8-PUZZLE

from collections import deque

# Class representing the 8 Puzzle problem

class Puzzle8:

def \_\_init\_\_(self, size=3):

self.size = size

def display\_state(self, state):

for i in range(self.size):

for j in range(self.size):

print(state[i \* self.size + j], end=" ")

print()

def get\_blank\_index(self, state):

return state.index(-1)

def get\_neighbors(self, state):

neighbors = []

blank\_index = self.get\_blank\_index(state)

row, col = divmod(blank\_index, self.size)

moves = [(0, 1), (1, 0), (0, -1), (-1, 0)] # right, down, left, up

for move in moves:

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

if 0 <= new\_row < self.size and 0 <= new\_col < self.size:

new\_state = state[:]

new\_blank\_index = new\_row \* self.size + new\_col

# Swap the blank tile with the neighbor

new\_state[blank\_index], new\_state[new\_blank\_index] = new\_state[new\_blank\_index], new\_state[blank\_index]

neighbors.append(new\_state)

return neighbors

def is\_goal\_state(self, state, target\_state):

return state == target\_state

def bfs(self, initial\_state, target\_state):

queue = deque([(initial\_state, [])])

visited = set()

while queue:

current\_state, path = queue.popleft()

if self.is\_goal\_state(current\_state, target\_state):

return path

if tuple(current\_state) not in visited:

visited.add(tuple(current\_state))

neighbors = self.get\_neighbors(current\_state)

for neighbor in neighbors:

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

return None

# Example usage:

initial\_state = [1, 2, 3, 4, -1, 5, 6, 7, 8]

goal\_state = [1, 2, 3, 4, 5, 6, 7, 8, -1]

puzzle = Puzzle8()

solution = puzzle.bfs(initial\_state, goal\_state)

if solution:

print("Solution found:")

for step, state in enumerate(solution):

print(f"Step {step + 1}:")

puzzle.display\_state(state)

print()

else:

print("No solution found.")

OUTPUT:

