Problem 362: Design Hit Counter

Problem Information

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

Problem Description

Design a hit counter which counts the number of hits received in the past

5

minutes (i.e., the past

300

seconds).

Your system should accept a

timestamp

parameter (

in seconds

granularity), and you may assume that calls are being made to the system in chronological order (i.e.,

timestamp

is monotonically increasing). Several hits may arrive roughly at the same time.

Implement the

HitCounter

class:

HitCounter()

Initializes the object of the hit counter system.

void hit(int timestamp)

Records a hit that happened at

timestamp

(

in seconds

). Several hits may happen at the same

timestamp

.

int getHits(int timestamp)

Returns the number of hits in the past 5 minutes from

timestamp

(i.e., the past

300

seconds).

Example 1:

Input

["HitCounter", "hit", "hit", "hit", "getHits", "hit", "getHits", "getHits"] [[], [1], [2], [3], [4], [300], [300], [301]]

Output

[null, null, null, null, 3, null, 4, 3]

Explanation

HitCounter hitCounter = new HitCounter(); hitCounter.hit(1); // hit at timestamp 1. hitCounter.hit(2); // hit at timestamp 2. hitCounter.hit(3); // hit at timestamp 3. hitCounter.getHits(4); // get hits at timestamp 4, return 3. hitCounter.hit(300); // hit at timestamp 300. hitCounter.getHits(300); // get hits at timestamp 300, return 4. hitCounter.getHits(301); // get hits at timestamp 301, return 3.

Constraints:

1 <= timestamp <= 2 * 10

9

All the calls are being made to the system in chronological order (i.e.,

timestamp

is monotonically increasing).

At most

300

calls will be made to

hit

and

getHits

.

Follow up:

What if the number of hits per second could be huge? Does your design scale?

Code Snippets

C++:

```
class HitCounter {
public:
HitCounter() {

}

void hit(int timestamp) {

}

int getHits(int timestamp) {

}
};

/**
* Your HitCounter object will be instantiated and called as such:
* HitCounter* obj = new HitCounter();
* obj->hit(timestamp);
* int param_2 = obj->getHits(timestamp);
*/
```

Java:

```
class HitCounter {

public HitCounter() {

}
public void hit(int timestamp) {

}

public int getHits(int timestamp) {

}
}

/**
* Your HitCounter object will be instantiated and called as such:
* HitCounter obj = new HitCounter();
* obj.hit(timestamp);
* int param_2 = obj.getHits(timestamp);
*/
```

Python3:

```
class HitCounter:

def __init__(self):


def hit(self, timestamp: int) -> None:


def getHits(self, timestamp: int) -> int:



# Your HitCounter object will be instantiated and called as such:
# obj = HitCounter()
# obj.hit(timestamp)
# param_2 = obj.getHits(timestamp)
```

Python:

```
class HitCounter(object):

def __init__(self):
```

```
def hit(self, timestamp):
"""
:type timestamp: int
:rtype: None
"""


def getHits(self, timestamp):
"""
:type timestamp: int
:rtype: int
"""



# Your HitCounter object will be instantiated and called as such:
# obj = HitCounter()
# obj.hit(timestamp)
# param_2 = obj.getHits(timestamp)
```

JavaScript:

```
var HitCounter = function() {

};

/**
* @param {number} timestamp
* @return {void}
*/
HitCounter.prototype.hit = function(timestamp) {

};

/**
* @param {number} timestamp
* @return {number}
*/
HitCounter.prototype.getHits = function(timestamp) {

};

/**
* Your HitCounter object will be instantiated and called as such:
* var obj = new HitCounter()
* obj.hit(timestamp)
* var param_2 = obj.getHits(timestamp)
*/
```

TypeScript:

```
class HitCounter {
constructor() {

}

hit(timestamp: number): void {

}

getHits(timestamp: number): number {

}
}

/**
* Your HitCounter object will be instantiated and called as such:
* var obj = new HitCounter()
* obj.hit(timestamp)
* var param_2 = obj.getHits(timestamp)
*/
```

C#:

```
public class HitCounter {

public HitCounter() {

}

public void Hit(int timestamp) {
}

public int GetHits(int timestamp) {

}
}

/**
* Your HitCounter object will be instantiated and called as such:
* HitCounter obj = new HitCounter();
* obj.Hit(timestamp);
* int param_2 = obj.GetHits(timestamp);
*/
```

C:

```


typedef struct {

} HitCounter;


HitCounter* hitCounterCreate() {

}

void hitCounterHit(HitCounter* obj, int timestamp) {

}

int hitCounterGetHits(HitCounter* obj, int timestamp) {

}

void hitCounterFree(HitCounter* obj) {

}

/**
* Your HitCounter struct will be instantiated and called as such:
```

```
* HitCounter* obj = hitCounterCreate();
* hitCounterHit(obj, timestamp);

* int param_2 = hitCounterGetHits(obj, timestamp);

* hitCounterFree(obj);
*/
```

Go:

```
type HitCounter struct {

}



func Constructor() HitCounter {

}



func (this *HitCounter) Hit(timestamp int) {

}



func (this *HitCounter) GetHits(timestamp int) int {

}



/**
* Your HitCounter object will be instantiated and called as such:
* obj := Constructor();
* obj.Hit(timestamp);
* param_2 := obj.GetHits(timestamp);
*/
```

Kotlin:

```
class HitCounter() {

fun hit(timestamp: Int) {
}

fun getHits(timestamp: Int): Int {

}

}

/**
* Your HitCounter object will be instantiated and called as such:
* var obj = HitCounter()
* obj.hit(timestamp)
* var param_2 = obj.getHits(timestamp)
*/
```

Swift:

```
class HitCounter {

init() {

}

func hit(_ timestamp: Int) {

}

func getHits(_ timestamp: Int) -> Int {

}
}

/**
* Your HitCounter object will be instantiated and called as such:
* let obj = HitCounter()
* obj.hit(timestamp)
* let ret_2: Int = obj.getHits(timestamp)
*/
```

Rust:

```
struct HitCounter {

}



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

fn new() -> Self {

}

fn hit(&self, timestamp: i32) {

}

fn get_hits(&self, timestamp: i32) -> i32 {

}
}

/**
* Your HitCounter object will be instantiated and called as such:
* let obj = HitCounter::new();
* obj.hit(timestamp);
* let ret_2: i32 = obj.get_hits(timestamp);
*/
```

Ruby:

```
class HitCounter
def initialize()

end



=begin
:type timestamp: Integer
```

```
:rtype: Void
=end
def hit(timestamp)

end



=begin
:type timestamp: Integer
:rtype: Integer
=end
def get_hits(timestamp)

end



end

# Your HitCounter object will be instantiated and called as such:
# obj = HitCounter.new()
# obj.hit(timestamp)
# param_2 = obj.get_hits(timestamp)
```

PHP:

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

}

/**
* @param Integer $timestamp
* @return NULL
*/
function hit($timestamp) {

}

/**
* @param Integer $timestamp
* @return Integer
*/
function getHits($timestamp) {

}
}

/**
* Your HitCounter object will be instantiated and called as such:
* $obj = HitCounter();
* $obj->hit($timestamp);
* $ret_2 = $obj->getHits($timestamp);
*/
```

Dart:

```
class HitCounter {

HitCounter() {

}

void hit(int timestamp) {

}

int getHits(int timestamp) {

}
}

/**
* Your HitCounter object will be instantiated and called as such:
* HitCounter obj = HitCounter();
* obj.hit(timestamp);
* int param2 = obj.getHits(timestamp);
*/
```

Scala:

```
class HitCounter() {
```

```

def hit(timestamp: Int): Unit = {

}

def getHits(timestamp: Int): Int = {

}

}

/**
* Your HitCounter object will be instantiated and called as such:
* val obj = new HitCounter()
* obj.hit(timestamp)
* val param_2 = obj.getHits(timestamp)
*/
```

Elixir:

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

end

@spec hit(timestamp :: integer) :: any
def hit(timestamp) do

end

@spec get_hits(timestamp :: integer) :: integer
def get_hits(timestamp) do

end
end

# Your functions will be called as such:
# HitCounter.init_()
# HitCounter.hit(timestamp)
# param_2 = HitCounter.get_hits(timestamp)

# HitCounter.init_ will be called before every test case, in which you can do
```

```
some necessary initializations.
```

Erlang:

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

-spec hit_counter_hit(Timestamp :: integer()) -> any().
hit_counter_hit(Timestamp) ->
.

-spec hit_counter_get_hits(Timestamp :: integer()) -> integer().
hit_counter_get_hits(Timestamp) ->
.



%% Your functions will be called as such:
%% hit_counter_init_(),
%% hit_counter_hit(Timestamp),
%% Param_2 = hit_counter_get_hits(Timestamp),

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

Racket:

```
(define hit-counter%
(class object%
(super-new)

(init-field)

; hit : exact-integer? -> void?
(define/public (hit timestamp)
)
; get-hits : exact-integer? -> exact-integer?
(define/public (get-hits timestamp)
)))

;; Your hit-counter% object will be instantiated and called as such:
;; (define obj (new hit-counter%))
;; (send obj hit timestamp)
;; (define param_2 (send obj get-hits timestamp))
```

Solutions

C++ Solution:

```
/*
* Problem: Design Hit Counter
* Difficulty: Medium
* Tags: array, search, queue
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class HitCounter {
public:
HitCounter() {

}

void hit(int timestamp) {

}

int getHits(int timestamp) {

}
};

/**
* Your HitCounter object will be instantiated and called as such:
* HitCounter* obj = new HitCounter();
* obj->hit(timestamp);
* int param_2 = obj->getHits(timestamp);
*/
```

Java Solution:

```
/**
* Problem: Design Hit Counter
* Difficulty: Medium
* Tags: array, search, queue
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class HitCounter {

public HitCounter() {

}

public void hit(int timestamp) {

}

public int getHits(int timestamp) {

}
}

/**
* Your HitCounter object will be instantiated and called as such:
* HitCounter obj = new HitCounter();
* obj.hit(timestamp);
* int param_2 = obj.getHits(timestamp);
*/
```

Python3 Solution:

```
"""
Problem: Design Hit Counter
Difficulty: Medium
Tags: array, search, queue

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(1) to O(n) depending on approach
```

```
"""

class HitCounter:

def __init__(self):



def hit(self, timestamp: int) -> None:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class HitCounter(object):

def __init__(self):



def hit(self, timestamp):
"""
:type timestamp: int
:rtype: None
"""



def getHits(self, timestamp):
"""
:type timestamp: int
:rtype: int
"""





# Your HitCounter object will be instantiated and called as such:
# obj = HitCounter()
# obj.hit(timestamp)
# param_2 = obj.getHits(timestamp)
```

JavaScript Solution:

```
/**
* Problem: Design Hit Counter
* Difficulty: Medium
* Tags: array, search, queue
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/


var HitCounter = function() {

};

/**
* @param {number} timestamp
* @return {void}
*/
HitCounter.prototype.hit = function(timestamp) {

};

/**
* @param {number} timestamp
* @return {number}
*/
HitCounter.prototype.getHits = function(timestamp) {

};

/**
* Your HitCounter object will be instantiated and called as such:
* var obj = new HitCounter()
* obj.hit(timestamp)
* var param_2 = obj.getHits(timestamp)
*/
```

TypeScript Solution:

```
/**
* Problem: Design Hit Counter
* Difficulty: Medium
* Tags: array, search, queue
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class HitCounter {
constructor() {

}

hit(timestamp: number): void {

}

getHits(timestamp: number): number {

}
}

/**
* Your HitCounter object will be instantiated and called as such:
* var obj = new HitCounter()
* obj.hit(timestamp)
* var param_2 = obj.getHits(timestamp)
*/
```

C# Solution:

```
/*
* Problem: Design Hit Counter
* Difficulty: Medium
* Tags: array, search, queue
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/
```

```
public class HitCounter {

public HitCounter() {

}

public void Hit(int timestamp) {

}

public int GetHits(int timestamp) {

}
}

/**
* Your HitCounter object will be instantiated and called as such:
* HitCounter obj = new HitCounter();
* obj.Hit(timestamp);
* int param_2 = obj.GetHits(timestamp);
*/
```

C Solution:

```
/*
* Problem: Design Hit Counter
* Difficulty: Medium
* Tags: array, search, queue
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/




typedef struct {

} HitCounter;
```

```
HitCounter* hitCounterCreate() {

}

void hitCounterHit(HitCounter* obj, int timestamp) {

}

int hitCounterGetHits(HitCounter* obj, int timestamp) {

}

void hitCounterFree(HitCounter* obj) {

}

/**
* Your HitCounter struct will be instantiated and called as such:
* HitCounter* obj = hitCounterCreate();
* hitCounterHit(obj, timestamp);

* int param_2 = hitCounterGetHits(obj, timestamp);

* hitCounterFree(obj);
*/
```

Go Solution:

```
// Problem: Design Hit Counter
// Difficulty: Medium
// Tags: array, search, queue
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

type HitCounter struct {

}
```

```
func Constructor() HitCounter {

}


func (this *HitCounter) Hit(timestamp int) {

}


func (this *HitCounter) GetHits(timestamp int) int {

}


/**
* Your HitCounter object will be instantiated and called as such:
* obj := Constructor();
* obj.Hit(timestamp);
* param_2 := obj.GetHits(timestamp);
*/
```

Kotlin Solution:

```
class HitCounter() {

fun hit(timestamp: Int) {

}

fun getHits(timestamp: Int): Int {

}

}

/**
* Your HitCounter object will be instantiated and called as such:
* var obj = HitCounter()
* obj.hit(timestamp)
* var param_2 = obj.getHits(timestamp)
*/
```

Swift Solution:

```
class HitCounter {

init() {

}

func hit(_ timestamp: Int) {

}

func getHits(_ timestamp: Int) -> Int {

}
}

/**
* Your HitCounter object will be instantiated and called as such:
* let obj = HitCounter()
* obj.hit(timestamp)
* let ret_2: Int = obj.getHits(timestamp)
*/
```

Rust Solution:

```
// Problem: Design Hit Counter
// Difficulty: Medium
// Tags: array, search, queue
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

struct HitCounter {


}
```

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

fn new() -> Self {

}

fn hit(&self, timestamp: i32) {

}

fn get_hits(&self, timestamp: i32) -> i32 {

}
}

/**
* Your HitCounter object will be instantiated and called as such:
* let obj = HitCounter::new();
* obj.hit(timestamp);
* let ret_2: i32 = obj.get_hits(timestamp);
*/
```

Ruby Solution:

```
class HitCounter
def initialize()

end



=begin
:type timestamp: Integer
:rtype: Void
=end
def hit(timestamp)
```

```
end


=begin
:type timestamp: Integer
:rtype: Integer
=end
def get_hits(timestamp)

end


end

# Your HitCounter object will be instantiated and called as such:
# obj = HitCounter.new()
# obj.hit(timestamp)
# param_2 = obj.get_hits(timestamp)
```

PHP Solution:

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

}

/**
* @param Integer $timestamp
* @return NULL
*/
function hit($timestamp) {

}

/**
* @param Integer $timestamp
* @return Integer
*/
function getHits($timestamp) {

}
}

/**
* Your HitCounter object will be instantiated and called as such:
* $obj = HitCounter();
* $obj->hit($timestamp);
* $ret_2 = $obj->getHits($timestamp);
*/
```

Dart Solution:

```
class HitCounter {

HitCounter() {

}

void hit(int timestamp) {

}

int getHits(int timestamp) {

}
}

/**
* Your HitCounter object will be instantiated and called as such:
* HitCounter obj = HitCounter();
* obj.hit(timestamp);
* int param2 = obj.getHits(timestamp);
*/
```

Scala Solution:

```
class HitCounter() {

def hit(timestamp: Int): Unit = {
}

def getHits(timestamp: Int): Int = {

}

}

/**
* Your HitCounter object will be instantiated and called as such:
* val obj = new HitCounter()
* obj.hit(timestamp)
* val param_2 = obj.getHits(timestamp)
*/
```

Elixir Solution:

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

end

@spec hit(timestamp :: integer) :: any
def hit(timestamp) do

end

@spec get_hits(timestamp :: integer) :: integer
def get_hits(timestamp) do

end
end

# Your functions will be called as such:
# HitCounter.init_()
# HitCounter.hit(timestamp)
# param_2 = HitCounter.get_hits(timestamp)

# HitCounter.init_ will be called before every test case, in which you can do
```

```
some necessary initializations.
```

Erlang Solution:

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

-spec hit_counter_hit(Timestamp :: integer()) -> any().
hit_counter_hit(Timestamp) ->
.

-spec hit_counter_get_hits(Timestamp :: integer()) -> integer().
hit_counter_get_hits(Timestamp) ->
.



%% Your functions will be called as such:
%% hit_counter_init_(),
%% hit_counter_hit(Timestamp),
%% Param_2 = hit_counter_get_hits(Timestamp),

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

Racket Solution:

```
(define hit-counter%
(class object%
(super-new)

(init-field)

; hit : exact-integer? -> void?
(define/public (hit timestamp)
)
; get-hits : exact-integer? -> exact-integer?
(define/public (get-hits timestamp)
)))

;; Your hit-counter% object will be instantiated and called as such:
;; (define obj (new hit-counter%))
;; (send obj hit timestamp)
;; (define param_2 (send obj get-hits timestamp))
```