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)
```

Credits:
Special thanks to @elmirap for adding this problem and creating all test cases.

Solution 1

```
class SnakeGame {
public:
/** Initialize your data structure here.
@param width - screen width
@param height - screen height
@param food - A list of food positions
E.g food = [[1,1], [1,0]] means the first food is positioned at [1,1], the second
is at [1,0]. */

int w, h, pos;
vector<pair<int, int>> food;
set<pair<int, int>> hist;
deque<pair<int, int>> q;

SnakeGame(int width, int height, vector<pair<int, int>> food) {
    this->food = food;
    w = width, h = height, pos = 0;
    q.push_back(make_pair(0, 0));
}

/** Moves the snake.
@param direction - 'U' = Up, 'L' = Left, 'R' = Right, 'D' = Down
@return The game's score after the move. Return -1 if game over.
Game over when snake crosses the screen boundary or bites its body. */
int move(string direction) {
    int row = q.back().first, col = q.back().second;
    pair<int, int> d = q.front(); q.pop_front();
    hist.erase(d);

    if (direction == "U")
        row--;
    else if (direction == "D")
        row++;
    else if (direction == "L")
        col--;
    else if (direction == "R")
        col++;

    if (row < 0 || col < 0 || col >= w || row >= h || hist.count(make_pair(row, co
l)))
        return -1;

    hist.insert(make_pair(row, col));
    q.push_back(make_pair(row, col));

    if (pos >= food.size())
        return q.size() - 1;

    if (row == food[pos].first && col == food[pos].second) {
        pos++;
        q.push_front(d);
        hist.insert(d);
    }
```

```
    return q.size() - 1;
  }
};
```

written by jaewoo original link here

Solution 2

public class SnakeGame {

```
    class Position{
        int x;
        int y;
        public Position(int x,int y){
            this.x = x;
            this.y = y;
        }
        public boolean isEqual(Position p){
            return this.x==p.x && this.y == p.y ;
        }
    }
    int len;
    int rows ,cols;

    int[][] food;
    LinkedList<Position> snake;

    /** Initialize your data structure here.
        @param width - screen width
        @param height - screen height
        @param food - A list of food positions
        E.g food = [[1,1], [1,0]] means the first food is positioned at [1,1], th
e second is at [1,0]. */
    public SnakeGame(int width, int height, int[][] food) {
        this.rows = height;
        this.cols = width;
        this.food = food;

        snake = new LinkedList<Position>();
        snake.add(new Position(0,0));
        len = 0;
    }

    /** Moves the snake.
        @param direction - 'U' = Up, 'L' = Left, 'R' = Right, 'D' = Down
        @return The game's score after the move. Return -1 if game over.
        Game over when snake crosses the screen boundary or bites its body. */
    public int move(String direction) {
        //if(len>=food.length) return len;

        Position cur = new Position(snake.get(0).x,snake.get(0).y);

        switch(direction){
        case "U":
            cur.x--;  break;
        case "L":
            cur.y--; break;
        case "R":
            cur.y++;   break;
        case "D":
            cur.x++;   break;
        }
```

```
        if(cur.x<0 || cur.x>= rows || cur.y<0 || cur.y>=cols) return -1;


        for(int i=1;i<snake.size()-1;i++){
            Position next = snake.get(i);
            if(next.isEqual(cur)) return -1;
        }
        snake.addFirst(cur);
        if(len<food.length){
            Position p = new Position(food[len][0],food[len][1]);
            if(cur.isEqual(p)){
                len++;
            }
        }
        while(snake.size()>len+1) snake.removeLast();

        return len;
    }


/**
 * Your SnakeGame object will be instantiated and called as such:
 * SnakeGame obj = new SnakeGame(width, height, food);
 * int param_1 = obj.move(direction);
 */
```

}

written by juanren original link here

Solution 3

Why the output is : [null,0,1,1,1,1,2,2,2,2,3,4,4,4,4,-1] instead of [null,0,1,1,1,1,2,2,2,2,3,4,4,4,4,4] for case: ["SnakeGame","move","move","move","move","move","move","move","move","move","move","move","move","move","move"] [[3,3,[[2,0],[0,0],[0,2],[0,1],[2,2],[0,1]]],["D"],["D"],["R"],["U"],["U"],["L"],["D"],["R"],["R"],["U"],["L"],["L"],["D"],["R"],["U"]]

I think the last return value of move should still be 4, why it is out the boundary??

written by Yun.Hu original link here