CODE IMPLEMENTATION

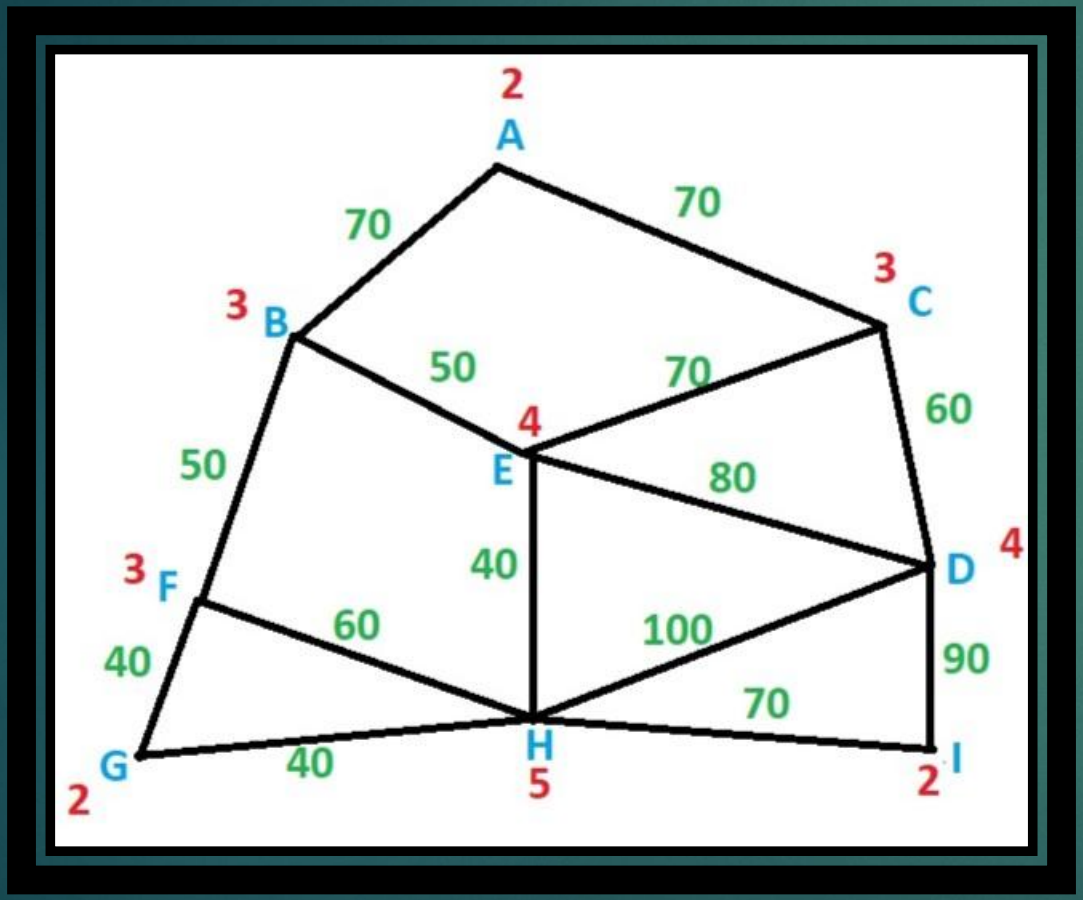
```c
#include <stdio.h>
#include <stdlib.h>
#include <limits.h>

#define MAX_VERTICES 100

// Structure to represent an edge
typedef struct {
    int u, v; // Vertices connected by the edge
    int weight; // Weight of the edge
} Edge;

// Structure to represent a graph
typedef struct {
    int numVertices;
    int numEdges;
    int adjMatrix[MAX_VERTICES][MAX_VERTICES];
} Graph;

// Function to create a new graph
Graph createGraph(int numVertices) {
    Graph graph;
    graph.numVertices = numVertices;
    graph.numEdges = 0;

    // Initialize adjacency matrix with -1 (indicating no edge)
    for (int i = 0; i < numVertices; i++) {
        for (int j = 0; j < numVertices; j++) {
            graph.adjMatrix[i][j] = -1;
        }
    }

    return graph;
}

```

```
36  // Function to add an edge to the graph
37  void addEdge(Graph* graph, int u, int v, int weight) {
38      graph->adjMatrix[u][v] = weight;
39      graph->adjMatrix[v][u] = weight;
40      graph->numEdges++;
41  }
42
43  // Function to find the minimum distance vertex from the set of vertices
44  int minDistance(int dist[], int visited[], int numVertices) {
45      int min = INT_MAX, minIndex;
46
47      for (int v = 0; v < numVertices; v++) {
48          if (!visited[v] && dist[v] <= min) {
49              min = dist[v];
50              minIndex = v;
51          }
52      }
53
54      return minIndex;
55  }
56
57  // Function to find the shortest route and distance using Dijkstra's algorithm
58  void chinesePostman(Graph* graph) {
59      int numVertices = graph->numVertices;
60      int numEdges = graph->numEdges;
61
62      // Array to store the distance of vertices from the source
63      int dist[numVertices];
64
65      // Array to track whether a vertex is visited or not
66      int visited[numVertices];
67
68      // Initialize the distance and visited arrays
69      for (int v = 0; v < numVertices; v++) {
70          dist[v] = INT_MAX;
71          visited[v] = 0;
72      }
```

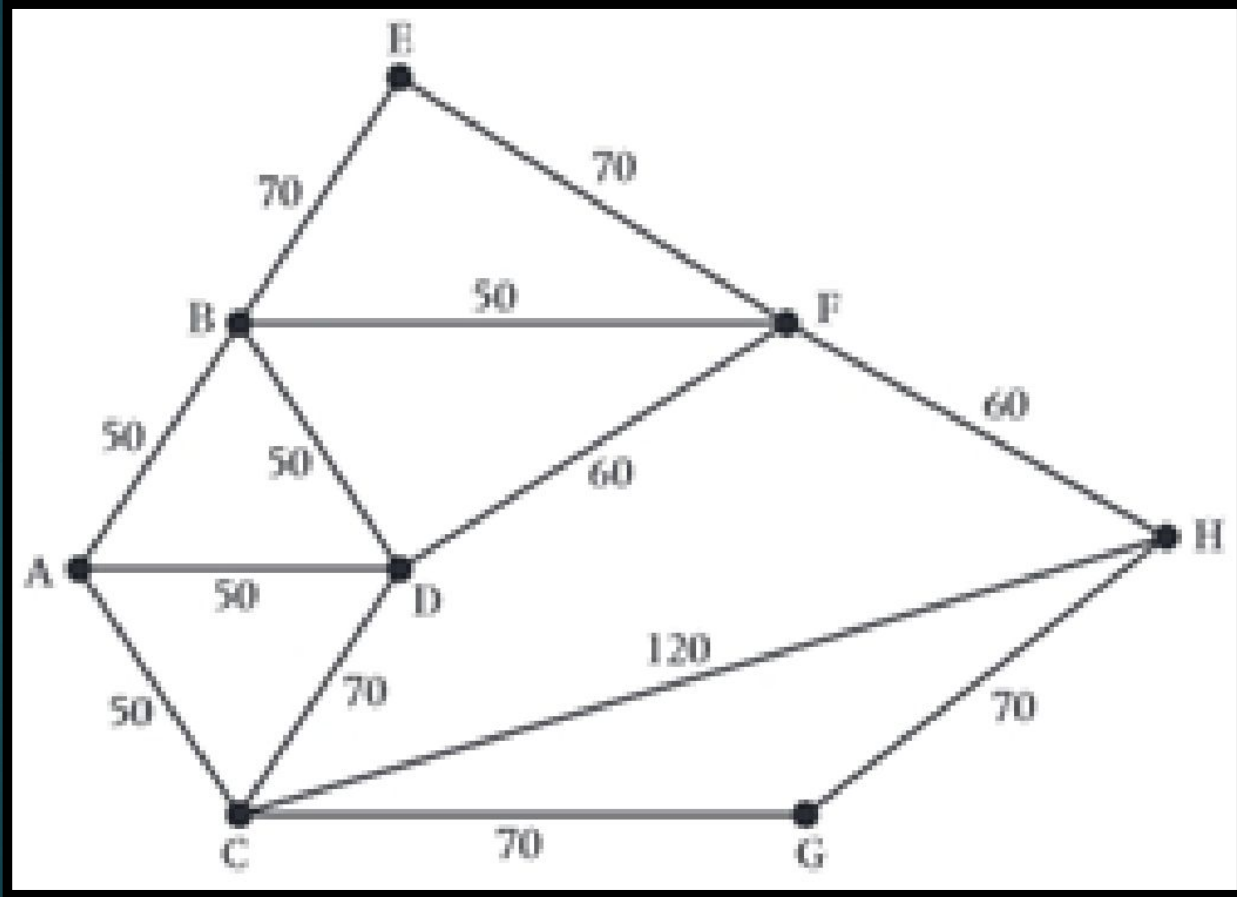
```c
    // Start with the first vertex
    int src = 0;
    dist[src] = 0;

    // Calculate the shortest path for all vertices
    for (int count = 0; count < numVertices - 1; count++) {
        int u = minDistance(dist, visited, numVertices);
        visited[u] = 1;

        for (int v = 0; v < numVertices; v++) {
            if (!visited[v] && graph->adjMatrix[u][v] != -1 && dist[u] != INT_MAX
                && dist[u] + graph->adjMatrix[u][v] < dist[v]) {
                dist[v] = dist[u] + graph->adjMatrix[u][v];
            }
        }
    }

    // Calculate the sum of all edge weights
    int sumOfEdges = 0;
    for (int i = 0; i < numVertices; i++) {
        for (int j = i + 1; j < numVertices; j++) {
            if (graph->adjMatrix[i][j] != -1) {
                sumOfEdges += graph->adjMatrix[i][j];
            }
        }
    }

    // Calculate the total distance of the shortest route
    int totalDistance = sumOfEdges + dist[src] * (numEdges - 1);

    printf("Shortest route distance: %d\n", totalDistance);
}
```

```c
int main() {
    int numVertices, numEdges;
    printf("Enter the number of vertices: ");
    scanf("%d", &numVertices);
    printf("Enter the number of edges: ");
    scanf("%d", &numEdges);

    Graph graph = createGraph(numVertices);

    for (int i = 0; i < numEdges; i++) {
        int u, v, weight;
        printf("Enter edge %d (u, v, weight): ", i + 1);
        scanf("%d %d %d", &u, &v, &weight);
        addEdge(&graph, u, v, weight);
    }

    chinesePostman(&graph);

    return 0;
}
```

INPUTS




```
Enter the number of vertices: 9
Enter the number of edges: 14
Enter edge 1 (u, v, weight): 0 1 70
Enter edge 2 (u, v, weight): 0 2 70
Enter edge 3 (u, v, weight): 1 4 50
Enter edge 4 (u, v, weight): 1 5 50
Enter edge 5 (u, v, weight): 4 7 40
Enter edge 6 (u, v, weight): 5 7 60
Enter edge 7 (u, v, weight): 5 6 40
Enter edge 8 (u, v, weight): 6 7 40
Enter edge 9 (u, v, weight): 2 3 60
Enter edge 10 (u, v, weight): 3 4 80
Enter edge 11 (u, v, weight): 2 4 70
Enter edge 12 (u, v, weight): 3 8 90
Enter edge 13 (u, v, weight): 7 8 70
Enter edge 14 (u, v, weight): 3 7 100
Shortest route distance: 1050
```




```
Enter the number of vertices: 8
Enter the number of edges: 13
Enter edge 1 (u, v, weight): 0 1 50
Enter edge 2 (u, v, weight): 0 3 50
Enter edge 3 (u, v, weight): 0 2 50
Enter edge 4 (u, v, weight): 1 3 50
Enter edge 5 (u, v, weight): 2 3 70
Enter edge 6 (u, v, weight): 1 5 50
Enter edge 7 (u, v, weight): 1 4 70
Enter edge 8 (u, v, weight): 3 5 60
Enter edge 9 (u, v, weight): 4 5 70
Enter edge 10 (u, v, weight): 5 7 60
Enter edge 11 (u, v, weight): 2 7 120
Enter edge 12 (u, v, weight): 2 6 70
Enter edge 13 (u, v, weight): 6 7 70
Shortest route distance: 1000
```