Problem 251: Flatten 2D Vector

Problem Information

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

Problem Description

Design an iterator to flatten a 2D vector. It should support the

next

and

hasNext

operations.

Implement the

Vector2D

class:

Vector2D(int[][] vec)

initializes the object with the 2D vector

vec

.

next()

returns the next element from the 2D vector and moves the pointer one step forward. You may assume that all the calls to

next

are valid.

hasNext()

returns

true

if there are still some elements in the vector, and

false

otherwise.

Example 1:

Input

["Vector2D", "next", "next", "next", "hasNext", "hasNext", "next", "hasNext"] [[[[1, 2], [3], [4]]], [], [], [], [], [], [], []]

Output

[null, 1, 2, 3, true, true, 4, false]

Explanation

Vector2D vector2D = new Vector2D([[1, 2], [3], [4]]); vector2D.next(); // return 1 vector2D.next(); // return 2 vector2D.next(); // return 3 vector2D.hasNext(); // return True vector2D.hasNext(); // return True vector2D.next(); // return 4 vector2D.hasNext(); // return False

Constraints:

0 <= vec.length <= 200

0 <= vec[i].length <= 500

-500 <= vec[i][j] <= 500

At most

10

5

calls will be made to

next

and

hasNext

.

Follow up:

As an added challenge, try to code it using only

iterators in C++

or

iterators in Java

.

Code Snippets

C++:

```
class Vector2D {
public:
Vector2D(vector<vector<int>>& vec) {

}

int next() {

}

bool hasNext() {

}
};

/**
* Your Vector2D object will be instantiated and called as such:
* Vector2D* obj = new Vector2D(vec);
* int param_1 = obj->next();
* bool param_2 = obj->hasNext();
*/
```

Java:

```
class Vector2D {

public Vector2D(int[][] vec) {

}

public int next() {

}

public boolean hasNext() {

}
}

/**
* Your Vector2D object will be instantiated and called as such:
* Vector2D obj = new Vector2D(vec);
* int param_1 = obj.next();
```

```
* boolean param_2 = obj.hasNext();
*/
```

Python3:

```
class Vector2D:

def __init__(self, vec: List[List[int]]):


def next(self) -> int:


def hasNext(self) -> bool:




# Your Vector2D object will be instantiated and called as such:
# obj = Vector2D(vec)
# param_1 = obj.next()
# param_2 = obj.hasNext()
```

Python:

```
class Vector2D(object):

def __init__(self, vec):
"""
:type vec: List[List[int]]
"""



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



def hasNext(self):
"""
:rtype: bool
"""
```

```
# Your Vector2D object will be instantiated and called as such:
# obj = Vector2D(vec)
# param_1 = obj.next()
# param_2 = obj.hasNext()
```

JavaScript:

```
/**
* @param {number[][]} vec
*/
var Vector2D = function(vec) {

};

/**
* @return {number}
*/
Vector2D.prototype.next = function() {

};

/**
* @return {boolean}
*/
Vector2D.prototype.hasNext = function() {

};

/**
* Your Vector2D object will be instantiated and called as such:
* var obj = new Vector2D(vec)
* var param_1 = obj.next()
* var param_2 = obj.hasNext()
*/
```

TypeScript:

```
class Vector2D {
constructor(vec: number[][]) {
```

```
}

next(): number {

}

hasNext(): boolean {

}
}

/**
* Your Vector2D object will be instantiated and called as such:
* var obj = new Vector2D(vec)
* var param_1 = obj.next()
* var param_2 = obj.hasNext()
*/
```

C#:

```
public class Vector2D {

public Vector2D(int[][] vec) {

}

public int Next() {

}

public bool HasNext() {

}
}

/**
* Your Vector2D object will be instantiated and called as such:
* Vector2D obj = new Vector2D(vec);
* int param_1 = obj.Next();
* bool param_2 = obj.HasNext();
*/
```

C:

```
typedef struct {

} Vector2D;


Vector2D* vector2DCreate(int** vec, int vecSize, int* vecColSize) {

}

int vector2DNext(Vector2D* obj) {

}

bool vector2DHasNext(Vector2D* obj) {

}

void vector2DFree(Vector2D* obj) {

}

/**
* Your Vector2D struct will be instantiated and called as such:
* Vector2D* obj = vector2DCreate(vec, vecSize, vecColSize);
* int param_1 = vector2DNext(obj);

* bool param_2 = vector2DHasNext(obj);

* vector2DFree(obj);
*/
```

Go:

```
type Vector2D struct {

}
```

```
func Constructor(vec [][]int) Vector2D {

}


func (this *Vector2D) Next() int {

}


func (this *Vector2D) HasNext() bool {

}


/**
* Your Vector2D object will be instantiated and called as such:
* obj := Constructor(vec);
* param_1 := obj.Next();
* param_2 := obj.HasNext();
*/
```

Kotlin:

```
class Vector2D(vec: Array<IntArray>) {

fun next(): Int {

}

fun hasNext(): Boolean {

}

}

/**
* Your Vector2D object will be instantiated and called as such:
* var obj = Vector2D(vec)
* var param_1 = obj.next()
* var param_2 = obj.hasNext()
```

```
*/
```

Swift:

```
class Vector2D {

init(_ vec: [[Int]]) {

}

func next() -> Int {

}

func hasNext() -> Bool {

}
}

/**
* Your Vector2D object will be instantiated and called as such:
* let obj = Vector2D(vec)
* let ret_1: Int = obj.next()
* let ret_2: Bool = obj.hasNext()
*/
```

Rust:

```
struct Vector2D {

}


/**
* `&self` means the method takes an immutable reference.
* If you need a mutable reference, change it to `&mut self` instead.
*/
impl Vector2D {

fn new(vec: Vec<Vec<i32>>) -> Self {
```

```
}

fn next(&self) -> i32 {

}

fn has_next(&self) -> bool {

}
}

/**
* Your Vector2D object will be instantiated and called as such:
* let obj = Vector2D::new(vec);
* let ret_1: i32 = obj.next();
* let ret_2: bool = obj.has_next();
*/
```

Ruby:

```
class Vector2D

=begin
:type vec: Integer[][]
=end
def initialize(vec)

end


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

end


=begin
:rtype: Boolean
=end
def has_next()
```

```
end


end

# Your Vector2D object will be instantiated and called as such:
# obj = Vector2D.new(vec)
# param_1 = obj.next()
# param_2 = obj.has_next()
```

PHP:

```
class Vector2D {
/**
* @param Integer[][] $vec
*/
function __construct($vec) {

}

/**
* @return Integer
*/
function next() {

}

/**
* @return Boolean
*/
function hasNext() {

}
}

/**
* Your Vector2D object will be instantiated and called as such:
* $obj = Vector2D($vec);
* $ret_1 = $obj->next();
* $ret_2 = $obj->hasNext();
*/
```

Dart:

```
class Vector2D {

Vector2D(List<List<int>> vec) {

}

int next() {

}

bool hasNext() {

}
}

/**
* Your Vector2D object will be instantiated and called as such:
* Vector2D obj = Vector2D(vec);
* int param1 = obj.next();
* bool param2 = obj.hasNext();
*/
```

Scala:

```
class Vector2D(_vec: Array[Array[Int]]) {

def next(): Int = {

}

def hasNext(): Boolean = {

}

}

/**
* Your Vector2D object will be instantiated and called as such:
* val obj = new Vector2D(vec)
* val param_1 = obj.next()
```

```
* val param_2 = obj.hasNext()
*/
```

Elixir:

```
defmodule Vector2D do
@spec init_(vec :: [[integer]]) :: any
def init_(vec) do

end

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

end

@spec has_next() :: boolean
def has_next() do

end
end

# Your functions will be called as such:
# Vector2D.init_(vec)
# param_1 = Vector2D.next()
# param_2 = Vector2D.has_next()

# Vector2D.init_ will be called before every test case, in which you can do
some necessary initializations.
```

Erlang:

```
-spec vector2_d_init_(Vec :: [[integer()]]) -> any().
vector2_d_init_(Vec) ->
.

-spec vector2_d_next() -> integer().
vector2_d_next() ->
.

-spec vector2_d_has_next() -> boolean().
vector2_d_has_next() ->
```

```
.


%% Your functions will be called as such:
%% vector2_d_init_(Vec),
%% Param_1 = vector2_d_next(),
%% Param_2 = vector2_d_has_next(),

%% vector2_d_init_ will be called before every test case, in which you can do some necessary initializations.
```

Racket:

```
(define vector2-d%
(class object%
(super-new)

; vec : (listof (listof exact-integer?))
(init-field
vec)

; next : -> exact-integer?
(define/public (next)
)
; has-next : -> boolean?
(define/public (has-next)
)))

;; Your vector2-d% object will be instantiated and called as such:
;; (define obj (new vector2-d% [vec vec]))
;; (define param_1 (send obj next))
;; (define param_2 (send obj has-next))
```

Solutions

C++ Solution:

```
/*
* Problem: Flatten 2D Vector
* Difficulty: Medium
```

```
* Tags: array
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Vector2D {
public:
Vector2D(vector<vector<int>>& vec) {

}

int next() {

}

bool hasNext() {

}
};

/**
* Your Vector2D object will be instantiated and called as such:
* Vector2D* obj = new Vector2D(vec);
* int param_1 = obj->next();
* bool param_2 = obj->hasNext();
*/
```

Java Solution:

```
/**
* Problem: Flatten 2D Vector
* Difficulty: Medium
* Tags: array
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/
```

```
class Vector2D {

public Vector2D(int[][] vec) {

}

public int next() {

}

public boolean hasNext() {

}
}

/**
* Your Vector2D object will be instantiated and called as such:
* Vector2D obj = new Vector2D(vec);
* int param_1 = obj.next();
* boolean param_2 = obj.hasNext();
*/
```

Python3 Solution:

```
"""
Problem: Flatten 2D Vector
Difficulty: Medium
Tags: array

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(1) to O(n) depending on approach
"""

class Vector2D:

def __init__(self, vec: List[List[int]]):


def next(self) -> int:
# TODO: Implement optimized solution
```

```
pass
```

Python Solution:

```
class Vector2D(object):

def __init__(self, vec):
"""
:type vec: List[List[int]]
"""



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



def hasNext(self):
"""
:rtype: bool
"""




# Your Vector2D object will be instantiated and called as such:
# obj = Vector2D(vec)
# param_1 = obj.next()
# param_2 = obj.hasNext()
```

JavaScript Solution:

```
/**
* Problem: Flatten 2D Vector
* Difficulty: Medium
* Tags: array
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/
```

```
/**
* @param {number[][]} vec
*/
var Vector2D = function(vec) {

};

/**
* @return {number}
*/
Vector2D.prototype.next = function() {

};

/**
* @return {boolean}
*/
Vector2D.prototype.hasNext = function() {

};

/**
* Your Vector2D object will be instantiated and called as such:
* var obj = new Vector2D(vec)
* var param_1 = obj.next()
* var param_2 = obj.hasNext()
*/
```

TypeScript Solution:

```
/**
* Problem: Flatten 2D Vector
* Difficulty: Medium
* Tags: array
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/
```

```
class Vector2D {
constructor(vec: number[][]) {

}

next(): number {

}

hasNext(): boolean {

}
}

/**
* Your Vector2D object will be instantiated and called as such:
* var obj = new Vector2D(vec)
* var param_1 = obj.next()
* var param_2 = obj.hasNext()
*/
```

C# Solution:

```
/*
* Problem: Flatten 2D Vector
* Difficulty: Medium
* Tags: array
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

public class Vector2D {

public Vector2D(int[][] vec) {

}

public int Next() {
```

```
}

public bool HasNext() {

}
}

/**
* Your Vector2D object will be instantiated and called as such:
* Vector2D obj = new Vector2D(vec);
* int param_1 = obj.Next();
* bool param_2 = obj.HasNext();
*/
```

C Solution:

```
/*
* Problem: Flatten 2D Vector
* Difficulty: Medium
* Tags: array
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/




typedef struct {

} Vector2D;


Vector2D* vector2DCreate(int** vec, int vecSize, int* vecColSize) {

}

int vector2DNext(Vector2D* obj) {

}
```

```
bool vector2DHasNext(Vector2D* obj) {

}

void vector2DFree(Vector2D* obj) {

}

/**
* Your Vector2D struct will be instantiated and called as such:
* Vector2D* obj = vector2DCreate(vec, vecSize, vecColSize);
* int param_1 = vector2DNext(obj);

* bool param_2 = vector2DHasNext(obj);

* vector2DFree(obj);
*/
```

Go Solution:

```
// Problem: Flatten 2D Vector
// Difficulty: Medium
// Tags: array
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

type Vector2D struct {

}


func Constructor(vec [][]int) Vector2D {

}


func (this *Vector2D) Next() int {
```

```
}


func (this *Vector2D) HasNext() bool {

}


/**
* Your Vector2D object will be instantiated and called as such:
* obj := Constructor(vec);
* param_1 := obj.Next();
* param_2 := obj.HasNext();
*/
```

Kotlin Solution:

```
class Vector2D(vec: Array<IntArray>) {

fun next(): Int {

}

fun hasNext(): Boolean {

}

}

/**
* Your Vector2D object will be instantiated and called as such:
* var obj = Vector2D(vec)
* var param_1 = obj.next()
* var param_2 = obj.hasNext()
*/
```

Swift Solution:

```
class Vector2D {
```

```
init(_ vec: [[Int]]) {

}

func next() -> Int {

}

func hasNext() -> Bool {

}
}

/**
* Your Vector2D object will be instantiated and called as such:
* let obj = Vector2D(vec)
* let ret_1: Int = obj.next()
* let ret_2: Bool = obj.hasNext()
*/
```

Rust Solution:

```
// Problem: Flatten 2D Vector
// Difficulty: Medium
// Tags: array
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

struct Vector2D {

}



/**
* `&self` means the method takes an immutable reference.
* If you need a mutable reference, change it to `&mut self` instead.
*/
impl Vector2D {
```

```
fn new(vec: Vec<Vec<i32>>) -> Self {

}

fn next(&self) -> i32 {

}

fn has_next(&self) -> bool {

}
}

/**
* Your Vector2D object will be instantiated and called as such:
* let obj = Vector2D::new(vec);
* let ret_1: i32 = obj.next();
* let ret_2: bool = obj.has_next();
*/
```

Ruby Solution:

```
class Vector2D

=begin
:type vec: Integer[][]
=end
def initialize(vec)

end


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

end


=begin
```

```
:rtype: Boolean
=end
def has_next()

end



end

# Your Vector2D object will be instantiated and called as such:
# obj = Vector2D.new(vec)
# param_1 = obj.next()
# param_2 = obj.has_next()
```

PHP Solution:

```
class Vector2D {
/**
* @param Integer[][] $vec
*/
function __construct($vec) {

}

/**
* @return Integer
*/
function next() {

}

/**
* @return Boolean
*/
function hasNext() {

}
}

/**
* Your Vector2D object will be instantiated and called as such:
```

```
* $obj = Vector2D($vec);
* $ret_1 = $obj->next();
* $ret_2 = $obj->hasNext();
*/
```

Dart Solution:

```
class Vector2D {

Vector2D(List<List<int>> vec) {

}

int next() {

}

bool hasNext() {

}
}

/**
* Your Vector2D object will be instantiated and called as such:
* Vector2D obj = Vector2D(vec);
* int param1 = obj.next();
* bool param2 = obj.hasNext();
*/
```

Scala Solution:

```
class Vector2D(_vec: Array[Array[Int]]) {

def next(): Int = {

}

def hasNext(): Boolean = {

}
```

```
}

/**
* Your Vector2D object will be instantiated and called as such:
* val obj = new Vector2D(vec)
* val param_1 = obj.next()
* val param_2 = obj.hasNext()
*/
```

Elixir Solution:

```
defmodule Vector2D do
@spec init_(vec :: [[integer]]) :: any
def init_(vec) do

end

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

end

@spec has_next() :: boolean
def has_next() do

end
end

# Your functions will be called as such:
# Vector2D.init_(vec)
# param_1 = Vector2D.next()
# param_2 = Vector2D.has_next()

# Vector2D.init_ will be called before every test case, in which you can do
some necessary initializations.
```

Erlang Solution:

```
-spec vector2_d_init_(Vec :: [[integer()]]) -> any().
vector2_d_init_(Vec) ->
.
```

```
-spec vector2_d_next() -> integer().
vector2_d_next() ->
.

-spec vector2_d_has_next() -> boolean().
vector2_d_has_next() ->
.


%% Your functions will be called as such:
%% vector2_d_init_(Vec),
%% Param_1 = vector2_d_next(),
%% Param_2 = vector2_d_has_next(),

%% vector2_d_init_ will be called before every test case, in which you can do some necessary initializations.
```

Racket Solution:

```
(define vector2-d%
(class object%
(super-new)

; vec : (listof (listof exact-integer?))
(init-field
vec)

; next : -> exact-integer?
(define/public (next)
)
; has-next : -> boolean?
(define/public (has-next)
)))

;; Your vector2-d% object will be instantiated and called as such:
;; (define obj (new vector2-d% [vec vec]))
;; (define param_1 (send obj next))
;; (define param_2 (send obj has-next))
```