2069. Walking Robot Simulation II Leetcode Link **Problem Description** Given a robot that starts at the origin position (0,0) and has a width and height associated with it, the robot is required to be able to move through a grid in a defined pattern. Initially, the robot moves in a round trip fashion, going from the origin position to the last position in the same row, then from the last position in a column to the last position in the last row, and finally back to the origin position, completing the cycle. The robot should be able to move num steps, and after moving the given number of steps, it should return its current position and direction. Walkthrough Let's start with an example, given a robot with width = 4 and height = 3, the grid would look like this: 7 12->7--6--5 The robot starts at position 0,0 and moves through the path as shown above. It should be able to perform steps and return the current position and direction. Approach The solution provided uses a vector of pairs, where each pair represents the position of the robot and the direction it is facing at that position. The solution initializes the pos vector by pushing the positions and directions during the construction of the robot object. The step() function increases the index i of the pos vector by the given number of steps modulo the total number of positions in the vector. The getPos() function returns the current position of the robot, and the getDir() function returns the current direction of the robot. **ASCII Illustration** 3 0--1--2--3 4 | | | 5 11—10—9 Example: 1. Robot initialized with width = 4 and height = 3, it starts at position (0, 0) and facing 'South'. 2. Robot performs step(3), it moves to position (3, 0) and faces 'North'. 3. Robot performs step(5), it moves to position (0, 2) and faces 'South'. Solutions **Python Solution** python class Solution: def __init__(self, width, height): self.pos = [((0, 0), "South")] for i in range(1, width): self.pos.append(((i, 0), "East")) for j in range(1, height): self.pos.append(((width - 1, j), "North")) 9 10 for i in range(width -2, -1, -1): self.pos.append(((i, height - 1), "West")) 11 12 for j in range(height -2, 0, -1): self.pos.append(((0, j), "South")) 13 self.i = 014 15 self.isOrigin = True 16 17 def step(self, num): 18 self.isOrigin = False 19 self.i = (self.i + num) % len(self.pos) 20 21 def getPos(self): 22 return self.pos[self.i][0] 23 24 def getDir(self): return "East" if self.isOrigin else self.pos[self.i][1] 25 **Java Solution** java import java.util.ArrayList; import java.util.Arrays; public class Solution { private ArrayList<int[]> pos; 8 private String[] dir; private int i; 9 10 private boolean isOrigin; 11 12 public Solution(int width, int height) { 13 pos = new ArrayList<>(); 14 dir = new String[width * 2 + height * 2 - 4]; 15 pos.add(new int[]{0, 0}); 16 dir[0] = "South"; 17 int k = 1; for (int i = 1; i < width; i++) { 18 19 pos.add(new int[]{i, 0}); 20 dir[k++] = "East"; 21 22 for (int j = 1; $j < height; j++) {$ 23 pos.add(new int[]{width - 1, j}); 24 dir[k++] = "North"; 25 26 for (int i = width - 2; i >= 0; i--) { 27 pos.add(new int[]{i, height - 1}); 28 dir[k++] = "West"; 29 30 for (int $j = height - 2; j > 0; j--) {$ 31 pos.add(new int[]{0, j}); 32 dir[k++] = "South"; 33 34 isOrigin = true; 35 i = 0;36 37 38 public void step(int num) { 39 isOrigin = false; 40 i = (i + num) % pos.size();41 42 43 public int[] getPos() { 44 return pos.get(i); 45 46 47 public String getDir() { 48 return isOrigin ? "East" : dir[i]; 49 50 }

16 for (int j = 1; j < height; ++j) {</pre> 17 18 pos.Add(Tuple.Create(new int[]{width - 1, j}, "North")); 19 for (int i = width - 2; i >= 0; --i) { 20 21 pos.Add(Tuple.Create(new int[]{i, height - 1}, "West")); 22 for (int $j = height - 2; j > 0; ---j) {$ 23 pos.Add(Tuple.Create(new int[]{0, j}, "South")); 24 25 26 isOrigin = true; 27 i = 0;28 29 30 public void Step(int num) { isOrigin = false; 31 32 i = (i + num) % pos.Count;33 34 35 public int[] GetPos() { 36 return pos[i].Item1; 37 38 39 public String GetDir() { return isOrigin ? "East" : pos[i].Item2; 40 41 42 } **Input and Output** The input for the given solutions is: 1. A robot object is created using two integers width and height of the grid. Example 1: 2 python 3 robot = Solution(4, 3) Example 2:

java

python

(0, 0)

JavaScript Solution

class Solution {

step(num) {

getPos() {

getDir() {

C++ Solution

3 #include <vector>

#include <string>

class Solution {

i = 0;

using namespace std;

isOrigin = true;

void step(int num) {

isOrigin = false;

vector<int> getPos() {

string getDir() {

bool isOrigin;

return pos[i].first;

i = (i + num) % pos.size();

this.pos = [];

this.i = 0;

constructor(width, height) {

this.pos.push([[0, 0], "South"]);

this.pos.push([[i, 0], "East"]);

for (let i = width - 2; i >= 0; --i)

for (let j = height - 2; j > 0; ---j)

this.pos.push([[0, j], "South"]);

this.pos.push([[width - 1, j], "North"]);

this.pos.push([[i, height - 1], "West"]);

this.i = (this.i + num) % this.pos.length;

return this.isOrigin ? "East" : this.pos[this.i][1];

for (let i = 1; i < width; ++i)

for (let j = 1; j < height; ++j)

this.isOrigin = true;

this.isOrigin = false;

return this.pos[this.i][0];

Solution(int width, int height) {

pos.push_back({{0, 0}, "South"});

pos.push_back({{i, 0}, "East"});

for (int i = width - 2; $i \ge 0$; --i)

for (int j = height - 2; j > 0; ---j)

pos.push_back({{0, j}, "South"});

return isOrigin ? "East" : pos[i].second;

vector<pair<vector<int>, string>> pos;

using System.Collections.Generic;

private bool isOrigin;

private List<Tuple<int[], string>> pos;

public Solution(int width, int height) {

for (int i = 1; i < width; ++i) {

pos = new List<Tuple<int[], string>>();

pos.Add(Tuple.Create(new int[]{0, 0}, "South"));

pos.Add(Tuple.Create(new int[]{i, 0}, "East"));

public class Solution {

private int i;

pos.push_back({{width - 1, j}, "North"});

pos.push_back({{i, height - 1}, "West"});

for (int i = 1; i < width; ++i)

for (int j = 1; j < height; ++j)

2 javascript

8

9

10

11

12

13

14

15

16

17

18

19

20

22

23

24

25

26

27

28

29

30

31 }

cpp

public:

6

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

26

27

28

29

30

31

32

33

34

35

36

38

39

40 };

8

9

10

11

12

13

14

15

private:

int i;

C# Solution

2 csharp

using System;

3 Solution robot = new Solution(4, 3); 2. The function step() is called for the given robot indicating the number of steps the robot should take. Example 1: 2 python robot.step(3) Example 2: 2 java 3 robot.step(3); The output for the given solutions consists of two functions: 1. The function getPos() returns the current position of the robot in the form of a pair (tuple in python, int[] in Java, array in JavaScript, vector in C++, int[] in C#). Example 1: 2 python 3 robot.getPos() Example 2: 2 java 3 robot.getPos(); 2. The function getDir() returns the current direction of the robot in the form of a string. Example 1: 2 python 3 robot.getDir() Example 2: java 3 robot.getDir();

These given solutions return the correct output, and the problem can be solved efficiently for the given constraints. **Examples** Example 1: Robot created with width = 4, height = 3, and starting position (0, 0) facing South. Input: python robot = Solution(4, 3)robot.getPos() 5 robot.getDir() Output:

4 "South" Example 2: Robot performs step(3) and moves to position (3, 0) facing North. Input: 2 python robot.step(3) robot.getPos() 5 robot.getDir() Output: 2 python (3, 0)"North"

Example 3: Robot performs step(5) and moves to position (0, 2) facing South. Input: python robot.step(5) robot.getPos() robot.getDir()

Output: python (0, 2)4 "South"

The given solutions are efficient, and the problem can be solved using these methods in Python, Java, JavaScript, C++, and C#.

Level Up Your

Algo Skills

Get Premium

Got a question? Ask the Teaching Assistant anything you don't understand.