Program 4: 8-Puzzle Problem

The **8-Puzzle** is a sliding puzzle that consists of a 3×3 grid with eight numbered tiles (1 to 8) and one blank space. The goal is to move the tiles in such a way that they reach a predefined goal state by sliding tiles into the blank space.

A typical goal state looks like:

1	2	3
4	5	6
7	8	_

Where _ represents the blank space.

Rules:

1. Only one tile can be moved at a time by sliding it into the blank space.
2. The possible moves are **up, down, left,** and **right**.
3. The puzzle is solvable only if the initial state is reachable from the goal state by a sequence of these moves.

Example

Initial State:

1	2	3
4	_	6
7	5	8

Goal State:

1	2	3
4	5	6
7	8	_

Steps to solve:

1. Move 5 to the blank position.
2. Move 8 up to its correct position.
3. Move the blank space to its correct position.

Algorithm (A* Search Algorithm using Heuristic - Manhattan Distance) :

A* algorithm is widely used to solve the 8-Puzzle problem efficiently.

Steps:

1. Initialize the start state and goal state.
2. Compute the heuristic value `h(n)` for each state (Manhattan Distance).
3. Compute `f(n) = g(n) + h(n)`, where:
    - `g(n)`: Cost to reach the current state.
    - `h(n)`: Heuristic function (Manhattan Distance).
4. Expand the least cost node.
5. If the goal state is reached, stop.
6. Otherwise, push new states into the **priority queue** and continue.

Flowchart

A flowchart follows this logic:

1. Start
2. Initialize the puzzle state and goal state
3. Calculate heuristic (h) and cost (g)
4. Expand the least cost state
5. Check if the goal state is reached
    - Yes → Stop
    - No → Generate new states and continue
6. Repeat until the goal is found

Program: A* Algorithm for 8-Puzzle

```
from queue import PriorityQueue

def get_input():
    print("Enter the 8-puzzle state row by row (use '0' for blank space):")
    state = []
    for _ in range(3):
        row = input().split()
        state.append([int(x) for x in row])
    return state
```

```
def heuristic(state, goal):
    """Calculate Manhattan Distance heuristic"""
    distance = 0
    goal_positions = {goal[i][j]: (i, j) for i in range(3) for j in range(3)}
    for i in range(3):
        for j in range(3):
            if state[i][j] != 0:  # Ignore the blank tile (0)
                target_x, target_y = goal_positions[state[i][j]]
                distance += abs(target_x - i) + abs(target_y - j)
    return distance


def get_neighbors(state):
    """Generate possible moves"""
    moves = []
    x, y = next((i, j) for i in range(3) for j in range(3) if state[i][j] == 0)
    directions = [("Up", -1, 0), ("Down", 1, 0), ("Left", 0, -1), ("Right", 0, 1)]
    for move, dx, dy in directions:
        new_x, new_y = x + dx, y + dy
        if 0 <= new_x < 3 and 0 <= new_y < 3:
            new_state = [row[:] for row in state]
            new_state[x][y], new_state[new_x][new_y] = new_state[new_x][new_y],
new_state[x][y]
            moves.append((new_state, move))


    return moves
```

```
def solve_8_puzzle(start, goal):
    """A* Search Algorithm with Open List and Closed List"""
    open_list = PriorityQueue()
    closed_set = set()
    open_list.put((heuristic(start, goal), 0, start, []))
    while not open_list.empty():
        _, g, state, path = open_list.get()
        #print("Current State:")
        #for row in state:
            #print(row)
        #print("Path taken:", path)
        #print("---")

        if state == goal:
            return path

        closed_set.add(tuple(map(tuple, state)))

        for new_state, move in get_neighbors(state):
            if tuple(map(tuple, new_state)) in closed_set:
                continue
            open_list.put((g + 1 + heuristic(new_state, goal), g + 1, new_state, path +
[move]))

    return None  # No solution
```

```
# Get user inputs
start_state = get_input()
print("Enter the goal state row by row (use '0' for blank space):")
goal_state = get_input()

# Solve the puzzle
solution = solve_8_puzzle(start_state, goal_state)

# Print result
if solution:
    print("Solution Moves:", solution)
else:
    print("No solution found!")
```

Example Input & Output

Input:

```
Enter the 8-puzzle state row by row (use '0' for blank space):
1 2 3
4 5 6
7 0 8
Enter the goal state row by row (use '0' for blank space):
Enter the 8-puzzle state row by row (use '0' for blank space):
1 2 3
4 5 6
7 8 0
```

Output:

Solution Moves: ['Right']

Input:

Enter the 8-puzzle state row by row (use '0' for blank space):

1 2 0

3 4 5

6 7 8

Enter the goal state row by row (use '0' for blank space):

Enter the 8-puzzle state row by row (use '0' for blank space):

1 2 3

4 5 6

7 8 0

Output:

Solution Moves: ['Down', 'Down', 'Left', 'Up', 'Left', 'Down', 'Right', 'Right', 'Up', 'Left', 'Left', 'Up', 'Right', 'Right', 'Down', 'Left', 'Up', 'Left', 'Down', 'Down', 'Right', 'Right']

==========
======