Assignment 6

Name: Krunal Rank
Roll No: U18CO081

IImplement A* algorithm and AO* algorithm in python without using existing libraries.
Consider n cities and generate distance between each pair of city with the help of random
function. You can provide range in a random number generating function.
Input/Output of the code:
When user enters 2 cities, program should provide the shorter path between the two cities using
both algorithms separately.
Compare the complexity of both algorithms for the same set of input.

Code:

```
'''
   Code by KRHero
   IDE: VSCode
'''
import numpy as np
from collections import defaultdict
from queue import PriorityQueue
import time
import networkx as nx
import matplotlib.pyplot as plt

class Graph:

   def __init__(self,n,coordinates,edges):
       '''
       Graph class Constructor
       Paramters:
       n - Number of nodes
       coordinates - List of coordinates of n nodes of the form (latitude, longitude)
       edges - List of edges of the form (starting node,ending node)
       Returns:
       Graph object
       '''
       self.n = n
       self.coordinates = coordinates
```

```
        self.rawEdges = edges
        d = defaultdict(list)
        for t in edges:
            u,v = t
            u = u - 1
            v = v - 1
            d[u].append(v)
            d[v].append(u)

        self.edges = d

    def displayGraph(self):
        '''
        Displays graph using networkx and matplotlib
        '''
        g = nx.Graph()
        for i in range(1,self.n+1):
            g.add_node(i,pos=self.coordinates[i-1])

        for i in self.rawEdges:
            g.add_edge(i[0],i[1],weight =
round(self.getEuclidianDistance(i[0]-1,i[1]-1),2))

        fig, ax = plt.subplots()

        pos1 = nx.get_node_attributes(g,'pos')
        nx.draw(g,pos1,ax=ax,with_labels = True)

        edge_labels=dict([((u,v,),d['weight']) for u,v,d in g.edges(data=True)])
        nx.draw_networkx_edge_labels(g,pos1,edge_labels = edge_labels)

        limits=plt.axis('on')
        ax.tick_params(left=True, bottom=True, labelleft=True, labelbottom=True)
        plt.show()

    def a_star(self,start,end):
        '''
            A star algorithm to find shortest distance between start node and end node
            Parameters:
            start - starting node
            end - ending node
            Returns:
```

```
        A list containing optimal path from start node to end node
        Shortest Distance
        No. of nodes generated
        No. of nodes expandeds
    '''

    if type(start) != int or type(end) != int:
        return None

    if start>self.n or end > self.n or start<1 or end<1:
        return None
    start = start - 1
    end = end - 1
    closed = list()

    frontier = PriorityQueue()

    parents = [-1]*self.n
    heuristicValues = [0]*self.n

    h = self.getHeuristicValue(0,self.getManhattanDistance(start,end))
    frontier.put((h,0,start,-1))

    nodesGenerated = 1
    nodesExpanded = 0

    path = []

    while frontier.empty()==False:
        h,f,nodeIdx,parent = frontier.get()
        parents[nodeIdx] = parent
        heuristicValues[nodeIdx] = f
        closed.append(nodeIdx)
        if nodeIdx == end:
            while(parents[nodeIdx]!=-1):
                path.append(nodeIdx+1)
                nodeIdx = parents[nodeIdx]

            path.append(nodeIdx+1)
            path.reverse()
            return (path,f,nodesGenerated,nodesExpanded)
```

```
            nodesExpanded = nodesExpanded + 1

            for edge in self.edges[nodeIdx]:
                nodeIdx1 = edge
                weight = self.getEuclidianDistance(nodeIdx1,nodeIdx)
                f1 = f + weight
                h1 =
self.getHeuristicValue(f1,self.getManhattanDistance(nodeIdx1,end))
                if nodeIdx1 not in closed or (f1 < heuristicValues[nodeIdx1]):
                    nodesGenerated = nodesGenerated + 1
                    frontier.put((h1,f1,nodeIdx1,nodeIdx))

        return (None,None,None,None)

    def getManhattanDistance(self,node1,node2):
        '''
        Returns Manhattan Distance between Node 1 and Node 2
        Parameters:
        node1 - index of first node
        node2 - index of second node
        Returns:
        Manhattan Distance between node1 and node2
        '''
        return abs(self.coordinates[node1][0]-self.coordinates[node2][0]) +
abs(self.coordinates[node1][1] - self.coordinates[node2][1])

    def getEuclidianDistance(self,node1,node2):
        '''
        Returns Euclidian Distance between Node 1 and Node 2
        Parameters:
        node1 - index of first node
        node2 - index of second node
        Returns:
        Euclidian Distance between node1 and node2
        '''
        return ((self.coordinates[node1][0]-self.coordinates[node2][0])**2 +
(self.coordinates[node1][1]-self.coordinates[node2][1])**2)**0.5

    def getHeuristicValue(self,f,g):
        '''
        Returns heuristic value for value of f and g
        Parameters:
```

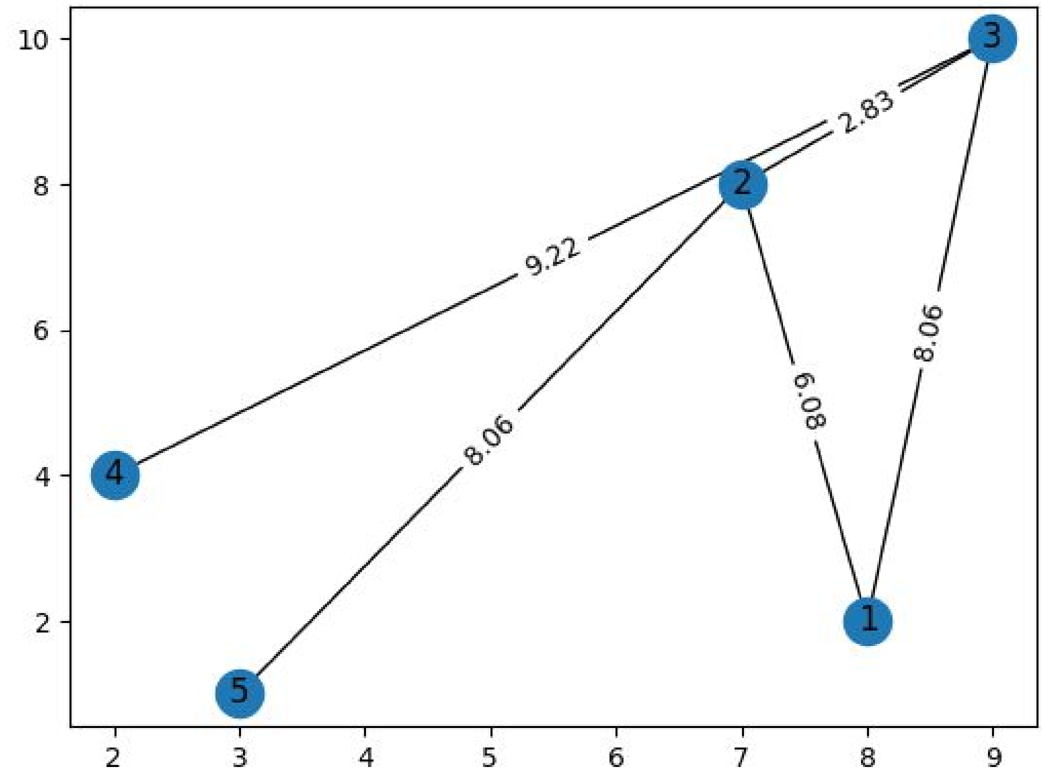
```
        f - distance between node and start node
        g - manhattan distance between node and end node
        '''
        return (f+g)



g = Graph(5,[(8,2),(7,8),(9,10),(2,4),(3,1)],[(1,2),(2,3),(1,3),(2,5),(3,4)])
start = 2
end = 1
print('--------------------------------A Star
Algorithm---------------------------------------')
startTime = time.time()
path,dist, nodesGenerated,nodesExpanded = g.a_star(start,end)
endTime = time.time()
executionTime = endTime - startTime
print('Starting Node:- {}'.format(start))
print('Destination Node:- {}'.format(end))
print('Execution Time:- {:.4f}ms'.format(executionTime*1000))
print('Distance:- {}'.format(dist))
print('Nodes Generated:- {}'.format(nodesGenerated))
print('Nodes Expanded:- {}'.format(nodesExpanded))
print('Path:- {}'.format(path))
print('--------------------------------------------------------------------------------
---------')

g.displayGraph()
```

Demo:




```
-------------------------------A Star Algorithm--------------------------------------
Starting Node:- 2
Destination Node:- 1
Execution Time:- 0.1323ms
Distance:- 6.082762530298219
Nodes Generated:- 4
Nodes Expanded:- 1
Path:- [2, 1]
-------------------------------------------------------------------------------------
```

```
-------------------------------A Star Algorithm--------------------------------------
Starting Node:- 5
Destination Node:- 4
Execution Time:- 0.2241ms
Distance:- 20.110229330337624
Nodes Generated:- 6
Nodes Expanded:- 4
Path:- [5, 2, 3, 4]
-------------------------------------------------------------------------------------
```

```
-------------------------------A Star Algorithm--------------------------------------
Starting Node:- 1
Destination Node:- 4
Execution Time:- 0.1764ms
Distance:- 17.281802205591436
Nodes Generated:- 6
Nodes Expanded:- 4
Path:- [1, 3, 4]
-------------------------------------------------------------------------------------
```

As far as AO star algorithm is concerned, it is a problem solving / decision making algorithm and is not suitable for path finding problems. Hence, it is not implemented.

Even if I implement it, it will work just like A star algorithm because there will not be any AND arcs in PathFinding problems.