

알튜비튜

DFS & BFS



오늘은 그래프 탐색 알고리즘인 깊이 우선 탐색(DFS)과 너비 우선 탐색(BFS)을 배웁니다.
앞으로 배울 그래프 알고리즘의 시작이자 코딩테스트에 높은 확률로 한 문제 이상 나오는 알고리즘이
죠.

DFS (깊이 우선 탐색)

- 최대한 깊게 탐색 후 빠져 나옴
- 한 정점을 깊게 탐색해서 빠져 나왔다면, 나머지 정점 계속 동일하게 탐색
- 스택(stack), 재귀함수로 구현

BFS (너비 우선 탐색)

- 자신의 자식들부터 순차적으로 탐색
- 순차 탐색 이후, 다른 정점의 자식들 탐색
- 큐(queue)로 구현

/<> 10026번 : 적록색약 - Gold 5

문제

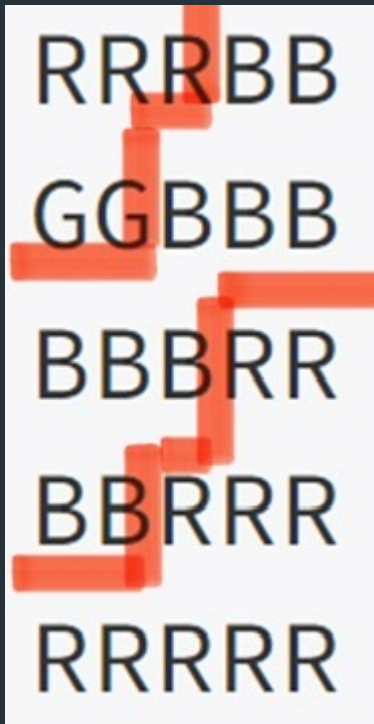
- 적록색약이 보는 구역 개수와 적록색약이 아닌 사람이 보는 구역의 개수를 찾는 문제
- $N \times N$ 각각의 칸에는 R, G, B 중 하나가 쓰여 있음
- 여기서 말하는 구역은 같은 색으로 이루어진 구역
- 상하좌우로 인접해 있는 경우에 두 글자는 같은 구역에 속함

제한 사항

- 세로 크기 & 가로 크기 == N , ($1 \leq N \leq 100$)

/<> 10026번 : 적록색약 - Gold 5

예시



적록색약이 보는 구역 => 3개

빨강과 초록이 인접해 있을 때
구역을 구분하지 못한다!



적록색약이 아닌 사람이 보는 구역 => 4개

/<> 10026번 : 적록색약 - Gold 5

```
RRRBBB
GGBBBB
BBBRRR
BBRRRR
RRRRRR
```

dfs함수로 설계가 가능합니다!

1. 적록색약이 아닌 사람이 구역을 인식하는 경우
2. 적록색약인 사람이 구역을 인식하는 경우

1번의 경우 어느 한 점에서 상하좌우 다른 색이 나올 때까지 탐색을 하면 됩니다.

2번의 경우 비슷하지만, 어느 한 점이 R 혹은 G 일 때 상하좌우 G 나 R이 나와도 계속 탐색을 하면 됩니다.

-> 쉽게 두 가지의 경우를 나누어 dfs 함수를 2개 짜서 구현할 수 있다.

/<> 10026번 : 적록색약 - Gold 5

```
void dfs(int y, int x, char c) {
    for (int i = 0; i < 4; ++i) {
        int ny = y + dy[i];
        int nx = x + dx[i];
        if (ny >= 0 && ny < n && nx >= 0 && nx < n) {
            if (!visit[ny][nx] && graph[ny][nx] == c) {
                visit[ny][nx] = true;
                dfs(ny, nx, c);
            }
        }
    }
}
```

적록 색약이 아닌 사람의 경우

```
void dfs2(int y, int x, char c) {
    for (int i = 0; i < 4; ++i) {
        int ny = y + dy[i];
        int nx = x + dx[i];
        if (ny >= 0 && ny < n && nx >= 0 && nx < n) {
            if (!visit[ny][nx]) {
                if ((c == 'R' || c == 'G') && (graph[ny][nx] == 'R' || graph[ny][nx] == 'G')) {
                    visit[ny][nx] = true;
                    dfs2(ny, nx, c);
                }
            }
            else if (c == 'B') {
                if (graph[ny][nx] == 'B') {
                    visit[ny][nx] = true;
                    dfs2(ny, nx, c);
                }
            }
        }
    }
}
```

적록 색약인 사람의 경우

현재의 위치에 따른 조건문을 추가합니다.

/<> 14502번 : 연구소 - Gold 4

문제

- 연구소는 크기가 $N \times M$ 인 직사각형으로 나타낼 수 있으며, 직사각형은 1×1 크기의 정사각형으로 나누어져 있음
- 연구소는 빈 칸, 벽으로 이루어져 있으며, 일부 칸에 바이러스가 존재
- 벽을 3개 세워서 바이러스가 더이상 퍼질 수 없는 지역(안전지역)의 최대 개수를 구하는 문제

제한 사항

- 세로 크기 N , 가로 크기 M , ($3 \leq N, M \leq 8$)

/<> 14502번 : 연구소 - Gold 4

예시

2	0	0	0	1	1	0
0	0	1	0	1	2	0
0	1	1	0	1	0	0
0	1	0	0	0	0	0
0	0	0	0	0	1	1
0	1	0	0	0	0	0
0	1	0	0	0	0	0

연구소
0: 빈 칸, 1: 벽, 2:바이러스



2	1	0	0	1	1	0
1	0	1	0	1	2	0
0	1	1	0	1	0	0
0	1	0	0	0	1	0
0	0	0	0	0	1	1
0	1	0	0	0	0	0
0	1	0	0	0	0	0

벽을 3개 세운 후



2	1	0	0	1	1	2
1	0	1	0	1	2	2
0	1	1	0	1	2	2
0	1	0	0	0	1	2
0	0	0	0	0	1	1
0	1	0	0	0	0	0
0	1	0	0	0	0	0

바이러스가 퍼진 후

/<> 14502번 : 연구소 - Gold 4

문제 풀이 방법

- 1. 빈 칸(0)에 벽 3개 세울 수 있는 경우 생각
- 2. 벽 3개를 세우고, 해당 경우 바이러스가 퍼질 수 있는 지역 bfs로 탐색
- 3. 바이러스가 퍼지고 나서 안전지대 탐색

/<> 14502번 : 연구소 - Gold 4

문제 풀이 방법 ● 1. 빈 칸(0)에 벽 3개 세울 수 있는 경우 생각

```
void wall(int cnt) {
    if (cnt == 3) {
        bfs(); //벽을 3개 세우면 안전지대 탐색
        return;
    }
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            if (li[i][j] == 0) {
                li[i][j] = 1;
                wall(cnt + 1);
                li[i][j] = 0;
            }
        }
    }
}
```

/<> 14502번 : 연구소 - Gold 4

문제 풀이 방법 ● 2. 벽 3개를 세우고, 해당 경우 바이러스가 퍼질 수 있는 지역 bfs로 탐색

```
void bfs() {
    queue<pair<int, int>> q;
    vector<vector<int>> tmp_map = li;
    for (auto& v : virus) {
        q.push(v);
    }

    while (!q.empty()) {
        int x = q.front().first;
        int y = q.front().second;
        q.pop();

        for (int i = 0; i < 4; i++) {
            int newx = x + dx[i];
            int newy = y + dy[i];
            if (0 <= newx && newx < n && 0 <= newy && newy < m && tmp_map[newx][newy] == 0) {
                tmp_map[newx][newy] = 2;
                q.push(make_pair(newx, newy));
            }
        }
    }
    checkSafeZone(tmp_map);
}
```

/<> 14502번 : 연구소 - Gold 4

문제 풀이 방법 ● 3. 바이러스가 퍼지고 나서 안전지대 탐색

```
void checkSafeZone(const vector<vector<int>>& tmp_map) { //바이러스가 퍼지지 않은 지역 계산
    int count = 0;
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            if (tmp_map[i][j] == 0) {
                count++;
            }
        }
    }
    ans = max(ans, count);
}
```

/<> 2615번: 오목 - Silver 1

문제

- 바둑판의 상태가 주어졌을 때, 검은색이 이겼는지, 흰색이 이겼는지 또는 아직 승부가 결정되지 않았는지를 판단하는 문제
- 한 방향으로 같은 색의 바둑알이 연속적으로 다섯 알 놓이면 승리
- 단, 여섯 알 이상이 연속적으로 놓인 경우는 이긴 것이 아님
- 방향은 가로, 세로, 대각선 모두 포함

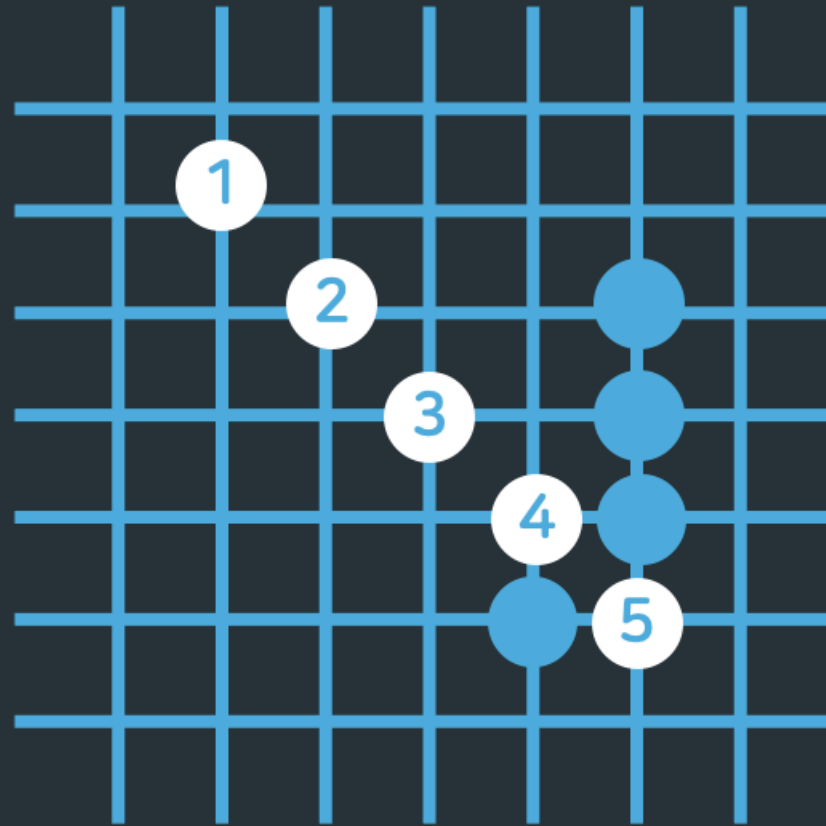
제한 사항

- 바둑판의 크기는 19 x 19
- 검은 바둑알은 1, 흰 바둑알은 2, 알이 놓이지 않는 자리는 0으로 표시 (한 칸씩 띄어서 표시됨)
- 둘 중 하나가 이겼을 경우, 연속된 다섯 개의 바둑알 중 가장 왼쪽 위에 있는 바둑알의 좌표를 출력

예제 출력

[illegible]

1
3 2



- ① 연속된 다섯 개의 바둑알 중에서 가장 왼쪽 위에 있는 바둑알을 출력해야 해요.
어느 위치부터 어떤 방향으로 탐색을 진행해야 할까요?
- ② 어떻게 여섯 개의 바둑알이 연속적으로 놓이는 경우를 제외할 수 있을까요?

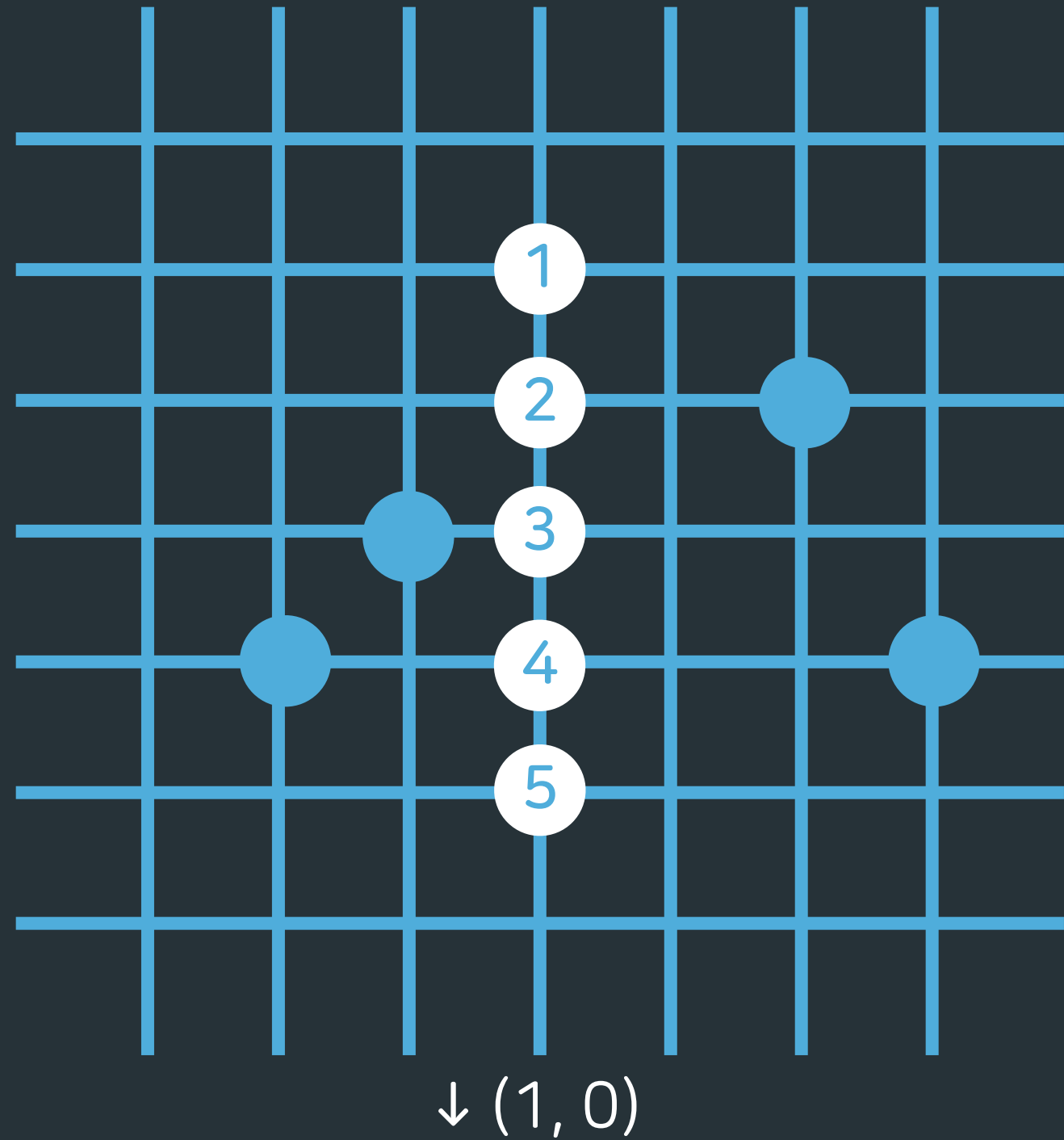
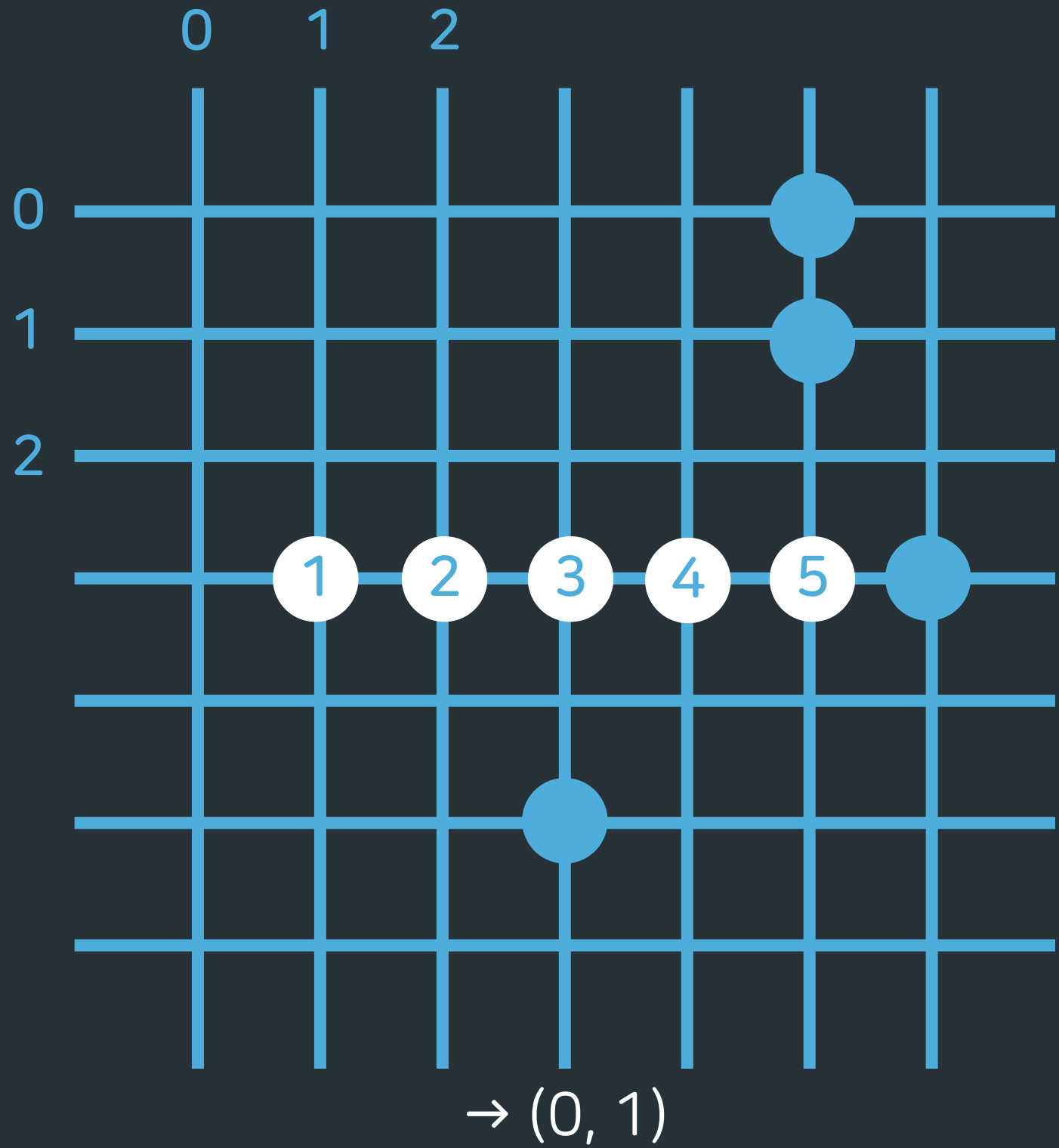
- 오목에 성공했을 시 가장 왼쪽 위에 있는 바둑알을 출력
→ 탐색 방향은 아래 4가지뿐

- $\rightarrow (0, 1) : \{0, 0\}, \{0, 1\}, \{0, 2\}, \{0, 3\}, \{0, 4\}, \{0, 5\}, \dots$
- $\downarrow (1, 0) : \{0, 0\}, \{1, 0\}, \{2, 0\}, \{3, 0\}, \{4, 0\}, \{5, 0\}, \dots$
- $\searrow (1, 1) : \{0, 0\}, \{1, 1\}, \{2, 2\}, \{3, 3\}, \{4, 4\}, \{5, 5\}, \dots$
- $\nearrow (-1, 1) : \{0, 0\}, \{-1, 1\}, \{-2, 2\}, \{-3, 3\}, \{-4, 4\}, \{-5, 5\}, \dots$

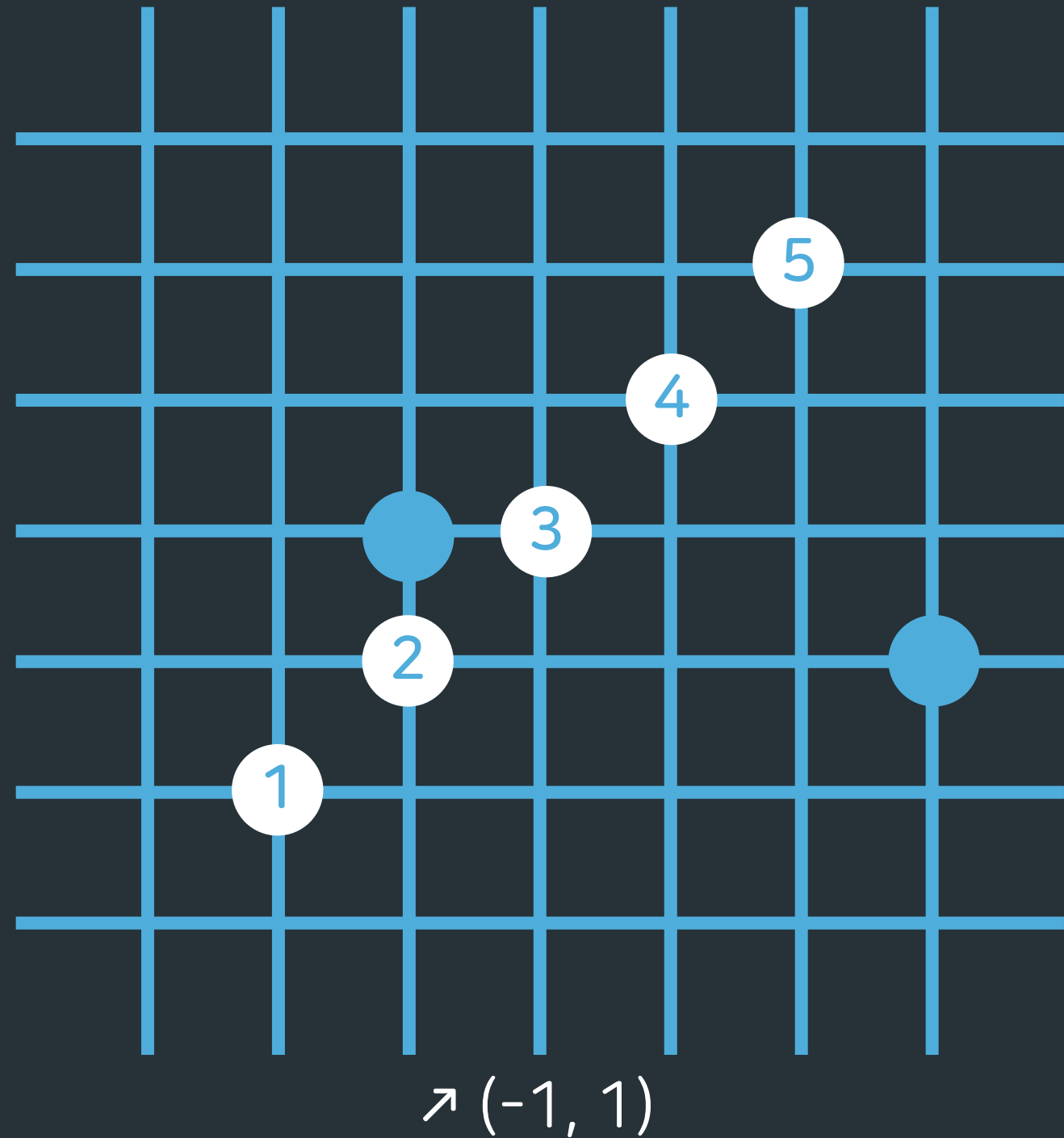
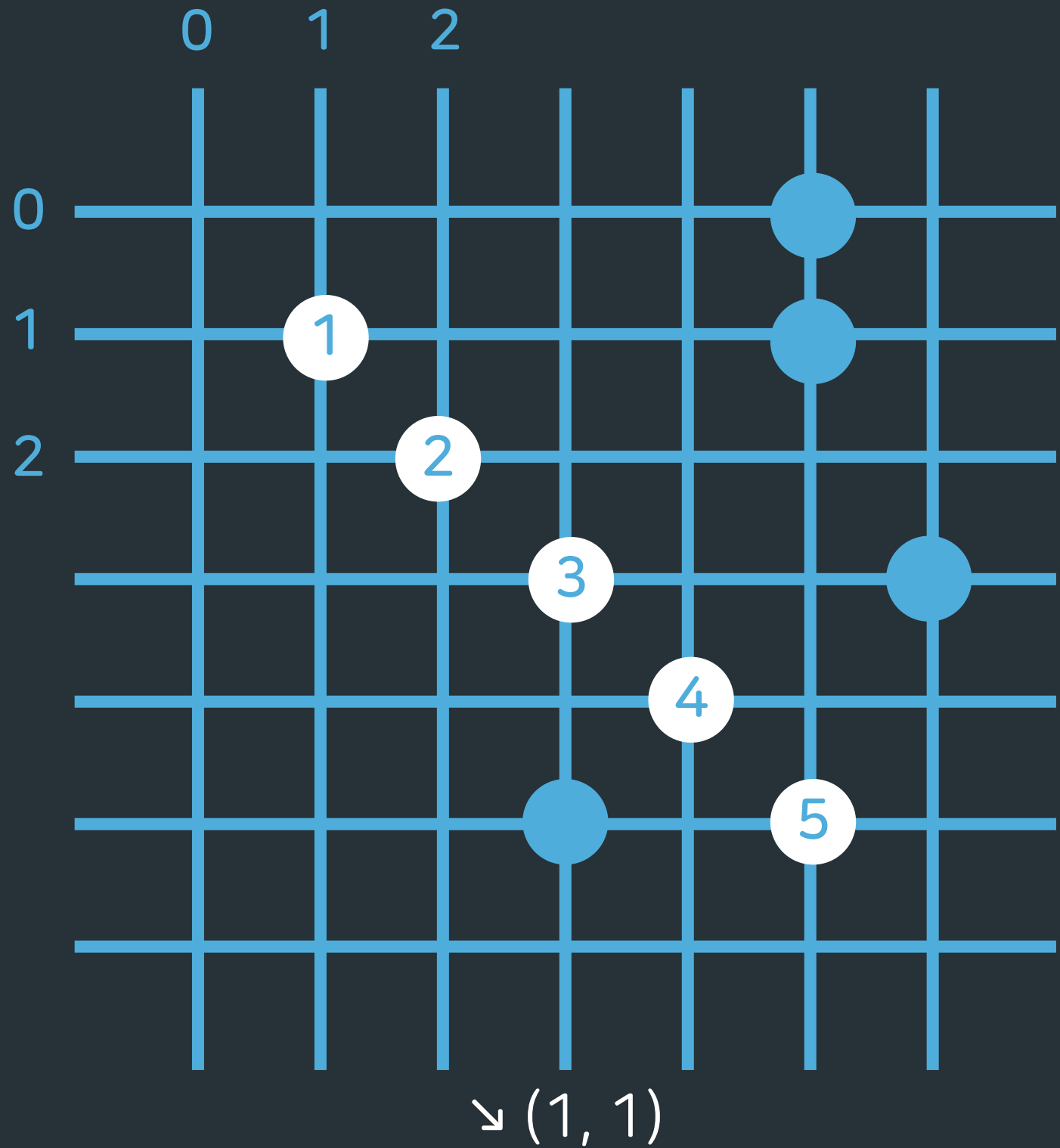
⇒ 방법 1: 브루트 포스

⇒ 방법 2: DFS

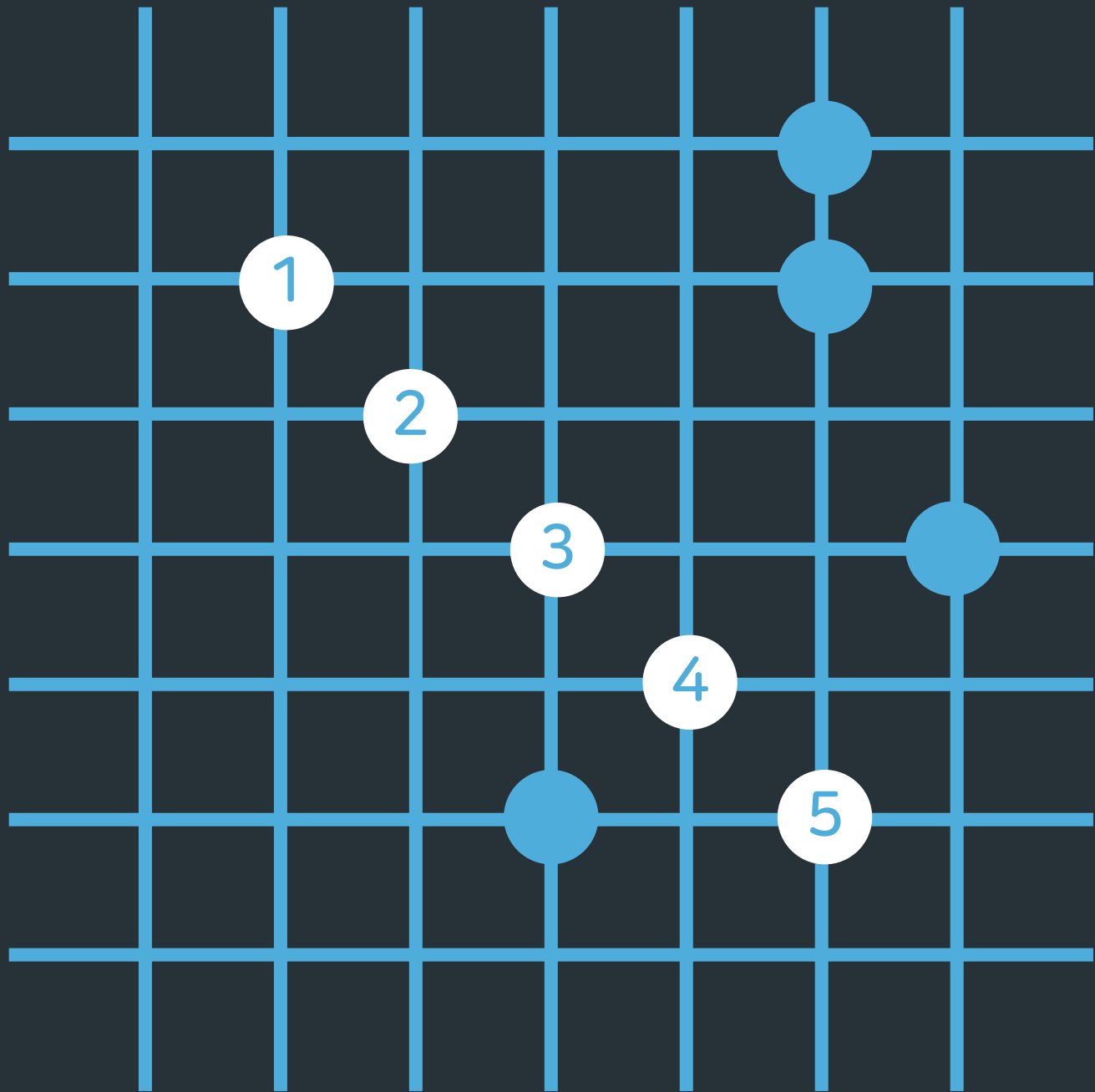
접근 방법



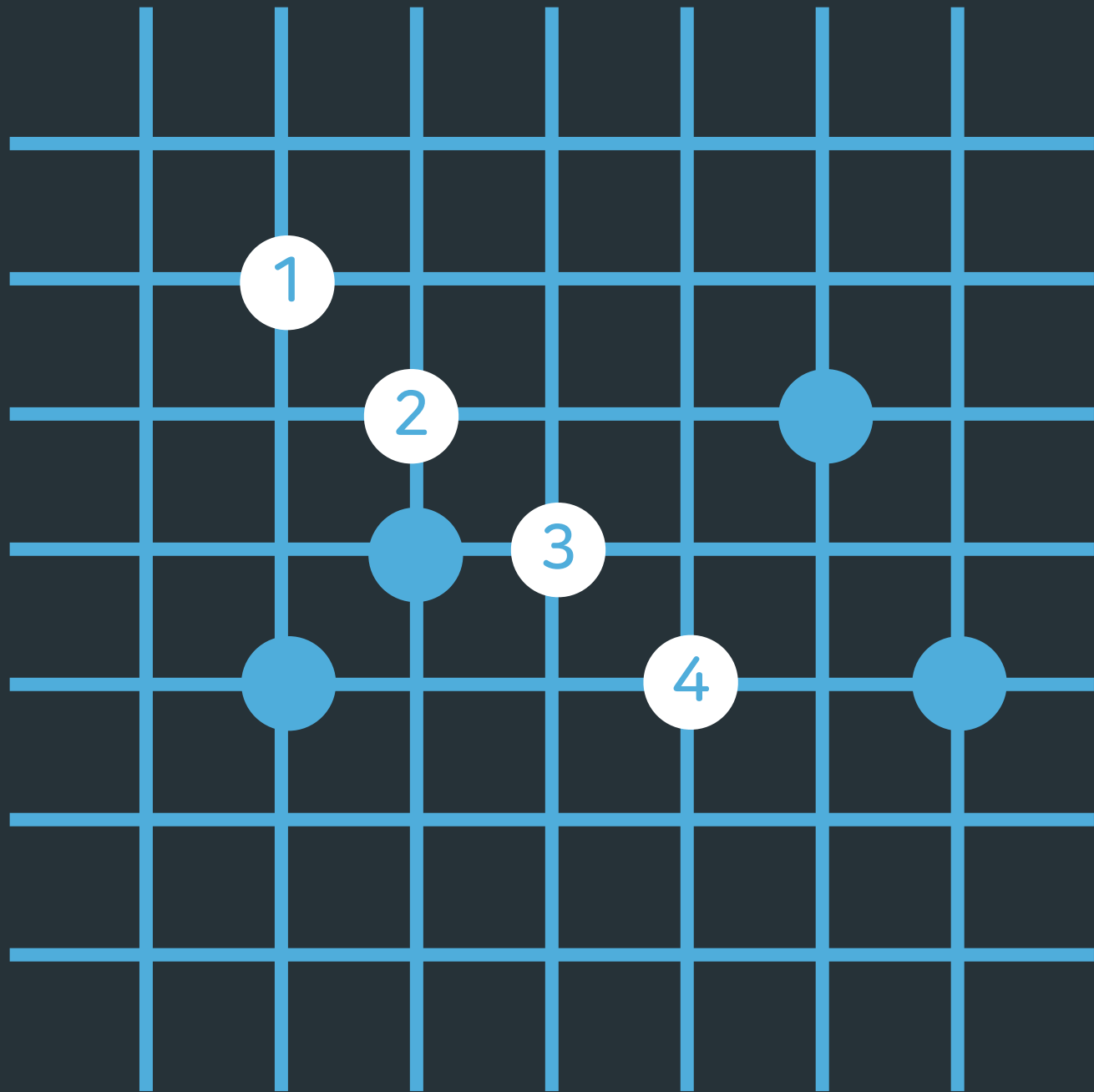
접근 방법



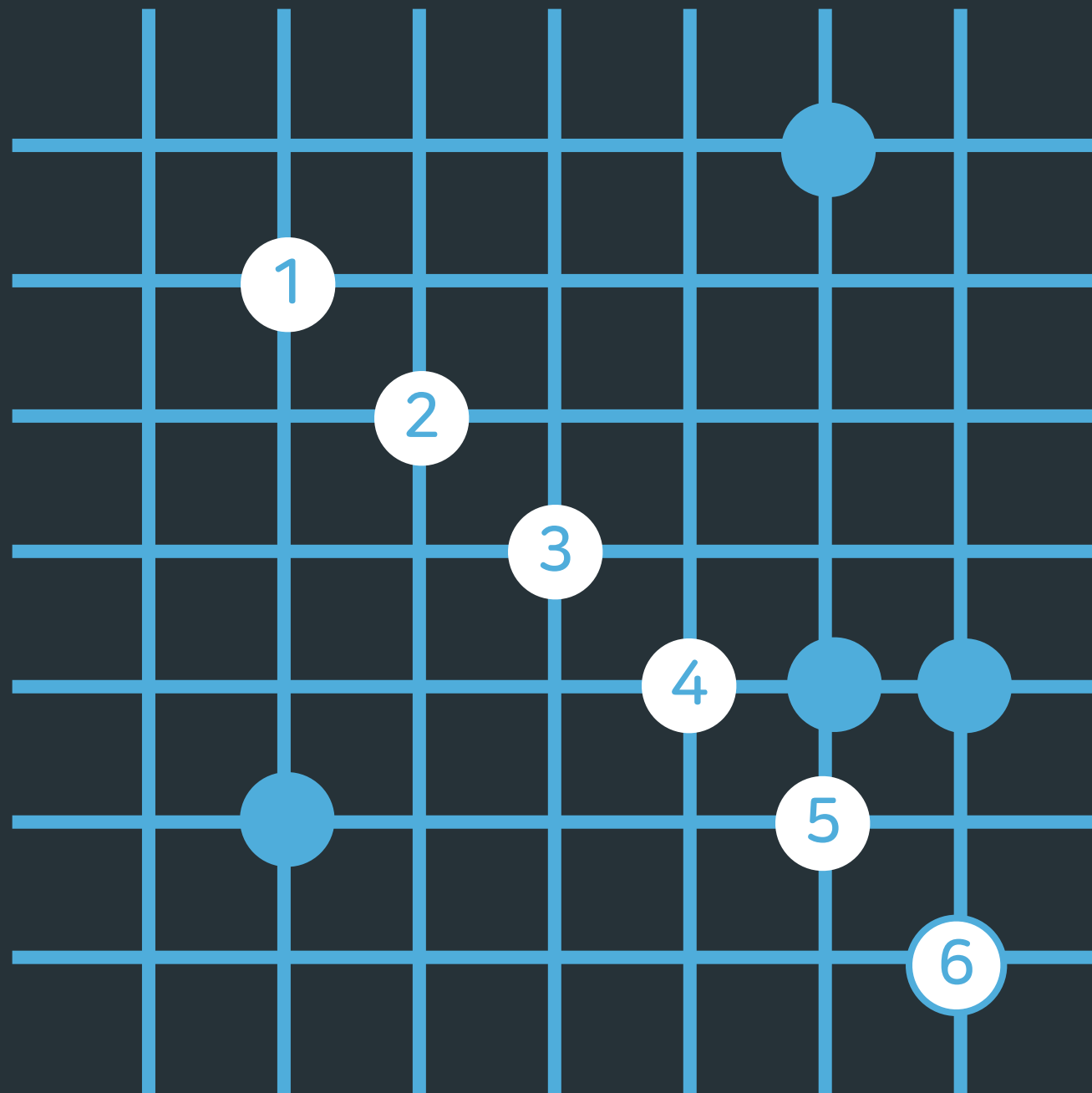
- 탐색하는 방향으로 바둑알이 6개 이상 존재하는 경우 오목 실패
→ 다섯 알까지만 연속한지 확인하고, 여섯 알 째는 연속하지 않는지를 확인
- 탐색 방향으로 5알이 연속이면 무조건 승리?
🙅 탐색 반대 방향으로 같은 색의 바둑알이 있으면 실패!



성공

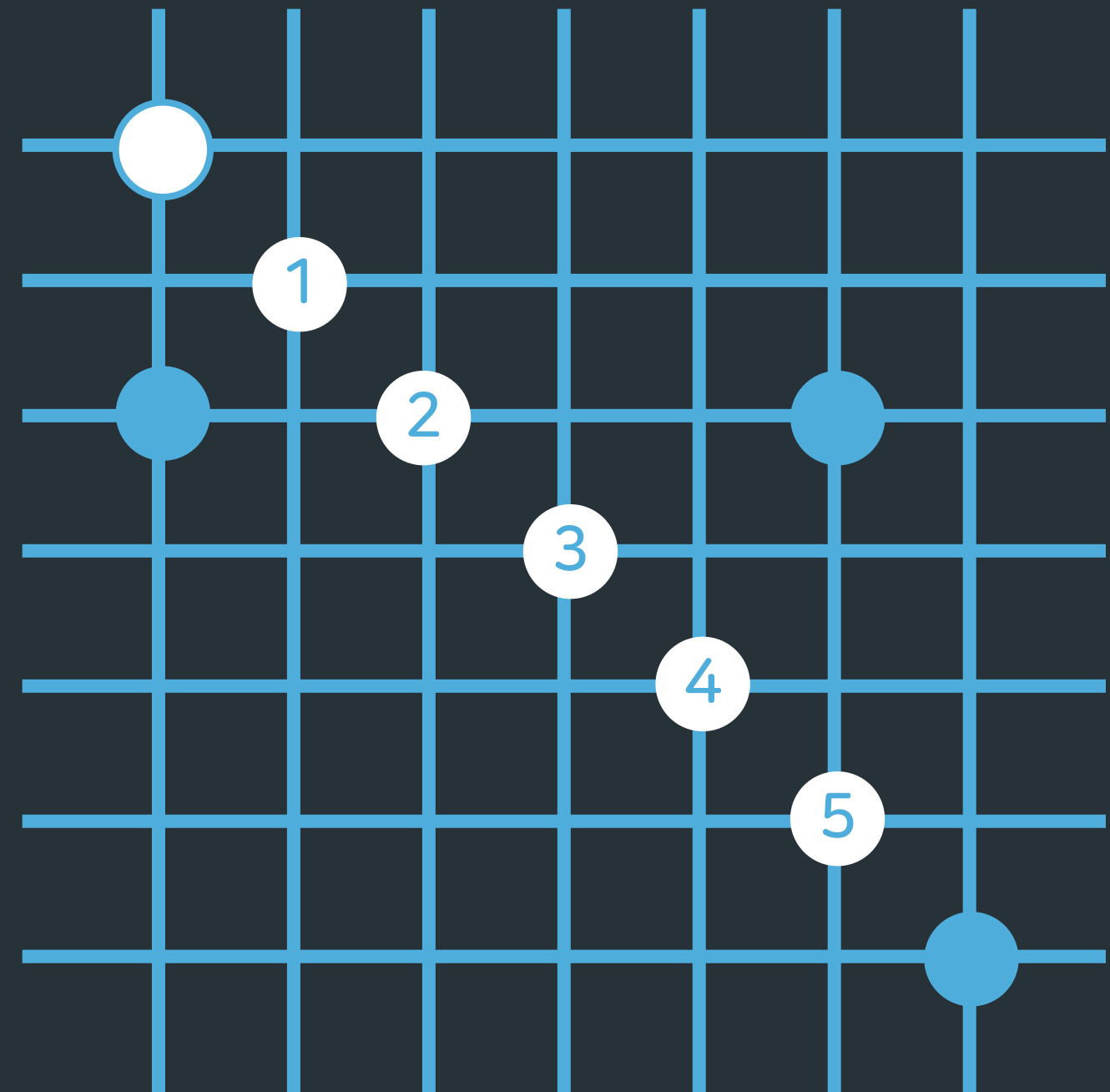


실패
: 4알 연속



실패

: 탐색 방향으로 6알 이상 연속



실패

: 탐색 반대 방향으로 6알 이상 연속

추가로 풀어보면 좋은 문제!

/<> 1012번 : 유기농 배추 - Silver 2

/<> 11725번 : 트리의 부모 찾기 - Silver 2

/<> 1697번 : 숨바꼭질 - Silver 1

/<> 1520번 : 내리막 길 - Gold 3