Assignment No. 5

Aim: From a given vertex in a weighted connected graph, find shortest paths to other vertices using Dijkstra's algorithm. Write the program in C++.

Objectives:
1. To understand the application of Dijkstra‘s algorithm

Theory:

- **What are Graphs:**

  In simple words, graphs are data structures that are used to depict connections amidst a couple of elements where these elements are called nodes (or vertex) that generally real-time objects, persons or entities and connections amid nodes are termed as edges. Also, two nodes only get connected if there is an edge between them. Generally, graphs are suited to real-world applications, such as graphs can be used to illustrate a transportation system/network, where nodes represent facilities that transfer or obtain products and edges show routes or subways that connect nodes.

  There are two methods of graphs

  - **Undirected Graphs:** For every couple of associated nodes, if an individual could move from one node to another in both directions, then the graph is termed as an undirected graph.
  - **Directed Graphs:** For every couple of associated graphs, if an individual could move from one node to another in a specific (single) direction, then the graph is known as the directed graph. In this case, arrows are implemented rather than simple lines in order to represent directed edges.

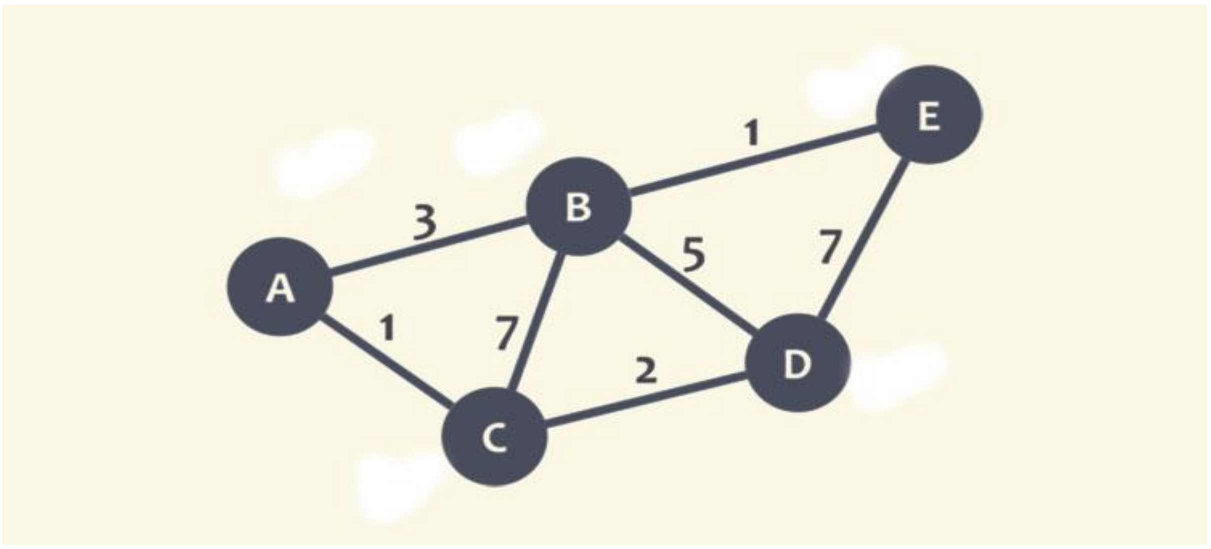
- **What is Dijkstra's Algorithm:**

  Like Prim's MST, generate a SPT (shortest path tree) with a given source as a root. Maintain two sets, one set contains vertices included in the shortest-path tree, other set includes vertices not yet included in the shortest-path tree. At every step of the algorithm, find a vertex that is in the other set (set not yet included) and has a minimum distance from the source.

- **Steps for Algorithm:**
  - Create a spSet (Shortest path tree set) that keeps the tracks of the vertices included in the Shortest path tree.
  - Assign a distance value to all the vertices in the input graph. Initialize all the distance values as INFINITE. Assign the distance value as 0 for the source vertex so that it is picked first.
  - While spSet does'nt include all the vertices
    - Pick a vertex u which is not there in sptSet and has a minimum distance value.

        - Include u to sptSet.
        - Then update distance value of all adjacent vertices of u
            - To update the distance values, iterate through all adjacent vertices.
            - For every adjacent vertex v, if the sum of the distance value of u (from source) and weight of edge u-v, is less than the distance value of v, then update the distance value of v.

- **Implementation of Dijkstra's Algorithm:**
    - **Example:**

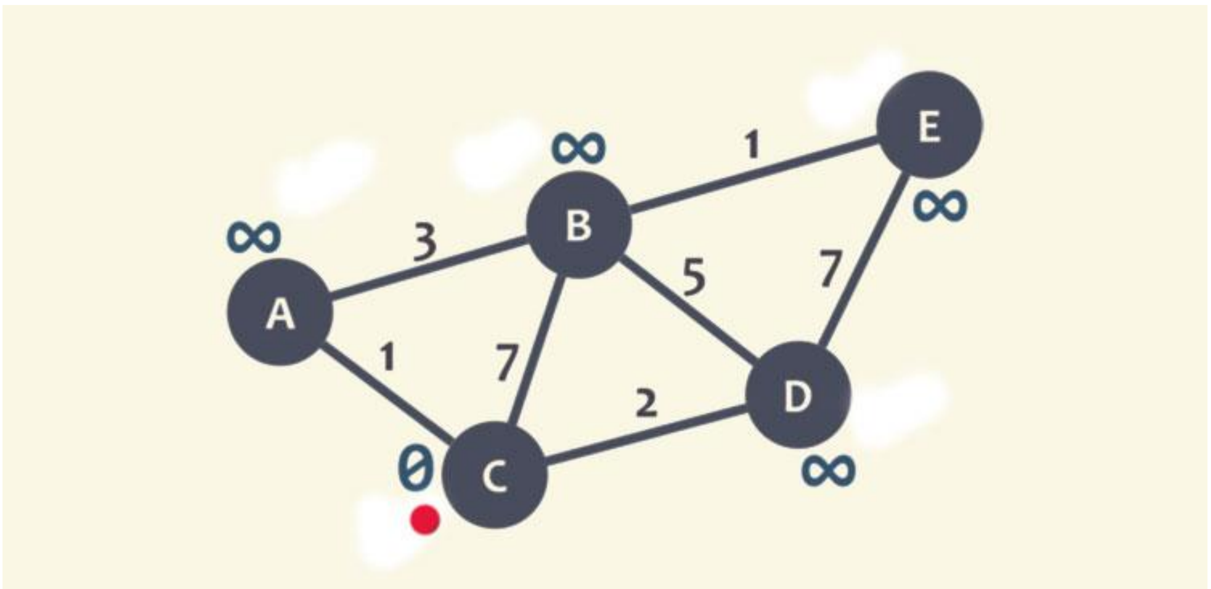


We will calculate the shortest path between node C and the other nodes in the graph.

- **Solution:**

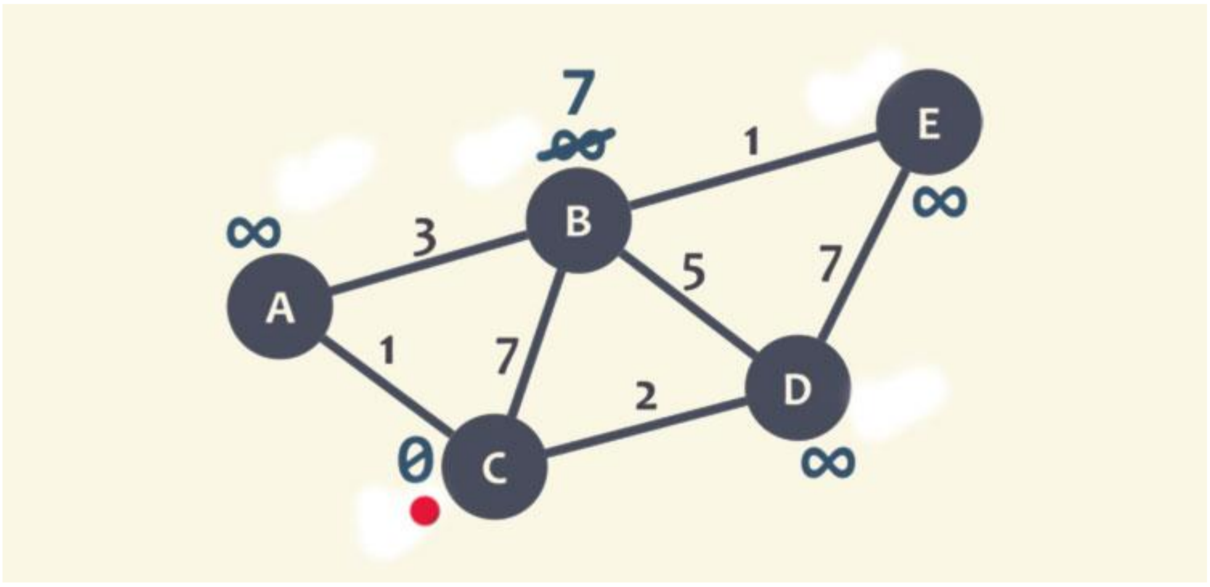
During the execution of the algorithm, each node will be marked with its minimum distance to node C as we have selected node C.

In this case, the minimum distance is 0 for node C. Also, for the rest of the nodes, as we don’t know this distance, they will be marked as infinity (∞), except node C (currently marked as red dot).

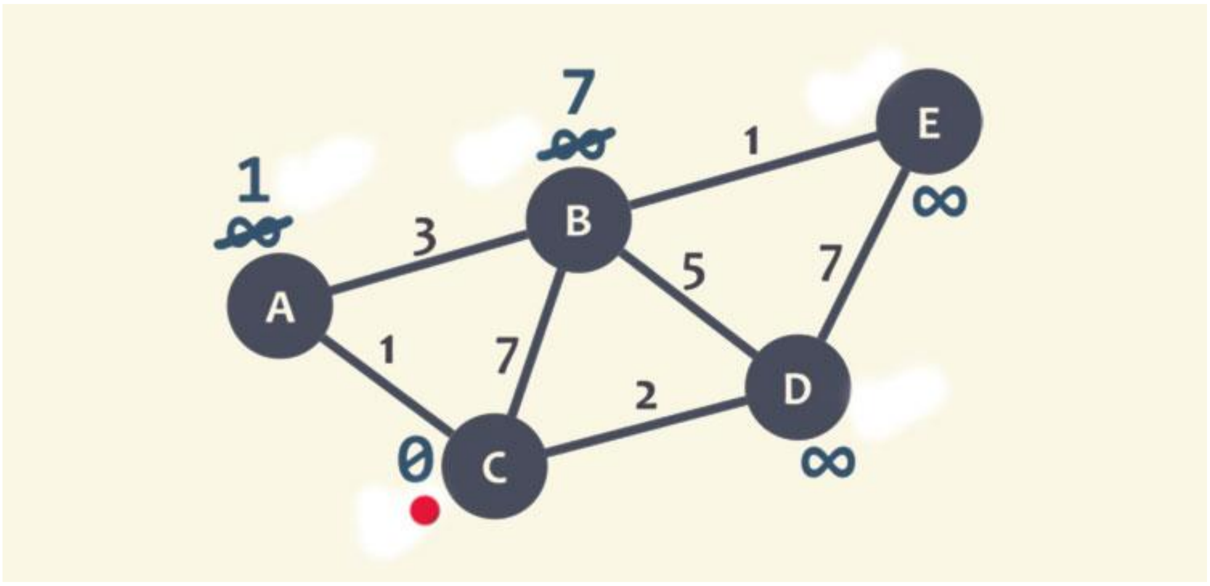


Now the neighbours of node C will be checked, i.e, node A, B, and D. We start with B, here we will add the minimum distance of current node (0) with the weight of the edge (7) that linked the node C to node B and get 0+ 7= 7.

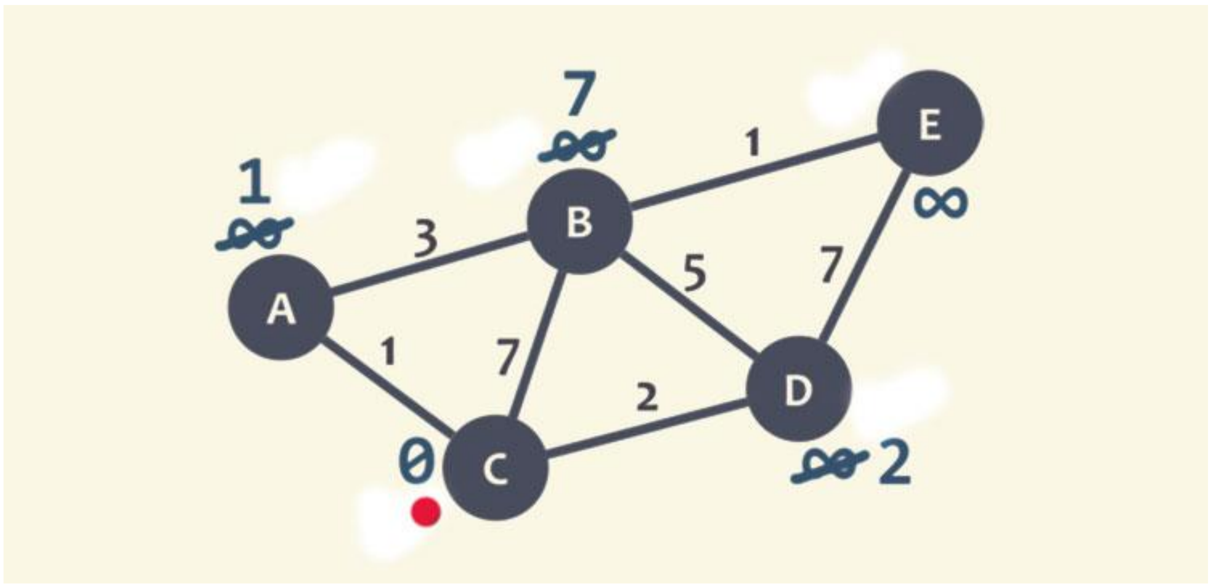
Now, this value will be compared with the minimum distance of B (infinity), the least value is the one that remains the minimum distance of B, like in this case, 7 is less than infinity, and marks the least value to node B.



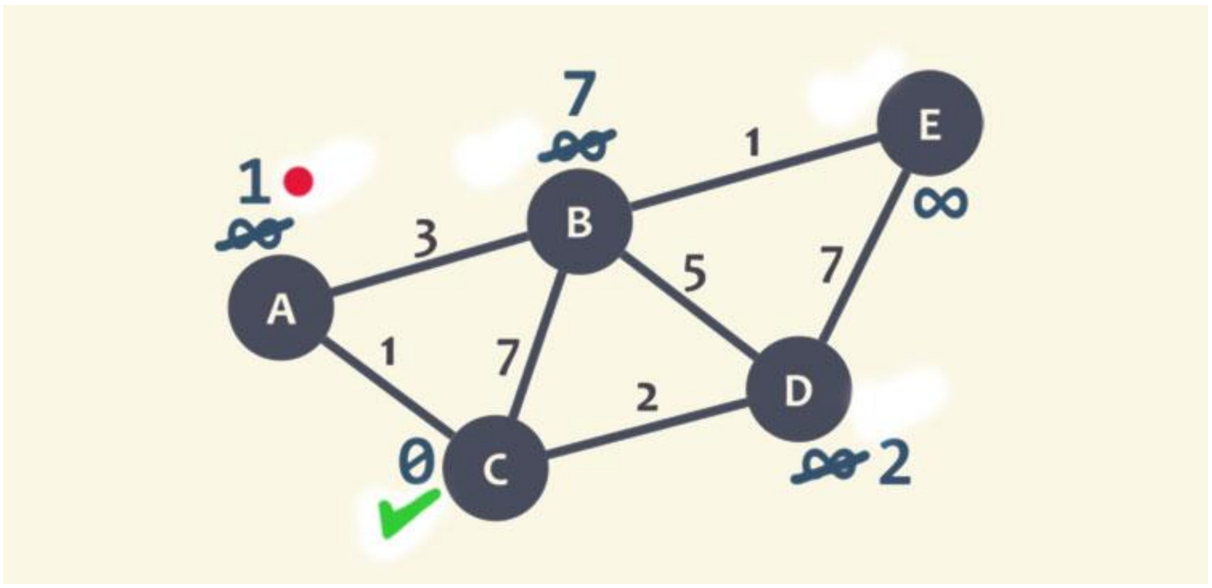
Now, the same process is checked with neighbour A. We add 0 with 1 (weight of edge that connects node C to A), and get 1. Again, 1 is compared with the minimum distance of A (infinity), and marks the lowest value.



The same is repeated with node D, and marked 2 as lowest value at D.

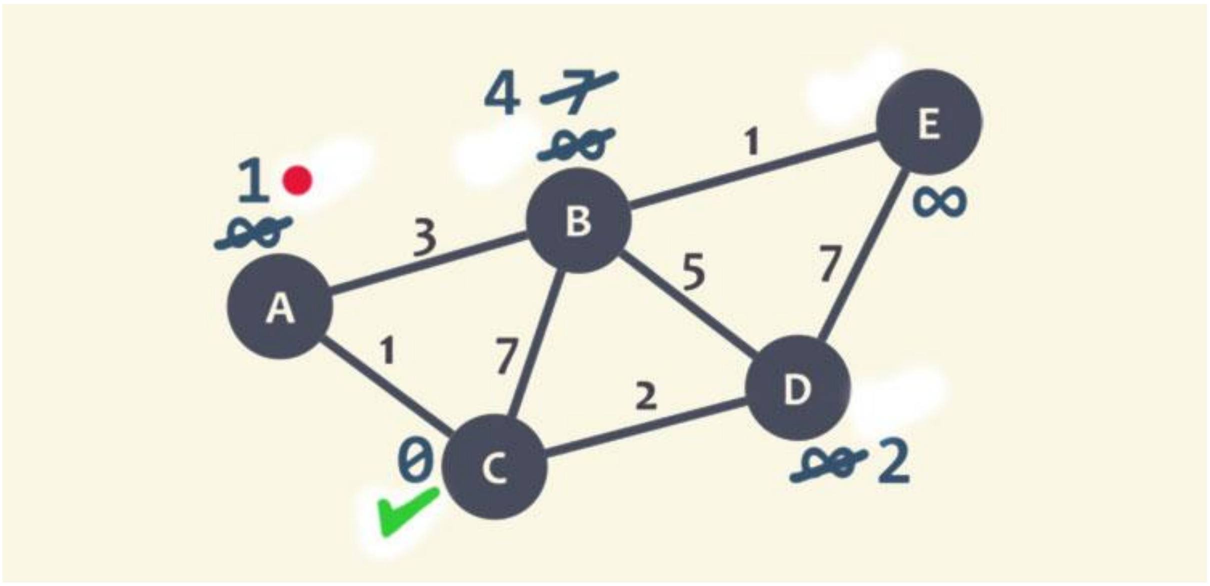


Now, we will select the new current node such that the node must be unvisited with the lowest minimum distance, or the node with the least number and no check mark. Here, node A is the unvisited with minimum distance 1, marked as current node with red dot.



We repeat the algorithm, checking the neighbour of the current node while ignoring the visited node, so only node B will be checked.

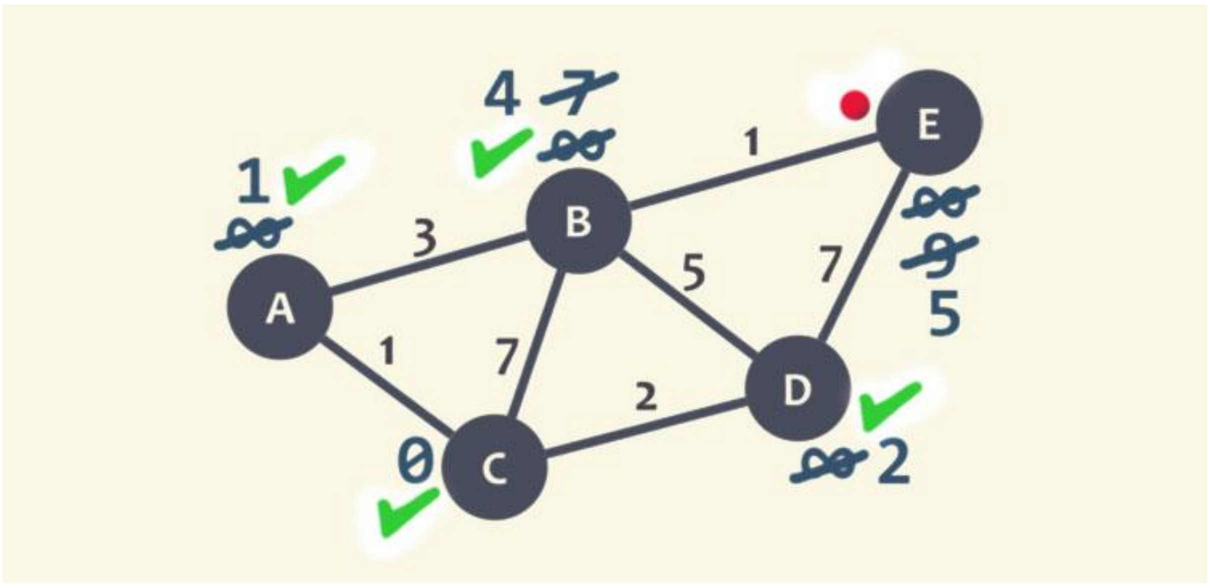
For node B, we add 1 with 3 (weight of the edge connecting node A to B) and obtain 4. This value, 4, will be compared with the minimum distance of B, 7, and mark the lowest value at B as 4.



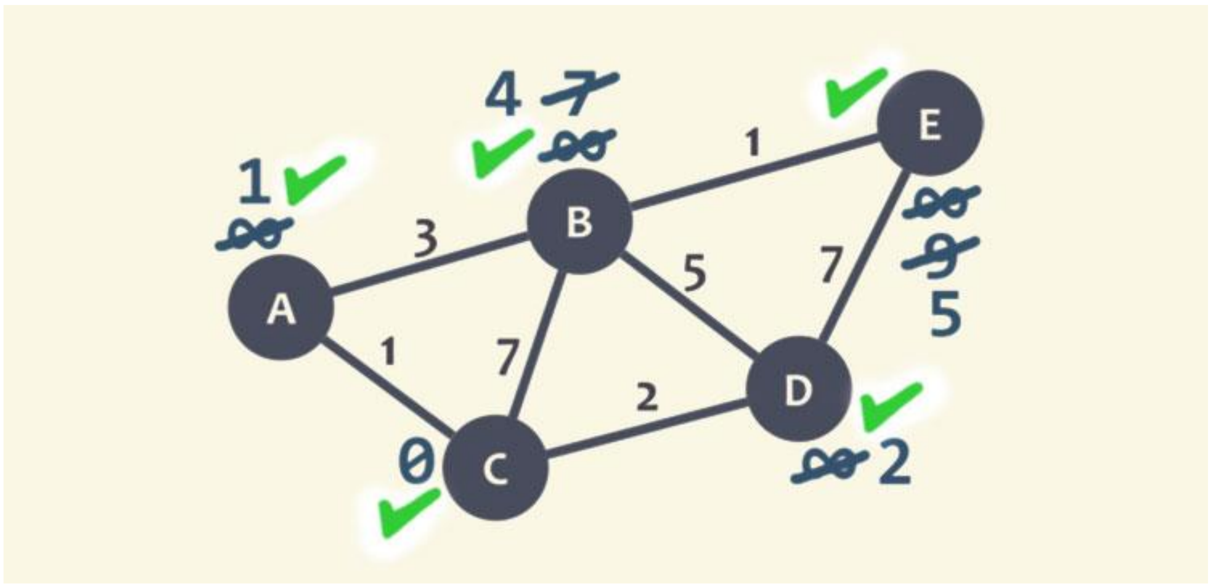
After this, node A marked as visited with a green check mark. The current node is selected as node D, it is unvisited and has a smallest recent distance. We repeat the algorithm and check for node B and E.

For node B, we add 2 to 5, get 7 and compare it with the minimum distance value of B, since 7>4, so leave the smallest distance value at node B as 4. For node E, we obtain 2+ 7= 9, and compare it with the minimum distance of E which is infinity, and mark the smallest value as node E as 9. The node D is marked as visited with a green check mark.

The current node is set as node B, here we need to check only node E as it is unvisited and the node D is visited. We obtain 4+ 1=5, compare it with the minimum distance of the node. As 9 > 5, leave the smallest value at node node E as 5. We mark D as visited node with a green check mark, and node E is set as the current node.



Since it doesn't have any unvisited neighbours, so there is not any requirement to check anything. Node E is marked as a visited node with a green mark.

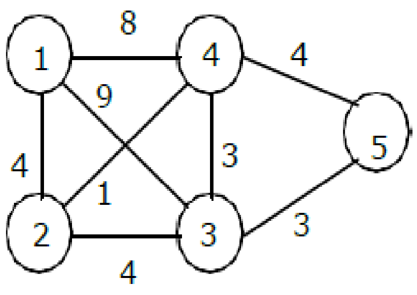


Program:

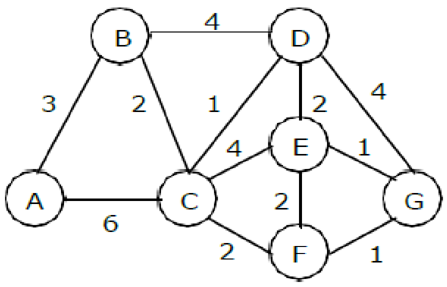
Output:

Solve the examples:

1.



2.



Conclusion: