BOJ 1726 로봇 심층 분석

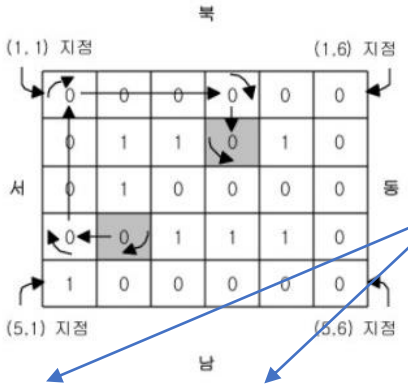
문제

많은 공장에서 로봇이 이용되고 있다. 우리 월드 공장의 로봇은 바라보는 방향으로 궤도를 따라 움직이며, 움직이는 방향은 동, 서, 남, 북 가운데 하나이다. 로봇의 이동을 제어하는 명령어는 다음과 같이 두 가지이다.

- 명령 1. Go k: k는 1, 2 또는 3일 수 있다. 현재 향하고 있는 방향으로 k칸 만큼 움직인다.
- 명령 2. Turn dir: dir은 left 또는 right 이며, 각각 왼쪽 또는 오른쪽으로 90° 회전한다.

구현해야 할 규칙
(문구 그대로)

공장 내 궤도가 설치되어 있는 상태가 아래와 같이 0과 1로 이루어진 직사각형 모양으로 로봇에게 입력된다. 0은 궤도가 깔려 있어 로봇이 갈 수 있는 지점이고, 1은 궤도가 없어 로봇이 갈 수 없는 지점이다. 로봇이 (4, 2) 지점에서 남쪽을 향하고 있을 때, 이 로봇을 (2, 4) 지점에서 동쪽으로 향하도록 이동시키는 것은 아래와 같이 9번의 명령으로 가능하다.



visited[][][] 3차원 방문 배열 필요
(행, 열, 방향 정보)

로봇의 현재 위치와 바라보는 방향이 주어졌을 때, 로봇을 원하는 위치로 이동시키고, 원하는 방향으로 바라보도록 하는데 최소 몇 번의 명령이 필요한지 구하는 프로그램을 작성하시오.

구하고자 하는 것 => 도착 지점까지의 최소 명령횟수(==너비) => BFS

입력

첫째 줄에 공장 내 궤도 설치 상태를 나타내는 직사각형의 세로 길이 M과 가로 길이 N이 빈칸을 사이에 두고 주어진다. 이때 M과 N은 둘 다 100이하의 자연수이다. 이어 M줄에 걸쳐 한 줄에 N개씩 각 지점의 궤도 설치 상태를 나타내는 숫자 0 또는 1이 빈칸을 사이에 두고 주어진다. 다음 줄에는 로봇의 출발 지점의 위치 (행과 열의 번호)와 바라보는 방향이 빈칸을 사이에 두고 주어진다. 마지막 줄에는 로봇의 도착 지점의 위치 (행과 열의 번호)와 바라보는 방향이 빈칸을 사이에 두고 주어진다. 방향은 동쪽이 1, 서쪽이 2, 남쪽이 3, 북쪽이 4로 주어진다. 출발지점에서 도착지점까지는 항상 이동이 가능하다.

입력 조건 : 동(1), 서(2), 남(3), 북(4)

출력

첫째 줄에 로봇을 도착 지점에 원하는 방향으로 이동시키는데 필요한 최소 명령 횟수를 출력한다.


0 0 0 0 0 0
0 1 1 0 1 0
0 1 0 0 0 0
0 0 1 1 1 0
1 0 0 0 0 0
위치정보 유지 & 방향정보 갱신
visited[r][c][right]=true
visited[r][c][left]
현재방향
visited[nr][nc][dir] = true
위치정보 갱신 & 방향정보 유지

- 명령 1. Go k: k는 1, 2 또는 3일 수 있다. 현재 향하고 있는 방향으로 k칸 만큼 움직인다.

```java
// case 1. 현재 로봇이 향하는 방향으로 움직이면 최대 3번 이동
for (int i = 1; i <= 3; i++) {
    int nr = r + (dr[dir-1] * i);
    int nc = c + (dc[dir-1] * i);

    if(nr <= 0 || nc <= 0 || nr > M || nc > N) continue; // map 경계 체크 (예외처리)


    // 다음 이동 방향이 1이면 for문 탈출(연속적으로 움직일 수 없으므로)
    if(map[nr][nc] == 1) break; // 예외처리

    // 방문한 곳이면 continue(지나갈 수는 있음), 방문하지 않은 곳이면 queue에 담기
    if(visited[nr][nc][dir]) continue; // 예외처리
    else {
        visited[nr][nc][dir] = true; // 방문 체크
        queue.add(new int[] {nr, nc, dir, cnt+1});
    }
}
```

현재 방향으로 1, 2, 3칸을 이동하는 반복문 코드블럭

최대한 else문을 줄이고 예외처리 방식으로 continue, break를 사용하여 코드 간결성 확보

- 명령 2. Turn dir: dir은 left 또는 right 이며, 각각 왼쪽 또는 오른쪽으로 90° 회전한다.

```java
// case 2. 현재 로봇 위치에서 방향만 좌우로 틀기

// (1) 현재 이동 방향에서의 좌우 방향 인덱스 구하기
int right = 0, left = 0;
switch(dir) {
    case 1: left = 4; right = 3; break; // 동
    case 2: left = 3; right = 4; break; // 서
    case 3: left = 1; right = 2; break; // 남
    case 4: left = 2; right = 1; break; // 북
}

// (2) 현재 위치에서 방향만 좌우로 틀었을 때, 해당 방향으로 현재 위치를 방문하지 않았다면 의 좌표값을 큐에 담기
if(!visited[r][c][left]) {
    visited[r][c][left] = true;
    queue.add(new int[] {r, c, left, cnt+1});
}
if(!visited[r][c][right]) {
    visited[r][c][right] = true;
    queue.add(new int[] {r, c, right, cnt+1});
}
```

현재 방향에 따라 좌, 우 방향이 달라지므로
현재 방향이 동, 서, 남, 북인지에 따라
현재 방향의 좌, 우의 방향 인덱스를 결정함
(코드 간결성을 위해 switch문 활용)

```java
private static int N;
private static int[] dr = {1,-1, 0, 0};
private static int[] dc = {0, 0, 1,-1};
public static void generalBFS() {
    // 큐와 방문체크 배열 생성
    LinkedList<int[]> queue = new LinkedList<>();
    boolean[][] visited = new boolean[N][N];

    // 탐색 시작 좌표를 queue에 담고 방문처리
    queue.offer(new int[] {0, 0});
    visited[0][0] = true;

    // bfs 탐색
    while(queue.isEmpty()) {
        int[] current = queue.poll();

        /*
         * bfs 탐색의 종료 조건이 있다면 이 위치에 들어감
         */

        // 4방 탐색
        for (int i = 0; i < 4; i++) {
            int nr = current[0] + dr[i];
            int nc = current[1] + dc[i];
            // 방문하지 않은 곳이라면 큐에 담고 방문처리
            if(0<=nr && nr<N && 0<=nc && nc<N && !visited[nr][nc]) {
                queue.offer(new int[] {nr,nc});
                visited[nr][nc] = true;
            }
        }
    }

} // end of method bfs
```

일반적인 BFS 틀
(**BFS 탐색 종료 조건 위치**)

```java
// 현재 위치에서 end 정보를 모두 만족한다면 현재까지 명령 횟수 반환하고 탐색 종료
if(r == end[0] && c == end[1] && dir == end[2]) {
    command = cnt;
    return;
}
```

종료 조건

결론

1. 구현해야 할 핵심 규칙을 **면밀히 분석**하여 **그대로 구현**하기

Ex) 현재 방향으로 최대 3번 이동 => 1을 만나면 break을 통해 바로 for문 탈출

2. 최대한 **구현량을 줄이기 위한 방법 고민하기**

Ex) 현재 방향에 따라 좌, 우 방향 인덱스를 정하는 방법을 switch문을 통해 구현량을 줄임

3. 최대한 **중복 제거 & 모듈화**

Ex) 명령1, 명령2, 종료조건 모듈화