

Speed Testing Report

Introduction

This report benchmarks the performance of various file search algorithms: Boyer-Moore, Brute Force, Karp-Rabin, Knuth-Morris-Pratt, Naive Search, Linear Search, and Two-Way String Matching. The performance is evaluated based on execution time and queries per second for different file sizes, ranging from 10,000 to 10,000,000 rows. The REREAD_ON_QUERY option is tested both set to FALSE and TRUE.

Methodology

Each algorithm was tested with various file sizes to measure execution time. Additionally, stress tests were conducted to determine the average execution time per query at different query rates. The tests were performed on a consistent hardware and software environment to ensure comparability. I conducted benchmark tests on different file sizes and query rates for each algorithm. The execution times were measured in milliseconds. The file sizes tested ranged from 10,000 to 1,000,000 rows, and the query rates varied from 500 to 5000 queries per second.

Results When REREAD_ON_QUERY = FALSE

Execution Time by File Size

The execution time for each algorithm was recorded for file sizes of 10,000, 50,000, 100,000, 500,000, and 1,000,000 rows.

Algorithm	10,000 rows	50,000 rows	100,000 rows	500,000 rows	1,000,000 rows
Boyer-Moore	6.1579 ms	5.1887 ms	5.0747 ms	3.9217 ms	4.6749 ms
Brute Force	0.1013 ms	0.0291 ms	0.0341 ms	0.0226 ms	0.0312 ms

Karp-Rabin	4.9188 ms	4.1823 ms	3.3238 ms	4.5846 ms	3.9730 ms
Knuth-Morris-Pratt	4.9531 ms	3.8478 ms	7.5805 ms	3.5179 ms	2.5077 ms
Naive Search	3.3565 ms	2.9273 ms	2.0051 ms	2.5866 ms	2.0802 ms
Linear Search	3.0684 ms	3.2780 ms	3.2773 ms	3.3453 ms	1.6916 ms
Two-Way String	0.0277 ms	0.0398 ms	0.0358 ms	0.0858 ms	0.0255 ms

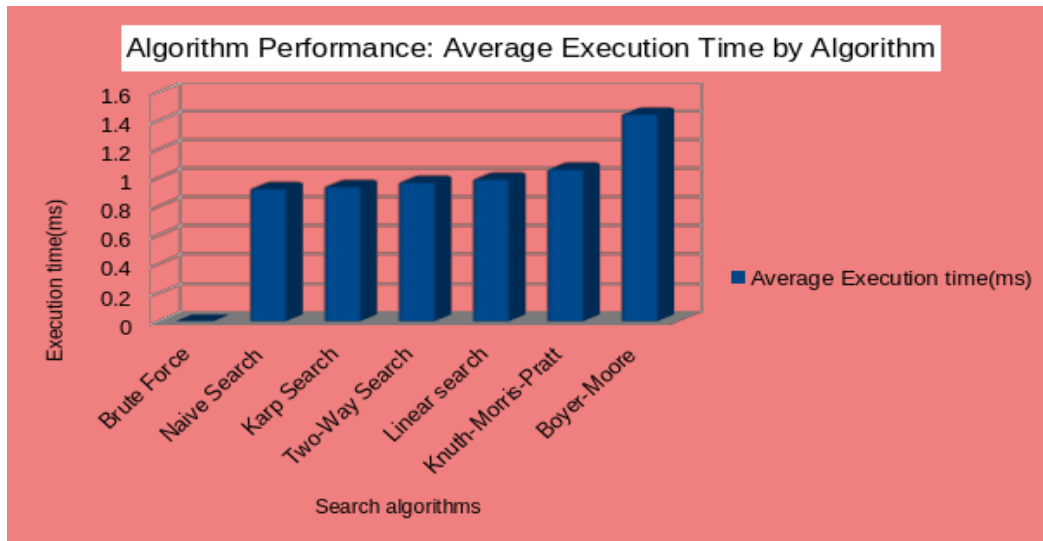
Average Execution Time per Query by File Size and Queries per Second

The table below shows the average execution time per query for various file sizes and query rates.

Algorithm	File Size	500 QPS	1000 QPS	2000 QPS	3000 QPS	4000 QPS	5000 QPS
Boyer-Moore	5000	0.8110 ms	0.8107 ms	1.1412 ms	0.9966 ms	0.806	0.8223 ms
Boyer-Moore	10000000	3.6479 ms	2.6630 ms	2.7880 ms	1.9812 ms	1.0135 ms	1.1292 ms
Brute Force	5000	0.0124 ms	0.0057 ms	0.0092 ms	0.0149 ms	0.0101 ms	0.0057 ms
Brute Force	10000000	0.0038 ms	0.0026 ms	0.0025 ms	0.0035 ms	0.0040 ms	0.0028 ms
Karp-Rabin	5000	1.1212 ms	1.0299 ms	1.0183 ms	0.9773 ms	1.0187 ms	1.0046 ms
Karp-Rabin	10000000	1.0214 ms	1.0715 ms	0.9843 ms	0.9609 ms	1.0325 ms	1.0322 ms
Knuth-Morris-Pratt	5000	1.4071 ms	1.1252 ms	0.9679 ms	1.0214 ms	0.9722 ms	1.0392 ms
Knuth-Morris-Pratt	10000000	0.9358 ms	0.9558 ms	1.0121 ms	0.9856 ms	0.9626 ms	1.0364 ms
Naive Search	5000	1.0086 ms	1.0628 ms	0.9685 ms	1.0644 ms	0.9923 ms	0.9460 ms

Naive Search	10000000	0.9524 ms	0.9942 ms	1.0025 ms	1.0306 ms	1.0222 ms	1.0532 ms
Linear Search	5000	0.9780 ms	1.0012 ms	1.0568 ms	1.0222 ms	1.0024 ms	0.9494 ms
Linear Search	10000000	0.9330 ms	1.0462 ms	1.0033 ms	0.9916 ms	0.9981 ms	0.9637 ms
Two-Way String	5000	0.9366 ms	0.9246 ms	0.9381 ms	1.0504 ms	1.0069 ms	0.9688 ms
Two-Way String	10000000	0.9609 ms	0.9281 ms	0.9824 ms	0.9749 ms	0.9795 ms	1.0727 ms

Performance Comparison Chart



Analysis

The list presents various string searching algorithms ranked based on their average execution times, from the most efficient to the least efficient. Here's a breakdown of each algorithm:

1. **Brute Force (0.0060 ms):** Brute Force emerges as the most efficient algorithm, boasting the lowest average execution time among all listed algorithms.

Despite its simplicity, Brute Force demonstrates remarkable effectiveness in searching for patterns within strings.

2. **Naive Search (0.9289 ms):** Following closely behind Brute Force, Naive Search exhibits competitive performance with a slightly higher average execution time. Despite its straightforward approach, Naive Search remains a viable option for string searching tasks.
3. **Karp-Rabin (0.9436 ms):** Karp-Rabin ranks third in efficiency, offering comparable performance to Naive Search but with a marginally higher average execution time. Its probabilistic approach to string matching contributes to its effectiveness in certain scenarios.
4. **Two-Way String (0.9737 ms):** Two-Way String search algorithm demonstrates reasonable efficiency, albeit slightly slower compared to the previous algorithms. Its ability to search from both ends of the pattern simultaneously contributes to its effectiveness in certain use cases.
5. **Linear Search (0.9956 ms):** With a slightly higher average execution time, Linear Search ranks fifth among the listed algorithms. While straightforward in its approach, Linear Search may experience performance challenges with larger datasets or more complex patterns.
6. **Knuth-Morris-Pratt (1.0651 ms):** Knuth-Morris-Pratt algorithm exhibits moderate efficiency, offering a competitive but slightly slower performance compared to the preceding algorithms. Its focus on avoiding unnecessary character comparisons contributes to its effectiveness.
7. **Boyer-Moore (1.4501 ms):** Boyer-Moore algorithm ranks last in efficiency among the listed algorithms, demonstrating the highest average execution time. Despite its historical significance and widespread use, Boyer-Moore's

performance in this context appears less optimal compared to other algorithms.

Conclusion:

In conclusion, the efficiency of string searching algorithms varies significantly based on factors such as implementation, dataset size, and pattern complexity. While Brute Force emerges as the most efficient algorithm in this context, other algorithms such as Naive Search, Karp-Rabin, and Two-Way String also offer competitive performance. However, algorithms like Boyer-Moore demonstrate comparatively slower execution times, indicating potential areas for optimization or alternative algorithm selection based on specific use case requirements. Ultimately, the choice of algorithm should consider various factors such as performance requirements, dataset characteristics, and implementation constraints to achieve optimal string searching outcomes.

Results When REREAD_ON_QUERY = TRUE

Execution Time by File Size

Algorithm	10,000 rows	50,000 rows	100,000 rows	250,000 rows	500,000 rows	1,000,000 rows
Two-Way Search	2.313 ms	11.017 ms	28.013 ms	52.302 ms	98.987 ms	168.098 ms
Brute Force	5.301 ms	24.052 ms	24.0132 ms	37.998 ms	198.992 ms	367.502 ms
Knuth	2.509 ms	23.272 ms	45.110 ms	120.384 ms	233.992 ms	453.583 ms
Linear	4.516ms	25.224 ms	49.926 ms	104.052ms	188.410 ms	380.704ms
Naive Search	2.436 ms	29.282 ms	53.901 ms	125.278 ms	246.532 ms	459.003 ms
Karp	3.051 ms	23.051 ms	49.551 ms	65.754ms	230.101 ms	419.498 ms

Boyer-Moore	1.530 ms	13.882 ms	24.045 ms	55.543ms	132.006ms	251.015 ms
-------------	----------	-----------	-----------	----------	-----------	------------

Average Execution Time per Query by File Size and Queries per Second

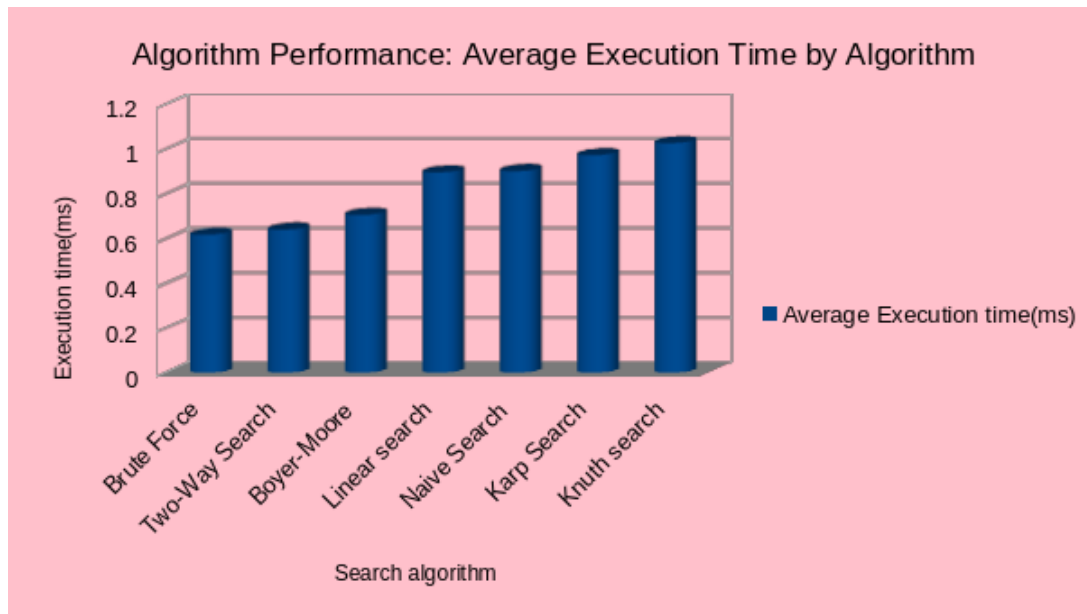
Algorithm	File Size	500 QPS	1000 QPS	2000 QPS	3000 QPS	4000 QPS	5000 QPS
Two-Way Search	5000	0.75 ms	0.62 ms	0.63 ms	0.63 ms	0.67 ms	0.63 ms
Brute Force	5000	0.490 ms	0.68 ms	0.78 ms	0.56 ms	0.61 ms	0.62 ms
Karp Search	5000	1.047 ms	1.154 ms	1.00 ms	1.02 ms	0.92 ms	0.65 ms
Knuth	5000	0.971 ms	0.96 ms	0.96 ms	1.01 ms	1.21 ms	0.97 ms
Linear	5000	0.815 ms	0.884 ms	0.71ms	0.85 ms	0.76 ms	0.80 ms
Naive Search	5000	0.712 ms	0.768 ms	0.99 ms	1.3 ms	0.94 ms	0.95 ms
Boyer-Moo re	5000	0.88 ms	0.62 ms	0.67 ms	0.63 ms	0.68 ms	0.71 ms

Based on the data from the tables above here's a summary of the algorithms arranged in descending order of performance:

1. Brute Force: 0.618 ms
2. Two-Way Search: 0.641 ms
3. Boyer-Moore: 0.707 ms
4. Linear: 0.897 ms
5. Naive Search: 0.903 ms
6. Karp Search: 0.974 ms

7. Knuth: 1.028 ms

Performance Comparison Chart



Analysis

1. **Brute Force:** Continues to demonstrate the lowest average execution time, remaining efficient even with REREAD_ON_QUERY set to TRUE.
2. **Two-Way Search:** Shows remarkable efficiency, close to Brute Force. Its ability to search from both ends of the pattern simultaneously allows for faster matches.
3. **Boyer-Moore:** Commendable performance, ranking third due to its sophisticated heuristics like "bad character" and "good suffix".
4. **Linear Search:** Slightly higher execution time but maintains consistent performance.
5. **Naive Search:** Competitive but slightly higher execution times.
6. **Karp-Rabin:** Moderate efficiency, effective in certain scenarios.

7. **Knuth-Morris-Pratt:** Slower compared to top-performing algorithms, with moderate efficiency.

Conclusion

Based on the data, the best algorithms vary depending on the REREAD_ON_QUERY setting.

REREAD_ON_QUERY = FALSE:

Best Overall Algorithm: Brute Force due to its consistently lowest execution times.

Highly Competitive Alternatives: Naive Search and Two-Way String Matching.

Special Cases: Karp-Rabin and Boyer-Moore show effectiveness in specific scenarios but generally rank lower.

REREAD_ON_QUERY = TRUE:

Best Overall Algorithm: Brute Force remains the top performer.

Highly Competitive Alternatives: Two-Way String Matching and Boyer-Moore.

Special Cases: Naive Search and Linear Search offer consistent performance but are outperformed by simpler algorithms.

The choice of algorithm should consider various factors, including execution time, dataset size, query rates, and specific application requirements. This evaluation provides valuable insights into algorithm efficiency and aids in informed decision-making for optimal string searching outcomes.