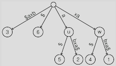
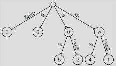


Datenstrukturen und Effiziente Algorithmen

Vorlesung *Datenstrukturen und Effiziente Algorithmen* im WS 18/19

Marc Hellmuth
Institut für Mathematik und Informatik
Universität Greifswald



Exact String Matching Problem

Definition 1

Let P and T be strings, called *pattern* and *text*, respectively, and let T be longer than P. The exact matching problem is to find all occurrences, if any, of pattern P in text T.

Instead of the simple linear matching algorithm (using the Z-algorithm), we give a further algorithm (Boyer-Moore) that typically runs in sub-linear time.

Important Rules

“Right-Left-Scan”

To check the occurence of P in (some part of) T we compare the characters of P in T from *left to right*.

Important Rules

Definition 2

For each $x \in \Sigma$ let $R(x)$ be the position of the right-most occurence of character x in P and put $R(x) = 0$, if x does not occur in P.

Exercise: $R(x)$ can be computed in $O(|P|)$ time

"Bad Character (Shift) Rule"

Suppose for a particular alignment of P against T, the right-most $n - i$ characters of P match their counterparts in T, but the next character to the left, $P(i)$, mismatches with its counterpart, say in position k of T.

The bad character rule says that P should be shifted right by $\max\{1, i - R(T(k))\}$ places.

That is, if the right-most occurrence in P of character $T(k)$ is in position $j < i$ (including the possibility that $j = 0$), then shift P so that character j of P is below character k of T. Otherwise, shift P by one position.

Important Rules

Definition 3

For each $x \in \Sigma$ let $R(x)$ be the position of the right-most occurence of character x in P and put $R(x) = 0$, if x does not occur in P.

Exercise: $R(x)$ can be computed in $O(|P|)$ time

"(Strong) Good Suffix Rule"

Suppose for a given alignment of P and T, a substring t of T matches a suffix of P, but a mismatch occurs at the next comparison to the left.

Then find, if it exists, the right-most copy t' of t in P such that

- t' is not a suffix of P and
- the character to the left of t' in P differs from the character to the left of t in P.

Shift P to the right so that substring t' in P is below substring t in T

If t' does not exist,then shift the left end of P past the left end of t in T by the least amount so that a prefix of the shifted pattern matches a suffix of t in T.

If no such shift is possible, then shift P by n places to the right.

If an occurrence of P is found, then shift P by the least amount so that a proper prefix of the shifted P matches a suffix of the occurrence of P in T.

If no such shift is possible, then shift P by n places, that is, shift P past t in T.

Important Rules

Theorem 4

The use of the good suffix rule never shifts P past an occurrence in T.

Proof.

(chalk board) □

Important Rules

Definition 5

For each i, $L'(i)$ is the largest position less than n such that string $P[i..n]$ matches a suffix of $P[1..L'(i)]$ and such that the character preceding that suffix is not equal to $P(i-1)$.

$L'(i) = 0$, if there is no position satisfying the conditions.

Let $l'(i)$ denote the length of the largest suffix of $P[i..n]$ that is also a prefix of P, if one exists. If none exists, then let $l'(i)$ be zero.

Compute $L'(i)$, $l'(i)$

Definition 6

For string $S = s_1 s_2 \dots s_n$, the inverse S^{-1} of S is $S^{-1} = s_n s_{n-2} \dots s_1$

Definition 7

For string P, $N_j(P)$ is the length of the longest suffix of the substring $P[1..j]$ that is also a suffix of the full string P.

$$N_j(P) = Z_{|P|-j+1}(P^{-1})$$

Thus, the $N_j(P)$ values can be computed $O(|P|)$ time using the Z-Algorithm on P^{-1}.

Theorem 8

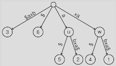
$L'(i)$ is the largest index $j < n$ such that $N_j(P) = |P[i..n]| = n - i + 1$.

Compute $L'(i)$, $l'(i)$

Compute $L'(i)$, $l'(i)$

Require: All $N_j(P)$ are computed using Z-alg on P^{-1}.

```
n=|P|
for i = 1 to n do L'(i) ← 0
for j = 1 to n − 1 do
    i ← n − N_j(P) + 1
    L'(i) ← j
j ← 0
for i = 1 to n do
    if N_j(P) = i then j ← i
    l'(n − i + 1) ← j
```



Boyer-Moore Algorithm

Boyer-Moore Algorithm

Require: Pattern P and Text T

$n \leftarrow |P|$, $m \leftarrow |T|$
Compute $L'(i)$, $l'(i)$, $R(x)$ for P
$k \leftarrow n$
while $k \leq m$ **do**
    $i \leftarrow n$, $h \leftarrow k$
    while $i > 0$ and $P(i) = T(h)$ **do**
        $i \leftarrow i - 1$, $h \leftarrow h - 1$
    if $i = 0$ **then** // P occurs in T
        output occurrence of P starting at position $k - n + 1$ in T
        $k \leftarrow k + n - l'(2)$
    else// P does not occur in T
        $shift_{BC} \leftarrow \max\{1, i - R(T(h))\}$
        if L'(i+1) = 0 **then** shift$_{GS} \leftarrow n - l'(i+1)$
        else $shift_{GS} \leftarrow n - L'(i+1)$
    $k \leftarrow k + \max\{shift_{BC}, shift_{GS}\}$