Algorithmen und Datenstrukturen (Th. Ottmann und P. Widmayer)

Folien: Knuth-Morris-Pratt

Autor: Sven Schuierer

Institut für Informatik

Georges-Köhler-Allee

Albert-Ludwigs-Universität Freiburg

1 Suchen in Texten

Anwendungen:
dynamisch: Texteditoren,
Symbolmanipulation
statisch: Literaturdatenbanken,
Bibliothekssysteme,
Gen-Datenbanken,
Internet

Datentyp:
string: array of character,
file of character

Operationen: Länge,
i-tes Zeichen,
Verkettung, ...

Brute Force Suche

Brute-Force

Input: Text T und Pattern P

Output: Verschiebungen für alle Vorkommen von P in T

```
1 n := length(T); m := length(P)
2 for i := 0 to n − m do
3     j := 1
4     while j ≤ m and T[i + j] = P[j] do
5         j := j + 1
      end while;
6     if j = m + 1
7       then gebe Verschiebung i aus
```

Brute Force Suche: Next

next

Input: Indizes i und j mit $P[1..j] = T[i-j..i-1]$

Output: Größter Index $j' < j$ mit
$P[1..j'] = T[i-j'..i-1]$

```
1 repeat
2    j := j − 1
3    k := 1
4    while k ≤ j and T[(i − 1) − j + k] = P[k] do
5        k := k + 1
     end while;
6 until k = j + 1;
7 return (j)
```

Brute Force Suche: Alternative

Brute-Force-2

Input: Text T und Pattern P

Output: Verschiebungen für alle Vorkommen von P in T

```
1 n := length(T); m := length(P)
2 j := 0
3 for i := 0 to n − 1 do
4     while j > 0 and T[i] ≠ P[j + 1] do
5         j := next(i, j)
      end while;
6     if T[i] = P[j + 1] then j := j + 1
7     if j = m
8       then gebe Verschiebung i − m aus
9            j := next(i,j)
```

2 Knuth-Morris-Pratt

KMP

Input: Text T und Pattern P

Output: Verschiebungen für alle Vorkommen von P in T

```
1  n := length(T); m := length(P)
2  berechne next-array
3  j := 0
4  for i := 1 to n do
5      while j > 0 and T[i] ≠ P[j + 1] do
6          j := next[j]
       end while;
7      if T[i] = P[j + 1] then j := j + 1
8      if j = m
9        then gebe Verschiebung i − m aus
10             j := next[j]
```

Invariante

for-Schleife: $P[1..j] = T[i-j..i-1]$ und
$P[1..k] \neq T[i-k..i-1]$ für $j < k < m$

Invariante while-Schleife (auch für $j = 0$):
$P[1..j] = T[i-j..i-1]$ und $P[1..k+1] \neq T[i-k..i]$
für $j < k < m$ ($k < m$, da wir $P[1..k+1]$ betrachten

Beispiel $next$-Array

$next[i] =$ Länge des längsten Präfixes von P, das zugleich ein echter Suffix von $P[1..i]$ ist.

Muster: `abrakadabra`

$next[i]$	betracht. Präf. $P[1..i]$	echte Suffixe	län. Suf.	Län.
$next[1]$	a	$\emptyset$	$\emptyset$	0
$next[2]$	ab	b	$\emptyset$	0
$next[3]$	abr	r, br	$\emptyset$	0
$next[4]$	abra	a, ra, bra	a	1
$next[5]$	abrak	k, ak, rak, brak	$\emptyset$	0
$next[6]$	abraka	a, ka, aka, ...	a	1
$next[7]$	abrakad	d, ad, kad, ...	$\emptyset$	0
$next[8]$	abrakada	a, da, ada, ...	a	1
$next[9]$	abrakadab	b, ab, dab, ...	ab	2
$next[10]$	abrakadabr	r, br, abr, dabr, ...	abr	3
$next[11]$	abrakadabra	a, ra, bra, abra, ...	abra	4

Tabelle

Verlangt man noch, daß $P[j+1] \neq P[next[j]]$, so ergibt sich folgende Tabelle $next'$

$$next'[j] = \begin{cases} next[j] & \text{if } P[j+1] \neq P[next[j]] \\ next'[next[j]] & \text{if } P[j+1] = P[next[j]] \end{cases}$$

Berechnung des $next$-Arrays

KMP-$next$

Input: Pattern P

Output: $next$-Array für P

```
1 m := length(P)
2 next[1] := 0;
3 j := 0
4 for i := 2 to m do
5     while j > 0 and P[i] ≠ P[j + 1] do
6         j := next[j]
      end while;
7     if P[i] = P[j + 1] then j := j + 1
8     next[i] := j
```

Invariante for-Schleife: $j = next[i-1]$ und
$next[1], \ldots, next[i-1]$ korrekt berechnet

Invariante while-Schleife (auch für $j = 0$):
$P[1..j] = P[i-j..i-1]$ und
$P[1..k+1] \neq P[i-k..i]$ für $j < k < m$ ($k < m$, da wir $P[1..k+1]$ betrachten

Berechnung des $next'$-Arrays

KMP-$next$

Input: Pattern P

Output: $next'$-Array für P

```
1  m := length(P)
2  next'[1] := 0;
3  j := 0
4  for i := 2 to m do
5      { j = next[i] }
6      while j > 0 and P[i] ≠ P[j + 1] do
7          j := next'[j]
       end while;
8      if P[i] = P[j + 1] then j := j + 1
9      if P[i] = P[j + 1]
10       then next'[i] := j
11       else next'[i] := next'[j]
```

Beispiel KMP

```
a b r a k a d a b r a b r a b a b r a k ···
| | | | | | | | | | |
a b r a k a d a b r a
```
$next[11] = 4$

```
a b r a k a d a b r a b r a b a b r a k ···
              - - - - ⁄
              a b r a k
```
$next[4] = 1$

```
a b r a k a d a b r a b r a b a b r a k ···
                  - | | | ⁄
                  a b r a k
```
$next[4] = 1$

```
a b r a k a d a b r a b r a b a b r a k ···
                        - | ⁄
                        a b r a k
```
$next[2] = 0$

```
a b r a k a d a b r a b r a b a b r a k ···
                            | | | | |
                            a b r a k
```