import heapq

class PuzzleState:

def \_\_init\_\_(self, board, parent=None, move="", depth=0):

self.board = board

self.parent = parent

self.move = move

self.depth = depth

self.zero\_pos = self.find\_zero()

self.cost = self.depth + self.manhattan\_distance()

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

return self.cost < other.cost

def find\_zero(self):

for i in range(3):

for j in range(3):

if self.board[i][j] == 0:

return i, j

def manhattan\_distance(self):

distance = 0

goal = {

1: (0, 0), 2: (0, 1), 3: (0, 2),

4: (1, 0), 5: (1, 1), 6: (1, 2),

7: (2, 0), 8: (2, 1)

}

for i in range(3):

for j in range(3):

val = self.board[i][j]

if val != 0:

goal\_i, goal\_j = goal[val]

distance += abs(goal\_i - i) + abs(goal\_j - j)

return distance

def generate\_neighbors(self):

neighbors = []

x, y = self.zero\_pos

moves = {

"Up": (x - 1, y),

"Down": (x + 1, y),

"Left": (x, y - 1),

"Right": (x, y + 1)

}

for move, (nx, ny) in moves.items():

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

new\_board = [row[:] for row in self.board]

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

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

return neighbors

def is\_goal(self):

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

return self.board == goal

def get\_path(self):

path = []

current = self

while current.parent is not None:

path.append(current.move)

current = current.parent

return path[::-1] # reverse

def board\_tuple(self):

return tuple(tuple(row) for row in self.board)

def a\_star\_search(start\_board):

start\_state = PuzzleState(start\_board)

open\_set = []

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

visited = set()

while open\_set:

current = heapq.heappop(open\_set)

if current.is\_goal():

return current.get\_path()

visited.add(current.board\_tuple())

for neighbor in current.generate\_neighbors():

if neighbor.board\_tuple() not in visited:

heapq.heappush(open\_set, neighbor)

return None

# Example usage

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

# 0 represents the blank space

start = [

[2, 8, 3],

[1, 6, 4],

[7, 0, 5]

]

print("Solving 8-puzzle problem using A\* search...")

solution = a\_star\_search(start)

if solution:

print("Solution found in", len(solution), "moves:")

print(" -> ".join(solution))

else:

print("No solution found.")

output:

