Problem 2102: Sequentially Ordinal Rank Tracker

Problem Information

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

Problem Description

A scenic location is represented by its

name

and attractiveness

score

, where

name

is a

unique

string among all locations and

score

is an integer. Locations can be ranked from the best to the worst. The

higher

the score, the better the location. If the scores of two locations are equal, then the location with the

lexicographically smaller

name is better.

You are building a system that tracks the ranking of locations with the system initially starting with no locations. It supports:

Adding

scenic locations,

one at a time

.

Querying

the

i

th

best

location of

all locations already added

, where

i

is the number of times the system has been queried (including the current query).

For example, when the system is queried for the

4

th

time, it returns the

4

th

best location of all locations already added.

Note that the test data are generated so that

at any time

, the number of queries

does not exceed

the number of locations added to the system.

Implement the

SORTracker

class:

SORTracker()

Initializes the tracker system.

void add(string name, int score)

Adds a scenic location with

name

and

score

to the system.

string get()

Queries and returns the

i

th

best location, where

i

is the number of times this method has been invoked (including this invocation).

Example 1:

Input

["SORTracker", "add", "add", "get", "add", "get", "add", "get", "add", "get", "add", "get", "get"]
[[], ["bradford", 2], ["branford", 3], [], ["alps", 2], [], ["orland", 2], [], ["orlando", 3], [], ["alpine", 2], [], []]

Output

[null, null, null, "branford", null, "alps", null, "bradford", null, "bradford", null, "bradford", "orland"]

Explanation

SORTracker tracker = new SORTracker(); // Initialize the tracker system.
tracker.add("bradford", 2); // Add location with name="bradford" and score=2 to the system.
tracker.add("branford", 3); // Add location with name="branford" and score=3 to the system.
tracker.get(); // The sorted locations, from best to worst, are: branford, bradford. // Note that

branford precedes bradford due to its

higher score

(3 > 2). // This is the 1

st

time get() is called, so return the best location: "branford". tracker.add("alps", 2); // Add location with name="alps" and score=2 to the system. tracker.get(); // Sorted locations: branford, alps, bradford. // Note that alps precedes bradford even though they have the same score (2). // This is because "alps" is

lexicographically smaller

than "bradford". // Return the 2

nd

best location "alps", as it is the 2

nd

time get() is called. tracker.add("orland", 2); // Add location with name="orland" and score=2 to the system. tracker.get(); // Sorted locations: branford, alps, bradford, orland. // Return "bradford", as it is the 3

rd

time get() is called. tracker.add("orlando", 3); // Add location with name="orlando" and score=3 to the system. tracker.get(); // Sorted locations: branford, orlando, alps, bradford, orland. // Return "bradford". tracker.add("alpine", 2); // Add location with name="alpine" and score=2 to the system. tracker.get(); // Sorted locations: branford, orlando, alpine, alps, bradford, orland. // Return "bradford". tracker.get(); // Sorted locations: branford, orlando, alpine, alps, bradford, orland. // Return "orland".

Constraints:

name

consists of lowercase English letters, and is unique among all locations.

1 <= name.length <= 10

1 <= score <= 10

5

At any time, the number of calls to

get

does not exceed the number of calls to

add

.

At most

4 * 10

4

calls

in total

will be made to

add

and

get

.

Code Snippets

C++:

```
class SORTracker {
public:
SORTracker() {

}

void add(string name, int score) {

}

string get() {

}
};

/**
* Your SORTracker object will be instantiated and called as such:
* SORTracker* obj = new SORTracker();
* obj->add(name,score);
* string param_2 = obj->get();
*/
```

Java:

```
class SORTracker {

public SORTracker() {

}

public void add(String name, int score) {

}

public String get() {

}
}
```

```
/**
* Your SORTracker object will be instantiated and called as such:
* SORTracker obj = new SORTracker();
* obj.add(name,score);
* String param_2 = obj.get();
*/
```

Python3:

```
class SORTracker:

def __init__(self):


def add(self, name: str, score: int) -> None:


def get(self) -> str:



# Your SORTracker object will be instantiated and called as such:
# obj = SORTracker()
# obj.add(name,score)
# param_2 = obj.get()
```

Python:

```
class SORTracker(object):

def __init__(self):


def add(self, name, score):
"""
:type name: str
:type score: int
:rtype: None
"""


def get(self):
```

```
"""
:rtype: str
"""



# Your SORTracker object will be instantiated and called as such:
# obj = SORTracker()
# obj.add(name,score)
# param_2 = obj.get()
```

JavaScript:

```
var SORTracker = function() {

};

/**
* @param {string} name
* @param {number} score
* @return {void}
*/
SORTracker.prototype.add = function(name, score) {

};

/**
* @return {string}
*/
SORTracker.prototype.get = function() {

};

/**
* Your SORTracker object will be instantiated and called as such:
* var obj = new SORTracker()
* obj.add(name,score)
* var param_2 = obj.get()
*/
```

TypeScript:

```
class SORTracker {
constructor() {

}

add(name: string, score: number): void {

}

get(): string {

}
}

/**
* Your SORTracker object will be instantiated and called as such:
* var obj = new SORTracker()
* obj.add(name,score)
* var param_2 = obj.get()
*/
```

C#:

```
public class SORTracker {

public SORTracker() {

}

public void Add(string name, int score) {

}

public string Get() {

}
}

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

```
* obj.Add(name,score);
* string param_2 = obj.Get();
*/
```

C:

```
typedef struct {

} SORTracker;


SORTracker* sORTrackerCreate() {

}

void sORTrackerAdd(SORTracker* obj, char* name, int score) {

}

char* sORTrackerGet(SORTracker* obj) {

}

void sORTrackerFree(SORTracker* obj) {

}

/**
* Your SORTracker struct will be instantiated and called as such:
* SORTracker* obj = sORTrackerCreate();
* sORTrackerAdd(obj, name, score);

* char* param_2 = sORTrackerGet(obj);

* sORTrackerFree(obj);
*/
```

Go:

```
type SORTracker struct {

}


func Constructor() SORTracker {

}


func (this *SORTracker) Add(name string, score int) {

}


func (this *SORTracker) Get() string {

}


/**
* Your SORTracker object will be instantiated and called as such:
* obj := Constructor();
* obj.Add(name,score);
* param_2 := obj.Get();
*/
```

Kotlin:

```
class SORTracker() {

fun add(name: String, score: Int) {

}

fun get(): String {

}

}

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

```
* var obj = SORTracker()
* obj.add(name,score)
* var param_2 = obj.get()
*/
```

Swift:

```
class SORTracker {

init() {

}

func add(_ name: String, _ score: Int) {

}

func get() -> String {

}
}

/**
* Your SORTracker object will be instantiated and called as such:
* let obj = SORTracker()
* obj.add(name, score)
* let ret_2: String = obj.get()
*/
```

Rust:

```
struct SORTracker {

}


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

```
fn new() -> Self {

}

fn add(&self, name: String, score: i32) {

}

fn get(&self) -> String {

}
}

/**
* Your SORTracker object will be instantiated and called as such:
* let obj = SORTracker::new();
* obj.add(name, score);
* let ret_2: String = obj.get();
*/
```

Ruby:

```
class SORTracker
def initialize()

end


=begin
:type name: String
:type score: Integer
:rtype: Void
=end
def add(name, score)

end


=begin
:rtype: String
=end
```

```
def get()

end


end

# Your SORTracker object will be instantiated and called as such:
# obj = SORTracker.new()
# obj.add(name, score)
# param_2 = obj.get()
```

PHP:

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

}

/**
* @param String $name
* @param Integer $score
* @return NULL
*/
function add($name, $score) {

}

/**
* @return String
*/
function get() {

}
}

/**
* Your SORTracker object will be instantiated and called as such:
* $obj = SORTracker();
* $obj->add($name, $score);
```

```
* $ret_2 = $obj->get();
*/
```

Dart:

```
class SORTracker {

SORTracker() {

}

void add(String name, int score) {

}

String get() {

}
}

/**
* Your SORTracker object will be instantiated and called as such:
* SORTracker obj = SORTracker();
* obj.add(name,score);
* String param2 = obj.get();
*/
```

Scala:

```
class SORTracker() {

def add(name: String, score: Int): Unit = {

}

def get(): String = {

}

}

/**
```

```
* Your SORTracker object will be instantiated and called as such:
* val obj = new SORTracker()
* obj.add(name,score)
* val param_2 = obj.get()
*/
```

Elixir:

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

end

@spec add(name :: String.t, score :: integer) :: any
def add(name, score) do

end

@spec get() :: String.t
def get() do

end
end

# Your functions will be called as such:
# SORTracker.init_()
# SORTracker.add(name, score)
# param_2 = SORTracker.get()

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

Erlang:

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

-spec sor_tracker_add(Name :: unicode:unicode_binary(), Score :: integer())
-> any().
sor_tracker_add(Name, Score) ->
```

```
.

-spec sor_tracker_get() -> unicode:unicode_binary().
sor_tracker_get() ->
.


%% Your functions will be called as such:
%% sor_tracker_init_(),
%% sor_tracker_add(Name, Score),
%% Param_2 = sor_tracker_get(),

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

Racket:

```
(define sor-tracker%
(class object%
(super-new)

(init-field)

; add : string? exact-integer? -> void?
(define/public (add name score)
)
; get : -> string?
(define/public (get)
)))

;; Your sor-tracker% object will be instantiated and called as such:
;; (define obj (new sor-tracker%))
;; (send obj add name score)
;; (define param_2 (send obj get))
```

Solutions

C++ Solution:

```
/*
* Problem: Sequentially Ordinal Rank Tracker
* Difficulty: Hard
* Tags: string, graph, sort, queue, heap
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class SORTracker {
public:
SORTracker() {

}

void add(string name, int score) {

}

string get() {

}
};

/**
* Your SORTracker object will be instantiated and called as such:
* SORTracker* obj = new SORTracker();
* obj->add(name,score);
* string param_2 = obj->get();
*/
```

Java Solution:

```
/**
* Problem: Sequentially Ordinal Rank Tracker
* Difficulty: Hard
* Tags: string, graph, sort, queue, heap
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
```

```
*/

class SORTracker {

public SORTracker() {

}

public void add(String name, int score) {

}

public String get() {

}
}

/**
* Your SORTracker object will be instantiated and called as such:
* SORTracker obj = new SORTracker();
* obj.add(name,score);
* String param_2 = obj.get();
*/
```

Python3 Solution:

```
"""
Problem: Sequentially Ordinal Rank Tracker
Difficulty: Hard
Tags: string, graph, sort, queue, heap

Approach: String manipulation with hash map or two pointers
Time Complexity: O(n) or O(n log n)
Space Complexity: O(1) to O(n) depending on approach
"""

class SORTracker:

def __init__(self):
```

```
def add(self, name: str, score: int) -> None:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class SORTracker(object):

def __init__(self):


def add(self, name, score):
"""
:type name: str
:type score: int
:rtype: None
"""


def get(self):
"""
:rtype: str
"""




# Your SORTracker object will be instantiated and called as such:
# obj = SORTracker()
# obj.add(name,score)
# param_2 = obj.get()
```

JavaScript Solution:

```
/**
* Problem: Sequentially Ordinal Rank Tracker
* Difficulty: Hard
* Tags: string, graph, sort, queue, heap
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
```

```
*/


var SORTracker = function() {

};

/**
* @param {string} name
* @param {number} score
* @return {void}
*/
SORTracker.prototype.add = function(name, score) {

};

/**
* @return {string}
*/
SORTracker.prototype.get = function() {

};

/**
* Your SORTracker object will be instantiated and called as such:
* var obj = new SORTracker()
* obj.add(name,score)
* var param_2 = obj.get()
*/
```

TypeScript Solution:

```
/**
* Problem: Sequentially Ordinal Rank Tracker
* Difficulty: Hard
* Tags: string, graph, sort, queue, heap
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/
```

```
class SORTracker {
constructor() {

}

add(name: string, score: number): void {

}

get(): string {

}
}

/**
* Your SORTracker object will be instantiated and called as such:
* var obj = new SORTracker()
* obj.add(name,score)
* var param_2 = obj.get()
*/
```

C# Solution:

```
/*
* Problem: Sequentially Ordinal Rank Tracker
* Difficulty: Hard
* Tags: string, graph, sort, queue, heap
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

public class SORTracker {

public SORTracker() {

}

public void Add(string name, int score) {
```

```
}

public string Get() {

}
}

/**
* Your SORTracker object will be instantiated and called as such:
* SORTracker obj = new SORTracker();
* obj.Add(name,score);
* string param_2 = obj.Get();
*/
```

C Solution:

```
/*
* Problem: Sequentially Ordinal Rank Tracker
* Difficulty: Hard
* Tags: string, graph, sort, queue, heap
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/




typedef struct {

} SORTracker;


SORTracker* sORTrackerCreate() {

}

void sORTrackerAdd(SORTracker* obj, char* name, int score) {
```

```
}

char* sORTrackerGet(SORTracker* obj) {

}

void sORTrackerFree(SORTracker* obj) {

}

/**
* Your SORTracker struct will be instantiated and called as such:
* SORTracker* obj = sORTrackerCreate();
* sORTrackerAdd(obj, name, score);

* char* param_2 = sORTrackerGet(obj);

* sORTrackerFree(obj);
*/
```

Go Solution:

```
// Problem: Sequentially Ordinal Rank Tracker
// Difficulty: Hard
// Tags: string, graph, sort, queue, heap
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

type SORTracker struct {

}


func Constructor() SORTracker {

}


func (this *SORTracker) Add(name string, score int) {
```

```
}


func (this *SORTracker) Get() string {

}


/**
* Your SORTracker object will be instantiated and called as such:
* obj := Constructor();
* obj.Add(name,score);
* param_2 := obj.Get();
*/
```

Kotlin Solution:

```
class SORTracker() {

fun add(name: String, score: Int) {

}

fun get(): String {

}

}

/**
* Your SORTracker object will be instantiated and called as such:
* var obj = SORTracker()
* obj.add(name,score)
* var param_2 = obj.get()
*/
```

Swift Solution:

```
class SORTracker {
```

```
init() {

}

func add(_ name: String, _ score: Int) {

}

func get() -> String {

}
}

/**
* Your SORTracker object will be instantiated and called as such:
* let obj = SORTracker()
* obj.add(name, score)
* let ret_2: String = obj.get()
*/
```

Rust Solution:

```
// Problem: Sequentially Ordinal Rank Tracker
// Difficulty: Hard
// Tags: string, graph, sort, queue, heap
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

struct SORTracker {

}


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

```
fn new() -> Self {

}

fn add(&self, name: String, score: i32) {

}

fn get(&self) -> String {

}
}

/**
* Your SORTracker object will be instantiated and called as such:
* let obj = SORTracker::new();
* obj.add(name, score);
* let ret_2: String = obj.get();
*/
```

Ruby Solution:

```
class SORTracker
def initialize()

end


=begin
:type name: String
:type score: Integer
:rtype: Void
=end
def add(name, score)

end


=begin
:rtype: String
```

```
=end
def get()

end



end

# Your SORTracker object will be instantiated and called as such:
# obj = SORTracker.new()
# obj.add(name, score)
# param_2 = obj.get()
```

PHP Solution:

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

}

/**
* @param String $name
* @param Integer $score
* @return NULL
*/
function add($name, $score) {

}

/**
* @return String
*/
function get() {

}
}

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

```
* $obj = SORTracker();
* $obj->add($name, $score);
* $ret_2 = $obj->get();
*/
```

Dart Solution:

```
class SORTracker {

SORTracker() {

}

void add(String name, int score) {

}

String get() {

}
}

/**
* Your SORTracker object will be instantiated and called as such:
* SORTracker obj = SORTracker();
* obj.add(name,score);
* String param2 = obj.get();
*/
```

Scala Solution:

```
class SORTracker() {

def add(name: String, score: Int): Unit = {

}

def get(): String = {

}
```

```
}

/**
* Your SORTracker object will be instantiated and called as such:
* val obj = new SORTracker()
* obj.add(name,score)
* val param_2 = obj.get()
*/
```

Elixir Solution:

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

end

@spec add(name :: String.t, score :: integer) :: any
def add(name, score) do

end

@spec get() :: String.t
def get() do

end
end

# Your functions will be called as such:
# SORTracker.init_()
# SORTracker.add(name, score)
# param_2 = SORTracker.get()

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

Erlang Solution:

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

```
-spec sor_tracker_add(Name :: unicode:unicode_binary(), Score :: integer())
-> any().
sor_tracker_add(Name, Score) ->
.

-spec sor_tracker_get() -> unicode:unicode_binary().
sor_tracker_get() ->
.


%% Your functions will be called as such:
%% sor_tracker_init_(),
%% sor_tracker_add(Name, Score),
%% Param_2 = sor_tracker_get(),

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

Racket Solution:

```
(define sor-tracker%
(class object%
(super-new)

(init-field)

; add : string? exact-integer? -> void?
(define/public (add name score)
)
; get : -> string?
(define/public (get)
)))

;; Your sor-tracker% object will be instantiated and called as such:
;; (define obj (new sor-tracker%))
;; (send obj add name score)
;; (define param_2 (send obj get))
```