

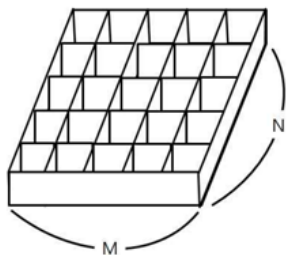
7576번 제출 맞은 사람 쏏코딩 풀이 풀이 작성 풀이 요청 재채점/수정 문제 추천 채점 현황 내 소스 강의▼ 질문 검색
질문 작성

토마토 풀이

시간 제한	메모리 제한	제출	정답	맞은 사람	정답 비율
1 초	128 MB	23352	6308	4064	24.887%

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.

문제유형 : 그래프, 탐색

접근 방법 :

시간초과가 난 알고리즘

1) 보통 BFS에서 flooding을 할때, 시작점이 한 곳이었다. 이 문제는 그와 다르게 flooding 시작점이 여러 곳이었다. 그래서 나는 예전에 풀었던 치즈 문제를 바탕으로 flooding 포인트가 아닌 외부에서 탐색을 하였다. DFS를 통해 빈 칸 전부를 탐색하되 근처에 flooding point 가 있으면 그 곳도 flooding point로 바꾸는 알고리즘이었다. 하지만 예외처리가 많았다.

```
package tomato;

import java.io.BufferedReader;
import java.io.InputStreamReader;
public class Main {
```

```

public static BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
public static int [][] tmt;
public static boolean [][] visit;
public static int n , m;
public static int dx[] = {0,1,0,-1};
public static int dy[] = {1,0,-1,0};
public static boolean change;
public static int cnt=0;
public static void main(String args[])throws Exception{
    getInput();
    do{
        find();
    }while(change);
    for(int i=0; i<n ; i++){
        for(boolean a : visit[i])
            if(a){
                System.out.println(-1);
                return;
            }
        else{
            System.out.println(cnt);
            return;
        }
    }
    in.close();
}

public static void find(){
    clean();
    for(int i =0; i< n; i++){
        for(int j =0; j<m; j++){
            if(visit[i][j]==false&&(tmt[i][j]==0||tmt[i][j]==2))DFS(i,j);
        }
    }
    test();
    if(change)cnt++;
}

public static boolean isSafe(int x, int y){
    return x>=0 && y>=0&& x<n && y<m;
}

public static void DFS(int x, int y){
    visit[x][y] = true;
    tmt[x][y] =2;
    for(int i =0; i<4; i++){
        int nx = x+dx[i];
        int ny = y+dy[i];
        if(tmt[x][y]==2&&isSafe(nx,ny)&&tmt[nx][ny]==1&&visit[nx][ny]==false){
            tmt[x][y]=1;
            change = true;
        }
    }
    for(int i =0; i<4; i++){
        int nx = x+dx[i];
        int ny = y+dy[i];
        if(isSafe(nx,ny)&&visit[nx][ny]==false&&tmt[nx][ny]==0){
            DFS(nx,ny);
        }
    }
}

public static void getInput()throws Exception{
    String line[] = in.readLine().split(" ");
    // m , n (6col ,4row)
    m = Integer.parseInt(line[0]);
    n = Integer.parseInt(line[1]);
    tmt = new int[n][m];
    visit = new boolean[n][m];
    for (int i =0; i< n; i++){
        line = in.readLine().split(" ");
        for(int j =0; j<m; j++){
            tmt[i][j] = Integer.parseInt(line[j]);
            visit[i][j] = false;
        }
    }
}

public static void clean(){
    change = false;
    // for(int i =0; i<n; i++){
    //     for(int j =0; j<m; j++){
    //         visit[i][j]=false;
    //     }
    // }
    // }
    visit = new boolean [n][m];
}

public static void test(){
    for (int i =0; i<n ; i++){
        for(int j=0; j<m; j++){
            System.out.print(tmt[i][j]+" ");
        } System.out.println();
    }
    for (int i =0; i<n ; i++){
        for(int j=0; j<m; j++){
            System.out.print(visit[i][j]+" ");
        } System.out.println();
    }
}

```

```
}  
}
```

두번째 풀이는 게시판을 참고하였다.

BFS를 개조해 queue에 넣을때 모든 플루딩 포인트를 넣고 BFS를 돌린다.

그다음 이것을 반복한다. 예외처리가 확 줄고, visit map을 사용 하지 않아도 된다.

```
import java.util.*;  
public class Main {  
  
    public static void main(String[] args) {  
        Scanner sc=new Scanner(System.in);  
        int w=sc.nextInt();  
        int h=sc.nextInt();  
        int map[][]=new int[w][h];  
        int totnum=w*h;  
        ArrayList<Integer> queue=new ArrayList<Integer>();  
        for(int i=0; i<h; i++) {  
            for(int j=0; j<w; j++) {  
                map[j][i]=sc.nextInt();  
                if(map[j][i]==1) {  
                    queue.add(10000*j+i);  
                }else if(map[j][i]==-1) {  
                    totnum--;  
                }  
            }  
        }  
  
        BFS(map, queue, w, h, totnum);  
    }  
  
    static void BFS(int map[][], ArrayList<Integer> queue, int w, int h, int totnum) {  
        int days=-1, tottom=0;  
        ArrayList<Integer> front=new ArrayList<Integer>();  
        int x, y;  
  
        while(!queue.isEmpty()) {  
            days++;  
            for(int tmp:queue) {  
                x=tmp/10000; y=tmp%10000;  
                tottom++;  
  
                if(x>0&&map[x-1][y]==0) {  
                    map[x-1][y]=1;  
                    front.add(tmp-10000);  
                }  
                if(x<w-1&&map[x+1][y]==0) {  
                    map[x+1][y]=1;  
                    front.add(tmp+10000);  
                }  
                if(y>0&&map[x][y-1]==0) {  
                    map[x][y-1]=1;  
                    front.add(tmp-1);  
                }  
                if(y<h-1&&map[x][y+1]==0) {  
                    map[x][y+1]=1;  
                    front.add(tmp+1);  
                }  
            }  
            queue.clear();  
            queue.addAll(front);  
            front.clear();  
        }  
        if(tottom==totnum) {  
            System.out.println(days);  
        }else if(tottom<totnum){  
            System.out.println(-1);  
        }  
    }  
}
```

문제를 풀지 못한 이유는 내가 아직 풀이량이 적기 때문에 좋은 알고리즘을 많이 접해 보지 않았다. 그렇기 때문에 매우 적은 알고리즘을 응용해 푼다. 문제를 많이 풀고 확실히 이해하고, 내것으로 만들면, 새로운 문제를 다방면에서 접근할 수 있을 것이다.

입력

첫 줄에는 상자의 크기를 나타내는 두 정수 M,N이 주어진다. M은 상자의 가로 칸의 수, N 은 상자의 세로 칸의 수를 나타낸다. 단, $2 \leq M, N \leq 1,000$ 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.

예제 입력

```
6 4
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 1
```

예제 출력

```
8
```

힌트

출처

Olympiad > [한국정보올림피아드시.도지역본선](#) > [지역본선 2013](#) > [고등부 1번](#)

알고리즘 분류

[보기](#)

Baekjoon Online Judge

- [소개](#)
- [도움말](#)
- [채점 환경](#)
- [뉴스](#)
- [생중계](#)
- [온라인 저지 Poll](#)
- [블로그](#)
- [라이선스](#)
- [캘린더](#)
- [Slack](#)
- [기부하기](#)
- [기능 추가 요청](#)
- [스페셜 저지 제작 프로젝트](#)

채점 현황

[채점 현황](#)

문제

- [문제](#)
- [단계별로 풀어보기](#)
- [알고리즘 분류](#)
- [새로 추가된 문제](#)
- [새로 추가된 영어 문제](#)
- [새로 추가된 문제 풀이](#)
- [문제 순위](#)
- [문제 번역하기](#)
- [랜덤 문제](#)
- [재채점 및 문제 수정](#)

출처

- [ACM-ICPC](#)
- [Olympiad](#)
- [한국정보올림피아드](#)
- [한국정보올림피아드시.도지역본선](#)
- [Coder's High](#)

ACM-ICPC

- [Regionals](#)
- [World Finals](#)
- [Daejeon Regionals](#)


- Africa and the Middle East Regionals
- Europe Regionals
- Latin America Regionals
- North America Regionals
- South Pacific Regionals

Coder's High

- 스코어보드
- 사진

앱


- iOS
- Android
- Command Line Submit Tool
- Contest Client



Baekjoon Online ...
5,331 개 좋아요

좋아요

회원님 외 친구 2명이 좋아합니다



트윗

© 2017 All Rights Reserved. 주식회사 스타트링크 | 서비스 약관 | 개인정보 보호 | 결제 이용 약관 | 도움말 | 광고 문의
사업자 등록 번호: 541-88-00682
대표자명: 최백준
주소: 서울시 서초구 강남대로 359 대우도씨에빛2 5층 502호
전화번호: 02-521-0487 (이메일로 연락 주세요)
이메일: contacts@startlink.io
통신판매신고번호: 제 2017-서울서초-2193 호

이 사이트는 ACM 또는 ACM-ICPC 대회와 무관하며, ACM으로부터 승인이나 지원을 받지 않고 있습니다.