

Hauptseminar im Wintersemester 06/07

Exact Text Matching

Inside Google-Algorithmen für Suchmaschinen

Jing ZHOU

04 . Dezember 2006

Betreuer: Dirk Nowotka

Inhaltsverzeichnis:

1. Einführung
2. Vorstellung von String-Matching
3. Knuth-Morris-Pratt Algorithmus (KMP)
4. Boyer-Morre Algorithmus (BM)
5. 2-Wege Algorithmus (Two-Way String-Matching)
 - 5.1 Grundlagen
 - 5.2 Algorithmus
 - 5.3 Aufwand
6. Crochemore-Perrin Algorithmus
7. Zusammenfassung
8. Quellenverzeichnis

1. Einführung

Das Hauptseminar Inside Google-Algorithmen für Suchmaschinen beschäftigt sich diese Arbeit mit der exakten Suche nach einem Wort, Begriff oder einer Wortgruppe, genannt Pattern in einem Text.

Die Text und Pattern bestehen aus einer beliebigen langen Folge von Zeichen. o.B.d.A ist Pattern kürzer als Text. „Exakt“ steht für die genaue Übereinstimmung des Patterns mit dem Textfragment in jedem einzelnen Zeichen.

Formal kann man das Problem der exakten Suche wie folgt beschreiben: Gegeben ist eine Zeichenreihe, der Text t und eine kürzere Zeichenreihe, der Suchbegriff oder das Pattern x . Gesucht sind alle Positionen i , an denen das Pattern x im Text t vorkommt. Die Suche ist durch den Vergleich einzelner Zeichen realisiert. Die Aufgabe bei der Entwicklung solcher Suchalgorithmen ist die Anzahl der Vergleiche zu reduzieren.

Der klassische Algorithmus besteht aus zwei Kategorien. In der ersten Kategorie ist das Pattern x als fest und der Text t ist als variable. Die umgekehrte Meinung wird in der zweiten Kategorie verwendet. In der ersten Kategorie gibt es die bekannte Algorithmus von Knuth-Morris-Pratt (KMP) und Boyer-Morre (BM). Die zweite Kategorie basiert auf „Suffix Tree“. Solche Algorithmus wurde studiert und bei vielen Leuten entwickelt.

Im Rahmen dieser Arbeit wird der 2-Wege Algorithmus (Two-Way String-Matching) vorgestellt, der zu der ersten Kategorie gehört. Dieser Algorithmus stellt eine Art Kompromiss zwischen KMP-Algorithmus und BM-Algorithmus. Er vereint die Vorteile der beiden Algorithmen. Die Zeichen werden teilweise von links nach rechts und teilweise von rechts nach links mit dem Text verglichen. Der 2-Wege Algorithmus verwendet ein bestimmtes Wissen über die Struktur des Wortes. Die Algorithmen von KMP und BM verwenden nur das Wissen über die Zusammensetzung des Patterns.

2. Vorstellung von String-Matching

Gegeben sind Zeichenreihe der Text t (die Länge ist n) und Pattern x (die Länge ist m). Gesucht sind alle Positionen i , an denen das Pattern x im Text t vorkommt.

o.B.d.A. $n > m$

Ergebnis der Suche ist ein Array:

$$P(x, t) = \{ k \in \mathbb{N} \mid 0 \leq k \leq n - m \text{ und } t[k+i] = x[i], 1 \leq i \leq m \}$$

Allgemeine Sting-Matching Algorithmen prüfen ob x an dem linken Rand von t auftaucht und wiederholen solche Prozess bei steigenden Positionen. Pattern x kann bei Verschiebung nach rechts so überwacht werden. Die Verschiebung muss so lange wie möglich, um die Zeit zu sparen.

In Beste-Situation wird nur ein Zeichen von Pattern x , der Länge m ist, betrachtet. Das braucht n/m Vergleiche.

In Schlechteste-Situation müssen alle Zeichen des Textes t mindesten einmal betrachtet werden.

Jetzt schauen wir wie der 2-Wege Algorithmus in Sting-Matching funktioniert. Bevor ich den 2-Wege Algorithmus vorstelle, mache ich eine kurze Vorstellung über den Knuth-Morris-Pratt Algorithmus und Boyer-Moore Algorithmus.

3. Knuth-Morris-Pratt Algorithmus (KMP)

In Knuth-Morris-Pratt Algorithmus, das Wort von Pattern x wird gegen dem Wort von Text t von links nach rechts prüft.

Wenn es eine Mismatch gibt, angenommen u ist das längste Präfix von Pattern x , die im Text t vorkommt. Dann wird die Verschiebung nach beiden der Periode von u und das Wort von Text t , an dem das Mismatch entsteht, nach rechts ausgeführt.

z.B:

	1	2	3	4	5	6	7	8	9	...
Text:	a	b	a	a	b	a	c	a	a	...
Pattern:	a	b	a	a	b	a	a			
Verschiebung:		a	b	a	a	b	a	a		

Das Präfix von Pattern x ist abaaba , wer Periode 3 ist. Das Mismatch entsteht an der 7-en Stelle. $7-3=4$. Dann wird Pattern x um 4-en Stelle nach rechts verschoben.

Deshalb wird die Funktion von Verschiebung , wer Bereite die Menge von Präfix des Patterns ist, vorher analysiert.

In einer Vorlaufphase (Preprocessing) analysiert der Algorithmus zunächst das Pattern und speichert Information über seine Struktur in einem Array der Länge m.

4. Boyer-Morre Algorithmus (BM)

In Boyer-Morre Algorithmus wird der Vergleich zwischen dem Pattern x und dem Text t von rechts nach links ausgeführt. Und das Pattern x wird nach wie vor von links nach rechts verschoben.

Wenn es ein Mismatch während dem Vergleich existiert, gibt es zwei Strategien, die Verschiebung nach rechts auszurechnen.

1-te Strategie : Character Strategie

Falls ein Mismatch beim Vergleich zwischen $x[i]$ und $t[j]$ existiert, wird das Pattern x nach rechts um k Positionen verschoben, so dass $x[i-k]=t[j]$. $x[i-k]$ soll das letzte Vorkommen des Zeichens $t[j]$ im Pattern x sein. Falls das letzte Vorkommen des Zeichens im Pattern x an der rechten Stellen von Mismatch , schieben wir Pattern x eins nach rechts.

z.B	1	2	3	4	5	6	7	8	9	...
Text	a	b	b	a	b	a	c	a	c	...
Pattern	b	c	b	a	a					
Verschiebung		b	c	b	a	a				

Wenn das Zeichen $t[j]$ nicht im Pattern x vorkommt, dann verschieben wir das Pattern x um m die Länge von Pattern nach rechts.

2-te Strategie : Suffix Strategie

Falls ein Mismatch beim Vergleich zwischen $x[i]$ und $t[j]$ gibt, wird das Pattern x bis zum nächsten Suffix, der beim bisher Vergleichen gleich ist, im Pattern verschoben.

z.B	1 2 3 4 5 6 7 8 9 ...
Text	a a b a b a c a b ...
Pattern	a b b a b
Verschiebung	a b b a b

Hier verschieben wir Pattern x um 3 Stellen nach rechts, weil ein Präfix von Pattern x (ab) mit einem Teil von übereinstimmenden Suffix (bab) übereinstimmt.

5. 2-Wege Algorithmus

5.1 Grundlagen

2-Wege Algorithmus ist eine Art Kompromiss zwischen Knuth-Morris-Pratt Algorithmus und Boyer-Moore Algorithmus.

Der Algorithmus benutzt einige Zeichen, die im folgenden vorgestellt werden:

Sei x das Wort von Alphabet A

Eine positive Zahl p nennen wir **Periode** von x , falls $x[i]=x[i+p]$ wenn beide Seiten der Gleichung schon definiert werden. Die kleinste Periode des Wortes x wird als $p(x)$ bezeichnet.

Hier können wir erfahren: $0 < p(x) \leq |x|$

z.B: für $x = babab$ ist die Periode $p(x)=2$

und für $x = abcde$ ist die Periode $p(x)=5$, weil an die undefinierte Teil man nicht denken braucht.

Sei x ein Wort und l eine Position in x mit $0 \leq l \leq |x|$. Eine Zahl $r \geq 1$ nennt man **lokale Periode** von x an der Position l , wenn $x[i]=x[i+r]$ wobei $l - r + 1 \leq i \leq l$.

Die lokale Periode von x an der Position l ist die kleinste lokale Periode an der Position l . Bezeichnen wir $r(x, l)$.

Man kann auch die lokale Periode r wie folgt definieren:

r ist eine lokale Periode von x an der Position l wenn und nur wenn ein Wort w der Länge r existiert sodass eine von vier folgenden Bedingungen erfüllt ist:

- (i) $x = x'wwx''$ mit $|x'w| = l$
- (ii) $x = x'wu$ mit $|x'w| = l$ und u Präfix von w
- (iii) $x = vwx''$ mit $|v| = l$ und v Suffix von w
- (iv) $x = vu$ mit $|v| = l$, v Suffix von w und u Präfix von w

z.B : 1 2 3 4 5 6 7 8 9 ...

d c a b c a b e f

hier ist $w=cab$, $l=4$, $r(x, l)=3$

(Critical Factorization Theorem) : Für jedes Wort x , es existiert mindestens eine Position l mit $0 \leq l < p(x)$

So dass $r(x, l)=p(x)$, eine solche Position l nennt man als eine **Kritische Position** von Wort x .

z.B : Sei $x=abaabaa$, Periode ist $p(x)=3$

für $l=2$, $r(x, l)=3$ erfüllt die (iii) Bedingung

1 2 3 4 5 6 7

a b a a b a a

hier $x = vwx''$, $w=aab$, $v=ab$ (Suffix von w), $x''=aa$. Aus der Definition wissen wir Position 2 ist eine kritische Position.

Es gibt noch zwei andere kritische Positionen : 4, 5:

1 2 3 4 5 6 7

a b a a b a a

hier $x = x'wwx''$, $w=baa$,

1 2 3 4 5 6 7
 a b a a b a a
 hier $x = x'wu$, $w = aab$, $u = aa$ (Präfix von w).

5.2 Algorithmus

Wie oben genannt, dieser Algorithmus vergleicht die Zeichen von Pattern x mit die Zeichen von Text t in beide Richtungen. Der Anfangspunkt von dem Prozess ist eine kritische Position l von x mit $l < p(x)$, wobei $p(x)$ Periode von x ist.

Wir bezeichnen P das Präfix von Pattern x der Länge l und S das Suffix von Pattern x . Dann ist $x = PS$.

Um den Vorkommen von Pattern x auf einer gegebenen Position vom Text t zu finden, teilt der Algorithmus die Suche in zwei Phasen.

In der ersten Phase wird nur Suffix S mit Text t verglichen. Die Zeichen von S werden von links nach rechts durchgesucht.

Falls während der ersten Phase gibt es ein Mismatch, gibt es keine zweite Phase und das Pattern x wird nach rechts verschoben. Die Verschiebung bringt die kritische Position rechts vom Buchstaben in dem Text t , der Mismatch verursacht.

Falls es kein Mismatch während der ersten Phase gibt, d.h ein Auftreten von Suffix S im Text t gefunden wurde, fängt die zweite Phase an. Jetzt wird P von Pattern x im Text t gesucht. Das Präfix P wird von rechts nach links durchgesucht wie in Boyer-Moore Algorithmus. Falls es ein Mismatch während dem Durchlauf gibt, wird das Pattern x um die Anzahl der Plätze, die gleich die Periode von Pattern x ist, verschoben.

Nach solcher Verschiebung wird irgendein Präfix von Pattern x mit dem Text t übereinstimmen. Dieses Präfix wird gespeichert um unnötige Vergleiche zu vermeiden.

Jetzt zeigen wir den Algorithmus in vereinfachter Art in Formel:

Angenommen, die Periode(x) vom Pattern x und eine kritische Position $l \leq p(x)$ sind vorher schon berechnet worden.

Der Algorithmus wird als eine Funktion von Berechnung der Menge $P(x, t)$ gezeigt. (Positionen von x in t).

Er benutzt vier lokale Variable i, j, s und pos .

Die Variable i und j werden als Cursor auf Pattern x verwendet, um die Übereinstimmung an beiden Seite von kritische Position zu zeigen.

Die Variable s wird verwendet, um ein Präfix von dem Pattern x zu speichern, das mit dem Text t auf der aktuellen Position übereinstimmt. Diese Position wird von der Variable pos gegeben. Die Variable s wird nach jedem Mismatch verändert.

```
function positions ( x , t )
// (p ist die Periode von x und l ist eine kritische Position mit  $l < p$  )
   $P := \phi$  ;  $pos := 0$  ;  $s := 0$  ,
1  while(  $pos + |x| \leq |t|$  ) do {
2     $i := \max( l, s ) + 1$  ;
3    while(  $i \leq |x|$  and  $x[i] = t[pos+i]$  ) do  $i := i + 1$  ;
    if (  $i \leq |x|$  ) then {
4       $pos := pos + \max( i - l , s - p + 1 )$  ;
       $s := 0$  ;
    } else {
5       $j := l$  ;
6      while (  $j > s$  and  $x[j] = t[pos+j]$  ) do  $j := j - 1$  ;
7      if (  $j \leq s$  ) then add  $pos$  to  $P$  ;
8       $pos := pos + p$ 
9       $s := |x| - p$  ;
    } end if
  } end while
  return (  $P$  ) ;
end function.
```

5.3 Aufwand

Der Aufwand hängt an dem benötigten Speicherplatz.

Weil man die Periode vorher schon berechnet hat, ist der Aufwand für den vorgestellten vereinfachten Algorithmus konstant. Da es verschiedene Methode zur Berechnung der Periode von Pattern x gibt, kann der Aufwand von Speicher verschiedene sein.

Im folgenden wir eine Methode vorgestellt, die bei Crochemore und Perrin erfunden wurde und benötigt konstanten Speicher.

6. Crochemore-Perrin Algorithmus

Sei $(\mathcal{P}[1\dots\alpha], \mathcal{P}[\alpha+1\dots m])$ ein kritische Faktorisierung von dem Pattern und sei $\rho \leq \max(\alpha, m-\alpha)$ der Länge von irgendeine Periode von dieser Faktorisierung. Dann ist ρ eine Multipl von π , die Länge von der Periode von Pattern ist.

Es existiert ein Konstant-Raum Linear-Zeit String-Matching Algorithmus, der 2-Wege-Algorithmus zeigt, dass 2-Wege-Algorithmus maximal $2n-m$ Vergleichen macht.

Crochemore-Perrin Algorithmus:

```
//  $\pi$  ist die Länge der Periode von Pattern  $\mathcal{P}[1\dots m]$ 
//  $(\mathcal{P}[1\dots\alpha], \mathcal{P}[\alpha+1\dots m])$  ist eine gegebene kritische Faktorisierung mit  $\alpha < \pi$ 
//  $\sigma$  ist die aktuelle Position vom Text, die das Pattern verbindet
//  $\theta$  ist die aktuelle Position, auf der wir vergleichen
 $\sigma = 1$ 
 $\theta = 1 + \alpha$ 
while  $\sigma \leq n - m + 1$  do // Try to match the pattern suffix
    while  $\theta < \sigma + m$  &&  $t[\theta] = \mathcal{P}[\theta - \sigma + 1]$  do // If there was a mismatch
         $\theta = \theta + 1$ 
    if  $\theta < \sigma + m$  then
         $\theta = \theta + 1$ 
         $\sigma = \theta - \alpha$ 
    else // Match the prefix
        if  $t[\sigma \dots \sigma + \alpha - 1] = \mathcal{P}[1 \dots \alpha]$  then
             $\sigma = \sigma + \pi$ 
        if  $\sigma + \alpha > \theta$  then
```

$$\theta = \sigma + \alpha$$

end

end

Es berechnet die Anzahl von Vergleichen, die bei dem Algorithmus gemacht wird. Es gibt maximal $n-\alpha$ Vergleichen, die in dem Durchlauf der Übereinstimmung von Pattern Suffix. Die zweite Phase macht maximal $n-m+\alpha$ Vergleichen (σ steigt mit π und $\alpha < \pi$).

Deshalb gibt es maximal $2n-m$ Vergleichen während dem ganzen Prozess.

7. Zusammenfassung

In dieser Arbeit wird 2-Wege Algorithmus vorgestellt, die in linearer Zeit läuft und braucht einen konstanten Speicher.

Es gibt zwei notwendige Vorläufe in 2-Wege Algorithmus, die Periode vom Pattern und die kritische Faktorisierung sind. 2-Wege Algorithmus speichert Text und Pattern in zwei Array. KMP kann ohne Speicherung von Text durchgeführt weil es wie ein deterministische Automat, der Text empfängt ohne zurück lesen. BM und 2-Wege Algorithmus brauchen zwei Array mit gleichen Speicherplätze. Eine ist für Pattern und andere wird als einen Puffer für den Text verwendet. Aber 2-Wege Algorithmus benutzt kein zusätzlichen Array für die Funktion von der Verschiebung wie KMP und BM.

8. Quellenverzeichnis

- [1] M. Crochemore und D. Perrin .
Two-way string-matching. (1991)
- [2] D.E. Knuth , J.H. Morris und V.R. Pratt .
Fast pattern matching in strings.(1977)
- [3] R.S. Boyer und J.S. Moore
A fast string searching algorithmus.(1977)
- [4] A.V. Aho
Pattern matching in Strings.(1980)
- [5] P. Weiner
Linear pattern matching algorithmus. (1972)
- [6] D. Breslauer und Z. Galil
Efficient comparison based string matching.(1993)