Name of Experiment:

Introduction to Floyd-Warshall and Johnson Algorithm

Objectives:

1. To understand the concept of Floyd warshall and Johnson algorithm.
2. To understand the implementation of the Algorithms to find the shortest path between all pairs of vector in weighted graph
3. To handle the graph with negative edge .
4. To implement the algorithm for solving shortest path problem in real life.
5. To analyze the time and space complexity for Floyd-Warshall and Johnson Algorithm and compare their efficiency.

Introduction:

Floyd-Warshall Algorithm is a dynamic programming based algorithm to find all pair shortest paths in a weighted graph. It can handle the graph that contain negative edges but it can not handle negative cycles. It continuously updates the distance matrix by checking if a vertex (k) can act as intermediate node to lessen the path between two vertices I and j .

dist[i][j[]= min((dist[i][j],dist[i][k]+dist[k][j])

This process is repeated k times for all nodes and finds all pair shortest path. This algorithm is efficient for dense graph .

On the other hand ,the Johnson algorithm is more efficient for sparse graph . It is the combination of both Bellman Ford and Dijkstra Algorithm . It modifies all edges to eliminate the negative weights using Bellman Ford algorithm and it uses Dijksta algorithm on the new edited graph for finding the shortest path between the nodes. The result for both the algorithm is same but Jhonson Algorithm works efficiently for sparse graph .

Reweighting edges is done by using the following equation:

For an edge (u,v), the weight, w'(u,v) = w(u,v)+h[u]-h[v]

This equation ensures that all paths between u and v are increased by same amount and all negative weights have become non-negative.

Algorithm: (Floyd-Warshall)

```
Let dist[][] be a 2D array of size VxV, where V is the number of vertices
Step 01. initialize dist[][] such that:
                if i == j, dist[i][j] = 0
                if there is an edge from vertex i to vertex j, dist[i][j] = weight of that edge
                if there is no edge from vertex i to vertex j, dist[i][j] = ∞ (infinity)
Step 02. for each intermediate vertex k from 0 to V-1:
Step 03.        for each vertex i from 0 to V-1:
Step 04.            for each vertex j from 0 to V-1:
Step 05.                if dist[i][j] > dist[i][k] + dist[k][j]:
Step 06.                dist[i][j] = dist[i][k] + dist[k][j]
The dist[][] matrix now contains the shortest distances between all pairs of vertices.
```

Algorithm: (Johnson)

```
Step 01: Add a new vertex q connected to every other vertex with edge weight 0.
Step 02: Run Bellman-Ford from q to calculate potential values h(v) for each vertex.
Step 03: Reweight all edges using formula:
                w'(u, v) = w(u, v) + h(u) - h(v)
         so all edges become non-negative.
Step 04: Run Dijkstra's algorithm from each vertex using reweighted edges.
Step 05: Adjust the final distances back using:
                dist(u, v) = dist'(u, v) - h(u) + h(v)
```

Code:

<u>Floyd-Warshall Algorithm :</u>

```
#include<bits/stdc++.h>
……………………………………using namespace std;
#define INF 100000000

void floyd(vector<vector<int>> &dist){
   int v= dist.size();
   for(int k=0; k<v; k++){
        for(int i=0; i<v; i++){
            for(int j=0; j<v; j++){
                dist[i][j]=min(dist[i][j],dist[i][k]+dist[k][j]);
            }
        }
    }

}
int main(){
    vector<vector<int>> dist={
        {0, 4, INF, 5, INF},
        {INF, 0, 1, INF, 6},
        {2, INF, 0, 3, INF},
        {INF, INF, 1, 0, 2},
        {1, INF, INF, 4, 0}
    };

    floyd(dist);
    for(int i = 0; i<dist.size(); i++) {
        for(int j = 0; j<dist.size(); j++) {
            cout<<dist[i][j]<<" ";
        }
        cout<<endl;
    }
}
```

Johnson's Algorithm :

```
#include <iostream>
#include <vector>
#include <queue>
using namespace std;

const int INF = 1000000000;

bool bellmanFord(int n, vector<vector<int>>& edges, vector<int>& h, int src) {
    h.assign(n, INF);
    h[src] = 0;

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

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

    for (int j = 0; j < edges.size(); j++) {
        int u = edges[j][0];
        int v = edges[j][1];
        int w = edges[j][2];

        if (h[u] != INF && h[u] + w < h[v]) {
            return false;
        }
    }
    return true;
}

void dijkstra(int n, vector<pair<int, int>> adj[], vector<int>& dist, int src) {
    priority_queue<pair<int, int>, vector<pair<int, int>>, greater<pair<int, int>>> pq;

    dist.assign(n, INF);
    dist[src] = 0;
    pq.push({0, src});

    while (!pq.empty()) {
        int d = pq.top().first;
        int u = pq.top().second;
        pq.pop();

        if (d > dist[u]) continue;

        for (int i = 0; i < adj[u].size(); i++) {
            int v = adj[u][i].first;
            int w = adj[u][i].second;

            if (dist[u] + w < dist[v]) {
                dist[v] = dist[u] + w;
                pq.push({dist[v], v});
            }
        }
    }
}

void johnsonAlgo(int n, vector<vector<int>>& edges) {


    vector<vector<int>> newEdges = edges;
    for (int i = 0; i < n; i++) {
        newEdges.push_back({n, i, 0});
    }

    vector<int> h;
    if (!bellmanFord(n + 1, newEdges, h, n)) {
        cout << "Graph contains negative weight cycle!\n";
        return;
    }

    vector<pair<int, int>> adj[n];
    for (int i = 0; i < edges.size(); i++) {
        int u = edges[i][0];
        int v = edges[i][1];
        int w = edges[i][2];

        int newW = w + h[u] - h[v];
        adj[u].push_back({v, newW});
    }

    vector<vector<int>> dist(n, vector<int>(n));

    for (int src = 0; src < n; src++) {
        vector<int> d;
        dijkstra(n, adj, d, src);

        for (int dst = 0; dst < n; dst++) {
            if (d[dst] == INF) {
                dist[src][dst] = INF;
            } else {
                dist[src][dst] = d[dst] - h[src] + h[dst];
            }
        }
    }

    for (int i = 0; i < n; i++) {
        cout << "From vertex " << i << ": ";
        for (int j = 0; j < n; j++) {
            if (dist[i][j] == INF) {
                cout << "INF ";
            } else {
                cout << dist[i][j] << " ";
            }
        }
        cout << endl;
    }
}

int main() {

    int n = 4;

    vector<vector<int>> edges = {
        {0, 1, -5},
        {0, 2, 2},
        {0, 3, 3},
        {1, 2, 4},
        {2, 3, 1}
    };

    johnsonAlgo(n, edges);

    return 0;
}
```

Time Complexity:

Floyd-Warshall Algorithm :	$O(V^3)$
Johnson Algorithm:	$O(VE+V^2 \log V)$ (faster for sparse graphs)

Space Complexity:

Floyd-Warshall Algorithm :	$O(V^2)$
Johnson Algorithm:	$O(V+E)$

Conclusion:

In this lab , we have implemented the Floyd-Warshall and Johnson Algorithm to find the shortest path between all pair of vertices. Floyd Warshal Algorithm is simple and efficient for dense graph where Johnson Algorithm is efficient for sparse graph . Both algorithm can handle the negative edge weight, but both cannot handle negative cycle . So through the analysis and comparison of these two algorithm , the efficiency depends on the graph density .

Reference :

1. Cormen, Leiserson, Rivest, and Stein, *Introduction to Algorithms*, MIT Press.
2. GeeksforGeeks: https://www.geeksforgeeks.org/floyd-warshall-algorithm-dp-16/
3. GeeksforGeeks: https://www.geeksforgeeks.org/johnsons-algorithm/
4. Programiz: https://www.programiz.com/dsa/floyd-warshall-algorithm
5. Class Lecture