```
import heapq

class PuzzleState:
    def __init__(self, state, parent=None, move=None, cost=0):
        self.state = state
        self.parent = parent
        self.move = move
        self.cost = cost
        self.heuristic = self.calculate_heuristic()

    def __lt__(self, other):
        return (self.cost + self.heuristic) < (other.cost + other.heuristic)

    def __eq__(self, other):
        return self.state == other.state

    def __hash__(self):
        return hash(tuple(self.state))

    def find_blank(self):
        return self.state.index(0)

    def calculate_heuristic(self):
        # Manhattan distance heuristic
        distance = 0
        goal_state = [1, 2, 3, 4, 5, 6, 7, 8, 0]
        for i, value in enumerate(self.state):
            if value != 0:
                goal_pos = goal_state.index(value)
                distance += abs(i % 3 - goal_pos % 3) + abs(i // 3 - goal_pos // 3)
        return distance

    def generate_neighbors(self):
        blank_pos = self.find_blank()
        neighbors = []
        moves = [(-1, 0), (1, 0), (0, -1), (0, 1)]  # Up, down, left, right
        for move in moves:
            new_row, new_col = blank_pos // 3 + move[0], blank_pos % 3 + move[1]
            if 0 <= new_row < 3 and 0 <= new_col < 3:
                new_blank_pos = new_row * 3 + new_col
                new_state = self.state[:]
                new_state[blank_pos], new_state[new_blank_pos] = new_state[new_blank_pos], 
                neighbors.append(PuzzleState(new_state, self, move, self.cost + 1))
        return neighbors

def solve_8_puzzle(initial_state):
    initial_puzzle = PuzzleState(initial_state)
    goal_state = [1, 2, 3, 4, 5, 6, 7, 8, 0]
    goal_puzzle = PuzzleState(goal_state)

    if initial_puzzle == goal_puzzle:
        return [initial_puzzle]

    open_list = []
    heapq.heappush(open_list, initial_puzzle)
    closed_set = set()

    while open_list:
        current_puzzle = heapq.heappop(open_list)

        if current_puzzle.state == goal_state:
            path = []
            while current_puzzle:
                path.append(current_puzzle)
                current_puzzle = current_puzzle.parent
            return path[::-1]

        closed_set.add(current_puzzle)

        for neighbor in current_puzzle.generate_neighbors():
            if neighbor not in closed_set:
                heapq.heappush(open_list, neighbor)

    return None  # No solution found

def get_user_input():
    print("Enter the initial state of the 8-puzzle (use 0 for the blank space):")
    print("Example input format: 2 8 3 1 6 4 7 0 5")

    while True:
        user_input = input("Enter 9 numbers separated by spaces: ")
        numbers = user_input.split()

        if len(numbers) != 9:
            print("Please enter exactly 9 numbers.")
            continue

        try:
            numbers = [int(num) for num in numbers]
        except ValueError:
            print("Please enter numbers only.")
            continue

        if sorted(numbers) != list(range(9)):
            print("Please use each digit from 0 to 8 exactly once.")
            continue

        return numbers

def main():
    initial_state = get_user_input()
    solution = solve_8_puzzle(initial_state)

    if solution:
        print("\nSolution found! Here are the steps:")
        for step, puzzle in enumerate(solution):
            print(f"\nStep {step}:")
            if step > 0:
                move = puzzle.move
                if move == (-1, 0):
                    print("Move: UP")
                elif move == (1, 0):
                    print("Move: DOWN")
                elif move == (0, -1):
                    print("Move: LEFT")
                elif move == (0, 1):
                    print("Move: RIGHT")

            for i in range(0, 9, 3):
                print(puzzle.state[i:i + 3])
    else:
        print("\nNo solution exists for this puzzle configuration.")

if __name__ == "__main__":
    main()
```

```
Enter the initial state of the 8-puzzle (use 0 for the blank space):
Example input format: 2 8 3 1 6 4 7 0 5
Enter 9 numbers separated by spaces: 1 2 3 4 5 6 7 0 8

Solution found! Here are the steps:

Step 0:
[1, 2, 3]
[4, 5, 6]
[7, 0, 8]

Step 1:
Move: RIGHT
[1, 2, 3]
[4, 5, 6]
[7, 8, 0]
```