STRING MATCHING ALGORITHM

ASSIGNMENT README.md

SOLUTION:

CODES: https://github.com/TharunPatel20/UST-techAcademy/blob/main/USTJavaCourse/BeginningDSA/src/stringMatchingAlgorithms

Part 1: Naive Search Algorithm

Steps:

1. **Understand the Algorithm**: The Naive Search algorithm scans the text file by comparing the `message_to_find` character-by-character at every position in the `innocent_text`. If the substring matches, the message is found.
2. **Calculate Time Complexity**:
    - For text length `n` and pattern length `m`,
    - the worst-case time complexity is O(n×m) O(n \times m) O(n×m),
    - as the algorithm compares characters for all positions of the pattern.

Part 2: Boyer-Moore Algorithm

Steps:

1. **Research and Understand**: The Boyer-Moore algorithm uses two techniques:
    - **Bad Character Rule**: Skips characters based on mismatches.
    - **Good Suffix Rule**: Leverages repeated patterns in the `message_to_find`.

    Compare Time Complexity:

    Boyer-Moore: Best-case time complexity is O(n/m)O(n/m)O(n/m) and worst-case O(n×m)O(n \times m)O(n×m), but it typically performs much faster than Naive Search for longer texts and patterns.

Part 3: Expanding the Investigation

Choose an Additional Algorithm: Knuth-Morris-Pratt (KMP) Algorithm

1. **Working Principle**: The KMP algorithm preprocesses the pattern into a "partial match" table (or LPS array), allowing it to skip unnecessary comparisons.

2. **Compare Time Complexity:**

    KMP: O(n+m)O(n + m)O(n+m),

    as the preprocessing step takes O(m)O(m)O(m) and the search takes O(n)O(n)O(n)