Practical No. - 3

Name- Bhavesh Kewalramani

Roll No.- A-25

Section- A

Semester- 6th

Shift- 1st

Aim:

Write a program to implement Breadth First Search and A* to solve 8-Puzzle problem

Code:

```
from queue import Queue

from queue import PriorityQueue

from time import time


class EightPuzzle:

    goalState=[1,2,3,4,5,6,7,8,0]

    heuristicSearch=None

    evaluationFunction=None

    needsHueristic=False

    numOfInstances=0


    def __init__(self,state,prev,moves,pathCost,needsHueristic=False):

        self.prev=prev
```

```
        self.state=state
        self.moves=moves
        if prev:
            self.pathCost = prev.pathCost + pathCost
        else:
            self.pathCost = pathCost
        if needsHueristic:
            self.needsHueristic=True
            self.generateHeuristic()
            self.evaluationFunction=self.heuristicSearch+self.pathCost
        EightPuzzle.numOfInstances+=1

    def __str__(self):
        return str(self.state[0:3])+'\n'+str(self.state[3:6])+'\n'+str(self.state[6:9])

    def generateHeuristic(self):
        self.heuristicSearch=0
        for num in range(1,9):
            distance=abs(self.state.index(num) - self.goalState.index(num))
            i=int(distance/3)
            j=int(distance%3)
            self.heuristicSearch=self.heuristicSearch+i+j

    def ifGoal(self):
        if self.state == self.goalState:
```

```
            return True
        return False


    @staticmethod
    def findMoves(i,j):
        moves = ['U', 'D', 'L', 'R']
        if i == 0:  # up is disable
            moves.remove('U')
        elif i == 2:  # down is disable
            moves.remove('D')
        if j == 0:   #left is disable
            moves.remove('L')
        elif j == 2:  #right is disable
            moves.remove('R')
        return moves


    def generateState(self):
        states=[]
        x = self.state.index(0)
        i = int(x / 3)
        j = int(x % 3)
        moves=self.findMoves(i,j)


        for move in moves:
            new_state = self.state.copy()
```

```
            if move == 'U':
                new_state[x], new_state[x-3] = new_state[x-3], new_state[x]
            elif move == 'D':
                new_state[x], new_state[x+3] = new_state[x+3], new_state[x]
            elif move == 'L':
                new_state[x], new_state[x-1] = new_state[x-1], new_state[x]
            elif move == 'R':
                new_state[x], new_state[x+1] = new_state[x+1], new_state[x]
            states.append(EightPuzzle(new_state,self,moves,1,self.needsHueristic))
        return states

    def findGoal(self):
        goal = []
        goal.append(self.moves)
        path = self
        while path.prev != None:
            path = path.prev
            goal.append(path.moves)
        goal = goal[:-1]
        goal.reverse()
        return goal


def bfs(startState):
    start = EightPuzzle(startState, None, None, 0)
```

```
    if start.ifGoal():
        return start.findGoal()
    q = Queue()
    q.put(start)
    explored=[]
    while not(q.empty()):
        state=q.get()
        explored.append(state.state)
        states=state.generateState()
        for newState in states:
            if newState.state not in explored:
                if newState.ifGoal():
                    return newState.findGoal()
                q.put(newState)
    return



def AStar(startState):
    count=0
    explored=[]
    start=EightPuzzle(startState,None,None,0,True)
    q = PriorityQueue()
    q.put((start.evaluationFunction,count,start))

    while not q.empty():
```

```
        state=q.get()
        state=state[2]
        explored.append(state.state)
        if state.ifGoal():
            return state.findGoal()

        nstate=state.generateState()
        for state in nstate:
            if state.state not in explored:
                count += 1
                q.put((state.evaluationFunction,count,state))
    return

state=[1, 2, 3,
       5, 6, 0,
       7, 8, 4]

EightPuzzle.numOfInstances=0
t0=time()
bfs=bfs(state)
t1=time()-t0
print('BFS:', bfs)
print('space:',EightPuzzle.numOfInstances)
print('time:',t1)
print()
```

```
EightPuzzle.numOfInstances = 0

t0 = time()

astar = AStar(state)

t1 = time() - t0

print('A*:',astar)

print('space:', EightPuzzle.numOfInstances)

print('time:', t1)

print()

print('----------------------------------------')
```

Output:

```
print('----------------------------------------')

BFS: [['U', 'D', 'L'], ['U', 'D', 'L', 'R'], ['U', 'D', 'R'], ['U', 'R'], ['U', 'L', 'R'], ['U', 'L'], ['U', 'D', 'L'], ['U',
'D', 'L', 'R'], ['U', 'L', 'R'], ['U', 'R'], ['U', 'D', 'R'], ['U', 'D', 'L', 'R'], ['U', 'D', 'L']]
space: 6308
time: 0.13888311386108398

A*: [['U', 'D', 'L'], ['U', 'D', 'L', 'R'], ['U', 'D', 'R'], ['U', 'R'], ['U', 'L', 'R'], ['U', 'L'], ['U', 'D', 'L'], ['U',
'D', 'L', 'R'], ['U', 'L', 'R'], ['U', 'R'], ['U', 'D', 'R'], ['U', 'D', 'L', 'R'], ['U', 'D', 'L']]
space: 267
time: 0.005301237106323242

----------------------------------------
```