9월 2주차 BFS 알고리즘 스터디

백준 1697 숨바꼭질

코드.

```
import java.util.LinkedList;
import java.util.Queue;
import java.util.Scanner;

public class Baekjoon_1697 { // 숨바꼭질

    static int N;
    static int K;
    static int[] check;

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);
        N = sc.nextInt(); // 수빈이의 위치 입력
        K = sc.nextInt(); // 동생의 위치 입력
        check = new int[100001]; // 가장 빠른 경우를 구하는 문제이므로 방문체크 배열

        bfs(N); // bfs

        System.out.println(check[K] - 1); // 첫 시작지점이 1 이므로 도착지점은 -1을 해 주어야 한다.
    }

    private static void bfs(int number) {

        Queue<Integer> q = new LinkedList<Integer>();

        q.add(number); // 첫 시작지점을 큐에 넣고
        check[number] = 1; // 첫 시작지점을 방문체크 하기
```

```
        while (!q.isEmpty()) { // 큐가 빌 때까지 반복하기

            int temp = q.poll(); // 큐에 값을 temp에 저장, temp는 이동위치라고 생각

            if (temp == K) // 이동위치가 동생위치랑 같으면 반복문 탈출
                break;

            if (temp - 1 >= 0 && (check[temp - 1] == 0)) { // 이동가능범위 인지 확인 후에 방문체크 확인 조건
                q.add(temp - 1); // 이동할 위치를 큐에 입력
                check[temp - 1] = check[temp] + 1; // 이동할 위치에 현재위치 +1을 해주어 방문체크 AND 걸린 시간까지 확인
            }

            if (temp + 1 <= 100000 && (check[temp + 1] == 0)) { // 이동가능범위 인지 확인 후에 방문체크 확인 조건
                q.add(temp + 1); // 이동할 위치를 큐에 입력
                check[temp + 1] = check[temp] + 1; // 이동할 위치에 현재위치 +1을 해주어 방문체크 AND 걸린 시간까지 확인
            }

            if (temp * 2 <= 100000 && (check[temp * 2] == 0)) { // 이동가능범위 인지 확인 후에 방문체크 확인 조건
                q.add(temp * 2); // 이동할 위치를 큐에 입력
                check[temp * 2] = check[temp] + 1; // 이동할 위치에 현재위치 +1을 해주어 방문체크 AND 걸린 시간까지 확인
            }
        }
    }
}
```

실행결과.

```
5 17
4
```

백준 1303 전쟁 - 전투

코드.

```java
import java.io.BufferedReader;

public class Baekjoon_1303 {

    static char[][] map;
    static int[][] visit;
    static int time;
    static Queue<point> queue;
    static int N, M;
    static int[] di = { -1, 0, 1, 0 };
    static int[] dj = { 0, 1, 0, -1 };
    static char currentColor;

    static class point {
        int x, y;

        public point(int x, int y) {
            super();
            this.x = x;
            this.y = y;
        }

    }

    public static void main(String[] args) throws NumberFormatException, IOException {
        // TODO Auto-generated method stub

        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
        StringTokenizer st = new StringTokenizer(br.readLine());
        N = Integer.parseInt(st.nextToken()); // 가로 - 열의 갯수 (↕)
        M = Integer.parseInt(st.nextToken()); // 세로 - 행의 갯수 (↔)
```

```java
        map = new char[M][M];
        visit = new int[M][M];
        queue = new LinkedList<point>();

        for (int i = 0; i < M; i++) {
            map[i] = br.readLine().toCharArray();
        }
```

```java
        // W
        currentColor = 'W';
        int whiteVal = 0;
        for (int i = 0; i < M; i++) {
            for (int j = 0; j < N; j++) {
                if (visit[i][j] == 0 && map[i][j] == 'W') {
                    time = 1;
                    visit[i][j] = time;
                    queue.add(new point(i, j));
                    whiteVal += bfs();
                }
            }
        }
        // B
        currentColor = 'B';
        int blueVal = 0;
        for (int i = 0; i < M; i++) {
            for (int j = 0; j < N; j++) {
                if (visit[i][j] == 0 && map[i][j] == 'B') {
                    time = 1;
                    visit[i][j] = time;
                    queue.add(new point(i, j));
                    blueVal += bfs();
                }
            }
        }
        System.out.println(whiteVal+" "+blueVal);

    }
```

```
    private static int bfs() {
        // TODO Auto-generated method stub

        point now = null;
        while (!queue.isEmpty()) {
            now = queue.poll();

            for (int k = 0; k < 4; k++) {
                int mi = now.x + di[k];
                int mj = now.y + dj[k];

                if (mi < 0 || mi >= M || mj < 0 || mj >= N) // 맵을 벗어나면 패스
                    continue;
                if (currentColor != map[mi][mj] || visit[mi][mj] != 0) // 이미 방문했던 곳이거나, 현재 찾고자하는 색이 아닌경우도 패스
                    continue;
                time = time + 1;
                visit[mi][mj] = time;
                queue.add(new point(mi, mj));
            }
        }
        return visit[now.x][now.y] * visit[now.x][now.y];
    }

}
```

실행결과.

```
5 5
WBWWW
WWWWW
BBBBB
BBBWW
WWWWW
130 65
```