

# BOJ 12851 숨바꼭질2 심층 분석

```

private static int N;
private static int[] dr = {1,-1, 0, 0};
private static int[] dc = {0, 0, 1,-1};
public static void generalBFS() {
    // 큐와 방문체크 배열 생성
    LinkedList<int[]> queue = new LinkedList<>();
    boolean[][] visited = new boolean[N][N];

    // 탐색 시작 좌표를 queue에 담고 방문처리
    queue.offer(new int[] {0, 0});
    visited[0][0] = true;

    // bfs 탐색
    while(queue.isEmpty()) {
        int[] current = queue.poll();

        /*
         * bfs 탐색의 종료 조건이 있다면 이 위치에 들어감
         */

        // 4방 탐색
        for (int i = 0; i < 4; i++) {
            int nr = current[0] + dr[i];
            int nc = current[1] + dc[i];
            // 방문하지 않은 곳이라면 큐에 담고 방문처리
            if(0<=nr && nr<N && 0<=nc && nc<N && !visited[nr][nc]) {
                queue.offer(new int[] {nr,nc});
                visited[nr][nc] = true;
            }
        }
    }
} // end of method bfs

```

큐에 넣을 때  
방문처리

일반적인 BFS 틀  
(queue에 추가할 때 중복 **완전** 불허)

```

private static int N;
private static int[] dr = {1,-1, 0, 0};
private static int[] dc = {0, 0, 1,-1};
public static void restrictedBFS() {
    // 큐와 방문체크 배열 생성
    LinkedList<int[]> queue = new LinkedList<>();
    boolean[][] visited = new boolean[N][N];

    // 탐색 시작 좌표를 queue에 담기
    queue.offer(new int[] {0, 0});

    // bfs 탐색
    while(queue.isEmpty()) {
        int[] current = queue.poll();
        visited[current[0]][current[1]] = true; // 큐에서 꺼낼 때 방문처리

        // 4방 탐색
        for (int i = 0; i < 4; i++) {
            int nr = current[0] + dr[i];
            int nc = current[1] + dc[i];
            // 방문하지 않은 곳이라면 큐에 담음
            if(0<=nr && nr<N && 0<=nc && nc<N && !visited[nr][nc]) {
                queue.offer(new int[] {nr,nc});
            }
        }
    }
} // end of method restrictedBFS

```

큐에서 뺄 때  
방문처리

비일반적인 BFS 틀  
(queue에 추가할 때 중복 **일부** 허용)

```

private static int N;
private static int[] dr = {1, -1, 0, 0};
private static int[] dc = {0, 0, 1, -1};
public static void generalBFS() {
    // 큐와 방문체크 배열 생성
    LinkedList<int[]> queue = new LinkedList<>();
    boolean[][] visited = new boolean[N][N];

    // 탐색 시작 좌표를 queue에 담고 방문처리
    queue.offer(new int[] {0, 0});
    visited[0][0] = true;

    // bfs 탐색
    while(!queue.isEmpty()) {
        int[] current = queue.poll();

        /*
         * bfs 탐색의 종료 조건이 있다면 이 위치에 들어감
         */

        // 4방 탐색
        for (int i = 0; i < 4; i++) {
            int nr = current[0] + dr[i];
            int nc = current[1] + dc[i];
            // 방문하지 않은 곳이라면 큐에 담고 방문처리
            if(0 <= nr && nr < N && 0 <= nc && nc < N && !visited[nr][nc]) {
                queue.offer(new int[] {nr, nc});
                visited[nr][nc] = true;
            }
        }
    }
} // end of method bfs

```

큐에 넣을 때  
방문처리

일반적인 BFS 틀  
(queue에 추가할 때 중복 완전 불허)

상좌하우 순으로 탐색한다고 가정

		2		
	2	1	2	
2	1	0	1	
	2	1		

두 번째 레벨에서 '좌' 위치에서 4방  
탐색 시 이 위치를 방문하지 않음  
(‘상’ 위치에서 탐색할 때 방문처리  
했기 때문)

```

private static int N;
private static int[] dr = {1,-1, 0, 0};
private static int[] dc = {0, 0, 1,-1};
public static void restrictedBFS() {
    // 큐와 방문체크 배열 생성
    LinkedList<int[]> queue = new LinkedList<>();
    boolean[][] visited = new boolean[N][N];

    // 탐색 시작 좌표를 queue에 담기
    queue.offer(new int[] {0, 0});

    // bfs 탐색
    while(queue.isEmpty()) {
        int[] current = queue.poll();
        visited[current[0]][current[1]] = true; // 큐에서 꺼낼 때 방문처리

        // 4방 탐색
        for (int i = 0; i < 4; i++) {
            int nr = current[0] + dr[i];
            int nc = current[1] + dc[i];
            // 방문하지 않은 곳이라면 큐에 담음
            if(0<=nr && nr<N && 0<=nc && nc<N && !visited[nr][nc]) {
                queue.offer(new int[] {nr,nc});
            }
        }
    }
} // end of method restrictedBFS

```

큐에서 뺄 때  
방문처리

상좌하우 순으로 탐색한다고 가정

		2		
	2	1	2	
2	1	0	1	
	2	1		

두 번째 레벨에서 '좌' 위치에서  
4방탐색 시 이 위치를 **중복 방문**  
함('상' 위치에서 탐색할 때 방문  
처리를 **아직** 하지 않음)

비일반적인 BFS 틀  
(queue에 추가할 때 중복 일부 허용)

## 결론

BFS 탐색 시, 큐에서 뺄 때 방문처리를 하게 되면  
**동일한 레벨(너비)**의 탐색에서 **중복 방문**하는 경우가 발생

## 문제

DFS는 고려 X (StackOverflow 예상됨)

수빈이는 동생과 숨바꼭질을 하고 있다. 수빈이는 현재 점  $N$  ( $0 \leq N \leq 100,000$ )에 있고, 동생은 점  $K$  ( $0 \leq K \leq 100,000$ )에 있다. 수빈이는 걷거나 순간이동을 할 수 있다. 만약, 수빈이의 위치가  $X$ 일 때 걷는다면 1초 후에  $X-1$  또는  $X+1$ 로 이동하게 된다. 순간이동을 하는 경우에는 1초 후에  $2 \times X$ 의 위치로 이동하게 된다.

수빈이와 동생의 위치가 주어졌을 때, 수빈이가 동생을 찾을 수 있는 가장 빠른 시간이 몇 초 후인지 그리고, 가장 빠른 시간으로 찾는 방법이 몇 가지인지 구하는 프로그램을 작성하시오.

문제 상황 이해 및 해석

$\{X-1, X+1, 2 \times X\}$  이동 방식을 통해, 최종 목적지인 동생의 좌표로 이동(탐색)하는 게 목표

## 입력

첫 번째 줄에 수빈이가 있는 위치  $N$ 과 동생이 있는 위치  $K$ 가 주어진다.  $N$ 과  $K$ 는 정수이다.

## 출력

첫째 줄에 수빈이가 동생을 찾는 가장 빠른 시간을 출력한다.

둘째 줄에는 가장 빠른 시간으로 수빈이가 동생을 찾는 방법의 수를 출력한다.

**"BFS"**

(단지, 1차원 탐색일 뿐)

가장 빠른 시간만 찾는다면, BFS 탐색이 완료 됐을 때의 시간만 출력하고 끝내면 되지만, 그 '방법의 수' 또한 구해야하므로 추가적으로 생각할 거리가 필요할 것임을 예측 가능



**"비일반적 BFS 활용"**



```

public static void bfs(int N, int K) {
    LinkedList<int[]> queue = new LinkedList<>();
    cnt = 0; // 방법의 수
    min = Integer.MAX_VALUE;
    int[] current;
    boolean[] visited = new boolean[100001]; // 애초에 100000을 넘어가면 손해이므로, queue에 담을 때 이 값을 넘지 않는 선에서 탐색

    // 현재 수빈이의 위치 큐에 담기
    queue.offer(new int[] {N, 0}); // 위치, 시간정보

    // bfs 탐색
    while(!queue.isEmpty()) {
        current = queue.poll();
        visited[current[0]] = true; // queue에서 뺄 때 방문처리

        if(current[0] == K) { // 현재 위치가 K이면 최소 시간 갱신
            if(current[1] == min) cnt++;
            if(current[1] < min) {
                min = current[1];
                cnt = 1;
            }
        }

        // 세 방향(경우)으로 이동 가능
        int dx1 = current[0]+1;
        int dx2 = current[0]-1;
        int dx3 = 2*current[0];

        // 이동한 위치를 방문 안 했다면 queue에 담기
        if(dx1 <= 100000 && !visited[dx1] && current[1] < min) queue.offer(new int[] {dx1, current[1]+1}); // 갱신된 최소 시간보다 작아야 함 (백트래킹)
        if(dx2 >= 0 && !visited[dx2] && current[1] < min) queue.offer(new int[] {dx2, current[1]+1}); // 갱신된 최소 시간보다 작아야 함 (백트래킹)
        if(dx3 <= 100000 && !visited[dx3] && current[1] < min) queue.offer(new int[] {dx3, current[1]+1}); // 갱신된 최소 시간보다 작아야 함 (백트래킹)
    }
}

} // end of method bfs

```

숨바꼭질2에 적용된 비일반적인 BFS 코드  
(queue에 추가할 때 중복 일부 허용)