2022 여름학기 동국대학교 SW역량강화캠프

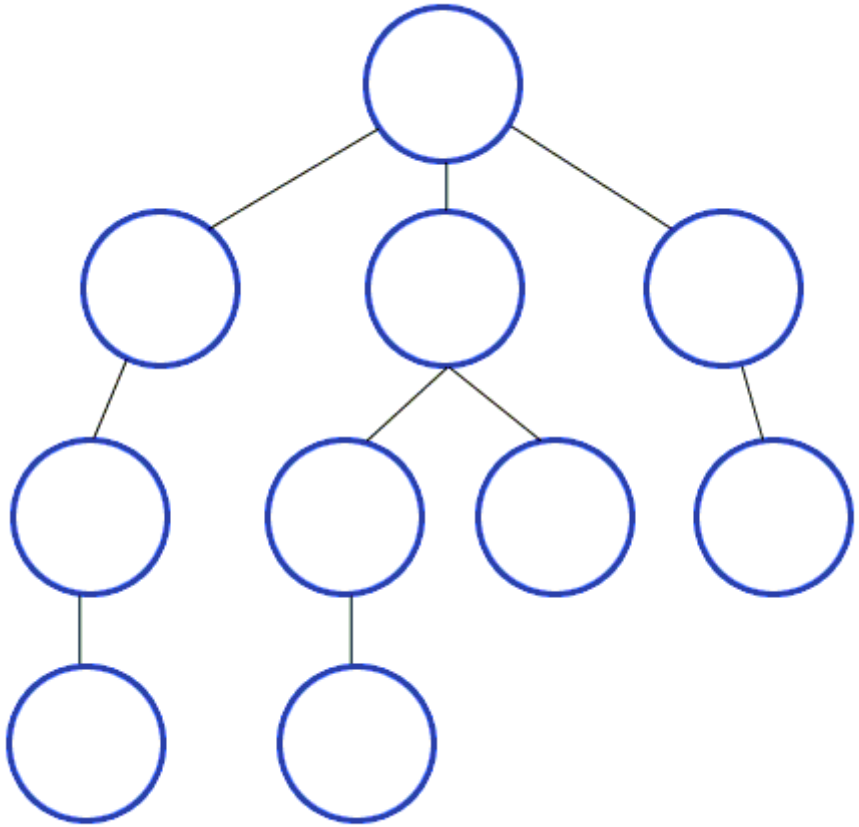
7일차. BFS 2

● BFS (너비우선탐색)

▶ DFS와는 다르게 재귀함수가 아닌 Queue를 이용해 그래프 탐색

▶ 답이 되는 경로가 여러가지여도 언제나 최단경로를 찾음을 보장함

▶ 최단, 최소해 찾을 때 사용



● BFS Flood Fill

▶ 좌표에서의 최단경로를 계산

▶ 만약 시작지점이 여러 개가 존재하고, 가장 가까운 시작지점의 거리를 찾으려 할 때에는 큐에 시작지점을 모두 넣고 BFS

0	@	-1	-1	0	0
-1	0	0	-1	0	-1
0	-1	0	0	0	@
0	0	0	-1	0	-1
-1	@	0	-1	0	-1
0	0	-1	-1	0	-1

1	@	-1	-1	0	0
-1	1	0	-1	0	-1
0	-1	0	0	1	@
0	1	0	-1	0	-1
-1	@	1	-1	0	-1
0	1	-1	-1	0	-1

1	@	-1	-1	0	0
-1	1	2	-1	2	-1
0	-1	0	2	1	@
2	1	2	-1	2	-1
-1	@	1	-1	0	-1
2	1	-1	-1	0	-1

● 미로 탐험(13)

문제

N×M크기의 배열로 표현되는 미로가 있다.

```
1  0  1  1  1  1
1  0  1  0  1  0
1  0  1  0  1  1
1  1  1  0  1  1
```

미로에서 1은 이동할 수 있는 칸을 나타내고, 0은 이동할 수 없는 칸을 나타낸다. 이러한 미로가 주어졌을 때, (1, 1)에서 출발하여 (N, M)의 위치로 이동할 때 지나야 하는 최소의 칸 수를 구하는 프로그램을 작성하시오. 한 칸에서 다른 칸으로 이동할 때, 서로 인접한 칸으로만 이동할 수 있다.

위의 예에서는 15칸을 지나야 (N, M)의 위치로 이동할 수 있다. 칸을 셀 때에는 시작 위치와 도착 위치도 포함한다.

문제 해설

▶ BFS를 이용하여 dist 배열을 채운다.

▶ dist[x][y] = (1,1)로부터의 최소 칸 수

▶ 초기값: dist[1][1] = 1, queue = [{1,1}]

▶ Queue에는 이차원 좌표를 넣어야 하므로 Integer[]이나 2개의 정수를 담을 수 있는 class/struct를 선언하여 사용

핵심 코드

```
Queue<Integer[]> queue = new LinkedList<>();
queue.offer(new Integer[] { 1, 1 });
dist[1][1] = 1;

while (!queue.isEmpty()) {
    Integer[] tmp = queue.poll();
    int x = tmp[0], y = tmp[1];
    for (int i = 0; i < 4; i++) {
        int newx = x + dx[i], newy = y + dy[i]; // (x,y)와 인접한 점 (newx, newy)
        if (newx < 1 || newx > N || newy < 1 || newy > M) // OOB 체크
            continue;
        if (arr[newx][newy] == 1 && dist[newx][newy] == 0) { // (newx, newy)가 queue에 들어간 적 없는 '1'인 좌표라면
            dist[newx][newy] = dist[x][y] + 1; // (newx, newy)까지의 거리는 (x,y)까지의 거리 + 1
            queue.offer(new Integer[] { newx, newy }); // queue에 (newx, newy) offer
        }
    }
}
```

● 불 (4699)

문제

윌리는 빈 공간과 벽으로 이루어진 건물에 갇혀있다. 건물의 일부에는 불이 났고, 윌리는 출구를 향해 뛰고 있다.

매 초마다, 불은 동서남북 방향으로 인접한 빈 공간으로 퍼져나간다. 벽에는 불이 붙지 않는다. 윌리는 동서남북 인접한 칸으로 이동할 수 있으며, 1초가 걸린다. 윌리는 벽을 통과할 수 없고, 불이 옮겨진 칸 또는 이제 불이 붙으려는 칸으로 이동할 수 없다. 윌리가 있는 칸에 불이 옮겨옴과 동시에 다른 칸으로 이동할 수 있다.

빌딩의 지도가 주어졌을 때, 얼마나 빨리 빌딩을 탈출할 수 있는지 구하는 프로그램을 작성하시오.

입력

첫째 줄에 테스트 케이스의 개수가 주어진다.

각 테스트 케이스의 첫째 줄에는 빌딩 지도의 너비와 높이 w와 h가 주어진다. ($1 \le w,h \le 1000$)

다음 h개 줄에는 w개의 문자, 빌딩의 지도가 주어진다.

'.': 빈 공간
'#': 벽
'@': 윌리의 시작 위치
'*': 불
각 지도에 @의 개수는 하나이다.

▶ BFS를 이용한 최단거리 풀이

▶ (x,y)에서 인접한 (newx, newy)로 이동할 수 있을 조건

1. newx, newy를 방문한 적이 없다
2. newx, newy가 빈칸이다.
3. newx, newy에 불이 붙는 시간보다 빨리 이동한다.

3번 조건을 처리하기 위해 모든 빈칸에 대해 불이 붙는 시간을 구해서 저장해놓자.

핵심 코드

```
for (int i = 1; i <= R; i++) {
    String str = br.readLine();
    for (int j = 1; j <= C; j++) {
        char tmp = str.charAt(j - 1);
        if (tmp == '.')
            arr[i][j] = 1;
        if (tmp == '@') { // 시작 위치는 sx, sy에 저장
            sx = i;
            sy = j;
            arr[i][j] = 1;
        }
        if (tmp == '*') { // 불의 위치는 전부 큐에 넣는다.
            queue.offer(new Integer[] { i, j });
            firedist[i][j] = 1;
            arr[i][j] = 1;
        }
    }
}
```

핵심 코드

```
while (!queue.isEmpty()) { // 빈칸에서 가장 가까운 불꽃지의 거리를 firedist에 저장, 즉 해당 빈칸은 firedist에 저장된 값만큼 시간이 지나면 불이 붙음
    Integer[] tmp = queue.poll();
    int x = tmp[0], y = tmp[1];
    for (int i = 0; i < 4; i++) {
        int newx = x + dx[i], newy = y + dy[i];
        if (newx < 1 || newx > R || newy < 1 || newy > C)
            continue;
        if (arr[newx][newy] == 1 && firedist[newx][newy] == 0) {
            firedist[newx][newy] = firedist[x][y] + 1;
            queue.offer(new Integer[] { newx, newy });
        }
    }
}
```

핵심 코드

```
queue.offer(new Integer[] { sx, sy });
dist[sx][sy] = 1;
int ans = Integer.MAX_VALUE;
while (!queue.isEmpty()) { // sx,sy부터 bfs
    Integer[] tmp = queue.poll();
    int x = tmp[0], y = tmp[1];
    for (int i = 0; i < 4; i++) {
        int newx = x + dx[i], newy = y + dy[i];
        if (newx < 1 || newx > R || newy < 1 || newy > C) { // (newx, newy)가 외부지점이라면 탈출할 수 있다.
            ans = Math.min(ans, dist[x][y] + 1);
            continue;
        }
        if (arr[newx][newy] == 1 && dist[newx][newy] == 0 // 방문한 적 없는 빈칸이고
                && (firedist[newx][newy] == 0 || firedist[newx][newy] > dist[x][y] + 1)) { // 불이 아직 옮겨붙지 않았다면
            dist[newx][newy] = dist[x][y] + 1;
            queue.offer(new Integer[] { newx, newy });
        }
    }
}
```

● 벽 부수고 이동하기(4702)

문제

N×M의 행렬로 표현되는 맵이 있다. 맵에서 0은 이동할 수 있는 곳을 나타내고, 1은 이동할 수 없는 벽이 있는 곳을 나타낸다. 당신은 (1, 1)에서 (N, M)의 위치까지 이동하려 하는데, 이때 최단 경로로 이동하려 한다. 최단경로는 맵에서 가장 적은 개수의 칸을 지나는 경로를 말하는데, 이때 시작하는 칸과 끝나는 칸도 포함해서 센다.

만약에 이동하는 도중에 한 개의 벽을 부수고 이동하는 것이 좀 더 경로가 짧아진다면, 벽을 한 개 까지 부수고 이동하여도 된다.

한 칸에서 이동할 수 있는 칸은 상하좌우로 인접한 칸이다.

맵이 주어졌을 때, 최단 경로를 구해 내는 프로그램을 작성하시오.

입력

첫째 줄에 N(1 ≤ N ≤ 1,000), M(1 ≤ M ≤ 1,000)이 주어진다. 다음 N개의 줄에 M개의 숫자로 맵이 주어진다. (1, 1)과 (N, M)은 항상 0이라고 가정하자.

출력

첫째 줄에 최단 거리를 출력한다. 불가능할 때는 -1을 출력한다.

▶ BFS를 이용한 최단경로 풀이

▶ (x,y) 좌표에 있는 벽을 부순다면 경로는 (1, 1) → (x, y) → (N, M)

▶ 벽을 하나만 부술 수 있으므로 (1, 1) → (x, y) 와 (x, y) → (N, M) 는 벽을 부수지 않고 가야 한다.

▶ 모든 벽에 대해서 (1,1)로부터의 거리와 (N,M)으로부터의 거리의 합이 최소가 되는 벽을 찾는다.

▶ BFS 2번

핵심 코드

```
Queue<Integer[]> queue = new LinkedList<>();
queue.offer(new Integer[] {1, 1}); dist1[1][1] = 1;

while(!queue.isEmpty()) {
    Integer[] tmp = queue.poll();
    int x = tmp[0], y = tmp[1];
    for(int i=0; i<4; i++) {
        int newx = x + dx[i], newy = y + dy[i];
        if(newx < 1 || newx > N || newy < 1 || newy > M) continue;
        if(dist1[newx][newy] == 0) {
            dist1[newx][newy] = dist1[x][y] + 1;
            if(arr[newx][newy] == 0) queue.offer(new Integer[]{newx, newy});
        }
    }
}
```

```
queue = new LinkedList<>();
queue.offer(new Integer[] {N, M}); dist2[N][M] = 1;

while(!queue.isEmpty()) {
    Integer[] tmp = queue.poll();
    int x = tmp[0], y = tmp[1];
    for(int i=0; i<4; i++) {
        int newx = x + dx[i], newy = y + dy[i];
        if(newx < 1 || newx > N || newy < 1 || newy > M) continue;
        if(dist2[newx][newy] == 0) {
            dist2[newx][newy] = dist2[x][y] + 1;
            if(arr[newx][newy] == 0) queue.offer(new Integer[]{newx, newy});
        }
    }
}
```

핵심 코드

```
int ans = Integer.MAX_VALUE;
for (int i = 1; i <= N; i++) {
    for (int j = 1; j <= M; j++) {
        if(dist1[i][j] == 0 || dist2[i][j] == 0) continue;
        int tmp = dist1[i][j] + dist2[i][j] - 1;
        ans = Math.min(ans, tmp);
    }
}
if(ans == Integer.MAX_VALUE) ans = -1;
System.out.println(ans);
```