17.Write the python program to solve 8-Puzzle problem

import heapq

# Goal state for reference

GOAL\_STATE = [[1, 2, 3],

[4, 5, 6],

[7, 8, 0]]

# Directions for moving the blank tile

DIRECTIONS = {'Up': (-1, 0), 'Down': (1, 0), 'Left': (0, -1), 'Right': (0, 1)}

class PuzzleNode:

def \_\_init\_\_(self, state, parent=None, move=None, depth=0, cost=0):

self.state = state

self.parent = parent

self.move = move

self.depth = depth

self.cost = cost

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

return self.cost < other.cost

def manhattan\_distance(state):

distance = 0

for i in range(3):

for j in range(3):

val = state[i][j]

if val != 0:

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

distance += abs(goal\_x - i) + abs(goal\_y - j)

return distance

def get\_blank\_position(state):

for i in range(3):

for j in range(3):

if state[i][j] == 0:

return i, j

def generate\_successors(node):

successors = []

x, y = get\_blank\_position(node.state)

for move, (dx, dy) in DIRECTIONS.items():

new\_x, new\_y = x + dx, y + dy

if 0 <= new\_x < 3 and 0 <= new\_y < 3:

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

new\_state[x][y], new\_state[new\_x][new\_y] = new\_state[new\_x][new\_y], new\_state[x][y]

new\_node = PuzzleNode(new\_state, node, move, node.depth + 1)

new\_node.cost = new\_node.depth + manhattan\_distance(new\_state)

successors.append(new\_node)

return successors

def is\_goal(state):

return state == GOAL\_STATE

def reconstruct\_path(node):

path = []

while node.parent is not None:

path.append(node.move)

node = node.parent

return path[::-1]

def a\_star(start\_state):

start\_node = PuzzleNode(start\_state)

start\_node.cost = manhattan\_distance(start\_state)

open\_set = []

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

visited = set()

while open\_set:

current\_node = heapq.heappop(open\_set)

state\_id = str(current\_node.state)

if state\_id in visited:

continue

visited.add(state\_id)

if is\_goal(current\_node.state):

return reconstruct\_path(current\_node)

for successor in generate\_successors(current\_node):

if str(successor.state) not in visited:

heapq.heappush(open\_set, successor)

return None

# Example usage:

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

# You can change the initial state here

initial\_state = [[1, 2, 3],

[4, 0, 5],

[6, 7, 8]]

solution = a\_star(initial\_state)

if solution:

print("Steps to solve the puzzle:")

for step in solution:

print(step)

print(f"Total moves: {len(solution)}")

else:

print("No solution found.")

OUTPUT:

