String Matching Algorithms

Antonio Carzaniga

Faculty of Informatics
Università della Svizzera italiana

December 22, 2011

Outline

- Problem definition
- Naïve algorithm
- Knuth-Morris-Pratt algorithm
- Boyer-Moore algorithm

Problem

Problem

- Given the text
  Nel mezzo del cammin di nostra vita
  mi ritrovai per una selva oscura
  che la dritta via era smarrita…

  Find the string “trova”

- Given the text
  Nel mezzo del cammin di nostra vita
  mi ritrovai per una selva oscura
  che la dritta via era smarrita…

  Find the string “trova”

- A more challenging example: How many times does the string “110011” appear in the following text

  001111010101101001100011010111101101 0111
  011011100100101010101111011110110000101
  1011000010111111011110011000011111000100
  100101001011101110101101111010100110010 1
  001011100100001111111001001101110101101 0
  0110011011101001010010101000010100111110

Problem

- Given the text
  Nel mezzo del cammin di nostra vita
  mi ritrovai per una selva oscura
  che la dritta via era smarrita…

  Find the string "trova"

- A more challenging example: How many times does the string "110011" appear in the following text

  0011110101011010011000110101111011010111
  0110111001001010101011111011110110000101
  10110000010111111011110011000011111000100
  1001010010111011101011011110101001100101
  0010111001000011111110010011011101011010
  011001101110100101001010100001010011110

String Matching: Definitions

- Given a ***text*** T
  - $T \in \Sigma^*$: finite alphabet Σ
  - $|T| = n$: the length of T is n

String Matching: Definitions

- Given a ***text*** T
  - $T \in \Sigma^*$: finite alphabet Σ
  - $|T| = n$: the length of T is n

- Given a ***pattern*** P
  - $P \in \Sigma^*$: same finite alphabet Σ
  - $|P| = m$: the length of P is m

String Matching: Definitions

- Given a ***text*** T
  - $T \in \Sigma^*$: finite alphabet Σ
  - $|T| = n$: the length of T is n
- Given a ***pattern*** P
  - $P \in \Sigma^*$: same finite alphabet Σ
  - $|P| = m$: the length of P is m
- Both T and P can be modeled as arrays
  - $T[1 \ldots n]$ and $P[1 \ldots m]$

String Matching: Definitions

- Given a ***text*** T
  - $T \in \Sigma^*$: finite alphabet Σ
  - $|T| = n$: the length of T is n

- Given a ***pattern*** P
  - $P \in \Sigma^*$: same finite alphabet Σ
  - $|P| = m$: the length of P is m

- Both T and P can be modeled as arrays
  - $T[1 \dots n]$ and $P[1 \dots m]$

- Pattern P occurs with ***shift*** s in T iff
  - $0 \leq s \leq n - m$
  - $T[s + i] = P[i]$ for all positions $1 \leq i \leq m$

Example

- Problem: find all s such that
  - $0 \leq s \leq n - m$
  - $T[s + i] = P[i]$ for $1 \leq i \leq m$

Example

- Problem: find all s such that
  - $0 \leq s \leq n - m$
  - $T[s + i] = P[i]$ for $1 \leq i \leq m$

$n = 14$

T	a	b	c	a	a	b	a	a	b	a	b	a	c	a

- Problem: find all s such that
  - $0 \le s \le n - m$
  - $T[s+i] = P[i]$ for $1 \le i \le m$

$n = 14$

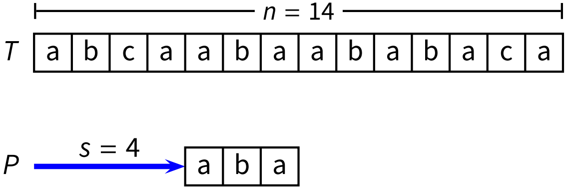
T														
	a	b	c	a	a	b	a	a	b	a	b	a	c	a

$m = 3$

P			
	a	b	a

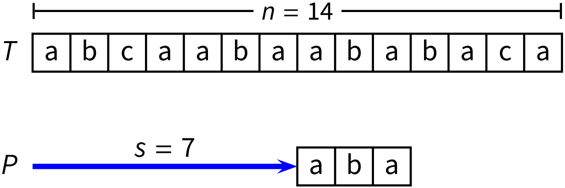
- Result

- Problem: find all s such that
  - $0 \leq s \leq n - m$
  - $T[s+i] = P[i]$ for $1 \leq i \leq m$



- Result
  $s = 4$

- Problem: find all s such that
  - $0 \leq s \leq n - m$
  - $T[s+i] = P[i]$ for $1 \leq i \leq m$

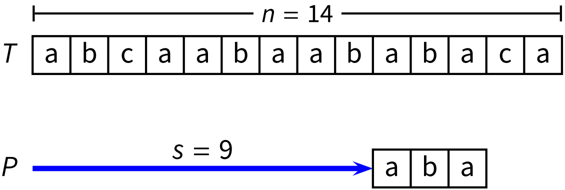


- Result

  $s = 4$

  $s = 7$

- Problem: find all s such that
  - $0 \le s \le n - m$
  - $T[s+i] = P[i]$ for $1 \le i \le m$



- Result

$s = 4$
$s = 7$
$s = 9$

Naïve Algorithm

Naïve Algorithm

- For each position s in $0 \ldots n - m$, see if $T[s+i] = P[i]$ for all $1 \leq i \leq m$

Naïve Algorithm

- For each position s in $0 \ldots n - m$, see if $T[s + i] = P[i]$ for all $1 \leq i \leq m$

```
NAIVE-STRING-MATCHING(T, P)
n = length(T)
m = length(P)
for s = 0 to n − m
    if SUBSTRING-AT(T, P, s)
        OUTPUT(s)
```

Naïve Algorithm

- For each position s in $0 \dots n - m$, see if $T[s+i] = P[i]$ for all $1 \leq i \leq m$

```
NAIVE-STRING-MATCHING(T, P)
  n = length(T)
  m = length(P)
  for s = 0 to n − m
      if SUBSTRING-AT(T, P, s)
          OUTPUT(s)
```

```
SUBSTRING-AT(T, P, s)
  for i = 1 to length(P)
      if T[s + i] ≠ P[i]
          return FALSE
  return TRUE
```

Complexity of the Naïve Algorithm

Complexity of the Naïve Algorithm

- Complexity of **Naive-String-Match** is $O((n-m+1)m)$

Complexity of the Naïve Algorithm

- Complexity of **NAIVE-STRING-MATCH** is $O((n - m + 1)m)$

- Worst case example

$$T = \mathrm{a}^n, \quad P = \mathrm{a}^m$$

i.e.,

$$T = \overbrace{\mathrm{aa}\cdots\mathrm{a}}^{n}, \quad P = \overbrace{\mathrm{aa}\cdots\mathrm{a}}^{m}$$

Complexity of the Naïve Algorithm

- Complexity of **Naive-String-Match** is $O((n - m + 1)m)$

- Worst case example

$$T = \mathrm{a}^n, \quad P = \mathrm{a}^m$$

i.e.,

$$T = \overbrace{\mathrm{aa}\cdots\mathrm{a}}^{n}, \quad P = \overbrace{\mathrm{aa}\cdots\mathrm{a}}^{m}$$

So, $(n - m + 1)m$ is a tight bound, so the (worst-case) complexity of **Naive-String-Match** is

$$\boxed{\Theta((n - m + 1)m)}$$

Improvement Strategy

Improvement Strategy

- Observation

T	a	b	c	a	a	b	a	a	b	a	b	a	c	a

P	a	b	a

- Observation

T	a	b	c	a	a	b	a	a	b	a	b	a	c	a

$=$

P	a	b	a

- Observation

T	a	b	c	a	a	b	a	a	b	a	b	a	c	a
	=	=												
P	a	b	a											

- Observation

T	a	b	c	a	a	b	a	a	b	a	b	a	c	a
	$=$	$=$	$\neq$											
P	a	b	a											

Improvement Strategy

- Observation

T	a	b	c	a	a	b	a	a	b	a	b	a	c	a
	$=$	$=$	$\neq$											
P	a	b	a											

- What now?

Improvement Strategy

- Observation

T	a	b	c	a	a	b	a	a	b	a	b	a	c	a
	$=$	$=$	$\neq$											
P	a	b	a											

- What now?
  - the naïve algorithm ***goes back to the second position in*** T and ***starts from the beginning of*** P

- Observation

T	a	b	c	a	a	b	a	a	b	a	b	a	c	a
	$=$	$=$	$\neq$											
P	a	b	a											

- What now?
  - the naïve algorithm ***goes back to the second position in*** T and ***starts from the beginning of*** P
  - can’t we simply move along through T?

Improvement Strategy

- Observation

T	a	b	c	a	a	b	a	a	b	a	b	a	c	a
	$=$	$=$	$\neq$											
P	a	b	a											

- What now?
  - the naïve algorithm ***goes back to the second position in*** T and ***starts from the beginning of*** P
  - can’t we simply move along through T?
  - why?

Improvement Strategy (2)

Improvement Strategy (2)

- Here’s a wrong but insightful strategy

Improvement Strategy (2)

- Here's a wrong but insightful strategy

```
WRONG-STRING-MATCHING(T, P)
n = length(T)
m = length(P)
q = 0            // number of characters matched in P
s = 1
while s ≤ n
    s = s + 1
    if T[s] == P[q + 1]
        q = q + 1
        if q == m
            OUTPUT(s − m)
            q = 0
    else q = 0
```

Improvement Strategy (3)

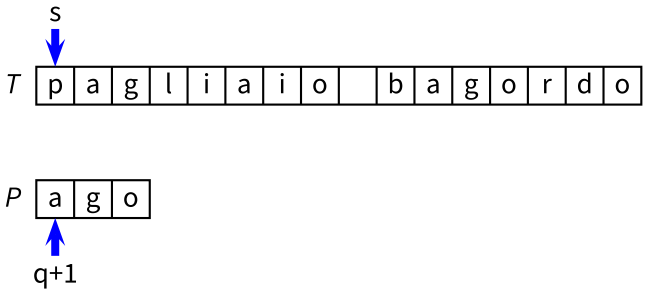
- Example run of **Wrong-String-Matching**

- Example run of **Wrong-String-Matching**

T	p	a	g	l	i	a	i	o		b	a	g	o	r	d	o

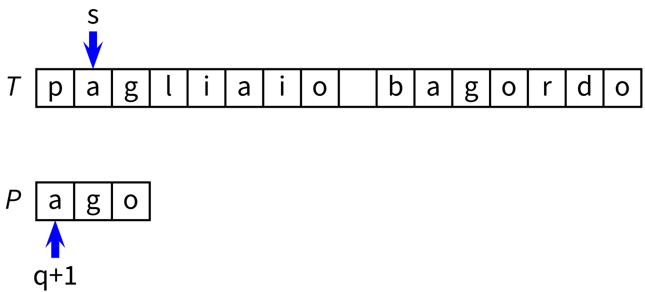
P	a	g	o

- Example run of **WRONG-STRING-MATCHING**

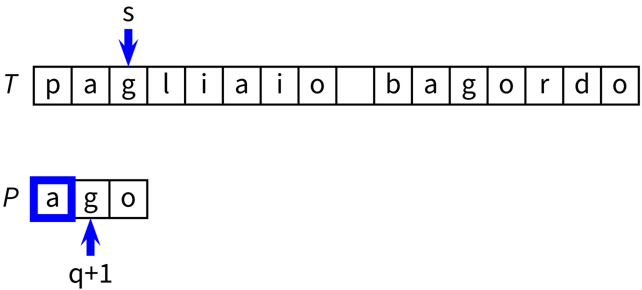


Improvement Strategy (3)

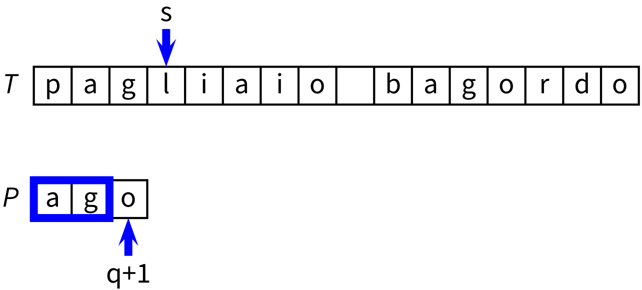
- Example run of **Wrong-String-Matching**



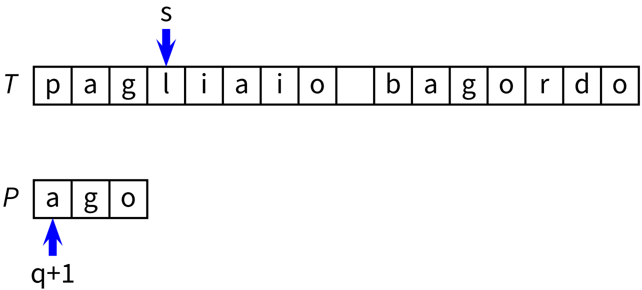
- Example run of **WRONG-STRING-MATCHING**



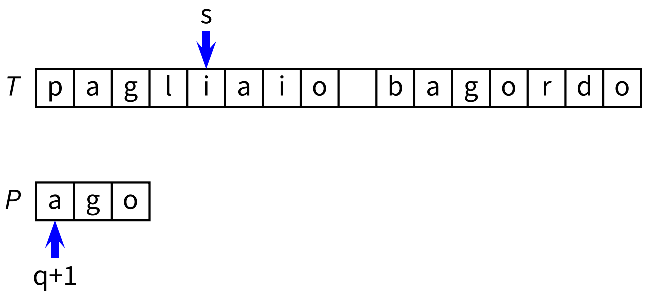
- Example run of **WRONG-STRING-MATCHING**



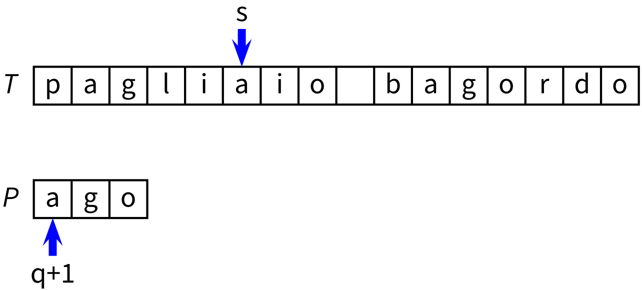
- Example run of **WRONG-STRING-MATCHING**



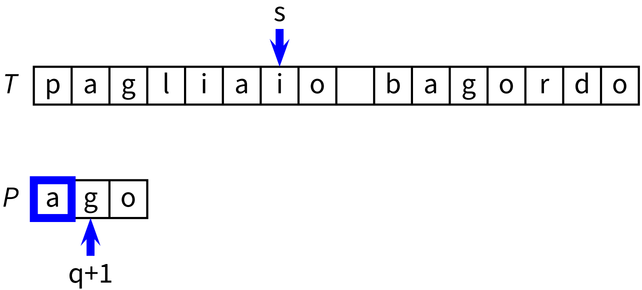
- Example run of **WRONG-STRING-MATCHING**



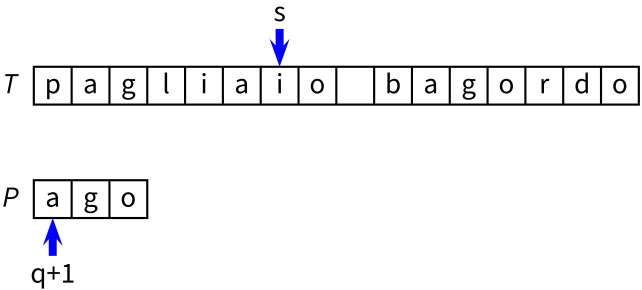
- Example run of **WRONG-STRING-MATCHING**



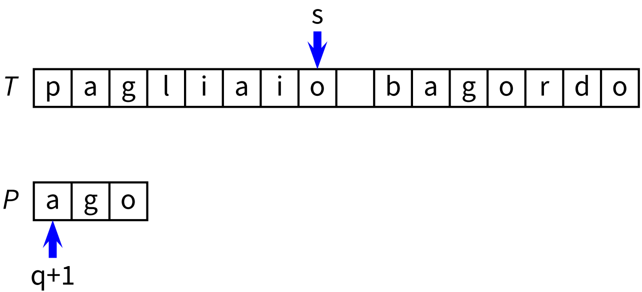
- Example run of **WRONG-STRING-MATCHING**



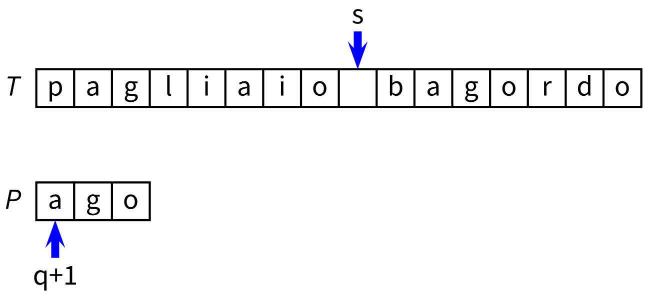
- Example run of **WRONG-STRING-MATCHING**



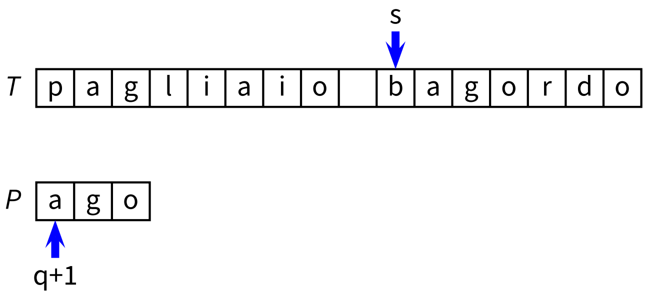
- Example run of **WRONG-STRING-MATCHING**



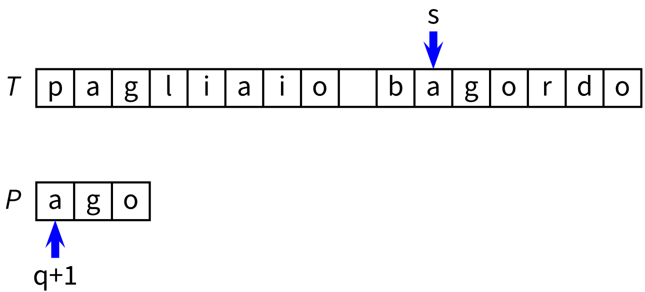
- Example run of **WRONG-STRING-MATCHING**



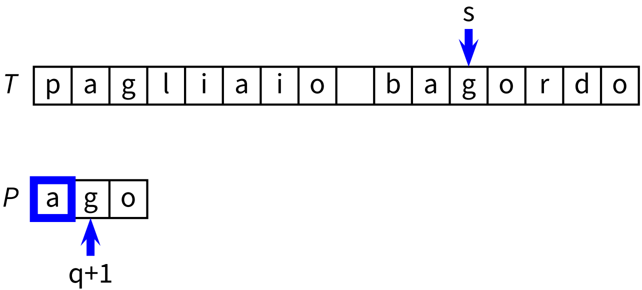
- Example run of **WRONG-STRING-MATCHING**



- Example run of **WRONG-STRING-MATCHING**

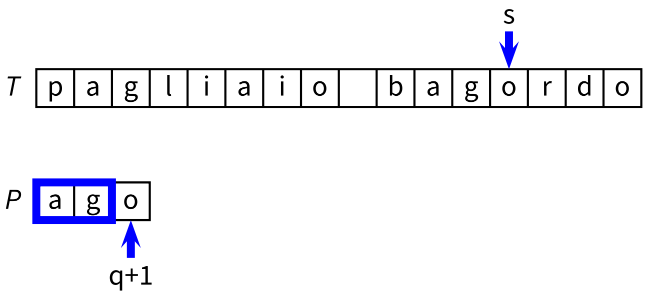


- Example run of **Wrong-String-Matching**



Improvement Strategy (3)

- Example run of **Wrong-String-Matching**



Improvement Strategy (3)

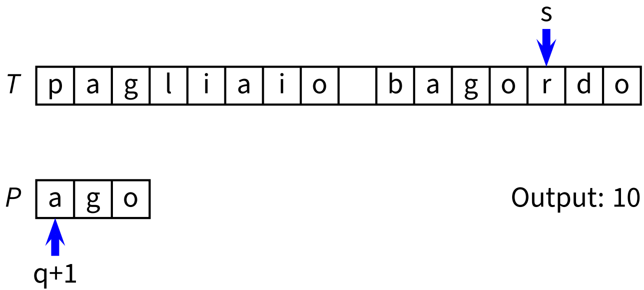
- Example run of **WRONG-STRING-MATCHING**

s

T	p	a	g	l	i	a	i	o		b	a	g	o	r	d	o

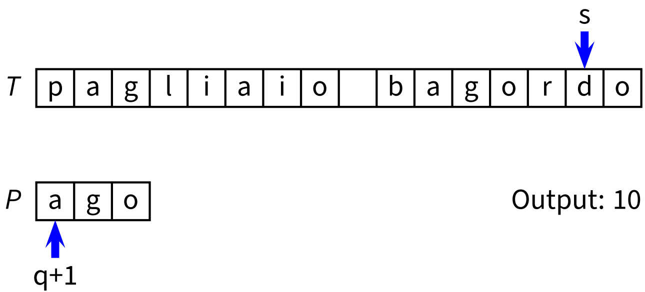
P	a	g	o

- Example run of **WRONG-STRING-MATCHING**

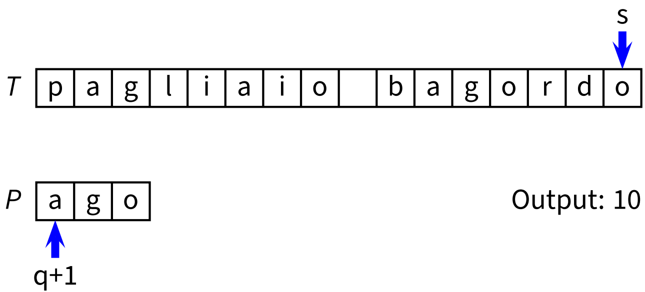


Output: 10

- Example run of **WRONG-STRING-MATCHING**



- Example run of **WRONG-STRING-MATCHING**



Improvement Strategy (3)

- Example run of **Wrong-String-Matching**

T																
	p	a	g	l	i	a	i	o		b	a	g	o	r	d	o

P			
	a	g	o

Output: 10

- Done. Perfect!

Improvement Strategy (3)

- Example run of **Wrong-String-Matching**

T	p	a	g	l	i	a	i	o		b	a	g	o	r	d	o

P	a	g	o

Output: 10

- Done. Perfect!
- Complexity: $\Theta(n)$

Improvement Strategy (4)

- What is wrong with **Wrong-String-Matching**?

Improvement Strategy (4)

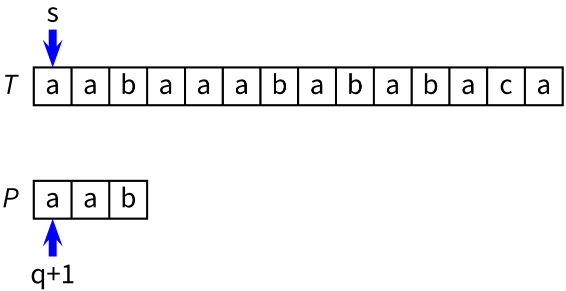
- What is wrong with **Wrong-String-Matching**?

T	a	a	b	a	a	a	b	a	b	a	b	a	c	a

P	a	a	b

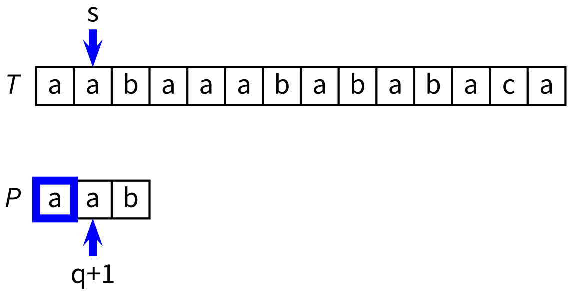
Improvement Strategy (4)

- What is wrong with **Wrong-String-Matching**?



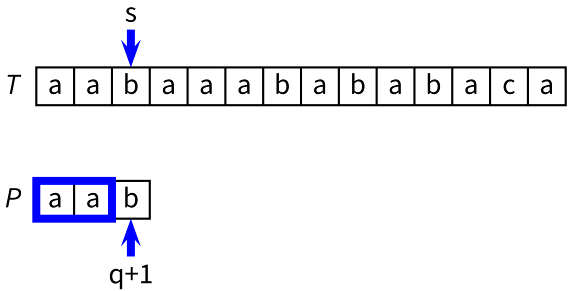
Improvement Strategy (4)

- What is wrong with **WRONG-STRING-MATCHING**?



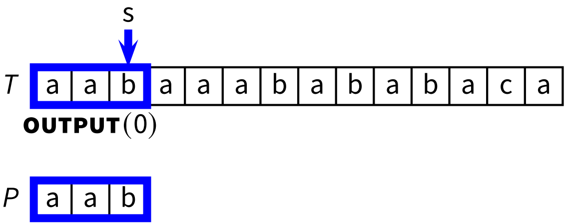
Improvement Strategy (4)

- What is wrong with **WRONG-STRING-MATCHING**?

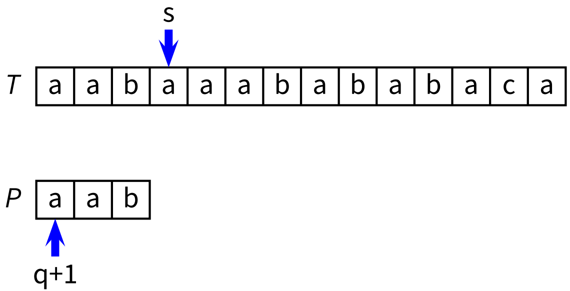


Improvement Strategy (4)

- What is wrong with **Wrong-String-Matching**?

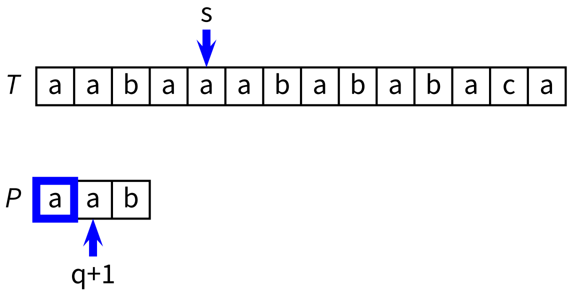


- What is wrong with **WRONG-STRING-MATCHING**?

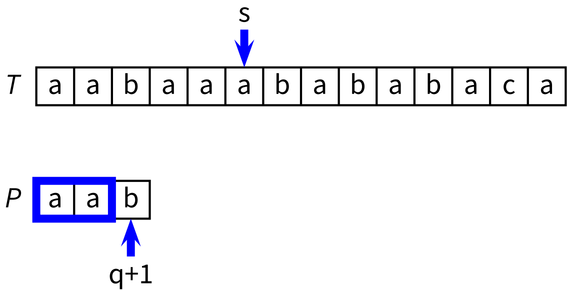


Improvement Strategy (4)

- What is wrong with **Wrong-String-Matching**?

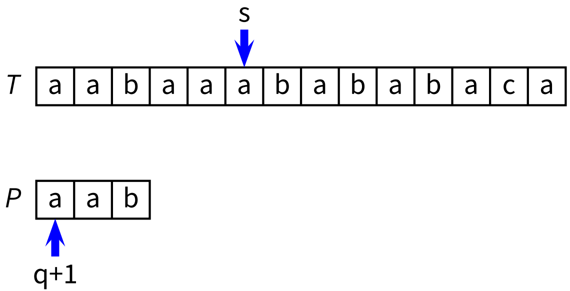


- What is wrong with **WRONG-STRING-MATCHING**?

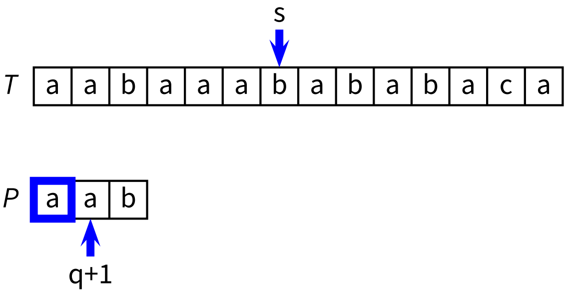


Improvement Strategy (4)

- What is wrong with **Wrong-String-Matching**?

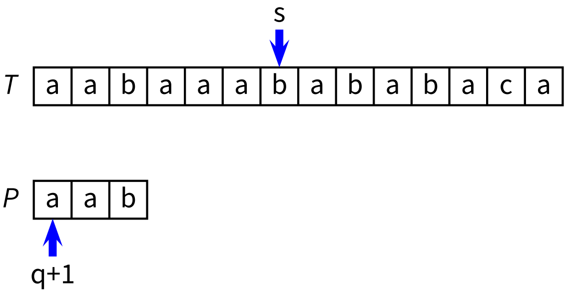


- What is wrong with **WRONG-STRING-MATCHING**?

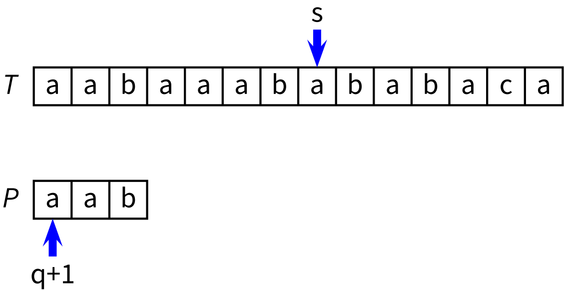


Improvement Strategy (4)

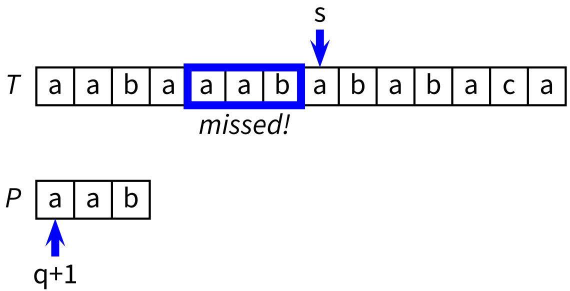
- What is wrong with **WRONG-STRING-MATCHING**?



- What is wrong with **Wrong-String-Matching**?

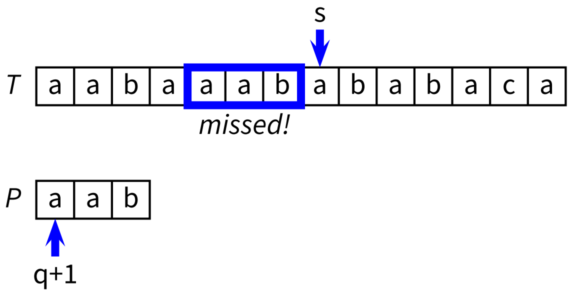


- What is wrong with **WRONG-STRING-MATCHING**?



Improvement Strategy (4)

- What is wrong with **WRONG-STRING-MATCHING**?



- So **WRONG-STRING-MATCHING** doesn't work, but it tells us something useful

Improvement Strategy (5)

- Where did **Wrong-String-Matching** go wrong?

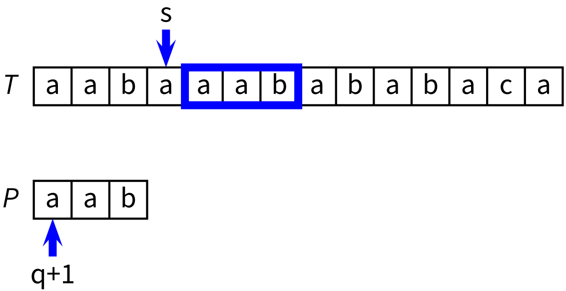
Improvement Strategy (5)

- Where did **Wrong-String-Matching** go wrong?

T	a	a	b	a	a	a	b	a	b	a	b	a	c	a

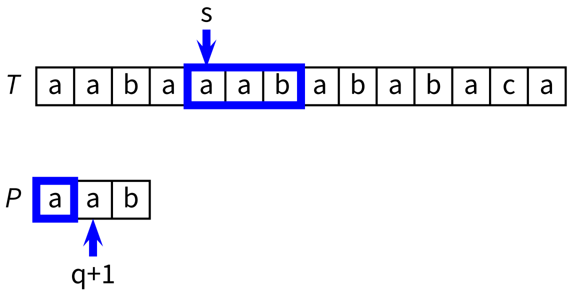
P	a	a	b

- Where did **WRONG-STRING-MATCHING** go wrong?



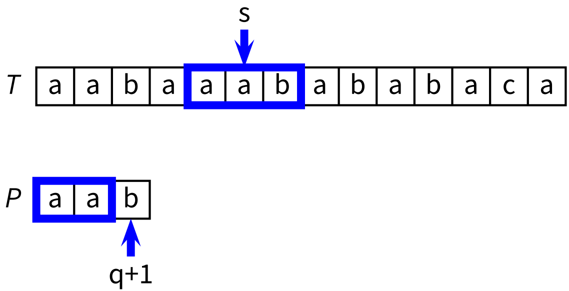
Improvement Strategy (5)

- Where did **WRONG-STRING-MATCHING** go wrong?



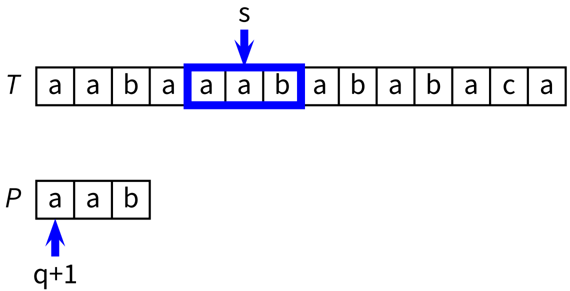
Improvement Strategy (5)

- Where did **Wrong-String-Matching** go wrong?

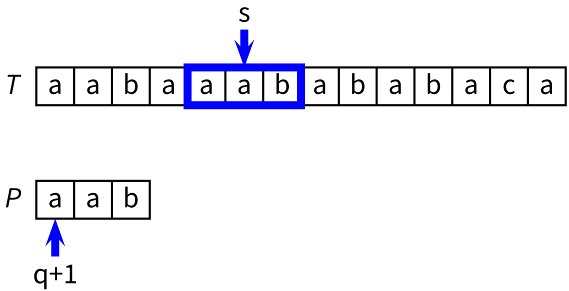


Improvement Strategy (5)

- Where did **WRONG-STRING-MATCHING** go wrong?



- Where did **Wrong-String-Matching** go wrong?



- Wrong: by going all the way back to $q = 0$ we throw away a good prefix of P that we already matched

Improvement Strategy (6)

- Another example

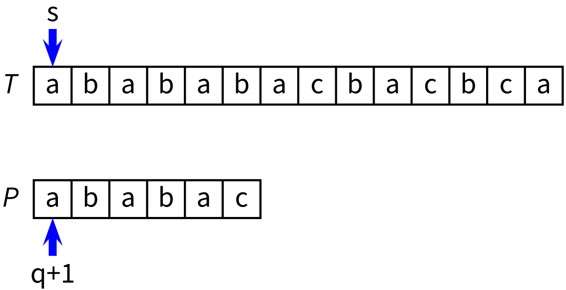
Improvement Strategy (6)

- Another example

T	a	b	a	b	a	b	a	c	b	a	c	b	c	a

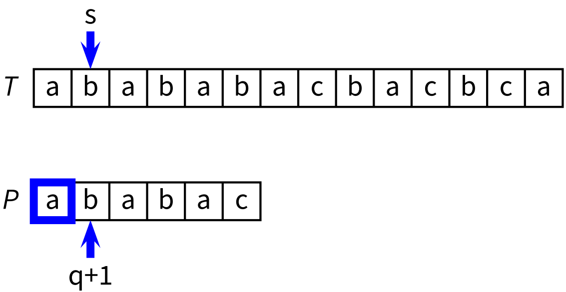
P	a	b	a	b	a	c

- Another example



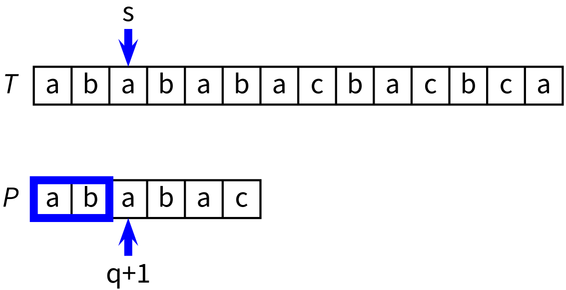
Improvement Strategy (6)

- Another example

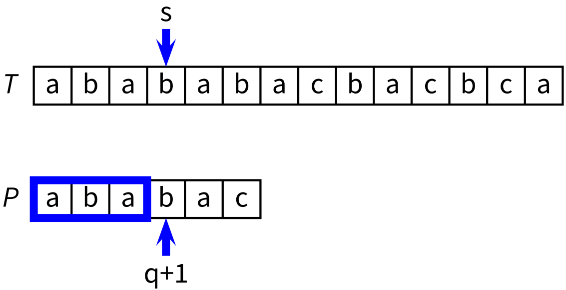


Improvement Strategy (6)

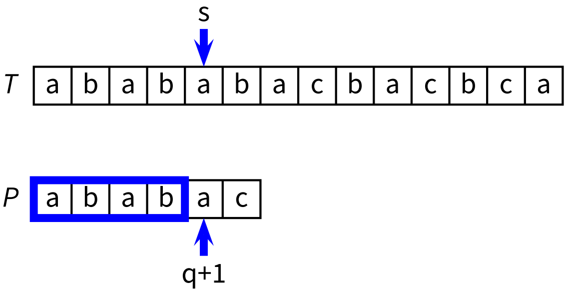
- Another example



- Another example

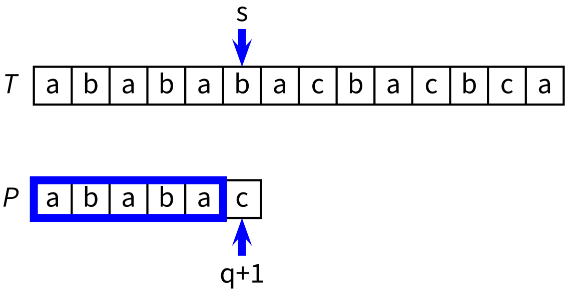


- Another example



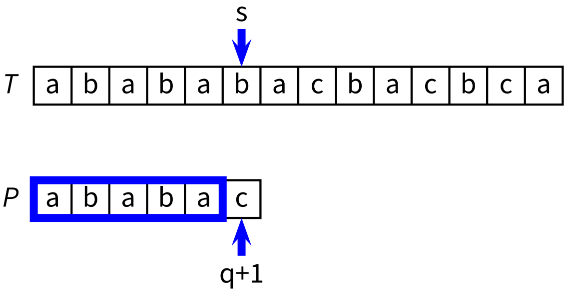
Improvement Strategy (6)

- Another example



Improvement Strategy (6)

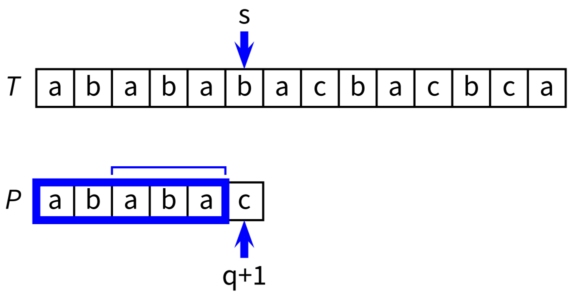
- Another example



- We have matched “ababa”

Improvement Strategy (6)

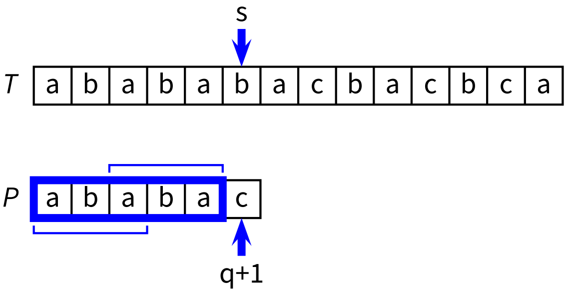
- Another example



- We have matched “ababa”
  - ***suffix*** “aba” can be ***reused as a prefix***

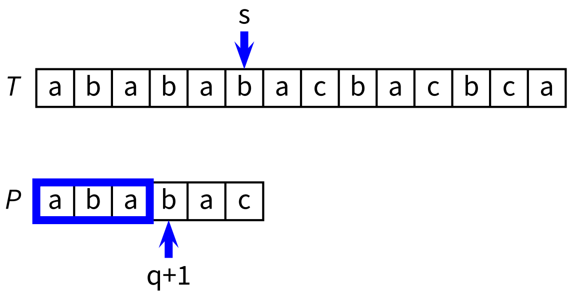
Improvement Strategy (6)

- Another example



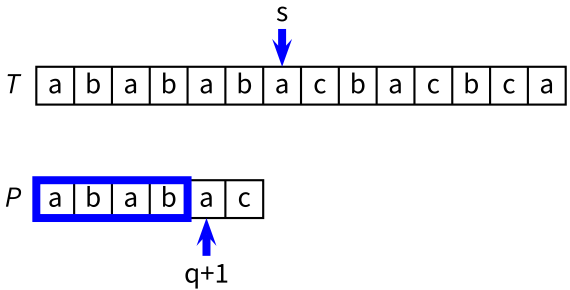
- We have matched “ababa”
  - ***suffix*** “aba” can be ***reused as a prefix***

- Another example



- We have matched “ababa”
  - ***suffix*** “aba” can be ***reused as a prefix***

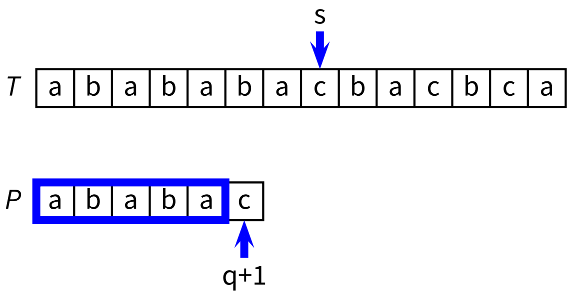
- Another example



- We have matched “ababa”
  - ***suffix*** “aba” can be ***reused as a prefix***

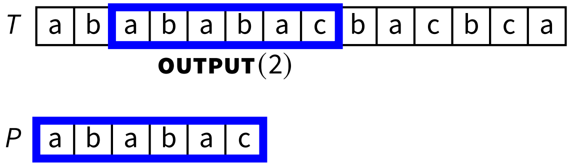
Improvement Strategy (6)

- Another example



- We have matched “ababa”
  - ***suffix*** “aba” can be ***reused as a prefix***

- Another example



- We have matched “ababa”
  - ***suffix*** “aba” can be ***reused as a prefix***

New Strategy

- $P[1 \ldots q]$ is the prefix of P matched so far

New Strategy

- $P[1 \dots q]$ is the prefix of P matched so far
- Find the longest prefix of P that is also a suffix of $P[2 \dots q]$
  - i.e., find $0 \leq \pi < q$ such that $P[q - \pi + 1 \dots q] = P[1 \dots \pi]$
    - $\pi = 0$ means that such a prefix does not exist

New Strategy

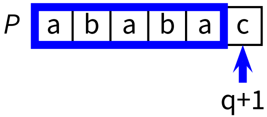
- $P[1 \ldots q]$ is the prefix of P matched so far

- Find the longest prefix of P that is also a suffix of $P[2 \ldots q]$
  - i.e., find $0 \leq \pi < q$ such that $P[q - \pi + 1 \ldots q] = P[1 \ldots \pi]$
    - $\pi = 0$ means that such a prefix does not exist

P	a	b	a	b	a	c

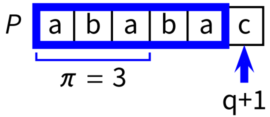
New Strategy

- $P[1 \ldots q]$ is the prefix of P matched so far
- Find the longest prefix of P that is also a suffix of $P[2 \ldots q]$
  - i.e., find $0 \leq \pi < q$ such that $P[q - \pi + 1 \ldots q] = P[1 \ldots \pi]$
    - $\pi = 0$ means that such a prefix does not exist



New Strategy

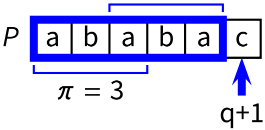
- $P[1 \dots q]$ is the prefix of P matched so far
- Find the longest prefix of P that is also a suffix of $P[2 \dots q]$
  - i.e., find $0 \leq \pi < q$ such that $P[q - \pi + 1 \dots q] = P[1 \dots \pi]$
    - $\pi = 0$ means that such a prefix does not exist



New Strategy

- $P[1 \dots q]$ is the prefix of P matched so far

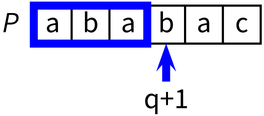
- Find the longest prefix of P that is also a suffix of $P[2 \dots q]$
  - i.e., find $0 \leq \pi < q$ such that $P[q - \pi + 1 \dots q] = P[1 \dots \pi]$
    - $\pi = 0$ means that such a prefix does not exist



New Strategy

- $P[1 \dots q]$ is the prefix of P matched so far

- Find the longest prefix of P that is also a suffix of $P[2 \dots q]$
  - i.e., find $0 \leq \pi < q$ such that $P[q - \pi + 1 \dots q] = P[1 \dots \pi]$
    - $\pi = 0$ means that such a prefix does not exist

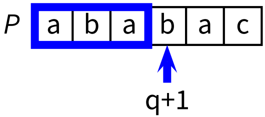


- Restart from $q = \pi$

New Strategy

- $P[1 \dots q]$ is the prefix of P matched so far

- Find the longest prefix of P that is also a suffix of $P[2 \dots q]$
  - i.e., find $0 \leq \pi < q$ such that $P[q - \pi + 1 \dots q] = P[1 \dots \pi]$
    - $\pi = 0$ means that such a prefix does not exist



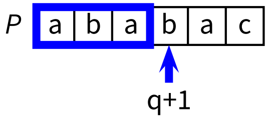
- Restart from $q = \pi$

- Iterate as usual

New Strategy

- $P[1 \dots q]$ is the prefix of P matched so far

- Find the longest prefix of P that is also a suffix of $P[2 \dots q]$
  - i.e., find $0 \le \pi < q$ such that $P[q - \pi + 1 \dots q] = P[1 \dots \pi]$
    - $\pi = 0$ means that such a prefix does not exist



- Restart from $q = \pi$

- Iterate as usual

- In essence, this is the Knuth-Morris-Pratt algorithm

- Given a pattern prefix $P[1 \ldots q]$, the longest prefix of P that is also a suffix of $P[2 \ldots q]$ depends only on P and q

- Given a pattern prefix $P[1 \ldots q]$, the longest prefix of P that is also a suffix of $P[2 \ldots q]$ depends only on P and q
- This prefix is identified by its length $\pi(q)$

The Prefix Function

- Given a pattern prefix $P[1 \dots q]$, the longest prefix of P that is also a suffix of $P[2 \dots q]$ depends only on P and q

- This prefix is identified by its length $\pi(q)$

- Because $\pi(q)$ depends only on P (and q), π can be computed at the beginning by **Prefix-Function**
  - we represent π as an array of length m

- Given a pattern prefix $P[1\ldots q]$, the longest prefix of P that is also a suffix of $P[2\ldots q]$ depends only on P and q

- This prefix is identified by its length $\pi(q)$

- Because $\pi(q)$ depends only on P (and q), π can be computed at the beginning by **Prefix-Function**
  - we represent π as an array of length m

- Example

P	a	b	a	b	a	c

The Prefix Function

- Given a pattern prefix $P[1 \dots q]$, the longest prefix of P that is also a suffix of $P[2 \dots q]$ depends only on P and q

- This prefix is identified by its length $\pi(q)$

- Because $\pi(q)$ depends only on P (and q), π can be computed at the beginning by **Prefix-Function**
  - we represent π as an array of length m

- Example

P	a	b	a	b	a	c
π	0	0	1	2	3	0

The Knuth-Morris-Pratt Algorithm

```
KMP-STRING-MATCHING(T, P)
n = length(T)
m = length(P)
π = PREFIX-FUNCTION(P)
q = 0                          // number of character matched
for i = 1 to n                 // scan the text left-to-right
    while q > 0 and P[q + 1] ≠ T[i]
        q = π[q]               // no match: go back using π
    if P[q + 1] == T[i]
        q = q + 1
    if q == m
        OUTPUT(i − m)
        q = π[q]               // go back for the next match
```

Prefix Function Algorithm

Prefix Function Algorithm

- Computing the prefix function amounts to finding all the occurrences of a pattern P *in itself*

Prefix Function Algorithm

- Computing the prefix function amounts to finding all the occurrences of a pattern *P in itself*
- In fact, **Prefix-Function** is remarkably similar to **KMP-String-Matching**

Prefix-Function(P)

```
m = length(P)
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
```

Prefix Function at Work

PREFIX-FUNCTION(P)

```
m = length(P)
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
```

P	a	b	a	b	a	c

π						

PREFIX-FUNCTION(P)

$m = \mathit{length}(P)$
$\pi[1] = 0$
$k = 0$
for $q = 2$ **to** m
    while $k > 0$ and $P[k+1] \neq P[q]$
        $k = \pi[k]$
    if $P[k+1] == P[q]$
        $k = k+1$
    $\pi[q] = k$

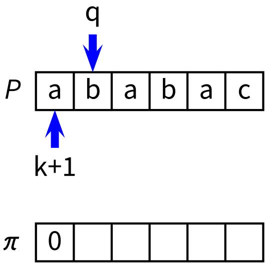
P						
	a	b	a	b	a	c

π						
	0					

Prefix Function at Work

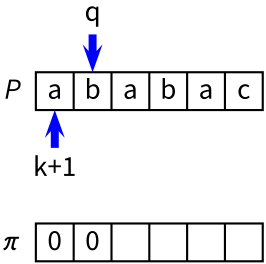
Prefix-Function(P)

```
m = length(P)
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
```



Prefix Function at Work

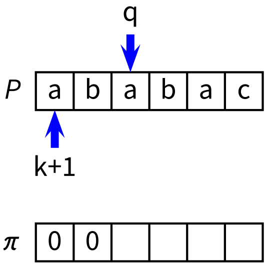
```
PREFIX-FUNCTION(P)
m = length(P)
π[1] = 0
k = 0
for q = 2 to m
        while k > 0 and P[k + 1] ≠ P[q]
                k = π[k]
        if P[k + 1] == P[q]
                k = k + 1
        π[q] = k
```



Prefix Function at Work

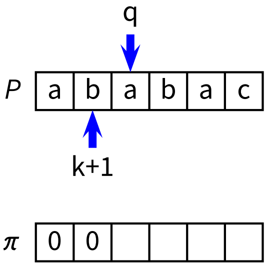
PREFIX-FUNCTION(P)

```
m = length(P)
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
```



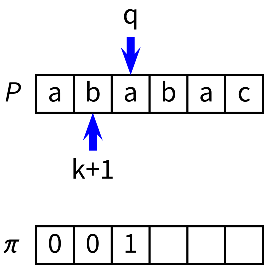
Prefix-Function(P)

```
m = length(P)
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
```



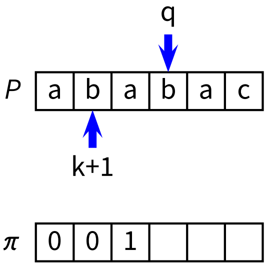
Prefix Function at Work

```
PREFIX-FUNCTION(P)
m = length(P)
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
```



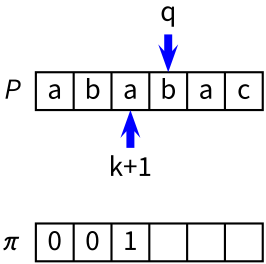
Prefix Function at Work

```
PREFIX-FUNCTION(P)
m = length(P)
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
```



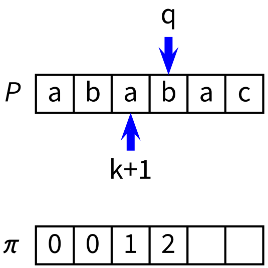
Prefix Function at Work

```
PREFIX-FUNCTION(P)
m = length(P)
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
```

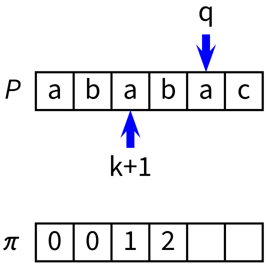


Prefix Function at Work

```
PREFIX-FUNCTION(P)
m = length(P)
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
```



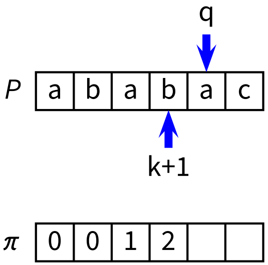
```
PREFIX-FUNCTION(P)
m = length(P)
π[1] = 0
k = 0
for q = 2 to m
        while k > 0 and P[k + 1] ≠ P[q]
                k = π[k]
        if P[k + 1] == P[q]
                k = k + 1
        π[q] = k
```



Prefix Function at Work

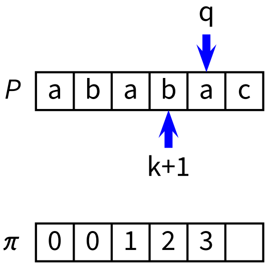
Prefix-Function(P)

$m = length(P)$
$\pi[1] = 0$
$k = 0$
for $q = 2$ **to** m
  while $k > 0$ and $P[k+1] \neq P[q]$
    $k = \pi[k]$
  if $P[k+1] == P[q]$
    $k = k + 1$
  $\pi[q] = k$



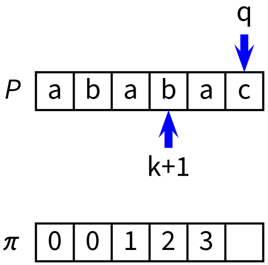
Prefix Function at Work

```
PREFIX-FUNCTION(P)
m = length(P)
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
```

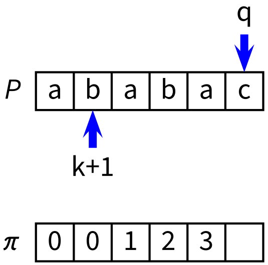


Prefix Function at Work

```
PREFIX-FUNCTION(P)
m = length(P)
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
```

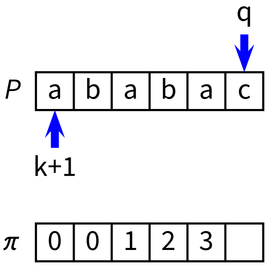


```
PREFIX-FUNCTION(P)
m = length(P)
π[1] = 0
k = 0
for q = 2 to m
      while k > 0 and P[k + 1] ≠ P[q]
            k = π[k]
      if P[k + 1] == P[q]
            k = k + 1
      π[q] = k
```



PREFIX-FUNCTION(P)

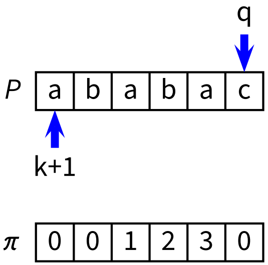
```
m = length(P)
π[1] = 0
k = 0
for q = 2 to m
        while k > 0 and P[k+1] ≠ P[q]
                k = π[k]
        if P[k+1] == P[q]
                k = k + 1
        π[q] = k
```



Prefix Function at Work

PREFIX-FUNCTION(P)

```
m = length(P)
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
```



Complexity of KMP

Complexity of KMP

- $O(n)$ for the search phase

Complexity of KMP

- $O(n)$ for the search phase
- $O(m)$ for the pre-processing of the pattern

Complexity of KMP

- $O(n)$ for the search phase
- $O(m)$ for the pre-processing of the pattern
- The complexity analysis is non-trivial

Complexity of KMP

- $O(n)$ for the search phase
- $O(m)$ for the pre-processing of the pattern
- The complexity analysis is non-trivial
- Can we do better?

Comments on KMP

- Knuth-Morris-Pratt is $\Omega(n)$
  - KMP will *always* go through *at least n* character comparisons
  - it fixes our “wrong” algorithm in the case of *periodic* patterns and texts

Comments on KMP

- Knuth-Morris-Pratt is $\Omega(n)$
  - KMP will *always* go through *at least n* character comparisons
  - it fixes our “wrong” algorithm in the case of *periodic* patterns and texts

- Perhaps there’s another algorithm that works better on the average case
  - e.g., in the absence of periodic patterns

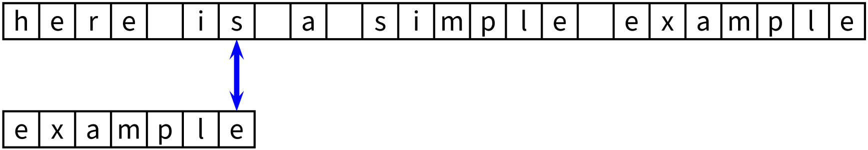
A New Strategy

h	e	r	e		i	s		a		s	i	m	p	l	e		e	x	a	m	p	l	e

A New Strategy

h	e	r	e		i	s		a		s	i	m	p	l	e		e	x	a	m	p	l	e

e	x	a	m	p	l	e



- We match the pattern right-to-left

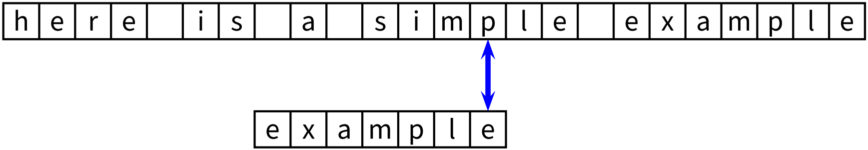
A New Strategy

h	e	r	e		i	s		a		s	i	m	p	l	e		e	x	a	m	p	l	e

e	x	a	m	p	l	e

- We match the pattern right-to-left
- If we find a bad character α in the text, we can shift
  - so that the pattern skips α, if α is not in the pattern

A New Strategy



- We match the pattern right-to-left
- If we find a bad character α in the text, we can shift
  - so that the pattern skips α, if α is not in the pattern

A New Strategy

h	e	r	e		i	s		a		s	i	m	p	l	e		e	x	a	m	p	l	e

e	x	a	m	p	l	e

- We match the pattern right-to-left

- If we find a bad character α in the text, we can shift
  - so that the pattern skips α, if α is not in the pattern
  - so that the pattern lines up with the rightmost occurrence of α in the pattern, if the pattern contains α

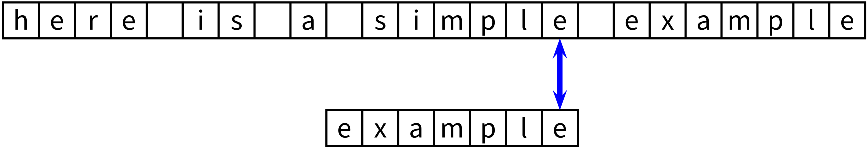
A New Strategy

h	e	r	e		i	s		a		s	i	m	p	l	e		e	x	a	m	p	l	e

e	x	a	m	p	l	e

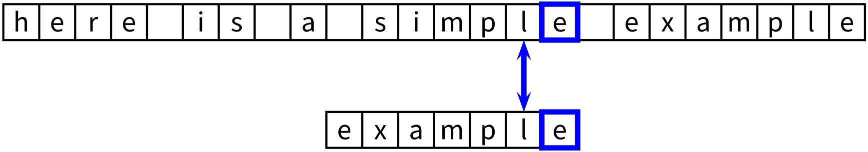
- We match the pattern right-to-left
- If we find a bad character α in the text, we can shift
  - so that the pattern skips α, if α is not in the pattern
  - so that the pattern lines up with the rightmost occurrence of α in the pattern, if the pattern contains α

A New Strategy



- We match the pattern right-to-left
- If we find a bad character α in the text, we can shift
  - so that the pattern skips α, if α is not in the pattern
  - so that the pattern lines up with the rightmost occurrence of α in the pattern, if the pattern contains α

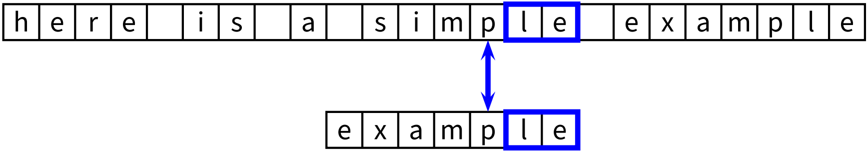
A New Strategy



- We match the pattern right-to-left

- If we find a bad character α in the text, we can shift
  - so that the pattern skips α, if α is not in the pattern
  - so that the pattern lines up with the rightmost occurrence of α in the pattern, if the pattern contains α

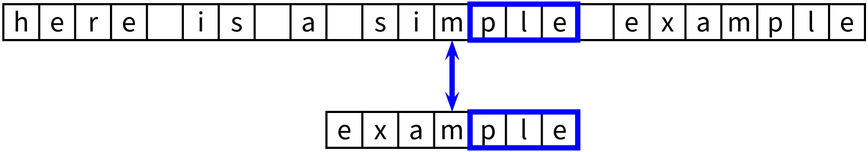
A New Strategy



- We match the pattern right-to-left

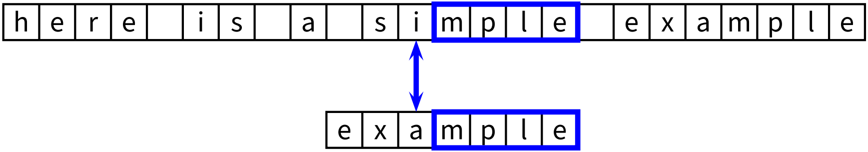
- If we find a bad character α in the text, we can shift
  - so that the pattern skips α, if α is not in the pattern
  - so that the pattern lines up with the rightmost occurrence of α in the pattern, if the pattern contains α

A New Strategy



- We match the pattern right-to-left

- If we find a bad character α in the text, we can shift
  - so that the pattern skips α, if α is not in the pattern
  - so that the pattern lines up with the rightmost occurrence of α in the pattern, if the pattern contains α



- We match the pattern right-to-left
- If we find a bad character α in the text, we can shift
  - so that the pattern skips α, if α is not in the pattern
  - so that the pattern lines up with the rightmost occurrence of α in the pattern, if the pattern contains α

A New Strategy

h	e	r	e		i	s		a		s	i	m	p	l	e		e	x	a	m	p	l	e

e	x	a	m	p	l	e

- We match the pattern right-to-left
- If we find a bad character α in the text, we can shift
  - so that the pattern skips α, if α is not in the pattern
  - so that the pattern lines up with the rightmost occurrence of α in the pattern, if the pattern contains α

h	e	r	e		i	s		a		s	i	m	p	l	e		e	x	a	m	p	l	e

e	x	a	m	p	l	e

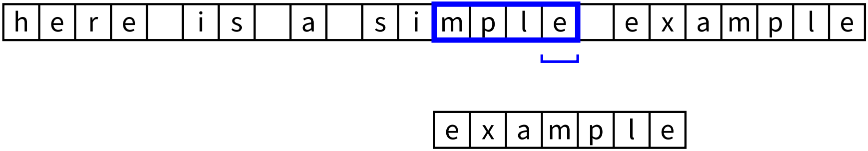
- We match the pattern right-to-left
- If we find a bad character α in the text, we can shift
  - so that the pattern skips α, if α is not in the pattern
  - so that the pattern lines up with the rightmost occurrence of α in the pattern, if the pattern contains α

h	e	r	e		i	s		a		s	i	m	p	l	e		e	x	a	m	p	l	e

e	x	a	m	p	l	e

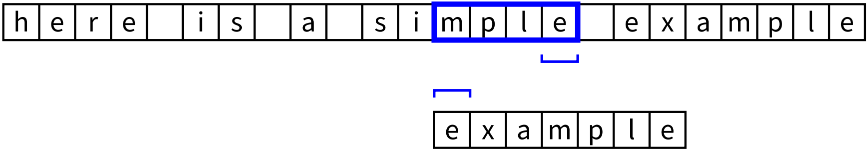
- We match the pattern right-to-left
- If we find a bad character α in the text, we can shift
  - so that the pattern skips α, if α is not in the pattern
  - so that the pattern lines up with the rightmost occurrence of α in the pattern, if the pattern contains α

A New Strategy

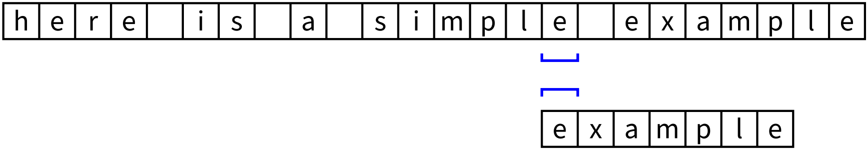


- We match the pattern right-to-left
- If we find a bad character α in the text, we can shift
  - so that the pattern skips α, if α is not in the pattern
  - so that the pattern lines up with the rightmost occurrence of α in the pattern, if the pattern contains α

A New Strategy

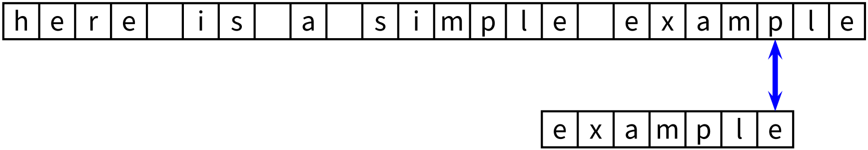


- We match the pattern right-to-left
- If we find a bad character α in the text, we can shift
  - so that the pattern skips α, if α is not in the pattern
  - so that the pattern lines up with the rightmost occurrence of α in the pattern, if the pattern contains α



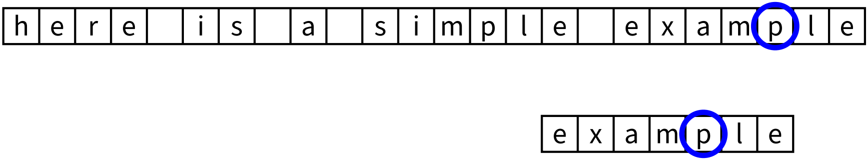
- We match the pattern right-to-left
- If we find a bad character α in the text, we can shift
  - so that the pattern skips α, if α is not in the pattern
  - so that the pattern lines up with the rightmost occurrence of α in the pattern, if the pattern contains α
  - so that a pattern prefix lines up with a suffix of the current partial (or complete) match

A New Strategy



- We match the pattern right-to-left
- If we find a bad character α in the text, we can shift
  - so that the pattern skips α, if α is not in the pattern
  - so that the pattern lines up with the rightmost occurrence of α in the pattern, if the pattern contains α
  - so that a pattern prefix lines up with a suffix of the current partial (or complete) match

A New Strategy



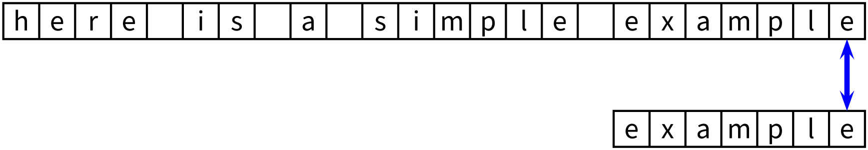
- We match the pattern right-to-left
- If we find a bad character α in the text, we can shift
  - so that the pattern skips α, if α is not in the pattern
  - so that the pattern lines up with the rightmost occurrence of α in the pattern, if the pattern contains α
  - so that a pattern prefix lines up with a suffix of the current partial (or complete) match

A New Strategy

h	e	r	e		i	s		a		s	i	m	p	l	e		e	x	a	m	p	l	e

e	x	a	m	p	l	e

- We match the pattern right-to-left
- If we find a bad character α in the text, we can shift
  - so that the pattern skips α, if α is not in the pattern
  - so that the pattern lines up with the rightmost occurrence of α in the pattern, if the pattern contains α
  - so that a pattern prefix lines up with a suffix of the current partial (or complete) match



- We match the pattern right-to-left
- If we find a bad character α in the text, we can shift
  - so that the pattern skips α, if α is not in the pattern
  - so that the pattern lines up with the rightmost occurrence of α in the pattern, if the pattern contains α
  - so that a pattern prefix lines up with a suffix of the current partial (or complete) match

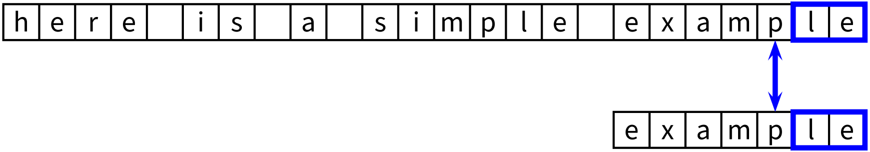
A New Strategy

h	e	r	e		i	s		a		s	i	m	p	l	e		e	x	a	m	p	l	$\boxed{e}$

e	x	a	m	p	l	$\boxed{e}$

- We match the pattern right-to-left
- If we find a bad character α in the text, we can shift
  - so that the pattern skips α, if α is not in the pattern
  - so that the pattern lines up with the rightmost occurrence of α in the pattern, if the pattern contains α
  - so that a pattern prefix lines up with a suffix of the current partial (or complete) match

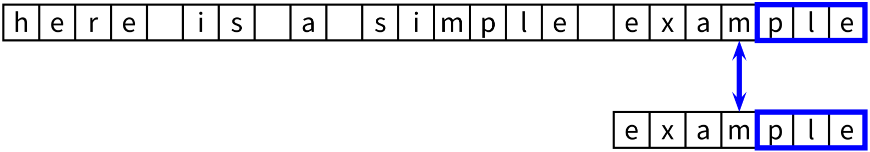
A New Strategy



- We match the pattern right-to-left

- If we find a bad character α in the text, we can shift
  - so that the pattern skips α, if α is not in the pattern
  - so that the pattern lines up with the rightmost occurrence of α in the pattern, if the pattern contains α
  - so that a pattern prefix lines up with a suffix of the current partial (or complete) match

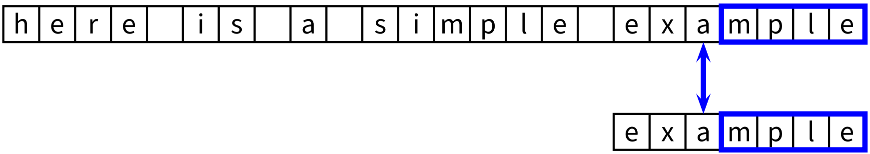
A New Strategy



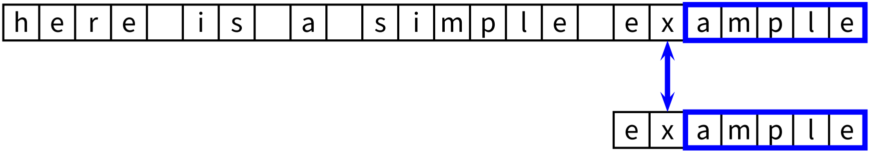
- We match the pattern right-to-left

- If we find a bad character α in the text, we can shift
  - so that the pattern skips α, if α is not in the pattern
  - so that the pattern lines up with the rightmost occurrence of α in the pattern, if the pattern contains α
  - so that a pattern prefix lines up with a suffix of the current partial (or complete) match

A New Strategy

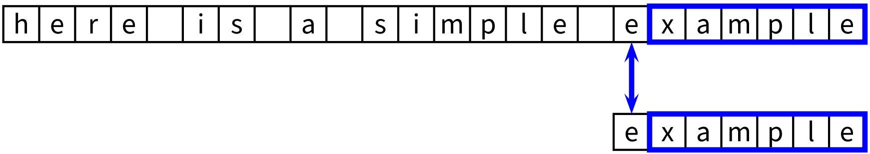


- We match the pattern right-to-left
- If we find a bad character α in the text, we can shift
  - so that the pattern skips α, if α is not in the pattern
  - so that the pattern lines up with the rightmost occurrence of α in the pattern, if the pattern contains α
  - so that a pattern prefix lines up with a suffix of the current partial (or complete) match



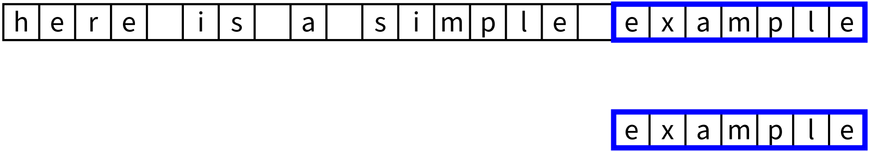
- We match the pattern right-to-left
- If we find a bad character α in the text, we can shift
  - so that the pattern skips α, if α is not in the pattern
  - so that the pattern lines up with the rightmost occurrence of α in the pattern, if the pattern contains α
  - so that a pattern prefix lines up with a suffix of the current partial (or complete) match

A New Strategy



- We match the pattern right-to-left
- If we find a bad character α in the text, we can shift
  - so that the pattern skips α, if α is not in the pattern
  - so that the pattern lines up with the rightmost occurrence of α in the pattern, if the pattern contains α
  - so that a pattern prefix lines up with a suffix of the current partial (or complete) match

A New Strategy



- We match the pattern right-to-left
- If we find a bad character α in the text, we can shift
  - so that the pattern skips α, if α is not in the pattern
  - so that the pattern lines up with the rightmost occurrence of α in the pattern, if the pattern contains α
  - so that a pattern prefix lines up with a suffix of the current partial (or complete) match

A New Strategy

h	e	r	e		i	s		a		s	i	m	p	l	e		e	x	a	m	p	l	e
																	e	x	a	m	p	l	e

- We match the pattern right-to-left
- If we find a bad character α in the text, we can shift
  - so that the pattern skips α, if α is not in the pattern
  - so that the pattern lines up with the rightmost occurrence of α in the pattern, if the pattern contains α
  - so that a pattern prefix lines up with a suffix of the current partial (or complete) match
- In essence, this is the Boyer-Moore algorithm

Comments on Boyer-Moore

Comments on Boyer-Moore

- Like KMP, Boyer-Moore includes a pre-processing phase

Comments on Boyer-Moore

- Like KMP, Boyer-Moore includes a pre-processing phase
- The pre-processing is $O(m)$

Comments on Boyer-Moore

- Like KMP, Boyer-Moore includes a pre-processing phase
- The pre-processing is $O(m)$
- The search phase is $O(nm)$

Comments on Boyer-Moore

- Like KMP, Boyer-Moore includes a pre-processing phase
- The pre-processing is $O(m)$
- The search phase is $O(nm)$
- The search phase can be as low as $O(n/m)$ in common cases

Comments on Boyer-Moore

- Like KMP, Boyer-Moore includes a pre-processing phase
- The pre-processing is $O(m)$
- The search phase is $O(nm)$
- The search phase can be as low as $O(n/m)$ in common cases
- In practice, Boyer-Moore is the fastest string-matching algorithm for most applications