Problem 2120: Execution of All Suffix Instructions Staying in a Grid

Problem Information

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

Problem Description

There is an

n x n

grid, with the top-left cell at

(0, 0)

and the bottom-right cell at

(n - 1, n - 1)

. You are given the integer

n

and an integer array

startPos

where

startPos = [start

row

, start

col

]

indicates that a robot is initially at cell

(start

row

, start

col

)

.

You are also given a

0-indexed

string

s

of length

m

where

s[i]

is the

i

th

instruction for the robot:

'L'

(move left),

'R'

(move right),

'U'

(move up), and

'D'

(move down).

The robot can begin executing from any

i

th

instruction in

s

. It executes the instructions one by one towards the end of

s

but it stops if either of these conditions is met:

The next instruction will move the robot off the grid.

There are no more instructions left to execute.

Return

an array

answer

of length

m

where

answer[i]

is

the number of instructions

the robot can execute if the robot

begins executing from

the

i

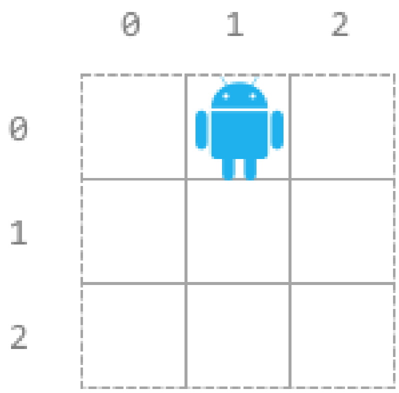
th

instruction in

s

.

Example 1:



Input:

n = 3, startPos = [0,1], s = "RRDDLU"

Output:

[1,5,4,3,1,0]

Explanation:

Starting from startPos and beginning execution from the i

th

instruction: - 0

th

: "

R

RDDLU". Only one instruction "R" can be executed before it moves off the grid. - 1

st

: "

RDDLU

". All five instructions can be executed while it stays in the grid and ends at (1, 1). - 2

nd

: "

DDLU

". All four instructions can be executed while it stays in the grid and ends at (1, 0). - 3

rd

: "

DLU

". All three instructions can be executed while it stays in the grid and ends at (0, 0). - 4

th

: "

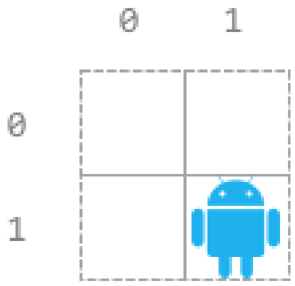
L

U". Only one instruction "L" can be executed before it moves off the grid. - 5

th

: "U". If moving up, it would move off the grid.

Example 2:



Input:

n = 2, startPos = [1,1], s = "LURD"

Output:

[4,1,0,0]

Explanation:

- 0

th

: "

LURD

". - 1

st

: "

U

RD". - 2

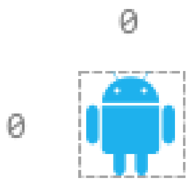
nd

: "RD". - 3

rd

: "D".

Example 3:



Input:

n = 1, startPos = [0,0], s = "LRUD"

Output:

[0,0,0,0]

Explanation:

No matter which instruction the robot begins execution from, it would move off the grid.

Constraints:

m == s.length

1 <= n, m <= 500

startPos.length == 2

0 <= start

row

, start

col

< n

s

consists of

'L'

,

'R'

,

'U'

, and

'D'

.

Code Snippets

C++:

```
class Solution {
public:
vector<int> executeInstructions(int n, vector<int>& startPos, string s) {

}
};
```

Java:

```
class Solution {
public int[] executeInstructions(int n, int[] startPos, String s) {

}
}
```

Python3:

```
class Solution:
def executeInstructions(self, n: int, startPos: List[int], s: str) ->
List[int]:
```

Python:

```
class Solution(object):
def executeInstructions(self, n, startPos, s):
"""
:type n: int
:type startPos: List[int]
:type s: str
:rtype: List[int]
"""
```

JavaScript:

```
/**
* @param {number} n
* @param {number[]} startPos
* @param {string} s
* @return {number[]}
*/
var executeInstructions = function(n, startPos, s) {

};
```

TypeScript:

```
function executeInstructions(n: number, startPos: number[], s: string):
number[] {

};
```

C#:

```
public class Solution {
public int[] ExecuteInstructions(int n, int[] startPos, string s) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
```

```
int* executeInstructions(int n, int* startPos, int startPosSize, char* s,
int* returnSize) {

}
```

Go:

```
func executeInstructions(n int, startPos []int, s string) []int {

}
```

Kotlin:

```
class Solution {
fun executeInstructions(n: Int, startPos: IntArray, s: String): IntArray {

}
}
```

Swift:

```
class Solution {
func executeInstructions(_ n: Int, _ startPos: [Int], _ s: String) -> [Int] {

}
}
```

Rust:

```
impl Solution {
pub fn execute_instructions(n: i32, start_pos: Vec<i32>, s: String) ->
Vec<i32> {

}
}
```

Ruby:

```
# @param {Integer} n
# @param {Integer[]} start_pos
# @param {String} s
# @return {Integer[]}
```

```
def execute_instructions(n, start_pos, s)

end
```

PHP:

```
class Solution {

/**
* @param Integer $n
* @param Integer[] $startPos
* @param String $s
* @return Integer[]
*/
function executeInstructions($n, $startPos, $s) {

}
}
```

Dart:

```
class Solution {
List<int> executeInstructions(int n, List<int> startPos, String s) {

}
}
```

Scala:

```
object Solution {
def executeInstructions(n: Int, startPos: Array[Int], s: String): Array[Int]
= {

}
}
```

Elixir:

```
defmodule Solution do
@spec execute_instructions(n :: integer, start_pos :: [integer], s ::
String.t) :: [integer]
def execute_instructions(n, start_pos, s) do
```

```
end
end
```

Erlang:

```
-spec execute_instructions(N :: integer(), StartPos :: [integer()], S ::
unicode:unicode_binary()) -> [integer()].
execute_instructions(N, StartPos, S) ->
.
```

Racket:

```
(define/contract (execute-instructions n startPos s)
(-> exact-integer? (listof exact-integer?) string? (listof exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Execution of All Suffix Instructions Staying in a Grid
* Difficulty: Medium
* 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:
vector<int> executeInstructions(int n, vector<int>& startPos, string s) {

}
};
```

Java Solution:

```
/**
* Problem: Execution of All Suffix Instructions Staying in a Grid
* Difficulty: Medium
* 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[] executeInstructions(int n, int[] startPos, String s) {

}
}
```

Python3 Solution:

```
"""
Problem: Execution of All Suffix Instructions Staying in a Grid
Difficulty: Medium
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 executeInstructions(self, n: int, startPos: List[int], s: str) ->
List[int]:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def executeInstructions(self, n, startPos, s):
"""
:type n: int
:type startPos: List[int]
:type s: str
```

```
:rtype: List[int]
"""
```

JavaScript Solution:

```
/**
* Problem: Execution of All Suffix Instructions Staying in a Grid
* Difficulty: Medium
* 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 {number[]} startPos
* @param {string} s
* @return {number[]}
*/
var executeInstructions = function(n, startPos, s) {

};
```

TypeScript Solution:

```
/**
* Problem: Execution of All Suffix Instructions Staying in a Grid
* Difficulty: Medium
* 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 executeInstructions(n: number, startPos: number[], s: string):
number[] {

};
```

C# Solution:

```
/*
* Problem: Execution of All Suffix Instructions Staying in a Grid
* Difficulty: Medium
* 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[] ExecuteInstructions(int n, int[] startPos, string s) {

}
}
```

C Solution:

```
/*
* Problem: Execution of All Suffix Instructions Staying in a Grid
* Difficulty: Medium
* 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
*/

/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* executeInstructions(int n, int* startPos, int startPosSize, char* s,
int* returnSize) {

}
```

Go Solution:

```
// Problem: Execution of All Suffix Instructions Staying in a Grid
// Difficulty: Medium
// 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 executeInstructions(n int, startPos []int, s string) []int {

}
```

Kotlin Solution:

```
class Solution {
fun executeInstructions(n: Int, startPos: IntArray, s: String): IntArray {

}
}
```

Swift Solution:

```
class Solution {
func executeInstructions(_ n: Int, _ startPos: [Int], _ s: String) -> [Int] {

}
}
```

Rust Solution:

```
// Problem: Execution of All Suffix Instructions Staying in a Grid
// Difficulty: Medium
// 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 execute_instructions(n: i32, start_pos: Vec<i32>, s: String) ->
Vec<i32> {

}

}
```

Ruby Solution:

```
# @param {Integer} n
# @param {Integer[]} start_pos
# @param {String} s
# @return {Integer[]}
def execute_instructions(n, start_pos, s)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $n
* @param Integer[] $startPos
* @param String $s
* @return Integer[]
*/
function executeInstructions($n, $startPos, $s) {

}
}
```

Dart Solution:

```
class Solution {
List<int> executeInstructions(int n, List<int> startPos, String s) {

}
}
```

Scala Solution:

```
object Solution {
def executeInstructions(n: Int, startPos: Array[Int], s: String): Array[Int]
= {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec execute_instructions(n :: integer, start_pos :: [integer], s ::
String.t) :: [integer]
def execute_instructions(n, start_pos, s) do

end
end
```

Erlang Solution:

```
-spec execute_instructions(N :: integer(), StartPos :: [integer()], S ::
unicode:unicode_binary()) -> [integer()].
execute_instructions(N, StartPos, S) ->
.
```

Racket Solution:

```
(define/contract (execute-instructions n startPos s)
(-> exact-integer? (listof exact-integer?) string? (listof exact-integer?))
)
```