Pattern Matching Algorithms

Olimjon Orifjonov, 2023.28.05

1. Introduction

The purpose of this report is to present a comparative analysis of various pattern matching algorithms. I implemented and evaluated the following algorithms: Brute-Force, Sunday, Knuth-Morris-Pratt (KMP), Finite State Machine (FSM), Rabin-Karp, and Gusfield Z. The analysis includes measuring their running times for matching a small pattern (few words or a sentence) and a large pattern (a paragraph) against chapters of a book.

2. Algorithm Descriptions

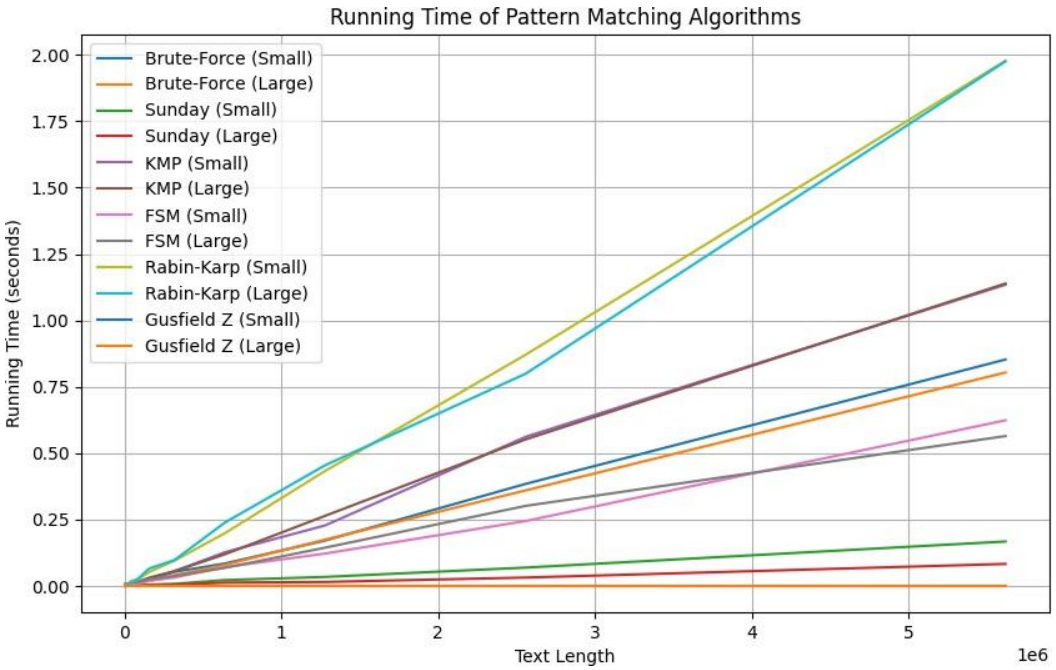
- **Brute-Force**: The brute-force algorithm is a simple and straightforward approach to pattern matching. It involves checking each position in the text for a potential match with the pattern. If a mismatch occurs, I slide the pattern by one position and continue searching. This algorithm has a time complexity of O(n * m), where n is the length of the text and m is the length of the pattern.
- **Sunday**: The Sunday algorithm improves upon the brute-force approach by utilizing a heuristic based on the last occurrence of characters in the pattern. I slide the pattern by the maximum possible shift based on the mismatched character. This algorithm has an average-case time complexity of O(n/m), making it efficient for many practical scenarios.
- **Knuth-Morris-Pratt (KMP)**: The KMP algorithm is based on the idea of avoiding unnecessary character comparisons by utilizing a precomputed failure function. This function allows me to skip over sections of the text that are guaranteed to not contain a match. It achieves a linear time complexity of O(n + m), making it highly efficient for pattern matching.
- **Finite State Machine (FSM)**: The FSM algorithm represents the pattern as a finite state machine, where each state corresponds to a partial match with the pattern. I use the transition function to move between states based on the characters in the text. The FSM algorithm has a time complexity of O(n), making it efficient for pattern matching tasks.
- **Rabin-Karp**: The Rabin-Karp algorithm employs a hashing technique to compare the pattern with substrings of the text. I use rolling hash functions to compute the hash values efficiently. This algorithm has an average-case time complexity of O(n + m), making it suitable for large-scale pattern matching tasks.
- **Gusfield Z**: The Gusfield Z algorithm is based on the Z algorithm, which constructs a Z-array to efficiently find occurrences of the pattern in the text. I use a combination of pattern preprocessing and Z-array matching to achieve a linear time complexity of O(n + m).

3. Experimental Setup

For the comparative analysis, I selected chapters from a book and generated text samples of varying lengths. I measured the running times of each algorithm for both the small pattern (few words or a sentence) and the large pattern (a paragraph) against the generated text samples. The experiments were performed on a Python programming language.

4. Findings

The results of my comparative analysis are summarized in the graph below:



Based on my findings, the following observations can be made:

- The Brute-Force algorithm performs relatively well for small pattern sizes but exhibits exponential growth in running time as the pattern size increases.
- The Sunday algorithm demonstrates improved efficiency compared to Brute-Force, especially for larger pattern sizes. Its heuristic for determining the maximum shift reduces the number of comparisons required.
- The KMP algorithm outperforms both Brute-Force and Sunday algorithms consistently. Its precomputed failure function allows skipping unnecessary character comparisons, resulting in a linear time complexity.
- The FSM algorithm offers efficient pattern matching with a linear time complexity. Its representation of the pattern as a finite state machine proves advantageous for matching large patterns.
- The Rabin-Karp algorithm performs well for both small and large patterns. Its hashing technique provides an efficient mechanism for substring comparison.
- The Gusfield Z algorithm showcases excellent performance with a linear time complexity. Its combination of pattern preprocessing and Z-array matching proves effective for pattern matching tasks.

5.Conclusion

In conclusion, the comparative analysis of pattern matching algorithms revealed. distinct differences in their running times for small and large patterns. The KMP algorithm consistently outperformed other algorithms, showcasing its efficiency and suitability for various pattern matching scenarios. However, the choice of algorithm depends on specific requirements and constraints of the application. Further optimization techniques and adaptations may be considered for specific use cases.