technische universität
dortmund

Algorithmen auf Sequenzen

Erweiterte Patternklassen

Dominik Kopczynski

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

Übersicht

- Bisher wurden nur exakte Pattern besprochen.
- Es ist auch interessant Pattern zu betrachten, bei denen bestimmte Zeichen an einer Position beliebig sein dürfen (verallgemeinerte Strings).
- Eine weitere Erweiterung der Pattern wäre es Gaps beschränkter Länge zu betrachten.
- Auch kann es vorkommen, dass Zeichen im Pattern optional sein können.
- In der Summe könnte so nach "einfachen" regulären Ausdrücken gesucht werden.

Verallgemeinerte Strings

- Verallgemeinerte Strings sind Strings, die *Teilmengen* des Alphabets als Zeichen besitzen.
- Die Patternmenge {Meier, Meyer} könnte somit zu Me[iy]er verallgemeinert werden.
- Wenn ein(!) beliebiges Zeichen erlaubt ist, kann ein Pattern folgendermaßen dargestellt werden: image#.jpg .

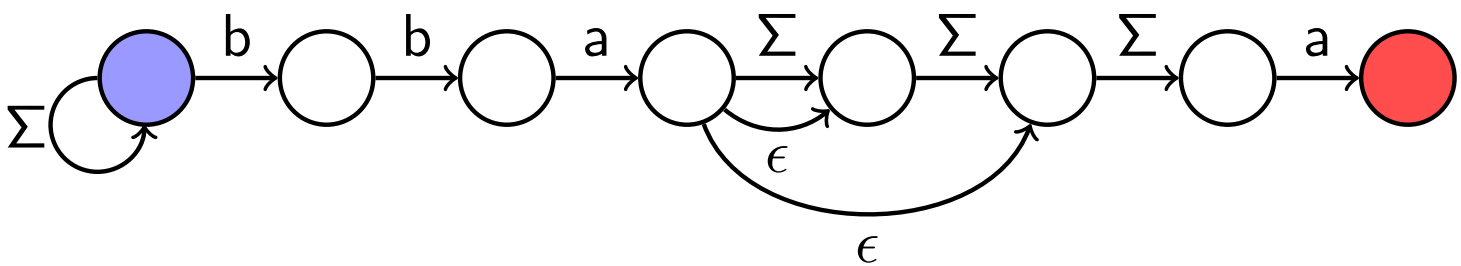
Verallgemeinerte Strings

- Verallgemeinerte Strings sind Strings, die *Teilmengen* des Alphabets als Zeichen besitzen.
- Die Patternmenge {Meier, Meyer} könnte somit zu Me[iy]er verallgemeinert werden.
- Wenn ein(!) beliebiges Zeichen erlaubt ist, kann ein Pattern folgendermaßen dargestellt werden: image#.jpg .
- Bitparallele Algorithmen lassen sich ohne viel Aufwand anpassen, indem beispielsweise beim Shift-And nur die Masken angepasst werden. Beispiel: $P = \texttt{abba\#b}$ mit $\Sigma = \{\texttt{a}, \texttt{b}\}$.

	b#abba
$mask^a$	011001_2
$mask^b$	110110_2

Gaps beschränkter Länge

- Unter Gaps beschränkter Länge wird eine nichtleere Folge beliebiger Zeichen verstanden.
- Ein Beispielpattern sei: $P = \texttt{bba\#(1,3)a}$.
- Hierbei muss es 1 beliebiges Zeichen und bis zu $3 - 1 = 2$ optionale beliebige Zeichen geben.
- Es werden also zwei ϵ-Übergänge benötigt.



Gaps beschränkter Länge

Einschränkungen:

1. Die Gaps $\#(,)$ sind nicht an erster oder letzter Position im Pattern erlaubt.
2. Zwei (oder mehrere) Gaps hintereinander sind nicht erlaubt, denn $\#(u,v)\#(u',v') \mathrel{\widehat{=}} \#(u+u',v+v')$.
3. Für $\#(u,v)$ muss gelten: $1 \leq u \leq v$.

Implementierung

Für die Masken des Alphabets gilt: bei einem Gap $\#(u, v)$ werden für jedes Zeichen aus dem Alphabet genau v `1`en hinzugefügt. Beispiel mit $P = \texttt{bba\#(1,3)a}$ und $\Sigma = \{\texttt{a}, \texttt{b}, \texttt{c}\}$:

	`a###abb`
$mask^a$	1111100_2
$mask^b$	0111011_2
$mask^c$	0111000_2

Implementierung

Zur Simulation der ϵ-Übergänge bringt die Subtraktion auf Bit-Ebene den gewünschten Effekt.

- Bitmaske I speichert für jedes Gap-Token die Position, von der ϵ-Kanten ausgehen. Angenommen an Position i im Pattern ist ein Gap-Token, so wird $I[i-1] := 1$ gesetzt.

Implementierung

Zur Simulation der ϵ-Übergänge bringt die Subtraktion auf Bit-Ebene den gewünschten Effekt.

- Bitmaske I speichert für jedes Gap-Token die Position, von der ϵ-Kanten ausgehen. Angenommen an Position i im Pattern ist ein Gap-Token, so wird $I[i-1] := 1$ gesetzt.
- Bitmaske F speichert für jedes Gap-Token die Position, die nach der letzten ϵ-Kante erreicht wird. Angenommen an Position i im Pattern ist ein Gap-Token, so wird $F[i+v-u+1] := 1$ gesetzt.

	a###abb
F	0100000_2
I	0000100_2

Implementierung

- Um festzustellen, ob sich ein aktiver Zustand am Beginn eines Gaps befindet, wird die aktive Zustandsmenge mit der Startbitmaske verundet: $A \,\&\, I$.
- Durch Subtraktion von F und $A \,\&\, I$ werden alle Bits zwischen $[u : v]$ auf `1` gesetzt.

F	0100000_2
$A \,\&\, I$	0000100_2
$-$	0011100_2

Implementierung

- Um festzustellen, ob sich ein aktiver Zustand am Beginn eines Gaps befindet, wird die aktive Zustandsmenge mit der Startbitmaske verundet: $A \,\&\, I$.
- Durch Subtraktion von F und $A \,\&\, I$ werden alle Bits zwischen $[u : v]$ auf $\mathtt{1}$ gesetzt.

F	$\mathtt{0100000}_2$
$A \,\&\, I$	$\mathtt{0000100}_2$
$-$	$\mathtt{0011100}_2$

- Problem, wenn es mehrere Gaps gibt:

F	$\mathtt{0100001000000}_2$
$A \,\&\, I$	$\mathtt{0000000000100}_2$
$-$	$\mathtt{0100000011100}_2$

Implementierung

- Lösung: Durch die Negation von F wird eine Maske erzeugt, die genutzt werden kann, um ungewünschte `1`en zu löschen.

F	$01000010000{0}_2$
$A \,\&\, I$	$00000000010{0}_2$
$F - (A \,\&\, I)$	$01000001110{0}_2$

Implementierung

- Lösung: Durch die Negation von F wird eine Maske erzeugt, die genutzt werden kann, um ungewünschte `1`en zu löschen.

F	010000100000_2
$A \,\&\, I$	000000000100_2
$F - (A \,\&\, I)$	010000011100_2
$\sim F$	101111011111_2
$(F - (A \,\&\, I)) \,\&\, \sim F$	000000011100_2

Implementierung

- Lösung: Durch die Negation von F wird eine Maske erzeugt, die genutzt werden kann, um ungewünschte `1`en zu löschen.

F	010000100000_2
$A \,\&\, I$	000000000100_2
$F - (A \,\&\, I)$	010000011100_2
$\sim F$	101111011111_2
$(F - (A \,\&\, I)) \,\&\, \sim F$	000000011100_2

- Wegen der Einschränkung, dass zwei Gaps nicht unmittelbar aufeinander folgen dürfen, ist kein F-Zustand gleichzeitig ein I-Zustand des nächsten Gaps.

Implementierung

Erweiterte Update-Funktion:

1. Herkömmliche Shift-And-Update-Funktion anwenden:
   ```
   A = ((A << 1) | 1) & mask[c]
   ```
2. Zustandsvektor um neue aktive Zustände erweitern:
   ```
   A = A | ((F - (A & I)) & ~F)
   ```

Beispiel für den Gap-Shift-And

Sei $P = \texttt{bba\#(1,3)a}$ und $T = \texttt{bbacca}$:

	a###abb
$mask^a$	1111100_2
$mask^b$	0111011_2
$mask^c$	0111000_2

F	0100000_2
$(A \,\&\, I)$	0000000_2
$=$	0000000_2

$$A = 0000000$$

Beispiel für den Gap-Shift-And

Sei $P = \texttt{bba\#(1,3)a}$ und $T = \texttt{bbacca}$:

	a###abb
$mask^a$	1111100_2
$mask^b$	0111011_2
$mask^c$	0111000_2

F	0100000_2
$(A \,\&\, I)$	0000000_2
$=$	0000000_2

Update 1 mit b: $A = 0000001$

Beispiel für den Gap-Shift-And

Sei $P = \texttt{bba\#(1,3)a}$ und $T = \texttt{bbacca}$:

	a###abb
$mask^a$	1111100_2
$mask^b$	0111011_2
$mask^c$	0111000_2

F	0100000_2
$(A \,\&\, I)$	0000000_2
$=$	0000000_2

Update 2: $A = 0000001$

Beispiel für den Gap-Shift-And

Sei $P = \texttt{bba\#(1,3)a}$ und $T = \texttt{bbacca}$:

	a###abb
$mask^a$	1111100_2
$mask^b$	0111011_2
$mask^c$	0111000_2

F	0100000_2
$(A \,\&\, I)$	0000000_2
$=$	0000000_2

Update 1 mit b: $A = 0000011$

Beispiel für den Gap-Shift-And

Sei $P = \texttt{bba\#(1,3)a}$ und $T = \texttt{bbacca}$:

	a###abb
$mask^a$	1111100_2
$mask^b$	0111011_2
$mask^c$	0111000_2

F	0100000_2
$(A \,\&\, I)$	0000000_2
$=$	0000000_2

Update 2: $A = 0000011$

Beispiel für den Gap-Shift-And

Sei $P = \texttt{bba\#(1,3)a}$ und $T = \texttt{bbacca}$:

	a###abb
$mask^a$	1111100_2
$mask^b$	0111011_2
$mask^c$	0111000_2

F	0100000_2
$(A \,\&\, I)$	0000000_2
$=$	0000000_2

Update 1 mit a: $A = 0000100$

Beispiel für den Gap-Shift-And

Sei $P = \texttt{bba\#(1,3)a}$ und $T = \texttt{bbacca}$:

	a###abb
$mask^a$	1111100_2
$mask^b$	0111011_2
$mask^c$	0111000_2

F	0100000_2
$(A \,\&\, I)$	0000100_2
$=$	0011100_2

Update 2: $A = 0011100$

Beispiel für den Gap-Shift-And

Sei $P = \texttt{bba\#(1,3)a}$ und $T = \texttt{bbacca}$:

	a###abb
$mask^a$	1111100_2
$mask^b$	0111011_2
$mask^c$	0111000_2

F	0100000_2
$(A \,\&\, I)$	0000000_2
$=$	0000000_2

Update 1 mit c: $A = 0111000$

Beispiel für den Gap-Shift-And

Sei $P = \texttt{bba\#(1,3)a}$ und $T = \texttt{bbacca}$:

	a###abb
$mask^a$	1111100_2
$mask^b$	0111011_2
$mask^c$	0111000_2

F	0100000_2
$(A \,\&\, I)$	0000000_2
$=$	0000000_2

Update 2: $A = 0111000$

Beispiel für den Gap-Shift-And

Sei $P = \texttt{bba\#(1,3)a}$ und $T = \texttt{bbacca}$:

	a###abb
$mask^a$	1111100_2
$mask^b$	0111011_2
$mask^c$	0111000_2

F	0100000_2
$(A \,\&\, I)$	0000000_2
$=$	0000000_2

Update 1 mit c: $A = 0110000$

Beispiel für den Gap-Shift-And

Sei $P = \texttt{bba\#(1,3)a}$ und $T = \texttt{bbacca}$:

	a###abb
$mask^a$	1111100_2
$mask^b$	0111011_2
$mask^c$	0111000_2

F	0100000_2
$(A \,\&\, I)$	0000000_2
$=$	0000000_2

Update 2: $A = 0110000$

Beispiel für den Gap-Shift-And

Sei $P = \texttt{bba\#(1,3)a}$ und $T = \texttt{bbacca}$:

	a###abb
$mask^a$	1111100_2
$mask^b$	0111011_2
$mask^c$	0111000_2

F	0100000_2
$(A \,\&\, I)$	0000000_2
$=$	0000000_2

Update 1 mit a: $A = 1100000$

Beispiel für den Gap-Shift-And

Sei $P = \texttt{bba\#(1,3)a}$ und $T = \texttt{bbacca}$:

	a###abb
$mask^a$	1111100_2
$mask^b$	0111011_2
$mask^c$	0111000_2

F	0100000_2
$(A \,\&\, I)$	0000000_2
$=$	0000000_2

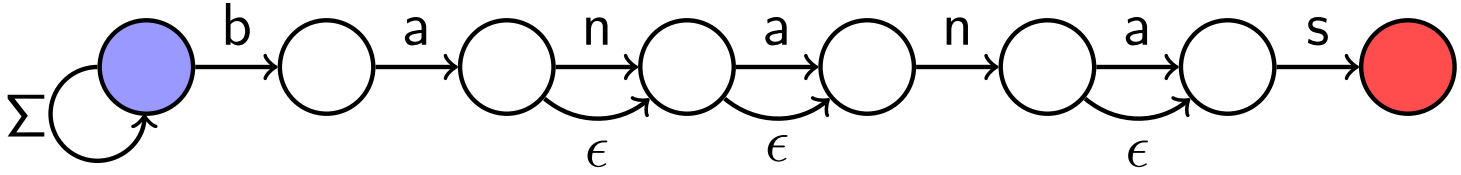
Update 2: $A = 1100000$ Treffer

Optionale Zeichen

- Mit einer zusätzlichen Erweiterung können Zeichen verarbeitet werden, die bis zu einmal vorkommen dürfen.
- Eine Notation für die Patternmenge {color, colour} wäre beispielsweise: $P = \texttt{colou?r}$.
- Konsekutive ϵ-Transitionen (Blöcke) sind erlaubt.
- Auch diese Erweiterung lässt sich mit dem Shift-And-Algorithmus realisieren.

Optionale Zeichen

Implementierung:
Für die folgenden Berechnungen sind drei zusätzliche Masken nötig. Am Beispiel-NFA für das Pattern $P = \texttt{ban?a?na?s}$ werden die Masken verdeutlicht:



I :	0	1	0	0	1	0	0
F :	0	0	0	1	0	1	0
O :	0	0	1	1	0	1	0

Optionale Zeichen

Implementierung:

- Sobald ein Zustand innerhalb eines Blocks aktiv ist, sind seine Folgezustände (innerhalb des Blocks) auch aktiv.
- Wie findet man den ersten aktiven Zustand?

Optionale Zeichen

Implementierung:

- Sobald ein Zustand innerhalb eines Blocks aktiv ist, sind seine Folgezustände (innerhalb des Blocks) auch aktiv.
- Wie findet man den ersten aktiven Zustand?
- Subtraktion wieder ausnutzen: wenn bei einer Binärzahl um 1 subtrahiert wird, werden alle niederwertigen Bits bis einschließlich der ersten 1 geflippt.
- Alle höherwertigen Bits bleiben erhalten.

Optionale Zeichen

Implementierung:

- Sobald ein Zustand innerhalb eines Blocks aktiv ist, sind seine Folgezustände (innerhalb des Blocks) auch aktiv.
- Wie findet man den ersten aktiven Zustand?
- Subtraktion wieder ausnutzen: wenn bei einer Binärzahl um 1 subtrahiert wird, werden alle niederwertigen Bits bis einschließlich der ersten 1 geflippt.
- Alle höherwertigen Bits bleiben erhalten.

```
  1101010000
-            1
------------
  1101001111
```

```
  1101011000
-          100
------------
  1101010100
```

Optionale Zeichen

Implementierung:

- Damit der Minuend immer größer als der Subtrahend ist, wird die aktive Zustandsmenge mit den Endpositionen der Blöcke verodert, sei $A_f := A \mid F$.

Optionale Zeichen

Implementierung:

- Damit der Minuend immer größer als der Subtrahend ist, wird die aktive Zustandsmenge mit den Endpositionen der Blöcke verodert, sei $A_f := A \mid F$.
- Mit der Subtraktion werden die niederwertigsten Bits geflippt: $A_f - I$.

Optionale Zeichen

Implementierung:

- Damit der Minuend immer größer als der Subtrahend ist, wird die aktive Zustandsmenge mit den Endpositionen der Blöcke verodert, sei $A_f := A \mid F$.
- Mit der Subtraktion werden die niederwertigsten Bits geflippt: $A_f - I$.
- Da es keine Maschinenoperation für bitweise Äquivalenz gibt, wird die exklusive Veroderung genutzt: $a \equiv b \mathrel{\widehat{=}} \sim a \oplus b$.

Optionale Zeichen

Implementierung:

- Damit der Minuend immer größer als der Subtrahend ist, wird die aktive Zustandsmenge mit den Endpositionen der Blöcke verodert, sei $A_f := A \mid F$.
- Mit der Subtraktion werden die niederwertigsten Bits geflippt: $A_f - I$.
- Da es keine Maschinenoperation für bitweise Äquivalenz gibt, wird die exklusive Veroderung genutzt: $a \equiv b \mathrel{\widehat{=}} \sim a \oplus b$.
- Seien also $\sim(A_f - I) \oplus A_f$ die äquivalenten Bits.

Optionale Zeichen

Implementierung:

- Damit der Minuend immer größer als der Subtrahend ist, wird die aktive Zustandsmenge mit den Endpositionen der Blöcke verodert, sei $A_f := A \mid F$.
- Mit der Subtraktion werden die niederwertigsten Bits geflippt: $A_f - I$.
- Da es keine Maschinenoperation für bitweise Äquivalenz gibt, wird die exklusive Veroderung genutzt: $a \equiv b \mathrel{\widehat{=}} \sim a \oplus b$.
- Seien also $\sim(A_f - I) \oplus A_f$ die äquivalenten Bits.
- Zuletzt sollen noch alle Bits auf 0 gesetzt werden, die sich nicht im Block befinden, mit $O \,\&\, (\sim(A_f - I) \oplus A_f)$.

Optionale Zeichen

Implementierung:

1. Masken für alle Zeichen aus Σ erstellen, dabei die optionalen Zeichen wie reguläre Zeichen behandeln.
2. Herkömmliche Shift-And-Update-Funktion anwenden:
   ```
   A = ((A << 1) | 1) & mask[c]
   ```
3. Zustandsvektor um neue aktive Zustände erweitern:
   ```
   A_f = A | F
   A = A | (O & (~(A_f - I) ^ A_f))
   ```

Beispiel für den Optional-Shift-And

Sei $P = \texttt{ban?a?na?s}$ und $T = \texttt{banns}$:

	sananab
$mask^a$	0101010_2
$mask^b$	0000001_2
$mask^n$	0010100_2
$mask^s$	1000000_2

I	0010010_2
F	0101000_2
O	0101100_2
$O \,\&\, (\sim(A_f - I) \oplus A_f)$	0000000_2

$$A = 0000000$$

Beispiel für den Optional-Shift-And

Sei $P = \texttt{ban?a?na?s}$ und $T = \texttt{banns}$:

	sananab
$mask^a$	0101010_2
$mask^b$	0000001_2
$mask^n$	0010100_2
$mask^s$	1000000_2

I	0010010_2
F	0101000_2
O	0101100_2
$O \;\&\; (\sim(A_f - I) \oplus A_f)$	0000000_2

Update 1 mit b: $A = 0000001$

Beispiel für den Optional-Shift-And

Sei $P = \texttt{ban?a?na?s}$ und $T = \texttt{banns}$:

	sananab
$mask^a$	0101010_2
$mask^b$	0000001_2
$mask^n$	0010100_2
$mask^s$	1000000_2

I	0010010_2
F	0101000_2
O	0101100_2
$O \,\&\, (\sim(A_f - I) \oplus A_f)$	0000000_2

Update 2: $A = 0000001$

Beispiel für den Optional-Shift-And

Sei $P = \texttt{ban?a?na?s}$ und $T = \texttt{banns}$:

	sananab
$mask^a$	0101010_2
$mask^b$	0000001_2
$mask^n$	0010100_2
$mask^s$	1000000_2

I	0010010_2
F	0101000_2
O	0101100_2
$O \,\&\, (\sim(A_f - I) \oplus A_f)$	0000000_2

Update 1 mit a: $A = 0000010$

Beispiel für den Optional-Shift-And

Sei $P = \texttt{ban?a?na?s}$ und $T = \texttt{banns}$:

	sananab
$mask^a$	0101010_2
$mask^b$	0000001_2
$mask^n$	0010100_2
$mask^s$	1000000_2

I	0010010_2
F	0101000_2
O	0101100_2
$O \,\&\, (\sim(A_f - I) \oplus A_f)$	0001100_2

Update 2: $A = 0001110$

Beispiel für den Optional-Shift-And

Sei $P = \texttt{ban?a?na?s}$ und $T = \texttt{banns}$:

	sananab
$mask^a$	0101010_2
$mask^b$	0000001_2
$mask^n$	0010100_2
$mask^s$	1000000_2

I	0010010_2
F	0101000_2
O	0101100_2
$O \,\&\, (\sim(A_f - I) \oplus A_f)$	0000000_2

Update 1 mit n: $A = 0010100$

Beispiel für den Optional-Shift-And

Sei $P = \texttt{ban?a?na?s}$ und $T = \texttt{banns}$:

	sananab
$mask^a$	0101010_2
$mask^b$	0000001_2
$mask^n$	0010100_2
$mask^s$	1000000_2

I	0010010_2
F	0101000_2
O	0101100_2
$O \,\&\, (\sim(A_f - I) \oplus A_f)$	0101100_2

Update 2: $A = 0111100$

Beispiel für den Optional-Shift-And

Sei $P = \texttt{ban?a?na?s}$ und $T = \texttt{banns}$:

	sananab
$mask^a$	0101010_2
$mask^b$	0000001_2
$mask^n$	0010100_2
$mask^s$	1000000_2

I	0010010_2
F	0101000_2
O	0101100_2
$O \;\&\; (\sim(A_f - I) \oplus A_f)$	0000000_2

Update 1 mit n: $A = 0010000$

Beispiel für den Optional-Shift-And

Sei $P = \texttt{ban?a?na?s}$ und $T = \texttt{banns}$:

	sananab
$mask^a$	0101010_2
$mask^b$	0000001_2
$mask^n$	0010100_2
$mask^s$	1000000_2

I	0010010_2
F	0101000_2
O	0101100_2
$O \,\&\, (\sim(A_f - I) \oplus A_f)$	0100000_2

Update 2: $A = 0110000$

Beispiel für den Optional-Shift-And

Sei $P = \texttt{ban?a?na?s}$ und $T = \texttt{banns}$:

	sananab
$mask^a$	0101010_2
$mask^b$	0000001_2
$mask^n$	0010100_2
$mask^s$	1000000_2

I	0010010_2
F	0101000_2
O	0101100_2
$O \;\&\; (\sim(A_f - I) \oplus A_f)$	0000000_2

Update 1 mit $\texttt{s}$: $A = 1000000$

Beispiel für den Optional-Shift-And

Sei $P = \texttt{ban?a?na?s}$ und $T = \texttt{banns}$:

	sananab
$mask^a$	0101010_2
$mask^b$	0000001_2
$mask^n$	0010100_2
$mask^s$	1000000_2

I	0010010_2
F	0101000_2
O	0101100_2
$O \,\&\, (\sim(A_f - I) \oplus A_f)$	0000000_2

Update 2: $A = 1000000$ Treffer

Zusammenfassung

- Durch einfache Tricks lässt sich der Shift-And-Algorithmus auf erweiterte Patternklassen ausbauen.
- Verallgemeinerte Strings, Gaps beschränkter Länge und optionale Zeichen sind möglich.
- Die Laufzeit für die Erstellung der Masken für Wildcards und Gaps beschränkter Länge beträgt $\mathcal{O}(m|\Sigma|)$.

Zusammenfassung

- Durch einfache Tricks lässt sich der Shift-And-Algorithmus auf erweiterte Patternklassen ausbauen.
- Verallgemeinerte Strings, Gaps beschränkter Länge und optionale Zeichen sind möglich.
- Die Laufzeit für die Erstellung der Masken für Wildcards und Gaps beschränkter Länge beträgt $\mathcal{O}(m|\Sigma|)$.
- Die Erstellung der Masken für optionale Zeichen bleibt unverändert, die Laufzeit betägt also $\mathcal{O}(m)$.
- Der Speicherplatz beträgt weiterhin $\mathcal{O}(|\Sigma|\lceil m/W \rceil)$.
- Die Laufzeit für die Mustersuche bleibt unverändert bei $\mathcal{O}(n\lceil m/W \rceil)$.