Name – Aryan Sharma

BT21GCS161

SECTION -B3

ASSIGNMENT-09

Question – WAP Bellman-fort Algorithm python program

Code

```
02.py > ...
#  bellman fort algorithm

def bellman_ford(graph, start):
    distances = {node: float('inf') for node in graph}
    distances[start] = 0
    prev = {}  # dictionary to store previous node on shortest path
    for i in range(len(graph) - 1):
        for u in graph:
            for v, weight in graph[u].items():
                if distances[u] != float('inf') and distances[u] + weight < distances[v]:
                    distances[v] = distances[u] + weight
                    prev[v] = u  # update previous node for v
    for u in graph:
        for v, weight in graph[u].items():
            if distances[u] != float('inf') and distances[u] + weight < distances[v]:
                raise ValueError("Negative weight cycle detected")
    return distances, prev


graph = {
    '1': {'2': 6, '3': 5, '4': 5},
    '2': {'5': -1},
    '3': {'2': -2, '5': 1},
    '4': {'3': -2, '6': -1},
    '5': {'7': 3},
    '6': {'7': 3},
    '7': {}
}
```

```
start_node = '1'
distances, prev = bellman_ford(graph, start_node)

# print the shortest path from start_node to all other nodes
for node, dist in distances.items():
    path = []
    curr_node = node
    while curr_node != start_node:
        path.append(curr_node)
        curr_node = prev[curr_node]  # use prev dictionary to find previous node
    path.append(start_node)
    path.reverse()
    print(f"Shortest path from {start_node} to {node}: {' -> '.join(path)}, cost: {dist}")
```

Output-

```
PS C:\Users\aryan\OneDrive - st.niituniversity.in\DAA Assignment\Assignment -9> & C:/Users
Shortest path from 1 to 1: 1, cost: 0
Shortest path from 1 to 2: 1 -> 4 -> 3 -> 2, cost: 1
Shortest path from 1 to 3: 1 -> 4 -> 3, cost: 3
Shortest path from 1 to 4: 1 -> 4, cost: 5
Shortest path from 1 to 5: 1 -> 4 -> 3 -> 2 -> 5, cost: 0
Shortest path from 1 to 6: 1 -> 4 -> 6, cost: 4
Shortest path from 1 to 7: 1 -> 4 -> 3 -> 2 -> 5 -> 7, cost: 3
PS C:\Users\aryan\OneDrive - st.niituniversity.in\DAA Assignment\Assignment -9>
```

Analysis –

#Time complexity:

→Best :- O(V^3) if the graph is complete, or O(V*E) otherwise.
→Average :- O(V*E)
→Worst :- O(E)

#Space Complexity :- O(V)

V is no. of vertices, E is no. of edges in the graph