<https://leetcode.ca/all/359.html>

Design a logger system that receives a stream of messages along with their timestamps. Each **unique** message should only be printed **at most every 10 seconds** (i.e. a message printed at timestamp t will prevent other identical messages from being printed until timestamp t + 10).

All messages will come in chronological order. Several messages may arrive at the same timestamp.

Implement the Logger class:

Logger() Initializes the logger object.

bool shouldPrintMessage(int timestamp, string message) Returns true if the message should be printed in the given timestamp, otherwise returns false.

**Example:**

Input

["Logger", "shouldPrintMessage", "shouldPrintMessage", "shouldPrintMessage", "shouldPrintMessage", "shouldPrintMessage", "shouldPrintMessage"]

[[], [1, "foo"], [2, "bar"], [3, "foo"], [8, "bar"], [10, "foo"], [11, "foo"]]

Output

[null, true, true, false, false, false, true]

Explanation

Logger logger = new Logger();

logger.shouldPrintMessage(1, "foo"); // return true, next allowed timestamp for "foo" is 1 + 10 = 11

logger.shouldPrintMessage(2, "bar"); // return true, next allowed timestamp for "bar" is 2 + 10 = 12

logger.shouldPrintMessage(3, "foo"); // 3 < 11, return false

logger.shouldPrintMessage(8, "bar"); // 8 < 12, return false

logger.shouldPrintMessage(10, "foo"); // 10 < 11, return false

logger.shouldPrintMessage(11, "foo"); // 11 >= 11, return true, next allowed timestamp for "foo" is 11 + 10 = 21

**Attempt 1: 2025-02-23**

**Solution 1: Hash Table (10 min)**

class Logger {

// Map to store the last timestamp when each message was printed

private Map<String, Integer> messageTimestamps;

/\*\* Initialize your data structure here. \*/

public Logger() {

messageTimestamps = new HashMap<>();

}

/\*\*

\* Returns true if the message should be printed in the given timestamp, otherwise returns false.

\* If this method returns false, the message will not be printed.

\*/

public boolean shouldPrintMessage(int timestamp, String message) {

// Check if the message has been printed before

if (messageTimestamps.containsKey(message)) {

// Get the last timestamp when the message was printed

int lastTimestamp = messageTimestamps.get(message);

// Check if the message was printed within the last 10 seconds

if (timestamp - lastTimestamp < 10) {

return false;

}

}

// Update the timestamp for the message

messageTimestamps.put(message, timestamp);

return true;

}

}

Time Complexity: O(n)

Space Complexity: O(n)

**Refer to Deepseek**

To solve the Logger Rate Limiter problem, we need to design a system that ensures each message is printed only if it hasn't been printed in the last 10 seconds. We can achieve this by using a HashMap to store the last timestamp when each message was printed.

Here's the Java implementation:

import java.util.HashMap;

import java.util.Map;

class Logger {

// Map to store the last timestamp when each message was printed

private Map<String, Integer> messageTimestamps;

/\*\* Initialize your data structure here. \*/

public Logger() {

messageTimestamps = new HashMap<>();

}

/\*\*

\* Returns true if the message should be printed in the given timestamp, otherwise returns false.

\* If this method returns false, the message will not be printed.

\*/

public boolean shouldPrintMessage(int timestamp, String message) {

// Check if the message has been printed before

if (messageTimestamps.containsKey(message)) {

// Get the last timestamp when the message was printed

int lastTimestamp = messageTimestamps.get(message);

// Check if the message was printed within the last 10 seconds

if (timestamp - lastTimestamp < 10) {

return false;

}

}

// Update the timestamp for the message

messageTimestamps.put(message, timestamp);

return true;

}

}

// Example usage:

public class Main {

public static void main(String[] args) {

Logger logger = new Logger();

System.out.println(logger.shouldPrintMessage(1, "foo")); // returns true

System.out.println(logger.shouldPrintMessage(2, "bar")); // returns true

System.out.println(logger.shouldPrintMessage(3, "foo")); // returns false

System.out.println(logger.shouldPrintMessage(8, "bar")); // returns false

System.out.println(logger.shouldPrintMessage(10, "foo")); // returns false

System.out.println(logger.shouldPrintMessage(11, "foo")); // returns true

}

}

**Explanation:**

\*\*Data Structure\*\*: We use a

HashMap called messageTimestamps to store the last timestamp when each message was printed. The key is the message, and the value is the timestamp.

\*\*shouldPrintMessage Method\*\*:

We first check if the message already exists in the HashMap.

If it does, we compare the current timestamp with the last timestamp when the message was printed.

If the difference is less than 10 seconds, we return false (indicating the message should not be printed again).

If the message hasn't been printed before or if it was printed more than 10 seconds ago, we update the timestamp in the HashMap and return true.

\*\*Example Usage\*\*: The

Main class demonstrates how to use the Logger class with the provided example.

**Complexity Analysis:**

\*\*Time Complexity\*\*: O(1) for both shouldPrintMessage and put operations in the HashMap.

\*\*Space Complexity\*\*: O(N), where N is the number of unique messages stored in the HashMap.

This solution efficiently handles the rate-limiting requirement for logging messages.