truyền rađiô trong cùng một lúc. Máy thu, dùng chu trình giải xáo trộn, chỉ "nghe" thấy những người gọi khác như "nhiễu âm" ở độ thấp mà thôi.

Một hạng mã nổi tiếng khác nữa là mã [Yêu cầu lặp lại tự động](http://vi.wikipedia.org/w/index.php?title=ARQ&action=edit&redlink=1) (Automatic Repeat reQuest - viết tắt là ARQ). Trong hạng phổ quát (general class) này, máy phát cho thêm các bit chẵn lẻ kiểm tra (parity check bits) vào các thông điệp dài hơn bình thường. Máy thu kiểm tra bit chẵn lẻ của thông điệp thu được và nếu nó phát hiện ra một sự bất đồng, nó sẽ yêu cầu máy phát truyền tin lại thông điệp. Hầu hết các [mạng diện rộng](http://vi.wikipedia.org/wiki/M%E1%BA%A1ng_m%C3%A1y_t%C3%ADnh#WAN) (wide area networks) và các giao thức, ngoại trừ những cái hết sức đơn giản, đều sử dụng tái truyền thông ARQ. Những giao thức thông dụng bao gồm [SDLC](http://vi.wikipedia.org/w/index.php?title=SDLC&action=edit&redlink=1) (IBM), [TCP](http://vi.wikipedia.org/wiki/TCP) (Internet), [X.25](http://vi.wikipedia.org/w/index.php?title=X.25&action=edit&redlink=1) (Quốc tế) và nhiều cái khác nữa. Hiện có nhiều nghiên cứu trên phạm vi rộng trong lĩnh vực về đề tài này do khó khăn trong việc so sánh giữa một gói dữ liệu đã bị từ bỏ (rejected packet) với một gói dữ liệu mới. Gói dữ liệu mới đến này là một gói mới hay là một gói cũ được truyền lại? Thường thì các hệ thống đánh số được sử dụng, tuy nhiên trong một số mạng truyền thông, gói dữ liệu có thể có một định danh khác, hoặc vấn đề này được dành lại cho tầng cao hơn (của giao thức) để nó yêu cầu truyền thông lại. [TCP/IP](http://vi.wikipedia.org/wiki/TCP/IP) là một giao thức điển hình sử dụng cả hai kỹ thuật. Trong bối cảnh đã kết nối, TCP/IP dành việc truyền thông lại cho mạng lưới truyền thông, và vì thế nó sử dụng mã hóa ARQ. Trong mạng lưới vô tuyến (wireless network), người ta không sử dụng ARQ. Thay vào đó, người ta dành trách nhiệm này cho chương trình ứng dụng kiểm nghiệm gói dữ liệu và yêu cầu việc truyền thông lại khi cần. Việc này có thể nâng cao lên đến mức độ đòi hỏi người dùng phải ấn nút "nạp lại" ("refresh" button) trên một trình duyệt web. Song, ngay cả cái này hiện nay vẫn ở trong hạng ARQ đang được nghiên cứu; người dùng bắt buộc phải can thiệp.

1. **Một số phương pháp và thuật toán mã hóa dữ liệu:**
   1. **Mã cổ điển**

Mã hoá cổ điển là phương pháp mã hoá đơn giản nhất xuất hiện đầu tiên trong lịch sử ngành mã hoá. Thuật toán đơn giản và dễ hiểu. Những phương pháp mã hoá này là cở sở cho việc nghiên cứu và phát triển thuật toán mã hoá đối xứng được sử dụng ngày nay. Trong mã hoá cổ điển có hai phương pháp nổi bật đó là:

* *Mã hoá thay thế:* là phương pháp mà từng kí tự (nhóm kí tự) trong bản rõ được thay thế bằng một kí tự (một nhóm kí tự) khác để tạo ra bản mã. Bên nhận chỉ cần thay thế ngược lại trên bản mã để có được bản rõ ban đầu.
* *Mã hoá hoán vị:* các kí tự trong bản rõ vẫn được giữ nguyên, chúng chỉ được sắp xếp lại vị trí để tạo ra bản mã. Tức là các kí tự trong bản rõ hoàn toàn không bị thay đổi bằng kí tự khác mà chỉ đảo chỗ của chúng để tạo thành bản mã.
  + 1. **Mã Ceasar**

Đây là mã thế được biết sớm nhất, được sáng tạo bởi Julius Ceasar. Lần đầu tiên được sử dụng trong quân sự. Việc mã hoá được thực hiện đơn giản là thay mỗi chữ trong bản rõ bằng chữ thứ ba tiếp theo trong bảng chữ cái. Ví dụ:

* Meet me after the toga party
* PHHW PH DIWHU WKH WRJD SDUWB

Ở đây thay chữ m bằng chữ đứng thứ 3 sau m là p (m, n, o, p); thay chữ e bằng chữ đứng thứ 3 sau e là h (e, f, g, h).

* + 1. **Mã Vigenere**

Mã thế đa bảng đơn giản nhất là mã Vigenere. Thực chất quá trình mã hoá Vigenere là việc tiếh hành đồng thời dùng nhiều mã Ceasar cùng một lúc trên bản rõ với nhiều khoá khác nhau. Khoá cho mỗi chữ dùng để mã phụ thuộc vào vị trí của chữ đó trong bản rõ và được lấy trong từ khoá theo thứ tự tương ứng.

Giả sử khoá là một chữ có độ dài d được viết dạng K = K1K2…Kd, trong đó Ki nhận giá trị nguyên từ 0 đến 25. Khi đó ta chia bản rõ thành các khối gồm d chữ. Mỗi chữ thứ i trong khối chỉ định dùng bảng chữ thứ i với tịnh tiến là Ki giống như trong mã Ceasar. Trên thực tế khi mã ta có thể sử dụng lần lượt các bảng chữ và lặp lại từ đầu sau d chữ của bản rõ. Vì có nhiều bảng chữ khac nhau, nên cùng một chữ ở các vị trí khác nhau sẽ có các bước nhảy khác nhau, làm cho tần suất các chữ trong bản mã dãn tương đối đều.

Giải mã đơn giản là quá trình làm ngược lại. Nghĩa là dùng bản mã và từ khoá với các bảng chữ tương ứng, nhưng với mỗi chữ sử dụng bước nhảy lui lại về đầu.

* + 1. **Phương pháp thám mã Kasiski**

Phương pháp phát triển bởi Babbage và Kasiski. Ta thấy các chữ như nhau trên bản rõ và cách nhau một khoảng đúng bằng độ dài từ khoá (chu kỳ), thì sẽ được mã bằng cùng một chữ. Như vậy từ độ lặp của các chữ trong bản mã có thể cho phép xác định chu kỳ. Tất nhiên không phải khi nào cũng tìm được độ dài từ khoá. Sau đó tìm các chữ trong từ khoá bằng cách tấn công từng bảng chữ đơn với cùng kỹ thuật dựa trên các bảng tần suất của các bộ chữ như trước.

* 1. **Thuật toán mã hóa MD5**

Trong [mật mã học](http://vi.wikipedia.org/wiki/M%E1%BA%ADt_m%C3%A3_h%E1%BB%8Dc), MD5 (viết tắt của [tiếng Anh](http://vi.wikipedia.org/wiki/Ti%E1%BA%BFng_Anh) Message-Digest algorithm 5, giải thuật Tiêu hóa tin 5) là một [hàm băm mật mã](http://vi.wikipedia.org/w/index.php?title=H%C3%A0m_b%C4%83m_m%E1%BA%ADt_m%C3%A3&action=edit&redlink=1) được sử dụng phổ biến với giá trị băm dài 128-[bit](http://vi.wikipedia.org/wiki/Bit). Là một chuẩn [Internet](http://vi.wikipedia.org/wiki/Internet) ([RFC 1321](http://tools.ietf.org/html/1321)), MD5 đã được dùng trong nhiều ứng dụng bảo mật, và cũng được dùng phổ biến để kiểm tra tính toàn vẹn của [tập tin](http://vi.wikipedia.org/w/index.php?title=T%E1%BA%ADp_tin_m%C3%A1y_t%C3%ADnh&action=edit&redlink=1). Một bảng băm MD5 thường được diễn tả bằng một số [hệ thập lục phân](http://vi.wikipedia.org/wiki/H%E1%BB%87_th%E1%BA%ADp_l%E1%BB%A5c_ph%C3%A2n) 32 ký tự.

MD5 được thiết kế bởi [Ronald Rivest](http://vi.wikipedia.org/w/index.php?title=Ronald_Rivest&action=edit&redlink=1) vào năm [1991](http://vi.wikipedia.org/wiki/1991) để thay thế cho hàm băm trước đó, [MD4](http://vi.wikipedia.org/w/index.php?title=MD4&action=edit&redlink=1). Vào năm [1996](http://vi.wikipedia.org/wiki/1996), người ta phát hiện ra một lỗ hổng trong MD5; trong khi vẫn chưa biết nó có phải là lỗi nghiêm trọng hay không, những chuyên gia mã hóa bắt đầu đề nghị sử dụng những giải thuật khác, như [SHA-1](http://vi.wikipedia.org/w/index.php?title=H%C3%A0m_b%C4%83m_SHA&action=edit&redlink=1) (khi đó cũng bị xem là không an toàn). Trong năm [2004](http://vi.wikipedia.org/wiki/2004), nhiều lỗ hổng hơn bị khám phá khiến cho việc sử dụng giải thuật này cho mục đích bảo mật đang bị đặt nghi vấn.

* 1. **Mã RSA**

RSA là mã công khai được sáng tạo bởi Rivest, Shamir & Adleman ở MIT (Trường Đại học Công nghệ Massachusetts) vào năm 1977. RSA là mã công khai được biết đến nhiều nhất và sử dụng rộng rãi nhất hiện nay. Nó dựa trên các phép toán lũy thừa trong trường hữu hạn các số nguyên theo modulo nguyên tố. Cụ thể, mã hoá hay giải mã là các phép toán luỹ thừa theo modulo số rất lớn. Việc thám mã, tức là tìm khoá riêng khi biết khoá công khai, dựa trên bài toán khó là phân tích một số rất lớn đó ra thừa số nguyên tố. Nếu không có thông tin gì, thì ta phải lần lượt kiểm tra tính chia hết của số đó cho tất cả các số nguyên tố nhỏ hơn căn của nó. Đây là việc làm không khả thi.

Người ta chứng minh được rằng, phép lũy thừa cần O((log n)3) phép toán, nên có thể coi lũy thừa là bài toán dễ. Cần chú ý rằng ở đây ta sử dụng các số rất lớn khoảng 1024 bit, tức là cỡ 10350. Tính an toàn dựa vào độ khó của bài toán phân tích ra thừa số các số lớn. Bài toán phân tích ra thừa số yêu cầu O(e log n log log n) phép toán, đây là bài toán khó.

* 1. **Mã hóa số học**

Là phương pháp mã hóa dữ liệu mà em xin trình bày sau đây:

1. **PHƯƠNG PHÁP MÃ HÓA SỐ HỌC**
2. **Chương 1: Giới thiệu các nguyên lý**
   1. **Nguyên lý chứa trong**

*Nội dung:* Một đối tượng được được đặt bên trong một đối tượng khác và bản thân nó lại chứa đối tượng khác nữa.

Đối tượng B

Đối tượng C

Đối tượng A

*Hình 1:* Nguyên lý chứa trong

* 1. **Nguyên lý trung gian**

*Nội dung:* Để đạt đến mục đích, chúng ta phải đi qua nhiều bước trung gian, kết quả của bước này sẽ là cơ sở để thực hiện bước kế.

*Hình 2:* Nguyên lý trung gian

* 1. **Nguyên lý chuyển sang chiều khác**

*Nội dung:* Mục đích mà ta muốn đạt được lên đối tượng ở một không gian khác dễ thấy hơn. Do vậy sau khi xem xét tổng quát, ta có thể đổi đối tượng sang chiều không gian khác, ở đó bài toán trở nên dễ dàng hơn.

Điểm đặc biệt ở đây là sự chuyển qua lại giữa các không gian là tương đương.

Không gian 1

Không gian 2 (tối ưu)

**Chuyển**

*Hình 3:* Nguyên lý chuyển sang chiều khác

* 1. **Nguyên lý rẻ thay cho đắt**

*Nội dung:* Nếu có nhiều giải pháp đưa đến các kết quả tương đương thì ta nên chọn giải pháp nào tốn ít chi phí nhất.

*Hình 4:* Nguyên lý rẻ thay cho đắt

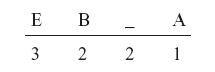
1. **Chương 2: Áp dụng trong việc nén dữ liệu**
   1. **Giới thiệu**

Một trong những kỹ thuật nén mạnh mẽ nhất là kỹ thuật mã hóa số học (Arithmetic coding). Nó chuyển đổi toàn bộ dữ liệu đầu vào thành những số thực chính xác đơn.

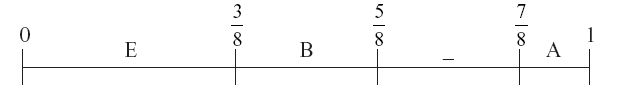
Chúng ta thử xét chuỗi ví dụ như sau: **BE\_A\_BEE** và nén nó bằng phương pháp mã hóa số học.

***Bước 1:* Chuyển đổi các ký tự thành các số thực trong khoảng [0, 1)**

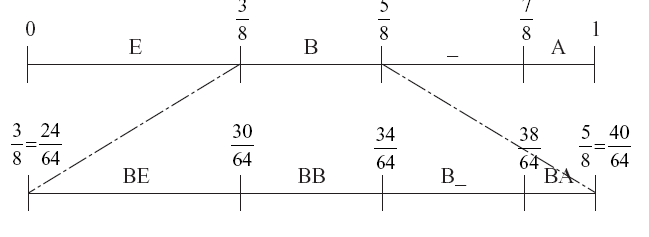
* Đầu tiên chúng ta tìm số lần lặp của các ký tự:



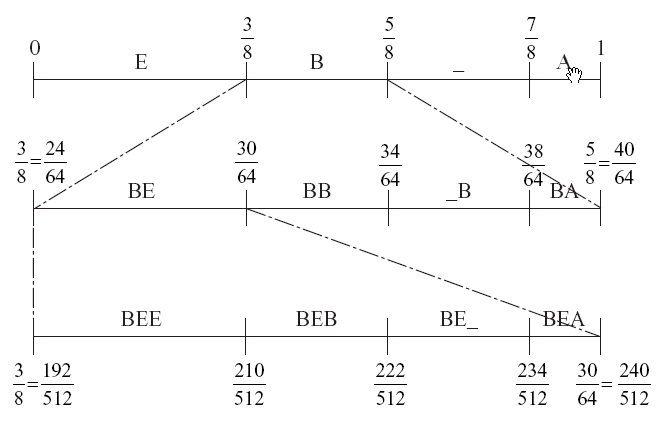
* Chúng ta mã hóa chuỗi bằng cách phân chia khoảng [0, 1) và định vị mỗi ký tự trong một phạm vi phụ thuộc vào số lần xuất hiện của nó trong chuỗi:



* Chúng ta bắt đầu với ‘B’. Chúng ta sẽ lấy khoảng chứa ‘B’ và phân chia theo cùng cách:



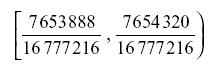
* Tiếp tục như trên với ký tự thứ 2 là ‘E’:



* Cứ thế tiếp tục, cuối cùng ta thu được:



* Chuỗi ‘BE\_A\_BEE’ đã được chuyển thành:



* Tuy nhiên chúng ta không thể gửi những số như **7653888 / 16777216** trong máy tính vì chúng chỉ làm việc với các số nhị phân.

**Kết luận bước 1**: Ở bước này chúng ta sử dụng các nguyên lý sau :

* Nguyên lý chứa trong: Chuỗi ký tự được mã hóa vào khoảng [0, 1) sau đó tùy thuộc thứ tự của các ký tự chúng lại được chuyển vào trong khoảng con của khoảng [0, 1). Điều này làm cho kích thước khoảng chứa ngày càng nhỏ.
* Nguyên lý trung gian: Đây chỉ là bước trung gian, kết quả được sử dụng cho bước sau.
* Nguyên lý chuyển sang chiều khác: Các ký tự được chuyển sang số thực của khoảng [0,1).

***Bước 2*:Đổi sang cơ số nhị phân**

*Nhận xét 1*: Cơ số của hệ đếm

Hệ thập phân:

7653888 = (7x106) + (6x105) + (5x104) + (3x103) + (8x102) + (8x10) + 8

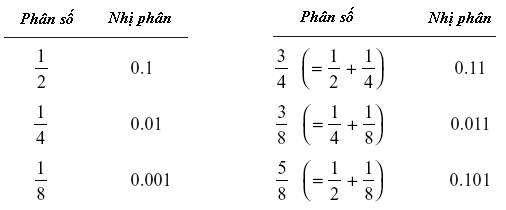
Hệ nhị phân:

110100111 = (1x28) + (1x27) + (2x26) + (1x25) + (0x24) + (0x23) + (1x22) + (1x2) + 1

= 256 + 128 + 32 + 4 + 1

= 432

*Nhận xét 2*: Chúng ta cũng có thể biểu diễn phân số ở dạng nhị phân như sau :



*Nhận xét 3*:



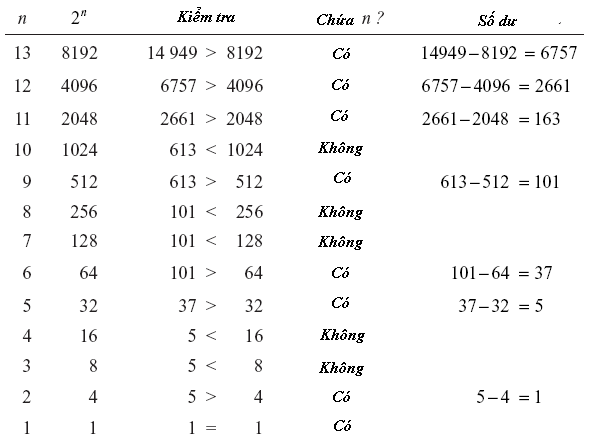


Tổng quát:

 Với dấu chấm thập phân sẽ di chuyển sang trái g vị trí

Chúng ta cũng có thể sử dụng cùng phương pháp trong hệ nhị phân: Chuyển 14949 sang hệ nhị phân, do mẫu số là 32768 = 215 nên chúng ta sẽ chuyển dấu chấm sang trái 15 vị trí.

Vấn đề là làm sao chuyển một số sang hệ nhị phân? Bắt đầu bằng cách tìm một bội số lớn nhất của 2 và nhỏ hơn số cần chuyển, trừ nó, ghi nhận số lũy thừa. Sau đó lặp lại thao tác trên cho phần dữ của phép trừ …



Cuối cùng ta có:



Từ 3 nhận xét trên, ta tính được kết quả như sau:





Trong 2 số nhị phân trên, số đầu có ít ký số hơn số thứ 2, do đó ta chọn:

**BE\_A\_BEE = 011101001100101**

Chỉ gồm 15 ký số (do không cần phải gởi số 0 đầu tiên và dấu chấm thập thân)

Nếu sử dụng mã ASCII phải cần đến 8 x 5 = 40 ký số.

**Kết luận bước 2:** Ở bước này, chúng ta áp dụng các nguyên lý :

* Nguyên lý trung gian.
* Nguyên lý chuyển sang chiều khác: chuyển các số từ hệ thập phân sang nhị phân cho phép sử dụng trên máy tính.
* Nguyên lý rẻ thay cho đắt: Xử lý trên số nhị phân có chi phí nhỏ hơn trên số thập phân (đối với máy tính). Dữ liệu sau khi nén sẽ có kích thước nhỏ hơn do đó ít hao tài nguyên lưu trữ và đường truyền hơn.
  1. **Phương pháp mã hóa số học**
     1. **Tag (thẻ)**
        1. **Phát sinh thẻ**

Thủ tục để phát sinh thẻ thực hiện bởi việc giảm kích thước hiện có của thẻ từ khoảng [ 0, 1 ) thành những khoảng nhỏ hơn.

Ta bắt đầu chia nhỏ khoảng thành nhiều khoảng con

*[ FX( i – 1 ), FX ( i ) ); i = 1, …, m*



Tương tự:



Khi thẻ ak được giới hạn thì thẻ aj là khoảng [ )



Ví dụ :

Cho A = { a1, a2, a3 } với p (a1) = 0.7 , p (a2) = 0.1 , p (a3) = 0.2

Suy ra hàm tích lũy là :

FX (1) =  = p (1) = 0.7

FX (2) = p (1) + p (2) = 0.8

FX (3) = p (1) + p (2) + p (3) = 1

Khi đó: a1  [ FX(0), FX(1) ) ; a 2  [ FX(1), FX(2) ) ; a 3  [FX(2), FX(3) )

**Trường hợp 1** : Giả sử đầu tiên chọn thẻ a1



**Trường hợp 2** : Giả sử đầu tiên chọn thẻ a2



Đặt:

trong đó

n : là thẻ cuối cùng là một số duy nhất đại diện cho dãy thẻ được giới hạn.



: là dãy thẻ được giới hạn.

* + - 1. **Khoảng chứa thẻ**



Thẻ ak được bao bởi [ l(k) , u(k) ]

Ta có :

l (0) = 0

u (0) = 1

l (n) = l (n – 1) + ( u (n – 1) – l (n – 1) ) \* FX (xn – 1)

u (n) = l (n – 1) + ( u (n – 1) – l (n – 1) ) \* FX (xn)

cho dãy x = ( a1 a2 .. an)

để mã hóa dãy x ta lần lượt tính l (k), u (k) theo công thức đệ qui trên.

***Ví dụ 1 :***

Cho A = { a1, a2, a3 } với p(a1) = 0.8; p(a2) = 0.02; p(a3) = 0.18

Hãy sinh thẻ cho dãy x = (1321).

***Bài giải :***

Suy ra các hàm tích lũy: FX(0) = 0; FX(1) = 0.8; FX(2) = 0.82; FX(3) = 1; FX(4) = 1.

l (0) = 0 ; u (0) = 1

Thẻ của 1:

l (1) = 0 + (1 – 0) \* 0 = 0

u (1) = 0 + (1 – 0) \* 0.8 = 0.8

[ 0 , 0.8 ]

Thẻ của 3:

l (2) = 0 + (0.8 – 0) \* 0.82 = 0.656

u (2) = 0 + (0.8 – 0) \* 1 = 0.8

[ 0.656 , 0.8 ]

Thẻ của 2:

l (3) = 0.656 + (0.8 – 0.656) \* 0.8 = 0.7712

u (3) = 0.656 + (0.8 – 0.656) \* 0.82 = 0.77468

[ 0.7712 , 0.77468 ]

Thẻ của 1:

l (4) = 0.7712 + (0.77468 – 0.7712) \* 0 = 0.7712

u (4) = 0.7712 + (0.77468 – 0.7712) \* 0.8 = 0.773504

[ 0.7712 , 0.773504 ]

Dãy thẻ được giới hạn



***Ví dụ 2 :***

Cho A = { C , G , N , T , V }

với p(C) = 0.12 ; p(G) = 0.03 ; p(N) = 0.01 ; p(T) = 0.21 ; p(V) = 0.63

Yêu cầu: sinh thẻ và tìm của dãy CNTTGTVT.



***Bài giải :***

Suy ra các hàm tích lũy :

FX (0) = 0 ; FX (1) = 0.12 ; FX (2) = 0.15 ; FX (3) = 0.16 ; FX (4) = 0.37 ; FX (5) = 1.

l (0) = 0 ; u (0) = 1



Thẻ của C:

l (1) = 0 + ( 1 – 0 ) \* 0 = 0

u (1) = 0 + ( 1 – 0 ) \* 0.12 = 0.12

[ 0 , 0.12 ]



Thẻ của N:

l (2) = 0 + ( 0.12 – 0 ) \* 0.15 = 0.018

u (2) = 0 + ( 0.12 – 0 ) \* 0.16 = 0.0192

[ 0.018 , 0.0192 ]

Thẻ của T :

l (3) = 0.018 + ( 0.0192 – 0.018 ) \* 0.16 = 0.018192

u (3) = 0.018 + ( 0.0192 – 0.018 ) \* 0.37 = 0.018444

[ 0.018192 , 0.018444 ]

Thẻ của T :

l (4) = 0.018192 + ( 0.018444 – 0.018192 ) \* 0.16 = 0.01823232

u (4) = 0.018192 + ( 0.018444 – 0.018192 ) \* 0.37 = 0.01828524

[0.01823232 , 0.01828524 ]

Thẻ của G :

l (5) = 0.01823232 + ( 0.01828524 - 0.01823232 ) \* 0.12 = 0.01823867

u (5) = 0.01823232 + ( 0.01828524 - 0.01823232 ) \* 0.15 = 0.018240258

[0.01823867 , 0.018240258 ]

Thẻ của T :

l (6) = 0.01823867 + ( 0.018240258 – 0.01823867 ) \* 0.16 = 0.018238924

u (6) = 0.01823867 + ( 0.018240258 – 0.01823867 ) \* 0.37 = 0.018239257

[ 0.018238924 , 0.018239257 ]

Thẻ của V :

l (7) = 0.018238924 + (0.018239257 - 0.018238924) \* 0.37 = 0.018239047

u (7) = 0.018238924 + ( 0.018239257 - 0.018238924 ) \* 1 = 0.018239257

[ 0.018239047 , 0.018239257 ]

Thẻ của T :

l (8) = 0.018239047 + ( 0.018239257 - 0.018239047 ) \* 0.16 = 0.01823908

u (8) =0.018239047 + ( 0.018239257 - 0.018239047 ) \* 0.37 = 0.018239124

[ 0.01823908 , 0.018239124 ]

Dãy thẻ được giới hạn:



* + - 1. **Giải mã thẻ**

***Thuật toán giải mã thẻ***

*Bước 1* : Khởi động l(0) = 0 , u(0) = 1

*Bước 2* : Với mỗi k tìm



*Bước 3* : Tìm xk sao cho FX (xk – 1) ≤ t \* < FX (xk)

*Bước 4* : Cập nhật l(k) và u(k)

*Bước 5* : Nếu số lượng các ký hiệu đã giải mã hết thì dừng, ngược lại k := k + 1 và sang bước 2.

***Ví dụ:***

Cho A = { a1, a2, a3 }  { 1, 2, 3 } với p(a1) = 0.8 ; p(a2) = 0.02 ; p(a3) = 0.18 và



Suy ra : FX (1) = 0.8 ; FX (2) = 0.82 ; FX (3) = 1

Hãy giải mã, biết số phần tử của dãy x là |x| = 4

***Bài giải*** :

l(0) = 0 , u(0) = 1

* l(1) = 0 + (1 – 0) \* FX (x1 – 1) = FX (x1 – 1)

u(1) = 0 + (1 – 0) \* FX (x1) = FX (x1)

[ FX (x1 – 1) , FX (x1) )



* Giả sử x1 = 1, suy ra [ FX (x1 – 1) , FX (x1) ) = [ 0 , 0.8 ] ∍
* Giả sử x1 = 2, suy ra [ FX (x1 – 1) , FX (x1) ) = [ 0.8 , 0.82 ]
* Giả sử x1 = 3, suy ra [ FX (x1 – 1) , FX (x1) ) = [ 0.82 , 1 ]

*Vậy chọn* ***x1 = 1***.

* l(2) = 0 + (0.8 – 0) \* FX (x2 – 1) = 0.8 \* FX (x2 – 1)

u(2) = 0 + (0.8 – 0) \* FX (x2) = 0.8 \* FX (x2)

[ 0.8 \* FX (x2 – 1) , 0.8 \* FX (x2) )

* Giả sử x2 = 1, suy ra [ 0.8 \* FX (x2 – 1) , 0.8 \* FX (x2) ) = [ 0 , 0.64 ]
* Giả sử x2 = 2, suy ra [ 0.8 \* FX (x2 – 1) , 0.8 \* FX (x2) ) = [ 0.64 , 0.656 ]



* Giả sử x2 = 3, suy ra [ 0.8 \* FX (x2 – 1) , 0.8 \* FX (x2) ) = [ 0.656 , 0.8 ] ∍

*Vậy chọn* ***x2 = 3***.

* l(3) = 0.656 + (0.8 – 0.656) \* FX (x3 – 1) = 0.656 + 0.144 \* FX (x3 – 1)

u(3) = 0.656 + (0.8 – 0.656) \* FX (x3) = 0.656 + 0.144 \* FX (x3)

[0.656 + 0.144 \* FX (x3 – 1) , 0.656 + 0.144 \* FX (x3) )



* Giả sử x3 = 1, suy ra

[0.656 + 0.144\*FX(x3 – 1), 0.656 + 0.144\*FX(x3)) = [ 0, 0.8 ]

* Giả sử x3 = 2, suy ra



[0.656 + 0.144 \* FX (x3 – 1), 0.656 + 0.144 \* FX (x3) ) = [ 0.8, 0.82 ] ∍

* Giả sử x3 = 3, suy ra

[0.656 + 0.144\*FX(x3 – 1) , 0.656 + 0.144 \* FX(x3)) = [0.82,1]

*Vậy chọn* ***x3 = 2***.

Khi đó l(3) = 0.7712 ; u(3) = 0.77408

* l(4) = 0.7712 + (0.77408 – 0.7712)\*FX(x4 – 1) = 0.7712 + 0.00288\*FX(x4 – 1)

u(4) = 0.7712 + (0.77408 – 0.7712) \* FX (x4) = 0.7712 + 0.00288 \* FX (x4)

[ 0.7712 + 0.00288 \* FX (x4 – 1) , 0.7712 + 0.00288 \* FX (x4) )





* Giả sử x4 = 1, suy ra [ … ) = [ 0, 0.8 ] ∍
* Giả sử x4 = 1, suy ra [ …) = [ 0.8, 0.82 ]
* Giả sử x4 = 1, suy ra [… ) = [ 0.82, 1 ]

*Vậy chọn* ***x4 = 1***.

Vì số phần tử của dãy x là 4 nên ta dừng ở đây.

Kết quả giải mã : x = 1321.

* + 1. **Sinh mã nhị phân**
       1. **Bài toán**
* Ta có :

l(n) = l(n – 1) + ( u(n – 1) – l(n – 1) ) \* FX (xn – 1)

u(n) = l(n – 1) + ( u(n – 1) – l(n – 1) ) \* FX (xn)

Khoảng [ l(n) , u(n) ) so với khoảng [0, 1) có các trường hợp sau :

* E1 : nằm trong nửa thấp của [0, 1) : ∈ [0, 0.5)
* E2 : nằm trong nửa cao của [0, 1) : ∈ [0.5, 1)
* E3 : chứa điểm giữa 0.5
* Giải quyết :

E1 : send bit 0

E1 : [0, 0.5) → [0, 1)

x ↦ E1(x) = 2x

E2 : send bit 1

E2 : [0.5, 1) → [0, 1)

x ↦ E2(x) = 2(x – 0.5)

E3 :

E3 : [0.25, 0.75) → [0, 1)

x ↦ E3(x) = 2(x – 0.25)

* + - 1. **Các ví dụ**

***Ví dụ 1***

Cho A = { a1, a2, a3 } với p(a1) = 0.8 ; p(a2) = 0.02 ; p(a3) = 0.18

Suy ra các hàm tích lũy: FX(0) = 0; FX(1) = 0.8; FX(2) = 0.82; FX(3) = 1; FX(4) = 1.

Hãy mã hóa dãy x = (1321)

***Bài giải*** :

l (0) = 0 ; u (0) = 1

**1 :**

l(1) = 0 + (1 – 0) \* FX(0) = 0

u(1) = 0 + (1 – 0) \* FX(1) = 0.8

Vì [ 0, 0.8 ) ∉ nửa cao hay thấp nên chọn phần tử kế.

**3 :**

l(2) = 0 + (0.8 – 0) \* FX(2) = 0.656

u(2) = 0 + (0.8 – 0) \* FX(3) = 0.8

Vì [ 0.656, 0.8 ) ∈ nửa cao nên send bit **1**

E 2: l(2) = 2 \* (0.656 – 0.5) = 0.312

u(2) = 2 \* (0.8 – 0.5) = 0.6

Vì [ 0.312, 0.6 ) ∉ nửa cao hay thấp nên chọn phần tử kế.

**2 :**

l(3) = 0.312 + (0.6 – 0.312) \* FX(1) = 0.5424

u(3) = = 0.312 + (0.6 – 0.312) \* FX(2) = 0.54816

Vì [ 0.5424, 0.54816 ) ∈ nửa cao nên send bit **1**

E2: l(3) = 2 \* (0.5424 – 0.5) = 0.0848

u(3) = 2 \* (0.54816 – 0.5) = 0.09632

Vì [ 0.0848, 0.09632 ) ∈ nửa thấp nên send bit **0**

E1: l(3) = 2 \* 0.0848 = 0.1696

u(3) = 2 \* 0.09632 = 0.19264

Vì [ 0.1696, 0.19264 ) ∈ nửa thấp nên send bit **0**

E1: l(3) = 2 \* 0.1696 = 0.3392

u(3) = 2 \* 0.19264 = 0.38528

Vì [ 0.3392, 0.38528 ) ∈ nửa thấp nên send bit **0**

E1: l(3) = 2 \* 0.3392 = 0.6784

u(3) = 2 \* 0.38528 = 0.77056

Vì [ 0.6784, 0.77056 ) ∈ nửa cao nên send bit **1**

E2: l(3) = 2 \* ( 0.6784 – 0.5 ) = 0.3568

u(3) = 2 \* ( 0.77056 – 0.5 ) = 0.54112

Vì [ 0.3568, 0.54112 ) ∉ nửa cao hay thấp nên chọn phần tử kế.

**1 :**

l(4) = 0.3568 + ( 0.54112 – 0.3568 ) \* FX (0) = 0.3568

u(4) = 0.3568 + ( 0.54112 – 0.3568 ) \* FX (1) = 0.504256

Khoảng cuối cùng [ 0.3568, 0.504256 )

Được dãy mã hóa nhị phân : 1 1 0 0 0 1

Chọn 1 giá trị thuộc khoảng cuối, ví dụ : 0.5

Vậy send bit 1

Kết quả encode : 1 1 0 0 0 1 1 0 0 … 0

***Ví dụ 2***

Ta có encode : 1 1 0 0 0 1 1 0 0 … 0

Hãy giải mã (decode)

***Bài giải :***

Khoảng nhỏ nhất của đề cho : [ 0.8, 0.82 ), size = 0.02

Vậy cần 2-k ≤ 0.02. Suy ra k = 6.

Do đó ta lấy 6 bit đầu của encode : 1 1 0 0 0 1 ( = 0.765625 = x )

l(0) = 0

u(0) = 1



Vì x’ ∈ [ 0, 0.8 ) nên chọn bit **1**

l(1) = 0 + ( 1 – 0 ) \* FX (0) = 0

u(1) = 0 + ( 1 – 0 ) \* FX (1) = 0.8

Vì [ 0, 0.8 ) ∉ nửa cao hay thấp nên chọn phần tử kế.



Vậy chọn **3**

l(2) = 0 + ( 0.8 – 0 ) \* FX (2) = 0.656

u(2) = 0 + ( 0.8 – 0 ) \* FX (3) = 0.8

Vì [ 0.656, 0.8 ) ∈ nửa cao nên bỏ bit đầu và lấy 6 bit tiếp theo: 100011 (=0.546875)

Rescale: l(2) = 2 \* ( 0.656 – 0.5 ) = 0.312

u(2) = 2 \* ( 0.8 – 0.5 ) = 0.6

Vì [ 0.312, 0.6 ) ∉ nửa cao hay thấp nên giải mã phần tử kế.



Giải được **2**

l(3) = 0.312 + ( 0.6 – 0.312 ) \* FX (1) = 0.5424

u(3) = 0.312 + ( 0.6 – 0.312 ) \* FX (2) = 0.54816

Vì [ 0.5424, 0.54816 ) ∈ nửa cao nên bỏ bit đầu và lấy 6 bit tiếp theo: 000110 (= 0.09375)

Rescale: l(3) = 2 \* ( 0.5424 – 0.5 ) = 0.0848

u(3) = 2 \* ( 0.54816 – 0.5 ) = 0.09632

Vì [0.0848, 0.09632) ∈ nửa thấp nên bỏ bit đầu và lấy 6 bit tiếp theo : 001100

Rescale: l(3) = 2 \* 0.0848 = 0.1696

u(3) = 2 \* 0.09632 = 0.1964

Vì [ 0.1696, 0.1964 ) ∈ nửa thấp nên bỏ bit đầu và lấy 6 bit tiếp theo : 0 1 1 0 0 0

Rescale: l(3) = 2 \* 0.1696 = 0.3392

u(3) = 2 \* 0.1964 = 0.38528

Vì [ 0.3392, 0.38528 ) ∈ nửa thấp nên bỏ bit đầu và lấy 6 bit tiếp theo : 1 1 0 0 0 0

Rescale: l(3) = 2 \* 0.3392 = 0.6784

u(3) = 2 \* 0.38528 = 0.77056

Vì [ 0.6784, 0.77056 ) ∈ nửa cao nên bỏ bit đầu và lấy 6 bit tiếp theo: 100000 ( x = 0.5 )

Rescale: l(3) = 2 \* ( 0.6784 – 0.5 ) = 0.3568

u(3) = 2 \* ( 0.77056 – 0.5 ) = 0.54112

Vì [ 0.3568, 0.54112 ) ∉ nửa cao hay thấp nên giải mã phần tử kế.



Vậy chọn bit **1**

Dừng

Kết quả: decode: 1321 cần 7 bit

* + 1. **Mã hóa với số nguyên**
       1. **Nhập đề**
* Gọi TotalCount là số lượng các phần tử dữ liệu
* Gọi nj là số phần tử j ∈ dữ liệu
* Hàm tích lũy : Cum - Count (K) = 
* 

Khi đó :





* Nhận xét :
* Bit đầu của l(n) và u(n) gọi là bit đáng giá nhất MSB (Most Significant Bit)
* Nếu MSB của l(n) và u(n) = 1 thì [ l(n), u(n) ) ∈ nửa cao : **E2**
* Nếu MSB của l(n) và u(n) = 0 thì [ l(n), u(n) ) ∈ nửa thấp : **E1**
* **E3** : MSB của l(n) và u(n) khác nhau và bit thứ hai khác.
  + - 1. **Thuật toán Encode**

***Thuật toán***

*Bước 1*: Khởi tạo l = 0 ; u = 1

*Bước 2*: Đọc phần tử kế của dãy x

*Bước 3*:





// gọi b là một bit { 0 , 1 }

while ( MSB của l & u = b hay E3 ){

if ( MSB của l & u = b ){

send b

Shift trái l một bit & Shift 0 vào l ;

Shift trái u một bit & Shift 1 vào u ;

while ( scaleE3 > 0 ) {

send bit bù của b

scaleE3 = scaleE3 – 1 ;

}

}

if ( E3 ){

Shift trái l một bit & Shift 0 vào l ;

Shift trái u một bit & Shift 1 vào u ;

MSB của l & u được thay bằng bit bù của nó

}

} //while

*Bước 4*: nếu x còn phần tử quay lại bước 2

ngược lại dừng .

***Ví dụ***

Cho x = 1321

TotalCount = 50

Count(1) = 40 , tương đương p(a1) = 0.8

Count(2) = 1 , tương đương p(a2) = 0.02

Count(3) = 9 , tương đương p(a3) = 0.18

Suy ra :

Cum-Count (0) = 0

Cum-Count (1) = 40

Cum-Count (2) = 41

Cum-Count (3) = 50

Chọn 2m chứa [ l, u )

Giả sử chọn m = 8 ( 8 bit )

l(0) = ( 0 0 0 0 0 0 0 0 ) 2 = 0

u(0) = ( 1 1 1 1 1 1 1 1 ) 2 = 255

**1 :**





**3 :**





Send **1**

l(2) = ( 01001110 ) 2 = 78

u(2) = ( 10010111 ) 2 = 151

thỏa E3

l(2) = ( 00011100 ) 2 = 28

u(2) = ( 10101111 ) 2 = 175

scaleE3 = 1

**2 :**





l(3) & u(3) thỏa E2 : send bit **1**

Vì scaleE3 = 1 nên send bit **0**

scaleE3 = 0

l(3) = (**0** 0 1 0 0 1 0 0 ) 2 = 36 ; u(3) = ( **0** 0 1 0 1 0 0 1 ) 2 = 41

thỏa E1 : send bit **0**

l(3) = ( **0** 1 0 0 1 0 0 0 ) 2 = 72

u(3) = ( **0** 1 0 1 0 0 1 1 ) 2 = 83

thỏa E1 : send bit **0**

l(3) = ( **1** 0 0 1 0 0 0 0 ) 2 = 144

u(3) = ( **1** 0 1 0 0 1 1 1 ) 2 = 167

thỏa E2 : send bit **1**

l(3) = ( **0** 0 1 0 0 0 0 0 ) 2 = 32

u(3) = ( **0** 1 0 0 1 1 1 1 ) 2 = 79

thỏa E1 : send bit **0**

l(3) = ( 0 1 0 0 0 0 0 0 ) 2 = 64

u(3) = ( 1 0 0 1 1 1 1 1 ) 2 = 159

thỏa E3

l(3) = ( 0 0 0 0 0 0 0 0 ) 2 = 0

u(3) = ( 1 0 1 1 1 1 1 1 ) 2 = 191

scaleE3 = 1

**1 :**



Hết dãy x

Lấy một giá trị của l(4) , u(4) mã hóa vào dãy bit.

Chọn 0 : send bit **0**

Vì scaleE3 = 1 và điều kiện phía trước là E1 nên send một bit bù là **1**

Cuối cùng : 1 1 0 0 0 1 0 0 1

* + - 1. **Thuật toán Decode**

***Thuật toán***

*Bước 1* : Khởi tạo l = 0 , u = 255 , k = 0

*Bước 2* : Đọc m bit từ dãy bit mã hóa

⇒ giá trị của m bit này là t

*Bước 3* :



**{** k = k + 1 ; **}**

Giải mã phần tử x ;



while ( MSB của l & u bằng b hay thỏa E3 )**{**

if ( MSB của l & u bằng b )**{**

Shift trái l 1 bit và Shift 0 vào l ;

Shift trái u 1 bit và Shift 1 vào u ;

Shift trái t 1 bit và đọc 1 bit từ dãy bit mã hóa rồi Shift vào t ;

**}**

if ( E3 )**{**

Shift trái l 1 bit và Shift 0 vào l ;

Shift trái u 1 bit và Shift 1 vào u ;

Shift trái t 1 bit và đọc 1 bit từ dãy bit mã hóa rồi Shift vào t ;

Lấy bit bù của MSB của l, u, t ;

**}** // if E3

**}** // while MSB

***Ví dụ***

Dãy bit mã hóa : 1 1 0 0 0 1 0 0 1 …



t nhận 8 bit : t = ( 1 1 0 0 1 0 0 ) 2 = 196

l = ( 0 0 0 0 0 0 0 0 ) 2 = 0

u = ( 1 1 1 1 1 1 1 1 ) 2 = 255



⇒ giải mã **1** ( k = 1 )

Cập nhật l & u



Vì không thỏa E1, E2, E3 nên



⇒ giải mã **3**

l = 167 = ( 1 0 1 0 0 1 1 1 ) 2

u = 203 = ( 1 1 0 0 1 0 1 1 ) 2 thỏa E2

Suy ra :

l = ( 0 1 0 0 1 1 1 0 ) 2

u = ( 1 0 0 1 0 1 1 1 ) 2

t = ( 1 0 0 0 1 0 0 1 ) 2

thỏa E3

Cập nhật :

l = ( 0 0 0 1 1 1 0 0 ) 2 = 28

u = ( 1 0 1 0 1 1 1 1 ) 2 = 175

t = ( 1 0 0 1 0 0 1 0 ) 2 = 146

Ta có:



⇒ giải mã **2**

Cập nhật :



Ta có :



⇒ giải mã **1**

Kết quả : 1 3 2 1

1. **CÀI ĐẶT THUẬT TOÁN VÀ CHƯƠNG TRÌNH DEMO**
2. **Cài đặt các thuật toán**
   1. **Thuật toán mã hóa**

static void encode\_symbol(int symbol,long cum\_freq[])

{

long range;

range =(long)(high-low)+1;

high = low + dss(range, cum\_freq[symbol - 1], cum\_freq[0]) - 1;

low = low + dss(range, cum\_freq[symbol], cum\_freq[0]);

low ^= (((long)1 << Code\_value\_bits) - 1);

high ^= (((long)1 << Code\_value\_bits) - 1);

swape = low;

low = high;

high = swape;

for(;;)

{

if (high < (2\*((((long)1 << Code\_value\_bits) - 1)/4 + 1)))

{

bit\_plus\_follow(0);

}

else if (low>=(2\*((((long) 1<< Code\_value\_bits)-1)/4 + 1)))

{

bit\_plus\_follow(1);

low -=(2\*((((long)1<<Code\_value\_bits)-1)/4+1));

high -=(2\*((((long)1<<Code\_value\_bits)-1)/4+1));

}

else if (low >=((((long)1<<Code\_value\_bits)- 1)/4 + 1) &&

high<(3\*((((long)1<<Code\_value\_bits) - 1)/4 + 1)))

{

bits\_to\_follow += 1;

low -= ((((long)1 << Code\_value\_bits) - 1)/4 + 1); high -=((((long)1 << Code\_value\_bits) - 1)/4 + 1);

}

else

break;

low = 2 \* low;

high = 2 \* high + 1;

}

low ^= (((long)1 << Code\_value\_bits) - 1);

high ^= (((long)1 << Code\_value\_bits) - 1);

swape = low;

low = high;

high = swape;

}

**Hàm compress:**

public static void compress(String input\_file, String output\_file)

{

int zerf = 0;

int onef = 0;

FileInputStream in = null;

try

{

in = new FileInputStream(input\_file);

out = new FileOutputStream(output\_file);

byte[] \_array = new byte[1];

int rb;

start\_model(); //Khởi tạo bảng mẫu.

start\_outputing\_bits();//Chuẩn bị cho bit ra.

start\_encoding();//Bắt đầu mã hóa dòng dữ liệu.

for(int i; i<200;i++)

{

int ch;

int symbol;

if ((rb = in.read(\_array, 0, 1)) == -1)

{

break;

}

ch = \_array[0] & 0xff;

symbol = char\_to\_index[ch];

if (symbol == 1) zerf = 1;

if (zerf == 1 && symbol == 2) onef = 1;

if (symbol > 1) zerf = 0;

if (symbol > 2) onef = 0;

encode\_symbol(symbol,cum\_freq);

update\_model(symbol);

}

if ((zerf + onef) > 0)

{

encode\_symbol(2, cum\_freq);

update\_model(2);

}

done\_encoding(); //kết thúc việc mã hóa ký hiệu.

done\_outputing\_bits(); //kết thúc mã hóa dòng dữ liệu.

in.close();

out.close();

main.lb3.setText("Compression successed!");

}

catch(Exception e){

main.lb3.setText("Error output file! Please check your output.");

}

}

* 1. **Thuật toán giải mã**

static int decode\_symbol(long cum\_freq[])

{

long range;

int symbol;

low ^= (((long)1 << Code\_value\_bits) - 1);

high ^= (((long)1 << Code\_value\_bits) - 1);

value ^= (((long)1 << Code\_value\_bits) - 1);

swape = low;

low = high;

high = swape;

range = (long)(high - low) + 1;

high = low + range;

symbol = 0;

while(value < high)

{

high = low + dss(range, cum\_freq[++symbol], cum\_freq[0]);

}

swape = high;

high = low + dss(range, cum\_freq[symbol - 1], cum\_freq[0]) - 1;

low = swape;

low ^= (((long)1 << Code\_value\_bits) - 1);

high ^= (((long)1 << Code\_value\_bits) - 1);

swape = low;

value ^= (((long)1 << Code\_value\_bits) - 1);

low = high;

high = swape;

for(;;)

{

if (high < (2\*((((long)1 << Code\_value\_bits) - 1)/4 + 1)))

{

}

else if (low >= (2\*((((long)1 << Code\_value\_bits)-1)/4+1)))

{

value -= (2\*((((long)1<<Code\_value\_bits) -1)/4+1));

low -= (2\*((((long)1<<Code\_value\_bits)-1)/4+1)); high -= (2\*((((long)1<<Code\_value\_bits)-1)/4+1));

}

else if (low>=((((long)1<<Code\_value\_bits)-1)/4+1)

&& high<(3\*((((long)1<<Code\_value\_bits)-1)/4+1)))

{

value -= ((((long)1<<Code\_value\_bits)-1)/4+1);

low -= ((((long)1<<Code\_value\_bits)-1)/4+1); high -= ((((long)1<<Code\_value\_bits)-1)/4+1);

}

else

break;

low = 2 \* low;

high = 2 \* high + 1;

value = 2 \* value+input\_bit();

if ((ZEND != 0) && (ZATE != 0))

{

if (ZATE++ > Code\_value\_bits - 0)

{

ZATE = 0;

}

}

if ((ZEND != 0) && (ZATE != 0) && value == 0))

{

ZATE = 0;

}

if ((ZEND != 0) && (ZATE != 0) && value ==

(2 \* ((((long)1 << Code\_value\_bits) - 1)/4 + 1)))

{

ZATE = 0;

}

}

return symbol;

}

**Hàm decompress:**

public static void decompress(String input\_file, String output\_file)

{

int symbol = 0;

try

{

in = new FileInputStream(input\_file);

out = new FileOutputStream(output\_file);

byte[] \_array = new byte[1];

start\_model();

start\_inputing\_bits();

start\_decoding();

for (;;)

{

int ch;

symbol = decode\_symbol(cum\_freq);

if (symbol == 1) zerf = true;

if (zerf && symbol == 2) onef = true;

if (symbol > 1) zerf = false;

if (symbol > 2) onef = false;

ch = index\_to\_char[symbol];

if ((ZATE == 0) && (symbol != 1))

{

break;

}

writeOut(ch);

update\_model(symbol);

}

if (!(zerf || onef))

{

writeOut(index\_to\_char[symbol]);

}

in.close();

out.close();

main.lb3.setText("Decompressing successed!");

}

catch(Exception e){

main.lb3.setText("Error output file! Please check your

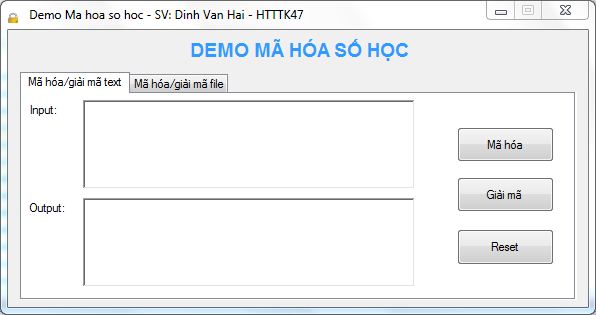
output.");

}

}

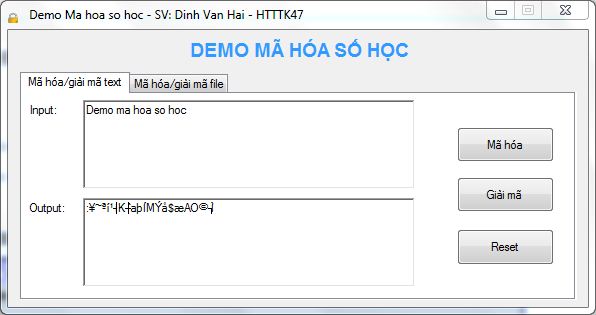
1. **Chương trình demo**

* Chương trình có hai chức năng chức năng chính đó là mã hóa – giải mã text và mã hóa – giải mã file. Giao diện chính của chương trình:

****

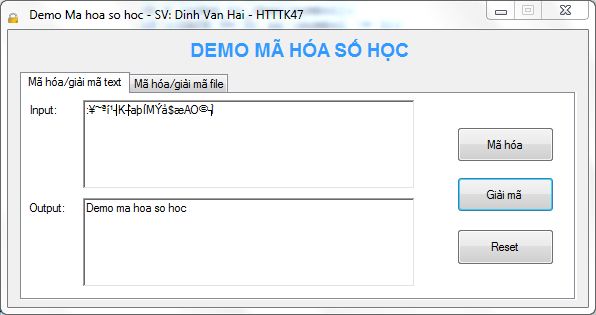
*Hình 5:* Giao diện chính của chương trình

* Chức năng mã hóa text: Trên tab *Mã hóa/giải mã text* của chương trình,bạn nhập một đoạn văn bản vào ô textbox *Input* sau đó bấm nút *Mã hóa*, kết quả mã hóa sẽ được xuất ra ô textbox *Output*:



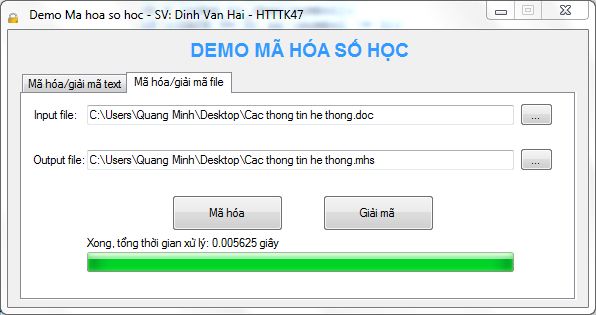
*Hình 6:* Mã hóa đoạn text

* Chức năng giải mã text: Vẫn trên tab *Mã hóa/giải mã text* của chương trình, bạn nhập chuỗi đã được mã hóa vào o textbox *Input*, sau đó bấm nút *Giải mã*, kết quả giải mã sẽ được xuất ra ô textbox *Output*:



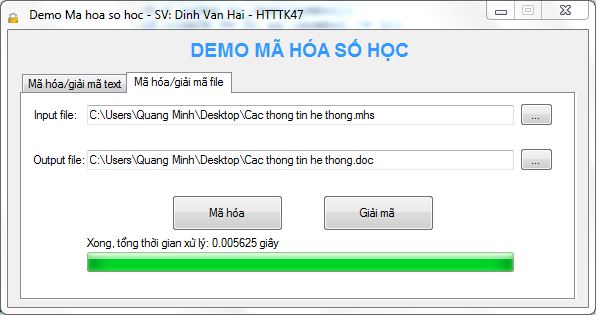
*Hình 7:* Giải mã đoạn text

* Chức năng mã hóa file: Trên tab *Mã hóa/giải mã file* của chương trình, ở ô textbox *Input file*, bạn nhập đường dẫn và tên file muốn mã hóa vào hoặc có thể click vào nút *Browse* để trỏ tới file đó. Mặc định chương trình sẽ tạo một file mã hóa cùng thư mục với file bị mã và có tên miền mở rộng là .mhs, bạn cũng có thể chọn tên file mã hóa bất kỳ theo ý mình. Sau đó click vào nút *Mã hóa* để bắt đầu mã hóa file, chương trình sẽ mã file đó và tính toán thời gian mã sau đó hiển thị lên thanh trạng thái:



*Hình 8:* Mã hóa file

* Chức năng giải mã file: Tương tự như chức năng mã hóa file, bạn cũng chọn file cần được giải mã và click vào nút *Giải mã* để lấy được file gốc:



*Hình 9:* Giải mã file

**TÀI LIỆU THAM KHẢO**

[1] Phạm Văn Ất – Khoa Công nghệ thông tin – Đại học Giao thông vận tải - Mã hóa số học

[2] Trần Văn Dũng - Khoa Công nghệ thông tin – Đại học Giao thông vận tải - Giáo trình An toàn và bảo mật thông tin

[3] Guy E. Blelloch. Computer Science Department.Carnegie Mellon University. Introduction to Data Compression. Oct 16,2001

[4] Các website sau :

<http://vi.wikipedia.org>

<http://www.fourmilab.ch/md5/>

<http://www.itgatevn.com.vn>