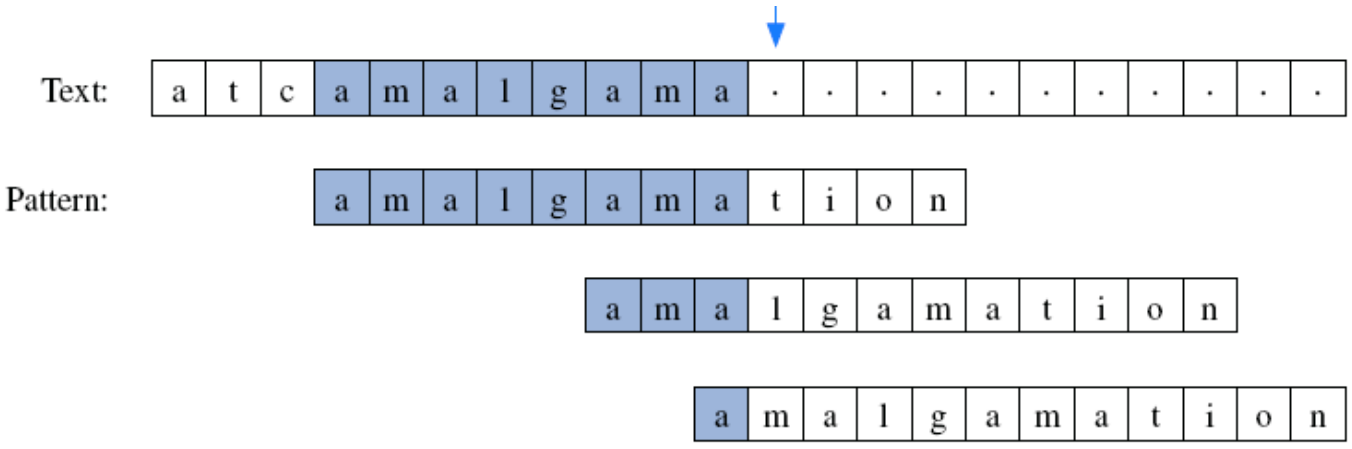
Knuth-Morris-Pratt (KMP) String Matching Algorithm

KMP is a linear-time string-matching algorithm. In Brute-Force pattern matching for a certain alignment of the pattern, if we find several matching characters but then detect a mismatch, we ignore all the information gained by the successful comparisons and restart with the next incremental placement of the pattern and again compare with text input.

The algorithm uses a table which is called *prefix function / failure function / π function* to avoid testing useless shifts in the naive pattern-matching algorithm.

Overview

Knowing these q characters have matched successfully text characters allows us to determine immediately that certain shifts are invalid. In other words, knowing that P_q is prefix of T_{s+q}, we want the longest proper prefix P_k of P_q that is also a suffix of T_{s+q}.



If a mismatch occurs at the indicated location, the pattern could be shifted to the second alignment, without explicit need to recheck the partial match with the prefix `ama`. If the mismatched character is not an `l`, then the next potential alignment of the pattern can take advantage of the common `a`.

Failure Function

To implement the KMP algorithm, we will pre-compute a *failure function* f, that indicates the proper shift of the pattern upon a failed comparison. Specifically, the failure function $f(k)$ is defined as the length of the longest prefix of the pattern that is a suffix of the substring `pattern[1 .. k]`. Intuitively, if

we find a mismatch upon character `pattern[k + 1]`, the function $f(k)$ tells us how many of the immediately preceding characters can be reused to restart the pattern.

k	0	1	2	3	4	5	6	7	8	9	10	11
$P[k]$	a	m	a	l	g	a	m	a	t	i	o	n
$f(k)$	0	0	1	0	0	1	2	3	0	0	0	0

Algorithm

The main part of the KMP algorithm is its while loop, each iteration of which performs a comparison between the character at index j in the text and the character at index k in the pattern. If the outcome of this comparison is a match, the algorithm moves on to the next characters in both (or reports a match if reaching the end of the pattern). If the comparison failed, the algorithm consults the failure function for a new candidate character in the pattern, or starts over with the next index in the text if failing on the first character of the pattern (since nothing can be reused).

let us define $s = j - k$ where s is the total amount by which the pattern has been shifted with respect to the text. Throughout the execution of the algorithm, we have $s \leq n$. One of the following three cases occurs at each iteration of the loop.

- If $text[j] = pattern[k]$, then j and k each increase by 1, thus s is unchanged.
- If $text[j] \neq pattern[k]$ and $k > 0$, then j does not change and s increases by at least 1, since in this case s changes from $j - k$ to $j - f(k-1)$ note that this is an addition of $k - f(k-1)$, which is positive because $f(k-1) < k$.
- If $text[j] \neq pattern[k]$ and $k = 0$, then j increases by 1 and s increases by 1, since k does not change.

Time and Space Complexity

- The Knuth-Morris-Pratt algorithm performs pattern matching on a text string of length n and a pattern string of length m in $O(n + m)$ time.
- KMP algorithm needs a preprocessing (*prefix function / failure function* computation) which takes $O(m)$ space and time complexity.

Applications

In real world KMP algorithm is used in those applications where pattern matching is done in long strings, whose symbols are taken from alphabets with little cardinality. An example is the DNA alphabet, which consists on only 4 symbols (A,C,G,T). KMP is really suitable because many repetition of the same letter allows many jumps, and so less computation time wasted.

References

- Introduction to Algorithms Book
- Data Structures and Algorithms in Java Book