Department of Computer Engineering
Class: S.Y. B.Tech. Semester: IV
Course Code: DJ19CEL405 Course Name: Computer Networks Lab

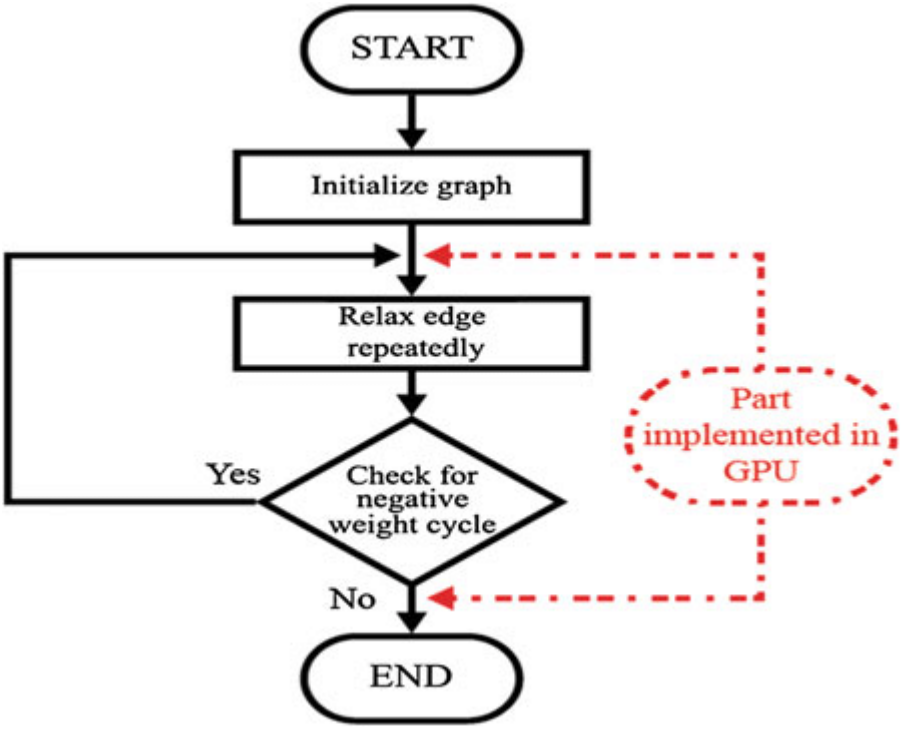
Name: Vinit Shah	SAP ID:60004220097
Date of Performance:04-09-24	Date of Submission:12-09-24

Experiment No: 6

Aim: Write a program to implement Dijikstra's and Bellman Ford Shortest Path Routing Algorithm.

Theory:

Bellman Ford:



Dijikstra's Algorithm:

Algorithm. Modified Dijkstra
Input: a graph G, a source vertex s and a destination vertex t
Output: a path from s to t with the maximum load
1. **for** each vertex v **do**
{ status[v]=0; wt[v]=-1; dad[v]=-1; }
2. status[s]=2; wt[s]=+∞;
3. **for** each edge [s, w] **do**
{ status[w]=1; wt[w]=weight(s,w); dad[w]=s; }
4. **while** there are fringes **do**
v = the fringe with the max wt-value;
status[v]=2;
for each edge[v, w] **do**
case 1. status[w]==0:
{ status[w]=1; wt[w]=min{wt[v], weight(v, w)}; dad[w]=v; }
case 2. (status[w]==1) and (wt[w]<min{wt[v], weight(v, w)}):
{ wt[w]=min{wt[v], weight(v, w)}; dad[w]=v; }

Figure 2. " The pseudo-code of the modified Dijkstra's algorithm

Program:

```
import sys
class Graph:
    def __init__(self, vertices):
        self.V = vertices  # Number of vertices
        self.graph = []  # Default dictionary to store the graph as an edge list
    # Function to add an edge to the graph
    def add_edge(self, u, v, w):
        self.graph.append([u, v, w])

    # Function to print the solution
    def print_solution(self, dist):
        print("Vertex Distance from Source")
        for i in range(self.V):
            print(f"{i}\t\t{dist[i]}")

    # Dijkstra's algorithm for shortest path
    def dijkstra(self, src):
        dist = [float('inf')] * self.V  # Initialize distances as infinite
```

```
    dist[src] = 0
    visited = [False] * self.V

    for _ in range(self.V):
        # Find the vertex with the minimum distance
        min_distance = float('inf')
        min_index = -1
        for v in range(self.V):
            if not visited[v] and dist[v] < min_distance:
                min_distance = dist[v]
                min_index = v

        u = min_index
        visited[u] = True

        # Update the distance of the adjacent vertices of the picked vertex
        for v, w in [(v, w) for u, v, w in self.graph if u == min_index]:
            if not visited[v] and dist[u] != float('inf') and dist[u] + w < dist[v]:
                dist[v] = dist[u] + w

    self.print_solution(dist)

# Bellman-Ford algorithm for shortest path
def bellman_ford(self, src):
    dist = [float('inf')] * self.V
    dist[src] = 0

    # Relax all edges V - 1 times
    for _ in range(self.V - 1):
```

```
        for u, v, w in self.graph:
            if dist[u] != float('inf') and dist[u] + w < dist[v]:
                dist[v] = dist[u] + w

        # Check for negative-weight cycles
        for u, v, w in self.graph:
            if dist[u] != float('inf') and dist[u] + w < dist[v]:
                print("Graph contains negative weight cycle")
                return

        self.print_solution(dist)

# Example usage
g = Graph(5)
g.add_edge(0, 1, 6)
g.add_edge(0, 3, 7)
g.add_edge(1, 2, 5)
g.add_edge(1, 3, 8)
g.add_edge(1, 4, -4)
g.add_edge(2, 1, -2)
g.add_edge(3, 2, -3)
g.add_edge(3, 4, 9)
g.add_edge(4, 0, 2)
g.add_edge(4, 2, 7)

print("Dijkstra's Algorithm:")
g.dijkstra(0)

print("\nBellman-Ford Algorithm:")
```

g.bellman_ford(0)

Screenshots:

Conclusion:

Thus, we have successfully studied and implemented Dijikstra's and Bellman Ford Shortest Path Routing Algorithm.