프로그래머스 - 캐릭터의 좌표

태그 Math.max(-maxX Math.min(x maxX))

캐릭터의 좌표
문제 설명
머쓱이는 RPG게임을 하고 있습니다. 게임에는 up, down, left, right 방향키가 있으며 각 키를 누르면 위, 아래, 왼쪽, 오른쪽으로 한 칸씩 이동합니다. 예를 들어 [0,0]에서 up을 누른다면 캐릭터의 좌표는 [0, 1], down을 누른다면 [0, -1], left를 누른다면 [-1, 0], right를 누른다면 [1, 0]입니다. 머쓱이가 입력한 방향키의 배열 keyinput와 맵의 크기 board이 매개변수로 주어집니다. 캐릭터는 항상 [0,0]에서 시작할 때 키 입력이 모두 끝난 뒤에 캐릭터의 좌표 [x, y]를 return하도록 solution 함수를 완성해주세요.

[0, 0]은 board의 정 중앙에 위치합니다. 예를 들어 board의 가로 크기가 9라면 캐릭터는 왼쪽으로 최대 [-4, 0]까지 오른쪽으로 최대 [4, 0]까지 이동할 수 있습니다.
제한사항
board은 [가로 크기, 세로 크기] 형태로 주어집니다.
board의 가로 크기와 세로 크기는 홀수입니다.
board의 크기를 벗어난 방향키 입력은 무시합니다.
0 ≤ keyinput의 길이 ≤ 50
1 ≤ board[0] ≤ 99
1 ≤ board[1] ≤ 99
keyinput은 항상 up, down, left, right만 주어집니다

keyinput:["left", "right", "up", "right", "right"] , board:[11, 11] , result:[2, 1]
[0, 0]에서 왼쪽으로 한 칸 오른쪽으로 한 칸 위로 한 칸 오른쪽으로 두 칸 이동한 좌표는 [2, 1]입니다.
keyinput:["down", "down", "down", "down", "down"] , board:[7, 9] , result:[0, -4]
[0, 0]에서 아래로 다섯 칸 이동한 좌표는 [0, -5]이지만 맵의 세로 크기가 9이므로 아래로는 네 칸을 넘어서 이동할 수 없습니다. 따라서 [0, -4]를 return합니다.

```
function solution(keyinput, board) {
  let x = 0;
  let y = 0;
  const maxX = (board[0] - 1) / 2;
  const maxY = (board[1] - 1) / 2;

  for (const key of keyinput) {
    if (key === "up") {
      y++;
    } else if (key === "down") {
      y--;
    } else if (key === "left") {
      x--;
    } else if (key === "right") {
      x++;
    }

    x = Math.max(-maxX, Math.min(x, maxX));
    y = Math.max(-maxY, Math.min(y, maxY));
```

```
  }

  return [x, y];
}
```

보드 주위를 이동할 때 캐릭터의 x 및 y 좌표를 추적합니다. 또한 캐릭터가 도달할 수 있는 최대 x 및 y 좌표를 추적하며 보드 크기를 기준으로 계산됩니다. 각 키 입력에 대해 솔루션은 캐릭터의 좌표를 업데이트한 다음 좌표가 허용 범위 내에 있는지 확인합니다. 좌표가 허용 범위를 벗어나면 허용되는 최대값 또는 최소값으로 설정됩니다. 시간 복잡도는 O(n)입니다.
** Math.max(-maxX, Math.min(x, maxX)) 식은 최소 허용 값과 최대 허용 값 사이에서 x 값을 고정하는 데 사용
표현식의 Math.min(x, maxX) 부분은 x와 maxX의 최소값을 사용하여 x가 maxX보다 크지 않도록 합니다. 표현식의 Math.max(-maxX, ...) 부분은 -maxX의 최대값과 Math.min(x, maxX)의 결과를 사용하여 x가 -maxX보다 작지 않도록 합니다. -maxX와 maxX 사이의 x 값을 고정하므로 x는 항상 허용 범위 내에 있습니다

```
const CONTROL = {
    up: [0, 1],
    down: [0, -1],
    left: [-1, 0],
    right: [1, 0],
}

function solution(key, [n, m]) {
    const [x1, x2] = [-(n-1)/2, (n-1)/2];
    const [y1, y2] = [-(m-1)/2, (m-1)/2];
    return key.reduce(([x, y], k) => {
        const [nx, ny] = [x + CONTROL[k][0], y + CONTROL[k][1]];
        if (x1 <= nx && nx <= x2 && y1 <= ny && ny <= y2) return [nx, ny];
        return [x, y];
    }, [0, 0]);
}
```

CONTROL이라는 개체를 사용하여 각 방향에 대한 x 및 y 좌표의 변경 사항을 저장합니다. 그런 다음 키 배열에서 reduce 메서드를 사용하여 키를 반복하고 캐릭터의 좌표를 업데이트합니다. 업데이트된 좌표가 허용 범위(x1, x2, y1 및 y2로 정의됨) 내에 있는지 확인하고 해당하는 경우 업데이트된 좌표를 반환합니다. 업데이트된 좌표가 허용 범위를 벗어나면 원래 좌표를 반환합니다. 시간 복잡도는 O(n)입니다.

```
class Solution {
    public int[] solution(String[] keyinput, int[] board) {
        int[] now = {0, 0};
        for (int i = 0; i < keyinput.length; i++){
            if(keyinput[i].equals("left")) now[0] -= now[0]>-(board[0]/2)?1:0;
            else if(keyinput[i].equals("right")) now[0] += now[0]<(board[0]/2)?1:0;
            else if(keyinput[i].equals("down")) now[1] -= now[1]>-(board[1]/2)?1:0;
            else if(keyinput[i].equals("up")) now[1] += now[1]<(board[1]/2)?1:0;
            }
        return now;
```

```
    }
}
```

```
import java.lang.Math;
class Solution {
    private String up = "up";
    private String down = "down";
    private String left = "left";
    private String right = "right";
    private int xPos = 0;
    private int yPos = 1;
    private int maxWidth = 0;
    private int maxHeight= 0;
    int[] answer = {0, 0};

    public int[] solution(String[] keyinput, int[] board) {
        maxWidth = board[xPos] / 2;
        maxHeight = board[yPos] / 2;
        for (String moveKey : keyinput) {
            move(moveKey);
        }
        return answer;
    }

    private void move(String key) {
        if (up.equals(key)) {
            // up
            moveUp();
        } else if (down.equals(key)) {
            // down
            moveDown();
        } else if (left.equals(key)) {
            // left
            moveLeft();
        } else {
            // right
            moveRight();
        }
    }

    private void moveUp() {
        int top = Math.abs(maxHeight);
        if (answer[yPos] + 1 <= top) {
            answer[yPos]++;
        };
    }

    private void moveDown() {
        int bottom = Math.abs(maxHeight) * -1;
        if (answer[yPos] - 1 >= bottom) {
            answer[yPos]--;
        };
    }
```

```
    private void moveLeft() {
        int left = Math.abs(maxWidth) * -1;
        if (answer[xPos] - 1 >= left) {
            answer[xPos]--;
        };
    }

    private void moveRight() {
        int right = Math.abs(maxWidth);
        if (answer[xPos] + 1 <= right) {
            answer[xPos]++;
        };
    }
}
```