

188. A game on an undirected graph is played by two players, Mouse and Cat, who alternate turns. The graph is given as follows: graph[a] is a list of all nodes b such that ab is an edge of the graph. The mouse starts at node 1 and goes first, the cat starts at node 2 and goes second, and there is a hole at node 0. During each player's turn, they must travel along one edge of the graph that meets where they are. For example, if the Mouse is at node 1, it must travel to any node in graph[1]. Additionally, it is not allowed for the Cat to travel to the Hole (node 0). Then, the game can end in three ways:

If ever the Cat occupies the same node as the Mouse, the Cat wins.

If ever the Mouse reaches the Hole, the Mouse wins.

If ever a position is repeated (i.e., the players are in the same position as a previous turn, and it is the same player's turn to move), the game is a draw.

Given a graph, and assuming both players play optimally, return

Program:from collections import deque

```
def catMouseGame(graph):
    n = len(graph)

    # State: (mouse position, cat position, turn)
    # turn: 1 for Mouse's turn, 2 for Cat's turn
    # Status: 0 for draw, 1 for mouse wins, 2 for cat wins

    dp = [[[0] * 3 for _ in range(n)] for _ in range(n)]

    # Initialize queue for BFS
    queue = deque()

    # Initialize mouse positions where mouse wins directly
    for i in range(1, n):
        dp[0][i][1] = 1 # Mouse wins if it reaches hole (node 0)
        queue.append((0, i, 1))

    dp[i][i][2] = 2 # Cat wins if it can catch the mouse
    queue.append((i, i, 2))

    # BFS to propagate game states
```

```

while queue:
    m, c, turn = queue.popleft()

    # Previous turn
    prev_turn = 3 - turn

    # Get all previous positions from current position
    for prev in graph[prev_turn == 1 and m or c]:
        if dp[prev][c][prev_turn] == 0: # not visited
            if prev_turn == 1: # Mouse's turn
                dp[prev][c][prev_turn] = 1
            else: # Cat's turn
                dp[m][prev][prev_turn] = 2

        queue.append((prev, c, prev_turn))

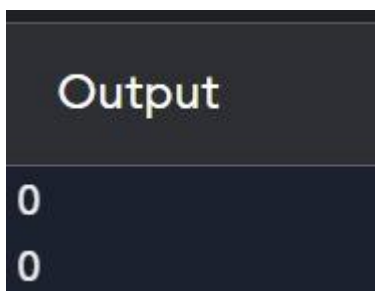
    return dp[1][2][1] # Result for mouse starting at node 1 and cat at node 2

# Example 1
graph1 = [[2,5],[3],[0,4,5],[1,4,5],[2,3],[0,2,3]]
print(catMouseGame(graph1)) # Output: 0

# Example 2
graph2 = [[1,3],[0],[3],[0,2]]
print(catMouseGame(graph2)) # Output: 1

```

Output:



```

Output
0
0

```

Timecomplexity: $O((V+E)\log v)$