## [**[백준] 2206번 - 벽 부수고 이동하기**](https://dheldh77.tistory.com/entry/%EB%B0%B1%EC%A4%80-2206%EB%B2%88-%EB%B2%BD-%EB%B6%80%EC%88%98%EA%B3%A0-%EC%9D%B4%EB%8F%99%ED%95%98%EA%B8%B0)

2020. 4. 9. 21:11

[BaekJoon](https://dheldh77.tistory.com/tag/BaekJoon), [BFS](https://dheldh77.tistory.com/tag/BFS), [알고리즘](https://dheldh77.tistory.com/tag/%EC%95%8C%EA%B3%A0%EB%A6%AC%EC%A6%98)

### 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.

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.

### 해결방법 >

BFS.

너비우선탐색을 하되, 벽을 부수고 이동하는 경우라는 특수한 조건을 고려해야한다.

인접한 노드를 탐색할 때,

방문하지 않았거나 방문했지만 벽을 부수고 탐색했던 경로에 포함된 경우 다시한번 탐색해준다.

이유는

0000

1110

0000

1111

0111

0000

과 같은 지도가 주어졌을 때,

일반적으로 벽을 한번 부수고 지나간다는 조건(방문하지 않았으면서 경로에 벽이 한번밖에 존재하지 않은 경우)으로

너비우선탐색을 할 경우s

1234

2005

3876

0987

0000

0000

으로 2행 0열 원소에서 이미 방문했던 정점이기때문에 탐색을 멈추게 된다.

 이러한 경우 때문에 방문했던 노드라도 벽을 부수고 탐색했던 노드라면

 벽을 부수지않고 탐색했던 경로에서 한번더 탐색해주어야한다.

### [JAVA]

package baekjoon;

import java.util.\*;

import java.io.\*;

public class BOJ\_2206 {

public static class Node{

int x;

int y;

Node(int x, int y){

this.x = x;

this.y = y;

}

}

public static void main(String[] args) throws IOException{

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

String[] arr = br.readLine().split(" ");

int n = Integer.parseInt(arr[0]);

int m = Integer.parseInt(arr[1]);

int[][] map = new int[n][m];

int[][] visit = new int[n][m];

int[][] len = new int[n][m];

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

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

int ans = -1;

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

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

String s = br.readLine();

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

map[i][j] = s.charAt(j) - '0';

}

}

q.add(new Node(0, 0));

visit[0][0] = 1;

len[0][0] = 1;

while(!q.isEmpty()) {

Node node = q.poll();

if(node.x == n - 1 && node.y == m - 1) {

ans = len[node.x][node.y];

break;

}

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

int nx = node.x + dx[i];

int ny = node.y + dy[i];

if(nx >= 0 && ny >= 0 && nx < n && ny < m) {

if(map[nx][ny] == 0 && (visit[nx][ny] == 0 || visit[nx][ny] > visit[node.x][node.y])) {

visit[nx][ny] = visit[node.x][node.y];

len[nx][ny] = len[node.x][node.y] + 1;

q.add(new Node(nx, ny));

}

else if(map[nx][ny] == 1 && visit[node.x][node.y] <= 1){

visit[nx][ny] = visit[node.x][node.y] + 1;

len[nx][ny] = len[node.x][node.y] + 1;

q.add(new Node(nx, ny));

}

}

}

}

System.out.println(ans);

}

}

## [**[백준] 1697번 - 숨바꼭질**](https://dheldh77.tistory.com/entry/%EB%B0%B1%EC%A4%80-1697%EB%B2%88-%EC%88%A8%EB%B0%94%EA%BC%AD%EC%A7%88)

2020. 4. 9. 21:00

[BaekJoon](https://dheldh77.tistory.com/tag/BaekJoon), [BFS](https://dheldh77.tistory.com/tag/BFS), [알고리즘](https://dheldh77.tistory.com/tag/%EC%95%8C%EA%B3%A0%EB%A6%AC%EC%A6%98)

### 문제 >

수빈이는 동생과 숨바꼭질을 하고 있다. 수빈이는 현재 점 N(0 ≤ N ≤ 100,000)에 있고, 동생은 점 K(0 ≤ K ≤ 100,000)에 있다. 수빈이는 걷거나 순간이동을 할 수 있다. 만약, 수빈이의 위치가 X일 때 걷는다면 1초 후에 X-1 또는 X+1로 이동하게 된다. 순간이동을 하는 경우에는 1초 후에 2\*X의 위치로 이동하게 된다.

수빈이와 동생의 위치가 주어졌을 때, 수빈이가 동생을 찾을 수 있는 가장 빠른 시간이 몇 초 후인지 구하는 프로그램을 작성하시오.

### 입력 >

첫 번째 줄에 수빈이가 있는 위치 N과 동생이 있는 위치 K가 주어진다. N과 K는 정수이다.

### 출력 >

수빈이가 동생을 찾는 가장 빠른 시간을 출력한다.

### 해결방법 >

BFS.

수빈이의 현재 위치에서 이동가능한 위치들을 너비우선탐색하면서

동생이 위치까지 최단시간으로 이동할 수 있는  경우를 찾는다.

1) N을 큐에 저장한다.

 2) 큐에서 값을 하나 뽑아낸다.

 3) 큐에서 뽑아낸 값 P와 인접한 위치(방문하지 않았으면서 P + 1, P - 1, P \* 2인 위치)들을 방문하면서K와 일치하는 지 확인하고 일치하지 않으면 다시 큐에 저장한다.

 4) K와 일치하는 값을 찾을 때까지 2) - 3) 과정을 반복한다.

### [JAVA]

package baekjoon;

import java.util.\*;

import java.io.\*;

public class BOJ\_1697 {

public static void main(String[] args) throws IOException{

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

String[] a = br.readLine().split(" ");

int n = Integer.parseInt(a[0]);

int k = Integer.parseInt(a[1]);

int[] visit = new int[100001];

int sec = 0;

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

q.add(n);

visit[n] = 1;

while(!q.isEmpty()) {

n = q.poll();

if(n == k) {

sec = visit[n] - 1;

break;

}

if(n + 1 >= 0 && n + 1 <= 100000 && visit[n + 1] == 0) {

visit[n + 1] = visit[n] + 1;

q.add(n + 1);

}

if(n - 1 >= 0 && n - 1 <= 100000 && visit[n - 1] == 0) {

visit[n - 1] = visit[n] + 1;

q.add(n - 1);

}

if(n \* 2 >= 0 && n \* 2 <= 100000 && visit[n \* 2] == 0) {

visit[n \* 2] = visit[n] + 1;

q.add(n \* 2);

}

}

System.out.println(sec);

}

}

## [**[백준] 8061번 - Bitmap**](https://dheldh77.tistory.com/entry/%EB%B0%B1%EC%A4%80-8061%EB%B2%88-Bitmap)

2020. 4. 9. 20:51

[BaekJoon](https://dheldh77.tistory.com/tag/BaekJoon), [BFS](https://dheldh77.tistory.com/tag/BFS), [알고리즘](https://dheldh77.tistory.com/tag/%EC%95%8C%EA%B3%A0%EB%A6%AC%EC%A6%98)

### 문제 >

here is given a rectangular bitmap of size n x m. Each pixel of the bitmap is either white or black, but at least one is white. The pixel in i-th line and j-th column is called the pixel (i,j). The distance between two pixels p1=(i1,j1) and p2=(i2,j2) is defined as:

d(p1,p2)=|i1-i2|+|j1-j2|.

Write a program which:

* reads the description of the bitmap from standard input,
* for each pixel, computes the distance to the nearest white pixel,
* writes the results to the standard output.

### 입력 >

First line of the standard input there is a pair of integer numbers n, m separated by a single space, 1 ≤ n ≤ 182, 1 ≤ m ≤ 182. In each of the following n lines of the input exactly one zero-one word of length m, the description of one line of the bitmap, is written. On the j-th position in the line (i+1), 1 ≤ i ≤ n, 1 ≤ j ≤ m, is 1 if, and only if the pixel (i,j) is white.

### 출력 >

In the i-th line of the standard output, 1 ≤ i ≤ n, there should be written m integers f(i,1),…,f(i,m) separated by single spaces, where f(i,j) is the distance from the pixel (i,j) to the nearest white pixel.

### 해결방법 >

BFS. 원소값이 1인 정점으로부터 원소값이 0인 정점까지 최단거리를 저장한다.

 1) 원소값이 1인 정점을 큐에 저장한다.

 2) 인접한 정점(원소값이 0이면서 방문하지 않는 정점)을 방문하면서 1부터 정점까지의 최단거리를 저장한다.

### [JAVA]

package baekjoon;

import java.util.\*;

import java.io.\*;

public class BOJ\_8061 {

public static class Node{

int x;

int y;

Node(int x, int y){

this.x = x;

this.y = y;

}

}

public static void main(String[] args) throws IOException{

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

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

int n = Integer.parseInt(tk.nextToken());

int m = Integer.parseInt(tk.nextToken());

int[][] map = new int[n][m];

int[][] visit = new int[n][m];

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

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

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

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

String s = br.readLine();

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

map[i][j] = s.charAt(j) - '0';

map[i][j] = map[i][j] == 1 ? 0 : 1;

if(map[i][j] == 0) {

visit[i][j] = 1;

q.add(new Node(i, j));

}

}

}

while(!q.isEmpty()) {

Node node = q.poll();

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

int nx = node.x + dx[i];

int ny = node.y + dy[i];

if(nx >= 0 && ny >= 0 && nx < n && ny < m && map[nx][ny] == 1 && visit[nx][ny] == 0) {

visit[nx][ny] = visit[node.x][node.y] + 1;

q.add(new Node(nx, ny));

}

}

}

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

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

System.out.print(visit[i][j] - 1 + " ");

}

System.out.println();

}

}

}

## [**[백준] 16637번 - 괄호 추가하기**](https://dheldh77.tistory.com/entry/%EB%B0%B1%EC%A4%80-16637%EB%B2%88-%EA%B4%84%ED%98%B8-%EC%B6%94%EA%B0%80%ED%95%98%EA%B8%B0)

2020. 4. 1. 07:43

[BaekJoon](https://dheldh77.tistory.com/tag/BaekJoon), [DFS](https://dheldh77.tistory.com/tag/DFS), [삼성A형](https://dheldh77.tistory.com/tag/%EC%82%BC%EC%84%B1A%ED%98%95), [알고리즘](https://dheldh77.tistory.com/tag/%EC%95%8C%EA%B3%A0%EB%A6%AC%EC%A6%98)

### 문제 >

길이가 N인 수식이 있다. 수식은 0보다 크거나 같고, 9보다 작거나 같은 정수와 연산자(+, -, ×)로 이루어져 있다. 연산자 우선순위는 모두 동일하기 때문에, 수식을 계산할 때는 왼쪽에서부터 순서대로 계산해야 한다. 예를 들어, 3+8×7-9×2의 결과는 136이다.

수식에 괄호를 추가하면, 괄호 안에 들어있는 식은 먼저 계산해야 한다. 단, 괄호 안에는 연산자가 하나만 들어 있어야 한다. 예를 들어, 3+8×7-9×2에 괄호를 3+(8×7)-(9×2)와 같이 추가했으면, 식의 결과는 41이 된다. 하지만, 중첩된 괄호는 사용할 수 없다. 즉, 3+((8×7)-9)×2, 3+((8×7)-(9×2))은 모두 괄호 안에 괄호가 있기 때문에, 올바른 식이 아니다.

수식이 주어졌을 때, 괄호를 적절히 추가해 만들 수 있는 식의 결과의 최댓값을 구하는 프로그램을 작성하시오. 추가하는 괄호 개수의 제한은 없으며, 추가하지 않아도 된다.

### 입력 >

첫째 줄에 수식의 길이 N(1 ≤ N ≤ 19)가 주어진다. 둘째 줄에는 수식이 주어진다. 수식에 포함된 정수는 모두 0보다 크거나 같고, 9보다 작거나 같다. 문자열은 정수로 시작하고, 연산자와 정수가 번갈아가면서 나온다. 연산자는 +, -, \* 중 하나이다. 여기서 \*는 곱하기 연산을 나타내는 × 연산이다. 항상 올바른 수식만 주어지기 때문에, N은 홀수이다.

### 출력 >

첫째 줄에 괄호를 적절히 추가해서 얻을 수 있는 결과의 최댓값을 출력한다. 정답은 231보다 작고, -231보다 크다.

### 해결방법 >

DFS를 사용했다.

 n개의 연산기호가 주어졌을 때, 연산기호의 부분집합을 구하는 방법을 생각할 수 있다.

 괄호안에는 한 개의 연산기호만 들어갈 수 있기 때문이다.

 여기서 부분집합이란 괄호로 묶어 먼저 처리되는 연산을 의미한다.

 또한 i번째 연산자가 부분집합에 포함되었을때,

 중첩괄호가 허용되지 않기때문에 i+1번째 연산자는 부분집합에 포함될 수 없다.

 아래와 같은 과정으로 진행된다.

 1) 입력된 문자열을 연산자와 숫자로 나눈어 LinkedList에 저장한다.

 2) 연산자가 저장된 LinkedList를 기준으로 DFS탐색을 한다.

 3) 조합이기 때문에 i번째 원소를 포함하는 경우와 포함하지 않는 경우로 DFS탐색을 한다.

 4) i번째 원소를 포함할 경우, i번째 연산자를 제거하고 i번째, i+1번째 숫자를 피연산자로 연산결과를 i번째에 숫자에 삽입한다.

 5) i+1번째 연산자로 3)번 과정을 다시 반복한다.

 i번째와 i+1번째 연산자를 부분집합으로 함께 묶을 수 없지만, i번째 연산자를 제거했을때 i+1번째 연산자는 원래 제거한 연산자의 i+2번째 이므로 가능한 부분집합이 된다.

 6) 부분집합을 구했을 때, LinkedList에 남아있는 숫자와 연산자를 순서대로 계산한다.

 7) 최솟값인 결과값을 구한다.

### [JAVA]

package baekjoon;

import java.io.\*;

import java.util.\*;

public class BOJ\_16637 {

static int len;

static int max = -2147483648;

public static void cal(LinkedList<Integer> n, LinkedList<Character> o, int idx) {

char op = o.remove(idx);

int num1 = n.remove(idx);

int num2 = n.remove(idx);

if(op == '+') {

n.add(idx, num1 + num2);

}

else if(op == '-') {

n.add(idx, num1 - num2);

}

else if(op == '\*') {

n.add(idx, num1 \* num2);

}

}

public static void dfs(int x, int flag, LinkedList<Integer> n, LinkedList<Character> o) {

LinkedList<Integer> nums = new LinkedList<>();

LinkedList<Character> op = new LinkedList<>();

nums.addAll(n);

op.addAll(o);

if(x > op.size() - 1 || x > len - 1) {

while(nums.size() > 1) {

cal(nums, op, 0);

}

max = Math.max(nums.get(0), max);

return;

}

if(flag == 1) {

cal(nums, op, x);

}

dfs(x + 1, 1, nums, op);

dfs(x + 1, 0, nums, op);

}

public static void main(String[] args) throws IOException {

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

int n = Integer.parseInt(br.readLine());

len = n / 2;

String s = br.readLine();

LinkedList<Integer> nums = new LinkedList<>();

LinkedList<Character> op = new LinkedList<>();

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

char c = s.charAt(i);

if(i % 2 == 0) nums.add(c - '0');

else op.add(c);

}

dfs(0, 0, nums, op);

dfs(0, 1, nums, op);

System.out.println(max);

}

}

## [**[백준] 17471번 - 게리멘더링**](https://dheldh77.tistory.com/entry/%EB%B0%B1%EC%A4%80-17471%EB%B2%88-%EA%B2%8C%EB%A6%AC%EB%A9%98%EB%8D%94%EB%A7%81)

2020. 3. 30. 08:47

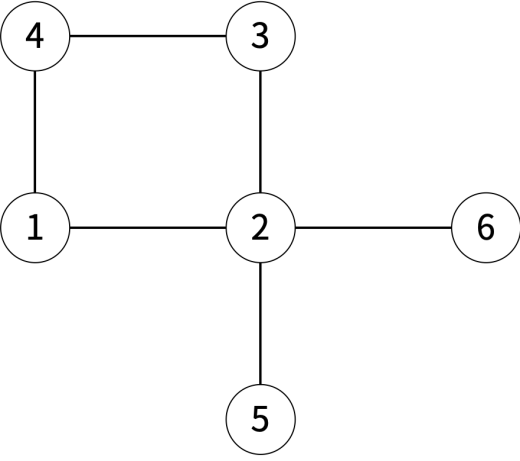
[17471](https://dheldh77.tistory.com/tag/17471), [BFS](https://dheldh77.tistory.com/tag/BFS), [DFS](https://dheldh77.tistory.com/tag/DFS), [삼성A형](https://dheldh77.tistory.com/tag/%EC%82%BC%EC%84%B1A%ED%98%95), [알고리즘](https://dheldh77.tistory.com/tag/%EC%95%8C%EA%B3%A0%EB%A6%AC%EC%A6%98)

### 문제 >

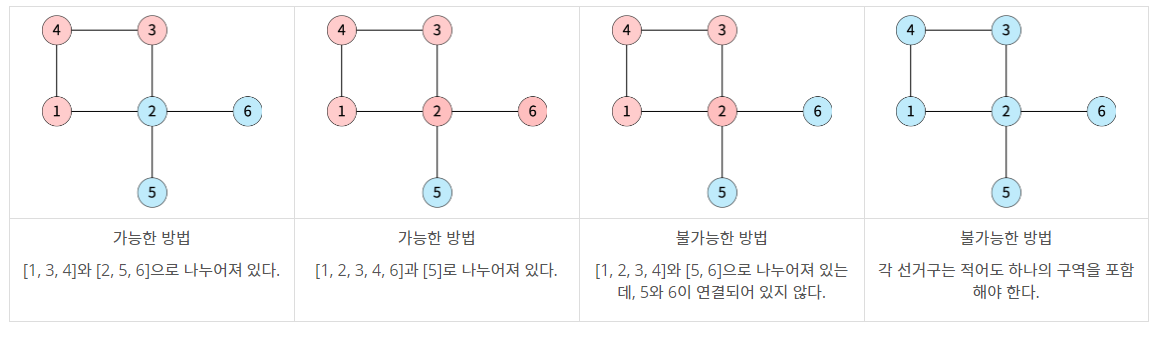
백준시의 시장 최백준은 지난 몇 년간 게리맨더링을 통해서 자신의 당에게 유리하게 선거구를 획정했다. 견제할 권력이 없어진 최백준은 권력을 매우 부당하게 행사했고, 심지어는 시의 이름도 백준시로 변경했다. 이번 선거에서는 최대한 공평하게 선거구를 획정하려고 한다.

백준시는 N개의 구역으로 나누어져 있고, 구역은 1번부터 N번까지 번호가 매겨져 있다. 구역을 두 개의 선거구로 나눠야 하고, 각 구역은 두 선거구 중 하나에 포함되어야 한다. 선거구는 구역을 적어도 하나 포함해야 하고, 한 선거구에 포함되어 있는 구역은 모두 연결되어 있어야 한다. 구역 A에서 인접한 구역을 통해서 구역 B로 갈 수 있을 때, 두 구역은 연결되어 있다고 한다. 중간에 통하는 인접한 구역은 0개 이상이어야 하고, 모두 같은 선거구에 포함된 구역이어야 한다.

아래 그림은 6개의 구역이 있는 것이고, 인접한 구역은 선으로 연결되어 있다.



아래는 백준시를 두 선거구로 나눈 4가지 방법이며, 가능한 방법과 불가능한 방법에 대한 예시이다.



공평하게 선거구를 나누기 위해 두 선거구에 포함된 인구의 차이를 최소로 하려고 한다. 백준시의 정보가 주어졌을 때, 인구 차이의 최솟값을 구해보자.

### 입력 >

첫째 줄에 배열 A의 크기 N, M, 회전 연산의 개수 K가 주어진다.

둘째 줄부터 N개의 줄에 배열 A에 들어있는 수 A[i][j]가 주어지고, 다음 K개의 줄에 회전 연산의 정보 r, c, s가 주어진다.

### 출력 >

배열 A의 값의 최솟값을 출력한다.

### 해결방법 >

DFS, BFS를 활용해서 해결할 수 있다.

선거구를 나누고 나눠진 선거구가 조건에 만족할 때, 두 선거구의 인구수의 차를 구한다.

선거구를 나누는 방식은 DFS를 사용했다.

 한 구역을 시작으로 인접한 구역을 방문한다. 이 때, 방문한 구역들이 한 선거구가 되고 방문하지 않는 구역들이 나머지 한 선거구가 된다.

 방문하지 않은 선거구를 BFS로 탐색해 모든 구역을 탐색할 수 있을 때, 문제에 주어진 조건을 만족함으로 이 때의 두 선거구의 인구수의 차를 구한다.

 이러한 방식으로 각 경우마다 인구수의 차를 구하고 최소값을 구한다.

### [JAVA]

import java.util.\*;

import java.io.\*;

public class Main {

static int[] people;

static int[][] adj;

static int n;

static int min = 999999;

static int total = 0;

static int[] visit;

public static boolean bfs() {

int[] v = new int[n];

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

int x = 0;

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

v[i] = visit[i];

if(v[i] == 0) x = i;

}

q.add(x);

v[x] = 1;

while(!q.isEmpty()) {

x = q.poll();

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

if(adj[x][i] == 1 && v[i] == 0) {

q.add(i);

v[i] = 1;

}

}

}

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

if(v[i] == 0) return false;

}

return true;

}

public static void dfs(int x, int depth, int sum) {

if(min == 0) return;

if(depth == n) return;

if(sum > total - sum) return;

visit[x] = 1;

if(Math.abs((total - sum) - sum) < min && bfs()) {

min = Math.min(min, Math.abs((total - sum) - sum));

}

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

if(visit[i] == 1) {

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

if(adj[i][j] == 1 && visit[j] == 0) dfs(j, depth + 1, sum + people[j]);

}

}

}

visit[x] = 0;

}

public static void main(String[] args) throws IOException {

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

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

n = Integer.parseInt(tk.nextToken());

people = new int[n];

adj = new int[n][n];

visit = new int[n];

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

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

people[i] = Integer.parseInt(tk.nextToken());

total += people[i];

}

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

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

int k = Integer.parseInt(tk.nextToken());

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

int to = Integer.parseInt(tk.nextToken()) - 1;

adj[i][to] = 1;

adj[to][i] = 1;

}

}

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

dfs(i, 1, people[i]);

}

if(min == 999999) {

System.out.println(-1);

}

else System.out.println(min);

}

}

## [**[백준] 17281번 - ⚾**](https://dheldh77.tistory.com/entry/%EB%B0%B1%EC%A4%80-17281%EB%B2%88-%E2%9A%BE)

2020. 3. 27. 06:28

[17281](https://dheldh77.tistory.com/tag/17281), [BaekJoon](https://dheldh77.tistory.com/tag/BaekJoon), [DFS](https://dheldh77.tistory.com/tag/DFS), [삼성A형](https://dheldh77.tistory.com/tag/%EC%82%BC%EC%84%B1A%ED%98%95), [알고리즘](https://dheldh77.tistory.com/tag/%EC%95%8C%EA%B3%A0%EB%A6%AC%EC%A6%98)

### 문제 >

⚾는 9명으로 이루어진 두 팀이 공격과 수비를 번갈아 하는 게임이다. 하나의 이닝은 공격과 수비로 이루어져 있고, 총 N이닝동안 게임을 진행해야 한다. 한 이닝에 3아웃이 발생하면 이닝이 종료되고, 두 팀이 공격과 수비를 서로 바꾼다.

두 팀은 경기가 시작하기 전까지 타순(타자가 타석에 서는 순서)을 정해야 하고, 경기 중에는 타순을 변경할 수 없다. 9번 타자까지 공을 쳤는데 3아웃이 발생하지 않은 상태면 이닝은 끝나지 않고, 1번 타자가 다시 타석에 선다. 타순은 이닝이 변경되어도 순서를 유지해야 한다. 예를 들어, 2이닝에 6번 타자가 마지막 타자였다면, 3이닝은 7번 타자부터 타석에 선다.

공격은 투수가 던진 공을 타석에 있는 타자가 치는 것이다. 공격 팀의 선수가 1루, 2루, 3루를 거쳐서 홈에 도착하면 1점을 득점한다. 타자가 홈에 도착하지 못하고 1루, 2루, 3루 중 하나에 머물러있을 수 있다. 루에 있는 선수를 주자라고 한다. 이닝이 시작될 때는 주자는 없다.

타자가 공을 쳐서 얻을 수 있는 결과는 안타, 2루타, 3루타, 홈런, 아웃 중 하나이다. 각각이 발생했을 때, 벌어지는 일은 다음과 같다.

* 안타: 타자와 모든 주자가 한 루씩 진루한다.
* 2루타: 타자와 모든 주자가 두 루씩 진루한다.
* 3루타: 타자와 모든 주자가 세 루씩 진루한다.
* 홈런: 타자와 모든 주자가 홈까지 진루한다.
* 아웃: 모든 주자는 진루하지 못하고, 공격 팀에 아웃이 하나 증가한다.

한 야구팀의 감독 아인타는 타순을 정하려고 한다. 아인타 팀의 선수는 총 9명이 있고, 1번부터 9번까지 번호가 매겨져 있다. 아인타는 자신이 가장 좋아하는 선수인 1번 선수를 4번 타자로 미리 결정했다. 이제 다른 선수의 타순을 모두 결정해야 한다. 아인타는 각 선수가 각 이닝에서 어떤 결과를 얻는지 미리 알고 있다. 가장 많은 득점을 하는 타순을 찾고, 그 때의 득점을 구해보자.

### 입력 >

첫째 줄에 이닝 수 N(2 ≤ N ≤ 50)이 주어진다. 둘째 줄부터 N개의 줄에는 각 선수가 각 이닝에서 얻는 결과가 1번 이닝부터 N번 이닝까지 순서대로 주어진다. 이닝에서 얻는 결과는 9개의 정수가 공백으로 구분되어져 있다. 각 결과가 의미하는 정수는 다음과 같다.

* 안타: 1
* 2루타: 2
* 3루타: 3
* 홈런: 4
* 아웃: 0

각 이닝에는 아웃을 기록하는 타자가 적어도 한 명 존재한다.

### 출력 >

아인타팀이 얻을 수 있는 최대 점수를 출력한다.

### 해결방법 >

DFS(깊이우선탐색) 해결할 수 있는 문제이다.

 9명의 선수를 순서대로 배치하는 모든 경우의 수를 찾고 이 때의 점수를 계산한다.

1) 9명의 선수의 순서 배치

DFS를 통해 순서를 배치하는데 이 때, 문제에서 1번 선수는 4번째 타자로 고정되어있으므로 이 경우를 고려해서 탐색한다.

2) 점수 계산

 점수 계산의 경우는 일반적인 야구와 같은 방식으로 계산한다.

 쓰리아웃이 발생하기 전까지 진루한 선수의 수를 카운트하고, 쓰리아웃이 발생했을 때 루에 남아있는 선수의 수를 빼준다.

 스택을 이용해 루에 남아있는 선수의 수를 저장했다

 선수들의 결과를 순서대로 스택에 담고

 쓰리아웃이 발생했을 때 ,다시 마지막 선수부터 결과를 더해 이 값이 3을 초과할때까지

 스택에서 뽑은 선수의 수를 총 진루한 선수의 수에서 빼준다.

 이러한 방식으로 각 선수 배치에 따른 결과를 구하고 이 중 최대값을 출력한다.

### [JAVA]

package baekjoon;

import java.util.\*;

import java.io.\*;

public class BOJ\_17281\_2 {

static int max = 0;

static int[] visit = new int[9];

static int[] order = new int[9];

static int[][] record;

static int n;

public static void play() {

int score = 0;

int j = 6;

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

int out = 0;

int run = 0;

Stack <Integer> st = new Stack<>();

while(out < 3) {

if(record[i][order[j]] == 0) out++;

else {

run++;

st.push(record[i][order[j]]);

}

j++;

if(j > 8) j = 0;

}

int k = 0;

int remain = 0;

while(!st.isEmpty()) {

k += st.pop();

if(k > 3) break;

remain++;

}

run -= remain;

score += run;

}

max = Math.max(score, max);

}

public static void dfs(int x, int depth) {

visit[x] = 1;

order[depth] = x;

if(depth == 8) {

play();

visit[x] = 0;

order[depth] = 0;

return;

}

for(int i = 0; i < 9; i++) {

if(visit[i] == 0) dfs(i, depth + 1);

}

order[depth] = 0;

visit[x] = 0;

}

public static void main(String[] args) throws IOException {

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

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

n = Integer.parseInt(tk.nextToken());

record = new int [n][9];

int[] scores = new int[9];

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

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

for(int j = 0; j < 9; j++) {

record[i][j] = Integer.parseInt(tk.nextToken());

scores[j] += record[i][j];

}

}

int best = 0;

int idx = 0;

for(int i = 0; i < 9; i++) {

if(scores[i] > best) {

idx = i;

best = scores[i];

}

}

order[0] = idx;

visit[idx] = 1;

for(int i = 0; i < 9; i++) {

if(i == idx) continue;

dfs(i, 1);

}

System.out.println(max);

}

}

## [**[백준] 17070번 - 파이프 옮기기 1**](https://dheldh77.tistory.com/entry/%EB%B0%B1%EC%A4%80-17070%EB%B2%88-%ED%8C%8C%EC%9D%B4%ED%94%84-%EC%98%AE%EA%B8%B0%EA%B8%B0-1)

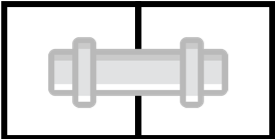
2020. 3. 27. 06:17

[17070](https://dheldh77.tistory.com/tag/17070), [BaekJoon](https://dheldh77.tistory.com/tag/BaekJoon), [DFS](https://dheldh77.tistory.com/tag/DFS), [삼성A형](https://dheldh77.tistory.com/tag/%EC%82%BC%EC%84%B1A%ED%98%95), [알고리즘](https://dheldh77.tistory.com/tag/%EC%95%8C%EA%B3%A0%EB%A6%AC%EC%A6%98)

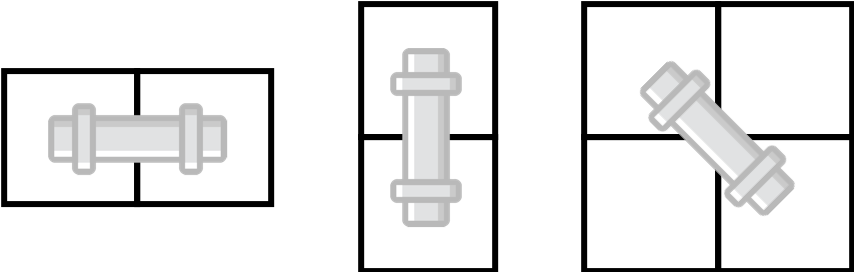
### 문제 >

유현이가 새 집으로 이사했다. 새 집의 크기는 N×N의 격자판으로 나타낼 수 있고, 1×1크기의 정사각형 칸으로 나누어져 있다. 각각의 칸은 (r, c)로 나타낼 수 있다. 여기서 r은 행의 번호, c는 열의 번호이고, 행과 열의 번호는 1부터 시작한다. 각각의 칸은 빈 칸이거나 벽이다.

오늘은 집 수리를 위해서 파이프 하나를 옮기려고 한다. 파이프는 아래와 같은 형태이고, 2개의 연속된 칸을 차지하는 크기이다.



파이프는 회전시킬 수 있으며, 아래와 같이 3가지 방향이 가능하다.

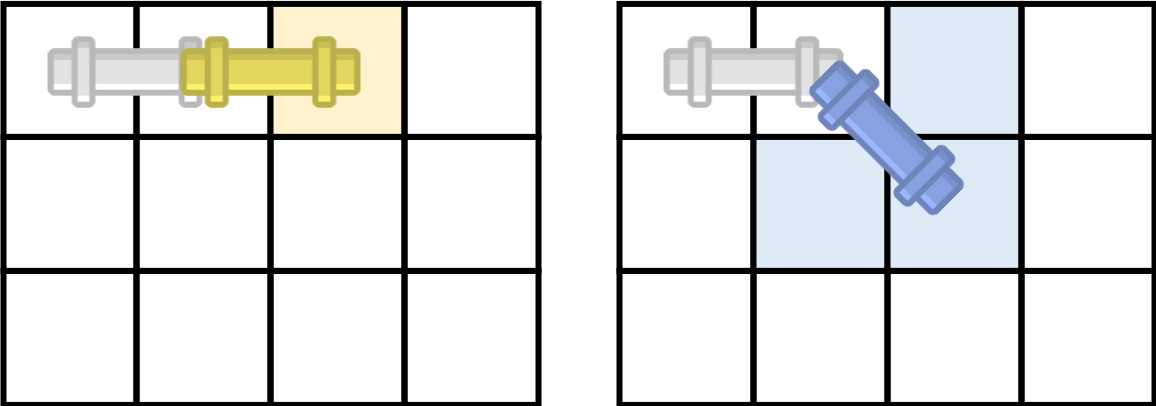


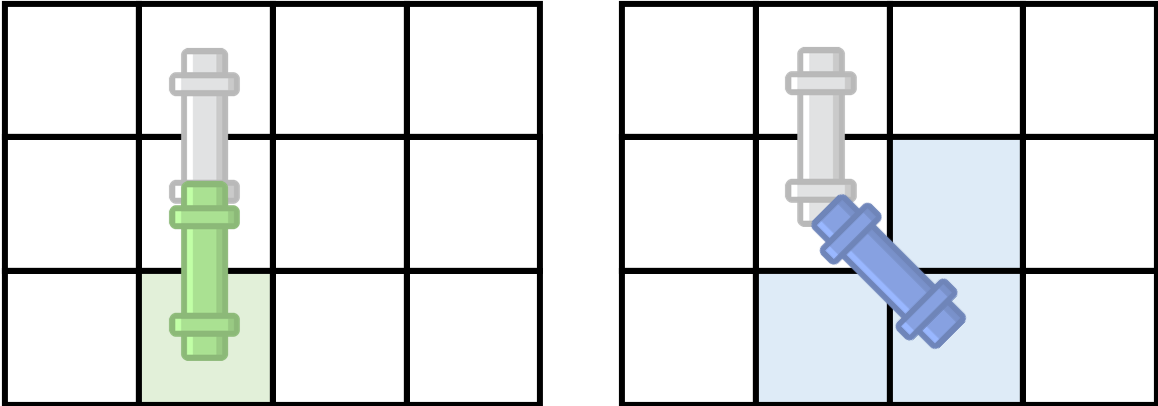
파이프는 매우 무겁기 때문에, 유현이는 파이프를 밀어서 이동시키려고 한다. 벽에는 새로운 벽지를 발랐기 때문에, 파이프가 벽을 긁으면 안 된다. 즉, 파이프는 항상 빈 칸만 차지해야 한다.

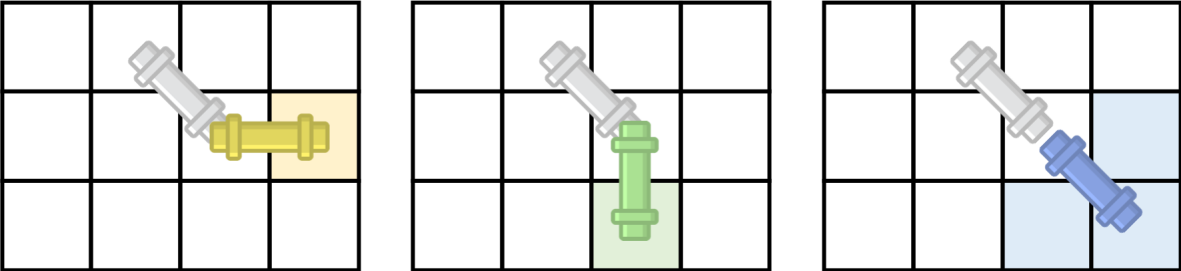
파이프를 밀 수 있는 방향은 총 3가지가 있으며, →, ↘, ↓ 방향이다. 파이프는 밀면서 회전시킬 수 있다. 회전은 45도만 회전시킬 수 있으며, 미는 방향은 오른쪽, 아래, 또는 오른쪽 아래 대각선 방향이어야 한다.

파이프가 가로로 놓여진 경우에 가능한 이동 방법은 총 2가지, 세로로 놓여진 경우에는 2가지, 대각선 방향으로 놓여진 경우에는 3가지가 있다.

아래 그림은 파이프가 놓여진 방향에 따라서 이동할 수 있는 방법을 모두 나타낸 것이고, 꼭 빈 칸이어야 하는 곳은 색으로 표시되어져 있다.

가로

세로

대각선

가장 처음에 파이프는 (1, 1)와 (1, 2)를 차지하고 있고, 방향은 가로이다. 파이프의 한쪽 끝을 (N, N)로 이동시키는 방법의 개수를 구해보자.

### 입력 >

첫째 줄에 집의 크기 N(3 ≤ N ≤ 16)이 주어진다. 둘째 줄부터 N개의 줄에는 집의 상태가 주어진다. 빈 칸은 0, 벽은 1로 주어진다. (1, 1)과 (1, 2)는 항상 빈 칸이다.

### 출력 >

첫째 줄에 파이프의 한쪽 끝을 (N, N)으로 이동시키는 방법의 수를 출력한다. 이동시킬 수 없는 경우에는 0을 출력한다. 방법의 수는 항상 1,000,000보다 작거나 같다.

### 해결방법 >

DFS(깊이우선탐색) 해결할 수 있는 문제이다.

파이프가 놓여있는 방향에 따라 이동할 수 있는 경우의 수가 달라진다.

1. 가로로 놓여있을 경우 :

가로 - 끝점을 기준으로 오른쪽 원소값이 0이거나 배열의 인덱스를 벗어나지 않을 때

대각선 - 끝점을 기준으로 오른쪽, 아래쪽, 대각선 원소값이 0이거나 배열의 인덱스를 벗어나지 않을 때

2. 세로로 놓여있을 경우 :

세로 - 끝점을 기준으로 아래쪽 원소값이 0이거나 배열의 인덱스를 벗어나지 않을 때

대각선 - 끝점을 기준으로 오른쪽, 아래쪽, 대각선 원소값이 0이거나 배열의 인덱스를 벗어나지 않을 때

3. 대각선으로 놓여있을 경우 :

가로 - 끝점을 기준으로 오른쪽 원소값이 0이거나 배열의 인덱스를 벗어나지 않을 때

세로 - 끝점을 기준으로 아래쪽 원소값이 0이거나 배열의 인덱스를 벗어나지 않을 때

대각선 - 끝점을 기준으로 오른쪽, 아래쪽, 대각선 원소값이 0이거나 배열의 인덱스를 벗어나지 않을 때

각 방향에서 가능한 경우를 모든 탐색한다.

### [JAVA]

package baekjoon;

import java.util.\*;

import java.io.\*;

public class BOJ\_17070 {

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

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

static int cnt = 0;

static int n;

static int[][] map;

public static boolean possible(int x, int y, int d) {

int nx = x + dx[d];

int ny = y + dy[d];

if(d == 0 || d == 1) {

if(nx >= 0 && ny >= 0 && nx < n && ny < n && map[nx][ny] == 0) return true;

}

else {

if(nx >= 0 && ny >= 0 && nx < n && ny < n) {

if(map[nx - 1][ny] == 0 && map[nx][ny - 1] == 0 && map[nx][ny] == 0) return true;

}

}

return false;

}

public static void dfs(int x, int y, int d) {

int nx = x + dx[d];

int ny = y + dy[d];

if(nx == n-1 && ny == n-1) {

cnt++;

return;

}

if(d == 0) {

if(possible(nx, ny, 0)) dfs(nx, ny, 0);

if(possible(nx, ny, 2)) dfs(nx, ny, 2);

}

else if(d == 1) {

if(possible(nx, ny, 1)) dfs(nx, ny, 1);

if(possible(nx, ny, 2)) dfs(nx, ny, 2);

}

else {

if(possible(nx, ny, 0)) dfs(nx, ny, 0);

if(possible(nx, ny, 1)) dfs(nx, ny, 1);

if(possible(nx, ny, 2)) dfs(nx, ny, 2);

}

}

public static void main(String[] args) throws IOException{

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

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

n = Integer.parseInt(tk.nextToken());

map = new int[n][n];

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

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

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

map[i][j] = Integer.parseInt(tk.nextToken());

}

}

dfs(0, 0, 0);

System.out.println(cnt);

}

}

## [**[백준] 11559번 - Puyo Puyo**](https://dheldh77.tistory.com/entry/%EB%B0%B1%EC%A4%80-11559%EB%B2%88-Puyo-Puyo)

2020. 3. 26. 06:14

[11559](https://dheldh77.tistory.com/tag/11559), [BaekJoon](https://dheldh77.tistory.com/tag/BaekJoon), [DFS](https://dheldh77.tistory.com/tag/DFS), [시뮬레이션](https://dheldh77.tistory.com/tag/%EC%8B%9C%EB%AE%AC%EB%A0%88%EC%9D%B4%EC%85%98), [알고리즘](https://dheldh77.tistory.com/tag/%EC%95%8C%EA%B3%A0%EB%A6%AC%EC%A6%98)

### 문제 >

뿌요뿌요의 룰은 다음과 같다.

*필드에 여러 가지 색깔의 뿌요를 놓는다. 뿌요는 중력의 영향을 받아 아래에 바닥이나 다른 뿌요가 나올 때까지 아래로 떨어진다.*

*뿌요를 놓고 난 후, 같은 색 뿌요가 4개 이상 상하좌우로 연결되어 있으면 연결된 같은 색 뿌요들이 한꺼번에 없어진다.*

*뿌요들이 없어지고 나서 위에 다른 뿌요들이 있다면, 역시 중력의 영향을 받아 차례대로 아래로 떨어지게 된다.*

*아래로 떨어지고 나서 다시 같은 색의 뿌요들이 4개 이상 모이게 되면 또 터지게 되는데, 터진 후 뿌요들이 내려오고 다시 터짐을 반복할 때마다 1연쇄씩 늘어난다.*

*터질 수 있는 뿌요가 여러 그룹이 있다면 동시에 터져야 하고 여러 그룹이 터지더라도 한번의 연쇄가 추가된다.*

남규는 최근 뿌요뿌요 게임에 푹 빠졌다. 이 게임은 1:1로 붙는 대전게임이라 잘 쌓는 것도 중요하지만, 상대방이 터뜨린다면 연쇄가 몇 번이 될지 바로 파악할 수 있는 능력도 필요하다. 하지만 아직 실력이 부족하여 남규는 자기 필드에만 신경 쓰기 바쁘다. 상대방의 필드가 주어졌을 때, 연쇄가 몇 번 연속으로 일어날지 계산하여 남규를 도와주자!

### 입력 >

12\*6의 문자가 주어진다.

이때 .은 빈공간이고 .이 아닌것은 각각의 색깔의 뿌요를 나타낸다.

R은 빨강, G는 초록, B는 파랑, P는 보라, Y는 노랑이다.(모두 대문자로 주어진다.)

입력으로 주어지는 필드는 뿌요들이 전부 아래로 떨어진 뒤의 상태(즉 뿌요 아래에 빈 칸이 있는 경우는 없음) 이다.

### 출력 >

현재 주어진 상황에서 몇연쇄가 되는지 출력하라. (하나도 터지지 않는다면 0을 출력하면 된다.)

### 해결방법 >

DFS(깊이우선탐색) 또는 BFS(너비우선탐색)알고리즘을 활용해 해결할 수 있는 시뮬레이션 문제이다.

두 가지 과정이 필요하다.

1) 삭제

 2) 이동

1)의 삭제의 경우는 문자가 같은 원소 4개 이상이 인접한 경우를 찾는 과정으로

 BFS나 DFS를 활용해서 4개 이상 인접한 경우를 모두 찾고,

 제거해준다.

2)의 이동의 경우는 삭제된 원소간 빈 공간을 없애주는 과정으로,

 열을 기준으로 빈 공간이 아닌 원소 즉 '.'이 아닌 문자가 입력된 원소들을 스택이나 큐에 담고 맨아래 행부터 다시 쌓아준다.

 비슷한 문제로 프렌즈4블록을 참고하면 좋을 듯하다.

<https://dheldh77.tistory.com/315>

[dheldh77.tistory.com](https://dheldh77.tistory.com/315" \t "_blank)

### [JAVA]

package baekjoon;

import java.util.\*;

import java.io.\*;

public class BOJ\_11559 {

static char[][] map;

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

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

static int depth = 0;

public static class Node{

int x;

int y;

Node(int x, int y){

this.x = x;

this.y = y;

}

}

public static void dfs(char c, int x, int y, int[][] visit, ArrayList<Node> list) {

visit[x][y] = 1;

list.add(new Node(x, y));

depth++;

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

int nx = x + dx[i];

int ny = y + dy[i];

if(nx >= 0 && ny >= 0 && nx < 12 && ny < 6 && visit[nx][ny] == 0 && map[nx][ny] == c) {

dfs(c, nx, ny, visit, list);

}

}

}

public static int bomb() {

int cnt = 0;

for(int i = 0; i < 12; i++) {

for(int j = 0; j < 6; j++) {

if(map[i][j] != '.') {

int[][] visit = new int[12][6];

ArrayList <Node> list = new ArrayList<>();

depth = 0;

dfs(map[i][j], i, j, visit, list);

if(depth >= 4) {

for(int k = 0; k < list.size(); k++) {

Node node = list.get(k);

map[node.x][node.y] = '.';

}

cnt += list.size();

}

}

}

}

return cnt;

}

public static char[][] move() {

char[][] n\_map = new char[12][6];

for(int i = 0; i < 12; i++) {

for(int j = 0; j < 6; j++) {

n\_map[i][j] = '.';

}

}

for(int i = 0; i < 6; i++) {

Stack <Character> st = new Stack<>();

for(int j = 0; j < 12; j++) {

if(map[j][i] != '.') st.push(map[j][i]);

}

int row = 11;

while(!st.isEmpty()) {

n\_map[row][i] = st.pop();

row--;

}

}

return n\_map;

}

public static void main(String[] args) throws IOException{

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

map = new char[12][6];

for(int i = 0; i < 12; i++) {

String s = br.readLine();

for(int j = 0; j < 6; j++) {

map[i][j] = s.charAt(j);

}

}

int cnt = 0;

while(true) {

int flag = bomb();

if(flag == 0) break;

cnt++;

map = move();

}

System.out.println(cnt);

}

}

## [**[백준] 6603번 - 로또**](https://dheldh77.tistory.com/entry/%EB%B0%B1%EC%A4%80-6603%EB%B2%88-%EB%A1%9C%EB%98%90)

2020. 3. 26. 06:04

[6603](https://dheldh77.tistory.com/tag/6603), [BaekJoon](https://dheldh77.tistory.com/tag/BaekJoon), [DFS](https://dheldh77.tistory.com/tag/DFS), [알고리즘](https://dheldh77.tistory.com/tag/%EC%95%8C%EA%B3%A0%EB%A6%AC%EC%A6%98)

### 문제 >

독일 로또는 {1, 2, ..., 49}에서 수 6개를 고른다.

로또 번호를 선택하는데 사용되는 가장 유명한 전략은 49가지 수 중 k(k>6)개의 수를 골라 집합 S를 만든 다음 그 수만 가지고 번호를 선택하는 것이다.

예를 들어, k=8, S={1,2,3,5,8,13,21,34}인 경우 이 집합 S에서 수를 고를 수 있는 경우의 수는 총 28가지이다. ([1,2,3,5,8,13], [1,2,3,5,8,21], [1,2,3,5,8,34], [1,2,3,5,13,21], ..., [3,5,8,13,21,34])

집합 S와 k가 주어졌을 때, 수를 고르는 모든 방법을 구하는 프로그램을 작성하시오.

### 입력 >

입력은 여러 개의 테스트 케이스로 이루어져 있다. 각 테스트 케이스는 한 줄로 이루어져 있다. 첫 번째 수는 k (6 < k < 13)이고, 다음 k개 수는 집합 S에 포함되는 수이다. S의 원소는 오름차순으로 주어진다.

입력의 마지막 줄에는 0이 하나 주어진다.

### 출력 >

각 테스트 케이스마다 수를 고르는 모든 방법을 출력한다. 이때, 사전 순으로 출력한다.

각 테스트 케이스 사이에는 빈 줄을 하나 출력한다.

### 해결방법 >

n개의 숫자가 주어졌을 때, n개 중 6개의 조합 즉, C(n, 6)을 구하는 문제이다.

DFS(깊이우선탐색)으로 가능한 모든 조합을 찾는다.

### [JAVA]

package baekjoon;

import java.util.\*;

import java.io.\*;

public class BOJ\_6603 {

static BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

static BufferedWriter bw = new BufferedWriter(new OutputStreamWriter(System.out));

public static void dfs (int x, int[] nums, int depth, int[] visit) throws IOException{

visit[x] = 1;

if(depth == 6) {

for(int i = 0; i < visit.length; i++) {

if(visit[i] == 1) bw.write(nums[i] + " ");

}

bw.write("\n");

visit[x] = 0;

return;

}

for(int i = x; i < nums.length; i++) {

if(visit[i] == 0) dfs(i, nums, depth + 1, visit);

}

visit[x] = 0;

}

public static void main(String[] args) throws IOException {

StringTokenizer tk;

while(true) {

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

int n = Integer.parseInt(tk.nextToken());

if(n == 0) break;

int[] nums = new int[n];

int[] visit = new int[n];

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

nums[i] = Integer.parseInt(tk.nextToken());

}

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

visit[i] = 1;

dfs(i, nums, 1, visit);

}

bw.write("\n");

}

bw.flush();

}

}

## [**[백준] 1963번 - 소수 경로**](https://dheldh77.tistory.com/entry/%EB%B0%B1%EC%A4%80-1963%EB%B2%88-%EC%86%8C%EC%88%98-%EA%B2%BD%EB%A1%9C)

2020. 3. 26. 05:56

[BaekJoon](https://dheldh77.tistory.com/tag/BaekJoon), [BFS](https://dheldh77.tistory.com/tag/BFS), [알고리즘](https://dheldh77.tistory.com/tag/%EC%95%8C%EA%B3%A0%EB%A6%AC%EC%A6%98)

### 문제 >

소수를 유난히도 좋아하는 창영이는 게임 아이디 비밀번호를 4자리 ‘소수’로 정해놓았다. 어느 날 창영이는 친한 친구와 대화를 나누었는데:

* “이제 슬슬 비번 바꿀 때도 됐잖아”
* “응 지금은 1033으로 해놨는데... 다음 소수를 무엇으로 할지 고민중이야"
* “그럼 8179로 해”
* “흠... 생각 좀 해볼게. 이 게임은 좀 이상해서 비밀번호를 한 번에 한 자리 밖에 못 바꾼단 말이야. 예를 들어 내가 첫 자리만 바꾸면 8033이 되니까 소수가 아니잖아. 여러 단계를 거쳐야 만들 수 있을 것 같은데... 예를 들면... 1033 1733 3733 3739 3779 8779 8179처럼 말이야.”
* “흠...역시 소수에 미쳤군. 그럼 아예 프로그램을 짜지 그래. 네 자리 소수 두 개를 입력받아서 바꾸는데 몇 단계나 필요한지 계산하게 말야.”
* “귀찮아”

그렇다. 그래서 여러분이 이 문제를 풀게 되었다. 입력은 항상 네 자리 소수만(1000 이상) 주어진다고 가정하자. 주어진 두 소수 A에서 B로 바꾸는 과정에서도 항상 네 자리 소수임을 유지해야 하고, ‘네 자리 수’라 하였기 때문에 0039 와 같은 1000 미만의 비밀번호는 허용되지 않는다.

### 입력 >

첫 줄에 test case의 수 T가 주어진다. 다음 T줄에 걸쳐 각 줄에 1쌍씩 네 자리 소수가 주어진다.

### 출력 >

각 test case에 대해 두 소수 사이의 변환에 필요한 최소 회수를 출력한다. 불가능한 경우 Impossible을 출력한다.

### 해결방법 >

네자리 소수 두개가 주어졌을 때, 첫번째 소수에서 두번째 소수까지의 최단 거리를 구하는 문제로 BFS(너비우선탐색) 알고리즘으로 해결.

 일단은 네자리로 만들 수 있는 소수를 모두 찾는다.

 첫번째 소수를 시작으로 인접한 소수를 탐색하는데 이 때 인접한 소수란 네자리 수 중에서 숫자 하나만 차이가 나는 소수이다.

 인접한 소수들을 계속해서 너비우선탐색해서 두번째 소수가 나올때까지 탐색을 하고, 최단거리를 구한다.

### [JAVA]

package baekjoon;

import java.util.\*;

import java.io.\*;

public class BOJ\_1963 {

static ArrayList <Integer> p = new ArrayList<>();

public static int solve(int x, int y) {

int answer = -1;

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

int[] visit = new int[10000];

q.add(x);

visit[x] = 1;

while(!q.isEmpty()) {

int nx = q.poll();

String num1 = String.valueOf(nx);

if(nx == y) {

answer = visit[nx] - 1;

break;

}

for(int i = 0; i < p.size(); i++) {

if(visit[p.get(i)] == 0) {

String num2 = String.valueOf(p.get(i));

int cnt = 0;

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

if(num1.charAt(j) != num2.charAt(j)) cnt++;

}

if(cnt == 1) {

q.add(p.get(i));

visit[p.get(i)] = visit[nx] + 1;

}

}

}

}

return answer;

}

public static void main(String[] args) throws IOException {

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

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

for(int i = 1000; i < 10000; i++) {

int flag = 0;

for(int j = 2; j <= Math.sqrt(i); j++) {

if(i % j == 0) {

flag = 1;

break;

}

}

if(flag == 0) p.add(i);

}

int tc = Integer.parseInt(tk.nextToken());

for(int t = 0; t < tc; t++) {

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

int x = Integer.parseInt(tk.nextToken());

int y = Integer.parseInt(tk.nextToken());

int r = solve(Math.min(x, y), Math.max(x, y));

if(r == -1) System.out.println("Impossible");

else System.out.println(r);

}

}

}

## [**[백준] 2644번 - 촌수계산**](https://dheldh77.tistory.com/entry/%EB%B0%B1%EC%A4%80-2644%EB%B2%88-%EC%B4%8C%EC%88%98%EA%B3%84%EC%82%B0)

2020. 3. 26. 05:52

[BaekJoon](https://dheldh77.tistory.com/tag/BaekJoon), [BFS](https://dheldh77.tistory.com/tag/BFS), [알고리즘](https://dheldh77.tistory.com/tag/%EC%95%8C%EA%B3%A0%EB%A6%AC%EC%A6%98)

### 문제 >

우리 나라는 가족 혹은 친척들 사이의 관계를 촌수라는 단위로 표현하는 독특한 문화를 가지고 있다. 이러한 촌수는 다음과 같은 방식으로 계산된다. 기본적으로 부모와 자식 사이를 1촌으로 정의하고 이로부터 사람들 간의 촌수를 계산한다. 예를 들면 나와 아버지, 아버지와 할아버지는 각각 1촌으로 나와 할아버지는 2촌이 되고, 아버지 형제들과 할아버지는 1촌, 나와 아버지 형제들과는 3촌이 된다.

여러 사람들에 대한 부모 자식들 간의 관계가 주어졌을 때, 주어진 두 사람의 촌수를 계산하는 프로그램을 작성하시오.

### 입력 >

사람들은 1, 2, 3, …, n (1≤n≤100)의 연속된 번호로 각각 표시된다. 입력 파일의 첫째 줄에는 전체 사람의 수 n이 주어지고, 둘째 줄에는 촌수를 계산해야 하는 서로 다른 두 사람의 번호가 주어진다. 그리고 셋째 줄에는 부모 자식들 간의 관계의 개수 m이 주어진다. 넷째 줄부터는 부모 자식간의 관계를 나타내는 두 번호 x,y가 각 줄에 나온다. 이때 앞에 나오는 번호 x는 뒤에 나오는 정수 y의 부모 번호를 나타낸다.

각 사람의 부모는 최대 한 명만 주어진다.

### 출력 >

입력에서 요구한 두 사람의 촌수를 나타내는 정수를 출력한다. 어떤 경우에는 두 사람의 친척 관계가 전혀 없어 촌수를 계산할 수 없을 때가 있다. 이때에는 -1을 출력해야 한다.

### 해결방법 >

두 노드 간 최단거리를 구하는 문제로 BFS(너비우선탐색)알고리즘으로 해결.

입력된 데이터로 인접행렬을 만들고 최단거리를 구한다.

### [JAVA]

package baekjoon;

import java.util.\*;

import java.io.\*;

public class BOJ\_2644 {

public static void main(String[] args) throws IOException{

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

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

int n = Integer.parseInt(tk.nextToken());

int[] a = new int[n+1];

int[][] map = new int[n+1][n+1];

int[] visit = new int[n+1];

int result = -1;

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

int f = Integer.parseInt(tk.nextToken());

int e = Integer.parseInt(tk.nextToken());

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

int m = Integer.parseInt(tk.nextToken());

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

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

int x = Integer.parseInt(tk.nextToken());

int y = Integer.parseInt(tk.nextToken());

map[x][y] = 1;

map[y][x] = 1;

}

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

for(int i = 1; i <= n; i++) {

if(map[f][i] == 1) {

q.add(i);

visit[i] = 1;

}

}

int flag = 0;

while(!q.isEmpty()) {

int x = q.poll();

for(int i = 1; i <= n; i++) {

if(map[x][i] == 1 && visit[i] == 0) {

q.add(i);

visit[i] = visit[x] + 1;

if(i == e) {

result = visit[i];

flag = 1;

break;

}

}

}

if(flag == 1) break;

}

System.out.println(result);

}

}

## [**[백준] 17142번 - 연구소 3**](https://dheldh77.tistory.com/entry/%EB%B0%B1%EC%A4%80-17142%EB%B2%88-%EC%97%B0%EA%B5%AC%EC%86%8C-3)

2020. 3. 19. 03:25

[17142](https://dheldh77.tistory.com/tag/17142), [BaekJoon](https://dheldh77.tistory.com/tag/BaekJoon), [BFS](https://dheldh77.tistory.com/tag/BFS), [DFS](https://dheldh77.tistory.com/tag/DFS), [SW역량테스트](https://dheldh77.tistory.com/tag/SW%EC%97%AD%EB%9F%89%ED%85%8C%EC%8A%A4%ED%8A%B8), [알고리즘](https://dheldh77.tistory.com/tag/%EC%95%8C%EA%B3%A0%EB%A6%AC%EC%A6%98)

### 문제 >

인체에 치명적인 바이러스를 연구하던 연구소에 승원이가 침입했고, 바이러스를 유출하려고 한다. 바이러스는 활성 상태와 비활성 상태가 있다. 가장 처음에 모든 바이러스는 비활성 상태이고, 활성 상태인 바이러스는 상하좌우로 인접한 모든 빈 칸으로 동시에 복제되며, 1초가 걸린다. 승원이는 연구소의 바이러스 M개를 활성 상태로 변경하려고 한다.

연구소는 크기가 N×N인 정사각형으로 나타낼 수 있으며, 정사각형은 1×1 크기의 정사각형으로 나누어져 있다. 연구소는 빈 칸, 벽, 바이러스로 이루어져 있으며, 벽은 칸 하나를 가득 차지한다. 활성 바이러스가 비활성 바이러스가 있는 칸으로 가면 비활성 바이러스가 활성으로 변한다.

예를 들어, 아래와 같이 연구소가 생긴 경우를 살펴보자. 0은 빈 칸, 1은 벽, 2는 바이러스의 위치이다.



M = 3이고, 바이러스를 아래와 같이 활성 상태로 변경한 경우 6초면 모든 칸에 바이러스를 퍼뜨릴 수 있다. 벽은 -, 비활성 바이러스는 \*, 활성 바이러스는 0, 빈 칸은 바이러스가 퍼지는 시간으로 표시했다.



시간이 최소가 되는 방법은 아래와 같고, 4초만에 모든 칸에 바이러스를 퍼뜨릴 수 있다.



연구소의 상태가 주어졌을 때, 모든 빈 칸에 바이러스를 퍼뜨리는 최소 시간을 구해보자.

### 입력 >

첫째 줄에 연구소의 크기 N(4 ≤ N ≤ 50), 놓을 수 있는 바이러스의 개수 M(1 ≤ M ≤ 10)이 주어진다.

둘째 줄부터 N개의 줄에 연구소의 상태가 주어진다. 0은 빈 칸, 1은 벽, 2는 바이러스를 놓을 수 있는 위치이다. 2의 개수는 M보다 크거나 같고, 10보다 작거나 같은 자연수이다.

### 출력 >

연구소의 모든 빈 칸에 바이러스가 있게 되는 최소 시간을 출력한다. 바이러스를 어떻게 놓아도 모든 빈 칸에 바이러스를 퍼뜨릴 수 없는 경우에는 -1을 출력한다.

### 해결방법 >

DFS(깊이우선탐색)과 BFS(너비우선탐색)을 사용해서 해결할 수 있는 문제이다.

두 가지 과정이 필요하다.

1) DFS로 활성화할 m개의 바이러스를 선택한다.

 2) BFS로 선택된 m개의 바이러스를 이동거리중 가장 긴 경로를 찾는다.

 단, 한 가지 고려해야할 점은 **빈 칸에 바이러스를 확산**시키는 시간을 찾는 문제이기 때문에 앞으로 탐색할 노드가 비활성화 바이러스만 남아있다면 탐색을 종료한다.

예로 예제 7번이 있다.

### [JAVA]

package baekjoon;

import java.util.\*;

import java.io.\*;

public class BOJ\_17142 {

static int[][] map;

static ArrayList <Node> virus = new ArrayList<>();

static int n;

static int m;

static int min = -1;

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

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

public static class Node{

int x;

int y;

Node(int x, int y){

this.x = x;

this.y = y;

}

}

public static void insfect(int[] s) {

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

int[][] n\_map = new int[n][n];

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

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

n\_map[i][j] = map[i][j];

}

}

for(int i = 0; i < s.length; i++) {

if(s[i] == 0)

n\_map[virus.get(i).x][virus.get(i).y] = -1;

}

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

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

if(n\_map[i][j] == 2) {

q.add(new Node(i, j));

}

}

}

while(!q.isEmpty()) {

int zero = 0;

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

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

if(n\_map[i][j] == 0) {

zero = 1;

break;

}

}

}

if(zero == 0) break;

Node node = q.poll();

int x = node.x;

int y = node.y;

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

int nx = x + dx[i];

int ny = y + dy[i];

if(nx >= 0 && ny >= 0 && nx < n && ny < n && (n\_map[nx][ny] == 0 || n\_map[nx][ny] == -1)) {

q.add(new Node(nx, ny));

n\_map[nx][ny] = n\_map[x][y] + 1;

}

}

}

int flag = 0;

int max = 0;

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

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

if(n\_map[i][j] == 0) {

flag = 1;

break;

}

max = Math.max(n\_map[i][j], max);

}

}

if(flag == 0) {

if(min == -1) min = max - 2;

else min = Math.min(max - 2, min);

}

}

public static void activate(int v, int flag, int depth, int[] s) {

if(v >= s.length) return;

int[] selected = s.clone();

selected[v] = flag;

if(depth == m) {

insfect(selected);

return;

}

activate(v + 1, 0, depth, selected);

activate(v + 1, 1, depth + 1, selected);

}

public static void main(String[] args) throws IOException {

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

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

n = Integer.parseInt(tk.nextToken());

m = Integer.parseInt(tk.nextToken());

map = new int[n][n];

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

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

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

map[i][j] = Integer.parseInt(tk.nextToken());

if(map[i][j] == 2) virus.add(new Node(i, j));

}

}

int[] selected = new int[virus.size()];

activate(0, 0, 0, selected);

activate(0, 1, 1, selected);

System.out.println(min);

}

}

## [**[백준] 13460번 - 구슬 탈출 2**](https://dheldh77.tistory.com/entry/%EB%B0%B1%EC%A4%80-13460%EB%B2%88-%EA%B5%AC%EC%8A%AC-%ED%83%88%EC%B6%9C-2)

2020. 3. 19. 03:16

[13460](https://dheldh77.tistory.com/tag/13460), [BaekJoon](https://dheldh77.tistory.com/tag/BaekJoon), [DFS](https://dheldh77.tistory.com/tag/DFS), [SW역량테스트](https://dheldh77.tistory.com/tag/SW%EC%97%AD%EB%9F%89%ED%85%8C%EC%8A%A4%ED%8A%B8), [시뮬레이션](https://dheldh77.tistory.com/tag/%EC%8B%9C%EB%AE%AC%EB%A0%88%EC%9D%B4%EC%85%98), [알고리즘](https://dheldh77.tistory.com/tag/%EC%95%8C%EA%B3%A0%EB%A6%AC%EC%A6%98)

### 문제 >

스타트링크에서 판매하는 어린이용 장난감 중에서 가장 인기가 많은 제품은 구슬 탈출이다. 구슬 탈출은 직사각형 보드에 빨간 구슬과 파란 구슬을 하나씩 넣은 다음, 빨간 구슬을 구멍을 통해 빼내는 게임이다.

보드의 세로 크기는 N, 가로 크기는 M이고, 편의상 1×1크기의 칸으로 나누어져 있다. 가장 바깥 행과 열은 모두 막혀져 있고, 보드에는 구멍이 하나 있다. 빨간 구슬과 파란 구슬의 크기는 보드에서 1×1크기의 칸을 가득 채우는 사이즈이고, 각각 하나씩 들어가 있다. 게임의 목표는 빨간 구슬을 구멍을 통해서 빼내는 것이다. 이때, 파란 구슬이 구멍에 들어가면 안 된다.

이때, 구슬을 손으로 건드릴 수는 없고, 중력을 이용해서 이리 저리 굴려야 한다. 왼쪽으로 기울이기, 오른쪽으로 기울이기, 위쪽으로 기울이기, 아래쪽으로 기울이기와 같은 네 가지 동작이 가능하다.

각각의 동작에서 공은 동시에 움직인다. 빨간 구슬이 구멍에 빠지면 성공이지만, 파란 구슬이 구멍에 빠지면 실패이다. 빨간 구슬과 파란 구슬이 동시에 구멍에 빠져도 실패이다. 빨간 구슬과 파란 구슬은 동시에 같은 칸에 있을 수 없다. 또, 빨간 구슬과 파란 구슬의 크기는 한 칸을 모두 차지한다. 기울이는 동작을 그만하는 것은 더 이상 구슬이 움직이지 않을 때 까지이다.

보드의 상태가 주어졌을 때, 최소 몇 번 만에 빨간 구슬을 구멍을 통해 빼낼 수 있는지 구하는 프로그램을 작성하시오.

### 입력 >

첫 번째 줄에는 보드의 세로, 가로 크기를 의미하는 두 정수 N, M (3 ≤ N, M ≤ 10)이 주어진다. 다음 N개의 줄에 보드의 모양을 나타내는 길이 M의 문자열이 주어진다. 이 문자열은 '.', '#', 'O', 'R', 'B' 로 이루어져 있다. '.'은 빈 칸을 의미하고, '#'은 공이 이동할 수 없는 장애물 또는 벽을 의미하며, 'O'는 구멍의 위치를 의미한다. 'R'은 빨간 구슬의 위치, 'B'는 파란 구슬의 위치이다.

입력되는 모든 보드의 가장자리에는 모두 '#'이 있다. 구멍의 개수는 한 개 이며, 빨간 구슬과 파란 구슬은 항상 1개가 주어진다.

### 출력 >

최소 몇 번 만에 빨간 구슬을 구멍을 통해 빼낼 수 있는지 출력한다. 만약, 10번 이하로 움직여서 빨간 구슬을 구멍을 통해 빼낼 수 없으면 -1을 출력한다.

### 해결방법 >

DFS(깊이우선탐색) 알고리즘으로 해결할 수 있는 시뮬레이션 문제이다.

문제를 해결하기 위해서 크게 세 가지를 생각해줘야 했다.

**1) 보드의 회전 방향**

  한 방향으로 두 구슬 중 하나라도 움직일 수 있다면 회전시켜줘고 DFS탐색을 시작한다.

  구슬이 움직일 수 있는 조건은 그 방향으로 벽이 없고, 다른 구슬이 존재하지 않아야 한다.

   또 한 가지, 코드 실행 속도를 위해서 바로 이전에 이동했던 방향의 역방향으로 다시 이동하지 않는다.

**2) 두 구슬의 움직임**

   구슬은 각각 독립적으로 움직이면서 서로의 영향을 받는다.

   즉 구슬 주변 상황에 따라서 움직이는 길이는 달라지지만,

   서로 인접한 위치에 있다면 앞에 있는 구슬이 움직이기 전까지 뒤에 있는 구슬이 움직일 수 없다.

   처음에는 고민을 많이 했다.

   만약 왼쪽으로 움직여야하는데 구슬이 파란색, 빨간색 순서로 붙어 있다면 파란색 부터 움직여줘야하고,

   반대라면 또 빨간색부터 옮겨줘야하고 그럼 그때마다 조건문을 달아줘야하는 건가...

   깊게 생각할 필요가 없었다.

   그냥 두 구슬이 더 이상 이동할 방향이 없을 때까지 계속해서 이동시켜주며면되는 문제였다.

   다른 구슬 때문에 막혀있더라고 다른 구슬이 움직이고 나면 이동할 자리가 생긴다.

**3) 탐색의 종료조건**

  처음에는 생각했던 조건은 세 가지였다.

  1) 10번 이상 회전을 시켰을 때 -> 실패 조건

  2) 파란색이 빠져나왔을 때 -> 실패 조건

   3) 빨간색만 빠져나왔을 때 -> 성공 조건

   이렇게 단순하게 생각한게 문제였다.

   예제에 없는 반례로, 빨간색이 먼저 빠져나오고 난 뒤에 파란색 구슬이 계속해서 이동해 빠져나오는 경우이다.

   빨간색 구슬이 빠져나왔다면 빨간색 구슬을 제거해주고 파란색 구슬을 계속해서 이동시킨 뒤

   파란색 구슬도 빠져나오는지 확인을 해야한다.

   이러한 반례나 공의 이동방식 등 여러가지로 도움이 되는 문제였다.

### [JAVA]

package baekjoon;

import java.util.\*;

import java.io.\*;

public class BOJ\_13460\_2 {

static int n;

static int m;

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

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

static int min = -1;

static int tx;

static int ty;

public static boolean possible(int x, int y, int d, char[][] map) {

int nx = x + dx[d];

int ny = y + dy[d];

if(nx > 0 && ny > 0 && nx < n - 1 && ny < m - 1 && (map[nx][ny] == '.' || map[nx][ny] == 'O')) return true;

else return false;

}

public static void dfs(int rx, int ry, int bx, int by, int d, char[][] map, int cnt) {

if(cnt > 10) return;

char[][] n\_map = new char[n][m];

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

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

n\_map[i][j] = map[i][j];

}

}

int nrx = rx, nry = ry;

int nbx = bx, nby = by;

int[] flag = new int[2];

if(rx == tx && ry == ty) flag[0] = 1;

if(bx == tx && by == ty) flag[1] = 1;

while((flag[0] == 0 && possible(rx, ry, d, n\_map)) || possible(bx, by, d, n\_map)) {

if(flag[0] == 0 && possible(rx, ry, d, n\_map)) {

nrx = rx + dx[d];

nry = ry + dy[d];

n\_map[rx][ry] = '.';

n\_map[nrx][nry] = 'R';

rx = nrx;

ry = nry;

if(rx == tx && ry == ty) {

flag[0] = 1;

n\_map[rx][ry] = 'O';

}

}

if(possible(bx, by, d, n\_map)) {

nbx = bx + dx[d];

nby = by + dy[d];

n\_map[bx][by] = '.';

n\_map[nbx][nby] = 'B';

bx = nbx;

by = nby;

if(bx == tx && by == ty) {

flag[1] = 1;

return;

}

}

}

if(flag[0] == 1) {

if(min == -1) min = cnt;

else min = Math.min(cnt, min);

return;

}

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

if(d == 0 && i == 2) continue;

if(d == 1 && i == 3) continue;

if(d == 2 && i == 4) continue;

if(d == 3 && i == 1) continue;

if(possible(nrx, nry, i, n\_map) || possible(nbx, nby, i, n\_map)) {

dfs(nrx, nry, nbx, nby, i, n\_map, cnt + 1);

}

}

}

public static void main(String[] args) throws IOException {

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

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

n = Integer.parseInt(tk.nextToken());

m = Integer.parseInt(tk.nextToken());

char[][] map = new char[n][m];

int rx = 0, ry = 0;

int bx = 0, by = 0;

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

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

String s = tk.nextToken();

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

map[i][j] = s.charAt(j);

if(map[i][j] == 'R') {

rx = i;

ry = j;

}

if(map[i][j] == 'B') {

bx = i;

by = j;

}

if(map[i][j] == 'O') {

tx = i;

ty = j;

}

}

}

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

if(possible(rx, ry, i, map) || possible(bx, by, i, map)) {

dfs(rx, ry, bx, by, i, map, 1);

}

}

System.out.println(min);

}

}

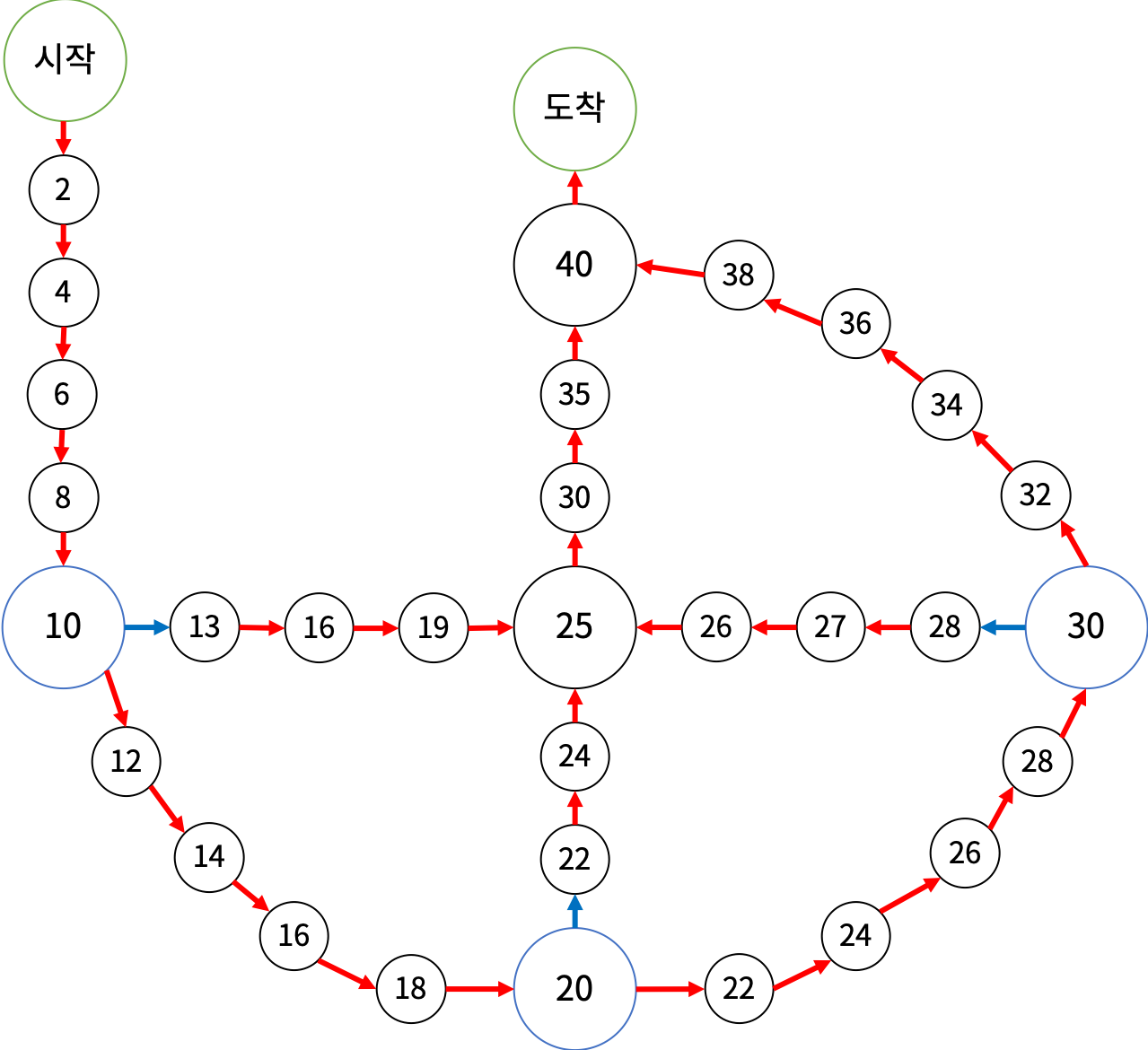
## [**[백준] 17825번 - 주사위 윷놀이**](https://dheldh77.tistory.com/entry/%EB%B0%B1%EC%A4%80-17825%EB%B2%88-%EC%A3%BC%EC%82%AC%EC%9C%84-%EC%9C%B7%EB%86%80%EC%9D%B4)

2020. 3. 18. 02:35

[17825](https://dheldh77.tistory.com/tag/17825), [Backtracking](https://dheldh77.tistory.com/tag/Backtracking), [BaekJoon](https://dheldh77.tistory.com/tag/BaekJoon), [DFS](https://dheldh77.tistory.com/tag/DFS), [SW역량테스트](https://dheldh77.tistory.com/tag/SW%EC%97%AD%EB%9F%89%ED%85%8C%EC%8A%A4%ED%8A%B8), [알고리즘](https://dheldh77.tistory.com/tag/%EC%95%8C%EA%B3%A0%EB%A6%AC%EC%A6%98)

### 문제 >

주사위 윷놀이는 다음과 같은 게임판에서 하는 게임이다.



* 처음에는 시작 칸에 말 4개가 있다.
* 말은 게임판에 그려진 화살표의 방향대로만 이동할 수 있다. 말이 파란색 칸에서 이동을 시작하면 파란색 화살표를 타야 하고, 이동하는 도중이거나 파란색이 아닌 칸에서 이동을 시작하면 빨간색 화살표를 타야 한다. 말이 도착 칸으로 이동하면 주사위에 나온 수와 관계 없이 이동을 마친다.
* 게임은 10개의 턴으로 이루어진다. 매 턴마다 1부터 5까지 한 면에 하나씩 적혀있는 5면체 주사위를 굴리고, 도착 칸에 있지 않은 말을 하나 골라 주사위에 나온 수만큼 이동시킨다.
* 말이 이동을 마치는 칸에 다른 말이 있으면 그 말은 고를 수 없다. 단, 이동을 마치는 칸이 도착 칸이면 고를 수 있다.
* 말이 이동을 마칠 때마다 칸에 적혀있는 수가 점수에 추가된다.

주사위에서 나올 수 10개를 미리 알고 있을 때, 얻을 수 있는 점수의 최댓값을 구해보자.

### 입력 >

첫째 줄에 주사위에서 나올 수 10개가 순서대로 주어진다.

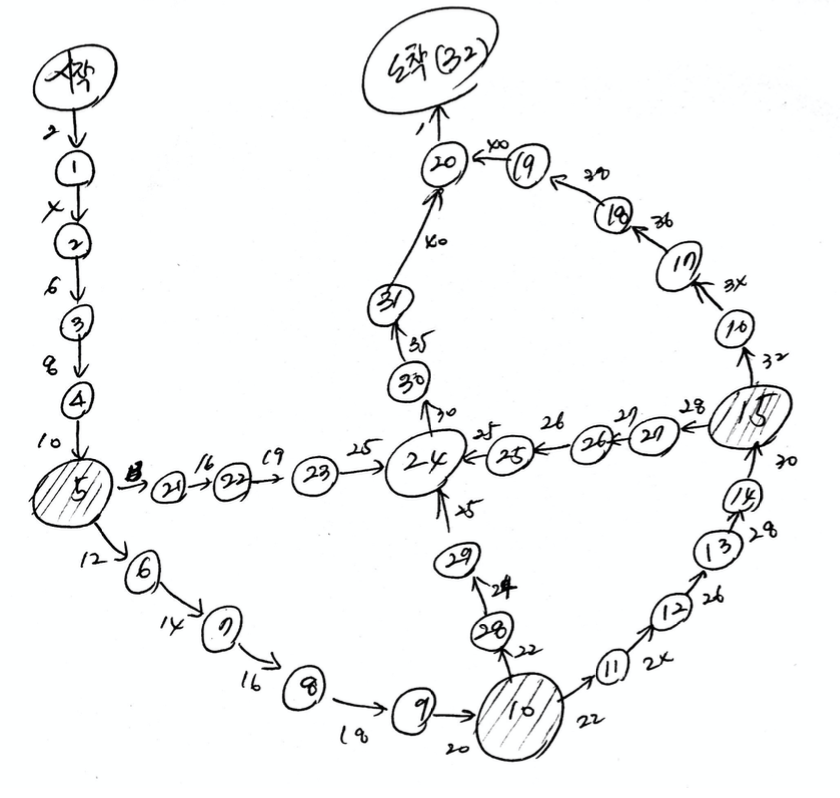
### 출력 >

얻을 수 있는 점수의 최댓값을 출력한다.

### 해결방법 >

DFS(깊이우선탐색)알고리즘으로 문제를 해결했다.

일단은 문제에 주어진 판을 아래와 같이 다시 만들었다.



각 노드마다 번호를 매기고, 문제에서 주어진 숫자는 간선의 가중치로 생각하는 방법이다.

 따라서 33 X 33 크기의 인접행렬을 만들고 간선의 가중치를 저장했다.

 지도 정보를 만들었다면 이제 최대값을 찾아야하는데 두 가지 과정이 필요했다.

 1) 깊이 우선 탐색을 통해 말이 움직이는 순서의 모든 경우의 수를 찾는다.

 2) 각 경우마다 깊이 우선 탐색을 사용해서 주사위의 숫자만큼 이동한다.

 각 과정을 자세히 살펴보면

**1) 깊이 우선 탐색을 통해 말이 움직이는 순서의 모든 경우의 수를 찾는다.**

1)번 과정 같은 경우는 주사위 숫자 10개에 따라 4개의 말을 선택할 수 있기 때문에 중복을 허용하는 순열과 같다.

 경우의 수는 4^10으로 연산 수가 크다.

 2)번 과정에서도 깊이우선탐색을 해야하는데 접근 방식이 잘못되었는가 생각을 많이 했지만 백트레킹으로 해결할 수 있었다.

**2) 각 경우마다 깊이 우선 탐색을 사용해서 주사위의 숫자만큼 이동한다.**

 2)번 과정은 선택된 말로 i번째 주사위 수만큼 움직이는 것이다. 다시말해 주사위 수만큼 인접한 노드로 깊이있게 방문한다.

 이 부분을 깊이우선탐색으로 해결한 이유 인접한 노드가 순서대로 있지않고,

 5, 10, 15번째 노드, 즉 경로가 두 갈래로 나뉘는 노드가 있기  때문이다.

 노드가 두 갈래로 나뉜다는 말은 간선의 정보가 두 개가 존재한다는 것인데, 두 가지 조건으로 해결할 수 있다

 a) 탐색을 시작한 노드가 두 갈래로 나뉘는 노드라면, 노드 값이 큰 노드를 방문한다.

게임판에 대한 정보를 새로 만들 때, 두 갈래에서 우선으로 방문해야 하는 노드의 원소 값을 크게 만들었기 때문에 가능하다.

 b) 탐색 중간에 방문한 노드가 두 갈래로 나뉘는 노드라면, 노드 값이 작은 노드를 방문한다.

 지금까지 말한 부분만 구현한다면 연산 시간이 오래 걸린다. 따라서 1)번 과정의 백트레킹할 조건을 만들어준다. 조건은 아래와 같다.

 a) 주사위 수만큼 이동했을 때, 다른 말이 존재한다면 경우(문제에서 요구한 조건)

 b) 현재까지 이동한 간선의 합이 최대값이 될 가능성이 없을 경우

 c) 도착지점에 도착한 말을 움직이려하는 경우

 이 문제는 조금 억지로 풀었다는 느낌을 많이 받았다. 코드를 작성하는 내내 DFS를 중첩해서 사용하니까 연산시간에 대한

 압박감도 많이 받았고, 이렇게 구현하는게 맞는가하는 생각도 많이 들었다.

 처음 제출했을 때 연산 시간이 너무 커서 백트레킹 조건도 덕지덕지 붙여넣은 느낌이다.

다시 생각해봐야할 문제인 것 같다.

### [JAVA]

package baekjoon;

import java.util.\*;

import java.io.\*;

public class BOJ\_17825\_2 {

static int[] dice = new int[10];

static int[][] map = new int[33][33];

static int next = 0;

static int weight = 0;

static int max = 0;

public static class Node implements Comparable<Node>{

int v;

int w;

Node(int v, int w) {

this.v = v;

this.w = w;

}

@Override

public int compareTo(Node n) {

if(this.v < n.v) return 1;

else return -1;

}

}

public static void set() {

for(int i = 0; i < 20; i++) {

map[i][i + 1] = 2 \* (i + 1);

}

map[5][21] = 13; map[21][22] = 16; map[22][23] = 19; map[23][24] = 25;

map[15][27] = 28; map[27][26] = 27; map[26][25] = 26; map[25][24] = 25;

map[10][28] = 22; map[28][29] = 24; map[29][24] = 25;

map[24][30] = 30; map[30][31] = 35; map[31][20] = 40;

map[20][32] = 1;

}

public static void move(int cnt, int start) {

if(cnt == 0) return;

if(next == 32) {

weight = 1;

return;

}

ArrayList<Node> l = new ArrayList<>();

for(int i = 0; i < 33; i++) {

if(map[next][i] != 0) {

l.add(new Node(i, map[next][i]));

}

}

if(l.size() == 1) {

next = l.get(0).v;

weight = l.get(0).w;

}

else if(l.size() > 1) {

Collections.sort(l);

if(next == start) {

next = l.get(0).v;

weight = l.get(0).w;

}

else {

next = l.get(1).v;

weight = l.get(1).w;

}

}

move(cnt-1, start);

}

public static void select(int h, int[] ph, int x, int sum) {

if(x == 10) {

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

if(ph[i] == 32) sum--;

}

max = Math.max(max, sum);

return;

}

if(ph[h] == 32) return;

int[] horse = ph.clone();

next = horse[h];

move(dice[x], next);

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

if(next != 32 && horse[i] == next) return;

}

horse[h] = next;

sum += weight;

if(sum + (9 - x) \* 40 < max) return;

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

select(i, horse, x + 1, sum);

}

}

public static void main(String[] args) throws IOException{

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

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

for(int i = 0; i < 10; i++) {

dice[i] = Integer.parseInt(tk.nextToken());

}

set();

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

int[] horse = new int[4];

select(i, horse, 0, 0);

}

System.out.println(max);

}

}

## [**[백준] 17822번 - 원판 돌리기**](https://dheldh77.tistory.com/entry/%EB%B0%B1%EC%A4%80-17822%EB%B2%88-%EC%9B%90%ED%8C%90-%EB%8F%8C%EB%A6%AC%EA%B8%B0)

2020. 3. 18. 01:40

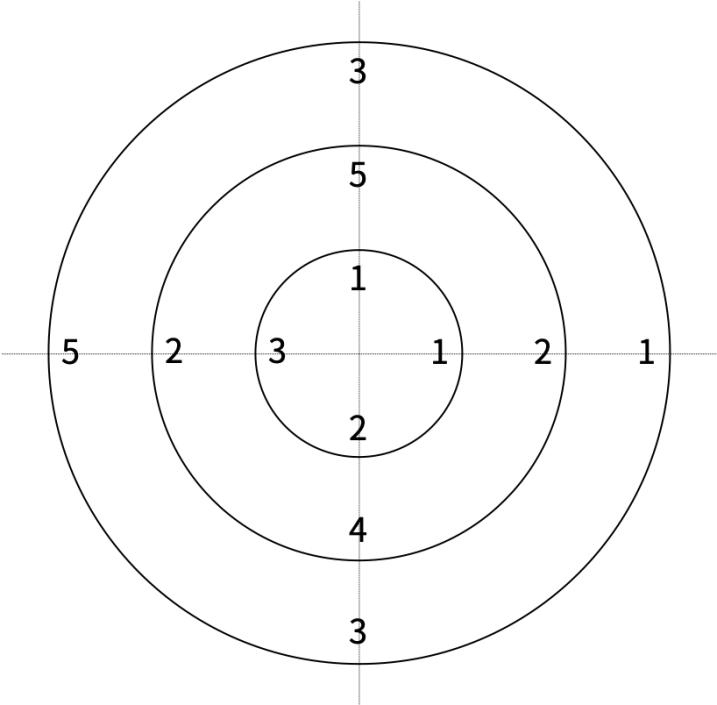
[17822](https://dheldh77.tistory.com/tag/17822), [BaekJoon](https://dheldh77.tistory.com/tag/BaekJoon), [DFS](https://dheldh77.tistory.com/tag/DFS), [SW역량테스트](https://dheldh77.tistory.com/tag/SW%EC%97%AD%EB%9F%89%ED%85%8C%EC%8A%A4%ED%8A%B8), [시뮬레이션](https://dheldh77.tistory.com/tag/%EC%8B%9C%EB%AE%AC%EB%A0%88%EC%9D%B4%EC%85%98), [알고리즘](https://dheldh77.tistory.com/tag/%EC%95%8C%EA%B3%A0%EB%A6%AC%EC%A6%98)

### 문제 >

반지름이 1, 2, ..., N인 원판이 크기가 작아지는 순으로 바닥에 놓여있고, 원판의 중심은 모두 같다. 원판의 반지름이 i이면, 그 원판을 i번째 원판이라고 한다. 각각의 원판에는 M개의 정수가 적혀있고, i번째 원판에 적힌 j번째 수의 위치는 (i, j)로 표현한다. 수의 위치는 다음을 만족한다.

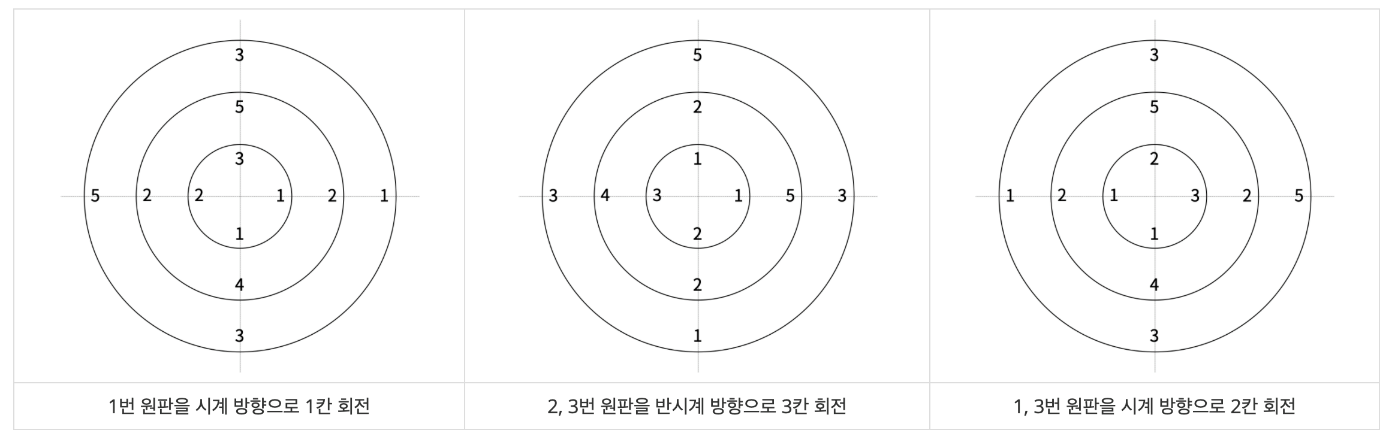
* (i, 1)은 (i, 2), (i, M)과 인접하다.
* (i, M)은 (i, M-1), (i, 1)과 인접하다.
* (i, j)는 (i, j-1), (i, j+1)과 인접하다. (2 ≤ j ≤ M-1)
* (1, j)는 (2, j)와 인접하다.
* (N, j)는 (N-1, j)와 인접하다.
* (i, j)는 (i-1, j), (i+1, j)와 인접하다. (2 ≤ i ≤ N-1)

아래 그림은 N = 3, M = 4인 경우이다.



원판의 회전은 독립적으로 이루어진다. 2번 원판을 회전했을 때, 나머지 원판은 회전하지 않는다. 원판을 회전시킬 때는 수의 위치를 기준으로 하며, 회전시킨 후의 수의 위치는 회전시키기 전과 일치해야 한다.

다음 그림은 원판을 회전시킨 예시이다.



원판을 아래와 같은 방법으로 총 T번 회전시키려고 한다. 원판의 회전 방법은 미리 정해져 있고, i번째 회전할때 사용하는 변수는 xi, di, ki이다.

1. 번호가 xi의 배수인 원판을 di방향으로 ki칸 회전시킨다. di가 0인 경우는 시계 방향, 1인 경우는 반시계 방향이다.
2. 원판에 수가 남아 있으면, 인접하면서 수가 같은 것을 모두 찾는다.
   1. 그러한 수가 있는 경우에는 원판에서 인접하면서 같은 수를 모두 지운다.
   2. 없는 경우에는 원판에 적힌 수의 평균을 구하고, 평균보다 큰 수에서 1을 빼고, 작은 수에는 1을 더한다.

원판을 T번 회전시킨 후 원판에 적힌 수의 합을 구해보자.

### 입력 >

첫째 줄에 N, M, T이 주어진다.

둘째 줄부터 N개의 줄에 원판에 적힌 수가 주어진다. i번째 줄의 j번째 수는 (i, j)에 적힌 수를 의미한다.

다음 T개의 줄에 xi, di, ki가 주어진다.

### 출력 >

원판을 T번 회전시킨 후 원판에 적힌 수의 합을 출력한다.

### 해결방법 >

시뮬레이션 문제로 원판이 돌아갈 때마다 인접한 노드가 같은 값을 가지는지 체크하고 삭제해준다.

두 가지 과정이 필요하다.

1) 원판돌리기

   양방향으로 데이터 입출력이 가능한 덱(Deque)를 사용해서 구현한다.

   시계 방향이라면 끝에 있는 원소를 뽑아 첫번째 원소에 넣는다.

   반시계 방향이라면 첫번째 원소를 뽑아 끝에 넣는다.

2) 인접한 노드가 같은 값을 가지는지 확인

   모든 원판을 2차원 배열로 생각하고 DFS(깊이우선탐색)을 사용해 인접한 노드가 같은 값을 가지는 모든 노드를 찾는다.

   단, 원판이므로 열을 탐색할 때, 열 끝에 도달했다면 반대열부터 계속해서 탐색해준다.

   2-1) 탐색한 모든 경로의 길이가 1일 때 즉, 인접한 수들 중 같은 수가 없다면 모든 노드의 평균값에서 각 노드의 값이 작다면 +1 크다면 -1

   2-2) 탐색한 경로 중 길이가 2이상이 있을 경우, 경로의 모든 노드들을 삭제해준다.(-1을 저장한다)

### [JAVA]

package baekjoon;

import java.io.\*;

import java.util.\*;

public class BOJ\_17822 {

static int n;

static int m;

static int[][] circle;

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

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

static ArrayList <Node> rlist = new ArrayList<>();

static int[][] visit;

static int depth = 0;

public static class Node{

int x;

int y;

Node(int x, int y){

this.x = x;

this.y = y;

}

}

public static void turn(int[] line, int d, int k) {

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

for(int i = 0; i < line.length; i++) {

q.addLast(line[i]);

}

if(d == 0) {

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

int tmp = q.pollLast();

q.addFirst(tmp);

}

}

if(d == 1) {

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

int tmp = q.pollFirst();

q.addLast(tmp);

}

}

int i = 0;

while(!q.isEmpty()) {

line[i] = q.poll();

i++;

}

}

public static void dfs(int x, int y, int v) {

depth++;

visit[x][y] = 1;

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

int nx = x + dx[i];

int ny = y + dy[i];

if(ny < 0) ny = m-1;

if(ny >= m) ny = 0;

if(nx >= 0 && ny >= 0 && nx < n && ny < m) {

if(visit[nx][ny] == 0 && circle[nx][ny] == v)

dfs(nx, ny, v);

}

}

}

public static void remove() {

visit = new int[n][m];

double sum = 0;

int flag = 0;

int zero = 0;

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

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

if(visit[i][j] == 0 && circle[i][j] != -1) {

depth = 0;

dfs(i, j, circle[i][j]);

if(depth == 1) {

visit[i][j] = 0;

sum += circle[i][j];

zero++;

}else {

flag = 1;

}

}

}

}

if(flag == 1) {

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

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

if(visit[i][j] == 1)

circle[i][j] = -1;

}

}

}

if(flag == 0) {

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

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

if(circle[i][j] > sum / zero)

circle[i][j]--;

else if (circle[i][j] != -1 && circle[i][j] < sum / zero)

circle[i][j]++;

}

}

}

}

public static void main(String[] args) throws IOException{

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

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

n = Integer.parseInt(tk.nextToken());

m = Integer.parseInt(tk.nextToken());

int t = Integer.parseInt(tk.nextToken());

circle = new int[n][m];

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

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

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

circle[i][j] = Integer.parseInt(tk.nextToken());

}

}

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

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

int x = Integer.parseInt(tk.nextToken());

int d = Integer.parseInt(tk.nextToken());

int k = Integer.parseInt(tk.nextToken());

int j = 1;

while(true) {

if(x \* j-1 >= n) break;

turn(circle[x \* j - 1], d, k);

j++;

}

remove();

}

int result = 0;

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

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

if(circle[i][j] != -1) {

result += circle[i][j];

}

}

}

System.out.println(result);

}

}

## [**[백준] 14500번 - 테트로미노**](https://dheldh77.tistory.com/entry/%EB%B0%B1%EC%A4%80-14500%EB%B2%88-%ED%85%8C%ED%8A%B8%EB%A1%9C%EB%AF%B8%EB%85%B8)

2020. 3. 16. 21:10

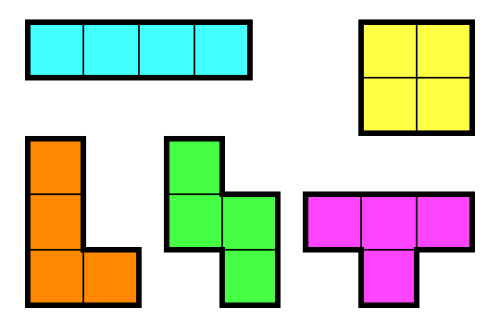
[14500](https://dheldh77.tistory.com/tag/14500), [DFS](https://dheldh77.tistory.com/tag/DFS), [SW역량테스트](https://dheldh77.tistory.com/tag/SW%EC%97%AD%EB%9F%89%ED%85%8C%EC%8A%A4%ED%8A%B8), [알고리즘](https://dheldh77.tistory.com/tag/%EC%95%8C%EA%B3%A0%EB%A6%AC%EC%A6%98)

### 문제 >

폴리오미노란 크기가 1×1인 정사각형을 여러 개 이어서 붙인 도형이며, 다음과 같은 조건을 만족해야 한다.

* 정사각형은 서로 겹치면 안 된다.
* 도형은 모두 연결되어 있어야 한다.
* 정사각형의 변끼리 연결되어 있어야 한다. 즉, 꼭짓점과 꼭짓점만 맞닿아 있으면 안 된다.

정사각형 4개를 이어 붙인 폴리오미노는 테트로미노라고 하며, 다음과 같은 5가지가 있다.



아름이는 크기가 N×M인 종이 위에 테트로미노 하나를 놓으려고 한다. 종이는 1×1 크기의 칸으로 나누어져 있으며, 각각의 칸에는 정수가 하나 쓰여 있다.

테트로미노 하나를 적절히 놓아서 테트로미노가 놓인 칸에 쓰여 있는 수들의 합을 최대로 하는 프로그램을 작성하시오.

테트로미노는 반드시 한 정사각형이 정확히 하나의 칸을 포함하도록 놓아야 하며, 회전이나 대칭을 시켜도 된다.

### 입력 >

첫째 줄에 종이의 세로 크기 N과 가로 크기 M이 주어진다. (4 ≤ N, M ≤ 500)

둘째 줄부터 N개의 줄에 종이에 쓰여 있는 수가 주어진다. i번째 줄의 j번째 수는 위에서부터 i번째 칸, 왼쪽에서부터 j번째 칸에 쓰여 있는 수이다. 입력으로 주어지는 수는 1,000을 넘지 않는 자연수이다.

### 출력 >

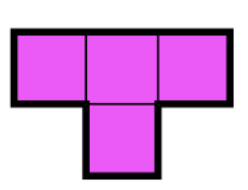
첫째 줄에 테트로미노가 놓인 칸에 쓰인 수들의 합의 최댓값을 출력한다.

### 해결방법 >

DFS(깊이우선탐색)으로 한 정점에서 시작으로 4개의 정점을 방문하면 위의 도형의 모든 모양을 만들 수 있다고 생각했다.

그렇다고 생각했는데 3번째 예제의 답이 틀렸다.

그래서 방문한 정점의 모양을 출력해봤더니

??????????

이게 나오질 않았다. 그래서 BFS 함수를 하나 더만들어서 해줬더니 입력값이 커져버렸을 때, 시간초과가 발생해버렸다...

그래서 DFS로 구할 수 있는 도형은 모두 찾되, 나오지 않는 하나의 도형의 모양을 만들어주는 함수를 따로하나 만들어주었다.

### [JAVA]

package baekjoon;

import java.util.\*;

import java.io.\*;

public class BOJ\_14500 {

static int n;

static int m;

static int[][] map;

static int[][] visit;

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

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

static int max;

static int[][] dir1 = {{0, 1}, {0, 2}, {1, 1}};

static int[][] dir2 = {{1, 0}, {2, 0}, {1, 1}};

static int[][] symm = {{1, 1}, {-1, 1}, {1, -1}, {-1, -1}};

public static void dfs(int x, int y, int depth, int sum) {

visit[x][y] = 1;

sum += map[x][y];

if(depth == 4) {

max = Math.max(sum, max);

visit[x][y] = 0;

return;

}

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

int nx = x + dx[i];

int ny = y + dy[i];

if(nx >= 0 && ny >= 0 && nx < n && ny < m && visit[nx][ny] == 0) {

dfs(nx, ny, depth+1, sum);

}

}

visit[x][y] = 0;

}

public static void shape(int x, int y) {

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

int sum = map[x][y];

for(int j = 0; j < 3; j++) {

int nx = x + symm[i][0] \* dir1[j][0];

int ny = y + symm[i][1] \* dir1[j][1];

if(nx >= 0 && ny >= 0 && nx < n && ny < m) {

sum += map[nx][ny];

}

else break;

}

max = Math.max(sum, max);

}

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

int sum = map[x][y];

for(int j = 0; j < 3; j++) {

int nx = x + symm[i][0] \* dir2[j][0];

int ny = y + symm[i][1] \* dir2[j][1];

if(nx >= 0 && ny >= 0 && nx < n && ny < m) {

sum += map[nx][ny];

}

else break;

}

max = Math.max(sum, max);

}

}

public static void main(String[] args) throws IOException{

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

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

n = Integer.parseInt(tk.nextToken());

m = Integer.parseInt(tk.nextToken());

map = new int[n][m];

visit = new int[n][m];

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

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

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

map[i][j] = Integer.parseInt(tk.nextToken());

}

}

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

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

dfs(i, j, 1, 0);

shape(i, j);

}

}

System.out.println(max);

}

}

## [**[백준] 16234번 - 인구 이동**](https://dheldh77.tistory.com/entry/%EB%B0%B1%EC%A4%80-16234%EB%B2%88-%EC%9D%B8%EA%B5%AC-%EC%9D%B4%EB%8F%99)

2020. 3. 16. 21:02

[16234](https://dheldh77.tistory.com/tag/16234), [BaekJoon](https://dheldh77.tistory.com/tag/BaekJoon), [BFS](https://dheldh77.tistory.com/tag/BFS), [DFS](https://dheldh77.tistory.com/tag/DFS), [SW역량테스트](https://dheldh77.tistory.com/tag/SW%EC%97%AD%EB%9F%89%ED%85%8C%EC%8A%A4%ED%8A%B8), [알고리즘](https://dheldh77.tistory.com/tag/%EC%95%8C%EA%B3%A0%EB%A6%AC%EC%A6%98)

### 문제 >

N×N크기의 땅이 있고, 땅은 1×1개의 칸으로 나누어져 있다. 각각의 땅에는 나라가 하나씩 존재하며, r행 c열에 있는 나라에는 A[r][c]명이 살고 있다. 인접한 나라 사이에는 국경선이 존재한다. 모든 나라는 1×1 크기이기 때문에, 모든 국경선은 정사각형 형태이다.

오늘부터 인구 이동이 시작되는 날이다.

인구 이동은 다음과 같이 진행되고, 더 이상 아래 방법에 의해 인구 이동이 없을 때까지 지속된다.

* 국경선을 공유하는 두 나라의 인구 차이가 L명 이상, R명 이하라면, 두 나라가 공유하는 국경선을 오늘 하루동안 연다.
* 위의 조건에 의해 열어야하는 국경선이 모두 열렸다면, 인구 이동을 시작한다.
* 국경선이 열려있어 인접한 칸만을 이용해 이동할 수 있으면, 그 나라를 오늘 하루 동안은 연합이라고 한다.
* 연합을 이루고 있는 각 칸의 인구수는 (연합의 인구수) / (연합을 이루고 있는 칸의 개수)가 된다. 편의상 소수점은 버린다.
* 연합을 해체하고, 모든 국경선을 닫는다.

각 나라의 인구수가 주어졌을 때, 인구 이동이 몇 번 발생하는지 구하는 프로그램을 작성하시오.

### 입력 >

첫째 줄에 N, L, R이 주어진다. (1 ≤ N ≤ 50, 1 ≤ L ≤ R ≤ 100)

둘째 줄부터 N개의 줄에 각 나라의 인구수가 주어진다. r행 c열에 주어지는 정수는 A[r][c]의 값이다. (0 ≤ A[r][c] ≤ 100)

인구 이동이 발생하는 횟수가 2,000번 보다 작거나 같은 입력만 주어진다.

### 출력 >

인구 이동이 몇 번 발생하는지 첫째 줄에 출력한다.

### 해결방법 >

DFS(깊이우선탐색) 또는 BFS(너비우선탐색) 알고리즘으로 해결할 수 있는 영역을 구하는 문제이다.

문제에서 국경선을 연다고 말했는데 국경선을 연다는 말은 한 영역이 될 수 있는 나라를 구하라는 말이다.

두 가지 과정이 필요하다.

1) 서로 국경선을 여는 나라를 찾는다(즉 한 영역이 될 수있는 나라들을 찾는다)

    BFS 또는 DFS를 사용해 한 영역이 나라를 찾는다. 인접한 국가 간의 인구 차가 L이상 R이하인 나라들이 한 영역이 된다.

2) 한 영역인 나라들끼리 인구이동을 한다.

    한 영역인 나라끼리 인구이동을 하는데,

    1)의 과정에서 총 인구 수 / 나라 수를 미리 구해서 ArrayList에 담아놓고 한 번에 값을 저장했다.

    지도마다 일일이 인구수를 찾고 나누고 하니까 시간초과가 발생해버렸다.

아래 코드는 각각 1)과정에서 DFS와 BFS로 구현한 코드이다.

### [JAVA]

package baekjoon;

import java.util.\*;

import java.io.\*;

public class BOJ\_16234\_DFS {

static int[][] map;

static int n;

static int l;

static int r;

static int flag = 0;

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

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

static int cnt = 0;

static int sum = 0;

public static void open(int x, int y, int[][] part, int num) {

part[x][y] = num;

cnt++;

sum += map[x][y];

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

int nx = x + dx[i];

int ny = y + dy[i];

if(nx >= 0 && ny >= 0 && nx < n && ny < n && part[nx][ny] == 0) {

int diff = Math.abs(map[nx][ny] - map[x][y]);

if(diff >= l && diff <= r) {

flag = 1;

open(nx, ny, part, num);

}

}

}

}

public static void move(int[][] part, ArrayList <Integer> p) {

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

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

map[i][j] = p.get(part[i][j] - 1);

}

}

}

public static void main(String[] args) throws IOException {

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

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

n = Integer.parseInt(tk.nextToken());

l = Integer.parseInt(tk.nextToken());

r = Integer.parseInt(tk.nextToken());

map = new int[n][n];

int time = 0;

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

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

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

map[i][j] = Integer.parseInt(tk.nextToken());

}

}

while(true) {

flag = 0;

int[][] part = new int[n][n];

ArrayList <Integer> population = new ArrayList<>();

int num = 1;

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

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

if(part[i][j] == 0) {

cnt = 0;

sum = 0;

open(i, j, part, num);

population.add(sum / cnt);

num++;

}

}

}

if(flag == 0) break;

move(part, population);

time++;

}

System.out.println(time);

}

}

package baekjoon;

import java.util.\*;

import java.io.\*;

public class BOJ\_16234\_BFS {

static int[][] map;

static int n;

static int l;

static int r;

static int flag = 0;

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

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

static ArrayList <Integer> list = new ArrayList<>();

public static class Node{

int x;

int y;

Node(int x, int y){

this.x = x;

this.y = y;

}

}

public static void open(int x, int y, int[][] part, int num) {

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

q.add(new Node(x, y));

part[x][y] = num;

int cnt = 1;

int sum = map[x][y];

while(!q.isEmpty()) {

Node node = q.poll();

x = node.x;

y = node.y;

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

int nx = x + dx[i];

int ny = y + dy[i];

if(nx >= 0 && ny >= 0 && nx < n && ny < n && part[nx][ny] == 0) {

int diff = Math.abs(map[x][y] - map[nx][ny]);

if(diff >= l && diff <= r) {

q.add(new Node(nx, ny));

part[nx][ny] = num;

sum += map[nx][ny];

cnt++;

flag = 1;

}

}

}

}

list.add(sum / cnt);

}

public static void move(int[][] part) {

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

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

map[i][j] = list.get(part[i][j] - 1);

}

}

list.clear();

}

public static void main(String[] args) throws IOException {

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

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

n = Integer.parseInt(tk.nextToken());

l = Integer.parseInt(tk.nextToken());

r = Integer.parseInt(tk.nextToken());

map = new int[n][n];

int time = 0;

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

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

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

map[i][j] = Integer.parseInt(tk.nextToken());

}

}

while(true) {

flag = 0;

int num = 1;

int[][] part = new int[n][n];

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

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

if(part[i][j] == 0) {

open(i, j, part, num);

num++;

}

}

}

if(flag == 0) break;

move(part);

time++;

}

System.out.println(time);

}

}

## [**[백준] 16236번 - 아기 상어**](https://dheldh77.tistory.com/entry/%EB%B0%B1%EC%A4%80-16236%EB%B2%88-%EC%95%84%EA%B8%B0-%EC%83%81%EC%96%B4)

2020. 3. 16. 01:07

[16236](https://dheldh77.tistory.com/tag/16236), [BaekJoon](https://dheldh77.tistory.com/tag/BaekJoon), [BFS](https://dheldh77.tistory.com/tag/BFS), [SW역량테스트](https://dheldh77.tistory.com/tag/SW%EC%97%AD%EB%9F%89%ED%85%8C%EC%8A%A4%ED%8A%B8), [시뮬레이션](https://dheldh77.tistory.com/tag/%EC%8B%9C%EB%AE%AC%EB%A0%88%EC%9D%B4%EC%85%98), [알고리즘](https://dheldh77.tistory.com/tag/%EC%95%8C%EA%B3%A0%EB%A6%AC%EC%A6%98)

### 문제 >

N×N 크기의 공간에 물고기 M마리와 아기 상어 1마리가 있다. 공간은 1×1 크기의 정사각형 칸으로 나누어져 있다. 한 칸에는 물고기가 최대 1마리 존재한다.

아기 상어와 물고기는 모두 크기를 가지고 있고, 이 크기는 자연수이다. 가장 처음에 아기 상어의 크기는 2이고, 아기 상어는 1초에 상하좌우로 인접한 한 칸씩 이동한다.

아기 상어는 자신의 크기보다 큰 물고기가 있는 칸은 지나갈 수 없고, 나머지 칸은 모두 지나갈 수 있다. 아기 상어는 자신의 크기보다 작은 물고기만 먹을 수 있다. 따라서, 크기가 같은 물고기는 먹을 수 없지만, 그 물고기가 있는 칸은 지나갈 수 있다.

아기 상어가 어디로 이동할지 결정하는 방법은 아래와 같다.

* 더 이상 먹을 수 있는 물고기가 공간에 없다면 아기 상어는 엄마 상어에게 도움을 요청한다.
* 먹을 수 있는 물고기가 1마리라면, 그 물고기를 먹으러 간다.
* 먹을 수 있는 물고기가 1마리보다 많다면, 거리가 가장 가까운 물고기를 먹으러 간다.
  + 거리는 아기 상어가 있는 칸에서 물고기가 있는 칸으로 이동할 때, 지나야하는 칸의 개수의 최솟값이다.
  + 거리가 가까운 물고기가 많다면, 가장 위에 있는 물고기, 그러한 물고기가 여러마리라면, 가장 왼쪽에 있는 물고기를 먹는다.

아기 상어의 이동은 1초 걸리고, 물고기를 먹는데 걸리는 시간은 없다고 가정한다. 즉, 아기 상어가 먹을 수 있는 물고기가 있는 칸으로 이동했다면, 이동과 동시에 물고기를 먹는다. 물고기를 먹으면, 그 칸은 빈 칸이 된다.

아기 상어는 자신의 크기와 같은 수의 물고기를 먹을 때 마다 크기가 1 증가한다. 예를 들어, 크기가 2인 아기 상어는 물고기를 2마리 먹으면 크기가 3이 된다.

공간의 상태가 주어졌을 때, 아기 상어가 몇 초 동안 엄마 상어에게 도움을 요청하지 않고 물고기를 잡아먹을 수 있는지 구하는 프로그램을 작성하시오.

### 입력 >

첫째 줄에 공간의 크기 N(2 ≤ N ≤ 20)이 주어진다.

둘째 줄부터 N개의 줄에 공간의 상태가 주어진다. 공간의 상태는 0, 1, 2, 3, 4, 5, 6, 9로 이루어져 있고, 아래와 같은 의미를 가진다.

* 0: 빈 칸
* 1, 2, 3, 4, 5, 6: 칸에 있는 물고기의 크기
* 9: 아기 상어의 위치

아기 상어는 공간에 한 마리 있다.

### 출력 >

첫째 줄에 아기 상어가 엄마 상어에게 도움을 요청하지 않고 물고기를 잡아먹을 수 있는 시간을 출력한다.

### 해결방법 >

시뮬레이션 문제로 한 마디로 설명하자면

자기 보다 작은 물고기를 먹으면서 계속 성장 아기 상어가 더 이상 먹을 것이 없을 때 이동한 거리를 구하는 문제이다.

문제는 두 가지 과정을 반복한다.

1) 찾는다.

  먹이를 찾는 과정이다. BFS(너비우선탐색) 알고리즘을 사용해 현재 아기 상어의 위치에서 인접한 노드를 너비 우선으로 먹이를 찾는다.

2) 먹는다.

  찾은 먹이 중에서 경로가 가장 짧으면서, 행의 가장 작으며, 열이 가장 작은 순으로 먹이 한마리를 먹는다.

  만약 현재까지 먹은 먹이 수가 아기 상어의 크기와 같아진다면 상어의 크기를 1증가시키고 현재까지 먹은 먹이 수를 0으로 초기화한다.

아래는 두 가지 코드가 있다. 첫 번째 코드는 먹이를 찾는 과정에서 가능한 모든 먹이를 찾는 방법으로 작성했는데,

다른 분들이 제출한 코드에 비해서 시간이 오래 걸려 두 번째 코드로 새로 작성했다.

두 코드에 차이점은 먹이를 찾는 과정 즉, BFS에서 모든 먹이를 찾느냐, 아니면 경로가 가장 짧은 먹이들만 찾느냐 차이다.

### [JAVA]

package baekjoon;

import java.util.\*;

import java.io.\*;

public class BOJ\_16236 {

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

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

static int n;

static int[][] map;

static int size = 2;

static int cnt = 0;

static int time = 0;

static ArrayList <fish> feed = new ArrayList<>();

static int x = 0;

static int y = 0;

public static class fish implements Comparable<fish>{

int x;

int y;

int depth;

fish(int x, int y, int depth){

this.x = x;

this.y = y;

this.depth = depth;

}

@Override

public int compareTo(fish f) {

if(this.depth == f.depth) {

if(this.x == f.x) {

if(this.y > f.y) return 1;

else return -1;

}

else if(this.x > f.x) return 1;

else return -1;

}

else if(this.depth > f.depth) return 1;

else return -1;

}

}

public static class Node{

int x;

int y;

Node(int x, int y){

this.x = x;

this.y = y;

}

}

public static void seek(int x, int y) {

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

int[][] visit = new int[n][n];

q.add(new Node(x, y));

visit[x][y] = 1;

while(!q.isEmpty()) {

Node node = q.poll();

int tmp\_x = node.x;

int tmp\_y = node.y;

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

int nx = tmp\_x + dx[i];

int ny = tmp\_y + dy[i];

if(nx >= 0 && ny >= 0 && nx < n && ny < n && map[nx][ny] <= size && visit[nx][ny] == 0) {

visit[nx][ny] = visit[tmp\_x][tmp\_y] + 1;

q.add(new Node(nx, ny));

if(map[nx][ny] < size && map[nx][ny] != 0) {

feed.add(new fish(nx, ny, visit[nx][ny] - 1));

}

}

}

}

}

public static void eat() {

Collections.sort(feed);

fish f = feed.get(0);

cnt++;

if(cnt == size) {

size++;

cnt = 0;

}

time += f.depth;

x = f.x;

y = f.y;

map[f.x][f.y] = 0;

feed.clear();

}

public static void main(String[] args) throws IOException{

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

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

n = Integer.parseInt(tk.nextToken());

map = new int[n][n];

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

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

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

map[i][j] = Integer.parseInt(tk.nextToken());

if(map[i][j] == 9) {

x = i;

y = j;

}

}

}

map[x][y] = 0;

while(true) {

seek(x, y);

if(feed.size() == 0) break;

eat();

}

System.out.println(time);

}

}

package baekjoon;

import java.util.\*;

import java.io.\*;

public class BOJ\_16236\_2 {

static int n;

static int[][] map;

static int x;

static int y;

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

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

static int size = 2;

static int cnt = 0;

static int time = 0;

static ArrayList <Fish> feed = new ArrayList<>();

public static class Fish implements Comparable<Fish>{

int x;

int y;

int d;

Fish(int x, int y, int d){

this.x = x;

this.y = y;

this.d = d;

}

@Override

public int compareTo(Fish f) {

if(this.d == f.d) {

if(this.x == f.x) {

if(this.y > f.y) return 1;

else return -1;

}

else if(this.x > f.x) return 1;

else return -1;

}

else if(this.d > f.d) return 1;

else return -1;

}

}

public static class Node{

int x;

int y;

Node(int x, int y){

this.x = x;

this.y = y;

}

}

public static void seek() {

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

int[][] visit = new int[n][n];

int flag = 0;

q.add(new Node(x, y));

visit[x][y] = 1;

while(!q.isEmpty()) {

Node node = q.poll();

int tmp\_x = node.x;

int tmp\_y = node.y;

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

int nx = tmp\_x + dx[i];

int ny = tmp\_y + dy[i];

if(nx >= 0 && ny >= 0 && nx < n && ny < n && visit[nx][ny] == 0 && map[nx][ny] <= size) {

if(map[nx][ny] != 0 && map[nx][ny] < size) {

if(feed.size() != 0 && feed.get(feed.size() - 1).d < visit[tmp\_x][tmp\_y]) {

flag = 1;

break;

}

feed.add(new Fish(nx, ny, visit[tmp\_x][tmp\_y]));

}

visit[nx][ny] = visit[tmp\_x][tmp\_y] + 1;

q.add(new Node(nx, ny));

}

}

if(flag == 1) break;

}

}

public static void eat() {

Collections.sort(feed);

cnt++;

if(cnt == size) {

size++;

cnt = 0;

}

time += feed.get(0).d;

x = feed.get(0).x;

y = feed.get(0).y;

map[x][y] = 0;

feed.clear();

}

public static void main(String[] args) throws IOException {

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

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

n = Integer.parseInt(tk.nextToken());

map = new int[n][n];

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

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

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

map[i][j] = Integer.parseInt(tk.nextToken());

if(map[i][j] == 9) {

x = i;

y = j;

}

}

}

map[x][y] = 0;

while(true) {

seek();

if(feed.size() == 0) break;

eat();

}

System.out.println(time);

}

}

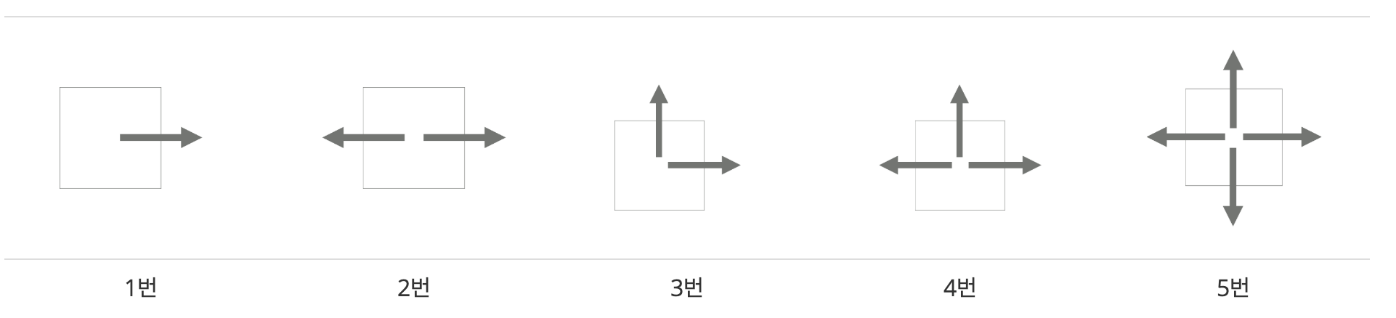
## [**[백준] 15683번 - 감시**](https://dheldh77.tistory.com/entry/%EB%B0%B1%EC%A4%80-15683%EB%B2%88-%EA%B0%90%EC%8B%9C)

2020. 3. 16. 00:21

[15683](https://dheldh77.tistory.com/tag/15683), [BaekJoon](https://dheldh77.tistory.com/tag/BaekJoon), [DFS](https://dheldh77.tistory.com/tag/DFS), [SW역량테스트](https://dheldh77.tistory.com/tag/SW%EC%97%AD%EB%9F%89%ED%85%8C%EC%8A%A4%ED%8A%B8), [알고리즘](https://dheldh77.tistory.com/tag/%EC%95%8C%EA%B3%A0%EB%A6%AC%EC%A6%98)

### 문제 >

스타트링크의 사무실은 1×1크기의 정사각형으로 나누어져 있는 N×M 크기의 직사각형으로 나타낼 수 있다. 사무실에는 총 K개의 CCTV가 설치되어져 있는데, CCTV는 5가지 종류가 있다. 각 CCTV가 감시할 수 있는 방법은 다음과 같다.



1번 CCTV는 한 쪽 방향만 감시할 수 있다. 2번과 3번은 두 방향을 감시할 수 있는데, 2번은 감시하는 방향이 서로 반대방향이어야 하고, 3번은 직각 방향이어야 한다. 4번은 세 방향, 5번은 네 방향을 감시할 수 있다.

CCTV는 감시할 수 있는 방향에 있는 칸 전체를 감시할 수 있다. 사무실에는 벽이 있는데, CCTV는 벽을 통과할 수 없다. CCTV가 감시할 수 없는 영역은 사각지대라고 한다.

CCTV는 회전시킬 수 있는데, 회전은 항상 90도 방향으로 해야 하며, 감시하려고 하는 방향이 가로 또는 세로 방향이어야 한다.



지도에서 0은 빈 칸, 6은 벽, 1~5는 CCTV의 번호이다. 위의 예시에서 1번의 방향에 따라 감시할 수 있는 영역을 '#'로 나타내면 아래와 같다.



CCTV는 벽을 통과할 수 없기 때문에, 1번이 → 방향을 감시하고 있을 때는 6의 오른쪽에 있는 벽을 감시할 수 없다.



위의 예시에서 감시할 수 있는 방향을 알아보면 아래와 같다.



CCTV는 CCTV를 통과할 수 있다. 아래 예시를 보자.



위와 같은 경우에 2의 방향이 ↕ 3의 방향이 ←와 ↓인 경우 감시받는 영역은 다음과 같다.



사무실의 크기와 상태, 그리고 CCTV의 정보가 주어졌을 때, CCTV의 방향을 적절히 정해서, 사각 지대의 최소 크기를 구하는 프로그램을 작성하시오.

### 입력 >

첫째 줄에 사무실의 세로 크기 N과 가로 크기 M이 주어진다. (1 ≤ N, M ≤ 8)

둘째 줄부터 N개의 줄에는 사무실 각 칸의 정보가 주어진다. 0은 빈 칸, 6은 벽, 1~5는 CCTV를 나타내고, 문제에서 설명한 CCTV의 종류이다.

CCTV의 최대 개수는 8개를 넘지 않는다.

### 출력 >

첫째 줄에 사각 지대의 최소 크기를 출력한다.

### 해결방법 >

DFS(깊이우선탐색) 알고리즘으로 각 CCTV의 방향을 설정했을 때 나올 수 있는 모든 경우의 수를 찾고 그 중 최소의 사각지대를 출력한다.

사무실에 대한 정보를 2차원 배열로 저장할 때, CCTV에 대한 좌표값과 번호를 저장한다.

저장된 CCTV를 하나씩 가능한 모든 경우에 따라 설치해본다.

각 CCTV의 번호에 따라 가능한 경우는 아래와 같다.

1번 CCTV > 동 / 서 / 남 / 북 (4 가지)

2번 CCTV > 동 - 서 / 남 - 북 (2 가지)

3번 CCTV > 북 - 동 / 동 - 남 / 남 - 서 / 서 - 북 (4 가지)

4번 CCTV > 서 - 북 - 동 / 북 - 동 - 남 / 동 - 남 - 서 / 남 - 서 - 북 (4 가지)

5번 CCTV > 동 - 서 - 남 - 북 (1 가지)

각 CCTV마다 한 가지 경우를 선택해 깊이우선탐색을 하고 마지막 CCTV를 선택했을 때 사각지대의 수를 구한다.

모든 가능한 경우를 탐색했을 때 사각지대의 최소값을 출력한다.

아래의 두 코드는 사실 메모리 사용량이나 시간에 차이는 없는데

단지 첫 번째 코드길이가 다른 분들이 제출한거에 비해 몇 배는 긴 거 같아서

다시 작성한 게 두 번째 코드....

되추적 알고리즘을 함께 사용하는 방법도 있는 거 같다.

이 문제는 다시 한 번 풀어봐야겠다

### [JAVA]

package baekjoon;

import java.util.\*;

public class BOJ\_15683 {

static int[][] map;

static ArrayList<Node> camera;

static int n;

static int m;

static int min;

public static class Node{

int x;

int y;

int k;

Node(int x, int y, int k){

this.x = x;

this.y = y;

this.k = k;

}

}

public static void see(int x, int y, int dx, int dy, int[][] v) {

int nx = x + dx;

int ny = y + dy;

while(nx >= 0 && ny >= 0 && nx < n && ny < m && map[nx][ny] != 6) {

v[nx][ny] = 1;

nx += dx;

ny += dy;

}

}

public static void dfs(Node node, int flag, int[][] v, int idx) {

int[][] visit = new int[n][m];

int x = node.x;

int y = node.y;

int k = node.k;

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

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

visit[i][j] = v[i][j];

}

}

visit[x][y] = 1;

if(k == 1) {

if(flag == 0) {

see(x, y, -1, 0, visit);

}

else if(flag == 1) {

see(x, y, 0, 1, visit);

}

else if(flag == 2) {

see(x, y, 1, 0, visit);

}

else {

see(x, y, 0, -1, visit);

}

}

else if(k == 2){

if(flag == 0) {

see(x, y, 0, -1, visit);

see(x, y, 0, 1, visit);

}

else {

see(x, y, -1, 0, visit);

see(x, y, 1, 0, visit);

}

}

else if(k == 3) {

if(flag == 0) {

see(x, y, -1, 0, visit);

see(x, y, 0, 1, visit);

}

else if(flag == 1) {

see(x, y, 0, 1, visit);

see(x, y, 1, 0, visit);

}

else if(flag == 2) {

see(x, y, 1, 0, visit);

see(x, y, 0, -1, visit);

}

else {

see(x, y, 0, -1, visit);

see(x, y, -1, 0, visit);

}

}

else if(k == 4) {

if(flag == 0) {

see(x, y, 0, -1, visit);

see(x, y, -1, 0, visit);

see(x, y, 0, 1, visit);

}

else if(flag == 1) {

see(x, y, -1, 0, visit);

see(x, y, 0, 1, visit);

see(x, y, 1, 0, visit);

}

else if(flag == 2) {

see(x, y, 0, 1, visit);

see(x, y, 1, 0, visit);

see(x, y, 0, -1, visit);

}

else {

see(x, y, 1, 0, visit);

see(x, y, 0, -1, visit);

see(x, y, -1, 0, visit);

}

}

else {

see(x, y, -1, 0, visit);

see(x, y, 0, 1, visit);

see(x, y, 1, 0, visit);

see(x, y, 0, -1, visit);

}

if(idx == camera.size()) {

int cnt = 0;

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

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

if(map[i][j] != 6 && visit[i][j] == 0) {

cnt++;

}

}

}

min = Math.min(cnt, min);

return;

}

node = camera.get(idx);

k = node.k;

if(k == 1 || k == 3 || k == 4) {

dfs(node, 0, visit, idx+1);

dfs(node, 1, visit, idx+1);

dfs(node, 2, visit, idx+1);

dfs(node, 3, visit, idx+1);

}

else if(k == 2) {

dfs(node, 0, visit, idx+1);

dfs(node, 1, visit, idx+1);

}

else {

dfs(node, 0, visit, idx+1);

}

}

public static void init() {

int[][] visit = new int[n][m];

Node node = camera.get(0);

int k = node.k;

if(k == 1 || k == 3 || k == 4) {

dfs(node, 0, visit, 1);

dfs(node, 1, visit, 1);

dfs(node, 2, visit, 1);

dfs(node, 3, visit, 1);

}

else if(k == 2) {

dfs(node, 0, visit, 1);

dfs(node, 1, visit, 1);

}

else {

dfs(node, 0, visit, 1);

}

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

n = sc.nextInt();

m = sc.nextInt();

map = new int[n][m];

min = 0;

camera = new ArrayList<>();

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

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

map[i][j] = sc.nextInt();

if(map[i][j] >= 1 && map[i][j] <= 5)

camera.add(new Node(i, j, map[i][j]));

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

min++;

}

}

if(camera.size() > 0) init();

System.out.println(min);

}

}

package baekjoon;

import java.util.\*;

public class BOJ\_15683\_2 {

static int n;

static int m;

static int[][] map;

static ArrayList <Camera> list = new ArrayList<>();

static int min = 0;

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

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

static int[][] one = {{0}, {1}, {2}, {3}};

static int[][] two = {{0, 2}, {1, 3}};

static int[][] three = {{0, 1}, {1, 2}, {2, 3}, {3, 0}};

static int[][] four = {{0, 1, 2}, {1, 2, 3}, {2, 3, 0}, {3, 0, 1}};

static int[][] five = {{0, 1, 2, 3}};

public static class Camera{

int x;

int y;

int num;

Camera(int x, int y, int num){

this.x = x;

this.y = y;

this.num = num;

}

}

public static void set(Camera c, int idx, int flag, int[][] num, int[][] v) {

int[][] visit = new int[n][m];

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

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

visit[i][j] = v[i][j];

}

}

visit[c.x][c.y] = 1;

for(int i = 0; i < num[flag].length; i++) {

int nx = c.x + dx[num[flag][i]];

int ny = c.y + dy[num[flag][i]];

while(nx >= 0 && ny >= 0 && nx < n && ny < m && map[nx][ny] != 6) {

visit[nx][ny] = 1;

nx += dx[num[flag][i]];

ny += dy[num[flag][i]];

}

}

classify(idx+1, visit);

}

public static void classify(int idx, int[][] v) {

if(idx == list.size()) {

int cnt = 0;

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

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

if(map[i][j] != 6 && v[i][j] == 0)

cnt++;

}

}

min = Math.min(min, cnt);

return;

}

Camera c = list.get(idx);

int[][] visit = new int[n][m];

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

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

visit[i][j] = v[i][j];

}

}

switch(c.num) {

case 1:

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

set(c, idx, i, one, visit);

}

break;

case 2:

for(int i = 0; i < 2; i++) {

set(c, idx, i, two, visit);

}

break;

case 3:

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

set(c, idx, i, three, visit);

}

break;

case 4:

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

set(c, idx, i, four, visit);

}

break;

case 5:

set(c, idx, 0, five, visit);

break;

}

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

n = sc.nextInt();

m = sc.nextInt();

map = new int[n][m];

int[][] visit = new int[n][m];

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

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

map[i][j] = sc.nextInt();

if(map[i][j] >= 1 && map[i][j] <= 5) {

list.add(new Camera(i, j, map[i][j]));

}

if(map[i][j] == 0) min++;

}

}

if(list.size() > 0) classify(0, visit);

System.out.println(min);

}

}

## [**[백준] 15686번 - 치킨 배달**](https://dheldh77.tistory.com/entry/%EB%B0%B1%EC%A4%80-15686%EB%B2%88-%EC%B9%98%ED%82%A8-%EB%B0%B0%EB%8B%AC)

2020. 3. 14. 16:07

[15686](https://dheldh77.tistory.com/tag/15686), [BaekJoon](https://dheldh77.tistory.com/tag/BaekJoon), [DFS](https://dheldh77.tistory.com/tag/DFS), [SW역량테스트](https://dheldh77.tistory.com/tag/SW%EC%97%AD%EB%9F%89%ED%85%8C%EC%8A%A4%ED%8A%B8), [알고리즘](https://dheldh77.tistory.com/tag/%EC%95%8C%EA%B3%A0%EB%A6%AC%EC%A6%98)

### 문제 >

크기가 N×N인 도시가 있다. 도시는 1×1크기의 칸으로 나누어져 있다. 도시의 각 칸은 빈 칸, 치킨집, 집 중 하나이다. 도시의 칸은 (r, c)와 같은 형태로 나타내고, r행 c열 또는 위에서부터 r번째 칸, 왼쪽에서부터 c번째 칸을 의미한다. r과 c는 1부터 시작한다.

이 도시에 사는 사람들은 치킨을 매우 좋아한다. 따라서, 사람들은 "**치킨 거리**"라는 말을 주로 사용한다. **치킨 거리**는 집과 가장 가까운 치킨집 사이의 거리이다. 즉, 치킨 거리는 집을 기준으로 정해지며, 각각의 집은 **치킨 거리**를 가지고 있다. **도시의 치킨 거리**는 모든 집의 **치킨 거리**의 합이다.

임의의 두 칸 (r1, c1)과 (r2, c2) 사이의 거리는 |r1-r2| + |c1-c2|로 구한다.

예를 들어, 아래와 같은 지도를 갖는 도시를 살펴보자.

0 2 0 1 0

1 0 1 0 0

0 0 0 0 0

0 0 0 1 1

0 0 0 1 2

0은 빈 칸, 1은 집, 2는 치킨집이다.

(2, 1)에 있는 집과 (1, 2)에 있는 치킨집과의 거리는 |2-1| + |1-2| = 2, (5, 5)에 있는 치킨집과의 거리는 |2-5| + |1-5| = 7이다. 따라서, (2, 1)에 있는 집의 치킨 거리는 2이다.

(5, 4)에 있는 집과 (1, 2)에 있는 치킨집과의 거리는 |5-1| + |4-2| = 6, (5, 5)에 있는 치킨집과의 거리는 |5-5| + |4-5| = 1이다. 따라서, (5, 4)에 있는 집의 치킨 거리는 1이다.

이 도시에 있는 치킨집은 모두 같은 프랜차이즈이다. 프렌차이즈 본사에서는 수익을 증가시키기 위해 일부 치킨집을 폐업시키려고 한다. 오랜 연구 끝에 이 도시에서 가장 수익을 많이 낼 수 있는  치킨집의 개수는 최대 M개라는 사실을 알아내었다.

도시에 있는 치킨집 중에서 최대 M개를 고르고, 나머지 치킨집은 모두 폐업시켜야 한다. 어떻게 고르면, **도시의 치킨 거리**가 가장 작게 될지 구하는 프로그램을 작성하시오.

### 입력 >

첫째 줄에 N(2 ≤ N ≤ 50)과 M(1 ≤ M ≤ 13)이 주어진다.

둘째 줄부터 N개의 줄에는 도시의 정보가 주어진다.

도시의 정보는 0, 1, 2로 이루어져 있고, 0은 빈 칸, 1은 집, 2는 치킨집을 의미한다. 집의 개수는 2N개를 넘지 않으며, 적어도 1개는 존재한다. 치킨집의 개수는 M보다 크거나 같고, 13보다 작거나 같다.

### 출력 >

첫째 줄에 폐업시키지 않을 치킨집을 최대 M개를 골랐을 때, 도시의 치킨 거리의 최솟값을 출력한다.

### 해결방법 >

그리디로 해결하려다가 최적값을 구하는 조건을 못찾았고 결국 완전탐색을 했다.

문제를 해결하기 위해 아래와 같은 과정을 거쳤다.

1) DFS(깊이우선탐색) m개의 치킨집을 선택하는 모든 경우를 찾는다.

2) 1)을 통해 m개의 치킨집을 선택했다면 각 집마다 치킨 거리를 구하고 총합을 구한다.

3) 각 경우의 총합 중 최솟값을 찾는다.

첫번째 코드는 2)번 과정에서 BFS로 치킨 거리를 구했더니

실행 시간이 다른 사람이 제출한 코드보다 5배가 넘었고 메모리 사용량을 말도 못한다;;;;;

잘생각해보니까 2)번 과정을 굳이 BFS로 할 필요없이 집마다 선택된 m개의 치킨집이랑 비교해가면서 치킨 거리를 찾으면 되는거였다.

수정한 코드는 두번째 코드이다. 왜 매번 굳이굳이 돌아서가는걸까...하...........ㅠㅠ

### [JAVA]

package baekjoon;

import java.util.\*;

import java.io.\*;

public class BOJ\_15686 {

static int cnt;

static int m;

static int n;

static int[][] a;

static ArrayList<Node> point;

static int min;

public static class Node{

int x;

int y;

Node(int x, int y){

this.x = x;

this.y = y;

}

}

public static void select(int idx, int flag, int[] s, int depth) {

if(idx >= cnt) return;

int[] store = new int[cnt];

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

store[i] = s[i];

}

store[idx] = flag;

if(depth == m) {

cal(store);

return;

}

select(idx+1, 0, store, depth);

select(idx+1, 1, store, depth+1);

}

public static void cal(int[] s) {

int[][] map = new int[n][n];

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

int[][] visit = new int[n][n];

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

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

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

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

map[i][j] = a[i][j];

}

}

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

if(s[i] == 0) {

map[point.get(i).x][point.get(i).y] = 0;

}

}

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

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

if(map[i][j] == 2) {

visit[i][j] = 1;

q.add(new Node(i, j));

}

}

}

int d = 0;

while(!q.isEmpty()) {

Node node = q.poll();

int x = node.x;

int y = node.y;

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

int nx = x + dx[i];

int ny = y + dy[i];

if(nx >= 0 && ny >= 0 && nx < n && ny < n && visit[nx][ny] == 0) {

if(a[nx][ny] == 1) {

d += visit[x][y];

}

visit[nx][ny] = visit[x][y] + 1;

q.add(new Node(nx, ny));

}

}

}

min = Math.min(d, min);

}

public static void main(String[] args) throws IOException{

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

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

n = Integer.parseInt(tk.nextToken());

m = Integer.parseInt(tk.nextToken());

a = new int[n][n];

point = new ArrayList<>();

cnt = 0;

min = 99999999;

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

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

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

a[i][j] = Integer.parseInt(tk.nextToken());

if(a[i][j] == 2) {

cnt++;

point.add(new Node(i, j));

}

}

}

int[] store = new int[cnt];

select(0, 0, store, 0);

select(0, 1, store, 1);

System.out.println(min);

}

}

package baekjoon;

import java.util.\*;

import java.io.\*;

public class BOJ\_15686\_2 {

static ArrayList <Node> people;

static ArrayList <Node> store;

static int n;

static int m;

static int cnt;

static int min;

public static class Node{

int x;

int y;

Node(int x, int y){

this.x = x;

this.y = y;

}

}

public static void dfs(int idx, int flag, int depth, int[] s) {

if(idx == cnt) return;

int[] b = new int[cnt];

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

b[i] = s[i];

}

b[idx] = flag;

if(depth == m) {

int total = 0;

for(int i = 0; i < people.size(); i++) {

int shortest = 9999999;

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

if(b[j] == 1) {

int d = Math.abs(people.get(i).x - store.get(j).x) + Math.abs(people.get(i).y - store.get(j).y);

shortest = Math.min(d, shortest);

}

}

total += shortest;

}

min = Math.min(total, min);

return;

}

dfs(idx+1, 0, depth, b);

dfs(idx+1, 1, depth+1, b);

}

public static void main(String[] args) throws IOException{

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

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

n = Integer.parseInt(tk.nextToken());

m = Integer.parseInt(tk.nextToken());

cnt = 0;

min = 99999999;

people = new ArrayList<>();

store = new ArrayList<>();

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

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

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

int tmp = Integer.parseInt(tk.nextToken());

if(tmp == 1) {

people.add(new Node(i, j));

}

else if(tmp == 2) {

store.add(new Node(i, j));

cnt++;

}

}

}

int[] a = new int[cnt];

dfs(0, 0, 0, a);

dfs(0, 1, 1, a);

System.out.println(min);

}

}

## [**[백준] 17779번 - 게리맨더링 2**](https://dheldh77.tistory.com/entry/%EB%B0%B1%EC%A4%80-17779%EB%B2%88-%EA%B2%8C%EB%A6%AC%EB%A7%A8%EB%8D%94%EB%A7%81-2)

2020. 3. 14. 15:58

[17779](https://dheldh77.tistory.com/tag/17779), [BaekJoon](https://dheldh77.tistory.com/tag/BaekJoon), [BFS](https://dheldh77.tistory.com/tag/BFS), [SW역량테스트](https://dheldh77.tistory.com/tag/SW%EC%97%AD%EB%9F%89%ED%85%8C%EC%8A%A4%ED%8A%B8), [알고리즘](https://dheldh77.tistory.com/tag/%EC%95%8C%EA%B3%A0%EB%A6%AC%EC%A6%98)

### 문제 >

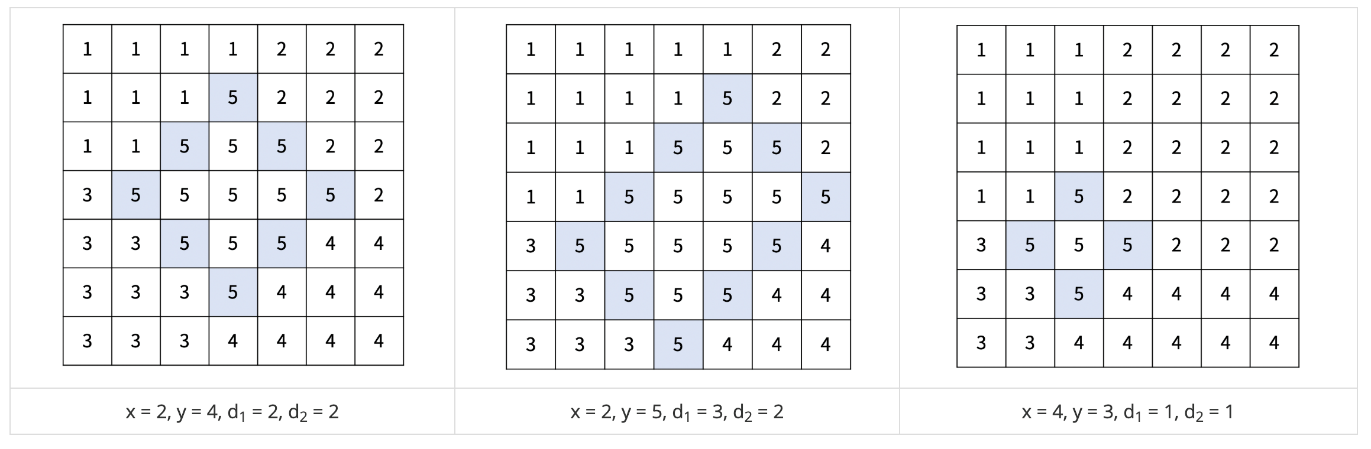
재현시의 시장 구재현은 지난 몇 년간 게리맨더링을 통해서 자신의 당에게 유리하게 선거구를 획정했다. 견제할 권력이 없어진 구재현은 권력을 매우 부당하게 행사했고, 심지어는 시의 이름도 재현시로 변경했다. 이번 선거에서는 최대한 공평하게 선거구를 획정하려고 한다.

재현시는 크기가 N×N인 격자로 나타낼 수 있다. 격자의 각 칸은 구역을 의미하고, r행 c열에 있는 구역은 (r, c)로 나타낼 수 있다. 구역을 다섯 개의 선거구로 나눠야 하고, 각 구역은 다섯 선거구 중 하나에 포함되어야 한다. 선거구는 구역을 적어도 하나 포함해야 하고, 한 선거구에 포함되어 있는 구역은 모두 연결되어 있어야 한다. 구역 A에서 인접한 구역을 통해서 구역 B로 갈 수 있을 때, 두 구역은 연결되어 있다고 한다. 중간에 통하는 인접한 구역은 0개 이상이어야 하고, 모두 같은 선거구에 포함된 구역이어야 한다.

선거구를 나누는 방법은 다음과 같다.

1. 기준점 (x, y)와 경계의 길이 d1, d2를 정한다. (d1, d2 ≥ 1, 1 ≤ x < x+d1+d2 ≤ N, 1 ≤ y-d1 < y < y+d2 ≤ N)
2. 다음 칸은 경계선이다.
   1. (x, y), (x+1, y-1), ..., (x+d1, y-d1)
   2. (x, y), (x+1, y+1), ..., (x+d2, y+d2)
   3. (x+d1, y-d1), (x+d1+1, y-d1+1), ... (x+d1+d2, y-d1+d2)
   4. (x+d2, y+d2), (x+d2+1, y+d2-1), ..., (x+d2+d1, y+d2-d1)
3. 경계선과 경계선의 안에 포함되어있는 5번 선거구이다.
4. 5번 선거구에 포함되지 않은 구역 (r, c)의 선거구 번호는 다음 기준을 따른다.
   1. 1번 선거구: 1 ≤ r < x+d1, 1 ≤ c ≤ y
   2. 2번 선거구: 1 ≤ r ≤ x+d2, y < c ≤ N
   3. 3번 선거구: x+d1 ≤ r ≤ N, 1 ≤ c < y-d1+d2
   4. 4번 선거구: x+d2 < r ≤ N, y-d1+d2 ≤ c ≤ N

아래는 크기가 7×7인 재현시를 다섯 개의 선거구로 나눈 방법의 예시이다.



구역 (r, c)의 인구는 A[r][c]이고, 선거구의 인구는 선거구에 포함된 구역의 인구를 모두 합한 값이다. 선거구를 나누는 방법 중에서, 인구가 가장 많은 선거구와 가장 적은 선거구의 인구 차이의 최솟값을 구해보자.

### 입력 >

첫째 줄에 재현시의 크기 N이 주어진다.

둘째 줄부터 N개의 줄에 N개의 정수가 주어진다. r행 c열의 정수는 A[r][c]를 의미한다.

### 출력 >

첫째 줄에 인구가 가장 많은 선거구와 가장 적은 선거구의 인구 차이의 최솟값을 출력한다.

### 해결방법 >

항상 많은 것을 배울 수 있게 해주는 친구가 SW역량테스트에서 풀었던 문제다.

그래서 좀더 흥미를 가지고 풀 수 있었던 문제였다.

문제는 다음과 같은 과정을 거친다.

1)  경계선을 만든다.

2) BFS로 경계선을 기준으로 선거구 번호를 매긴다.

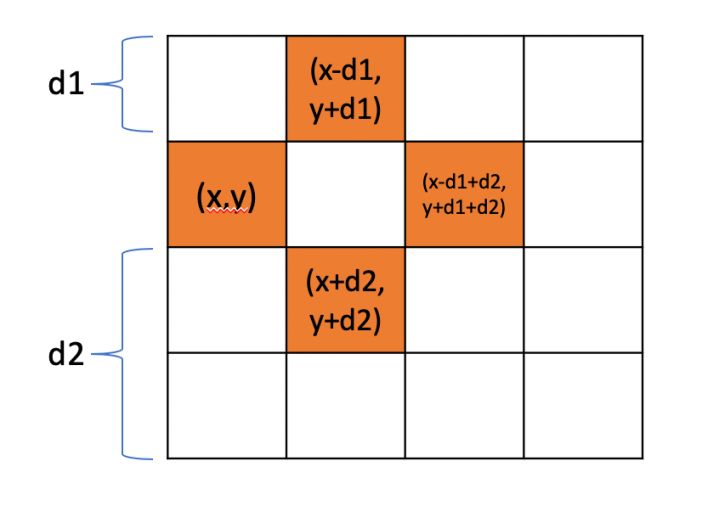
3) 인원수를 구한다.

n = 4일 때 선거구역을 나누는 예를 들어보자

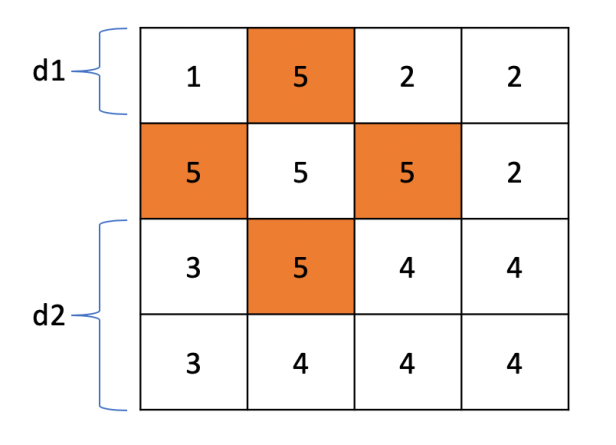
1) 경계선을 만든다.

경계선을 만드는 조건은 (d1, d2 ≥ 1, 1 ≤ y < y+d1+d2 ≤ N, 1 ≤ x-d1 < x < x+d2 ≤ N)이므로

조건을 만족하는 x, y, d1, d2를 경우를 찾는다. (편의상 문제의 보여준 x,y를 바꿔줌)



2) 경계선을 만들었다면 BFS로 각 선거구의 번호를 정점마다 매겨준다.



3) 선거구의 번호를 기준으로 입력받았던 이차원 배열과 비교해 각 선거구마다 인구수를 합해준다.

선거구를 나눌 때마다 가장 큰 인원과 작은 인원의 차를 구하고 이 중 최솟값을 출력한다.

### [JAVA]

package baekjoon;

import java.util.\*;

public class BOJ\_17779 {

static int[][] map;

static int n;

static int[][] a;

static int diff;

public static void cal() {

int[] vote = new int[5];

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

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

vote[map[i][j] - 1] += a[i][j];

}

}

int max = vote[0];

int min = vote[0];

for(int i = 1; i < 5; i++) {

min = Math.min(min, vote[i]);

max = Math.max(max, vote[i]);

}

diff = Math.min(diff, max - min);

}

public static class Node{

int x;

int y;

Node(int x, int y){

this.x = x;

this.y = y;

}

}

public static void bfs(int x, int y, int v) {

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

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

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

q.add(new Node(x, y));

map[x][y] = v;

while(!q.isEmpty()) {

Node node = q.poll();

x = node.x;

y = node.y;

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

int nx = x + dx[i];

int ny = y + dy[i];

if(nx >= 0 && ny >= 0 && nx < n && ny < n && map[nx][ny] == 0) {

map[nx][ny] = v;

q.add(new Node(nx, ny));

}

}

}

}

public static void part(int x, int y, int d1, int d2) {

draw(x, y, -1, 1, d1);

draw(x-d1, y+d1, 1, 1, d2);

draw(x-d1+d2, y+d1+d2, 1, -1, d1);

draw(x+d2, y+d2, -1, -1, d2);

for(int i = 0; i < x-d1; i++) {

map[i][y+d1] = 1;

}

for(int i = y+d1+d2+1; i < n; i++) {

map[x-d1+d2][i] = 2;

}

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

map[x][i] = 3;

}

for(int i = x+d2+1; i < n; i++) {

map[i][y+d2] = 4;

}

bfs(0, 0, 1);

bfs(0, n-1, 2);

bfs(n-1, 0, 3);

bfs(n-1, n-1, 4);

bfs(x, y, 5);

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

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

if(map[i][j] == 0) map[i][j] = 5;

}

}

}

public static void draw(int x, int y, int dx, int dy, int depth) {

if(depth == 0) return;

map[x][y] = 5;

draw(x+dx, y+dy, dx, dy, depth-1);

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

n = sc.nextInt();

a = new int[n][n];

diff = 9999999;

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

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

a[i][j] = sc.nextInt();

}

}

for(int i = 1; i < n; i++) {

for(int j = 0; j < n - 2; j++) {

for(int l = 1; l <= i; l++) {

for(int k = 1; k < n - i; k++) {

if(j + l + k < n) {

map = new int[n][n];

part(i, j, l, k);

cal();

}

}

}

}

}

System.out.println(diff);

}

}

## [**[백준] 14888번 - 연산자 끼워넣기**](https://dheldh77.tistory.com/entry/%EB%B0%B1%EC%A4%80-14888%EB%B2%88-%EC%97%B0%EC%82%B0%EC%9E%90-%EB%81%BC%EC%9B%8C%EB%84%A3%EA%B8%B0)

2020. 3. 14. 13:46

[14888](https://dheldh77.tistory.com/tag/14888), [BaekJoon](https://dheldh77.tistory.com/tag/BaekJoon), [DFS](https://dheldh77.tistory.com/tag/DFS), [SW역량테스트](https://dheldh77.tistory.com/tag/SW%EC%97%AD%EB%9F%89%ED%85%8C%EC%8A%A4%ED%8A%B8), [알고리즘](https://dheldh77.tistory.com/tag/%EC%95%8C%EA%B3%A0%EB%A6%AC%EC%A6%98)

### 문제 >

N개의 수로 이루어진 수열 A1, A2, ..., AN이 주어진다. 또, 수와 수 사이에 끼워넣을 수 있는 N-1개의 연산자가 주어진다. 연산자는 덧셈(+), 뺄셈(-), 곱셈(×), 나눗셈(÷)으로만 이루어져 있다.

우리는 수와 수 사이에 연산자를 하나씩 넣어서, 수식을 하나 만들 수 있다. 이때, 주어진 수의 순서를 바꾸면 안 된다.

예를 들어, 6개의 수로 이루어진 수열이 1, 2, 3, 4, 5, 6이고, 주어진 연산자가 덧셈(+) 2개, 뺄셈(-) 1개, 곱셈(×) 1개, 나눗셈(÷) 1개인 경우에는 총 60가지의 식을 만들 수 있다. 예를 들어, 아래와 같은 식을 만들 수 있다.

* 1+2+3-4×5÷6
* 1÷2+3+4-5×6
* 1+2÷3×4-5+6
* 1÷2×3-4+5+6

식의 계산은 연산자 우선 순위를 무시하고 앞에서부터 진행해야 한다. 또, 나눗셈은 정수 나눗셈으로 몫만 취한다. 음수를 양수로 나눌 때는 C++14의 기준을 따른다. 즉, 양수로 바꾼 뒤 몫을 취하고, 그 몫을 음수로 바꾼 것과 같다. 이에 따라서, 위의 식 4개의 결과를 계산해보면 아래와 같다.

* 1+2+3-4×5÷6 = 1
* 1÷2+3+4-5×6 = 12
* 1+2÷3×4-5+6 = 5
* 1÷2×3-4+5+6 = 7

N개의 수와 N-1개의 연산자가 주어졌을 때, 만들 수 있는 식의 결과가 최대인 것과 최소인 것을 구하는 프로그램을 작성하시오.

### 입력 >

첫째 줄에 수의 개수 N(2 ≤ N ≤ 11)가 주어진다. 둘째 줄에는 A1, A2, ..., AN이 주어진다. (1 ≤ Ai ≤ 100) 셋째 줄에는 합이 N-1인 4개의 정수가 주어지는데, 차례대로 덧셈(+)의 개수, 뺄셈(-)의 개수, 곱셈(×)의 개수, 나눗셈(÷)의 개수이다.

### 출력 >

첫째 줄에 만들 수 있는 식의 결과의 최댓값을, 둘째 줄에는 최솟값을 출력한다. 최댓값과 최솟값이 항상 -10억보다 크거나 같고, 10억보다 작거나 같은 결과가 나오는 입력만 주어진다. 또한, 앞에서부터 계산했을 때, 중간에 계산되는 식의 결과도 항상 -10억보다 크거나 같고, 10억보다 작거나 같다.

### 해결방법 >

DFS(깊이우선탐색) 알고리즘으로 가능한 모든 경우의 수를 찾는다.

각 연산자의 수를 배열로 담고 1번째 숫자부터 깊이 탐색을 시작한다.

한 정점에서 다음 정점으로 갈 때마다, 사용한 연산자의 수를 감소시키고,

선택한 연산자로 계산한 결과 값을 다음 노드로 전달하면서 탐색을 진행한다.

한 정점에서 다음 정점으로 갈 수 있는 경우는 최대 4가지('+', '-', '/', '\*' 연산자가 모두 있을  때)이다.

현재 존재하는 연산자의 종류의 수만큼 i+1번째 수로 방문한다.

연산자의 남은 수가 0일 때 결과값을 찾고, 이중 최댓값과최솟값을 출력한다.

### [JAVA]

package baekjoon;

import java.util.\*;

public class BOJ\_14888 {

static int[] a;

static int n;

static int max;

static int min;

public static void dfs(int idx, int sum, int[] op, int s) {

if(idx == n) {

max = Math.max(max, sum);

min = Math.min(min, sum);

return;

}

int[] cop = new int[4];

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

cop[i] = op[i];

}

if(s != -1) {

cop[s]--;

}

if(cop[0] > 0) {

dfs(idx+1, sum + a[idx], cop, 0);

}

if(cop[1] > 0) {

dfs(idx+1, sum - a[idx], cop, 1);

}

if(cop[2] > 0) {

dfs(idx+1, sum \* a[idx], cop, 2);

}

if(cop[3] > 0) {

dfs(idx+1, sum / a[idx], cop, 3);

}

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

n = sc.nextInt();

a = new int[n];

max = -1000000001;

min = 1000000001;

int[] op = new int[4];

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

a[i] = sc.nextInt();

}

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

op[i] = sc.nextInt();

}

dfs(1, a[0], op, -1);

System.out.println(max + "\n" + min);

}

}

## [**[백준] 14501번 - 퇴사**](https://dheldh77.tistory.com/entry/%EB%B0%B1%EC%A4%80-14501%EB%B2%88-%ED%87%B4%EC%82%AC)

2020. 3. 14. 13:37

[14501](https://dheldh77.tistory.com/tag/14501), [BOJ](https://dheldh77.tistory.com/tag/BOJ), [DFS](https://dheldh77.tistory.com/tag/DFS), [SW역량테스트](https://dheldh77.tistory.com/tag/SW%EC%97%AD%EB%9F%89%ED%85%8C%EC%8A%A4%ED%8A%B8), [알고리즘](https://dheldh77.tistory.com/tag/%EC%95%8C%EA%B3%A0%EB%A6%AC%EC%A6%98)

### 문제 >

상담원으로 일하고 있는 백준이는 퇴사를 하려고 한다.

오늘부터 N+1일째 되는 날 퇴사를 하기 위해서, 남은 N일 동안 최대한 많은 상담을 하려고 한다.

백준이는 비서에게 최대한 많은 상담을 잡으라고 부탁을 했고, 비서는 하루에 하나씩 서로 다른 사람의 상담을 잡아놓았다.

각각의 상담은 상담을 완료하는데 걸리는 기간 Ti와 상담을 했을 때 받을 수 있는 금액 Pi로 이루어져 있다.

N = 7인 경우에 다음과 같은 상담 일정표를 보자.



1일에 잡혀있는 상담은 총 3일이 걸리며, 상담했을 때 받을 수 있는 금액은 10이다. 5일에 잡혀있는 상담은 총 2일이 걸리며, 받을 수 있는 금액은 15이다.

상담을 하는데 필요한 기간은 1일보다 클 수 있기 때문에, 모든 상담을 할 수는 없다. 예를 들어서 1일에 상담을 하게 되면, 2일, 3일에 있는 상담은 할 수 없게 된다. 2일에 있는 상담을 하게 되면, 3, 4, 5, 6일에 잡혀있는 상담은 할 수 없다.

또한, N+1일째에는 회사에 없기 때문에, 6, 7일에 있는 상담을 할 수 없다.

퇴사 전에 할 수 있는 상담의 최대 이익은 1일, 4일, 5일에 있는 상담을 하는 것이며, 이때의 이익은 10+20+15=45이다.

상담을 적절히 했을 때, 백준이가 얻을 수 있는 최대 수익을 구하는 프로그램을 작성하시오.

### 입력 >

첫째 줄에 N (1 ≤ N ≤ 15)이 주어진다.

둘째 줄부터 N개의 줄에 Ti와 Pi가 공백으로 구분되어서 주어지며, 1일부터 N일까지 순서대로 주어진다. (1 ≤ Ti ≤ 5, 1 ≤ Pi ≤ 1,000)

### 출력 >

첫째 줄에 백준이가 얻을 수 있는 최대 이익을 출력한다.

### 해결방법 >

DFS(깊이우선탐색) 알고리즘으로 가능한 모든 경우의 수를 찾는다.

각 정점을 포함할 경우와 포함하지 않을 경우로 나누어 모든 경우의 수를 찾되, 조건이 붙는다.

각 날짜를 a[n] = {1, 2, 3, 4, ...., n} 라고 했을 때, a[i]를 선택하면 a[i] + t[i] + 1일부터 다시 일을 할 수 있다.

또한 t[i]가 남은 일 수보다 크다면 a[i]를 포함할 수 없다.

위의 조건을 만족하며 모든 경우의 수를 찾는다.

### [JAVA]

package baekjoon;

import java.util.\*;

public class BOJ\_14501 {

static int max;

static int[][] a;

static int n;

public static void dfs(int x, int sum) {

if(x >= n) {

max = Math.max(sum, max);

return;

}

if(a[x][0] + x <= n)

dfs(a[x][0] + x, sum + a[x][1]);

dfs(x + 1, sum);

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

n = sc.nextInt();

a = new int[n][2];

max = 0;

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

a[i][0] = sc.nextInt();

a[i][1] = sc.nextInt();

}

dfs(0, 0);

System.out.println(max);

}

}

## [**[백준] 15685번 - 드래곤 커브**](https://dheldh77.tistory.com/entry/%EB%B0%B1%EC%A4%80-15685%EB%B2%88-%EB%93%9C%EB%9E%98%EA%B3%A4-%EC%BB%A4%EB%B8%8C)

2020. 3. 13. 12:40

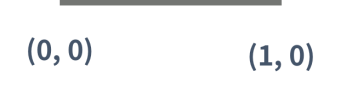
[15685](https://dheldh77.tistory.com/tag/15685), [BaekJoon](https://dheldh77.tistory.com/tag/BaekJoon), [SW역량테스트](https://dheldh77.tistory.com/tag/SW%EC%97%AD%EB%9F%89%ED%85%8C%EC%8A%A4%ED%8A%B8), [알고리즘](https://dheldh77.tistory.com/tag/%EC%95%8C%EA%B3%A0%EB%A6%AC%EC%A6%98)

### 문제 >

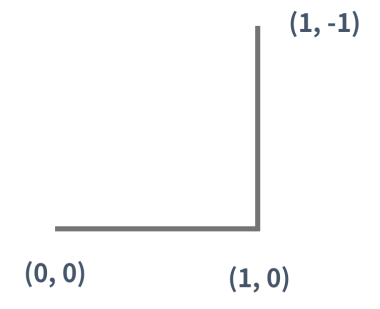
드래곤 커브는 다음과 같은 세 가지 속성으로 이루어져 있으며, 이차원 좌표 평면 위에서 정의된다. 좌표 평면의 x축은 → 방향, y축은 ↓ 방향이다.

1. 시작 점
2. 시작 방향
3. 세대

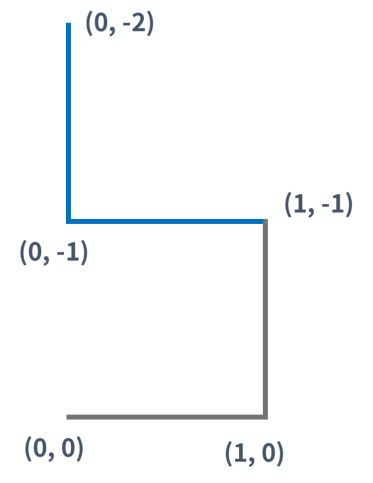
0세대 드래곤 커브는 아래 그림과 같은 길이가 1인 선분이다. 아래 그림은 (0, 0)에서 시작하고, 시작 방향은 오른쪽인 0세대 드래곤 커브이다.



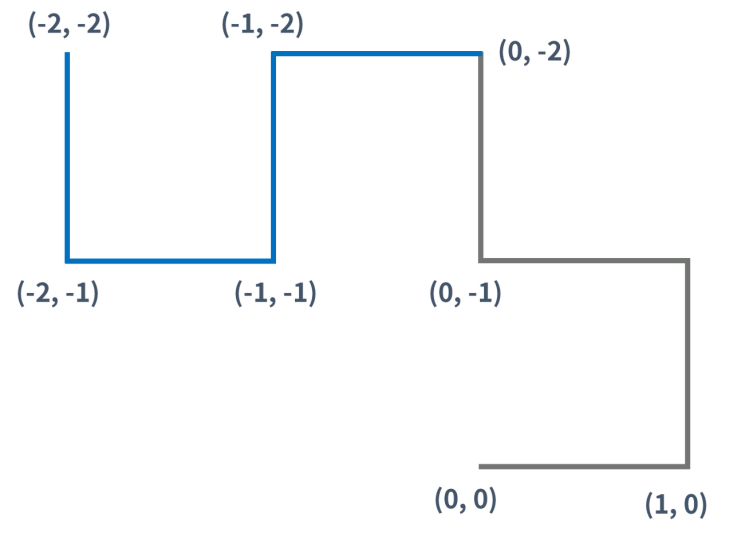
1세대 드래곤 커브는 0세대 드래곤 커브를 끝 점을 기준으로 시계 방향으로 90도 회전시킨 다음 0세대 드래곤 커브의 끝 점에 붙인 것이다. 끝 점이란 시작 점에서 선분을 타고 이동했을 때, 가장 먼 거리에 있는 점을 의미한다.



2세대 드래곤 커브도 1세대를 만든 방법을 이용해서 만들 수 있다. (파란색 선분은 새로 추가된 선분을 나타낸다)



3세대 드래곤 커브도 2세대 드래곤 커브를 이용해 만들 수 있다. 아래 그림은 3세대 드래곤 커브이다.



즉, K(K > 1)세대 드래곤 커브는 K-1세대 드래곤 커브를 끝 점을 기준으로 90도 시계 방향 회전 시킨 다음, 그것을 끝 점에 붙인 것이다.

크기가 100×100인 격자 위에 드래곤 커브가 N개 있다. 이때, 크기가 1×1인 정사각형의 네 꼭짓점이 모두 드래곤 커브의 일부인 정사각형의 개수를 구하는 프로그램을 작성하시오. 격자의 좌표는 (x, y)로 나타내며, 0 ≤ x ≤ 100, 0 ≤ y ≤ 100만 유효한 좌표이다.

### 입력 >

첫째 줄에 드래곤 커브의 개수 N(1 ≤ N ≤ 20)이 주어진다. 둘째 줄부터 N개의 줄에는 드래곤 커브의 정보가 주어진다. 드래곤 커브의 정보는 네 정수 x, y, d, g로 이루어져 있다. x와 y는 드래곤 커브의 시작 점, d는 시작 방향, g는 세대이다. (0 ≤ x, y ≤ 100, 0 ≤ d ≤ 3, 0 ≤ g ≤ 10)

입력으로 주어지는 드래곤 커브는 격자 밖으로 벗어나지 않는다. 드래곤 커브는 서로 겹칠 수 있다.

방향은 0, 1, 2, 3 중 하나이고, 다음을 의미한다.

* 0: x좌표가 증가하는 방향 (→)
* 1: y좌표가 감소하는 방향 (↑)
* 2: x좌표가 감소하는 방향 (←)
* 3: y좌표가 증가하는 방향 (↓)

### 출력 >

첫째 줄에 크기가 1×1인 정사각형의 네 꼭짓점이 모두 드래곤 커브의 일부인 것의 개수를 출력한다.

### 해결방법 >

두 가지 방법으로 해결해봤다.

**1) 첫 번째는 좌표를 모두 직접 찍어보는 방법이다.**

드래곤 커브는 한 세대를 거칠 때마다 끝점을 기준으로 대칭성을 보인다.

문제에서 나온 4세대를 기준으로 보면 각 좌표는 아래와 같다.(빨간색으로 표시된 좌표는 이전 세대에서 끝점이다)

좌표 : (0, 0) (1, 0) (1, -1) (0, -1) (0, -2) (-1, -2) (-1, -1) (-2, -1) (-2, -2)

              \   /    \    /   \   /     \   /     \    /     \   /     \    /    \     /

차이 :    (1, 0) (0, -1) (-1, 0) (0, -1)  (1, 0)   (0, 1)  (-1, 0)  (0, -1)

3세대의 끝점을 p[i]라고 하면 (0, -2)

p[i+1] = p[i] + (1, 0)

p[i-1] = p[i] - (0, -1)

좌표 : (0, 0) (1, 0) (1, -1) (0, -1) (0, -2) (-1, -2) (-1, -1) (-2, -1) (-2, -2)

              \   /    \    /   \   /     \   /     \    /     \   /     \    /    \     /

차이 :    (1, 0) (0, -1) (-1, 0) (0, -1)  (1, 0)   (0, 1)  (-1, 0)  (0, -1)

p[i+2] = p[i+1] + (0, 1)

p[i-2] = p[i-1] - (-1, 0)

좌표 : (0, 0) (1, 0) (1, -1) (0, -1) (0, -2) (-1, -2) (-1, -1) (-2, -1) (-2, -2)

              \   /    \    /   \   /     \   /     \    /     \   /     \    /    \     /

차이 :    (1, 0) (0, -1) (-1, 0) (0, -1)  (1, 0)   (0, 1)  (-1, 0)  (0, -1)

p[i+3] = p[i+2] + (-1, 0)

p[i-3] = p[i-2] - (0, -1)

좌표 : (0, 0) (1, 0) (1, -1) (0, -1) (0, -2) (-1, -2) (-1, -1) (-2, -1) (-2, -2)

              \   /    \    /   \   /     \   /     \    /     \   /     \    /    \     /

차이 :    (1, 0) (0, -1) (-1, 0) (0, -1)  (1, 0)   (0, 1)  (-1, 0)  (0, -1)

위와 같은 방식으로 대칭성을 찾아보면 아래와 같다.

   fp                gp

(0, -1)    ->   (1, 0)

(-1, 0)    ->   (0, 1)

(0, -1)    ->   (-1, 0)

(1, 0)     ->   (0, -1)

즉, p[n]을 이전 세대의 끝점이라고 한다면

p[n-i] = p[n-i-1] - fp를 통해

p[n+i] = p[n+i-1] + gp를 구할 수 있다.

한마리도 끝점을 기준으로 이전 좌표를 통해 다음 좌표를 구할 수 있다.

**2) 두 번째 좌표는 방향만으로 좌표를 모두 찾는 것이다.**

문제에 나온 예시를 들어보면

1세대일 때 방향은 0

2세대일 때 방향은 0, 1

3세대일 때 방향은 0, 1, 2, 1

4세대일 때 방향은 0, 1, 2, 1, 2, 3, 2, 1이다

i세대일 때 방향은 (i-1)세대일 때 방향에 모두 1씩 증가시켜준 값이다.

따라서 i세대까지의 모든 방향을 구하고 방향에 따라 좌표를 찾는다.

### [JAVA]

package baekjoon;

import java.util.\*;

public class s\_15685 {

static int[][] map;

static int cnt;

public static class Node{

int x;

int y;

Node(int x, int y){

this.x = x;

this.y = y;

}

}

public static Node go(int x, int y) {

Node next;

if(x == 1 && y == 0) next = new Node(0, -1);

else if(x == -1 && y == 0) next = new Node(0, 1);

else if(x == 0 && y == -1) next = new Node(-1, 0);

else next = new Node(1, 0);

return next;

}

public static void dragon(int x, int y, int[] d, int period) {

ArrayList<Node> a = new ArrayList<>();

ArrayList<Node> dp = new ArrayList<>();

a.add(new Node(x, y));

x = x + d[0];

y = y + d[1];

a.add(new Node(x, y));

dp.add(new Node(a.get(1).x - a.get(0).x, a.get(1).y - a.get(0).y));

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

int dp\_size = dp.size();

for(int j = 0; j < dp\_size; j++) {

int tmpx = dp.get(dp\_size - 1 - j).x;

int tmpy = dp.get(dp\_size - 1 - j).y;

Node node = go(tmpx, tmpy);

int nx = a.get(a.size()-1).x + node.x;

int ny = a.get(a.size()-1).y + node.y;

a.add(new Node(nx, ny));

dp.add(new Node(node.x, node.y));

}

}

for(int i = 0; i < a.size(); i++) {

if(a.get(i).x >= 0 && a.get(i).y >= 0 && a.get(i).x <= 100 && a.get(i).y <= 100){

map[a.get(i).x][a.get(i).y] = 1;

}

}

}

public static void count() {

for(int i = 0; i <= 100; i++) {

for(int j = 0; j <= 100; j++) {

if(map[i][j] == 1) {

if(i+1 <= 100 && j+1 <= 100) {

if(map[i+1][j] == 1 && map[i][j+1] == 1 && map[i+1][j+1] == 1) cnt++;

}

}

}

}

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

int[][] direct = {{1,0}, {0, -1}, {-1, 0}, {0, 1}};

map = new int[101][101];

int n = sc.nextInt();

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

int x = sc.nextInt();

int y = sc.nextInt();

int d = sc.nextInt();

int p = sc.nextInt();

dragon(x, y, direct[d], p);

}

count();

System.out.println(cnt);

}

}

package baekjoon;

import java.util.\*;

public class s\_15685\_2 {

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

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

static int[][] map;

static int cnt;

public static void go(int x, int y, int d, int p) {

ArrayList <Integer> a = new ArrayList<>();

map[x][y] = 1;

a.add(d);

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

int size = a.size();

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

d = a.get(size - 1 - j) + 1;

if(d > 3) d = 0;

a.add(d);

}

}

for(int i = 0; i < a.size(); i++) {

x += dx[a.get(i)];

y += dy[a.get(i)];

map[x][y] = 1;

}

}

public static void count() {

for(int i = 0; i < 100; i++) {

for(int j = 0; j < 100; j++) {

if(map[i][j] == 1 && map[i+1][j] == 1 && map[i][j+1] == 1 && map[i+1][j+1] == 1) cnt++;

}

}

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

int n = sc.nextInt();

map = new int[101][101];

cnt = 0;

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

int x = sc.nextInt();

int y = sc.nextInt();

int d = sc.nextInt();

int p = sc.nextInt();

go(x, y, d, p);

}

count();

System.out.println(cnt);

}

}

## [**[백준] 14889번 - 스타트와 링크**](https://dheldh77.tistory.com/entry/%EB%B0%B1%EC%A4%80-14889%EB%B2%88-%EC%8A%A4%ED%83%80%ED%8A%B8%EC%99%80-%EB%A7%81%ED%81%AC)

2020. 3. 13. 12:07

[14889](https://dheldh77.tistory.com/tag/14889), [BaekJoon](https://dheldh77.tistory.com/tag/BaekJoon), [DFS](https://dheldh77.tistory.com/tag/DFS), [SW역량테스트](https://dheldh77.tistory.com/tag/SW%EC%97%AD%EB%9F%89%ED%85%8C%EC%8A%A4%ED%8A%B8), [알고리즘](https://dheldh77.tistory.com/tag/%EC%95%8C%EA%B3%A0%EB%A6%AC%EC%A6%98)

### 문제 >

오늘은 스타트링크에 다니는 사람들이 모여서 축구를 해보려고 한다. 축구는 평일 오후에 하고 의무 참석도 아니다. 축구를 하기 위해 모인 사람은 총 N명이고 신기하게도 N은 짝수이다. 이제 N/2명으로 이루어진 스타트 팀과 링크 팀으로 사람들을 나눠야 한다.

BOJ를 운영하는 회사 답게 사람에게 번호를 1부터 N까지로 배정했고, 아래와 같은 능력치를 조사했다. 능력치 Sij는 i번 사람과 j번 사람이 같은 팀에 속했을 때, 팀에 더해지는 능력치이다. 팀의 능력치는 팀에 속한 모든 쌍의 능력치 Sij의 합이다. Sij는 Sji와 다를 수도 있으며, i번 사람과 j번 사람이 같은 팀에 속했을 때, 팀에 더해지는 능력치는 Sij와 Sji이다.

N=4이고, S가 아래와 같은 경우를 살펴보자.



예를 들어, 1, 2번이 스타트 팀, 3, 4번이 링크 팀에 속한 경우에 두 팀의 능력치는 아래와 같다.

* 스타트 팀: S12 + S21 = 1 + 4 = 5
* 링크 팀: S34 + S43 = 2 + 5 = 7

1, 3번이 스타트 팀, 2, 4번이 링크 팀에 속하면, 두 팀의 능력치는 아래와 같다.

* 스타트 팀: S13 + S31 = 2 + 7 = 9
* 링크 팀: S24 + S42 = 6 + 4 = 10

축구를 재미있게 하기 위해서 스타트 팀의 능력치와 링크 팀의 능력치의 차이를 최소로 하려고 한다. 위의 예제와 같은 경우에는 1, 4번이 스타트 팀, 2, 3번 팀이 링크 팀에 속하면 스타트 팀의 능력치는 6, 링크 팀의 능력치는 6이 되어서 차이가 0이 되고 이 값이 최소이다.

### 입력 >

첫째 줄에 N(4 ≤ N ≤ 20, N은 짝수)이 주어진다. 둘째 줄부터 N개의 줄에 S가 주어진다. 각 줄은 N개의 수로 이루어져 있고, i번 줄의 j번째 수는 Sij 이다. Sii는 항상 0이고, 나머지 Sij는 1보다 크거나 같고, 100보다 작거나 같은 정수이다.

### 출력 >

첫째 줄에 스타트 팀과 링크 팀의 능력치의 차이의 최솟값을 출력한다.

### 해결방법 >

DFS(깊이우선탐색)으로 문제 해결.

1) 깊이 우선 탐색으로 N명을  2개의 팀으로 나누는 경우를 모두 찾는다.

2) 1)과정에서 2팀으로 나눈 경우를 찾을 때마다, 양 팀의 능력치 차이를 구한다.

3) 능력치 차이가 이전에 구했던 능력치 차이보다 작을 경우 저장한다.

아래 구현 코드에서 첫번째 소스코드는 한 정점을 시작으로 깊이우선탐색을 진행하는 코드이다.

시간 초과가 되었다. 프루닝조건을 아무리 찾아도 생각나지가 않았다.

그래서 각 원소를 포함하는 경우, 포함하지 않는 경우로 깊이우선탐색을 했다.(두번째 소스코드)

이전에 구현했던 코드가 민망할 정도로 그냥 통과해버렸다.

하지만, 실행시간이 조금 걸리는 것 같아서 두가지 조건을 추가했다.

1. 최소값이 0이 구해졌다면 탐색을 종료한다.

(절대값의 최소값은 0이므로)

2. 현재 탐색하고 있는 인덱스에서 포함하고있는 수와 남은 원소의 수의 합이 총 원소의 개수가 반이 되지 않으면 탐색을 하지않는다.

(각 팀은 반으로 나누어져야하므로)

### [JAVA]

import java.util.\*;

public class Main {

static int n;

static int[] visit;

static int[][] a;

static int min;

public static void dfs(int s, int x, int depth) {

visit[x] = 1;

if(depth == n/2 - 1) {

score();

visit[x] = 0;

return;

}

for(int i = s; i < n; i++) {

if(visit[i] == 0 && (depth + (n-i)) >= n/2) {

dfs(s, i, depth + 1);

}

}

visit[x] = 0;

}

public static void score() {

int one = 0;

int two = 0;

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

if(visit[i] == 1) {

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

if(visit[j] == 1) {

one += a[i][j];

}

}

}

if(visit[i] == 0) {

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

if(visit[j] == 0) {

two += a[i][j];

}

}

}

}

min = Math.min(min, Math.abs(one-two));

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

n = sc.nextInt();

a = new int[n][n];

visit = new int[n];

min = 20000;

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

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

a[i][j] = sc.nextInt();

}

}

for(int i = 0; i < n/2; i++) {

dfs(i, i, 0);

}

System.out.println(min);

}

}

package baekjoon;

import java.util.\*;

public class s\_14889 {

static int n;

static int[] visit;

static int[][] a;

static int min;

public static void dfs(int idx, int flag, int depth, int[] visit) {

int[] arr = new int[n];

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

arr[i] = visit[i];

}

arr[idx] = flag;

if(depth == n/2) {

score(arr);

return;

}

if(n-idx+depth < n/2 || idx == n - 1 || min == 0) return;

dfs(idx+1, 0, depth, arr);

dfs(idx+1, 1, depth+1, arr);

}

public static void score(int[] visit) {

int one = 0;

int two = 0;

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

if(visit[i] == 1) {

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

if(visit[j] == 1) {

one += a[i][j];

}

}

}

if(visit[i] == 0) {

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

if(visit[j] == 0) {

two += a[i][j];

}

}

}

}

min = Math.min(min, Math.abs(one-two));

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

n = sc.nextInt();

a = new int[n][n];

visit = new int[n];

min = 20000;

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

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

a[i][j] = sc.nextInt();

}

}

dfs(0, 0, 0, visit);

dfs(0, 1, 1, visit);

System.out.println(min);

}

}

## [**[백준] 14503번 - 로봇 청소기**](https://dheldh77.tistory.com/entry/%EB%B0%B1%EC%A4%80-14503%EB%B2%88-%EB%A1%9C%EB%B4%87-%EC%B2%AD%EC%86%8C%EA%B8%B0)

2020. 3. 13. 11:21

[14503](https://dheldh77.tistory.com/tag/14503), [BaekJoon](https://dheldh77.tistory.com/tag/BaekJoon), [DFS](https://dheldh77.tistory.com/tag/DFS), [SW역량테스트](https://dheldh77.tistory.com/tag/SW%EC%97%AD%EB%9F%89%ED%85%8C%EC%8A%A4%ED%8A%B8), [알고리즘](https://dheldh77.tistory.com/tag/%EC%95%8C%EA%B3%A0%EB%A6%AC%EC%A6%98)

### 문제 >

로봇 청소기가 주어졌을 때, 청소하는 영역의 개수를 구하는 프로그램을 작성하시오.

로봇 청소기가 있는 장소는 N×M 크기의 직사각형으로 나타낼 수 있으며, 1×1크기의 정사각형 칸으로 나누어져 있다. 각각의 칸은 벽 또는 빈 칸이다. 청소기는 바라보는 방향이 있으며, 이 방향은 동, 서, 남, 북중 하나이다. 지도의 각 칸은 (r, c)로 나타낼 수 있고, r은 북쪽으로부터 떨어진 칸의 개수, c는 서쪽으로 부터 떨어진 칸의 개수이다.

로봇 청소기는 다음과 같이 작동한다.

1. 현재 위치를 청소한다.
2. 현재 위치에서 현재 방향을 기준으로 왼쪽방향부터 차례대로 탐색을 진행한다.
   1. 왼쪽 방향에 아직 청소하지 않은 공간이 존재한다면, 그 방향으로 회전한 다음 한 칸을 전진하고 1번부터 진행한다.
   2. 왼쪽 방향에 청소할 공간이 없다면, 그 방향으로 회전하고 2번으로 돌아간다.
   3. 네 방향 모두 청소가 이미 되어있거나 벽인 경우에는, 바라보는 방향을 유지한 채로 한 칸 후진을 하고 2번으로 돌아간다.
   4. 네 방향 모두 청소가 이미 되어있거나 벽이면서, 뒤쪽 방향이 벽이라 후진도 할 수 없는 경우에는 작동을 멈춘다.

로봇 청소기는 이미 청소되어있는 칸을 또 청소하지 않으며, 벽을 통과할 수 없다.

### 입력 >

첫째 줄에 세로 크기 N과 가로 크기 M이 주어진다. (3 ≤ N, M ≤ 50)

둘째 줄에 로봇 청소기가 있는 칸의 좌표 (r, c)와 바라보는 방향 d가 주어진다. d가 0인 경우에는 북쪽을, 1인 경우에는 동쪽을, 2인 경우에는 남쪽을, 3인 경우에는 서쪽을 바라보고 있는 것이다.

셋째 줄부터 N개의 줄에 장소의 상태가 북쪽부터 남쪽 순서대로, 각 줄은 서쪽부터 동쪽 순서대로 주어진다. 빈 칸은 0, 벽은 1로 주어진다. 지도의 첫 행, 마지막 행, 첫 열, 마지막 열에 있는 모든 칸은 벽이다.

로봇 청소기가 있는 칸의 상태는 항상 빈 칸이다.

### 출력 >

로봇 청소기가 청소하는 칸의 개수를 출력한다.

### 해결방법 >

DFS(깊이우선탐색)로 문제 해결

한 정점에서 인접한 정점을 탐색할 때, 현재 바라보고 있는 방향을 중심으로 왼쪽부터 탐색을 시작한다.

그렇게 되려면 탐색 시 방향 정보도 함께 가지고 있어야한다.

또한 인접한 네 개의 노드를 탐색 시에 조건에 맞는 노드를 찾을 시에는 나머지 인접한 노드는 더 이상 탐색하지 않는다.

인접한 네 개의 노드가 모두 벽이거나(원소값이 1이거나), 모두 방문했을 경우,

현재의 방향정보를 가지고 뒤로 한 칸 감소시킨다.

처음에 방향 정보를 잘못 입력해서 시간이 오래걸렸다.

방향은 왼쪽부터 탐색하므로, 북(-1, 0) -> 서(0, -1) -> 남(1, 0) -> 동(0, 1) -> 북(-1, 0) -> ...이며

입력의 주어진 방향은 0(북), 1(동), 2(남), 3(서)로 탐색방향과 반대인점을 유의한다.

### [JAVA]

package baekjoon;

import java.util.\*;

public class s\_14503 {

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

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

static int[][] map;

static int[][] visit;

static int n;

static int m;

static int cnt;

public static void dfs(int x, int y, int d) {

if(visit[x][y] == 0) {

cnt++;

visit[x][y] = 1;

}

int flag = 0;

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

// 현재 방향을 기준으로 왼쪽부터 4번 탐색

d = d - 1;

if(d < 0) d = 3;

int nx = x + dx[d];

int ny = y + dy[d];

if(nx >= 0 && ny >= 0 && nx < n && ny < m) {

if(map[nx][ny] == 0 && visit[nx][ny] == 0) {

flag = 1;

dfs(nx, ny, d);

break;

}

}

}

if(flag == 0) {

// 현재 방향을 유지한 채 한 칸 후진

int nx = x - dx[d];

int ny = y - dy[d];

if(nx >= 0 && ny >= 0 && nx < n && ny < m) {

if(map[nx][ny] == 0)

dfs(nx, ny, d);

}

}

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

n = sc.nextInt();

m = sc.nextInt();

map = new int[n][m];

cnt = 0;

visit = new int[n][m];

int x = sc.nextInt();

int y = sc.nextInt();

int d = sc.nextInt();

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

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

map[i][j] = sc.nextInt();

}

}

dfs(x, y, d);

System.out.println(cnt);

}

}

## [**[백준] 14890번 - 경사로**](https://dheldh77.tistory.com/entry/%EB%B0%B1%EC%A4%80-17144%EB%B2%88-%EB%AF%B8%EC%84%B8%EB%A8%BC%EC%A7%80-%EC%95%88%EB%85%95)

2020. 3. 12. 11:45

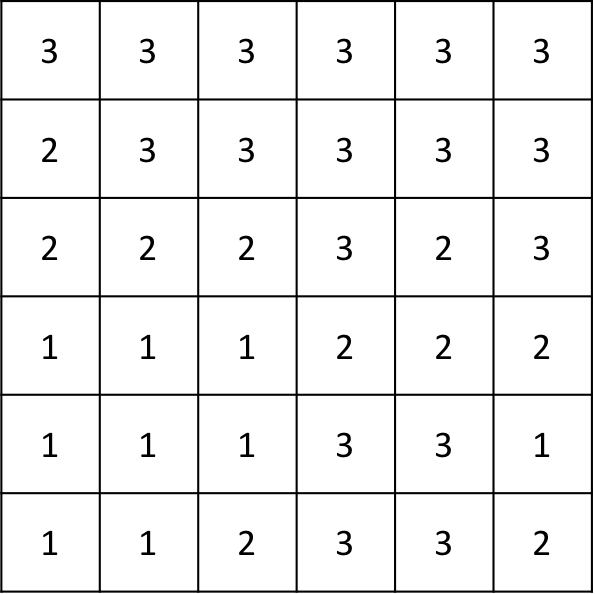
[14890](https://dheldh77.tistory.com/tag/14890), [BOJ](https://dheldh77.tistory.com/tag/BOJ), [DFS](https://dheldh77.tistory.com/tag/DFS), [SW역량테스트](https://dheldh77.tistory.com/tag/SW%EC%97%AD%EB%9F%89%ED%85%8C%EC%8A%A4%ED%8A%B8), [알고리즘](https://dheldh77.tistory.com/tag/%EC%95%8C%EA%B3%A0%EB%A6%AC%EC%A6%98)

### 문제 >

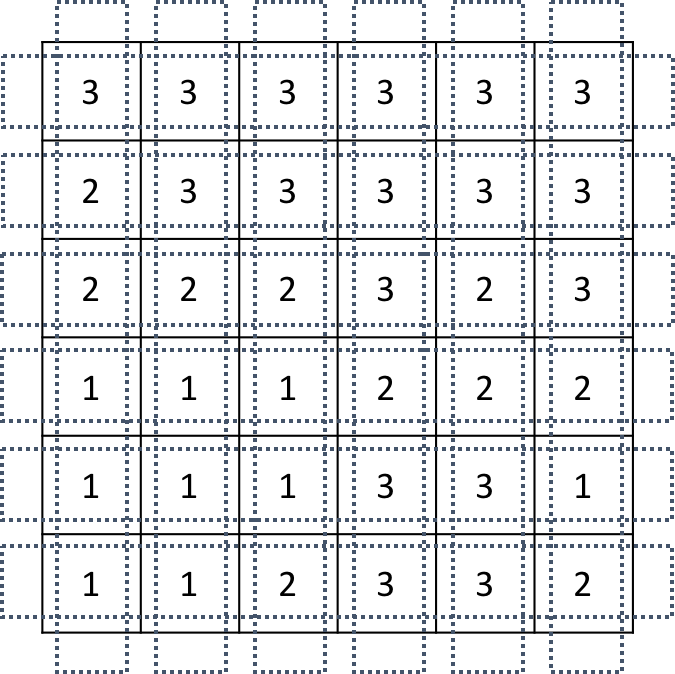
크기가 N×N인 지도가 있다. 지도의 각 칸에는 그 곳의 높이가 적혀져 있다.

오늘은 이 지도에서 지나갈 수 있는 길이 몇 개 있는지 알아보려고 한다. 길이란 한 행 또는 한 열 전부를 나타내며, 한쪽 끝에서 다른쪽 끝까지 지나가는 것이다.

다음과 같은 N=6인 경우 지도를 살펴보자.



이때, 길은 총 2N개가 있으며, 아래와 같다.



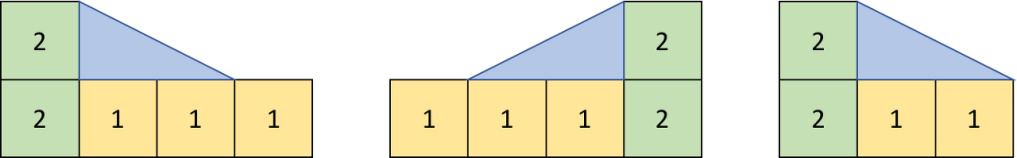
길을 지나갈 수 있으려면 길에 속한 모든 칸의 높이가 모두 같아야 한다. 또는, 경사로를 놓아서 지나갈 수 있는 길을 만들 수 있다. 경사로는 높이가 항상 1이며, 길이는 L이다. 또, 개수는 매우 많아 부족할 일이 없다. 경사로는 낮은 칸과 높은 칸을 연결하며, 아래와 같은 조건을 만족해야한다.

* 경사로는 낮은 칸에 놓으며, L개의 연속된 칸에 경사로의 바닥이 모두 접해야 한다.
* 낮은 칸과 높은 칸의 높이 차이는 1이어야 한다.
* 경사로를 놓을 낮은 칸의 높이는 모두 같아야 하고, L개의 칸이 연속되어 있어야 한다.

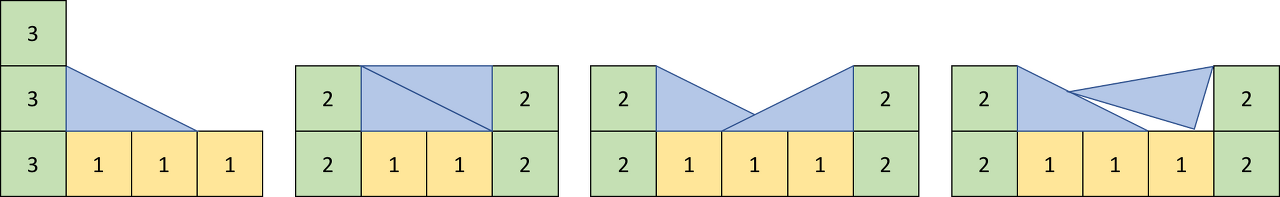
아래와 같은 경우에는 경사로를 놓을 수 없다.

* 경사로를 놓은 곳에 또 경사로를 놓는 경우
* 낮은 칸과 높은 칸의 높이 차이가 1이 아닌 경우
* 낮은 지점의 칸의 높이가 모두 같지 않거나, L개가 연속되지 않은 경우
* 경사로를 놓다가 범위를 벗어나는 경우

L = 2인 경우에 경사로를 놓을 수 있는 경우를 그림으로 나타내면 아래와 같다.

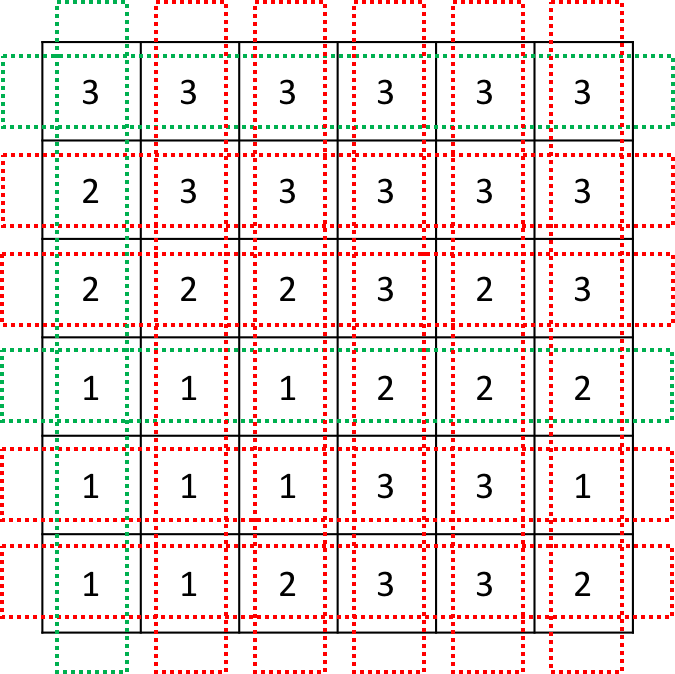


경사로를 놓을 수 없는 경우는 아래와 같다.



위의 그림의 가장 왼쪽부터 1번, 2번, 3번, 4번 예제라고 했을 때, 1번은 높이 차이가 1이 아니라서, 2번은 경사로를 바닥과 접하게 놓지 않아서, 3번은 겹쳐서 놓아서, 4번은 기울이게 놓아서 불가능한 경우이다.

가장 위에 주어진 그림 예의 경우에 지나갈 수 있는 길은 초록색으로, 지나갈 수 없는 길은 빨간색으로 표시되어 있으며, 아래와 같다. 경사로의 길이 L = 2이다.



지도가 주어졌을 때, 지나갈 수 있는 길의 개수를 구하는 프로그램을 작성하시오.

### 입력 >

첫째 줄에 N (2 ≤ N ≤ 100)과 L (1 ≤ L ≤ N)이 주어진다. 둘째 줄부터 N개의 줄에 지도가 주어진다. 각 칸의 높이는 10보다 작거나 같은 자연수이다.

### 출력 >

첫째 줄에 지나갈 수 있는 길의 개수를 출력한다.

### 해결방법 >

행과 열을 각각 탐색해서 경사로를 설치해야하는 경우, 설치해야한다면 조건을 만족하는지 검사한다.

경사로를 설치해야하는 경우는 행을 탐색한다고 예를 들었을 때,

a[i][j] 와 a[i][j-1]의 원소값이 1이 차이나는 경우이다. 만약 1이상 차이가 난다면 경사로를 설치할 수 없는 경우이다.

경사로를 설치해야한다면,

a[i][j-1]이 더 작다면 i행 j-1열부터 {0, -1} 방향으로

a[i][j]가 더 작다면 i행 j열부터 {0, 1} 방향으로 DFS(깊이 우선 탐색)를 한다.

너비우선탐색을 할 때, 탐색해야할 노드가 존재하는 경우는

탐색의 시작한 노드와 원소값이 같고, 배열을 벗어나지 않으면서, 방문하지 않은 경우이다.

여기서 방문했다는 의미는 경사로를 이미 설치했다는 의미이다.

깊이 우선 탐색이 끝났을 때,

경로의 길이가 L과 같다면 경사로를 설치할 수 있는 경우이며,

그렇지 않은 경우 경사로를 설치할 수 없는 경우이다.

위의 과정을 열 단위로도 반복하고,

가능한 총 경로 수(2N)에서 경사로를 설치해야하지만 설치하지 못한 경우의 수를 뺀다.

### [JAVA]

package baekjoon;

import java.util.\*;

public class BOJ\_14890 {

static int n;

static int l;

static int[][] visit;

static int[][] a;

static int flag;

public static void dfs(int x, int y, int[] d, int depth, int v) {

if(visit[x][y] == 1)

return;

visit[x][y] = 1;

if(depth == l) {

flag = 0;

return;

}

int nx = x + d[0];

int ny = y + d[1];

if(nx >= 0 && ny >= 0 && nx < n && ny < n && visit[nx][ny] == 0 && a[nx][ny] == v) {

dfs(nx, ny, d, depth + 1, v);

}

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

n = sc.nextInt();

l = sc.nextInt();

a = new int[n][n];

visit = new int[n][n];

int[][] direct = {{0, -1}, {0, 1}, {-1, 0}, {1, 0}};

int cnt = 2 \* n;

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

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

a[i][j] = sc.nextInt();

}

}

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

for(int j = 1; j < n; j++) {

flag = 1;

if(a[i][j] == a[i][j-1]) {

flag = 0;

}

else if(a[i][j] - 1 == a[i][j-1]) {

dfs(i, j-1, direct[0], 1, a[i][j-1]);

}

else if(a[i][j] == a[i][j-1] - 1) {

dfs(i, j, direct[1], 1, a[i][j]);

}

if(flag == 1) {

cnt--;

break;

}

}

}

visit = new int[n][n];

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

for(int j = 1; j < n; j++) {

flag = 1;

if(a[j][i] == a[j-1][i]) {

flag = 0;

}

else if(a[j][i] - 1 == a[j-1][i]) {

dfs(j-1, i, direct[2], 1, a[j-1][i]);

}

else if(a[j][i] == a[j-1][i] - 1){

dfs(j, i, direct[3], 1, a[j][i]);

}

if(flag == 1) {

cnt--;

break;

}

}

}

System.out.println(cnt);

}

}

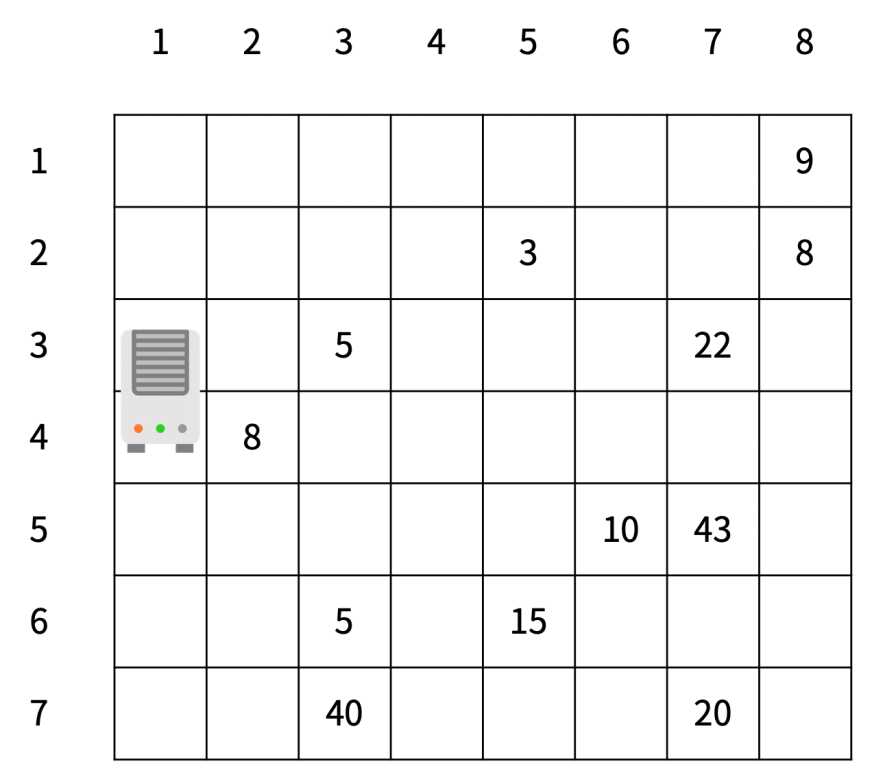
## [**[백준] 17144번 - 미세먼지 안녕!**](https://dheldh77.tistory.com/entry/%EB%B0%B1%EC%A4%80-14502%EB%B2%88-%EC%97%B0%EA%B5%AC%EC%86%8C-1)

2020. 3. 12. 11:27

[17144](https://dheldh77.tistory.com/tag/17144), [BOJ](https://dheldh77.tistory.com/tag/BOJ), [SW역량테스트](https://dheldh77.tistory.com/tag/SW%EC%97%AD%EB%9F%89%ED%85%8C%EC%8A%A4%ED%8A%B8), [백준](https://dheldh77.tistory.com/tag/%EB%B0%B1%EC%A4%80), [알고리즘](https://dheldh77.tistory.com/tag/%EC%95%8C%EA%B3%A0%EB%A6%AC%EC%A6%98)

### 문제 >

미세먼지를 제거하기 위해 구사과는 공기청정기를 설치하려고 한다. 공기청정기의 성능을 테스트하기 위해 구사과는 집을 크기가 R×C인 격자판으로 나타냈고, 1×1 크기의 칸으로 나눴다. 구사과는 뛰어난 코딩 실력을 이용해 각 칸 (r, c)에 있는 미세먼지의 양을 실시간으로 모니터링하는 시스템을 개발했다. (r, c)는 r행 c열을 의미한다.

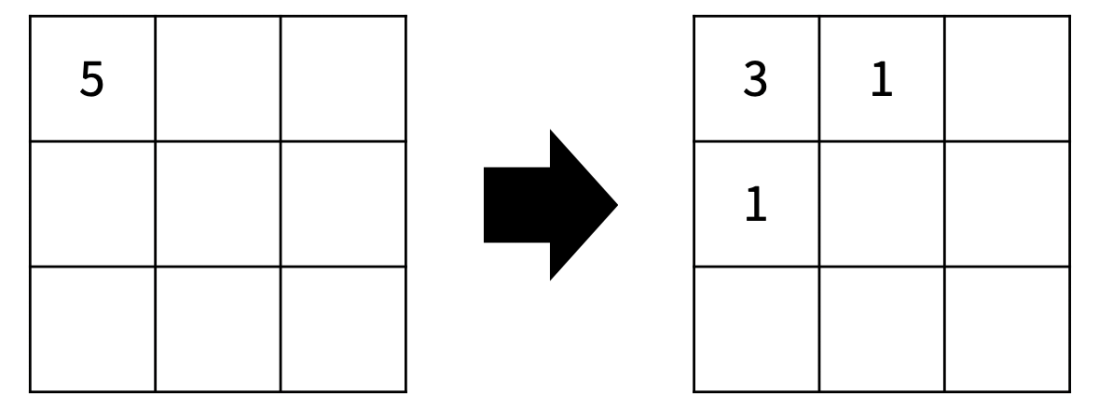


공기청정기는 항상 1번 열에 설치되어 있고, 크기는 두 행을 차지한다. 공기청정기가 설치되어 있지 않은 칸에는 미세먼지가 있고, (r, c)에 있는 미세먼지의 양은 Ar,c이다.

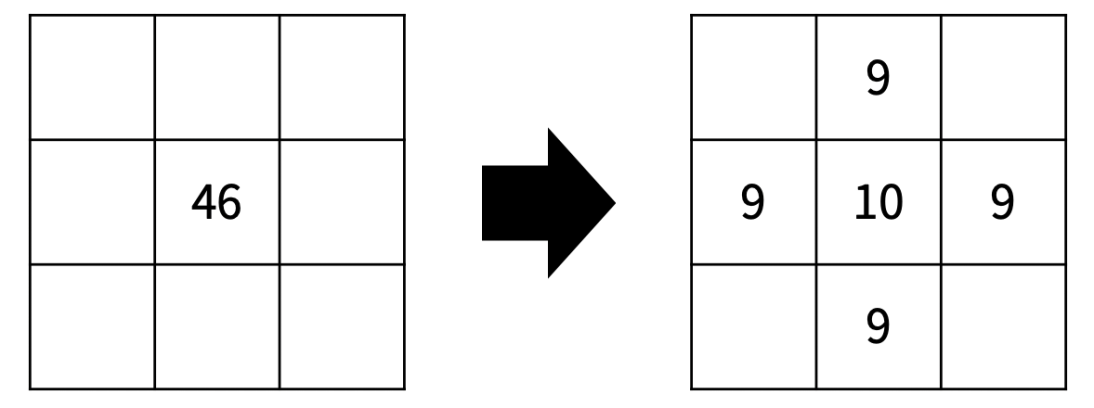
1초 동안 아래 적힌 일이 순서대로 일어난다.

1. 미세먼지가 확산된다. 확산은 미세먼지가 있는 모든 칸에서 동시에 일어난다.
   * (r, c)에 있는 미세먼지는 인접한 네 방향으로 확산된다.
   * 인접한 방향에 공기청정기가 있거나, 칸이 없으면 그 방향으로는 확산이 일어나지 않는다.
   * 확산되는 양은 Ar,c/5이고 소수점은 버린다.
   * (r, c)에 남은 미세먼지의 양은 Ar,c - (Ar,c/5)×(확산된 방향의 개수) 이다.
2. 공기청정기가 작동한다.
   * 공기청정기에서는 바람이 나온다.
   * 위쪽 공기청정기의 바람은 반시계방향으로 순환하고, 아래쪽 공기청정기의 바람은 시계방향으로 순환한다.
   * 바람이 불면 미세먼지가 바람의 방향대로 모두 한 칸씩 이동한다.
   * 공기청정기에서 부는 바람은 미세먼지가 없는 바람이고, 공기청정기로 들어간 미세먼지는 모두 정화된다.

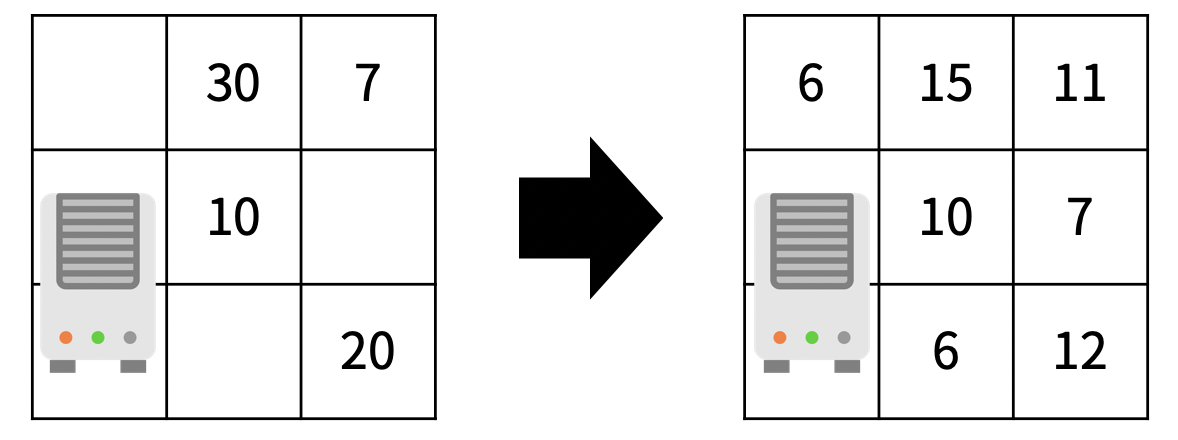
다음은 확산의 예시이다.



왼쪽과 오른쪽에 칸이 없기 때문에, 두 방향으로만 확산이 일어났다.

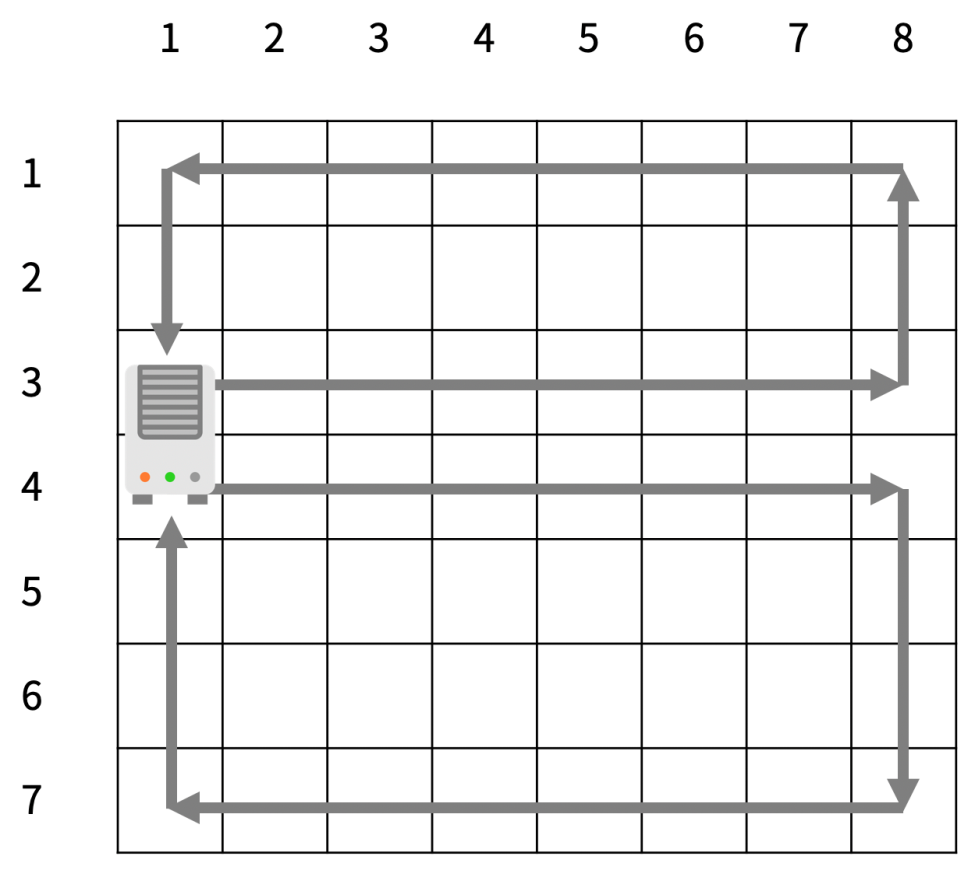


인접한 네 방향으로 모두 확산이 일어난다.



공기청정기가 있는 칸으로는 확산이 일어나지 않는다.

공기청정기의 바람은 다음과 같은 방향으로 순환한다.



방의 정보가 주어졌을 때, T초가 지난 후 구사과의 방에 남아있는 미세먼지의 양을 구해보자.

### 입력 >

첫째 줄에 R, C, T (6 ≤ R, C ≤ 50, 1 ≤ T ≤ 1,000) 가 주어진다.

둘째 줄부터 R개의 줄에 Ar,c (-1 ≤ Ar,c ≤ 1,000)가 주어진다. 공기청정기가 설치된 곳은 Ar,c가 -1이고, 나머지 값은 미세먼지의 양이다. -1은 2번 위아래로 붙어져 있고, 가장 윗 행, 아랫 행과 두 칸이상 떨어져 있다.

### 출력 >

첫째 줄에 T초가 지난 후 구사과 방에 남아있는 미세먼지의 양을 출력한다.

### 해결방법 >

입력을 2차원 배열 a[r][c]에 저장하고, 1 cycle 마다 두 단계를 거친다. 여기서 1cycle은 1초를 의미한다.

**1) 미세먼지 확산**

1 이상인 원소가 상하좌우로 인접한 4개의 원소로 확산된다.

인접한 4개의 방향의 원소에 a[i][j] / 5를 저장하고, 확산된 방의 개수만큼 a[i][j]에  a[i][j]/5를 뺀다.

이 때, 확산되는 원소의 값이 -1이면 확산이 되지 않는다.

**단, 확산을 할 때, 미리 배열의 값을 복사한다. 이유는 1cycle 안에 확산에 의한 확산이 일어나서는 안되기 때문이다.**

**2) 공기청정**

공기청정기은 0열에 존재하고 길이는 세로 2이다.

따라서 두 파트로 나누어 원소를 하나씩 밀어준다.

예를 들어 공기 청정기가 (x, 0), (x+1, 0) 인덱스에 존재한다면

위쪽 공기 청정기는 0에서 x행까지 0에서 c열까지 순환을하고

아래쪽 공기 청정기는 x+1에서 r행까지 0에서 c열까지 순환을 한다.

위쪽, 아래쪽 각 각 원소값을 하나씩 밀어준다.

### [JAVA]

package baekjoon;

import java.util.\*;

public class BOJ\_17144 {

static int r;

static int c;

public static void spread(int[][] a) {

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

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

int[][] b = new int[r][c];

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

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

b[i][j] = a[i][j];

}

}

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

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

if(b[i][j] >= 0) {

int cnt = 0;

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

int nx = i + dx[k];

int ny = j + dy[k];

if(nx >= 0 && ny >= 0 && nx < r && ny < c && a[nx][ny] != -1) {

a[nx][ny] += b[i][j] / 5;

cnt++;

}

}

a[i][j] = a[i][j] - (b[i][j]/5) \* cnt;

}

}

}

}

public static void clear(int[][] a, int x, int y, int flag, int[][] d) {

int bx, tx, ly = 0, ry = c - 1;

int tmpx = x, tmpy = y;

int idx = 0;

if(flag == 0) {

bx = x;

tx = 0;

}

else {

bx = r-1;

tx = x;

}

while(true) {

if(idx > 3) break;

int nx = tmpx + d[idx][0];

int ny = tmpy + d[idx][1];

if(nx >= tx && ny >= ly && nx <= bx && ny <= ry) {

a[tmpx][tmpy] = a[nx][ny];

tmpx = nx;

tmpy = ny;

}

else idx++;

}

a[x][y] = -1;

a[x][y+1] = 0;

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

r = sc.nextInt();

c = sc.nextInt();

int time = sc.nextInt();

int[] idx = new int[2];

int[][] up = {{-1, 0}, {0, 1}, {1, 0}, {0, -1}};

int[][] down = {{1, 0}, {0, 1}, {-1, 0}, {0, -1}};

int[][] a = new int[r][c];

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

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

a[i][j] = sc.nextInt();

if(a[i][j] == -1) {

idx[0] = i;

idx[1] = j;

}

}

}

for(int t = 0; t < time; t++) {

spread(a);

clear(a, idx[0]-1, idx[1], 0, up);

clear(a, idx[0], idx[1], 1, down);

}

int sum = 0;

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

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

if(a[i][j] == -1) continue;

sum += a[i][j];

}

}

System.out.println(sum);

}

}

## [**[백준] 14502번 - 연구소**](https://dheldh77.tistory.com/entry/%EB%B0%B1%EC%A4%80-14502%EB%B2%88-%EC%97%B0%EA%B5%AC%EC%86%8C)

2020. 3. 11. 16:32

[14502](https://dheldh77.tistory.com/tag/14502), [BaekJoon](https://dheldh77.tistory.com/tag/BaekJoon), [BFS](https://dheldh77.tistory.com/tag/BFS), [DFS](https://dheldh77.tistory.com/tag/DFS), [백준](https://dheldh77.tistory.com/tag/%EB%B0%B1%EC%A4%80), [알고리즘](https://dheldh77.tistory.com/tag/%EC%95%8C%EA%B3%A0%EB%A6%AC%EC%A6%98)

### 문제 >

인체에 치명적인 바이러스를 연구하던 연구소에서 바이러스가 유출되었다. 다행히 바이러스는 아직 퍼지지 않았고, 바이러스의 확산을 막기 위해서 연구소에 벽을 세우려고 한다.

연구소는 크기가 N×M인 직사각형으로 나타낼 수 있으며, 직사각형은 1×1 크기의 정사각형으로 나누어져 있다. 연구소는 빈 칸, 벽으로 이루어져 있으며, 벽은 칸 하나를 가득 차지한다.

일부 칸은 바이러스가 존재하며, 이 바이러스는 상하좌우로 인접한 빈 칸으로 모두 퍼져나갈 수 있다. 새로 세울 수 있는 벽의 개수는 3개이며, 꼭 3개를 세워야 한다.

예를 들어, 아래와 같이 연구소가 생긴 경우를 살펴보자.

2 0 0 0 1 1 0

0 0 1 0 1 2 0

0 1 1 0 1 0 0

0 1 0 0 0 0 0

0 0 0 0 0 1 1

0 1 0 0 0 0 0

0 1 0 0 0 0 0

이때, 0은 빈 칸, 1은 벽, 2는 바이러스가 있는 곳이다. 아무런 벽을 세우지 않는다면, 바이러스는 모든 빈 칸으로 퍼져나갈 수 있다.

2행 1열, 1행 2열, 4행 6열에 벽을 세운다면 지도의 모양은 아래와 같아지게 된다.

2 1 0 0 1 1 0

1 0 1 0 1 2 0

0 1 1 0 1 0 0

0 1 0 0 0 1 0

0 0 0 0 0 1 1

0 1 0 0 0 0 0

0 1 0 0 0 0 0

바이러스가 퍼진 뒤의 모습은 아래와 같아진다.

2 1 0 0 1 1 2

1 0 1 0 1 2 2

0 1 1 0 1 2 2

0 1 0 0 0 1 2

0 0 0 0 0 1 1

0 1 0 0 0 0 0

0 1 0 0 0 0 0

벽을 3개 세운 뒤, 바이러스가 퍼질 수 없는 곳을 안전 영역이라고 한다. 위의 지도에서 안전 영역의 크기는 27이다.

연구소의 지도가 주어졌을 때 얻을 수 있는 안전 영역 크기의 최댓값을 구하는 프로그램을 작성하시오.

### 입력 >

첫째 줄에 지도의 세로 크기 N과 가로 크기 M이 주어진다. (3 ≤ N, M ≤ 8)

둘째 줄부터 N개의 줄에 지도의 모양이 주어진다. 0은 빈 칸, 1은 벽, 2는 바이러스가 있는 위치이다. 2의 개수는 2보다 크거나 같고, 10보다 작거나 같은 자연수이다.

빈 칸의 개수는 3개 이상이다.

### 출력 >

첫째 줄에 얻을 수 있는 안전 영역의 최대 크기를 출력한다.

### 해결방법 >

DFS(깊이우선탐색) 알고리즘과 BFS(너비우선탐색)알고리즘을 함께 사용한다.

1) 연구소에 대한 입력이 주어지면 2차원 배열에 저장한다.

2) 연구소에 벽 3개를 두는 경우를 깊이우선탐색을 사용해 모두 찾는다.

3) 2)의 과정에서 경우를 찾을 때마다 너비우선탐색으로 바이러스를 확산 시켜보고, 안전영역의 수를 찾는다.

각 경우 마다 안전영역 수를 찾고, 이 중에서 최대값을 출력한다.

### [JAVA]

package baekjoon;

import java.util.\*;

public class BOJ\_14502 {

static int n;

static int m;

static int max;

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

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

public static class Node{

int x;

int y;

Node(int x, int y){

this.x = x;

this.y = y;

}

}

public static void bfs(int[][] a) {

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

int[][] visit = new int[n][m];

int[][] b = new int[n][m];

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

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

b[i][j] = a[i][j];

}

}

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

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

if(b[i][j] == 2) {

visit[i][j] = 1;

q.add(new Node(i, j));

}

}

}

while(!q.isEmpty()) {

Node node = q.poll();

int x = node.x;

int y = node.y;

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

int nx = x + dx[i];

int ny = y + dy[i];

if(nx >= 0 && ny >= 0 && nx < n && ny <m) {

if(b[nx][ny] == 0 && visit[nx][ny] == 0) {

visit[nx][ny] = 1;

b[nx][ny] = 2;

q.add(new Node(nx, ny));

}

}

}

}

int cnt = 0;

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

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

if(b[i][j] == 0) cnt++;

}

}

max = Math.max(cnt, max);

}

public static void dfs(int x,int y, int[][] a, int cnt) {

a[x][y] = 1;

if(cnt == 3) {

bfs(a);

a[x][y] = 0;

return;

}

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

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

if(a[i][j] == 0) {

dfs(i, j, a, cnt + 1);

}

}

}

a[x][y] = 0;

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

n = sc.nextInt();

m = sc.nextInt();

max = 0;

int[][] a = new int[n][m];

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

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

a[i][j] = sc.nextInt();

}

}

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

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

if(a[i][j] == 0) {

dfs(i, j, a, 1);

}

}

}

System.out.println(max);

}

}

## [**[백준] 1389번 - 케빈 베이컨의 6단계 법칙**](https://dheldh77.tistory.com/entry/%EB%B0%B1%EC%A4%80-1389%EB%B2%88-%EC%BC%80%EB%B9%88-%EB%B2%A0%EC%9D%B4%EC%BB%A8%EC%9D%98-6%EB%8B%A8%EA%B3%84-%EB%B2%95%EC%B9%99)

2020. 1. 16. 00:45

### 문제 >

케빈 베이컨의 6단계 법칙에 의하면 지구에 있는 모든 사람들은 최대 6단계 이내에서 서로 아는 사람으로 연결될 수 있다. 케빈 베이컨 게임은 임의의 두 사람이 최소 몇 단계 만에 이어질 수 있는지 계산하는 게임이다.

예를 들면, 전혀 상관없을 것 같은 인하대학교의 이강호와 서강대학교의 민세희는 몇 단계만에 이어질 수 있을까?

천민호는 이강호와 같은 학교에 다니는 사이이다. 천민호와 최백준은 Baekjoon Online Judge를 통해 알게 되었다. 최백준과 김선영은 같이 Startlink를 창업했다. 김선영과 김도현은 같은 학교 동아리 소속이다. 김도현과 민세희는 같은 학교에 다니는 사이로 서로 알고 있다. 즉, 이강호-천민호-최백준-김선영-김도현-민세희 와 같이 5단계만 거치면 된다.

케빈 베이컨은 미국 헐리우드 영화배우들 끼리 케빈 베이컨 게임을 했을때 나오는 단계의 총 합이 가장 적은 사람이라고 한다.

오늘은 Baekjoon Online Judge의 유저 중에서 케빈 베이컨의 수가 가장 작은 사람을 찾으려고 한다. 케빈 베이컨 수는 모든 사람과 케빈 베이컨 게임을 했을 때, 나오는 단계의 합이다.

예를 들어, BOJ의 유저가 5명이고, 1과 3, 1과 4, 2와 3, 3과 4, 4와 5가 친구인 경우를 생각해보자.

1은 2까지 3을 통해 2단계 만에, 3까지 1단계, 4까지 1단계, 5까지 4를 통해서 2단계 만에 알 수 있다. 따라서, 케빈 베이컨의 수는 2+1+1+2 = 6이다.

2는 1까지 3을 통해서 2단계 만에, 3까지 1단계 만에, 4까지 3을 통해서 2단계 만에, 5까지 3과 4를 통해서 3단계 만에 알 수 있다. 따라서, 케빈 베이컨의 수는 2+1+2+3 = 8이다.

3은 1까지 1단계, 2까지 1단계, 4까지 1단계, 5까지 4를 통해 2단계 만에 알 수 있다. 따라서, 케빈 베이컨의 수는 1+1+1+2 = 5이다.

4는 1까지 1단계, 2까지 3을 통해 2단계, 3까지 1단계, 5까지 1단계 만에 알 수 있다. 4의 케빈 베이컨의 수는 1+2+1+1 = 5가 된다.

마지막으로 5는 1까지 4를 통해 2단계, 2까지 4와 3을 통해 3단계, 3까지 4를 통해 2단계, 4까지 1단계 만에 알 수 있다. 5의 케빈 베이컨의 수는 2+3+2+1 = 8이다.

5명의 유저 중에서 케빈 베이컨의 수가 가장 작은 사람은 3과 4이다.

BOJ 유저의 수와 친구 관계가 입력으로 주어졌을 때, 케빈 베이컨의 수가 가장 작은 사람을 구하는 프로그램을 작성하시오.

### 입력 >

첫째 줄에 유저의 수 N (2 ≤ N ≤ 100)과 친구 관계의 수 M (1 ≤ M ≤ 5,000)이 주어진다. 둘째 줄부터 M개의 줄에는 친구 관계가 주어진다. 친구 관계는 A와 B로 이루어져 있으며, A와 B가 친구라는 뜻이다. A와 B가 친구이면, B와 A도 친구이며, A와 B가 같은 경우는 없다. 친구 관계는 중복되어 들어올 수도 있으며, 친구가 한 명도 없는 사람은 없다. 또, 모든 사람은 친구 관계로 연결되어져 있다.

### 출력 >

첫째 줄에 BOJ의 유저 중에서 케빈 베이컨의 수가 가장 작은 사람을 출력한다. 그런 사람이 여러 명일 경우에는 번호가 가장 작은 사람을 출력한다.

### 해결방법 >

**너비 우선 탐색(BFS) 알고리즘**으로 문제 해결

너비 우선 탐색으로 해결할 수 있는 대표적인 문제로 각 i, j간의 최단 경로의 길이를 구한다

### [JAVA]

package baekjoon;

import java.util.\*;

public class BOJ\_1389 {

static int n;

static int[][] map;

static int[][] d;

public static void bfs(int x) {

Queue <Integer> q = new LinkedList();

boolean[] visit = new boolean[n + 1];

for(int i = 1; i <= n; i++) {

if(map[x][i] == 1) {

visit[i] = true;

d[x][i] = 1;

q.add(i);

}

}

while(!q.isEmpty()) {

int y = q.poll();

for(int i = 1; i <= n; i++) {

if(map[y][i] == 1 && visit[i] == false) {

q.add(i);

d[x][i] = d[x][y] + 1;

visit[i] = true;

}

}

}

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

n = sc.nextInt();

map = new int[n + 1][n + 1];

d = new int[n +1][n + 1];

int e = sc.nextInt();

int min = 7 \* n;

int idx = 0;

for(int i = 1; i <= e; i++) {

int x = sc.nextInt();

int y = sc.nextInt();

map[x][y] = 1;

map[y][x] = 1;

}

for(int i = 1; i <= n; i++) {

bfs(i);

}

for(int i = 1; i <= n; i++) {

int sum = 0;

for(int j = 1; j <= n; j++) {

sum += d[i][j];

}

if(sum < min) {

min = sum;

idx = i;

}

}

System.out.println(idx);

}

}

### 문제링크 >

<https://www.acmicpc.net/problem/2668>

## [**[백준] 2668번 - 숫자 고르기**](https://dheldh77.tistory.com/entry/%EB%B0%B1%EC%A4%80-1946%EB%B2%88-%EC%8B%A0%EC%9E%85-%EC%82%AC%EC%9B%90-1)

2020. 1. 16. 00:27

### 문제 >

세로 두 줄, 가로로 N개의 칸으로 이루어진 표가 있다. 첫째 줄의 각 칸에는 정수 1, 2, …, N이 차례대로 들어 있고 둘째 줄의 각 칸에는 1이상 N이하인 정수가 들어 있다. 첫째 줄에서 숫자를 적절히 뽑으면, 그 뽑힌 정수들이 이루는 집합과, 뽑힌 정수들의 바로 밑의 둘째 줄에 들어있는 정수들이 이루는 집합이 일치한다. 이러한 조건을 만족시키도록 정수들을 뽑되, 최대로 많이 뽑는 방법을 찾는 프로그램을 작성하시오. 예를 들어, N=7인 경우 아래와 같이 표가 주어졌다고 하자.

https://k.kakaocdn.net/dn/uPkMY/btqBeLgn46k/FikeBikNkeR3xjyQMld8bk/img.png

이 경우에는 첫째 줄에서 1, 3, 5를 뽑는 것이 답이다. 첫째 줄의 1, 3, 5밑에는 각각 3, 1, 5가 있으며 두 집합은 일치한다. 이때 집합의 크기는 3이다. 만약 첫째 줄에서 1과 3을 뽑으면, 이들 바로 밑에는 정수 3과 1이 있으므로 두 집합이 일치한다. 그러나, 이 경우에 뽑힌 정수의 개수는 최대가 아니므로 답이 될 수 없다.

### 입력 >

첫째 줄에는 N(1≤N≤100)을 나타내는 정수 하나가 주어진다. 그 다음 줄부터는 표의 둘째 줄에 들어가는 정수들이 순서대로 한 줄에 하나씩 입력된다.

### 출력 >

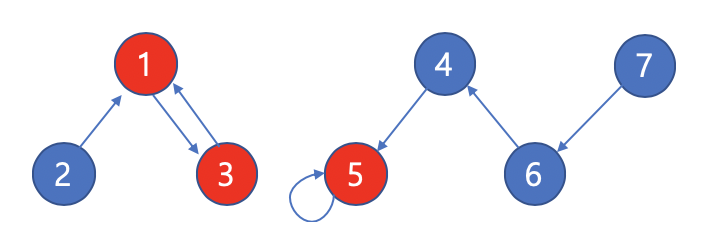
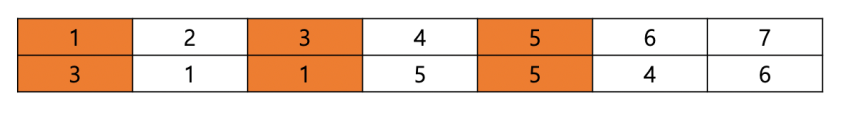
첫째 줄에 뽑힌 정수들의 개수를 출력하고, 그 다음 줄부터는 뽑힌 정수들을 작은 수부터 큰 수의 순서로 한 줄에 하나씩 출력한다.

### 해결방법 >

**깊이 우선 탐색(DFS) 알고리즘**으로 문제 해결

예제에 있는 데이터로 그래프를 그려보면 경로가 cycle을 가졌을 때, 두 집합에 모두 포함되게 된다.

즉, 입력받은 데이터가 방향이 있는 그래프의 간선에 대한 정보로 보고, 이 그래프에서 cycle을 가지는 경로의 정점을 찾는 문제이다.



dfs로 cycle을 가지는 경로에 포함되는 정점을 모두 찾아준다.

### [JAVA]

package baekjoon;

import java.util.\*;

public class BOJ\_2668 {

static int[][] map;

static boolean[] visit;

static int v;

static int cnt = 0;

static ArrayList <Integer> list = new ArrayList<>();

public static void dfs(int x, int y) {

if(x == y) {

list.add(x);

cnt++;

}

visit[x] = true;

for(int i = 1; i <= v; i++) {

if(map[x][i] == 1 && visit[i] == false) {

dfs(i, y);

}

visit[i] = false;

}

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

v = sc.nextInt();

map = new int[v+1][v+1];

visit = new boolean[v+1];

for(int i = 1; i <= v; i++) {

int x = sc.nextInt();

map[i][x] = 1;

}

for(int i = 1; i <= v; i++) {

for(int j = 1; j <= v; j++) {

if(map[i][j] == 1 && visit[j] == false) {

dfs(j, i);

}

}

}

System.out.println(cnt);

for(int i = 0; i < list.size(); i++) {

System.out.println(list.get(i));

}

}

}

### 문제링크 >

<https://www.acmicpc.net/problem/2668>

## [**[백준] 1012번 - 유기농 배추**](https://dheldh77.tistory.com/entry/%EB%B0%B1%EC%A4%80-1012%EB%B2%88-%EC%9C%A0%EA%B8%B0%EB%86%8D-%EB%B0%B0%EC%B6%94)

2020. 1. 14. 02:45

### 문제 >

차세대 영농인 한나는 강원도 고랭지에서 유기농 배추를 재배하기로 하였다. 농약을 쓰지 않고 배추를 재배하려면 배추를 해충으로부터 보호하는 것이 중요하기 때문에, 한나는 해충 방지에 효과적인 배추흰지렁이를 구입하기로 결심한다. 이 지렁이는 배추근처에 서식하며 해충을 잡아 먹음으로써 배추를 보호한다. 특히, 어떤 배추에 배추흰지렁이가 한 마리라도 살고 있으면 이 지렁이는 인접한 다른 배추로 이동할 수 있어, 그 배추들 역시 해충으로부터 보호받을 수 있다.

(한 배추의 상하좌우 네 방향에 다른 배추가 위치한 경우에 서로 인접해있다고 간주한다)

한나가 배추를 재배하는 땅은 고르지 못해서 배추를 군데군데 심어놓았다. 배추들이 모여있는 곳에는 배추흰지렁이가 한 마리만 있으면 되므로 서로 인접해있는 배추들이 몇 군데에 퍼져있는지 조사하면 총 몇 마리의 지렁이가 필요한지 알 수 있다.

예를 들어 배추밭이 아래와 같이 구성되어 있으면 최소 5마리의 배추흰지렁이가 필요하다.

(0은 배추가 심어져 있지 않은 땅이고, 1은 배추가 심어져 있는 땅을 나타낸다.)

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **1** | **1** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | **1** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | **1** | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | **1** | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | **1** | **1** | 0 | 0 | 0 | **1** | **1** | **1** |
| 0 | 0 | 0 | 0 | **1** | 0 | 0 | **1** | **1** | **1** |

### 입력 >

입력의 첫 줄에는 테스트 케이스의 개수 T가 주어진다. 그 다음 줄부터 각각의 테스트 케이스에 대해 첫째 줄에는 배추를 심은 배추밭의 가로길이 M(1 ≤ M ≤ 50)과 세로길이 N(1 ≤ N ≤ 50), 그리고 배추가 심어져 있는 위치의 개수 K(1 ≤ K ≤ 2500)이 주어진다. 그 다음 K줄에는 배추의 위치 X(0 ≤ X ≤ M-1), Y(0 ≤ Y ≤ N-1)가 주어진다.

### 출력 >

 테스트 케이스에 대해 필요한 최소의 배추흰지렁이 마리 수를 출력한다.

### 해결방법 >

**깊이 우선 탐색(BFS) 알고리즘**으로 문제 해결

깊이 우선 탐색으로  풀 수 있는 기본적인 문제로 각 영역의 개수를 구하는 문제. 입력

받은 데이터로부터 존재하는 경로의 개수를 구하는 문제

### [JAVA]

package baekjoon;

import java.util.\*;

public class BOJ\_1012 {

static int[][] map;

static boolean[][] c;

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

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

static int[] cnt;

static int m,n;

public static void dfs(int x, int y) {

c[x][y] = true;

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

int nx = x + dx[i];

int ny = y + dy[i];

if(nx >= 0 && ny >= 0 && nx < m && ny < n) {

if(map[nx][ny] == 1 && c[nx][ny] == false) {

dfs(nx,ny);

}

}

}

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

int t = sc.nextInt();

cnt = new int[t];

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

m = sc.nextInt();

n = sc.nextInt();

int b = sc.nextInt();

map = new int[m][n];

c = new boolean[m][n];

cnt[i] = 0;

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

int x = sc.nextInt();

int y = sc.nextInt();

map[x][y] = 1;

}

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

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

if(map[j][k] == 1 && c[j][k] == false) {

cnt[i]++;

dfs(j,k);

}

}

}

}

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

System.out.println(cnt[i]);

}

}

}

### 문제링크 >

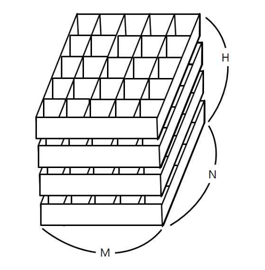
<https://www.acmicpc.net/problem/1012>

## [**[백준] 7569번 - 토마토**](https://dheldh77.tistory.com/entry/%EB%B0%B1%EC%A4%80-7569%EB%B2%88-%ED%86%A0%EB%A7%88%ED%86%A0)

2020. 1. 14. 00:33

### 문제 >

철수의 토마토 농장에서는 토마토를 보관하는 큰 창고를 가지고 있다. 토마토는 아래의 그림과 같이 격자모양 상자의 칸에 하나씩 넣은 다음, 상자들을 수직으로 쌓아 올려서 창고에 보관한다.



창고에 보관되는 토마토들 중에는 잘 익은 것도 있지만, 아직 익지 않은 토마토들도 있을 수 있다. 보관 후 하루가 지나면, 익은 토마토들의 인접한 곳에 있는 익지 않은 토마토들은 익은 토마토의 영향을 받아 익게 된다. 하나의 토마토에 인접한 곳은 위, 아래, 왼쪽, 오른쪽, 앞, 뒤 여섯 방향에 있는 토마토를 의미한다. 대각선 방향에 있는 토마토들에게는 영향을 주지 못하며, 토마토가 혼자 저절로 익는 경우는 없다고 가정한다. 철수는 창고에 보관된 토마토들이 며칠이 지나면 다 익게 되는지 그 최소 일수를 알고 싶어 한다.

토마토를 창고에 보관하는 격자모양의 상자들의 크기와 익은 토마토들과 익지 않은 토마토들의 정보가 주어졌을 때, 며칠이 지나면 토마토들이 모두 익는지, 그 최소 일수를 구하는 프로그램을 작성하라. 단, 상자의 일부 칸에는 토마토가 들어있지 않을 수도 있다.

### 입력 >

첫 줄에는 상자의 크기를 나타내는 두 정수 M,N과 쌓아올려지는 상자의 수를 나타내는 H가 주어진다. M은 상자의 가로 칸의 수, N은 상자의 세로 칸의 수를 나타낸다. 단, 2 ≤ M ≤ 100, 2 ≤ N ≤ 100, 1 ≤ H ≤ 100 이다. 둘째 줄부터는 가장 밑의 상자부터 가장 위의 상자까지에 저장된 토마토들의 정보가 주어진다. 즉, 둘째 줄부터 N개의 줄에는 하나의 상자에 담긴 토마토의 정보가 주어진다. 각 줄에는 상자 가로줄에 들어있는 토마토들의 상태가 M개의 정수로 주어진다. 정수 1은 익은 토마토, 정수 0 은 익지 않은 토마토, 정수 -1은 토마토가 들어있지 않은 칸을 나타낸다. 이러한 N개의 줄이 H번 반복하여 주어진다.

### 출력 >

여러분은 토마토가 모두 익을 때까지 최소 며칠이 걸리는지를 계산해서 출력해야 한다. 만약, 저장될 때부터 모든 토마토가 익어있는 상태이면 0을 출력해야 하고, 토마토가 모두 익지는 못하는 상황이면 -1을 출력해야 한다.

### 해결방법 >

**너비 우선 탐색(BFS) 알고리즘**으로 문제 해결 [[백준] 7576번 - 토마토] 문제에 배열의 차원이 한차원 더 높아진 문제이다.

[[백준] 7576번 - 토마토] 문제와 마찬가지로 상자 안에는 익은 토마토가 여러개 존재할 수 있다. 따라서, 입력 데이터에서 **1로 표시된 정점을 먼저 큐에 넣은 다음** BFS 탐색으로 각 정점을 1로 표시된 정점으로부터 최단 거리로 값을 변경해준다. 이 때, 탐색 시 **z축이 방향으로 탐색**을 추가해야한다.

만약 탐색을 한 번도 하지 않았다면 저장될 때부터 모든 토마토가 익어있는 상태이므로 0을 출력

BFS 탐색으로 모든 정점에 값을 변경한 뒤에도 0이 존재한다면 익지 않은 토마토가 존재하므로 -1을 출력

위의 두 경우에도 포함되지 않는다면 최대값을 출력한다.

### [JAVA]

package baekjoon;

import java.util.\*;

public class BOJ\_7569 {

static int[][][] map;

static boolean[][][] visit;

static int m, n, h;

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

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

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

static boolean flag;

static class Node{

int x;

int y;

int z;

Node(int z, int x, int y){

this.x = x;

this.y = y;

this.z = z;

}

}

public static void bfs() {

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

int max = 0;

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

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

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

if(map[i][j][k] == 1) {

q.add(new Node(i, j, k));

visit[i][j][k] = true;

}

}

}

}

while(!q.isEmpty()) {

Node tmp = q.poll();

int x = tmp.x;

int y = tmp.y;

int z = tmp.z;

for(int i = 0; i < 6; i++) {

int nx = x + dx[i];

int ny = y + dy[i];

int nz = z + dz[i];

if(nx >= 0 && ny >= 0 && nz >= 0 && nz < h && nx < m && ny < n) {

if(map[nz][nx][ny] == 0 && visit[nz][nx][ny] == false) {

flag = true;

q.add(new Node(nz, nx, ny));

visit[nz][nx][ny] = true;

map[nz][nx][ny] = map[z][x][y] + 1;

}

}

}

}

if(flag == false) {

System.out.println(0);

}

else {

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

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

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

if(map[i][j][k] == 0) {

System.out.println(-1);

return;

}

max = (max > map[i][j][k]) ? max : map[i][j][k];

}

}

}

System.out.println(max - 1);

}

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

n = sc.nextInt();

m = sc.nextInt();

h = sc.nextInt();

map = new int[h][m][n];

visit = new boolean[h][m][n];

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

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

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

map[i][j][k] = sc.nextInt();

}

}

}

bfs();

}

}

### 문제링크 >

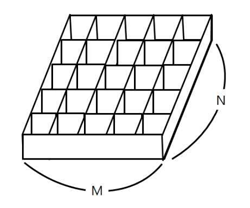
<https://www.acmicpc.net/problem/7569>

## [**[백준] 7576번 - 토마토**](https://dheldh77.tistory.com/entry/%EB%B0%B1%EC%A4%80-7576%EB%B2%88-%ED%86%A0%EB%A7%88%ED%86%A0)

2020. 1. 14. 00:28

### 문제 >

철수의 토마토 농장에서는 토마토를 보관하는 큰 창고를 가지고 있다. 토마토는 아래의 그림과 같이 격자 모양 상자의 칸에 하나씩 넣어서 창고에 보관한다.



창고에 보관되는 토마토들 중에는 잘 익은 것도 있지만, 아직 익지 않은 토마토들도 있을 수 있다. 보관 후 하루가 지나면, 익은 토마토들의 인접한 곳에 있는 익지 않은 토마토들은 익은 토마토의 영향을 받아 익게 된다. 하나의 토마토의 인접한 곳은 왼쪽, 오른쪽, 앞, 뒤 네 방향에 있는 토마토를 의미한다. 대각선 방향에 있는 토마토들에게는 영향을 주지 못하며, 토마토가 혼자 저절로 익는 경우는 없다고 가정한다. 철수는 창고에 보관된 토마토들이 며칠이 지나면 다 익게 되는지, 그 최소 일수를 알고 싶어 한다.

토마토를 창고에 보관하는 격자모양의 상자들의 크기와 익은 토마토들과 익지 않은 토마토들의 정보가 주어졌을 때, 며칠이 지나면 토마토들이 모두 익는지, 그 최소 일수를 구하는 프로그램을 작성하라. 단, 상자의 일부 칸에는 토마토가 들어있지 않을 수도 있다.

### 입력 >

첫 줄에는 상자의 크기를 나타내는 두 정수 M,N이 주어진다. M은 상자의 가로 칸의 수, N은 상자의 세로 칸의 수를 나타낸다. 단, 2 ≤ M,N ≤ 1,000 이다. 둘째 줄부터는 하나의 상자에 저장된 토마토들의 정보가 주어진다. 즉, 둘째 줄부터 N개의 줄에는 상자에 담긴 토마토의 정보가 주어진다. 하나의 줄에는 상자 가로줄에 들어있는 토마토의 상태가 M개의 정수로 주어진다. 정수 1은 익은 토마토, 정수 0은 익지 않은 토마토, 정수 -1은 토마토가 들어있지 않은 칸을 나타낸다.

### 출력 >

여러분은 토마토가 모두 익을 때까지의 최소 날짜를 출력해야 한다. 만약, 저장될 때부터 모든 토마토가 익어있는 상태이면 0을 출력해야 하고, 토마토가 모두 익지는 못하는 상황이면 -1을 출력해야 한다.

### 해결방법 >

**너비 우선 탐색(BFS) 알고리즘**으로 문제 해결

상자 안에는 익은 토마토가 여러개 존재할 수 있다. 따라서, 입력 데이터에서 **1로 표시된 정점을 먼저 큐에 넣은 다음** BFS 탐색으로 각 정점을 1로 표시된 정점으로부터 최단 거리로 값을 변경해준다.

만약 탐색을 한 번도 하지 않았다면 저장될 때부터 모든 토마토가 익어있는 상태이므로 0을 출력

BFS 탐색으로 모든 정점에 값을 변경한 뒤에도 0이 존재한다면 익지 않은 토마토가 존재하므로 -1을 출력

위의 두 경우에도 포함되지 않는다면 최대값을 출력한다.

### [JAVA]

package baekjoon;

import java.util.\*;

public class BOJ\_7576{

static int[][] map;

static boolean[][] visit;

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

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

static int m, n;

static boolean flag;

static class Node{

int x;

int y;

Node(int x, int y){

this.x = x;

this.y= y;

}

}

public static void bfs() {

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

int max = 0;

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

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

if(map[i][j] == 1) {

q.add(new Node(i, j));

visit[i][j] = true;

}

}

}

while(!q.isEmpty()) {

Node tmp = q.poll();

int x = tmp.x;

int y = tmp.y;

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

int nx = x + dx[i];

int ny = y + dy[i];

if(nx >= 0 && ny >= 0 && nx < m && ny < n) {

if(map[nx][ny] == 0 && visit[nx][ny] == false) {

flag = true;

q.add(new Node(nx, ny));

visit[nx][ny] = true;

map[nx][ny] = map[x][y] + 1;

}

}

}

}

if(flag == false)

System.out.println(0);

else {

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

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

if(map[i][j] == 0) {

System.out.println(-1);

return;

}

max = (max > map[i][j]) ? max : map[i][j];

}

}

System.out.println(max - 1);

}

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

n = sc.nextInt();

m = sc.nextInt();

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

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

map[i][j] = sc.nextInt();

}

}

bfs();

}

}

### 문제링크 >

<https://www.acmicpc.net/problem/7576>

## [**[백준] 2178번 - 미로 탐색**](https://dheldh77.tistory.com/entry/%EB%B0%B1%EC%A4%80-2178%EB%B2%88-%EB%AF%B8%EB%A1%9C-%ED%83%90%EC%83%89)

2020. 1. 14. 00:19

### 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.

### 입력 >

첫째 줄에 두 정수 N, M(2 ≤ N, M ≤ 100)이 주어진다. 다음 N개의 줄에는 M개의 정수로 미로가 주어진다. 각각의 수들은 **붙어서** 입력으로 주어진다.

### 출력 >

첫째 줄에 지나야 하는 최소의 칸 수를 출력한다. 항상 도착위치로 이동할 수 있는 경우만 입력으로 주어진다.

### 해결방법 >

**너비 우선 탐색(BFS) 알고리즘**으로 해결

너비 우선 탐색을 사용해 각 정점에 최단 거리를 찾는다.

### [JAVA]

package baekjoon;

import java.util.\*;

public class BOJ\_2178{

static int[][] map;

static boolean[][] visit;

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

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

static int m, n;

static class Node{

int x;

int y;

Node(int x, int y){

this.x = x;

this.y = y;

}

}

public static void bfs(int x, int y) {

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

q.add(new Node(x, y));

visit[x][y] = true;

while(!q.isEmpty()) {

Node tmp = q.poll();

x = tmp.x;

y = tmp.y;

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

int nx = x + dx[i];

int ny = y + dy[i];

if(nx >= 0 && ny >= 0 && nx < m && ny <n) {

if(map[nx][ny] == 1 && visit[nx][ny] == false) {

q.add(new Node(nx, ny));

map[nx][ny] = map[x][y] + 1;

visit[nx][ny] = true;

}

}

}

}

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

m = sc.nextInt();

n = sc.nextInt();

map = new int[m][n];

visit = new boolean[m][n];

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

String s = sc.next();

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

map[i][j] = s.charAt(j) - '0';

}

}

bfs(0,0);

System.out.println(map[m-1][n-1]);

}

}

### 문제링크 >

<https://www.acmicpc.net/problem/2178>

## [**[백준] 1206번 - DFS와 BFS**](https://dheldh77.tistory.com/entry/%EB%B0%B1%EC%A4%80-1206%EB%B2%88-DFS%EC%99%80-BFS)

2020. 1. 14. 00:11

### 문제 >

그래프를 DFS로 탐색한 결과와 BFS로 탐색한 결과를 출력하는 프로그램을 작성하시오. 단, 방문할 수 있는 정점이 여러 개인 경우에는 정점 번호가 작은 것을 먼저 방문하고, 더 이상 방문할 수 있는 점이 없는 경우 종료한다. 정점 번호는 1번부터 N번까지이다.

### 입력 >

첫째 줄에 정점의 개수 N(1 ≤ N ≤ 1,000), 간선의 개수 M(1 ≤ M ≤ 10,000), 탐색을 시작할 정점의 번호 V가 주어진다. 다음 M개의 줄에는 간선이 연결하는 두 정점의 번호가 주어진다. 어떤 두 정점 사이에 여러 개의 간선이 있을 수 있다. 입력으로 주어지는 간선은 양방향이다.

### 출력 >

첫째 줄에 DFS를 수행한 결과를, 그 다음 줄에는 BFS를 수행한 결과를 출력한다. V부터 방문된 점을 순서대로 출력하면 된다.

### 해결방법 >

**깊이 우선 탐색(DFS)**와**너비 우선 탐색(BFS) 알고리즘**으로 해결

### [JAVA]

package baekjoon;

import java.util.\*;

public class BOJ\_1260 {

static int[][] map;

static boolean[] visit;

static int n;

public static void dfs(int x) {

visit[x] = true;

System.out.print((x+1) + " ");

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

if(map[x][i] == 1 && visit[i] == false) {

dfs(i);

}

}

}

public static void bfs(int x) {

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

q.add(x);

visit[x] = true;

while(!q.isEmpty()) {

x = q.poll();

System.out.print((x + 1) + " ");

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

if(map[x][i] == 1 && visit[i] == false) {

q.add(i);

visit[i] = true;

}

}

}

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

n = sc.nextInt();

int m = sc.nextInt();

int start = sc.nextInt();

map = new int[n][n];

visit = new boolean[n];

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

int x = sc.nextInt();

int y = sc.nextInt();

map[x - 1][y - 1] = 1;

map[y - 1][x - 1] = 1;

}

dfs(start - 1);

System.out.println();

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

visit[i] = false;

}

bfs(start - 1);

}

}

### 문제링크 >

<https://www.acmicpc.net/problem/1260>

## [**[백준] 1987번 - 알파벳**](https://dheldh77.tistory.com/entry/%EB%B0%B1%EC%A4%80-1986%EB%B2%88-%EC%95%8C%ED%8C%8C%EB%B2%B3)

2020. 1. 14. 00:01

### 문제 >

세로 R칸, 가로 C칸으로 된 표 모양의 보드가 있다. 보드의 각 칸에는 대문자 알파벳이 하나씩 적혀 있고, 좌측 상단 칸 (1행 1열) 에는 말이 놓여 있다.

말은 상하좌우로 인접한 네 칸 중의 한 칸으로 이동할 수 있는데, 새로 이동한 칸에 적혀 있는 알파벳은 지금까지 지나온 모든 칸에 적혀 있는 알파벳과는 달라야 한다. 즉, 같은 알파벳이 적힌 칸을 두 번 지날 수 없다.

좌측 상단에서 시작해서, 말이 최대한 몇 칸을 지날 수 있는지를 구하는 프로그램을 작성하시오. 말이 지나는 칸은 좌측 상단의 칸도 포함된다.

### 입력 >

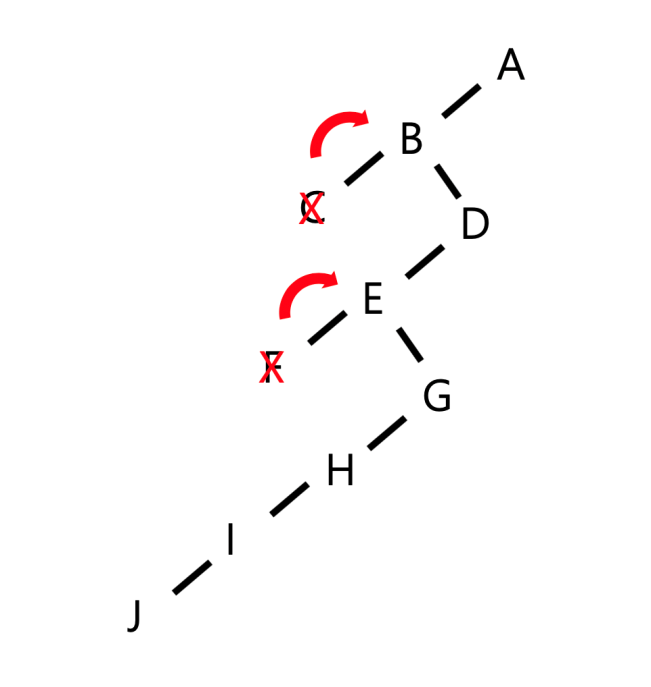
첫째 줄에 R과 C가 빈칸을 사이에 두고 주어진다. (1<=R,C<=20) 둘째 줄부터 R개의 줄에 걸쳐서 보드에 적혀 있는 C개의 대문자 알파벳들이 빈칸 없이 주어진다.

### 출력 >

첫째 줄에 말이 지날 수 있는 최대의 칸 수를 출력한다.

### 해결방법 >

**깊이 우선 탐색(DFS) 되추적(Backtracking)**으로 해결



깊이 우선 탐색으로 탐색을 계속 진행하다가 더 이상 인접한 정점이 없는 정점에 방문했을 때, 이전 정점으로 되돌아가 다시 탐색을 한다.

탐색한 경로 중 제일 긴 경로의 길이를 찾는다.

### [JAVA]

package baekjoon;

import java.util.\*;

public class BOJ\_1987 {

static int[][] map;

static boolean[] visit;

static int m, n;

static int cnt = 0;

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

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

public static void dfs(int x, int y, int depth) {

visit[map[x][y]] = true;

depth++;

cnt = (cnt > depth) ? cnt : depth;

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

int nx = x + dx[i];

int ny = y + dy[i];

if(nx >= 0 && ny >= 0 && nx < m && ny < n) {

if(visit[map[nx][ny]] == false) {

dfs(nx, ny, depth);

visit[map[nx][ny]] = false;

}

}

}

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

visit = new boolean[43];

m = sc.nextInt();

n = sc.nextInt();

map = new int[m][n];

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

String s = sc.next();

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

map[i][j] = s.charAt(j) - '0';

}

}

dfs(0, 0, 0);

System.out.println(cnt);

}

}

### 문제링크 >

<https://www.acmicpc.net/problem/1987>

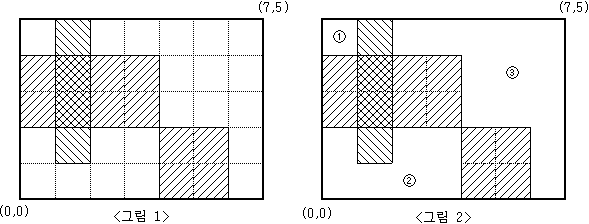
## [**[백준] 2583번 - 영역 구하기**](https://dheldh77.tistory.com/entry/%EB%B0%B1%EC%A4%80-2583%EB%B2%88-%EC%98%81%EC%97%AD-%EA%B5%AC%ED%95%98%EA%B8%B0)

2020. 1. 13. 22:02

### 문제 >

눈금의 간격이 1인 M×N(M,N≤100)크기의 모눈종이가 있다. 이 모눈종이 위에 눈금에 맞추어 K개의 직사각형을 그릴 때, 이들 K개의 직사각형의 내부를 제외한 나머지 부분이 몇 개의 분리된 영역으로 나누어진다.

예를 들어 M=5, N=7 인 모눈종이 위에 <그림 1>과 같이 직사각형 3개를 그렸다면, 그 나머지 영역은 <그림 2>와 같이 3개의 분리된 영역으로 나누어지게 된다.



<그림 2>와 같이 분리된 세 영역의 넓이는 각각 1, 7, 13이 된다.

M, N과 K 그리고 K개의 직사각형의 좌표가 주어질 때, K개의 직사각형 내부를 제외한 나머지 부분이 몇 개의 분리된 영역으로 나누어지는지, 그리고 분리된 각 영역의 넓이가 얼마인지를 구하여 이를 출력하는 프로그램을 작성하시오.

### 입력 >

첫째 줄에 M과 N, 그리고 K가 빈칸을 사이에 두고 차례로 주어진다. M, N, K는 모두 100 이하의 자연수이다. 둘째 줄부터 K개의 줄에는 한 줄에 하나씩 직사각형의 왼쪽 아래 꼭짓점의 x, y좌표값과 오른쪽 위 꼭짓점의 x, y좌표값이 빈칸을 사이에 두고 차례로 주어진다. 모눈종이의 왼쪽 아래 꼭짓점의 좌표는 (0,0)이고, 오른쪽 위 꼭짓점의 좌표는(N,M)이다. 입력되는 K개의 직사각형들이 모눈종이 전체를 채우는 경우는 없다.

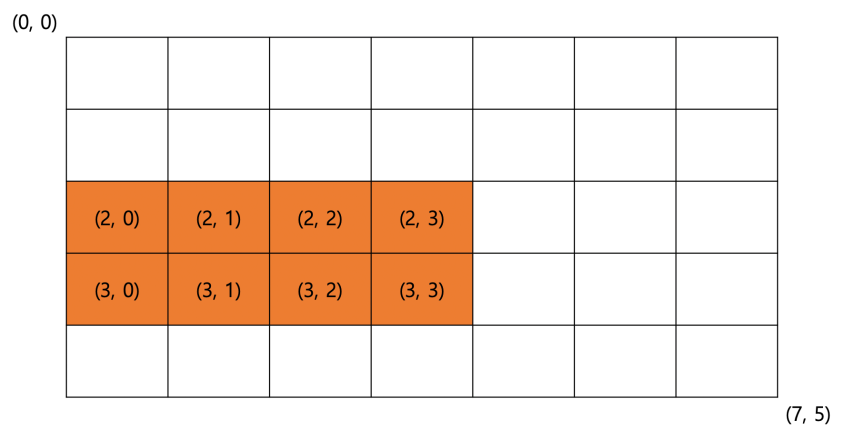
### 출력 >

첫째 줄에 분리되어 나누어지는 영역의 개수를 출력한다. 둘째 줄에는 각 영역의 넓이를 오름차순으로 정렬하여 빈칸을 사이에 두고 출력한다.

### 해결방법 >

**깊이 우선 탐색(DFS) 알고리즘**으로 해결

입력 받은 데이터의 영역을 제외하고 나머지 부분의 영역의 개수와 영역의 크기를 구하는 문제



입력받은 직사각형의 좌표가 0 2 4 4일 때,

행은 2부터 (4-1)까지 열은 0부터 (4-1)까지 방문에서 제외되는 원소로 표시한다.

문제에서 왼쪽 아랫부분부터 (0, 0)으로 시작된다고 하는데 신경쓰지 않아도 된다.

2차원 배열은 실제로 메모리상에서 2차원으로 존재하지도 않고 경로의 개수와 길이만 구하면 되기 때문이다.

### [JAVA]

package baekjoon;

import java.util.\*;

public class BOJ\_2583 {

static int[][] map;

static boolean[][] visit;

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

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

static int m, n;

static int cnt;

static ArrayList area;

public static void dfs(int x, int y) {

visit[x][y] = true;

area.set(cnt, (int)area.get(cnt)+1);

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

int nx = x + dx[i];

int ny = y + dy[i];

if(nx >= 0 && ny >= 0 && nx < m && ny < n) {

if(map[nx][ny] == 0 && visit[nx][ny] == false)

dfs(nx, ny);

}

}

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

m = sc.nextInt();

n = sc.nextInt();

map = new int[m][n];

visit = new boolean[m][n];

area = new ArrayList();

cnt = 0;

int a = sc.nextInt();

int[] p = new int[4];

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

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

p[j] = sc.nextInt();

}

for(int j = p[1]; j < p[3]; j++) {

for(int k = p[0]; k < p[2]; k++) {

map[j][k] = 1;

}

}

}

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

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

if(map[i][j] == 0 && visit[i][j] == false) {

area.add(0);

dfs(i, j);

cnt++;

}

}

}

Collections.sort(area);

System.out.println(cnt);

for(int i = 0; i < area.size(); i++) {

System.out.print(area.get(i) + " ");

}

}

}

### 문제링크 >

<https://www.acmicpc.net/problem/2583>

## [**[백준] 10026번 - 적록색약**](https://dheldh77.tistory.com/entry/%EB%B0%B1%EC%A4%80-10026%EB%B2%88-%EC%A0%81%EB%A1%9D%EC%83%89%EC%95%BD)

2020. 1. 11. 23:59

### 문제 >

적록색약은 빨간색과 초록색의 차이를 거의 느끼지 못한다. 따라서, 적록색약인 사람이 보는 그림은 아닌 사람이 보는 그림과는 좀 다를 수 있다.

크기가 N×N인 그리드의 각 칸에 R(빨강), G(초록), B(파랑) 중 하나를 색칠한 그림이 있다. 그림은 몇 개의 구역으로 나뉘어져 있는데, 구역은 같은 색으로 이루어져 있다. 또, 같은 색상이 상하좌우로 인접해 있는 경우에 두 글자는 같은 구역에 속한다. (색상의 차이를 거의 느끼지 못하는 경우도 같은 색상이라 한다)

예를 들어, 그림이 아래와 같은 경우에

RRRBB

GGBBB

BBBRR

BBRRR

RRRRR

적록색약이 아닌 사람이 봤을 때 구역의 수는 총 4개이다. (빨강 2, 파랑 1, 초록 1) 하지만, 적록색약인 사람은 구역을 3개 볼 수 있다. (빨강-초록 2, 파랑 1)

그림이 입력으로 주어졌을 때, 적록색약인 사람이 봤을 때와 아닌 사람이 봤을 때 구역의 수를 구하는 프로그램을 작성하시오.

### 입력 >

첫째 줄에 N이 주어진다. (1 ≤ N ≤ 100)

둘째 줄부터 N개 줄에는 그림이 주어진다.

### 출력 >

적록색약이 아닌 사람이 봤을 때의 구역의 개수와 적록색약인 사람이 봤을 때의 구역의 수를 공백으로 구분해 출력한다.

### 해결방법 >

**깊이 우선 탐색(DFS) 알고리즘**으로 해결

입력받은 2차원 배열에서 영역을 구하는 문제 또는 그래프에서 연결의 개수를 구하는 문제이다.

이 때, 적록색약을 경우 'R'과 'G'를 같은 영역으로 포함시킨다.

정상인의 경우 (4)

**RRRBB**

**GGBBB**

**BBBRR**

**BBRRR**

**RRRRR**

적록색약의 경우 (3)

**RRRBB**

**GGBBB**

**BBBRR**

**BBRRR**

**RRRRR**

### [JAVA]

package baekjoon;

import java.util.\*;

public class BOJ\_10026 {

static char[][] map;

static boolean[][] c;

static int n;

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

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

static int cnt\_a;

static int cnt\_b;

public static void dfs(int x, int y, char color) {

c[x][y] = true;

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

int nx = x + dx[i];

int ny = y + dy[i];

if(nx >= 0 && ny >= 0 && nx < n && ny < n) {

if(map[nx][ny] == color && c[nx][ny] == false) {

dfs(nx, ny, color);

}

}

}

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

n = sc.nextInt();

map = new char[n][n];

c = new boolean[n][n];

cnt\_a = 0;

cnt\_b = 0;

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

String s = sc.next();

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

map[i][j] = s.charAt(j);

}

}

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

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

if(c[i][j] == false) {

cnt\_a++;

dfs(i, j, map[i][j]);

}

}

}

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

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

c[i][j] = false;

if(map[i][j] == 'R') {

map[i][j] = 'G';

}

}

}

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

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

if(c[i][j] == false) {

cnt\_b++;

dfs(i, j, map[i][j]);

}

}

}

System.out.println(cnt\_a + " " + cnt\_b);

}

}

### 문제링크 >

<https://www.acmicpc.net/problem/10026>

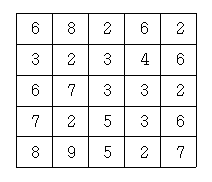
## [**[백준] 2468번 - 안전 영역**](https://dheldh77.tistory.com/entry/%EB%B0%B1%EC%A4%80-2468%EB%B2%88-%EC%95%88%EC%A0%84-%EC%98%81%EC%97%AD)

2020. 1. 11. 23:50

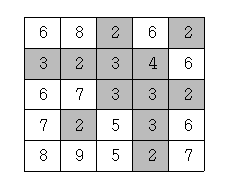
### 문제 >

재난방재청에서는 많은 비가 내리는 장마철에 대비해서 다음과 같은 일을 계획하고 있다. 먼저 어떤 지역의 높이 정보를 파악한다. 그 다음에 그 지역에 많은 비가 내렸을 때 물에 잠기지 않는 안전한 영역이 최대로 몇 개가 만들어 지는 지를 조사하려고 한다. 이때, 문제를 간단하게 하기 위하여, 장마철에 내리는 비의 양에 따라 일정한 높이 이하의 모든 지점은 물에 잠긴다고 가정한다.

어떤 지역의 높이 정보는 행과 열의 크기가 각각 N인 2차원 배열 형태로 주어지며 배열의 각 원소는 해당 지점의 높이를 표시하는 자연수이다. 예를 들어, 다음은 N=5인 지역의 높이 정보이다.

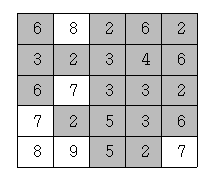


이제 위와 같은 지역에 많은 비가 내려서 높이가 4 이하인 모든 지점이 물에 잠겼다고 하자. 이 경우에 물에 잠기는 지점을 회색으로 표시하면 다음과 같다.



물에 잠기지 않는 안전한 영역이라 함은 물에 잠기지 않는 지점들이 위, 아래, 오른쪽 혹은 왼쪽으로 인접해 있으며 그 크기가 최대인 영역을 말한다. 위의 경우에서 물에 잠기지 않는 안전한 영역은 5개가 된다(꼭짓점으로만 붙어 있는 두 지점은 인접하지 않는다고 취급한다).

또한 위와 같은 지역에서 높이가 6이하인 지점을 모두 잠기게 만드는 많은 비가 내리면 물에 잠기지 않는 안전한 영역은 아래 그림에서와 같이 네 개가 됨을 확인할 수 있다.



이와 같이 장마철에 내리는 비의 양에 따라서 물에 잠기지 않는 안전한 영역의 개수는 다르게 된다. 위의 예와 같은 지역에서 내리는 비의 양에 따른 모든 경우를 다 조사해 보면 물에 잠기지 않는 안전한 영역의 개수 중에서 최대인 경우는 5임을 알 수 있다.

어떤 지역의 높이 정보가 주어졌을 때, 장마철에 물에 잠기지 않는 안전한 영역의 최대 개수를 계산하는 프로그램을 작성하시오.

### 입력 >

첫째 줄에는 어떤 지역을 나타내는 2차원 배열의 행과 열의 개수를 나타내는 수 N이 입력된다. N은 2 이상 100 이하의 정수이다. 둘째 줄부터 N개의 각 줄에는 2차원 배열의 첫 번째 행부터 N번째 행까지 순서대로 한 행씩 높이 정보가 입력된다. 각 줄에는 각 행의 첫 번째 열부터 N번째 열까지 N개의 높이 정보를 나타내는 자연수가 빈 칸을 사이에 두고 입력된다. 높이는 1이상 100 이하의 정수이다.

### 출력 >

첫째 줄에 장마철에 물에 잠기지 않는 안전한 영역의 최대 개수를 출력한다.

### 해결방법 >

**깊이 우선 탐색(DFS : Depth First Search)**를 사용해 문제 해결

가중치가 있는 그래프에서 임계값을 넘는 간선들로만 이루어진 연결의 최대 개수를 구하는 문제이다.

이 때, 임계값은 1부터 원소 중 가장 큰 값이다.

### [JAVA]

package baekjoon;

import java.util.\*;

public class BOJ\_2486 {

static boolean[][] visit;

static int[][] map;

static int n;

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

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

static int max = 0;

static int cnt = 0;

static int level;

public static void dfs(int x, int y) {

visit[x][y] = true;

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

int nx = x + dx[i];

int ny = y + dy[i];

if(nx >= 0 && ny >= 0 && nx < n && ny < n) {

if(visit[nx][ny] == false && map[nx][ny] >= level) {

dfs(nx, ny);

}

}

}

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

n = sc.nextInt();

map = new int[n][n];

visit = new boolean[n][n];

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

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

map[i][j] = sc.nextInt();

max = (max > map[i][j]) ? max : map[i][j];

}

}

for(int k = 1; k <= max; k++) {

level = k;

int tmp = 0;

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

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

visit[i][j] = false;

}

}

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

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

if(visit[i][j] == false && map[i][j] >= level) {

tmp++;

dfs(i, j);

}

}

}

cnt = (cnt > tmp) ? cnt : tmp;

}

System.out.println(cnt);

}

}

### 문제링크 >

<https://www.acmicpc.net/problem/2468>

## [**[백준] 11724번 - 연결 요소의 개수**](https://dheldh77.tistory.com/entry/%EB%B0%B1%EC%A4%80-11724%EB%B2%88-%EC%97%B0%EA%B2%B0-%EC%9A%94%EC%86%8C%EC%9D%98-%EA%B0%9C%EC%88%98)

2020. 1. 11. 23:25

### 문제 >

방향 없는 그래프가 주어졌을 때, 연결 요소 (Connected Component)의 개수를 구하는 프로그램을 작성하시오.

### 입력 >

첫째 줄에 정점의 개수 N과 간선의 개수 M이 주어진다. (1 ≤ N ≤ 1,000, 0 ≤ M ≤ N×(N-1)/2) 둘째 줄부터 M개의 줄에 간선의 양 끝점 u와 v가 주어진다. (1 ≤ u, v ≤ N, u ≠ v) 같은 간선은 한 번만 주어진다.

### 출력 >

첫째 줄에 연결 요소의 개수를 출력한다.

### 해결방법 >

**깊이 우선 탐색(DFS : Depth First Search) 알고리즘**으로 문제해결.

시작 노드부터 인접한 모든 노드들을 방문하면 카운트해준다. 이 때, 노드에 간선이 하나도 연결되어 있지 않는 상황도 고려해준다.

### [JAVA]

package baekjoon;

import java.util.\*;

public class BOJ\_11724 {

static int[][] map;

static boolean[] c;

static int n;

public static void dfs(int x) {

c[x] = true;

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

if(map[x][i] == 1 && c[i] == false) {

dfs(i);

}

}

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

n = sc.nextInt();

map = new int[n][n];

c = new boolean[n];

int cnt = 0;

int e = sc.nextInt();

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

int x = sc.nextInt();

int y = sc.nextInt();

map[x-1][y-1] = 1;

map[y-1][x-1] = 1;

}

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

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

if(map[i][j] == 1 && c[j] == false) {

cnt++;

dfs(j);

}

}

}

//노드에 연결된 간선이 하나도 없을 경우

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

int sum = 0;

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

sum += map[i][j];

}

if(sum == 0)

cnt++;

}

System.out.print(cnt);

}

}

### 문제링크 >

<https://www.acmicpc.net/problem/11724>

## [**[백준] 11403번 - 경로 찾기**](https://dheldh77.tistory.com/entry/%EB%B0%B1%EC%A4%80-11403%EB%B2%88-%EA%B2%BD%EB%A1%9C-%EC%B0%BE%EA%B8%B0)

2020. 1. 11. 00:44

### 문제 >

가중치 없는 방향 그래프 G가 주어졌을 때, 모든 정점 (i, j)에 대해서, i에서 j로 가는 경로가 있는지 없는지 구하는 프로그램을 작성하시오.

### 입력 >

첫째 줄에 정점의 개수 N (1 ≤ N ≤ 100)이 주어진다. 둘째 줄부터 N개 줄에는 그래프의 인접 행렬이 주어진다. i번째 줄의 j번째 숫자가 1인 경우에는 i에서 j로 가는 간선이 존재한다는 뜻이고, 0인 경우는 없다는 뜻이다. i번째 줄의 i번째 숫자는 항상 0이다.

### 출력 >

총 N개의 줄에 걸쳐서 문제의 정답을 인접행렬 형식으로 출력한다. 정점 i에서 j로 가는 경로가 있으면 i번째 줄의 j번째 숫자를 1로, 없으면 0으로 출력해야 한다.

### 해결방법 >

**깊이 우선 탐색(DFS : Depth First Search) 알고리즘**으로 문제해결.

노드 i에서 단말 노드 사이의 모든 노드들은

노드 i에서 그 노드로 가는 경로가 존재하는 노드이므로

각 노드를 탐색 시마다 경로가 있는 노드로 체크해준다.

### [JAVA]

package baekjoon;

import java.util.\*;

public class Main {

static int[][] map;

static boolean[] c;

static int n;

static int[][] ans;

public static void dfs(int x, int y) {

c[y] = true;

// x에 y로 이동하는 경로가 존재.

ans[x][y] = 1;

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

if(map[y][i] == 1 && c[i] == false) {

dfs(x, i);

}

}

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

n = sc.nextInt();

map = new int[n][n];

ans = new int[n][n];

c = new boolean[n];

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

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

map[i][j] = sc.nextInt();

}

}

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

// i 마다 경로를 탐색해야하므로 방문한 노드 초기화

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

c[j] = false;

}

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

if(map[i][j] == 1 && c[j] == false) {

dfs(i, j);

}

}

}

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

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

System.out.print(ans[i][j] + " ");

}

System.out.println();

}

}

}

### 문제링크 >

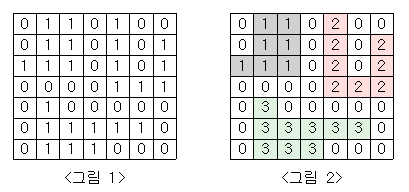
<https://www.acmicpc.net/problem/11403>

## [**[백준] 2667번 - 단지번호붙이기**](https://dheldh77.tistory.com/entry/%EB%B0%B1%EC%A4%80-2667%EB%B2%88-%EB%8B%A8%EC%A7%80%EB%B2%88%ED%98%B8%EB%B6%99%EC%9D%B4%EA%B8%B0)

2020. 1. 10. 18:04

### 문제 >

<그림 1>과 같이 정사각형 모양의 지도가 있다. 1은 집이 있는 곳을, 0은 집이 없는 곳을 나타낸다. 철수는 이 지도를 가지고 연결된 집들의 모임인 단지를 정의하고, 단지에 번호를 붙이려 한다. 여기서 연결되었다는 것은 어떤 집이 좌우, 혹은 아래위로 다른 집이 있는 경우를 말한다. 대각선상에 집이 있는 경우는 연결된 것이 아니다. <그림 2>는 <그림 1>을 단지별로 번호를 붙인 것이다. 지도를 입력하여 단지수를 출력하고, 각 단지에 속하는 집의 수를 오름차순으로 정렬하여 출력하는 프로그램을 작성하시오.



### 입력 >

첫 번째 줄에는 지도의 크기 N(정사각형이므로 가로와 세로의 크기는 같으며 5≤N≤25)이 입력되고, 그 다음 N줄에는 각각 N개의 자료(0혹은 1)가 입력된다.

### 출력 >

첫 번째 줄에는 총 단지수를 출력하시오. 그리고 각 단지내 집의 수를 오름차순으로 정렬하여 한 줄에 하나씩 출력하시오.

### 해결방법 >

**깊이 우선 탐색(DFS : Depth First Search) 알고리즘**으로 문제해결.

2차원 배열을 만들고 (0,0)부터 시작해서 각 원소에 인접한 원소 중 값이 1이면서 방문하지 않은 원소를 탐색.

### [JAVA]

package baekjoon;

import java.util.\*;

public class Main{

static int[][] map;

static boolean[][] c;

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

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

static int id;

static ArrayList list;

static int n;

public static void dfs(int x, int y) {

c[x][y] = true;

list.set(id, (int)list.get(id)+1);

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

int nx = x + dx[i];

int ny = y + dy[i];

if(nx >= 0 && ny >= 0 && nx < n && ny < n) {

if(map[nx][ny] == 1 && c[nx][ny] == false)

dfs(nx,ny);

}

}

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

n = sc.nextInt();

map = new int[n][n];

c = new boolean[n][n];

id = 0;

list = new ArrayList();

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

String s = sc.next();

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

map[i][j] = s.charAt(j) - '0';

}

}

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

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

if(map[i][j] == 1 && c[i][j] == false) {

list.add(0);

dfs(i,j);

id++;

}

}

}

Collections.sort(list);

System.out.println(id);

for(int i = 0; i < list.size(); i++) {

System.out.println(list.get(i));

}

}

}

### 문제링크 >

<https://www.acmicpc.net/problem/2667>

## [**[백준] 2606번 - 바이러스**](https://dheldh77.tistory.com/entry/%EB%B0%B1%EC%A4%80-2606%EB%B2%88-%EB%B0%94%EC%9D%B4%EB%9F%AC%EC%8A%A4)

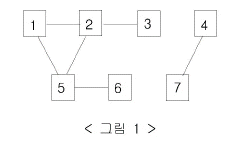
2020. 1. 10. 17:38

[BaekJoon](https://dheldh77.tistory.com/tag/BaekJoon), [DFS](https://dheldh77.tistory.com/tag/DFS), [알고리즘](https://dheldh77.tistory.com/tag/%EC%95%8C%EA%B3%A0%EB%A6%AC%EC%A6%98), [유니온파인드](https://dheldh77.tistory.com/tag/%EC%9C%A0%EB%8B%88%EC%98%A8%ED%8C%8C%EC%9D%B8%EB%93%9C)

### 문제 >

신종 바이러스인 웜 바이러스는 네트워크를 통해 전파된다. 한 컴퓨터가 웜 바이러스에 걸리면 그 컴퓨터와 네트워크 상에서 연결되어 있는 모든 컴퓨터는 웜 바이러스에 걸리게 된다.

예를 들어 7대의 컴퓨터가 <그림 1>과 같이 네트워크 상에서 연결되어 있다고 하자. 1번 컴퓨터가 웜 바이러스에 걸리면 웜 바이러스는 2번과 5번 컴퓨터를 거쳐 3번과 6번 컴퓨터까지 전파되어 2, 3, 5, 6 네 대의 컴퓨터는 웜 바이러스에 걸리게 된다. 하지만 4번과 7번 컴퓨터는 1번 컴퓨터와 네트워크상에서 연결되어 있지 않기 때문에 영향을 받지 않는다.



어느 날 1번 컴퓨터가 웜 바이러스에 걸렸다. 컴퓨터의 수와 네트워크 상에서 서로 연결되어 있는 정보가 주어질 때, 1번 컴퓨터를 통해 웜 바이러스에 걸리게 되는 컴퓨터의 수를 출력하는 프로그램을 작성하시오.

### 입력 >

첫째 줄에는 컴퓨터의 수가 주어진다. 컴퓨터의 수는 100 이하이고 각 컴퓨터에는 1번 부터 차례대로 번호가 매겨진다. 둘째 줄에는 네트워크 상에서 직접 연결되어 있는 컴퓨터 쌍의 수가 주어진다. 이어서 그 수만큼 한 줄에 한 쌍씩 네트워크 상에서 직접 연결되어 있는 컴퓨터의 번호 쌍이 주어진다.

### 출력 >

1번 컴퓨터가 웜 바이러스에 걸렸을 때, 1번 컴퓨터를 통해 웜 바이러스에 걸리게 되는 컴퓨터의 수를 첫째 줄에 출력한다.

### 해결방법 >

인접행렬을 구해 **깊이 우선 탐색(DFS : Depth First Search) 알고리즘**으로 문제해결.

**인접행렬?**

인접행렬이란 그래프가 주어졌을 때, 각 노드와의 연결관계를 이차원 배열로 나타낸 것이다.

인접행렬을 a[n][m]이라고 할 때,

a[i][j]는 노드 i에서 노드 j로가는 간선을 의미하며 간선이 있을 경우 1, 없을 경우 0으로 나타낸다.

만약, 그래프가 가중치 그래프라면 1대신 가중치 w를 넣어준다.

[2020.04.05]

유니온파인드를 사용해서 해결.

### [JAVA]

package baekjoon;

import java.util.Scanner;

public class Main {

static boolean c[];

static int map[][];

static int n;

static int cnt = 0;

public static void dfs(int x) {

c[x] = true;

for(int i = 1; i <= n; i++) {

if(map[x][i] == 1 && c[i] == false) {

dfs(i);

cnt++;

}

}

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

n = sc.nextInt();

map = new int[n + 1][n + 1];

c = new boolean[n + 1];

int v = sc.nextInt();

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

int a = sc.nextInt();

int b = sc.nextInt();

map[a][b] = 1;

map[b][a] = 1;

}

dfs(1);

System.out.println(cnt);

}

}

package baekjoon;

import java.util.\*;

import java.io.\*;

public class BOJ\_2606 {

static int[] parent;

public static int find(int x) {

if(x == parent[x]) return x;

int y = find(parent[x]);

parent[x] = y;

return y;

}

public static void union(int x, int y) {

x = find(x);

y = find(y);

if(x != y) parent[y] = x;

}

public static void main(String[] args) throws IOException {

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

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

int v = Integer.parseInt(tk.nextToken());

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

int e = Integer.parseInt(tk.nextToken());

int cnt = 0;

parent = new int[v + 1];

for(int i = 1; i <= v; i++) {

parent[i] = i;

}

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

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

int x = Integer.parseInt(tk.nextToken());

int y = Integer.parseInt(tk.nextToken());

union(x, y);

}

for(int i = 2; i <= v; i++) if(find(1) == find(i)) cnt++;

System.out.println(cnt);

}

}

### 문제링크 >

<https://www.acmicpc.net/problem/2606>