THUẬT TOÁN
TÌM KIẾM CHUỖI
(Phần 1)

1) Giải thuật Brute-Force:

a) Ý tưởng:

- So sánh từng ký tự của từ cần tìm với chuỗi chứa từ cần tìm, từ trái sang phải
- Luôn luôn dịch chuyển mẫu sang phải một vị trí.
- Nếu bằng nhau thì trả về vị trí
- Nếu không thì trả về -1

b) Ví dụ:

♦ Input:

- Xâu văn bản y=”Co cong mai sat, co ngay nen kim”
- Từ cần tìm x=“cong”

♦ Output:

- Trả về vị trí của x trong y

♦ Mô tả:

- Bước 1:

  Co cong mai sat, co ngay nen kim

  cong

- Bước 2:

  Co cong mai sat, co ngay nen kim

  cong

- Bước 3:

  Co cong mai sat, co ngay nen kim

  cong

- Bước 4:

  Co cong mai sat, co ngay nen kim

  cong

  ....

- Bước 6:

  Co cong mai sat, co ngay nen kim

  cong

  Tìm thấy x, trả về vị trí 4

- Minh họa bằng code:

```
int StringSearch(char *x, char *y) {
    for (int i=0; i<=strlen(y)-strlen(x); i++) {
        int j = 0;
        while (j < strlen(x)) {
            if (y[i+j]==x[j])
                j++;
            else
                break;
        }
        if(j==strlen(x))
            return i;  // tìm thấy
    }
    return -1 ;   // không tìm thấy
}
```

THUẬT TOÁN
TÌM KIẾM CHUỖI
(Phần 2)

2) Giải thuật Knuth-Morris-Pratt:

Nhận thấy thuật toán **Brute-Force** chưa tối ưu, nhiều trường hợp dư thừa, cho nên thuật toán Knuth-Morris-Pratt ra đời nhằm làm cho việc tìm kiếm nhanh hơn, tìm kiếm từ trái sang phải và không tìm lại những từ đã kiểm tra.

Tạo một bảng Next để Next đến các vị trí chưa kiểm tra

a) Bảng Next:

- Ví dụ:
  - Xâu cần tìm S = tartan
  - Xâu chuỗi T = tartaric_acid
  - Bước 1:

    tartaric_acid

    tartan

  - Theo giải thuật Brute-Force, bước 2 sẽ là:

    tartaric_acid

    tartan

  - Nhưng theo giải thuật Knuth-Morris-Pratt thì, bước 2 sẽ là:

    tartaric_acid

    tartan

Ở bước 1, ta nhận thấy:

- S[0]!=S[1], S[0]!=S[2] và S[1]=T[1], S[2]=T[2]
- Nên S[0]!=T[1] và S[0]!=T[2]
- Vì thế cho nên, xâu S cần tìm không thể xuất hiện trong đoạn từ T[1] hoặc T[2], nên theo giải thuật Knuth-Morris-Pratt thì ta tiếp tục tìm từ vị trí T[3] do:
  + S[3]=T[3] và S[4]=T[4]
  + Mà S[0]=S[3] và S[1]=T[4]
  + Cho nên khả năng xuất hiện xâu S ở vị trí T[3] trở đi rất cao
- Từ đó, ta xây dựng được bảng Next

P= tartan	**0**	**1**	**2**	**3**	**4**	**5**
Next	-1	0	0	-1	0	2

♦ Cài đặt thuật toán tạo bảng **NEXT**:

```
void NEXT_KMP(char p[], int NEXT[]) {
    int i, j;
    int m = strlen(p);
    i = 0;
    j = NEXT[0] = -1;
    while (i < m-1) {
        if (j == -1 || p[i] == p[j]) {
            i++; j++;
            if (p[i] != p[j])
                NEXT[i] = j;
            else
```

```
                NEXT[i] = NEXT[j];
            }
            else
                j = NEXT[j];
        }
    }
```

b) Cài đặt thuật toán tìm kiếm xâu P trong chuỗi T:

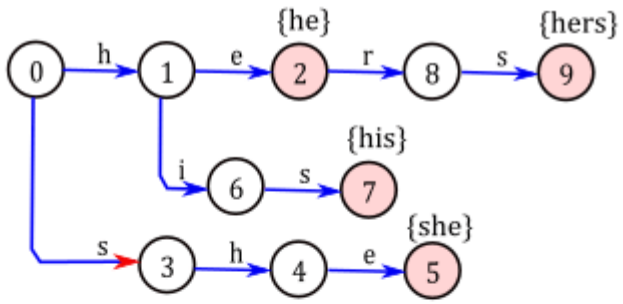
```
    int Search(char p[], char t[]) {
        int n = strlen(t);
        int m = strlen(p);
        int i = 0, j=0;
        while (j<=n-m) {
            if (p[j]==t[i+j]) {
                j++;
                // khi đã đi hết độ dài của chuỗi P
                //đã tìm được P, trả về vị trí tìm được
                if (j==m)
                    return i;
            }
            else {
                // khi không so khớp, dịch chuyển về vị trí NEXT[j]
                i = i + j – NEXT[j];
                j=0;
            }
```

```
        return -1;    // không tìm thấy
}
```

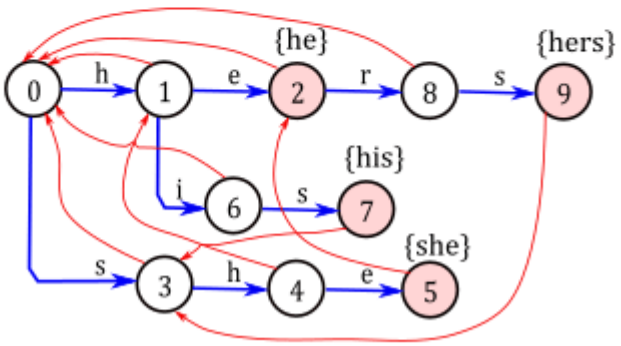
ĐỐI SÁNH MẪU

Đối sánh mẫu là công việc so sánh 2 mẫu ký tự với nhau, thực chất là quá trình tìm kiếm sự tồn tại của mẫu này có trong mẫu kia hay không. Để làm được điều đó ta sử dụng giải thuật Aho-Corasick Algorithm – tìm kiếm đa mẫu.

- Ý tưởng: Xây dựng cây từ khóa và tìm kiếm sự xuất hiện của từ khóa trong mẫu còn lại
- Cây từ khóa: Sử dụng Máy trạng thái hữu hạn
- Tìm từ khóa sử dụng thuật toán Knuth-Morris-Pratt
- Máy trạng thái hữu hạn:
  - Ví dụ: có tập từ khóa $\mathcal{K} = \{he, she, his, hers\}$
  - Áp dụng Máy trạng thái hữu hạn ta có cây từ khóa như sau:



(a) A Trie for {he, she, his, hers}

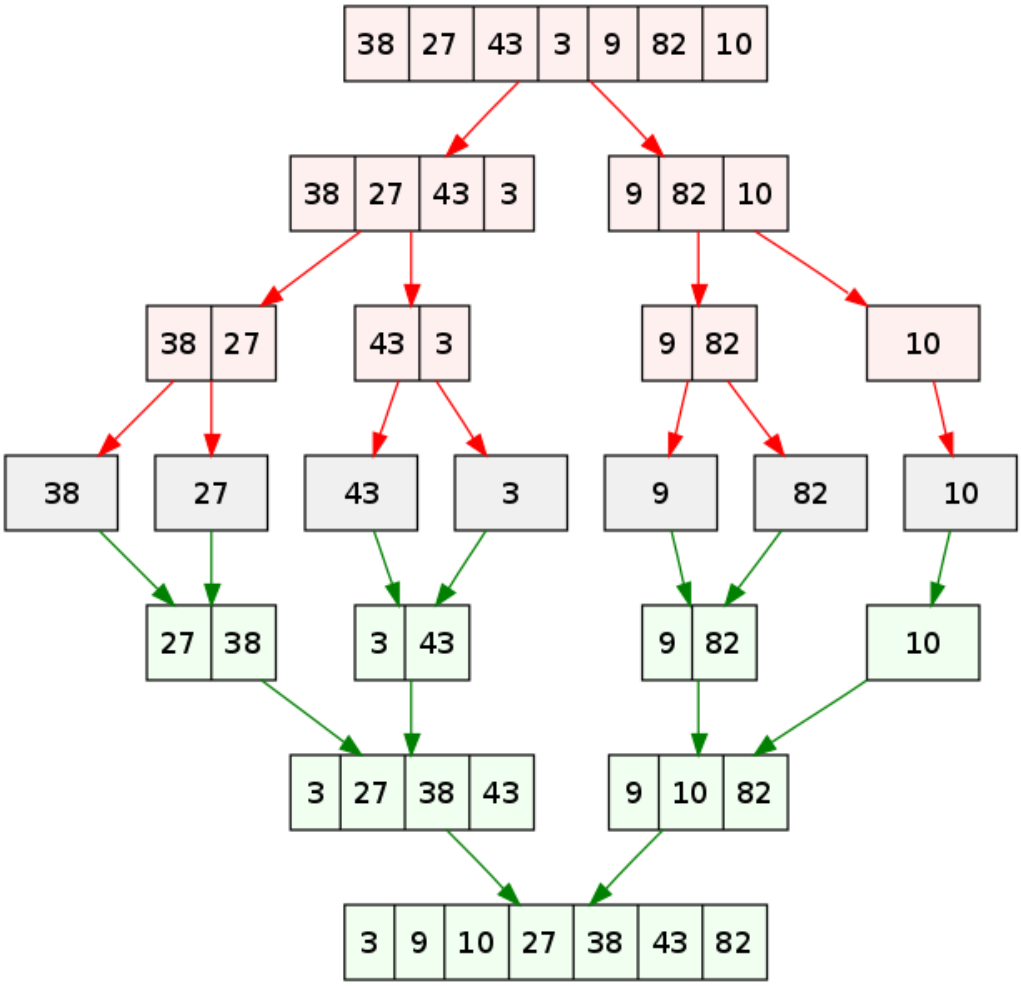


(b) A finite state machine for {he, she, his, hers}

- Phân tích:
  - Các Vector màu xanh đại diện cho 1 ký tự, nối các Vector lại ta có 1 từ khóa, mỗi từ khóa được xác định bởi các nút được tô màu, VD: Nút 2 được tô màu ta có từ khóa he
  - Các đường đi màu đỏ là cạnh không khớp, các cạnh không khớp để xác định đường đi trong cây, VD như cây trên sẽ đi từ trạng thái 0 đến trạng thái 7 ta được từ his, từ 7 đi theo cạnh không khớp đến trạng thái 3, sau đó đến 5 ta được từ she. Trạng thái 5 đi theo cạnh không khớp đến trạng thái 2 ta được he, từ 2 đến 9 ta được hers
  - Sau khi xây dựng xong cây từ khóa ta dùng thuật toán Knuth-Morris-Pratt để tìm số lần xuất hiện.
- Áp dụng thuật toán vào công việc đối sánh mẫu, cụ thể là so sánh 2 bài văn giống nhau bao nhiêu %
  - Bước 1: Lấy các ký tự của bài văn 1 và bài văn 2 xây dựng thành 2 cây từ khóa
  - Bước 2: So sánh 2 cây với nhau tìm các từ khóa bằng nhau
  - Bước 3: Tùy vào mức độ giống khác của 2 cây ta kết luận giống nhau bao nhiêu %

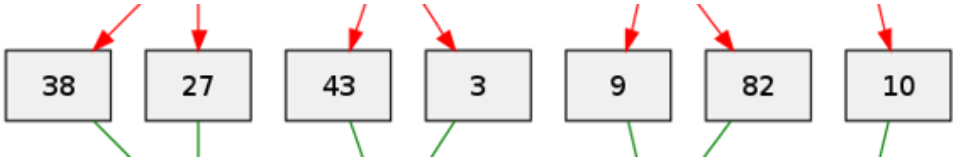
SẮP XẾP
BẰNG TRỘN

- Ý tưởng của thuật toán:
  - Sử dụng thuật toán chia để trị
  - Cụ thể, lấy phần tử giữa dãy số cần sắp xếp để chia dãy thành 2 nửa
  - Sắp xếp 2 nửa vừa chia, bằng cách đệ quy lại việc chia dãy số cho cả 2 nửa
  - Sau khi đệ quy xong, ta nhận được 2 nửa đã sẵp xếp
  - Nối 2 nửa đã sắp xếp bằng cách trộn 2 nửa lại với nhau
- Triển khai:
  - Hàm chia dãy (mergeSort):
    + Lấy phần tử ở giữa có mid=(left+right)/2
    + Chia đôi mảng thành Mảng 1(left,mid) và Mảng 2(mid+1,right)
    + Đệ quy việc chia mảng cho 2 nhánh con mergeSort(arr, left, mid) và mergeSort(arr, mid+1, right)
    + Gọi hàm gộp các chuỗi đã tách (merge), quy tắc gộp phần tử nhỏ hơn nằm đầu tiên
  - Hàm gộp chuỗi (merge):
    + Tạo 2 mảng tạm
    + Copy dữ liệu của 2 nửa sang 2 mảng tạm
    + Gộp 2 mảng tạm theo thứ tự tăng dần
- Ví dụ:



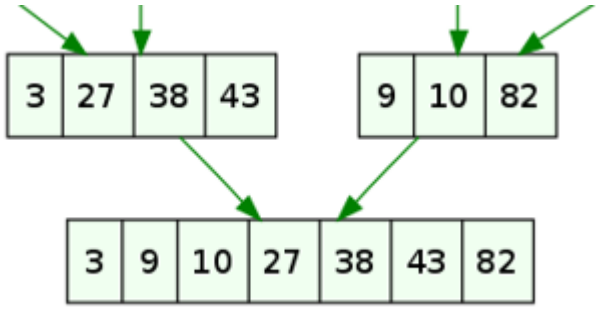
♦ Mô tả:

- Chia dãy số đến mức không chia được nữa (chia thành các dãy con, dãy chỉ có 1 chữ số)



- So sánh theo từng cặp nếu số nào nhỏ hơn sắp lên đầu tiên

- Gộp 2 nửa đầu tiên ta được dãy số đã sắp xếp



NÉN TẬP TIN
(Phần 1)

1) Các khái niệm:
   a) Phân loại:
      - ♦ Nén bảo toàn thông tin:
        - Không mất mát thông tin nguyên thủy
        - Hiệu suất nén không cao: 10-60%
        - Các giải thuật tiêu biểu: RLE, Arithmetic, Huffman, LZ77, LZ78,…
      - ♦ Nén không bảo toàn thông tin:
        - Thông tin nguyên thủy bị mất mát
        - Hiệu suất nén cao 40-90%
        - Các giải thuật tiêu biểu: JPEG, MP3, MP4,…
   b) Hiệu suất nén:
      - Là tỉ lệ % kích thước dữ liệu giảm được sau khi áp dụng thuật toán nén
      - D (%) = (N – M)/N*100
        - + D: Hiệu suất nén
        - + N: kích thước data trước khi nén
        - + M: kích thước data sau khi nén
      - Hiệu suất nén tùy thuộc:
        - + Phương pháp nén
        - + Đặc trưng của dữ liệu
2) Các loại giải thuật:
   a) Giải thuật nén RLE:
      - ♦ Mã hoá theo độ dài lặp lại của dữ liệu

- Ví dụ:
  - Data = AAAABBBBBBBBCCCCCCCCCCDEE (# 25 bytes)
  - $Data_{nén}$ = 4A8B10C1D2E (# 10 bytes)
- Ưu điểm:
  - Đối với các dữ liệu có sự lặp đi lặp lại nhiều lần của một kí tự thì thuật toán này cực kì phù hợp. Nó giảm được đáng kể dung lượng của dữ liệu.
- Nhược điểm:
  - Với những loại dữ liệu mà thông tin ít lặp thì việc sử dụng RLE không thật sự hiệu quả. Nó tạo ra dữ liệu sau khi nén có dung lượng lớn hơn cả dữ liệu "nguyên thuỷ". Đây được gọi là hiệu ứng ngược.
  - Xét ví dụ sau:
    + Cho đoạn dữ liệu: abcdefgh #8byte
    + Sau khi nén bằng RLE sẽ là: a1b1c1d1e1f1g1h1 #16 byte

b) Nén Huffman:

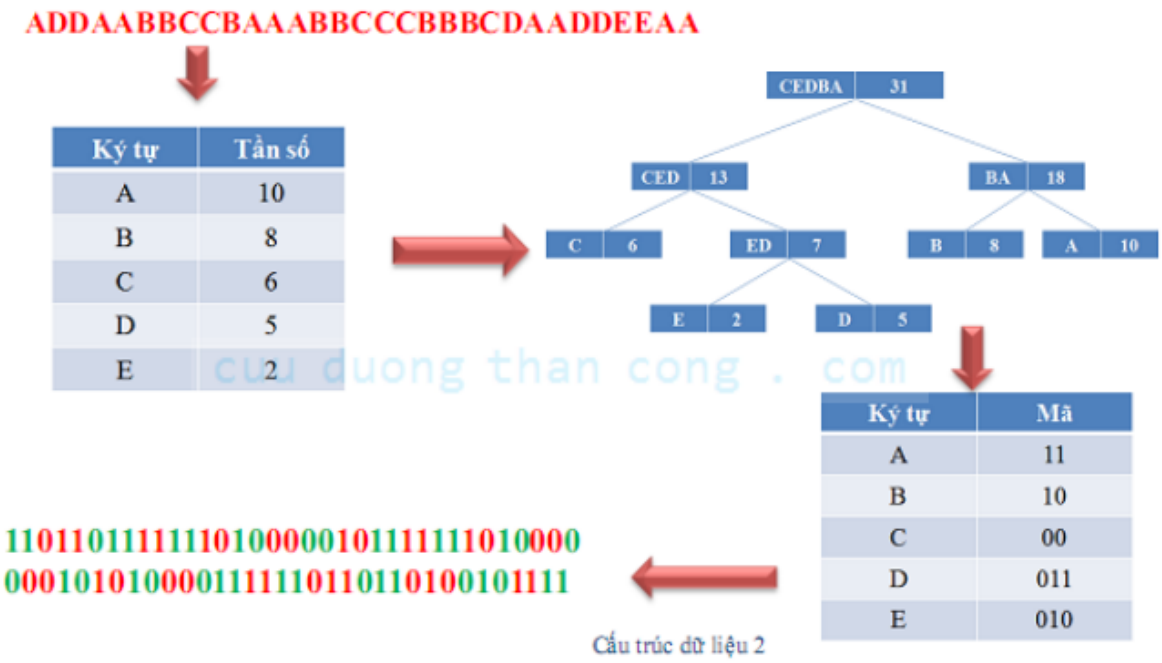
- Ý tưởng: thông thường mỗi ký tự được biểu diễn dưới dạng 8 bit, với thuật toán này ta có thể không cần dùng đủ 8 bít để mã hóa cho một ký hiệu, hơn nữa độ dài (số bít) dành cho mỗi ký hiệu có thể khác nhau, ký hiệu nào xuất hiện nhiều lần thì nên dùng số bít ít, ký hiệu nào xuất hiện ít thì có thể mã hóa bằng từ mã dài hơn. Như vậy ta có việc mã hóa với độ dài thay đổi.
- Ví dụ: Giả sử mã hóa từ "ARRAY", tập các ký hiệu cần mã hóa gồm 3 chữ cái "A","R","Y".
  + Nếu mã hóa bằng các từ mã có độ dài bằng nhau ta dùng ít nhất 2 bit cho một chữ cái chẳng hạn "A"=00, "R"=01, "Y"=10. Khi đó mã hóa của cả từ là 0001010010. Để giải mã ta đọc hai bit một và đối chiếu với bảng mã.
  + Nếu mã hóa "A"=0, "R"=01, "Y"=11 thì bộ từ mã này không là mã tiền tố vì từ mã của "A" là tiền tố của từ mã của "R". Để mã hóa cả từ ARRAY phải đặt dấu ngăn cách vào giữa các từ mã 0,01,01,0,11

+ Nếu mã hóa "A"=0, "R"=10, "Y"=11 thì bộ mã này là mã tiền tố. Với bộ mã tiền tố này khi mã hóa xâu "ARRAY" ta có 01010011.

♦ Thực hiện:

- Bước 1: Lập bảng thống kê tần số xuất hiện của các ký tự.
- Bước 2: Xây dựng cây Huffman dựa vào bảng thống kê tần số xuất hiện.
- Bước 3: Phát sinh bảng mã bit cho từng ký tự tương ứng, mã bit với độ dài thay đổi.
- Bước 4: Duyệt tập tin, thay thế các ký tự trong tập tin bằng mã bit tương ứng.
- Bước 5: Lưu lại thông tin của cây Huffman cho giải nén.
- Việc giải nén chỉ cần dựa vào cây Huffman và bảng mã bit

♦ Minh họa:



Cấu trúc dữ liệu 2

NÉN TẬP TIN
(Phần 2)

c) Nén LZW:

Link: https://medium.com/@kongminh/n%C3%A9n-lzw-813b7e4cd7eb

3) Các hình thức nén khác:

a) Giảm dung lượng file .JS (Javascript):

- Xóa toàn bộ dấu cách và dấu xuống dòng có trong file
- Việc xóa dấu cách và dấu xuống dòng sẽ không ảnh hưởng đến Code vì các câu lệnh được phân biệt bởi dấu ; và ngoặc {}

b) Nén Video:

- Nén video sang các định dạng nén cho ít dung lượng, tùy vào loại định dạng sẽ cho chất lượng video vượt trội hơn hoặc thấp hơn
- Giảm tần số, FPS hoặc chất lượng video xuất ra, video xuất sẽ có dung lượng ít hơn video gốc nhưng chất lượng sẽ thấp hơn ban đầu
- Tần số nén chuẩn là 2400 kb/s, FPS chuẩn là 29.97 , với tần số và FPS vừa nêu video xuất ra sẽ có chất lượng gần như video gốc nhưng dung lượng giảm rất đáng kể.
- Phần mềm thường dùng để nén video là Format Factory

c) Nén hình ảnh:

- Giảm điểm ảnh
- Giảm kích thước
- Có nhiều trang web hỗ trợ việc nén ảnh, theo các tỉ lệ nén từ 20% - 100%, tỉ lệ nén 20% là chuẩn nhất vì không làm chất lượng ảnh nhiều nhưng dung lượng giảm đáng kể
- Việc nén ảnh được dùng nhiều trong việc tối ưu hóa Website

MÃ HÓA

Ngày nay, con người luôn sống trong môi trường trao đổi thông tin hàng ngày, hàng giờ. Trong phần lớn trường hợp trao đổi thông tin giữa hai đối tác, người ta không hề muốn để thông tin bị lộ cho người thứ ba biết vì điều đó có thể gây ra những tổn thất cả về vật chất cũng như về tinh thần.

Để bảo vệ bí mật cho thông tin của mình được gửi đi trong một môi trường "mở" tức là môi trường có thể có nhiều tác nhân tiếp cận ngoài hai đối tác trao đổi thông tin, người ta phải dùng mật mã tức là dùng những phương pháp mã hóa thông tin thành một dạng bí mật mà chỉ có những người nắm được quy luật mới có thể hiểu được.

Để làm được điều này, ta sử dụng cấu trúc dữ liệu Bảng Băm để lưu Key.