**EXPERIMENT 1**

import heapq

class PuzzleState:

def \_\_init\_\_(self, board, moves=0, previous=None):

self.board = board

self.moves = moves

self.previous = previous

self.zero\_pos = board.index(0)

def is\_goal(self):

return self.board == [1, 2, 3, 4, 5, 6, 7, 8, 0]

def get\_neighbors(self):

neighbors = []

x, y = divmod(self.zero\_pos, 3)

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

for dx, dy in directions:

nx, ny = x + dx, y + dy

if 0 <= nx < 3 and 0 <= ny < 3:

new\_zero = nx \* 3 + ny

new\_board = self.board[:]

new\_board[self.zero\_pos], new\_board[new\_zero] = new\_board[new\_zero], new\_board[self.zero\_pos]

neighbors.append(PuzzleState(new\_board, self.moves + 1, self))

return neighbors

def manhattan\_distance(self):

distance = 0

for i, val in enumerate(self.board):

if val == 0:

continue

target\_x, target\_y = divmod(val - 1, 3)

current\_x, current\_y = divmod(i, 3)

distance += abs(target\_x - current\_x) + abs(target\_y - current\_y)

return distance

def \_\_lt\_\_(self, other):

return (self.moves + self.manhattan\_distance()) < (other.moves + other.manhattan\_distance())

def solve\_puzzle(start\_board):

start\_state = PuzzleState(start\_board)

frontier = []

heapq.heappush(frontier, start\_state)

explored = set()

while frontier:

current = heapq.heappop(frontier)

if current.is\_goal():

return reconstruct\_path(current)

explored.add(tuple(current.board))

for neighbor in current.get\_neighbors():

if tuple(neighbor.board) not in explored:

heapq.heappush(frontier, neighbor)

def reconstruct\_path(state):

path = []

while state:

path.append(state.board)

state = state.previous

return path[::-1]

initial\_board = [1, 2, 3, 4, 0, 6, 7, 5, 8]

solution = solve\_puzzle(initial\_board)

print("Steps to solve the puzzle:")

for step in solution:

for i in range(0, 9, 3):

print(step[i:i+3])

print()

**OUTPUT:**

