G5

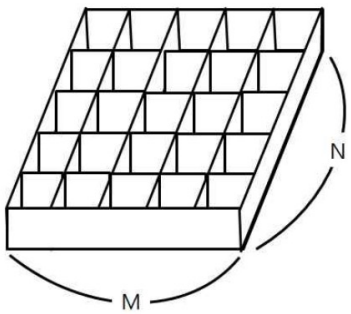
토마토 (Baekjoon 7576)

categories: 그래프 이론, 그래프 탐색, 너비 우선 탐색

시간 제한	메모리 제한	제출	정답	맞힌 사람	정답 비율
1 초	256 MB	143082	53764	33994	35.443%

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.

Woody K
Contact: woody35545@gmail.com
Github: https://github.com/woody35545
CNU, Computer Science and Engineering

G5

토마토 (Baekjoon 7576)

categories: 그래프 이론, 그래프 탐색, 너비 우선 탐색

입력

첫 줄에는 상자의 크기를 나타내는 두 정수 M,N이 주어진다.
M은 상자의 가로 칸의 수, N은 상자의 세로 칸의 수를 나타낸다.
(단, $2 \le M,N \le 1,000$ 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.
토마토가 하나 이상 있는 경우만 입력으로 주어진다.

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.

GitHub
Woody K
Contact: woody35545@gmail.com
Github: https://github.com/woody35545
CNU, Computer Science and Engineering

G5

토마토 (Baekjoon 7576)

categories: 그래프 이론, 그래프 탐색, 너비 우선 탐색

예제입력 1

```
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 1
```

예제출력 1

```
8
```

예제입력 2

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

예제출력 2

```
-1
```

예제입력 3

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

예제출력 3

```
6
```

예제입력 4

```
5 5
-1 1 0 0 0
0 -1 -1 -1 0
0 -1 -1 -1 0
0 -1 -1 -1 0
0 0 0 0 0
```

예제출력 4

```
14
```

예제입력 5

```
2 2
1 -1
-1 1
```

예제출력 5

```
0
```


Woody K
Contact: woody35545@gmail.com
Github: https://github.com/woody35545
CNU, Computer Science and Engineering

G5 토마토 (Baekjoon 7576)

categories: 그래프 이론, 그래프 탐색, 너비 우선 탐색

Answer(1/2)

```python
from collections import deque

# variables about input
R, C = -1, -1
graph = []

# notations
RIPE_TOMATO = 1
UNRIPE_TOMATO = 0
EMPTY_SPACE = -1

directions = [(0, 1), (0, -1), (1, 0), (-1, 0)]


def init_input():
    global R, C, graph

    # Reset Graph
    graph = []

    # init map size
    C, R = map(int, input().split(" "))

    # init map
    for i in range(R):
        graph.append(list(map(int, input().split(" "))))


def find_ripe_tomato():
    res = []
    for i in range(R):
        for j in range(C):
            if graph[i][j] == RIPE_TOMATO:
                res.append((i, j))

    return res

def check_all_tomato_ripe():
    for i in range(R):
        for j in range(C):
            if graph[i][j] == UNRIPE_TOMATO:
                return False

    return True


```

Woody K
Contact: woody35545@gmail.com
Github: https://github.com/woody35545
CNU, Computer Science and Engineering

G5

토마토 (Baekjoon 7576)

categories: 그래프 이론, 그래프 탐색, 너비 우선 탐색

Answer(2/2)

```python
def bfs(_graph, _start_node_list):
    global graph
    visited = [[False] * C for _ in range(R)]
    #queue = []
    queue = deque()
    cost = [[0] * C for _ in range(R)]  # step 당 1 cost 씩 올려서 최소기간을 계산할 때 사용
    cost_max = 0
    # 익은 토마토가 여러개인 경우 시작점이 여러개가 되므로 리스트로 받도록 함
    queue.extend(_start_node_list)

    while queue:
        # 현재 Queue에 있는 노드들의 자식들이 모두 추가되는 것을 한 스텝으로 묶어줌
        for _ in range(len(queue)):

            cx, cy = queue.popleft()

            if not visited[cx][cy]:
                visited[cx][cy] = True

                # check for neighbors
                for dir in directions:
                    nx, ny = cx + dir[0], cy + dir[1]

                    # check it's valid position
                    if 0 <= nx < R and 0 <= ny < C:
                        if graph[nx][ny] != EMPTY_SPACE and graph[nx][ny] != 1 and not visited[nx][ny]:
                            queue.append((nx, ny))
                            # 익은 토마토로 변경
                            graph[nx][ny] = 1
                            cost[nx][ny] = cost[cx][cy] + 1

                            if cost_max < cost[nx][ny]:
                                cost_max = cost[nx][ny]

    return cost, cost_max


def solve():
    init_input()
    start_position = find_ripe_tomato()
    cost, cost_max = bfs(graph, start_position)

    if check_all_tomato_ripe():

        print(cost_max)
    else:
        print(-1)

solve()
```


GitHub

Woody K

Contact: woody35545@gmail.com

Github: https://github.com/woody35545

CNU, Computer Science and Engineering