Algorytmy i struktury danych

Kodowanie Huffmana, algorytmy tekstowe

Krzysztof M. Ocetkiewicz
Krzysztof.Ocetkiewicz@eti.pg.edu.pl

Katedra Algorytmów i Modelowania Systemów, WETI, PG

Kodowanie Huffmana

- jedna z bardziej skutecznych metod bezstratnej kompresji danych
- wykorzystywana jako jeden z kroków w bardziej zaawansowanych algorytmach kompresji
- liczba bitów przypadająca na symbol nie jest stała — zależy od ilości wystąpień symbolu w napisie (im większa, tym mniejsza liczba bitów)
- kody prefiksowe — żaden z kodów nie jest prefiksem innego

Kodowanie Huffmana

- kompresja polega na wyznaczeniu, dla każdego symbolu wejściowego, ciągu bitów, które go reprezentują
- ciągi te wyznaczamy budując drzewo binarne z symbolami w liściach
- kod wyznaczony jest przez ścieżkę od korzenia do symbolu — przejście w lewo to 0, przejście w prawo to 1
- drzewo budujemy wybierając w każdym kroku dwa węzły o najmniejszej liczbie wystąpień symbolu i łącząc je w jeden

Kody Huffmana — budowa drzewa

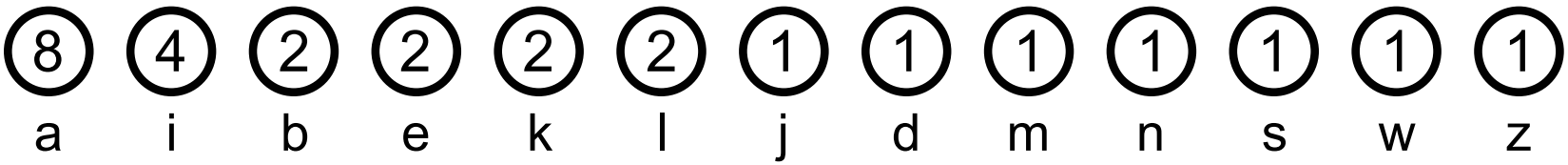
1: $n =$ liczba symboli
2: włóż do kolejki priorytetowej (minimalizującej) Q wszystkie symbole, przy czym kluczem jest ilość wystąpień symbolu w napisie wejściowym
3: **for** $i = 1, \ldots, n-1$ **do**
4: $w =$ Create-Node()
5: $w.left =$ Extract-Min(Q)
6: $w.right =$ Extract-Min(Q)
7: $w.klucz = w.left.klucz + w.right.klucz$
8: Insert(Q, w)
9: **end for**
10: **return** Extract-Min(Q)

Kody Huffmana — przykład

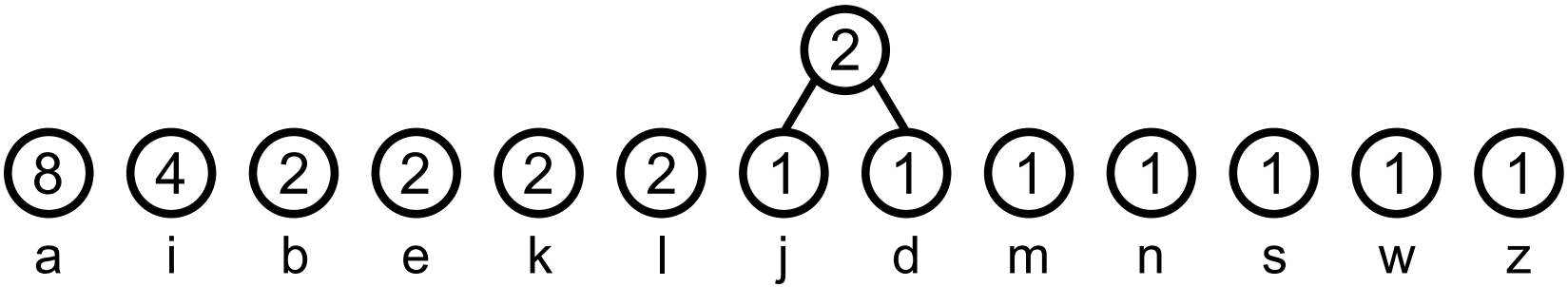
- np.: sialababamakniewiedzialajak
- częstotliwości wystąpień liter:

a	8	d	1
i	4	m	1
b	2	n	1
e	2	s	1
k	2	w	1
l	2	z	1
j	1		

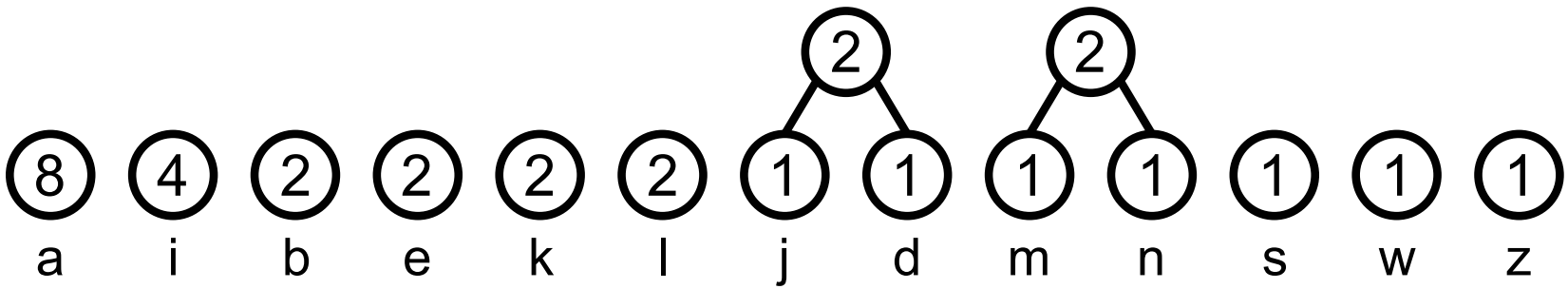
Kody Huffmana — przykład



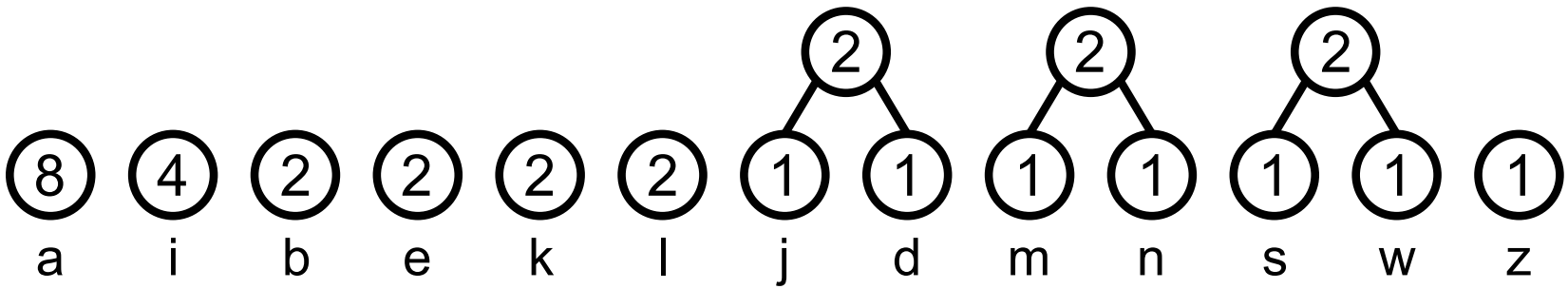
Kody Huffmana — przykład



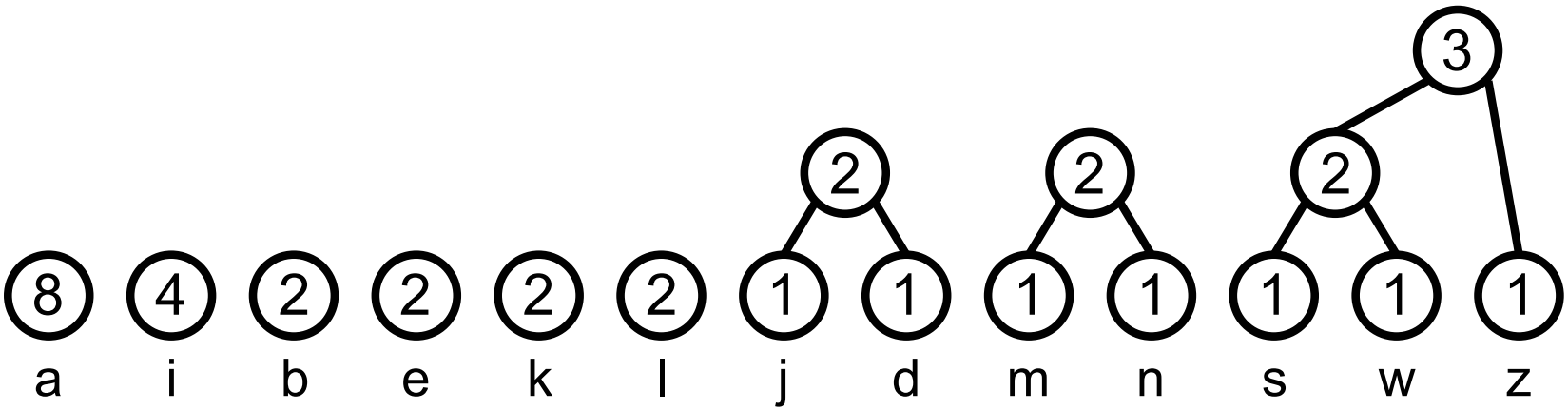
Kody Huffmana — przykład



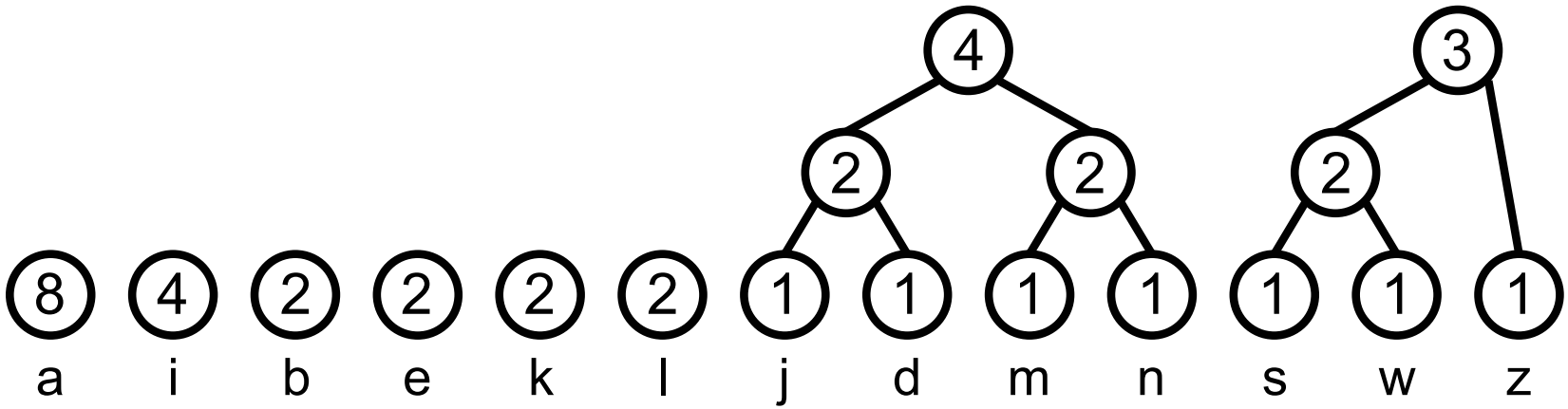
Kody Huffmana — przykład



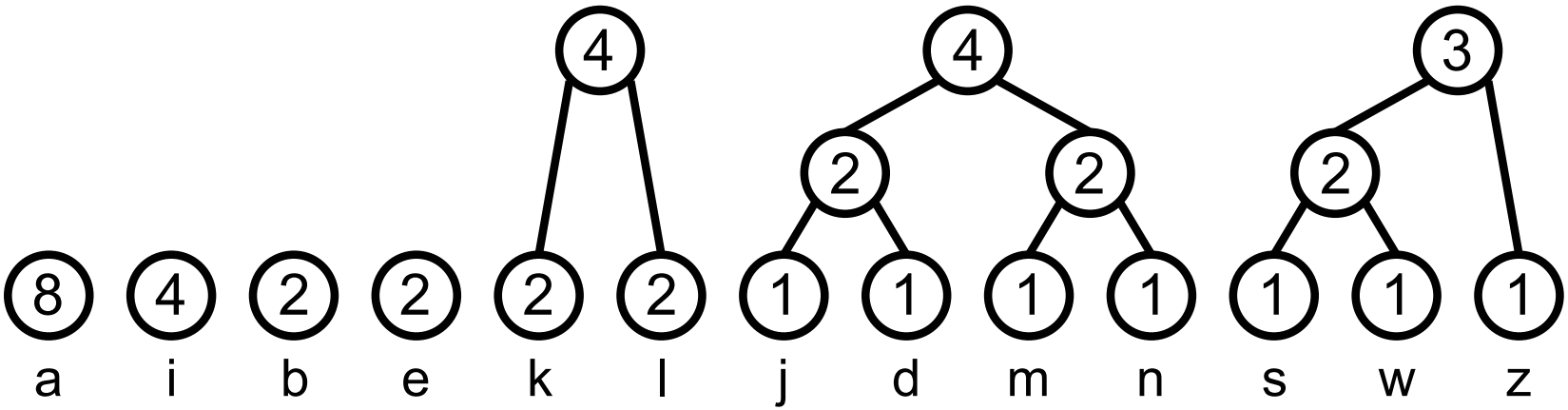
Kody Huffmana — przykład



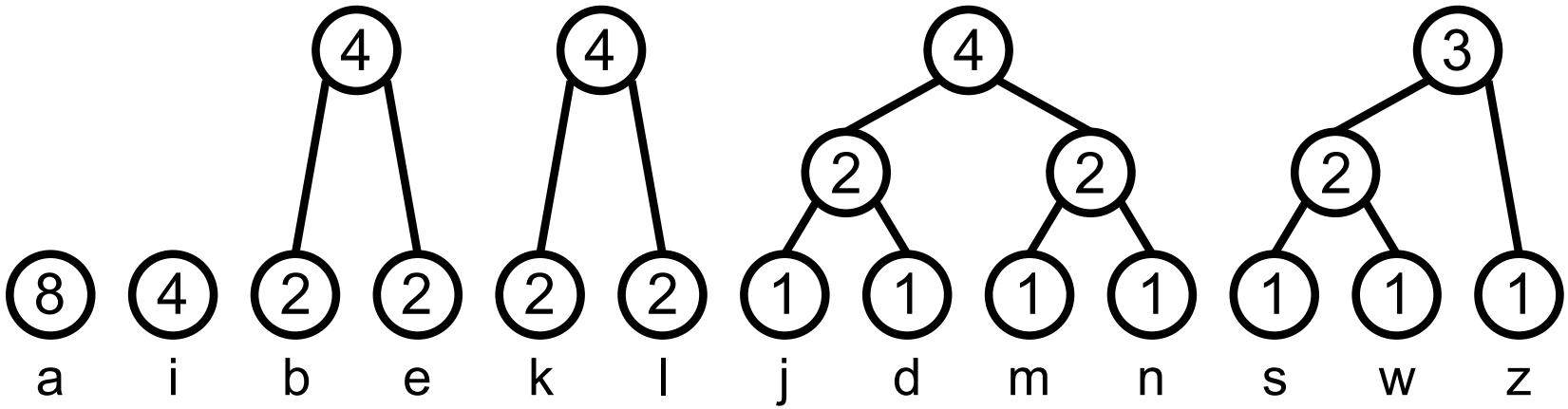
Kody Huffmana — przykład



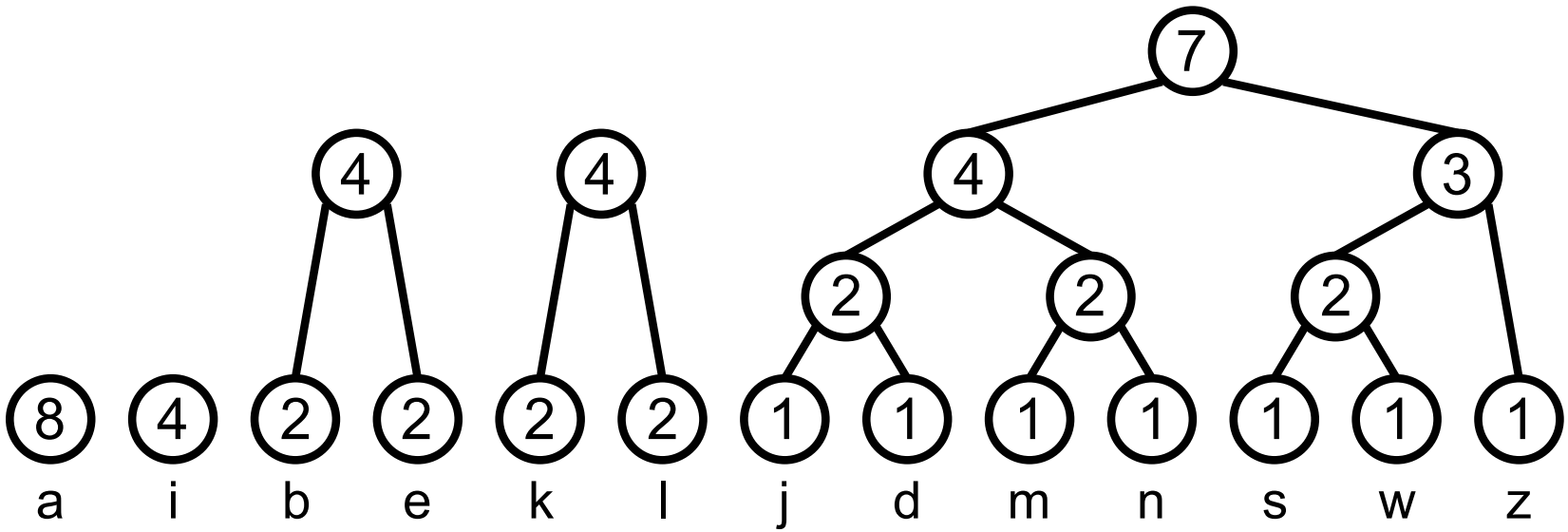
Kody Huffmana — przykład



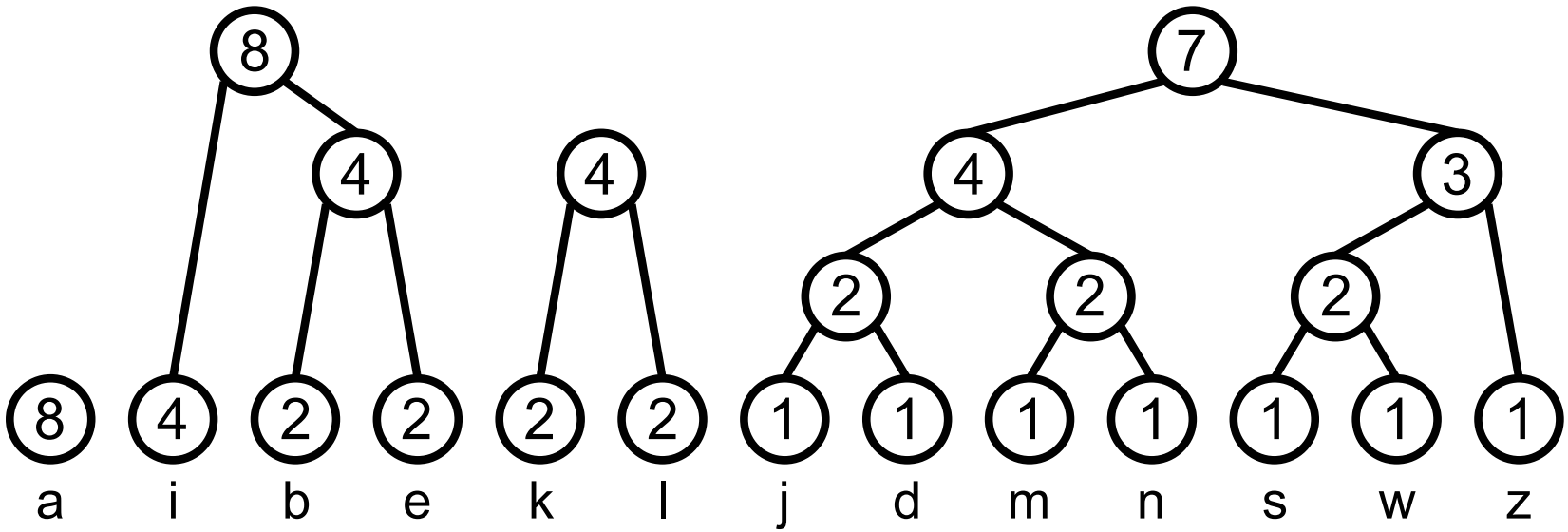
Kody Huffmana — przykład



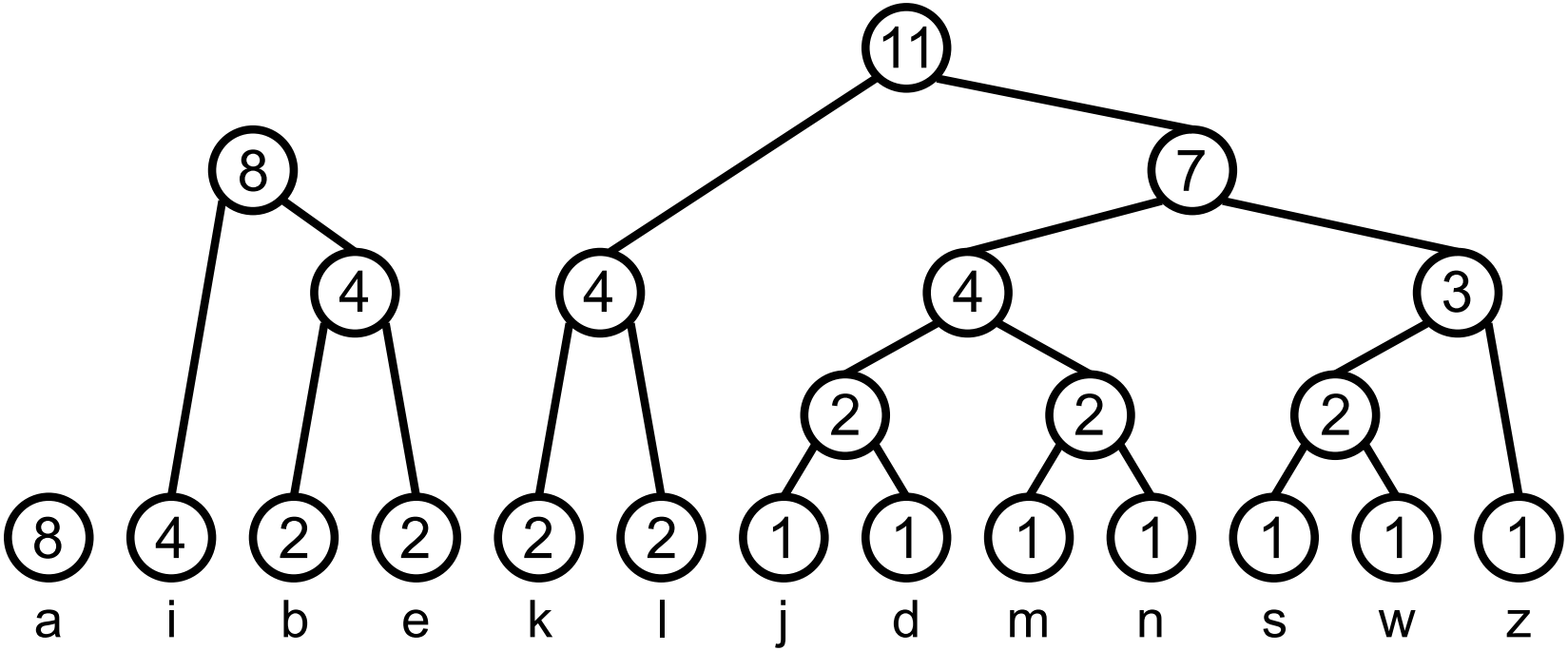
Kody Huffmana — przykład



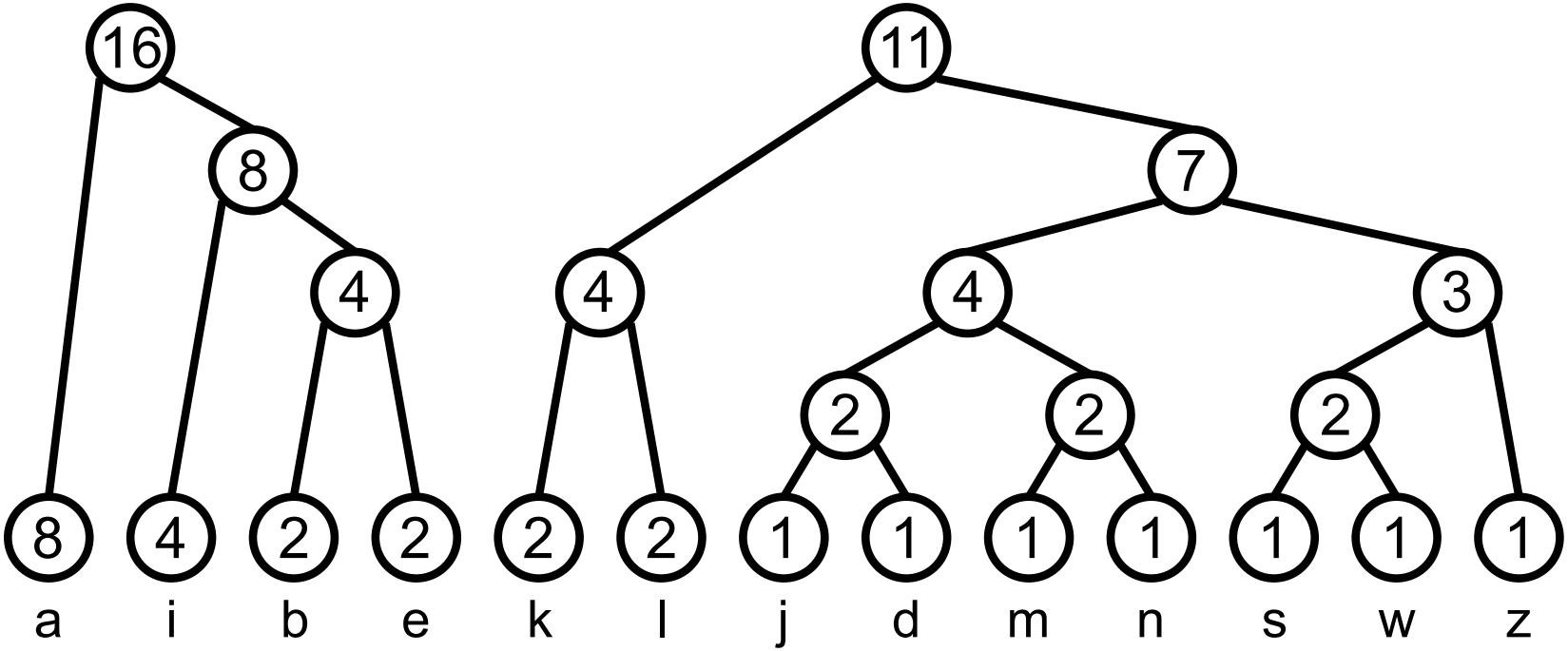
Kody Huffmana — przykład



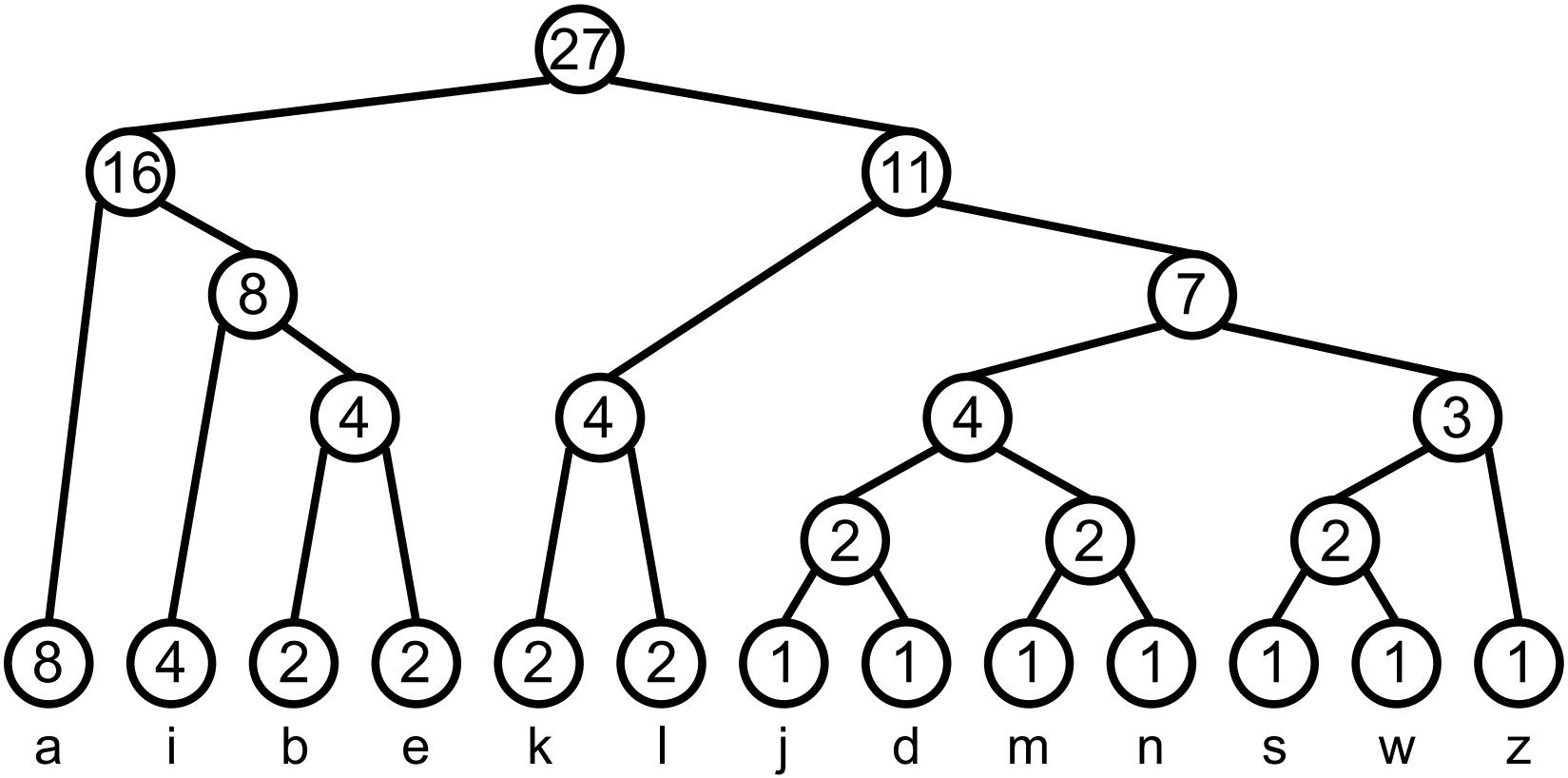
Kody Huffmana — przykład



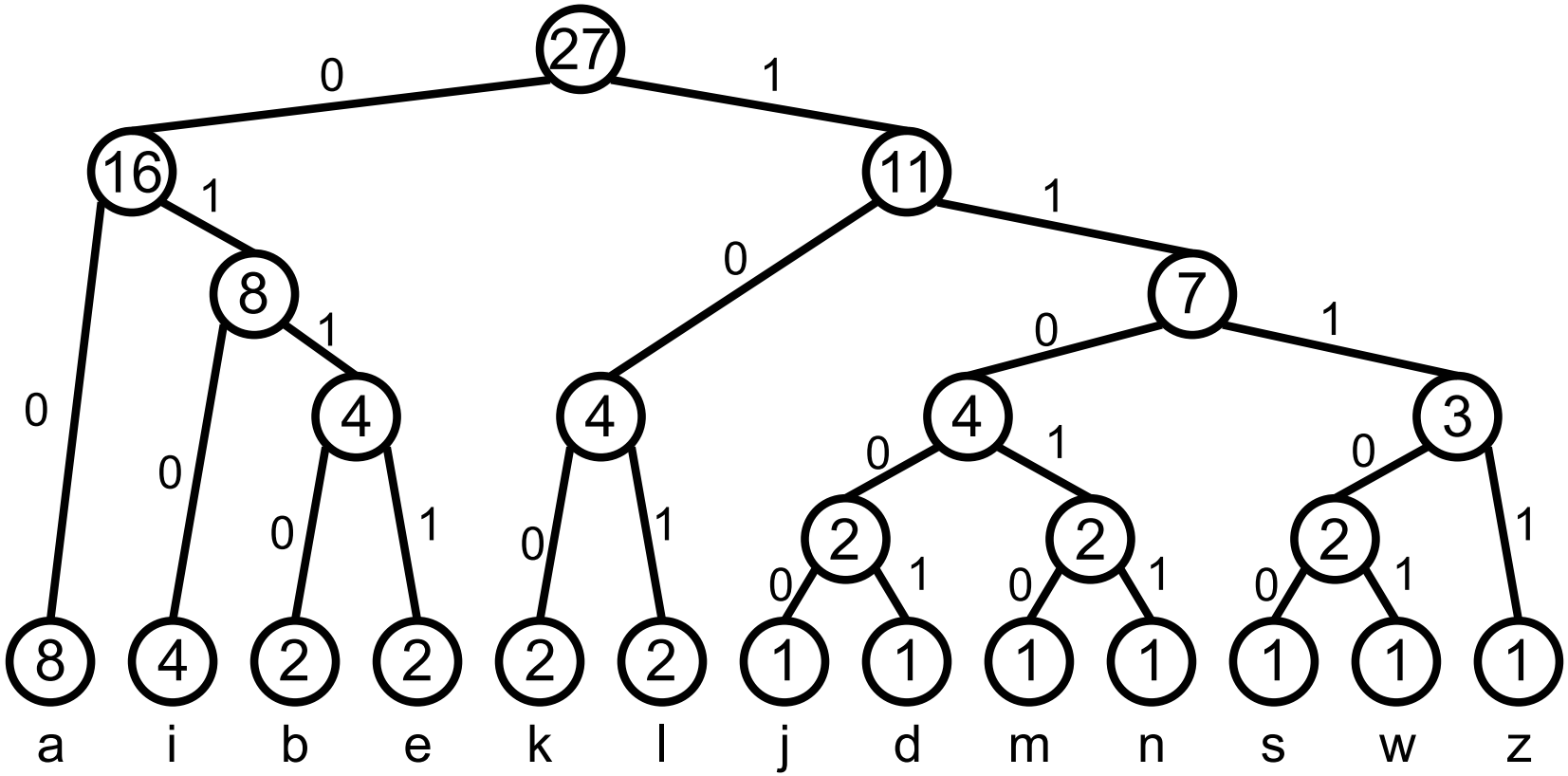
Kody Huffmana — przykład



Kody Huffmana — przykład



Kody Huffmana — przykład



Kody Huffmana — przykład

- kody liter:

a	00	d	11001
i	010	m	11010
b	0110	n	11011
e	0111	s	11100
k	100	w	11101
l	101	z	1111
j	11000		

Kody Huffmana — przykład

- przed kompresją (5 bitów na znak) — 135 bitów:
  100110100100001011000000100010000010001000001
  011010000101011011100100100101101110100100101
  001001101001001000010110000001010100000101011
- po kompresji — 90 bitów:
  111000100010100011000011000110100010011011010
  011111101010011111001111101000101001100000100

Kody Huffmana — dekodowanie

- dekodowanie polega na wczytywaniu kolejnych bitów
  i przechodzeniu po zbudowanym drzewie — bit 0 to przejście w lewo,
  1 — w prawo
- gdy osiągniemy symbol, wypisujemy go na wyjście
  i wracamy do korzenia

Algorytmy tekstowe

- alfabet to zbiór wszystkich dostępnych symboli (litery, cyfry, znaki przestankowe, znaki specjalne)
- słowo to ciąg symboli alfabetu
- słowem jest np. “ala ma kota” – ma ono 11 znaków: ‘a’, ‘l’, ‘a’, ‘ ’, ‘m’, ‘a’, ‘ ’, ‘k’, ‘o’, ‘t’, ‘a’
- alfabetem nie muszą być litery, słowami wyrazy czy napisy: równie dobrze alfabetem może być zbiór możliwych wartości próbki dźwięku, a słowem – nagranie, alfabetem może być zbiór wszystkich możliwych kolorów jednego piksela a wyrazem obraz itp.

Algorytmy tekstowe

- długość słowa – liczba znaków w słowie
- $T[i]$ – i-ty symbol słowa T
- prefiks słowa – początkowy fragment słowa – dla słowa S długości n prefiksami są słowa $S[1, \ldots, i]$ dla $i = 1, \ldots, n$
- sufiks słowa – końcowy fragment słowa – dla słowa S długości n sufiksami są słowa $S[n - i, \ldots, n]$ dla $i = 0, \ldots, n - 1$
- dla uproszczenia niektórych algorytmów czasem rozszerzamy alfabet o pewne specjalne symbole (np. “koniec wyrazu”, “przerwa” itp.)

Minimalna odległość edycyjna

- dane są dwa słowa A i B
- dozwolone są operacje: wstawienie symbolu, usunięcie symbolu, zamiana symbolu na inny
- ile minimalnie operacji należy wykonać, aby przekształcić słowo A w słowo B
- problem ten możemy rozwiązać stosując programowanie dynamiczne

Minimalna odległość edycyjna

- rozważmy prefiks słowa A długości i i prefiks słowa B długości j
- minimalna odległość edycyjna pomiędzy $A[1,\ldots,i]$ a $B[1,\ldots,j]$ to mniejsza z:
- $OE(A[1,\ldots i-1], B[1,\ldots j]) + 1$ – usunięcie znaku
- $OE(A[1,\ldots i], B[1,\ldots j-1]) + 1$ – dołączenie znaku
- $OE(A[1,\ldots i-1], B[1,\ldots j-1])$ jeżeli $A[i] = B[j]$
- $OE(A[1,\ldots i-1], B[1,\ldots j-1]) + 1$ jeżeli $A[i] \neq B[j]$ – zamiana znaku

Minimalna odległość edycyjna

1: K - tablica o wymiarach $m+1 \times n+1$
2: **for** $i = 0, \ldots, m$ **do** $K[i, 0] = i$
3: **for** $j = 0, \ldots, n$ **do** $K[0, j] = j$
4: **for** $i = 1, \ldots n$ **do**
5: **for** $j = 1, \ldots m$ **do**
6: **if** $A[i] = B[j]$ **then**
7: $K[i, j] = K[i-1, j-1]$
8: **else**
9: $$K[i, j] = \min \begin{cases} K[i-1, j] + 1, \\ K[i, j-1] + 1, \\ K[i-1, j-1] + 1 \end{cases}$$
10: **end if**
11: **end for**
12: **end for**
13: **return** $K[m, n]$

Wyszukiwanie wzorca

- mamy dane dwa słowa T i W (wzorzec)
- mamy sprawdzić, czy W jest podsłowem T, tzn. czy istnieje takie k, że $W[1+i] = T[k+i]$ dla $i = 0, \ldots m$ gdzie m to długość słowa W

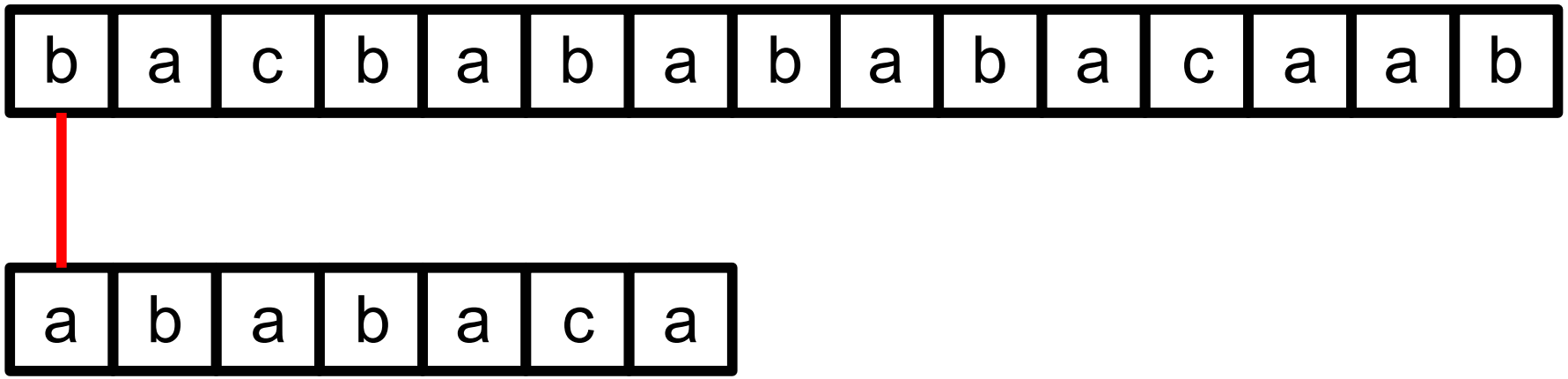
Algorytm naiwny

- najprostszy, "oczywisty" algorytm: dla każdego k, sprawdź, czy W jest podsłowem T rozpoczynając od pozycji k
- złożoność $O(nm)$

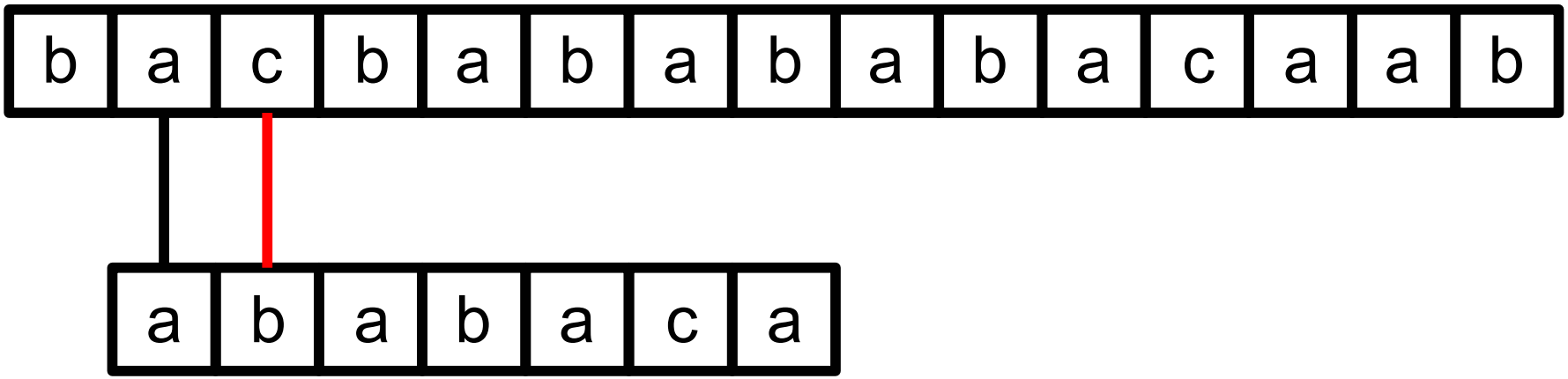
Algorytm naiwny

1: n - długość T
2: m - długość W
3: **for** $k = 1, \ldots n - m + 1$ **do**
4: **for** $i = 0, \ldots m - 1$ **do**
5: **if** $T[k + i] \neq W[i + 1]$ **then break**
6: **end for**
7: **if** $i = m$ **then**
8: znaleziono rozpoczynając od pozycji k
9: **end if**
10: **end for**

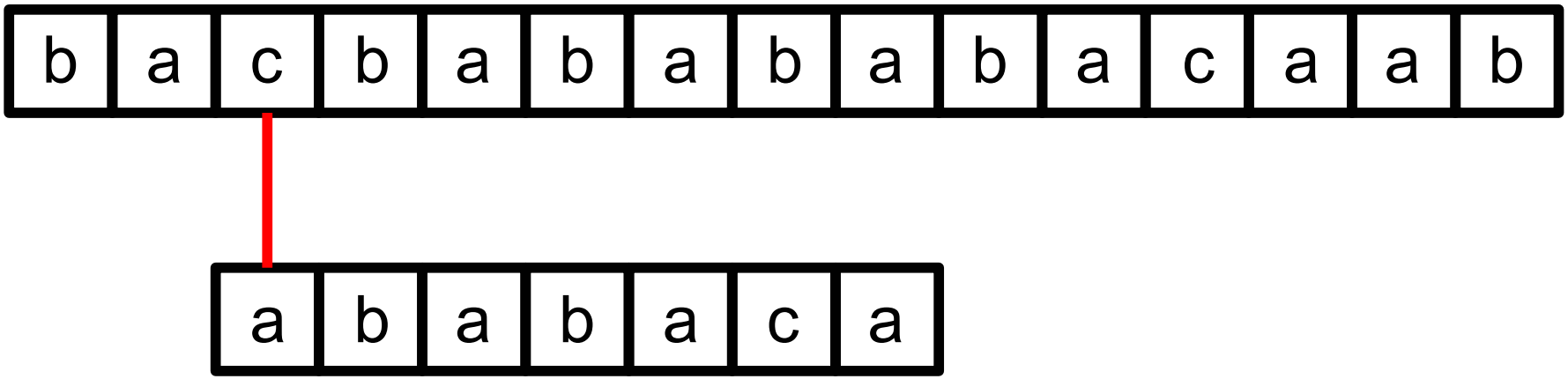
Algorytm naiwny



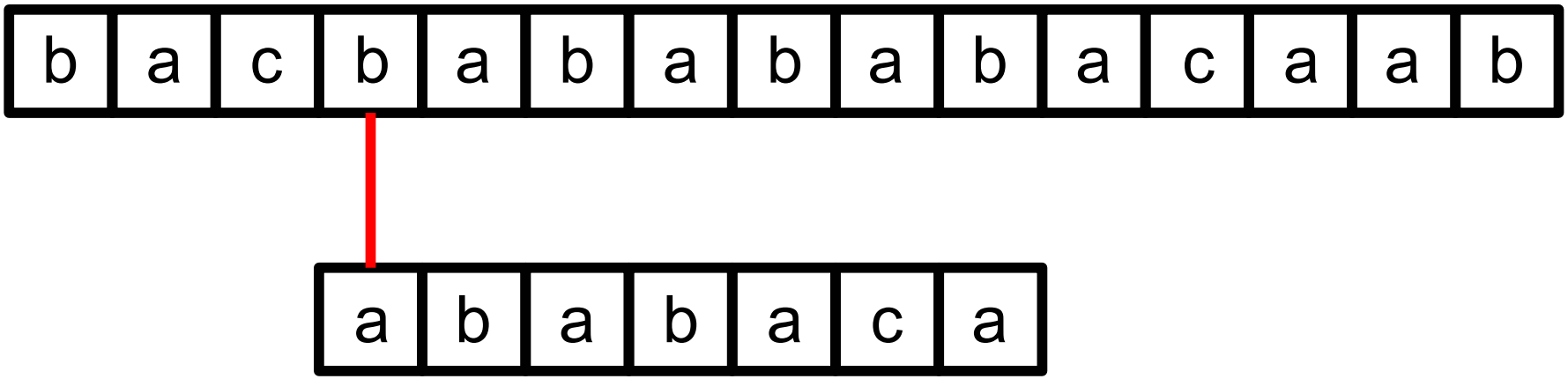
Algorytm naiwny



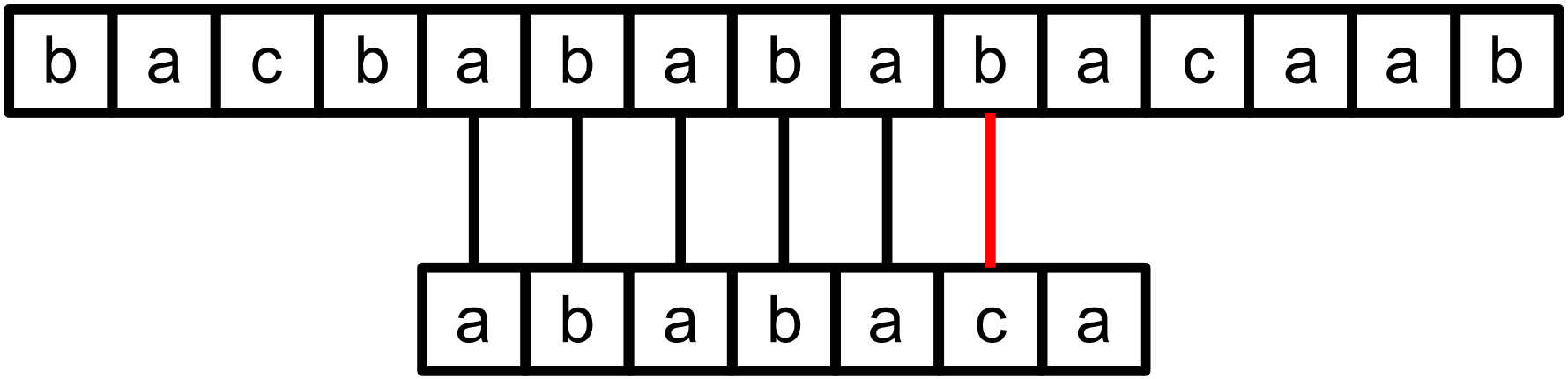
Algorytm naiwny



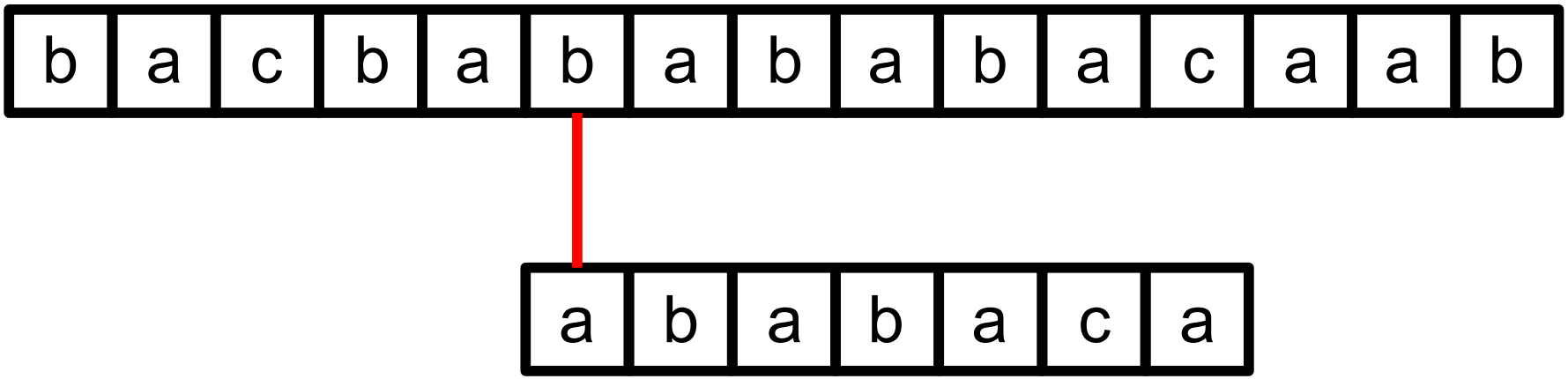
Algorytm naiwny



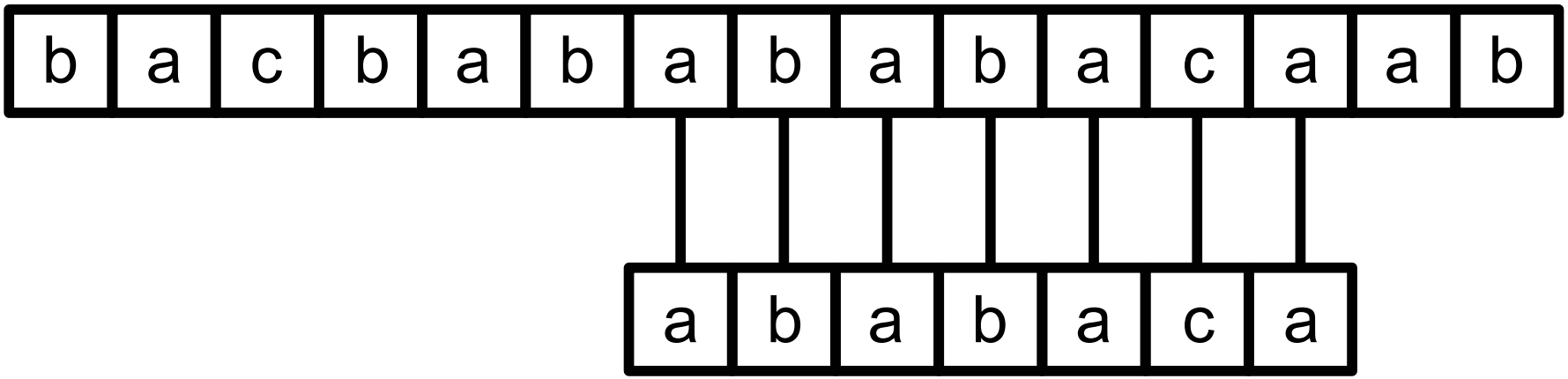
Algorytm naiwny



Algorytm naiwny



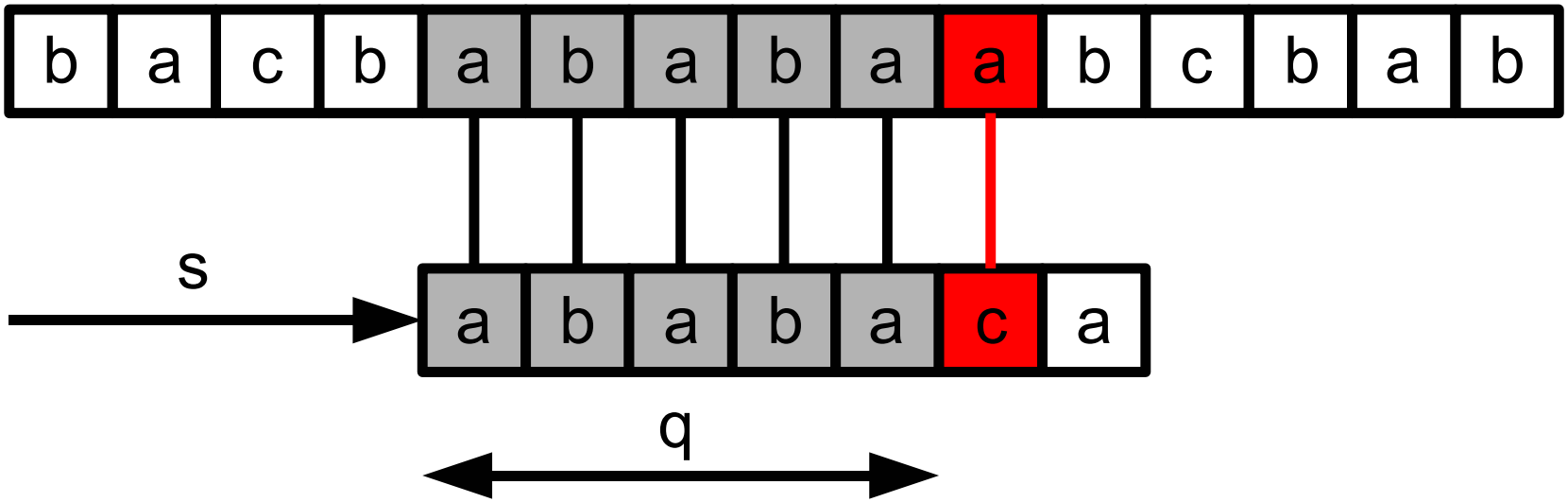
Algorytm naiwny

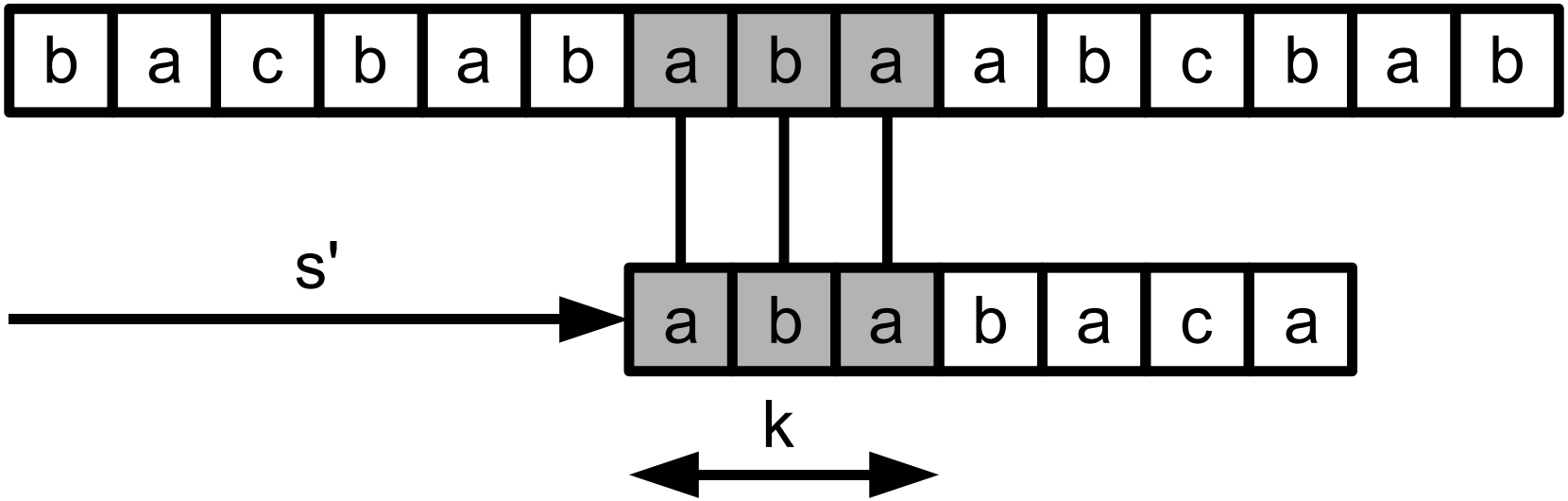


Algorytm KMP

- złożoność $O(n+m)$
- korzysta z pomocniczej tablicy π o m komórkach
- tablica ta pomaga eliminować niektóre próby dopasowania wzorca
- zakładając, że symbole wzorca $W[1,\ldots,q]$ pasują do tekstu $T[s+1,\ldots,s+q]$, jakie jest najmniejsze przesunięcie $s' > s$ takie, że: $W[1,\ldots k] = T[s'+1,\ldots s'+k]$, gdzie $s'+k = s+q$

Algorytm KMP





Algorytm KMP

- takie przesunięcie s' jest następnym przesunięciem (większym niż s), które może okazać się poprawnym zgodnie z naszą wiedzą o $T[s+1, \ldots, s+q]$
- w najlepszym wypadku będzie to $s' = s + q$ i wyeliminujemy $q - 1$ niepotrzebnych dopasowań
- dla nowego przesunięcia s' nie musimy sprawdzać pierwszych k symboli wzorca – wiemy, że pasują one do słowa dla nowego przesunięcia
- informacja ta może być obliczona porównując wzorzec ze sobą samym

Algorytm KMP

- $T[s'+1, \ldots, s'+k]$ jest częścią znanego segmentu tekstu – jest sufiksem $W[1, \ldots, q]$
- jakie jest największe $k < q$ takie, że $W[1 \ldots k]$ jest sufiksem $W[1 \ldots q]$
- następnym potencjalnie poprawnym przesunięciem jest więc $s' = s + (q - k)$
- tablica π pamięta funkcję prefiksową: $\pi[q] = \max\{k : k < q \text{ i } W[1, \ldots k] \text{ jest sufiksem } W[1, \ldots q]\}$

Obliczenie funkcji prefiksowej

1: m - długość W
2: $\pi[1] = 0$
3: $k = 0$
4: **for** $q = 2, \ldots m$ **do**
5: **while** $k > 0$ **i** $W[k+1] \neq W[q]$ **do**
6: $k = \pi[k]$
7: **end while**
8: **if** $W[k+1] = W[q]$ **then**
9: $k = k + 1$
10: **end if**
11: $\pi[q] = k$
12: **end for**

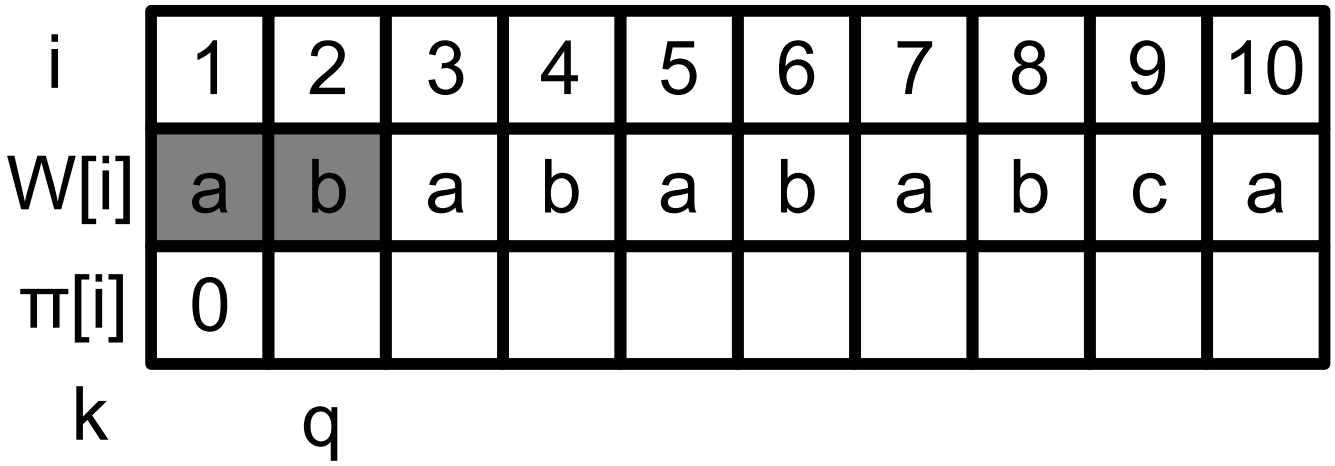
Funkcja prefiksowa

i	1	2	3	4	5	6	7	8	9	10
W[i]	a	b	a	b	a	b	a	b	c	a
π[i]	0	0	1	2	3	4	5	6	0	1

Funkcja prefiksowa

i	1	2	3	4	5	6	7	8	9	10
W[i]	a	b	a	b	a	b	a	b	c	a
π[i]	0									
k		q								

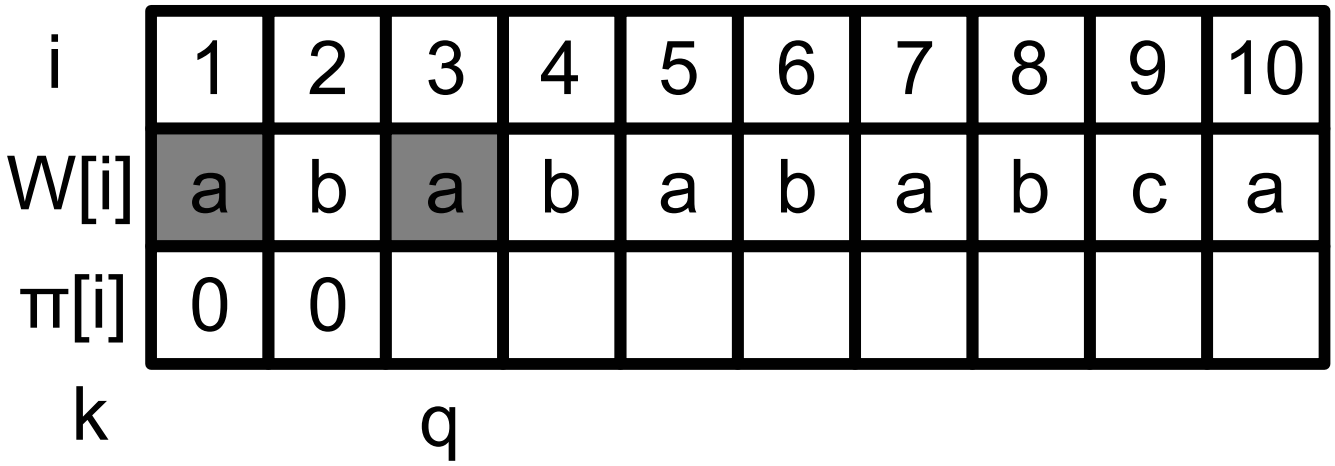
Funkcja prefiksowa



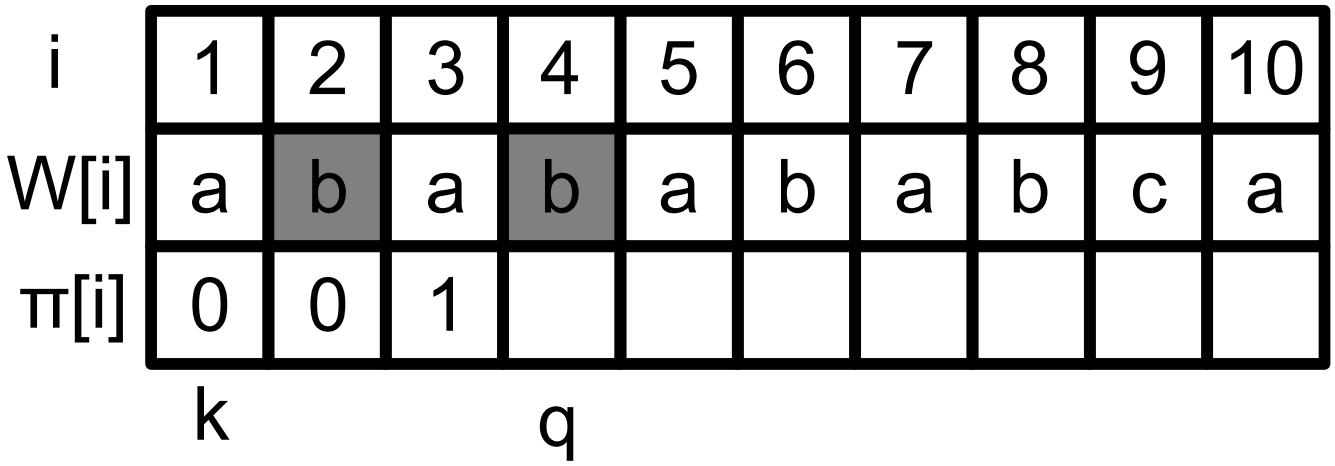
Funkcja prefiksowa

i	1	2	3	4	5	6	7	8	9	10
W[i]	a	b	a	b	a	b	a	b	c	a
π[i]	0	0								
k		q								

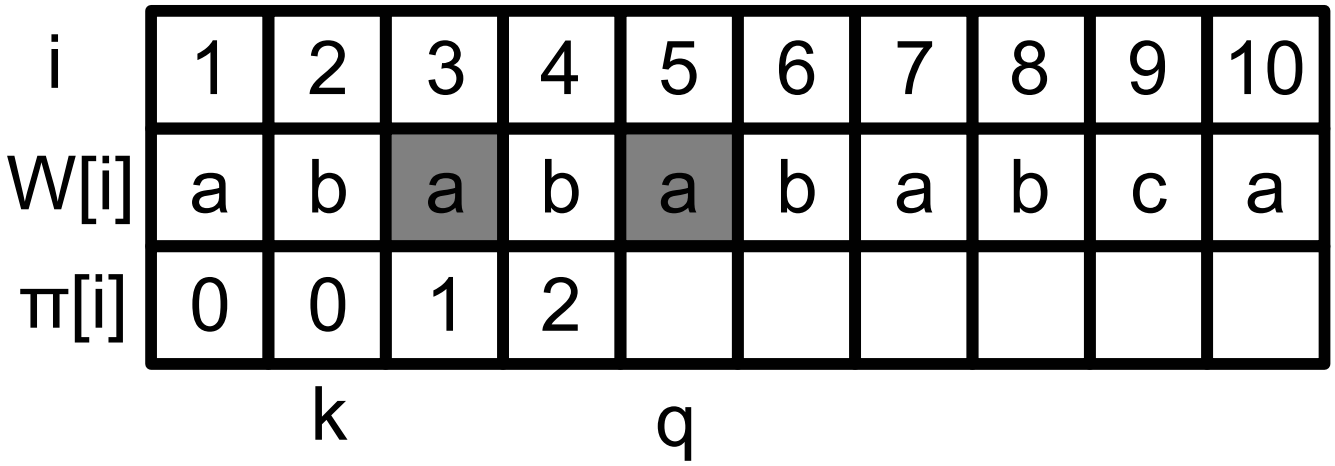
Funkcja prefiksowa



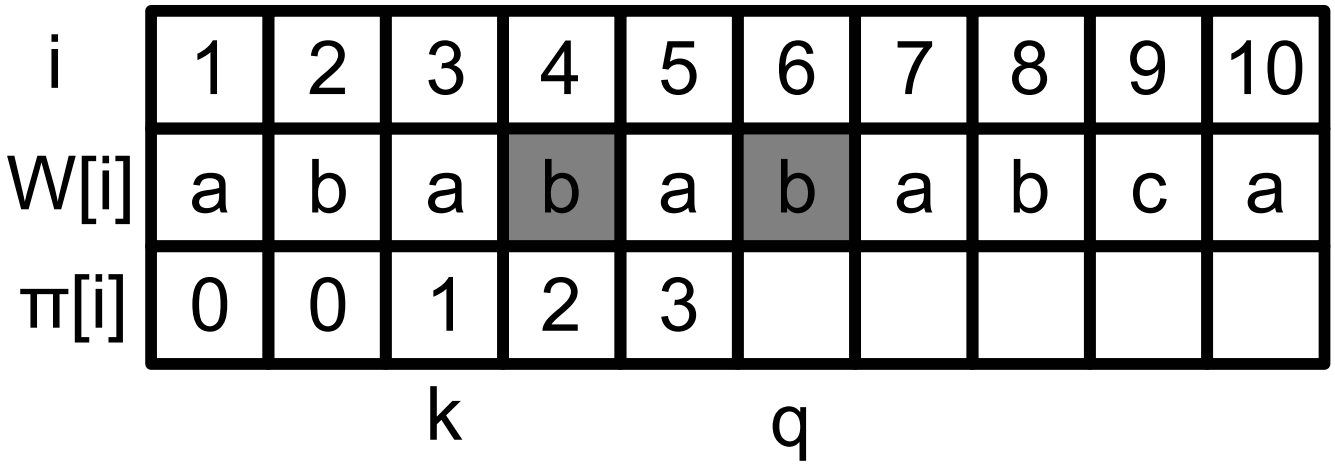
Funkcja prefiksowa



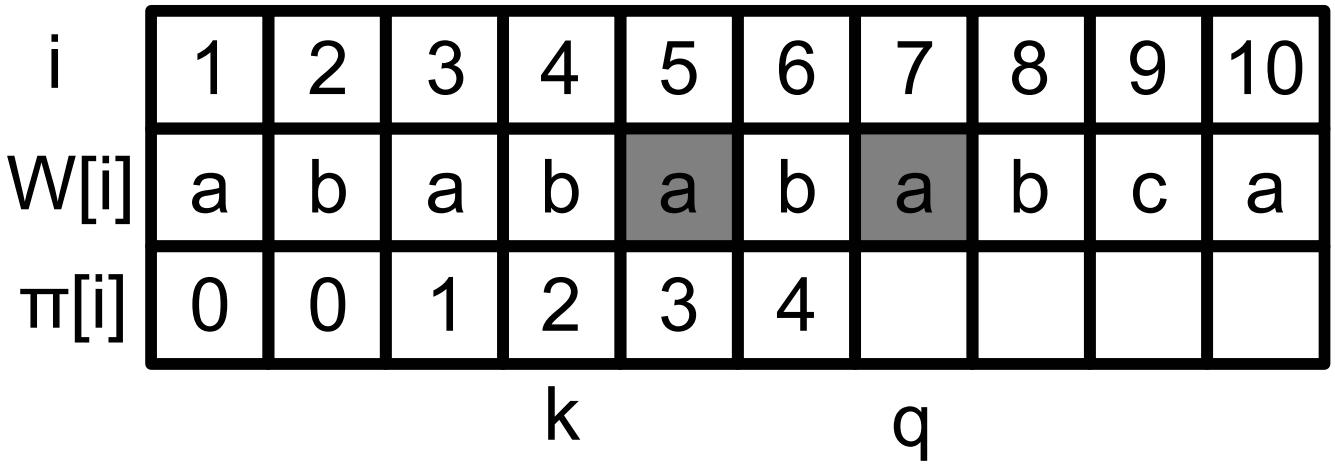
Funkcja prefiksowa



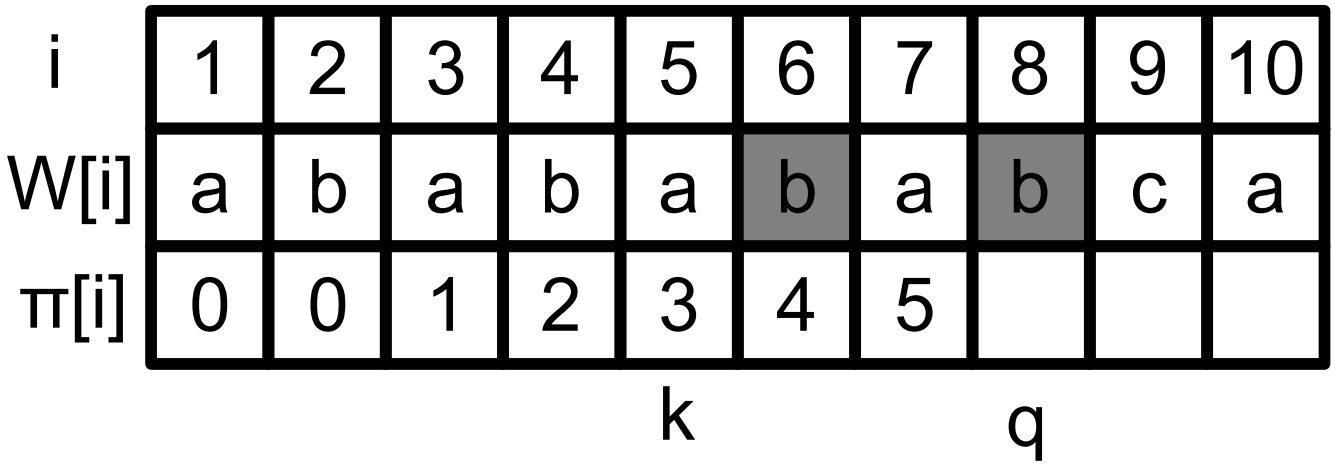
Funkcja prefiksowa



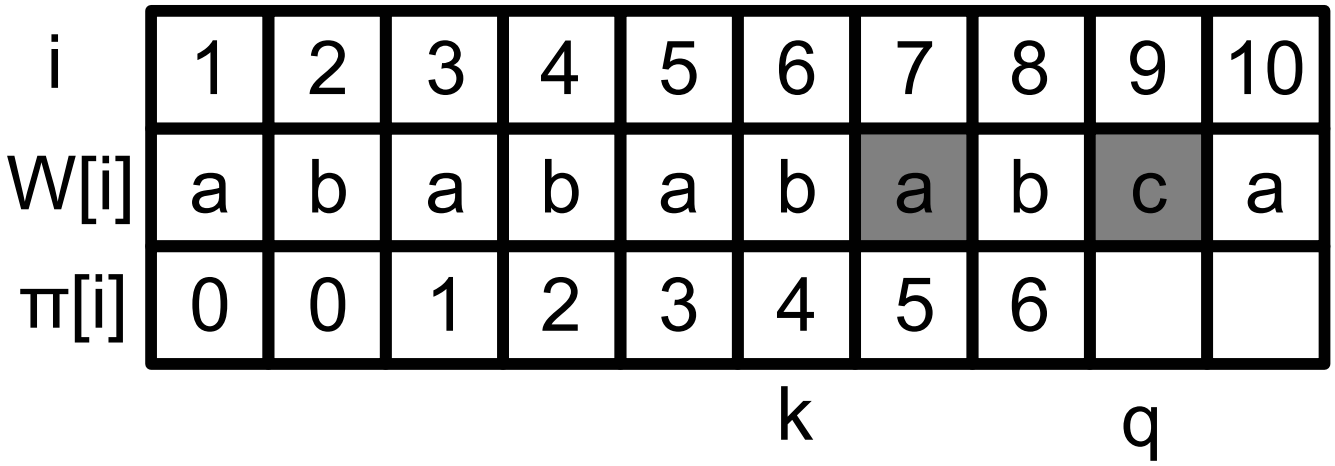
Funkcja prefiksowa



Funkcja prefiksowa



Funkcja prefiksowa

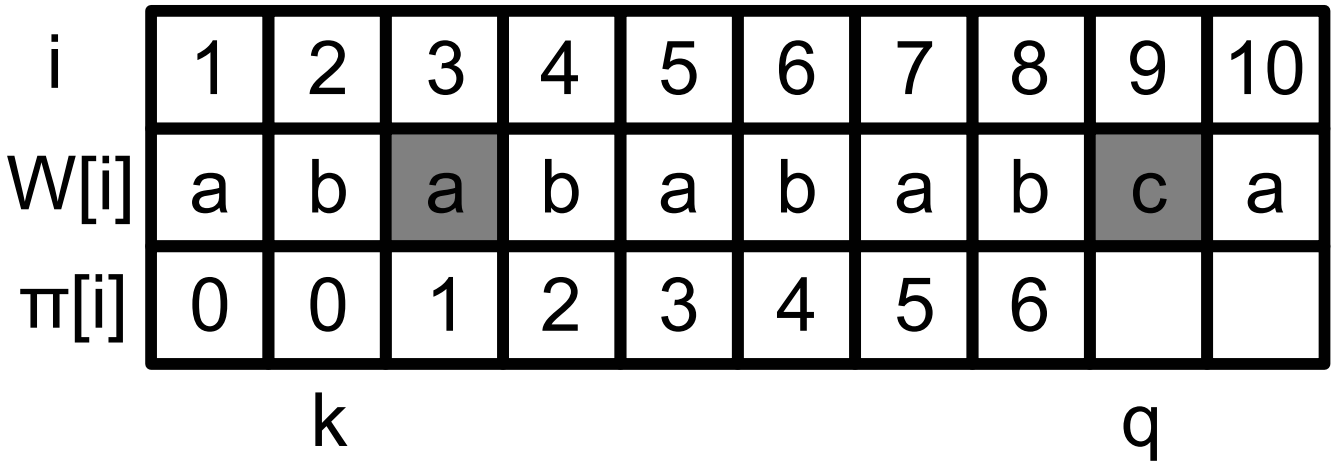


Funkcja prefiksowa

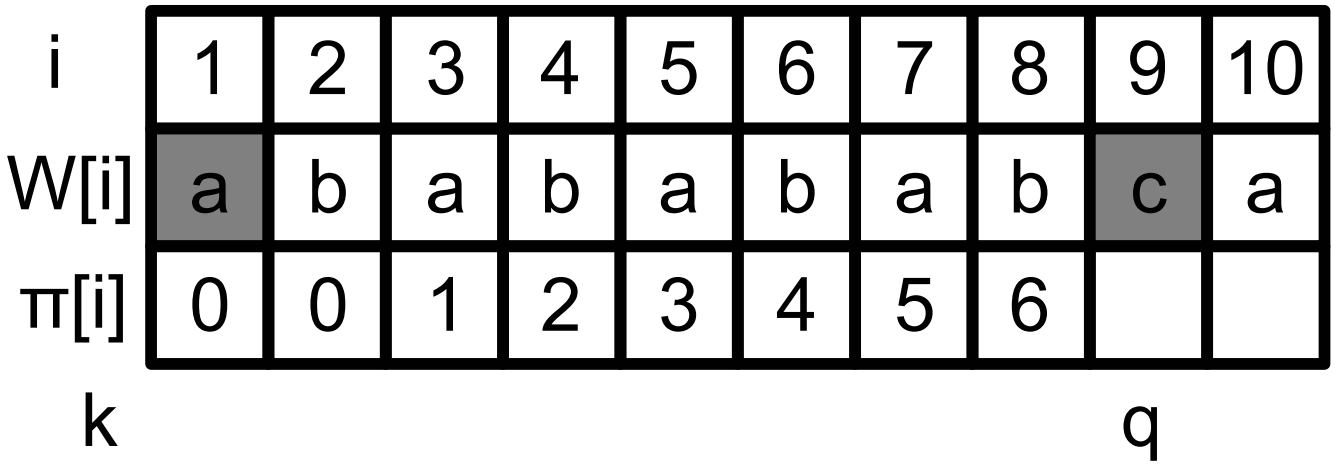
i	1	2	3	4	5	6	7	8	9	10
W[i]	a	b	a	b	a	b	a	b	c	a
π[i]	0	0	1	2	3	4	5	6		

k q

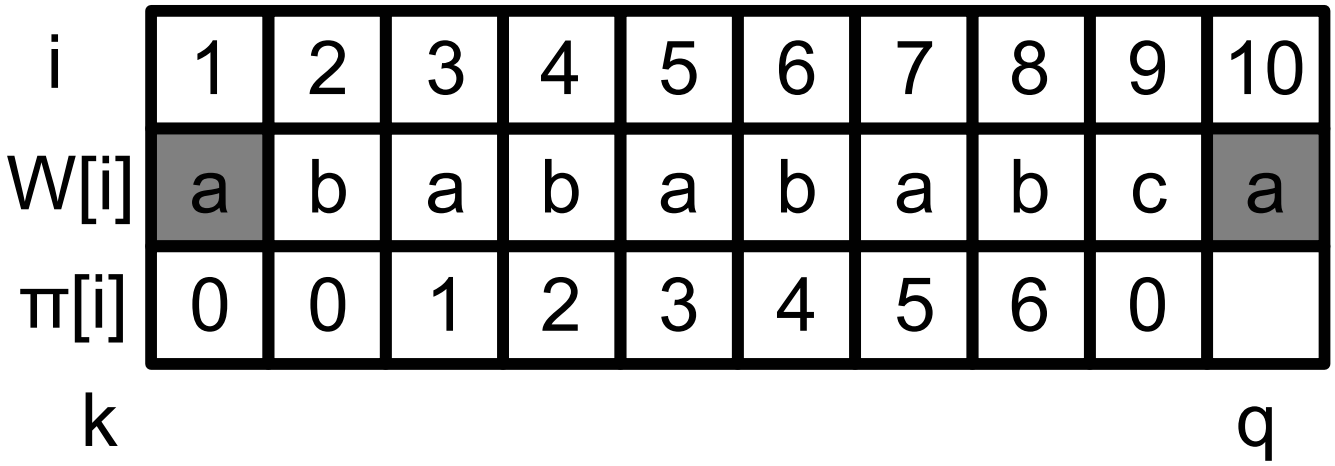
Funkcja prefiksowa



Funkcja prefiksowa



Funkcja prefiksowa



Funkcja prefiksowa

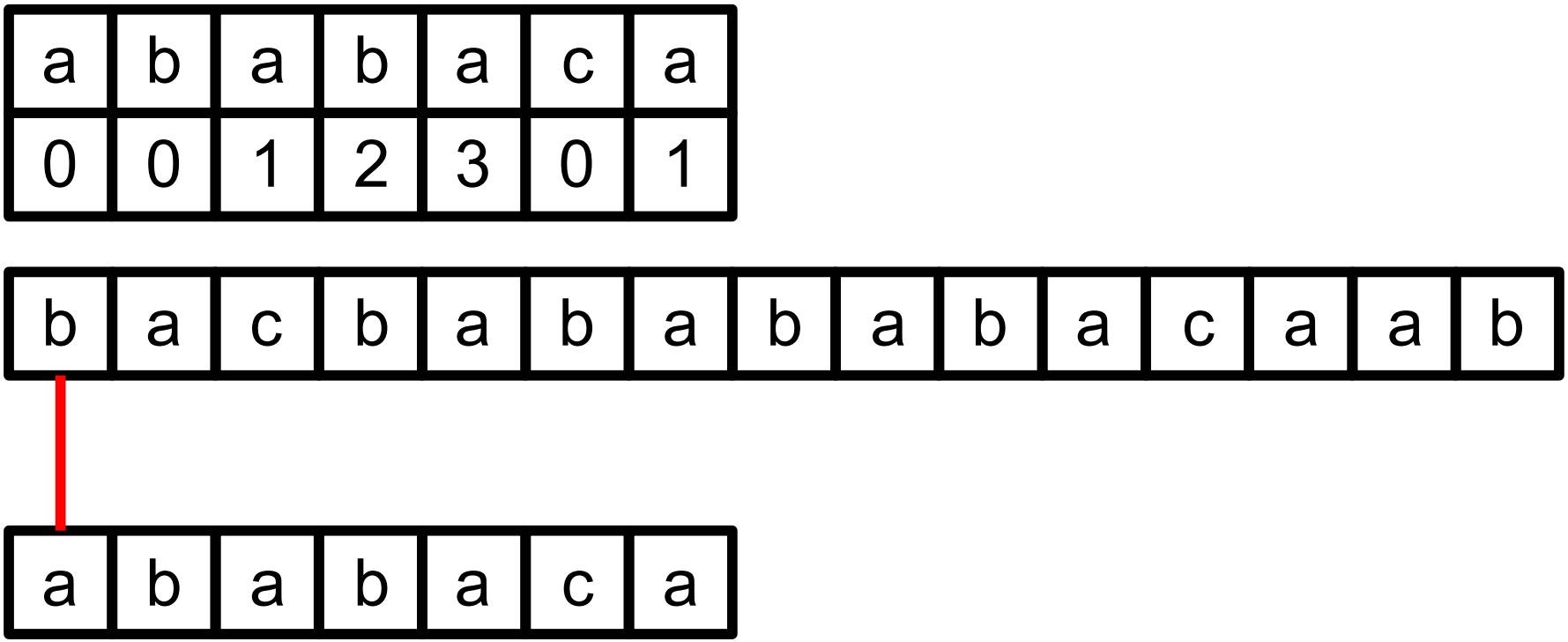
i	1	2	3	4	5	6	7	8	9	10
W[i]	a	b	a	b	a	b	a	b	c	a
π[i]	0	0	1	2	3	4	5	6	0	1

k q

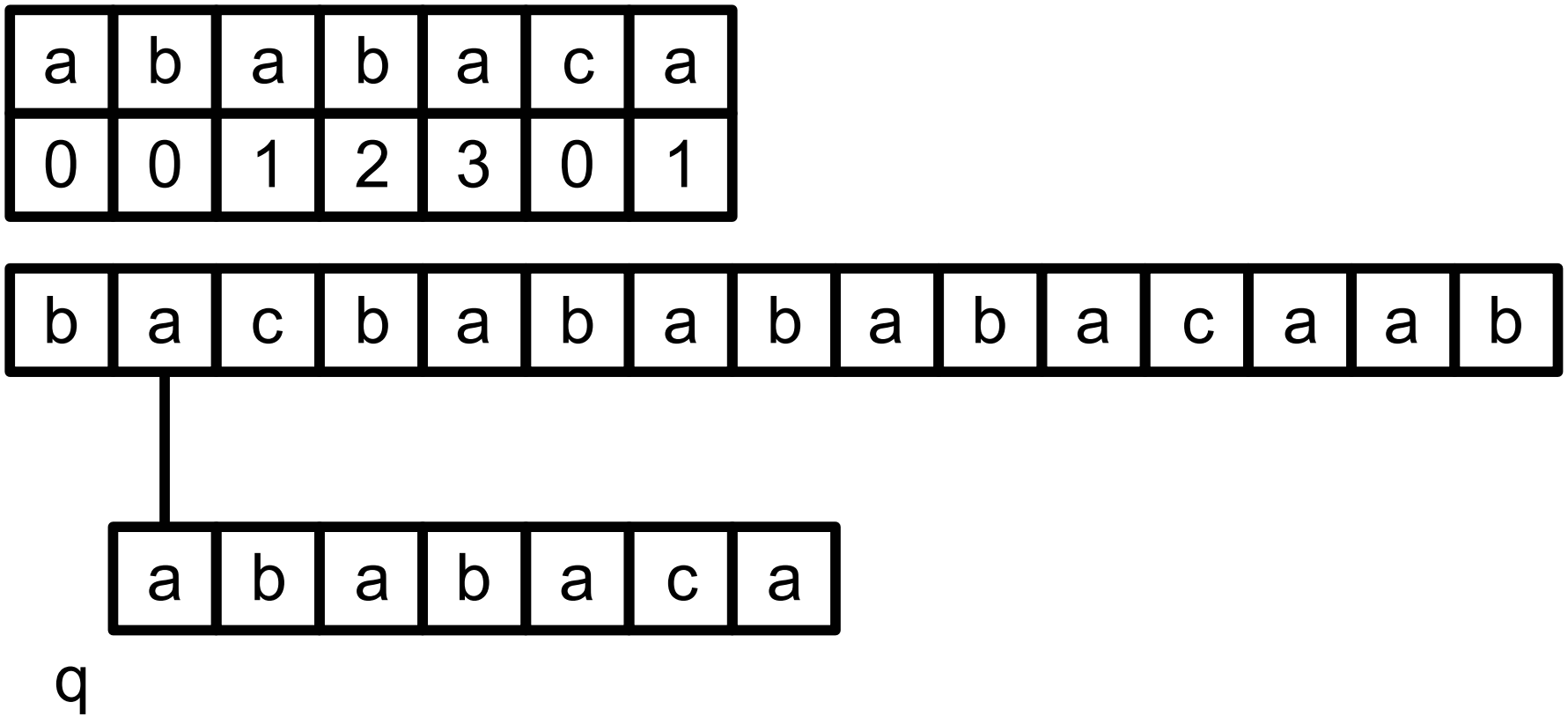
Algorytm KMP

1: n - długość T, m - długość W
2: $\pi = Funkcja - prefiksowa(W)$
3: $q = 0$
4: **for** $i = 1, \ldots n$ **do**
5: **while** $k > 0$ **i** $W[k+1] \neq T[i]$ **do**
6: $q = \pi[q]$
7: **end while**
8: **if** $W[k+1] = T[q]$ **then**
9: $k = k + 1$
10: **end if**
11: **if** $k = m$ **then**
12: znaleziono rozpoczynając od pozycji $i - m$
13: **end if**
14: **end for**

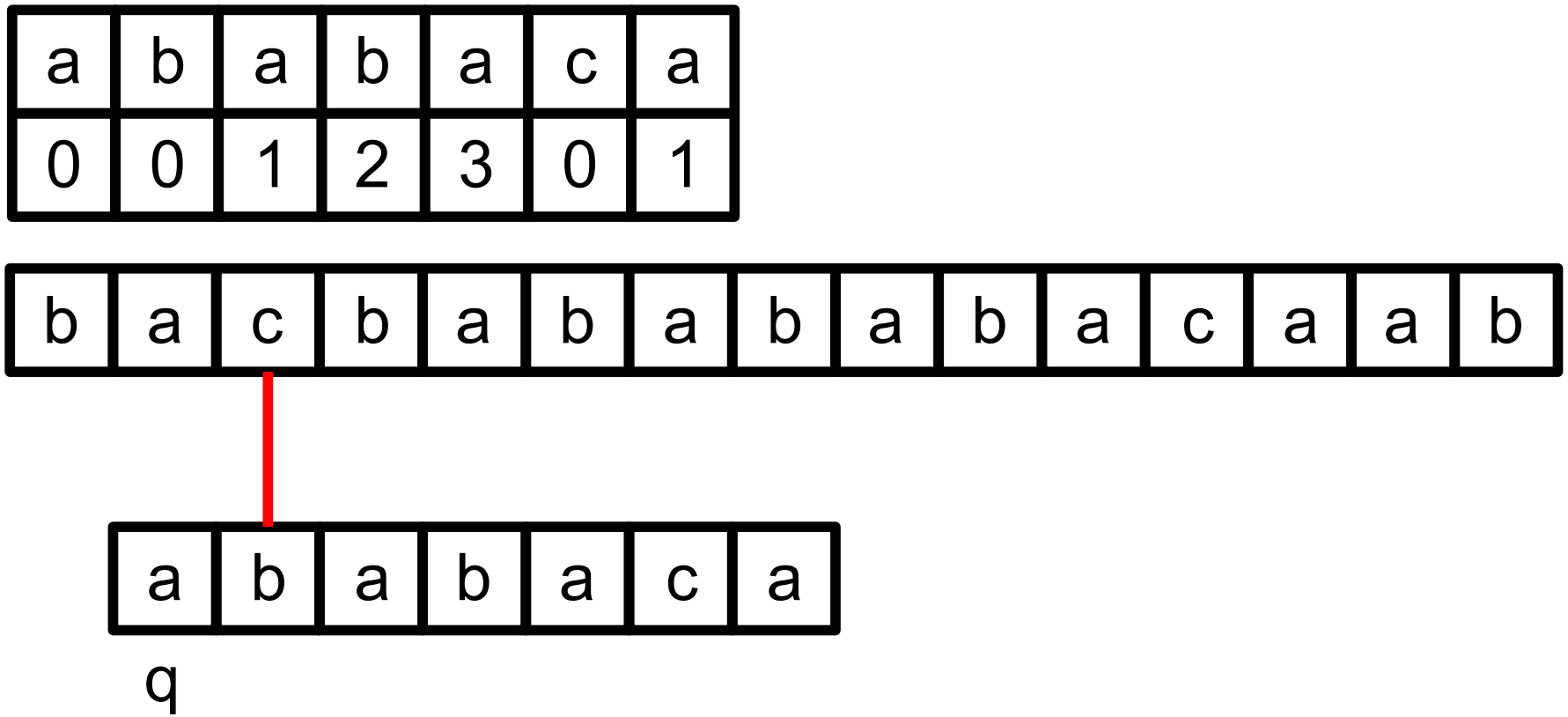
Algorytm KMP



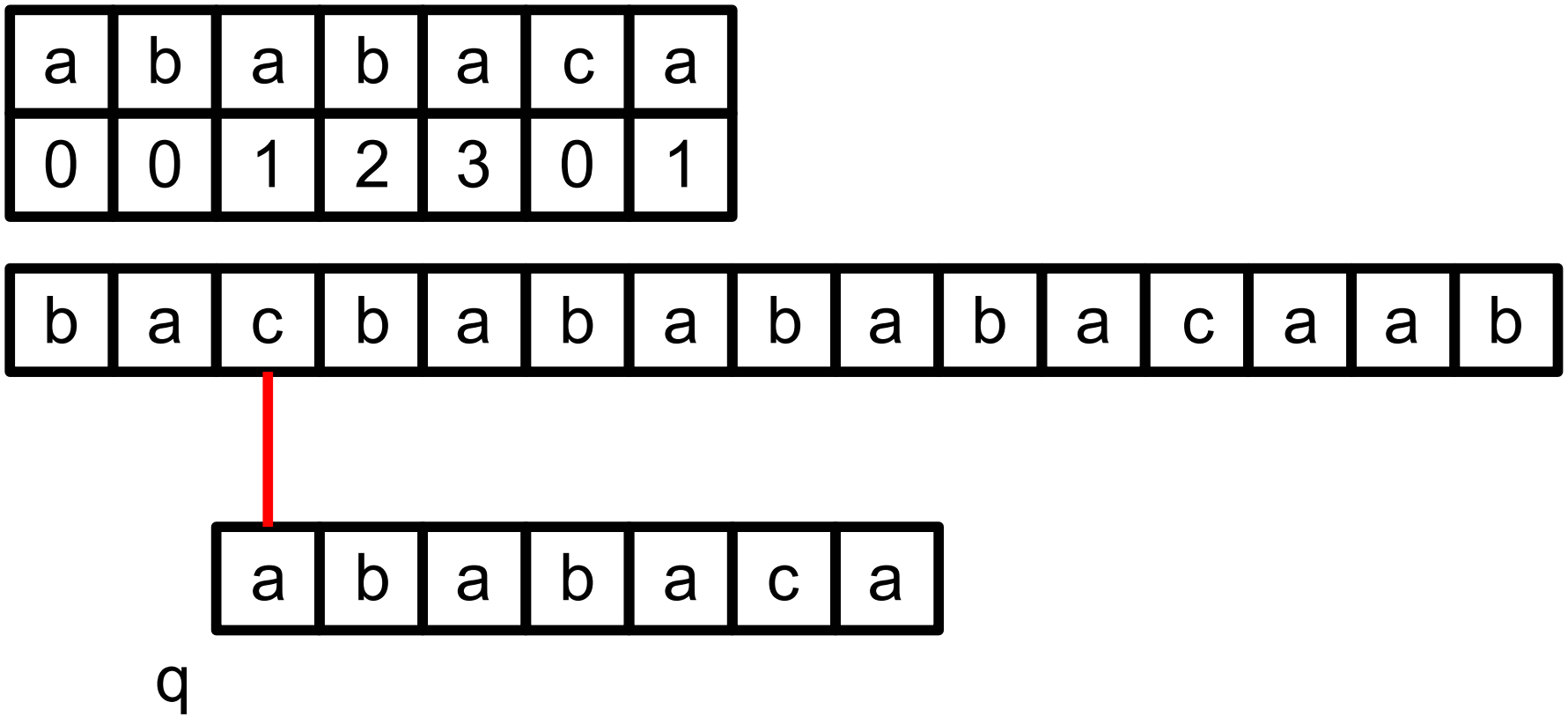
Algorytm KMP



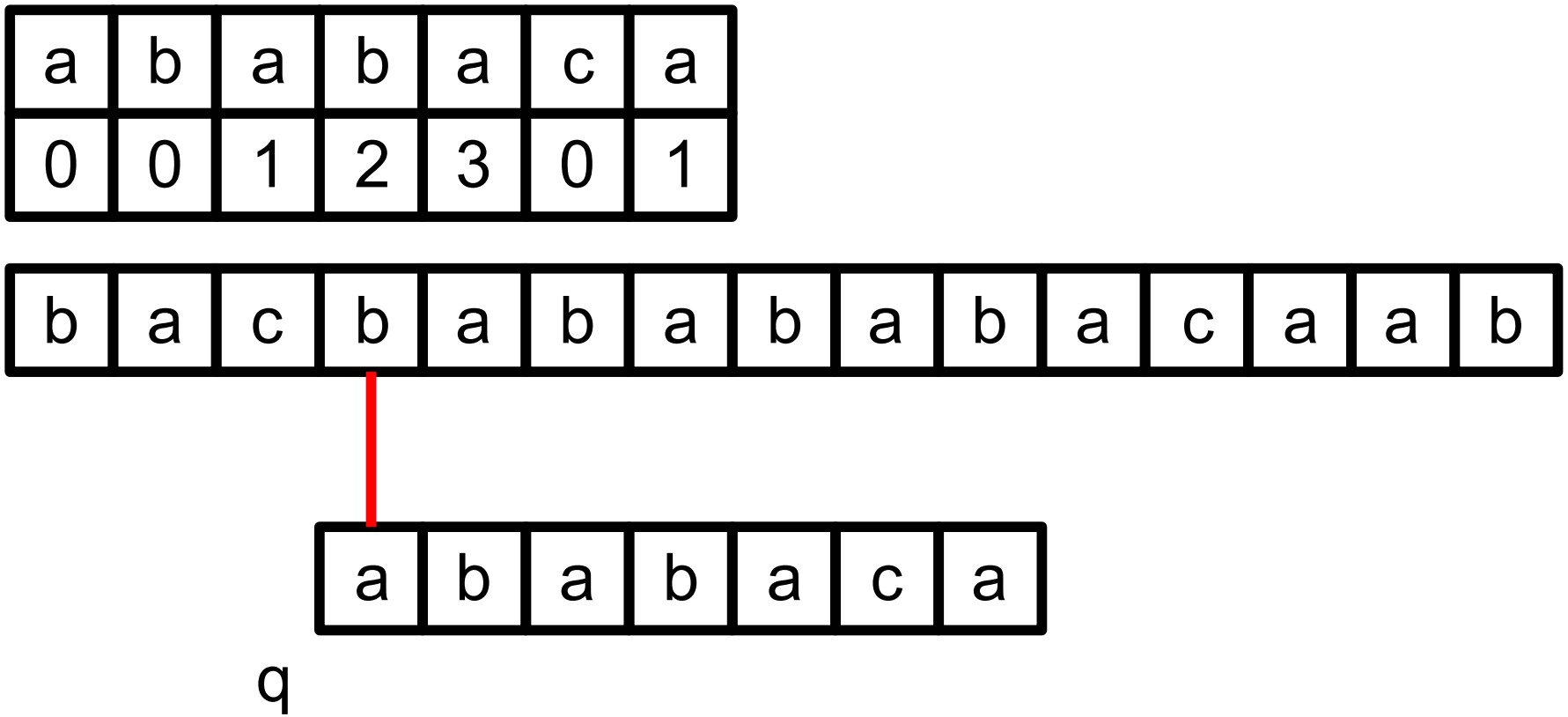
Algorytm KMP



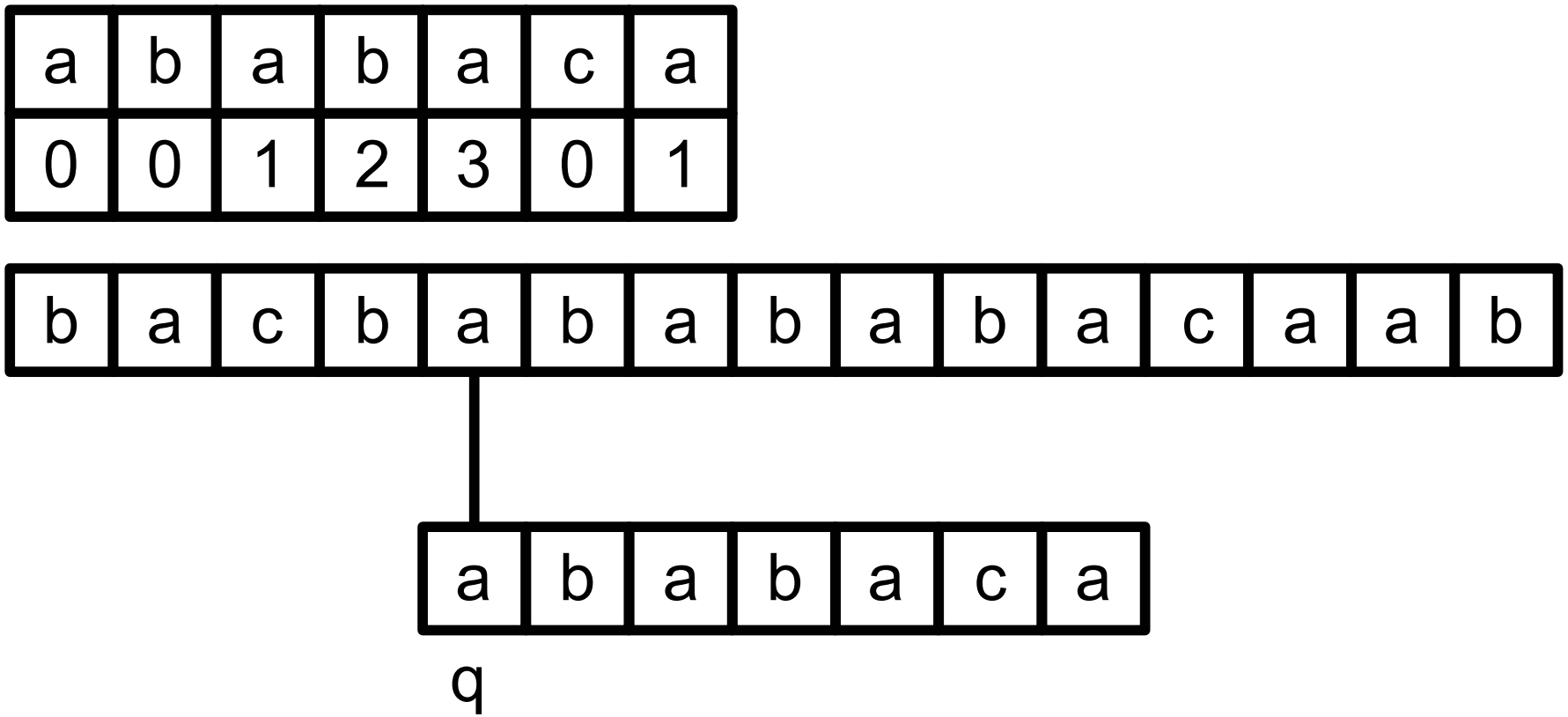
Algorytm KMP



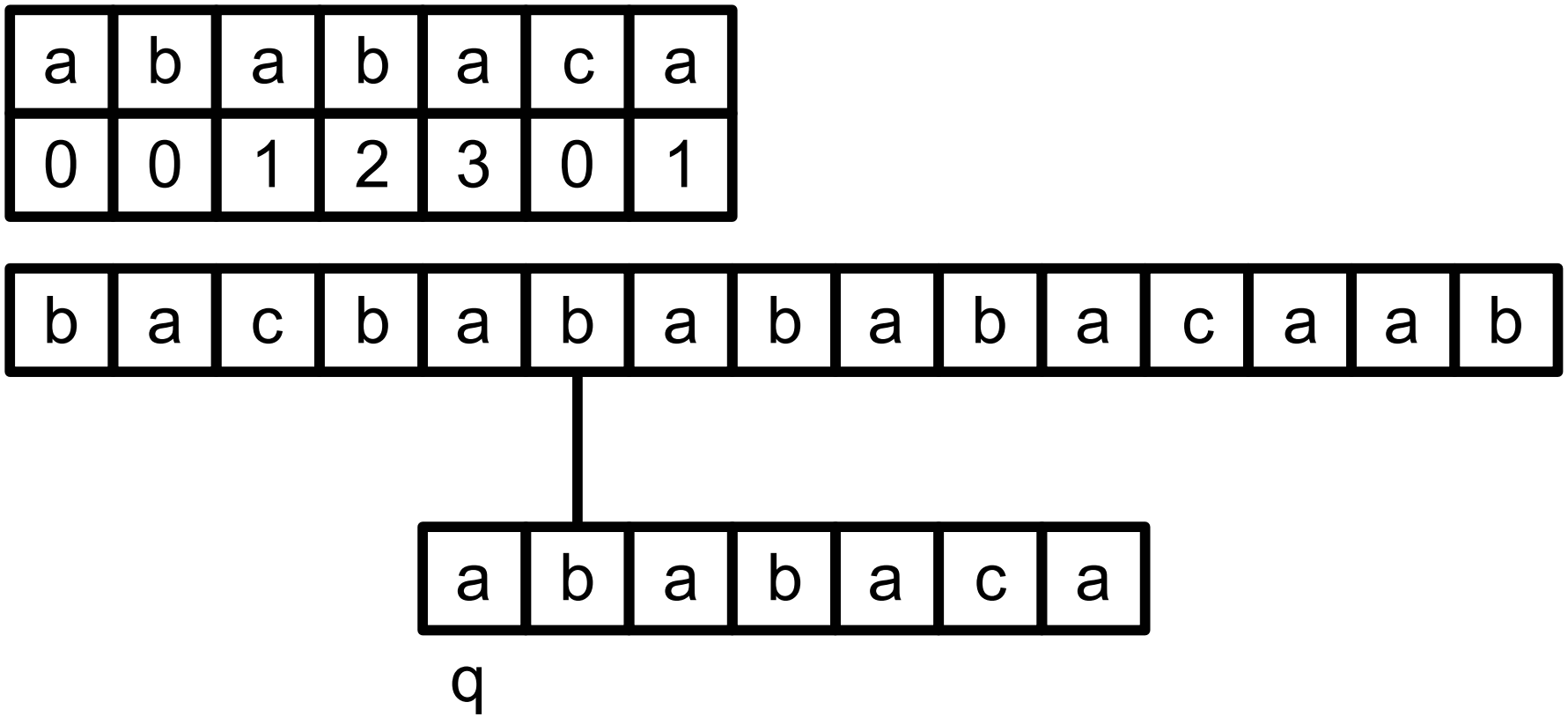
Algorytm KMP



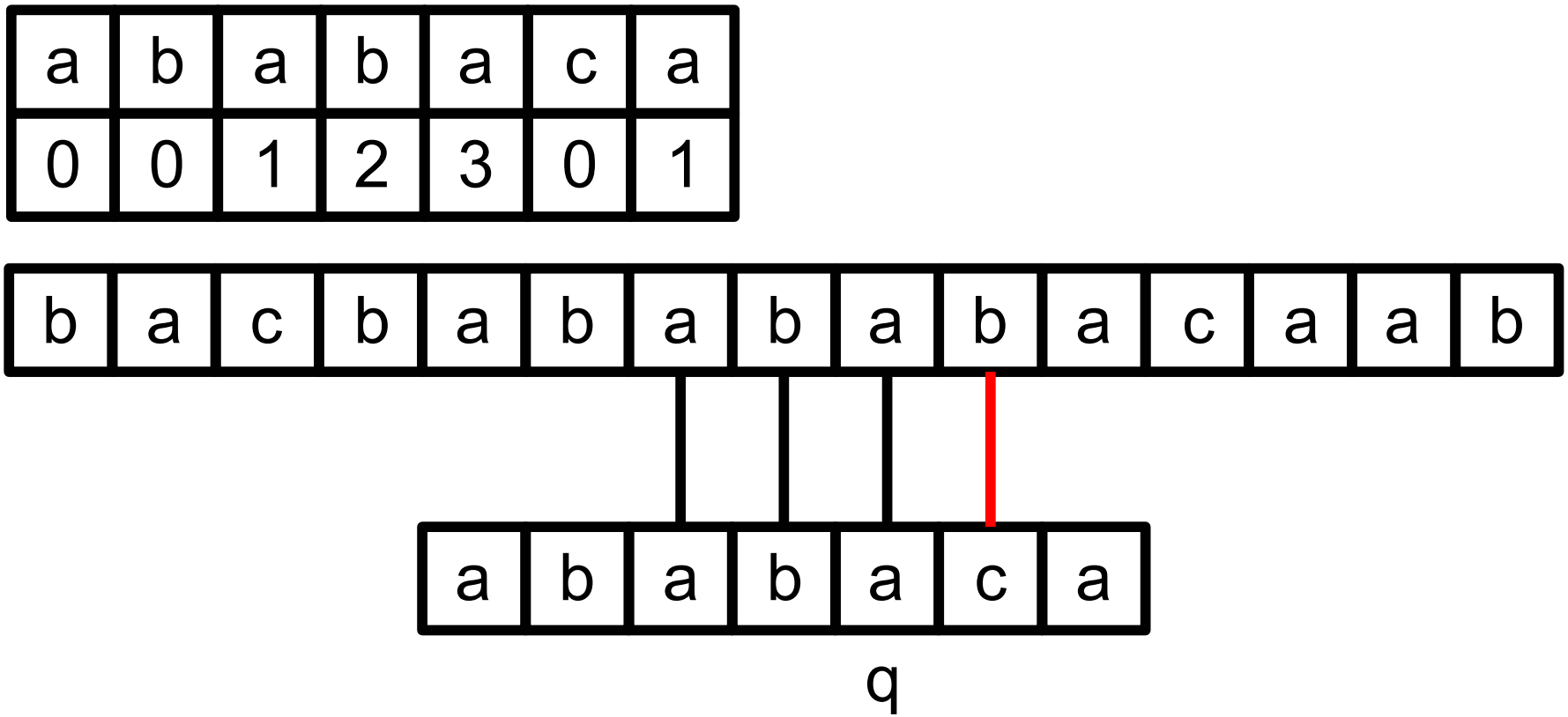
Algorytm KMP



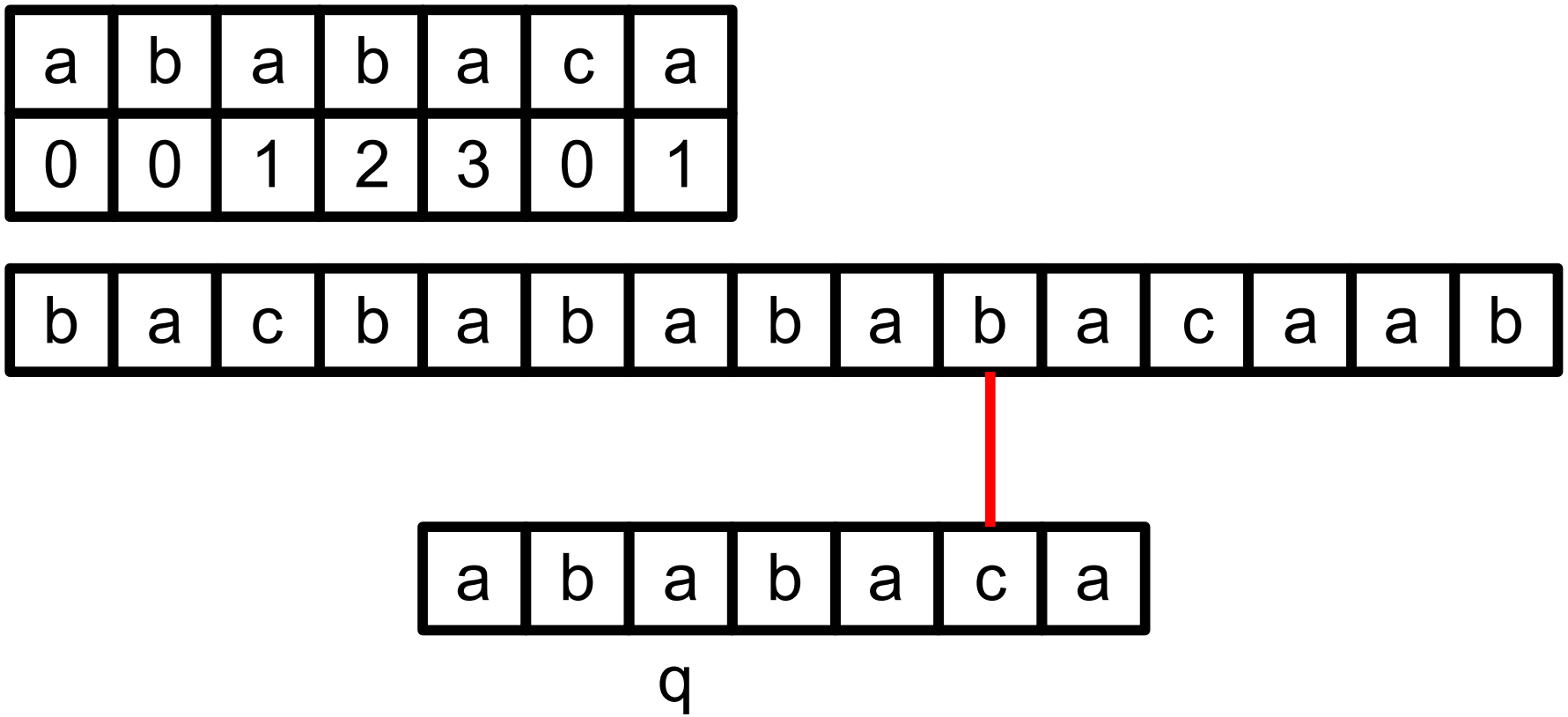
Algorytm KMP



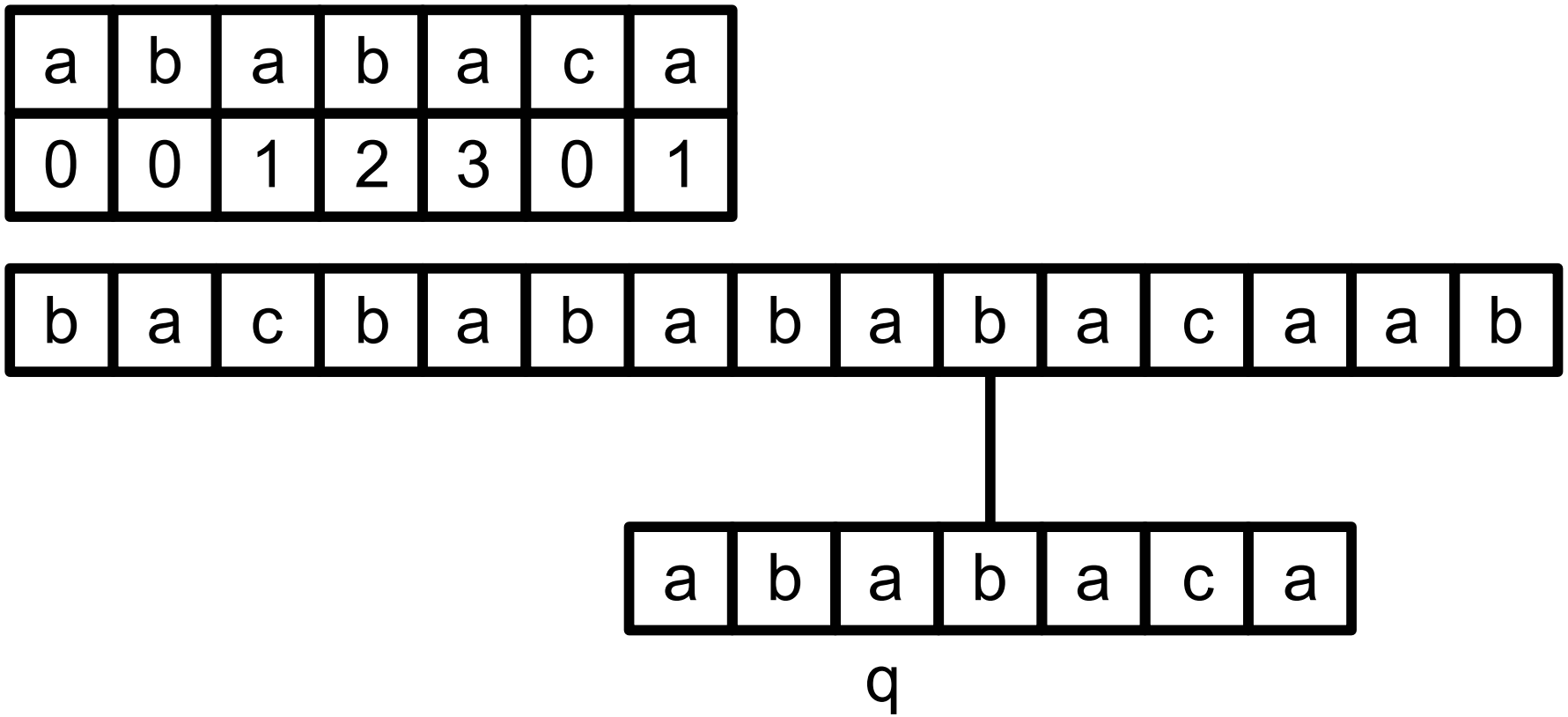
Algorytm KMP



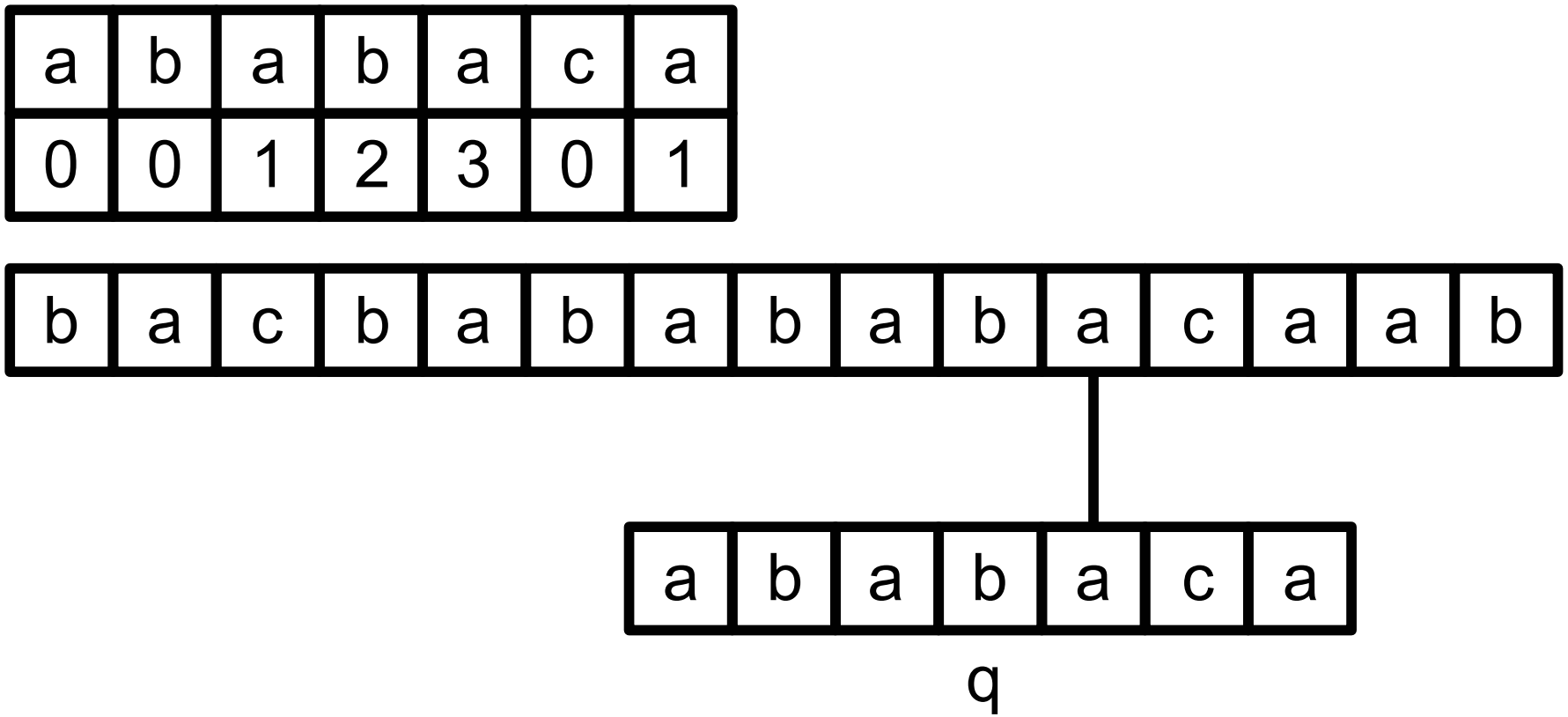
Algorytm KMP



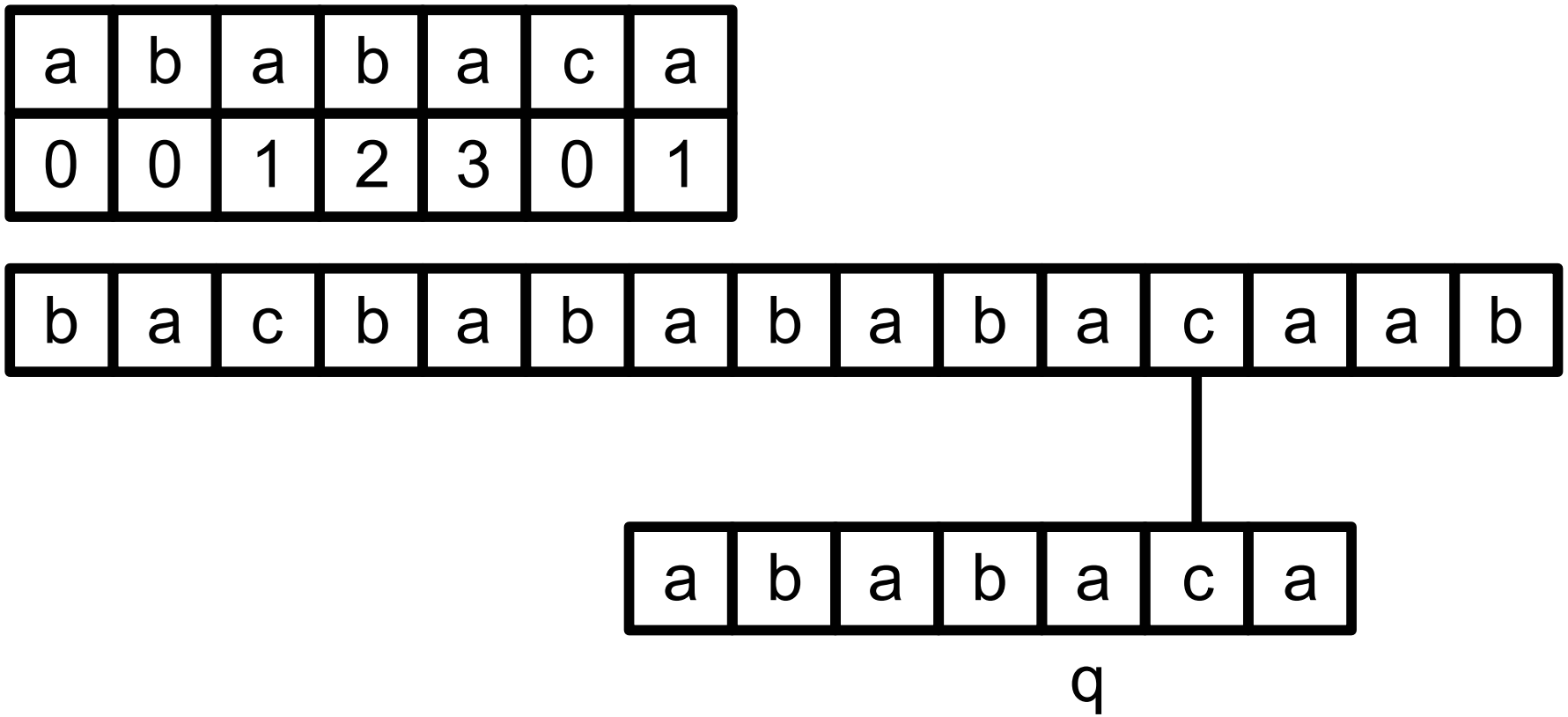
Algorytm KMP



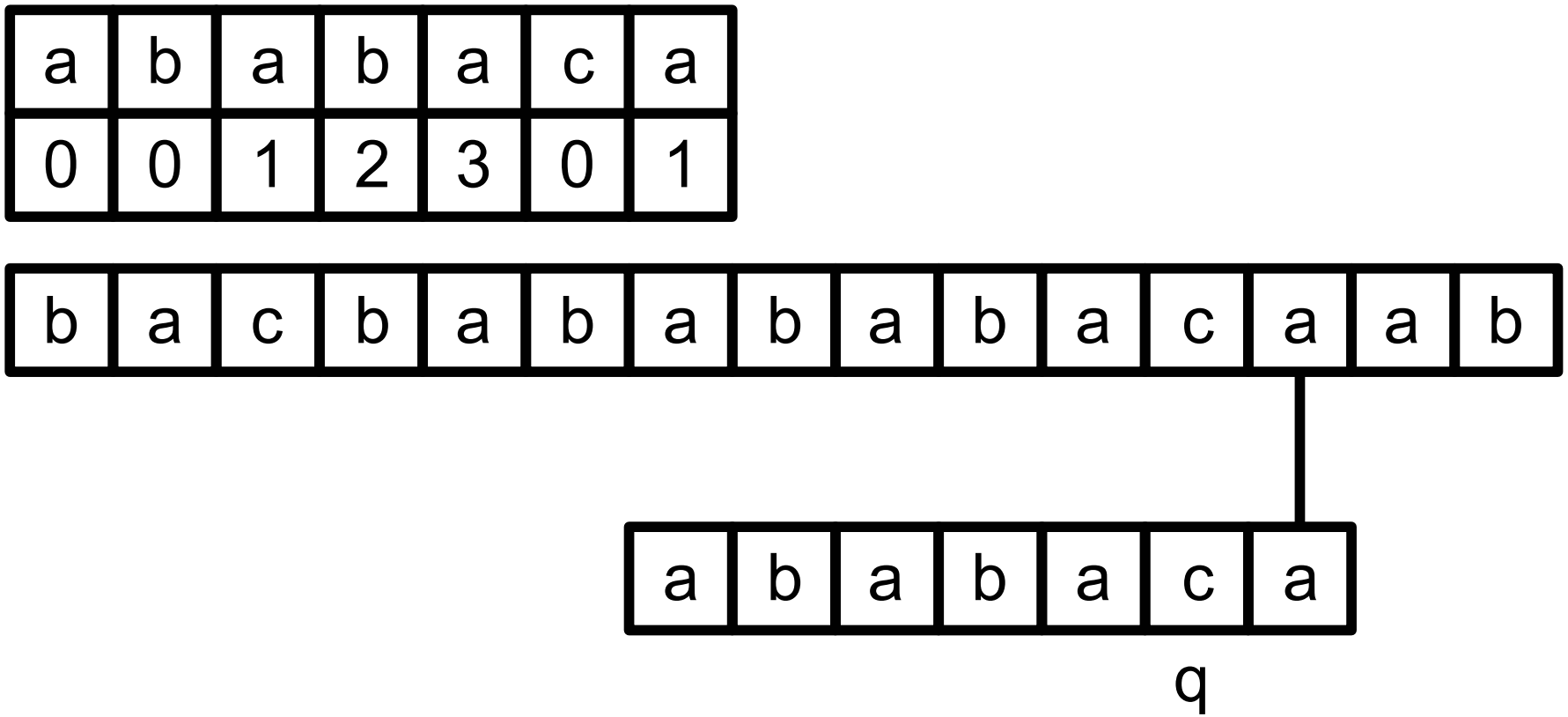
Algorytm KMP



Algorytm KMP



Algorytm KMP



Algorytm KMP

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

b	a	c	b	a	b	a	b	a	b	a	c	a	a	b

a	b	a	b	a	c	a

q