Code and analyze to find shortest paths in a graph with arbitrary edge weights using Bellman-Ford algorithm.

#include <stdio.h>

#include <limits.h>

#define MAX\_VERTICES 100

#define INF INT\_MAX

typedef struct {

int u, v, weight;

} Edge;

void BellmanFord(int vertices, int edges, Edge edgeList[], int source) {

int dist[vertices];

for (int i = 0; i < vertices; i++) dist[i] = INF;

dist[source] = 0;

// Relax edges V-1 times

for (int i = 1; i <= vertices - 1; i++) {

for (int j = 0; j < edges; j++) {

int u = edgeList[j].u, v = edgeList[j].v, weight = edgeList[j].weight;

if (dist[u] != INF && dist[u] + weight < dist[v]) dist[v] = dist[u] + weight;

}

}

// Check for negative-weight cycles

for (int j = 0; j < edges; j++) {

int u = edgeList[j].u, v = edgeList[j].v, weight = edgeList[j].weight;

if (dist[u] != INF && dist[u] + weight < dist[v]) {

printf("Graph contains negative weight cycle\n");

return;

}

}

// Print shortest distances

for (int i = 0; i < vertices; i++) {

if (dist[i] == INF) printf("%d: INF\n", i);

else printf("%d: %d\n", i, dist[i]);

}

}

int main() {

int vertices = 5, edges = 8;

Edge edgeList[] = {

{0, 1, -1}, {0, 2, 4}, {1, 2, 3}, {1, 3, 2},

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

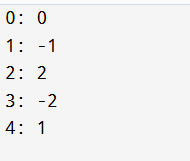
};

int source = 0;

BellmanFord(vertices, edges, edgeList, source);

return 0;

}



**Aim:**

The aim of the **Bellman-Ford algorithm** is to compute the **shortest path** from a **single source vertex** to all other vertices in a **weighted graph**, which can have edges with **negative weights**. It also serves the additional purpose of detecting **negative-weight cycles** in the graph.

**Objectives:**

1. **Find the shortest path**: Compute the shortest path from a given source vertex to all other vertices in the graph.
2. **Handle negative edge weights**: Unlike Dijkstra's algorithm, which doesn't handle negative edge weights, Bellman-Ford can handle them and correctly compute the shortest paths.
3. **Detect negative-weight cycles**: The algorithm can detect if there is a **negative-weight cycle** in the graph, which would make shortest paths undefined or infinitely negative.

**Procedure:**

1. **Initialize distances**:
   * Start by setting the distance from the source vertex to **0** (since the distance from the source to itself is 0).
   * Set the distance to all other vertices as **infinity** (INF), meaning they are initially unreachable.
2. **Relax edges (V - 1 times)**:
   * Perform **V-1 iterations** where **V** is the number of vertices in the graph. In each iteration, for each edge (u, v) with weight w, check if the current known distance to vertex v can be shortened by taking the edge (u, v). If so, update the distance of vertex v.
   * The relaxation condition is:

markdown

Copy

if (dist[u] + weight < dist[v])

dist[v] = dist[u] + weight

1. **Check for negative-weight cycles**:
   * After **V-1** iterations, check each edge one more time. If any edge can still be relaxed, it means there is a **negative-weight cycle** in the graph.
   * If such an edge is found, the graph contains a negative-weight cycle, and the algorithm should report it.
2. **Output the shortest distances**:
   * After all relaxations, output the shortest distances from the source vertex to all other vertices. If a vertex is unreachable, its distance will remain **infinity**.
3. **Result**:
   * The Bellman-Ford algorithm ensures the correct shortest paths even with negative edge weights and detects the presence of negative weight cycles. If no cycle is found, the distances printed are the shortest distances from the source vertex to each other vertex in the graph.

**Conclusion:**

* **Bellman-Ford** is an important algorithm for finding the shortest paths in graphs, especially when there are edges with negative weights.
* It is capable of detecting negative weight cycles, which makes it suitable for applications where such cycles need to be identified, such as in financial systems or in modeling certain types of physical systems.
* Although its time complexity is **O(V \* E)**, making it slower than other algorithms like Dijkstra's algorithm for graphs with non-negative weights, its ability to handle negative edge weights and detect negative cycles is a significant advantage in many