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

텀 프로젝트: https://www.acmicpc.net/problem/9466

혼자 힘으로 풀어보기

문제 제목: 텀 프로젝트

문제 난이도: ★★★☆☆

문제 유형: DFS, 방향 그래프 내 사이클 판별

추천 풀이 시간: 60분

텀 프로젝트: https://www.acmicpc.net/problem/9466

문제 풀이 핵심 아이디어

- 모든 학생들은 프로젝트를 함께 하고 싶은 학생을 **한 명씩 선택**한다.
- 자기 자신을 선택하는 것도 가능하다.
- 결과적으로 어느 프로젝트 팀에도 속하지 못한 학생들의 수를 계산해야 한다.

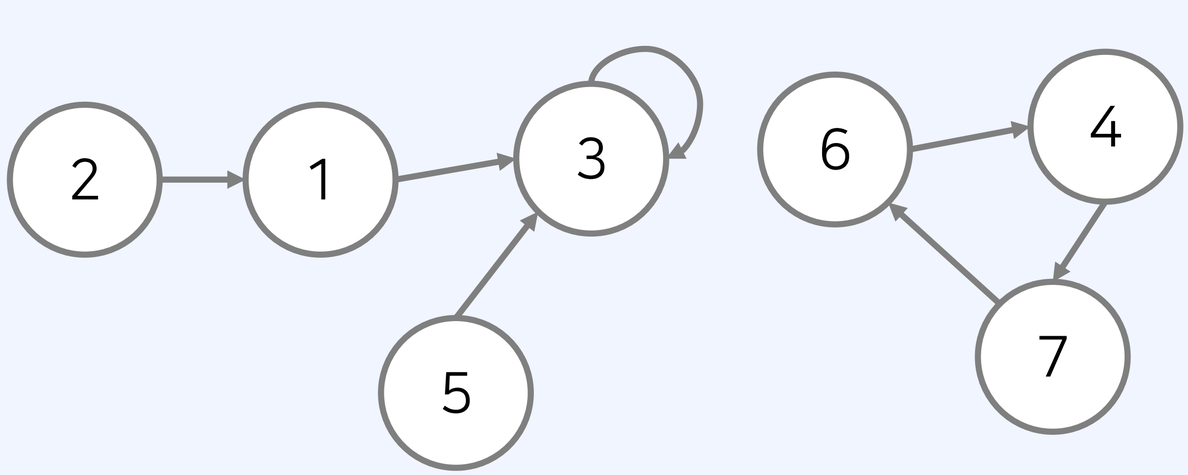
텀 프로젝트: https://www.acmicpc.net/problem/9466

문제 풀이 핵심 아이디어

[핵심 아이디어]

- 모든 학생들은 자신이 원하는 학생과 같은 팀에 소속되고자 한다.

  각 학생들의 선택을 **방향 간선으로 표현하여 그래프를 구성**할 수 있다.
- 한 팀에 포함된 임의의 학생 A와 B가 있을 때, A에서 B로 도달할 수 있어야 한다.
- 즉, 본 문제는 **사이클(cycle)을 구성하는 부분 그래프에 포함된 노드의 개수를 세는 문제**다.

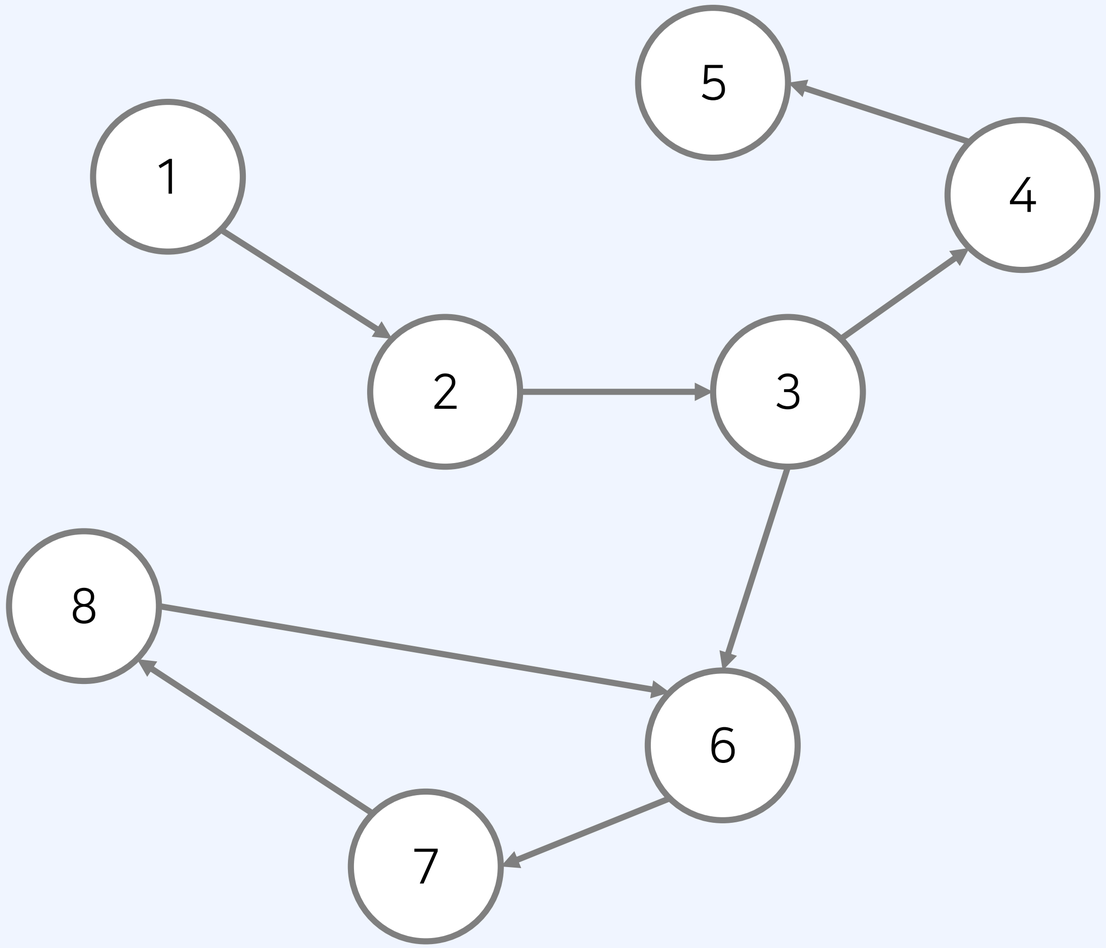


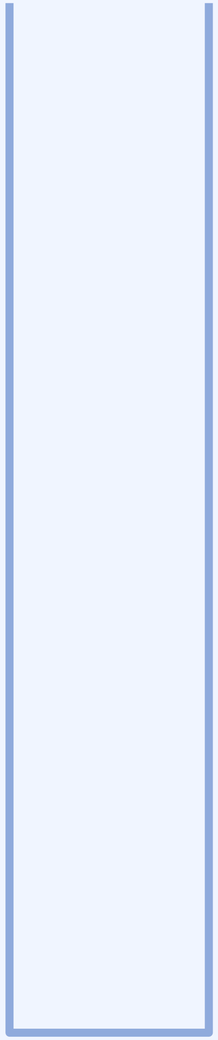
사이클 노드: {3, 4, 6, 7}

텀 프로젝트: https://www.acmicpc.net/problem/9466

문제 풀이 핵심 아이디어

- 방향 그래프 내 사이클 판별





- 처리가 완료된 노드: {}
- 방문한 노드: {}

: 방문한 노드

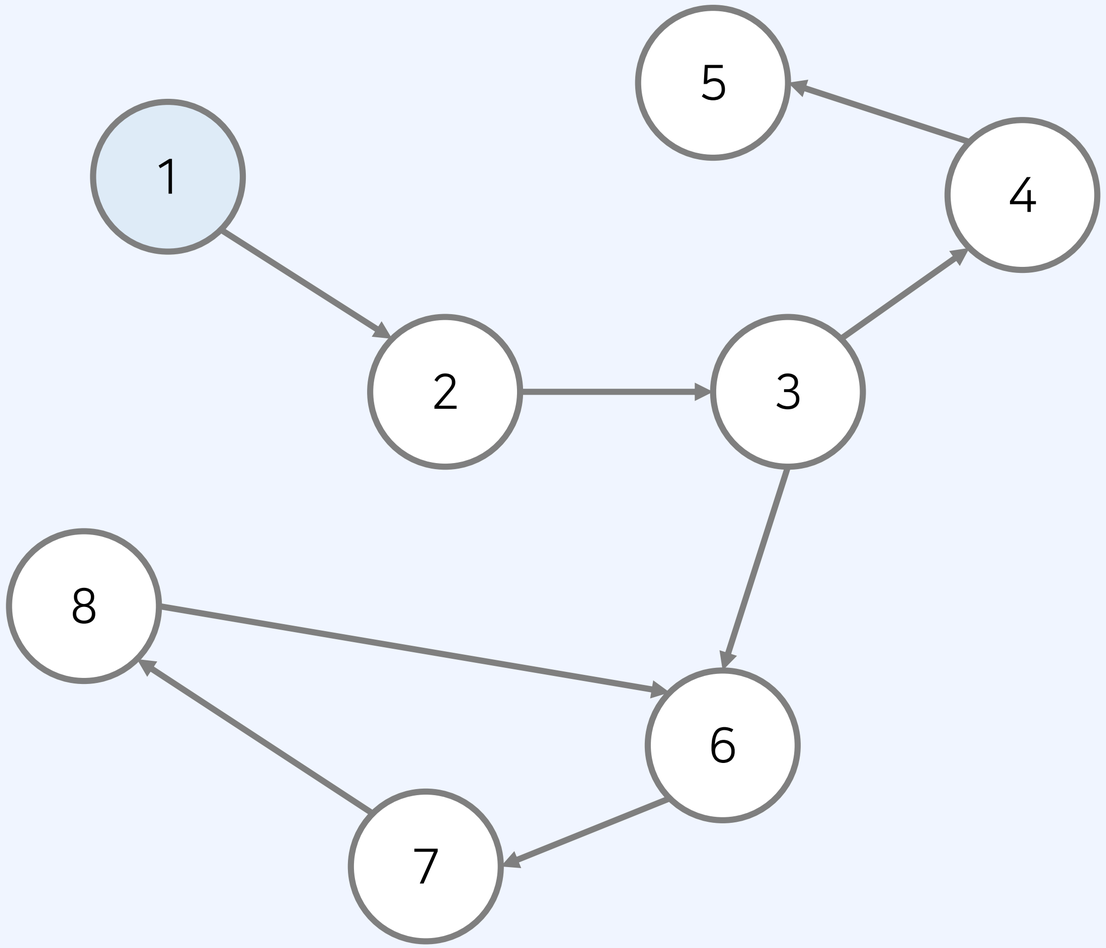
: 처리가 완료된 노드 (스택에서 추출된)

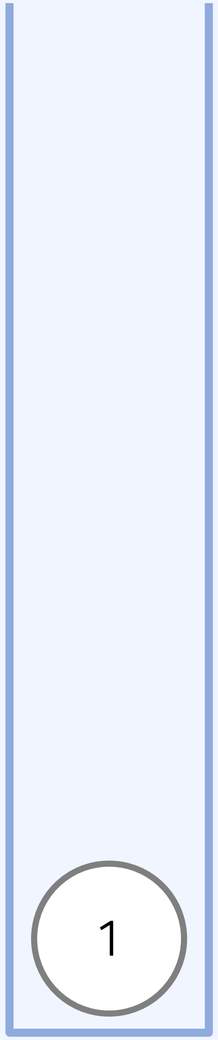
[해설] 전체 그래프를 확인한다.

텀 프로젝트: https://www.acmicpc.net/problem/9466

문제 풀이 핵심 아이디어

- 방향 그래프 내 사이클 판별





- 처리가 완료된 노드: {}
- 방문한 노드: {1}

: 방문한 노드

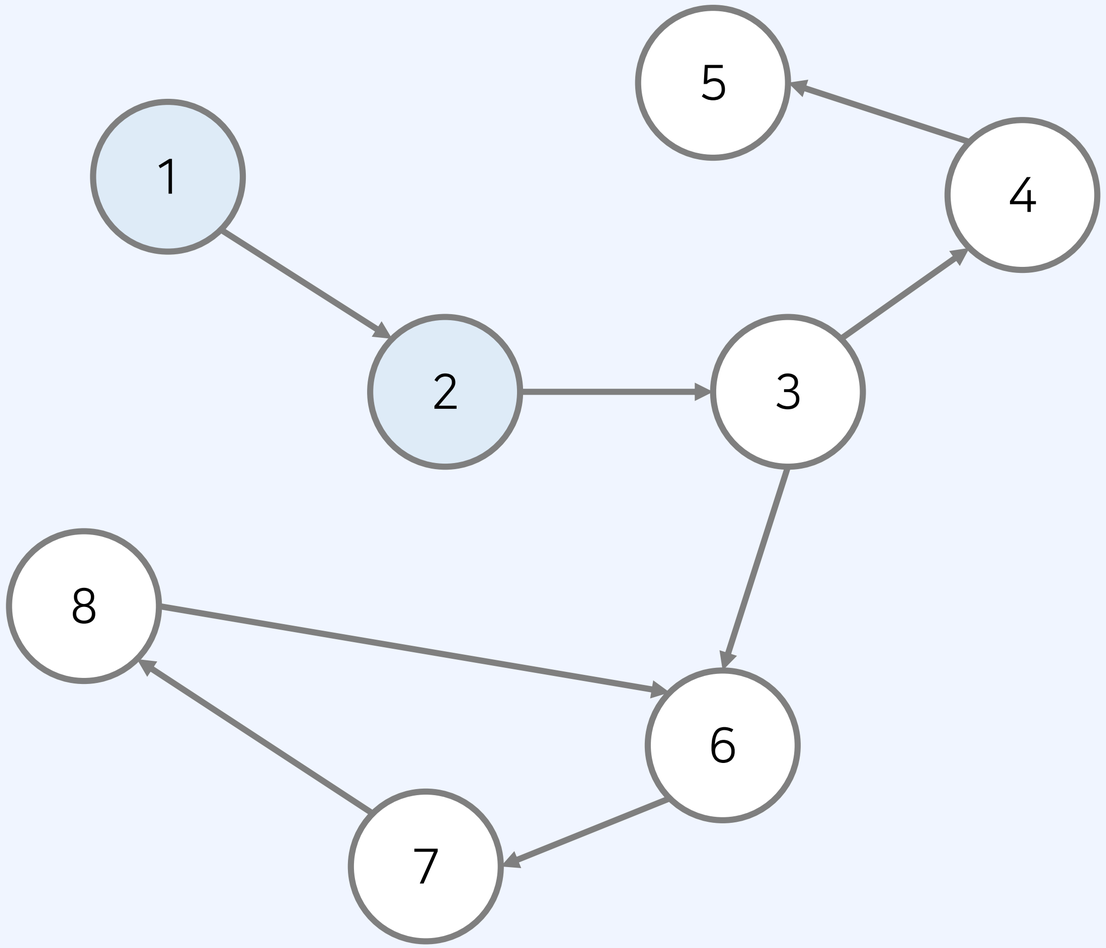
: 처리가 완료된 노드 (스택에서 추출된)

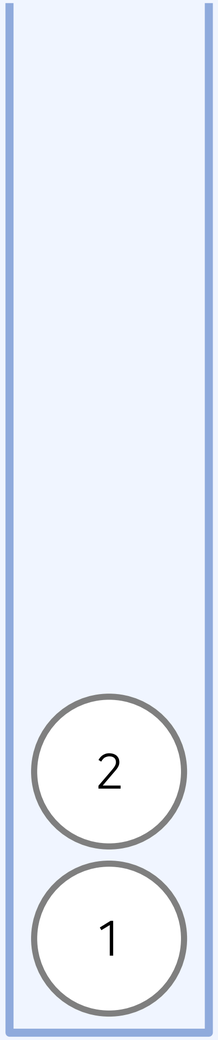
[해설] 노드 1을 방문해 스택에 삽입한다.

텀 프로젝트: https://www.acmicpc.net/problem/9466

문제 풀이 핵심 아이디어

- 방향 그래프 내 사이클 판별





- 처리가 완료된 노드: $\{\}$
- 방문한 노드: $\{1, 2\}$

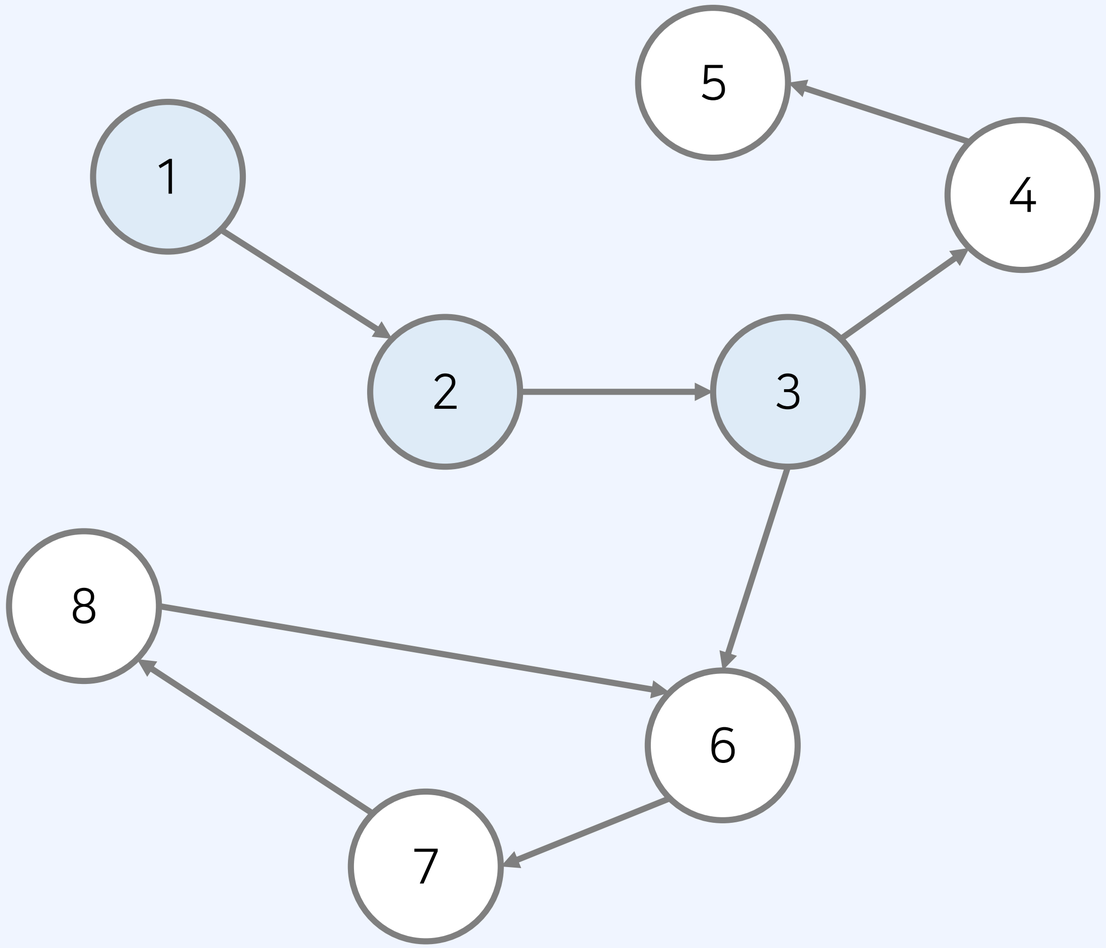
: 방문한 노드

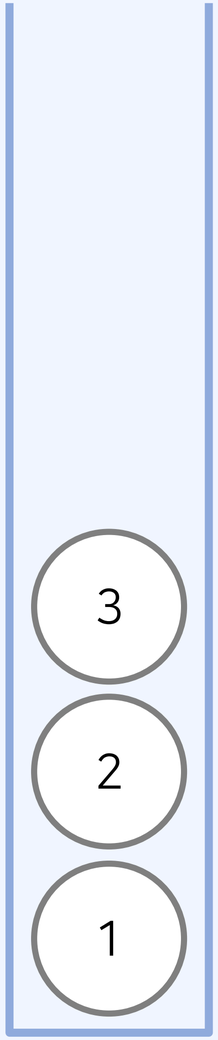
: 처리가 완료된 노드 (스택에서 추출된)

[해설] 스택의 최상단 노드와 인접한 **노드 2**를 방문해 스택에 삽입한다.

문제 풀이 핵심 아이디어

- 방향 그래프 내 사이클 판별





- 처리가 완료된 노드: $\{\}$
- 방문한 노드: $\{1, 2, 3\}$

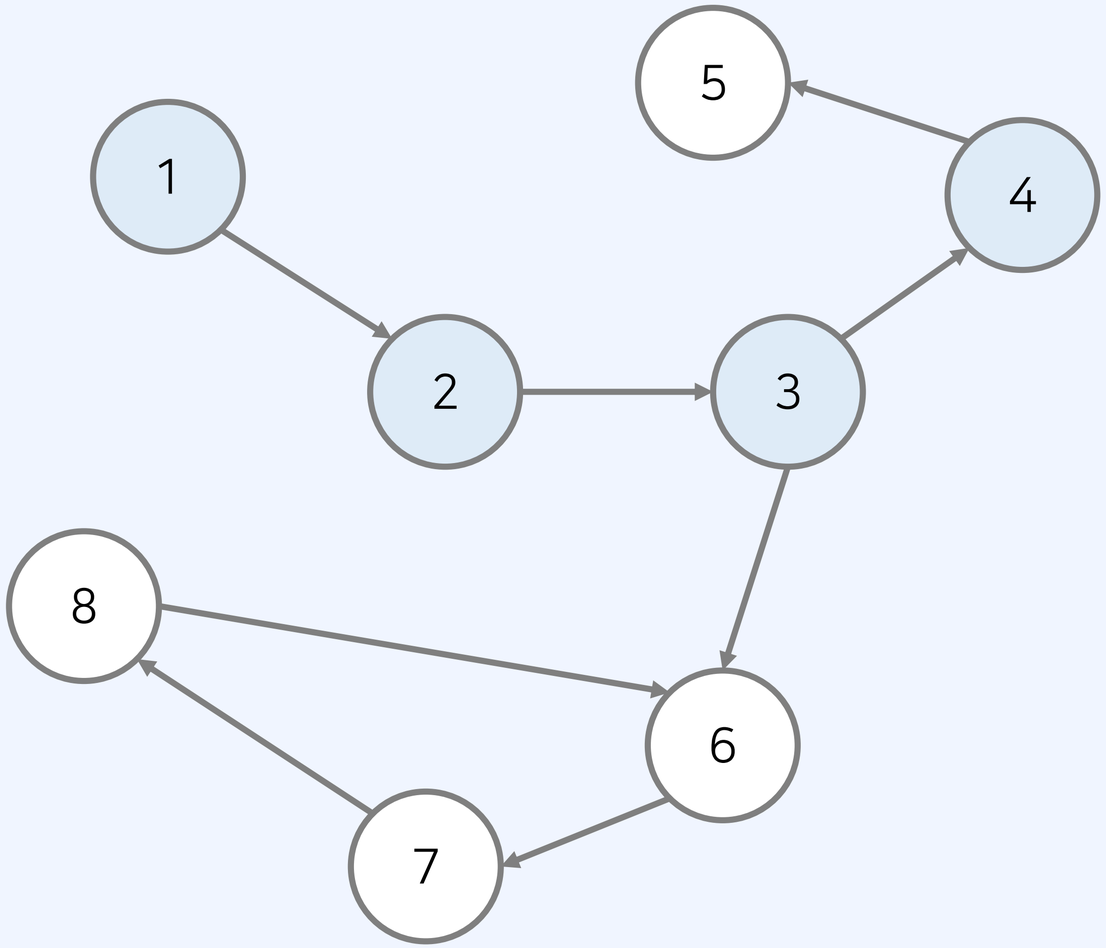
: 방문한 노드

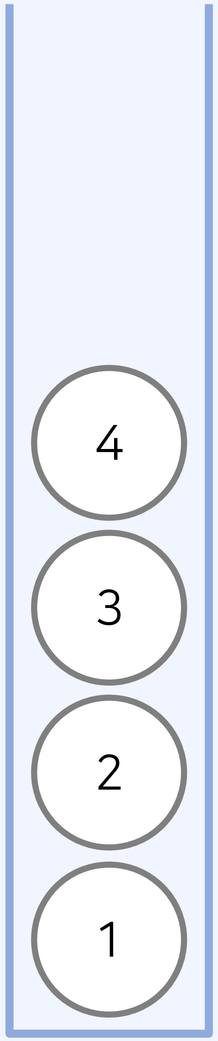
: 처리가 완료된 노드 (스택에서 추출된)

[해설] 스택의 최상단 노드와 인접한 **노드 3**을 방문해 스택에 삽입한다.

문제 풀이 핵심 아이디어

- 방향 그래프 내 사이클 판별





- 처리가 완료된 노드: $\{\}$
- 방문한 노드: $\{1, 2, 3, 4\}$

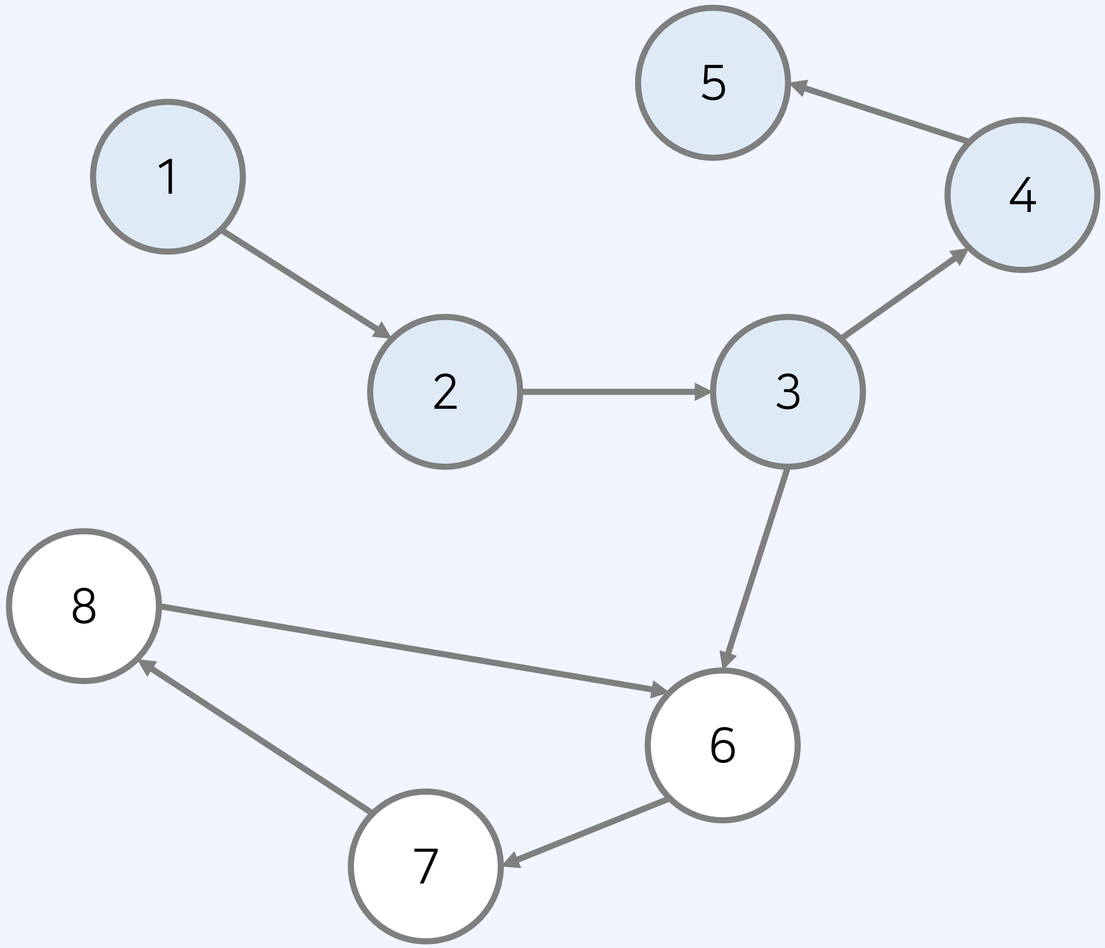
: 방문한 노드

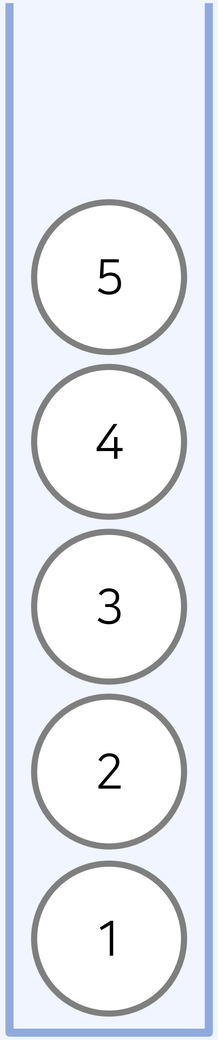
: 처리가 완료된 노드 (스택에서 추출된)

[해설] 스택의 최상단 노드와 인접한 **노드 4**를 방문해 스택에 삽입한다.

문제 풀이 핵심 아이디어

- 방향 그래프 내 사이클 판별





- 처리가 완료된 노드: {}
- 방문한 노드: $\{1, 2, 3, 4, 5\}$

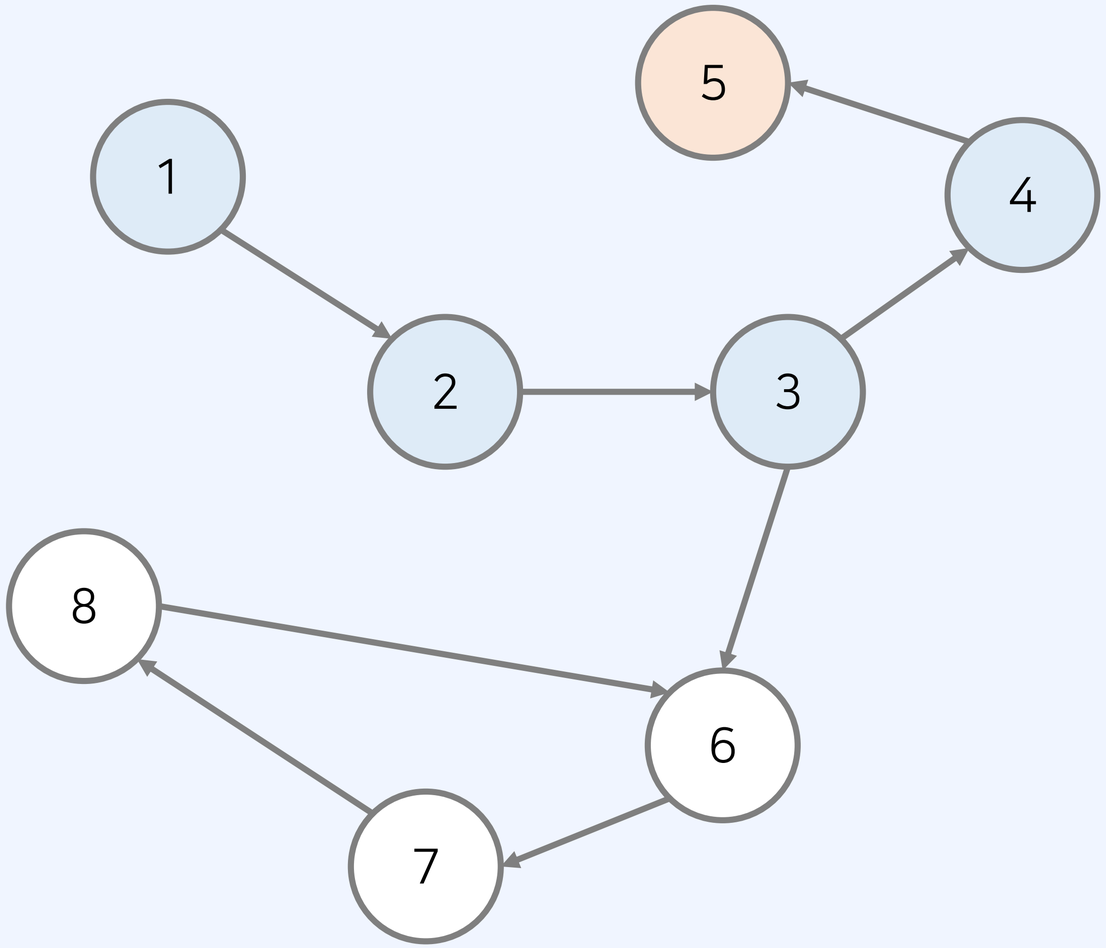
: 방문한 노드

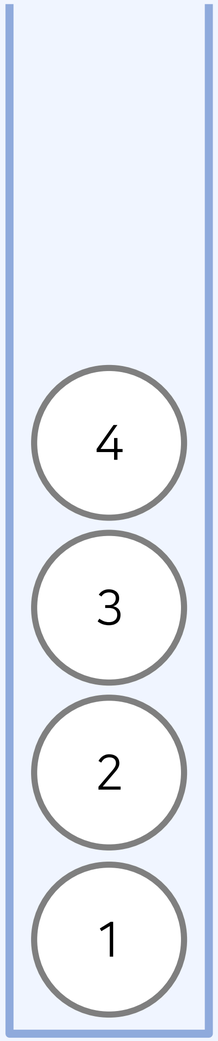
: 처리가 완료된 노드 (스택에서 추출된)

[해설] 스택의 최상단 노드와 인접한 **노드 5**를 방문해 스택에 삽입한다.

문제 풀이 핵심 아이디어

- 방향 그래프 내 사이클 판별





- 처리가 완료된 노드: $\{5\}$
- 방문한 노드: $\{1, 2, 3, 4, 5\}$

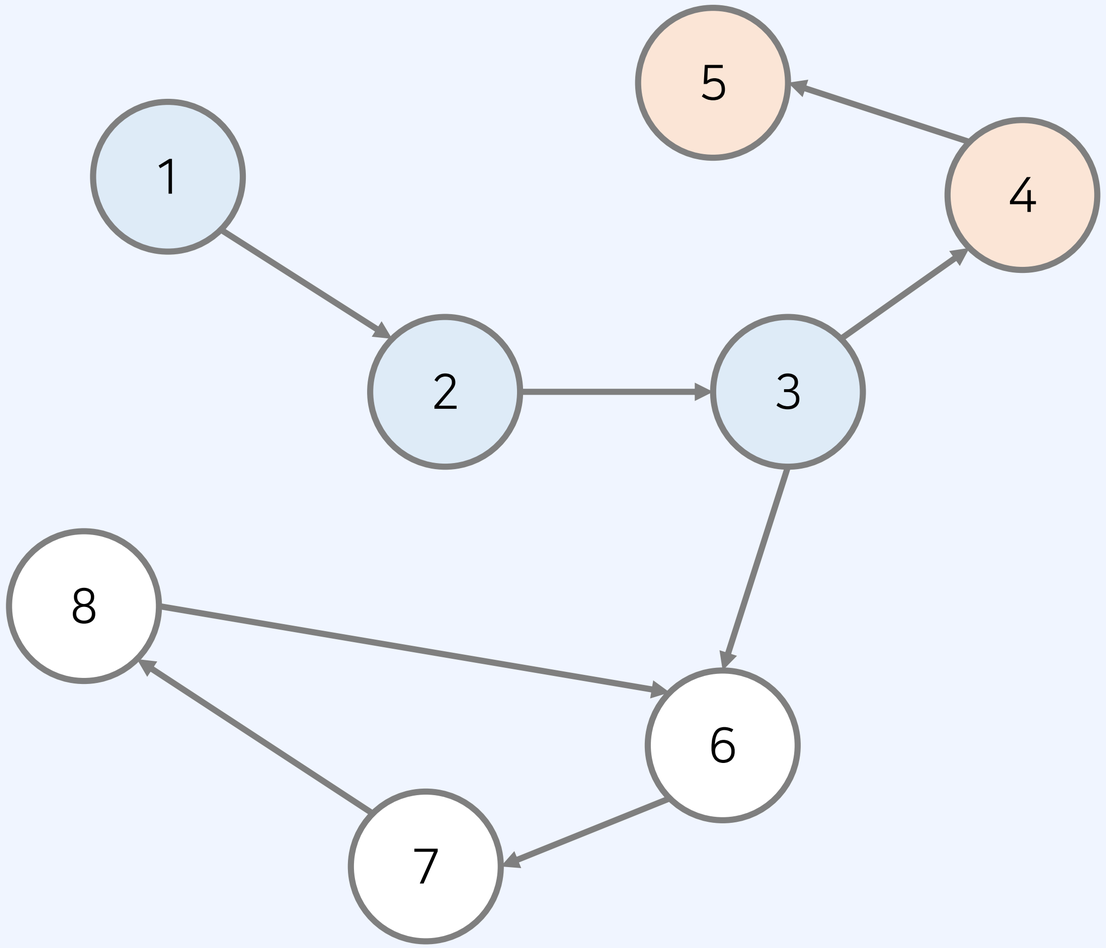
: 방문한 노드

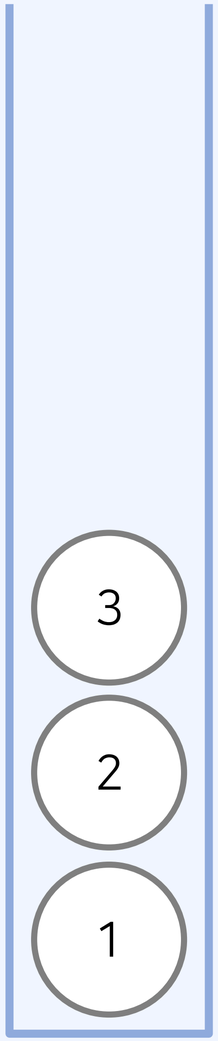
: 처리가 완료된 노드 (스택에서 추출된)

[해설] 스택의 최상단 노드에게
방문하지 않은 인접 노드가 없으므로
최상단 노드를 추출한다.

문제 풀이 핵심 아이디어

- 방향 그래프 내 사이클 판별





- 처리가 완료된 노드: $\{4, 5\}$
- 방문한 노드: $\{1, 2, 3, 4, 5\}$

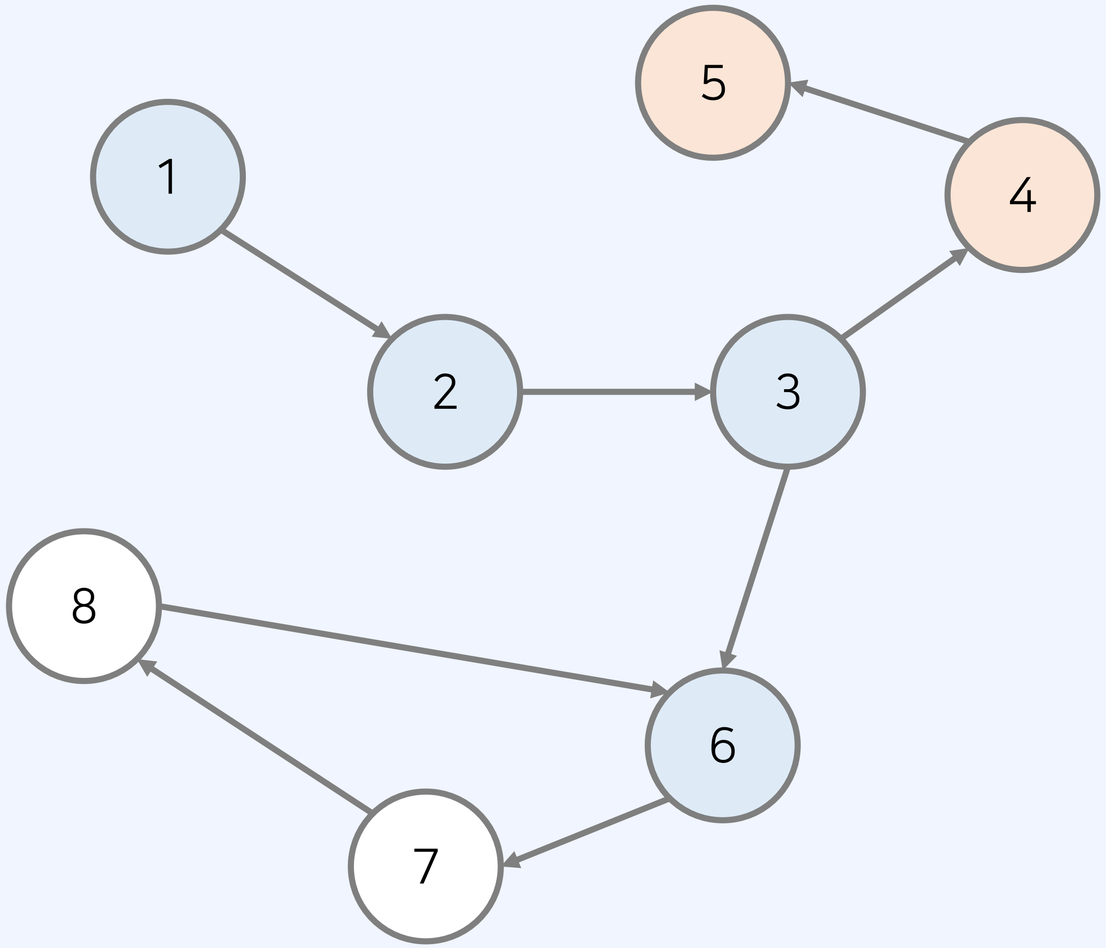
: 방문한 노드

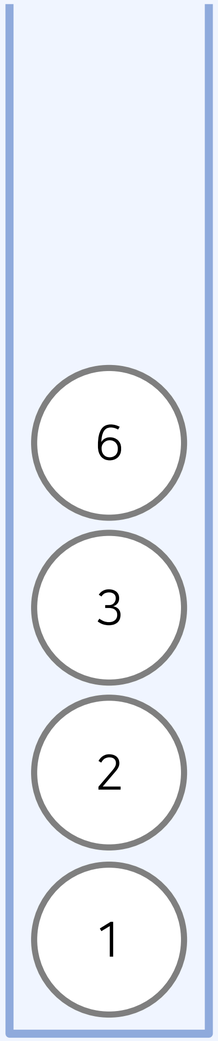
: 처리가 완료된 노드 (스택에서 추출된)

[해설] 스택의 최상단 노드에게
방문하지 않은 인접 노드가 없으므로
최상단 노드를 추출한다.

문제 풀이 핵심 아이디어

- 방향 그래프 내 사이클 판별





- 처리가 완료된 노드: $\{4, 5\}$
- 방문한 노드: $\{1, 2, 3, 4, 5, 6\}$

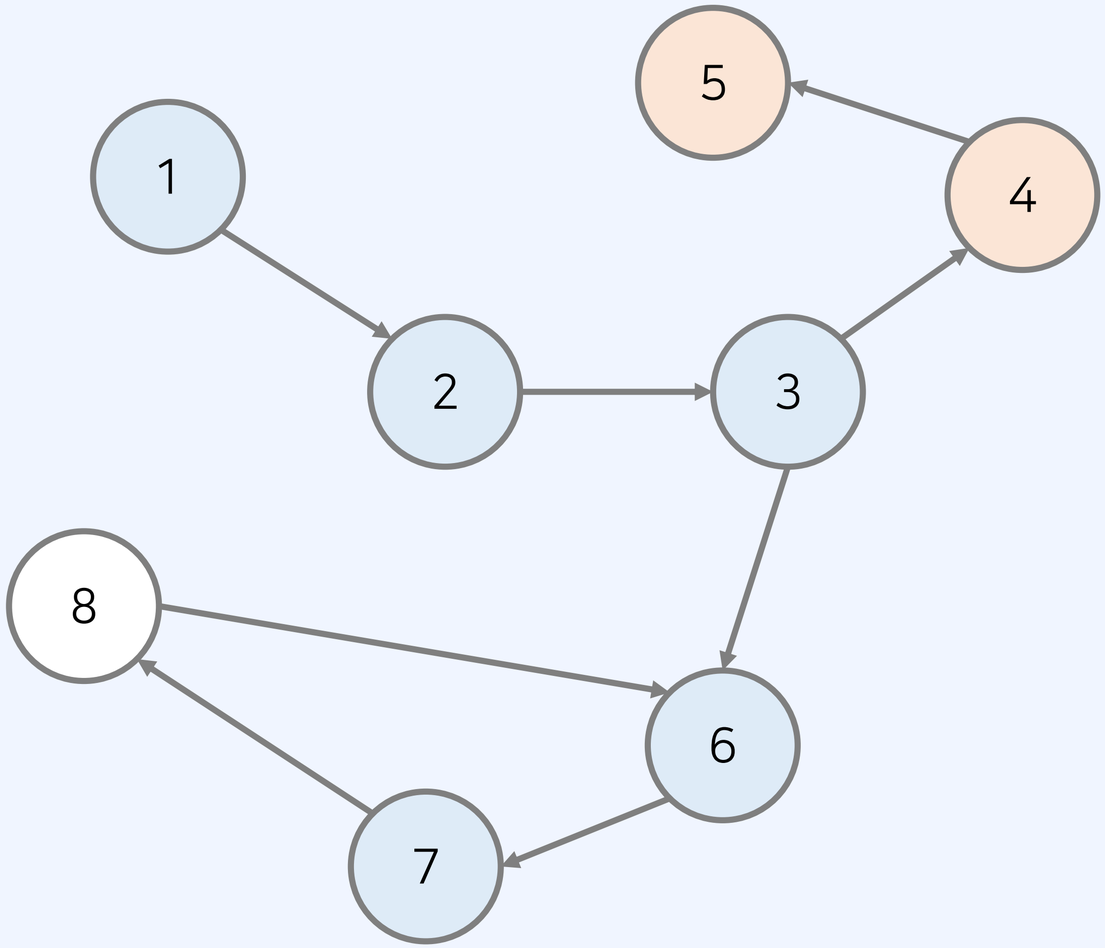
: 방문한 노드

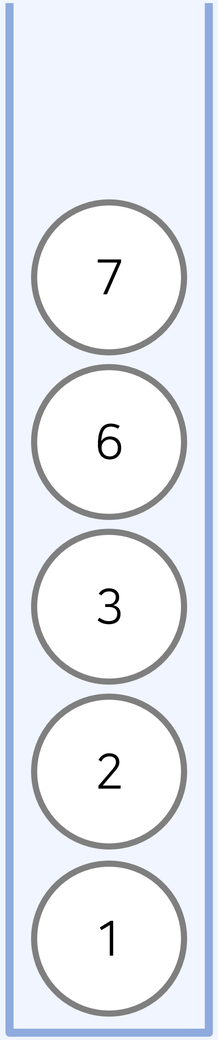
: 처리가 완료된 노드 (스택에서 추출된)

[해설] 스택의 최상단 노드와 인접한 **노드 6**를 방문해 스택에 삽입한다.

문제 풀이 핵심 아이디어

- 방향 그래프 내 사이클 판별





- 처리가 완료된 노드: $\{4, 5\}$
- 방문한 노드: $\{1, 2, 3, 4, 5, 6, 7\}$

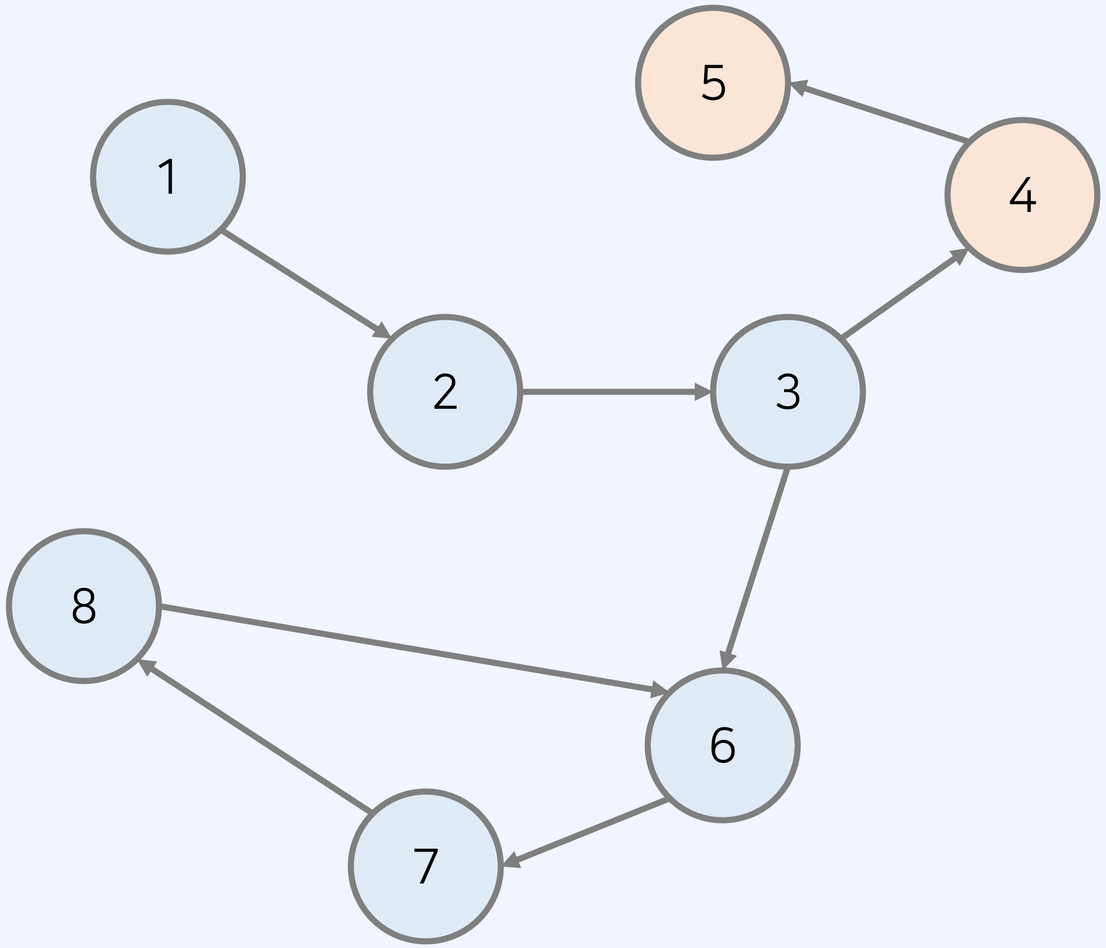
: 방문한 노드

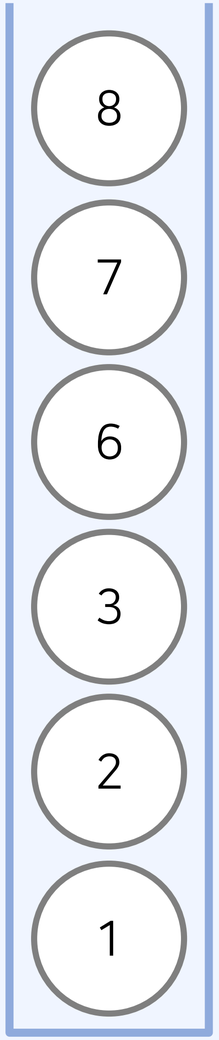
: 처리가 완료된 노드 (스택에서 추출된)

[해설] 스택의 최상단 노드와 인접한 **노드 7**을 방문해 스택에 삽입한다.

문제 풀이 핵심 아이디어

- 방향 그래프 내 사이클 판별





- 처리가 완료된 노드: $\{4, 5\}$
- 방문한 노드: $\{1, 2, 3, 4, 5, 6, 7, 8\}$

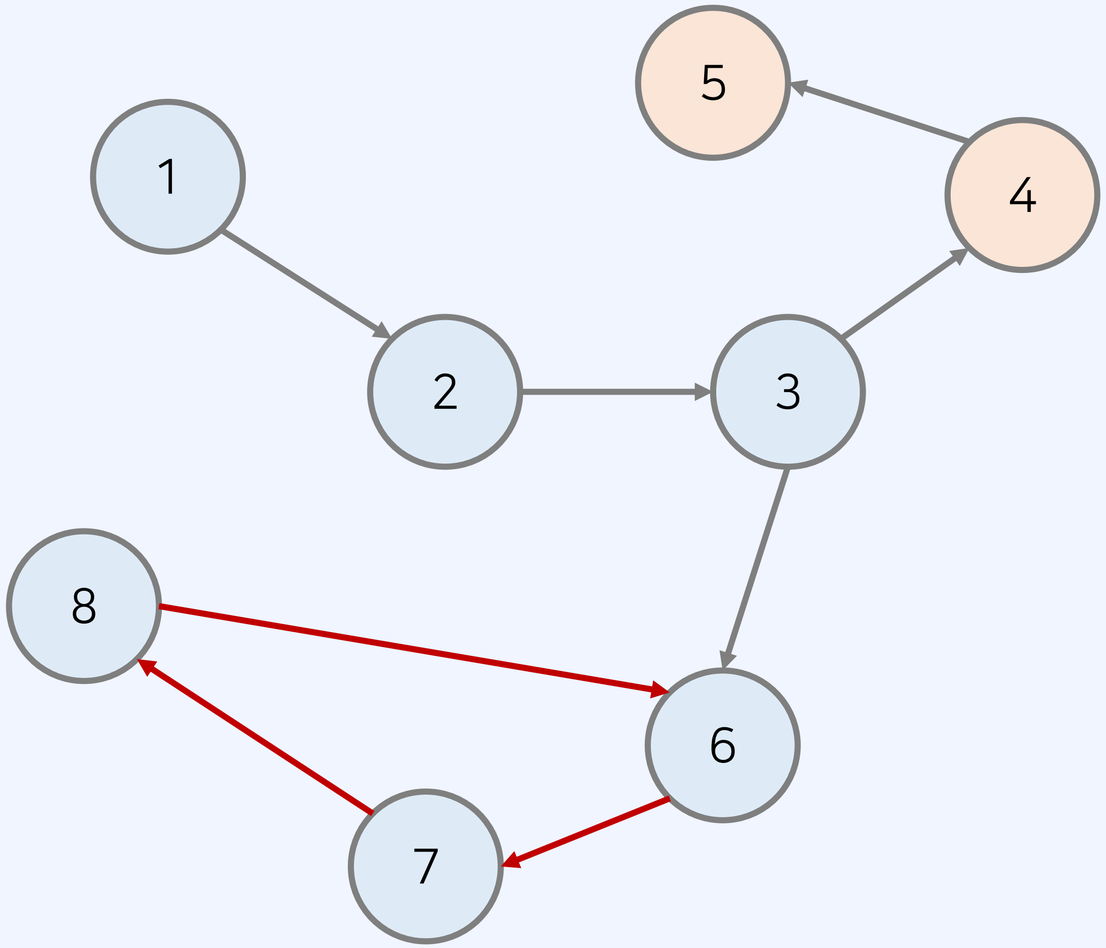
: 방문한 노드

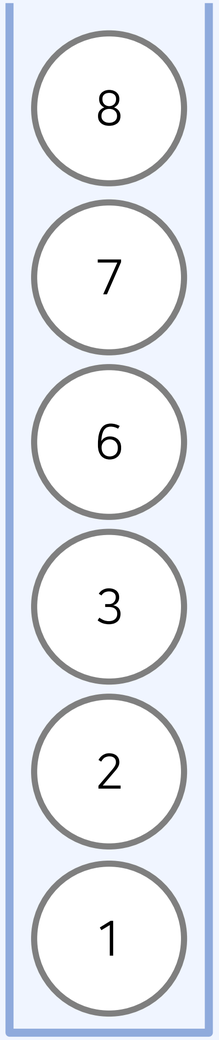
: 처리가 완료된 노드 (스택에서 추출된)

[해설] 스택의 최상단 노드와 인접한 **노드 8**을 방문해 스택에 삽입한다.

문제 풀이 핵심 아이디어

- 방향 그래프 내 사이클 판별





- 처리가 완료된 노드: $\{4, 5\}$
- 방문한 노드: $\{1, 2, 3, 4, 5, 6, 7, 8\}$

: 방문한 노드

: 처리가 완료된 노드 (스택에서 추출된)

[해설] 스택의 최상단 노드와 인접한 **노드 6**을 이미 방문한 적이 있으며 **처리가 완료되지 않았다.**

사이클 발생

텀 프로젝트: https://www.acmicpc.net/problem/9466

문제 풀이 핵심 아이디어


```python
import sys
# 빠른 입력 함수 사용
input = sys.stdin.readline
# 재귀 제한 변경
sys.setrecursionlimit(int(1e6))

def is_cycle(x):
    visited[x] = True # 현재 노드 방문 처리
    y = graph[x] # 다음 노드
    # 다음 노드를 아직 방문하지 않았다면
    if not visited[y]:
        is_cycle(y)
    # 다음 노드를 방문한 적 있고, 완료되지 않았다면
    elif not finished[y]: # 사이클 발생
        # 사이클에 포함된 노드 저장
        while y != x:
            result.append(y)
            y = graph[y]
        result.append(x)
    # 현재 노드의 처리 완료
    finished[x] = True
```

```python
# 각 테스트 케이스 수행
for _ in range(int(input())):
    n = int(input())
    graph = [0] + list(map(int, input().split()))
    visited = [False] * (n + 1)
    finished = [False] * (n + 1)
    result = []

    # 각 노드에서 DFS로 사이클 판별
    for x in range(1, n + 1):
        if not visited[x]:
            is_cycle(x)

    print(n - len(result))
```

숫자고르기: https://www.acmicpc.net/problem/2668

혼자 힘으로 풀어보기

문제 제목: 숫자고르기

문제 난이도: ★★★☆☆

문제 유형: DFS, 방향 그래프 내 사이클 판별

추천 풀이 시간: 50분

숫자고르기: https://www.acmicpc.net/problem/2668

문제 풀이 핵심 아이디어

- 첫째 줄에서 뽑은 정수들이 이루는 집합 A와 뽑힌 정수들의 바로 밑에 있는 정수들이 이루는 집합 B가 **일치**하도록 하는 집합 A의 **최대 크기를 계산**한다.
- 아래 예시에서는 $A = \{1, 3, 5\}$일 때 $B = \{3, 1, 5\}$이며, 이것이 최대 크기다.

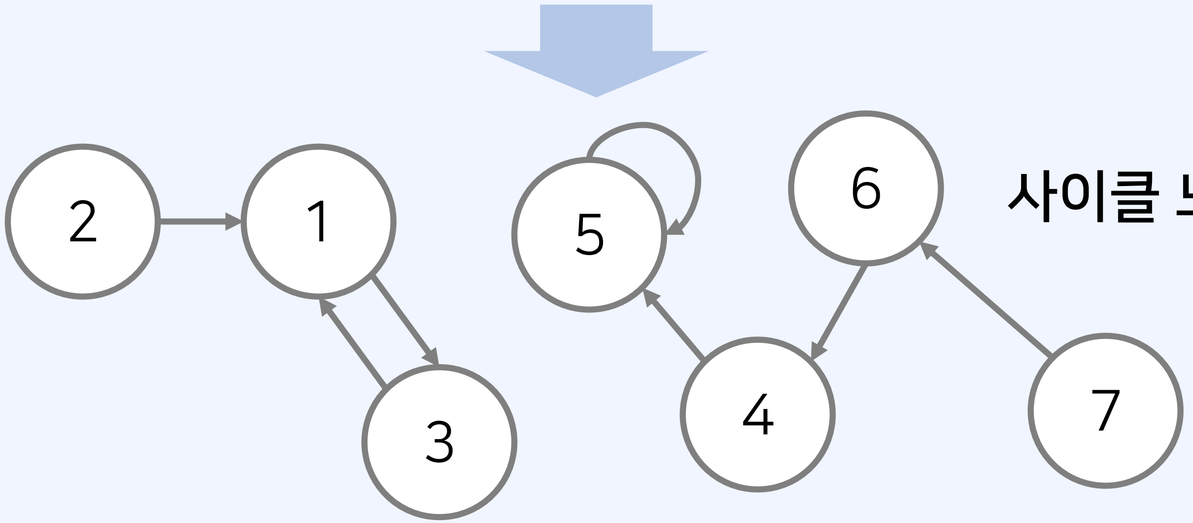
1	2	3	4	5	6	7
3	1	1	5	5	4	6

숫자고르기: https://www.acmicpc.net/problem/2668

문제 풀이 핵심 아이디어

- 첫째 줄과 둘째 줄의 관계를 **방향 간선으로 표현하여 그래프를 구성**할 수 있다.
- [핵심] 본 문제는 **사이클(cycle)을 구성하는 부분 그래프에 포함된 노드의 개수를 세는 문제**다.

1	2	3	4	5	6	7
3	1	1	5	5	4	6



사이클 노드: {1, 3, 5}

숫자고르기: https://www.acmicpc.net/problem/2668

문제 풀이 핵심 아이디어


```python
import sys
# 빠른 입력 함수 사용
input = sys.stdin.readline
# 재귀 제한 변경
sys.setrecursionlimit(int(1e6))

def is_cycle(x):
    visited[x] = True # 현재 노드 방문 처리
    y = graph[x] # 다음 노드
    # 다음 노드를 아직 방문하지 않았다면
    if not visited[y]:
        is_cycle(y)
    # 다음 노드를 방문한 적 있고, 완료되지 않았다면
    elif not finished[y]: # 사이클 발생
        # 사이클에 포함된 노드 저장
        while y != x:
            result.append(y)
            y = graph[y]
        result.append(x)
    # 현재 노드의 처리 완료
    finished[x] = True
```

```python
# 전체 그래프 정보 입력받기
n = int(input())
graph = [0] * (n + 1)
for i in range(1, n + 1):
    graph[i] = int(input())

visited = [False] * (n + 1)
finished = [False] * (n + 1)
result = []

# 각 노드에서 DFS로 사이클 판별
for x in range(1, n + 1):
    if not visited[x]:
        is_cycle(x)

print(len(result))
for x in sorted(result):
    print(x)
```

트리: https://www.acmicpc.net/problem/4803

혼자 힘으로 풀어보기

문제 제목: 트리

문제 난이도: ★★★☆☆

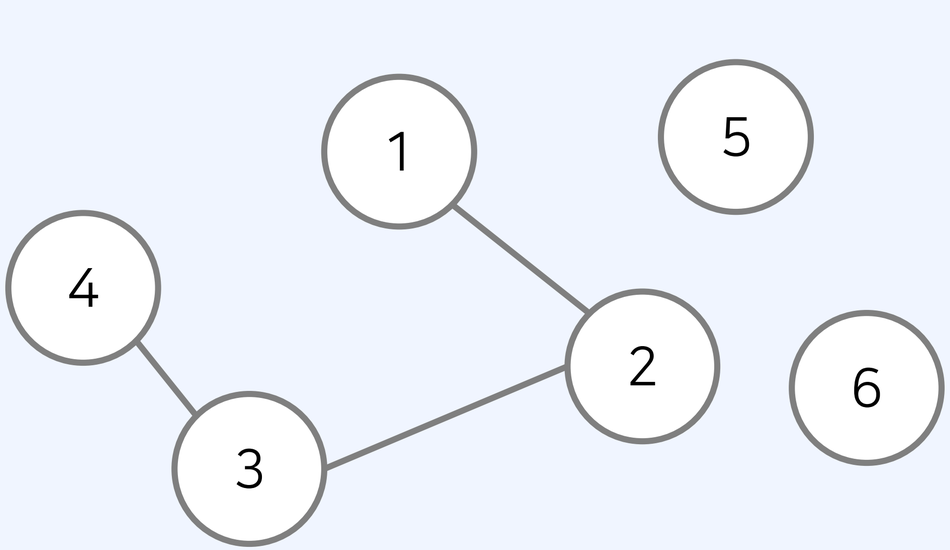
문제 유형: DFS, 무방향 그래프 내 사이클 판별

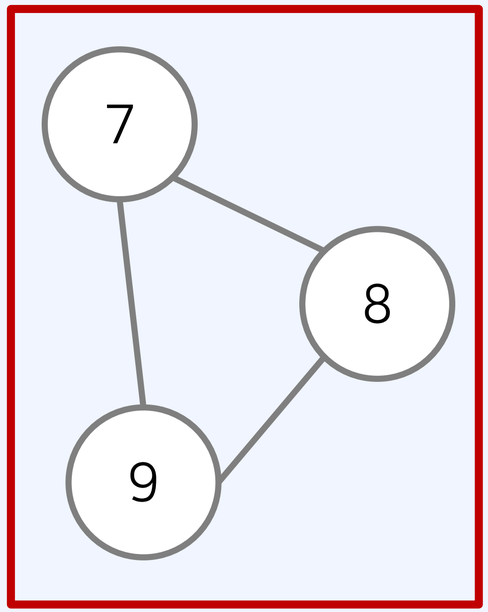
추천 풀이 시간: 50분

트리: https://www.acmicpc.net/problem/4803

문제 풀이 핵심 아이디어

- 하나의 그래프 안에 포함된 **트리(tree)의 개수를 세는 문제**다.
- **트리**: 사이클이 없는 연결 요소
  - 트리의 정의에 따라서 **DFS를 이용해 트리의 개수를 계산**하여 문제를 해결할 수 있다.





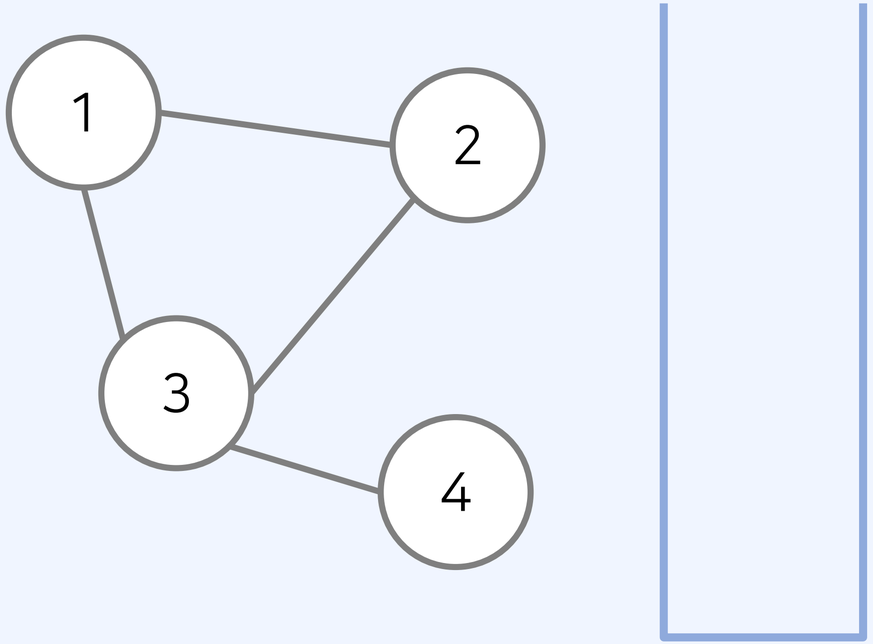
사이클 존재

3개의 트리 존재

트리: https://www.acmicpc.net/problem/4803

문제 풀이 핵심 아이디어

- 무방향 그래프 내 사이클 판별 알고리즘은 다음과 같습니다.
- 특정 노드에서 DFS를 수행하는 과정에서 "인접 노드가 **이미 방문한 노드라면**" 사이클입니다.
- 단, 무방향 그래프이므로 직전 노드는 제외합니다.

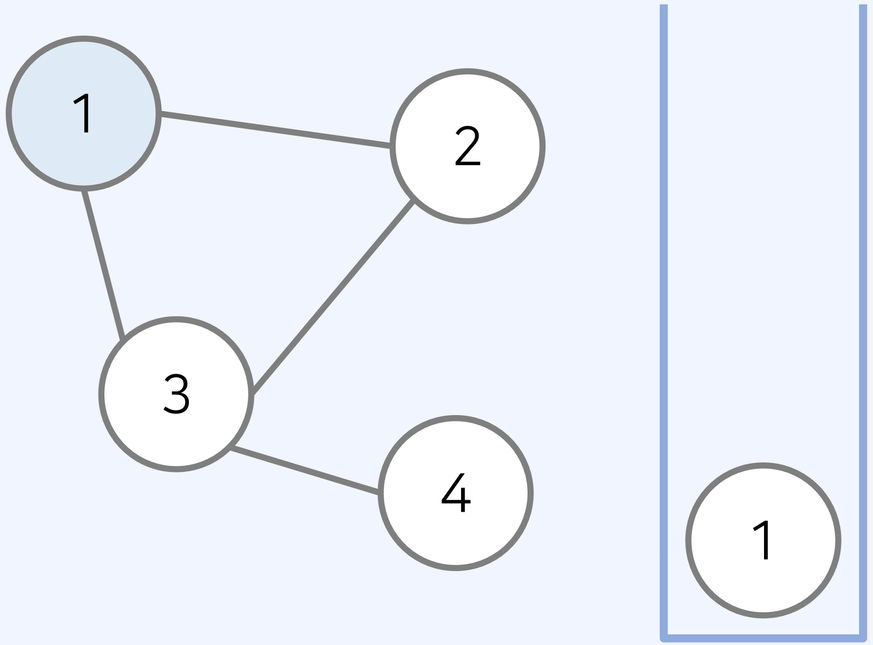



```python
# 무방향 그래프에서 사이클 여부 확인
def is_cycle(x, prev):
    # 현재 노드 방문 처리
    visited[x] = True
    # 다음 노드(인접한 노드)를 하나씩 확인하며
    for y in graph[x]:
        # 다음 노드가 이미 방문한 노드라면
        if visited[y]:
            # 직전 노드가 아니라면(무방향 그래프이므로 필요)
            if y != prev:
                return True # 사이클 발생
        else:
            # 다음 노드를 기준으로 사이클이 발생한다면
            if is_cycle(y, x):
                return True # 사이클 발생
    return False
```

트리: https://www.acmicpc.net/problem/4803

문제 풀이 핵심 아이디어

- 무방향 그래프 내 사이클 판별 알고리즘은 다음과 같습니다.
- 특정 노드에서 DFS를 수행하는 과정에서 "인접 노드가 **이미 방문한 노드라면**" 사이클입니다.
- 단, 무방향 그래프이므로 직전 노드는 제외합니다.

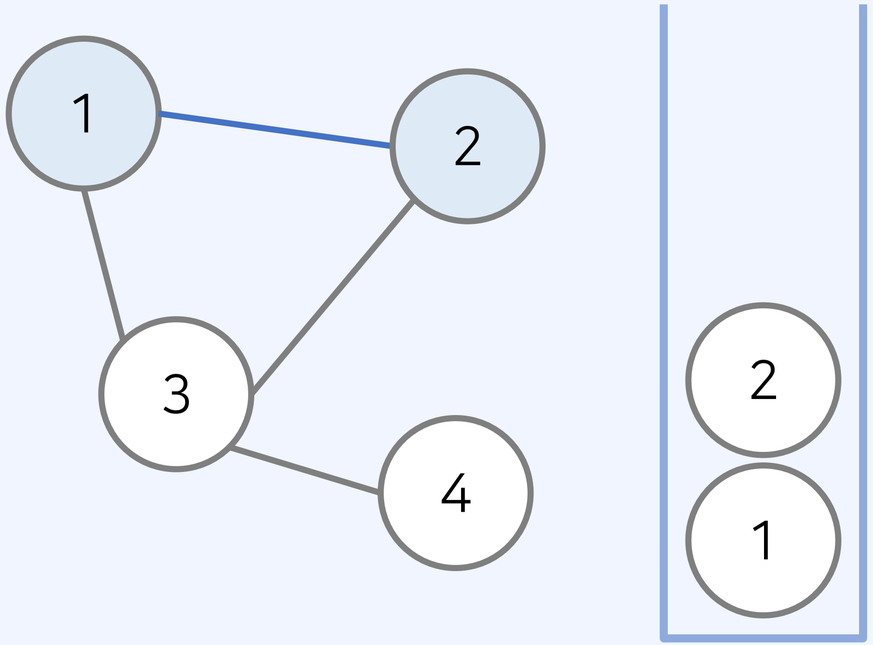



```python
# 무방향 그래프에서 사이클 여부 확인
def is_cycle(x, prev):
    # 현재 노드 방문 처리
    visited[x] = True
    # 다음 노드(인접한 노드)를 하나씩 확인하며
    for y in graph[x]:
        # 다음 노드가 이미 방문한 노드라면
        if visited[y]:
            # 직전 노드가 아니라면(무방향 그래프이므로 필요)
            if y != prev:
                return True # 사이클 발생
        else:
            # 다음 노드를 기준으로 사이클이 발생한다면
            if is_cycle(y, x):
                return True # 사이클 발생
    return False
```

트리: https://www.acmicpc.net/problem/4803

문제 풀이 핵심 아이디어

- 무방향 그래프 내 사이클 판별 알고리즘은 다음과 같습니다.
- 특정 노드에서 DFS를 수행하는 과정에서 "인접 노드가 **이미 방문한 노드라면**" 사이클입니다.
- 단, 무방향 그래프이므로 직전 노드는 제외합니다.

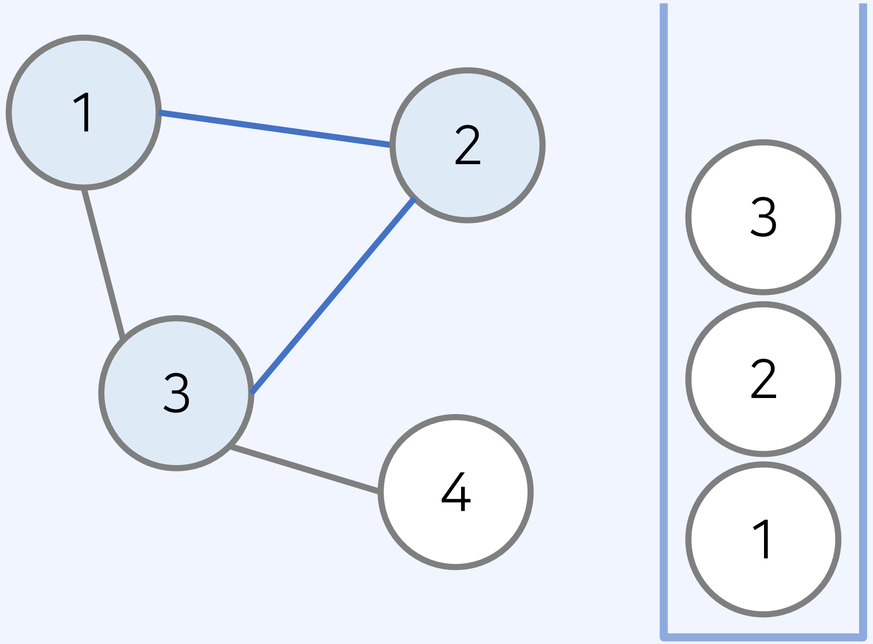



```python
# 무방향 그래프에서 사이클 여부 확인
def is_cycle(x, prev):
    # 현재 노드 방문 처리
    visited[x] = True
    # 다음 노드(인접한 노드)를 하나씩 확인하며
    for y in graph[x]:
        # 다음 노드가 이미 방문한 노드라면
        if visited[y]:
            # 직전 노드가 아니라면(무방향 그래프이므로 필요)
            if y != prev:
                return True # 사이클 발생
        else:
            # 다음 노드를 기준으로 사이클이 발생한다면
            if is_cycle(y, x):
                return True # 사이클 발생
    return False
```

트리: https://www.acmicpc.net/problem/4803

문제 풀이 핵심 아이디어

- 무방향 그래프 내 사이클 판별 알고리즘은 다음과 같습니다.
- 특정 노드에서 DFS를 수행하는 과정에서 "인접 노드가 **이미 방문한 노드라면**" 사이클입니다.
- 단, 무방향 그래프이므로 직전 노드는 제외합니다.

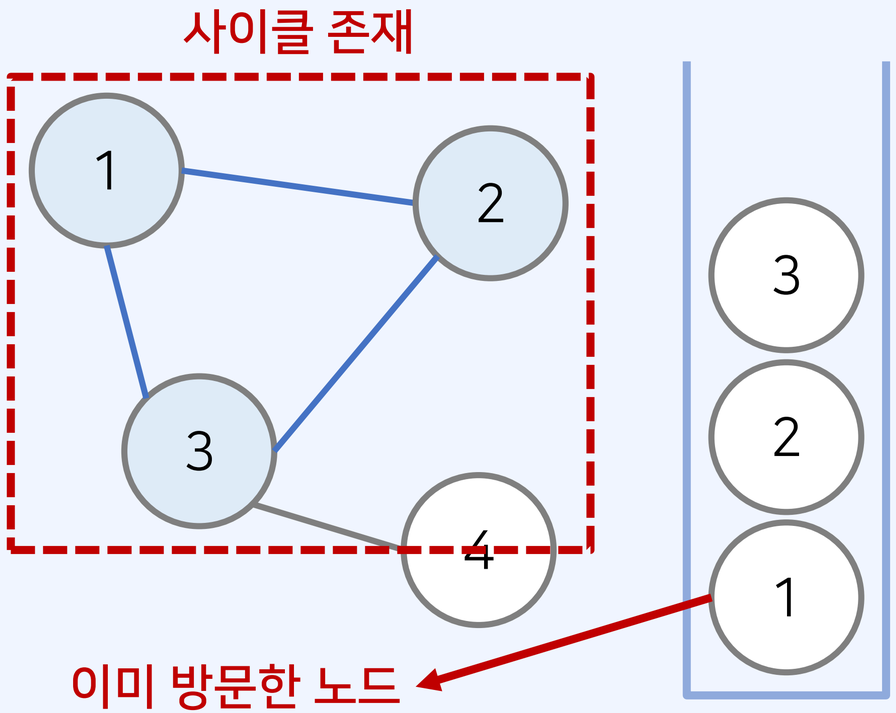



```python
# 무방향 그래프에서 사이클 여부 확인
def is_cycle(x, prev):
    # 현재 노드 방문 처리
    visited[x] = True
    # 다음 노드(인접한 노드)를 하나씩 확인하며
    for y in graph[x]:
        # 다음 노드가 이미 방문한 노드라면
        if visited[y]:
            # 직전 노드가 아니라면(무방향 그래프이므로 필요)
            if y != prev:
                return True # 사이클 발생
        else:
            # 다음 노드를 기준으로 사이클이 발생한다면
            if is_cycle(y, x):
                return True # 사이클 발생
    return False
```

트리: https://www.acmicpc.net/problem/4803

문제 풀이 핵심 아이디어

- 무방향 그래프 내 사이클 판별 알고리즘은 다음과 같습니다.
- 특정 노드에서 DFS를 수행하는 과정에서 "인접 노드가 **이미 방문한 노드라면**" 사이클입니다.
- 단, 무방향 그래프이므로 직전 노드는 제외합니다.




```python
# 무방향 그래프에서 사이클 여부 확인
def is_cycle(x, prev):
    # 현재 노드 방문 처리
    visited[x] = True
    # 다음 노드(인접한 노드)를 하나씩 확인하며
    for y in graph[x]:
        # 다음 노드가 이미 방문한 노드라면
        if visited[y]:
            # 직전 노드가 아니라면(무방향 그래프이므로 필요)
            if y != prev:
                return True # 사이클 발생
        else:
            # 다음 노드를 기준으로 사이클이 발생한다면
            if is_cycle(y, x):
                return True # 사이클 발생
    return False
```

트리: https://www.acmicpc.net/problem/4803

문제 풀이 핵심 아이디어


```python
import sys
# 빠른 입력 함수 사용
input = sys.stdin.readline
# 재귀 제한 변경
sys.setrecursionlimit(int(1e6))

# 무방향 그래프에서 사이클 여부 확인
def is_cycle(x, prev):
    # 현재 노드 방문 처리
    visited[x] = True
    # 다음 노드(인접한 노드)를 하나씩 확인하며
    for y in graph[x]:
        # 다음 노드가 이미 방문한 노드라면
        if visited[y]:
            # 직전 노드가 아니라면(무방향 그래프이므로 필요)
            if y != prev:
                return True # 사이클 발생
        else:
            # 다음 노드를 기준으로 사이클이 발생한다면
            if is_cycle(y, x):
                return True # 사이클 발생
    return False
```

```python
test_case = 1
while True: # 각 테스트 케이스 확인
    n, m = map(int, input().split())
    if n == 0 and m == 0: # 종료
        break

    graph = [[] for _ in range(n + 1)]
    visited = [False] * (n + 1)

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

    cnt = 0 # 그래프 내 트리의 개수
    for i in range(1, n + 1):
        if not visited[i]: # 연결 요소이면서
            if not is_cycle(i, 0): # 사이클이 아니라면
                cnt += 1 # 트리이므로, 카운트하기
    if cnt == 0:
        print(f'Case {test_case}: No trees.')
    elif cnt == 1:
        print(f'Case {test_case}: There is one tree.')
    else:
        print(f'Case {test_case}: A forest of {cnt} trees.')
    test_case += 1
```