## Table of Contents

## 1. Introduction

A hash map, also known as a hash table, is a data structure that provides fast and efficient lookup, insertion, and deletion operations. It is based on the concept of hashing, which allows for constant-time average-case performance for these operations.

A hash map uses an array as its underlying data structure and employs a hashing function to map keys to indices in the array. The hashing function takes the key as input and generates a hash code, which is an integer that represents the index.

Hash sets are similar to hash maps, but they store only unique keys (without associated values). They are often used to efficiently store a collection of distinct elements and support operations such as insertion, deletion, and membership testing (checking if an element exists in the set).

Overall, hash maps and hash sets are widely used data structures due to their efficient lookup performance and ability to handle large amounts of data. They find applications in various domains, including databases, caching systems, language interpreters, and more.

## 2. Algorithm

Here's a high-level overview of the operations and characteristics of a hash map:

1. Insertion: To insert a key-value pair into a hash map, the hashing function is applied to the key to determine the corresponding index in the array. If there is no collision (i.e., another key already occupies that index), the key-value pair is stored at that index. If a collision occurs, a technique like chaining or open addressing is used to handle it.
2. Lookup: To retrieve the value associated with a given key, the hashing function is applied to the key to compute the index. The hash map then navigates to that index and returns the corresponding value. This operation is typically performed in constant time, making hash maps efficient for lookups.

3. Deletion: Deleting a key-value pair from a hash map involves locating the index based on the key using the hashing function. The pair is then removed from that index, potentially resolving any collisions that may have occurred.
4. Collision handling: Hash maps employ techniques to handle collisions when two or more keys generate the same hash code. Some common collision resolution methods include chaining (using linked lists or other data structures to store multiple values at the same index) and open addressing (finding the next available index through probing or rehashing).
5. Efficiency: In an ideal scenario with a good hash function and minimal collisions, hash map operations have an average-case time complexity of O(1). However, in the worst case, when many collisions occur, the time complexity may degrade to O(n) since the hash map would effectively behave like a linear array.

## 3. Example Usage

Here are some example implementations of using hash maps and hash sets in Python:

Example usage of a Hash Map (dictionary in Python):

```
# Phone Book
phone_book = {
    "Alice": "123-456-7890",
    "Bob": "987-654-3210",
    "Charlie": "555-123-4567"
}

# Lookup phone number by name
name = "Alice"
if name in phone_book:
    phone_number = phone_book[name]
    print(f"{name}'s phone number is {phone_number}")
else:
    print(f"{name} is not found in the phone book.")

# Output: Alice's phone number is 123-456-7890
```

Example usage of a Hash Set (set in Python):

```
# Deduplication
numbers = [1, 2, 3, 4, 2, 3, 1, 5, 6, 4]
unique_numbers = set(numbers)
```

```
print(unique_numbers)  # Output: {1, 2, 3, 4, 5, 6}

# Membership Testing
liked_users = {1001, 1002, 1005, 1009}
user_id = 1005
if user_id in liked_users:
    print(f"User {user_id} has liked the post.")
else:
    print(f"User {user_id} has not liked the post.")

# Output: User 1005 has liked the post.
```

In the first example, a hash map (dictionary) is used to store phone numbers with corresponding names. It allows for efficient lookup of phone numbers by name.

In the second example, a hash set (set) is used to eliminate duplicates from a list of numbers and perform membership testing to check if a specific number exists in the set.

These examples showcase the simplicity and convenience of using hash maps and hash sets in Python for various tasks.

## 4. Advantages

Hash maps and hash sets offer several advantages that make them valuable data structures in many scenarios:

1. Fast Lookup: Hash maps and hash sets provide fast lookup operations. On average, the time complexity for lookup operations is O(1), which means they can quickly retrieve values or check for the presence of an element.
2. Efficient Insertion and Deletion: Similarly, hash maps and hash sets offer efficient insertion and deletion operations. On average, these operations also have a time complexity of O(1). This makes them suitable for scenarios that require frequent additions or removals of elements.
3. Handling Large Data: Hash maps and hash sets can handle large amounts of data efficiently. The lookup, insertion, and deletion operations have constant or near-constant time complexity, regardless of the size of the data set. This scalability makes them suitable for applications that deal with massive data sets.
4. Deduplication: Hash sets are particularly useful for eliminating duplicates from a collection of elements. By adding elements to a hash set, duplicates are automatically eliminated, as hash

sets only store unique elements. This simplifies deduplication tasks and improves data integrity.

5. Flexible Key-Value Structure: Hash maps provide a flexible key-value structure, allowing you to associate values with keys. This makes them suitable for a wide range of applications, such as storing mappings, performing word frequency counts, implementing caches, and more.
6. Membership Testing: Hash sets provide efficient membership testing. You can quickly check if an element exists in the set or not. This is useful for tasks like checking if a user has liked a post, determining if a word is present in a dictionary, or verifying if an item is part of a set of eligible choices.
7. Implementation Availability: Most programming languages provide built-in hash map and hash set implementations (e.g., dictionaries and sets in Python). These implementations are usually well-optimized and thoroughly tested, saving you from implementing these data structures from scratch.

Overall, the advantages of hash maps and hash sets stem from their efficient lookup, insertion, and deletion operations, scalability, deduplication capabilities, and availability in popular programming languages.

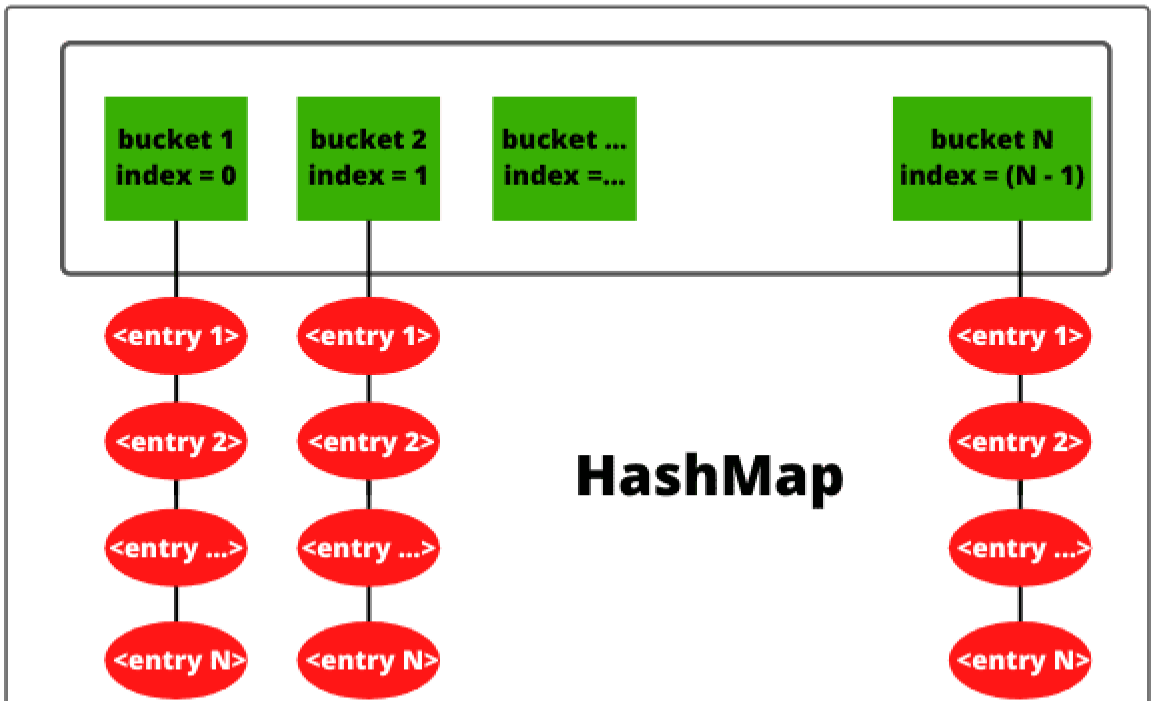

Fig: A Typical Hashing Data Structure