Theory: Knuth-Morris-Pratt algorithm

24 minutes 5 / 6 problems solved

Start practicing

The Knuth-Morris-Pratt (KMP) algorithm is an efficient approach to the substring searching problem. To check whether a pattern s is a substring of a text t, the algorithm requires $O(|s| + |t|)$ operations in the worst case. In this topic, we will learn this algorithm and see how it works by example. After completing the topic, you will be able to implement the algorithm and use it to efficiently solve various problems connected to substring searching.

§1. Algorithm description

The KMP algorithm is similar to the naive substring searching approach: at each step, it compares a pattern with a substring of a text trying to find a complete matching. The difference between these two algorithms is how they process the case of mismatched symbols. In the naive algorithm, we simply shift a pattern by one symbol and start a new iteration. In some cases, this may lead to many unnecessary comparisons. To process this case more efficiently, the KMP algorithm calculates the prefix function for the pattern and then uses it to shift the pattern more optimally.

Let's see in more details how this step works for $s = BACBAD$ and $t = BACBACBAD$. First, we calculate the prefix function p for the pattern:

	0	1	2	3	4	5
s	B	A	C	B	A	D
p	0	0	0	1	2	0

Then, we start searching for the pattern in the text:

	0	1	2	3	4	5	6	7	8
t	B	A	C	B	A	C	B	A	D
s	B	A	C	B	A	D			
	0	1	2	3	4	5			

At the first iteration, there is a mismatch in the last pair of symbols: $t[5] \neq s[5]$. We can see that $s[0..5] = BACBA$ is a substring of the pattern that matches the beginning of the text. Since the length of this substring is 5 and the length of its longest border is $p[4] = 2$, we may shift the pattern by $5 - 2 = 3$ symbols and continue the comparison with the 5th symbol of the text and 2nd symbol of the pattern knowing that the previous symbols already match:

	0	1	2	3	4	5	6	7	8
t	B	A	C	B	A	C	B	A	D
				s	B	A	C	B	A	D
					0	1	2	3	4	5

The yellow symbols on the figure above correspond to the current position in the text and in the pattern. So, using the prefix function, we may find an optimal shift for the pattern avoiding comparisons that won't result in matching. This allows us to significantly reduce the total number of symbol comparisons.

For arbitrary pattern s and text t, the KMP algorithm can be formulated as follows:

1. Calculate the prefix function p for the pattern s.
2. Set the first substring of the text with length $|s|$ as current.

Current topic:

- ✓ Knuth-Morris-Pratt algorithm

Topic depends on:

- ✓ Searching a substring
- ✓ Prefix function

Topic is required for:

- Knuth-Morris-Pratt algorithm in Java

Table of contents:

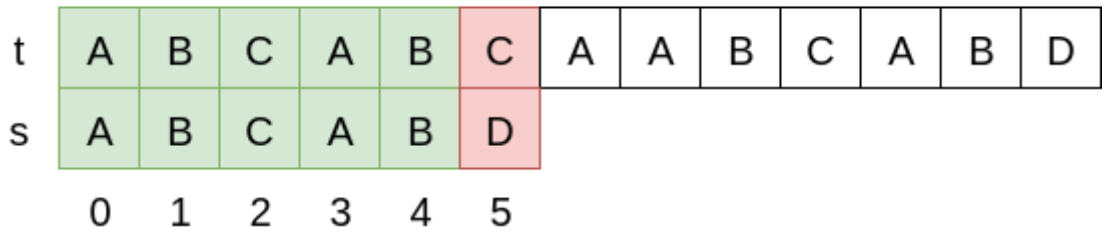
3. Compare the pattern with the current substring of the text. If all symbols match, save the index of the found occurrence. Otherwise, shift the pattern by $L - p[L-1]$ symbols, where L is the length of the matched substring of the pattern.
4. Continue step 3 until all substrings of the text are processed. Then, return the found occurrences.

§2. An example

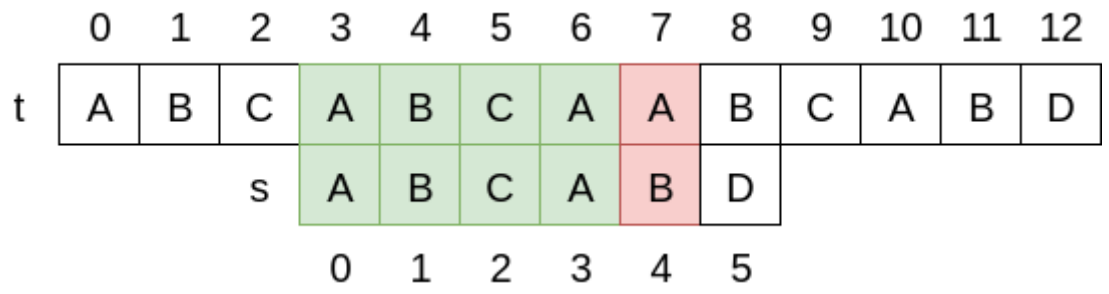
Let's apply the Knuth-Morris-Pratt algorithm for a pattern $s = ABCABD$ in a text $t = ABCABCAABCABD$. First, we need to calculate the prefix function p for the pattern:

	0	1	2	3	4	5
s	A	B	C	A	B	D
p	0	0	0	1	2	0

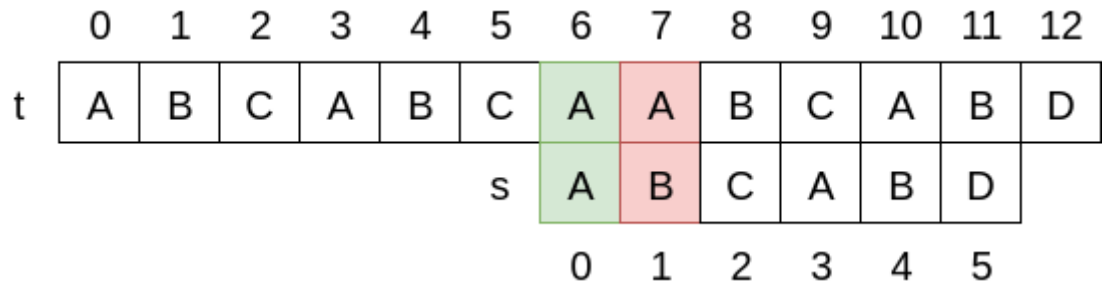
Then, we start comparing the pattern with substrings of the text:



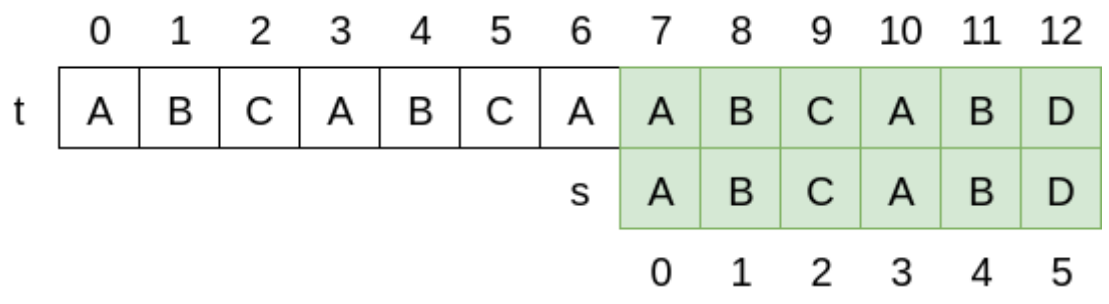
At the first iteration, there is a mismatch in the last pair of symbols: $t[5] \neq s[5]$. Since $p[4] = 2$, we shift the pattern by $5 - 2 = 3$ symbols and start a new iteration:



Now, there is a mismatch again: $t[7] \neq s[4]$. Since $p[3] = 1$, we shift the pattern by $4 - 1 = 3$ symbols and start a new iteration:



We have a mismatch again: $t[7] \neq s[1]$. Since $p[0] = 0$, we shift the pattern by $1 - 0 = 1$ symbol:



Now, all symbols of the pattern match with the current substring of the text, so an occurrence is found. Since all symbols of the text are processed, the algorithm is finished.

If you want to see a visualization of the KMP algorithm, you may take a look at this site.

§3. Complexity analysis

Assume we want to find a pattern s in a text t. First, we calculate the prefix function for s, which requires $O(|s|)$ operations. After that, we start searching s in t performing a symbol-by-symbol comparison of the pattern with substrings of t. If the corresponding symbols match, we simply move to the next pair of symbols. The total number of such cases is no more than $|t|$. If we have a mismatch, we use the calculated prefix function to find an optimal shift for the pattern. At each such case, the number of times we can shift the pattern is limited by the number of symbols that matched before. Therefore, the total number of such operations is no more than $|t|$ as well. Thus, the total running time is $O(|s| + |t|)$.

The algorithm requires $O(|s|)$ additional memory to store the prefix function for s.

Report a typo

44 users liked this theory. **2** didn't like it. **What about you?**

Start practicing

Comments (2) Hints (0) Useful links (0) Show discussion