AI – Practical 3

Vraj Chetan Patel
21BCP362
G11

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

Algorithm

1. **Importing the Necessary Library**
   - The code imports the NumPy library for numerical operations.
2. **Defining the read_matrix Function**
   - The function displays a prompt to the user.
   - It then reads and converts a 3x3 matrix from user input into a NumPy array.
3. **Initializing Matrices and Lists**
   - The code initializes the start and end matrices by calling **read_matrix**.
   - Lists for **visited**, **open**, and **closed** states are initialized.
4. **Appending Start Matrix to Closed List**
   - The start matrix is appended to the **closed** list.
5. **Defining the heuristic Function**
   - The function calculates and returns the heuristic value, based on the difference between a given matrix and the end matrix.
6. **Defining the possibleChildren Function**
   - The current matrix is marked as visited.
   - The function locates the position of zero (empty space) in the matrix.
   - Possible movements (up, down, left, right) are defined.
   - For each movement: a. The function checks if the movement is within bounds. b. If valid, it swaps the empty space with an adjacent element, creating a new matrix. c. If the new matrix is not previously visited, its heuristic is calculated, it's marked as visited, and added to the list of children.
   - The list of children matrices is sorted based on their heuristic value.
7. **Defining the main Function**
   - The heuristic for the start matrix is calculated.
   - If the start matrix matches the end matrix, the solution path is printed.
   - Otherwise, possible children of the start matrix are added to the **open** list.
   - While the **open** list is not empty: a. The first matrix is removed from **open** and its heuristic is calculated. b. This matrix is added to **closed**. c.

If this matrix matches the end matrix, the solution path is printed; if not, its children are added to **open**.
   - If no solution is found, the function returns **False**.
8. **Executing the Main Function**
   - The **main** function is executed if the script is the main program

CODE:-

```
import numpy as np

def read_matrix(prompt):
    print(prompt)  # Displaying the prompt to the user.
    # Reading a 3x3 matrix from user input and converting it to a numpy array.
    return np.array([list(map(int, input("Enter row {} (separated by spaces): ".format(i+1)).split())) for i in range(3)])

# Reading the start and end matrices from the user.
start_matrix = read_matrix("Enter the start matrix (3x3) row by row:")
end_matrix = read_matrix("Enter the end matrix (3x3) row by row:")

# Initializing lists for visited, open, and closed matrices.
visited = []
open = []
closed = []

closed.append(start_matrix)  # Adding the start matrix to the closed list.

def heuristic(matrix, end_matrix):
    # Calculating the heuristic as the count of non-matching elements with the end matrix.
    return 9 - np.count_nonzero(matrix == end_matrix)

def possibleChildren(matrix, e_matrix):
    visited.append(matrix)  # Marking the current matrix as visited.
    [i],[j] = np.where(matrix == 0)  # Finding the position of the zero (empty space).
    # Defining possible directions to move the empty space.
    direction = [[-1, 0], [0, -1], [1, 0],[0, 1]]
    children = []

    for dir in direction:
        ni, nj = i + dir[0], j + dir[1]
        if 0 <= ni <= 2 and 0 <= nj <= 2:  # Checking if the new position is valid.
            newMatrix = matrix.copy()
            # Swapping the empty space with the adjacent element.
            newMatrix[i, j], newMatrix[ni, nj] = matrix[ni, nj], matrix[i, j]
            # Checking if the new matrix is not already visited.
            if not any(np.array_equal(newMatrix, visited_mat) for visited_mat in visited):
                visited.append(newMatrix)  # Marking the new matrix as visited.
                # Calculating heuristic for the new matrix.
                newMatrix_heu = heuristic(newMatrix, e_matrix)
                children.append([newMatrix_heu, newMatrix])
```

```
    # Sorting children matrices based on their heuristic value.
    children.sort(key=lambda x: x[0])
    return [child[1] for child in children]

def main(start_matrix, end_matrix):
    # Calculating the heuristic value for the start matrix.
    start_heuristic = heuristic(start_matrix, end_matrix)

    if start_heuristic == 0:  # Checking if the start matrix is already the end matrix.
        for node in closed:
            print(node)
        return True
    else:
        # Getting possible children for the start matrix.
        children = possibleChildren(start_matrix, end_matrix)
        for child in children:
            open.append(child)  # Adding children to the open list.

        while open:
            newMatrix = open.pop(0)  # Taking the first matrix from the open list.
            newHeu = heuristic(newMatrix, end_matrix)
            closed.append(newMatrix)  # Adding the new matrix to the closed list.

            if newHeu == 0:  # Checking if the new matrix is the end matrix.
                for node in closed:
                    print(node)
                return True
            else:
                # Getting possible children for the new matrix.
                children = possibleChildren(newMatrix, end_matrix)
                for child in children:
                    open.append(child)  # Adding children to the open list.

        return False

if __name__ == "__main__":
    main(start_matrix, end_matrix)  # Executing the main function.
```

OUTPUT-

```
Enter the start matrix (3x3) row by row:
Enter row 1 (separated by spaces): 2 8 3
Enter row 2 (separated by spaces): 1 6 4
Enter row 3 (separated by spaces): 7 0 5
Enter the end matrix (3x3) row by row:
Enter row 1 (separated by spaces): 1 2 3
Enter row 2 (separated by spaces): 8 0 4
Enter row 3 (separated by spaces): 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]]
```