

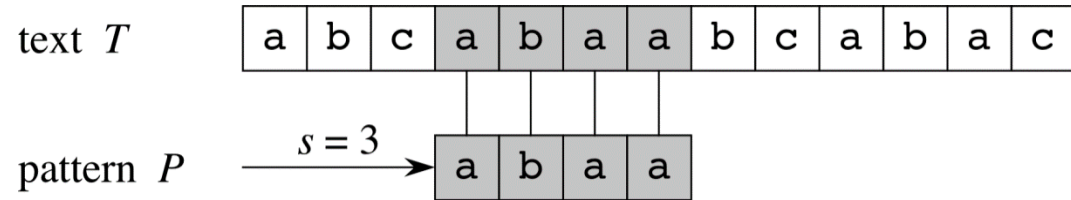
# STRING MATCHING

---

Prof. Michael Tsai

2017/03/21

# 問題：字串比對



- 陣列  $T[1..n]$  中有一個長度為  $n$  的字串
- 陣列  $P[1..m]$  中有一個長度為  $m$  的字串
- $m \leq n$
- 要在  $T$  中找  $P$  是否出現
- $P$  和  $T$  的字串從一個字元的集合  $\Sigma$  中拿出
- 如:  $\Sigma = \{0,1\}$  或  $\Sigma = \{a, b, \dots, z\}$
- Pattern  $P$  **occurs with shift  $s$**  in text  $T$   
(Pattern  $P$  occurs beginning at position  $s+1$  in text  $T$ )  
if  $T[s + j] = P[j]$ , for  $1 \leq j \leq m$ .
- If  $P$  occurs with shift  $s$  in  $T$ , we call  $s$  a **valid shift**. Otherwise, we call  $s$  an **invalid shift**.
- 字串比對問題是要在  $T$  中間找到**所有** $P$ 出現的位置(valid shift)

# 一些定義

- $\Sigma^*$ : 所有使用 $\Sigma$ 中字元組成的有限長度字串(包括長度為0的空字串)
- $|x|$ : 字串 $x$ 的長度
- $xy$ : 把字串 $x$ 和 $y$ 接起來 (concatenation)
- $w \sqsubset x$ : 字串 $w$ 是字串 $x$ 的prefix (也就是 $x=wy, y \in \Sigma^*$ )  
( $w \sqsubset x$  表示  $|w| \leq |x|$ )
- $w \sqsupset x$ : 字串 $w$ 是字串 $x$ 的suffix (也就是 $x=yw, y \in \Sigma^*$ )  
( $w \sqsupset x$  表示  $|w| \leq |x|$ )
- 例如:  $ab \sqsubset abcca, cca \sqsupset abcca$
- 空字串 $\epsilon$ 為任何字串的prefix & suffix
- 對任何字串 $x, y$ 和字元 $a, x \sqsupset y$  iff  $xa \sqsupset ya$
- $\sqsubset$ 和 $\sqsupset$ 為transitive(具遞移律)的operator

# 方法一：笨蛋暴力法

Native-String-Matcher( $T, P$ )

$N = T.length$

$M = P.length$

for  $s = 0$  to  $n - m$

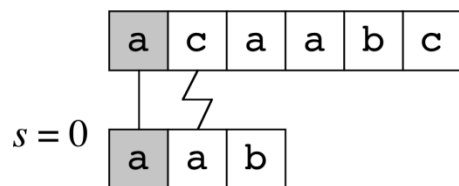
$n - m + 1$ 次

if  $P[1..m] == T[s+1..s+m]$

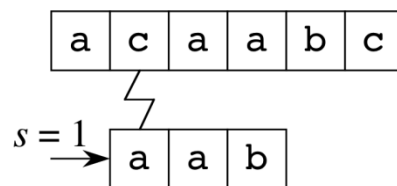
print "Pattern occurs with shift"  $s$

$O(m)$

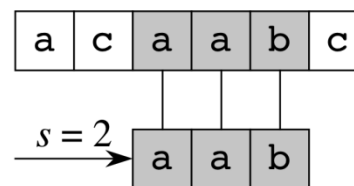
$O((n - m + 1)m)$



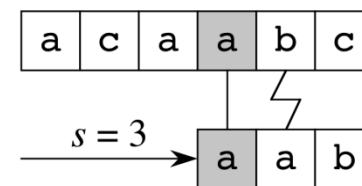
(a)



(b)



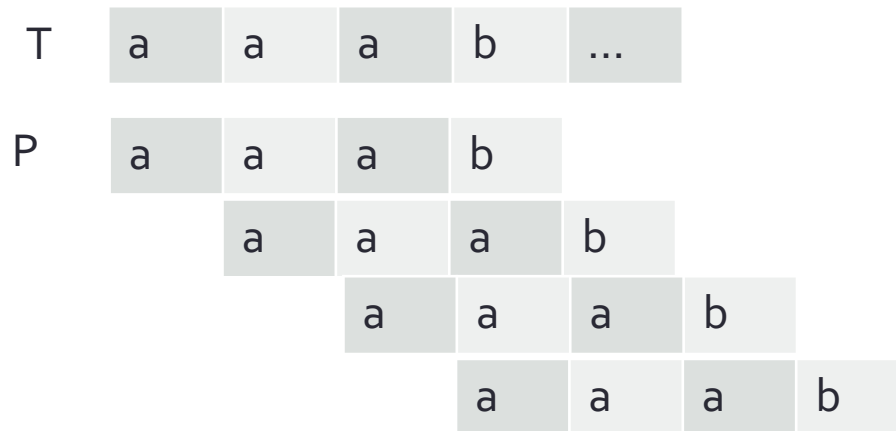
(c)



(d)

# 為什麼不好?

- 因為每次for執行比對, 如果錯了, 這回合的資訊完全丟掉.
- 例: P=aaab
- 如果我們發現s=0是valid shift (表示T開頭為aaab),
- 那麼從之前的結果應該可以知道shift 1, 2, 3 都可以直接跳過, 不需要一一比對.



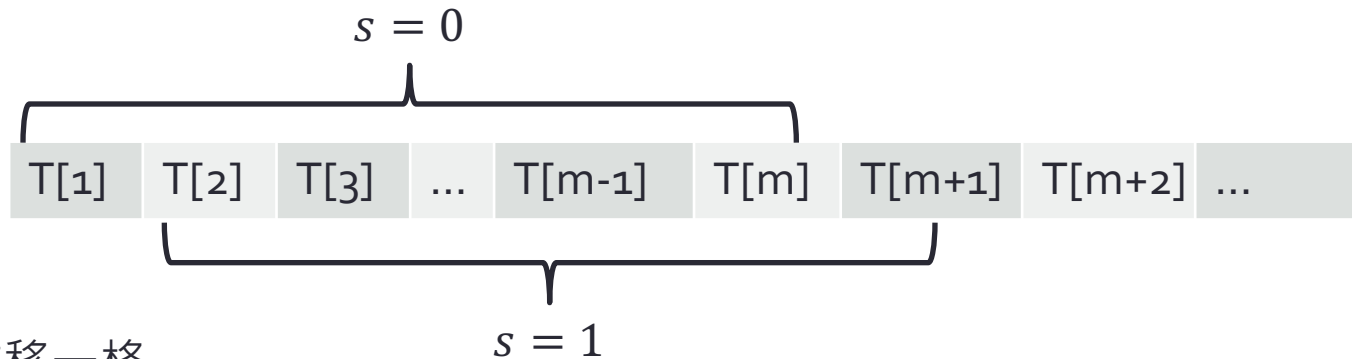
## 方法二：The Rabin-Karp Algorithm

- 假設  $\Sigma = \{0, 1, \dots, 9\}$
- 那麼每個長度為  $k$  的字串可以想成是一個  $k$  位數的十進位數
- 例如字串 "31415" 可以想成是十進位數 31415
- 把  $t_s$  設為代表  $T[s+1..s+m]$  的十進位數
- $p$  設為代表  $P$  的十進位數
- 那麼  $t_s = p$  iff  $T[s+1..s+m] = P[1..m]$ , 也就是  $s$  是 valid shift
- 怎麼從  $P$  計算  $p$  呢?
 

P[1]	P[2]	P[3]	...	P[m-1]	P[m]
------	------	------	-----	--------	------
- $p = P[m] + 10(P[m-1] + 10(P[m-2] + \dots + 10(P[2] + 10P[1]) \dots))$

## 方法二：The Rabin-Karp Algorithm

- 怎麼從P計算p呢?
- $t_0 = T[m] + 10(T[m-1] + 10(T[m-2] + \dots + 10(T[2] + 10T[1]) \dots))$



整個往右移一格

$$t_{s+1} = 10(t_s - 10^{m-1}T[s+1]) + T[s+m+1]$$

拿掉最左邊那一格

加上最右邊那一格

## 方法二：The Rabin-Karp Algorithm

- 那麼用這個方法要多花少時間呢? (簡易分析版)
- $p = P[m] + 10(P[m - 1] + 10(P[m - 2] + \dots + 10(P[2] + 10P[1]) \dots))$   $O(m)$
- $t_0 = T[m] + 10(T[m - 1] + 10(T[m - 2] + \dots + 10(T[2] + 10T[1]) \dots))$   $O(m)$
- 然後用  $t_{s+1} = 10(t_s - 10^{m-1}T[s + 1]) + T[s + m + 1]$ , 算  $t_1, t_2, \dots, t_{n-m}$  的值 (每次都是constant time, 共n-m次)
- 所以總共:  $O(m)$  preprocessing時間,  $O(n - m)$  比對時間



## 方法二：The Rabin-Karp Algorithm

• 之前的兩個問題：

1.  $\Sigma$ 如果是general的character set, 怎麼辦? (不再是 $\{0,1,\dots,9\}$ )

如何解決呢?

假設 $|\Sigma| = d$ ,  $\Sigma = \{c_1, c_2, c_3, \dots, c_d\}$ . 可以把之前的式子改成

$$\begin{aligned}
 & p \\
 &= \text{idx}(P[m]) \\
 &+ d(\text{idx}(P[m-1]) \\
 &+ d(\text{idx}(P[m-2]) + \dots + d(\text{idx}(P[2]) + d \cdot \text{idx}(P[1])) \dots))
 \end{aligned}$$

(a) 把整個string看成一個d進位的數.

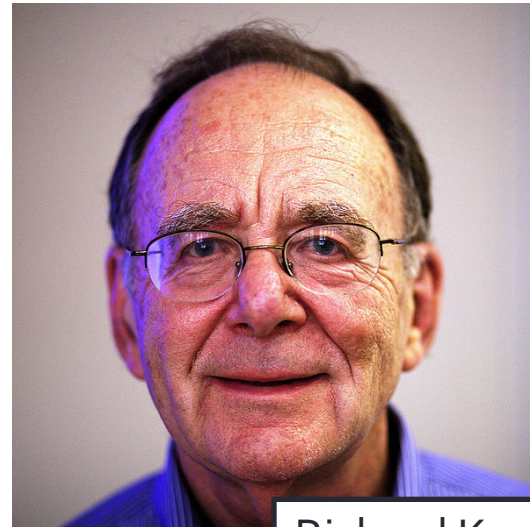
(b) 字元在 $\Sigma$ 中index當作該字元所代表的值

## 方法二：The Rabin-Karp Algorithm

2. 當 $m$ 比較大的時候,  $p$ 和 $t_s$ 將很難用電腦直接處理 (用long long也存不下)
  - 加一加乘一乘最後總是會overflow
- 如何解決? 利用同餘理論.



Michael Rabin



Richard Karp

# 同餘理論 (Modular Arithmetic)

- 假設 $a, b$ 都為整數
- $a \equiv b \pmod{n}$ : 表示 $a$ 和 $b$ 除以 $n$ 的餘數相等
- 例如:
  - $38 \equiv 14 \pmod{12}$
  - $0 \equiv 12 \pmod{12}$
  - $-13 \equiv -3 \pmod{5}$
- 更棒的性質:
- $a_1 \equiv b_1 \pmod{n}$
- $a_2 \equiv b_2 \pmod{n}$
- 則
  1.  $a_1 + a_2 \equiv b_1 + b_2 \pmod{n}$
  2.  $a_1 - a_2 \equiv b_1 - b_2 \pmod{n}$
  3.  $a_1 a_2 \equiv b_1 b_2 \pmod{n}$

# 同餘理論 (Modular Arithmetic)

- $a_1 \equiv b_1 \pmod{n}$
- $a_2 \equiv b_2 \pmod{n}$
- 則
- $a_1 + a_2 \equiv b_1 + b_2 \pmod{n}$
- 證明:
- $a_1 \equiv b_1 \pmod{n}$  表示
  - $a_1 = c_{a_1}n + r_1$
  - $b_1 = c_{b_1}n + r_1$
- $a_2 \equiv b_2 \pmod{n}$  表示
  - $a_2 = c_{a_2}n + r_2$
  - $b_2 = c_{b_2}n + r_2$
- 所以
  - $a_1 + a_2 = (c_{a_1} + c_{a_2})n + (r_1 + r_2)$
  - $b_1 + b_2 = (c_{b_1} + c_{b_2})n + (r_1 + r_2)$
- 兩者餘數相同!
- $a_1 + a_2 \equiv b_1 + b_2 \pmod{n}$  得證.

# 同餘理論 (Modular Arithmetic)

- $a_1 \equiv b_1 \pmod{n}$
- $a_2 \equiv b_2 \pmod{n}$
- 則
- $a_1 a_2 \equiv b_1 b_2 \pmod{n}$
- 證明:
- $a_1 \equiv b_1 \pmod{n}$  表示
  - $a_1 = c_{a_1} n + r_1$
  - $b_1 = c_{b_1} n + r_1$
- $a_2 \equiv b_2 \pmod{n}$  表示
  - $a_2 = c_{a_2} n + r_2$
  - $b_2 = c_{b_2} n + r_2$
- 所以
  - $a_1 a_2$ 

$$= (c_{a_1} n + r_1)(c_{a_2} n + r_2)$$

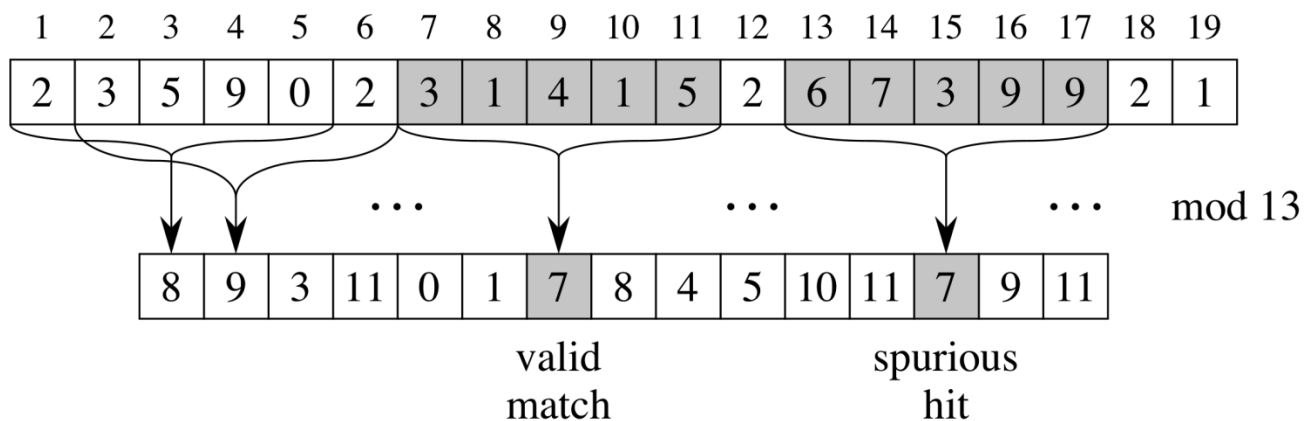
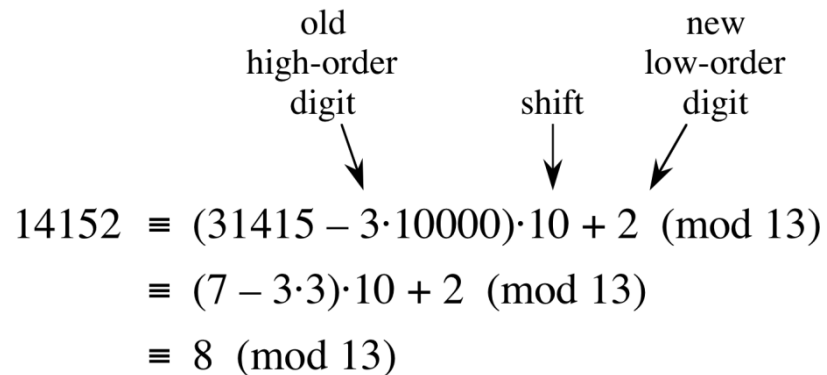
$$= (c_{a_1} c_{a_2} n + c_{a_1} r_2 + c_{a_2} r_1)n + r_1 r_2$$
  - $b_1 b_2 = (c_{b_1} c_{b_2} n + c_{b_1} r_2 + c_{b_2} r_1)n + r_1 r_2$
- 兩者餘數相同! 得證!

# The Rabin-Karp Algorithm 修正版

- 取 $q$ 使得 $dq$ 可以用一個電腦word (32-bit or 64-bit)來表示
- 既然 $\text{mod}$ 後再加, 減, 乘也會保持原本的關係, 我們可以把這些operation都變成 $\text{mod}$ 版本的
- $p = P[m] + d(P[m-1] + d(P[m-2] + \dots + d(P[2] + dP[1]) \dots)) \pmod{q}$
- $t_0 = T[m] + d(T[m-1] + d(T[m-2] + \dots + d(T[2] + dT[1]) \dots)) \pmod{q}$
- $t_{s+1} = d(t_s - d^{m-1}T[s+1]) + T[s+m+1] \pmod{q}$

# The Rabin-Karp Algorithm 修正版

- 新的mod版algorithm會造成一個問題:
- 雖然 $t_s = p \Rightarrow t_s \equiv p \pmod{q}$
- 但 $t_s = p \not\Leftrightarrow t_s \equiv p \pmod{q}$
- 例如 $38 \equiv 14 \pmod{12}$ ,  $38 \neq 14$
- 但是如果 $t_s \not\equiv p \pmod{q} \Rightarrow t_s \neq p$
- 所以演算法變成這樣:
  1. 如果 $t_s \not\equiv p \pmod{q}$ , 那麼現在這個s為invalid shift
  2. 如果 $t_s \equiv p \pmod{q}$ , 那麼必須額外檢查  
(直接比對範圍內的字串→花很多時間)
- 當 $t_s \equiv p \pmod{q}$ , 但是 $t_s \neq p$ 時, 稱為spurious hit
- 當q夠大的時候, 希望spurious hit會相當少





# Pseudo Code: Rabin-Karp

Rabin-Karp-Matcher (T, P, d, q)

n=T.length

m=P.length

$h = d^{m-1} \bmod q$

p=0

t=0

for i=1 to m

$p = (dp + P[i]) \bmod q$

$t = (dt + T[i]) \bmod q$

for s=0 to n-m

迴圈跑n-m+1次

    if p==t

        if  $P[1..m] == T[s+1..s+m]$

Hit的時候比對:  $O(m)$

        print "Pattern occurs with shift" s

    if s<n-m

$t = (d(t - T[s+1]h) + T[s+m+1]) \bmod q$

T: string to be searched

P: pattern to be matched

d: size of the character set

q: max number

Pre-processing:  $O(m)$

# Worst-case Running Time

Rabin-Karp-Matcher ( $T, P, d, q$ )

$n = T.length$

$m = P.length$

$h = d^{m-1} \bmod q$

$p = 0$

$t = 0$

for  $i = 1$  to  $m$

$p = (dp + P[i]) \bmod q$

$t = (dt + T[i]) \bmod q$

for  $s = 0$  to  $n - m$

迴圈跑  $n - m + 1$  次

if  $p == t$

if  $P[1..m] == T[s+1..s+m]$

Hit的時候比對:  $O(m)$

print "Pattern occurs with shift"  $s$

if  $s < n - m$

$t = (d(t - T[s+1]h) + T[s+m+1]) \bmod q$

Worst case的時候:

$T = a^n$  ( $n$ 個 $a$ )

$P = a^m$  ( $m$ 個 $a$ )

比對的時間為

$O(m(n - m + 1))$



Pre-processing:  $O(m)$

# Average Running Time

```
Rabin-Karp-Matcher (T, P, d, q)
n=T.length
m=P.length
h= $d^{m-1} \bmod q$ 
p=0
```

Pre-processing:  $O(m)$

```
for i=1 to m
```

```
    p=(dp+P[i]) mod q
```

迴圈跑 $n-m+1$ 次

```
    t=(dt+T[i]) mod q
```

```
for s=0 to n-m
```

```
    if p==t
```

Hit的時候比對:  $O(m)$

```
        if P[1..m]==T[s+1..s+m]
```

```
            print "Pattern occurs with shift" s
```

```
    if s<n-m
```

```
        t=(d(t-T[s+1]h)+T[s+m+1]) mod q
```

平常的時候, valid shift很少  
(假設有 $c$ 個)

不會每次都有modulo的hit.

假設字串各種排列組合出現的機率相等  
則spurious hit的機率可當成 $1/q$ .

則比對花的時間:

Spurious hit共花 $O((n-m+1)/q)=O(n/q)$ 次

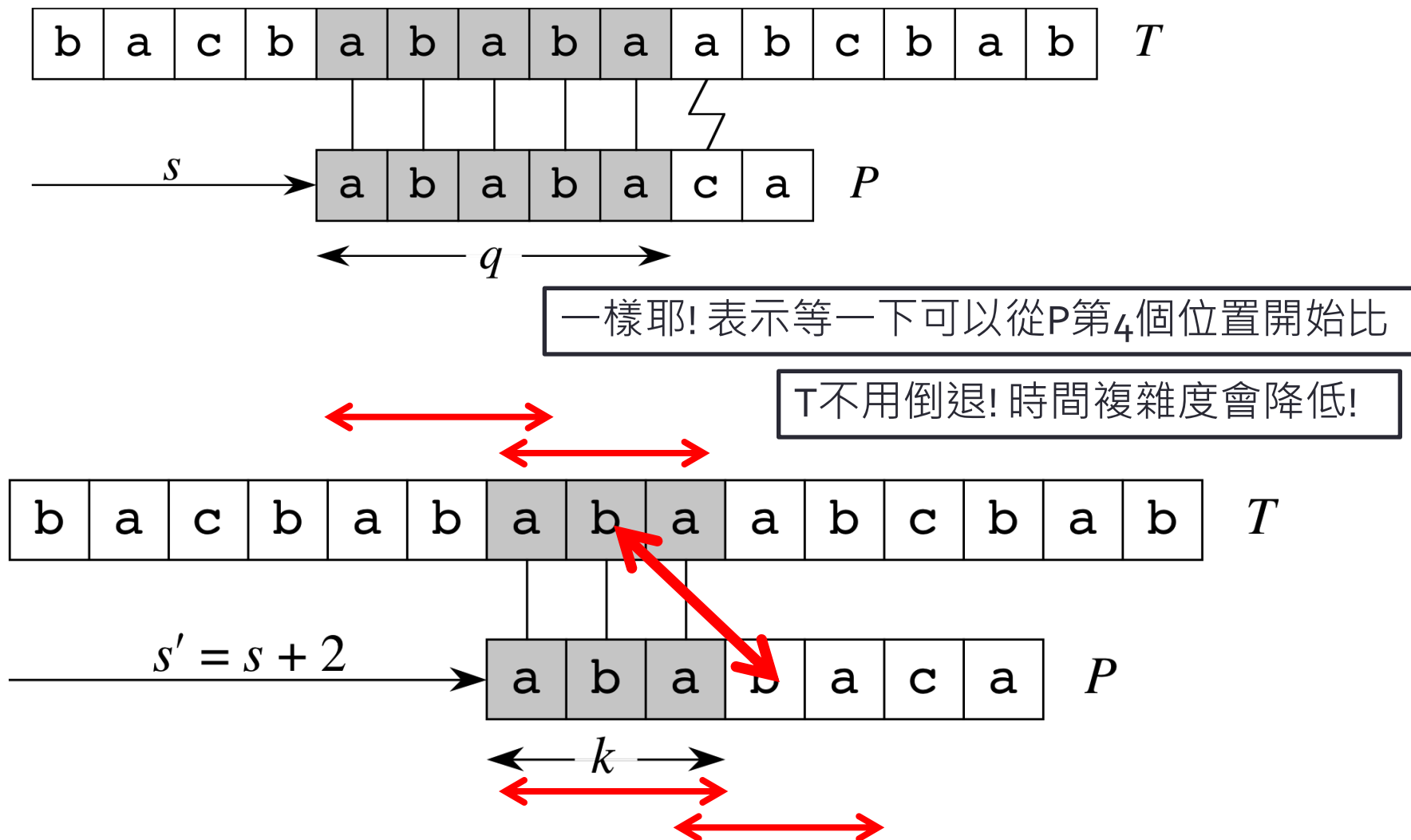
總共比對花的時間為

$O((n-m+1)+(m(c+n/q)))$

If  $c=O(1)$  and  $q \geq m$ ,  $\rightarrow O(n+m)=O(n)$



# 方法三：The Knuth-Morris-Pratt Algo.



# Knuth-Morris-Pratt



Don Knuth



James Morris

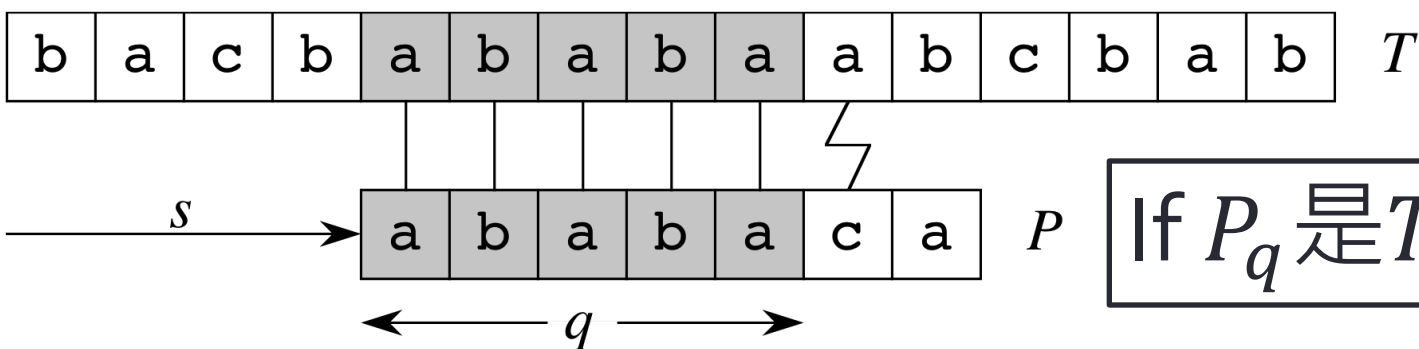


Vaughan Pratt

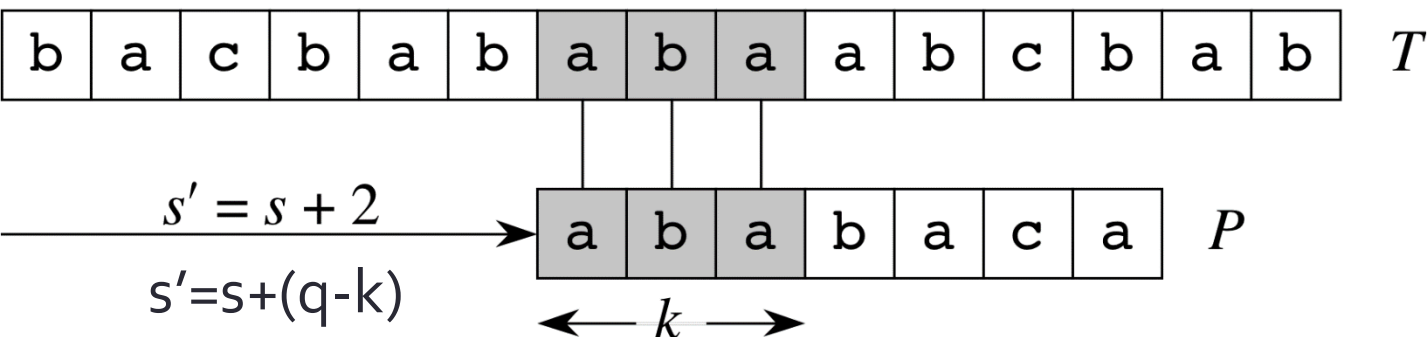
# 正式一點的說法

$$P_q : P[1..q]$$

- 假設  $P[1..q]$  和  $T[s+1..s+q]$  已經 match 了
- 要找出最小的 shift  $s'$  使得某個  $k < q$  可以滿足
- $P[1..q] = T[s' + 1..s' + k], s' + k = s + q$

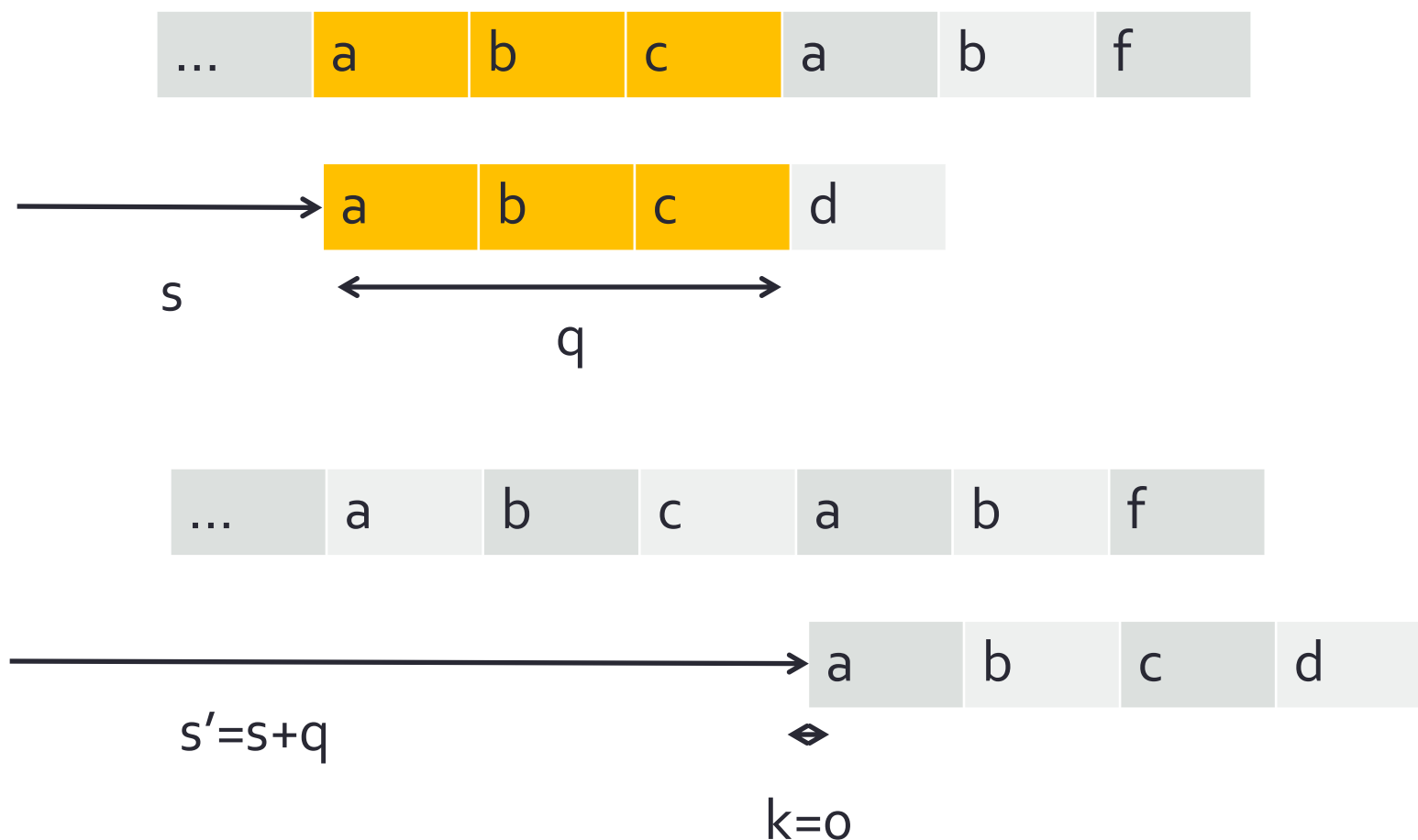


If  $P_q$  是  $T_{s+q}$  的 suffix



找  $P_q$  的最長 prefix 使得它是  $T_{s+q}$  的 suffix

# 最好的狀況下：沒有重複的pattern



## 先處理P來取得”重複pattern”的資訊

找 $P_q$ 的最長prefix使得它是 $T_{s+q}$ 的suffix

找 $P_q$ 的最長prefix  $P_k$  使得它是 $P_q$ 的suffix

- 定義Prefix function (failure function)  $\pi$ :
- Input:  $\{1, 2, \dots, m\}$
- Output:  $\{0, 1, \dots, m-1\}$
- $\pi[q] = \max\{k: k < q \text{ and } P_k \text{ is a suffix of } P_q\}$
- (也就是前面例子中的k值, 可以想成最長的重複pattern的長度)



# Prefix function example

$$\pi[q] = \max\{k: k < q \text{ and } P_k \text{ is a suffix of } P_q\}$$

i	1	2	3	4	5	6	7
P[i]	A	B	A	B	A	C	A
$\pi[i]$							
$\pi[i]$	0	0	1	2	3	0	1

[illegible]

# Pseudo-code: Prefix function

Compute-Prefix-Function( $P$ )

$m = P.length$

let  $\pi[1..m]$  be a new array

$\pi[1] = 0$

$k = 0$

for  $q = 2$  to  $m$

    while  $k > 0$  and  $P[k+1] \neq P[q]$

$k = \pi[k]$

    if  $P[k+1] == P[q]$

$k = k + 1$

$\pi[q] = k$

return  $\pi$

# 例子：Matching

- Ex. 1:  $T = \text{BACBABABAABCBAAB}$
- Ex. 2:  $T = \text{BABABABABACA}$

i	1	2	3	4	5	6	7
P[i]	A	B	A	B	A	C	A
$\pi[i]$	0	0	1	2	3	0	1

- 實際的Matching Pseudo Code和計算prefix function非常像
- 請見Cormen p. 1005 KMP-Matcher

# 算Prefix function花多少時間?

Compute-Prefix-Function(P)

$m = P.length$

let  $\pi[1..m]$  be a new array

$\pi[1] = 0$

$k = 0$

for  $q = 2$  to  $m$  共 $O(m)$ 次

    while  $k > 0$  and  $P[k+1] \neq P[q]$

$k = \pi[k]$

    if  $P[k+1] == P[q]$

$k = k + 1$

$\pi[q] = k$

return  $\pi$

麻煩的是這邊：  
總共會跳多少次呢?

# 算Prefix function花多少時間?

Compute-Prefix-Function(P)

$m = P.length$

let  $\pi[1..m]$  be a new array

$\pi[1] = 0$

$k = 0$

for  $q = 2$  to  $m$

while  $k > 0$  and  $P[k+1] \neq P[q]$

$k = \pi[k]$

if  $P[k+1] == P[q]$

$k = k + 1$

$\pi[q] = k$

return  $\pi$

進入迴圈的時候  $k < q$ , 且  $q$  每次增加,  $k$  有時候不增加  
所以  $k < q$  永遠成立

所以  $\pi[q] = k < q$ .

所以每執行一次迴圈  
就減少  $k$  一次  
且  $k$  永遠不是負的

$k$  只會在這邊增加,  
因此最多總共增加  $m-1$  次(迴圈執行次數)

最後: 既然有增加才有得減少, while  
loop 總共執行的次數不會超過  $O(m)$

Total:  
 $O(m)$

# KMP執行時間

- 類似的方法可以證明比對的部分執行時間為 $O(n)$
- 所以總和來看:
- Preprocessing時間  $O(m)$
- 比對時間  $O(n)$



# Reading Assignment

- Textbook (Cormen)  
ch. 32, 32.1, 32.2, 32.4 (正確性的證明略為複雜)