Technische Universität München
Institut für Informatik
Lehrstuhl für Effiziente Algorithmen
Volker Heun

SS 2000
Lösungsblatt 1
25. Mai 2000

Algorithmen der Bioinformatik

Abgabetermin: 16. Mai

Aufgabe 1

Bestimmen Sie für die Zeichenreihe

$$B = 01011010100$$

die Tabelle $border[]$ für den Knuth-Morris-Pratt-Algorithmus.

Lösungsvorschlag

Laut Definition ist

$$border[i] = \begin{cases} \max\{j \mid 0 < j < i-1, b_{j-i}b_{i-j+1}\cdots b_{i-1} = b_1 b_2 \cdots b_j\} & \text{falls es ex.} \\ 0 & \text{sonst} \end{cases}$$

Dabei ist $border[i]$ intuitiv die Länge des längsten Suffixes von $b_1 \dots b_{i-1}$, das zugleich ein Präfix von B ist. Ferner setzt man $border[1] = -1$ und erhält $border[2] = 0$ automatisch.

Man kann die Tabelle in linearer Zeit durch folgende Rekursion berechnen:

- **Induktionsbeginn:** $border[2] = 0$, siehe oben
- **Induktionsschritt:** Seien die Einträge $1 \dots i-1$ bereits berechnet. Falls $b_{i-1} = b_{border[i-1]+1}$, so erweitert das nächste Zeichen das bisherige Präfix, und es gilt: $border[i] = border[i-1]+1$. Ansonsten liegt dieselbe Situation wie bei einem mismatch an Zeichen $b_{border[i-1]+1}$ vor, so daß gilt: $border[i] = border[border[i-1]+1]+1$, falls $b_{i-1} = b_{border[border[i-1]+1]+1}$. Ist letzteres auch nicht der Fall, so fahren wir rekursiv fort, bis ein match eintritt.

i	1	2	3	4	5	6	7	8	9	10	11
b_i	0	1	0	1	1	0	1	0	1	0	0
$border[i]$	-1	0	0	1	2	0	1	2	3	4	3

Aufgabe 2

Modifizieren Sie den KMP-Algorithmus so, daß er für jedes Vorkommen einer Zeichenreihe s in t die Position des Vorkommens ausgibt.

Lösungsvorschlag

Beim ersten gefundenen Vorkommen lassen wir die Funktion nicht mit "`return true`" terminieren, sondern lassen die gefundene Position ausgeben und so weiterarbeiten wie gewohnt. Eine Implementierung (in `C`) könnte etwa wie folgt aussehen.

```
int kmp (char *t, char *s)
{
    int treffer = 0;
    int n = strlen (t);
    int m = strlen (s);
    int *border = compute_borders (s);
    int i = 1, j = 1;

    while (i <= n-m)
    {
        while (t[i+j] == s[j])
        {
            if (++j == m)
            {
                /* Komplette Laenge von s okay!
                   Hier stand vorher ein "return TRUE" */

                ++treffer;
                printf ("  Treffer ab t[%d]\n", i);
                break;  /* Verl. innere while-Schleife */
            }
        }

        i += (j - border[j]);   /* shift */

        // j = max{1, border[j]}
        j = (border[j] > 0) ? border[j] : 1;
    }

    return treffer;
}
```

Hier wird außerdem am Ende noch die Anzahl der Treffer zurückgegeben.

Aufgabe 3

Geben Sie eine Optimierung des KMP-Algorithmus für den Fall an, daß das vorliegende Alphabet binär ist. Definieren Sie hierzu die $border[]$-Tabelle geeignet um und modifizieren Sie den KMP-Algorithmus. Wie sieht die Tabelle für das Beispiel in Aufgabe 1 aus?

Lösungsvorschlag

Bei einem binären Alphabet können wir zusäzlich zu den Informationen in der bisherigen $border$-Tabelle noch folgendes nutzen: Liegt ein mismatch bei Zeichen b_i vor, so ist eindeutig klar, daß das entsprechende Zeichen im Text $\overline{b_i}$ ist.

Daher kann man definieren:

$border[i]$ = Länge des längsten Präfixes von B, das Suffix von $b_1 \ldots b_{i-1}\overline{b_i}$ ist, d.h. $border[i] = \max\{j \mid 0/leqj \leq i-1, b_1 \ldots b_j = b_{i-j+1} \ldots b_{i-1}\overline{b_i}\}$.

Insbesondere ist $border[1] = 0$ und $border[2] = 1$, falls $b_1 == \overline{b_2}$ und 0 sonst.

Für den KMP-Algorithmus ergibt sich folgende Modifikation: Falls zwischen Zeichen t_k und b_i ein mismatch auftritt, so braucht t_k nicht mit $b_{border[i]+1}$ verglichen werden. Stattdessen kann sofort t_{k+1} mit $b_{border[i]+1}$ verglichen werden, so daß im Text $T = t_1 \ldots t_n$ in jeder Iteration ein Fortschritt gemacht wird.

Im Beispiel:

i	1	2	3	4	5	6	7	8	9	10	11
b_i	0	1	0	1	1	0	1	0	1	0	0
$border[i]$	0	1	0	1	3	0	1	0	1	5	4