## Name of Experiment:

Introduction to Bellman-Ford Algorithm

## Objectives:

1. To understand the concept of Bellman-Ford algorithm
2. To understand the implementation of the Algorithm to find the shortest path in weighted graph
3. To detect the negative cycle in the graph
4. To implement the algorithm for solving shortest path problem in real life.
5. To analyze the time and space complexity for Bellman-Ford Algorithm and comparison with Dijkstra algorithm.

## Introduction:

The Bellman-Ford algorithm is a shortest path algorithm used to find the minimum distance from a single source vertex to all other vertex in a weighted graph. Dijkstra algorithm cannot handle negative weighted edges where Bellman-Ford algorithm can handle, making it more versatile in graph problems.

It works on the principle of edge relaxation. Each edge is repeatedly checked t o see that if the current path can be changed or improved by going through another vertex.

The key idea is ,

*For all edges (u, v) with wight w,*

*if distance [v] > distance [u] + w, then update distance [v]= distance [u] + w*

This process is repleted again and again for V- times.

## Algorithm:

```
Step 01: Distance array or list is initialized with infinity and distance from source to source is
Step 02: For V-1 times :
        For each Edge (u,v) with weight w,
        if distance [v] > distance [u] + w , then update distance [v]= distance [u] + w
Step 03: Check for negative cycle by extra one iteration to check if any edge can still be relaxed
Step 04: Print the shortest distance or report if  negative cycle is detected.
```

## Code:

```
#include <bits/stdc++.h>
using namespace std;
#define INF 1e9

void bellmanFord(int V, int E, vector<vector<int>>& edges, int src) {
   vector<int> dist(V, INF);
   dist[src] = 0;

   for (int i = 0; i < V - 1; i++) {
      for (int j = 0; j < E; j++) {
         int u = edges[j][0];
         int v = edges[j][1];
         int w = edges[j][2];

         if (dist[u] != INF && dist[u] + w < dist[v]) {
            dist[v] = dist[u] + w;
         }
      }
   }

   bool hasNegativeCycle = false;
   for (int j = 0; j < E; j++) {
      int u = edges[j][0];
      int v = edges[j][1];
      int w = edges[j][2];

      if (dist[u] != INF && dist[u] + w < dist[v]) {
         hasNegativeCycle = true;
         break;
      }
   }

   if (hasNegativeCycle) {
      cout << "Graph contains negative weight cycle!" << endl;
   } else {
      cout << "\nShortest distances from source " << src << ":\n";
```

```
        for (int i = 0; i < V; i++) {
            cout << "Vertex " << i << ": ";
            if (dist[i] == INF) {
                cout << "INF" << endl;
            } else {
                cout << dist[i] << endl;
            }
        }
    }
}
int main() {
    int V, E;
    cin >> V >> E;
    vector<vector<int>> edges;

    for (int i = 0; i < E; i++) {
        int u, v, w;
        cin >> u >> v >> w;
        edges.push_back({u, v, w});
    }
    int src;
    cin >> src;
    bellmanFord(V, E, edges, src);

    return 0;
}
```

## Time Complexity:

O( V * E)
where V= No of Vertices , E = No of Edges

## Space Complexity:

O( V )
where V= No of Vertices

## Conclusion:

In this lab , we have implemented the Bellman-Ford Algorithm using CPP language , to find the shortest path from a single source to all other vertices . It effectively calculated the shortest path even if the graph contained negative edges and cycles . It detected negative cycle through one extra iteration . Though Dijkstra Algorithm cannot be applicable while negative edge is available in graph , but Bellman-ford can easily detect it effectively.

However, it is slower than Dijkstra algorithm as in dense graph we need to perform V-1 times full iteration .

## Reference :

1. Cormen, Leiserson, Rivest, and Stein, *Introduction to Algorithms*, MIT Press.
2. GeeksforGeeks: https://www.geeksforgeeks.org/bellman-ford-algorithm-dp-23/
3. Programiz: https://www.programiz.com/dsa/bellman-ford-algorithm
4. Class Lecture