Boyer Moore String Matching Algorithm

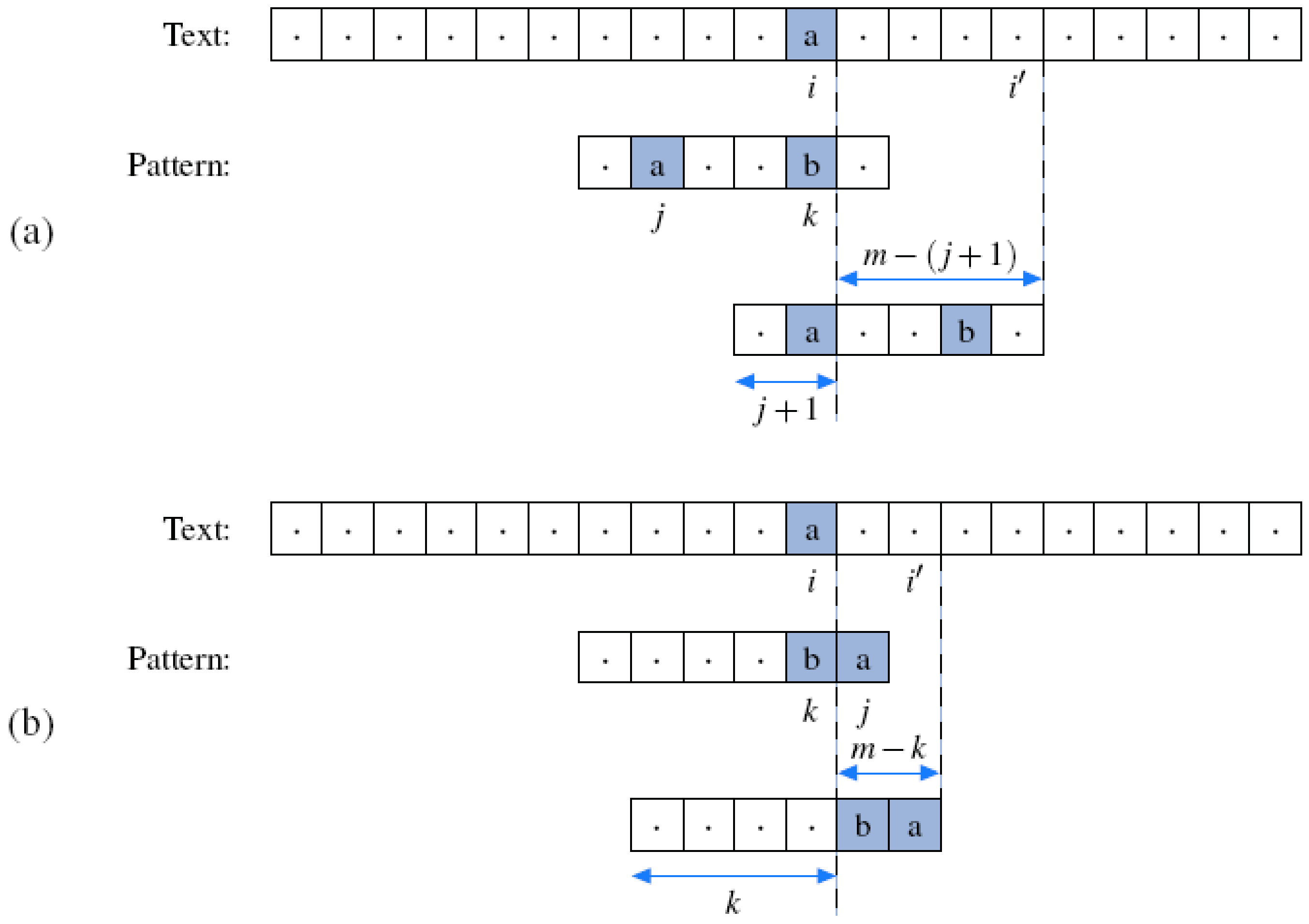
The below presented algorithm is a simplified version of the Boyer-Moore original algorithm.

Overview

The Boyer-Moore pattern-matching algorithm, can sometimes avoid examining a significant fraction of the character in the text. The main idea of the Boyer-Moore algorithm is to improve the running time of the brute-force algorithm by using two time-saving heuristics.

- *Heuristic 1:* When testing a possible placement of the pattern against the text, perform the comparisons against the pattern from **right-to-left**.
- *Character-Jump Heuristic:* During the testing of a possible placement of the pattern within the text, a mismatch of character $text[i] = c$ with the corresponding character $pattern[k]$ is handled as follows
  - If c is not contained anywhere in the pattern, then shift the pattern completely past $text[i] = c$.
  - Otherwise, shift the pattern until an occurrence of character c gets aligned with $text[i]$.

Algorithm



We let i represent the index of the mismatched character in the text, k represent the corresponding index in the pattern, and j represent the index of the last occurrence of $text[i]$ within the pattern.

We distinguish two cases

(a) $j < k$, in which case we shift the pattern by $k - j$ units and thus index i advances by $m - (j + 1)$ units.
(b) $j > k$ in which case we shift the pattern by one unit, and index i advances by $m - k$ units.

The efficiency of the Boyer-Moore algorithm relies on quickly determining where a mismatched character occurs elsewhere in the pattern. Boyer Moore algorithm uses a preprocessing function $last(c)$ where c is in the pattern, $last(c)$ is the index of the last (rightmost) occurrence of c in the pattern. Otherwise if c is not present in pattern, we define $last(c) = -1$.

We can use a hash table to represent the last function, with only those characters from the pattern occurring in it. The space usage for this approach is proportional to the number of distinct alphabet symbols that occur in the pattern, and thus $O(\max(m, |\Sigma|))$. The expected lookup time remains $O(1)$.

Text: a b a c a a b a d c c a b
Pattern: a b a c a b

c	a	b	c	d
$last(c)$	4	5	3	-1

Time and Space Complexity

If using a traditional lookup table, the worst-case running time of the Boyer-Moore algorithm is $O(nm + |\Sigma|)$. The computation of the last function takes $O(m + |\Sigma|)$ time, although the dependence on $|\Sigma|$ is removed if using a hash table. The actual search for the pattern takes $O(nm)$ time in the worst case which is the same as the brute force algorithm.

The worst-case performance, however, is unlikely to be achieved for English text in that case, the Boyer-Moore algorithm is often able to skip large portions of text.

Applications

- The Boyer-Moore algorithm is used in GNU's Linux grep command utility.

References

- Data Structures and Algorithms in Java Book