Mã Shannon-Fano là một thuật toán mã hóa dùng để nén dữ liệu. Nó dựa trên bảng tần

suất xuất hiện các kí tự cần mã hóa để xây dựng một bộ mã nhị phân cho các kí tự đó

sao cho dung lượng (số bít) sau khi mã hóa là nhỏ nhất.

Các tập tin của máy tính được lưu dưới dạng các kí tự có chiều dài không đổi là 8 bits.

Trong nhiều tập tin, xác suất xuất hiện các kí tự này là nhiều hơn các kí tự khác, từ đó

ta thấy ngay rằng nếu chỉ dùng một vài bit để biểu diễn cho các kí tự có xác suất xuất

hiện lớn và dùng nhiều bit hơn để biểu diễn cho các kí tự có xác suất xuất hiện nhỏ thì

có thể tiết kiệm được độ dài tập tin một cách đáng kể. Ví dụ, để mã hoá một chuỗi như

sau: "ABRACADABRA"

Nếu mã hoá chuỗi trên trong dạng mã nhị phân 5 bit ta sẽ có dãy bit sau:

0000100010100100000100011000010010000001000101001000001

Ðể giải mã thôngđiệp này, chỉ đơn giản là đọc ra 5 bits ở từng thời điểm và chuyển đổi

nó tương ứng với việc mã hoá nhị phân đã được định nghĩa ở trên. Trong mã chuẩn

này, chữ D xuất hiện chỉ một lần sẽ cần số lượng bit giống chữ A xuất hiện nhiều lần.

Ta có thể gán các chuỗi bit ngắn nhất cho các kí tự được dùng phổ biến nhất, giả sử ta

gán: A là 0, B là 1, R là 01, C là 10 và D là 11 thì chuỗi trên được biễu diễn như sau:

0 1 01 0 10 0 11 0 1 01 0

Ví dụ này chỉ dùng 15 bits so với 55 bits như ở trên, nhưng nó không thực sự là một

mã vì phải lệ thuộc vào khoảng trống để phân cách các kí tự. Nếu không có dấu phân

cách thì ta không thể giải mã được thông điệp này. Ta cũng có thể chọn các từ mã sao

cho thông điệp có thể được giải mã mà không cần dấu phân cách, ví dụ như: A là 11, B

là 00, C là 010, D là 10 và R là 011, các từ mã này gọi là các từ mã có tính prefix

(Không có từ mã nào là tiền tố của từ mã khác). Với các từ mã này ta có thể mã hoá

thông điệp trên như sau : 1100011110101110110001111

Với chuỗi đã mã hoá này ta hoàn toàn có thể giải mã được mà không cần dấu phân

cách. Nhưng bằng cách nào để tìm ra bảng mã một cách tốt nhất ?

- Bước đầu tiên trong việc xây dựng mã Shannon-Fano là đếm số lần xuất hiện của mỗi

kí tự trong tập tin sẽ được mã hoá (trong phần này chỉ ví dụ hạn chế một số ký tự).

- Bước tiếp theo là xây dựng bảng mã dựa vào thuật toán Shannon-Fano. Thuật toán

này được thực hiện bằng các bước sau:

Bước 1: Sắp xếp thứ tự các lớp tin tăng (hay giảm), nguồn tin được thống kê từ tập tin

cần thực hiện mã hóa.

Bước 2: Chia tổng nguồn tin ra làm 2 nhóm sao cho tổng của mỗi nhóm xấp xỉ bằng

nhau (hiệu của 2 nhóm là nhỏ nhất).

Bước 3: Gán cho mỗi nhóm ký hiệu là 0 hay 1.

Bước 4: Lặp lại bước 2 cho đến khi chỉ còn lại 1 nhóm tin.

- Sau khi có bảng mã Shannon-Fano, ta có thể mã hóa các gói tin. Ưu điểm của

phương pháp mã hoá Shannon-Fano là đạt được hệ số nén tương đối cao (Hệ số nén

tuỳ thuộc vào cấu trúc của các tập tin). Nhược điểm của phương pháp này là bên

nhận muốn giải mã được thông điệp thì phải có một bảng mã giống như bảng mã ở

bên gửi, do đó khi nén các tập tin bé hệ số nén không được cao.

- Để giải mã gói tin, như đã nói ở trên ta phải có bảng mã. Lần lượt ta so sánh từng

nhóm tin, nếu trùng với bảng mã thì ta có được gói tin như ban đầu.

Chương trình mã hóa nguồn dùng mã Shannon-Fano là chương trình dùng thuật

toán Shannon-Fano (có trong lý thuyết truyền tin hay kỹ thuật truyền số liệu) để giải

quyết bài toán mã hóa theo xác suất xuất hiện của các ký tự có trong bảng mã ASCII.

Sau đó, dựa vào bảng mã này ta có thể mã hóa các chuỗi ký tự hay dòng văn bản ra

thành mã máy tính gồm các ký tự 0 và 1. Các ký tự được mã hóa sẽ có độ dài khác

nhau, tuy nhiên xét về tổng thể thì độ dài của chuỗi hay của văn bản đã được mã hóa sẽ

ngắn hơn khi ta chưa mã hóa. Như vậy, khi lưu trữ sẽ ít tốn bộ nhớ hơn cũng như khi

truyền tin sẽ chiếm ít băng thông hơn.

Các chức năng của hệ thống/chương trình :

Tổng thể cấu trúc của chương trình lần lượt sẽ thực hiện như sau :

1. Khai báo các hằng, biến toàn cục, các cấu trúc sẽ thực hiện trong suốt chương

trình.

typedef struct Node

{

char kytu;

float xacsuat;

char code[32];

};

Node a[256]; // bảng mã ASCII gồm 256 ký tự, biến a chứa mảng các ký tự trong

bảng mã, là biến toàn cục.

2. Khai báo các chương trình con hay cấu trúc của các chương trình con :

a) Hàm read: đọc file và thống kê xác suất hiện của nó : Ta tiến hành đọc các

ký tự từ file rồi tăng số lần xuất hiện lên, lưu vào trong mảng a. Mảng a là

nơi để lưu trữ số lần xuất hiện của ký tự có trong file, vị trí trong mảng a là

mã ASCIIcủa ký tự đó.

b) Hàm sapxeptang : Để sắp xếp theo số lần xuất hiện của ký tự trong mảng a

theo hướng tăng dần.

c) Hàm mahoa : Hàm này sẽ mã hóa nguồn dùng mã Shannon-Fano các ký tự

có số lần xuất hiện lớn hơn 0 trong file. Mỗi ký tự đều có 1 mã bằng nhị

phân riêng.

d) Hàm giaima: Giải mã file đã mã hóa theo bảng mã.

e) Hàm insapxep: In các ký tự sau khi đã sắp xếp theo xác suất xuất hiện ra

màn hình để tiện theo dõi.

f) Hàm thuchien: Đây là hàm làm công việc chính của chương trình. Hàm này

sẽ thực hiện công việc mã hóa file rồi lưu vào file khác mã của chương trình

nén.

g) Hàm main: chứa các thông tin cần thiết của đề tài và thực hiện gọi các

chương trình con.

Ví dụ : cho nguồn tin gồm 10 lớp tin như sau :

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| X | A | B | C | H | N | U | G | O | M | I |
| Px | 0.20 | 0.11 | 0.08 | 0.10 | 0.07 | 0.09 | 0.13 | 0.06 | 0.12 | 0.04 |

a) Hàm sắp xếp : Hàm này sẽ tiến hành sắp xếp lại các ký tự theo xác suất xuất

hiện đã có ở trên theo thứ tự giảm dần nhằm phục cho quá trình mã hóa các

ký tự này theo thuật toán Shannon-Fano.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| X | A | G | M | B | H | U | C | N | O | I |
| Px | 0.20 | 0.13 | 0.12 | 0.11 | 0.10 | 0.09 | 0.08 | 0.07 | 0.06 | 0.04 |

b) Hàm mã hóa : Hàm này có nhiệm vụ tối quan trọng đối với bài toán này.

Hàm sẽ mã hóa các ký tự đã nhập hay cho trước sao cho các mã này không

được trùng nhau. Các bước thực hiện như sau :

∙ Bước 1 : Dựa vào thứ tự đã sắp xếp ở trên chia mảng a ra làm 2 phần. Điều

kiện là hiệu của tổng các xác suất có trong từng phần sẽ là nhỏ nhất.

∙ Bước 2 : Gán cho mỗi ký tự của nhóm đầu giá trị 0 vào trong phần code

của nó (ký tự) và nhóm thứ 2 giá trị 1.

∙ Bước 3 : Làm lại bước 1 với nhóm thứ nhất sau đó là với nhóm thứ 2.

Điều kiện để dừng là không thể chia nhỏ được nữa (mảng chỉ còn 1 phần

tử).

Như vậy, sau khi hoàn tất các bước trên, ta có được 1 bảng mã với các chữ

cái được gắn với một mã tương ứng nhất định, các mã này sẽ không trùng

nhau.

Ví dụ : Bảng mã sau khi được mã hóa :

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| X | A | G | M | B | H | U | C | N | O | I |
| Px | 0.20 | 0.13 | 0.12 | 0.11 | 0.10 | 0.09 | 0.08 | 0.07 | 0.06 | 0.04 |
| Code | 00 | 010 | 011 | 100 | 1010 | 1011 | 1100 | 1101 | 1110 | 1111 |

c) Hàm giải mã : Hàm này sẽ dựa vào bảng mã ở trên để giải mã một chuỗi

gồm các ký tự 0 và 1. Hàm có kiểm tra xem có phải dữ liệu nhập vào có đúng như yêu cầu không và các ký tự đó phải có trong bảng mã đã thống kê

ở trên. Trình tự các bước như sau :

∙ Bước 1 : Kiểm tra xem ký tự đầu tiên có trong bảng mã hay không. Nếu

có thì lưu vào trong mảng d và tiếp tục làm bước 1 nhưng vị trí đầu tiên là vị

trí kế tiếp (chưa kiểm tra) , nếu không có thì tiếp tục bước 2.

∙ Bước 2 : Kiểm tra cặp ký tự đầu tiên và ký tự thứ 2 xem có trong bảng

mã hay không. Nếu có thì lưu vào trong mảng d, và quay lại bước 1 với ký

tự đầu tiên là ký tự kế tiếp, nếu không có thì làm lại bước 2 với 3, 4,…,n (số

phần tử của chuỗi nhập vào) vị trí kế tiếp.

∙ Bước 3 : Kiểm tra xem nếu kiểm tra đến phần tử cuối cùng mà không

thấy trùng với phần tử nào trong bảng mã thì phát thông báo “ Dữ liệu nhập

vào bị lỗi”. Làm đến khi không còn phần tử nào trong mảng thì dừng.

∙ Bước 4 : in các ký tự vừa được giải mã hoặc thông báo lỗi lên màn hình.

d) Hàm main : Hàm này hiển thị thông tin người thực hiện đề tài…. Các lựa

chọn mã hóa file hay giải mã. Hiện lên màn hình thông báo để người sử

dụng biết đã thực hiện mã hóa hay giải mã xong chưa, file mã hóa hay giải

mã được lưu ở đâu.

Thuật toán Huffman:

∙ Thành lập cây nhị phân từ tập hợp các kí hiệu trong thông báo, mỗi kí hiệu là

một nút lá của cây. Cách thành lập cây như sau:

Chọn hai nút a,b có xác suất nhỏ nhất trong tập hợp các nút, giả sử xác suất nút a nhỏ

hơn hoặc bằng xác suất nút b. Thành lập cây nhị phân có nút gốc x, con trái là a, con

phải là b. Nút x có xác suất bằng tổng xác suất của a và b.

∙ Tập hợp các nút bây giờ là các nút còn lại (đã loại bỏ a, và nút x. Lặp lại một

cách đệ qui quá trình trên tập hợp đang xét cho đến khi tập này chỉ còn lại một nút.

∙ Mã của a, b sẽ tìm được bằng cách lấy mã của x nối thêm 0 cho a và 1 cho b.

Mã của nút gốc là rỗng.

∙ Như vậy thực chất quá trình trên là ta xây dựng một cây nhị phân từ tập hợp các

ký tự muốn mã hoá, cuối cùng ta được một cây nhị phân có lá là các ký tự đó. Mã của

một ký tự là một đường đi trên cây từ gốc đến lá chứa kí tự, với 0 đi sang trái còn 1 đi

sang phải. Ý tưởng của giải thuật mã hoá cũng hết sức đơn giản, ta tìm bộ mã cho các

kí tự sao cho các kí tự có tần suất xuất hiện cao (xác suất xuất hiện là lớn) sẽ được mã

ngắn (gần với gốc) để độ dài trung bình để mã hoá một kí tự là nhỏ nhất.

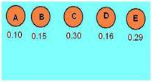
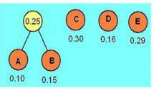
∙ Ví dụ:

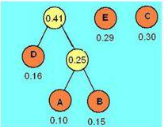
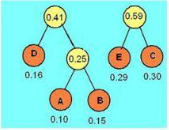
Cho bảng tần suất của 5 chữ cái A,B,C,D,E như sau tương ứng là 0.10; 0.15; 0.30;

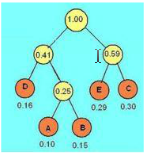
0.16; 0.29.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| A | B | C | D | E |
| 0.10 | 0.15 | 0.30 | 0.16 | 0.29 |

Quá trình xây dựng cây Huffman diễn ra như sau:



Như vậy bộ mã tối ưu tươngứng là:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| A | B | C | D | E |
| 010 | 011 | 11 | 00 | 10 |

- Đọc file văn bản, lưu vào mảng a.

- Xây dựng cây Huffman để giải mã dòng văn bản

- Hiển thị cây Huffman và bảng mã Huffman ra màn hình

- Thực hiện mã hóa dòng văn bản và dải mã

- Mở rộng mã hóa và giải mã một file văn bản. Kết quả giải mã và mã hóa được ghi

vào 2 file văn bản khác.

∙ Cấu trúc chương trình:

- File huffman.cpp dùng để cài đặt cây huffman. Nó bao gồm các hàm:

- mahoa\_chuoi(): Mã hóa chuỗi.

- giaima\_chuoi(): Giải mã chuỗi.

- general(): Cài đặt cây Huffman, được sử dụng trong các hàm trên.

- make\_table(): Tạo bảng mã cho các ký tự, được sử dụng trong các hàm mã hóa.

- quit(): Các thao tác cuối cùng trước khi return.

∙ Các vấn đề chung trong xây dựng chương trình:

Các cấu trúc dữ liệu sử dụng trong chương trình:

Code:

typedef struct {

char kytu;

int tansuat;

char code[MAXBITS];

}hlist;

- Kiểu dữ liệu của mảng a[] chứa tập ký tự (a[i].kytu), tần số tươngứng

a[i].tansuat) và chuỗi mã hóa ký tự đó (a[i].code)

Code:

typedef struct{

char kytu;

int contrai,conphai,nutcha;

int tansuat;

int isleft;

}hnode;

- Kiểu dữ liệu của mảng node[] dùng để cài đặt cây Huffman. Các node tương ứng với ký tự (node[i].kytu nếu có). Node.contrai, node.conphai, node.nutcha tương ứng là chỉ số của nút con trái, con phải, và nút cha (Nếu không có thì có giá trị là -1).

Node.tansuat chứa tổng tần số các nút lá thuộc nhánh của nó. Node.isleft bằng 1 nếu

nút đó là con trái của cha nó, bằng 0 nếu là con phải, bằng -1 nếu là nút gốc (goc).

Cài đặt cây Huffman từ tập ký tự đọc từ file:

- Thủ tục general()

- Sắp xếp lại các a[i]

- Khởi tạo các node[i] (i=0 đến n-1), node[i].kytu và node[i].tansuat tương ứng với

a[i].kytu và a[i].tansuat sau khi đã sắp xếp. Các thành phần còn lại có giá trị là –1

(chưa xác định).

- Tạo cây Huffman bằng cách chèn thêm nút mới đồng thời sắp xếp lại theo thứ tự tần

số tăng dần.

Các vấn đề khác cần giải quyết:

1. Lập bộ ký tự (a[i].kytu) và tần số tương ứng (a[i].tansuat) từ một xâu ký tự (s):

- Đọc ký tự đầu tiên của xâu cho vào a[0].kytu tương ứng là a[0].tansuat bằng 1.

- Duyệt từng ký tự còn lại của xâu, nếu gặp ký tự nào đã có trong mảng a[i].kytu thì

tăng a[i].tansuat lên 1, nếu chưa có ký tự đó thì thêm phần tử mới vào mảng và cho tần

số tươngứng bằng 1.

2. Lập tập ký tự trong file text và tần số tươngứng: (Tương tự với xâu ký tự)

3. Xây dựng bảng mã Huffman (lưu trong a[i].code): Thủ tục make\_table()

- Duyệt tất cả các node[i] (i từ 0 tới 2\*n-2)

- Nếu gặp node nào có chứa ký tự (node[i].kytu khác NULL) thì duyệt lên tới root để

được xâu mã hóa (a[i].code) viết theo chiều ngược lại.

- Đảo ngược a[i].code bằng hàm strrev(), ta được xâu mã hóa của ký tự tươngứng.

4. Mã hóa:

- Đọc từng ký tự của chuỗi (hoặc file), gặp phần tử nào thì hiển thị xâu mã hóa tương

ứng hoặc ghi thêm xâu mã hóa tươngứng của ký tự đó vào file đã mã hóa (fileout).

5. Giải mã:

- Goc trỏ vào node gốc (goc=2\*n-2)

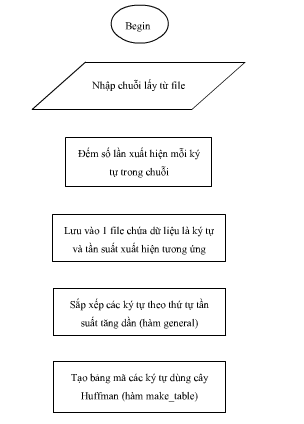
- Duyệt cây Huffman từ trên xuống, gặp 0 thì nhảy xuống con trái, gặp 1 thì nhảy

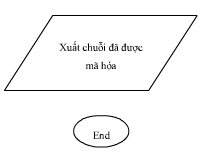
xuống con phải, cho tới khi gặp node có thành phần kytu khác -1.

- Nếu gặp node có thành phần kytu khác -1 thì hiển thị ký tự của node đó và nhảy về

gốc (goc=2\*n-2).

Lưu đồ thuật toán huffman





Ưu điểm, nhược điểm của phương pháp mã hoá Shannon-Fano và Huffman

➢ Ưu điểm:

- Thuật toán nén Shannon-Fano và Huffman là thuật toán nén không tổn hao.

Do vậy, dữ liệu sau khi giải nén là nguyên vẹn.

- Hai thuật toán nén trên đều sử dụng phương pháp nén dữ liệu bằng cách mã

hóa các ký tự sao cho số lần xuất hiện của ký tự tỉ lệ nghịch với số bits được mã hóa,

làm cho dữ liệu được lưu trữ hay truyền đi có kích thước nhỏ hơn rất nhiều so với ban

đầu.

- Tốc độ nén nhanh do việc xây dựng bảng mã không phải mất nhiều thời gian.

- Sử dụng hai thuật toán nén trên không làm tốn bộ nhớ nhiều, việc lưu trữ dễ

dàng.

- Nhìn vào bảng thống kê và biểu đồ, ta có thể thấy được tỉ lệ nén file text của 2

mã nén này là rất tốt, file có kích thước càng lớn thì tỉ lệ nén càng cao (nhất là mã nén

Huffman).

➢ Nhược điểm:

- Việc nén dữ liệu là tương đối tốt đối với các loại file văn bản, tuy nhiên nó

không hiệu quả đối với các loại file khác.

- Bảng mã luôn phải đi kèm với dữ liệu nên phải mất một khoảng vùng nhớ cho

nó.

- Luôn luôn phải duyệt thông tin đến 2 lần. Một lần đọc dữ liệu để tạo bảng mã

và một lần để mã hóa.

- Việc mã hóa file có kích thước nhỏ thường không cho tỉ lệ nén tốt so với file

có kích thước lớn hơn.