SCV/2021

Escaping

- Greedy, Geometry -

안태진(taejin7824@gmail.com)
GitHub(github.com/taejin1221)
상명대학교 소프트웨어학과
201821002

Contents

Problem

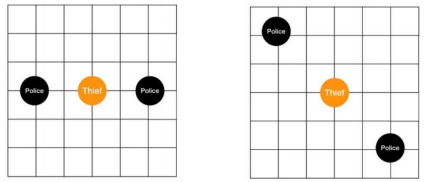
- Problem
  - 드디어..! 작년 예선 마지막 골드 문제!
  - 도둑이 도망칠 수 있는가?!

문제

무한 정수 좌표 평면 위 한 격자 점에 있는 도둑 잡기 게임을 해 보자. 이 도둑을 잡기 위해 *N*명의 경찰이 도둑이 없는 *N*개의 다른 격자 점에 배치되어 있다. 도둑 한 명과 *N*명 경찰들은 서로 한번씩 번갈아 가면서 움직인다. 도둑이 먼저 시작한다. 경찰 차례일 때는 *N* 명의 경찰 모두가 동시에 움직인다. 경찰과 도둑은 상하좌우 인접한 네 개의 격자 점 중 하나로 이동할 수 있다. 단, 경찰은 그대로 머물러 있을 수 있지만 도둑은 자신의 차례가 오면 반드시 이웃 격자 점으로 이동해야 한다. 이때 경찰과 도둑이 같은 격자 점에서 만나면 도둑은 잡힌 것이 되고 게임은 끝이 난다. 이 게임에서 경찰과 도둑은 최선을 다한다. 즉, 도둑은 최대한 잡히지 않으려는 전략을, 경찰은 최대한 빨리 잡으려는 전략을 쓴다.

입력으로 주어진 초기 위치에서 경찰이 어떤 전략을 사용하더라도 도둑이 경찰에 의해 잡히지 않고 영원히 도망 다닐 수 있다면, 그 초기 조건은 도둑이 "**탈출 가능한**" 조건이라고 부른다. 여러분은 초기 조건, 즉 한 명의 도둑과 *N* 명의 경찰의 처음 위치를 보고 도둑이 탈출 가능한지 판단해야 한다.

예를 들어, 다음 아래 왼쪽 그림과 같은 초기 조건에서 도둑이 북쪽 또는 남쪽 방향으로만 움직인다면 경찰은 도둑을 영원히 잡을 수 없다. 그러나 아래 오른쪽 그림과 같은 조건이라면 도둑이 어떻게 도망을 가더라도 최선을 다하는 경찰에 의해서 결국은 잡히게 된다.



탈출 가능

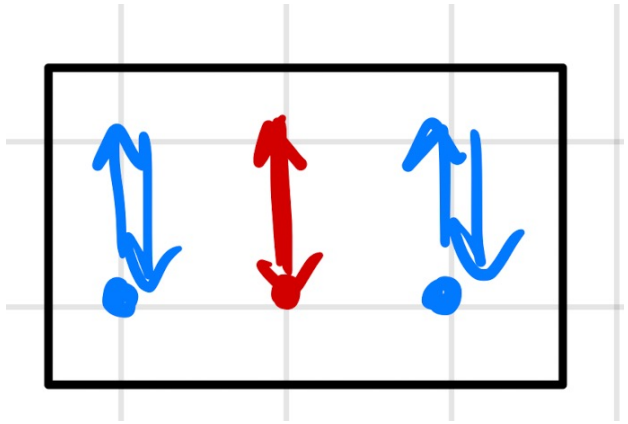
탈출 불가능

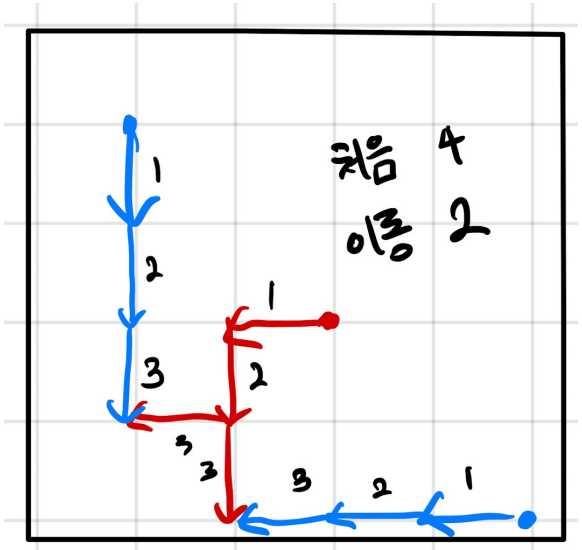
Contents

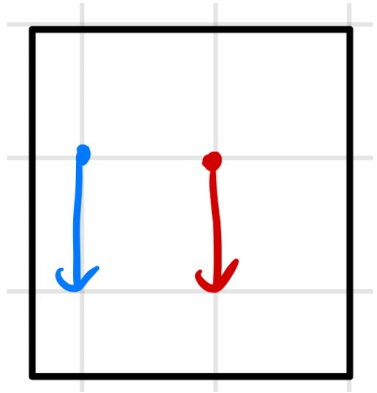
- Problem
- Approach
- Code

Approach

- Approach 1
  1. 도둑과 경찰이 거리를 좁힐 수 없다면 무조건 도망칠 수 있을 듯!
     - 만약 거리를 좁힐 수 있다면 도망칠 수 없음
     - 어차피 도둑이 움직이는 곳은 4군데 뿐이니 $4 \times 500{,}000 = 2{,}000{,}000$ 으로 충분!

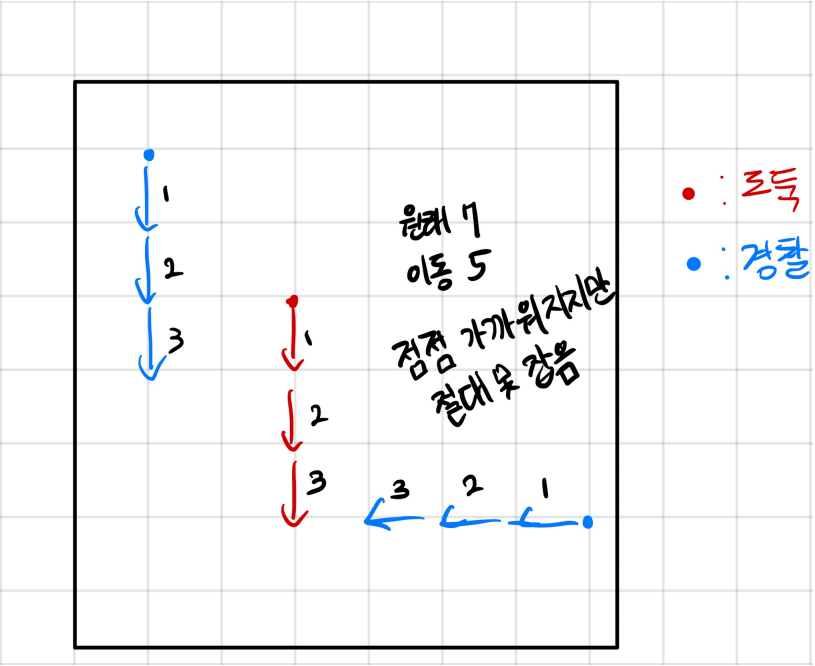






Approach

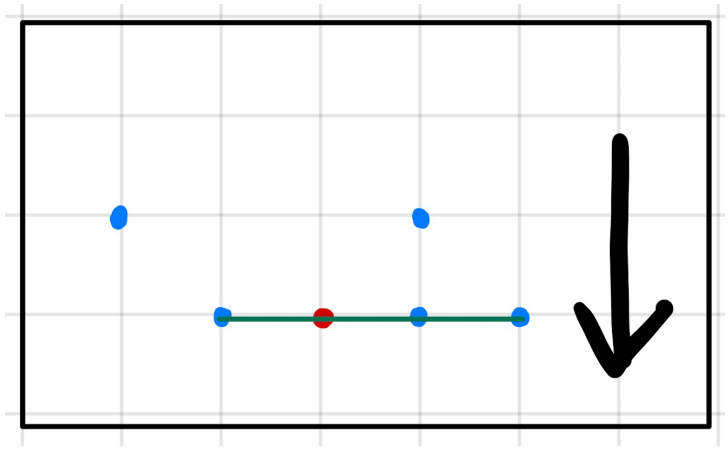
- Approach 1
  - 반례
    - 오른쪽 아래의 경찰과 가까워지지만 절대 못 잡음

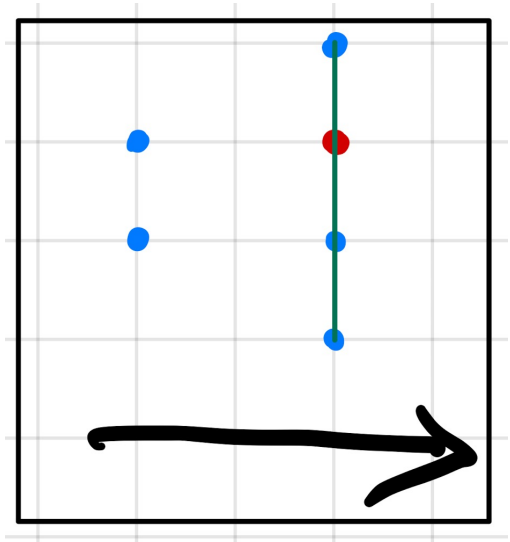


Approach

- 자명한 사실
  1. 도둑은 무조건 같은 행 or 열 뒤에 경찰을 둬야 함
     - 다음 그림처럼 진행 방향 기준 같은 줄과 그 뒤에 경찰을 둬야 함

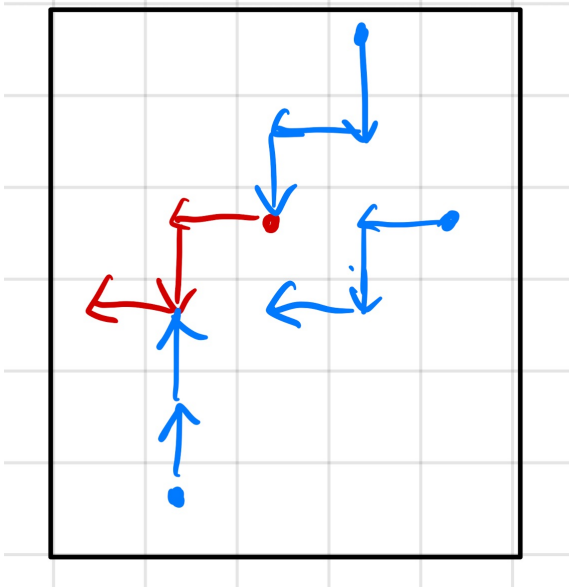
     - 따라서 모든 경찰들을 오른쪽과 같은 상황처럼 만들어야 함 (Escapable 상황)





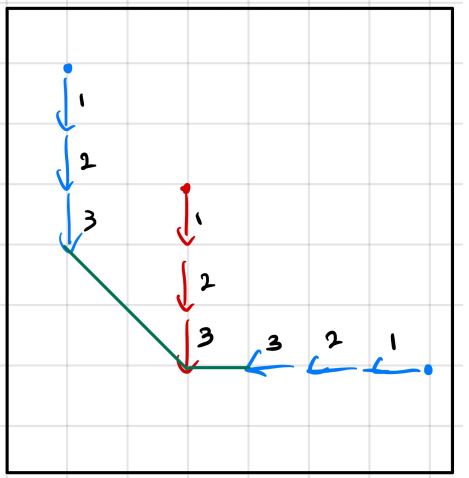
Approach

- 자명한 사실
  2. 도둑은 직선으로 움직이는게 최적
     - 도둑이 직선이 아닌 방향으로 움직이면 경찰도 어차피 그 방향으로 움직일 것
     - 따라서 경찰로부터 벗어나려고 직선이 아니도록 움직여도 경찰과의 거리는 똑같음
     - 오히려 그 방향에 있는 경찰과 거리가 가까워지고 다른 경찰과의 거리는 똑같은 결과 초래



Approach

- Approach 2
  - 따라서 문제는
    - "직선으로 4방향으로 쭉 갔을 때 Escapable 상황을 만들 수 있는가?"
    - 그 방향으로 쭉 갔을 때 일직선을 이루려면 가장 끝에 있는 점과 같은 선상에 있어야 함
    - 가장 위(U), 아래(D), 왼(L), 오(R)로 쭉 갈 때 중간에 경찰에게 안 잡히는 방향이 존재한다면 Escapble 한 것
    - 그 조건은
      - (제일 UDLR로 가는 거리) > (도둑이 최종적으로 도착할 점과 모든 점과의 거리)



Contents

- Problem
- Approach
- Code

Code

- 시작
  - 좌표 구조체
    - 를 만들지 않고 define으로 모두 선언
    - 후에 나오는 x, y를 first, second 등으로 보지 말고 x, y로 보기
  - ABS는 절댓값

```cpp
#include <iostream>

#include <vector>

#define x first
#define y second
#define INF 1'000'000'000
#define ABS(x) (((x) < 0) ? (-(x)) : (x))

using namespace std;

typedef long long ll;
typedef pair<int, int> Coords;
```

Code

- escapable 함수
  - 목표 점과 모든 경찰과의 거리가 distance보다 큰지 판단하는 함수
    - (제일 UDLR로 가는 거리) > (도둑이 최종적으로 도착할 점과 모든 점과의 거리)

  - 좌표가 Taxi Geometry이기 때문에 단순히 x 좌표, y 좌표의 차이로 거리 계산

```
bool escapable(vector<Coords>& polices, Coords target, int distance) {
    for (int i = 0; i < polices.size(); i++) {
        ll policeDistance = (ll)ABS(target.x - polices[i].x) + ABS(target.y - polices[i].y);
        if (policeDistance <= distance)
            return false;
    }

    return true;
}
```

Code

- Input
  - 입력 받기
  - 입력 받으면서 가장 UDRL 점 구하기

```
int main(void) {
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);

    int n;
    cin >> n;

    int up = -INF, down = INF, left = INF, right = -INF;

    vector<Coords> polices (n);
    for (int i = 0; i < n; i++) {
        cin >> polices[i].x >> polices[i].y;

        up = max(up, polices[i].y);
        down = min(down, polices[i].y);
        right = max(right, polices[i].x);
        left = min(left, polices[i].x);
    }

    Coords thief;
    cin >> thief.x >> thief.y;
```

Code

- 초기 조건
  - 도둑이 가장 UDRL 이라면 YES

```
if (up <= thief.y || thief.y <= down || thief.x <= left || right <= thief.x)
    cout << "YES";
```

  - 직선으로 갔을 때 어느 한 점이라도 escapable이 true가 나오면 true

```
else {
    bool ans = false;

    ans |= escapable(polices, { thief.x, up }, ABS(thief.y - up));
    ans |= escapable(polices, { thief.x, down }, ABS(thief.y - down));
    ans |= escapable(polices, { left, thief.y }, ABS(thief.x - left));
    ans |= escapable(polices, { right, thief.y }, ABS(thief.x - right));

    if (ans)
        cout << "YES";
    else
        cout << "NO";
}
```

감사합니다!