# 1981 배열에서 이동

### 배열에서 이동 🚜 🙀 🕬 🛱



| 시간 제한 | 메모리 제한 | 제출   | 정답   | 맞은 사람 | 정답 비율   |
|-------|--------|------|------|-------|---------|
| 1 초   | 256 MB | 4416 | 1068 | 697   | 23.133% |

#### 문제

n×n짜리의 배열이 하나 있다. 이 배열의 (1, 1)에서 (n, n)까지 이동하려고 한다. 이동할 때는 상, 하, 좌, 우의 네 인접한 칸으로만 이동할 수 있다.

이와 같이 이동하다 보면, 배열에서 몇 개의 수를 거쳐서 이동하게 된다. 이동하기 위해 거쳐 간 수들 중 최댓값과 최솟값의 차이가 가장 작아지는 경우를 구하는 프로그램을 작성하시오.

#### 입력

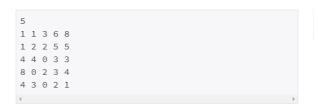
첫째 줄에 n(2≤n≤100)이 주어진다. 다음 n개의 줄에는 배열이 주어진다. 배열의 각 수는 0보다 크거나 같고, 200보다 작거나 같은 정수이다.

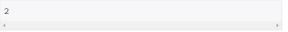
#### 출력

첫째 줄에 (최대 - 최소)가 가장 작아질 때의 그 값을 출력한다.

#### 예제 입력 1 복사

#### 예제 출력 1 복사





keyPoint : ( 1, 1) ~ ( n , n ) 까지 이동하기 위해 "거쳐간 수" (경로 상) 중

최대값과 최소값의 차이

# 고민 해본 방식

1. 완전 탐색을 통해 , 최대 최소 값을 갱신하며 진행하고, ( n , n ) 에 도착하였을 때 계산 후 답 도출

=> 경우의 수가 많고 연산도 많아 시간초과

### 문제 해결 포인트

```
어떻게 하면 탐색하는 경로를 제한적으로 만들 수 있을까?
=> 파라메트릭서치 : 기본적으로 이분탐색과 같지만 , 답을 기준으로 이분 탐색을 진행한다.
```

## 알고리즘 진행

```
    입력을 받을 때, 최대값 최소값을 갱신하면서 받는다.
    이 값을 통해 답으로 나올 수 있는 범위는 0 <= 답 <= (최대값 - 최소값) 이 된다.</li>
    양 범위의 중간값을 답이라고 가정하고, 탐색 범위를 제한하기 위해 최소값이 나올 수 없는 길을 방문처리 한다.
    그 후 탐색을 통해 (N, N) 에 도달 할 수 있는지를 판별한 후 파라메트릭서치를 진행한다.

            만약 도달 할 수 있다. => 더 작은 값으로 도달 할 수 있는지 시도
            만약 도달 할 수 없다. => 더 큰쪽에서 도달 할 수 있는지 시도

    과정을 통해 가장 적은 값으로 도출 된 해답이 정답.
```

## 코드

```
public class Main_1981_배열에서이동 {
 static int[] dx;
 static int[] dy;
 static int N;
 static int min, max;
 static int[][] map;
 static boolean[][] visit;
 static Queue<Point> queue;
 private static void clear() {
   // 방문처리 초기화
   for(int i = 0; i < N; i++) {
     for(int j = 0; j < N; j++) {
       visit[i][j] = false;
     }
   // 남은 q에 대해서 초기화
   queue.clear();
 }
```

```
private static boolean bfs(int mid) {
  for(int k = min; k \le max - mid; k++) {
    clear(); // 초기화
    for(int i = 0; i < N; i++) {
      for(int j = 0; j < N; j++) {
        if(map[i][j] < k) {
          visit[i][j] = true;
        else if(map[i][j] >= k && map[i][j] <= k + mid) {
          visit[i][j] = false;
        }
        else{
          visit[i][j] = true;
      }
    if(visit[0][0]) {
      continue;
    visit[0][0] = true;
    queue.add(new Point(0,0));
    while(!queue.isEmpty()) {
      Point tmp = queue.poll();
      int x = tmp.x;
      int y = tmp.y;
      if(x == (N - 1) \&\& y == (N - 1)) {
        return true;
      }
      for(int i = 0; i < 4; i++) {
        int nx = x + dx[i];
        int ny = y + dy[i];
        if(nx \ge 0 \&\& ny \ge 0 \&\& nx < N \&\& ny < N \&\& !visit[nx][ny]) {
          visit[nx][ny] = true;
          queue.add(new Point(nx, ny));
        }
      }
   }
  }
  return false;
public static void main(String[] args) throws Exception {
 BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
  N = Integer.parseInt(br.readLine());
 dx = new int[] { -1, 0, 1, 0 };
  dy = new int[] { 0, 1, 0, -1 };
  map = new int[N][N];
  visit = new boolean[N][N];
```

```
queue = new LinkedList<Point>();
   max = Integer.MIN_VALUE;
   min = Integer.MAX_VALUE;
   for(int i = 0; i < N; i++) {
     StringTokenizer str = new StringTokenizer(br.readLine());
     for(int j = 0; j < N; j++) {
       map[i][j] = Integer.parseInt(str.nextToken());
       max = Math.max(max, map[i][j]);
       min = Math.min(min, map[i][j]);
     }
   }
   int start = 0; // 시작값
   int end = max - min; // 입력받으면서 받는 차이의 최대값
   while(start <= end) {</pre>
     int mid = (start + end) / 2;
     // 중간값 수치의 차이로 갈 수 있는지 탐색
     if(bfs(mid)) {
      end = mid - 1;
     }
     else {
      start = mid + 1;
   System.out.println(end + 1);
}
```

## 어려웠던 점

```
1. 답으로 나올 수 있는 경우의 수를 가정하는 방식을 떠올리지 못했음.
```

- 2. BFS로 탐색 하기 전에 사전에 루트를 차단하는 아이디어가 생각나지 않았음.
- 3. 문제를 잘못 이해하여, 정확한 요구사항을 이해하지 못했음.