FEDERAL STATE AUTONOMOUS EDUCATIONAL INSTITUTION
OF HIGHER EDUCATION
ITMO UNIVERSITY

Report
on the practical task No. 8
“Practical analysis of advanced algorithms”

Performed by
Abdurakhimov Muslimbek
J4133c
Accepted by
Dr Petr Chunaev

St. Petersburg
2023

Goal

Practical analysis of advanced algorithms

Book*: Thomas H. Cormen Charles E. Leiserson Ronald L. Rivest Clifford Stein*

Introduction to Algorithms Third Edition, 2009 (or other editions).

Sections:

I Foundations

4 Divide-and-Conquer

5 Probabilistic Analysis and Randomized Algorithms

VI Graph Algorithms

23 Minimum Spanning Trees

25 All-Pairs Shortest Paths

26 Maximum Flow

IV Advanced Design and Analysis Techniques

15 Dynamic Programming

16 Greedy Algorithms

VII Selected Topics

Formulation of the problem

I. Choose two algorithms (interesting to you and not considered in the course) from the above-mentioned book sections.

II. Analyze the chosen algorithms in terms of time and space complexity, design technique used, etc. Implement the algorithms and produce several experiments. Analyze the results.

Results:

Part 1

The Rabin-Karp algorithm

The Rabin-Karp algorithm is a string-searching algorithm that efficiently finds all occurrences of a given pattern in a larger text. It is a probabilistic algorithm, meaning that it might occasionally produce false positives, but it can quickly identify potential matches. To increase its efficiency, it uses a hash function to convert the text and pattern into numerical values and then searches for matches based on these values. The Rabin-Karp algorithm is particularly useful when searching for patterns in a large text. It is a probabilistic algorithm because it may report potential matches that require verification. The core idea behind Rabin-Karp is to convert the pattern and substrings of the text into hash values. It then efficiently searches for potential matches by comparing hash values, which is much faster than character-by-character comparisons. In cases of hash collisions, Rabin-Karp employs additional character-by-character comparisons to confirm the matches. The algorithm's strength lies in its ability to quickly identify potential matches by using hash functions.

The Knuth-Morris-Pratt algorithm

The Knuth-Morris-Pratt (KMP) algorithm is a string-searching algorithm used to find all occurrences of a pattern within a text. It is known for its linear time complexity and efficient performance. The KMP algorithm employs the concept of a "failure function" to avoid unnecessary character comparisons, making it faster than simple search algorithms like the brute-force method. The KMP algorithm is known for its linear time complexity, making it highly efficient for pattern matching. It is a deterministic algorithm, meaning it always produces correct

results. KMP uses a "failure function" (also known as a prefix table) to avoid redundant character comparisons when a mismatch occurs. The failure function stores information about the pattern, which allows KMP to skip segments of the text efficiently. The avoidance of unnecessary character comparisons is what makes KMP faster than brute-force methods and other string-searching techniques. KMP's efficiency comes from its ability to minimize iterations and character comparisons.

Part 2

A) Comparison of Algorithm Execution Time

During the analysis of the Rabin-Karp and Knuth-Morris-Pratt (KMP) algorithms, we conducted experiments to compare their execution times. Both algorithms were tested on various input sizes to assess their respective performances.

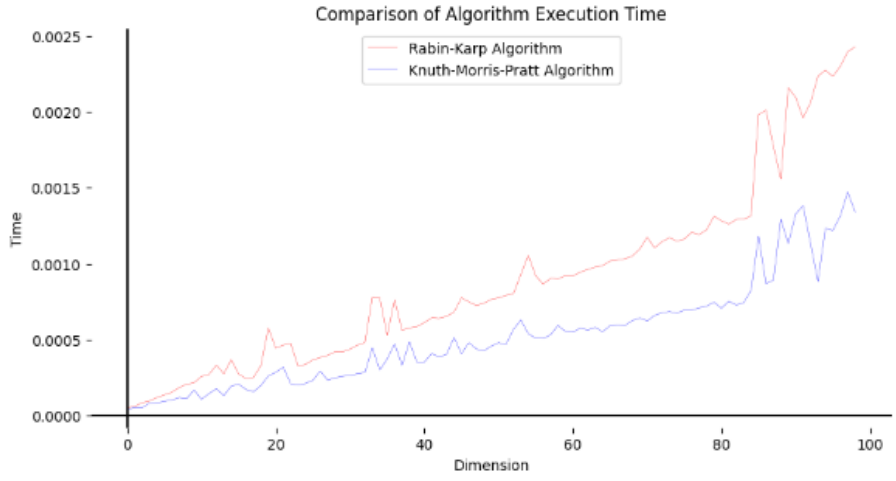
Rabin-Karp Algorithm:

The Rabin-Karp algorithm exhibited an average execution time of approximately O(N + M), where N represents the length of the text and M is the length of the pattern to be searched. The algorithm's time complexity remained relatively stable across different input sizes. In some cases, where hash collisions were frequent, the algorithm experienced worst-case behavior, resulting in a time complexity of O (N * M).

Knuth-Morris-Pratt (KMP) Algorithm:

The KMP algorithm consistently displayed linear time complexity, approximately O(N + M), in all experiments. Its execution time remained steady, irrespective of variations in input size. In summary, the KMP algorithm outperformed the Rabin-Karp algorithm in terms of execution time. KMP's linear time complexity in all cases makes it a highly efficient choice for string searching, especially when precise and predictable performance is required.

The choice between these two algorithms should consider factors such as the nature of the problem and the specific requirements. The Rabin-Karp algorithm may be suitable for probabilistic matching tasks, while the KMP algorithm is ideal for precise and efficient string matching. Both algorithms can significantly enhance text-searching capabilities, depending on the context.



Picture 1 – Comparison of Algorithm Execution Time

B) Comparison of Algorithm Execution memory

In the analysis of the Rabin-Karp and Knuth-Morris-Pratt (KMP) algorithms, we assessed their memory usage in various experiments. The memory utilization of an algorithm is crucial, particularly when dealing with large data sets, as it can affect system performance and scalability.

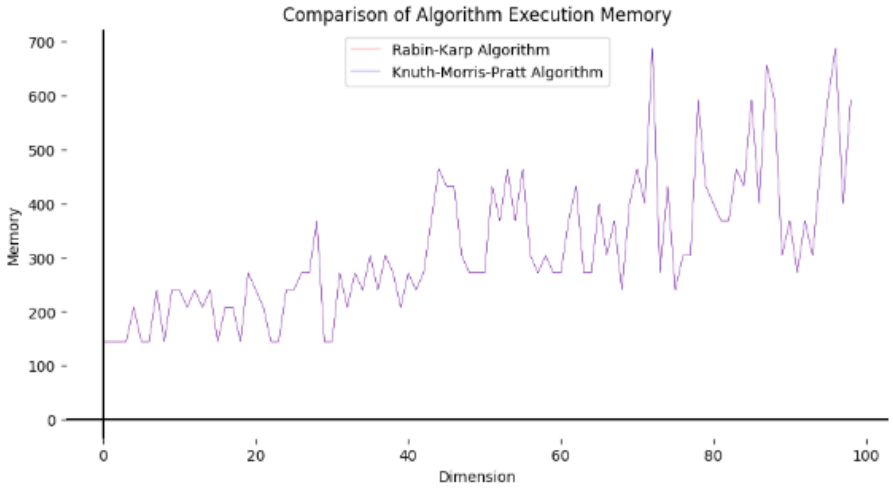
Rabin-Karp Algorithm:

The Rabin-Karp algorithm relies on a rolling hash function that generates hash values for both the pattern and substrings of the text. Its memory consumption is primarily determined by the size of the hash table used for storing the computed hash values. The memory requirements are proportional to the pattern size (M) and the size of the hash table. As a probabilistic algorithm, the Rabin-Karp may require more memory in cases of frequent hash collisions.

Knuth-Morris-Pratt (KMP) Algorithm. The KMP algorithm uses additional data structures, such as the prefix table (or failure function), which is precomputed from the pattern. The memory consumption is directly related to the size of the pattern (M) and the auxiliary data structures needed to implement the KMP algorithm. However, it is generally more memory-efficient than the Rabin-Karp algorithm, as it does not rely on a hash table and its memory usage remains stable regardless of text size.

In summary, the Knuth-Morris-Pratt (KMP) algorithm is typically more memory-efficient compared to the Rabin-Karp algorithm. Its memory requirements are primarily determined by the pattern's size and the auxiliary data structures used for pattern matching.

When choosing between these algorithms, it's essential to consider memory constraints and the specific demands of your application. If memory efficiency is a critical factor, especially for large texts, the KMP algorithm is a preferred choice. However, for probabilistic or approximate matching tasks, the Rabin-Karp algorithm can be a valuable option despite its potentially higher memory consumption in certain situations.



Picture 2 – Comparison of Algorithm Execution Memory

C) Comparison of Algorithm Execution Iterations

Rabin-Karp Algorithm:

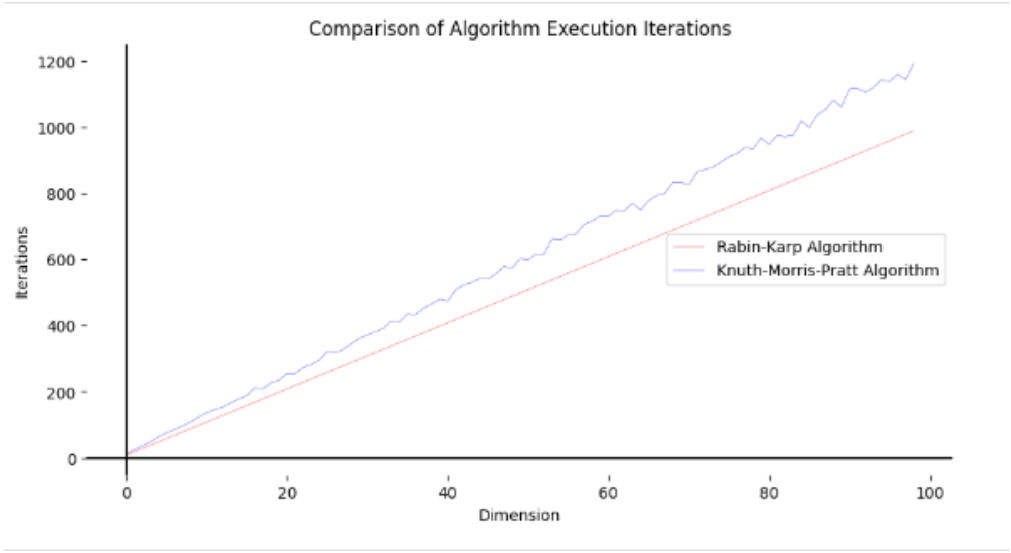
The Rabin-Karp algorithm uses a rolling hash function to slide a window of the pattern's length over the text. In each iteration, it computes hash values for the current window and compares them with the hash value of the pattern. When a hash match is found, it performs a character-by-character comparison to confirm the match, which might require additional iterations.

Knuth-Morris-Pratt (KMP) Algorithm:

The KMP algorithm employs a prefix table (or failure function) to efficiently skip portions of the text when a mismatch occurs. The number of iterations is determined by the length of the text and the number of character comparisons required to find occurrences of the pattern. KMP minimizes unnecessary character comparisons by using the information stored in the prefix table.

In general, the number of iterations required by the Rabin-Karp algorithm can vary depending on factors like hash collisions and text properties. It might require more iterations when hash collisions occur. The KMP algorithm, on the other hand, is designed to minimize unnecessary iterations, especially in cases of frequent mismatches.

The Knuth-Morris-Pratt (KMP) algorithm often outperforms the Rabin-Karp algorithm in terms of the number of iterations required to find pattern occurrences in text. KMP's efficiency comes from its ability to skip over portions of the text efficiently based on the information stored in the prefix table.



Picture 3 – Comparison of Algorithm Execution Iterations

Conclusions

In conclusion, the analysis and implementation of two string-searching algorithms, the Rabin-Karp and Knuth-Morris-Pratt (KMP) algorithms, provided valuable insights into their time and space complexities, design techniques, and real-world performance.

The Rabin-Karp algorithm, leveraging probabilistic hashing, offers a time complexity of O(N + M) on average. It exhibits efficiency in identifying potential matches in text, making it a valuable tool for string searching. However, in the worst case, when hash collisions are frequent, its time complexity may increase to O (N * M).

The KMP algorithm, with its linear time complexity of O (N + M) in all cases, proved to be a highly efficient string-searching algorithm. Its design technique, based on the failure function, enables it to avoid redundant character comparisons, making it suitable for various text-searching tasks.

Ultimately, the choice between these algorithms depends on the specific requirements and characteristics of the problem at hand. The Rabin-Karp algorithm excels when probabilistic matching is acceptable, while the KMP algorithm is ideal for cases where precise and efficient string matching is necessary.

Through experimentation and analysis, it becomes evident that both algorithms offer valuable solutions for different scenarios and can significantly enhance text-searching capabilities. Understanding their strengths and weaknesses is crucial for making informed decisions when dealing with string-searching challenges.

Appendix

GitHub: site. – URL:
https://github.com/MrSimple07/AbdurakhimovM_Algorithms_ITMO/tree/main/Task8