**Pattern Matching Algorithms:**

**Introduction**

**Pattern matching is a fundamental problem in computer science with applications in text processing, bioinformatics, and information retrieval. This report presents an empirical analysis of the performance of six pattern matching algorithms: Brute-force, Sunday, Knuth-Morris-Pratt (KMP), Finite State Machine (FSM), Rabin-Karp, and Gusfield Z. The algorithms were tested on text excerpts of varying lengths from a book to evaluate their efficiency with small and large patterns.**

**Algorithms Overview**

**1. Brute-force: This naive algorithm checks for the pattern at every possible position in the text. It has a worst-case time complexity of O(nm), where n is the length of the text and m is the length of the pattern.**

**2. Sunday: An efficient algorithm based on the Boyer-Moore algorithm, utilizing a shift table to skip unnecessary comparisons. It has an average-case time complexity of O(n/m).**

**3. Knuth-Morris-Pratt (KMP): Utilizes preprocessing to construct a longest prefix suffix (LPS) array to skip redundant comparisons. It has a worst-case time complexity of O(n+m).**

**4. Finite State Machine (FSM): Constructs a state machine representing the pattern, with a worst-case time complexity of O(nm).**

**5. Rabin-Karp: Utilizes hashing to find the pattern, with an expected time complexity of O(n+m) but can degrade to O(nm) in the worst case.**

**6. Gusfield Z: Relies on the Z-array to match the pattern efficiently, with a time complexity of O(n+m).**

**Experimental Setup**

**The book's content was duplicated to ensure sufficient text length. Two sets of patterns were used: small patterns and a large pattern. The algorithms were tested on text lengths of 100, 500, 1000, 5000, and 10000 characters, and the execution times were measured and recorded in milliseconds.**

**Results**

**Small Patterns: The running times for small patterns were recorded for each text length. Brute-force exhibited linear growth, while other algorithms like Sunday, KMP, FSM, Rabin-Karp, and Gusfield Z showed varying degrees of efficiency.**

A graph with different colored lines

Description automatically generated

**Large Pattern: Brute-force struggled significantly with longer texts and large patterns, while advanced algorithms like KMP, FSM, and Gusfield Z maintained efficient performance.**

A graph of different colored lines

Description automatically generated

**Visualization**

**Two plots were generated: one illustrating the performance of each algorithm with increasing text length for small patterns, and another showing the distinction between the performance of brute-force and advanced algorithms for the large pattern.**

**Conclusion**

**Advanced pattern matching algorithms like KMP, FSM, and Gusfield Z offer significant performance benefits over the brute-force approach, especially for longer texts and larger patterns. However, algorithms like Sunday and Rabin-Karp also provide efficient solutions with some variability in performance.**

**Pattern Matching Algorithms with Wildcards:**

**Introduction**

A graph of different colored rectangular shapes

Description automatically generated

**This report presents a comparative analysis of the Brute-force and Sunday algorithms extended to handle wildcard characters and escape sequences. Both algorithms were examined for their implementations, performance, strengths, and weaknesses.**

**Brute-force Algorithm with Wildcards: A simple pattern matching algorithm that iterates through all possible positions in the text string and compares the pattern with substrings at each position. It can be inefficient for large text strings and patterns due to its time complexity.**

**Sunday Algorithm with Wildcards: A more efficient pattern matching algorithm that utilizes a shift table mechanism. It exhibits better performance, especially for large text strings, with a time complexity of O(n + m) in the average case.**

**Comparative Analysis: Empirical tests were conducted to compare the performance of both algorithms with wildcards using various text strings and patterns. The Sunday algorithm demonstrated superior performance, particularly for large inputs, owing to its linear time complexity and efficient shift table mechanism.**

**Conclusion: While both algorithms can effectively handle pattern matching with wildcard characters and escape sequences, the Sunday algorithm offers better runtime performance and scalability, making it a preferred choice for practical applications involving large text strings and patterns.**

Part two  
  
**Pattern Matching Algorithms with Wildcards: A Comparative Analysis**

n this report, we present a comparative analysis of two widely used pattern matching algorithms, the Brute-force algorithm and the Sunday algorithm, extended to handle wildcard characters and escape sequences. We examine their implementations, evaluate their performance, and discuss their strengths and weaknesses.

**1. Brute-force Algorithm with Wildcards:** The Brute-force algorithm is a simple and straightforward pattern matching algorithm that iterates through all possible positions in the text string and compares the pattern with substrings at each position. To accommodate wildcard characters and escape sequences, the algorithm checks for wildcard characters and escapes during the comparison process. Despite its simplicity, the Brute-force algorithm can be inefficient for large text strings and patterns due to its time complexity of O(n\*m) in the worst case.

**2. Sunday Algorithm with Wildcards:** The Sunday algorithm is a more efficient pattern matching algorithm that utilizes a shift table to skip over sections of the text string during the matching process. Like the Brute-force algorithm, the Sunday algorithm is extended to handle wildcard characters and escape sequences by appropriately updating the shift table and incorporating wildcard checks into the matching logic. The Sunday algorithm typically exhibits better performance than the Brute-force algorithm, especially for large text strings, with a time complexity of O(n + m) in the average case.

**3. Comparative Analysis:** We conducted a series of experiments to compare the performance of the Brute-force and Sunday algorithms with wildcards using various text strings and patterns. The experiments evaluated the algorithms' ability to correctly identify matches, their runtime efficiency, and their scalability to handle large inputs.

* Both algorithms were able to accurately match patterns containing wildcard characters and escape sequences.
* The Brute-force algorithm exhibited consistent performance across different input sizes but showed limitations in handling large inputs due to its quadratic time complexity.
* The Sunday algorithm demonstrated superior performance, particularly for large inputs, owing to its linear time complexity and efficient shift table mechanism.

**4. Conclusion:** In conclusion, both the Brute-force and Sunday algorithms can effectively handle pattern matching with wildcard characters and escape sequences. While the Brute-force algorithm is simpler to implement and understand, it may suffer from inefficiency for large inputs. On the other hand, the Sunday algorithm offers better runtime performance and scalability, making it a preferred choice for practical applications involving large text strings and patterns.