Problem 3248: Snake in Matrix

Problem Information

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

Problem Description

There is a snake in an

n x n

matrix

grid

and can move in

four possible directions

. Each cell in the

grid

is identified by the position:

grid[i][j] = (i * n) + j

.

The snake starts at cell 0 and follows a sequence of commands.

You are given an integer

n

representing the size of the

grid

and an array of strings

commands

where each

command[i]

is either

"UP"

,

"RIGHT"

,

"DOWN"

, and

"LEFT"

. It's guaranteed that the snake will remain within the

grid

boundaries throughout its movement.

Return the position of the final cell where the snake ends up after executing

commands

.

Example 1:

Input:

n = 2, commands = ["RIGHT","DOWN"]

Output:

3

Explanation:

0

1

2

3

0

1

2

3

0

1

2

3

Example 2:

Input:

n = 3, commands = ["DOWN","RIGHT","UP"]

Output:

1

Explanation:

0

1

2

3

4

5

6

7

8

0

1

2

3

4

6

7

8

Constraints:

2 <= n <= 10

1 <= commands.length <= 100

commands

consists only of

"UP"

,

"RIGHT"

,

"DOWN"

, and

"LEFT"

.

The input is generated such the snake will not move outside of the boundaries.

Code Snippets

C++:

```
class Solution {
public:
int finalPositionOfSnake(int n, vector<string>& commands) {

}
};
```

Java:

```
class Solution {
public int finalPositionOfSnake(int n, List<String> commands) {

}
}
```

Python3:

```
class Solution:
def finalPositionOfSnake(self, n: int, commands: List[str]) -> int:
```

Python:

```
class Solution(object):
def finalPositionOfSnake(self, n, commands):
"""
:type n: int
:type commands: List[str]
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number} n
* @param {string[]} commands
* @return {number}
*/
var finalPositionOfSnake = function(n, commands) {

};
```

TypeScript:

```
function finalPositionOfSnake(n: number, commands: string[]): number {

};
```

C#:

```
public class Solution {
public int FinalPositionOfSnake(int n, IList<string> commands) {

}
}
```

C:

```
int finalPositionOfSnake(int n, char** commands, int commandsSize) {

}
```

Go:

```
func finalPositionOfSnake(n int, commands []string) int {

}
```

Kotlin:

```
class Solution {
fun finalPositionOfSnake(n: Int, commands: List<String>): Int {

}
}
```

Swift:

```
class Solution {
func finalPositionOfSnake(_ n: Int, _ commands: [String]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn final_position_of_snake(n: i32, commands: Vec<String>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer} n
# @param {String[]} commands
# @return {Integer}
def final_position_of_snake(n, commands)

end
```

PHP:

```
class Solution {

/**
* @param Integer $n
* @param String[] $commands
* @return Integer
*/
function finalPositionOfSnake($n, $commands) {

}
}
```

Dart:

```
class Solution {
int finalPositionOfSnake(int n, List<String> commands) {

}
}
```

Scala:

```
object Solution {
def finalPositionOfSnake(n: Int, commands: List[String]): Int = {

}
```

```
}
```

Elixir:

```
defmodule Solution do
@spec final_position_of_snake(n :: integer, commands :: [String.t]) ::
integer
def final_position_of_snake(n, commands) do

end
end
```

Erlang:

```
-spec final_position_of_snake(N :: integer(), Commands ::
[unicode:unicode_binary()]) -> integer().
final_position_of_snake(N, Commands) ->
.
```

Racket:

```
(define/contract (final-position-of-snake n commands)
(-> exact-integer? (listof string?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Snake in Matrix
* Difficulty: Easy
* Tags: array, string
*
* 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 Solution {
```

```
public:
int finalPositionOfSnake(int n, vector<string>& commands) {

}
};
```

Java Solution:

```
/**
* Problem: Snake in Matrix
* Difficulty: Easy
* Tags: array, string
*
* 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 Solution {
public int finalPositionOfSnake(int n, List<String> commands) {

}
}
```

Python3 Solution:

```
"""
Problem: Snake in Matrix
Difficulty: Easy
Tags: array, string

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 Solution:
def finalPositionOfSnake(self, n: int, commands: List[str]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def finalPositionOfSnake(self, n, commands):
"""
:type n: int
:type commands: List[str]
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Snake in Matrix
* Difficulty: Easy
* Tags: array, string
*
* 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} n
* @param {string[]} commands
* @return {number}
*/
var finalPositionOfSnake = function(n, commands) {

};
```

TypeScript Solution:

```
/**
* Problem: Snake in Matrix
* Difficulty: Easy
* Tags: array, string
*
* 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 finalPositionOfSnake(n: number, commands: string[]): number {

};
```

C# Solution:

```
/*
* Problem: Snake in Matrix
* Difficulty: Easy
* Tags: array, string
*
* 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 Solution {
public int FinalPositionOfSnake(int n, IList<string> commands) {

}
}
```

C Solution:

```
/*
* Problem: Snake in Matrix
* Difficulty: Easy
* Tags: array, string
*
* 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
*/

int finalPositionOfSnake(int n, char** commands, int commandsSize) {

}
```

Go Solution:

```
// Problem: Snake in Matrix
// Difficulty: Easy
// Tags: array, string
//
// 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

func finalPositionOfSnake(n int, commands []string) int {

}
```

Kotlin Solution:

```
class Solution {
fun finalPositionOfSnake(n: Int, commands: List<String>): Int {

}
}
```

Swift Solution:

```
class Solution {
func finalPositionOfSnake(_ n: Int, _ commands: [String]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Snake in Matrix
// Difficulty: Easy
// Tags: array, string
//
// 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

impl Solution {
pub fn final_position_of_snake(n: i32, commands: Vec<String>) -> i32 {

}
```

```
}
```

Ruby Solution:

```
# @param {Integer} n
# @param {String[]} commands
# @return {Integer}
def final_position_of_snake(n, commands)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $n
* @param String[] $commands
* @return Integer
*/
function finalPositionOfSnake($n, $commands) {

}
}
```

Dart Solution:

```
class Solution {
int finalPositionOfSnake(int n, List<String> commands) {

}
}
```

Scala Solution:

```
object Solution {
def finalPositionOfSnake(n: Int, commands: List[String]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec final_position_of_snake(n :: integer, commands :: [String.t]) ::
integer
def final_position_of_snake(n, commands) do

end
end
```

Erlang Solution:

```
-spec final_position_of_snake(N :: integer(), Commands ::
[unicode:unicode_binary()]) -> integer().
final_position_of_snake(N, Commands) ->
.
```

Racket Solution:

```
(define/contract (final-position-of-snake n commands)
(-> exact-integer? (listof string?) exact-integer?)
)
```