COMPARISON OF STRING MATCHING ALGORITHMS

Yusuf Şahin	150200016
Mustafa Can Çalışkan	150200097
Mehmet Ali Balıkçı	
Metin Ertekin Küçük	
Yusuf Emir Sezgin	150200066

OVERVIEW

- **Context:**
    - *Motivation*
    - *Methods and Design*
    - *Results and Analysis*

- **Purpose:**
    - *Experimental analysis of string matching algorithms*
        - *Brute-Force*
        - *Rabin-Karp*
        - *Knuth-Morris-Pratt*

MOTIVATION

- **The String Matching Problem**

  A fundamental problem often encountered in the field of text processing.

- **Brute Force Approach?**

  Correct! Guarantees finding a match if it exists.

  Inefficient! Requires excessive time and resources, especially with large texts.

- **Thus, efficiency is key.**

  Computing time and memory are limited resources.

  Computers are fast — But not infinitely fast.

  Memory is affordable — But not infinite.

- **Therefore, it's crucial to optimize algorithm efficiency**

  Develop innovative algorithms like Rabin-Karp and KMP that use advanced techniques (hashing, prefix computation) to improve performance.

  Enhance existing algorithms to better handle modern computing challenges.

- **Aim of this Presentation**

  To explore how Rabin-Karp and KMP algorithms offer solutions that are not only correct but also more efficient than brute force for practical applications.

METHODS AND DESIGN

- Implement code in C++ for
  - *Brute Force Algortihm*
  - *Rabin-Karp Algorithm*
  - *Knuth-Morris-Pratt (KMP) Algorithm*
- Test code on input sequences
  - *Text Length: [0, 200000]*
  - *Pattern Length: [0, 200000]*

BRUTE-FORCE

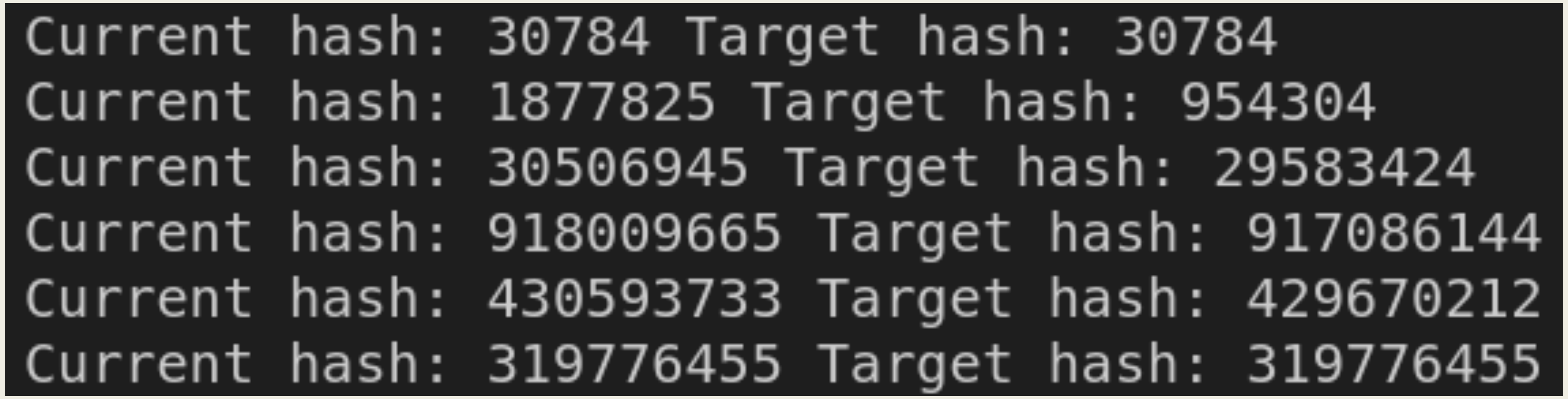
- The brute force algorithm is a straightforward approach to string matching. It involves comparing every possible substring of the text with the pattern, sliding one character at a time.

- The time complexity of the brute force algorithm is $O((n - m + 1) * m)$, where:
  - *n is the length of the text*
  - *m is the length of the pattern*

RABIN-KARP

- The Rabin-Karp algorithm is a probabilistic string matching algorithm that uses hashing to efficiently search for a pattern within a text.

- The worst time complexity of the Rabin-Karp algorithm is *O(n * m),* where:
  - *n is the length of the text*
  - *m is the length of the pattern*



```
Current hash: 30784 Target hash: 30784
Current hash: 1877825 Target hash: 954304
Current hash: 30506945 Target hash: 29583424
Current hash: 918009665 Target hash: 917086144
Current hash: 430593733 Target hash: 429670212
Current hash: 319776455 Target hash: 319776455
```

- Pattern:
  - string p = "aaaa",

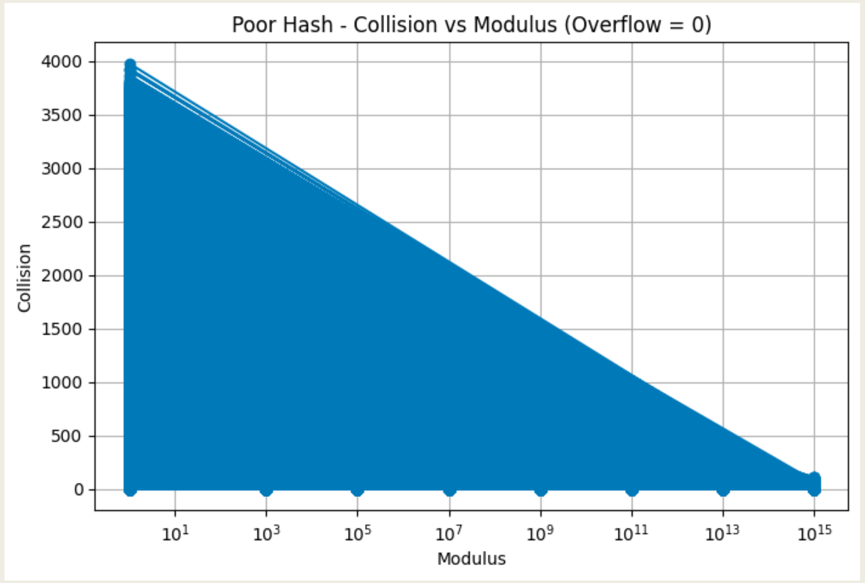
- Text:
  - string t = "aaaabaaaa";

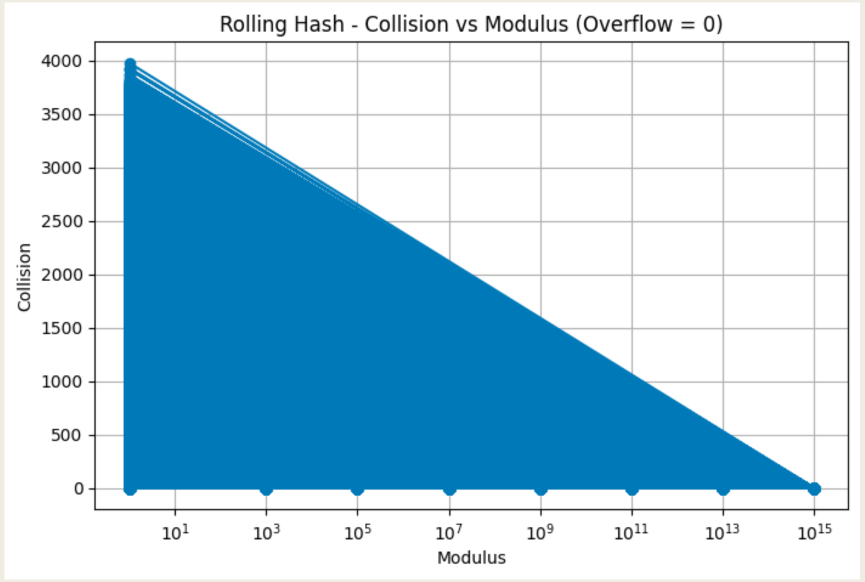
KNUTH-MORRIS-PRATT (KMP)

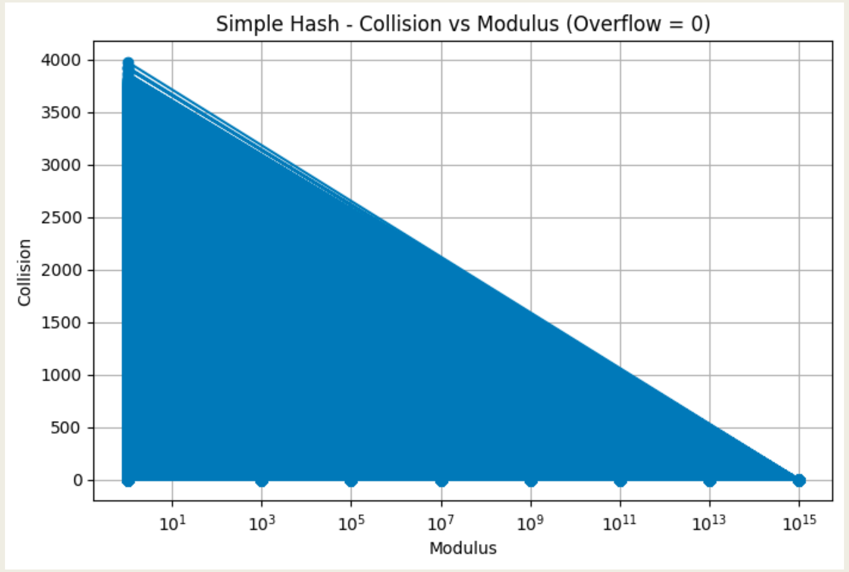
- The KMP algorithm is an efficient string matching algorithm that avoids unnecessary comparisons by precomputing a partial match table based on the pattern.

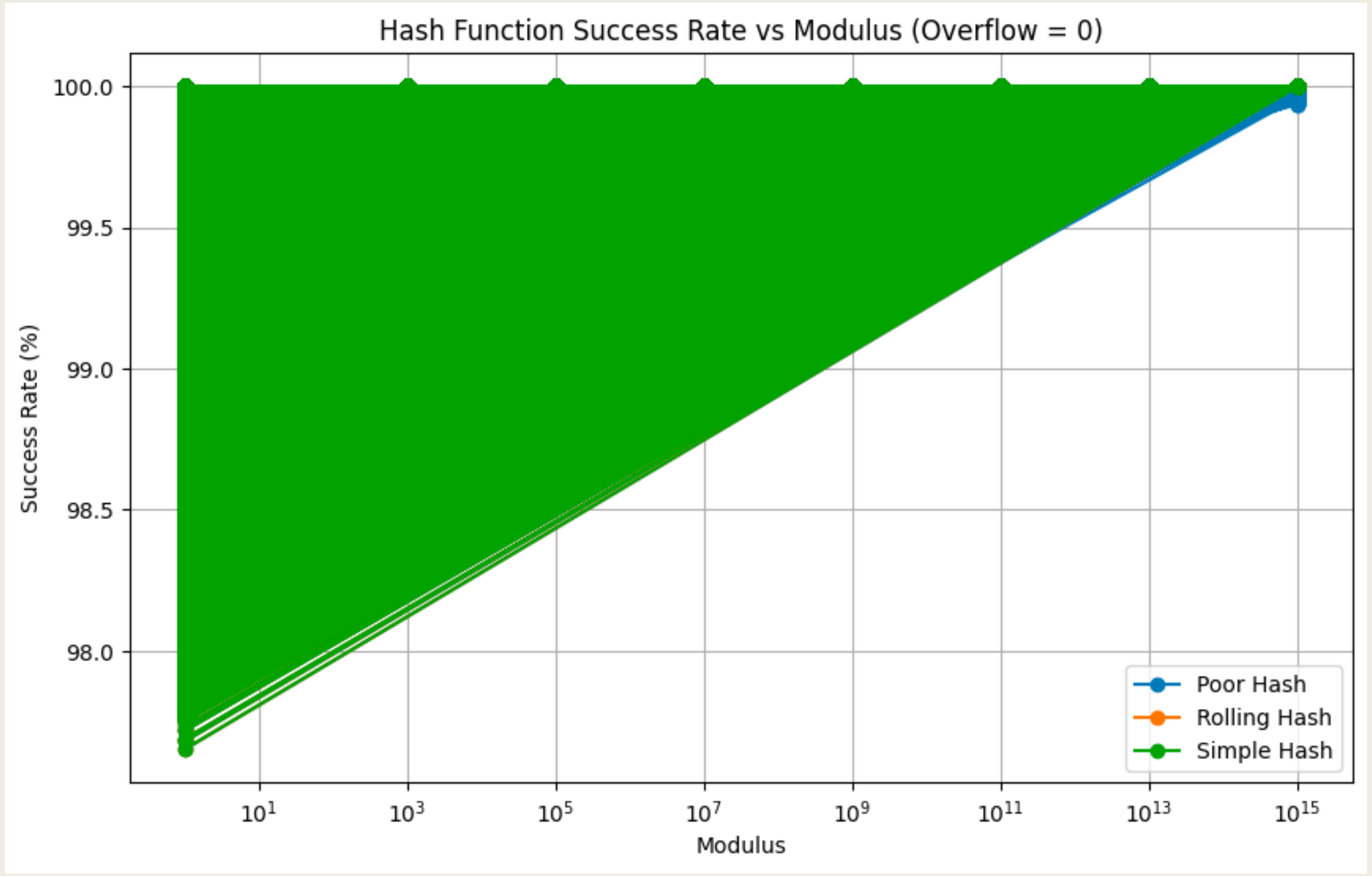
- The time complexity of the KMP algorithm is $O(n + m)$, where:
  - *n is the length of the text*
  - *m is the length of the pattern*

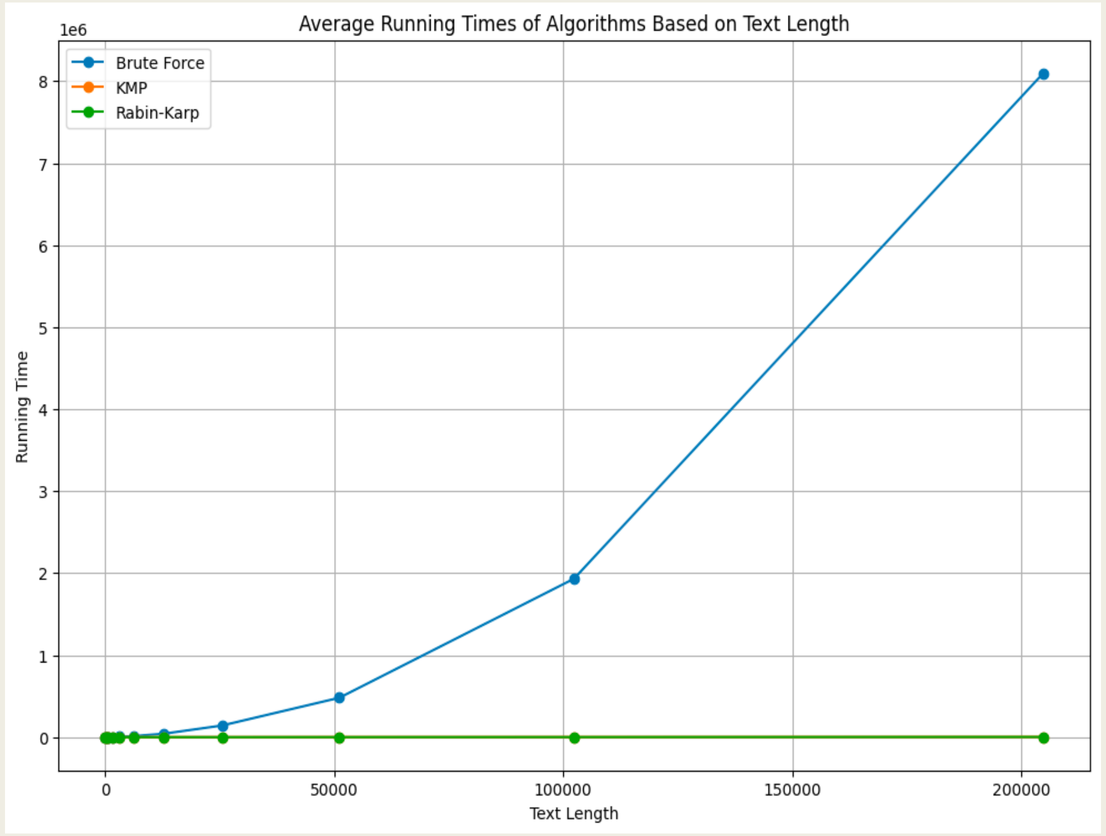
ANALYSIS

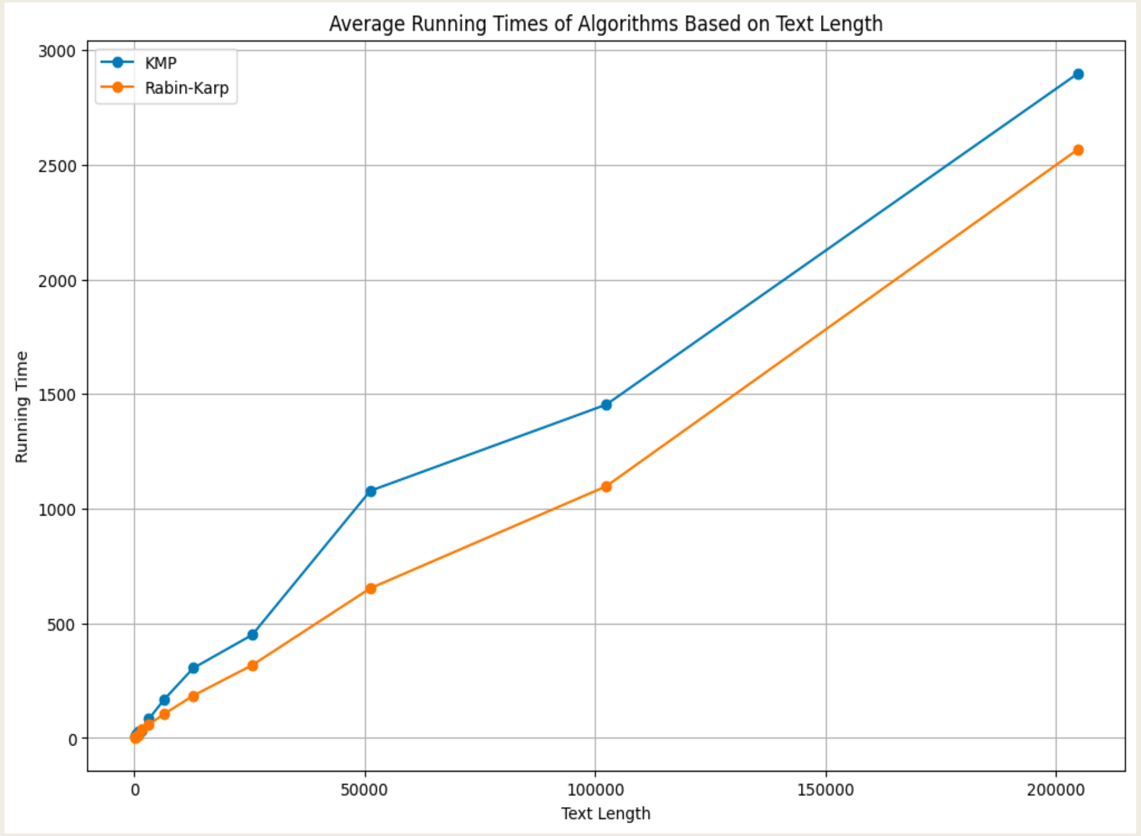

Poor Hash - Collision vs Modulus (Overflow = 0)
Collision
Modulus
4000
3500
3000
2500
2000
1500
1000
500
0
10^1
10^3
10^5
10^7
10^9
10^{11}
10^{13}
10^{15}


Rolling Hash - Collision vs Modulus (Overflow = 0)
Collision
Modulus
4000
3500
3000
2500
2000
1500
1000
500
0
10^1
10^3
10^5
10^7
10^9
10^{11}
10^{13}
10^{15}


Simple Hash - Collision vs Modulus (Overflow = 0)
Collision
Modulus
4000
3500
3000
2500
2000
1500
1000
500
0
10^1
10^3
10^5
10^7
10^9
10^{11}
10^{13}
10^{15}


Hash Function Success Rate vs Modulus (Overflow = 0)
Success Rate (%)
100.0
99.5
99.0
98.5
98.0
Modulus
10^1
10^3
10^5
10^7
10^9
10^{11}
10^{13}
10^{15}
Poor Hash
Rolling Hash
Simple Hash


Average Running Times of Algorithms Based on Text Length
1e6
Brute Force
KMP
Rabin-Karp
Running Time
Text Length
0
50000
100000
150000
200000
0
1
2
3
4
5
6
7
8


Average Running Times of Algorithms Based on Text Length
KMP
Rabin-Karp
Running Time
Text Length
0
50000
100000
150000
200000
0
500
1000
1500
2000
2500
3000

CONCLUSION

- Brute Force Algorithm:
  - *While simple and straightforward, the brute force approach is inefficient for large texts or patterns due to its high time complexity. It systematically checks for the pattern at every possible position in the text, which can be computationally expensive.*
- Rabin-Karp Algorithm:
  - *This algorithm uses hashing to find any set of pattern matches in text, allowing for a faster average-case time complexity. It is especially efficient when dealing with multiple patterns or sets of multiple characters. However, its performance can degrade in the worst case to match that of the brute force approach due to spurious hits.*
- Knuth-Morris-Pratt (KMP) Algorithm:
  - *KMP excels by eliminating the needless re-checking of previously matched characters. Its preprocessing phase constructs a partial match table that speeds up the searching phase significantly, ensuring a worst-case time complexity that is linear with respect to the text length.*
- Key Takeaway:
  - *The choice of algorithm depends on the specific requirements of the application, including text size, pattern characteristics, and computational resources. For optimal performance, it is crucial to consider these factors when selecting a string matching algorithm.*

References

- Detailed KMP Algorithm Visualization: https://www.youtube.com/watch?v=pu2aO_3R118