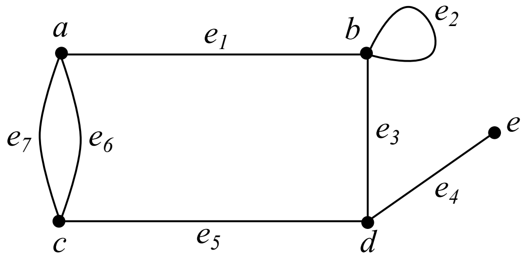
Unit 6

Relations and Graphs

6.1 Graphs

6.1.1 Graph Basics

An **undirected graph** G is an ordered pair (V, E) where V is a set of vertices and E is a set of undirected edges each of which joins a pair of vertices.
For example, $G = (V, E)$ where $V = \{a, b, c, d, e\}$ and $E = \{e_1, e_2, e_3, e_4, e_5, e_6, e_7\}$ is an undirected graph as given below:



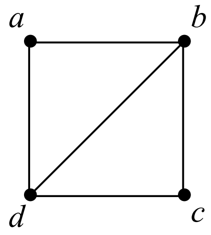
A pair of vertices that are connected by an edge are said to be **adjacent vertices**. For example, in above graph, a and b are adjacent vertices whereas a and d are non-adjacent. The vertices that an edge joins are called its **endpoints** such as vertices d and e which are endpoints of the edge e_4.

Loop: An edge that has the same vertex as both its endpoints, is called a loop. For example, e_2 is a loop in the above graph.

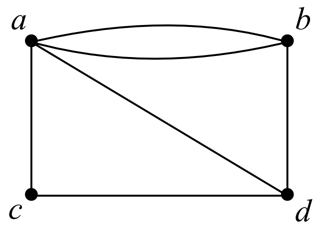
Multiple/Parallel Edges: Two or more edges are said to be multiple or parallel edges if they join the same pair of vertices. For example, e_6 and e_7 are multiple edges.

Types of Undirected Graphs:

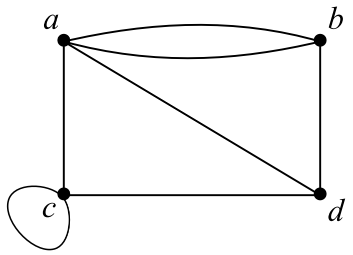
1. **Simple Graph:** An undirected graph that has neither loops not multiple edges is called a simple graph. For example, the following graph is a simple graph.



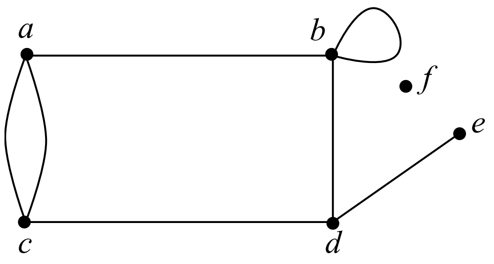
2. **Multigraph:** An undirected graph that can have multiple edges but not loops, is called a multigraph. For example, the following graph is a multigraph.



3. **Pseudograph:** An undirected graph that can have both loops as well as multiple edges is called a pseudograph. For example, the following graph is a pseudograph.



Degree of a vertex: The degree of a vertex x in an undirected graph is the number of edges in the graph which has x as one of its endpoints with loops counted twice. It is denoted by $\deg(x)$. For example in the following graph, we have $\deg(a) = \deg(c) = \deg(d) = 3$, $\deg(b) = 4$, $\deg(e) = 1$ and $\deg(f) = 0$.



A non increasing sequence that lists all the degrees of vertices in a graph is called the **degree sequence** of that graph. For example, the degree sequence of above graph is $4, 3, 3, 3, 1, 0$.

Regular Graph: A simple graph in which every vertex has the same degree is called a regular graph. If that degree is n, then it is called an n-regular graph.

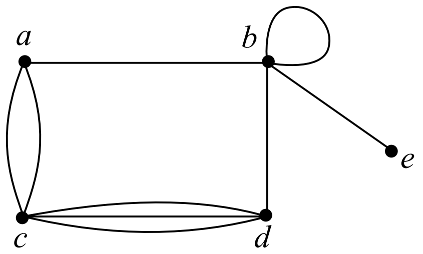
Isolated Vertex and Pendant Vertex: A vertex of degree zero is called an isolated vertex. A vertex of degree one is called a pendant vertex.

Theorem 1 (Handshaking Theorem): Let $G = (V, E)$ be an undirected graph with m edges. Then $2m = \sum\limits_{v\in V} \deg(v)$.

Proof: Since one edge contributes exactly two to the sum of the degrees of vertices, so the sum of the degrees of all the vertices is twice the total number of edges i.e., $2m = \sum\limits_{v\in V} \deg(v)$. □

Example:

1. Verify the handshaking theorem in the graph below:



Solution: In the given graph, we have

$$\text{number of edges } (m) = 9$$

and

$$\begin{gathered}\deg(a) = 3,\\ \deg(b) = 5,\\ \deg(c) = 1,\\ \deg(d) = 4\\ \deg(e) = 5\end{gathered}$$

Therefore,

$$\sum_{v\in V} \deg(v) = 3 + 5 + 5 + 1 + 4 = 18 = 2m.$$

Theorem 2: An undirected graph has an even number of vertices of odd degree.

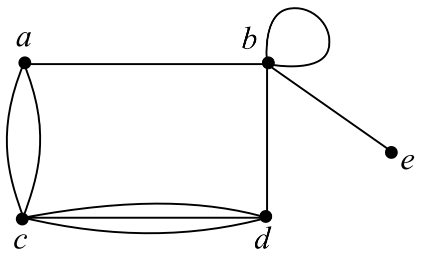
Proof: Let $G = (V, E)$ be an undirected graph. Let V_e be the set of vertices of even degree and V_o be the set of vertices of odd degree. Then by handshaking theorem,

$$2m = \sum_{v\in V} \deg(v) = \sum_{v\in V_e} \deg(v) + \sum_{v\in V_o} \deg(v).$$

But $\deg(v)$ is even for all vertices $v \in V_e$, so $\sum\limits_{v\in V_e} \deg(v)$ is an even number. Therefore, $2m - \sum\limits_{v\in V_e} \deg(v) = \sum\limits_{v\in V_o} \deg(v)$ is an even number, being the difference of two even numbers. Since

$\deg(v)$ is an odd number for all $v \in V_o$, so $\sum_{v\in V_o} \deg(v)$ is an even number which is a sum of odd numbers. Hence there must be an even number of terms in $\sum_{v\in V_o} \deg(v)$ i.e., the number of vertices in V_o must be even. Therefore the number of vertices of odd degree is even. □

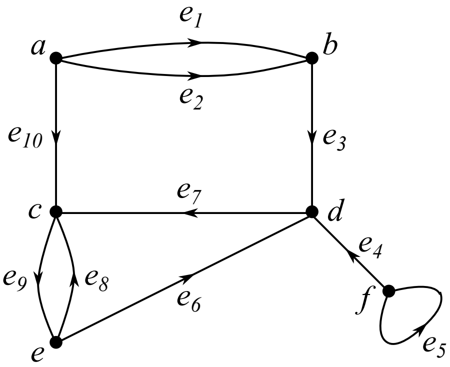
Example:
Here in the graph below, d is the only vertex of even degree and all the other four vertices are of odd degree. So the number of vertices of odd degree is even.



Directed Graph: A **directed graph** (or **digraph**) G is an ordered pair (V, E) where V is a set of vertices and E is a set of directed edges each of which is associated with an ordered pair of vertices.
For example, $G = (V, E)$ where $V = \{a, b, c, d, e, f\}$ and

$$E = \{e_1, e_2, e_3, e_4, e_5, e_6, e_7, e_8, e_9, e_{10}\}$$

is a directed graph as below:



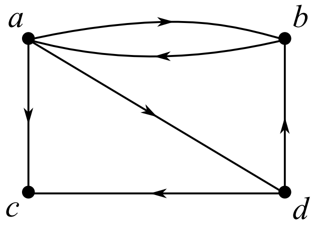
If a directed edge is associated with an ordered pair (a, b), then a is called the **initial vertex** and b is called the **final/terminal vertex** of that edge. Also, we say that a **is adjacent to** b or b **is adjacent from** a.

Loop: A directed edge that has the same initial and terminal vertices is called a loop. For example, e_5 in the above graph is a loop.

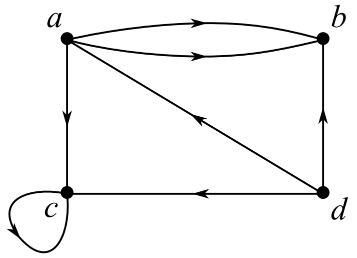
Multiple directed edges: Two or more directed edges are called multiple directed edges if they have the same pair of initial and terminal vertices. For example, e_1 and e_2 are multiple directed edges.

Types of Directed Graphs:

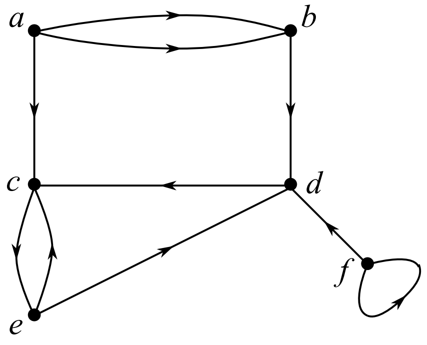
1. **Simple Directed Graph:** A directed graph that has neither loops nor multiple directed edges is called a simple directed graph. For example, the following is a simple directed graph.



2. **Directed Multigraphs:** A directed graph that can have loops as well as multiple directed edges is called a directed multigraph. For example, the following graph is a directed multigraph.



Indegree and Outdegree of a vertex: Let x be a vertex of a directed graph G. Then the indegree of x, denoted by $\deg^-(x)$, is the number of edges with x as their terminal vertex. The outdegree of x, denoted by $\deg^+(x)$, is the number of edges with x as their initial vertex.
For example, in the following directed graph, the indegree and outdegree of each vertex is listed below:



$$\begin{array}{ll}
\deg^-(a) = 0 & \deg^+(a) = 3 \\
\deg^-(b) = 2 & \deg^+(b) = 1 \\
\deg^-(c) = 3 & \deg^+(c) = 1 \\
\deg^-(d) = 3 & \deg^+(d) = 1 \\
\deg^-(e) = 1 & \deg^+(e) = 2 \\
\deg^-(f) = 1 & \deg^+(f) = 2
\end{array}$$

Theorem 3: Let $G = (V, E)$ be a directed graph with m edges. Then

$$\sum_{v\in V} \deg^{-}(v) = \sum_{v\in V} \deg^{+}(v) = m.$$

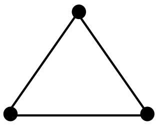
Proof: Since each directed edge has exactly one initial vertex, so each directed edge contributes exactly one to the sum of the indegrees of the vertices. Therefore, $m = \sum_{v\in V} \deg^{-}(v)$. Similarly, each directed edge has exactly one terminal vertex, so each directed edge contributes exactly one to the sum of the outdegrees of the vertices. Therefore $m = \sum_{v\in V} \deg^{+}(v)$. □

For example, in the previous digraph,

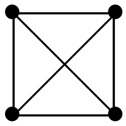
$$\sum_{v\in V} \deg^{-}(v) = \sum_{v\in V} \deg^{+}(v) = m = 10.$$

6.1.2 Graph Types

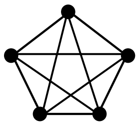
1. Complete Graph: The complete graph on n vertices, denoted by K_n, is the simple graph that has exactly one edge between each pair of distinct vertices. For example, the following graphs are K_3, K_4 and K_5.



K_3



K_4



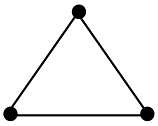
K_5

A complete graph K_n has n vertices. Each of these vertices has degree $n - 1$ and so

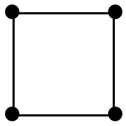
$$\sum_{v\in V} \deg(v) = n(n-1).$$

Therefore by the Handshaking theorem, the number of edges in K_n is $\dfrac{n(n-1)}{2}$.

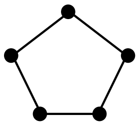
2. Cycle: For $n \geq 3$, the cycle C_n is a simple graph consisting of n vertices $v_1, v_2, \cdots, v_n$ and n edges $\{v_1, v_2\}$, $\{v_2, v_3\}$, $\cdots$, $\{v_{n-1}, v_n\}$ and $\{v_n, v_1\}$. For example,



C_3



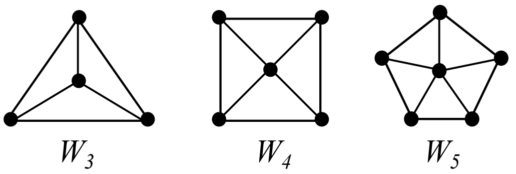
C_4



C_5

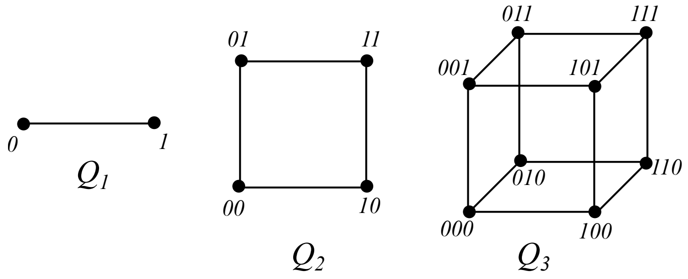
The cycle C_n has n vertices and n edges.

3. Wheel: For $n \geq 3$, the wheel W_n is the simple graph obtained from the cycle C_n by adding one new vertex and n new edges connecting this new vertex to all the other n vertices in C_n. For example,



Note that in a wheel W_n, there are $n + 1$ vertices and $2n$ edges.

4. n-Cube The n-cube, denoted by Q_n, is the graph whose vertices are labeled using length n bit strings and the two vertices joined by an edge if and only if their labels differ in exactly one bit position. For example,



Note that an n-cube Q_n has 2^n vertices. Each of these vertices are connected with n other vertices corresponding to a change in one bit position among n bits. So the degree of each vertex is n and therefore $\sum_{v \in V} \deg(v) = n2^n$. By the Handshaking theorem, we therefore have

$$\frac{n2^n}{2} = n2^{n-1}$$

edges.

5. Bipartite Graphs: Let G be a simple graph. If the vertex set V of G can be divided into two subsets U and W such that

(i) U and W are nonempty,

(ii) $U \cup W = V$,

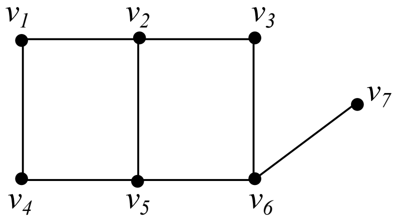
(iii) $U \cap W = \emptyset$ and

(iv) every edge in G connects a vertex in U and a vertex in W

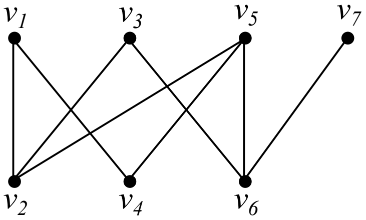
then G is called a bipartite graph.

Examples:

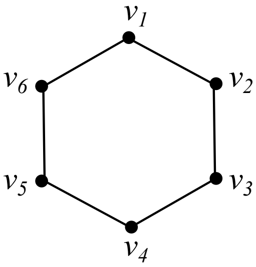
1. The graph below



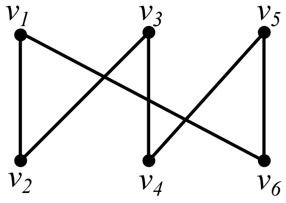
is a bipartite graph because the set of vertices $V = \{v_1, v_2, v_3, v_4, v_5, v_6, v_7\}$ can be partitioned into two subsets $U = \{v_1, v_3, v_5, v_7\}$ and $W = \{v_2, v_4, v_6\}$ such that each edge connects a vertex in U with a vertex in W as below:



2. The cycle C_6 as given below is bipartite.



The set $V = \{v_1, v_2, v_3, v_4, v_5, v_6\}$ can be partitioned into two subsets $U = \{v_1, v_3, v_5\}$ and $W = \{v_2, v_4, v_5\}$ such that each edge connects a vertex in U with a vertex in W as below:



In fact, any cycle C_n with n even, is bipartite.

3. The cycle C_5 is not a bipartite graph. In fact, any cycle C_n with n odd, is not bipartite.

To determine whether a graph $G = (V, E)$ with $V = \{v_1, v_2, \cdots, v_n\}$ is bipartite or not, we follow these steps:

STEP I. Assign a vertex v_1 to the subset U.

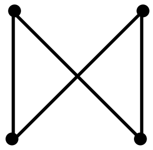
STEP II. Assign all the vertices adjacent to v_1 to the subset W.

STEP III. For a vertex in W, assign all the vertices adjacent to that vertex to the subset V.

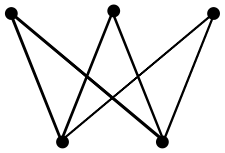
STEP IV. Continue this process until all the vertices have been assigned to either U or W. If the subsets U and W so formed satisfy the required conditions, then G is a bipartite graph.

Theorem 4: A simple graph is bipartite if and only if it is possible to assign two different colors to each vertex of the graph so that no two adjacent vertices are assigned the same color.

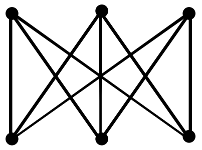
Complete Bipartite Graphs: The complete bipartite graph is a bipartite graph with the partition of the vertex set V into U and W such that every vertex in U is joined to every vertex in W. It is denoted by $K_{m,n}$ where $m = |U|$ and $n = |W|$.
For example, the following graphs are complete bipartite graphs.



$K_{2,2}$



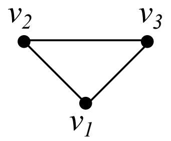
$K_{3,2}$



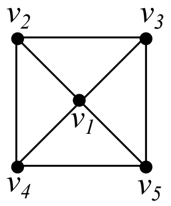
$K_{3,3}$

Subgraph of a graph: A subgraph of a graph $G = (V, E)$ is a graph $H = (W, F)$ where $W \subset V$ and $F \subset E$.

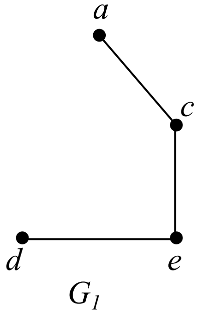
For example



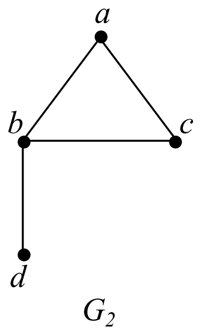
is a subgraph of the graph



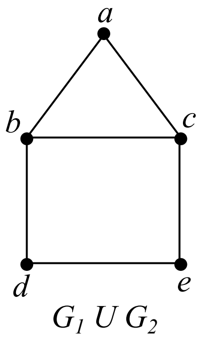
Union of graphs: The union of two simple graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ is the simple graph $G = (V, E)$ where $V = V_1 \cup V_2$ and $E = E_1 \cup E_2$. For example, the union of the graphs G_1 and G_2 below is the graph G.



G_1



G_2



G_1 U G_2

6.1.3 Graph Representation

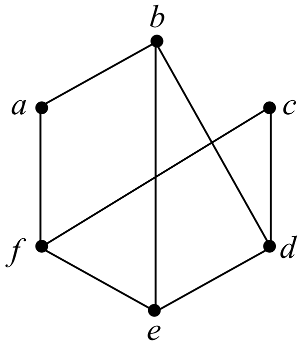
There are three ways to represent graphs:

(i) Adjacency Lists

(ii) Adjacency Matrix

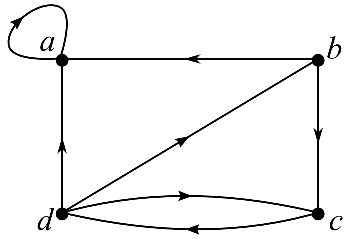
(iii) Incidence Matrix

Adjacency Lists: A graph with no multiple edges can be represented by using adjacency lists which specify the vertices that are adjacent to each vertex of the graph.
For example, the undirected graph below is represented using the adjacency list as follows:



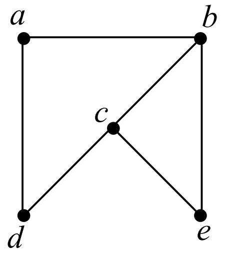
Vertex	Adjacent Vertices
a	b, f
b	a, e, d
c	f, d
d	e, b, c
e	f, b, d
f	a, c, e

Below, a directed graph is represented using the adjacency list:



Initial Vertex	Terminal Vertices
a	a
b	a, c
c	d
d	a, b, c

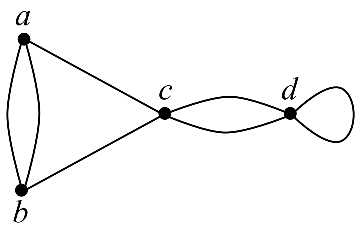
Adjacency Matrices: Let G be an undirected graph with n vertices $v_1, v_2, \cdots, v_n$. Then the adjacency matrix of G is an $n \times n$ matrix $A_G = [a_{ij}]_{n\times n}$ where a_{ij} is the number of edges joining vertices v_i and v_j.
For example, the adjacency matrix of the graph below



is

$$A_G = \begin{array}{c} \\ a \\ b \\ c \\ d \\ e \end{array} \begin{array}{c} \begin{array}{ccccc} a & b & c & d & e \end{array} \\ \begin{pmatrix} 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \end{pmatrix} \end{array}$$

The adjacency matrix of graph



is

$$A_G = \begin{array}{c} \\ a \\ b \\ c \\ d \end{array} \begin{array}{c} \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{pmatrix} 0 & 2 & 1 & 0 \\ 2 & 0 & 1 & 0 \\ 1 & 1 & 0 & 2 \\ 0 & 0 & 2 & 1 \end{pmatrix} \end{array}$$

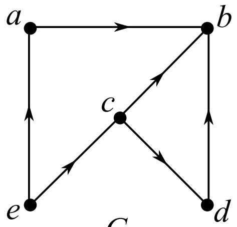
Remark:

1. The adjacency matrix of an undirected graph is always symmetric.

2. The sum of the rows and columns for each vertex are always equal and its value is the degree of that vertex except when that vertex has a loop on it.

Let G be a directed graph with n vertices $v_1, v_2, \cdots, v_n$. Then the adjacency matrix of G is the $n \times n$ matrix $A_G = [a_{ij}]_{n\times n}$ where a_{ij} is the number of edges with initial vertex v_i and the final vertex v_j.

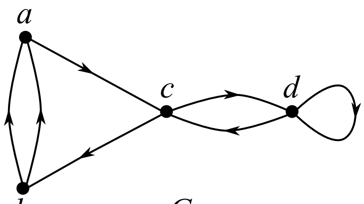
For example, the adjacency matrix of the directed graph G_1 is written as below:



G1

$$A_{G_1} = \begin{array}{c} \\ a \\ b \\ c \\ d \\ e \end{array} \begin{array}{c} \begin{array}{ccccc} a & b & c & d & e \end{array} \\ \begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \end{pmatrix} \end{array}$$

The adjacency matrix of the directed graph G_2 is written as below:



G2

$$A_{G_2} = \begin{array}{c} \\ a \\ b \\ c \\ d \end{array} \begin{array}{c} \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{pmatrix} 0 & 0 & 1 & 0 \\ 2 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{pmatrix} \end{array}$$

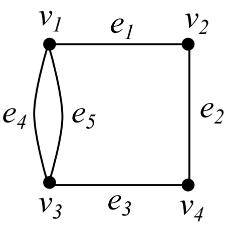
Remark:

1. The adjacency matrix of a directed graph is not necessarily symmetric.
2. The sum of the rows gives the outdegree and the sum of the columns gives the indegree of the corresponding vertex.

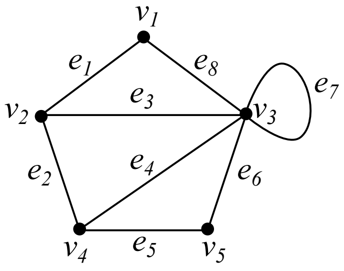
Incidence Matrices: Let G be an undirected graph with n vertices $v_1, v_2, \cdots, v_n$ and m edges $e_1, e_2, \cdots, e_m$. Then the incidence matrix of G is an $n \times m$ matrix $M = [a_{ij}]_{n\times m}$ where

$$a_{ij} = \begin{cases} 1 & \text{if vertex } v_i \text{ is an end vertex of edge } e_j \\ 0 & \text{otherwise} \end{cases}$$

For example, the incidence matrices M_1 and M_2 of the graphs G_1 and G_2 are written below:



G_1



G_2

$$M_1 = \begin{array}{c} \\ v_1 \\ v_2 \\ v_3 \\ v_4 \end{array} \begin{array}{c} \begin{array}{ccccc} e_1 & e_2 & e_3 & e_4 & e_5 \end{array} \\ \begin{pmatrix} 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 \end{pmatrix} \end{array} \qquad M_2 = \begin{array}{c} \\ v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \end{array} \begin{array}{c} \begin{array}{cccccccc} e_1 & e_2 & e_3 & e_4 & e_5 & e_6 & e_7 & e_8 \end{array} \\ \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \end{pmatrix} \end{array}$$

Remark:

1. Each column of an incidence matrix has exactly two 1's except in the column corresponding to a loop in which there is only one 1.
2. The sum of each row equals the degree of that vertex except in the row corresponding to the vertex with a loop in which case the sum is one less than the degree.
3. Parallel edges have identical columns in an incidence matrix.

6.1.4 Graph Isomorphism

Two simple graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are called **isomorphic** if there exists a one-to-one and onto function f from V_1 to V_2 with the property that a and b are adjacent in G_1 if and only if $f(a)$ and $f(b)$ are adjacent in G_2 for all vertices a and b in V_1. Such a function f is called an **isomorphism**.

Invariant Property of a Graph: A property of a graph G is said to be an invariant property if every other graph isomorphic to G also has that property. Some invariant properties of graphs are:

(i) The number of vertices: If two graphs G_1 and G_2 are isomorphic, then by definition of the graph isomorphism, we can conclude that the number of vertices in G_1 and G_2 must be same.

(ii) The number of edges: If two graphs G_1 and G_2 are isomorphic, then again by definition of the graph isomorphism, we can say that the number of edges in G_1 and G_2 must be equal.

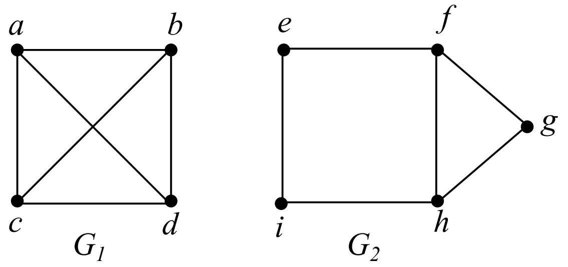
(iii) Degree sequence of a graph: The degree sequence of a graph is the listing of the degrees of all the vertices of that graph written in descending order. If two graphs G_1 and G_2 are isomorphic, then both the graphs must have identical degree sequence.

Apart from these three invariant properties, there is a fourth invariant property which we will be studying in the next section after paths and circuits have been defined.

Note: If two graphs G_1 and G_2 are isomorphic, then by definition of the invariant property, G_1 and G_2 both must have the same invariant properties. Hence, by contrapositive argument, we can say that if G_1 and G_2 differ in any one of the invariant properties, then G_1 and G_2 cannot be isomorphic.

Problems:
Determine whether the following graphs are isomorphic or not.

A.

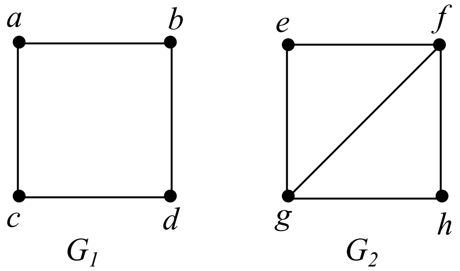


Solution: Here,

$$\text{number of vertices in } G_1 = 4$$
$$\text{number of vertices in } G_2 = 5.$$

Since G_1 and G_2 have different number of vertices, so G_1 and G_2 are not isomorphic.

B.

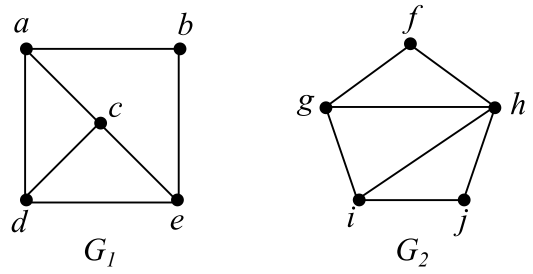


Solution: Here,

$$\text{number of edges in } G_1 = 4$$
$$\text{number of edges in } G_2 = 5.$$

Since G_1 and G_2 have different number of edges, so G_1 and G_2 are not isomorphic.

C.

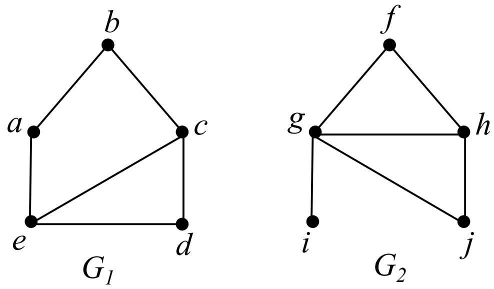


Solution: Here,

$$\text{degree sequence of } G_1 \text{ is } 3,3,3,3,2$$
$$\text{degree sequence of } G_2 \text{ is } 4,3,3,2,2.$$

Since G_1 and G_2 have different degree sequences, so G_1 and G_2 are not isomorphic.

D.

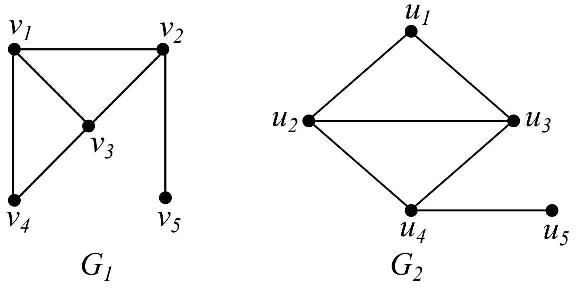


Solution: Here,

$$\text{degree sequence of } G_1 \text{ is } 3,3,2,2,2$$
$$\text{degree sequence of } G_2 \text{ is } 4,3,2,2,1.$$

Since G_1 and G_2 have different degree sequences, so G_1 and G_2 are not isomorphic.

E.



Solution: Here, $V_1 = \{v_1, v_2, v_3, v_4, v_5\}$ and $V_2 = \{u_1, u_2, u_3, u_4, u_5\}$. Let $f : V_1 \to V_2$ be a function defined as follows:

$$f(v_1) = u_2$$
$$f(v_2) = u_4$$
$$f(v_3) = u_3$$
$$f(v_4) = u_1$$
$$f(v_5) = u_5$$

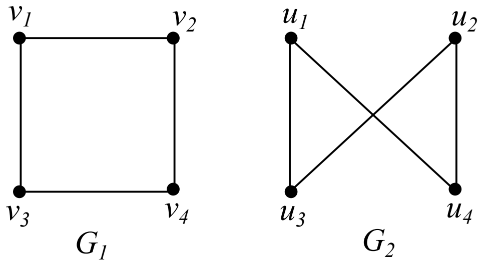
Then clearly, f is a bijective function. Also

$$A_{G_1} = \begin{array}{c c} & \begin{array}{ccccc} v_1 & v_2 & v_3 & v_4 & v_5 \end{array} \\ \begin{array}{c} v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \end{array} & \begin{pmatrix} 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{pmatrix} \end{array}$$

$$A_{G_2} = \begin{array}{c c} & \begin{array}{ccccc} u_2 & u_4 & u_3 & u_1 & u_5 \end{array} \\ \begin{array}{c} u_2 \\ u_4 \\ u_3 \\ u_1 \\ u_5 \end{array} & \begin{pmatrix} 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{pmatrix} \end{array}$$

Since G_1 and G_2 have identical adjacency matrices as well, so G_1 and G_2 are isomorphic graphs.

F.



Solution: Here, $V_1 = \{v_1, v_2, v_3, v_4\}$ and $V_2 = \{u_1, u_2, u_3, u_4\}$. Let $f : V_1 \to V_2$ be a function defined as follows:

$$\begin{aligned} f(v_1) &= u_1 \\ f(v_2) &= u_4 \\ f(v_3) &= u_3 \\ f(v_4) &= u_2 \end{aligned}$$

Then clearly, f is a bijective function. Also

$$A_{G_1} = \begin{array}{c c} & \begin{array}{cccc} v_1 & v_2 & v_3 & v_4 \end{array} \\ \begin{array}{c} v_1 \\ v_2 \\ v_3 \\ v_4 \end{array} & \begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{pmatrix} \end{array}$$

$$A_{G_2} = \begin{array}{c c} & \begin{array}{cccc} u_1 & u_4 & u_3 & u_2 \end{array} \\ \begin{array}{c} u_1 \\ u_4 \\ u_3 \\ u_2 \end{array} & \begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{pmatrix} \end{array}$$

Since G_1 and G_2 have identical adjacency matrices as well, so G_1 and G_2 are isomorphic graphs.

6.1.5 Connectivity in Graphs

Paths and Circuits in Undirected Graphs:

Path: Let G be an undirected graph with vertices u and v. Then a path between u and v is a sequence of edges $e_1, e_2, \cdots, e_n$ of G such that e_1 is associated with $\{x_0, x_1\}$, e_2 is associated with $\{x_1, x_2\}$, $\cdots$, e_n is associated with $\{x_{n-1}, x_n\}$, where $x_0 = u$ and $x_n = v$. The length of a path is defined to be the number of edges in that path.

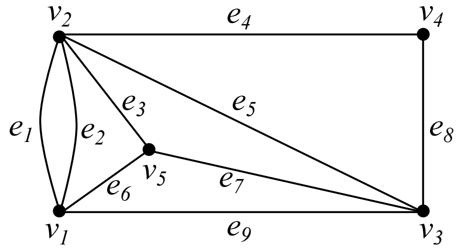
Circuit: A circuit is a path of length greater than zero that begins and ends at the same vertex.

Simple Path: A path is called a simple path if it does not contain the same edge more than once.

Simple Circuit: A circuit is called a simple circuit if it does not contain the same edge more than once.

Examples:

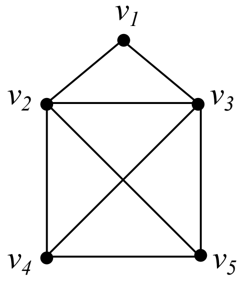
1. Let G be the following graph.



Then e_3, e_5, e_7, e_3, e_4 is a path of length 5 between the pair of vertices v_5, v_4 whereas e_3, e_5, e_8 is a simple path of length 3 between v_5 and v_4. Also, $e_1, e_5, e_9, e_2, e_5, e_7, e_6$ is a circuit that starts and ends in the vertex v_1 and e_1, e_3, e_7, e_9 is a simple circuit that also starts and ends in the vertex v_1.

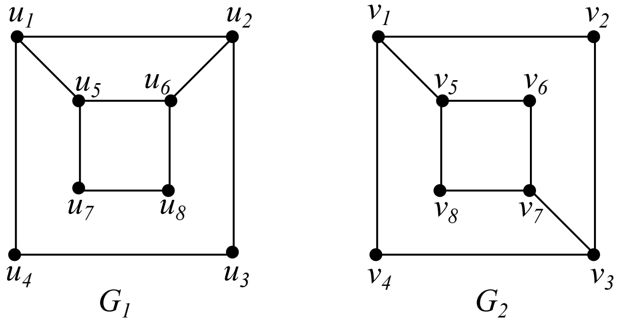
Note: When the graph G is simple, then paths or circuits in G can be specified by a sequence of vertices rather than edges as in the example below. This is because, in a simple graph, there can be at most one edge between any pair of vertices.

2. Let G be the simple graph as below:



Then $v_1, v_3, v_5, v_2, v_1, v_3$ is a path of length 5 between the vertices v_1 and v_3 and v_1, v_2, v_5, v_3 is a simple path of length 3 between v_1 and v_3. Also, $v_1, v_2, v_3, v_4, v_2, v_1$ is a circuit of length 5 that starts and ends in v_1 and $v_1, v_2, v_5, v_4, v_3, v_1$ is a simple circuit of length 5 that starts and ends in v_1.

One more invariant: The number of simple circuits of length k in a graph G is an invariant property of that graph. This invariant property can be used to show that two graphs G_1 and G_2 are not isomorphic.
For example, the following graphs G_1 and G_2 are not isomorphic because G_1 has three simple circuits of length 4 but G_2 has only two.



Paths and Circuits in Directed Graphs:

Path: Let G be a directed graph with vertices u and v. Then a path from u to v is a sequence of edges $e_1, e_2, \cdots, e_n$ of G such that e_1 is associated with (x_0, x_1), e_2 is associated with (x_1, x_2), $\cdots$, e_n is associated with (x_{n-1}, x_n) where $x_0 = u$ and $x_n = v$. As for paths in undirected graphs, the number of edges in a path is called its length.

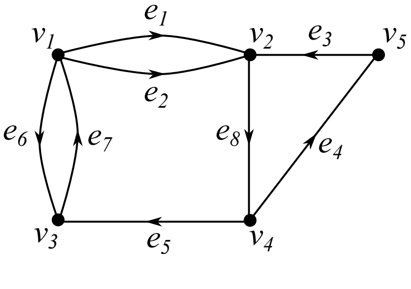
Circuit: A circuit is a path of length greater than zero that begins and ends at the same vertex.

Simple Path: A path is called a simple path if it does not contain the same edge more than once.

Simple Circuit: A circuit is called a simple circuit if it does not contain the same edge more than once.

Examples:

1. Let G be a directed graph as below:



Then e_5, e_7, e_6, e_7, e_2 is a path of length 5 from vertex v_4 to vertex v_2 and e_7, e_1, e_8, e_4 is a simple path of length 4 from vertex v_3 to vertex v_5 but e_7, e_1, e_3 is not a path. Also,

$e_1, e_8, e_5, e_7, e_6, e_7$ is a circuit of length 6 starting and ending in vertex v_1 whereas e_8, e_4, e_3 is a simple circuit of length 3 starting and ending in vertex v_2.

Note: When the graph G is a simple directed graph, then paths or circuits in G can be specified by a sequence of vertices rather than edges because in a simple digraph, there can be at most one edge between any pair of vertices.

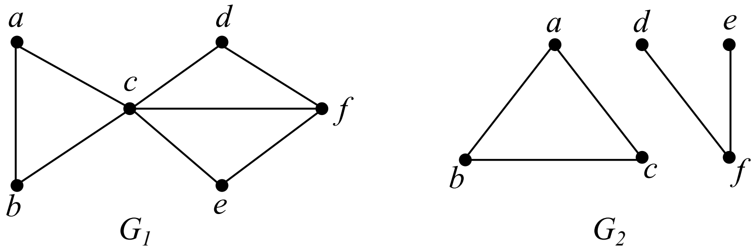
Counting paths between vertices:
Given a graph with n vertices $v_1, v_2, \cdots, v_n$, we can determine the number of paths between any two vertices v_i and v_j (or from v_i to v_j in case of digraphs) using its adjacency matrix A. The number of paths of length r in the graph will be the ij^{th} entry in the matrix A^r. **Example:**

1. Find the number of paths between the vertices v_1 and v_5 of length 3 in the following graph.

2. Find the number of paths from v_1 to v_4 of length 4 in the following digraph.

Connectedness in Undirected Graphs:

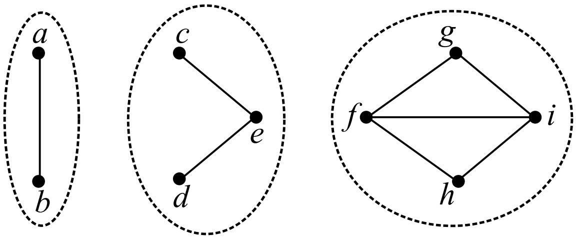
Connected Undirected Graphs: An undirected graph G is said to be connected if there is a path between every pair of distinct vertices in the graph. For example, G_1 is a connected graph and G_2 is not a connected graph.



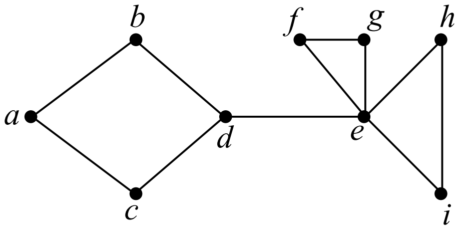
Theorem 1: There is a simple path between every pair of distinct vertices of a connected undirected graph.

Proof: Let u and v be two distinct vertices of the connected undirected graph G. Since G is connected, there is at least one path between u and v. Let $x_0, x_1, \cdots, x_n$ where $x_0 = u$ and $x_n = v$ be a path of least length. We claim that this path of least length is a simple path. To prove this, suppose that this path is not simple. Then $x_i = x_j$ for some i and j with $0 \leq i < j$. Then $x_0, x_1, \cdots, x_{i-1}, x_j, \cdots, x_n$ is a path from u to v obtained from the path $x_0, x_1, \cdots, x_n$ by removing the vertices $x_i, x_{i+1}, \cdots, x_{j-1}$. So the path $x_0, x_1, \cdots, x_{i-1}, x_j, \cdots, x_n$ is a path of shorter length than the path $x_0, x_1, \cdots, x_n$ i.e., $x_0, x_1, \cdots, x_n$ is not a path of shortest length. Hence (by indirect proof method) the path $x_0, x_1, \cdots, x_n$ from u to v must be a simple path. □

Connected Components: A connected component of a graph G is a connected subgraph of G that is not a proper subgraph of another connected subgraph of G. In other words, a maximal connected subgraph of G is called its connected component.
For example, the graph below has three connected components:

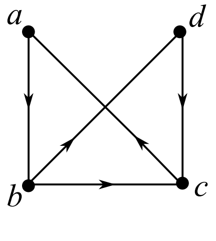


Cut Vertices and Cut Edges: A vertex in G is said to be a cut vertex if removing that vertex and all the edges incident on it produces a subgraph of G with more connected components than in G.
An edge in G is said to be a cut edge or bridge if removing that edge produces a disconnected subgraph of G.
For example, in the graph below, d and e are cut vertices and $\{d, e\}$ is a cut edge.

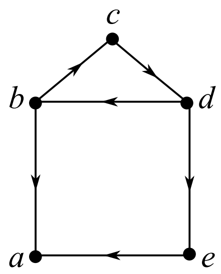


Connectedness in Directed Graphs:

Strongly Connected Directed Graphs: A directed graph is said to be strongly connected if there is a path from a to b and from b to a for every pair of vertices a, b in the graph.
For example, the following directed graph is strongly connected.



Weakly Connected Directed Graphs: A directed graph is said to be weakly connected if there is a path between every two vertices in the underlying undirected graph i.e., if the underlying undirected graph is connected.
For example, the following graph is weakly connected.

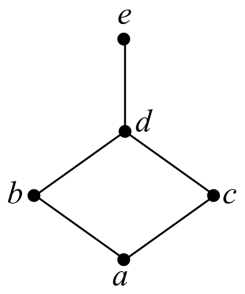


This graph is not strongly connected because there is no path from a to b but it is a weakly connected graph because the underlying undirected graph is connected.

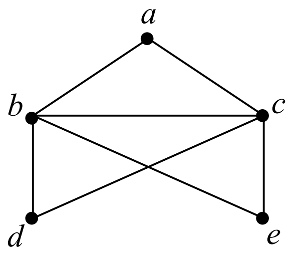
6.1.6 Euler and Hamiltonian Paths and Circuits

Euler Paths and Circuits:

Euler Path: An Euler path in a graph G is a simple path that contains every edge of G. For example, in the graph G below, e, d, b, a, c, d is an Euler path.

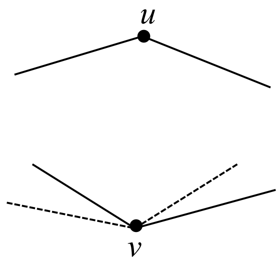


Euler Circuit: An Euler circuit in a graph G is a simple circuit that contains every edge of G. For example, the graph below has an Euler circuit a, b, c, d, b, e, c, a.



Theorem 1: A connected multigraph has an Euler circuit if and only if each of its vertices has even degree.

Proof: Suppose that a connected multigraph G has an Euler circuit. We need to prove that each of the vertices of G has even degree. Now suppose that the Euler circuit in G start and ends at a vertex u. If v is any vertex of G different from u then v must be on the Euler circuit because G is connected and the Euler circuit contains every edge of G. Moreover, each time v occurs in the Euler circuit, it enters and leaves v by different edges because each edge of G occurs only once in the Euler circuit. Thus each occurrence of v in the Euler circuit adds two to $deg(v)$ i.e., $deg(v)$ is even. Finally, since the Euler circuit must end at u, the first and last edges add two to $deg(u)$ as will any intermediate occurrence if u in the Euler circuit. So $deg(u)$ is also even.

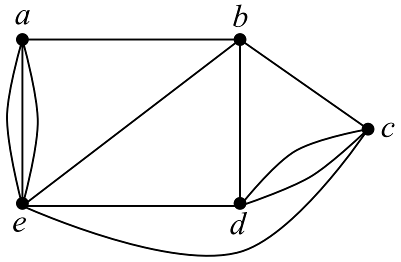


Conversely suppose that a connected multigraph G has all its vertices of even degree. We need to show that G has an Euler circuit. So let u be any arbitrary vertex of G. Starting from u, we construct a simple path that is as long as possible. Since every vertex is of even degree, we can exit from every vertex we enter so the path can only stop at vertex u. We then have a simple circuit in G. If this simple circuit contains all the edges of G, then this is an Euler circuit as required. If not, then consider a subgraph H of G obtained by removing all the edges in the circuit and the resulting isolated vertices if any. Since both G and the simple circuit have all their vertices of even degree, so the degrees of the vertices of H are also even. Also, since G is connected, H has at least one vertex in common with the simple circuit that was removed. Let v be that vertex. Starting at the vertex v, we can again construct a new simple path. Since all the vertices of H are of even degree, this path must terminate at vertex v thus forming a simple circuit. Now this simple circuit can be combined with the previous simple circuit to obtain a larger simple circuit that starts and ends at vertex u. This process is continued until one obtains a simple circuit that contains all the edges of G. We would then have an Euler circuit in G. □

Problems:

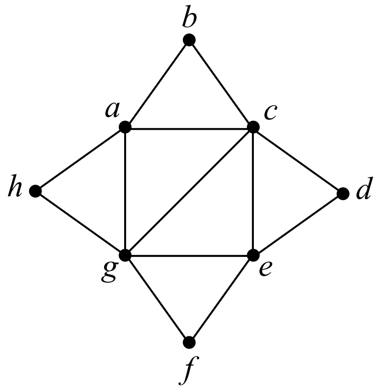
Which of the following graphs have an Euler circuit?

(a)



Solution: Here $\deg(a) = \deg(b) = \deg(c) = \deg(d) = 4$ and $\deg(e) = 6$. So the degree of all the vertices are even and hence the graph must have an Euler circuit. e.g., $abcdeaebdcea$

(b)



Solution: Here $\deg(b) = \deg(d) = \deg(f) = \deg(h) = 2$, $\deg(a) = \deg(e) = 4$ and $\deg(c) = \deg(g) = 5$. So this graph has vertices of odd degree and hence it cannot have an Euler circuit.

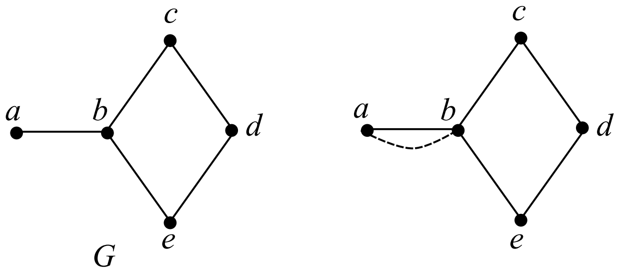
Algorithm for constructing Euler circuits:

```
PROCEDURE Euler ($G$: connected multigraph with all vertices of even degree)

    Circuit:= a circuit in $G$ beginning at an arbitrarily chosen vertex with edges successively added to form a path that returns to this vertex
    $H$:= $G$ with edges of circuit removed
    WHILE $H$ has edges

        BEGIN

            Subcircuit:= a circuit in $H$ beginning at a vertex in $H$ that also is an endpoint of an edge of Circuit
            $H$:= $H$ with edges of Subcircuit and all isolated vertices removed
            Circuit:= Circuit with Subcircuit inserted at the appropriate vertex

        END
```

Circuit is the required Euler circuit.

Theorem 2: A connected multigraph has an Euler path but not an Euler circuit if and only if it has exactly two vertices of odd degree.

Proof: Let G be a connected multigraph that doesn't have an Euler circuit but has an Euler path from a to b. If we join the vertices a and b with the edge $\{a, b\}$, then this Euler path becomes an Euler circuit and the degrees of a and b increases by one whereas the degrees of other vertices remains unchanged. By Theorem 1, the degree of all vertices of this newly formed graph must be even i.e., the degree of a and b in this new graph must be even i.e., the degree of a and b in the original graph must have been odd and the degree of all other vertices must be even. So G has exactly two verices a and b of odd degree.



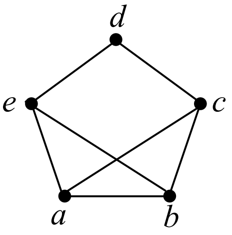
Conversely suppose that a connected multigraph G has exactly two vertices a and b of odd degree. If the vertices a and b are joined by another edge $\{a, b\}$ then this new graph has all the vertices of even degree and so by Theorem 1, there must be an Euler circuit in this new graph. If the newly added edge $\{a, b\}$ is removed from this Euler circuit, then it becomes an Euler path from vertex a to vertex b in the original graph G. □

Theorem 3: A connected multigraph has an Euler path if and only if it has either no vertices of odd degree or exactly two vertices of odd degree.

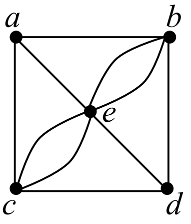
Proof: Let G be a connected multigraph which has an Euler path. If this Euler path is an Euler circuit as well, then by Theorem 1, G has no vertices of odd degree. If this Euler path is not an Euler circuit, then by Theorem 2, G has exactly 2 vertices of odd degree.

Conversely suppose that a connected multigraph G has no vertices of odd degree. Then all the vertices of G has even degree and so by Theorem 1, G must have an Euler circuit and hence an Euler path. If G has exactly two vertices of odd degree, then G must have an Euler path by Theorem 2. □

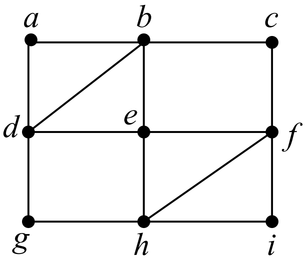
Problems:
Which of the following graphs has an Euler path?



(i)



(ii)



(iii)

Solution:

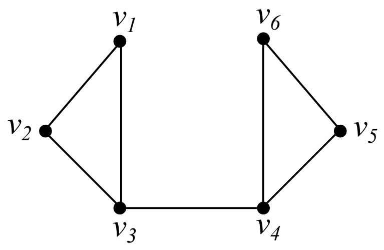
(i) Here $\deg(a) = \deg(b) = \deg(c) = \deg(e) = 3$. So this graph does not have an Euler path because it has more that two vertices of odd degree.

(ii) This graph has exactly two vertices, a and d, of odd degree. So it has an Euler path.

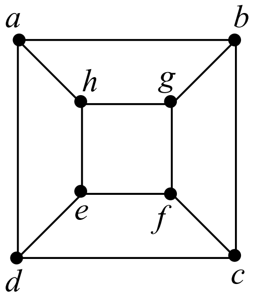
(iii) Here, $\deg(a) = \deg(c) = \deg(i) = \deg(g) = 2$ and $\deg(b) = \deg(f) = \deg(h) = \deg(d) = \deg(e) = 4$. So this graph has no vertices of odd degree and therefore it has an Euler path.

Hamiltonian Paths and Circuits:

Hamiltonian Path: A Hamiltonian path in a graph G is a simple path that passes through every vertex of G exactly once.
For example, $v_1v_2v_3v_4v_5v_6$ is a Hamiltonian path in the graph below.



Hamiltonian Circuit: A Hamiltonian circuit in a graph G is a simple circuit that passes through every vertex of G exactly once except the starting vertex which must also be the final vertex.
For example, in the graph below, $abgfcdeha$ is a Hamiltonian circuit.



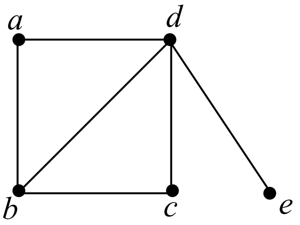
Proposition 1: The complete graph K_n has a Hamiltonian circuit whenever $n \geq 3$.

Proof: Let K_n, $n \geq 3$ be a complete graph of n vertices $v_1, v_2, \cdots, v_n$. Since there is an edge joining v_i and v_j for each distinct i and j, so the cycle $v_1 v_2 \cdots v_n v_1$ is a subgraph of K_n. Clearly, this cycle is a Hamiltonian circuit of K_n. □

Proposition 2: A graph with vertex of degree one cannot have a Hamiltonian circuit.

Proof: Suppose u is a vertex of degree one. Then whenever a path reaches this vertex, there is no way to leave this vertex through a different edge. So this graph cannot have a Hamiltonian circuit. □

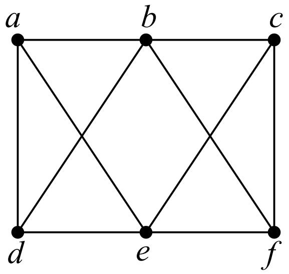
For example, the following graph below cannot have a Hamiltonian circuit.



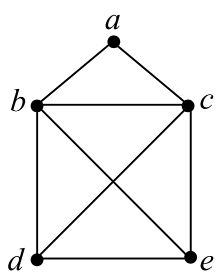
Dirac's Theorem: (Statement only) If G is a simple graph with n vertices, $n \geq 3$, such that the degree of every vertex in G is at least $\left\lceil \frac{n}{2} \right\rceil$, then G has a Hamiltonian circuit.

Ore's Theorem: (Statement only) If G is a simple graph with n vertices, $n \geq 3$, such that $\deg(u) + \deg(v) \geq n$ for every pair of nonadjacent vertices u and v in G, then G has a Hamiltonian circuit.

Problems:
Determine whether the following graphs have Hamiltonian circuits:



(i)



(ii)

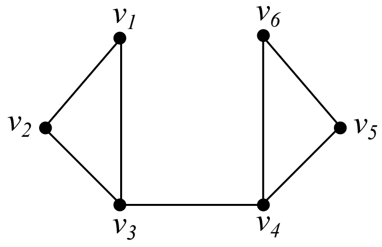
Solution:

(i) In this graph, the number of vertices $n = 6$ and so $\left\lceil \frac{n}{2} \right\rceil = 3$. Now $\deg(a) = \deg(c) = \deg(f) = \deg(d) = 3$ and $\deg(b) = \deg(e) = 4$. So the degree of all the vertices is at least $\left\lceil \frac{n}{2} \right\rceil = 3$ and so by Dirac's Theorem, this graph must have a Hamiltonian circuit.

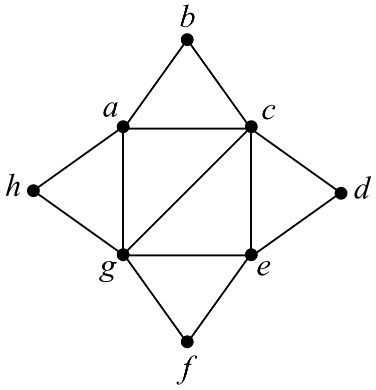
(ii) Here, the number of vertices $n = 5$. Now the nonadjacent pairs of vertices in this graph are $\{a, d\}$ and $\{a, e\}$. So $\deg(a) + \deg(d) = 2 + 3 = 5$, $\deg(a) + \deg(e) = 2 + 3 = 5$ i.e., $\deg(u) + \deg(v) \geq n$ for all nonadjacent pairs of vertices u and v in G. So by Ore's Theorem, G must have a Hamiltonian circuit.

6.1.7 Matching Theory

A **matching** in a simple graph is a subset of the set of edges of the graph such that no two edges in the set share a common vertex. In other words, a matching is a subset of edges such that if $\{s, t\}$ and $\{u, v\}$ are edges of the matching, then s, t, u, v are all distinct. A **maximal matching** of a graph is a matching with the largest number of edges.
For example, in the graph below,



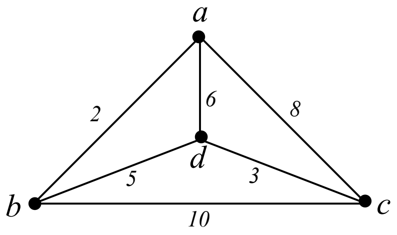
a subset consisting of edges $\{v_1, v_2\}$ and $\{v_4, v_5\}$ is a matching where as a subset consisting of edges $\{v_1, v_2\}$ and $\{v_2, v_3\}$ is not a matching. The subset consisting of edges $\{v_1, v_2\}$, $\{v_3, v_4\}$ and $\{v_5, v_6\}$ is a maximal matching. The graph below



has the subset consisting of edges $\{a, b\}$, $\{c, d\}$, $\{e, f\}$, $\{g, h\}$ as a maximal matching.

6.1.8 Shortest Path Algorithm

Weighted graph: Graphs that have a number assigned to each edge are called weighted graphs. For example,

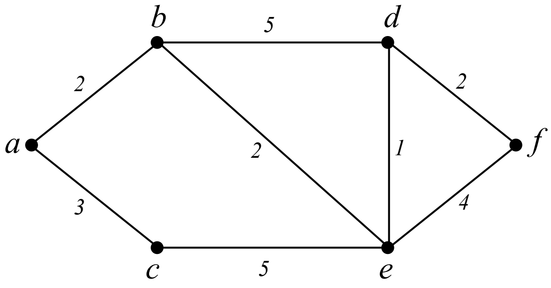


The number assigned to an edge e is called its weight and is denoted by $w(e)$. So in the above figure, $w(a, b) = 2$, $w(c, d) = 3$ etc.

Length of a path: The length of a path in a weighted graph is the sum of all the weights of the edges in that path. For example, in the previous weighted graph, the length of the path a, d, b, c is $6 + 5 + 10 = 21$.

Weighted graphs can arise in many situations while modeling real-world problems such as finding the path of shortest distance from one city to another while traveling or finding the best route to transfer data from one computer to another in a computer network.

Shortest-Path Problem: Given a weighted graph G, the problem of finding a path of least length between two of its vertices is called the shortest-path problem.
For example, in the graph below, we can see that the shortest path from a to f is a, b, e, d, f whose length is 7.



Dijkstra's Algorithm for solving shortest-path problem:
Dijkstra's algorithm is used for finding a path of shortest length between two given vertices in a weighted connected simple graph where all the weights are positive.

Dijkstra's Algorithm:
PROCEDURE Dijkstra (G: weighted connected simple graph, with all weights positive)
$\{G$ has vertices $a = v_0, v_1, \cdots, v_n = z$ and weights $w(v_i, v_j)$ where $w(v_i, v_j) = \infty$ if $\{v_i, v_j\}$ is not an edge in $G\}$

```
    FOR $i := 1$ TO $n$
        $L(v_i) := \infty$
```

```
$L(a) := 0$
$S := \emptyset$
$\{$The labels are now initialized so that the label of $a$ is $0$ and all other labels are $\infty$
and $S$ is the empty set.$\}$
WHILE $z \notin S$
    BEGIN
        $u :=$ a vertex not in $S$ with $L(u)$ minimal.
        $\{$If there is more than one vertex with same minimum label
        $L(u)$, then select one arbitrarily.$\}$ $S := S \cup \{u\}$
        FOR all vertices $v$ not in $S$ and adjacent to $u$
            IF $L(u) + w(u,v) < L(v)$ THEN $L(v) = L(u) +$
            $w(u,v)$
        $\{$This adds a vertex to $S$ with minimal label and updates the
        labels of the vertices not in $S$.$\}$
    END
$\{$Length of the shortest path from $a$ to $z$ is $L(z)$.$\}$
```

Description of Dijkstra's Algorithm:
Suppose we are given a weighted connected simple graph G with the weight of all the edges positive. Let a and z be two vertices in G and suppose that we have to find the path of shortest length from a to z.
Dijkstra's algorithm to find this path of shortest length proceeds iteratively by first finding the length of a shortest path from a to the first vertex, then the length of a shortest path from a to the second vertex, and so on, until the length of a shortest path from a to z is found. For this, the algorithm labels each vertex v of G by $L(v)$ which denotes the length of path from a to v. It also maintains a set S of vertices whose final shortest path length from the vertex a have already been determined.
The algorithm proceeds stepwise as follows:

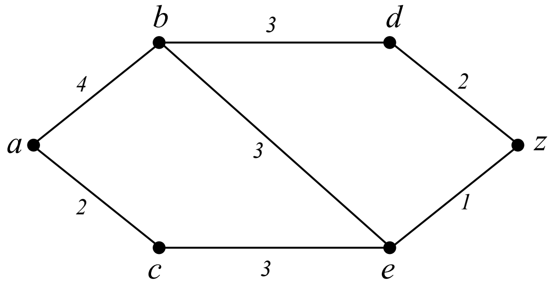
STEP 1: Set $L(a) = 0$ and $L(v) = \infty$ for all other vertices.

STEP 2: If z is selected, then stop; otherwise proceed to next step.

STEP 3: Let u be an unselected vertex such that $L(u)$ is the minimum. (If there are more than one such vertices, select one arbitrarily.) Select u and for each unselected vertex v adjacent to u, if $L(u) + w(u,v) < L(v)$, then change the value of $L(v)$ to $L(u) + w(u,v)$, otherwise don't change the value of $L(v)$. Proceed to STEP 2.

The algorithm terminates when z is a selected and the value of $L(z)$ at that time is the length of a shortest-path from a to z.

Examples:
(i) Find the path of shortest length from a to z in the following weighted connected simple graph:

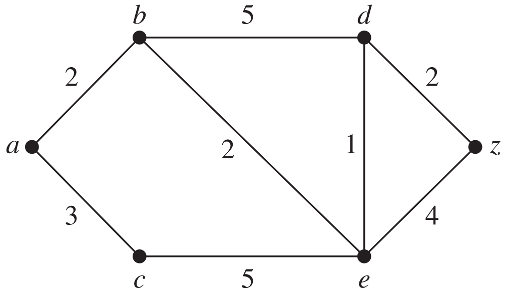


Solution:

$L(a)$	$L(b)$	$L(c)$	$L(d)$	$L(e)$	$L(z)$
$\boxed{0}$	∞	∞	∞	∞	∞
$\boxed{0}$	4 (a)	$\boxed{2}$ (a)	∞	∞	∞
$\boxed{0}$	$\boxed{4}$ (a)	$\boxed{2}$ (a)	∞	5 (a, c)	∞
$\boxed{0}$	$\boxed{4}$ (a)	$\boxed{2}$ (a)	7 (a, b)	$\boxed{5}$ (a, c)	∞
$\boxed{0}$	$\boxed{4}$ (a)	$\boxed{2}$ (a)	7 (a, b)	$\boxed{5}$ (a, c)	$\boxed{6}$ (a, c, e)

Hence the shortest path from a to z is $acez$ with length 6.

(ii) Find the path of shortest length from a to all other vertices in the following weighted connected simple graph:



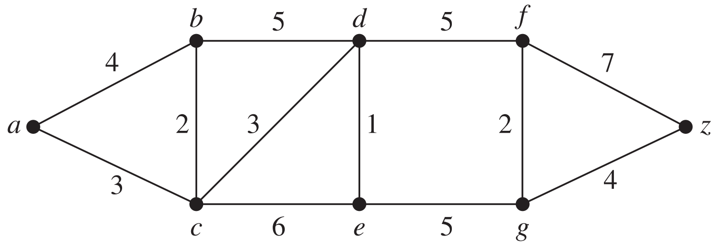
Solution:

$L(a)$	$L(b)$	$L(c)$	$L(d)$	$L(e)$	$L(z)$
$\boxed{0}$	∞	∞	∞	∞	∞
$\boxed{0}$	$\boxed{2}$ (a)	3 (a)	∞	∞	∞
$\boxed{0}$	$\boxed{2}$ (a)	$\boxed{3}$ (a)	7 (a,b)	4 (a,b)	∞
$\boxed{0}$	$\boxed{2}$ (a)	$\boxed{3}$ (a)	7 (a,b)	$\boxed{4}$ (a,b)	∞
$\boxed{0}$	$\boxed{2}$ (a)	$\boxed{3}$ (a)	$\boxed{5}$ (a,b,e)	$\boxed{4}$ (a,b)	8 (a,b,e)
$\boxed{0}$	$\boxed{2}$ (a)	$\boxed{3}$ (a)	$\boxed{5}$ (a,b,e)	$\boxed{4}$ (a,b)	$\boxed{7}$ (a,b,e,d)

Hence the shortest path from

a to b is ab with length 2
a to c is ac with length 3
a to d is $abed$ with length 5
a to e is abe with length 4
a to z is $abedz$ with length 7.

(iii)



Solution:

6.1.9 Traveling Salesman Problem

Traveling Salesman Problem (TSP): Given n number of cities and the distance between each pair of those cities, the traveling salesman problem is to find a path that the salesman should travel so as to visit every city precisely once and return home, with the minimum distance traveled. Therefore in graph-theoretic terms, the traveling salesman problem is equivalent to finding a Hamiltonian circuit that has minimum total weight in a weighted complete undirected graph.

Solving TSP: The total number of different Hamiltonian circuits in a complete undirected graph of n vertices is $\dfrac{(n-1)!}{2}$. So, theoretically, the TSP can always be solved by finding these $\dfrac{(n-1)!}{2}$ different Hamiltonian circuits, finding the total weight of each of those circuits and then choosing a circuit with the least weight.

But practically, this method is very time consuming because for large n, finding $\dfrac{(n-1)!}{2}$ different Hamiltonian circuits is very inefficient. For example, if $n = 25$, then

$$\frac{(n-1)!}{2} = \frac{24!}{2} \approx 3.1 \times 10^{23}.$$

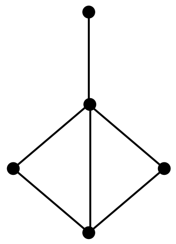
Assuming that it take just one nanosecond (10^{-9} second) to examine each Hamiltonian circuit, a total of approximately ten million years would be required to find a minimum-length Hamiltonian circuit.

Therefore TSP are practically solved using approximation algorithms which do not necessarily produce the exact solution to the problem but instead produce a solution that is close to the exact solution in a reasonable period of time.

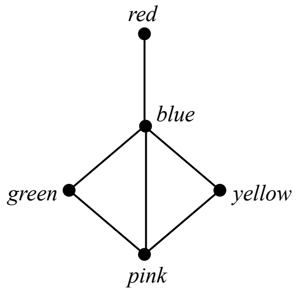
6.1.10 Graph Coloring

Graph Coloring: The assignment of a color to each vertex of a simple graph so that no two adjacent vertices are assigned the same color is called the graph coloring or coloring of the graph.

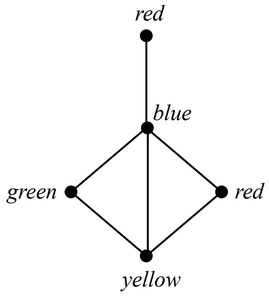
For example, in the figures below, (ii), (iii) and (iv) are the graph colorings of the graph is figure (i).



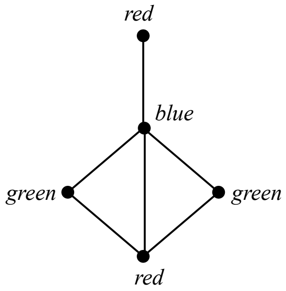
(i)



(ii)
5 colors used



(iii)
4 colors used



(iv)
3 colors used

Chromatic Number: The chromatic number of a graph G, denoted by $\chi(G)$, is the least number of colors needed for a coloring of the graph.
For example, the chromatic number of the above graph is 3.

Chromatic number of some common graphs:

1. $\chi(K_n)$: Each vertex of a complete graph K_n is connected with every other vertex. So if a color is used for one vertex of the graph, that color cannot be reused for any other vertex. So we need at least n colors to properly color K_n. Hence $\chi(K_n) = n$.

2. $\chi(C_n)$: A cycle C_n needs either two colors or three colors to color its vertices depending upon whether n is even or odd. So

$$\chi(C_n)=\begin{cases} 2 & \text{if } n \text{ is even} \\ 3 & \text{if } n \text{ is odd.} \end{cases}$$

3. $\chi(W_n)$: A wheel graph W_n has one more vertex that C_n and this vertex is connected with all the other vertices. So

$$\chi(W_n)=\begin{cases} 3 & \text{if } n \text{ is even} \\ 4 & \text{if } n \text{ is odd.} \end{cases}$$

4. χ(Bipartite graph): If the vertex set V of a bipartite graph G is partitioned into two subsets U and W, then the vertices in U can be given the same color since none of the vertices in U are adjacent. Similarly the vertices in W can be given the same color. Therefore $\chi(G) = 2$ for any bipartite graph G.

Applications of Graph Coloring

Scheduling Exams:

Graph coloring can be used to schedule the exams so that no student has two exams at the same time and also the exams are completed in the minimum amount of time possible. For this we proceed as follows:

(1) Represent the courses by vertices.

(2) If there is a common student in the courses, then join the corresponding vertices by an edge.

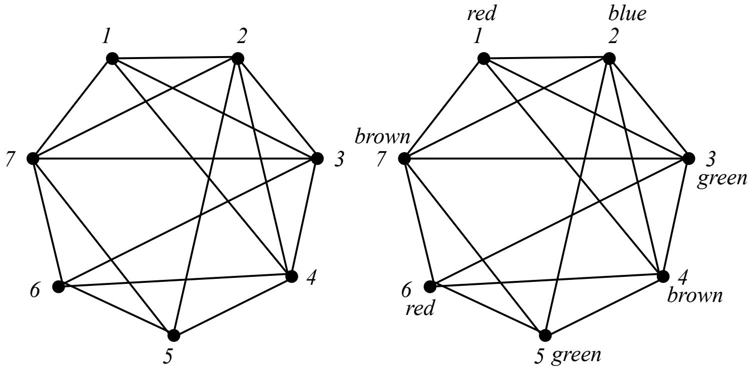
(3) Find a coloring of this associated graph.

(4) Schedule the exam such that each time slot for the exam is represented by a different color.

Examples:

1. Suppose that there are seven courses which are numbered as $1, 2, 3, 4, 5, 6, 7$ and suppose that the following courses have common students:
   $\{1, 2\}$, $\{1, 3\}$, $\{1, 4\}$, $\{1, 7\}$, $\{2, 3\}$, $\{2, 4\}$, $\{2, 5\}$, $\{2, 7\}$, $\{3, 4\}$, $\{3, 6\}$, $\{3, 7\}$, $\{4, 5\}$, $\{4, 6\}$, $\{5, 6\}$, $\{5, 7\}$, $\{6, 7\}$.
   Make an optimum schedule for the exam of these subjects.

   Solution: The graph associated with the subjects is as in figure (i) below:



(i) *(ii)*

The coloring of the graph with minimum number of colors is shown in figure (ii).

Since 4 different colors are used, the exam can be scheduled using 4 time slots as follows:

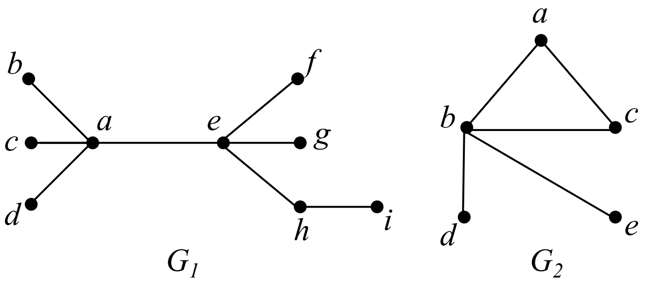
Time Slots	Courses
I (red)	1, 6
II (blue)	2
III (green)	3, 5
IV (brown)	4, 7

2. Schedule the exam for ten subjects $1, 2, 3, 4, 5, 6, 7, 8, 9, 10$ assuming that the pairs of subjects $\{1, 2\}, \{1, 5\}, \{1, 8\}, \{2, 4\}, \{2, 9\}, \{2, 7\}, \{3, 6\}, \{3, 7\}, \{3, 10\}, \{4, 8\}, \{4, 3\}, \{4, 10\}, \{5, 6\}, \{5, 7\}$ have common students.

3. Schedule the exams for Math115, Math116, Math185, Math195, CS101, CS102, CS273, and CS473, using the fewest number of different time slots, if there are no students taking the following pairs of subjects but there are students in every other combination of courses:
{Math115, CS473}, {Math115, Math116}, {Math115, Math185}, {Math116, CS 473}, {Math185, Math195}, {Math195, CS101}, {Math195, CS102}.

6.2 Trees and Spanning Trees

Tree: A tree is a connected undirected graph which has no simple circuits. For example, the following graph G_1 is a tree and G_2 is NOT a tree.

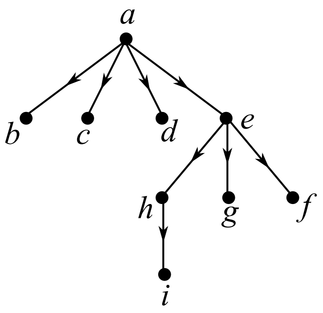


Forest: A collection of disjoint trees is called a forest.

Theorem 1: An undirected graph is a tree if and only if there is a unique simple path between any two of its vertices.

Proof: Suppose that an undirected graph T is a tree. We need to show that there is a unique simple path between any two of its vertices, say x and y. Since T is connected, there must exist a simple path between x and y. We claim that this path must be unique, for if there were a second such path, then the path formed by combining the first path from x to y followed by the second path from y to x would contain a circuit and hence, a simple circuit. This contradicts the fact that T is a tree. So there is a unique simple path between any two vertices x and y of a tree.
Conversely, suppose that an undirected graph T has a unique simple path between any two of its vertices. We need to show that T is a tree. Since there is a path between any two vertices of T, so T is connected. Also, T cannot have simple circuits because if it had a simple circuit containing the vertices, say x and y, then there would be two simple paths between x and y formed by dividing the simple circuit at x and y. This contradicts the fact that T has a unique simple path between any two of its vertices. Hence T is a tree. □

Rooted Tree: A rooted tree is a tree in which one vertex has been designated as the root and every edge is directed away from the root. For example, in the graph G_1 above, the rooted tree obtained by designating the vertex a as the root is



Note: Rooted trees are usually drawn with the root at the top of the graph. So the arrows indicating the directions of the edges in a rooted tree can be omitted, because the edges are always directed "away" or "downward" from the root.

Some definitions:
Let T be a rooted tree as in the above figure.

(a) For a vertex T other than the root, the **parent** of v is the unique vertex u such that there is a directed edge from u to v, e.g., parent of h is e.

(b) A vertex v is called a **child** of vertex u if there is a directed edge from u to v, e.g., h is a child of e.

(c) Vertices with the same parent are called **siblings**, e.g., h, g and f are siblings.

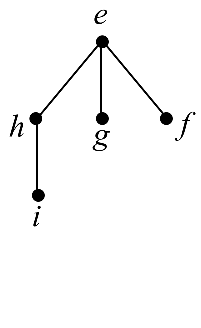
(d) The **ancestors** of a vertex other than the root are the vertices in the path from root to this vertex, excluding the vertex itself and including the root, e.g., ancestors of i are h, e and a.

(e) The **descendants** of a vertex v are those vertices that have v as an ancestor, e.g., descendants of e are h, g, f, i.

(f) A vertex of a tree is called a **leaf** if it has no children, e.g., b, c, d, i, g, f are leafs.

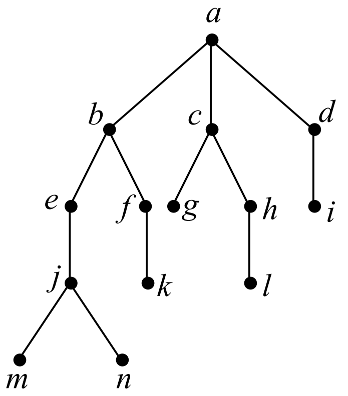
(g) A vertex of a tree is called an **internal vertex** if it has children. The root is an internal vertex unless it is the only vertex in the graph, in which case it is a leaf, e.g., a, e, h are internal vertices.

(h) If v is a vertex in a tree, the **subtree** with v as its root is the subgraph of the tree consisting of v and its descendants and all edges incident to these descendants, e.g., the subtree with e as root is



(i) The **level of a vertex** v in a rooted tree is the length of the unique path from the root to this vertex.

(j) The **height** of a rooted tree is the maximum of the levels of its vertices. For example, the height of the following rooted tree is 4.

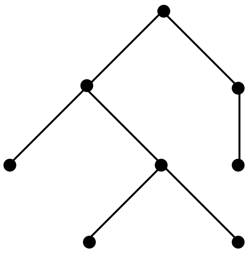


level-0 vertices: a
level-1 vertices: b, c, d
level-2 vertices: e, f, g, h, i
level-3 vertices: j, k, l
level-4 vertices: m, n

m**-ary tree:** A rooted tree is called an m-ary tree if every internal vertex has no more than m children.

Full m**-ary tree:** A rooted tree is called a full m-ary tree if every internal vertex has exactly m children.

Binary tree: An m-ary tree with $m = 2$ is called a binary tree.



Ordered rooted tree: A rooted tree where the children of each internal vertex are ordered as first child, second child, third child and so on is called an ordered rooted tree.
The first child of an internal vertex in an ordered binary tree is called the left child and the second child is called the right child.

Theorem 2: A tree with n vertices has $n - 1$ edges.

Proof: We use mathematical induction to prove this theorem which proceeds in two steps as below:

Basis step: We need to show that the theorem is true when $n = 1$ i.e., a tree with 1 vertex has $1 - 1 = 0$ edges which is clearly true.
Inductive step: We need to show that if the theorem is true for $n = k$, then the theorem must also be true for $n = k + 1$ for some positive integer k. So let T be a tree with $k + 1$ vertices and let v be a leaf of T whose parent is w. If the vertex v along with the edge connecting w to v is removed from T, then a tree T' is produced with k vertices. By induction hypothesis, T' has $k - 1$ edges. Since T' is formed from T by removing exactly one edge, so T must have $(k-1)+1 = k$ edges. Therefore the induction step is true as well and so the theorem is proved.

□

6.2.1 Tree Traversal

The process of visiting each vertex of an ordered rooted tree in some order is called tree traversal.

There are three recursive methods of tree traversal.

1. **Preorder traversal:** In preorder traversal of an ordered rooted tree T, a vertex (starting from the root) is visited and then the subtrees of this vertex is traversed in preorder from left to right.
   More precisely, if T is an ordered rooted tree with root r and subtrees $T_1, T_2, \cdots, T_n$ at r, then preorder traversal of T proceeds recursively as follows:

   I. Visit the root r.

   II. Traverse the first subtree T_1 in preorder.

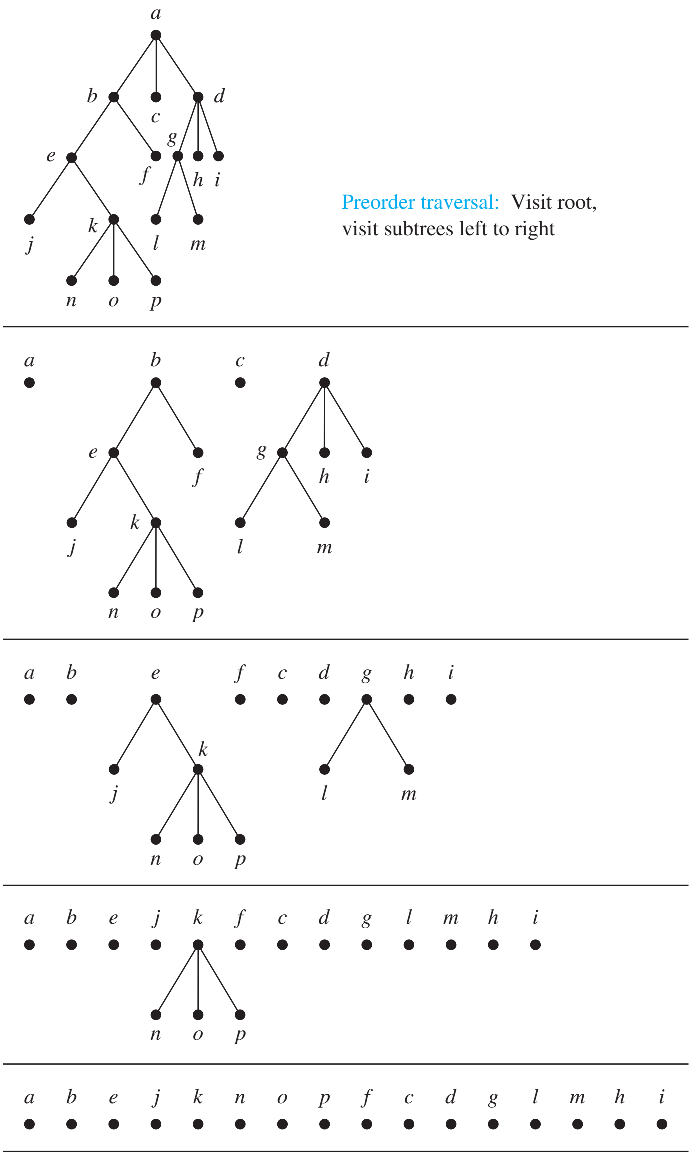
   III. Traverse the second subtree T_2 in preorder.

   IV. Proceeding similarly, traverse the n-th subtree T_n in preorder.

   Algorithm for preorder traversal:

   ```
   PROCEDURE preorder ($T$: ordered rooted tree)
       $r$ = root of $T$
       list $r$
       FOR each child $c$ of $r$ from left to right {
           $T(c)$ = subtree with $c$ as its root
           preorder ($T(c)$)
       }
   ```

   Example:


a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
p
a b c d e f g h i j k l m n o p
a b e f c d g h i j k l m n o p
a b e j k f c d g l m h i n o p
a b e j k n o p f c d g l m h i

Preorder traversal: Visit root, visit subtrees left to right

2. **Postorder Traversal:** In postorder traversal of an ordered rooted tree T, all the subtrees starting from left to right are traversed in postorder and then the root of the tree is visited. More precisely, if T is an ordered rooted tree with root r and subtrees $T_1, T_2, \cdots, T_n$ at r, then postorder traversal of T proceeds recursively as follows:

   I. Traverse the first subtree T_1 in postorder.

   II. Traverse the second subtree T_2 in postorder.

   III. Proceeding similarly, traverse the n-th subtree T_n in postorder.

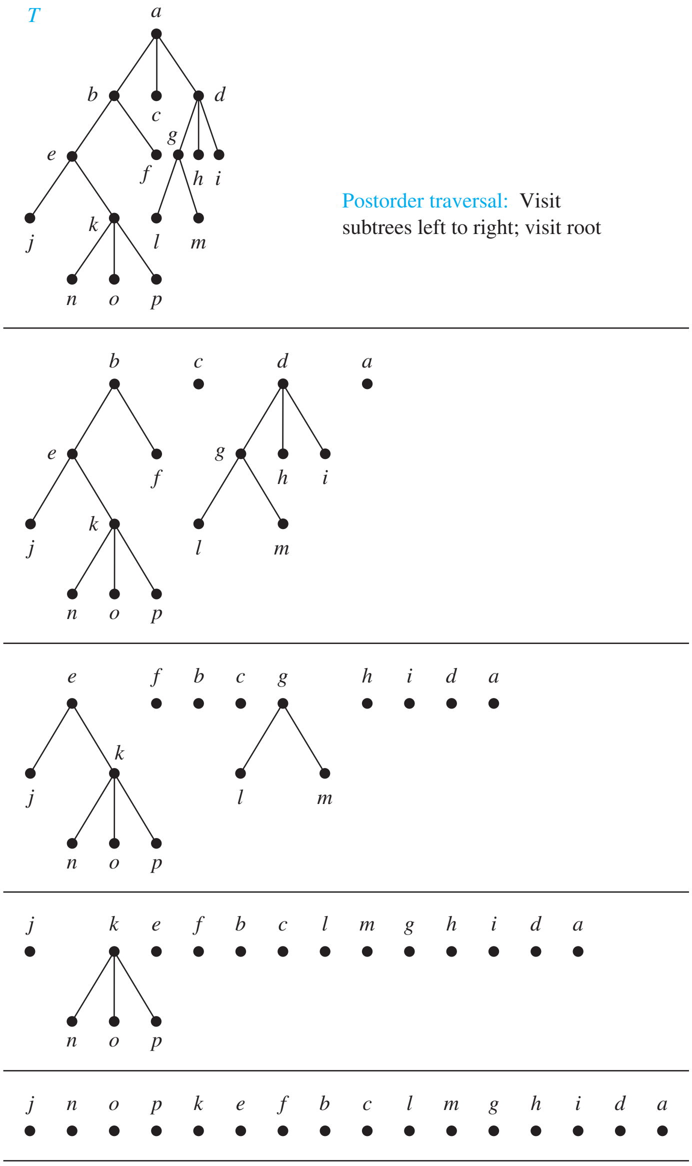
   IV. Visit the root r.

   Algorithm for postorder traversal:

   PROCEDURE postorder (T: ordered rooted tree)

```
$r$ = root of $T$
FOR each child $c$ of $r$ from left to right {
    $T(c)$ = subtree with $c$ as its root
    postorder $(T(c))$
} list $r$
```

Example:



Postorder traversal: Visit subtrees left to right; visit root

3. **Inorder Traversal:** In inorder traversal of an ordered rooted tree T, the first subtree starting at the left is traversed in inorder, then the root of the tree is visited and then the subtrees from second to last are traversed in inorder.
More precisely, if T is an ordered rooted tree with root r and subtrees $T_1, T_2, \cdots, T_n$ at r, then inorder traversal of T proceeds recursively as follows:

I. Traverse the first subtree T_1 in inorder.

IV. Visit the root r.

II. Traverse the second subtree T_2 in inorder.

III. Proceeding similarly, traverse the n-th subtree T_n in inorder.

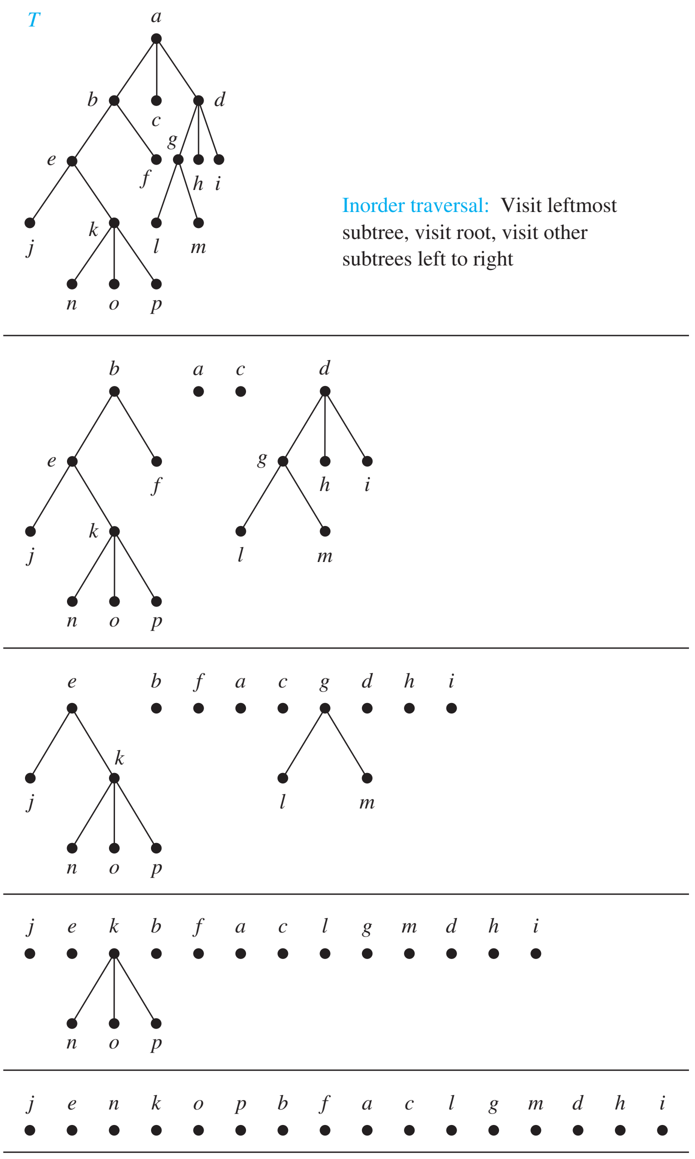
Note: When traversing a binary tree, if the first subtree (left subtree) is absent then we first visit the root and then traverse the second (right) subtree in inorder.

Algorithm for inorder traversal:

PROCEDURE inorder (T: ordered rooted tree)

```
    $r$ = root of $T$
    IF $r$ is a leaf THEN list $r$
    ELSE
    {
        $l$ = first child of $r$ from left to right
        $T(l)$ = subtree with $l$ as its root
        inorder($T(l)$)
        list $r$ FOR each child $c$ of $r$ except for $l$ from left to right
            $T(c)$ = subtree with $c$ as its root
            inorder($T(c)$)
    }
```

Example:



Inorder traversal: Visit leftmost subtree, visit root, visit other subtrees left to right

6.2.2 Binary Search Trees (BST)

Simplest way to store data is in linear data structures such as arrays but searching for data stored in linear data structure can be time consuming. For example, given a sequence of numbers as $17, 23, 4, 7, 9, 19, 45, 6, 2, 37, 99$, we can store them in an array as:

17	23	4	7	9	19	45	6	2	37	99

If one wants to search for the number, say 57, then we have to make 11 comparisons before knowing that this number is not in the sequence.
Another way to store data in, is to use hierarchical data structures such as binary search tree as below. The data stored in binary search trees is much simpler and less time-consuming to search.

Algorithm for locating and inserting items in a BST:

PROCEDURE Insertion (T: binary search tree; x: item)

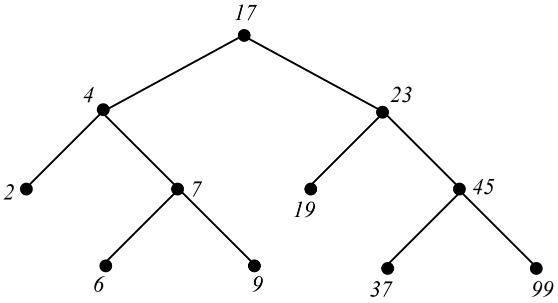
```
    $v$:= root of $T$
    {a vertex not present in $T$ has the value $null$}
    WHILE $v \neq null$ and $label(v) \neq x$
        BEGIN
            IF $x < label(v)$ THEN
                IF left child of $v \neq null$ THEN $v :=$ left child of $v$
                ELSE add new vertex as a left child of $v$ and set $v :=$
                $null$
            ELSE
                IF right child of $v \neq null$ THEN $v :=$ right child of $v$
                ELSE add new vertex as a right child of $v$ to $T$ and set
                $v := null$
        END
    IF root of $T = null$ THEN add a vertex $v$ to the tree and label it with $x$
    ELSE IF $v$ is $null$ or $label(v) \neq x$ THEN label new vertex with $x$ and let $v$ be this
    new vertex
{$v$ =location of $x$}
```

Problems:

Construct binary search trees for the following data:

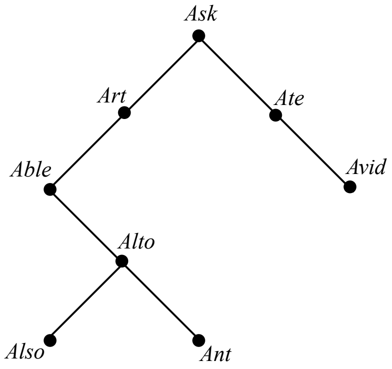
(i) 17, 23, 4, 7, 9, 19, 45, 6, 2, 37, 99

Solution: The final tree is



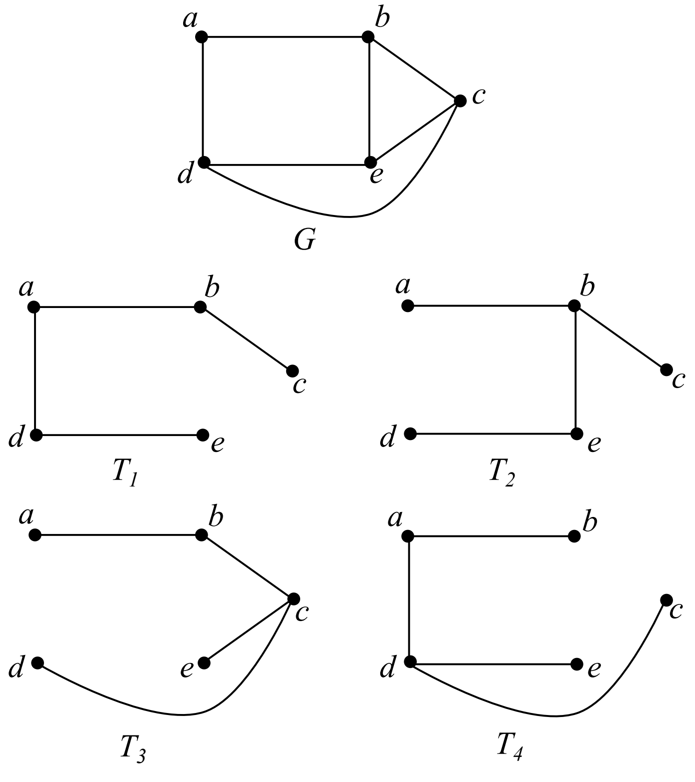
(ii) Ask, Art, Ate, Able, Alto, Also, Avid, Ant

Solution: The final tree is



6.2.3 Spanning Tree

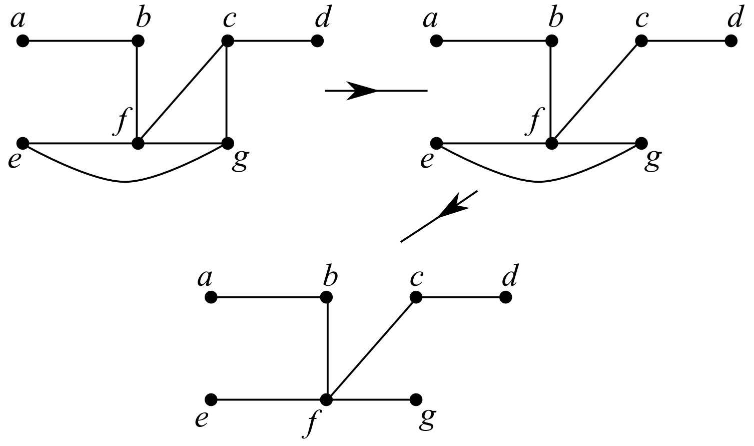
Spanning tree: A spanning tree of a simple graph G is a subgraph of G which is also a tree containing all the vertices of G.
For example, given a simple graph G below, some of its spanning trees are T_1, T_2, T_3, T_4 etc.



Theorem: A simple graph is connected if and only if it has a spanning tree.

Proof: Suppose that a simple graph G has a spanning tree T. Then every vertex of G is also in T and there is a path in T between any two of its vertices. Since T is a subgraph of G, so there is a path in G between any two vertices of G. Hence G is connected.
Conversely, suppose that a simple graph G is connected. We need to show that G has a spanning tree. If G is itself a tree, then our theorem is proved. If not, then it must contain a simple circuit. If an edge is removed from one of these simple circuits then the resulting subgraph has all the

vertices of G but one less edge. This subgraph is still connected because when two vertices are connected by a path containing the removed edge, they are connected by a path not containing this edge. Such path can be constructed by inserting into the original path, at the point where removed edge once was, the simple circuit with this edge removed. If this subgraph is not a tree, it must have a simple circuit and so as before, remove an edge that is in a simple circuit. Since there are only a finite number of edges in G, so this process can be repeated until no simple circuits remain. When no simple circuits remain, the process terminates. So we have produced a connected subgraph of G that has no simple circuits and contains all the vertices of G i.e., a spanning tree of G is formed. □

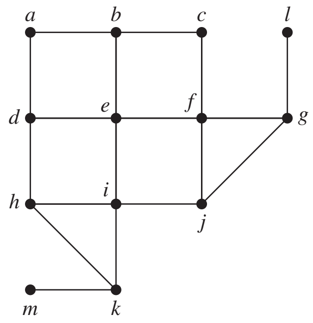


Methods of forming a spanning tree: Given a connected simple graph G, we can make a rooted tree which is a spanning tree of G by following systematic procedure as follows.

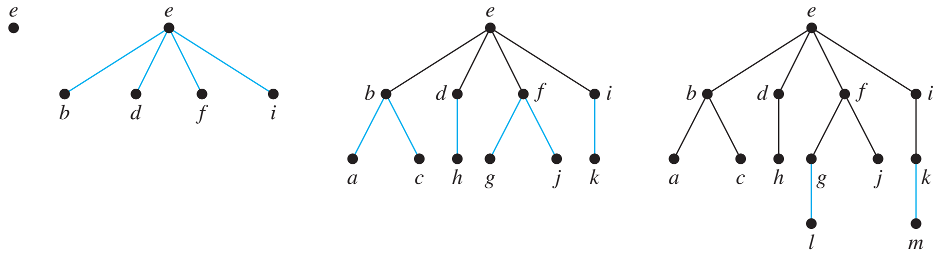
1. **Breadth-First Search (BFS):** The breadth-first search method of forming a spanning tree from a graph G attempts to make a spanning tree which is as fat/wide as possible.
   For this, we select a vertex of G arbitrarily as root of the tree. Then add all edges incident to this vertex. The new vertices added at this stage becomes the vertices at level 1 in the spanning tree. Arbitrarily order these vertices. Next, for each vertex at level 1, visited in order, add each edge incident to this vertex to the tree as long as it does not produce a cycle. Arbitrarily order the children of each vertex at level 1. This produces the vertices at level 2 in the tree. Follow the same procedure until all the vertices in the tree have been added.

   Examples:

   a. Form a spanning tree of the following graph using breadth-first search with vertex e as root.



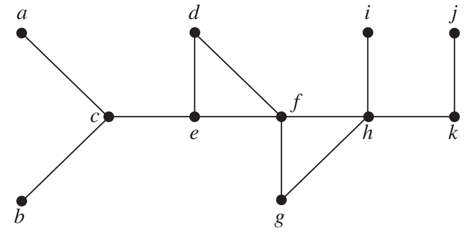
Solution:



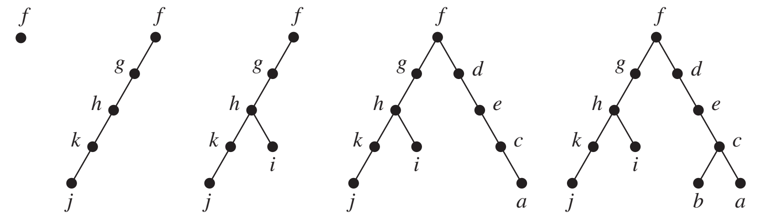
2. **Depth-First Search (DFS):** The depth-first search method of forming a spanning tree of a graph G attempts to make a spanning tree that is as tall/long as possible.
   To start, we choose an arbitrary vertex of the graph G as the root. Then form a path starting at this vertex by successively adding vertices and edges where each new edge is incident with the last vertex in the path and a vertex not already in the path. Continue adding vertices and edges to this path as long as possible. If the path goes through all the vertices of the graph, the tree consisting of this path is a spanning tree. However, if the path does not go through all vertices, move back to the next to last vertex in the path and if possible, form a new path starting at this vertex passing through vertices that were not already visited. If this cannot be done, move back another vertex in the path i.e., two vertices back in the path and try again. Repeat this process until no more edges can be added.

   Examples:

   a. Form a spanning tree of the following graph using depth-first search with vertex f as root.

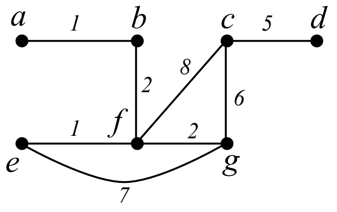


Solution:

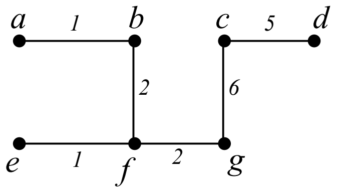


(a) (b) (c) (d) (e)

Minimum Spanning Tree (MST): A minimum spanning tree in a connected weighted graph is a spanning tree that has the smallest possible sum of weights of its edges.
For example, for the weighted connected graph shown below, its minimum spanning tree is as given below:



G



minimum spanning tree of G

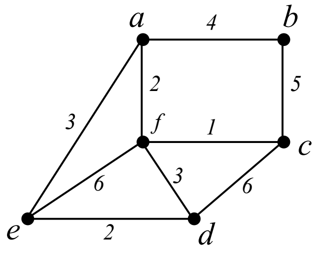
Kruskal's Algorithm for finding MST:

PROCEDURE Kruskal (G: weighted connected undirected graph with n vertices)

```
    $T$:= empty graph
    FOR $i := 1$ TO $n-1$
        BEGIN
            $e :=$ any edge in $G$ with smallest weight that does not form a
            simple circuit when added to $T$.
            $T := T$ with $e$ added
        END
```

$\{T$ is the minimum spanning tree of $G\}$

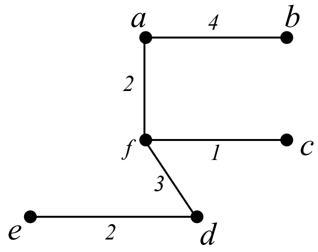
Problems:
Use Kruskal's algorithm to find the minimum spanning tree of the following graphs:
(a)



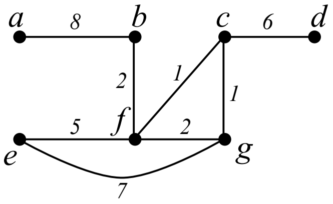
Solution: The following edges are chosen to form MST using Kruskal's algorithm.

Choice	Edge	Weight
1	$\{f, c\}$	1
2	$\{e, d\}$	2
3	$\{a, f\}$	2
4	$\{f, d\}$	3
5	$\{a, b\}$	4
		12

Therefore the MST formed has minimum weight 12.



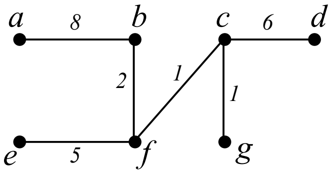
(b)



Solution: The following edges are chosen to form MST using Kruskal's algorithm.

Choice	Edge	Weight
1	$\{f, c\}$	1
2	$\{c, g\}$	1
3	$\{b, f\}$	2
4	$\{e, f\}$	5
5	$\{c, d\}$	6
6	$\{a, b\}$	8
		23

Therefore the MST formed has minimum weight 23.



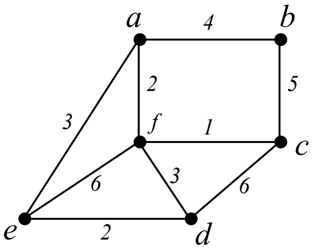
Prim's Algorithm for finding MST:

PROCEDURE Prim (G: weighted connected undirected graph with n vertices)

```
    $T$:= a minimum weight edge
    FOR $i := 1$ TO $n - 2$
        BEGIN
            $e :=$ an edge of minimum weight incident to a vertex in $T$ and
            not forming a simple circuit in $T$ if added to $T$.
            $T := T$ with $e$ added
        END
```

$\{T$ is the minimum spanning tree of $G\}$

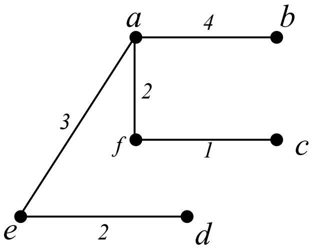
Problems:
Use Prim's algorithm to find the minimum spanning tree of the following graphs:
(a)



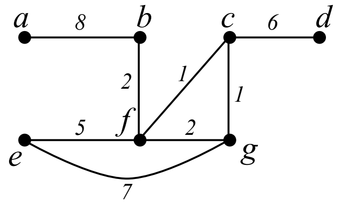
Solution: The following edges are chosen to form MST using Prim's algorithm.

Choice	Edge	Weight	Vertices
1	$\{f, c\}$	1	f, c
2	$\{f, a\}$	2	f, c, a
3	$\{a, e\}$	3	f, c, a, e
4	$\{e, d\}$	2	f, c, a, e, d
5	$\{a, b\}$	4	f, c, a, e, d, b
		$\overline{12}$	

Therefore the MST formed has minimum weight 12.



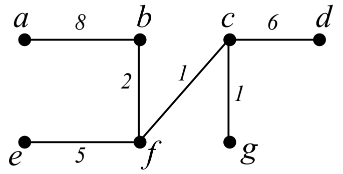
(b)



Solution: The following edges are chosen to form MST using Prim's algorithm.

Choice	Edge	Weight	Vertices
1	$\{f, c\}$	1	f, c
2	$\{c, g\}$	1	f, c, g
3	$\{b, f\}$	2	f, c, g, b
4	$\{e, f\}$	5	f, c, g, b, e
5	$\{c, d\}$	6	f, c, g, b, e, d
6	$\{a, b\}$	8	f, c, g, b, e, d, a
		23	

Therefore the MST formed has minimum weight 23.



6.3 Network Flows

Transport Network: Let $G = (V, E)$ be a weakly connected directed graph that does not contain any loops such that

(i) there are exactly two distinguished vertices S and D called the source and sink of G respectively and

(ii) there is a nonnegative real-valued function k defined on E called the capacity function of G.

Then (G, k) is called a transport network. Also, for an edge $e \in E$, the number $k(e)$ is called the capacity of e.

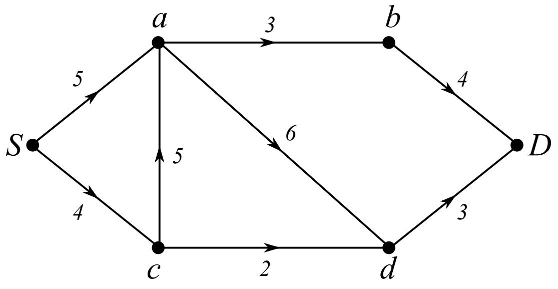


Fig. 1

Notations: Let $G = (V, E)$ be a directed graph and x be a vertex. Then $A(x) = \{y \in V \mid (x, y) \in E\}$ and $B(x) = \{y \in V \mid (y, x) \in E\}$. For example, in the above graph, $A(a) = \{b, d\}$ and $B(a) = \{S, c\}$.

Flow: Let (G, k) be a transport network. Then a flow in G is a nonnegative real-valued function F defined on E such that

(i) $0 \le F(e) \le k(e)$ for each edge $e \in E$. (Capacity constraint)

(ii) if x is any vertex of G other that the source or the sink, then the sum of all values of $F(x, y)$ where $y \in A(x)$ is equal to the sum of all values $F(z, x)$ where $z \in B(x)$ i.e.,

$$\sum_{y \in A(x)} F(x, y) = \sum_{z \in B(x)} F(z, x)$$

for all $x \in V - \{S, D\}$. (Conservation equation)

(iii) $F(e) = 0$ for any edge e incident to the source S or incident from the sink D.

For example, the following diagram shows a flow in the transport network in figure 1 above.

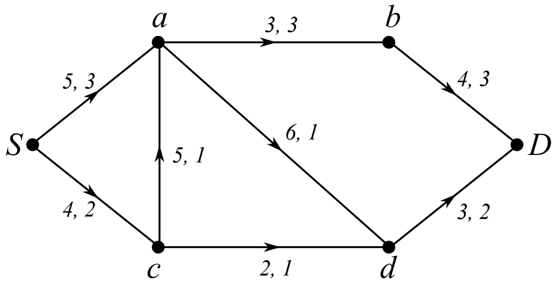


Fig. 2

Saturated and Unsaturated Edge: The edges for which the flow and capacity are equal are called saturated. Otherwise they are called unsaturated edges.
If e is an unsaturated edge, then the slack of e denoted by $s(e)$ is defined as $s(e) = k(e) - F(e)$. For example, the edge ab is saturated and the edge cd is unsaturated in the network flow in figure 2 above. Here slack of cd is $2 - 1 = 1$.

Notation: If $X \subset V$ and $Y \subset V$, then (X, Y) denotes the set of all edges which go from a vertex in X to a vertex in Y. For example, if $X = \{a, b\}$ and $Y = \{c, d, D\}$ in previous graph,

then $(X, Y) = \{(a, d), (b, D)\}$.
If g is any real-valued function on E, then we define $g(X, Y) = \sum_{e \in (X,Y)} g(e)$. For example, if (X, Y) is as above, then $k(X, Y) = \sum_{e \in (X,Y)} k(e) = k(a, d) + k(b, D) = 6 + 4 = 10$ and $F(X, Y) = \sum_{e \in (X,Y)} F(e) = F(a, d) + F(b, D) = 1 + 3 = 4$.

Theorem: Let S and D be the source and sink respectively of a transport network (G, k). Let F be a flow in G. Then, the flow out of $S = \sum_{x \in A(S)} F(S, x)$ is equal to the flow into $D = \sum_{x \in B(D)} F(x, D)$.

Value of the flow: Let F be a flow defined on the transport network (G, k). Let $A(S) = \{a_1, a_2, \cdots, a_m\}$ and $B(D) = \{b_1, b_2, \cdots, b_t\}$. Then the value of the flow F, denoted by $|F|$, is defined as

$$|F| = F(S, a_1) + F(S, a_2) + \cdots + F(S, a_m) = F(b_1, D) + F(b_2, D) + \cdots + F(b_t, D).$$

For example, in the previous example, $|F| = F(S, a) + F(S, c) = F(b, D) + F(d, D) = 5$.

Maximal Flow: A flow F in a network (G, k) is called a maximal flow if $|F'| \le |F|$ for any flow F' in (G, k).

S-D Cut: Let (G, k) be a transport network with source S and sink D. Let $X \subset V$ and $\overline{X} = V - X$ such that $S \in X$ and $D \in \overline{X}$. Then the set $(X, \overline{X})$ of all the edges from a vertex in X to a vertex in $\overline{X}$ is called an $S - D$ cut.
For example, in our previous network, $(X, \overline{X})$ is an $S - D$ cut where $X = \{S, a, d\}$ (so that $\overline{X} = \{b, c, D\}$) and $(X, \overline{X}) = \{(S, c), (a, b), (d, D)\}$.

Capacity of a cut: The capacity of an $S - D$ cut $(X, \overline{X})$, denoted by $k(X, \overline{X})$ is defined as the sum of the capacities of all the edges belonging to that cut.
For example, the capacity of the previous cut is $k(X, \overline{X}) = 4 + 3 + 3 = 10$.

Minimal Cut: An $S - D$ cut $(X, \overline{X})$ is called a minimal cut in a transport network (G, k) if there is no $S - D$ cut $(Y, \overline{Y})$ such that $k(Y, \overline{Y}) < k(X, \overline{X})$.
For example, the minimal cut in the previous transport network is $(X, \overline{X})$ where $X = \{S, a, c, d\}$ and $\overline{X} = \{b, D\}$. So $(X, \overline{X}) = \{(a, b), (d, D)\}$ with capacity $k(X, \overline{X}) = 3 + 3 = 6$.

Max Flow-Min Cut Theorem: In any transport network, the value of any maximal flow is equal to the capacity of a minimal cut.

This theorem asserts the following:

(1) the existence of a minimal cut $(X, \overline{X})$,

(2) the existence of a maximal flow F, and

(3) the equality of $|F|$ and $k(X, \overline{X})$ for any maximal flow F and any minimal cut $(X, \overline{X})$.

To find the maximal flow in a transport network, there are two basic strategies as follows:

(i) If an edge is not being used to capacity, we could try to send some of the commodity through it.

(ii) If an edge is working against us by sending some of the commodity back towards the source, we could try to reduce the flow along this edge and redirect in a more practical direction.

Forward and Backward Edges: Let (G, k) be a transport network with source S, sink D and a flow F. A non-directed path P from S to D is a sequence of edges $e_1, e_2, \cdots, e_n$ and a sequence of vertices $S = v_0, v_1, \cdots, v_n = D$ where edge e_i is directed from either

(a) v_{i-1} to v_i

(b) v_i to v_{i-1}

Then the edge e_i is called a forward edge of P if (a) is true and backward edge if (b) is true. For example, in our previous transport network, $SacdD$ is a non-directed path from S to D. In this path, $(S, a), (c, d)$ and (d, D) are forward edges whereas (c, a) is a backward edge.

F-augmenting Path: Let P be a path from S to D in a transport network (G, k) with flow F. Then P is called an F-augmenting path if

(i) each forward edge of P is unsaturated i.e., $F(e) < k(e)$ and hence $s(e) > 0$ for any forward edge e of P

(ii) each backward edge of P, if it exists, has positive flow i.e., $F(e) > 0$ for any backward edge e of P

For example, in the previous transport network, $SacdD$ is an F-augmenting path. Its forward edges $(S, a), (c, d)$ and (d, D) are all unsaturated and its backward edge (c, a) has a positive flow. Another F-augmenting path in the same transport network is $SadD$ which consists entirely of three forward edges, all of which are unsaturated.

Notation: For an F-augmenting path P, let $\epsilon(P) = \min \epsilon_i(P)$ where

$$\epsilon_i(P)=\begin{cases} s(e_i) = k(e_i) - F(e_i) & \text{if } e_i \text{ is a forward edge of } P \\ F(e_i) & \text{if } e_i \text{ is a backward edge of } P \end{cases}$$

Clearly, $\epsilon(P) > 0$ for an F-augmenting path P.

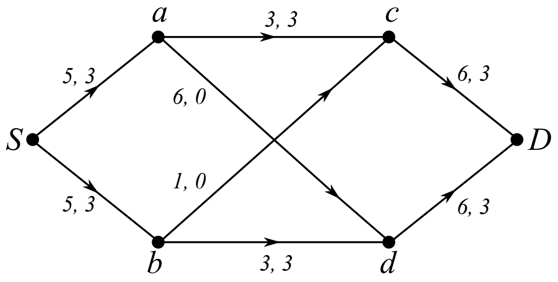
Algorithm for constructing maximal flow:

Input: A transport network (G, k) with a given flow F
Output: A maximal flow
Procedure:
Repeat steps 1 and 2 below until there are no F-augmenting paths left.

1. Choose any F-augmenting path P.
2. Form a new flow of higher value equal to $|F| + \epsilon(P)$ using F and the F-augmenting path P as follows:
   (a) Add $\epsilon(P)$ to the value of the flow along all the forward edges on P.
   (b) Subtract $\epsilon(P)$ from the value of the flow along all the backward edges on P.

Note: To construct a minimal cut $(X, \overline{X})$ corresponding to a maximal flow, we note that all the edges that are in the minimal cut must be saturated (in the final flow diagram). So if any vertex in X forms an unsaturated edge ending in some vertex v, then v must also belong to X and if any unsaturated edge starts from a vertex v and ends in a vertex in $\overline{X}$ then v must also belong to $\overline{X}$.

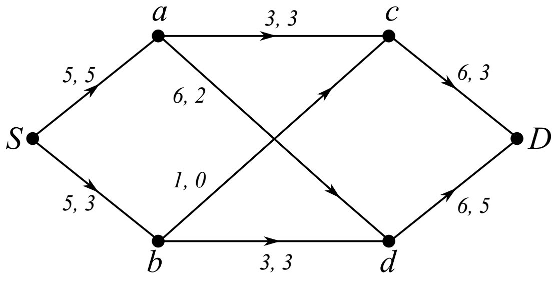
Problems:
Find a maximum flow for the network in the figures below:
(i)



Solution: The path P given by $SadD$ is an F-augmenting path from S to D. For this path,

$$\epsilon(P) = \min\{5 - 3, 6 - 0, 6 - 3\} = \min\{2, 6, 3\} = 2.$$

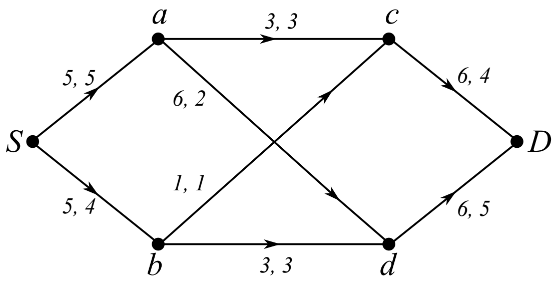
So the flow can be increased along the path $SadD$ by 2 as follows:



The resulting network has the path P given by $SbcD$ as an F-augmenting path from S to D for which

$$\epsilon(P) = \min\{5 - 3, 1 - 0, 6 - 3\} = \min\{2, 1, 3\} = 1.$$

So the flow along the path $SbcD$ can be increased by 1 as follows:

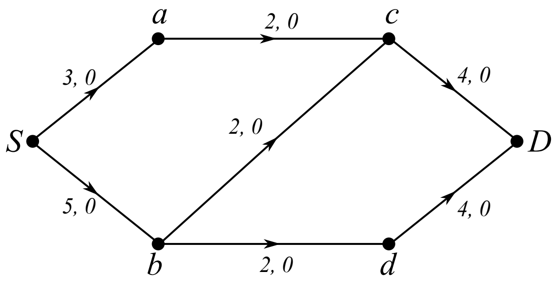


Now there are no more F-augmenting paths in this network and so we have obtained a maximal flow whose value is

$$|F| = 5 + 4 = 4 + 5 = 9.$$

Note: The minimal cut in this network is $(X, \overline{X})$ where $X = \{S, b\}$ and $\overline{X} = \{a, c, d, D\}$.

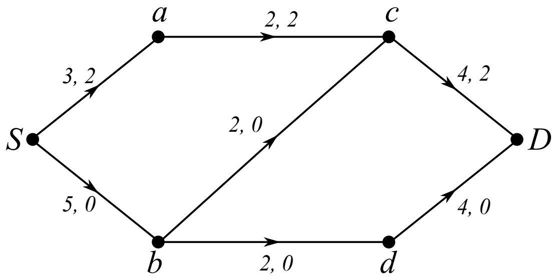
(ii)



Solution: The flow along the F-augmenting path P given by $SacD$ can be increased by

$$\epsilon(P) = \min\{3 - 0, 2 - 0, 4 - 0\} = \min\{3, 2, 4\} = 2$$

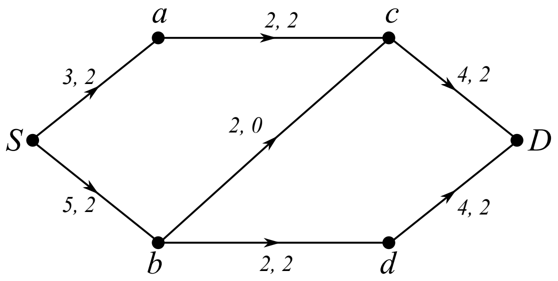
as follows:



The flow along the F-augmenting path P given by $SbdD$ can be increased by

$$\epsilon(P) = \min\{5 - 0, 2 - 0, 4 - 0\} = \min\{5, 2, 4\} = 2$$

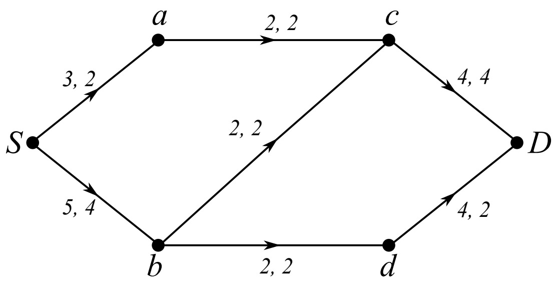
as follows:



The flow along the F-augmenting path P given by $SbcD$ can be increased by

$$\epsilon(P) = \min\{5-2, 2-0, 4-2\} = \min\{3, 2, 2\} = 2$$

as follows:

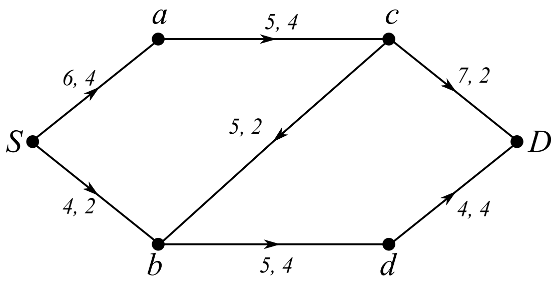


Now there are no more F-augmenting paths in this network and so we have reached a maximal flow whose value is

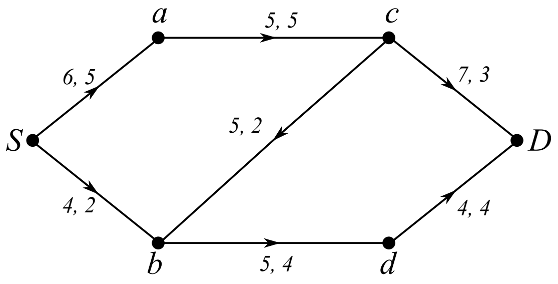
$$|F| = 2+4 = 4+2 = 6.$$

Note: The minimal cut in this network is $(X, \overline{X})$ where $X = \{S, a, b\}$ and $\overline{X} = \{c, d, D\}$ OR $X = \{S, a, b, c\}$ and $\overline{X} = \{d, D\}$.

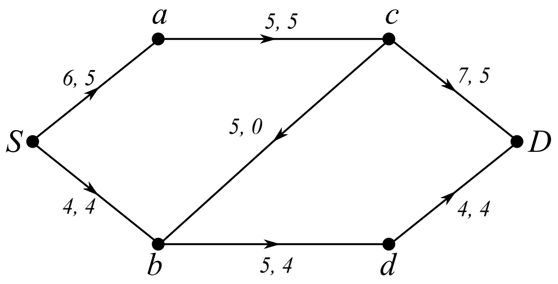
(iii)



Solution: The flow along the F-augmenting path P given by $SacD$ can be increased by $\epsilon(P) = \min\{6-4, 5-4, 7-2\} = \min\{2, 1, 5\} = 1$ as follows:



The resulting network has an F-augmenting path P given by $SbcD$ where (S, b) and (c, D) are forward edges and (c, b) is a backward edge. So the flow along the path $SbcD$ can be increased by $\epsilon(P) = \min\{4-2, 2, 7-3\} = \min\{2, 2, 4\} = 2$ as follows:

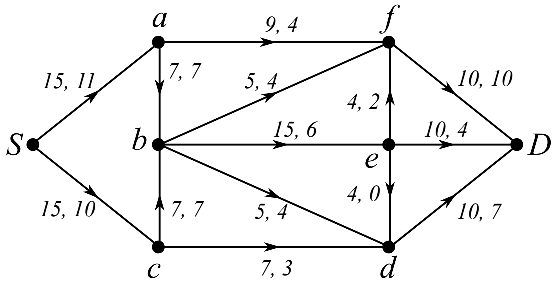


Now there are no more F-augmenting paths in this network and so we have reached a maximal flow whose value is

$$|F| = 5 + 4 = 9.$$

Note: The minimal cut in this network is $(X, \overline{X})$ where $X = \{S, a\}$ and $\overline{X} = \{b, c, d, D\}$.

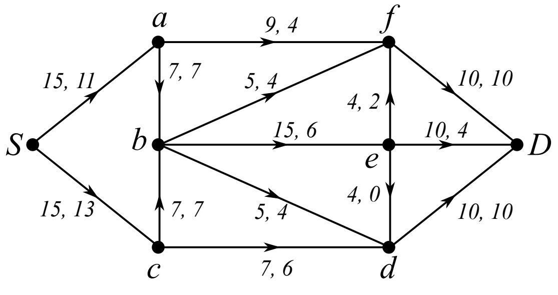
(iv)



Solution: The flow along the F-augmenting path $ScdD$ can be increased by

$$\epsilon(P) = \min\{15 - 10, 7 - 3, 10 - 7\} = \min\{5, 4, 3\} = 3$$

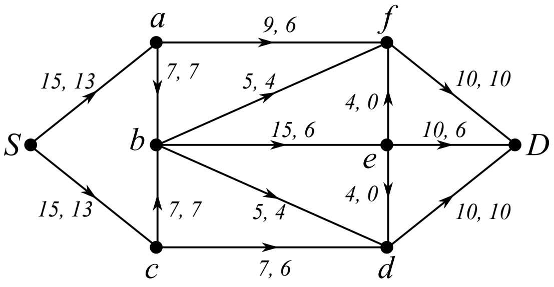
as follows:



The flow along the F-augmenting path $SafeD$ can be increased by

$$\epsilon(P) = \min\{15 - 11, 9 - 4, 2, 10 - 4\} = \min\{4, 5, 2, 6\} = 2$$

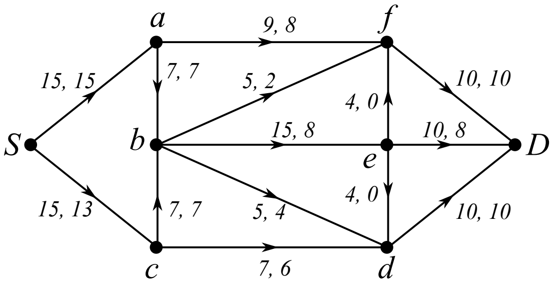
as follows:



The flow along the F-augmenting path $SafbeD$ can be increased by

$$\epsilon(P) = \min\{15 - 13, 9 - 6, 4, 15 - 6, 10 - 6\} = \min\{2, 3, 4, 9, 4\} = 2$$

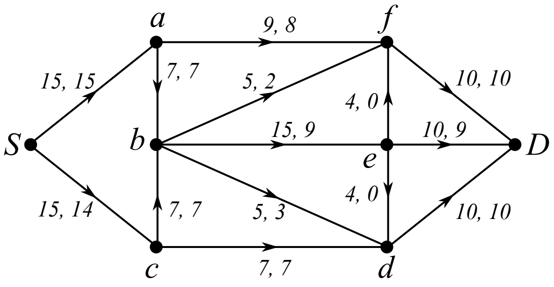
as follows:



The flow along the F-augmenting path $ScdbeD$ can be increased by

$$\epsilon(P) = \min\{15 - 13, 7 - 6, 4, 15 - 8, 10 - 8\} = \min\{2, 1, 4, 7, 2\} = 1$$

as follows:



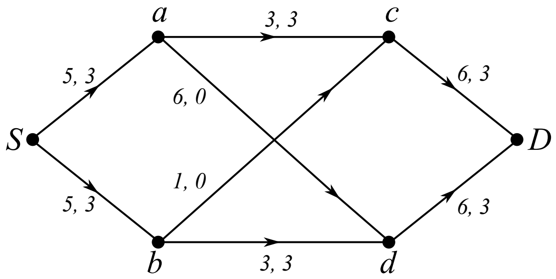
There are no more F-augmenting paths. So we have reached a maximal flow whose value is

$$|F| = 15 + 14 = 10 + 9 + 10 = 29.$$

Note: The minimal cut in this network is $(X, \overline{X})$ where $X = \{S, c\}$ and $\overline{X} = \{a, b, d, e, f, D\}$.

Find a minimal cut for the network in the figures below:

(i)



Solution: The possible cuts of the given network are shown in the table below:

X	$\overline{X}$	$(X,\overline{X})$	Capacity
$\{S\}$	$\{a,b,c,d,D\}$	$\{(S,a),(S,b)\}$	$5+5=10$
$\{S,a\}$	$\{b,c,d,D\}$	$\{(a,c),(a,d),(S,b)\}$	$3+6+5=14$
$\{S,b\}$	$\{a,c,d,D\}$	$\{(S,a),(b,c),(b,d)\}$	$5+1+3=9$
$\{S,c\}$	$\{a,b,d,D\}$	$\{(S,a),(S,b),(c,D)\}$	$5+5+6=16$
$\{S,d\}$	$\{a,b,c,D\}$	$\{(S,a),(S,b),(d,D)\}$	$5+5+6=16$
$\{S,a,b\}$	$\{c,d,D\}$	$\{(a,c),(a,d),(b,c),(b,d)\}$	$3+6+1+3=13$
$\{S,a,c\}$	$\{b,d,D\}$	$\{(S,b),(a,d),(c,D)\}$	$5+6+6=17$
$\{S,a,d\}$	$\{b,c,D\}$	$\{(S,b),(a,c),(d,D)\}$	$5+3+6=14$
$\{S,b,c\}$	$\{a,d,D\}$	$\{(S,a),(b,d),(c,D)\}$	$5+3+6=14$
$\{S,b,d\}$	$\{a,c,D\}$	$\{(S,a),(b,c),(d,D)\}$	$5+1+6=12$
$\{S,c,d\}$	$\{a,b,D\}$	$\{(S,a),(S,b),(c,D),(d,D)\}$	$5+5+6+6=22$
$\{S,a,b,c\}$	$\{d,D\}$	$\{(a,d),(b,d),(c,D)\}$	$6+3+6=15$
$\{S,a,b,d\}$	$\{c,D\}$	$\{(a,c),(b,c),(d,D)\}$	$3+1+6=10$
$\{S,b,c,d\}$	$\{a,D\}$	$\{(S,a),(c,D),(d,D)\}$	$5+6+6=17$
$\{S,a,c,d\}$	$\{b,D\}$	$\{(S,b),(c,D),(d,D)\}$	$5+6+6=17$
$\{S,a,b,c,d\}$	$\{D\}$	$\{(c,D),(d,D)\}$	$6+6=12$

From the table, we can see that the minimal cut is the cut $(X,\overline{X}) = \{(S,a),(b,c),(b,d)\}$ where $X = \{S,b\}$ and $\overline{X} = \{a,c,d,D\}$ with capacity 9.