

## The Boyer-Moore algorithm

**Step 1** For a given pattern and the alphabet used in both the pattern and the text, construct the bad-symbol shift table as described earlier.

**Step 2** Using the pattern, construct the good-suffix shift table as described earlier.

**Step 3** Align the pattern against the beginning of the text.

**Step 4** Repeat the following step until either a matching substring is found or the pattern reaches beyond the last character of the text. Starting with the last character in the pattern, compare the corresponding characters in the pattern and the text until either all  $m$  character pairs are matched (then stop) or a mismatching pair is encountered after  $k \geq 0$  character pairs are matched successfully. In the latter case, retrieve the entry  $t_1(c)$  from the  $c$ 's column of the bad-symbol table where  $c$  is the text's mismatched character. If  $k > 0$ , also retrieve the corresponding  $d_2$  entry from the good-suffix table. Shift the pattern to the right by the number of positions computed by the formula

$$d = \begin{cases} d_1 & \text{if } k = 0, \\ \max\{d_1, d_2\} & \text{if } k > 0, \end{cases} \quad (7.3)$$

where  $d_1 = \max\{t_1(c) - k, 1\}$ .

Shifting by the maximum of the two available shifts when  $k > 0$  is quite logical. The two shifts are based on the observations—the first one about a text's mismatched character, and the second one about a matched group of the pattern's rightmost characters—that imply that shifting by less than  $d_1$  and  $d_2$  characters, respectively, cannot lead to aligning the pattern with a matching substring in the text. Since we are interested in shifting the pattern as far as possible without missing a possible matching substring, we take the maximum of these two numbers.

**EXAMPLE** As a complete example, let us consider searching for the pattern BAOBAB in a text made of English letters and spaces. The bad-symbol table looks as follows:

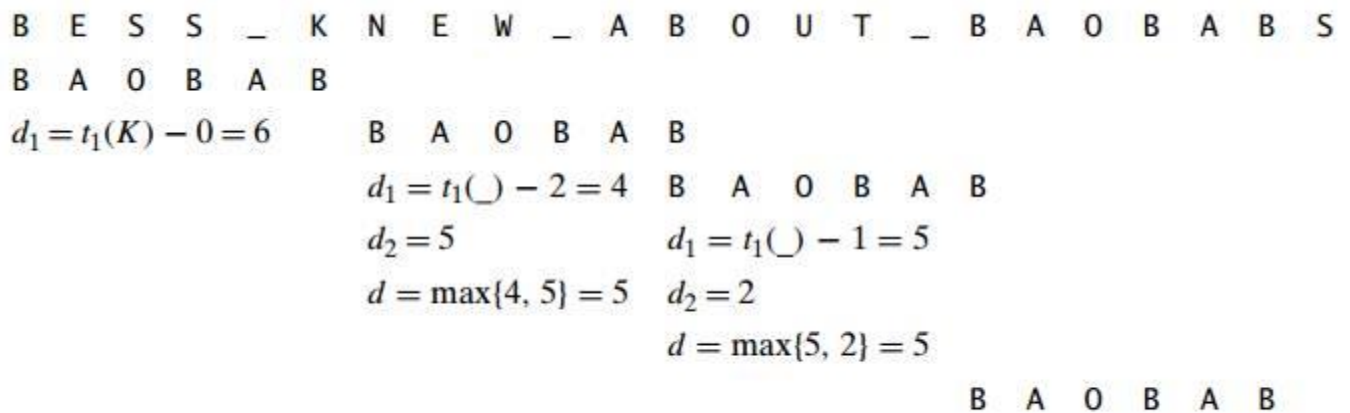
<i>c</i>	A	B	C	D	...	0	...	Z	_
$t_1(c)$	1	2	6	6	6	3	6	6	6

The good-suffix table is filled as follows:

<i>k</i>	pattern	$d_2$
1	BAO <u>B</u> AB	2
2	<u>B</u> AOBAB	5
3	<u>B</u> A <u>O</u> BAB	5
4	<u>B</u> A <u>O</u> B <u>A</u> B	5
5	<u>B</u> A <u>O</u> B <u>A</u> B <u>B</u>	5

The actual search for this pattern in the text given in Figure 7.3 proceeds as follows. After the last B of the pattern fails to match its counterpart K in the text, the algorithm retrieves  $t_1(K) = 6$  from the bad-symbol table and shifts the pattern by  $d_1 = \max\{t_1(K) - 0, 1\} = 6$  positions to the right. The new try successfully matches two pairs of characters. After the failure of the third comparison on the space character in the text, the algorithm retrieves  $t_1( ) = 6$  from the bad-symbol table and  $d_2 = 5$  from the good-suffix table to shift the pattern by  $\max\{d_1, d_2\} = \max\{6 - 2, 5\} = 5$ . Note that on this iteration it is the good-suffix rule that leads to a farther shift of the pattern.

The next try successfully matches just one pair of B's. After the failure of the next comparison on the space character in the text, the algorithm retrieves  $t_1( ) = 6$  from the bad-symbol table and  $d_2 = 2$  from the good-suffix table to shift



**FIGURE 7.3** Example of string matching with the Boyer-Moore algorithm.

the pattern by  $\max\{d_1, d_2\} = \max\{6 - 1, 2\} = 5$ . Note that on this iteration it is the bad-symbol rule that leads to a farther shift of the pattern. The next try finds a matching substring in the text after successfully matching all six characters of the pattern with their counterparts in the text.

When searching for the first occurrence of the pattern, the worst-case efficiency of the Boyer-Moore algorithm is known to be linear. Though this algorithm runs very fast, especially on large alphabets (relative to the length of the pattern), many people prefer its simplified versions, such as Horspool's algorithm, when dealing with natural-language-like strings.