Hash Map [Link]

A HashMap (or hash table) is a data structure that provides a way to store key-value pairs, enabling efficient data retrieval. Here is a brief overview of its key concepts and operations:

Key Concepts

1. Key-Value Pairs :
   - Each entry in a HashMap consists of a key and a value. Keys are unique identifiers used to retrieve the associated value.

2. Hash Function :
   - A hash function converts a key into an index in an array where the value is stored. A good hash function distributes keys uniformly across the array to minimize collisions.

3. Buckets :
   - The array where values are stored is divided into slots called buckets. Each bucket can contain multiple key-value pairs in case of collisions.

4. Collisions :
   - Collisions occur when multiple keys hash to the same index. Common strategies to handle collisions include chaining (storing multiple entries in a linked list at each index) and open addressing (finding another slot within the array).

Common Operations

1. Insertion :
   - To insert a key-value pair, the hash function calculates the index for the key, and the value is stored at that index. If a collision occurs, the entry is added to the appropriate bucket.

2. Lookup :
   - To retrieve a value, the hash function calculates the index for the key, and the value is retrieved from the corresponding bucket.

3. Deletion :
   - To remove a key-value pair, the hash function calculates the index for the key, and the entry is removed from the corresponding bucket.

4. Updating :
   - To update the value for a key, the hash function calculates the index for the key, and the value is updated in the corresponding bucket.

Example

Here's a simplified example in pseudocode:

```
HashMap {
    Array of Buckets
    Hash Function
}

Insert(key, value):
    index = HashFunction(key)
    if Buckets[index] is empty:
        Buckets[index] = new LinkedList()
    Buckets[index].add((key, value))

Get(key):
    index = HashFunction(key)
    if Buckets[index] is not empty:
        for each (k, v) in Buckets[index]:
            if k == key:
                return v
    return null

Delete(key):
    index = HashFunction(key)
    if Buckets[index] is not empty:
        for each (k, v) in Buckets[index]:
            if k == key:
                Buckets[index].remove((k, v))
```

Advantages

1. Fast Access :
   - Average-case time complexity for insertion, lookup, and deletion is \(O(1)\).

2. Efficient Use of Space :
   - When designed well, HashMaps make efficient use of memory.

Disadvantages

1. Collisions :
   - Poor hash functions can result in many collisions, degrading performance to \(O(n)\) in the worst case.

2. Memory Overhead :
   - HashMaps can require more memory than other data structures due to the need for a hash table and handling collisions.

Applications

- Databases : Used for indexing to quickly locate data.
- Caching : Storing and retrieving frequently accessed data.
- Sets : Implementing sets where each element is unique.

HashMaps are widely used in computer science due to their efficiency and versatility in managing and accessing data.