ARTIFICIAL INTELLIGENCE

Name :- Shreyash Tekade PRN No. :- 12110840 Roll No. : 71

Problem Statement : - Implement Simple Hill Climb and Steepest Ascent

Code :-

```
import copy
import numpy as np
class main:
    def __init__(self):
        self.moves = 0
        self.matrix = [[2, 8, 3], [1, 6, 4], [7, 0, 5]]
        self.goal_state = [[1, 2, 3], [8, 0, 4], [7, 6, 5]]
        self.i_blank = 0
        self.j_blank = 0
        # self.matrix = [[0, 1, 3], [4, 2, 5], [7, 8, 6]]
        # self.goal_state = [[1, 2, 3], [4, 5, 6], [7, 8, 0]]
        # self.i_blank = 0
        # self.j_blank = 0
        self.outputMatrix = [[]]

        self.f_A = np.sum(np.sqrt(abs(np.square(np.array(self.matrix)) -
np.square(np.array(self.goal_state)))))

    # def takeInput(self):
    #     self.i_blank = int(input("Enter the blank postion(row)"))
    #     self.j_blank = int(input("Enter the blank position(column)"))
    #     for i in range(3):
    #         for j in range(3):
    #             if(i == self.i_blank and j == self.j_blank):
```

```
    #             continue
    #         number : int = int(input(f"Enter the number at position {i}th row and {j}th column"))
    #         if(number <=8 and number >= 1):
    #             self.matrix[i][j] = number
    #         else:
    #             print("Not a valid number")


    #   print(self.matrix)


    def generateAllMoves(self):
        self.outputMatrix = [copy.deepcopy(self.matrix) for i in range(4)]
        n = self.i_blank
        p = self.j_blank
        self.possibleMoves = [(n - 1, p), (n, p - 1), (n, p + 1), (n + 1, p)]
        if((self.i_blank == 0 or self.i_blank == len(self.matrix) - 1) and (self.j_blank == 0 or self.j_blank ==
len(self.matrix) - 1)):
            self.moves = 2
            # print(self.outputMatrix)
            count = 0
            for i in range(4):
                if((self.possibleMoves[i][0] >= 0 and self.possibleMoves[i][0] <= 2) and
(self.possibleMoves[i][1] >= 0 and self.possibleMoves[i][1] <= 2)):
                    # print(self.possibleMoves[i])
                    self.outputMatrix[count][n][p] =
self.matrix[self.possibleMoves[i][0]][self.possibleMoves[i][1]]
                    self.outputMatrix[count][self.possibleMoves[i][0]][self.possibleMoves[i][1]] = 0
                    count = count + 1
        elif(self.i_blank + self.j_blank == 1 or self.j_blank == 3):
            self.moves = 3
            count = 0
            for i in range(4):
                if((self.possibleMoves[i][0] >= 0 and self.possibleMoves[i][0] <= 2) and
(self.possibleMoves[i][1] >= 0 and self.possibleMoves[i][1] <= 2)):
```

```
                # print(self.possibleMoves[i])
                self.outputMatrix[count][n][p] =
self.matrix[self.possibleMoves[i][0]][self.possibleMoves[i][1]]
                self.outputMatrix[count][self.possibleMoves[i][0]][self.possibleMoves[i][1]] = 0
                count = count + 1
        else:
            self.moves = 4
            count = 0
            for i in range(4):
                if((self.possibleMoves[i][0] >= 0 and self.possibleMoves[i][0] <= 2) and
(self.possibleMoves[i][1] >= 0 and self.possibleMoves[i][1] <= 2)):
                # print(self.possibleMoves[i])
                self.outputMatrix[count][n][p] =
self.matrix[self.possibleMoves[i][0]][self.possibleMoves[i][1]]
                self.outputMatrix[count][self.possibleMoves[i][0]][self.possibleMoves[i][1]] = 0
                count = count + 1

        return self.outputMatrix

    def calculateBestMoves(self):
        self.s = []
        for i in range(self.moves):
            array = np.array(self.outputMatrix[i])
            self.s.append(np.sum(np.sqrt(abs(np.square(array) - np.square(np.array(self.goal_state))))))
        # print(self.s)
        return self.s


class Node:
    def __init__(self, matrix, g_A, h_A) -> None:
        self.matrix = matrix
```

```
        self.g_A = g_A
        self.h_A = h_A
        self.f_A = self.g_A + self.h_A



class HillClimbing:
    def __init__(self) -> None:
        self.puzzle = main()
        self.outputMatrix = self.puzzle.generateAllMoves()
        self.ed = self.puzzle.calculateBestMoves()

    def chooseMove(self):
        n = self.puzzle.moves
        j = 0
        min = self.puzzle.f_A
        allMoves = []
        print("f_A -> for the starting state", min)

        for i in range(n):
            temp = Node(self.outputMatrix[i], 1, self.ed[i])
            allMoves.append(temp)
            print(temp.matrix, " ", temp.f_A)

        for i in range(n):
            if(min > allMoves[i].f_A):
                j = allMoves[i].f_A
                break

        return j
```

```
class Steepest:
    def __init__(self) -> None:
        self.puzzle = main()
        self.outputMatrix = self.puzzle.generateAllMoves()
        self.ed = self.puzzle.calculateBestMoves()

    def chooseMove(self):
        n = self.puzzle.moves
        j = 0
        min = self.puzzle.f_A
        allMoves = []
        print("f_A ->", min)

        for i in range(n):
            temp = Node(self.outputMatrix[i], 1, self.ed[i])
            allMoves.append(temp)
            print(temp.matrix, " ", temp.f_A)

        for i in range(n):
            if(min > allMoves[i].f_A):
                j = allMoves[i].f_A

        return j


print("--------------------------Hill Climbing----------------------")
hill = HillClimbing()
move = hill.chooseMove()
print("Best move is ::", end=" ")
print(move)
```

```
print("--------------------------Steepest----------------------")


steep = Steepest()

move = steep.chooseMove()

print("Best move is:: ", end = " ")

print(move)
```

Output :-

```
@Shreyash ➜ Lab3 python main.py
---------------------------Hill Climbing-----------------
f_A -> for the starting state 29.415271433177484
[[8, 0, 3], [1, 6, 4], [7, 0, 5]]   30.874507866387546
[[1, 8, 3], [0, 6, 4], [7, 0, 5]]   28.745966692414832
Best move is :: 28.745966692414832
---------------------------Steepest----------------------
f_A -> 29.415271433177484
[[8, 0, 3], [1, 6, 4], [7, 0, 5]]   30.874507866387546
[[1, 8, 3], [0, 6, 4], [7, 0, 5]]   28.745966692414832
Best move is::  28.745966692414832
```