Program title: Solve 8 puzzle problems, Implement Iterative deepening search algorithm.
Code:

```
from collections import deque

# Define the goal state
GOAL_STATE = [[1, 2, 3], [4, 5, 6], [7, 8, 0]]

# Utility functions
def find_blank_tile(state):
    """Find the position of the blank tile (0) in the puzzle."""
    for i in range(len(state)):
        for j in range(len(state[i])):
            if state[i][j] == 0:
                return i, j

def is_goal(state):
    """Check if the current state is the goal state."""
    return state == GOAL_STATE

def move_tile(state, x1, y1, x2, y2):
    """Move the tile and return the new state after the move."""
    new_state = [row[:] for row in state]
    new_state[x1][y1], new_state[x2][y2] = new_state[x2][y2], new_state[x1][y1]
    return new_state

def get_neighbors(state):
    """Get all possible moves (neighbors) of the current state."""
    neighbors = []
    x, y = find_blank_tile(state)

    moves = [
        (x-1, y),  # Move Up
        (x+1, y),  # Move Down
        (x, y-1),  # Move Left
        (x, y+1)   # Move Right
    ]

    for nx, ny in moves:
        if 0 <= nx < 3 and 0 <= ny < 3:
            neighbors.append(move_tile(state, x, y, nx, ny))

    return neighbors
```

```
# Iterative Deepening Search (IDS)
class IterativeDeepeningSearch:
    def __init__(self, start_state):
        self.start_state = start_state

    def depth_limited_search(self, state, depth, visited):
        """Perform a depth-limited search."""
        if is_goal(state):
            return state
        if depth == 0:
            return None

        visited.append(state)
        for neighbor in get_neighbors(state):
            if neighbor not in visited:
                result = self.depth_limited_search(neighbor, depth - 1, visited)
                if result:
                    return result
        return None

    def iterative_deepening_search(self):
        """Perform an iterative deepening search."""
        depth = 0
        while True:
            visited = []
            result = self.depth_limited_search(self.start_state, depth, visited)
            if result:
                return result
            depth += 1

# Initial state of the puzzle (0 represents the blank space)
initial_state = [[1, 2, 3], [4, 0, 6], [7, 5, 8]]

# Run the Iterative Deepening Search
ids_solver = IterativeDeepeningSearch(initial_state)
solution = ids_solver.iterative_deepening_search()

if solution:
    print("Solution found:")
    for row in solution:
        print(row)
else:
    print("No solution found.")
```

8/10/24
Tuesday

Lab 2

Solve 8 puzzle problems, Implement Iterative deepening Search algorithm.

1. Initialization:
   - Represent the puzzle as a 3x3 grid with 0 as the empty space.
   - Define the goal state where tiles are ordered from 1 to 8 with 0 in the bottom-right corner.
   - Define valid moves for the empty space: up, down, left, right.
2. Check Goal:
   - Compare the current puzzle state with the goal state
3. Get Empty Position:
   - Locate the position of the empty space (0)
4. Move the Empty Space:
   - Try moving the empty space in all directions and generate new valid state
5. Depth-Limited Search (DLS):
   - Explore states up to a given depth limit
   - If the goal isn't reach with the limit backtrace and try new paths.
6. Iterative Deepening Search (IDS)
   - Start with depth 0 and incrementally increase the depth limit.
   - perform DLS for each limit until the goal state is found.
7. Print Solution:
   - Once the goal is found, print the sequence of steps reaching from

Date: Page:

the initial state to the goal state.

output:

Solution found in 14 steps.

Steps to reach the goal:

Step 0:

1	2	3
4	0	5
6	7	8

Step 2 :

1	2	3
4	5	0
6	7	8

Step 3 :

1	2	3
4	5	8
6	0	7

Step 4 :

1	2	3
4	5	8
0	6	7

Step 5 :

1	2	3
0	5	8
4	6	7

Step 6 :

1	2	3
5	0	8
4	6	7

step 6 :

1	2	3
5	0	8
4	6	7

Step 7 :

1	2	3
5	6	8
4	6	7

step 8 :

1	2	3
5	6	8
4	7	0

Step 9 :

1	2	3
5	6	0
4	7	8

Step 10 :

1	2	3
5	0	6
4	7	8

Step 11 :

1	2	3
0	5	6
4	7	8

Step 12 :

1	2	3
4	5	6
0	7	8

Step 13 :

1	2	3
4	5	6
7	0	8

step 14 :

1	2	3
4	5	6
7	8	0