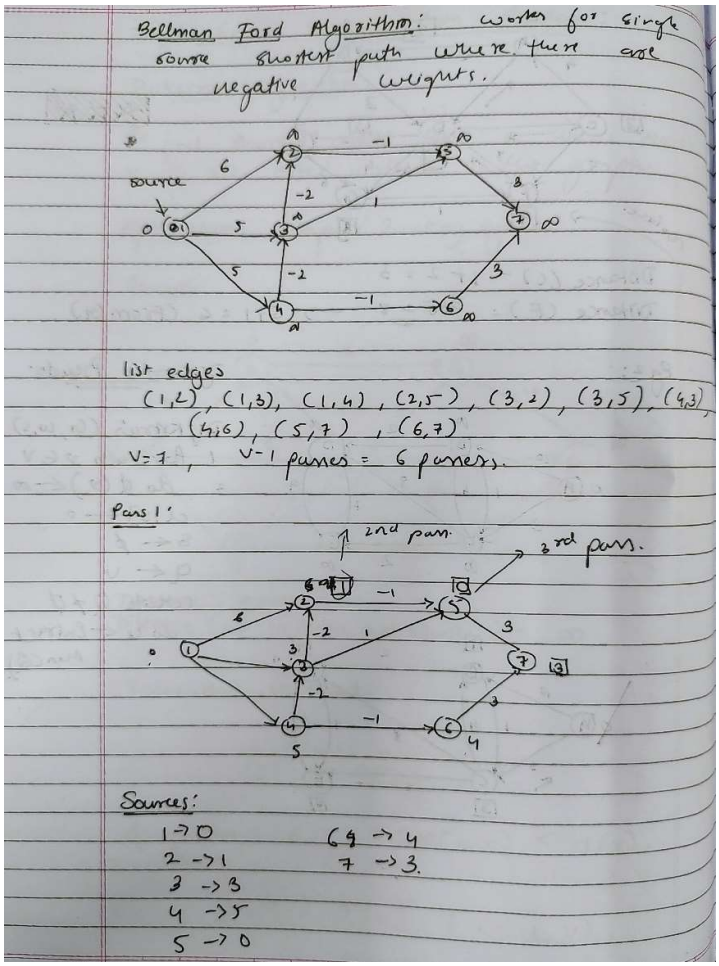
Bharatiya Vidya Bhavan's
SARDAR PATEL INSTITUTE OF TECHNOLOGY
(Autonomous Institute Affiliated to University of Mumbai)
Munshi Nagar, Andheri (W), Mumbai – 400 058.
Department of Master of Computer Application

Experiment	4
Aim	To understand and implement Single Source Shortest path
Objective	1) Write Pseudocode for given problems and understanding the implementation of Single source shortest path 2) Implementing single source shortest paths. 3) Calculating time complexity of the given problems
Name	Vivek Tiwari
UCID	2023510059
Class	FY MCA
Batch	C
Date of Submission	18-03-2024

Algorithm and Explanation of the technique used	***Algorithm:*** 1. Begin by setting the distances from the source vertex to all other vertices as infinite, and set the distance to the source vertex itself as 0. 2. Iterate through all edges V-1 times. For each edge u-v, if the distanceto v through u is shorter than the currently known distance to v, update the distance to v. 3. If updates to the distances persist after V-1 iterations, it indicates the presence of a negative weight cycle in the graph. Conversely, if no negative weight cycles are found, the distances obtained after V-1 iterations represent the shortest paths from the source vertex to all othervertices.

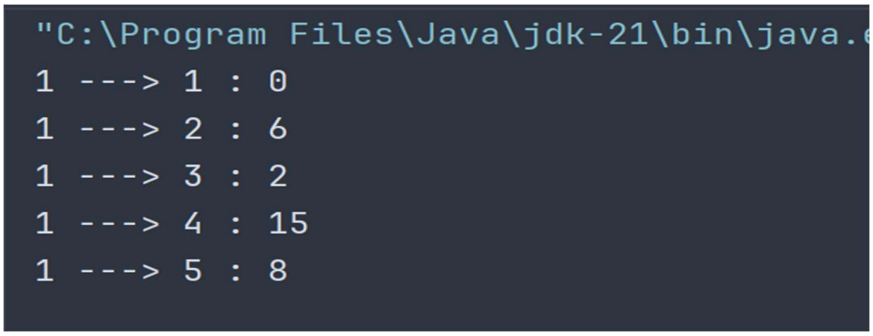


Program(Code)

```
class bellmanFord {
    int V;
    public bellmanFord(int V) {
        this.V = V;
    }
    public void shortestPath(int src, int[][] graph, int[]
dist) {
        for (int i = 0; i < V; i++) {
            dist[i] = Integer.MAX_VALUE;
        }
        dist[src] = 0;
        for (int i = 0; i < V - 1; i++) {
            for (int j = 0; j < V; j++) {
                for (int k = 0; k < V; k++) {
                    if (graph[j][k] != 0 && dist[j] !=
                            Integer.MAX_VALUE && dist[j] +
graph[j][k] < dist[k]) {
                        dist[k] = dist[j] + graph[j][k];
                    }
                }
            }
        }
    }
    public static void main(String[] args) {
        int V = 5; // Change the number of vertices
        bellmanFord obj = new bellmanFord(V);
```

```
        int[][] graph = {
                {0, 6, 2, 0, 0, 0},
                {0, 0, 4, 9, 0, 0},
                {0, 0, 0, 0, 6, 0},
                {0, 0, 0, 0, 0, 8},
                {0, 0, 0, 0, 0, 3}
        };
        int[] dist = new int[V];
        int src = 0;
        obj.shortestPath(src, graph, dist);
        for (int i = 0; i < V; i++) {
            System.out.println((src + 1) + " ---> " + (i +
1) + " : " +
                    dist[i]);
        }
    }
}
```

Output



Justification of the complexity calculated

The Bellman-Ford algorithm serves to determine the shortest pathsfrom a designated source vertex to all other vertices within a weighted graph. The time complexity of this algorithm is justified through the following analysis:

1. Initializing all distances in the `dist` array to `Integer.MAX_VALUE` requires O(V) time, where V signifies thenumber of vertices.

2. The outer loop executes V - 1 times, encompassing all vertices inthe graph. Simultaneously, the middle loop iterates V times, traversing all vertices, while the inner loop iterates V times, addressing all vertices. Consequently, this nested loop structure contributes to an overall time complexity of O(V^3).

3. Within the innermost loop, the algorithm assesses whether a shorter path exists from vertex j to vertex k through vertex i. Thisevaluation is achieved in constant time, O(1).

In essence, the time complexity of the Bellman-Ford algorithm in this scenario amounts to O(V^3), with V representing the number of vertices in the graph.

Conclusion	Advantages: 1. Manages Negative Weight Edges: In contrast to Dijkstra's algorithm,which cannot handle negative weight edges, Bellman-Ford is adept at navigating graphs containing such edges, provided they don't form negative cycles. 2. Identifies Negative Cycles: Bellman-Ford possesses the capability to detect negative cycles within a graph. The persistence of distance updatesduring the V-1 iterations signals the presence of a negative cycle. 3. Versatility: Bellman-Ford offers greater flexibility compared to Dijkstra's algorithm, particularly in scenarios involving fluctuating edgeweights or graphs featuring negative edge weights. Applications: 1. Routing Protocols: Bellman-Ford is integral to routing protocols likeRIP (Routing Information Protocol) for determining optimal routes in computer networks. 2. Network Analysis: Bellman-Ford is instrumental in network analysistools for determining the shortest paths between nodes within intricate networks. Traffic Engineering: Bellman-Ford finds utility in traffic engineering efforts aimed at streamlining traffic flow and mitigating congestion within transportation networks