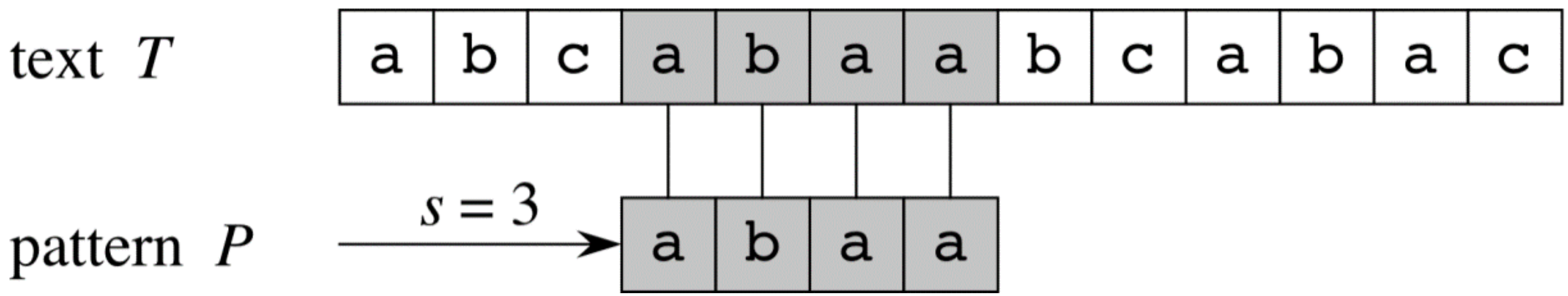
STRING MATCHING

Prof. Michael Tsai

2025/04

問題定義： 字串比對



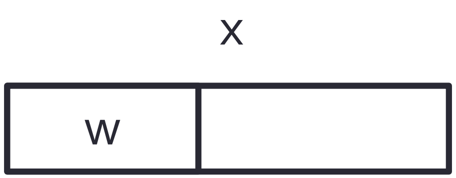
- 陣列T[1:n]中有一個長度為n的字串
- 陣列P[1:m]中有一個長度為m的字串 $m \le n$
- P和T的字串從一個字元的集合Σ中拿出
  ($\Sigma = \{0,1\}$ 或 $\Sigma = \{a, b, \dots, z\}$)
- 在T中間找到**所有**P出現的位置 (valid shifts)
- Pattern P **occurs with shift s** in text T (beginning at position s+1)
  ➔if $T[s+j] == P[j]$, for $1 \le j \le m$.
- If P occurs with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

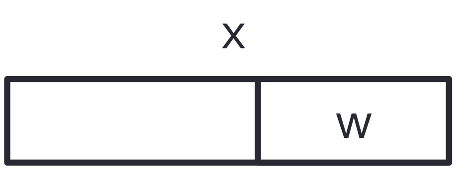
一些定義

- Σ^*: 所有使用Σ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 $|w| \le |x|$)
- $w \sqsupset x$: 字串w是字串x的suffix (也就是x=yw, $y \in \Sigma^*$)

  ($w \sqsupset x$ 表示 $|w| \le |x|$)
- 例: ab ⊏abcca, cca ⊐ abcca





一些定義

- 空字串ϵ為任何字串的prefix & suffix
- 對任何字串x, y和字元a, $x \sqsupset y$ iff $xa \sqsupset ya$
- ⊏和⊐為transitive(具遞移律)的operator, e.g., $x \sqsupset y \ and \ y \sqsupset z \Rightarrow x \sqsupset z$

iff means “if and only if”

Running Time of Different String-Matching Algorithms

Algorithm	Preprocessing time	Matching time
Naïve	0	$O((n-m+1)m)$
Rabin-Karp	$\Theta(m)$	$O((n-m+1)m)$
Knuth-Morris-Pratt	$\Theta(m)$	$O(n)$

方法一：暴力法（Naïve）

Cormen 32.1

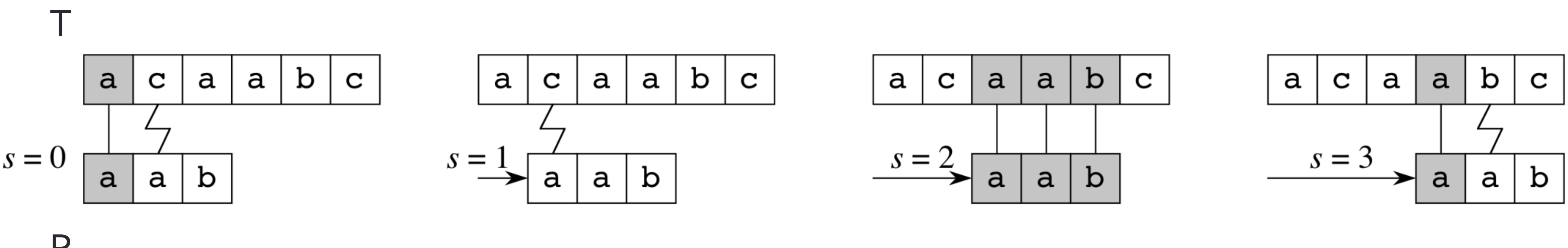
```
Native-String-Matcher(T, P, n, m)

/* n=T.length
   m=P.length */
for s=0 to n-m
      if P[1:m]==T[s+1:s+m]
            print “Pattern occurs with shift” s
```

n-m+1次

$O(m)$

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



(a) (b) (c) (d)

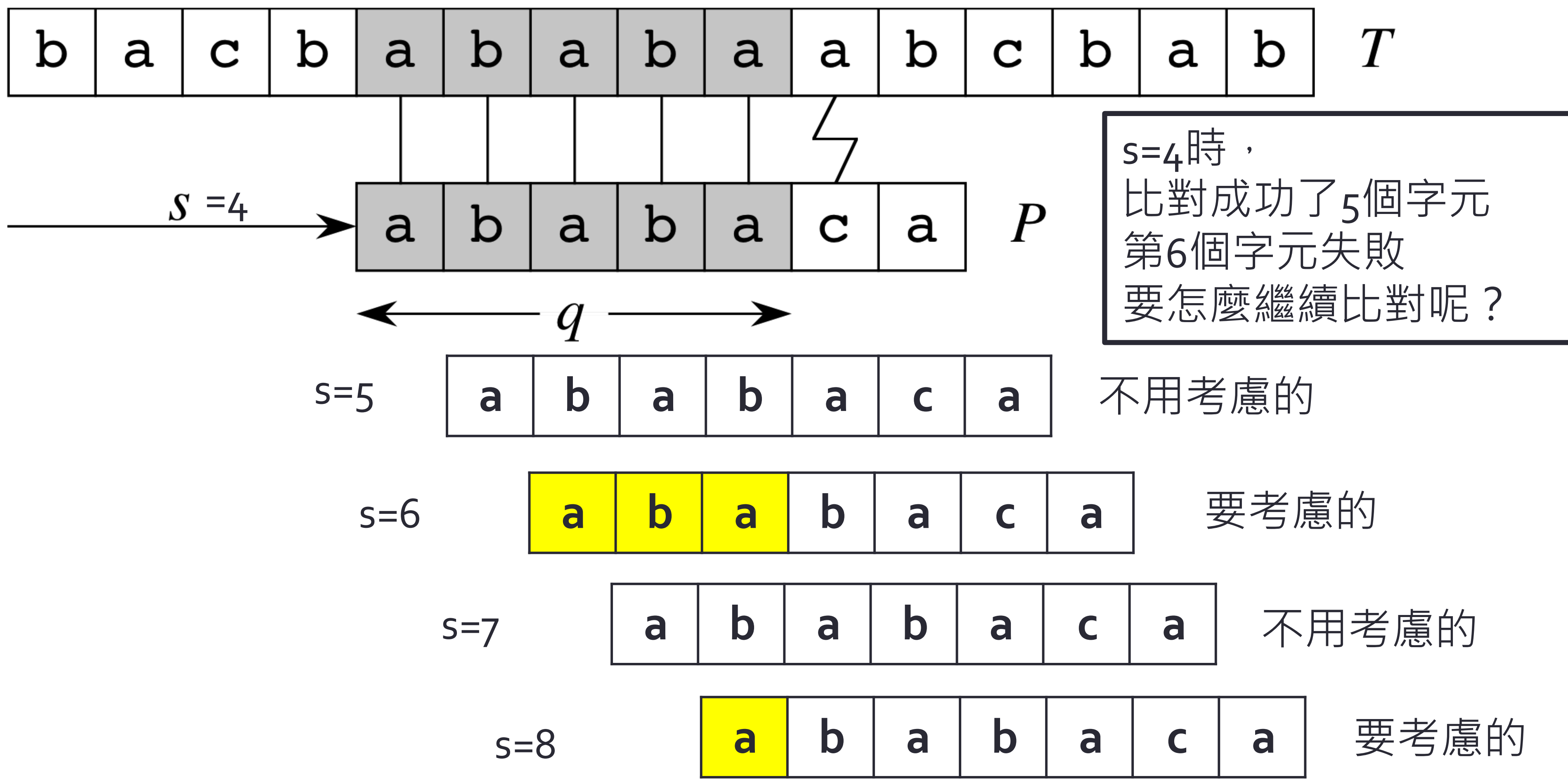
暴力法浪費時間的地方?

- 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.
- 這個演算法應該要有一些「記憶」

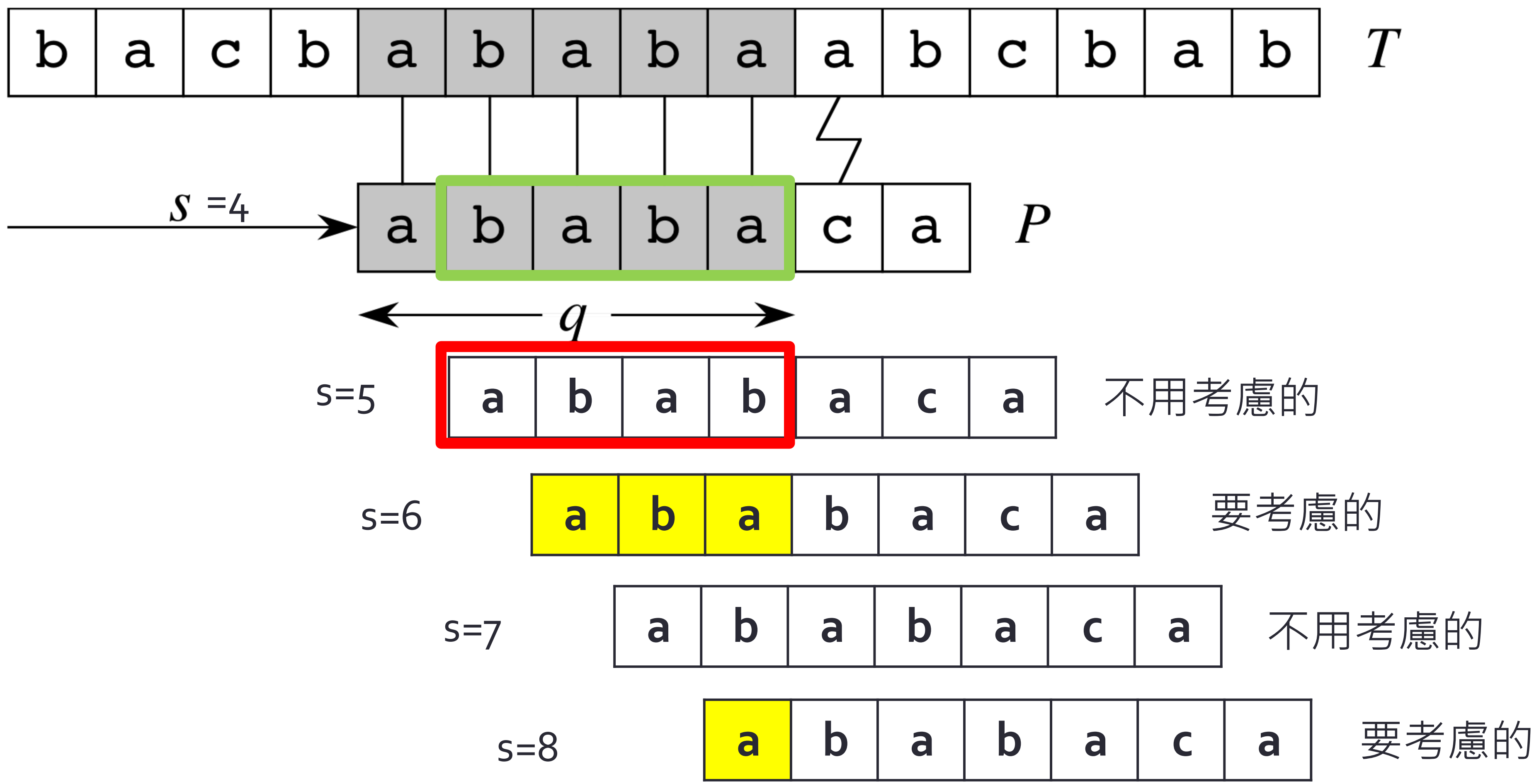
T	a	a	a	b	...			
P	a	a	a	b				
s=1		a	a	a	b			
s=2			a	a	a	b		
s=3				a	a	a	b	
s=4					a	a	a	b

Cormen 32.4

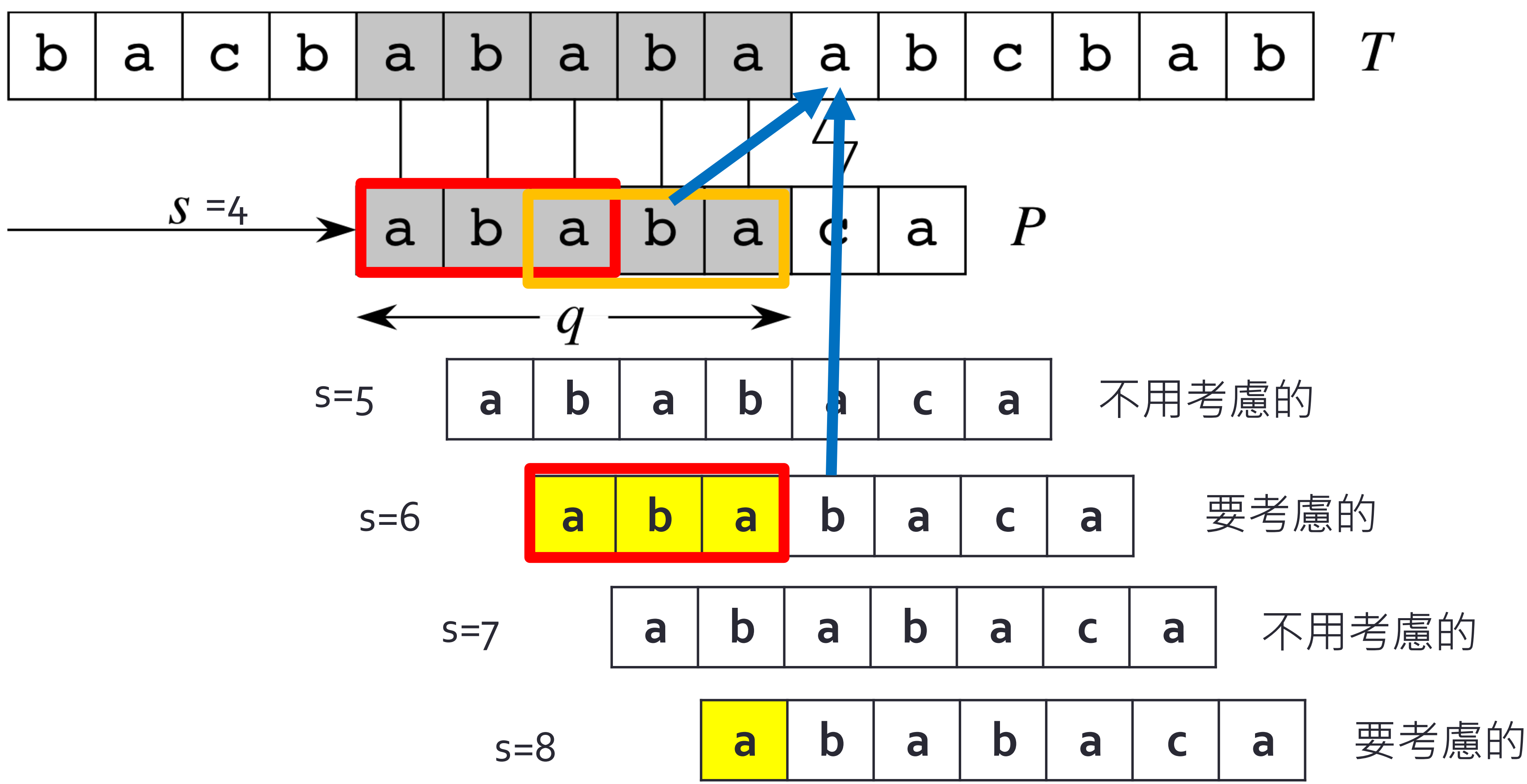
方法二：The Knuth-Morris-Pratt Algo.



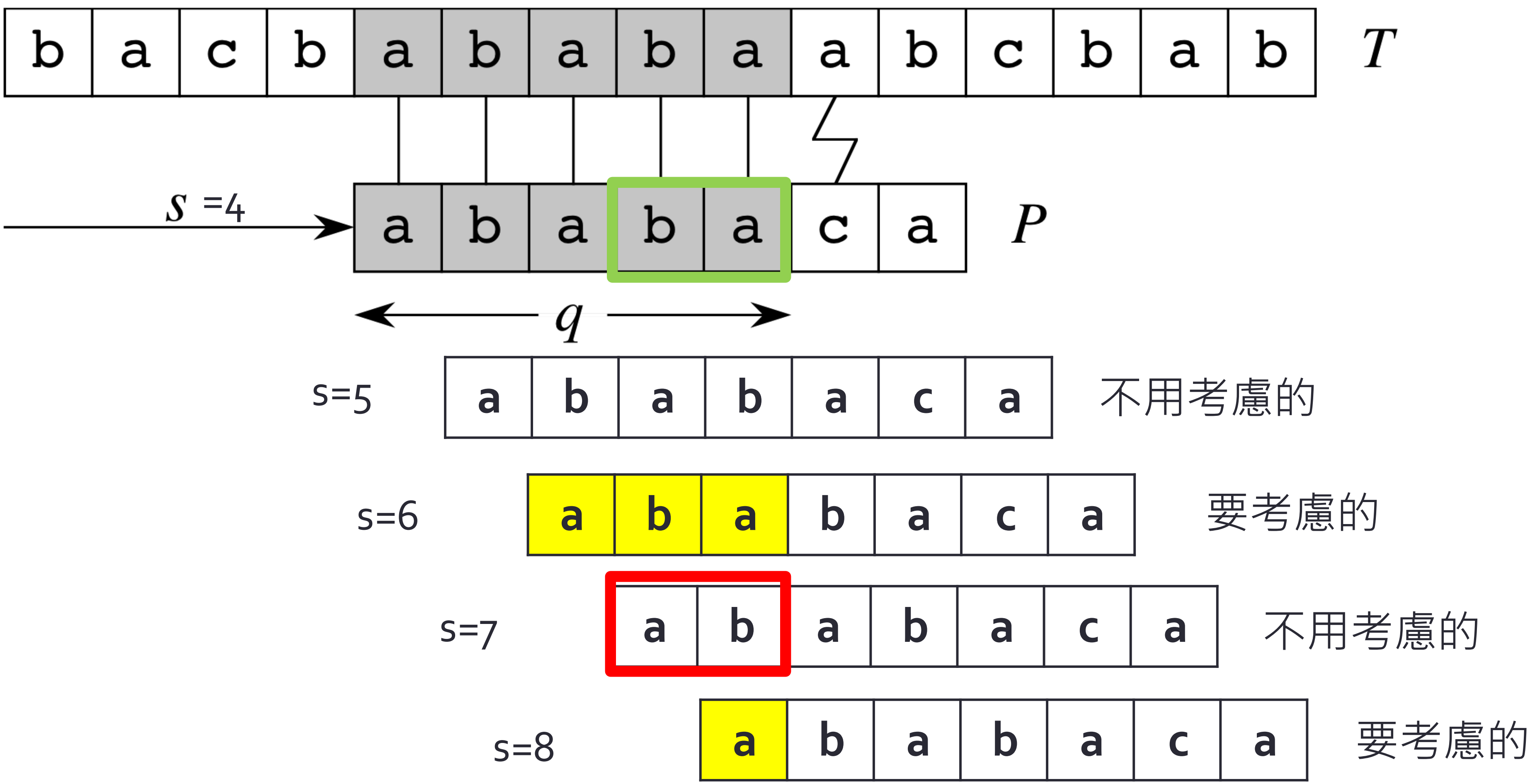
方法二：The Knuth-Morris-Pratt Algo.



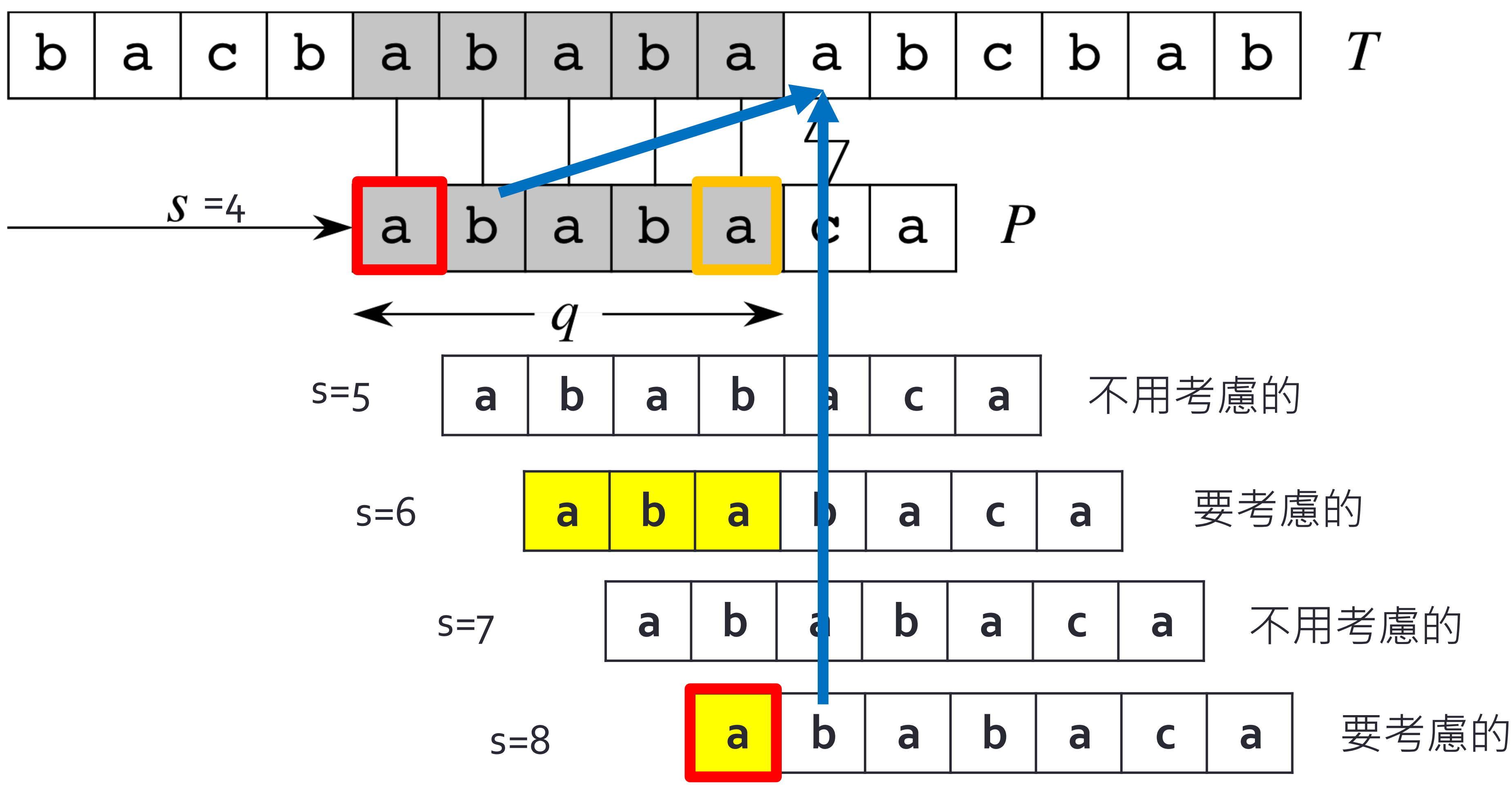
方法二：The Knuth-Morris-Pratt Algo.



方法二: The Knuth-Morris-Pratt Algo.



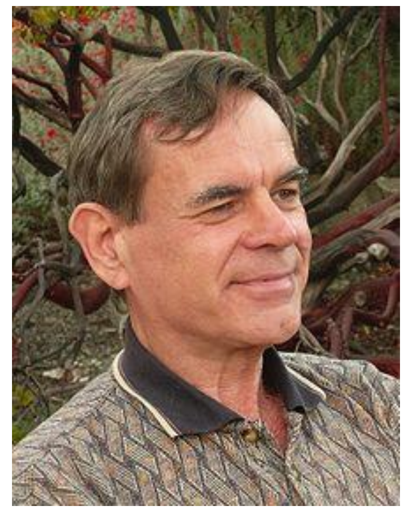
方法二：The Knuth-Morris-Pratt Algo.



Knuth-Morris-Pratt

Don Knuth

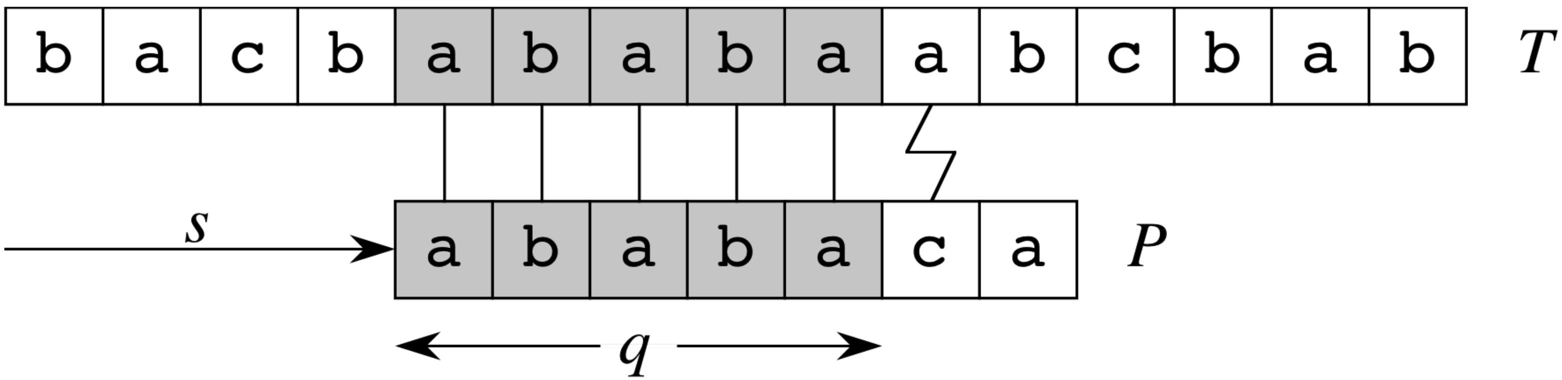
James Morris



Vaughan Pratt

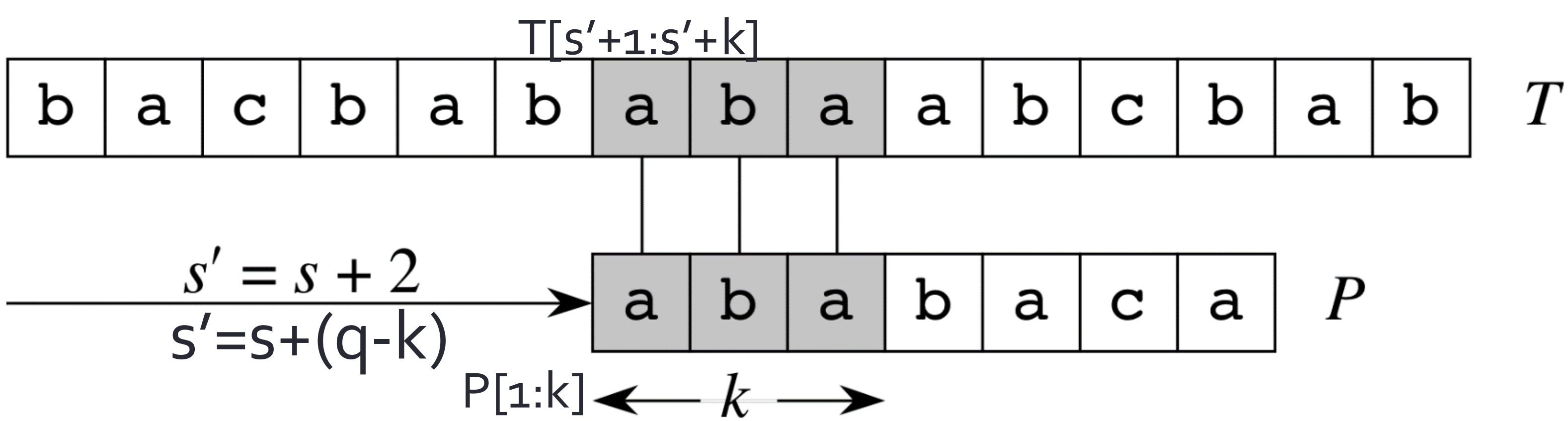
正式一點的說法

- 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

$$P[1:k] = T[s' + 1: s' + k], \text{且} s' + k = s + q$$

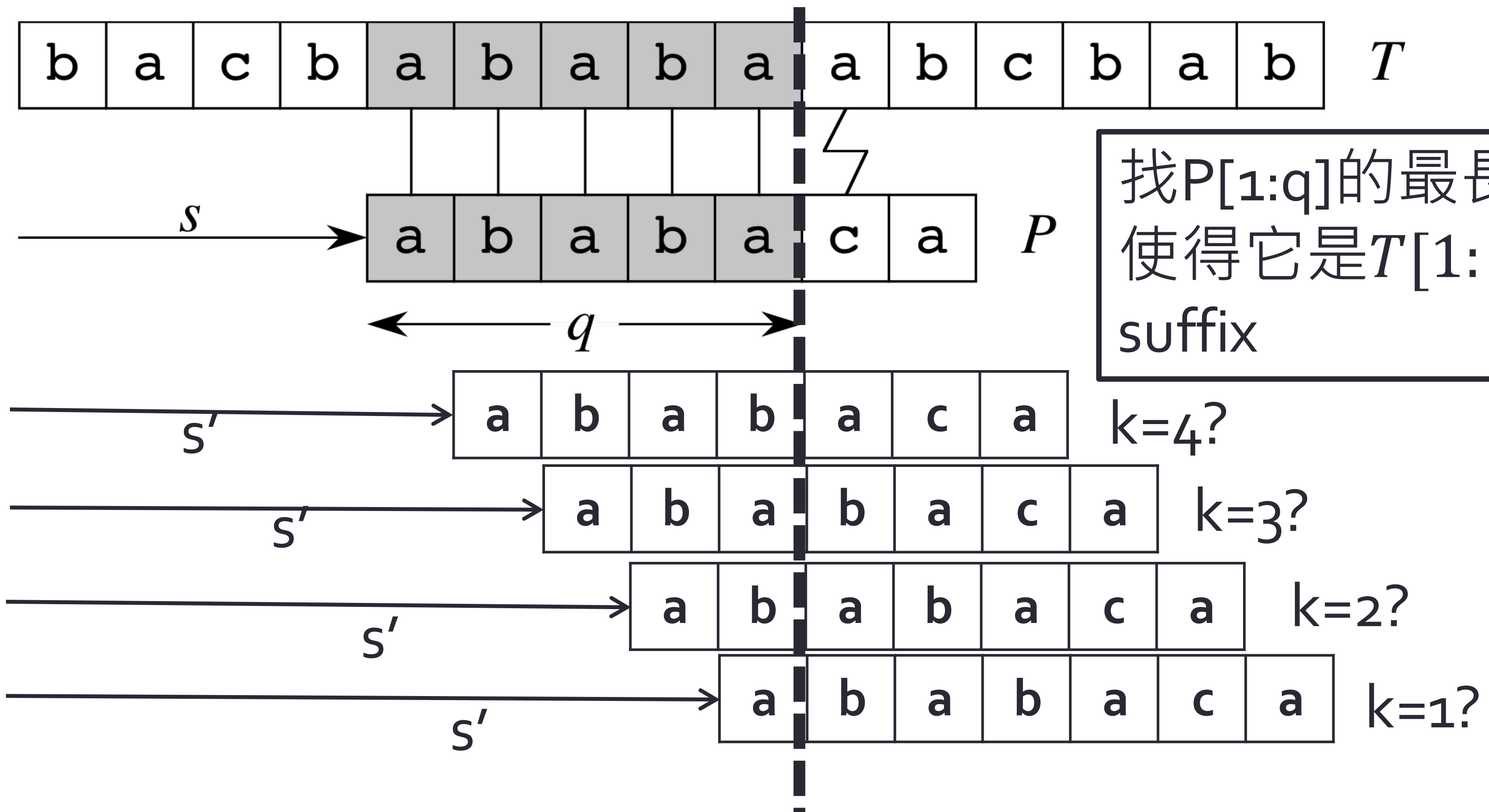


正式一點的說法

$\boxed{P[1:q]\text{是}T[1:s+q]\text{ 的suffix}}$

- 假設P[1:q]和T[s+1:s+q]已經match了

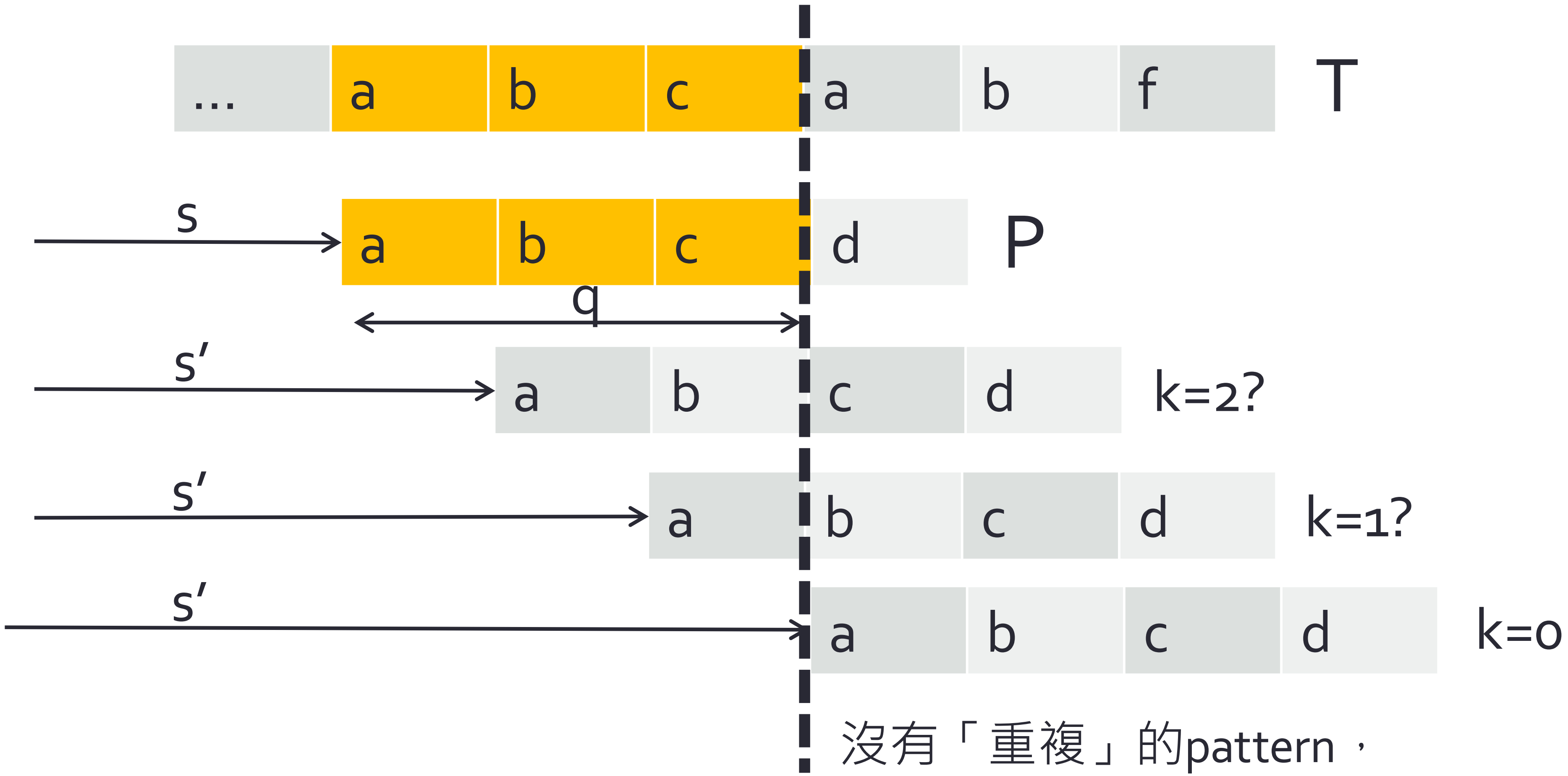
找P[1:q]的最長prefix 使得它是$T[1:s+q]$的suffix



- 要找出最小的shift s′使得某個k<q可以滿足

$$P[1:k] = T[s'+1:s'+k], \text{且} s'+k = s+q$$

沒有重複的pattern

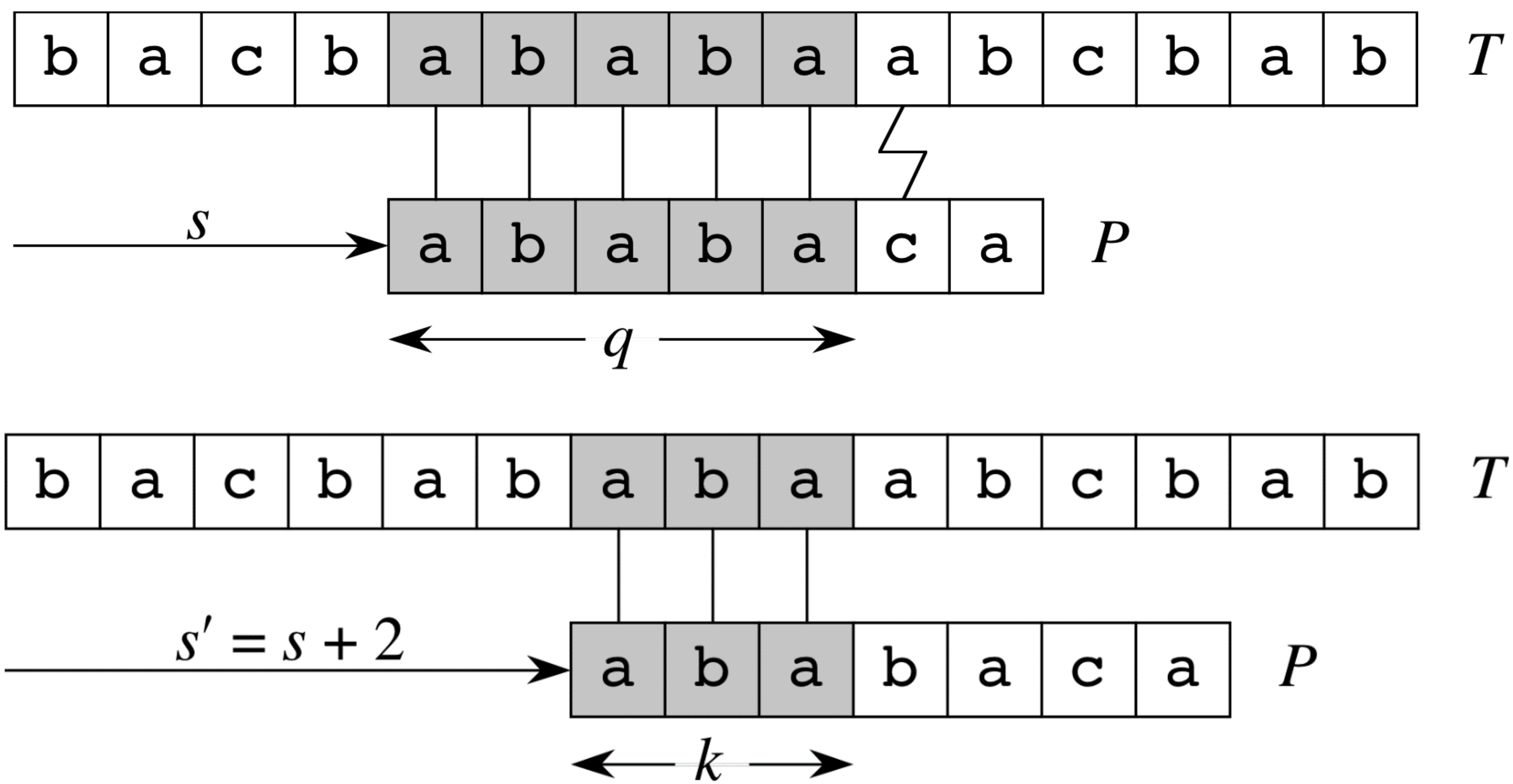


沒有「重複」的pattern，
可從頭、繼續比對

先處理P來取得”重複pattern”的資訊

找P[1:q]的最長prefix
使得它是T[1:s+q]的suffix

找P[1:q]的最長prefix
使得它是P[1:q]的suffix



先處理P來取得”重複pattern”的資訊

- 定義Prefix function (failure function) π:
- Input: {1,2,…,m}
- Output: {0,1,…,m-1}
- $\pi[q] = \max\{k: k < q \text{ and } P[1:k] \text{ is a suffix of } P[1:q]\}$

- (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

i	1	2	3	4	5	6	7	8	9	10	11	12
P[i]	A	B	A	B	A	C	A	B	A	B	A	B
$\pi[i]$												

Pseudo-code: Prefix function

```
Compute-Prefix-Function(P)
m=P.length
let π[1:m] be a new array
π[1] = 0
k=0
for q=2 to m
    while k>0 and P[k+1]!=P[q]
        k=π[k]
    if P[k+1]==P[q]
        k=k+1
    π[q] = k
return π
```

Pseudo-code: Matcher

```
KMP-Matcher(T,P,n,m)
π = Compute-Prefix-Function
q=0
k=0
for i=1 to n
     while q>0 and P[q+1]!=T[i]
          q=π[q]
     if P[q+1]==T[i]
          q=q+1
     if q==m
          print "Pattern occurs with shift" i-m
          q=π[q]
```

例子：Matching

- Ex. 1: T=BACBABABAABCBAB
- Ex. 2: T=BABABABACA

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

- 請見Cormen p. 978 KMP-Matcher

算Prefix function花多少時間?

```
Compute-Prefix-Function(P)
m=P.length
let π[1:m] be a new array
π[1] = 0
k=0
for q=2 to m
      while k>0 and P[k+1]!=P[q]
            k=π[k]
      if P[k+1]==P[q]
            k=k+1
      π[q] = k
return π
```

共$O(m)$次

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 π[1..m] be a new array
π[1] = 0
k=0
for q=2 to m
      while k>0 and P[k+1]!=P[q]
            k=π[k]
      if P[k+1]==P[q]
            k=k+1
      π[q] = k
return π
```

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)

Matcher要花多少時間？

```
KMP-Matcher(T,P,n,m)
π = Compute-Prefix-Function
q=0
k=0
for i=1 to n
    while q>0 and P[q+1]!=T[i]
        q=π[q]
    if P[q+1]==T[i]
        q=q+1
    if q==m
        print "Pattern occurs with shift" i-m
        q=π[q]
```

KMP執行時間

- 所以總和來看:
- Preprocessing時間 O(m)
- 比對時間 O(n)

方法三: 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

checksum

資料

檢查碼

- 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?
- $p = P[m] + 10\,P[m-1] + 10^2 P[m-2] + \cdots + 10^{m-2} P[2] + 10^m P[1]$
- O(m)的算法
- $p = P[m] + 10\big(P[m-1] + 10(P[m-2] + \cdots + 10(P[2] + 10P[1])\dots)\big)$

$$P[1] \cdot 10^{m-1} \quad P[2] \cdot 10^{m-2} \quad P[3] \cdot 10^{m-3} \quad \cdots \quad P[m-1] \cdot 10^{1} \quad P[m] \cdot 10^{0}$$

Q1：10^k 要算多久？ $(0 \le k \le m-1)$ $\boxed{O(\ \ )}$

Q2：整個 P 要算多久？ $\boxed{O(\ \ )}$

123

另一種算法 =

$$[10\,P[1] + P[2]] \times 10$$

$$[([\quad\quad] + P[3]] \times 10$$

$$[\quad\quad] \times 10$$

$$\cdots$$

$$\Rightarrow P = 10 \ldots 10\,(10\,(10\,P[1] + P[2]) + P[3]) + P[4]) \cdots + P[m])$$

每一項都用 一個 乘法

一個 加法.

m 項 $\Rightarrow$ running time

方法三: The Rabin-Karp Algorithm

- 怎麼從~~p~~ t_s 計算~~p~~ t_{s+1} 呢?
- $t_0 = T[m] + 10\big(T[m-1] + 10(T[m-2] + \cdots + 10(T[2] + 10T[1]) \ldots)\big)$

$s = 0$

T[1]	T[2]	T[3]	…	T[m-1]	T[m]	T[m+1]	T[m+2]	…

$s = 1$

整個往右移一格

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

拿掉最左邊那一格

加上最右邊那一格

$$t_s = 10^{m-1}T[s+1] + 10^{m-2}T[s+2] + \cdots + 10T[s+m-1] + T[s+m]$$

$$t_{s+1} = 10^{m-1}T[s+2] + \cdots + 10^2T[s+m-1] + 10T[s+m] + T[s+m+1]$$

for

		string	checksum
t_s	$T[s+1]\ T[s+2] \cdots T[s+m]\ T[s+m+1]$ $10^{m-1}\ 10^{m-2} \cdots\cdots 10^0$	$T[s+1:s+m]$	t_s
t_{s+1}	$\Rightarrow$ $10^{m-1} \cdots 10^1\ 10^0$	$T[s+2:s+m+1]$	t_{s+1}

pre-calculated in ? time

10^{m-1} $\boxed{O(\ )}$

$$t_s = 10^{m-1}\cdot T[s+1] + \boxed{}$$

$$t_{s+1} = \boxed{}\cdot 10 + T[s+m+1]$$

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

$\boxed{O(\ )}$

方法三: The Rabin-Karp Algorithm

- 那麼用這個方法要多花少時間呢? (簡易分析版)
- $p = P[m] + 10\big(P[m-1] + 10(P[m-2] + \cdots + 10(P[2] + 10P[1])\ldots)\big)$ $\boxed{O(m)}$
- $t_0 = T[m] + 10\big(T[m-1] + 10(T[m-2] + \cdots + 10(T[2] + 10T[1])\ldots)\big)$ $\boxed{O(m)}$
- 然後用$t_{s+1} = 10(t_s - 10^{m-1}T[s+1]) + T[s+m+1]$, 算$t_1, t_2, \ldots, t_{n-m}$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

方法三: The Rabin-Karp Algorithm

- 之前的兩個問題:

1. Σ如果是general的character set, 怎麼辦? (不再是{0,1,…,9})

如何解決呢?

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

$$p = idx(P[m]) + d\big(idx(P[m-1]) + d(idx(P[m-2]) + \cdots + d(idx(P[2]) + d \cdot idx(P[1])) \dots)\big)$$

(a) 把整個string看成一個d進位的數.
(b)字元在Σ 中index當作該字元所代表的的值

mini - HW - H

$\Sigma = \{A, B, C, \cdots, Z, a, b, \cdots, z\}$ $|\Sigma| = 52$

index: $0, 1, 2, \cdots, 25, 26, 27, \cdots, 51$

$P =$ ABCxyz $m = 6$

C	idx(C)
A	0
B	1
C	2
x	49
y	50
z	51

$p =$

$= 7727979$

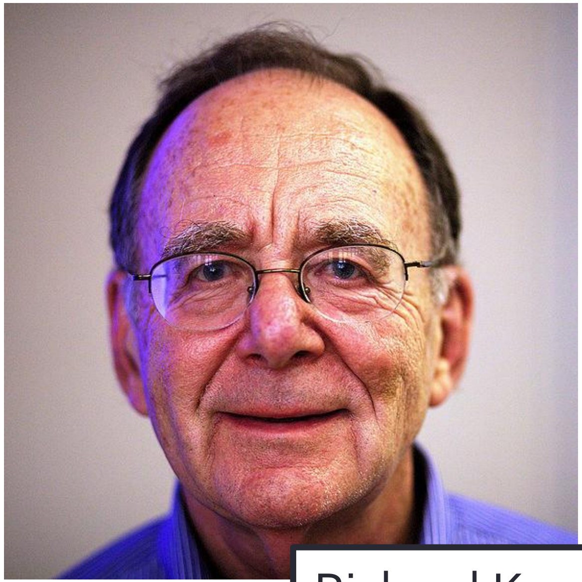
方法三: The Rabin-Karp Algorithm

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
- $a \equiv b \ (mod\ n)$: 表示a和b除以n的餘數相等 $0 \leq c < 12$
- 例如:
  - $38 \equiv 14 \ (mod\ 12)$
  - $0 \equiv 12 \ (mod\ 12)$
  - $-13 \equiv -3 \ (mod\ 5)$
- 更棒的性質:
- $a_1 \equiv b_1 \ (mod\ n)$
- $a_2 \equiv b_2 \ (mod\ n)$
- 則

1. $a_1 + a_2 \equiv b_1 + b_2 (mod\ n)$
2. $a_1 - a_2 \equiv b_1 - b_2 (mod\ n)$
3. $a_1 a_2 \equiv b_1 b_2 (mod\ n)$

① what if in $a \pmod{n}$ $a<0$?

the least positive number that should be subtracted from the dividend to make it divisible.

$-13 \bmod 5 \rightarrow -13-2=-15$ divisible by 5

$-3 \bmod 5 \rightarrow -3-2=-5$ divisible by 5

②

$a_1 \equiv b_1 \pmod{n}$

$a_2 \equiv b_2 \pmod{n}$

$a_1+a_2 \equiv b_1+b_2 \pmod{n}$

沒mod過　mod過

$\overset{a_1}{17} \bmod 5 = \overset{b_1}{2}$

$\overset{a_2}{8} \bmod 5 = \overset{b_2}{3}$

$(17+8) \bmod 5 = 0$

$(2+3) \bmod 5 = 0$

實際計算時，先mod比較好算。

同餘理論 (Modular Arithmetic)

- $a_1 \equiv b_1 \ (mod\ n)$
- $a_2 \equiv b_2 \ (mod\ n)$
- 則
- $a_1 + a_2 \equiv b_1 + b_2 (mod\ n)$

- 證明:
- $a_1 \equiv b_1 \ (mod\ n)$ 表示
  - $a_1 = c_{a_1} n + r_1$
  - $b_1 = c_{b_1} n + r_1$
- $a_2 \equiv b_2 \ (mod\ n)$ 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!
- $a_1 + a_2 \equiv b_1 + b_2 (mod\ n)$得證.

同餘理論 (Modular Arithmetic)

- $a_1 \equiv b_1 \ (mod\ n)$
- $a_2 \equiv b_2 \ (mod\ n)$
- 則
- $a_1 a_2 \equiv b_1 b_2 (mod\ n)$

- 證明:
- $a_1 \equiv b_1 \ (mod\ n)$ 表示
  - $a_1 = c_{a_1} n + r_1$
  - $b_1 = c_{b_1} n + r_1$
- $a_2 \equiv b_2 \ (mod\ n)$ 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mod版本的
- $p = P[m] + d\big(P[m-1] + d(P[m-2] + \cdots + d(P[2] + dP[1]) \ldots)\big)\ (mod\ q)$
- $t_0 = T[m] + d\big(T[m-1] + d(T[m-2] + \cdots + d(T[2] + dT[1]) \ldots)\big)\ (mod\ q)$
- $t_{s+1} = d(t_s - d^{m-1}T[s+1]) + T[s+m+1] (mod\ q)$

The Rabin-Karp Algorithm 修正版

- 新的mod版algorithm會造成一個問題:
- 雖然$t_s = p \Rightarrow t_s \equiv p \ (mod\ q)$
- 但$t_s = p \nLeftarrow t_s \equiv p \ (mod\ q)$
- 例如$38 \equiv 14 \ (mod\ 12), 38 \neq 14$
- 但是如果$t_s \not\equiv p \ (mod\ q) \Rightarrow t_s \neq p$

data1 $\rightarrow$ same checksum

data2 $\rightarrow$

- 所以演算法變成這樣:

1. 如果$t_s \not\equiv p \ (mod\ q)$, 那麼現在這個s為invalid shift
2. 如果$t_s \equiv p \ (mod\ q)$, 那麼必須額外檢查
(直接比對範圍內的字串→花很多時間)

- 當$t_s \equiv p \ (mod\ q)$, 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

為什麼q最好和d互質? $d = |\Sigma|$

希望checksum越分散越好

q最好是質數，因為d不確定

比如說 $d = 10$ $q = 500$

$$T[s+1] \cdot 10^{m-1} + \quad \cdots + T[s+m-4] \times 10^4 +$$

$$T[s+m-3] \times 10^3 + T[s+m-2] \times 10^2 + T[s+m-1] \times 10$$

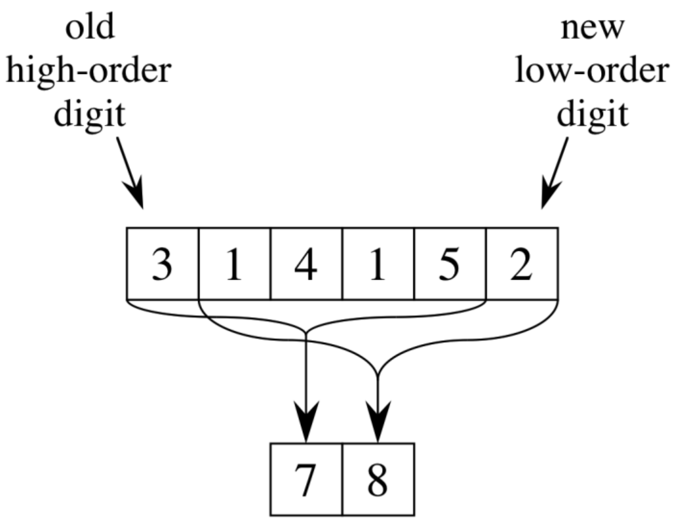
$$+ T[s+m]$$

$(\mathrm{mod}\ q)$

不會被checksum考慮.

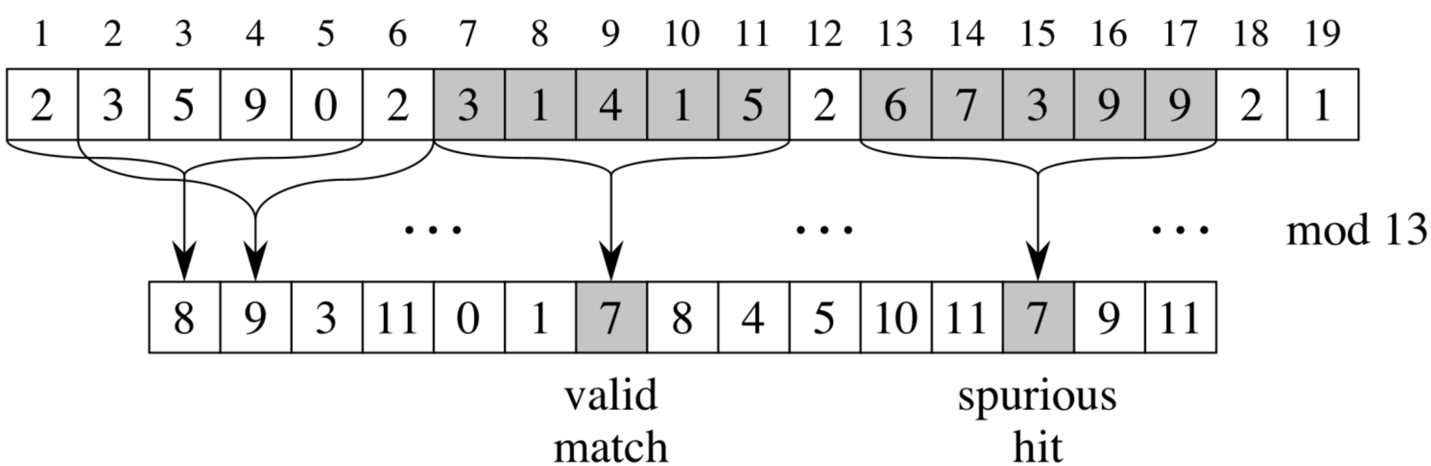
$(\mathrm{mod}\ q)$

例子：Rabin-Karp Algorithm



old high-order digit, shift, new low-order digit

$$\begin{aligned}14152 &\equiv (31415 - 3\cdot 10000)\cdot 10 + 2 \pmod{13}\\ &\equiv (7 - 3\cdot 3)\cdot 10 + 2 \pmod{13}\\ &\equiv 8 \pmod{13}\end{aligned}$$



Pseudo Code: Rabin-Karp

```
Rabin-Karp-Matcher(T,P,d,q)
n=T.length
m=P.length
h=d^(m-1) mod q
p=0
t=0
for i=1 to m
        p=(dp+P[i]) mod q
        t=(dt+T[i]) mod q
for s=0 to n-m
        if p==t
                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
```

T: string to be searched
P: pattern to be matched
d: size of the character set
q: max number

Pre-processing: $O(m)$

迴圈跑n-m+1次

Hit的時候比對: $O(m)$

Worst-case Running Time

```
Rabin-Karp-Matcher(T,P,d,q)
n=T.length
m=P.length
h=d^(m-1) mod q
p=0
t=0
for i=1 to m
      p=(dp+P[i]) mod q
      t=(dt+T[i]) mod q
for s=0 to n-m
      if p==t
            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
```

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))

$O(nm)$

Pre-processing: $O(m)$

迴圈跑n-m+1次

Hit的時候比對: $O(m)$

Average Running Time

```
Rabin-Karp-Matcher(T,P,d,q
n=T.length
m=P.length
h=d^(m-1) mod q
p=0
```

Pre-processing: $O(m)$

```
    for i=1 to m
          p=(dp+P[i]) mod q
           (dt+T[i]) mod q
    for s=0 to n-m
          if p==t
```

迴圈跑n-m+1次

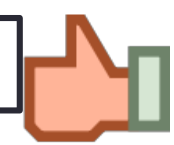
```
                if P[1..m]==T[s+1..s+m]
                      print "Pattern occurs with shift" s
          if s<n-m
                t=(d(t-T[s+1]h)+T[s+m+1]) mod q
```

Hit的時候比對: $O(m)$

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≥m, → O(n+m)=O(n)



Assumption:

c 個 valid shifts $\quad c = O(1)$

$\frac{1}{q}$ spurious hit 機率 $\quad q \geq m$

(不含 valid shift 機率)

valid shift:

額外花 $O(cm)$ 比對

spurious hit:

共有 $n-m+1$ 可能的 shift values

spurious hit 次數 $\frac{n-m+1}{q}$

額外花 $O(\frac{n-m+1}{q}m)$

比對 total:

$$O\left(m\left(c + \frac{n-m+1}{q}\right)\right)$$

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

Related Reading

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