WEEK 04

8 PUZZLE USING ITERATIVE DEEPENING DEPTH FIRST SEARCH

class PuzzleNode:

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

        self.state = state

        self.parent = parent

        self.action = action

    def get\_path(self): # Not required

        path = []

        current = self

        while current:

            path.append((current.state, current.action))

            current = current.parent

        return path[::-1]

def is\_goal(state):

    goal\_state = (1,2,3,6,4,5,0,7,8)

    return state == goal\_state

def get\_neighbors(state):

    neighbors = []

    empty\_index = state.index(0)

    row, col = divmod(empty\_index, 3)

    for move in [(0, 1), (1, 0), (0, -1), (-1, 0)]:

        new\_row, new\_col = row + move[0], col + move[1]

        if 0 <= new\_row < 3 and 0 <= new\_col < 3:

            neighbor\_state = list(state)

            neighbor\_index = new\_row \* 3 + new\_col

            neighbor\_state[empty\_index], neighbor\_state[neighbor\_index] = (

                neighbor\_state[neighbor\_index],

                neighbor\_state[empty\_index],

            )

            neighbors.append(tuple(neighbor\_state))

    return neighbors

def depth\_limited\_search(node, goal\_state, depth\_limit):

    if is\_goal(node.state):

        return True

    elif depth\_limit == 0:

        return False

    else:

        for neighbor\_state in get\_neighbors(node.state):

            child = PuzzleNode(neighbor\_state, node)

            if depth\_limited\_search(child, goal\_state, depth\_limit - 1):

                return True

        return False

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

    initial\_state = (1, 2, 3, 0, 4, 5, 6, 7, 8)

    depth\_limit = 1  # Set the depth limit as needed

    initial\_node = PuzzleNode(initial\_state)

    result = depth\_limited\_search(initial\_node, (1,2,3,6,4,5,0,7,8), depth\_limit)

    print(result)

OUTPUT:

