Bahria University, Islamabad

Department of Software Engineering

Artificial Intelligence Lab

(Fall-2021)

Teacher: Engr. M Waleed Khan

Student : M Iqrar Ijaz Malik

Enrollment : 01-131182-021

Lab Journal: **Open Ended**

Date: 13-12-2021

Task No:	Task Wise Marks		Documentation Marks		Total Marks (20)
	Assigned	Obtained	Assigned	Obtained	
1					

Comments:

Signature

LAB

Open Ended

Tools Used

Tool used to perform this task is **PyCharm Community Addition**

Code:

Refactor Run Tools VCS Window Help OpenEnded - main.py

main.py

```python

class Graph:
    # Initialize the class
    def __init__(self, graph_dict=None, directed=True):
        self.graph_dict = graph_dict or {}
        self.directed = directed
        if not directed:
            self.make_undirected()

    # Create an undirected graph by adding symmetric edges
    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)
```

main.py

```python
        links = self.graph_dict.setdefault(a, {})
        if b is None:
            return links
        else:
            return links.get(b)

    # Return a list of nodes in the graph
    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  # Distance to start node
        self.h = 0  # Distance to goal node
        self.f = 0  # Total cost
```

```python

    # Sort nodes
    def __lt__(self, other):
        return self.f < other.f

    # Print node
    def __repr__(self):
        return '({0},{1})'.format(self.name, self.f)


def astar_search(graph, heuristics, start, end):
    # Create lists for open nodes and closed nodes
    open = []
    closed = []
    # Create a start node and an goal node
    start_node = Node(start, None)
    goal_node = Node(end, None)
    # Add the start node
    open.append(start_node)

```

```python
    # Loop until the open list is empty
    while len(open) > 0:
        open.sort()
        # Get the node with the lowest cost
        current_node = open.pop(0)
        # Add the current node to the closed list
        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 reversed path
            return path[::-1]
            # Get neighbours
        neighbors = graph.get(current_node.name)
            # Loop neighbors
        for key, value in neighbors.items():
                # Create a neighbor node
                neighbor = Node(key, current_node)
                # Check if the neighbor is in the closed list
                if (neighbor in closed):
                    continue
                # Calculate full path cost
                neighbor.g = current_node.g + graph.get(current_node.name, neighbor.name)
                neighbor.h = heuristics.get(neighbor.name)
                neighbor.f = neighbor.g + neighbor.h
                # Check if neighbor is in open list and if it has a lower f value
                if (add_to_open(open, neighbor) == True):
                    open.append(neighbor)

                # Everything is green, add neighbor to open list
            # Return None, no path is found

def add_to_open(open, neighbor):
    for node in open:
        if (neighbor == node and neighbor.f > node.f):
            return False
    return True


def main():
    # Create a graph
    graph = Graph()

    # Create graph connections (Actual distance)

```

```python
    graph.connect('Vertice A', 'Vertice B', 2)
    graph.connect('Vertice A', 'Vertice C', 2)
    graph.connect('Vertice B', 'Vertice C', 10)
    graph.connect('Vertice C', 'Vertice D', 15)
    graph.connect('Vertice D', 'Vertice F', 9)
    # Make graph undirected, create symmetric connections


    graph.make_undirected()
    # Create heuristics (straight-line distance, air-travel distance)

    heuristics = {}
    heuristics['Vertice A'] = 100
    heuristics['Vertice B'] = 200
    heuristics['Vertice C'] = 300
    heuristics['Vertice D'] = 400
    heuristics['Vertice E'] = 500
    heuristics['Vertice F'] = 600

    # Run the search algorithm
    path = astar_search(graph, heuristics, 'Vertice A', 'Vertice F')
    print(path)
    print()
if __name__ == "__main__": main()
```

output:

```
Run:   main
C:\Users\01-131182-021\AppData\Local\Microsoft\WindowsApps\python3.9.exe "F:/Study/Le
['Vertice A: 0', 'Vertice C: 2', 'Vertice D: 17', 'Vertice F: 26']


Process finished with exit code 0
```

Conclusion

I completed the tasks given to us and pasted the output above.