طراحی و تحلیل الگوریتم‌ها

مبحث دوازدهم

مطالعه‌ی موضوعی الگوریتم‌ها

تطابق رشته‌ها

String Matching

کاظم فولادی قلعه

دانشکده مهندسی، دانشکدگان فارابی

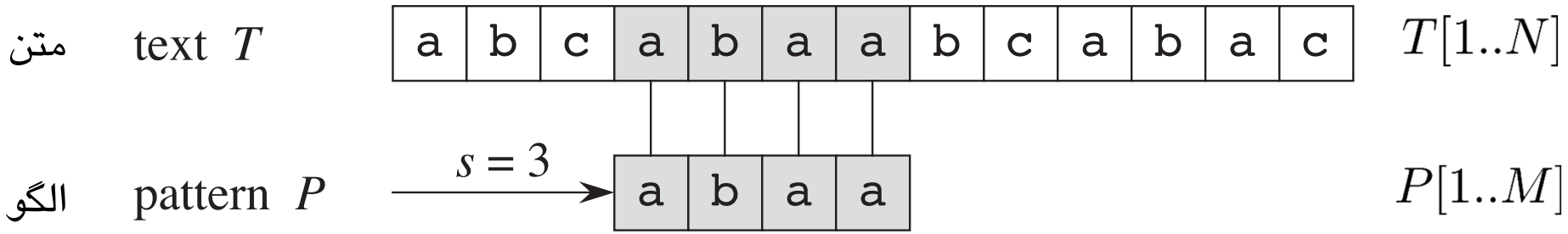
دانشگاه تهران

http://courses.fouladi.ir/algorithm

تطابق رشته‌ها

STRING MATCHING

تطابق رشته‌ها: یافتن همه‌ی موارد وقوع یک الگو در یک متن



An example of the string-matching problem, where we want to find all occurrences of the pattern $P = \texttt{abaa}$ in the text $T = \texttt{abcabaabcabac}$. The pattern occurs only once in the text, at shift $s = 3$, which we call a valid shift. A vertical line connects each character of the pattern to its matching character in the text, and all matched characters are shaded.

تطابق رشته‌ها

مهم‌ترین الگوریتم‌های موجود

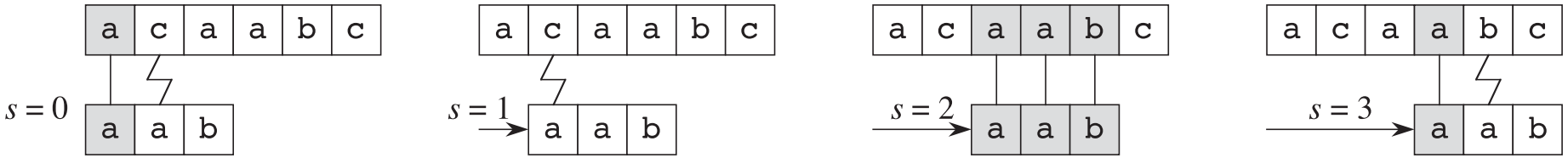
STRING MATCHING

Algorithm	Preprocessing time	Matching time
Naive	0	$O((n-m+1)m)$
Rabin-Karp	$\Theta(m)$	$O((n-m+1)m)$
Finite automaton	$O(m\|\Sigma\|)$	$\Theta(n)$
Knuth-Morris-Pratt	$\Theta(m)$	$\Theta(n)$

تطابق رشته‌ها

الگوریتم ساده: مثال

NAIVE ALGORITHM



The operation of the naive string matcher for the pattern $P = \texttt{aab}$ and the text $T = \texttt{acaabc}$. We can imagine the pattern P as a template that we slide next to the text. **(a)–(d)** The four successive alignments tried by the naive string matcher. In each part, vertical lines connect corresponding regions found to match (shown shaded), and a jagged line connects the first mismatched character found, if any. The algorithm finds one occurrence of the pattern, at shift $s = 2$, shown in part (c).

تطابق رشته‌ها

الگوریتم جستجوی کورکورانه

BRUTE-FORCE ALGORITHM

الگوریتم کورکورانه، الگو را با متن مقایسه می‌کند
(هر بار یک کاراکتر)
تا یک کاراکتر نامطابق پیدا شود.

تطابق رشته‌ها

الگوریتم جستجوی کورکورانه : مثال

```
TWO ROADS DIVERGED IN A YELLOW WOOD
ROADS
---------------------------------------
TWO ROADS DIVERGED IN A YELLOW WOOD
  ROADS
---------------------------------------
TWO ROADS DIVERGED IN A YELLOW WOOD
   ROADS
---------------------------------------
TWO ROADS DIVERGED IN A YELLOW WOOD
    ROADS
---------------------------------------
TWO ROADS DIVERGED IN A YELLOW WOOD
     ROADS
---------------------------------------
```

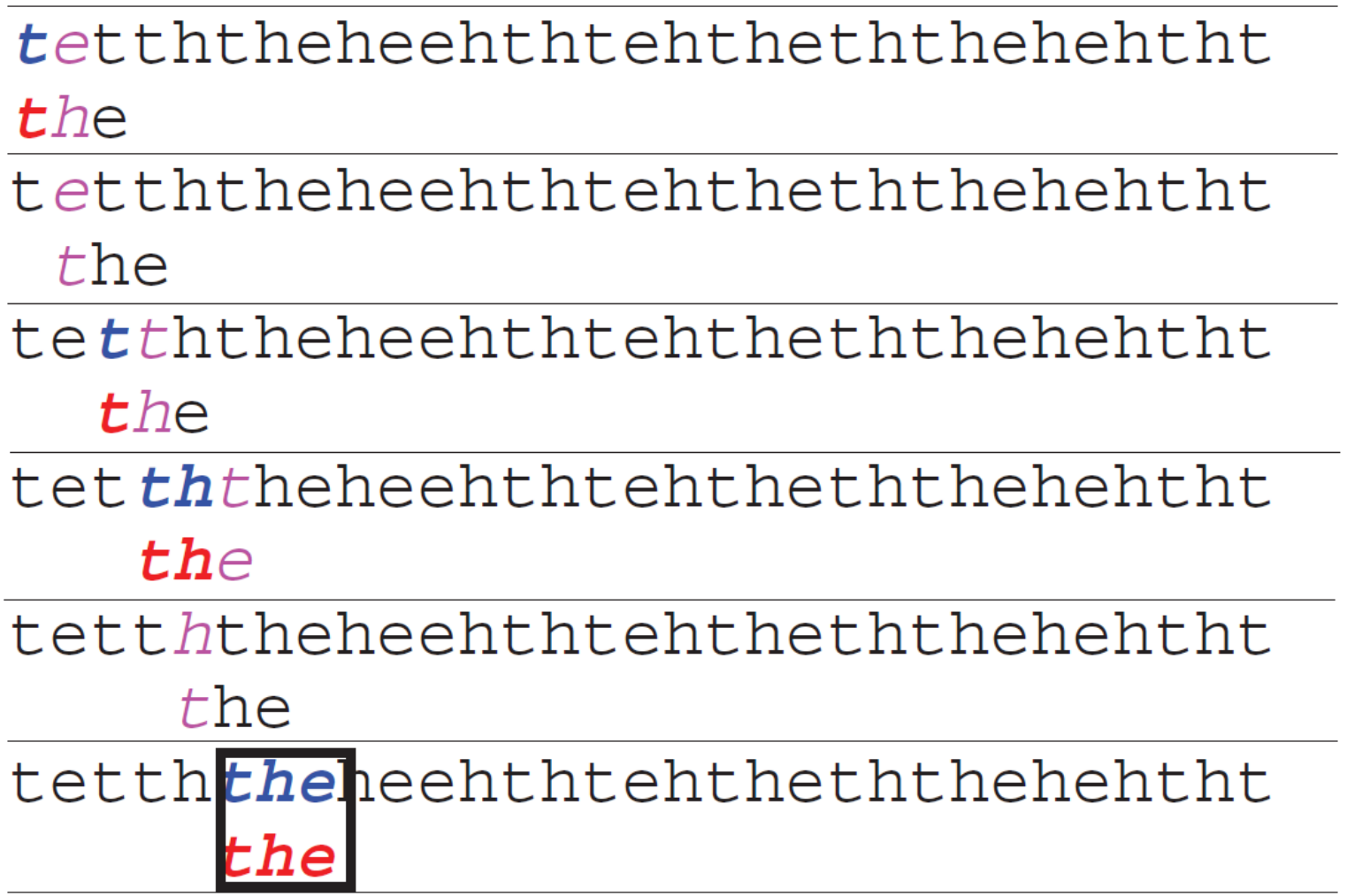
تطابق رشته‌ها

الگوریتم جستجوی کورکورانه : شبه‌کد

```
do
   if (text letter == pattern letter)
      compare next letter of pattern to next
      letter of text
   else
      move pattern down text by one letter
while (entire pattern found or end of text)
```

تطابق رشته‌ها

الگوریتم جستجوی کورکورانه : مثال



تطابق رشته‌ها

الگوریتم جستجوی کورکورانه: زمان اجرا

تعداد مقایسه‌ها: $O(N - M + 1)$

زمان اجرای بدترین حالت: $O(NM)$

تطابق رشته‌ها

الگوریتم جستجوی کورکورانه: زمان اجرا: مثال

1) *AAAAA*AAAAAAAAAAAAAAAAAAAAAH
AAAAH **5 comparisons made**

2) *AAAAA*AAAAAAAAAAAAAAAAAAAAAH
AAAAH **5 comparisons made**

3) *AAAAA*AAAAAAAAAAAAAAAAAAAAAH
AAAAH **5 comparisons made**

4) *AAAAA*AAAAAAAAAAAAAAAAAAAAAH
AAAAH **5 comparisons made**

5) *AAAAA*AAAAAAAAAAAAAAAAAAAAAH
AAAAH **5 comparisons made**

....

N) AAAAAAAAAAAAAAAAAAAAAAAAA*AAAAH*
5 comparisons made *AAAAH*

تطابق رشته‌ها

الگوریتم کورکورانه

```
NAIVE-STRING-MATCHER(T, P)
1   n = T.length
2   m = P.length
3   for s = 0 to n − m
4       if P[1..m] == T[s + 1..s + m]
5           print "Pattern occurs with shift" s
```

الگوریتم کورکورانه، الگو را با متن مقایسه می‌کند
(هر بار یک کاراکتر)
تا یک کاراکتر نامطابق پیدا شود.

تطابق رشته‌ها: الگوریتم رابین ـ کارپ

RABIN-KARP ALGORITHM

الگوریتم جستجوی رشته‌ی رابین ـ کارپ،
برای الگو
و
برای هر زیردنباله‌ی M کاراکتری (که باید مقایسه شود)
یک مقدار **هش** را محاسبه می‌کند.

اگر مقادیر هش مساوی نبودند،
الگوریتم مقدار هش را برای زیردنباله‌ی M کاراکتری بعدی محاسبه می‌کند.

اگر مقادیر هش مساوی بودند،
الگوریتم بین الگو و زیردنباله‌ی M کاراکتری مقایسه‌ی کورکورانه انجام می‌دهد.

تطابق رشته‌ها: الگوریتم رابین ـ کارپ

مثال

Hash value of "AAAAA" is 37

Hash value of "AAAAH" is 100

1) AAAAAAAAAAAAAAAAAAAAAAAAAAAAH
   AAAAH
   37≠100 **1 comparison made**
2) AAAAAAAAAAAAAAAAAAAAAAAAAAAAH
    AAAAH
   37≠100 **1 comparison made**
3) AAAAAAAAAAAAAAAAAAAAAAAAAAAAH
     AAAAH
   37≠100 **1 comparison made**
4) AAAAAAAAAAAAAAAAAAAAAAAAAAAAH
      AAAAH
   37≠100 **1 comparison made**

...

N) AAAAAAAAAAAAAAAAAAAAAAAAAAAAH
                              AAAAH
6 comparisons made 100=100

تطابق رشته‌ها: الگوریتم رابین ـ کارپ

شبه کد

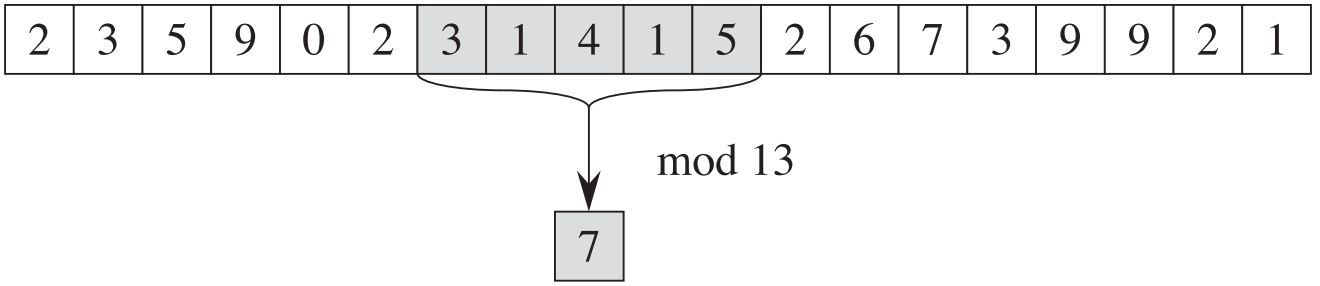
pattern is M characters long

hash_p=hash value of pattern
hash_t=hash value of first M letters in
body of text

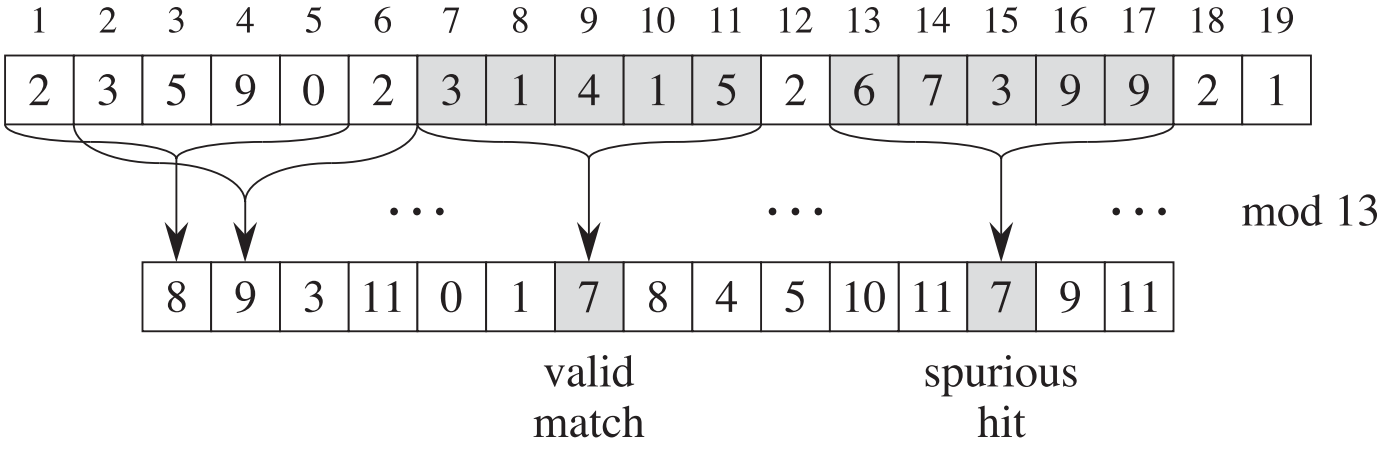
```
do
  if (hash_p == hash_t)
    brute force comparison of pattern
    and selected section of text
  hash_t = hash value of next section of
           text, one character over
while (end of text or
       brute force comparison == true)
```

تطابق رشته‌ها: الگوریتم رابین ـ کارپ

مثال: محاسبه و مقایسه‌ی هش‌ها



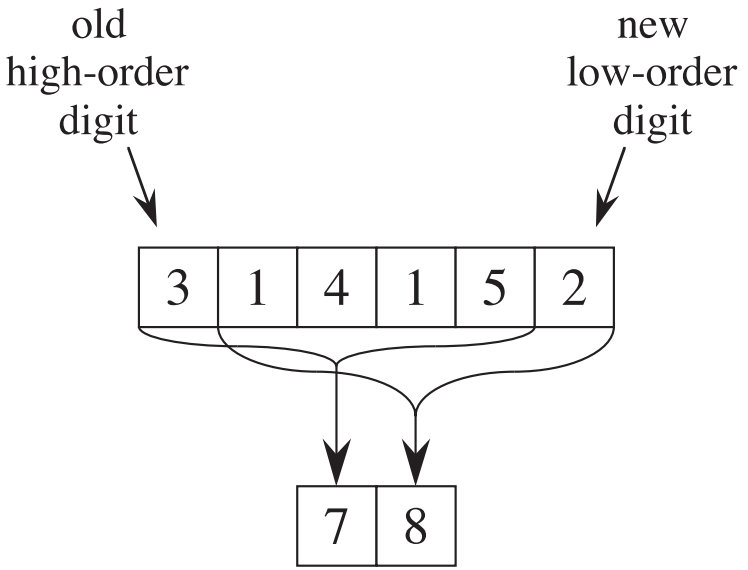
(a)



(b)

تطابق رشته‌ها: الگوریتم رابین ـ کارپ

محاسبه‌ی مقدار هش



(c)

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

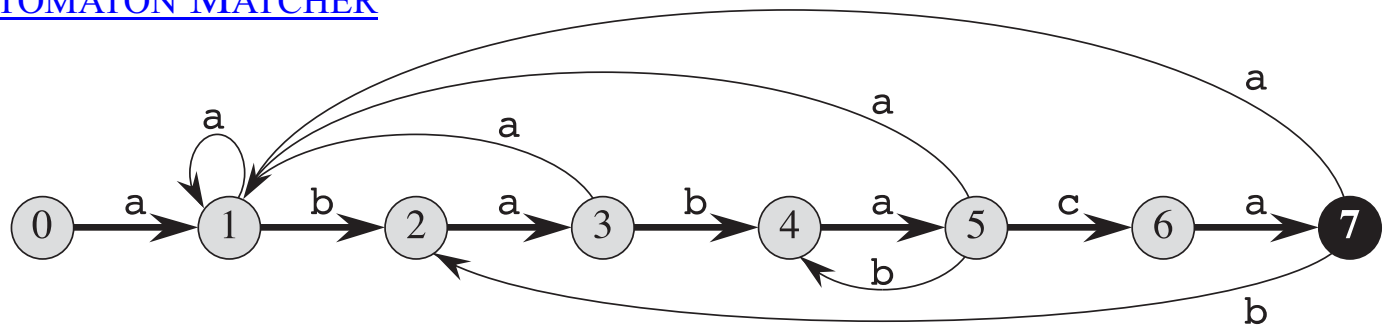
تطابق رشته‌ها: الگوریتم رابین ـ کارپ

شبه‌کد

```
RABIN-KARP-MATCHER(T, P, d, q)
 1  n = T.length
 2  m = P.length
 3  h = d^(m−1) mod q
 4  p = 0
 5  t_0 = 0
 6  for i = 1 to m                          // preprocessing
 7       p = (dp + P[i]) mod q
 8       t_0 = (dt_0 + T[i]) mod q
 9  for s = 0 to n − m                      // matching
10       if p == t_s
11            if P[1..m] == T[s + 1..s + m]
12                 print "Pattern occurs with shift" s
13       if s < n − m
14            t_{s+1} = (d(t_s − T[s + 1]h) + T[s + m + 1]) mod q
```

تطابق رشته‌ها: الگوریتم تطابق با آتوماتون متناهی

FINITE AUTOMATON MATCHER



(a)

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

(b)

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

(c)

تطابق رشته‌ها: الگوریتم تطابق با آتوماتون متناهی

شبه‌کد

FINITE AUTOMATON MATCHER

```
FINITE-AUTOMATON-MATCHER(T, δ, m)
1   n = T.length
2   q = 0
3   for i = 1 to n
4       q = δ(q, T[i])
5       if q == m
6           print "Pattern occurs with shift" i − m
```

تطابق رشته‌ها: الگوریتم کنوث ـ موریس ـ پرَت

KNUTH-MORRIS-PRATT ALGORITHM (KMP)

الگوریتم KMP از ردگیری اطلاعات حاصل از مقایسه‌های قبلی استفاده می‌کند:
یک تابع پیشوند π محاسبه می‌شود
تا نشان دهد در صورت شکست، چه میزان از مقایسه‌ی قبلی می‌تواند استفاده‌ی مجدد شود.

تطابق رشته‌ها: الگوریتم کنوث ـ موریس ـ پرَت

تابع پیشوند

KNUTH-MORRIS-PRATT ALGORITHM (KMP)

تابع پیشوند π

$\pi(i)$ = طول بلندترین پیشوند الگوی $P[1..i]$ که پسوند $P[2..i]$ باشد.

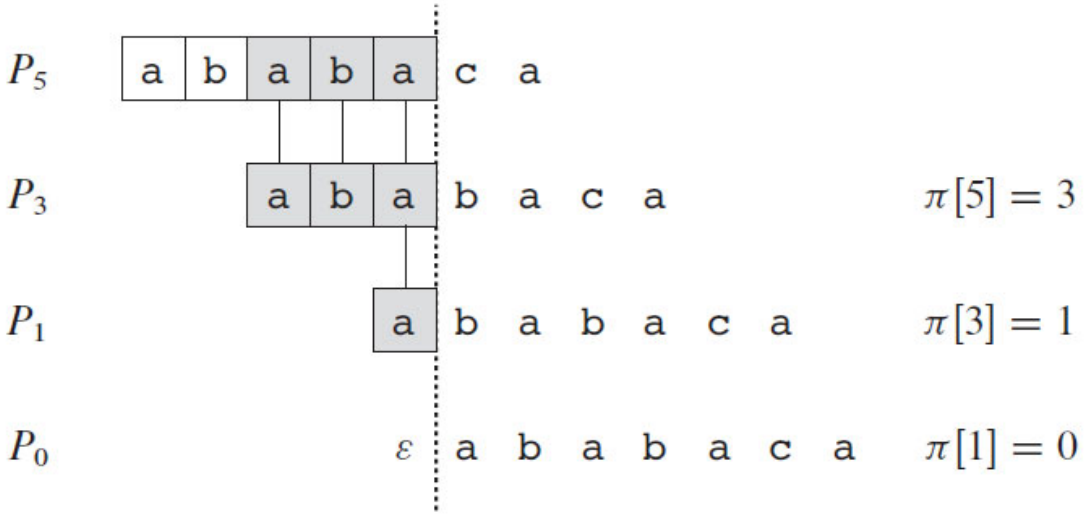
نشان می‌دهد که
چه‌میزان از ابتدای رشته
با بخشی که بلافاصله قبل از مقایسه‌ی شکست‌خورده قرار دارد،
تطابق می‌یابد.

تطابق رشته‌ها: الگوریتم کنوث ـ موریس ـ پرَت

KNUTH-MORRIS-PRATT ALGORITHM (KMP)

i	1	2	3	4	5	6	7
$P[i]$	a	b	a	b	a	c	a
$\pi[i]$	0	0	1	2	3	0	1

(a)

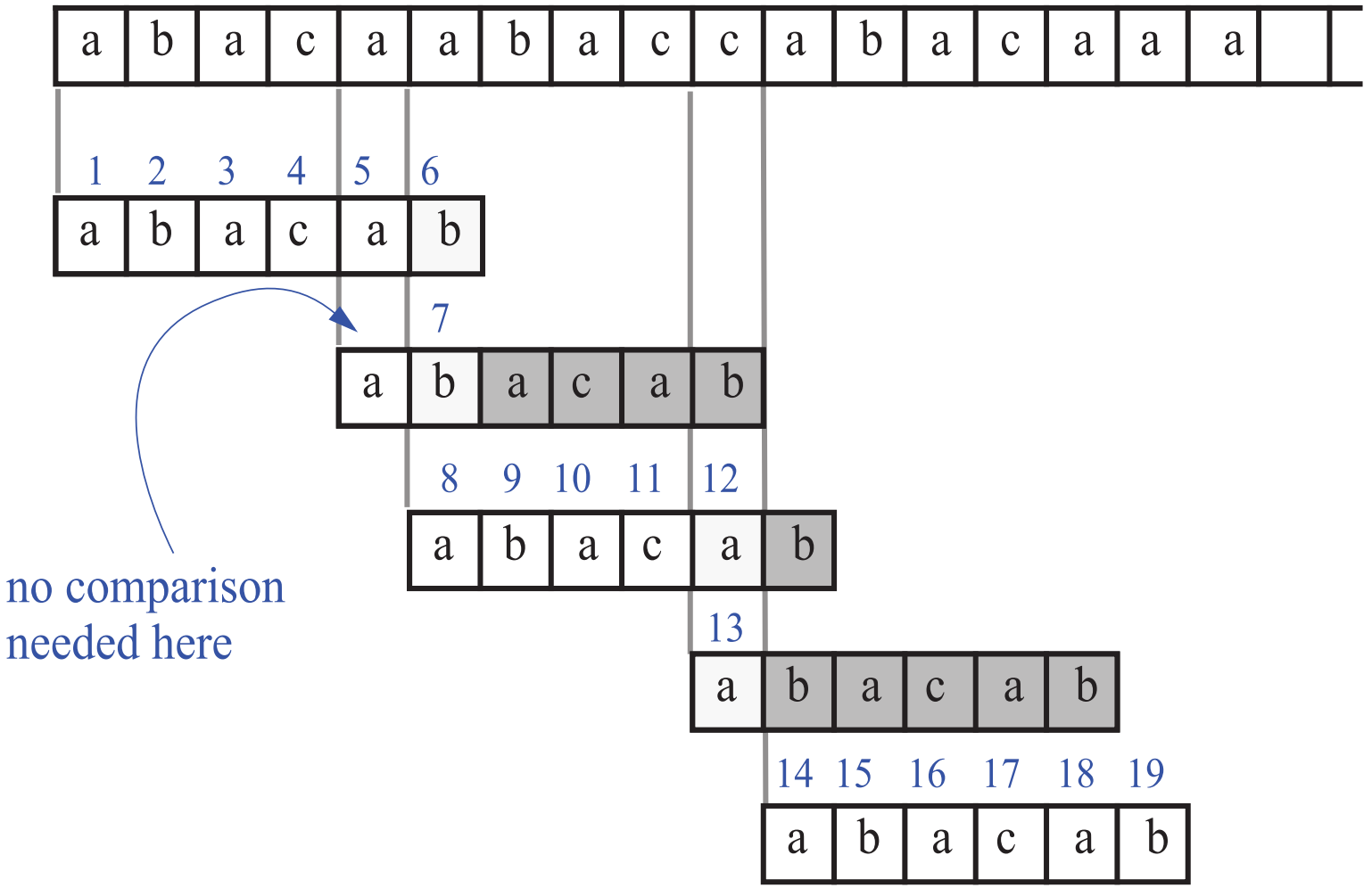


(b)

تطابق رشته‌ها: الگوریتم کنوث ـ موریس ـ پرَت

مثال

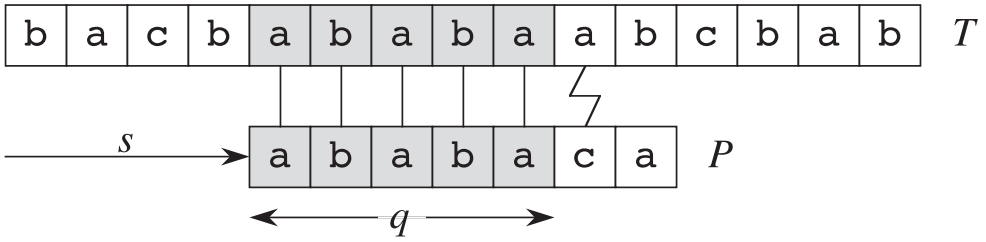
KNUTH-MORRIS-PRATT ALGORITHM (KMP)



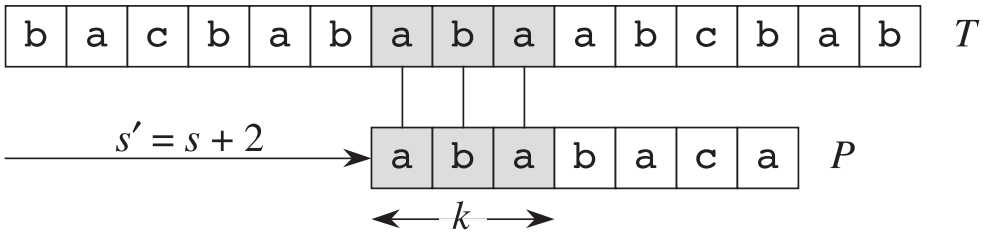
تطابق رشته‌ها: الگوریتم کنوث ـ موریس ـ پرَت

مثال

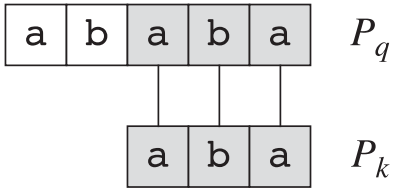
KNUTH-MORRIS-PRATT ALGORITHM (KMP)



(a)



(b)



(c)

تطابق رشته‌ها: الگوریتم کنوث ـ موریس ـ پرَت

شبه‌کد

KNUTH-MORRIS-PRATT ALGORITHM (KMP)

```
KMP-MATCHER(T, P)
 1  n = T.length
 2  m = P.length
 3  π = COMPUTE-PREFIX-FUNCTION(P)
 4  q = 0                                  // number of characters matched
 5  for i = 1 to n                         // scan the text from left to right
 6      while q > 0 and P[q + 1] ≠ T[i]
 7          q = π[q]                       // next character does not match
 8      if P[q + 1] == T[i]
 9          q = q + 1                      // next character matches
10      if q == m                          // is all of P matched?
11          print "Pattern occurs with shift" i − m
12          q = π[q]                       // look for the next match
```

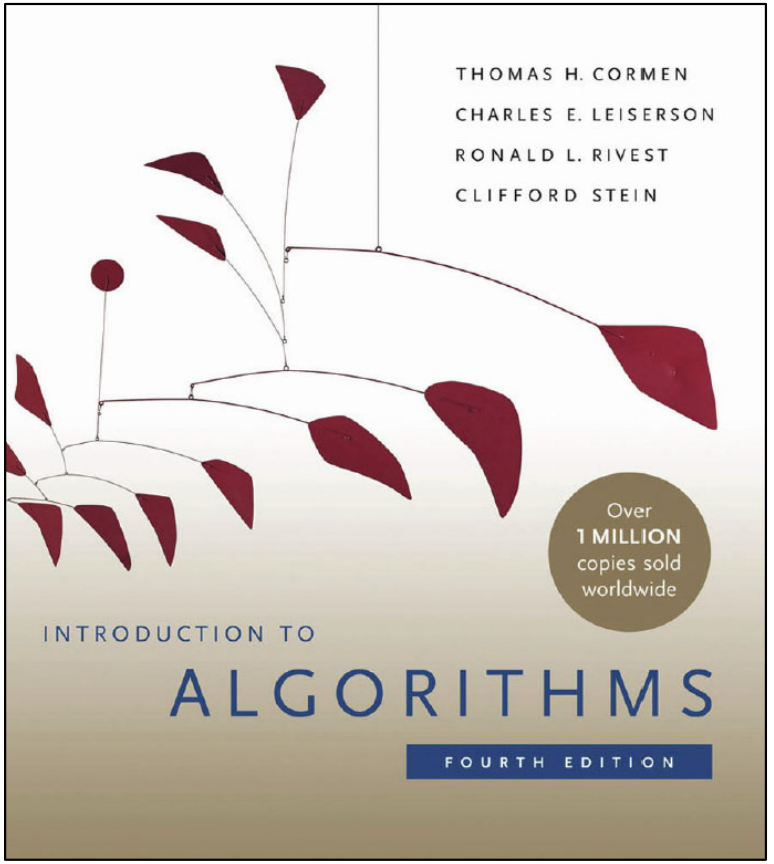
تطابق رشته‌ها: الگوریتم کنوث ـ موریس ـ پرَت

شبه‌کد: تابع محاسبه‌ی پیشوند

KNUTH-MORRIS-PRATT ALGORITHM (KMP)

```
COMPUTE-PREFIX-FUNCTION(P)
 1  m = P.length
 2  let π[1..m] be a new array
 3  π[1] = 0
 4  k = 0
 5  for q = 2 to m
 6       while k > 0 and P[k + 1] ≠ P[q]
 7            k = π[k]
 8       if P[k + 1] == P[q]
 9            k = k + 1
10       π[q] = k
11  return π
```

مرجع



T.H. Cormen, C.E. Leiserson, R.L. Rivest, C. Stein,
Introduction to Algorithms,
4th Edition, MIT Press, 2022.

Chapter 32

32 String Matching

Text-editing programs frequently need to find all occurrences of a pattern in the text. Typically, the text is a document being edited, and the pattern searched for is a particular word supplied by the user. Efficient algorithms for this problem —called "string matching"—can greatly aid the responsiveness of the text-editing program. Among their many other applications, string-matching algorithms search for particular patterns in DNA sequences. Internet search engines also use them to find web pages relevant to queries.

The string-matching problem can be stated formally as follows. The text is given as an array $T[1:n]$ of length n, and the pattern is an array $P[1:m]$ of length $m \le n$. The elements of P and T are characters drawn from an alphabet Σ, which is a finite set of characters. For example, Σ could be the set $\{0, 1\}$, or it could be the set $\{a, b, \ldots, z\}$. The character arrays P and T are often called *strings* of characters.

As Figure 32.1 shows, pattern P *occurs with shift* s in text T (or, equivalently, that pattern P *occurs beginning at position* $s + 1$ in text T) if $0 \le s \le n - m$ and $T[s + 1 : s + m] = P[1 : m]$, that is, if $T[s + j] = P[j]$, for $1 \le j \le m$. If P occurs with shift s in T, then s is a *valid shift*, and otherwise, s is an *invalid shift*. The *string-matching problem* is the problem of finding all valid shifts with which a given pattern P occurs in a given text T.

Figure 32.1 An example of the string-matching problem to find all occurrences of the pattern P = abaa in the text T = abcabaabcabac. The pattern occurs only once in the text, at shift s = 3, which is a valid shift. A vertical line connects each character of the pattern to its matching character in the text, and all matched characters are shaded blue.