PROGRAM TITLE-1

8-PUZZLE PROBLEM

AIM:

To write and execute the python program for solving 8 puzzle problem.

PROCEDURE:

1. **Define Puzzle State:** Create a class Puzzle State representing the puzzle configuration, including methods for moving the blank space and calculating the heuristic using Manhattan distance.
2. **A Algorithm: *** Implement the A* algorithm using a priority queue to explore puzzle states based on their total cost (path cost + heuristic).
3. **Heuristic Function:** Use the Manhattan distance as a heuristic to estimate the distance of each tile from its goal position.
4. **Solve Puzzle:** Start with an initial puzzle state, expand possible moves, and iteratively choose the most promising state until the goal state is reached.
5. **Output Solution:** Trace back the path from the goal state to the initial state to obtain the sequence of moves required to solve the 8-puzzle.

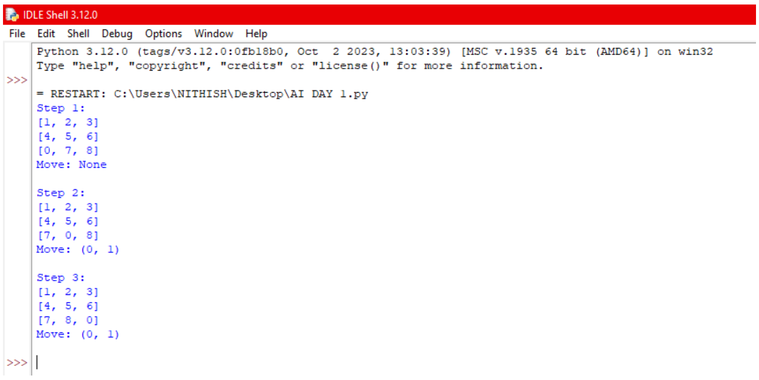
CODING:

```
import heapq


class PuzzleNode:
    def __init__(self, state, parent=None, move=None, cost=0):
        self.state = state
        self.parent = parent
        self.move = move
        self.cost = cost
        self.priority = self.cost + self.heuristic()

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

    def __eq__(self, other):
        return self.state == other.state


    def __hash__(self):
        return hash(str(self.state))


    def heuristic(self):
        # Manhattan distance heuristic
        goal_state = [[1, 2, 3], [4, 5, 6], [7, 8, 0]]
        h = 0
        for i in range(3):
            for j in range(3):
                if self.state[i][j] != 0:
                    row, col = divmod(self.state[i][j] - 1, 3)
                    h += abs(i - row) + abs(j - col)
        return h


    def get_successors(self):
        successors = []
        zero_row, zero_col = next((i, j) for i, row in enumerate(self.state) for j, val in enumerate(row) if
val == 0)
        moves = [(0, 1), (0, -1), (1, 0), (-1, 0)]


        for move in moves:
            new_row, new_col = zero_row + move[0], zero_col + move[1]


            if 0 <= new_row < 3 and 0 <= new_col < 3:
                new_state = [row.copy() for row in self.state]
                new_state[zero_row][zero_col], new_state[new_row][new_col] =
new_state[new_row][new_col], 0
                successors.append(PuzzleNode(new_state, self, move, self.cost + 1))


        return successors
```

```
def solve_8_puzzle(initial_state):
    initial_node = PuzzleNode(initial_state)
    frontier = [initial_node]
    explored = set()

    while frontier:
        current_node = heapq.heappop(frontier)

        if current_node.state == [[1, 2, 3], [4, 5, 6], [7, 8, 0]]:
            # Goal state reached, reconstruct the path
            path = []
            while current_node:
                path.append((current_node.state, current_node.move))
                current_node = current_node.parent
            path.reverse()
            return path

        explored.add(current_node)

        successors = current_node.get_successors()
        for successor in successors:
            if successor not in explored and successor not in frontier:
                heapq.heappush(frontier, successor)

    return None

if __name__ == "__main__":
    # Example usage:
    initial_state = [[1, 2, 3], [4, 5, 6], [0, 7, 8]]
    solution_path = solve_8_puzzle(initial_state)
```

```
if solution_path:
    for step, (state, move) in enumerate(solution_path):
        print(f"Step {step + 1}:")
        for row in state:
            print(row)
        print(f"Move: {move}\n")
else:
    print("No solution found.")
```

OUTPUT:



```
Python 3.12.0 (tags/v3.12.0:0fb18b0, Oct  2 2023, 13:03:39) [MSC v.1935 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> 
= RESTART: C:\Users\NITHISH\Desktop\AI DAY 1.py
Step 1:
[1, 2, 3]
[4, 5, 6]
[0, 7, 8]
Move: None

Step 2:
[1, 2, 3]
[4, 5, 6]
[7, 0, 8]
Move: (0, 1)

Step 3:
[1, 2, 3]
[4, 5, 6]
[7, 8, 0]
Move: (0, 1)

>>> 
```

RESULT:

Hence the program has been successfully executed and verified.