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	Durgesh Dilip Mandge
UCID	2023510032
Class	FYMCA
Batch	B
Date of Submission	18-03-2024

Algorithm and Explanation of the technique used

Algorithm

1> Statement: Go and repeatedly relax all edge (n-1) times where n is no. of vertices

Relaxation: if (d[u] + c(u,v) < d[v]

d[v] = d[u] + c(u,v)

2> Steps:

i> Choose the initial vertex

ii> Assign '0' distance to it '∞' to rest edges

iii> Apply relaxation (n-1) times to every vertex.

3> Pseudocode:-

```
Input: Graph (V, E), source vertex: s
Initialize:
    For each vertex v in V:
        dist [v] = Infinity
    dist [s] = 0
For i = 1 to |V|-1 :
    For each edge (u,v) in E:
        if dist [u] + weight [u,v) < dist [v]:
            dist [v] = dist [u] + weight (u,v)
For each edge (u,v) in E :
    if dist [u] + weight (u,v) < dist [v] :
        return "Negative cycle detected"
OUTPUT: Array dist []
```

Time Complexity:
The Bellman-Ford algorithm has a time complexity of **O(|V|*|E|),**
where |V| is the number of vertices
and |E| is the number of edges in the weighted graph.

Program(Code)

```java
import java.util.ArrayList;
import java.util.List;
```

```java
public class BellmanFord {
    static class Edge {
        int src, dest, weight;

        Edge(int src, int dest, int weight) {
            this.src = src;
            this.dest = dest;
            this.weight = weight;
        }
    }

    static void bellmanFord(List<Edge> edges, int V, int src) {
        int[] dist = new int[V];
        for (int i = 0; i < V; i++) {
            dist[i] = Integer.MAX_VALUE;
        }
        dist[src] = 0;

        for (int i = 1; i < V; i++) {
            for (Edge edge : edges) {
                if (dist[edge.src] != Integer.MAX_VALUE &&
                    dist[edge.src] + edge.weight < dist[edge.dest]) {
                    dist[edge.dest] = dist[edge.src] + edge.weight;
                }
            }
        }

        for (Edge edge : edges) {
            if (dist[edge.src] != Integer.MAX_VALUE &&
                dist[edge.src] + edge.weight < dist[edge.dest]) {
                System.out.println("Negative cycle detected");
                return;
            }
        }

        for (int i = 0; i < V; i++) {
            System.out.println("Distance from " + src + " to " + i + " is " +
dist[i]);
        }
    }

    public static void main(String[] args) {
        int V = 8;
        List<Edge> edges = new ArrayList<>();
        edges.add(new Edge(1, 2, 6));
        edges.add(new Edge(1, 3, 5));
        edges.add(new Edge(1, 4, 5));
        edges.add(new Edge(2, 5, -1));
```

<table>
<tr><td></td><td>   edges.add(new Edge(3, 2, -2));
        edges.add(new Edge(3, 5, 1));
        edges.add(new Edge(4, 3, -2));
        edges.add(new Edge(4, 6, -1));
        edges.add(new Edge(5, 7, 3));
        edges.add(new Edge(6, 7, 3));

        bellmanFord(edges, V, 1);
    }
}</td></tr>
<tr><td>Output</td><td>va.exe' '--enable-preview' '-XX:+ShowCodeDetailsInExceptionMessages'
torage\21e4554268b67b9d646dfc8274bffa51\redhat.java\jdt_ws\Thread_f0b
Distance from 1 to 0 is 2147483647
Distance from 1 to 1 is 0
Distance from 1 to 2 is 1
Distance from 1 to 3 is 3
Distance from 1 to 4 is 5
Distance from 1 to 5 is 0
Distance from 1 to 6 is 4
Distance from 1 to 7 is 3
PS C:\Users\smart\Documents\JAVA\Thread></td></tr>
<tr><td>Justification of the complexity calculated</td><td>Here's why a time complexity of <u>O(|V|*|E|),</u>:
The outer loop runs |V|-1 times, which is O(|V|).
Inside the outer loop, we have another loop that iterates through all the edges |E|.
For each edge (u, v), we perform the relaxation step, which takes constant time O(1).
Therefore, the total time complexity is:
O(|V|) * O(|E|) * O(1) = O(|V|*|E|)
This makes the Bellman-Ford algorithm quite efficient for sparse graphs,
where the number of edges |E| is much less than the square of the number of vertices |V|^2.
However, for dense graphs, where |E| is close to |V|^2, the time complexity becomes O(|V|^3),
which is less efficient than other algorithms like Dijkstra's algorithm
(which has a time complexity of O(|V|^2 * log|V|) with a Fibonacci heap).</td></tr>
<tr><td>Conclusion</td><td>Algorithm for finding the shortest paths from a single source vertex to all other vertices in a weig
graph.
It is capable of handling graphs with negative edge weights, which is a significant advantage over
other shortest path algorithms like Dijkstra's algorithm.
The algorithm works by repeatedly relaxing the edges of the graph, updating the tentative distanc
vertices from the source vertex until the shortest paths are found.
It does this by iterating over all edges |V|-1 times, where |V| is the number of vertices in the graph
This is because the shortest path between any two vertices can have at most |V|-1 edges in a graph
without negative cycles.</td></tr>
</table>