Problem 359: Logger Rate Limiter

Problem Information

Difficulty: Easy
Acceptance Rate: 0.00%
Paid Only: No

Problem Description

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

["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

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

.

Code Snippets

C++:

```
class Logger {
public:
Logger() {

}

bool shouldPrintMessage(int timestamp, string message) {

}
};

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

Java:

```
class Logger {

public Logger() {

}

public boolean shouldPrintMessage(int timestamp, String message) {

}
}

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

Python3:

```
class Logger:

def __init__(self):


def shouldPrintMessage(self, timestamp: int, message: str) -> bool:



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

Python:

```
class Logger(object):

def __init__(self):


def shouldPrintMessage(self, timestamp, message):
"""
:type timestamp: int
:type message: str
:rtype: bool
"""



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

JavaScript:

```

var Logger = function() {

};

/**
* @param {number} timestamp
* @param {string} message
```

```
* @return {boolean}
*/
Logger.prototype.shouldPrintMessage = function(timestamp, message) {

};

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

TypeScript:

```
class Logger {
constructor() {

}

shouldPrintMessage(timestamp: number, message: string): boolean {

}
}

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

C#:

```
public class Logger {

public Logger() {

}

public bool ShouldPrintMessage(int timestamp, string message) {

}
}
```

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

C:

```
typedef struct {

} Logger;


Logger* loggerCreate() {

}

bool loggerShouldPrintMessage(Logger* obj, int timestamp, char* message) {

}

void loggerFree(Logger* obj) {

}

/**
* Your Logger struct will be instantiated and called as such:
* Logger* obj = loggerCreate();
* bool param_1 = loggerShouldPrintMessage(obj, timestamp, message);

* loggerFree(obj);
*/
```

Go:

```
type Logger struct {

}
```

```
func Constructor() Logger {

}


func (this *Logger) ShouldPrintMessage(timestamp int, message string) bool {

}


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

Kotlin:

```
class Logger() {

fun shouldPrintMessage(timestamp: Int, message: String): Boolean {

}

}

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

Swift:

```
class Logger {

init() {

}
```

```
func shouldPrintMessage(_ timestamp: Int, _ message: String) -> Bool {

}
}

/**
* Your Logger object will be instantiated and called as such:
* let obj = Logger()
* let ret_1: Bool = obj.shouldPrintMessage(timestamp, message)
*/
```

Rust:

```
struct Logger {

}


/**
* `&self` means the method takes an immutable reference.
* If you need a mutable reference, change it to `&mut self` instead.
*/
impl Logger {

fn new() -> Self {

}

fn should_print_message(&self, timestamp: i32, message: String) -> bool {

}
}

/**
* Your Logger object will be instantiated and called as such:
* let obj = Logger::new();
* let ret_1: bool = obj.should_print_message(timestamp, message);
*/
```

Ruby:

```
class Logger
def initialize()

end


=begin
:type timestamp: Integer
:type message: String
:rtype: Boolean
=end
def should_print_message(timestamp, message)

end



end

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

PHP:

```
class Logger {
/**
*/
function __construct() {

}

/**
* @param Integer $timestamp
* @param String $message
* @return Boolean
*/
function shouldPrintMessage($timestamp, $message) {

}
}

/**
* Your Logger object will be instantiated and called as such:
```

```
* $obj = Logger();
* $ret_1 = $obj->shouldPrintMessage($timestamp, $message);
*/
```

Dart:

```
class Logger {

Logger() {

}

bool shouldPrintMessage(int timestamp, String message) {

}
}

/**
* Your Logger object will be instantiated and called as such:
* Logger obj = Logger();
* bool param1 = obj.shouldPrintMessage(timestamp,message);
*/
```

Scala:

```
class Logger() {

def shouldPrintMessage(timestamp: Int, message: String): Boolean = {

}

}

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

Elixir:

```
defmodule Logger do
@spec init_() :: any
def init_() do

end

@spec should_print_message(timestamp :: integer, message :: String.t) ::
boolean
def should_print_message(timestamp, message) do

end
end

# Your functions will be called as such:
# Logger.init_()
# param_1 = Logger.should_print_message(timestamp, message)

# Logger.init_ will be called before every test case, in which you can do
some necessary initializations.
```

Erlang:

```
-spec logger_init_() -> any().
logger_init_() ->
.

-spec logger_should_print_message(Timestamp :: integer(), Message ::
unicode:unicode_binary()) -> boolean().
logger_should_print_message(Timestamp, Message) ->
.


%% Your functions will be called as such:
%% logger_init_(),
%% Param_1 = logger_should_print_message(Timestamp, Message),

%% logger_init_ will be called before every test case, in which you can do
some necessary initializations.
```

Racket:

```
(define logger%
(class object%
```

```
(super-new)

(init-field)

; should-print-message : exact-integer? string? -> boolean?
(define/public (should-print-message timestamp message)
)))

;; Your logger% object will be instantiated and called as such:
;; (define obj (new logger%))
;; (define param_1 (send obj should-print-message timestamp message))
```

Solutions

C++ Solution:

```
/*
* Problem: Logger Rate Limiter
* Difficulty: Easy
* Tags: string, dp, hash
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Logger {
public:
Logger() {

}

bool shouldPrintMessage(int timestamp, string message) {

}
};

/**
* Your Logger object will be instantiated and called as such:
* Logger* obj = new Logger();
```

```
* bool param_1 = obj->shouldPrintMessage(timestamp,message);
*/
```

Java Solution:

```
/**
* Problem: Logger Rate Limiter
* Difficulty: Easy
* Tags: string, dp, hash
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Logger {

public Logger() {

}

public boolean shouldPrintMessage(int timestamp, String message) {

}
}

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

Python3 Solution:

```
"""
Problem: Logger Rate Limiter
Difficulty: Easy
Tags: string, dp, hash

Approach: String manipulation with hash map or two pointers
Time Complexity: O(n) or O(n log n)
```

```
Space Complexity: O(n) or O(n * m) for DP table
"""

class Logger:

def __init__(self):


def shouldPrintMessage(self, timestamp: int, message: str) -> bool:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Logger(object):

def __init__(self):


def shouldPrintMessage(self, timestamp, message):
"""
:type timestamp: int
:type message: str
:rtype: bool
"""




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

JavaScript Solution:

```
/**
* Problem: Logger Rate Limiter
* Difficulty: Easy
* Tags: string, dp, hash
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
```

```
* Space Complexity: O(n) or O(n * m) for DP table
*/


var Logger = function() {

};

/**
* @param {number} timestamp
* @param {string} message
* @return {boolean}
*/
Logger.prototype.shouldPrintMessage = function(timestamp, message) {

};

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

TypeScript Solution:

```
/**
* Problem: Logger Rate Limiter
* Difficulty: Easy
* Tags: string, dp, hash
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Logger {
constructor() {

}

shouldPrintMessage(timestamp: number, message: string): boolean {
```

```
}
}

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

C# Solution:

```
/*
* Problem: Logger Rate Limiter
* Difficulty: Easy
* Tags: string, dp, hash
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

public class Logger {

public Logger() {

}

public bool ShouldPrintMessage(int timestamp, string message) {

}
}

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

C Solution:

```c
/*
* Problem: Logger Rate Limiter
* Difficulty: Easy
* Tags: string, dp, hash
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/




typedef struct {

} Logger;


Logger* loggerCreate() {

}

bool loggerShouldPrintMessage(Logger* obj, int timestamp, char* message) {

}

void loggerFree(Logger* obj) {

}

/**
* Your Logger struct will be instantiated and called as such:
* Logger* obj = loggerCreate();
* bool param_1 = loggerShouldPrintMessage(obj, timestamp, message);

* loggerFree(obj);
*/
```

Go Solution:

```go
// Problem: Logger Rate Limiter
// Difficulty: Easy
```

```
// Tags: string, dp, hash
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

type Logger struct {

}


func Constructor() Logger {

}


func (this *Logger) ShouldPrintMessage(timestamp int, message string) bool {

}


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

Kotlin Solution:

```
class Logger() {

fun shouldPrintMessage(timestamp: Int, message: String): Boolean {

}

}

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

```
*/
```

Swift Solution:

```
class Logger {

init() {

}

func shouldPrintMessage(_ timestamp: Int, _ message: String) -> Bool {

}
}

/**
* Your Logger object will be instantiated and called as such:
* let obj = Logger()
* let ret_1: Bool = obj.shouldPrintMessage(timestamp, message)
*/
```

Rust Solution:

```
// Problem: Logger Rate Limiter
// Difficulty: Easy
// Tags: string, dp, hash
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

struct Logger {

}


/**
* `&self` means the method takes an immutable reference.
* If you need a mutable reference, change it to `&mut self` instead.
*/
```

```
impl Logger {

fn new() -> Self {

}

fn should_print_message(&self, timestamp: i32, message: String) -> bool {

}
}

/**
* Your Logger object will be instantiated and called as such:
* let obj = Logger::new();
* let ret_1: bool = obj.should_print_message(timestamp, message);
*/
```

Ruby Solution:

```
class Logger
def initialize()

end


=begin
:type timestamp: Integer
:type message: String
:rtype: Boolean
=end
def should_print_message(timestamp, message)

end



end

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

PHP Solution:

```
class Logger {
/**
*/
function __construct() {

}

/**
* @param Integer $timestamp
* @param String $message
* @return Boolean
*/
function shouldPrintMessage($timestamp, $message) {

}
}

/**
* Your Logger object will be instantiated and called as such:
* $obj = Logger();
* $ret_1 = $obj->shouldPrintMessage($timestamp, $message);
*/
```

Dart Solution:

```
class Logger {

Logger() {

}

bool shouldPrintMessage(int timestamp, String message) {

}
}

/**
* Your Logger object will be instantiated and called as such:
* Logger obj = Logger();
* bool param1 = obj.shouldPrintMessage(timestamp,message);
*/
```

Scala Solution:

```
class Logger() {

def shouldPrintMessage(timestamp: Int, message: String): Boolean = {

}

}

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

Elixir Solution:

```
defmodule Logger do
@spec init_() :: any
def init_() do

end

@spec should_print_message(timestamp :: integer, message :: String.t) ::
boolean
def should_print_message(timestamp, message) do

end
end

# Your functions will be called as such:
# Logger.init_()
# param_1 = Logger.should_print_message(timestamp, message)

# Logger.init_ will be called before every test case, in which you can do
some necessary initializations.
```

Erlang Solution:

```
-spec logger_init_() -> any().
logger_init_() ->
```

```
.

-spec logger_should_print_message(Timestamp :: integer(), Message ::
unicode:unicode_binary()) -> boolean().
logger_should_print_message(Timestamp, Message) ->
.


%% Your functions will be called as such:
%% logger_init_(),
%% Param_1 = logger_should_print_message(Timestamp, Message),

%% logger_init_ will be called before every test case, in which you can do
some necessary initializations.
```

Racket Solution:

```
(define logger%
(class object%
(super-new)

(init-field)

; should-print-message : exact-integer? string? -> boolean?
(define/public (should-print-message timestamp message)
)))

;; Your logger% object will be instantiated and called as such:
;; (define obj (new logger%))
;; (define param_1 (send obj should-print-message timestamp message))
```