Name - Mandira Nirankari

Class Roll No. – 46

University Roll No. – 202401100400119

Branch – CSE (AIML)

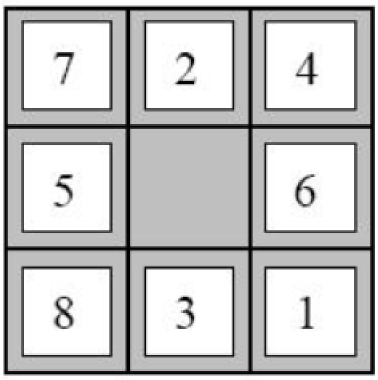
Section – ‘B’

Problem statement - 8-Puzzle Solver

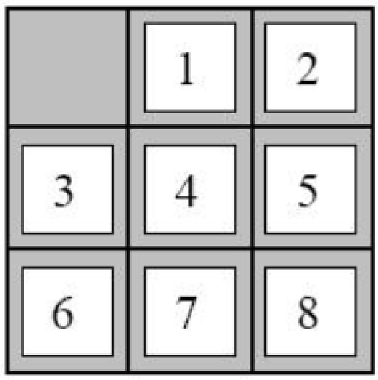
Introduction

In this problem, we'll explore the 8 Puzzle Problem, how it's structured, the search algorithms used to solve it, and the role of heuristics in finding optimal solutions.

The 8 Puzzlesolver is a sliding puzzle that consists of eight numbered tiles (1-8) placed randomly on a 3x3 grid along with one empty slot. The player can move adjacent tiles into the blank space, and the objective is to arrange the tiles in a specific goal state by sliding them one at a time. It is a classic problem in artificial intelligence (AI) and is often used to teach problem-solving techniques. It consists of a 3x3 grid with 8 numbered tiles and one empty space.



Start State



Goal State

Methodology

1. **State Representation**: The 8-Puzzle is represented as a list of 9 integers, where 0 represents the blank tile, and the other numbers represent the numbered tiles.
2. **A* Search Algorithm**: The program employs A* search, an informed search algorithm that combines the actual cost to reach a state (depth) with a heuristic (Manhattan distance) to estimate the cost to reach the goal. This helps prioritize the most promising states during the search.
3. **Heuristic (Manhattan Distance)**: The heuristic used here is **Manhattan distance**, which calculates the total distance each tile is from its goal position (in terms of grid steps).
4. **Priority Queue**: A **min-heap** (priority queue) is used to store states, prioritizing states with the lowest cost (depth + heuristic). This ensures that the most promising states are explored first.
5. **State Expansion**: Starting from the initial state, the algorithm explores all valid moves for the blank tile (up, down, left, right), generating new states. States are added to the queue if they haven't been visited already.
6. **Termination**: The algorithm continues until the goal state is reached or no solution exists. If a solution is found, the program traces back the path of moves to reconstruct and display the solution sequence.

Code

```
import heapq
from termcolor import colored


# Class to represent the state of the 8-puzzle
class PuzzleState:
    def __init__(self, board, parent, move, depth, cost):
        self.board = board  # The puzzle board configuration
        self.parent = parent  # Parent state
        self.move = move  # Move to reach this state
        self.depth = depth  # Depth in the search tree
        self.cost = cost  # Cost (depth + heuristic)


    def __lt__(self, other):
        return self.cost < other.cost


# Function to display the board in a visually appealing format
def print_board(board):
    print("+---+---+---+")
    for row in range(0, 9, 3):
        row_visual = "|"
        for tile in board[row:row + 3]:
```

```
            if tile == 0:  # Blank tile
                row_visual += f" {colored(' ', 'cyan')} |"
            else:
                row_visual += f" {colored(str(tile), 'yellow')} |"
        print(row_visual)
        print("+---+---+---+")

# Goal state for the puzzle
goal_state = [1, 2, 3, 4, 5, 6, 7, 8, 0]

# Possible moves for the blank tile (up, down, left, right)
moves = {
    'U': -3,  # Move up
    'D': 3,   # Move down
    'L': -1,  # Move left
    'R': 1    # Move right
}

# Function to calculate the heuristic (Manhattan distance)
def heuristic(board):
    distance = 0
    for i in range(9):
        if board[i] != 0:
```

```
        x1, y1 = divmod(i, 3)
        x2, y2 = divmod(board[i] - 1, 3)
        distance += abs(x1 - x2) + abs(y1 - y2)
    return distance

# Function to get the new state after a move
def move_tile(board, move, blank_pos):
    new_board = board[:]
    new_blank_pos = blank_pos + moves[move]
    new_board[blank_pos], new_board[new_blank_pos] = 
new_board[new_blank_pos], new_board[blank_pos]
    return new_board

# A* search algorithm
def a_star(start_state):
    open_list = []
    closed_list = set()
    heapq.heappush(open_list, PuzzleState(start_state, None, None, 0, 
heuristic(start_state)))

    while open_list:
        current_state = heapq.heappop(open_list)
```

```
        if current_state.board == goal_state:
            return current_state

        closed_list.add(tuple(current_state.board))

        blank_pos = current_state.board.index(0)

        for move in moves:
            if move == 'U' and blank_pos < 3:  # Invalid move up
                continue
            if move == 'D' and blank_pos > 5:  # Invalid move down
                continue
            if move == 'L' and blank_pos % 3 == 0:  # Invalid move left
                continue
            if move == 'R' and blank_pos % 3 == 2:  # Invalid move right
                continue

            new_board = move_tile(current_state.board, move,
blank_pos)

            if tuple(new_board) in closed_list:
                continue
```

```
            new_state = PuzzleState(new_board, current_state, move,
current_state.depth + 1, current_state.depth + 1 +
heuristic(new_board))
            heapq.heappush(open_list, new_state)


    return None


# Function to print the solution path
def print_solution(solution):
    path = []
    current = solution
    while current:
        path.append(current)
        current = current.parent
    path.reverse()


    for step in path:
        print(f"Move: {step.move}")
        print_board(step.board)


# Initial state of the puzzle
initial_state = [1, 2, 3, 4, 0, 5, 6, 7, 8]
```

```
# Solve the puzzle using A* algorithm
solution = a_star(initial_state)


# Print the solution
if solution:
    print(colored("Solution found:", "green"))
    print_solution(solution)
else:
    print(colored("No solution exists.", "red"))
```

Output

```
Solution found:
Move: None
+---+---+---+
| 1 | 2 | 3 |
+---+---+---+
| 4 |   | 5 |
+---+---+---+
| 6 | 7 | 8 |
+---+---+---+
Move: R
+---+---+---+
| 1 | 2 | 3 |
+---+---+---+
| 4 | 5 |   |
+---+---+---+
| 6 | 7 | 8 |
+---+---+---+
Move: D
+---+---+---+
| 1 | 2 | 3 |
+---+---+---+
| 4 | 5 | 8 |
+---+---+---+
| 6 | 7 |   |
+---+---+---+
Move: L
+---+---+---+
| 1 | 2 | 3 |
+---+---+---+
| 4 | 5 | 8 |
+---+---+---+
| 6 |   | 7 |
+---+---+---+
Move: L
+---+---+---+
| 1 | 2 | 3 |
+---+---+---+
| 4 | 5 | 8 |
+---+---+---+
|   | 6 | 7 |
+---+---+---+
Move: U
+---+---+---+
| 1 | 2 | 3 |
+---+---+---+
|   | 5 | 8 |
+---+---+---+
| 4 | 6 | 7 |
+---+---+---+
```

```
Move: R
+---+---+---+
| 1 | 2 | 3 |
+---+---+---+
| 5 |   | 8 |
+---+---+---+
| 4 | 6 | 7 |
+---+---+---+
Move: D
+---+---+---+
| 1 | 2 | 3 |
+---+---+---+
| 5 | 6 | 8 |
+---+---+---+
| 4 |   | 7 |
+---+---+---+
Move: R
+---+---+---+
| 1 | 2 | 3 |
+---+---+---+
| 5 | 6 | 8 |
+---+---+---+
| 4 | 7 |   |
+---+---+---+
Move: U
+---+---+---+
| 1 | 2 | 3 |
+---+---+---+
| 5 | 6 |   |
+---+---+---+
| 4 | 7 | 8 |
+---+---+---+
Move: L
+---+---+---+
| 1 | 2 | 3 |
+---+---+---+
| 5 |   | 6 |
+---+---+---+
| 4 | 7 | 8 |
+---+---+---+
Move: L
+---+---+---+
| 1 | 2 | 3 |
+---+---+---+
|   | 5 | 6 |
+---+---+---+
| 4 | 7 | 8 |
+---+---+---+
```

```
Move: D
+---+---+---+
| 1 | 2 | 3 |
+---+---+---+
| 4 | 5 | 6 |
+---+---+---+
|   | 7 | 8 |
+---+---+---+
Move: R
+---+---+---+
| 1 | 2 | 3 |
+---+---+---+
| 4 | 5 | 6 |
+---+---+---+
| 7 |   | 8 |
+---+---+---+
Move: R
+---+---+---+
| 1 | 2 | 3 |
+---+---+---+
| 4 | 5 | 6 |
+---+---+---+
| 7 | 8 |   |
+---+---+---+
```

References

Problem overview - https://www.geeksforgeeks.org/8-puzzle-problem-in-ai/

Methodology - Chatgpt