줄기세포배양

줄기세포 배양 시뮬레이션 프로그램을 만들려고 한다.
줄기세포들을 배양 용기에 도포한 후 일정 시간 동안 배양을 시킨 후 줄기 세포의 개수가 몇 개가 되는지 계산하는 시뮬레이션 프로그램을 만들어야 한다.

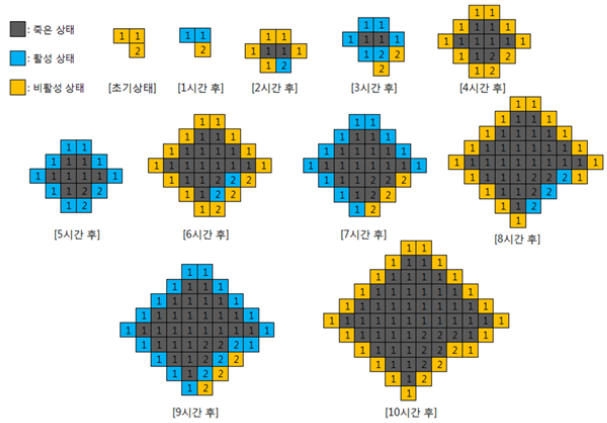
하나의 줄기 세포는 가로, 세로 크기가 1인 정사각형 형태로 존재하며 배양 용기는 격자 그리드로 구성되어 있으며 하나의 그리드 셀은 줄기 세포의 크기와 동일한 가로, 세로 크기가 1인 정사각형이다.

각 줄기 세포는 생명력이라는 수치를 가지고 있다.
초기 상태에서 줄기 세포들은 비활성 상태이며 생명력 수치가 X인 줄기 세포의 경우 X시간 동안 비활성 상태이고 X시간이 지나는 순간 활성 상태가 된다.
줄기 세포가 활성 상태가 되면 X시간 동안 살아있을 수 있으며 X시간이 지나면 세포는 죽게 된다.
세포가 죽더라도 소멸되는 것은 아니고 죽은 상태로 해당 그리드 셀을 차지하게 된다.

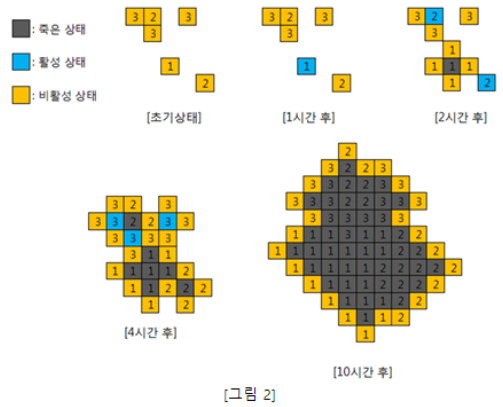
활성화된 줄기 세포는 **첫 1시간 동안 상, 하, 좌, 우 네 방향으로 동시에 번식**을 한다.
번식된 줄기 세포는 비활성 상태이다.
하나의 그리드 셀에는 하나의 줄기 세포만 존재할 수 있기 때문에 번식하는 방향에 이미 줄기 세포가 존재하는 경우 해당 방향으로 추가적으로 번식하지 않는다.
두 개 이상의 줄기 세포가 하나의 그리드 셀에 동시 번식하려고 하는 경우 생명력 수치가 높은 줄기 세포가 해당 그리드 셀을 혼자서 차지하게 된다.

줄기 세포의 크기에 비해 배양 용기의 크기가 매우 크기 때문에 시뮬레이션에서 배양 용기의 크기는 무한하다고 가정한다.

아래 [그림1]과 [그림2]는 줄기 세포가 번식하는 예를 나타낸다.



[그림 1]



[그림 2]

줄기 세포의 초기 상태 정보와 배양 시간 K시간이 주어질 때, **K시간 후 살아있는 줄기 세포(비활성 상태 + 활성 상태)의 총 개수를 구하는 프로그램**을 작성하라.

[제약 사항]

초기 상태에서 줄기 세포가 분포된 영역의 넓이는 세로 크기 N, 가로 크기 M이며 N, M은 각각 1 이상 50 이하의 정수이다. (1≤N≤50, 1≤M≤50)
배양 시간은 K시간으로 주어지며 K는 1 이상 300 이하의 정수이다. (1≤K≤300)
배양 용기의 크기는 무한하다. 따라서 줄기 세포가 배양 용기 가장자리에 닿아서 번식할 수 없는 경우는 없다.
줄기 세포의 생명력 X는 1 이상 10 이하의 정수이다. (1≤X≤10)

[입력]

입력의 가장 첫 줄에는 총 테스트 케이스의 개수 T가 주어진다.
그 다음 줄부터는 각 테스트 케이스가 주어지며
각 테스트 케이스의 첫째 줄에는 초기 상태에서줄기 세포가 분포된 세로 크기 N, 가로 크기 M, 배양 시간 K가 순서대로 주어진다.
다음 N 줄에는 각 줄마다 M개의 그리드 상태 정보가 주어진다.
1~10까지의 숫자는 해당 그리드 셀에 위치한 줄기 세포의 생명력을 의미하며,
0인 경우 줄기 세포가 존재하지 않는 그리드이다.

[출력]

테스트 케이스 T에 대한 결과는 "#T"을 찍고,
배양을 K시간 시킨 후 배양 용기에 있는 살아있는 줄기 세포(비활성 상태 + 활성 상태)의 개수를 출력한다. (T는 테스트케이스의 번호를 의미하며 1부터 시작한다.)

접근 방식

그림만 보았을 때는 전형적인 BFS 문제.

접근 방식에서 특이하게 고려해야 하는점.

1. 배양 용기의 크기는 무한하다.
   따라서 줄기 세포가 배양 용기 가장자리에 닿아서 번식할 수 없는 경우는 없다.

-> 무한하다고 하는 크기를 어떻게 제한 할 수 있는가?

세로 크기 N, 가로 크기 M이며 N, M은 각각 1 이상 50 이하의 정수이다. (1≤N≤50, 1≤M≤50)

K시간으로 주어지며 K는 1 이상 300 이하의 정수이다. (1≤K≤300)

크기와 시간이 제한적으로 주어지고,

하나의 세포가 가장 많이 번식하는 경우는

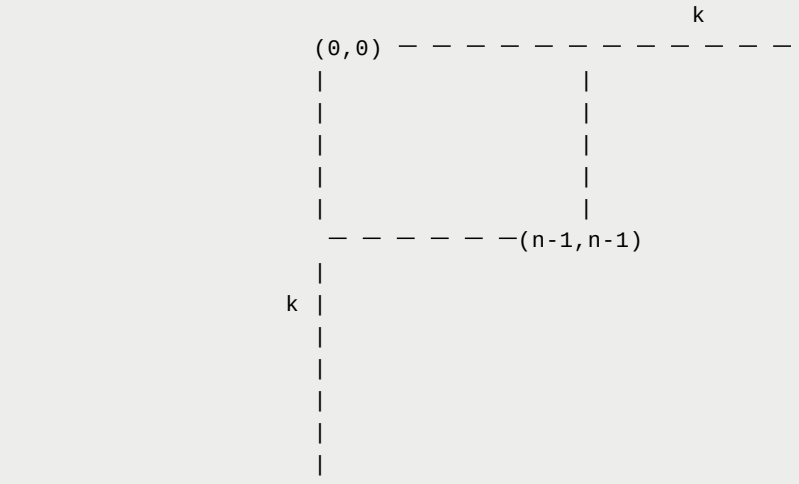
생명력이 1인 세포가 가장 멀리 번식 할 것이고,

1의 세포가 번식을 위해 (활성 1, 번식 1) 2초가 필요하므로,

한쪽으로 계속 퍼져나갔다고 가정 하였을 때, 최대로 K/2 만큼 퍼져나갈 수 있게 된다.

따라서,

(0 , 0) 에서 [우 , 하] 방향으로 K칸



을 확장하고, 값들을 K / 2 만큼 늘려주게 되면,
모든 범위의 세포를 담을 수 있는 용기 크기를 만들 수 있다.

2. 두 개 이상의 줄기 세포가 하나의 그리드 셀에
   동시 번식하려고 하는 경우 생명력 수치가 높은 줄기 세포가
   해당 그리드 셀을 혼자서 차지하게 된다.

동시에 처리하기 위해 BFS 방식을 사용하게되지만, 프로그램적으로 동시를 구현 할 수 없음.

=> 우선 순위를 이용하여, 높은 우선 순위를 갖는 셀이 먼저 차지하게 하므로써,
   불필요한 연산이 없도록 적용

3. 시간의 경과에 따라 셀의 상태가 변화함.

   현재 시간에 활동하는 셀의 집합과,

   시간이 경과 후 다음 탐색을 이어 갈 때에 활동 할 셀을 구분하여야 함.

풀이 방식

```
1. 우선순위 큐를 활용하여, 생명력이 높은 순으로 셀을 저장 할 수 있도록 한다.

2. 입력을 받으며, 값이 0 보다 큰 경우 ( 생명력이 있다 ) 좌표를 저장한다.

3. 저장 된 셀을 하나씩 뽑으면서 상태를 확인하고, 상태에 따라 배양한다.

  ⓐ 상태가 비활동 상태일 경우

     1. 대기 시간을 1 줄이고, 그 값이 0이라면 활동 상태로 바꿔주고 다시 저장한다.

     2. 줄인 값도 0 보다 크다면, 다음 시간에 판별 할 수 있도록 그대로 다시 저장한다.

  ⓑ 상태가 활동 상태 일 경우

     1. 4방 탐색을 하면서, 아직 방문하지 않은 경우
        그 값을 다음 시간에 사용 할 수 있도록 넣는다.

     => 먼저 탐색이 되었다면, 나보다 큰 생명력을 가진 셀이 점령하였거나,
                              초기의 값으로 있었던 셀인 경우이다.
       ( 생명력 기준으로 우선 탐색 하였기 때문에 )

    2. 셀의 생명력을 1 줄이고, 여전히 생명력이 남아있다면 다음 타임에 넣을 수 있도록 한다.

4. 시간이 끝나면, 여태까지 저장 된 값들을 다시 우선 순위 큐로 옮겨주고,

   1 ~ 3 의 과정을 반복한다.
```

코드

```
package Algo_Study_SWEA;

import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.util.*;

public class Solution_줄기세포배양 {
  static class Cell implements Comparable<Cell> {
    int r, c;
    int waitTime;
    int life;
    int status;

    public Cell(int r, int c, int life) {
      this.r = r;
      this.c = c;
      this.waitTime = this.life = life;
    }
    @Override
    public int compareTo(Cell o) {
```

```
      return Integer.compare(o.life, life);
    }

  }
  static int[] dx = new int[] { -1, 1, 0, 0 };
  static int[] dy = new int[] {  0, 0,-1, 1 };
  static int N, M, K; // 가로 , 세로 , 시간
  static int active = 1, dead = -1, unactive = 0; // 활성, 죽음, 비활성
  static boolean[][] visit;
  static PriorityQueue<Cell> serviveCell;
  static Queue<Cell> nextTimeCell;
  public static void main(String[] args) throws Exception {
    BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

    int TC = Integer.parseInt(br.readLine().trim());

    for(int tc = 1; tc <= TC; tc++) {

      StringTokenizer str = new StringTokenizer(br.readLine());

      N = Integer.parseInt(str.nextToken());
      M = Integer.parseInt(str.nextToken());
      K = Integer.parseInt(str.nextToken());

      visit = new boolean[N + K + 2][M + K + 2];
      serviveCell = new PriorityQueue<>();
      nextTimeCell = new LinkedList<>();
      for(int i = 0; i < N; i++) {
        str = new StringTokenizer(br.readLine());
        for(int j = 0; j < M; j++) {
          int life = Integer.parseInt(str.nextToken());
          if(life > 0) {
            Cell cell = new Cell(i + K / 2, j + K / 2, life);
            visit[cell.r][cell.c] = true;
            serviveCell.add(cell);
          }
        }
      }
      for(int time = 0; time < K; time++) {

        nextTimeCell.clear();

        while(!serviveCell.isEmpty()) {
          Cell cell = serviveCell.poll();

          if(cell.status == unactive) {
            cell.waitTime--;
            if(cell.waitTime == 0) {
              cell.status = active;
            }
            nextTimeCell.add(cell);
          }
          else if(cell.status == active) {
            for(int i = 0; i < 4; i++) {
              int ni = cell.r + dx[i];
              int nj = cell.c + dy[i];

              Cell nCell = new Cell(ni , nj , cell.life);
```

```
              // 방문 한적이 없다면 추가, 기존에 있다면 원래부터 있었거나 동일 시간에 더 큰 값의 세포
              if(!visit[ni][nj]) {
                visit[ni][nj] = true;
                nextTimeCell.add(nCell);
              }
            }
            cell.life--;
            if(cell.life > 0) {
              nextTimeCell.add(cell);
            }
          }
        }
        while(!nextTimeCell.isEmpty()) {
          serviveCell.add(nextTimeCell.poll());
        }
      }
      System.out.println("#" + tc + " " + serviveCell.size());
    }
  }
}
```