2. Implement the 8 Puzzle Breadth First Search Algorithm.

from queue import Queue

def print\_board(board):

for row in board:

print(" ".join(map(str, row)))

def is\_goal\_state(board, goal):

return board == goal

def get\_blank\_position(board):

for i in range(3):

for j in range(3):

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

return i, j

def generate\_successors(board):

successors = []

blank\_i, blank\_j = get\_blank\_position(board)

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

for move\_i, move\_j in moves:

new\_i, new\_j = blank\_i + move\_i, blank\_j + move\_j

if 0 <= new\_i < 3 and 0 <= new\_j < 3:

successor = [row.copy() for row in board]

successor[blank\_i][blank\_j], successor[new\_i][new\_j] = successor[new\_i][new\_j], successor[blank\_i][blank\_j]

successors.append(successor)

return successors

def breadth\_first\_search(initial\_state, goal\_state):

visited = set()

queue = Queue()

queue.put((initial\_state, []))

while not queue.empty():

current\_state, path = queue.get()

visited.add(tuple(map(tuple, current\_state)))

if is\_goal\_state(current\_state, goal\_state):

return path

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

if tuple(map(tuple, successor)) not in visited:

queue.put((successor, path + [successor]))

return None

# Example:

initial\_state = [

[1, 2, 3],

[0, 4, 6],

[7, 5, 8]

]

goal\_state = [

[1, 2, 3],

[4, 5, 6],

[7, 8, 0]

]

solution = breadth\_first\_search(initial\_state, goal\_state)

if solution:

print("Solution found:")

for step, state in enumerate(solution):

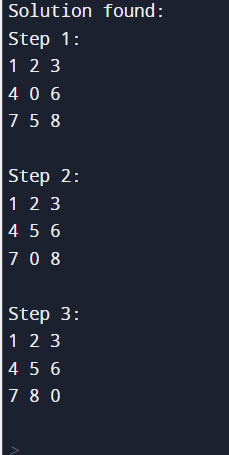
print(f"Step {step + 1}:")

print\_board(state)

print()

else:

print("No solution found.")



Q. Explore the working of Tic Tac Toe using Min max strategy.

**function** minimax(board, depth, isMaximizingPlayer):  
  
 **if** current board state is a terminal state :  
 **return** value of the board  
   
 **if** isMaximizingPlayer :  
 bestVal = -INFINITY   
 **for each** move in board :  
 value = minimax(board, depth+1, false)  
 bestVal = max( bestVal, value)   
 **return** bestVal  
  
 **else** :  
 bestVal = +INFINITY   
 **for each** move in board :  
 value = minimax(board, depth+1, true)  
 bestVal = min( bestVal, value)   
 **return** bestVal