Knuth-Morris-Pratt-Algorithmus

PS: Stringmatching-Algorithmen in C
Dozent: Max Hadersbeck
Referentinnen: Joanna Rymarska, Alfina Druzhkova
Datum: 5.07.2006
Folien: www.cip.ifi.lmu.de/~droujkov

Agenda

- Historisches
- Merkmale des KMP-Algorithmus
- Die Struktur des Algorithmus
    - Preprocessing phase
    - Searching phase
- KMP-Algorithmus vs. Brute-Force-Algorithmus
- Zusammenfassung

Historisches

- Donald **Knuth** und sein Schüler Vaughan Ronald **Pratt** arbeiten an einem Fast-String-Searching-Algorithmus (Stanford-Universität, Kalifornien)
- James **Morris** (Berkeley-Universität, Kalifornien) löst das gleiche Problem unabhängig von den beiden
- **1977** wird KMP-Algorithmus gemeinsam veröffentlicht
- gleichzeitig wird **Boyer-Moore**-Algorithmus entwickelt

Merkmale des KMP-Algorithmus

- die Suche erfolgt von links nach rechts
- Komplexität der Vorverarbeitung: $O(m)$ bezüglich Laufzeit und Speicher ("space and time")
- Komplexität der eigentlichen Suche: $O(m+n)$ bzgl. Laufzeit unabhängig von der Alphabet-Größe
- höchstens $2n-1$ Vergleiche zwischen Zeichen des Musters und Textes während der Suche

Die Idee von KMP-Algorithmus

- **Naives Verfahren**: alles, was vor dem Mismatch war, wird wieder vergessen
- **KMP**: auf eine geschickte Weise können wir nach einem Mismatch im Text weiter nach vorne als nur 1 Zeichen springen

Struktur des Algorithmus

1. Phase: **Preprocessing**, die Präfix-Funktion
2. Phase: **Searching**, die eigentliche Suche

1. Phase: Präfix-Code

Pseudo-Code (Vgl. Sedgewick 1989, S. 247)

```
procedure initnext;
   var  i, j: integer;
   begin
   i:=1;  j:=0;  next[1]:=0;
  repeat
     if (j=0)  or  (p[i]=p[j])
          then begin i:=i+1;j:=j+1;next[i]:=j end
          else  begin j:=next[j]  end;
  until (j>M);
   end;
```

C-Code (Vgl. Lecroq/Charras 2004, S. 48)

```
void preKmp (char *x, int m, int kmpNext[])
   {
    int i, j;
    i = 0;
    j = kmpNext[0] = -1
     while (i < m) {
          while (j >-1 && x[i] != x[j])
               {
               j = kmpNext[j];
               }
          i++; j++;
          if (x[i] == x[j])) {
             kmpNext[i] = kmpNext[j];
          }
          else {
             kmpNext[i] = j;
          }
     }
}
```

2. Phase: Searching-Code

Pseudo-Code (Vgl. Sedgewick 1989, S. 246)

```
function kmpSearch: integer;
var i, j: integer;
begin
i:=1; j:=1;
repeat
     if (j=0) or (a[i]=p[j])
        then begin i:=i+1; j:=j+1 end
        else begin j:=next[j] end;
until (j>M) or (i>N);
if j>M then kmpSearch:=i-M
else kmpSearch:=i;
end;
```

C-Code (Vgl. Lecroq/Charras 2004, S. 48f)

```
void kmpSearch (char *x, int m, char *y, int n) {
    int i, j, kmpNext[XSIZE];
    preKmp (x, m, kmpNext);
    i = j = 0;
    while (j < n) {
        while (i > -1 && x[i] != y[j]) {
            i = kmpNext[i];
        }
        i++; j++;
        if (i >= m) {
            OUTPUT (j - i);
            i = kmpNext[i];
        }
    }
}
```

Aufbau von kmpSearch-Funktion

```
void kmpSearch (char *x, int m, char *y, int n)
 {
    int i, j, kmpNext [XSIZE];

    preKmp (x, m, kmpNext);

    i = j = 0;

    while (j < n)
    {
        while (i > -1  &&  x[i] != y[j])
          {
           i = kmpNext[i];
          }
        i++; j++;
        if (i >= m)
          {
           OUTPUT (j - i);
           i = kmpNext[i];
          }
     }
}
```

Muster-Array ***x** Text-Array ***y**

Muster-Länge **m** Text-Länge **n**

i Index von **x[ ]** **j** ist Index von **y [ ]**

i aktuelle Position von **x** **j** aktuelle Position von **y**

kmpNext[ ] ist die **Präfix-Tabelle** und hat die Form **(-1, 0, 0, -1** usw.)

preKmp ist die Präfix-Funktion

1. while solange Textende nicht erreicht ist

2. while solange **kmpNext**-Funktion anwenden, d.h. Muster **x** verschieben, bis Text und Muster an Stelle **i** und **j** übereinstimmen

in Text **y** und Muster **x** je eine Stelle weitergehen

if wenn das Ende des Musters erreicht ist:

die erste Position des Treffers ausgeben (**j-i**)

Muster verschieben

Beispiel 1 (Vgl. Wikipedia)

Text y [j]	a	b	a	b	a	b	c	b	a	b	a	b	c	a	b	a	b
Pattern x [i]	a	b	a	b	c	a	b	a	b								

Preprocessing Phase

Position	0	1	2	3	4	5	6	7	8
Muster	a	b	a	b	c	a	b	a	b
1. Präfix			a						
2. Präfix			a	b					
3. Präfix						a			
...									
letztes Präfix						a	b	a	b

i	0	1	2	3	4	5	6	7	8	
x [i]	a	b	a	b	c	a	b	a	b	
kmpNext	-1	0	0	1	2	0	1	2	3	4

Beispiel 1

Preprocessing Phase

i	0	1	2	3	4	5	6	7	8	
x [i]	a	b	a	b	c	a	b	a	b	
kmpNext	-1	0	0	1	2	0	1	2	3	4

Searching Phase

j	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
text y [j]	a	b	a	b	a	b	c	b	a	b	a	b	c	a	b	a	b
pattern x [ ]	a	b	a	b	c	a	b	a	b								

j	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
text y [j]	a	b	a	b	a	b	c	b	a	b	a	b	c	a	b	a	b
pattern x [ ]			a	b	a	b	c	a	b	a	b						

j	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
text y [j]	a	b	a	b	a	b	c	b	a	b	a	b	c	a	b	a	b
pattern x [ ]								a	b	a	b	c	a	b	a	b	

j	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
text y [j]	a	b	a	b	a	b	c	b	a	b	a	b	c	a	b	a	b
pattern x [ ]									a	b	a	b	c	a	b	a	b

Beispiel 2 (Vgl. Lecroq/Charras 2004, S. 49f.)

Text y [j]	G	C	A	T	C	G	C	A	G	A	G	A	G	T	A	T	A	C	A	G	T	A	C	G

Pattern x [i]	G	C	A	G	A	G	A	G

Preprocessing Phase

i	0	1	2	3	4	5	6	7	8
x [i]	G	C	A	G	A	G	A	G	
kmpNext[i]	-1	0	0	-1	1	-1	1	-1	1

Beispiel 2

Searching Phase

1. Versuch

j	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
y [j]	G	C	A	T	C	G	C	A	G	A	G	A	G	T	A	T	A	C	A	G	T	A	C
	1	2	3	4																			
x [i]	G	C	A	G	A	G	A	G															
i	0	1	2	3	4	5	6	7															

Shift by: 4 (i – kmpNext [i] = 3 - - 1 = 4)

2. Versuch

j	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
y [j]	G	C	A	T	C	G	C	A	G	A	G	A	G	T	A	T	A	C	A	G	T	A	C
					1																		
x [i]					G	C	A	G	A	G	A	G											
i					0	1	2	3	4	5	6	7											

Shift by: 1 (i – kmpNext [i]= 0 - - 1 = 1)

Beispiel 2

3. Versuch

j	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
y [j]	G	C	A	T	C	G	C	A	G	A	G	A	G	T	A	T	A	C	A	G	T	A	C	G
						1	2	3	4	5	6	7	8											
x [i]						G	C	A	G	A	G	A	G											
i						0	1	2	3	4	5	6	7											

Shift by: 7 (i – kmpNext [i] = 8 - 1 = 7)

4. Versuch

j	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
y [j]	G	C	A	T	C	G	C	A	G	A	G	A	G	T	A	T	A	C	A	G	T	A	C	G
														1										
x [i]													G	C	A	G	A	G	A	G				
i													0	1	2	3	4	5	6	7				

Shift by: 1 (i - kmpNext [i] = 1 - 0 = 1)

Beispiel 2

5. Versuch

j	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
y [j]	G	C	A	T	C	G	C	A	G	A	G	A	G	T	A	T	A	C	A	G	T	A	C	G
														1										
x [i]														G	C	A	G	A	G	A	G			
i														0	1	2	3	4	5	6	7			

Shift by: 1 (i – kmpNext [i]= 0 - - 1 = 1)

6. Versuch

j	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
y [j]	G	C	A	T	C	G	C	A	G	A	G	A	G	T	A	T	A	C	A	G	T	A	C	G
															1									
x [i]															G	C	A	G	A	G	A	G		
i															0	1	2	3	4	5	6	7		

Shift by: 1 (i – kmpNext [i] = 0 - - 1 = 1)

Beispiel 2

7. Versuch

j	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
y [j]	G	C	A	T	C	G	C	A	G	A	G	A	G	T	A	T	A	C	A	G	T	A	C	G
																1								
x [i]																G	C	A	G	A	G	A	G	
i																0	1	2	3	4	5	6	7	

Shift by: 1 (i - kmpNext [i] = 0 - - 1 = 1)

8. Versuch

j	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
y [j]	G	C	A	T	C	G	C	A	G	A	G	A	G	T	A	T	A	C	A	G	T	A	C	G
																	1							
x [i]																	G	C	A	G	A	G	A	G
i																	0	1	2	3	4	5	6	7

Shift by: 1 (i – kmpNext [i]= 0 - - 1 = 1)

Brute Force vs. KMP

Brute Force	***KMP***
Komplexität:	
– keine Vorverarbeitung	– Präfix-Funktion: $O(m)$
– Suche: $O((n-m+1)m)$	– Suche-Funktion: $O(m+n)$
Vergleiche:	
– für jede Startposition m Vergleiche	– je nach Inhalt des Musters kann das Muster (=die Startposition) um mehr als eine Stelle verschoben werden

Zusammenfassung

- der Text wird **genau einmal** durchlaufen
- der Algorithmus bewegt sich entweder im Text nach rechts, oder er bleibt im Text stehen und verschiebt das Muster
- Laufzeit $O(m+n)$
- höchstens $2n-1$ Vergleiche zwischen Zeichen des Musters und des Textes
- eignet sich nur für die Verarbeitung von Zeichenketten, bei denen sich ein Teil mehrmals wiederholt

  oder für große Texte

Literatur

- Knuth, D. / Morris, J.H., Jr. / Pratt, V. (1977): ***Fast pattern matching in strings.*** SIAM Journal of Computing 6, 2, 323-350
  (oder unter http://citeseer.ist.psu.edu/context/23820/0)
- Lecroq, Thierry / Charras, Christian (2004): ***Handbook of Exact String-Matching Algorithms***. - King's College London Publications.
  (oder unter http://www-igm.univ-mlv.fr/~lecroq/string/string.pdf
  http://www-igm.univ-mlv.fr/~lecroq/string/node8.html
  http://www-igm.univ-mlv.fr/~lecroq/string/examples/exp8.html)
- Orwant, Jon / Hietaniemi, Jarkko / Macdonald, John (2000): ***Algorithmen mit Perl***. - O'Reilly.
- Sedgewick, Robert (1989): ***Algorithms in C***. - Addison-Wesley.
- Corman, T. / Leiserson, C.E. / Rivest, R.L. / Stein C. (2001): ***Introduction to Algorithms***. (Kapitel 32.4, The Knuth-Morris-Pratt algorithm). - Second edition, 923-931, MIT Press and McGraw-Hill.
- Wikipedia: http://de.wikipedia.org/wiki/Algorithmus_von_Knuth-Morris-Pratt
  (Zugriff am 27.06.2006)

Unser Programm

http://www.cip.ifi.lmu.de/~droujkov/referate/kmp.c