1 Motivation

Probleme, in denen Texte verarbeitet werden müssen, tauchen nicht nur im Zusammenhang mit Textverarbeitungssystemen (z.B. den bekannten UNIX-Programmen grep, awk, emacs, vi, compress und zip) auf, sondern kommen in Bereichen wie der Theorie der formalen Sprachen, beim Übersetzerbau, bei der Implementierung von Programmiersprachen, bei Volltextdatenbanken, bei Multimedia Systemen, der Bildverarbeitung und der Bioinformatik vor - insbesondere die Bioinformatik hat in den letzten Jahren zu einem erneuerten Forschungsinteresse an Textalgorithmen geführt. Das wohl grundlegendste Problem in der Textverarbeitung ist das Suchproblem (Pattern Matching), in dem es darum geht, zu entscheiden ob, wo und wie oft ein Muster in einem Text vorkommt.

2 Knuth-Morris-Pratt Algorithmus

2.1 Definitionen

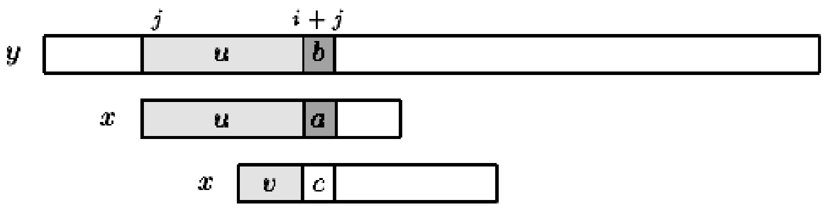
Sei im folgenden $x = x_0 \dots x_{n-1}$ eine Zeichenkette der Länge n.

- Ein *Präfix* einer Zeichenkette x ist eine Teilzeichenkette u mit $u = x_0 \dots x_b$, wobei $b <= n - 1$.
- Ein *Postfix* einer Zeichenkette x ist eine Teilzeichenkette u mit $u = x_b \dots x_{n-1}$, wobei $b >= 0$.
- Prä- oder Postfixe heissen *echt*, wenn ihre Länge kleiner als n ist.
- Eine Teilzeichenkette $r = x_0 \dots x_{b-1} = x_{n-b} \dots x_{n-1}$ heisst *Rand* von x. b wird als *Breite* des Randes bezeichnet. Ein Rand r von x lässt sich durch ein Textzeichen t *erweitern*, wenn rt Rand von xt ist.

2.2 Idee

Tritt beim Vergleich des Musters mit dem Text an einer Stelle des Musters ein Mismatch auf, so haben die vorangegangenen Zeichen des Musters mit dem Text übereingestimmt. Wie kann man diese Information nutzen, um das Muster möglichst weit zu verschieben ?

Betrachte folgendes Beispiel: Das Muster x ist an der j-ten Position des Textes y angelegt (siehe Skizze). Angenommen, das erste Mismatch tritt an der Stelle $x[i]$ bzw. $y[i+j]$ auf. Es gilt also $x[0 \dots i-1] = y[j \dots i+j-1] = u$.



Wie weit kann das Muster nun verschoben werden, ohne Gefahr zu laufen ein Vorkommen des Musters im Text zu übersehen ? Angenommen das bisher verglichene Teilmuster u besitzt ein Suffix v, das zugleich ein Prefix von x ist - d.h. v ist ein Rand. Wenn nun das Muster nun um $|u| - |v|$ nach rechts verschoben wird, wissen wir das die ersten $|v|$ Zeichen mit dem Text übereinstimmen, und vergleichen als erstes die Position $i + j$ des Textes mit der $|v| + 1$ Stelle des Musters. Beachte, dass auf diese Weise Vorkommen von x übersehen werden können, wenn v nicht der breiteste Rand von u ist !
Die Ränder für alle möglichen Präfixe u sollten natürlich vorberechnet werden. Der nächste Teil zeigt, wie das gemacht werden kann.

2.3 Preprocessing: Randberechnung

Satz 1 *Seien r, s Ränder einer Zeichenkette x, wobei $|r| < |s|$. Dann ist r ein Rand von s.*

Beweis: Als Rand von x ist r Präfix von x und damit, weil kürzer als s, auch ein *echtes* Präfix (.d.h ein Präfix kleinerer Länge) von s. Aber r ist auch Suffix von x und damit echtes Suffix von s. Also ist r Rand von s. □

Ist s der breiteste Rand von x, so ergibt sich der nächstschmalere Rand von r von x als breitester Rand von s usw.

In der Vorlaufphase wird ein Array b der Länge $m + 1$ berechnet. Der Eintrag $b[i]$ enthält für jedes Präfix der Länge i des Musters die Breite seines breitesten Randes $(i = 0, \dots, m)$. Das Präfix der Länge $i = 0$ hat keinen Rand; daher wird $b[0] = -1$ gesetzt.

Beispiel

```
Position   0 1 2 3 4 5 6
Pattern     a b a b a a
b[i]      -1 0 0 1 2 3 1
```

Sind die Werte $b[0], ..., b[i]$ bereits bekannt, so ergibt sich der Wert $b[i + 1]$, indem geprüft wird, ob sich ein Rand des Präfixes $p_0...p_{i-1}$ durch p_i erweitern lässt. Dies ist der Fall, wenn $p_b[i] = p_i$ ist. Die zu prüfenden Ränder ergeben sich nach obigem Satz in absteigender Breite aus den Werten $b[i], b[b[i]]$ usw. Der Preprocessing-Algorithmus enthält daher eine Schleife, die diese Werte durchläuft. Ein Rand der Breite j lässt sich durch p_i erweitern, wenn $p_j = p_i$ ist. Wenn nicht, wird $j = b[j]$ gesetzt und damit der nächstschmalere Rand geprüft. Die Schleife endet spätestens, wenn sich kein Rand erweitern lässt $(j = -1)$.

Nach Erhöhung von j durch die Anweisung $j + +$ enthält j in jedem Fall die Breite des breitesten Randes von $p_0 \dots p_i$. Dieser Wert wird in $b[i + 1]$ eingetragen (in $b[i]$ nach Erhöhung von i durch die Anweisung $i + +$).

Pseudocode

```
Procedure PrePro
{
   i = 0
   j = -1
   b[i] = j
   while i < m
   {
      while( j >= 0 ) AND ( p[i] != p[j] )
         j = b[j]
      i++
      j++
      b[i] = j
   }
}
```

Beispiel

```
Position  0 1 2 3 4 5 6 7 8 9 ...
Text      a b a b b a b a a ...

          a b a b a c
              a b a b a c
                  a b a b a c
                    a b a b a c
                        a b a b a c
```

2.4 Laufzeit

Die innere *while*-Schleife des Preprocessing-Algorithmus vermindert bei jedem Durchlauf den Wert von j um mindestens 1, denn es ist stets $b[j] < j$. Da die Schleife spätestens bei $j = -1$ abbricht, kann sie den Wert von j höchstens so oft vermindern, wie er vorher durch $j++$ erhöht wurde. Da $j++$ in der äußeren Schleife insgesamt genau m-mal ausgeführt wird, kann die Gesamtanzahl aller Durchläufe durch die $while$-Schleife auch nur maximal m betragen. Der Preprocessing-Algorithmus benötigt daher höchstens $O(m)$ Schritte.

Mit derselben Argumentation ist einzusehen, dass der Suchalgorithmus höchstens $O(n)$ Schritte benötigt.