EXPERIMENT NO. 07

Student Name and Roll Number: Piyush Gambhir – 21CSU349	
Semester /Section: Semester-V – AIML-V-B (AL-3)	
Link to Code: NCU-Lab-Manual-And-End-Semester-Projects/NCU-CSL347-AAIES-Lab_Manual at main · Piyush-Gambhir/NCU-Lab-Manual-And-End-Semester-Projects (github.com)	
Date: 23.09.2023	
Faculty Signature:	
Grade:	

Objective(s):

- Understand and study Visualization techniques for graphs.
- Apply logical reasoning over the visualized graph.

Outcome:

Students will be familiarized with Graph Based Visualization and applying logical reasoning over the graph.

Problem Statement:

Implement a Python Code for the following problem:
A logistics company is trying to optimize their delivery routes. They have a dataset of historical delivery data, which includes the start and end points of each delivery, as well as the distance between each point. They want to use graph-based visualization and logical reasoning to identify the most efficient delivery routes.
The dataset is:

Delivery ID	Start Point	End Point	Distance (in miles)
1	Warehouse	Point A	10
2	Point A	Point B	5
3	Point A	Point C	8
4	Point B	Point C	7
5	Point B	Point D	12
6	Point C	Point D	6
7	Point C	Point E	9
8	Point D	Point E	11

Background Study:
Graph-based visualization is a powerful technique for representing and understanding complex relationships and connections among various data elements. It involves creating visual representations of data as nodes (vertices) connected by edges (lines) that indicate the relationships between them. Graphs allow for a clear depiction of patterns, clusters, and dependencies, enabling users to uncover insights that might be less apparent in raw data.

Question Bank:

1. How can you visualize graphs from a given dataset?
   Visualizing Graphs from a Dataset: Graphs can be visualized from a dataset using graph visualization tools or libraries like NetworkX (Python), Gephi, or D3.js. These tools help represent nodes (vertices) and edges, allowing you to visualize relationships and structures present in the data.

2. Which algorithms could have been applied to get identify the efficient delivery routes?
   - Dijkstra's Algorithm: Used for finding the shortest path between nodes in a weighted graph, applicable to identifying efficient routes.
   - A Algorithm*: Combines Dijkstra's with heuristics for optimal pathfinding in graphs, often used in route planning with distance and estimated cost considerations.
   - Traveling Salesman Problem (TSP) Algorithms: Various algorithms exist to solve the TSP, including Genetic Algorithms, Ant Colony Optimization, and Dynamic Programming.
   - Floyd-Warshall Algorithm: Finds shortest paths between all pairs of nodes in a weighted graph, useful for identifying optimal routes in delivery networks.
   - Constrained Shortest Path Algorithms: Incorporates constraints like time windows, capacity, and vehicle availability into route optimization.

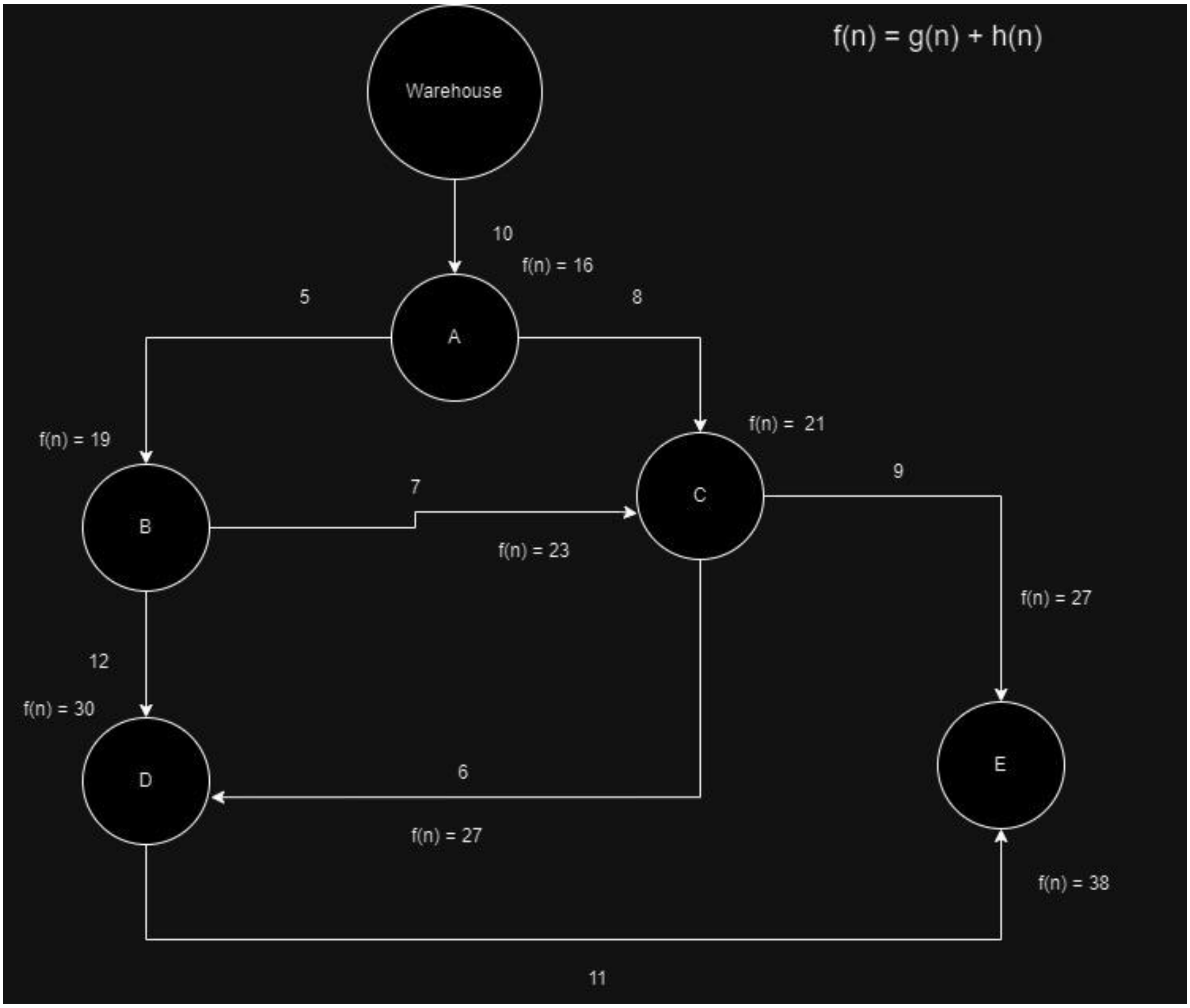
Student Work Area

Algorithm/Flowchart/Code/Sample Outputs

Pseudocode

```
Initialize empty priority queue pq with (0, start, empty list, 0)
Initialize empty set visited

While pq is not empty:
    Pop (priority, current, path, cost) from pq
    If current is visited:
        Continue
    Add current to visited
    Update path by appending current

    If current equals end:
        Return path and cost

    For each neighbor of current:
        If current or end not in heuristic_table:
            Continue
        Get weight from graph for edge (current, neighbor)
        Get heuristic from heuristic_table for (current, end)
        Calculate new_cost = cost + weight
        Calculate new_priority = new_cost + heuristic

        Add (new_priority, neighbor, path, new_cost) to pq

If function reaches this point:
    Return None, None
```

Graph



f(n) = g(n) + h(n)
Warehouse
10
f(n) = 16
5
8
A
f(n) = 19
f(n) = 21
B
7
C
9
f(n) = 23
f(n) = 27
12
f(n) = 30
D
6
E
f(n) = 27
f(n) = 38
11

Delivery ID	Start Point	End Point	Distance (in miles)
1	Warehouse	Point A	10
2	Point A	Point B	5
3	Point A	Point C	8
4	Point B	Point C	7
5	Point B	Point D	12
6	Point C	Point D	6
7	Point C	Point E	9
8	Point D	Point E	11

Heuristic table

	A	B	C	D	E
W	6	5	4	5	6
A	5	4	3	4	5
B	4	3	2	3	4
C	3	2	1	2	3
D	4	3	2	3	4
E	5	4	3	4	5

Shortest Logical Path: Warehouse -> Point A -> Point C -> Point E

Code:

Experiment 7

Problem Statement:

Implement a Python Code for the following problem: A logistics company is trying to optimize their delivery routes. They have a dataset of historical delivery data, which includes the start and end points of each delivery, as well as the distance between each point. They want to use graph-based visualization and logical reasoning to identify the most efficient delivery routes.

Delivery ID	Start Point	End Point	Distance (in miles)
1	Warehouse	Point A	10
2	Point A	Point B	5
3	Point A	Point C	8
4	Point B	Point C	7
5	Point B	Point D	12
6	Point C	Point D	6
7	Point C	Point E	9
8	Point D	Point E	11

Heuristic table

	A	B	C	D	E
W	6	5	4	5	6
A	5	4	3	4	5
B	4	3	2	3	4
C	3	2	1	2	3
D	4	3	2	3	4
E	5	4	3	4	5

Code:

```python
# To plot a networkx graph in pyvis
import numpy as np
import pandas as pd
import networkx as nx
from pyvis.network import Network
import heapq
```

[12] Python

Graph Building using Networkx

```python
# Step 1: Create a graph representation of delivery routes
def create_delivery_graph(data):
    # TODO: Implement this function to create a graph from the given delivery data using NetworkX
    G = nx.DiGraph()
    for _, start, end, distance in data:
        G.add_edge(start, end, weight=distance)
        G.add_edge(end, start, weight=distance)
    return G
```

[13] Python

Graph Visualize using Pyvis

+ Code + Markdown

```python
# Step 2: Visualize the graph using Pyvis
def visualize_graph(graph):
    # TODO: Implement this function to visualize the graph using Pyvis
    # Create an empty PyVis Network object
    net = Network(notebook=True)

    # Add nodes and edges to the PyVis graph
    for node in graph.nodes():
        net.add_node(node, label=node)
    for edge in graph.edges():
        weight = graph[edge[0]][edge[1]]['weight']
        net.add_edge(edge[0], edge[1], title=str(weight), label=str(weight))

    # Save the graph as an HTML file
    net.show("delivery_routes_graph.html")
```

[14] Python

A* Search Algorithm

```python
# Step 3: Implement A* heuristic search algorithm


def a_star_search(graph, start, end, heuristic_table):
    pq = [(0, start, [start], 0)]
    visited = set()

    while pq:
        priority, current, path, cost = heapq.heappop(pq)

        if current in visited:
            continue
        visited.add(current)

        if current == end:
            return path, cost

        for neighbor in graph.neighbors(current):
            if neighbor in visited:
                continue

            weight = graph[current][neighbor]['weight']
            heuristic = 0
            if neighbor in heuristic_table.index:
                heuristic = heuristic_table.loc[neighbor, end]

            new_cost = cost + weight
            new_priority = new_cost + heuristic

            heapq.heappush(pq, (new_priority, neighbor,
                                path + [neighbor], new_cost))

    return None, None
```

[15] Python

Main function to solve the problem

```python
if __name__ == "__main__":
    # Sample dataset
    delivery_data = [
        (1, 'Warehouse', 'Point A', 10),
        (2, 'Point A', 'Point B', 5),
        (3, 'Point A', 'Point C', 8),
        (4, 'Point B', 'Point C', 7),
        (5, 'Point B', 'Point D', 12),
        (6, 'Point C', 'Point D', 6),
        (7, 'Point C', 'Point E', 9),
        (8, 'Point D', 'Point E', 11)
    ]

    # making delivery_data into a pandas dataframe
    delivery_data_df = pd.DataFrame(delivery_data, columns=[
        'id', 'start', 'end', 'distance'])

    # printing the dataframe
    print(delivery_data_df.to_markdown(), end="\n\n")

    # Create the heuristic table
    heuristic_table = pd.DataFrame({
        'Warehouse': [6, 5, 4, 5, 6],
        'Point A': [5, 4, 3, 4, 5],
        'Point B': [4, 3, 2, 3, 4],
        'Point C': [3, 2, 1, 2, 3],
        'Point D': [4, 3, 2, 3, 4],
        'Point E': [5, 4, 3, 4, 5]
    }, index=['Point A', 'Point B', 'Point C', 'Point D', 'Point E'])

    print(heuristic_table.to_markdown(), end="\n\n")

    # Create the delivery graph
    delivery_graph = create_delivery_graph(delivery_data)

    # Visualize the graph
    visualize_graph(delivery_graph)

    # Find the shortest distance using A* heuristic search
    start_point = 'Warehouse'
    end_point = 'Point E'
    shortest_path, shortest_distance = a_star_search(
        delivery_graph, start_point, end_point, heuristic_table)

    if shortest_path:
        print(
            f"Shortest path from {start_point} to {end_point}: {' -> '.join(shortest_path)}")
        print(f"Shortest distance: {shortest_distance} miles")
    else:
        print(f"No path found from {start_point} to {end_point}")
```

[16] Python

Output:

```
...  |    |   id | start     | end     |   distance |
     |---:|-----:|:----------|:--------|-----------:|
     |  0 |    1 | Warehouse | Point A |         10 |
     |  1 |    2 | Point A   | Point B |          5 |
     |  2 |    3 | Point A   | Point C |          8 |
     |  3 |    4 | Point B   | Point C |          7 |
     |  4 |    5 | Point B   | Point D |         12 |
     |  5 |    6 | Point C   | Point D |          6 |
     |  6 |    7 | Point C   | Point E |          9 |
     |  7 |    8 | Point D   | Point E |         11 |

     |         |   Warehouse |   Point A |   Point B |   Point C |   Point D |   Point E |
     |:--------|------------:|----------:|----------:|----------:|----------:|----------:|
     | Point A |           6 |         5 |         4 |         3 |         4 |         5 |
     | Point B |           5 |         4 |         3 |         2 |         3 |         4 |
     | Point C |           4 |         3 |         2 |         1 |         2 |         3 |
     | Point D |           5 |         4 |         3 |         2 |         3 |         4 |
     | Point E |           6 |         5 |         4 |         3 |         4 |         5 |

     Warning: When  cdn_resources is 'local' jupyter notebook has issues displaying graphics on chrome/safari. Use cdn_resources='in_line' or cdn_resources='remote' if you have issues view
     delivery_routes_graph.html
     Shortest path from Warehouse to Point E: Warehouse -> Point A -> Point C -> Point E
     Shortest distance: 27 miles
```