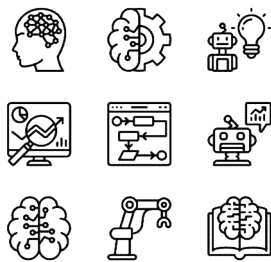


## Computer Science for Practicing Engineers

### So khớp các chuỗi (Pattern Matching)



TS. Huỳnh Bá Diệu  
Email: dieuhb@gmail.com  
Phone: 0914146868

### So khớp mẫu (Pattern Matching)

---

Nội dung:

1. Brute-force algorithm
2. Boyer-Moore algorithm
3. Knuth-Morris-Pratt algorithm

## String

---



- A string is a sequence of characters
- An alphabet  $\Sigma$  is the set of possible characters for a family of strings
- Example of alphabets:
  - ASCII
  - Unicode
  - $\{0, 1\}$
  - $\{A, C, G, T\}$

## String

---



Let  $P$  be a string of size  $m$

- A substring  $P[i..j]$  of  $P$  is the subsequence of  $P$  consisting of the characters with ranks between  $i$  and  $j$
- A prefix of  $P$  is a substring of the type  $P[0..i]$
- A suffix of  $P$  is a substring of the type  $P[i..m-1]$

## String

Given strings  $T$  (text) and  $P$  (pattern), the pattern matching problem consists of finding a substring of  $T$  equal to  $P$

Nêu các phần mềm hay ứng dụng mà bạn nghĩ là có sử dụng so sánh chuỗi???



## So khớp mẫu (Pattern Matching)

### Brute-Force Algorithm



- The brute-force pattern matching algorithm compares the pattern  $P$  with the text  $T$  for each possible shift of  $P$  relative to  $T$ , until either
  - a match is found, or
  - all placements of the pattern have been tried

**Algorithm** *BruteForceMatch*( $T, P$ )

**Input** text  $T$  of size  $n$  and pattern  $P$  of size  $m$

**Output** starting index of a substring of  $T$  equal to  $P$  or  $-1$  if no such substring exists

## So khớp mẫu (Pattern Matching)

### Brute-Force Algorithm

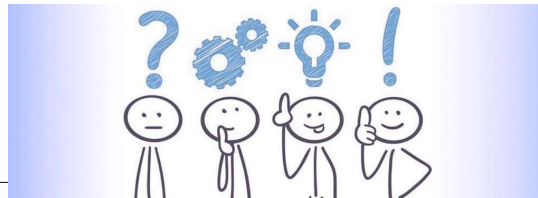


Algorithm *BruteForceMatch*( $T, P$ )

```

for  $i \leftarrow 0$  to  $n - m$ 
  { test shift  $i$  of the pattern }
   $j \leftarrow 0$ 
  while  $j < m \wedge T[i + j] = P[j]$ 
     $j \leftarrow j + 1$ 
  if  $j = m$  return  $i$  {match at  $i$ }
  else
    break while loop {mismatch}
return  $-1$  {no match anywhere}
  
```

Độ phức tạp của thuật toán là bao nhiêu???



## So khớp mẫu (Pattern Matching)

### Brute-Force Algorithm



- Brute-force pattern matching runs in time  $O(nm)$
- Example of worst case:
  - $T = aaa \dots ah$
  - $P = aaah$
  - may occur in images and DNA sequences
  - unlikely in English text



## So khớp mẫu (Pattern Matching)

### Brute-Force Algorithm



```
int BruteForceMatch(string T, string P) {
    int n = T.length(), m = P.length();
    for(int i=0; i<=n-m; i++) {
        int j=0;
        while( j < m && T[i + j] == P[j]) j++;
        if(j==m) return i;
    }
    return -1;
}
```

**Hãy thử nghiệm với hai chuỗi sau và cho biết kết quả???**

```
string pat = "ABABCABAB";
string text= "ABABDABACDABABCABAB";
```

## So khớp mẫu (Pattern Matching)

### The KMP Algorithm

Thuật toán Knuth-Morris-Pratt's so sánh mẫu theo thứ tự **left-to-right**, nhưng dịch chuyển các mẫu thông minh hơn phương pháp brute-force.

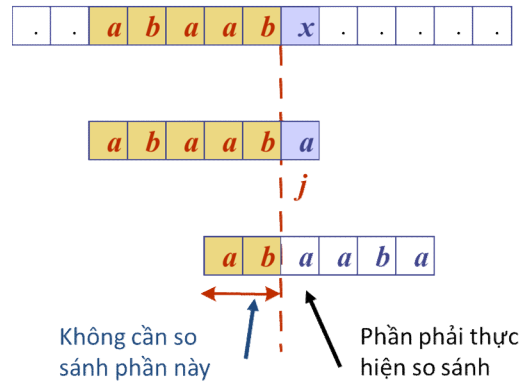
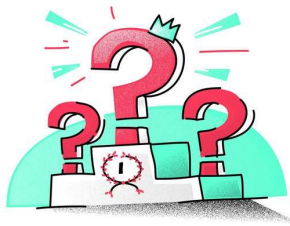
D.E. Knuth, J.H. Morris, V.R. Pratt: Fast Pattern Matching in Strings.  
SIAM Journal of Computing 6, 2, 323-350 (1977)

<https://www.inf.hs-flensburg.de/lang/algorithmen/pattern/kmpen.htm>

## So khớp mẫu (Pattern Matching)

### The KMP Algorithm - Motivation

Khi có ký tự không giống nhau, ta có thể dịch vị trí để bắt đầu so sánh về trước nhiều nhất là bao nhiêu để tránh các so sánh không cần thiết???

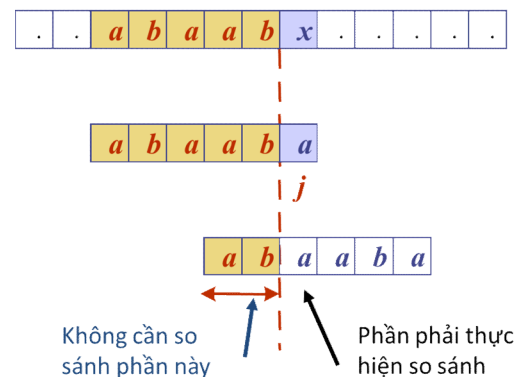


## So khớp mẫu (Pattern Matching)

### The KMP Algorithm - Motivation

*Khi có ký tự không giống nhau, ta có thể dịch vị trí để bắt đầu so sánh về trước nhiều nhất là bao nhiêu để tránh các so sánh không cần thiết?*

Answer: Tiền tố lớn nhất của  $P[0..j]$  là hậu tố của  $P[1..j]$  (the largest prefix of  $P[0..j]$  that is a suffix of  $P[1..j]$ )



## So khớp mẫu (Pattern Matching)

### KMP Failure Function

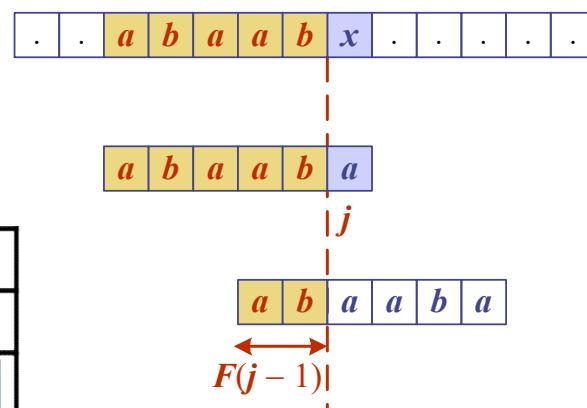
- Thuật toán Knuth-Morris-Pratt thực hiện tiền xử lý trên mẫu để tìm các tiền tố (preprocesses the pattern to find matches of prefixes of the pattern with the pattern itself)
- The **failure function**  $F(j)$  is defined as the size of the largest prefix of  $P[0..j]$  that is also a suffix of  $P[1..j]$

## So khớp mẫu (Pattern Matching)

### KMP Failure Function

Knuth-Morris-Pratt's algorithm modifies the brute-force algorithm so that if a mismatch occurs at  $P[j] \neq T[i]$  we set  $j \leftarrow F(j - 1)$

$j$	0	1	2	3	4	5
$P[j]$	$a$	$b$	$a$	$a$	$b$	$a$
$F(j)$	Hãy điền cho hàng $F(j)$ ???					

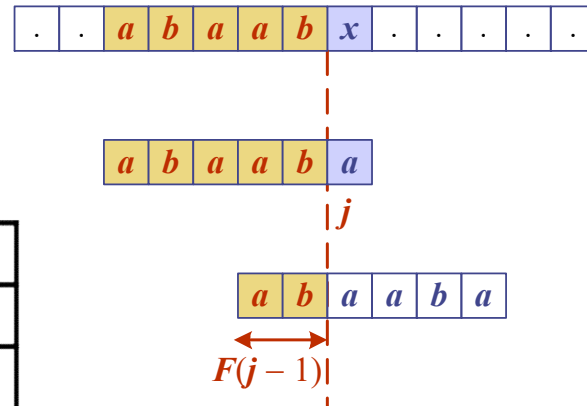


## So khớp mẫu (Pattern Matching)

### KMP Failure Function

Knuth-Morris-Pratt's algorithm modifies the brute-force algorithm so that if a mismatch occurs at  $P[j] \neq T[i]$  we set  $j \leftarrow F(j - 1)$

$j$	0	1	2	3	4	5
$P[j]$	$a$	$b$	$a$	$a$	$b$	$a$
$F(j)$	0	0	1	1	2	3



## So khớp mẫu (Pattern Matching)

### Computing the Failure Function



#### Algorithm *failureFunction*( $P$ )

```

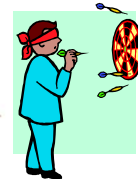
 $F[0] \leftarrow 0$ ;  $i \leftarrow 1$ ;  $j \leftarrow 0$ 
while  $i < m$ 
  if  $P[i] = P[j]$ 
    {we have matched  $j + 1$  chars}
     $F[i] \leftarrow j + 1$ ;  $i \leftarrow i + 1$ ;  $j \leftarrow j + 1$ 
  else if  $j > 0$  then
    {use failure function to shift  $P$ }
     $j \leftarrow F[j - 1]$ 
  else
     $F[i] \leftarrow 0$  { no match }
     $i \leftarrow i + 1$ 

```



## So khớp mẫu (Pattern Matching)

### Computing the Failure Function



```
void failureFunction(string P){
    F[0] = 0; int i = 1, j = 0;
    while (i < m)
        if( P[i] == P[j] ) {
            F[i] = j + 1; i = i + 1; j = j + 1;
        }
        else if ( j > 0 ) j = F[j - 1];
        else { F[i] = 0; i = i + 1; }
}
```

Hãy tính bảng *F* cho chuỗi *P* sau???

*P* = ACBNABACBNAQ

## So khớp mẫu (Pattern Matching)

### The KMP Algorithm

Cho biết độ phức tạp khi thực hiện tính hàm failure???



## So khớp mẫu (Pattern Matching)

### The KMP Algorithm

```

Algorithm KMPMatch(T, P)
  F ← failureFunction(P)
  i ← 0; j ← 0
  while i < n
    if T[i] = P[j]
      if j = m - 1
        return i - j { match }
      else
        i ← i + 1;    j ← j + 1
    else
      if j > 0 then j ← F[j - 1]
      else i ← i + 1
  return -1 { no match }

```

## So khớp mẫu (Pattern Matching)

### The KMP Algorithm

```

int KMPMatch(string T, string P) {
  n= T.length(); m= P.length();
  failureFunction(P);
  int i=0, j=0;
  while (i < n)
    if (T[i]==P[j])
      if (j== m - 1) return i-j; // có giống nhau
      else { i++; j++; }
    else if (j>0) j = F[j-1]; else i = i + 1;
  return -1; // không có chuỗi con giống nhau
}

```

```

char *txt =
"ABABDABACDABABCABAB";
char *pat = "ABABCABAB";
Lập trình thử nghiệm tìm cho
hai chuỗi trên???

```

## So khớp mẫu (Pattern Matching)

### The KMP Algorithm

Cho biết độ phức tạp của thuật toán KMP???

```

Algorithm KMPMatch(T, P)
  F ← failureFunction(P)
  i ← 0
  j ← 0
  while i < n
    if T[i] = P[j]
      if j = m - 1
        return i - j { match }
      else
        i ← i + 1
        j ← j + 1
    else
      if j > 0
        j ← F[j - 1]
      else
        i ← i + 1
  return -1 { no match }

```

## So khớp mẫu (Pattern Matching)

### The KMP Algorithm

- The failure function can be represented by an array and can be computed in  $O(m)$  time
- At each iteration of the while-loop, either
  - *i* increases by one, or
  - the shift amount *i* - *j* increases by at least one (observe that  $F(j - 1) < j$ )
- Hence, there are no more than  $2n$  iterations of the while-loop
- Thus, KMP's algorithm runs in optimal time  $O(m + n)$

## So khớp mẫu (Pattern Matching)

### Computing the Failure Function



```
void failureFunction(string P){
    F[0] = 0; int i =1, j=0;
    while (i < m)
        if( P[i] == P[j]) {
            F[i] = j + 1; i = i + 1; j = j + 1 ;
        }
        else if (j > 0 ) j = F[j - 1];
        else { F[i] = 0 ; i =i + 1;}
    cout<<"\n Bang F :\n";
    for(i=0; i<m; i++) cout<<F[i]<<" ";
}
```

```
string T, P;
int F[100], m, n;
int main()
{
    P="ACBNABACBNAQ";
    n= T.length(); m= P.length();
    failureFunction(P);
}
```

**Dự đoán kết quả của chương trình???**

## So khớp mẫu (Pattern Matching)

### Computing the Failure Function



"ACBNABACBNAQ"

Bang F tương ứng với chuỗi ACBNABACBNAQ:

0 0 0 0 1 0 1 2 3 4 5 0

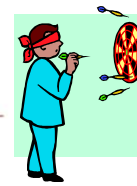
A	C	B	N	A	B	A	C	B	N	A	Q
0	0	0	0	1	0	1	2	3	4	5	0

**Cho biết bảng F với chuỗi sau???**

M	N	A	B	M	N	B	A	M	N	A	M

## So khớp mẫu (Pattern Matching)

### Computing the Failure Function

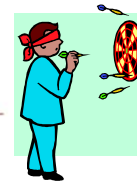


M	N	A	B	M	N	B	A	M	N	A	M

```
Bang F tuong ung voi chuoai MNABMNBAMNAM:
0 0 0 0 1 2 0 0 1 2 3 1
-----
Process exited after 0.01612 seconds with ret
```

## So khớp mẫu (Pattern Matching)

### Computing the Failure Function



**Thực hiện tính các bước so sánh cho hai chuỗi sau dựa vào bảng F???**

a	b	a	c	a	a	b	a	c	c	a	b	a	c	a	b	a	a	b	b
1	2	3	4	5	6														
a	b	a	c	a	b														

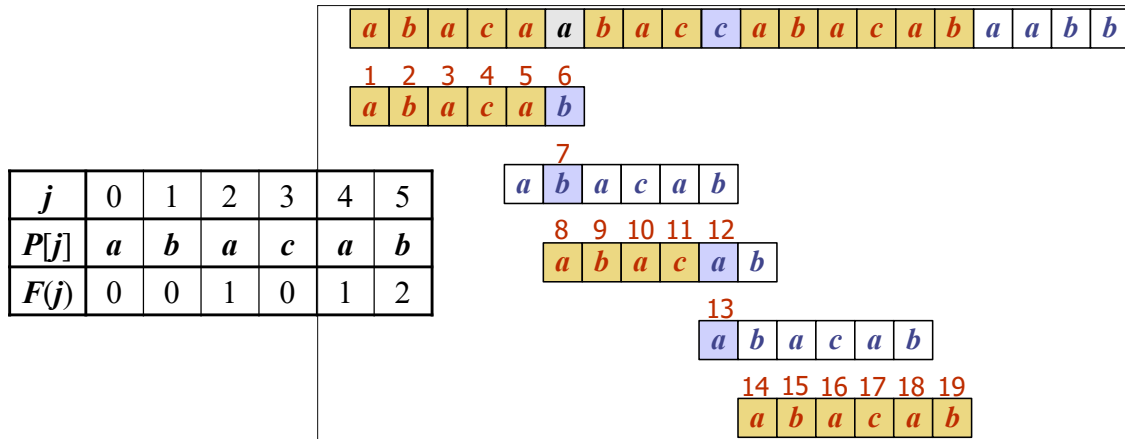
$j$	0	1	2	3	4	5
$P[j]$	$a$	$b$	$a$	$c$	$a$	$b$
$F(j)$	0	0	1	0	1	2

## So khớp mẫu (Pattern Matching)

### Computing the Failure Function



Các bước thực hiện so khớp:



## So khớp mẫu (Pattern Matching)

### Boyer-Moore Heuristics

The Boyer-Moore's pattern matching algorithm is based on two heuristics:

- Looking-glass heuristic (right-to-left matching): Compare  $P$  with a subsequence of  $T$  moving backwards
- Character-jump heuristic (bad character shift rule): When a mismatch occurs at  $T[i] = c$ 
  - + If  $P$  contains  $c$ , shift  $P$  to align the last occurrence of  $c$  in  $P$  with  $T[i]$
  - + Else, shift  $P$  to align  $P[0]$  with  $T[i + 1]$

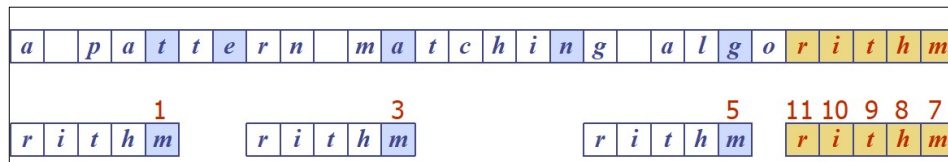
## So khớp mẫu (Pattern Matching)

### Boyer-Moore Heuristics

**Looking-glass heuristic (right-to-left matching):** Compare  $P$  with a subsequence of  $T$  moving backwards

**Character-jump heuristic (bad character shift rule):** When a mismatch occurs at  $T[i] = c$

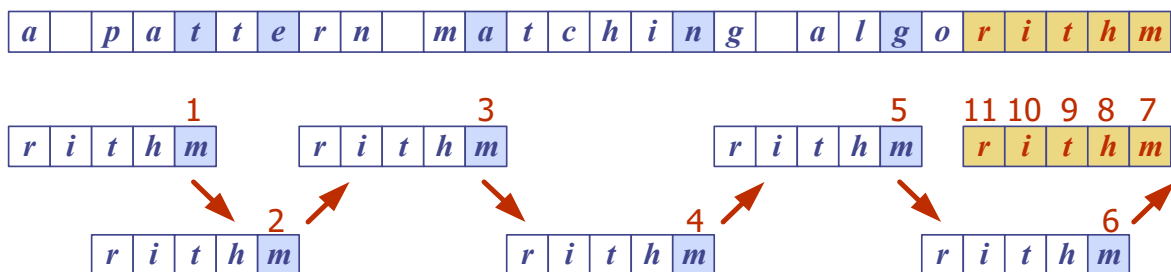
- If  $P$  contains  $c$ , shift  $P$  to align the last occurrence of  $c$  in  $P$  with  $T[i]$
- Else, shift  $P$  to align  $P[0]$  with  $T[i + 1]$



## So khớp mẫu (Pattern Matching)

### Boyer-Moore Heuristics

**Example :**



## So khớp mẫu (Pattern Matching)

### Boyer-Moore Heuristics

Text : A A B A A C A A D A A B A A B A

Pattern : A A B A

A A B A                      A A B A  
 A A B A A C A A D A A B A A B A  
 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15  
    A A B A

Pattern Found at 1, 9 and 12

## So khớp mẫu (Pattern Matching)

### Last-Occurrence Function

Boyer-Moore's algorithm preprocesses the pattern  $P$  and the alphabet  $\Sigma$  to build the last-occurrence function  $L$  mapping  $\Sigma$  to integers, where  $L(c)$  is defined as

- the largest index  $i$  such that  $P[i] = c$  or
- $-1$  if no such index exists

Example:

- $\Sigma = \{a, b, c, d\}$
- $P = abacab$

$c$	$a$	$b$	$c$	$d$
$L(c)$	4	5	3	-1



## So khớp mẫu (Pattern Matching)

### Last-Occurrence Function

Example:

–  $\Sigma = \{a, b, c, d\}$

–  $P = abacab$

$c$	$a$	$b$	$c$	$d$
$L(c)$	4	5	3	–1

The last-occurrence function can be represented by an array indexed by the numeric codes of the characters.

**Cho biết độ phức tạp của thuật toán khi tính hàm Last-Occurrence???**

## So khớp mẫu (Pattern Matching)

### Last-Occurrence Function

Example:

–  $\Sigma = \{a, b, c, d\}$

–  $P = abacab$

$c$	$a$	$b$	$c$	$d$
$L(c)$	4	5	3	–1

The last-occurrence function can be represented by an array indexed by the numeric codes of the characters

The last-occurrence function can be computed in time  $O(m + s)$ , where  $m$  is the size of  $P$  and  $s$  is the size of  $\Sigma$

## So khớp mẫu: The Boyer-Moore Algorithm

**Algorithm** *BoyerMooreMatch*( $T, P, \Sigma$ )

$L \leftarrow \text{lastOccurrenceFunction}(P, \Sigma)$

$i \leftarrow m - 1; \quad j \leftarrow m - 1$

**repeat**

**if**  $T[i] = P[j]$

**if**  $j = 0$  **return**  $i$  { match at  $i$  }

**else**

$i \leftarrow i - 1; \quad j \leftarrow j - 1$

**else**

    { character-jump }

$l \leftarrow L[T[i]]; \quad i \leftarrow i + m - \min(j, 1 + l); \quad j \leftarrow m - 1$

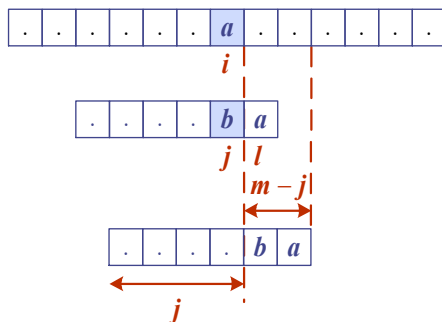
**until**  $i > n - 1$

**return**  $-1$  { no match }

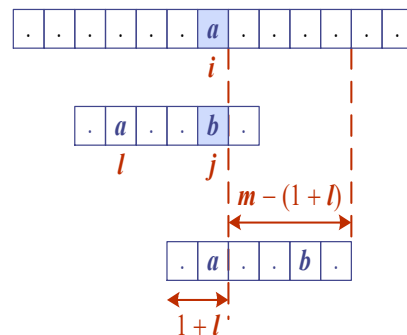
## So khớp mẫu (Pattern Matching)

### The Boyer-Moore Algorithm

Case 1:  $j \leq 1 + l$

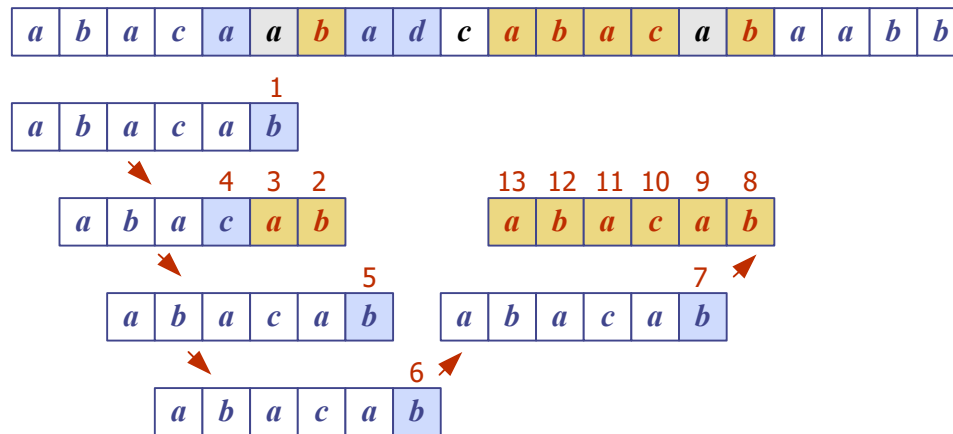


Case 2:  $1 + l \leq j$



## So khớp mẫu (Pattern Matching)

### The Boyer-Moore Algorithm



## So khớp mẫu (Pattern Matching) : Analysis

Boyer-Moore's algorithm runs in time  $O(nm + s)$

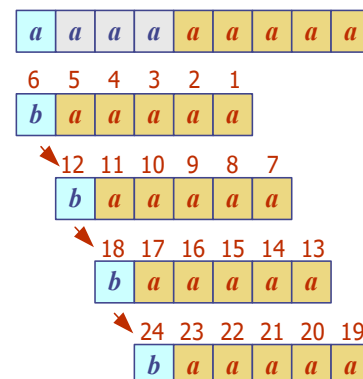
Example of worst case:

–  $T = aaa \dots a$

–  $P = baaa$

The worst case may occur in images and DNA sequences but is unlikely in English text

Boyer-Moore's algorithm is significantly faster than the brute-force algorithm on English text



## So khớp mẫu (Pattern Matching) : Analysis

---

### Thảo luận nhóm!!!

Vì sao trường hợp xấu nhất của thuật toán Boyer-Moore có thể xuất hiện khi so khớp ảnh hoặc chuỗi DNA?



## Boyer-Moore's algorithm: **Bad Character Heuristic**

---

The character of the text which doesn't match with the current character of the pattern is called the **Bad Character**. Upon mismatch, we shift the pattern until :

- 1) The mismatch becomes a match
- 2) Pattern P move past the mismatched character.


## Boyer-Moore's algorithm: **Bad Character Heuristic**

**Case 1 – Mismatch become match:** We will lookup the position of last occurrence of mismatching character in pattern and if mismatching character exist in pattern then we'll shift the pattern such that it get aligned to the mismatching character in text T.

## Boyer-Moore's algorithm: **Bad Character Heuristic**

**Case 1 – Mismatch become match:**

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
G	C	A	A	T	G	C	C	T	A	T	G	T	G	A	C	C
T	A	T	G	T	G											



0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
G	C	A	A	T	G	C	C	T	A	T	G	T	G	A	C	C
			T	A	T	G	T	G								

## Boyer-Moore's algorithm: **Bad Character Heuristic**

---

**Case 2: Pattern move past the mismatch character:**

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

G C A A T G C C T A T G T G A C C

○ T A T G T G

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

G C A A T G C C T A T G T G A C C

T A T G T G

## Boyer-Moore's algorithm: **Bad Character Heuristic**

---

**Cài đặt thuật toán Boyer-Moore tìm chuỗi văn bản???**

## Boyer-Moore's algorithm: **Bad Character Heuristic**

---

```
# define NO_OF_CHARS 256
void badCharHeuristic( string str, int size, int badchar[NO_OF_CHARS]) {
    for (int i = 0; i < NO_OF_CHARS; i++) badchar[i] = -1;
    for (int i = 0; i < size; i++) badchar[(int) str[i]] = i;
}
int main() {
    string txt= "ABAAABCD";    string pat = "ABC";
    search(txt, pat);
    return 0;
}
```

## Boyer-Moore's algorithm: **Bad Character Heuristic**

```
void search( string txt, string pat) {
    int m = pat.size(); int n = txt.size(); int s = 0;
    int badchar[NO_OF_CHARS]; badCharHeuristic(pat, m, badchar);
    while(s <= (n - m)) {
        int j = m - 1;
        while(j >= 0 && pat[j] == txt[s + j]) j--;
        if (j < 0) {
            cout << "pattern occurs at shift = " << s << endl;
            s += (s + m < n)? m - badchar[txt[s + m]] : 1;
        } else s += max(1, j - badchar[txt[s + j]]);
    }
}
```

## Các nội dung cần đọc trước [review FC2]

- Danh sách liên kết (Đơn, Đôi, Vòng)
- Ngăn xếp (Stack)
- Hàng đợi (Queue)
- Cây (Tree)
- Bảng băm (Hash Table)



## Tài liệu đọc thêm về các thuật toán so khớp chuỗi

---

<http://www-igm.univ-mlv.fr/~lecroq/string/node22.html>

<https://www.topcoder.com/community/competitive-programming/tutorials/introduction-to-string-searching-algorithms/>

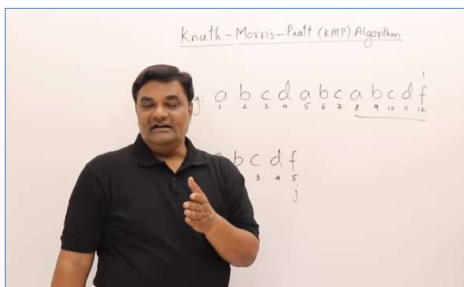
<https://study.com/academy/lesson/string-searching-algorithms-methods-types.html>

## Link YouTube

---

<https://www.youtube.com/watch?v=V5-7GzOfADQ>

<https://www.youtube.com/watch?v=PHXAOKQk2dw>



### Boyer Moore Horspool Analysis

- Worst case same as naïve example:
  - 1<sup>st</sup> input text (length  $n$ )
  - 0111...1 pattern (length  $m$ )
- Worst Case  $O(nm)$
- Best case
  - 1<sup>st</sup> input text (length  $n$ )
  - 0<sup>th</sup> pattern (length  $m$ )