Results Analysis

Given the text "ILOVECPT212ILOVEUSM" and the pattern "LOVE", the output of the code is:

```
Text: ILOVECPT212ILOVEUSM
Pattern: LOVE
Pattern occurs at shift = 1
Pattern occurs at shift = 12
```

First Occurrence at Position 1:

The pattern “LOVE” starts at index 1 in the text ”ILOVECPT212ILOVEUSM”. The algorithm successfully matches all characters in the pattern with the corresponding characters in the text starting from index 1.

Second Occurrence at Position 12:

The pattern “LOVE” starts again at index 12 in the text “ILOVECPT212ILOVEUSM”. The algorithm finds another match starting from index 12.

Conclusion

The pattern “LOVE” is found at two positions in the text, demonstrating the algorithm's ability to handle multiple occurrences efficiently. The preprocessing steps ensure that the search phase is fast and skips as many characters as possible upon mismatches.

Strengths of the Boyer-Moore Algorithm

Efficacy in Big Alphabetization:

The technique is effective, particularly for big alphabets, because the bad character heuristic permits it to skip textual portions.

Performance:

Assuming n and m are the text and pattern lengths, respectively, the Boyer-Moore method operates in sublinear time in the best scenario. Considering this, it is much faster than simple string matching techniques.

Heuristics-based:

Significant reductions in the number of comparisons required are possible when the bad character and excellent suffix heuristics work together, particularly in cases where mismatches develop.

Usability:

The practicality and effectiveness of the Boyer-Moore algorithm make it a popular choice for real-world applications such as DNA sequence analysis, text editors, and search engines.

Weakness of the Boyer-Moore Algorithm

Preprocessing Duration:

It takes more time and space to do the preprocessing processes necessary to create the good suffix tables and poor character. To be more precise, the bad character table needs space that matches the size of the alphabet.

Using Small Patterns to Perform:

The preprocessing expense may be more for minor patterns than the heuristic gains. In certain situations, simpler algorithms like the Knuth-Morris-Pratt (KMP) algorithm might be more effective.

The most complex case:

The Boyer-Moore algorithm's worst-case time complexity, O(nm), is uncommon but happens in some situations where the heuristics don't produce helpful shifts. For example, if a large number of the characters in the text do not fit into the pattern.

Implementation Complexity:

Compared to other string matching algorithms, this approach requires more work to execute correctly. It can be difficult to make sure that both heuristics are applied and utilised correctly.

Detailed Explanation of the Code:

1. **Initialization and Preprocessing:**

   Text: "ILOVECPT212ILOVEUSM"

   Pattern: "LOVE"

   Text length (n): 19

Pattern length (m): 4

2. **Bad Character Heuristic Preprocessing:**

   Initialize the badchar array with -1.

   Update the array with the last occurrence of each character in the pattern:
   css
   Copy code
   badchar['L'] = 0

   badchar['O'] = 1

   badchar['V'] = 2

   badchar['E'] = 3

   ○
3. **Strong Good Suffix Rule Preprocessing:**

   Initialize shiftArray and suffixArray.

   Set suffixArray[m] = m + 1.

   Process the pattern to find the widest border for each suffix. For "LOVE", there are no borders other than the entire string itself.

4. **Case 2 of the Good Suffix Rule Preprocessing:**

   Adjust the shift values for cases where no strong suffix match is found.

Pattern Matching:

1. **First Alignment (Shift s = 0):**
   a. Text Window: "ILOV"
   b. Compare characters from right to left.
      i. Compare 'V' (text[3]) with 'E' (pattern[3]): Mismatch.
   c. Use the bad character heuristic:
      i. The mismatched character is 'V' (text[3]).
      ii. Last occurrence of 'V' in the pattern is at index 2.
      iii. Shift the pattern by “max(1, 3 - 2) = 1”.

2. **Second Alignment (Shift s = 1):**
    a. Text Window: "LOVE"
    b. Compare characters from right to left.
        i. Compare 'E' (text[4]) with 'E' (pattern[3]): Match.
        ii. Compare 'V' (text[3]) with 'V' (pattern[2]): Match.
        iii. Compare 'O' (text[2]) with 'O' (pattern[1]): Match.
        iv. Compare 'L' (text[1]) with 'L' (pattern[0]): Match.
    c. All characters match.
    d. Pattern occurs at shift = 1.
    e. Shift the pattern by shift[0] = 4.

3. **Third Alignment (Shift s = 5):**
    a. Text Window: "CPT2"
    b. Compare characters from right to left.
        i. Compare '2' (text[8]) with 'E' (pattern[3]): Mismatch.
    c. Use the bad character heuristic:
        i. The mismatched character is '2' (text[8]).
        ii. '2' does not occur in the pattern, so shift the pattern by m = 4.

4. **Fourth Alignment (Shift s = 9):**
    a. Text Window: "12IL"
    b. Compare characters from right to left.
        i. Compare 'L' (text[11]) with 'E' (pattern[3]): Mismatch.
    c. Use the bad character heuristic:
        i. The mismatched character is 'L' (text[11]).
        ii. Last occurrence of 'L' in the pattern is at index 0.
        iii. Shift the pattern by “max(1, 3 - 0) = 3”.

5. **Fifth Alignment (Shift s = 12):**
    a. Text Window: "LOVE"
    b. Compare characters from right to left.
        i. Compare 'E' (text[15]) with 'E' (pattern[3]): Match.
        ii. Compare 'V' (text[14]) with 'V' (pattern[2]): Match.
        iii. Compare 'O' (text[13]) with 'O' (pattern[1]): Match.
        iv. Compare 'L' (text[12]) with 'L' (pattern[0]): Match.
    c. All characters match.
    d. Pattern occurs at shift = 12.
    e. Shift the pattern by shift[0] = 4.

6. **Sixth Alignment (Shift s = 16):**
    a. Text Window: "USM"

    b. Compare characters from right to left.
        i. Compare 'M' (text[18]) with 'E' (pattern[3]): Mismatch.
    c. Use the bad character heuristic:
        i. The mismatched character is 'M' (text[18]).
        ii. 'M' does not occur in the pattern, so shift the pattern by m = 4.

Final Output:

- The pattern "LOVE" is found at indices 1 and 12 in the text "ILOVECPT212ILOVEUSM".