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

이분 그래프: https://www.acmicpc.net/problem/1707

혼자 힘으로 풀어보기

문제 제목: 이분 그래프

문제 난이도: ★★★☆☆

문제 유형: BFS, DFS, 이분 그래프 판별

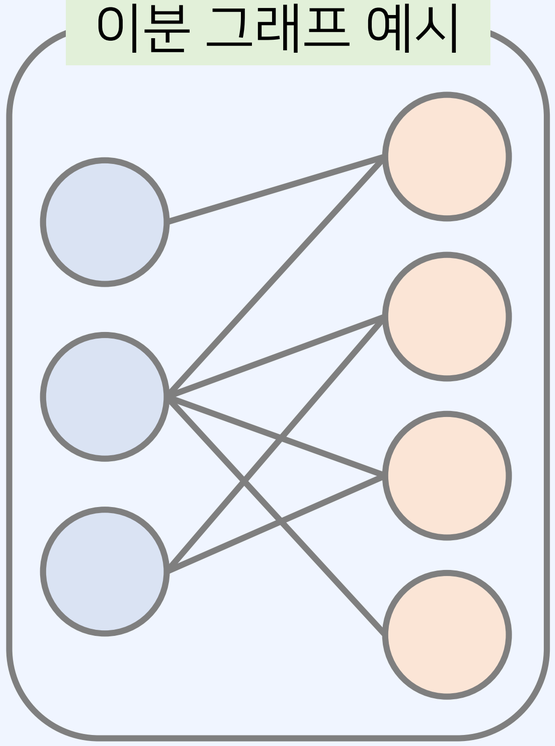
추천 풀이 시간: 60분

이분 그래프: https://www.acmicpc.net/problem/1707

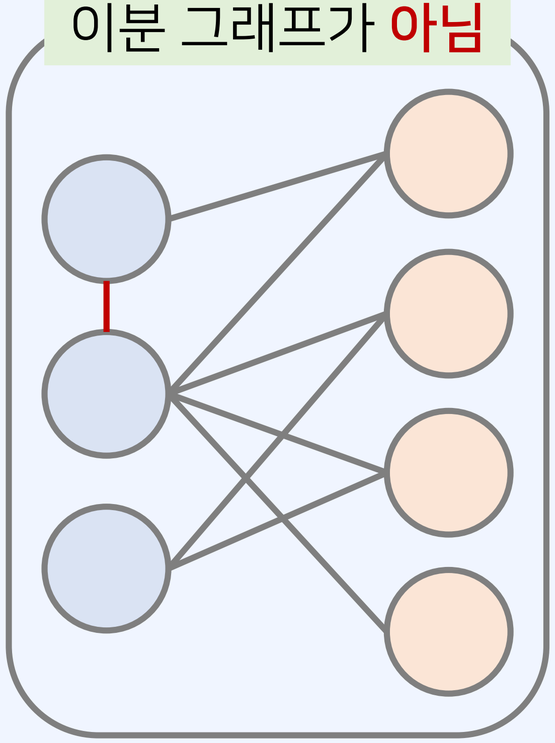
문제 풀이 핵심 아이디어

- **이분 그래프(Bipartite Graph)**는 노드들을 두 개의 집합으로 분할하여, 같은 집합에 속한 정점끼리 서로 인접하지 않는 그래프를 의미한다.

이분 그래프 예시



이분 그래프가 **아님**



1. 아직 방문하지 않은 각각의 노드에서 시작해 BFS로 **인접 노드로 이동**: **빨** → **파** → **빨** → **파** … 색칠하기
2. **이분 그래프 판별**: 모든 노드에 대해 인접 노드와 색상이 다른 지 여부를 판별하기

이분 그래프: https://www.acmicpc.net/problem/1707

문제 풀이 핵심 아이디어


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

def bfs(x):
    queue = deque([x])
    # 처음 노드는 빨간색으로 칠하기
    visited[x] = 0 # (0: 빨강 / 1: 파랑)
    while len(queue) != 0: # 큐가 빌 때까지 반복
        x = queue.popleft()
        # 인접한 노드를 하나씩 확인
        for y in graph[x]:
            if visited[y] == -1: # 빨강 ↔ 파랑
                visited[y] = (visited[x] + 1) % 2
                queue.append(y)

# 이분 그래프(bipartite graph) 판별 함수
def is_bipartite():
    for x in range(1, v + 1): # 하나씩 노드를 확인
        for y in graph[x]: # 인접 노드를 하나씩 확인
            # 같은 색상인 경우가 있다면
            if visited[x] == visited[y]:
                return False
    return True
```

```python
# 각 테스트 케이스만큼 반복
for _ in range(int(input())):
    # 정점의 개수(v)와 간선의 개수(e)
    v, e = map(int, input().split())
    # 그래프 정보 입력받기
    graph = [[] for _ in range(v + 1)]
    for i in range(e):
        a, b = map(int, input().split())
        graph[a].append(b)
        graph[b].append(a)
    visited = [-1] * (v + 1) # 방문 정보
    # BFS를 이용해 그래프 색칠하기
    for x in range(1, v + 1):
        if visited[x] == -1:
            bfs(x)
    if is_bipartite(): print("YES")
    else: print("NO")
```

알파벳: https://www.acmicpc.net/problem/1987

혼자 힘으로 풀어보기

문제 제목: 알파벳

문제 난이도: ★★★☆☆

문제 유형: DFS, BFS, 백트래킹, 완전 탐색

추천 풀이 시간: 50분

알파벳: https://www.acmicpc.net/problem/1987

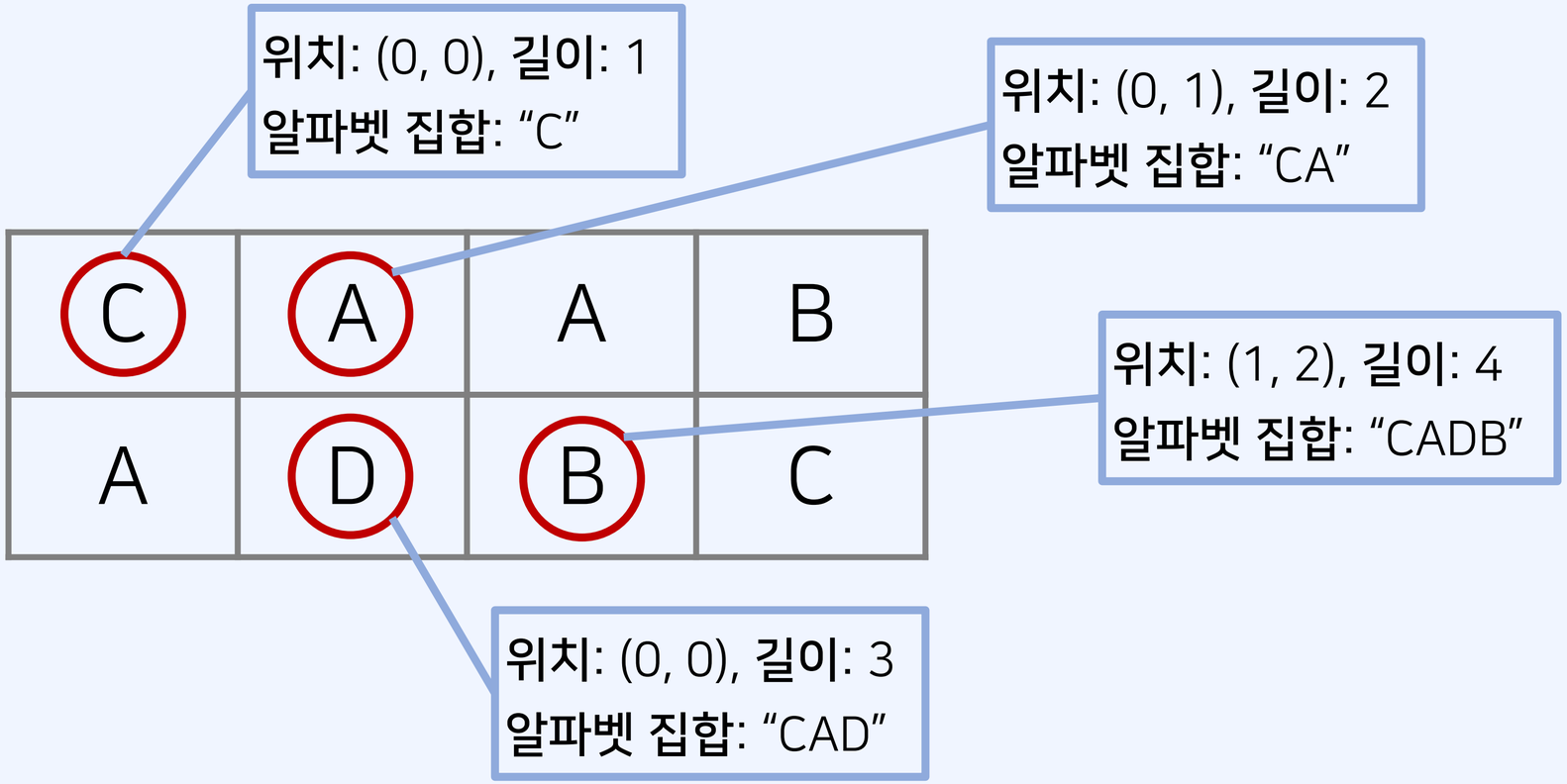
문제 풀이 핵심 아이디어

- 본 문제는 "최대한 어디까지 갈 수 있는가"를 구하는 문제다.
- 이를 위해 **완전 탐색**을 수행하여 문제를 해결할 수 있다.
- 완전 탐색을 위해서는 DFS 혹은 BFS를 사용하면 된다.
- 본 문제는 DFS 기반의 **백트래킹(back-tracking)**으로 해결하는 것이 구현상 간단하다.
- 높이(R)와 너비(C)가 최대 20이므로, 완전 탐색으로 문제를 해결할 수 있다.

알파벳: https://www.acmicpc.net/problem/1987

문제 풀이 핵심 아이디어

- 각 노드를 기준으로 인접한 노드를 확인한다.
- 인접 노드가 "현재까지의 알파벳 집합"에 포함되지 않은 알파벳을 가진다면 DFS를 호출한다.



알파벳: https://www.acmicpc.net/problem/1987

문제 풀이 핵심 아이디어


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

def dfs(x, y, cost):
    # 최대 거리(cost) 계산
    global result
    result = max(result, cost)
    # 인접한 위치를 확인하며
    for i in range(4):
        nx = x + dx[i]
        ny = y + dy[i]
        # 맵을 벗어나는 경우 무시
        if nx < 0 or nx >= r or ny < 0 or ny >= c:
            continue
        # 방문하지 않은 알파벳인 경우
        if not visited[ord(graph[nx][ny]) - 65]:
            # 백트래킹(back-tracking) 수행
            visited[ord(graph[nx][ny]) - 65] = True
            dfs(nx, ny, cost + 1)
            visited[ord(graph[nx][ny]) - 65] = False
```

```python
# 4가지 방향 정의(상, 하, 좌, 우)
dx = [-1, 1, 0, 0]
dy = [0, 0, -1, 1]

# 행(r)과 열(c)의 크기 입력
r, c = map(int, input().split())
graph = [] # 전체 맵 정보 입력
for _ in range(r):
    graph.append(input())

result = 0 # 최대 거리
visited = [False] * 26 # 방문한 알파벳 집합
visited[ord(graph[0][0]) - 65] = True
dfs(0, 0, 1) # 가장 왼쪽 위에서 출발
print(result)
```

환승: https://www.acmicpc.net/problem/5214

혼자 힘으로 풀어보기

문제 제목: 환승

문제 난이도: ★★★☆☆

문제 유형: BFS, 최단 거리

추천 풀이 시간: 50분

환승: https://www.acmicpc.net/problem/5214

문제 풀이 핵심 아이디어

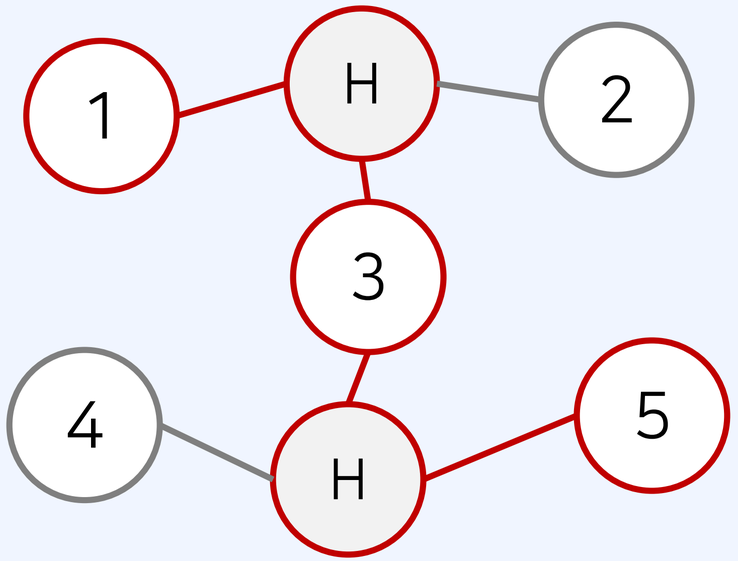
- 하이퍼튜브 하나는 역 K개를 서로 연결한다.
- 1번 역에서 N번 역으로 가는데 방문하는 최소 역의 수를 출력하면 된다.
- 역의 개수 N이 최대 100,000이고, 하이퍼튜브의 개수 M은 최대 1,000이다.
- 일종의 최단 거리 문제이다.
  - **각 간선의 비용이 모두 동일**하기 때문에, BFS로 최단 거리를 계산할 수 있다.
  - 만약 각 간선의 비용이 다를 수 있다면, 다익스트라와 같은 방법을 이용해야 한다.

환승: https://www.acmicpc.net/problem/5214

문제 풀이 핵심 아이디어

[문제 해결 아이디어]

- **하이퍼튜브**를 포함하여 전체 노드를 그래프로 구성한다.
- BFS를 이용해서 1번에서 N번까지의 최단 거리를 계산한다.
- $N = 5$일 때의 예시로는 다음과 같은 것이 있다.



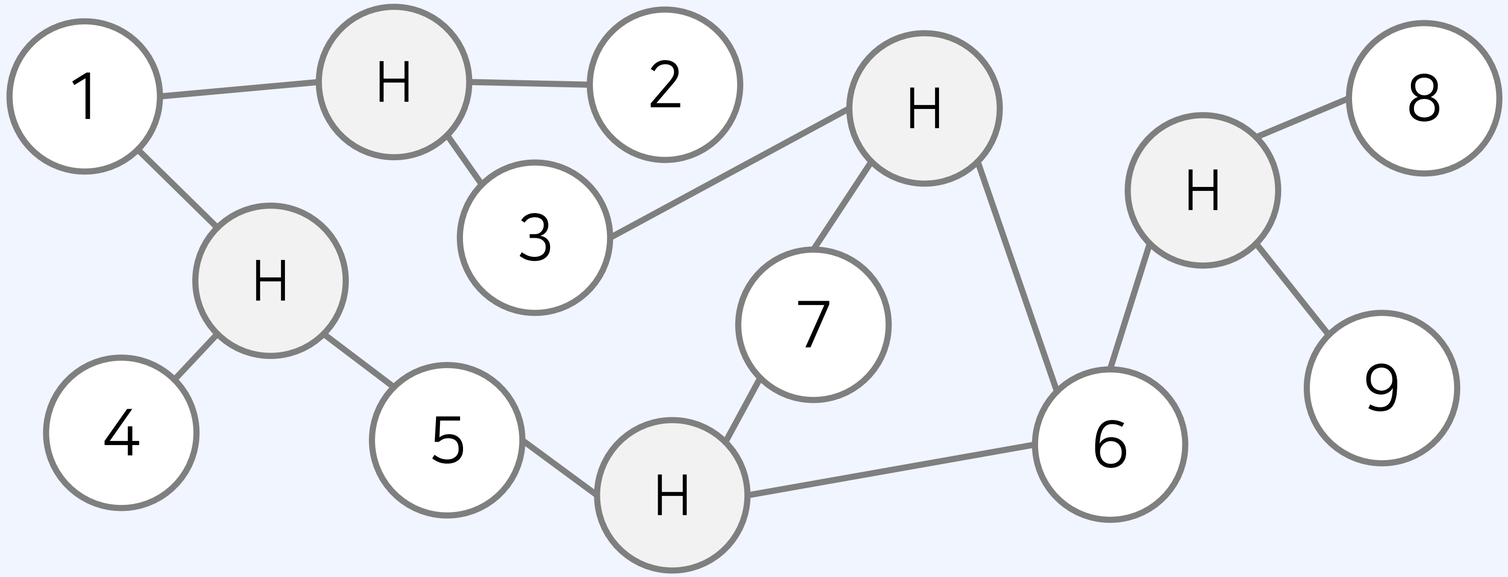
최단 경로: ① → H → ③ → H → ⑤

따라서, **거쳐 가는 역의 수**는 3이다.

환승: https://www.acmicpc.net/problem/5214

문제 풀이 핵심 아이디어

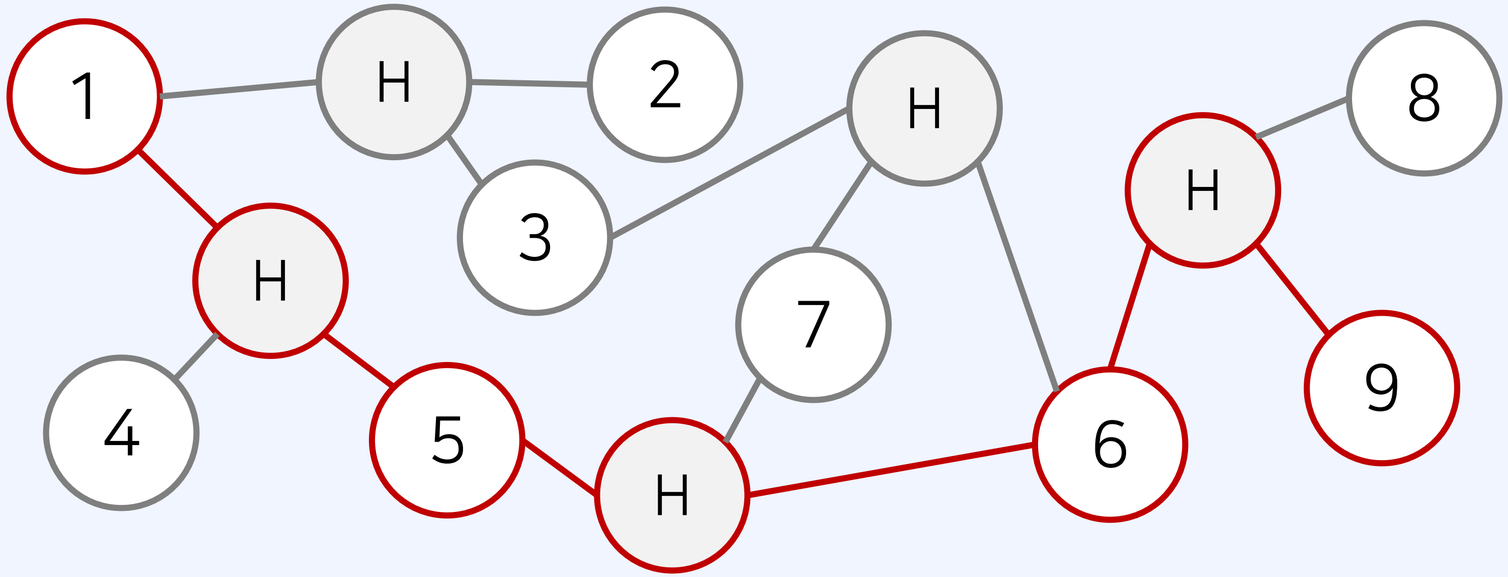
- 1번에서 N번까지 가는데 방문하는 **최소 역의 수**를 구해야 한다.
- [핵심 아이디어] 하이퍼튜브를 노드로 간주하고, **너비 우선 탐색(BFS)**을 사용한다.
- 각 하이퍼튜브와 하이퍼튜브가 연결하는 노드들을 모두 양방향 간선으로 연결한다.



환승: https://www.acmicpc.net/problem/5214

문제 풀이 핵심 아이디어

- 1번 노드에서부터 **너비 우선 탐색(BFS)을 수행**한다.
  - N번 노드까지의 거리가 $distance$일 때 정답은 $distance$ // 2 + 1이다.
  - 아래 예시에서 **정답은 4**이다.



환승: https://www.acmicpc.net/problem/5214

문제 풀이 핵심 아이디어


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

# 역의 개수(N), 간선의 개수(K), 하이퍼튜브의 개수(M)
n, k, m = map(int, input().split())
# 그래프 정보(N개의 역과 M개의 하이퍼튜브는 모두 노드)
graph = [[] for _ in range(n + m + 1)]
for i in range(1, m + 1):
    arr = list(map(int, input().split()))
    for x in arr:
        # 하이퍼튜브의 노드 번호는 N + 1번부터 출발
        graph[x].append(n + i) # 노드 → 하이퍼 튜브
        graph[n + i].append(x) # 하이퍼 튜브 → 노드
```

```python
visited = set([1]) # 1번 노드에서 출발
queue = deque([(0, 1)]) # (거리, 노드 번호)
found = False
while queue: # 큐가 빌 때까지 반복
    cost, now = queue.popleft()
    if now == n: # N번 노드에 도착한 경우
        print(cost // 2 + 1) # 절반은 하이퍼 튜브
        found = True
        break
    for y in graph[now]: # 인접한 노드를 하나씩 확인하며
        if y not in visited: # 아직 방문하지 않았다면
            queue.append((cost + 1, y))
            visited.add(y) # 방문 처리

# N번 노드에 도달이 불가능한 경우
if not found:
    print(-1)
```