8382. 방향 전환

문제를 한번 읽어보자!

정우는 그냥 (x_1, y_1)에서 (x_2, y_2)로 이동하는 것은 재미가 없다고 생각한다.

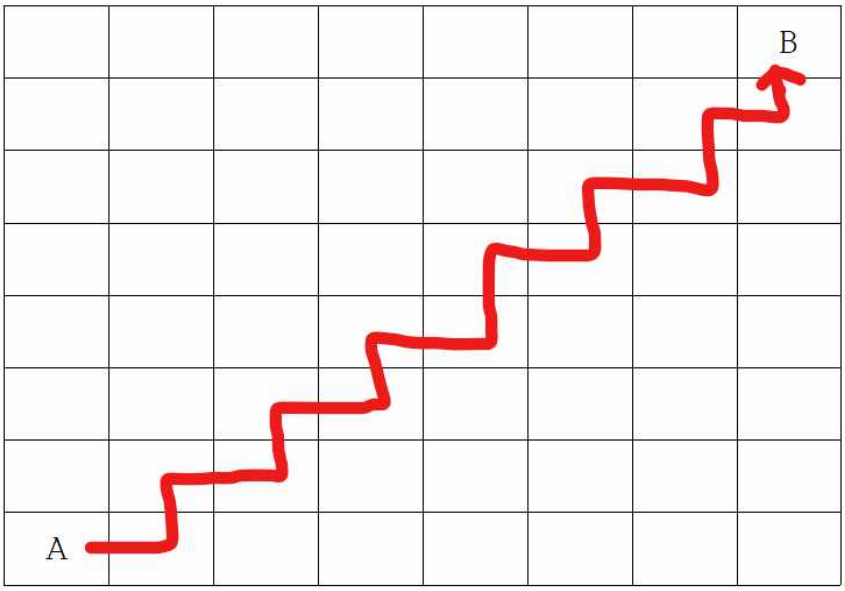
그래서 이전 이동이 가로 이동이었다면, 이번에는 세로 이동으로 이동하고, 이전 이동이 세로 이동이었다면, 이번에는 가로 이동으로 이동하여 (x_1, y_1)에서 (x_2, y_2)로 이동하려고 한다.

정우는 그냥 (x_1, y_1)에서 (x_2, y_2)로 이동하는 것은 재미가 없다고 생각한다.

재미가 없다고 생각한다.

!!!???

정우야 맞을래??



1. 점 A에서 B까지 가기 위해선 세로-가로 혹은 가로-세로 순으로 이동해야 한다.

뭔가 백트래킹을 이용하면 풀 수 있다는 생각이 든다.
물론 가능하다! 하지만 규칙을 찾아낸다면 더욱 빠르게 결과값을 찾을 수 있다!

							14
						12	13
					10	11	
				8	9		
			6	7			
		4	5				
	2	3					
0	1						

2. A에서 B까지 위에 그림처럼 갔을 때, 최소 몇 번 이동 했는지 생각해보자!

가로로 먼저 출발 했을 때, 최소 14번 만에 B에 도착할 수 있다.
다음은 세로로 먼저 출발 했을 경우를 보자

						13	14
					11	12	13
				9	10	11	
			7	8	9		
		5	6	7			
	3	4	5				
1	2	3					
0	1						

(결과 값이 대칭이다..!)

2. 결국 y=x 선을 기준으로 대칭이란 것을 알 수 있다.

이런 식으로 남은 빈칸을 전부 채워보면...

13	14	13	14	13	14	13	14
12	11	12	11	12	11	12	13
9	10	9	10	9	10	11	14
8	7	8	7	8	9	12	13
5	6	5	6	7	10	11	14
4	3	4	5	8	9	12	13
1	2	3	6	7	10	11	14
0	1	4	5	8	9	12	13

이런 규칙을 가진 표가 나오게 된다..!

아닛!?

값이 퍼져나가는 규칙이군!

13	14	13	14	13	14	13	14
12	11	12	11	12	11	12	13
9	10	9	10	9	10	11	14
8	7	8	7	8	9	12	13
5	6	5	6	7	10	11	14
4	3	4	5	8	9	12	13
1	2	3	6	7	10	11	14
0	1	4	5	8	9	12	13

3. 결국 y=x 선은 2씩 증가한다.

f(n) = 2 * n

우린 규칙을 찾았고,

y=x 선을 가지고 결과값을 찾을 수 있게 되었다!!

먼저 y=x선 위에 있는 값들은 전부 날려 버리고 생각해보자!

							14
						12	13
					10	11	14
				8	9	12	13
			6	7	10	11	14
		4	5	8	9	12	13
	2	3	6	7	10	11	14
0	1	4	5	8	9	12	13

4. 선을 기준으로 값이 대칭이기 때문에 결과값은 같다.

원점에서 (3, 2)까지의 결과값은

f(3)에서 -1한 값이라고 볼 수 있다.

원점에서 (3, 1)까지의 결과값은?

f(3) 자체이다.

(y-x)가 짝수면 f(x)-1이고, 홀수면 f(x)이다!

따라서 결과값은,,

2 * x 에서 (x - y)의 나머지를 뺀 값이다!

```cpp
int startX, startY;
int endX, endY;

cin >> startX >> startY;
cin >> endX >> endY;

endX = abs(endX - startX);
endY = abs(endY - startY);

int maxVal = max(endX, endY);
int minVal = min(endX, endY);
int diff = maxVal - minVal;

int result = 2 * maxVal - (diff % 2);

cout << "#" << test_case << " " << result << "\n";
```

바로 이 부분이
2 * x − (x − y) % 2
라능~!

5. 이 식이 성립하려면 좌표조정을 해야한다.

우린 최소 이동 횟수만 찾으면 된다!
따라서 원점을 기준으로
시작점과 도착점의 좌표를 조정해주면
쉽게 결과를 얻을 수 있다.

|bx − ax, by − ay| 하여
A점이 원점일 때, B점의 좌표를 얻을 수 있다.

그리고 한번 더 X 값에 따라 Y 가 바뀌도록 대칭이동한다.

```cpp
int startX, startY;
int endX, endY;

cin >> startX >> startY;
cin >> endX >> endY;

endX = abs(endX - startX);
endY = abs(endY - startY);

int maxVal = max(endX, endY);
int minVal = min(endX, endY);
int diff = maxVal - minVal;

int result = 2 * maxVal - (diff % 2);

cout << "#" << test_case << " " << result << "\n";
```

B 좌표를 원점 기준으로 조정하고,
X 값에 따라 Y가 바뀌도록 조정한다능~

좌표 조정이 끝나면 규칙에 맞게 수식을 만들면 된다!
-끗-

N. 다른 규칙을 이용한 방법

13	14	13	14	13	14	13	14
12	11	12	11	12	11	12	13
9	10	9	10	9	10	11	14
8	7	8	7	8	9	12	13
5	6	5	6	7	10	11	14
4	3	4	5	8	9	12	13
1	2	3	6	7	10	11	14
0	1	4	5	8	9	12	13

y=x 함수와 평행하게 증가하는 다른 규칙도 찾을 수 있다.
이 같은 경우엔

f(n) = f(n−2) + 4
함수를 구현하여 문제를 풀 수도 있다.

다만 재귀로 구현해야 해서 더 느릴 수 있고,
무엇보다 귀찮다..!

```int f(int n) {
    if (n <= 1) return n;

    return f(n - 2) + 4;
}
```

```int startX, startY;
int endX, endY;

cin >> startX >> startY;
cin >> endX >> endY;

endX = abs(endX - startX);
endY = abs(endY - startY);

int maxVal = max(endX, endY);
int minVal = min(endX, endY);
int diff = maxVal - minVal;

//int result = 2 * maxVal - (diff % 2);
int result = f(diff) + 2 * minVal;

cout << "#" << test_case << " " << result << "\n";
```