Hashing Basics

Introduction: The Art of Efficient Data Storage and Retrieval

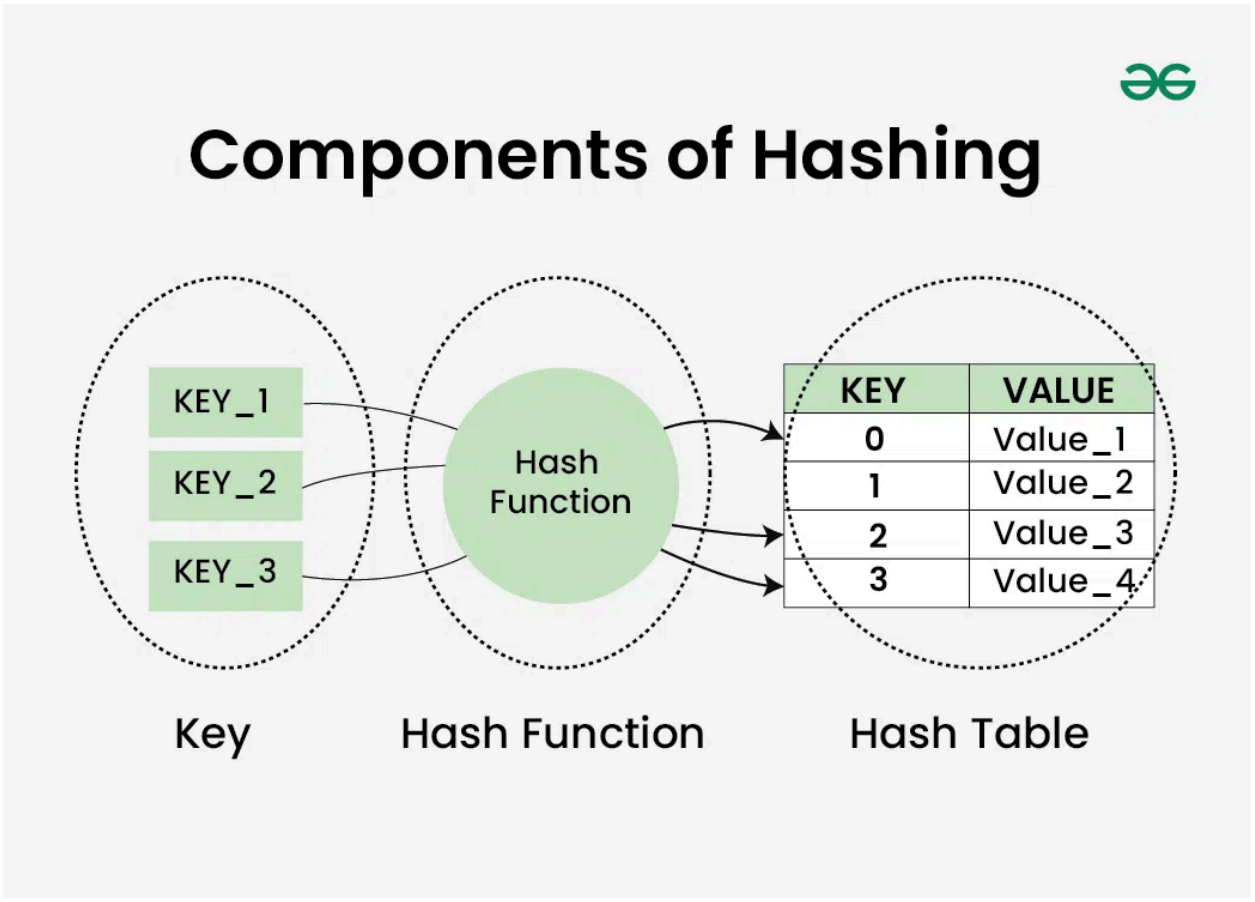
In the vast world of data structures, where efficiency and speed reign supreme, hashing emerges as a game-changer. It is a technique used to map data to a fixed-size table using a mathematical function known as a hash function. Hashing provides rapid access to data, making it a crucial component in databases, caching, cryptography, and various real-world applications.

What is Hashing?

Hashing is a method of converting an input (keys) into a fixed-length numerical value called a hash code. This hash code is then used as an index to store and retrieve data efficiently within a hash table.

Components of Hashing

1. **Hash Function**: A function that takes an input (key) and produces a hash value, determining the index in the hash table.
2. **Hash Table**: A data structure that stores data at indices derived from hash functions.
3. **Collision Handling**: A mechanism to resolve conflicts when two keys hash to the same index.



Properties of a Good Hash Function

A well-designed hash function should:

- Distribute keys uniformly across the hash table.
- Minimize collisions.
- Be efficient in computation.

Collision and Collision Handling Techniques

Collisions occur when two different keys produce the same hash value. Several strategies exist to handle collisions:

1. Chaining (Separate Chaining): Each index in the hash table contains a linked list where colliding keys are stored.

2. Open Addressing: If a collision occurs, the algorithm searches for the next available slot in the table.
    - Linear Probing: Finds the next available slot sequentially.
    - Quadratic Probing: Searches for an empty slot using a quadratic function.
    - Double Hashing: Uses a secondary hash function to find the next available slot.

Applications of Hashing

Hashing is widely used in various fields, including:

- Databases: Hashing allows quick lookup of records using unique identifiers.
- Cryptography: Secure hashing algorithms (SHA, MD5) ensure data integrity.
- Caching: Web browsers and databases use hashing for faster retrieval.
- Symbol Tables in Compilers: Hashing helps in quick access to variable names and functions.

Advantages of Hashing

- Fast Lookup: On average, hashing provides O(1) time complexity for search, insert, and delete operations.
- Efficient Memory Usage: Well-designed hash functions distribute data uniformly, reducing space wastage.
- Flexibility: Hashing supports dynamic data insertion without significant performance degradation.

Disadvantages of Hashing

- Collisions: Can impact performance if not handled properly.
- Extra Space Requirement: In some cases, unused slots lead to memory overhead.
- Complexity in Choosing a Hash Function: A poorly designed hash function can lead to inefficiencies.

Conclusion

Hashing is a fundamental concept in data structures that enhances the efficiency of data storage and retrieval. With proper implementation and effective collision-handling mechanisms, hashing ensures optimal performance for various applications. Mastering hashing techniques is essential for any programmer or computer scientist aiming to build high-performance systems.

CodeVerse

Questions: (All the 5 are mandatory *)

1) Given the following input (4322, 1334, 1471, 9679, 1989, 6171, 6173, 4199) and the hash function x mod 10, which of the following statements are true? (GATE CS 2004)

i. 9679, 1989, 4199 hash to the same value

ii. 1471, 6171 hash to the same value

iii. All elements hash to the same value

iv. Each element hashes to a different value

(A) i only

(B) ii only

(C) i and ii only

(D) iii or iv

2) In linear hashing, if blocking factor bfr, loading factor i and file buckets N are known, the number of records will be

A) r=i+bfr+N

B) r=i-bfr-N

C) r=i+bfr-N

D) r=i*bfr*N

3) A hash table of length 10 uses open addressing with hash function h(k)=k mod 10, and linear probing. After inserting 6 values into an empty hash table, the table is as shown below.

0	
1	
2	42
3	23
4	34
5	52
6	46
7	33
8	
9	

Which one of the following choices gives a possible order in which the key values could have been inserted in the table?

(A) 46, 42, 34, 52, 23, 33

(B) 34, 42, 23, 52, 33, 46

(C) 46, 34, 42, 23, 52, 33

(D) 42, 46, 33, 23, 34, 52

4) The keys 12, 18, 13, 2, 3, 23, 5 and 15 are inserted into an initially empty hash table of length 10 using open addressing with hash function h(k) = k mod 10 and linear probing. What is the resultant hash table?

0	
1	
2	2
3	23
4	
5	15
6	
7	
8	18
9	

(A)

0	
1	
2	12
3	13
4	
5	5
6	
7	
8	18
9	

(B)

0	
1	
2	12
3	13
4	2
5	3
6	23
7	5
8	18
9	15

(C)

0	
1	
2	12, 2
3	13, 3, 23
4	
5	5, 15
6	
7	
8	18
9	

(D)

5) An algorithm has to store several keys generated by an adversary in a hash table. The adversary is malicious who tries to maximize the number of collisions. Let k be the number of keys, m be the number of slots in the hash table, and $k>m$.

Which one of the following is the best hashing strategy to counteract the adversary?

A) Division method, i.e., use the hash function h(k)=kmfdm
B) Multiplication method, i.e., use the hash function $h(k)=\lfloor m(kA-\lfloor kA\rfloor)\rfloor$, where A is a carefully chosen constant.
C) If k is a prime number, use the Division method. Otherwise, use the Multiplication method.
D) Universal Hashing method.

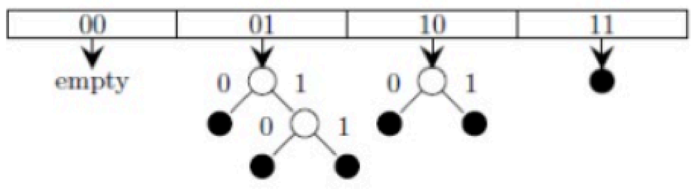
Extra Questions: (Non Mandatory)

6) Consider a dynamic hashing approach for 4-bit integer keys:

- There is a main hash table of size 4.
- The 2 least significant bits of a key are used to index into the main hash table.
- Initially, the main hash table entries are empty.

- Thereafter, when more keys are hashed into it, to resolve collisions, the set of all keys corresponding to a main hash table. entry is organized as a binary tree that grows on demand.
- First, the 3rd least significant bit is used to divide the keys into left and right subtrees.
- To resolve more collisions, each node of the binary tree is further sub-divided into left and right subtrees based on the 4th least significant bit.
- A split is done only if it is needed, i.e., only when there is a collision.

Consider the following state of the hash table.



Which of the following sequences of key insertions can cause the above state of the hash table (assume the keys are in decimal notation)?

A) 5,9,4,13,10,7

B) 9,5,10,6,7,1

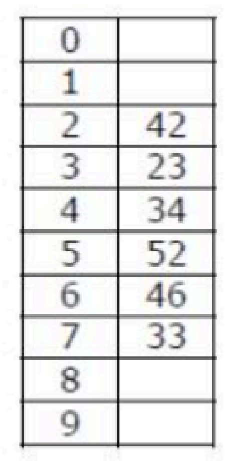
C) 10,9,6,7,5,13

D) 9,5,13,6,10,14

CodeVerse

Google Developer Groups
On Campus • RV University

7) How many different insertion sequences of the key values using the same hash function and linear probing will result in the hash table given in



0	
1	
2	42
3	23
4	34
5	52
6	46
7	33
8	
9	

(A) 10

(B) 20

(C) 30

(D) 40