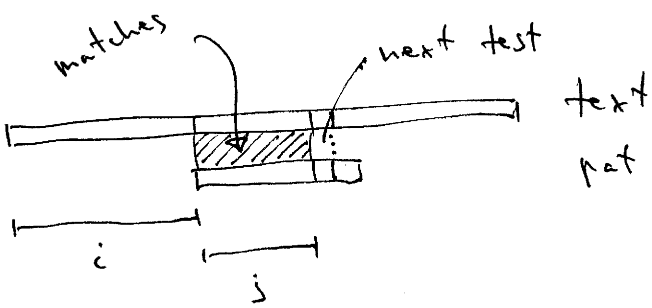
KMP-Algorithm

(For reporting all occurences of pat in text)

An Extension of the brute-force alg.

Consider checking pat at shift i in text, where j chars have been matched so far:



Consider the next test $text[i+j+1] \stackrel{?}{=} pat[j+1]$.

We aim at maintaining the following invariants:

1) For current shift i of pattern, the first j chars matches text.

2) All shifts $< i$ of pat have been tested (and reported if pat matched there).

<u>Case A</u>: Next test is positive (chars matches).

We then do the update

$$j \to j+1$$

[Shift i is still a possible occurrence, now with one more char known to match]

<u>Case B1</u>: Next test is negative, and $j = 0$

We then do the update

$$i \to i+1$$

[Shift i is now known <u>not</u> to be an occ., and shift should increase. We increase the smallest possible step, and keep $j=0$.]

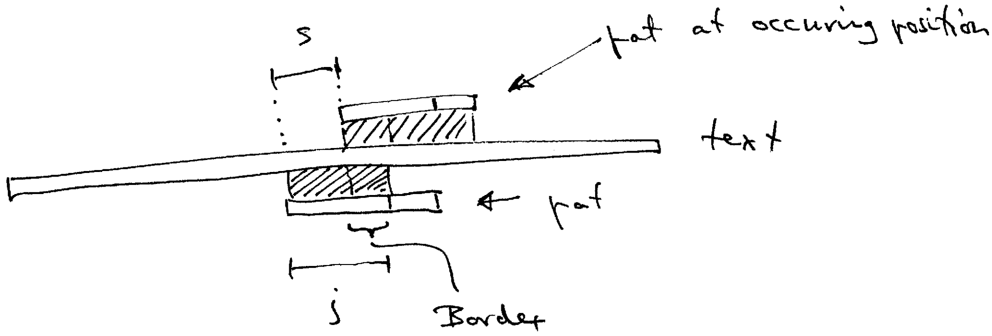
Case B2 : Next test is negative, and $j \geq 1$

[Shift i is now known not to be an occ., and shift should increase. How much can we guarantee it must move ?]

~~Consider the next~~ Assume there is an occ. of pat in text at position $i+s$ [i.e., s further to the right],

for an $s \leq j$. [We know $s \geq 1$, so $1 \leq s \leq j$, which is the reason why the idea here in B2 does not hold for $0 = j$, hence the need for the (simple) separate case B1].

That is, we have



Here, we see a border (a suffix which is also a prefix) of the string pat[1..j] which has length $j-s$. The border is non-trivial since $s \geq 1$.

$B_{pat}(j)$ is the length of the longest such border,

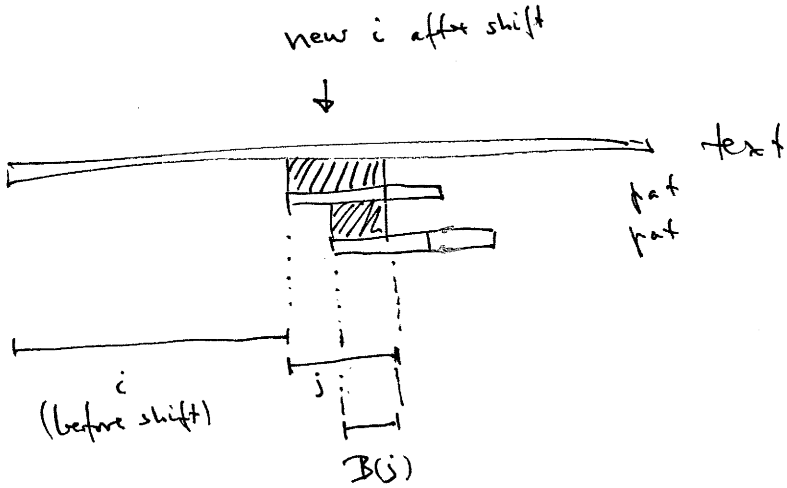
so $j - s \leq B(j)$. [We write $B(j)$ for $B_{pat}(j)$ from now]

$$\Updownarrow$$

$$j - B(j) \leq s .$$

Hence, there can be no occurences the next $j - B(j)$ shifts, so we can ~~update~~ increase i by this amount.

Doing so leaves $B(j)$ known matches of chars after the shift [as $B(j)$ corresponds to an <u>actual</u> border]:



We therefore do the updates

$$i \longrightarrow i + (j - B(j))$$

$$j \longrightarrow j - (j - B(j)) = B(j)$$

(and this keeps invariants 1) and 2)).

<u>Note</u>: By the definition $B(0) = -1$, and the observation $B(k) \geq 0$ for $k \geq 1$ ($B(k)$ is a length) we can merge cases B1 & B2 into the single update:

$$i \longrightarrow i + (j - B(j))$$

$$j \longrightarrow \max\{0, B(j)\}$$

Above, we have not considered checking whether we do lookups in text and pat past their end.

Adding these, the above analysis (and aimed-at invariants) suggests the following algorithm:

<u>KMP(text, pat)</u>

$n = |text|$
$m = |pat|$

```
i = 0
j = 0
while [i ≤ n - m]
    if [j < m AND] text[i+j+1] == pat[j+1]
        j = j+1
    else
        [if j == m
            report match at i]
        i = i + (j - B(j))
        j = max{0, B(j)}
```

Ensures no lookups past ends

Ensures reporting of occurrences (all)

It is easy to verify that invariants 1) and 2) are maintained, from which correctness follows.

Complexity:

Let $k = i + j$

In each case, the change (Δ) to i, j, k are as follows

Case	Δi	Δj	Δk
A	0	1	1
B1	1	0	1
B2	≥ 1	(?)	0

$\Delta k = \Delta i + \Delta j$
$= (j - B(j)) - (j - B(j))$
$= 0$

$\Delta i = (j - B(j)) \geq 1$ ← as borders are nontrivial ($B(j) < j$)

So for $z = k + i$ we see that:

i) z $(= k + i = i + j + i)$ is 0 after initialization in KMP [first two lines]

$z = k + i$
$\Downarrow$
$\Delta z = \Delta k + \Delta i$

ii) $\Delta z \geq 1$ for each loop traversed (which falls in exactly one of cases A, B1, or B2).

Assume the loop is traversed t times.

At entry to t'th loop :

as we do enter loop t
[see loop condition]

$$\#\text{ loops already done} = t-1 \le z = 2\cdot i + j \le 2(n-m) + j$$

$$\le 2(n-m) + m$$

j can never exceed m in code. [as $B(j) < j$].
for last line of code.

$$\Updownarrow$$

$$\underline{t \le 2n - m + 1}$$ (So alg. <u>terminates</u>)

Note : last iteration must use the <u>else</u> - part (as i is not changed in <u>if</u> - part and loop condition is based on i).

So <u># comparisons performed</u> (<u>if</u> - part) is $\le 2n - m$.

For the text $\underbrace{aaa\cdots a}_{n}$ and pattern $\underbrace{abbb\ldots b}_{m}$ the algorithm <u>will</u> perform $2n-m$ comparisons. Hence, this can be said to be <u>the exact</u> <u>worst case complexity</u> of KMP.

Summing up:

For $|text| = n$ and $|pat| = m$ we have

Brute-Force takes time $\Theta((n-m) \cdot m)$
[which can be $\Theta(n^2)$, eg. for $m = n/2$].

KMP takes time $\Theta(n)$, assuming the function $B(j)$ is available (as a precomputed table, eg.). Note that $B(j)$ depends only on pat, and can be reused for other text's.

However, for a single text and pat, the time for KMP is really $\Theta(n + \text{preprocessing}(m))$, where preprocessing(m) designates the time to find $B(j)$ for all j.

There is a straightforward alg. arising from the definition of B which runs in time $\Theta(m^2)$ [two for-loops]. Can we do better?