Problem 2757: Generate Circular Array Values

Problem Information

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

Problem Description

Given a

circular

array

arr

and an integer

startIndex

, return a generator object

gen

that yields values from

arr

.

The first time

gen.next()

is called on the generator, it should should yield

arr[startIndex]

.

Each subsequent time

gen.next()

is called, an integer

jump

will be passed into the function (Ex:

gen.next(-3)

).

If

jump

is positive, the index should increase by that value, however if the current index is the last index, it should instead jump to the first index.

If

jump

is negative, the index should decrease by the magnitude of that value, however if the current index is the first index, it should instead jump to the last index.

Example 1:

Input:

arr = [1,2,3,4,5], steps = [1,2,6], startIndex = 0

Output:

[1,2,4,5]

Explanation:

const gen = cycleGenerator(arr, startIndex); gen.next().value; // 1, index = startIndex = 0
 gen.next(1).value; // 2, index = 1, 0 -> 1 gen.next(2).value; // 4, index = 3, 1 -> 2 -> 3
 gen.next(6).value; // 5, index = 4, 3 -> 4 -> 0 -> 1 -> 2 -> 3 -> 4

Example 2:

Input:

arr = [10,11,12,13,14,15], steps = [1,4,0,-1,-3], startIndex = 1

Output:

[11,12,10,10,15,12]

Explanation:

const gen = cycleGenerator(arr, startIndex); gen.next().value;   // 11, index = 1
 gen.next(1).value; // 12, index = 2 gen.next(4).value; // 10, index = 0 gen.next(0).value; // 10, index = 0 gen.next(-1).value; // 15, index = 5 gen.next(-3).value; // 12, index = 2

Example 3:

Input:

arr = [2,4,6,7,8,10], steps = [-4,5,-3,10], startIndex = 3

Output:

[7,10,8,4,10]

Explanation:

const gen = cycleGenerator(arr, startIndex); gen.next().value   // 7, index = 3
 gen.next(-4).value // 10, index = 5 gen.next(5).value // 8, index = 4 gen.next(-3).value // 4,
index = 1   gen.next(10).value // 10, index = 5

Constraints:

1 <= arr.length <= 10

4

1 <= steps.length <= 100

-10

4

<= steps[i], arr[i] <= 10

4

0 <= startIndex < arr.length

Code Snippets

JavaScript:

```
/**
* @param {Array<number>} arr
* @param {number} startIndex
* @yields {number}
*/
var cycleGenerator = function* (arr, startIndex) {

};

/**
* const gen = cycleGenerator([1,2,3,4,5], 0);
* gen.next().value // 1
* gen.next(1).value // 2
* gen.next(2).value // 4
* gen.next(6).value // 5
*/
```

TypeScript:

```
function* cycleGenerator(arr: number[], startIndex: number):
Generator<number, void, number> {

};

/**
* const gen = cycleGenerator([1,2,3,4,5], 0);
* gen.next().value // 1
* gen.next(1).value // 2
* gen.next(2).value // 4
* gen.next(6).value // 5
*/
```

Solutions

JavaScript Solution:

```
/**
* Problem: Generate Circular Array Values
* 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 {Array<number>} arr
* @param {number} startIndex
* @yields {number}
*/
var cycleGenerator = function* (arr, startIndex) {
};

/**
* const gen = cycleGenerator([1,2,3,4,5], 0);
* gen.next().value // 1
* gen.next(1).value // 2
* gen.next(2).value // 4
* gen.next(6).value // 5
*/
```

TypeScript Solution:

```
/**
* Problem: Generate Circular Array Values
* 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
*/

function* cycleGenerator(arr: number[], startIndex: number):
Generator<number, void, number> {

};

/**
* const gen = cycleGenerator([1,2,3,4,5], 0);
* gen.next().value // 1
* gen.next(1).value // 2
* gen.next(2).value // 4
* gen.next(6).value // 5
*/
```