1. 8-puzzle problem

from collections import deque

class Puzzle:

def \_\_init\_\_(self, start, goal):

self.start = start

self.goal = goal

def is\_valid(self, x, y):

return 0 <= x < 3 and 0 <= y < 3

def get\_neighbors(self, state):

neighbors = []

x, y = [(ix, iy) for ix, row in enumerate(state) for iy, i in enumerate(row) if i == 0][0]

directions = [(-1, 0), (1, 0), (0, -1), (0, 1)]

for dx, dy in directions:

nx, ny = x + dx, y + dy

if self.is\_valid(nx, ny):

new\_state = [row[:] for row in state]

new\_state[x][y], new\_state[nx][ny] = new\_state[nx][ny], new\_state[x][y]

neighbors.append(new\_state)

return neighbors

def bfs(self):

queue = deque([(self.start, [])])

visited = set()

while queue:

state, path = queue.popleft()

if state == self.goal:

return path

state\_tuple = tuple(tuple(row) for row in state)

if state\_tuple in visited:

continue

visited.add(state\_tuple)

for neighbor in self.get\_neighbors(state):

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

return None

def print\_puzzle(state):

for row in state:

print(" ".join(str(i) for i in row))

print()

start\_state = [[1, 2, 3], [4, 0, 5], [6, 7, 8]]

goal\_state = [[1, 2, 3], [4, 5, 6], [7, 8, 0]]

puzzle = Puzzle(start\_state, goal\_state)

solution = puzzle.bfs()

if solution:

print("Steps to solve the puzzle:")

for step in solution:

print\_puzzle(step)

else:

print("No solution found")

output:

