Problem 1146: Snapshot Array

Problem Information

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

Problem Description

Implement a SnapshotArray that supports the following interface:

SnapshotArray(int length)

initializes an array-like data structure with the given length.

Initially, each element equals 0

.

void set(index, val)

sets the element at the given

index

to be equal to

val

.

int snap()

takes a snapshot of the array and returns the

snap_id

: the total number of times we called

snap()

minus

1

.

int get(index, snap_id)

returns the value at the given

index

, at the time we took the snapshot with the given

snap_id

Example 1:

Input:

["SnapshotArray","set","snap","set","get"] [[3],[0,5],[],[0,6],[0,0]]

Output:

[null,null,0,null,5]

Explanation:

SnapshotArray snapshotArr = new SnapshotArray(3); // set the length to be 3 snapshotArr.set(0,5); // Set array[0] = 5 snapshotArr.snap(); // Take a snapshot, return snap_id = 0 snapshotArr.set(0,6); snapshotArr.get(0,0); // Get the value of array[0] with snap_id = 0, return 5

Constraints:

1 <= length <= 5 * 10

4

0 <= index < length

0 <= val <= 10

9

0 <= snap_id <

(the total number of times we call

snap()

)

At most

5 * 10

4

calls will be made to

set

,

snap

, and

get

.

Code Snippets

C++:

```
class SnapshotArray {
public:
SnapshotArray(int length) {

}

void set(int index, int val) {

}

int snap() {

}

int get(int index, int snap_id) {

}
};

/**
* Your SnapshotArray object will be instantiated and called as such:
* SnapshotArray* obj = new SnapshotArray(length);
* obj->set(index,val);
* int param_2 = obj->snap();
* int param_3 = obj->get(index,snap_id);
*/
```

Java:

```
class SnapshotArray {

public SnapshotArray(int length) {

}

public void set(int index, int val) {
```

```
}

public int snap() {

}

public int get(int index, int snap_id) {

}
}

/**
* Your SnapshotArray object will be instantiated and called as such:
* SnapshotArray obj = new SnapshotArray(length);
* obj.set(index,val);
* int param_2 = obj.snap();
* int param_3 = obj.get(index,snap_id);
*/
```

Python3:

```
class SnapshotArray:

def __init__(self, length: int):


def set(self, index: int, val: int) -> None:


def snap(self) -> int:


def get(self, index: int, snap_id: int) -> int:




# Your SnapshotArray object will be instantiated and called as such:
# obj = SnapshotArray(length)
# obj.set(index,val)
# param_2 = obj.snap()
# param_3 = obj.get(index,snap_id)
```

Python:

```
class SnapshotArray(object):

def __init__(self, length):
"""
:type length: int
"""



def set(self, index, val):
"""
:type index: int
:type val: int
:rtype: None
"""



def snap(self):
"""
:rtype: int
"""



def get(self, index, snap_id):
"""
:type index: int
:type snap_id: int
:rtype: int
"""




# Your SnapshotArray object will be instantiated and called as such:
# obj = SnapshotArray(length)
# obj.set(index,val)
# param_2 = obj.snap()
# param_3 = obj.get(index,snap_id)
```

JavaScript:

```
/**
* @param {number} length
*/
var SnapshotArray = function(length) {

};

/**
* @param {number} index
* @param {number} val
* @return {void}
*/
SnapshotArray.prototype.set = function(index, val) {

};

/**
* @return {number}
*/
SnapshotArray.prototype.snap = function() {

};

/**
* @param {number} index
* @param {number} snap_id
* @return {number}
*/
SnapshotArray.prototype.get = function(index, snap_id) {

};

/**
* Your SnapshotArray object will be instantiated and called as such:
* var obj = new SnapshotArray(length)
* obj.set(index,val)
* var param_2 = obj.snap()
* var param_3 = obj.get(index,snap_id)
*/
```

TypeScript:

```
class SnapshotArray {
constructor(length: number) {

}

set(index: number, val: number): void {

}

snap(): number {

}

get(index: number, snap_id: number): number {

}
}

/**
* Your SnapshotArray object will be instantiated and called as such:
* var obj = new SnapshotArray(length)
* obj.set(index,val)
* var param_2 = obj.snap()
* var param_3 = obj.get(index,snap_id)
*/
```

C#:

```
public class SnapshotArray {

public SnapshotArray(int length) {

}

public void Set(int index, int val) {

}

public int Snap() {

}

public int Get(int index, int snap_id) {
```

```
}
}

/**
* Your SnapshotArray object will be instantiated and called as such:
* SnapshotArray obj = new SnapshotArray(length);
* obj.Set(index,val);
* int param_2 = obj.Snap();
* int param_3 = obj.Get(index,snap_id);
*/
```

C:

```
typedef struct {

} SnapshotArray;


SnapshotArray* snapshotArrayCreate(int length) {

}

void snapshotArraySet(SnapshotArray* obj, int index, int val) {

}

int snapshotArraySnap(SnapshotArray* obj) {

}

int snapshotArrayGet(SnapshotArray* obj, int index, int snap_id) {

}

void snapshotArrayFree(SnapshotArray* obj) {

}
```

```
/**
* Your SnapshotArray struct will be instantiated and called as such:
* SnapshotArray* obj = snapshotArrayCreate(length);
* snapshotArraySet(obj, index, val);

* int param_2 = snapshotArraySnap(obj);

* int param_3 = snapshotArrayGet(obj, index, snap_id);

* snapshotArrayFree(obj);
*/
```

Go:

```
type SnapshotArray struct {

}


func Constructor(length int) SnapshotArray {

}


func (this *SnapshotArray) Set(index int, val int) {

}


func (this *SnapshotArray) Snap() int {

}


func (this *SnapshotArray) Get(index int, snap_id int) int {

}


/**
* Your SnapshotArray object will be instantiated and called as such:
* obj := Constructor(length);
```

```
* obj.Set(index,val);
* param_2 := obj.Snap();
* param_3 := obj.Get(index,snap_id);
*/
```

Kotlin:

```
class SnapshotArray(length: Int) {

fun set(index: Int, `val`: Int) {

}

fun snap(): Int {

}

fun get(index: Int, snap_id: Int): Int {

}

}

/**
* Your SnapshotArray object will be instantiated and called as such:
* var obj = SnapshotArray(length)
* obj.set(index,`val`)
* var param_2 = obj.snap()
* var param_3 = obj.get(index,snap_id)
*/
```

Swift:

```
class SnapshotArray {

init(_ length: Int) {

}

func set(_ index: Int, _ val: Int) {
```

```
}

func snap() -> Int {

}

func get(_ index: Int, _ snap_id: Int) -> Int {

}
}

/**
* Your SnapshotArray object will be instantiated and called as such:
* let obj = SnapshotArray(length)
* obj.set(index, val)
* let ret_2: Int = obj.snap()
* let ret_3: Int = obj.get(index, snap_id)
*/
```

Rust:

```
struct SnapshotArray {

}


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

fn new(length: i32) -> Self {

}

fn set(&self, index: i32, val: i32) {

}

fn snap(&self) -> i32 {
```

```
}

fn get(&self, index: i32, snap_id: i32) -> i32 {

}
}

/**
* Your SnapshotArray object will be instantiated and called as such:
* let obj = SnapshotArray::new(length);
* obj.set(index, val);
* let ret_2: i32 = obj.snap();
* let ret_3: i32 = obj.get(index, snap_id);
*/
```

Ruby:

```
class SnapshotArray

=begin
:type length: Integer
=end
def initialize(length)

end


=begin
:type index: Integer
:type val: Integer
:rtype: Void
=end
def set(index, val)

end


=begin
:rtype: Integer
=end
def snap()
```

```
end


=begin
:type index: Integer
:type snap_id: Integer
:rtype: Integer
=end
def get(index, snap_id)

end



end

# Your SnapshotArray object will be instantiated and called as such:
# obj = SnapshotArray.new(length)
# obj.set(index, val)
# param_2 = obj.snap()
# param_3 = obj.get(index, snap_id)
```

PHP:

```
class SnapshotArray {
/**
* @param Integer $length
*/
function __construct($length) {

}

/**
* @param Integer $index
* @param Integer $val
* @return NULL
*/
function set($index, $val) {

}

/**
* @return Integer
```

```
*/
function snap() {

}

/**
* @param Integer $index
* @param Integer $snap_id
* @return Integer
*/
function get($index, $snap_id) {

}
}

/**
* Your SnapshotArray object will be instantiated and called as such:
* $obj = SnapshotArray($length);
* $obj->set($index, $val);
* $ret_2 = $obj->snap();
* $ret_3 = $obj->get($index, $snap_id);
*/
```

Dart:

```
class SnapshotArray {

SnapshotArray(int length) {

}

void set(int index, int val) {

}

int snap() {

}

int get(int index, int snap_id) {

}
```

```
}

/**
* Your SnapshotArray object will be instantiated and called as such:
* SnapshotArray obj = SnapshotArray(length);
* obj.set(index,val);
* int param2 = obj.snap();
* int param3 = obj.get(index,snap_id);
*/
```

Scala:

```
class SnapshotArray(_length: Int) {

def set(index: Int, `val`: Int): Unit = {

}

def snap(): Int = {

}

def get(index: Int, snap_id: Int): Int = {

}

}

/**
* Your SnapshotArray object will be instantiated and called as such:
* val obj = new SnapshotArray(length)
* obj.set(index,`val`)
* val param_2 = obj.snap()
* val param_3 = obj.get(index,snap_id)
*/
```

Elixir:

```
defmodule SnapshotArray do
@spec init_(length :: integer) :: any
def init_(length) do
```

```
end

@spec set(index :: integer, val :: integer) :: any
def set(index, val) do

end

@spec snap() :: integer
def snap() do

end

@spec get(index :: integer, snap_id :: integer) :: integer
def get(index, snap_id) do

end
end

# Your functions will be called as such:
# SnapshotArray.init_(length)
# SnapshotArray.set(index, val)
# param_2 = SnapshotArray.snap()
# param_3 = SnapshotArray.get(index, snap_id)

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

Erlang:

```
-spec snapshot_array_init_(Length :: integer()) -> any().
snapshot_array_init_(Length) ->
.

-spec snapshot_array_set(Index :: integer(), Val :: integer()) -> any().
snapshot_array_set(Index, Val) ->
.

-spec snapshot_array_snap() -> integer().
snapshot_array_snap() ->
.

-spec snapshot_array_get(Index :: integer(), Snap_id :: integer()) ->
```

```
integer().
snapshot_array_get(Index, Snap_id) ->
.


%% Your functions will be called as such:
%% snapshot_array_init_(Length),
%% snapshot_array_set(Index, Val),
%% Param_2 = snapshot_array_snap(),
%% Param_3 = snapshot_array_get(Index, Snap_id),

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

Racket:

```
(define snapshot-array%
(class object%
(super-new)

; length : exact-integer?
(init-field
length)

; set : exact-integer? exact-integer? -> void?
(define/public (set index val)
)
; snap : -> exact-integer?
(define/public (snap)
)
; get : exact-integer? exact-integer? -> exact-integer?
(define/public (get index snap_id)
)))

;; Your snapshot-array% object will be instantiated and called as such:
;; (define obj (new snapshot-array% [length length]))
;; (send obj set index val)
;; (define param_2 (send obj snap))
;; (define param_3 (send obj get index snap_id))
```

Solutions

C++ Solution:

```
/*
* Problem: Snapshot Array
* Difficulty: Medium
* Tags: array, hash, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class SnapshotArray {
public:
SnapshotArray(int length) {

}

void set(int index, int val) {

}

int snap() {

}

int get(int index, int snap_id) {

}
};

/**
* Your SnapshotArray object will be instantiated and called as such:
* SnapshotArray* obj = new SnapshotArray(length);
* obj->set(index,val);
* int param_2 = obj->snap();
* int param_3 = obj->get(index,snap_id);
*/
```

Java Solution:

```
/**
* Problem: Snapshot Array
```

```
* Difficulty: Medium
* Tags: array, hash, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class SnapshotArray {

public SnapshotArray(int length) {

}

public void set(int index, int val) {

}

public int snap() {

}

public int get(int index, int snap_id) {

}
}

/**
* Your SnapshotArray object will be instantiated and called as such:
* SnapshotArray obj = new SnapshotArray(length);
* obj.set(index,val);
* int param_2 = obj.snap();
* int param_3 = obj.get(index,snap_id);
*/
```

Python3 Solution:

```
"""
Problem: Snapshot Array
Difficulty: Medium
Tags: array, hash, search
```

```
Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) for hash map
"""

class SnapshotArray:

def __init__(self, length: int):


def set(self, index: int, val: int) -> None:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class SnapshotArray(object):

def __init__(self, length):
"""
:type length: int
"""



def set(self, index, val):
"""
:type index: int
:type val: int
:rtype: None
"""



def snap(self):
"""
:rtype: int
"""



def get(self, index, snap_id):
"""
```

```
:type index: int
:type snap_id: int
:rtype: int
"""



# Your SnapshotArray object will be instantiated and called as such:
# obj = SnapshotArray(length)
# obj.set(index,val)
# param_2 = obj.snap()
# param_3 = obj.get(index,snap_id)
```

JavaScript Solution:

```
/**
* Problem: Snapshot Array
* Difficulty: Medium
* Tags: array, hash, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

/**
* @param {number} length
*/
var SnapshotArray = function(length) {

};

/**
* @param {number} index
* @param {number} val
* @return {void}
*/
SnapshotArray.prototype.set = function(index, val) {

};
```

```
/**
* @return {number}
*/
SnapshotArray.prototype.snap = function() {

};

/**
* @param {number} index
* @param {number} snap_id
* @return {number}
*/
SnapshotArray.prototype.get = function(index, snap_id) {

};

/**
* Your SnapshotArray object will be instantiated and called as such:
* var obj = new SnapshotArray(length)
* obj.set(index,val)
* var param_2 = obj.snap()
* var param_3 = obj.get(index,snap_id)
*/
```

TypeScript Solution:

```
/**
* Problem: Snapshot Array
* Difficulty: Medium
* Tags: array, hash, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class SnapshotArray {
constructor(length: number) {

}
```

```
set(index: number, val: number): void {

}

snap(): number {

}

get(index: number, snap_id: number): number {

}
}

/**
* Your SnapshotArray object will be instantiated and called as such:
* var obj = new SnapshotArray(length)
* obj.set(index,val)
* var param_2 = obj.snap()
* var param_3 = obj.get(index,snap_id)
*/
```

C# Solution:

```
/*
* Problem: Snapshot Array
* Difficulty: Medium
* Tags: array, hash, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

public class SnapshotArray {

public SnapshotArray(int length) {

}

public void Set(int index, int val) {
```

```
}

public int Snap() {

}

public int Get(int index, int snap_id) {

}
}

/**
* Your SnapshotArray object will be instantiated and called as such:
* SnapshotArray obj = new SnapshotArray(length);
* obj.Set(index,val);
* int param_2 = obj.Snap();
* int param_3 = obj.Get(index,snap_id);
*/
```

C Solution:

```
/*
* Problem: Snapshot Array
* Difficulty: Medium
* Tags: array, hash, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/




typedef struct {

} SnapshotArray;


SnapshotArray* snapshotArrayCreate(int length) {
```

```
}

void snapshotArraySet(SnapshotArray* obj, int index, int val) {

}

int snapshotArraySnap(SnapshotArray* obj) {

}

int snapshotArrayGet(SnapshotArray* obj, int index, int snap_id) {

}

void snapshotArrayFree(SnapshotArray* obj) {

}

/**
* Your SnapshotArray struct will be instantiated and called as such:
* SnapshotArray* obj = snapshotArrayCreate(length);
* snapshotArraySet(obj, index, val);

* int param_2 = snapshotArraySnap(obj);

* int param_3 = snapshotArrayGet(obj, index, snap_id);

* snapshotArrayFree(obj);
*/
```

Go Solution:

```
// Problem: Snapshot Array
// Difficulty: Medium
// Tags: array, hash, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

type SnapshotArray struct {
```

```
}


func Constructor(length int) SnapshotArray {

}


func (this *SnapshotArray) Set(index int, val int) {

}


func (this *SnapshotArray) Snap() int {

}


func (this *SnapshotArray) Get(index int, snap_id int) int {

}


/**
* Your SnapshotArray object will be instantiated and called as such:
* obj := Constructor(length);
* obj.Set(index,val);
* param_2 := obj.Snap();
* param_3 := obj.Get(index,snap_id);
*/
```

Kotlin Solution:

```
class SnapshotArray(length: Int) {

fun set(index: Int, `val`: Int) {

}

fun snap(): Int {
```

```
}

fun get(index: Int, snap_id: Int): Int {

}

}

/**
* Your SnapshotArray object will be instantiated and called as such:
* var obj = SnapshotArray(length)
* obj.set(index,`val`)
* var param_2 = obj.snap()
* var param_3 = obj.get(index,snap_id)
*/
```

Swift Solution:

```
class SnapshotArray {

init(_ length: Int) {

}

func set(_ index: Int, _ val: Int) {

}

func snap() -> Int {

}

func get(_ index: Int, _ snap_id: Int) -> Int {

}
}

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

```
* let obj = SnapshotArray(length)
* obj.set(index, val)
* let ret_2: Int = obj.snap()
* let ret_3: Int = obj.get(index, snap_id)
*/
```

Rust Solution:

```
// Problem: Snapshot Array
// Difficulty: Medium
// Tags: array, hash, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

struct SnapshotArray {

}


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

fn new(length: i32) -> Self {

}

fn set(&self, index: i32, val: i32) {

}

fn snap(&self) -> i32 {

}

fn get(&self, index: i32, snap_id: i32) -> i32 {
```

```
}
}

/**
* Your SnapshotArray object will be instantiated and called as such:
* let obj = SnapshotArray::new(length);
* obj.set(index, val);
* let ret_2: i32 = obj.snap();
* let ret_3: i32 = obj.get(index, snap_id);
*/
```

Ruby Solution:

```
class SnapshotArray

=begin
:type length: Integer
=end
def initialize(length)

end


=begin
:type index: Integer
:type val: Integer
:rtype: Void
=end
def set(index, val)

end


=begin
:rtype: Integer
=end
def snap()

end
```

```
=begin
:type index: Integer
:type snap_id: Integer
:rtype: Integer
=end
def get(index, snap_id)

end



end

# Your SnapshotArray object will be instantiated and called as such:
# obj = SnapshotArray.new(length)
# obj.set(index, val)
# param_2 = obj.snap()
# param_3 = obj.get(index, snap_id)
```

PHP Solution:

```
class SnapshotArray {
/**
* @param Integer $length
*/
function __construct($length) {

}

/**
* @param Integer $index
* @param Integer $val
* @return NULL
*/
function set($index, $val) {

}

/**
* @return Integer
*/
function snap() {
```

```
}

/**
* @param Integer $index
* @param Integer $snap_id
* @return Integer
*/
function get($index, $snap_id) {

}
}

/**
* Your SnapshotArray object will be instantiated and called as such:
* $obj = SnapshotArray($length);
* $obj->set($index, $val);
* $ret_2 = $obj->snap();
* $ret_3 = $obj->get($index, $snap_id);
*/
```

Dart Solution:

```
class SnapshotArray {

SnapshotArray(int length) {

}

void set(int index, int val) {

}

int snap() {

}

int get(int index, int snap_id) {

}
}
```

```
/**
* Your SnapshotArray object will be instantiated and called as such:
* SnapshotArray obj = SnapshotArray(length);
* obj.set(index,val);
* int param2 = obj.snap();
* int param3 = obj.get(index,snap_id);
*/
```

Scala Solution:

```
class SnapshotArray(_length: Int) {

def set(index: Int, `val`: Int): Unit = {

}

def snap(): Int = {

}

def get(index: Int, snap_id: Int): Int = {

}

}

/**
* Your SnapshotArray object will be instantiated and called as such:
* val obj = new SnapshotArray(length)
* obj.set(index,`val`)
* val param_2 = obj.snap()
* val param_3 = obj.get(index,snap_id)
*/
```

Elixir Solution:

```
defmodule SnapshotArray do
@spec init_(length :: integer) :: any
def init_(length) do
```

```
end

@spec set(index :: integer, val :: integer) :: any
def set(index, val) do

end

@spec snap() :: integer
def snap() do

end

@spec get(index :: integer, snap_id :: integer) :: integer
def get(index, snap_id) do

end
end

# Your functions will be called as such:
# SnapshotArray.init_(length)
# SnapshotArray.set(index, val)
# param_2 = SnapshotArray.snap()
# param_3 = SnapshotArray.get(index, snap_id)

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

Erlang Solution:

```
-spec snapshot_array_init_(Length :: integer()) -> any().
snapshot_array_init_(Length) ->
.

-spec snapshot_array_set(Index :: integer(), Val :: integer()) -> any().
snapshot_array_set(Index, Val) ->
.

-spec snapshot_array_snap() -> integer().
snapshot_array_snap() ->
.
```

```
-spec snapshot_array_get(Index :: integer(), Snap_id :: integer()) ->
integer().
snapshot_array_get(Index, Snap_id) ->
.


%% Your functions will be called as such:
%% snapshot_array_init_(Length),
%% snapshot_array_set(Index, Val),
%% Param_2 = snapshot_array_snap(),
%% Param_3 = snapshot_array_get(Index, Snap_id),

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

Racket Solution:

```
(define snapshot-array%
(class object%
(super-new)

; length : exact-integer?
(init-field
length)

; set : exact-integer? exact-integer? -> void?
(define/public (set index val)
)
; snap : -> exact-integer?
(define/public (snap)
)
; get : exact-integer? exact-integer? -> exact-integer?
(define/public (get index snap_id)
)))

;; Your snapshot-array% object will be instantiated and called as such:
;; (define obj (new snapshot-array% [length length]))
;; (send obj set index val)
;; (define param_2 (send obj snap))
;; (define param_3 (send obj get index snap_id))
```