a) Write a C/C++ program for shortest path between any two nodes in the graph.

```
#include <iostream>
#include <vector>
#include <queue>
#include <limits>

using namespace std;

// Structure to represent a graph edge
struct Edge
{
   int to;
   int weight;
   Edge(int t, int w) : to(t), weight(w) {}
};

// Function to find the shortest path using Dijkstra's algorithm
void shortestPath(const vector<vector<Edge>> &graph, int source, int destination)
{
   int n = graph.size();
   vector<int> distance(n, numeric_limits<int>::max());
   vector<int> parent(n, -1);

   distance[source] = 0;
   priority_queue<pair<int, int>, vector<pair<int, int>>, greater<pair<int, int>>> pq;
   pq.push({0, source});

   while (!pq.empty())
   {
      int u = pq.top().second;
      pq.pop();

      for (const Edge &edge : graph[u])
      {
         int v = edge.to;
         int weight = edge.weight;
         if (distance[u] + weight < distance[v])
         {
            distance[v] = distance[u] + weight;
            parent[v] = u;
            pq.push({distance[v], v});
         }
      }
   }

   if (distance[destination] == numeric_limits<int>::max())
   {
      cout << "No path from source to destination.\n";
   }
   else
   {
```

```
        cout << "Shortest path length: " << distance[destination] << endl;
        cout << "Shortest path: ";
        vector<int> path;
        int current = destination;
        while (current != -1)
        {
            path.push_back(current);
            current = parent[current];
        }
        for (int i = path.size() - 1; i >= 0; --i)
        {
            cout << path[i] << " ";
        }
        cout << endl;
    }
}



// Function to detect cycles using DFS
bool hasCycleDFS(const vector<vector<Edge>> &graph, int node, vector<int> &visited, vector<int>
&recStack)
{
    if (visited[node] == 0)
    {
        visited[node] = 1; // Mark as visited
        recStack[node] = 1; // Add to recursion stack

        for (const Edge &edge : graph[node])
        {
            int neighbor = edge.to;
            if (visited[neighbor] == 0 && hasCycleDFS(graph, neighbor, visited, recStack))
            {
                return true;
            }
            else if (recStack[neighbor] == 1) // Found back edge in recursion stack
            {
                return true;
            }
        }
    }
    recStack[node] = 0; // Remove from recursion stack
    return false;
}

// Function to count the number of cycles in the graph
int countCycles(const vector<vector<Edge>> &graph)
{
    int n = graph.size();
    vector<int> visited(n, 0);   // 0: unvisited, 1: visited, 2: fully explored
    vector<int> recStack(n, 0);  // 0: not in recursion stack, 1: in recursion stack
```

```
    int cycleCount = 0;

    for (int i = 0; i < n; ++i)
    {
        if (visited[i] == 0 && hasCycleDFS(graph, i, visited, recStack))
        {
            cycleCount++;
        }
    }

    return cycleCount;
}


int main()
{
    int n, m; // n = number of nodes, m = number of edges
    cout << "Enter n nodes and m edges:  ";
    cin >> n >> m;

    vector<vector<Edge>> graph(n);
    cout << "Enter Edge from node u to node v with weight w:  ";
    for (int i = 0; i < m; ++i)
    {
        int u, v, w;
        cin >> u >> v >> w; // Edge from node u to node v with weight w
        graph[u].emplace_back(v, w);
    }



    //  int cycleCount = countCycles(graph);
    // cout << "Number of cycles: " << cycleCount << endl;

    int source, destination;
    cout << "Enter source and destination:  ";
    cin >> source >> destination;

    shortestPath(graph, source, destination);



    return 0;
}
```

```
PS C:\Users\RAMAVATH SANTHOSH\OneDrive\Desktop\ALL SEMs\SEM3\CCN>  & 'c:\Users\RAMAVATH SANT
-vscode.cpptools-1.16.3-win32-x64\debugAdapters\bin\WindowsDebugLauncher.exe' '--stdin=Micro
enj' '--stdout=Microsoft-MIEngine-Out-1ztw0n45.mx4' '--stderr=Microsoft-MIEngine-Error-gw3x1
MIEngine-Pid-5sl4yhla.ht4' '--dbgExe=C:\msys64\mingw64\bin\gdb.exe' '--interpreter=mi'
Enter n nodes and m edges:  4 4
Enter Edge from node u to node v with weight w:  1 3 5
2 4 10
3 1 5
1 4 10
Enter source and destination:  1 3
Shortest path length: 5
Shortest path: 1 3
PS C:\Users\RAMAVATH SANTHOSH\OneDrive\Desktop\ALL SEMs\SEM3\CCN>
```