**Analysis of Algorithm**

**Practical no 8 : Bellman Ford Algorithm**

**Code :**

import java.util.Arrays;

class bFord2 {

    static int[] bellmanFord(int V, int[][] edges, int src) {

        int[] dist = new int[V];

        Arrays.fill(dist, Integer.MAX\_VALUE);

        dist[src] = 0;

        for (int i = 0; i < V; i++) {

            for (int[] edge : edges) {

                int u = edge[0];

                int v = edge[1];

                int wt = edge[2];

                if (dist[u] != Integer.MAX\_VALUE && dist[u] + wt < dist[v]) {

                    if (i == V - 1)

                        return new int[]{-1};

                    dist[v] = dist[u] + wt;

                }

            }

        }

        return dist;

    }

    public static void main(String[] args) {

        int V = 5;

        int[][] edges = new int[][] {

            {1, 3, 2},

            {4, 3, -1},

            {2, 4, 1},

            {1, 2, 1},

            {0, 1, 5}

        };

        int src = 0;

        int[] ans = bellmanFord(V, edges, src);

        System.out.println("Vertex Distance from source : ");

        for(int i = 0; i < V; i++) {

            if(ans[i] == Integer.MAX\_VALUE)

                System.out.println(i + " : INF");

            else

                System.out.println(i + " : " + ans[i]);

        }

    }

}

**Output :**

A screen shot of a computer

AI-generated content may be incorrect.

**Analysis :**

**1. Function: bellmanFord**

* **Inputs**:
  + V: The number of vertices in the graph.
  + edges: A 2D array representing the edges of the graph, where each edge is defined by a 3-element array [u, v, wt] indicating an edge from vertex u to vertex v with weight wt.
  + src: The source vertex from which the shortest paths will be computed.
* **Outputs**:
  + An array dist[] that holds the shortest distance from the source to every vertex. If a vertex is unreachable, its distance will be set to Integer.MAX\_VALUE.

**Steps**:

* **Step 1: Initialization**:
  + The dist[] array is initialized with Integer.MAX\_VALUE to represent "infinity," meaning initially all vertices are unreachable from the source except the source itself, which has a distance of 0.
* **Step 2: Relaxation**:
  + The main part of the algorithm involves relaxing all edges V - 1 times. During each relaxation, for each edge (u, v, wt), if the current distance to vertex u is not infinity and the distance through u is shorter than the current distance to vertex v, we update the distance to vertex v.
* **Step 3: Negative Weight Cycle Detection**:
  + After V - 1 relaxations, the algorithm performs an additional relaxation. If any distance can still be reduced, it indicates the presence of a negative weight cycle, and the function returns {-1} to indicate the cycle.

**2. Function: main**

* **Graph Representation**:
  + A graph with 5 vertices and 5 edges is represented by the 2D array edges. Each row defines an edge from vertex u to vertex v with a given weight wt.
* **Calling the Bellman-Ford Algorithm**:
  + The algorithm is executed starting from vertex 0 and the resulting shortest distances are printed.
* **Output**:
  + The distances from the source vertex (src = 0) to all other vertices are displayed. If a vertex is unreachable, it will be displayed as "INF".

**Summary of Time and Space Complexities**

* **Time Complexity**:
  + **Best Case**: O(V \* E)
  + **Average Case**: O(V \* E)
  + **Worst Case**: O(V \* E)
* **Space Complexity**:
  + **Best Case**: O(V + E)
  + **Average Case**: O(V + E)
  + **Worst Case**: O(V + E)