

Problem 981: Time Based Key-Value Store

Problem Information

Difficulty: Medium

Acceptance Rate: 0.00%

Paid Only: No

Problem Description

Design a time-based key-value data structure that can store multiple values for the same key at different time stamps and retrieve the key's value at a certain timestamp.

Implement the

TimeMap

class:

TimeMap()

Initializes the object of the data structure.

void set(String key, String value, int timestamp)

Stores the key

key

with the value

value

at the given time

timestamp

```
String get(String key, int timestamp)
```

Returns a value such that

set

was called previously, with

timestamp_prev <= timestamp

. If there are multiple such values, it returns the value associated with the largest

timestamp_prev

. If there are no values, it returns

""

Example 1:

Input

```
["TimeMap", "set", "get", "get", "set", "get", "get"] [], ["foo", "bar", 1], ["foo", 1], ["foo", 3], ["foo", "bar2", 4], ["foo", 4], ["foo", 5]]
```

Output

```
[null, null, "bar", "bar", null, "bar2", "bar2"]
```

Explanation

```
TimeMap timeMap = new TimeMap(); timeMap.set("foo", "bar", 1); // store the key "foo" and  
value "bar" along with timestamp = 1. timeMap.get("foo", 1); // return "bar" timeMap.get("foo",  
3); // return "bar", since there is no value corresponding to foo at timestamp 3 and timestamp  
2, then the only value is at timestamp 1 is "bar". timeMap.set("foo", "bar2", 4); // store the key
```

```
"foo" and value "bar2" along with timestamp = 4. timeMap.get("foo", 4); // return "bar2"  
timeMap.get("foo", 5); // return "bar2"
```

Constraints:

$1 \leq \text{key.length}, \text{value.length} \leq 100$

key

and

value

consist of lowercase English letters and digits.

$1 \leq \text{timestamp} \leq 10$

7

All the timestamps

timestamp

of

set

are strictly increasing.

At most

$2 * 10$

5

calls will be made to

set

and

get

Code Snippets

C++:

```
class TimeMap {  
public:  
TimeMap() {  
  
}  
  
void set(string key, string value, int timestamp) {  
  
}  
  
string get(string key, int timestamp) {  
  
}  
};  
  
/**  
* Your TimeMap object will be instantiated and called as such:  
* TimeMap* obj = new TimeMap();  
* obj->set(key,value,timestamp);  
* string param_2 = obj->get(key,timestamp);  
*/
```

Java:

```
class TimeMap {  
  
public TimeMap() {  
  
}  
  
public void set(String key, String value, int timestamp) {
```

```

}

public String get(String key, int timestamp) {

}

/**
 * Your TimeMap object will be instantiated and called as such:
 * TimeMap obj = new TimeMap();
 * obj.set(key,value,timestamp);
 * String param_2 = obj.get(key,timestamp);
 */

```

Python3:

```

class TimeMap:

def __init__(self):

def set(self, key: str, value: str, timestamp: int) -> None:

def get(self, key: str, timestamp: int) -> str:

# Your TimeMap object will be instantiated and called as such:
# obj = TimeMap()
# obj.set(key,value,timestamp)
# param_2 = obj.get(key,timestamp)

```

Python:

```

class TimeMap(object):

def __init__(self):

def set(self, key, value, timestamp):

```

```

"""
:type key: str
:type value: str
:type timestamp: int
:rtype: None
"""

def get(self, key, timestamp):
"""
:type key: str
:type timestamp: int
:rtype: str
"""

# Your TimeMap object will be instantiated and called as such:
# obj = TimeMap()
# obj.set(key,value,timestamp)
# param_2 = obj.get(key,timestamp)

```

JavaScript:

```

var TimeMap = function() {

};

/**
 * @param {string} key
 * @param {string} value
 * @param {number} timestamp
 * @return {void}
 */
TimeMap.prototype.set = function(key, value, timestamp) {

};

/**
 * @param {string} key
 * @param {number} timestamp

```

```

* @return {string}
*/
TimeMap.prototype.get = function(key, timestamp) {

};

/**
* Your TimeMap object will be instantiated and called as such:
* var obj = new TimeMap()
* obj.set(key,value,timestamp)
* var param_2 = obj.get(key,timestamp)
*/

```

TypeScript:

```

class TimeMap {
constructor() {

}

set(key: string, value: string, timestamp: number): void {

}

get(key: string, timestamp: number): string {

}

/**
* Your TimeMap object will be instantiated and called as such:
* var obj = new TimeMap()
* obj.set(key,value,timestamp)
* var param_2 = obj.get(key,timestamp)
*/

```

C#:

```

public class TimeMap {

public TimeMap() {

```

```
}

public void Set(string key, string value, int timestamp) {

}

public string Get(string key, int timestamp) {

}

/**
* Your TimeMap object will be instantiated and called as such:
* TimeMap obj = new TimeMap();
* obj.Set(key,value,timestamp);
* string param_2 = obj.Get(key,timestamp);
*/

```

C:

```
typedef struct {

} TimeMap;

TimeMap* timeMapCreate() {

}

void timeMapSet(TimeMap* obj, char* key, char* value, int timestamp) {

}

char* timeMapGet(TimeMap* obj, char* key, int timestamp) {

}

void timeMapFree(TimeMap* obj) {
```

```
}
```

```
/**
```

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

```
* TimeMap* obj = timeMapCreate();
```

```
* timeMapSet(obj, key, value, timestamp);
```

```
* char* param_2 = timeMapGet(obj, key, timestamp);
```

```
* timeMapFree(obj);
```

```
*/
```

Go:

```
type TimeMap struct {
```

```
}
```

```
func Constructor() TimeMap {
```

```
}
```

```
func (this *TimeMap) Set(key string, value string, timestamp int) {
```

```
}
```

```
func (this *TimeMap) Get(key string, timestamp int) string {
```

```
}
```

```
/**
```

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

```
* obj := Constructor();
```

```
* obj.Set(key,value,timestamp);
```

```
* param_2 := obj.Get(key,timestamp);
```

```
*/
```

Kotlin:

```
class TimeMap() {  
  
    fun set(key: String, value: String, timestamp: Int) {  
  
    }  
  
    fun get(key: String, timestamp: Int): String {  
  
    }  
  
}  
  
/**  
 * Your TimeMap object will be instantiated and called as such:  
 * var obj = TimeMap()  
 * obj.set(key,value,timestamp)  
 * var param_2 = obj.get(key,timestamp)  
 */
```

Swift:

```
class TimeMap {  
  
    init() {  
  
    }  
  
    func set(_ key: String, _ value: String, _ timestamp: Int) {  
  
    }  
  
    func get(_ key: String, _ timestamp: Int) -> String {  
  
    }  
  
}  
  
/**  
 * Your TimeMap object will be instantiated and called as such:  
 * let obj = TimeMap()  
 * obj.set(key, value, timestamp)
```

```
* let ret_2: String = obj.get(key, timestamp)
*/
```

Rust:

```
struct TimeMap {

}

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

    fn new() -> Self {
        ...
    }

    fn set(&self, key: String, value: String, timestamp: i32) {
        ...
    }

    fn get(&self, key: String, timestamp: i32) -> String {
        ...
    }
}

/***
* Your TimeMap object will be instantiated and called as such:
* let obj = TimeMap::new();
* obj.set(key, value, timestamp);
* let ret_2: String = obj.get(key, timestamp);
*/

```

Ruby:

```
class TimeMap
  def initialize()
  end
```

```

=begin
:type key: String
:type value: String
:type timestamp: Integer
:rtype: Void
=end

def set(key, value, timestamp)

end

=begin
:type key: String
:type timestamp: Integer
:rtype: String
=end

def get(key, timestamp)

end

end

# Your TimeMap object will be instantiated and called as such:
# obj = TimeMap.new()
# obj.set(key, value, timestamp)
# param_2 = obj.get(key, timestamp)

```

PHP:

```

class TimeMap {

/**
 */
function __construct() {

}

/**
 * @param String $key
 * @param String $value

```

```

* @param Integer $timestamp
* @return NULL
*/
function set($key, $value, $timestamp) {

}

/**
* @param String $key
* @param Integer $timestamp
* @return String
*/
function get($key, $timestamp) {

}

/**
* Your TimeMap object will be instantiated and called as such:
* $obj = TimeMap();
* $obj->set($key, $value, $timestamp);
* $ret_2 = $obj->get($key, $timestamp);
*/

```

Dart:

```

class TimeMap {

TimeMap() {

}

void set(String key, String value, int timestamp) {

}

String get(String key, int timestamp) {

}

/**

```

```
* Your TimeMap object will be instantiated and called as such:  
* TimeMap obj = TimeMap();  
* obj.set(key,value,timestamp);  
* String param2 = obj.get(key,timestamp);  
*/
```

Scala:

```
class TimeMap() {  
  
    def set(key: String, value: String, timestamp: Int): Unit = {  
  
    }  
  
    def get(key: String, timestamp: Int): String = {  
  
    }  
  
    /**  
     * Your TimeMap object will be instantiated and called as such:  
     * val obj = new TimeMap()  
     * obj.set(key,value,timestamp)  
     * val param_2 = obj.get(key,timestamp)  
     */
```

Elixir:

```
defmodule TimeMap do  
  @spec init_() :: any  
  def init_() do  
  
  end  
  
  @spec set(key :: String.t, value :: String.t, timestamp :: integer) :: any  
  def set(key, value, timestamp) do  
  
  end  
  
  @spec get(key :: String.t, timestamp :: integer) :: String.t  
  def get(key, timestamp) do  
  
  end
```

```

end
end

# Your functions will be called as such:
# TimeMap.init_()
# TimeMap.set(key, value, timestamp)
# param_2 = TimeMap.get(key, timestamp)

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

```

Erlang:

```

-spec time_map_init_() -> any().
time_map_init_() ->
.

-spec time_map_set(Key :: unicode:unicode_binary(), Value :: unicode:unicode_binary(), Timestamp :: integer()) -> any().
time_map_set(Key, Value, Timestamp) ->
.

-spec time_map_get(Key :: unicode:unicode_binary(), Timestamp :: integer())
-> unicode:unicode_binary().
time_map_get(Key, Timestamp) ->
.

%% Your functions will be called as such:
%% time_map_init(),
%% time_map_set(Key, Value, Timestamp),
%% Param_2 = time_map_get(Key, Timestamp),

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

```

Racket:

```

(define time-map%
  (class object%
    (super-new))

```

```

(init-field)

; set : string? string? exact-integer? -> void?
(define/public (set key value timestamp)
)
; get : string? exact-integer? -> string?
(define/public (get key timestamp)
))

;; Your time-map% object will be instantiated and called as such:
;; (define obj (new time-map%))
;; (send obj set key value timestamp)
;; (define param_2 (send obj get key timestamp))

```

Solutions

C++ Solution:

```

/*
 * Problem: Time Based Key-Value Store
 * Difficulty: Medium
 * Tags: string, hash, search
 *
 * 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 TimeMap {
public:
TimeMap() {

}

void set(string key, string value, int timestamp) {

}

string get(string key, int timestamp) {

```

```

}

};

/***
* Your TimeMap object will be instantiated and called as such:
* TimeMap* obj = new TimeMap();
* obj->set(key,value,timestamp);
* string param_2 = obj->get(key,timestamp);
*/

```

Java Solution:

```

/**
* Problem: Time Based Key-Value Store
* Difficulty: Medium
* Tags: string, hash, search
*
* 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 TimeMap {

    public TimeMap() {

    }

    public void set(String key, String value, int timestamp) {

    }

    public String get(String key, int timestamp) {

    }
}

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

```

```
* obj.set(key,value,timestamp);
* String param_2 = obj.get(key,timestamp);
*/
```

Python3 Solution:

```
"""
Problem: Time Based Key-Value Store
Difficulty: Medium
Tags: string, hash, search

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 TimeMap:

    def __init__(self):

        def set(self, key: str, value: str, timestamp: int) -> None:
            # TODO: Implement optimized solution
            pass
```

Python Solution:

```
class TimeMap(object):

    def __init__(self):

        def set(self, key, value, timestamp):
            """
            :type key: str
            :type value: str
            :type timestamp: int
            :rtype: None
            """
```

```

def get(self, key, timestamp):
    """
    :type key: str
    :type timestamp: int
    :rtype: str
    """

    # Your TimeMap object will be instantiated and called as such:
    # obj = TimeMap()
    # obj.set(key,value,timestamp)
    # param_2 = obj.get(key,timestamp)

```

JavaScript Solution:

```

/**
 * Problem: Time Based Key-Value Store
 * Difficulty: Medium
 * Tags: string, hash, search
 *
 * 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 TimeMap = function() {

};

/**
 * @param {string} key
 * @param {string} value
 * @param {number} timestamp
 * @return {void}
 */
TimeMap.prototype.set = function(key, value, timestamp) {

};

```

```

/**
 * @param {string} key
 * @param {number} timestamp
 * @return {string}
 */
TimeMap.prototype.get = function(key, timestamp) {

};

/**
 * Your TimeMap object will be instantiated and called as such:
 * var obj = new TimeMap()
 * obj.set(key,value,timestamp)
 * var param_2 = obj.get(key,timestamp)
 */

```

TypeScript Solution:

```

/**
 * Problem: Time Based Key-Value Store
 * Difficulty: Medium
 * Tags: string, hash, search
 *
 * 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 TimeMap {
constructor() {

}

set(key: string, value: string, timestamp: number): void {

}

get(key: string, timestamp: number): string {

}
}

```

```

/**
 * Your TimeMap object will be instantiated and called as such:
 * var obj = new TimeMap()
 * obj.set(key,value,timestamp)
 * var param_2 = obj.get(key,timestamp)
 */

```

C# Solution:

```

/*
 * Problem: Time Based Key-Value Store
 * Difficulty: Medium
 * Tags: string, hash, search
 *
 * 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 TimeMap {

    public TimeMap() {

    }

    public void Set(string key, string value, int timestamp) {

    }

    public string Get(string key, int timestamp) {

    }

}

/**
 * Your TimeMap object will be instantiated and called as such:
 * TimeMap obj = new TimeMap();
 * obj.set(key,value,timestamp);
 * string param_2 = obj.get(key,timestamp);
 */

```

C Solution:

```
/*
 * Problem: Time Based Key-Value Store
 * Difficulty: Medium
 * Tags: string, hash, search
 *
 * 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 {

} TimeMap;

TimeMap* timeMapCreate() {

}

void timeMapSet(TimeMap* obj, char* key, char* value, int timestamp) {

char* timeMapGet(TimeMap* obj, char* key, int timestamp) {

void timeMapFree(TimeMap* obj) {

}

/***
 * Your TimeMap struct will be instantiated and called as such:
 * TimeMap* obj = timeMapCreate();
 * timeMapSet(obj, key, value, timestamp);
 *
 * char* param_2 = timeMapGet(obj, key, timestamp);
 */
```

```
* timeMapFree(obj);  
*/
```

Go Solution:

```
// Problem: Time Based Key-Value Store  
// Difficulty: Medium  
// Tags: string, hash, search  
  
// 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 TimeMap struct {  
  
}  
  
func Constructor() TimeMap {  
  
}  
  
func (this *TimeMap) Set(key string, value string, timestamp int) {  
  
}  
  
func (this *TimeMap) Get(key string, timestamp int) string {  
  
}  
  
/**  
* Your TimeMap object will be instantiated and called as such:  
* obj := Constructor();  
* obj.Set(key,value,timestamp);  
* param_2 := obj.Get(key,timestamp);  
*/
```

Kotlin Solution:

```
class TimeMap() {  
  
    fun set(key: String, value: String, timestamp: Int) {  
  
    }  
  
    fun get(key: String, timestamp: Int): String {  
  
    }  
  
    /**  
     * Your TimeMap object will be instantiated and called as such:  
     * var obj = TimeMap()  
     * obj.set(key,value,timestamp)  
     * var param_2 = obj.get(key,timestamp)  
     */
```

Swift Solution:

```
class TimeMap {  
  
    init() {  
  
    }  
  
    func set(_ key: String, _ value: String, _ timestamp: Int) {  
  
    }  
  
    func get(_ key: String, _ timestamp: Int) -> String {  
  
    }  
  
    /**  
     * Your TimeMap object will be instantiated and called as such:  
     * let obj = TimeMap()  
     * obj.set(key, value, timestamp)
```

```
* let ret_2: String = obj.get(key, timestamp)
*/
```

Rust Solution:

```
// Problem: Time Based Key-Value Store
// Difficulty: Medium
// Tags: string, hash, search
//
// 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 TimeMap {

}

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

    fn new() -> Self {
        }
    }

    fn set(&self, key: String, value: String, timestamp: i32) {
        }
    }

    fn get(&self, key: String, timestamp: i32) -> String {
        }
    }

}

/**
 * Your TimeMap object will be instantiated and called as such:
 * let obj = TimeMap::new();
 * obj.set(key, value, timestamp);
 */
```

```
* let ret_2: String = obj.get(key, timestamp);
*/
```

Ruby Solution:

```
class TimeMap
def initialize()

end

=begin
:type key: String
:type value: String
:type timestamp: Integer
:rtype: Void
=end
def set(key, value, timestamp)

end

=begin
:type key: String
:type timestamp: Integer
:rtype: String
=end
def get(key, timestamp)

end

end

# Your TimeMap object will be instantiated and called as such:
# obj = TimeMap.new()
# obj.set(key, value, timestamp)
# param_2 = obj.get(key, timestamp)
```

PHP Solution:

```

class TimeMap {
    /**
     */
    function __construct() {

    }

    /**
     * @param String $key
     * @param String $value
     * @param Integer $timestamp
     * @return NULL
     */
    function set($key, $value, $timestamp) {

    }

    /**
     * @param String $key
     * @param Integer $timestamp
     * @return String
     */
    function get($key, $timestamp) {

    }
}

/**
* Your TimeMap object will be instantiated and called as such:
* $obj = TimeMap();
* $obj->set($key, $value, $timestamp);
* $ret_2 = $obj->get($key, $timestamp);
*/

```

Dart Solution:

```

class TimeMap {

TimeMap() {
}

```

```

void set(String key, String value, int timestamp) {

}

String get(String key, int timestamp) {

}

/**
* Your TimeMap object will be instantiated and called as such:
* TimeMap obj = TimeMap();
* obj.set(key,value,timestamp);
* String param2 = obj.get(key,timestamp);
*/

```

Scala Solution:

```

class TimeMap() {

def set(key: String, value: String, timestamp: Int): Unit = {

}

def get(key: String, timestamp: Int): String = {

}

/**
* Your TimeMap object will be instantiated and called as such:
* val obj = new TimeMap()
* obj.set(key,value,timestamp)
* val param_2 = obj.get(key,timestamp)
*/

```

Elixir Solution:

```

defmodule TimeMap do
@spec init_() :: any

```

```

def init_() do
  end

  @spec set(key :: String.t, value :: String.t, timestamp :: integer) :: any
  def set(key, value, timestamp) do
    end

  @spec get(key :: String.t, timestamp :: integer) :: String.t
  def get(key, timestamp) do
    end
  end

  # Your functions will be called as such:
  # TimeMap.init_()
  # TimeMap.set(key, value, timestamp)
  # param_2 = TimeMap.get(key, timestamp)

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

```

Erlang Solution:

```

-spec time_map_init_() -> any().
time_map_init_() ->
  .

-spec time_map_set(Key :: unicode:unicode_binary(), Value :: unicode:unicode_binary(), Timestamp :: integer()) -> any().
time_map_set(Key, Value, Timestamp) ->
  .

-spec time_map_get(Key :: unicode:unicode_binary(), Timestamp :: integer()) -> unicode:unicode_binary().
time_map_get(Key, Timestamp) ->
  .

%% Your functions will be called as such:

```

```

%% time_map_init(),
%% time_map_set(Key, Value, Timestamp),
%% Param_2 = time_map_get(Key, Timestamp),

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

```

Racket Solution:

```

(define time-map%
  (class object%
    (super-new)

    (init-field)

    ; set : string? string? exact-integer? -> void?
    (define/public (set key value timestamp)
      )
    ; get : string? exact-integer? -> string?
    (define/public (get key timestamp)
      )))

;; Your time-map% object will be instantiated and called as such:
;; (define obj (new time-map%))
;; (send obj set key value timestamp)
;; (define param_2 (send obj get key timestamp))

```