Problem 900: RLE Iterator

Problem Information

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

Problem Description

We can use run-length encoding (i.e.,

RLE

) to encode a sequence of integers. In a run-length encoded array of even length

encoding

(

0-indexed

), for all even

i

,

encoding[i]

tells us the number of times that the non-negative integer value

encoding[i + 1]

is repeated in the sequence.

For example, the sequence

arr = [8,8,8,5,5]

can be encoded to be

encoding = [3,8,2,5]

.

encoding = [3,8,0,9,2,5]

and

encoding = [2,8,1,8,2,5]

are also valid

RLE

of

arr

.

Given a run-length encoded array, design an iterator that iterates through it.

Implement the

RLEIterator

class:

RLEIterator(int[] encoded)

Initializes the object with the encoded array

encoded

.

int next(int n)

Exhausts the next

n

elements and returns the last element exhausted in this way. If there is no element left to exhaust, return

-1

instead.

Example 1:

Input

["RLEIterator", "next", "next", "next", "next"] [[[3, 8, 0, 9, 2, 5]], [2], [1], [1], [2]]

Output

[null, 8, 8, 5, -1]

Explanation

RLEIterator rLEIterator = new RLEIterator([3, 8, 0, 9, 2, 5]); // This maps to the sequence [8,8,8,5,5]. rLEIterator.next(2); // exhausts 2 terms of the sequence, returning 8. The remaining sequence is now [8, 5, 5]. rLEIterator.next(1); // exhausts 1 term of the sequence, returning 8. The remaining sequence is now [5, 5]. rLEIterator.next(1); // exhausts 1 term of the sequence, returning 5. The remaining sequence is now [5]. rLEIterator.next(2); // exhausts 2 terms, returning -1. This is because the first term exhausted was 5, but the second term did not exist. Since the last term exhausted does not exist, we return -1.

Constraints:

2 <= encoding.length <= 1000

encoding.length

is even.

0 <= encoding[i] <= 10

9

1 <= n <= 10

9

At most

1000

calls will be made to

next

.

Code Snippets

C++:

```
class RLEIterator {
public:
RLEIterator(vector<int>& encoding) {

}

int next(int n) {

}
};

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

```
* RLEIterator* obj = new RLEIterator(encoding);
* int param_1 = obj->next(n);
*/
```

Java:

```
class RLEIterator {

public RLEIterator(int[] encoding) {

}

public int next(int n) {

}
}

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

Python3:

```
class RLEIterator:

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



def next(self, n: int) -> int:




# Your RLEIterator object will be instantiated and called as such:
# obj = RLEIterator(encoding)
# param_1 = obj.next(n)
```

Python:

```
class RLEIterator(object):
```

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



def next(self, n):
"""
:type n: int
:rtype: int
"""




# Your RLEIterator object will be instantiated and called as such:
# obj = RLEIterator(encoding)
# param_1 = obj.next(n)
```

JavaScript:

```
/**
* @param {number[]} encoding
*/
var RLEIterator = function(encoding) {

};

/**
* @param {number} n
* @return {number}
*/
RLEIterator.prototype.next = function(n) {

};

/**
* Your RLEIterator object will be instantiated and called as such:
* var obj = new RLEIterator(encoding)
* var param_1 = obj.next(n)
*/
```

TypeScript:

```
class RLEIterator {
constructor(encoding: number[]) {

}

next(n: number): number {

}
}

/**
* Your RLEIterator object will be instantiated and called as such:
* var obj = new RLEIterator(encoding)
* var param_1 = obj.next(n)
*/
```

C#:

```
public class RLEIterator {

public RLEIterator(int[] encoding) {

}

public int Next(int n) {

}
}

/**
* Your RLEIterator object will be instantiated and called as such:
* RLEIterator obj = new RLEIterator(encoding);
* int param_1 = obj.Next(n);
*/
```

C:

```




typedef struct {
} RLEIterator;


RLEIterator* rLEIteratorCreate(int* encoding, int encodingSize) {

}

int rLEIteratorNext(RLEIterator* obj, int n) {

}

void rLEIteratorFree(RLEIterator* obj) {

}

/**
* Your RLEIterator struct will be instantiated and called as such:
* RLEIterator* obj = rLEIteratorCreate(encoding, encodingSize);
* int param_1 = rLEIteratorNext(obj, n);

* rLEIteratorFree(obj);
*/
```

Go:

```
type RLEIterator struct {

}


func Constructor(encoding []int) RLEIterator {

}


func (this *RLEIterator) Next(n int) int {

}


/**
```

```
* Your RLEIterator object will be instantiated and called as such:
* obj := Constructor(encoding);
* param_1 := obj.Next(n);
*/
```

Kotlin:

```
class RLEIterator(encoding: IntArray) {

fun next(n: Int): Int {

}

}

/**
* Your RLEIterator object will be instantiated and called as such:
* var obj = RLEIterator(encoding)
* var param_1 = obj.next(n)
*/
```

Swift:

```
class RLEIterator {

init(_ encoding: [Int]) {

}

func next(_ n: Int) -> Int {

}
}

/**
* Your RLEIterator object will be instantiated and called as such:
* let obj = RLEIterator(encoding)
* let ret_1: Int = obj.next(n)
*/
```

Rust:

```
struct RLEIterator {

}



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

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

}

fn next(&self, n: i32) -> i32 {

}
}

/**
* Your RLEIterator object will be instantiated and called as such:
* let obj = RLEIterator::new(encoding);
* let ret_1: i32 = obj.next(n);
*/
```

Ruby:

```
class RLEIterator

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

end



=begin
:type n: Integer
:rtype: Integer
=end
def next(n)
```

```
end


end

# Your RLEIterator object will be instantiated and called as such:
# obj = RLEIterator.new(encoding)
# param_1 = obj.next(n)
```

PHP:

```
class RLEIterator {
/**
* @param Integer[] $encoding
*/
function __construct($encoding) {

}

/**
* @param Integer $n
* @return Integer
*/
function next($n) {

}
}

/**
* Your RLEIterator object will be instantiated and called as such:
* $obj = RLEIterator($encoding);
* $ret_1 = $obj->next($n);
*/
```

Dart:

```
class RLEIterator {

RLEIterator(List<int> encoding) {

}
int next(int n) {

}
}

/**
* Your RLEIterator object will be instantiated and called as such:
* RLEIterator obj = RLEIterator(encoding);
* int param1 = obj.next(n);
*/
```

Scala:

```
class RLEIterator(_encoding: Array[Int]) {

def next(n: Int): Int = {

}

}

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

Elixir:

```
defmodule RLEIterator do
@spec init_(encoding :: [integer]) :: any
def init_(encoding) do

end

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

end
end
```

```
# Your functions will be called as such:
# RLEIterator.init_(encoding)
# param_1 = RLEIterator.next(n)

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

Erlang:

```
-spec rle_iterator_init_(Encoding :: [integer()]) -> any().
rle_iterator_init_(Encoding) ->
.

-spec rle_iterator_next(N :: integer()) -> integer().
rle_iterator_next(N) ->
.



%% Your functions will be called as such:
%% rle_iterator_init_(Encoding),
%% Param_1 = rle_iterator_next(N),

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

Racket:

```
(define rle-iterator%
(class object%
(super-new)

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

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

;; Your rle-iterator% object will be instantiated and called as such:
;; (define obj (new rle-iterator% [encoding encoding]))
;; (define param_1 (send obj next n))
```

Solutions

C++ Solution:

```
/*
* Problem: RLE Iterator
* 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 RLEIterator {
public:
RLEIterator(vector<int>& encoding) {

}

int next(int n) {

}
};

/**
* Your RLEIterator object will be instantiated and called as such:
* RLEIterator* obj = new RLEIterator(encoding);
* int param_1 = obj->next(n);
*/
```

Java Solution:

```
/**
* Problem: RLE Iterator
* 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 RLEIterator {

public RLEIterator(int[] encoding) {

}

public int next(int n) {

}
}

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

Python3 Solution:

```
"""
Problem: RLE Iterator
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 RLEIterator:

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



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

Python Solution:

```
class RLEIterator(object):

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



def next(self, n):
"""
:type n: int
:rtype: int
"""




# Your RLEIterator object will be instantiated and called as such:
# obj = RLEIterator(encoding)
# param_1 = obj.next(n)
```

JavaScript Solution:

```
/**
* Problem: RLE Iterator
* 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[]} encoding
*/
var RLEIterator = function(encoding) {

};

/**
* @param {number} n
* @return {number}
```

```
*/
RLEIterator.prototype.next = function(n) {

};

/**
* Your RLEIterator object will be instantiated and called as such:
* var obj = new RLEIterator(encoding)
* var param_1 = obj.next(n)
*/
```

TypeScript Solution:

```
/**
* Problem: RLE Iterator
* 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 RLEIterator {
constructor(encoding: number[]) {

}

next(n: number): number {

}
}

/**
* Your RLEIterator object will be instantiated and called as such:
* var obj = new RLEIterator(encoding)
* var param_1 = obj.next(n)
*/
```

C# Solution:

```
/*
* Problem: RLE Iterator
* 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 RLEIterator {

public RLEIterator(int[] encoding) {

}

public int Next(int n) {

}
}

/**
* Your RLEIterator object will be instantiated and called as such:
* RLEIterator obj = new RLEIterator(encoding);
* int param_1 = obj.Next(n);
*/
```

C Solution:

```
/*
* Problem: RLE Iterator
* 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 {

} RLEIterator;


RLEIterator* rLEIteratorCreate(int* encoding, int encodingSize) {

}

int rLEIteratorNext(RLEIterator* obj, int n) {

}

void rLEIteratorFree(RLEIterator* obj) {

}

/**
* Your RLEIterator struct will be instantiated and called as such:
* RLEIterator* obj = rLEIteratorCreate(encoding, encodingSize);
* int param_1 = rLEIteratorNext(obj, n);

* rLEIteratorFree(obj);
*/
```

Go Solution:

```
// Problem: RLE Iterator
// 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 RLEIterator struct {

}



func Constructor(encoding []int) RLEIterator {
}


func (this *RLEIterator) Next(n int) int {

}


/**
* Your RLEIterator object will be instantiated and called as such:
* obj := Constructor(encoding);
* param_1 := obj.Next(n);
*/
```

Kotlin Solution:

```
class RLEIterator(encoding: IntArray) {

fun next(n: Int): Int {

}

}

/**
* Your RLEIterator object will be instantiated and called as such:
* var obj = RLEIterator(encoding)
* var param_1 = obj.next(n)
*/
```

Swift Solution:

```
class RLEIterator {

init(_ encoding: [Int]) {

}

func next(_ n: Int) -> Int {
}
}

/**
* Your RLEIterator object will be instantiated and called as such:
* let obj = RLEIterator(encoding)
* let ret_1: Int = obj.next(n)
*/
```

Rust Solution:

```
// Problem: RLE Iterator
// 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 RLEIterator {

}



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

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

}

fn next(&self, n: i32) -> i32 {

}
}

/**
```

```
* Your RLEIterator object will be instantiated and called as such:
* let obj = RLEIterator::new(encoding);
* let ret_1: i32 = obj.next(n);
*/
```

Ruby Solution:

```
class RLEIterator

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

end



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

end



end

# Your RLEIterator object will be instantiated and called as such:
# obj = RLEIterator.new(encoding)
# param_1 = obj.next(n)
```

PHP Solution:

```
class RLEIterator {
/**
* @param Integer[] $encoding
*/
function __construct($encoding) {

}
/**
* @param Integer $n
* @return Integer
*/
function next($n) {

}
}

/**
* Your RLEIterator object will be instantiated and called as such:
* $obj = RLEIterator($encoding);
* $ret_1 = $obj->next($n);
*/
```

Dart Solution:

```
class RLEIterator {

RLEIterator(List<int> encoding) {

}

int next(int n) {

}
}

/**
* Your RLEIterator object will be instantiated and called as such:
* RLEIterator obj = RLEIterator(encoding);
* int param1 = obj.next(n);
*/
```

Scala Solution:

```
class RLEIterator(_encoding: Array[Int]) {

def next(n: Int): Int = {
}

}

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

Elixir Solution:

```
defmodule RLEIterator do
@spec init_(encoding :: [integer]) :: any
def init_(encoding) do

end

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

end
end

# Your functions will be called as such:
# RLEIterator.init_(encoding)
# param_1 = RLEIterator.next(n)

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

Erlang Solution:

```
-spec rle_iterator_init_(Encoding :: [integer()]) -> any().
rle_iterator_init_(Encoding) ->
.

-spec rle_iterator_next(N :: integer()) -> integer().
rle_iterator_next(N) ->
.
```

```
%% Your functions will be called as such:
%% rle_iterator_init_(Encoding),
%% Param_1 = rle_iterator_next(N),

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

Racket Solution:

```
(define rle-iterator%
(class object%
(super-new)

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

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

;; Your rle-iterator% object will be instantiated and called as such:
;; (define obj (new rle-iterator% [encoding encoding]))
;; (define param_1 (send obj next n))
```