Algorithms, Correctness & Efficiency – G52ACE

KMP Fast String Matching (guest lecture, covering for Professor Brian Logan)

Thomas Gärtner

Professor of Data Science
School of Computer Science, University of Nottingham

University of Nottingham
UK | CHINA | MALAYSIA

Naïve string matching

```
  P: dear_alice,_dear_bob
  T: Lorem_ipsum_dolor_sit_amet, consectetur…
i=0  dear_alice,_dear_bob          j=0
  1   dear_alice,_dear_bob         j=0
  2    dear_alice,_dear_bob        j=0
  3     dear_alice,_dear_bob       j=0
  4      dear_alice,_dear_bob      j=0
  5       dear_alice,_dear_bob     j=0
  6        dear_alice,_dear_bob     j=0
  7         dear_alice,_dear_bob     j=0
  8          dear_alice,_dear_bob     j=0
  9           dear_alice,_dear_bob     j=0
 10            dear_alice,_dear_bob     j=0
 11             dear_alice,_dear_bob     j=0
```

NaiveStringMatch(T, P):
input: strings T, P
output: 1st position of P in T
or -1 if P does not occur in T

for i **from** 0 **to** $|T| - |P|$
  for j **from** 0 **to** $|P| - 1$
    if $P_j \neq T_{i+j}$ **then** **break**
  if $j = |P|$ **then** **return** i
return -1

Partial Matches

could be any character **BUT** a d

```
        i
  T: …dear_alice,_#…
i-1     dear_alice,_dear_bob
i       dear_alice,_dear_bob      j=12
```

NaiveStringMatch(T, P):
input: strings T, P
output: 1st position of P in T
or -1 if P does not occur in T

for i **from** 0 **to** $|T| - |P|$
  for j **from** 0 **to** $|P| - 1$
    if $P_j \neq T_{i+j}$ **then** **break**
  if $j = |P|$ **then** **return** i
return -1

Shifting (case 1)

```
        i
i       dear_alice,_dear_bob      j=12
i+1      dear_alice,_dear_bob
i+2       dear_alice,_dear_bob
…          ddddddd
i+11                dear_alice,_dear_bob
i+12                 dear_alice,_dear_bob
i+13                  dear_alice,_dear_bob
```

NaiveStringMatch(T, P):
input: strings T, P
output: 1st position of P in T
or -1 if P does not occur in T

for i **from** 0 **to** $|T| - |P|$
  for j **from** 0 **to** $|P| - 1$
    if $P_j \neq T_{i+j}$ **then** **break**
  if $j = |P|$ **then** **return** i
return -1

Shifting (case 1)

```
        i
i       dear_alice,_dear_bob      j=12

i+13                  dear_alice,_dear_bob
```

NaiveStringMatch(T, P):
input: strings T, P
output: 1st position of P in T
or -1 if P does not occur in T

for i **from** 0 **to** $|T| - |P|$
  for j **from** 0 **to** $|P| - 1$
    if $P_j \neq T_{i+j}$ **then** **break**
  if $j = |P|$ **then** **return** i
return -1

we break on the pattern character in j[th] position
-> we were able to match the j characters before that

the pattern character in 0th position appears again **at but not before**
the jth position -> move i by $j + 1$ positions., i.e., $i \leftarrow i + j + 1$, and restart j

Shifting (case 2)

```
        i
i       dear_alice,_dear_bob      j=5
         dddd
i+5          dear_alice,_dear_bob  j=0
```

NaiveStringMatch(T, P):
input: strings T, P
output: 1st position of P in T
or -1 if P does not occur in T

for i **from** 0 **to** $|T| - |P|$
  for j **from** 0 **to** $|P| - 1$
    if $P_j \neq T_{i+j}$ **then** **break**
  if $j = |P|$ **then** **return** i
return -1

Shifting (case 2)

```
        i
i       dear_alice,_dear_bob      j=5
i+5     dear_alice,_dear_bob      j=0
```

NaiveStringMatch(T, P):
input: strings T, P
output: 1st position of P in T
or -1 if P does not occur in T

```
for i from 0 to |T| − |P|
    for j from 0 to |P| − 1
        if P_j ≠ T_{i+j} then break
    if j = |P| then return i
return −1
```

we break on the pattern character in j[th] position
-> we were able to match the j characters before that

the pattern character in 0th position does not appear again **at or before** the jth position -> move i by j positions., i.e., $i \leftarrow i + j$, and restart j

Shifting (case 3: the tricky one)

```
        i
i       dear_alice,_dear_bob      j=17
i+17    dear_alice,
i+18     dear_alice,
```

NaiveStringMatch(T, P):
input: strings T, P
output: 1st position of P in T
or -1 if P does not occur in T

```
for i from 0 to |T| − |P|
    for j from 0 to |P| − 1
        if P_j ≠ T_{i+j} then break
    if j = |P| then return i
return −1
```

Shifting (case 3: the tricky one)

```
        i
i       dear_alice,_dear_bob      j=17
         ddddddddddd
i+12            dear_alice,_dear_bob  j=5
```

NaiveStringMatch(T, P):
input: strings T, P
output: 1st position of P in T
or -1 if P does not occur in T

```
for i from 0 to |T| − |P|
    for j from 0 to |P| − 1
        if P_j ≠ T_{i+j} then break
    if j = |P| then return i
return −1
```

Shifting (case 3: the small print—part a)

```
        i
i       dear_alice,_dear_bob      j=17
i+12            dear_alice,_dear_bob  j=5
```

FastKMPStringMatch(T, P):
input: strings T, P
output: 1st position of P in T
or -1 if P does not occur in T

?

we break on the pattern character in j[th] position
-> we were able to match the j characters before that

if a **suffix** of the *partial pattern that we matched* is a **prefix** of the pattern of length ℓ and the $(\ell + 1)^{\text{th}}$ character is **not** the same as the $(j + 1)^{\text{th}}$ character
-> move i by $j - \ell$ positions., i.e., $i \leftarrow i + j - \ell$, and set $j \rightarrow \ell$
if there is more than one suffix, consider the largest one

Shifting (case 3: the small print)

```
        i
i       dear_alice,_dear_bob      j=14
i+14            dear_alice,_dear_bob  j=0
```

FastKMPStringMatch(T, P):
input: strings T, P
output: 1st position of P in T
or -1 if P does not occur in T

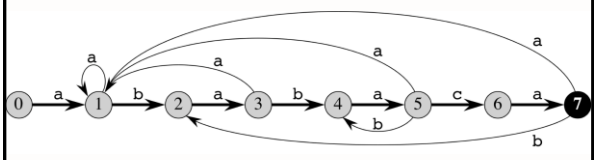
?

we break on the pattern character in j[th] position
-> we were able to match the j characters before that

if a **suffix** of the *partial pattern that we matched* is a **prefix** of the pattern of length ℓ and the $(\ell + 1)^{\text{th}}$ character **is the same** as the $(j + 1)^{\text{th}}$ character
-> move i by j positions., i.e., $i \leftarrow i + j$, and set $j \rightarrow 0$

Excursion: KMP automaton ("nano")

Excursion: "traditional" Finite State Automaton ("ababaca")



i	–	1	2	3	4	5	6	7	8	9	10	11
$T[i]$	–	a	b	a	b	a	b	a	c	a	b	a
state $\phi(T_i)$	0	1	2	3	4	5	4	5	6	7	2	3

PS links to starting state not depicted

Magic shifts

```
   0    5     12   17     24   29     35   40     47   52     59   64
   |    |     |    |      |    |      |    |      |    |      |    |
P: dear_alice,_dear_bob
T: dear_carol,_dear_alice,_dear_dave,_dear_alice,_dear_alice,_dear_bob
   dear_alice,_dear_bob
      dddddd
              dear_alice,_dear_bob
                      dear_alice,_dear_bob
                          de
                            ddddd
                                  dear_alice,_dear_bob
                                          dear_alice,_dear_bob
```

Doing better in the worst cases!

```
        i->
        0 1 2 3 4 5 6 7 8 9
        A A A A A A A A A A
j  0 A  =
|  1 A     =
V  2 A        =
   3 B           X→X→X→X→X→X→X→X
                                  -1
```

KMPStringMatch(T, P):
input: strings T, P
output: 1st position of P in T
or -1 if P does not occur in T

begin at 1st pattern & text position

while *pattern not fully matched*

if *pattern character differs from text character* **then**

shift pattern super magically

move to next character (in pattern & text)

Doing better in many cases!

```
        i->
        0 1 2 3 4 5 6 7 8 9
        A B R C A B A B R C
j  0 A  =           =
|  1 B     =     =     =     =
V  2 R        =           X
   3 A           X           X
                                  -1
```

KMPStringMatch(T, P):
input: strings T, P
output: 1st position of P in T
or -1 if P does not occur in T

begin at 1st pattern & text position

while *pattern not fully matched*

if *pattern character differs from text character* **then**

shift pattern super magically

move to next character (in pattern & text)

Doing better in many cases!

```
        i->
        0 1 2 3 4 5 6 7 8 9
        A B R C A B A B R C
j  0 A  =           =
|  1 B     =     =     =     =
V  2 R        =           X
   3 A           X           X
                                  -1

eg  abc abd abc abe abc abd abc abf
```

KMPStringMatch(T, P):
input: strings T, P
output: 1st position of P in T
or -1 if P does not occur in T

begin at 1st pattern & text position

while *pattern not fully matched*

while *pattern character differs from text character*

shift pattern magically

move to next character (in pattern & text)

Doing better in many cases!

```
        i->
        0 1 2 3 4 5 6 7 8 9
        A B R C A B A B R C
j  0 A  =           =
|  1 B     =     =     =     =
V  2 R        =           X
   3 A           X           X
                                  -1
   k = i + j ->
```

KMPStringMatch(T, P):
input: strings T, P
output: 1st position of P in T
or -1 if P does not occur in T

$j \leftarrow 0, k \leftarrow 0$
while $k < |T|$ **and** $j < |P|$
  while $j \geq 0$ **and** $T_k \neq P_j$
    $j \leftarrow$ magic_shift(j)
  $j \leftarrow j + 1,\ k \leftarrow k + 1$

if $j = |P|$ **then**
  return $k - |P|$
else
  return -1

Magic shifts

```
P: a t c a t c a c a t g
      a t c a t c a c a t g

P: a t c a t c a c a t g
        a t c a t c a c a t g

T: tatctatcatcaatcatcatcacatg
   a
    atcatcacatg
      atcatcacatg
        atcatcacatg
          atcatcacatg
            atcatcacatg
```

Finding magic shifts: it's (almost) string matching again

	0	1	2	3	4	5	6	7	8	9	10
	a	t	c	a	t	c	a	c	a	t	g
0 a	=	X	X	=	X	X	=	X	=	X	X
1 t	X	=	X	X	=	X	X	X	=	X	=
2 c	X	X	=	X	X	=	X	=	X	X	X
3 a	=	X	X	=	X	X	=	X	=	X	X
4 t	X	=	X	X	=	X	X	X	=	X	X
5 c	X	X	=	X	X	=	X	=	X	X	X
6 a	=	X	X	=	X	X	=	X	=	X	X
7 c	X	X	X	X	X	X	X	=	X	X	X
8 a	=	X	X	=	X	X	=	X	=	X	X
9 t	X	=	X	X	=	X	X	X	X	=	X
10 g	X	X	X	X	X	X	X	X	X	X	=

j	0	1	2	3	4	5	6	7	8	9	10
P_j	a	t	c	a	t	c	a	c	a	t	g
S_j	*	0	0	*	0	0	*	4	*	0	2
F_j	*	0	0	0	1	2	3	4	0	1	2

number of characters
before the current character
(length of the suffix of the partial match)
in common with but not identical to
the **beginning of the pattern** (prefix)

Finding magic shifts: it's (almost) string matching again **QUIZ**

https://goo.gl/P9JiXY

	-1	0	1	2	3	4	5	6	7	8	9
	*	t	r	i	t	r	a	t	r	u	m
0 t											
1 r											
2 i											
3 t											
4 r											
5 a											
6 t											
7 r											
8 u											
9 m											

j	0	1	2	3	4	5	6	7	8	9	10
P_j	a	t	c	a	t	c	a	c	a	t	g
S_j	*	0	0	*	0	0	*	4	*	0	2
F_j	*	0	0	0	1	2	3	4	0	1	2

For $j > 0$: $S_j = \begin{cases} F_j: & P_j \neq P_{F_j} \\ S_{F_j}: & P_j = P_{F_j} \end{cases}$

number of characters
before the current character
(length of the suffix of the partial match)
in common with but not identical to
the **beginning of the pattern** (prefix)

$F_j = \max\{t \in \mathbb{N}, t < j \mid (P_0, \dots P_{t-1}) = (P_{j-t}, \dots P_{j-1})\} \cup \{-1\}$
$S_j = \max\{t \in \mathbb{N}, t < j \mid (P_0, \dots P_{t-1}) = (P_{j-t}, \dots P_{j-1}) \wedge P_t \neq P_j\} \cup \{-1\}$

Finding magic shifts: it's (almost) string matching again

	0	1	2	3	4	5	6	7	8	9	10
	a	t	c	a	t	c	a	c	a	t	g
0 a	=	X	X	=	X	X	=	X	=	X	X
1 t	X	=	X	X	=	X	X	X	=	X	=
2 c	X	X	=	X	X	=	X	=	X	X	X
3 a	=	X	X	=	X	X	=	X	=	X	X
4 t	X	=	X	X	=	X	X	X	=	X	X

j	0	1	2	3	4	5	6	7	8	9	10
P_j	a	t	c	a	t	c	a	c	a	t	g
S_j	*	0	0	*	0	0	*	4	*	0	2
F_j	*	0	0	0	1	2	3	4	0	1	2

For $j > 0$: $S_j = \begin{cases} F_j: & P_j \neq P_{F_j} \\ S_{F_j}: & P_j = P_{F_j} \end{cases}$

$F_j = t \quad \Rightarrow \quad (P_0, \dots P_{t-1}) = (P_{j-t}, \dots P_{j-1})$

number of characters
before the current character
(length of the suffix of the partial match)
in common with but not identical to
the **beginning of the pattern** (prefix)

$F_j = \max\{t \in \mathbb{N}, t < j \mid (P_0, \dots P_{t-1}) = (P_{j-t}, \dots P_{j-1})\} \cup \{-1\}$
$S_j = \max\{t \in \mathbb{N}, t < j \mid (P_0, \dots P_{t-1}) = (P_{j-t}, \dots P_{j-1}) \wedge P_t \neq P_j\} \cup \{-1\}$

Finding magic shifts: it's (almost) string matching again

	0	1	2	3	4	5	6	7	8	9	10
	a	t	c	a	t	c	a	c	a	t	g
0 a	=	X	X	=	X	X	=	X	=	X	X
1 t	X	=	X	X	=	X	X	X	=	X	=
2 c	X	X	=	X	X	=	X	=	X	X	X
3 a	=	X	X	=	X	X	=	X	=	X	X
4 t	X	=	X	X	=	X	X	X	=	X	X

j	0	1	2	3	4	5	6	7	8	9	10
P_j	a	t	c	a	t	c	a	c	a	t	g
S_j	*	0	0	*	0	0	*	4	*	0	2
F_j	*	0	0	0	1	2	3	4	0	1	2

$F_0 \leftarrow -1, S_0 = -1$
for j **from** 0 **to** $|P| - 1$
  $t \leftarrow F_j$
  while $t \geq 0$ **and** $P_j \neq P_t$ **do** $t \leftarrow S_t$
  $F_{j+1} \leftarrow t + 1$

For $j > 0$: $S_j = \begin{cases} F_j: & P_j \neq P_{F_j} \\ S_{F_j}: & P_j = P_{F_j} \end{cases}$

$j \leftarrow 0; t \leftarrow -1; (< 0); F_1 \leftarrow 0;$ $j \leftarrow 1; t \leftarrow 0; (\geq, \neq)\ t \leftarrow -1; F_2 \leftarrow 0;$ $j \leftarrow 2; t \leftarrow 0\ ; (\geq, \neq)\ t \leftarrow -1; F_3 \leftarrow 0$
$j \leftarrow 3; t \leftarrow 0; (=); F_4 \leftarrow 1;$ $j \leftarrow 4; t \leftarrow 1; (=); F_5 \leftarrow 2;$ $j \leftarrow 5; t \leftarrow 2; (=); F_6 \leftarrow 3;$ $j \leftarrow 6; t \leftarrow 3; (=); F_7 \leftarrow 4;$
$j \leftarrow 7; t \leftarrow 4; (\geq, \neq)\ t \leftarrow 0; (\geq, \neq)\ t \leftarrow -1; F_8 \leftarrow 0;$ $j \leftarrow 8; t \leftarrow 0; (=); F_9 \leftarrow 1;$ $j \leftarrow 9; t \leftarrow 1; (=); F_{10} \leftarrow 2;$

Complexity

every operation increases $2j - t$.
This can only happen $2|P|$ times.

$j \leftarrow 0, t \leftarrow -1, S_0 = -1$
while $j < |P|$
  while $t \geq 0$ **and** $P_j \neq P_t$ **do** $t \leftarrow S_t$
  $t \leftarrow t + 1, j \leftarrow j + 1$
  if $P_j = P_t$ **then** $S_j = S_t$ **else** $S_j = t$

every operation increases $2k - j$.
This can only happen $2|T|$ times.

$j \leftarrow 0, k \leftarrow 0$
while $k < |T|$ **and** $j < |P|$
  while $j \geq 0$ **and** $T_k \neq P_j$ **do** $j \leftarrow S_j$
  $j \leftarrow j + 1,\ k \leftarrow k + 1$

invariant
$\forall\, 0 \leq u < j: T_{k-j+u} = P_u$
$\forall\, 0 \leq v < k - j\ \exists 0 \leq u < j: T_{k-j+u} \neq P_u$

if $j = |P|$ **then return** $k - |P|$
  else return -1