Chapter 02.
BFS - 너비우선 탐색

Clip 01 | [2178] 미로 탐색
4방향 탐색과 BFS

Clip 02 | [1697] 숨바꼭질
배열의 인덱스를 이용한 BFS

Clip 03 | [12851] 숨바꼭질2
조건이 추가된 숨바꼭질

Clip 04 | [7562] 나이트의 이동
8방향 탐색과 BFS

Clip 05 | [7576] 토마토
가장 깊은 깊이의 탐색

Clip 06 | [5427] 불
장애물이 움직이는 공간에서의 탐색

Clip 07 | [9019] DSLR
연산의 종류별 탐색

Clip 08 | [2206] 벽 부수고 이동하기
특수한 이동이 1회 포함된 탐색

Clip 09 | [14442] 벽 부수고 이동하기 2
특수한 이동이 여러 번 포함된 탐색

Clip 10 | [1194] 달이 차오른다, 가자
장애물이 변동되는 공간에서의 탐색

Ch02. BFS – 너비우선 탐색

1. [2178] 미로 탐색

BOJ2178: 미로탐색

문제 요약

- N*M 크기의 배열로 표현되는 미로
- (1, 1)에서 출발해서 (N, M)으로 도착해야 한다
- 지나가야 하는 최소 칸 수를 출력
- 2 <= N, M <= 100

BOJ2178: 미로탐색

문제 분석

- 최대 사이즈가 100 * 100

- 상 / 하 / 좌 / 우 이동하며
  1. 이동 가능한 칸 인지
  2. 방문한적이 없는지 확인

- 좌표를 객체로 묶어서 관리하면 편리하다

BOJ2178: 미로탐색

좌표 탐색

- dr[] = {-1 , 0, 1, 0}
- dc[] = {0, 1, 0, -1}

- 현재 좌표가 r, c 라면?

- 다음 탐색할 좌표: {r + dr[i]}, {c + dc[i]}

BOJ2178: 미로탐색

BFS

- 너비 우선 탐색

- 현재 위치로부터 인접한 정점부터 방문

- 어떻게 구현해야 할까?
  - 큐를 이용하면 먼저 찾은 정점을 먼저 탐색할 수 있다

BOJ2178: 미로탐색

BFS와 큐

1 (0, 0)	0 (0, 1)	1 (0, 2)	1 (0, 3)
1 (1, 0)	1 (1, 1)	1 (1, 2)	0 (1, 3)
1 (2, 0)	0 (2, 1)	1 (2, 2)	0 (2, 3)
1 (3, 0)	1 (3, 1)	1 (3, 2)	1 (3, 3)

BOJ2178: 미로탐색

BFS와 큐

1 (0, 0)	0 (0, 1)	1 (0, 2)	1 (0, 3)
1 (1, 0)	1 (1, 1)	1 (1, 2)	0 (1, 3)
1 (2, 0)	0 (2, 1)	1 (2, 2)	0 (2, 3)
1 (3, 0)	1 (3, 1)	1 (3, 2)	1 (3, 3)

BOJ2178: 미로탐색

BFS와 큐

1 (0, 0)	0 (0, 1)	1 (0, 2)	1 (0, 3)
1 (1, 0)	1 (1, 1)	1 (1, 2)	0 (1, 3)
1 (2, 0)	0 (2, 1)	1 (2, 2)	0 (2, 3)
1 (3, 0)	1 (3, 1)	1 (3, 2)	1 (3, 3)

(0, 0) 에서
갈 수 있는 좌표를
큐에 넣기

BOJ2178: 미로탐색

BFS와 큐

1 (0, 0)	0 (0, 1)	1 (0, 2)	1 (0, 3)
1 (1, 0)	1 (1, 1)	1 (1, 2)	0 (1, 3)
1 (2, 0)	0 (2, 1)	1 (2, 2)	0 (2, 3)
1 (3, 0)	1 (3, 1)	1 (3, 2)	1 (3, 3)

(0, 0) 에서
갈 수 있는 좌표를
큐에 넣기

BOJ2178: 미로탐색

BFS와 큐

1 (0, 0)	0 (0, 1)	1 (0, 2)	1 (0, 3)
1 (1, 0)	1 (1, 1)	1 (1, 2)	0 (1, 3)
1 (2, 0)	0 (2, 1)	1 (2, 2)	0 (2, 3)
1 (3, 0)	1 (3, 1)	1 (3, 2)	1 (3, 3)

BOJ2178: 미로탐색

BFS와 큐

1 (0, 0)	0 (0, 1)	1 (0, 2)	1 (0, 3)
1 (1, 0)	1 (1, 1)	1 (1, 2)	0 (1, 3)
1 (2, 0)	0 (2, 1)	1 (2, 2)	0 (2, 3)
1 (3, 0)	1 (3, 1)	1 (3, 2)	1 (3, 3)

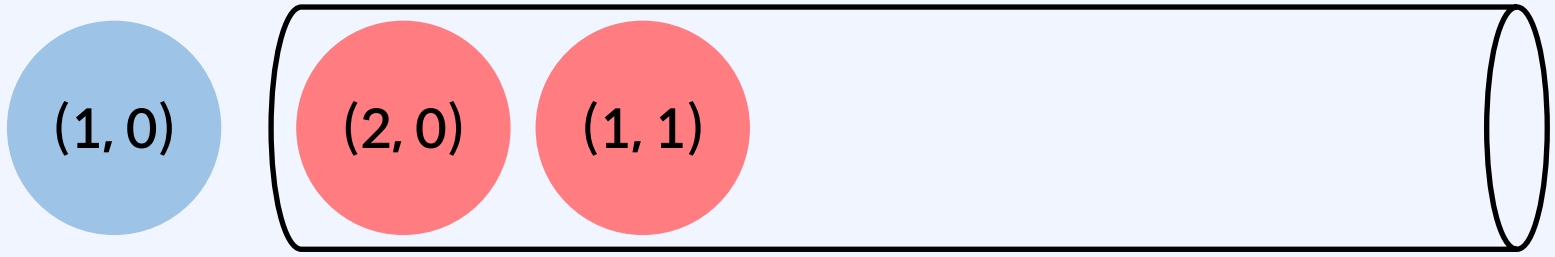
(1, 0) 에서
갈 수 있는 좌표를
큐에 넣기

BOJ2178: 미로탐색

BFS와 큐

1 (0, 0)	0 (0, 1)	1 (0, 2)	1 (0, 3)
1 (1, 0)	1 (1, 1)	1 (1, 2)	0 (1, 3)
1 (2, 0)	0 (2, 1)	1 (2, 2)	0 (2, 3)
1 (3, 0)	1 (3, 1)	1 (3, 2)	1 (3, 3)

(1, 0) 에서
갈 수 있는 좌표를
큐에 넣기

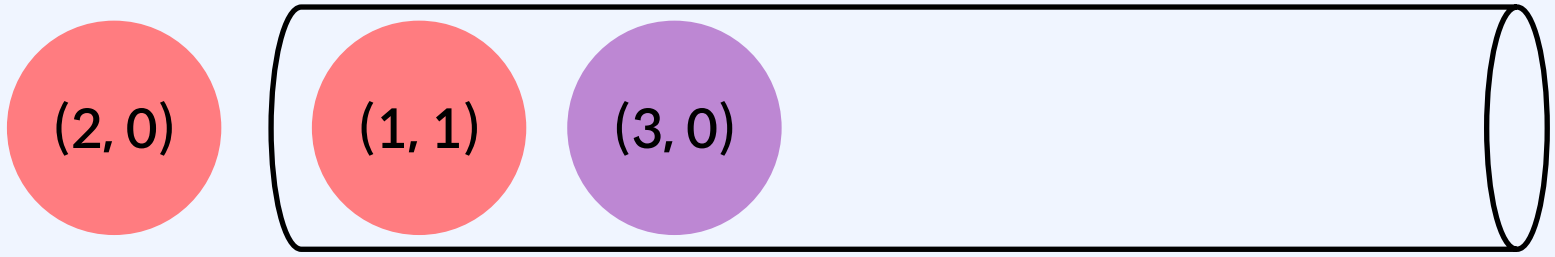


BOJ2178: 미로탐색

BFS와 큐

1 (0, 0)	0 (0, 1)	1 (0, 2)	1 (0, 3)
1 (1, 0)	1 (1, 1)	1 (1, 2)	0 (1, 3)
1 (2, 0)	0 (2, 1)	1 (2, 2)	0 (2, 3)
1 (3, 0)	1 (3, 1)	1 (3, 2)	1 (3, 3)

(2, 0) 에서
갈 수 있는 좌표를
큐에 넣기

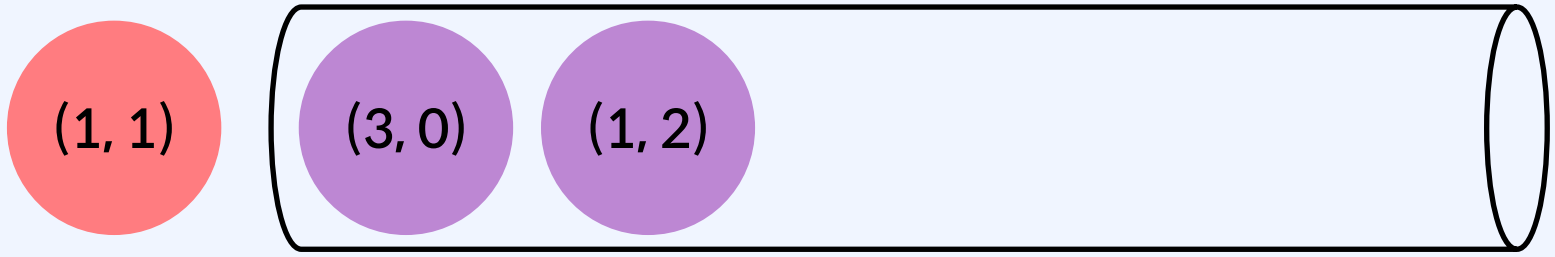


BOJ2178: 미로탐색

BFS와 큐

1 (0, 0)	0 (0, 1)	1 (0, 2)	1 (0, 3)
1 (1, 0)	1 (1, 1)	1 (1, 2)	0 (1, 3)
1 (2, 0)	0 (2, 1)	1 (2, 2)	0 (2, 3)
1 (3, 0)	1 (3, 1)	1 (3, 2)	1 (3, 3)

(1, 1) 에서
갈 수 있는 좌표를
큐에 넣기



BOJ2178: 미로탐색

BFS와 큐

1 (0, 0)	0 (0, 1)	1 (0, 2)	1 (0, 3)
1 (1, 0)	1 (1, 1)	1 (1, 2)	0 (1, 3)
1 (2, 0)	0 (2, 1)	1 (2, 2)	0 (2, 3)
1 (3, 0)	1 (3, 1)	1 (3, 2)	1 (3, 3)

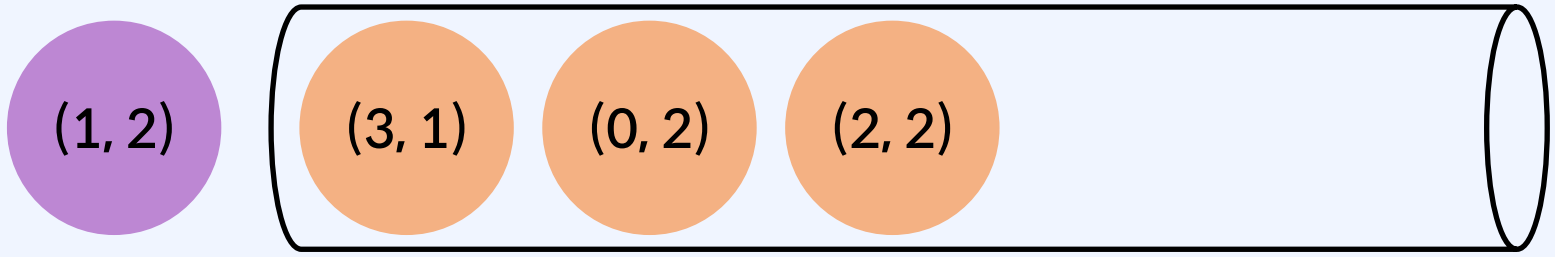
(3, 0) 에서
갈 수 있는 좌표를
큐에 넣기

BOJ2178: 미로탐색

BFS와 큐

1 (0, 0)	0 (0, 1)	1 (0, 2)	1 (0, 3)
1 (1, 0)	1 (1, 1)	1 (1, 2)	0 (1, 3)
1 (2, 0)	0 (2, 1)	1 (2, 2)	0 (2, 3)
1 (3, 0)	1 (3, 1)	1 (3, 2)	1 (3, 3)

(1, 2) 에서
갈 수 있는 좌표를
큐에 넣기

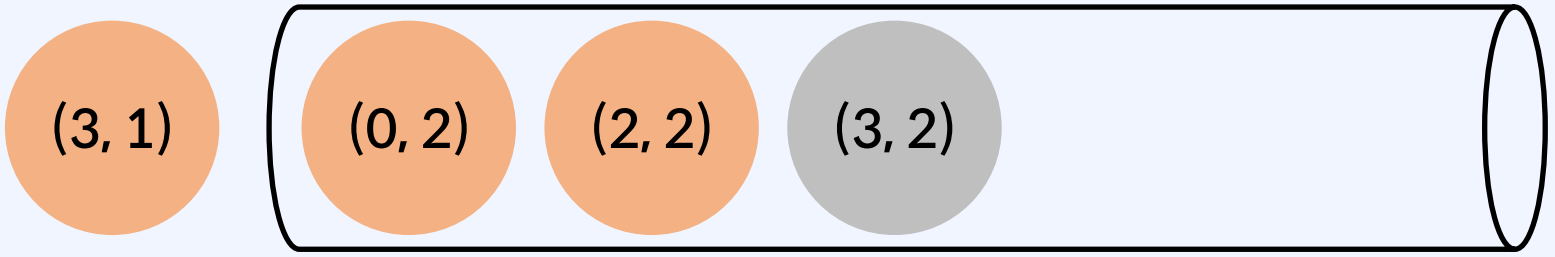


BOJ2178: 미로탐색

BFS와 큐

1 (0, 0)	0 (0, 1)	1 (0, 2)	1 (0, 3)
1 (1, 0)	1 (1, 1)	1 (1, 2)	0 (1, 3)
1 (2, 0)	0 (2, 1)	1 (2, 2)	0 (2, 3)
1 (3, 0)	1 (3, 1)	1 (3, 2)	1 (3, 3)

(3, 1) 에서
갈 수 있는 좌표를
큐에 넣기

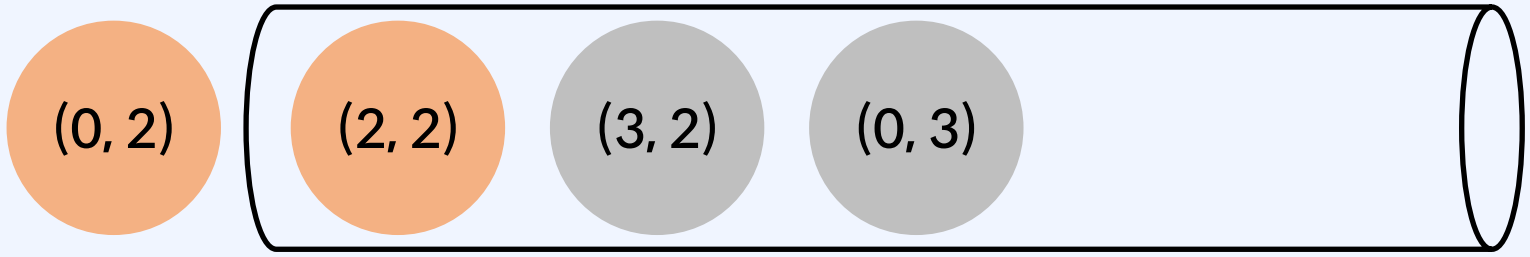


BOJ2178: 미로탐색

BFS와 큐

1 (0, 0)	0 (0, 1)	1 (0, 2)	1 (0, 3)
1 (1, 0)	1 (1, 1)	1 (1, 2)	0 (1, 3)
1 (2, 0)	0 (2, 1)	1 (2, 2)	0 (2, 3)
1 (3, 0)	1 (3, 1)	1 (3, 2)	1 (3, 3)

(0, 2) 에서
갈 수 있는 좌표를
큐에 넣기

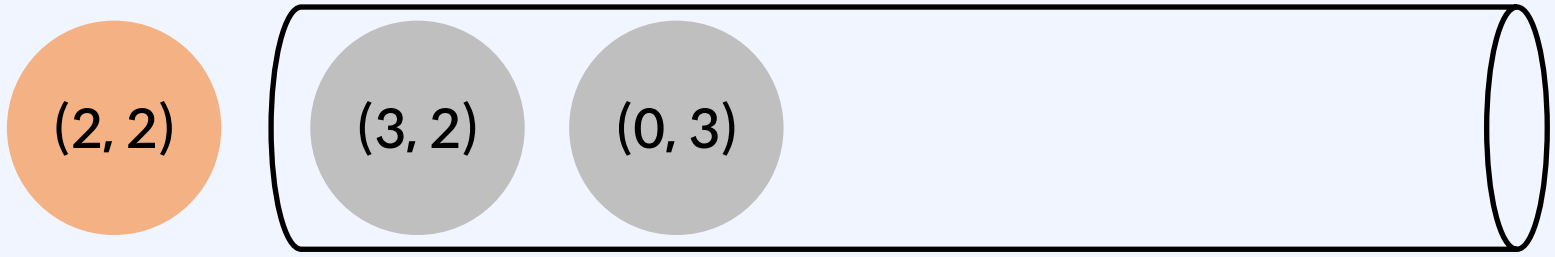


BOJ2178: 미로탐색

BFS와 큐

1 (0, 0)	0 (0, 1)	1 (0, 2)	1 (0, 3)
1 (1, 0)	1 (1, 1)	1 (1, 2)	0 (1, 3)
1 (2, 0)	0 (2, 1)	1 (2, 2)	0 (2, 3)
1 (3, 0)	1 (3, 1)	1 (3, 2)	1 (3, 3)

(2, 2) 에서
갈 수 있는 좌표를
큐에 넣기

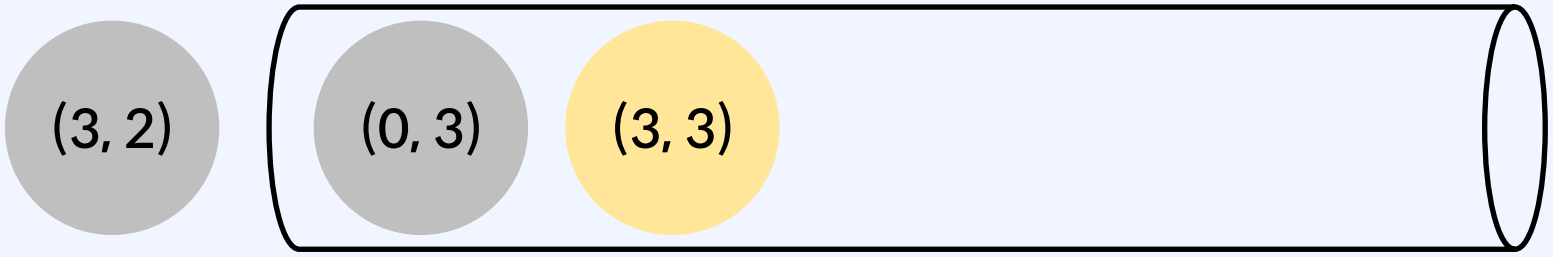


BOJ2178: 미로탐색

BFS와 큐

1 (0, 0)	0 (0, 1)	1 (0, 2)	1 (0, 3)
1 (1, 0)	1 (1, 1)	1 (1, 2)	0 (1, 3)
1 (2, 0)	0 (2, 1)	1 (2, 2)	0 (2, 3)
1 (3, 0)	1 (3, 1)	1 (3, 2)	1 (3, 3)

(3, 2) 에서
갈 수 있는 좌표를
큐에 넣기



BOJ2178: 미로탐색

BFS와 큐

1 (0, 0)	0 (0, 1)	1 (0, 2)	1 (0, 3)
1 (1, 0)	1 (1, 1)	1 (1, 2)	0 (1, 3)
1 (2, 0)	0 (2, 1)	1 (2, 2)	0 (2, 3)
1 (3, 0)	1 (3, 1)	1 (3, 2)	1 (3, 3)

(0, 3) 에서
갈 수 있는 좌표를
큐에 넣기

BOJ2178: 미로탐색

BFS와 큐

1 (0, 0)	0 (0, 1)	1 (0, 2)	1 (0, 3)
1 (1, 0)	1 (1, 1)	1 (1, 2)	0 (1, 3)
1 (2, 0)	0 (2, 1)	1 (2, 2)	0 (2, 3)
1 (3, 0)	1 (3, 1)	1 (3, 2)	1 (3, 3)

(3, 3) 에서
갈 수 있는 좌표를
큐에 넣기

(3, 3)

BOJ2178: 미로탐색

Queue 라이브러리

선언

```
Queue<Point> q = new LinkedList<>();
```

추가

```
q.add(0);
```

꺼내기

```
int now = q.poll();
```

BOJ2178: 미로탐색

지나야 하는 칸수는?

1 (0, 0)	X (0, 1)	5 (0, 2)	6 (0, 3)
2 (1, 0)	3 (1, 1)	4 (1, 2)	X (1, 3)
3 (2, 0)	X (2, 1)	5 (2, 2)	X (2, 3)
4 (3, 0)	5 (3, 1)	6 (3, 2)	7 (3, 3)

- 칸수를 기록하는 배열 선언
- 다음에 이동할 칸에
  현재 위치의 수치 + 1 을 넣는다

BOJ2178: 미로탐색

좌표 객체 구현

```
class Point {
    int r, c;

    Point(int r, int c) {
        this.r = r;
        this.c = c;
    }
}
```

- row, col 좌표를 기록하는 클래스
- Point p = new Point(0 ,0)

BOJ2178: 미로탐색

탐색 구현 -초기화

```
Queue<Point> q = new LinkedList<>();
q.add(new Point(0, 0));
visited[0][0] = 1;
```

- 큐 선언
- 출발 좌표 추가
- 출발 좌표 이동 칸 수 기록

BOJ2178: 미로탐색

탐색 구현

탐색 좌표 획득

4방향 탐색 시도
- 범위 내인지 체크
- 방문 여부 체크
- 이동가능 여부 체크
- 가능 시 큐에 좌표 추가
- 이동 횟수 기록

```
while (!q.isEmpty()) {
    Point now = q.poll();

    for (int i = 0; i < 4; i++) {
        int nr = now.r + dr[i];
        int nc = now.c + dc[i];
        if (nr < 0 || nr >= n || nc < 0 || nc >= m) continue;
        if (visited[nr][nc] == 0 && maze[nr][nc] == 1) {
            visited[nr][nc] = visited[now.r][now.c] + 1;
            q.add(new Point(nr, nc));
        }
    }
}
System.out.println(visited[n - 1][m - 1]);
```

Ch02. BFS – 너비우선 탐색

2. [1697] 숨바꼭질

BOJ1697: 숨바꼭질

문제 요약

- 1차원 배열 범위에서 탐색
  0 <= N, K <= 100,000
- 수빈, 동생의 좌표가 입력으로 주어짐
- (X + 1), (X – 1), (2 * X) 방법으로 이동 가능
- 동생을 찾는 가장 빠른 시간을 출력

BOJ1697: 숨바꼭질

문제 분석

- 탐색할 범위가 입력 범위를 벗어날 수 있다
  - (X + 1), (X – 1), (2 * X)

- 입력 범위 내에 속해 있는지 검사하는 기능이 필요하다

```
static boolean isRange(int x) {
    return x >= 0 && x <= 100000;
}
```

BOJ1697: 숨바꼭질

문제 분석

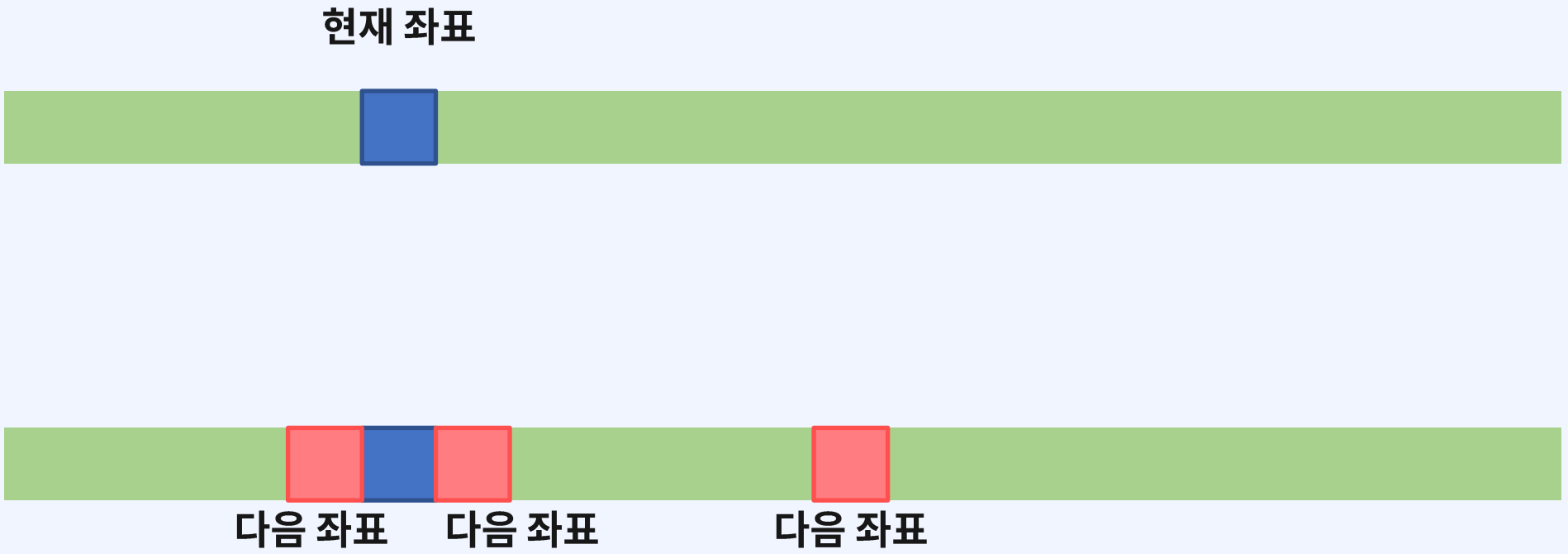
- 범위가 10만으로, 모든 범위를 다 돌아다녀도
  시간초과가 발생하지 않는다 (중복하지 않는다면)

- BFS를 통해 시간 순서대로 탐색하며
  정답을 찾으면 종료하면 된다

BOJ1697: 숨바꼭질

구현

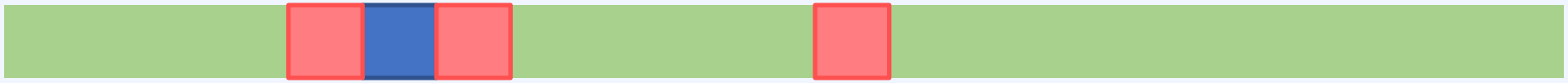
- Queue를 이용해 BFS탐색을 수행한다

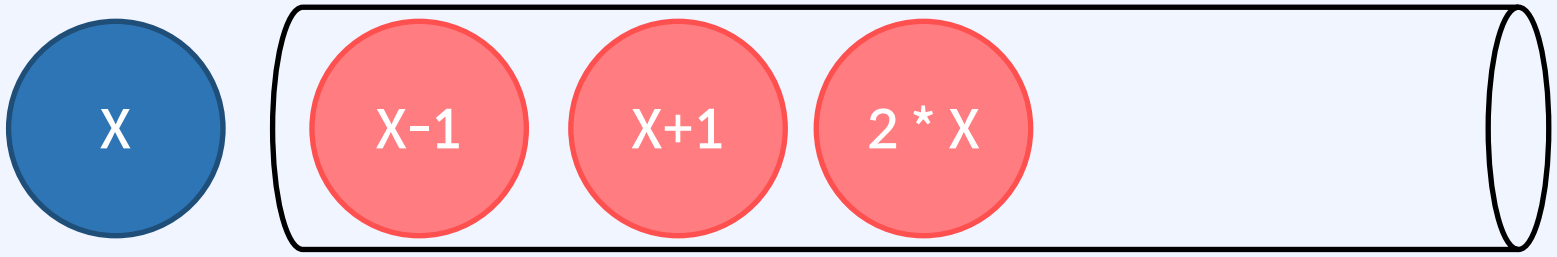


BOJ1697: 숨바꼭질

구현

- Queue를 이용해 BFS탐색을 수행한다





BOJ1697: 숨바꼭질

구현: Queue를 이용해 BFS탐색을 수행한다

```
while (!q.isEmpty()) {
    int now = q.poll();
    if (now == k) break;
    int[] next = {now - 1, now + 1, now * 2};
    for (int i = 0; i < 3; i++) {
        if(!isRange(next[i])) continue;
        if (visited[next[i]] == 0) {
            visited[next[i]] = visited[now] + 1;
            q.add(next[i]);
        }
    }
}
```

Ch02. BFS – 너비우선 탐색

3. [12851] 숨바꼭질2

BOJ12851: 숨바꼭질2

문제 요약

- 1차원 배열 범위에서 탐색
  0 <= N, K <= 100,000
- 수빈, 동생의 좌표가 입력으로 주어짐
- (X + 1), (X – 1), (2 * X) 방법으로 이동 가능
- 동생을 찾는 가장 빠른 시간과 방법의 수를 출력 <- NEW

BOJ12851: 숨바꼭질2

문제 분석

- 특정 좌표에 도착하는 모든 경우의 수를 구하는게 아니다!!

- 최단 시간에 도달하는 방법의 수를 구해야 한다
  1. 최단 시간을 BFS를 통해 먼저 구해야 한다
  2. 이후에 동일한 시간으로 좌표에 도달하면 방법의 수를 누적한다

BOJ12851: 숨바꼭질2

문제 분석

- 특정 좌표에 도착하는 모든 경우의 수를 구하는게 아니다!!

- 최단 시간에 도달하는 방법의 수를 구해야 한다
  1. 최단 시간을 BFS를 통해 먼저 구해야 한다
  2. 이후에 동일한 시간으로 좌표에 도달하면 방법의 수를 누적한다

BOJ12851: 숨바꼭질2

구현

```
    for (int i = 0; i < 3; i++) {
        if(!isRange(next[i])) continue;
        if (visited[next[i]] == 0) {
            visited[next[i]] = visited[now] + 1;
            q.add(next[i]);
        }
    }
}
```

- 이전 문제에서 구현한 코드

1. 현재 좌표까지 방문하는 방법의 수를 세야한다

BOJ12851: 숨바꼭질2

구현

```
    for (int i = 0; i < 3; i++) {
        if(!isRange(next[i])) continue;
        if (visited[next[i]] == 0) {
            visited[next[i]] = visited[now] + 1;
            count[next[i]] = count[now];
            q.add(next[i]);
        }
    }
}
```

- 이전 문제에서 구현한 코드

1. 현재 좌표까지 방문하는 방법의 수를 세야한다

BOJ12851: 숨바꼭질2

구현

```
    for (int i = 0; i < 3; i++) {
        if(!isRange(next[i])) continue;
        if (visited[next[i]] == 0) {
            visited[next[i]] = visited[now] + 1;
            count[next[i]] = count[now];
            q.add(next[i]);
        }
    }
}
```

- 이전 문제에서 구현한 코드

2. 최단 거리를 구했다면
   구한 카운트를 누적한다

BOJ12851: 숨바꼭질2

구현

```
    for (int i = 0; i < 3; i++) {
        if(!isRange(next[i])) continue;
        if (visited[next[i]] == 0) {
            visited[next[i]] = visited[now] + 1;
            count[next[i]] = count[now];
            q.add(next[i]);
        }
        else if (visited[next[i]] == visited[now] + 1) {
            count[next[i]] += count[now];
        }
    }
}
```

- 이전 문제에서 구현한 코드

2. 최단 거리를 구했다면
   구한 카운트를 누적한다

Ch02. BFS – 너비우선 탐색

4. [7562] 나이트의 이동

BOJ7562: 나이트의 이동

문제 요약

- 체스의 나이트 방식(2칸 직진 후 1칸 옆)으로 이동하여 시작 지점에서 목표 지점까지 필요한 이동 횟수 출력
- 보드는 항상 가로, 세로가 같은 길이
  - 4 <= l <= 300
- 시작 좌표와 목표 좌표가 주어지는데 이 때 시작점과 도착점이 같을 수 있다(0 출력)

BOJ7562: 나이트의 이동

문제 분석

- 직전에 구현했던 미로 탐색은? 4방향 탐색 (상, 하, 좌, 우)
- 이번 문제의 탐색 → 8방향 탐색
- 탐색을 어떻게 수행해야 할까?

BOJ7562: 나이트의 이동

상 하 좌 우

```
static int[] dr = {-1, 1, 0, 0};
static int[] dc = {0, 0, -1, 1};
```

체스 8방향

```
static int[] dr = {-2, -1, 1, 2, 2, 1, -1, -2};
static int[] dc = {1, 2, 2, 1, -1, -2, -2, -1};
```

체스 나이트의 규칙 맞게 이동할 좌표를 미리 매핑한다

BOJ7562: 나이트의 이동

구현 - 초기화

```
Point start = new Point(sc.nextInt(), sc.nextInt());
Point end = new Point(sc.nextInt(), sc.nextInt());
Queue<Point> q = new LinkedList<>();
q.add(start);
visited[start.r][start.c] = 1;
while (!q.isEmpty()) {
```

BFS 탐색에서 템플릿처럼 사용되는 코드

BOJ7562: 나이트의 이동

구현

- 8방향 탐색
- 다음 좌표 획득
- 범위 검증
- 탐색 대상에 다음 좌표 추가

```
for (int i = 0; i < 8; i++) {
    int nr = now.r + dr[i];
    int nc = now.c + dc[i];
    if (nr < 0 || nr >= n || nc < 0 || nc >= n) continue;
    if (visited[nr][nc] == 0) {
        visited[nr][nc] = visited[now.r][now.c] + 1;
        q.add(new Point(nr, nc));
    }
}
```

Ch02. BFS – 너비우선 탐색

5. [7576] 토마토

BOJ7576: 토마토

문제 요약

- 2 <= N, M <= 1000 크기의 배열이 입력으로 주어진다
- 배열의 값은 -1, 0, 1 중 하나이다
- 1인 칸들에서 탐색으로 각 칸들까지의 거리를 구한다
- -1인 칸으로는 이동할 수 없다
- 가장 먼 이동 거리를 출력한다

BOJ7576: 토마토

문제 요약

- 2 <= N, M <= 1000 크기의 배열이 입력으로 주어진다
- 배열의 값은 -1, 0, 1 중 하나이다
- 1인 칸들에서 탐색으로 각 칸들까지의 거리를 구한다
- -1인 칸으로는 이동할 수 없다
- 가장 먼 이동 거리를 출력한다

BOJ7576: 토마토

문제 분석

입력 데이터
6 4 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1

출력 데이터
8

BOJ7576: 토마토

문제 분석

입력 데이터
6 4 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1

출력 데이터
8

BOJ7576: 토마토

문제 분석

입력 데이터
6 4 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1

출력 데이터
8

BOJ7576: 토마토

문제 분석

입력 데이터
6 4 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1

출력 데이터
8

BOJ7576: 토마토

문제 분석

입력 데이터
6 4 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1

출력 데이터
8

BOJ7576: 토마토

문제 분석

입력 데이터
6 4 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1

출력 데이터
8

BOJ7576: 토마토

문제 분석

입력 데이터
6 4 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1

출력 데이터
8

BOJ7576: 토마토

문제 분석

입력 데이터
6 4 0 -1 0 0 0 0 -1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1

출력 데이터
8

BOJ7576: 토마토

문제 분석

입력 데이터
6 4 0 -1 0 0 0 0 -1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1

출력 데이터
8

토마토가 모두 익는데 걸린 날짜 → 탐색의 최대 Depth

BOJ7576: 토마토

문제 분석

입력 데이터
6 4 0 -1 0 0 0 0 -1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1

출력 데이터
-1

토마토가 모두 익지 못하는 상황은?

BOJ7576: 토마토

문제 분석

입력 데이터
6 4 0 -1 0 0 0 0 -1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1

출력 데이터
-1

토마토가 모두 익지 못하는 상황은?

BOJ7576: 토마토

문제 분석

입력 데이터
6 4 0 -1 0 0 0 0 -1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1

출력 데이터
-1

탐색이 끝나고, 방문하지 못한 영역이 있는지 검사

BOJ7576: 토마토

문제 분석

입력 데이터
6 4 0 -1 0 0 0 0 -1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1

출력 데이터
-1

탐색이 끝나고, 방문하지 못한 영역이 있는지 검사

BOJ7576: 토마토

구현

```
Queue<Point> q = new LinkedList<>();
for(int i = 0; i < n; i++) {
    for(int j = 0; j < m; j++) {
        board[i][j] = sc.nextInt();
        if(board[i][j] == 1) {
            q.add(new Point(i, j));
            visited[i][j] = 1;
        }
    }
}
```

반복문을 순회하며 입력을 받는다

입력 중에 익은 토마토가 보이면?

- 탐색의 대상에 될 수 있도록 큐에 넣는다

BOJ7576: 토마토

구현

```
while (!q.isEmpty()) {
    Point now = q.poll();
    for(int i = 0; i < 4; i++) {
        int nr = now.r + dr[i];
        int nc = now.c + dc[i];
        if(nr < 0 || nr >= n || nc < 0 || nc >= m) continue;
        if(visited[nr][nc] == 0 && board[nr][nc] == 0) {
            visited[nr][nc] = visited[now.r][now.c] + 1;
            q.add(new Point(nr, nc));
        }
    }
}
```

큐에 있는 좌표를 대상으로
4방향을 향해
다음 좌표 탐색

단 방문한 적이 없고,
토마토가 익어 있어야 한다

BOJ7576: 토마토

구현

```
int max = 0;
boolean yummy = true;
for(int i = 0; i < n; i++) {
    for(int j = 0; j < m; j++) {
        max = Math.max(max, visited[i][j]);
        if(visited[i][j] == 0 && board[i][j] == 0) {
            yummy = false;
            break;
        }
    }
}
if(yummy) System.out.println(max - 1);
else System.out.println(-1);
```

익지 않은 토마토가 발견되면
yummy 변수에 기록한다
탐색의 최대 깊이를 찾는다

위의 변수 값을 체크하여
탐색의 최대 깊이 or -1
출력한다

Ch02. BFS – 너비우선 탐색

6. [5427] 불

BOJ5427: 불

문제 요약

- 문자열로 지도가 들어온다
- 지도의 요소는 4가지가 존재하는데
- 빈 공간, 벽, 시작 위치, 불이다.

- 1 <= w, h <= 1000
- 시작지점에서 지도 밖으로 이동(x<0, w<x,y<0,h<y)에 걸리는 시간을 출력한다

BOJ5427: 불

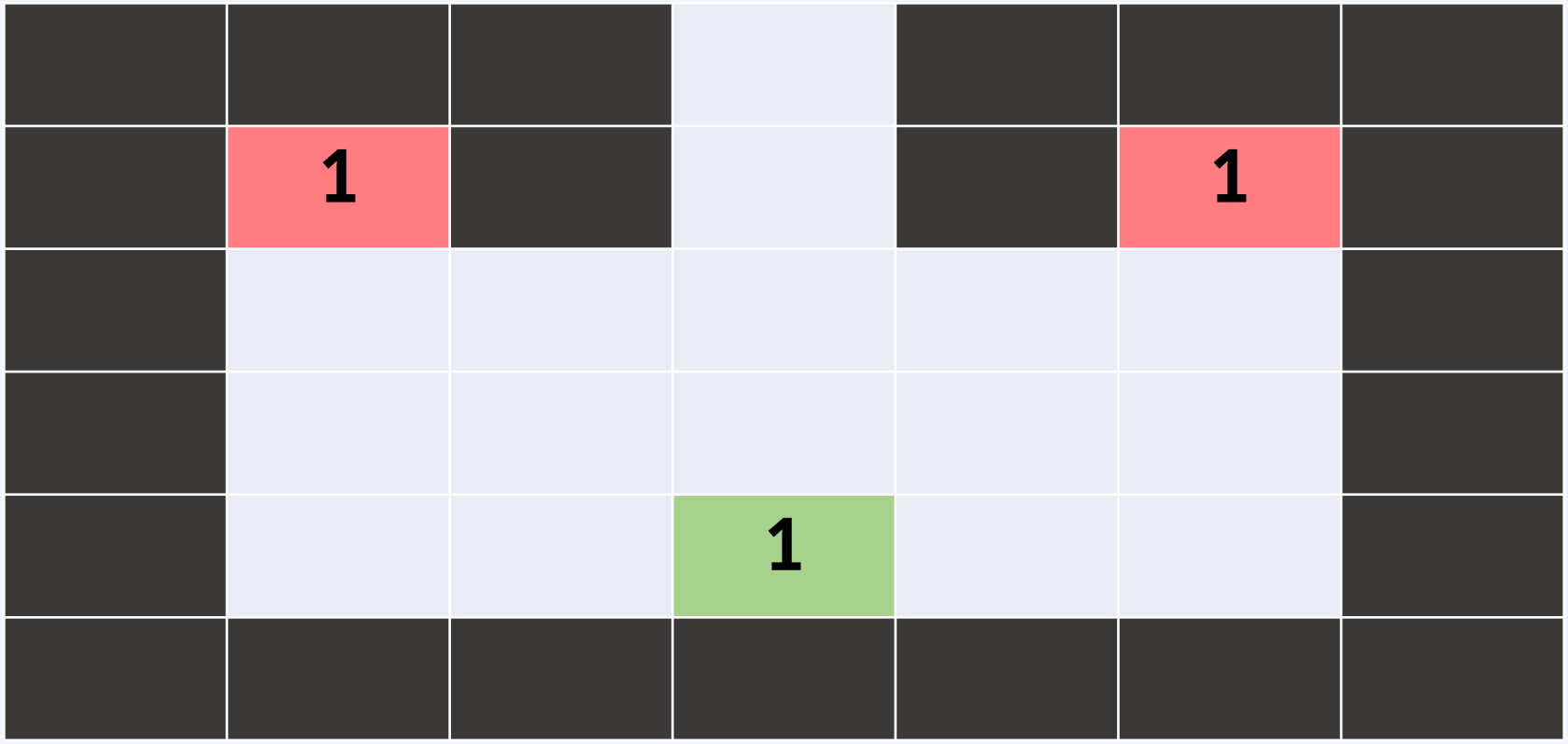
문제 분석

- 탈출경로: 탐색
- 불의 전파경로: 탐색

- 불의 경로를 먼저 탐색해야 한다
  - 불이 이동할 때: 탈출 경로에 대해서 고려할 필요 없다
  - 탈출 할 때: 현재 옮겨 붙은 불의 위치가 영향을 준다

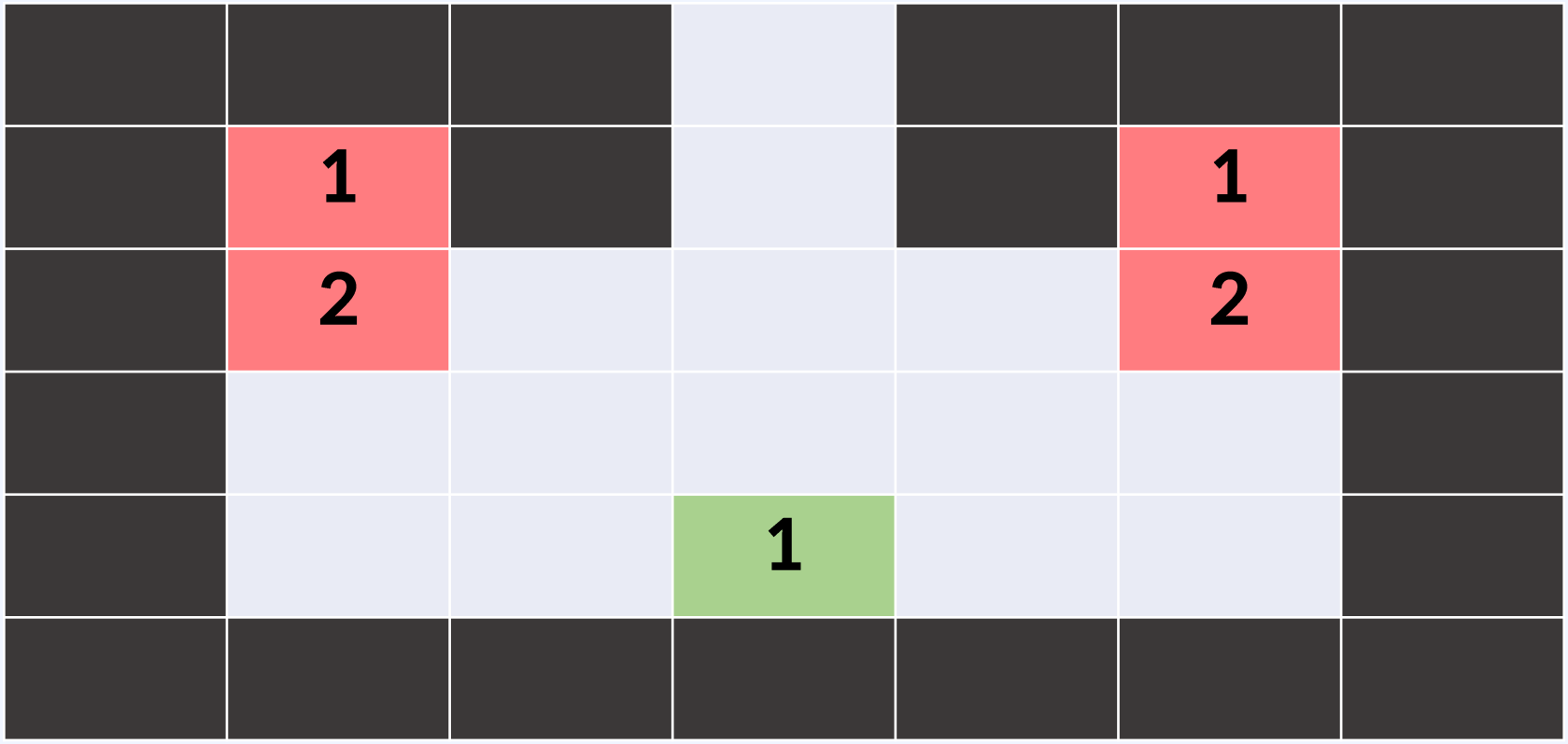
BOJ5427: 불

문제 분석



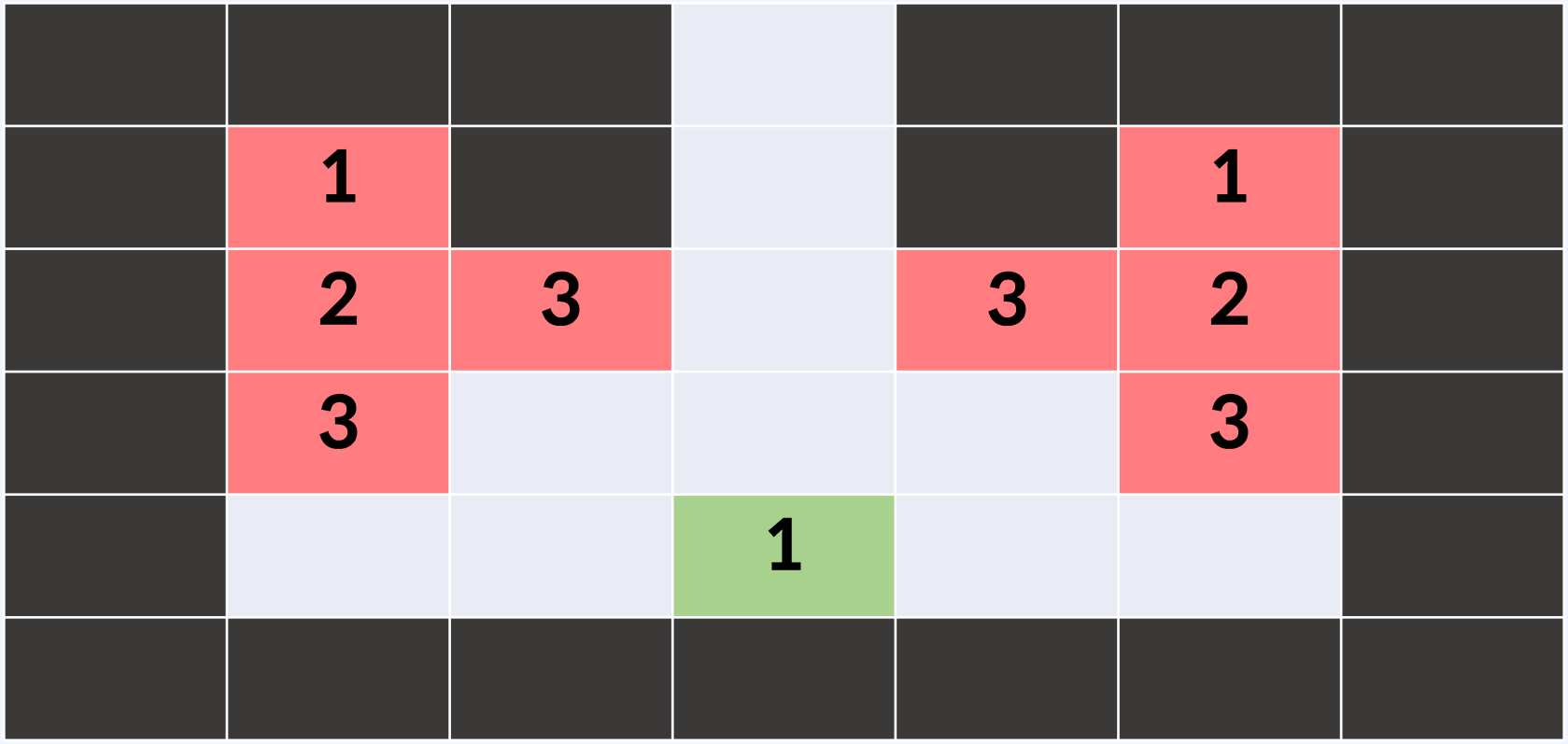
BOJ5427: 불

문제 분석



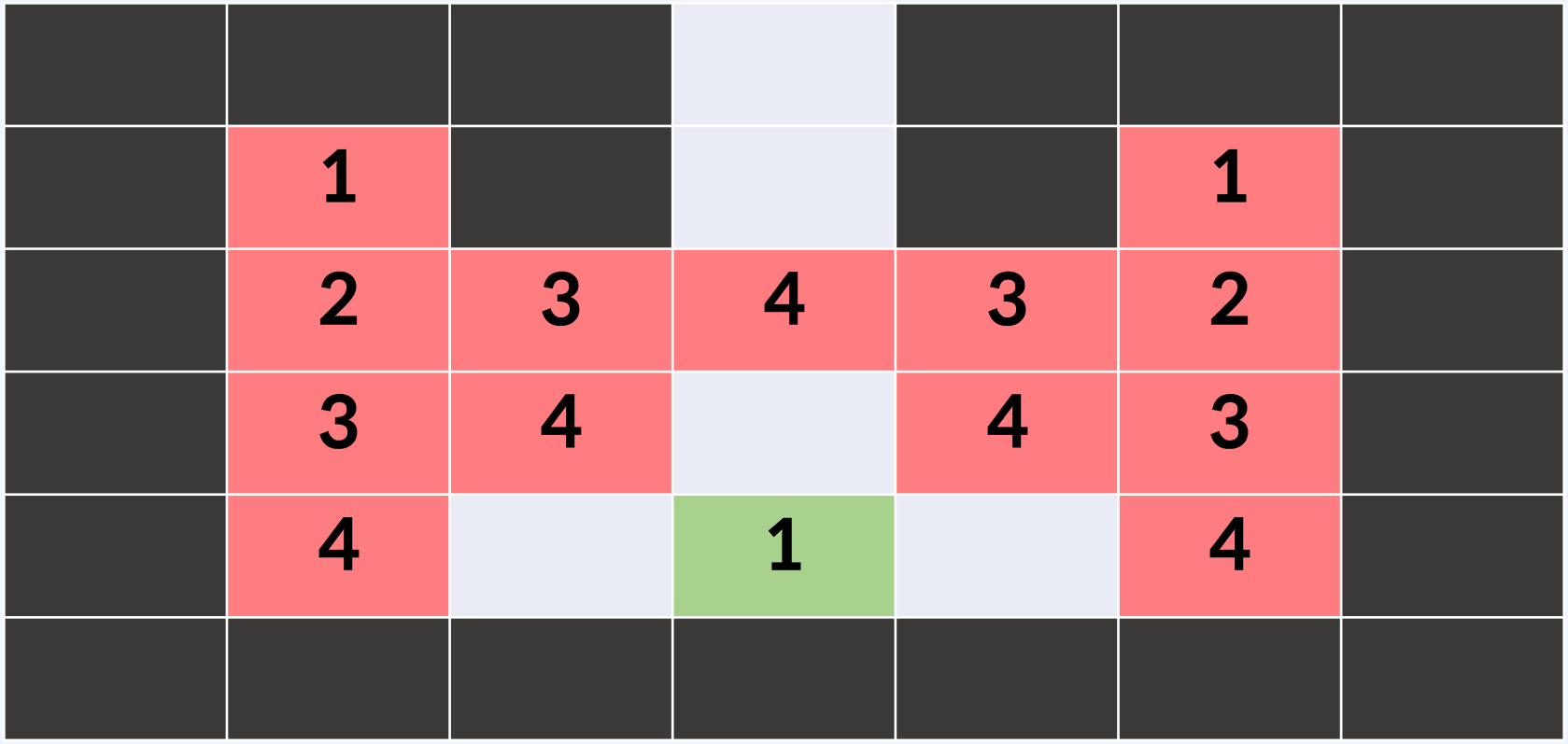
BOJ5427: 불

문제 분석



BOJ5427: 불

문제 분석



BOJ5427: 불

문제 분석

	1		5		1	
	2	3	4	3	2	
	3	4	5	4	3	
	4	5	1	5	4	

BOJ5427: 불

문제 분석

			6			
	1		5		1	
	2	3	4	3	2	
	3	4	5	4	3	
	4	5	1	5	4	

BOJ5427: 불

문제 분석

불보다 Depth가 낮은 경우만 이동 가능

			6			
	1		5		1	
	2	3	4	3	2	
	3	4	5	4	3	
	4	5	6/1	5	4	

BOJ5427: 불

문제 분석

불보다 Depth가 낮은 경우만 이동 가능

			6			
	1		5		1	
	2	3	4	3	2	
	3	4	5/2	4	3	
	4	5/2	6/1	5/2	4	

BOJ5427: 불

문제 분석

불보다 Depth가 낮은 경우만 이동 가능

			6			
	1		5		1	
	2	3	4/3	3	2	
	3	4/3	5/2	4/3	3	
	4/3	5/2	6/1	5/2	4/3	

BOJ5427: 불

문제 분석

불보다 Depth가 낮은 경우만 이동 가능

			6			
	1		5/4		1	
	2	3	4/3	3	2	
	3	4/3	5/2	4/3	3	
	4/3	5/2	6/1	5/2	4/3	

BOJ5427: 불

문제 분석

불보다 Depth가 낮은 경우만 이동 가능

			6/5			
	1		5/4		1	
	2	3	4/3	3	2	
	3	4/3	5/2	4/3	3	
	4/3	5/2	6/1	5/2	4/3	

BOJ5427: 불

문제 분석

- 여러 번의 테스트 케이스로 진행되는 문제
  - 초기화에 주의한다

- 지도에 대한 문자열을 입력 받을 때
  불과 상근이의 좌표를 기록

BOJ5427: 불

구현

fire[]: 불의 상태를 기록
visited[]: 재환이의 이동을 기록

```
visited = new int[m][n];
fire = new int[m][n];
```

```
for(int i = 0; i < m; i++) {
    String line = sc.next();
    for(int j = 0; j < n; j++) {
        char c = line.charAt(j);
        if(c == '#') {
            fire[i][j] = -1;
            visited[i][j] = -1;
        }
        else if(c == '@') {
            q.add(new Point(i, j));
            visited[i][j] = 1;
        }
        else if(c == '*') {
            fireQ.add(new Point(i, j));
            fire[i][j] = 1;
        }
    }
}
```

BOJ5427: 불

구현

4방향 좌표를 구해

fire[][] 에 탐색을 통해
언제 불이 전파되는지 기록

```
while (!fireQ.isEmpty()) {
    Point now = fireQ.poll();
    for(int i = 0; i < 4; i++) {
        int nr = now.r + dr[i];
        int nc = now.c + dc[i];
        if(isOutOfRange(nr, nc)) continue;
        if(fire[nr][nc] == 0) {
            fire[nr][nc] = fire[now.r][now.c] + 1;
            fireQ.add(new Point(nr, nc));
        }
    }
}
```

BOJ5427: 불

구현

탈출 가능한 좌표(외벽) 에
도착하면 정답 출력 후 탈출

4방향 좌표를 구해

이동 가능한 좌표이고,
불이 번지지 않거나,
아직 번지지 않은 경우에 탐색

```
boolean isEscaped = false;
while (!q.isEmpty()) {
    Point now = q.poll();
    if(isExit(now.r, now.c)) {
        System.out.println(visited[now.r][now.c]);
        isEscaped = true;
        break;
    }
    for(int i = 0; i < 4; i++) {
        int nr = now.r + dr[i];
        int nc = now.c + dc[i];
        if(isOutOfRange(nr, nc)) continue;
        if(visited[nr][nc] != 0) continue;
        if(fire[nr][nc] == 0 || fire[nr][nc] > visited[now.r][now.c] + 1) {
            visited[nr][nc] = visited[now.r][now.c] + 1;
            q.add(new Point(nr, nc));
        }
    }
}
```

BOJ5427: 불

구현

탈출 가능한 좌표(외벽) 에
도착하면 정답 출력 후 탈출

4방향 좌표를 구해

이동 가능한 좌표이고,
불이 번지지 않거나,
아직 번지지 않은 경우에 탐색

```
boolean isEscaped = false;
while (!q.isEmpty()) {
    Point now = q.poll();
    if(isExit(now.r, now.c)) {
        System.out.println(visited[now.r][now.c]);
        isEscaped = true;
        break;
    }
    for(int i = 0; i < 4; i++) {
        int nr = now.r + dr[i];
        int nc = now.c + dc[i];
        if(isOutOfRange(nr, nc)) continue;
        if(visited[nr][nc] != 0) continue;
        if(fire[nr][nc] == 0 || fire[nr][nc] > visited[now.r][now.c] + 1) {
            visited[nr][nc] = visited[now.r][now.c] + 1;
            q.add(new Point(nr, nc));
        }
    }
}
```

Ch02. BFS – 너비우선 탐색

7. [9019] DSLR

BOJ9019: DSLR

문제 요약

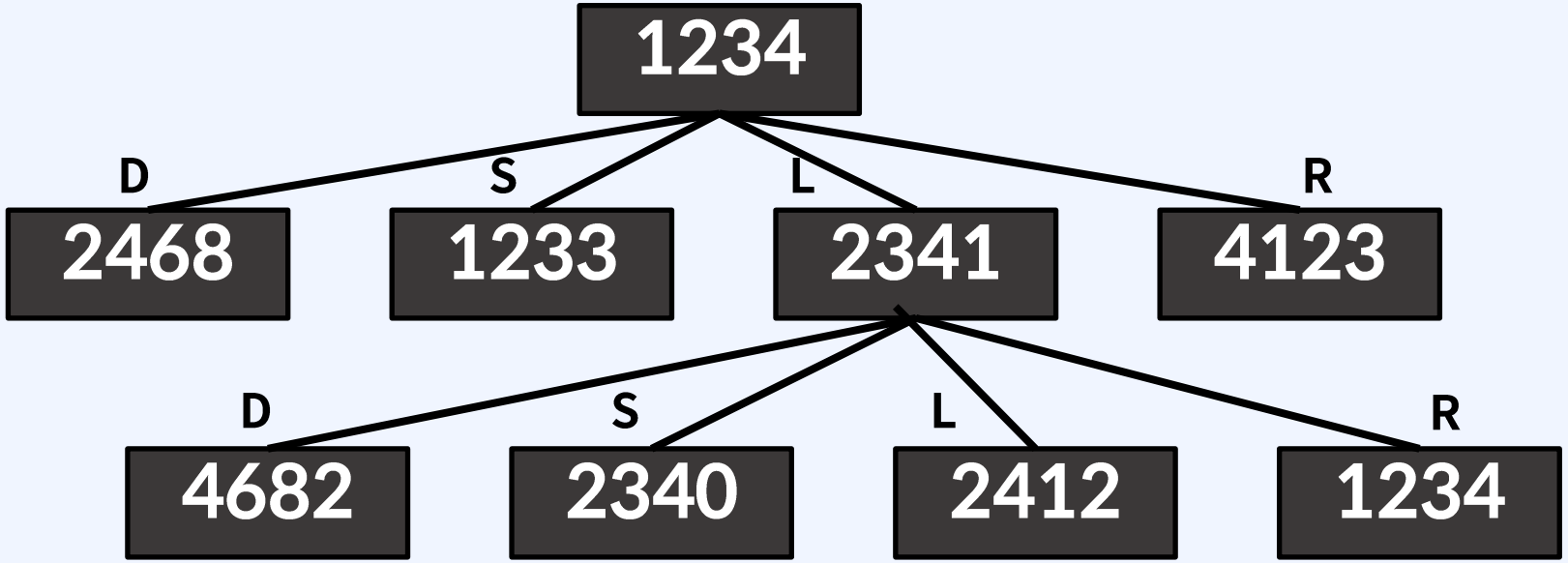
- 0000~9999 사이의 4자리 숫자 2개가 입력으로 주어진다
- (앞부분의 0은 생략)

- 숫자에 대해서 4종류의 연산을 할 수 있다
  - (*2, -1, <<1, >>1)
- 먼저 들어온 숫자(A)를 나중에 들어온 숫자(B)로 가장 적은 횟수로 바꿀 수 있는 명령어를 순서대로 출력한다.
- 방법이 여러 개일 경우에는 아무거나 출력한다

BOJ9019: DSLR

문제 분석

- 자식을 최대 4개씩 가지는 트리라고 생각하고 BFS를 수행한다
- 이때 깊이 뿐만 아니라 경로도 기록해야 한다



BOJ9019: DSLR

구현

레지스터 값과, 명령어를 기록할 클래스를 작성

D / S / L / R 에 해당하는 함수를 작성하면, main문에서 코드를 깔끔하게 작성할 수 있다

```
class Register {
    int num;
    StringBuilder cmd;
    Register(int num, StringBuilder cmd) {
        this.num = num;
        this.cmd = cmd;
    }
    public int calcD() {
        return (num * 2) % 10000;
    }
    public int calcS() {
        return (num - 1) < 0 ? 9999 : num - 1;
    }
    public int calcL() {
        return (num % 1000) * 10 + num / 1000;
    }
    public int calcR() {
        return (num % 10) * 1000 + num / 10;
    }
}
```

BOJ9019: DSLR

구현

초기화:
테스트 케이스 별로 check 배열을 새로 할당
입력으로 받은 수를 큐에 넣으며 탐색 시작

```
int in = sc.nextInt();
int out = sc.nextInt();
boolean[] check = new boolean[10000];
char[] cmd = {'D', 'S', 'L', 'R'};

Queue<Register> q = new LinkedList<>();
q.add(new Register(in, new StringBuilder()));
check[in] = true;
```

BOJ9019: DSLR

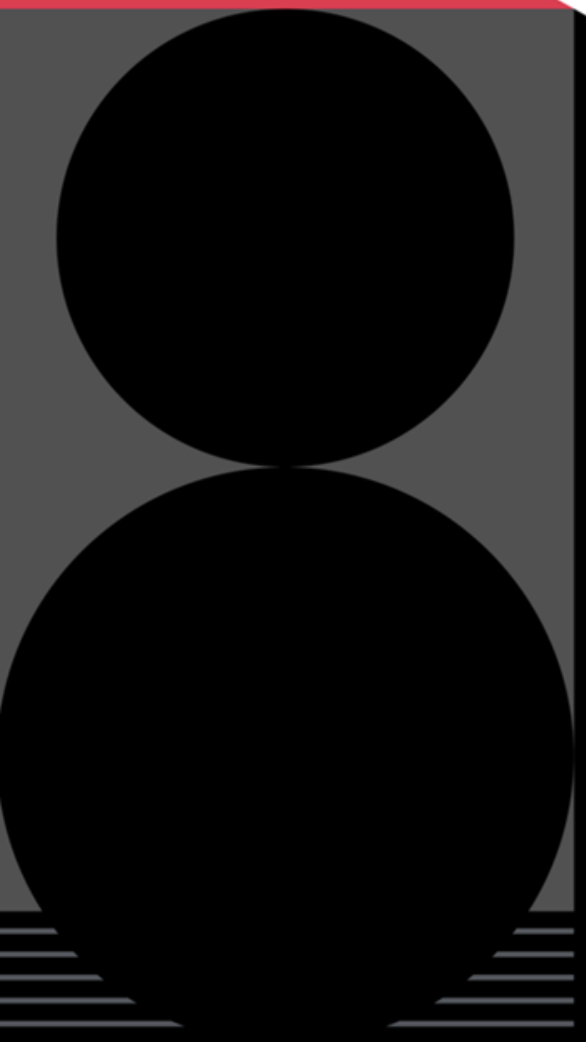
구현

수가 원하는 상태에 도달하면
출력 후 종료

D / S / L / R 을 수행한 결과를
배열에 담고

해당 수를 만든 적이 없다면 큐에
담아 다음 탐색을 수행

```
while (!q.isEmpty()) {
    Register now = q.poll();
    if (now.num == out) {
        System.out.println(now.cmd);
        break;
    }
    int[] next = {now.calcD(), now.calcS(),
now.calcL(), now.calcR()};
    for (int i = 0; i < 4; i++) {
        if (!check[next[i]]) {
            check[next[i]] = true;
            q.add(new Register(next[i], new
StringBuilder(now.cmd).append(cmd[i])));
        }
    }
}
```



Ch02. BFS – 너비우선 탐색

8. [2206] 벽 부수고 이동하기

BOJ2206: 벽 부수고 이동하기

문제 요약

- 1<=N,M<=1000 크기의 배열이 주어지고 BFS로 경로를 찾는 문제
  [Clip 1,2] 숨바꼭질과 유사함

- 차이점: 이동 불가능한 칸인 벽을 1개 무시할 수 있다

BOJ2206: 벽 부수고 이동하기

문제 분석

- 벽을 지나쳤는지를 구분하기 위한 축을 추가해
  3차원 배열로 BFS를 수행한다 [1000][1000][2]

- 시작 지점은 [1][1][0]이고 도착 지점은 [N][M][*]이다
- (벽을 부술 수 있는 거지 부서야 하는 게 아니다)

BOJ2206: 벽 부수고 이동하기

문제 분석

- 벽이 없어 이동할 수 있다면?
  - 4방향 탐색

- 벽에 가로 막혀있다면?
  - (r, c)의 좌표에 도착할 때 벽을 부순 적이 있는지 확인
  - 없다면 벽을 1회 부수고 [nr][nc][1] 로 탐색한다

BOJ2206: 벽 부수고 이동하기

구현

- 초기화
  - 공백이 없으므로 string으로 받고 파싱
- 상/하/좌/우 탐색 경로 선언
- 큐에 첫번째 좌표 넣고 탐색 준비

```
int[][] board = new int[n][m];
int[][][] visited = new int[n][m][2];
for(int i = 0; i < n; i++) {
    String line = sc.next();
    for(int j = 0; j < m; j++) {
        board[i][j] = line.charAt(j) - '0';
    }
}
int[] dr = {-1, 1, 0, 0};
int[] dc = {0, 0, -1 ,1};
Queue<Point> q = new LinkedList<>();
q.add(new Point(0, 0, 0));
visited[0][0][0] = 1;
```

BOJ2206: 벽 부수고 이동하기

구현

- 좌표를 다루는 클래스 선언
- 벽을 부수고 이동했는지 기록할 isBroken 멤버 추가 선언

```
class Point {
    int r, c;
    int isBroken;
    Point(int r, int c, int isBroken) {
        this.r = r;
        this.c = c;
        this.isBroken = isBroken;
    }
}
```

BOJ2206: 벽 부수고 이동하기

구현

- 초기화
  - 공백이 없으므로 string으로 받고 파싱

- 상/하/좌/우 탐색 경로 선언
- 큐에 첫번째 좌표 넣고 탐색 준비

```
while (!q.isEmpty()) {
    Point now = q.poll();
    if(now.r == n - 1 && now.c == m - 1) {

System.out.println(visited[now.r][now.c][now.isBroken]);
        return;
    }
    for(int i = 0; i < 4; i++) {
        int nr = now.r + dr[i];
        int nc = now.c + dc[i];
// ...
```

BOJ2206: 벽 부수고 이동하기

구현

- 다음 좌표 획득 및 유효성 검증
- 이동 가능하면 BFS 탐색
- 막혀 있더라도, 벽을 부순 적이 없다면 부수고 탐색

```
for(int i = 0; i < 4; i++) {
    int nr = now.r + dr[i];
    int nc = now.c + dc[i];
    if(nr < 0 || nr >= n || nc < 0 || nc >= m) continue;
    if(visited[nr][nc][now.isBroken] == 0) {
        if(board[nr][nc] == 0) {
            visited[nr][nc][now.isBroken] =
visited[now.r][now.c][now.isBroken] + 1;
            q.add(new Point(nr, nc, now.isBroken));
        }
        else if(board[nr][nc] == 1 && now.isBroken == 0) {
            visited[nr][nc][1] = visited[now.r][now.c][now.isBroken] + 1;
            q.add(new Point(nr, nc, 1));
        }
    }
}
```

Ch02. BFS – 너비우선 탐색

9. [14442] 벽 부수고 이동하기 2

BOJ14442: 벽 부수고 이동하기 2

문제 요약

- 1<=N, M<=1000 크기의 배열이 주어지고
  BFS로 경로를 찾는 문제

- 이동 불가능한 칸인 벽을 K개 무시할 수 있다
  - 이전 문제에서 부술 수 있는 벽의 1 → K개로 변했다

BOJ14442: 벽 부수고 이동하기 2

문제 분석

- 벽을 지나쳤는지를 구분하기 위한 축을 추가해
  3차원 배열로 BFS를 수행한다 [1000][1000][K]

- 시작 지점은 [1][1][0]이고 도착 지점은 [N][M][*]이다
- (벽을 부술 수 있는 거지 부서야 하는 게 아니다)

BOJ14442: 벽 부수고 이동하기 2

구현

- 초기화
  - 공백이 없으므로 string으로 받고 파싱

```
int[][] board = new int[n][m];
int[][][] visited = new int[n][m][k];
for(int i = 0; i < n; i++) {
    String line = sc.next();
    for(int j = 0; j < m; j++) {
        board[i][j] = line.charAt(j) - '0';
        for(int l = 0; l < k; l++) {
            visited[i][j][l] = 0;
        }
    }
}
```

- 상/하/좌/우 탐색 경로 선언
- 큐에 첫번째 좌표 넣고 탐색 준비

BOJ14442: 벽 부수고 이동하기 2

구현

- 좌표를 다루는 클래스 선언
- 벽을 부수고 이동했는지 기록할 isBroken 멤버 추가 선언

```
class Point {
    int r, c;
    int isBroken;
    Point(int r, int c, int isBroken) {
        this.r = r;
        this.c = c;
        this.isBroken = isBroken;
    }
}
```

BOJ14442: 벽 부수고 이동하기 2

구현

- BFS 탐색 시작
  - 만약 좌표가 끝에 도달했다면 출력 후 종료

- 상/하/좌/우 탐색 경로 선언
- 큐에 첫번째 좌표 넣고 탐색 준비

```
while (!q.isEmpty()) {
    Point now = q.poll();
    if(now.r == n - 1 && now.c == m - 1) {

System.out.println(visited[now.r][now.c][now.isBroken]);
        return;
    }
    for(int i = 0; i < 4; i++) {
        int nr = now.r + dr[i];
        int nc = now.c + dc[i];
// ...
```

BOJ14442: 벽 부수고 이동하기 2

구현

- 벽을 k번 이하로 부수고, 처음 방문하는지 체크한다
  - 직전에 부순 횟수 + 1 만큼 다음 좌표에 반영한다

```
else if(board[nr][nc] == 1 && now.isBroken < k
            && visited[nr][nc][now.isBroken + 1] == 0) {
   visited[nr][nc][now.isBroken + 1] = visited[now.r][now.c][now.isBroken] + 1;
   q.add(new Point(nr, nc, now.isBroken + 1));
}
```

Ch02. BFS – 너비우선 탐색

10. [1194] 달이 차오른다, 가자

BOJ1194: 달이 차오른다, 가자

문제 요약

- 경로를 찾는 BFS 문제
- 열쇠와 문을 이용한 특수 조건
  - 열쇠를 획득해야 문을 지나갈 수 있음
- [벽부수고이동하기2] 와 유사하지만, 열쇠가 소모되지 않음
  - 어떻게 구현??

BOJ1194: 달이 차오른다, 가자

문제 분석

- 열쇠의 종류는 총 6개
  - 각 좌표(r, c)마다 열쇠 보유 여부에 따라 이동 경로가 달라짐

- 모든 좌표마다 어떤 열쇠를 보유하고 있는지 상태를 알아야 한다
  - 열쇠 보유: ture / 미보유: false
  - 비트마스크로 표현할 수 있다

BOJ1194: 달이 차오른다, 가자

비트마스크

- [Part2. Ch6. Clip1] BOJ1987: 알파벳 에서 사용한 적이 있다
- 모든 열쇠를 보유하고 있는 경우

1	1	1	1	1	1

63

- a, c, f 열쇠만 보유하고 있는 경우

1	0	0	1	0	1

37

BOJ1194: 달이 차오른다, 가자

비트마스크

- 열쇠 표현
  - 1 << (next – ‘A’)
  - 1 << (next – ‘a’)

- 문에 맞는 열쇠가 있는지 확인
  - (now.key & (1 << (next - 'A'))) == 0 ← 열쇠가 없음

- 열쇠를 획득해서 다음 열쇠 꾸러미에 포함
  - int nextKey = now.key | (1 << (next - 'a'));

BOJ1194: 달이 차오른다, 가자

문제 분석

메모리는 안전할까?

- 열쇠와 문의 쌍은 총 6개로 1<<6 = 64이므로
- 64*50*50 = 160000 크기가 나온다
  - 160KB

BOJ1194: 달이 차오른다, 가자

구현

공백이 주어지지 않으므로
문자열로 받아서 파싱

출발 좌표를 찾으면
큐에 넣는다

```
maze = new char[n][m];
distance = new int[n][m][1 << 6];
Queue<Point> q = new LinkedList<>();
for (int r = 0; r < n; r++) {
    String line = sc.next();
    for (int c = 0; c < m; c++) {
        char temp = line.charAt(c);
        maze[r][c] = temp;
        if (temp == '0') {
            q.add(new Point(r, c, 0));
            distance[r][c][0] = 1;
        }
    }
}
```

BOJ1194: 달이 차오른다, 가자

```
static final int DOOR = 1;
static final int KEY = 2;
static final int EXIT = 3;
static final int WALL = 4;
static final int NONE = 5;
```


구현

코드 가독성을 위해
유틸리티성 함수 구현

- 탐색 범위체크
- 문자열이 무슨 의미인지 상수변수 반환

```
static boolean isOutOfRange(int r, int c) {
    return r < 0 || r >= n || c < 0 || c >= m;
}

static int getType(int c) {
    if (c == '#') return WALL;
    else if (c == '.' || c == '0') return NONE;
    else if (c >= 'a' && c <= 'f') return KEY;
    else if (c >= 'A' && c <= 'F') return DOOR;
    else if (c == '1') return EXIT;
    else return -1;
}
```

BOJ1194: 달이 차오른다, 가자

구현

탐색 중에 탈출 좌표를 찾으면
바로 출력하고 종료

그렇지 않다면 4방향으로
좌표 탐색 시도
(유효성 검증 포함)

```
while (!q.isEmpty()) {
    Point now = q.poll();
    if (maze[now.r][now.c] == '1') {
        System.out.println(
            distance[now.r][now.c][now.key] – 1
        );
        return;
    }
    for (int i = 0; i < 4; i++) {
        int nr = now.r + dr[i];
        int nc = now.c + dc[i];
        if (isOutOfRange(nr, nc)) continue;
        int next = maze[nr][nc];
```

BOJ1194: 달이 차오른다, 가자

구현

큐에는 입력으로 받은
출발 좌표가 들어있다

4방향으로
좌표 탐색 시도
(유효성 검증 포함)

```
while (!q.isEmpty()) {
    Point now = q.poll();

    for (int i = 0; i < 4; i++) {
        int nr = now.r + dr[i];
        int nc = now.c + dc[i];
        if (isOutOfRange(nr, nc)) continue;
        int next = maze[nr][nc];
```

BOJ1194: 달이 차오른다, 가자

구현

다음 미로 좌표에 있는 문자열을 읽어서

어떤 유형의 좌표인지 분기

(벽을 만나면 바로 continue)

```
switch (getType(next)) {
    case WALL -> {
        continue;
    }
    case NONE -> {

    }
    case EXIT -> {

    }
    case KEY -> {

    }
    case DOOR -> {

    }
}
```

BOJ1194: 달이 차오른다, 가자

구현

- 다음 좌표가 이동할 수 있는 좌표이고, 아직 이동한 적이 없다면 해당 좌표를 갱신한다
- 열쇠의 상태도 함께 처리한다

```
case NONE -> {
    if (distance[nr][nc][now.key] == 0) {
        distance[nr][nc][now.key] = distance[now.r][now.c][now.key] + 1;
        q.add(new Point(nr, nc, now.key));
    }
}
case EXIT -> {
    System.out.println(distance[now.r][now.c][now.key]);
    return;
}
```

- 탈출 좌표이면 정답을 출력하고 종료한다

BOJ1194: 달이 차오른다, 가자

구현

- 열쇠가 있는 좌표라면
  - 아직 해당 열쇠를 들고 방문한적이 없다면 탐색

- 문이 있는 좌표라면
  - 들고 있는 열쇠와 문을 대조
  - 아직 방문한적이 없다면 탐색 대상에 추가

```
case KEY -> {
    int nextKey = now.key | (1 << (next - 'a'));
    if (distance[nr][nc][nextKey] == 0) {
        distance[nr][nc][nextKey] =
                distance[now.r][now.c][now.key] + 1;
        q.add(new Point(nr, nc, nextKey));
    }
}
case DOOR -> {
    if ((now.key & (1 << (next - 'A'))) == 0) continue;
    if (distance[nr][nc][now.key] == 0) {
        distance[nr][nc][now.key] =
                distance[now.r][now.c][now.key] + 1;
        q.add(new Point(nr, nc, now.key));
    }
}
```