Program 5

Implement the 8-puzzle problem using A* algorithm, using Heuristic function as Manhattan distance with depth not more the 3. If goal state is not reached within this limit, agent must report “NOSOLUTION”.

```python
from copy import deepcopy
import numpy as np
import time

# takes the input of current states and evaluvates the best path to goal state
def bestsolution(state):
    bestsol = np.array([], int).reshape(-1, 9)
    count = len(state) - 1
    while count != -1:
        bestsol = np.insert(bestsol, 0, state[count]['puzzle'], 0)
        count = (state[count]['parent'])
    return bestsol.reshape(-1, 3, 3)


# this function checks for the uniqueness of the iteration(it) state, weather it has been previously traversed
def all(checkarray):
    set=[]
    for it in set:
        for checkarray in it:
            return 1
        else:
            return 0


# calculate Manhattan distance cost between each digit of puzzle(start state) and the goal state
def manhattan(puzzle, goal):
    a = abs(puzzle // 3 - goal // 3)
    b = abs(puzzle % 3 - goal % 3)
    mhcost = a + b
    return sum(mhcost[1:])



# will calcuates the number of misplaced tiles in the current state as compared to the goal state
def misplaced_tiles(puzzle,goal):
    mscost = np.sum(puzzle != goal) - 1
    return mscost if mscost > 0 else 0


#3[on_true] if [expression] else [on_false]


# will indentify the coordinates of each of goal or initial state values
def coordinates(puzzle):
    pos = np.array(range(9))
    for p, q in enumerate(puzzle):
```

```
        pos[q] = p
    return pos



# start of 8 puzzle evaluvation, using Manhattan heuristics
def evaluvate(puzzle, goal):
    steps = np.array([('up', [0, 1, 2], -3),('down', [6, 7, 8],  3),('left', [0, 3, 6], -1),('right', [2, 5, 8],  1)
                dtype =  [('move',  str, 1),('position', list),('head', int)])

    dtstate = [('puzzle',  list),('parent', int),('gn',  int),('hn',  int)]

     # initializing the parent, gn and hn, where hn is manhattan distance function call
    costg = coordinates(goal)
    parent = -1
    gn = 0
    hn = manhattan(coordinates(puzzle), costg)
    state = np.array([(puzzle, parent, gn, hn)], dtstate)

# We make use of priority queues with position as keys and fn as value.
    dtpriority = [('position', int),('fn', int)]
    priority = np.array( [(0, hn)], dtpriority)

```

Output

```
Given StartNode is:  [[8, 2, 3], [0, 4, 6], [7, 5, 1]]

        Given GoalNode is:  [[1, 2, 3], [4, 5, 6], [7, 8, 0]]

####################################

        h1 : Number of misplaced tiles => 4

####################################

Distances of the tiles from their goal positions are:

        [3]
        [3, 0]
        [3, 0, 0]
        [3, 0, 0, 1]
        [3, 0, 0, 1, 0]
        [3, 0, 0, 1, 0, 0]
        [3, 0, 0, 1, 0, 0, 1]
        [3, 0, 0, 1, 0, 0, 1, 4]

####################################

h2 :  The sum of the distances of the tiles from their goal positions => 9

       So, the instance of given 8-puzzle solution is 13 steps long.
```