DHBW
Duale Hochschule
Baden-Württemberg
Stuttgart
Campus Horb

Algorithmen und Datenstrukturen

Teil 13:
Suchen (IV) – Suchen in Texten

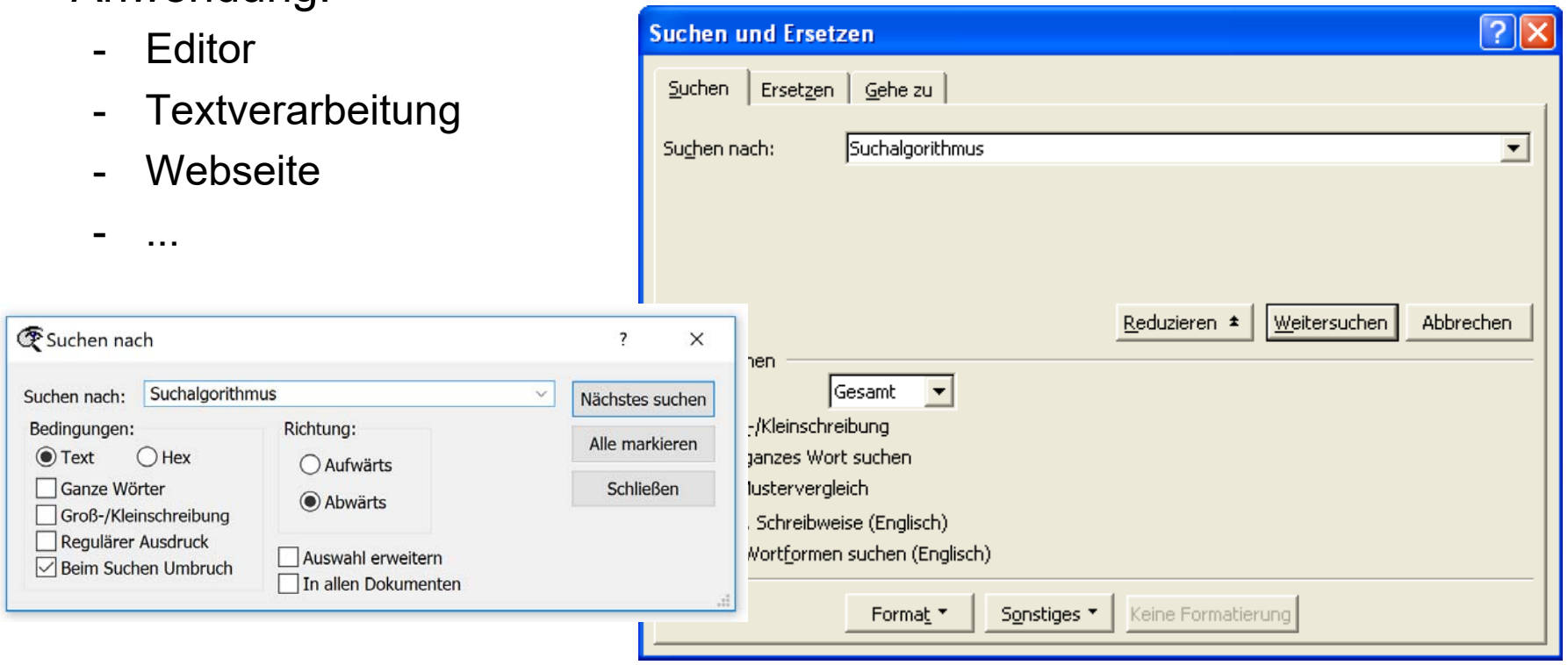
DHBW Stuttgart Campus Horb
Fakultät Technik
Studiengang Informatik
Dozent: Olaf Herden
Stand: 06/2020

Gliederung

- Motivation
- Brute Force Algorithmus
- Algorithmus von Knuth-Morris-Pratt
- Algorithmus von Boyer-Moore

Motivation

- Ziel: Suche bestimmte Zeichenkette (Pattern/Muster) in Text
- Anwendung:
  - Editor
  - Textverarbeitung
  - Webseite
  - ...



Motivation (II)

- Dynamische vs. statische Texte
  - Dynamische Texte (z.B. im Texteditor): Aufwändige Vorverarbeitung / Indizierung i.a. nicht sinnvoll
  - Relativ statische Texte: Erstellung von Indexstrukturen zur Suchbeschleunigung denkbar
- Suche nach beliebigen Strings/Zeichenketten vs. Worten/Begriffen
- Exakte Suche vs. approximative Suche (Ähnlichkeitssuche)

Definitionen (I)

Die Suche nach allen (exakten) Vorkommen eines Musters in einem Text ist ein Problem, das häufig auftritt (z.B. in Editoren, Datenbanken etc.). Wir vereinbaren folgende Konventionen (wobei Σ ein Alphabet ist):

$$
\begin{aligned}
\text{Muster } P &= P[0 \ldots m-1] \in \Sigma^m \\
\text{Text } T &= T[0 \ldots n-1] \in \Sigma^n \\
P[j] &= \text{Zeichen an der Position } j \\
P[b \ldots e] &= \text{Teilwort von } P \text{ zwischen den Positionen } b \text{ und } e
\end{aligned}
$$

Beispiel

$P = \mathrm{abcde}$, $P[0] = \mathrm{a}$, $P[3] = \mathrm{d}$, $P[2 \ldots 4] = \mathrm{cde}$.

[Steffen 2009]

Definitionen (I)

- leeres Wort: ε
- Länge eines Wortes x: $|x|$
- Konkatenation zweier Worte x und y: xy

Definition

Ein Wort w ist ein *Präfix* eines Wortes x ($w \sqsubset x$), falls $x = wy$ gilt für ein $y \in \Sigma^*$. w ist ein *Suffix* von x ($w \sqsupset x$), falls $x = yw$ gilt für ein $y \in \Sigma^*$. x ist also auch ein Präfix bzw. Suffix von sich selbst.

Lemma

Es seien x, y und z Zeichenketten, so dass $x \sqsupset z$ und $y \sqsupset z$ gilt. Wenn $|x| \leq |y|$ ist, dann gilt $x \sqsupset y$. Wenn $|x| \geq |y|$ ist, dann gilt $y \sqsupset x$. Wenn $|x| = |y|$ ist, dann gilt $x = y$.

[Steffen 2009]

Definitionen (III)

DHBW
Duale Hochschule
Baden-Württemberg
Stuttgart
Campus Horb

Definition

Ein Muster P kommt mit der Verschiebung s im Text T vor, falls $0 \leq s \leq n - m$ und $T[s \ldots s + m - 1] = P[0 \ldots m - 1]$ gilt. In dem Fall nennen wir s eine *gültige Verschiebung*.

Das Problem der exakten Textsuche ist nun, alle gültigen Verschiebungen zu finden. Mit den Bezeichnungen $S_0 = \varepsilon$ und $S_k = S[0 \ldots k - 1]$ (d.h. S_k ist das k-lange Präfix eines Wortes $S \in \Sigma^\ell$ für $0 \leq k \leq \ell$) können wir das Problem der exakten Textsuche folgendermaßen formulieren:

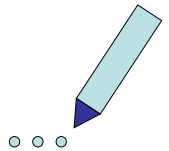
Definition

Finde alle Verschiebungen s, so dass $P \sqsupset T_{s+m}$.

[Steffen 2009]

Aufgabe

- Gegeben sei Σ = {a,b,c}, T = „abccabcabc“, P = „ab“
  (1) Geben sie alle Worte der maximalen Länge 3 über Σ an!
  (2) Geben Sie alle Präfixe und Suffixe des Wortes „abcac“ an!
  (3) Geben Sie T[0], T[7] und T[3...5] an!
  (4) Geben Sie alle gültigen Verschiebungen von P in T an!

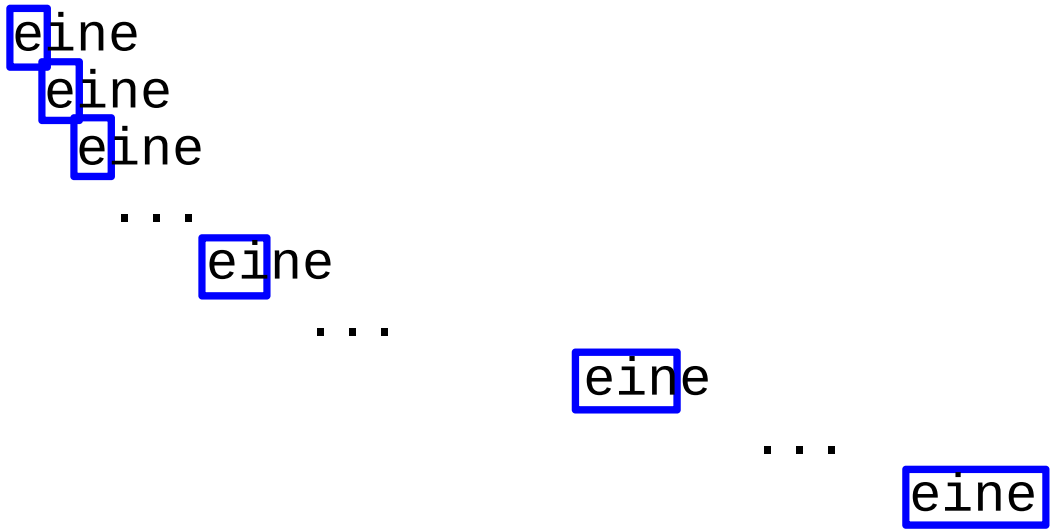


Übersicht

- Motivation
- Brute Force Algorithmus
- Algorithmus von Knuth-Morris-Pratt
- Algorithmus von Boyer-Moore

Idee

- Schiebe Pattern zeichenweise am Text entlang
- Vergleiche jeweils von Anfang an, bis das Ende des Pattern oder eine Unstimmigkeit (Mismatch) zwischen korrespondierenden Elementen des Pattern und des zu durchsuchenden Textes auftritt
- Beispiel: Suche nach „eine“



Brute Force-Algorithmus

- Pseudocode:

Beispiel (Naive-StringMatcher)

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

[Steffen 2009]

- Aufwand: O((n-m)*m) = O(n*m)
- Begründung:
  - (n-m) Durchgänge der äußeren Schleife
  - m Durchgänge der inneren Schleife

Bewertung

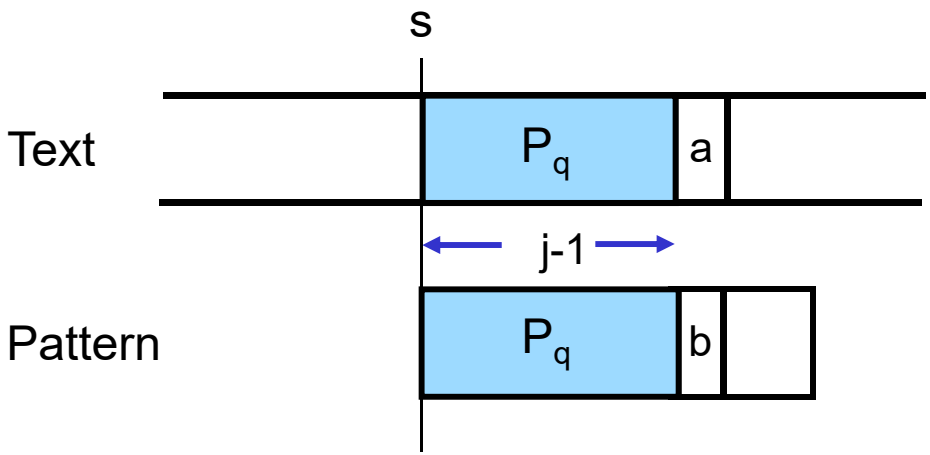
- Brute Force-Algorithmen sind (im worst case) zu ineffizient
- Ideen zur Verbesserung:
  - Nutzung der Musterstruktur, Kenntnis der im Muster vorkommenden Zeichen
  - Knuth-Morris-Pratt (1974): Nutze bereits geprüften Musteranfang um ggf. Muster um mehr als eine Stelle nach rechts zu verschieben
  - Boyer-Moore (1976): Teste Muster von hinten nach vorne
  - Horspool (1980): Optimierung Boyer-Moore-Algorithmus

Übersicht

- Motivation
- Brute Force Algorithmus
- Algorithmus von Knuth-Morris-Pratt
- Algorithmus von Boyer-Moore

Funktionsweise (I)

- Vergleiche Muster P von links nach rechts mit Text T
- Zur Vermeidung unnützer Vergleiche: Nutze Information über durchgeführte Vergleiche aus
- Sei P_q Präfix von P, tritt an Position s in T auf



- Frage: Wie weit darf verschoben werden?
- Abhängig von P_q, z.B.

```
Text:     DATENSTRUKTUREN        GEGEBENENFALLS
Muster    DATUM                  GEGENSATZ
```

- Unterscheidung verschiedener Fälle

Präfixfunktion π (I)

- Gibt an, wie weit Muster verschoben werden darf
- Gegeben Muster P[1 … m]
- Präfixfunktion $\pi : \{1,2,\ldots,m\} \rightarrow \{0,1,\ldots,m-1\}$
  $\pi(q) := max\{k : k < q \text{ und } P_k \sqsupset P_q\}$
- In Worten: π liefert Länge des längsten Präfix von P, das echtes Suffix von P_q ist
- Berechnung:

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

Präfixfunktion π (II): Beispiel (I)

- Berechnung π für Muster „ababaca“!

i	1	2	3	4	5	6	7
P[i]	a	b	a	b	a	c	a
π[i]	0	0	1				

- $q = 1 \Rightarrow \pi(1) = 0$ (Initialisierung)
- $q = 2$:
  ababaca
  ababaca

  Mismatch
  $\Rightarrow \pi(2) = 0$, Schiebe P um 1 weiter
- $q = 3$:
  ababaca
  ababaca

  Match
  $\Rightarrow \pi(3) = 1$, Erhöhe k auf 2

Präfixfunktion π (III): Beispiel (II)

i	1	2	3	4	5	6	7
P[i]	a	b	a	b	a	c	a
π[i]	0	0	1	2	3		

- $q = 4$:

  ababaca
  ababaca

  Match

  $\Rightarrow \pi(4) = 2$, Erhöhe k auf 3

- $q = 5$:

  ababaca
  ababaca

  Match

  $\Rightarrow \pi(5) = 3$, Erhöhe k auf 4

Präfixfunktion π (IV): Beispiel (III)

i	1	2	3	4	5	6	7
P[i]	a	b	a	b	a	c	a
π[i]	0	0	1	2	3	0	

- $q = 6$:

  abababca
  ababaca

  Mismatch, Lookup bei $\pi(3) = 1$
  $\Rightarrow$ Schiebe P um 2 weiter

  abababca
  abababca

  Mismatch, Lookup bei $\pi(1) = 0$
  $\Rightarrow$ Schiebe P um 1 weiter

  abababca
  ababaca

  Mismatch
  $\Rightarrow \pi(6) = 0$, Schiebe P um 1 weiter

Präfixfunktion π (V): Beispiel (IV)

i	1	2	3	4	5	6	7
P[i]	a	b	a	b	a	c	a
π[i]	0	0	1	2	3	0	1

- $q = 7$:

  ababaca

  abababaca

  Match

  $\Rightarrow \pi(7) = 1$, Erhöhe k auf 2

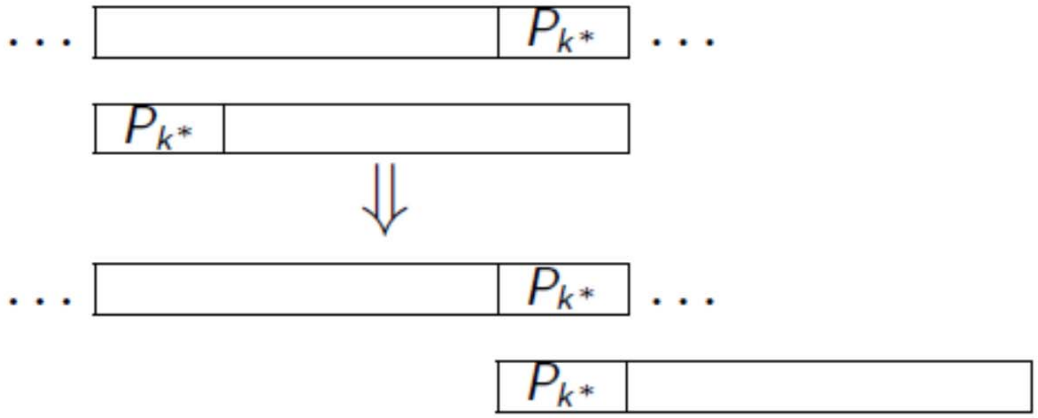
- Ende T erreicht $\Rightarrow$ Terminierung

Funktionsweise (II)

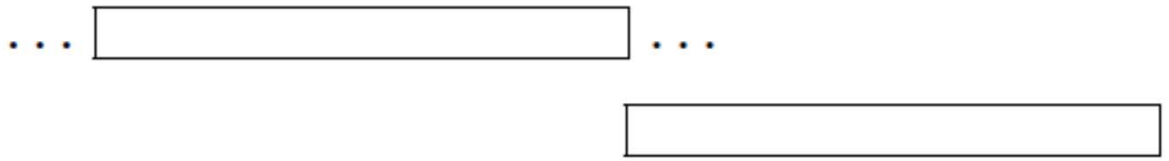
(A) $q = m$, d.h. s ist gültige Verschiebung:

1. Bestimme $k^* = \pi[m] = \max\{k : k < m \text{ und } P_k \sqsupset P_m\}$.

2.a) Falls $k^* > 0$, verschiebe das Muster nach rechts, so dass das Präfix P_{k^*} von P unter dem Suffix P_{k^*} von $T[s \ldots s + |P| - 1]$ liegt.



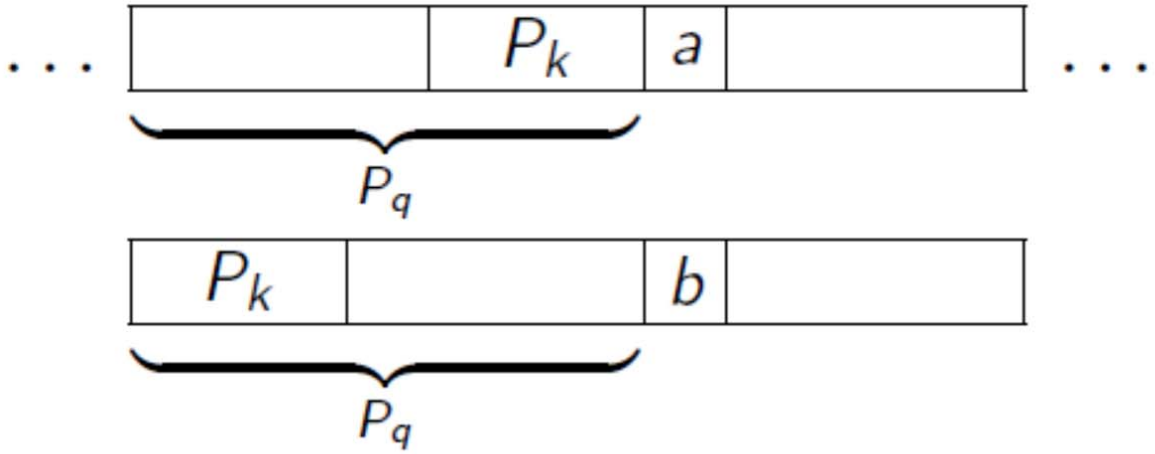
2.b) Falls $k^* = 0$, verschiebe das Muster P um die Länge $|P|$.



[Steffen 2009]

Funktionsweise (III)

(B) $q < m$:

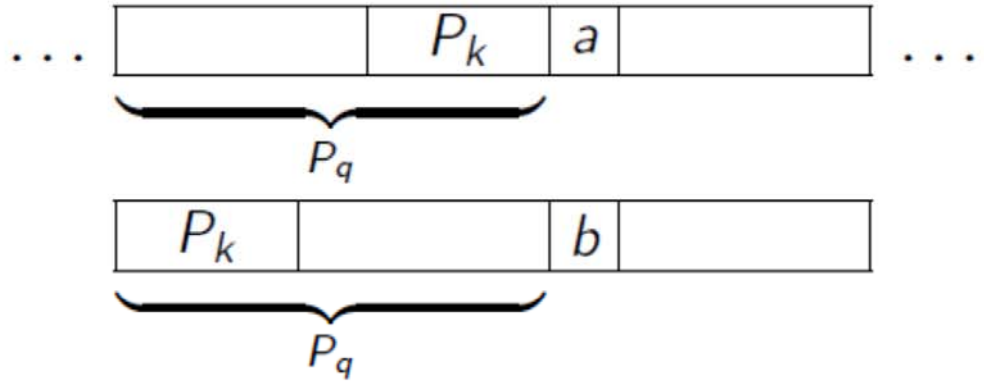


- Nun:
  - a) $a \neq b$ und P_q länger als ein Zeichen ($q > 0$)
  - b) $a \neq b$ und P_q ein Zeichen ($q = 0$)
  - c) $a = b$

[Steffen 2009]

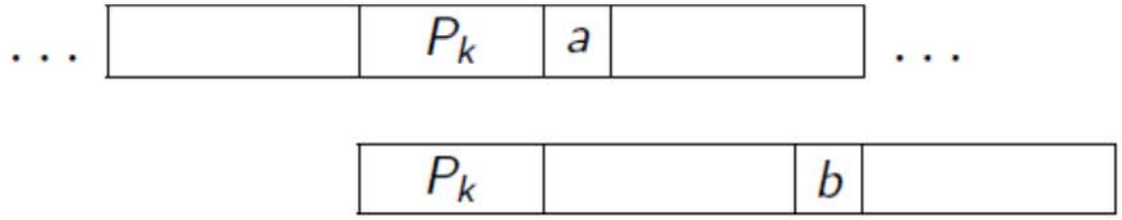
Funktionsweise (IV)

(B) $q < m$:



(a) $a \neq b$ und $q \neq 0$:

- bestimme $\pi[q] = \max\{k : k < q \text{ und } P_k \sqsupset P_q\}$, d.h. die Länge des längsten Präfixes von P, das auch Suffix von P_q ist;
- verschiebe P nach rechts, so dass das Suffix P_k von P_q über dem Präfix P_k von P liegt.

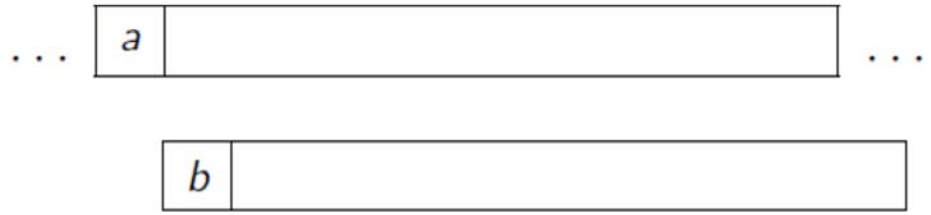


[Steffen 2009]

Funktionsweise (V)

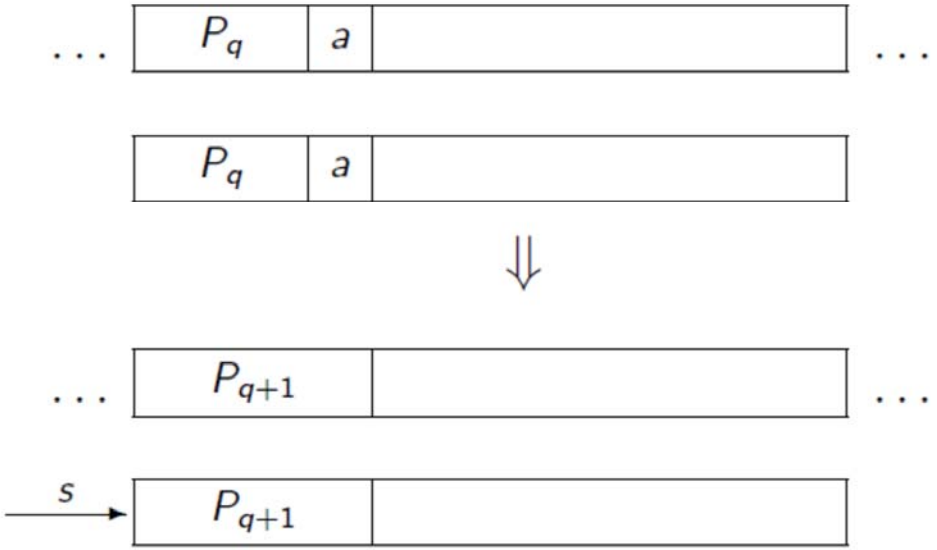
(b) $a \neq b$ und $q = 0$:

- verschiebe P um 1 nach rechts.



(c) $a = b$:

- mache weiter mit $q + 1$.



[Steffen 2009]

Algorithmus

- Ähnlich zu Präfixberechnung: Beide vergleichen Text T gegen Muster P
- Mustersuche: Vergleicht Text T gegen Muster P
- Präfixberechnung: Vergleicht Muster P gegen sich selbst (d.h. T = P)

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

Beispiel (I)

- Gegeben seien der Text „babababacababacaababababab“ und das Muster „ababaca“

```
bababababacababacaababababab
ababaca
```

  - Mismatch ⇒ Schiebe P um 1 weiter (Fall B.b)

```
babababacababacaababababab
 ababaca
```

  - Match ⇒ Setze q = 1 (Fall B.c)

```
babababacababacaababababab
 ababaca
```

  - Match ⇒ Setze q = 2 (Fall B.c)
  - …
  - Match ⇒ Setze q = 5 (Fall B.c)

```
babababacababacaababababab
 ababaca
```

Beispiel (II)

bababacababacaababababab
ababaca

- Mismatch $\Rightarrow$ Lookup $\pi(5) = 3$, Schiebe P um 2 weiter (Fall B.a)

bababacababacaababababab
ababaca

- Match $\Rightarrow$ Setze q = 1 (Fall B.c)
- …
- Match $\Rightarrow$ Setze q = 7 (Fall B.c)

bababacababacaababababab
ababaca

- Match und $q = m \Rightarrow$ Treffer an Position 4, Lookup $\pi(7) = 1$,
  Schiebe P um $m - \pi(7) = 6$ (Fall A.2a)

bababacababacaababababab
ababaca

- Match $\Rightarrow$ Setze q = 1 (Fall B.c)
- …
- Match $\Rightarrow$ Setze q = 7 (Fall B.c)

Beispiel (III)

bababababacabacaababababab
ababaca

- Match und $q = m \Rightarrow$ Treffer an Position 10, Lookup $\pi(7) = 1$,
  Schiebe P um $m - \pi(7) = 6$ (Fall A.2a)

bababababacabacaababababab
ababaca

- Match $\Rightarrow$ Setze $q = 1$ (Fall B.c)

bababababacabacaaababababab
ababaca

- Mismatch $\Rightarrow$ Lookup $\pi(1) = 0$, Schiebe P um 1 weiter (Fall B.a)

bababababacabacaaababababab
ababaca

- Match $\Rightarrow$ Setze q = 1 (Fall B.c)
- …
- Match $\Rightarrow$ Setze q = 5 (Fall B.c)

Beispiel (IV)

bababababacababacaababababab
ababaca

- Mismatch $\Rightarrow$ Lookup $\pi(5) = 3$, Schiebe P um 2 $(q - \pi(5))$ weiter (Fall B.a)

bababababacababacaababababab
ababaca

- Match $\Rightarrow$ Setze q = 1 (Fall B.c)
- …
- Match $\Rightarrow$ Setze q = 5 (Fall B.c)

bababababacababacaababababab
ababaca

- Mismatch $\Rightarrow$ Lookup $\pi(5) = 3$, Schiebe P um 2 $(q - \pi(5))$ weiter (Fall B.a)

bababababacababacaababababab
ababaca

- Match $\Rightarrow$ Setze q = 1 (Fall B.c)
- …
- Match $\Rightarrow$ Setze q = 4 (Fall B.c)

Beispiel (V)

bababababacababacaababababa
ababaca

- Ende des Textes $\Rightarrow$ Terminierung

Komplexität

- In extremen Fällen deutlich geringer als Brute Force Algorithmus
- Zwischen zwei Vergleichen wird entweder Pattern verschoben (kann höchstens (n-m)-mal geschehen) oder Vergleichsposition wandert nach rechts (höchstens (n-1)-mal)
- Daher: Maximale Zahl an Vergleichen: $n-m+n-1+1=2n-m$
- Da i.A. $n >> m$ ist dies O(n)
- Aufwand O(m) für Präfixfunktion
- Insgesamt also O(n+m)
- Da $n >> m$ angenommen werden

  $\Rightarrow$ Komplexität linear zur Länge des Textes, d.h. O(n)

Anmerkungen

- Weiterer Vorteil KMP-Algorithmus gegenüber Brute Force Ansatz:
  - Im Text wird nie zurückgegangen
  - D.h. Textzeiger wird zu keinem Zeitpunkt zurückgesetzt
  - Vorteilhaft bei Suche in sequentiellen Dateien

- Zusammenfassend: KMP-Algorithmus entschärft den ungünstigsten Fall erheblich

- Korrektheitsbeweis relativ aufwändig (nicht in dieser Vorlesung)

Übersicht

- Motivation
- Brute Force Algorithmus
- Algorithmus von Knuth-Morris-Pratt
- Algorithmus von Boyer-Moore

Idee

- Auswertung Muster von rechts nach links
- Ziel: Bei Mismatch Muster möglichst weit verschieben
- Nutze zur Verschiebung zwei Heuristiken:
  - Bad-Character
  - Good-Suffix

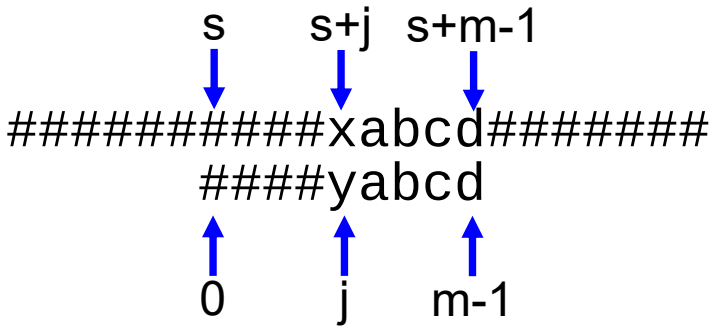
Bad-Character-Heuristik (I)

Falls beim Vergleich von $P[0 \ldots m-1]$ und $T[s \ldots s+m-1]$ (von rechts nach links) ein *Mismatch*

$$P[j] \neq T[s+j] \text{ für ein } j \text{ mit } 0 \leq j \leq m-1$$

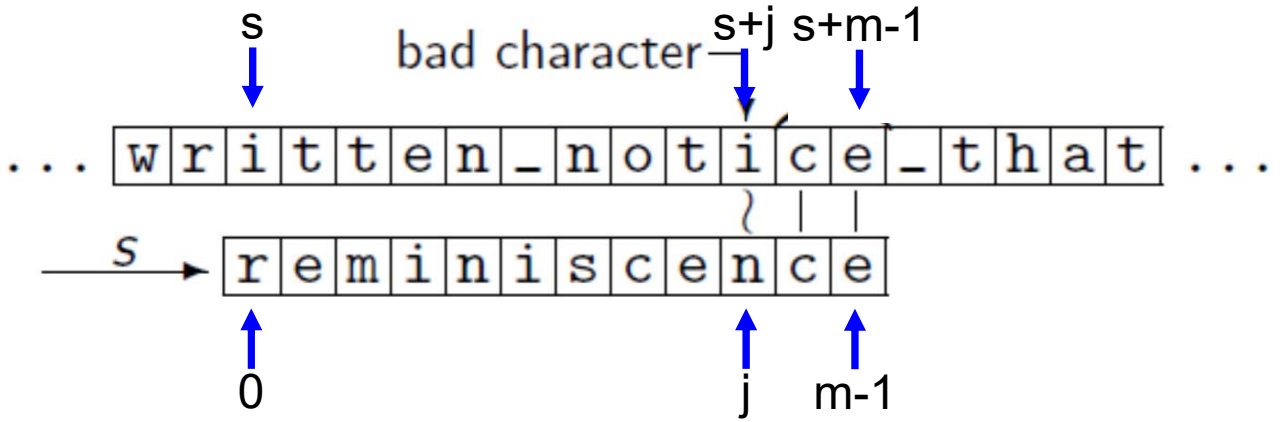
festgestellt wird, so schlägt die *bad-character* Heuristik eine Verschiebung des Musters um $j-k$ Positionen vor, wobei k der größte Index $(0 \leq k \leq m-1)$ ist mit $T[s+j] = P[k]$. Wenn kein k mit $T[s+j] = P[k]$ existiert, so sei $k = -1$. Man beachte, dass $j-k$ negativ sein kann.

[Steffen 2009]

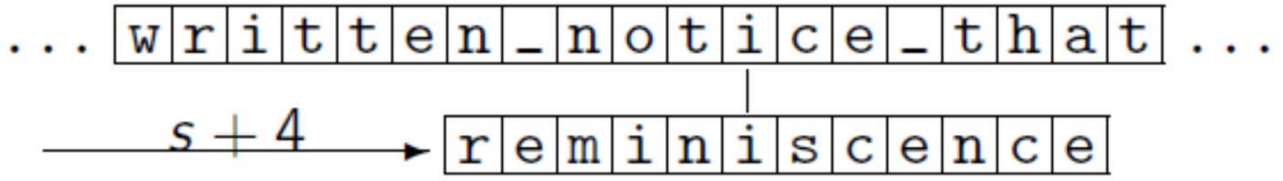


- Suche am weitesten rechts im Muster stehendes x (Position k)
- Subtrahiere j-k

Bad-Character-Heuristik (II): Beispiel



- k = Größter Index im Muster mit Bad Character „i“ = 5
- Verschiebe um j – k = 9 – 5 = 4



[Steffen 2009]

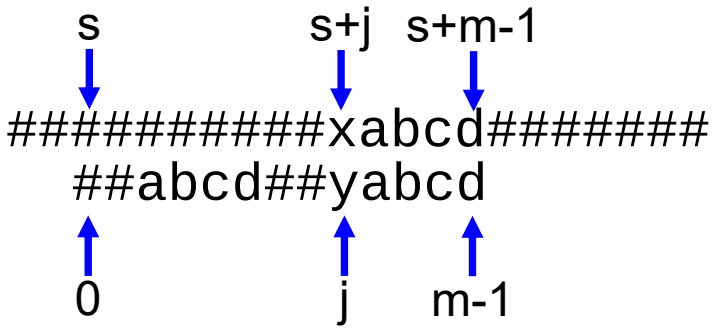
Good-Suffix-Heuristik (I)

Falls beim Vergleich von $P[0 \ldots m-1]$ und $T[s \ldots s+m-1]$ (von rechts nach links) ein *Mismatch*

$$P[j] \neq T[s+j] \text{ für ein } j \text{ mit } 0 \leq j \leq m-1$$

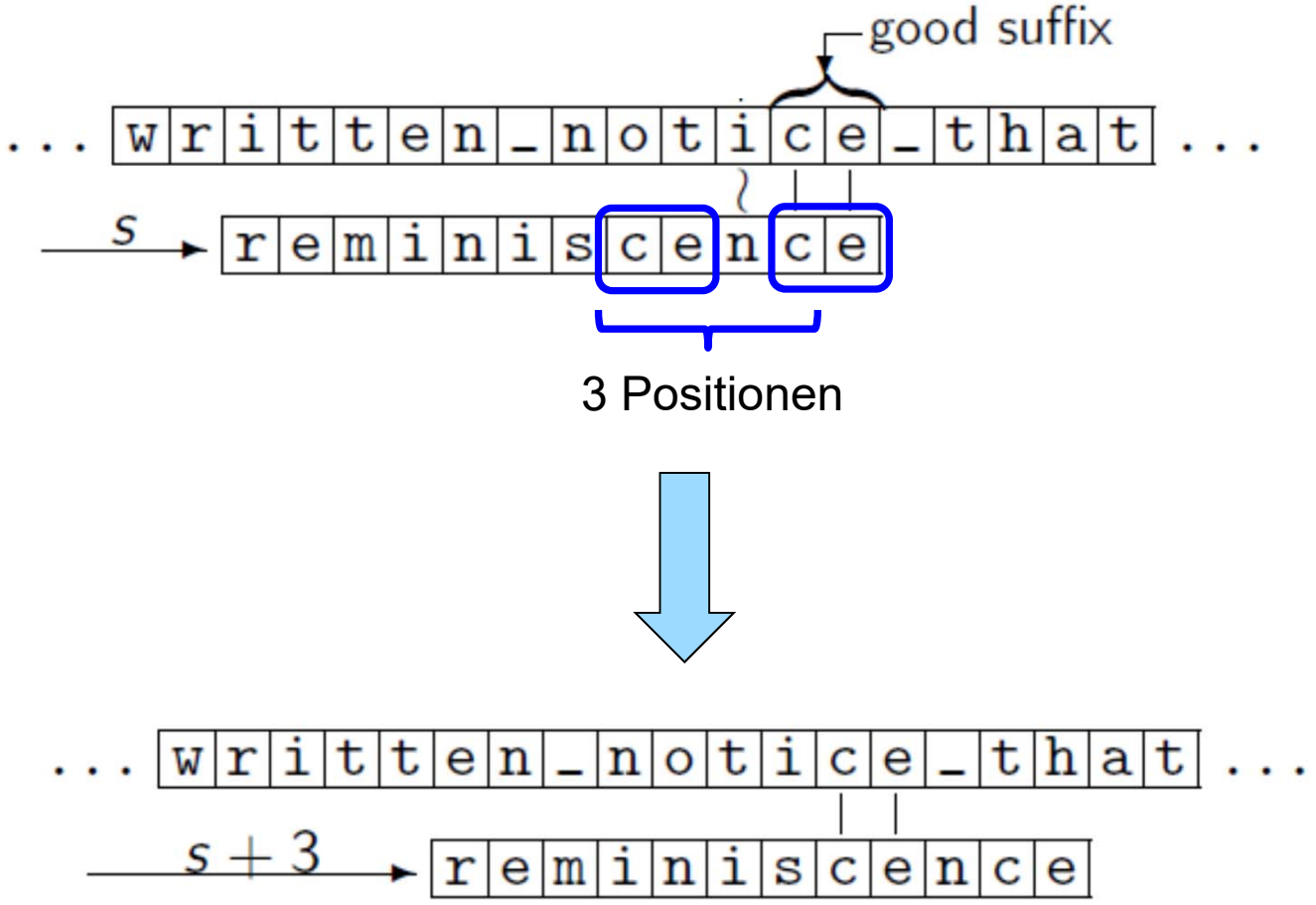
festgestellt wird, so wird das Muster so weit nach rechts geschoben, bis das bekannte Suffix $T[s+j+1 \ldots s+m-1]$ wieder auf ein Teilwort des Musters passt. Hierfür ist eine Vorverarbeitung des Musters notwendig, auf die wir hier aber nicht näher eingehen wollen.

[Steffen 2009]



- Verschiebe Muster, bis nächstes Good Suffix in Muster passt

Good-Suffix-Heuristik (II): Beispiel



[Steffen 2009]

Optimierung (I)

- Vorgestellte Variante Originalarbeit
- Wesentlicher Performancetreiber: Bad-Character-Heuristik
- Verbesserung (Horspool 1980):
  - Modifiziere Bad-Character-Heuristik
  - Soll immer positive Verschiebung vorschlagen
  - Damit:
    - Good-Suffix-Heuristik überflüssig
    - Vorverarbeitung einfacher
- Verbesserter Algorithmus: Boyer-Moore-Horspool-Algorithmus (BMH)

Optimierung (II)

- falls $P[j] \neq T[s+j]$ für ein j $(0 \leq j \leq m-1)$

$$s \leftarrow s + m - 1 - k$$

wobei k größter Index zwischen 0 und $m-2$
mit $T[s+m-1] = P[k]$

- wenn kein k $(0 \leq k \leq m-2)$ mit $T[s+m-1] = P[k]$ existiert

$$s \leftarrow s + m$$

- Verschiebespanne:

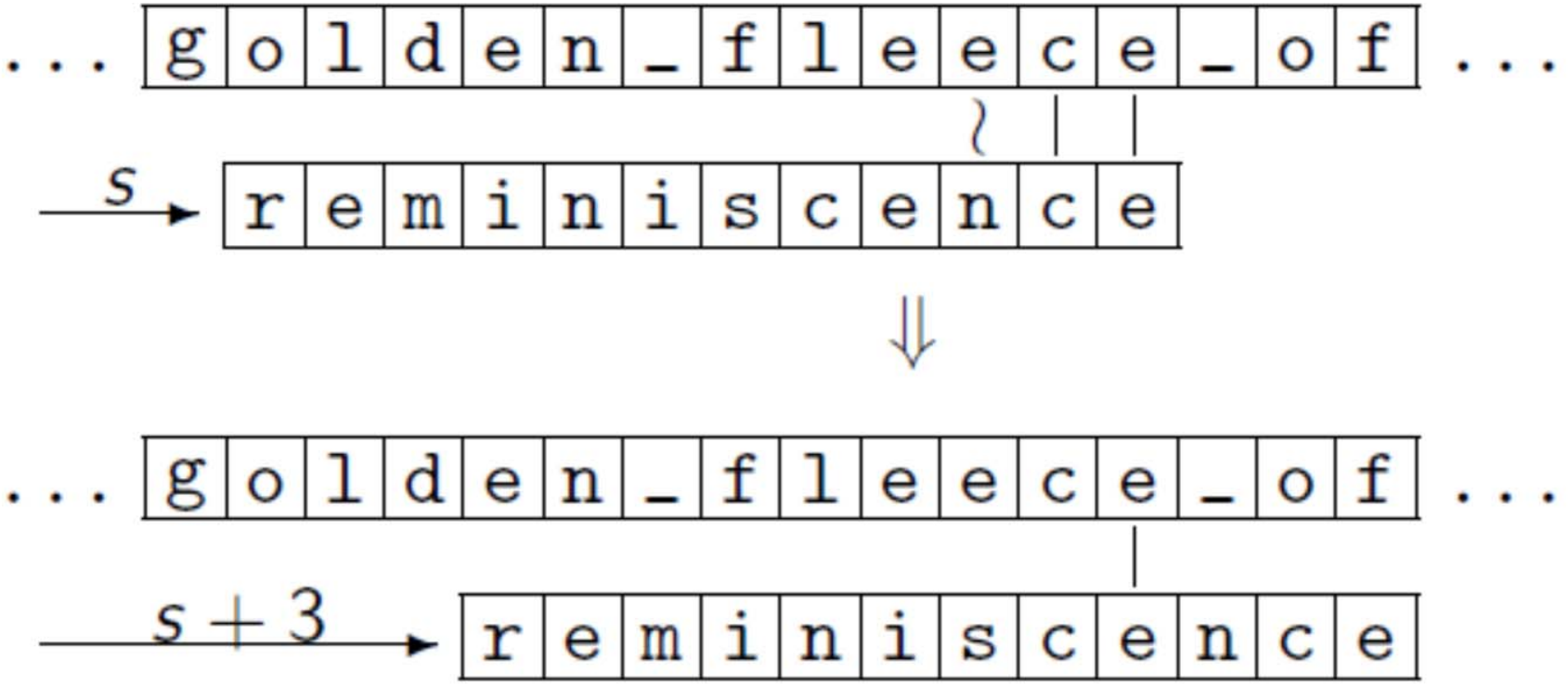
$$\lambda[T[s+m-1]] = \min\left(\{m\} \cup \{m-1-k \mid 0 \leq k \leq m-2 \text{ und } T[s+m-1] = P[k]\}\right)$$

„Letztes Vorkommen vom Zeichen im Muster ohne letztes Zeichen“

- bei einem Match wird um $\lambda[T[s+m-1]]$ verschoben

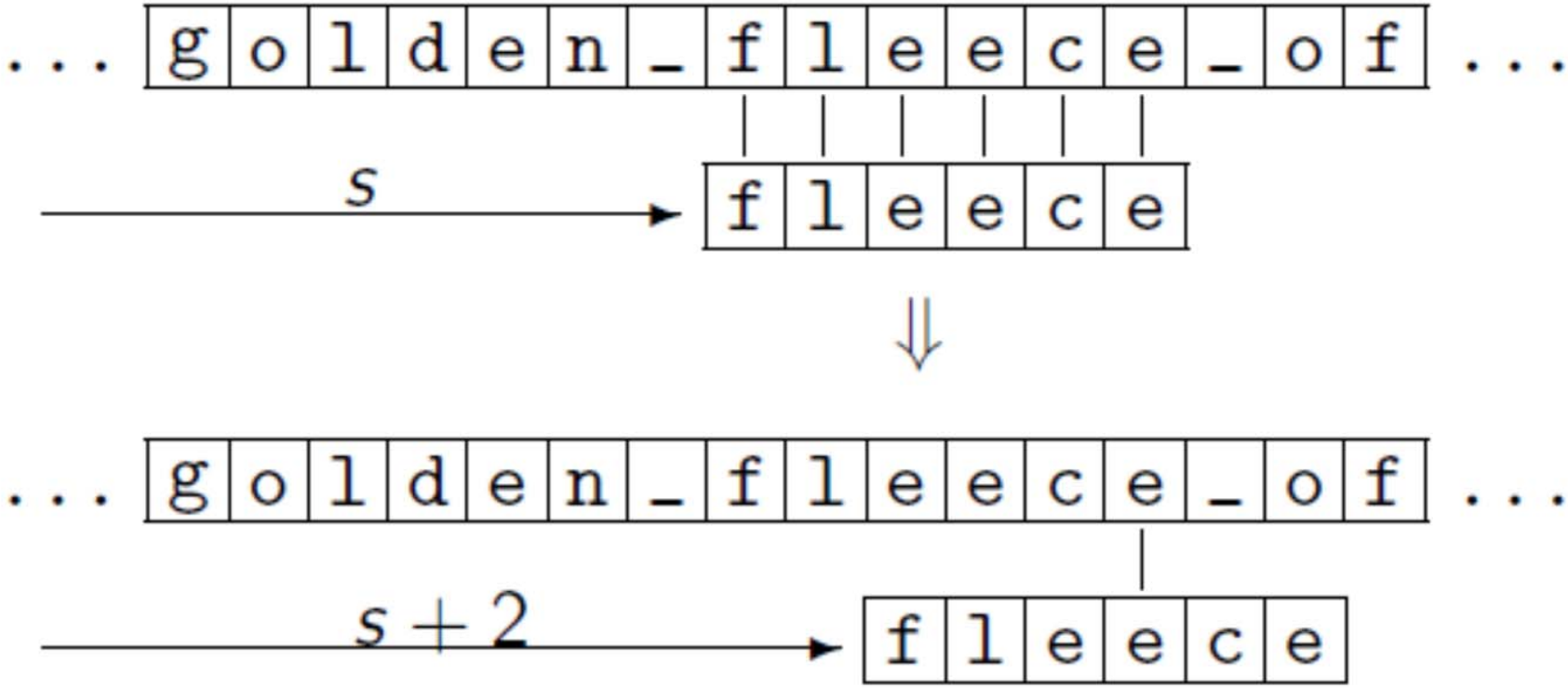
[Steffen 2009]

BMH: Verhalten bei Mismatch



[Steffen 2009]

BMH: Verhalten bei Treffer



[Steffen 2009]

BMH: Vorverarbeitung

DHBW
Duale Hochschule
Baden-Württemberg
Stuttgart
Campus Horb

Beispiel (Berechnung der Funktion λ)

```
computeLastOccurrenceFunction(P, Σ)
 1  m ← length[P]
 2  for each character a ∈ Σ do
 3     λ[a] = m
 4  for j ← 0 to m − 2 do
 5     λ[P[j]] ← m − 1 − j
 6  return λ
```

Die *worst-case*-Komplexität von `computeLastOccurrenceFunction` ist $\mathcal{O}(|\Sigma| + m)$.

[Steffen 2009]

BMH: Vorverarbeitung (II): Beispiel (I)

- Berechnung λ für Muster „ababaca“
- Initialisierung:
  - $m = 7$

	a	b	c
λ	7	7	7

- $j = 0$:
  - $\lambda[P(0)] = \lambda[a] = m - 1 - j = 6$

	a	b	c
λ	6	7	7

- $j = 1$:
  - $\lambda[P(1)] = \lambda[b] = m - 1 - j = 5$

	a	b	c
λ	6	5	7

BMH: Vorverarbeitung (III): Beispiel (II)

- $j = 2$:
  - $\lambda[P(2)] = \lambda[a] = m - 1 - j = 4$

	a	b	c
λ	4	5	7

- $j = 3$:
  - $\lambda[P(3)] = \lambda[b] = m - 1 - j = 3$

	a	b	c
λ	4	3	7

- $j = 4$:
  - $\lambda[P(4)] = \lambda[a] = m - 1 - j = 2$

	a	b	c
λ	2	3	7

BMH: Vorverarbeitung (IV): Beispiel (III)

- $j = 5$:
  - $\lambda[P(5)] = \lambda[c] = m - 1 - j = 1$

	a	b	c
λ	2	3	1

BMH: Algorithmus (I)

DHBW Duale Hochschule Baden-Württemberg Stuttgart Campus Horb

Beispiel (BMH-Matcher)

```
BMH-Matcher(T, P, Σ)
 1  n ← length[T]
 2  m ← length[P]
 3  λ ← computeLastOccurrenceFunction(P, Σ)
 4  s ← 0
 5  while s ≤ n − m do
 6     j ← m − 1
 7     while j ≥ 0 and P[j] = T[s + j] do
 8        j ← j − 1
 9     if j = −1 then
10        print "Pattern occurs with shift" s
11     s ← s + λ[T[s + m − 1]]
```

[Steffen 2009]

BMH: Algorithmus (II): Beispiel (I)

- Gegeben seien der Text „bababababacababacaabababab“ und das Muster „ababaca“

```
bababababacababacaabababab
ababaca
```

  - Mismatch $\Rightarrow$ Schiebe P um $\lambda[b] = 3$ weiter, setze $j = 6$

```
bababababacababacaabababab
   ababaca
```

  - Match $\Rightarrow$ Setze $j = j - 1 = 5$, weitervergleichen
  - …
  - Match $\Rightarrow$ Setze $j = j - 1 = -1$

```
babababababacababacaabababab
   ababaca
```

  - Treffer $(j = -1) \Rightarrow$ „Muster gefunden an Position 3“
  - Schiebe P um $\lambda[a] = 2$ weiter , setze $j = 6$

```
bababababacababacaabababab
     ababaca
```

BMH: Algorithmus (III): Beispiel (II)

bababababacabacaababababab
ababaca

- Match $\Rightarrow$ Setze $j = j - 1 = 5$, weitervergleichen

bababababacabacaababababab
ababaca

- Mismatch $\Rightarrow$ Schiebe P um $\lambda[a] = 2$ weiter, setze $j = 6$

bababababacabacaababababab
ababaca

- Match $\Rightarrow$ Setze $j = j - 1 = 5$, weitervergleichen

bababababacabacaababababab
ababaca

- Mismatch $\Rightarrow$ Schiebe P um $\lambda[a] = 2$ weiter, setze $j = 6$

bababababacabacaababababab
ababaca

BMH: Algorithmus (IV): Beispiel (III)

bababababacababacaababababab
ababaca

- Match $\Rightarrow$ Setze $j = j - 1 = 5$, weitervergleichen
- …
- Match $\Rightarrow$ Setze $j = j - 1 = -1$

bababababacababacaababababab
ababaca

- Treffer ($j = -1$) $\Rightarrow$ „Muster gefunden an Position 9“
- Schiebe P um $\lambda[a] = 2$ weiter, setze $j = 6$

bababababacababacaababababab
ababaca

- Mismatch $\Rightarrow$ Schiebe P um $\lambda[b] = 3$ weiter, setze $j = 6$

bababababacababacaababababab
ababaca

- Match $\Rightarrow$ Setze $j = j - 1 = 5$, weitervergleichen

BMH: Algorithmus (V): Beispiel (IV)

```
bababababacababacaababbabab
                  ababaca
```

- Mismatch $\Rightarrow$ Schiebe P um $\lambda[a] = 2$ weiter, setze $j = 6$

```
bababababacababacaababbabab
                    ababaca
```

- Match $\Rightarrow$ Setze $j = j - 1 = 5$, weitervergleichen

```
bababababacababacaababbabab
                    ababaca
```

- Mismatch $\Rightarrow$ Schiebe P um $\lambda[a] = 2$ weiter, setze $j = 6$
- Ende Text erreicht ($s > n - m$) $\Rightarrow$ Terminierung

BMH: Komplexität

- Überprüfung auf Gültigkeit Verschiebung: O(m)
- Worst-Case:
  - O($n * m + |\Sigma|$)
  - Beispiel: T = a^n und P = a^m
- In „normalen“ Texten:
  - $m << n$ und $|\Sigma| << n$
  - Damit: O(n)
- Für große Alphabete/kleine Muster wird meist O(n/m) erreicht, d.h. zumeist ist nur jedes m-te Zeichen zu inspizieren

Zusammenfassung (I)

- Motivation:
  - Beispiele
  - Präfix, Suffix, Verschiebung

- Brute Force Algorithmus:
  - „Probiere durch“
  - Mustervergleich links nach rechts um 1
  - Worst Case „unangenehm“

- Algorithmus von Knuth-Morris-Pratt:
  - Mustervergleich links nach rechts
  - Längere Verschiebungen
  - Präfixfunktion

Zusammenfassung (II)

- Algorithmus von Boyer-Moore:
  - Mustervergleich rechts nach links
  - Heuristiken:
    - Bad-Character
    - Good-Suffix
  - Modifiziere Bad-Character-Heuristik
  - Boyer-Moore-Horspool-Algorithmus

Übungen (I)

- **Aufgabe 1**

  Finden Sie im folgenden Rätsel fünf Begriffe aus diesem Vorlesungsabschnitt!

```
B J H S W N P Y D Z M M U F P K M P N C
U C C M R S Q Z A M I E Y R M N G V Y T
P M E O Y J W T B J C N A P L U X I M B
N F A H U B T N B M F E E C Q T F Z K N
C Z V J K H M C L W F N B U L H H U R Q
C F Y K K W C R S I D T E A F M V E J L
M H E S B U C T X Z G J F G A O P H D I
C O Z W N P H F A Y H I G Z X R O L U K
P E N Z B N U M A M B E S I S R A D K O
K P L O J N T Y J H S J F L Q I V Q Y B
Y L N U K R V Q I D K I G S K S R U P U
P U L T Z E D Z L M N K M U Q P Q W O N
X N I G K T I C H J Q V C Y Y R C A J K
Z O A U F T I U A Z Y K D E Z A Q S X R
N B C Y I A E Z Y L Q H B H C T F N L A
M Y X U A P H G B U O Y N U O T H T L U
E R O O M R E Y O B F H D W M C D Z G D
N N O I Q J P Q Q Y R M F U Q M X H U B
C V S Y A Y K G C T W I M X U P A O I Q
A N Q O D W Y U F S O U U D R J T Q O X
```

Übungen (II)

- **Aufgabe 2**
  - Welche Komplexität hat der Brute-Force-Algorithmus?
  - Was ist der Unterschied zwischen dem Brute-Force-Algorithmus und dem Algorithmus von Knuth-Morris-Pratt?
  - Welche Algorithmen benötigen eine Vorverarbeitung?
  - Welche Algorithmen vergleichen von links nach rechts?
  - Was unterscheidet den BMH-Algorithmus von den anderen?
  - Was ist die Good-Suffix-Heuristik?

Übungen (III)

- **Aufgabe 3**

  Wenden Sie auf das folgende Beispiel den Brute-Force-Algorithmus, den Algorithmus von Knuth-Morris-Pratt und den Algorithmus von Boyer-Moore-Horspool an!

  Vergleichen Sie die Anzahl der durchgeführten Vergleiche!

  Text: ABCAABABAABABC

  Muster: ABABC