

김다인

알고리즘 스터디 3주

목차

- 어떤 문제가 시뮬레이션인가?
- 문제 풀이 예시 (뱀 / 빗물 / 감시)
- 수업 문제 1문제 풀이
- 공통 문제 3문제 안내

시뮬레이션과 구현

- 둘은 매우 유사하지만
- 시뮬레이션은 일련의 명령에 따라 개체를 차례대로 이동시키는 것. 완전탐색과도 비슷함
- 구현은 문제가 시키는 그대로 구현하는 것
- 이라는 아주 약간의 차이가 존재함. 딱히 구분할 필요는 없어 보임
- 근데 이제 완전탐색을 곁들인



어떤 문제가 시뮬레이션인가?

- 대표적으로 이런 문제들
- 2차원 리스트가 주어지고 주어진 조건에 따라 어떤 물체 등을 바꿔가면서 대입해 봐야 되는 문제
- 문제 설명이 길고 풀기 싫게 생긴(풀이가 복잡하고 노가다 코드를 짜야 할 것 같은) 문제들
- 그냥 많이 풀어서 익숙해지는 게 유일한 방법..

14500번

제출

맞힌 사람

숫코딩

재제정 결과

제정 현황

내 제출

강의

질문 검색

테트로미노

5 골드 V

시간 제한	메모리 제한	제출	정답	맞힌 사람	정답 비율
2 초	512 MB	49793	18610	12040	35.305%

문제

폴리오미노란 크기가 1×1 인 정사각형을 여러 개 이어서 붙인 도형이며, 다음과 같은 조건을 만족해야 한다.

- 정사각형은 서로 겹치면 안 된다.
- 도형은 모두 연결되어 있어야 한다.
- 정사각형의 변끼리 연결되어 있어야 한다. 즉, 꼭짓점과 꼭짓점만 맞닿아 있으면 안 된다.

정사각형 4개를 이어 붙인 폴리오미노는 테트로미노라고 하며, 다음과 같은 5가지가 있다.



뱀

다국어

☆

한국어

5 골드 V

시간 제한	메모리 제한	제출	정답	맞힌 사람	정답 비율
1 초	128 MB	42623	17158	11325	38.582%

문제

'Dummy' 라는 도스게임이 있다. 이 게임에는 뱀이 나와서 기어다니다는데, 사과를 먹으면 뱀 길이가 늘어난다. 뱀이 이리저리 기어다니다가 벽 또는 자기자신의 몸과 부딪하면 게임이 끝난다.

게임은 $N \times N$ 정사각 보드 위에서 진행되고, 몇몇 칸에는 사과가 놓여져 있다. 보드의 상하좌우 끝에 벽이 있다. 게임이 시작할때 뱀은 맨위 맨좌측에 위치하고 뱀의 길이는 1 이다. 뱀은 처음에 오른쪽을 향한다.

뱀은 매 초마다 이동을 하는데 다음과 같은 규칙을 따른다.

- 먼저 뱀은 몸길이를 늘려 머리를 다음칸에 위치시킨다.
- 만약 이동한 칸에 사과가 있다면, 그 칸에 있던 사과가 없어지고 꼬리는 움직이지 않는다.
- 만약 이동한 칸에 사과가 없다면, 몸길이를 줄여서 꼬리가 위치한 칸을 비워준다. 즉, 몸길이는 변하지 않는다.

사과의 위치와 뱀의 이동경로가 주어질 때 이 게임이 몇 초에 끝나는지 계산하라.

뱀

- 백준 3190 (삼성 SW 역량 기출)
- 셋 중에 굳이 하나로 분류하자면 구현 feel
- 진짜 처음 봤을 때 도대체 어떤 알고리즘을 써야 하는지 알 수 없는 문제인데
- 이건.. 알고리즘을 쓸 수 없고 그냥 시키는대로 구현할 수밖에

문제

'Dummy' 라는 도스게임이 있다. 이 게임에는 뱀이 나와서 기어다니는데, 사과를 먹으면 뱀 길이가 늘어난다. 뱀이 이리저리 기어다니다가 벽 또는 자기자신의 몸과 부딪하면 게임이 끝난다.

게임은 $N \times N$ 정사각 보드위에서 진행되고, 몇몇 칸에는 사과가 놓여져 있다. 보드의 상하좌우 끝에 벽이 있다. 게임이 시작할때 뱀은 맨위 맨좌측에 위치하고 뱀의 길이는 1 이다. 뱀은 처음에 오른쪽을 향한다.

뱀은 매 초마다 이동을 하는데 다음과 같은 규칙을 따른다.

- 먼저 뱀은 몸길이를 늘려 머리를 다음칸에 위치시킨다.
- 만약 이동한 칸에 사과가 있다면, 그 칸에 있던 사과가 없어지고 꼬리는 움직이지 않는다.
- 만약 이동한 칸에 사과가 없다면, 몸길이를 줄여서 꼬리가 위치한 칸을 비워준다. 즉, 몸길이는 변하지 않는다.

사과의 위치와 뱀의 이동경로가 주어질 때 이 게임이 몇 초에 끝나는지 계산하라.

입력

첫째 줄에 보드의 크기 N 이 주어진다. ($2 \leq N \leq 100$) 다음 줄에 사과의 개수 K 가 주어진다. ($0 \leq K \leq 100$)

다음 K 개의 줄에는 사과의 위치가 주어지는데, 첫 번째 정수는 행, 두 번째 정수는 열 위치를 의미한다. 사과의 위치는 모두 다르며, 맨 위 맨 좌측 (1행 1열) 에는 사과가 없다.

다음 줄에는 뱀의 방향 변환 횟수 L 이 주어진다. ($1 \leq L \leq 100$)

다음 L 개의 줄에는 뱀의 방향 변환 정보가 주어지는데, 정수 X 와 문자 C 로 이루어져 있으며, 게임 시작 시간으로부터 X 초가 지난 뒤에 왼쪽(C 가 'L') 또는 오른쪽(C 가 'D')로 90도 방향을 회전시킨다는 뜻이다. X 는 10,000 이하의 양의 정수이며, 방향 전환 정보는 X 가 증가하는 순으로 주어진다.





뱀 문제, 어떻게 풀어야 할까?

- 일단 우리는 프로니까 보드의 변 길이 N 을 받고 $N*N$ 보드를 바로 만들어준다
- 사과 개수와 위치를 받아서 사과를 보드에 놔준다
- 뱀이 언제 방향을 바꾸는지 받아준다
 - 2차원 리스트이기 때문에 뱀이 상하좌우 360도로 회전하며 움직인다
 - 따라서 방향 벡터도 필요하다

```
N = int(input())
board = [[0 for _ in range(N)] for _ in range(N)]
K = int(input())
for _ in range(K):
    x, y = map(int, input().split())
    x, y = x-1, y-1 # (0, 0)을 시작점으로 할 수 있도록 좌표 줄여주기
    board[x][y] = 1 # 사과를 놓는다

# 방향
L = int(input())
rotate = deque([])
for _ in range(L):
    x, c = input().split() # 초, 회전 방향 받기
    x = int(x)
    rotate.append((x, c))
```

뱀 문제, 어떻게 풀어야 할까?

- 방향 벡터가 동, 남, 서, 북 순서인 것은 처음에는 뱀 머리가 오른쪽을 향하고 오른쪽으로 이동하기 때문이다.
- 즉, 뱀은 오른쪽으로 회전할 때     순으로 회전한다.
- 뱀의 위치, 초를 셀 변수, 방향 변수를 설정한다
 - 뱀의 몸 길이는 사과를 먹으면 늘어나며, 이 뱀은 길이가 충분히 늘어나면 1자로만 움직이지 않을 수도 있다. (몸이 꺾일 수 있음)
 - 그렇기 때문에 뱀의 몸통 전체 좌표를 나타낼 리스트 등으로 뱀을 것이 좋을 것이다.
- 이제 구현한다

```
# 동남서북 (0, 1, 2, 3) 순서
dx = [0, 1, 0, -1]
dy = [1, 0, -1, 0]

snake = [(0, 0)] # 뱀은 길이가 늘어날 수 있으므로 리스트로 함
sec = 0 # 초
d = 0 # 방향
```

매 초의 시작 : 회전 확인하기

- 이 게임은 뱀이 벽에 머리를 부딪히거나 자기 자신의 몸과 부딪히면 끝난다. 즉, 언제 끝날지 모르므로 일단 무한루프를 만든다
- 매 초가 시작될 때, 뱀은 지금 회전해야 하는지 아닌지부터 파악을 해야 한다
- Rotate가 있는지 조사하고, 있다면 아까 텍으로 받아준 rotate[0][0]을 조회해서 현재 회전해야 하는 때인지 아닌지 파악한다.
 - 만약 회전해야 한다면, rotate[0][1]을 조회해 'L'인지 'D'인지 조사한다.
 - 'D'(우측 90도 회전)라면 방향 변수에 1을 더하고 모듈러 연산을 해 준다. (계산을 하다 보면 d는 마이너스가 될 수도 있고 4 이상이 될 수도 있으므로 모듈러 연산을 해 주는 것이다)
 - rotate[0]은 이미 처리된 것이므로 없애기 위해 rotate.popleft()를 한다.

```
while True:
    if rotate and sec == rotate[0][0]:
        if rotate[0][1] == 'D':
            d = (d + 1) % 4
        else:
            d = (d - 1) % 4
        rotate.popleft()
```


늘어날 머리가 종료 조건인지 확인하기

- 뱀 머리(snake[-1]) 좌표를 복사하고, 방향 변수에 맞는 방향 벡터를 x, y축에 각각 더해준다.
 - 이 뱀은 움직일 때 머리가 늘어나기 때문에 늘어날 머리 좌표를 새로 찾아주는 것이다.
- 만약 이렇게 늘어난 뱀 머리가 벽에 부딪히거나 자기 몸에 부딪힌다면 게임을 종료한다
- Snake는 뱀의 몸 좌표를 들고 있는 리스트이다. 이 안에 현재 늘어날 머리 좌표가 있는지 확인하는 것이다.

```
x, y = snake[-1]
x, y = x + dx[d], y + dy[d]

if x < 0 or x >= N or y < 0 or y >= N:
    break
if (x, y) in snake:
    break
```

사과가 있을 때

- 만약 사과가 있다면 몸을 굳이 줄일 필요가 없다. 지금 늘어난 머리 그대로 늘어나는 것이므로 (x, y)를 뱀에 추가해준다.
- 사과를 먹었으므로 그 자리를 0으로 만들어주고, 1초를 추가한 뒤 넘어간다

```
if board[x][y] == 1:  
    snake.append((x, y))  
    sec += 1  
    board[x][y] = 0  
    continue
```

일반적인 경우

- 사과를 먹지 않고, 뱀 머리가 벽에 부딪히지도 않고 자기 몸에 부딪히지도 않은 채로 그냥 이동만 하는 경우이다.
- 그럼 이제 뱀의 몸통을 다 움직여야 할 때이다.
- 뱀의 i 번째 몸통은 뱀의 $i+1$ 번째 몸통의 위치로 움직여야 하므로 for문으로 구현한다.
- 뱀 머리는 현재 (x, y) 로 대체해준다.
- 1초가 지났으므로 초를 늘려준다

```
for i in range(len(snake)-1):  
    snake[i] = snake[i+1][0], snake[i+1][1]  
  
snake[-1] = (x, y)  
sec += 1
```

```

from collections import deque

N = int(input())
board = [[0 for _ in range(N)] for _ in range(N)]
K = int(input())
for _ in range(K):
    x, y = map(int, input().split())
    x, y = x-1, y-1 # (0, 0)을 시작점으로 할 수 있도록 좌표 줄여주기
    board[x][y] = 1 # 사과를 놓는다

# 방향
L = int(input())
rotate = deque([])
for _ in range(L):
    x, c = input().split() # 초, 회전 방향 받기
    x = int(x)
    rotate.append((x, c))

# 동남서북 (0, 1, 2, 3) 순서
dx = [0, 1, 0, -1]
dy = [1, 0, -1, 0]

snake = [(0, 0)] # 뱀은 길이가 늘어날 수 있으므로 덱으로 함
sec = 0 # 초
d = 0 # 방향

```

```

while True:
    if rotate and sec == rotate[0][0]:
        if rotate[0][1] == 'D':
            d = (d + 1) % 4
        else:
            d = (d - 1) % 4
        rotate.popleft()

    x, y = snake[-1]
    x, y = x + dx[d], y + dy[d]

    if x < 0 or x >= N or y < 0 or y >= N:
        break
    if (x, y) in snake:
        break

    if board[x][y] == 1:
        snake.append((x, y))
        sec += 1
        board[x][y] = 0
        continue

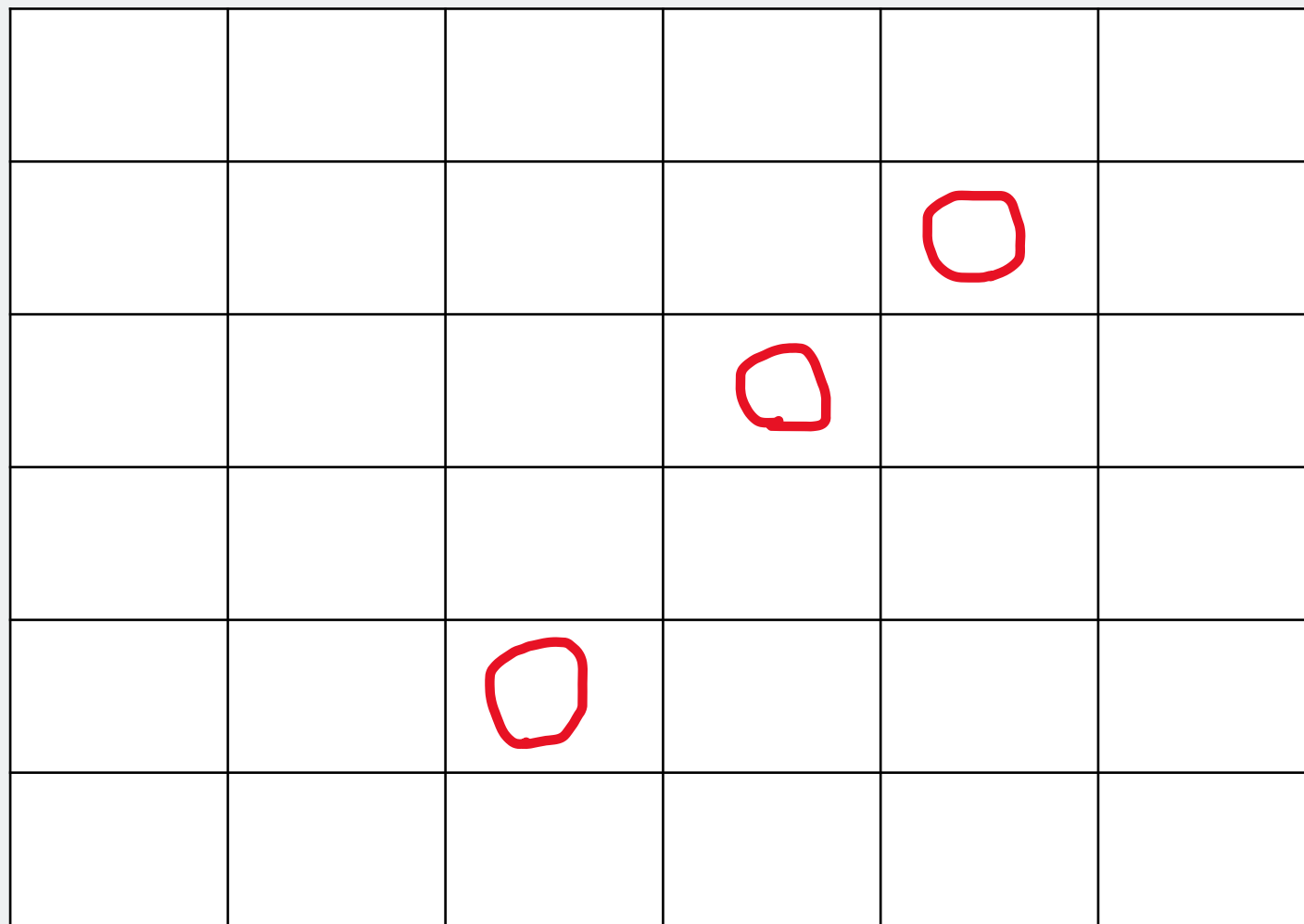
    for i in range(len(snake)-1):
        snake[i] = snake[i+1][0], snake[i+1][1]

    snake[-1] = (x, y)
    sec += 1

print(sec + 1)

```

6 보드 크기
3 사과 개수
3 4
2 5
5 3
3 회전 횟수
3 D
15 L
17 D



눈치채셨죠?

- 문제는 계속 왔다갔다하면서 읽기보다 한 번에 짹 읽고 종이에 정리해서 어떻게 코드로 바꿀지 생각하는 게 더 도움이 될 듯합니다
- 여기 알고리즘이란 딱히 없습니다..
- 만약에 BFS/DFS를 쓰더라도 기록지 않습니다 우선탐색 속 완탐일뿐,,
- 그냥 시키는대로 하는 수밖에.. 정말 말그대로 구현력을 테스트합니다
- 물론 문제따라 탐색을 할 수는 있겠죠 하지만.....

빗물

- 백준 14719
- 대표적인 시뮬/구현 문제
- 규칙을 잘 보면 풀 수 있는 문제

문제

2차원 세계에 블록이 쌓여있다. 비가 오면 블록 사이에 빗물이 고인다.



비는 충분히 많이 온다. 고이는 빗물의 총량은 얼마일까?

입력

첫 번째 줄에는 2차원 세계의 세로 길이 H 와 2차원 세계의 가로 길이 W 가 주어진다. ($1 \leq H, W \leq 500$)

두 번째 줄에는 블록이 쌓인 높이를 의미하는 0이상 H 이하의 정수가 2차원 세계의 맨 왼쪽 위치부터 차례대로 W 개 주어진다.

따라서 블록 내부의 빈 공간이 생길 수 없다. 또 2차원 세계의 바닥은 항상 막혀있다고 가정하여도 좋다.

출력

2차원 세계에서는 한 칸의 용량은 1이다. 고이는 빗물의 총량을 출력하여라.

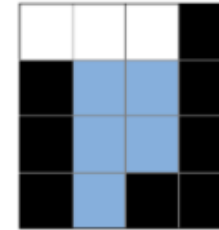
빗물이 전혀 고이지 않을 경우 0을 출력하여라.

쉽게 풀 수 있는 방법이 있을까?

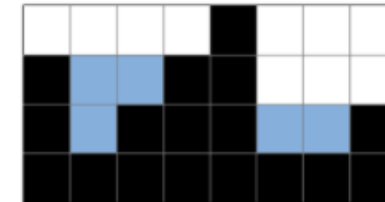
- 저 2차원 리스트를 실제로 구현하는 것도 좋은 방법이지만, 규칙을 찾는 것이 더 좋은 방법일 때도 있다
- 예시를 보면, 맨 왼쪽과 맨 오른쪽 세로줄에는 빗물이 고일 수 없다.
- 빗물은 블록 **사이에**만 고이기 때문이다
- 너비가 5(0~4)일 때, 빗물은 1~3 사이에만 고인다는 사실을 관찰할 수 있다

힌트

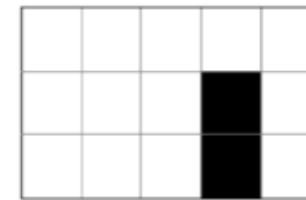
힌트 1:



힌트 2:



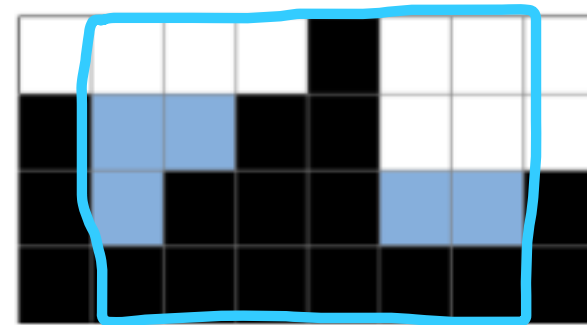
힌트 3:



두 번째 예시로 관찰

- 0~7까지 공간이 있으며, 약속에 따라 빗물은 1~6까지 6줄에만 고일 수 있다.
- 1번 줄의 경우, 0번 줄 높이가 1번 줄보다 높다. 오른쪽을 봤을 때, 가장 높은 줄은 4번 줄이다.
- 하지만 빗물은 1번 줄 높이까지만 고인다. 4번 높이만큼은 빗물이 넘치기 때문에 고이지 못한다.
- 즉, 우리는 현재 줄 기준으로 (가장 높은 왼쪽 줄과 가장 높은 오른쪽 중 작은 높이 - 내 높이)만큼 현재 줄에 빗물이 고인다는 것을 알 수 있다.

힌트 2:



코드로 구현

- 중요한 건 반드시 현재 줄 높이가 왼쪽과 오른쪽의 최대로 높은 줄 중 더 낮은 줄 높이보다 낮아야 빗물이 고인다는 것
- 만약에 높이가 같다면 평지라서 빗물이 고일 수 없고, 높다면 말할 것도 없음
- 다소 허망한 수준의 코드
- 대표적 구현 문제라 찾아왔고 이렇게 규칙이 중요한 간단한 문제도 있음을 소개합니다

```
h, w = map(int, input().split())
heights = list(map(int, input().split()))
result = 0

for i in range(1, w-1):
    left_max = max(heights[:i])
    right_max = max(heights[i+1:])

    height = min(left_max, right_max)
    if heights[i] < height:
        result += height - heights[i]

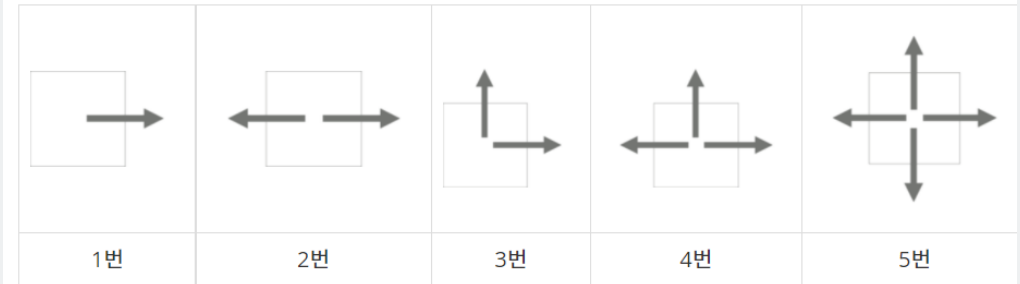
print(result)
```

감시

- 백준 15683 (삼성 SW 역량 테스트 기출)
- 문제가 너무 길어서 브라우저로 볼게요
- <https://www.acmicpc.net/problem/15683>
- 뭐 이것도 시뮬레이션 문제죠 그냥?
- DFS를 사용해서 풀 수 있는 문제입니다. 근데 이제 백트래킹 + 완전탐색을 곁들인...
- 모든 경우의 수를 다 시험해보기 전에는 어떤 것이 최적인지 알 수 없기 때문에 완전탐색
- 예시에서도 친절하게 보여주듯이 모든 조합을 그냥 다 시험해보아야합니다

문제

스타트링크의 사무실은 1×1 크기의 정사각형으로 나누어져 있는 $N \times M$ 크기의 직사각형으로 나타낼 수 있다. 사무실에는 총 K 개의 CCTV가 설치되어져 있는데, CCTV는 5가지 종류가 있다. 각 CCTV가 감시할 수 있는 방법은 다음과 같다.



1번 CCTV는 한 쪽 방향만 감시할 수 있다. 2번과 3번은 두 방향을 감시할 수 있는데, 2번은 감시하는 방향이 서로 반대방향이어야 하고, 3번은 직각 방향이어야 한다. 4번은 세 방향, 5번은 네 방향을 감시할 수 있다.

CCTV는 감시할 수 있는 방향에 있는 칸 전체를 감시할 수 있다. 사무실에는 벽이 있는데, CCTV는 벽을 통과할 수 없다. CCTV가 감시할 수 없는 영역은 사각지대라고 한다.

CCTV는 회전시킬 수 있는데, 회전은 항상 90도 방향으로 해야 하며, 감시하려고 하는 방향이 가로 또는 세로 방향이어야 한다.

일단 침착하게

- 할 수 있는 일부터 한다
- 가로 세로 받고 사무실 구조부터 받아준다
- 이 때 CCTV는 따로 빼두시고 방향 벡터도 만들어두세요
- CCTV는 모드 번호, 좌표 순으로

```
import copy

n, m = map(int, input().split())
office = [list(map(int, input().split())) for _ in range(n)]
cam = []
# 동남서북 방향
dx = [1, 0, -1, 0]
dy = [0, 1, 0, -1]

✓ for i in range(n):
✓   for j in range(m):
✓       if office[i][j] in (1, 2, 3, 4, 5):
           # 카메라 모드 번호, 좌표가 들어감
           cam.append((office[i][j], i, j))
```

CCTV 방향.. 그거 어떻게 하는건데

- 그런데 1번부터 5번까지의 CCTV 방향은 어떻게 해야?
- Mode를 정해주는 게 도움이 된다 (이건 저도 몰랐던 사실)
- 1번부터 5번까지 CCTV가 탐색할 수 있는 경우의 수가 있기 때문에 번호별로 경우의 수를 정리해준다
- 이 때 사각지대를 담은 result 변수도 만들어준다
- 처음에는 최대값인 $n * m$ 을 넣어주기

```
mode = [  
    [],  
    [[0], [1], [2], [3]],  
    [[0, 2], [1, 3]],  
    [[0, 1], [1, 2], [2, 3], [0, 3]],  
    [[0, 1, 2], [0, 1, 3], [1, 2, 3], [0, 2, 3]],  
    [[0, 1, 2, 3]],  
]  
result = n * m
```

그래서 이제 어쩔 건가?

- DFS 방식으로 문제를 풀어보자. 이렇게 하면 모든 경우의 수를 다 체크해볼 수 있다
- 카메라의 전체 개수를 depth라고 치면, 아래와 같은 경우에는 3 깊이까지 들어가서 탐색한다
- 그래서 아래 경우의 수를 전부 탐색할 수 있음
- [상하 상하 상하좌우] [상하 좌우 상하좌우]
[좌우 상하 상하좌우] [좌우 좌우 상하좌우]

예제 입력 2 복사

```
6 6
0 0 0 0 0 0
0 2 0 0 0 0
0 0 0 0 6 0
0 6 0 0 2 0
0 0 0 0 0 0
0 0 0 0 0 5
```

DFS 함수 작성

- 일단 사각지대의 최소값을 담을 result는 전역 변수의 것을 그대로 쓴다
- If depth == len(cam)에 걸릴 때는 1가지 경우의 수를 모두 완료했을 때이다.
- 즉, 위의 경우 if문에는 4번 걸리게 된다
- 1가지 경우의 수를 모두 봤을 때는 사무실에서 아직 0으로 남은 사각지대를 세서 최소값을 계산해주면 된다

```
# depth는 몇 단계까지 짚어 내려왔는지를 알림
def dfs(depth, office):
    global result

    # 여기서 잡았다는 건 1가지 경우의 수를 완료했다는 것
    # 즉, 이 if문은 모든 경우의 수만큼 걸림
    if depth == len(cam):
        count = 0
        for i in range(n):
            count += office[i].count(0)
        # 사각지대가 더 작아졌다면 갱신
        result = min(result, count)
    return
```

만약 아직 마지막 카메라까지 못 봤다면

- 지금 사무실을 temp로 복사함. (이 때, office는 원본 그대로의 office가 아닐 수 있음. DFS에 넘어가는 office는 depth 카메라가 감시한 뒤의 office이기 때문에)
- Cam에 저장해둔 CCTV 모드 번호와 좌표를 각각 저장해준다
- 그런 다음 해당 모드에서 가능한 경우들을 for문으로 테스트하는데, 이 때 fill 함수를 사용
- Fill 후에는 다음 카메라까지 보기 위해 DFS 함수를 호출

```
# temp = office[:]는 잘못된 결과가 나옴
# copy.deepcopy와 [:] 차이를 알아둬시다
temp = copy.deepcopy(office)
# 현재 depth의 카메라 모드 번호와 좌표
cam_num, x, y = cam[depth]

for i in mode[cam_num]:
    # 일단 이 카메라 모드가 감시할 수 있는 방향으로 테스트
    fill(temp, i, x, y)
    # 다음 카메라로 넘어가면서 DFS
    dfs(depth+1, temp)
    # for문 다시 돌아서 fill 함수 해야되니까 temp를 다시 원상복구시킴
    temp = copy.deepcopy(office)
```


Fill 함수

- 카메라가 보는 방향이 주어졌을 때 실제로 감시해서 office를 변경시키는 함수이다.
- 사무실, [방향], 좌표를 매개변수로 넘김
- 방향 벡터에서 인덱스에 해당하는 방향으로 진행하며 벽을 만나거나 오피스 벽을 뚫고 갈 때까지 쭉 진행하며 감시 구역으로 만듦

```
def fill(office, d, x, y):  
    for i in d:  
        nx = x  
        ny = y  
        while True:  
            nx += dx[i]  
            ny += dy[i]  
            if nx < 0 or ny < 0 or nx >= n or ny >= m:  
                break  
            if office[nx][ny] == 6:  
                break  
            elif office[nx][ny] == 0:  
                office[nx][ny] = -1
```

```

import copy

n, m = map(int, input().split())
office = [list(map(int, input().split())) for _ in range(n)]
cam = []
# 동남서북 방향
dx = [1, 0, -1, 0]
dy = [0, 1, 0, -1]

for i in range(n):
    for j in range(m):
        if office[i][j] in (1, 2, 3, 4, 5):
            # 카메라 모드 번호, 좌표가 들어감
            cam.append((office[i][j], i, j))

mode = [
    [],
    [[0], [1], [2], [3]],
    [[0, 2], [1, 3]],
    [[0, 1], [1, 2], [2, 3], [0, 3]],
    [[0, 1, 2], [0, 1, 3], [1, 2, 3], [0, 2, 3]],
    [[0, 1, 2, 3]],
]
result = n * m

def fill(office, d, x, y):
    for i in d:
        nx = x
        ny = y
        while True:
            nx += dx[i]
            ny += dy[i]
            if nx < 0 or ny < 0 or nx >= n or ny >= m:
                break
            if office[nx][ny] == 6:
                break
            elif office[nx][ny] == 0:
                office[nx][ny] = -1

```

```

def dfs(depth, office):
    global result

    # 여기서 잡았다는 건 1가지 경우의 수를 완료했다는 것
    # 즉, 이 if문은 모든 경우의 수만큼 걸림
    if depth == len(cam):
        count = 0
        for i in range(n):
            count += office[i].count(0)
        # 사각지대가 더 작아졌다면 갱신
        result = min(result, count)
        return

    # temp = office[:]는 잘못된 결과가 나옴
    # copy.deepcopy와 [:] 차이를 알아둬시다
    # temp = office[:]
    temp = copy.deepcopy(office)
    # 현재 depth의 카메라 모드 번호와 좌표
    cam_num, x, y = cam[depth]

    for i in mode[cam_num]:
        # 일단 이 카메라 모드가 감시할 수 있는 방향으로 테스트
        fill(temp, i, x, y)
        # 다음 카메라로 넘어가면서 DFS
        dfs(depth+1, temp)
        # for문 다시 돌아서 fill 함수 해야되니까 temp를 다시 원상복구시킴
        # temp = office[:]
        temp = copy.deepcopy(office)

    dfs(0, office)
    print(result)

```

Deepcopy와 [:] 차이

- Deepcopy는 깊은 복사로, 기존 객체와 완전 다른 객체이므로 기존 객체 값 변경에 영향을 받지 않음
- [:]를 이용할 경우 copy.copy를 사용한 것과 같은 결과. 객체를 복사하되 내부객체는 참조 객체임
- 만약 2차원 리스트 a가 있고 $b = a[:]$ 한다면, $\text{id}(a)$ 와 $\text{id}(b)$ 는 다르지만 $\text{id}(a[0])$ 와 $\text{id}(b[0])$ 은 같다는 뜻
- <https://wikidocs.net/16038>

```
import copy
```

```
a = [[1,2], [3,4]]
```

```
b = a[:]
```

```
c = copy.deepcopy(a)
```

```
# [:]를 하거나 deepcopy를 하거나 id는 달라짐
```

```
print(id(a), id(b), id(c))
```

```
# 하지만 [:]의 경우 id가 같음
```

```
print(id(a[0]), id(b[0]), id(c[0]))
```

```
# 재할당의 경우 문제 없음. id가 달라짐
```

```
a[0] = [8, 9]
```

```
print(*a, id(a[0]))
```

```
print(*b, id(b[0]))
```

```
# 하지만 리스트 내부 값을 직접 변경할 경우 문제가 생김
```

```
# 내부 리스트의 id가 같기 때문이다
```

```
a[1].append(10)
```

```
print(*a, id(a[1]))
```

```
print(*b, id(b[1]))
```

```
print(*c)
```

a

b

c

1692793146624 1692793337216 1692793337
856

1692792864256 1692792864256 1692793337
600

[8, 9] [3, 4] 1692793337280)

[1, 2] [3, 4] 1692792864256)

[8, 9] [3, 4, 10] 1692793178624)

[1, 2] [3, 4, 10] 1692793178624)

[1, 2] [3, 4]

수업 문제

- 지금 풀어봅시다!
- 백준 16236 아기 상어
- 어려우므로 지금 당장은 1시간 내에 못 풀어도 괜찮습니다 :) 저도 지금 풀 거예요!
- 만약에 못 풀더라도 풀이를 찾아서 꼭 이해하면 좋을 유명한 문제입니다

공통 문제

- 백준 14500 테트로미노
 - 백준 16234 인구 이동
 - 백준 17144 미세먼지 안녕!
-
- 전부 삼성 SW 역량 테스트 기출 문제입니다.
 - 구현 문제 풀이에 도움이 되시길 바랍니다 ☺