# PRACTICAL 3

## Objective

Solve 8 puzzle problem using A* algorithm where initial state and Goal state will be given by the users.

## Program

```
import numpy as np


# Function to get matrix input from the user
def get_matrix_input(prompt):
    print(prompt)
    matrix = []
    for i in range(3):
        # Get each row of the matrix from the user
        row = list(
            map(
                int,
                input(
                    "Enter row {} (separate numbers with space): ".format(i + 1)
                ).split(),
            )
        )
        matrix.append(row)
    return np.array(matrix)


# Function to calculate heuristic of a matrix
def heuristic(matrix, end_matrix):
    # Compare each element of the matrix with the end matrix
    res = matrix == end_matrix
    # Return the number of elements that are not in their correct position
    return 9 - np.count_nonzero(res)
```

```
# Function to generate possible children of a matrix
def possibleChildren(matrix, e_matrix):
    visited.append(matrix)
    [i], [j] = np.where(matrix == 0)  # Find the position of the empty space (0)
    direction = [
        [-1, 0],
        [0, -1],
        [1, 0],
        [0, 1],
    ]  # Possible directions to move the empty space
    children = []
    for dir in direction:
        ni = i + dir[0]
        nj = j + dir[1]
        newMatrix = matrix.copy()
        # Check if the move is within the bounds of the matrix
        if ni >= 0 and ni <= 2 and nj >= 0 and nj <= 2:
            # Swap the empty space with the adjacent element
            newMatrix[i, j], newMatrix[ni, nj] = matrix[ni, nj], matrix[i, j]
            # Check if the new matrix has been visited before
            if not (any(np.array_equal(newMatrix, i) for i in visited)):
                visited.append(newMatrix)
                newMatrix_heu = heuristic(newMatrix, end_matrix)
                children.append([newMatrix_heu, newMatrix])

    # Sort the children based on their heuristic
    children = sorted(children, key=lambda x: x[0])
    for i in range(len(children)):
        children[i] = children[i][1]
```

```
    return children


# Function to solve the 8-puzzle problem using A* algorithm
def a_star_8_puzzle(start_matrix, end_matrix):
    start_heuristic = heuristic(start_matrix, end_matrix)
    if start_heuristic == 0:
        for node in closed:
            print(node)
        return True
    else:
        children = possibleChildren(start_matrix, end_matrix)
        if len(children) > 0:
            for i in range(len(children)):
                open.insert(i, children[i])

        if len(open) > 0:
            newHeu = heuristic(open[0], end_matrix)
            newMatrix = open[0]
            closed.append(open[0])
            open.pop(0)

            if newHeu == 0:
                for node in closed:
                    print(node)
                return True
            else:
                a_star_8_puzzle(newMatrix, end_matrix)
        else:
            return False

# Get the start and end matrices from the user
```

```
start_matrix = get_matrix_input("Enter the start matrix:")
end_matrix = get_matrix_input("Enter the end matrix:")

visited = []
open = []
closed = []

closed.append(start_matrix)

if __name__ == "__main__":
    a_star_8_puzzle(start_matrix, end_matrix)
```

## Output

```
Enter the start matrix:
Enter row 1 (separate numbers with space): 2 8 3
Enter row 2 (separate numbers with space): 1 6 4
Enter row 3 (separate numbers with space): 7 0 5
Enter the end matrix:
Enter row 1 (separate numbers with space): 1 2 3
Enter row 2 (separate numbers with space): 8 0 4
Enter row 3 (separate numbers with space): 7 6 5
```

```
[[2 8 3]
 [1 6 4]
 [7 0 5]]
[[2 8 3]
 [1 0 4]
 [7 6 5]]
[[2 0 3]
 [1 8 4]
 [7 6 5]]
[[0 2 3]
 [1 8 4]
 [7 6 5]]
[[1 2 3]
 [0 8 4]
 [7 6 5]]
[[1 2 3]
 [8 0 4]
 [7 6 5]]
```