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:

```


0--1--2--3
|        |
11--10--9
|        |
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

```


0--1--2--3
|        |
11--10--9
|        |
12->7--6--5
```

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"))
        for i in range(width - 2, -1, -1):
            self.pos.append(((i, height - 1), "West"))
        for j in range(height - 2, 0, -1):
            self.pos.append(((0, j), "South"))
        self.i = 0
        self.isOrigin = True

    def step(self, num):
        self.isOrigin = False
        self.i = (self.i + num) % len(self.pos)

    def getPos(self):
        return self.pos[self.i][0]

    def getDir(self):
        return "East" if self.isOrigin else self.pos[self.i][1]
```

Java Solution

```

java
import java.util.ArrayList;
import java.util.Arrays;

public class Solution {
    private ArrayList<int[]> pos;
    private String[] dir;
    private int i;
    private boolean isOrigin;

    public Solution(int width, int height) {
        pos = new ArrayList<>();
        dir = new String[width * 2 + height * 2 - 4];
        pos.add(new int[]{0, 0});
        dir[0] = "South";
        int k = 1;
        for (int i = 1; i < width; i++) {
            pos.add(new int[]{i, 0});
            dir[k++] = "East";
        }
        for (int j = 1; j < height; j++) {
            pos.add(new int[]{width - 1, j});
            dir[k++] = "North";
        }
        for (int i = width - 2; i >= 0; i--) {
            pos.add(new int[]{i, height - 1});
            dir[k++] = "West";
        }
        for (int j = height - 2; j > 0; j--) {
            pos.add(new int[]{0, j});
            dir[k++] = "South";
        }
        isOrigin = true;
        i = 0;
    }

    public void step(int num) {
        isOrigin = false;
        i = (i + num) % pos.size();
    }

    public int[] getPos() {
        return pos.get(i);
    }

    public String getDir() {
        return isOrigin ? "East" : dir[i];
    }
}
```

JavaScript Solution

```

javascript
class Solution {
  constructor(width, height) {
    this.pos = [];
    this.isOrigin = true;
    this.i = 0;
    this.pos.push([[0, 0], "South"]);
    for (let i = 1; i < width; ++i)
      this.pos.push([[i, 0], "East"]);
    for (let j = 1; j < height; ++j)
      this.pos.push([[width - 1, j], "North"]);
    for (let i = width - 2; i >= 0; --i)
      this.pos.push([[i, height - 1], "West"]);
    for (let j = height - 2; j > 0; --j)
      this.pos.push([[0, j], "South"]);
  }

  step(num) {
    this.isOrigin = false;
    this.i = (this.i + num) % this.pos.length;
  }

  getPos() {
    return this.pos[this.i][0];
  }

  getDir() {
    return this.isOrigin ? "East" : this.pos[this.i][1];
  }
}
```

C++ Solution

```

cpp
#include <vector>
#include <string>
using namespace std;

class Solution {
public:
  Solution(int width, int height) {
    pos.push_back({{0, 0}, "South"});
    for (int i = 1; i < width; ++i)
      pos.push_back({{i, 0}, "East"});
    for (int j = 1; j < height; ++j)
      pos.push_back({{width - 1, j}, "North"});
    for (int i = width - 2; i >= 0; --i)
      pos.push_back({{i, height - 1}, "West"});
    for (int j = height - 2; j > 0; --j)
      pos.push_back({{0, j}, "South"});
    isOrigin = true;
    i = 0;
  }

  void step(int num) {
    isOrigin = false;
    i = (i + num) % pos.size();
  }

  vector<int> getPos() {
    return pos[i].first;
  }

  string getDir() {
    return isOrigin ? "East" : pos[i].second;
  }

 private:
  bool isOrigin;
  int i;
  vector<pair<vector<int>, string>> pos;
};
```

C# Solution

```

csharp
using System;
using System.Collections.Generic;

public class Solution {
    private List<Tuple<int[], string>> pos;
    private bool isOrigin;
    private int i;

    public Solution(int width, int height) {
        pos = new List<Tuple<int[], string>>();
        pos.Add(Tuple.Create(new int[]{0, 0}, "South"));
        for (int i = 1; i < width; ++i) {
            pos.Add(Tuple.Create(new int[]{i, 0}, "East"));
        }
        for (int j = 1; j < height; ++j) {
            pos.Add(Tuple.Create(new int[]{width - 1, j}, "North"));
        }
        for (int i = width - 2; i >= 0; --i) {
            pos.Add(Tuple.Create(new int[]{i, height - 1}, "West"));
        }
        for (int j = height - 2; j > 0; --j) {
            pos.Add(Tuple.Create(new int[]{0, j}, "South"));
        }
        isOrigin = true;
        i = 0;
    }

    public void Step(int num) {
        isOrigin = false;
        i = (i + num) % pos.Count;
    }

    public int[] GetPos() {
        return pos[i].Item1;
    }

    public String GetDir() {
        return isOrigin ? "East" : pos[i].Item2;
    }
}
```

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:

```

python
robot = Solution(4, 3)
```

Example 2:

```

java
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:

```

python
robot.step(3)
```

Example 2:

```

java
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:

```

python
robot.getPos()
```

Example 2:

```

java
robot.getPos();
```

2. The function `getDir()` returns the current direction of the robot in the form of a string.

Example 1:

```

python
robot.getDir()
```

Example 2:

```

java
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()
robot.getDir()
```

Output:

```

python
(0, 0)
"South"
```

Example 2:

Robot performs `step(3)` and moves to position `(3, 0)` facing `North`.

Input:

```

python
robot.step(3)
robot.getPos()
robot.getDir()
```

Output:

```

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)
"South"
```

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

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