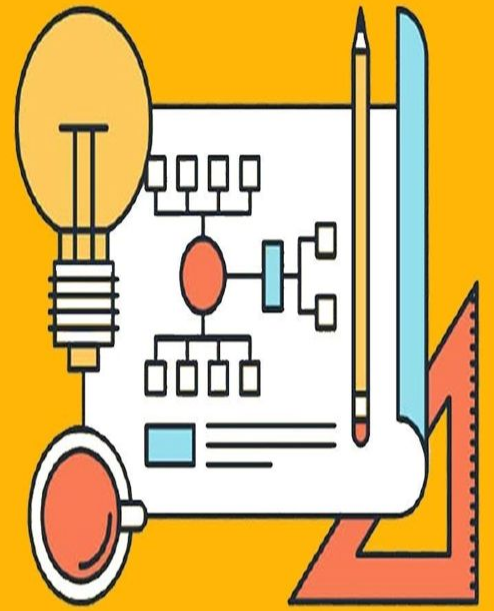
Hashing Techniques: The Key to Efficient Data Storage and Retrieval

By: Ranjan, Suraj & Sahil



Data Structure and Algorithm

Introduction:

- Let assume that we have a set |s| = n element to store
- Let’s take an array A of size m >= n
- Now we need to map x ∈ s to array A then we require a function also known as hash function h(x)
- The occupancy of a hash table is ratio α = n /m where alpha is the load factor.

Assumptions taken

- **Integer universe assumption :** all keys in hash table are integer within certain range.
- **Random probability assumption:** each element are coming as black box that comes with an infinite random probe sequence.
- To generate good performance of hash table we require $\alpha << 1$

Hash Functions under Integer Universe Assumption

1) Hashing by Division
   - *h(x)=x mod m*
   - Has problem when elements are of the form mx

2) Hashing by Multiplication
   - *h(x) = Integer(mxA) mod m*
   - Motivated by the theorem that *imA mod m, for i* $\in$ *Integer* will partition array into 3 distinct length only
   - Even these 3 distinct length can go big.

Hashing Functions Continued…

3) Universal Hashing

- Whatever the hash function we choose there will be some worst case.
- Take set of function H and depending on situation use one of them.
- $h(x) = ((ax + b) \bmod p) \bmod m$, p is prime, a & b are chosen randomly < p

Hash Functions under Random Probe Assumption

1) Linear Probing
   - $h(x) = (x + i) \bmod m$

2) Quadratic Probing
   - $h(x) = (x + i^2) \bmod m$

3) Double Hashing
   - $h(x) = (x + i*f(x)) \bmod m$

Hashing Applications: Securing and Optimizing Data

1. **Compiler Operation:** To differentiate between the keywords of a programming language(if, else, for, return etc.) and other identifiers

2. **Message Digest:**

   This is an application of cryptographic Hash Functions. Related to cloud storage.

3. **Linking File name and path together:**

   Hashing can be used to quickly compare the content of files, allowing for efficient detection of duplicates or changes.

Rabin-Karp Algorithm: Efficient String Matching

Hash-Based Comparison

1

The Rabin-Karp algorithm computes a hash value for the search pattern and compares it to the hash values of substrings in the text, rather than performing character-by-character comparisons.

Sliding Window

2

As the algorithm scans the text, it updates the hash value of the current window by removing the contribution of the leftmost character and adding the contribution of the rightmost character.

Applications

3

The Rabin-Karp algorithm is widely used in applications such as plagiarism detection, file comparison, and data deduplication.

Hashing for Secure Password Storage

One-Way Hashing

Passwords are never stored in plain text. Instead, a one-way hash function is used to transform the password into a unique fixed-length value.

Salt and Pepper

Adding a random salt value to each password before hashing, and optionally a global "pepper" value, further enhances the security of the hashed passwords.

Slow Hash Functions

Using computationally intensive hash functions, such as bcrypt or Argon2, makes it extremely difficult for attackers to brute-force password hashes.

References:

1. HandBook of data structures
   https://repository.gctu.edu.gh/files/original/a3d9ae2260132c98e676de18c30d9ff1.pdf

2. https://www.geeksforgeeks.org/applications-of-hashing/