from collections import defaultdict

class Graph:

def \_\_init\_\_(self):

self.graph = defaultdict(list)

def add\_edge(self, u, v):

self.graph[u].append(v)

self.graph[v].append(u) # Assuming an undirected graph

def iterative\_dfs(self, start, end):

if start == end:

return [start]

visited = set()

stack = [(start, [start])]

while stack:

current\_vertex, path = stack.pop()

visited.add(current\_vertex)

for neighbor in self.graph[current\_vertex]:

if neighbor not in visited:

if neighbor == end:

return path + [neighbor]

stack.append((neighbor, path + [neighbor]))

return None # No path found

# Example usage:

if \_\_name\_\_ == "\_\_main\_\_":

g = Graph()

g.add\_edge(1, 2)

g.add\_edge(1, 3)

g.add\_edge(2, 4)

g.add\_edge(2, 5)

g.add\_edge(3, 6)

g.add\_edge(3, 7)

g.add\_edge(4, 8)

g.add\_edge(4, 9)

g.add\_edge(5, 10)

g.add\_edge(5, 11)

g.add\_edge(6, 12)

g.add\_edge(6, 13)

g.add\_edge(7, 14)

g.add\_edge(7, 15)

start\_node = 1

end\_node = 9

shortest\_path = g.iterative\_dfs(start\_node, end\_node)

if shortest\_path:

print(f"Shortest path from {start\_node} to {end\_node}: {shortest\_path}")

else:

print(f"No path found from {start\_node} to {end\_node}")