Cấu trúc dữ liệu và giải thuật

NÉN DỮ LIỆU

Văn Chí Nam – Nguyễn Thị Hồng Nhung – Đặng Nguyễn Đức Tiến – Vũ Thanh Hưng

Nội dung trình bày

- Giới thiệu
- Một số khái niệm
- Nén RLE
- Giải thuật nén Huffman tĩnh
- Giải thuật nén Huffman động

Giới thiệu

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

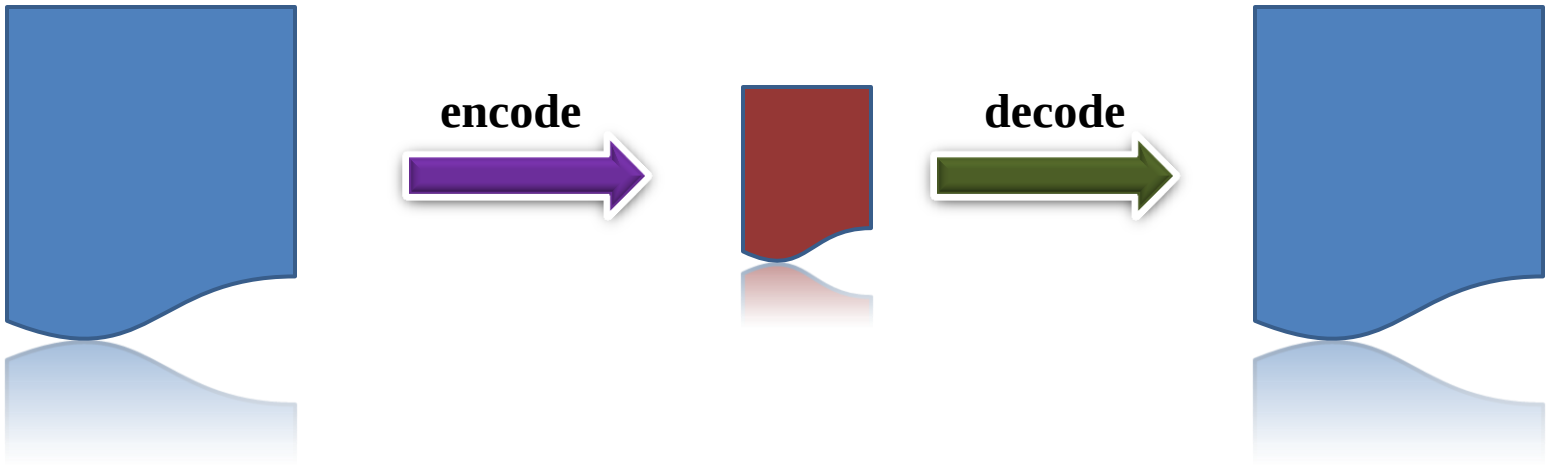
Giới thiệu

- 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

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



Khái niệm

- 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

- 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

- Nén dữ liệu không mất mát thông tin (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

- Nén dữ liệu mất mát thông tin (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

Nén Run-Length Encoding

Giới thiệu

- Một thuật toán nén đơn giản
- Dạng nén không mất mát dữ liệu

Khái niệm

- 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

- 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

- 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 cho phù hợp.

Nén RLE trên PCX

- 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

- 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

- 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

- 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 PCX - Cài đặt


```
#define MAX_RUNLENGTH 63
int PCXEncode_a_String(char *aString, int nLen, FILE *fEncode)
{
   unsigned char cThis, cLast;
   int nTotal = 0; // Tổng số byte sau khi mã hoá
   int nRunCount = 1; // Chiều dài của 1 run
   cLast = *(aString);
   for (int i=0; i<nLen; i++) {
         cThis = *(++aString);
         if (cThis == cLast) { // Tồn tại 1 run
                  nRunCount++;
                  if (nRunCount == MAX_RUNLENGTH) {
                  nTotal += PCXEncode_a_Run(cLast,nRunCount,fEncode);
                  nRunCount = 0;
                  }
         }
```

Nén RLE trên PCX - Cài đặt


```
        else // Hết 1 run, chuyển sang run kế tiếp
        {
        if (nRunCount)
        nTotal += PCXEncode_a_Run(cLast,nRunCount,fEncode);
        cLast = cThis;
        nRunCount = 1;
        }
    } // end for
    if (nRunCount) // Ghi run cuối cùng lên file
    nTotal += PCXEncode_a_Run(cLast, nRunCount, fEncode);
    return (nTotal);
}
```

Nén RLE trên PCX - Cài đặt


```
int PCXEncode_a_Run(unsigned char c, int nRunCount, FILE
   *fEncode)
{
   if (nRunCount) {
        if ((nRunCount == 1) && (c < 192)) {
              putc(c, fEncode);
              return 1;
        }
   else {
        putc(0xC0 | nRunCount, fEncode);
        putc(c, fEncode);
        return 2;
   }
}
```

Nén RLE trên BMP

- 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

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

Nén RLE trên BMP

- 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

- 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>

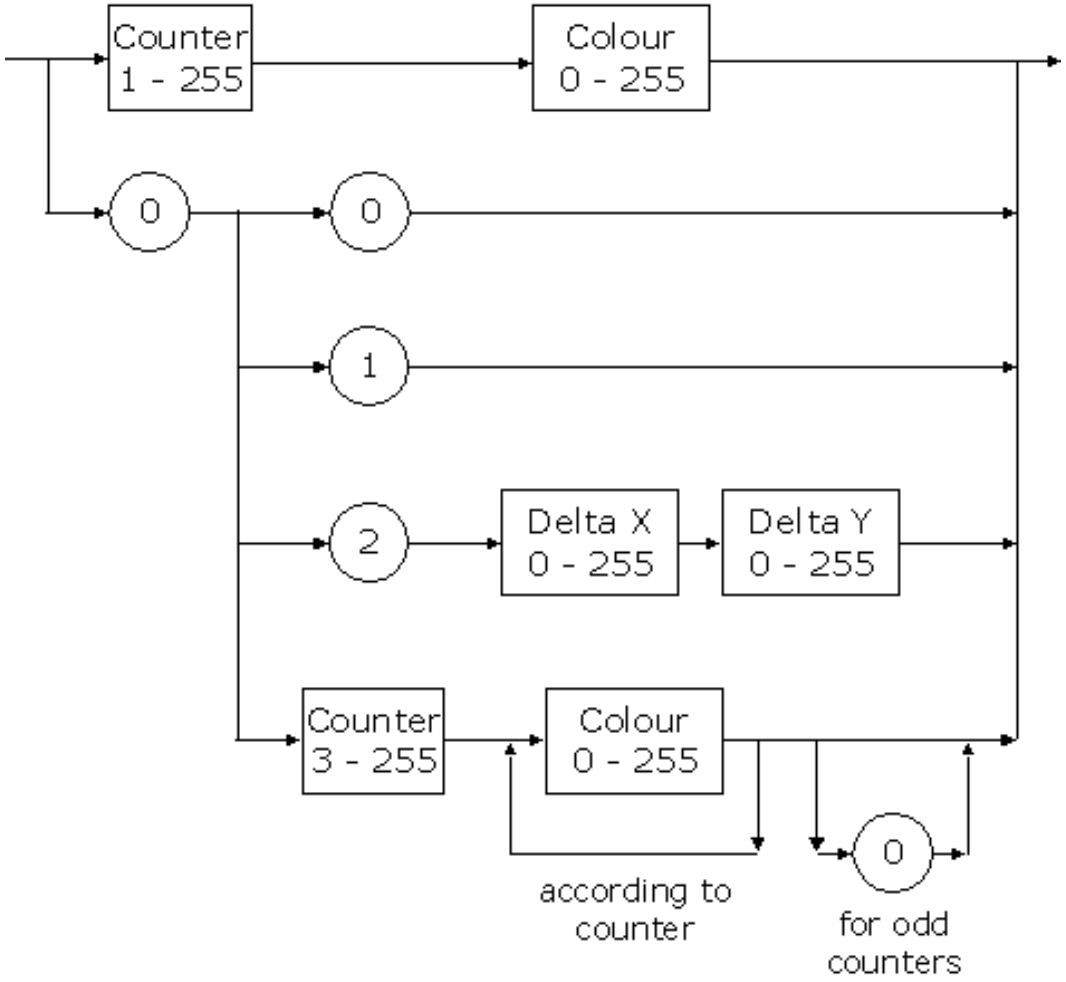
    - Ký tự đánh dấu: **0x00**
    - Dùng trong trường hợp dãy có từ 3 ký tự riêng lẻ trở lên.
    - Ví dụ:

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

- 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

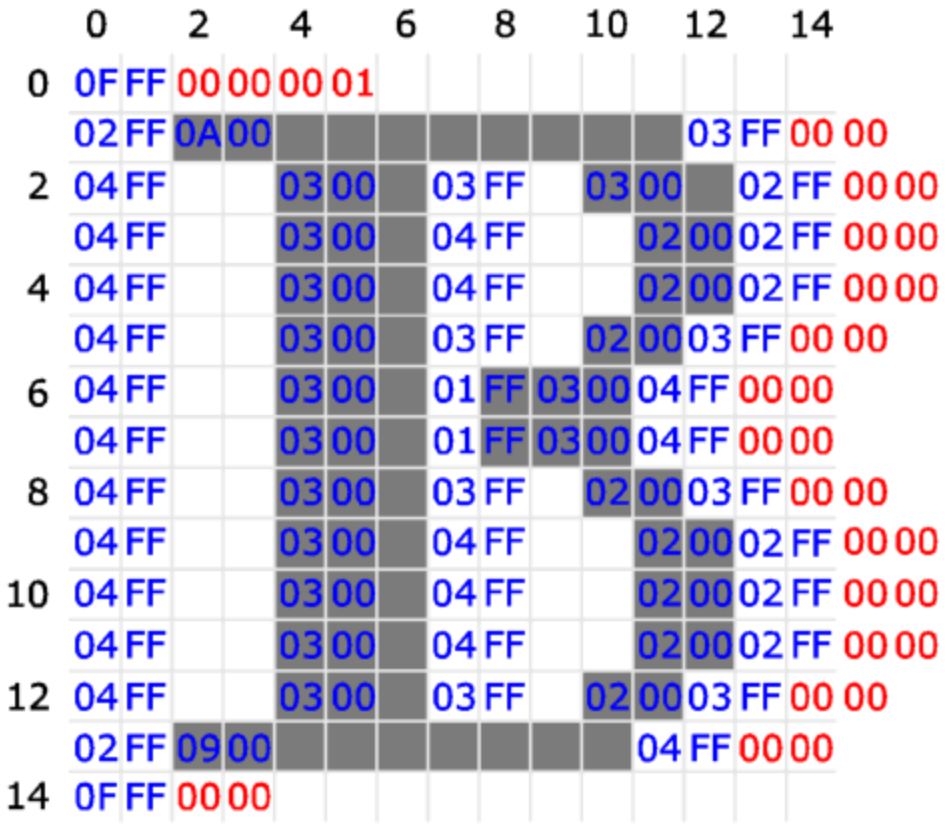


Nén RLE trên BMP – Ví dụ

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 09 01

Nén RLE trên BMP – Ví dụ

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



So sánh

So sánh giữa RLE trên PCX và trên BMP?

Nhận xét

- 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

Bài tập Nén RLE

- S = a a a A A B c d D D D D b
- RLE
- RLE_PCX(HEX)
- RLE_BMP(HEX)

Nén Huffman tĩnh

Giới thiệu

- 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

- 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

- 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

- 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

[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

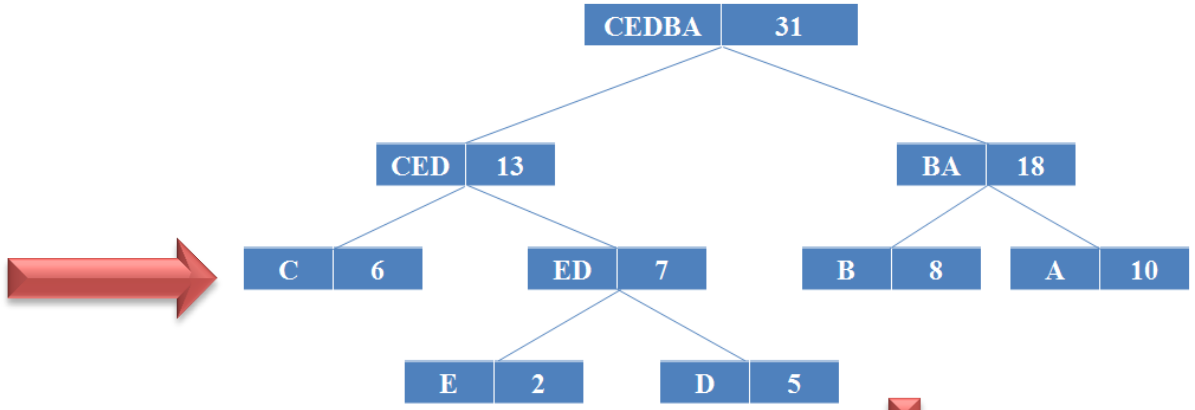
[B4]: Duyệt tập tin -> Thay thế các ký tự trong tập tin bằng mã bit tương ứng.

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

Thuật toán nén

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

11011011111110100000101111111010000
00010101000011111101101101001011111

Thuật toán nén – Thống kê tần số

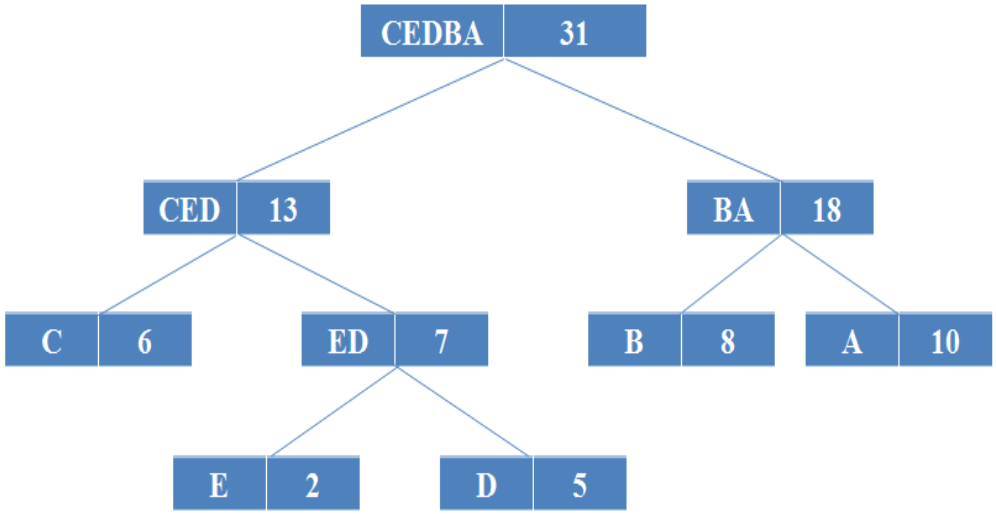
- Dữ liệu:

ADDAABBCCBAAABBCCCBBBCDAADDEEAA

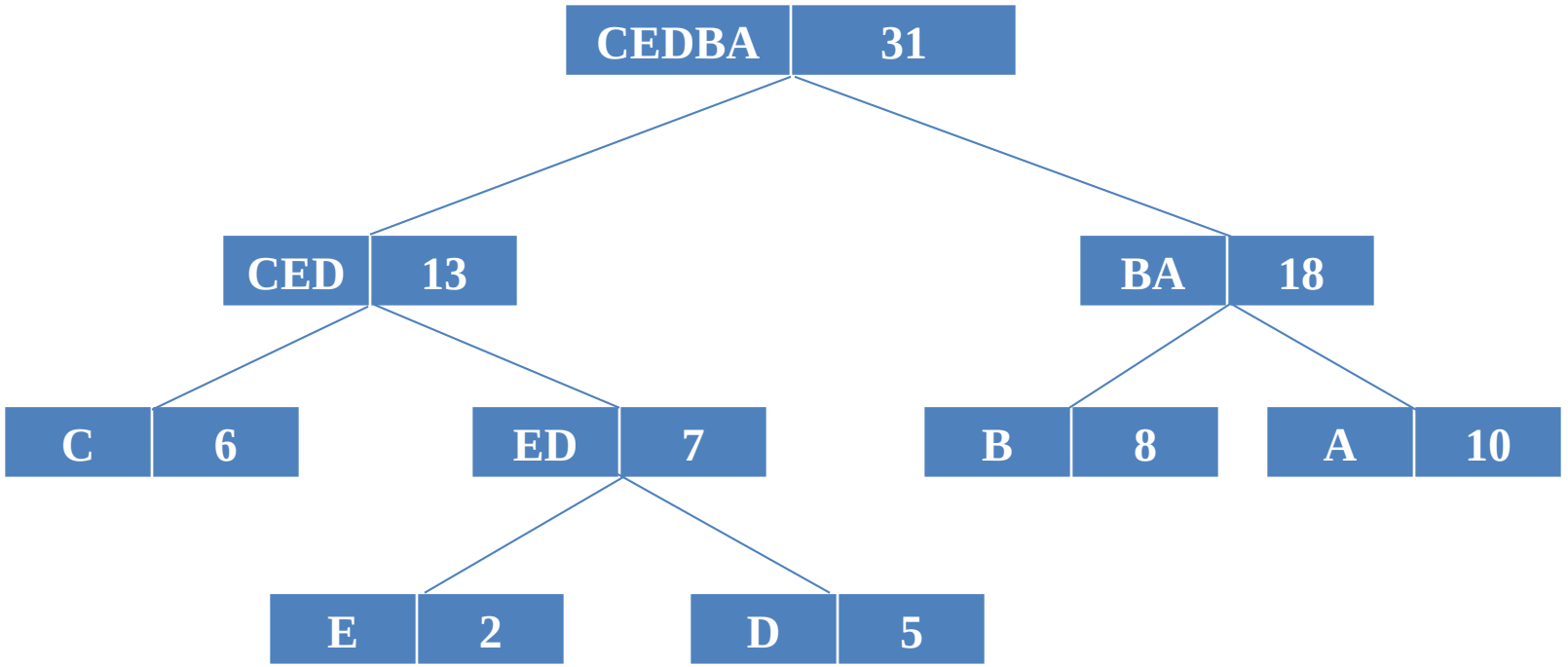
Ký tự	Tần số xuất hiện
A	10
B	8
C	6
D	5
E	2

Thuật toán nén – Tạo cây Huffman

- 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



Thuật toán nén – Tạo cây Huffman

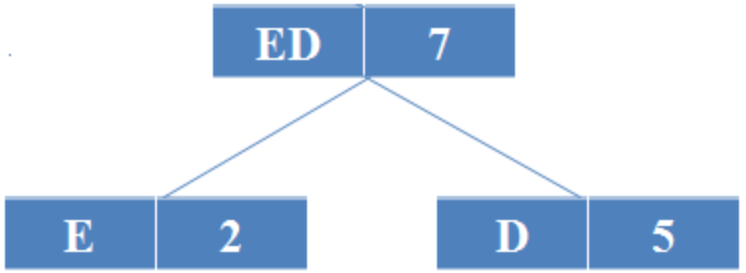
- Phát sinh cây:
  - Bước 1: Chọn trong bảng thống kê hai phần tử ***x***,***y*** có trọng số thấp nhất.
  - Bước 2: Tạo 2 node của cây cùng với node cha ***z*** có trọng số bằng tổng trọng số của hai node con.
  - Bước 3: Loại 2 phần tử ***x***,***y*** ra khỏi bảng thống kê.
  - Bước 4: Thêm phần tử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

- 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.

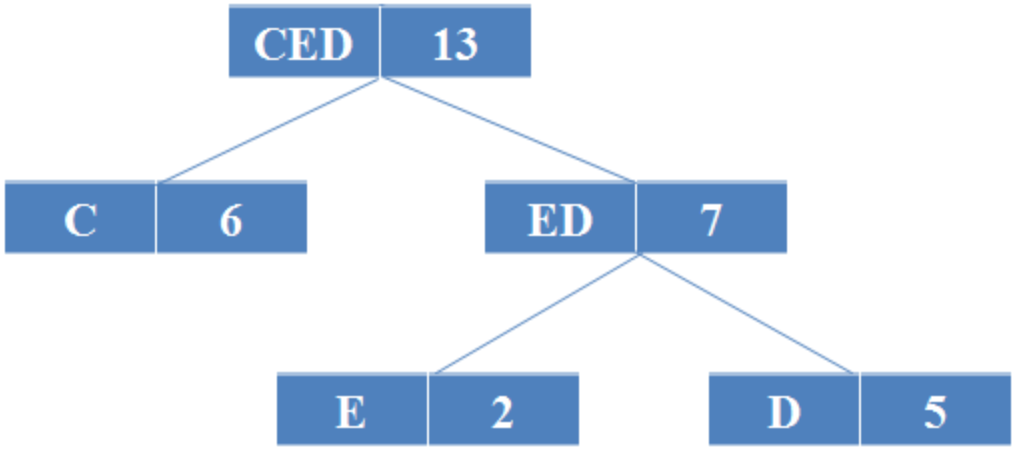
Thuật toán nén – Tạo cây Huffman

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



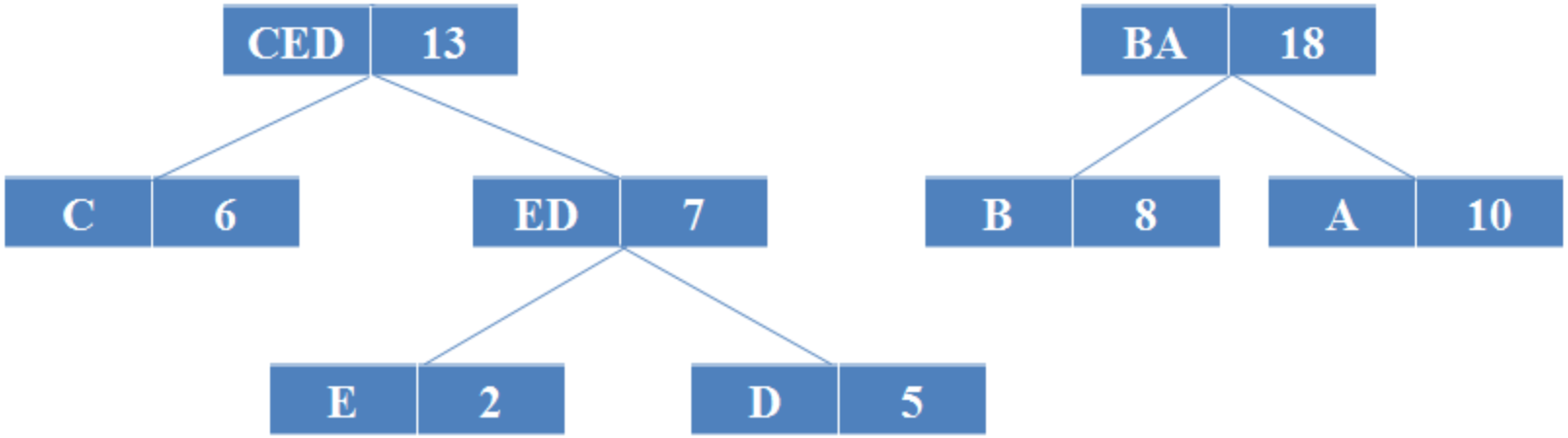
Thuật toán nén – Tạo cây Huffman

Ký tự	Tần số
A	10
B	8
ED	7
C	6



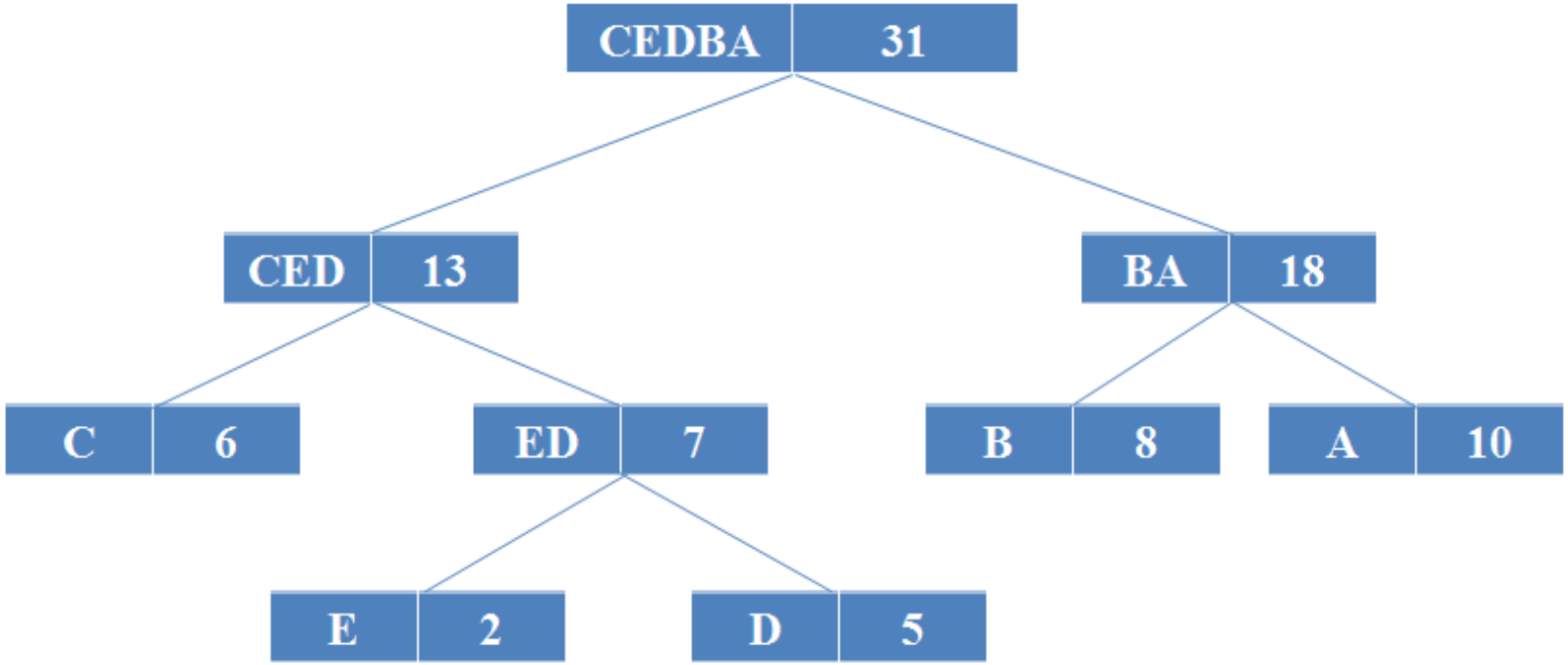
Thuật toán nén – Tạo cây Huffman

Ký tự	Tần số
CED	13
A	10
B	8



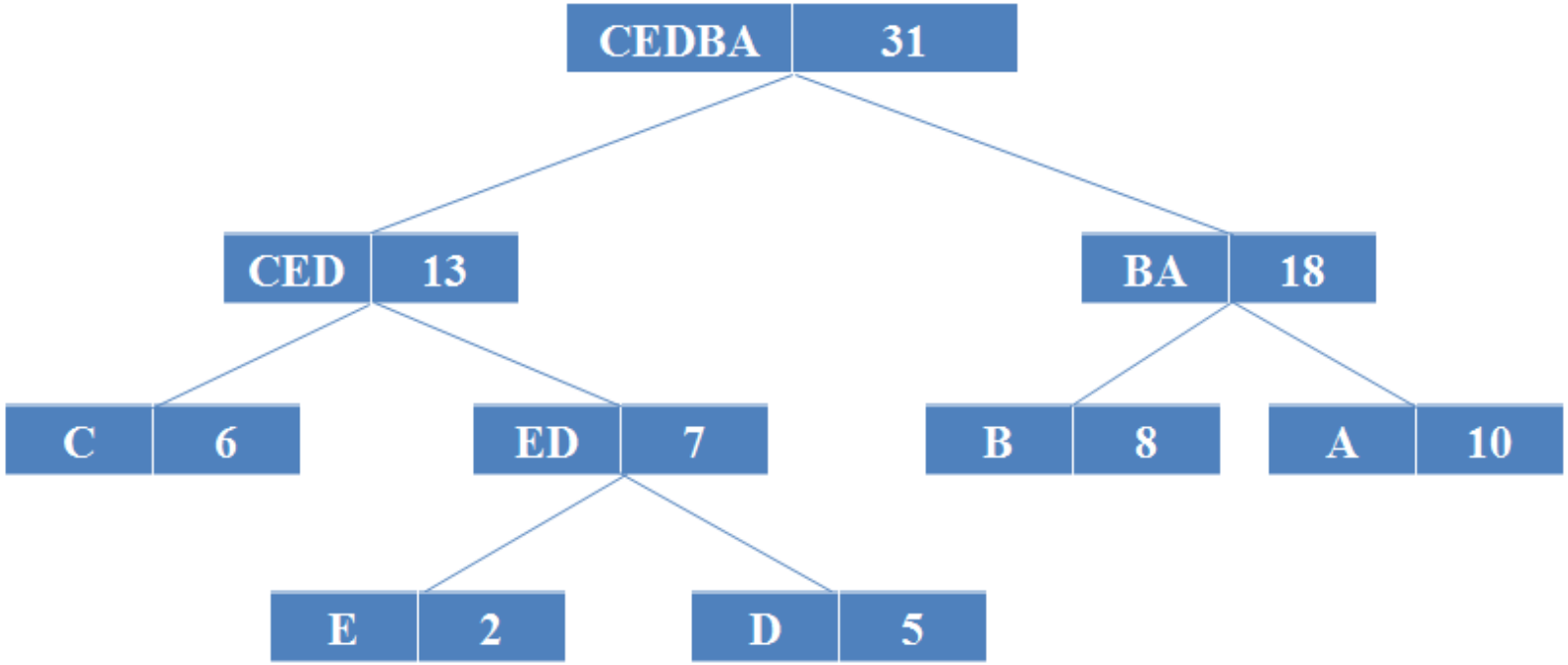
Thuật toán nén – Tạo cây Huffman

Ký tự	Tần số
BA	18
CED	13



Thuật toán nén – Tạo cây Huffman

Ký tự	Tần số
CEDBA	31



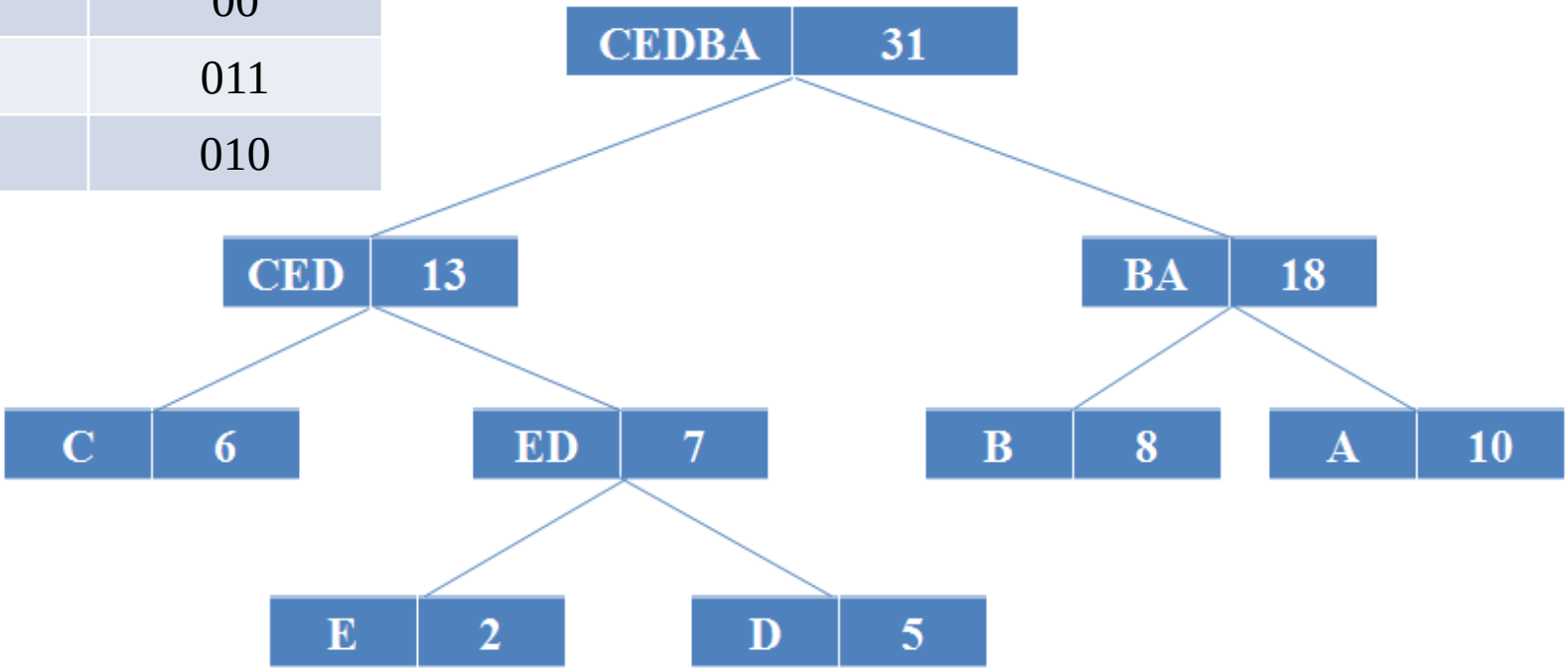
Thuật toán nén – Phát sinh mã bit

- 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

Ký tự	Mã
A	11
B	10
C	00
D	011
E	010



Thuật toán nén – Nén dữ liệu

- 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

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

Thuật toán giải nén

- Phục hồi cây Huffman dựa trên thông tin đã lưu trữ.

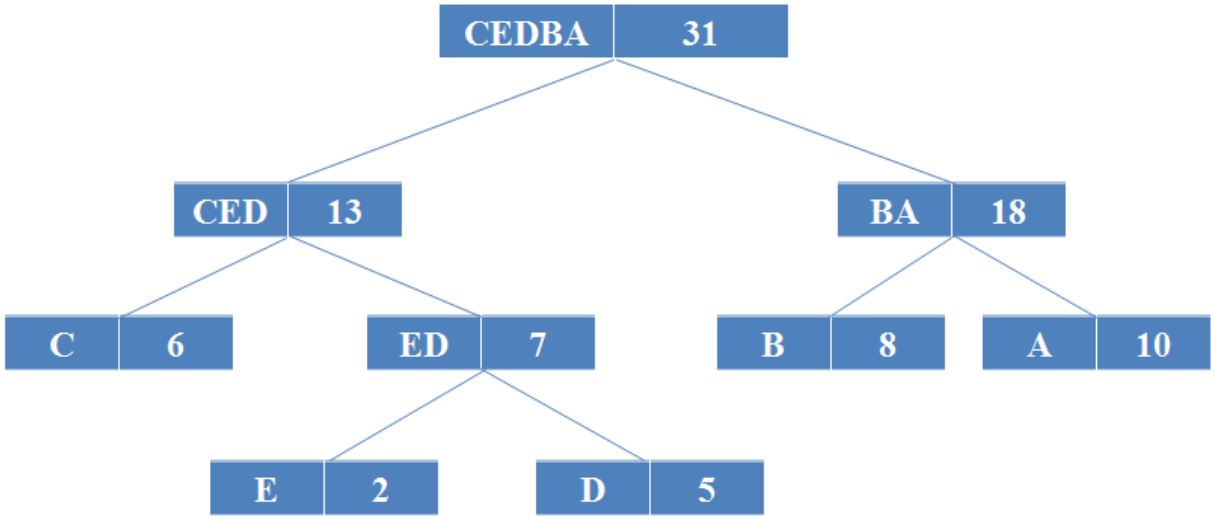
- Lặp
  - Đi từ gốc cây Huffman
  - Đọc từng bit từ tập tin đã được nén
    - Nếu bit 0: đi qua nhánh trái
    - Nếu bit 1: đi qua nhánh phải
    - Nếu đến node lá: xuất ra ký tự tại node lá này.
- Cho đến khi nào hết dữ liệu

Giải nén

11011011111110100000101111111010000000101010000111111011011010100101111

ADDAABBCCBAAABBCCCBBBCDAADDEEAA

Ký tự	Mã
A	11
B	10
C	00
D	011
E	010



Vấn đề khác

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

Vấn đề khác

- 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

Bài tập

- Nén chuỗi sau dùng Static Huffman

  ZWHZZAAFFAZZZAAFFFAAAFAZZWWHHZZH
- Giải nén chuỗi bit sau:

  0111000101110

Bài giải

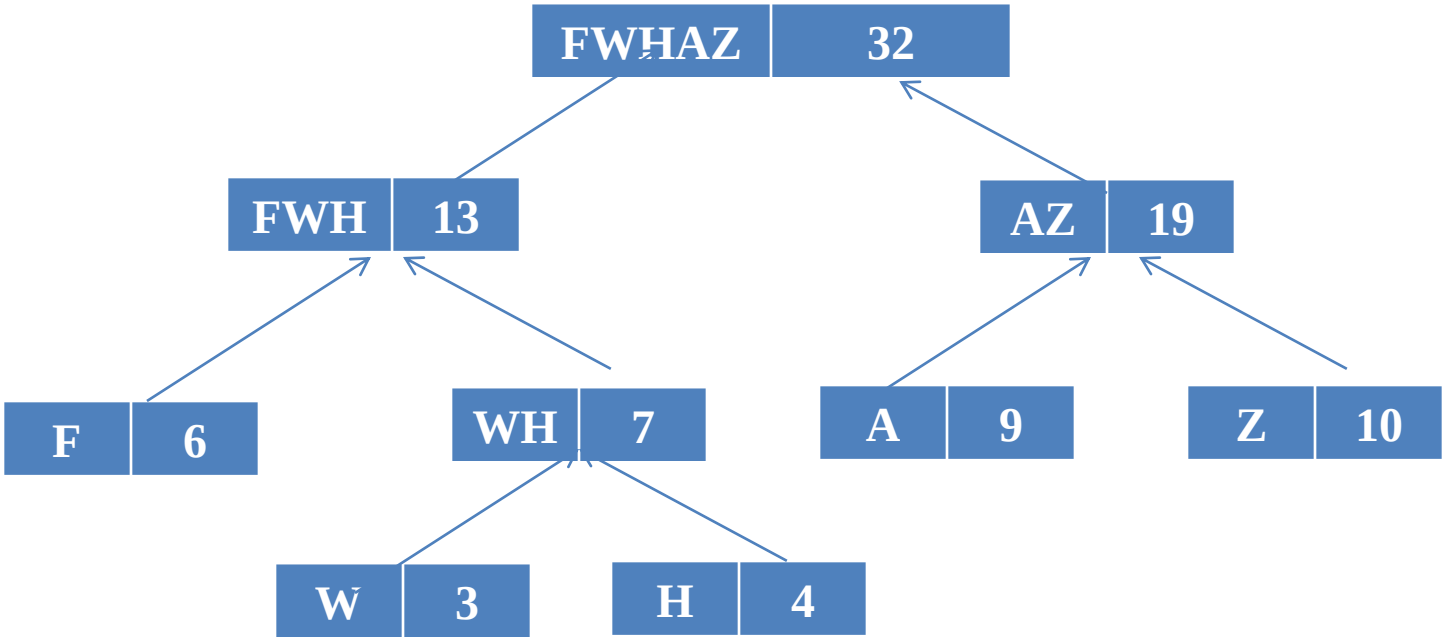
- ZWHZZAAFFAZZZAAFFFAAAFAZZWWHHZZH
- A: 9, Z: 10, W:3, H:4, F:6
- Z:11

  A:10

  F:00

  H:011

  W:010



Nén Huffman động

Điểm hạn chế của nén Huffman tĩnh

- Duyệt tập tin hai lần (thống kê và mã hóa) -> tốn chi phí.
- Phải lưu trữ cây Huffman/bảng tần số trong dữ liệu nén -> tăng kích thước dữ liệu nén.
- Chỉ sử dụng được trong trường hợp dữ liệu cần nén đã có sẵn đầy đủ.

Ý tưởng cải tiến

- Thích nghi: Adaptive Huffman Compression

- Vừa nhận/đọc dữ liệu (cần nén) vừa xây dựng cây Huffman và nén dữ liệu

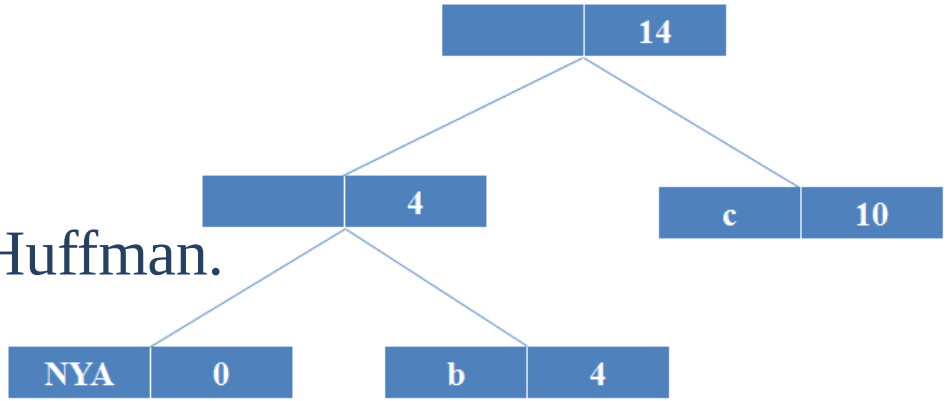
  => nén dữ liệu ĐỘNG.

Phân tích

- Ưu điểm:
  - Có thể tiến hành nén dữ liệu theo thời gian thực.
  - Không cần lưu trữ thông tin cây Huffman.
  - Chỉ cần việc đọc dữ liệu một lần.

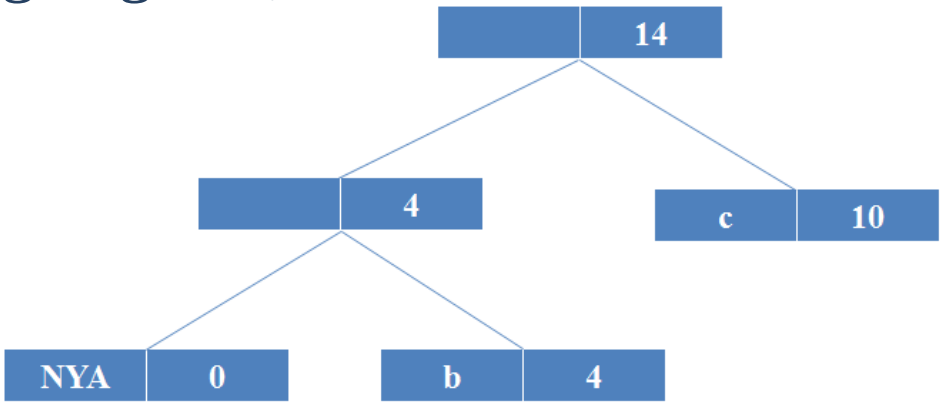
Một số quy ước

- Trọng số của node
  - Node lá: Tần số xuất hiện của ký tự (mà node đại diện) tính đến thời điểm được xem xét.
  - Node trong: tổng trọng số của các node con.
  - Ký tự chưa từng xuất hiện nằm chung một node có trọng số 0.
- Node gốc (root):
  - Node có trọng số lớn nhất cây Huffman.



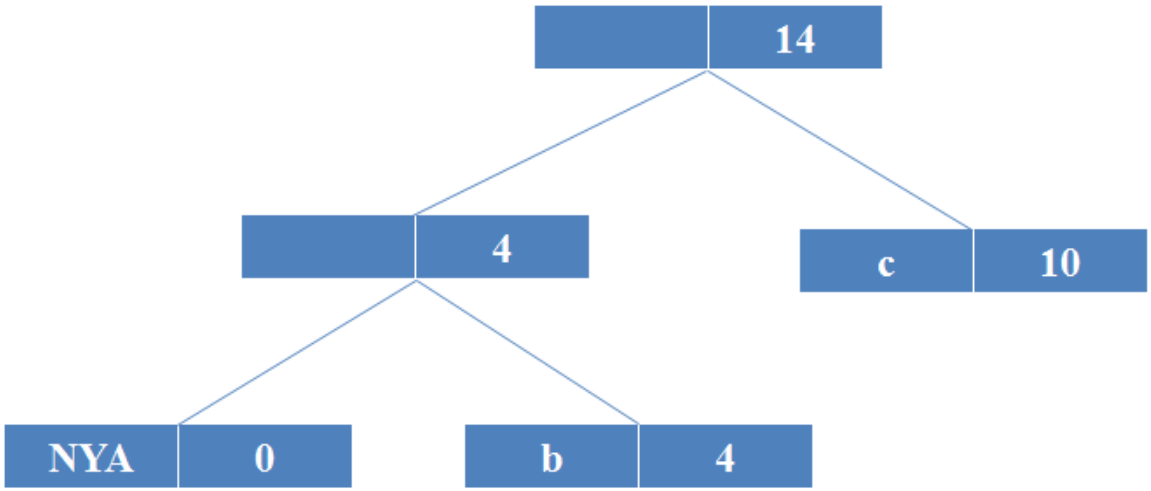
Một số quy ước

- Ký tự điều khiển:
  - Là một ký tự đặc biệt, ký hiệu **NYA** (Not Yet Available).
  - Dùng cho node có trọng số 0 (chứa các ký tự chưa tồn tại trên cây tính đến thời điểm được xem xét.)
- Khởi tạo cây:
  - Cây gồm duy nhất một node gán giá trị NYA và có trọng số là 0.



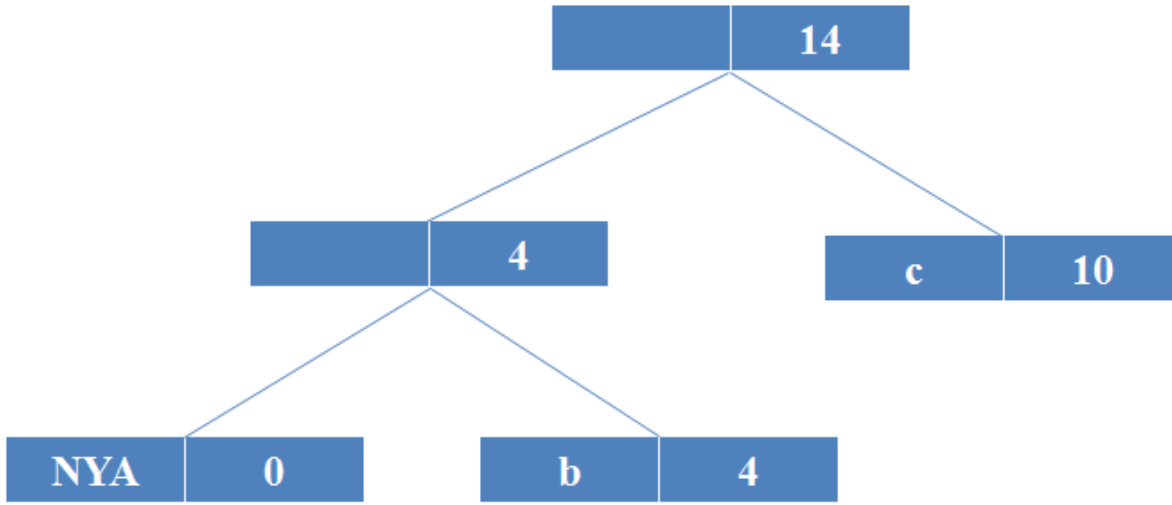
Một số quy ước

- Tính chất Anh/em:
  - Mỗi node trên cây (trừ node gốc) đều có node anh/em (cùng node cha).
  - Khi sắp xếp trọng số của các node theo chiều tăng dần thì các cặp node anh/em luôn đứng liền kề nhau.

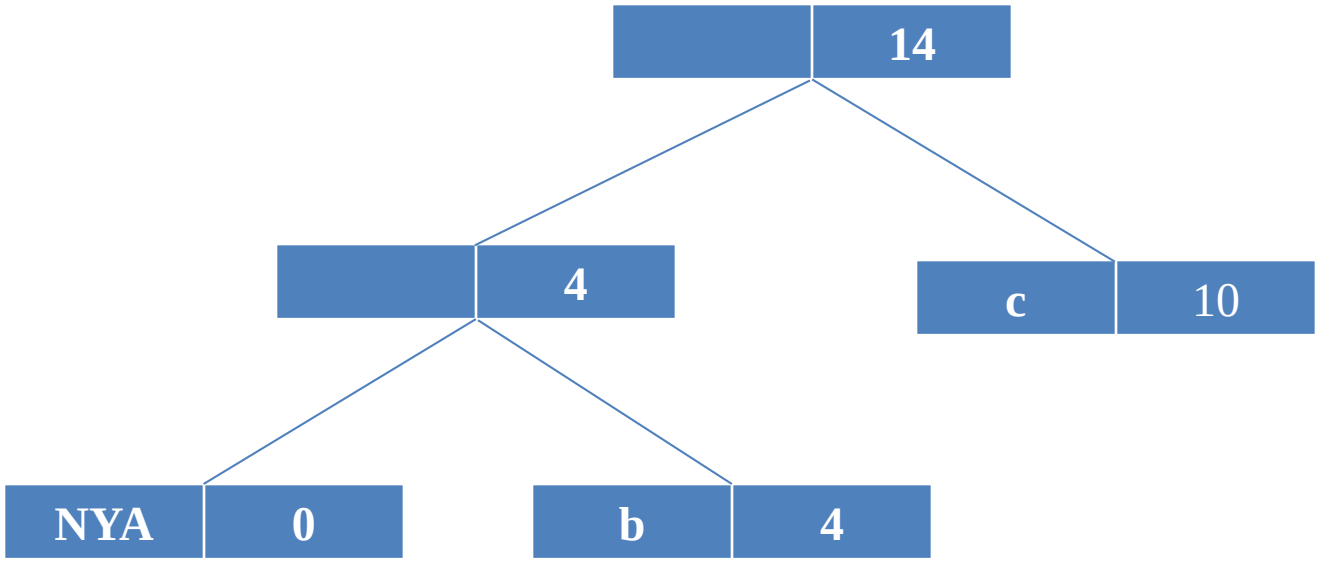


Tính chất cây Huffman động

- Cây Huffman ***n*** node lá có tổng cộng (2**n* – 1) node
- Các node trên cây thỏa mãn tính chất Anh/em.
- Mỗi node trên cây có trọng số thỏa mãn tính chất quy ước.



Ví dụ cây Huffman động



Thuật toán nén

- B1: Khởi tạo cây Huffman ban đầu (1 node NYA).
- B2: Còn có thể đọc được dữ liệu. Đọc ký tự ***c*** cần mã hóa
- B3: Mã hóa ký tự ***c***
- B4: Cập nhật cây Huffman.
- B5: Quay lại từ bước 2.

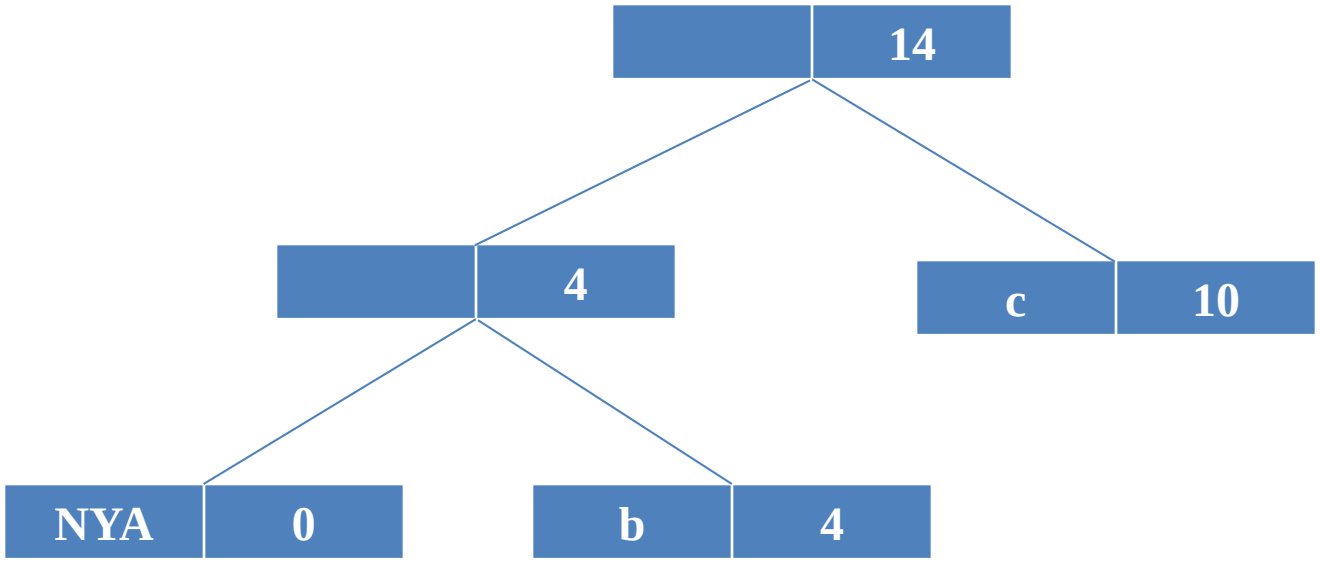
Thuật toán nén

- Cách thức mã hóa ký tự ***c***:
  - Kiểm tra ***c*** có trong cây Huffman chưa?
    - Nếu có:
      - Phát sinh mã bit cho ***c*** (dựa trên cây Huffman theo cách thông thường) để mã hóa cho ký tự ***c***.
    - Nếu chưa có:
      - Phát sinh mã bit cho ***c***:

      Mã bit ***c*** = mã bit của node NYA và mã ASCII (8 bit) của ***c***.

Thuật toán nén

- Ví dụ Mã hóa ký tự:



Mã hóa ký tự ***b***: **01**
Mã hóa ký tự ***c***: **1**

Thuật toán nén

- Cách thức cập nhật cây Huffman khi thêm ký tự ***c*** vào cây:
  - Kiểm tra ***c*** có tồn tại trên cây Huffman chưa?
    - Nếu có:
      - Tăng trọng số của node ***c*** thêm **1**.
    - Nếu chưa có:
      - Tách node **NYA** (cũ) thành hai node: **NYA** và node ***c*** (có trọng số là **1**).
  - Tăng trọng số của các node trên đường đi từ node gốc đến node ***c*** thêm **1**.
  - Xử lý vi phạm tính chất anh/em -> hiệu chỉnh cây.

Thuật toán nén

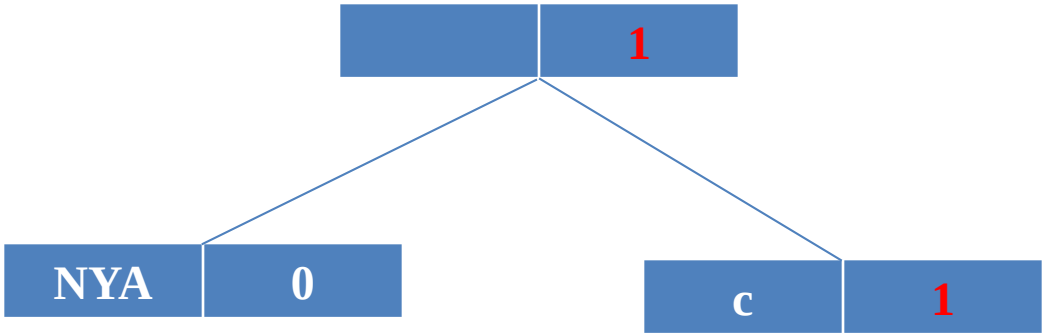
- Khởi tạo cây ban đầu:

NYA	0

Cập nhật cây với dãy ký tự ***ccb***

Thuật toán nén

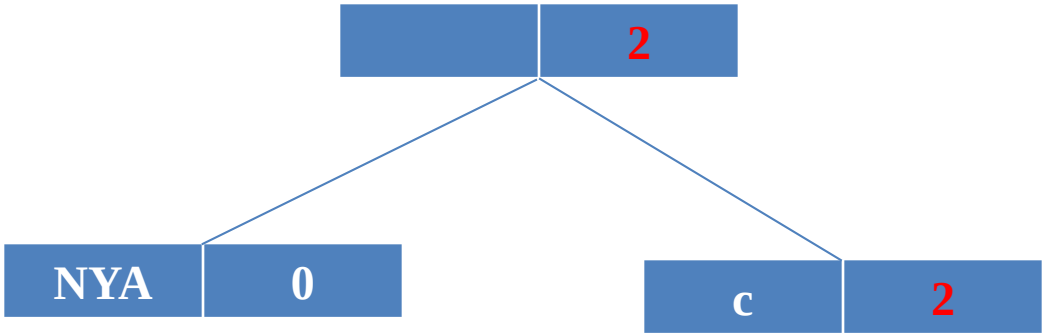
- Cập nhật khi thêm vào ký tự ***c***:



Cập nhật cây với dãy ký tự ***ccb***

Thuật toán nén

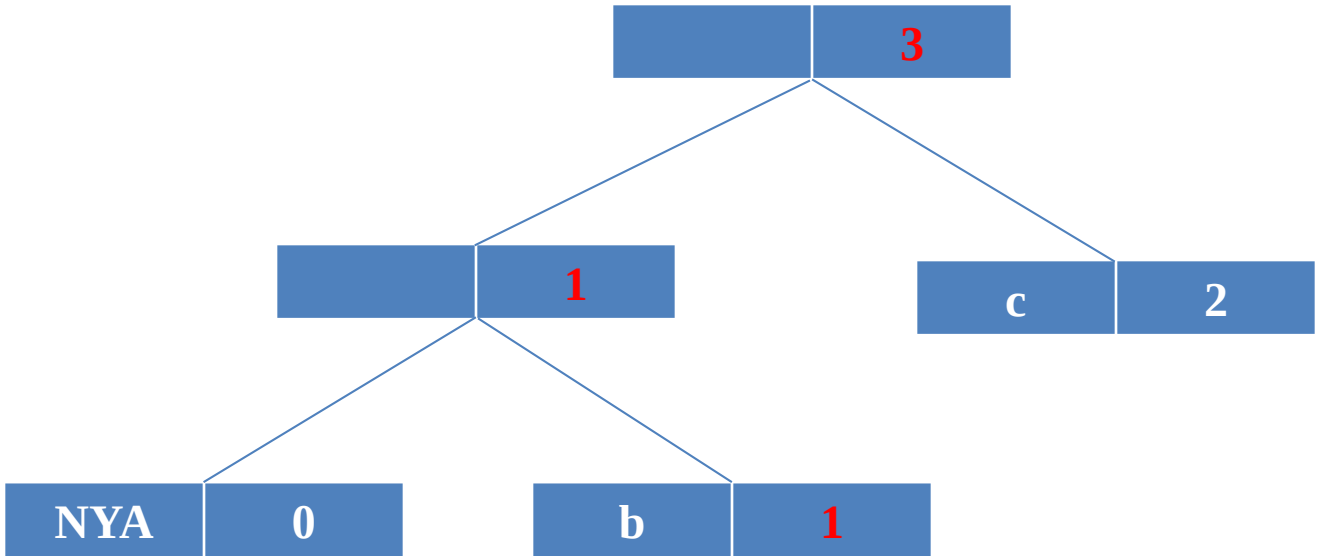
- Cập nhật khi thêm vào ký tự ***c***:



Cập nhật cây với dãy ký tự ***ccb***

Thuật toán nén

- Cập nhật khi thêm vào ký tự ***b***:



Cập nhật cây với dãy ký tự ***ccb***

Thuật toán nén

- Xác định node vi phạm tính chất Anh/em:
  - Gọi ***x*** là node hiện hành.
  - Duyệt từ trái sang phải, từ dưới lên trên:

    Nếu tồn tại node ***y*** sao cho trọng số của ***y*** nhỏ hơn trọng số của ***x***

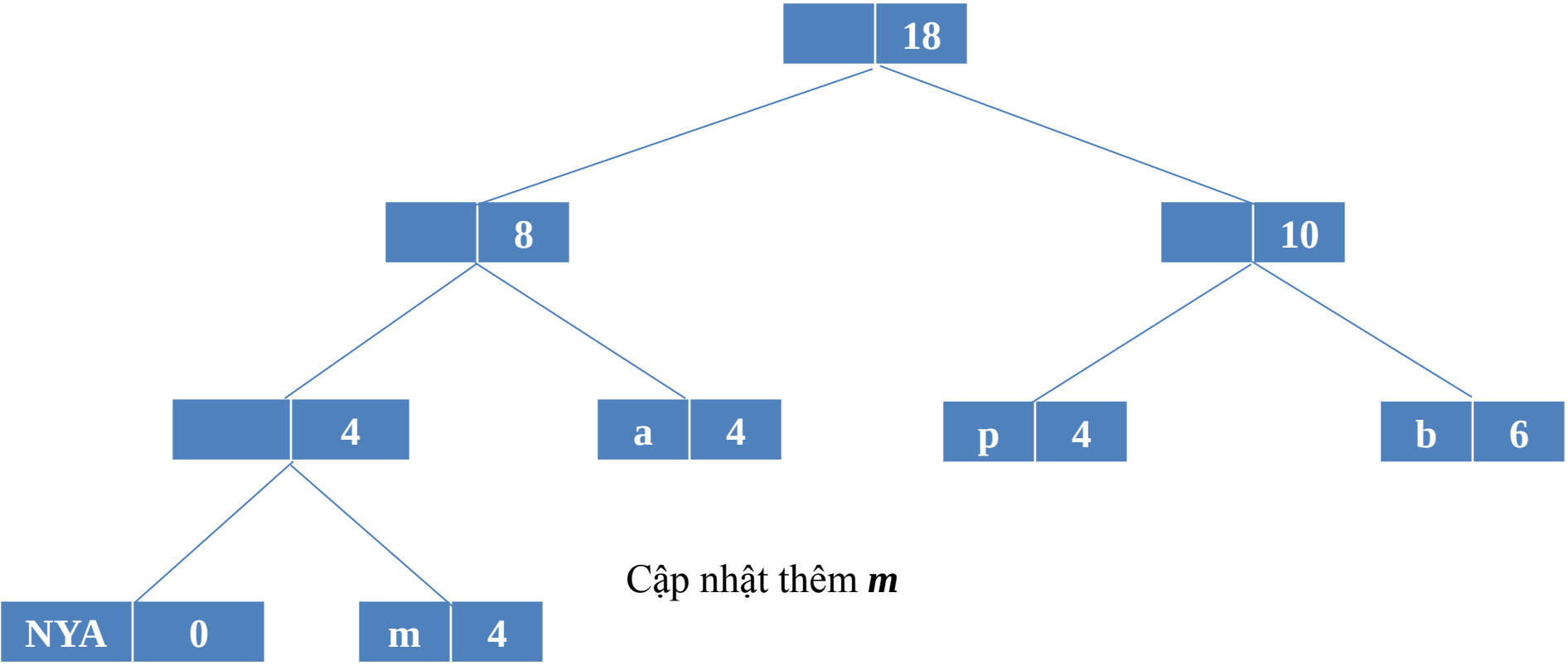
    Thì ***x*** là node vi phạm tính chất Anh/em.

Thuật toán nén

- Hiệu chỉnh cây khi node vi phạm tính chất Anh/em:
  - Gọi ***x*** là node vi phạm tính chất Anh/em.
  - Duyệt từ trái sang phải, từ dưới lên trên:

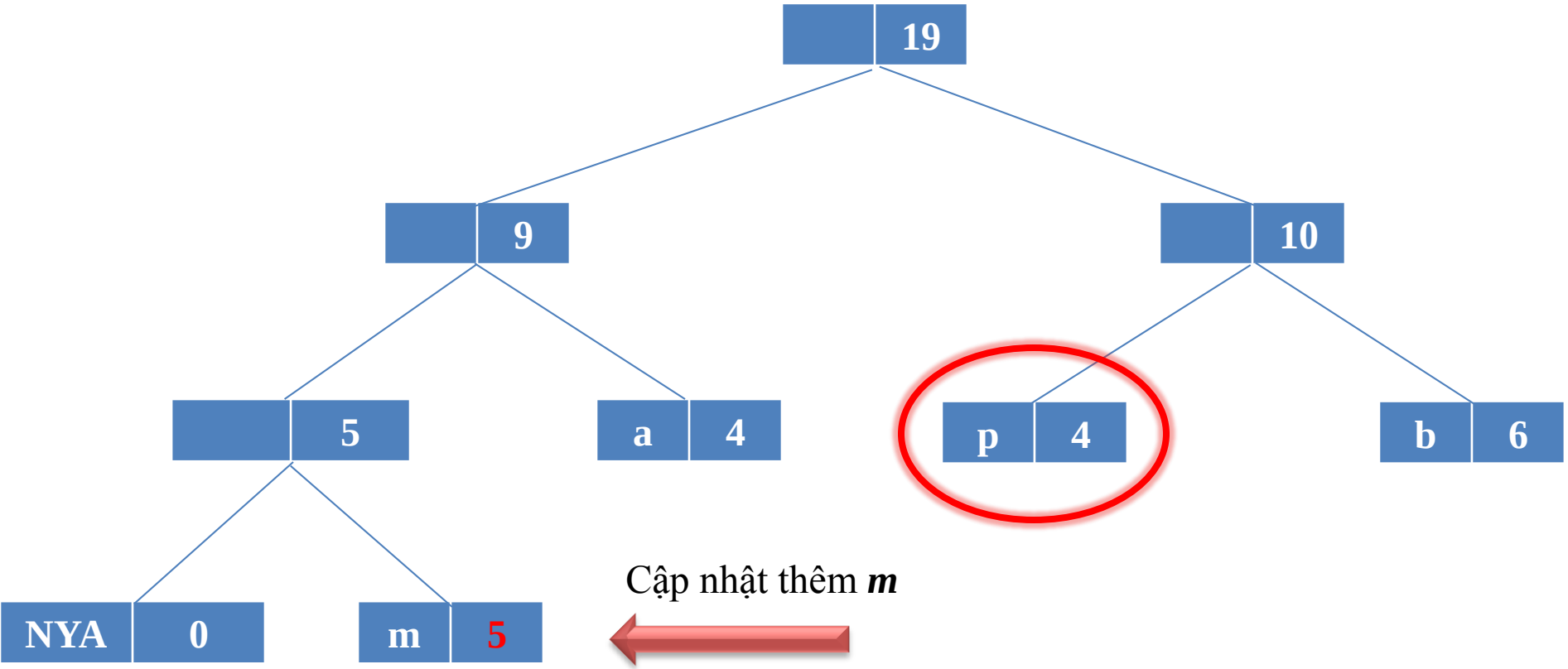
    Tìm node ***y*** ở xa ***x*** nhất có trọng số nhỏ hơn trọng số của ***x.***
  - Đổi chỗ ***x*** và ***y***.
  - Cập nhật giá trị các node cha tương ứng.
  - Lặp lại cho đến khi không còn node nào vi phạm.

Thuật toán nén

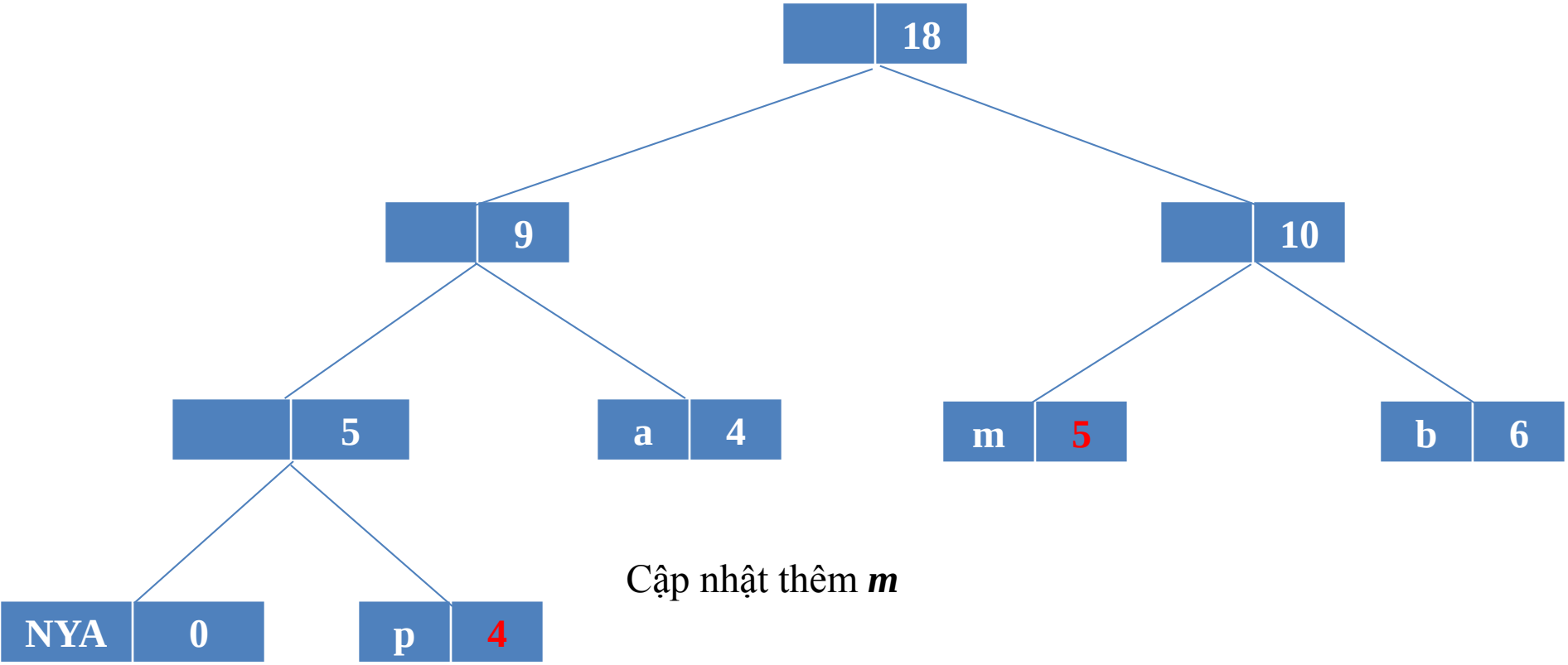


Cập nhật thêm ***m***

Thuật toán nén

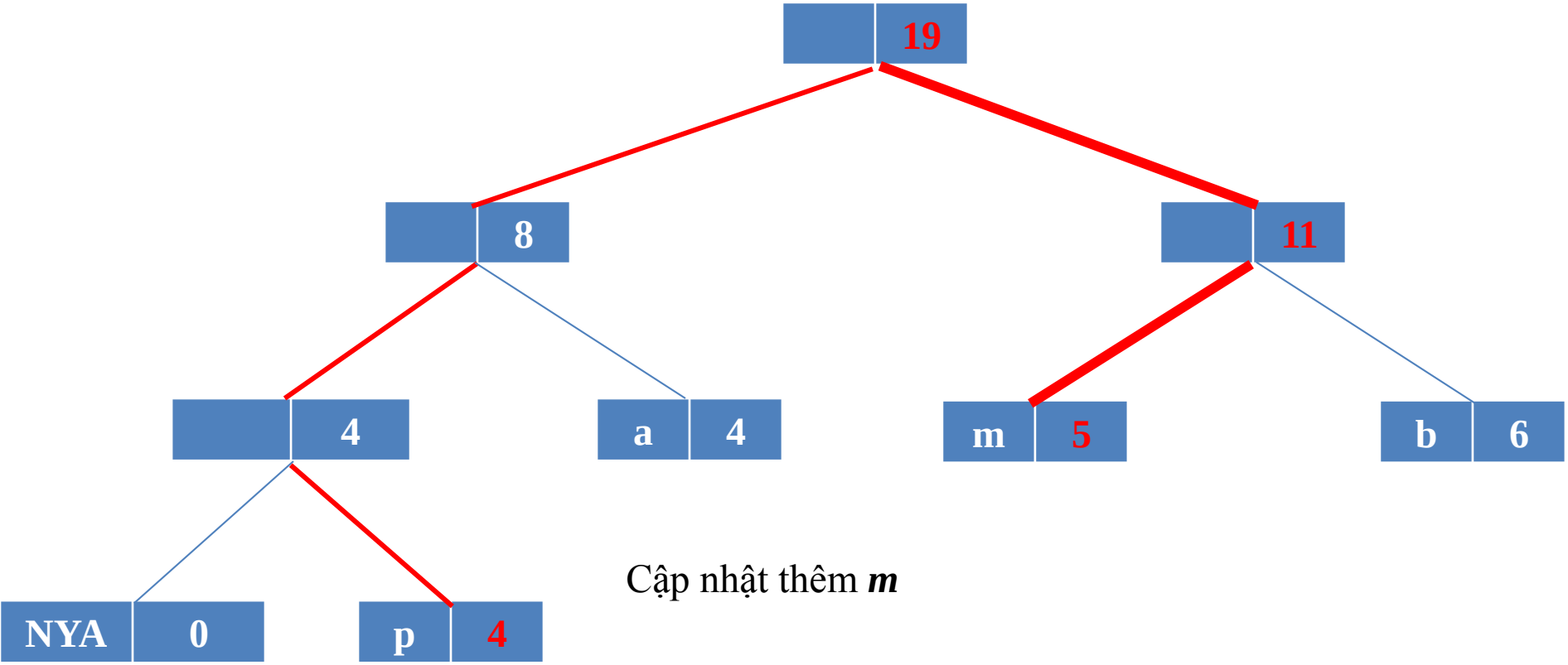


Thuật toán nén



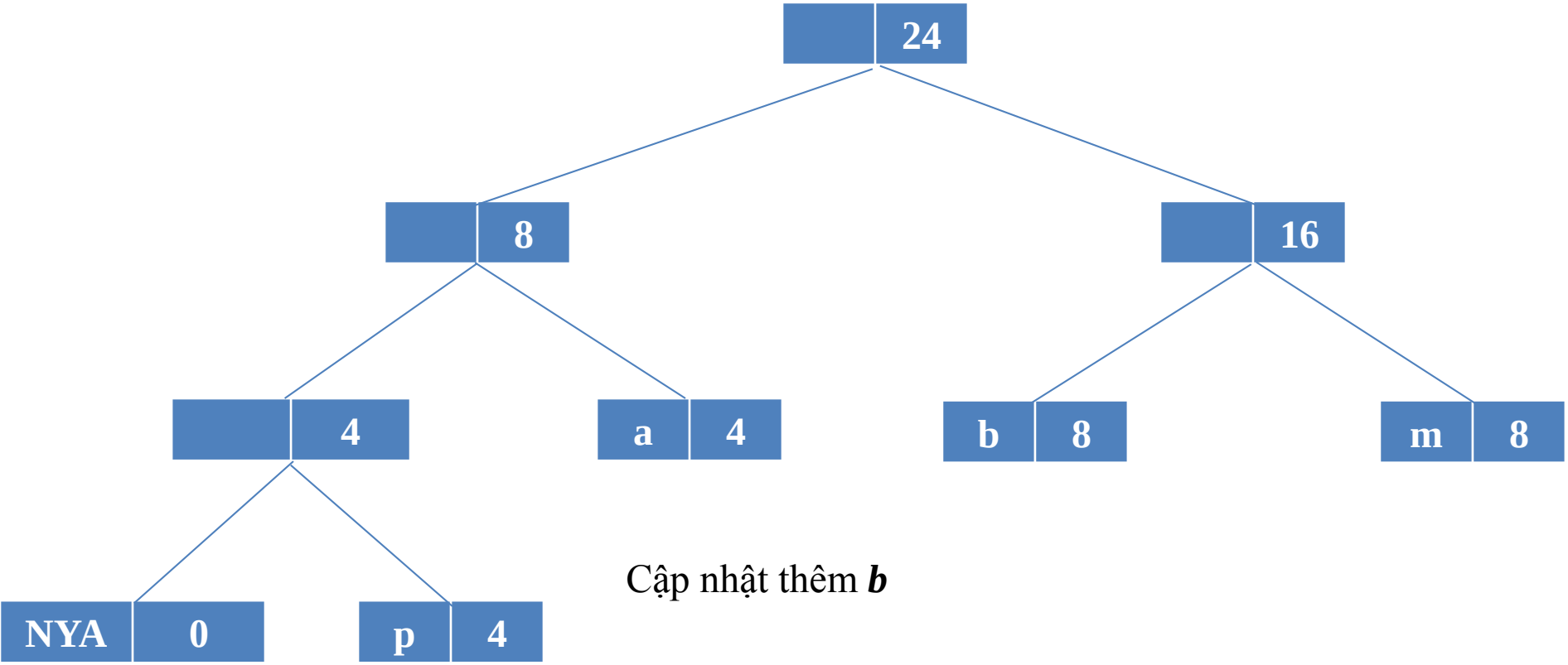
Cập nhật thêm ***m***

Thuật toán nén



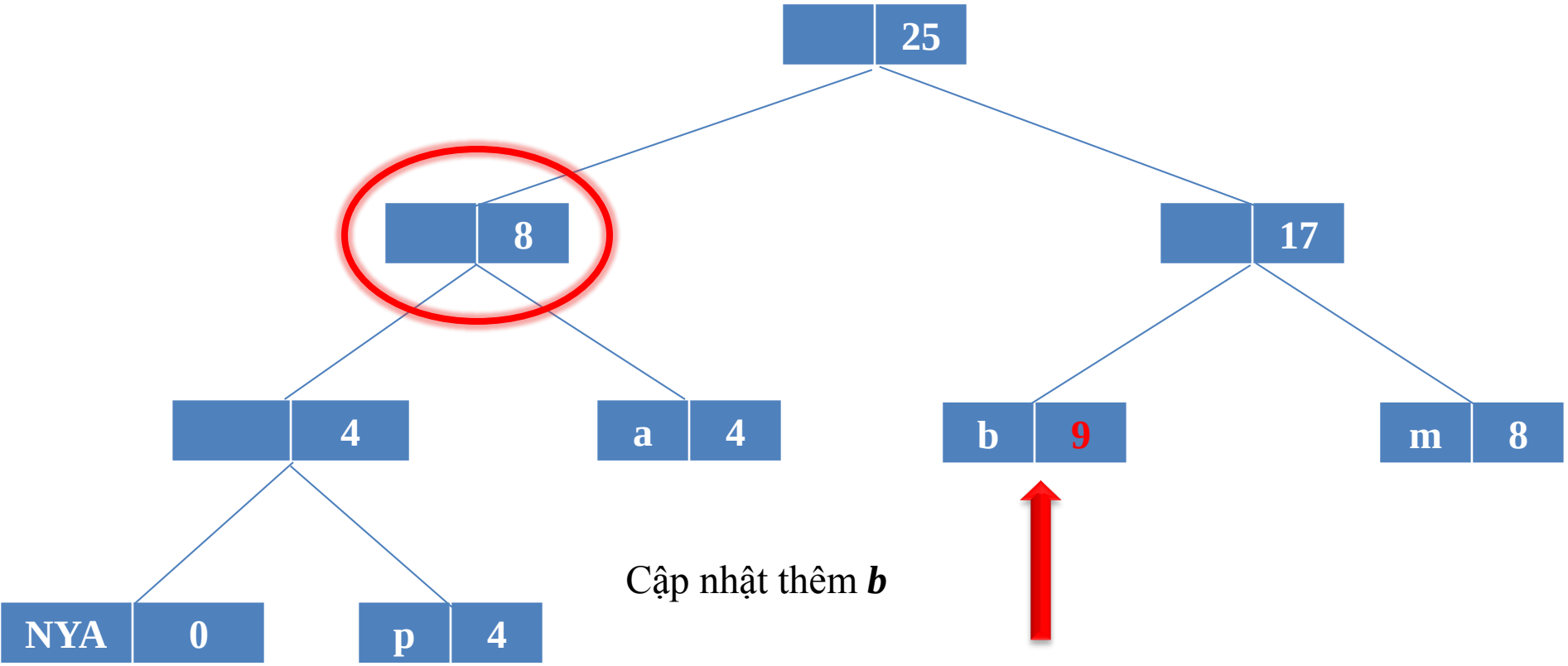
Cập nhật thêm ***m***

Thuật toán nén

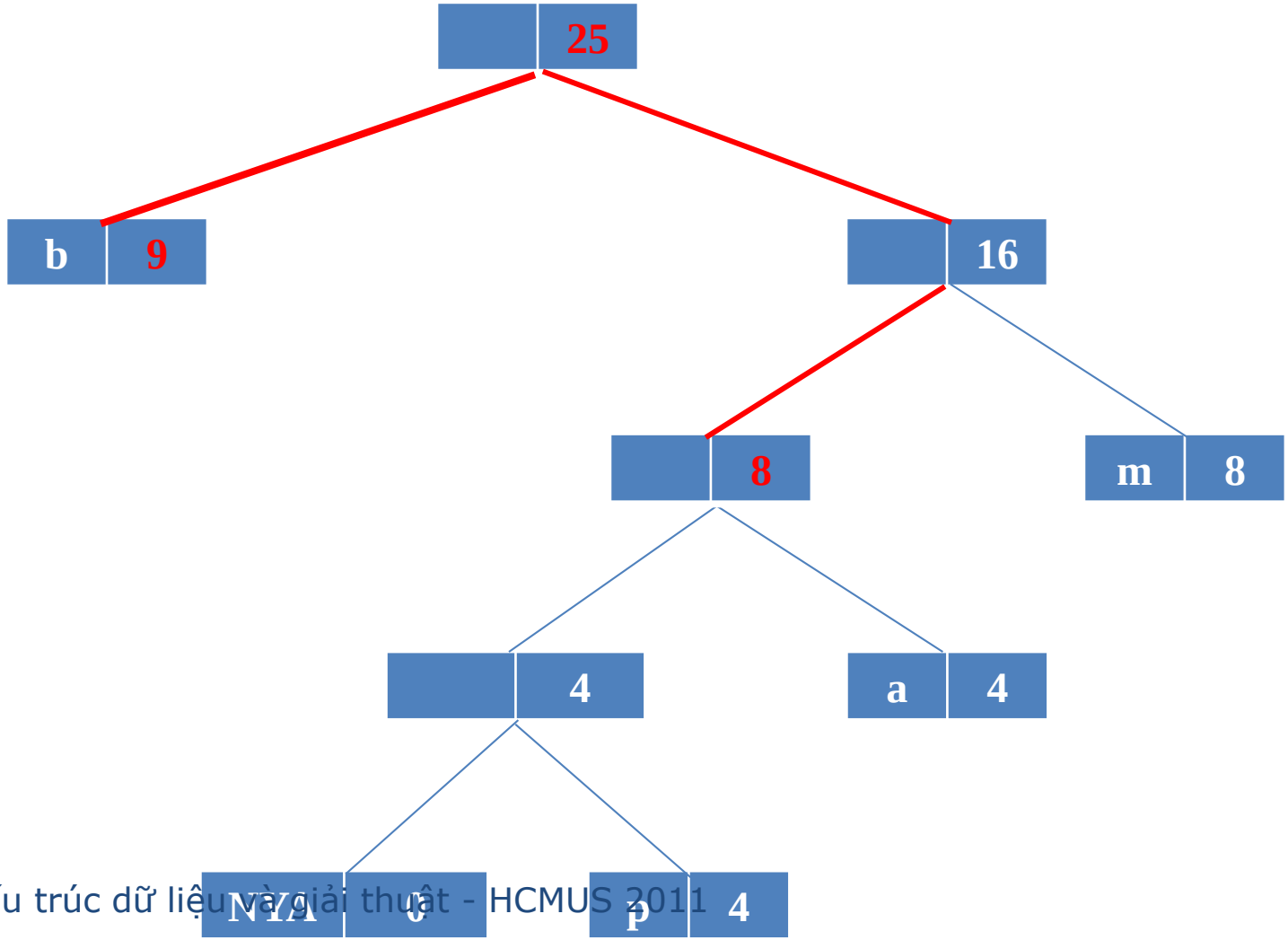


Cập nhật thêm ***b***

Thuật toán nén



Thuật toán nén

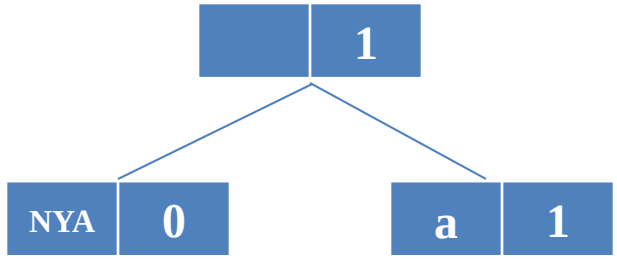


Ví dụ: Nén aafccc

Ký tự: ‘a’ → không có trong cây
Mã của ‘a’ = Mã NYA + biểu diễn 8 bít của ‘a’
= rỗng + ‘0110 0001’ = ‘0110 0001’

Cây ban đầu

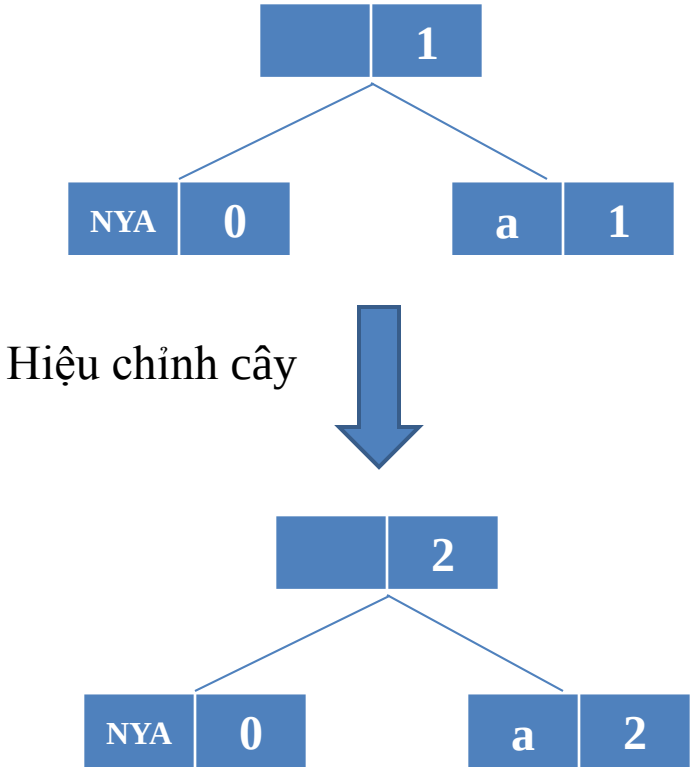
Hiệu chỉnh cây



Chuỗi nén: 0110 0001

Ví dụ: Nén aafccc

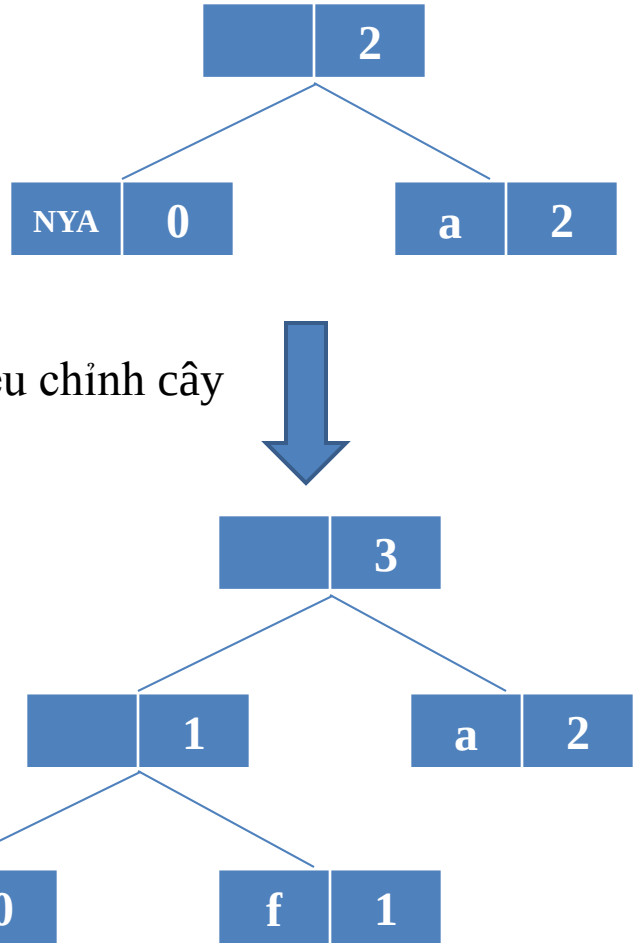
Ký tự: ‘a’ → có trong cây
Mã của ‘a’ = 1



Chuỗi nén: 0110 00011

Ví dụ: Nén aafccc

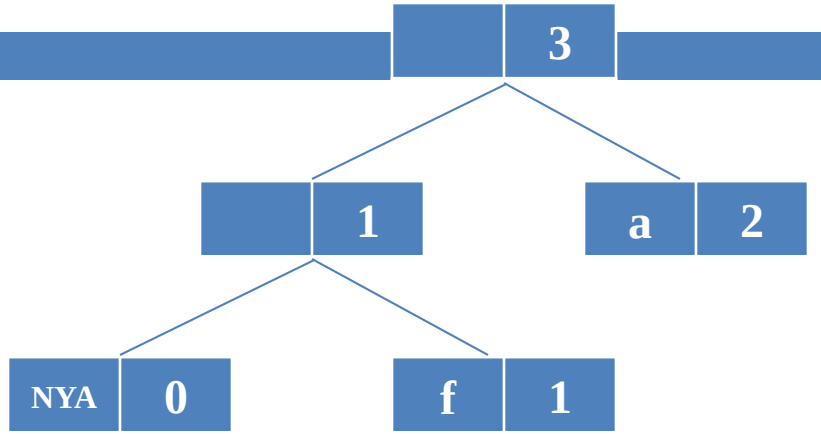
Ký tự: ‘f’ → không có trong cây
Mã của ‘f’ = Mã NYA + biểu diễn 8 bít của ‘f’
= ‘0’ + ‘0110 0110’ = ‘0 0110 0110’



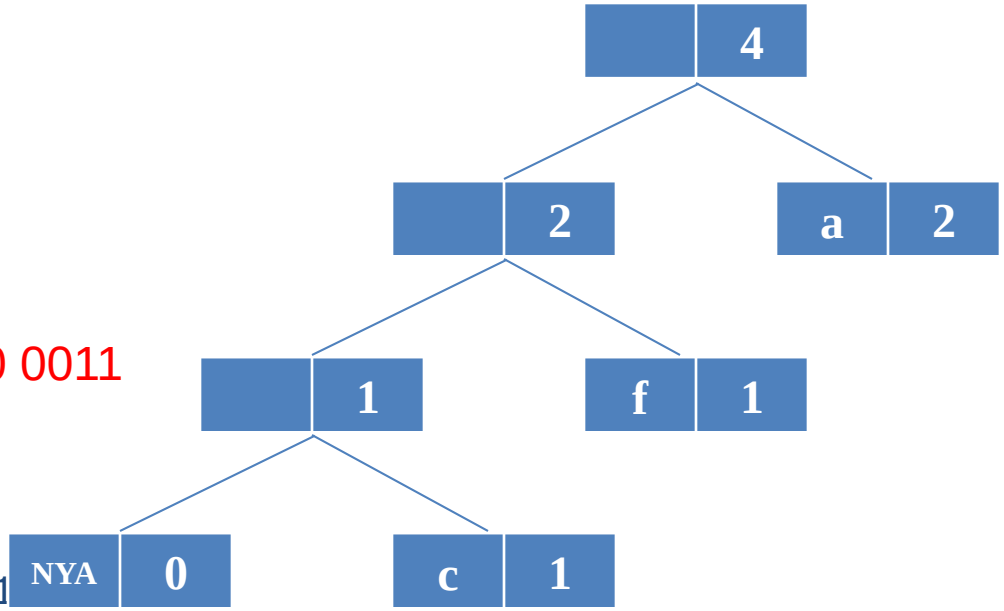
Chuỗi nén: 0110 00011001100110

Ví dụ: Nén aafccc

Ký tự: ‘c’ → không có trong cây
Mã của ‘c’ = Mã NYA + biểu diễn 8 bít của ‘c’
= ‘00’ + ‘0110 0011’ = ‘00 0110 0011’



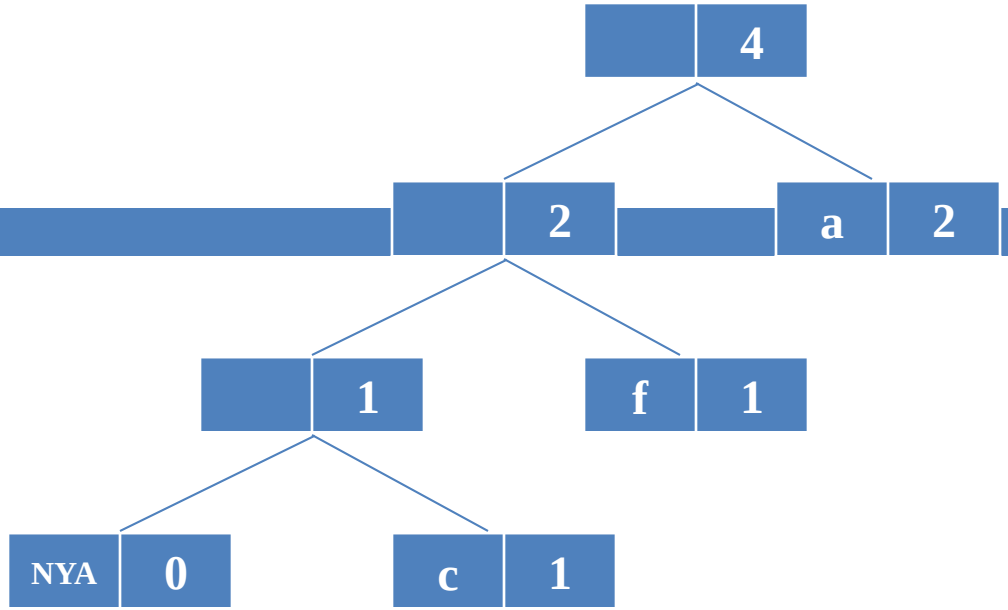
Hiệu chỉnh cây



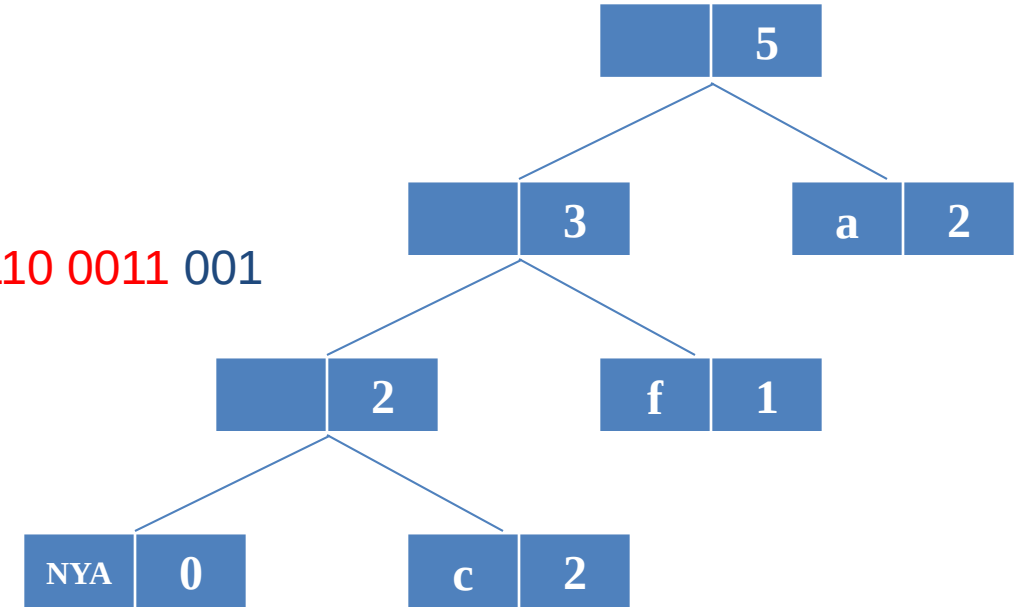
Chuỗi nén: 0110 00011001100110 00 0110 0011

Ví dụ: Nén aafccc

Ký tự: ‘c’ → có trong cây
Mã của ‘c’ = 001

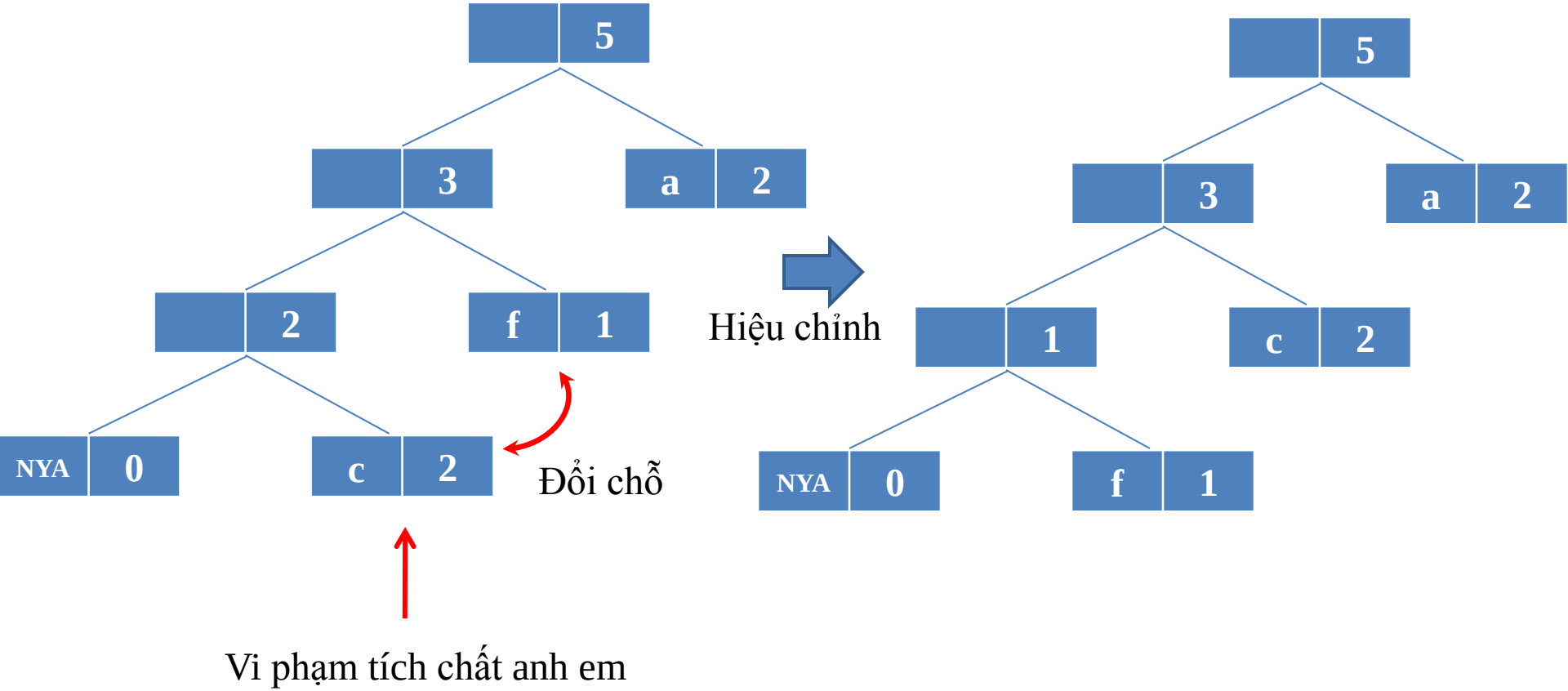


Hiệu chỉnh cây

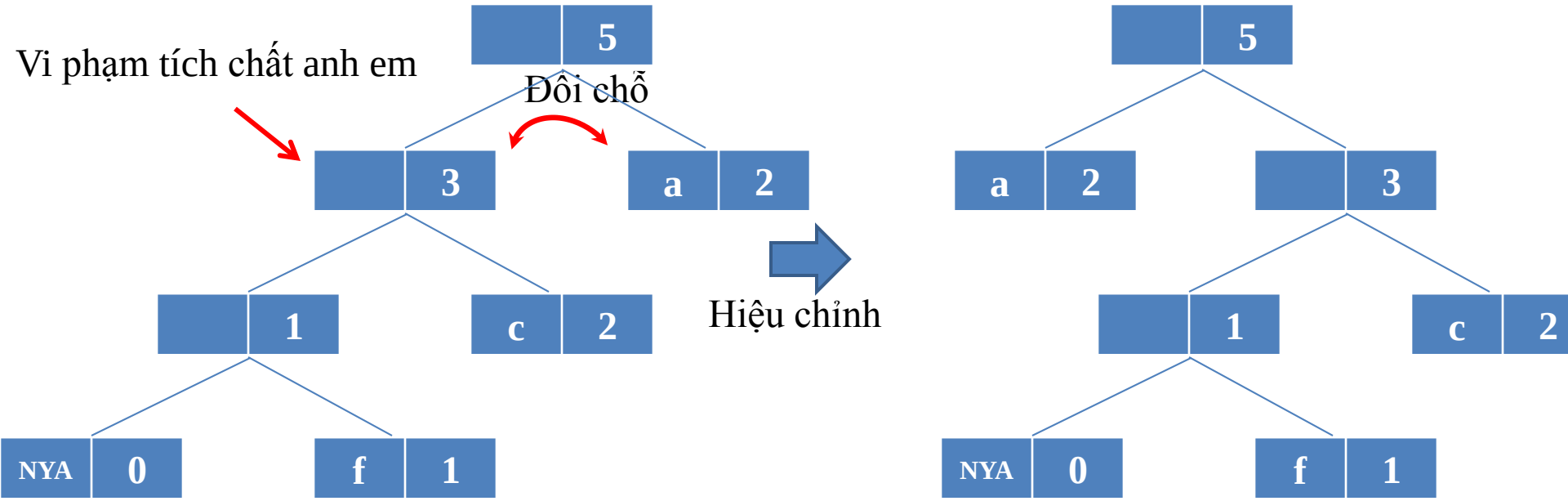


Chuỗi nén: 0110 00011001100110 00 0110 0011 001

Ví dụ: Nén aafccc

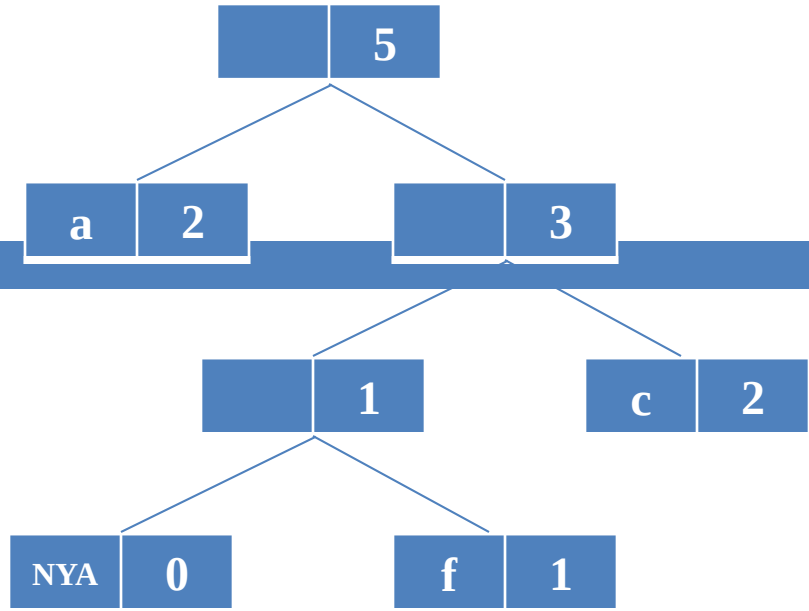


Ví dụ: Nén aafccc

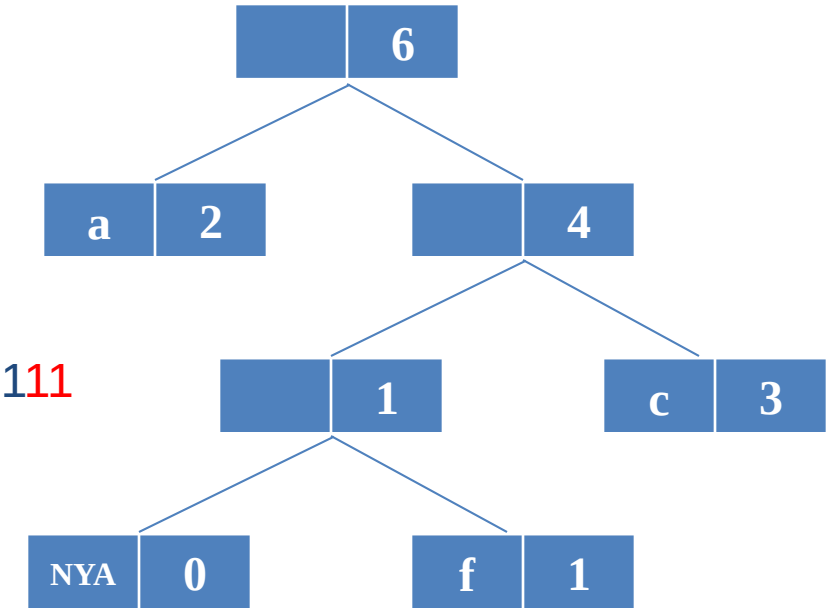


Ví dụ: Nén aafccc

Ký tự: ‘c’ → có trong cây
Mã của ‘c’ = 11

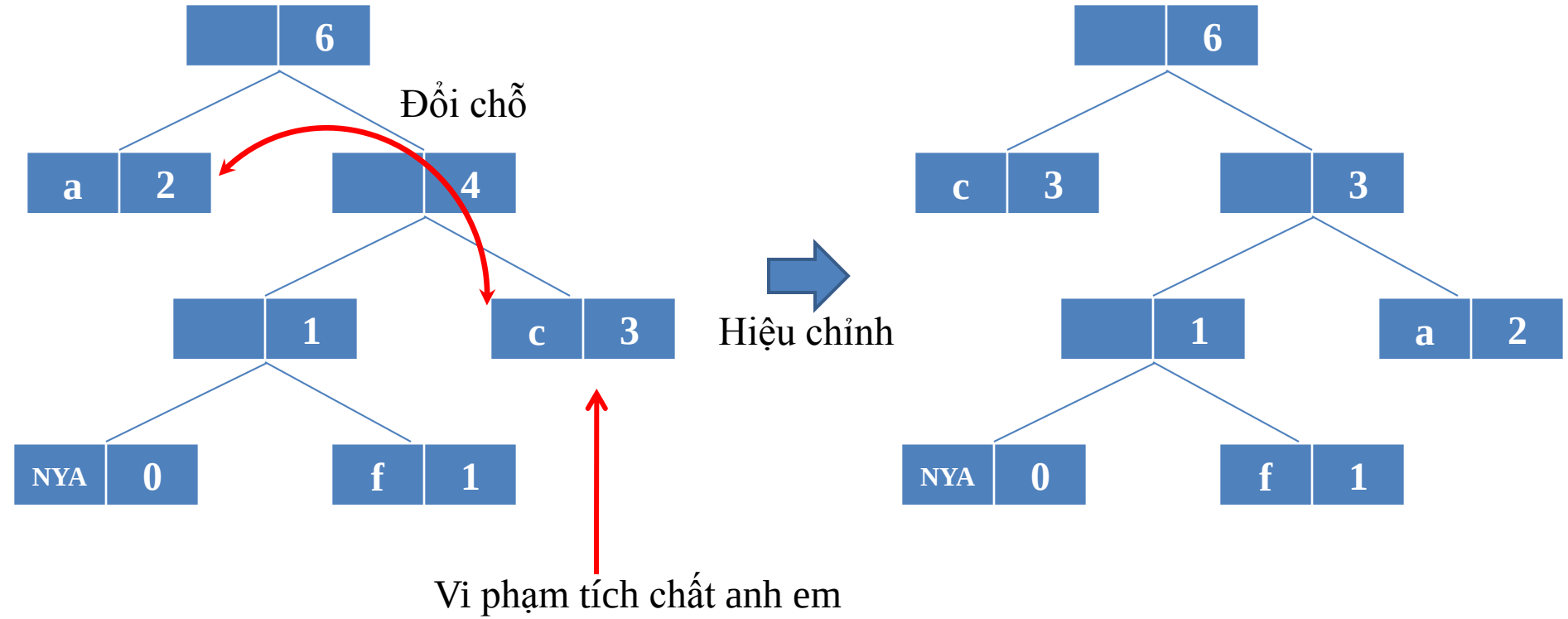


Hiệu chỉnh cây



Chuỗi nén: 0110 00011001100110 00 0110 0011 00111

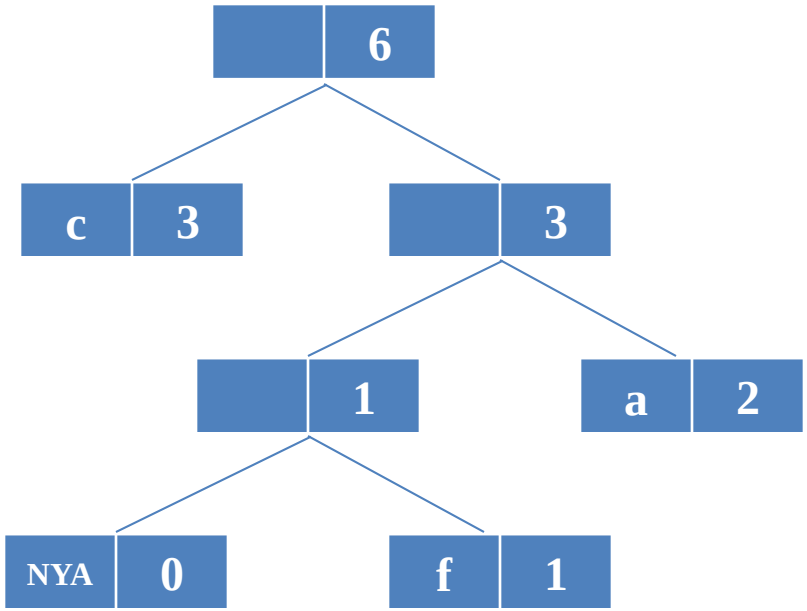
Ví dụ: Nén aafccc



Ví dụ: Nén aafccc

Chuỗi nén: 0110 000110011001100110 00 0110 0011 00111

Cây Huffman động tại thời điểm sau cùng:



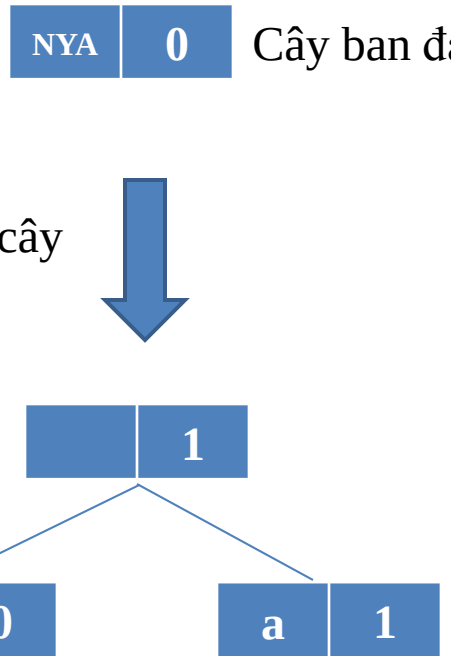
Thuật toán giải nén

- B1: Khởi tạo cây Huffman ban đầu (gồm 1 node NYA).
- B2: Giải nén dữ liệu dựa trên cây Huffman và dữ liệu nhận được (bit 0: nhánh trái, bit 1: nhánh phải).
  - Nếu nhận được dữ liệu **NYA**.
    - Đọc thêm 8 bit để xác định được ký tự ***c*** tương ứng.
  - Nếu không:
    - Giải nén được ký tự ***c***.
- B3: Thêm ký tự ***c*** vào cây Huffman. Cập nhật cây.
- B4: Quay lại từ bước 2.

Ví dụ: Giải nén 0110 00011001100110 00 0110 0011 00111

Chuỗi bit : 0110 00011001100110 00 0110 0011 00111

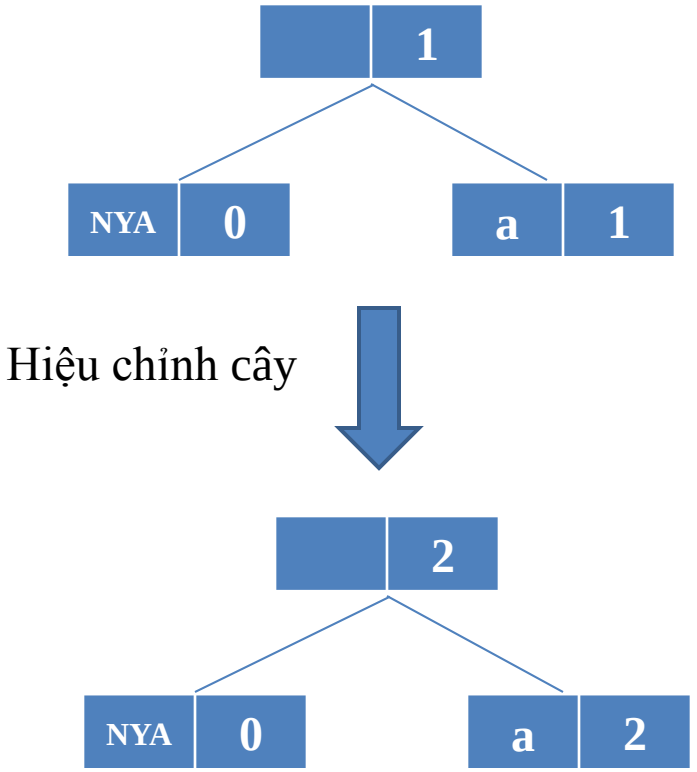
Gặp kí tự NYA ngay từ đầu → đọc 8 bit '0110 0001' → kí tự 'a'



Đã giải nén: a

Ví dụ: Giải nén 0110 00011001100110 00 0110 0011 00111

Chuỗi bit : 1001100110 00 0110 0011 00111
Di chuyển 1 (phải) → gặp node lá → kí tự 'a'

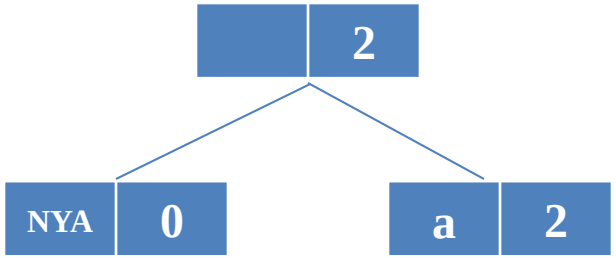


Đã giải nén: aa

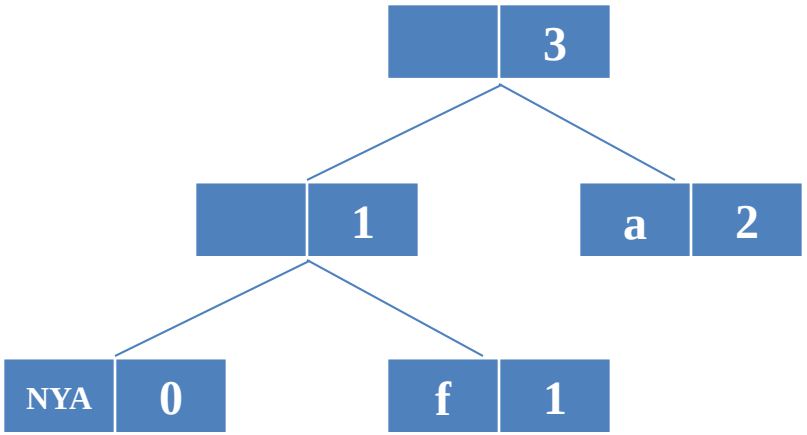
Ví dụ: Giải nén 0110 00011001100110 00 0110 0011 00111

Chuỗi bit : 0 01100110 00 0110 0011 00111

Đọc 0 (trái) → gặp node NYA → đọc 8 bit tiếp → kí tự ‘f’



Hiệu chỉnh cây

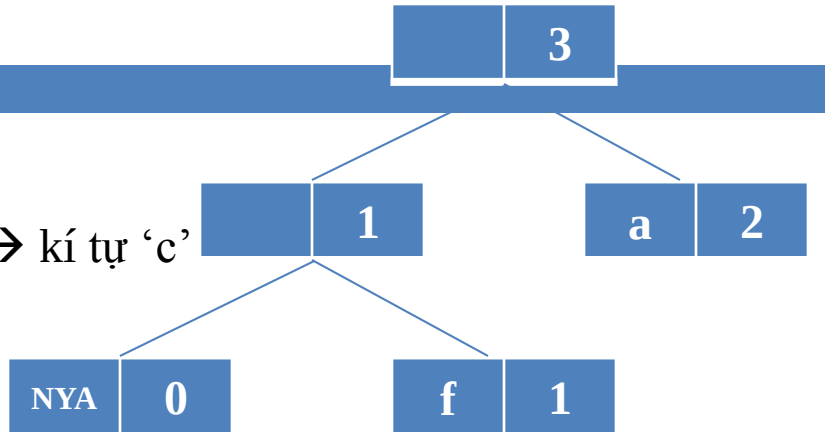


Đã giải nén: aaf

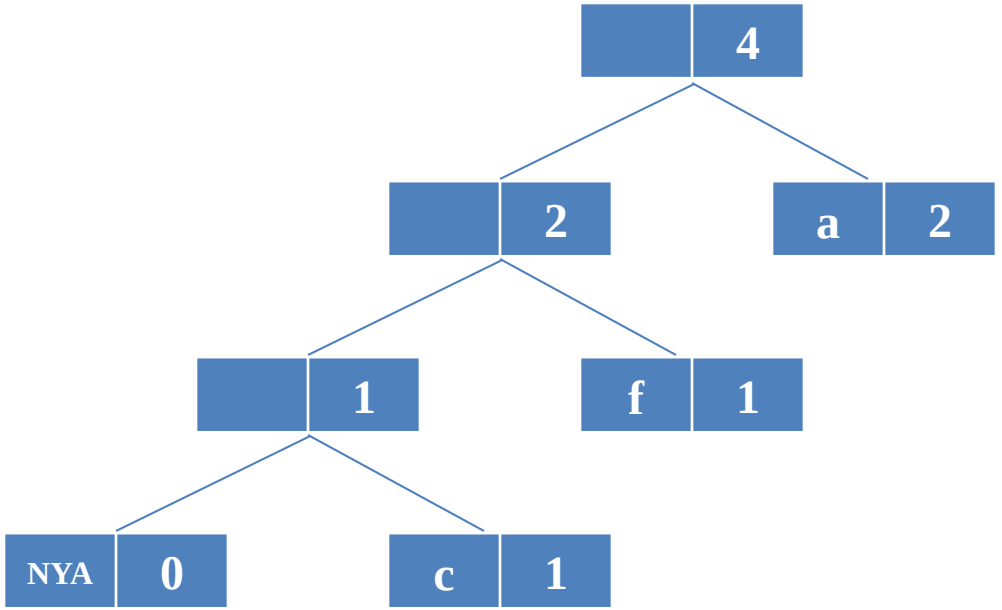
Ví dụ: Giải nén 0110 00011001100110 00 0110 0011 00111

Chuỗi bit : 00 0110 0011 00111

Đọc 0 (trái) 0 (trái) → gặp node NYA → đọc 8 bit tiếp → kí tự ‘c’



Hiệu chỉnh cây

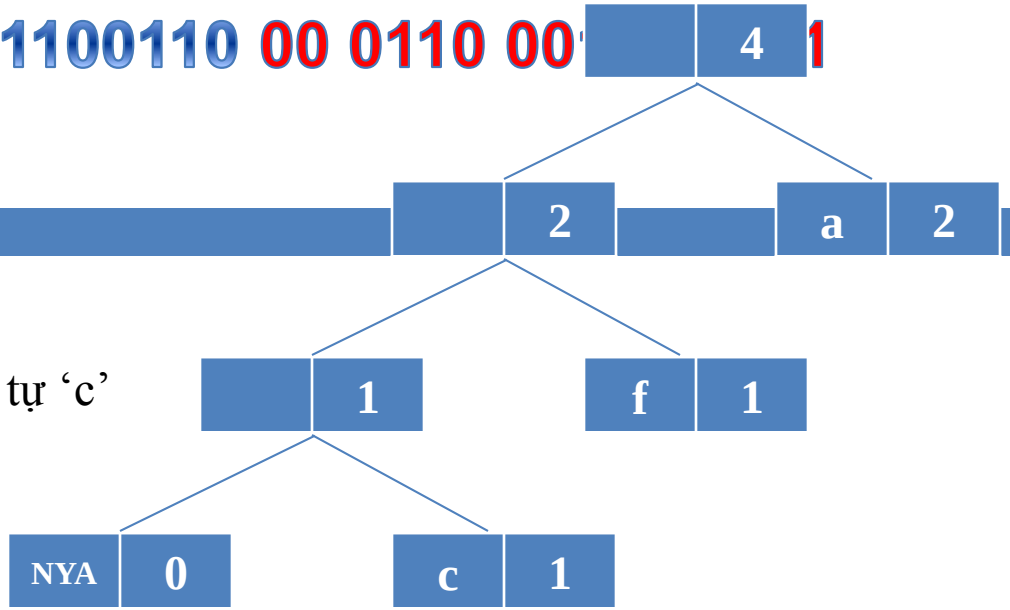


Đã giải nén: aafc

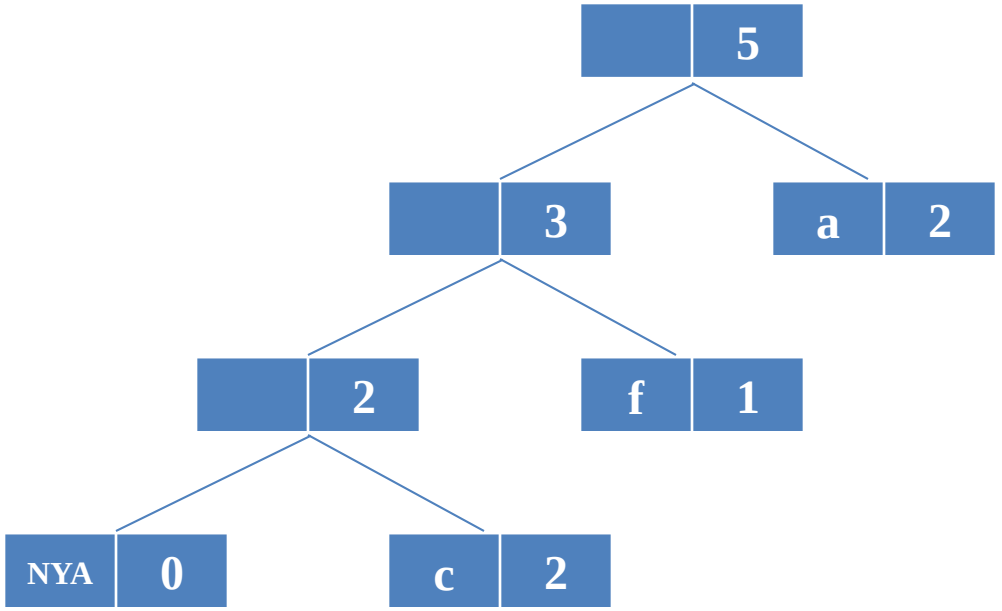
Ví dụ: Giải nén 0110 00011001100110 00 0110 00

Chuỗi bit : 00111

Đọc 0 (trái) 0 (trái) 1(phải)→ gặp node lá→ kí tự ‘c’

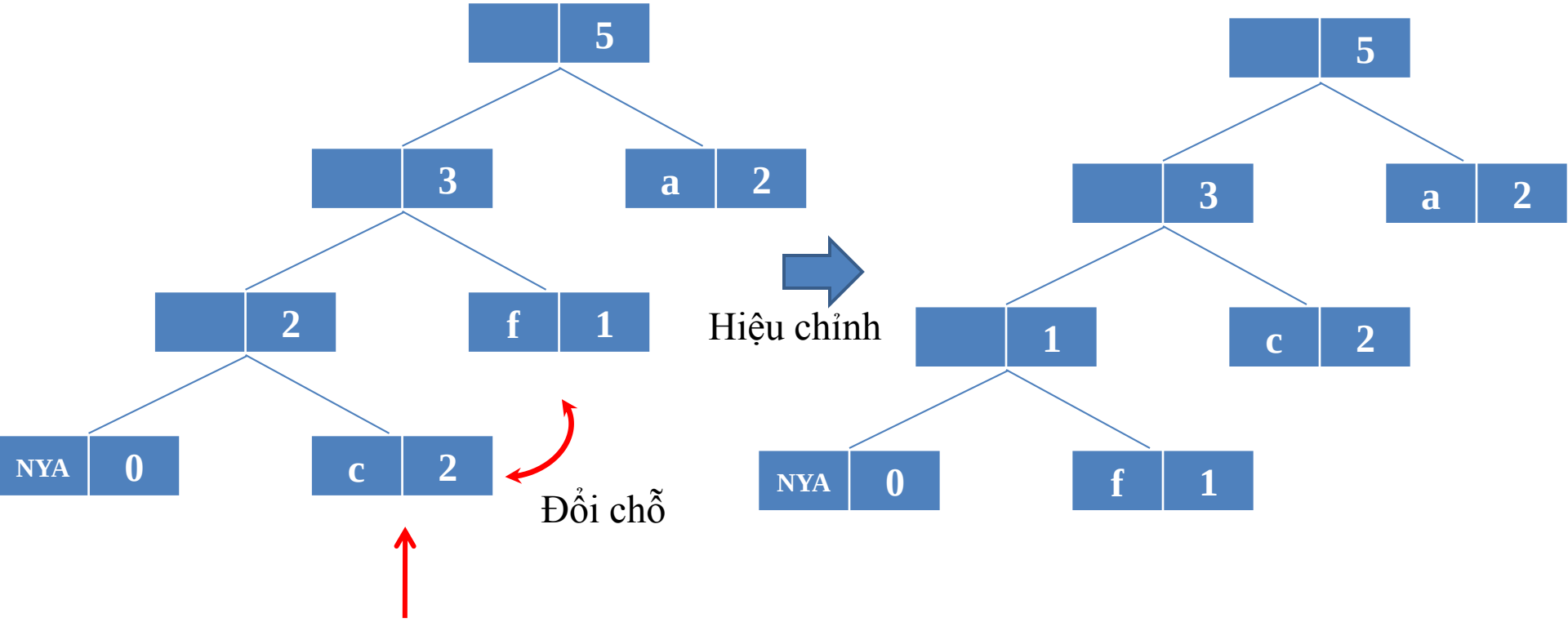


Hiệu chỉnh cây

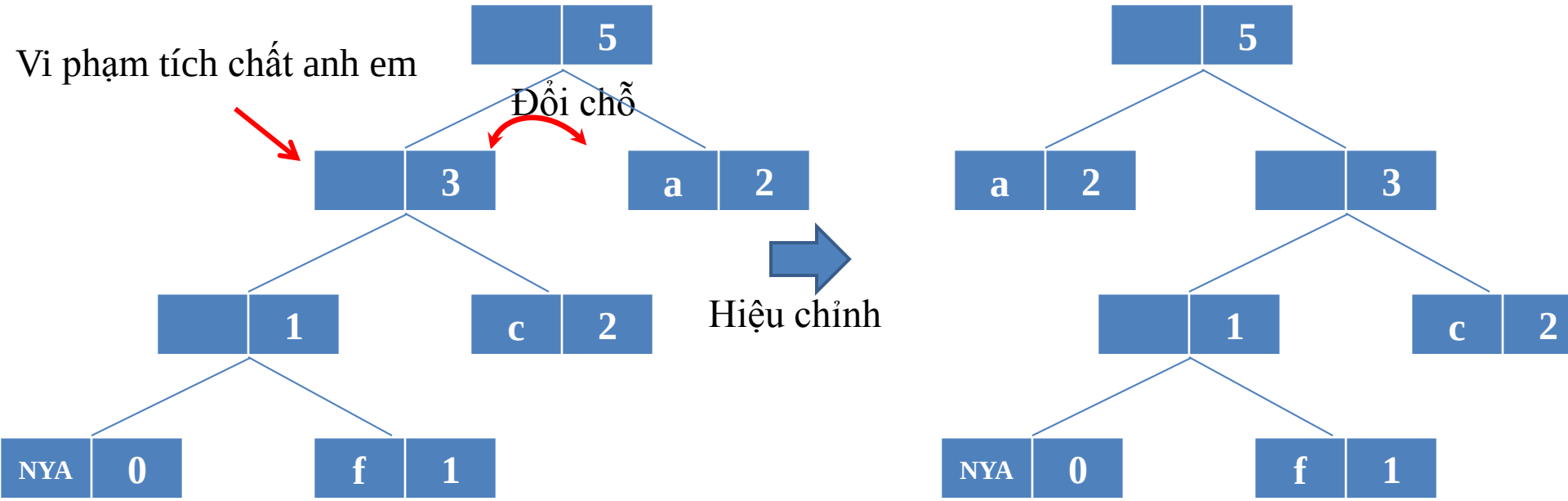


Đã giải nén: aafcc

Ví dụ: Giải nén 0110 00011001100110 00 0110 0011 00111



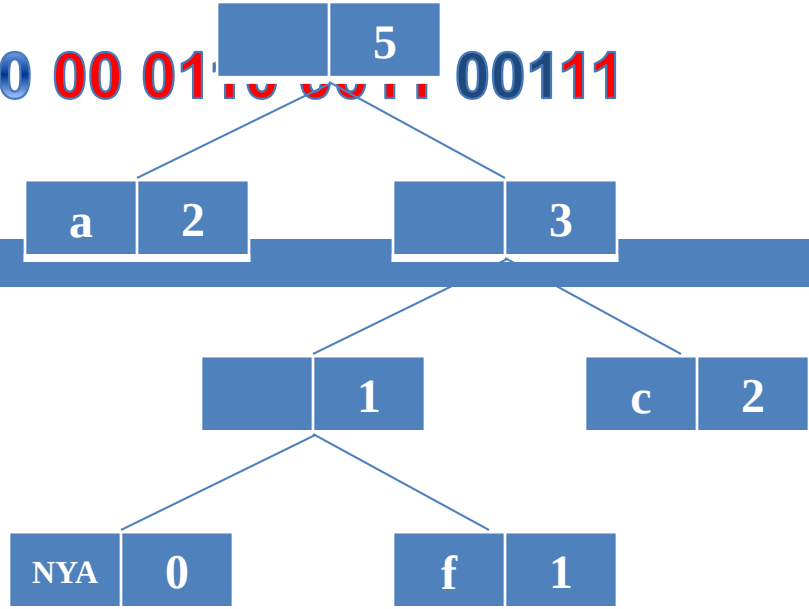
Ví dụ: Giải nén 0110 00011001100110 00 0110 0011 00111



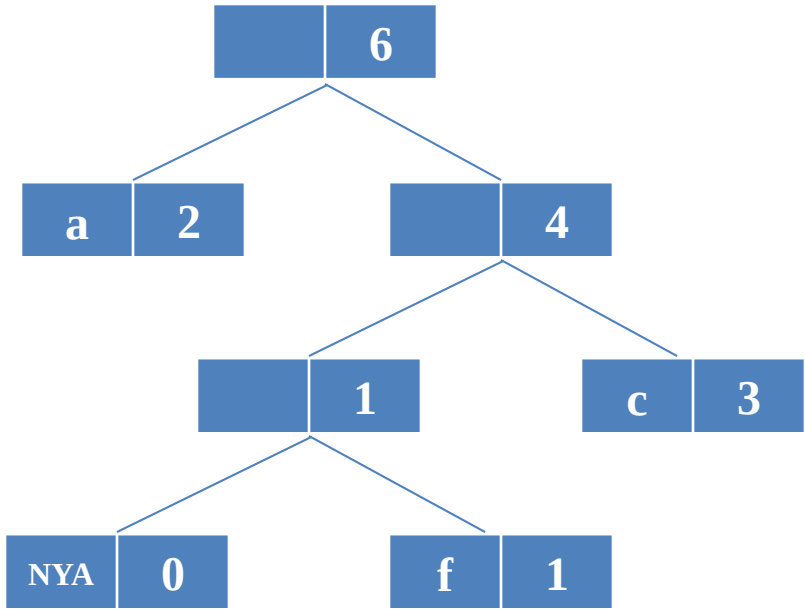
Ví dụ: Giải nén 0110 00011001100110 00 01[illegible] 00111

Chuỗi bit : 11

Đọc 1(phải) 1(phải) → gặp node lá→ kí tự ‘c’

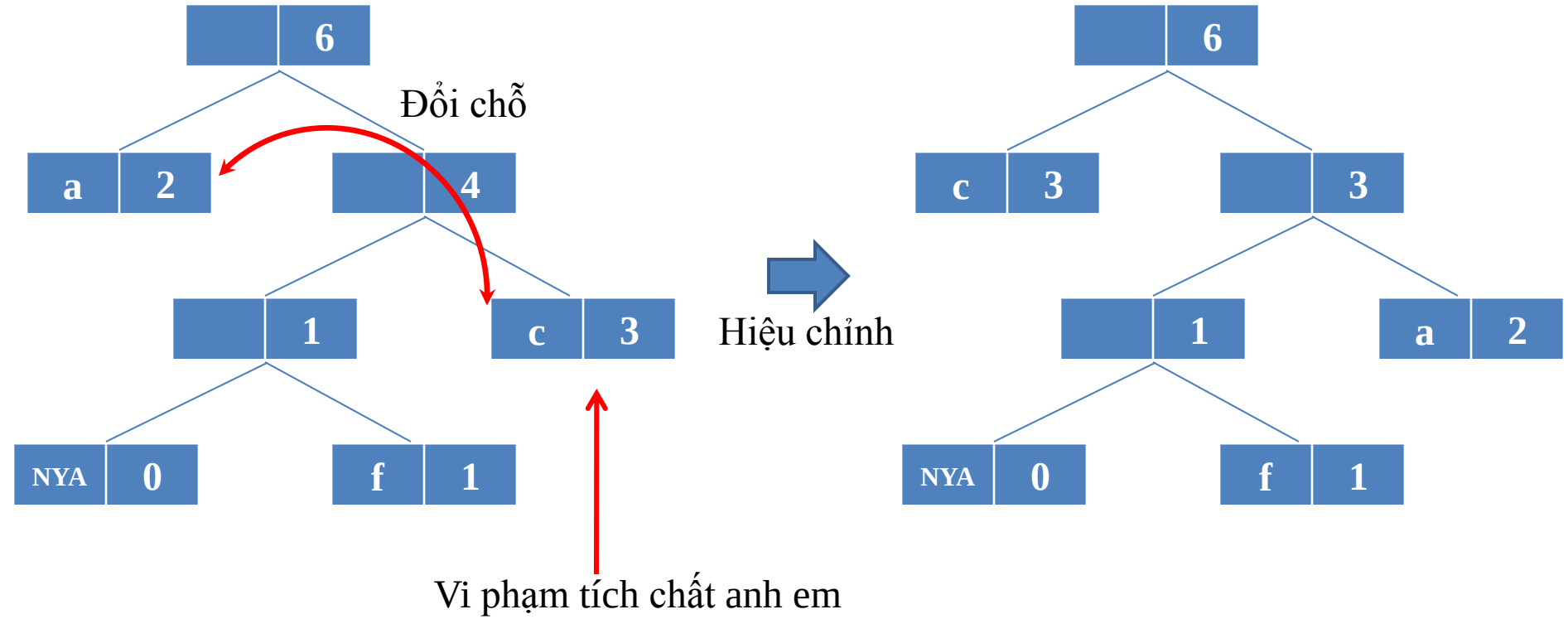


Hiệu chỉnh cây



Đã giải nén: aafccc

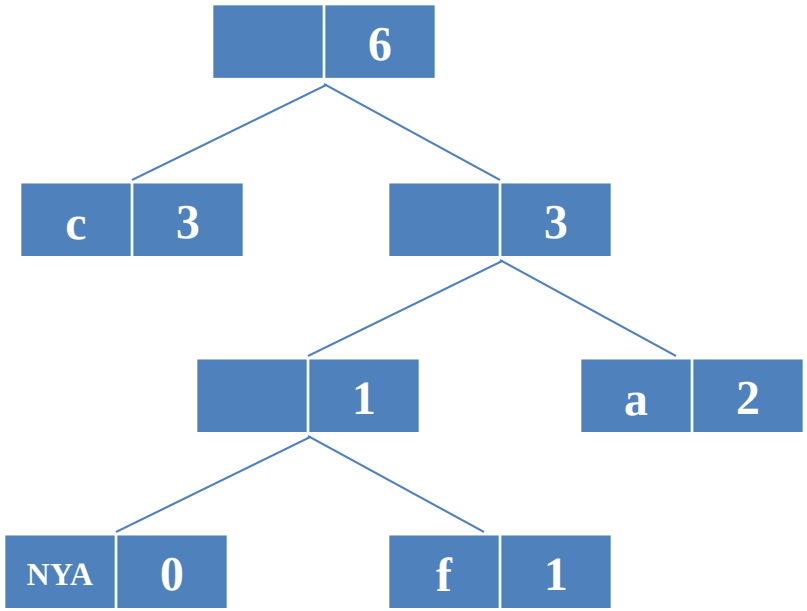
Ví dụ: Giải nén 0110 00011001100110 00 0110 0011 00111



Ví dụ: Giải nén 0110 00011001100110 00 0110 0011 00111

Chuỗi giải nén: aafccc

Cây Huffman động tại thời điểm sau cùng:



Tóm tắt

- Giải thuật nén Huffman là giải thuận nén dạng không mất mát thông tin.
- Các ký tự được nén và giải nén dựa trên mã bit.
- Chiều dài các mã bit là không giống nhau.
- Có thể áp dụng thuật toán này cho các loại dữ liệu khác nhau: tập tin văn bản, nhị phân,..

Tóm tắt

- Nén Huffman tĩnh:
  - Xây dựng cây Huffman dựa trên việc bảng thống kê dữ liệu (từ dữ liệu nén hoặc trên dữ liệu lớn có sẵn).
- Nén Huffman động:
  - Xây dựng cây Huffman theo thời gian thực.
  - Không cần biết trước toàn bộ nội dung dữ liệu cần nén.

Ví dụ

- Thực hiện việc nén các dữ liệu sau bằng thuật toán nén Huffman động:
  - fffcabcdaee
  - abracadabra