Problem 635: Design Log Storage System

Problem Information

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

Problem Description

You are given several logs, where each log contains a unique ID and timestamp. Timestamp is a string that has the following format:

Year:Month:Day:Hour:Minute:Second

, for example,

2017:01:01:23:59:59

. All domains are zero-padded decimal numbers.

Implement the

LogSystem

class:

LogSystem()

Initializes the

LogSystem

object.

void put(int id, string timestamp)

Stores the given log

(id, timestamp)

in your storage system.

int[] retrieve(string start, string end, string granularity)

Returns the IDs of the logs whose timestamps are within the range from

start

to

end

inclusive.

start

and

end

all have the same format as

timestamp

, and

granularity

means how precise the range should be (i.e. to the exact

Day

,

Minute

, etc.). For example,

start = "2017:01:01:23:59:59"

,

end = "2017:01:02:23:59:59"

, and

granularity = "Day"

means that we need to find the logs within the inclusive range from

Jan. 1st 2017

to

Jan. 2nd 2017

, and the

Hour

,

Minute

, and

Second

for each log entry can be ignored.

Example 1:

Input

["LogSystem", "put", "put", "put", "retrieve", "retrieve"] [[], [1, "2017:01:01:23:59:59"], [2, "2017:01:01:22:59:59"], [3, "2016:01:01:00:00:00"], ["2016:01:01:01:01:01", "2017:01:01:23:00:00", "Year"], ["2016:01:01:01:01:01", "2017:01:01:23:00:00", "Hour"]]

Output

[null, null, null, null, [3, 2, 1], [2, 1]]

Explanation

LogSystem logSystem = new LogSystem(); logSystem.put(1, "2017:01:01:23:59:59"); logSystem.put(2, "2017:01:01:22:59:59"); logSystem.put(3, "2016:01:01:00:00:00");

// return [3,2,1], because you need to return all logs between 2016 and 2017. logSystem.retrieve("2016:01:01:01:01:01", "2017:01:01:23:00:00", "Year");

// return [2,1], because you need to return all logs between Jan. 1, 2016 01:XX:XX and Jan. 1, 2017 23:XX:XX. // Log 3 is not returned because Jan. 1, 2016 00:00:00 comes before the start of the range. logSystem.retrieve("2016:01:01:01:01:01", "2017:01:01:23:00:00", "Hour");

Constraints:

1 <= id <= 500

2000 <= Year <= 2017

1 <= Month <= 12

1 <= Day <= 31

0 <= Hour <= 23

0 <= Minute, Second <= 59

granularity

is one of the values

["Year", "Month", "Day", "Hour", "Minute", "Second"]

.

At most

500

calls will be made to

put

and

retrieve

.

Code Snippets

C++:

```
class LogSystem {
public:
LogSystem() {

}

void put(int id, string timestamp) {

}

vector<int> retrieve(string start, string end, string granularity) {

}
};

/**
* Your LogSystem object will be instantiated and called as such:
* LogSystem* obj = new LogSystem();
* obj->put(id,timestamp);
* vector<int> param_2 = obj->retrieve(start,end,granularity);
*/
```

Java:

```
class LogSystem {

public LogSystem() {

}

public void put(int id, String timestamp) {

}

public List<Integer> retrieve(String start, String end, String granularity) {

}
}

/**
* Your LogSystem object will be instantiated and called as such:
* LogSystem obj = new LogSystem();
* obj.put(id,timestamp);
* List<Integer> param_2 = obj.retrieve(start,end,granularity);
*/
```

Python3:

```
class LogSystem:

def __init__(self):


def put(self, id: int, timestamp: str) -> None:


def retrieve(self, start: str, end: str, granularity: str) -> List[int]:



# Your LogSystem object will be instantiated and called as such:
```

```
# obj = LogSystem()
# obj.put(id,timestamp)
# param_2 = obj.retrieve(start,end,granularity)
```

Python:

```
class LogSystem(object):

def __init__(self):



def put(self, id, timestamp):
"""
:type id: int
:type timestamp: str
:rtype: None
"""



def retrieve(self, start, end, granularity):
"""
:type start: str
:type end: str
:type granularity: str
:rtype: List[int]
"""




# Your LogSystem object will be instantiated and called as such:
# obj = LogSystem()
# obj.put(id,timestamp)
# param_2 = obj.retrieve(start,end,granularity)
```

JavaScript:

```
var LogSystem = function() {

};

/**
```

```
* @param {number} id
* @param {string} timestamp
* @return {void}
*/
LogSystem.prototype.put = function(id, timestamp) {

};

/**
* @param {string} start
* @param {string} end
* @param {string} granularity
* @return {number[]}
*/
LogSystem.prototype.retrieve = function(start, end, granularity) {

};

/**
* Your LogSystem object will be instantiated and called as such:
* var obj = new LogSystem()
* obj.put(id,timestamp)
* var param_2 = obj.retrieve(start,end,granularity)
*/
```

TypeScript:

```
class LogSystem {
constructor() {

}

put(id: number, timestamp: string): void {

}

retrieve(start: string, end: string, granularity: string): number[] {

}
}

/**
```

```
* Your LogSystem object will be instantiated and called as such:
* var obj = new LogSystem()
* obj.put(id,timestamp)
* var param_2 = obj.retrieve(start,end,granularity)
*/
```

C#:

```
public class LogSystem {

public LogSystem() {

}

public void Put(int id, string timestamp) {

}

public IList<int> Retrieve(string start, string end, string granularity) {

}
}

/**
* Your LogSystem object will be instantiated and called as such:
* LogSystem obj = new LogSystem();
* obj.Put(id,timestamp);
* IList<int> param_2 = obj.Retrieve(start,end,granularity);
*/
```

C:

```
typedef struct {

} LogSystem;


LogSystem* logSystemCreate() {
}

void logSystemPut(LogSystem* obj, int id, char* timestamp) {

}

int* logSystemRetrieve(LogSystem* obj, char* start, char* end, char*
granularity, int* retSize) {

}

void logSystemFree(LogSystem* obj) {

}

/**
* Your LogSystem struct will be instantiated and called as such:
* LogSystem* obj = logSystemCreate();
* logSystemPut(obj, id, timestamp);

* int* param_2 = logSystemRetrieve(obj, start, end, granularity, retSize);

* logSystemFree(obj);
*/
```

Go:

```
type LogSystem struct {

}


func Constructor() LogSystem {

}


func (this *LogSystem) Put(id int, timestamp string) {

}
```

```
func (this *LogSystem) Retrieve(start string, end string, granularity string)
[]int {

}



/**
* Your LogSystem object will be instantiated and called as such:
* obj := Constructor();
* obj.Put(id,timestamp);
* param_2 := obj.Retrieve(start,end,granularity);
*/
```

Kotlin:

```
class LogSystem() {

fun put(id: Int, timestamp: String) {

}

fun retrieve(start: String, end: String, granularity: String): List<Int> {

}

}

/**
* Your LogSystem object will be instantiated and called as such:
* var obj = LogSystem()
* obj.put(id,timestamp)
* var param_2 = obj.retrieve(start,end,granularity)
*/
```

Swift:

```
class LogSystem {

init() {

}
func put(_ id: Int, _ timestamp: String) {

}

func retrieve(_ start: String, _ end: String, _ granularity: String) -> [Int]
{

}
}

/**
* Your LogSystem object will be instantiated and called as such:
* let obj = LogSystem()
* obj.put(id, timestamp)
* let ret_2: [Int] = obj.retrieve(start, end, granularity)
*/
```

Rust:

```
struct LogSystem {

}



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

fn new() -> Self {

}

fn put(&self, id: i32, timestamp: String) {

}

fn retrieve(&self, start: String, end: String, granularity: String) ->
Vec<i32> {
}
}

/**
* Your LogSystem object will be instantiated and called as such:
* let obj = LogSystem::new();
* obj.put(id, timestamp);
* let ret_2: Vec<i32> = obj.retrieve(start, end, granularity);
*/
```

Ruby:

```
class LogSystem
def initialize()

end



=begin
:type id: Integer
:type timestamp: String
:rtype: Void
=end
def put(id, timestamp)

end



=begin
:type start: String
:type end: String
:type granularity: String
:rtype: Integer[]
=end
def retrieve(start, end, granularity)

end



end

# Your LogSystem object will be instantiated and called as such:
```

```

# obj = LogSystem.new()
# obj.put(id, timestamp)
# param_2 = obj.retrieve(start, end, granularity)
```

PHP:

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

}

/**
* @param Integer $id
* @param String $timestamp
* @return NULL
*/
function put($id, $timestamp) {

}

/**
* @param String $start
* @param String $end
* @param String $granularity
* @return Integer[]
*/
function retrieve($start, $end, $granularity) {

}
}

/**
* Your LogSystem object will be instantiated and called as such:
* $obj = LogSystem();
* $obj->put($id, $timestamp);
* $ret_2 = $obj->retrieve($start, $end, $granularity);
*/
```

Dart:

```
class LogSystem {

LogSystem() {

}

void put(int id, String timestamp) {

}

List<int> retrieve(String start, String end, String granularity) {

}
}

/**
* Your LogSystem object will be instantiated and called as such:
* LogSystem obj = LogSystem();
* obj.put(id,timestamp);
* List<int> param2 = obj.retrieve(start,end,granularity);
*/
```

Scala:

```
class LogSystem() {

def put(id: Int, timestamp: String): Unit = {

}

def retrieve(start: String, end: String, granularity: String): List[Int] = {

}

}

/**
* Your LogSystem object will be instantiated and called as such:
* val obj = new LogSystem()
* obj.put(id,timestamp)
* val param_2 = obj.retrieve(start,end,granularity)
*/
```

Elixir:

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

end

@spec put(id :: integer, timestamp :: String.t) :: any
def put(id, timestamp) do

end

@spec retrieve(start :: String.t, end :: String.t, granularity :: String.t)
:: [integer]
def retrieve(start, end, granularity) do

end
end

# Your functions will be called as such:
# LogSystem.init_()
# LogSystem.put(id, timestamp)
# param_2 = LogSystem.retrieve(start, end, granularity)

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

Erlang:

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

-spec log_system_put(Id :: integer(), Timestamp :: unicode:unicode_binary())
-> any().
log_system_put(Id, Timestamp) ->
.

-spec log_system_retrieve(Start :: unicode:unicode_binary(), End ::
unicode:unicode_binary(), Granularity :: unicode:unicode_binary()) ->
[integer()].
```

```
log_system_retrieve(Start, End, Granularity) ->
.



%% Your functions will be called as such:
%% log_system_init_(),
%% log_system_put(Id, Timestamp),
%% Param_2 = log_system_retrieve(Start, End, Granularity),

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

Racket:

```
(define log-system%
(class object%
(super-new)

(init-field)

; put : exact-integer? string? -> void?
(define/public (put id timestamp)
)
; retrieve : string? string? string? -> (listof exact-integer?)
(define/public (retrieve start end granularity)
)))

;; Your log-system% object will be instantiated and called as such:
;; (define obj (new log-system%))
;; (send obj put id timestamp)
;; (define param_2 (send obj retrieve start end granularity))
```

Solutions

C++ Solution:

```
/*
* Problem: Design Log Storage System
* Difficulty: Medium
* Tags: string, hash
```

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

class LogSystem {
public:
LogSystem() {

}

void put(int id, string timestamp) {

}

vector<int> retrieve(string start, string end, string granularity) {

}
};

/**
* Your LogSystem object will be instantiated and called as such:
* LogSystem* obj = new LogSystem();
* obj->put(id,timestamp);
* vector<int> param_2 = obj->retrieve(start,end,granularity);
*/
```

Java Solution:

```
/**
* Problem: Design Log Storage System
* Difficulty: Medium
* Tags: string, hash
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class LogSystem {
```

```
public LogSystem() {

}

public void put(int id, String timestamp) {

}

public List<Integer> retrieve(String start, String end, String granularity) {

}
}

/**
* Your LogSystem object will be instantiated and called as such:
* LogSystem obj = new LogSystem();
* obj.put(id,timestamp);
* List<Integer> param_2 = obj.retrieve(start,end,granularity);
*/
```

Python3 Solution:

```
"""
Problem: Design Log Storage System
Difficulty: Medium
Tags: string, hash

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

class LogSystem:

def __init__(self):



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

Python Solution:

```
class LogSystem(object):

def __init__(self):



def put(self, id, timestamp):
"""
:type id: int
:type timestamp: str
:rtype: None
"""



def retrieve(self, start, end, granularity):
"""
:type start: str
:type end: str
:type granularity: str
:rtype: List[int]
"""




# Your LogSystem object will be instantiated and called as such:
# obj = LogSystem()
# obj.put(id,timestamp)
# param_2 = obj.retrieve(start,end,granularity)
```

JavaScript Solution:

```
/**
* Problem: Design Log Storage System
* Difficulty: Medium
* Tags: string, hash
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/
```

```


var LogSystem = function() {

};

/**
* @param {number} id
* @param {string} timestamp
* @return {void}
*/
LogSystem.prototype.put = function(id, timestamp) {

};

/**
* @param {string} start
* @param {string} end
* @param {string} granularity
* @return {number[]}
*/
LogSystem.prototype.retrieve = function(start, end, granularity) {

};

/**
* Your LogSystem object will be instantiated and called as such:
* var obj = new LogSystem()
* obj.put(id,timestamp)
* var param_2 = obj.retrieve(start,end,granularity)
*/
```

TypeScript Solution:

```
/**
* Problem: Design Log Storage System
* Difficulty: Medium
* Tags: string, hash
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
```

```
*/

class LogSystem {
constructor() {

}

put(id: number, timestamp: string): void {

}

retrieve(start: string, end: string, granularity: string): number[] {

}
}

/**
* Your LogSystem object will be instantiated and called as such:
* var obj = new LogSystem()
* obj.put(id,timestamp)
* var param_2 = obj.retrieve(start,end,granularity)
*/
```

C# Solution:

```
/*
* Problem: Design Log Storage System
* Difficulty: Medium
* Tags: string, hash
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

public class LogSystem {

public LogSystem() {

}
public void Put(int id, string timestamp) {

}

public IList<int> Retrieve(string start, string end, string granularity) {

}
}

/**
* Your LogSystem object will be instantiated and called as such:
* LogSystem obj = new LogSystem();
* obj.Put(id,timestamp);
* IList<int> param_2 = obj.Retrieve(start,end,granularity);
*/
```

C Solution:

```
/*
* Problem: Design Log Storage System
* Difficulty: Medium
* Tags: string, hash
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/




typedef struct {

} LogSystem;


LogSystem* logSystemCreate() {

}

void logSystemPut(LogSystem* obj, int id, char* timestamp) {
}

int* logSystemRetrieve(LogSystem* obj, char* start, char* end, char*
granularity, int* retSize) {

}

void logSystemFree(LogSystem* obj) {

}

/**
* Your LogSystem struct will be instantiated and called as such:
* LogSystem* obj = logSystemCreate();
* logSystemPut(obj, id, timestamp);

* int* param_2 = logSystemRetrieve(obj, start, end, granularity, retSize);

* logSystemFree(obj);
*/
```

Go Solution:

```
// Problem: Design Log Storage System
// Difficulty: Medium
// Tags: string, hash
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

type LogSystem struct {

}



func Constructor() LogSystem {

}
```

```
func (this *LogSystem) Put(id int, timestamp string) {

}


func (this *LogSystem) Retrieve(start string, end string, granularity string)
[]int {

}


/**
* Your LogSystem object will be instantiated and called as such:
* obj := Constructor();
* obj.Put(id,timestamp);
* param_2 := obj.Retrieve(start,end,granularity);
*/
```

Kotlin Solution:

```
class LogSystem() {

fun put(id: Int, timestamp: String) {

}

fun retrieve(start: String, end: String, granularity: String): List<Int> {

}

}

/**
* Your LogSystem object will be instantiated and called as such:
* var obj = LogSystem()
* obj.put(id,timestamp)
* var param_2 = obj.retrieve(start,end,granularity)
*/
```

Swift Solution:

```
class LogSystem {

init() {

}

func put(_ id: Int, _ timestamp: String) {

}

func retrieve(_ start: String, _ end: String, _ granularity: String) -> [Int]
{

}
}

/**
* Your LogSystem object will be instantiated and called as such:
* let obj = LogSystem()
* obj.put(id, timestamp)
* let ret_2: [Int] = obj.retrieve(start, end, granularity)
*/
```

Rust Solution:

```
// Problem: Design Log Storage System
// Difficulty: Medium
// Tags: string, hash
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

struct LogSystem {

}



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

```
*/
impl LogSystem {

fn new() -> Self {

}

fn put(&self, id: i32, timestamp: String) {

}

fn retrieve(&self, start: String, end: String, granularity: String) ->
Vec<i32> {

}
}

/**
* Your LogSystem object will be instantiated and called as such:
* let obj = LogSystem::new();
* obj.put(id, timestamp);
* let ret_2: Vec<i32> = obj.retrieve(start, end, granularity);
*/
```

Ruby Solution:

```
class LogSystem
def initialize()

end



=begin
:type id: Integer
:type timestamp: String
:rtype: Void
=end
def put(id, timestamp)

end
```

```
=begin
:type start: String
:type end: String
:type granularity: String
:rtype: Integer[]
=end
def retrieve(start, end, granularity)

end


end

# Your LogSystem object will be instantiated and called as such:
# obj = LogSystem.new()
# obj.put(id, timestamp)
# param_2 = obj.retrieve(start, end, granularity)
```

PHP Solution:

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

}

/**
* @param Integer $id
* @param String $timestamp
* @return NULL
*/
function put($id, $timestamp) {

}

/**
* @param String $start
* @param String $end
* @param String $granularity
* @return Integer[]
*/
function retrieve($start, $end, $granularity) {

}
}

/**
* Your LogSystem object will be instantiated and called as such:
* $obj = LogSystem();
* $obj->put($id, $timestamp);
* $ret_2 = $obj->retrieve($start, $end, $granularity);
*/
```

Dart Solution:

```
class LogSystem {

LogSystem() {

}

void put(int id, String timestamp) {

}

List<int> retrieve(String start, String end, String granularity) {

}
}

/**
* Your LogSystem object will be instantiated and called as such:
* LogSystem obj = LogSystem();
* obj.put(id,timestamp);
* List<int> param2 = obj.retrieve(start,end,granularity);
*/
```

Scala Solution:

```
class LogSystem() {

def put(id: Int, timestamp: String): Unit = {

}

def retrieve(start: String, end: String, granularity: String): List[Int] = {

}

}

/**
* Your LogSystem object will be instantiated and called as such:
* val obj = new LogSystem()
* obj.put(id,timestamp)
* val param_2 = obj.retrieve(start,end,granularity)
*/
```

Elixir Solution:

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

end

@spec put(id :: integer, timestamp :: String.t) :: any
def put(id, timestamp) do

end

@spec retrieve(start :: String.t, end :: String.t, granularity :: String.t)
:: [integer]
def retrieve(start, end, granularity) do

end
end

# Your functions will be called as such:
# LogSystem.init_()
# LogSystem.put(id, timestamp)
# param_2 = LogSystem.retrieve(start, end, granularity)

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

Erlang Solution:

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

-spec log_system_put(Id :: integer(), Timestamp :: unicode:unicode_binary())
-> any().
log_system_put(Id, Timestamp) ->
.

-spec log_system_retrieve(Start :: unicode:unicode_binary(), End ::
unicode:unicode_binary(), Granularity :: unicode:unicode_binary()) ->
[integer()].
log_system_retrieve(Start, End, Granularity) ->
.



%% Your functions will be called as such:
%% log_system_init_(),
%% log_system_put(Id, Timestamp),
%% Param_2 = log_system_retrieve(Start, End, Granularity),

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

Racket Solution:

```
(define log-system%
(class object%
(super-new)

(init-field)

; put : exact-integer? string? -> void?
(define/public (put id timestamp)
)
; retrieve : string? string? string? -> (listof exact-integer?)
(define/public (retrieve start end granularity)
)))

;; Your log-system% object will be instantiated and called as such:
;; (define obj (new log-system%))
;; (send obj put id timestamp)
;; (define param_2 (send obj retrieve start end granularity))
```