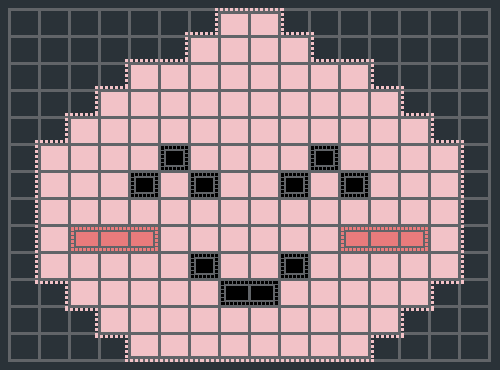
**카카오 프렌즈 컬러링북**

출판사의 편집자인 어피치는 네오에게 컬러링북에 들어갈 원화를 그려달라고 부탁하여 여러 장의 그림을 받았다. 여러 장의 그림을 난이도 순으로 컬러링북에 넣고 싶었던 어피치는 영역이 많으면 색칠하기가 까다로워 어려워진다는 사실을 발견하고 그림의 난이도를 영역의 수로 정의하였다. (영역이란 상하좌우로 연결된 같은 색상의 공간을 의미한다.)

그림에 몇 개의 영역이 있는지와 가장 큰 영역의 넓이는 얼마인지 계산하는 프로그램을 작성해보자.



위의 그림은 총 12개 영역으로 이루어져 있으며, 가장 넓은 영역은 어피치의 얼굴면으로 넓이는 120이다.

**입력 형식**

입력은 그림의 크기를 나타내는 m과 n, 그리고 그림을 나타내는 m × n 크기의 2차원 배열 picture로 주어진다. 제한조건은 아래와 같다.

* 1 <= m, n <= 100
* picture의 원소는 0 이상 2^31 - 1 이하의 임의의 값이다.
* picture의 원소 중 값이 0인 경우는 색칠하지 않는 영역을 뜻한다.

**출력 형식**

리턴 타입은 원소가 두 개인 정수 배열이다. 그림에 몇 개의 영역이 있는지와 가장 큰 영역은 몇 칸으로 이루어져 있는지를 리턴한다.

**예제 입출력**

| **m** | **n** | **picture** | **answer** |
| --- | --- | --- | --- |
| 6 | 4 | [[1, 1, 1, 0], [1, 2, 2, 0], [1, 0, 0, 1], [0, 0, 0, 1], [0, 0, 0, 3], [0, 0, 0, 3]] | [4, 5] |

**예제에 대한 설명**

예제로 주어진 그림은 총 4개의 영역으로 구성되어 있으며, 왼쪽 위의 영역과 오른쪽의 영역은 모두 1로 구성되어 있지만 상하좌우로 이어져있지 않으므로 다른 영역이다. 가장 넓은 영역은 왼쪽 위 1이 차지하는 영역으로 총 5칸이다.

import java.util.LinkedList;

import java.util.Queue;

class Solution {

static int dx[] = { -1, 1, 0, 0 };

static int dy[] = { 0, 0, -1, 1 };

static class Point {

public int x, y;

public Point(int a, int b) {

this.x = a;

this.y = b;

}

}

public int[] solution(int m, int n, int[][] picture) {

int numberOfArea = 0;

int maxSizeOfOneArea = 0;

boolean visited[][] = new boolean[m][n];

for (int i = 0; i < m; i++) {

for (int j = 0; j < n; j++) {

// 색칠 외 영역이면 다음 원소 탐색

if (picture[i][j] == 0)

continue;

// bfs 탐색 전에 이미 방문했다면 다음 원소 탐색

if (visited[i][j])

continue;

// bfs 출발

Queue<Point> queue = new LinkedList<>();

visited[i][j] = true; //시작지점 방문 체크

queue.add(new Point(i, j));

int count = 1;

while (!queue.isEmpty()) {

//현재 탐색 지점

Point cur = queue.poll();

// System.out.println("현재 탐색 지점 좌표 :"+cur.x+" "+cur.y+ ", 색값 : "+cur.value);

//다음 탐색 지점

for (int k = 0; k < 4; k++) {

int nx = cur.x + dx[k];

int ny = cur.y + dy[k];

//범위 판정

if(nx<0|| ny<0|| nx>=m || ny>=n) continue;

//색이 같고, 방문하지 않았다면

if(picture[cur.x][cur.y] == picture[nx][ny] && !visited[nx][ny]){

visited[nx][ny] = true;

count++;

queue.add(new Point (nx, ny));

}

}

}

if(count >maxSizeOfOneArea )

maxSizeOfOneArea = count ;

numberOfArea++;

}

}

int[] answer = new int[2];

answer[0] = numberOfArea;

answer[1] = maxSizeOfOneArea;

return answer;

}

}