ARTIFICIAL INTELLIGENCE

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

Problem Statement :- Implement A* Algorithm

Code :-

```
# Goals

# Wrie a program that takes a board position from a user and generate all possible moves in the computer memory

# Implementing priority queue to store the best puzzle moves

# A* algorithm to solve the 8 puzzle problem

import copy

import numpy as np

import queue


class Node:
    def __init__(self, matrix, parent, goal_state, h2_A, g_A):
        self.matrix = matrix
        self.parent = parent
        self.goal_state = goal_state
        self.g_A = g_A
        self.h1_A = self.calculatemisplaced(self.goal_state)
        self.h2_A = h2_A
        self.f_A = g_A + h2_A
        self.next = None
        self.value = self.f_A

    def calculatemisplaced(self, goal_state):
        count : int = 0
        for i in range(3):
            for j in range(3):
```

```
            if(self.matrix[i][j] != goal_state[i][j]):
                count += 1
        return count

class PriorityQueue:
    def __init__(self, goal_state, starting_state):
        self.head : Node = None
        self.tail : Node = None
        self.goal_state = goal_state
        self.starting_state = starting_state

    def insert(self, node : Node, position : int = None):
        if position == None:

            if self.head is None:
                self.head = node
                self.tail = node
                return
            if self.head.value > node.value:
                node.next = self.head
                self.head = node
                return
            prev = self.head
            current = self.head.next

            while current is not None and current.value <= node.value:
                prev = current
                current = current.next

            prev.next = node
            node.next = current
```

```
    # else:


    #     if position == 0:
    #         node.next = self.head
    #         self.head = node
    #         return
    #     count : int = 0
    #     temp = self.head
    #     while(count <= position - 1):
    #         temp = temp.next
    #         count = count + 1
    #     _temp = temp.next.next
    #     temp.next.next = None
    #     temp.next = node
    #     node.next = _temp
def INSERT(self, matrix, parent, goal_state, h2_A, g_A):
    node = Node(matrix, parent, goal_state, h2_A, g_A)
    if self.head is None:
        self.head = node
        self.tail = node
        return
    else:
        start = self.head
        while start.next is not None:
            start = start.next
        node.next = None
        start.next = node

    return self.head


def getMin(self):
```

```
    if self.head is None:
        return None
    if(self.head.next is None):
        temp = self.head
        self.head = None
        self.tail = None
        return temp
    temp = self.head
    self.head = self.head.next
    temp.next = None
    return temp

def isSafe(self):
    return self.goal_state == self.starting_state

def ifExists(self, state):
    temp = self.head
    count : int = 0
    while(temp is not None):
        if(np.array_equal(temp.matrix, state)):
            count = count + 1
            return True, count
        temp = temp.next
        count = count + 1

    return False, -1


def print(self):
    temp = self.head
    print("-------------------------------------")
```

```
        while temp is not None:
            print("->", temp.matrix,"value ->", temp.value)
            temp = temp.next

class main:
    def __init__(self):
        self.moves = 0
        # self.matrix = [[1, 2, 3], [5, 6, 0], [7, 8, 4]]
        # self.goal_state = [[1, 2, 3], [5, 8, 6], [0, 7, 4]]
        # self.matrix = [[0, 1, 3], [4, 2, 5], [7, 8, 6]]
        # self.goal_state = [[1, 2, 3], [4, 5, 6], [7, 8, 0]]



        # self.matrix = [[8, 6, 7], [2, 5, 4], [3, 0, 1]]
        # self.goal_state = [[6, 4, 7], [8, 5, 0], [3, 2 ,1]]



        self.matrix = [[1, 2, 3], [0, 4, 6], [7, 5, 8]]
        self.goal_state = [[1, 2, 3], [4, 5, 6], [7, 8, 0]]

        self.i_blank = 1
        self.j_blank = 0
        self.outputMatrix = [[]]



    # 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(abs(array - np.array(self.goal_state))))
        # print(self.s)
        return self.s


def calculateHeuristic(state, goal_state):
    return np.sum(abs(np.array(state) - np.array(goal_state)))
```

```
def updateParent(closed, open, newParent, parent, g_A):
    temp = closed
    temp1 = open
    while(temp is not None):
        if(parent == temp.parent):
            while(temp1 is not None):
                temp1.f_A = temp1.f_A - temp1.g_A + g_A + 1
                temp1.g_A = g_A + 1
                temp1 = temp1.next
            temp.parent = newParent
            temp.f_A = temp.f_A - temp.g_A + g_A + 1
            temp.g_A = g_A

        temp = temp.next



puzzle = main()
starting_state = puzzle.matrix
pq = PriorityQueue(puzzle.goal_state, starting_state)
pq.insert(Node(starting_state, None, puzzle.goal_state, calculateHeuristic(starting_state,
puzzle.goal_state), 0))
count = 0
temp = pq.head
q = queue.Queue()



pq_c = PriorityQueue(puzzle.goal_state, starting_state)


count : int = 0 # This is gA
flag = 0
while pq.head is not None and flag == 0:
```

```
temp = pq.getMin()
print(temp.matrix)
node_c = pq_c.INSERT(temp.matrix, temp.parent, temp.goal_state, temp.h2_A, temp.g_A)
_i = 0
_j = 0
q.put(temp)

g_A = temp.g_A
for i in range(3):
    for j in range(3):
        if(temp.matrix[i][j] == 0):
            _i = i
            _j = j
            break

if(temp.f_A - temp.g_A == 0):
    print("Goal State Found!")
    break

puzzle.matrix = temp.matrix
puzzle.i_blank = _i
puzzle.j_blank = _j
goal_states = puzzle.generateAllMoves()
scores = puzzle.calculateBestMoves()
# print("min", temp)
# print("goal_states", goal_states)
# print("scores", scores)
temp_ = pq.head
temp1 = pq_c.head
# print(temp_.matrix)
c = False
```

```
print(scores)
count = count + 1
for i in range(len(scores)):
    print(goal_states[i])
    print(scores[i])
    if(temp_ == None and temp1 == None):
        print("hi")
        pq.insert(Node(goal_states[i], temp.matrix, puzzle.goal_state, scores[i], g_A + 1))
    else:
        if(c == False):
            print(1)
            while(temp_ is not None):
                if(temp_.matrix == goal_states[i]):
                    print("bye")
                    if(temp_.f_A > scores[i] + g_A + 1):
                        print('hah')
                        temp_.f_A = scores[i] + g_A + 1
                        temp_.g_A = g_A + 1
                        temp_.parent = temp.matrix
                        c = True
                        break
                temp_ = temp_.next
        if(c == False):
            print(2)
            while(temp1 is not None):
                if(temp1.matrix == goal_states[i]):
                    print('Hello')
                    print("f_a", temp1.value)
                    print("scores", scores[i] + g_A + 1)
                    print("scoresi", scores[i])
                    print("g_A", g_A + 1)
```

```
                print("parent g_A", temp1.g_A)
                if(temp1.f_A > scores[i] + g_A + 1):
                    print("parent updated")
                    updateParent(pq_c.head, pq.head, temp.matrix, temp1.parent, g_A + 1)

                    c = True
                    break
            temp1 = temp1.next
        if(c == False):
            print(3)
            pq.insert(Node(goal_states[i], temp.matrix, puzzle.goal_state, scores[i], g_A + 1))
    pq.print()




pq_c.print()
while not q.empty():
    print(q.get().matrix)
    print("-----------------------------------------------")
```

Output :-

```
-> [[1, 2, 3], [5, 6, 4], [7, 8, 0]] value -> 17
-> [[1, 2, 3], [5, 6, 0], [7, 8, 4]] value -> 17
-> [[1, 2, 3], [5, 8, 6], [7, 4, 0]] value -> 17
-> [[1, 2, 3], [5, 8, 6], [7, 0, 4]] value -> 17
-> [[1, 0, 2], [5, 6, 3], [7, 8, 4]] value -> 18
-> [[1, 2, 3], [5, 6, 0], [7, 8, 4]] value -> 18
-> [[1, 0, 3], [5, 2, 6], [7, 8, 4]] value -> 18
-> [[1, 2, 3], [0, 5, 6], [7, 8, 4]] value -> 18
-> [[1, 2, 3], [5, 6, 0], [7, 8, 4]] value -> 18
-> [[1, 2, 3], [5, 0, 6], [7, 8, 4]] value -> 19
[[1, 2, 3], [5, 8, 6], [0, 7, 4]]
Goal State Found!
------------------------------------
-> [[1, 2, 3], [5, 6, 0], [7, 8, 4]] value -> 16
-> [[1, 2, 0], [5, 6, 3], [7, 8, 4]] value -> 17
-> [[1, 2, 3], [5, 0, 6], [7, 8, 4]] value -> 17
-> [[1, 2, 3], [5, 8, 6], [7, 0, 4]] value -> 16
-> [[1, 2, 3], [5, 8, 6], [0, 7, 4]] value -> 3
[[1, 2, 3], [5, 6, 0], [7, 8, 4]]
------------------------------------------------
[[1, 2, 0], [5, 6, 3], [7, 8, 4]]
------------------------------------------------
[[1, 2, 3], [5, 0, 6], [7, 8, 4]]
------------------------------------------------
[[1, 2, 3], [5, 8, 6], [7, 0, 4]]
------------------------------------------------
[[1, 2, 3], [5, 8, 6], [0, 7, 4]]
------------------------------------------------
```