**DEPARTMENTOFINFORMATIONTECHNOLOGY**

**COURSECODE:DJS22ITL501 DATE:16-8-24**

**COURSENAME:Artificial Intelligence Laboratory CLASS:TY-IT**

**Name : Anish Sharma Sap ID : 60003220045**

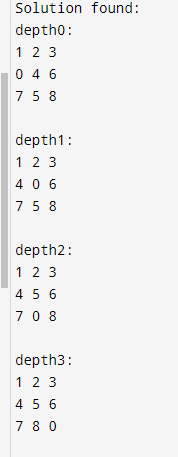
**EXPERIMENTNO.03**

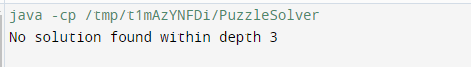
**CO/LO:** Apply various AI approaches to knowledge intensive problem solving, reasoning, planning and uncertainty.

**AIM/OBJECTIVE:** Implement DFID search algorithms to reach goal state

**Code:**

import java.util.ArrayList;  
import java.util.Arrays;  
import java.util.List;  
  
public class PuzzleSolver {  
    // The goal state of the 8-puzzle  
    static final int[][] goalState = {  
        {1, 2, 3},  
        {4, 5, 6},  
        {7, 8, 0}  
    };  
  
    // Directions for moving the blank tile (0): up, down, left, right  
    static final int[][] directions = {  
        {-1, 0}, {1, 0}, {0, -1}, {0, 1}  
    };  
  
    public static void main(String[] args) {  
        // Initial state of the puzzle  
        int[][] initialState = {  
            {1, 2, 3},  
            {0, 4, 6},  
            {7, 5, 8}  
        };  
  
        int maxDepth = 3;  
        List<int[][]> solutionPath = iterativeDeepeningSearch(initialState, maxDepth);  
  
        if (solutionPath != null) {  
            System.out.println("Solution found:");  
            for (int step = 0; step < solutionPath.size(); step++) {  
                System.out.println("depth" + step + ":");  
                printState(solutionPath.get(step));  
                System.out.println();  
            }  
        } else {  
            System.out.println("No solution found within depth " + maxDepth);  
        }  
    }  
  
    static List<int[][]> iterativeDeepeningSearch(int[][] startState, int maxDepth) {  
        for (int depth = 0; depth <= maxDepth; depth++) {  
            List<int[][]> path = new ArrayList<>();  
            path.add(startState);  
            List<int[][]> result = dfs(startState, depth, path);  
            if (result != null) {  
                return result;  
            }  
        }  
        return null;  
    }  
  
    static List<int[][]> dfs(int[][] state, int depth, List<int[][]> path) {  
        if (depth > 0 && isGoal(state)) {  
            return new ArrayList<>(path);  
        }  
  
        if (depth == 0) {  
            return null;  
        }  
  
        for (int[][] neighbor : getNeighbors(state)) {  
            if (!isInPath(neighbor, path)) { // Avoid revisiting the same state  
                path.add(neighbor);  
                List<int[][]> result = dfs(neighbor, depth - 1, path);  
                if (result != null) {  
                    return result;  
                }  
                path.remove(path.size() - 1); // Backtrack  
            }  
        }  
        return null;  
    }  
  
    static boolean isGoal(int[][] state) {  
        return Arrays.deepEquals(state, goalState);  
    }  
  
    static int[] getBlankPosition(int[][] state) {  
        for (int i = 0; i < 3; i++) {  
            for (int j = 0; j < 3; j++) {  
                if (state[i][j] == 0) {  
                    return new int[]{i, j};  
                }  
            }  
        }  
        return null;  
    }  
  
    static int[][] swap(int[][] state, int[] pos1, int[] pos2) {  
        int[][] newState = new int[3][3];  
        for (int i = 0; i < 3; i++) {  
            newState[i] = state[i].clone();  
        }  
        int temp = newState[pos1[0]][pos1[1]];  
        newState[pos1[0]][pos1[1]] = newState[pos2[0]][pos2[1]];  
        newState[pos2[0]][pos2[1]] = temp;  
        return newState;  
    }  
  
    static List<int[][]> getNeighbors(int[][] state) {  
        List<int[][]> neighbors = new ArrayList<>();  
        int[] blankPos = getBlankPosition(state);  
  
        for (int[] direction : directions) {  
            int newBlankRow = blankPos[0] + direction[0];  
            int newBlankCol = blankPos[1] + direction[1];  
            if (newBlankRow >= 0 && newBlankRow < 3 && newBlankCol >= 0 && newBlankCol < 3) {  
                int[][] newState = swap(state, blankPos, new int[]{newBlankRow, newBlankCol});  
                neighbors.add(newState);  
            }  
        }  
        return neighbors;  
    }  
  
    static boolean isInPath(int[][] state, List<int[][]> path) {  
        for (int[][] pastState : path) {  
            if (Arrays.deepEquals(state, pastState)) {  
                return true;  
            }  
        }  
        return false;  
    }  
  
    static void printState(int[][] state) {  
        for (int[] row : state) {  
            for (int num : row) {  
                System.out.print(num + " ");  
            }  
            System.out.println();  
        }  
    }  
}

**output:**



**Conclusion**: In this experiment we learnt to implement DFID to reach the goal state.