Problem 432: All O`one Data Structure

Problem Information

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

Problem Description

Design a data structure to store the strings' count with the ability to return the strings with minimum and maximum counts.

Implement the

AllOne

class:

AllOne()

Initializes the object of the data structure.

inc(String key)

Increments the count of the string

key

by

1

. If

key

does not exist in the data structure, insert it with count

1

.

dec(String key)

Decrements the count of the string

key

by

1

. If the count of

key

is

0

after the decrement, remove it from the data structure. It is guaranteed that

key

exists in the data structure before the decrement.

getMaxKey()

Returns one of the keys with the maximal count. If no element exists, return an empty string

""

.

getMinKey()

Returns one of the keys with the minimum count. If no element exists, return an empty string

""

.

Note

that each function must run in

O(1)

average time complexity.

Example 1:

Input

["AllOne", "inc", "inc", "getMaxKey", "getMinKey", "inc", "getMaxKey", "getMinKey"] [[], ["hello"], ["hello"], [], [], ["leet"], [], []]

Output

[null, null, null, "hello", "hello", null, "hello", "leet"]

Explanation

AllOne allOne = new AllOne(); allOne.inc("hello"); allOne.inc("hello"); allOne.getMaxKey(); // return "hello" allOne.getMinKey(); // return "hello" allOne.inc("leet"); allOne.getMaxKey(); // return "hello" allOne.getMinKey(); // return "leet"

Constraints:

1 <= key.length <= 10

key

consists of lowercase English letters.

It is guaranteed that for each call to

dec

,

key

is existing in the data structure.

At most

5 * 10

4

calls will be made to

inc

,

dec

,

getMaxKey

, and

getMinKey

.

Code Snippets

C++:

```
class AllOne {
public:
AllOne() {

}

void inc(string key) {

}

void dec(string key) {

}

string getMaxKey() {

}

string getMinKey() {

}
};

/**
* Your AllOne object will be instantiated and called as such:
* AllOne* obj = new AllOne();
* obj->inc(key);
* obj->dec(key);
* string param_3 = obj->getMaxKey();
* string param_4 = obj->getMinKey();
*/
```

Java:

```
class AllOne {

public AllOne() {

}

public void inc(String key) {
```

```
}

public void dec(String key) {

}

public String getMaxKey() {

}

public String getMinKey() {

}
}

/**
* Your AllOne object will be instantiated and called as such:
* AllOne obj = new AllOne();
* obj.inc(key);
* obj.dec(key);
* String param_3 = obj.getMaxKey();
* String param_4 = obj.getMinKey();
*/
```

Python3:

```
class AllOne:

def __init__(self):


def inc(self, key: str) -> None:


def dec(self, key: str) -> None:


def getMaxKey(self) -> str:


def getMinKey(self) -> str:
```

```
# Your AllOne object will be instantiated and called as such:
# obj = AllOne()
# obj.inc(key)
# obj.dec(key)
# param_3 = obj.getMaxKey()
# param_4 = obj.getMinKey()
```

Python:

```
class AllOne(object):

def __init__(self):


def inc(self, key):
"""
:type key: str
:rtype: None
"""


def dec(self, key):
"""
:type key: str
:rtype: None
"""


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


def getMinKey(self):
"""
:rtype: str
"""
```

```
# Your AllOne object will be instantiated and called as such:
# obj = AllOne()
# obj.inc(key)
# obj.dec(key)
# param_3 = obj.getMaxKey()
# param_4 = obj.getMinKey()
```

JavaScript:

```
var AllOne = function() {

};

/**
* @param {string} key
* @return {void}
*/
AllOne.prototype.inc = function(key) {

};

/**
* @param {string} key
* @return {void}
*/
AllOne.prototype.dec = function(key) {

};

/**
* @return {string}
*/
AllOne.prototype.getMaxKey = function() {

};

/**
* @return {string}
*/
```

```
AllOne.prototype.getMinKey = function() {

};

/**
* Your AllOne object will be instantiated and called as such:
* var obj = new AllOne()
* obj.inc(key)
* obj.dec(key)
* var param_3 = obj.getMaxKey()
* var param_4 = obj.getMinKey()
*/
```

TypeScript:

```
class AllOne {
constructor() {

}

inc(key: string): void {

}

dec(key: string): void {

}

getMaxKey(): string {

}

getMinKey(): string {

}
}

/**
* Your AllOne object will be instantiated and called as such:
* var obj = new AllOne()
* obj.inc(key)
* obj.dec(key)
```

```
* var param_3 = obj.getMaxKey()
* var param_4 = obj.getMinKey()
*/
```

C#:

```
public class AllOne {

public AllOne() {

}

public void Inc(string key) {

}

public void Dec(string key) {

}

public string GetMaxKey() {

}

public string GetMinKey() {

}
}

/**
* Your AllOne object will be instantiated and called as such:
* AllOne obj = new AllOne();
* obj.Inc(key);
* obj.Dec(key);
* string param_3 = obj.GetMaxKey();
* string param_4 = obj.GetMinKey();
*/
```

C:

```
```

```
typedef struct {

} AllOne;


AllOne* allOneCreate() {

}

void allOneInc(AllOne* obj, char* key) {

}

void allOneDec(AllOne* obj, char* key) {

}

char* allOneGetMaxKey(AllOne* obj) {

}

char* allOneGetMinKey(AllOne* obj) {

}

void allOneFree(AllOne* obj) {

}

/**
* Your AllOne struct will be instantiated and called as such:
* AllOne* obj = allOneCreate();
* allOneInc(obj, key);

* allOneDec(obj, key);

* char* param_3 = allOneGetMaxKey(obj);

* char* param_4 = allOneGetMinKey(obj);

* allOneFree(obj);
```

```
*/
```

Go:

```
type AllOne struct {

}


func Constructor() AllOne {

}


func (this *AllOne) Inc(key string) {

}


func (this *AllOne) Dec(key string) {

}


func (this *AllOne) GetMaxKey() string {

}


func (this *AllOne) GetMinKey() string {

}


/**
* Your AllOne object will be instantiated and called as such:
* obj := Constructor();
* obj.Inc(key);
* obj.Dec(key);
* param_3 := obj.GetMaxKey();
* param_4 := obj.GetMinKey();
*/
```

Kotlin:

```
class AllOne() {

fun inc(key: String) {

}

fun dec(key: String) {

}

fun getMaxKey(): String {

}

fun getMinKey(): String {

}

}

/**
* Your AllOne object will be instantiated and called as such:
* var obj = AllOne()
* obj.inc(key)
* obj.dec(key)
* var param_3 = obj.getMaxKey()
* var param_4 = obj.getMinKey()
*/
```

Swift:

```


class AllOne {

init() {

}

func inc(_ key: String) {
```

```
}

func dec(_ key: String) {

}

func getMaxKey() -> String {

}

func getMinKey() -> String {

}
}

/**
* Your AllOne object will be instantiated and called as such:
* let obj = AllOne()
* obj.inc(key)
* obj.dec(key)
* let ret_3: String = obj.getMaxKey()
* let ret_4: String = obj.getMinKey()
*/
```

Rust:

```
struct AllOne {

}


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

fn new() -> Self {

}

fn inc(&self, key: String) {
```

```
}

fn dec(&self, key: String) {

}

fn get_max_key(&self) -> String {

}

fn get_min_key(&self) -> String {

}
}

/**
* Your AllOne object will be instantiated and called as such:
* let obj = AllOne::new();
* obj.inc(key);
* obj.dec(key);
* let ret_3: String = obj.get_max_key();
* let ret_4: String = obj.get_min_key();
*/
```

Ruby:

```
class AllOne
def initialize()

end


=begin
:type key: String
:rtype: Void
=end
def inc(key)

end
```

```
=begin
:type key: String
:rtype: Void
=end
def dec(key)

end


=begin
:rtype: String
=end
def get_max_key()

end


=begin
:rtype: String
=end
def get_min_key()

end


end

# Your AllOne object will be instantiated and called as such:
# obj = AllOne.new()
# obj.inc(key)
# obj.dec(key)
# param_3 = obj.get_max_key()
# param_4 = obj.get_min_key()
```

PHP:

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

}
```

```
/**
* @param String $key
* @return NULL
*/
function inc($key) {

}

/**
* @param String $key
* @return NULL
*/
function dec($key) {

}

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

}

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

}
}

/**
* Your AllOne object will be instantiated and called as such:
* $obj = AllOne();
* $obj->inc($key);
* $obj->dec($key);
* $ret_3 = $obj->getMaxKey();
* $ret_4 = $obj->getMinKey();
*/
```

Dart:

```
class AllOne {

AllOne() {

}

void inc(String key) {

}

void dec(String key) {

}

String getMaxKey() {

}

String getMinKey() {

}
}

/**
* Your AllOne object will be instantiated and called as such:
* AllOne obj = AllOne();
* obj.inc(key);
* obj.dec(key);
* String param3 = obj.getMaxKey();
* String param4 = obj.getMinKey();
*/
```

Scala:

```
class AllOne() {

def inc(key: String): Unit = {

}

def dec(key: String): Unit = {
```

```
}

def getMaxKey(): String = {

}

def getMinKey(): String = {

}

}

/**
* Your AllOne object will be instantiated and called as such:
* val obj = new AllOne()
* obj.inc(key)
* obj.dec(key)
* val param_3 = obj.getMaxKey()
* val param_4 = obj.getMinKey()
*/
```

Elixir:

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

end

@spec inc(key :: String.t) :: any
def inc(key) do

end

@spec dec(key :: String.t) :: any
def dec(key) do

end

@spec get_max_key() :: String.t
def get_max_key() do
```

```
end

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

end
end

# Your functions will be called as such:
# AllOne.init_()
# AllOne.inc(key)
# AllOne.dec(key)
# param_3 = AllOne.get_max_key()
# param_4 = AllOne.get_min_key()

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

Erlang:

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

-spec all_one_inc(Key :: unicode:unicode_binary()) -> any().
all_one_inc(Key) ->
.

-spec all_one_dec(Key :: unicode:unicode_binary()) -> any().
all_one_dec(Key) ->
.

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

-spec all_one_get_min_key() -> unicode:unicode_binary().
all_one_get_min_key() ->
.
```

```
%% Your functions will be called as such:
%% all_one_init_(),
%% all_one_inc(Key),
%% all_one_dec(Key),
%% Param_3 = all_one_get_max_key(),
%% Param_4 = all_one_get_min_key(),

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

Racket:

```
(define all-one%
(class object%
(super-new)

(init-field)

; inc : string? -> void?
(define/public (inc key)
)
; dec : string? -> void?
(define/public (dec key)
)
; get-max-key : -> string?
(define/public (get-max-key)
)
; get-min-key : -> string?
(define/public (get-min-key)
)))

;; Your all-one% object will be instantiated and called as such:
;; (define obj (new all-one%))
;; (send obj inc key)
;; (send obj dec key)
;; (define param_3 (send obj get-max-key))
;; (define param_4 (send obj get-min-key))
```

Solutions

C++ Solution:

```
/*
* Problem: All O`one Data Structure
* Difficulty: Hard
* Tags: string, hash, linked_list
*
* 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 AllOne {
public:
AllOne() {

}

void inc(string key) {

}

void dec(string key) {

}

string getMaxKey() {

}

string getMinKey() {

}
};

/**
* Your AllOne object will be instantiated and called as such:
* AllOne* obj = new AllOne();
* obj->inc(key);
* obj->dec(key);
* string param_3 = obj->getMaxKey();
* string param_4 = obj->getMinKey();
*/
```

Java Solution:

```
/**
* Problem: All O`one Data Structure
* Difficulty: Hard
* Tags: string, hash, linked_list
*
* 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 AllOne {

public AllOne() {

}

public void inc(String key) {

}

public void dec(String key) {

}

public String getMaxKey() {

}

public String getMinKey() {

}
}

/**
* Your AllOne object will be instantiated and called as such:
* AllOne obj = new AllOne();
* obj.inc(key);
* obj.dec(key);
* String param_3 = obj.getMaxKey();
* String param_4 = obj.getMinKey();
```

```
*/
```

Python3 Solution:

```
"""
Problem: All O`one Data Structure
Difficulty: Hard
Tags: string, hash, linked_list

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 AllOne:

def __init__(self):


def inc(self, key: str) -> None:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class AllOne(object):

def __init__(self):


def inc(self, key):
"""
:type key: str
:rtype: None
"""


def dec(self, key):
"""
:type key: str
:rtype: None
```

```
"""



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


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




# Your AllOne object will be instantiated and called as such:
# obj = AllOne()
# obj.inc(key)
# obj.dec(key)
# param_3 = obj.getMaxKey()
# param_4 = obj.getMinKey()
```

JavaScript Solution:

```
/**
* Problem: All O`one Data Structure
* Difficulty: Hard
* Tags: string, hash, linked_list
*
* 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 AllOne = function() {

};

/**
```

```
* @param {string} key
* @return {void}
*/
AllOne.prototype.inc = function(key) {

};

/**
* @param {string} key
* @return {void}
*/
AllOne.prototype.dec = function(key) {

};

/**
* @return {string}
*/
AllOne.prototype.getMaxKey = function() {

};

/**
* @return {string}
*/
AllOne.prototype.getMinKey = function() {

};

/**
* Your AllOne object will be instantiated and called as such:
* var obj = new AllOne()
* obj.inc(key)
* obj.dec(key)
* var param_3 = obj.getMaxKey()
* var param_4 = obj.getMinKey()
*/
```

TypeScript Solution:

```
/**
* Problem: All O`one Data Structure
* Difficulty: Hard
* Tags: string, hash, linked_list
*
* 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 AllOne {
constructor() {

}

inc(key: string): void {

}

dec(key: string): void {

}

getMaxKey(): string {

}

getMinKey(): string {

}
}

/**
* Your AllOne object will be instantiated and called as such:
* var obj = new AllOne()
* obj.inc(key)
* obj.dec(key)
* var param_3 = obj.getMaxKey()
* var param_4 = obj.getMinKey()
*/
```

C# Solution:

```
/*
* Problem: All O`one Data Structure
* Difficulty: Hard
* Tags: string, hash, linked_list
*
* 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 AllOne {

public AllOne() {

}

public void Inc(string key) {

}

public void Dec(string key) {

}

public string GetMaxKey() {

}

public string GetMinKey() {

}
}

/**
* Your AllOne object will be instantiated and called as such:
* AllOne obj = new AllOne();
* obj.Inc(key);
* obj.Dec(key);
* string param_3 = obj.GetMaxKey();
* string param_4 = obj.GetMinKey();
*/
```

C Solution:

```
/*
* Problem: All O`one Data Structure
* Difficulty: Hard
* Tags: string, hash, linked_list
*
* 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 {

} AllOne;


AllOne* allOneCreate() {

}

void allOneInc(AllOne* obj, char* key) {

}

void allOneDec(AllOne* obj, char* key) {

}

char* allOneGetMaxKey(AllOne* obj) {

}

char* allOneGetMinKey(AllOne* obj) {

}

void allOneFree(AllOne* obj) {

}
```

```
/**
* Your AllOne struct will be instantiated and called as such:
* AllOne* obj = allOneCreate();
* allOneInc(obj, key);

* allOneDec(obj, key);

* char* param_3 = allOneGetMaxKey(obj);

* char* param_4 = allOneGetMinKey(obj);

* allOneFree(obj);
*/
```

Go Solution:

```
// Problem: All O`one Data Structure
// Difficulty: Hard
// Tags: string, hash, linked_list
//
// 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 AllOne struct {

}


func Constructor() AllOne {

}


func (this *AllOne) Inc(key string) {

}


func (this *AllOne) Dec(key string) {
```

```
}


func (this *AllOne) GetMaxKey() string {

}


func (this *AllOne) GetMinKey() string {

}


/**
* Your AllOne object will be instantiated and called as such:
* obj := Constructor();
* obj.Inc(key);
* obj.Dec(key);
* param_3 := obj.GetMaxKey();
* param_4 := obj.GetMinKey();
*/
```

Kotlin Solution:

```
class AllOne() {

fun inc(key: String) {

}

fun dec(key: String) {

}

fun getMaxKey(): String {

}

fun getMinKey(): String {
```

```
}

}

/**
* Your AllOne object will be instantiated and called as such:
* var obj = AllOne()
* obj.inc(key)
* obj.dec(key)
* var param_3 = obj.getMaxKey()
* var param_4 = obj.getMinKey()
*/
```

Swift Solution:

```
class AllOne {

init() {

}

func inc(_ key: String) {

}

func dec(_ key: String) {

}

func getMaxKey() -> String {

}

func getMinKey() -> String {

}
}

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

```
* let obj = AllOne()
* obj.inc(key)
* obj.dec(key)
* let ret_3: String = obj.getMaxKey()
* let ret_4: String = obj.getMinKey()
*/
```

Rust Solution:

```
// Problem: All O`one Data Structure
// Difficulty: Hard
// Tags: string, hash, linked_list
//
// 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 AllOne {

}


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

fn new() -> Self {

}

fn inc(&self, key: String) {

}

fn dec(&self, key: String) {

}

fn get_max_key(&self) -> String {
```

```
}

fn get_min_key(&self) -> String {

}
}

/**
* Your AllOne object will be instantiated and called as such:
* let obj = AllOne::new();
* obj.inc(key);
* obj.dec(key);
* let ret_3: String = obj.get_max_key();
* let ret_4: String = obj.get_min_key();
*/
```

Ruby Solution:

```
class AllOne
def initialize()

end


=begin
:type key: String
:rtype: Void
=end
def inc(key)

end


=begin
:type key: String
:rtype: Void
=end
def dec(key)

end
```

```
=begin
:rtype: String
=end
def get_max_key()

end


=begin
:rtype: String
=end
def get_min_key()

end


end

# Your AllOne object will be instantiated and called as such:
# obj = AllOne.new()
# obj.inc(key)
# obj.dec(key)
# param_3 = obj.get_max_key()
# param_4 = obj.get_min_key()
```

PHP Solution:

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

}

/**
* @param String $key
* @return NULL
*/
function inc($key) {
```

```
}

/**
* @param String $key
* @return NULL
*/
function dec($key) {

}

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

}

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

}
}

/**
* Your AllOne object will be instantiated and called as such:
* $obj = AllOne();
* $obj->inc($key);
* $obj->dec($key);
* $ret_3 = $obj->getMaxKey();
* $ret_4 = $obj->getMinKey();
*/
```

Dart Solution:

```
class AllOne {

AllOne() {
```

```
}

void inc(String key) {

}

void dec(String key) {

}

String getMaxKey() {

}

String getMinKey() {

}
}

/**
* Your AllOne object will be instantiated and called as such:
* AllOne obj = AllOne();
* obj.inc(key);
* obj.dec(key);
* String param3 = obj.getMaxKey();
* String param4 = obj.getMinKey();
*/
```

Scala Solution:

```
class AllOne() {

def inc(key: String): Unit = {

}

def dec(key: String): Unit = {

}

def getMaxKey(): String = {
```

```
}

def getMinKey(): String = {

}

}

/**
* Your AllOne object will be instantiated and called as such:
* val obj = new AllOne()
* obj.inc(key)
* obj.dec(key)
* val param_3 = obj.getMaxKey()
* val param_4 = obj.getMinKey()
*/
```

Elixir Solution:

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

end

@spec inc(key :: String.t) :: any
def inc(key) do

end

@spec dec(key :: String.t) :: any
def dec(key) do

end

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

end
```

```
@spec get_min_key() :: String.t
def get_min_key() do

end
end

# Your functions will be called as such:
# AllOne.init_()
# AllOne.inc(key)
# AllOne.dec(key)
# param_3 = AllOne.get_max_key()
# param_4 = AllOne.get_min_key()

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

Erlang Solution:

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

-spec all_one_inc(Key :: unicode:unicode_binary()) -> any().
all_one_inc(Key) ->
.

-spec all_one_dec(Key :: unicode:unicode_binary()) -> any().
all_one_dec(Key) ->
.

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

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


%% Your functions will be called as such:
%% all_one_init_(),
```

```
%% all_one_inc(Key),
%% all_one_dec(Key),
%% Param_3 = all_one_get_max_key(),
%% Param_4 = all_one_get_min_key(),

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

Racket Solution:

```
(define all-one%
(class object%
(super-new)

(init-field)

; inc : string? -> void?
(define/public (inc key)
)
; dec : string? -> void?
(define/public (dec key)
)
; get-max-key : -> string?
(define/public (get-max-key)
)
; get-min-key : -> string?
(define/public (get-min-key)
)))

;; Your all-one% object will be instantiated and called as such:
;; (define obj (new all-one%))
;; (send obj inc key)
;; (send obj dec key)
;; (define param_3 (send obj get-max-key))
;; (define param_4 (send obj get-min-key))
```