Chapter 09.
그래프 탐색 알고리즘
핵심 유형 문제풀이

핵심 유형 문제풀이 | 다양한 문제를 접하며 코딩 테스트에 익숙해지기

강사 나동빈

Chapter 09.
그래프 탐색 알고리즘

핵심 유형 문제풀이

단지번호붙이기: https://www.acmicpc.net/problem/2667

혼자 힘으로 풀어보기

문제 제목: 단지번호붙이기

문제 난이도: 중(Medium)

문제 유형: DFS, 연결 요소(Connected Component)

추천 풀이 시간: 50분

단지번호붙이기: https://www.acmicpc.net/problem/2667

문제 풀이 핵심 아이디어

- 맵의 크기는 $N \times N$ 크기의 **정사각형** 형태입니다.
- N은 최대 25이므로, DFS 혹은 BFS를 이용할 수 있습니다.

단지번호붙이기: https://www.acmicpc.net/problem/2667

문제 풀이 핵심 아이디어

- **연결 요소**란, 모든 두 꼭짓점 사이에 경로가 존재하는 그래프를 의미합니다.
- 본 문제는 이러한 연결 요소(connected component)를 찾는 문제입니다.
- 구체적으로 각 연결 요소에 포함된 원소의 개수를 계산하는 것이 요구됩니다.

0	1	1	0	2	0	0
0	1	1	0	2	0	2
1	1	1	0	2	0	2
0	0	0	0	2	2	2
0	3	0	0	0	0	0
0	3	3	3	3	3	0
0	3	3	3	0	0	0

문제 풀이 핵심 아이디어

- **연결 요소**의 개수를 세는 방법은 다음과 같습니다.

1. 방문하지 않은 노드를 만날 때마다 카운트(count)하고, **DFS를 호출**합니다.
2. DFS는 **해당 위치로부터 연결된(연결요소에 포함된)** 모든 노드를 방문 처리합니다.

단지번호붙이기: https://www.acmicpc.net/problem/2667

문제 풀이 핵심 아이디어

1. 방문하지 않은 노드를 만날 때마다 카운트(count)하고, **DFS를 호출**합니다.
2. DFS는 **해당 위치로부터 연결된(연결요소에 포함된)** 모든 노드를 방문 처리합니다.

0	1	1	0	2	0	0
0	1	1	0	2	0	2
1	1	1	0	2	0	2
0	0	0	0	2	2	2
0	3	0	0	0	0	0
0	3	3	3	3	3	0
0	3	3	3	0	0	0

○ : DFS가 시작되는 위치

단지번호붙이기: https://www.acmicpc.net/problem/2667

소스 코드

```python
import sys
# 빠른 입력 함수 사용
input = sys.stdin.readline

# 상, 하, 좌, 우 방향 정의
dx = [-1, 1, 0, 0]
dy = [0, 0, -1, 1]

def dfs(x, y):
    result = 1 # 원소의 개수 세기
    for i in range(4): # 인접한 위치 확인
        nx = x + dx[i]
        ny = y + dy[i]
        # 인접한 위치가 범위를 벗어나는 경우 무시
        if nx <= -1 or nx >= n or ny <= -1 or ny >= n:
            continue
        # 처음 방문하는 경우
        if graph[nx][ny] == 1:
            graph[nx][ny] = -1 # 방문 처리
            result += dfs(nx, ny)
    return result
```

```python
n = int(input()) # 지도의 크기(N)
graph = [[0] * n for _ in range(n)] # 2차원 맵
for i in range(n):
    row = input()
    for j in range(n):
        graph[i][j] = int(row[j])

# 단지의 수(연결 요소의 수) 계산
answer = []
for i in range(n):
    for j in range(n):
        # 처음 방문하는 경우 DFS 수행
        if graph[i][j] == 1:
            graph[i][j] = -1 # 방문 처리
            answer.append(dfs(i, j))

# 단지의 수와 정렬된 각 단지내의 집의 수 출력
answer.sort()
print(len(answer))
for i in range(len(answer)):
    print(answer[i])
```

적록색약: https://www.acmicpc.net/problem/10026

혼자 힘으로 풀어보기

문제 제목: 적록색약

문제 난이도: 중(Medium)

문제 유형: DFS, 연결 요소(Connected Component)

추천 풀이 시간: 50분

적록색약: https://www.acmicpc.net/problem/10026

문제 풀이 핵심 아이디어

- 맵의 크기는 $N \times N$ 크기의 **정사각형** 형태입니다.
- N은 최대 100이므로, DFS 혹은 BFS를 이용할 수 있습니다.

적록색약: https://www.acmicpc.net/problem/10026

문제 풀이 핵심 아이디어

- 연결 요소(connected component)의 개수를 세는 문제입니다.
- 초기 상태에서 **연결 요소의 개수를 계산**하고, 빨간색을 초록색으로 변경한 상태에서 **다시 한 번 계산**합니다.

R	R	R	B	B
G	G	B	B	B
B	B	B	R	R
B	B	R	R	R
R	R	R	R	R

연결 요소 **4개**

G	G	G	B	B
G	G	B	B	B
B	B	B	G	G
B	B	G	G	G
G	G	G	G	G

연결 요소 **3개**

적록색약: https://www.acmicpc.net/problem/10026

문제 풀이 핵심 아이디어

- **연결 요소**의 개수를 세는 방법은 다음과 같습니다.

1. 방문하지 않은 노드를 만날 때마다 카운트(count)하고, **DFS를 호출**합니다.
2. DFS는 **해당 위치로부터 연결된(연결요소에 포함된)** 모든 노드를 방문 처리합니다.

적록색약: https://www.acmicpc.net/problem/10026

문제 풀이 핵심 아이디어

- **연결 요소**의 개수를 세는 방법은 다음과 같습니다.

1. 방문하지 않은 노드를 만날 때마다 카운트(count)하고, **DFS를 호출**합니다.
2. DFS는 **해당 위치로부터 연결된(연결요소에 포함된)** 모든 노드를 방문 처리합니다.

1. 방문하지 않은 노드를 만나면 **DFS 호출** 시작

G	G	G	B
G	G	G	B
G	B	B	G
B	B	G	G

2. 인접한 노드에 대해서도 재귀적으로 방문 처리

G	G	G	B
G	G	G	B
G	B	B	G
B	B	G	G

적록색약: https://www.acmicpc.net/problem/10026

소스 코드


```python
import sys
# 빠른 입력 함수 사용
input = sys.stdin.readline
# 재귀 제한 변경
sys.setrecursionlimit(int(1e5))

# 상, 하, 좌, 우 방향 정의
dx = [-1, 1, 0, 0]
dy = [0, 0, -1, 1]

def dfs(x, y):
    for i in range(4): # 인접한 위치 확인
        nx = x + dx[i]
        ny = y + dy[i]
        # 인접한 위치가 범위를 벗어나는 경우 무시
        if nx <= -1 or nx >= n or ny <= -1 or ny >= n:
            continue
        # 처음 방문하고, 같은 색상이라면
        if not visited[nx][ny]:
            if graph[x][y] == graph[nx][ny]:
                visited[nx][ny] = -1 # 방문 처리
                dfs(nx, ny)

n = int(input()) # 맵의 크기(N)
graph = [[] for _ in range(n)] # 2차원 배열
for i in range(n):
    for x in input():
        graph[i].append(x)

# 연결 요소(connected component)의 개수 세기
answer = 0
visited = [[False] * n for _ in range(n)] # 방문 여부
for i in range(n):
    for j in range(n):
        if not visited[i][j]: # 처음 방문하는 경우
            visited[i][j] = True # 방문 처리
            dfs(i, j)
            answer += 1
print(answer, end=' ')

# R → G 변환
for i in range(n):
    for j in range(n):
        if graph[i][j] == 'R':
            graph[i][j] = 'G'

# 연결 요소(connected component)의 개수 세기
answer = 0
visited = [[False] * n for _ in range(n)] # 방문 여부
for i in range(n):
    for j in range(n):
        if not visited[i][j]: # 처음 방문하는 경우
            visited[i][j] = True # 방문 처리
            dfs(i, j)
            answer += 1
print(answer)
```

혼자 힘으로 풀어보기

문제 제목: 노드사이의 거리

문제 난이도: 중(Medium)

문제 유형: DFS, 트리(Tree)

추천 풀이 시간: 50분

노드사이의 거리: https://www.acmicpc.net/problem/1240

문제 풀이 핵심 아이디어

- 본 문제에서는 트리가 **양방향 간선**으로 구성되어 있다고 가정합니다.
- 문제에서 주어지는 노드의 개수 N은 최대 1000입니다.

  주어지는 그래프는 항상 **트리(tree)** 형식이므로, 간선의 개수는 $N-1$개입니다.
- **트리에서 노드 A와 B를 잇는 경로는 오직 1개만 존재**합니다.

  따라서 트리 내에 존재하는 노드 A와 B의 거리를 구하기 위한 시간 복잡도는 $O(N)$입니다.
- 쿼리의 개수 M은 최대 1000입니다.

  따라서, 매 쿼리마다 노드 A와 B의 거리를 계산해도 요구되는 시간 복잡도는 $O(NM)$입니다.

노드사이의 거리: https://www.acmicpc.net/problem/1240

문제 풀이 핵심 아이디어

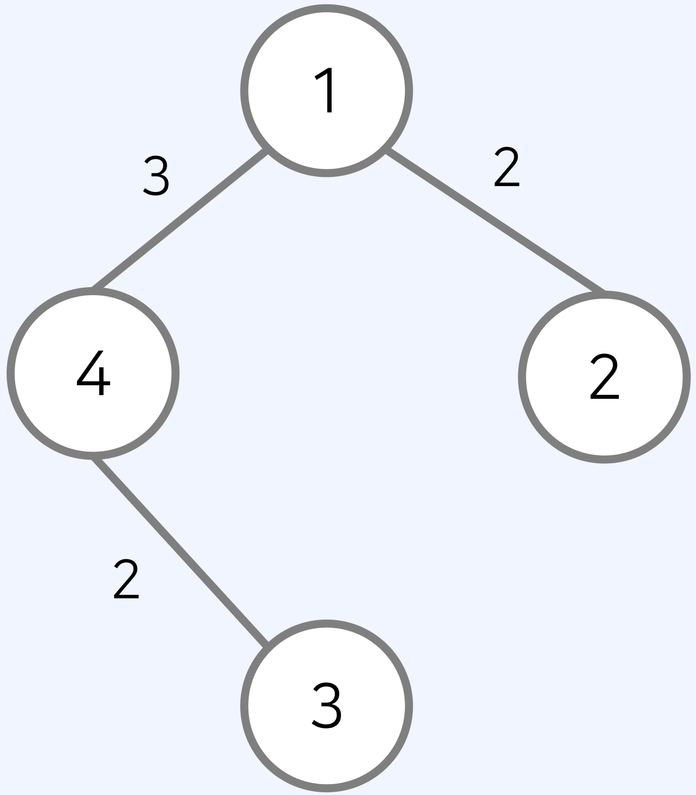
[핵심 아이디어 요약]

- 트리에서는 임의의 두 노드 간의 **경로**가 오직 1개입니다.
- 따라서 트리에서는 BFS가 아닌 DFS로도 간단히 최단 거리를 계산할 수 있습니다.
- 단순히 매 쿼리(query)마다, 노드 A에서 B까지의 거리를 계산합니다.

노드사이의 거리: https://www.acmicpc.net/problem/1240

문제 풀이 핵심 아이디어

- 문제에서의 예시를 **그래프로 표현**하면 다음과 같습니다.



- 1번 노드부터 2번 노드까지의 거리: 2
- 3번 노드부터 2번 노드까지의 거리: 7

노드사이의 거리: https://www.acmicpc.net/problem/1240

소스 코드


```python
import sys
# 빠른 입력 함수 사용
input = sys.stdin.readline
# 재귀 제한 변경
sys.setrecursionlimit(int(1e5))

# DFS(깊이 우선 탐색) 함수 구현
def dfs(x):
    # x의 인접 노드를 하나씩 확인
    for data in graph[x]:
        y = data[0]
        cost = data[1] # x에서 y로 가는 간선 비용
        if not visited[y]:
            visited[y] = True # 방문 처리
            distance[y] = distance[x] + cost
            dfs(y)

# 노드의 개수(N)와 간선의 개수(M) 입력받기
n, m = map(int, input().split())
# 트리 정보 입력받기
graph = [[] for _ in range(n + 1)]
for i in range(n - 1):
    # 노드 a와 노드 b가 연결(간선 비용은 c)
    a, b, c = map(int, input().split())
    graph[a].append((b, c))
    graph[b].append((a, c))

# 각 쿼리(query)마다 매번 DFS를 수행
for i in range(m):
    # x에서 y로 가기 위한 최단 거리 계산
    x, y = map(int, input().split())
    # 방문 여부 및 최단 거리 테이블 초기화
    visited = [False] * (n + 1)
    distance = [-1] * (n + 1)
    visited[x] = True # 시작점 방문 처리
    distance[x] = 0 # 자신까지의 거리는 0
    dfs(x) # x부터의 최단 거리 계산
    print(distance[y])
```