Computing the Border Table

Recall that $\text{border}_x(k)$ = length of longest border of $x[0..k]$. For brevity, we in this note just write $b(k)$ for $\text{border}_x(k)$.

Main idea: compute $b(k)$ for increasing k. When computing $b(k)$, we may refer to $b(k')$ for $k' < k$ (so idea is a type of dynamic programming). The base case is $b(0) = 0$. In the general case, we are currently looking for $b(k)$.

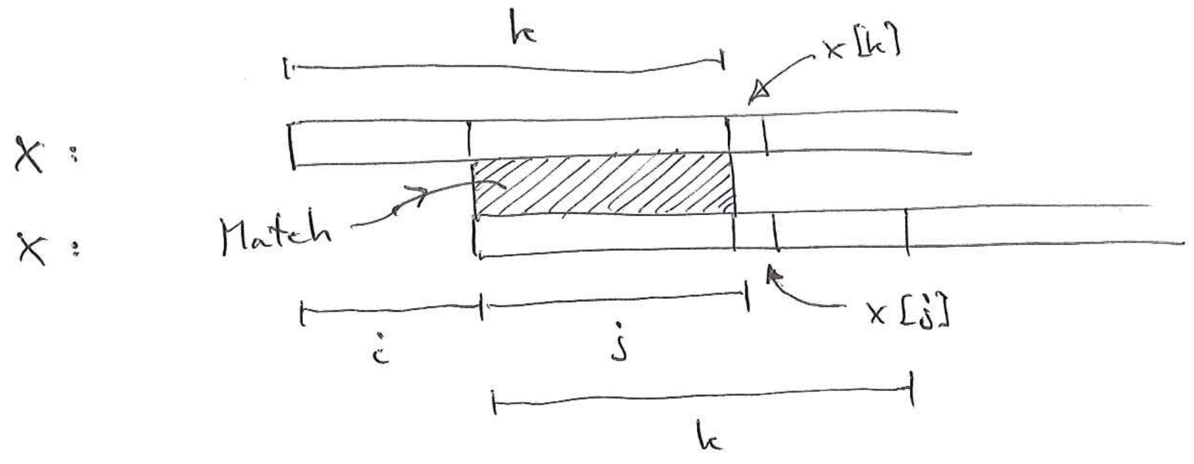
The algorithm will maintain two variables i and j under the following invariant:

i) $i + j = k$

ii) There is a border of $x[0..k-1]$ of length j

iii) $b(k) \leq j + 1$

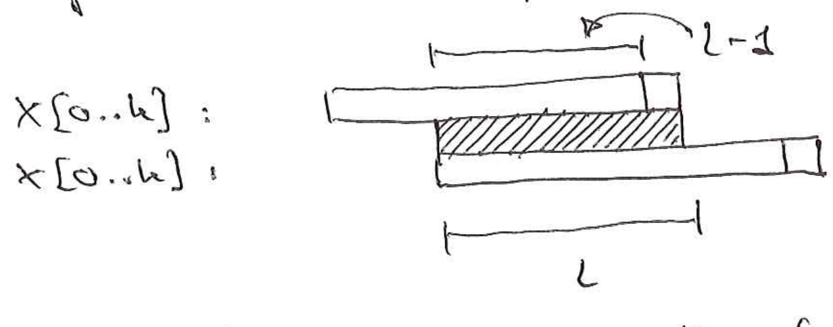
The following figure illustrates i) and ii):



We note that if $j = b(k-1)$, then ii) is obviously fulfilled. Also iii) is fulfilled, by the following lemma:

<u>Lemma</u>: $b(k) - 1 \leq b(k-1)$ for all k

<u>Proof</u>: Any border of $x[0..k]$ of length $l \geq 1$ implies a border of $x[0..k-1]$ of length $l-1$:



Since $b(k-1)$ is the length of the longest border of $x[0..k-1]$, the lemma follows for k where $b(k) \geq 1$. When $b(k) = 0$, the lemma is trivially true ($b(k-1) \geq 0$ always).

$\square$

In each iteration, the algorithm performs the test

$$X[i+j] \overset{?}{=} X[j]$$

<u>Case A</u>: The test is positive.

By i) + ii) we have found a border of $X[0..k]$ of length $j+1$. By iii) this means $b(k) = j+1$. We therefore perform the updates

$$b(i+j) = j+1$$

$$j = j+1$$

We found $b(k)$ and should look at next k, so i) is maintained. Also ii) and iii) are maintained, by the note above the Lemma on page 2.

<u>Case B</u>: The test is negative.

We then know $b(k) < j+1$, by i) + iii).

<u>Subcase B1</u>: $j = 0$.

By $0 \leq b(k) < j+1 = 0+1 = 1$,

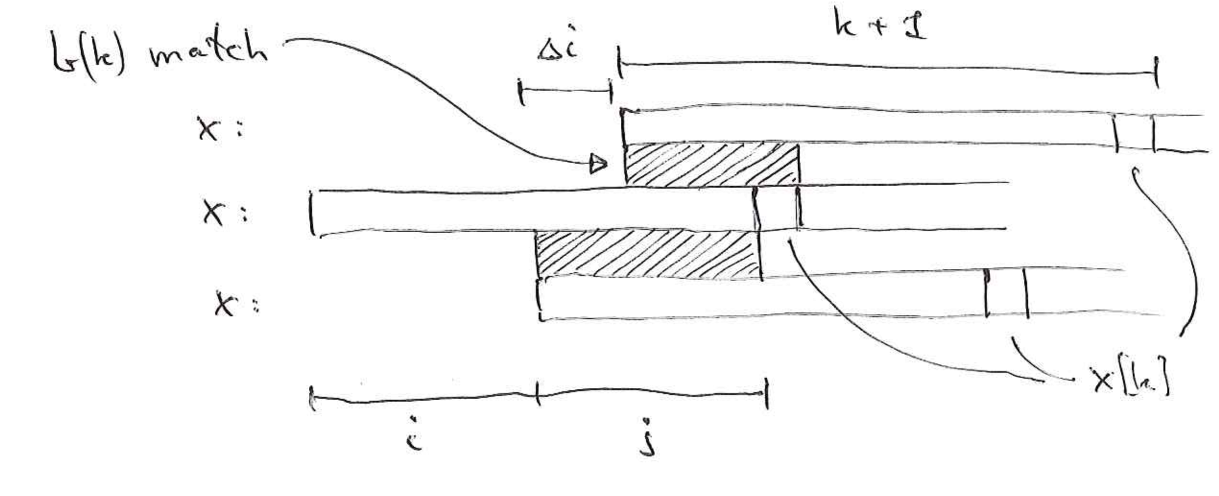
we see $b(k) = 0$. We perform the updates

$$b(i+j) = 0$$
$$i = i + 1$$

We found $b(k)$ and should look at next k, so i) is maintained. By $j = 0$ and the note above the lemma on page 2, also ii) and iii) are maintained.

Subcase B2: $j \geq 1$.

Let us add the border of $x[0..k]$ giving $b(k)$ to the figure on top of page 2:



Since $b(k) < j + 1$, we know that $\Delta i \geq 1$.

By the matching areas, we see a border of length $j - \Delta i$ for $x[0..j-1]$ (compare top and bottom copies of x). Thus, we have $b(j-1) \geq j - \Delta i$, which means $\Delta i \geq j - b(j-1)$

From the figure, we see

$$b(k) = j - \Delta i + 1 .$$

Hence,

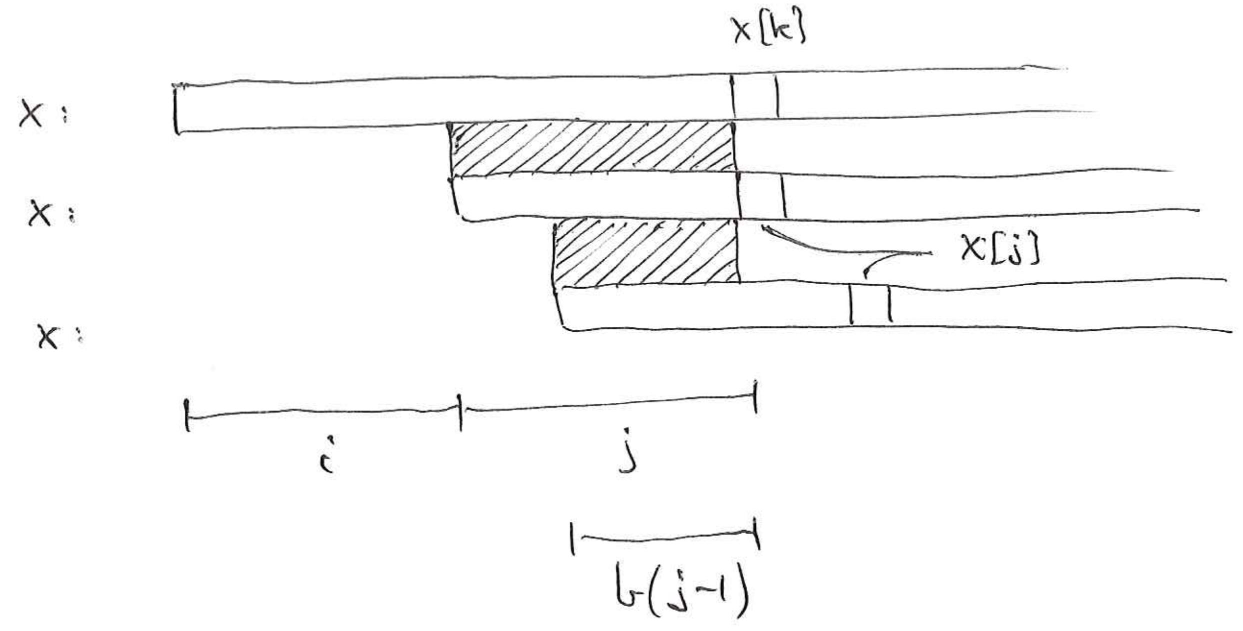
$$\begin{aligned} b(k) &\leq j - (j - b(j-1)) + 1 \\ &= b(j-1) + 1 \qquad *) \end{aligned}$$

We perform the updates

$$\begin{aligned} i &= i + (j - b(j-1)) \\ j &= b(j-1) . \end{aligned}$$

We have not found $b(k)$ yet, so k does not change. As $i+j$ does not change, i) is maintained. By *), iii) is maintained.

By the definition of $b(j-1)$, ii) is maintained, as the following figure illustrates (value of j shown is before the update):



Since j decreases in case B2 (but never to a negative value), we must eventually reach case B1 or case A, and advance k.

By the convention $b(-1) = -1$, we can merge the updates of cases B1 and B2 into

$$i = i + (j - b(j-1))$$
$$j = \max\{0, b(j-1)\}$$

if we also assume that the entries of the $b()$ table are initialized to zero.

Here is the algorithm above as code :

```
$b(-1) = -1$
for $t = 0$ to $|x| - 1$
    $b(t) = 0$

$i = 1$
$j = 0$
while $i + j < |x|$
    if $x[i+j] == x[j]$
        $b(i+j) = j + 1$
        $j = j + 1$
    else
        $i = i + (j - b(j-1))$
        $j = \max\{0, b(j-1)\}$
```

After initialization (lines before while) we have $b(-1)$ and $b(0)$ correct, and we have invariant i), ii), and iii) fulfilled [iii) by the Lemma] for $k = 1$. Since $j - b(j-1) \geq 1$ by the definition of a border, the value of i will increase by at least one in the else case and be unchanged in the *if* case. In particular, it can never get below its initial value of 1.

This means $k = i+j > j > j-1$, so the lookup $b(j-1)$ will return the correct value, since by the analysis above the code maintains the following invariant:

iv) The entries $b(t)$ are correct for $-1 \le t < i+j$.

[Note that $b(t)$ in i) - iii) and in the analysis refers to its real (mathematical) value, while $b(t)$ in the code and in iv) refers to a table in the code which is filled during the execution of the code.]

Hence, correctness of the code is clear.

For complexity, we have the following changes for i, j and $i+j$ for the cases:

Case	Δi	Δj	$\Delta(i+j)$
A	0	1	1
B1	1	0	1
B2	≥ 1	≤ -1	0

By the condition in the <u>while</u>-loop and $\Delta(i+j) \leq 1$, we see $i+j = |X|$ at end of the algorithm.

Let $z = 2i+j$. Then initially z is $2 \cdot 1 + 0 = 2$, at end $z \leq 2(i+j)$ [since $j \geq 0$ always in the code], so $z \leq 2 \cdot |X|$ at end, and each iteration of the <u>while</u>-loop increases z by at least 1.

Thus, there can be at most $2(|X|-1)$ iterations of the <u>while</u>-loop, so the algorithm runs in $O(|X|)$ time.