Use of Statistical Techniques for Optimizing Hashing

Pankaj Kumar Gupta[1], P. K. Tyagi[2], R. K. Agrawal[3]
[1]*Assistant Professor & Head, BCA Department, DPBS College, Anupshahr Distt. BulandShahr (UP) India.*
[2]*Professor & Head, Department of Statistics, DPBS College, Anupshahr Distt. BulandShahr (UP) India.*
[3]*Professor & Head, Department of Mathematics, DPBS College, Anupshahr Distt. BulandShahr (UP) India.*

Abstract: A process of mapping large amount of data item to smaller table with the help of some special function is known as Hashing and this special function is termed as hash function. In this research paper we are going to discuss how statistical techniques can be used for optimization of hashing. Hashing can also be known as Hashing Algorithm or Message Digest Function. Hashing can also be used to convert a range of key values into a range of indexes of an array. Hashing is used with a database to retrieve items more quickly. It can be used in the encryption and decryption of Digital Signatures.

Keywords: Hashing, complexity, load factor, collision, Reinforcement, Cryptographic, Entropy, Markov Models.

INTRODUCTION

Hashing:

A process of mapping large amount of data item to smaller table with the help of some special function is known as Hashing and this special function is termed as hash function. In this research paper we are going to discuss how statistical techniques can be used for optimization of hashing. Hashing can also be known as Hashing Algorithm or Message Digest Function. Hashing can also be used to convert a range of key values into a range of indexes of an array. Hashing is used with a database to retrieve items more quickly. It can be used in the encryption and decryption of Digital Signatures. Hash technique is used to facilitate the next level searching method when it is compared with the Sequential(i.e. Linear) or Binary search. [1]Hashing allows to update and retrieve any data entry in a constant time O(1). Constant time O(1) means the operation does not depend on the size of the data. Hashing is used with a database to enable items to be retrieved more quickly. It is used in the encryption and decryption of digital signatures.

Hash Function:

[2]A fixed process that converts a key to a hash key is known as a Hash Function. This function takes a key and maps it to a value of a certain length, which is called a Hash value or Hash. It transfers the digital signature and then both hash value and signature are sent to the receiver. The receiver uses the same hash function to generate the hash value and then compares it to that received with the message. If the hash values are the same, the message is transmitted without errors.

Hash table:

Hash table or hash map is a data structure that can provide constant time complexity O(1) lookup on an average and speed up information searching by a particular aspect of that information, regardless of the number of elements in the table. It is used to store key-value pairs. It can also be used to process hash value, generated by applying some function on the key, which determines where the record will be stored in the data structure. [3]Hash table is a collection of items stored to make it easy to find them later. A hash function enables to find an index into an array of buckets or slots from which the desired value can be found. It is an array of list where each list is known as bucket. It contains value based on the key. [1]Hash table is used to implement the map interface and extends Dictionary class. Hash table is synchronized and contains only unique elements.

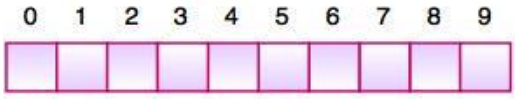

Fig. Hash Table

Assigning Elements:

The figure shown below is the hash table with the size

of n = 10. We term Slot for every position in the hash table. In the same hash table, there are n slots in the table (Slot 0, slot 1, slot 2 and so on). This hash table has 0 elements, this is why every slot is empty.

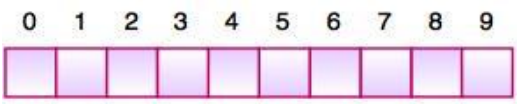

Fig. Hash Table

The hash function takes any item in the collection and returns an integer in the range of slot names between 0 and n-1.
We assume we have integer items {86, 30, 48, 61, 84, 43}. A method for the determination of hash key is the division method of hashing and the formula for same is:

Hash_Key(hk) = Key_Value(kv) % Number of Slots in the Table(n)

In the division method or reminder method we take an item and divide it by the table size(n) and return the remainder as its hash value.

Data Item	Value % No. of Slots	Hash Value
86	86 % 10 = 6	6
30	30 % 10 = 0	0
48	48 % 10 = 8	8
61	61 % 10 = 1	1
84	84 % 10 = 4	4
43	43 % 10 = 3	3

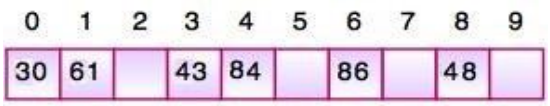

Fig. Hash Table

When we calculate the hash values, we can assign each item into the hash table at the designated position as shown in the above figure. In the hash table, 6 of the 10 slots are occupied, so load factor is calculated by following formula:

$$\lambda = \text{No. of items / table size}$$

Load factor for above figure is: $\lambda = 6/10$.

Problems in Hashing:
The primary problem with hashing is collisions, which occur when multiple keys map to the same hash value, causing issues with data retrieval and potentially impacting performance, especially when dealing with large datasets; other concerns include designing a good hash function, handling dynamic resizing of the hash table, and potential security risks depending on the application.

Key points about hashing problems:

- Collision handling:

The most significant challenge with hashing is how to effectively resolve collisions when two different keys map to the same hash value in the table.

- Poor hash function design:

Using a hash function that doesn't distribute keys evenly across the hash table can lead to a high probability of collisions, significantly impacting performance.

- Dynamic resizing:

As the number of elements in a hash table grows, the table may need to be resized to maintain good performance, which can be computationally expensive.

- Key ordering:

Hashing does not inherently preserve the order of keys, which can be a problem in certain applications.

- Security concerns:

In cryptographic applications, a poorly designed hash function can be vulnerable to attacks like collisions, allowing malicious actors to manipulate data.

Recovery from problems in hashing:

- Choose a good hash function:

Select a hash function that is designed to distribute keys evenly across the hash table.

- Collision resolution techniques:
  - Chaining: Store colliding elements in a linked list at the corresponding hash table index.
  - Open addressing: Probe for an empty slot in the table when a collision occurs.
- Load factor management:

Monitor the load factor (ratio of elements to table size) and resize the table as needed to avoid excessive collisions.

Use of statistical techniques for optimizing hashing:

The use of statistical techniques for optimizing hashing involves employing data analysis and

probabilistic methods to improve the performance, efficiency, and reliability of hashing algorithms. Hashing is critical in computer science for applications like data storage, retrieval, and cryptography. Below are some ways statistical techniques are used to optimize hashing:

1. Analyzing Hash Distribution
- Objective: Ensure uniform distribution of hash values to minimize collisions.
- Statistical Techniques Used:
  - Chi-Square Tests: Evaluate the uniformity of hash value distributions.
  - Entropy Analysis: Measure the randomness in hash outputs.
  - Kolmogorov-Smirnov Test: Compare the hash value distribution to an ideal uniform distribution.

2. Collision Minimization
- Objective: Reduce the probability of multiple keys mapping to the same hash value.
- Statistical Techniques Used:
  - Probability Analysis: Use the birthday paradox to estimate and reduce collision probabilities.
  - Monte Carlo Simulations: Simulate hashing operations to assess collision rates under different inputs.

3. Hash Function Selection
- Objective: Choose or design hash functions that perform well for specific datasets.
- Statistical Techniques Used:
  - Empirical Testing: Measure hash performance across large datasets.
  - Regression Analysis: Predict performance based on dataset characteristics.
  - Bayesian Optimization: Automate the tuning of hash function parameters for specific workloads.

4. Dynamic Hash Table Sizing
- Objective: Adjust hash table size dynamically to maintain efficiency as data grows.
- Statistical Techniques Used:
  - Load Factor Analysis: Use statistical thresholds to trigger resizing (e.g., when the table exceeds a specific load factor).
  - Markov Models: Predict future usage patterns for optimal table resizing.

5. Optimization for Locality-Sensitive Hashing (LSH)
- Objective: Optimize LSH for applications like nearest-neighbor searches and clustering.
- Statistical Techniques Used:
  - Clustering Algorithms (e.g., K-Means): Group similar items to fine-tune hash function parameters.
  - Dimensionality Reduction: Use techniques like PCA or t-SNE to preprocess data for better LSH performance.

6. Performance Evaluation
- Objective: Compare hash function performance using quantitative metrics.
- Statistical Techniques Used:
  - Hypothesis Testing: Determine if one hash function performs significantly better than others.
  - Confidence Intervals: Quantify the reliability of performance metrics.
  - Bootstrapping: Assess hash performance robustness over resampled datasets.

7. Adaptive Hashing Techniques
- Objective: Adjust hashing dynamically based on observed patterns or feedback.
- Statistical Techniques Used:
  - Reinforcement Learning: Adapt the hashing strategy in response to access patterns.
  - Anomaly Detection: Identify non-random patterns or skewed distributions in real-time.

8. Application-Specific Optimization
- Objective: Tailor hashing for domain-specific use cases, such as cryptography or databases.
- Statistical Techniques Used:
  - Cryptographic Strength Analysis: Use randomness tests to ensure security in cryptographic hash functions.
  - Data Profiling: Analyze dataset characteristics to design optimal hashing schemes for databases.

Statistical techniques offer powerful tools for analyzing, evaluating, and optimizing hashing algorithms. By leveraging these methods, we can design more efficient, collision-resistant, and application-specific hash functions tailored to diverse computational needs.

REFERENCES

[1] https://www.tutorialride.com
[2] https://medium.com
[3] https://ratanshreshtha.github.io/GrokkingCS/data-structures
[4] www.biitonline.co.in (Author Pankaj Kumar Gupta, Head, BCA Department, DPBS College, Anupshahr)
[5] Computer Science with C++ By SumitaAroraby SumitaArora.
[6] Let Us C by Yashavant Kanetkar 4. Introduction to Algorithms by Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein.
[7] Sartajsahni, "Data structures, algorithms and applications in C++", University press.
[8] Seymour Lipschutz. "Theory and problems of data structures", Tata Mcgraw hill international editions".
[9] The C Programming Language by Brian W. Kernighan / Dennis Ritchie
[10] www.en.wikipedia.org/wiki/Array.
[11] https://www.codecademy.com
[12] www.geeksforgeeks.org
[13] www.w3schools.com
[14] Introduction to Algorithms, 3rd Edition (The MIT Press).
[15] Data Structures (Revised First Edition) | Schaum's Outline Series by Seymour Lipschutz
[16] Algorithms and Data Structures Foundations and Probabilistic Methods for Design and Analysis By Helmut Knebl
[17] Algorithms in a Nutshell By George T. Heineman, Gary Pollice, Stanley Selkow .
[18] Data Structures and Algorithm Analysis in C++, Third Edition By Clifford A. Shaffer.
[19] Arrays: A Theoretical Approach of Memory Allocation, Pankaj Kumar Gupta & Dr. P. K. Tyagi, International Journal of Essential Sciences, Vol-14 No. 1 & 2 2020.
[20] Conceptual Discussion on Operations of Array: Traveral, Insertion & Deletion by Pankaj Kumar Gupta & Dr. P. K. Tyagi, International Journal of Research in all subjects in Multi Languages, Vol-10 Issue 3, March: 2022.
[21] https://www.programiz.com/dsa/stack
[22] https://www.geeksforgeeks.org/stack-data-structure/