Discrete Mathematics 8
Graph Theory

Dr. Maged Kassab

Graph Theory Basics

A graph is a data structure that is defined by two components :

- A **node** or a vertex.
- An edge E or ordered pair is a connection between two nodes **u,v** that is identified by unique pair(u,v). The pair (u,v) is ordered because (u,v) is not same as (v,u) in case of directed graph. The edge may have a weight or is set to one in case of un weighted graph.

Graph Theory Basics

Components of a Graph

•**Vertices:** Vertices are the fundamental units of the graph. Sometimes, vertices are also known as vertex or nodes. Every node/vertex can be labeled or unlabelled.

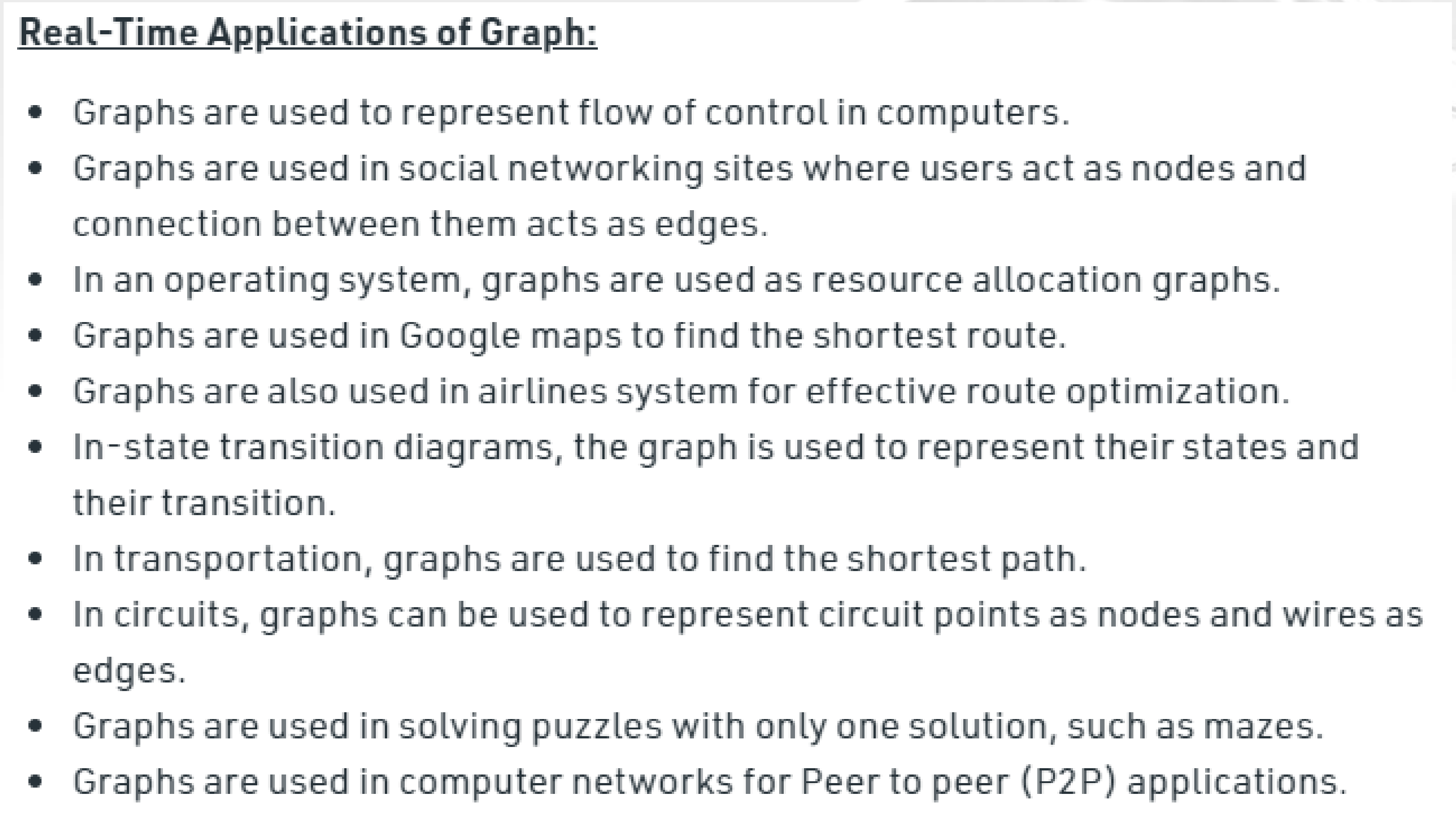
•**Edges:** Edges are drawn or used to connect two nodes of the graph. It can be ordered pair of nodes in a directed graph. Edges can connect any two nodes in any possible way. There are no rules. Sometimes, edges are also known as arcs. Every edge can be labeled/unlabelled.

Graph Theory Basics

A graph G=(V, E) is a structure consisting of a set of objects called vertices V and a set of objects called edges E. An edge e ∊ E is denoted in the form e={x,y}, where the vertices x,y ∊ V. Two vertices x and y connected by the edge e={x,y}, are said to be adjacent , with x and y, called the endpoints.

A simple graph is one in which there are no loops (edges joining the same vertex) and no two edges joining the same pair of vertices. The graphs we consider are assumed to be simple unless stated otherwise.

Graph Theory Basics



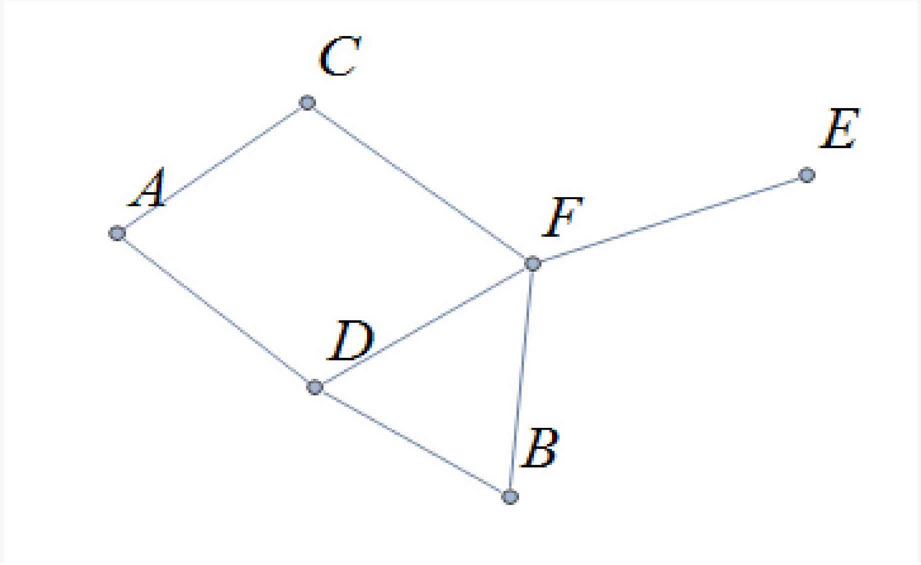
Real-Time Applications of Graph:

- Graphs are used to represent flow of control in computers.
- Graphs are used in social networking sites where users act as nodes and connection between them acts as edges.
- In an operating system, graphs are used as resource allocation graphs.
- Graphs are used in Google maps to find the shortest route.
- Graphs are also used in airlines system for effective route optimization.
- In-state transition diagrams, the graph is used to represent their states and their transition.
- In transportation, graphs are used to find the shortest path.
- In circuits, graphs can be used to represent circuit points as nodes and wires as edges.
- Graphs are used in solving puzzles with only one solution, such as mazes.
- Graphs are used in computer networks for Peer to peer (P2P) applications.

Graph Theory Basics

Example 1 - an undirected graph.

The graph shown has vertex set $V = \{A,\ B,\ C,\ D,\ E,\ F\}$ and edge set $E = \{\{A, C\}, \{A, D\}, \{B, D\}\{B, F\}, \{C, F\}, \{D, F\}, \{F, E\}\}$

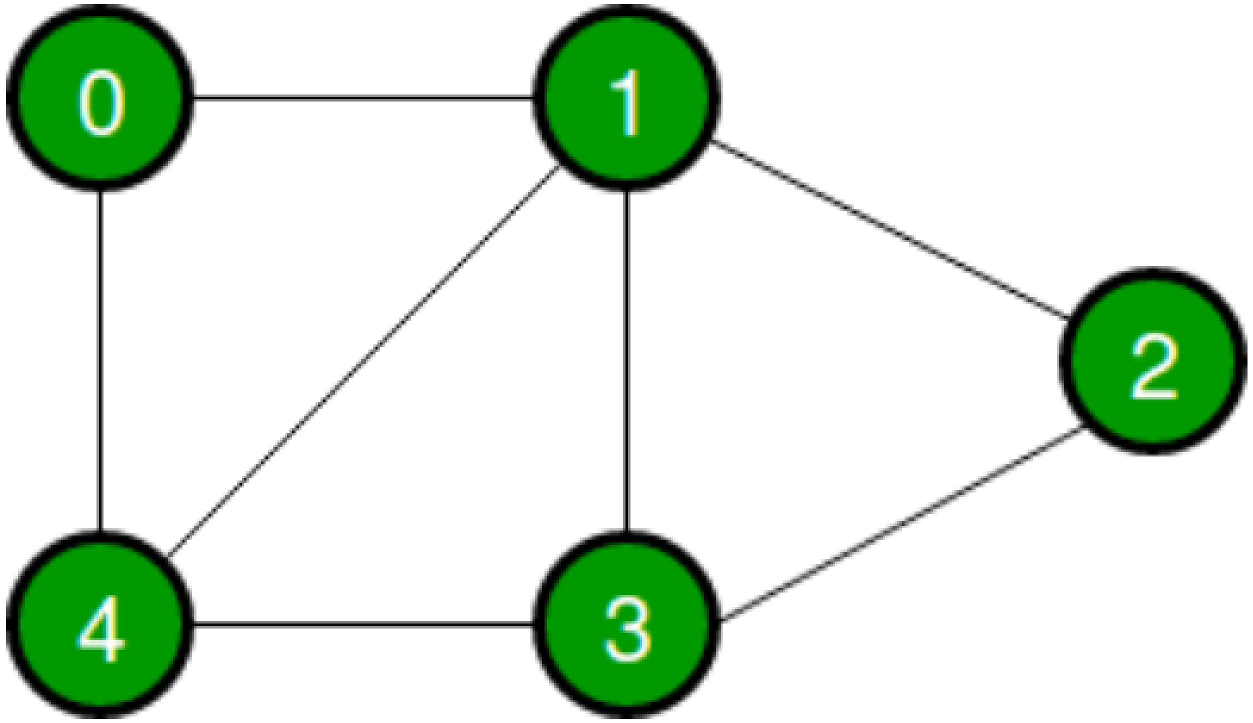


Graph Theory Basics

Graphs are used to represent many real-life applications: Graphs are used to represent networks. The networks may include paths in a city or telephone network or circuit network. Graphs are also used in social networks like linkedIn, Facebook. For example, in Facebook, each person is represented with a vertex(or node). Each node is a structure and contains information like person id, name, gender, and locale. See this for more applications of graph.

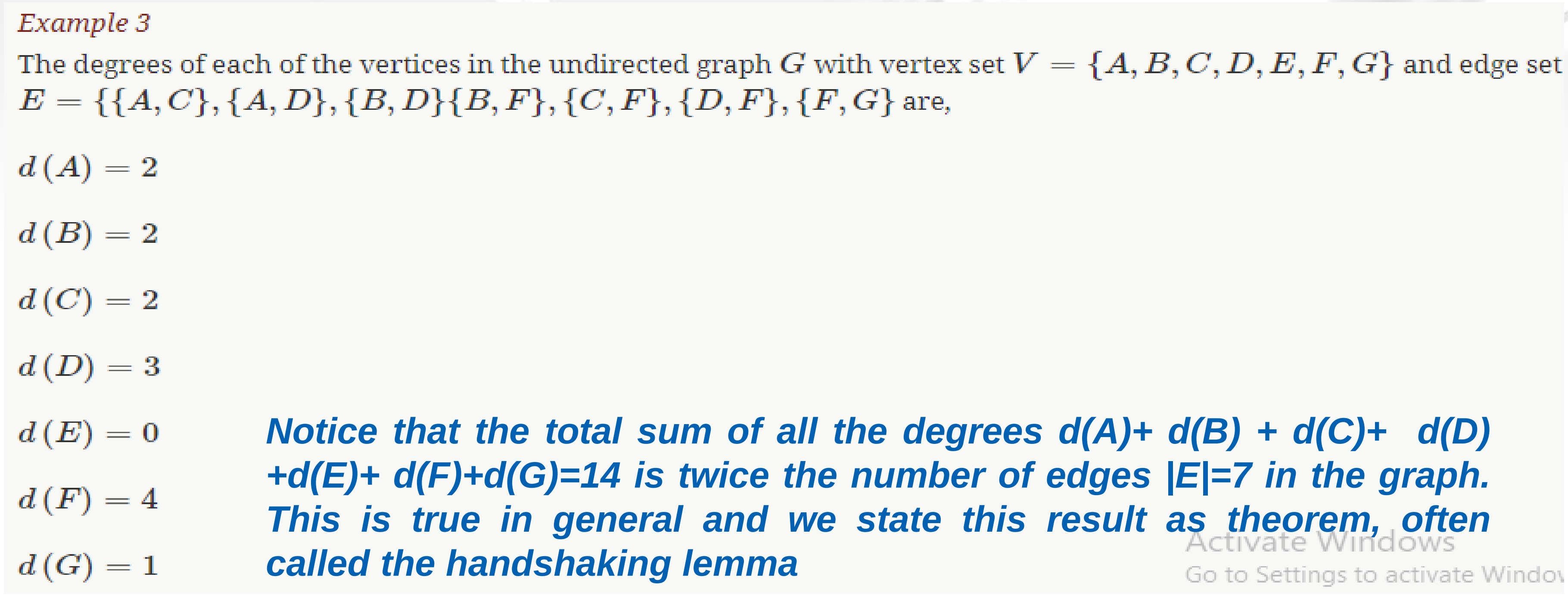
Graph Theory Basics

Following is an example of an undirected graph with 5 vertices.



Graph Theory Basics

The ***degree of a vertex*** v ∈ V, denoted d(v), is the number of edges in the graph G containing the vertex v.



Example 3

The degrees of each of the vertices in the undirected graph G with vertex set $V = \{A, B, C, D, E, F, G\}$ and edge set $E = \{\{A, C\}, \{A, D\}, \{B, D\}\{B, F\}, \{C, F\}, \{D, F\}, \{F, G\}$ are,

$d(A) = 2$

$d(B) = 2$

$d(C) = 2$

$d(D) = 3$

$d(E) = 0$

$d(F) = 4$

$d(G) = 1$

Notice that the total sum of all the degrees d(A)+ d(B) + d(C)+ d(D) +d(E)+ d(F)+d(G)=14 is twice the number of edges |E|=7 in the graph. This is true in general and we state this result as theorem, often called the handshaking lemma

Graph Theory Basics

The following two are the most commonly used representations of a graph.

1. Adjacency Matrix

2. Adjacency List

There are other representations also like, Incidence Matrix and Incidence List. The choice of graph representation is situation-specific. It totally depends on the type of operations to be performed and ease of use.

Graph Theory Basics

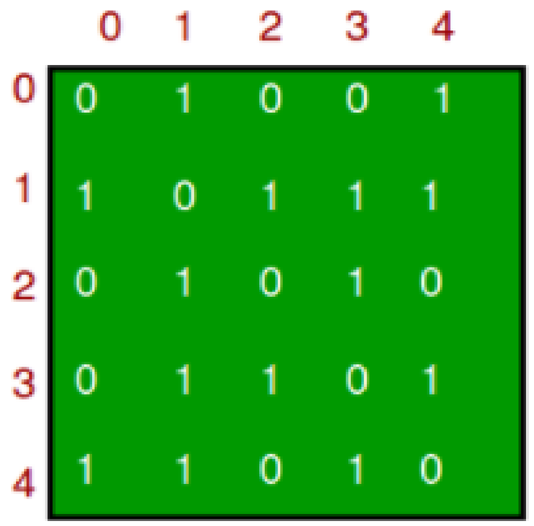
Adjacency Matrix:

Adjacency Matrix is a 2D array of size V x V where V is the number of vertices in a graph. Let the 2D array be adj[][], a slot adj[i][j] = 1 indicates that there is an edge from vertex i to vertex j. Adjacency matrix for undirected graph is always symmetric. Adjacency Matrix is also used to represent weighted graphs. If adj[i][j] = w, then there is an edge from vertex i to vertex j with weight w.

Graph Theory Basics

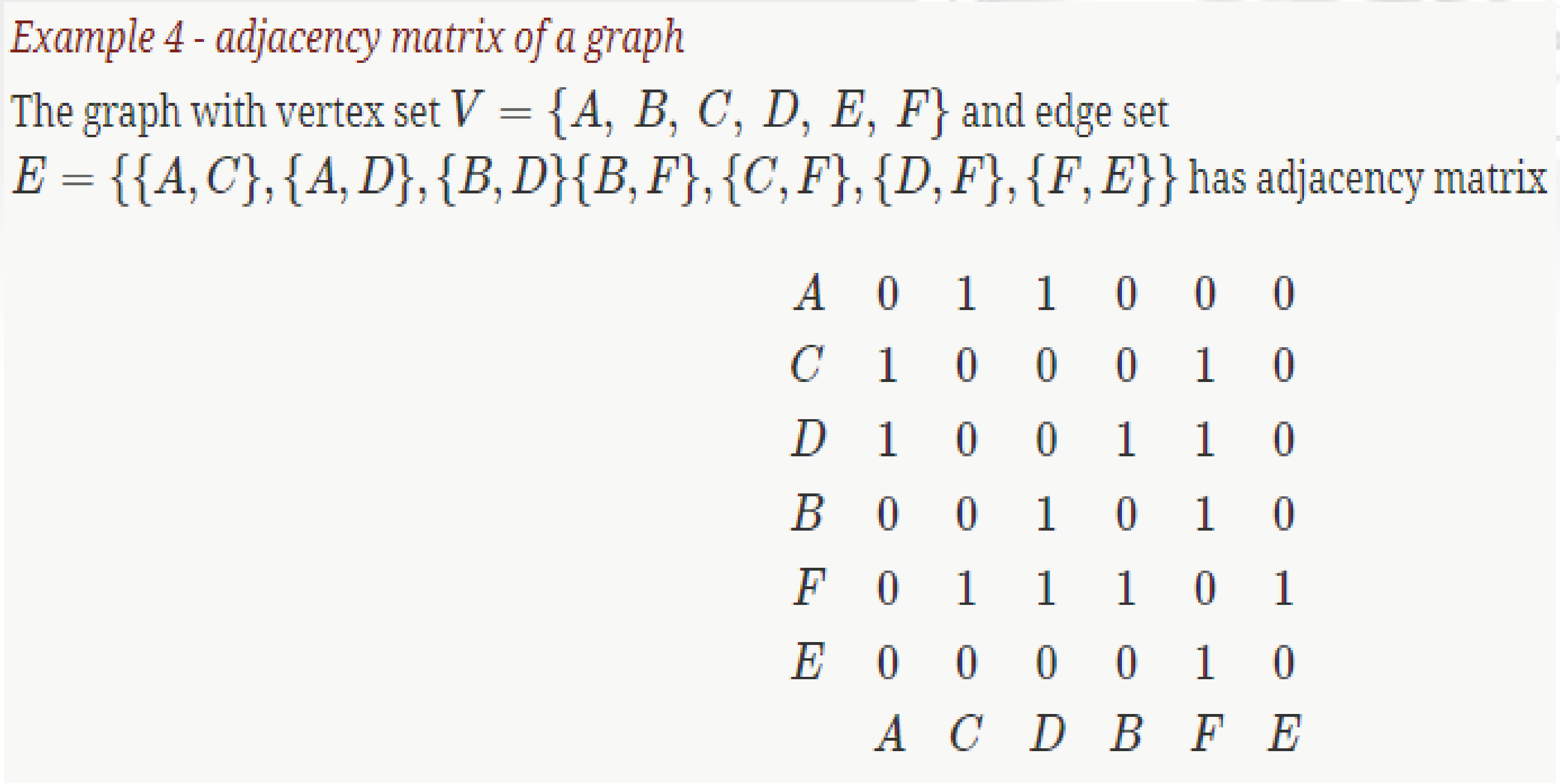
In case of a directed graph, if there is an edge from vertex i to vertex j then we just assign adj[i][j]=1

The adjacency matrix for the above example graph is:



$$\begin{array}{c|ccccc}
 & 0 & 1 & 2 & 3 & 4 \\
\hline
0 & 0 & 1 & 0 & 0 & 1 \\
1 & 1 & 0 & 1 & 1 & 1 \\
2 & 0 & 1 & 0 & 1 & 0 \\
3 & 0 & 1 & 1 & 0 & 1 \\
4 & 1 & 1 & 0 & 1 & 0
\end{array}$$

Graph Theory Basics



Example 4 - adjacency matrix of a graph

The graph with vertex set $V = \{A,\ B,\ C,\ D,\ E,\ F\}$ and edge set $E = \{\{A, C\}, \{A, D\}, \{B, D\}\{B, F\}, \{C, F\}, \{D, F\}, \{F, E\}\}$ has adjacency matrix

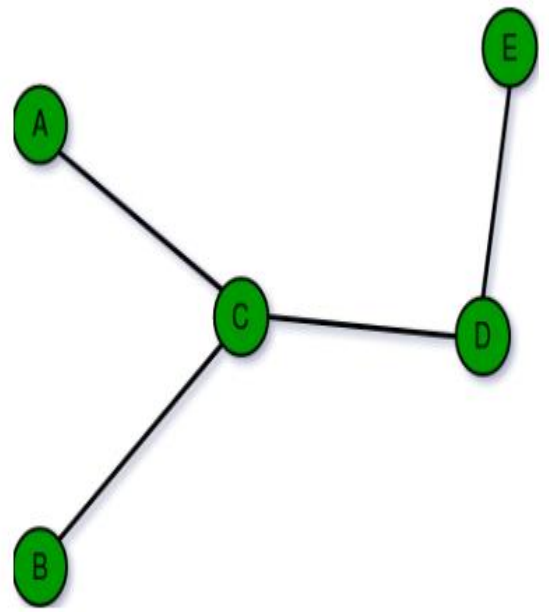
$$\begin{array}{ccccccc}
A & 0 & 1 & 1 & 0 & 0 & 0 \\
C & 1 & 0 & 0 & 0 & 1 & 0 \\
D & 1 & 0 & 0 & 1 & 1 & 0 \\
B & 0 & 0 & 1 & 0 & 1 & 0 \\
F & 0 & 1 & 1 & 1 & 0 & 1 \\
E & 0 & 0 & 0 & 0 & 1 & 0 \\
 & A & C & D & B & F & E
\end{array}$$

Graph Theory Basics

Types of Graphs:

1. Finite Graphs

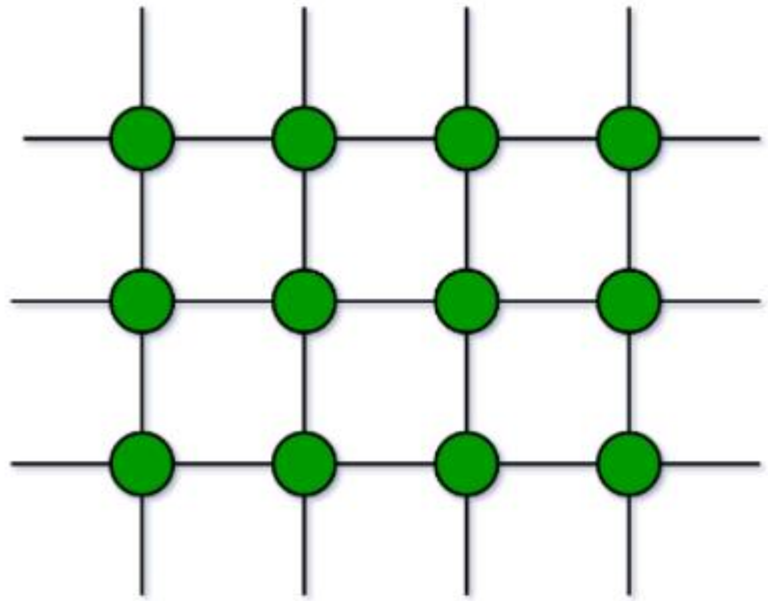
A graph is said to be finite if it has a finite number of vertices and a finite number of edges. A finite graph is a graph with a finite number of vertices and edges. In other words, both the number of vertices and the number of edges in a finite graph are limited and can be counted. Finite graphs are often used to model real-world situations, where there is a limited number of objects and relationships between the



Types of Graphs:

2. Infinite Graph:

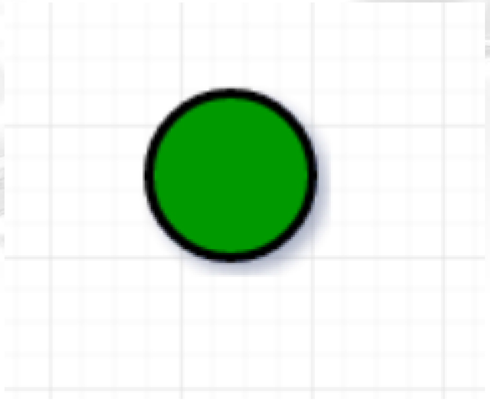
A graph is said to be infinite if it has an infinite number of vertices as well as an infinite number of edges.



Types of Graphs:

3. Trivial Graph:

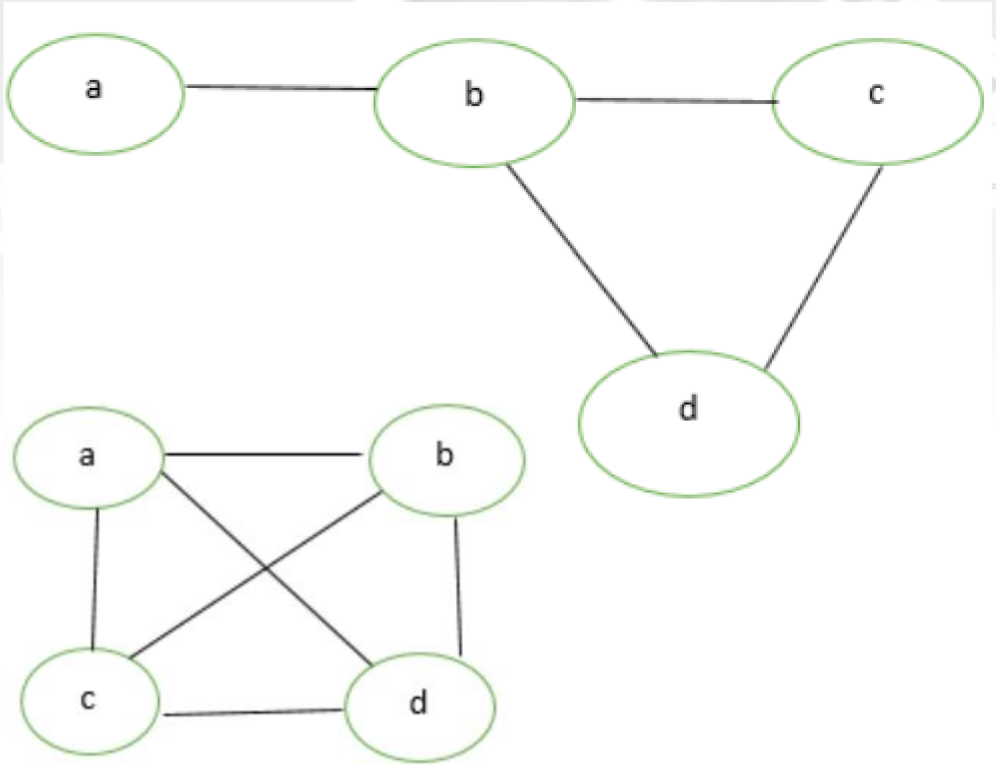
A graph is said to be trivial if a finite graph contains only one vertex and no edge. A trivial graph is a graph with only one vertex and no edges. A trivial graph is the simplest type of graph and is often used as a starting point for building more complex graphs.



Types of Graphs:

4. Simple Graph:

A simple graph is a graph that does not contain more than one edge between the pair of vertices. A simple railway track connecting different cities is an example of a simple graph.



Types of Graphs:

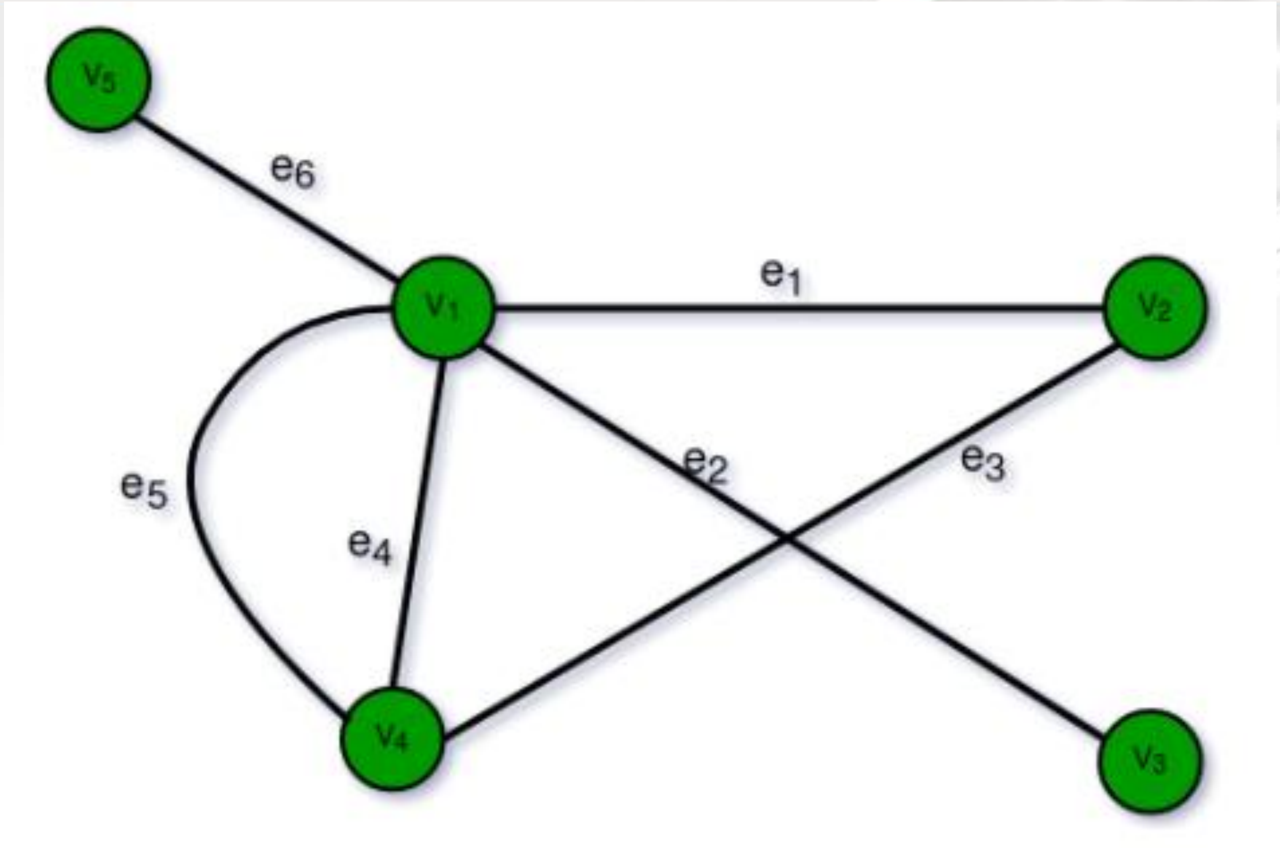
5. Multi Graph:

Any graph which contains some parallel edges but doesn’t contain any self-loop is called a multi graph. For example a Road Map.

1. **Parallel Edges:** If two vertices are connected with more than one edge then such edges are called parallel edges that are many routes but one destination.

2. **Loop:** An edge of a graph that starts from a vertex and ends at the same vertex is called a loop or a self-loop.

Types of Graphs:



Types of Graphs:

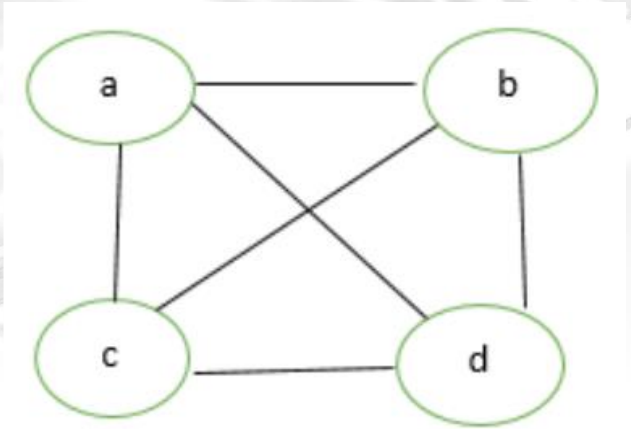
6. Null Graph:

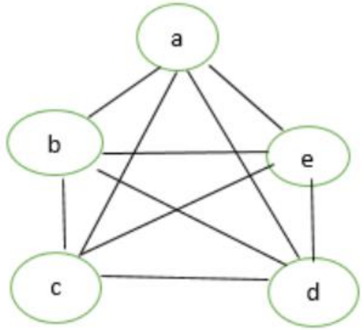
A graph of order n and size zero is a graph where there are only isolated vertices with no edges connecting any pair of vertices. A null graph is a graph with no edges. In other words, it is a graph with only vertices and no connections between them. A null graph can also be referred to as an edgeless graph, an isolated graph, or a discrete graph

Types of Graphs:

7. Complete Graph:

A simple graph with n vertices is called a complete graph if the degree of each vertex is n-1, that is, one vertex is attached with n-1 edges or the rest of the vertices in the graph. A complete graph is also called Full Graph.

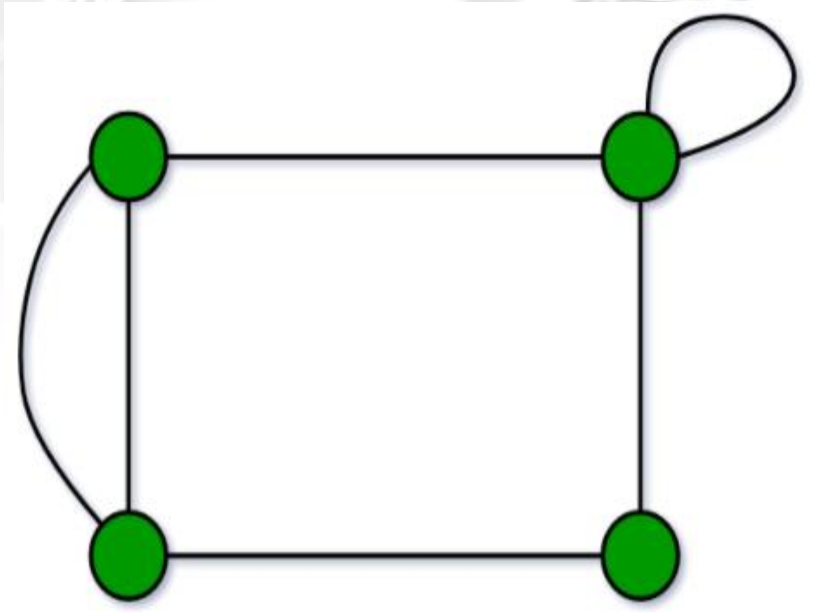




Types of Graphs:

8. Pseudo Graph:

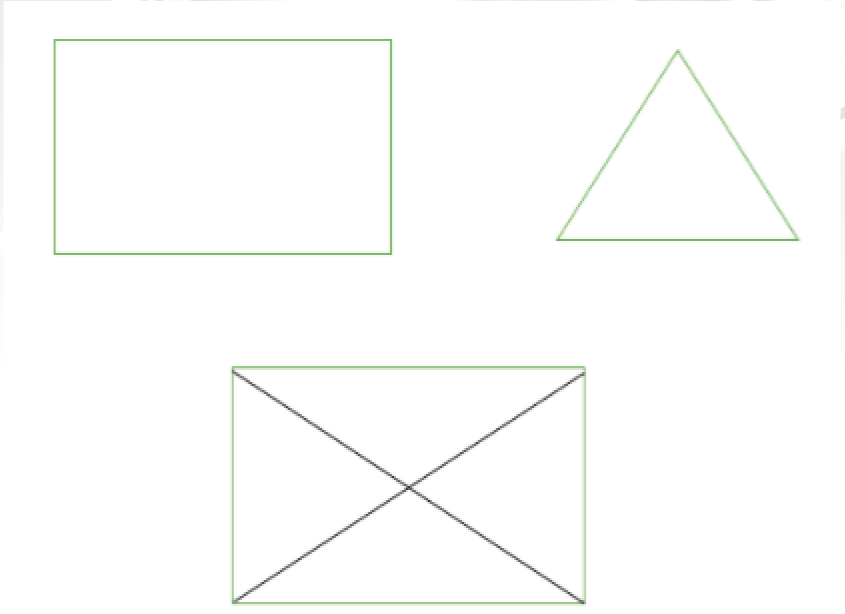
A graph G with a self-loop and some multiple edges is called a pseudo graph. A pseudograph is a type of graph that allows for the existence of loops (edges that connect a vertex to itself) and multiple edges (more than one edge connecting two vertices). In contrast, a simple graph is a graph that does not allow for loops or multiple edges.



Types of Graphs:

9. Regular Graph:

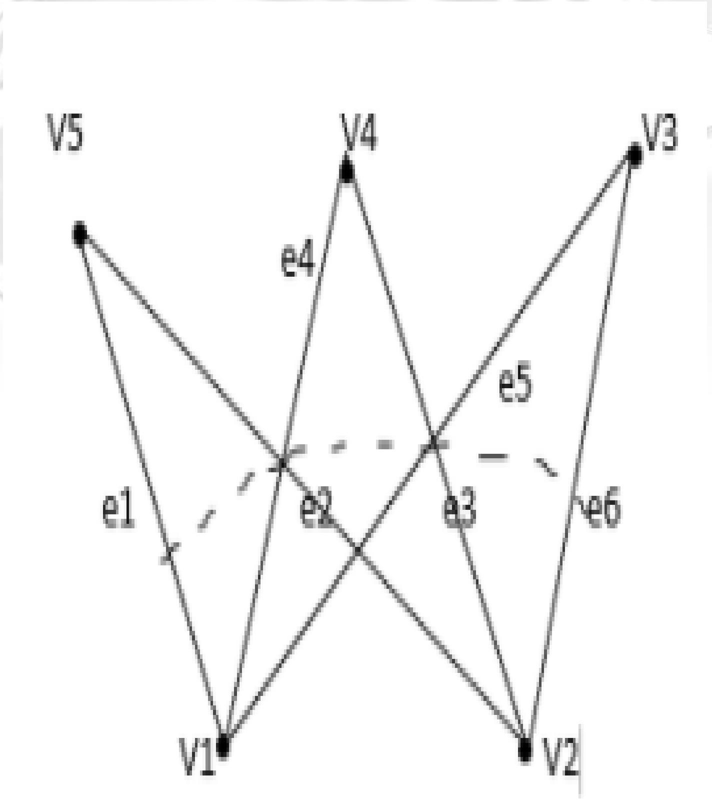
A simple graph is said to be regular if all vertices of graph G are of equal degree. All complete graphs are regular but vice versa is not possible. A regular graph is a type of undirected graph where every vertex has the same number of edges or neighbors.



Types of Graphs:

10. Bipartite Graph:

A graph G = (V, E) is said to be a bipartite graph if its vertex set V(G) can be partitioned into two non-empty disjoint subsets. V1(G) and V2(G) in such a way that each edge e of E(G) has one end in V1(G) and another end in V2(G). The partition V1 U V2 = V is called Bipartite of G. Here in the figure: V1(G)={V5, V4, V3} and V2(G)={V1, V2}



Types of Graphs:

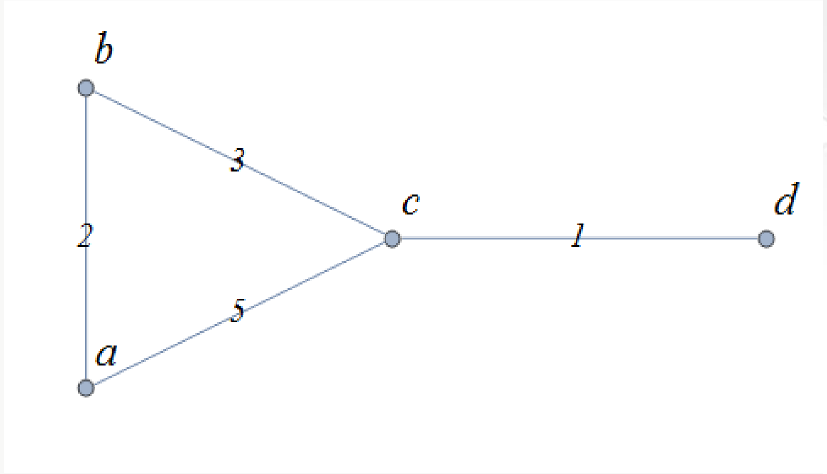
11. Weighted Graphs:

A weighted graph is one in which each edge e is assigned a nonnegative number w(e), called the weight of that edge. Weights are typically associated with costs, or capacities of some type like distance or speed. The adjacency matrices for weighted graphs are very similar to those for graphs that are not necessarily weighted. Instead of using a 1 to represent an edge between two vertices, say vi, and vj, we place the the weight of the edge w(e) in position mi,j of the adjacency matrix as shown in the following two examples.

Types of Graphs:

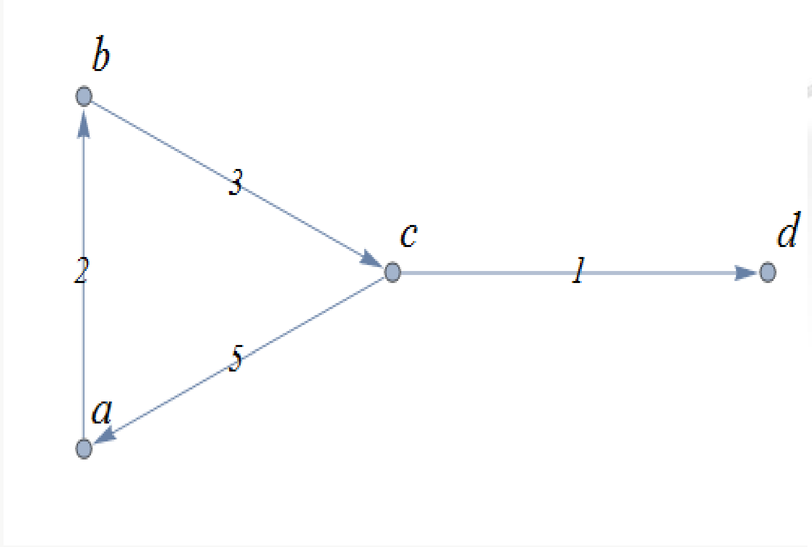
Example 7

Consider first the following weighted undirected graph



Its adjacency matrix is $\begin{pmatrix} 0 & 2 & 5 & 0 \\ 2 & 0 & 3 & 0 \\ 5 & 3 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$.

By contrast, the directed weighted graph below

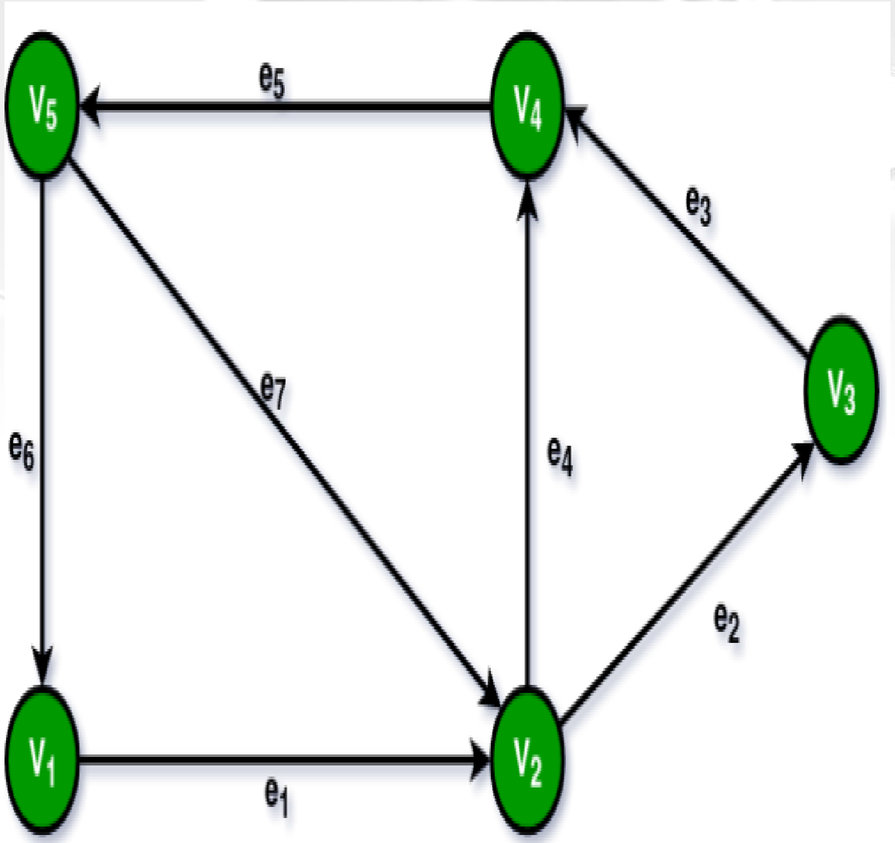


has adjacency matrix $\begin{pmatrix} 0 & 2 & 0 & 0 \\ 0 & 0 & 3 & 0 \\ 5 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}$.

Types of Graphs:

12. Digraph Graph:

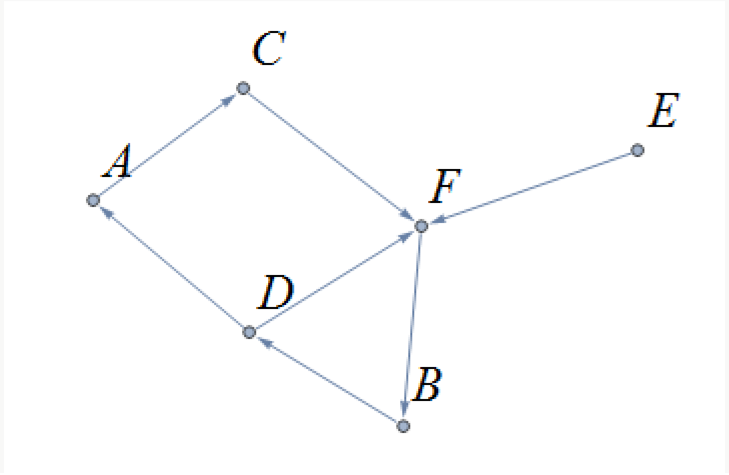
A graph G = (V, E) with a mapping f such that every edge maps onto some ordered pair of vertices (Vi, Vj) are called a Digraph. It is also called ***Directed Graph***. The ordered pair (Vi, Vj) means an edge between Vi and Vj with an arrow directed from Vi to Vj. Here in the figure: e1 = (V1, V2) e2 = (V2, V3) e4 = (V2, V4)



Types of Graphs:

Example 2 - a directed graph.

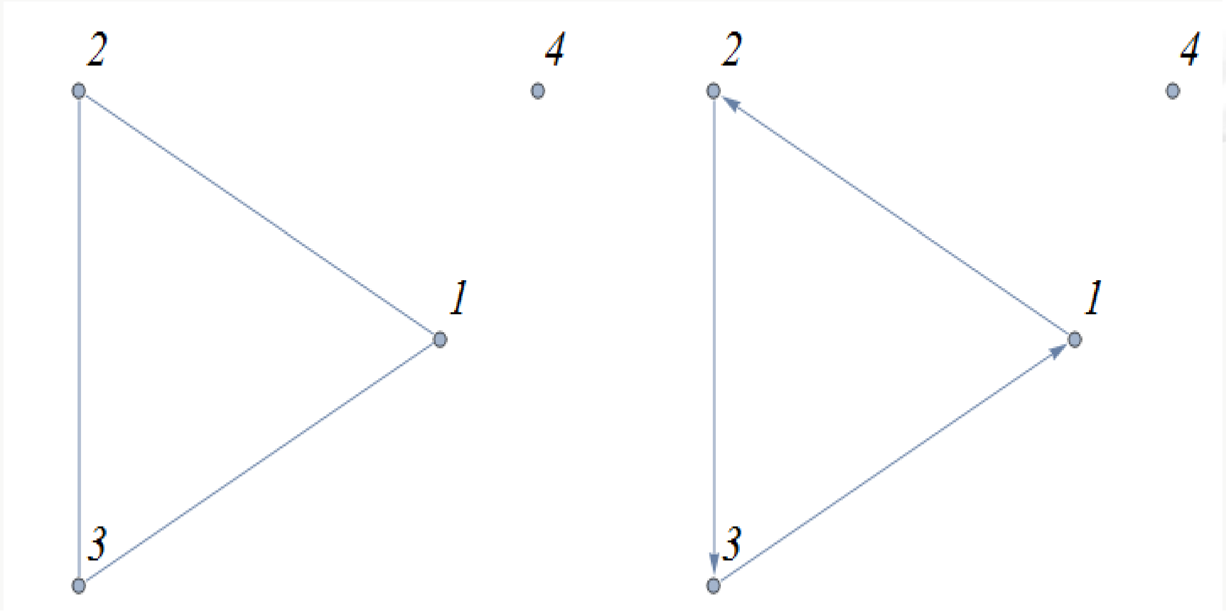
The graph $G = (V, E)$ with vertex set $V = \{A, B, C, D, E, F\}$ and edge set $E = \{\{A, C\}, \{D, A\}, \{B, D\}\{F, B\}, \{C, F\}, \{D, F\}, \{F, E\}\}$ with edge set considered graph and shown below.



Types of Graphs:

Example 6 - adjacency matrices for an undirected graph and for a directed graph

In the figure below the first graph is undirected while the second is a digraph.



Their adjacency matrices are respectively,

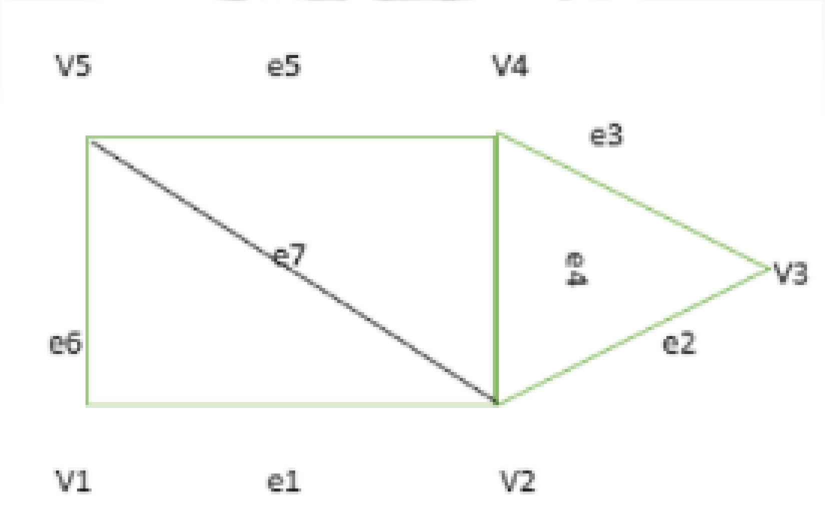
$$\begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \text{ and } \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}.$$

Types of Graphs:

Types of Graphs:

13. Subgraph:

A graph G1 = (V1, E1) is called a subgraph of a graph G(V, E) if V1(G) is a subset of V(G) and E1(G) is a subset of E(G) such that each edge of G1 has same end vertices as in G.



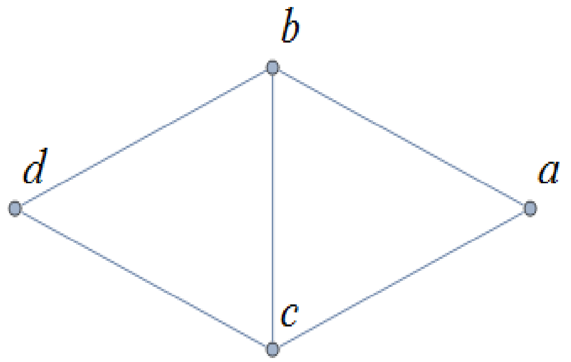
Types of Graphs:

A graph $H=(V1,E1)$ is said to be a subgraph of the graph $G=(V, E)$ if $V1\subseteq V$ and $E1\subseteq E$.

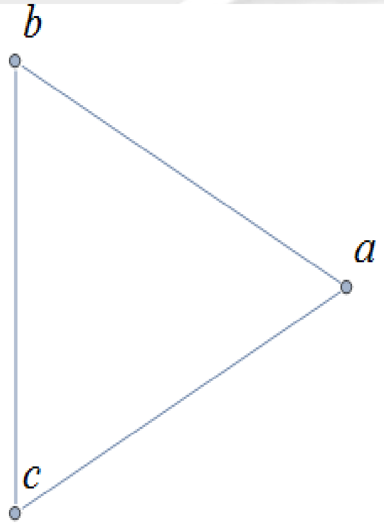
If the vertex $v\in V$ belongs to the graph $G=(V,E)$, we denote by $G-v$, the subgraph obtained from G by removing the vertex v and all edges in E adjacent to the vertex v.

Below is shown a graph G, and the subgraph $G-d$ formed by removing the vertex d.

Types of Graphs:



The graph G



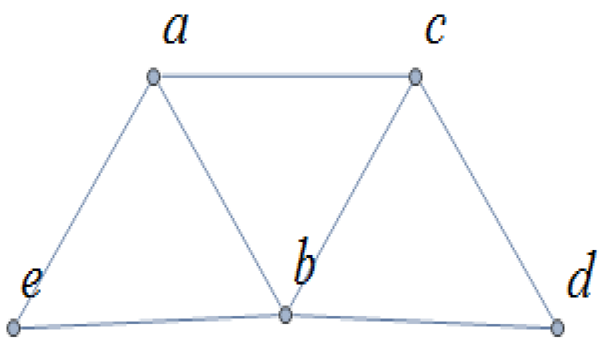
The graph $G - d$

A natural generalization of the subgraph obtained by removing a vertex is the subgraph obtained by removing multiple vertices and the edges associated with the removed vertices. The subgraph obtained is called the subgraph **induced** by removing those vertices.

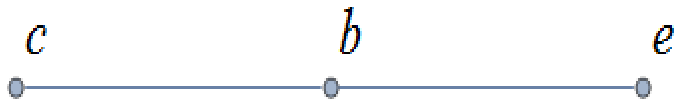
Types of Graphs:

Example 8

Below is a graph $G(V,E)$ and the subgraph obtained by $V - \{a,d\}$, called the induced subgraph $G - \{a,d\}$, with a slight abuse of notation



The graph G



The induced subgraph $G - \{a, d\}$

Types of Graphs:

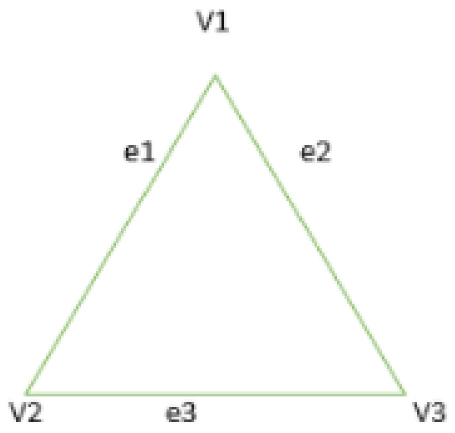
14. Connected or Disconnected Graph:

Graph G is said to be connected if any pair of vertices (Vi, Vj) of a graph G is reachable from one another. Or a graph is said to be connected if there exists at least one path between each and every pair of vertices in graph G, otherwise, it is disconnected. A null graph with n vertices is a disconnected graph consisting of n components. Each component consists of one vertex and no edge.

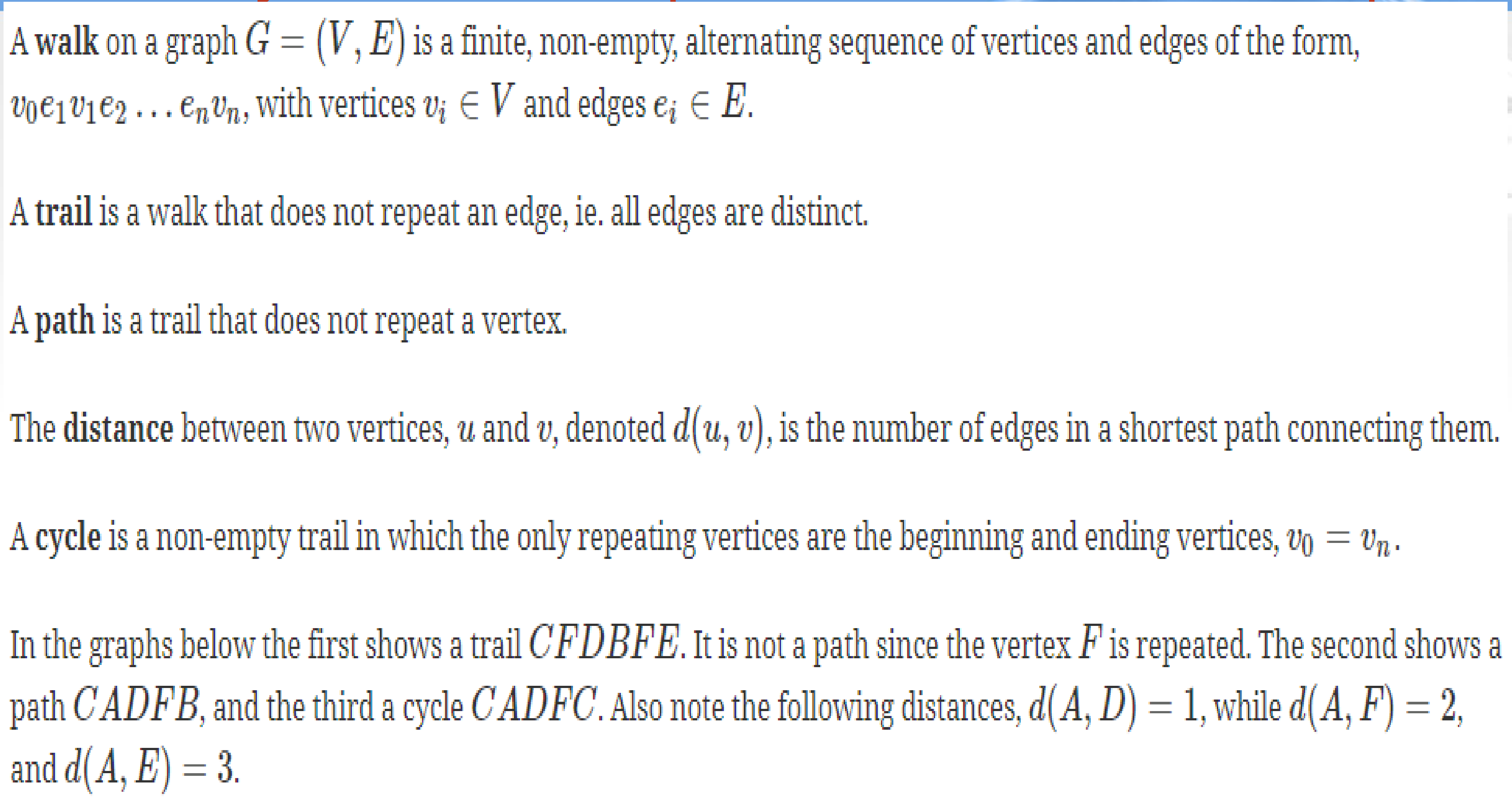
Types of Graphs:

15. Cyclic Graph:

A graph G consisting of n vertices and n> = 3 that is V1, V2, V3- – – – Vn and edges (V1, V2), (V2, V3), (V3, V4)- – – – (Vn, V1) are called cyclic graph.



Connectivity, Eulerian Graphs, and Hamiltonian Graphs



A **walk** on a graph $G = (V, E)$ is a finite, non-empty, alternating sequence of vertices and edges of the form, $v_0 e_1 v_1 e_2 \ldots e_n v_n$, with vertices $v_i \in V$ and edges $e_i \in E$.

A **trail** is a walk that does not repeat an edge, ie. all edges are distinct.

A **path** is a trail that does not repeat a vertex.

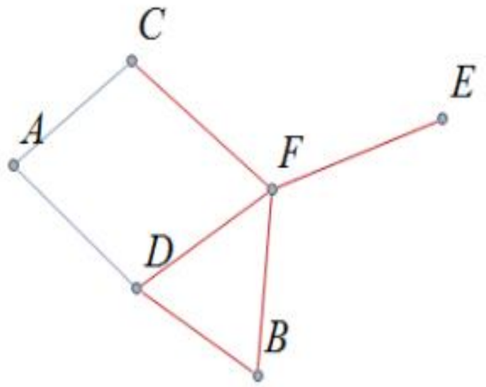
The **distance** between two vertices, u and v, denoted $d(u, v)$, is the number of edges in a shortest path connecting them.

A **cycle** is a non-empty trail in which the only repeating vertices are the beginning and ending vertices, $v_0 = v_n$.

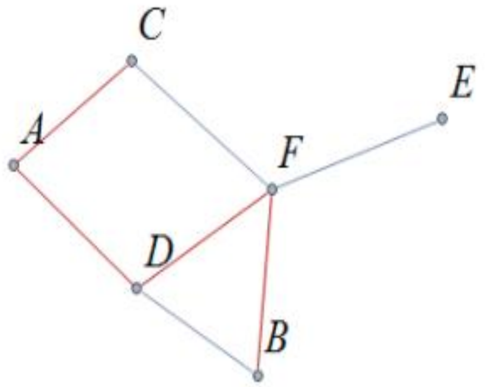
In the graphs below the first shows a trail $CFDBFE$. It is not a path since the vertex F is repeated. The second shows a path $CADFB$, and the third a cycle $CADFC$. Also note the following distances, $d(A, D) = 1$, while $d(A, F) = 2$, and $d(A, E) = 3$.

Connectivity, Eulerian Graphs, and Hamiltonian Graphs

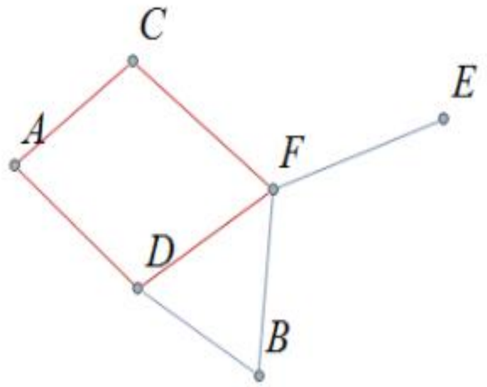
and $d(A, E) = 3$.



The trail CFDBFE



The path CADFB

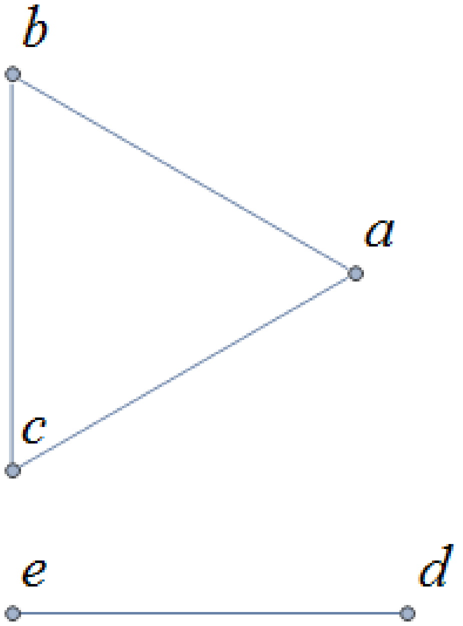


The cycle CADFC

A graph is **connected** if there is a path from each vertex to every other vertex.

Connectivity, Eulerian Graphs, and Hamiltonian Graphs

The graph below is not connected,



and has adjacency matrix,

$$\begin{pmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{pmatrix}.$$

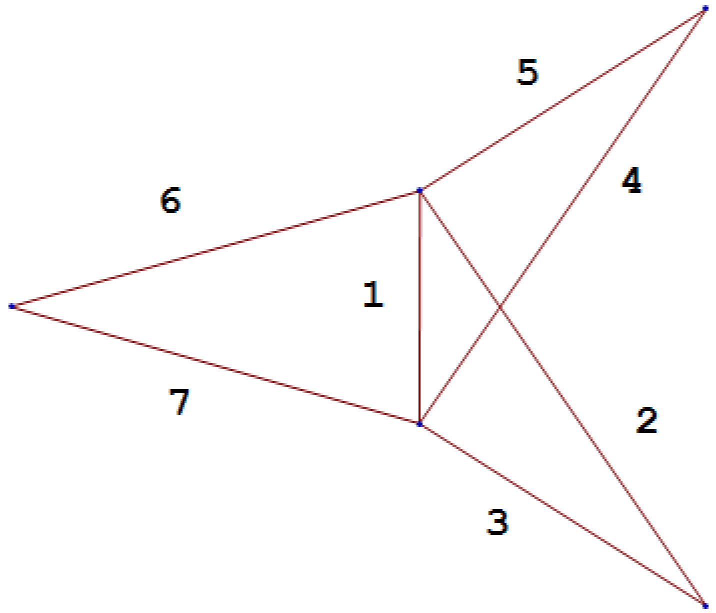
Eulerian Graphs

Informally an Eulerian graph is one in which there is a closed (beginning and ending with the same vertex) trail that includes all edges. To define this precisely, we use the idea of an Eulerian trail.

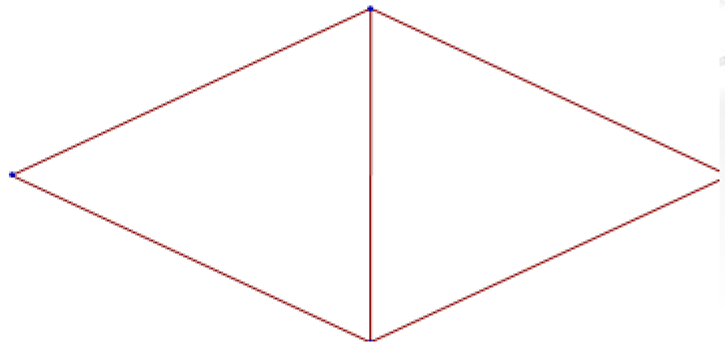
An Eulerian trail or Eulerian circuit is a closed trail containing each edge of the graph G=(V, G) exactly once and returning to the start vertex. A graph with an Eulerian trail is considered Eulerian or is said to be an Eulerian graph .

In the following, the first graph is Eulerian with the Eulerian circuit sequenced from 1 to 7. The second is not an Eulerian graph. Convince yourself of this fact by looking at all necessary trails or closed trails.

Eulerian Graphs



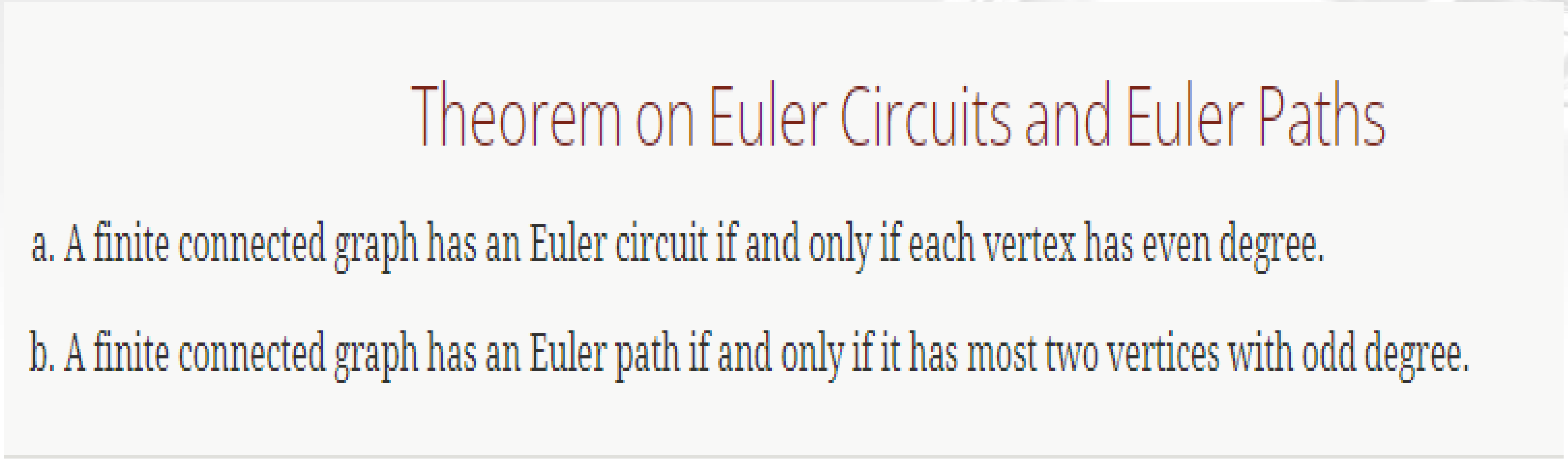
An Eulerian graph,with
Eulerian circuit 1, 2, ..., 7



A graph that is not Eulerian

An **Euler path** on a graph is a path that uses each edge of the graph exactly once. The following are useful characterizations of graphs with Euler circuits and Euler paths and are due to Leonhard Euler

Eulerian Graphs



Theorem on Euler Circuits and Euler Paths

a. A finite connected graph has an Euler circuit if and only if each vertex has even degree.

b. A finite connected graph has an Euler path if and only if it has most two vertices with odd degree.

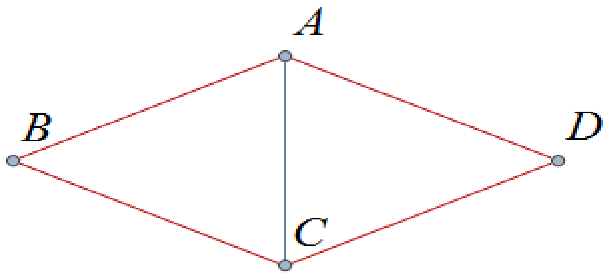
Hamiltonian Graphs

A cycle in a graph G=(V,E), is said to be a Hamiltonian cycle if every vertex, except for the starting and ending vertex in V, is visited exactly once.

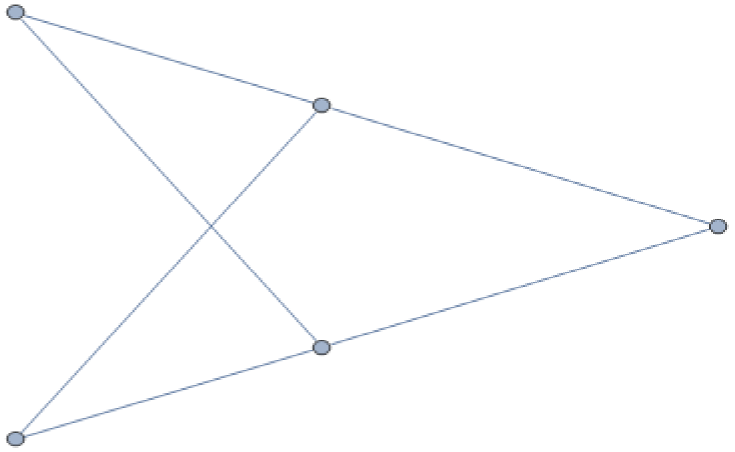
A graph is Hamiltonian , or said to be a Hamiltonian graph , if it contains a Hamiltonian cycle.

The following graph is Hamiltonian and shows a Hamiltonian cycle ABCDA, highlighted, while the second graph is not Hamiltonian.

Hamiltonian Graphs



A Hamiltonian graph with Hamiltonian cycle, ABCDA



A graph that is not Hamiltonian

Theorem (Dirac) on Hamiltonian graphs

A simple graph, with $n \geq 3$ vertices, is Hamiltonian if every vertex v has degree $d(v) \geq \frac{n}{2}$.

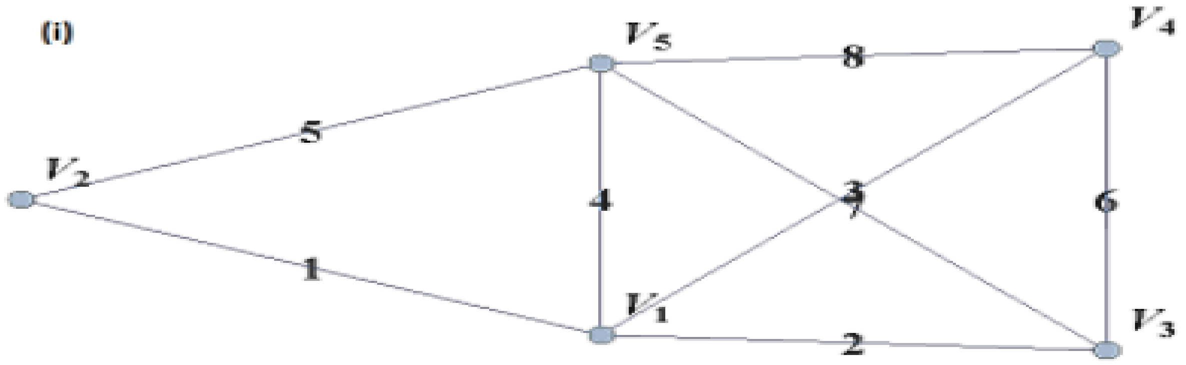
Exercises

1. For each of the following graphs, find their

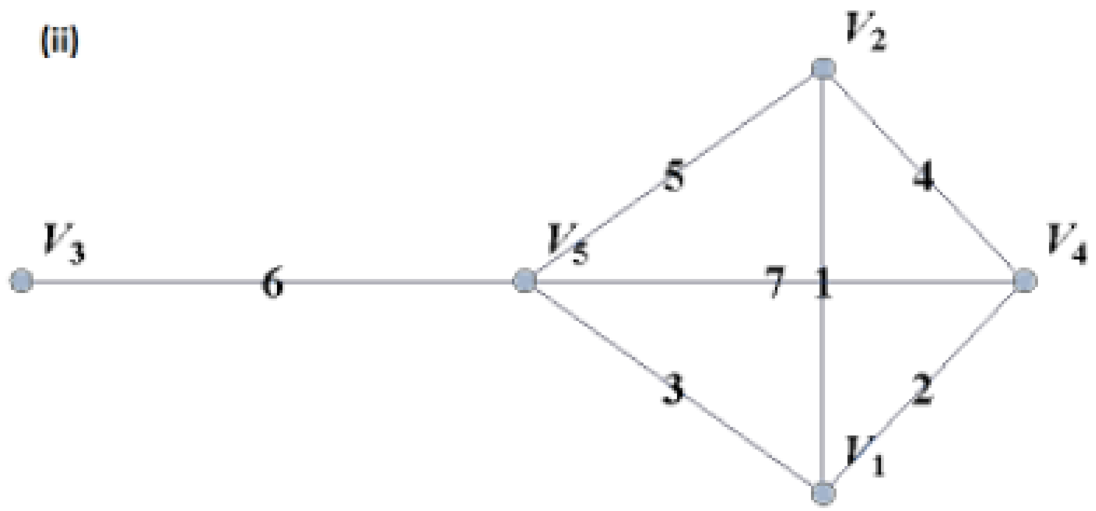
   a. Adjacency matrices

   b. Adjacency lists

(i)



Exercises



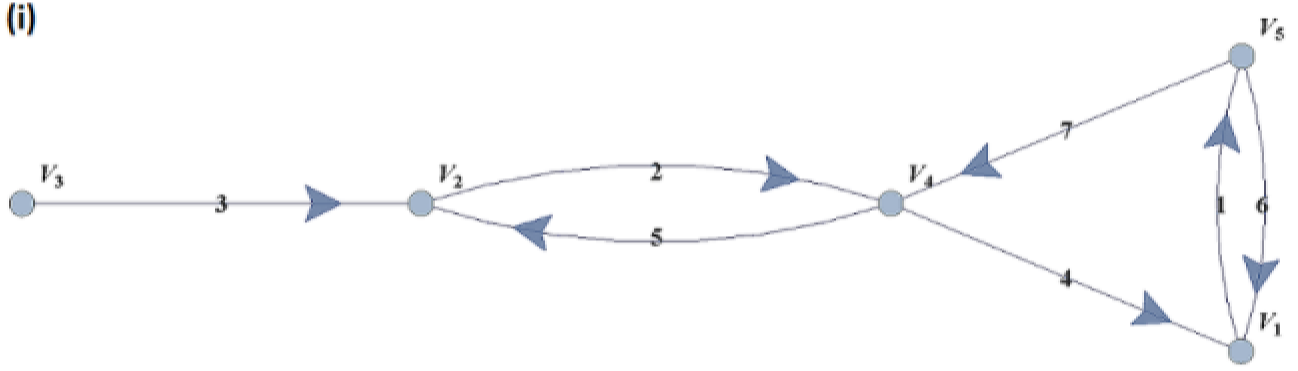
Exercises

2. For each of the following digraphs, find their

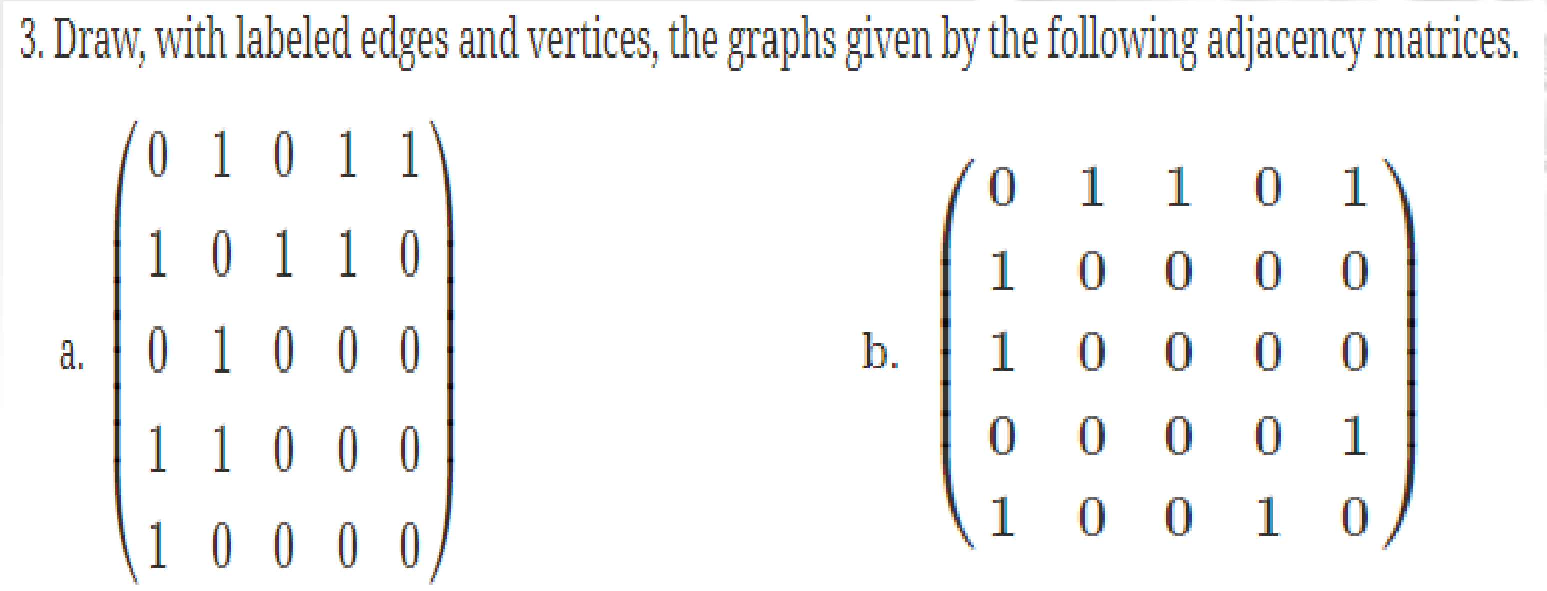
   a. Adjacency matrices

   b. Adjacency lists

(i)



Exercises



3. Draw, with labeled edges and vertices, the graphs given by the following adjacency matrices.

a. $$\begin{pmatrix} 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{pmatrix}$$

b. $$\begin{pmatrix} 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 \end{pmatrix}$$

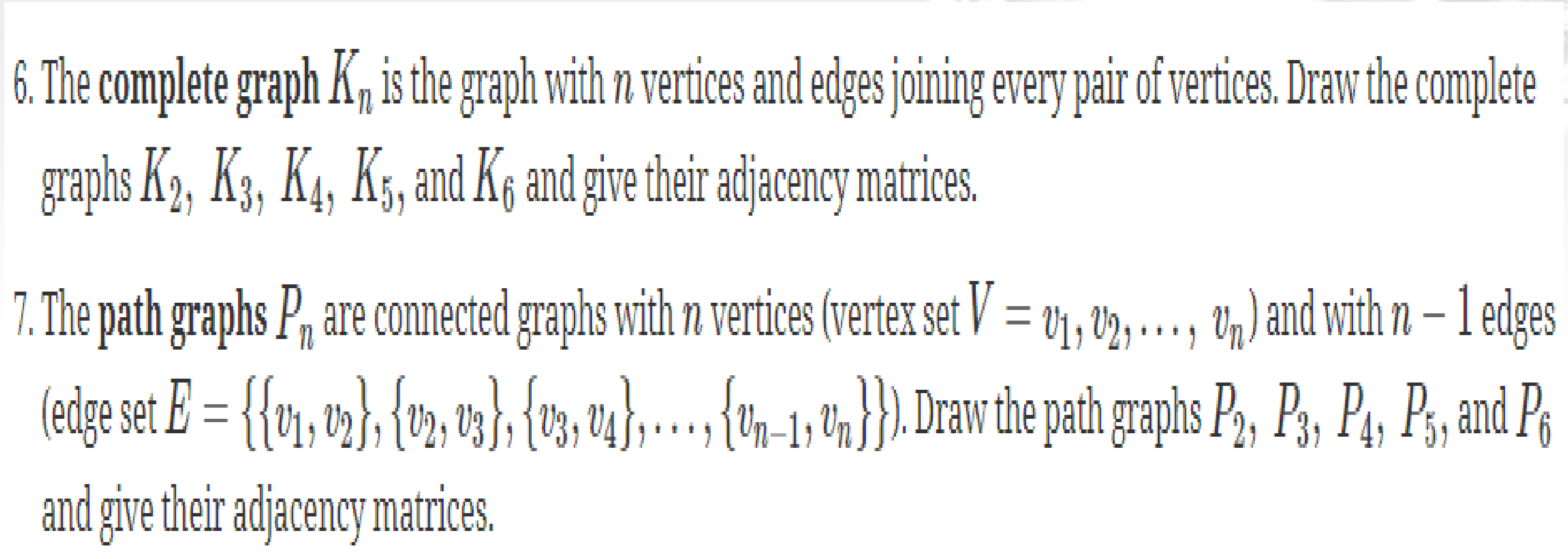
Exercises

5. Draw, with labeled edges and vertices, the weighted graphs (or digraphs) given by the following adjacency matrices.

a. $\begin{pmatrix} 0 & 10 & 3 & 0 & 5 \\ 10 & 0 & 2 & 3 & 0 \\ 3 & 2 & 0 & 7 & 4 \\ 0 & 3 & 7 & 0 & 1 \\ 5 & 0 & 4 & 1 & 0 \end{pmatrix}$

b. $\begin{pmatrix} 0 & 2 & 3 & 4 \\ 0 & 0 & 5 & 7 \\ 0 & 0 & 0 & 6 \\ 5 & 8 & 8 & 0 \end{pmatrix}$

Exercises



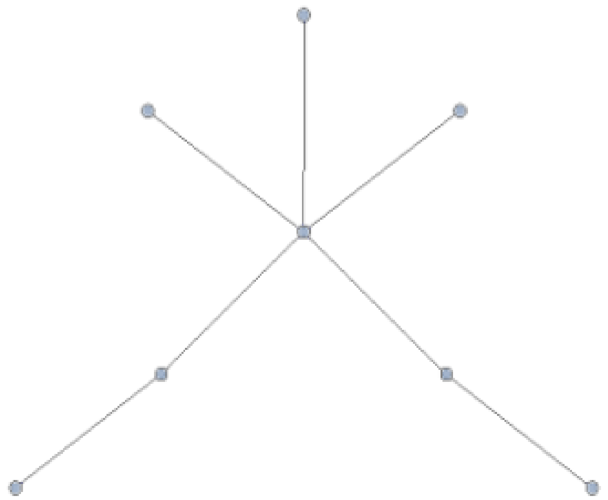
6. The **complete graph** K_n is the graph with n vertices and edges joining every pair of vertices. Draw the complete graphs $K_2,\ K_3,\ K_4,\ K_5$, and K_6 and give their adjacency matrices.

7. The **path graphs** P_n are connected graphs with n vertices (vertex set $V = v_1, v_2, \ldots,\ v_n$) and with $n - 1$ edges (edge set $E = \{\{v_1, v_2\}, \{v_2, v_3\}, \{v_3, v_4\}, \ldots, \{v_{n-1}, v_n\}\}$). Draw the path graphs $P_2,\ P_3,\ P_4,\ P_5$, and P_6 and give their adjacency matrices.

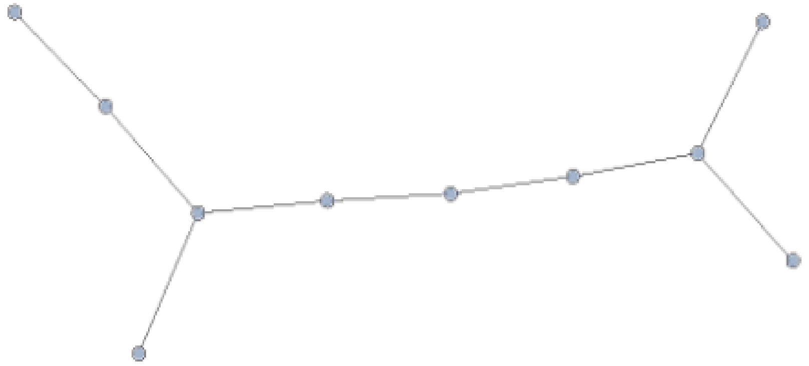
Exercises

10. A **tree** is a connected graph with no cycles. It can be shown, using mathematical induction, that a tree with n vertices must have exactly $n-1$ edges. Determine which of following graphs are trees and which are not. Explain your reasoning.

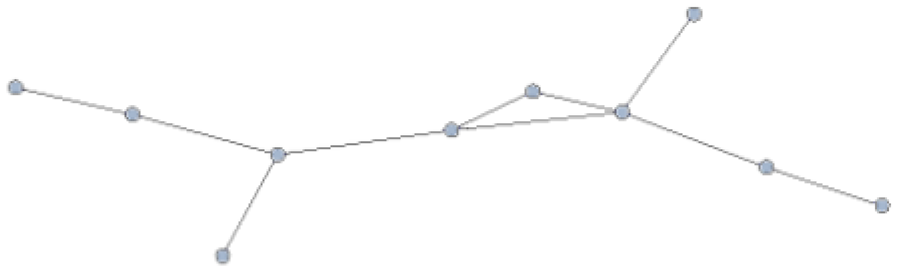
(i)

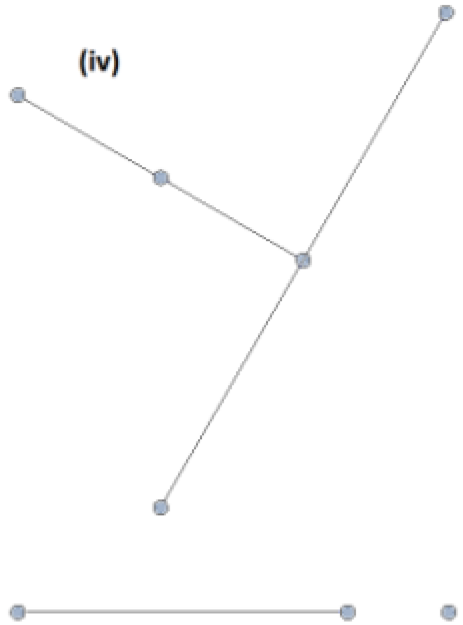


(ii)



Exercises

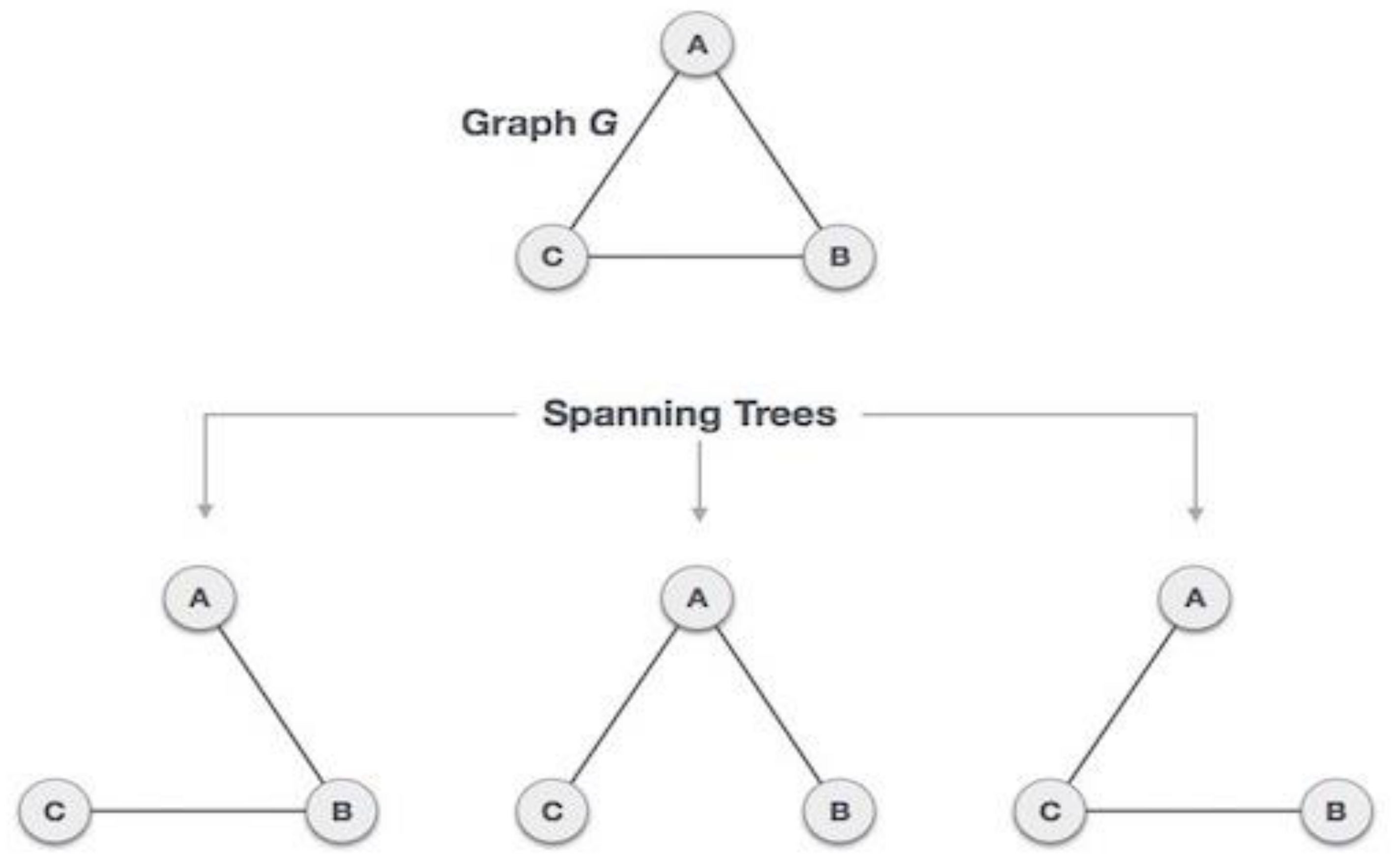




Spanning Tree

A spanning tree is a subset of Graph G, which has all the vertices covered with minimum possible number of edges. Hence, a spanning tree does not have cycles and it cannot be disconnected.. By this definition, we can draw a conclusion that every connected and undirected Graph G has at least one spanning tree. A disconnected graph does not have any spanning tree, as it cannot be spanned to all its vertices.

Spanning Tree



Spanning Tree

We found three spanning trees off one complete graph. A complete undirected graph can have maximum $\mathbf{n^{n-2}}$ number of spanning trees, where **n** is the number of nodes. In the above addressed example, **n is 3,** hence $\mathbf{3^{3-2} = 3}$ spanning trees are possible.

General Properties of Spanning Tree

Following are a few properties of the spanning tree connected to graph G

- A connected graph G can have more than one spanning tree.
- All possible spanning trees of graph G, have the same number of edges and vertices.
- The spanning tree does not have any cycle (loops).
- Removing one edge from the spanning tree will make the graph disconnected, i.e. the spanning tree is **minimally connected**.
- Adding one edge to the spanning tree will create a circuit or loop, i.e. the spanning tree is **maximally acyclic**.

Mathematical Properties of Spanning Tree

- Spanning tree has **n-1** edges, where **n** is the number of nodes (vertices).
- From a complete graph, by removing maximum **(e - n + 1)** edges, we can construct a spanning tree.
- A complete graph can have maximum $\mathbf{n^{n-2}}$ number of spanning trees.

Thus, we can conclude that spanning trees are a subset of connected Graph G and disconnected graphs do not have spanning tree.

Application of Spanning Tree

Spanning tree is basically used to find a minimum path to connect all nodes in a graph. Common application of spanning trees are −

- **Civil Network Planning**
- **Computer Network Routing Protocol**
- **Cluster Analysis**

Let us understand this through a small example. Consider, city network as a huge graph and now plans to deploy telephone lines in such a way that in minimum lines we can connect to all city nodes. This is where the spanning tree comes into picture.

Minimum Spanning Tree (MST)

In a weighted graph, a minimum spanning tree is a spanning tree that has minimum weight than all other spanning trees of the same graph. In real-world situations, this weight can be measured as distance, congestion, traffic load or any arbitrary value denoted to the edges.

Minimum Spanning-Tree Algorithm

We shall learn about two most important spanning tree algorithms here −

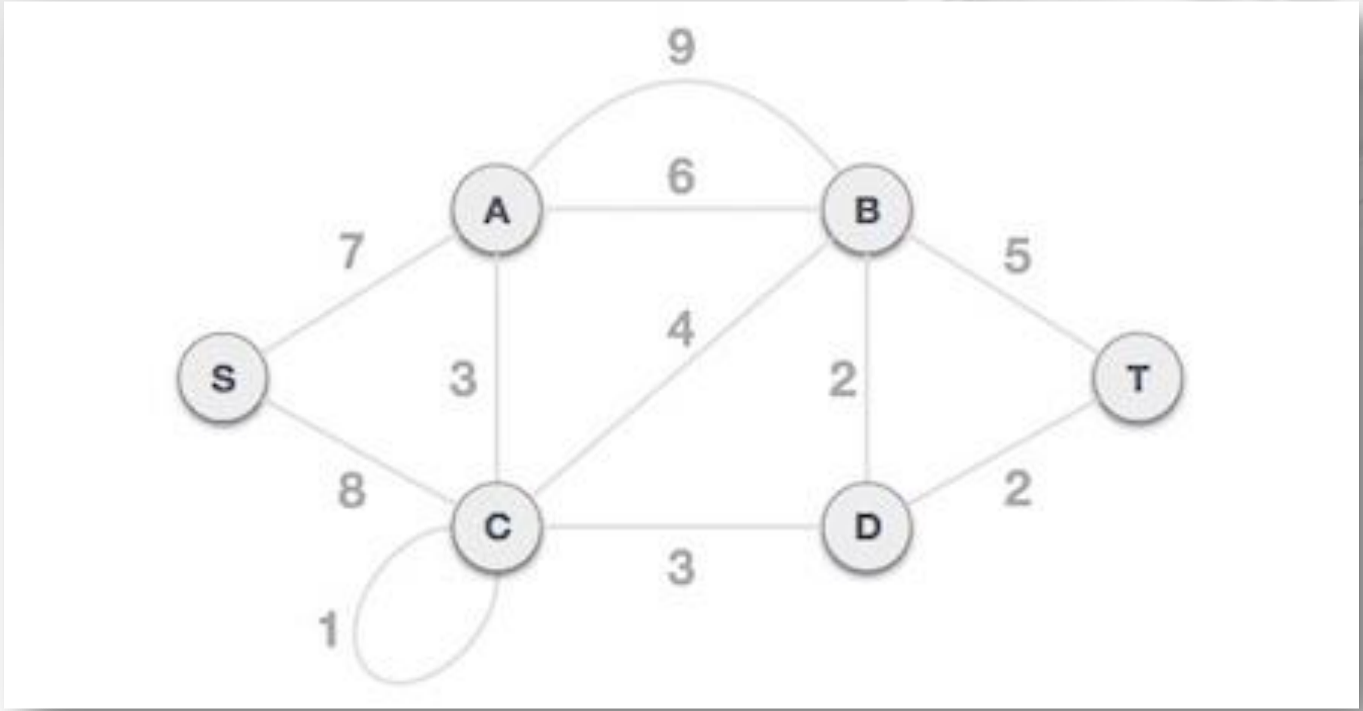
- Kruskal's Algorithm
- Prim's Algorithm

Kruskal's Algorithm

Kruskal's algorithm to find the minimum cost spanning tree uses the greedy approach. This algorithm treats the graph as a forest and every node it has as an individual tree. A tree connects to another only and only if, it has the least cost among all available options and does not violate MST properties.

To understand Kruskal's algorithm let us consider the following example

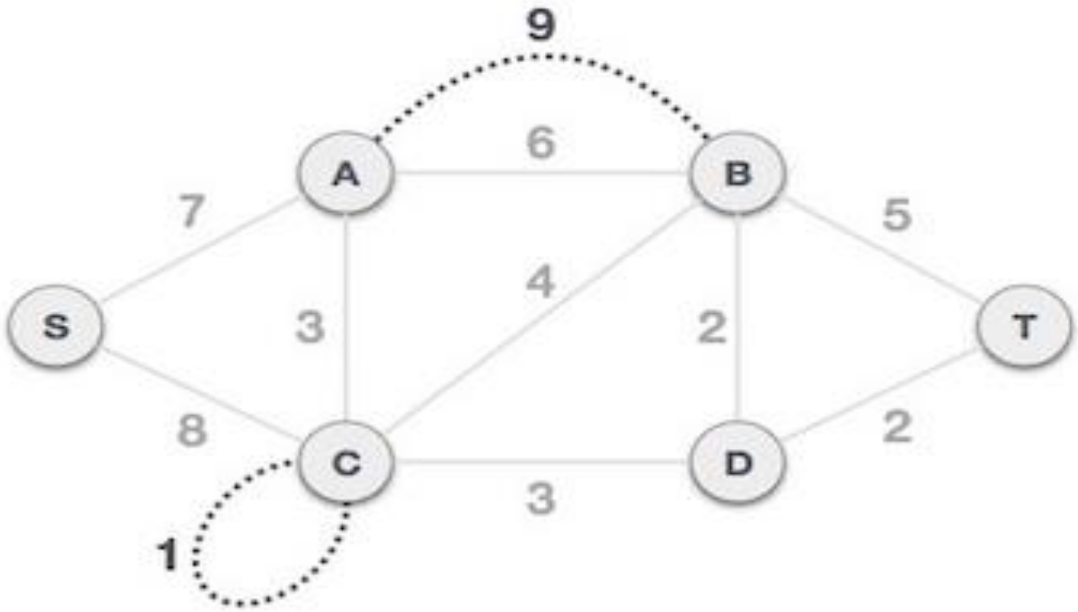
Kruskal's Algorithm



Kruskal's Algorithm

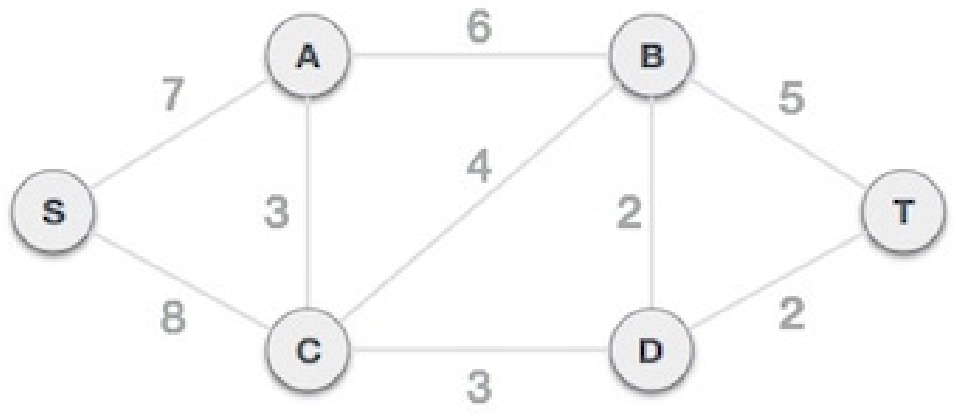
Step 1 - Remove all loops and Parallel Edges

Remove all loops and parallel edges from the given graph.

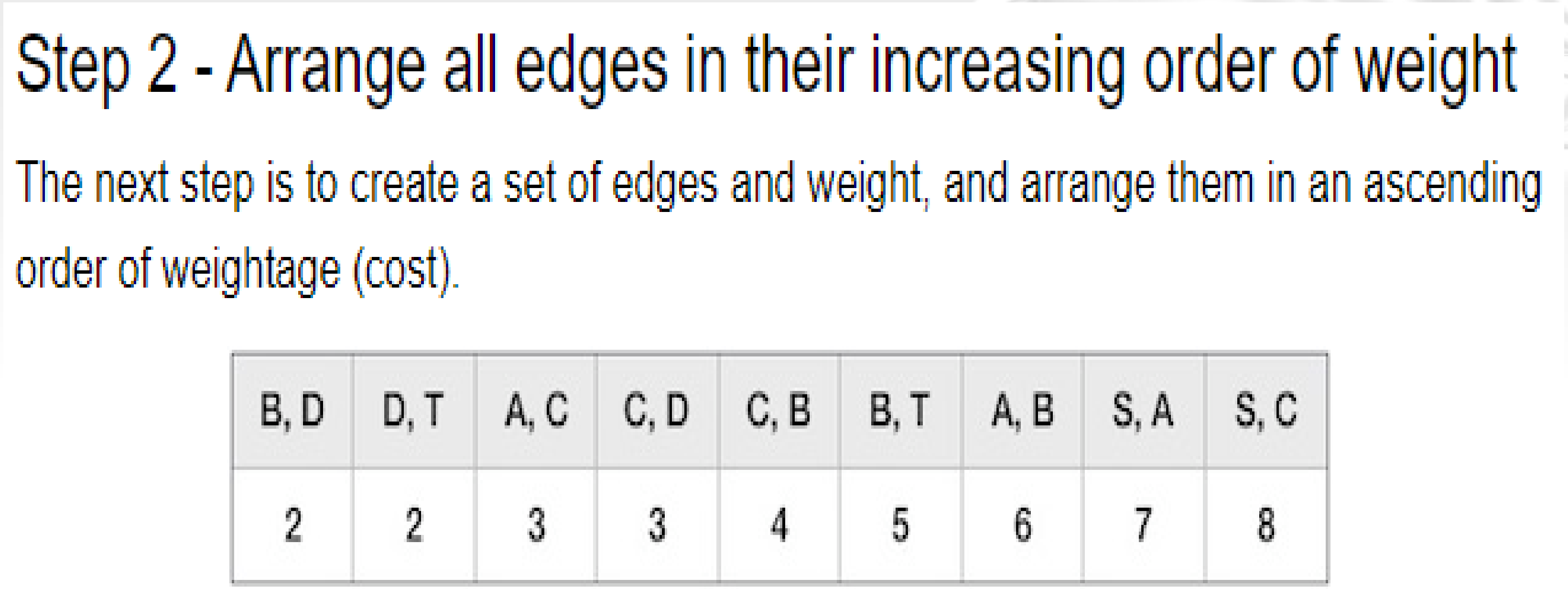


Kruskal's Algorithm

Remove all loops and parallel edges from the given graph. In case of parallel edges, keep the one which has the least cost associated and remove all others.



Kruskal's Algorithm



Step 2 - Arrange all edges in their increasing order of weight

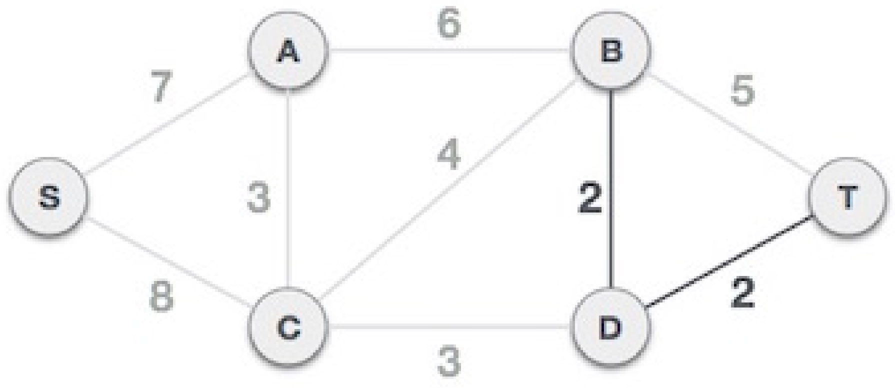
The next step is to create a set of edges and weight, and arrange them in an ascending order of weightage (cost).

B, D	D, T	A, C	C, D	C, B	B, T	A, B	S, A	S, C
2	2	3	3	4	5	6	7	8

Kruskal's Algorithm

Step 3 - Add the edge which has the least weightage

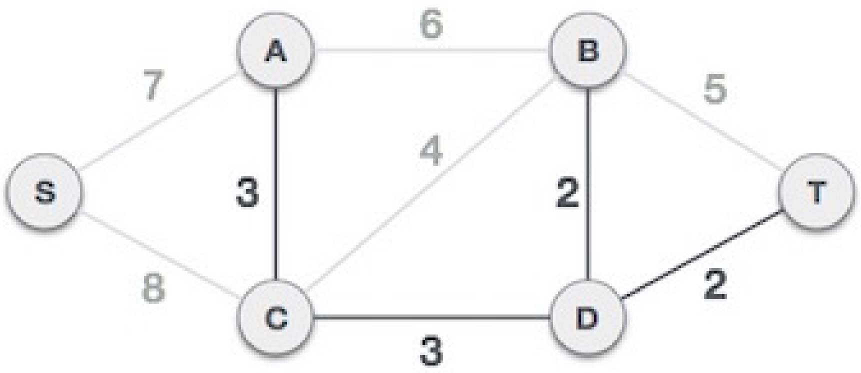
Now we start adding edges to the graph beginning from the one which has the least weight. Throughout, we shall keep checking that the spanning properties remain intact. In case, by adding one edge, the spanning tree property does not hold then we shall consider not to include the edge in the graph.



Kruskal's Algorithm

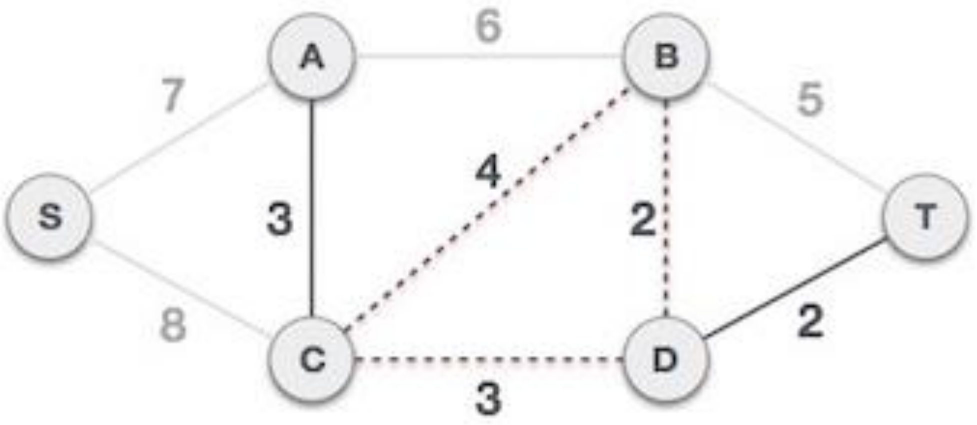
The least cost is 2 and edges involved are B,D and D,T. We add them. Adding them does not violate spanning tree properties, so we continue to our next edge selection.

Next cost is 3, and associated edges are A,C and C,D. We add them again –



Kruskal's Algorithm

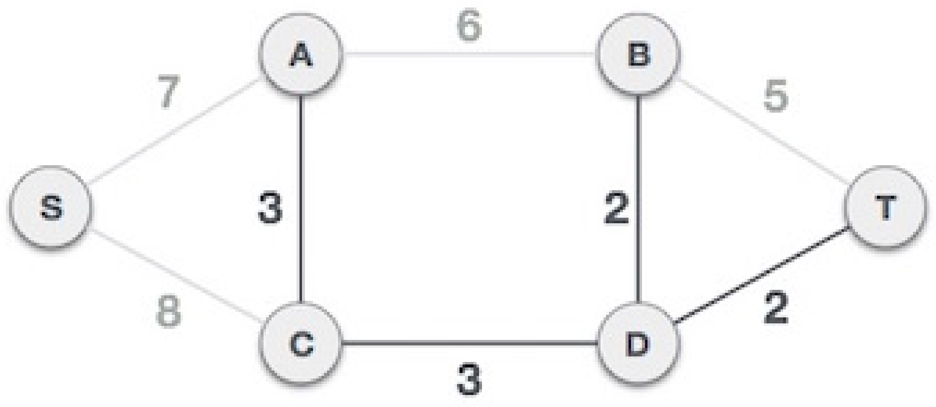
Next cost in the table is 4, and we observe that adding it will create a circuit in the graph. –



We ignore it. In the process we shall ignore/avoid all edges that create a circuit.

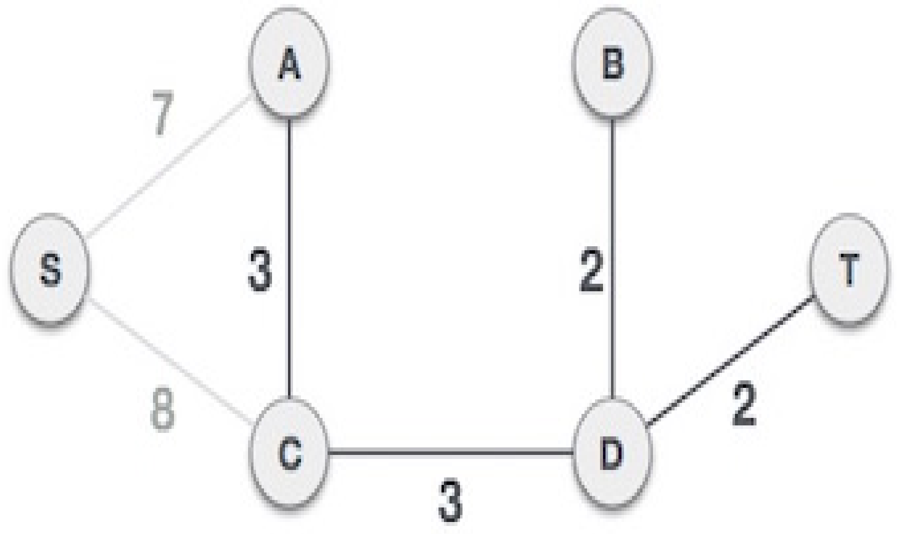
Kruskal's Algorithm

We ignore it. In the process we shall ignore/avoid all edges that create a circuit.



We observe that edges with cost 5 and 6 also create circuits. We ignore them and move on.

Kruskal's Algorithm



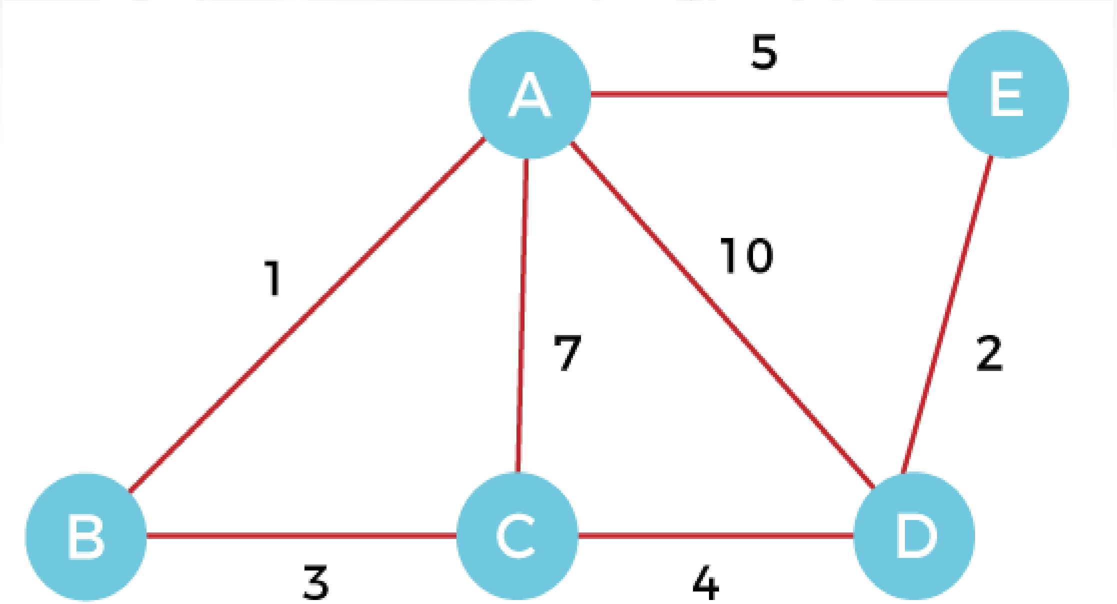
Now we are left with only one node to be added. Between the two least cost edges available 7 and 8, we shall add the edge with cost 7.

Kruskal's Algorithm

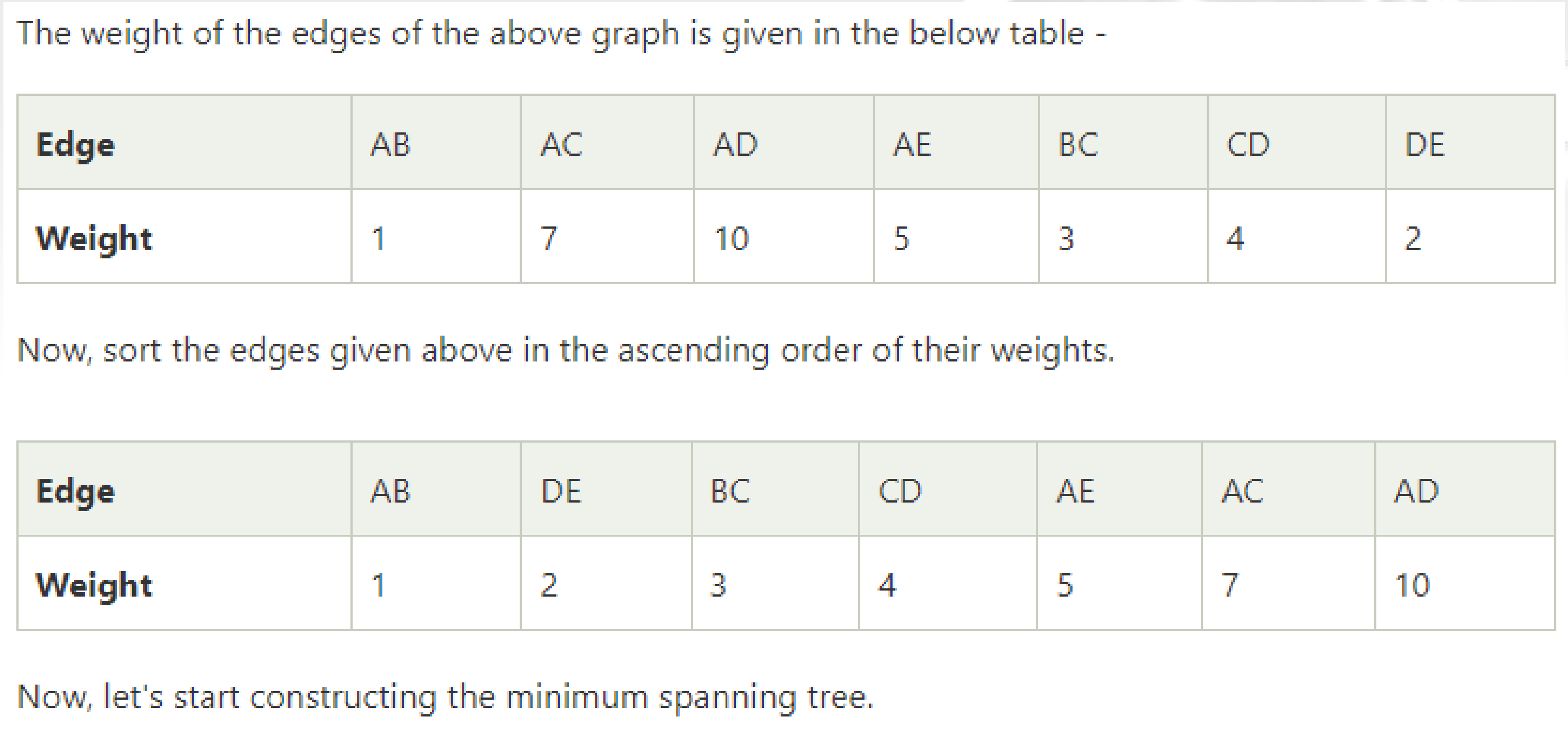
Example of Kruskal's algorithm

Now, let's see the working of Kruskal's algorithm using an example. It will be easier to understand Kruskal's algorithm using an example.

Suppose a weighted graph is -



Kruskal's Algorithm



The weight of the edges of the above graph is given in the below table -

Edge	AB	AC	AD	AE	BC	CD	DE
Weight	1	7	10	5	3	4	2

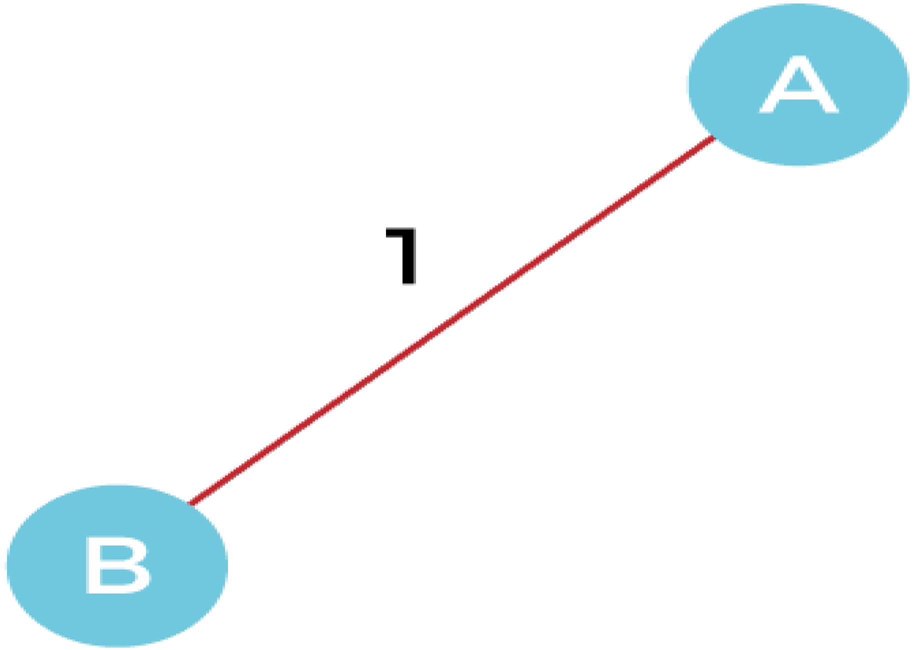
Now, sort the edges given above in the ascending order of their weights.

Edge	AB	DE	BC	CD	AE	AC	AD
Weight	1	2	3	4	5	7	10

Now, let's start constructing the minimum spanning tree.

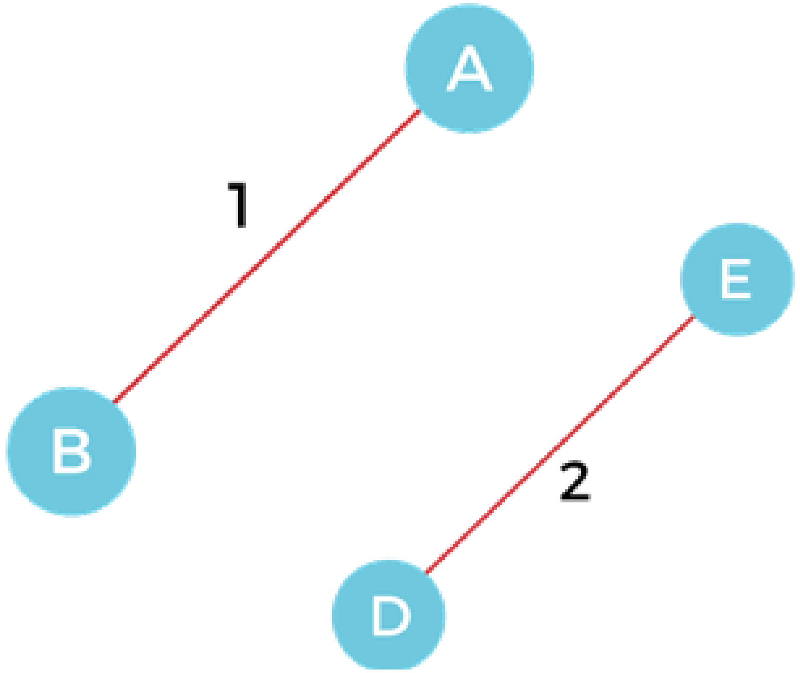
Kruskal's Algorithm

Step 1 - First, add the edge **AB** with weight **1** to the MST.



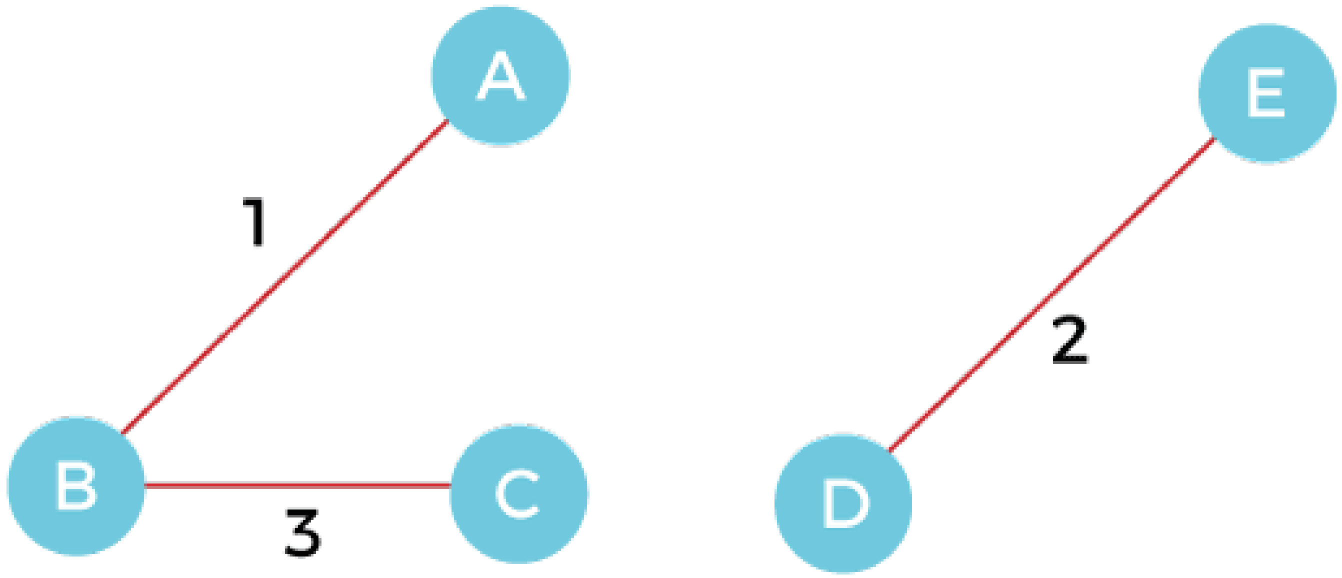
Kruskal's Algorithm

Step 2 - Add the edge **DE** with weight **2** to the MST as it is not creating the cycle.



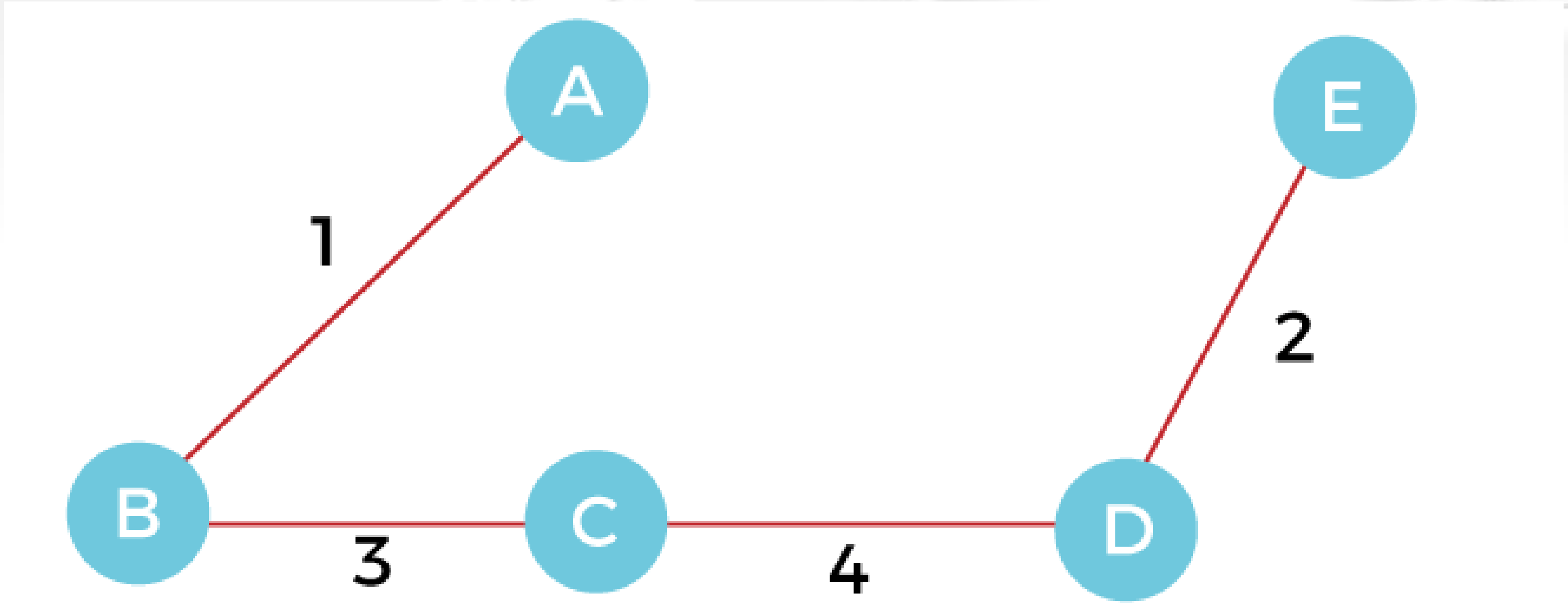
Kruskal's Algorithm

Step 3 - Add the edge **BC** with weight **3** to the MST, as it is not creating any cycle or loop.

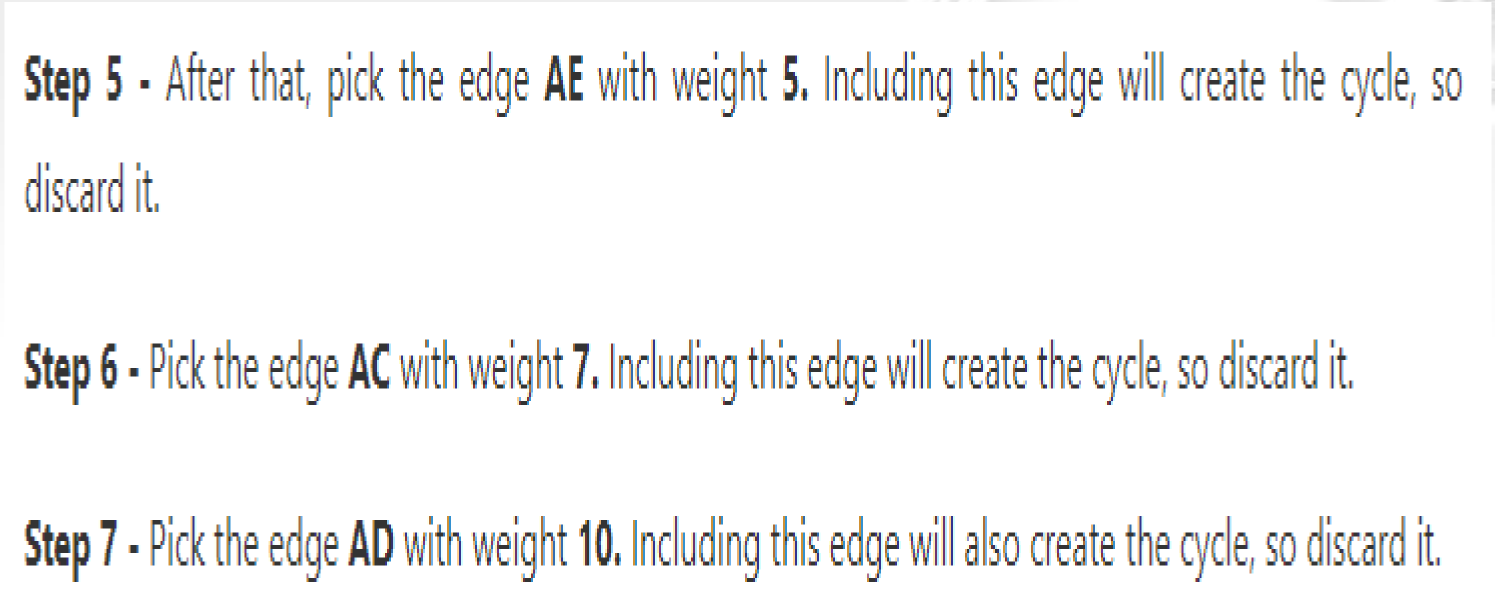


Kruskal's Algorithm

Step 4 - Now, pick the edge **CD** with weight **4** to the MST, as it is not forming the cycle.



Kruskal's Algorithm



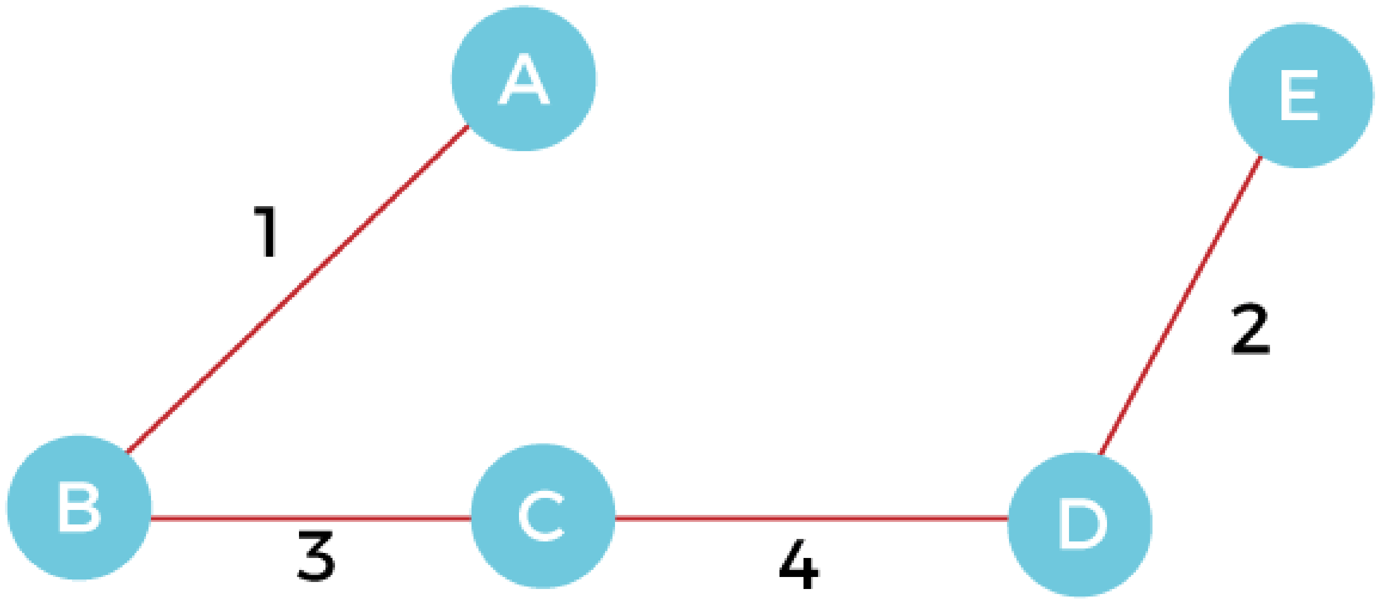
Step 5 - After that, pick the edge **AE** with weight **5.** Including this edge will create the cycle, so discard it.

Step 6 - Pick the edge **AC** with weight **7.** Including this edge will create the cycle, so discard it.

Step 7 - Pick the edge **AD** with weight **10.** Including this edge will also create the cycle, so discard it.

Kruskal's Algorithm

So, the final minimum spanning tree obtained from the given weighted graph by using Kruskal's algorithm is -



The cost of the MST is = AB + DE + BC + CD = 1 + 2 + 3 + 4 = 10.

Boolean Algebra Simplification

https://www.electronics-tutorials.ws/boolean/boolean-algebra-simplification.html

شكراً لحسن المتابعة