1797. Design Authentication Manager

Medium Design Hash Table

Problem Description

The problem describes an authentication system that issues authentication tokens, each with an expiration time set in seconds. The expiration time (`timeToLive`) is a fixed duration from the moment the token is either generated or renewed. The `currentTime` is a timestamp indicating the current time in seconds. The system should perform three main functions:

1. Generate a new token with a unique `tokenId` and set its expiry time.
2. Renew an existing unexpired token with the given `tokenId`. If the token is expired or does not exist, no action is taken.
3. Count the number of unexpired tokens at a given `currentTime`.

The catch is that the expiration is processed before any other action if they both occur at the same time. The task is to implement the `AuthenticationManager` class that can handle these operations.

Intuition

The solution is to implement the `AuthenticationManager` class with the methods that handle tokens and their expiration times.

- When a new token is generated, we just add the `tokenId` and its expiration time (current time + `timeToLive`) to a dictionary. In this dictionary, the keys are the `tokenIds`, and the values are their respective expiration times.
- To renew a token, we check if the token exists and is not expired by comparing its stored expiration time with the `currentTime`. If it's not expired, we update its expiration time. If the token is expired or doesn't exist, we ignore the renew request.
- To count the unexpired tokens, we iterate through the dictionary's values, which are the expiration times, and count how many of these times are greater than the `currentTime`.

Solution Approach

The solution utilizes a Python dictionary for storing tokens and their expiration times, which allows for quick lookups, inserts, and updates. Here's how the implementation works:

- **Initialization (`__init__` method):** The `AuthenticationManager` is initialized with the `timeToLive` value, which determines the lifespan of each token. We also define a dictionary `self.d` that will map each `tokenId` to its expiration time.
- **Generating tokens (`generate` method):** When a new token is generated, we calculate its expiration time (`currentTime + self.t`) and store it in the dictionary `self.d` with `tokenId` as the key. This process is instant due to the efficient nature of dictionary operations in Python.

```
def generate(self, tokenId: str, currentTime: int) -> None:
    self.d[tokenId] = currentTime + self.t
```

- **Renewing tokens (`renew` method):** To renew a token, we first check whether the token exists and is not expired by comparing the current time with the stored expiration time. If these conditions are met, we update the token's expiration time to the new current time plus `timeToLive`. If the token is not found or is expired, we do nothing.

```
def renew(self, tokenId: str, currentTime: int) -> None:
    if tokenId in self.d and self.d[tokenId] > currentTime:
        self.d[tokenId] = currentTime + self.t
```

- **Counting unexpired tokens (`countUnexpiredTokens` method):** This function iterates over all expiration times in the dictionary and counts the number of times greater than the current time, indicating the tokens still valid.

```
def countUnexpiredTokens(self, currentTime: int) -> int:
    return sum(exp > currentTime for exp in self.d.values())
```

The use of the `sum` function combined with a generator expression makes counting efficient. The expression `exp > currentTime` evaluates to `True` or `False`, which in Python correspond to `1` and `0`. Thus, `sum` effectively counts the number of unexpired tokens.

With this implementation, the `AuthenticationManager` accomplishes the requested operations with a time complexity of O(1) for generating or renewing a token and O(n) for counting unexpired tokens, where n is the number of tokens being managed.

Example Walkthrough

Let's walk through an example to understand how the `AuthenticationManager` is implemented and functions with the given solution approach.

- Assume the `timeToLive` value for tokens is set to 5 seconds.
- The `currentTime` when a token is generated is 1.

Now, let's run through the methods:

1. **Initialization:**

   ```
   manager = AuthenticationManager(5)
   ```

   We have created an instance of `AuthenticationManager` named `manager` with `timeToLive` set to 5 seconds. This means each token will expire 5 seconds after it has been created or renewed.

2. **Generating a token:**

   ```
   manager.generate("token123", 1)
   ```

   After this call, the internal dictionary `self.d` will contain:

   ```
   {"token123": 6}
   ```

   This indicates that `token123` will expire at time 6 (`currentTime` + `timeToLive`).

3. **Renewing a token:** Suppose the current time now is 3, and we wish to renew `token123`:

   ```
   manager.renew("token123", 3)
   ```

   The internal dictionary gets updated:

   ```
   {"token123": 8}
   ```

   This is because the current time is 3, and adding the `timeToLive` value of 5 seconds gives us the new expiration time at 8.

4. **Attempting to renew an expired token:** If we fast-forward the time to 7 and attempt to renew `token123`:

   ```
   manager.renew("token123", 7)
   ```

   Nothing changes since `token123` was set to expire at time 8, so it's still valid and can be renewed:

   ```
   {"token123": 12}
   ```

5. **Counting unexpired tokens:** Let's now count the unexpired tokens at the current time, which is 7:

   ```
   count = manager.countUnexpiredTokens(7)
   ```

   The count will be 1 because `token123` will now expire at 12, which is greater than the current time of 7.

This example demonstrates how the `AuthenticationManager` efficiently manages tokens, handles renewals, and counts unexpired tokens as required by the problem description. The implementation ensures rapid generation and renewal using dictionary operations and provides an O(n) approach to count unexpired tokens, where n is the total number of tokens.

Solution Implementation

Python

```
from collections import defaultdict

class AuthenticationManager:
    def __init__(self, timeToLive: int):
        # Initialize the time to live for the tokens
        self.time_to_live = timeToLive
        # Use a dictionary to keep track of the tokens and their expiration times
        self.token_expirations = defaultdict(int)

    def generate(self, token_id: str, current_time: int) -> None:
        # Create a new token with an expiration time based on current time + time_to_live
        self.token_expirations[token_id] = current_time + self.time_to_live

    def renew(self, token_id: str, current_time: int) -> None:
        # Renew a token's expiration time if it hasn't already expired
        if self.token_expirations[token_id] > current_time:
            self.token_expirations[token_id] = current_time + self.time_to_live

    def count_unexpired_tokens(self, current_time: int) -> int:
        # Count the number of tokens that have not yet expired
        return sum(expiration_time > current_time for expiration_time in self.token_expirations.values())

# Example of how the AuthenticationManager class can be used:
# auth_manager = AuthenticationManager(time_to_live)
# auth_manager.generate(token_id, current_time)
# auth_manager.renew(token_id, current_time)
# unexpired_tokens_count = auth_manager.count_unexpired_tokens(current_time)
```

Java

```
import java.util.HashMap;
import java.util.Map;

class AuthenticationManager {
    private int timeToLive; // duration for which the tokens are valid
    private Map<String, Integer> tokenExpiryMap; // holds token IDs and their corresponding expiration times

    // Constructor initializes the AuthenticationManager with a specific time to live for tokens
    public AuthenticationManager(int timeToLive) {
        this.timeToLive = timeToLive;
        this.tokenExpiryMap = new HashMap<>();
    }

    // Generates a new token with an expiration time based on the current time
    public void generate(String tokenId, int currentTime) {
        // Add the token and its expiration time to the map
        tokenExpiryMap.put(tokenId, currentTime + timeToLive);
    }

    // Renews an existing token if it hasn't expired yet
    public void renew(String tokenId, int currentTime) {
        // Retrieve the current expiration time for the token
        Integer expirationTime = tokenExpiryMap.getOrDefault(tokenId, 0);

        // If the token is still valid (hasn't expired), renew it by updating its expiration time
        if (expirationTime > currentTime) {
            generate(tokenId, currentTime);
        }
    }

    // Counts the number of unexpired tokens at a given current time
    public int countUnexpiredTokens(int currentTime) {
        // Initialize a counter to keep track of valid tokens
        int count = 0;

        // Iterate through all tokens and increment the count if they're not expired
        for (int expirationTime : tokenExpiryMap.values()) {
            if (expirationTime > currentTime) {
                count++;
            }
        }

        // Return the total number of unexpired tokens
        return count;
    }
}

// The AuthenticationManager class functionality can be utilized as shown below:
// AuthenticationManager obj = new AuthenticationManager(timeToLive);
// obj.generate(tokenId, currentTime);
// obj.renew(tokenId, currentTime);
// int param_3 = obj.countUnexpiredTokens(currentTime);
```

C++

```
#include <unordered_map>
#include <string>
using namespace std;

class AuthenticationManager {
public:
    // Constructor with initialization of timeToLive for tokens
    AuthenticationManager(int timeToLive) : timeToLive_(timeToLive) {
    }

    // Generate a new token with a given tokenId and current time
    void generate(string tokenId, int currentTime) {
        // Token is valid from the current time until 'currentTime + timeToLive_'
        tokens_[tokenId] = currentTime + timeToLive_;
    }

    // Renew a token if it's still valid at the given current time
    void renew(string tokenId, int currentTime) {
        // If the token exists and is not expired, renew its expiration time
        if (tokens_.find(tokenId) != tokens_.end() && tokens_[tokenId] > currentTime) {
            generate(tokenId, currentTime);
        }
    }

    // Return the count of tokens that are not yet expired at the given current time
    int countUnexpiredTokens(int currentTime) {
        int count = 0;
        for (const auto& tokenPair : tokens_) {
            // If the token's expiration time is greater than currentTime, increment the count
            if (tokenPair.second > currentTime) {
                ++count;
            }
        }
        return count;
    }

private:
    // Time to live for each token
    int timeToLive_;

    // Stores token IDs and their respective expiration times
    unordered_map<string, int> tokens_;
};

/**
 * Usage:
 * AuthenticationManager authManager = new AuthenticationManager(timeToLive);
 * authManager->generate(tokenId, currentTime);
 * authManager->renew(tokenId, currentTime);
 * int activeTokens = authManager->countUnexpiredTokens(currentTime);
 * delete authManager; // Remember to deallocate memory
 */
```

TypeScript

```
// Time-to-live for authentication tokens
let timeToLive: number;

// A map to store tokens IDs and their expiration times
let tokenMap: Map<string, number>;

// Initializes variables with the specified timeToLive
function initializeAuthenticationManager(ttl: number): void {
    timeToLive = ttl;
    tokenMap = new Map<string, number>();
}

// Generates a new token with a given tokenId that expires after timeToLive
function generate(tokenId: string, currentTime: number): void {
    tokenMap.set(tokenId, currentTime + timeToLive);
}

// Renews the token with the given tokenId if it hasn't expired
function renew(tokenId: string, currentTime: number): void {
    const expirationTime = tokenMap.get(tokenId);
    if (expirationTime && expirationTime > currentTime) {
        tokenMap.set(tokenId, currentTime + timeToLive);
    }
}

// Counts the number of unexpired tokens at the given currentTime
function countUnexpiredTokens(currentTime: number): number {
    let count = 0;
    tokenMap.forEach((expirationTime) => {
        if (expirationTime > currentTime) {
            count++;
        }
    });
    return count;
}

// Example usage:
// initializeAuthenticationManager(3600); // Initialize with 1 hour TTL
// generate("token123", 100); // Generate a new token with ID "token123"
// renew("token123", 500); // Attempt to renew "token123" at time 500
// let unexpiredCount = countUnexpiredTokens(1000); // Count unexpired tokens at time 1000
```

```
from collections import defaultdict

class AuthenticationManager:
    def __init__(self, time_to_live: int):
        # Initialize the time to live for the tokens
        self.time_to_live = time_to_live
        # Use a dictionary to keep track of the tokens and their expiration times
        self.token_expirations = defaultdict(int)

    def generate(self, token_id: str, current_time: int) -> None:
        # Create a new token with an expiration time based on current time + time_to_live
        self.token_expirations[token_id] = current_time + self.time_to_live

    def renew(self, token_id: str, current_time: int) -> None:
        # Renew a token's expiration time if it hasn't already expired
        if self.token_expirations[token_id] > current_time:
            self.token_expirations[token_id] = current_time + self.time_to_live

    def count_unexpired_tokens(self, current_time: int) -> int:
        # Count the number of tokens that have not yet expired
        return sum(expiration_time > current_time for expiration_time in self.token_expirations.values())

# Example of how the AuthenticationManager class can be used:
# auth_manager = AuthenticationManager(time_to_live)
# auth_manager.generate(token_id, current_time)
# auth_manager.renew(token_id, current_time)
# unexpired_tokens_count = auth_manager.count_unexpired_tokens(current_time)
```

Time and Space Complexity

Time Complexity:

- `__init__`: O(1) because we are just setting up variables.
- `generate`: O(1) as it involves a single operation of assigning a new expiration time for a token.
- `renew`: O(1) for accessing the dictionary and updating a token's expiration time conditionally. The conditional check and dictionary update happen in constant time.
- `countUnexpiredTokens`: O(n) where n is the number of tokens in the dictionary. This is due to the iteration over all values to sum the number of unexpired tokens.

Space Complexity:

- The space complexity of the entire class is O(n), where n is the number of different tokens stored. The dictionary `self.d` grows as new tokens are generated or renewed.