Program : **B.Tech**

Subject Name: **Discrete Structure**

Subject Code: **CS-302**

Semester: **3rd**

Subject Notes
CS301 - Discrete Structures
UNIT-4

Graph

Definition – A graph (denoted as *G*=(*V*,*E*)) consists of a non-empty set of vertices or nodes V and a set of edges E.

Example – Let us consider, a Graph is *G*=(*V*,*E*)where *V*={*a*,*b*,*c*,*d*} and *E*={{*a*,*b*},{*a*,*c*},{*b*,*c*},{*c*,*d*}}

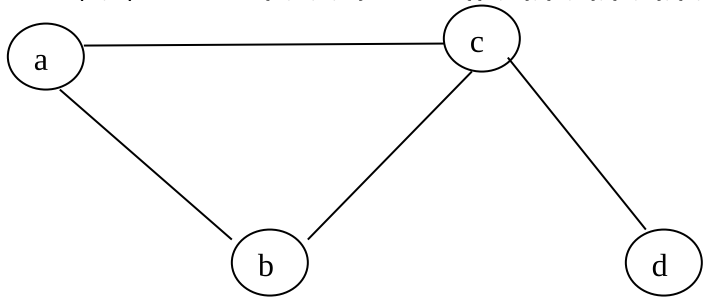

Figure 4.1 Example of Graph

Degree of a Vertex − The degree of a vertex V of a graph G (denoted by deg (V)) is the number of edges incident with the vertex V.

Example Consider the figure no. 4.1

Vertex	Degree	Even / Odd
a	2	even
b	2	even
c	3	odd
d	1	odd

Table 4.1 Degree of Vertex

Even and Odd Vertex − If the degree of a vertex is even, the vertex is called an even vertex and if the degree of a vertex is odd, the vertex is called an odd vertex.

Degree of a Vertex − The degree of a graph is the largest vertex degree of that graph. For the above graph the degree of the graph is 3.

The Handshaking Lemma − In a graph, the sum of all the degrees of all the vertices is equal to twice the number of edges.

Types of Graphs

There are different types of graphs, which we will learn in the following section.

1. **Null Graph -** A null graph has no edges. The null graph of n vertices is denoted by Nn

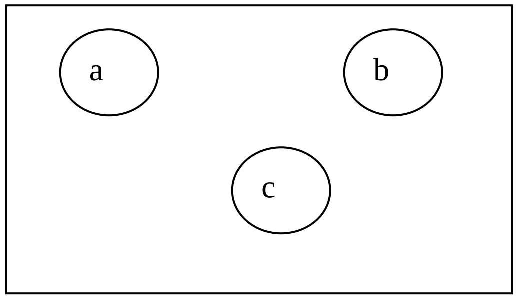

Figure 4.2 Null Graph

2. **Simple Graph** -A graph is called simple graph/strict graph if the graph is undirected and does not contain any loops or multiple edges.

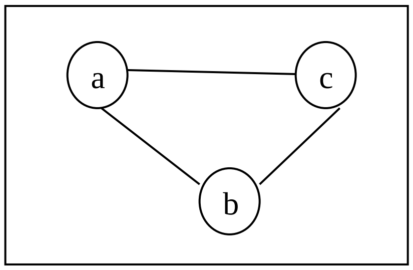

Figure 4.3 Simple Graph

3. **Directed and Undirected Graph -** A graph *G*=(*V*,*E*) is called a directed graph if the edge set is made of ordered vertex pair and a graph is called undirected if the edge set is made of unordered vertex pair.

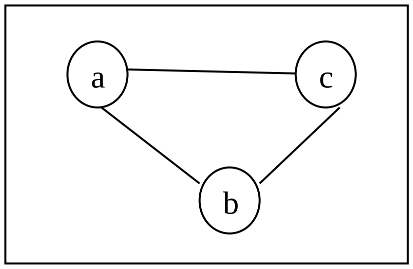

Figure 4.4 Undirected Graph

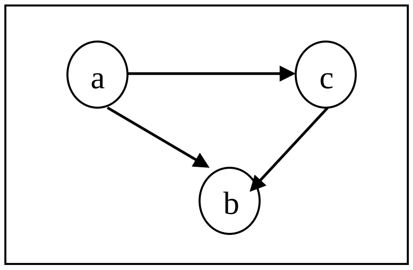

Figure 4.5 Directed Graph

4. **Connected and Disconnected Graph -** A graph is connected if any two vertices of the graph are connected by a path; while a graph is disconnected if at least two vertices of the graph are not connected by a path. If a graph G is disconnected, then every maximal connected subgraph of *G* is called a connected component of the graph *G*

.

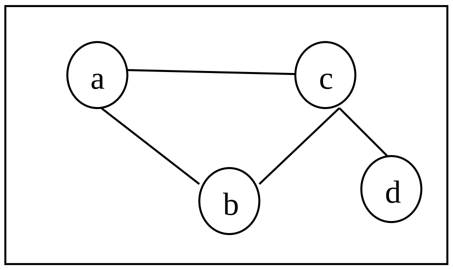

Figure 4.6 Connected Graph

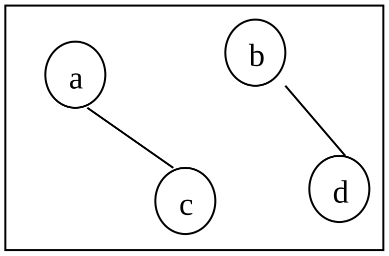

Figure 4.7 Disconnected Graph

5. **Regular Graph -** A graph is regular if all the vertices of the graph have the same degree. In a regular graph G of degree *r*, the degree of each vertex of *G* is r.

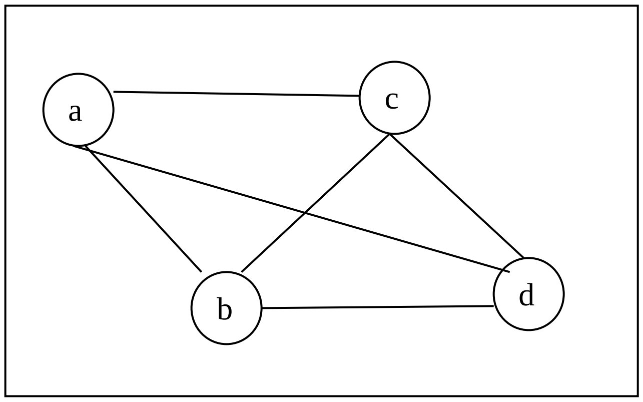

Figure 4.8 Regular Graph

6. **Complete Graph-** A graph is called complete graph if every two vertices pair are joined by exactly one edge. The complete graph with n vertices is denoted by *Kn*

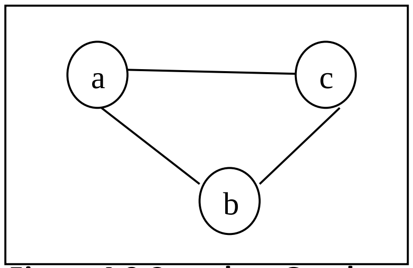

Figure 4.9 Complete Graph

7. **Cycle Graph**- If a graph consists of a single cycle, it is called cycle graph. The cycle graph with n vertices is denoted by *Cn*

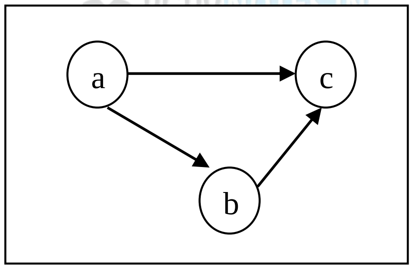

Figure 4.10 Cycle Graph

8. **Bipartite Graph** - If the vertex-set of a graph G can be split into two disjoint sets, *V*1 and *V*2, in such a way that each edge in the graph joins a vertex in *V*1 to a vertex in *V*2, and there are no edges in G that connect two vertices in *V*1 or two vertices in *V*2, then the graph *G* is called a bipartite graph.

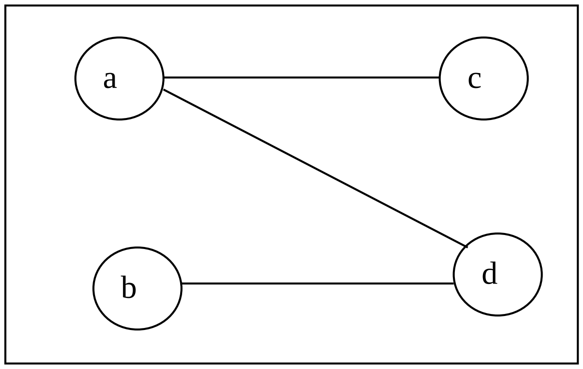

Figure 4.11 Bipartite Graph

9. **Complete Bipartite Graph-** A complete bipartite graph is a bipartite graph in which each vertex in the first set is joined to every single vertex in the second set. The complete bipartite graph is denoted by *Kx,y* where the graph *G* contains *x* vertices in the first set and *y* vertices in the second set.

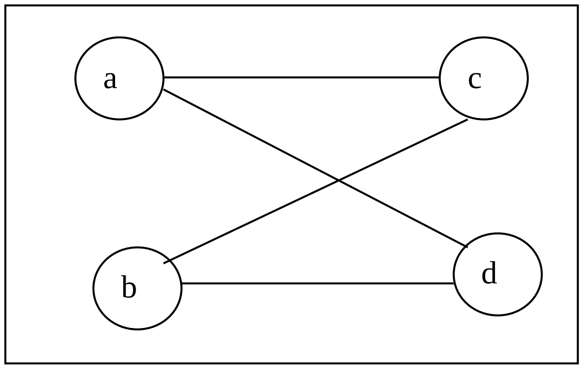

Figure 4.12 Complete Bipartite Graph

Representation of Graphs

There are mainly two ways to represent a graph –

1. Adjacency Matrix
2. Adjacency List

1. **Adjacency Matrix -** An Adjacency Matrix *A*[*V*][*V*] is a 2D array of size *V*×*V* where *V* is the number of vertices in a undirected graph. If there is an edge between *Vx* to *Vy* then the value of *A*[*Vx*][*Vy*]=1 and *A*[*Vy*][*Vx*]=1, otherwise the value will be zero. And for a directed graph, if there is an edge between *Vx* to *Vy*, then the value of *A*[*Vx*][*Vy*]=1 , otherwise the value will be zero.

Adjacency Matrix of an Undirected Graph

Let us consider the following undirected graph and construct the adjacency matrix –

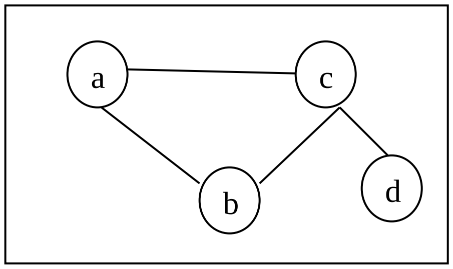

Figure 4.13 Undirected Graph

Adjacency matrix of the above undirected graph will be –

	a	**b**	**c**	**d**
a	0	1	1	0
b	1	0	1	0
c	1	1	0	1
d	0	0	1	0

Table 4.2 Representation of Adjacency matrix of an Undirected Graph

Adjacency Matrix of a Directed Graph

Let us consider the following directed graph and construct its adjacency matrix –

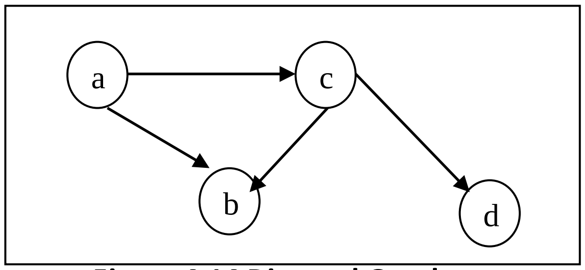

Figure 4.14 Directed Graph

Adjacency matrix of the above directed graph will be –

	a	b	c	d
a	0	1	1	0
b	0	0	1	0
c	0	0	0	1
d	0	0	0	0

Table 4.3 Representation of Adjacency matrix of an Directed Graph

2. **Adjacency List -** In adjacency list, an array (*A*[*V*]) of linked lists is used to represent the graph G with *V* number of vertices. An entry *A*[*Vx*] represents the linked list of vertices adjacent to the *Vx–th* vertex. The adjacency list of the undirected graph is as shown in the figure below –

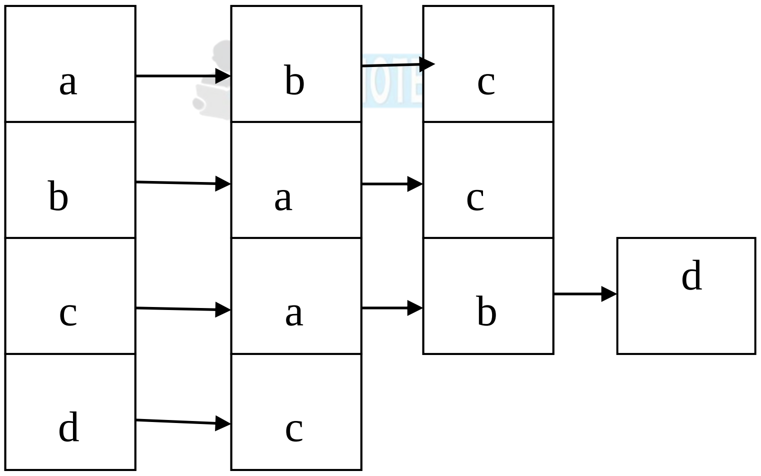

Figure 4.15 Adjacency List

Planar graph – A graph *G* is called a planar graph if it can be drawn in a plane without any edges crossed. If we draw graph in the plane without edge crossing, it is called embedding the graph in the plane.

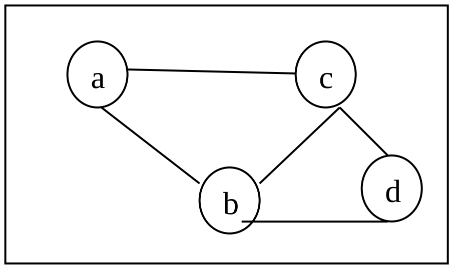

Figure 4.15 Planar Graph

Non-planar graph – A graph is non-planar if it cannot be drawn in a plane without graph edges crossing.

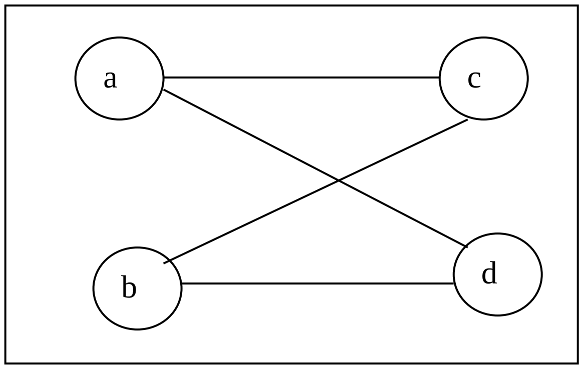

Figure 4.16 Non Planar Graph

Multi-Graph- If in a graph multiple edges between the same set of vertices are allowed, it is called Multigraph. In other words, it is a graph having at least one loop or multiple edges.

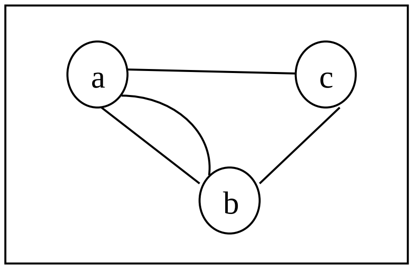

Figure 4.17 Multi Graph

Weighted Graph- In a weighted graph, each edge is assigned a weight or cost.

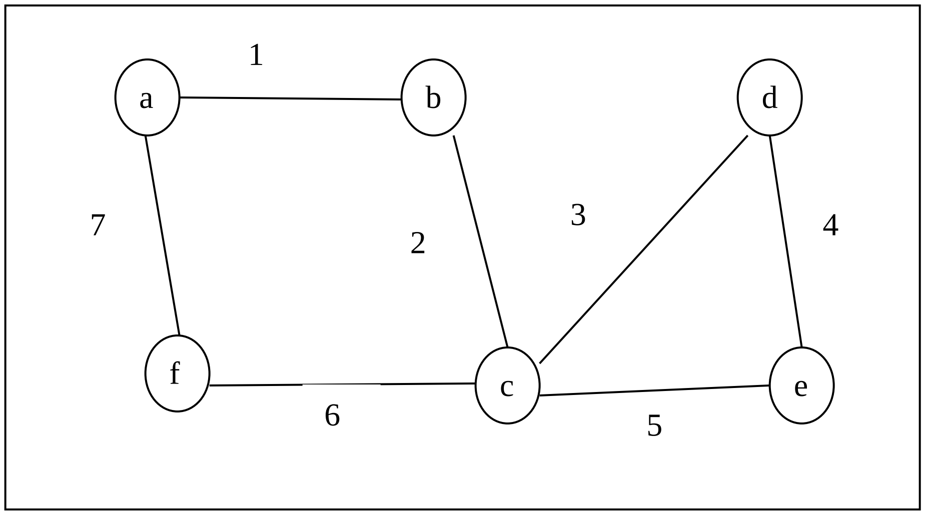

Figure 4.18 Weighted Graph

Isomorphic Graphs- If two graphs G and H contain the same number of vertices connected in the same way, they are called isomorphic graphs (denoted by $G \cong H$).

It is easier to check non-isomorphism than isomorphism. If any of these following conditions occurs, then two graphs are non-isomorphic –

i. The number of connected components are different
ii. Vertex-set cardinalities are different
iii. Edge-set cardinalities are different
iv. Degree sequences are different

Example

The following graphs are isomorphic –

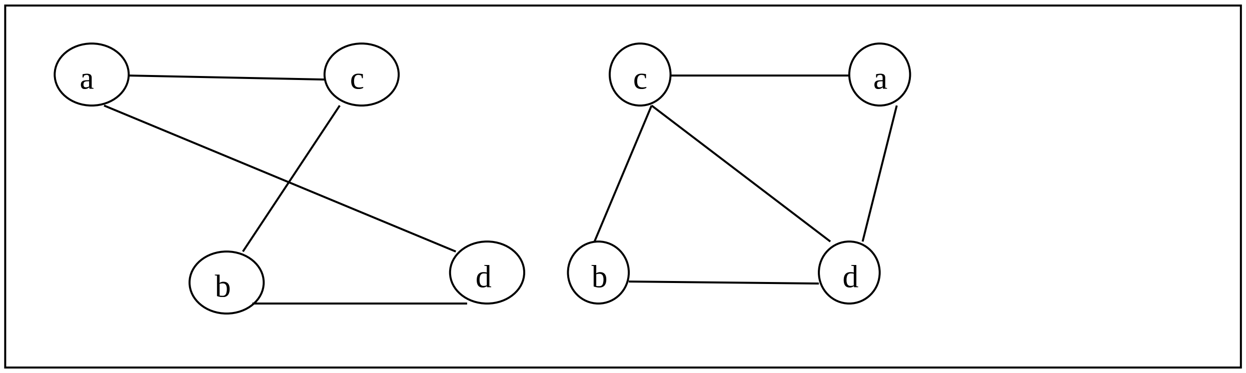

Figure 4.19 Isomorphic Graphs

Path - A path is a sequence of vertices such that each vertex is adjacent to the next. In a path, each edge can be traveled **only once**. The length of a path is the number of edges in that path.

Cycle - A path that starts and ends at the same vertex is called a circuit or Cycle.

Connectivity - A graph is *connected* if any two vertices can be joined by a path. If this is not possible then the graph is disconnected.

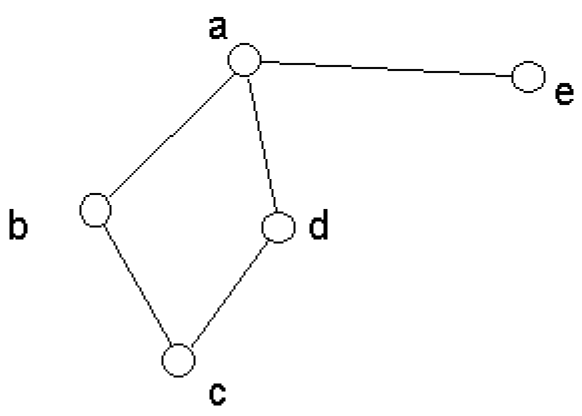

Figure 4.20 Path, Cycle, Connectivity

Shortest Path in Weighted Graph

Dijkstra algorithm-

1. It maintains a list of unvisited vertices.
2. It chooses a vertex (the source) and assigns a maximum possible cost (i.e. infinity) to every other vertex.
3. The cost of the source remains zero as it actually takes nothing to reach from the source vertex to itself.
4. In every subsequent step of the algorithm it tries to improve(minimize) the cost for each vertex. Here the cost can be distance, money or time taken to reach that vertex from the source vertex. The minimization of cost is a multi-step process.
    a. For each unvisited neighbor (vertex 2, vertex 3, vertex 4) of the current vertex (vertex 1) calculate the new cost from the vertex (vertex 1).
    b. For e.g. the new cost of vertex 2 is calculated as the minimum of the two ((existing cost of vertex 2) or (sum of cost of vertex 1 + the cost of edge from vertex 1 to vertex 2))
5. When all the neighbors of the current node are considered, it marks the current node as visited and is removed from the unvisited list.

6. Select a vertex from the list of unvisited nodes (which has the smallest cost) and repeat step 4.
7. At the end there will be no possibilities to improve it further and then the algorithm ends

For demonstration we will consider the below graph:

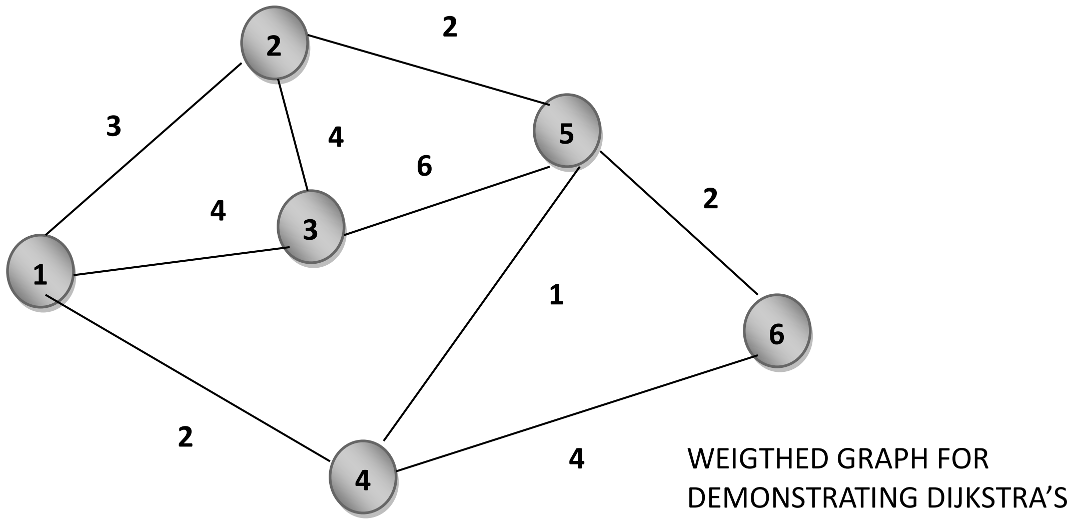

Figure 4.21 Weighted Graph

Step 1:Mark Vertex 1 as the source vertex. Assign a cost zero to Vertex 1 and (infinite to all other vertices). The state is as follows:

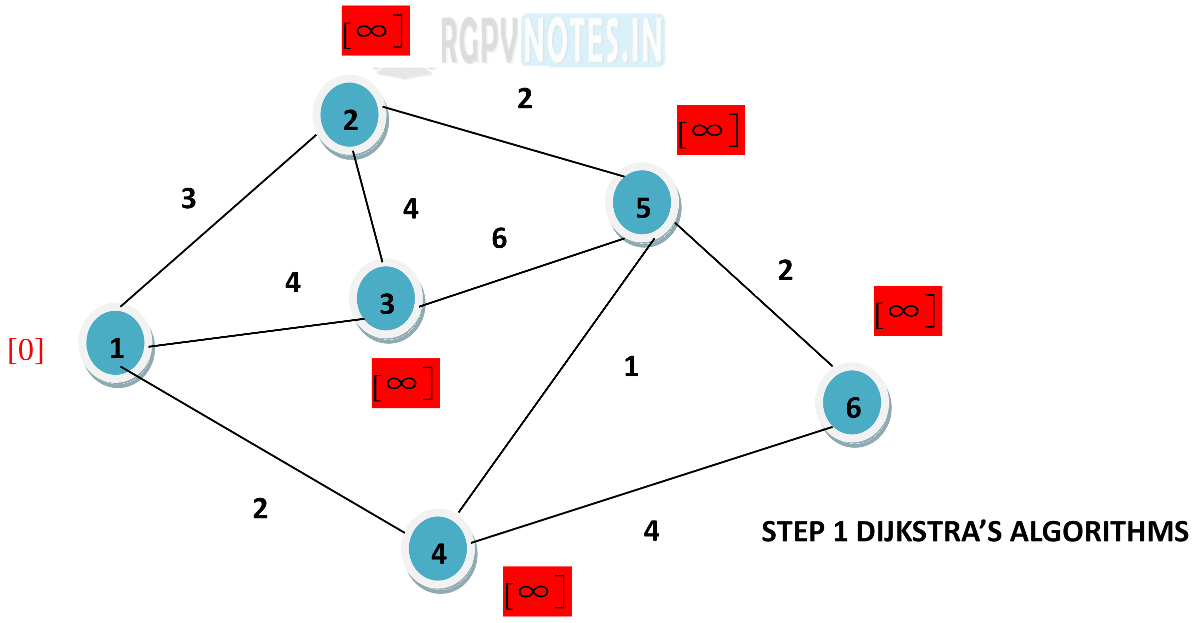

Figure 4.21(a) Step 1

Step 2: For each of the unvisited neighbors (Vertex 2, Vertex 3 and Vertex 4) calculate the minimum cost as min(current cost of vertex under consideration, sum of cost of vertex 1 and connecting edge). Mark Vertex 1 as visited , in the diagram we border it black. The new state would be as follows:

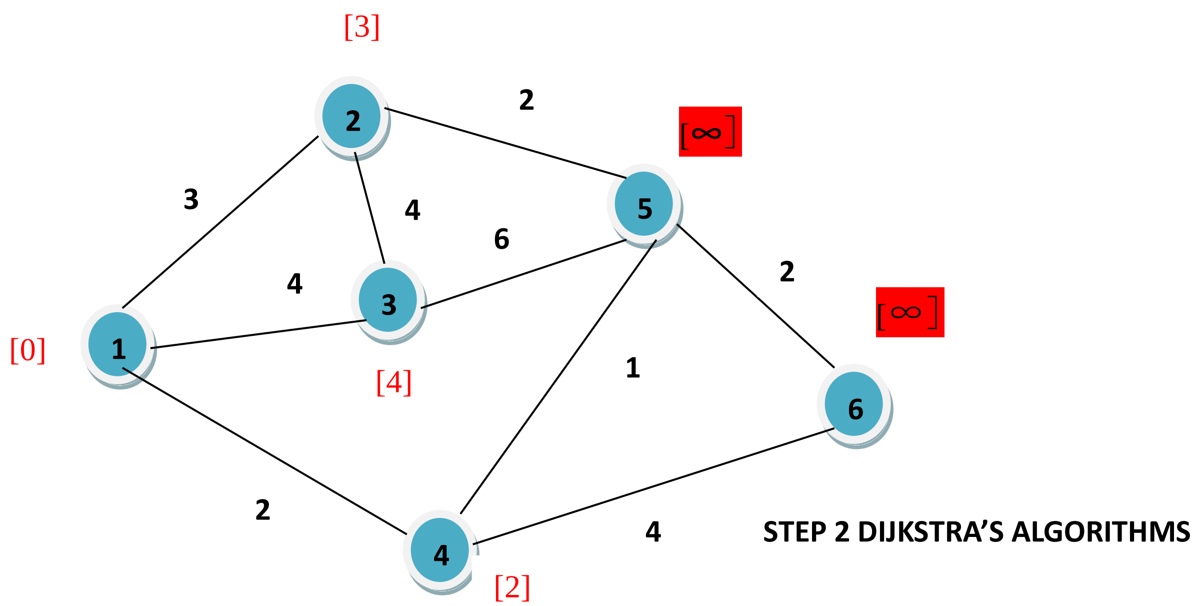

Figure 4.21(b) Step 2

Step 3: Choose the unvisited vertex with minimum cost (vertex 4) and consider all its unvisited neighbors (Vertex 5 and Vertex 6) and calculate the minimum cost for both of them. The state is as follows:

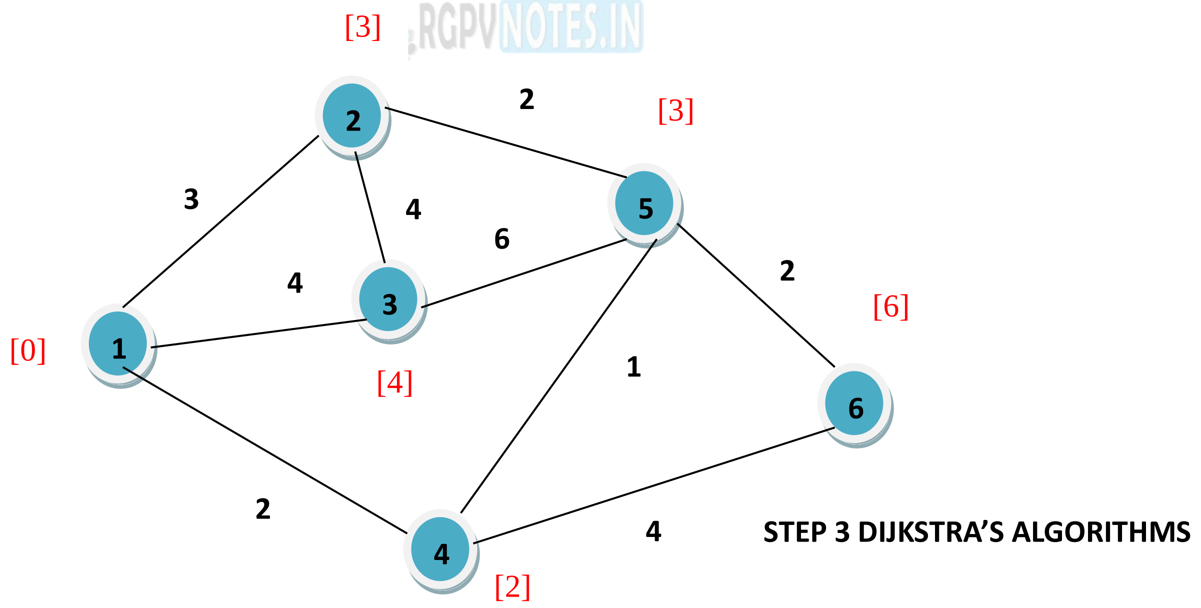

Figure 4.21(c) Step 3

Step 4: Choose the unvisited vertex with minimum cost (vertex 2 or vertex 5, here we choose vertex 2) and consider all its unvisited neighbors (Vertex 3 and Vertex 5) and calculate the minimum cost for both of them. Now, the current cost of Vertex 3 is [4] and the sum of (cost of Vertex 2 + cost of edge (2,3)) is 3 + 4 = [7]. Minimum of 4, 7 is 4. Hence the cost of vertex 3 won't change. By the same argument the cost of vertex 5 will not change. We just mark the vertex 2 as visited, all the costs remain same. The state is as follows:

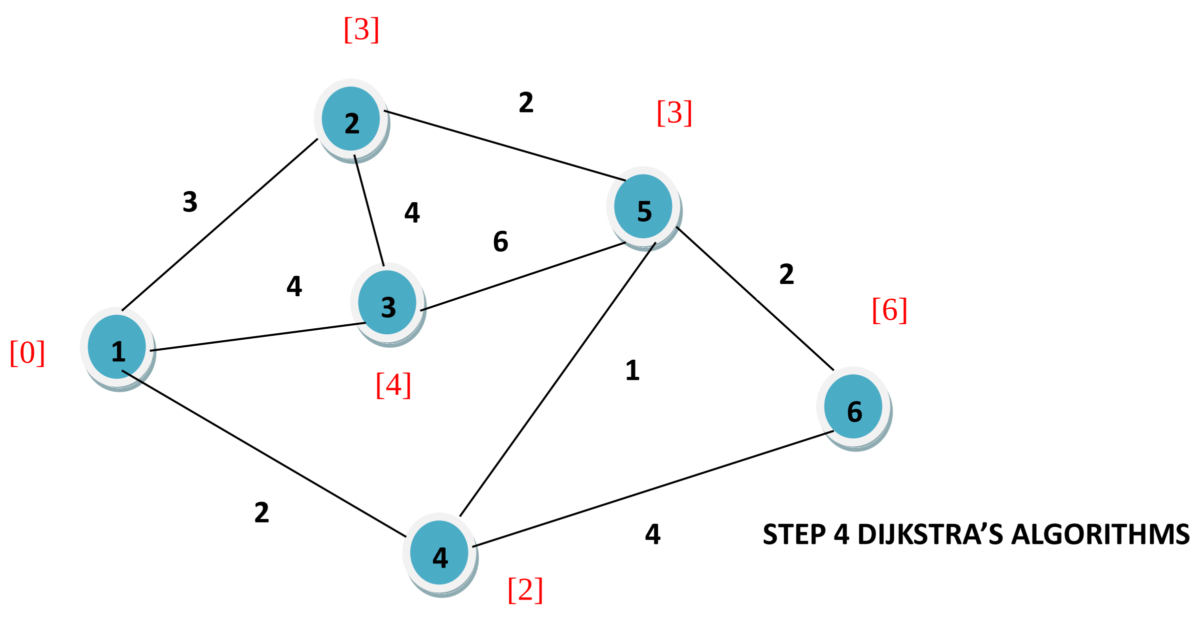

Figure 4.21(d) Step 4

Step 5: Choose the unvisited vertex with minimum cost (vertex 5) and consider all its unvisited neighbors (Vertex 3 and Vertex 6) and calculate the minimum cost for both of them. Now, the current cost of Vertex 3 is [4] and the sum of (cost of Vertex 5 + cost of edge (5,3)) is 3 + 6 = [9]. Minimum of 4, 9 is 4. Hence the cost of vertex 3 won't change. Now, the current cost of Vertex 6 is [6] and the sum of (cost of Vertex 5 + cost of edge (3,6)) is 3 + 2 = [5]. Minimum of 6, 5 is 45. Hence the cost of vertex 6 changes to 5. The state is as follows:

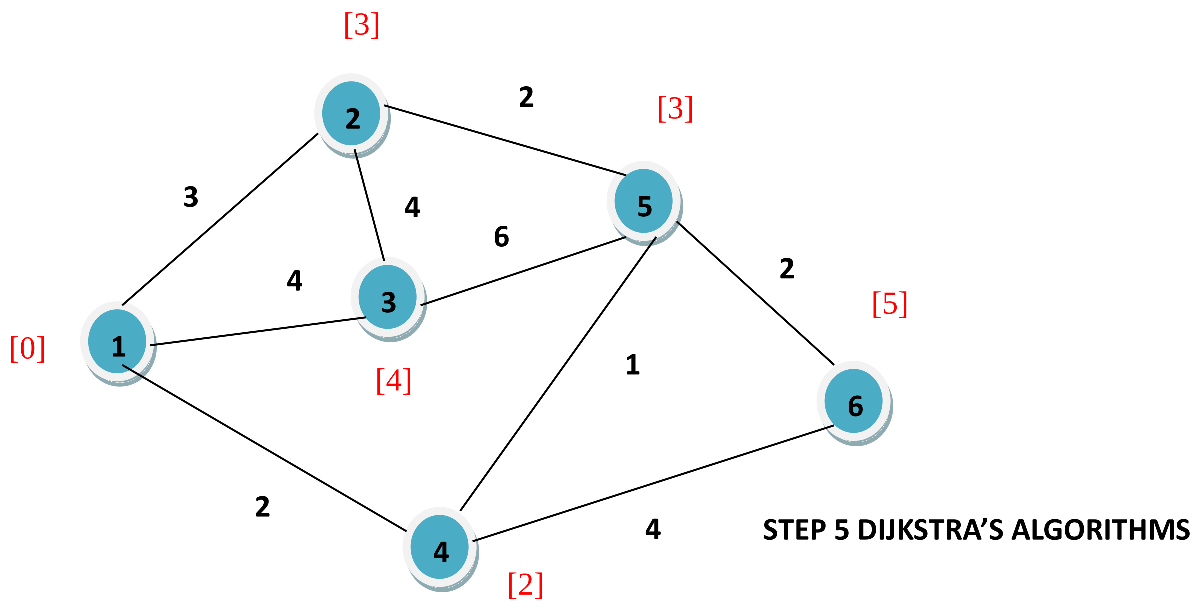

Figure 4.21(e) Step 5

Step 6: Choose the unvisited vertex with minimum cost (vertex 3) and consider all its unvisited neighbors (none). So mark it visited. The state is as follows:

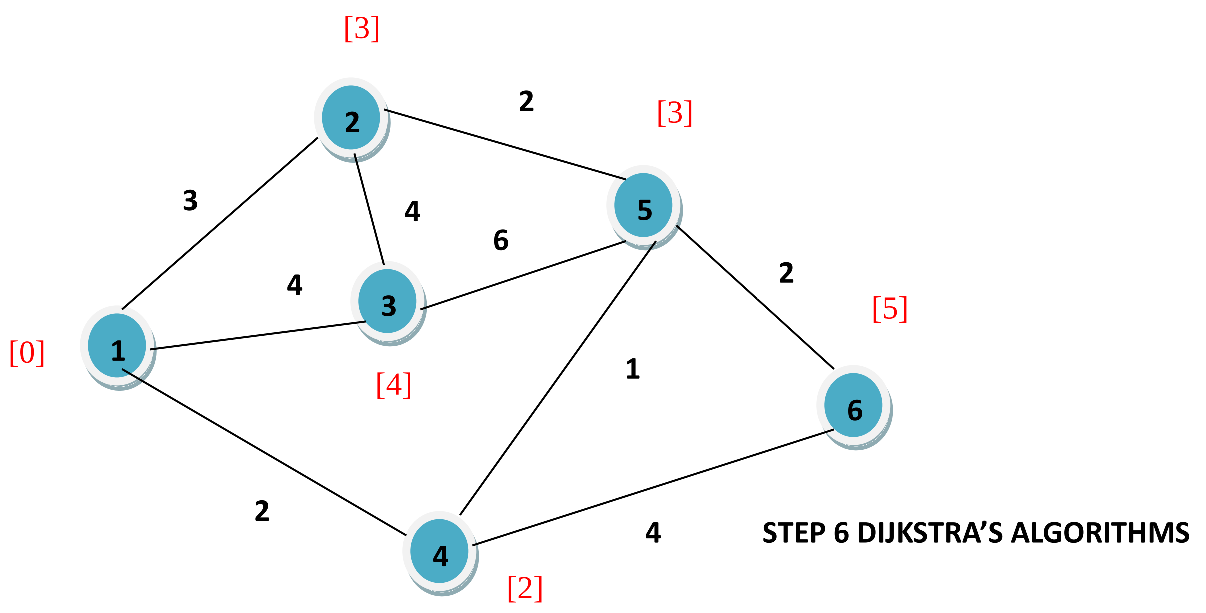

Figure 4.21(f) Step 6

Step 7: Choose the unvisited vertex with minimum cost (vertex 6) and consider all its unvisited neighbors (none). So mark it visited. The state is as follows:

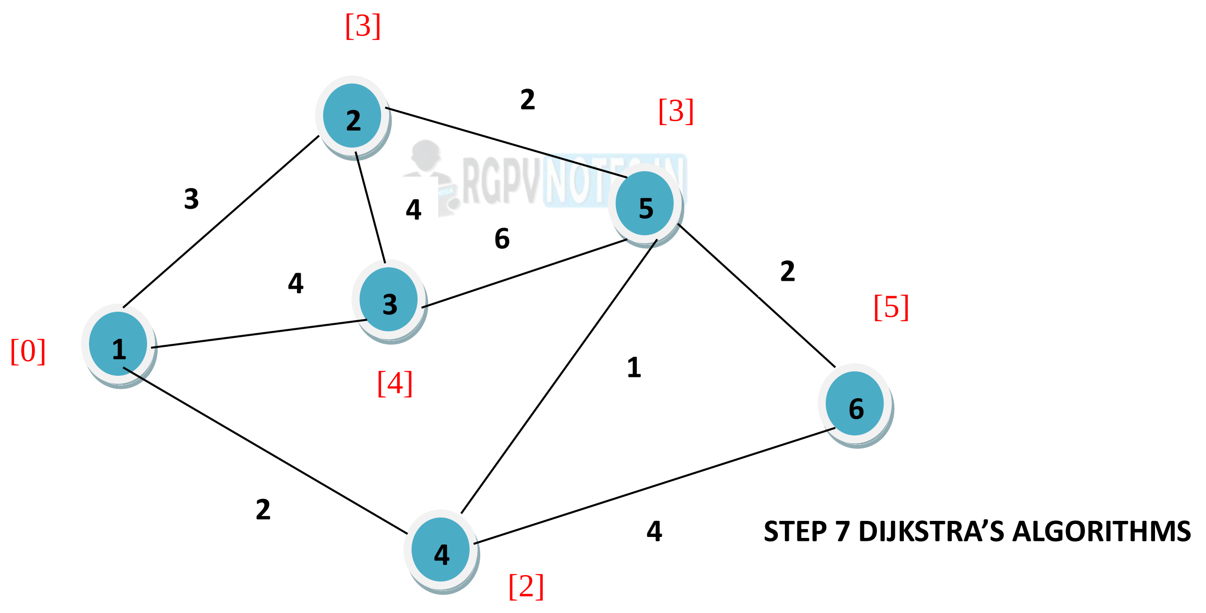

Figure 4.21(g) Step 7

Now there is no unvisited vertex left and the execution ends. At the end we know the shortest paths for all the vertices from the source vertex 1. Even if we know the shortest path length, we do not know the exact list of vertices which contributes to the shortest path until we maintain them separately or the data structure supports it.

Numerical

Find the shortest path between a to z using Dijkstra's algorithm.

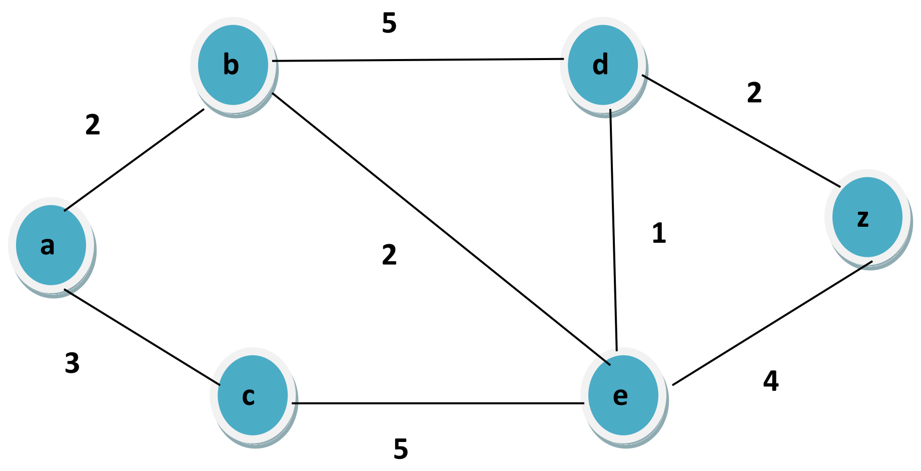

Figure 4.22 Example to find shortest path

Solution

Consider the graph G=(V,E), Where V is the set of Vertices and E is the set of Edges.

V= {a,b,c,d,e,z}

Step -1 P1 ={a} **T1= V-P1**

T1 = {b,c,d,e,z}

l(b) =2, l(c)=3, l(d)= ∞ , l(e)= ∞, l(z)= ∞

b has the minimum index 2.

Step-2 P2 ={a,b} **T2= V-P2**

T2= {c,d,e,z}

l(c)= min(3, 2+∞) =3

l(d)= min(∞, 2+5)=7

l(e)= min(∞, 2+2)=4

l(z)= min (∞, 2+∞)=∞

c has the minimum index 3.

Step-3 P3 ={a,b,c} **T3= V-P3**

T3={d,e,z}

l(d)=min{7, 3+5} = 7

l(e)=min(4, 4+1} = 4

l(z)=min(∞, 3+∞}=∞

e has the minimum index 4.

Step-4 P4 ={a,b,c,e} **T4= V-P4**

T4={d,z}

l(d)=min{7, 4+1} = 5

l(z)=min(∞, 4+4}=8

d has the minimum index 5.

Step-5 P5 ={a,b,c,e,d} **T5= V-P5**

T5={z}

l(z)=min(8, 5+2}=7

Hence the minimum distance from the source a to destination z is 7.

The Shortest path is **a -> b -> e -> d ->z**

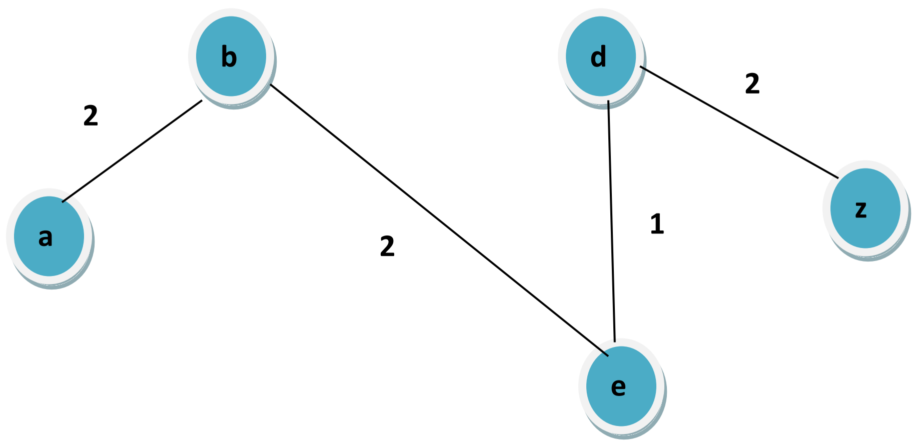

Figure 4.22 (a) Solution of shortest path

Eulerian Path and Circuits - A connected graph ***G*** is called an Euler graph, if there is a closed trail which includes every edge of the graph ***G*** . An Euler path is a path that uses every edge of a graph exactly once. An Euler path starts and ends at different vertices. An Euler circuit is a circuit that uses every edge of a graph exactly once. An Euler circuit always starts and ends at the same vertex. A connected graph ***G*** is an Euler graph if and only if all vertices of ***G*** are of even degree, and a connected graph ***G*** is Eulerian if and only if its edge set can be decomposed into cycles.

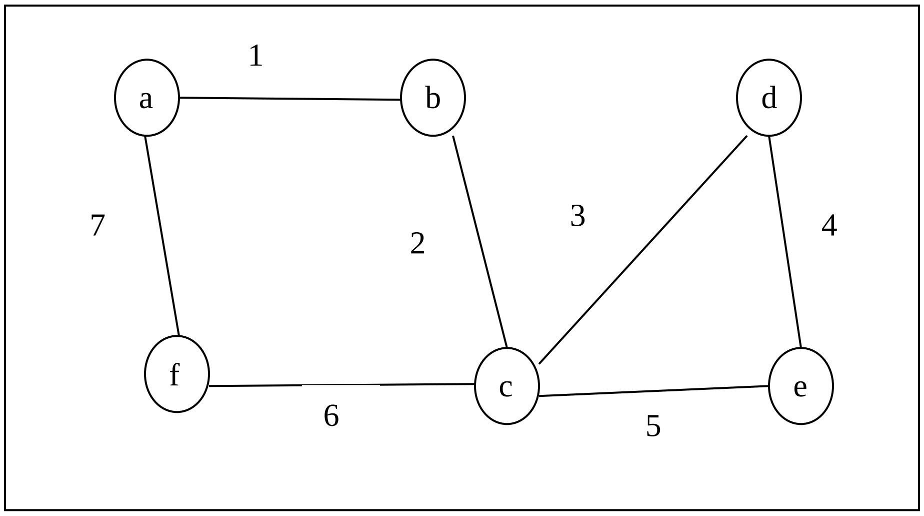

Figure 4.23 Eulerian Path and Circuits

The above graph is an Euler graph as "*a*1*b*2*c*3*d*4*e*5*c*6*f*7*g*" covers all the edges of the graph.

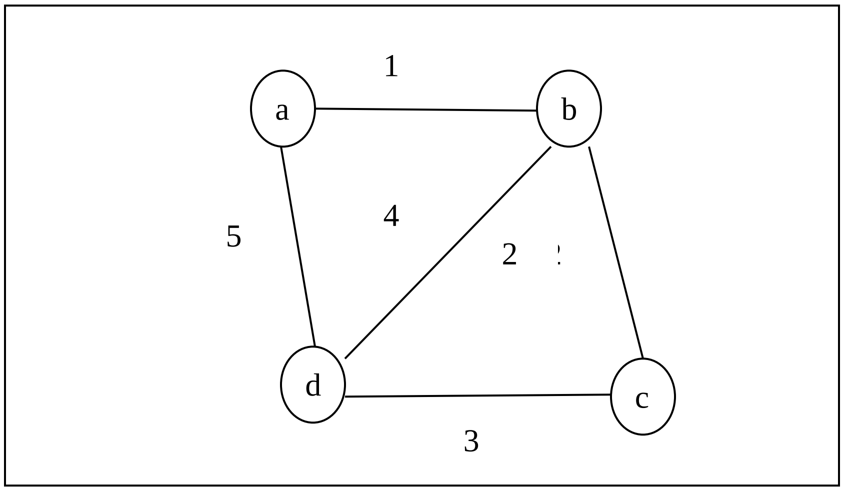

Figure 4.24 Eular Graph

Hamiltonian Paths and circuits - A connected graph ***G*** is called Hamiltonian graph if there is a cycle which includes every vertex of ***G*** and the cycle is called Hamiltonian cycle. Hamiltonian walk in graph ***G*** is a walk that passes through each vertex exactly once. If ***G*** is a simple graph with n vertices, where ***n***≥3 If ***deg***(***v***)≥***n***2 for each vertex ***v***, then the graph ***G*** is Hamiltonian graph. This is called Dirac's Theorem. If ***G*** is a simple graph with ***n*** vertices, where ***n***≥2 if ***deg***(***x***)+***deg***(***y***)≥***n*** for each pair of non-adjacent vertices x and y, then the graph ***G*** is Hamiltonian graph. This is called Ore's theorem.

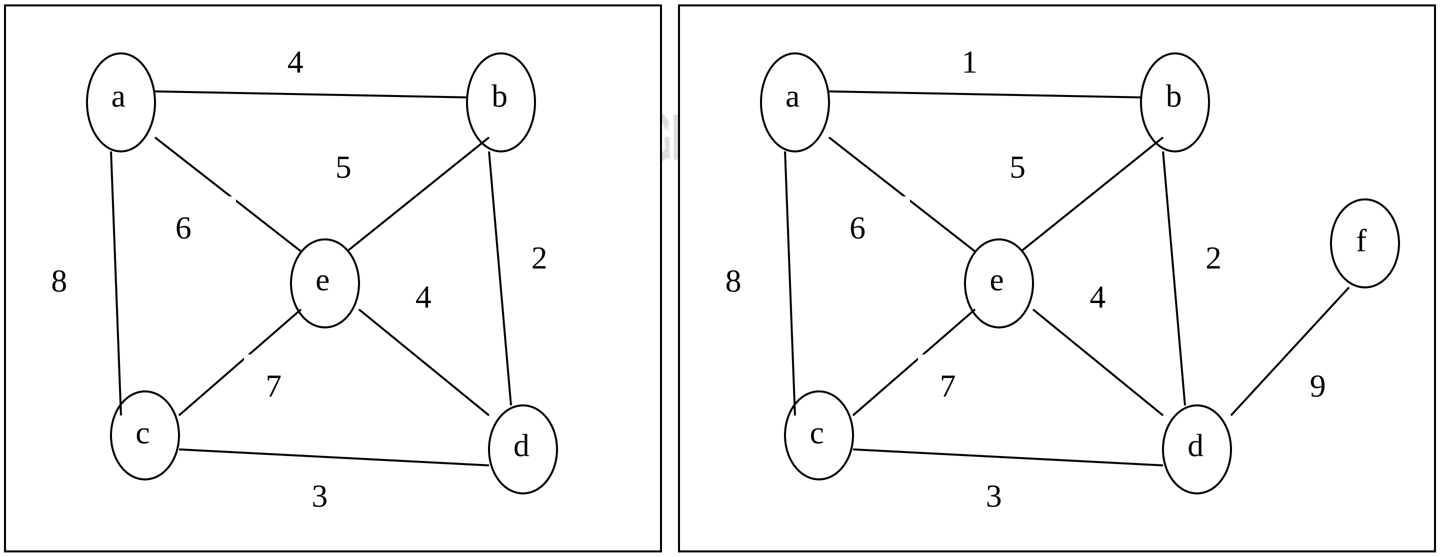

Figure 4.25 Hamiltonian Paths and circuits

Graph Coloring - Graph coloring is the procedure of assignment of colors to each vertex of a graph G such that no adjacent vertices get same color. The objective is to minimize the number of colors while coloring a graph. The smallest number of colors required to color a graph G is called its chromatic number of that graph. Graph coloring problem is a NP Complete problem.

Method to Color a Graph

The steps required to color a graph G with n number of vertices are as follows −

Step 1 − Arrange the vertices of the graph in some order.

Step 2 − Choose the first vertex and color it with the first color.

Step 3 − Choose the next vertex and color it with the lowest numbered color that has not been colored on any vertices adjacent to it. If all the adjacent vertices are colored with this color, assign a new color to it. Repeat this step until all the vertices are colored.

Example

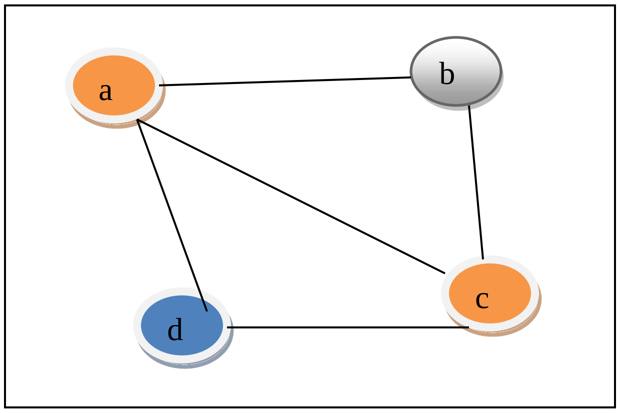

Figure 4.26 Graph Coloring

In the above figure, at first vertex *a* is colored red. As the adjacent vertices of vertex a are again adjacent, vertex *b* and vertex *d* are colored with different color, green and blue respectively. Then vertex *c* is colored as red as no adjacent vertex of *c* is colored red. Hence, we could color the graph by 3 colors. Hence, the chromatic number of the graph is 3.

Chromatic Number- The chromatic number of a graph G is the smallest number of colors needed to color the vertices of G so that no two adjacent vertices share the same color, i.e., the smallest value of k possible to obtain a k-coloring.

<u>**Isomorphism**</u> **-** If two graphs G and H contain the same number of vertices connected in the same way, they are called isomorphic graphs (denoted by ***G≅H***).
It is easier to check non-isomorphism than isomorphism. If any of these following conditions occurs, then two graphs are non-isomorphic –

v. The number of connected components are different
vi. Vertex-set cardinalities are different
vii. Edge-set cardinalities are different
viii. Degree sequences are different

Example

The following graphs are isomorphic –

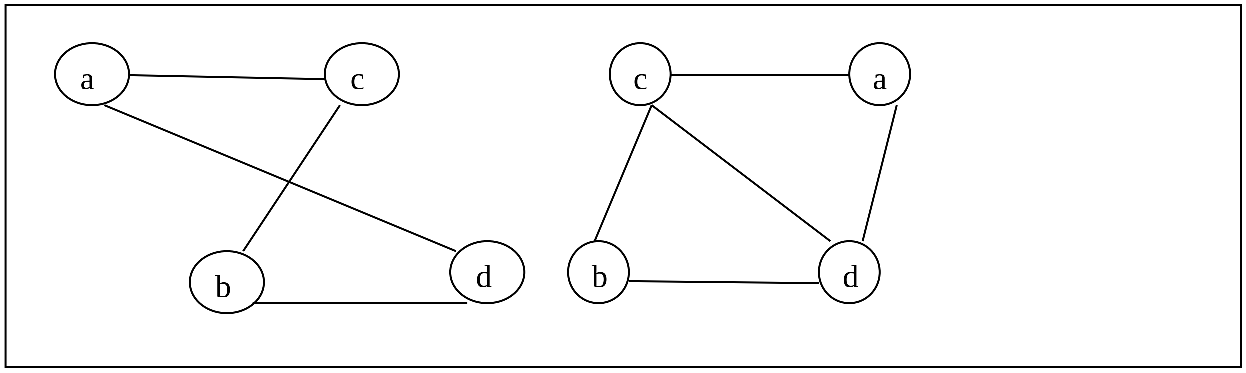

Figure 4.27 Isomorphism

<u>**Homomorphism**</u> **-** A homomorphism from a graph ***G*** to a graph ***H*** is a mapping (May not be a bijective mapping)***h***:***G***→***H*** such that – **(*x*,*y*)∈*E*(*G*)→(*h*(*x*),*h*(*y*))∈*E*(*H*)**. It maps adjacent vertices of graph ***G*** to the adjacent vertices of the graph ***H.***

Properties of Homomorphisms

i. A homomorphism is an isomorphism if it is a bijective mapping.

ii. Homomorphism always preserves edges and connectedness of a graph.
iii. The compositions of homomorphisms are also homomorphisms.