Cấu trúc dữ liệu và giải thuật

NÉN DỮ LIỆU

Giảng viên:
Văn Chí Nam – Nguyễn Thị Hồng Nhung – Đặng Nguyễn Đức Tiến

Nội dung trình bày

- Giới thiệu
- Một số khái niệm
- Nén Huffman tĩnh
- Nén Run-Length Encoding
- Nén LZW

Giới thiệu

3

- Thuật ngữ:
  - Data compression
  - Encoding
  - Decoding
  - Lossless data compression
  - Lossy data compression
  - …

Giới thiệu

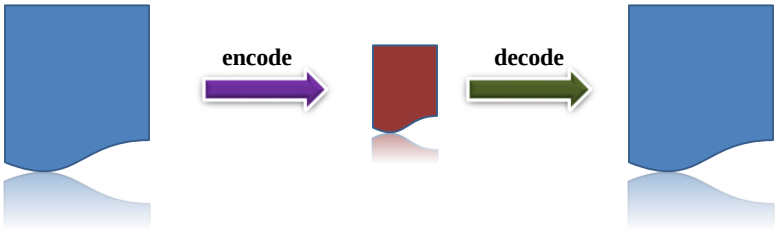
4

- Nén dữ liệu
  - Nhu cầu xuất hiện ngay sau khi hệ thống máy tính đầu tiên ra đời.
  - Hiện nay, phục vụ cho các dạng dữ liệu đa phương tiện
  - Tăng tính bảo mật.
- Ứng dụng:
  - Lưu trữ
  - Truyền dữ liệu

Giới thiệu

5

- Nguyên tắc:
  - Encode và decode sử dụng cùng một scheme.



Khái niệm

6

- Tỷ lệ nén (Data compression ratio)
  - Tỷ lệ giữa kích thước của dữ liệu nguyên thủy và của dữ liệu sau khi áp dụng thuật toán nén.
  - Gọi:
    - N là kích thước của dữ liệu nguyên thủy,
    - N_1 là kích thước của dữ liệu sau khi nén.
    - Tỷ lệ nén R: $R = \frac{N}{N_1}$
  - Ví dụ:
    - Dữ liệu ban đầu 8KB, nén còn 2 KB. Tỷ lệ nén: 4-1

Khái niệm

7

- Tỷ lệ nén (Data compression ratio)
  - Về khả năng tiết kiệm không gian: Tỷ lệ của việc giảm kích thước dữ liệu sau khi áp dụng thuật toán nén.
  - Gọi:
    - N là kích thước của dữ liệu nguyên thủy,
    - N_1 là kích thước của dữ liệu sau khi nén.
    - Tỷ lệ nén R: $$R = 1 - \frac{N_1}{N}$$
  - Ví dụ:
    - Dữ liệu ban đầu 8KB, nén còn 2 KB. Tỷ lệ nén: 75%

Khái niệm

8

- Nén dữ liệu không mất mát (Lossless data compression)
  - Cho phép dữ liệu nén được phục hồi nguyên vẹn như dữ liệu nguyên thủy (lúc chưa được nén).
  - Ví dụ:
    - Run-length encoding
    - LZW
    - …
  - Ứng dụng:
    - Ảnh PCX, GIF, PNG,..
    - Tập tin *. ZIP
    - Ứng dụng gzip (Unix)

Khái niệm

9

- Nén dữ liệu có mất mát (Lossy data compression)
  - Dữ liệu nén được phục hồi
    - không giống hoàn toàn với dữ liệu nguyên thủy;
    - gần đủ giống để có thể sử dụng được.
  - Ứng dụng:
    - Dùng để nén dữ liệu đa phương tiện (hình ảnh, âm thanh, video):
      - Ảnh: JPEG, DjVu;
      - Âm thanh: AAC, MP2, MP3;
      - Video: MPEG-2, MPEG-4

10

Nén Huffman tĩnh

Giới thiệu

11

- Mong muốn:
  - Một giải thuật nén bảo toàn thông tin;
  - Không phụ thuộc vào tính chất của dữ liệu;
  - Ứng dụng rộng rãi trên bất kỳ dữ liệu nào, với hiệu suất tốt.

Giới thiệu

12

- Tư tưởng chính:
  - Phương pháp cũ: dùng 1 dãy bit cố định để biểu diễn 1 ký tự
  - David Huffman (1952): tìm ra phương pháp xác định mã tối ưu trên dữ liệu tĩnh :
    - Sử dụng vài bit để biểu diễn 1 ký tự (gọi là "mã bit" – bit code)
    - Độ dài "mã bit" cho các ký tự không giống nhau:
    - Ký tự xuất hiện nhiều lần: biểu diễn bằng mã ngắn;
    - Ký tự xuất hiện ít : biểu diễn bằng mã dài

  => Mã hóa bằng mã có độ dài thay đổi (Variable Length Encoding)

Giới thiệu

13

- Giả sử có dữ liệu sau đây:

  ADDAABBCCBAAABBCCCBBBCDAADDEEAA

Ký tự	Tần số xuất hiện
A	10
B	8
C	6
D	5
E	2

- Biểu diễn 8 bit/ký tự cần:

  (10 + 8 + 6 + 5 + 2) * 8 = **248 bit**

Giới thiệu

14

- Dữ liệu:

  ADDAABBCCBAAABBCCCBBBCDAADDEEAA

- Biểu diễn bằng chiều dài thay đổi:

Ký tự	Tần số	Mã
A	10	11
B	8	10
C	6	00
D	5	011
E	2	010

(10*2 + 8*2 + 6*2 + 5*3 + 2*3) = **69 bit**

Thuật toán nén

15

[B1]: Duyệt tập tin -> Lập bảng thống kê tần số xuất hiện của các ký tự.

[B2]: Xây dựng cây Huffman dựa vào bảng thống kê tần số xuất hiện

[B3]: Phát sinh bảng mã bit cho từng ký tự tương ứng

[B4]: Duyệt tập tin -> Thay thế các ký tự trong tập tin bằng mã bit tương ứng.

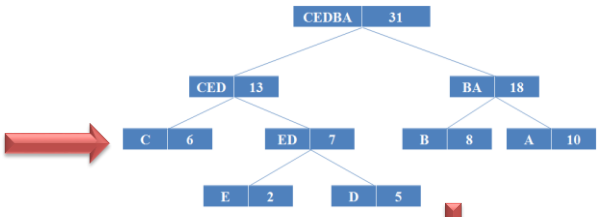
[B5]: Lưu lại thông tin của cây Huffman cho giải nén

Thuật toán nén

16

ADDAABBCCBAAABBCCCBBBCDAADDEEAA

Ký tự	Tần số
A	10
B	8
C	6
D	5
E	2



Ký tự	Mã
A	11
B	10
C	00
D	011
E	010

11011011111110100000101111110100000001010100001111110110110100101111

Thuật toán nén – Thống kê tần số

17

- Dữ liệu:

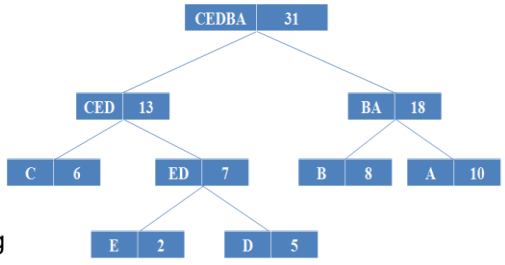
ADDAABBCCBAAABBCCCBBBCDAADDEEAA

Ký tự	Tần số xuất hiện
A	10
B	8
C	6
D	5
E	2

Thuật toán nén – Tạo cây Huffman

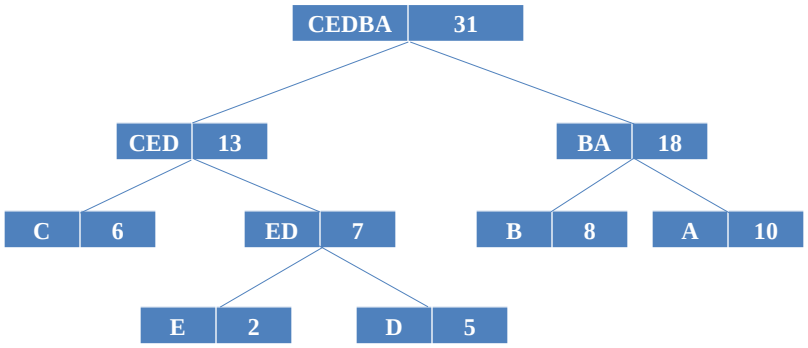
18

- Cây Huffman: cây nhị phân
  - Mỗi node lá chứa 1 ký tự
  - Mỗi node cha chứa các ký tự của những node con.
  - Trọng số của node:
    - Node con: tần số xuất hiện của ký tự tương ứng
    - Node cha: Tổng trọng số của các node con.



Thuật toán nén – Tạo cây Huffman

19



Thuật toán nén – Tạo cây Huffman

20

- Phát sinh cây:
  - Bước 1: Chọn trong bảng thống kê hai phần tử ***x***,***y*** có trọng số thấp nhất.
  - Bước 2: Tạo 2 node của cây cùng với node cha ***z*** có trọng số bằng tổng trọng số của hai node con.
  - Bước 3: Loại 2 phần tử ***x***,***y*** ra khỏi bảng thống kê.
  - Bước 4: Thêm phần tử ***z*** vào trong bảng thống kê.
  - Bước 5: Lặp lại Bước 1-4 cho đến khi còn 1 phần tử trong bảng thống kê.

Thuật toán nén – Tạo cây Huffman

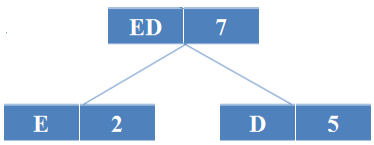
21

- Quy ước:
  - Node có trọng số nhỏ hơn sẽ nằm bên nhánh trái. Node còn lại nằm bên nhánh phải.
  - Nếu 2 node có trọng số bằng nhau
    - Node nào có ký tự nhỏ hơn thì nằm bên trái
    - Node có ký tự lớn hơn nằm bên phải.

Thuật toán nén – Tạo cây Huffman

22

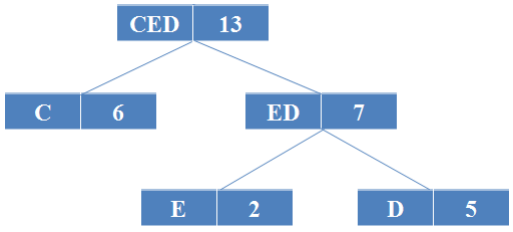
Ký tự	Tần số
A	10
B	8
C	6
D	5
E	2



Thuật toán nén – Tạo cây Huffman

23

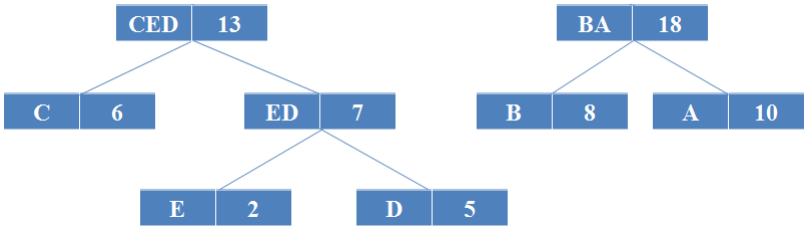
Ký tự	Tần số
A	10
B	8
ED	**7**
C	**6**



Thuật toán nén – Tạo cây Huffman

24

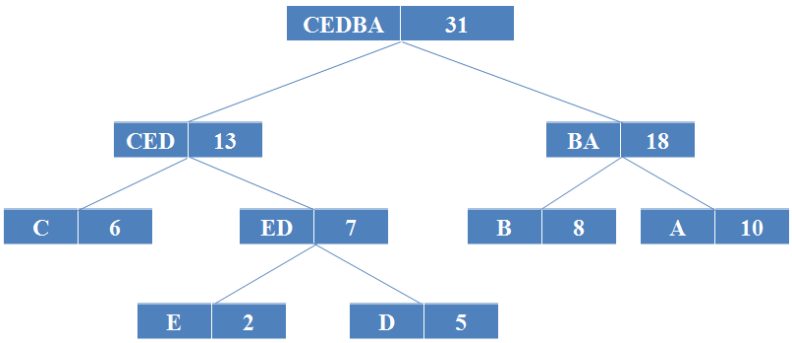
Ký tự	Tần số
CED	13
A	**10**
B	**8**



Thuật toán nén – Tạo cây Huffman

25

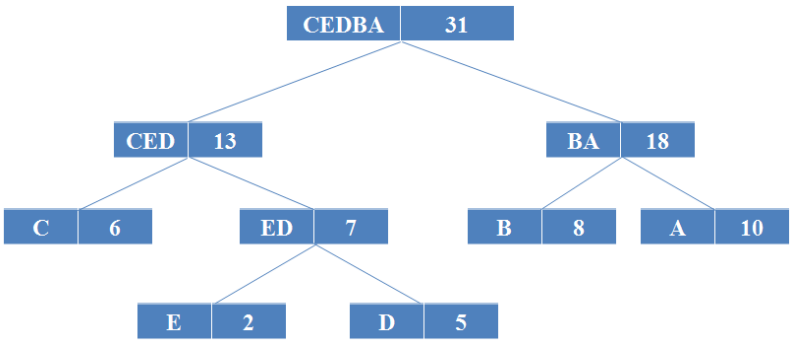
Ký tự	Tần số
BA	18
CED	13



Thuật toán nén – Tạo cây Huffman

26

Ký tự	Tần số
CEDBA	31



Thuật toán nén – Phát sinh mã bit

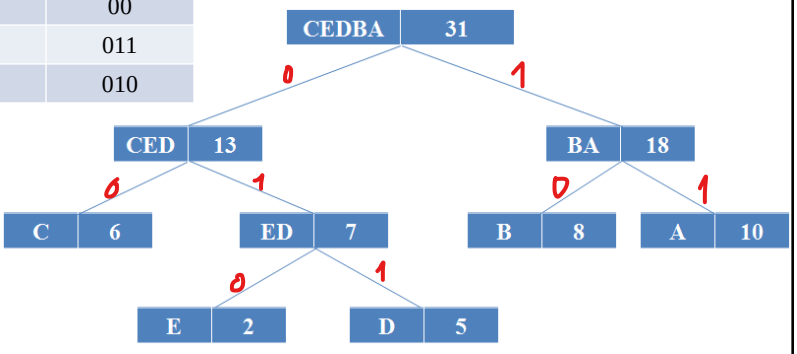
27

- Mã bit của từng ký tự: đường đi từ node gốc của cây Huffman đến node lá của ký tự đó.
- Cách thức:
  - Bit 0 được tạo ra khi đi qua nhánh trái
  - Bit 1 được tạo ra khi đi qua nhánh phải

Thuật toán nén – Phát sinh mã bit

28

Ký tự	Mã
A	11
B	10
C	00
D	011
E	010



Thuật toán nén – Nén dữ liệu

29

- Duyệt tập tin cần nén
- Thay thế tất cả các ký tự trong tập tin bằng mã bit tương ứng của nó.

Thuật toán nén – Lưu lại thông tin

30

- Phục vụ cho việc giải nén.
- Cách thức:
  - Cây Huffman
  - Bảng tần số

Thuật toán giải nén

31

- Phục hồi cây Huffman dựa trên thông tin đã lưu trữ.
- Lặp
  - Đi từ gốc cây Huffman
  - Đọc từng bit từ tập tin đã được nén
    - Nếu bit 0: đi qua nhánh trái
    - Nếu bit 1: đi qua nhánh phải
    - Nếu đến node lá: xuất ra ký tự tại node lá này.
- Cho đến khi nào hết dữ liệu

Vấn đề khác

32

- Có thể không lưu trữ cây Huffman hoặc bảng thống kê tần số vào trong tập tin nén hay không?

Vấn đề khác

33

- Thống kê sẵn trên dữ liệu lớn và tính toán sẵn cây Huffman cho bộ mã hóa và bộ giải mã.
- Ưu điểm:
  - Giảm thiểu kích thước của tập tin cần nén.
  - Giảm thiểu chi phí của việc duyệt tập tin để lập bảng thống kê
- Khuyết điểm:
  - Hiệu quả không cao trong trường hợp khác dạng dữ liệu đã thống kê

34

Nén Run-Length Encoding

Giới thiệu

35

- Một thuật toán nén đơn giản
- Dạng nén không mất mát dữ liệu
- Có vài 'biến thể' cải tiến để đạt hiệu quả nén cao hơn
  - Nén trên ảnh PCX
  - Nén trên ảnh BMP
  - ..

Khái niệm

36

- Đường chạy (run)
  - Dãy các ký tự giống nhau liên tiếp
- Ví dụ:
  - Chuỗi: **AAAbbbbbCdddEbbbb**
  - Các đường chạy:
    - AAA
    - bbbbb
    - C
    - ddd
    - E
    - bbbb

Ý tưởng

37

- Run-Length-Encoding: mã hóa (nén) dựa trên chiều dài của đường chạy.
  - Đường chạy được biểu diễn lại:
    <Số lượng ký tự> <Ký tự>
- Ví dụ:
  - Chuỗi đầu vào: **AAAbbbbbCdddEbbbb** (#17 bytes)
  - Kết quả nén: **3A5b1C3d1E4b** (#12 bytes)

Ý tưởng

38

- Trong thực tế, có khả năng gây '**hiệu ứng ngược**':
  - Dữ liệu nén: ABCDEFGH (8 bytes)
  - Kết quả nén: 1A1B1C1D1E1F1G1H (16 bytes)
- Cần phải có những hiệu chỉnh thích hợp
  - Nén RLE trên ảnh PCX
  - Nén RLE trên ảnh BMP

Nén RLE trên PCX

39

- Khắc phục trường hợp 'hiệu ứng ngược':
  - Byte xác định số lượng (nhiều hơn 1): 2 bit 6,7 được bật.
- Ví dụ:
  - Chuỗi gồm 5 ký tự A, 0x41, (**AAAAA**) được mã hóa

1	1	0	0	0	1	0	1	0	1	0	0	0	0	0	1
0xC5								0x41							
197_{10}								65_{10}							

Nén RLE trên PCX

40

- Khắc phục trường hợp 'hiệu ứng ngược':
  - Byte xác định số lượng : 2 bit 6,7 được bật.
    - Số lần lặp (số lượng) tối đa: **63**
    - Giá trị dữ liệu tối đa: **191** (0-191)
  - Số lần lặp là 1?
    - Dữ liệu có giá trị dưới 192?
    - Dữ liệu có giá trị từ 192?

Nén RLE trên PCX

41

- Số lần lặp là 1?
  - Dữ liệu có giá trị dưới 192?
    - Không ảnh hưởng
    - Ví dụ: nén 2 ký tự **0x41 0x43**

0	1	0	0	0	0	0	1	0	1	0	0	0	0	1	1

  - Dữ liệu có giá trị từ 192?
    - Ảnh hưởng (nhầm lẫn với thông tin số lượng).
    - Sử dụng 2 byte: <Số lượng = 1> <Dữ liệu>
    - Ví dụ: nén ký tự **0xDB** (219_{10})

1	1	0	0	0	0	0	1	1	1	0	1	1	0	1	1

Nén RLE trên PCX - Nhận xét

42

- Ưu điểm:
  - Cài đặt đơn giản
  - Giảm các trường hợp "hiệu ứng ngược" của những đường chạy đặc biệt
- Khuyết điểm:
  - Dùng 6 bit biểu diễn số lần lặp chỉ thể hiện được chiều dài tối đa 63.
  - Các đoạn lặp dài sẽ phải lưu trữ lặp lại
  - Không giải quyết được trường hợp "hiệu ứng ngược" với đường chạy đặc biệt có mã ASCII >= 192

Nén RLE trên BMP

46

- Điểm hạn chế của RLE trên PCX:
  - Nén 255 ký tự A?

AAA...AAA...AAA

0xFF 'A' 0xFF 'A' 0xFF 'A' 0xFF 'A' 0xC3 'A'

(Do 255 = 4 x 63 + 3)

Nén RLE trên BMP

47

- Ý tưởng:
  - Xử lý riêng biệt trường hợp đường chạy với trường hợp dãy các ký tự riêng lẻ.
    - Ví dụ: AAAAABCDEF
  - Có sử dụng các ký hiệu đánh dấu

Nén RLE trên BMP

48

- Hiện thực:
  - Trường hợp là đường chạy:

    <Số lượng lặp lại> <Ký tự>

Dữ liệu mã hóa	Dữ liệu giải mã
0x01 0x00	0x00
0x0A 0xFF	0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF

Nén RLE trên BMP

49

- Hiện thực:
  - Trường hợp là ký tự riêng lẻ:

    <Ký tự đánh dấu> <Số lượng ký tự của dãy> <Dãy các ký tự đơn lẻ>

    - Ký tự đánh dấu: **0x00**
    - Dùng trong trường hợp dãy có từ 3 ký tự riêng lẻ trở lên.
    - Ví dụ:

Dữ liệu mã hóa	Dữ liệu giải mã
00 03 01 02 03	01 02 03
00 04 0x41 0x42 0x43 0x44	0x41 0x42 0x43 0x44

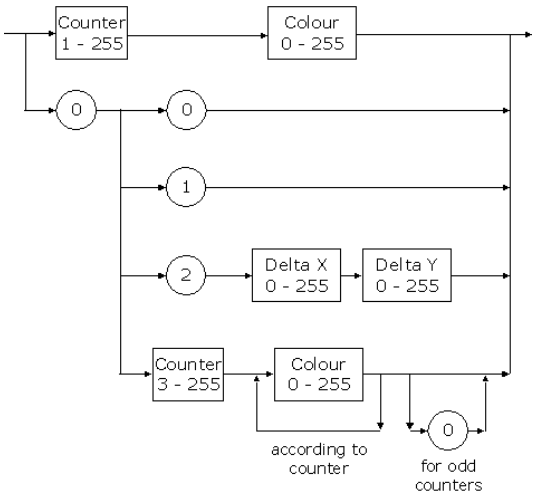
Nén RLE trên BMP

50

- Hiện thực:
  - Các trường hợp khác:
    - **0x00** 0x00: kết thúc dòng
    - **0x00** 0x01: kết thúc tập tin
    - **0x00** 0x02 <DeltaX> <DeltaY>: đoạn nhảy (DeltaX, DeltaY) tính từ vị trí hiện tại. Dữ liệu kế tiếp được áp dụng tại vị trí mới.

Nén RLE trên BMP – Tóm tắt

51



Nén RLE trên BMP – Ví dụ

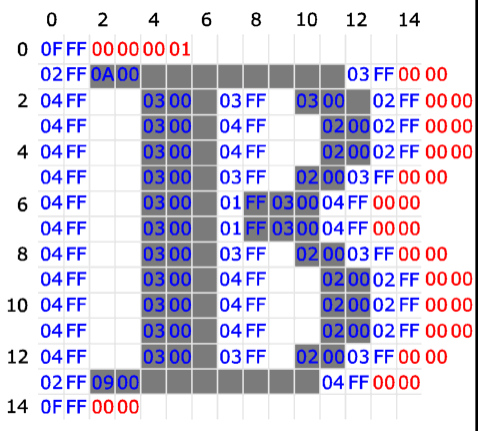
52

14	0F FF 00 00
13	02 FF 09 00 04 FF 00 00
12	04 FF 03 00 03 FF 02 00 03 FF 00 00
11	04 FF 03 00 04 FF 02 00 02 FF 00 00
10	04 FF 03 00 04 FF 02 00 02 FF 00 00
09	04 FF 03 00 04 FF 02 00 02 FF 00 00
08	04 FF 03 00 03 FF 02 00 03 FF 00 00
07	04 FF 03 00 01 FF 03 00 04 FF 00 00
06	04 FF 03 00 01 FF 03 00 04 FF 00 00
05	04 FF 03 00 03 FF 02 00 03 FF 00 00
04	04 FF 03 00 04 FF 02 00 02 FF 00 00
03	04 FF 03 00 04 FF 02 00 02 FF 00 00
02	04 FF 03 00 03 FF 03 00 02 FF 00 00
01	02 FF 0A 00 03 FF 00 00
00	0F FF 00 00 00 01

Nén RLE trên BMP – Ví dụ

53

14	0F FF 00 00
13	02 FF 09 00 04 FF 00 00
12	04 FF 03 00 03 FF 02 00 03 FF 00 00
11	04 FF 03 00 04 FF 02 00 02 FF 00 00
10	04 FF 03 00 04 FF 02 00 02 FF 00 00
09	04 FF 03 00 04 FF 02 00 02 FF 00 00
08	04 FF 03 00 03 FF 02 00 03 FF 00 00
07	04 FF 03 00 01 FF 03 00 04 FF 00 00
06	04 FF 03 00 01 FF 03 00 04 FF 00 00
05	04 FF 03 00 03 FF 02 00 03 FF 00 00
04	04 FF 03 00 04 FF 02 00 02 FF 00 00
03	04 FF 03 00 04 FF 02 00 02 FF 00 00
02	04 FF 03 00 03 FF 03 00 02 FF 00 00
01	02 FF 0A 00 03 FF 00 00
00	0F FF 00 00 00 01



Nén RLE trên BMP – Ví dụ

54

Cho đoạn dữ liệu trong tập tin BMP (đã được mã hóa bằng thuật toán nén Run Length Encoding):

0x01 0x00 0x00 0x04 0x4F 0xFC 0xA7 0x42 0x03 0xFF 0x02 0x00 0x00 0x03 0xFF 0xFE 0xFF 0x00

Đoạn dữ liệu được giải mã:

Số byte của đoạn giải mã được là:

So sánh

55

So sánh giữa RLE trên PCX và trên BMP?

Nhận xét

56

- Dùng để nén các dữ liệu có nhiều đoạn lặp lại.
- Thích hợp cho dữ liệu ảnh -> ứng dụng hẹp
- Chưa phải là một thuật toán nén có hiệu suất cao
- Đơn giản, dễ cài đặt

57

Nén LZW

Giới thiệu

58

- LZW được phát minh bởi Abraham **L**empel, Jacob **Z**iv, và Terry **W**elch.
- Thuật toán này được ra đời năm 1984 khi Terry Welch cải tiến thuật toán LZ78 (năm 1978).
- Thuộc họ thuật toán LZ, sử dụng bộ từ điển động.
  - Chuỗi ký tự trong văn bản gốc được thay thế bằng mã xác định một cách tự động.
  - Người mã hoá và người giải mã cùng xây dựng bảng mã.

Ý tưởng

59

- Ghi nhớ tất cả các chuỗi ký tự (từ 2 ký tự trở lên) đã gặp và gán cho nó một ký hiệu (token) riêng.
- Nếu lần sau gặp lại chuỗi ký tự đó, chuỗi ký tự sẽ được thay thế bằng ký hiệu (đã được gán trước đó).
- Bảng mã không cần được lưu trữ vào trong tập tin mã hóa vì hoàn toàn có thể được tạo lại trong quá trình giải nén.

Ý tưởng

60

- Cần một "từ điển" (bảng mã) để lưu giữ các chuỗi ký tự đã gặp.
- Dữ liệu cần nén được so sánh với "từ điển"
  - Nếu đã có trong "từ điển" thì đưa ra ký hiệu tương ứng của chuỗi.
  - Nếu không có thì thêm ký hiệu mới vào "từ điển".

Thuật toán nén

61

- Bước 1: Khởi tạo từ điển gồm tất cả các ký tự (chiều dài là 1).
- Bước 2: Tìm chuỗi dài nhất W trong từ điển khớp hoàn toàn với chuỗi ký tự cần nén hiện tại.
- Bước 3: Xuất ký hiệu của W (từ từ điển).
- Bước 4: Thêm chuỗi W và ký tự đằng sau vào từ điển. Gán ký hiệu thích hợp cho chuỗi này.
- Bước 5: Khi chưa hết chuỗi cần nén, lặp lại bước 2.

Mã giả

62

```
string Pre;
char CurrentValue;
Pre = empty string;
while (Vẫn còn ký tự để đọc) {
   CurrentValue = Đọc một ký tự;
        if (Pre+CurrentValue Có trong Từ điển)
                Pre = Pre+CurrentValue;
        else {
                Ghi ký hiệu của Pre vào tập tin;
                Thêm Pre+CurrentValue vào Từ điển;
                Pre = CurrentValue;
        }
}
Ghi ký hiệu của Pre vào tập tin;
```


Ví dụ minh họa

63

- Giả sử các ký tự trong chuỗi cần nén chỉ gồm {a, b}.
  - Trong thực tế, tập ký tự có thể bao gồm cả 256 ký tự ASCII.
- Các ký tự được mã với các con số bắt đầu từ 0.
- Bảng mã ban đầu:

Mã	0	1
Khóa	a	b

Ví dụ minh họa

64

- Việc mã hóa được thực hiện bằng việc quét chuỗi ban đầu từ trái sang phải.
- Tìm chuỗi p dài nhất đã tồn tại trong bảng mã.
- Biểu diễn p bằng mã pCode của nó.
- Tạo chuỗi mới pc với c là ký tự tiếp theo trong chuỗi mã hóa. Thêm chuỗi pc vào trong bảng mã và gán một mã kế tiếp cho chuỗi pc.

Ví dụ minh họa

65

Chuỗi ban đầu = abababbabaabbabbaabba

- p = a
- pCode = 0
- c = b
- Biểu diễn a bằng 0 và thêm ab vào bảng mã.
- Chuỗi mã hóa = 0

code	0	1	2
key	a	b	ab

Ví dụ minh họa

66

- Chuỗi ban đầu = abababbabaabbabbaabba
- Chuỗi mã hóa = 0
- p = b
- pCode = 1
- c = a
- Biểu diễn b bằng 1 và thêm ba vào bảng mã.
- Chuỗi mã hóa = 01

code	0	1	2	3
key	a	b	ab	ba

Ví dụ minh họa

67

- Chuỗi ban đầu = abababbabaabbabbaabba
- Chuỗi mã hóa = 01
- p = ab
- pCode = 2
- c = a
- Biểu diễn ab bằng 2 và thêm aba vào bảng mã.
- Chuỗi mã hóa = 012

code	0	1	2	3	4
key	a	b	ab	ba	aba

Ví dụ minh họa

68

- Chuỗi ban đầu = ababababbabaabbabbaabba
- Chuỗi mã hóa = 012
- p = ab
- pCode = 2
- c = b
- Biểu diễn ab bằng 2 và thêm abb vào bảng mã.
- Chuỗi mã hóa = 0122

code	0	1	2	3	4	5
key	a	b	ab	ba	aba	abb

Ví dụ minh họa

69

- Chuỗi ban đầu = abababbabaabbabbaabba
- Chuỗi mã hóa = 0122
- p = ba
- pCode = 3
- c = b
- Biểu diễn ba bằng 3 và thêm bab vào bảng mã.
- Chuỗi mã hóa = 01223

code	0	1	2	3	4	5	6
key	a	b	ab	ba	aba	abb	bab

Ví dụ minh họa

70

- Chuỗi ban đầu = abababbabaabbabbaabba
- Chuỗi mã hóa = 01223
- p = ba
- pCode = 3
- c = a
- Biểu diễn ba bằng 3 và thêm baa vào bảng mã.
- Chuỗi mã hóa = 012233

code	0	1	2	3	4	5	6	7
key	a	b	ab	ba	aba	abb	bab	baa

Ví dụ minh họa

71

- Chuỗi ban đầu = abababbabaabbabbaabba
- Chuỗi mã hóa = 012233
- p = abb
- pCode = 5
- c = a
- Biểu diễn abb bằng 5 và thêm abba vào bảng mã.
- Chuỗi mã hóa = 0122335

code	0	1	2	3	4	5	6	7	8
key	a	b	ab	ba	aba	abb	bab	baa	abba

Ví dụ minh họa

72

- Chuỗi ban đầu = abababbabaabbabbaabba
- Chuỗi mã hóa = 0122335
- p = abba
- pCode = 8
- c = a
- Biểu diễn abba bằng 8 và thêm abbaa vào bảng mã.
- Chuỗi mã hóa = 01223358

code	0	1	2	3	4	5	6	7	8	9
key	a	b	ab	ba	aba	abb	bab	baa	abba	abbaa

Ví dụ minh họa

73

- Chuỗi ban đầu = abababbabaabbabbaabba
- Chuỗi mã hóa = 01223358
- p = abba
- pCode = 8
- c = null
- Biểu diễn abba bằng 8.
- Chuỗi mã hóa = **012233588**

code	0	1	2	3	4	5	6	7	8	9
key	a	b	ab	ba	aba	abb	bab	baa	abba	abbaa

Giải nén LZW

74

- Giải nén là khôi phục lại dữ liệu gốc từ dữ liệu nén.
- Đưa những ký hiệu thành các chuỗi ban đầu.
- Vừa giải nén vừa hình thành lại bảng mã.
- Giống như quá trình nén, giải nén sử dụng bảng mã ban đầu gồm các chuỗi gồm 1 ký tự.

Mã giả

75

```
string entry;
int currvalue = Đọc vào một giá trị mã;
string prev = Sử dụng bảng mã để giải mã currvalue
Xuất prev;
while (còn dữ liệu để đọc)
{
   currvalue = Đọc vào một giá trị mã;
   entry = Sử dụng bảng mã để giải mã currvalue;
   Xuất entry;
   Thêm (prev + first char of entry) vào bảng mã;
    prev = entry;
 }
```

Ví dụ minh họa

76

- Chuỗi mã hóa = 012233588
- pCode = 0 và p = a.
- Chuỗi được giải mã = **a**

code	0	1
key	a	b

Ví dụ minh họa

77

- Chuỗi mã hóa = 012233588
- pCode = 1 và p = b.
- prev = a cùng với ký tự đầu tiên của p được thêm vào bảng mã.
- Chuỗi được giải mã = a**b**

code	0	1	2
key	a	b	ab

Ví dụ minh họa

78

- Chuỗi mã hóa = 012233588
- pCode = 2 và p = ab.
- prev = b cùng với ký tự đầu tiên của p được thêm vào bảng mã.
- Chuỗi được giải mã = abab

code	0	1	2	3
key	a	b	ab	ba

Ví dụ minh họa

79

- Chuỗi mã hóa = 012233588
- pCode = 2 và p = ab.
- prev = ab cùng với ký tự đầu tiên của p được thêm vào bảng mã.
- Chuỗi được giải mã = ababab.

code	0	1	2	3	4
key	a	b	ab	ba	aba

Ví dụ minh họa

80

- Chuỗi mã hóa = 012233588
- pCode = 3 và p = ba.
- prev = ab cùng với ký tự đầu tiên của p được thêm vào bảng mã.
- Chuỗi được giải mã = ababab**ba**.

code	0	1	2	3	4	5
key	a	b	ab	ba	aba	abb

Ví dụ minh họa

81

- Chuỗi mã hóa = 012233588
- pCode = 3 và p = ba.
- prev = ba cùng với ký tự đầu tiên của p được thêm vào bảng mã.
- Chuỗi được giải mã = abababba**ba**.

code	0	1	2	3	4	5	6
key	a	b	ab	ba	aba	abb	bab

Ví dụ minh họa

82

- Chuỗi mã hóa = 012233588
- pCode = 5 và p = abb.
- prev = ba cùng với ký tự đầu tiên của p được thêm vào bảng mã.
- Chuỗi được giải mã = ababababa**abb**.

code	0	1	2	3	4	5	6	7
key	a	b	ab	ba	aba	abb	bab	baa

Ví dụ minh họa

83

- Chuỗi mã hóa = 012233588
- 8 không tìm thấy trong bảng mã
- **Khi mã không tìm thấy trong bảng mã, khóa của nó là prev cùng với ký tự đầu tiên của prev.**
- prev = abb
- Vì vậy, 8 biểu diễn abba.
- Chuỗi được giải mã = ababababaabb**abba**

code	0	1	2	3	4	5	6	7	8
key	a	b	ab	ba	aba	abb	bab	baa	abba

Ví dụ minh họa

84

- Chuỗi mã hóa = 012233588
- pCode = 8 và p = abba.
- prev = abba cùng với ký tự đầu tiên của p được thêm vào bảng mã.
- Chuỗi được giải mã = ababbabaabbabba**abba**

code	0	1	2	3	4	5	6	7	8	9
key	a	b	ab	ba	aba	abb	bab	baa	abba	abbaa

Đánh giá

85

- Ưu điểm:
  - Hệ số nén cao, không cần kèm theo bảng mã khi nén
  - Có thể dùng để nén nhiều loại tập tin
- Nhược điểm:
  - Tốn nhiều bộ nhớ để tạo từ điển
  - Khó thực hiện trên dữ liệu kích thước nhỏ

Đánh giá

86

Độ phức tạp:

- Nén:
  - O(n) với n là độ dài chuỗi cần nén
- Giải nén:
  - O(n) với n là độ dài dữ liệu cần giải nén

Ứng dụng

87

- Là phương thức nén đầu tiên được sử dụng rộng rãi trên máy tính.
- Là tiện ích trên nền của hệ điều hành Unix.
- Một số tiện ích nén sử dụng LZW hoặc phương pháp của thuật toán nén LZW.
- Trở nên phổ biến khi nó trở thành một phần của định dạng GIF và có thể được sử dụng trong TIFF, PDF.

Các biến thể

88

- LZMW (năm 1985, V. Miller, M. Wegman).
- LZAP (năm 1988, James Storer).
- LZWL là một biến thể âm tiết (syllable-based) dựa trên LZW.

89

Hỏi và Đáp