A* ALGORITHM USING MANHATTAN DISTANCE

CODE :

```
print(f"Santhosh N (1BM23CS302)")
print(f"A* ALGORITHM USING MANHATTAN DISTANCE")

from queue import PriorityQueue

def manhattan_distance(state, goal):
    """Calculate total Manhattan distance of tiles from their goal positions."""
    distance = 0
    for tile in '12345678':
        current_index = state.index(tile)
        goal_index = goal.index(tile)
        current_row, current_col = divmod(current_index, 3)
        goal_row, goal_col = divmod(goal_index, 3)
        distance += abs(current_row - goal_row) + abs(current_col - goal_col)
    return distance

def get_neighbors(state):
    """Generate all possible states by sliding a tile into the blank space."""
    neighbors = []
    blank_idx = state.index('_')
    row, col = divmod(blank_idx, 3)

    moves = []
    if row > 0: moves.append(blank_idx - 3)
    if row < 2: moves.append(blank_idx + 3)
    if col > 0: moves.append(blank_idx - 1)
    if col < 2: moves.append(blank_idx + 1)

    for move in moves:
        new_state = list(state)
        new_state[blank_idx], new_state[move] = new_state[move], new_state[blank_idx]
        neighbors.append(''.join(new_state))

    return neighbors

def reconstruct_path(came_from, current):
    """Reconstruct the path from start to goal."""
    path = [current]
    while current in came_from:
        current = came_from[current]
        path.append(current)
    path.reverse()
    return path
```

A* ALGORITHM USING MANHATTAN DISTANCE

```
def a_star(start, goal):
    """A* search algorithm with Manhattan distance heuristic."""
    open_set = PriorityQueue()
    open_set.put((manhattan_distance(start, goal), 0, start))

    came_from = {}
    g_score = {start: 0}

    while not open_set.empty():
        f, g, current = open_set.get()

        if current == goal:
            return reconstruct_path(came_from, current)

        for neighbor in get_neighbors(current):
            tentative_g_score = g + 1

            if neighbor not in g_score or tentative_g_score < g_score[neighbor]:
                came_from[neighbor] = current
                g_score[neighbor] = tentative_g_score
                f_score = tentative_g_score + manhattan_distance(neighbor, goal)
                open_set.put((f_score, tentative_g_score, neighbor))

    return None

def print_state(state):
    """Print the 8-puzzle state in 3x3 format."""
    for i in range(0, 9, 3):
        print(state[i:i+3].replace('_', ' '))
    print()

def valid_state(state):
    """Check if input state is valid (must contain 1-8 and _ exactly once)."""
    return (
        len(state) == 9 and
        set(state) == set('12345678_') and
        all(state.count(ch) == 1 for ch in '12345678_')
    )

if __name__ == "__main__":

    while True:
        start_state = input("Start: ").strip()
```

```
    if valid_state(start_state):
        break
    print("Invalid input. Try again.")


while True:
    goal_state = input("Goal: ").strip()
    if valid_state(goal_state):
        break
    print("Invalid input. Try again.")

print("\nSolving...\n")
solution = a_star(start_state, goal_state)

if solution:
    print(f"Solution found in {len(solution) - 1}th Depth\n")
    for step in solution:
        print_state(step)
else:
    print("No solution found.")
print(f"TOTAL COST {len(solution) - 1}\n")
```

A* ALGORITHM USING MANHATTAN DISTANCE

OUTPUT :

```
>
= RESTART: C:/Users/student/AppData/Local/Programs/Python/Python313/302/lab4_man
.py
Santhosh N (1BM23CS302)
A* ALGORITHM USING MANHATTAN DISTANCE
Start: 2831647_5
Goal: 1238_4765

Solving...

Solution found in 5th Depth

283
164
7 5

283
1 4
765

2 3
184
765

 23
184
765

123
 84
765

123
8 4
765

TOTAL COST 5
```