ITERATIVE DEEPENING SEARCH ALGORITHM

class Node:

def \_\_init\_\_(self, state, parent=None, depth=0):

self.state = state # Current state of the node

self.parent = parent # Parent node

self.depth = depth # Depth of the node in the tree

def path(self):

"""Returns the path from the root to this node."""

node, path\_back = self, []

while node:

path\_back.append(node.state)

node = node.parent

return list(reversed(path\_back))

def depth\_limited\_search(start, goal, graph, limit):

"""

Perform a Depth-Limited Search (DLS) from the start node.

"""

stack = [Node(start)]

while stack:

current\_node = stack.pop()

# If the goal is found

if current\_node.state == goal:

return current\_node

# If the current depth is less than the limit, explore further

if current\_node.depth < limit:

for neighbor in graph[current\_node.state]:

stack.append(Node(neighbor, current\_node, current\_node.depth + 1))

return None # Return None if no solution is found within the limit

def iterative\_deepening\_search(start, goal, graph):

"""

Perform an Iterative Deepening Search (IDS) from the start node.

"""

depth = 0

while True:

print(f"Searching at depth limit: {depth}")

result = depth\_limited\_search(start, goal, graph, depth)

if result: # If a solution is found

return result.path()

depth += 1 # Increment the depth limit

# Example usage

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

# Graph represented as an adjacency list

graph = {

'A': ['B', 'C'],

'B': ['D', 'E'],

'C': ['F', 'G'],

'D': [],

'E': [],

'F': [],

'G': []

}

start = 'A'

goal = 'F'

path = iterative\_deepening\_search(start, goal, graph)

if path:

print(f"Path to goal: {path}")

else:

print("Goal not found.")

