WEEK-04

Implement **Iterative Deepening Search Algorithm** using **8 puzzle problem**

import copy

class Node:

    def \_\_init\_\_(self, state, parent=None, action=None, depth=0):

        self.state = state

        self.parent = parent

        self.action = action

        self.depth = depth

    def \_\_lt\_\_(self, other):

        return self.depth < other.depth

    def expand(self):

        children = []

        row, col = self.find\_blank()

        possible\_actions = []

        if row > 0:  # Can move the blank tile up

            possible\_actions.append('Up')

        if row < 2:  # Can move the blank tile down

            possible\_actions.append('Down')

        if col > 0:  # Can move the blank tile left

            possible\_actions.append('Left')

        if col < 2:  # Can move the blank tile right

            possible\_actions.append('Right')

        for action in possible\_actions:

            new\_state = copy.deepcopy(self.state)

            if action == 'Up':

                new\_state[row][col], new\_state[row - 1][col] = new\_state[row - 1][col], new\_state[row][col]

            elif action == 'Down':

                new\_state[row][col], new\_state[row + 1][col] = new\_state[row + 1][col], new\_state[row][col]

            elif action == 'Left':

                new\_state[row][col], new\_state[row][col - 1] = new\_state[row][col - 1], new\_state[row][col]

            elif action == 'Right':

                new\_state[row][col], new\_state[row][col + 1] = new\_state[row][col + 1], new\_state[row][col]

            children.append(Node(new\_state, self, action, self.depth + 1))

        return children

    def find\_blank(self):

        for row in range(3):

            for col in range(3):

                if self.state[row][col] == 0:

                    return row, col

        raise ValueError("No blank tile found")

def depth\_limited\_search(node, goal\_state, limit):

    if node.state == goal\_state:

        return node

    if node.depth >= limit:

        return None

    for child in node.expand():

        result = depth\_limited\_search(child, goal\_state, limit)

        if result is not None:

            return result

    return None

def iterative\_deepening\_search(initial\_state, goal\_state, max\_depth):

    for depth in range(max\_depth):

        result = depth\_limited\_search(Node(initial\_state), goal\_state, depth)

        if result is not None:

            return result

    return None

def print\_solution(node):

    path = []

    while node is not None:

        path.append((node.action, node.state))

        node = node.parent

    path.reverse()

    for action, state in path:

        if action:

            print(f"Action: {action}")

        for row in state:

            print(row)

        print()

initial\_state = [[1, 2, 3], [0, 4, 6], [7, 5, 8]]

goal\_state = [[1, 2, 3], [4, 5, 6], [7, 8, 0]]

max\_depth = 5

solution = iterative\_deepening\_search(initial\_state, goal\_state, max\_depth)

if solution:

    print("Solution found:")

    print\_solution(solution)

else:

    print("Solution not found.")

**OUTPUT:**



