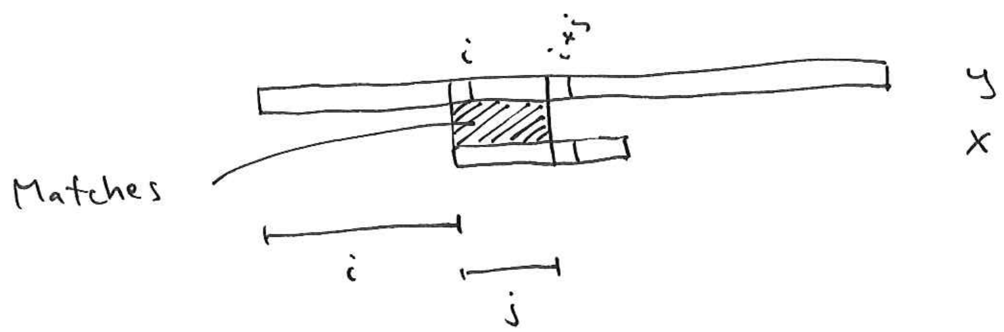
The KMP-algorithm

Reports all occurences of a pattern x in a text y. An extension of the naive sliding window algorithm. Uses left to right check in window.

Consider checking x at pos i in y, after $j \geq 0$ chars have been matched so far in window:



Next test is $y[i+j] \overset{?}{=} x[j]$

We will aim at maintaining the following <u>invariants</u>:

1) At current position i of x, the first j chars matches y.

2) All positions less than i have been tested (and reported if a full match of x).

Case A: Next test is positive (i.e., $y[i+j] = x[j]$ and $j < |x|$).

We do the update $\boxed{j \to j+1}$

Clearly maintains invariants. Position is still a possible match.

Case B1: Next test is negative (i.e., $j < |x|$ and $y[i+j] \neq x[j]$) and $j = 0$.

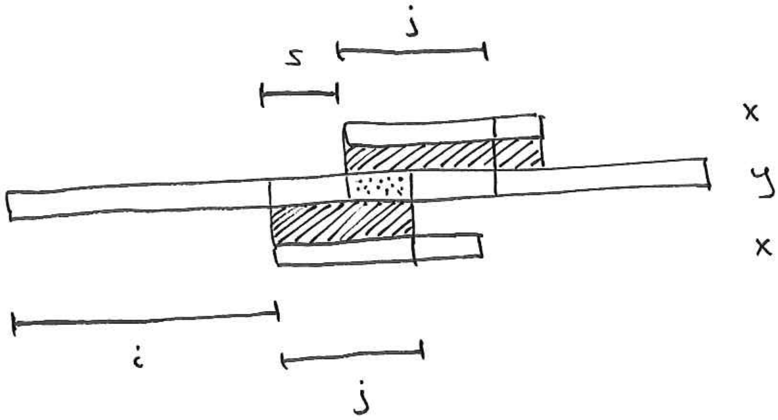
Position is now no longer a possible match. We can safely move window by one, hence we do the update $\boxed{i \to i+1}$

Clearly maintains the invariants.

Case B2: Next test is negative ($j < |x|$ and $y[i+j] \neq x[j]$) and $j \geq 1$.

Position is now no longer a possible match. So position should move. How much can we guarantee it must move?

Assume there is a match of x at position $i + s$ for some s, $1 \leq s \leq j$. Then we have:



We see a string of length $j - s$ which is both a prefix and a suffix of the string $x[0..j-1]$. As $s \geq 1$, it is a proper suffix/prefix, hence it is a <u>border</u> of $x[0..j-1]$.

Recall that border$(j-1)$ for the string x denotes the length of the <u>longest</u> such border. Hence

$$j - s \leq \text{border}(j-1)$$
$$\Updownarrow$$
$$j - \text{border}(j-1) \leq s$$

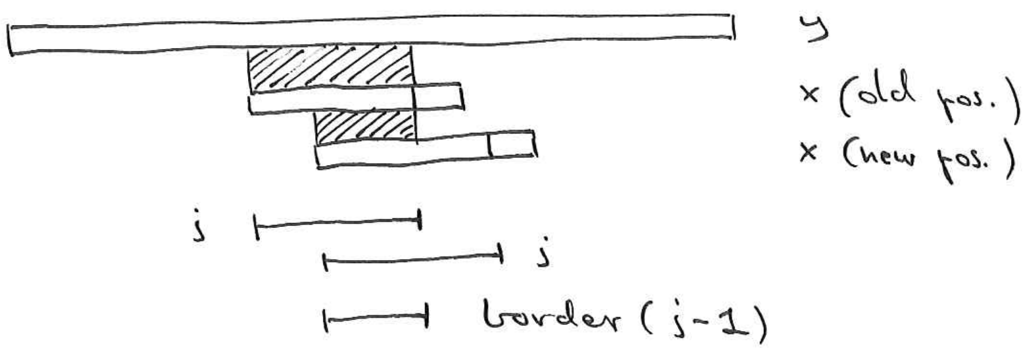
Thus, we have a lower bound on the distance to the next possible match of x.

So we can safely do the update

$$\boxed{i \to i + (j - \text{border}(j-1))}\ .$$

(Here and later, border(k) is understood to be the border table for x.)

The existence of a border of length border(j-1) tells us that after moving the position of the window this much, we know that border(j-1) chars of x matches y:



We therefore do the update $\boxed{j \to \text{border}(j-1)}$

The updates of Cases B1 and B2 can be merged using that border(k) ≥ 0 for all k [since it is a length] and defining border(-1) to be -1. Then the following updates cover both cases:

$$\boxed{\begin{aligned} i &\to i + j - \text{border}(j-1) \\ j &\to \max\{0, \text{border}(j-1)\} \end{aligned}}$$

<u>Case C:</u> $j = |x|$.

A match of x should be reported at position i. Additionally, window should move. Here, same analysis as for case B2 applies, hence we do the same updates.

Combined, we have the following algorithm:

<u>KMP(x, y)</u>

```
i = 0
j = 0
while i ≤ |y| - |x|
    if j < |x| AND y[i+j] == x[j]
        j = j + 1
    else
        if j == |x|
            report match at position i
        i = i + j - border(j-1)
        j = max{0, border(j-1)}
```

Complexity

Let $z = 2i + j$.

In Case A we have $\Delta i = 0$, $\Delta j = 1 \Rightarrow \Delta z = 1$

In Case B1 we have $\Delta i = 1$, $\Delta j = 0 \Rightarrow \Delta z = 2$

In Case B2/C we have

$$\Delta i = j - \mathrm{border}(j-1)$$
$$\Delta j = \mathrm{border}(j-1) - j$$

So $\Delta z = 2\Delta i + \Delta j = \Delta i + \underbrace{\Delta i + \Delta j}_{0}$

$= \Delta i \geq 1$ (as border$(j-1) < j$, since borders are proper substrings)

After initialization (first two lines) of KMP, $z = 2 \cdot i + j = 0$. For each traversal of while loop body, $\Delta z \geq 1$. At entry to last traversal we have $i \leq |y| - |x|$. At no place in the algorithm we set $j > |x|$. So if t traversals are performed in total, we have $t - 1 \leq \Delta z_{total} = z - 0 = z = 2i + j \leq 2(|y| - |x|) + |x| = 2|y| - |x|$

So KMP runs in time $O(|y|)$ [given the border array].