2.8 PUZZLE A* IMPLEMENTATION USING MANHATTAN DISTANCE

```
import heapq

class PuzzleState:
    def __init__(self, board, zero_pos, moves=0):
        self.board = board
        self.zero_pos = zero_pos
        self.moves = moves
        self.heuristic = self.calculate_manhattan_distance()

    def calculate_manhattan_distance(self):
        distance = 0
        goal_positions = {
            1: (0, 0), 2: (0, 1), 3: (0, 2),
            8: (1, 0), 0: (1, 1), 4: (1, 2),
            7: (2, 0), 6: (2, 1), 5: (2, 2)
        }

        for i in range(3):
            for j in range(3):
                tile = self.board[i][j]
                if tile != 0:
                    goal_x, goal_y = goal_positions[tile]
                    distance += abs(i - goal_x) + abs(j - goal_y)

        return distance

    def get_neighbors(self):
        neighbors = []
        x, y = self.zero_pos
        directions = [(-1, 0), (1, 0), (0, -1), (0, 1)]

        for dx, dy in directions:
            new_x, new_y = x + dx, y + dy
            if 0 <= new_x < 3 and 0 <= new_y < 3:
                new_board = [row[:] for row in self.board]
                new_board[x][y], new_board[new_x][new_y] =
new_board[new_x][new_y], new_board[x][y]
                neighbors.append((new_board, (new_x, new_y)))

        return neighbors

    def __lt__(self, other):
        return (self.moves + self.heuristic) < (other.moves +
other.heuristic)
```

```
    def __repr__(self):
        return f"{self.board} (Moves: {self.moves}, Heuristic:
{self.heuristic})"

def a_star(initial_board):
    initial_zero_pos = next((i, j) for i in range(3) for j in range(3) if
initial_board[i][j] == 0)
    initial_state = PuzzleState(initial_board, initial_zero_pos)

    priority_queue = []
    heapq.heappush(priority_queue, initial_state)
    visited = set()

    while priority_queue:
        current_state = heapq.heappop(priority_queue)
        print(f"Exploring State: {current_state}")
        if current_state.heuristic == 0:
            print(f"Reached goal in {current_state.moves} moves.")
            return

        visited.add(tuple(map(tuple, current_state.board)))

        for neighbor_board, neighbor_zero_pos in
current_state.get_neighbors():
            neighbor_state = PuzzleState(neighbor_board,
neighbor_zero_pos, current_state.moves + 1)

            if tuple(map(tuple, neighbor_board)) not in visited:
                heapq.heappush(priority_queue, neighbor_state)

    print("No solution found.")

if __name__ == "__main__":
    initial_board = [
        [2, 8, 3],
        [1, 6, 4],
        [0, 7, 5]
    ]
    a_star(initial_board)
```

```
Exploring State: [[2, 8, 3], [1, 6, 4], [0, 7, 5]] (Moves: 0, Heuristic: 6)
Exploring State: [[2, 8, 3], [1, 6, 4], [7, 0, 5]] (Moves: 1, Heuristic: 5)
Exploring State: [[2, 8, 3], [1, 0, 4], [7, 6, 5]] (Moves: 2, Heuristic: 4)
Exploring State: [[2, 0, 3], [1, 8, 4], [7, 6, 5]] (Moves: 3, Heuristic: 3)
Exploring State: [[0, 2, 3], [1, 8, 4], [7, 6, 5]] (Moves: 4, Heuristic: 2)
Exploring State: [[1, 2, 3], [0, 8, 4], [7, 6, 5]] (Moves: 5, Heuristic: 1)
Exploring State: [[1, 2, 3], [8, 0, 4], [7, 6, 5]] (Moves: 6, Heuristic: 0)
Reached goal in 6 moves.
```

Manhattan Distance:

Initial State:

2	8	3
1	6	4
	7	5

Final State

1	2	3
8		4
7	6	5

Algorithm:

```
CLASS PuzzleState:
FUNCTION __init__(board, zero_pos, move=0):
  SET self.board = board
SET self.zero_pos = zero_pos
  SET self.move = moves
SET self.heuristic = self.calculate_manhattan_
                     distance()
FUNCTION Calculate_manhattan_distance():
  SET distance = 0
  SET goal_positions =
1: (0,0), 2: (0,1), 3: (0,2),
8: (1,0), 0: (1,1), 4: (1,2),
7: (2,0), 6: (2,1), 5: (2,2).
}

FOR i IN range(3):
  FOR j IN range(3):
  SET tile = self.board[i][j]
  IF tile != 0:
  SET goal_x, goal_y = goal_Positions[tile]
  distance += abs(i - goal_x) + abs(j - goal_y)
```

```
FUNCTION get_neighbour():
    SET neighbors = []
    SET x, y = self.zero_pos
    SET directions = [(-1,0), (1,0), (0,-1), (0,1)]
    FOR (dx, dy) IN directions:
        SET new_x = x + dx
        SET new_y = y + dy
        IF (0 <= new_x < 3 AND 0 <= new_y < 3):
            SET new_board = COPY of self.board
            SWAP new_board[x][y] WITH new_board
                                    [new_x][new_y]
            APPEND (new_board, (new_x, new_y)) TO neighbors
    RETURN neighbors

FUNCTION __lt__(other):
    RETURN (self.moves + self.heuristic) < (other.moves
                                    + other.heuristic)

FUNCTION __repr__():
    RETURN STRING representation of self.board
    with moves and heuristic

FUNCTION a_star(initial_board):
    SET initial_zero_pos = FIND position (i, j)
    WHERE initial_board[i][j] == 0
    SET initial_state = PuzzleState(initial_board,
        initial_zero_pos)
    INITIALIZE priority_queue as Empty
    HEAP_PUSH priority_queue WITH initial_state
    INITIALIZE visited as Empty set
    WHILE priority_queue is not Empty:
    SET current_state = HEAP_POP priority_queue
```

```
PRINT "Exploring State:", current_state
IF current_state.heuristic == 0:
PRINT "Reached goal in", current_state.
moves, "moves."
RETURN
ADD tuple representation of current_state.
board TO visited
FOR (neighbor_board, neighbor_zero_pos)
IN current_state.get_neighbors():
SET neighbor_state = PuzzleState(neighbor.
board, neighbor_zero_pos, current_state.
moves + 1)
IF tuple representation of neighbor_board
NOT IN visited:
HEAP_PUSH priority_queue WITH neighbor_state

PRINT "No Solution Found"
FUNCTION main():
SET initial_board = [
[2, 8, 3],
[1, 6, 4],
[0, 7, 5]
]
CALL a_star(initial_board)
CALL main()
```

Output:

Exploring state: [[2, 8, 3], [1, 6, 4], [0, 7, 5]]
(moves: 0, Heuristic: 6)
Exploring state: [[2, 8, 3], [1, 6, 4], [7, 0, 5]]
(moves: 1, Heuristic: 5)
Exploring state: [[2, 8, 3]

(moves : 2, Heuristic : 4)
Exploring state : [[1, 0, 3], [1, 8, 4], [7, 6, 5]]
(moves : 3, Heuristic : 3)
Exploring state : [[0, 2, 3], [1, 8, 4], [7, 6, 5]]
(moves : 4 ; Heuristic : 2)
Exploring state : [[1, 2, 3], [0, 8, 4], [7, 6, 5]]
(moves : 5, Heuristic : 1)
Exploring state : [[1, 2, 3], [8, 0, 4], [7, 6, 5]]
(moves : 6, Heuristic : 0)