Script generated by TTT

Title: TÃ¤ubig: GAD (18.07.2013)

Date: Thu Jul 18 12:00:43 CEST 2013

Duration: 36:24 min

Pages: 24

Übersicht

10 Algorithmen zur Textsuche
- Naiver Algorithmus
- Knuth-Morris-Pratt-Algorithmus

Alphabet, Wörter, Wortlänge, Wortmengen

Definition

Ein Alphabet Σ ist eine endliche Menge von Symbolen.

Wörter über Σ sind endliche Folgen von Symbolen aus Σ (meist $w = w_0 \cdots w_{n-1}$ oder $w = w_1 \cdots w_n$).

Notation:

- $|w|$ Länge des Wortes w (Anzahl der Zeichen in w)
- ε leeres Wort (Wort der Länge 0)
- Σ^* Menge aller Wörter über Σ
- Σ^+ Menge aller Wörter der Länge ≥ 1 über Σ ($\Sigma^+ = \Sigma^* \setminus \{\varepsilon\}$)
- Σ^k Menge aller Wörter über Σ der Länge k

Präfix, Suffix, Teilwort

Definition

$[a : b] := \{n \in \mathbb{Z} \mid a \leq n \wedge n \leq b\}$ für $a, b \in \mathbb{Z}$

Sei $w = w_1 \cdots w_n$ ein Wort der Länge n über Σ, dann heißt

- w' Präfix von w, wenn $w' = w_1 \cdots w_\ell$ mit $\ell \in [0 : n]$
- w' Suffix von w, wenn $w' = w_\ell \cdots w_n$ mit $\ell \in [1 : n+1]$
- w' Teilwort von w, wenn $w' = w_i \cdots w_j$ mit $i, j \in [1 : n]$

Für $w' = w_i \cdots w_j$ mit $i > j$ soll gelten $w' = \epsilon$.

Das leere Wort ε ist also Präfix, Suffix und Teilwort eines jeden Wortes.

Textsuche

Problem:

Gegeben: Text $t \in \Sigma^*$; $|t| = n$;
Suchwort $s \in \Sigma^*$; $|s| = m \leq n$

Gesucht: $\exists\, i \in [0 : n - m]$ mit $t_i \cdots t_{i+m-1} = s$?
(bzw. alle solchen Positionen i)

Übersicht

10 Algorithmen ~~zur~~ Textsuche
- Naiver Algorithmus
- Knuth-Morris-Pratt-Algorithmus

Naiver Algorithmus: Beispiele

t	=	a	a	a	a	a	a	a	a	a	a
		a	a	a	b						
			a	a	a	b					
				a	a	a	b				

Naiver Algorithmus: Beispiele

t	=	a	a	a	a	a	a	a	a	a	a
		a	a	a	b						
			a	a	a	b					
				a	a	a	b				

t	=	a	a	b	a	a	b	a	a	b	a	a	b	a	a	b
		a	a	b	a	a	a									
			a	a	b	a	a	a								
				a	a	b	a	a	a							
					a	a	b	a	a	a						

Naiver Algorithmus: Implementation

```
bool NaiveSearch (char t[], int n, char s[], int m)
```

int $i := 0$, $j := 0$;
while $(i \leq n - m)$ **do**
  while $(t[i + j] = s[j])$ **do**
    $j{+}{+}$;
    if $(j = m)$ **then**
      return TRUE;
  $i{+}{+}$;
  $j := 0$;
return FALSE;

Analyse des naiven Algorithmus

- zähle Vergleiche von Zeichen,
- äußere Schleife wird $(n - m + 1)$-mal durchlaufen,
- die innere Schleife wird maximal m-mal durchlaufen.
- maximale Anzahl von Vergleichen: $(n \cancel{- m} \cancel{+ 1})m$,
- Laufzeit: $O(nm)$

Übersicht

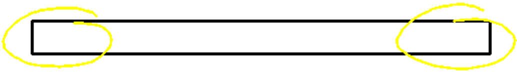
Bessere Idee

- frühere erfolgreiche Vergleiche von zwei Zeichen ausnutzen

- Idee:

  Suchwort so weit nach rechts verschieben, dass in dem Bereich von t, in dem bereits beim vorherigen Versuch erfolgreiche Zeichenvergleiche durchgeführt wurden, nun nach dem Verschieben auch wieder die Zeichen in diesem Bereich übereinstimmen

Rand und eigentlicher Rand



Definition

Ein Wort r heißt Rand eines Wortes w, wenn r sowohl Präfix als auch Suffix von w ist.

Bemerkung: Für jedes Wort w sind damit immer auch das leere Wort ε und w selbst Ränder von w.

Ein Rand r eines Wortes w heißt eigentlicher Rand, wenn $r \neq w$ und wenn es außer w selbst keinen längeren Rand gibt.

Rand und eigentlicher Rand

Beispiel

Das Wort $w =$ `aabaabaa` besitzt folgende Ränder:

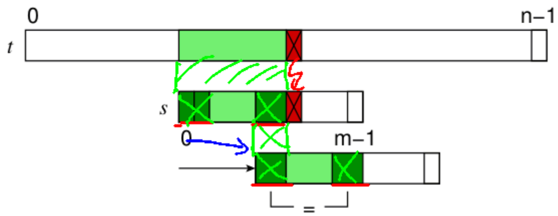
- ε
- `a`
- `aa`
- `aabaa`
- `aabaabaa` $= w$

Der eigentliche Rand ist `aabaa`.

Man beachte, dass sich bei der Darstellung eines Rands im Wort das entsprechende Präfix und Suffix in der Mitte des Worts überlappen können.

Shift-Idee

- Pattern s so verschieben, dass im bereits gematchten Bereich wieder Übereinstimmung herrscht.
- Dazu müssen überlappendes Präfix und Suffix dieses Bereichs übereinstimmen.



Shifts und sichere Shifts

Definition

Ein Verschiebung der Anfangsposition i des zu suchenden Wortes (d.h. eine Erhöhung des Index $i \to i'$) heißt Shift.

Ein Shift von $i \to i'$ heißt sicherer Shift, wenn s nicht als Teilwort von t an der Position $k \in [i+1 : i'-1]$ vorkommt, d.h. $s \neq t_k \cdots t_{k+m-1}$ für alle $k \in [i+1 : i'-1]$.

- Sinn eines sicheren Shifts: dass man beim Verschieben des Suchworts kein eventuell vorhandenes Vorkommen von s in t überspringt.

Sichere Shifts

Definition

Sei $\partial(s)$ der eigentliche Rand von s und sei

$$border[j] = \begin{cases} -1 & \text{für } j = 0 \\ |\partial(s_0 \cdots s_{j-1})| & \text{für } j \geq 1 \end{cases}$$

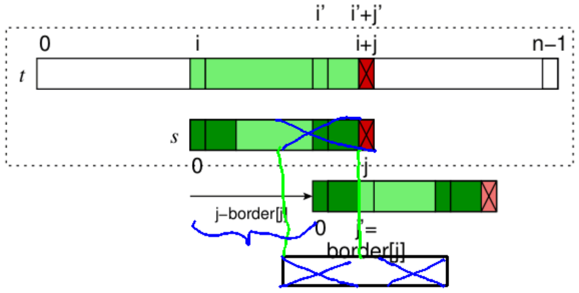
die Länge des eigentlichen Rands des Präfixes der Länge j.

Lemma

Ist das Präfix der Länge j gematcht (also gilt $s_k = t_{i+k}$ für alle $k \in [0 : j-1]$) und haben wir ein Mismatch an der nächsten Position j ($s_j \neq t_{i+j}$), dann ist der Shift $i \to i + j - border[j]$ sicher.

Sichere Shifts

Shift um $j - border[j]$



Sichere Shifts

Beweis.

- (siehe Skizze)

$$\begin{aligned} s_0 \cdots s_{j-1} &= t_i \cdots t_{i+j-1}, \\ s_j &\neq t_{i+j} \end{aligned}$$

- Der eigentliche Rand von $s_0 \cdots s_{j-1}$ hat die Länge $\mathrm{border}[j]$.
- Verschiebt man s um $j - \mathrm{border}[j]$ nach rechts, so liegt der linke Rand von $s_0 \cdots s_{j-1}$ nun genau da, wo vorher der rechte Rand lag, d.h. im Präfix/Suffix-Überlappungsbereich besteht Übereinstimmung zwischen Präfix, Suffix und Text.
- Da es keinen längeren Rand von $s_0 \cdots s_{j-1}$ als diesen gibt (außer $s_0 \cdots s_{j-1}$ selbst), ist dieser Shift sicher.

□

KMP-Algorithmus

```
bool KMP(char t[], int n, char s[], int m)
int border[m + 1];
compute_borders(int border[], int m, char s[]);
int i := 0, j := 0;
while i ≤ n − m do
    while t[i + j] = s[j] do
        j++;
        if j = m then
            return TRUE;
    i := i + (j − border[j]) ;          // Es gilt j − border[j] > 0
    j := max{0, border[j]};
return FALSE;
```

Laufzeit des KMP-Algorithmus: erfolglose Vergleiche

Nach erfolglosem Vergleich (Mismatch) wird $(i + j)$ nie kleiner:

- Seien dazu i und j die Werte vor einem erfolglosen Vergleich und i' und j' die Werte nach einem erfolglosen Vergleich.
- Wert vor dem Vergleich: $i + j$
- Wert nach dem Vergleich:
  $i' + j' = (i + j - border[j]) + (max\{0, border[j]\})$.
- Fallunterscheidung: $border[j]$ negativ oder nicht.
  - $border[j] < 0$, also $border[j] = -1$, dann muss $j = 0$ sein.
    Das bedeutet $i' + j' = i' + 0 = (i + 0 - (-1)) + 0 = i + 1$.
  - $border[j] \geq 0$, dann gilt $i' + j' = i + j$
- Also wird $i + j$ nach einem erfolglosen Vergleich nicht kleiner.

Laufzeit des KMP-Algorithmus

- Nach jedem erfolglosen Vergleich wird $i \in [0 : n - m]$ erhöht.
- Im Verlaufe des Algorithmus wird i nie erniedrigt wird.

$\Rightarrow$ maximal $n - m + 1$ erfolglose Vergleiche

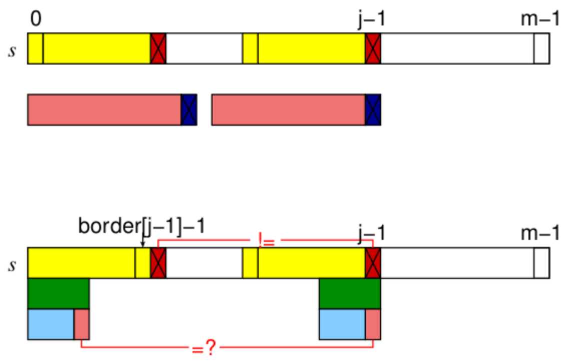
- Nach einem erfolgreichen Vergleich wird $i + j$ um 1 erhöht.
- maximal n erfolgreiche Vergleiche, da $i + j \in [0 : n - 1]$.

- Somit werden insgesamt maximal $2n - m + 1$ Vergleiche ausgeführt.

Berechnung der border-Tabelle

- In der $border[]$-Tabelle wird für jedes Präfix $s_0 \cdots s_{j-1}$ der Länge $j \in [0 : m]$ des Suchstrings s der Länge m gespeichert, wie groß dessen eigentlicher Rand ist.
- Initialisierung: $border[0] = -1$ und $border[1] = 0$
- Annahme: $border[0], \ldots, border[j-1]$ sind schon berechnet
- Ziel: Berechnung von $border[j]$
  (Länge des eigentlichen Randes von Präfix der Länge j)

Berechnung der border-Tabelle



Laufzeit des KMP-Algorithmus

Theorem

Der Algorithmus von Knuth, Morris und Pratt benötigt maximal $2n + m$ Vergleiche, um festzustellen, ob ein Muster s der Länge m in einem Text t der Länge n enthalten ist.

Der Algorithmus lässt sich leicht derart modifizieren, dass er alle Positionen der Vorkommen von s in t ausgibt, ohne dabei die asymptotische Laufzeit zu erhöhen.

Donald E. Knuth, James H. Morris, Jr. and Vaughan R. Pratt
Fast Pattern Matching in Strings

SIAM Journal on Computing 6(2):323–350, 1977.