technische universität
dortmund

Algorithmen auf Sequenzen

Pattern-Matching

Dominik Kopczynski

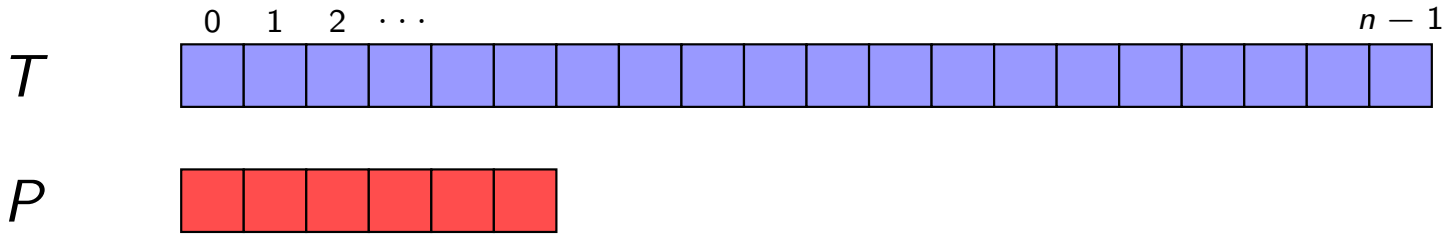
Lehrstuhl für Algorithm Engineering (LS11)
Fakultät für Informatik
TU Dortmund

Beschreibung des Pattern-Matchings

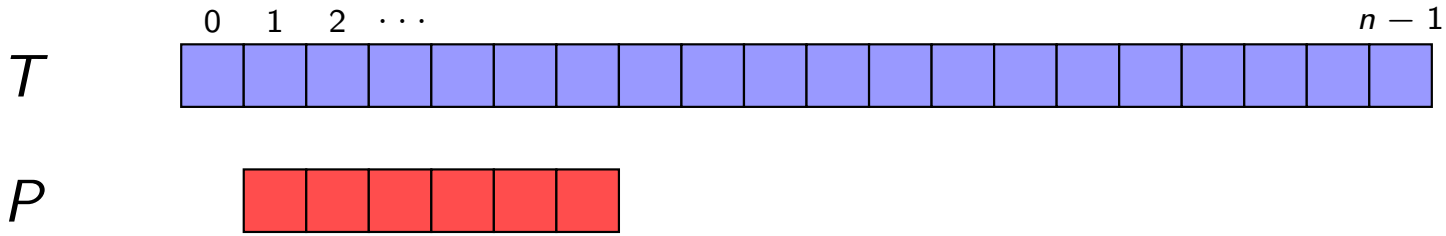
Das Pattern-Matching sei folgendermaßen beschrieben:
Gegeben sei ein endliches Alphabet Σ, ein Text $T \in \Sigma^n$ und ein Muster (Pattern) $P \in \Sigma^m$ i.d.R. $m \ll n$.

Gesucht sind alle Positionen i, für die gilt: $T[i : i + m] = P$.

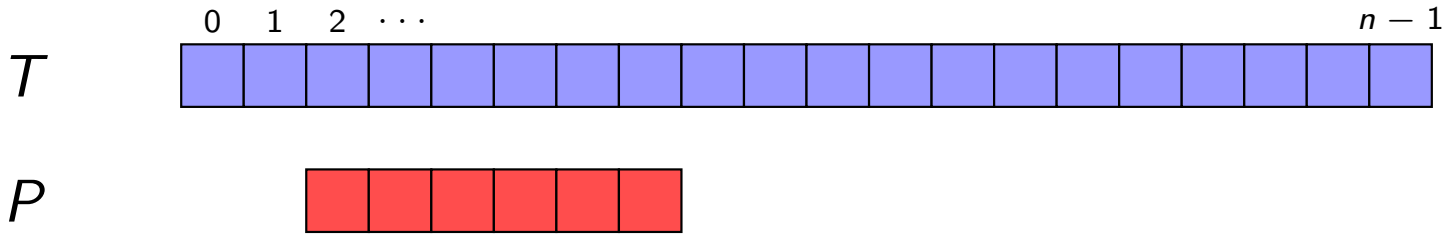
Naives Pattern-Matching



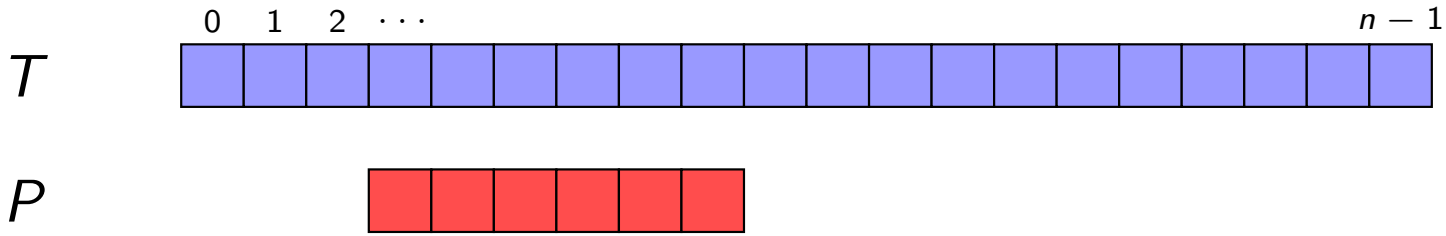
Naives Pattern-Matching



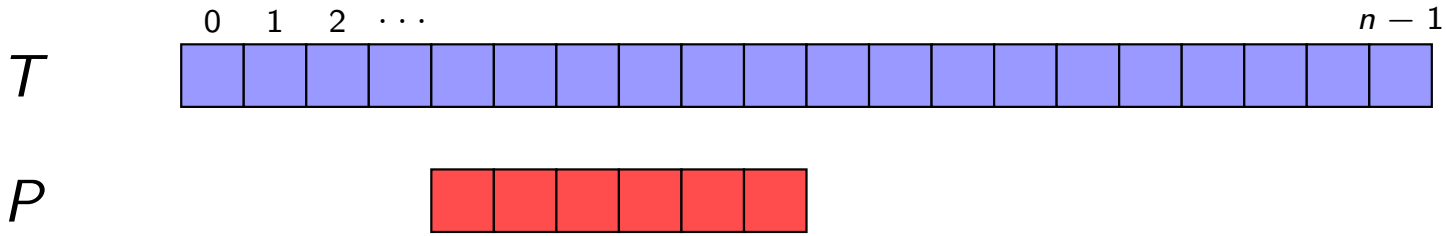
Naives Pattern-Matching



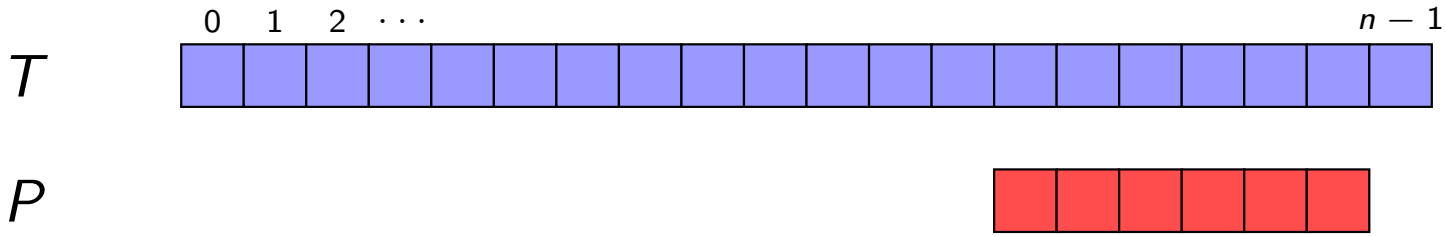
Naives Pattern-Matching



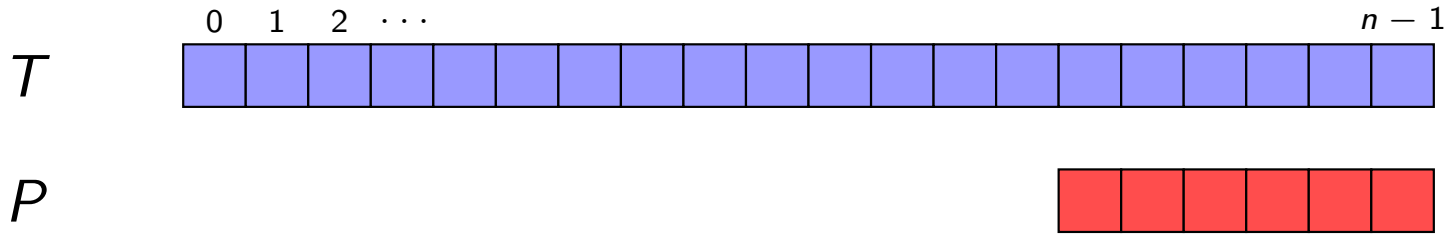
Naives Pattern-Matching



Naives Pattern-Matching

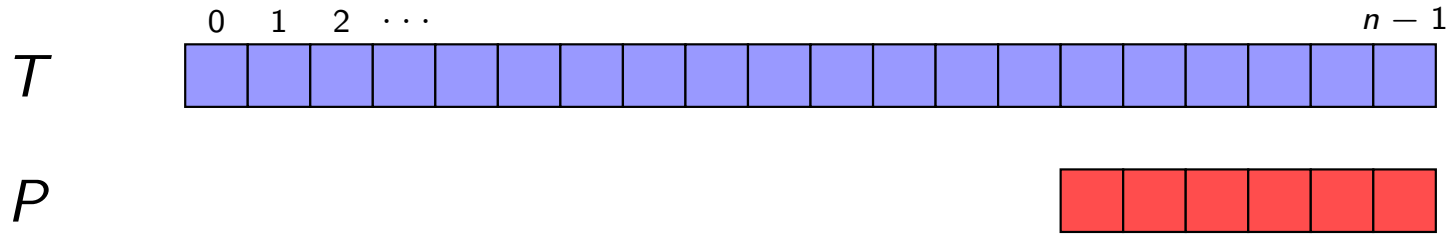


Naives Pattern-Matching



Insgesamt $n - m + 1$ Verschiebungen. Pro Verschiebung bis zu m Vergleiche. Gesamte Laufzeit: $\mathcal{O}(mn)$.

Naives Pattern-Matching



Insgesamt $n - m + 1$ Verschiebungen. Pro Verschiebung bis zu m Vergleiche. Gesamte Laufzeit: $\mathcal{O}(mn)$.

```
1 def naive(P, T):
2     m, n = len(P), len(T)
3     for i in range(n - m + 1):
4         if T[i:i+m] == P:        #naive_comparison
5             yield i, i + m
```

Laufzeitanalyse des naiven PMs

Theorem (Erwartete Laufzeit)

Sei $|\Sigma| \geq 2$. Seien ein Muster der Länge m und ein Text der Länge n zufällig gleichverteilt gewählt. Dann beträgt die Worst-case-Laufzeit des naiven Algorithmus $\mathcal{O}(mn)$, aber die erwartete Laufzeit lediglich $\mathcal{O}(n)$.

Laufzeitanalyse des naiven PMs

Die Wahrscheinlichkeit p, dass jeweils ein zufällig gezogenes Zeichen aus P und T übereinstimmen, beträgt

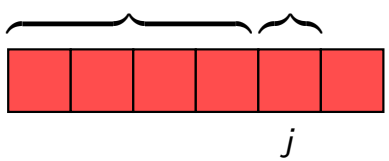
$$p := \frac{|\Sigma|}{|\Sigma|^2} = \frac{1}{|\Sigma|}.$$

Laufzeitanalyse des naiven PMs

Die Wahrscheinlichkeit p, dass jeweils ein zufällig gezogenes Zeichen aus P und T übereinstimmen, beträgt

$$p := \frac{|\Sigma|}{|\Sigma|^2} = \frac{1}{|\Sigma|}.$$

- Die W'keit, dass alle Zeichen des Musters mit dem Text übereinstimmen, beträgt p^m.
- Die W'keit, dass das Muster erst an Stelle j mit dem Text nicht übereinstimmt, beträgt $p^{j-1} \cdot (1-p)$.



Laufzeitanalyse des naiven PMs

Die erwartete Anzahl an Vergleichen für ein Pattern der Länge m beträgt demnach:

$$E_m := mp^m + \sum_{j=1}^{m} jp^{j-1} \cdot (1-p)$$

Laufzeitanalyse des naiven PMs

Die erwartete Anzahl an Vergleichen für ein Pattern der Länge m beträgt demnach:

$$E_m := mp^m + \sum_{j=1}^{m} jp^{j-1} \cdot (1-p)$$

Und für beliebige Längen $m \to \infty$:

$$E_m < E_\infty := (1-p) \sum_{j=0}^{\infty} jp^{j-1}$$

Laufzeitanalyse des naiven PMs

Der Term $\sum_{j=0}^{\infty} jp^{j-1}$ ist die Ableitung von $\sum_{j=0}^{\infty} p^j = 1/(1-p)$.
Somit gilt $\sum_{j=0}^{\infty} jp^{j-1} = 1/(1-p)^2$.

Laufzeitanalyse des naiven PMs

Der Term $\sum_{j=0}^{\infty} jp^{j-1}$ ist die Ableitung von $\sum_{j=0}^{\infty} p^j = 1/(1-p)$.
Somit gilt $\sum_{j=0}^{\infty} jp^{j-1} = 1/(1-p)^2$.

Eingesetzt in E_∞ ergibt es

$$E_\infty = \frac{1-p}{(1-p)^2} = \frac{1}{1-p}.$$

Laufzeitanalyse des naiven PMs

Der Term $\sum_{j=0}^{\infty} jp^{j-1}$ ist die Ableitung von $\sum_{j=0}^{\infty} p^j = 1/(1-p)$.
Somit gilt $\sum_{j=0}^{\infty} jp^{j-1} = 1/(1-p)^2$.

Eingesetzt in E_∞ ergibt es

$$E_\infty = \frac{1-p}{(1-p)^2} = \frac{1}{1-p}.$$

Aus der Definition $p = 1/|\Sigma|$ folgt nun

$$E_m < \frac{|\Sigma|}{|\Sigma|-1}.$$

Laufzeitanalyse des naiven PMs

Der Term $\sum_{j=0}^{\infty} jp^{j-1}$ ist die Ableitung von $\sum_{j=0}^{\infty} p^j = 1/(1-p)$.
Somit gilt $\sum_{j=0}^{\infty} jp^{j-1} = 1/(1-p)^2$.

Eingesetzt in E_∞ ergibt es

$$E_\infty = \frac{1-p}{(1-p)^2} = \frac{1}{1-p}.$$

Aus der Definition $p = 1/|\Sigma|$ folgt nun

$$E_m < \frac{|\Sigma|}{|\Sigma|-1}.$$

Für ein 2-buchstabiges Alphabet beträgt $E_m < 2$, für $|\Sigma| \to \infty$ sogar nur $E_m = 1$. Die Laufzeit entspricht somit $\mathcal{O}(nE_m) = \mathcal{O}(n)$.

Optimales Pattern Matching

Minimale optimale Laufzeit für Pattern Matching
offensichtlich $\mathcal{O}(m+n)$, Pattern mind. einmal lesen $\mathcal{O}(m)$ und Text mind. einmal lesen $\mathcal{O}(n)$.

Idealerweise wird jedes Zeichen des Textes nur ein einziges mal gelesen und abhängig vom Zustand des bereits gelesenen Pattern der Zustand geändert $\rightarrow$ Automaten.

NFA

Definition

Ein nichtdeterministischer endlicher Automat (NFA) ist ein Tupel $(Q, Q_0, F, \Sigma, \Delta)$, wobei

- Q eine endliche Menge von Zuständen,
- $Q_0 \subset Q$ eine Menge von Startzuständen,
- $F \subset Q$ eine Menge von akzeptierenden Zuständen,
- Σ das Eingabealphabet und
- $\Delta\colon Q \times \Sigma \to 2^Q$ eine nichtdeterministische Übergangsfunktion ist.

NFA

Funktionsweise:

- Es gibt stets eine Menge aktiver Zustände $A \subset Q$.
- Am Anfang ist $A = Q_0$.
- Nach Lesen von Zeichen $c \in \Sigma$ sind die Zustände aktiv die gemäß Δ von A über c erreicht werden können.
- Der gelesene String wird akzeptiert, wenn $A \cap F \neq \emptyset$.

NFA

Die Übergangsfunktion $\Delta\colon Q \times \Sigma \to 2^Q$ gibt für jedes (q, c) eine Menge von Folgezuständen an. Erweiterung des Definitionsbereiches von Q auf 2^Q.

$$\Delta(A, c) := \bigcup_{q \in A} \Delta(q, c)\,.$$

Erweiterung des Definitionsbereiches von $c \in \Sigma$ auf $x \in \Sigma^*$.

$$\begin{aligned}\Delta(A, \epsilon) &:= A \\ \Delta(A, xc) &:= \Delta(\Delta(A, x), c)\end{aligned}$$

NFA

Einführung von Epsilon-Transitionen. Für jeden Zustand q sei E_q sein ϵ-*Abschluss*; das ist die Menge der Zustände, die von q aus „sofort“ erreicht wird, es folgt $\Delta(q, \epsilon) := E_q$.

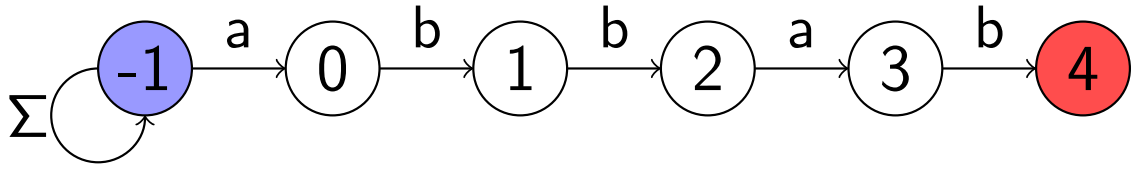
NFA für Pattern Matching

Ein NFA akzeptiert P für alle Strings der Form Σ^*. Solch ein NFA besteht aus einer Kette von Zuständen an deren Kanten entlang P buchstabiert ist. Zusätzlich besitzt der Startzustand eine Schleife, so dass Zustand beim Lesen eines Zeichens nie verlassen wird.

- Sei $Q = \{-1, 0, \ldots, m-1\}$ mit $m = |P|$
- $Q_0 = \{-1\}$
- $F = \{m-1\}$
- $\Delta(-1, P[0]) = \{-1, 0\}$ und $\Delta(-1, c) = \{-1\}$ für alle $c \neq P[0]$;
  für $0 \leq q \leq m-2$ ist $\Delta(q, P[q+1]) = \{q+1\}$ und $\Delta(q, c) = \{\}$ für alle $c \neq P[q+1]$
- und $\Delta(m-1, c) = \{\}$ für alle $c \in \Sigma$.

NFA für Pattern Matching

Beispiel für NFA des Patterns **abbab**:



Algorithmusidee: Man kann beim Lesen eines Texts die aktive Zustandsmenge A eines NFA verfolgen und erhält so einen Algorithmus, der aber auch die Laufzeit $\mathcal{O}(mn)$ hat, denn die Menge A hat die Größe $\mathcal{O}(m)$.

DFA

Idee: DFA konstruieren, der (armotisiert) konstante Zeit für die Zustandsänderung braucht.

Definition

Ein deterministischer endlicher Automat (DFA) ist ein Tupel $(Q, q_0, F, \Sigma, \delta)$, wobei

- Q eine endliche Menge von Zuständen,
- Startzustand $q_0 \in Q$,
- $F \subset Q$ eine Menge von akzeptierenden Zuständen,
- Σ das Eingabealphabet und
- $\delta \colon Q \times \Sigma \to Q$ eine deterministische Übergangsfunktion ist.

DFA

Funktionsweise:

- Der Automat startet am Zustand q_0 und liest nacheinander Zeichen aus Σ.
- δ ordnet jedem Paar (q, c) einen neuen Zustand zu.
- Ist der neue Zustand in F, gibt der Automat das Signal „akzeptiert“.

Bei Transformation von NFA zu DFA mittels *Teilmengenkonstrution* können bei k Zuständen eines NFA bis zu 2^k Zustände des DFA entstehen. Erfreulicherweise ändert sich die Anzahl der Zustände zwischen des NFA und DFA für Pattern-Matching *nicht*.

DFA

Theorem

Sei A die aktive Zustandsmenge des Pattern-Matching-NFA. Sei $a^* := \max A$. Dann ist A durch a^* eindeutig bestimmt. Der äquivalente DFA hat genauso viele Zustände wie der NFA.

Beweis:

- Der Wert von a^* bestimmt die letzten $a^* + 1$ gelesenen Zeichen des Textes
- Diese sind gleich dem Präfix $P[: a^*]$.
- Ein Zustand $q < a^*$ ist genau dann aktiv, wenn die letzten $q + 1$ gelesenen Zeichen ebenso gleich dem Präfix $P[: q]$ und einem Suffix von $P[: a^*]$ sind.

DFA

Beweis:

- Es gilt also $P[a^* - q : a^*] = P[q]$.

  Beispiel: P = $\overbrace{\underbrace{\text{aabb}}_{P[:q]}\underbrace{\text{aaabb}}_{P[a^*-q:a^*]}}^{P[:a^*]}\text{aba}$

- Da es also zu jedem a^* nur eine mögliche Zustandsmenge A mit $a^* = \max A$ gibt, hat der DFA auf jeden Fall nicht mehr Zustände als der NFA.
- Da aber auch jeder NFA-Zustand vom Startzustand aus erreichbar ist, hat der DFA auch nicht weniger Zustände als der NFA. □

DFA

Für **abbab** gibt es im NFA die folgenden möglichen aktiven Zustandsmengen, und keine weiteren:

$a^* = -1$: $\{-1\}$
$a^* = 0$: $\{-1, 0\}$
$a^* = 1$: $\{-1, 1\}$
$a^* = 2$: $\{-1, 2\}$
$a^* = 3$: $\{-1, 0, 3\}$
$a^* = 4$: $\{-1, 1, 4\}$

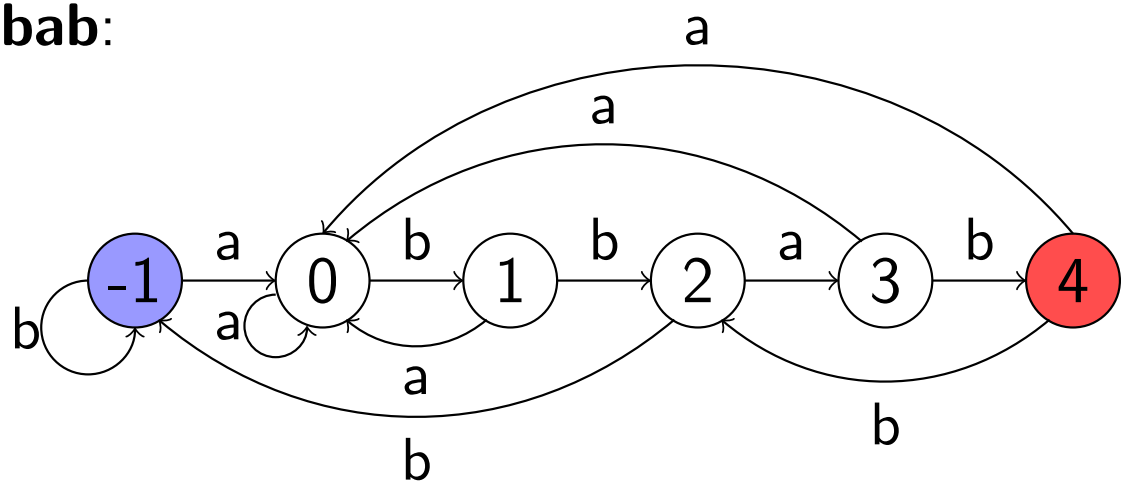
DFA

Formal ergibt sich der DFA wie folgt:

- $Q = \{-1, 0, \ldots, m-1\}$ ($m+1$ Zustände)
- $q_0 = -1$
- Σ ist das Alphabet des Textes und Patterns
- $F = \{m-1\}$
- Übergangsfunktion $\delta : Q \times \Sigma \to Q$ wie folgt: berechne eindeutige NFA-Zustandsmenge $A(q)$ mit $q = \max A(q)$. Wende hierauf Δ für c an und extrahiere das maximale Element als neuen Zustand, berechne also $\max \Delta(A(q), c)$.

DFA

DFA für **abbab**:

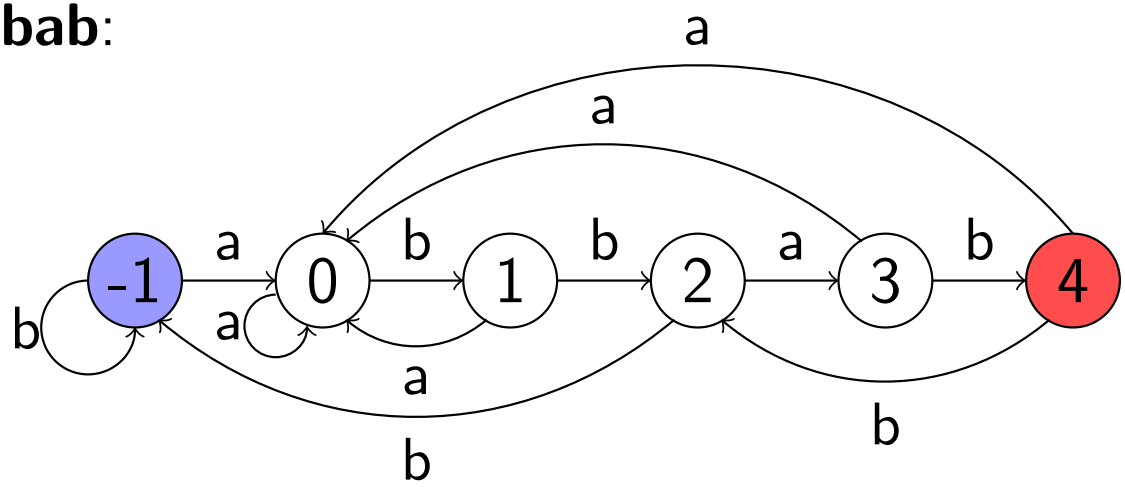


Beispiel für Zustandsübergang mit $q = 3, (a^* = 3 : \{-1, 0, 3\}), c =$ 'a':

$$\left.\begin{aligned} P[0] = \text{'a'} &\Rightarrow \{-1, 0\} \\ \Delta(0, \text{'a'}) &\Rightarrow \{\ \} \\ \Delta(3, \text{'a'}) &\Rightarrow \{\ \} \end{aligned}\right\} \text{Vereinigung: } \{-1, 0\}$$

DFA

DFA für **abbab**:



Beispiel für Zustandsübergang mit $q = 3, (a^* = 3 : \{-1, 0, 3\}), c =$ 'b':

$$\left.\begin{array}{l} P[0] \neq \text{'b'} \Rightarrow \{-1\} \\ \Delta(0, \text{'b'}) \Rightarrow \{1\} \\ \Delta(3, \text{'b'}) \Rightarrow \{4\} \end{array}\right\} \text{Vereinigung: } \{-1, 1, 4\}$$

Quellcode für DFA

```
1  def DFA_exec(P, T):
2      delta = DFA_delta_table(P)
3      q, m, n = -1, len(P), len(T)
4      for i in range(n):
5          q = delta(q, T[i])
6          if q == m - 1:
7              yield i - m + 1, i + 1
8
9  def DFA(P, T):
10     for s, e in DFA_exec(P, T):
11         print("found:", s, e)
```

Der Knuth-Morris-Pratt-Algorithmus

Merkmal	*DFA*	*KMP*
Konstruktion von δ-Tabelle	$\mathcal{O}(m^2\|\Sigma\|)$	$\mathcal{O}(m)$
Speicherplatz von δ-Tabelle	$\mathcal{O}(m\|\Sigma\|)$	$\mathcal{O}(m)$
Laufzeit für Patternsuche	$\mathcal{O}(mn)$	$\mathcal{O}(n)$

Der Knuth-Morris-Pratt-Algorithmus

Idee: Da durch $\max A(q)$ die Zustandsmenge eindeutig definiert ist, wäre es sinnvoll für q das **l**ängste vom **P**räfix von $P[:q]$, das echtes aktives **S**uffix von $P[:q]$ ist, abzuspeichern.

Der Knuth-Morris-Pratt-Algorithmus

Idee: Da durch $\max A(q)$ die Zustandsmenge eindeutig definiert ist, wäre es sinnvoll für q das **l**ängste vom **P**räfix von $P[: q]$, das echtes aktives **S**uffix von $P[: q]$ ist, abzuspeichern.

Definition (lps-Funktion)

Zu $P \in \Sigma^m$ definieren wir $\text{lps} : \{0, \ldots, m-1\} \to \mathbb{N}$ folgendermaßen:

$$\text{lps}(q) := \max\{|s| \leq q \ : \ s \text{ ist Präfix von } P \text{ und Suffix von } P[: q]\}.$$

Der Knuth-Morris-Pratt-Algorithmus

Beispiel: lps-Funktion

q	0	1	2	3	4	5	6
P[q]	a	b	a	b	a	c	a
lps[q]	0	0	1	2	3	0	1

In der obersten Zeile steht der Index der Position, darunter das Pattern P und darunter der Wert von lps an dieser Stelle. Es gilt:

$$A(q) = \{q, \mathrm{lps}(q) - 1, \mathrm{lps}(\mathrm{lps}(q) - 1) - 1, \ldots, -1\}.$$

Der Knuth-Morris-Pratt-Algorithmus

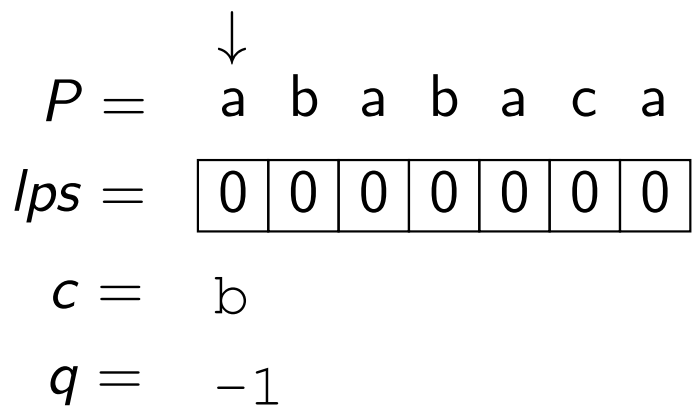
Der Algorithmus im Sinne des DFA:

```
1  def create_lps(P):
2      m, q, lps = len(P), -1, [0] * len(P)
3      for i in range(1, m):
4          q = delta(q, P[i], P, lps)
5          lps[i] = q + 1
6      return lps
7
8  def DFA_exec(P, T):
9      q, m, n, lps = -1, len(P), len(T), create_lps(P)
10     for i in range(n):
11         q = delta(q, T[i], P, lps)
12         if q == m - 1:
13             yield i - m + 1, i + 1
```

Der Knuth-Morris-Pratt-Algorithmus

Erstellung der lps-Tabelle mit der Deltafunktion:

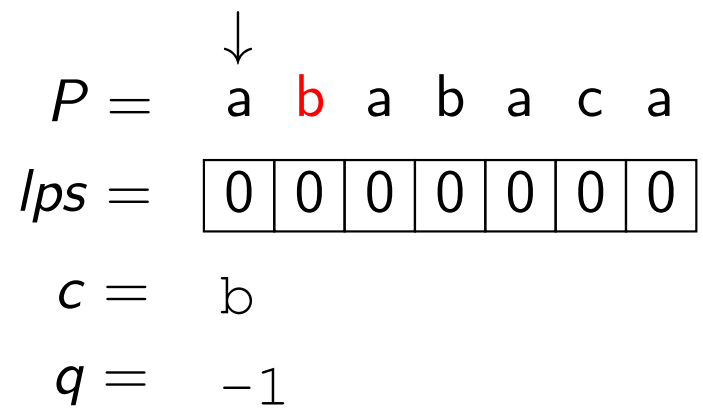
```
1 def delta(q, c, P, lps):
2     m = len(P)
3     while q == m - 1 or (P[q + 1] != c and q > -1):
4         q = lps[q] - 1
5     if P[q + 1] == c:  q += 1
6     return q
```



Der Knuth-Morris-Pratt-Algorithmus

Erstellung der lps-Tabelle mit der Deltafunktion:

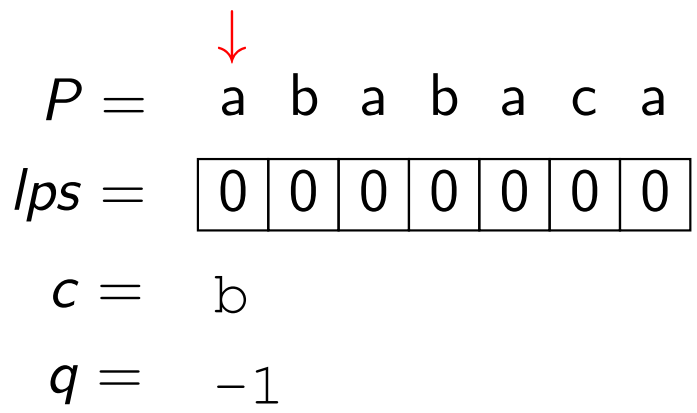
```
1 def delta(q, c, P, lps):
2     m = len(P)
3     while q == m - 1 or (P[q + 1] != c and q > -1):
4         q = lps[q] - 1
5     if P[q + 1] == c:  q += 1
6     return q
```



Der Knuth-Morris-Pratt-Algorithmus

Erstellung der lps-Tabelle mit der Deltafunktion:

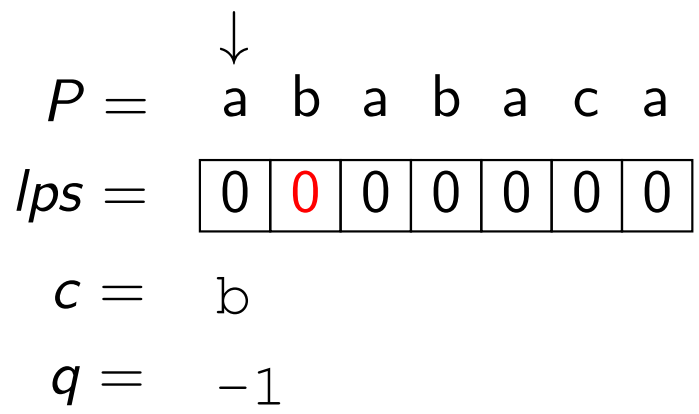
```
1 def delta(q, c, P, lps):
2     m = len(P)
3     while q == m - 1 or (P[q + 1] != c and q > -1):
4         q = lps[q] - 1
5     if P[q + 1] == c:  q += 1
6     return q
```



Der Knuth-Morris-Pratt-Algorithmus

Erstellung der lps-Tabelle mit der Deltafunktion:

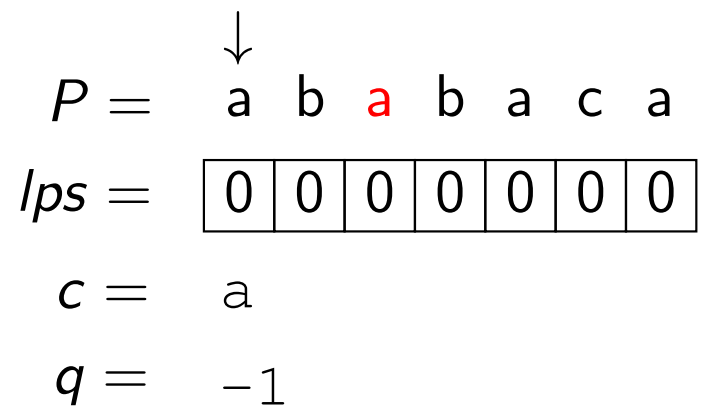
```
1 def delta(q, c, P, lps):
2     m = len(P)
3     while q == m - 1 or (P[q + 1] != c and q > -1):
4         q = lps[q] - 1
5     if P[q + 1] == c:  q += 1
6     return q
```



Der Knuth-Morris-Pratt-Algorithmus

Erstellung der lps-Tabelle mit der Deltafunktion:

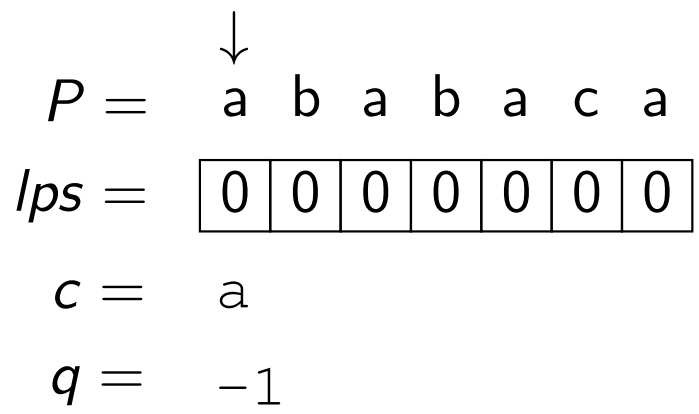
```
1 def delta(q, c, P, lps):
2     m = len(P)
3     while q == m - 1 or (P[q + 1] != c and q > -1):
4         q = lps[q] - 1
5     if P[q + 1] == c:  q += 1
6     return q
```



Der Knuth-Morris-Pratt-Algorithmus

Erstellung der lps-Tabelle mit der Deltafunktion:

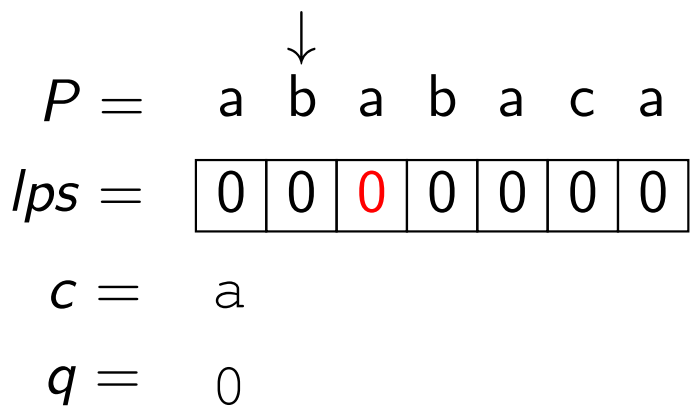
```
1 def delta(q, c, P, lps):
2     m = len(P)
3     while q == m - 1 or (P[q + 1] != c and q > -1):
4         q = lps[q] - 1
5     if P[q + 1] == c:  q += 1
6     return q
```



Der Knuth-Morris-Pratt-Algorithmus

Erstellung der lps-Tabelle mit der Deltafunktion:

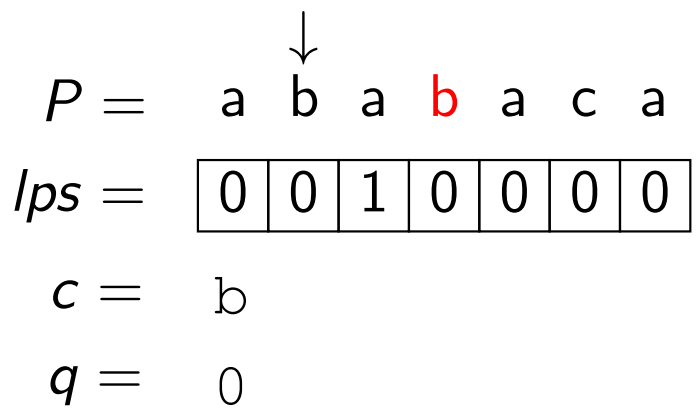
```
1 def delta(q, c, P, lps):
2     m = len(P)
3     while q == m - 1 or (P[q + 1] != c and q > -1):
4         q = lps[q] - 1
5     if P[q + 1] == c:  q += 1
6     return q
```



Der Knuth-Morris-Pratt-Algorithmus

Erstellung der lps-Tabelle mit der Deltafunktion:

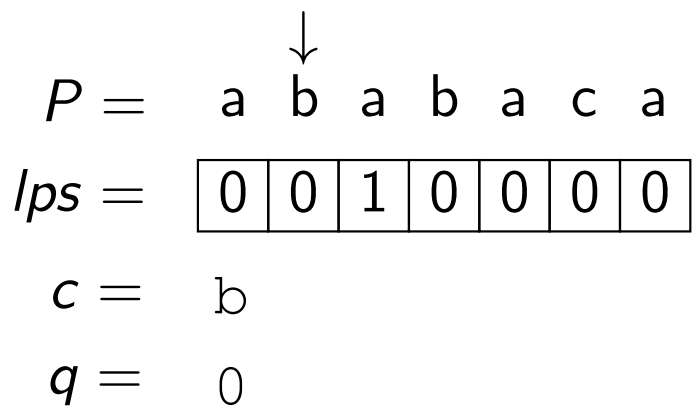
```
1 def delta(q, c, P, lps):
2     m = len(P)
3     while q == m - 1 or (P[q + 1] != c and q > -1):
4         q = lps[q] - 1
5     if P[q + 1] == c:  q += 1
6     return q
```



Der Knuth-Morris-Pratt-Algorithmus

Erstellung der lps-Tabelle mit der Deltafunktion:

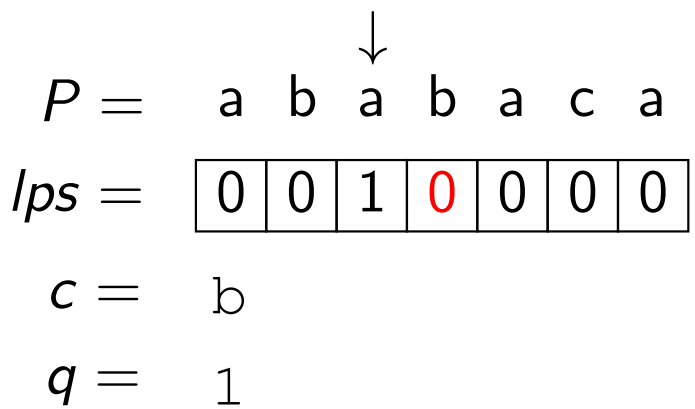
```
1 def delta(q, c, P, lps):
2     m = len(P)
3     while q == m - 1 or (P[q + 1] != c and q > -1):
4         q = lps[q] - 1
5     if P[q + 1] == c:  q += 1
6     return q
```



Der Knuth-Morris-Pratt-Algorithmus

Erstellung der lps-Tabelle mit der Deltafunktion:

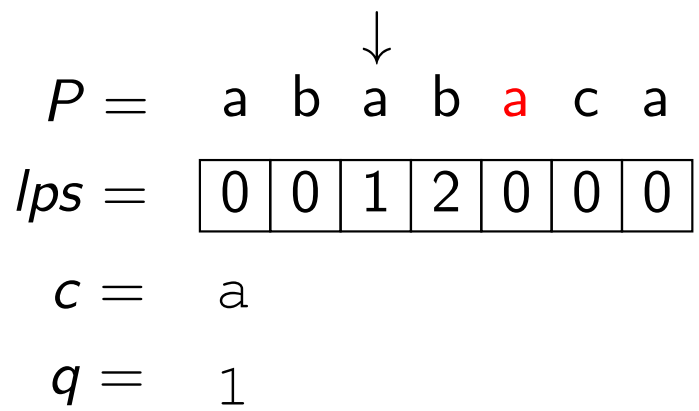
```
1 def delta(q, c, P, lps):
2     m = len(P)
3     while q == m - 1 or (P[q + 1] != c and q > -1):
4         q = lps[q] - 1
5     if P[q + 1] == c:  q += 1
6     return q
```



Der Knuth-Morris-Pratt-Algorithmus

Erstellung der lps-Tabelle mit der Deltafunktion:

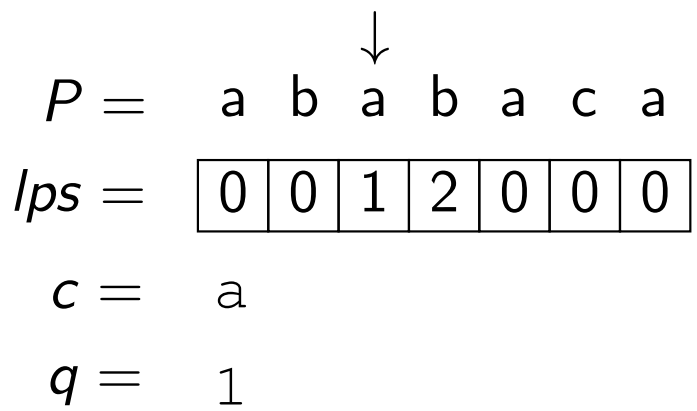
```
1 def delta(q, c, P, lps):
2     m = len(P)
3     while q == m - 1 or (P[q + 1] != c and q > -1):
4         q = lps[q] - 1
5     if P[q + 1] == c:  q += 1
6     return q
```



Der Knuth-Morris-Pratt-Algorithmus

Erstellung der lps-Tabelle mit der Deltafunktion:

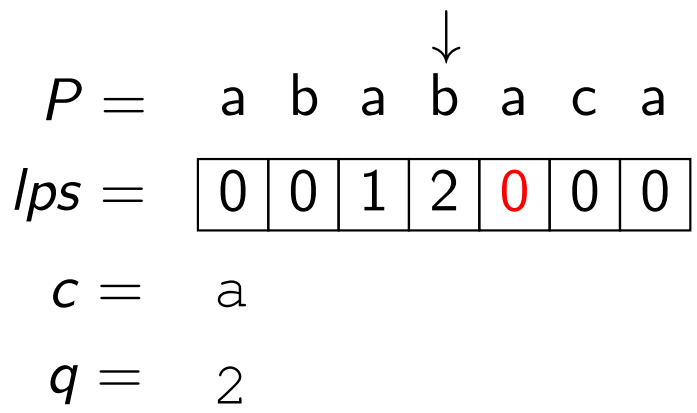
```
1 def delta(q, c, P, lps):
2     m = len(P)
3     while q == m - 1 or (P[q + 1] != c and q > -1):
4         q = lps[q] - 1
5     if P[q + 1] == c:  q += 1
6     return q
```



Der Knuth-Morris-Pratt-Algorithmus

Erstellung der lps-Tabelle mit der Deltafunktion:

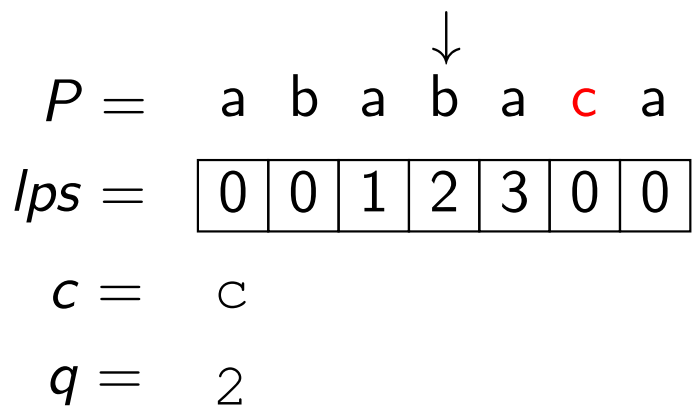
```
1 def delta(q, c, P, lps):
2     m = len(P)
3     while q == m - 1 or (P[q + 1] != c and q > -1):
4         q = lps[q] - 1
5     if P[q + 1] == c:  q += 1
6     return q
```



Der Knuth-Morris-Pratt-Algorithmus

Erstellung der lps-Tabelle mit der Deltafunktion:

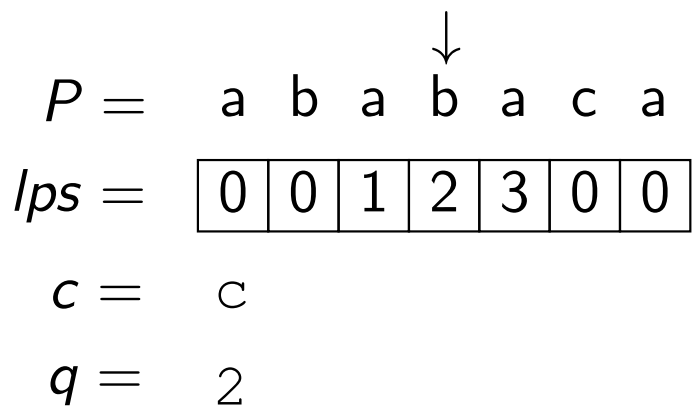
```
1 def delta(q, c, P, lps):
2     m = len(P)
3     while q == m - 1 or (P[q + 1] != c and q > -1):
4         q = lps[q] - 1
5     if P[q + 1] == c:  q += 1
6     return q
```



Der Knuth-Morris-Pratt-Algorithmus

Erstellung der lps-Tabelle mit der Deltafunktion:

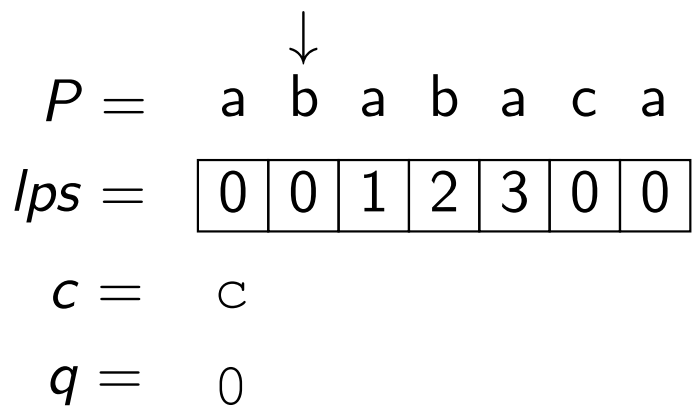
```
1 def delta(q, c, P, lps):
2     m = len(P)
3     while q == m - 1 or (P[q + 1] != c and q > -1):
4         q = lps[q] - 1
5     if P[q + 1] == c:  q += 1
6     return q
```



Der Knuth-Morris-Pratt-Algorithmus

Erstellung der lps-Tabelle mit der Deltafunktion:

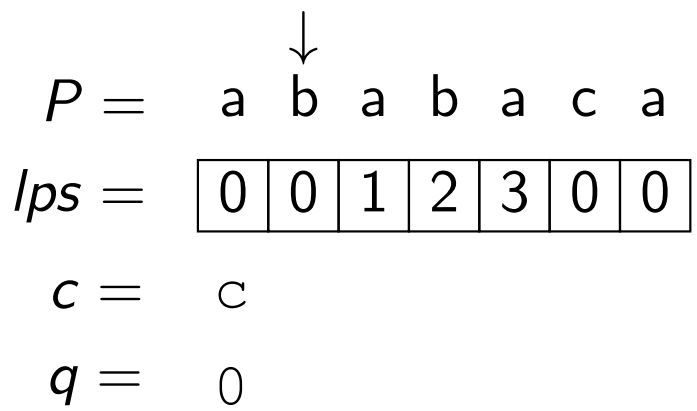
```
1 def delta(q, c, P, lps):
2     m = len(P)
3     while q == m - 1 or (P[q + 1] != c and q > -1):
4         q = lps[q] - 1
5     if P[q + 1] == c:  q += 1
6     return q
```



Der Knuth-Morris-Pratt-Algorithmus

Erstellung der lps-Tabelle mit der Deltafunktion:

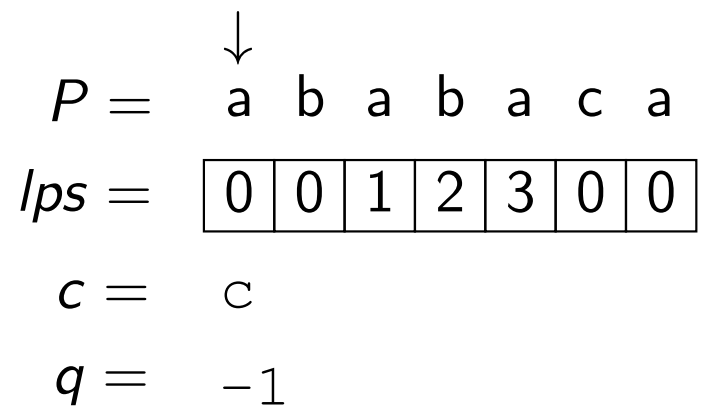
```
1 def delta(q, c, P, lps):
2     m = len(P)
3     while q == m - 1 or (P[q + 1] != c and q > -1):
4         q = lps[q] - 1
5     if P[q + 1] == c:  q += 1
6     return q
```



Der Knuth-Morris-Pratt-Algorithmus

Erstellung der lps-Tabelle mit der Deltafunktion:

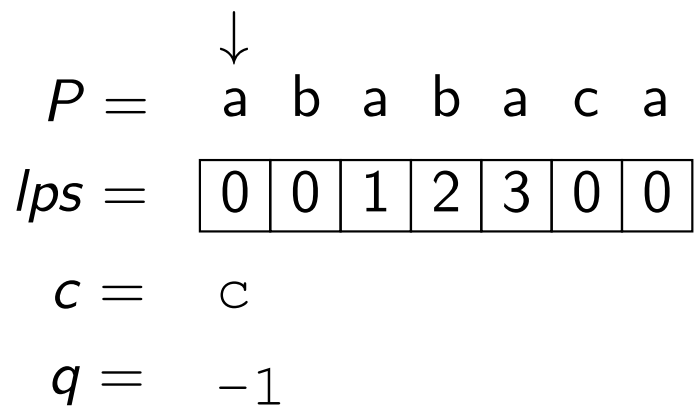
```
1 def delta(q, c, P, lps):
2     m = len(P)
3     while q == m - 1 or (P[q + 1] != c and q > -1):
4         q = lps[q] - 1
5     if P[q + 1] == c:  q += 1
6     return q
```



Der Knuth-Morris-Pratt-Algorithmus

Erstellung der lps-Tabelle mit der Deltafunktion:

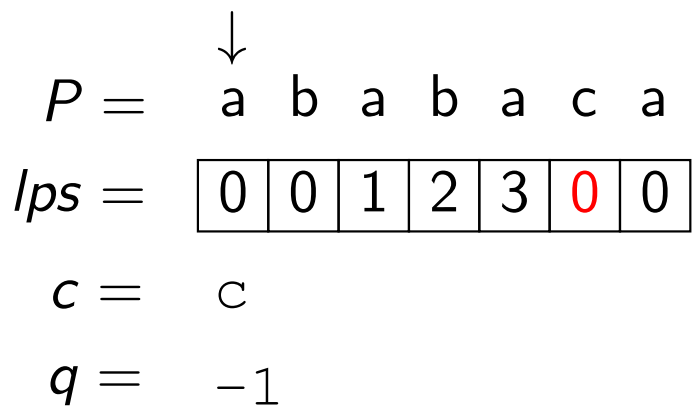
```
1 def delta(q, c, P, lps):
2     m = len(P)
3     while q == m - 1 or (P[q + 1] != c and q > -1):
4         q = lps[q] - 1
5     if P[q + 1] == c:  q += 1
6     return q
```



Der Knuth-Morris-Pratt-Algorithmus

Erstellung der lps-Tabelle mit der Deltafunktion:

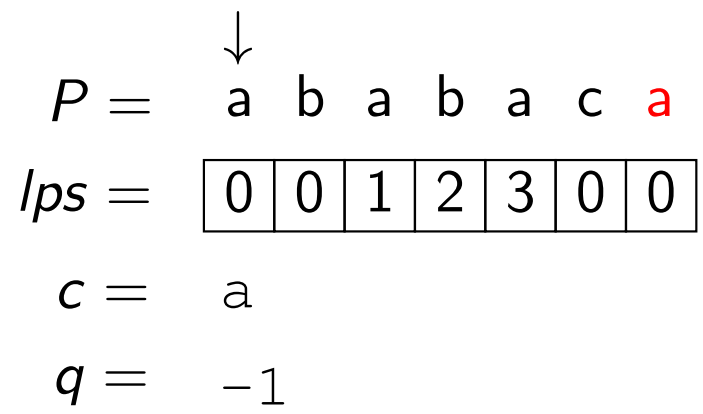
```
1 def delta(q, c, P, lps):
2     m = len(P)
3     while q == m - 1 or (P[q + 1] != c and q > -1):
4         q = lps[q] - 1
5     if P[q + 1] == c:  q += 1
6     return q
```



Der Knuth-Morris-Pratt-Algorithmus

Erstellung der lps-Tabelle mit der Deltafunktion:

```
1 def delta(q, c, P, lps):
2     m = len(P)
3     while q == m - 1 or (P[q + 1] != c and q > -1):
4         q = lps[q] - 1
5     if P[q + 1] == c:  q += 1
6     return q
```



Der Knuth-Morris-Pratt-Algorithmus

Erstellung der lps-Tabelle mit der Deltafunktion:

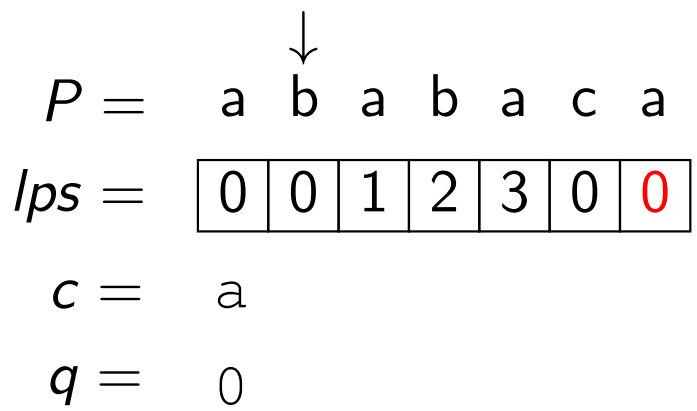
```
1 def delta(q, c, P, lps):
2     m = len(P)
3     while q == m - 1 or (P[q + 1] != c and q > -1):
4         q = lps[q] - 1
5     if P[q + 1] == c:  q += 1
6     return q
```

$$
\begin{aligned}
P &= \begin{array}{c} \downarrow \\ \mathtt{a\ b\ a\ b\ a\ c\ a} \end{array} \\
lps &= \begin{array}{|c|c|c|c|c|c|c|} \hline 0 & 0 & 1 & 2 & 3 & 0 & 0 \\ \hline \end{array} \\
c &= \mathtt{a} \\
q &= -1
\end{aligned}
$$

Der Knuth-Morris-Pratt-Algorithmus

Erstellung der lps-Tabelle mit der Deltafunktion:

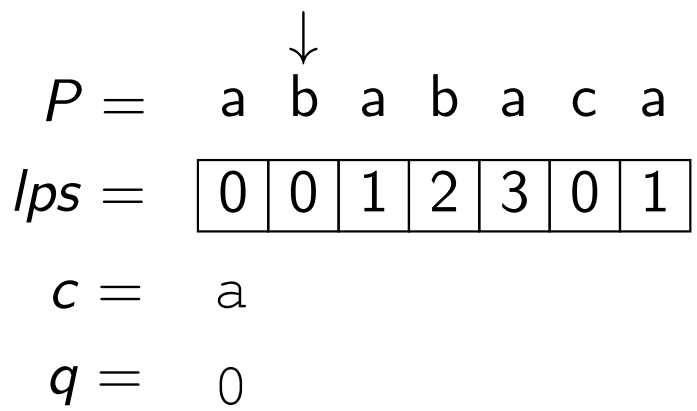
```
1 def delta(q, c, P, lps):
2     m = len(P)
3     while q == m - 1 or (P[q + 1] != c and q > -1):
4         q = lps[q] - 1
5     if P[q + 1] == c:  q += 1
6     return q
```



Der Knuth-Morris-Pratt-Algorithmus

Erstellung der lps-Tabelle mit der Deltafunktion:

```
1 def delta(q, c, P, lps):
2     m = len(P)
3     while q == m - 1 or (P[q + 1] != c and q > -1):
4         q = lps[q] - 1
5     if P[q + 1] == c:  q += 1
6     return q
```



Der Knuth-Morris-Pratt-Algorithmus

Erstellung der lps-Tabelle mit der Deltafunktion:

```
1 def delta(q, c, P, lps):
2     m = len(P)
3     while q == m - 1 or (P[q + 1] != c and q > -1):
4         q = lps[q] - 1
5     if P[q + 1] == c:  q += 1
6     return q
```

Die (amortisierte) Laufzeit beträgt $\mathcal{O}(m)$.

- Pro delta-Aufruf wird q max. um 1 erhöht.
- $0 \leq \text{lps}[q] < q \quad \rightarrow \quad q \geq -1$
- While-Schleife kann max. m mal betreten werden.

Der Knuth-Morris-Pratt-Algorithmus

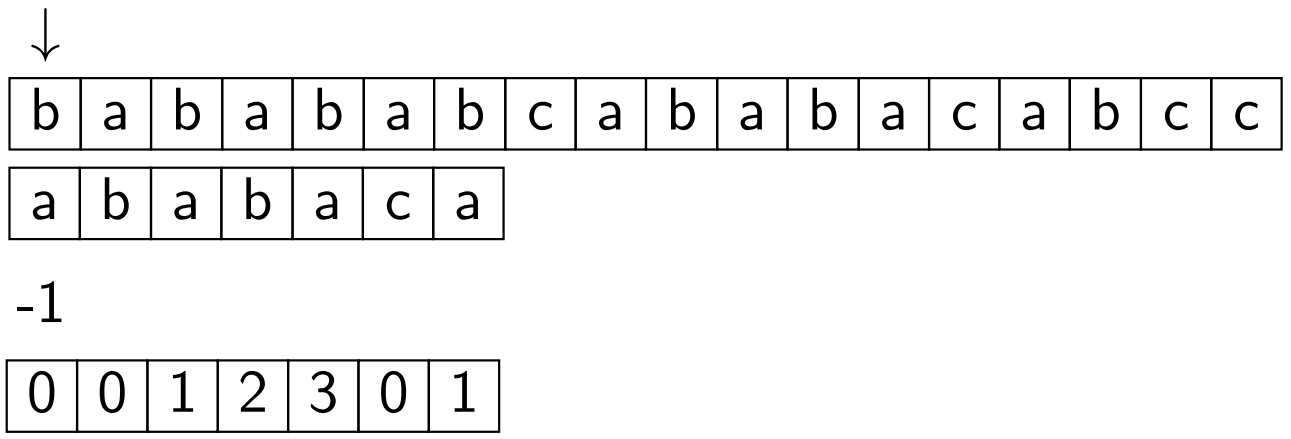
Mustersuche mit dem KMP-Algorithmus:

```
1 def DFA_exec(P, T):
2     q, m, n, lps = -1, len(P), len(T), create_lps(P)
3     for i in range(n):
4         q = delta(q, T[i], P, lps)
5         if q == m - 1:
6             yield i - m + 1, i + 1
```

Der Knuth-Morris-Pratt-Algorithmus

Mustersuche mit dem KMP:

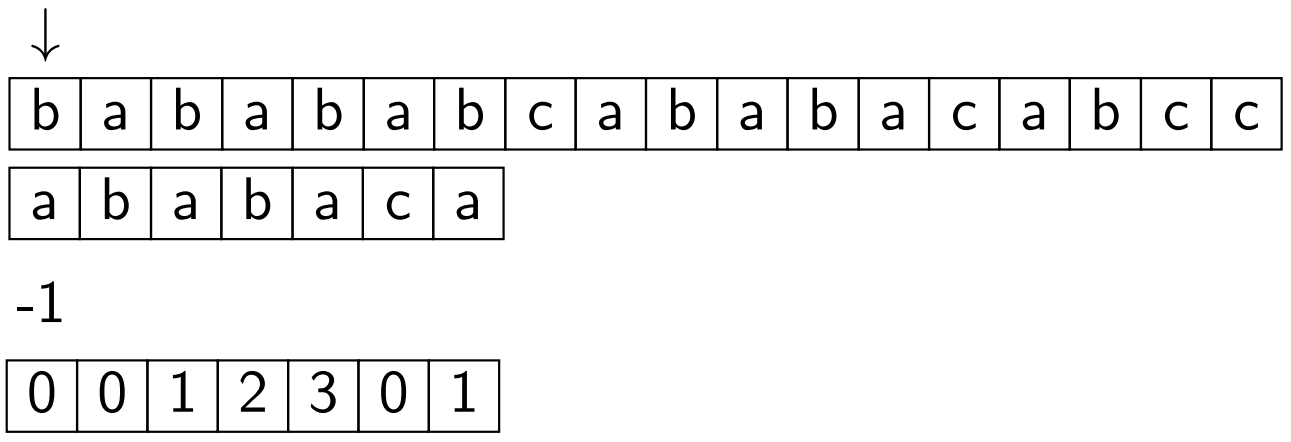
```
1 def delta(q, c, P, lps):
2     m = len(P)
3     while q == m - 1 or (P[q + 1] != c and q > -1):
4         q = lps[q] - 1
5     if P[q + 1] == c:  q += 1
6     return q
```



Der Knuth-Morris-Pratt-Algorithmus

Mustersuche mit dem KMP:

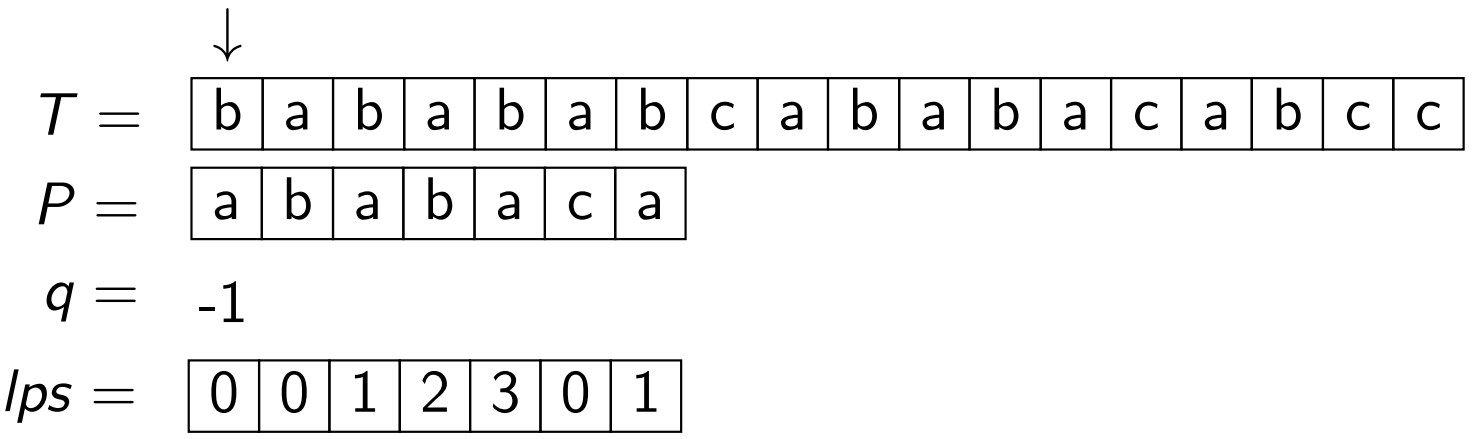
```
1 def delta(q, c, P, lps):
2     m = len(P)
3     while q == m - 1 or (P[q + 1] != c and q > -1):
4         q = lps[q] - 1
5     if P[q + 1] == c:  q += 1
6     return q
```



Der Knuth-Morris-Pratt-Algorithmus

Mustersuche mit dem KMP:

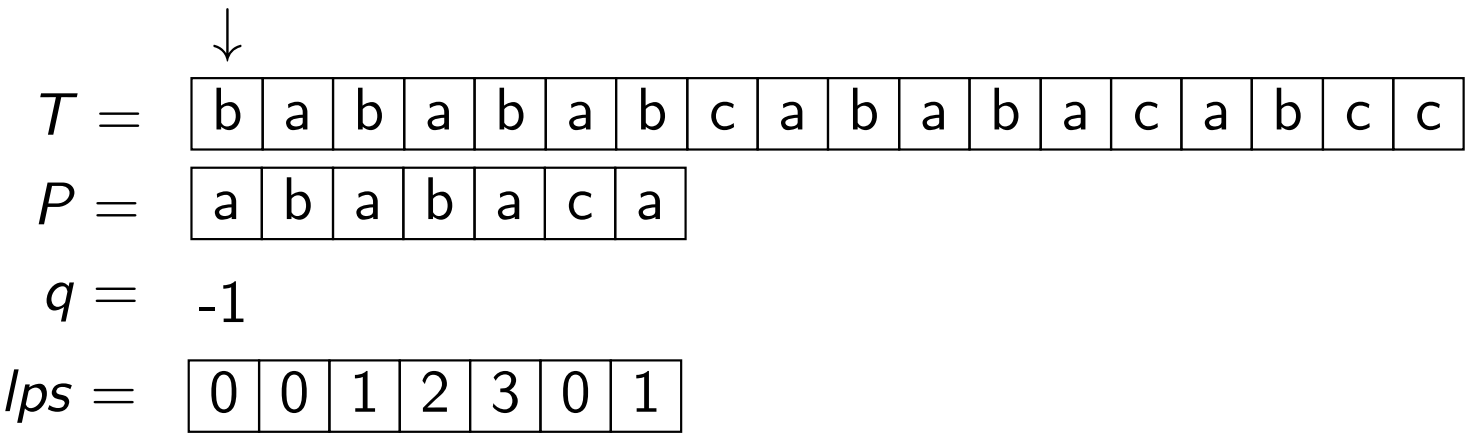
```
1 def delta(q, c, P, lps):
2     m = len(P)
3     while q == m - 1 or (P[q + 1] != c and q > -1):
4         q = lps[q] - 1
5     if P[q + 1] == c:  q += 1
6     return q
```



Der Knuth-Morris-Pratt-Algorithmus

Mustersuche mit dem KMP:

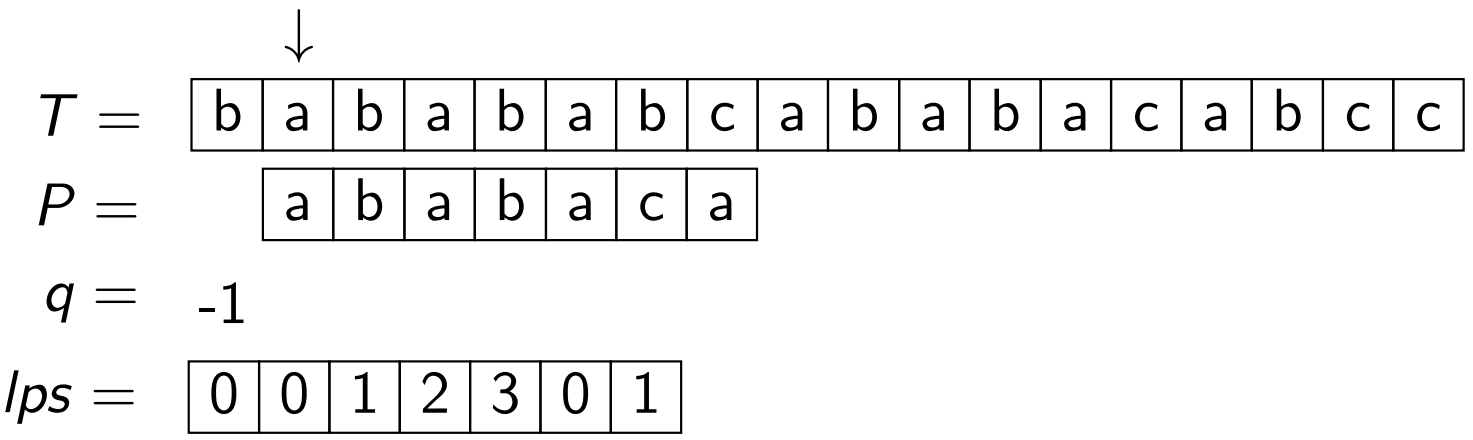
```
1 def delta(q, c, P, lps):
2     m = len(P)
3     while q == m - 1 or (P[q + 1] != c and q > -1):
4         q = lps[q] - 1
5     if P[q + 1] == c:  q += 1
6     return q
```



Der Knuth-Morris-Pratt-Algorithmus

Mustersuche mit dem KMP:

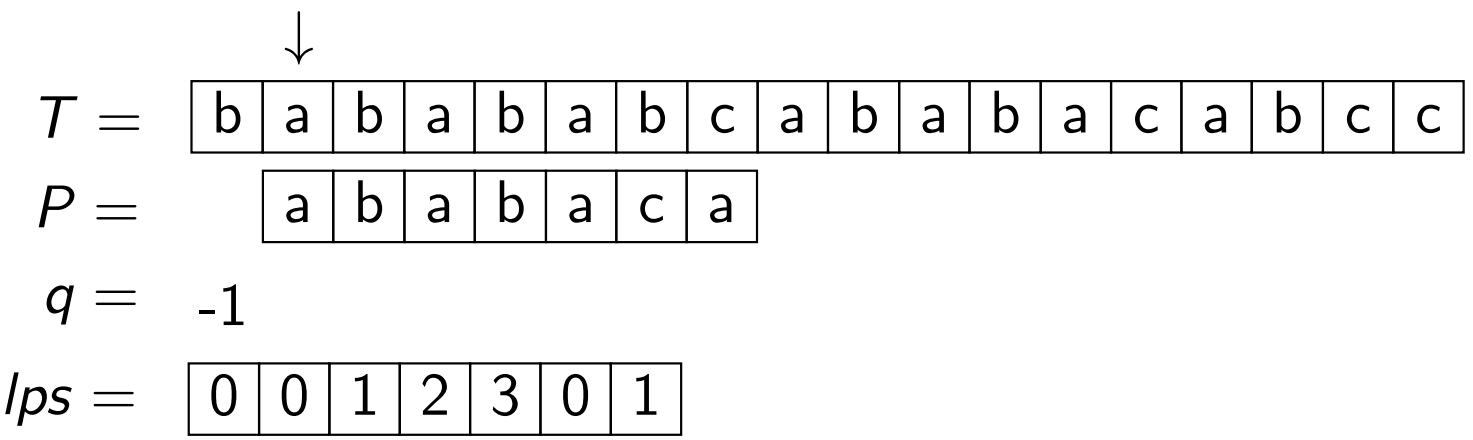
```
1 def delta(q, c, P, lps):
2     m = len(P)
3     while q == m - 1 or (P[q + 1] != c and q > -1):
4         q = lps[q] - 1
5     if P[q + 1] == c:  q += 1
6     return q
```



Der Knuth-Morris-Pratt-Algorithmus

Mustersuche mit dem KMP:

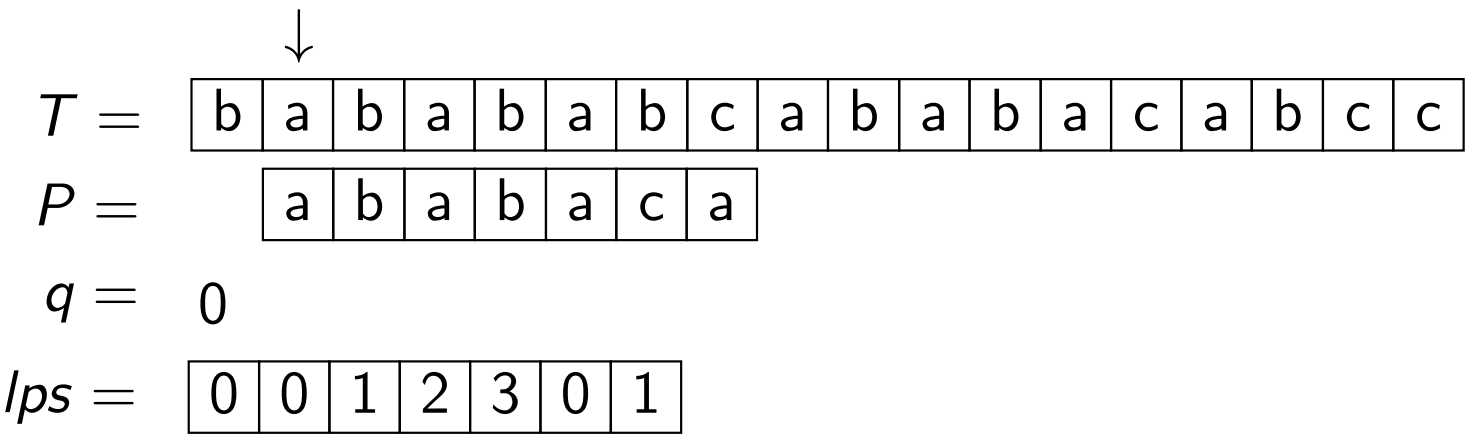
```
1 def delta(q, c, P, lps):
2     m = len(P)
3     while q == m - 1 or (P[q + 1] != c and q > -1):
4         q = lps[q] - 1
5     if P[q + 1] == c:  q += 1
6     return q
```



Der Knuth-Morris-Pratt-Algorithmus

Mustersuche mit dem KMP:

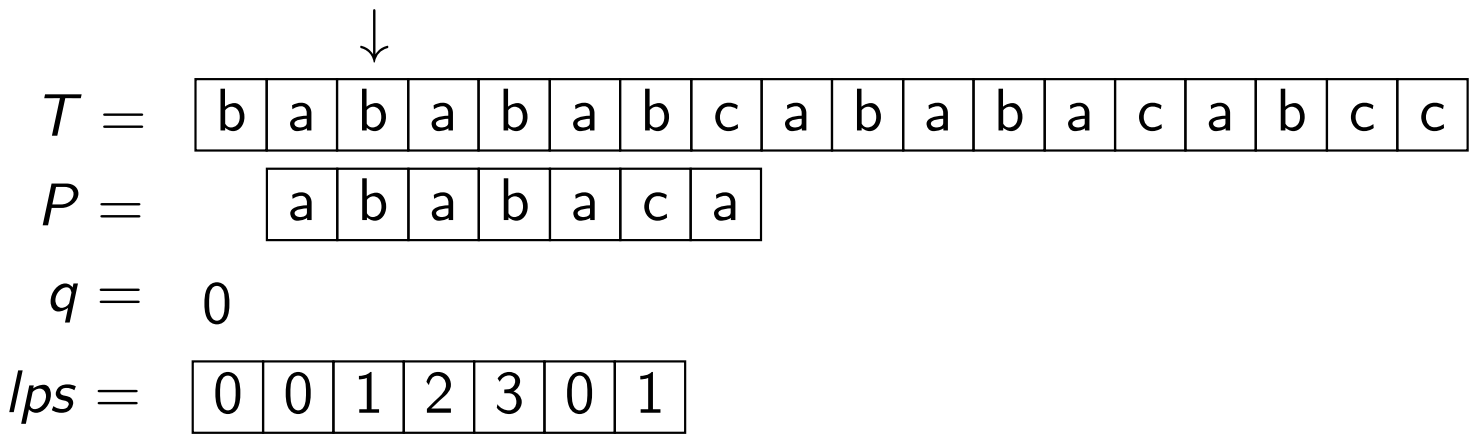
```
1 def delta(q, c, P, lps):
2     m = len(P)
3     while q == m - 1 or (P[q + 1] != c and q > -1):
4         q = lps[q] - 1
5     if P[q + 1] == c:  q += 1
6     return q
```



Der Knuth-Morris-Pratt-Algorithmus

Mustersuche mit dem KMP:

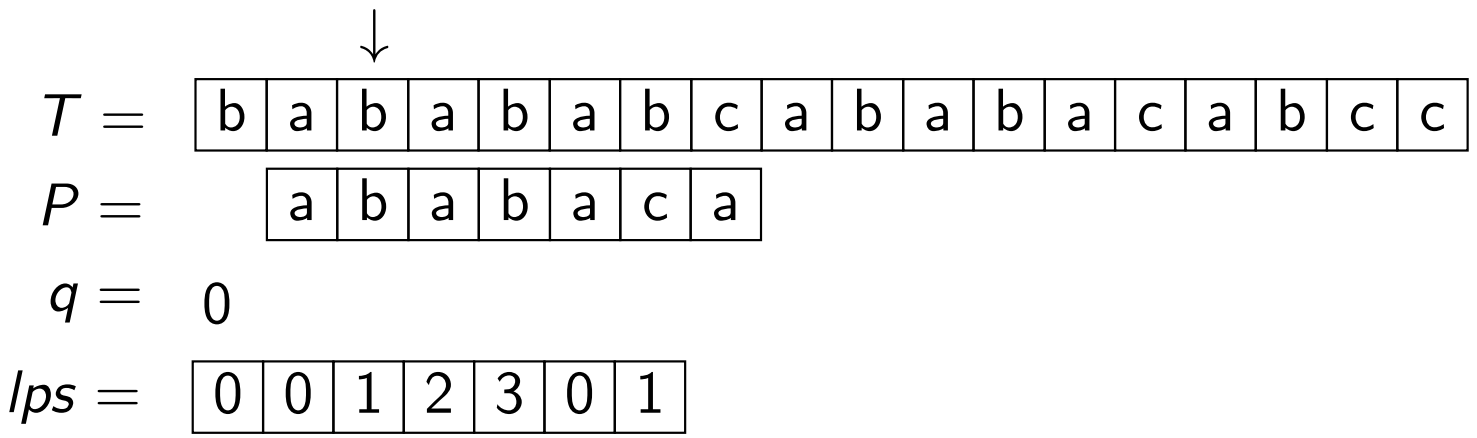
```
1 def delta(q, c, P, lps):
2     m = len(P)
3     while q == m - 1 or (P[q + 1] != c and q > -1):
4         q = lps[q] - 1
5     if P[q + 1] == c:  q += 1
6     return q
```



Der Knuth-Morris-Pratt-Algorithmus

Mustersuche mit dem KMP:

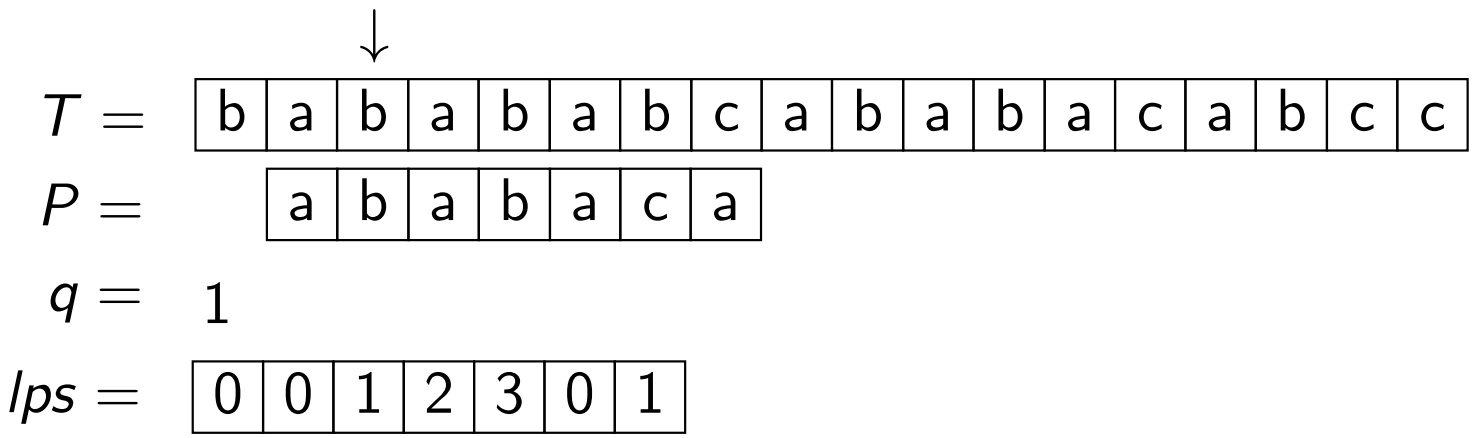
```
1 def delta(q, c, P, lps):
2     m = len(P)
3     while q == m - 1 or (P[q + 1] != c and q > -1):
4         q = lps[q] - 1
5     if P[q + 1] == c:  q += 1
6     return q
```



Der Knuth-Morris-Pratt-Algorithmus

Mustersuche mit dem KMP:

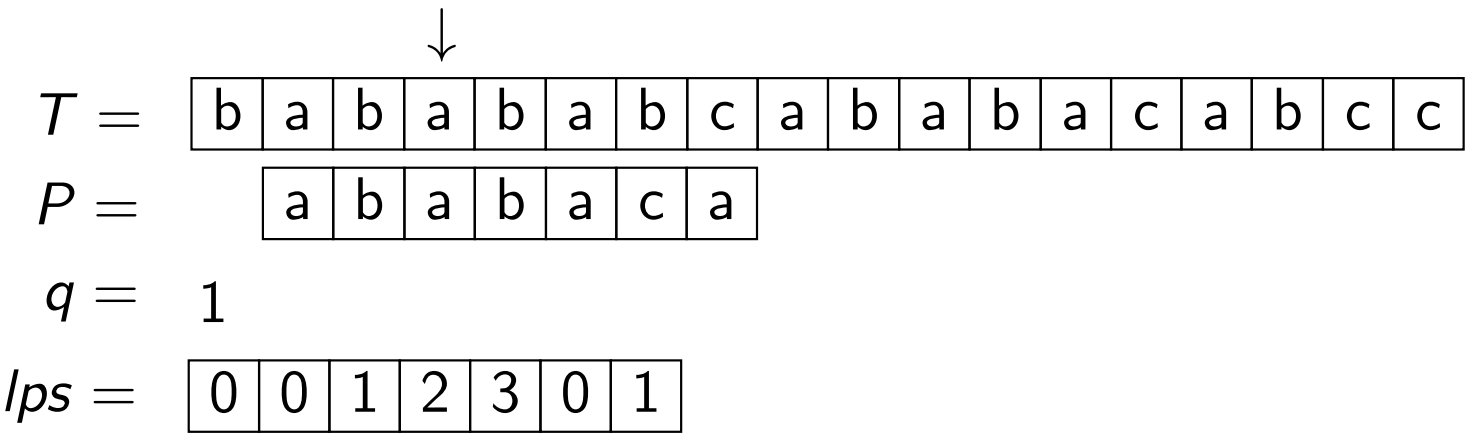
```
1 def delta(q, c, P, lps):
2     m = len(P)
3     while q == m - 1 or (P[q + 1] != c and q > -1):
4         q = lps[q] - 1
5     if P[q + 1] == c:  q += 1
6     return q
```



Der Knuth-Morris-Pratt-Algorithmus

Mustersuche mit dem KMP:

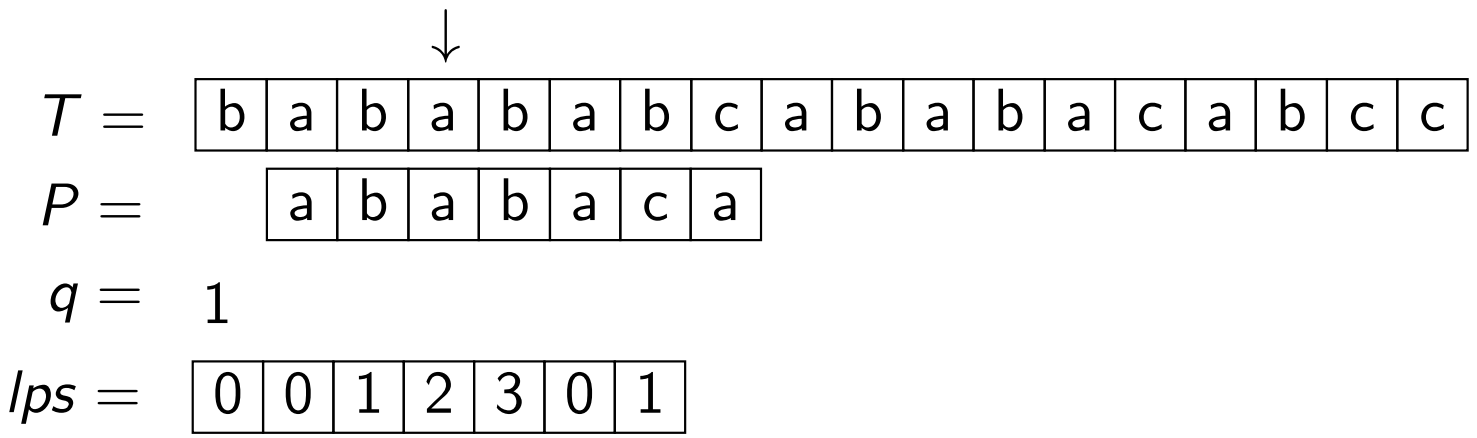
```
1 def delta(q, c, P, lps):
2     m = len(P)
3     while q == m - 1 or (P[q + 1] != c and q > -1):
4         q = lps[q] - 1
5     if P[q + 1] == c:  q += 1
6     return q
```



Der Knuth-Morris-Pratt-Algorithmus

Mustersuche mit dem KMP:

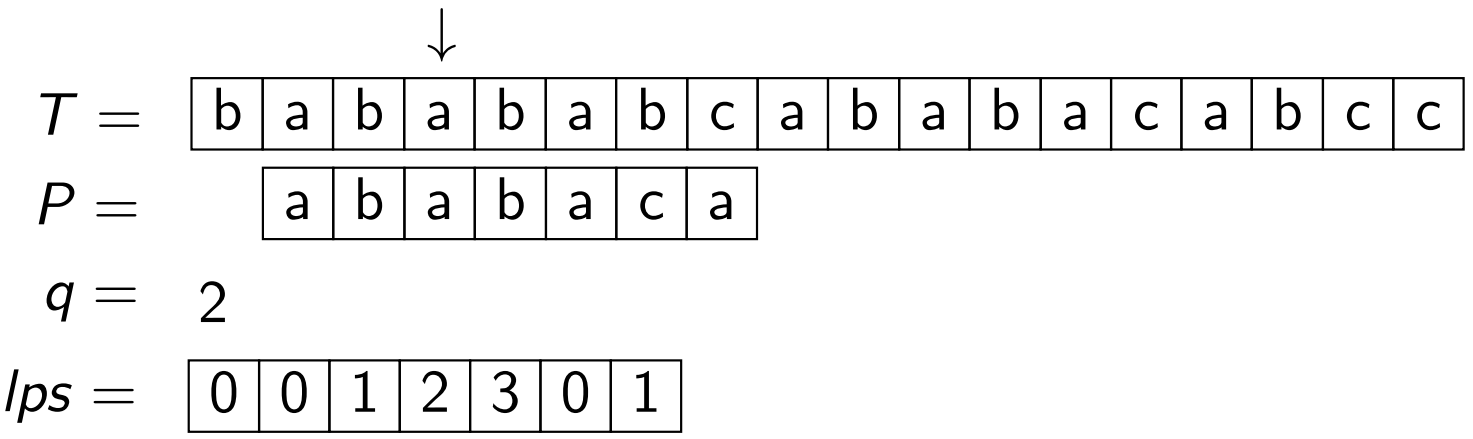
```
1 def delta(q, c, P, lps):
2     m = len(P)
3     while q == m - 1 or (P[q + 1] != c and q > -1):
4         q = lps[q] - 1
5     if P[q + 1] == c:  q += 1
6     return q
```



Der Knuth-Morris-Pratt-Algorithmus

Mustersuche mit dem KMP:

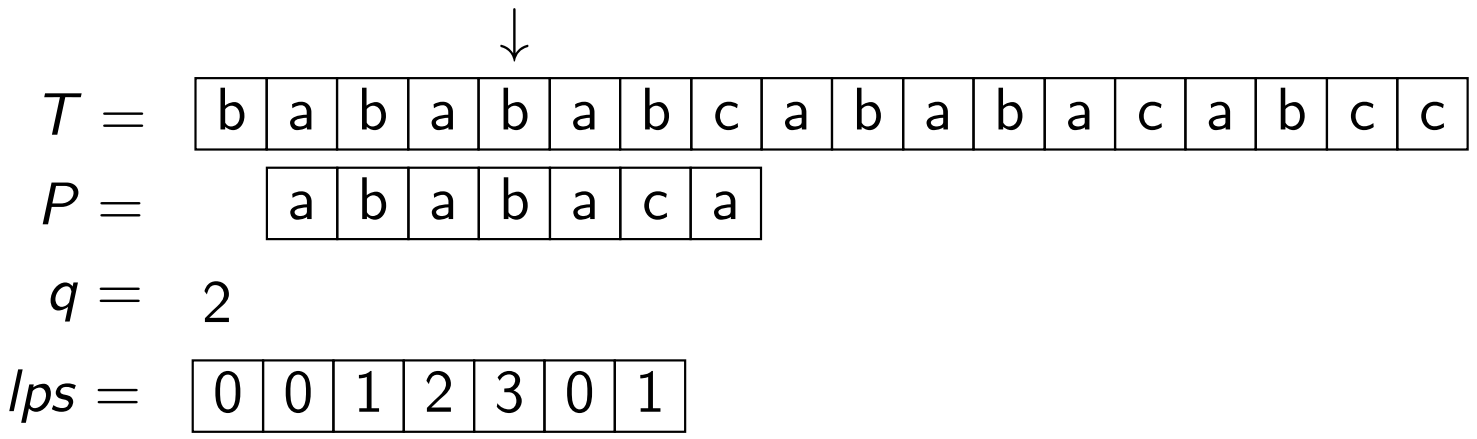
```
1 def delta(q, c, P, lps):
2     m = len(P)
3     while q == m - 1 or (P[q + 1] != c and q > -1):
4         q = lps[q] - 1
5     if P[q + 1] == c:  q += 1
6     return q
```



Der Knuth-Morris-Pratt-Algorithmus

Mustersuche mit dem KMP:

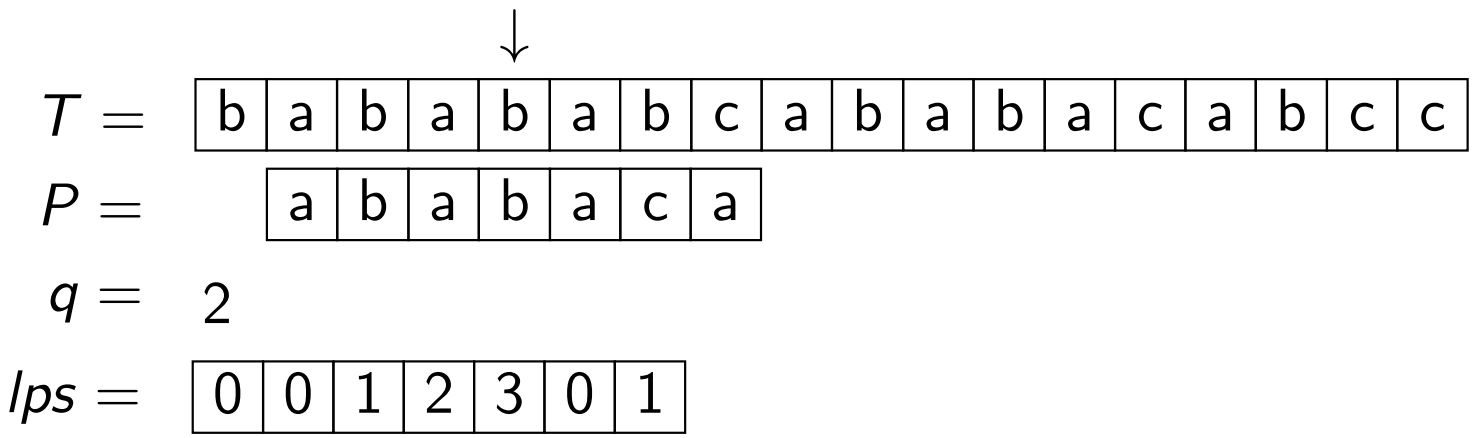
```
1 def delta(q, c, P, lps):
2     m = len(P)
3     while q == m - 1 or (P[q + 1] != c and q > -1):
4         q = lps[q] - 1
5     if P[q + 1] == c:  q += 1
6     return q
```



Der Knuth-Morris-Pratt-Algorithmus

Mustersuche mit dem KMP:

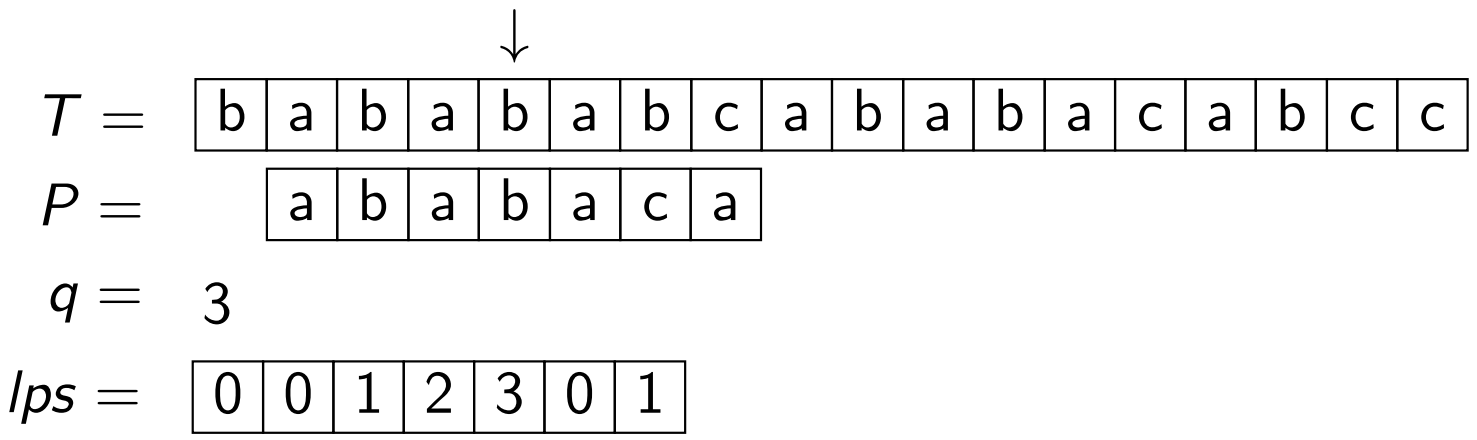
```
1 def delta(q, c, P, lps):
2     m = len(P)
3     while q == m - 1 or (P[q + 1] != c and q > -1):
4         q = lps[q] - 1
5     if P[q + 1] == c:  q += 1
6     return q
```



Der Knuth-Morris-Pratt-Algorithmus

Mustersuche mit dem KMP:

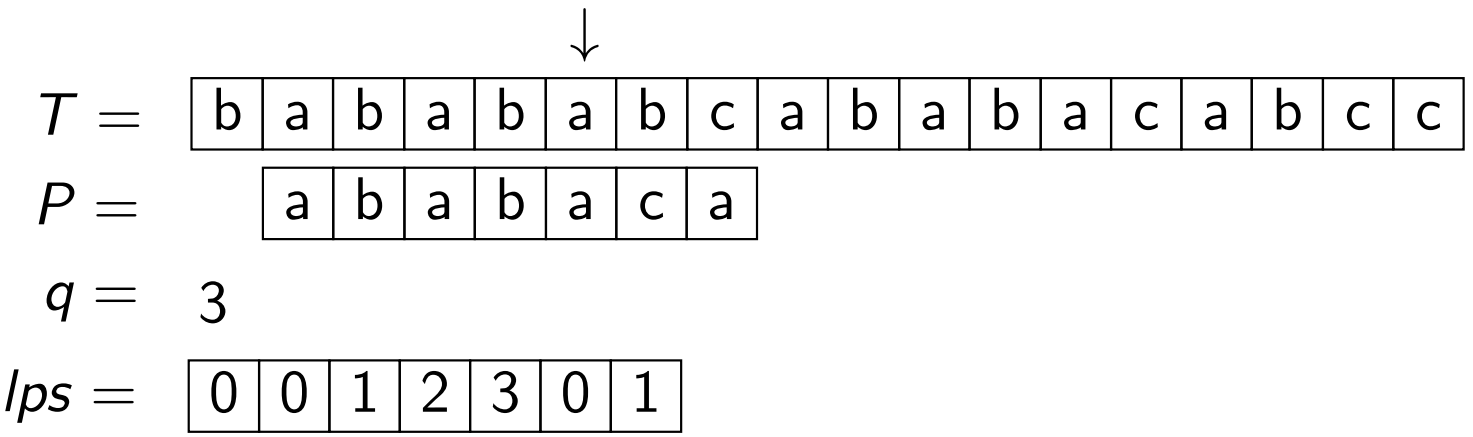
```
1 def delta(q, c, P, lps):
2     m = len(P)
3     while q == m - 1 or (P[q + 1] != c and q > -1):
4         q = lps[q] - 1
5     if P[q + 1] == c:  q += 1
6     return q
```



Der Knuth-Morris-Pratt-Algorithmus

Mustersuche mit dem KMP:

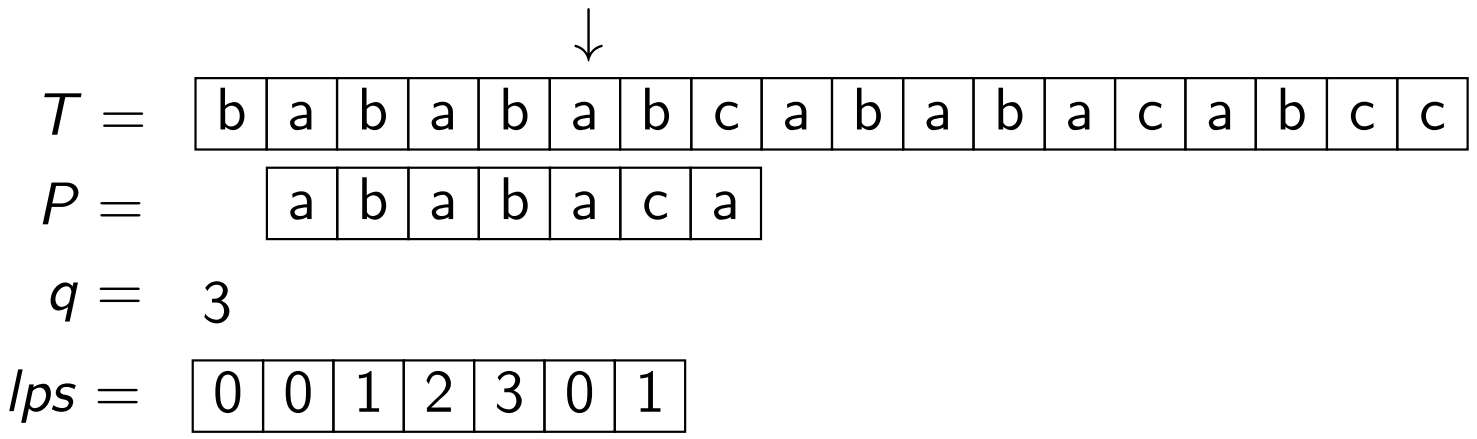
```
1 def delta(q, c, P, lps):
2     m = len(P)
3     while q == m - 1 or (P[q + 1] != c and q > -1):
4         q = lps[q] - 1
5     if P[q + 1] == c:  q += 1
6     return q
```



Der Knuth-Morris-Pratt-Algorithmus

Mustersuche mit dem KMP:

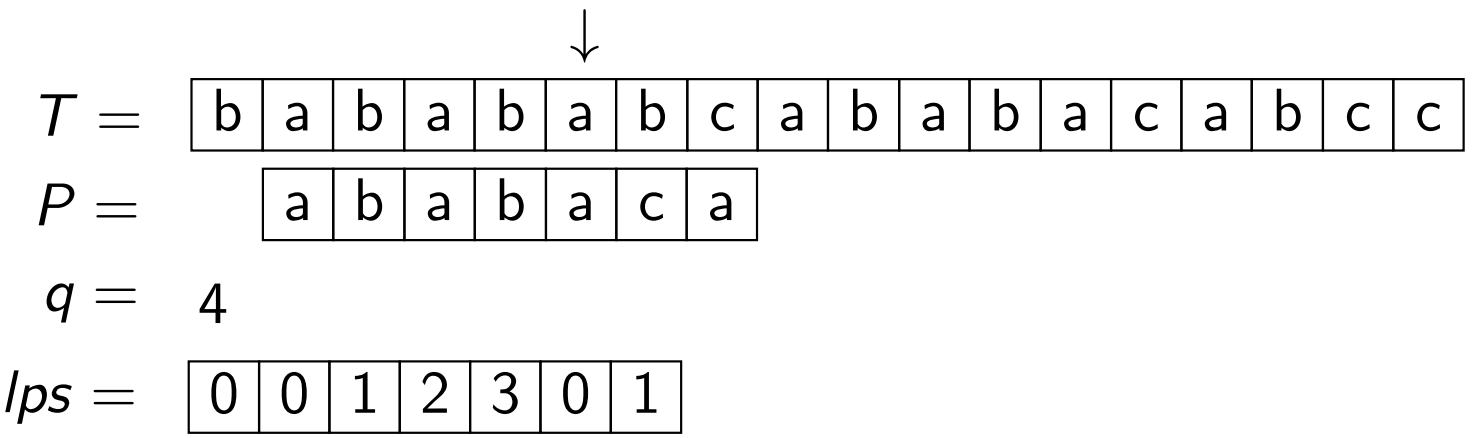
```
1 def delta(q, c, P, lps):
2     m = len(P)
3     while q == m - 1 or (P[q + 1] != c and q > -1):
4         q = lps[q] - 1
5     if P[q + 1] == c:  q += 1
6     return q
```



Der Knuth-Morris-Pratt-Algorithmus

Mustersuche mit dem KMP:

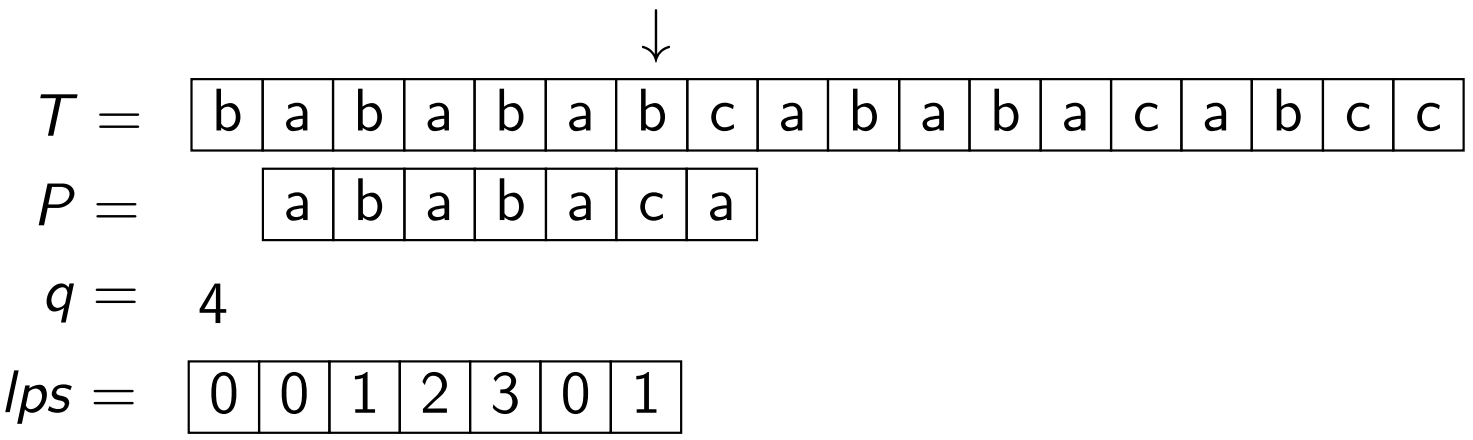
```
1 def delta(q, c, P, lps):
2     m = len(P)
3     while q == m - 1 or (P[q + 1] != c and q > -1):
4         q = lps[q] - 1
5     if P[q + 1] == c:  q += 1
6     return q
```



Der Knuth-Morris-Pratt-Algorithmus

Mustersuche mit dem KMP:

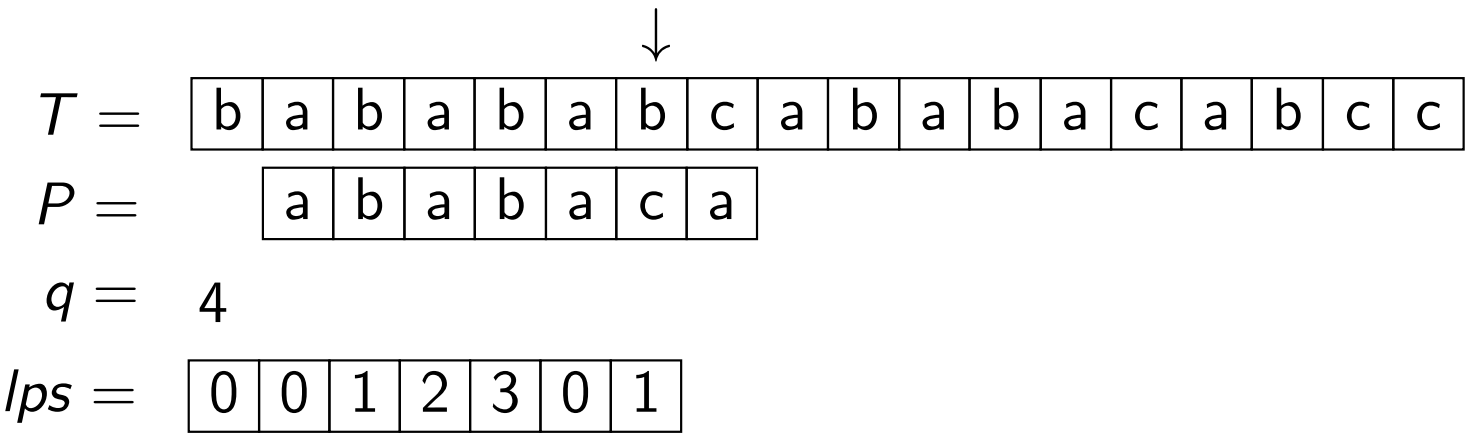
```
1 def delta(q, c, P, lps):
2     m = len(P)
3     while q == m - 1 or (P[q + 1] != c and q > -1):
4         q = lps[q] - 1
5     if P[q + 1] == c:  q += 1
6     return q
```



Der Knuth-Morris-Pratt-Algorithmus

Mustersuche mit dem KMP:

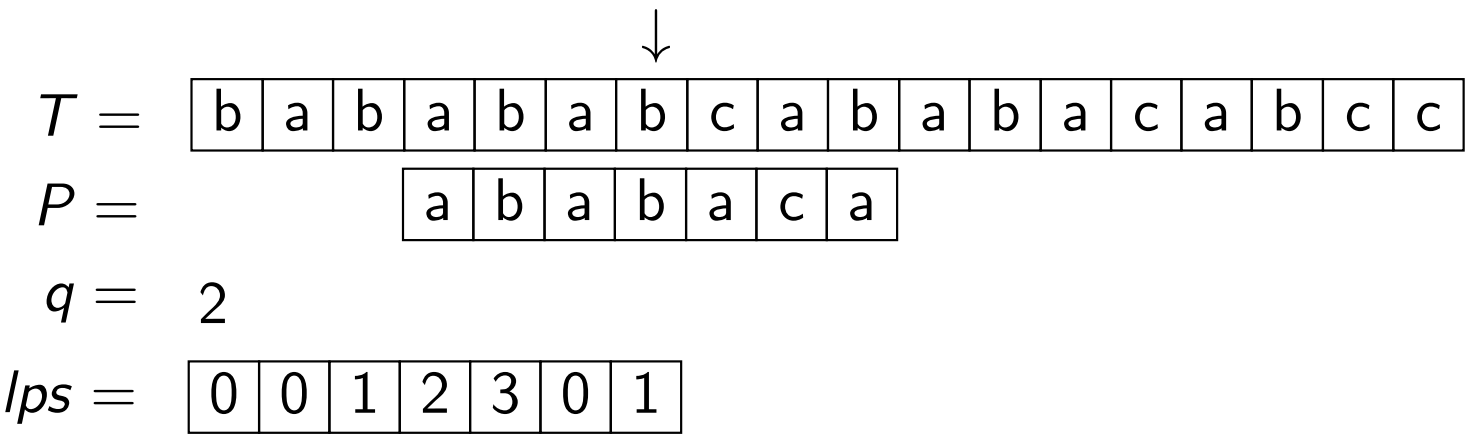
```
1 def delta(q, c, P, lps):
2     m = len(P)
3     while q == m - 1 or (P[q + 1] != c and q > -1):
4         q = lps[q] - 1
5     if P[q + 1] == c:  q += 1
6     return q
```



Der Knuth-Morris-Pratt-Algorithmus

Mustersuche mit dem KMP:

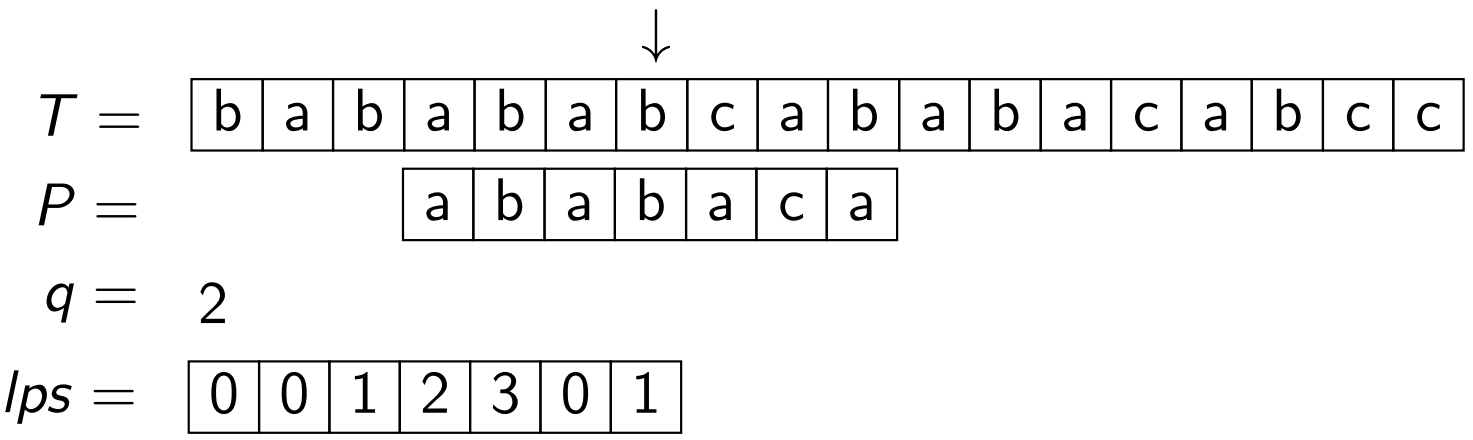
```
1 def delta(q, c, P, lps):
2     m = len(P)
3     while q == m - 1 or (P[q + 1] != c and q > -1):
4         q = lps[q] - 1
5     if P[q + 1] == c:  q += 1
6     return q
```



Der Knuth-Morris-Pratt-Algorithmus

Mustersuche mit dem KMP:

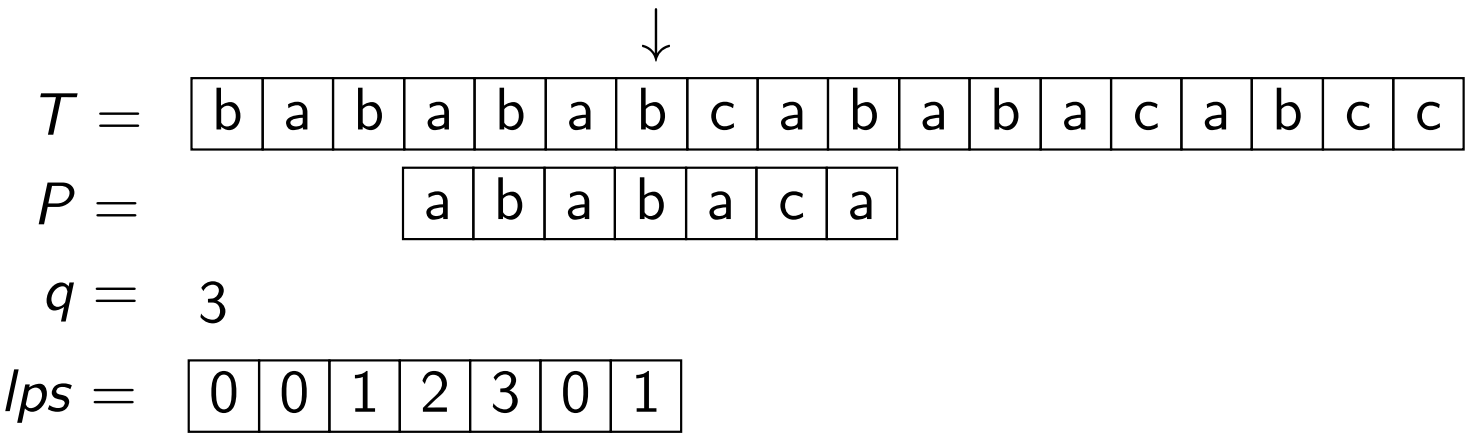
```
1 def delta(q, c, P, lps):
2     m = len(P)
3     while q == m - 1 or (P[q + 1] != c and q > -1):
4         q = lps[q] - 1
5     if P[q + 1] == c:  q += 1
6     return q
```



Der Knuth-Morris-Pratt-Algorithmus

Mustersuche mit dem KMP:

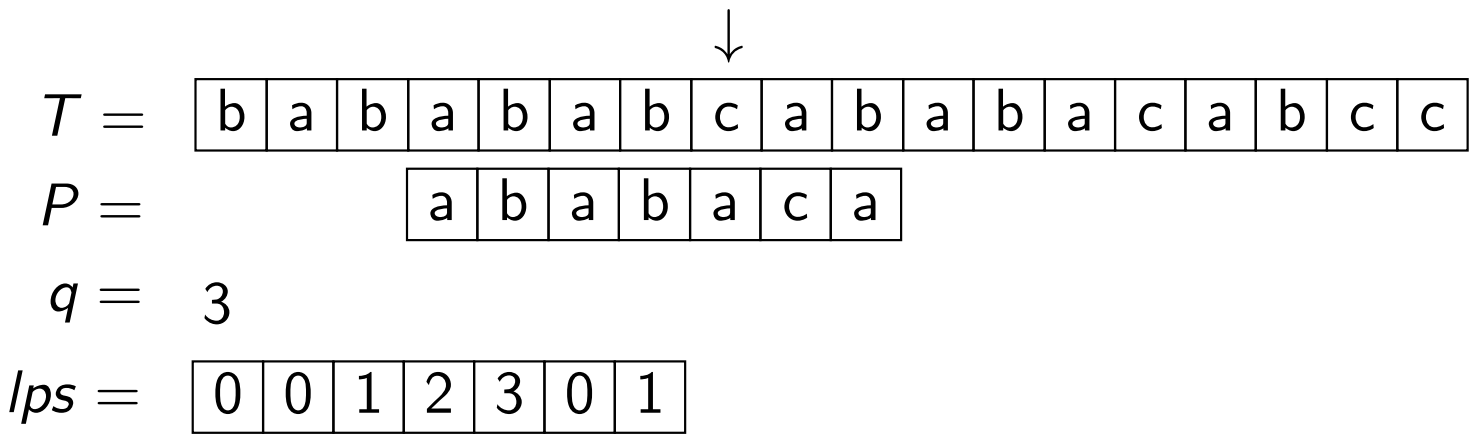
```
1 def delta(q, c, P, lps):
2     m = len(P)
3     while q == m - 1 or (P[q + 1] != c and q > -1):
4         q = lps[q] - 1
5     if P[q + 1] == c:  q += 1
6     return q
```



Der Knuth-Morris-Pratt-Algorithmus

Mustersuche mit dem KMP:

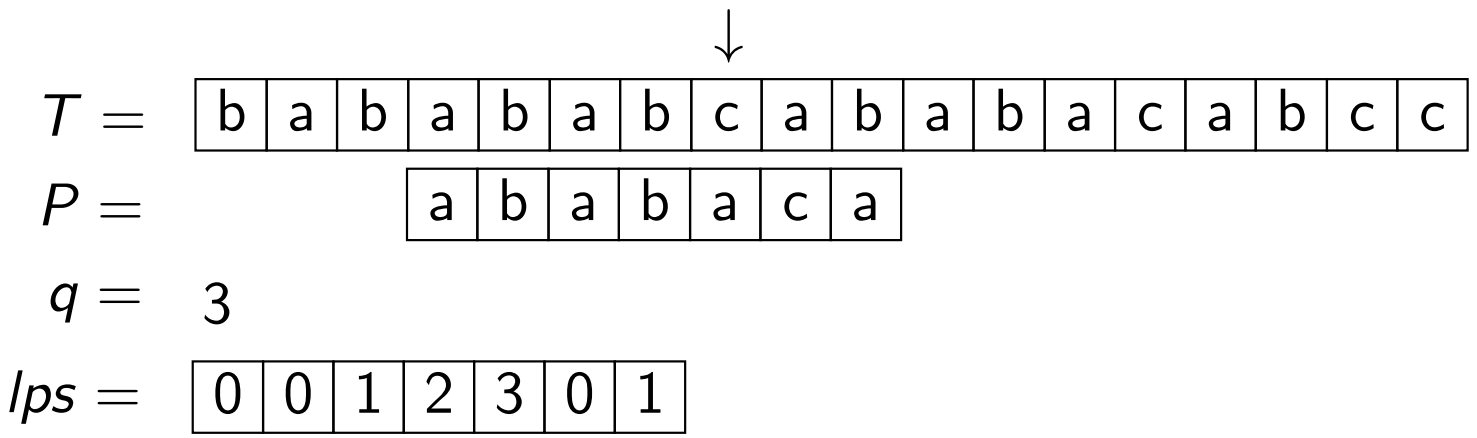
```
1 def delta(q, c, P, lps):
2     m = len(P)
3     while q == m - 1 or (P[q + 1] != c and q > -1):
4         q = lps[q] - 1
5     if P[q + 1] == c:  q += 1
6     return q
```



Der Knuth-Morris-Pratt-Algorithmus

Mustersuche mit dem KMP:

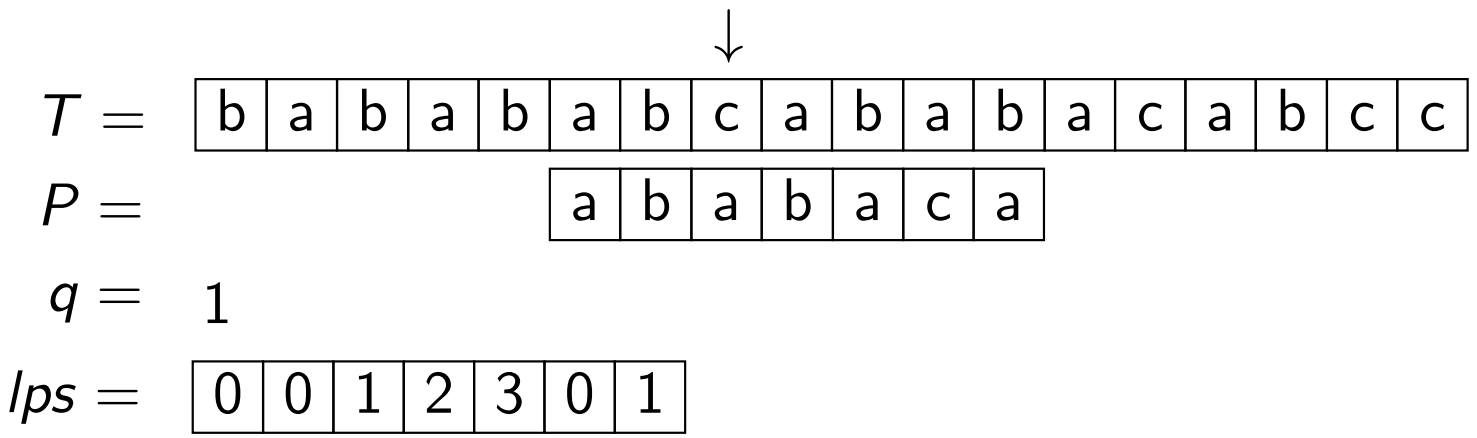
```
1 def delta(q, c, P, lps):
2     m = len(P)
3     while q == m - 1 or (P[q + 1] != c and q > -1):
4         q = lps[q] - 1
5     if P[q + 1] == c:  q += 1
6     return q
```



Der Knuth-Morris-Pratt-Algorithmus

Mustersuche mit dem KMP:

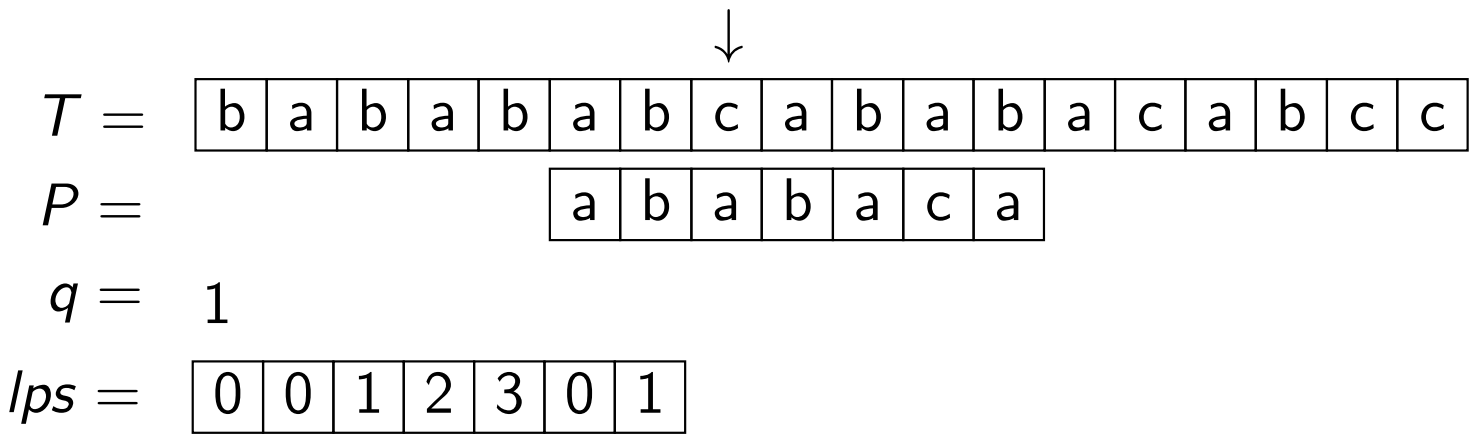
```
1 def delta(q, c, P, lps):
2     m = len(P)
3     while q == m - 1 or (P[q + 1] != c and q > -1):
4         q = lps[q] - 1
5     if P[q + 1] == c:  q += 1
6     return q
```



Der Knuth-Morris-Pratt-Algorithmus

Mustersuche mit dem KMP:

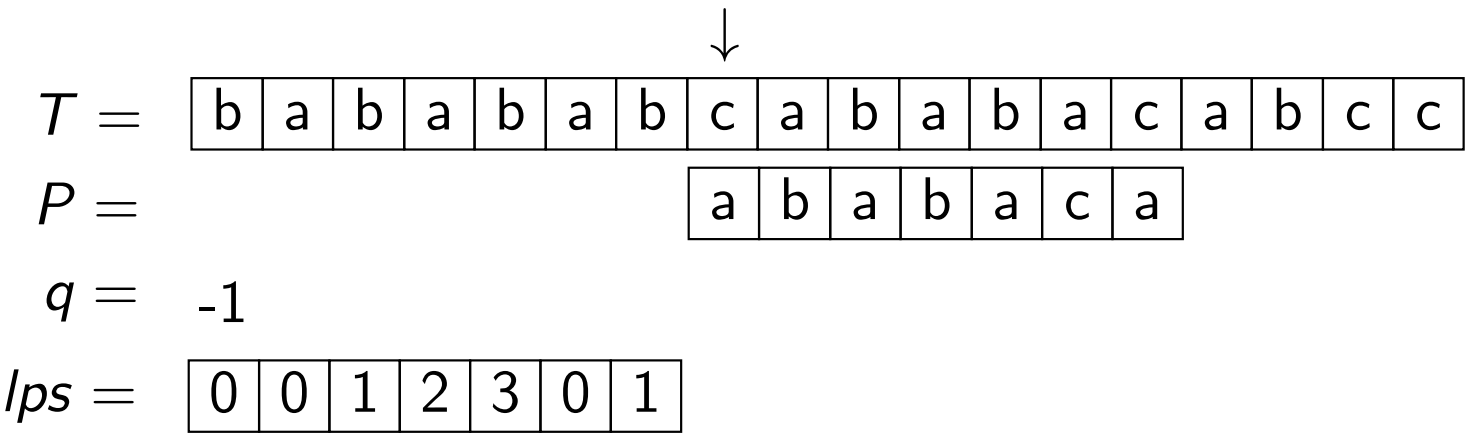
```
1 def delta(q, c, P, lps):
2     m = len(P)
3     while q == m - 1 or (P[q + 1] != c and q > -1):
4         q = lps[q] - 1
5     if P[q + 1] == c:  q += 1
6     return q
```



Der Knuth-Morris-Pratt-Algorithmus

Mustersuche mit dem KMP:

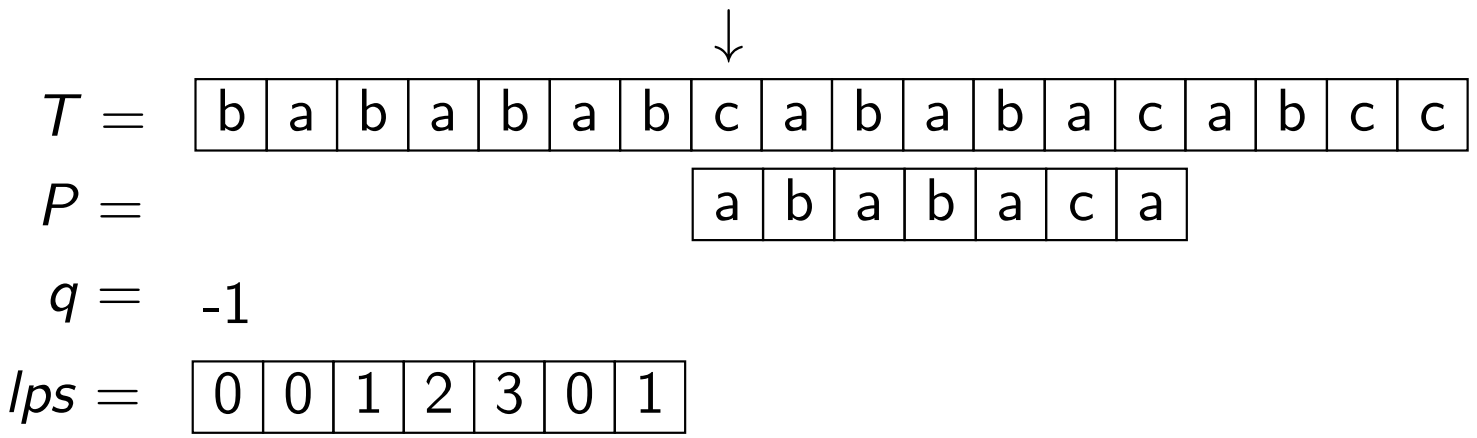
```
1 def delta(q, c, P, lps):
2     m = len(P)
3     while q == m - 1 or (P[q + 1] != c and q > -1):
4         q = lps[q] - 1
5     if P[q + 1] == c:  q += 1
6     return q
```



Der Knuth-Morris-Pratt-Algorithmus

Mustersuche mit dem KMP:

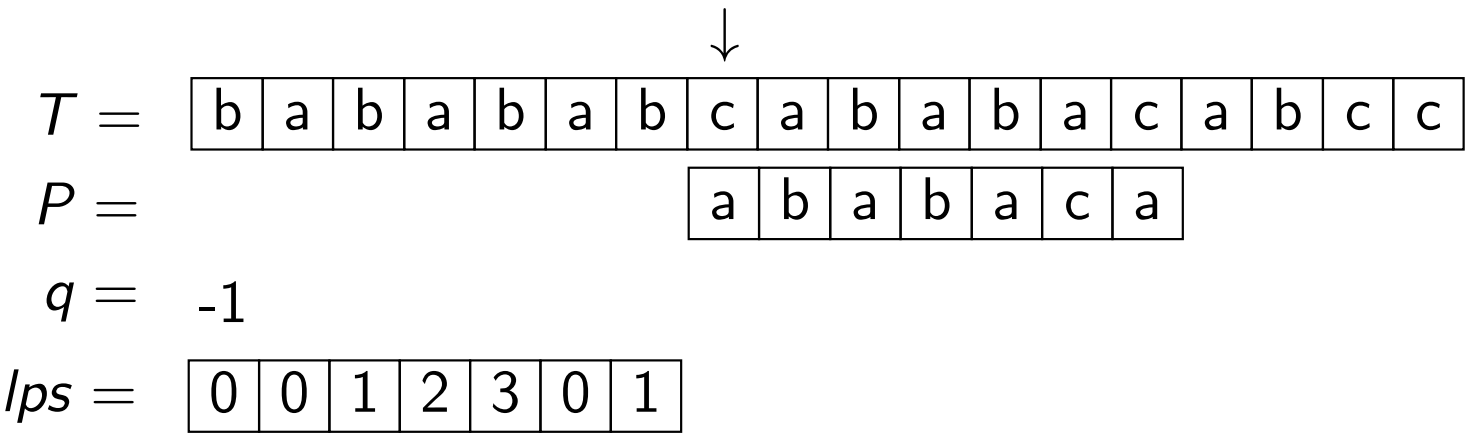
```
1 def delta(q, c, P, lps):
2     m = len(P)
3     while q == m - 1 or (P[q + 1] != c and q > -1):
4         q = lps[q] - 1
5     if P[q + 1] == c:  q += 1
6     return q
```



Der Knuth-Morris-Pratt-Algorithmus

Mustersuche mit dem KMP:

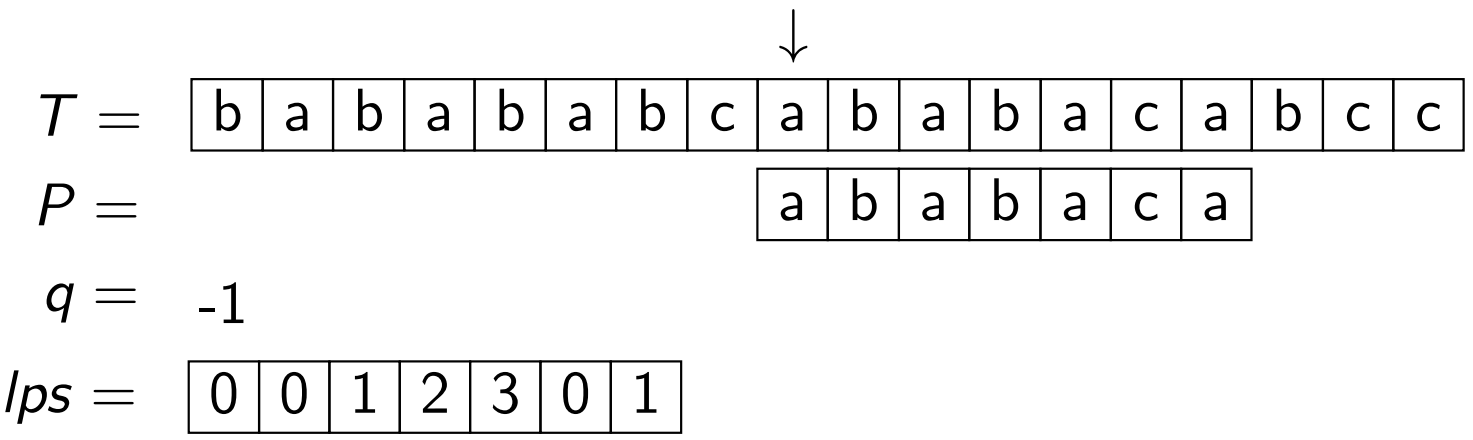
```
1 def delta(q, c, P, lps):
2     m = len(P)
3     while q == m - 1 or (P[q + 1] != c and q > -1):
4         q = lps[q] - 1
5     if P[q + 1] == c:  q += 1
6     return q
```



Der Knuth-Morris-Pratt-Algorithmus

Mustersuche mit dem KMP:

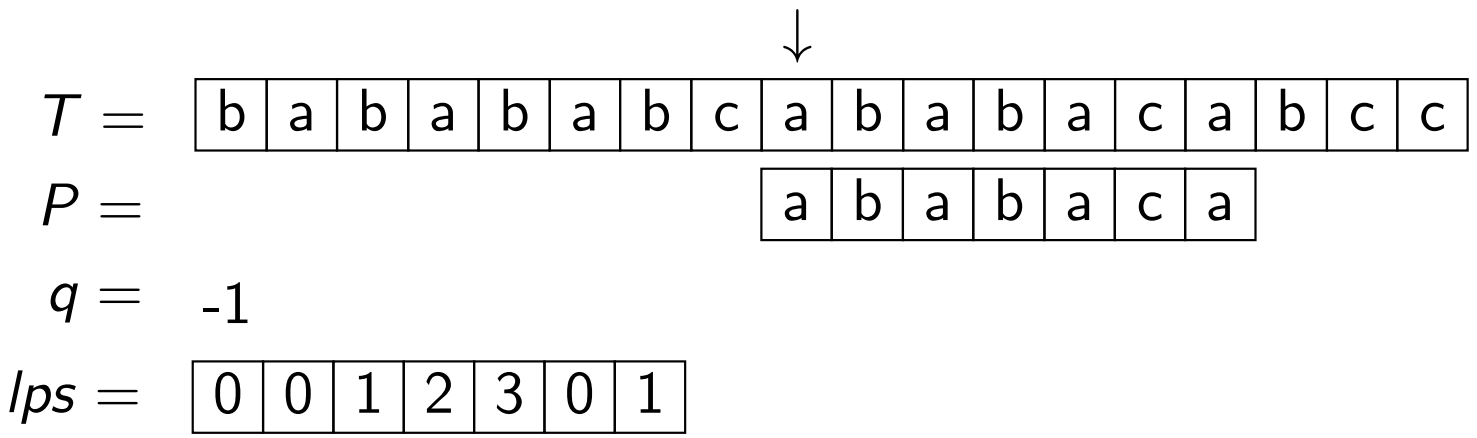
```
1 def delta(q, c, P, lps):
2     m = len(P)
3     while q == m - 1 or (P[q + 1] != c and q > -1):
4         q = lps[q] - 1
5     if P[q + 1] == c:  q += 1
6     return q
```



Der Knuth-Morris-Pratt-Algorithmus

Mustersuche mit dem KMP:

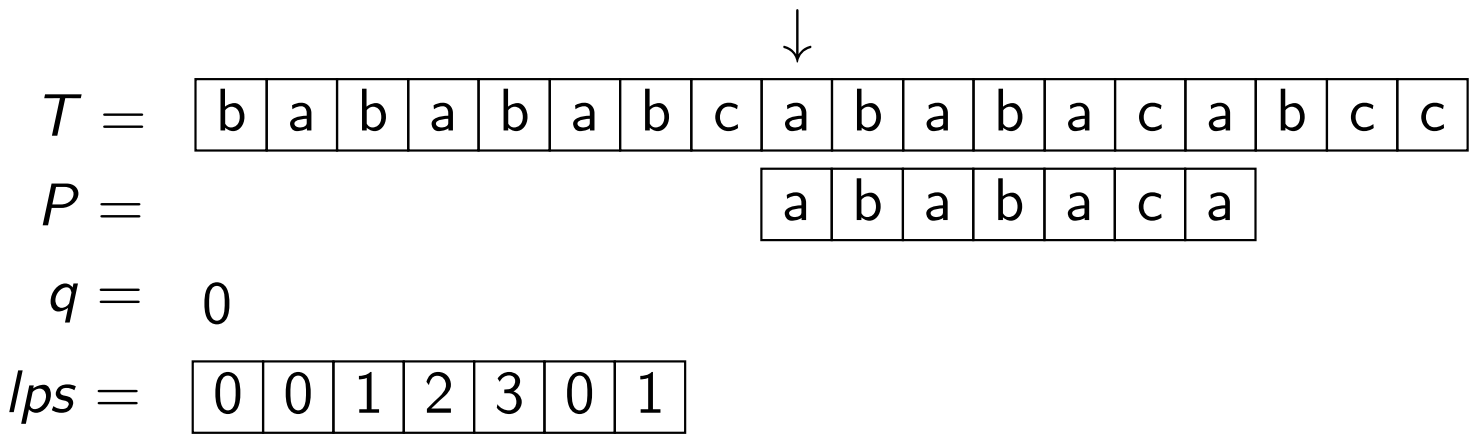
```
1 def delta(q, c, P, lps):
2     m = len(P)
3     while q == m - 1 or (P[q + 1] != c and q > -1):
4         q = lps[q] - 1
5     if P[q + 1] == c:  q += 1
6     return q
```



Der Knuth-Morris-Pratt-Algorithmus

Mustersuche mit dem KMP:

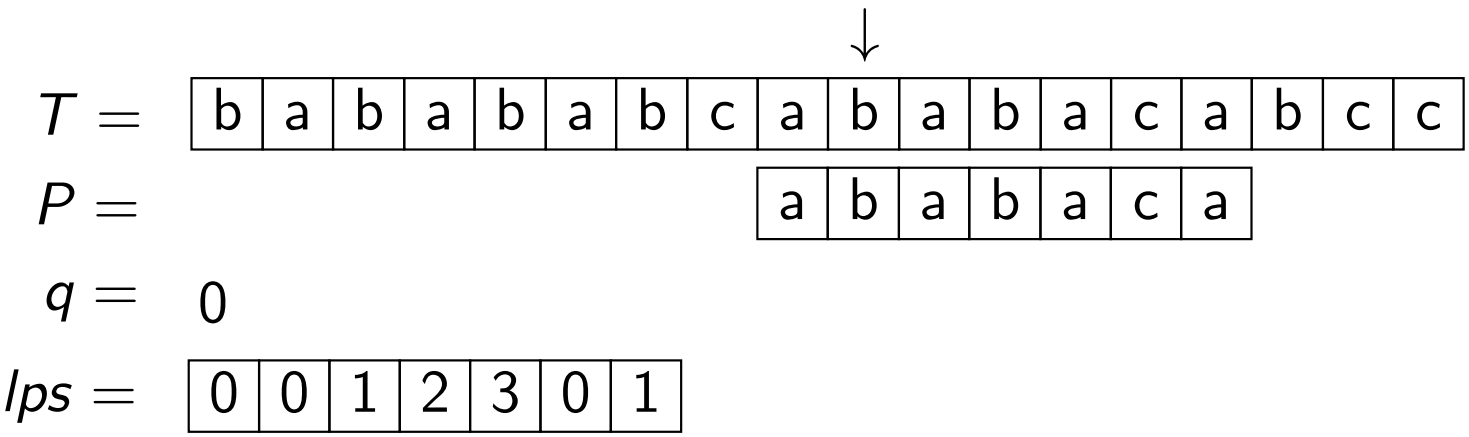
```
1 def delta(q, c, P, lps):
2     m = len(P)
3     while q == m - 1 or (P[q + 1] != c and q > -1):
4         q = lps[q] - 1
5     if P[q + 1] == c:  q += 1
6     return q
```



Der Knuth-Morris-Pratt-Algorithmus

Mustersuche mit dem KMP:

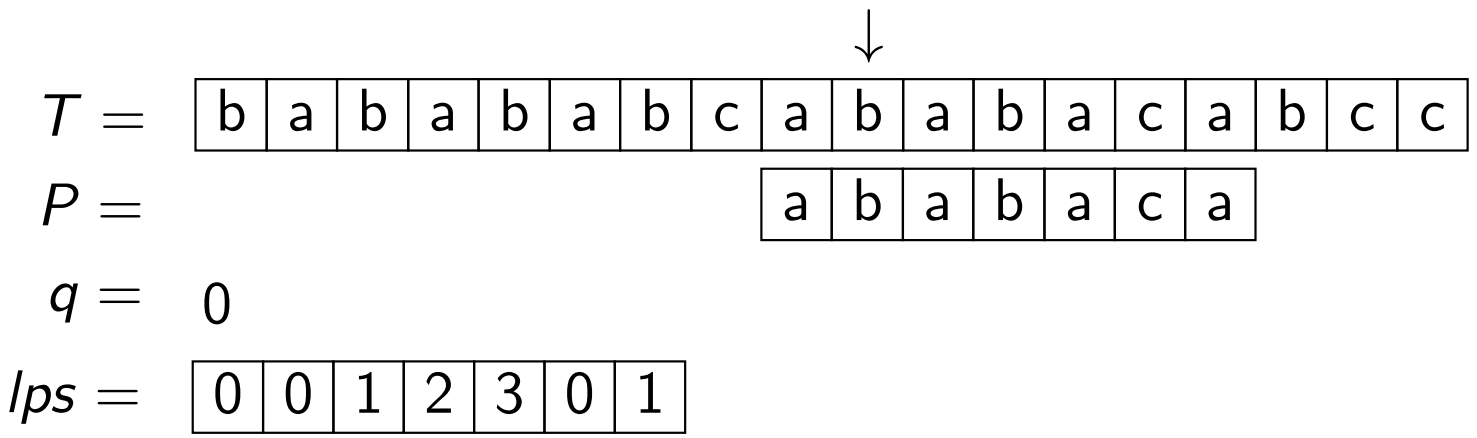
```
1 def delta(q, c, P, lps):
2     m = len(P)
3     while q == m - 1 or (P[q + 1] != c and q > -1):
4         q = lps[q] - 1
5     if P[q + 1] == c:  q += 1
6     return q
```



Der Knuth-Morris-Pratt-Algorithmus

Mustersuche mit dem KMP:

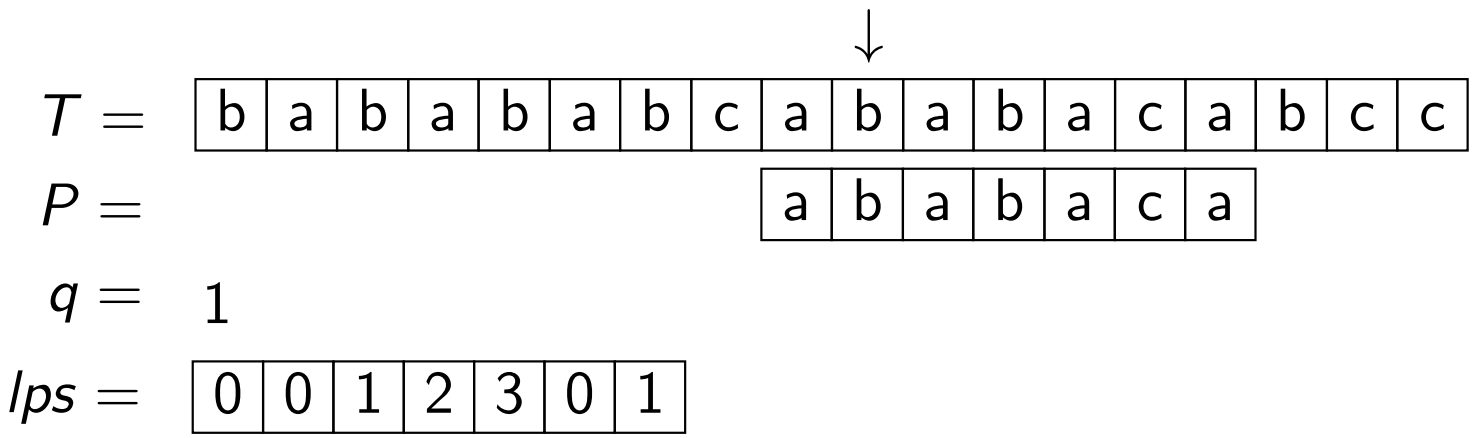
```
1 def delta(q, c, P, lps):
2     m = len(P)
3     while q == m - 1 or (P[q + 1] != c and q > -1):
4         q = lps[q] - 1
5     if P[q + 1] == c:  q += 1
6     return q
```



Der Knuth-Morris-Pratt-Algorithmus

Mustersuche mit dem KMP:

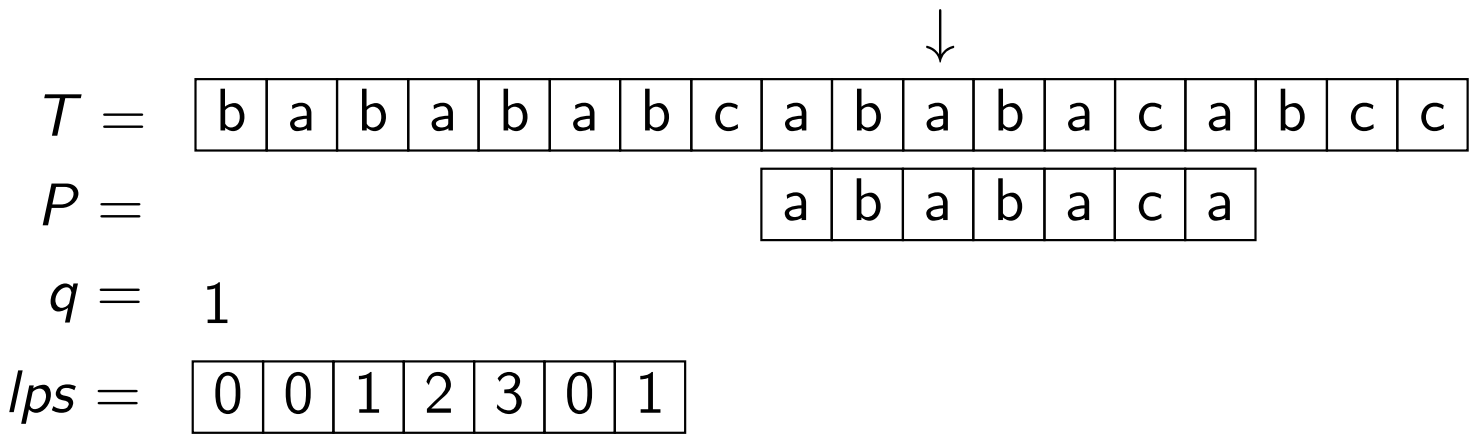
```
1 def delta(q, c, P, lps):
2     m = len(P)
3     while q == m - 1 or (P[q + 1] != c and q > -1):
4         q = lps[q] - 1
5     if P[q + 1] == c:  q += 1
6     return q
```



Der Knuth-Morris-Pratt-Algorithmus

Mustersuche mit dem KMP:

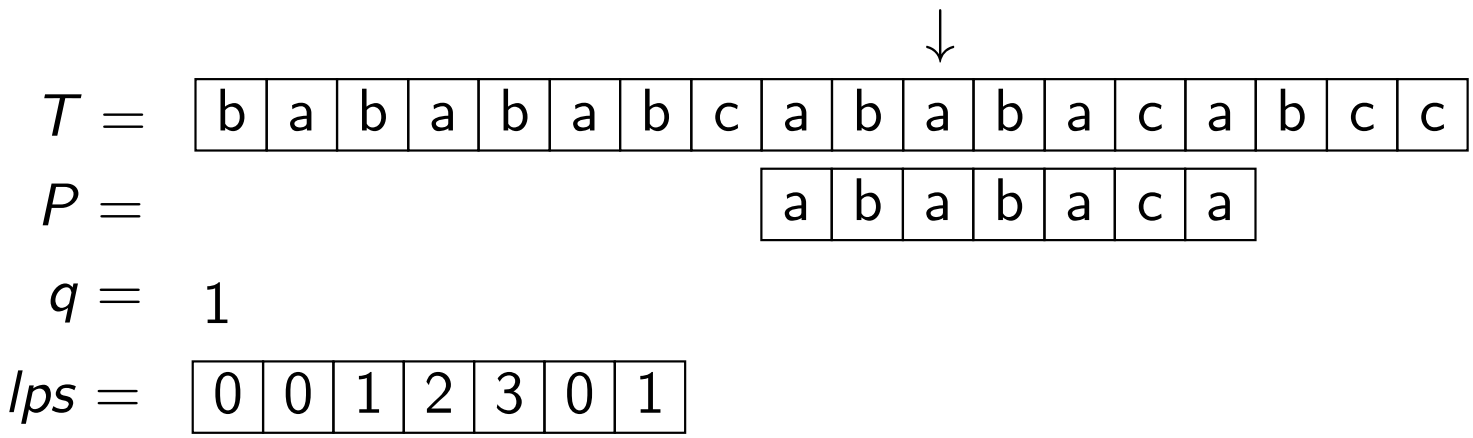
```
1 def delta(q, c, P, lps):
2     m = len(P)
3     while q == m - 1 or (P[q + 1] != c and q > -1):
4         q = lps[q] - 1
5     if P[q + 1] == c:  q += 1
6     return q
```



Der Knuth-Morris-Pratt-Algorithmus

Mustersuche mit dem KMP:

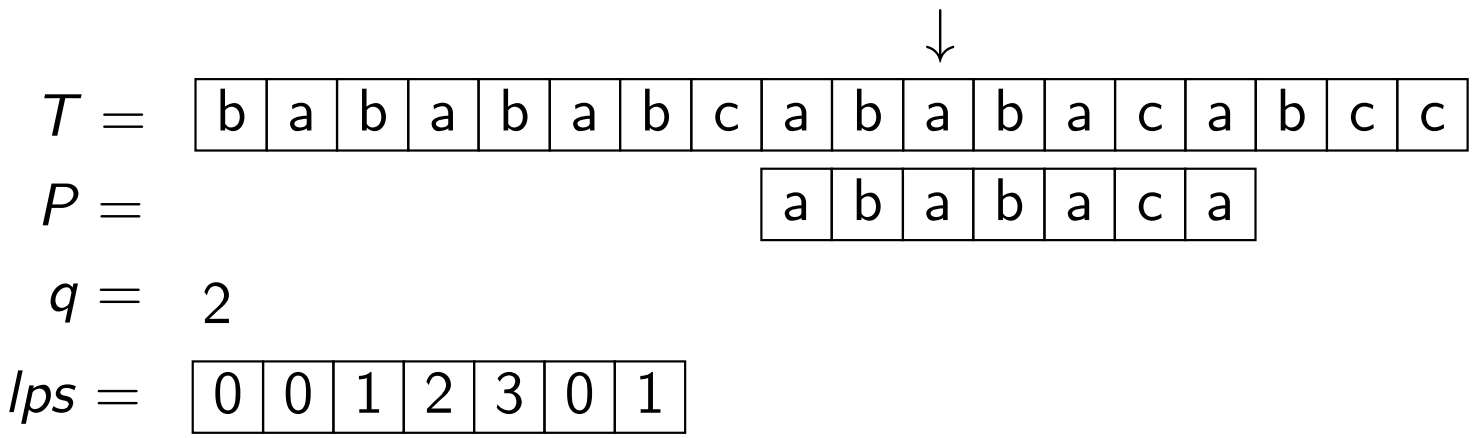
```
1 def delta(q, c, P, lps):
2     m = len(P)
3     while q == m - 1 or (P[q + 1] != c and q > -1):
4         q = lps[q] - 1
5     if P[q + 1] == c:  q += 1
6     return q
```



Der Knuth-Morris-Pratt-Algorithmus

Mustersuche mit dem KMP:

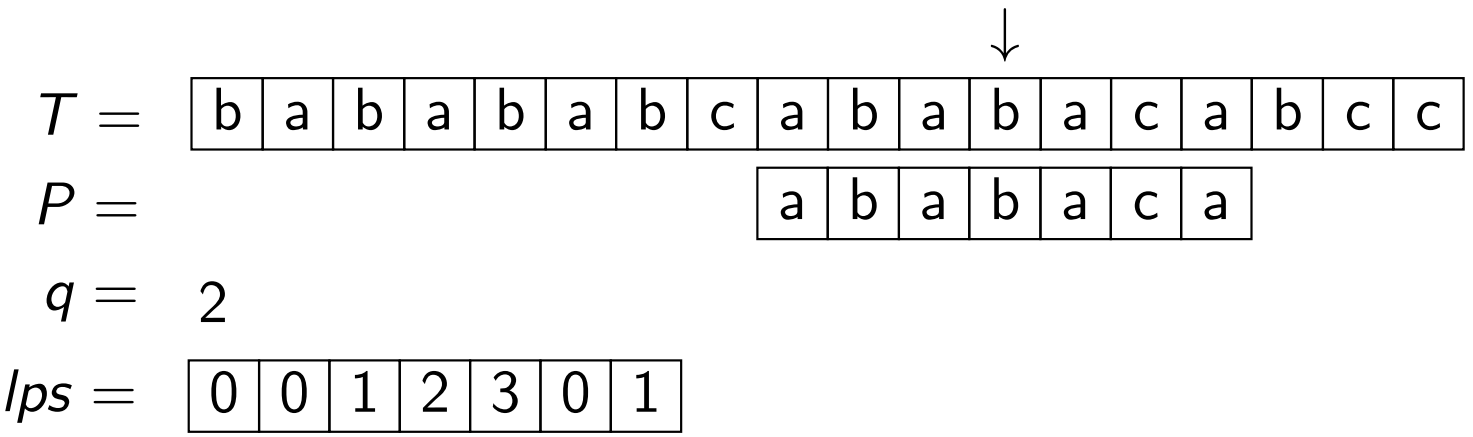
```
1 def delta(q, c, P, lps):
2     m = len(P)
3     while q == m - 1 or (P[q + 1] != c and q > -1):
4         q = lps[q] - 1
5     if P[q + 1] == c:  q += 1
6     return q
```



Der Knuth-Morris-Pratt-Algorithmus

Mustersuche mit dem KMP:

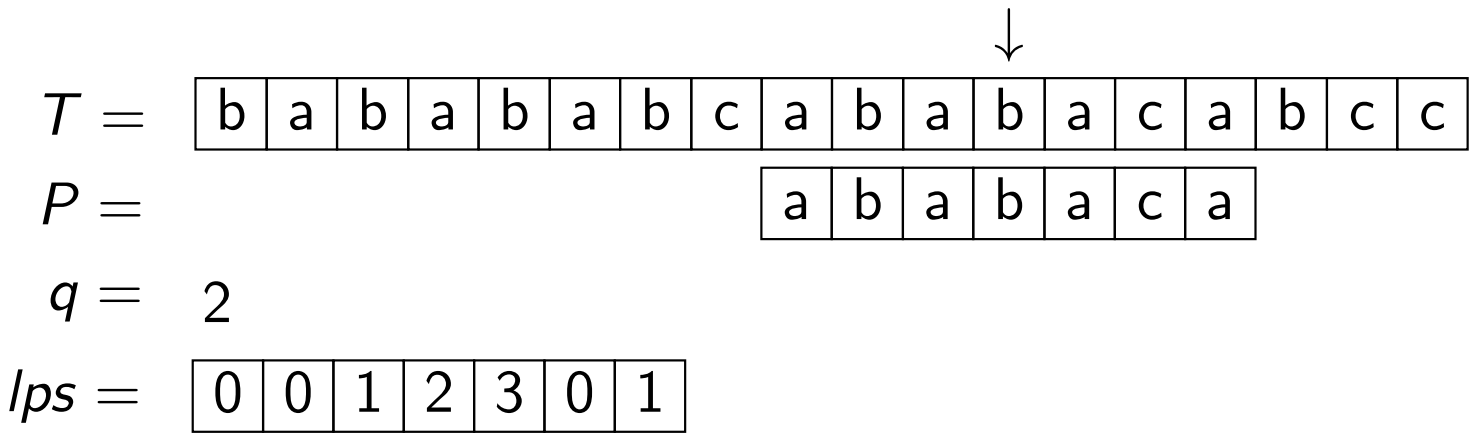
```
1 def delta(q, c, P, lps):
2     m = len(P)
3     while q == m - 1 or (P[q + 1] != c and q > -1):
4         q = lps[q] - 1
5     if P[q + 1] == c:  q += 1
6     return q
```



Der Knuth-Morris-Pratt-Algorithmus

Mustersuche mit dem KMP:

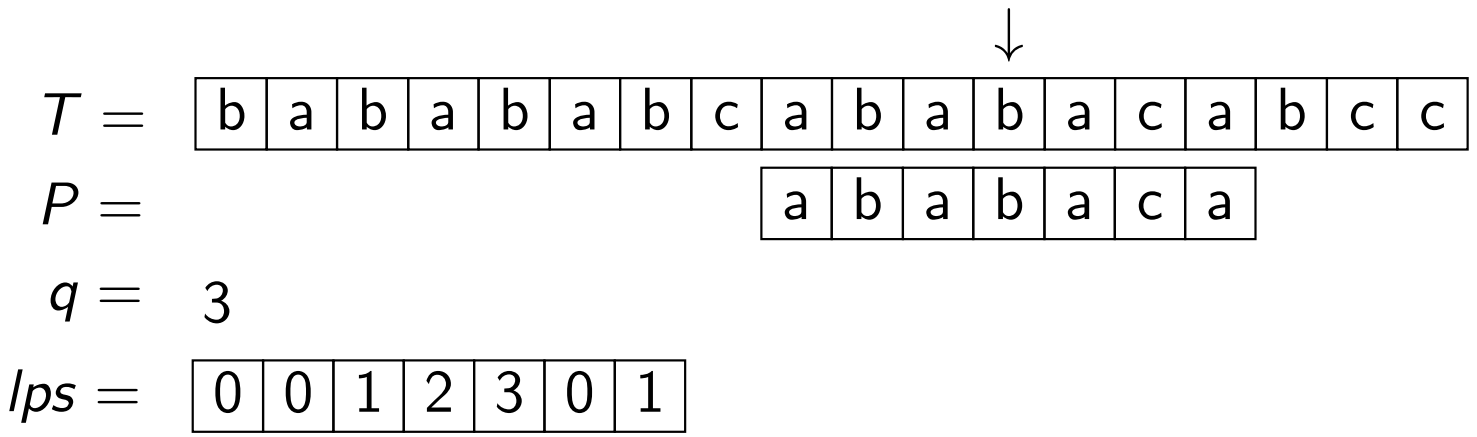
```
1 def delta(q, c, P, lps):
2     m = len(P)
3     while q == m - 1 or (P[q + 1] != c and q > -1):
4         q = lps[q] - 1
5     if P[q + 1] == c:  q += 1
6     return q
```



Der Knuth-Morris-Pratt-Algorithmus

Mustersuche mit dem KMP:

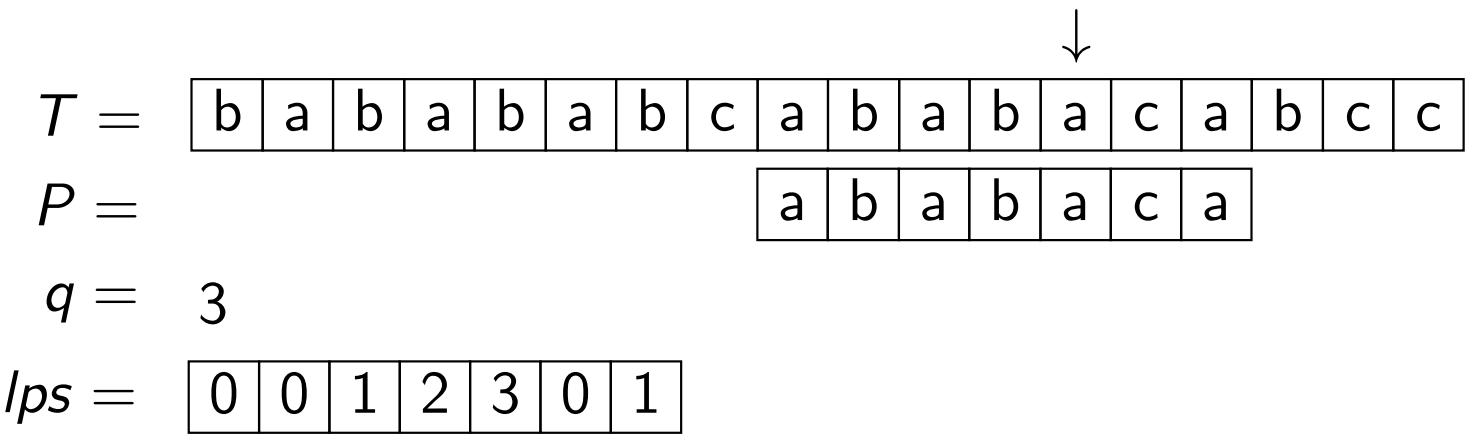
```
1 def delta(q, c, P, lps):
2     m = len(P)
3     while q == m - 1 or (P[q + 1] != c and q > -1):
4         q = lps[q] - 1
5     if P[q + 1] == c:  q += 1
6     return q
```



Der Knuth-Morris-Pratt-Algorithmus

Mustersuche mit dem KMP:

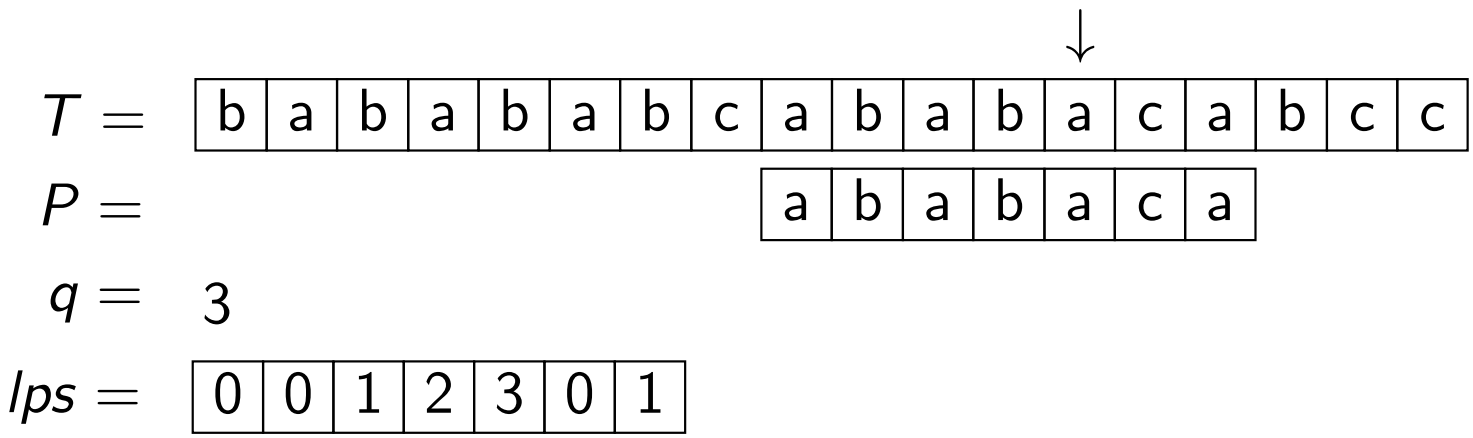
```
1 def delta(q, c, P, lps):
2     m = len(P)
3     while q == m - 1 or (P[q + 1] != c and q > -1):
4         q = lps[q] - 1
5     if P[q + 1] == c:  q += 1
6     return q
```



Der Knuth-Morris-Pratt-Algorithmus

Mustersuche mit dem KMP:

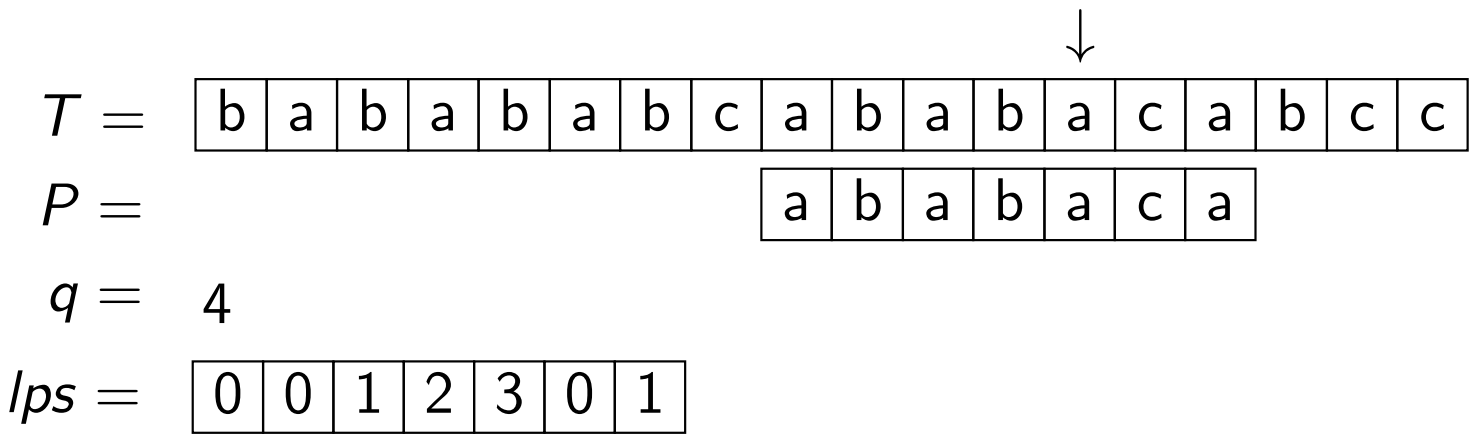
```
1 def delta(q, c, P, lps):
2     m = len(P)
3     while q == m - 1 or (P[q + 1] != c and q > -1):
4         q = lps[q] - 1
5     if P[q + 1] == c:  q += 1
6     return q
```



Der Knuth-Morris-Pratt-Algorithmus

Mustersuche mit dem KMP:

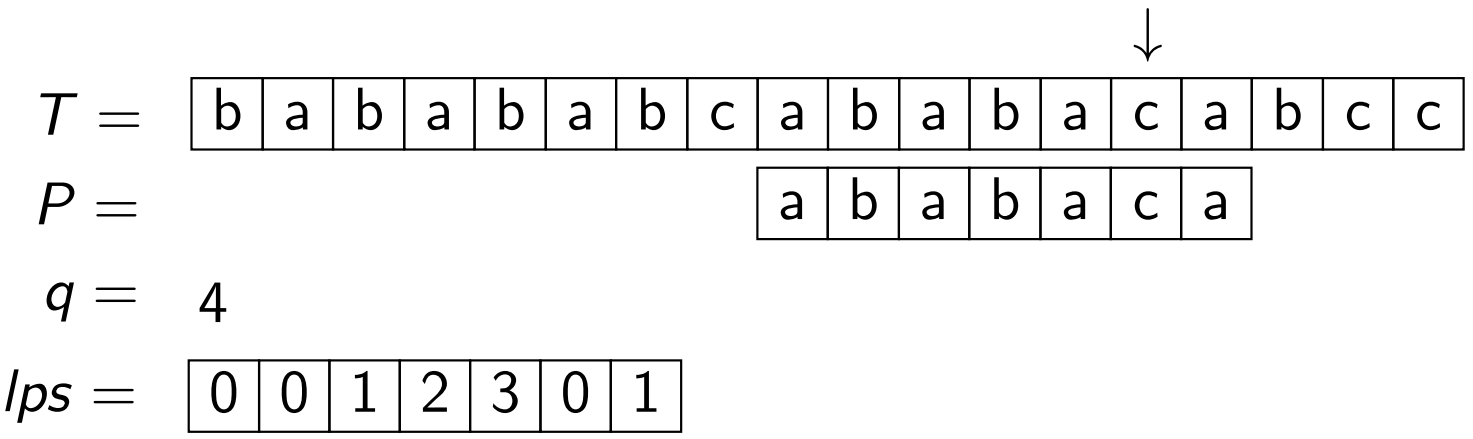
```
1 def delta(q, c, P, lps):
2     m = len(P)
3     while q == m - 1 or (P[q + 1] != c and q > -1):
4         q = lps[q] - 1
5     if P[q + 1] == c:  q += 1
6     return q
```



Der Knuth-Morris-Pratt-Algorithmus

Mustersuche mit dem KMP:

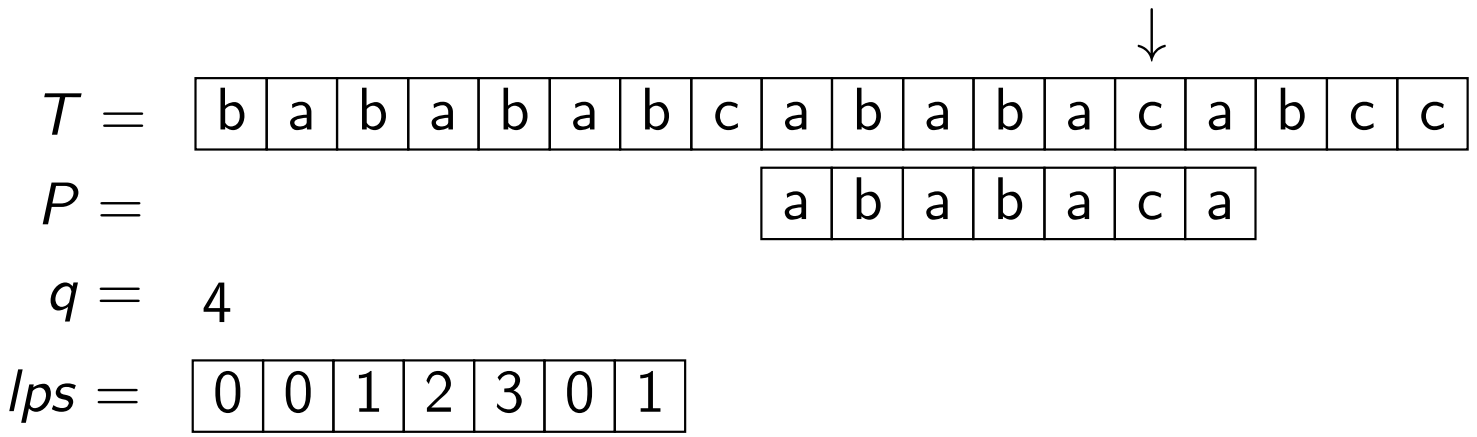
```
1 def delta(q, c, P, lps):
2     m = len(P)
3     while q == m - 1 or (P[q + 1] != c and q > -1):
4         q = lps[q] - 1
5     if P[q + 1] == c:  q += 1
6     return q
```



Der Knuth-Morris-Pratt-Algorithmus

Mustersuche mit dem KMP:

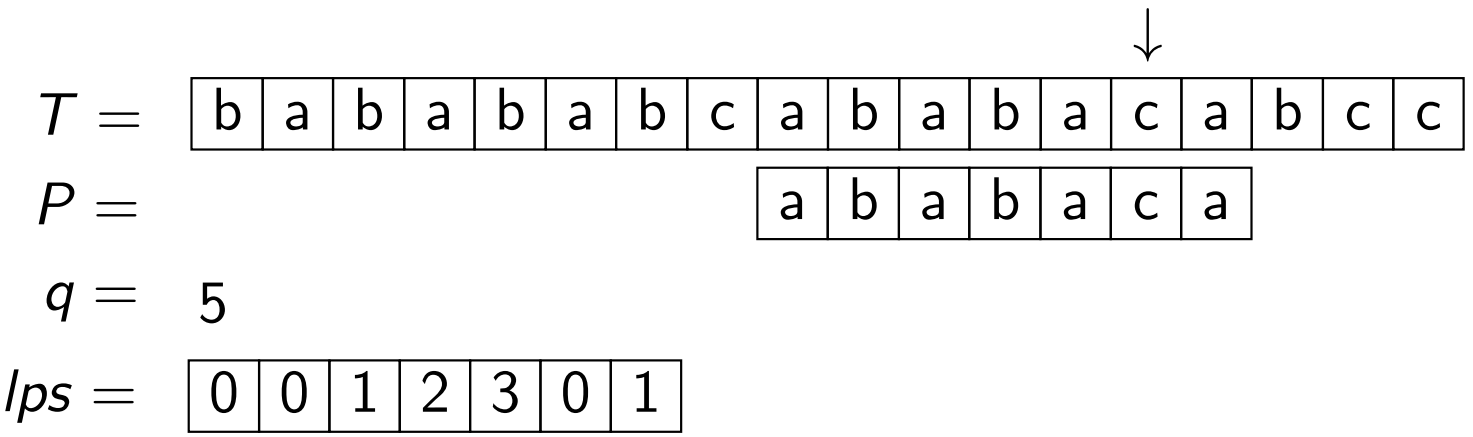
```
1 def delta(q, c, P, lps):
2     m = len(P)
3     while q == m - 1 or (P[q + 1] != c and q > -1):
4         q = lps[q] - 1
5     if P[q + 1] == c:  q += 1
6     return q
```



Der Knuth-Morris-Pratt-Algorithmus

Mustersuche mit dem KMP:

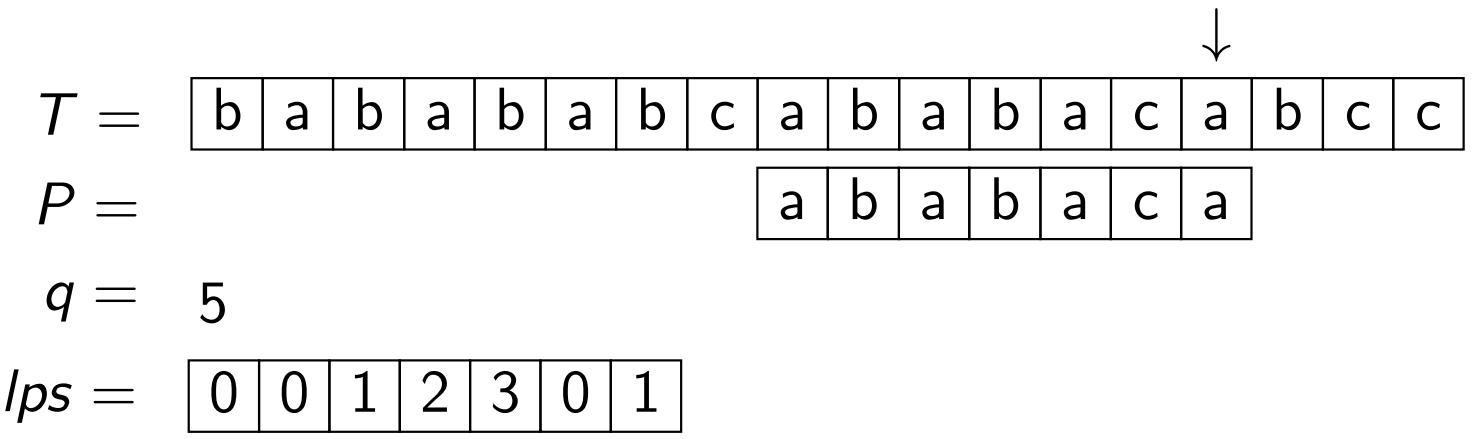
```
1 def delta(q, c, P, lps):
2     m = len(P)
3     while q == m - 1 or (P[q + 1] != c and q > -1):
4         q = lps[q] - 1
5     if P[q + 1] == c:  q += 1
6     return q
```



Der Knuth-Morris-Pratt-Algorithmus

Mustersuche mit dem KMP:

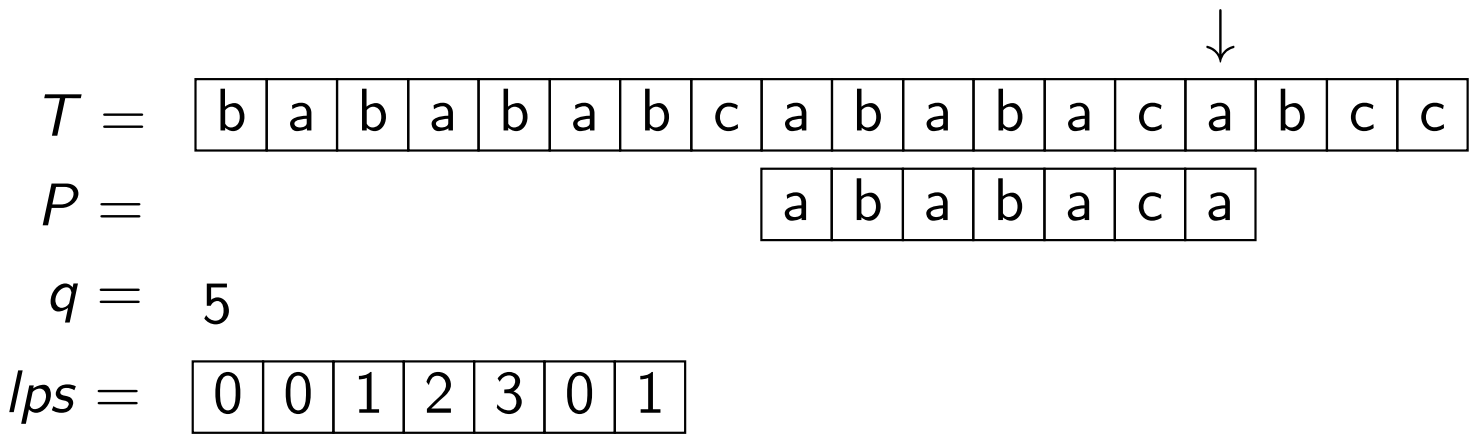
```
1 def delta(q, c, P, lps):
2     m = len(P)
3     while q == m - 1 or (P[q + 1] != c and q > -1):
4         q = lps[q] - 1
5     if P[q + 1] == c:  q += 1
6     return q
```



Der Knuth-Morris-Pratt-Algorithmus

Mustersuche mit dem KMP:

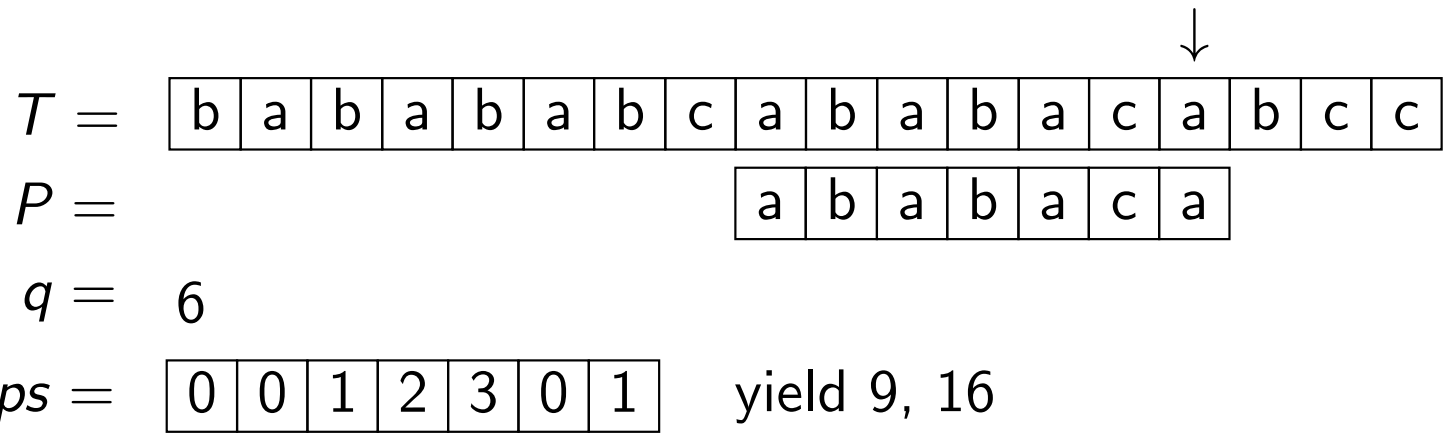
```
1 def delta(q, c, P, lps):
2     m = len(P)
3     while q == m - 1 or (P[q + 1] != c and q > -1):
4         q = lps[q] - 1
5     if P[q + 1] == c:  q += 1
6     return q
```



Der Knuth-Morris-Pratt-Algorithmus

Mustersuche mit dem KMP:

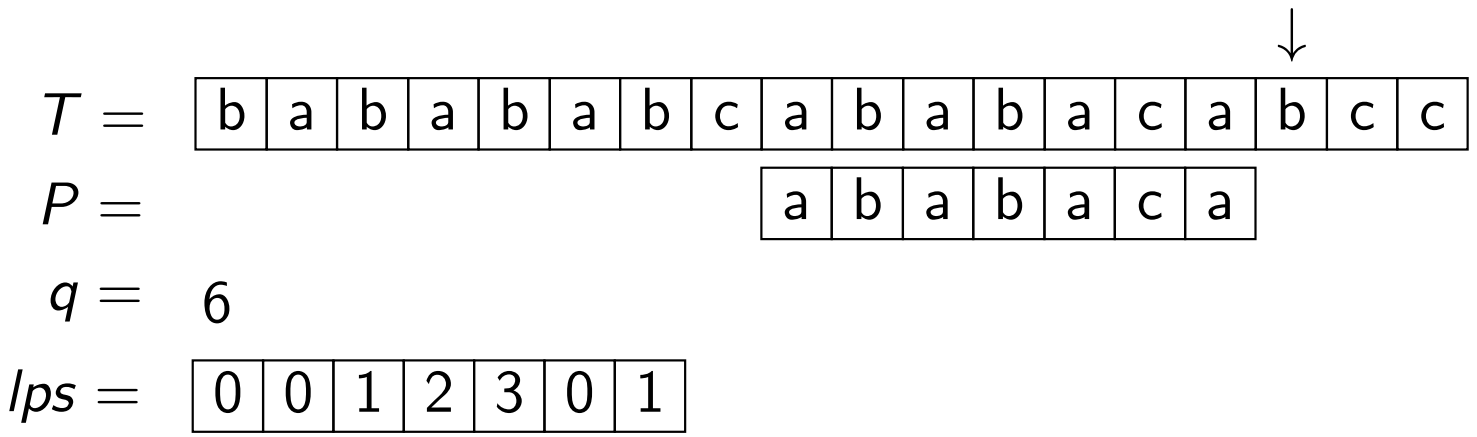
```
1 def delta(q, c, P, lps):
2     m = len(P)
3     while q == m - 1 or (P[q + 1] != c and q > -1):
4         q = lps[q] - 1
5     if P[q + 1] == c:  q += 1
6     return q
```



Der Knuth-Morris-Pratt-Algorithmus

Mustersuche mit dem KMP:

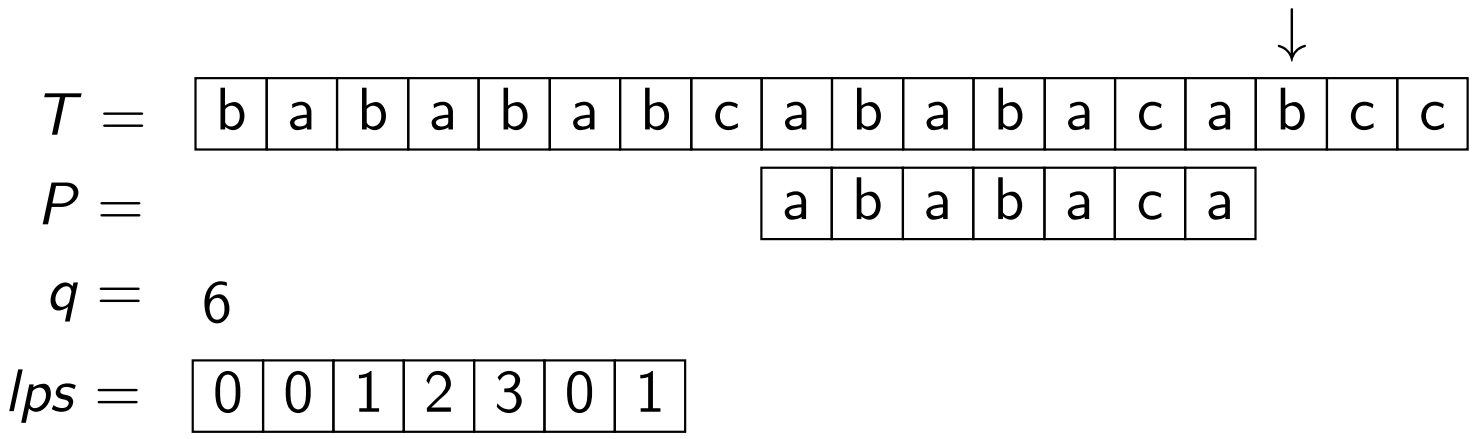
```
1 def delta(q, c, P, lps):
2     m = len(P)
3     while q == m - 1 or (P[q + 1] != c and q > -1):
4         q = lps[q] - 1
5     if P[q + 1] == c:  q += 1
6     return q
```



Der Knuth-Morris-Pratt-Algorithmus

Mustersuche mit dem KMP:

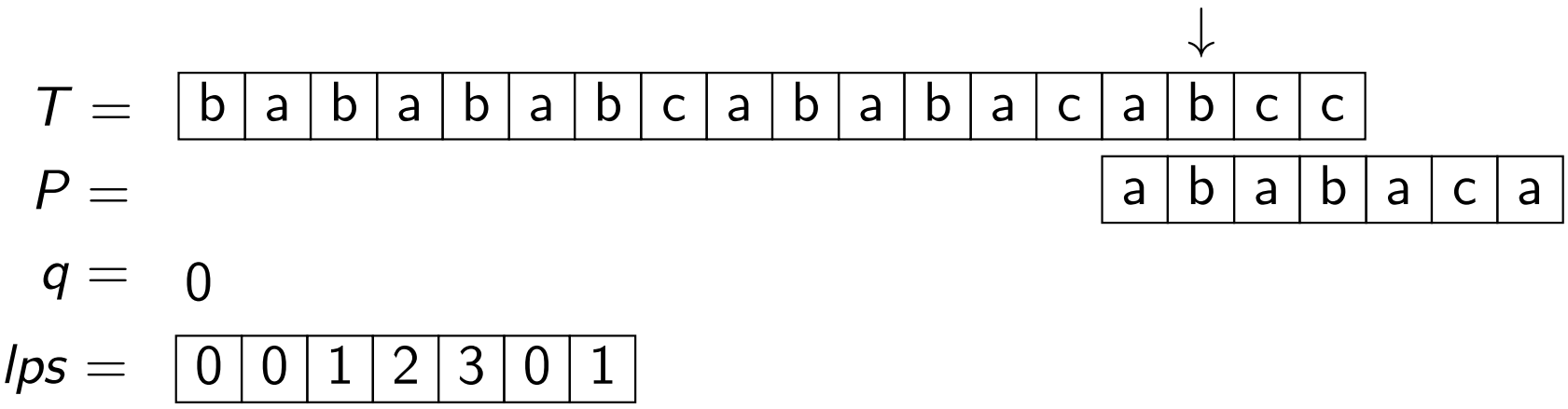
```
1 def delta(q, c, P, lps):
2     m = len(P)
3     while q == m - 1 or (P[q + 1] != c and q > -1):
4         q = lps[q] - 1
5     if P[q + 1] == c:  q += 1
6     return q
```



Der Knuth-Morris-Pratt-Algorithmus

Mustersuche mit dem KMP:

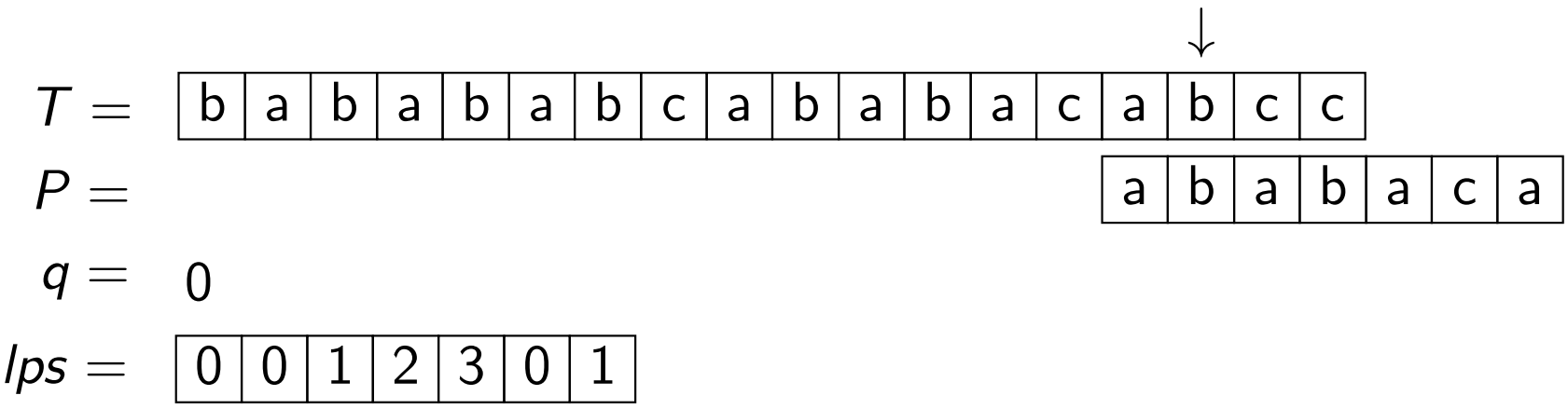
```
1 def delta(q, c, P, lps):
2     m = len(P)
3     while q == m - 1 or (P[q + 1] != c and q > -1):
4         q = lps[q] - 1
5     if P[q + 1] == c:  q += 1
6     return q
```



Der Knuth-Morris-Pratt-Algorithmus

Mustersuche mit dem KMP:

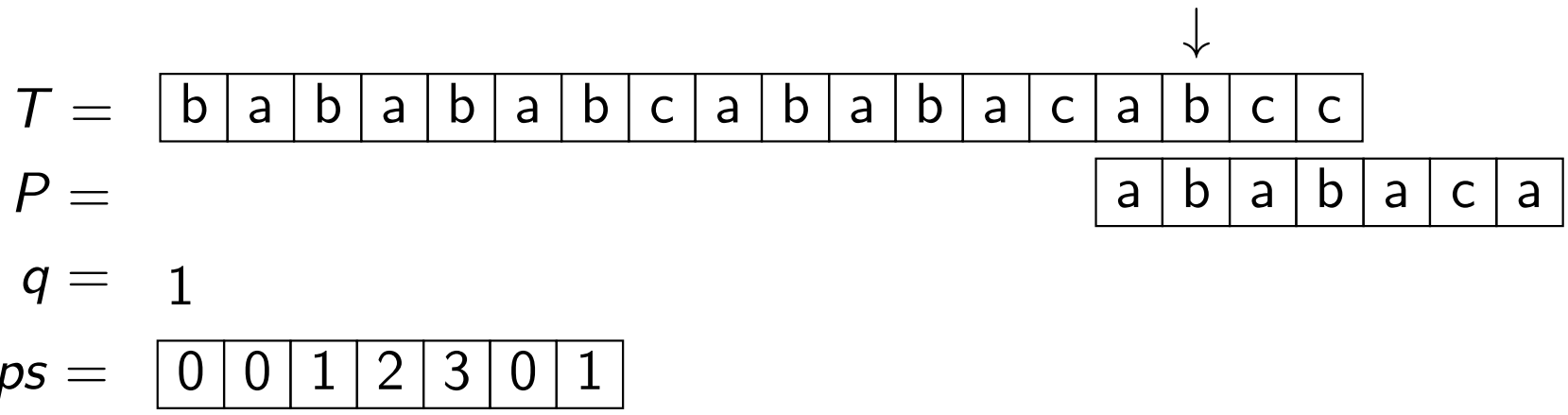
```
1 def delta(q, c, P, lps):
2     m = len(P)
3     while q == m - 1 or (P[q + 1] != c and q > -1):
4         q = lps[q] - 1
5     if P[q + 1] == c:  q += 1
6     return q
```



Der Knuth-Morris-Pratt-Algorithmus

Mustersuche mit dem KMP:

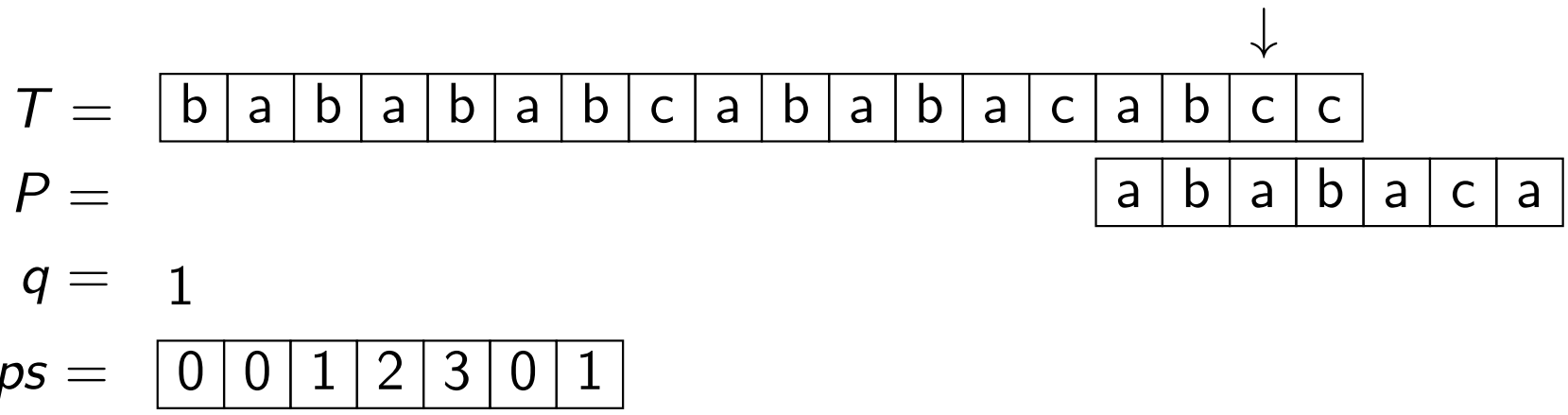
```
1 def delta(q, c, P, lps):
2     m = len(P)
3     while q == m - 1 or (P[q + 1] != c and q > -1):
4         q = lps[q] - 1
5     if P[q + 1] == c:  q += 1
6     return q
```



Der Knuth-Morris-Pratt-Algorithmus

Mustersuche mit dem KMP:

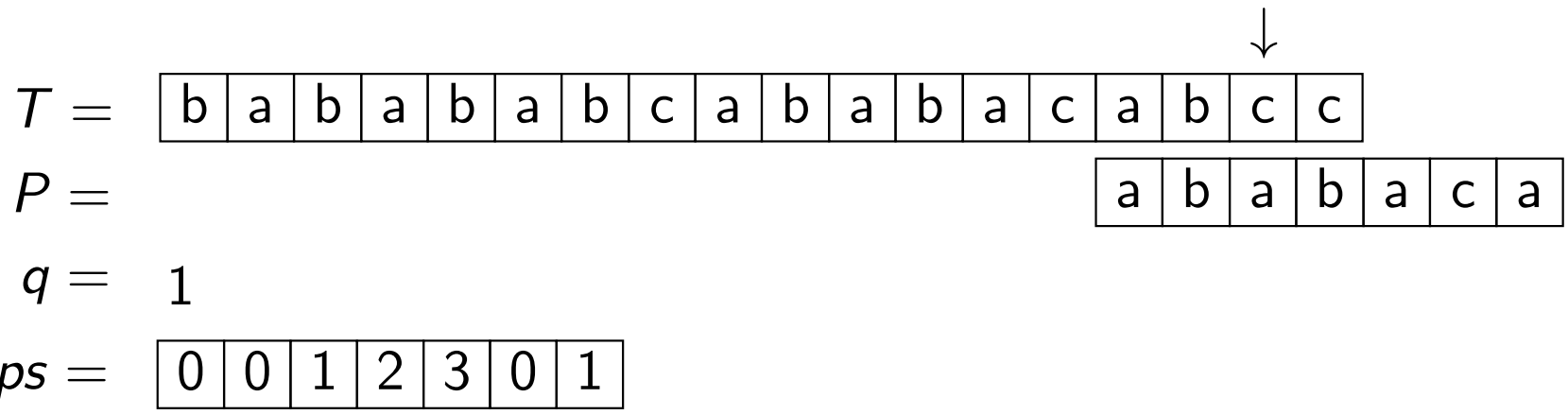
```
1 def delta(q, c, P, lps):
2     m = len(P)
3     while q == m - 1 or (P[q + 1] != c and q > -1):
4         q = lps[q] - 1
5     if P[q + 1] == c:  q += 1
6     return q
```



Der Knuth-Morris-Pratt-Algorithmus

Mustersuche mit dem KMP:

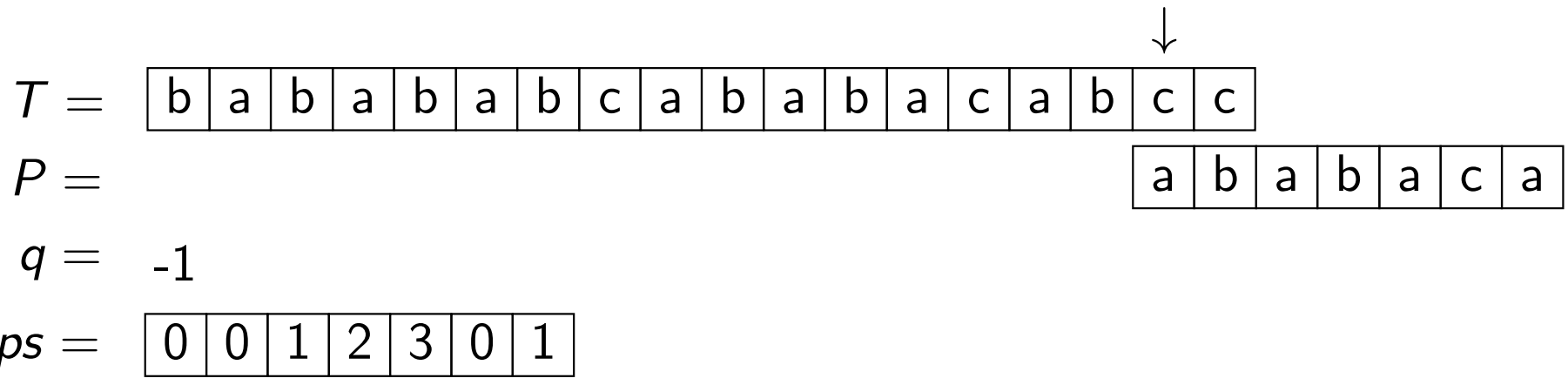
```
1 def delta(q, c, P, lps):
2     m = len(P)
3     while q == m - 1 or (P[q + 1] != c and q > -1):
4         q = lps[q] - 1
5     if P[q + 1] == c:  q += 1
6     return q
```



Der Knuth-Morris-Pratt-Algorithmus

Mustersuche mit dem KMP:

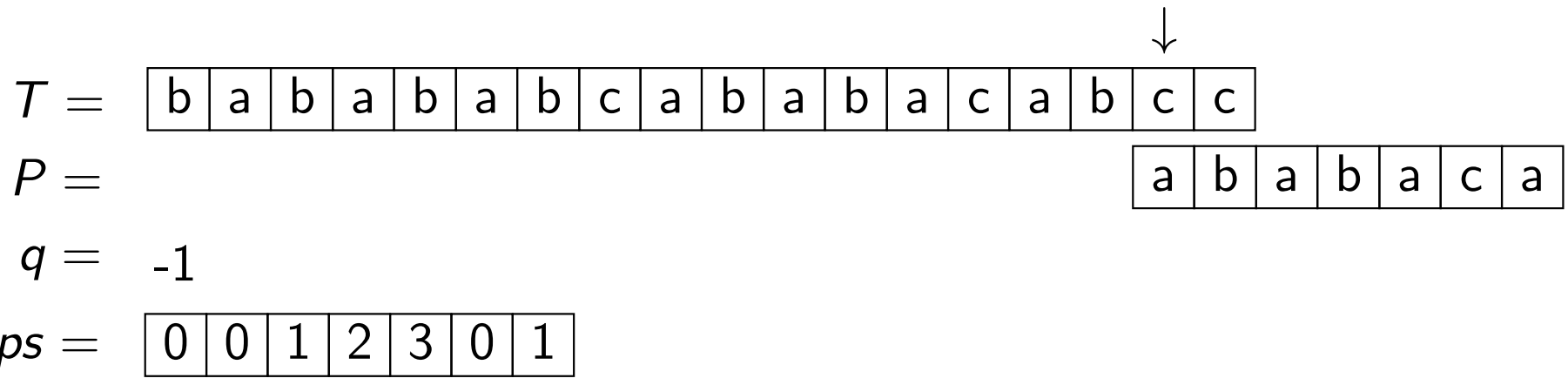
```
1 def delta(q, c, P, lps):
2     m = len(P)
3     while q == m - 1 or (P[q + 1] != c and q > -1):
4         q = lps[q] - 1
5     if P[q + 1] == c:  q += 1
6     return q
```



Der Knuth-Morris-Pratt-Algorithmus

Mustersuche mit dem KMP:

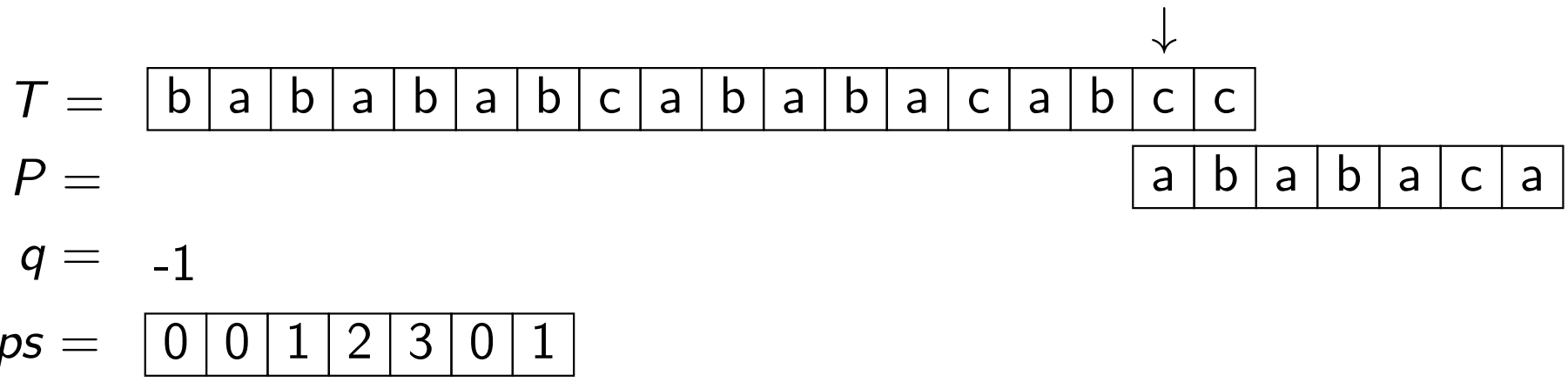
```
1 def delta(q, c, P, lps):
2     m = len(P)
3     while q == m - 1 or (P[q + 1] != c and q > -1):
4         q = lps[q] - 1
5     if P[q + 1] == c:  q += 1
6     return q
```



Der Knuth-Morris-Pratt-Algorithmus

Mustersuche mit dem KMP:

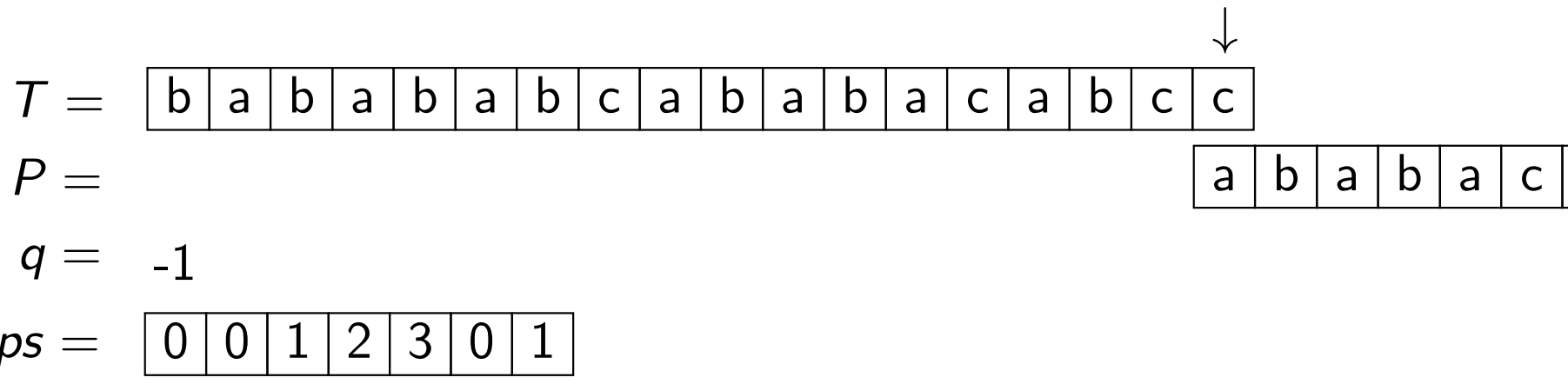
```
1 def delta(q, c, P, lps):
2     m = len(P)
3     while q == m - 1 or (P[q + 1] != c and q > -1):
4         q = lps[q] - 1
5     if P[q + 1] == c:  q += 1
6     return q
```



Der Knuth-Morris-Pratt-Algorithmus

Mustersuche mit dem KMP:

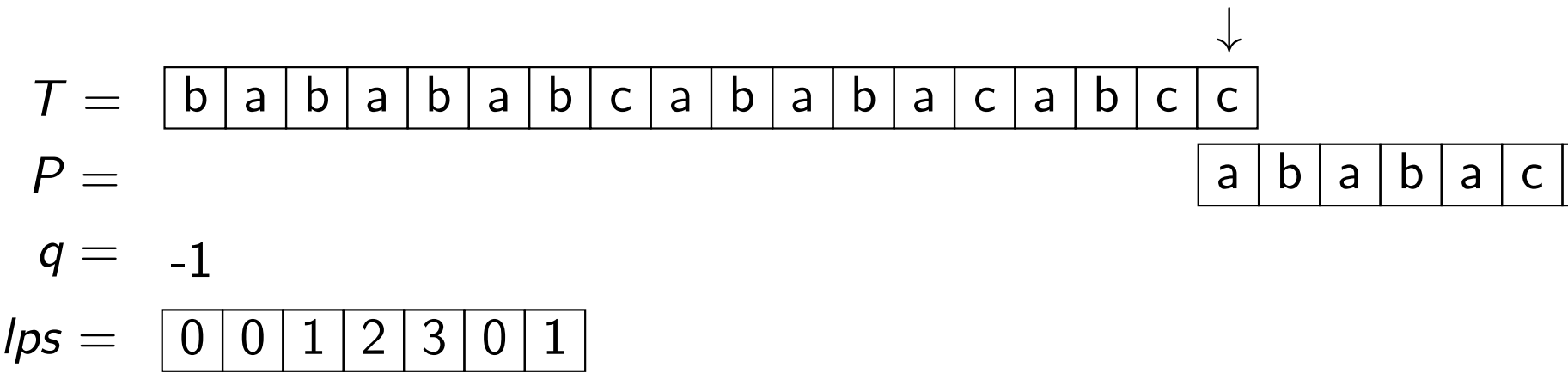
```
1 def delta(q, c, P, lps):
2     m = len(P)
3     while q == m - 1 or (P[q + 1] != c and q > -1):
4         q = lps[q] - 1
5     if P[q + 1] == c:  q += 1
6     return q
```



Der Knuth-Morris-Pratt-Algorithmus

Mustersuche mit dem KMP:

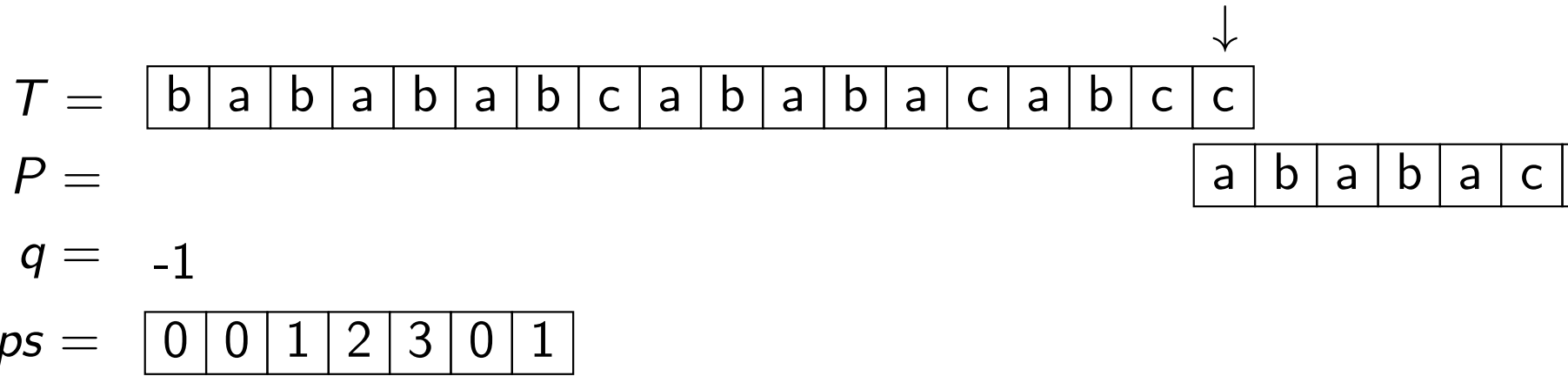
```
1 def delta(q, c, P, lps):
2     m = len(P)
3     while q == m - 1 or (P[q + 1] != c and q > -1):
4         q = lps[q] - 1
5     if P[q + 1] == c:  q += 1
6     return q
```



Der Knuth-Morris-Pratt-Algorithmus

Mustersuche mit dem KMP:

```
1 def delta(q, c, P, lps):
2     m = len(P)
3     while q == m - 1 or (P[q + 1] != c and q > -1):
4         q = lps[q] - 1
5     if P[q + 1] == c:  q += 1
6     return q
```



Zusammenfassung DFA + KMP

- Mit Hilfe von Automaten muss jedes Zeichen im Text nur einmal betrachtet werden.
- Der Knuth-Morris-Pratt-Algorithmus realisiert einen DFA für Pattern-Matching.
- Laufzeiten:
  - Konstruktion: $\mathcal{O}(m)$
  - Suche: $\mathcal{O}(n)$
- Der KMP ist sehr Cache-effizient, da der Text sequenziell von vorn nach hinten gelesen wird.

Ein weiterer PM-Algorithmus

Beobachtungen:

- Die Verwaltung der aktiven Zustandsmenge ist relativ kompliziert.
- Die Suche nach dem nächsten aktiven Zustand kann bis zu $\mathcal{O}(m)$ dauern.
- Information, ob ein Zustand aktiv ist oder nicht, ist binär.

Ein weiterer PM-Algorithmus

Beobachtungen:

- Die Verwaltung der aktiven Zustandsmenge ist relativ kompliziert.
- Die Suche nach dem nächsten aktiven Zustand kann bis zu $\mathcal{O}(m)$ dauern.
- Information, ob ein Zustand aktiv ist oder nicht, ist binär.
- Idee: Wenn Patternlänge m kleiner Registergröße W ist, kann die Information über die aktiven Zustände in einem Register gespeichert werden.

Der Shift-And-Algorithmus

- Die Zustände des NFA sollen parallel simuliert werden.
- Wenn $m \leq W$, beträgt die Such-Laufzeit dann $\mathcal{O}(n\lceil m/W \rceil) = \mathcal{O}(n)$.
- Ein Zustand $0 \leq q < |P|$ ist aktiv, wenn der Präfix $P[: q+1]$ gelesen wurde.
- Da der Zustand -1 immer aktiv ist, muss dieser nicht mitsimuliert werden.

Der Shift-And-Algorithmus

Durchführung:

- Integer A speichert mit $|P|$ Bits alle Zustände.
- Zu Beginn ist kein Zustand aktiv, also $A = 0$.
- Beim Lesen eines Zeichens sollen alle eine Position links shiften.
- Durch Ver*oder*n mit 1 wird der Startzustand hinzugefügt.
- Durch Ver*und*en mit $mask^c$ für alle $c \in \Sigma$ werden alle nicht mehr aktiven Zustände gestrichen.
- Masken folgendermaßen definiert: $mask^c[i] := \big[P[i] = c\big]$.
- Nach jedem Ver*und*en prüfen ob Zustand $m - 1$ aktiv ist.

Der Shift-And-Algorithmus

```
1  def create_masks(P, S):
2      mask = {s: 0 for s in S}
3      for i, c in enumerate(P):
4          mask[c] |= 1 << i
5      return mask
6
7  def shift_and(P, T):
8      m, n, A, S = len(P), len(T), 0, set(P + T)
9      accept, mask = 1 << (m - 1), create_masks(P, S)
10     for i in range(n):
11         A = ((A << 1) | 1) & mask[T[i]]
12         if A & accept:
13             yield i - m + 1, i + 1
```

Der Shift-And-Algorithmus

Erstellen der Masken:

```
1 def create_masks(P, S):
2     mask = {s: 0 for s in S}
3     for i, c in enumerate(P):
4         mask[c] |= 1 << i
5     return mask
```

$P =$ a b a b a c a

$mask = c$:

	6 … 0
a	0000000
b	0000000
c	0000000

Der Shift-And-Algorithmus

Erstellen der Masken:

```
1 def create_masks(P, S):
2     mask = {s: 0 for s in S}
3     for i, c in enumerate(P):
4         mask[c] |= 1 << i
5     return mask
```

$P =$ a b a b a c a

$mask = c$:

	6 … 0
a	0000000
b	0000000
c	0000000

Der Shift-And-Algorithmus

Erstellen der Masken:

```
1 def create_masks(P, S):
2     mask = {s: 0 for s in S}
3     for i, c in enumerate(P):
4         mask[c] |= 1 << i
5     return mask
```

$P =$ a b a b a c a

$mask = c$:

	6 … 0
a	0000000
b	0000000
c	0000000

Der Shift-And-Algorithmus

Erstellen der Masken:

```
1 def create_masks(P, S):
2     mask = {s: 0 for s in S}
3     for i, c in enumerate(P):
4         mask[c] |= 1 << i
5     return mask
```

$P =$ a b a b a c a

$mask = c$:

	6 ... 0
a	0000001
b	0000000
c	0000000

Der Shift-And-Algorithmus

Erstellen der Masken:

```
1 def create_masks(P, S):
2     mask = {s: 0 for s in S}
3     for i, c in enumerate(P):
4         mask[c] |= 1 << i
5     return mask
```

$P =$ a b a b a c a

$mask = c$:

	6 … 0
a	0000001
b	0000000
c	0000000

Der Shift-And-Algorithmus

Erstellen der Masken:

```
1 def create_masks(P, S):
2     mask = {s: 0 for s in S}
3     for i, c in enumerate(P):
4         mask[c] |= 1 << i
5     return mask
```

$P =$ a b a b a c a

$mask = c$:

	6 ... 0
a	0000001
b	0000010
c	0000000

Der Shift-And-Algorithmus

Erstellen der Masken:

```
1 def create_masks(P, S):
2     mask = {s: 0 for s in S}
3     for i, c in enumerate(P):
4         mask[c] |= 1 << i
5     return mask
```

$P =$ a b a b a c a

$mask = c$:

	6 … 0
a	0000001
b	0000010
c	0000000

Der Shift-And-Algorithmus

Erstellen der Masken:

```
1 def create_masks(P, S):
2     mask = {s: 0 for s in S}
3     for i, c in enumerate(P):
4         mask[c] |= 1 << i
5     return mask
```

$P =$ a b a b a c a

$mask = c$:

	6 ... 0
a	0000101
b	0000010
c	0000000

Der Shift-And-Algorithmus

Erstellen der Masken:

```
1 def create_masks(P, S):
2     mask = {s: 0 for s in S}
3     for i, c in enumerate(P):
4         mask[c] |= 1 << i
5     return mask
```

$P =$ a b a b a c a

$mask = c$:

	6 ... 0
a	0000101
b	0000010
c	0000000

Der Shift-And-Algorithmus

Erstellen der Masken:

```
1 def create_masks(P, S):
2     mask = {s: 0 for s in S}
3     for i, c in enumerate(P):
4         mask[c] |= 1 << i
5     return mask
```

$P =$ a b a b a c a

$mask = c$:

	6 ... 0
a	0000101
b	0001010
c	0000000

Der Shift-And-Algorithmus

Erstellen der Masken:

```
1 def create_masks(P, S):
2     mask = {s: 0 for s in S}
3     for i, c in enumerate(P):
4         mask[c] |= 1 << i
5     return mask
```

$P =$ a b a b a c a

$mask = c$:

	6 … 0
a	0000101
b	0001010
c	0000000

Der Shift-And-Algorithmus

Erstellen der Masken:

```
1 def create_masks(P, S):
2     mask = {s: 0 for s in S}
3     for i, c in enumerate(P):
4         mask[c] |= 1 << i
5     return mask
```

$P =$ a b a b a c a

$mask = c$:

	6						0
a	0	0	1	0	1	0	1
b	0	0	0	1	0	1	0
c	0	0	0	0	0	0	0

Der Shift-And-Algorithmus

Erstellen der Masken:

```
1 def create_masks(P, S):
2     mask = {s: 0 for s in S}
3     for i, c in enumerate(P):
4         mask[c] |= 1 << i
5     return mask
```

$P =$ a b a b a c a

$mask = c$:

	6 ... 0
a	0010101
b	0001010
c	0000000

Der Shift-And-Algorithmus

Erstellen der Masken:

```
1 def create_masks(P, S):
2     mask = {s: 0 for s in S}
3     for i, c in enumerate(P):
4         mask[c] |= 1 << i
5     return mask
```

$P =$ a b a b a c a

$mask = c$:

	6						0
a	0	0	1	0	1	0	1
b	0	0	0	1	0	1	0
c	0	1	0	0	0	0	0

Der Shift-And-Algorithmus

Erstellen der Masken:

```
1 def create_masks(P, S):
2     mask = {s: 0 for s in S}
3     for i, c in enumerate(P):
4         mask[c] |= 1 << i
5     return mask
```

$P =$ a b a b a c a

$mask = c$:

	6 ... 0
a	0010101
b	0001010
c	0100000

Der Shift-And-Algorithmus

Erstellen der Masken:

```
1 def create_masks(P, S):
2     mask = {s: 0 for s in S}
3     for i, c in enumerate(P):
4         mask[c] |= 1 << i
5     return mask
```

$P =$ a b a b a c a

$mask = c$:

	6						0
a	1	0	1	0	1	0	1
b	0	0	0	1	0	1	0
c	0	1	0	0	0	0	0

Der Shift-And-Algorithmus

Erstellen der Masken:

```
1 def create_masks(P, S):
2     mask = {s: 0 for s in S}
3     for i, c in enumerate(P):
4         mask[c] |= 1 << i
5     return mask
```

$P =$ a b a b a c a

$mask = c$:

	6 ... 0
a	1010101
b	0001010
c	0100000

Der Shift-And-Algorithmus

Mustersuche mit dem Shift-And-Algorithmus:

```
1 [...]
2     for i in range(n):
3         A = ((A << 1) | 1) & mask[T[i]]
4         if A & accept:
5             yield i - m + 1, i + 1
```

$T =$	a b c a b a b a c a b c	*mask:*	
$A =$	0000000	a:	1010101
op:		b:	0001010
accept $=$	1000000	c:	0100000

Der Shift-And-Algorithmus

Mustersuche mit dem Shift-And-Algorithmus:

```
1 [...]
2     for i in range(n):
3         A = ((A << 1) | 1) & mask[T[i]]
4         if A & accept:
5             yield i - m + 1, i + 1
```

$T =$	a b c a b a b a c a b c	*mask:*	
$A =$	0000000	a:	1010101
op:	$\ll$	b:	0001010
accept $=$	1000000	c:	0100000

Der Shift-And-Algorithmus

Mustersuche mit dem Shift-And-Algorithmus:

```
1 [...]
2     for i in range(n):
3         A = ((A << 1) | 1) & mask[T[i]]
4         if A & accept:
5             yield i - m + 1, i + 1
```

$T =$	a b c a b a b a c a b c	*mask:*
$A =$	0000000	a: 1010101
op:	| 0000001	b: 0001010
accept $=$	1000000	c: 0100000

Der Shift-And-Algorithmus

Mustersuche mit dem Shift-And-Algorithmus:

```
1 [...]
2     for i in range(n):
3         A = ((A << 1) | 1) & mask[T[i]]
4         if A & accept:
5             yield i - m + 1, i + 1
```

$T =$	a b c a b a b a c a b c		*mask:*	
$A =$	0000001		a:	1010101
op:	&	1010101	b:	0001010
accept $=$	1000000		c:	0100000

Der Shift-And-Algorithmus

Mustersuche mit dem Shift-And-Algorithmus:

```
1 [...]
2     for i in range(n):
3         A = ((A << 1) | 1) & mask[T[i]]
4         if A & accept:
5             yield i - m + 1, i + 1
```

$T =$	a b c a b a b a c a b c	*mask:*	
$A =$	0000001	a:	1010101
op:	A & *accept*	b:	0001010
accept $=$	1000000	c:	0100000

Der Shift-And-Algorithmus

Mustersuche mit dem Shift-And-Algorithmus:

```
1 [...]
2     for i in range(n):
3         A = ((A << 1) | 1) & mask[T[i]]
4         if A & accept:
5             yield i - m + 1, i + 1
```

$T =$	a b c a b a b a c a b c	*mask:*	
$A =$	0000001	a:	1010101
op:	$\ll$	b:	0001010
accept $=$	1000000	c:	0100000

Der Shift-And-Algorithmus

Mustersuche mit dem Shift-And-Algorithmus:

```
1 [...]
2     for i in range(n):
3         A = ((A << 1) | 1) & mask[T[i]]
4         if A & accept:
5             yield i - m + 1, i + 1
```

$T =$	a b c a b a b a c a b c	*mask:*	
$A =$	0000010	a:	1010101
op:	| 0000001	b:	0001010
accept $=$	1000000	c:	0100000

Der Shift-And-Algorithmus

Mustersuche mit dem Shift-And-Algorithmus:

```
1 [...]
2     for i in range(n):
3         A = ((A << 1) | 1) & mask[T[i]]
4         if A & accept:
5             yield i - m + 1, i + 1
```

$T =$ a b c a b a b a c a b c

$A =$ 0000011

op: & 0001010

$accept =$ 1000000

mask:

a: 1010101

b: 0001010

c: 0100000

Der Shift-And-Algorithmus

Mustersuche mit dem Shift-And-Algorithmus:

```
1 [...]
2     for i in range(n):
3         A = ((A << 1) | 1) & mask[T[i]]
4         if A & accept:
5             yield i - m + 1, i + 1
```

$T =$ a b c a b a b a c a b c

$A =$ 0000010

op: A & $accept$

$accept =$ 1000000

mask:

a: 1010101

b: 0001010

c: 0100000

Der Shift-And-Algorithmus

Mustersuche mit dem Shift-And-Algorithmus:

```
1 [...]
2     for i in range(n):
3         A = ((A << 1) | 1) & mask[T[i]]
4         if A & accept:
5             yield i - m + 1, i + 1
```

$T =$	a b c a b a b a c a b c	*mask:*	
$A =$	0000010	a:	1010101
op:	$\ll$	b:	0001010
accept $=$	1000000	c:	0100000

Der Shift-And-Algorithmus

Mustersuche mit dem Shift-And-Algorithmus:

```
1 [...]
2     for i in range(n):
3         A = ((A << 1) | 1) & mask[T[i]]
4         if A & accept:
5             yield i - m + 1, i + 1
```

$T =$ a b c a b a b a c a b c

$A =$ 0000100

op: | 0000001

$accept =$ 1000000

mask:

a: 1010101

b: 0001010

c: 0100000

Der Shift-And-Algorithmus

Mustersuche mit dem Shift-And-Algorithmus:

```
1 [...]
2     for i in range(n):
3         A = ((A << 1) | 1) & mask[T[i]]
4         if A & accept:
5             yield i - m + 1, i + 1
```

$T =$ a b c a b a b a c a b c

$A =$ 0000101

op: & 0100000

$accept =$ 1000000

mask:

a: 1010101

b: 0001010

c: 0100000

Der Shift-And-Algorithmus

Mustersuche mit dem Shift-And-Algorithmus:

```
1 [...]
2     for i in range(n):
3         A = ((A << 1) | 1) & mask[T[i]]
4         if A & accept:
5             yield i - m + 1, i + 1
```

$T =$	a b c a b a b a c a b c	*mask:*	
$A =$	0000000	a:	1010101
op:	A & *accept*	b:	0001010
accept $=$	1000000	c:	0100000

Der Shift-And-Algorithmus

Mustersuche mit dem Shift-And-Algorithmus:

```
1 [...]
2     for i in range(n):
3         A = ((A << 1) | 1) & mask[T[i]]
4         if A & accept:
5             yield i - m + 1, i + 1
```

$T =$	a b c a b a b a c a b c	*mask:*	
$A =$	0000000	a:	1010101
op:	$\ll$	b:	0001010
accept $=$	1000000	c:	0100000

Der Shift-And-Algorithmus

Mustersuche mit dem Shift-And-Algorithmus:

```
1 [...]
2     for i in range(n):
3         A = ((A << 1) | 1) & mask[T[i]]
4         if A & accept:
5             yield i - m + 1, i + 1
```

$T =$ a b c a b a b a c a b c

$A =$ 0000000

op: | 0000001

$accept =$ 1000000

mask:

a: 1010101

b: 0001010

c: 0100000

Der Shift-And-Algorithmus

Mustersuche mit dem Shift-And-Algorithmus:

```
1 [...]
2     for i in range(n):
3         A = ((A << 1) | 1) & mask[T[i]]
4         if A & accept:
5             yield i - m + 1, i + 1
```

$T =$ a b c a b a b a c a b c

$A =$ 0000001

op: & 1010101

$accept =$ 1000000

mask:

a: 1010101

b: 0001010

c: 0100000

Der Shift-And-Algorithmus

Mustersuche mit dem Shift-And-Algorithmus:

```
1 [...]
2     for i in range(n):
3         A = ((A << 1) | 1) & mask[T[i]]
4         if A & accept:
5             yield i - m + 1, i + 1
```

$T =$	a b c a b a b a c a b c	*mask:*	
$A =$	0000001	a:	1010101
op:	A & *accept*	b:	0001010
accept $=$	1000000	c:	0100000

Der Shift-And-Algorithmus

Mustersuche mit dem Shift-And-Algorithmus:

```
1 [...]
2     for i in range(n):
3         A = ((A << 1) | 1) & mask[T[i]]
4         if A & accept:
5             yield i - m + 1, i + 1
```

$T =$	a b c a b a b a c a b c	*mask:*
$A =$	0000001	a: 1010101
op:	$\ll$	b: 0001010
accept $=$	1000000	c: 0100000

Der Shift-And-Algorithmus

Mustersuche mit dem Shift-And-Algorithmus:

```
1 [...]
2     for i in range(n):
3         A = ((A << 1) | 1) & mask[T[i]]
4         if A & accept:
5             yield i - m + 1, i + 1
```

$T =$	a b c a b a b a c a b c	*mask:*	
$A =$	0000010	a:	1010101
op:	| 0000001	b:	0001010
accept $=$	1000000	c:	0100000

Der Shift-And-Algorithmus

Mustersuche mit dem Shift-And-Algorithmus:

```
1 [...]
2     for i in range(n):
3         A = ((A << 1) | 1) & mask[T[i]]
4         if A & accept:
5             yield i - m + 1, i + 1
```

$T =$	a b c a b a b a c a b c		*mask:*	
$A =$	0000011		a:	1010101
op:	&	0001010	b:	0001010
accept $=$	1000000		c:	0100000

Der Shift-And-Algorithmus

Mustersuche mit dem Shift-And-Algorithmus:

```
1 [...]
2     for i in range(n):
3         A = ((A << 1) | 1) & mask[T[i]]
4         if A & accept:
5             yield i - m + 1, i + 1
```

$T =$	a b c a b a b a c a b c	*mask:*
$A =$	0000010	a: 1010101
op:	A & *accept*	b: 0001010
accept $=$	1000000	c: 0100000

Der Shift-And-Algorithmus

Mustersuche mit dem Shift-And-Algorithmus:

```
1 [...]
2     for i in range(n):
3         A = ((A << 1) | 1) & mask[T[i]]
4         if A & accept:
5             yield i - m + 1, i + 1
```

$T =$	a b c a b a b a c a b c	*mask:*	
$A =$	0000010	a:	1010101
op:	$\ll$	b:	0001010
accept $=$	1000000	c:	0100000

Der Shift-And-Algorithmus

Mustersuche mit dem Shift-And-Algorithmus:

```
1 [...]
2     for i in range(n):
3         A = ((A << 1) | 1) & mask[T[i]]
4         if A & accept:
5             yield i - m + 1, i + 1
```

$T =$	a b c a b a b a c a b c	*mask:*
$A =$	0000100	a: 1010101
op:	| 0000001	b: 0001010
accept $=$	1000000	c: 0100000

Der Shift-And-Algorithmus

Mustersuche mit dem Shift-And-Algorithmus:

```
1 [...]
2     for i in range(n):
3         A = ((A << 1) | 1) & mask[T[i]]
4         if A & accept:
5             yield i - m + 1, i + 1
```

$T =$ a b c a b a b a c a b c

$A =$ 0000101

op: & 1010101

$accept =$ 1000000

mask:

a: 1010101

b: 0001010

c: 0100000

Der Shift-And-Algorithmus

Mustersuche mit dem Shift-And-Algorithmus:

```
1 [...]
2     for i in range(n):
3         A = ((A << 1) | 1) & mask[T[i]]
4         if A & accept:
5             yield i - m + 1, i + 1
```

$T =$ a b c a b a b a c a b c

$A =$ 0000101

op: A & $accept$

$accept =$ 1000000

mask:

a: 1010101

b: 0001010

c: 0100000

Der Shift-And-Algorithmus

Mustersuche mit dem Shift-And-Algorithmus:

```
1 [...]
2     for i in range(n):
3         A = ((A << 1) | 1) & mask[T[i]]
4         if A & accept:
5             yield i - m + 1, i + 1
```

$T =$	a b c a b a b a c a b c	*mask:*	
$A =$	0000101	a:	1010101
op:	$\ll$	b:	0001010
accept $=$	1000000	c:	0100000

Der Shift-And-Algorithmus

Mustersuche mit dem Shift-And-Algorithmus:

```
1 [...]
2     for i in range(n):
3         A = ((A << 1) | 1) & mask[T[i]]
4         if A & accept:
5             yield i - m + 1, i + 1
```

$T =$ a b c a b a b a c a b c

$A =$ 0001010

op: | 0000001

$accept =$ 1000000

mask:

a: 1010101

b: 0001010

c: 0100000

Der Shift-And-Algorithmus

Mustersuche mit dem Shift-And-Algorithmus:

```
1 [...]
2     for i in range(n):
3         A = ((A << 1) | 1) & mask[T[i]]
4         if A & accept:
5             yield i - m + 1, i + 1
```

$T =$ a b c a b a b a c a b c

$A =$ 0001011

op: & 0001010

$accept =$ 1000000

mask:

a: 1010101

b: 0001010

c: 0100000

Der Shift-And-Algorithmus

Mustersuche mit dem Shift-And-Algorithmus:

```
1 [...]
2     for i in range(n):
3         A = ((A << 1) | 1) & mask[T[i]]
4         if A & accept:
5             yield i - m + 1, i + 1
```

$T =$ a b c a b a b a c a b c

$A =$ 0001010

op: A & $accept$

$accept =$ 1000000

mask:

a: 1010101

b: 0001010

c: 0100000

Der Shift-And-Algorithmus

Mustersuche mit dem Shift-And-Algorithmus:

```
1 [...]
2     for i in range(n):
3         A = ((A << 1) | 1) & mask[T[i]]
4         if A & accept:
5             yield i - m + 1, i + 1
```

$T =$	a b c a b a b a c a b c	*mask:*	
$A =$	0001010	a:	1010101
op:	$\ll$	b:	0001010
accept $=$	1000000	c:	0100000

Der Shift-And-Algorithmus

Mustersuche mit dem Shift-And-Algorithmus:

```
1 [...]
2     for i in range(n):
3         A = ((A << 1) | 1) & mask[T[i]]
4         if A & accept:
5             yield i - m + 1, i + 1
```

$T =$ a b c a b a b a c a b c

$A =$ 0010100

op: | 0000001

$accept =$ 1000000

mask:

a: 1010101

b: 0001010

c: 0100000

Der Shift-And-Algorithmus

Mustersuche mit dem Shift-And-Algorithmus:

```
1 [...]
2     for i in range(n):
3         A = ((A << 1) | 1) & mask[T[i]]
4         if A & accept:
5             yield i - m + 1, i + 1
```

$T =$ a b c a b a b a c a b c

$A =$ 0010101

op: & 1010101

$accept =$ 1000000

mask:

a: 1010101

b: 0001010

c: 0100000

Der Shift-And-Algorithmus

Mustersuche mit dem Shift-And-Algorithmus:

```
1 [...]
2     for i in range(n):
3         A = ((A << 1) | 1) & mask[T[i]]
4         if A & accept:
5             yield i - m + 1, i + 1
```

$T =$ a b c a b a b a c a b c

$A =$ 0010101

op: A & $accept$

$accept =$ 1000000

mask:

a: 1010101

b: 0001010

c: 0100000

Der Shift-And-Algorithmus

Mustersuche mit dem Shift-And-Algorithmus:

```
1 [...]
2     for i in range(n):
3         A = ((A << 1) | 1) & mask[T[i]]
4         if A & accept:
5             yield i - m + 1, i + 1
```

$T =$	a b c a b a b a c a b c	*mask:*	
$A =$	0010101	a:	1010101
op:	$\ll$	b:	0001010
accept $=$	1000000	c:	0100000

Der Shift-And-Algorithmus

Mustersuche mit dem Shift-And-Algorithmus:

```
1 [...]
2     for i in range(n):
3         A = ((A << 1) | 1) & mask[T[i]]
4         if A & accept:
5             yield i - m + 1, i + 1
```

$T =$ a b c a b a b a c a b c

$A =$ 0101010

op: | 0000001

$accept =$ 1000000

mask:

a: 1010101

b: 0001010

c: 0100000

Der Shift-And-Algorithmus

Mustersuche mit dem Shift-And-Algorithmus:

```
1 [...]
2     for i in range(n):
3         A = ((A << 1) | 1) & mask[T[i]]
4         if A & accept:
5             yield i - m + 1, i + 1
```

$T =$ a b c a b a b a c a b c

$A =$ 0101011

op: & 0100000

$accept =$ 1000000

mask:

a: 1010101

b: 0001010

c: 0100000

Der Shift-And-Algorithmus

Mustersuche mit dem Shift-And-Algorithmus:

```
1 [...]
2     for i in range(n):
3         A = ((A << 1) | 1) & mask[T[i]]
4         if A & accept:
5             yield i - m + 1, i + 1
```

$T =$ a b c a b a b a c a b c

$A =$ 0100000

op: A & $accept$

$accept =$ 1000000

mask:

a: 1010101

b: 0001010

c: 0100000

Der Shift-And-Algorithmus

Mustersuche mit dem Shift-And-Algorithmus:

```
1 [...]
2     for i in range(n):
3         A = ((A << 1) | 1) & mask[T[i]]
4         if A & accept:
5             yield i - m + 1, i + 1
```

$T =$	a b c a b a b a c a b c	*mask:*	
$A =$	0100000	a:	1010101
op:	$\ll$	b:	0001010
accept $=$	1000000	c:	0100000

Der Shift-And-Algorithmus

Mustersuche mit dem Shift-And-Algorithmus:

```
1 [...]
2     for i in range(n):
3         A = ((A << 1) | 1) & mask[T[i]]
4         if A & accept:
5             yield i - m + 1, i + 1
```

$T =$	a b c a b a b a c a b c	*mask:*
$A =$	1000000	a: 1010101
op:	| 0000001	b: 0001010
accept $=$	1000000	c: 0100000

Der Shift-And-Algorithmus

Mustersuche mit dem Shift-And-Algorithmus:

```
1 [...]
2     for i in range(n):
3         A = ((A << 1) | 1) & mask[T[i]]
4         if A & accept:
5             yield i - m + 1, i + 1
```

$T =$		a b c a b a b a c a b c	*mask:*	
$A =$		1000001	a:	1010101
op:	&	1010101	b:	0001010
accept $=$		1000000	c:	0100000

Der Shift-And-Algorithmus

Mustersuche mit dem Shift-And-Algorithmus:

```
1 [...]
2     for i in range(n):
3         A = ((A << 1) | 1) & mask[T[i]]
4         if A & accept:
5             yield i - m + 1, i + 1
```

$T =$	a b c a b a b a c a b c		*mask:*
$A =$	1000001		a: 1010101
op:	A & *accept*	yield 3, 10	b: 0001010
accept $=$	1000000		c: 0100000

Der Shift-And-Algorithmus

Mustersuche mit dem Shift-And-Algorithmus:

```
1 [...]
2     for i in range(n):
3         A = ((A << 1) | 1) & mask[T[i]]
4         if A & accept:
5             yield i - m + 1, i + 1
```

$T =$	a b c a b a b a c a b c	*mask:*	
$A =$	1000001	a:	1010101
op:	$\ll$	b:	0001010
accept $=$	1000000	c:	0100000

Der Shift-And-Algorithmus

Mustersuche mit dem Shift-And-Algorithmus:

```
1 [...]
2     for i in range(n):
3         A = ((A << 1) | 1) & mask[T[i]]
4         if A & accept:
5             yield i - m + 1, i + 1
```

$T =$	a b c a b a b a c a b c	*mask:*
$A =$	0000010	a: 1010101
op:	| 0000001	b: 0001010
accept $=$	1000000	c: 0100000

Der Shift-And-Algorithmus

Mustersuche mit dem Shift-And-Algorithmus:

```
1 [...]
2     for i in range(n):
3         A = ((A << 1) | 1) & mask[T[i]]
4         if A & accept:
5             yield i - m + 1, i + 1
```

$T =$		a b c a b a b a c a b c	*mask:*	
$A =$		0000011	a:	1010101
op:	&	0001010	b:	0001010
accept $=$		1000000	c:	0100000

Der Shift-And-Algorithmus

Mustersuche mit dem Shift-And-Algorithmus:

```
1 [...]
2     for i in range(n):
3         A = ((A << 1) | 1) & mask[T[i]]
4         if A & accept:
5             yield i - m + 1, i + 1
```

$T =$	a b c a b a b a c a b c	*mask:*	
$A =$	0000010	a:	1010101
op:	A & *accept*	b:	0001010
accept $=$	1000000	c:	0100000

Der Shift-And-Algorithmus

Mustersuche mit dem Shift-And-Algorithmus:

```
1 [...]
2     for i in range(n):
3         A = ((A << 1) | 1) & mask[T[i]]
4         if A & accept:
5             yield i - m + 1, i + 1
```

$T =$	a b c a b a b a c a b c	*mask:*	
$A =$	0000010	a:	1010101
op:	$\ll$	b:	0001010
accept $=$	1000000	c:	0100000

Der Shift-And-Algorithmus

Mustersuche mit dem Shift-And-Algorithmus:

```
1 [...]
2     for i in range(n):
3         A = ((A << 1) | 1) & mask[T[i]]
4         if A & accept:
5             yield i - m + 1, i + 1
```

$T =$ a b c a b a b a c a b c

$A =$ 0000100

op: | 0000001

$accept =$ 1000000

mask:

a: 1010101

b: 0001010

c: 0100000

Der Shift-And-Algorithmus

Mustersuche mit dem Shift-And-Algorithmus:

```
1 [...]
2     for i in range(n):
3         A = ((A << 1) | 1) & mask[T[i]]
4         if A & accept:
5             yield i - m + 1, i + 1
```

$T =$ a b c a b a b a c a b c

$A =$ 0000101

op: & 0100000

$accept =$ 1000000

mask:

a: 1010101

b: 0001010

c: 0100000

Der Shift-And-Algorithmus

Mustersuche mit dem Shift-And-Algorithmus:

```
1 [...]
2     for i in range(n):
3         A = ((A << 1) | 1) & mask[T[i]]
4         if A & accept:
5             yield i - m + 1, i + 1
```

$T =$	a b c a b a b a c a b c	*mask:*	
$A =$	0000000	a:	1010101
op:	A & *accept*	b:	0001010
accept $=$	1000000	c:	0100000

Der Shift-And-Algorithmus

Mustersuche mit dem Shift-And-Algorithmus:

```
1 [...]
2     for i in range(n):
3         A = ((A << 1) | 1) & mask[T[i]]
4         if A & accept:
5             yield i - m + 1, i + 1
```

$T =$	a b c a b a b a c a b c	*mask:*	
$A =$	0000000	a:	1010101
op:		b:	0001010
accept $=$	1000000	c:	0100000

Zusammenfassung

- Der Shift-And-Algorithmus nutzt Bitparallelismus für die Zustandsübergangsfunktion des NFA aus.
- Die Laufzeit beträgt $\mathcal{O}(n\lceil m/W \rceil)$, Algorithmus also nur sinnvoll, wenn $m \leq W$.
- Der zusätzliche Speicherplatz beträgt $\mathcal{O}(|\Sigma|\lceil m/W \rceil)$.
- Sehr schlanker und eleganter Code.
- Algorithmus sollte auf hardwarenahen Programmiersprachen (wie C/C++) implementiert werden, da alles Bitoperationen sind.
- Durch Umstellen der Bitlogik zum Shift-Or entfällt eine Operation (setzen des Startbits).

Noch ein weiterer PM-Algorithmus

Beobachtungen:

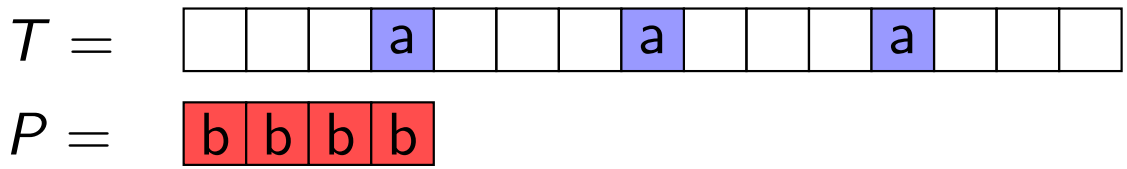
- Bisherige PM-Algorithmen haben jeden Buchstaben im Text exakt einmal gelesen.
- Was ist die theoretisch kleinste Anzahl von Vergleichen zwischen P und T?

Noch ein weiterer PM-Algorithmus

Beobachtungen:

- Bisherige PM-Algorithmen haben jeden Buchstaben im Text exakt einmal gelesen.
- Was ist die theoretisch kleinste Anzahl von Vergleichen zwischen P und T?
- Wenn das Pattern von hinten getestet wird und der Testbuchstabe nicht in P vorkommt, kann man direkt das Pattern um m Stellen verschieben.
- Eine untere Schranke für PM ist demnach $\mathcal{O}(n/m)$.

Szenario für untere Schranke



- Offensichtlich kann in diesem Szenario das Pattern immer um m Stellen nach rechts verschoben werden.
- Für jedes $\sigma \in \Sigma$ muss also ermittelt werden, wie weit gesprungen werden darf.
- Der Horspool-Algoithmus realisiert diesen Ansatz.

Der Horspool-Algorithmus

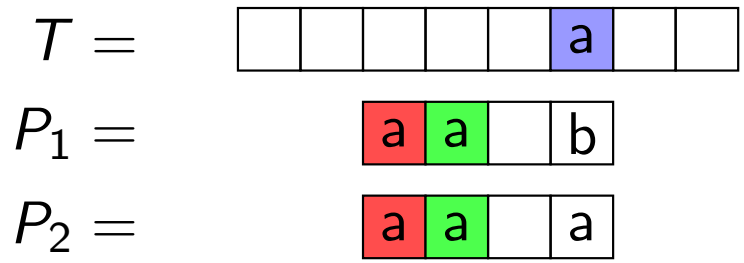
- Das letzte (rechteste) Zeichen $c \in \Sigma$ des Suchfensters wird mit dem Text verglichen.
- Es wird zwischen der *Shift-Phase* und der *Test-Phase* alterniert.

Der Horspool-Algorithmus

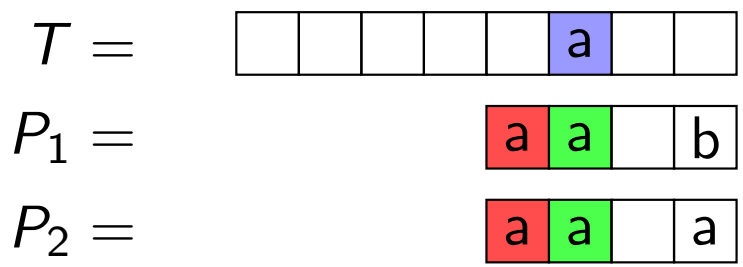
- Das letzte (rechteste) Zeichen $c \in \Sigma$ des Suchfensters wird mit dem Text verglichen.
- Es wird zwischen der *Shift-Phase* und der *Test-Phase* alterniert.
- *Shift-Phase*:
  Sei $l[c] := \max\{\{0 \leq j < m-1 : P[j] = c\} \cup \{-1\}\}$ die rechteste Position von c in P oder -1, wenn nicht vorhanden.
  Da von rechts betrachtet wird, muss das Pattern um $shift[c] := m - 1 - l[c]$ Stellen nach rechts verschoben werden bis der Text zu Ende ist oder c und $T[i]$ matchen.
- *Test-Phase*: Naives PM zwischen $T[i-m+1 : i]$ und $P[: m-1]$.

Der Horspool-Algorithmus

Die Shift-Phase im Detail:



Nach der Shift-Phase:



Der Horspool-Algorithmus

Erstellen der Sprungtabelle:

```
1 def create_shift(P, S):
2     m = len(P)
3     shift = {s: m for s in S}
4     for i, c in enumerate(P[:m-1]):
5         shift[c] = m - 1 - i
6     return shift
```

$P =$ a b a b a c a

$shift = c$:

a	7
b	7
c	7

Der Horspool-Algorithmus

Erstellen der Sprungtabelle:

```
1 def create_shift(P, S):
2     m = len(P)
3     shift = {s: m for s in S}
4     for i, c in enumerate(P[:m-1]):
5         shift[c] = m - 1 - i
6     return shift
```

$P =$ a b a b a c a

$shift = c$:

a	7
b	7
c	7

Der Horspool-Algorithmus

Erstellen der Sprungtabelle:

```
1 def create_shift(P, S):
2     m = len(P)
3     shift = {s: m for s in S}
4     for i, c in enumerate(P[:m-1]):
5         shift[c] = m - 1 - i
6     return shift
```

$P =$ a b a b a c a

$shift = c$:

a	7
b	7
c	7

Der Horspool-Algorithmus

Erstellen der Sprungtabelle:

```
1 def create_shift(P, S):
2     m = len(P)
3     shift = {s: m for s in S}
4     for i, c in enumerate(P[:m-1]):
5         shift[c] = m - 1 - i
6     return shift
```

$P =$ a b a b a c a

$shift = c$:

a	6
b	7
c	7

Der Horspool-Algorithmus

Erstellen der Sprungtabelle:

```
1 def create_shift(P, S):
2     m = len(P)
3     shift = {s: m for s in S}
4     for i, c in enumerate(P[:m-1]):
5         shift[c] = m - 1 - i
6     return shift
```

$P =$ a b a b a c a

$shift = c$:

a	6
b	7
c	7

Der Horspool-Algorithmus

Erstellen der Sprungtabelle:

```
1 def create_shift(P, S):
2     m = len(P)
3     shift = {s: m for s in S}
4     for i, c in enumerate(P[:m-1]):
5         shift[c] = m - 1 - i
6     return shift
```

$P =$ a b a b a c a

$shift = c$:

a	6
b	5
c	7

Der Horspool-Algorithmus

Erstellen der Sprungtabelle:

```
1 def create_shift(P, S):
2     m = len(P)
3     shift = {s: m for s in S}
4     for i, c in enumerate(P[:m-1]):
5         shift[c] = m - 1 - i
6     return shift
```

$P =$ a b a b a c a

$shift = c$:

a	6
b	5
c	7

Der Horspool-Algorithmus

Erstellen der Sprungtabelle:

```
1 def create_shift(P, S):
2     m = len(P)
3     shift = {s: m for s in S}
4     for i, c in enumerate(P[:m-1]):
5         shift[c] = m - 1 - i
6     return shift
```

$P =$ a b a b a c a

$shift = c$:

a	4
b	5
c	7

Der Horspool-Algorithmus

Erstellen der Sprungtabelle:

```
1 def create_shift(P, S):
2     m = len(P)
3     shift = {s: m for s in S}
4     for i, c in enumerate(P[:m-1]):
5         shift[c] = m - 1 - i
6     return shift
```

$P =$ a b a b a c a

$shift = c$:

a	4
b	5
c	7

Der Horspool-Algorithmus

Erstellen der Sprungtabelle:

```
1 def create_shift(P, S):
2     m = len(P)
3     shift = {s: m for s in S}
4     for i, c in enumerate(P[:m-1]):
5         shift[c] = m - 1 - i
6     return shift
```

$P =$ a b a b a c a

$shift = c$:

a	4
b	3
c	7

Der Horspool-Algorithmus

Erstellen der Sprungtabelle:

```
1 def create_shift(P, S):
2     m = len(P)
3     shift = {s: m for s in S}
4     for i, c in enumerate(P[:m-1]):
5         shift[c] = m - 1 - i
6     return shift
```

$P =$ a b a b a c a

$shift = c$:

a	4
b	3
c	7

Der Horspool-Algorithmus

Erstellen der Sprungtabelle:

```
1 def create_shift(P, S):
2     m = len(P)
3     shift = {s: m for s in S}
4     for i, c in enumerate(P[:m-1]):
5         shift[c] = m - 1 - i
6     return shift
```

$P =$ a b a b a c a

$shift = c$:

a	2
b	3
c	7

Der Horspool-Algorithmus

Erstellen der Sprungtabelle:

```
1 def create_shift(P, S):
2     m = len(P)
3     shift = {s: m for s in S}
4     for i, c in enumerate(P[:m-1]):
5         shift[c] = m - 1 - i
6     return shift
```

$P =$ a b a b a c a

$shift = c$:

a	2
b	3
c	7

Der Horspool-Algorithmus

Erstellen der Sprungtabelle:

```
1 def create_shift(P, S):
2     m = len(P)
3     shift = {s: m for s in S}
4     for i, c in enumerate(P[:m-1]):
5         shift[c] = m - 1 - i
6     return shift
```

$P =$ a b a b a c a

$shift = c$:

a	2
b	3
c	1

Der Horspool-Algorithmus

Erstellen der Sprungtabelle:

```
1 def create_shift(P, S):
2     m = len(P)
3     shift = {s: m for s in S}
4     for i, c in enumerate(P[:m-1]):
5         shift[c] = m - 1 - i
6     return shift
```

$P =$ a b a b a c a

$shift = c$:

a	2
b	3
c	1

Der Horspool-Algorithmus

Der Algorithmus:

```
1  def horspool(P, T):
2      m, n, S = len(P), len(T), set(P + T)
3      shift = create_shift(P, S)
4      last, last_P = m - 1, P[-1]
5      while True:
6          # shift-phase
7          while last < n and T[last] != last_P:
8              last += shift[T[last]]
9          if last >= n: break
10         # test-phase
11         if T[last - m + 1 : last] == P[:m-1]:
12             yield last - m + 1, last + 1
13         last += shift[last_P]
```

Der Horspool-Algorithmus

Mustersuche mit dem Horspool-Algorithmus:

```
1       while True:
2           while last < n and T[last] != last_P:
3               last += shift[T[last]]
4           if last >= n: break
5           if T[last - m + 1:last] == P[:m-1]:
6               yield last - m + 1, last + 1
7           last += shift[last_P]
```

$T =$ | b | a | b | a | b | a | b | c | a | b | a | b | a | c | a | b | c | c |

$P =$ | a | b | a | b | a | c | a |

$last = 6$

shift:

a: 2, b: 3, c: 1

Der Horspool-Algorithmus

Mustersuche mit dem Horspool-Algorithmus:

```
1       while True:
2           while last < n and T[last] != last_P:
3               last += shift[T[last]]
4           if last >= n: break
5           if T[last - m + 1:last] == P[:m-1]:
6               yield last - m + 1, last + 1
7           last += shift[last_P]
```

$T =$ | b | a | b | a | b | a | b | c | a | b | a | b | a | c | a | b | c | c |

$P =$ | a | b | a | b | a | c | a |

$last = 6$

shift:

a: 2, b: 3, c: 1

Der Horspool-Algorithmus

Mustersuche mit dem Horspool-Algorithmus:

```
1       while True:
2           while last < n and T[last] != last_P:
3               last += shift[T[last]]
4           if last >= n: break
5           if T[last - m + 1:last] == P[:m-1]:
6               yield last - m + 1, last + 1
7           last += shift[last_P]
```

$T =$ | b | a | b | a | b | a | b | c | a | b | a | b | a | c | a | b | c | c |

$P =$ | a | b | a | b | a | c | a |

$last = 6$

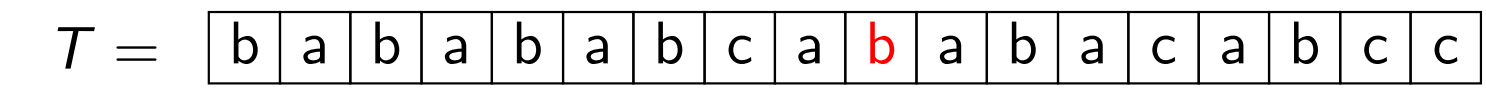
shift:

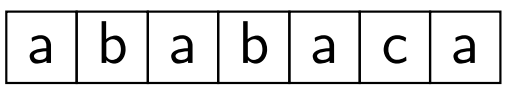
a: 2, b: 3, c: 1

Der Horspool-Algorithmus

Mustersuche mit dem Horspool-Algorithmus:

```
1       while True:
2           while last < n and T[last] != last_P:
3               last += shift[T[last]]
4           if last >= n: break
5           if T[last - m + 1:last] == P[:m-1]:
6               yield last - m + 1, last + 1
7           last += shift[last_P]
```





last = 9

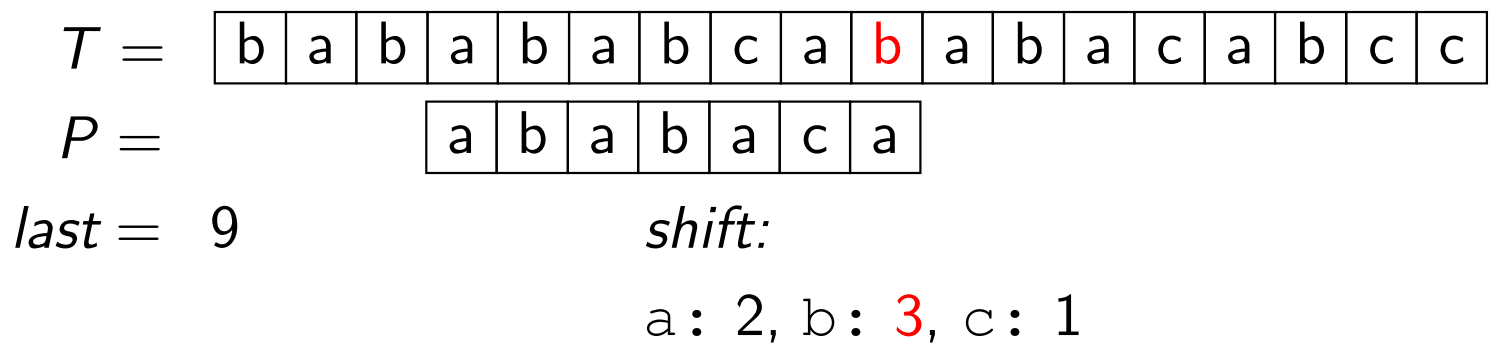
shift:

a: 2, b: 3, c: 1

Der Horspool-Algorithmus

Mustersuche mit dem Horspool-Algorithmus:

```
1       while True:
2           while last < n and T[last] != last_P:
3               last += shift[T[last]]
4           if last >= n: break
5           if T[last - m + 1:last] == P[:m-1]:
6               yield last - m + 1, last + 1
7           last += shift[last_P]
```



Der Horspool-Algorithmus

Mustersuche mit dem Horspool-Algorithmus:

```
1       while True:
2           while last < n and T[last] != last_P:
3               last += shift[T[last]]
4           if last >= n: break
5           if T[last - m + 1:last] == P[:m-1]:
6               yield last - m + 1, last + 1
7           last += shift[last_P]
```

$T =$ | b | a | b | a | b | a | b | c | a | b | a | b | a | c | a | b | c | c |

$P =$ | a | b | a | b | a | c | a |

$last = 12$

shift:

a: 2, b: 3, c: 1

Der Horspool-Algorithmus

Mustersuche mit dem Horspool-Algorithmus:

```
1       while True:
2           while last < n and T[last] != last_P:
3               last += shift[T[last]]
4           if last >= n: break
5           if T[last - m + 1:last] == P[:m-1]:
6               yield last - m + 1, last + 1
7           last += shift[last_P]
```

$T =$ | b | a | b | a | b | a | b | c | a | b | a | b | a | c | a | b | c | c |

$P =$ | a | b | a | b | a | c | a |

$last =$ 12

shift:

a: 2, b: 3, c: 1

Der Horspool-Algorithmus

Mustersuche mit dem Horspool-Algorithmus:

```
1       while True:
2           while last < n and T[last] != last_P:
3               last += shift[T[last]]
4           if last >= n: break
5           if T[last - m + 1:last] == P[:m-1]:
6               yield last - m + 1, last + 1
7           last += shift[last_P]
```

$T =$ | b | a | b | a | b | a | b | c | a | b | a | b | a | c | a | b | c | c |

$P =$ | a | b | a | b | a | c | a |

$last =$ 12

shift:

a: 2, b: 3, c: 1

Der Horspool-Algorithmus

Mustersuche mit dem Horspool-Algorithmus:

```
1       while True:
2           while last < n and T[last] != last_P:
3               last += shift[T[last]]
4           if last >= n: break
5           if T[last - m + 1:last] == P[:m-1]:
6               yield last - m + 1, last + 1
7           last += shift[last_P]
```

$T =$ | b | a | b | a | b | a | b | c | a | b | a | b | a | c | a | b | c | c |

$P =$ | a | b | a | b | a | c | a |

$last = 12$

shift:

a: 2, b: 3, c: 1

Der Horspool-Algorithmus

Mustersuche mit dem Horspool-Algorithmus:

```
1       while True:
2           while last < n and T[last] != last_P:
3               last += shift[T[last]]
4           if last >= n: break
5           if T[last - m + 1:last] == P[:m-1]:
6               yield last - m + 1, last + 1
7           last += shift[last_P]
```

$T =$ | b | a | b | a | b | a | b | c | a | b | a | b | a | c | a | b | c | c |

$P =$ | a | b | a | b | a | c | a |

$last =$ 14

shift:

a: 2, b: 3, c: 1

Der Horspool-Algorithmus

Mustersuche mit dem Horspool-Algorithmus:

```
1       while True:
2           while last < n and T[last] != last_P:
3               last += shift[T[last]]
4           if last >= n: break
5           if T[last - m + 1:last] == P[:m-1]:
6               yield last - m + 1, last + 1
7           last += shift[last_P]
```

$T =$ | b | a | b | a | b | a | b | c | a | b | a | b | a | c | a | b | c | c |

$P =$ | a | b | a | b | a | c | a |

$last = 14$

shift:

a: 2, b: 3, c: 1

Der Horspool-Algorithmus

Mustersuche mit dem Horspool-Algorithmus:

```
1       while True:
2           while last < n and T[last] != last_P:
3               last += shift[T[last]]
4           if last >= n: break
5           if T[last - m + 1:last] == P[:m-1]:
6               yield last - m + 1, last + 1
7           last += shift[last_P]
```

$T =$ | b | a | b | a | b | a | b | c | a | b | a | b | a | c | a | b | c | c |

$P =$ | a | b | a | b | a | c | a |

$last = 14$

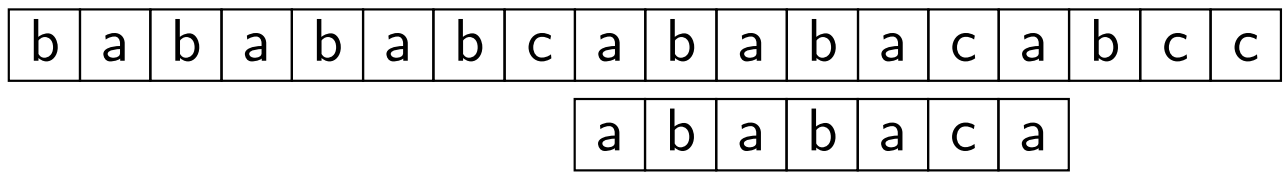
shift:

a: 2, b: 3, c: 1

Der Horspool-Algorithmus

Mustersuche mit dem Horspool-Algorithmus:

```
1       while True:
2           while last < n and T[last] != last_P:
3               last += shift[T[last]]
4           if last >= n: break
5           if T[last - m + 1:last] == P[:m-1]:
6               yield last - m + 1, last + 1
7           last += shift[last_P]
```



last = 14

yield 8, 15

shift:

`a`: 2, `b`: 3, `c`: 1

Der Horspool-Algorithmus

Mustersuche mit dem Horspool-Algorithmus:

```
1       while True:
2           while last < n and T[last] != last_P:
3               last += shift[T[last]]
4           if last >= n: break
5           if T[last - m + 1:last] == P[:m-1]:
6               yield last - m + 1, last + 1
7           last += shift[last_P]
```

$T =$ | b | a | b | a | b | a | b | c | a | b | a | b | a | c | a | b | c | c |

$P =$ | a | b | a | b | a | c | a |

$last = 14$

shift:

a: 2, b: 3, c: 1

Der Horspool-Algorithmus

Mustersuche mit dem Horspool-Algorithmus:

```
1     while True:
2         while last < n and T[last] != last_P:
3             last += shift[T[last]]
4         if last >= n: break
5         if T[last - m + 1:last] == P[:m-1]:
6             yield last - m + 1, last + 1
7         last += shift[last_P]
```

$T =$ | b | a | b | a | b | a | b | c | a | b | a | b | a | c | a | b | c | c |

$P =$ | a | b | a | b | a | c | a |

$last = 16$

shift:

a: 2, b: 3, c: 1

Der Horspool-Algorithmus

Mustersuche mit dem Horspool-Algorithmus:

```
1       while True:
2           while last < n and T[last] != last_P:
3               last += shift[T[last]]
4           if last >= n: break
5           if T[last - m + 1:last] == P[:m-1]:
6               yield last - m + 1, last + 1
7           last += shift[last_P]
```

$T =$ | b | a | b | a | b | a | b | c | a | b | a | b | a | c | a | b | c | c |

$P =$ | a | b | a | b | a | c | a |

$last = 16$

shift:

a: 2, b: 3, c: 1

Der Horspool-Algorithmus

Mustersuche mit dem Horspool-Algorithmus:

```
1     while True:
2         while last < n and T[last] != last_P:
3             last += shift[T[last]]
4         if last >= n: break
5         if T[last - m + 1:last] == P[:m-1]:
6             yield last - m + 1, last + 1
7         last += shift[last_P]
```

$T =$ | b | a | b | a | b | a | b | c | a | b | a | b | a | c | a | b | c | c |

$P =$ | a | b | a | b | a | c | a |

$last = 17$

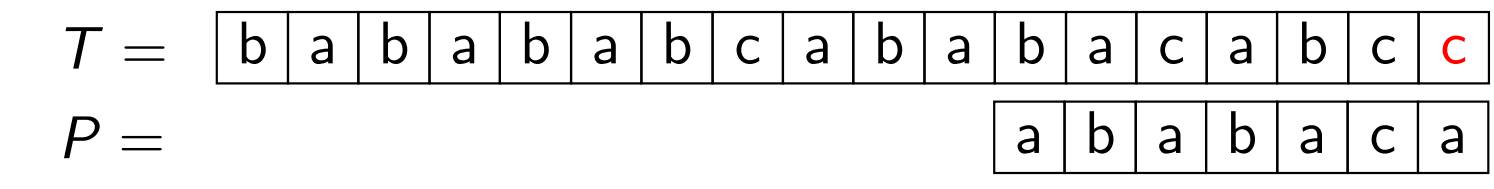
shift:

a: 2, b: 3, c: 1

Der Horspool-Algorithmus

Mustersuche mit dem Horspool-Algorithmus:

```
1       while True:
2           while last < n and T[last] != last_P:
3               last += shift[T[last]]
4           if last >= n: break
5           if T[last - m + 1:last] == P[:m-1]:
6               yield last - m + 1, last + 1
7           last += shift[last_P]
```



$last = 17$

shift:

a: 2, b: 3, c: 1

Der Horspool-Algorithmus

Mustersuche mit dem Horspool-Algorithmus:

```
1       while True:
2           while last < n and T[last] != last_P:
3               last += shift[T[last]]
4           if last >= n: break
5           if T[last - m + 1:last] == P[:m-1]:
6               yield last - m + 1, last + 1
7           last += shift[last_P]
```

$T =$ | b | a | b | a | b | a | b | c | a | b | a | b | a | c | a | b | c | c |

$P =$ | a | b | a | b | a | c | a |

$last = 18$

shift:

a: 2, b: 3, c: 1

Der Horspool-Algorithmus

Mustersuche mit dem Horspool-Algorithmus:

```
1       while True:
2           while last < n and T[last] != last_P:
3               last += shift[T[last]]
4           if last >= n: break
5           if T[last - m + 1:last] == P[:m-1]:
6               yield last - m + 1, last + 1
7           last += shift[last_P]
```

$T =$ | b | a | b | a | b | a | b | c | a | b | a | b | a | c | a | b | c | c |

$P =$ | a | b | a | b | a | c | a |

$last = 18$

shift:

a: 2, b: 3, c: 1

Laufzeitanalyse des Horspool-Algorithmus

- Die Best-case-Laufzeit ist $\mathcal{O}(m + n/m)$ für den kompletten Durchlauf.
- Da der naive Vergleich $\mathcal{O}(m)$ dauert, und die while-Schleife bis zu $\mathcal{O}(n)$-mal unterbrochen werden kann, beträgt die Worst-case-Laufzeit $\mathcal{O}(m + nm)$.

Laufzeitanalyse des Horspool-Algorithmus

- Die Best-case-Laufzeit ist $\mathcal{O}(m + n/m)$ für den kompletten Durchlauf.
- Da der naive Vergleich $\mathcal{O}(m)$ dauert, und die while-Schleife bis zu $\mathcal{O}(n)$-mal unterbrochen werden kann, beträgt die Worst-case-Laufzeit $\mathcal{O}(m + nm)$.
- Der Horspool-Algorithmus ist eine vereinfachte Version des *Boyer-Moore*-Algorithmus, welcher auch eine Best-case-Laufzeit von $\mathcal{O}(m + n/m)$ aber mit Tricks eine Worst-case-Laufzeit von $\mathcal{O}(m + n)$ hat.

Laufzeitanalyse des Horspool-Algorithmus

Eine erwartete Laufzeit lässt sich ebenfalls abschätzen:

- Sei Σ_P die Menge der Zeichen, die in P vorkommen
- Es kann angenommen werden, dass ein Zeichen in Σ_P die Shiftlänge $1, 2, 3, ..., |\Sigma_P|$ hat.
- Die Zeichen $\Sigma \setminus \Sigma_P$ haben die Shiftlänge m.
- Sei das letzte Zeichen aus dem Suchfenster ein Zufälliges mit W'keit $1/|\Sigma|$.

Laufzeitanalyse des Horspool-Algorithmus

Die erwartete Shiftlänge $\tilde{s}$ beträgt also mindestens:

$$\tilde{s} \geq \frac{\sum_{i=1}^{|\Sigma_P|} i + (|\Sigma| - |\Sigma_P|) \cdot m}{|\Sigma|} = \frac{|\Sigma_P|(|\Sigma_P| + 1)}{2|\Sigma|} + m\left(1 - \frac{|\Sigma_P|}{|\Sigma|}\right).$$

Laufzeitanalyse des Horspool-Algorithmus

Die erwartete Shiftlänge $\tilde{s}$ beträgt also mindestens:

$$\tilde{s} \geq \frac{\sum_{i=1}^{|\Sigma_P|} i + (|\Sigma| - |\Sigma_P|) \cdot m}{|\Sigma|} = \frac{|\Sigma_P|(|\Sigma_P| + 1)}{2|\Sigma|} + m\left(1 - \frac{|\Sigma_P|}{|\Sigma|}\right).$$

- **Großes Alphabet** $|\Sigma| \in \Theta(m)$:
  - Sei $|\Sigma_P| \in \mathcal{O}(1)$, also $|\Sigma_P| \ll |\Sigma|$, dann liefert der 2. Summand $\tilde{s} \geq \Theta(m)$.
  - Sei $|\Sigma_P| \in \mathcal{O}(m)$, dann liefert der 1. Summand $\tilde{s} \geq \Theta(m)$.

Laufzeitanalyse des Horspool-Algorithmus

Die erwartete Shiftlänge $\tilde{s}$ beträgt also mindestens:

$$\tilde{s} \geq \frac{\sum_{i=1}^{|\Sigma_P|} i + (|\Sigma| - |\Sigma_P|) \cdot m}{|\Sigma|} = \frac{|\Sigma_P|(|\Sigma_P| + 1)}{2|\Sigma|} + m\left(1 - \frac{|\Sigma_P|}{|\Sigma|}\right).$$

- **Großes Alphabet** $|\Sigma| \in \Theta(m)$:
  - Sei $|\Sigma_P| \in \mathcal{O}(1)$, also $|\Sigma_P| \ll |\Sigma|$, dann liefert der 2. Summand $\tilde{s} \geq \Theta(m)$.
  - Sei $|\Sigma_P| \in \mathcal{O}(m)$, dann liefert der 1. Summand $\tilde{s} \geq \Theta(m)$.
- **Kleines Alphabet** $|\Sigma| \in \Theta(1)$:
  - Im Fall $|\Sigma_P| < |\Sigma|$ liefert der 2. Summand $\tilde{s} \geq \Theta(m)$.
  - Im Fall $|\Sigma_P| = |\Sigma|$ ist $\tilde{s} \geq (|\Sigma| + 1)/2$, also $\mathcal{O}(1)$.

Zusammenfassung

- In der Praxis ist der Horspool-Algorithmus gut, wenn $\Sigma_P \ll \Sigma$ gilt.
- Shifts bis zu $\mathcal{O}(m)$ sind möglich, dadurch beträgt die Best-case-Laufzeit $\mathcal{O}(m + n/m)$.
- Der Horspool-A' ist durch geringere Konstanten bei der Laufzeitanalyse in der Regel schneller als der Boyer-Moore-A'.
- Algorithmus nicht Cache-effizient, da rückwärts gelesen wird.

Und noch ein weiterer PM-Algorithmus

- Es ist weiterhin wünschenswert weite Teile des Textes zu überspringen.
- Es sollte nicht abgebrochen werden, sobald es einen Mismatch zwischen Text und Pattern gibt, sondern
- erst abbrechen, sobald der gelesene Teil kein Teilstring vom Pattern ist.
- Hier spricht man von einem Teilstring-basierten Ansatz.

Beobachtung

Wenn nachgehalten wird, was das längste Suffix des aktuellen Fensters ist das auch Präfix des Patterns ist, kann effizient das Fenster verschoben werden. Eine hierfür sinnvolle Datenstruktur erlaubt

- einem gelesenen Fenster von rechts nach links Zeichen anzufügen,
- festzustellen, ob der bisher gelesene Teil ein Teilstring des Patterns ist,
- festzustellen, ob der bisher gelesene Teil sogar ein Präfix des Patterns ist.

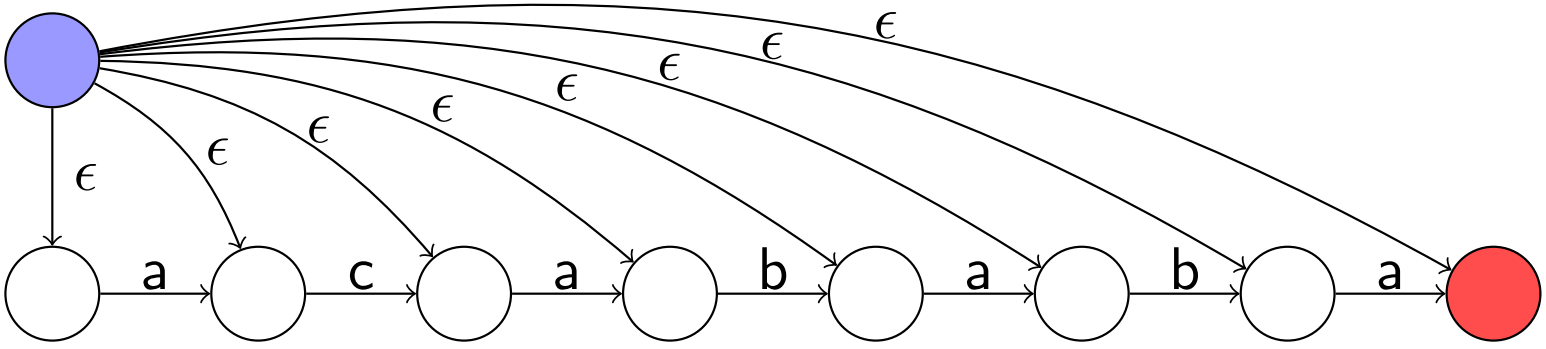
Der Suffixautomat

Der (deterministische) *Suffixautomat* für den String x ist ein DFA mit folgenden Eigenschaften:

- Es existiert vom Startzustand aus ein Pfad mit Label y genau dann, wenn y ein Teilstring von x ist.
- Der Pfad mit Label y endet genau dann in einem akzeptierenden Zustand, wenn y ein Suffix von x ist.
- Es muss nicht zu jedem Zustand und jedem Buchstaben eine ausgehende Kante geben.

Wird der Suffixautomat für das reverse Pattern P^{rev} zu P konstruiert, so erlaubt die zweite Eigenschaft das Erkennen Präfixen von P.

Der Suffixautomat



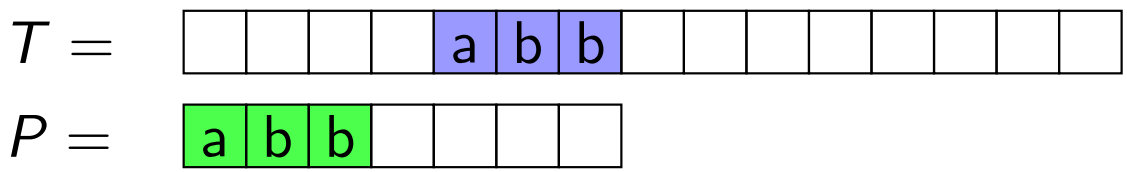
Beispiel-Suffixautomat für das reverse Pattern $P^{\text{rev}} = \texttt{acababa}$.

Eigenschaften:

- Es gibt im Startzustand keine Σ-Schleife.
- Zu Beginn sind alle Zustände aktiv.
- Sobald die aktive Zustandsmenge leer wird, wird das Fenster nicht mehr weiter bearbeitet.

Backward Nondeterministic DAWG Matching (BNDM)

- Implementierung eines *directed acyclic word graph* (DAWG).
- Bitparallele Realisierung, ähnlich wie beim Shift-And.
- Die aktive Zustandsmenge A wird durch Links-Schieben der Bits und Ver*und*en mit Masken aktualisiert.
- Die Masken werden exakt, wie beim Shift-And erstellt.
- Wenn nach j gelesenen Zeichen der akzeptierende Zustand aktiv ist, wurde ein Präfix $P[:j]$ gefunden.



Backward Nondeterministic DAWG Matching (BNDM)

- Wenn ein akzeptierender Zustand erreicht ist und
  - das komplette Fenster betrachtet wurde, wird dies gemeldet (match).
  - noch nicht das komplette Fenster betrachtet wurde, wird die Position des Fensters in `lastsuffix` gespeichert.
- Anschließend wird das Pattern um $m - |P[:j]|$ die Länge des größten echten Präfixes verschoben, welche in `lastsuffix` gespeichert ist.

Der BNDM-Algorithmus

```
1  def bndm(P, T):
2      S, m, n = set(P + T), len(P), len(T)
3      A_0, last = (1 << m) - 1, m
4      accept = 1 << (m - 1)
5      mask = create_masks(P[::-1], S)  #rev. pattern
6      while last <= n:
7          A, j, lastsuffix = A_0, 1, 0
8          while A:
9              A &= mask[T[last - j]]
10             if A & accept:
11                 if j == m:
12                     yield last - m, last; break
13                 else: lastsuffix = j
14             j, A = j + 1, A << 1
15         last += m - lastsuffix
```

Der BNDM-Algorithmus

Mustersuche mit dem BNDM-Algorithmus:

```
1     while last <= n:
2         A, j, lastsuffix = A_0, 1, 0
3         while A:
4             A &= mask[T[last - j]]
5             if A & accept:
6                 if j == m:
7                     yield last - m, last; break
8                 else: lastsuffix = j
9             j, A = j + 1, A << 1
10        last += m - lastsuffix
```

$T =$ | a | b | c | a | b | a | b | a | c | a | b | c |

$P =$ | a | b | a | b | a | c | a |

mask:

```
a: 1010101
b: 0101000
c: 0000010
```

$last = 7$ $\quad j = \text{-}$ $\quad A = {}^{-}$ $\quad lastsuffix = \text{-}$

Der BNDM-Algorithmus

Mustersuche mit dem BNDM-Algorithmus:

```
1     while last <= n:
2         A, j, lastsuffix = A_0, 1, 0
3         while A:
4             A &= mask[T[last - j]]
5             if A & accept:
6                 if j == m:
7                     yield last - m, last; break
8                 else: lastsuffix = j
9             j, A = j + 1, A << 1
10        last += m - lastsuffix
```

$T =$ | a | b | c | a | b | a | b | a | c | a | b | c |

$P =$ | a | b | a | b | a | c | a |

mask:

```
a: 1010101
b: 0101000
c: 0000010
```

$last = 7$ $\quad j = -$ $\quad A = {}^{-}$ $\quad lastsuffix = -$

Der BNDM-Algorithmus

Mustersuche mit dem BNDM-Algorithmus:

```
1     while last <= n:
2         A, j, lastsuffix = A_0, 1, 0
3         while A:
4             A &= mask[T[last - j]]
5             if A & accept:
6                 if j == m:
7                     yield last - m, last; break
8                 else: lastsuffix = j
9             j, A = j + 1, A << 1
10        last += m - lastsuffix
```

$T =$ | a | b | c | a | b | a | b | a | c | a | b | c |

$P =$ | a | b | a | b | a | c | a |

mask:

```
a: 1010101
b: 0101000
c: 0000010
```

$last = 7 \quad j = 1 \quad A = 1111111 \quad lastsuffix = 0$

Der BNDM-Algorithmus

Mustersuche mit dem BNDM-Algorithmus:

```
1     while last <= n:
2         A, j, lastsuffix = A_0, 1, 0
3         while A:
4             A &= mask[T[last - j]]
5             if A & accept:
6                 if j == m:
7                     yield last - m, last; break
8                 else: lastsuffix = j
9             j, A = j + 1, A << 1
10        last += m - lastsuffix
```

$T =$ | a | b | c | a | b | a | b | a | c | a | b | c |

$P =$ | a | b | a | b | a | c | a |

mask:

```
a: 1010101
b: 0101000
c: 0000010
```

$last = 7 \quad j = 1 \quad A = 1111111 \quad lastsuffix = 0$

Der BNDM-Algorithmus

Mustersuche mit dem BNDM-Algorithmus:

```
1     while last <= n:
2         A, j, lastsuffix = A_0, 1, 0
3         while A:
4             A &= mask[T[last - j]]
5             if A & accept:
6                 if j == m:
7                     yield last - m, last; break
8                 else: lastsuffix = j
9             j, A = j + 1, A << 1
10        last += m - lastsuffix
```

$T =$ | a | b | c | a | b | a | b | a | c | a | b | c |

$P =$ | a | b | a | b | a | c | a |

mask:

```
a: 1010101
b: 0101000
c: 0000010
```

$last = 7 \quad j = 1 \quad A = 0101000 \quad lastsuffix = 0$

Der BNDM-Algorithmus

Mustersuche mit dem BNDM-Algorithmus:

```
1     while last <= n:
2         A, j, lastsuffix = A_0, 1, 0
3         while A:
4             A &= mask[T[last - j]]
5             if A & accept:
6                 if j == m:
7                     yield last - m, last; break
8                 else: lastsuffix = j
9             j, A = j + 1, A << 1
10        last += m - lastsuffix
```

$T =$ | a | b | c | a | b | a | b | a | c | a | b | c |

$P =$ | a | b | a | b | a | c | a |

mask:

```
a: 1010101
b: 0101000
c: 0000010
```

$last = 7 \quad j = 1 \quad A = 0101000 \quad lastsuffix = 0$

Der BNDM-Algorithmus

Mustersuche mit dem BNDM-Algorithmus:

```
1     while last <= n:
2         A, j, lastsuffix = A_0, 1, 0
3         while A:
4             A &= mask[T[last - j]]
5             if A & accept:
6                 if j == m:
7                     yield last - m, last; break
8                 else: lastsuffix = j
9             j, A = j + 1, A << 1
10        last += m - lastsuffix
```

$T =$ | a | b | c | a | b | a | b | a | c | a | b | c |

$P =$ | a | b | a | b | a | c | a |

mask:

```
a: 1010101
b: 0101000
c: 0000010
```

$last = 7 \quad j = 2 \quad A =$ 1010000 $\quad lastsuffix = 0$

Der BNDM-Algorithmus

Mustersuche mit dem BNDM-Algorithmus:

```
1     while last <= n:
2         A, j, lastsuffix = A_0, 1, 0
3         while A:
4             A &= mask[T[last - j]]
5             if A & accept:
6                 if j == m:
7                     yield last - m, last; break
8                 else: lastsuffix = j
9             j, A = j + 1, A << 1
10        last += m - lastsuffix
```

$T =$ | a | b | c | a | b | a | b | a | c | a | b | c |

$P =$ | a | b | a | b | a | c | a |

mask:

```
a: 1010101
b: 0101000
c: 0000010
```

$last = 7 \quad j = 2 \quad A = 1010000 \quad lastsuffix = 0$

Der BNDM-Algorithmus

Mustersuche mit dem BNDM-Algorithmus:

```
1     while last <= n:
2         A, j, lastsuffix = A_0, 1, 0
3         while A:
4             A &= mask[T[last - j]]
5             if A & accept:
6                 if j == m:
7                     yield last - m, last; break
8                 else: lastsuffix = j
9             j, A = j + 1, A << 1
10        last += m - lastsuffix
```

$T =$ | a | b | c | a | b | a | b | a | c | a | b | c |

$P =$ | a | b | a | b | a | c | a |

mask:

```
a: 1010101
b: 0101000
c: 0000010
```

$last = 7 \quad j = 2 \quad A = 1010000 \quad lastsuffix = 0$

Der BNDM-Algorithmus

Mustersuche mit dem BNDM-Algorithmus:

```
1     while last <= n:
2         A, j, lastsuffix = A_0, 1, 0
3         while A:
4             A &= mask[T[last - j]]
5             if A & accept:
6                 if j == m:
7                     yield last - m, last; break
8                 else: lastsuffix = j
9             j, A = j + 1, A << 1
10        last += m - lastsuffix
```

$T =$ | a | b | c | a | b | a | b | a | c | a | b | c |

$P =$ | a | b | a | b | a | c | a |

mask:

```
a: 1010101
b: 0101000
c: 0000010
```

$last = 7 \quad j = 2 \quad A = 1010000 \quad lastsuffix = 0$

Der BNDM-Algorithmus

Mustersuche mit dem BNDM-Algorithmus:

```
1     while last <= n:
2         A, j, lastsuffix = A_0, 1, 0
3         while A:
4             A &= mask[T[last - j]]
5             if A & accept:
6                 if j == m:
7                     yield last - m, last; break
8                 else: lastsuffix = j
9             j, A = j + 1, A << 1
10        last += m - lastsuffix
```

$T =$ | a | b | c | a | b | a | b | a | c | a | b | c |

$P =$ | a | b | a | b | a | c | a |

mask:

```
a: 1010101
b: 0101000
c: 0000010
```

$last = 7 \quad j = 2 \quad A = 1010000 \quad lastsuffix = 0$

Der BNDM-Algorithmus

Mustersuche mit dem BNDM-Algorithmus:

```
1     while last <= n:
2         A, j, lastsuffix = A_0, 1, 0
3         while A:
4             A &= mask[T[last - j]]
5             if A & accept:
6                 if j == m:
7                     yield last - m, last; break
8                 else: lastsuffix = j
9             j, A = j + 1, A << 1
10        last += m - lastsuffix
```

$T =$ | a | b | c | a | b | a | b | a | c | a | b | c |

$P =$ | a | b | a | b | a | c | a |

mask:

```
a: 1010101
b: 0101000
c: 0000010
```

$last = 7 \quad j = 2 \quad A =$ 1010000 $\quad lastsuffix = 2$

Der BNDM-Algorithmus

Mustersuche mit dem BNDM-Algorithmus:

```
1     while last <= n:
2         A, j, lastsuffix = A_0, 1, 0
3         while A:
4             A &= mask[T[last - j]]
5             if A & accept:
6                 if j == m:
7                     yield last - m, last; break
8                 else: lastsuffix = j
9             j, A = j + 1, A << 1
10        last += m - lastsuffix
```

$T =$ | a | b | c | a | b | a | b | a | c | a | b | c |

$P =$ | a | b | a | b | a | c | a |

mask:

```
a: 1010101
b: 0101000
c: 0000010
```

$last = 7 \quad j = 3 \quad A = 0100000 \quad lastsuffix = 2$

Der BNDM-Algorithmus

Mustersuche mit dem BNDM-Algorithmus:

```
1     while last <= n:
2         A, j, lastsuffix = A_0, 1, 0
3         while A:
4             A &= mask[T[last - j]]
5             if A & accept:
6                 if j == m:
7                     yield last - m, last; break
8                 else: lastsuffix = j
9             j, A = j + 1, A << 1
10        last += m - lastsuffix
```

$T =$ | a | b | c | a | b | a | b | a | c | a | b | c |

$P =$ | a | b | a | b | a | c | a |

mask:

```
a: 1010101
b: 0101000
c: 0000010
```

$last = 7 \quad j = 3 \quad A = 0100000 \quad lastsuffix = 2$

Der BNDM-Algorithmus

Mustersuche mit dem BNDM-Algorithmus:

```
1     while last <= n:
2         A, j, lastsuffix = A_0, 1, 0
3         while A:
4             A &= mask[T[last - j]]
5             if A & accept:
6                 if j == m:
7                     yield last - m, last; break
8                 else: lastsuffix = j
9             j, A = j + 1, A << 1
10        last += m - lastsuffix
```

$T =$ | a | b | c | a | b | a | b | a | c | a | b | c |

$P =$ | a | b | a | b | a | c | a |

mask:

```
a: 1010101
b: 0101000
c: 0000010
```

$last = 7 \quad j = 3 \quad A = 0100000 \quad lastsuffix = 2$

Der BNDM-Algorithmus

Mustersuche mit dem BNDM-Algorithmus:

```
1     while last <= n:
2         A, j, lastsuffix = A_0, 1, 0
3         while A:
4             A &= mask[T[last - j]]
5             if A & accept:
6                 if j == m:
7                     yield last - m, last; break
8                 else: lastsuffix = j
9             j, A = j + 1, A << 1
10        last += m - lastsuffix
```

$T =$ | a | b | c | a | b | a | b | a | c | a | b | c |

$P =$ | a | b | a | b | a | c | a |

mask:

```
a: 1010101
b: 0101000
c: 0000010
```

$last = 7 \quad j = 3 \quad A = 0100000 \quad lastsuffix = 2$

Der BNDM-Algorithmus

Mustersuche mit dem BNDM-Algorithmus:

```
1     while last <= n:
2         A, j, lastsuffix = A_0, 1, 0
3         while A:
4             A &= mask[T[last - j]]
5             if A & accept:
6                 if j == m:
7                     yield last - m, last; break
8                 else: lastsuffix = j
9             j, A = j + 1, A << 1
10        last += m - lastsuffix
```

$T =$ | a | b | c | a | b | a | b | a | c | a | b | c |

$P =$ | a | b | a | b | a | c | a |

mask:

```
a: 1010101
b: 0101000
c: 0000010
```

$last = 7 \quad j = 4 \quad A = 1000000 \quad lastsuffix = 2$

Der BNDM-Algorithmus

Mustersuche mit dem BNDM-Algorithmus:

```
1     while last <= n:
2         A, j, lastsuffix = A_0, 1, 0
3         while A:
4             A &= mask[T[last - j]]
5             if A & accept:
6                 if j == m:
7                     yield last - m, last; break
8                 else: lastsuffix = j
9             j, A = j + 1, A << 1
10        last += m - lastsuffix
```

$T =$ | a | b | c | a | b | a | b | a | c | a | b | c |

$P =$ | a | b | a | b | a | c | a |

mask:

```
a: 1010101
b: 0101000
c: 0000010
```

$last = 7 \quad j = 4 \quad A = 1000000 \quad lastsuffix = 2$

Der BNDM-Algorithmus

Mustersuche mit dem BNDM-Algorithmus:

```
1     while last <= n:
2         A, j, lastsuffix = A_0, 1, 0
3         while A:
4             A &= mask[T[last - j]]
5             if A & accept:
6                 if j == m:
7                     yield last - m, last; break
8                 else: lastsuffix = j
9             j, A = j + 1, A << 1
10        last += m - lastsuffix
```

$T =$ | a | b | c | a | b | a | b | a | c | a | b | c |

$P =$ | a | b | a | b | a | c | a |

mask:

```
a: 1010101
b: 0101000
c: 0000010
```

$last = 7 \quad j = 4 \quad A = 1000000 \quad lastsuffix = 2$

Der BNDM-Algorithmus

Mustersuche mit dem BNDM-Algorithmus:

```
1     while last <= n:
2         A, j, lastsuffix = A_0, 1, 0
3         while A:
4             A &= mask[T[last - j]]
5             if A & accept:
6                 if j == m:
7                     yield last - m, last; break
8                 else: lastsuffix = j
9             j, A = j + 1, A << 1
10        last += m - lastsuffix
```

$T =$ | a | b | c | a | b | a | b | a | c | a | b | c |

$P =$ | a | b | a | b | a | c | a |

mask:

```
a: 1010101
b: 0101000
c: 0000010
```

$last = 7 \quad j = 4 \quad A = 1000000 \quad lastsuffix = 2$

Der BNDM-Algorithmus

Mustersuche mit dem BNDM-Algorithmus:

```
1     while last <= n:
2         A, j, lastsuffix = A_0, 1, 0
3         while A:
4             A &= mask[T[last - j]]
5             if A & accept:
6                 if j == m:
7                     yield last - m, last; break
8                 else: lastsuffix = j
9             j, A = j + 1, A << 1
10        last += m - lastsuffix
```

$T =$ | a | b | c | a | b | a | b | a | c | a | b | c |

$P =$ | a | b | a | b | a | c | a |

mask:

```
a: 1010101
b: 0101000
c: 0000010
```

$last = 7 \quad j = 4 \quad A = 1000000 \quad lastsuffix = 2$

Der BNDM-Algorithmus

Mustersuche mit dem BNDM-Algorithmus:

```
1     while last <= n:
2         A, j, lastsuffix = A_0, 1, 0
3         while A:
4             A &= mask[T[last - j]]
5             if A & accept:
6                 if j == m:
7                     yield last - m, last; break
8                 else: lastsuffix = j
9             j, A = j + 1, A << 1
10        last += m - lastsuffix
```

$T =$ | a | b | c | a | b | a | b | a | c | a | b | c |

$P =$ | a | b | a | b | a | c | a |

mask:

```
a: 1010101
b: 0101000
c: 0000010
```

$last = 7 \quad j = 4 \quad A = 1000000 \quad lastsuffix = 4$

Der BNDM-Algorithmus

Mustersuche mit dem BNDM-Algorithmus:

```
1     while last <= n:
2         A, j, lastsuffix = A_0, 1, 0
3         while A:
4             A &= mask[T[last - j]]
5             if A & accept:
6                 if j == m:
7                     yield last - m, last; break
8                 else: lastsuffix = j
9             j, A = j + 1, A << 1
10        last += m - lastsuffix
```

$T =$ | a | b | c | a | b | a | b | a | c | a | b | c |

$P =$ | a | b | a | b | a | c | a |

mask:

```
a: 1010101
b: 0101000
c: 0000010
```

$last = 7 \quad j = 5 \quad A = 0000000 \quad lastsuffix = 4$

Der BNDM-Algorithmus

Mustersuche mit dem BNDM-Algorithmus:

```
1     while last <= n:
2         A, j, lastsuffix = A_0, 1, 0
3         while A:
4             A &= mask[T[last - j]]
5             if A & accept:
6                 if j == m:
7                     yield last - m, last; break
8                 else: lastsuffix = j
9             j, A = j + 1, A << 1
10        last += m - lastsuffix
```

$T =$ | a | b | c | a | b | a | b | a | c | a | b | c |

$P =$ | a | b | a | b | a | c | a |

mask:

```
a: 1010101
b: 0101000
c: 0000010
```

$last = 7 \quad j = 5 \quad A = 0000000 \quad lastsuffix = 4$

Der BNDM-Algorithmus

Mustersuche mit dem BNDM-Algorithmus:

```
1     while last <= n:
2         A, j, lastsuffix = A_0, 1, 0
3         while A:
4             A &= mask[T[last - j]]
5             if A & accept:
6                 if j == m:
7                     yield last - m, last; break
8                 else: lastsuffix = j
9             j, A = j + 1, A << 1
10        last += m - lastsuffix
```

$T =$ | a | b | c | a | b | a | b | a | c | a | b | c |

$P =$ | a | b | a | b | a | c | a |

mask:

```
a: 1010101
b: 0101000
c: 0000010
```

$last = 10 \quad j = 5 \quad A = 0000000 \quad lastsuffix = 4$

Der BNDM-Algorithmus

Mustersuche mit dem BNDM-Algorithmus:

```
1     while last <= n:
2         A, j, lastsuffix = A_0, 1, 0
3         while A:
4             A &= mask[T[last - j]]
5             if A & accept:
6                 if j == m:
7                     yield last - m, last; break
8                 else: lastsuffix = j
9             j, A = j + 1, A << 1
10        last += m - lastsuffix
```

$T =$ | a | b | c | a | b | a | b | a | c | a | b | c |

$P =$ | a | b | a | b | a | c | a |

mask:

```
a: 1010101
b: 0101000
c: 0000010
```

$last = 10 \quad j = 5 \quad A = 0000000 \quad lastsuffix = 4$

Der BNDM-Algorithmus

Mustersuche mit dem BNDM-Algorithmus:

```
1     while last <= n:
2         A, j, lastsuffix = A_0, 1, 0
3         while A:
4             A &= mask[T[last - j]]
5             if A & accept:
6                 if j == m:
7                     yield last - m, last; break
8                 else: lastsuffix = j
9             j, A = j + 1, A << 1
10        last += m - lastsuffix
```

$T =$ | a | b | c | a | b | a | b | a | c | a | b | c |

$P =$ | a | b | a | b | a | c | a |

mask:

```
a: 1010101
b: 0101000
c: 0000010
```

$last = 10 \quad j = 1 \quad A = 1111111 \quad lastsuffix = 0$

Der BNDM-Algorithmus

Mustersuche mit dem BNDM-Algorithmus:

```
1     while last <= n:
2         A, j, lastsuffix = A_0, 1, 0
3         while A:
4             A &= mask[T[last - j]]
5             if A & accept:
6                 if j == m:
7                     yield last - m, last; break
8                 else: lastsuffix = j
9             j, A = j + 1, A << 1
10        last += m - lastsuffix
```

$T =$ | a | b | c | a | b | a | b | a | c | a | b | c |

$P =$ | a | b | a | b | a | c | a |

mask:

```
a: 1010101
b: 0101000
c: 0000010
```

$last = 10 \quad j = 1 \quad A = 1111111 \quad lastsuffix = 0$

Der BNDM-Algorithmus

Mustersuche mit dem BNDM-Algorithmus:

```
1     while last <= n:
2         A, j, lastsuffix = A_0, 1, 0
3         while A:
4             A &= mask[T[last - j]]
5             if A & accept:
6                 if j == m:
7                     yield last - m, last; break
8                 else: lastsuffix = j
9             j, A = j + 1, A << 1
10        last += m - lastsuffix
```

$T =$ | a | b | c | a | b | a | b | a | c | a | b | c |

$P =$ | a | b | a | b | a | c | a |

mask:

```
a: 1010101
b: 0101000
c: 0000010
```

$last = 10 \quad j = 1 \quad A = 1010101 \quad lastsuffix = 0$

Der BNDM-Algorithmus

Mustersuche mit dem BNDM-Algorithmus:

```
1     while last <= n:
2         A, j, lastsuffix = A_0, 1, 0
3         while A:
4             A &= mask[T[last - j]]
5             if A & accept:
6                 if j == m:
7                     yield last - m, last; break
8                 else: lastsuffix = j
9             j, A = j + 1, A << 1
10        last += m - lastsuffix
```

$T =$ | a | b | c | a | b | a | b | a | c | a | b | c |

$P =$ | a | b | a | b | a | c | a |

mask:

```
a: 1010101
b: 0101000
c: 0000010
```

$last = 10 \quad j = 1 \quad A = 1010101 \quad lastsuffix = 0$

Der BNDM-Algorithmus

Mustersuche mit dem BNDM-Algorithmus:

```
1     while last <= n:
2         A, j, lastsuffix = A_0, 1, 0
3         while A:
4             A &= mask[T[last - j]]
5             if A & accept:
6                 if j == m:
7                     yield last - m, last; break
8                 else: lastsuffix = j
9             j, A = j + 1, A << 1
10        last += m - lastsuffix
```

$T =$ | a | b | c | a | b | a | b | a | c | a | b | c |

$P =$ | a | b | a | b | a | c | a |

mask:

```
a: 1010101
b: 0101000
c: 0000010
```

$last = 10 \quad j = 1 \quad A = 1010101 \quad lastsuffix = 0$

Der BNDM-Algorithmus

Mustersuche mit dem BNDM-Algorithmus:

```
1     while last <= n:
2         A, j, lastsuffix = A_0, 1, 0
3         while A:
4             A &= mask[T[last - j]]
5             if A & accept:
6                 if j == m:
7                     yield last - m, last; break
8                 else: lastsuffix = j
9             j, A = j + 1, A << 1
10        last += m - lastsuffix
```

$T =$ | a | b | c | a | b | a | b | a | c | a | b | c |

$P =$ | a | b | a | b | a | c | a |

mask:

```
a: 1010101
b: 0101000
c: 0000010
```

$last = 10 \quad j = 1 \quad A = 1010101 \quad lastsuffix = 1$

Der BNDM-Algorithmus

Mustersuche mit dem BNDM-Algorithmus:

```
1     while last <= n:
2         A, j, lastsuffix = A_0, 1, 0
3         while A:
4             A &= mask[T[last - j]]
5             if A & accept:
6                 if j == m:
7                     yield last - m, last; break
8                 else: lastsuffix = j
9             j, A = j + 1, A << 1
10        last += m - lastsuffix
```

$T =$ | a | b | c | a | b | a | b | a | c | a | b | c |

$P =$ | a | b | a | b | a | c | a |

mask:

```
a: 1010101
b: 0101000
c: 0000010
```

$last = 10 \quad j = 2 \quad A = 0101010 \quad lastsuffix = 1$

Der BNDM-Algorithmus

Mustersuche mit dem BNDM-Algorithmus:

```
1     while last <= n:
2         A, j, lastsuffix = A_0, 1, 0
3         while A:
4             A &= mask[T[last - j]]
5             if A & accept:
6                 if j == m:
7                     yield last - m, last; break
8                 else: lastsuffix = j
9             j, A = j + 1, A << 1
10        last += m - lastsuffix
```

$T =$ | a | b | c | a | b | a | b | a | c | a | b | c |

$P =$ | a | b | a | b | a | c | a |

mask:

```
a: 1010101
b: 0101000
c: 0000010
```

$last = 10 \quad j = 2 \quad A = 0101010 \quad lastsuffix = 1$

Der BNDM-Algorithmus

Mustersuche mit dem BNDM-Algorithmus:

```
1     while last <= n:
2         A, j, lastsuffix = A_0, 1, 0
3         while A:
4             A &= mask[T[last - j]]
5             if A & accept:
6                 if j == m:
7                     yield last - m, last; break
8                 else: lastsuffix = j
9             j, A = j + 1, A << 1
10        last += m - lastsuffix
```

$T =$ | a | b | c | a | b | a | b | a | c | a | b | c |

$P =$ | a | b | a | b | a | c | a |

mask:

```
a: 1010101
b: 0101000
c: 0000010
```

$last = 10 \quad j = 2 \quad A = 0000010 \quad lastsuffix = 1$

Der BNDM-Algorithmus

Mustersuche mit dem BNDM-Algorithmus:

```
1     while last <= n:
2         A, j, lastsuffix = A_0, 1, 0
3         while A:
4             A &= mask[T[last - j]]
5             if A & accept:
6                 if j == m:
7                     yield last - m, last; break
8                 else: lastsuffix = j
9             j, A = j + 1, A << 1
10        last += m - lastsuffix
```

$T =$ | a | b | c | a | b | a | b | a | c | a | b | c |

$P =$ | a | b | a | b | a | c | a |

mask:

```
a: 1010101
b: 0101000
c: 0000010
```

$last = 10 \quad j = 2 \quad A = 0000010 \quad lastsuffix = 1$

Der BNDM-Algorithmus

Mustersuche mit dem BNDM-Algorithmus:

```
1     while last <= n:
2         A, j, lastsuffix = A_0, 1, 0
3         while A:
4             A &= mask[T[last - j]]
5             if A & accept:
6                 if j == m:
7                     yield last - m, last; break
8                 else: lastsuffix = j
9             j, A = j + 1, A << 1
10        last += m - lastsuffix
```

$T =$ | a | b | c | a | b | a | b | a | c | a | b | c |

$P =$ | a | b | a | b | a | c | a |

mask:

```
a: 1010101
b: 0101000
c: 0000010
```

$last = 10 \quad j = 3 \quad A = 0000100 \quad lastsuffix = 1$

Der BNDM-Algorithmus

Mustersuche mit dem BNDM-Algorithmus:

```
1     while last <= n:
2         A, j, lastsuffix = A_0, 1, 0
3         while A:
4             A &= mask[T[last - j]]
5             if A & accept:
6                 if j == m:
7                     yield last - m, last; break
8                 else: lastsuffix = j
9             j, A = j + 1, A << 1
10        last += m - lastsuffix
```

$T =$ | a | b | c | a | b | a | b | a | c | a | b | c |

$P =$ | a | b | a | b | a | c | a |

mask:

```
a: 1010101
b: 0101000
c: 0000010
```

$last = 10 \quad j = 3 \quad A = 0000100 \quad lastsuffix = 1$

Der BNDM-Algorithmus

Mustersuche mit dem BNDM-Algorithmus:

```
1     while last <= n:
2         A, j, lastsuffix = A_0, 1, 0
3         while A:
4             A &= mask[T[last - j]]
5             if A & accept:
6                 if j == m:
7                     yield last - m, last; break
8                 else: lastsuffix = j
9             j, A = j + 1, A << 1
10        last += m - lastsuffix
```

$T =$ | a | b | c | a | b | a | b | a | c | a | b | c |

$P =$ | a | b | a | b | a | c | a |

mask:

```
a: 1010101
b: 0101000
c: 0000010
```

$last = 10 \quad j = 3 \quad A = 0000100 \quad lastsuffix = 1$

Der BNDM-Algorithmus

Mustersuche mit dem BNDM-Algorithmus:

```
1     while last <= n:
2         A, j, lastsuffix = A_0, 1, 0
3         while A:
4             A &= mask[T[last - j]]
5             if A & accept:
6                 if j == m:
7                     yield last - m, last; break
8                 else: lastsuffix = j
9             j, A = j + 1, A << 1
10        last += m - lastsuffix
```

$T =$ | a | b | c | a | b | a | b | a | c | a | b | c |

$P =$ | a | b | a | b | a | c | a |

mask:

```
a: 1010101
b: 0101000
c: 0000010
```

$last = 10 \quad j = 3 \quad A = 0000100 \quad lastsuffix = 1$

Der BNDM-Algorithmus

Mustersuche mit dem BNDM-Algorithmus:

```
1     while last <= n:
2         A, j, lastsuffix = A_0, 1, 0
3         while A:
4             A &= mask[T[last - j]]
5             if A & accept:
6                 if j == m:
7                     yield last - m, last; break
8                 else: lastsuffix = j
9             j, A = j + 1, A << 1
10        last += m - lastsuffix
```

$T =$ | a | b | c | a | b | a | b | a | c | a | b | c |

$P =$ | a | b | a | b | a | c | a |

mask:

```
a: 1010101
b: 0101000
c: 0000010
```

$last = 10 \quad j = 4 \quad A = 0001000 \quad lastsuffix = 1$

Der BNDM-Algorithmus

Mustersuche mit dem BNDM-Algorithmus:

```
1     while last <= n:
2         A, j, lastsuffix = A_0, 1, 0
3         while A:
4             A &= mask[T[last - j]]
5             if A & accept:
6                 if j == m:
7                     yield last - m, last; break
8                 else: lastsuffix = j
9             j, A = j + 1, A << 1
10        last += m - lastsuffix
```

$T =$ | a | b | c | a | b | a | b | a | c | a | b | c |

$P =$ | a | b | a | b | a | c | a |

mask:

```
a: 1010101
b: 0101000
c: 0000010
```

$last = 10 \quad j = 4 \quad A = 0001000 \quad lastsuffix = 1$

Der BNDM-Algorithmus

Mustersuche mit dem BNDM-Algorithmus:

```
1     while last <= n:
2         A, j, lastsuffix = A_0, 1, 0
3         while A:
4             A &= mask[T[last - j]]
5             if A & accept:
6                 if j == m:
7                     yield last - m, last; break
8                 else: lastsuffix = j
9             j, A = j + 1, A << 1
10        last += m - lastsuffix
```

$T =$ | a | b | c | a | b | a | b | a | c | a | b | c |

$P =$ | a | b | a | b | a | c | a |

mask:

```
a: 1010101
b: 0101000
c: 0000010
```

$last = 10 \quad j = 4 \quad A = 0001000 \quad lastsuffix = 1$

Der BNDM-Algorithmus

Mustersuche mit dem BNDM-Algorithmus:

```
1     while last <= n:
2         A, j, lastsuffix = A_0, 1, 0
3         while A:
4             A &= mask[T[last - j]]
5             if A & accept:
6                 if j == m:
7                     yield last - m, last; break
8                 else: lastsuffix = j
9             j, A = j + 1, A << 1
10        last += m - lastsuffix
```

$T =$ | a | b | c | a | b | a | b | a | c | a | b | c |

$P =$ | a | b | a | b | a | c | a |

mask:

```
a: 1010101
b: 0101000
c: 0000010
```

$last = 10 \quad j = 4 \quad A = 0001000 \quad lastsuffix = 1$

Der BNDM-Algorithmus

Mustersuche mit dem BNDM-Algorithmus:

```
1     while last <= n:
2         A, j, lastsuffix = A_0, 1, 0
3         while A:
4             A &= mask[T[last - j]]
5             if A & accept:
6                 if j == m:
7                     yield last - m, last; break
8                 else: lastsuffix = j
9             j, A = j + 1, A << 1
10        last += m - lastsuffix
```

$T =$ | a | b | c | a | b | a | b | a | c | a | b | c |

$P =$ | a | b | a | b | a | c | a |

mask:

```
a: 1010101
b: 0101000
c: 0000010
```

$last = 10 \quad j = 5 \quad A = 0010000 \quad lastsuffix = 1$

Der BNDM-Algorithmus

Mustersuche mit dem BNDM-Algorithmus:

```
1     while last <= n:
2         A, j, lastsuffix = A_0, 1, 0
3         while A:
4             A &= mask[T[last - j]]
5             if A & accept:
6                 if j == m:
7                     yield last - m, last; break
8                 else: lastsuffix = j
9             j, A = j + 1, A << 1
10        last += m - lastsuffix
```

$T =$ | a | b | c | a | b | a | b | a | c | a | b | c |

$P =$ | a | b | a | b | a | c | a |

mask:

```
a: 1010101
b: 0101000
c: 0000010
```

$last = 10 \quad j = 5 \quad A = 0010000 \quad lastsuffix = 1$

Der BNDM-Algorithmus

Mustersuche mit dem BNDM-Algorithmus:

```
1     while last <= n:
2         A, j, lastsuffix = A_0, 1, 0
3         while A:
4             A &= mask[T[last - j]]
5             if A & accept:
6                 if j == m:
7                     yield last - m, last; break
8                 else: lastsuffix = j
9             j, A = j + 1, A << 1
10        last += m - lastsuffix
```

$T =$ | a | b | c | a | b | a | b | a | c | a | b | c |

$P =$ | a | b | a | b | a | c | a |

mask:

```
a: 1010101
b: 0101000
c: 0000010
```

$last = 10 \quad j = 5 \quad A = 0010000 \quad lastsuffix = 1$

Der BNDM-Algorithmus

Mustersuche mit dem BNDM-Algorithmus:

```
1     while last <= n:
2         A, j, lastsuffix = A_0, 1, 0
3         while A:
4             A &= mask[T[last - j]]
5             if A & accept:
6                 if j == m:
7                     yield last - m, last; break
8                 else: lastsuffix = j
9             j, A = j + 1, A << 1
10        last += m - lastsuffix
```

$T =$ | a | b | c | a | b | a | b | a | c | a | b | c |

$P =$ | a | b | a | b | a | c | a |

mask:

```
a: 1010101
b: 0101000
c: 0000010
```

$last = 10 \quad j = 5 \quad A = 0010000 \quad lastsuffix = 1$

Der BNDM-Algorithmus

Mustersuche mit dem BNDM-Algorithmus:

```
1     while last <= n:
2         A, j, lastsuffix = A_0, 1, 0
3         while A:
4             A &= mask[T[last - j]]
5             if A & accept:
6                 if j == m:
7                     yield last - m, last; break
8                 else: lastsuffix = j
9             j, A = j + 1, A << 1
10        last += m - lastsuffix
```

$T =$ | a | b | c | a | b | a | b | a | c | a | b | c |

$P =$ | a | b | a | b | a | c | a |

mask:

```
a: 1010101
b: 0101000
c: 0000010
```

$last = 10 \quad j = 6 \quad A = 0100000 \quad lastsuffix = 1$

Der BNDM-Algorithmus

Mustersuche mit dem BNDM-Algorithmus:

```
1     while last <= n:
2         A, j, lastsuffix = A_0, 1, 0
3         while A:
4             A &= mask[T[last - j]]
5             if A & accept:
6                 if j == m:
7                     yield last - m, last; break
8                 else: lastsuffix = j
9             j, A = j + 1, A << 1
10        last += m - lastsuffix
```

$T =$ | a | b | c | a | b | a | b | a | c | a | b | c |

$P =$ | a | b | a | b | a | c | a |

mask:

```
a: 1010101
b: 0101000
c: 0000010
```

$last = 10$ $j = 6$ $A = 0100000$ $lastsuffix = 1$

Der BNDM-Algorithmus

Mustersuche mit dem BNDM-Algorithmus:

```
1     while last <= n:
2         A, j, lastsuffix = A_0, 1, 0
3         while A:
4             A &= mask[T[last - j]]
5             if A & accept:
6                 if j == m:
7                     yield last - m, last; break
8                 else: lastsuffix = j
9             j, A = j + 1, A << 1
10        last += m - lastsuffix
```

$T =$ | a | b | c | a | b | a | b | a | c | a | b | c |

$P =$ | a | b | a | b | a | c | a |

mask:

```
a: 1010101
b: 0101000
c: 0000010
```

$last = 10 \quad j = 6 \quad A = 0100000 \quad lastsuffix = 1$

Der BNDM-Algorithmus

Mustersuche mit dem BNDM-Algorithmus:

```
1     while last <= n:
2         A, j, lastsuffix = A_0, 1, 0
3         while A:
4             A &= mask[T[last - j]]
5             if A & accept:
6                 if j == m:
7                     yield last - m, last; break
8                 else: lastsuffix = j
9             j, A = j + 1, A << 1
10        last += m - lastsuffix
```

$T =$ | a | b | c | a | b | a | b | a | c | a | b | c |

$P =$ | a | b | a | b | a | c | a |

mask:

```
a: 1010101
b: 0101000
c: 0000010
```

$last = 10 \quad j = 6 \quad A = 0100000 \quad lastsuffix = 1$

Der BNDM-Algorithmus

Mustersuche mit dem BNDM-Algorithmus:

```
1     while last <= n:
2         A, j, lastsuffix = A_0, 1, 0
3         while A:
4             A &= mask[T[last - j]]
5             if A & accept:
6                 if j == m:
7                     yield last - m, last; break
8                 else: lastsuffix = j
9             j, A = j + 1, A << 1
10        last += m - lastsuffix
```

$T =$ | a | b | c | a | b | a | b | a | c | a | b | c |

$P =$ | a | b | a | b | a | c | a |

mask:

```
a: 1010101
b: 0101000
c: 0000010
```

$last = 10 \quad j = 7 \quad A = 1000000 \quad lastsuffix = 1$

Der BNDM-Algorithmus

Mustersuche mit dem BNDM-Algorithmus:

```
1     while last <= n:
2         A, j, lastsuffix = A_0, 1, 0
3         while A:
4             A &= mask[T[last - j]]
5             if A & accept:
6                 if j == m:
7                     yield last - m, last; break
8                 else: lastsuffix = j
9             j, A = j + 1, A << 1
10        last += m - lastsuffix
```

$T =$ | a | b | c | a | b | a | b | a | c | a | b | c |

$P =$ | a | b | a | b | a | c | a |

mask:

```
a: 1010101
b: 0101000
c: 0000010
```

$last = 10 \quad j = 7 \quad A = 1000000 \quad lastsuffix = 1$

Der BNDM-Algorithmus

Mustersuche mit dem BNDM-Algorithmus:

```
1     while last <= n:
2         A, j, lastsuffix = A_0, 1, 0
3         while A:
4             A &= mask[T[last - j]]
5             if A & accept:
6                 if j == m:
7                     yield last - m, last; break
8                 else: lastsuffix = j
9             j, A = j + 1, A << 1
10        last += m - lastsuffix
```

$T =$ | a | b | c | a | b | a | b | a | c | a | b | c |

$P =$ | a | b | a | b | a | c | a |

mask:

```
a: 1010101
b: 0101000
c: 0000010
```

$last = 10 \quad j = 7 \quad A = 1000000 \quad lastsuffix = 1$

Der BNDM-Algorithmus

Mustersuche mit dem BNDM-Algorithmus:

```
1     while last <= n:
2         A, j, lastsuffix = A_0, 1, 0
3         while A:
4             A &= mask[T[last - j]]
5             if A & accept:
6                 if j == m:
7                     yield last - m, last; break
8                 else: lastsuffix = j
9             j, A = j + 1, A << 1
10        last += m - lastsuffix
```

$T =$ | a | b | c | a | b | a | b | a | c | a | b | c |

$P =$ | a | b | a | b | a | c | a |

mask:

```
a: 1010101
b: 0101000
c: 0000010
```

$last = 10 \quad j = 7 \quad A = 1000000 \quad lastsuffix = 1$

Der BNDM-Algorithmus

Mustersuche mit dem BNDM-Algorithmus:

```
1     while last <= n:
2         A, j, lastsuffix = A_0, 1, 0
3         while A:
4             A &= mask[T[last - j]]
5             if A & accept:
6                 if j == m:
7                     yield last - m, last; break
8                 else: lastsuffix = j
9             j, A = j + 1, A << 1
10        last += m - lastsuffix
```

$T =$ | a | b | c | a | b | a | b | a | c | a | b | c |

$P =$ | a | b | a | b | a | c | a |

yield $3, 10$

mask:

```
a: 1010101
b: 0101000
c: 0000010
```

$last = 10$ $j = 7$ $A =$ 1000000 $lastsuffix = 1$

Der BNDM-Algorithmus

Mustersuche mit dem BNDM-Algorithmus:

```
1     while last <= n:
2         A, j, lastsuffix = A_0, 1, 0
3         while A:
4             A &= mask[T[last - j]]
5             if A & accept:
6                 if j == m:
7                     yield last - m, last; break
8                 else: lastsuffix = j
9             j, A = j + 1, A << 1
10        last += m - lastsuffix
```

$T =$ | a | b | c | a | b | a | b | a | c | a | b | c |

$P =$ | a | b | a | b | a | c | a |

mask:

```
a: 1010101
b: 0101000
c: 0000010
```

$last = 10 \quad j = 7 \quad A = 1000000 \quad lastsuffix = 1$

Der BNDM-Algorithmus

Mustersuche mit dem BNDM-Algorithmus:

```
1     while last <= n:
2         A, j, lastsuffix = A_0, 1, 0
3         while A:
4             A &= mask[T[last - j]]
5             if A & accept:
6                 if j == m:
7                     yield last - m, last; break
8                 else: lastsuffix = j
9             j, A = j + 1, A << 1
10        last += m - lastsuffix
```

$T =$ | a | b | c | a | b | a | b | a | c | a | b | c |

$P =$ | a | b | a | b | a | c | a |

mask:

a:	1010101
b:	0101000
c:	0000010

$last = 16 \quad j = 7 \quad A = 1000000 \quad lastsuffix = 1$

Der BNDM-Algorithmus

Mustersuche mit dem BNDM-Algorithmus:

```
1     while last <= n:
2         A, j, lastsuffix = A_0, 1, 0
3         while A:
4             A &= mask[T[last - j]]
5             if A & accept:
6                 if j == m:
7                     yield last - m, last; break
8                 else: lastsuffix = j
9             j, A = j + 1, A << 1
10        last += m - lastsuffix
```

$T =$ | a | b | c | a | b | a | b | a | c | a | b | c |

$P =$ | a | b | a | b | a | c | a |

mask:

a:	1010101
b:	0101000
c:	0000010

$last = 16 \quad j = 7 \quad A = 1000000 \quad lastsuffix = 1$

Zusammenfassung

- Der BNDM-Algorithmus nutzt einen Suffixautomaten um Präfixe im Pattern zu einem gegebenen Textausschnitt zu finden.
- Auch der BNDM ist nur effizient, wenn die Patternlänge kleiner als die Registergröße W ist.
- Die Best-case-Laufzeit beträgt $\mathcal{O}(m + n/m \cdot \lceil m/W \rceil)$, da das Pattern bis zu Patternlänge verschoben werden kann.
- Die Worst-case-Laufzeit beträgt $\mathcal{O}(m + nm \cdot \lceil m/W \rceil)$, wenn Text und Pattern aus demselben Buchstaben bestehen.

Alle PM-Algorithmen im Überblick

Algorithmus	*Best-case-Lz*	*Worst-case-Lz*	*Speicherpl.*
Naiv	$\mathcal{O}(n)$	$\mathcal{O}(nm)$	$\mathcal{O}(1)$
KMP	$\mathcal{O}(m+n)$		$\mathcal{O}(m)$
Shift-And	$\mathcal{O}(m+n\cdot r)$		$\mathcal{O}(\lvert\Sigma\rvert\cdot r)$
Horspool	$\mathcal{O}(m+n/m)$	$\mathcal{O}(m+nm)$	$\mathcal{O}(\lvert\Sigma\rvert)$
BNDM	$\mathcal{O}(m+n/m\cdot r)$	$\mathcal{O}(m+nm\cdot r)$	$\mathcal{O}(\lvert\Sigma\rvert\cdot r)$

$r = \lceil m/W \rceil$