

**Artificial Intelligence Systems**

Lab Report # 03

**Submitted by:** Ali Hamza

**Registration No:** B23F0063AI106

**Submitted to:** Dr. Mohsin

Date: 25th February , 2025

# Lab Task:

# Find solution to the 8 puzzle problem using iterative depth first search.

*"""  
Find solution to the 8 puzzle problem using iterative depth first search.  
"""  
  
from* collections *import* deque  
  
*# Define the goal state*goal\_state = [  
 [1, 2, 3],  
 [4, 5, 6],  
 [7, 8, 0]  
]  
  
*# Define the possible moves*moves = {  
 "up": (-1, 0),  
 "down": (1, 0),  
 "left": (0, -1),  
 "right": (0, 1)  
}  
  
  
*# Function to print the board  
def* print\_board(board):  
 *for* row *in* board:  
 print(' '.join(str(cell) *for* cell *in* row))  
 print()  
  
  
*# Function to find the blank space (zero position)  
def* find\_zero(state):  
 *for* i *in* range(3):  
 *for* j *in* range(3):  
 *if* state[i][j] == 0:  
 *return* i, j  
 *return None  
  
  
# Function to apply a move to the board  
def* apply\_move(state, move):  
 zero\_pos = find\_zero(state)  
 new\_pos = (zero\_pos[0] + moves[move][0], zero\_pos[1] + moves[move][1])  
  
 *# Check if the new position is within bounds  
 if* 0 <= new\_pos[0] < 3 *and* 0 <= new\_pos[1] < 3:  
 new\_state = [row[:] *for* row *in* state] *# Create a copy of the current state  
 # Swap the zero with the target position* new\_state[zero\_pos[0]][zero\_pos[1]], new\_state[new\_pos[0]][new\_pos[1]] = new\_state[new\_pos[0]][new\_pos[1]], \  
 new\_state[zero\_pos[0]][zero\_pos[1]]  
 *return* new\_state  
 *return None  
  
  
# Function to perform depth-limited search  
def* dls(state, depth, visited):  
 *if* state == goal\_state:  
 *return* []  
 *if* depth == 0:  
 *return None* visited.add(tuple(map(tuple, state))) *# Add the current state to visited  
  
 for* move *in* moves:  
 new\_state = apply\_move(state, move)  
 *if* new\_state *and* tuple(map(tuple, new\_state)) *not in* visited:  
 result = dls(new\_state, depth - 1, visited)  
 *if* result *is not None*:  
 *return* [move] + result *# Return the move sequence if found* visited.remove(tuple(map(tuple, state))) *# Backtrack  
 return None  
  
  
# Function to perform iterative deepening depth-first search  
def* iddfs(start\_state):  
 depth = 0  
 *while True*:  
 visited = set()  
 result = dls(start\_state, depth, visited)  
 *if* result *is not None*:  
 *return* result *# Return the solution if found* depth += 1 *# Increase depth limit  
  
  
# Main function to execute the algorithm  
if* \_\_name\_\_ == "\_\_main\_\_":  
 start\_state = [  
 [1, 2, 3],  
 [4, 5, 6],  
 [7, 0, 8]  
 ]  
  
 print("Initial State:")  
 print\_board(start\_state)  
  
 solution = iddfs(start\_state)  
  
 *if* solution:  
 print("Solution found:")  
 *for* move *in* solution:  
 print(f"Move: {move}")  
 start\_state = apply\_move(start\_state, move)  
 print\_board(start\_state)  
 *else*:  
 print("No solution found.")

# 