A correctness proof of the Knuth-Morris-Pratt string-matching algorithm

ASP

May 6, 2009

1 Introduction

This note contains a proof of the correctness of the KMP string-matching algorithm – it is meant as an informal supplement to the exposition on the KMP algorithm found in CLRS [Sec. 32.4]. The fundamental idea of the KMP string-matching algorithm is to use the prefix function of the pattern for which we are searching. The *prefix function* for a pattern $P[1 \ldots m]$ is the function $\pi : \{1, \ldots, m\} \to \{0, 1, \ldots, m-1\}$ such that

$$\pi[q] = \max\{k \mid k < q \text{ and } P_k \sqsupset P_q\}$$

That is, $\pi[q]$ is the length of the longest prefix of P that is a *proper* suffix of P_q. The reader is recommended to skim through [CLRS, p. 923–926] to get familiar with the notion of a prefix function.

The KMP algorithm consists of two routines: KMP-MATCHER and COMPUTE-PREFIX-FUNCTION. The correctness of KMP-MATCHER is quite evident once the notion of a prefix function is understood. The reader is recommended to skip the proof of the correctness of KMP-MATCHER on a first reading.

2 Correctness of KMP-MATCHER

Theorem 2.1. *For any given finite pattern P and finite text T, the* KMP-MATCHER *(Algorithm 1) correctly reports each occurence of P in T.*

Proof. Let's just begin by convincing ourselves that the algorithm is reasonable. It clearly finishes in a finite number of steps. The prefix function π is only defined on $\{1, 2, \ldots, m\}$. Does the algorithm ever try to evaluate the function π on integers not contained in the set $\{1, 2, \ldots, m\}$? No[1].

We prove that the following statement is a loop-invariant for the for-loop (lines 5-16).

---

[1]By definition, $0 \le \pi[q] < q$. Hence, the value of the variable q decreases to non-negative integer-value in line 7 and increases by one unit in line 10. If the variable attains the value m, then q is assigned the value $\pi[m] \in \{0, \ldots, m-1\}$ in line 12. Thus, the value of q is also an integer in the set $\{0, 1, \ldots, m\}$, and, just before executing line 7, we have $q \in \{1, \ldots, m\}$.

(*) After each execution of the body of the for-loop, the value of the variable q is equal to the largest integer $j \in \{0, \dots, m-1\}$ for which P_j is a suffix of T_i.

Claim 2.2. *The statement (*) is a loop-invariant for the for-loop (lines 5-16).*

Argument. We apply induction on the value of loop-variable i. If $i = 1$, then (*) hold. Suppose that s is some integer from the set $\{2, \dots, m\}$, and suppose that (*) hold for $i = s - 1$. Let q' denote the value of the variable q just prior to the execution of the body of the for-loop with $i = s$. Then $P_{q'} \sqsupset T_{s-1}$.

Now let's just start by seeing what goes on in the lines 12-15: If line 13 is executed, then $q = m$ in line 12 and P_m is a suffix of T_s. However, we are looking for the largest integer $j \in \{0, \dots, m-1\}$ such that P_j is a suffix of T_s. It follows directly from the definition of the prefix function that the value of j is $\pi[m]$, and so line 12 assigns the correcet value to the variable q.

Right, now let's get back to the begining of the body of the for-loop. The value of j (when $i = s$) is, by the maximality condition on j and the induction hypothesis, at most $1 + q' \le 1 + (m - 1)$. At the first execution of line 6 (with $i = s$), we have $q = q'$. If $P[q' + 1] = T[s]$, then $P_{q'+1} \sqsupset T_s$ and statement (*) hold with $j = q' + 1$. If $q' = 0$, then (*) hold with $j \in \{0, 1\}$.

Suppose that $P[q' + 1] \ne T[s]$, then $P_{q'+1}$ is not a prefix of T_s and so $j < q' + 1$. Now the value of j will have the property that P_{j-1} is a proper suffix of T_{s-1}. Since $\pi[q'] + 1$ is the largest integer $< q' + 1$ with this property, line 7 assigns the value $\pi[q']$ to q. The algorithm will eventually breakout of the while-loop, and when it does, the variable q will be assigned (1) the greatest *positive* integer $\le q'$ such $P_q \sqsupset T_{s-1}$ and $P[q+1] = T[s]$, or (2) if no such positive integer exist, $q = 0$. In both cases (1) and (2), the algorithm insures that (*) hold after the execution of the last line of the body of the for-loop with $i = s$.

$\diamond$

Now, suppose that P occurs in T with shift $r - m$. Then $P_{m-1} \sqsupset T_{r-1}$. Thus, it follows from the loop-invariant, that just after the execution of the body of the for-loop with[2] $i = r - 1$ we have $q = m - 1$. Since P occurs in T with shift $r - m$, we have $P[m] = T[r]$, and so during the execution of the body of the for-loop with $i = r$, the algorithm doesn't execution line 7, but executes line 10 and line 13, that is, it reports that P occurs with shift $i - m$ $(= r - m)$.

Conversely, suppose that the algorithm reports that P occurs with shift $s - m$. Let q' denote the value of the variable q just after the execution of the body of the for-loop with $i = s - 1$, and let q'' denote the value of the variable q just prior to executing line 12 in the for-loop with $i = s$. Then $q' < m$ and $q'' = m$. Moreover, $q'' \le q' + 1$, and so we must have $q' = m - 1$, which implies $P_{m-1} \sqsupset T_{s-1}$. In addition, $q'' = q' + 1$, and so during the execution of the body of the for-loop with $i = s$, line 10 *is* executed (otherwise q'' could not be greater

---

[2] Let's just assume that $r \ge 2$ so that $i = r - 1 \ge 1$, otherwise, if $r \le 1$, then $m = 1$ and the pattern P consists of just one character, in which case it is easy to see that the KMP matcher works properly.

than q'). This means that the condition for executing line 9 was satisfied, that is, $P[m] = T[s]$. Altogether we obtain $P_m \sqsupset T_s$, and so the algorithm correctly reported an occurence of P with shift $s - m$.

$\square$

Algorithm 1 KMP-MATCHER(T, P)

Require: A pattern P and a text T.
Ensure: Reports each occurence of the pattern P in the text T.

 1: $n \leftarrow \text{length}(T)$
 2: $m \leftarrow \text{length}(P)$
 3: $\pi \leftarrow$ COMPUTE-PREFIX-FUNCTION(P)
 4: $q \leftarrow 0$
 5: **for** $i \leftarrow 1$ to n **do**
 6:     **while** $q > 0$ and $P[q + 1] \neq T[i]$ **do**
 7:         $q \leftarrow \pi[q]$
 8:     **end while**
 9:     **if** $P[q + 1] = T[i]$ **then**
10:         $q \leftarrow q + 1$
11:     **end if**
12:     **if** $q = m$ **then**
13:         **print** "Pattern P occurs with shift" $i - m$
14:         $q \leftarrow \pi[q]$
15:     **end if**
16: **end for**

Algorithm 2 COMPUTE-PREFIX-FUNCTION(P)

Require: A pattern P.
Ensure: Return the prefix function π of P.

 1: $m \leftarrow \text{length}(P)$
 2: $\pi[1] \leftarrow 0$
 3: $k \leftarrow 0$
 4: **for** $q \leftarrow 2$ to m **do**
 5:     **while** $k > 0$ and $P[k + 1] \neq P[q]$ **do**
 6:         $k \leftarrow \pi[k]$
 7:     **end while**
 8:     **if** $P[k + 1] = P[q]$ **then**
 9:         $k \leftarrow k + 1$
10:     **end if**
11:     $\pi[q] \leftarrow k$
12: **end for**
13: **return** π

---

3 Correctness of Compute-Prefix-Function

Notation: For any positive integer n, we let $[n]$ denote the set $\{1, \ldots, n\}$.

Theorem 3.1. *The algorithm* Compute-Prefix-Function(P) *(Algorithm 2) correctly computes the prefix function of any pattern P.*

Our proof of Theorem 3.1 is based on the following observation.

Observation 3.2. *Given some pattern P, suppose we are executing the algorithm* Compute-Prefix-Function(P). *Then, for each iteration of the for-loop (lines 4-12) we have:*

(i) Before *the execution of the first line of the body of the for-loop (line 5), $k < q$;*

(ii) after *the execution of the last line of the body of the for-loop (line 11), $\pi[i]$ is defined and $\pi[i] < i$ for all $i \in [q]$; and*

(iii) after *the execution of the last line of the body of the for-loop (line 11), $\pi[k] < k < q$.*

Remark 3.3. *Observe that the value of the variable k may be changed during the execution of the body of the for-loop (line 4-12), and therefore the value of k in* (i) *need not be the same as the value of k in* (iii). *Notice that* (iii) *follows immediately from* (ii) *and the fact that $\pi[q]$ is assigned the value of k in line 11, that is, $q > \pi[q] = k > \pi[k]$.*

The proof of Observation 3.2 is a straight-forward inductive argument on the value of ℓ, and we postpone it until later.

Proof of Theorem 3.1. Let the prefix function of P be denoted φ, that is, for any integer $q \in [m]$,

$$\varphi[q] = \max\{k : k < q \text{ and } P_k \sqsupset P_q\}$$

We prove the following statement by induction on the value of $\ell \in [m]$.

(*) *After the execution of the algorithm, the value of the variable $\pi[\ell]$ is equal to the value $\varphi[\ell]$ of the prefix function φ of P.*

The statement (*) is easily seen to be true for $\ell \in \{1, 2\}$, since for $\ell = 1$ the value of $\pi[\ell]$ is set to 0, and for $\ell = 2$ the value of $\pi[\ell]$ is set to 0 or 1 depending on whether or not $P[1] = P[2]$.

Notice that the value of $\pi[1]$ is never changed after the execution of line 2, and that the value of $\pi[i]$ for any $i \in [m] \backslash \{1\}$ is only changed (and defined) during the execution of the for-loop (line 4-12) with $q = i$.

Suppose that the statement (*) is true for any $\ell < j$ for some $j \in \{3, \ldots, m-1\}$. We need to show that the algorithm computes $\pi[j]$ such that $\pi[j] = \varphi[j]$. As mentioned above, in order to prove the induction hypothesis for j, we only need to study the behavior of the algorithm during the execution of the body of the

4

for-loop (line 4-12) with $q = j$.

Before execution of the first line of the body of the for-loop with $q = j$, we have $\varphi[q-1] = \pi[q-1] = k$ [by the inductive hypothesis and line 11], in particular, P_k is a proper suffix of P_{q-1} and $\varphi[q] \leq k+1$, since $\varphi[q] > k+1$ would imply $\varphi[q-1] > k$. In addition, $0 \leq k < q-1$, since $k = \phi[q-1]$.

The following statement will be the loop-invariant for the while-loop (lines 5-7):

$$\text{(a) } 0 \leq k < q-1, \qquad \text{(b) } P_k \sqsupset P_{q-1} \qquad \text{and} \qquad \text{(c) } \varphi[q] \leq k+1 \qquad (1)$$

Claim 3.4. *The statement* (1) *is a loop-invariant for the while-loop (line 5-7).*

Argument. Initialization: The statement (1) is, as previously noted, true prior to the first iteration of the loop.

Maintenance: If $k > 0$ and $P[k+1] \neq P[q]$ when executing line 5, then, clearly, $\varphi[q] < k+1$. By the induction hypothesis (here we use the fact that $k < q$) and (1), $P_{\pi[k]} \sqsupset P_k \sqsupset P_{q-1}$, and so, by transitivity, $P_{\pi[k]} \sqsupset P_{q-1}$. By the induction hypothesis, the truth of statement (1) prior to the execution of line 6 and Observation 3.2, we obtain $\pi[k] = \varphi[k] < k < q-1$ and so (a) and (b) of statement (1) also hold after the execution of line 6.
What then might be the value of $\varphi[q]$? The prefix function has already been computed correctly for all values less than q, that is, $\pi[t] = \varphi[t]$ for every $t \leq q-1$. Since $\varphi[q] - 1 < k$, $P_{\varphi[q]-1} \sqsupset P_{q-1}$ and $P_k \sqsupset P_{q-1}$ before the execution of line 6, we obtain

$$P_{\varphi[q]-1} \sqsupset P_k \qquad (2)$$

Comparing this with the definition of the prefix function, we find that $\varphi[k] \geq \varphi[q] - 1$. The algorithm assigns the value $\pi[k]$ (which already is cumputed correctly such that $\varphi[k] = \pi[k]$) to the variable k, and so we obtain $\varphi[q] \leq k+1$ after the execution of line 6, that is, (c) of statement (1) also holds after the execution of line 6.

$\diamond$

The while-loop (lines 5-7) will not loop infinitely, since each time the algorithm executes line 6, it decreases the value of k by at least one unit, cf. Observation 3.2. Thus, eventually, $k \leq 0$ or $P[k+1] = P[q]$, and the algorithm will proceed to execute line 8.

If $P[k+1] = P[q]$ when executing line 8, then, it follows from Claim 3.4 that $\varphi[q]$ is equal to $k+1$ and in this case the algorithm assigns the value $k+1$ to $\pi[q]$.

If $P[k+1] \neq P[q]$ when executing line 8, then, since the algorithm jumped out of the while-loop, we must have $k = 0$ (recall that we always have $k \geq 0$). It follows from Claim 3.4 that $\varphi[q]$ is zero, and in this case the algorithm assigns the value $k = 0$ to $\pi[q]$. Thus, the value of $\pi[q]$ has been computed correctly during the execution of the for-loop, and so the desired statement follows by induction. $\square$

Here follows the proof of Observation 3.2. The proof is straight-forward and it might be appropriate to skip it during you presentation at the oral exam.

Proof of Observation 3.2. We are only concerned with (i) and (ii) of Observation 3.2, as (iii) follows from (ii), cf. Remark 3.3. We apply induction on the value of q.

If $q = 2$, then $k = 0$ before the execution of the first line of the body of the for-loop (line 5), and so $k < q$, that is, (i) holds. After the execution of the last line of the body of the for-loop (line 11), we have $k \leq 1$, $\pi[2] \leq 1 < 2$ and $\pi[1] = 0 < 1$, and therefore (ii) also holds.

Let's suppose that (i) and (ii) hold for all values of $q \leq j$ for some integer $j \geq 2$. We entend to show that (i) and (ii) also hold for $q = j + 1$ (where $j + 1 \leq m$), and then the desired result follows by induction.

Notation[3]: Let k_i denote the value of the variable k just before the algorithm executes the first line of the body of the for-loop (line 5) with $q = i$, and let k'_i denote the value of k just after the algorithm has executed the last line of the body of the for-loop (line 11).

Observe that $k_{i+1} = k'_i$ for all $i \in \{2, \ldots, q - 1\}$.

According to line 9 of the algorithm, we have $\pi[i] = k'_i$. Next, we argue that, for any $i \in \{2, \ldots, j + 1\}$,
$$k'_i \leq k_i + 1 \tag{3}$$

By induction, we have $\pi[i] < i$ for all $i \in [j]$. In particular, since $k_i < j + 1$, we have $\pi[k_i] < k_i$. Hence, the first, and all subsequent[4], assignments of the form $k \leftarrow \pi[k]$ (line 6) with $q = i$ decreases the value of k (of course, line 6 may not be executed during the iteration of the for-loop with $q = i$). Line 9 increase the value of the variable k by 1 (again, line 9 may not be executed during the iteration of the for-loop with $q = i$). In any case, we find that the value of the variable k *after* the execution of line 11 is at most $k_i + 1$, that is, $k'_i \leq k_i + 1$.

Thus, we have $q = j + 1 > k_j + 1 \geq k'_j = k_{j+1}$, by the induction hypothesis and (3), which proves (i). As for (ii), we observe that $\pi[i]$ with $i < j + 1$ isn't changed during the execution of the for-loop with $q = j + 1$. Obviously, $\pi[j+1]$ *is* defined during the execution of the for-loop with $q = j + 1$. Thus, in order to establish (2), we need only observe that $\pi[j + 1] < j + 1$:

$$\pi[j + 1] = k'_{j+1} \leq k_{j+1} + 1 = k'_j + 1 = \pi[j] + 1 < j + 1$$

where we used the induction hypothesis and (3). □

---

[3]It isn't strictly necessary to introduce *both* k_i and k'_i as $k_{i+1} = k'_i$, except for $i = q$. Anyway, it seems convenient to use both k_i and k'_i in the explanation.
[4]Is it possible to give a better argument for this (obvious) claim?