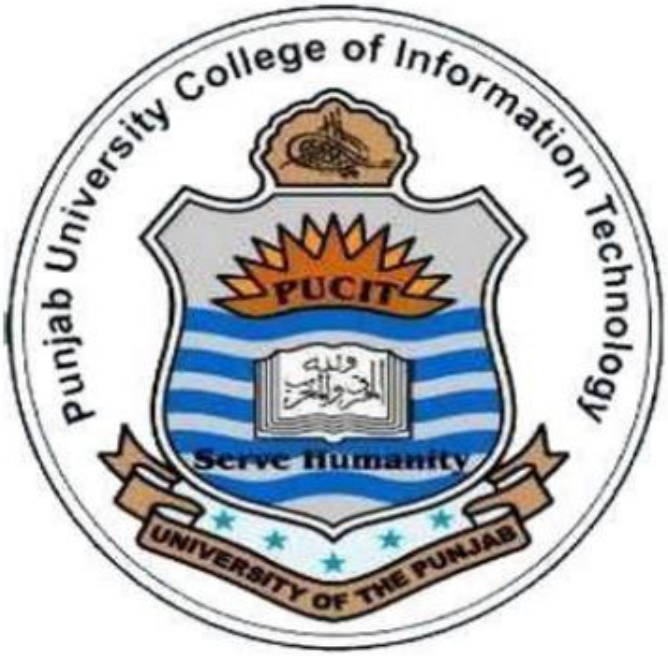

LAB 02

Course: Artificial Intelligence Lab

Submitted to: Dr. Muhammad Rizwan

Submitted by: Bushra Shahbaz

Roll No: BSDS-F21M020

LAB TASK:

Implement Depth first with iterative deepening to solve 8 puzzle problems using python. Make sure that the problem instance is solvable.

```
def get_user_input():
    print("Enter the initial state of the puzzle (3x3 grid):")
    initial_state = []
    for _ in range(3):
        row = list(map(int, input().split()))
        if len(row) != 3:
            raise ValueError(
                "Invalid input. Please enter 3 integers separated by spaces for each row."
            )
        initial_state.append(row)
    return initial_state



class State:
    def __init__(self, state):
        self.state = state

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

```
def __hash__(self):
    return hash(tuple(map(tuple, self.state)))

def find_empty_tile(self):
    for row in range(len(self.state)):
        for col in range(len(self.state[0])):
            if self.state[row][col] == 0:
                return row, col

def generate_successor_states(self):
    successor_states = []
    empty_row, empty_col = self.find_empty_tile()

    def move_tile(row, col):
        new_state = [list(row) for row in self.state]
        new_state[empty_row][empty_col] = new_state[row][col]
        new_state[row][col] = 0
        return new_state

    if empty_row > 0:
        successor_states.append(State(move_tile(empty_row - 1, empty_col)))
    if empty_row < 2:
        successor_states.append(State(move_tile(empty_row + 1, empty_col)))
    if empty_col > 0:
        successor_states.append(State(move_tile(empty_row, empty_col - 1)))
    if empty_col < 2:
        successor_states.append(State(move_tile(empty_row, empty_col + 1)))
    return successor_states
```

```
class Puzzle:
    def __init__(self, initial_state, goal_state, max_depth):
        self.initial_state = State(initial_state)
        self.goal_state = State(goal_state)
        self.max_depth = max_depth
        self.visited_states = set()

    def depth_limited_search(self, state, depth_limit, moves):
        if state == self.goal_state:
            return True
        if depth_limit == 0:
            return False
        if depth_limit > self.max_depth:
            return False
        successor_states = state.generate_successor_states()

        for successor_state in successor_states:
            if successor_state not in self.visited_states:
                self.visited_states.add(successor_state)
                moves.append(successor_state.state)

                if self.depth_limited_search(successor_state, depth_limit - 1, moves):
                    return True
                moves.pop()
                self.visited_states.remove(successor_state)
        return False
```

```
def solve(self):
        depth_limit = 0
        moves = [self.initial_state.state]

        while True:
            if self.depth_limited_search(self.initial_state, depth_limit, moves):
                print("Solution found within the depth limit.")
                print("\nSolution Path:")
                for move in moves:
                    print_puzzle(move)
                    print()
                print("Total number of moves:", len(moves) - 1)
                break
            else:
                self.visited_states.clear()
                depth_limit += 1
            if depth_limit > self.max_depth:
                print("Search terminated. Depth limit exceeds maximum depth.")
                break



def print_puzzle(state):
    for row in state:
        for tile in row:
            print(tile, end=" ")
        print()
```

```
def main():
    # Define the initial state of the puzzle and the goal state.
    # initial_state = [[1, 5, 3], [2, 7, 4], [6, 0, 8]]
    initial_state = get_user_input()
    print()
    goal_state = [[1, 2, 3], [4, 5, 6], [7, 8, 0]]
    max_depth = 100

    puzzle = Puzzle(initial_state, goal_state, max_depth)

    print("Initial State:")
    print_puzzle(puzzle.initial_state.state)
    print()
    print("Goal State:")
    print_puzzle(puzzle.goal_state.state)
    print("_____________________________________________________________")

    print("\nSolving with Depth-Limited Search...")
    puzzle.solve()


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