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.n
ame)
            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

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL                                  1: Code

D:\xampp\htdocs\Assignments>python -u "d:\xampp\htdocs\Assignments\AIML\Assignment-6.py"
Enter No. of Cities: 50
Enter City 1: 1
Enter City 2: 25
['1: 0', '11: 17', '9: 275', '25: 564']

D:\xampp\htdocs\Assignments>python -u "d:\xampp\htdocs\Assignments\AIML\Assignment-6.py"
Enter No. of Cities: 400
Enter City 1: 251
Enter City 2: 364
['251: 0', '111: 15', '179: 95', '353: 240', '230: 291', '364: 411']

D:\xampp\htdocs\Assignments>python -u "d:\xampp\htdocs\Assignments\AIML\Assignment-6.py"
Enter No. of Cities: 1000
Enter City 1: 456
Enter City 2: 987
['456: 0', '300: 58', '939: 97', '987: 147']

D:\xampp\htdocs\Assignments>python -u "d:\xampp\htdocs\Assignments\AIML\Assignment-6.py"
Enter No. of Cities: 120
Enter City 1: 12
Enter City 2: 110
['12: 0', '0: 56', '28: 315', '39: 434', '110: 719']

D:\xampp\htdocs\Assignments>python -u "d:\xampp\htdocs\Assignments\AIML\Assignment-6.py"
Enter No. of Cities: 450
Enter City 1: 444
Enter City 2: 125
['444: 0', '189: 19', '110: 52', '424: 95', '361: 306', '125: 346']

D:\xampp\htdocs\Assignments>
```