Algorithmics

Sebastian Iwanowski
FH Wedel

5. String Matching

Algorithmics 5

String Matching

Task: Given a text T = $\{t_1,\ldots,t_n\}$ with n literals and a pattern P = $\{p_1,\ldots,p_m\}$ with m literals:
Find the starting positions where P occurs in T.

naive algorithm: needs O(nm) time

Algorithm of Knuth-Morris-Pratt: needs O(n) time

Def.: P_q denotes the prefix of P consisting of the first q literals. (P_q = P[1],…,P[q])

Def.: The prefix function $\pi: \mathbb{N}\backslash\{0\} \rightarrow \mathbb{N}$ for the pattern P is defined as:
$\pi(q) = k \Leftrightarrow$ k is the length of the longest strict prefix of P_q (*strict* means: k < q)
which is also a Suffix of P_q

General method of the KMP algorithm:

For each q ≤ m, compute the value π(q) of the prefix function and store it.
Then scan T in only one iteration and shift P at any mismatch in pattern position q+1 by q - π(q). This does not omit any valid match.

In class: Why is this correct?

References:

Alt, Kap. 4.8
Cormen, ch. 32 (String matching), esp. 32.4 (KMP)

Algorithmics 5

String Matching

Algorithm of Knuth-Morris-Pratt: needs O(n) time

Implementation of main procedure:

```
i := 1; q := 0;
while i ≤ n do
{
    while (q>0) and (T[i] ≠ P[q+1])
       q := π (q);
    if T[i] = P[q+1] then q := q+1;
    if q = m
       then
       {
            print („Matching at position “, i-m);
            q := π (q);
       }
    i := i+1;
}
```

Invariant: *q=0:* *no prefix of P coincides at a suffix of T ending at i*
q>0 corresponds to the maximum index ≤ i s.t. (T[i-q+1],…,T[i]) coincides with (P[1],…,P[q])

In class: Why is this algorithm correct?

Home work:
Why does this algorithm need O(n) time?

References:

Alt, Kap. 4.8
Cormen, ch. 32 (String matching), esp. 32.4 (KMP)

Algorithmics 5

String Matching

Proof of the invariant:

Mathematical induction using i (where i is the loop counter which is increased at the end of the loop)**:**

i = 1:

Then q has an initial value of 0. => Inner loop is not executed.
q = 1 if comparison with P(1) is true and 0 otherwise which is the statement of the invariant.

Assume the invariant holds for i and consider i+1:

If T(i+1) = P(q+1) before the inner loop,
then the maximum prefix of P until i+1 has length maximum length of P until i plus 1
which is by assumption q + 1. So the assignment of the new q is correct.
If T(i+1) ≠ P(q+1) before the inner loop and q = 0,
then q remains 0 which is correct.
If T(i+1) ≠ P(q+1) before the inner loop and q > 0,
then (T[i-q+1],…,T[i]) coincides with (P[1],…,P[q]) by inductive assumption for i.
If there is no q‘ < q where T(i+1) = P(q‘+1), then no prefix coincides at a suffix of T ending at i+1.
The inner while loop will then eventually set q to 0 which is the correct invariant for i+1.
If for some q‘ < q, T(i+1) = P(q‘+1), and this is the end of a matched prefix,
such that T[i-q‘+1],…,T[i]) coincides with (P[1],…,P[q‘]), then by the above assumption
(P[1],…,P[q‘]) also coincides with (P[q-q‘+1],…,P[q]) which means that q‘ ≤ π(q).
Thus, it is ok if q is reduced accordingly in the inner while loop.
If q‘ = π(q), it is clearly the maximum and thus the q defined in the invariant.
Otherwise, q‘ < π(q) and will be found in a later loop iteration.

References:

Alt, Kap. 4.8
Cormen, ch. 32 (String matching), esp. 32.4 (KMP)

Algorithmics 5

String Matching

Algorithm of Knuth-Morris-Pratt: needs O(n) time

Implementation of prefix function (according to Cormen/Alt): needs O(m) time

```
π(1) := 0;
i := 2; q := 0;
while i ≤ m do
{
    while (q>0) and (P[i]≠P[q+1]) do
        q := π(q);
    if P[i]=P[q+1] then q := q+1;
    π(i) := q;
    i := i+1;
}
```

Invariant: *q=0: no strict prefix of P coincides at a suffix of P ending at i*
q>0 corresponds to the maximum index < i s.t. (P[i-q+1],…,P[i]) coincides with (P[1],…,P[q])

Home work: Why is this algorithm correct?

In class:
Why does this algorithm need O(m) time?

References:

Alt, Kap. 4.8
Cormen, ch. 32 (String matching), esp. 32.4 (KMP)