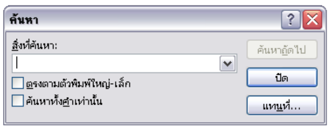
String Matching/ Pattern Matching algorithm

Mr. Akarapon Watcharapalakorn
Doctor of Optometry (O.D.)

Syllabus

String Matching/Pattern Matching algorithm
- Brute Force algorithm (Naive)
- Rabin-Karp algorithm
- Finite Automaton algorithm
- Knutt-Morris-Pratt algorithm
- Boyer-Moore algorithm
- Horspool algorithm



Exact 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.

you

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

Approximate String Match.

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.

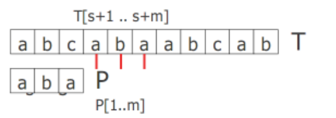
Output

ปัญหา String Matching

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

ต้องการหา
all valid shifts
first valid shift
last valid shift

$0 \le s \le n - m$

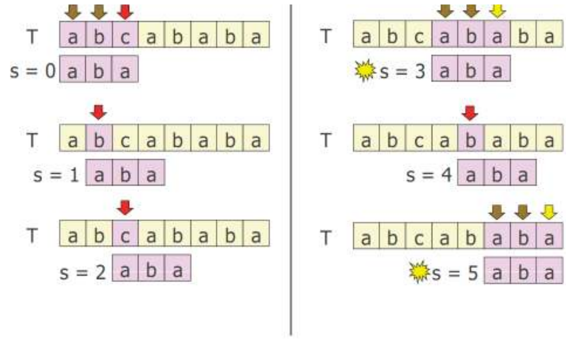


s เป็น valid shift เมื่อ P[1..m] = T[s+1 .. s+m]

อัลกอริทึมการจับคู่สตริง

- ✓ 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

Brute Force(Naive) Algo.



Brute Force(Naive) Algo.

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

worst case : T = AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAZ
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	$\left(\frac{1}{d}\right)^2\left(\frac{d-1}{d}\right)$	$\times 3$	$\sum_{k=1}^{m}\frac{k}{d^{k-1}}\left(\frac{d-1}{d}\right)$
...			
m – 1 ตัวแรกเหมือน ตัวที่ m ไม่เหมือน	$\left(\frac{1}{d}\right)^{m-1}\left(\frac{d-1}{d}\right)$	$\times m$	+
เหมือนกันทั้ง m ตัว	$\left(\frac{1}{d}\right)^m$	$\times m$	$\frac{m}{d^m}$

Random T&P : Avg #Cmps

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

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

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

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

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

Random T&P : Avg #Cmps

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

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

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

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

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

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 )
  }
}
```

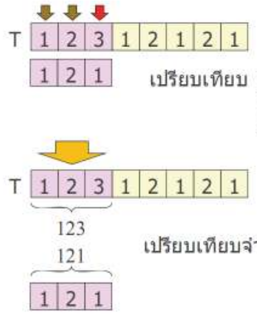
$$\frac{1-d^{-m}}{1-d^{-1}}$$

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

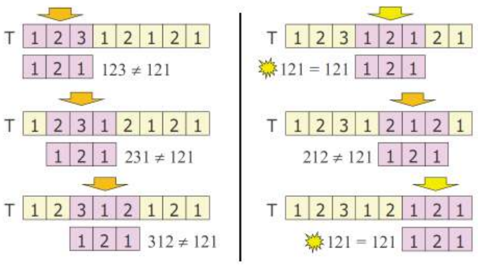
$$\frac{1-d^{-m}}{1-d^{-1}} < \frac{1}{1-d^{-1}},\quad d \ge 2$$

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

Rabin-Karp Matcher

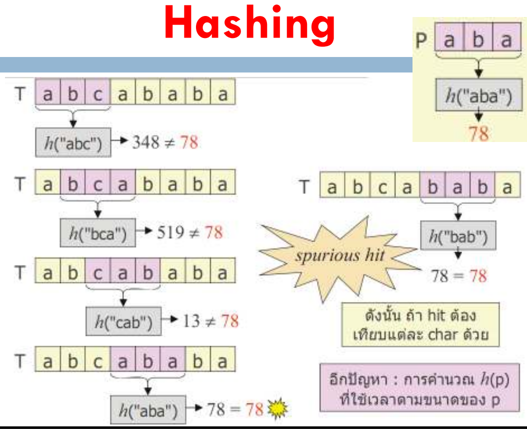


Rabin-Karp Algorithm



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

Hashing



การคำนวณค่า h: incremental

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

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

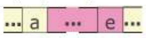
$= 37\times($ h("abc") $- 1\times37^2) + 5\times37^0$

เลื่อนตำแหน่ง "bc" ไปทางซ้าย

ลบ "a"

เพิ่ม "e"

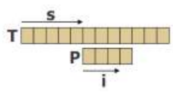
h("...e") = $37\times($ h("a...") $- 1\times37^{m-1}) + 5\times37^0$



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...") $- 1\times37^{m-1}) + 5\times37^0$

Rabin-Karp Algorithm

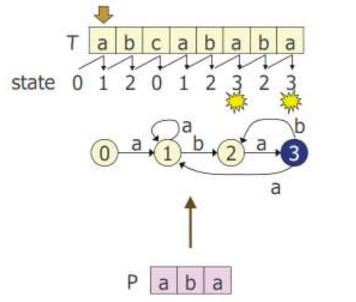
```
Rabin-Karp-Matching( T[1..n], P[1..m], d, q ) {
  ht = 0; hp = 0; dm1 = d^(m-1) % q
  for ( i = 1 to m ) {
    ht = (d*ht + T[i]) % q
    hp = (d*hp + P[i]) % q
  }
  for ( s = 0 to n - m ) {
    if (ht == hp) {
      for ( i = 1 to m )
        if (T[s+i] ≠ P[i]) break
      if (i > m) print( s )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

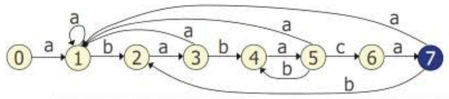
Finite Automaton Matcher



สร้าง state transition diagram จาก pattern

Finite Automaton Matcher

P a b a b a c a



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 )
  }
}
```

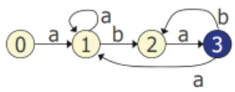
transition function

$\Theta(n)$

สร้าง transition function อย่างไร ?

Transition Function

P a b a



Σ = {a, b, c}

input |Σ|

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

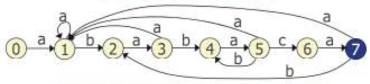
m+1

ตารางมีขนาด (m+1)|Σ|

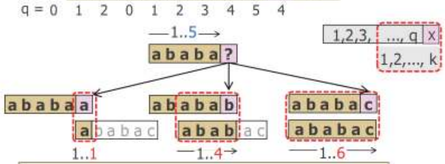
δ(state, input)

Transition Function

P a b a b a c a



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



δ(q, x) = max{ k : (P[1..q]+x).endsWith(P[1..k]) }

ตัวอย่าง Transition Function

δ(q, x) = max{ k : (P[1..q]+x).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

δ(5,"b") = 4

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

P[1..k]: ~~a~~, ab, ~~aba~~, abab, ~~ababa~~, ~~ababac~~, ~~ababaca~~

abab: P[1..4]

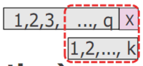
สร้างแบบตรงไปตรงมาใช้เวลา O(m³|Σ|)

มีวิธีสร้างที่ใช้เวลา O(m|Σ|)

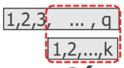
รวมการค้นด้วย ใช้เวลา O(n + m|Σ|)

Knutt-Morris-Pratt Matcher

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

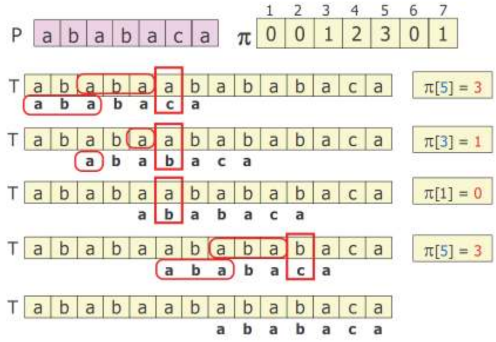


- **ของใหม่ (prefix function)**
  - $\pi(q)$ = max{ k : k<q and (P[1..q]).endsWith(P[1..k]) }
  - match มาได้ q ตัว แต่ mismatch ตัวที่ q+1 ให้ทำต่อที่ P[π(q)+1]
  - ตารางมีขนาด m
  - ใช้เวลาสร้าง $\Theta(m)$



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

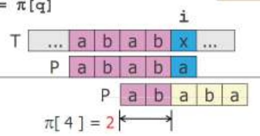
การใช้ Prefix Function



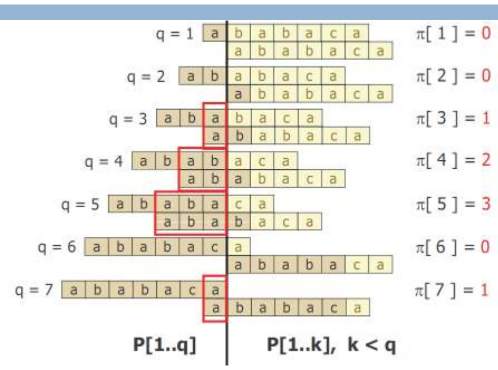
KMP Matcher

```
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]
    }
  }
}
```

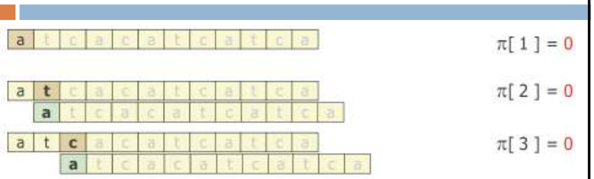
$\Theta(n+m)$



π(q) = max{ k : k<q and (P[1..q]).endsWith(P[1..k]) }



π(q) = max{ k : k<q and (P[1..q]).endsWith(P[1..k]) }

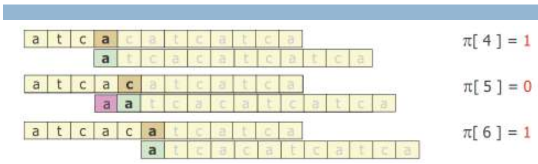



```
Compute-Prefix-Function( P[1..m] ) {
  π[1] = 0;  k = 0
  for ( q = 2 to m ) {

    ...

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

π(q) = max{ k : k<q and (P[1..q]).endsWith(P[1..k]) }

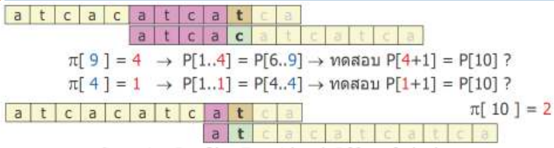



```
Compute-Prefix-Function( P[1..m] ) {
  π[1] = 0;  k = 0
  for ( q = 2 to m ) {
    while (k > 0 and P[k+1] ≠ P[q]) {
      k = π[k]
    }
    if (P[k+1] == P[q]) k++
    π[q] = k
  }
  return π
```

π(q) = max{ k : k<q and (P[1..q]).endsWith(P[1..k]) }

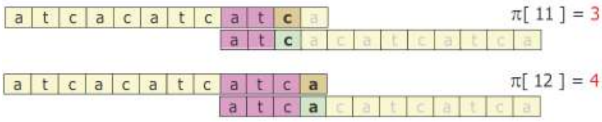

```
Compute-Prefix-Function( P[1..m] ) {
  π[1] = 0;  k = 0
  for ( q = 2 to m ) {
    while (k > 0 and P[k+1] ≠ P[q]) {
      k = π[k]
    }
    if (P[k+1] == P[q]) k++
    π[q] = k
  }
  return π
```

π(q) = max{ k : k<q and (P[1..q]).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
```

π(q) = max{ k : k<q and (P[1..q]).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

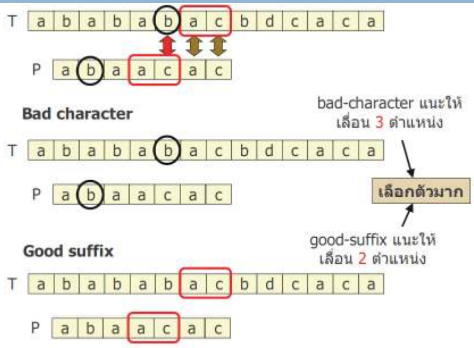
Knuth-Morris-Pratt (left-to-right)

```
A STRING SEARCHING EXAMPLE CONSISTING OF ...
                ...        STING
```

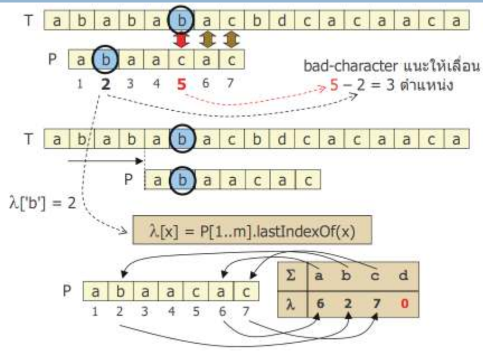
Boyer Moore (right-to-left)

```
A STRING SEARCHING EXAMPLE CONSISTING OF ...
                              STING
```

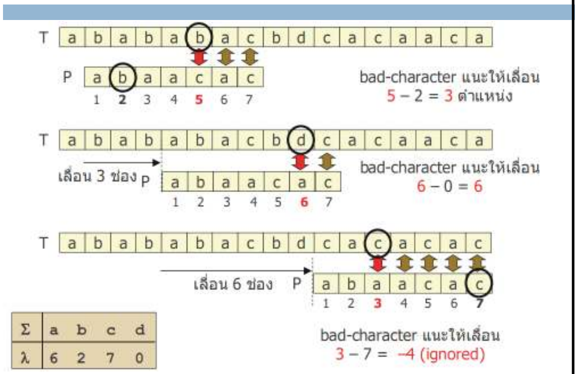
Boyer-Moore: two heuristics



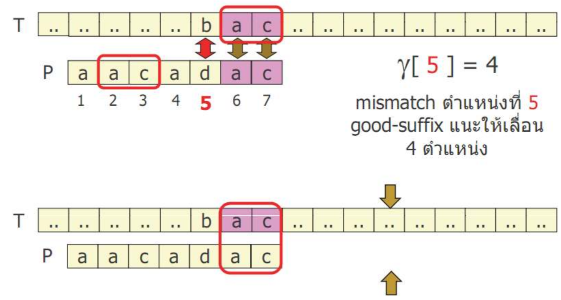
Bad-Character Heuristics



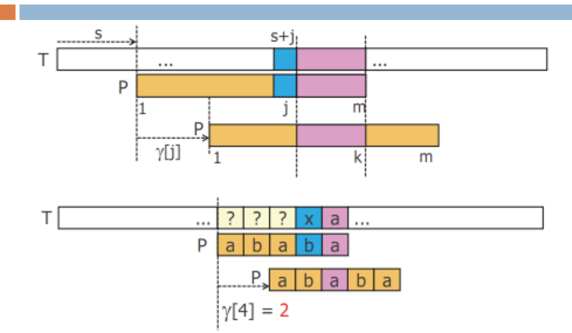
Bad-Character Heuristics



Good-Suffix Heuristics

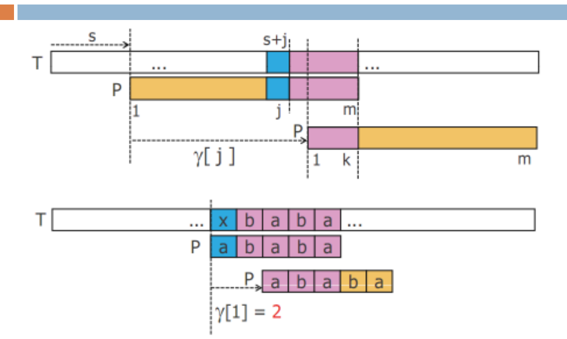


Good-Suffix Heuristics



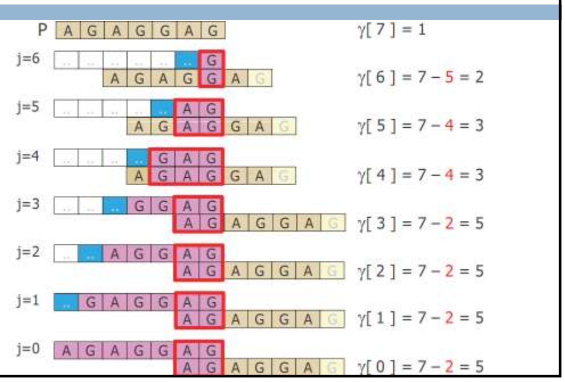
γ[j] = m – max { k : 0 ≤ k < m and P[1..k].endsWith(P[j+1...m]) }

Good-Suffix Heuristics



γ[j] = m – max { k : 0 ≤ k < m and P[j+1...m].endsWith(P[1..k]) }

γ[j] = m – max { k : 0 ≤ k < m and P[j+1...m] ~ P[1..k] }



Boyer-Moore Matcher

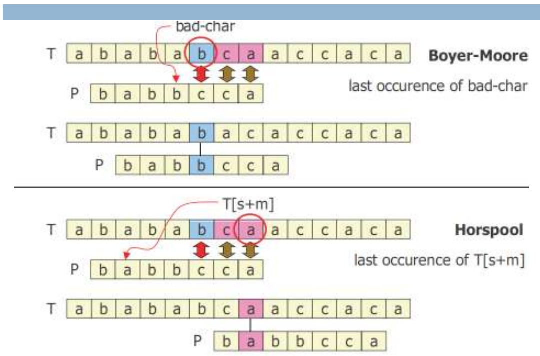
```
Boyer-Moore-Matcher(T[1..n],P[1..m], Σ)
  λ = LAST-OCCURRENCE(P, m, Σ)   // O(|Σ|+m)
  γ = GOOD-SUFFIX(P, m)          // O(m)
  s = 0
  while( s ≤ n – m ) {
    j = m
    while( j > 0 and P[j] = T[s+j]) j--
    if (j==0) {
      print( s )
      s = s + γ[0]
    } else {
      s = s + max(γ[j], j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

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

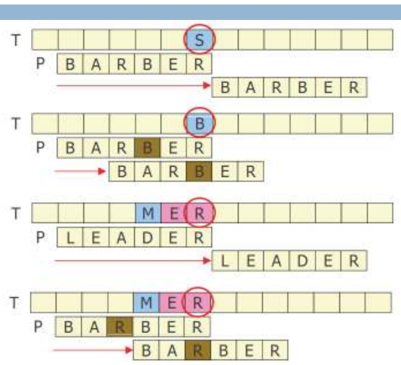
Boyer-Moore Matcher

```
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
    }
  }
}
```

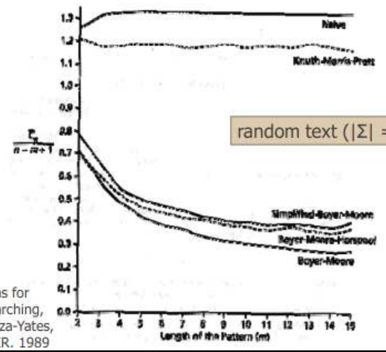
Horspool Matcher



Horspool Matcher

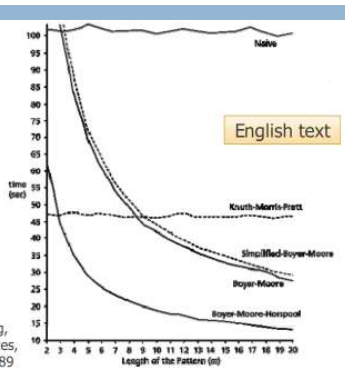


ประสิทธิภาพการทำงาน



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

ประสิทธิภาพการทำงาน



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

THE END