Wyszukiwanie wzorca

Na podstawie Cormen et al - Wprowadzenie do algorytmów

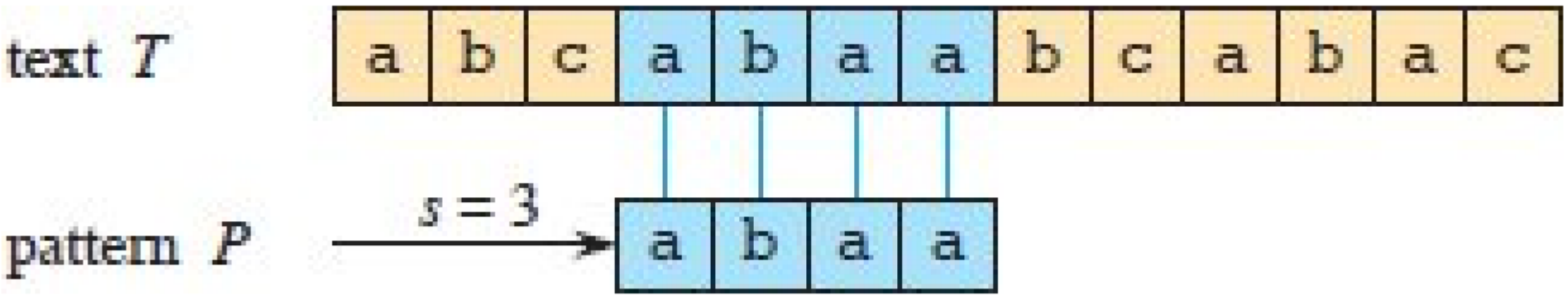
Problem wyszukiwania wzorca

- **Tekst** jest tablicą T[1..n] długości n
- **Wzorzec** jest tablicą P[1..m] długości m <= n
- Elementami tablic P i T są symbole należące do pewnego skończonego **alfabetu ∑**, np. ∑={0,1}, ∑={a,b,c,...,x,y,z}
- Tablice zawierające symbole P i T są także często nazywane **słowami**

Problem wyszukiwania wzorca

- Mówimy, że wzorzec P występuje z **przesunięciem s** w tekście T lub, równoważnie, że wzorzec P występuje począwszy **od pozycji s + 1** w tekście T, gdy 0 <= s <= n-m oraz **T[s+1..s + m] = P[1..m]**, tzn. gdy **T[s + j] = P[j] dla 1 <= j <= m**.
- Jeśli P występuje z przesunięciem s w T, to s nazywamy **poprawnym przesunięciem** w przeciwnym razie s nazywamy **niepoprawnym przesunięciem**.
- Problem wyszukiwania wzorca polega na znajdowaniu wszystkich poprawnych przesunięć dla danego wzorca P w tekście T.

Przykład



Algorytmy

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

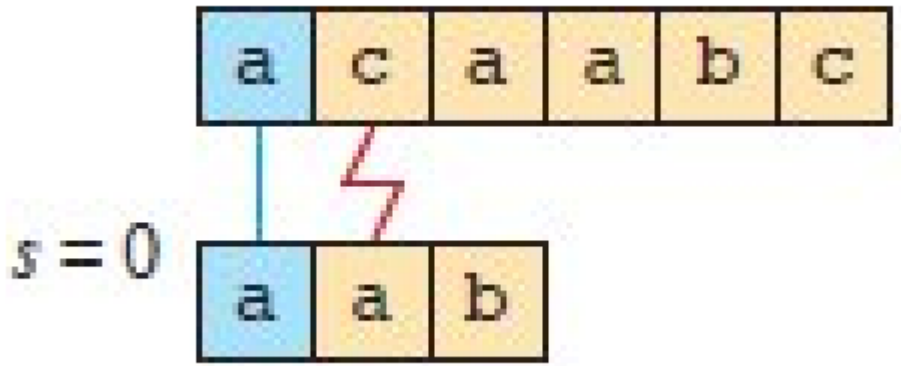
Algorytm naiwny

Algorytm „naiwny” znajduje wszystkie poprawne przesunięcia, korzystając z pętli, w której jest sprawdzany warunek P[1..m] = T[s + 1..s + m] dla każdej spośród n - m + 1 możliwych wartości s.

NAIVE-STRING-MATCHER(T, P)

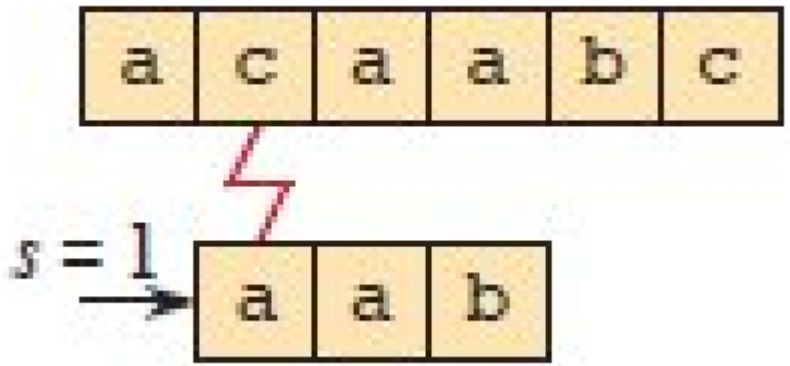
```
1  n = T.length
2  m = P.length
3  for s = 0 to n − m
4       if P[1..m] == T[s + 1..s + m]
5            print “Pattern occurs with shift” s
```

Przykład



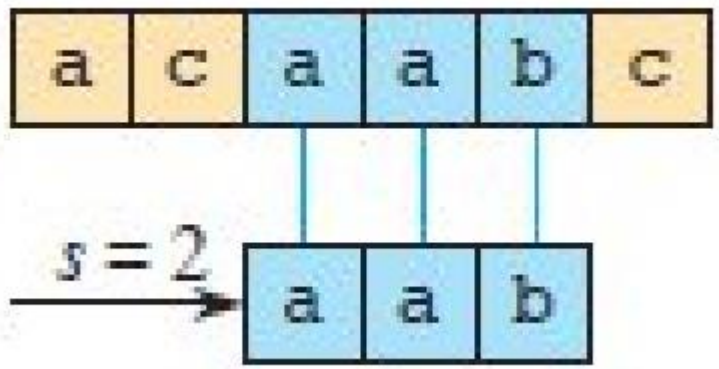
2 porównania dla s=0

Przykład



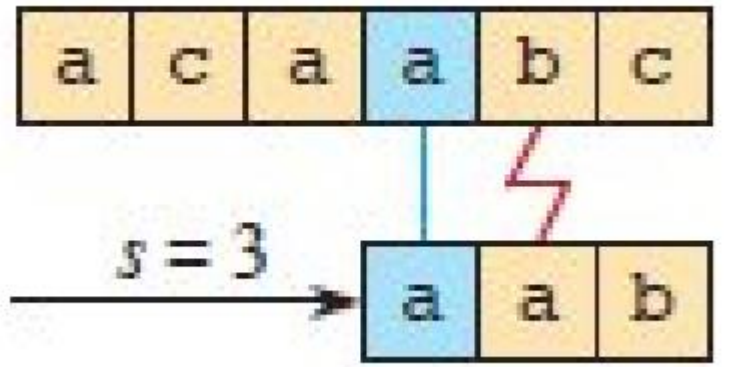
1 porównanie dla s=1

Przykład

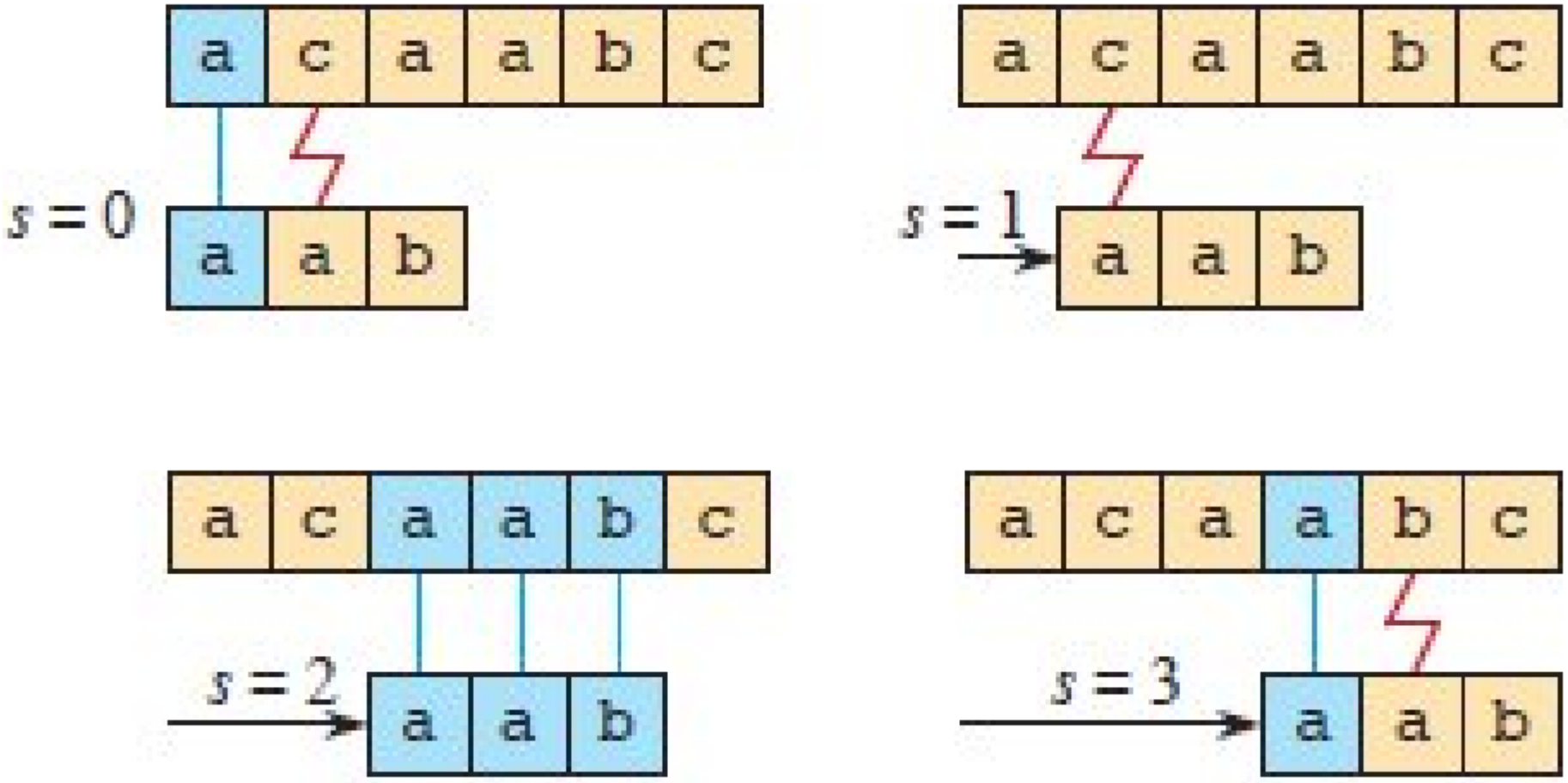


3 porównania dla s=2

Przykład



2 porównania dla s=3



W sumie 8 porównań

Złożoność

- Najgorszy przypadek:

```
   Tekst: aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaab
Wzorzec: aaaaaaaaaaaaaaaaaaaab
```

Zadanie

Pokaż jakie porównania wykonuje algorytm naiwny, szukając wzorca P = 0001 w tekście T = 000010001010001. Ile ich jest?

Algorytm Rabina-Karpa

- Algorytm Rabina-Karpa ma złożoność pesymistyczną równie złą jak algorytm naiwny.
- Algorytm Rabina-Karpa zachowuje się bardzo dobrze w praktyce, a także daje się łatwo modyfikować.

Algorytm Rabina-Karpa

- Dla uproszczenia zakładamy, że $\sum = \{0, 1, 2, \ldots, 9\}$
- W ogólnym przypadku możemy przyjąć, że każdy symbol jest cyfrą w systemie o podstawie d, gdzie $d = |\sum|$
- Tekst złożony z k symboli możemy interpretować jako liczbę k-cyfrową
- Dla wzorca P[1..m] niech p oznacza odpowiadającą mu wartość dziesiętną
- Dla tekstu T[1..n] niech t_s oznacza wartość dziesiętną podsłowa T[s+1..s+m]
- Oczywiście $t_s = p$ wtedy i tylko wtedy, gdy T[s + 1..s+m] = P[1..m]
- Potencjalny problem: p i t_s mogą być bardzo dużymi liczbami

Schemat Hornera

$$
\begin{aligned}
P(x) &= \sum_{k=0}^{n} a_k x^k \\
&= a_0 + x(a_1 + x(a_2 + \cdots + x(a_{n-1} + x a_n) \cdots))
\end{aligned}
$$

```
1   y = 0
2   for i = n downto 0
3       y = a_i + x · y
```

Algorytm Rabina-Karpa

- Znając P[1..m] możemy obliczyć p używając schematu Hornera
- Znając T[1..m] możemy obliczyć t_0 używając schematu Hornera
- W celu wyliczenia pozostałych wartości t_1, t_2, …,t_{n-m} wystarczy zauważyć, że t_{s+1} można otrzymać z t_s w stałym czasie, ponieważ

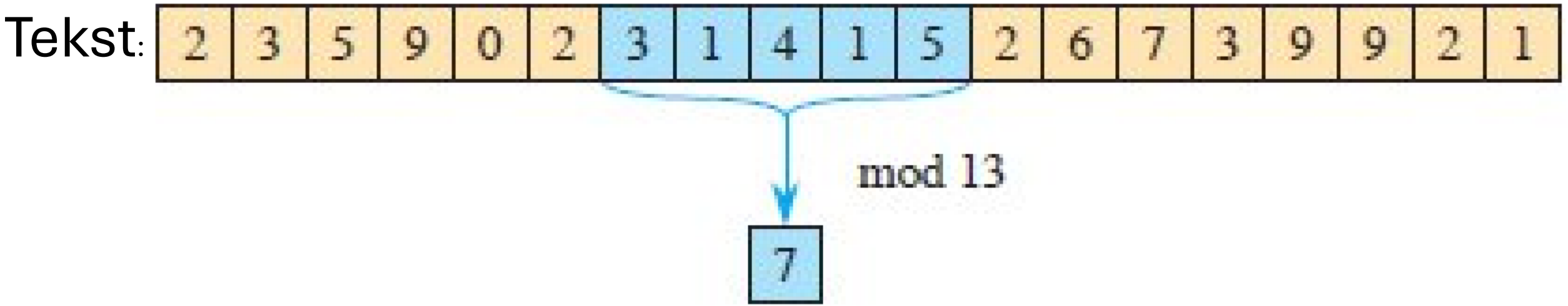
$$t_{s+1} = 10(t_s - 10^{m-1}T[s + 1]) + T[s + m + 1]$$

Innymi słowy usuwamy najbardziej znaczącą cyfrę i dopisujemy z prawej nową najmłodszą cyfrę

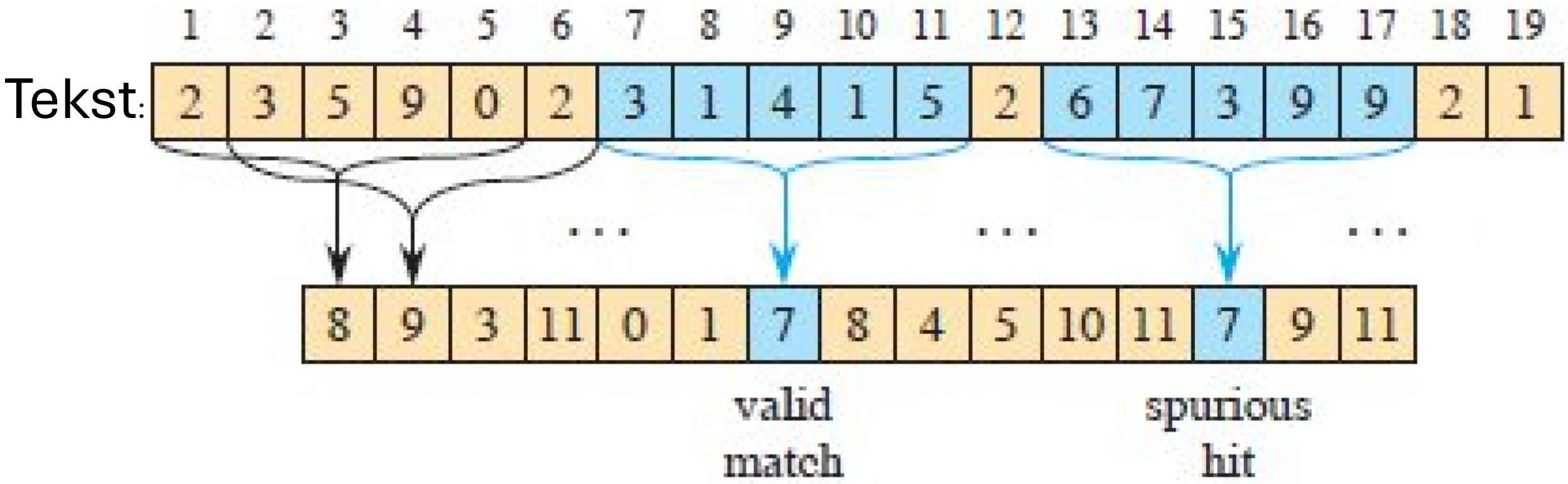
Algorytm Rabina-Karpa

- Aby uniknąć problemu dużych liczb obliczamy p i t_s modulo q dla odpowiednio dobranego q
- Działanie w arytmetyce modulo q może prowadzić teraz do błędu ze względu na to, że z kongruencji $t_s = p \pmod q$ nie wynika równość $t_s = p$.
- Z drugiej strony, jeśli $t_s \mathrel{!}= p \pmod q$, to z pewnością wiemy, że $t_s \mathrel{!}= p$ i że przesunięcie s jest niepoprawne.
- Możemy więc użyć testu $t_s = p \pmod q$ jako szybkiej heurystyki do eliminowania niepoprawnych przesunięć s
- Każde przesunięcie s dla którego $t_s = p \pmod q$ trzeba jeszcze zweryfikować, tzn. musimy sprawdzić czy s jest poprawne, czy też nastąpił fałszywy alarm

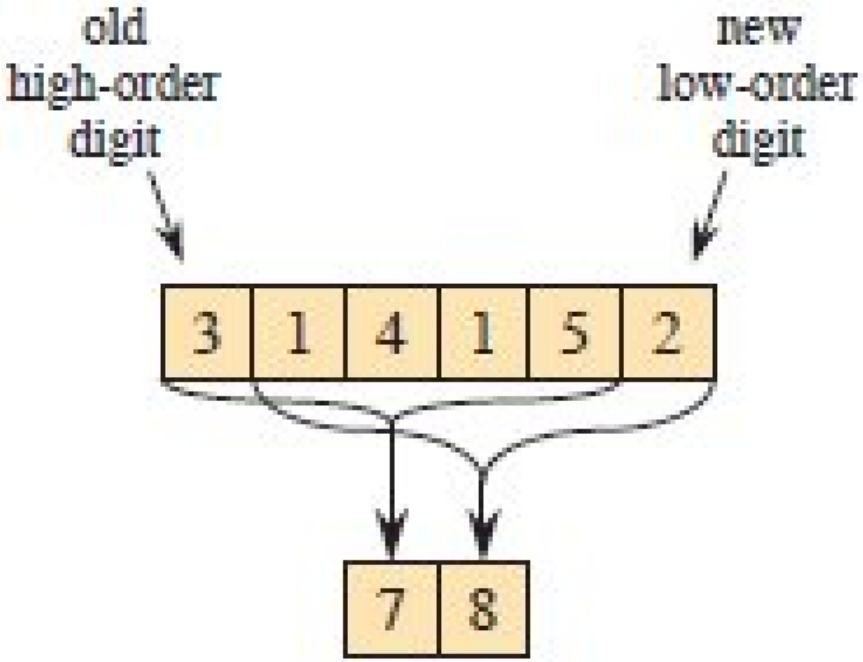
Wzorzec: 31415, mamy 31415 = 7 (mod 13)

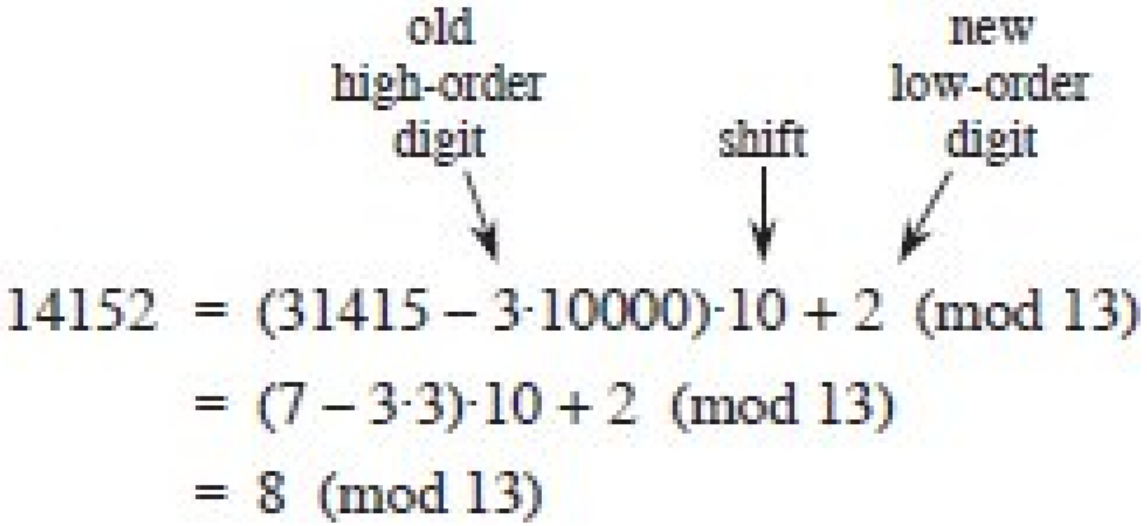


Wzorzec: 31415, mamy 31415 = 7 (mod 13)



Obliczanie t_{s+1} na podstawie t_s





```
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
```

Zadanie

Ile fałszywych alarmów wystąpi w algorytmie Rabina-Karpa podczas szukania wzorca P = 26 w tekście T = 3141592653589793, jeśli obliczenia będą wykonywane modulo q = 11?

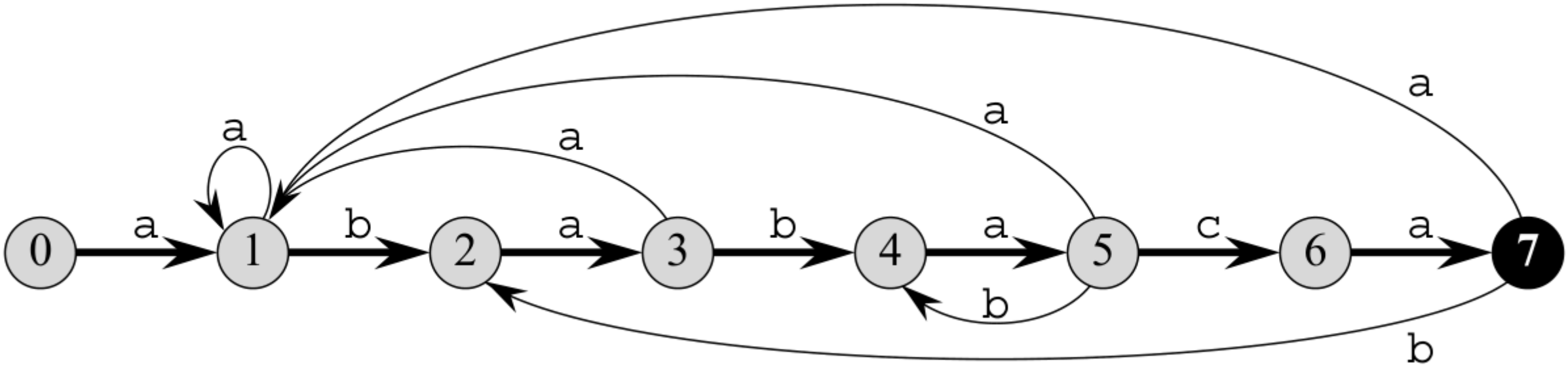
Wyszukiwanie wzorca z wykorzystaniem automatów skończonych

- Automaty szukające wzorca są bardzo efektywne: badają każdy symbol tekstu dokładnie raz, przeznaczając na pojedynczy symbol stały czas.
- Czas konstruowania automatu może być długi, gdy alfabet jest duży.

Prefiks, sufiks

- Mówimy, że słowo w jest **prefiksem** słowa x, gdy x = wy dla pewnego słowa y, np. ab jest prefiksem abcca
- Mówimy, że słowo w jest **sufiksem** słowa x, gdy x = yw dla pewnego słowa y, np. cca jest sufiksem abcca

Automat dla wzorca ababaca



Automat akceptujący wszystkie teksty,
których sufiksem jest wzorzec ababaca.

Krawędzie prowadzące do stanu 0 zostały pominięte; przyjmujemy konwencję, że jeśli w stanie i brakuje krawędzi wychodzącej z etykietą a dla pewnego $a \in \Sigma$, to $\delta(i, a) = 0$

Mając dany automat łatwo wyszukać wzorzec

```
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
```

Funkcja sufiksowa

- **Definiujemy σ(x) jako długość najdłuższego prefiksu wzorca P, który jest jednocześnie sufiksem x**
- Przykładowo dla $P = ab$ mamy:
  - $\sigma(ccaca) = 1$
  - $\sigma(ccab) = 2$
- Dla wzorca P długości m mamy

$\sigma(x) = m \leftrightarrow$ gdy P jest sufiksem x

Definicja automatu

- Zbiorem stanów jest Q = {0,1,2,…,m}
- Stanem początkowym jest 0
- Jedynym stanem akceptującym jest m
- Funkcja przejść δ dla każdego stanu q i symbolu a jest zdefiniowana równaniem $\delta(q, a) = \sigma(P[1..q]a)$
- Niezmiennik automatu: po wczytaniu pierwszych i symboli tekstu T automat jest w stanie q, gdzie q jest długością najdłuższego sufiksu T[1..i], który jest jednocześnie prefiksem wzorca P

Przykład

Znajdziemy automat dla wzorca $P = \text{ababaca}$

Ściąga: $\delta(q, a) = \sigma(P[1..q]a)$

Rozwiązanie dla $P = \text{ababaca}$

$\delta(0, a) = \sigma(a) = 1$
$\delta(0, b) = \sigma(b) = 0$
$\delta(0, c) = \sigma(c) = 0$

$\delta(1, a) = \sigma(aa) = 1$
$\delta(1, b) = \sigma(ab) = 2$
$\delta(1, c) = \sigma(ac) = 0$

$\delta(2, a) = \sigma(aba) = 3$
$\delta(2, b) = \sigma(abb) = 0$
$\delta(2, c) = \sigma(abc) = 0$

$\delta(3, a) = \sigma(abaa) = 1$
$\delta(3, b) = \sigma(abab) = 4$
$\delta(3, c) = \sigma(abac) = 0$

$\delta(4, a) = \sigma(ababa) = 5$
$\delta(4, b) = \sigma(ababb) = 0$
$\delta(4, c) = \sigma(ababc) = 0$

$\delta(5, a) = \sigma(ababaa) = 1$
$\delta(5, b) = \sigma(ababab) = 4$
$\delta(5, c) = \sigma(ababac) = 6$

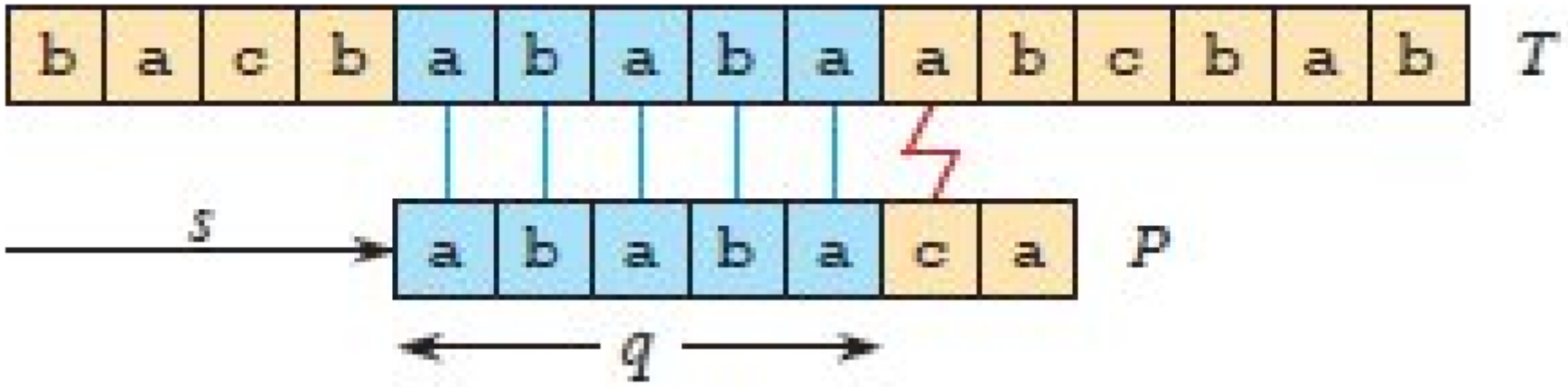
$\delta(6, a) = \sigma(ababaca) = 7$
$\delta(6, b) = \sigma(ababacb) = 0$
$\delta(6, c) = \sigma(ababacc) = 0$

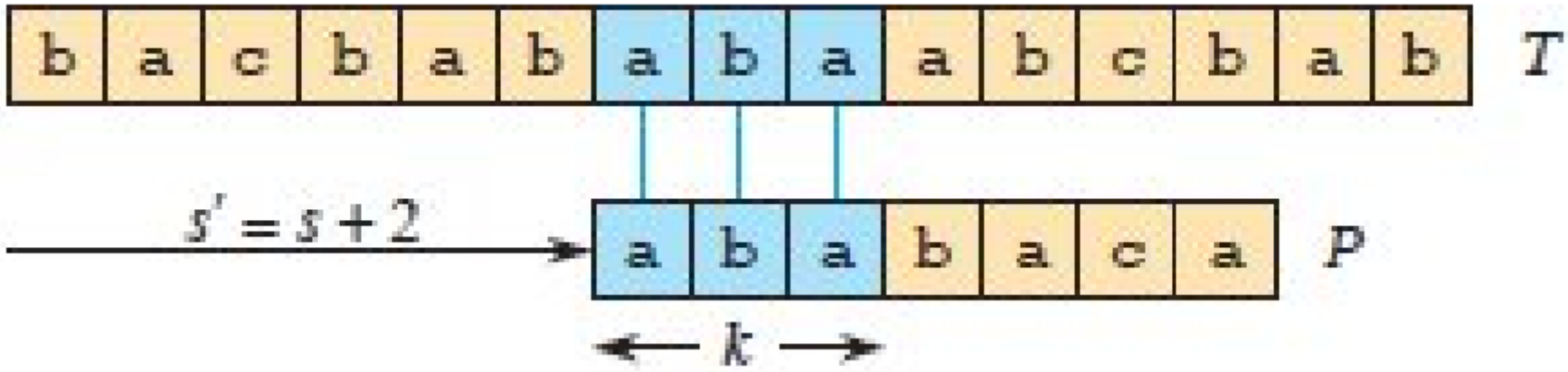
$\delta(7, a) = \sigma(ababacaa) = 1$
$\delta(7, b) = \sigma(ababacab) = 2$
$\delta(7, c) = \sigma(ababacac) = 0$

Zadania

1. Skonstruuj automat wyszukiwania wzorca P = aabab i przetestuj jego działanie dla tekstu T = aaababaabaababababaab.
2. Narysuj diagram przejść automatu wyszukiwania wzorca ababbabbababbababbabb nad alfabetem {a, b}.

Algorytm Knutha-Morrisa-Pratta





Funkcja prefiksowa

Funkcję prefiksową π: $\{1, 2, \dots, m\} \to \{0, 1, \dots, m-1\}$ definiujemy jako:

$$\pi[q] = \max\{k < q:\ P[1..k] \text{ jest sufiksem } P[1..q]\}$$

Innymi słowy:

$\pi[q]$ jest maksymalną długością właściwego prefikso-sufiksu słowa P[1..q]

Słownik:

właściwy – różny od całego słowa
prefikso-sufiks – prefiks, który jest jednocześnie sufiksem

Przykład: ababaca

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

Prefiks wzorca	*max pref-suf*	*długość max pref-suf*	
a	słowo puste	0	
ab	słowo puste	0	
aba	a	1	
abab	ab	2	
ababa	aba	3	// pref i suf nachodzą na siebie!
ababac	słowo puste	0	
ababaca	a	1	

```
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
```

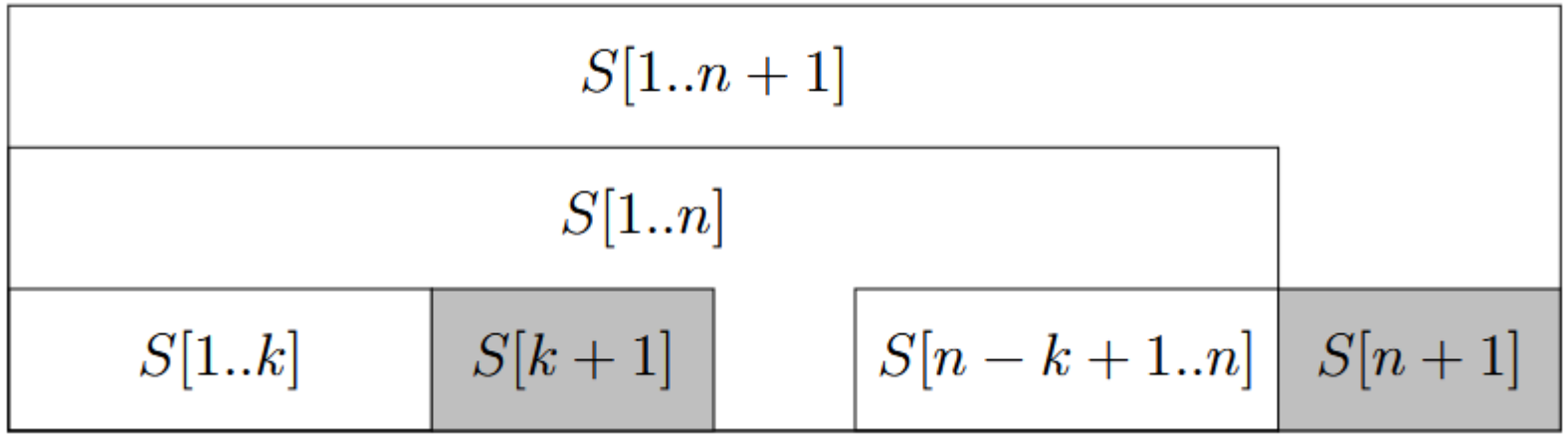
KMP-MATCHER(T, P)

```
1   n = T.length
2   m = P.length
3   π = COMPUTE-PREFIX-FUNCTION(P)
4   q = 0
5   for i = 1 to n
6       while q > 0 and P[q + 1] ≠ T[i]
7           q = π[q]
8       if P[q + 1] == T[i]
9           q = q + 1
10      if q == m
11          print "Pattern occurs with shift"
12          q = π[q]
```

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
```

Rozszerzenie pref-suf z S[1..n] na S[1..n+1]



Przykład dla P = abacabab

a

słowo puste

$$\pi[1] = 0$$

Przykład dla P = abacabab

ab

Próbujemy rozszerzyć słowo puste, czyli sprawdzamy, czy a = b?

Ponieważ to nieprawda, więc rozszerzyć się nie da i prefikso-sufiksem dalej jest słowo puste

$$\pi[2] = 0$$

Przykład dla P = abacabab

aba

Próbujemy rozszerzyć słowo puste, czyli sprawdzamy, czy a = a?

Ponieważ to prawda, więc prefiksem-sufiksem jest słowo a

$$\pi[3] = 1$$

Przykład dla P = abacabab

abac

Próbujemy rozszerzyć a, czyli sprawdzamy czy b=c. To nieprawda, więc próbujemy rozszerzyć słowo puste, które jest prefikso-sufiksem słowa a. Sprawdzamy, czy a=c. To również nieprawda, czyli prefikso-sufiksem jest słowo puste

$$\pi[4] = 0$$

Przykład dla P = abacabab

abaca

Próbujemy rozszerzyć słowo puste. Ponieważ a=a więc prefikso-sufiksem jest a

$$\pi[5] = 1$$

Przykład dla P = abacabab

abacab

Próbujemy rozszerzyć a. Ponieważ b=b, więc prefikso-sufiksem jest ab

$$\pi[6] = 2$$

Przykład dla P = abacabab

abacaba

Próbujemy rozszerzyć słowo ab. Ponieważ a=a, więc prefikso-sufiksem jest aba

$$\pi[7] = 3$$

Przykład dla P = abacabab

abacabab

Próbujemy rozszerzyć aba. Ponieważ c != b więc jest to niemożliwe.

Następnie bierzemy maksymalny prefikso-sufiks słowa aba, jest to a (jak wiadomo z poprzednich obliczeń) i próbujemy go rozszerzyć.

Ponieważ b=b, to się udaje i prefikso-sufiksem jest ab

$$\pi[7] = 2$$

Inne spojrzenie na algorytm KMP

- Dane: wzorzec P, tekst T.
- Zakładamy, że symbol $ nie należy do P i T
- Tworzymy słowo P$T
- Wyznaczamy funkcję π dla słowa P$T
- Przykład

P = ab T = baba

P$T = ab$baba

i	1234567
pi	0000121

Zadanie

Wyznacz funkcję prefiksową π dla wzorca ababbabbabbababbabb

Koniec ☺