Assignment 4: Hash Tables in Real-time Scenarios

**Duration: 30 minutes**

**Assignment** **Description**:

Explore the practical applications of hash tables in real-time scenarios. Investigate the significance of hash functions, collision resolution, and compare hash tables to arrays.

**Questions**:

Real-time Scenario: In a real-time scenario, such as a web-based chat application, describe how hash tables can be used. Explain their role, the advantages they offer, and how they enhance the user experience.

**Solution** 1:

java

In a web-based chat application, hash tables can be used to manage user profiles and active chat sessions.

Each user's unique session ID can serve as a key, and the data can include user details and the current chat session.

This allows quick access to user-specific information, enhancing the user experience.

HashMap<String, User> userProfiles = new HashMap<>();

HashMap<String, ChatSession> chatSessions = new HashMap<>();

Add a user profile

userProfiles.put("user123", new User("User123", "user123@example.com"));

Start a chat session

chatSessions.put("session456", new ChatSession("session456", "user123", "user789"));

Choosing Hash Functions: You're designing a database system with a hash table. Explain the criteria for selecting an appropriate hash function. What factors should you consider in the selection process?

**Solution** 2:

java

Criteria for Selecting a Hash Function:

1. Data Distribution: The hash function should distribute data evenly across the table to minimize collisions.

2. Table Size: Consider the size of the hash table; the hash function should be designed to fit it.

3. Data Type Features: Different data types may require specific hash function features.

4. Testing and Refinement: Test the hash function with representative data to ensure even distribution.

class DatabaseSystem {

int tableSize;

HashFunction hashFunction;

DatabaseSystem(int tableSize, HashFunction hashFunction) {

this.tableSize = tableSize;

this.hashFunction = hashFunction;

}

}

Collision Resolution: Describe a scenario where hash collisions can occur and explain how to resolve these collisions while maintaining data integrity.

**Solution** 3:

java

Scenario: In a document version control system, two different versions of a document may have the same hash.

To resolve collisions, use chaining - a linked list at each table index.

When a collision occurs, add the new key-value pair to the linked list at the corresponding index.

class DocumentVersionControlSystem {

LinkedList<DocumentVersion>[] hashTable;

DocumentVersionControlSystem(int tableSize) {

hashTable = new LinkedList[tableSize];

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

hashTable[i] = new LinkedList<>();

}

}

}

Hash Tables in Java: Provide a code example of using a hash table in Java.

**Solution** 4:

java

import java.util.HashMap;

public class HashTableExample {

public static void main(String[] args) {

Create a hash table (HashMap) in Java

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

Add key-value pairs

hashMap.put("age", 30);

hashMap.put("height", 180);

Access a value

int age = hashMap.get("age");

System.out.println("Age: " + age);

}

}