Compare the performance of Dijkstra and Bellman ford algorithm for the single source shortest path problem.

#include <stdio.h>

#include <stdlib.h>

#include <limits.h>

#include <stdbool.h>

#include <time.h>

#define INF INT\_MAX

#define V 5 // Number of vertices in the graph

// Edge structure for Bellman-Ford

struct Edge {

int src, dest, weight;

};

// Utility function to print distances

void printDistances(int dist[], int n) {

printf("Vertex \t Distance from Source\n");

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

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

}

}

// Dijkstra's algorithm using adjacency matrix and a boolean visited array

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

int dist[V];

bool visited[V];

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

dist[i] = INF;

visited[i] = false;

}

dist[src] = 0;

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

int u = -1;

// Find the unvisited vertex with the smallest distance

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

if (!visited[i] && (u == -1 || dist[i] < dist[u]))

u = i;

visited[u] = true;

// Update distance value of adjacent vertices

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

if (graph[u][v] && !visited[v] && dist[u] != INF && dist[u] + graph[u][v] < dist[v])

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

}

printf("\nDijkstra's Algorithm Result:\n");

printDistances(dist, V);

}

// Bellman-Ford algorithm

void bellmanFord(struct Edge edges[], int edgeCount, int src) {

int dist[V];

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

dist[i] = INF;

dist[src] = 0;

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

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

int u = edges[j].src;

int v = edges[j].dest;

int weight = edges[j].weight;

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

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

}

}

// Check for negative weight cycles

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

int u = edges[i].src;

int v = edges[i].dest;

int weight = edges[i].weight;

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

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

return;

}

}

printf("\nBellman-Ford Algorithm Result:\n");

printDistances(dist, V);

}

// Main function to compare performance

int main() {

// Graph representation as adjacency matrix for Dijkstra

int graph[V][V] = {

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

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

{0, 0, 0, 4, 0},

{7, 0, 6, 0, 0},

{0, 3, 9, 2, 0}

};

// Edge list for Bellman-Ford

struct Edge edges[] = {

{0, 1, 10}, {0, 4, 5}, {1, 2, 1}, {1, 4, 2},

{2, 3, 4}, {3, 0, 7}, {3, 2, 6}, {4, 1, 3},

{4, 2, 9}, {4, 3, 2}

};

int edgeCount = sizeof(edges) / sizeof(edges[0]);

int src = 0; // Source vertex

// Measure time for Dijkstra's algorithm

clock\_t start = clock();

dijkstra(graph, src);

clock\_t end = clock();

double dijkstraTime = ((double)(end - start)) / CLOCKS\_PER\_SEC;

printf("Dijkstra's Algorithm Execution Time: %.6f seconds\n", dijkstraTime);

// Measure time for Bellman-Ford algorithm

start = clock();

bellmanFord(edges, edgeCount, src);

end = clock();

double bellmanFordTime = ((double)(end - start)) / CLOCKS\_PER\_SEC;

printf("Bellman-Ford Algorithm Execution Time: %.6f seconds\n", bellmanFordTime);

return 0;

}