[BOJ] 10026. 적록색약

Date	@March 24, 2025
Problem Link	https://www.acmicpc.net/problem/10026
Assignment Link	https://www.acmicpc.net/problem/27737

문제

10026. 적록색약

- NxN 크기의 그리드에서 R(빨강), G(초록), B(파랑) 중 하나로 색칠된 그림이 주어진다.
- 같은 색으로 상하좌우 인접한 칸들은 같은 구역으로 본다.

```
# 예제 입력
5
RRRBB
GGBBB
BBBRR
BBRRR
RRRRR

# 예제 출력
4 3
```

  - 적록색약이 아닌 사람은 R, G, B를 각각 구분해서 구역을 센다.
    - `R` 2개, `G` 1개, `B` 1개 → 총 4개
  - 적록색약(빨간색과 초록색의 차이를 거의 느끼지 못하는 사람)인 사람은 R과 G를 같은 색으로 간주하여 구역을 센다.
    - `R`과 `G`를 같은 색으로 간주 (`R`=`G`) → `RG` 2개, `B` 1개 → `총` 3개
- **두 경우의 구역 개수**를 구한다.

힌트

▼ 문제 유형 / 시간 복잡도

💡 **문제 유형**

BFS/DFS

2차원 배열의 상하좌우 인접 탐색은 대표적으로 BFS/DFS를 활용해서 해결할 수 있다.

▼ 아이디어

💡 **아이디어**

- 일반인 기준과 적록색약 기준, 두 가지 조건으로 **각각 한 번씩 구역 세기 작업을 수행한다.**
  - (1) 일반인 기준: `R`, `G`, `B`가 각각 다른 경우
  - (2) 적록색약 기준: `R`, `G`를 같은 색으로 보는 경우
    - 이를 위해 그리드에서 `G`를 `R`로 바꾸거나, 탐색 시 색상 비교 조건을 조정한다.
- 따라서 DFS/BFS 탐색 함수는 그대로 두고, **두 번 호출하여 다른 조건으로 구역 수를 세면 된다.**

▼ 시간 복잡도

⌛ **시간 복잡도**

- 전체 그리드는 NxN이고, **모든 칸을 한 번씩만 방문**하므로 탐색 자체는 $O(N \times N)$의 시간 복잡도가 소요된다.
- 각 칸에서 상하좌우 4방향을 보기 때문에4x$O(N^2)$의 시간복잡도가 소요된다.
- 두 조건에 대해 각각 DFS/BFS를 수행하므로 2번의 탐색이 진행되어 최종적으로 2x4x$O(N^2)$의 시간복잡도가 소요된다.
- 중간에 적록색약 처리를 위해 **그리드를 함께 탐색해서 값을 바꿔주는 과정**이 존재한다.
  이 경우도
  $O(N^2)$의 시간복잡도가 소요된다.
- 총 9x$O(N^2)$이 소요되며, 상수를 무시하면 **최종 시간복잡도는 $O(N^2)$**이다.
- N ≤ 100 이므로 최대 100x100=10,000 (**10^4**) 개의 노드를 탐색해야 한다.
  BFS/DFS를 사용하여
  $O(N^2)$의 시간복잡도로 해결이 가능하다. (**10^8보다 훨씬 작다.**)

정답

1. N을 입력받고 NxN 크기의 그리드를 저장한다.
2. BFS/DFS를 이용하여 같은 색의 영역을 찾는다.
   - 적록색약이 아닌 경우와 적록색약인 경우를 각각 계산해야 한다.
   - **모든 칸을 훑어보면서 아직 방문하지 않은 색이 있다면, 그 칸을 시작점으로 탐색을 시작한다.**
   - **탐색하면서 상하좌우로 같은 색인 곳을 찾아가고, 방문 표시를 남긴다. 이게 하나의 구역이 된다.**
3. 적록색약이 아닌 사람 기준으로 구역 탐색을 진행한다.
4. 적록색약 탐색을 위해 그리드를 변경한다.
5. 적록색약 기준으로 구역 탐색을 진행한다.

BFS/DFS에서 항상 두 조건을 확인한다.

- **(1) 주어진 배열 범위를 벗어나지는 않았는가?**
- **(2) 이미 방문된 곳은 아닌가? 현재 탐색중인 구역과 같은 색인가?**
  - 방문 배열을 별도로 만들수도 만들지 않을수도 있다.

구분	DFS (Depth-First Search)	BFS (Breadth-First Search)
탐색 방식	깊이 우선 (한 방향으로 끝까지 파고듦)	너비 우선 (가까운 곳부터 레벨별로 탐색)
자료구조	재귀 호출 or 스택	큐 사용
구현 방식	재귀 함수 사용 (Python에선 `sys.setrecursionlimit` 필요)	deque 이용 (`queue.append` , `queue.popleft`)
사용 시기	간단한 연결 확인, 백트래킹 문제	최단 거리 탐색, 레벨 순서 탐색
시간/공간	일반적으로 비슷 ($O(N^2)$)	일반적으로 비슷 ($O(N^2)$)
장점	구현 간단, 코드 짧음	무한루프 위험 적고 구조적임
단점	재귀 깊이 제한에 걸릴 수 있음	큐를 위한 메모리 필요

풀이 1 DFS

```
import sys
input = sys.stdin.readline
```

```
sys.setrecursionlimit(10000) # 파이썬 최대 재귀 깊이

N = int(input().strip())
grid = [list(input().strip()) for _ in range(N)]

# 방향 벡터
dx = [-1, 1, 0, 0]
dy = [0, 0, -1, 1]

def dfs(x, y, color, visited, grid):
    visited[x][y] = True # 현재 위치 방문 처리
    for i in range(4): # 상하좌우 탐색
        nx, ny = x + dx[i], y + dy[i] # 새로운 좌표
        # 새로운 위치가 범위 내에 있고, 방문하지 않았으며, 같은 색이면 탐색 진행
        if 0 <= nx < N and 0 <= ny < N and not visited[nx][ny]:
            if grid[nx][ny] == color:
                dfs(nx, ny, color, visited, grid) # 재귀 호출

def count_regions(grid, is_color_blind):
    visited = [[False] * N for _ in range(N)] # 방문 여부
    count = 0 # 구역 수

    for i in range(N):
        for j in range(N): # 모든 칸을 탐색
            if not visited[i][j]:  # 방문하지 않은 경우 DFS 탐색 시작
                count += 1
                color = grid[i][j] # 해당 색을 기준으로 탐색을 시작

                # 적록색약 모드라면 R과 G를 동일하게 처리
                if is_color_blind and color in "RG":
                    dfs(i, j, "R", visited, grid)
                else:
                    dfs(i, j, color, visited, grid)

    return count

# 적록색약이 아닌 경우
normal_count = count_regions(grid, is_color_blind=False)

# 적록색약인 경우 (R과 G를 동일하게 처리한 grid 생성)
for i in range(N):
    for j in range(N):
        if grid[i][j] == 'G':
            grid[i][j] = 'R'

color_blind_count = count_regions(grid, is_color_blind=True)

print(normal_count, color_blind_count)
```

제출 번호	아이디	문제	결과	메모리	시간	언어	코드 길이	제출한 시간
90985756	learntosurf	10026	맞았습니다!!	33016 KB	56 ms	Python 3 / 수정	1459 B	3분 전

- 방향벡터를 사용해서 상하좌우 이동

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

```
for i in range(4):
    nx = x + dx[i]
    ny = y + dy[i]
```

- DFS 동작 방식
  - 현재 위치 `(x, y)`를 방문 처리 (`vitisted[x][y]=True`)한다.
  - for문을 이용해 상하좌우 이동한다.
  - 이동할 곳이 범위 내에 있고, 방문하지 않았으며, 같은 색이면 DFS를 다시 호출한다. (재귀 함수 사용)
  - 끝까지 탐색한 후 되돌아와서 다른 방향을 탐색한다.

풀이 2 BFS

```
import sys
input = sys.stdin.readline
from collections import deque

N = int(input().strip())
grid = [list(input().strip()) for _ in range(N)]

# 방향 벡터
dx = [-1, 1, 0, 0]
dy = [0, 0, -1, 1]

def bfs(x, y, color, visited, grid):
    queue = deque([(x, y)]) # 큐 초기화 및 시작점 추가
    visited[x][y] = True # 방문 처리

    while queue:
        x, y = queue.popleft() # 큐에서 하나를 꺼냄
        for i in range(4): # 상하좌우 탐색
            nx, ny = x + dx[i], y + dy[i]
            # 새로운 위치가 범위 내에 있고, 방문하지 않았으며, 같은 색이면 탐색 진행
            if 0 <= nx < N and 0 <= ny < N and not visited[nx][ny]:
                if grid[nx][ny] == color:
                    visited  [nx][ny] = True # 방문 처리
                    queue.append((nx, ny)) # 큐에 추가

def count_regions(grid, is_color_blind):
    visited = [[False] * N for _ in range(N)]
    count = 0

    for i in range(N):
        for j in range(N):
            if not visited[i][j]:  # 방문하지 않은 경우 BFS 탐색 시작
                count += 1
                color = grid[i][j]

                # 적록색약 모드라면 R과 G를 동일하게 처리
                if is_color_blind and color in "RG":
                    bfs(i, j, "R", visited, grid)
                else:
                    bfs(i, j, color, visited, grid)

    return count
```

```
# 적록색약이 아닌 경우
normal_count = count_regions(grid, is_color_blind=False)

# 적록색약인 경우 (R과 G를 동일하게 처리한 grid 생성)
 for i in range(N):
   for j in range(N):
       if grid[i][j] == 'G':
           grid[i][j] = 'R'

color_blind_count = count_regions(grid, is_color_blind=True)
print(normal_count, color_blind_count)
```

제출 번호	아이디	문제	결과	메모리	시간	언어	코드 길이	제출한 시간
90986122	learntosurf	10026	맞았습니다!!	35016 KB	68 ms	Python 3 / 수정	1564 B	7초 전
90985756	learntosurf	10026	맞았습니다!!	33016 KB	56 ms	Python 3 / 수정	1459 B	7분 전

- BFS 동작 방식
  - 시작 위치를 `queue`에 추가하고 방문 처리한다.
  - `while queue:`를 사용해 큐가 빌 때까지 반복한다.

단계	큐에서 꺼낸 위치	탐색 후 큐에 추가된 위치	현재 큐 상태
1	(0,0)	(0,1)	[(0,1)]
2	(0,1)	(0,2)	[(0,2)]
3	(0,2)	-	[ ] (큐가 비어서 종료)

    - 탐색이 끝나면 BFS는 하나의 구역을 완전히 탐색한다.
  - 현재 위치 `(x,y)`를 꺼내고 상하좌우를 탐색한다.
  - 이동할 곳이 범위 내에 있고, 방문하지 않았으며, 같은 색이면 큐에 추가한다.
  - 큐에서 꺼낸 후 탐색하므로 가까운 곳부터 탐색된다.

과제

27737. 버섯농장