import heapq

# Goal state

GOAL\_STATE = (1, 2, 3,

4, 5, 6,

7, 8, 0)

# Directions: up, down, left, right

MOVES = {

'up': -3,

'down': 3,

'left': -1,

'right': 1

}

class PuzzleState:

def \_\_init\_\_(self, board, parent=None, move='', g=0):

self.board = board

self.parent = parent

self.move = move

self.g = g # Cost from start to current state

self.h = self.heuristic() # Estimated cost to goal

self.f = self.g + self.h # Total estimated cost

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

return self.f < other.f

def heuristic(self):

# Manhattan distance

distance = 0

for idx, value in enumerate(self.board):

if value == 0:

continue

target\_idx = value - 1

x1, y1 = divmod(idx, 3)

x2, y2 = divmod(target\_idx, 3)

distance += abs(x1 - x2) + abs(y1 - y2)

return distance

def get\_neighbors(self):

neighbors = []

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

for move, pos\_change in MOVES.items():

new\_index = zero\_index + pos\_change

if self.is\_valid\_move(zero\_index, new\_index, move):

new\_board = list(self.board)

new\_board[zero\_index], new\_board[new\_index] = new\_board[new\_index], new\_board[zero\_index]

neighbors.append(PuzzleState(tuple(new\_board), self, move, self.g + 1))

return neighbors

def is\_valid\_move(self, zero\_index, new\_index, move):

# Check if move is within bounds

if new\_index < 0 or new\_index >= 9:

return False

if move == 'left' and zero\_index % 3 == 0:

return False

if move == 'right' and zero\_index % 3 == 2:

return False

return True

def get\_path(self):

path = []

state = self

while state.parent:

path.append((state.move, state.board))

state = state.parent

path.reverse()

return path

def a\_star(start\_board):

start\_state = PuzzleState(start\_board)

open\_list = []

heapq.heappush(open\_list, start\_state)

closed\_set = set()

while open\_list:

current\_state = heapq.heappop(open\_list)

if current\_state.board == GOAL\_STATE:

return current\_state.get\_path()

closed\_set.add(current\_state.board)

for neighbor in current\_state.get\_neighbors():

if neighbor.board not in closed\_set:

heapq.heappush(open\_list, neighbor)

return None

# ---------------------

# Example Usage

# ---------------------

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

# Define start state (0 is the blank)

start\_board = (1, 2, 3,

4, 0, 6,

7, 5, 8)

path = a\_star(start\_board)

if path:

print(f"Solution found in {len(path)} moves:")

print("Start:")

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

print(start\_board[i:i+3])

print()

for move, board in path:

print(f"Move: {move}")

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

print(board[i:i+3])

print()

else:

print("No solution found.")

