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

결혼식: https://www.acmicpc.net/problem/5567

혼자 힘으로 풀어보기

문제 제목: 결혼식

문제 난이도: ★★☆☆☆

문제 유형: BFS, 최단 거리

추천 풀이 시간: 40분

결혼식: https://www.acmicpc.net/problem/5567

문제 풀이 핵심 아이디어

- 문제의 요구사항은 1번 학생의 "친구"와 "친구의 친구"의 수를 출력하는 것이다.

  [해결 방법] 그래프로 표현한 뒤에 거리가 2 이하인 노드의 수를 출력한다.
- 너비 우선 탐색(BFS)을 사용하면, 각 노드까지의 최단 거리를 구할 수 있다.
- 결과적으로 BFS 호출 이후에 **최단 거리가 2 이하**인 노드의 수를 계산하면 된다.

결혼식: https://www.acmicpc.net/problem/5567

문제 풀이 핵심 아이디어

- 노드의 개수(N)는 최대 500이다.
- 간선의 개수(M)는 최대 10,000이다.
- 기본적인 BFS를 이용하여 문제를 해결할 수 있다.

결혼식: https://www.acmicpc.net/problem/5567

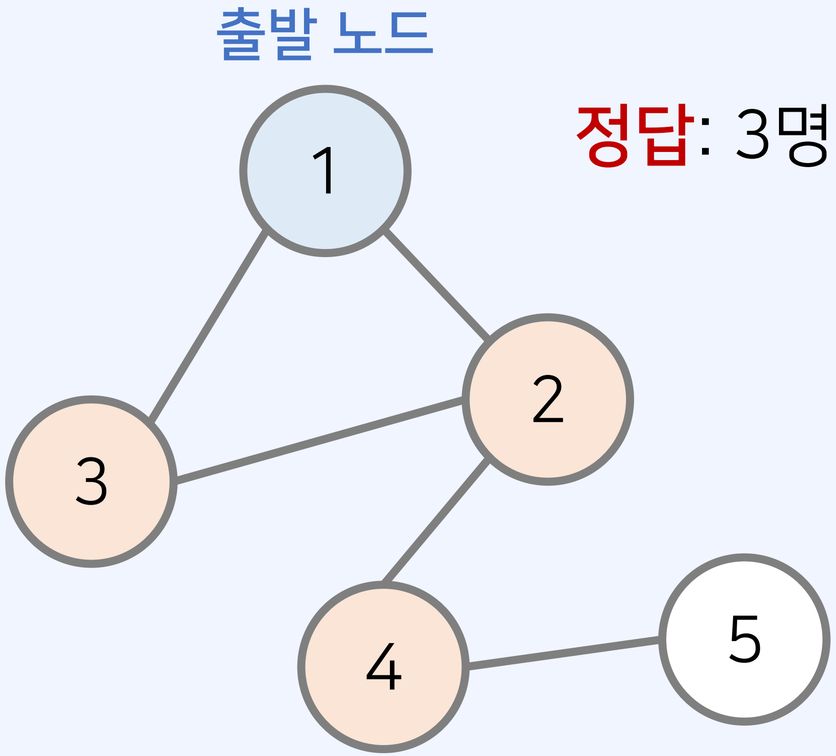
문제 풀이 핵심 아이디어

- ① **자신의 친구**와 ② **친구의 친구**의 수를 세는 문제입니다.
- 따라서 가장 먼저 자신을 기준으로 하여 너비 우선 탐색(BFS)을 수행합니다.
- 이후에 최단 거리가 2 이하인 사람의 수를 계산합니다.

결혼식: https://www.acmicpc.net/problem/5567

문제 풀이 핵심 아이디어

- 문제의 예시를 그래프로 표현하면 다음과 같다.



정답: 3명

결혼식: https://www.acmicpc.net/problem/5567

소스 코드


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

n = int(input()) # 학생(노드)의 수
m = int(input()) # 친구 관계(간선)의 수
# 양방향 그래프 정보 입력
graph = [[] for _ in range(n + 1)]
for _ in range(m):
    a, b = map(int, input().split())
    graph[a].append(b)
    graph[b].append(a)

# 모든 친구(노드)에 대한 최단 거리 초기화
distance = [-1] * (n + 1)
distance[1] = 0 # 시작점까지의 거리는 0으로 설정
```

```python
# 너비 우선 탐색(BFS) 수행
queue = deque([1])
while len(queue) != 0: # 큐가 빌 때까지 반복
    x = queue.popleft()
    # 현재 노드에서 이동할 수 있는 모든 노드를 확인
    for y in graph[x]:
        if distance[y] == -1: # 방문하지 않은 노드라면
            distance[y] = distance[x] + 1
            queue.append(y)

# 최단 거리가 2 이하인 모든 친구(노드)의 수를 계산
result = 0
for i in range(1, n + 1):
    if distance[i] != -1 and distance[i] <= 2:
        result += 1
print(result - 1) # 자기 자신은 제외
```

미로만들기: https://www.acmicpc.net/problem/2665

혼자 힘으로 풀어보기

문제 제목: 미로만들기

문제 난이도: ★★★☆☆

문제 유형: BFS, 최단 거리

추천 풀이 시간: 60분

미로만들기: https://www.acmicpc.net/problem/2665

문제 풀이 핵심 아이디어

- $N \times N$ 바둑판 모양으로 총 N^2개의 방이 있다. (N은 최대 50)
- 일부분은 검은 방이고, 나머지는 모두 흰 방이다.

  이때, 흰 방을 통해서만 이동할 수 있으며, 검은 방으로 막혀 있을 수 있다.
- (1, 1)에서 출발하여 (N, N)까지 최단 거리로 도달하는 것이 목적이다.
- 이때, **검은 방에서 흰 방으로 바꾸어야 할 최소의 수**를 계산해야 한다.

미로만들기: https://www.acmicpc.net/problem/2665

문제 풀이 핵심 아이디어

- 문제에서의 예시는 다음과 같다. (검은 방: 0, 흰 방: 1)

1	1	1	0	0	1	1	0
1	1	0	1	0	0	1	0
1	0	0	1	1	0	1	0
1	1	1	0	1	1	0	0
0	1	0	0	0	1	1	1
0	0	1	1	0	0	0	1
1	1	0	1	1	0	0	0
1	1	0	0	0	1	1	1

미로만들기: https://www.acmicpc.net/problem/2665

문제 풀이 핵심 아이디어

- (4, 4)의 방과 (7, 8)의 방을 흰 방으로 바꾸면 시작 방에서 끝 방으로 갈 수 있다.

1	1	1	0	0	1	1	0
1	1	0	1	0	0	1	0
1	0	0	1	1	0	1	0
1	1	1	0	1	1	0	0
0	1	0	0	0	1	1	1
0	0	1	1	0	0	0	1
1	1	0	1	1	0	0	0
1	1	0	0	0	1	1	1

미로만들기: https://www.acmicpc.net/problem/2665

문제 풀이 핵심 아이디어

- 본 문제는 **검은 방을 없애는 최소 횟수**를 구하므로, BFS를 이용해 해결할 수 있다.
- 각 위치를 (검은 방을 없앤 횟수, x, y)의 노드 형태로 기록한다.
- 단, 검은 방을 없앤 횟수가 낮을수록 높은 우선순위를 가진다.
- 따라서 **큐(queue) 대신에 힙(heap)을 사용하여 BFS**를 하면 문제를 해결할 수 있다.

미로만들기: https://www.acmicpc.net/problem/2665

문제 풀이 핵심 아이디어

- BFS를 사용할 때는 이미 한 번 방문한 상태를 다시 방문하지 않아야 한다.
- 일반적인 BFS에서는 방문 여부를 $visited[x][y]$ 형태로 체크한다.
  결과적으로 똑같은 곳을 여러 번 방문하지 않게 된다.
- 본 문제에서는 (검은 방을 없앤 횟수, x, y)를 **우선순위 큐에 삽입**한다.
  단, **힙(heap)**을 우선순위 큐 대신에 사용하면 최적의 해를 보장할 수 있다.

미로만들기: https://www.acmicpc.net/problem/2665

소스 코드


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

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

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

# 방문 처리 배열
visited = [[-1] * n for _ in range(n)]
# 너비 우선 탐색(BFS) 수행
queue = []
# (검은 방을 없앤 횟수, x, y)를 큐에 삽입
heapq.heappush(queue, (0, 0, 0))
visited[0][0] = 0 # 검은 방을 없앤 횟수
while queue: # 큐가 빌 때까지 반복
    count, x, y = heapq.heappop(queue)
    for i in range(4):
        nx = x + dx[i]
        ny = y + dy[i]
        # 맵을 벗어나는 경우 무시
        if nx < 0 or nx >= n or ny < 0 or ny >= n:
            continue
        # 방문하지 않은 위치라면
        if visited[nx][ny] == -1:
            # 검은 방인 경우 카운트를 증가해서 삽입
            if graph[nx][ny] == 0:
                visited[nx][ny] = count + 1
                heapq.heappush(queue, (count + 1, nx, ny))
            # 흰 방인 경우 카운트를 그대로 삽입
            else:
                visited[nx][ny] = count
                heapq.heappush(queue, (count, nx, ny))

print(visited[n - 1][n - 1])
```

치즈: https://www.acmicpc.net/problem/2638

혼자 힘으로 풀어보기

문제 제목: 치즈

문제 난이도: ★★★☆☆

문제 유형: BFS

추천 풀이 시간: 60분

치즈: https://www.acmicpc.net/problem/2638

문제 풀이 핵심 아이디어

- 전체 맵은 $N \times M$ 형태를 가진다. (N과 M은 최대 100)
- 한 시간마다 어떤 위치의 치즈를 녹이게 될 지 일일이 계산하면 어떨까?
- BFS로 녹일 위치를 선택하고, **녹이는 과정을 반복 수행**하여 본 문제를 해결할 수 있다.

치즈: https://www.acmicpc.net/problem/2638

문제 풀이 핵심 아이디어

- 어떤 위치의 치즈를 녹여야 할까?
- 치즈 내부의 공간은 치즈 외부 공기와 접촉하지 않는 것으로 간주한다.

  따라서, 단순히 주변이 0인지를 확인하는 방식으로는 해결할 수 없다.
- 즉, 외부 공기를 정확히 파악하기 위해 **(0, 0)의 위치에서 출발**하여 BFS를 진행한다.

  가장자리에는 치즈가 없기 때문이다.

치즈: https://www.acmicpc.net/problem/2638

문제 풀이 핵심 아이디어

[BFS 파트]

- (0, 0)의 위치에서 출발하여 BFS를 진행한다.
- 큐에서 하나의 원소를 꺼낸 뒤에는 상, 하, 좌, 우 위치를 확인한다.

  인접한 위치에 치즈가 있다면, **치즈가 있는 위치에 대하여 카운트(count)**한다.

[녹이기 파트]

- 카운트가 2 이상인 치즈를 없앤다.

치즈: https://www.acmicpc.net/problem/2638

소스 코드 1)


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

# 맵의 크기(N과 M) 정보 입력
n, m = map(int, input().split())
# 2차원 맵 입력받기
graph = []
for i in range(n):
    row = list(map(int, input().split()))
    graph.append(row)

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

```python
def bfs():
    # 방문 처리 배열
    visited = [[False] * m for _ in range(n)]
    # 제일 왼쪽 위에서 출발
    visited[0][0] = True
    queue = deque([(0, 0)])
    while queue: # 큐가 빌 때까지 반복
        x, y = queue.popleft()
        for i in range(4):
            nx = x + dx[i]
            ny = y + dy[i]
            # 맵을 벗어나는 경우 무시
            if nx < 0 or nx >= n or ny < 0 or ny >= m:
                continue
            if not visited[nx][ny]:
                if graph[nx][ny] >= 1: # 치즈라면
                    graph[nx][ny] += 1 # 카운트 증가
                else: # 빈 공간이라면
                    queue.append((nx, ny))
                    visited[nx][ny] = True
```

치즈: https://www.acmicpc.net/problem/2638

소스 코드 2)


```
def melt():
    finish = True # 더 녹일 치즈가 없는지 여부
    for i in range(n):
        for j in range(m):
            if graph[i][j] >= 3: # 녹일 치즈라면
                graph[i][j] = 0 # 녹이기
                finish = False
            elif graph[i][j] == 2: # 한 면만 닿은 치즈
                graph[i][j] = 1 # 무시
    return finish


result = 0
while True:
    bfs()
    if melt(): # 전부 녹았다면
        print(result)
        break
    else:
        result += 1
```