6문제를 풀어보았다.
백준 1012번: 유기농 배추
백준 1260번: DFS와 BFS
백준 2178번: 미로 탐색
백준 2606번: 바이러스
백준 2644번: 촌수계산
백준 7576번: 토마토

백준 1012번: 유기농 배추

유기농 배추 성공

시간 제한	메모리 제한	제출	정답	맞힌 사람	정답 비율
1 초	512 MB	94027	36084	24438	36.592%

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 인접해있는 것이다.

한나가 배추를 재배하는 땅은 고르지 못해서 배추를 군데군데 심어 놓았다. 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. 두 배추의 위치가 같은 경우는 없다.

출력

각 테스트 케이스에 대해 필요한 최소의 배추흰지렁이 마리 수를 출력한다.

https://www.acmicpc.net/problem/1012

아래 코드를 제출했더니 런타임 에러(RecursionError)가 발생했다. 최대 재귀 깊이를 초과한 것이 이유로 보인다.

```
def dfs(x, y):
    if x < 0 or x >= M or y < 0 or y >= N:
        return False
    if graph[y][x] == 1:
        graph[y][x] = 0
        dfs(x - 1, y)
        dfs(x + 1, y)
        dfs(x, y - 1)
        dfs(x, y + 1)
        return True
    return False

T = int(input())
for _ in range(T):
    M, N, K = map(int, input().split())
    graph = [[0] * M for _ in range(N)]

    for _ in range(K):
        i, j = map(int, input().split())
        graph[j][i] = 1

    result = 0
    for i in range(N):
        for j in range(M):
            if dfs(j, i) == True:
                result += 1

    print(result)
```

반례는 아래처럼 M, N, K가 최댓값일 때, 즉 모든 땅에 배추가 심어져 있을 때를 들 수 있다.
아래 코드를 실행하면 RecursionError: maximum recursion depth exceeded in comparison이 발생한다.

```
def dfs(x, y):
    if x < 0 or x >= M or y < 0 or y >= N:
        return False
    if graph[y][x] == 1:
        graph[y][x] = 0
        dfs(x - 1, y)
        dfs(x + 1, y)
        dfs(x, y - 1)
        dfs(x, y + 1)
        return True
    return False

M, N, K = 50, 50, 2500
graph = [[1] * M for _ in range(N)]

result = 0
for i in range(N):
    for j in range(M):
        if dfs(j, i) == True:
            result += 1

print(result)
```

RecursionError는 재귀와 관련된 에러입니다. 가장 많이 발생하는 이유는 Python이 정한 최대 재귀 깊이보다 재귀의 깊이가 더 깊어질 때입니다.

Python이 정한 최대 재귀 깊이는 sys.getrecursionlimit()을 이용해 확인할 수 있습니다. BOJ의 채점 서버에서 이 값은 1,000으로 되어 있습니다.

(중략)

DFS를 이용했다면 BFS로, 다이나믹 프로그래밍을 재귀로 구현했다면 반복문으로 구현하는 것 처럼 재귀를 사용하지 않게 구현하는 방법으로 해결할 수 있습니다.

https://help.acmicpc.net/judge/rte/RecursionError

재귀를 쓰지 않거나, sys.setrecursionlimit(n)을 사용해 채점 서버가 감당할 수 있는 한도 내에서 최대 재귀 깊이를 n으로 변경하면 해결된다고 한다.

(최대 재귀 깊이를 크게 변경하더라도, 재귀의 깊이가 채점 서버가 감당할 수 없는 정도로 깊어지면 Segmentation fault가 발생한다.)

속 편하게 재귀를 사용하는 DFS 대신 BFS로 변경하기로 했다. 난잡한 코드로 어찌저찌 채점 통과했다.

```
from collections import deque

def bfs(x, y):
    if graph[y][x] == 0:
        return False
    graph[y][x] = 0
    queue = deque()
    queue.append((x, y))
    while queue:
        x, y = queue.popleft()
        for i in range(4):
            nx = x + dx[i]
            ny = y + dy[i]
            if nx < 0 or nx >= M or ny < 0 or ny >= N:
                continue
            if graph[ny][nx] == 0:
                continue
            if graph[ny][nx] == 1:
                graph[ny][nx] = 0
                queue.append((nx, ny))
    return True

dx = [-1, 1, 0, 0]
dy = [0, 0, -1, 1]

T = int(input())
for _ in range(T):
    M, N, K = map(int, input().split())
    graph = [[0] * M for _ in range(N)]
    klist = []
    result = 0

    for _ in range(K):
        i, j = map(int, input().split())
        graph[j][i] = 1
        klist.append([i, j])

    for i, j in klist:
        if bfs(i, j) == True:
            result += 1

    print(result)
```

백준 1260번: DFS와 BFS

DFS와 BFS 성공

☆

시간 제한	메모리 제한	제출	정답	맞힌 사람	정답 비율
2 초	128 MB	161767	56986	33512	34.565%

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.

https://www.acmicpc.net/problem/1260

가볍게 복습 겸 선택한 문제였는데 생각보다 까다로웠다.
작은 인덱스부터 탐색하도록 하기 위해 sort()를 쓰느라 생략할 수 있을 것 같은 M번 반복문을 사용해서 그다지 만족스럽지 않은 코드다.

```python
N, M, V = map(int, input().split())
graph = [[] for _ in range(10001)]

for _ in range(M):
    i, j = map(int, input().split())
    graph[i].append(j)
    graph[j].append(i)

for i in range(M):
    if graph[i]:
        graph[i].sort()

# dfs
def dfs(graph, v, visited):
    visited[v] = True
    print(v, end=' ')
    for i in graph[v]:
        if not visited[i]:
            dfs(graph, i, visited)

visited = [False] * (N + 1)

dfs(graph, V, visited)

print()

# bfs
from collections import deque

def bfs(graph, start, visited):
    queue = deque([start])
    visited[start] = True
    while queue:
        v = queue.popleft()
        print(v, end=' ')
        for i in graph[v]:
            if not visited[i]:
                queue.append(i)
                visited[i] = True

visited = [False] * (N + 1)

bfs(graph, V, visited)
```

백준 2178번: 미로 탐색

미로 탐색 성공

시간 제한	메모리 제한	제출	정답	맞힌 사람	정답 비율
1 초	192 MB	109390	45219	28946	39.946%

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.

https://www.acmicpc.net/problem/2178

미로 탈출과 상당히 유사한 문제다.
('미로 탈출' 문제가 포함된 글: https://blog.naver.com/hyeonjun7/222620009176)

```
from collections import deque

def bfs(x, y):
    queue = deque()
    queue.append((x, y))
    while queue:
        x, y = queue.popleft()
        for i in range(4):
            nx = x + dx[i]
            ny = y + dy[i]
            if nx < 0 or nx >= n or ny < 0 or ny >= m:
                continue
            if graph[nx][ny] == 0:
                continue
            if graph[nx][ny] == 1:
                graph[nx][ny] = graph[x][y] + 1
                queue.append((nx, ny))
    return graph[n - 1][m - 1]

n, m = map(int, input().split())
graph = []
for _ in range(n):
    graph.append(list(map(int, input())))

dx = [-1, 1, 0 , 0]
dy = [0, 0, -1, 1]

print(bfs(0, 0))
```

백준 2606번: 바이러스

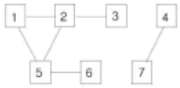
바이러스 성공

시간 제한	메모리 제한	제출	정답	맞힌 사람	정답 비율
1 초	128 MB	83990	39406	26824	45.515%

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.



< 그림 1 >

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.

https://www.acmicpc.net/problem/2606

아래 코드를 제출했더니 런타임 에러(IndexError)가 발생했다. 이유가 좀 특이한데, 전역변수 이름 때문이었다.

```
computer = int(input())
m = int(input())
graph = [[] for _ in range(101)]
count = -1

for _ in range(m):
    i, j = map(int, input().split())
    graph[i].append(j)
    graph[j].append(i)

for i in range(m):
    if graph[i]:
        graph[i].sort()

def dfs(graph, v, visited):
    global count
    visited[v] = True
    count += 1
    for i in graph[v]:
        if not visited[i]:
            dfs(graph, i, visited)

visited = [False] * 101

dfs(graph, 1, visited)

print(count)
```

기존의 count와 이름이 겹쳐서 그렇다고 한다.
(https://www.acmicpc.net/board/view/74831, https://en.cppreference.com/w/cpp/algorithm/count)

질문자와 언어가 다르긴 한데(질문자는 C++17), count 변수명을 cnt로 바꾸니 해결되어서 같은 이유겠거니 하고 생각했다.
아래는 수정한 정답 코드이다.

```
computer = int(input())
m = int(input())
graph = [[] for _ in range(101)]
cnt = -1

for _ in range(m):
    i, j = map(int, input().split())
    graph[i].append(j)
    graph[j].append(i)

def dfs(graph, v, visited):
    global cnt
    visited[v] = True
    cnt += 1
    for i in graph[v]:
        if not visited[i]:
            dfs(graph, i, visited)

visited = [False] * 101

dfs(graph, 1, visited)

print(cnt)
```

백준 2644번: 촌수계산

촌수계산

시간 제한	메모리 제한	제출	정답	맞힌 사람	정답 비율
1 초	128 MB	25744	12205	9245	46.711%

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.

https://www.acmicpc.net/problem/2644

서로 다른 두 사람의 촌수 계산을 두 노드 사이의 거리로 생각해도 된다.
촌수의 특성상 최단거리를 구할 필요가 없다. 자식이 부모를 낳을 수 없으니 간선이 양방향 없는 단방향뿐이고, 한 부모에게 여러 자식이 있을 수는 있어도 한 자식에게 여러 부모가 있을 수는 없다. 입양, 근친상간 등의 이유로 억지로 반례를 만들 수 있을 것 같긴 한데 생각하지 않기로 했다.
기존에 작성한 dfs 함수 코드에 촌수 파라미터 하나만 추가했다.

입력받은 문장(ex. 엄마의 아빠)에서 "엄마", "아빠", "형" 등의 키워드를 이용해 그 사람과 나의 촌수 관계를 출력하는 프로그램을 만들면 재밌겠다는 생각을 잠깐 했는데, 은근 고려할 게 많을 거 같다.

```
n = int(input())
person1, person2 = map(int, input().split())
m = int(input())
graph = [[] for _ in range(101)]

for _ in range(m):
    i, j = map(int, input().split())
    graph[i].append(j)
    graph[j].append(i)

def dfs(graph, v, visited, kinship):
    visited[v] = kinship + 1
    for i in graph[v]:
        if not visited[i]:
            dfs(graph, i, visited, visited[v])

visited = [False] * 101

dfs(graph, person1, visited, 0)

if visited[person2] - 1 != 0:
    print(visited[person2] - 1)
else:
    print(-1)
```

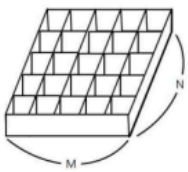
백준 7576번: 토마토

토마토 성공

시간 제한	메모리 제한	제출	정답	맞힌 사람	정답 비율
1 초	256 MB	106882	38269	24028	34.352%

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.

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.

https://www.acmicpc.net/problem/7576

최소 일수라는 말에서 BFS를 떠올렸다.
익은 토마토가 있는 좌표들을 찾아 큐에 삽입하고 상하좌우로 익지 않은 토마토가 있으면 해당 좌표를 큐에 삽입한다.
1이 있는 위치 주변으로 숫자가 1씩 커지면서 퍼져나간다고 이미지를 떠올렸더니 이해하기 쉬웠다.
주석처리된 부분을 해제하면 입/출력 그래프를 비교해볼 수 있다.

```
from collections import deque

m, n = map(int, input().split())
graph = [list(map(int,input().split())) for i in range(n)]

dx, dy = [-1, 1, 0, 0], [0, 0, -1, 1]
ans = 0
flag = 0

def bfs(x, y):
    queue = deque()
    for i in range(y):
        for j in range(x):
            if graph[i][j] == 1:
                queue.append((i, j))
    while queue:
        x, y = queue.popleft()
        for i in range(4):
            nx, ny = dx[i] + x, dy[i] + y
            if 0 <= nx < n and 0 <= ny < m and graph[nx][ny] == 0:
                graph[nx][ny] = graph[x][y] + 1
                queue.append((nx, ny))

bfs(m, n)

# for i in range(n):
#     for j in range(m):
#         print(graph[i][j], end=' ')
#     print()

for i in graph:
    for j in i:
        if j == 0:
            flag = 1
    ans = max(ans, max(i))

if flag == 1:
    print(-1)
else:
    print(ans - 1)
```

총평

나름 이해한 상태로 쓰고 있긴 하지만, 처음부터 코드를 작성하는 것이 아니라 강의에서 본 DFS, BFS 함수 코드를 계속 참고하여 짜깁기하다보니 이렇게 공부해도 되는 건지, 제대로 알고 있는 건지 스스로 의문이 들었다.
관련된 문제를 많이 풀어보면서 꾸준히 복습해야겠다.