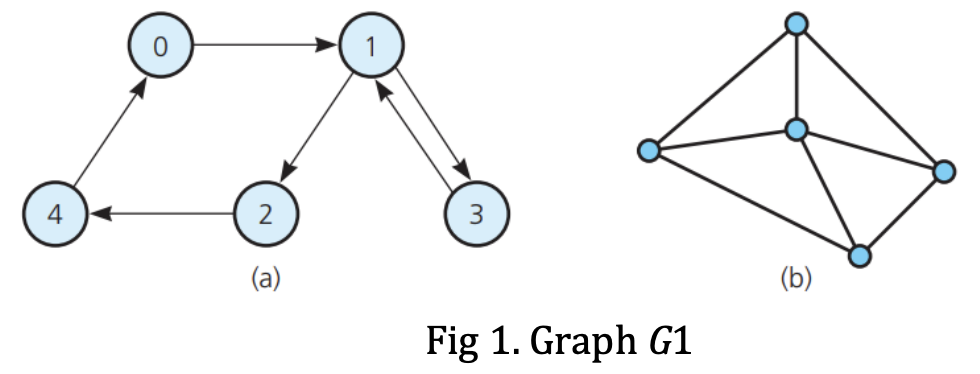
|  |
| --- |
| **CHAPTER 7 HOMEWORK** |

****

**PROBLEM 1:**

**Grap (a):**  - Directed graph

- The vertex 0 is of order 2.

- The vertex 1 is of order 4.

- The vertex 2 is of order 2.

- The vertex 3 is of order 2.

- The vertex 4 is of order 2.

**Grap (b):** - Undirected graph, wheel graph.

- The vertex in the middle of the graph is of order 4.

- The other vertices are of order 3.

**PROBLEM 2:** *Graph (a)*

**- DFS strategy:** 4, 2, 1, 3, 0

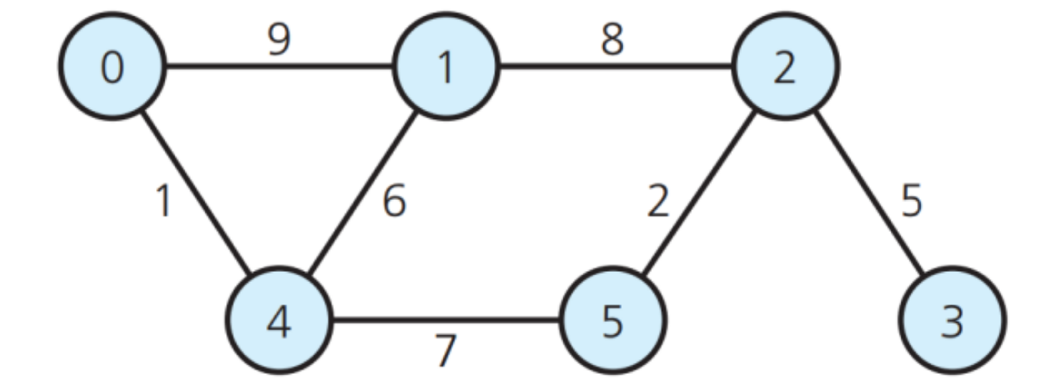
**- BFS strategy:** 4, 2, 1, 3, 0

**PROBLEM 3:** *Graph (a)*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **0** | **1** | **2** | **3** | **4** |
| **0** | 0 | 1 | 0 | 0 | 0 |
| **1** | 0 | 0 | 1 | 1 | 0 |
| **2** | 0 | 0 | 0 | 0 | 1 |
| **3** | 0 | 1 | 0 | 0 | 0 |
| **4** | 1 | 0 | 0 | 0 | 0 |

**PROBLEM 4:** No, it isn’t. Because to contain a simple cycle, a connected undirected single graph must have at least n vertices and n edges.

**PROBLEM 5:**

****

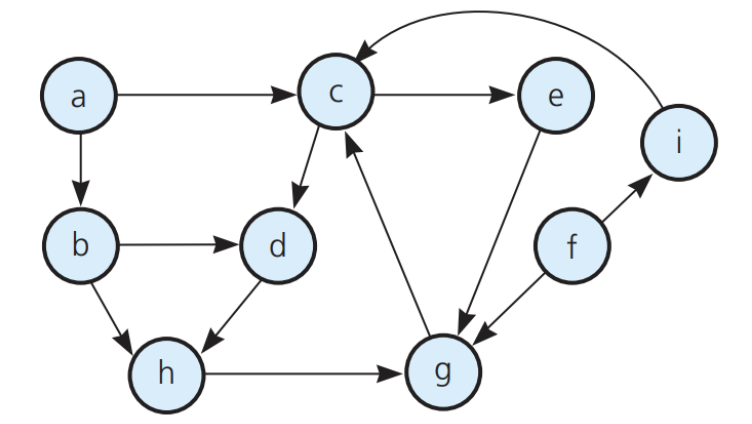
**Adjacency matrix**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **0** | **1** | **2** | **3** | **4** | **5** |
| **0** |  | 9 |  |  | 1 |  |
| **1** | 9 |  | 8 |  | 6 |  |
| **2** |  | 8 |  | 5 |  | 2 |
| **3** |  |  | 5 |  |  |  |
| **4** | 1 | 6 |  |  |  | 7 |
| **5** |  |  | 2 |  | 7 |  |

**Adjacency List**

|  |  |
| --- | --- |
| **NODE** | **Adjacency List** |
| **0** | 1, 4 |
| **1** | 0, 2, 4 |
| **2** | 1, 3, 5 |
| **3** | 2 |
| **4** | 0, 1, 5 |
| **5** | 2, 4 |

**PROBLEM 6:**

****

**Adjacency matrix**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **a** | **b** | **c** | **d** | **e** | **f** | **g** | **h** | **i** |
| **a** | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| **b** | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 |
| **c** | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| **d** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| **e** | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| **f** | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 |
| **g** | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| **h** | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| **i** | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |

**Adjacency List**

|  |  |  |  |
| --- | --- | --- | --- |
| **NODE** | **Adjacency List** | **NODE** | **Adjacency List** |
| **a** | b, c | **f** | g, i |
| **b** | d, h | **g** | c |
| **c** | d, e | **h** | g |
| **d** | h | **i** | c |
| **e** | g |

**PROBLEM 7: A graph conatins a cycle**

**class** Graph

{

**int** V; // No. of vertices

list<**int**> \*adj; // Pointer to an array containing adjacency lists

**bool** isCyclicUtil(**int** v, **bool** visited[], **bool** \*rs);

**public**:

**bool** isCyclic(); // returns true if there is a cycle in this graph

};

**bool** Graph::isCyclicUtil(**int** v, **bool** visited[], **bool** \*recStack)

{

**if**(visited[v] == **false**)

{

// Mark the current node as visited and part of recursion stack

visited[v] = **true**;

recStack[v] = **true**;

// Recur for all the vertices adjacent to this vertex

list<**int**>::iterator i;

**for**(i = adj[v].begin(); i != adj[v].end(); ++i)

{

**if** ( !visited[\*i] && isCyclicUtil(\*i, visited, recStack) )

**return** **true**;

**else** **if** (recStack[\*i])

**return** **true**;

}

}

recStack[v] = **false**; // remove the vertex from recursion stack

**return** **false**;

}

// Returns true if the graph contains a cycle, else false.

**bool** Graph::isCyclic()

{

// Mark all the vertices as not visited and not part of recursion stack

**bool** \*visited = **new** **bool**[V];

**bool** \*recStack = **new** **bool**[V];

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

{

visited[i] = **false**;

recStack[i] = **false**;

}

// Call the recursive helper function to detect cycle in different DFS trees

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

**if** (isCyclicUtil(i, visited, recStack))

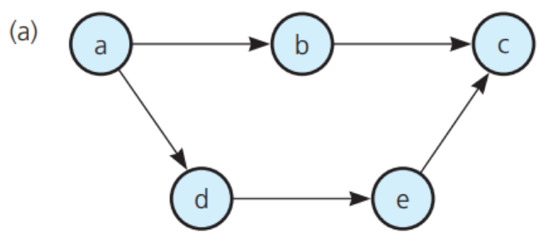
**return** **true**;

**return** **false**;

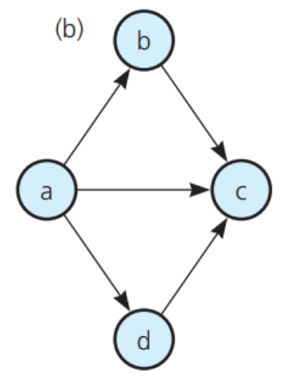
}

**PROBLEM 8:**

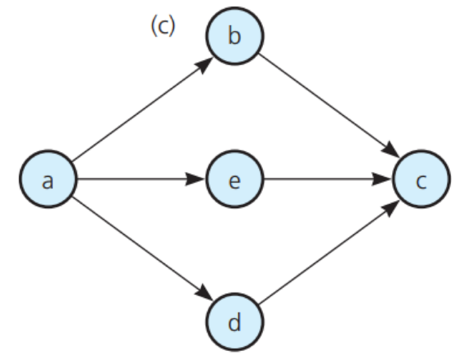
**a) a d e b c**

****

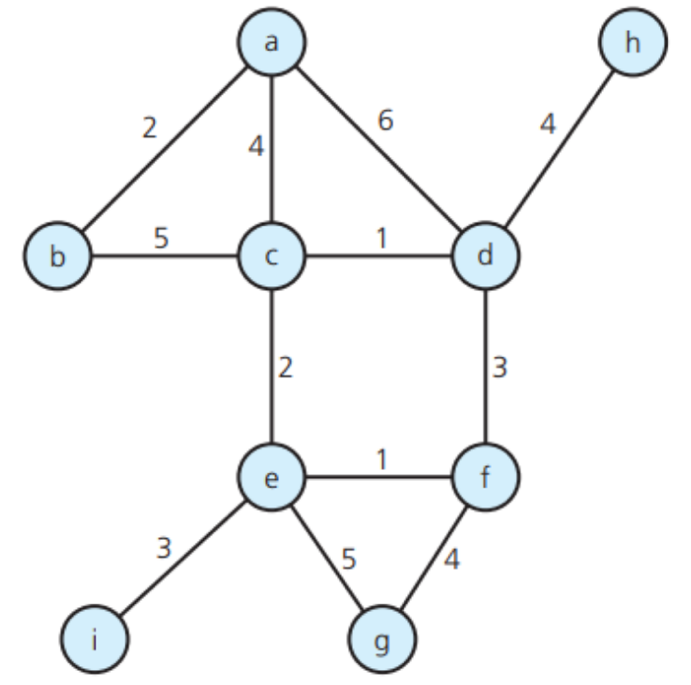
**b) a d b c**

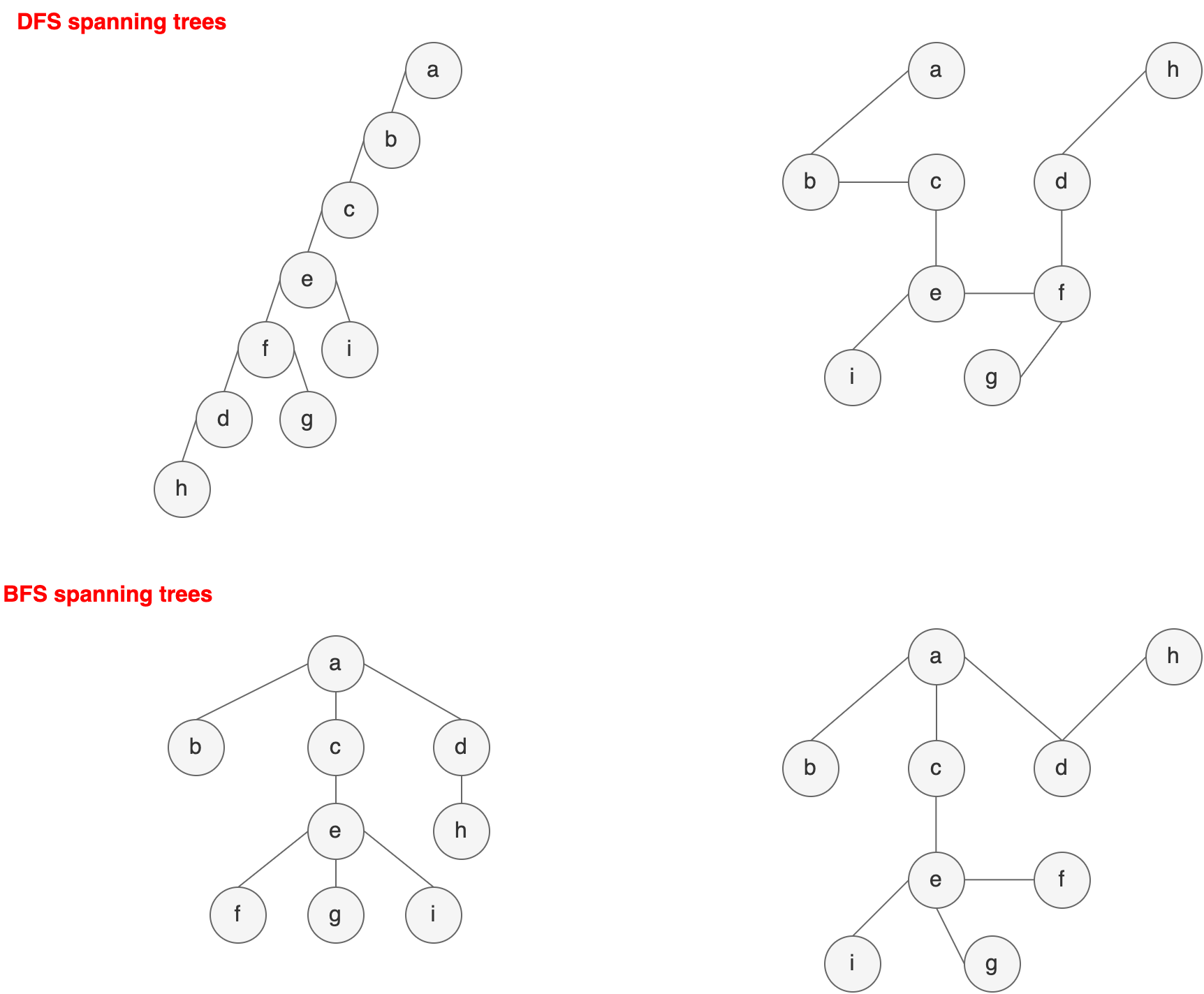
****

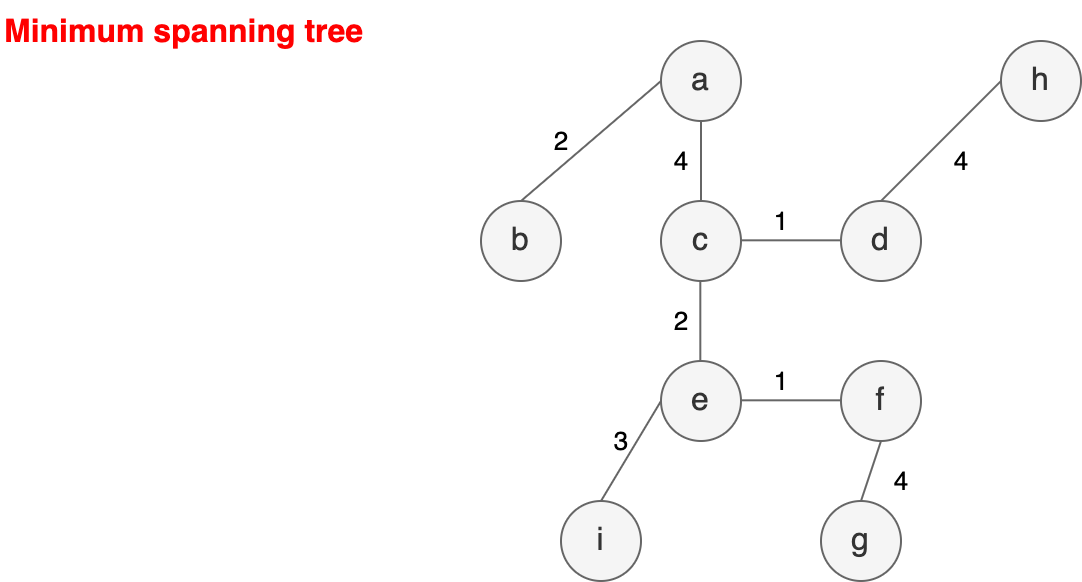
**c) a e d b c**

****

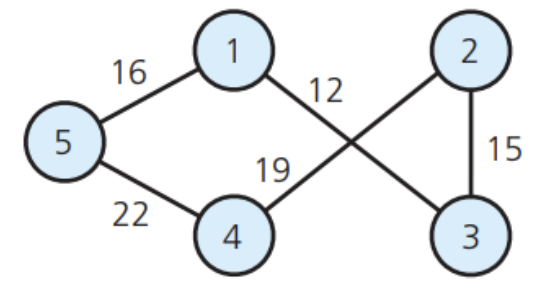
**PROBLEM 10:**

****

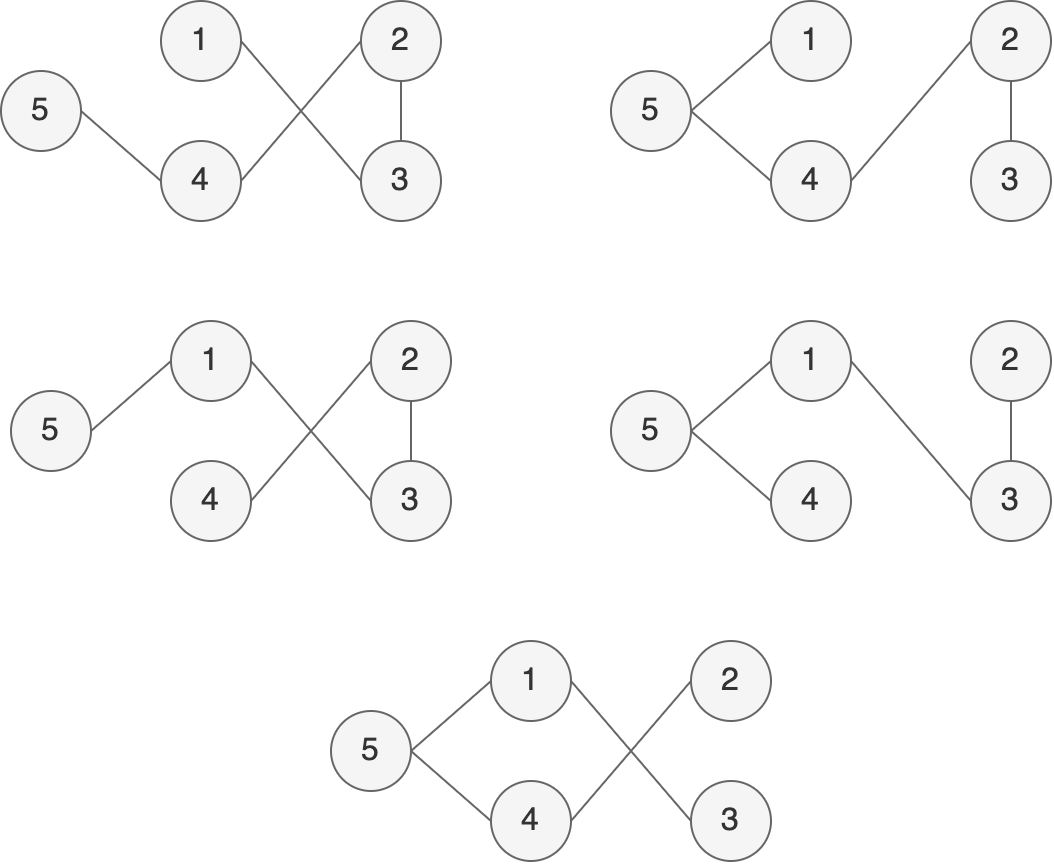
****



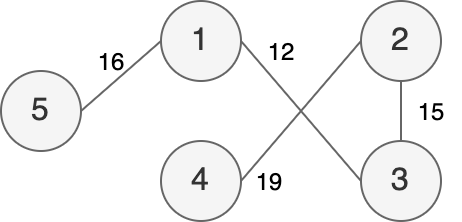
**PROBLEM 11:**

****

**a) All the possible spanning trees**

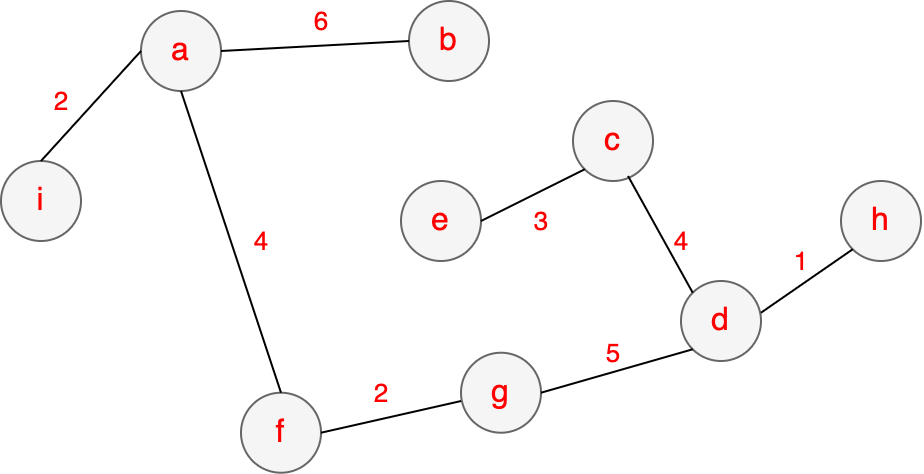


**b) The minimum spanning tree**

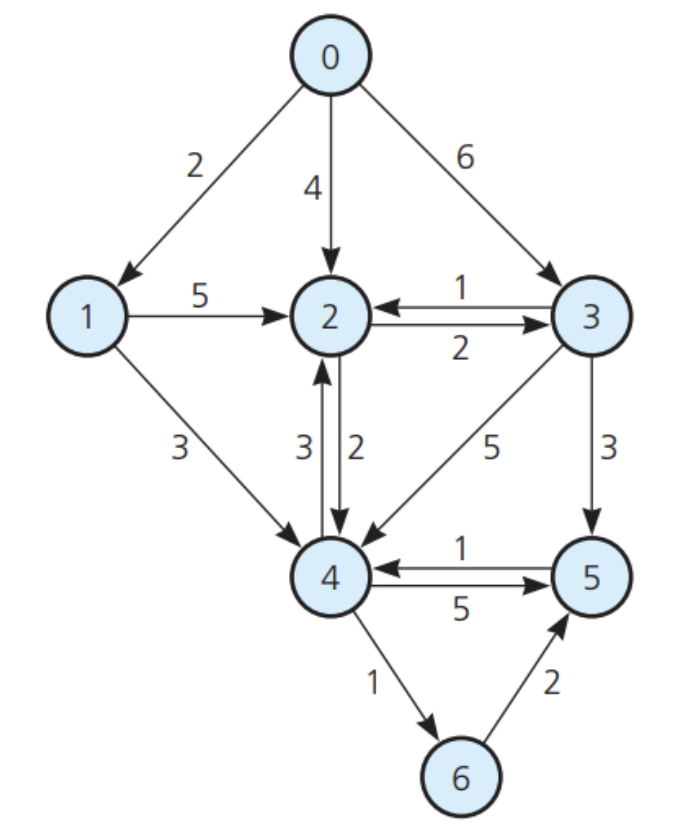


**PROBLEM 12:**

The minimum spanning tree for the graph in Fig. 7 when I start with vertex g and vertex c



**PROBLEM 13:**

****

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Step by step** | **0** | **1** | **2** | **3** | **4** | **5** | **6** |
| **Step 1** | **0 (0)** | + | + | + | + | + | + |
| **Step 2** |  | **2 (0)** | **4 (0)** | **6 (0)** | + | + | + |
| **Step 3** |  |  | **4 (0)** | **6 (0)** | **5 (1)** | + | + |
| **Step 4** |  |  |  | **6 (2)** | **5 (1)** | + | + |
| **Step 5** |  |  |  | **6 (2)** |  | **10 (4)** | **6 (4)** |
| **Step 6** |  |  |  | **6 (2)** |  | **8 (6)** |  |
| **Step 7** |  |  |  |  |  | **8 (6)** |  |