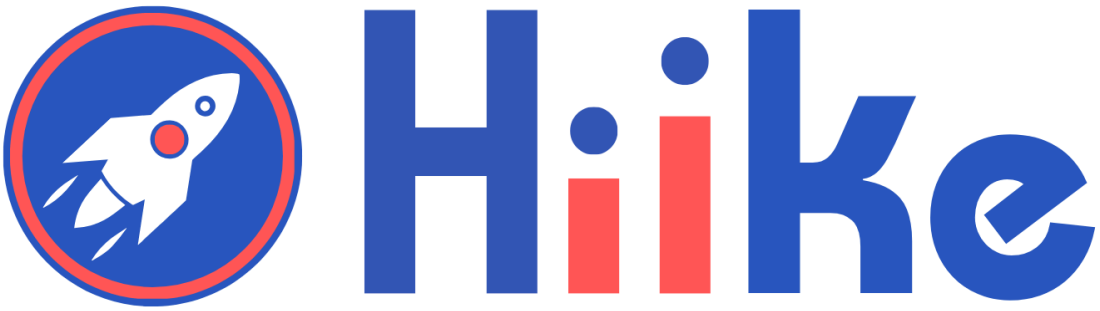
Hashing

Detailed Notes



Introduction to Hashing

Hashing is a technique used to uniquely identify a specific object from a group of similar objects. It is used in various applications such as database indexing, password storage, and caches. The main idea behind hashing is to map data of arbitrary size to fixed-size values using a hash function.

Hash Functions

A hash function is used to convert input data into a fixed-size integer, which serves as the index of the data in the hash table. An effective hash function has the following properties:

- Deterministic: The same input will always produce the same output.
- Efficiently Computable: It should be quick to compute the hash value.
- Uniform Distribution: The hash values should be uniformly distributed to avoid clustering.
- Minimize Collisions: Different inputs should ideally produce different hash values.

Hash Tables

A hash table is a data structure that implements an associative array abstract data type, a structure that can map keys to values. It uses a hash function to compute an index into an array of buckets or slots, from which the desired value can be found.

Basic Operations in a Hash Table:

- Insertion: Add a new key-value pair to the table.
- Deletion: Remove a key-value pair from the table.
- Search: Find the value associated with a given key.

Handling Collisions

Collisions occur when two different keys hash to the same index. There are several methods to handle collisions:

- Chaining: Store multiple elements at the same index using a linked list.
- Open Addressing: Find another open slot within the table using techniques such as linear probing, quadratic probing, or double hashing.

Example Problems with Pseudocode

Example 1: Subarray with 0 Sum

Problem: Check if there is a subarray with sum 0 in an array.

Pseudocode

```
function subArrayExists(arr, n):
    create an empty hash map 'm'
    set sum to 0
    insert 0 in hash map with count 1

    for i from 0 to n-1:
        sum = sum + arr[i]

        if 'sum' exists in hash map:
            return True

        increment the count of 'sum' in hash map

    return False
```

Explanation: This algorithm uses a hash map to keep track of the cumulative sum. If a cumulative sum repeats, it means there is a subarray with a sum of 0.

Time Complexity: O(n) - Each element is processed once.

Space Complexity: O(n) - In the worst case, all elements are stored in the hash map.

Example 2: Check if Two Strings are Anagrams

Problem: Check if two strings are anagrams.

Pseudocode:

```
function isAnagram(string1, string2):
    if length of string1 is not equal to length of string2:
        return False

    create a frequency array 'freq' of size 26 and initialize to 0

    for each character in string1:
        increment the count of character in 'freq'

    for each character in string2:
        decrement the count of character in 'freq'

    for each value in 'freq':
        if value is not 0:
            return False

    return True
```

Explanation: This algorithm uses a frequency array to count character occurrences. If the counts match for both strings, they are anagrams.

Time Complexity: O(n) - Each character is processed once.

Space Complexity: O(1) - Fixed size frequency array is used.

Conclusion

Hashing is a powerful technique used in various applications for efficient data retrieval. Understanding hash functions, hash tables, and collision resolution methods is crucial for solving competitive programming problems effectively. The provided implementations and examples demonstrate how hashing can be applied in real-world scenarios to solve complex problems efficiently.