

**ASSIGNAMENT NO 03**

**(AUST)**

Name: SHAKEEL AHMAD

BSSE: 3rED

Section: C

Roll No: 12393

Instructor: Sir Jamal Abdul Ahad

Subject: DSA

Date : 22/11/2023

**Q1: When designing a user authentication system, explain how a hash table can store user credentials securely. Discuss the use of hash functions in password hashing and the importance of collision resistance.**

***In a user authentication system, storing user credentials securely is crucial to protect sensitive information like passwords. Hash tables play a vital role in achieving this security, especially when it comes to password storage.***

**Here's how a hash table can be used to store user credentials securely:**

* **Hash Functions and Password Hashing:**
* Instead of storing plain-text passwords, systems store the result of applying a hash function to the user's password. A hash function takes an input (in this case, the password) and produces a fixed-size string of characters, which is the hash value.
* This hashed password is what is stored in the hash table, not the actual password.
* **Collision Resistance:**
* A good hash function should have the property of collision resistance. Collision resistance means that it is computationally infeasible for two different inputs to produce the same hash output.
* In the context of password hashing, this property is crucial because if two different passwords produce the same hash (a collision), it could lead to security vulnerabilities. An attacker who discovers a collision might gain unauthorized access to an account without knowing the actual password.
* **Salting:**
* To enhance security further, a unique random value called a "salt" is often added to each password before hashing. This means that even if two users have the same password, their hashed passwords will be different due to the unique salt.
* Salting prevents attackers from using precomputed tables (rainbow tables) to quickly look up hash values for commonly used passwords.
* **Key Derivation Functions (KDFs):**
* Hash functions designed for password hashing should be slow and computationally expensive to resist brute-force attacks. Key Derivation Functions (KDFs) are often employed to achieve this purpose.
* KDFs, such as bcrypt, scrypt, or Argon2, are designed specifically for secure password hashing. They incorporate elements like iteration count and memory hardness to increase the computational cost of hashing, making it more challenging for attackers to crack passwords.

By using hash tables with properly designed hash functions and additional security measures like salting and KDFs, user authentication systems can store credentials securely, protecting user passwords even in the event of a data breach.

**Q2: In a task scheduling application, describe how a hash table can be used to store and quickly retrieve scheduled tasks based on their unique identifiers or names. Discuss the advantages of using a hash table for task management.**

***In a task scheduling application, a hash table can be a valuable data structure for efficiently storing and retrieving scheduled tasks based on their unique identifiers or names. Here's how a hash table can be utilized for task management:***

**Unique Identifiers as Keys:**

* Each scheduled task can be assigned a unique identifier or name. This unique identifier serves as the key in the hash table.
* When a task is scheduled, its details, such as description, due date, and other relevant information, are stored in the hash table using its unique identifier as the key.

**Efficient Retrieval:**

* Hash tables provide constant-time average-case complexity for retrieval operations. This means that, on average, the time required to retrieve a scheduled task is constant and doesn't depend on the size of the data set.
* Given the unique identifier, the hash table can quickly locate the corresponding entry, making task retrieval fast and efficient.

**Avoiding Collisions:**

* To ensure effective use of hash tables, it's essential to handle potential collisions, where two tasks might have the same hash value (due to hash function limitations).
* Techniques such as separate chaining or open addressing can be employed to manage collisions, ensuring that each unique identifier maps to its respective task.

**Flexibility and Scalability:**

* Hash tables are flexible and can adapt to varying numbers of tasks. As the number of scheduled tasks increases, a well-designed hash table can efficiently handle the growing dataset without a significant degradation in performance.

**Quick Updates:**

* Hash tables allow for quick updates and modifications to scheduled tasks. Whether you need to reschedule a task, update its details, or remove it, the constant-time complexity of hash table operations ensures that these tasks can be performed swiftly.

**Space Efficiency:**

* Hash tables can be space-efficient, especially when dealing with sparse data. Only slots corresponding to scheduled tasks need to be occupied in the hash table, leading to efficient memory usage.

**Query Performance:**

* When searching for a task based on its unique identifier, hash tables offer optimal query performance. This is particularly advantageous in scenarios where tasks need to be quickly retrieved and displayed to users.

In summary, using a hash table for task management in a scheduling application provides advantages such as efficient retrieval, constant-time complexity, flexibility, and quick updates. It offers a well-balanced trade-off between speed and memory usage, making it a suitable choice for scenarios where fast access to scheduled tasks is a key requirement.

**Q3: Suppose you have a sorted list of student exam scores. Explain how Binary Search can be applied to identify the position of a particular score in the list. Discuss any assumptions or requirements for using Binary Search in this scenario.**

***Binary Search is a divide-and-conquer algorithm that efficiently finds the position of a target value within a sorted list. Here's how Binary Search can be applied to identify the position of a particular score in a sorted list of student exam scores:***

**Initial Setup:**

* Ensure that the list of student exam scores is sorted in ascending order. Binary Search works on sorted data.

**Define Target Score:**

* Determine the student exam score you want to find in the sorted list. This score is the "target."

**Define Search Range:**

* Initially, the entire sorted list is considered as the search range.
* Set two pointers, low and high, to the first and last indices of the list, respectively.

**Binary Search Iteration:**

* In each iteration, calculate the middle index (mid) of the current search range.
* Compare the score at the middle index with the target score.
* If the middle score is equal to the target score, you have found the position.
* If the middle score is less than the target score, update low to mid + 1 (narrow the search to the upper half).
* If the middle score is greater than the target score, update high to mid - 1 (narrow the search to the lower half).

**Repeat Iteration:**

* Repeat the iteration until the low index is greater than the high index or the target score is found.

**Result:**

* If the target score is found, the position in the list is the index where the target score was located.
* If the target score is not in the list, the search will terminate with low greater than high, indicating that the target score is not present in the list.

***Assumptions/Requirements for Binary Search:***

**Sorted Data:**

* The list of student exam scores must be sorted in ascending order. Binary Search doesn't work correctly on unsorted data.

**Random Access:**

* Binary Search assumes that random access to elements in the list is possible. This is because it involves accessing the middle element during each iteration.

**Equality Comparison:**

* Binary Search is designed for finding equality, meaning it's used to locate an exact match. If there are duplicate scores and you want to find the first or last occurrence, additional modifications may be needed.

Binary Search is particularly advantageous for large datasets because it has a time complexity of O(log n), making it more efficient than linear search for sorted lists. However, it is important to ensure that the initial assumption of sorted data is maintained to guarantee the algorithm's correctness and efficiency.

**Q4: In a scientific experiment, data points are collected and sorted based on a parameter. Explain how Binary Search could be applied to locate specific data points efficiently. Discuss the scalability of Binary Search for large datasets.**

***In a scientific experiment where data points are collected and sorted based on a parameter, Binary Search can be applied to efficiently locate specific data points. Here's how:***

**Sorted Data:**

* Ensure that the collected data points are sorted based on the parameter of interest. Binary Search requires the data to be in sorted order.

**Define Search Criteria:**

* Determine the specific data point or range of data points you are looking for based on the parameter.

**Binary Search Iteration:**

* Set up the initial search range using two pointers, low and high, corresponding to the first and last indices of the sorted data.
* In each iteration, calculate the middle index (mid) of the current search range.
* Compare the parameter value of the data point at the middle index with the target parameter value.
* If they are equal, you have found the data point.
* If the middle parameter value is less than the target, update low to mid + 1 (narrowing the search to the upper half).
* If the middle parameter value is greater than the target, update high to mid - 1 (narrowing the search to the lower half).

**Repeat Iteration:**

* Repeat the iteration until the low index is greater than the high index or the target parameter value is found.

**Result:**

* If the target parameter value is found, the index of that data point gives its position in the sorted list.

**Scalability for Large Datasets:**

* Binary Search is highly scalable for large datasets due to its time complexity of O(log n).
* As the dataset grows, Binary Search remains efficient because it divides the search space in half with each iteration, resulting in a logarithmic growth rate.
* In contrast to linear search (O(n)), Binary Search's efficiency becomes more pronounced as the dataset size increases.

**Advantages:**

* Efficient Retrieval: Binary Search is significantly faster than linear search, especially as the dataset becomes larger.

**Consistent Time Complexity:**

* The time complexity of Binary Search remains logarithmic regardless of the size of the dataset, ensuring predictable performance.

**Considerations:**

* Pre-sorting Requirement: Binary Search assumes that the data is pre-sorted. Sorting the data initially can be a one-time cost, but if the data frequently changes, maintaining the sorted order might become an additional consideration.

In scientific experiments with large datasets, Binary Search provides an efficient means of locating specific data points based on a parameter, making it a favorable choice for applications where quick and scalable retrieval is essential.

**………………..THE END……………….**