**PRACTICAL 3**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Name:** | Harsh Shah | **Semester:** | VI | **Division:** | 6 |
| **Roll No.:** | 21BCP359 | **Date:** |  | **Batch:** | G11 |
| **Aim:** | 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**



