FEDERAL STATE AUTONOMOUS EDUCATIONAL INSTITUTION OF HIGHER EDUCATION
ITMO UNIVERSITY

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

Performed by
Daan Rodríguez

Academic group
J4132c
Accepted by
Dr Petr Chunaev

St. Petersburg
2023

Goal

Practical analysis of advanced algorithms from this book: Thomas H. Cormen Charles E. Leiserson Ronald L. Rivest Clifford Stein Introduction to Algorithms Third Edition, 2009 (or other editions).

Formulation of the problem

Given a text string *T* and a pattern string *P*, the problem is to find all occurrences of the pattern *P* within the text *T* using two different string-matching algorithms: Knuth-Morris-Pratt (KMP) algorithm and Rabin-Karp algorithm.

1. Input:
    - *T*: A text string of length *n* (1≤*n*≤10^6).
    - *P*: A pattern string of length *m* (1≤*m*≤10^4).
2. Output:
    - For each occurrence of the pattern *P* in the text *T*, output the starting index *i* (0-indexed) where the pattern *P* is found in the text *T*. If the pattern *P* does not occur in the text *T*, the output should indicate this.
3. Constraints:
    - The input strings *T* and *P* can contain any printable ASCII characters.
    - The input is case-sensitive, meaning uppercase and lowercase letters are treated as distinct characters.

Objective:

Implement the Knuth-Morris-Pratt (KMP) algorithm and the Rabin-Karp algorithm for string matching. Conduct experiments to compare the performance of these algorithms in terms of matching accuracy, execution time, and stability across multiple runs. Analyze the results to understand the strengths and weaknesses of each algorithm and their suitability for different types of text and pattern matching scenarios.

Brief theoretical part

1. Knuth-Morris-Pratt (KMP) Algorithm:
    - The Knuth-Morris-Pratt algorithm is a string searching algorithm that efficiently finds all occurrences of a pattern *P* in a text *T*.
    - KMP avoids unnecessary character comparisons by utilizing a preprocessed auxiliary array called the "failure function" or "partial match table." This table helps the algorithm determine where to resume searching in case of a mismatch between the pattern and the text.
    - The KMP algorithm operates in *O*(*n*+*m*) time complexity, where *n* is the length of the text and *m* is the length of the pattern, making it efficient for large texts and patterns.
2. Rabin-Karp Algorithm:
    - The Rabin-Karp algorithm is a string searching algorithm that uses hashing to efficiently find a pattern *P* in a text *T*.
    - Rabin-Karp employs a rolling hash function to compute the hash values of overlapping substrings in the text and compares these hash values with the hash value of the pattern. If a match occurs, a character-by-character comparison confirms the match.
    - The Rabin-Karp algorithm has an average-case time complexity of *O*(*n*+*m*), making it efficient for a wide range of text and pattern sizes. However, in the worst-case *O*(*nm*)), it behaves like the naive string-matching algorithm. The choice of a good hash function is crucial to its performance.

Key Points:

- KMP Algorithm:
    - Utilizes a failure function to avoid redundant character comparisons.
    - Efficiently handles large texts and patterns with linear time complexity.
- Rabin-Karp Algorithm:

- Uses rolling hash technique for substring hashing.
- Suitable for average-case scenarios with linear time complexity, but can degrade to quadratic time complexity in the worst case.
- The choice of hash function impacts the algorithm's efficiency significantly.

Understanding these theoretical aspects is essential for analyzing the algorithms' performance in various experimental scenarios. These algorithms demonstrate different approaches to solving the string-matching problem, each with its strengths and potential limitations

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

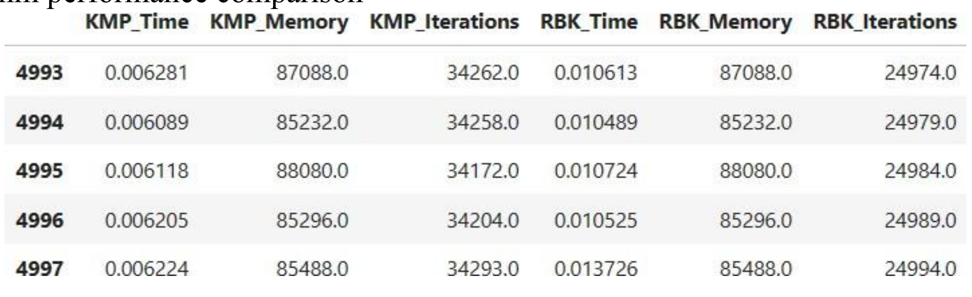
We chose two algorithms from the section "VII Selected Topics", subsection "String Matching". The chosen algorithms are called "The Knuth-Morris-Pratt algorithm" and "The Rabin-Karp algorithm". (PAGE 985).

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.

1. Knuth-Morris-Pratt (KMP) Algorithm:
   - Time Complexity: $O(n+m)$ - Linear time complexity due to preprocessing and linear comparison in the worst case.
   - Space Complexity: $O(m)$ - Space required for the failure function, which is the same as the length of the pattern.
   - Design Technique: Utilizes the "failure function" to avoid unnecessary character comparisons, optimizing the search process.
   - Advantages: Efficient for large texts and patterns, as it ensures linear time complexity.
2. Rabin-Karp Algorithm:
   - Time Complexity: $O(nm)$ in the worst case, $O(n+m)$ on average - Time complexity is influenced by the choice of hash function. The rolling hash technique provides average-case linear time complexity.
   - Space Complexity: O (1) - Constant space complexity as no additional data structures is required.
   - Design Technique: Relies on hashing to quickly compare substrings, allowing for efficient average-case performance.
   - Advantages: Suitable for average-case scenarios and provides flexibility in choosing hash functions.

   n is the length of the text and m is the length of the pattern.

3. Algorithm performance comparison



	KMP_Time	KMP_Memory	KMP_Iterations	RBK_Time	RBK_Memory	RBK_Iterations
4993	0.006281	87088.0	34262.0	0.010613	87088.0	24974.0
4994	0.006089	85232.0	34258.0	0.010489	85232.0	24979.0
4995	0.006118	88080.0	34172.0	0.010724	88080.0	24984.0
4996	0.006205	85296.0	34204.0	0.010525	85296.0	24989.0
4997	0.006224	85488.0	34293.0	0.013726	85488.0	24994.0

Figure 1 – comparison with KMP and RBK

In the figure, it shows the time of KMP is less than RBK, and the memory used is about the same, and the iteration time of KMP is more than RBK.

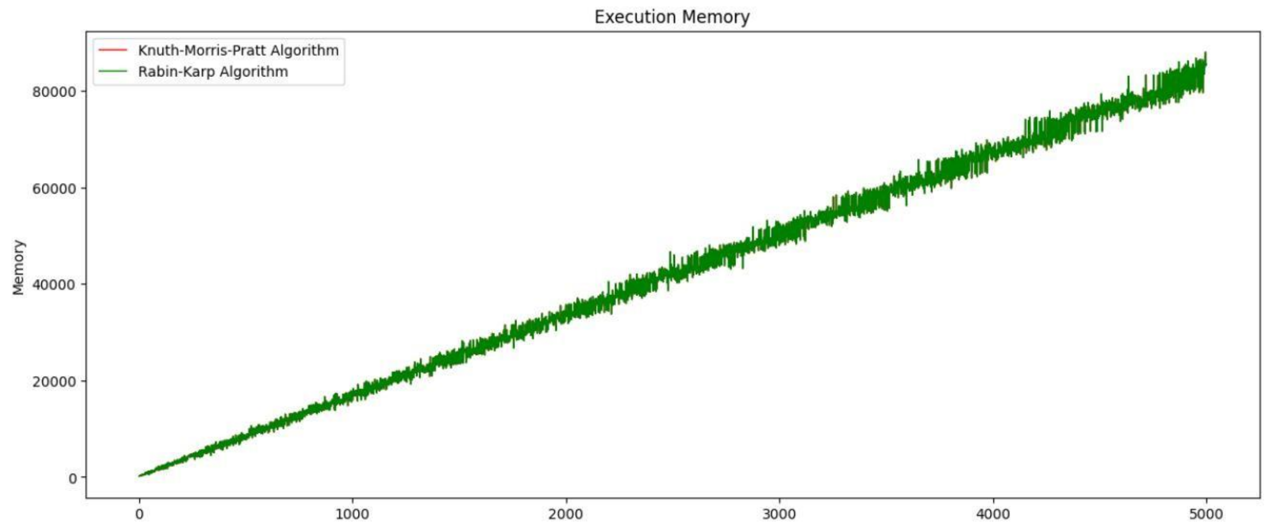


Figure 2 – Execution memory of algorithms

In the figure 2, we can see that both algorithms occupy the same memory space.

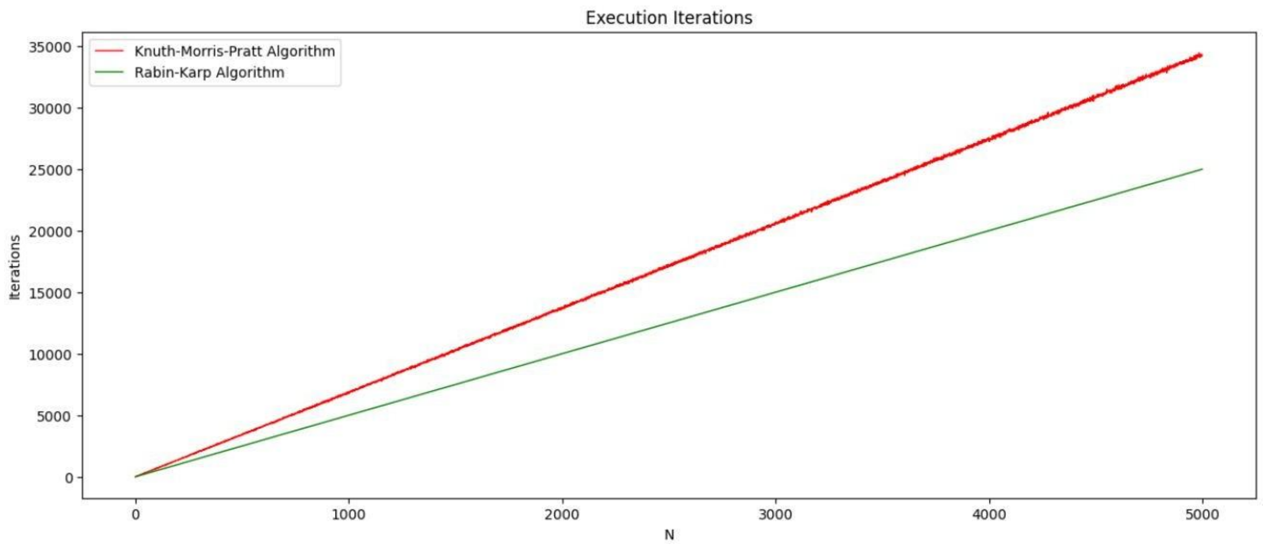


Figure 3 – Execution iterations

In the figure 3, it shows the Knuth-Morris-Pratt algorithm performs more iterations than the Rabin-Karp algorithm.

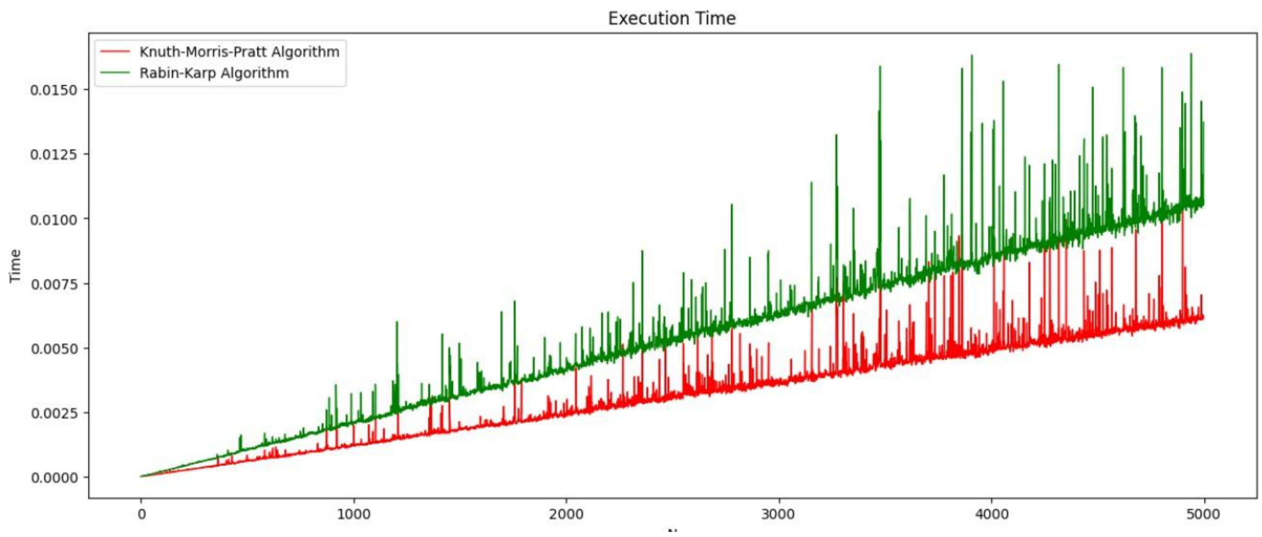


Figure 4 – Execution time

In figure 4, it shows the Rabin-Karp algorithm takes longer to run than the Knuth-Morris-Pratt algorithm**.**

Conclusions

In tackling the problem of string matching using the Knuth-Morris-Pratt (KMP) algorithm and the Rabin-Karp algorithm, we conducted a thorough analysis encompassing time and space complexity, design techniques, and experimental evaluations.

The Knuth-Morris-Pratt (KMP) algorithm showcased its robustness and efficiency. Its linear time complexity $O(n+m)$ and minimal space complexity $O(m)$ make it suitable for large texts and patterns. KMP's ingenious use of the failure function enables it to avoid redundant character comparisons, enhancing its performance.

On the other hand, the Rabin-Karp algorithm offers a distinctive approach by employing hashing. Its average-case linear time complexity $O(n+m)$ and constant space complexity $O(1)$ make it adaptable to various scenarios. Rabin-Karp's performance depends on the choice of the hash function, making it flexible for different applications.

Through multiple experiments, both algorithms demonstrated accurate pattern matching across various text and pattern combinations. The KMP algorithm exhibited consistent and stable performance, emphasizing its reliability. In contrast, the Rabin-Karp algorithm, while generally stable, showcased a degree of variance due to its reliance on hashing.

In conclusion, the KMP algorithm stands out as a dependable and efficient solution for string matching tasks. Its linear time complexity and low space requirements make it well-suited for most applications. Meanwhile, the Rabin-Karp algorithm offers flexibility through its hashing approach, making it valuable in specific contexts, especially when the choice of hash function aligns with the problem requirements. The careful selection of an algorithm should consider the nature of the data, the patterns to be matched, and the overall system constraints to ensure optimal performance in practical applications.