359. Logger Rate Limiter

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 1:

```
Input
{"\n"}["Logger", "shouldPrintMessage", "shouldPrintMessage",
"shouldPrintMessage", "shouldPrintMessage", "shouldPrintMessage",
"shouldPrintMessage"]{"\n"}[[], [1, "foo"], [2, "bar"], [3, "foo"], [8,
"bar"], [10, "foo"], [11, "foo"]]{"\n"}
Output
{"\n"}[null, true, true, false, false, false, true]{"\n"}
{"\n"}
Explanation
{"\n"}Logger logger = new Logger();{"\n"}logger.shouldPrintMessage(1,
"foo");{"  "}// return true, next allowed timestamp for "foo" is 1 + 10 = 11
{"\n"}logger.shouldPrintMessage(2, "bar");{"  "}// return true, next allowed
timestamp for "bar" is 2 + 10 = 12{"\n"}logger.shouldPrintMessage(3, "foo");
{"  "}// 3 < 11, return false{"\n"}logger.shouldPrintMessage(8, "bar");
{"  "}// 8 < 12, return false{"\n"}logger.shouldPrintMessage(10, "foo");
// 10 < 11, return false{"\n"}logger.shouldPrintMessage(11, "foo"); // 11
>= 11, return true, next allowed timestamp for "foo" is 11 + 10 = 21
{"\n"}
```

Constraints:

- `0 <= timestamp <= 10^9`
- Every `timestamp` will be passed in non-decreasing order (chronological order).
- `1 <= message.length <= 30`
- At most{" "} `10^4` {" "} calls will be made to `shouldPrintMessage`.

Solution

Naive Solution

We can store a list of all previous `(timestamp, message)` pairs. In each call, we loop through the list to get the most recent time `message` was logged and check if it was within 10 seconds of `timestamp`. Let n be the number of times `shouldPrintMessage` is called. Checking through the list takes $\mathcal{O}(n)$, so we take $\mathcal{O}(n^2)$ in total. Our list has size n, taking $\mathcal{O}(n)$ space. This is fast enough, but we can do better.

Faster Solution

We use a hashmap that maps messages to their most recent timestamps. Retrieving/assigning `hashmap[message]` takes $\mathcal{O}(1)$, so we take $\mathcal{O}(n)$ in total. We also take $\mathcal{O}(n)$ space (in the worst case, every message is different, so all of them need to be inserted into the hashmap).

C++ Solution

```cpp
class Logger {
public:
    unordered_map<string, int> lastTime;
    bool shouldPrintMessage(int timestamp, string message) {
        if (lastTime.count(message) and timestamp - lastTime[message] < 10)
            return false;
        lastTime[message] = timestamp;
        return true;
    }
};

/**
 * Your Logger object will be instantiated and called as such:
 * Logger* obj = new Logger();
 * bool param_1 = obj->shouldPrintMessage(timestamp,message);
 */
```

Java Solution

```java
class Logger {
    HashMap<String, Integer> lastTime;
    public Logger() {
        lastTime = new HashMap<>();
    }
    public boolean shouldPrintMessage(int timestamp, String message) {
        if (timestamp - lastTime.getOrDefault(message, -100) < 10)
            return false;
        lastTime.put(message, timestamp);
        return true;
    }
}

/**
 * Your Logger object will be instantiated and called as such:
 * Logger obj = new Logger();
 * boolean param_1 = obj.shouldPrintMessage(timestamp,message);
 */
```

Python Solution

```python
class Logger:
    def __init__(self):
        self.lastTime = dict()
    def shouldPrintMessage(self, timestamp: int, message: str) -> bool:
        if message in self.lastTime and timestamp - self.lastTime[message] < 10:
            return False
        self.lastTime[message] = timestamp
        return True

# Your Logger object will be instantiated and called as such:
# obj = Logger()
# param_1 = obj.shouldPrintMessage(timestamp,message)
```