U18CO018

Shubham Shekhaliya

AIML

Assignment – 6

**Implement A\* algorithm and AO\* algorithm in python without using existing libraries.**

**Compare the complexity of both algorithms for same set of input.**

Consider n cities and generate distance between each pair of city with the help of random function. You can provide range in 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.

(As we know AO\* Algorithm is a Problem Solving Algorithm, which decomposes the problem into subproblem and solution is generated by combining those solution and represented in form of AND-OR Graph. While This is a Path Finding Problem in which there will be no AND arcs. So, AO\* Algorithm isn't implemented here. And even if it is implemented, it won't work any different from A\* Algorithm.)

**Code**

import random

class Graph:

    def \_\_init\_\_(self, graph\_dict=None, directed=True):

        self.graph\_dict = graph\_dict or {}

        self.directed = directed

        if not directed:

            self.make\_undirected()

    def make\_undirected(self):

        for a in list(self.graph\_dict.keys()):

            for (b, dist) in self.graph\_dict[a].items():

                self.graph\_dict.setdefault(b, {})[a] = dist

    def connect(self, A, B, distance=1):

        self.graph\_dict.setdefault(A, {})[B] = distance

        if not self.directed:

            self.graph\_dict.setdefault(B, {})[A] = distance

    def get(self, a, b=None):

        links = self.graph\_dict.setdefault(a, {})

        if b is None:

            return links

        else:

            return links.get(b)

    def nodes(self):

        s1 = set([k for k in self.graph\_dict.keys()])

        s2 = set([k2 for v in self.graph\_dict.values() for k2, v2 in v.items()])

        nodes = s1.union(s2)

        return list(nodes)

class Node:

    def \_\_init\_\_(self, name:str, parent:str):

        self.name = name

        self.parent = parent

        self.g = 0

        self.h = 0

        self.f = 0

    def \_\_eq\_\_(self, other):

        return self.name == other.name

    def \_\_lt\_\_(self, other):

         return self.f < other.f

    def \_\_repr\_\_(self):

        return ('({0},{1})'.format(self.name, self.f))

def astar\_search(graph, heuristics, start, end):

    open = []

    closed = []

    start\_node = Node(start, None)

    goal\_node = Node(end, None)

    open.append(start\_node)

    while len(open) > 0:

        open.sort()

        current\_node = open.pop(0)

        closed.append(current\_node)

        if current\_node == goal\_node:

            path = []

            while current\_node != start\_node:

                path.append(current\_node.name + ': ' + str(current\_node.g))

                current\_node = current\_node.parent

            path.append(start\_node.name + ': ' + str(start\_node.g))

            return path[::-1]

        neighbors = graph.get(current\_node.name)

        for key, \_ in neighbors.items():

            neighbor = Node(key, current\_node)

            if(neighbor in closed):

                continue

            neighbor.g = current\_node.g + graph.get(current\_node.name, neighbor.name)

            neighbor.h = heuristics.get(neighbor.name)

            neighbor.f = neighbor.g + neighbor.h

            if(add\_to\_open(open, neighbor) == True):

                open.append(neighbor)

    return None

def add\_to\_open(open, neighbor):

    for node in open:

        if (neighbor == node and neighbor.f > node.f):

            return False

    return True

def main():

    graph = Graph()

    N = int(input("Enter No. of Cities: "))

    for i in range(N):

        x = random.randint(5,N/10)

        for \_ in range(x):

            y = random.randint(0,N-1)

            if(i == y):

                continue

            graph.connect(str(i),str(y),random.randint(10,1000))

    graph.make\_undirected()

    heuristics = {}

    for i in range(N):

        x = random.randint(10,1000)

        heuristics[str(i)] = x

    City1 = int(input("Enter City 1: "))

    City2 = int(input("Enter City 2: "))

    heuristics[str(City2)] = 0

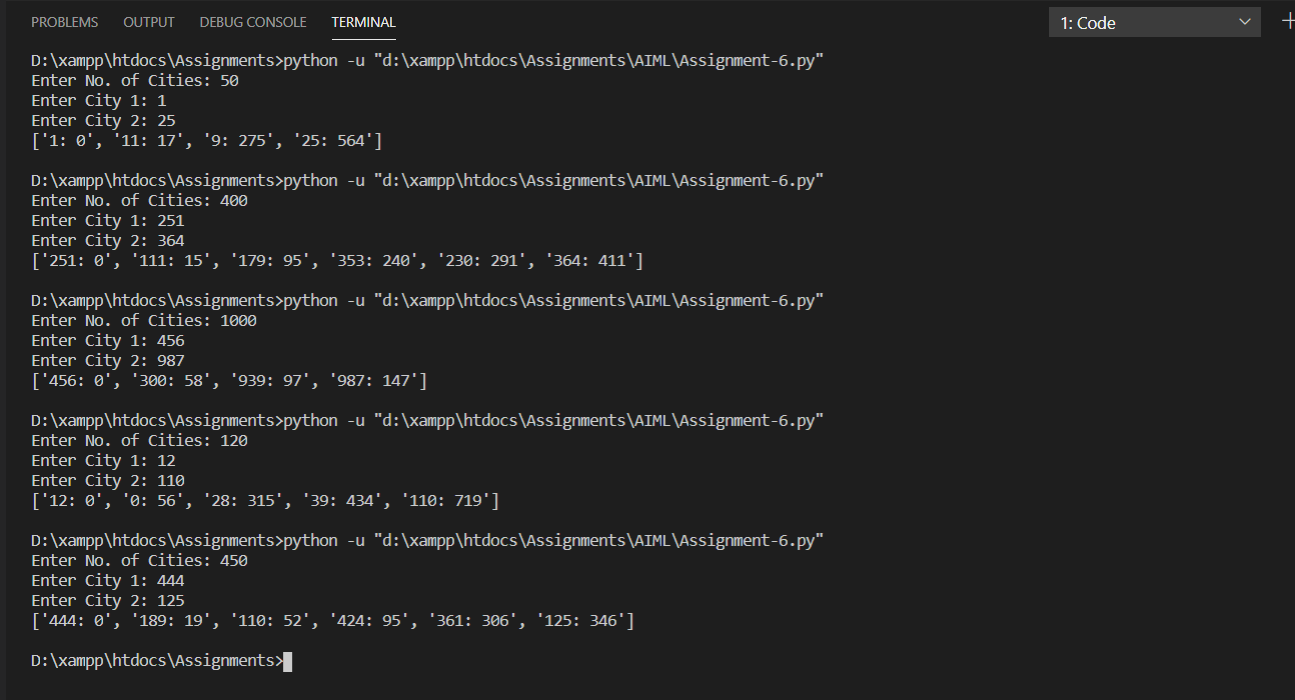
    path = astar\_search(graph, heuristics, str(City1), str(City2))

    print(path)

if \_\_name\_\_ == "\_\_main\_\_":

    main()

**Output**

****