TITTLE : 8-PUZZLE SOLVER

Project Title: 8-Puzzle Solver

Submitted by: [Himanshu Kumar]

Roll No. : [26]

Introduction

The 8-puzzle problem is a classic problem in artificial intelligence, where a 3×3 grid contains eight numbered tiles and a blank space. The goal is to arrange the tiles in numerical order by sliding them into the blank space using the fewest moves possible. This report discusses the implementation of an 8-puzzle solver using the A* search algorithm with the Manhattan distance heuristic.

Methodology

Approach Used:

1. **State Representation:** The puzzle is represented as a 3×3 NumPy array.
2. **Heuristic Function:** The Manhattan distance heuristic is used to estimate the cost to reach the goal state.
3. **Search Algorithm:** The A* algorithm is implemented using a priority queue (heap), expanding the lowest-cost node first.
4. **Successor Generation:** The possible moves (Up, Down, Left, Right) are checked, and valid moves are added to the search tree.
5. **Solution Retrieval:** The sequence of moves from the initial state to the goal state is traced back.

Code

```
import heapq

import numpy as np


# Define the goal state for the 8-puzzle

GOAL_STATE = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 0]])
```

```
class Puzzle:
    def __init__(self, state, parent=None, move=None, depth=0, cost=0):
        self.state = np.array(state)
        self.parent = parent
        self.move = move
        self.depth = depth
        self.cost = cost + self.manhattan_distance()

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

    def manhattan_distance(self):
        distance = 0
        for i in range(1, 9):
            x, y = np.where(self.state == i)
            goal_x, goal_y = np.where(GOAL_STATE == i)
            distance += abs(x[0] - goal_x[0]) + abs(y[0] - goal_y[0])
        return int(distance)

    def possible_moves(self):
        x, y = np.where(self.state == 0)
        moves = []
        if x > 0: moves.append(('Up', (x[0] - 1, y[0])))
        if x < 2: moves.append(('Down', (x[0] + 1, y[0])))
        if y > 0: moves.append(('Left', (x[0], y[0] - 1)))
        if y < 2: moves.append(('Right', (x[0], y[0] + 1)))
        return moves
```

```
    def generate_child(self, move, new_pos):
        new_state = self.state.copy()
        x, y = np.where(new_state == 0)
        new_x, new_y = new_pos
        new_state[x[0], y[0]], new_state[new_x, new_y] = new_state[new_x, new_y], new_state[x[0], y[0]]
        return Puzzle(new_state, parent=self, move=move, depth=self.depth + 1, cost=self.depth + 1)

    def is_goal(self):
        return np.array_equal(self.state, GOAL_STATE)

    def path(self):
        path = []
        current = self
        while current.parent:
            path.append(current.move)
            current = current.parent
        return path[::-1]

# A* Search Algorithm
def a_star(initial_state):
    initial_puzzle = Puzzle(initial_state)
    open_list = []
    closed_set = set()
    heapq.heappush(open_list, initial_puzzle)

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

```
        if current.is_goal():
            return current.path()
        closed_set.add(tuple(map(tuple, current.state)))

        for move, new_pos in current.possible_moves():
            child = current.generate_child(move, new_pos)
            if tuple(map(tuple, child.state)) not in closed_set:
                heapq.heappush(open_list, child)

    return None

# Input Handling
print("Enter the 8-puzzle numbers row-wise (use 0 for empty space, separated by spaces):")
user_input = list(map(int, input().strip().split()))
initial_state = np.array(user_input).reshape(3, 3)

# Solve the puzzle
solution = a_star(initial_state)
if solution:
    print("Solution found:", solution)
else:
    print("No solution exists")
```

Input & Output/Result

```
Enter the 8-puzzle numbers row-wise (use 0 for empty space, separated by spaces):
3 2 1 4 7 5 8 6 0
Solution found: ['Up', 'Up', 'Left', 'Left', 'Down', 'Right', 'Right', 'Up', 'Left', 'Down', 'Right', 'Down', 'Left', 'Left', 'Up', 'Up', 'Right', 'Down', 'Right', 'Down']
Final arranged puzzle:
[[1 2 3]
 [4 5 6]
 [7 8 0]]
```

```
Enter the 8-puzzle numbers row-wise (use 0 for empty space, separated by spaces):
6 5 4 7 8 2 3 1 0
Solution found: ['Up', 'Left', 'Down', 'Left', 'Up', 'Right', 'Right', 'Down', 'Left', 'Up', 'Up', 'Right', 'Down', 'Down', 'Left', 'Up', 'Up', 'Left', 'Down', 'Right', 'Up', 'R
Final arranged puzzle:
[[1 2 3]
 [4 5 6]
 [7 8 0]]
```

References/Credits

1. Artificial Intelligence: A Modern Approach by Stuart Russell and Peter Norvig.
2. Online resources and documentation on the A* algorithm.
3. NumPy official documentation for handling arrays efficiently.

Thankyou.