

JooHyun – Lee (comkiwer)

TS죄인의 숲

Hancom Education Co. Ltd.

Problem

어떤 한 죄인이 숲에 숨어 있다.

숲은 $N * N$ 크기의 정사각형의 모양을 가진다.

칸의 위치는 (r, c) 로 표시한다.

(r, c) 의 칸은 위에서 아래 방향으로 r 번째이고
왼쪽에서 오른쪽 방향으로 c 번째 있는 칸을 의미한다.

r 과 c 모두 0부터 시작한다.

[Fig. 1]은 $N = 10$ 일 때 숲의 예이다.

	c=0	c=1	c=2	c=3	c=4	c=5	c=6	c=7	c=8	c=9
r=0	1	1	1	1	1	1	1	1	1	1
r=1	1	1	2	1	1	1	1	1	1	1
r=2	1	1	1	1	1	1	1	1	1	2
r=3	1	1	1	1	1	1	1	1	1	1
r=4	1	1	2	1	1	1	1	1	1	2
r=5	1	1	1	1	2	1	1	1	1	2
r=6	1	1	1	2	2	1	1	1	2	1
r=7	1	1	1	1	1	1	1	1	1	1
r=8	1	1	1	1	1	1	1	1	1	1
r=9	1	1	1	2	2	1	1	1	1	1

[Fig. 1]

**또한, 숲의 맨 왼쪽과 맨 오른쪽은 서로 연결되어 있고
맨 위쪽과 맨 아래쪽은 서로 연결되어 있다.**

즉, $(r, 0)$ 의 칸에서 왼쪽으로 한 칸 이동하면 $(r, N - 1)$ 의 칸으로 이동하고
 $(r, N - 1)$ 의 칸에서 오른쪽으로 한 칸 이동하면 $(r, 0)$ 의 칸으로 이동한다.
 $(0, c)$ 의 칸에서 위로 한 칸 이동하면 $(N - 1, c)$ 의 칸으로 이동하고
 $(N - 1, c)$ 의 칸에서 아래로 한 칸 이동하면 $(0, c)$ 의 칸으로 이동한다.

숲의 각 칸마다 나무가 심어져 있다.

나무의 종류는 2가지이고 1과 2로 표시된다.

죄인은 숲 중 어느 한 칸에 있고 경찰은 죄인의 위치를 모른다.

경찰은 가로, 세로 범위가 D 인 탐지기를 이용해서 죄인을 잡으려고 한다.

탐지기는 죄인을 중심으로 $D * D$ 칸의 나무의 종류를 출력한다. 여기서, D 는 홀수인 정수이다.

하지만, 탐지기는 죄인의 위치를 출력하지 않는다.

또한, 탐지기를 사용할 때마다 죄인은 한 칸 이동한다.

엄밀히 말하면 죄인은 경찰이 탐지기를 사용하기 바로 전 한 칸 이동한다.

그 이후 탐지기가 작동한다.

탐지기를 사용하지 않으면 죄인은 이동하지 않는다.

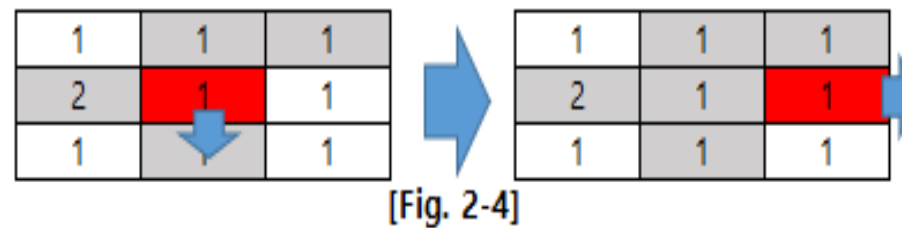
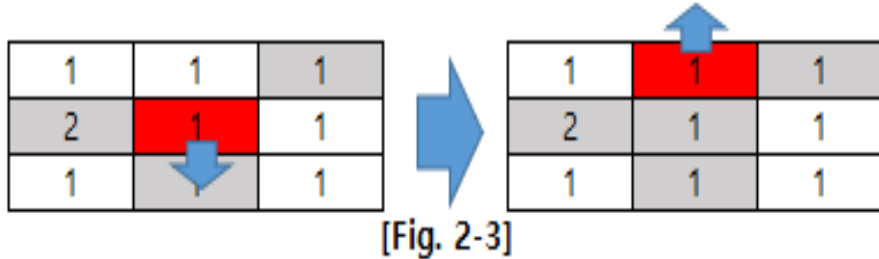
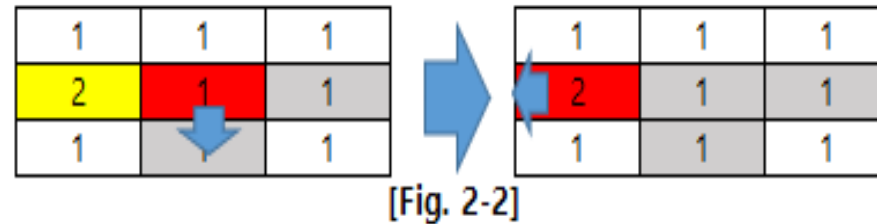
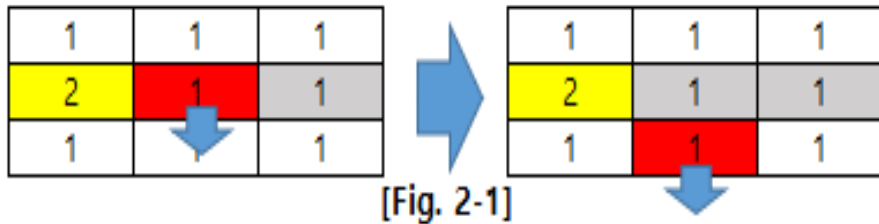
초기에 죄인은 상하좌우 중 한 방향을 정하고 그 방향으로 계속 전진한다.

만약 죄인이 이미 지나간 칸을 지나가려고 하는 경우 지나가지 않는 칸이 나올 때까지

시계 방향으로 계속 회전하고 지나가지 않은 칸이 나오면 그 방향으로 계속 전진한다.

[Fig. 2-1] ~ [Fig. 2-4]는 죄인이 이동하는 예들이다.

그림에서 붉은색은 죄인이 있는 칸이고 회색은 이미 지나간 칸이다.
또한 초기에 죄인의 이동 방향은 아래이다.



탐지기를 사용할 수 있는 기회는 제한되어 있다.

경찰은 제한된 횟수 이하로 탐지기를 이용해서
그 때의 죄인의 위치를 유일하게 특정 시켜야 한다.

주어진 테스트 케이스에서는 제한된 횟수 이내에
탐지기를 사용해서 죄인의 위치를 특정할 수 있음을 보장한다.

또한, 인접한 칸이 모두 이미 지나간 칸이어서
죄인이 이동할 수 없는 경우는 발생하지 않는다.

본 문제는 한 테스트 케이스에서 같은 숲의 정보를 사용하고 죄인의 위치를 K 번 찾는 것으로 구성되어 있다.

테스트 케이스에서 점수를 얻기 위해서는 K 번 모두 제한된 횟수 이하로 탐지기를 사용하여 죄인의 위치를 맞춰야 한다.

아래 API 설명을 참조하여 각 함수를 구현하라.

void init(**int** N, **int** D, **int** mForest[][], **int** K)

테스트 케이스의 시작을 의미하는 함수로 각 테스트 케이스 맨 처음 1회 호출된다.

N은 숲의 한 변의 길이이다. 숲은 정사각형 모양을 가진다.

D는 탐지기를 사용했을 때, 스캔이 되는 범위이다.

D는 언제나 홀수이고 N보다 작다. (Main API 설명 참조)

mForest[r][c]는 (r, c)의 칸에 있는 나무의 종류를 의미하며 1 또는 2 값을 가진다.

해당 함수가 호출된 후 findCriminal() 함수가 K번 호출된다.

Parameters

N : 숲의 한 변의 길이 ($7 \leq N \leq 50$)

D : 탐지기의 탐색 범위 ($3 \leq D \leq 11$, D는 홀수이다)

mForest : 숲 나무의 종류 ($\text{mForest}[r][c] = 1 \text{ or } 2, 0 \leq r, c \leq N - 1$)

K : findCriminal() 함수호출 횟수 ($1 \leq K \leq 5$)

Result findCriminal()

함수가 호출되면 죄인이 지나간 칸들이 모두 초기화되어 지나가지 않은 칸들이 된다.
숲의 임의의 어느 칸에 죄인이 있다.

MainAPI인 useDetector() 함수를 사용하여 죄인의 위치를 특정하고
그 위치를 Result 구조체에 저장하여 반환한다.

죄인이 있는 칸의 위치가 (r, c)이라고 하면 Result 구조체의 r과 c에 각각 저장한다.

Main API인 useDetector() 함수를 호출할 때마다 죄인은 1칸씩 이동함을 명심하라.

useDetector() 함수 호출 횟수가 제한된 사용 횟수를 초과하거나
찾은 위치가 죄인의 현재 위치와 다른 경우 해당 테스트 케이스는 점수를 얻을 수 없다.
제한된 사용 횟수는 "죄인의 위치를 특정 시킬 수 있는 최소 호출 횟수 + 1"이다.

Returns

```
Result {  
    int r, c;    // 죄인의 현재 r,c 위치  
}
```

```
int useDetector(int P[][])
```

죄인이 있는 칸 중심으로 $D * D$ 칸에 있는 나무의 종류를 출력한다.

죄인이 있는 칸의 나무의 종류는 $P[(D - 1) / 2][(D - 1) / 2]$ 에 저장된다.


그 위치를 중심으로 각각 대응되는 $D * D$ 칸의 나무 종류는 $P[0][0] \sim P[D - 1][D - 1]$ 에 저장된다.

아래의 그림은 $D = 3$ 인 경우의 나무의 종류가 $P[][]$ 에 저장되는 예이다. 붉은색 칸에 죄인이 있다.

탐지기의 범위가 숲의 가장자리를 넘어갈 때,

숲은 좌우와 위 아래가 각각 서로 연결되어 있음을 유의하라. ([Fig. 6] 참조)

1	1	2
1	1	1
1	1	2



$P[0][0]$	$P[0][1]$	$P[0][2]$
$P[1][0]$	$P[1][1]$	$P[1][2]$
$P[2][0]$	$P[2][1]$	$P[2][2]$

[Fig. 2-5]

`int useDetector(int P[][])` (계속)

해당 함수의 호출 횟수가 제한된 사용 횟수 이내이면 성공하고 1을 반환한다.
그렇지 않으면 실패하고 0을 반환한다.

실패한 경우 나무의 종류가 출력되지 않는다.

Parameters

P : 죄인이 있는 칸 중심으로 탐색된 나무의 종류

Returns

성공할 경우 1, 실패할 경우 0

[제약사항]

1. 각 테스트 케이스는 맨 처음 `init()` 함수가 1번 호출된다.
2. 숲의 한 변의 길이 N 은 7 이상 50 이하의 정수이다.
3. 탐지기의 탐지 범위 D 는 3 이상 11 이하의 홀수이고 N 보다 작다.
4. 각 테스트 케이스에서 `findCriminal()` 함수 호출 횟수 K 는 1 이상 5 이하이다.
5. `findCriminal()` 함수를 호출할 시 죄인의 위치를 특정할 수 있는 `useDetector()` 함수 호출 횟수는 250 이하임을 보장한다.
6. `useDetector()` 함수를 호출할 때마다 죄인이 있을 수 있는 위치들을 좁혀 나갈 수 있다.
그 가능한 위치의 수가 1일 때 죄인의 위치를 확정하면 점수를 얻을 수 있음을 보장한다.

Problem analysis

[Fig. 3]과 같이 숲 정보가 주어지는 경우를 살펴보자.

N = 10이고 D = 3이고 죄인은 초기에 (3, 6)의 칸에 있고 이동 방향은 아래쪽 방향이다.

Sample TC #1의 첫번째 findCriminal() 함수가 호출된 경우이다.

1	1	1	1	1	1	1	1	1	1
1	1	2	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	2
1	1	1	1	1	1	1	1	1	1
1	1	2	1	1	1	1	1	1	2
1	1	1	1	2	1	1	1	1	2
1	1	1	2	2	1	1	1	2	1
1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1
1	1	1	2	2	1	1	1	1	1

[Fig. 3]

[Fig. 4]는 1번째 useDetector() 함수를 호출한 후 죄인이 이동한 위치를 보여준다.

탐지기로 탐지된 지역은 파란색 테두리 안에 있는 칸들이고 회색 칸은 죄인이 지나온 칸이다.

1	1	1	1	1	1	1	1	1	1
1	1	2	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	2
1	1	1	1	1	1	1	1	1	1
1	1	2	1	1	1	1	1	1	2
1	1	1	1	2	1	1	1	1	2
1	1	1	2	2	1	1	1	2	1
1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1
1	1	1	2	2	1	1	1	1	1

[Fig. 4]

[Fig. 5]는 2번째 useDetector() 함수를 호출한 후 죄인이 이동한 위치를 보여준다.

1	1	1	1	1	1	1	1	1	1
1	1	2	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	2
1	1	1	1	1	1	1	1	1	1
1	1	2	1	1	1	1	1	1	2
1	1	1	1	2	1	1	1	1	2
1	1	1	2	2	1	1	1	2	1
1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1
1	1	1	2	2	1	1	1	1	1

[Fig. 5]

[Fig. 6]은 6번째 useDetector() 함수를 호출한 후 죄인이 이동한 위치를 보여준다.

1	1	1	1	1	1	1	1	1	1
1	1	2	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	2
1	1	1	1	1	1	1	1	1	1
1	1	2	1	1	1	1	1	1	2
1	1	1	1	2	1	1	1	1	2
1	1	1	2	2	1	1	1	2	1
1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1
1	1	1	2	2	1	1	1	1	1

[Fig. 6]

[Fig. 7]은 10번째 useDetector() 함수를 호출한 후 죄인이 이동한 위치를 보여준다.

죄인은 이미 지나간 칸을 지나지 않기 위해서 시계 방향으로 회전하여 왼쪽 방향으로 방향을 바꾼다.

1	1	1	1	1	1	1	1	1	1
1	1	2	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	2
1	1	1	1	1	1	1	1	1	1
1	1	2	1	1	1	1	1	1	2
1	1	1	1	2	1	1	1	1	2
1	1	1	2	2	1	1	1	2	1
1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1
1	1	1	2	2	1	1	1	1	1

[Fig. 7]

[Fig. 8]은 12번째 useDetector() 함수를 호출한 후 죄인이 이동한 위치를 보여준다.

지금까지 탐색한 정보를 바탕으로 죄인의 위치를 특정할 수 있다.

죄인의 위치인 (2, 3)을 반환한다.

1	1	1	1	1	1	1	1	1	1
1	1	2	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	2
1	1	1	1	1	1	1	1	1	1
1	1	2	1	1	1	1	1	1	2
1	1	1	1	2	1	1	1	1	2
1	1	1	2	2	1	1	1	2	1
1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1
1	1	1	2	2	1	1	1	1	1

[Fig. 8]

[Table 1]은 useDetector() 함수를 호출할 때마다
죄인의 칸과 방향에 대한 가능한 경우의 수를 보여준다.

12번째 useDetector() 함수를 호출할 때 가능한 경우의 수가 1이 되어
죄인의 위치를 특정할 수 있다.

# of useDetector() Calls	# of Possible Cases	Remark
0	400	$(N) * (N) * (4 \text{ directions}) = 400$
1	148	
2	110	
3	80	
4	54	
5	36	
6	28	
7	22	
8	20	
9	20	
10	10	
11	6	
12	1	죄인의 위치를 특정할 수 있음

[Table 1]

[문제 요약]

- 숲의 최대 크기는 $50 * 50$, 1과 2 두 종류의 나무가 각 셀에 위치.
- 숲의 한쪽 가장 자리는 반대쪽 가장자리로 이어진다.
- 숲에 죄인이 도망 중이다. 탐지기를 이용하여 죄인의 위치를 찾아내라.
- 탐지기는 죄인을 중심으로 $D * D$ 영역의 정보를 알려준다. ($3 \leq D \leq 11$ 의 홀수)
- 탐지가 시작되면 죄인은 한 칸 움직이고 탐지기는 이동한 죄인의 정보를 알려준다.
- 죄인은 현재 방향으로 한 칸 움직이고 이미 방문한 곳이라면 시계 방향으로 회전한 후 한 칸 움직인다.
- 사용자는 main의 useDetector() 함수를 이용하여 탐지 정보를 얻을 수 있는데 TC마다 함수 사용 제한(최대 250번) 이 걸려 있다.
- 함수 사용 제한 이전에 답을 반드시 구할 수 있도록 문제가 주어진다.

- `init()` 함수를 제외하면 작성할 함수는 `findCriminal()` 뿐이다
- 아래와 같이 구현 가이드 라인이 주어져 있다.
`useDetector()` 함수를 호출할 때마다 죄인이 있을 수 있는 위치들을 좁혀 나갈 수 있다.
그 가능한 위치의 수가 1일 때 죄인의 위치를 확정하면 점수를 얻을 수 있음을 보장한다.
- 숲의 크기도 작은 편이고 TC당 `findCriminal()` 호출 수도 1 ~ 5회뿐이다.
- 그런데 죄인의 현재위치가 어디인지 도망중인 방향은 어떻게 되는지 알 수 없다.
- 죄인의 현재 위치에 대한 단서는 $D \times D$ 로 주어진다.
이 단서와 일치하는 것이 1개 뿐일 때, 죄인의 위치를 확정할 수 있다.
- 어떻게 탐색을 시작할 것인가?

Solution sketch

- 탐사 함수는 사용할 필요가 없는 버전으로 문제를 단순하게 바꾸어 생각해 보자.

1. 죄인의 **현재위치**와 **현재 방향**이 주어지고

T 시간 후에 가능한 죄인의 위치와 방향을 구하는 문제라면 어떻게 할 수 있을까?

=> 현재 위치에서 죄인 1명을 이동시키다가 이미 방문한 곳이라면 시계방향으로 바꾸어 진행하는 것을 T 시간 진행하면 된다.

2. 죄인의 **현재위치만** 주어지고 **방향은 주어지지 않은 상황**에서 T 시간 후에 가능한

죄인의 위치와 방향을 구하는 문제라면 어떻게 할 수 있을까?

=> 현재 위치에서 4방향으로 죄인 4명을 T 시간 이동시키는 문제로 보면 될 것이다.

이때 큐를 이용한 BFS 자료구조를 사용할 수 있다.

방문 체크를 위하여 각 위치에서는 4개의 체크 배열이 필요할 것이다.

3. 이제 **N의 현재 위치**가 주어지고 **방향은 주어지지 않은 상황**에서 **T**시간 후에 가능한 죄인의 위치와 방향을 구하는 문제라면 어떻게 할 수 있을까?

=> 현재 위치에서 죄인 $4*N$ 명을 **T**시간 이동시키는 문제로 보면 될 것이다.

이때 큐를 이용한 BFS 자료구조를 사용할 수 있다.

방문 체크를 위하여 각 위치에서는 $4*N$ 개의 체크 배열이 필요할 것이다.

- 이제 원래의 문제로 돌아가서 생각해 보자.

죄인의 현위치를 모르는 상황이므로 일단

`useDetector(int res[][])` 함수를 호출하여 `res[] []`를 얻고

숲 정보에서 `res[] []`와 같은 모든 곳을 죄인이 한 칸 움직인 이후의 지점으로

가정할 수 있다. (`useDetector` 함수는 한 칸 움직인 이후의 지점 정보를 알려준다.)

죄인이 한 칸 움직인 이후의 지점을 가정할 수 있으므로

이를 이용하여 시작점 또한 가정할 수 있다.

- 가능한 시작점의 최대 개수는 2500개 이고
각 지점마다 4개의 방향으로 출발할 수 있으므로
10000명의 죄인을 동시에 출발시키는 문제로 생각할 수 있다.
- 따라서 방문 체크 배열은 `visit[50][50][10000]`이 될 것이다.
25,000,000 이므로 `int`로 선언하면 약 100MB,
`short`로 선언하면 약 50MB의 메모리가 사용된다.

- BFS를 위한 큐의 크기는
 $50 * 50 * 250(\text{탐색 제한 값}) * 4 = 2,500,000$ 이면 가능하다.
- 큐에 추가되는 데이터는 r(행), c(열), d(방향), g(죄수번호) 가 되므로
아래와 같이 선언한다면 25만 * 16 = 4천만으로
int로 선언하는 경우 40메가, short로 선언하는 경우 20메가의
메모리가 필요하다.

```
struct Data {  
    int r, c, d, g; // row, column, dir, visit group  
    // short r, c, d, g;  
}que[QLM];
```

- `useDetector(int res[][])` 함수 호출로 얻어진 탐사 결과 일치하는 곳을 확인하기 위하여 어떤 방법을 사용할 수 있을까?
- 확인할 영역의 크기가 최대 $11 * 11 = 121$ 개이므로 단순 탐색할 경우 $2,500,000(\text{가능한 상태}) * 121 * 5(\text{findCriminal() 함수 호출 횟수}) =$ 약 1,500,000,000 이다.

이러한 TC가 50개 주어질 수 있으므로 시간은 더 걸린다.
아무래도 좀더 효율적인 방법이 필요할 것으로 보인다.

어떤 방법이 있을까?

- 가능한 한 가지 방법은 해시를 이용하는 것이다.
11*11 영역에 1또는 2 두 가지 정보만 있으므로 2진 비트로 바꾸어
121자리 2진수를 만들어 해시에 이용할 수 있다.
- 또다른 한 가지는 121자리 정수를
61자리와 60자리 unsigned long long 정수로 바꾸어
pair<unsigned long long, unsigned long long> 으로 다룰 수도 있다.

- 사용할 자료구조를 정의해 보자.

```
const int LM = 50;
const int DLM = 11;
const int QLM = 50 * 50 * 250 * 4;

using ULL = unsigned long long;

// 이동하는 죄수의 수 : 출발지가 여러 곳일 수 있으므로 여러 명의 죄인이 이동하는 것으로 생각한다.
int gcnt;

struct Data {
    int r, c, d, g;          // row, column, dir, prisoner number
} que[QLM];
int fr, re;
short visit[LM][LM][10000], vn; // 약 50 MB사용 (2500만 * 2Byte)
unordered_map<ULL, vector<Result>> hmap; // 최초시점을 정하는데 사용(생략가능)

int N, D;
int detectRes[DLM][DLM];    // useDetector() 결과
int Map[LM][LM];            // mForest[][]
ULL htab[LM][LM];           // hash table[][]
```

이제 각 함수 별 할 일을 정리해 보자.

void init(**int** N, **int** D, **int** mForest[][**int**], **int** K)

- N, D를 저장한다.
- 숲의 상태 mForest[] []를 백업해 둔다.
- mForest[r][c]를 중심으로 $D \times D$ 영역에 대하여 해시코드를 생성하여 htab[r][c]에 저장한다.

또는 두 자리 (unsigned) long long 변수로 만들어 저장할 수도 있다.

Result findCriminal() :

- 큐를 초기화 한다.
fr = re = 0;
- 죄인 번호 gid를 초기화 한다.
gid = 0;
- 방문 체크를 초기화 한다.
배열을 모두 0으로 초기화 하기보다는
아래와 같이 초기화 하여 시간을 절약할 수 있다.
++vn;

- 탐색을 일단 실행하고 해시코드를 구한다.

```
useDetector(detectRes);           // 탐색을 일단 실행 : 죄인이 한 칸 이동한 상태
ULL hcode = getCode();           // 죄인이 한칸 이동한 상태를 해싱
```

Result findCriminal() : (계속)

- 최초 시점으로 가능한 후보를 구한다.
해시를 사용한 경우라면 아래와 같이 구할 수 있다.

```
for (auto&t : hmap[hcode]) { // 최초시점으로 가능한 후보 구하기
    if (probing(t.r, t.c)) { // detectRes[][]은 죄수가 한 칸 이동한 후의 상태
        for (int d = 0; d < 4; ++d) {
            int sr = (t.r - dr[d] + N) % N; // 죄수의 출발지로 추측되는 지점
            int sc = (t.c - dc[d] + N) % N;
            visit[sr][sc][gcnt] = vn; // 출발지를 방문체크
            visit[t.r][t.c][gcnt] = vn; // 두번째 지점을 방문체크
            que[re++] = { t.r, t.c, d, gcnt }; // 두번째 지점부터 큐에 추가(detecting 결과이므로)
            gcnt++;
        }
    }
}
```

Result findCriminal() : (계속)

- 이 상태에서 유일하게 결정된 경우 답을 반환하고 함수를 종료한다.

```
if (re == 4) return { que[0].r, que[0].c }; // 현시점에서 유일하게 결정된 경우
```

- 그렇지 않다면 답을 구할 때 까지 다음 프로세스를 반복한다.
 - useDetector(detectRes);를 이용하여 탐색하고 해시코드 구하기
 - 큐의 $fr \sim re-1$ 구간의 상태를 업데이트하기
 - ✓ 현재 방향 업데이트가 불가능한 경우라면 시계방향으로 전환하기
 - ✓ 상태 전이가 가능한 경우라면 큐에 추가하고
 - 답을 가정해 보기
 - 가능한 답 후보가 여러 가지라면 계속진행시키기
 - 가능한 답 후보가 한 가지 뿐이라면 답을 반환하고 함수를 종료한다.

```
bool loop = true;           // 답이 유일하게 결정되는 경우 false로 변경된다.
while (fr < re && loop) {
    useDetector(detectRes);
    hcode = getCode();

    ans = {-1, -1};
    int ed = re;             // 현재 detecting 결과를 반영해야 하는 구간의 끝
    while (fr < ed) {
        Data t = que[fr++];
        for (int i = 0; i < 4; ++i, t.d = (t.d + 1) & 3) {
            int nr = (t.r + dr[t.d] + N) % N;
            int nc = (t.c + dc[t.d] + N) % N;
            if (visit[nr][nc][t.g] == vn) continue;
            if (hcode == htab[nr][nc] && probing(nr, nc)) {
                visit[nr][nc][t.g] = vn;
                que[re++] = { nr, nc, t.d, t.g };
                if (ans.r < 0) { // 답이 정해지지 않은 경우
                    ans = { nr, nc };
                    loop = 0;
                }
                else if (ans.r != nr || ans.c != nc) { // 답이 유일하지 않은 경우
                    loop = 1;
                }
            }
        }
        break;               // 방향 전환이 필요없는 경우
    }
}
return ans;
}
```

[Summary]

- 그래프 탐색방법 중에 BFS를 사용할 수 있다.
- 하나의 정수로 표현하기 어려운 2차원 구간을 압축하여 표현할 수 있다.
- 출발지가 여러 개인 문제를 BFS를 이용하여 해결할 수 있다.

[One more step]

- $D * D$ 구간을 두 개의 long long형 변수로 다룰 수 있다면 실행 시간을 많이 줄일 수 있다.

Code example

: BFS

full search compare

Code example 1

TS죄인의 숲

```
#include <vector>
#include <unordered_map>
using namespace std;

const int LM = 50;
const int DLM = 11;
const int QLM = 50 * 50 * 250 * 4;

using ULL = unsigned long long;
extern int useDetector(int p[DLM][DLM]);
static const int dr[4] = { -1, 0, 1, 0 }; // ↑,→, ↓, ←
static const int dc[4] = { 0, 1, 0, -1 };

struct Result { int r, c; };

int gcnt;
struct Data {
    int r, c, d, g; // row, column, dir, visit group
} que[QLM];
int fr, re;
short visit[LM][LM][10000], vn; // 약 50 MB사용 (2500만 * 2Byte)
unordered_map<ULL, vector<Result>> hmap; // 최초시점을 정하는데 사용(생략가능)
```

Code example 1

TS죄인의 숲

```
int N, D;
int detectRes[DLM][DLM];    // useDetector() 결과
int Map[LM][LM];            // mForest[][]
ULL htab[LM][LM];           // hash table[][]

int probing(int r, int c) { // 구간이 일치하는지 최종 확인하기
    int sr = r - D / 2 + N, sc = c - D / 2 + N;
    for (int i = 0; i < D; ++i) {
        for (int j = 0; j < D; ++j) {
            if (Map[(sr + i) % N][(sc + j) % N] != detectRes[i][j]) return 0;
        }
    }
    return 1;
}

ULL getCode(int r, int c, ULL code = 0) { // 숲 해싱
    int sr = r - D / 2 + N, sc = c - D / 2 + N;
    for (int i = 0; i < D; ++i) {
        for (int j = 0; j < D; ++j) {
            code = code * 2 + Map[(sr + i) % N][(sc + j) % N] - 1;
        }
    }
    return code;
}
```

Code example 1

TS죄인의 숲

```
ULL getCode(ULL ret = 0) {                                // detecting 결과 해싱
    for (int i = 0; i < D; ++i)
        for (int j = 0; j < D; ++j)
            ret = ret * 2 + (detectRes[i][j] - 1);
    return ret;
}

void init(int N, int D, int mForest[LM][LM], int K) {
    ::N = N, ::D = D;
    for (int i = 0; i < N; ++i)                            // 숲의 상태를 백업
        for (int j = 0; j < N; ++j)
            Map[i][j] = mForest[i][j];

    for (int i = 0; i < N; ++i) {                            // (i, j)를 중심으로 D*D영역을 해싱
        for (int j = 0; j < N; ++j) {
            htab[i][j] = getCode(i, j);
            hmap[htab[i][j]].push_back({ i, j }); // 최초시점을 정하는데 사용
        }
    }
}
```

Code example 1

TS죄인의 숲

```
Result findCriminal() {
    Result ans;

    fr = re = gcnt = 0;
    vn++;
    useDetector(detectRes);          // 탐색을 일단 실행 : 죄인이 한 칸 이동한 상태
    ULL hcode = getCode();           // 죄인이 한칸 이동한 상태를 해싱

    for (auto&t : hmap[hcode]) {      // 최초시점으로 가능한 후보 구하기
        if (probing(t.r, t.c)) {     // detectRes[][]은 죄수가 한 칸 이동한 후의 상태
            for (int d = 0; d < 4; ++d) {
                int sr = (t.r - dr[d] + N) % N;    // 죄수의 출발지로 추측되는 지점
                int sc = (t.c - dc[d] + N) % N;
                visit[sr][sc][gcnt] = vn;           // 출발지를 방문체크
                visit[t.r][t.c][gcnt] = vn;         // 두번째 지점을 방문체크
                que[re++] = { t.r, t.c, d, gcnt };  // 두번째 지점부터 큐에 추가(detecting 결과이므로)
                gcnt++;
            }
        }
    }

    if (re == 4) return { que[0].r, que[0].c };    // 현시점에서 유일하게 결정된 경우
}
```

Code example 1

TS죄인의 숲

```
bool loop = true;           // 답이 유일하게 결정되는 경우 false로 변경된다.
while (fr < re && loop) {
    useDetector(detectRes);
    hcode = getCode();

    ans = {-1, -1};
    int ed = re;             // 현재 detecting 결과를 반영해야 하는 구간의 끝
    while (fr < ed) {
        Data t = que[fr++];
        for (int i = 0; i < 4; ++i, t.d = (t.d + 1) & 3) {
            int nr = (t.r + dr[t.d] + N) % N;
            int nc = (t.c + dc[t.d] + N) % N;
            if (visit[nr][nc][t.g] == vn) continue;
            if (hcode == htab[nr][nc] && probing(nr, nc)) {
                visit[nr][nc][t.g] = vn;
                que[re++] = { nr, nc, t.d, t.g };
                if (ans.r < 0) { // 답이 정해지지 않은 경우
                    ans = { nr, nc };
                    loop = 0;
                }
                else if (ans.r != nr || ans.c != nc) { // 답이 유일하지 않은 경우
                    loop = 1;
                }
            }
        }
        break;               // 방향 전환이 필요없는 경우
    }
}
return ans;
}
```

Code example2

: BFS

user defined bitset

Code example 2

TS죄인의 숲

```
#include <utility>
using namespace std;

const int LM = 50;
const int DLM = 11;
const int QLM = 50 * 50 * 250 * 4; // queue size

extern int useDetector(int detectResult[DLM][DLM]);
struct Result{ int r, c; };

using ULL = unsigned long long;
using pll = pair<ULL, ULL>;

int N, D;
int Map[LM][LM];           // mForest[][] 백업
pll htab[LM][LM];          // D*D단위로 bitset으로 묶기
int detectResult[DLM][DLM]; // useDetector() 결과
int dr[] = {-1, 0, 1, 0}, dc[] = {0, 1, 0, -1}; // clockwise : 12시, 3시, 6시, 9시

struct Data {
    int r, c, d, g;          // row, column, dir, visit group
} que[QLM];
int fr, re, gcnt;
short visit[LM][LM][10000], vn; // 약 50 MB사용 (2500만 * 2Byte)
```

Code example 2

TS죄인의 숲

```
p11 getCode(int r, int c) { // 숲을 bitset으로
    r = (r - D / 2 + N) % N, c = (c - D / 2 + N) % N;
    ULL acode = 0, bcode = 0;
    int i, j, k = 0;
    for (i = 0; i < D; ++i) for (j = 0; j < D; ++j, ++k) {
        if (k < 61) acode = acode * 2 + (Map[(r + i) % N][(c + j) % N] - 1);
        else bcode = bcode * 2 + (Map[(r + i) % N][(c + j) % N] - 1);
    }
    return { acode, bcode };
}

p11 getCode() { // detecting 결과를 bitset으로
    int i, j, k = 0;
    ULL acode = 0, bcode = 0;
    for (i = 0; i < D; ++i) for (j = 0; j < D; ++j, ++k) {
        if (k < 61) acode = acode * 2 + detectResult[i][j] - 1;
        else bcode = bcode * 2 + detectResult[i][j] - 1;
    }
    return { acode, bcode };
}
```


Code example 2

TS죄인의 숲

```
void init(int N, int D, int mForest[LM][LM], int K){
    ::N = N, ::D = D;
    for (int i = 0; i < N; ++i) for (int j = 0; j < N; ++j) // 숲의 상태를 백업
        Map[i][j] = mForest[i][j];

    for (int i = 0; i < N; ++i) for (int j = 0; j < N; ++j) { // (i, j)를 중심으로 D*D영역을 bitset으로
        htab[i][j] = getCode(i, j);
    }
}

Result findCriminal(){
    Result ans = {-1, -1};
    useDetector(detectResult); // 탐색을 일단 실행 : 죄인이 한 칸 이동한 상태
    pll code = getCode();      // 죄인이 한칸 이동한 상태를 pll 타입의 code로 구하기

    fr = re = gcnt = 0;
    ++vn;

    // 최초시점으로 가능한 후보 구하기
    // detectResult[][] (즉 pll code) 는 죄수가 한 칸 이동한 후의 상태
    for(int r=0;r<N;++r) for(int c=0;c<N;++c) if(code == htab[r][c]){
        for (int d = 0; d < 4; ++d) {
            int sr = (r - dr[d] + N) % N;      // 죄수의 출발지로 추측되는 지점
            int sc = (c - dc[d] + N) % N;
            visit[sr][sc][gcnt] = vn;          // 출발지를 방문체크
            visit[r][c][gcnt] = vn;           // 두번째 지점을 방문체크
            que[re++] = {r, c, d, gcnt};      // 두번째 지점부터 큐에 추가(detecting 결과이므로)
            gcnt ++;
        }
    }
}
```

Code example 2

TS죄인의 숲

```
if (re == 4) return { que[0].r, que[0].c }; // 현시점에서 유일하게 결정된 경우

bool answerFixed = false; // 답이 유일하게 결정되는 경우 true로 변경된다.
while (answerFixed == false && fr < re) {
    useDetector(detectResult);
    code = getCode();
    ans = {-1, -1};
    int ed = re; // 현재 detecting 결과를 반영해야 하는 구간의 끝
    while (fr < ed) {
        Data t = que[fr++];
        for (int i = 0; i < 4; ++i, t.d = (t.d + 1) & 3) { // t로부터 한칸 이동하기
            int nr = (t.r + dr[t.d] + N) % N; // *****
            int nc = (t.c + dc[t.d] + N) % N; // *****
            if (visit[nr][nc][t.g] == vn) continue;
            if (htab[nr][nc] == code) {
                visit[nr][nc][t.g] = vn;
                que[re++] = { nr, nc, t.d, t.g };
                if (ans.r < 0) { // 답이 정해지지 않은 경우
                    ans = { nr, nc };
                    answerFixed = true;
                }
                else if (ans.r != nr || ans.c != nc) // 답이 유일하지 않은 경우
                    answerFixed = false;
            }
            break; // 방향전환이 필요없는 경우
        }
    }
}
return ans;
}
```

Thank you.