Assignment 6: Advanced Hash Tables

**Duration: 30 minutes**

Assignment Description:

Delve into advanced concepts related to hash tables, including hash table resizing, load factors, and hash table optimizations.

**Questions**:

Hash Table Resizing: Explain the concept of hash table resizing. When and why is it necessary? Provide a code example in Java to demonstrate how hash table resizing works.

**Solution** 1:

java

// Hash Table Resizing: Hash table resizing is the process of increasing the size of a hash table when it reaches a certain threshold, typically defined by a load factor. This is necessary when the load factor exceeds a certain limit to maintain an efficient data structure.

// Hash table resizing is necessary to prevent too many collisions and keep the time complexity of operations (e.g., insert, search) close to O(1).

// Example of Hash Table Resizing in Java:

import java.util.HashMap;

public class HashTableResizingExample {

public static void main(String[] args) {

// Create a hash table with an initial capacity

HashMap<String, Integer> hashMap = new HashMap<>(5);

// Insert key-value pairs

for (int i = 0; i < 10; i++) {

hashMap.put("key" + i, i);

}

// As load factor exceeds the threshold, the hash table is resized

System.out.println("Current Capacity: " + hashMap.size());

}

}

Load Factors: What is a load factor in a hash table? How does it relate to resizing? Provide guidelines on choosing an appropriate load factor for a hash table. Create a Java code example that showcases load factor management.

**Solution** 2:

java

// Load Factors: A load factor in a hash table represents the ratio of the number of stored elements to the total number of buckets (table size). It indicates how full the hash table is. When the load factor exceeds a certain threshold, resizing is triggered to maintain efficiency.

// Guidelines for Choosing an Appropriate Load Factor:

// - A lower load factor means more empty buckets and less space utilization but lower collision risk.

// - A higher load factor means more filled buckets, efficient space usage, but a higher risk of collisions.

// - Common load factors are 0.75 or 0.5 for a balance between efficiency and space utilization.

// Example of Load Factor Management in Java:

import java.util.HashMap;

public class LoadFactorExample {

public static void main(String[] args) {

// Create a hash table with a specified load factor

HashMap<String, Integer> hashMap = new HashMap<>(10, 0.75f);

// Insert key-value pairs

for (int i = 0; i < 8; i++) {

hashMap.put("key" + i, i);

}

// As load factor exceeds the threshold, the hash table is resized

System.out.println("Current Capacity: " + hashMap.size());

}

}