1. Hash 문제

두 문자열 비교하여 한쪽에만 있고 다른 한쪽에 없는 것 찾기

HashMap 으로 key를 해당 값으로 하고 value를 카운터로 사용.

처음 한쪽의 값들을 모두 카운트 1로 put 하고,

다른 한쪽의 값들을 map에서 get 하여 넣는다. 이때, get 한 결과가 없으면 0.

### 문제 > 1260 – DFS 와 BFS

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>

그래프를 완전탐색

1. DFS

- 한 방향으로 쭉 가다가 막히면 가장 가까운 갈림길로 돌아와 다시 탐색

- 스택이나 재귀함수 사용

- 무한루프가 될 가능성이 있기 때문에 방문여부를 검사해야 함

- 복잡도

- 인접 리스트의 경우 : O(N+E)

- 인접 행렬의 경우 : O(N^2)

(N : 노드의 개수 , E : 간선의 개수

2. BFS

- 시작점을 기준으로 인접한 노드를 먼저 탐색해나가는 방법

- 두 노드 사이의 최단경로 혹은 임의의 경로를 찾고 싶을 때 이용

- 큐 사용

- 무한루프가 될 가능성이 있기 때문에 방문여부를 검사해야 함

- 복잡도

- 깊이 우선 탐색과 다를바 없음

문제 해석

​

이 문제를 풀기위해 이해해야 할 내용은 다음과 같습니다.

​

**목표**

그래프를 DFS와 BFS를 이용하여 탐색하라.

​

**규칙**

1. 방문할 수 있는 정점이 여러개인 경우 번호가 작은 것부터 방문한다.

2. 간선은 양방향이다

​

**입력**

정점개수(N) 간선개수(M) 탐색시작 정점번호(V)

(M개의 줄 동안) 연결된 두 정점 번호

​

**출력**

DFS 수행 결과

BFS 수행 결과

**[출처]** [[자바] 백준 1260번 DFS와 BFS](https://blog.naver.com/lm040466/221787478911)|**작성자** [으니](https://blog.naver.com/lm040466)

문제 풀이

​

제목을 보자마자 DFS와 BFS를 사용하는 문제라고 생각하시면 됩니다!

하지만 각각 어떻게 구현할 지를 고민하셔야 되는데요.

​

**DFS**는 Depth-First Search로 깊이 우선 탐색입니다.

한 우물만 판다 라고 생각하시면 될 것같아요.

​

**BFS**는 Breadth-First Search로 너비 우선 탐색입니다.

같은 레벨에 있는 정점들은 같이 탐색하는 알고리즘이에요.

**[출처]** [[자바] 백준 1260번 DFS와 BFS](https://blog.naver.com/lm040466/221787478911)|**작성자** [으니](https://blog.naver.com/lm040466)

알고리즘을 공부하기 위해 봤던 블로그를 보면

DFS는 Stack(FILO)으로, BFS는 Queue(FIFO)를 이용해서 구현했는데요.

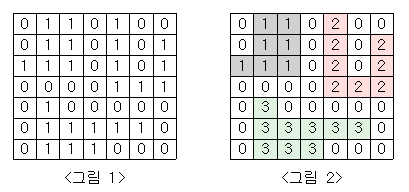
저는 동일하게 BFS는 Queue를 이용해서 구현했지만,

DFS는 Stack을 사용하지 않고, 재귀를 통해 구현했습니다!

**[출처]** [[자바] 백준 1260번 DFS와 BFS](https://blog.naver.com/lm040466/221787478911)|**작성자** [으니](https://blog.naver.com/lm040466)

**문제 > 2667**

<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.

11403

**문제 > 11403**

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.

11724 : 연결 요소의 개수

### 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.

**문제링크 >**

<https://www.acmicpc.net/problem/11724>

### 문제 > 2217-로프

N(1≤N≤100,000)개의 로프가 있다. 이 로프를 이용하여 이런 저런 물체를 들어올릴 수 있다. 각각의 로프는 그 굵기나 길이가 다르기 때문에 들 수 있는 물체의 중량이 서로 다를 수도 있다.

하지만 여러 개의 로프를 병렬로 연결하면 각각의 로프에 걸리는 중량을 나눌 수 있다. k개의 로프를 사용하여 중량이 w인 물체를 들어올릴 때, 각각의 로프에는 모두 고르게 w/k 만큼의 중량이 걸리게 된다.

각 로프들에 대한 정보가 주어졌을 때, 이 로프들을 이용하여 들어올릴 수 있는 물체의 최대 중량을 구해내는 프로그램을 작성하시오. 모든 로프를 사용해야 할 필요는 없으며, 임의로 몇 개의 로프를 골라서 사용해도 된다. 단, 각각의 로프는 한 개씩만 존재한다.

### 입력 >

첫째 줄에 정수 N이 주어진다. 다음 N개의 줄에는 각 로프가 버틸 수 있는 최대 중량이 주어진다. 이 값은 10,000을 넘지 않는 정수이다.

### 출력 >

첫째 줄에 답을 출력한다.

### 해결방법 >

**그리디 알고리즘**으로 문제를 해결. 가장 낮은 중량부터 사용해 최대 중량을 계산한다.

 최대 중량 = 사용한 로프 중 가장 낮은 중량 \* 사용한 로프의 개수이므로,

 로프의 각 중량들을 오름차순으로 정렬했을 때(w[] = {w1, w2, w3, ... , wn} // w1 < w2 < w3 < ... <wn)

weight = w[i] \* [n - i] 중에 최대값을 구한다.

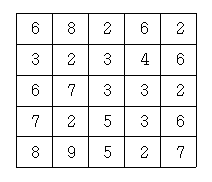
### 문제링크 >

<https://www.acmicpc.net/problem/2217>

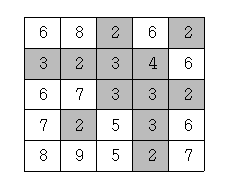
**문제 > 2468-안전영역**

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.

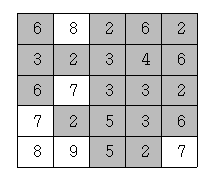


이제 위와 같은 지역에 많은 비가 내려서 높이가 4 이하인 모든 지점이 물에 잠겼다고 하자. 이 경우에 물에 잠기는 지점을 회색으로 표시하면 다음과 같다.



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>

### 문제 > 10026 – 적록색약

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>

### 문제 > 2875 – 대회 or 인턴

백준대학교에서는 대회에 나갈 때 2명의 여학생과 1명의 남학생이 팀을 결성해서 나가는 것이 원칙이다. (왜인지는 총장님께 여쭈어보는 것이 좋겠다.)

백준대학교는 뛰어난 인재들이 많아 올해에도 N명의 여학생과 M명의 남학생이 팀원을 찾고 있다.

그런데 올해에는 대회에 참여하려는 학생들 중 K명을 반드시 인턴쉽 프로그램에 참여하라는 학교의 방침이 생기게 되었다. 인턴쉽에 참여하는 학생은 대회에 참여하지 못한다.

백준대학교에서는 뛰어난 인재들이 많기 때문에, 많은 팀을 만드는 것이 최선이다.

여러분은 N명의 여학생과 M명의 남학생, K명의 인턴쉽에 참여해야하는 인원이 주어질 때 만들 수 있는 최대의 팀 수를 구하면 된다.

### 입력 >

첫째 줄에 N, M, K가 순서대로 주어진다. (0 ≤ M ≤ 100), (0 ≤ N ≤ 100), (0 ≤ K ≤ M+N),

### 출력 >

만들 수 있는 팀의 최댓값을 출력하면 된다.

### 해결방법 >

**그리디 알고리즘**으로 문제해결

대회에 참가하는 팀을 이루기 위해서는 다음 두 조건을 만족해야한다. 여학생 수(a), 남학생 수(b), 인턴 참가학생 수(c), 대회참가 팀 수(i)가  있을 때

1) 대회에 참가하는 팀의 수는 여학생 수의 절반을 넘지 못한다. (a >= 2i)

2) 총 학생 수는 중 대회에 참가하는 팀원을 제외한 인원이 인턴에 참가하는 인원 수보다 크거나 같아야한다. ((a+b) - 3i >= c)

대회에 참가하는 팀의 최대값은 남학생의 수와 같으므로 가장 큰 값부터 조건을 만족하는 지 탐색해나간다.

### [JAVA]

package baekjoon;

import java.util.\*;

public class BOJ\_2875{

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

int a = sc.nextInt();

int b = sc.nextInt();

int c = sc.nextInt();

int i;

for(i = b; i >= 0; i--) {

if((2 \* i) <= a && ((a + b) - 3 \* i) >= c)

break;

}

System.out.println(i);

}

}

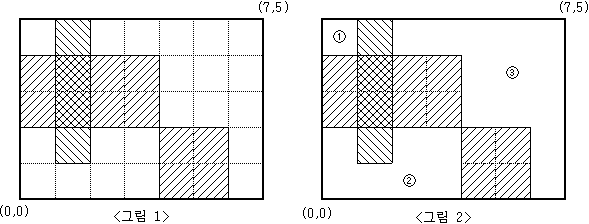
### 문제링크 >

<https://www.acmicpc.net/problem/2875>

**문제 > 2583 – 영역 구하기**

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.

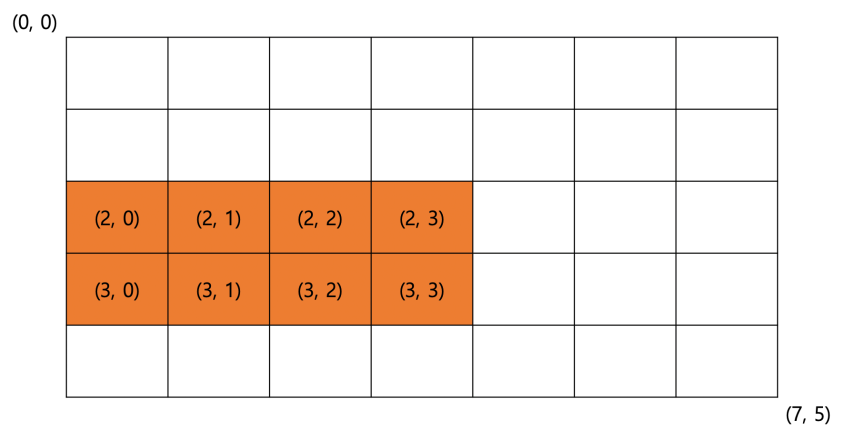
**출력 >**

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>

### 문제 > 10610 - 30

어느 날, 미르코는 우연히 길거리에서 양수 N을 보았다. 미르코는 30이란 수를 존경하기 때문에, 그는 길거리에서 찾은 수에 포함된 숫자들을 섞어 30의 배수가 되는 가장 큰 수를 만들고 싶어한다.

미르코를 도와 그가 만들고 싶어하는 수를 계산하는 프로그램을 작성하라.

### 입력 >

N을 입력받는다. N는 최대 10^5개의 숫자로 구성되어 있으며, 0으로 시작하지 않는다.

### 출력 >

미르코가 만들고 싶어하는 수가 존재한다면 그 수를 출력하라. 그 수가 존재하지 않는다면, -1을 출력하라.

### 해결방법 >

**그리디(DFS) 알고리즘**으로 해결이라고 하기엔 3의 배수가 가지는 규칙으로 해결했다는 말이 어울릴 것 같다

우선 3의 배수가 가지는 성질은

1) 배수의 끝자리는 1 ~ 9 까지 모두 나올 수 있다. (3 \* 7 = 21이므로 2**1**, 4**2**, 6**3**, 8**4**, ... , 18**9**)

2) 각자리 수의 합은 3이다. (3(**3**), 9(**9**), 12(**3**), 15(**6**), 18(**9**), 21(**3**), ...)

따라서 30의 배수가 되려면 2가지 조건을 만족해야한다

1) 10의 자리수는 어떤 수가 나와도 상관없지만 1의 자리수가 0이되야한다. 즉, 입력받은 데이터 중 0이 존재해야한다.

2) 각 자리수의 합이 3이되어야한다.

두가지 조건을 만족하는 가장 큰 수를 구한다.

### [JAVA]

package baekjoon;

import java.util.\*;

import java.io.\*;

public class BOJ\_10610 {

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

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

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

String input = br.readLine();

String output = new String();

ArrayList nums = new ArrayList();

int n = input.length();

int sum = 0;

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

int tmp = input.charAt(i) - '0';

sum += tmp;

nums.add(tmp);

}

Collections.sort(nums, Comparator.reverseOrder());

if(sum % 3 == 0 && (int)nums.get(n-1) == 0) {

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

bw.write(nums.get(i) + "");

}

}

else

bw.write("-1");

bw.flush();

}

}

### 문제링크 >

<https://www.acmicpc.net/problem/10610>

**문제 > 1987 - 알파벳**

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 >**

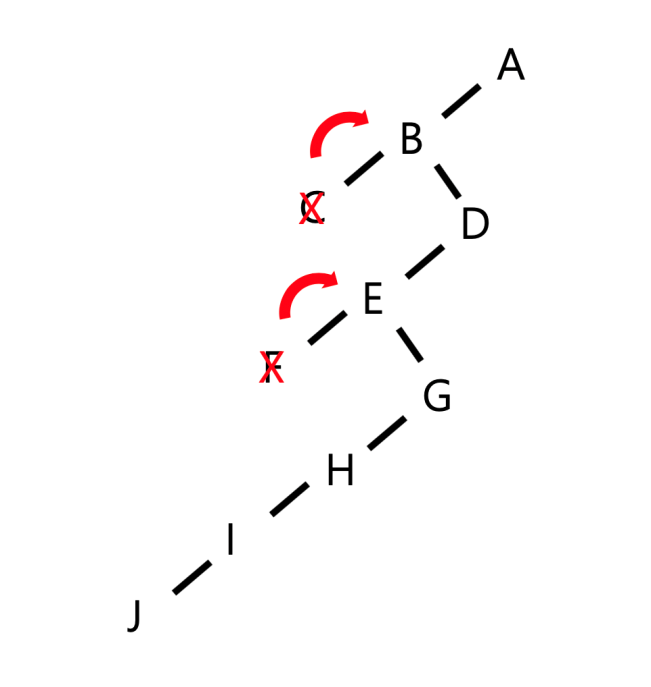
첫째 줄에 R과 C가 빈칸을 사이에 두고 주어진다. (1<=R,C<=20) 둘째 줄부터 R개의 줄에 걸쳐서 보드에 적혀 있는 C개의 대문자 알파벳들이 빈칸 없이 주어진다.

**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] 1260번 - DFS와 BFS**](https://dheldh77.tistory.com/entry/%EB%B0%B1%EC%A4%80-1260%EB%B2%88-DFS%EC%99%80-BFS)

2020. 3. 5. 11:11

### 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 >

정점과 간선이 입력으로 주어졌을 때, 정점과 간선의 관계를 나타내는 인접행렬을 만들고

너비우선탐색(BFS), 깊이우선탐색(DFS)를 활용해 각 각의 노드의 방문 순서를 출력한다.

### [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);

}

}