เอกสารประกอบการอบรม

เรื่อง String Matching Algorithms

อาจารย์ ดร.พิชญะ สิทธีอมร

นำมาจาก Slide วิชา Algorithms

โดย รองศาสตราจารย์ ดร.สมชาย ประสิทธิ์จูตระกูล

การจับคู่สตริง

พิชญะ สิทธื่อมร
นำมาจาก slides ของ อ. สมชาย ประสิทธิ์จูตระกูล

หัวข้อ

นิยามปัญหา

Brute-force (naïve)

Rabin-Karp

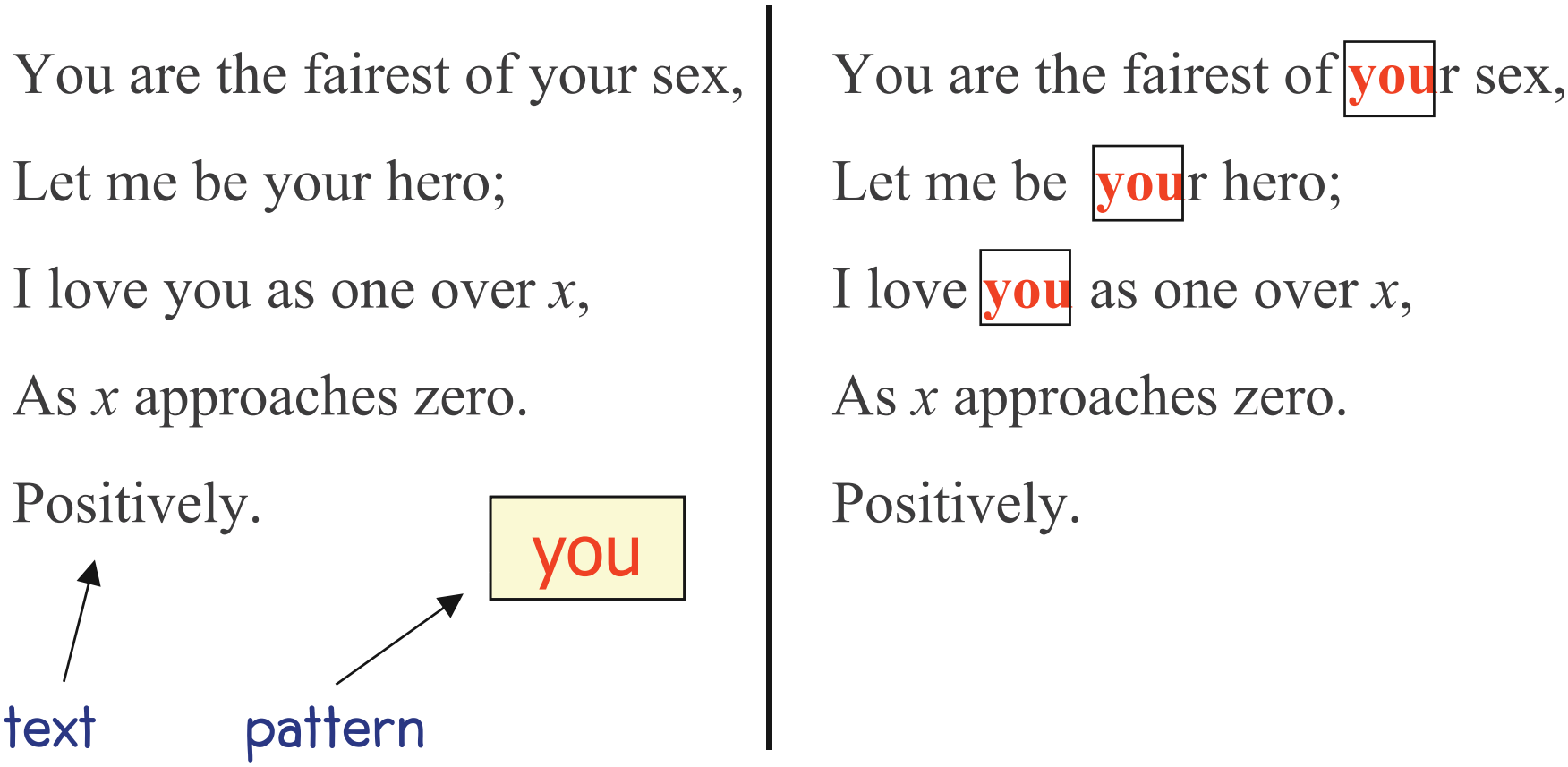
Finite-automaton

Knuth-Morris-Pratt

Boyer-Moore

Horspool

Exact String Matching



Approximate String Matching

You are the fairest of your sex,

Let me be your hero;

I love you as one over *x*.

As *x* approaches zero.

Positively.

heero

text

pattern

Input

You are the fairest of your sex,

Let me be your **hero**;

I love you as one over *x*,

As *x* approaches **zero**.

Positively.

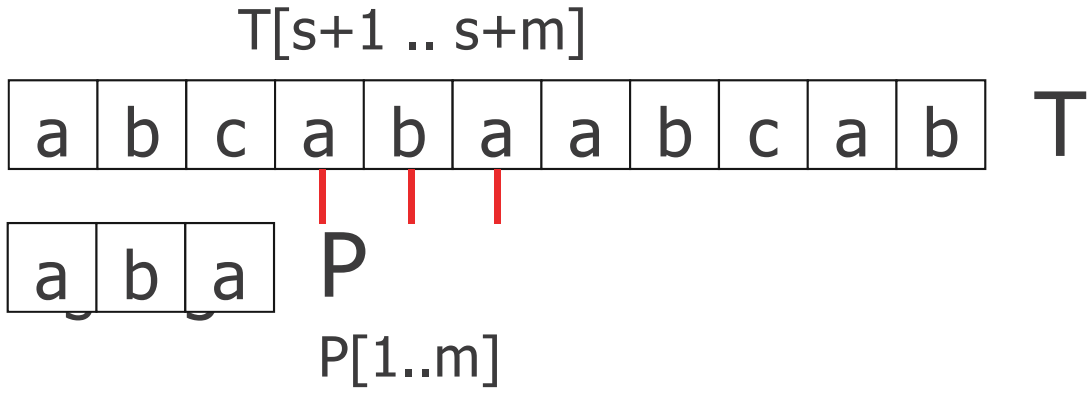
Output

ปัญหา String Matching

- **Finite alphabet : $\sum$**
  - $\sum$ = {a, b, c, …, z}
  - $\sum$ = {0, 1}
  - $\sum$ = {A,C,G,T}, …
- **Text : T[1..n]**
- **Pattern : P[1..m]**

ต้องการหา
all valid shifts
first valid shift
last valid shift

$0 \le s \le n - m$

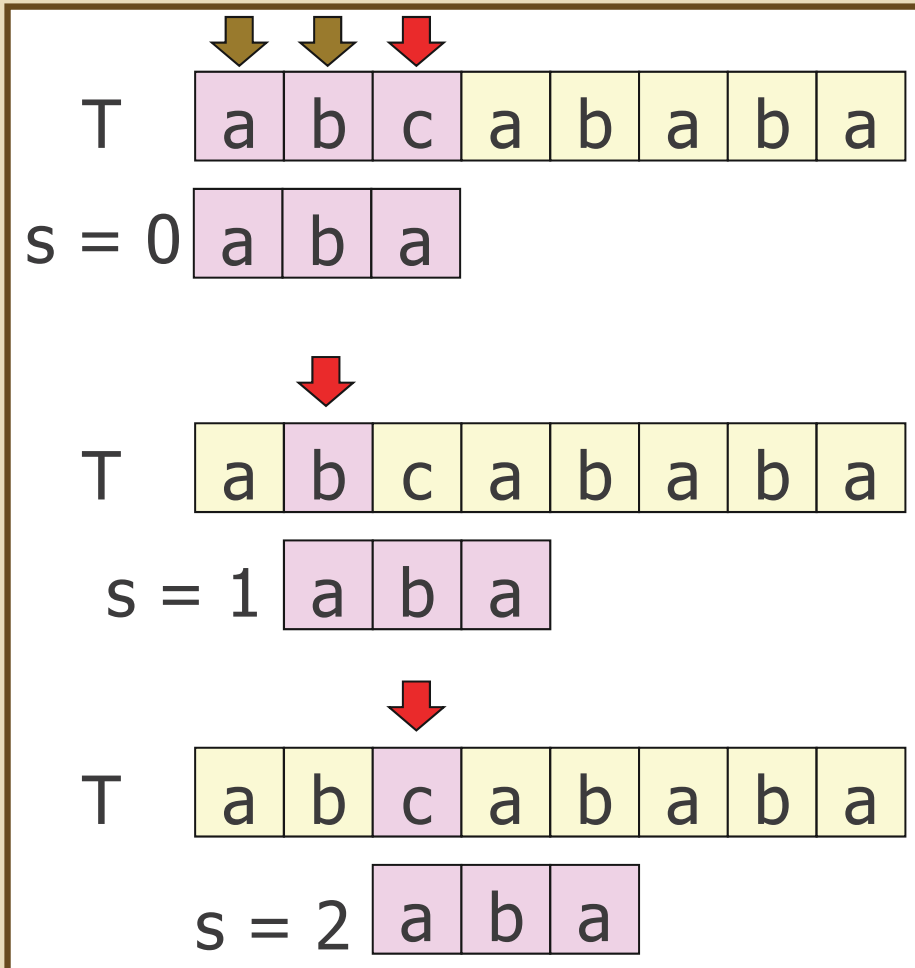


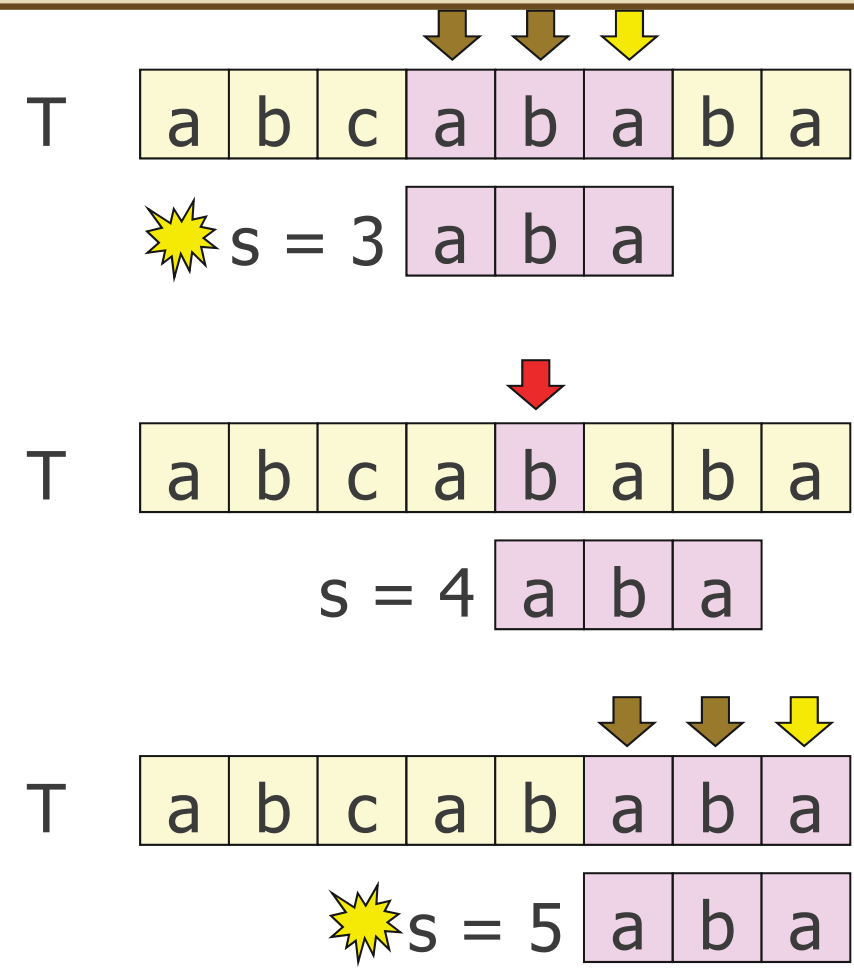
s เป็น valid shift เมื่อ P[1..m] = T[s+1 .. s+m]

อัลกอริทึมการเทียบสตริงแบบแม่นตรง

- ✓ Brute Force algorithm
- ✓ Deterministic Finite Automaton algorithm
- ✓ Rabin-Karp algorithm
- Shift Or algorithm
- Morris-Pratt algorithm
- ✓ Knuth-Morris-Pratt algorithm
- Simon algorithm
- Colussi algorithm
- Galil-Giancarlo algorithm
- Apostolico-Crochemore algorithm
- ✓ Boyer-Moore algorithm
- Turbo BM algorithm
- Apostolico-Giancarlo algorithm
- Reverse Colussi algorithm
- ✓ Horspool algorithm
- Quick Search algorithm
- Tuned Boyer-Moore algorithm
- Zhu-Takaoka algorithm
- Berry-Ravindran algorithm
- Smith algorithm
- Raita algorithm
- Reverse Factor algorithm
- Turbo Reverse Factor algorithm
- Forward Dawg Matching algorithm
- Backward Nondeterministic Dawg Matching algorithm
- Backward Oracle Matching algorithm
- Galil-Seiferas algorithm
- Two Way algorithm
- String Matching on Ordered Alphabets algorithm
- Optimal Mismatch algorithm
- Maximal Shift algorithm
- Skip Search algorithm
- KMP Skip Search algorithm
- Alpha Skip Search algorithm
- ...

Brute Force (Naïve) Algorithm





Brute Force (Naïve) Algorithm

```
Naive-String-Matching( T[1..n], P[1..m] ) {
  for( s = 0 to n-m ) {
    for( i = 1 to m ) {
      if (T[s+i] ≠ P[i]) break;
    }
    if (i > m) print( s )
  }
}
```

$\Theta(n-m)$

$O(m)$

$O(nm)$

worst case : T = AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAZ
P = AAAAZ

average # of comparisons : (random text and random pattern)

d = |Σ|

$$\frac{1-d^{-m}}{1-d^{-1}}(n-m+1) < 2(n-m+1)$$

Random T&P : Avg #Cmps

ตัวที่ 1 ไม่เหมือน $\left(\frac{d-1}{d}\right) \times 1$

ตัวแรกเหมือน
ตัวที่ 2 ไม่เหมือน $\left(\frac{1}{d}\right)^{1}\left(\frac{d-1}{d}\right) \times 2$

สองตัวแรกเหมือน
ตัวที่ 3 ไม่เหมือน $\left(\frac{1}{d}\right)^{2}\left(\frac{d-1}{d}\right) \times 3$

...

m – 1 ตัวแรกเหมือน
ตัวที่ m ไม่เหมือน $\left(\frac{1}{d}\right)^{m-1}\left(\frac{d-1}{d}\right) \times m$

$$\sum_{k=1}^{m} \frac{k}{d^{k-1}}\left(\frac{d-1}{d}\right) +$$

เหมือนกันทั้ง m ตัว $\left(\frac{1}{d}\right)^{m} \times m$

$$\frac{m}{d^{m}}$$

Random T&P : Avg #Cmps

$$\frac{m}{d^m} + \sum_{k=1}^{m} \frac{k}{d^{k-1}} \left(\frac{d-1}{d} \right) = \frac{m}{d^m} + \left(1 - \frac{1}{d} \right) \sum_{k=1}^{m} \frac{k}{d^{k-1}} = ?$$

$$\sum_{k=0}^{m} x^k = \frac{1 - x^{m+1}}{1 - x}$$

$$\sum_{k=1}^{m} k x^{k-1} = \frac{-(1-x)(m+1)x^m + (1 - x^{m+1})}{(1-x)^2}$$

$$= \frac{1 - (m+1)x^m + m x^{m+1}}{(1-x)^2}$$

$$= \frac{1 - x^m + m x^m (x-1)}{(1-x)^2}$$

Random T&P : Avg #Cmps

$$\frac{m}{d^m} + \left(1 - \frac{1}{d}\right)\sum_{k=1}^{m} \frac{k}{d^{k-1}} = \frac{1 - d^{-m}}{1 - d^{-1}}$$

$$\sum_{k=1}^{m} k x^{k-1} = \frac{1 - x^m + m x^m (x-1)}{(1-x)^2}$$

$$mx^m + (1-x)\sum_{k=1}^{m} k x^{k-1} = mx^m + (1-x)\left(\frac{1 - x^m + m x^m (x-1)}{(1-x)^2}\right)$$

$$= mx^m + \left(\frac{1 - x^m + m x^m (x-1)}{(1-x)}\right)$$

$$= \frac{1 - x^m}{1 - x} \longrightarrow \text{ให้ } x = d^{-1}$$

Random T&P : Avg #Cmps

```
Naive-String-Matching( T[1..n], P[1..m] ) {
  for( s = 0 to n-m ) {
    for( i = 1 to m ) {
      if (T[s+i] ≠ P[i]) break;
    }
    if (i > m) print( s )
  }
}
```

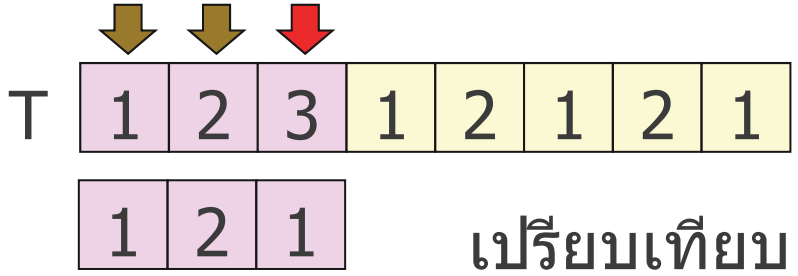
$$\frac{1-d^{-m}}{1-d^{-1}}$$

$$\frac{1-d^{-m}}{1-d^{-1}}(n-m+1) < 2(n-m+1)$$

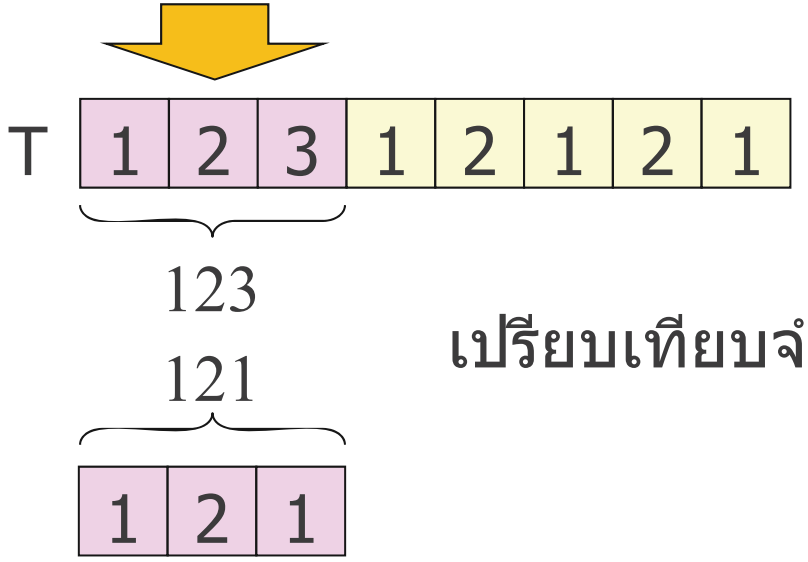
$$\frac{1-d^{-m}}{1-d^{-1}} < \frac{1}{1-d^{-1}}, \quad d \geq 2$$

$$\leq \frac{1}{1-2^{-1}} = 2$$

Rabin-Karp Matcher

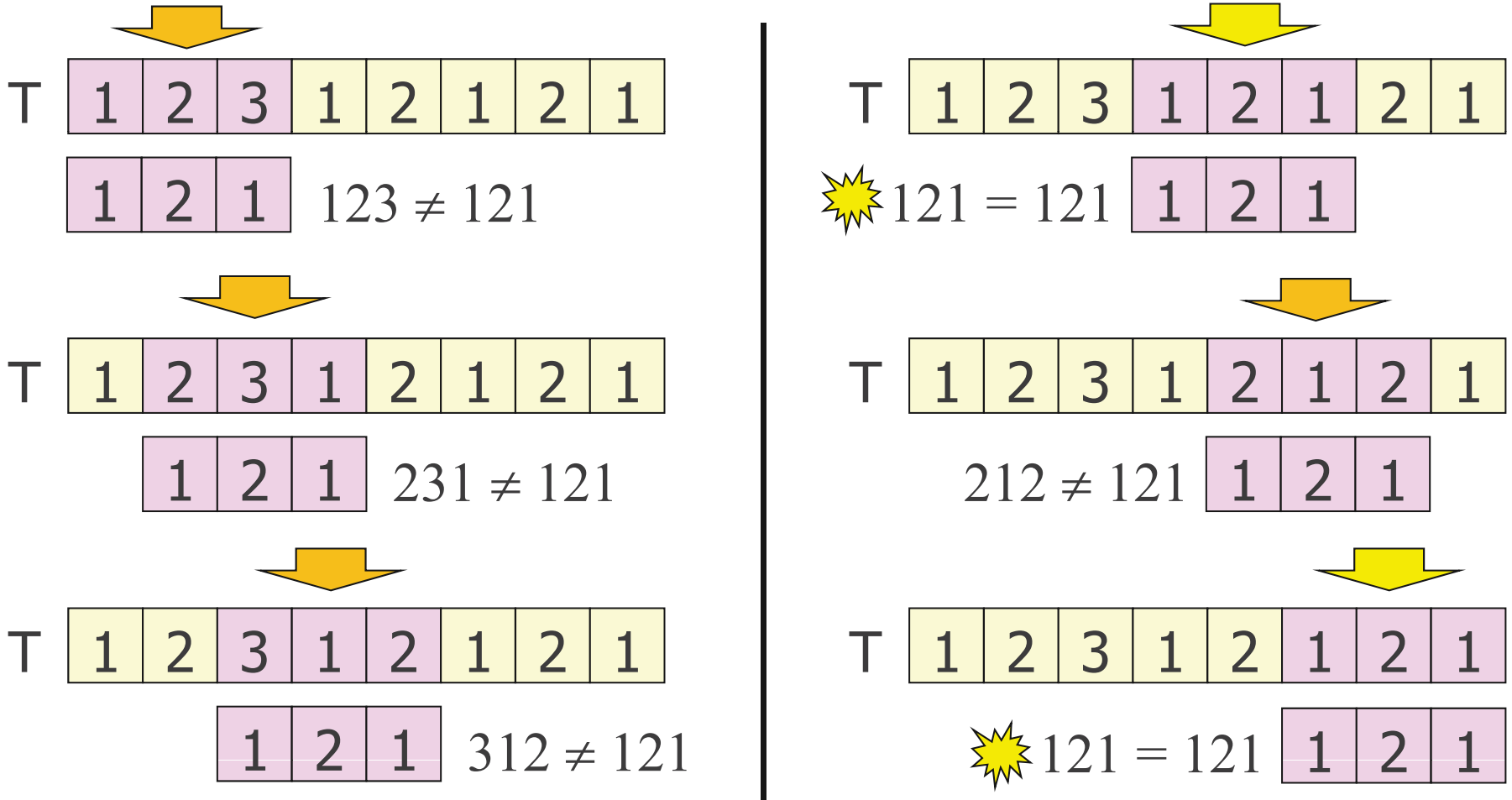


เปรียบเทียบ 1 กับ 1,
2 กับ 2,
3 กับ 1



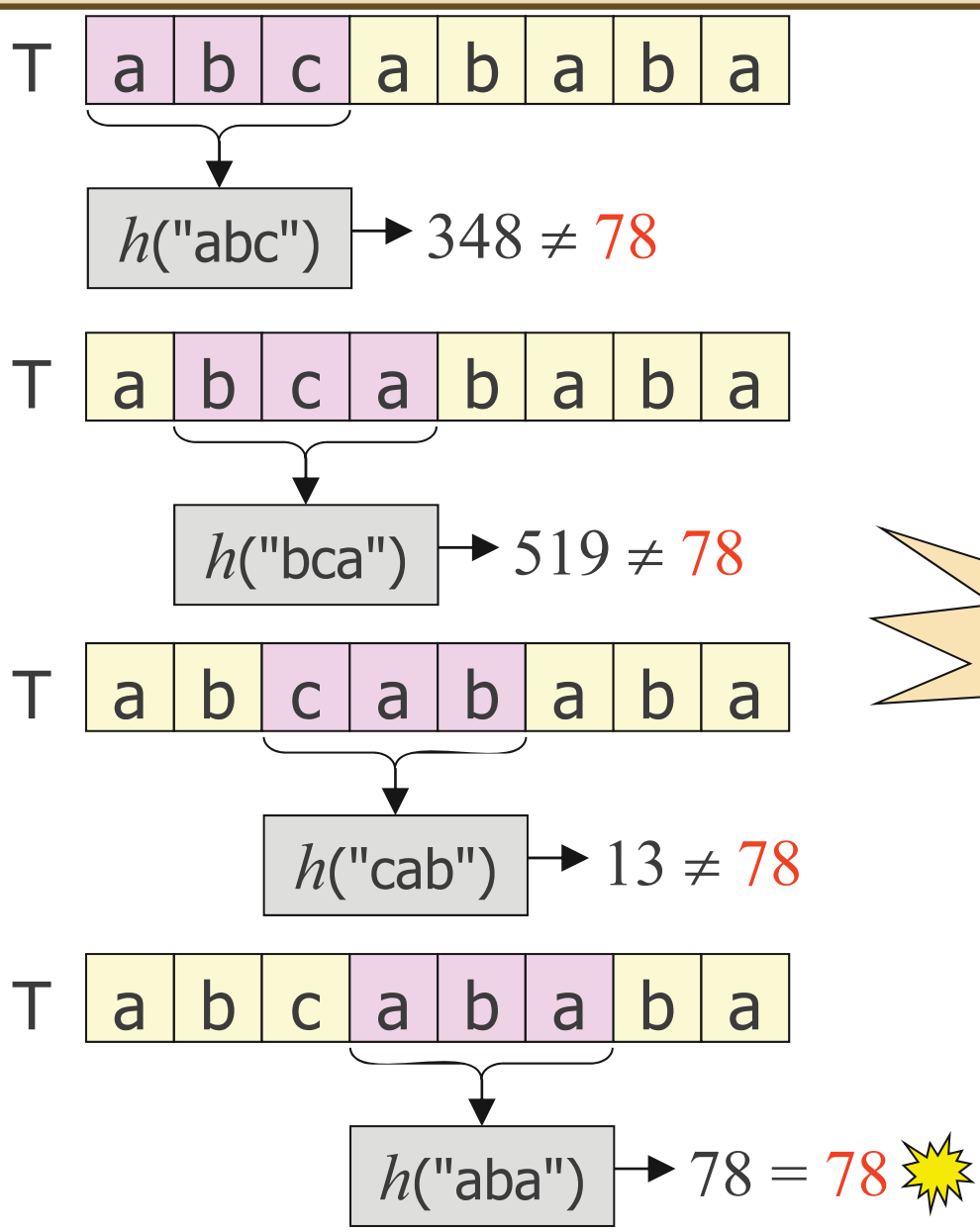
เปรียบเทียบจำนวน 123 กับ 121

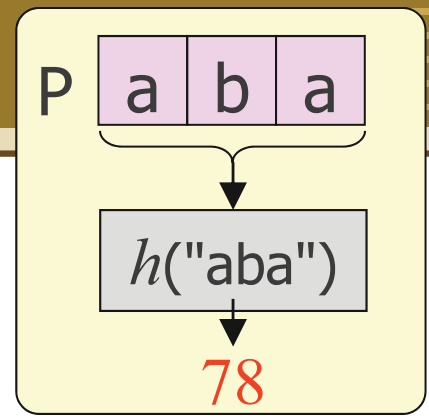
Rabin-Karp Algorithm

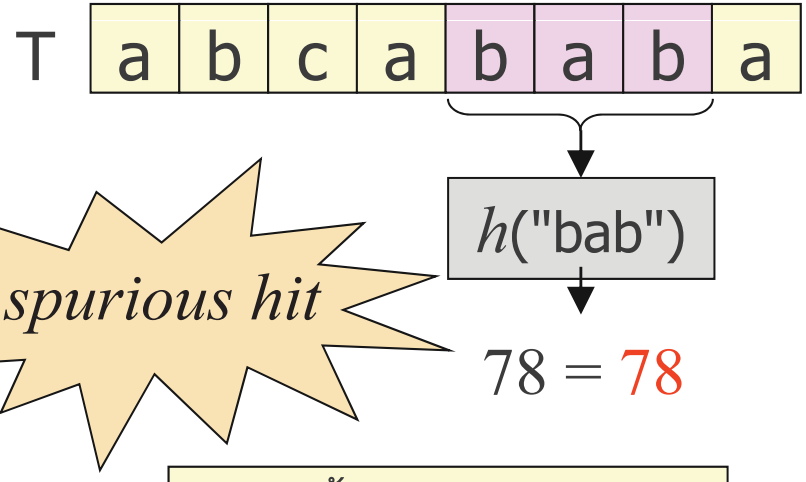


ถ้า pattern เป็นตัวอักษร หรือ
ถ้าเป็นตัวเลข แต่มีขนาดใหญ่เกิน int เกิน long } จะทำอย่างไร ?

Hashing







spurious hit

ดังนั้น ถ้า hit ต้อง
เทียบแต่ละ char ด้วย

อีกปัญหา : การคำนวณ $h(p)$
ที่ใช้เวลาตามขนาดของ p

การคำนวณค่า h : incremental

- h(s) = (มอง s เป็นจำนวน ในระบบเลขฐาน)
- h("abc") = $(1\times37^2 + 2\times37^1 + 3\times37^0)$ = 1446
- h("bce") = $(2\times37^2 + 3\times37^1 + 5\times37^0)$ = 2853

$$= 37\times(\cancel{1\times37^2} + 2\times37^1 + 3\times37^0) + 5\times37^0$$

$$= 37\times(\ h(\text{"abc"}) - 1\times37^2\ ) + 5\times37^0$$

เลื่อนตำแหน่ง "bc" ไปทางซ้าย

ลบ "a"

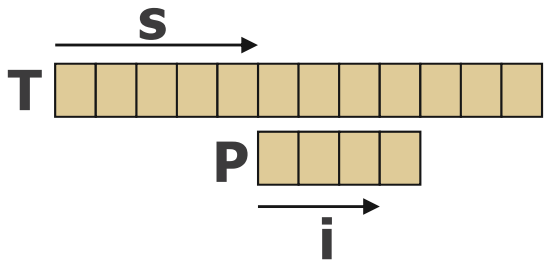
เพิ่ม "e"

$$h(\text{"...e"}) = 37\times(\ h(\text{"a..."}) - 1\times37^{m-1}\ ) + 5\times37^0$$

Rabin-Karp Algorithm

```
Rabin-Karp-Matching( T[1..n], P[1..m], d ) {
  ht = 0; hp = 0
  for ( i = 1 to m ) {
    ht = d*ht + T[i]
    hp = d*hp + P[i]
  }
  for ( s = 0 to n - m ) {
    if (ht == hp) {
      for ( i = 1 to m )
        if (T[s+i] ≠ P[i]) break
      if (i > m) print( s )
    }
    if (s < n-m) {
      ht = d*(ht - T[s+1]*d^(m-1)) + T[s+m+1]
    }
  }
}
```

d = |Σ|



ถ้า m มีค่ามาก ขนาดของ ht และ hp ก็ใหญ่

$h("...e") = 37 \times (h("a...") - 1 \times 37^{m-1}) + 5 \times 37^{0}$

Rabin-Karp Algorithm

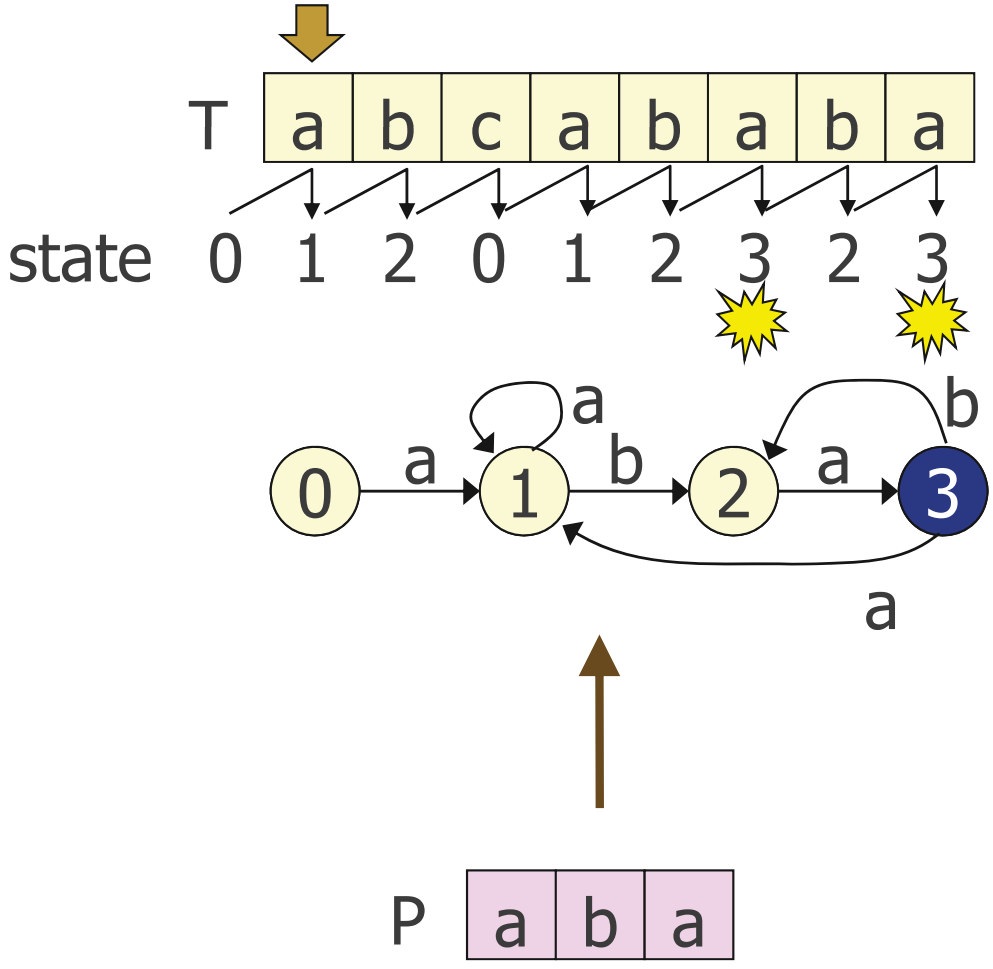
```
Rabin-Karp-Matching( T[1..n], P[1..m], d, q ) {
  ht = 0; hp = 0; dm1 = d^(m-1) % q
  for ( i = 1 to m ) {
    ht = (d*ht + T[i]) % q
    hp = (d*hp + P[i]) % q
  }
  for ( s = 0 to n - m ) {
    if (ht == hp) {
      for ( i = 1 to m )
        if (T[s+i] ≠ P[i]) break
      if (i > m) print( s )
    }
    if (s < n-m) {
      ht = (d*(ht - T[s+1]*dm1) + T[s+m+1]) % q
    }
  }
}
```

worst case $\Theta(nm)$

avg case $O(n + m)$

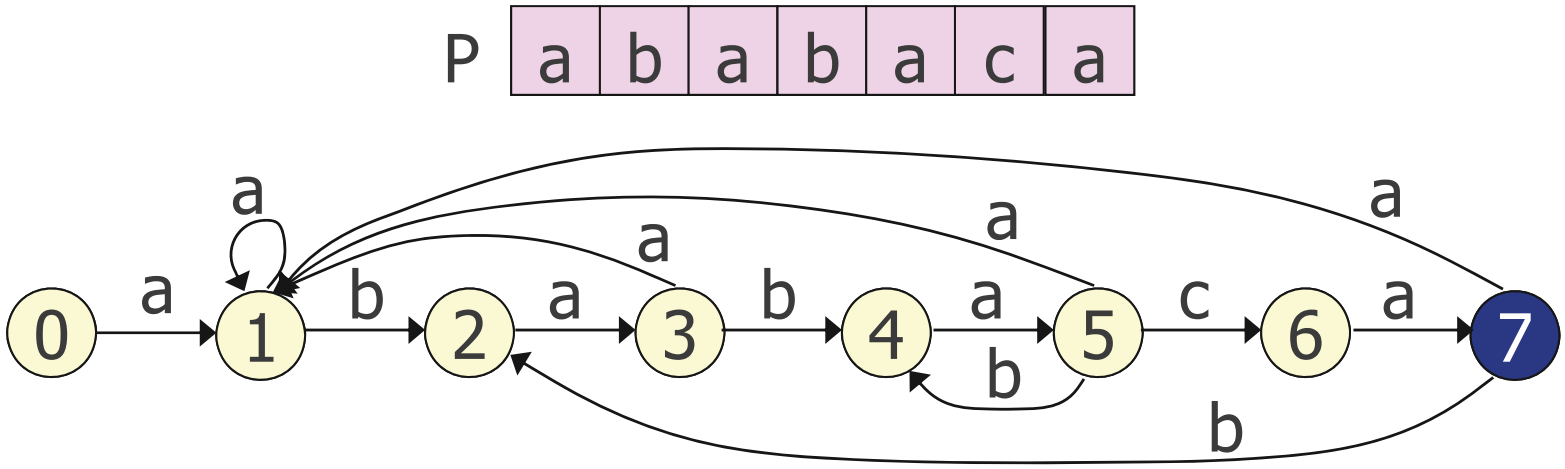
h("...e") = 37×(h("a...") – 1×37^{m-1}) + 5×37^{0}

Finite Automaton Matcher



สร้าง state transition diagram จาก pattern

Finite Automaton Matcher



T	a	b	a	b	a	b	a	c	b	a	b	a	b	a	c	a
q = 0	1	2	3	4	5	4	5	6	0	1	2	3	4	5	6	7

```
Finite-Automaton-Matching(T[1..n], δ, m ) {
  q = 0
  for( i = 1 to n ) {
    q = δ(q, T[i])
    if (q == m) print( i-m )
  }
}
```

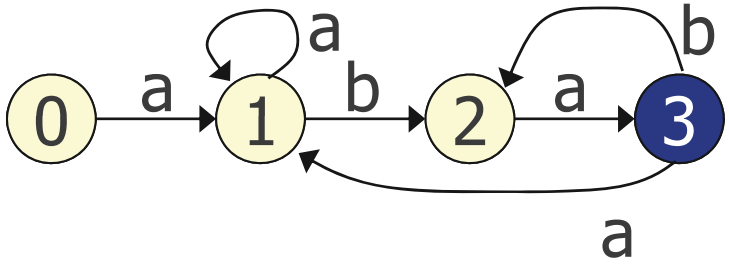
transition function

$\Theta(n)$

สร้าง transition function อย่างไร ?

Transition Function

P | a | b | a |



$\Sigma = \{a, b, c\}$

ตารางมีขนาด $(m+1)|\Sigma|$

input

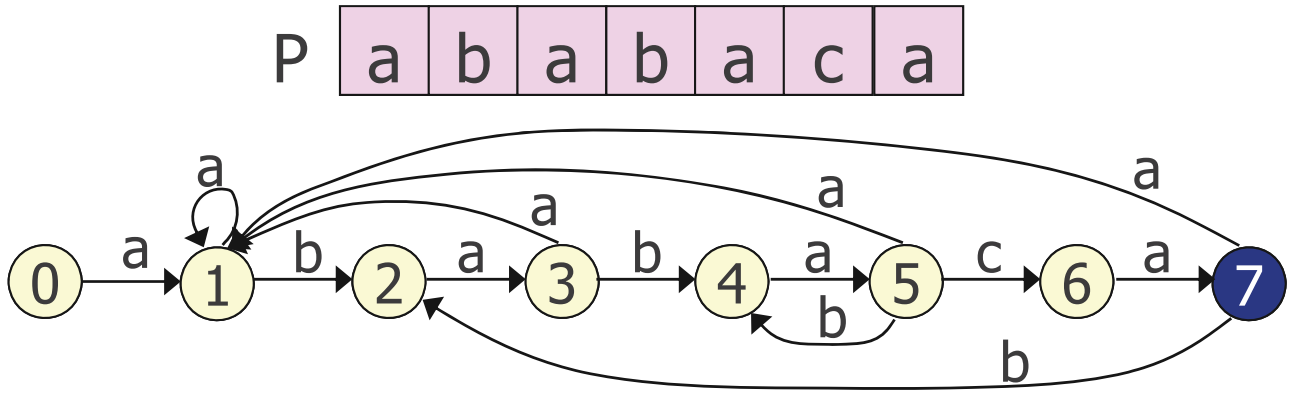
$\longrightarrow |\Sigma| \longrightarrow$

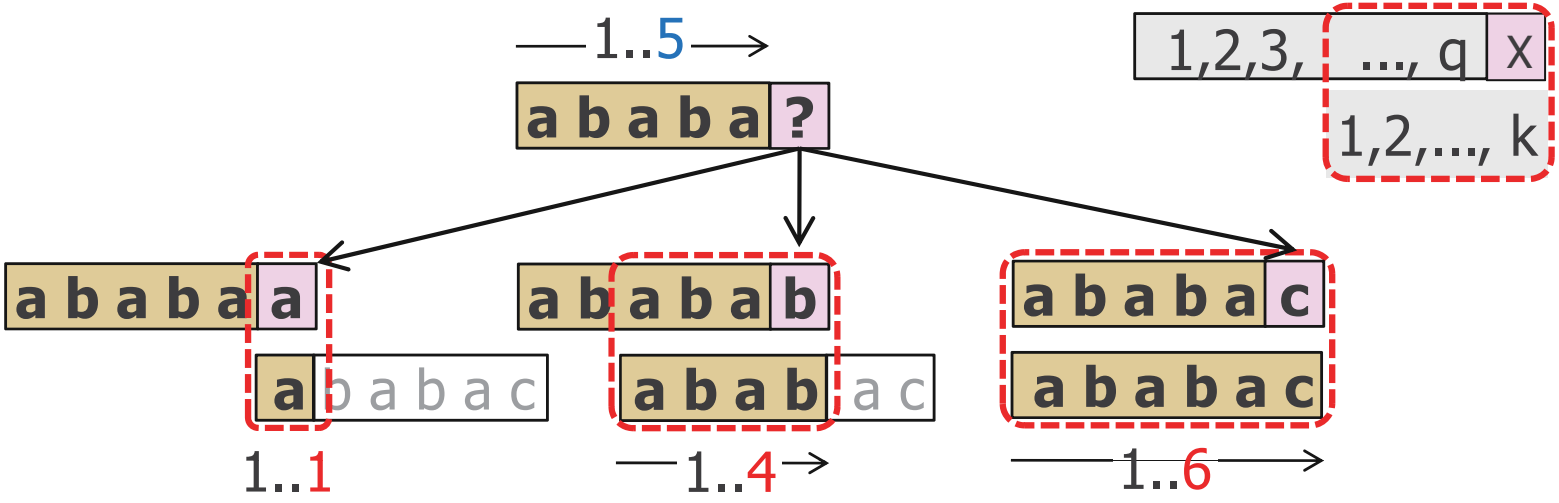
state	a	b	c
0	1	0	0
1	1	2	0
2	3	0	0
3	1	2	0

m+1

δ(state, input)

Transition Function





$$\delta(q, x) = \max\{ k : (P[1..q]+x).\text{endsWith}(P[1..k]) \}$$

ตัวอย่าง Transition Function

$\delta(q, x) = \max\{ k : (P[1..q]+x).\text{endsWith}(P[1..k]) \}$

state	a	b	c
0	1	0	0
1	1	2	0
2	3	0	0
3	1	4	0
4	5	0	0
5	1	4	6
6	7	0	0
7	1	2	0

P	a	b	a	b	a	c	a

$\delta(5,\text{"b"}) = 4$

P[1..5]+"b" = ababab

P[1..k]

~~a~~
ab
~~aba~~
abab
~~ababa~~
~~ababac~~
~~ababaca~~

P[1..4]

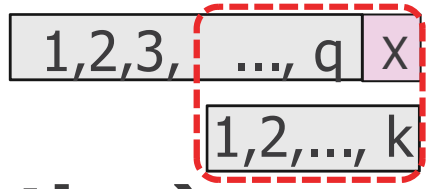
สร้างแบบตรงไปตรงมาใช้เวลา $O(m^3|\Sigma|)$

มีวิธีสร้างที่ใช้เวลา $O(m|\Sigma|)$

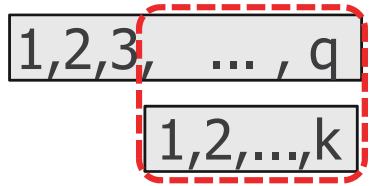
รวมการค้นด้วย ใช้เวลา $O(n + m|\Sigma|)$

Knuth-Morris-Pratt Matcher

- **ของเดิม (transition function)**
  - $\delta(q, x)$ = max{ k : (P[1..q]+x).endsWith(P[1..k]) }
  - ตารางมีขนาด $(m+1)|\Sigma|$
  - ใช้เวลาสร้าง $O(m|\Sigma|)$

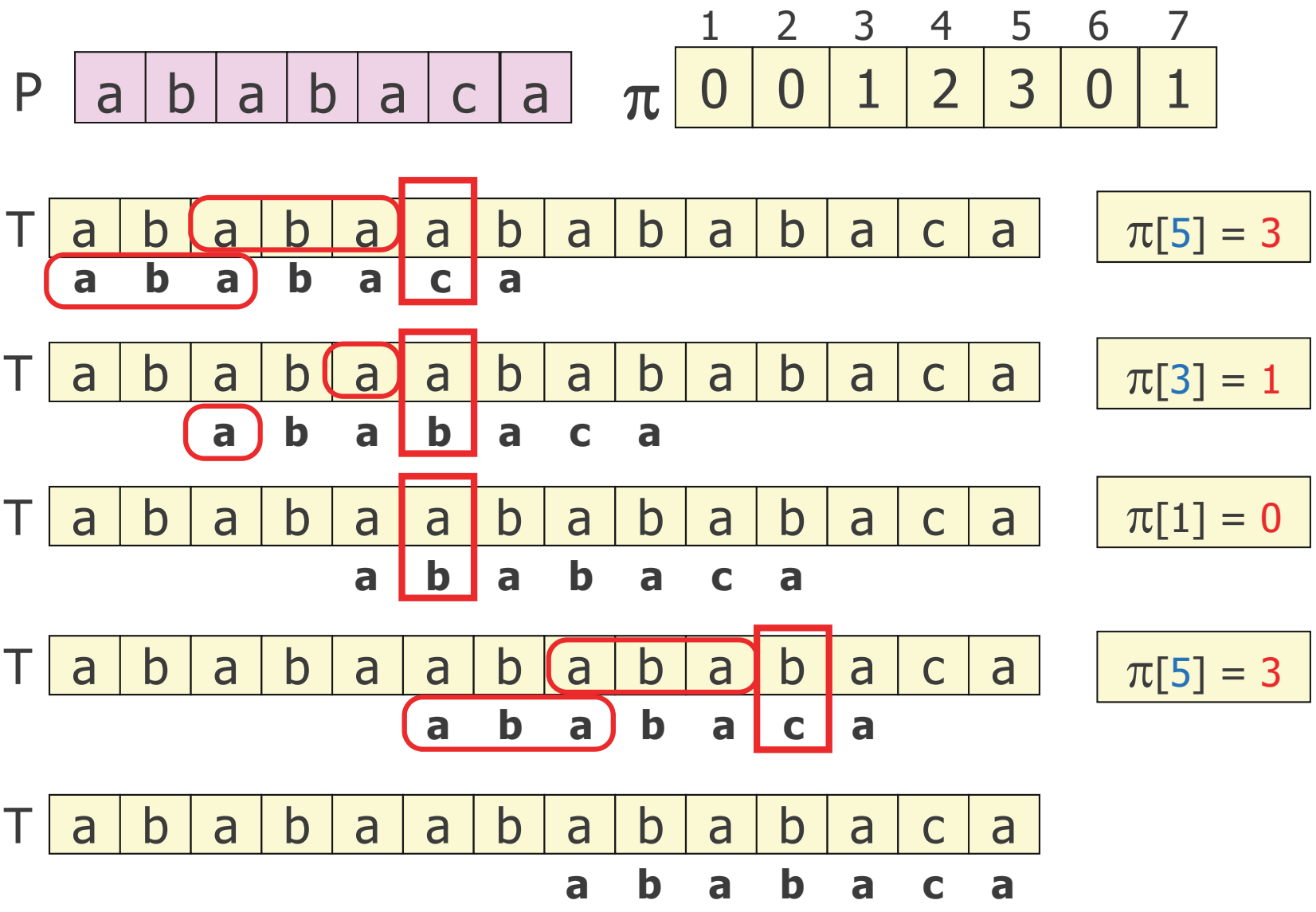


- **ของใหม่ (prefix function)**
  - $\pi(q)$ = max{ k : k<q and (P[1..q]).endsWith(P[1..k]) }
  - match มาได้ q ตัว แต่ mismatch ตัวที่ q+1 ให้ทำต่อที่ $P[\pi(q)+1]$
  - ตารางมีขนาด m
  - ใช้เวลาสร้าง $\Theta(m)$



- **สร้าง π และค้น ใช้เวลารวม $\Theta(n+m)$**

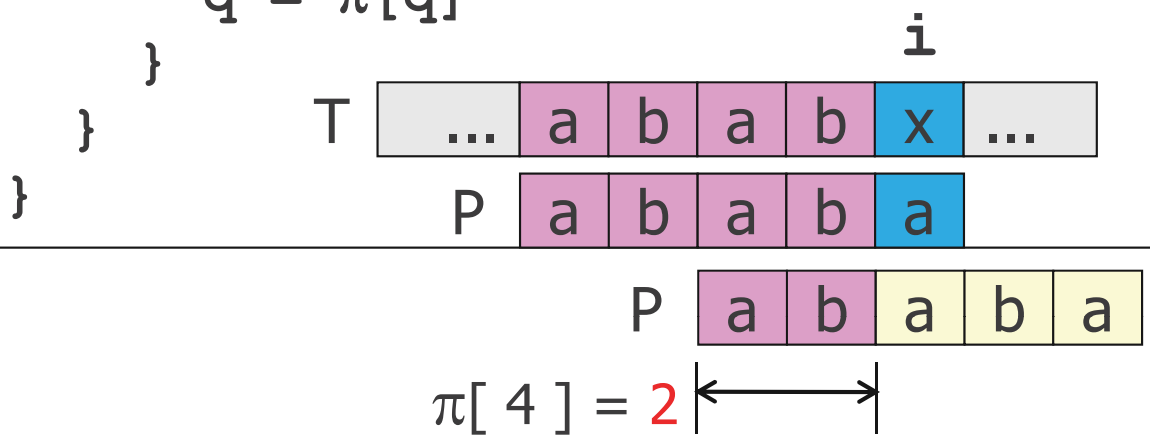
การใช้ Prefix Function



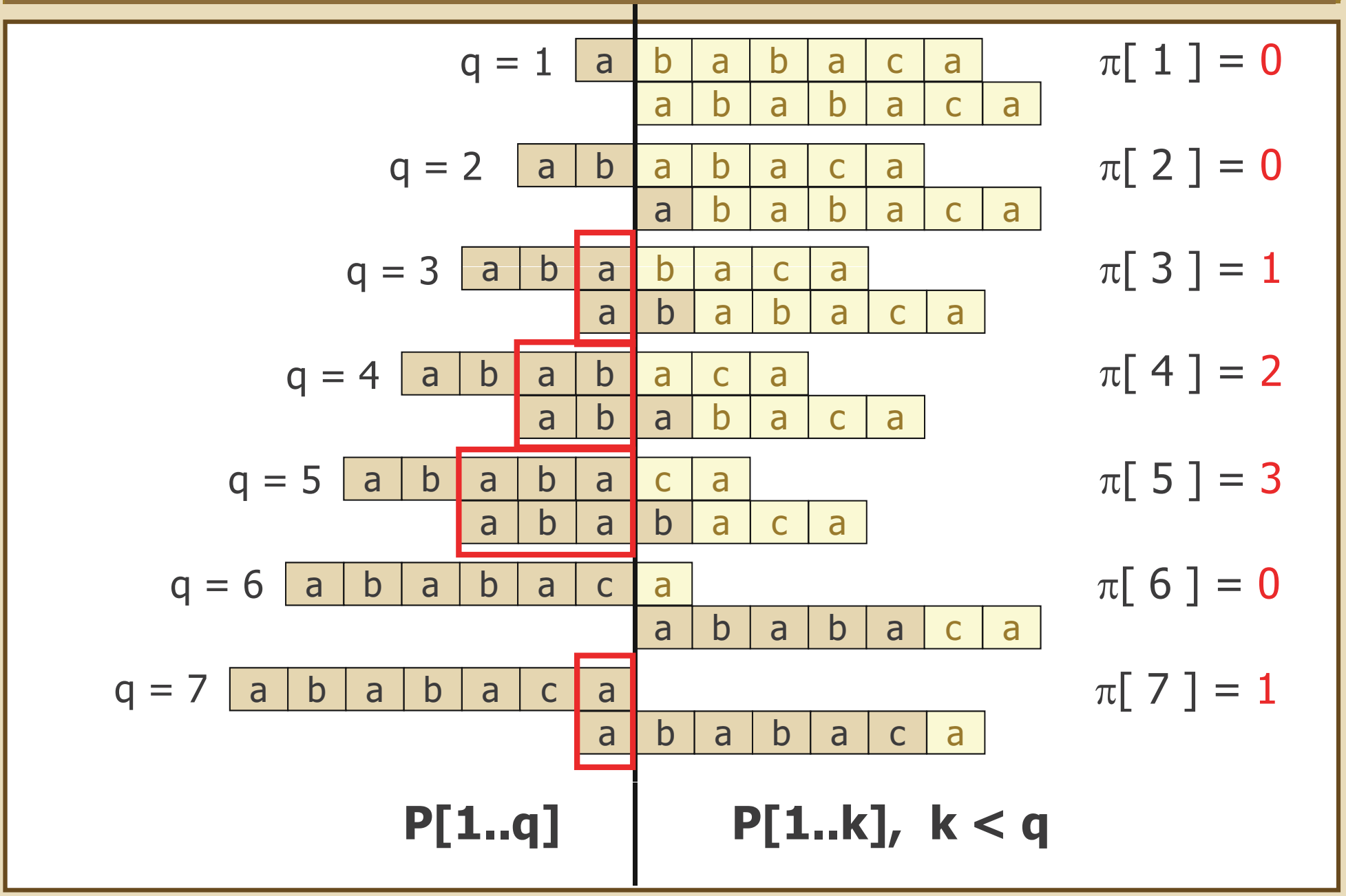
KMP-Matcher

```
KMP-Matcher( T[1..n], P[1..m] ) {
  π = Compute-Prefix-Function(P)   // Θ(m)
  q = 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++
    if (q == m) {
      print( i - m )
      q = π[q]
    }
  }
}
```

$\Theta(n+m)$

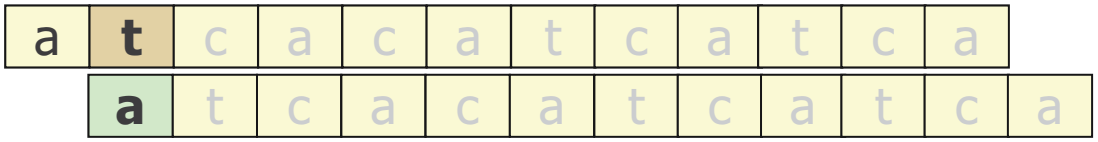


$\pi(q) = \max\{ k : k<q \text{ and } (P[1..q]).\text{endsWith}(P[1..k]) \}$

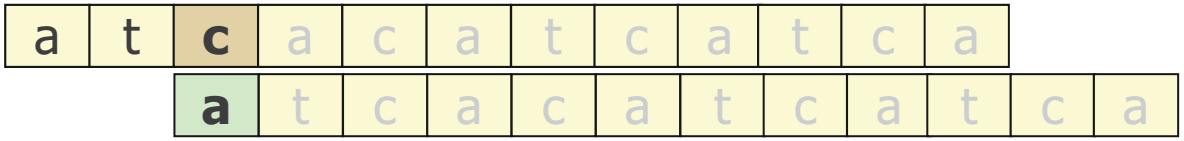


$\pi(q) = \max\{ k : k<q \text{ and } (P[1..q]).\text{endsWith}(P[1..k]) \}$

$\pi[\,1\,] = 0$



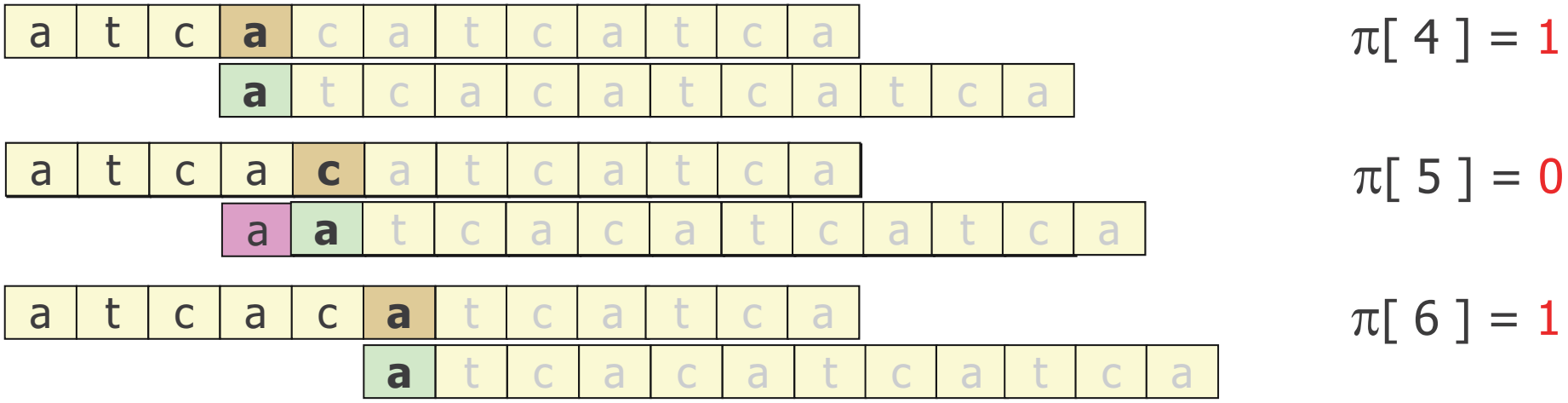
$\pi[\,2\,] = 0$



$\pi[\,3\,] = 0$

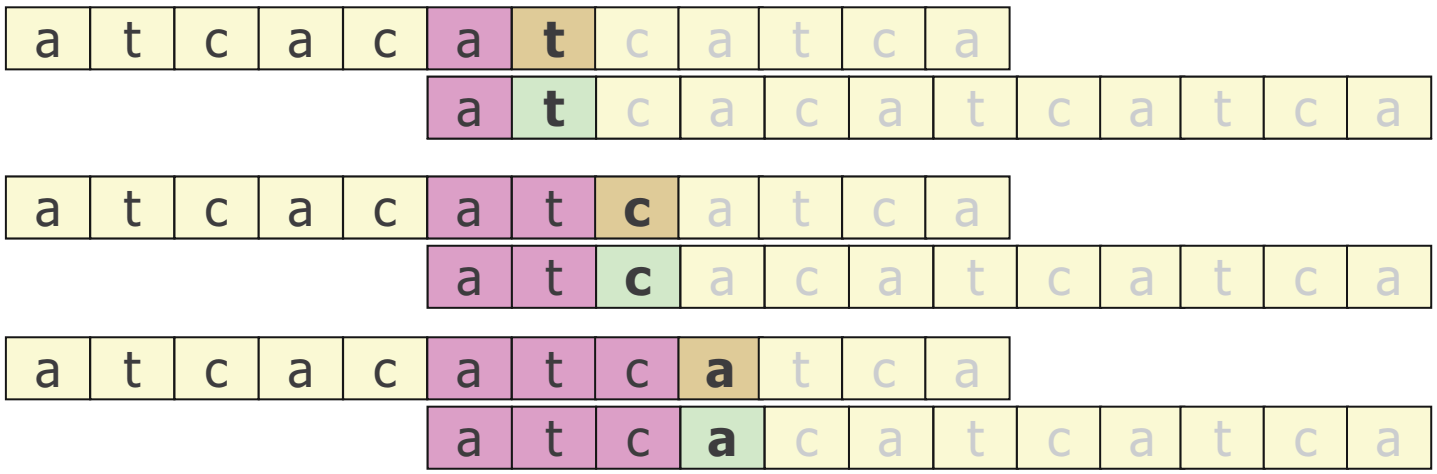
```
Compute-Prefix-Function( P[1..m] ) {
  π[1] = 0;  k = 0
  for ( q = 2 to m ) {

    ...

    if (P[k+1] == P[q]) k++
    π[q] = k
  }
  return π
```

$\pi(q) = \max\{ k : k<q \text{ and } (P[1..q]).\text{endsWith}(P[1..k]) \}$



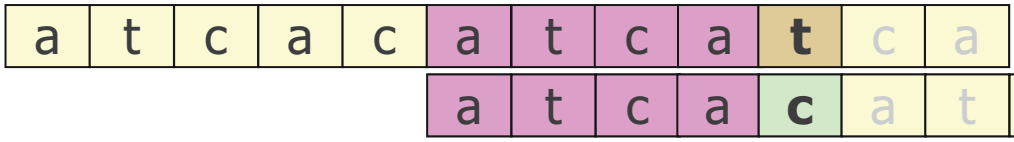

```
Compute-Prefix-Function( P[1..m] ) {
  π[1] = 0;  k = 0
  for ( q = 2 to m ) {
    while (k > 0 and P[k+1] ≠ P[q]) {
      k = π[k]
    }
    if (P[k+1] == P[q]) k++
    π[q] = k
  }
  return π
```

$\pi(q) = \max\{ k : k<q \text{ and } (P[1..q]).\text{endsWith}(P[1..k]) \}$



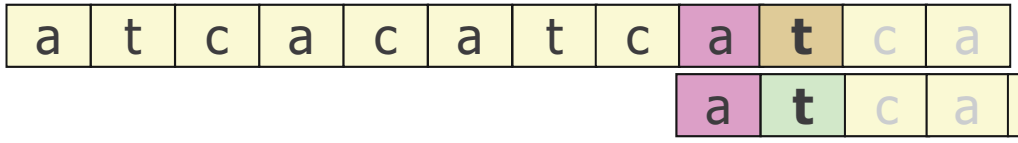

```
Compute-Prefix-Function( P[1..m] ) {
  π[1] = 0;  k = 0
  for ( q = 2 to m ) {
    while (k > 0 and P[k+1] ≠ P[q]) {
      k = π[k]
    }
    if (P[k+1] == P[q]) k++
    π[q] = k
  }
  return π
```

$\pi(q) = \max\{ k : k<q \text{ and } (P[1..q]).\text{endsWith}(P[1..k]) \}$



$\pi[\ 9\ ] = 4 \rightarrow P[1..4] = P[6..9] \rightarrow$ ทดสอบ $P[4+1] = P[10]$?

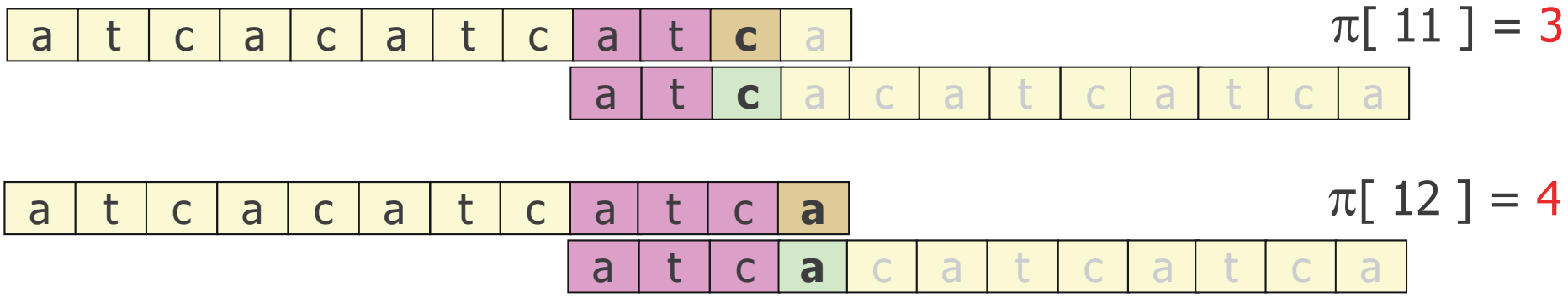
$\pi[\ 4\ ] = 1 \rightarrow P[1..1] = P[4..4] \rightarrow$ ทดสอบ $P[1+1] = P[10]$?



$\pi[\ 10\ ] = 2$

```
Compute-Prefix-Function( P[1..m] ) {
  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++
    π[q] = k
  }
  return π
```

$\pi(q) = \max\{ k : k<q \text{ and } (P[1..q]).\text{endsWith}(P[1..k]) \}$




```
Compute-Prefix-Function( P[1..m] ) {
  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++
    π[q] = k
  }
  return π
```

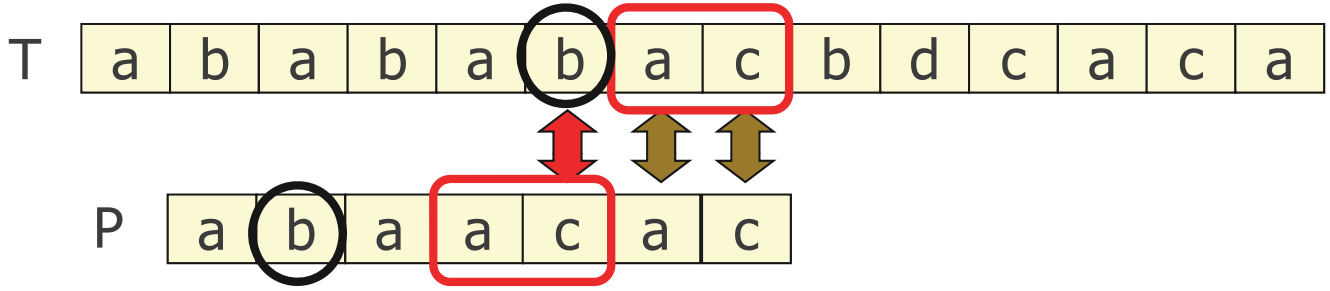
$\Theta(m)$

Boyer-Moore Matcher

Knuth-Morris-Pratt (left-to-right)

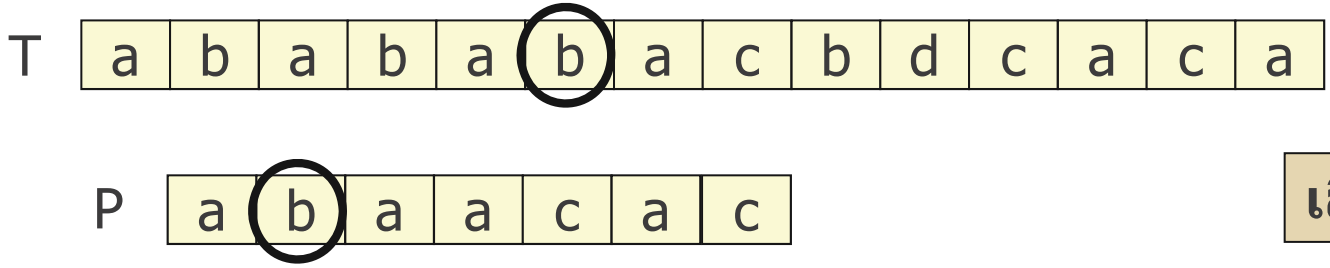
Boyer Moore (right-to-left)

Boyer-Moore : Two heuristics



Bad character

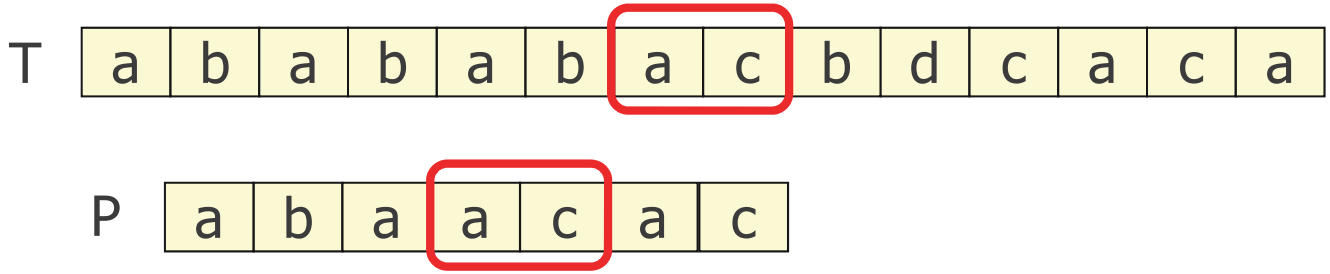
bad-character แนะให้
เลื่อน 3 ตำแหน่ง



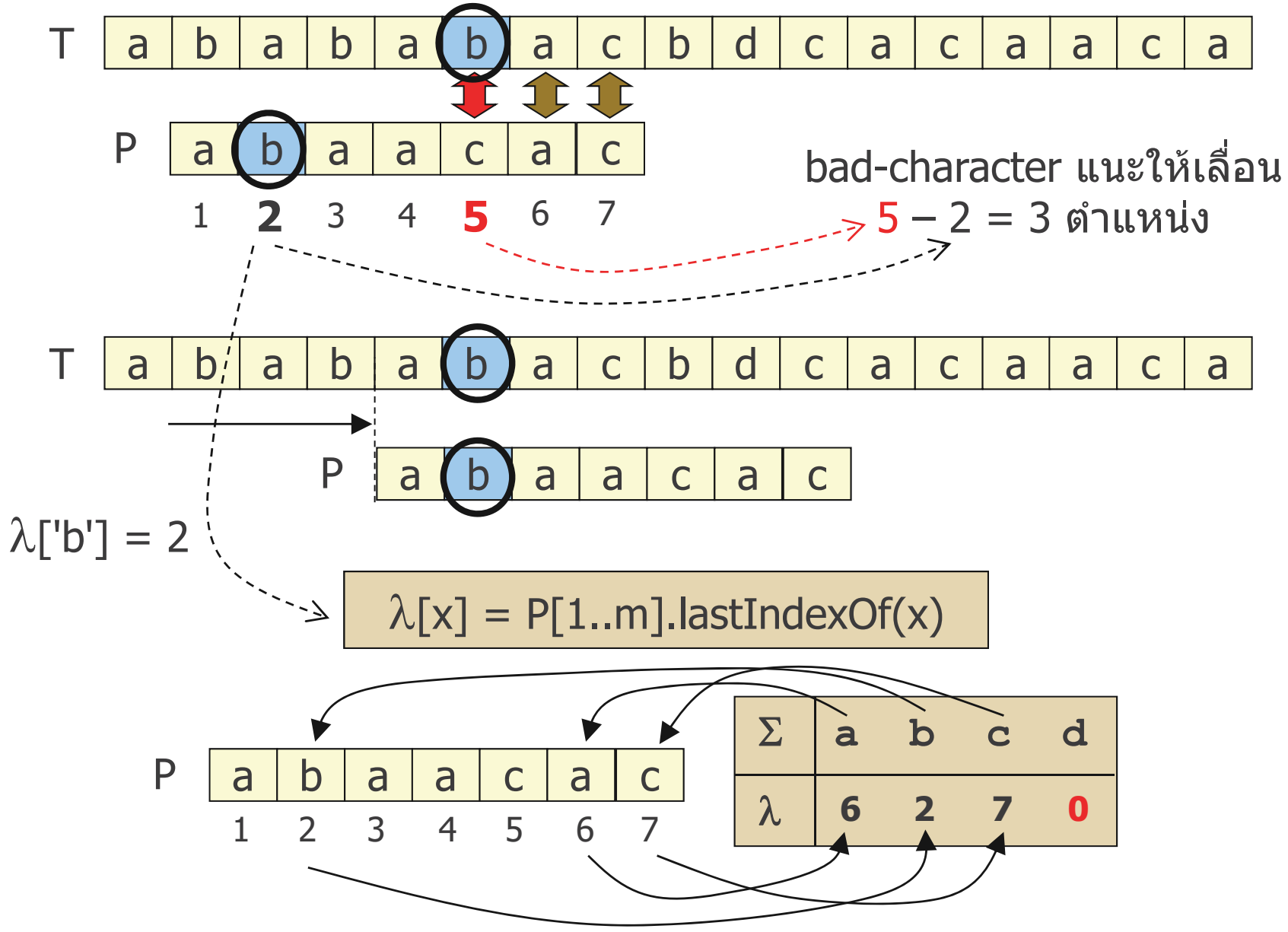
เลือกตัวมาก

Good suffix

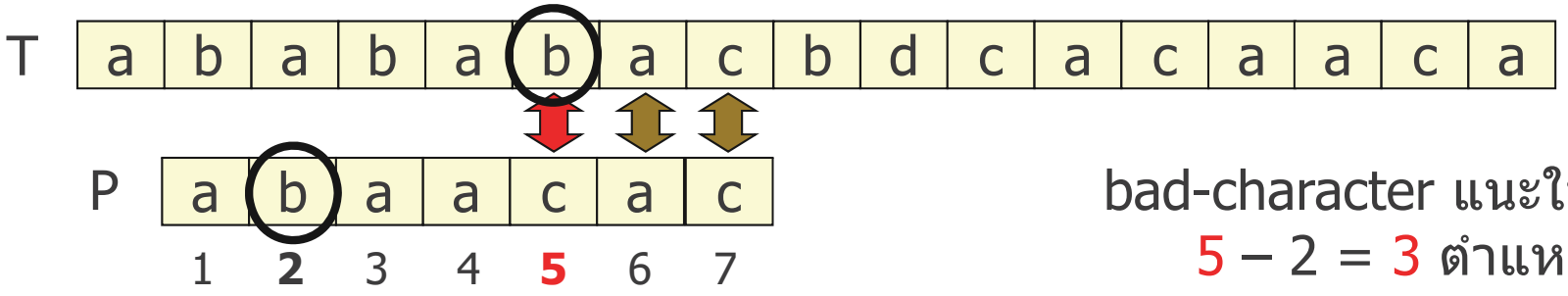
good-suffix แนะให้
เลื่อน 2 ตำแหน่ง



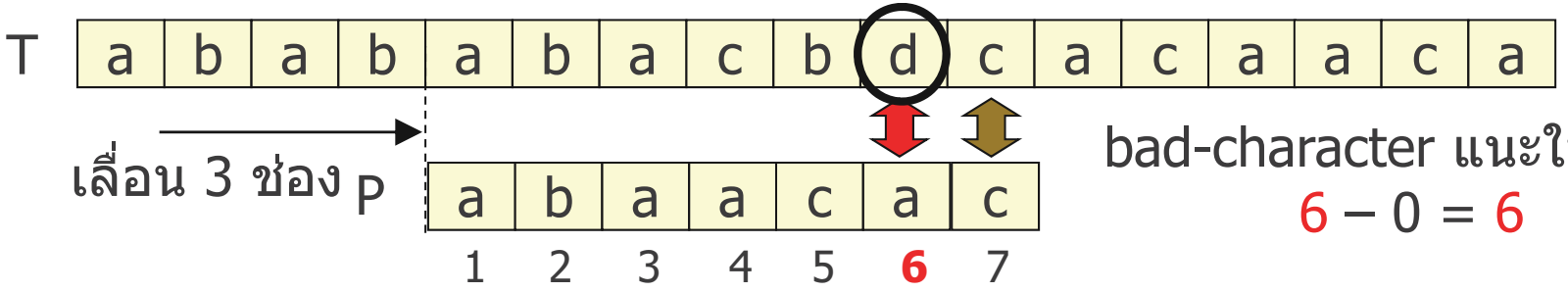
Bad-Character Heuristics



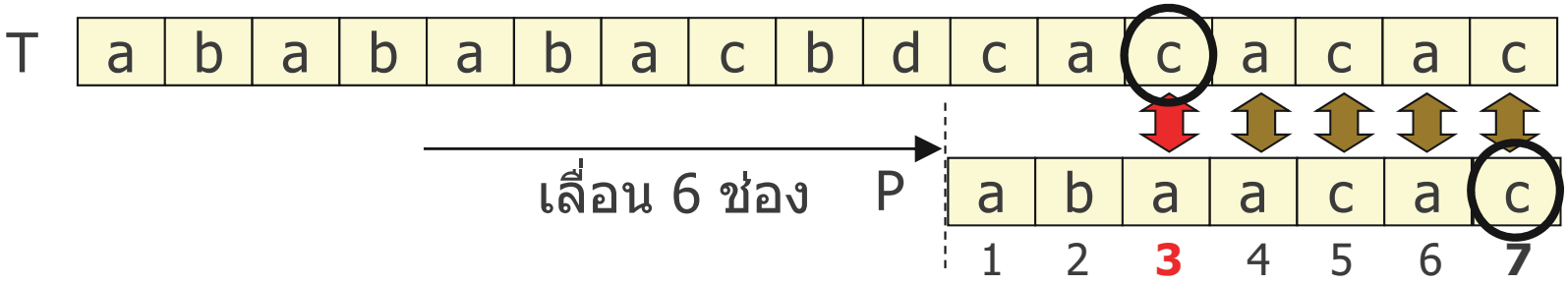
Bad-Character Heuristics



bad-character แนะให้เลื่อน
5 – 2 = 3 ตำแหน่ง



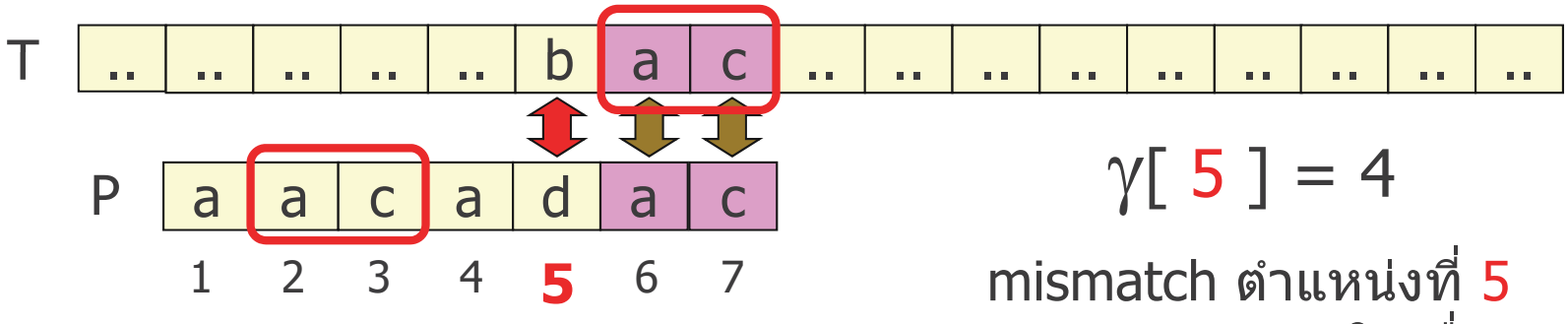
bad-character แนะให้เลื่อน
6 – 0 = 6



bad-character แนะให้เลื่อน
3 – 7 = –4 (ignored)

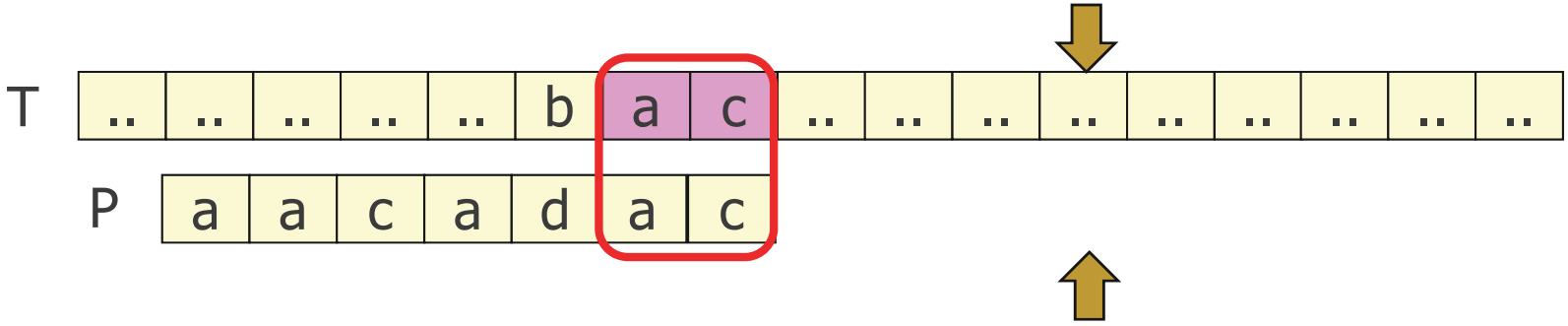
Σ	a	b	c	d
λ	6	2	7	0

Good-Suffix Heuristics

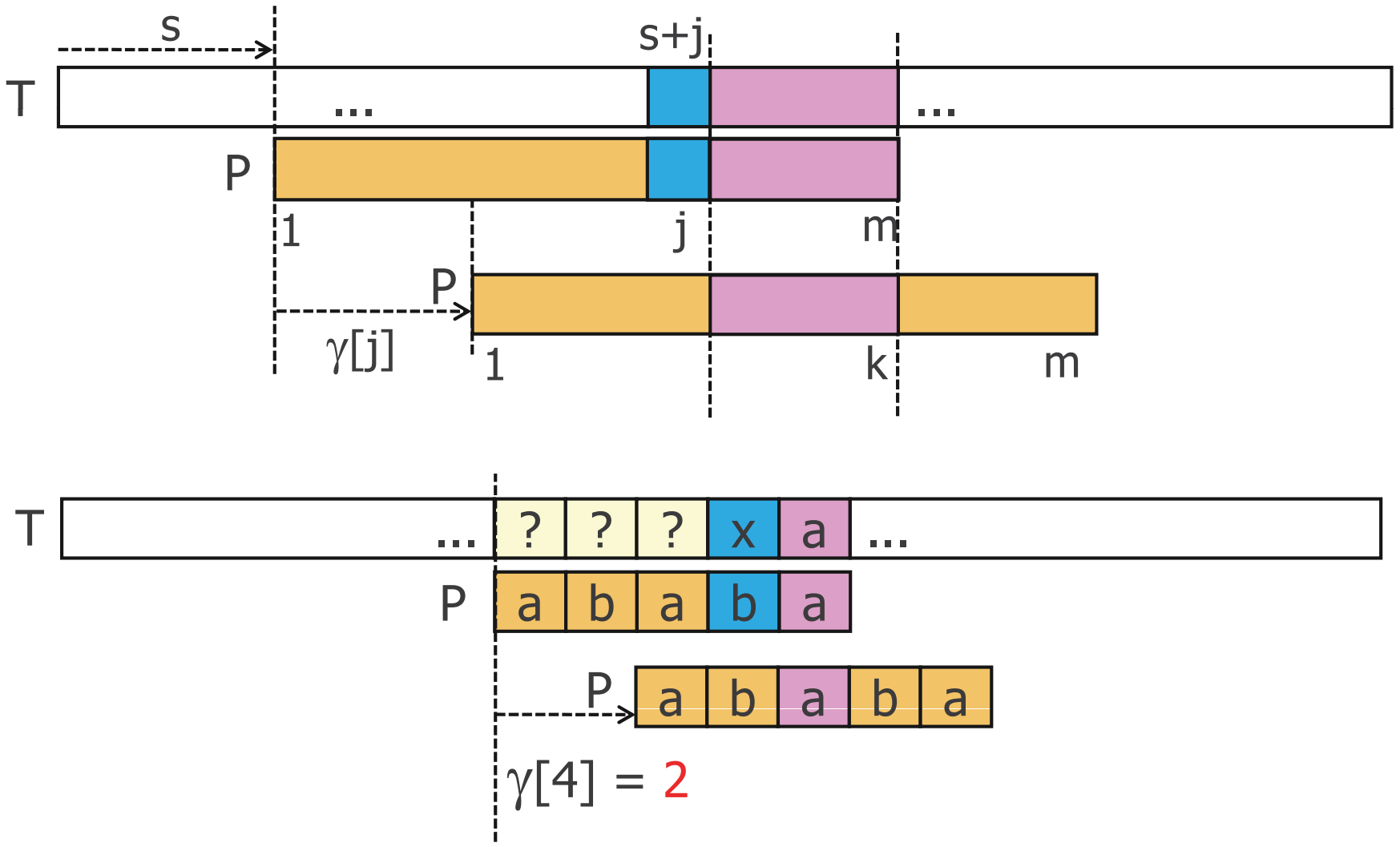


$\gamma[\ 5\ ] = 4$

mismatch ตำแหน่งที่ 5
good-suffix แนะให้เลื่อน
4 ตำแหน่ง

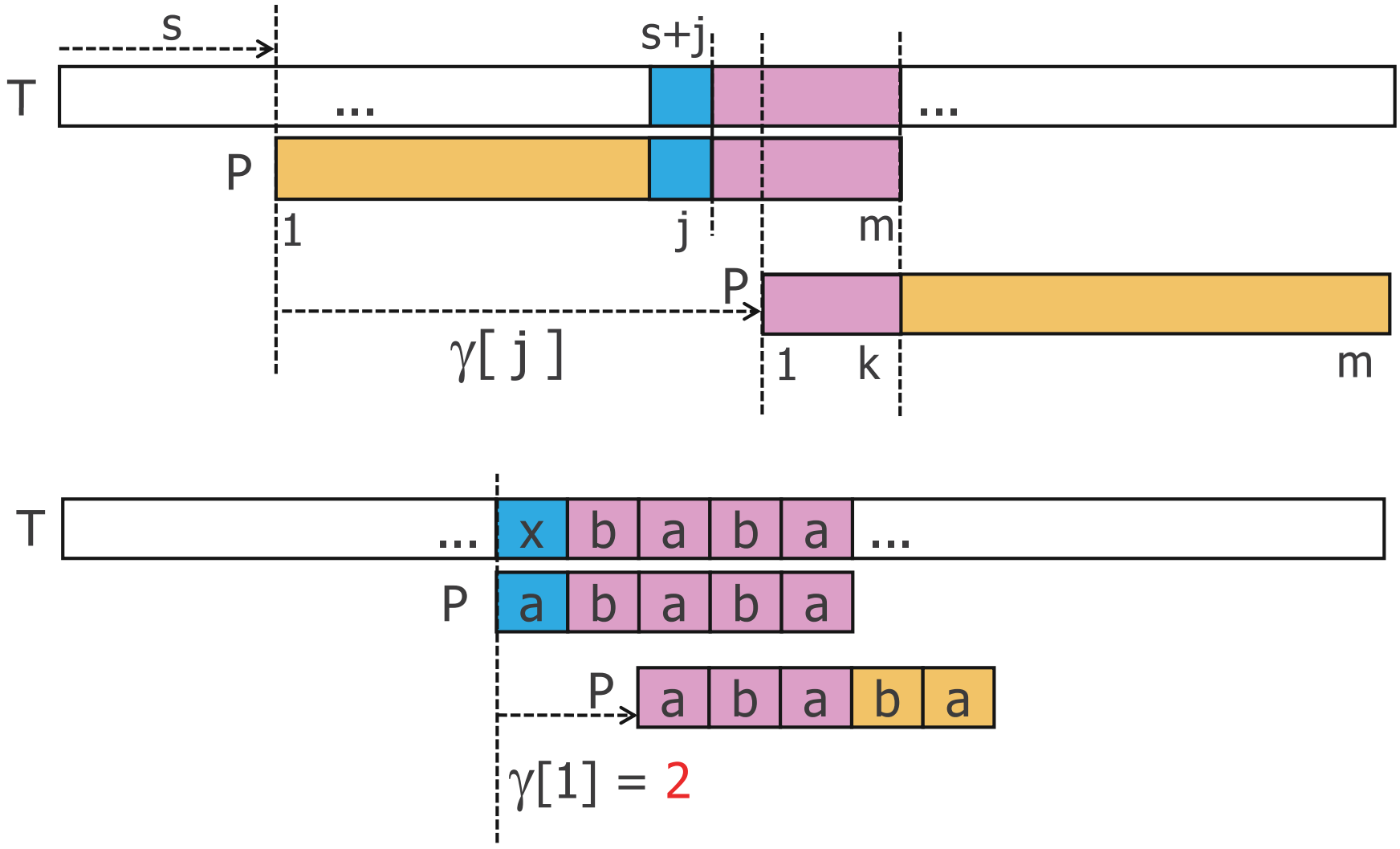


Good-Suffix Heuristics



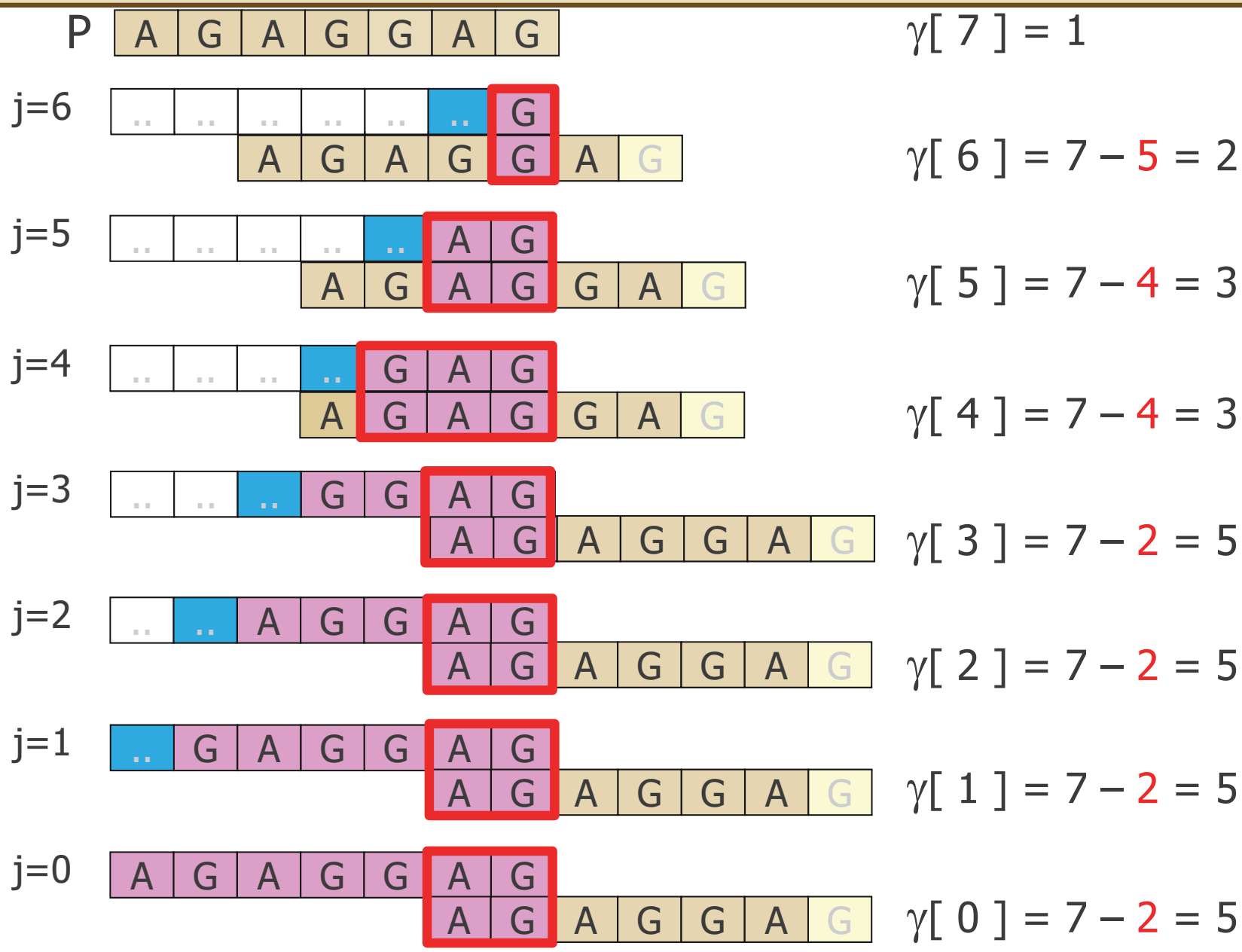
$\gamma[\,j\,] = m - \max\{\, k : 0 \le k < m \text{ and } P[1..k].\text{endsWith}(P[j+1...m]) \,\}$

Good-Suffix Heuristics



$\gamma[\,j\,] = m - \max\{\, k : 0 \le k < m \text{ and } P[j+1...m].\text{endsWith}(P[1..k]) \,\}$

$\gamma[\,j\,] = m - \max\{\, k : 0 \le k < m \text{ and } P[j+1\ldots m] \sim P[1..k] \,\}$



Boyer-Moore Matcher

```
Boyer-Moore-Matcher(T[1..n],P[1..m], Σ)
  λ = LAST-OCCURRENCE(P, m, Σ)   // O(|Σ|+m)
  γ = GOOD-SUFFIX(P, m)          // O(m)
  s = 0
  while( s ≤ n - m ) {
    j = m
    while( j > 0 and P[j] = T[s+j]) j--
    if (j==0) {
      print( s )
      s = s + γ[0]
    } else {
      s = s + max(γ[j], j - λ[T[s+j]])
    }
  }
```

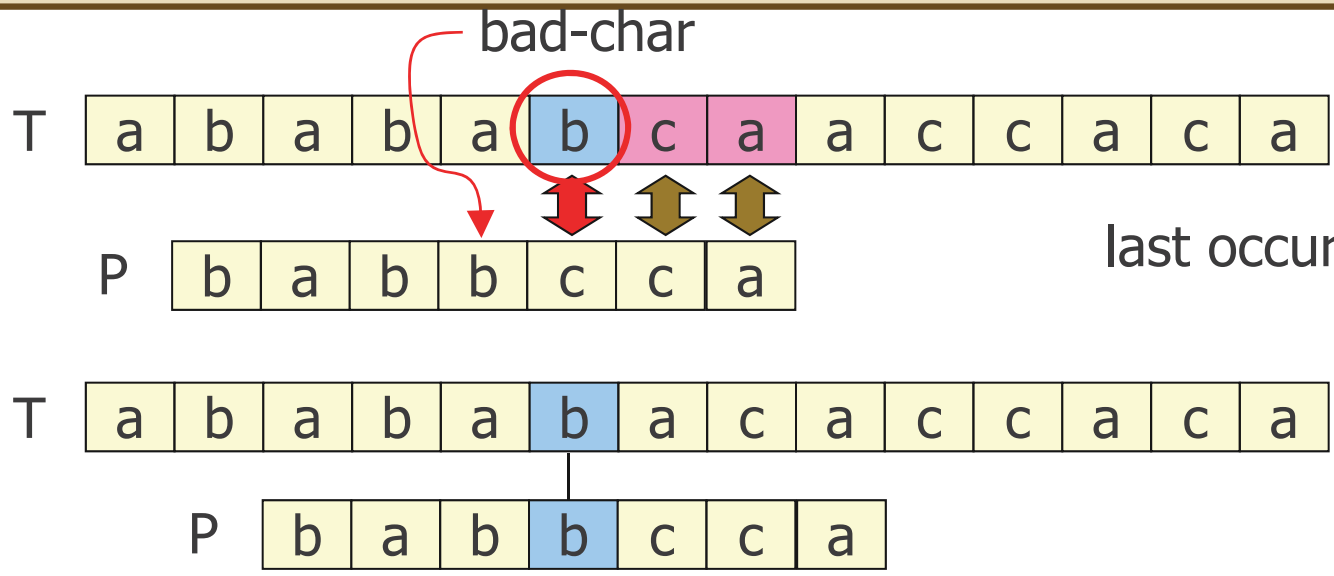
worst : $O((n-m+1)m+|\Sigma|)$

best : $O(n/m+m+|\Sigma|)$

Boyer-Moore Matcher

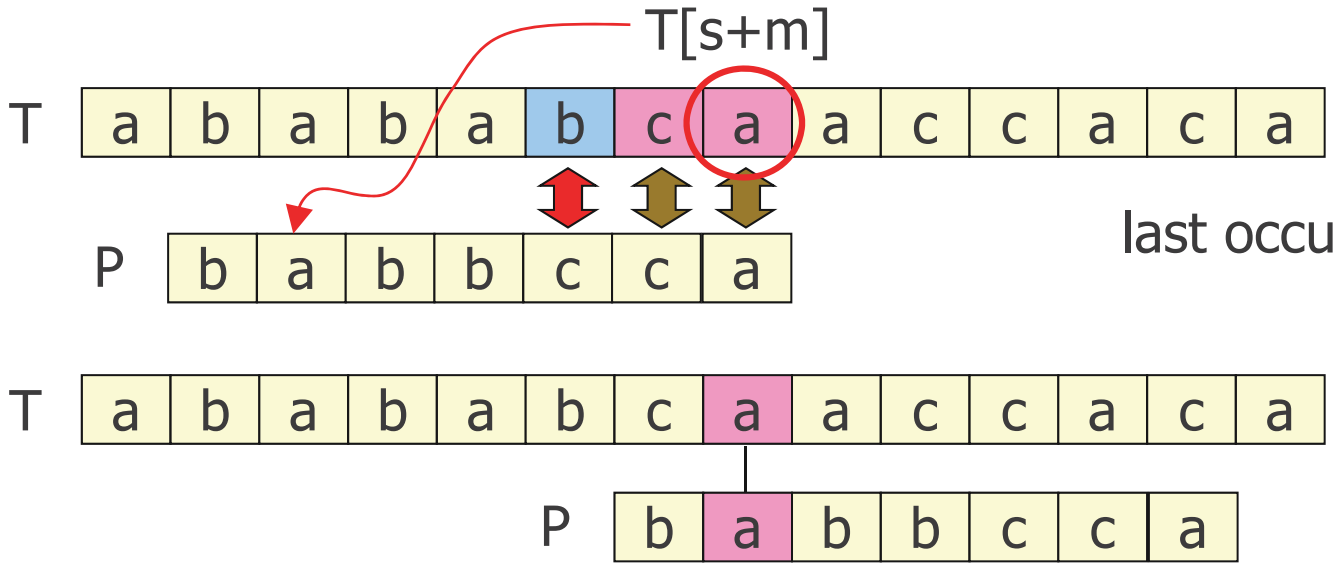
```
Boyer-Moore-Matcher(T[1..n],P[1..m], Σ)
  λ = LAST-OCCURRENCE(P, m, Σ)
  γ = GOOD-SUFFIX(P, m)
  s = 0
  while( s ≤ n - m ) {
    j = m
    while( j > 0 and P[j] = T[s+j]) j--
    if (j==0) {
      print( s )
      s = s + γ[0];   1
    } else {
      s = s + max(γ[j], j - λ[T[s+j]]);   1
    }
  }
}
```

Horspool Matcher



Boyer-Moore

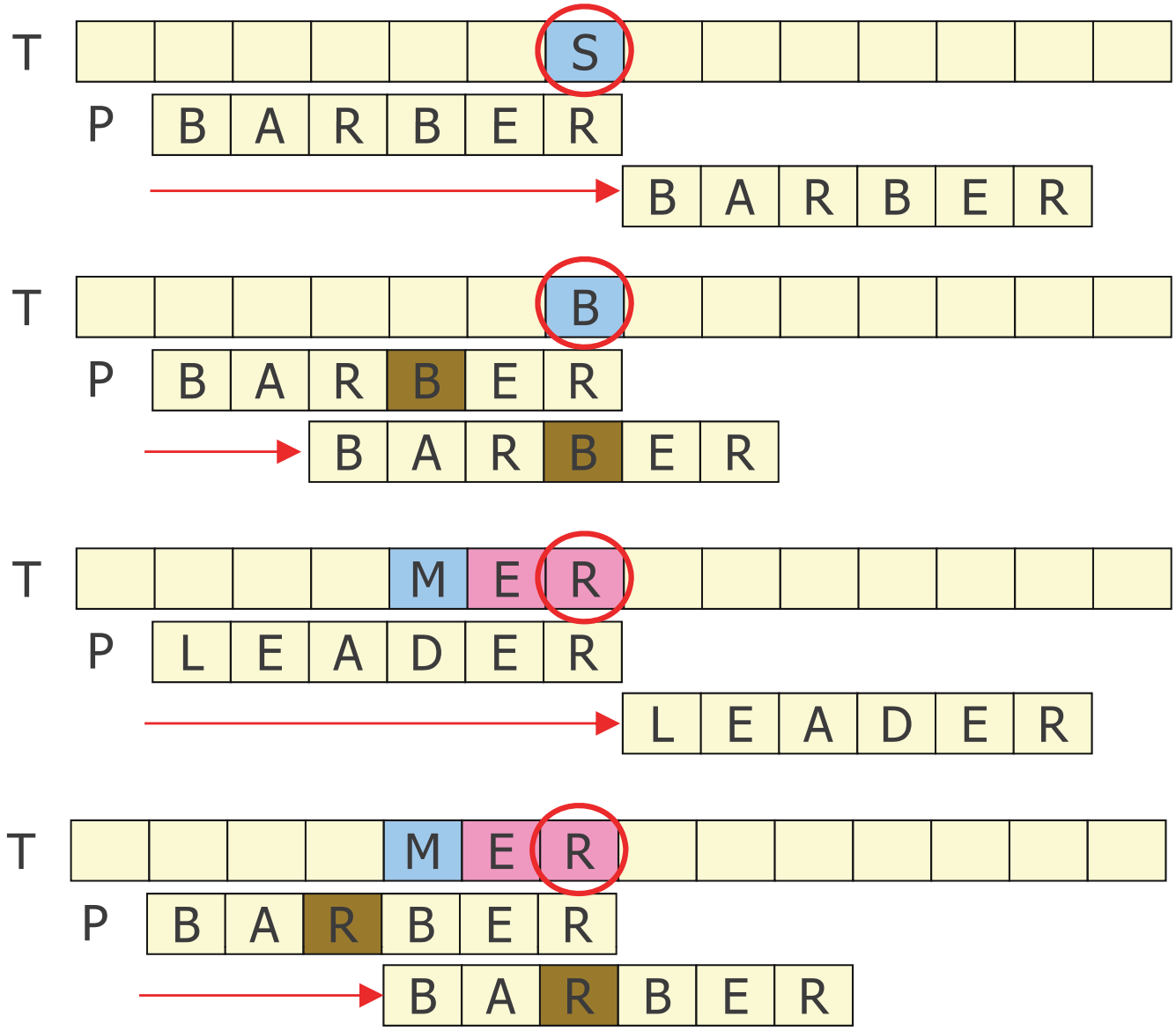
last occurence of bad-char



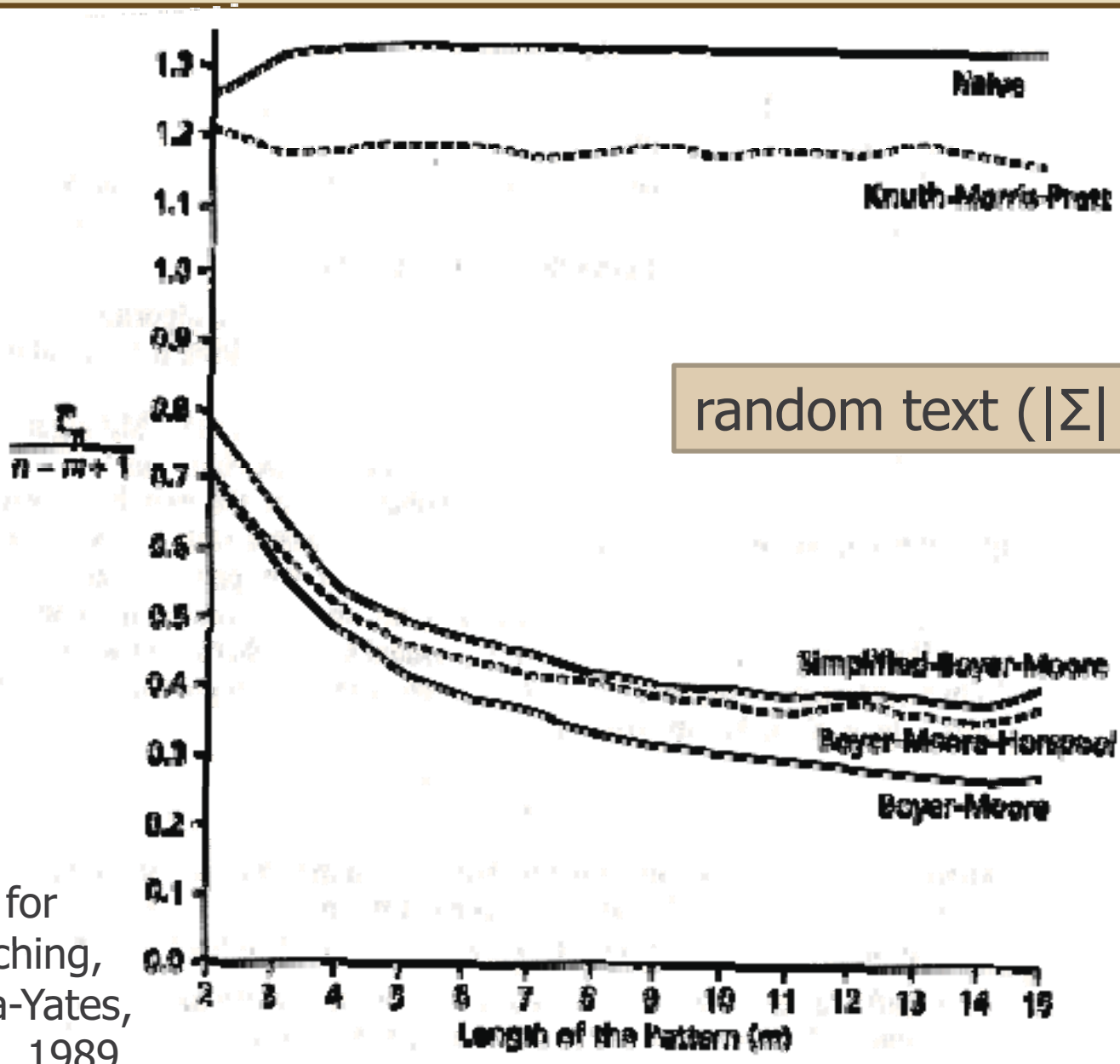
Horspool

last occurence of T[s+m]

Horspool Matcher



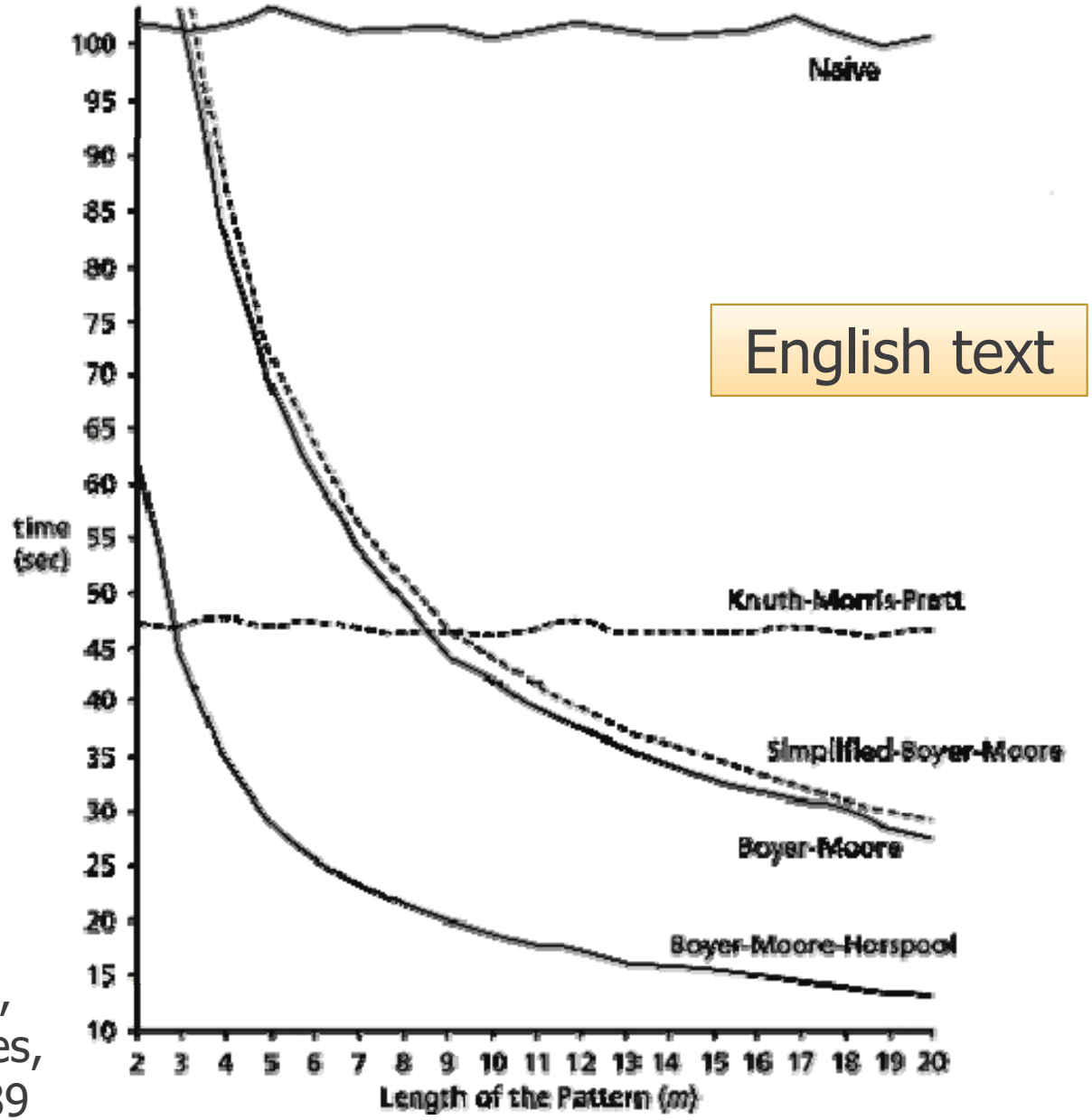
ประสิทธิภาพการทำงาน



random text (|Σ| = 4)

Algorithms for string searching, R. A. Baeza-Yates, ACM SIGIR, 1989

ประสิทธิภาพการทำงาน



English text

Algorithms for string searching, R. A. Baeza-Yates, ACM SIGIR, 1989