Chapter 06.
파트별 쪽지시험 #3

Clip 01 | [2998] 8진수
재귀

Clip 02 | [1213] 팰린드롬 만들기
구현

Clip 03 | [17225] 세훈이의 선물가게
구현

Clip 04 | [1707] 이분 그래프
탐색

Clip 05 | [1726] 로봇
탐색

Ch06. 파트별 쪽지시험 #3

1. [2998] 8진수

BOJ2998: 8진수

문제 요약

- 2진수를 8진법으로 변환
- 단, 수를 3개의 숫자단위로 끊어서 8진수를 계산한다
  - 3으로 나누어질 때까지 앞자리에 숫자 0을 붙인다

BOJ2998: 8진수

문제 요약

000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7

입력 데이터
11001100
011001100

출력 데이터
314

BOJ2998: 8진수

구현

```
String binary = sc.next();
if(binary.length() % 3 == 1)
    binary = "00" + binary;
else if(binary.length() % 3 == 2)
    binary = "0" + binary;
```

입력을 받고 길이가 3으로 나누어 떨어지도록 0을 붙여준다

BOJ2998: 8진수

문제 분석

- 길이가 3으로 나누어 떨어지도록 처리가 끝났다면 진법 변환하는 함수는 어떻게 작성할 수 있을까?

- change("011001100")
  - change("011001") + print("100")
    - change("011") + print("001")
      - print("011")

BOJ2998: 8진수

구현

```
public static int o2d(String binary) {
    final String[] octal = {
        "000", "001", "010",
        "011", "100", "101",
        "110", "111"
    };
    for (int i = 0; i < 8; i++) {
        if(binary.equals(octal[i])) return i;
    }
    return -1;
}
```

8진수를 문자열로
하드코딩하고

매칭되는 문자열이
들어오면 10진수로
변환한다

BOJ2998: 8진수

구현

```
public static void change (String binary) {
    int length = binary.length();
    if(length == 0) return;

    change(binary.substring(0, length - 3));
    int d = o2d(binary.substring(length - 3));
    System.out.print(d);
}
```

substring()으로 3글자를 떼서, 재귀함수와 출력을 호출

Ch06. 파트별 쪽지시험 #3

2. [1213] 팰린드롬 만들기

BOJ1213: 팰린드롬 만들기

문제 요약

- 입력 받은 알파벳을 이용해 팰린드롬(대칭)을 만들어야 한다
  - (length <= 50, 알파벳 대문자)
- 정답이 여러 개가 만들어 질 수 있으므로 사전순으로 1개 출력
- 만들 수 없다면 “I’m Sorry Hansoo”를 출력

BOJ1213: 팰린드롬 만들기

문제 요약

입력 데이터
AABB

출력 데이터
ABBA

입력 데이터
AAABB

출력 데이터
ABABA

입력 데이터
ABCD

출력 데이터
I’m Sorry hansoo

BOJ1213: 팰린드롬 만들기

문제 분석

- 팰린드롬이 되기 위해서는 입력이 어떻게 주어져야 할까?
- 문자열 내 문자의 개수가 각각 짝수여야 한다
  - 단, 전체 길이가 홀수라면, 1개는 홀수일 수 있다
- 사전순으로 출력하도록 낮은 인덱스부터 고른다
  - 홀수가 1개이면 해당 알파벳만 중앙에 배치한다
  - 홀수가 2개 이상이면 팰린드롬 생성이 불가능하다

BOJ1213: 팰린드롬 만들기

구현

1. 입력 받고 문자 개수 카운트

```
String name = sc.next();
int[] cnt = new int[26];
for(int i = 0; i < name.length(); i++) {
    cnt[name.charAt(i) - 'A']++;
}
```

BOJ1213: 팰린드롬 만들기

구현

2. 홀수 문자 카운트, 2개 이상이면 종료

```
int odd = 0;
int oddIdx = -1;
for(int i = 0; i < 26; i++) {
    if(cnt[i] % 2 == 1) {
        odd++;
        oddIdx = i;
    }
}
```

```
if(odd > 1) {
    System.out.println("I'm Sorry Hansoo");
    return;
}
```

BOJ1213: 팰린드롬 만들기

구현

3. {문자 개수} / 2개만큼 문자를 사전순으로 선택

```
StringBuilder sb = new StringBuilder();
for(int i = 0; i < 26; i++) {
    for(int j = 0; j < cnt[i] / 2; j++) {
        sb.append((char)(i + 'A'));
    }
}
```

BOJ1213: 팰린드롬 만들기

구현

4.

(3) 에서 생성한 문자열을 ans로 이동
홀수개인 문자가 1개 있었다면 그 뒤에 추가
(3) 에서 생성한 문자열을 뒤집어서 뒤에 추가

```
StringBuilder ans = new StringBuilder(sb);
if(oddIdx != -1) {
    ans.append((char)(oddIdx + 'A'));
}
ans.append(sb.reverse());
```

Ch06. 파트별 쪽지시험 #3

3. [17225] 세훈이의 선물가게

BOJ17225: 세훈이의 선물가게

문제 요약

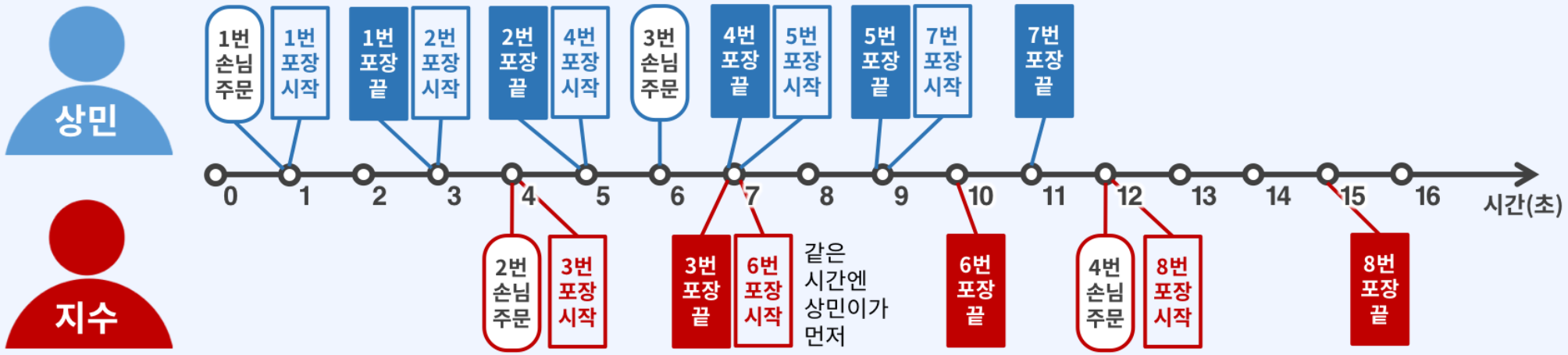
- 상민이와 지수가 각각 파랑 / 빨강색으로 선물을 포장
  - 같은 시각에 주문이 오면 파랑을 먼저 선택
- 각각 선물을 포장하는데 걸리는 시간이 있음 (0도 가능)
- 손님 / 색상 / 개수가 N개 라인에 걸쳐 입력됨
  - (1 <= N <= 1,000)
- 누가 어떤 선물을 포장했는지 출력

BOJ17225: 세훈이의 선물가게

문제 요약

입력 데이터
2 3 4 1 B 3 4 R 2 6 B 2 12 R 1

출력 데이터
5 1 2 4 5 7 3 3 6 8



BOJ17225: 세훈이의 선물가게

문제 요약

- 특정 선물을 언제 포장을 시작하고 종료할 건지
  데이터 가공이 필요하다

- 두개의 큐 {blueQueue}, {redQueue} 를 생성하고,
  작업 시작 시각을 기준으로 데이터를 넣어보자

  단, 큐의 데이터를 활용해, 아직 일이 끝나지 않았는데
  주문이 들어오면 작업 중인 일이 끝난 뒤로 시각을 배정

BOJ17225: 세훈이의 선물가게

문제 요약

- 두개의 큐 {blueQueue}, {redQueue} 를 생성하고,
  작업 시작 시각을 기준으로 데이터를 넣어보자

  단, 큐의 데이터를 활용해, 아직 일이 끝나지 않았는데 주문이 들어오면
  작업 중인 일이 끝난 뒤로 시각을 배정

A == 2
1 B 3
6 B 2

BOJ17225: 세훈이의 선물가게

문제 요약

- 두개의 큐 {blueQueue}, {redQueue} 를 생성하고,
  작업 시작 시각을 기준으로 데이터를 넣어보자

  단, 큐의 데이터를 활용해, 아직 일이 끝나지 않았는데 주문이 들어오면
  작업 중인 일이 끝난 뒤로 시각을 배정

A == 2
1 B 3
6 B 2

1	3	5	7	

손님은 {시각 6}에 왔지만, {시각 5}에 시작한 일은 {시각 7}에 끝난다
따라서 두번째 손님은 {시각 7}부터 시작한다

BOJ17225: 세훈이의 선물가게

문제 요약

- 두개의 큐 {blueQueue}, {redQueue} 를 생성하고,
  작업 시작 시각을 기준으로 데이터를 넣어보자

  단, 큐의 데이터를 활용해, 아직 일이 끝나지 않았는데 주문이 들어오면
  작업 중인 일이 끝난 뒤로 시각을 배정

A == 2
1 B 3
6 B 2

1	3	5	7	9

손님은 {시각 6}에 왔지만, {시각 5}에 시작한 일은 {시각 7}에 끝난다
따라서 두번째 손님은 {시각 7}부터 시작한다

BOJ17225: 세훈이의 선물가게

문제 요약

- {blueQueue}와 {redQueue}의 front를 바라보며 시작 시각이 작은 값의 일을 먼저 꺼내서 순서를 배정한다

- 둘중 하나의 큐가 비면, 나머지 큐의 일을 모두 뽑아서 배정한다
  - (분할정복의 conquer 과정과 비슷하다)

BOJ17225: 세훈이의 선물가게

구현

```
Deque<Integer> blueQueue = new LinkedList<>();
Deque<Integer> redQueue = new LinkedList<>();

for (int i = 0; i < n; i++) {
    int time = sc.nextInt();
    char color = sc.next().charAt(0);
    int cnt = sc.nextInt();

    // ...
```

주문을 입력 받으며 {blueQueue}, {redQueue}에 가공해 넣는다
* Deque: 원소를 앞, 뒤로 뽑을 수 있는 큐

BOJ17225: 세훈이의 선물가게

구현

```
case 'B' -> {
    if (!blueQueue.isEmpty() && blueQueue.getLast() + bTime > time) {
        time = blueQueue.getLast() + bTime;
    }
    for (int j = 0; j < cnt; j++) {
        blueQueue.offer(time);
        time += bTime;
    }
}
```

아직 일이 끝나지 않았는데 주문이 들어오면
작업 중인 일이 끝난 뒤로 시각을 배정

큐에 포장을 시작하는 시각을 기준으로 넣는다
작업에는 {bTime}이 소요되므로 시간을 증가시킨다

BOJ17225: 세훈이의 선물가게

구현

```
case 'R' -> {
    if (!redQueue.isEmpty() && redQueue.getLast() + rTime > time) {
        time = redQueue.getLast() + rTime;
    }
    for (int j = 0; j < cnt; j++) {
        redQueue.offer(time);
        time += rTime;
    }
}
```

redQueue도 동일한 조건으로 구현

BOJ17225: 세훈이의 선물가게

구현

```
while (!blueQueue.isEmpty() && !redQueue.isEmpty()) {
    if(blueQueue.peek() <= redQueue.peek()) {
        blueOrder.add(++orderNumber);
        blueQueue.poll();
    }
    else {
        redOrder.add(++orderNumber);
        redQueue.poll();
    }
}
```

두개의 큐의 front중에 시각이 더 빠른 일을 뽑는다

BOJ17225: 세훈이의 선물가게

구현

```
while(!blueQueue.isEmpty()) {
    blueOrder.add(++orderNumber);
    blueQueue.poll();
}

while(!redQueue.isEmpty()) {
    redOrder.add(++orderNumber);
    redQueue.poll();
}
```

blue / red 큐에 남은 잔여 선물을 뽑는다

Ch06. 파트별 쪽지시험 #3

4. [1707] 이분 그래프

BOJ1707: 이분 그래프

문제 요약

- 서로 인접하지 않은 영역으로 그래프를 분할 가능하면
  이분 그래프라고 정의함

- 이분 그래프인지 아닌지 판별하는 문제

- 케이스 개수: (2 <= K <= 5)
- 정점의 개수: (1 <= V <= 20,000)
- 간선의 개수: (1 <= E <= 200,000)

- 케이스 별로 그래프가 주어지며
  각 그래프가 이분그래프인지 판별

BOJ1707: 이분 그래프

문제 요약

- 케이스
- 정점 간선
- 간선의 정보

입력 데이터
2 3 2 1 3 2 3 4 4 1 2 2 3 3 4 4 2

출력 데이터
YES NO

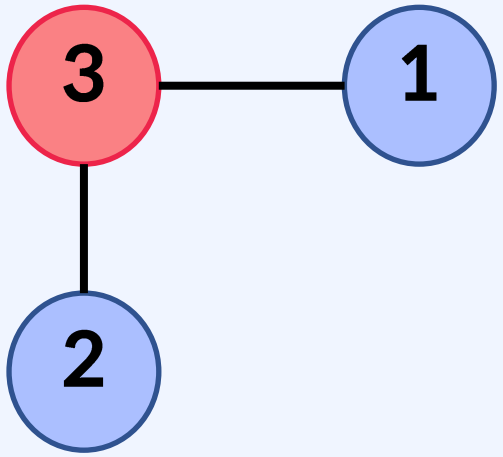
BOJ1707: 이분 그래프

문제 분석

case1

```
3 2
1 3
2 3
```

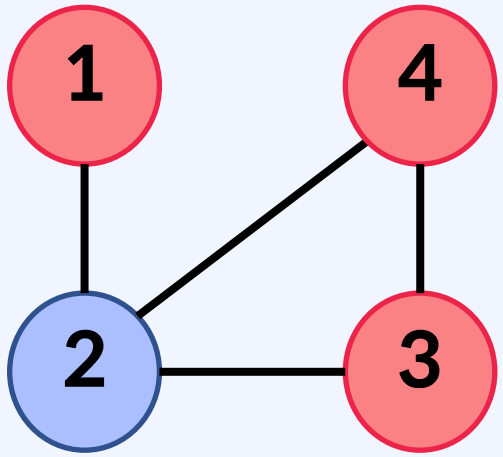
빨강과 파랑 집합
둘로 나눌 수 있다



case2

```
4 4
1 2
2 3
3 4
4 2
```

2, 3, 4가 서로 인접하여
둘로 나눌 수 없다



BOJ1707: 이분 그래프

문제 분석

- 한 정점에 연결 되어있는 모든 정점은
  다른 영역에 있어야 한다
  - 영역을 빨강과 파랑으로 나누어 생각해보자

- 빨간색 정점은 파란색 정점과만 연결될 수 있다
- 파란색 정점은 빨간색 정점과만 연결될 수 있다

1. 정점을 색칠하는 방법은?
2. 연결 여부를 파악하는 방법은?

BOJ1707: 이분 그래프

문제 분석

1. 정점을 색칠하는 방법은?

   그래프를 탐색하면서 (BFS / DFS)

   - 홀수 번째 깊이는 RED
   - 짝수 번째 깊이는 BLUE

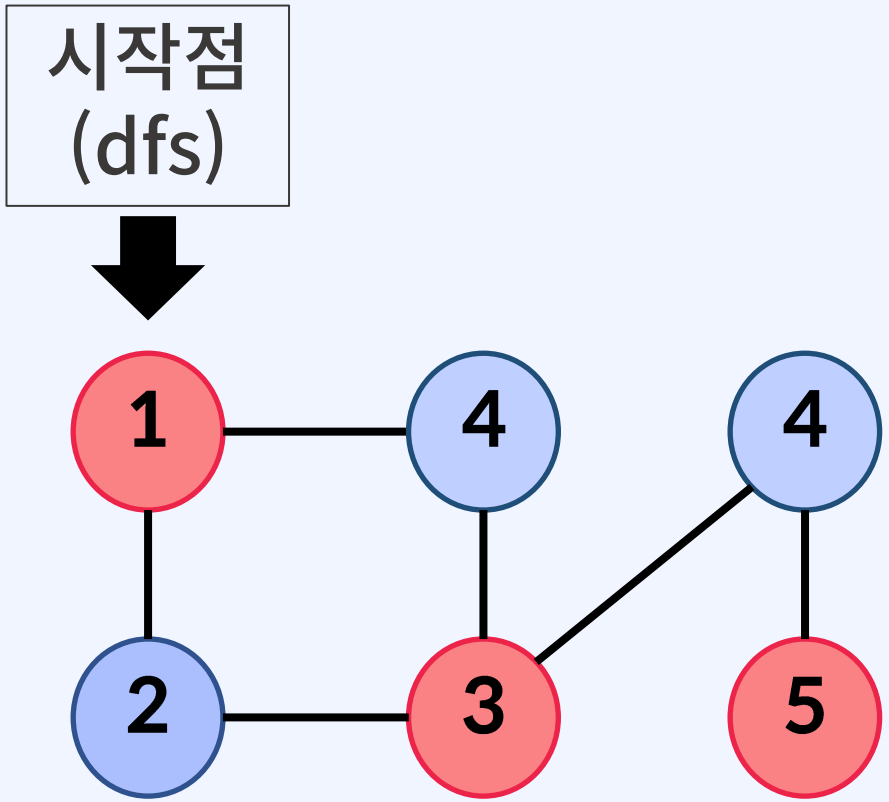
   로 취급하고 정점에 마킹을 해보자

BOJ1707: 이분 그래프

문제 분석

1. 정점을 색칠하는 방법은?

   홀수 번째 깊이는 RED
   짝수 번째 깊이는 BLUE



BOJ1707: 이분 그래프

문제 분석

2. 연결 여부를 파악하는 방법은?

- 그래프를 다루는 자료구조를 인접리스트로 구현하면 각 정점에 어떤 노드가 연결되어 있는지 알 수

1	B	
2	A	D
3		
4	A	C

- 1번 노드가 RED라면?
  - 연결된 다른 노드는 모두 BLUE여야 한다
- 1번 노드가 BLUE라면?
  - 연결된 다른 노드는 모두 RED여야 한다

BOJ1707: 이분 그래프

문제 분석

- 1번 노드가 RED라면?
  - 연결된 다른 노드는 모두 BLUE여야 한다
- 1번 노드가 BLUE라면?
  - 연결된 다른 노드는 모두 RED여야 한다

정점 번호				
1	2	3	5	7
2	1	4		
3	1			
4	2	7		

BOJ1707: 이분 그래프

문제 분석

- 정답은?
  - 그래프를 탐색하며 깊이의 홀 / 짝을 기준으로 RED / BLUE로 칠한다

  - 인접리스트를 순회하며 {현재 정점}과 연결된 {다른 정점}들이 서로 다른 색인지 확인한다
    - 만약 같은 색이 있다면 순회를 중단하고 “NO”출력

BOJ1707: 이분 그래프

구현

```
for (int i = 0; i < e; i++) {
    int a = sc.nextInt();
    int b = sc.nextInt();
    graph[a].add(b);
    graph[b].add(a);
}

for (int i = 1; i <= v; i++) {
    dfs(i, 0);
}
```

인접리스트를 생성하고
dfs 탐색을 시작한다

간선은 양방향 그래프로
취급

BOJ1707: 이분 그래프

구현

```
static void dfs(int nodeNum, int depth) {
    if (color[nodeNum] != 0) return;
    if (depth % 2 == 0) color[nodeNum] = RED;
    else color[nodeNum] = BLUE;

    for(int next : graph[nodeNum]) {
        dfs(next, depth + 1);
    }
}
```

홀수 depth는 RED
짝수 depth는 BLUE

연결된 정점을
dfs로 탐색

BOJ1707: 이분 그래프

구현

```
boolean isBinary = true;
for (int i = 1; i <= v; i++) {
    for(int next : graph[i]) {
        if (color[i] == color[next]) {
            isBinary = false;
            break;
        }
    }
}
```

인접 리스트를 모든 정점마다 순회
연결된 정점이 서로 같은색이면
이분 그래프가 아니다

Ch06. 파트별 쪽지시험 #3

5. [1726] 로봇

BOJ1726: 로봇

문제 요약

- 로봇의 현재 좌표와 바라보는 방향이 주어짐
- 원하는 위치와 원하는 방향으로 움직일 때 최소 몇번의 명령이 필요한지 출력
- 공장: (1 <= M, M <= 100), 이동가능여부 0 or 1
- 탐색의 종류는 총 5가지
  - 1칸 전진, 2칸 전진, 3칸 전진
  - 좌회전, 우회전

BOJ1726: 로봇

문제 분석

- 회전이 없다면?
  - 일반적인 좌표 탐색(BFS / DFS)으로 풀이가 가능하다
  - 단, 전진을 {1} {2} {3} 칸을 움직이도록 반복문은 추가해야 한다

- 그렇다면 회전을 어떻게 처리해야 할까?

BOJ1726: 로봇

문제 분석

- 거리를 기록하는 배열에 상태를 하나 더 추가한다
  - d[행][열] → d[로봇방향][행][열]
  - 각 좌표별로 로봇이 어디를 보고 있는지 관리

- 현재의 좌표를 기준으로 이동 뿐만 아니라 회전하는 케이스를 큐에 담는다
  - 어떻게?

BOJ1726: 로봇

문제 분석

- 로봇이 바라볼 수 있는 방향은 4종류이다
  - EAST, WEST, SOUTH, NORTH

- 로봇이 동 방향을 바라보고 있는 경우
  - 다음 방향이 북 / 남 방향이면 1번 회전으로 취급
  - 다음 방향이 서 방향이면 2번 회전으로 취급
  - 다음 방향이 동 방향이면 무시(continue)

BOJ1726: 로봇

문제 분석

- 로봇이 서 방향을 바라보고 있는 경우
  - 다음 방향이 북 / 남 방향이면 1번 회전으로 취급
  - 다음 방향이 동 방향이면 2번 회전으로 취급
  - 다음 방향이 서 방향이면 무시(continue)

- 로봇이 남 / 북 방향을 바라보고 있는 경우
  - 다음 방향이 동 / 서 방향이면 1번 회전으로 취급
  - 다음 방향이 북 | 남 방향이면 2번 회전으로 취급
  - 다음 방향이 남 | 북 방향이면 무시(continue)

BOJ1726: 로봇

구현 1 / 2 / 3칸 전진, dir 방향 추가

```
while (!q.isEmpty()) {
    Point now = q.poll();
    for (int k = 1; k <= 3; k++) {
        int nr = now.r + dr[now.dir] * k;
        int nc = now.c + dc[now.dir] * k;
        if (!isRange(nr, nc, m, n)) break;
        if (a[nr][nc] == 1) break;
        if (dist[now.dir][nr][nc] != -1) continue;
        dist[now.dir][nr][nc] = dist[now.dir][now.r][now.c] + 1;
        q.add(new Point(nr, nc, now.dir));
    }
```

전진 횟수를 k로 관리

다음 좌표 계산

방문 가능성 판단

다음 탐색 좌표 추가

BOJ1726: 로봇

구현 회전 케이스 추가

```
for (int k = 1; k <= 4; k++) {
    if (k == now.dir) continue;
    if (dist[k][now.r][now.c] != -1) continue;
    int cnt = switch (k) {
        case EAST, WEST -> (now.dir == NORTH || now.dir == SOUTH) ? 1 : 2;
        case SOUTH, NORTH -> (now.dir == EAST || now.dir == WEST) ? 1 : 2;
        default -> 0; // error
    };
    dist[k][now.r][now.c] = dist[now.dir][now.r][now.c] + cnt;
    q.add(new Point(now.r, now.c, k));
}
```

회전 전 / 후 가 같은 방향이면 무시
이미 계산한 적이 있다면 무시

현재 보고있는 방향과 대조하여 회전 횟수 계산

다음 탐색 좌표 추가