BRACT’s

Vishwakarma Institute of Information Technology, Pune

**Practical Implementation Sheet**

| **Department:** IT | **Semester:** IV | **Academic Year:** 2024-25 | **Practical No: 6** |
| --- | --- | --- | --- |
| **Class/ Division/ Batch: SY (B)- B3** | | **Roll no: 70** | |
| **Course:** Data Structures and Analysis of Algorithms | | **Name of Student**: Anushka Kadam | |

**Aim: a)** Use the map of the area around the college as the graph. Identify the prominent landmarks as nodes and find minimum distance to various landmarks from the college as the source. Represent this graph using an adjacency matrix. Find the

shortest path using Dijkstra’s algorithm.

**b)** Write a program to implement the Bellman-Ford algorithm to find the shortest

path from a single source to all other nodes in a graph with negative edge weights.

Verify its results for a sample graph and compare it with Dijkstra’s algorithm.

**Code: a)Shortest Path using Dijkstra’s Algorithm**

#include <iostream>

#include <vector>

#include <climits>

using namespace std;

#define V 5

int minDistance(const vector<int>& dist, const vector<bool>& visited) {

int min = INT\_MAX, min\_index;

for (int v = 0; v < V; v++)

if (!visited[v] && dist[v] <= min)

{

min = dist[v];

min\_index = v;

}

return min\_index;

}

void dijkstra(int graph[V][V], int src)

{

vector<int> dist(V, INT\_MAX);

vector<bool> visited(V, false);

dist[src] = 0;

for (int count = 0; count < V - 1; count++)

{

int u = minDistance(dist, visited);

visited[u] = true;

for (int v = 0; v < V; v++)

if (!visited[v] && graph[u][v] && dist[u] != INT\_MAX && dist[u] + graph[u][v] < dist[v])

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

}

cout << "Shortest distances from the college (node " << src << ") to other landmarks:\n";

for (int i = 0; i < V; i++)

cout << "Node " << i << " -> Distance: " << dist[i] << endl;

}

int main()

{

int graph[V][V] =

{

{0, 10, 0, 5, 0},

{10, 0, 1, 2, 0},

{0, 1, 0, 9, 3},

{5, 2, 9, 0, 2},

{0, 0, 3, 2, 0}

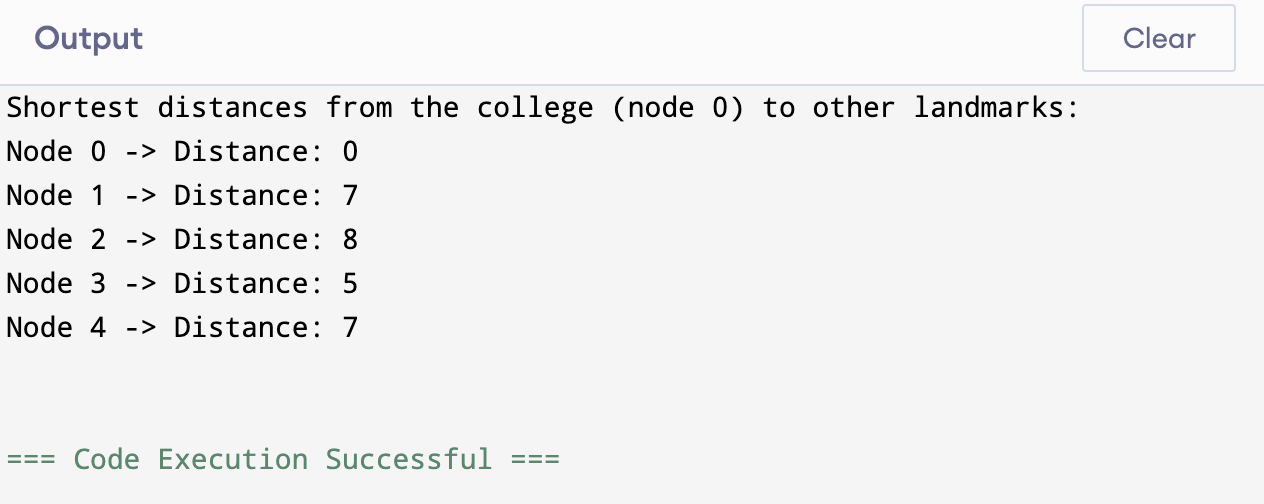
};

dijkstra(graph, 0);

return 0;

}

**Output: a) Shortest Path using Dijkstra’s Algorithm**

****

**Code: b) Shortest Path using Bellman-Ford Algorithm**

#include <iostream>

#include <vector>

#include <climits>

using namespace std;

struct Edge

{

int src, dest, weight;

};

void bellmanFord(vector<Edge>& edges, int V, int src)

{

vector<int> dist(V, INT\_MAX);

dist[src] = 0;

for (int i = 0; i < V - 1; i++)

{

for (const auto& edge : edges)

{

if (dist[edge.src] != INT\_MAX && dist[edge.src] + edge.weight < dist[edge.dest])

dist[edge.dest] = dist[edge.src] + edge.weight;

}

}

for (const auto& edge : edges)

{

if (dist[edge.src] != INT\_MAX && dist[edge.src] + edge.weight < dist[edge.dest])

{

cout << "Graph contains a negative weight cycle!\n";

return;

}

}

cout << "\nShortest distances from node " << src << " using Bellman-Ford Algorithm:\n";

for (int i = 0; i < V; i++)

cout << "Node " << i << " -> Distance: " << dist[i] << endl;

}

int main()

{

int V = 5;

int E = 8;

vector<Edge> edges =

{

{0, 1, -1}, {0, 2, 4},

{1, 2, 3}, {1, 3, 2}, {1, 4, 2},

{3, 2, 5}, {3, 1, 1}, {4, 3, -3}

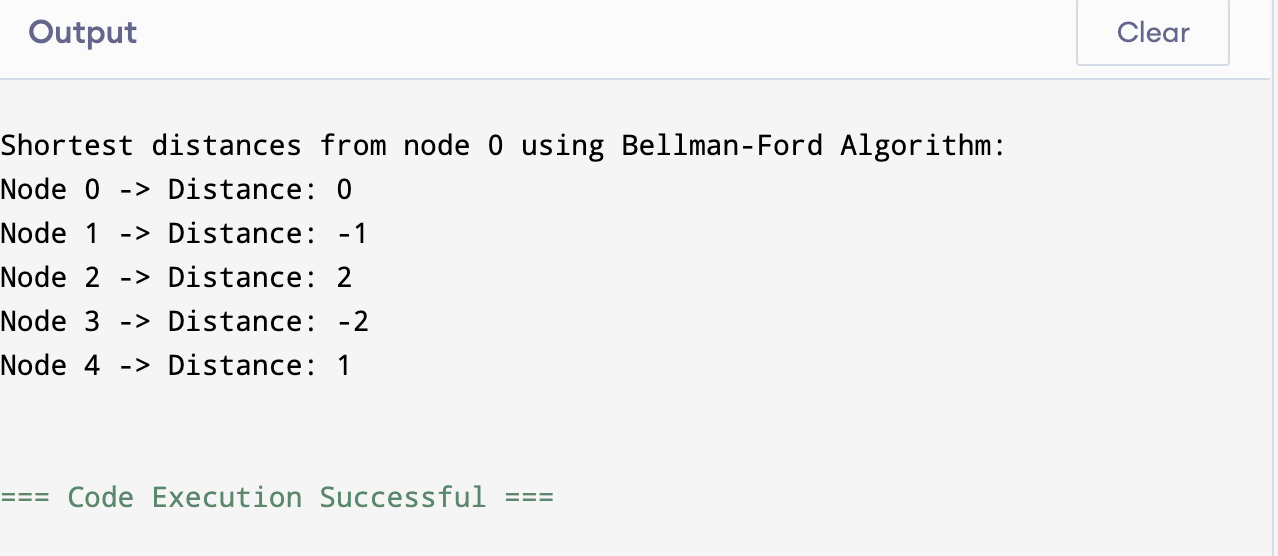
};

bellmanFord(edges, V, 0);

return 0;

}

**Output: b) Shortest Path using Bellman-Ford Algorithm**

****