Table of prefixes

Let x be a string of length $m \geq 1$. We define the table

$$pref\colon \{0, 1, \ldots, m-1\} \to \{0, 1, \ldots, m-1\}$$

by

$$pref[k] = |lcp(x, x[k \mathinner{.\,.} m-1])|$$

for $k = 0, 1, \ldots, m-1$, where lcp(u,v) is the ***longest common prefix*** of strings u and v.

The table *pref* is called the ***table of prefixes*** for the string x. It memorizes the prefixes of x that occur inside the string itself. We note that $pref[0] = |x|$. The following example shows the table of prefixes for the string $x =$ `abbabaabbabaaaabbabbaa`.

k	0	1	2	3	4	5	6	7	8	9	10	11
$x[k]$	a	b	b	a	b	a	a	b	b	a	b	a
$pref[k]$	22	0	0	2	0	1	7	0	0	2	0	1

k	12	13	14	15	16	17	18	19	20	21
$x[k]$	a	a	a	b	b	a	b	b	a	a
$pref[k]$	1	1	5	0	0	4	0	0	1	1

Some string matching algorithms (see Chapter 3) use the table *suff* which is nothing but the analogue of the table of prefixes obtained by considering the reverse of the string x.

The method for computing *pref* that is presented below proceeds by determining $pref[i]$ by increasing values of the position i on x. A naive method would consist in evaluating each value $pref[i]$ independently of the previous values by direct comparisons; but it would then lead to a quadratic-time computation, in the case where x is the power of a single letter, for example. The utilization of already computed values yields a linear-time algorithm. For that, we introduce, the index i being fixed, two values g and f that constitute the key elements of the method. They satisfy the relations

$$g = \max\{j + pref[j] : 0 < j < i\} \tag{1.5}$$

and

$$f \in \{j : 0 < j < i \text{ and } j + pref[j] = g\}\ . \tag{1.6}$$

We note that g and f are defined when $i > 1$. The string $x[f \mathinner{.\,.} g-1]$ is then a prefix of x, thus also a border of $x[0 \mathinner{.\,.} g-1]$. It is the empty string when $f = g$. We can note, moreover, that if $g < i$ we have then $g = i-1$, and that on the contrary, by definition of f, we have $f < i \leq g$.

The following lemma provides the justification for the correctness of the function Prefixes.

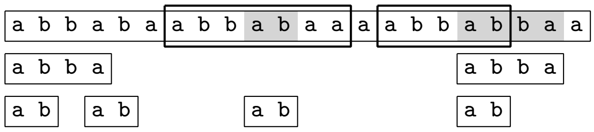

Figure 1.16 Illustration of the function PREFIXES. The framed factors $x[6\,..\,12]$ and $x[14\,..\,18]$, and the gray factors $x[9\,..\,10]$ and $x[17\,..\,20]$ are prefixes of string $x =$ `abbabaabbabaaaabbabbaa`. For $i = 9$, we have $f = 6$ and $g = 13$. The situation at this position is the same that at position $3 = 9 - 6$. We have $pref[9] = pref[3] = 2$ which means that `ab`, of length 2, is the longest factor at position 9 that is a prefix of x. For $i = 17$, we have $f = 14$ and $g = 19$. As $pref[17 - 14] = 2 \geq 19 - 17$, we deduce that string $\texttt{ab} = x[i\,..\,g-1]$ is a prefix of x. Letters of x and $x[i\,..\,m-1]$ have to be compared from respective positions 2 and g for determining $pref[i] = 4$.

Lemma 1.25
If $i < g$, we have the relation

$$pref[i] = \begin{cases} pref[i-f] & \text{if } pref[i-f] < g - i\ , \\ g - i & \text{if } pref[i-f] > g - i\ , \\ g - i + \ell & \text{otherwise}\ , \end{cases}$$

where $\ell = |lcp(x[g-i\,..\,m-1], x[g\,..\,m-1])|$.

Proof Let us set $u = x[f\,..\,g-1]$. The string u is a prefix of x by the definition of f and g. Let us also set $k = pref[i-f]$. By the definition of $pref$, the string $x[i-f\,..\,i-f+k-1]$ is a prefix of x but $x[i-f\,..\,i-f+k]$ is not.

In the case where $pref[i-f] < g-i$, an occurrence of $x[i-f\,..\,i-f+k]$ starts at the position $i-f$ on u—thus also at the position i on x—which shows that $x[i-f\,..\,i-f+k-1]$ is the longest prefix of x starting at position i. Therefore, we get $pref[i] = k = pref[i-f]$.

In the case where $pref[i-f] > g-i$, $x[0\,..\,g-i-1] = x[i-f\,..\,g-f-1] = x[i\,..\,g-1]$, and $x[g-i] = x[g-f] \neq x[g]$. We have thus $pref[i] = g - i$.

In the case where $pref[i-f] = g-i$, we have $x[g-i] \neq x[g-f]$ and $x[g-f] \neq x[g]$, therefore we cannot decide on the result of the comparison between $x[g-i]$ and $x[g]$. Extra letter comparisons are necessary and we conclude that $pref[i] = g - i + \ell$. ∎

In the computation of $pref$, we initialize the variable g to 0 to simplify the writing of the code of the function PREFIXES, and we leave f initially undefined. The first step of the computation consists thus in determining $pref[1]$ by letter comparisons. The utility of the above statement comes for computing next values. An illustration of how the function works is given in Figure 1.16. A schema showing the correspondence between the variables of the function and the notation used in the statement of Lemma 1.25 and its proof is given in Figure 1.17.

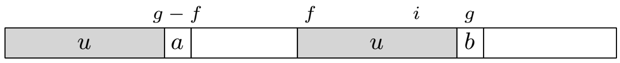

Figure 1.17 Variables i, f, and g of the function PREFIXES. The main loop has for invariants: $u = lcp(x, x[f \,..\, m-1])$ and thus $a \neq b$ with $a, b \in A$, then $f < i$ when f is defined. The schema corresponds to the situation in which $i < g$.

PREFIXES(x, m)
1 $pref[0] \leftarrow m$
2 $g \leftarrow 0$
3 **for** $i \leftarrow 1$ **to** $m - 1$ **do**
4   **if** $i < g$ and $pref[i - f] \neq g - i$ **then**
5         $pref[i] \leftarrow \min\{pref[i - f], g - i\}$
6   **else** $(g, f) \leftarrow (\max\{g, i\}, i)$
7   **while** $g < m$ and $x[g] = x[g - f]$ **do**
8             $g \leftarrow g + 1$
9         $pref[i] \leftarrow g - f$
10 **return** $pref$

Proposition 1.26
The function PREFIXES *applied to a string x and to its length m produces the table of prefixes for x.*

Proof We can verify that the variables f and g satisfy the relations (1.5) and (1.6) at each step of the execution of the loop.

We note then that, for i fixed satisfying the condition $i < g$, the function applies the relation stated in Lemma 1.25, which produces a correct computation. It remains thus to check that the computation is correct when $i \geq g$. But in this situation, lines 6–8 compute $|lcp(x, x[i \,..\, m-1])| = |x[f \,..\, g-1]| = g - f$ which is, by definition, the value of $pref[i]$.

Therefore, the function produces the table $pref$. ■

Proposition 1.27
The execution of the operation PREFIXES(x, m) *runs in time* $\Theta(m)$*. Less than* $2m$ *comparisons between letters of the string x are performed.*

Proof Comparisons between letters are performed in line 7. Every comparison between equal letters increments the variable g. As the value of g never decreases and that it varies from 0 to at most m, there are at most m positive comparisons. Each negative comparison leads to the next step of the loop. Then there are at most $m - 1$ of them. Thus less than $2m$ comparisons on the overall.

The previous argument also shows that the total time of all the executions of the loop of lines 7–8 is $\Theta(m)$. The other instructions of the

a	b	b	a	b	a	a	b	b	a	b	a	a	a	a	b	b	a	b	b	a	a

Figure 1.18 Relations between borders and prefixes. Considering the string $x =$ abbabaabbabaaaabbabbaa, we have the equality $\mathit{pref}[9] = 2$ but $\mathit{border}[9+2-1] = 5 \neq 2$. We also have both $\mathit{border}[15] = 2$ and $\mathit{pref}[15-2+1] = 5 \neq 2$.

loop 3–9 take a constant time for each value of i giving again a global time $\Theta(m)$ for their execution and that of the function. ■

The bound of $2m$ on the number of comparisons performed by the function Prefixes is relatively tight. For instance, we get $2m-3$ comparisons for a string of the form $a^{m-1}b$ with $m \geq 2$, $a, b \in A$, and $a \neq b$. Indeed, it takes $m-1$ comparisons to compute $\mathit{pref}[1]$, then one comparison for each of the $m-2$ values $\mathit{pref}[i]$ with $1 < i < m$.

Relation between borders and prefixes

The tables *border* and *pref*, whose computation is described above, both memorize occurrences of prefixes of x. We explicit here a relation between these two tables.

The relation is not immediate for the reason that follows, which is illustrated in Figure 1.18. When $\mathit{pref}[i] = \ell$, the factor $u = x[i \mathinner{..} i+\ell-1]$ is a prefix of x but it is not necessarily the border of $x[0 \mathinner{..} i+\ell-1]$ because this border can be longer than u. In the same way, when $\mathit{border}[j] = \ell$, the factor $v = x[j-\ell+1 \mathinner{..} j]$ is a prefix of x but it is not necessarily the *longest* prefix of x occurring at position $j-\ell+1$.

The proposition that follows shows how the table *border* is expressed using the table *pref*. One can deduce from the statement an algorithm for computing the table *border* knowing the table *pref*.

Proposition 1.28
Let $x \in A^+$ and j be a position on x. Then:

$$\mathit{border}[j] = \begin{cases} 0 & \text{if } I = \emptyset \ , \\ j - \min I + 1 & \text{otherwise} \ , \end{cases}$$

where $I = \{i : 0 < i \leq j \text{ and } i + \mathit{pref}[i] - 1 \geq j\}$.

Proof We first note that, for $0 < i \leq j$, $i \in I$ if and only if $x[i \mathinner{..} j] \preceq_{\text{pref}} x$. Indeed, if $i \in I$, we have $x[i \mathinner{..} j] \preceq_{\text{pref}} x[i \mathinner{..} i+\mathit{pref}[i]-1] \preceq_{\text{pref}} x$, thus $x[i \mathinner{..} j] \preceq_{\text{pref}} x$. Conversely, if $x[i \mathinner{..} j] \preceq_{\text{pref}} x$, we deduce, by definition of $\mathit{pref}[i]$, $\mathit{pref}[i] \geq j-i+1$. And thus $i + \mathit{pref}[i] - 1 \geq j$. Which shows that $i \in I$. We also note that $\mathit{border}[j] = 0$ if and only if $I = \emptyset$.

It follows that if $\mathit{border}[j] \neq 0$ (thus $\mathit{border}[j] > 0$) and $k = j - \mathit{border}[j] + 1$, we have $k \leq j$ and $x[k \mathinner{..} j] \preceq_{\text{pref}} x$. No factor $x[i \mathinner{..} j]$, $i < k$, satisfies the relation $x[i \mathinner{..} j] \preceq_{\text{pref}} x$ by definition of $\mathit{border}[j]$. Thus $k = \min I$ by the first remark, and $\mathit{border}[j] = j - k + 1$ as stated. ■