3.36pt

UA RUHR

Algorithmen auf Sequenzen

Exakte Mustersuche: NFAs

Sven Rahmann

Genominformatik
Universitätsklinikum Essen
Universität Duisburg-Essen
Universitätsallianz Ruhr

Exakte Mustersuche

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: $T[i : i + m] = P$.

Exakte Mustersuche

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: $T[i : i + m] = P$.

Naiver Algorithmus: $\mathcal{O}(mn)$ Laufzeit

Exakte Mustersuche

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: $T[i : i + m] = P$.

Naiver Algorithmus: $\mathcal{O}(mn)$ Laufzeit

Idee:
Jedes Zeichen des Musters nur einmal lesen,
jedes Zeichen des Textes nur einmal lesen

Exakte Mustersuche

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: $T[i : i + m] = P$.

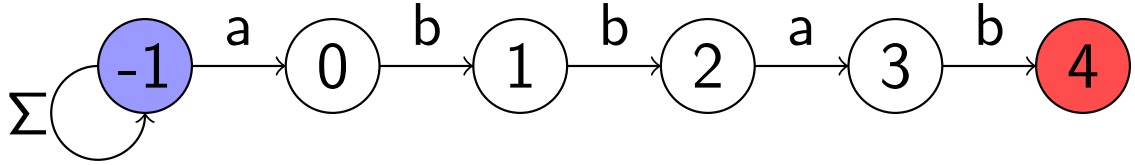
Naiver Algorithmus: $\mathcal{O}(mn)$ Laufzeit

Idee:
Jedes Zeichen des Musters nur einmal lesen,
jedes Zeichen des Textes nur einmal lesen

Definieren Automaten, der alle Strings der Form $\Sigma^* P$ akzeptiert.

NFA für die exakte Mustersuche

Beispiel für NFA des Musters **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 $\mathcal{O}(m)$.

NFA

UA RUHR

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

UA RUHR

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 $x \in \Sigma^*$.

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

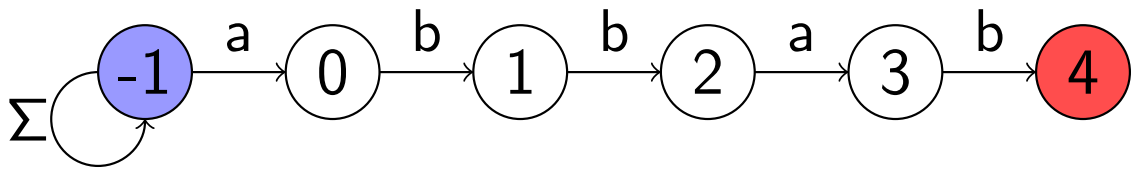
NFA

Einführung von Epsilon-Transitionen:
Für jeden Zustand q sei E_q sein ϵ-**Abschluss**:
Menge der Zustände, die von q aus „sofort“ erreicht wird.

$$\Delta(q, \epsilon) := E_q$$

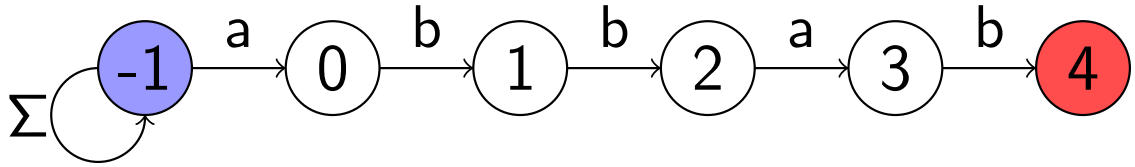
NFA für die exakte Mustersuche

NFA soll alle Strings der Form $\Sigma^* P$ akzeptieren:
Kette von Zuständen, an deren Kanten entlang P buchstabiert ist.
Der Startzustand besitzt eine Schleife, so dass er immer aktiv ist.



NFA für die exakte Mustersuche

NFA soll alle Strings der Form $\Sigma^* P$ akzeptieren:
Kette von Zuständen, an deren Kanten entlang P buchstabiert ist.
Der Startzustand besitzt eine Schleife, so dass er immer aktiv ist.



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

Bitparalelle NFA-Simulation

UA RUHR

Beobachtungen:

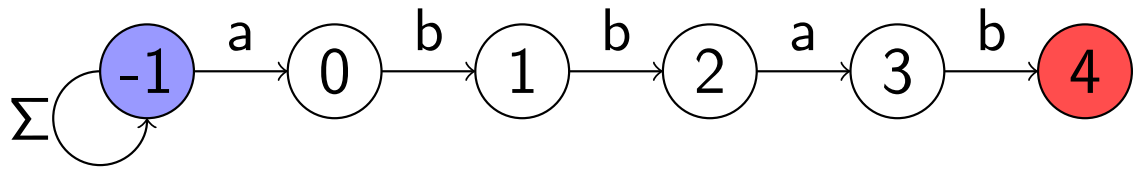
- Direkte Verwendung des NFA schien ineffizient:
  $\mathcal{O}(m)$ aktive Zustände
- Information, ob ein Zustand aktiv ist oder nicht, ist binär.

Bitparalelle NFA-Simulation

Beobachtungen:

- Direkte Verwendung des NFA schien ineffizient:
  $\mathcal{O}(m)$ aktive Zustände
- Information, ob ein Zustand aktiv ist oder nicht, ist binär.
- Idee: Ist Musterlänge m kleiner als Registergröße W, dann kann man die aktiven Zustände in einem Register speichern.
- Zustands-Update erfolgt über elementare Bit-Operationen

Der Shift-And-Algorithmus



- Zustand -1 immer aktiv, nicht Teil des Registers
- Zustand $0 \leq q < |P|$ ist aktiv,
  wenn Präfix $P[: q+1]$ gelesen wurde;
  Repräsentation durch Bit q im Register
- Zustände des NFA werden parallel aktualisiert.
- Ist $m \leq W$, dann ist die Such-Laufzeit $\mathcal{O}(n\lceil m/W \rceil) = \mathcal{O}(n)$.

Der Shift-And-Algorithmus

Durchführung:

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

Der Shift-And-Algorithmus

Suche mit dem Shift-And-Algorithmus nach $P =$ ababaca:

```
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:

$accept =$ 1000000

mask:

a: 1010101

b: 0001010

c: 0100000

Der Shift-And-Algorithmus

Suche mit dem Shift-And-Algorithmus nach $P =$ ababaca:

```
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: $\ll$

$accept =$ 1000000

mask:

a: 1010101

b: 0001010

c: 0100000

Der Shift-And-Algorithmus

Suche mit dem Shift-And-Algorithmus nach $P =$ ababaca:

```
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

Suche mit dem Shift-And-Algorithmus nach $P =$ ababaca:

```
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

Suche mit dem Shift-And-Algorithmus nach $P =$ ababaca:

```
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

Suche mit dem Shift-And-Algorithmus nach $P =$ ababaca:

```
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: $\ll$

$accept =$ 1000000

mask:

a: 1010101

b: 0001010

c: 0100000

Der Shift-And-Algorithmus

Suche mit dem Shift-And-Algorithmus nach $P =$ ababaca:

```
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: | 0000001

$accept =$ 1000000

mask:

a: 1010101

b: 0001010

c: 0100000

Der Shift-And-Algorithmus

Suche mit dem Shift-And-Algorithmus nach $P =$ ababaca:

```
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

Suche mit dem Shift-And-Algorithmus nach $P =$ ababaca:

```
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

Suche mit dem Shift-And-Algorithmus nach $P =$ ababaca:

```
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

Suche mit dem Shift-And-Algorithmus nach $P =$ ababaca:

```
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

Suche mit dem Shift-And-Algorithmus nach $P =$ ababaca:

```
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:	& 0100000	b: 0001010
accept $=$	1000000	c: 0100000

Der Shift-And-Algorithmus

Suche mit dem Shift-And-Algorithmus nach $P =$ ababaca:

```
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

Suche mit dem Shift-And-Algorithmus nach $P =$ ababaca:

```
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: $\ll$

$accept =$ 1000000

mask:

a: 1010101

b: 0001010

c: 0100000

Der Shift-And-Algorithmus

Suche mit dem Shift-And-Algorithmus nach $P =$ ababaca:

```
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

Suche mit dem Shift-And-Algorithmus nach $P =$ ababaca:

```
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

Suche mit dem Shift-And-Algorithmus nach $P =$ ababaca:

```
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

Suche mit dem Shift-And-Algorithmus nach $P =$ ababaca:

```
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: $\ll$

$accept =$ 1000000

mask:

a: 1010101

b: 0001010

c: 0100000

Der Shift-And-Algorithmus

Suche mit dem Shift-And-Algorithmus nach $P =$ ababaca:

```
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: | 0000001

$accept =$ 1000000

mask:

a: 1010101

b: 0001010

c: 0100000

Der Shift-And-Algorithmus

Suche mit dem Shift-And-Algorithmus nach $P =$ ababaca:

```
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

Suche mit dem Shift-And-Algorithmus nach $P =$ ababaca:

```
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

Suche mit dem Shift-And-Algorithmus nach $P =$ ababaca:

```
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

Suche mit dem Shift-And-Algorithmus nach $P =$ ababaca:

```
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

Suche mit dem Shift-And-Algorithmus nach $P =$ ababaca:

```
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

Suche mit dem Shift-And-Algorithmus nach $P =$ ababaca:

```
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:	A & *accept*	b: 0001010
accept $=$	1000000	c: 0100000

Der Shift-And-Algorithmus

Suche mit dem Shift-And-Algorithmus nach $P =$ ababaca:

```
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: $\ll$

$accept =$ 1000000

mask:

a: 1010101

b: 0001010

c: 0100000

Der Shift-And-Algorithmus

Suche mit dem Shift-And-Algorithmus nach $P =$ ababaca:

```
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

Suche mit dem Shift-And-Algorithmus nach $P =$ ababaca:

```
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

Suche mit dem Shift-And-Algorithmus nach $P =$ ababaca:

```
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:	A & *accept*	b: 0001010
accept $=$	1000000	c: 0100000

Der Shift-And-Algorithmus

Suche mit dem Shift-And-Algorithmus nach $P =$ ababaca:

```
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: $\ll$

$accept =$ 1000000

mask:

a: 1010101

b: 0001010

c: 0100000

Der Shift-And-Algorithmus

Suche mit dem Shift-And-Algorithmus nach $P =$ ababaca:

```
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

Suche mit dem Shift-And-Algorithmus nach $P =$ ababaca:

```
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

Suche mit dem Shift-And-Algorithmus nach $P =$ ababaca:

```
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:	A & *accept*	b: 0001010
accept $=$	1000000	c: 0100000

Der Shift-And-Algorithmus

Suche mit dem Shift-And-Algorithmus nach $P =$ ababaca:

```
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: $\ll$

$accept =$ 1000000

mask:

a: 1010101

b: 0001010

c: 0100000

Der Shift-And-Algorithmus

Suche mit dem Shift-And-Algorithmus nach $P =$ ababaca:

```
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

Suche mit dem Shift-And-Algorithmus nach $P =$ ababaca:

```
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

Suche mit dem Shift-And-Algorithmus nach $P =$ ababaca:

```
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:	A & *accept*	b:	0001010
accept $=$	1000000	c:	0100000

Der Shift-And-Algorithmus

Suche mit dem Shift-And-Algorithmus nach $P =$ ababaca:

```
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: $\ll$

$accept =$ 1000000

mask:

a: 1010101

b: 0001010

c: 0100000

Der Shift-And-Algorithmus

Suche mit dem Shift-And-Algorithmus nach $P =$ ababaca:

```
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

$A =$ 1000000

op: | 0000001

$accept =$ 1000000

mask:

a: 1010101

b: 0001010

c: 0100000

Der Shift-And-Algorithmus

Suche mit dem Shift-And-Algorithmus nach $P =$ ababaca:

```
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

$A =$ 1000001

op: & 1010101

$accept =$ 1000000

mask:

a: 1010101

b: 0001010

c: 0100000

Der Shift-And-Algorithmus

Suche mit dem Shift-And-Algorithmus nach $P =$ ababaca:

```
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

Suche mit dem Shift-And-Algorithmus nach $P =$ ababaca:

```
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

Suche mit dem Shift-And-Algorithmus nach $P =$ ababaca:

```
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

Suche mit dem Shift-And-Algorithmus nach $P =$ ababaca:

```
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

Suche mit dem Shift-And-Algorithmus nach $P =$ ababaca:

```
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

Suche mit dem Shift-And-Algorithmus nach $P =$ ababaca:

```
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

Suche mit dem Shift-And-Algorithmus nach $P =$ ababaca:

```
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

Suche mit dem Shift-And-Algorithmus nach $P =$ ababaca:

```
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

Suche mit dem Shift-And-Algorithmus nach $P =$ ababaca:

```
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

Suche mit dem Shift-And-Algorithmus nach $P =$ ababaca:

```
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:

$accept =$ 1000000

mask:

a: 1010101

b: 0001010

c: 0100000

Der Shift-And-Algorithmus

```
1  def create_masks(P, S):
2      masks = {s: 0 for s in S}
3      for i, c in enumerate(P):
4          masks[c] |= (1 << i)
5      return masks
6
7  def shift_and(P, T):
8      m, n, A, S = len(P), len(T), 0, set(P+T)
9      accept = 1 << (m-1)
10     masks = create_masks(P, S)
11     for i in range(n):
12         A = ((A << 1) | 1) & masks[T[i]]
13         if A & accept:
14             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

UA RUHR

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	0	0	0	0	0	0	1
b	0	0	0	0	0	0	0
c	0	0	0	0	0	0	0

Der Shift-And-Algorithmus

UA RUHR

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

UA RUHR

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

UA RUHR

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

UA RUHR

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	0	0	1	0	1
b	0	0	0	1	0	1	0
c	0	0	0	0	0	0	0

Der Shift-And-Algorithmus

UA RUHR

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	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	0010101
b	0001010
c	0000000

Der Shift-And-Algorithmus

UA RUHR

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

UA RUHR

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

UA RUHR

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

UA RUHR

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

Zusammenfassung: Shift-And-Algorithmus

- Shift-And-Algorithmus nutzt Bitparallelismus für die Zustandsübergangsfunktion des NFA aus
- Laufzeit $\mathcal{O}(n)$ für $m \leq W$
- Schlanker und eleganter Code
- Algorithmus hardwarenah (C, Assembler) implementieren
- Optimierung: Inversion der Bitlogik
  Setzen des Startzustands entfällt; Shift-Or-Algorithmus