Batch: B2	**Roll No.: 16010122221**
Experiment / assignment / tutorial No. 4	
Grade: AA / AB / BB / BC / CC / CD /DD	
Signature of the Staff In-charge with date	

Title: Implementation of uninformed search algorithms – BFS,DFS, DLS for the given problem

Objective: Examine the efficiency and performance of uninformed search algorithms in solving various problems

Expected Outcome of Experiment:

Course Outcome	After successful completion of the course students should be able to
CO2	Analyse and solve problems for goal based agent architecture (searching and planning algorithms)

Books/ Journals/ Websites referred:

1. **"Artificial Intelligence: a Modern Approach" by Russell and Norving, Pearson education Publications**
2. **"Artificial Intelligence" By Rich and knight, Tata Mcgraw Hill Publications**
3. **http://people.cs.pitt.edu/~milos/courses/cs2710/lectures/Class4.pdf**
4. **http://cs.williams.edu/~andrea/cs108/Lectures/InfSearch/infSearch.html**
5. **http://www.cs.mcgill.ca/~dprecup/courses/AI/Lectures/ai-lecture02.pdf http://homepage.cs.uiowa.edu/~hzhang/c145/notes/04a-search.pdf**
6. **http://wiki.answers.com/Q/Informed_search_techniques_and_uninformed_search_techniques**

Pre Lab/ Prior Concepts:

Problem solving, state-space trees, problem formulation, goal based agent architecture

Historical Profile:

Problem-Solving Agent

A problem-solving agent is designed to find solutions to well-defined problems. This agent typically follows these steps:

1. **Formulate the Problem**: Define the initial state, goal state, and possible actions.
2. **Search for a Solution**: Use an appropriate search strategy to explore the problem space.
3. **Execute the Solution**: Apply the sequence of actions derived from the search.

Uninformed Search Algorithms

Uninformed search algorithms, also known as blind search algorithms, are basic search strategies that explore the search space without any additional information about the goal's location beyond what is provided in the problem definition.

New Concepts to be learned:

Uninformed (blind) search.

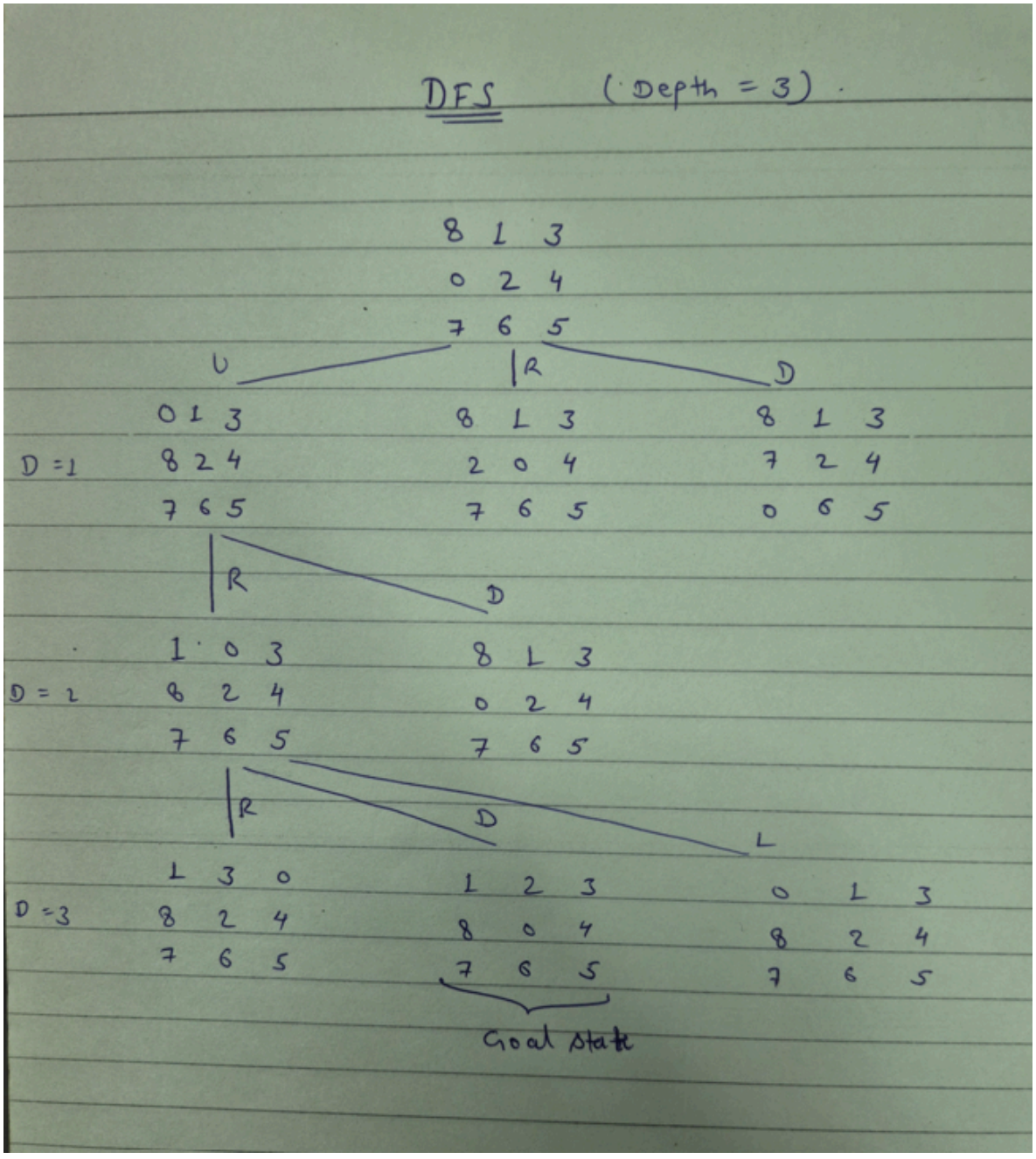
Uninformed Searching Technique

- B**readth-First Search (BFS)**:
  - Explores all nodes at the present depth level before moving on to nodes at the next depth level.
  - Complete and optimal if the cost is uniform.
- **Depth-First Search (DFS)**:
  - Explores as far as possible along each branch before backtracking.
  - Not complete or optimal, but requires less memory.
- **Depth-Limited Search (DLS)**:
  - Depth-first search with a predetermined depth limit.
  - Can overcome infinite path problems.

Problem Statement

The 8-Puzzle problem consists of a 3x3 grid with 8 numbered tiles and one blank tile. The objective is to rearrange the tiles from an initial configuration to a goal configuration by using a sequence of legal moves. Depth-First Search (DFS) is used as the search strategy to explore possible configurations.

- States: All possible configurations of the 3x3 grid, including the position of the blank tile and the numbered tiles.
- Initial State: A solvable configuration of the puzzle (e.g., a scrambled arrangement of the tiles).
- Transition Model: Legal moves involve swapping the blank tile with one of its adjacent tiles (horizontal or vertical).
- Actions: The set of possible moves, including moving the blank tile up, down, left, or right, based on the current state.
- Goal Test: Check if the current configuration matches the predefined goal configuration (e.g., tiles arranged in ascending order with the blank tile at the bottom-right corner).
- Search Strategy:
    - DFS explores the puzzle by diving deep into one possible sequence of moves before backtracking if a solution is not found.
    - DFS is implemented using a stack (or recursion) to manage the sequence of states.
- Path Cost: Each move has a uniform cost of 1, but DFS does not inherently guarantee the shortest path.
- Key Considerations:
    - Cyclic paths must be avoided by maintaining a record of visited states.
    - A depth limit can be introduced to mitigate infinite exploration in larger or unsolvable configurations.


DFS (Depth = 3)
8 1 3
0 2 4
7 6 5
U
R
D
0 1 3
8 2 4
7 6 5
8 1 3
2 0 4
7 6 5
8 1 3
7 2 4
0 6 5
D = 1
R
D
1 0 3
8 2 4
7 6 5
8 1 3
0 2 4
7 6 5
D = 2
R
D
L
1 3 0
8 2 4
7 6 5
1 2 3
8 0 4
7 6 5
0 1 3
8 2 4
7 6 5
D = 3
Goal state

Source code for the assigned problem with assigned algorithm:

```
from collections import deque


def print_board(board):
    for row in board:
        print(' '.join(map(str, row)))
    print()


def get_blank_position(board):
    for i in range(3):
        for j in range(3):
            if board[i][j] == 0:
                return (i, j)


def is_valid_move(i, j):
    return 0 <= i < 3 and 0 <= j < 3


def swap_tiles(board, blank_pos, new_pos):
    i1, j1 = blank_pos
    i2, j2 = new_pos
    board[i1][j1], board[i2][j2] = board[i2][j2], board[i1][j1]
```

```
def is_goal_state(board, goal_state):

     return all(board[i][j] == goal_state[i][j] for i in range(3) for
j in range(3))



def get_neighbors(board):

    blank_pos = get_blank_position(board)

    neighbors = []

    moves = [(0, 1), (1, 0), (0, -1), (-1, 0)]



    for move in moves:

        new_pos = (blank_pos[0] + move[0], blank_pos[1] + move[1])

        if is_valid_move(*new_pos):

            new_board = [row.copy() for row in board]

            swap_tiles(new_board, blank_pos, new_pos)

            neighbors.append(new_board)



    return neighbors



def dfs_8_puzzle(initial_state, goal_state):

    visited = set()

    stack = [(initial_state, [])]



    while stack:

        current_state, path = stack.pop()
```

```
        if is_goal_state(current_state, goal_state):
            return path + [current_state]

        state_tuple = tuple(tuple(row) for row in current_state)
        if state_tuple not in visited:
            visited.add(state_tuple)
            neighbors = get_neighbors(current_state)

            for neighbor in neighbors:
                stack.append((neighbor, path + [current_state]))

    return None

initial_state = [
    [8, 1, 3],
    [0, 2, 4],
    [7, 6, 5]
]

goal_state = [
    [1, 2, 3],
    [8, 0, 4],
```

```
    [7, 6, 5]
]


solution = dfs_8_puzzle(initial_state, goal_state)


if solution:
    print("Solution found in {} steps:".format(len(solution)-1))
    for i, step in enumerate(solution):
        if i == 0:
            print("Initial state:")
        else:
            print("Step {}:".format(i))
        print_board(step)
else:
    print("No solution found.")
```

Output screenshots:

```
(base) PS C:\Users\aksha\Desktop\Sem-6\AI\AI_codes> python -u "c:\Users\ak
Solution found in 3 steps:
Initial state:
8 1 3
0 2 4
7 6 5

Step 1:
0 1 3
8 2 4
7 6 5

Step 2:
1 0 3
8 2 4
7 6 5

Step 3:
1 2 3
8 0 4
7 6 5
```

Comparison of performance of uninformed Algorithm:

Given the characteristics of the 8 Puzzle problem, which has a relatively small state space, BFS is generally the preferred choice because:

• The state space of the 8 Puzzle problem is not excessively large, so the memory-intensive nature of BFS is less of a concern.

• BFS guarantees the optimal solution, which is desirable for solving puzzles.

• BFS will explore the shallowest nodes first, which can be advantageous in situations where the solution is closer to the initial state.

Therefore, in this specific case, BFS is considered the best algorithm. However, if memory constraints were a significant concern or if the state space were much larger, DFS or DLS might be more practical despite their lack of optimality.

Conclusion:

The 8-Puzzle problem demonstrates the application of Depth-First Search (DFS) as a systematic approach to explore potential configurations in achieving the goal state. DFS effectively traverses deep paths within the search space, providing insights into problem-solving strategies where exhaustive exploration is crucial. However, it lacks guarantees for optimality, often resulting in suboptimal solutions if shorter paths exist. By incorporating mechanisms such as depth limits and state revisitation checks, DFS can address challenges like infinite loops and redundant explorations. This problem highlights the importance of selecting appropriate search strategies based on the problem's requirements, constraints, and desired outcomes.

Post Lab Objective Questions

1. Which is not a Goal-based agent?

a. Inference
b. Search
c. Planning
d. Conclusion
e. Dynamic search.

Answer:

2. Which were built in such a way that humans had to supply the inputs and interpret the outputs?

a. Agents
b. Sensor
c. AI System

d. Actuators

Answer:

Post Lab Objective questions

1. **Which search algorithm imposes a fixed depth limit on nodes?**
   a. Depth-limited search
   b. Depth-first search
   c. Iterative Deepening search
   d. Only (a) and (b)
   e. Only (a), (b) and (c).

Answer:

2. **Optimality of BFS is**
   a. When all step costs are equal
   b. When all step costs are unequal
   c. When there is less number of nodes
   d. Both a & c

Answer:

3. What is a common application of Depth-First Search?
   a. Finding the shortest path
   b. Solving puzzles
   c. Web crawling
   d. Scheduling processes

Answer:

Post Lab Subjective Questions:

1. Provide a real-world example where BFS would be preferable over DFS, and another where DFS would be a better choice. Explain your reasoning.

- **BFS Example**: Finding the shortest path in an unweighted graph, such as navigating through a city grid to reach the nearest gas station.
  - **Reasoning**: BFS ensures the shortest path is found because it explores all nodes at the current depth before proceeding deeper.
- **DFS Example**: Solving a puzzle like Sudoku or finding a Hamiltonian path in a graph.

- **Reasoning**: DFS explores one path fully before backtracking, making it suitable for problems requiring a complete exploration of one branch at a time.

2. If your search algorithm did not find a solution, what could be the possible reasons? How would adding a depth limit to DFS affect the outcome?

- **Possible Reasons for Search Failure**:
  - The solution does not exist in the graph or search space.
  - The graph is infinite or too large to explore completely.
  - Cycles in the graph cause infinite loops.
  - The algorithm lacks sufficient memory or computational power.
- **Effect of Adding a Depth Limit to DFS**:
  - Depth-limited DFS prevents exploring paths beyond a certain depth, reducing memory usage and mitigating the risk of infinite loops in cyclic graphs. However, this can lead to missed solutions if they lie beyond the depth limit.

3. If you modified BFS to use a priority queue instead of a regular queue, how would this change the search behavior? What kind of search algorithm would it resemble?

- **Change in Behavior**:
  - Nodes would be explored based on their priority (e.g., path cost or heuristic value) rather than their depth in the graph.
  - The algorithm would no longer guarantee level-order exploration.
- **Resemblance**:
  - If the priority is based on path cost, the algorithm becomes **Uniform-Cost Search**.

- If the priority combines path cost with a heuristic, it becomes *A Search**.