Graph Traversal

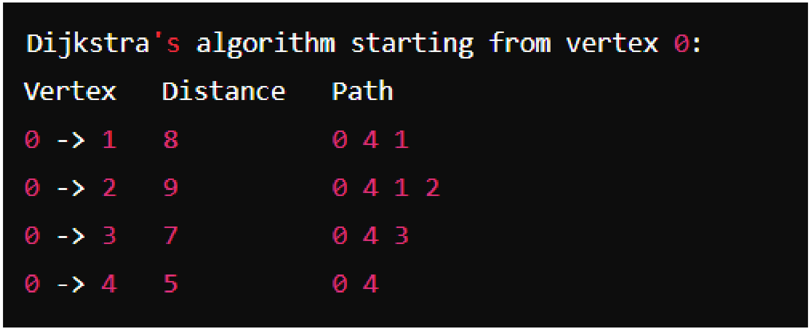
Write a C program to create a graph and find the shortest path using Dijikstra's Algorithm.

PROGRAM:

```
#include <stdio.h>

#include <stdlib.h>

#include <limits.h>


// Define the structure for the adjacency list node

struct AdjListNode {

   int dest;

   int weight;

   struct AdjListNode* next;

};


// Define the structure for the adjacency list

struct AdjList {

   struct AdjListNode* head;

};


// Define the structure for the graph

struct Graph {

   int V;

   struct AdjList* array;

};


// Create a new adjacency list node

struct AdjListNode* newAdjListNode(int dest, int weight) {

   struct AdjListNode* newNode = (struct AdjListNode*)malloc(sizeof(struct AdjListNode));

   newNode->dest = dest;

   newNode->weight = weight;

   newNode->next = NULL;

   return newNode;

}
```

```
// Create a graph with V vertices
struct Graph* createGraph(int V) {
    struct Graph* graph = (struct Graph*)malloc(sizeof(struct Graph));
    graph->V = V;
    graph->array = (struct AdjList*)malloc(V * sizeof(struct AdjList));
    for (int i = 0; i < V; ++i)
        graph->array[i].head = NULL;
    return graph;
}

// Add an edge to the graph
void addEdge(struct Graph* graph, int src, int dest, int weight) {
    struct AdjListNode* newNode = newAdjListNode(dest, weight);
    newNode->next = graph->array[src].head;
    graph->array[src].head = newNode;

    // For undirected graph, add reverse edge
    newNode = newAdjListNode(src, weight);
    newNode->next = graph->array[dest].head;
    graph->array[dest].head = newNode;
}

// A utility function to find the vertex with minimum distance value
int minDistance(int dist[], int sptSet[], int V) {
    int min = INT_MAX, min_index;
    for (int v = 0; v < V; v++)
        if (sptSet[v] == 0 && dist[v] <= min)
            min = dist[v], min_index = v;
    return min_index;
}

// Function to print the shortest path from source to j using parent array
void printPath(int parent[], int j) {
    if (parent[j] == -1)
        return;
    printPath(parent, parent[j]);
    printf("%d ", j);
}

// A utility function to print the constructed distance array
void printSolution(int dist[], int V, int parent[], int src) {
    printf("Vertex\t Distance\tPath");
    for (int i = 0; i < V; i++) {
        if (i != src) {
            printf("\n%d -> %d\t %d\t\t%d ", src, i, dist[i], src);
            printPath(parent, i);
        }
    }
    printf("\n");
}

// Function that implements Dijkstra's single source shortest path algorithm
void dijkstra(struct Graph* graph, int src) {
    int V = graph->V;
    int dist[V];
    int sptSet[V];
    int parent[V];

    for (int i = 0; i < V; i++) {
        dist[i] = INT_MAX;
        sptSet[i] = 0;
        parent[i] = -1;
    }

    dist[src] = 0;

    for (int count = 0; count < V - 1; count++) {
        int u = minDistance(dist, sptSet, V);
        sptSet[u] = 1;

        struct AdjListNode* node = graph->array[u].head;
        while (node != NULL) {
            int v = node->dest;
            if (!sptSet[v] && dist[u] != INT_MAX && dist[u] + node->weight < dist[v]) {
                parent[v] = u;
                dist[v] = dist[u] + node->weight;
            }
            node = node->next;
        }
    }

    printSolution(dist, V, parent, src);
}

// Main function to test the above functions
int main() {
    int V = 5; // Number of vertices in the graph
    struct Graph* graph = createGraph(V);
    addEdge(graph, 0, 1, 10);
    addEdge(graph, 0, 4, 5);
    addEdge(graph, 1, 2, 1);
    addEdge(graph, 1, 4, 2);
    addEdge(graph, 2, 3, 4);
    addEdge(graph, 3, 0, 7);
    addEdge(graph, 3, 2, 6);
    addEdge(graph, 4, 1, 3);
    addEdge(graph, 4, 2, 9);
    addEdge(graph, 4, 3, 2);

    printf("Dijkstra's algorithm starting from vertex 0:\n");
    dijkstra(graph, 0);
    return 0;
}
```

OUTPUT:



```
Dijkstra's algorithm starting from vertex 0:
Vertex   Distance   Path
0 -> 1   8          0 4 1
0 -> 2   9          0 4 1 2
0 -> 3   7          0 4 3
0 -> 4   5          0 4
```