LOBAM: A LOAD-RESISTANT HASHMAP IMPLEMENTATION BASED ON THE BATHROOM MODEL

DONGGI AHN

Abstract. Proposed by Wang[3], the bathroom model demonstrated performance improvement in open addressing by adjusting probing strategies dynamically. However, the paper did not provide a method for optimally combining multiple probing strategies. Furthermore, under high-load conditions, it has been shown that performance significantly degrades with the proposed model, which combines exponential probing and linear probing. For optimal performance, particularly under high-load conditions, we benchmarked each probing strategy that performs better under either high-load or low-load scenarios. Based on the benchmark results, we used the Kneedle Algorithm[2], an elbow-point detection method, to identify the point where performance starts to degrade sharply. Using this algorithm, we propose a method to determine the point at which the probing strategy should be switched, based on the number of entries in the hash table. In addition, we implemented a proof-of-concept version of LoBAM, a load-resistant hashmap implementation based on the bathroom model, using the proposed method.

1. INTRODUCTION

Improving hashmap efficiency has long been an important subject in data structures. Bathroom Model, proposed by Wang [3], approached this problem by adjusting probing strategies dynamically, whereas traditional approaches focused on single probing strategies. By adjusting probing strategies according to the count of elements in the hashmap, the bathroom model successfully improved lookup efficiency and worst-case probing complexity, particularly under low-load conditions. However, it has been shown that under high-load conditions, the probing count of the bathroom model grew significantly, degrading its performance. To mitigate this peak of probing steps, this paper focuses on combining probing steps optimally.

2. PREVIOUS WORKS

2.1 RESEARCHES ON PROBING METHODS

The random probing strategy [4] skips throughout the hash table using a pseudo random number until finding an unutilized area of the hash table to avoid collision in the hash table.

The elastic hashing [1] is a technique for storing useful and efficient data through elastic node management for scalability and stability. The bathroom model utilizes part of the elastic hash function's virtual multi-level structure. The funnel hashing strategy [1] is the method searching parts of the hash table, dividing the search domain every time and ending up gradually finding places in the hash table.

Date: May 27, 2025.
Konkuk University

Bathroom Model [3] uses multiple probing methods with flexibility, depending on the load of the hash table. This approach is a unique method, inspired by real-life situations, of solving the collision in the hash table. The idea that the bathroom method is giving new solutions is a persuasive concept assuming that it may be more efficient than traditional strategies. However, it was proven ineffective in cases where the number of keys inserted exceeds certain amounts. After inserting a certain amount of keys, the bathroom model seems to repeat the same pattern searching the same area of the hash table, lowering the memory usage. Therefore, the main point of this paper is to mitigate the performance degradation of the bathroom model and to advance it into a more efficient, load-resistant model.

3. METHODOLOGY

3.1 BENCHMARKING PROBING STRATEGIES

To select optimal probing strategies according to the number of elements in the hashmap, several benchmarks on probing strategies have been performed. We primarily focused on the worst-case probing counts, as the purpose of this paper is to mitigate the peak of worst-case probing counts. The performance indexes we used are:

- **Table Size**(sz_t): Size of hash table, equivalent to the size of memory allocated.
- **Element Count**(c_e): Number of elements in the hash table, pre-determined as $sz_t \times 1.5$ to make sure that hash collision occurs.
- **Worst-Case Probing Counts**(w): Worst-case probing counts in the given condition.

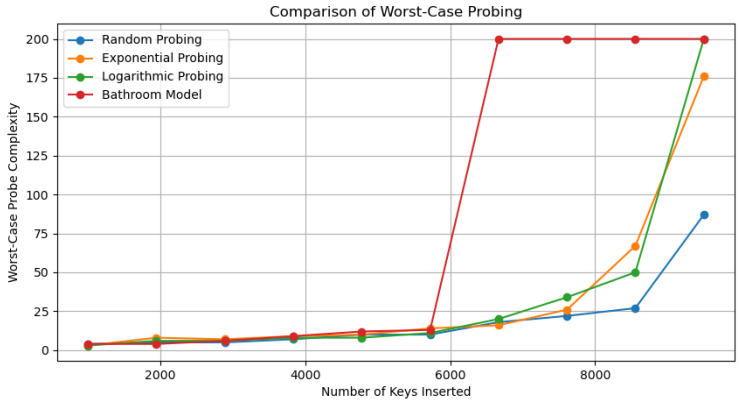


FIGURE 1. Benchmark Result

As shown in Fig 1, logarithmic probing outperforms other probing methods under low-load circumstances. However, as logarithmic probing continuously decreases its probing step as the probing count becomes larger, significant performance degradation can be found under high-load situations. Random probing provides exactly the opposite performance compared to the logarithmic method. As Andrew [4] suggested, its performance is inferior to that of logarithmic probing but continuously improves compared to exponential probing. This result suggests that using logarithmic probing under low-load conditions and then switching to random probing at a certain predetermined threshold will maximize the performance of the hashmap.

3.2 FURTHER BENCHMARK

With selected probing methods, we ran deeper benchmarks with the following additional indexes:

- **Table Size(sz_t)**: Size of hash table, equivalent to the size of memory allocated. Predetermined arithmetic sequence starting at 100, ending at 100 000, with common difference 100.
- **Worst-Case Probing Counts(w_r, w_l)**: The worst-case probing counts in the given condition with each probing strategy.
- **Optimal($O(n)$)**:

$$O(n) = \begin{cases} 1, & w_r < w_l \\ 0, & w_r > w_l \end{cases}$$

- **Total Optimal ($T_m(n)$)**: Let $m \in \{l, r\}$ be the probing method. Then,

$$T_m(n) = \begin{cases} n(C) & \text{where } C = \{O(N) = 0 \mid 0 < N \leq n\}, \quad \text{if } m = l \\ n(C) & \text{where } C = \{O(N) = 1 \mid 0 < N \leq n\}, \quad \text{if } m = r \end{cases}$$

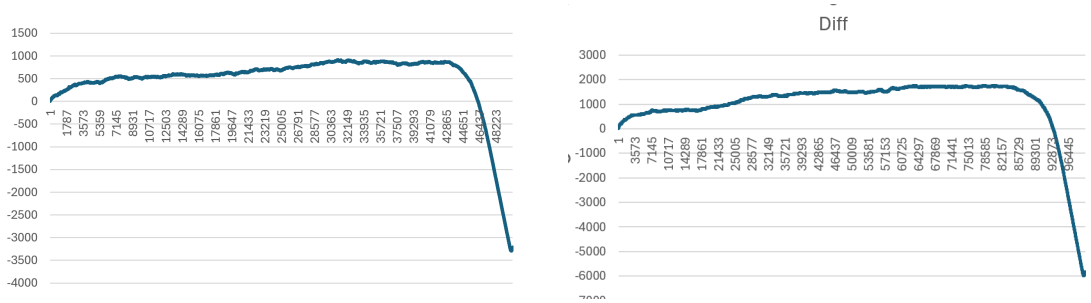
- **Difference($D(n)$)**: $T_l(n) - T_r(n)$

In particular, $D(n)$ denotes:

- $D(N) > 0$: overall performance of logarithmic probing is better on $c_e = N$ with given sz_t
- $D(N) < 0$: overall performance of random probing is better on $c_e = N$ with given sz_t

For each table size, 500 tests have been run with different pseudo-random generated elements. Each tests results were averaged accordingly based on the sz_t.

4. DETERMINING THRESHOLD



(A) Table size of 50 000 (B) Table size of 100 000

FIGURE 2. Comparison of $D(n)$ for different table sizes

As shown in Fig. 2, $D(n)$ gradually increases up to a threshold θ, and then drops sharply beyond θ. Since the slope of $D(n)$ changes significantly at $n = \theta$, traditional elbow-point detection algorithms can be used to determine the optimal value of θ. For this, we employed

the Kneedle algorithm [2]. This algorithm requires defining a straight line segment connecting the first and last data points as follows:

$$M(n) = \frac{D(sz_t - 1) - D(0)}{sz_t - 1} \cdot n$$

Using this segment line, we define the set C as

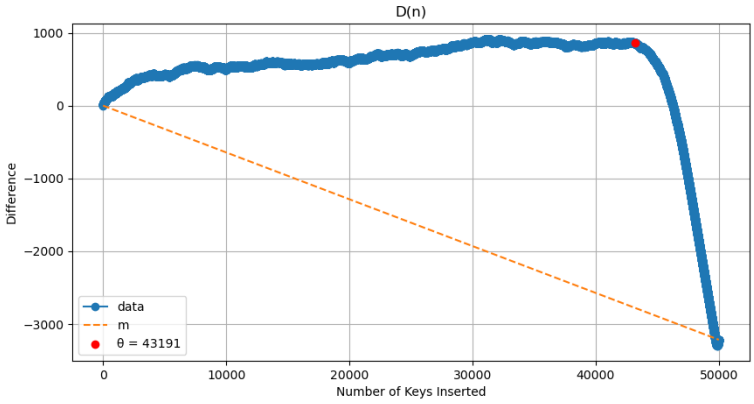
$$C = \{D(n) - M(n) \,|\, n \in \mathbb{N},\ 0 \leq n < sz_t\}$$

Then, the threshold θ is given by

$$\theta = \arg\max C$$

sz_t	m	θ
10000	-0.09470947	7895
30000	-0.06783559	25389
50000	-0.06430129	43191
70000	-0.06678667	60101
90000	-0.06158957	76514
90000	0.06158957	76514

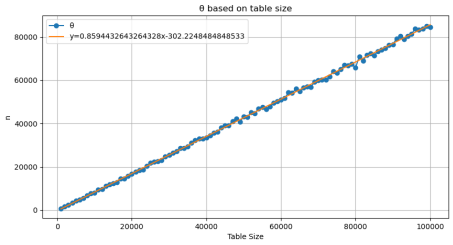
(A) Partial m, θ based on each sz_t



(B) Graph of $D(n)$ and θ for $sz_t = 50000$

FIGURE 3. Result of θ on each sz_t using Kneedle algorithm

From Fig. 3, we found there is a linear correlation between sz_t and θ. To find the correlation, we performed linear regression with sz_t as the independent variable and θ as the dependent variable.



(A) θ on sz_t

Parameter	Estimate	Std. Error
Intercept (β_0)	-302.2248	–
Slope (β_1)	0.8594	0.0025
$R^2 = 0.9992$, p-value $= 1.71 \times 10^{-153}$		

(B) Linear regression results for n vs. sz_t

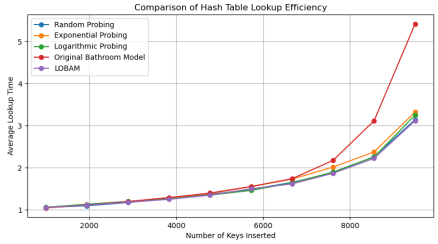
FIGURE 4. Result of linear regression

5. IMPLEMENTATION AND RESULTS

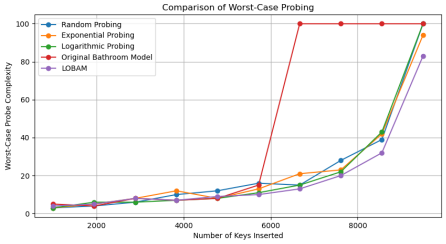
With result from Fig. 4, we implemented LoBAM model and tested its performance. Our evaluation criteria include:

- **Average Lookup Time**: The mean number of probes required for retrieval under different load factors.
- **Worst-case Complexity**: The maximum probes observed under high-load conditions, highlighting the scalability of each method.

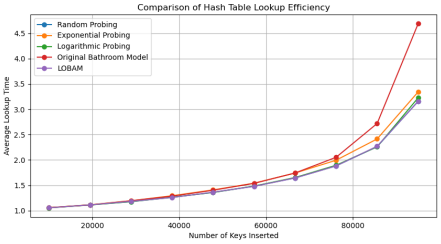
as Wang [3] suggested. Memory utilization was not included, as low memory utilization on fixed table size could imply failed insertions when reaching max probing limits.



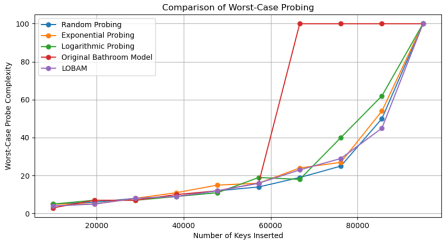
(A) Lookup efficiency under $sz_t = 10000$



(B) Worst case complexity under $sz_t = 10000$



(C) Lookup efficiency under $sz_t = 100000$



(D) Worst case complexity under $sz_t = 100000$

FIGURE 5. Benchmarks of LoBAM under different sz_t

As shown in Fig. 5, our proposed LoBAM effectively mitigates the worst-case peak time complexity in the original Bathroom Model. In additiopn, by combining the strengths of both random probing and logarithmic probing, our LoBAM outperforms each individual probing strategy under both low- and high-load conditions.

6. CONCLUSION

Bathroom Model by Wang [3] demonstrated that the efficiency of hash tables can be improved by combining multiple probing strategies. However, it did not propose a method for determining the threshold at which performance begins to degrade, resulting in a spike in worst-case probing counts beyond a certain point. This paper introduces a method for identifying such thresholds in the Bathroom Model and implements a load-resistant hashmap based on an adaptive probing strategy. We found that logarithmic probing is relatively more efficient under low load, whereas random probing performs better under high load.

Through benchmark tests on these probing strategies and the Kneedle algorithm [2], we developed a linear regression-based model that predicts the threshold where random probing begins to outperform logarithmic probing. This model effectively mitigates the peak in worst-case probing steps observed in the original Bathroom Model.

Although this may not be the most efficient approach to open addressing, especially since hash table sizes can be increased once collisions are detected, it offers a unique advantage in certain environments. Overall, in systems with limited memory capacity, such as embedded systems where resizing hash tables is not feasible, our proposed LoBAM serves as a practical alternative.

7. ACKNOWLEDGMENTS

This paper was highly inspired by Wang [3], particularly one of its future work suggestions.
All source codes can be accessed via GitHub repository: [todo]
This paper was reviewed by Taeho Kim[hessndietz@konkuk.ac.kr] and
Seungeun Lee[ttukttak17@konkuk.ac.kr]

References

[1] Martín Farach-Colton, Andrew Krapivin, and William Kuszmaul. Optimal bounds for open addressing without reordering. In *2024 IEEE 65th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 594–605, 2024.

[2] Ville Satopaa, Jeannie Albrecht, David Irwin, and Barath Raghavan. Finding a "kneedle" in a haystack: Detecting knee points in system behavior. In *2011 31st International Conference on Distributed Computing Systems Workshops*, pages 166–171, 2011.

[3] Qiantong Wang. The bathroom model: A realistic approach to hash table algorithm optimization, 2025.

[4] Andrew C. Yao. Uniform hashing is optimal. *J. ACM*, 32(3):687–693, July 1985.

Donggi Ahn
Current address: Konkuk University
Email address: flyahn06@konkuk.ac.kr