Programmers_77485

2021 Dev-Matching: 웹 백엔드 개발자 (상반기)_행렬 테두리 회전하기

문제 설명

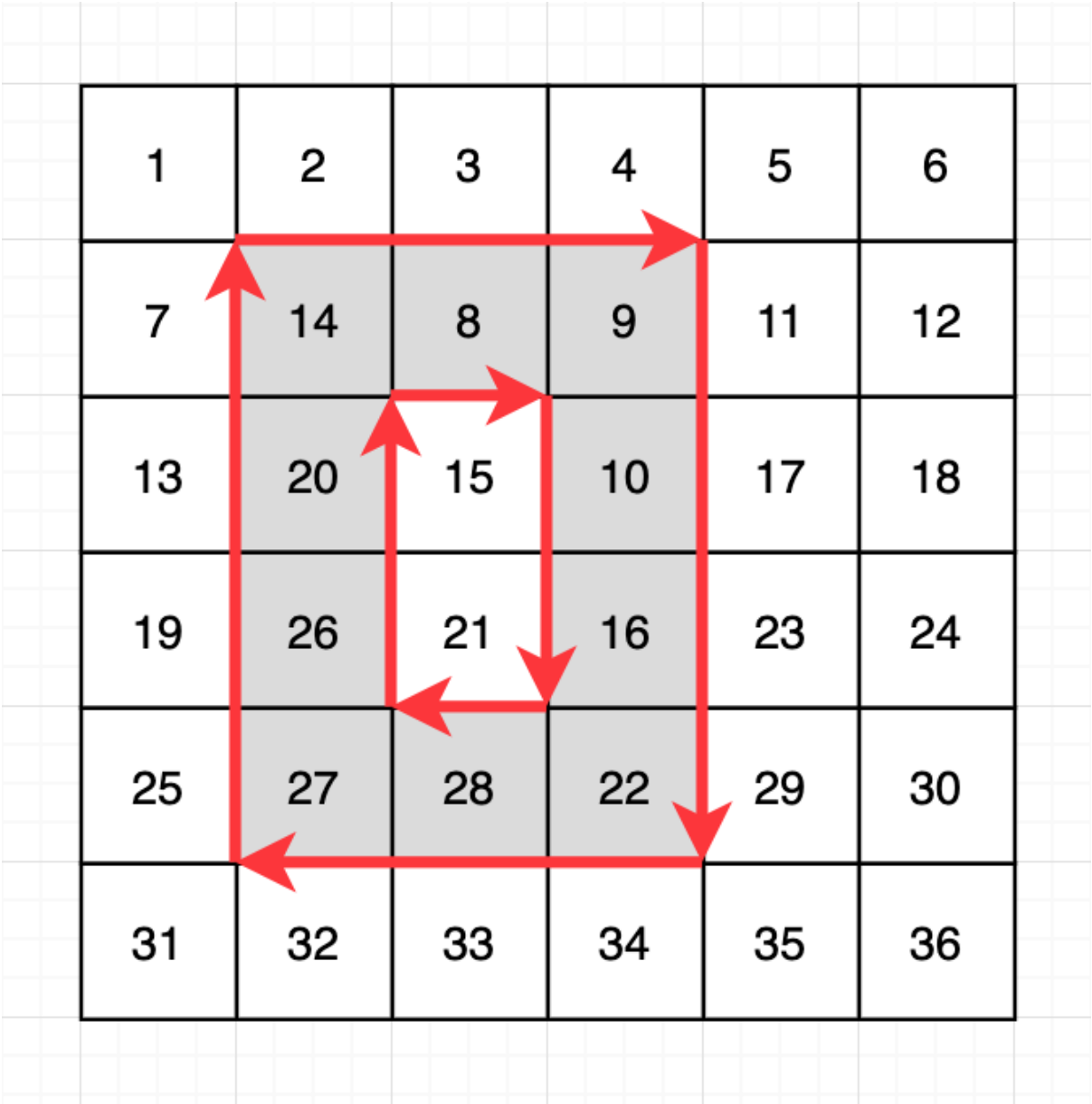
rows x columns 크기인 행렬이 있습니다. 행렬에는 1부터 rows x columns까지의 숫자가 한 줄씩 순서대로 적혀있습니다. 이 행렬에서 직사각형 모양의 범위를 여러 번 선택해, 테두리 부분에 있는 숫자들을 시계방향으로 회전시키려 합니다. 각 회전은 (x1, y1, x2, y2)인 정수 4개로 표현하며, 그 의미는 다음과 같습니다.

- x1 행 y1 열부터 x2 행 y2 열까지의 영역에 해당하는 직사각형에서 테두리에 있는 숫자들을 한 칸씩 시계방향으로 회전합니다.

다음은 6 x 6 크기 행렬의 예시입니다.

1	2	3	4	5	6
7	8	9	10	11	12
13	14	15	16	17	18
19	20	21	22	23	24
25	26	27	28	29	30
31	32	33	34	35	36

이 행렬에 (2, 2, 5, 4) 회전을 적용하면, 아래 그림과 같이 2행 2열부터 5행 4열까지 영역의 테두리가 시계방향으로 회전합니다. 이때, 중앙의 15와 21이 있는 영역은 회전하지 않는 것을 주의하세요.



행렬의 세로 길이(행 개수) rows, 가로 길이(열 개수) columns, 그리고 회전들의 목록 queries가 주어질 때, 각 회전들을 배열에 적용한 뒤, 그 회전에 의해 위치가 바뀐 숫자들 중 **가장 작은 숫자들을 순서대로 배열에 담아** return 하도록 solution 함수를 완성해주세요.

제한사항

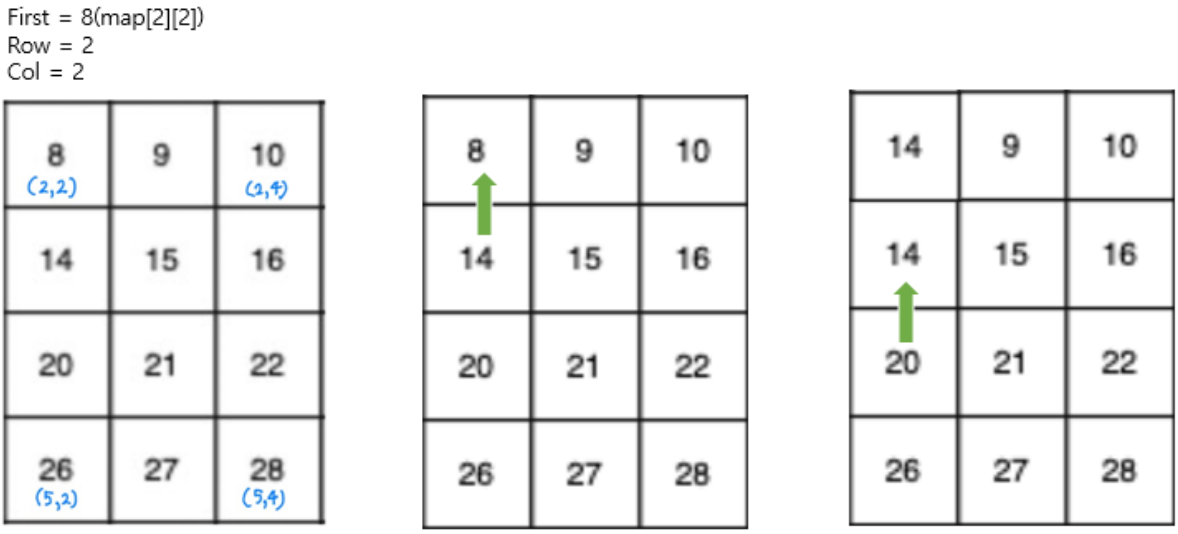
- rows는 2 이상 100 이하인 자연수입니다.
- columns는 2 이상 100 이하인 자연수입니다.
- 처음에 행렬에는 가로 방향으로 숫자가 1부터 하나씩 증가하면서 적혀있습니다.
  - 즉, 아무 회전도 하지 않았을 때, i 행 j 열에 있는 숫자는 ((i-1) x columns + j)입니다.

- queries의 행의 개수(회전의 개수)는 1 이상 10,000 이하입니다.
- queries의 각 행은 4개의 정수 [x1, y1, x2, y2]입니다.
  - x1 행 y1 열부터 x2 행 y2 열까지 영역의 테두리를 시계방향으로 회전한다는 뜻입니다.
  - 1 ≤ x1 < x2 ≤ rows, 1 ≤ y1 < y2 ≤ columns입니다.
  - 모든 회전은 순서대로 이루어집니다.
  - 예를 들어, 두 번째 회전에 대한 답은 첫 번째 회전을 실행한 다음, 그 상태에서 두 번째 회전을 실행했을 때 이동한 숫자 중 최솟값을 구하면 됩니다.

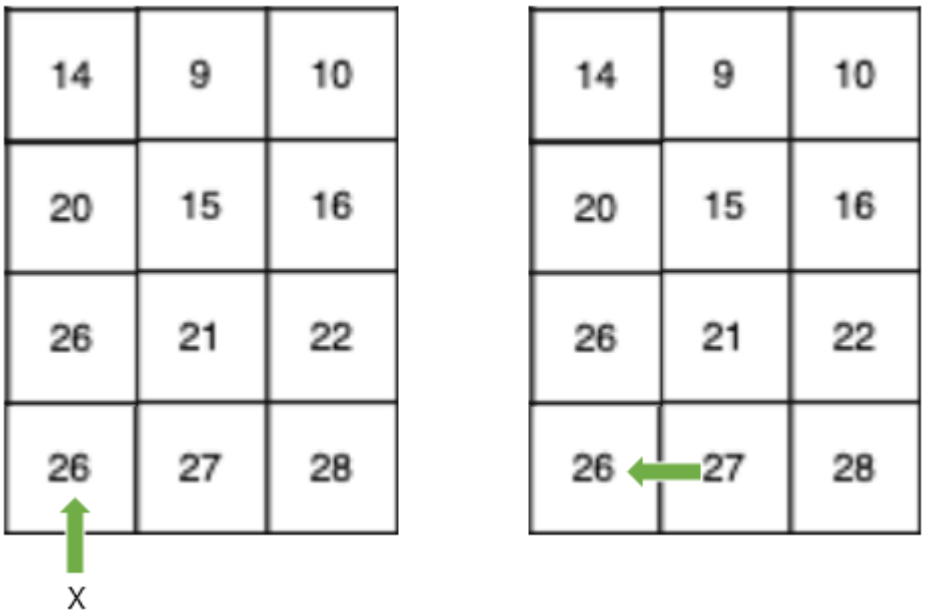
풀이

먼저 회전하는 부분 중에서 좌측 상단의 수를 first로 지정한다.

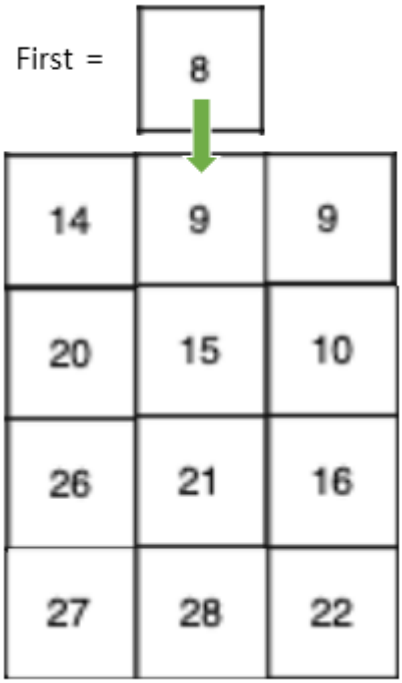
다음으로 좌측 상단을 기준(row,col)으로 아래칸을 탐색한다. 탐색하려는 칸(r,c)이 범위 안에 있다면 탐색하려는 칸의 값을 기준 칸에 넣어준다. 그리고 기준 칸을 변경해 주고(row←r, col←c) 최속값을 갱신해주고 다음 탐색을 한다.



만약 범위를 벗어나면 방향을 바꾸어준다.



탐색하려는 칸이 처음 위치로 돌아오면 탐색을 끝낸다. 그리고 저장해 놨던 수를 한칸 오른쪽 칸에 넣어준다.



코드

```
// Programmers_77485_2021 Dev-Matching: 웹 백엔드 개발자 (상반기)_행렬 테두리 회전하기

import java.util.Arrays;

public class Programmers_77485 {
    public static void main(String[] args) {
        int rows = 6;
        int columns = 6;
        int[][] queries = {{2,2,5,4},{3,3,6,6},{5,1,6,3}};
        Solution sol = new Solution();
        System.out.println(Arrays.toString(sol.solution(rows,columns, queries)));
    }

    static public class Solution {
        int[] dx ={1,0,-1,0};
        int[] dy ={0,1,0,-1};
        public int[] solution(int rows, int columns, int[][] queries) {
            int[] answer = new int [queries.length];
            int[][] map = new int[rows+1][columns+1];
            int count=1;
            for(int i=1;i<=rows;i++){
                for(int j=1;j<=columns;j++){
                    map[i][j]=count++; // 외쪽 상단부터 1씩 증가해주며 값을 넣어준다.
                }
            }
            for(int t=0;t<queries.length;t++){
                int x1 =queries[t][0];
                int y1 =queries[t][1];
                int x2 =queries[t][2];
                int y2 =queries[t][3];
                int row = x1; // 초기 커서의 행
                int col = y1; // 초기 커서의 행
                int dir = 0; // 초기 커서 가장 먼저 아래로 이동을 한다.
                int first = map[x1][y1]; // 가장 왼쪽 상단의 수를 저장한다.
                int min = map[x1][y1];
                while (true){
                    int r = row + dx[dir];
                    int c = col + dy[dir];
                    if(r==x1&&c==y1) break;
                    if(r >=x1&&r<=x2&&c>=y1&&c<=y2){ // 범위 안이라면
                        min=Math.min(min,map[r][c]); // 값을 비교해주며 최속값을 구해준다.
                        map[row][col]=map[r][c]; // 이전위치 현재위치의 값을 넣어준다.
                        row=r; //다음 칸을 탐색하기위해 위치를 변경한다.
                        col=c;
                    }else { // 범위를 벗어나면 방향을 바꾸어준다.
                        dir++;
                    }
                }
                map[x1][y1+1]=first; // 저장해 놨던 수를 한칸 오른쪽 칸에 넣어준다.
//                for(int i=1;i<=rows;i++){
//                    for(int j=1;j<=columns;j++){
//                        System.out.print(map[i][j]+"\t");
//                    }
//                    System.out.println();
//                }
//                System.out.println();
```

```
                answer[t]=min;
            }
            return answer;
        }
    }
}
```