Name: PRANAV ARORA
USN: 1BM18CS072

Q A 8 puzzle using A* with max depth as 3 & heuristic as Manhattan distance

```
import random
import math

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

def index (item, sequence):
    if item in sequence
        return sequence.index(item)
    else:
        return -1

def distance (puzzle, item, total_calc, total_calc)
    t=0
    for row in range (3):
        for col in range (3):
            val = puzzle.get((row,col) -1
            target_col = val % 3
            target_row = val / 3

    if ( target_row < 0:
        target_row = 2
    t += item_total_calc(row, target_row, col,
                         target col)

return total_calc (t)
```

एकलव्य
Date: / /201
Pg No:

```
def manhattan(puzzle):
    return distance(puzzle,
                    lambda x1, y1, x2, y2:
                        abs(x2-x1)+abs(y2-y1),
                    lambda t: t)

class SlidePuzzle():
    def __init__(self):
        self.heuristic_value = 0
        self.depth = 0
        self.parent = None
        self.initMat = []
        for i in range(3):
            self.initMat.append(goalState
                                [i][:])

    def __eq__(self, other):
        return self.initMat == other.initMat

    def __str__(self):
        res = ''
        for row in range(3):
            rest = ''.join(map(str, self.
                               initMat[row]))
            res += '\n'
        return res
```

```
def replicate(self):
    a = SlidePuzzle()
    for i in range(3):
        a.initMat[i] = self.initMat[i][:]
    return a

def get_legal_moves(self):
    row,col = self.find(0)
    free = []
    if (row > 0):
        free.append((row-1, col))
    if (col > 0):
        free.append((row, col-1))
    if (row < 2):
        free.append((row, col))
    if (col < 2)
        free.append((row, col+1))
    return free

def generate_moves(self):
    free = self.get_legal_moves()
    zero = self.find(0)

    def swap_and_replicate(first, second)
        p = self.replicate()
        p.swap(first, second)
        p.depth = self.depth + 1
        p.parent = self
```

एकलव्य
Date: / /201
Pg.No:

return p

return (lambda pair: swap and replicate (zero, pair), free)

def generate_solution_path(self, path):
    if self.parent is None:
        return path
    else:
        path.append(self)
        return self.parent.generate_solution_path(path)

def solve(self, h):
    def is_solved(puzzle):
        return puzzle.initMat == goal_state

    openList = [self]
    closedList = []
    move_count = 0

    while len(openList) > 0
        x = openlist.pop(0)
        move_count += 1
        if is_solved(x)
            if(len(closedList) > 0:
                return x.generate_solution_path([]), move_count

else:
return [x]
successors = x.generate_moves()
index_open = index_closed = -1
for move in successors:
index_open = index(move, openlist)
index_closed = index(move, closed list)
heuristic_val = h(move)
if index_closed == -1 and index_open == -1:
move.heuristic_val = heuristic_value
openlist.append(move)
elif index_open > -1:
copy = openlist[index_open]
if Fval < copy.heuristicval + copy.depth
move.heuristic value = heuristic value
copy.parent = move.parent
copy.depth = move.depth
elif index_closed > -1:
copy = closed list[index_closed]
if Fval < copy.heuristicval + copy.depth:
move.heuristic val = heuristic val
closed list.remove(copy)
openlist.append(move)

if(move.depth) > 3:
print("Depth exceeded")
return([],0)

एकलव्य
Date: / /201
Pg No:

closed list.append(x)

open list = sorted(open list,
key = lambda p: p.heuristic value
+ p.depth)

return [], 0