Các thuật toán so sánh mẫu lần lượt từ trái sang phải thường là các dạng cải tiến (và cải lùi) của thuật toán Knuth-Morris-Pratt và thuật toán sử dụng Automat như: Forward Dawg Matching, Apostolico-Crochemore, Not So Naive, …

 Các thuật toán so sánh mẫu từ phải sang trái đều là các dạng của thuật toán Boyer-Moore.

* Thuật toán Colussi và Galil-Giancarlo lại chia mẫu thành hai tập và tiến hành tìm kiếm trên mỗi tập với một chiều khác nhau.
* Thuật toán Skip Search, dựa sự phân bố các ký tự để quyết đinh vị trí bắt đầu của mẫu trên văn bản.

1. **Thuật toán Brute Force:**

**Brute Force là một thuật toán vét cạn**,

Có lẽ cái tên của thuật toán này đã nói lên tất cả (brute nghĩa là xúc vật, force nghĩa là sức mạnh). Thuật toán brute force thử kiểm tra tất cả các vị trí trên văn bản từ 1 cho đến n-m+1. Sau mỗi lần thử thuật toán brute force dịch mẫu sang phải một ký tự cho đến khi kiểm tra hết văn bản.

Thuật toán brute force không cần công việc chuẩn bị cũng như các mảng phụ cho quá trình tìm kiếm. Độ phức tạp tính toán của thuật **toán này là**O**(n\*m)**

1. **Thuật toán Knuth-Morris-Pratt** là d**ựa trên thuật toán brute force với ý tưởng lợi dụng lại những thông tin của lần thử trước cho lần sau.** Trong thuật toán brute force vì chỉ dịch cửa sổ đi một ký tự nên có đến m-1 ký tự của cửa sổ mới là những ký tự của cửa sổ vừa xét. Trong đó có thể có rất nhiều ký tự đã được so sánh giống với mẫu và bây giờ lại nằm trên cửa sổ mới nhưng được dịch đi về vị trí so sánh với mẫu. Việc xử lý những ký tự này có thể được tính toán trước rồi lưu lại kết quả. Nhờ đó lần thử sau có thể dịch đi được nhiều hơn một ký tự, và giảm số ký tự phải so sánh lại.

Thuật toán so khớp chuỗi Knuth–Morris–Pratt (hay thuật toán KMP) tìm kiếm sự xuất hiện của một "từ" W trong một "xâu văn bản" S bằng cách tiếp tục quá trình tìm kiếm khi không phù hợp, bản thần "từ" W cho ta đầy đủ thông tin để xác định vị trí bắt đầu của ký tự so sánh tiếp theo, do đó bỏ qua quá trình kiểm tra lại các ký tự đã so sánh trước đó.

1. **Thuật toán tìm kiếm Rabin Karp**

Tư tưởng chính của phương pháp này là **sử dụng phương pháp băm (hashing)**. Thay vì phải so sánh tất các vị trí của văn bản, **ta chỉ cần so sánh những cửa sổ bao gồm những ký tự “có vẻ giống” mẫu.**

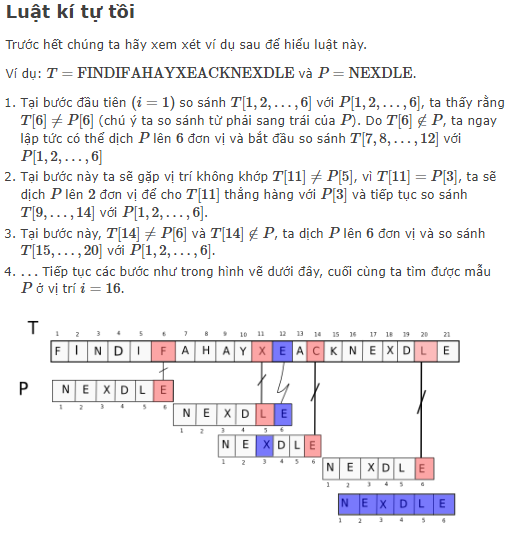
Tức là mỗi một xâu sẽ được gán với một giá trị của hàm băm (hash function), ví dụ xâu “hello” được gán với giá trị 5 chẳng hạn, và hai xâu được gọi là bằng nhau nếu giá trị băm của nó bằng nhau. Như vậy thay vì việc phải đối sánh các xâu con của y với mẫu x, ta chỉ cần so sánh giá trị hàm băm của chúng và đưa ra kết luận. Hàm băm được dùng ở đây là hàm băm rollHash tức là giá trị băm của chuỗi con tiếp theo được tính dựa theo giá trị băm của chuỗi con trước đó như thế sẽ tiết kiệm được nhiều công sức

1. **PHẦN 2**
2. **Thuật toán Boyer Moore**

**thuật toán Boyer-Moore kiểm tra các ký tự của mẫu từ phải sang trái và khi phát hiện sự khác nhau đầu tiên thuật toán sẽ tiến hành dịch cửa sổ đi Trong thuật toán này có hai cách dịch của sổ:**

**Cách thứ 1: gần giống như cách dịch trong thuật toán KMP, dịch sao**

**cho những phần đã so sánh trong lần trước khớp với những phần giống nó trong lần sau**



## **Zhu Kataok**

### **2.1 Đặc điểm**

* Là một biến thể của Boyer Moore.
* Sử dụng 2 kí tự liên tiếp nhau để tính toán bước dịch bad charater
* Pha cài đặt có độ phức tạp thuật toán và không gian nhớ là O(m+σ2)
* Pha thực thi có độ phức tạp là O(mxn)

### **2.2 Trình bày thuật toán**

* Zhu và Takaoka thiết kế thuật toán mà chúng thực thi dựa trên bad charater. Trong quá trình tìm kiếm, việc so sánh được thực hiện từ phải qua trái, và khi cửa sổ đang ở vị trí y[j…j+m-1] và xuất hiện sự khác nhau giữa x[m- k] và y[j+m-k] trong khi x[m-k+1 .. m-1]=y[j+m-k+1 .. j+m-1] . bước dịch good suffix cũng được sử dụng để tính toán bước dịch.

## **Colussi algorithm**

### **1.1 Đặc điểm**

* **đây là thuật toán cái tiến từ thuật toán Knuth, Morris and Pratt.**
* mẫu tìm kiếm sẽ được chia thành 2 tập con. Tập thứ nhất sẽ được duyệt từ trái sang phải khi không gặp mismatch ta duyệt tập thứ hai từ trái qua phải.

### **Trình bày thuật toán**

* Thuật toán Colussi được thiết kế dựa theo những phân tích từ thuật toán knuth, Morris, và Pratt. các vị trí kí tự trong mẫu được chia thành 2 tập riêng biệt. mỗi phép thử sẽ bao gồm 2 pha :
* Trong pha thứ nhất thực hiên so sánh từ trái sang phải lần lượt các kí tự tại những vị trí kmpNext > -1. Những vị trí này gọi là noholes.
* Pha thứ hai so sánh kí tự từ phải sang trái ( các kí tự còn lại ) nhưng kí tự này có kmpNext = -1 và gọi là holes.
* Chiến lược này có 2 lợi thế :
* khi gặp mismath trong pha thứ nhất, sau khi thực hiện shift bước dịch ta không cần phải so sánh lại các kí tự tại những vị trí noholes phía trước.
* khi gặp mismath trong pha thứ hai có nghĩa là đoạn hậu tố đã matches tất cả kí tự

1. **Tìm kiếm mẫu từ vị trí cụ thể**

## **Skip Search Algorithm**

### **2.1 Đặc điểm**

* Sử dụng thùng chứa (bucket) các vị trí xuất hiện của kí tự trong xâu mẫu.
* Pha xử lý có độ phực tạp thời gian và không gian chứa O(m+ ∂)
* Pha tìm kiếm có độ phức tạp thời gian O(mn)

### **2.2 Trình bày thuật toán**

* Với mỗi kí tự trong bảng chữ cái, một thùng chứa (bucket) sẽ chứa tất cả các vị trí xuất hiện của kí tự đó trong xâu mẫu x. khi một kí tự xuất hiện k lần trong mẫu. bucket sẽ lưu k vị trí của kí tự đó. Khi mà xâu y chứa it kí tự hơn trong bản chữ cái thì sẽ có nhiều bucket rỗng.
* Quá trình xử lý của thuật toán Skip Search bao gồm việc tính các buckets cho tất cả các kí tự trong bảng chữ cái. for c in z[c] = {i : 0 i m-1 and x[i] = c}
* Thuật toán Skip Search có độ phức tạp bình phương trong trường hợp tồi nhất. nhưng cũng có trường hợp là O(n).

## **Horspool algorithm**

### **1.1 Đặc điểm**

* Là thuật toán đơn giản hơn của Boyer Moore
* Sử dụng dụng bad- character
* Dễ để cài đặt
* Pha cài đặt có độ phức tạp thuật toán là O(m+σ) và độ phức tạp bộ nhớ là O(σ)
* Pha thực thi có độ phức tạp là O(mxn)
* Số lượng so sánh trung bình cho một văn bản là trong khoảng 1/và 2/( +1)

### **1.2 Trình bày thuật toán**

* Quy tắc dịch bad- character được sử dụng trong Boyer Moore không có hiệu quả trong đoạn văn bản kí tự nhỏ, nhưng khi đoạn văn bản là lớn và phải so sánh với mẫu dài, nó thường trong trường hợp bảng bã ASCII và tìm kiếm thông thường trong soạn thảo văn bản, chúng rất h ữu dụng. Sử dụng nó như những kết quả riêng biệt lại rất hiệu quả.

**Thuật toán tìm kiếm Raita**, trước khi so sánh 2 mảng với nhau thì ta so sánh 2 kí tự đầu nếu giống nhau thì so sánh tiếp 2 kí tự giữa rồi đến 2 kí tự cuối tương ứng của 2 mảng trước

Thuật toán Raita là một thuật toán tìm kiếm chuỗi cải thiện hiệu suất của thuật toán Boyer – Moore – Horspool . Thuật toán này xử lý trước chuỗi đang được tìm kiếm cho mẫu, tương tự như thuật toán tìm kiếm chuỗi Boyer – Moore .Thuật toán Raita tìm kiếm một mẫu "P" trong một văn bản "T" cho trước bằng cách so sánh từng ký tự của mẫu trong văn bản đã cho. Việc tìm kiếm sẽ được thực hiện như sau. Cửa sổ cho văn bản "T" được định nghĩa là độ dài của "P".

Ký tự đầu tiên, ký tự cuối cùng của mẫu được so sánh với ký tự ngoài cùng bên phải của cửa sổ.

Nếu khớp, ký tự đầu tiên của mẫu sẽ được so sánh với ký tự ngoài cùng bên trái của cửa sổ.

Nếu chúng khớp lại, nó sẽ so sánh ký tự giữa của mẫu với ký tự giữa của cửa sổ.Wikipedia site:vi2.wiki

# **Thuật toán tìm kiếm Smith**

Kết hợp 2 thuật toán **Error! Bookmark not defined.** và Horspool, tại mỗi bước nhảy sẽ kiểm tra xem nhảy theo QuickSearch hay Horspool xa hơn thì chọn.