LeetCode 353

https://leetcode.com/problems/design-snake-game/description/

Yifeng Zeng

Description

353. Design Snake Game

Design a Snake game that is played on a device with screen size = width x height. Play the game online if you are not familiar with the game.

The snake is initially positioned at the top left corner (0,0) with length = 1 unit.

You are given a list of food's positions in row-column order. When a snake eats the food, its length and the game's score both increase by 1.

Each food appears one by one on the screen. For example, the second food will not appear until the first food was eaten by the snake.

When a food does appear on the screen, it is guaranteed that it will not appear on a block occupied by the snake.

Example:
Given width = 3, height = 2, and food = [[1,2],[0,1]].

Snake snake = new Snake(width, height, food);

Initially the snake appears at position (0,0) and the food at (1,2).

```
|S| | |
| | |F|
```

snake.move("R"); -> Returns 0

```
| |S| |
| | |F|
```

snake.move("D"); -> Returns 0

```
| | | |
| |S|F|
```

snake.move("R"); -> Returns 1 (Snake eats the first food and right after that, the second food appears at (0,1))

```
| |F| |
| |S|S|
```

snake.move("U"); -> Returns 1

```
| |F|S|
| | |S|
```

snake.move("L"); -> Returns 2 (Snake eats the second food)

```
| |S|S|
| | |S|
```

snake.move("U"); -> Returns -1 (Game over because snake collides with border)

Idea Report

Basically this is a step by step procedure problem, we anaylize all the possible situations:

1. Initialization, we save the width, height so we can check if snake is within the boundary. We

also need to save the foods in a queue, because the food appears only one at a time.

2. For snake object, while it moves, its head will get to a new location and tail will disappear from the last location, so we can use a queue to save all the locations that snake is at. We enqueue the next location, and dequeue the tail.
3. Now we handle all the moves, we first get the next location based on the current head of the snake and the moving direction from the input string.
    - a. If next location is out of display boundary, we clear out the snake, add the current location back to the snake and return -1. Because the requirement is even if the snake is dead, we can still play starting from the current location with snake length only be 1.
    - b. If next location is the snake itself, that means it eats itself and dies, we do the same thing as a. (so I put both of these checks in isInbound()).
        - Note: There is one exception where the next location is the current tail of the snake, we should discuss about this particular situation, but this question shows it's still valid and snake doesn't die, so we code this as a isInbound() = true.
    - c. If next location is inbound (snake doesn't die), we have two different situations.
        - If next location is a food, the snake size increase by one, so we enqueue the new location, and keep the tail where it was.
        - If next location is not a food, the snake size doesn't change, so we enqueue the new location, and dequeue the tail (last location in the queue).

Code:

```
class SnakeGame {
    // AC
    private int width;
    private int height;
    private Deque<int[]> snake;
    private Deque<int[]> foods;
    private int[] dx = {-1, 0, 0, 1};
    private int[] dy = {0, -1, 1, 0};

    public SnakeGame(int width, int height, int[][] food) {
        this.width = width;
        this.height = height;
        snake = new ArrayDeque<>();
        snake.offerLast(new int[]{0, 0});
        // foods = new LinkedList<>();
        // for (int[] f : food) {
        //     foods.offer(f);
        // }
        foods = new LinkedList<>(Arrays.asList(food));
    }

    public int move(String direction) {
```

```
            int dir = direction.equals("U") ? 0 : direction.equals("L") ?
                      1 : direction.equals("R") ? 2 : 3;
            int[] cur = snake.peekLast();
            int[] next = new int[]{cur[0] + dx[dir], cur[1] + dy[dir]};

            if (!isInbound(next)) {
                snake.clear();
                snake.offerLast(cur);
                return -1;
            }

            snake.offerLast(next);
            if (Arrays.equals(next, foods.peek())) {
                foods.poll();
            } else {
                snake.pollFirst();
            }
            return snake.size() - 1;
        }

        private boolean isInbound(int[] next) {
            // not within the screen
            if (next[0] < 0 || height <= next[0]
                || next[1] < 0 || width <= next[1]) {
                return false;
            }

            // check if next move eats the snake body or not
            // if next location is the current tail of the snake,
            // we can still move and snake doesn't die
            if (Arrays.equals(next, snake.peekFirst())) {
                return true;
            }

            for (int[] iter : snake) {
                if (Arrays.equals(next, iter)) {
                    return false;
                }
            }

            return true;
        }
    }
```

Summary

- Discuss and list all the situations and code them accordingly, the coding is actually not

difficult, but we need to pay attention to details/corner cases when we discuss them.