Chapter 4. Graph Theory [15 hrs.][32 Marks]

Graph Theory

Graph is a discrete structure consisting of vertices and edges that connect these vertices.

A graph G = (V, E) consists of two things.

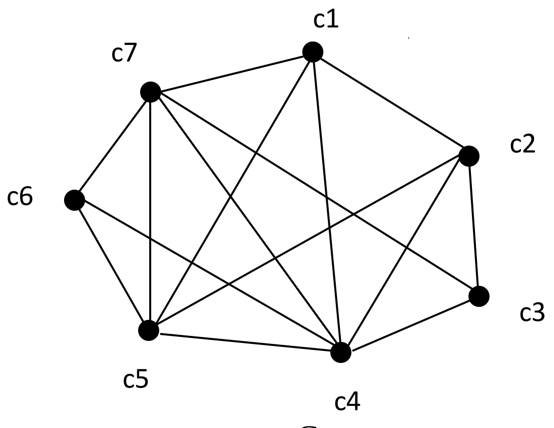
i. A set V=V(G) whose elements are called vertices or nodes of G.
ii. A set E=E(G) of unordered pairs of distinct vertices called edges of G.

Use of graph (example)

A major publishing company has ten editors in the scientific, technical and computing areas. These ten editors have a standard meeting time during the first Friday of every month and have divided themselves into seven committee to meet later in the day to discuss specific topics of interest of company. Namely, advertising, recurring reviewers, contacting new potential authors, finances, used copies and new editors, competing text books and text book representatives.

The 10 editors have decided on seven committees C_1 = {1, 2, 3} , C_2={1,3,4,5}, C_3={2, 5, 6, 7}, C_4={4, 7, 8, 9}, C_5={2, 6, 7}, C_6={8, 9, 10} C_7={1, 3, 9, 10}

They have set aside three time periods for the seven committees to meet on those Fridays when a ten editors are present. Some committees can not meet during the same period because one or two of the editors are on both committees.



G

In this figure, there are seven small circles representing the seven committees and a straight line segment is drawn between two circles if the committees they represent have at least one committee members in common. In other words this means that two committee should not be scheduled to meet at a same time.

A graph consist of a finite non empty set V of objects called vertices and a set E of element subsets of V called edges. So, graph is a pair of two sets V and E, vertex set and edge set respectively.

Vertices are also called points or nodes. Edges are sometimes called lines or arcs.

The vertex of set G is V(G) = { c_1, c_2,…………… c_7} and the edge set E(G)= { { c_1, c_2}, { c_1, c_3}, { c_1, c_5}, { c_1, c_7}, { c_2, c_3}, { c_2, c_4}, { c_2, c_7},{ c_3, c_4}, { c_3, c_5}, { c_4, c_5}, { c_4, c_6},{ c_4, c_7}, { c_6, c_7}}

- The number of vertices in G is often called the order of G while the number of edges is its size. We use n and m for order and size respectively.
- Since, the vertex set of every graph is non-empty, the order of every graph is at least 1.
- A graph with exactly one vertex is called a trivial graph.

Types of graphs

1. Labeled graph
2. Unlabeled graph

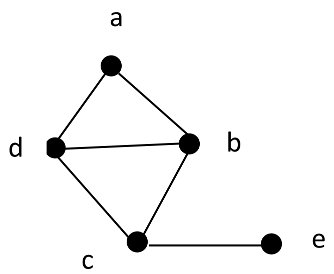


Fig: Labeled graph

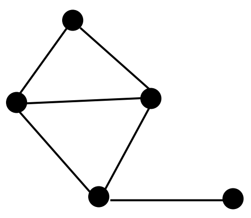


Fig: Unlabeled graph

Simple Graph:

Simple graph is 2-tuple consists of non-empty set of vertices V, a set of unordered pairs of distinct elements of vertices called edges. We represent this graph as G=(V,E). This kind of graph has undirected edges, no loops and no multiple edges.

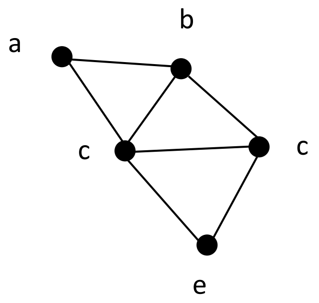


Fig : Simple graph

In above graph G=(V, E) , set of vertices.

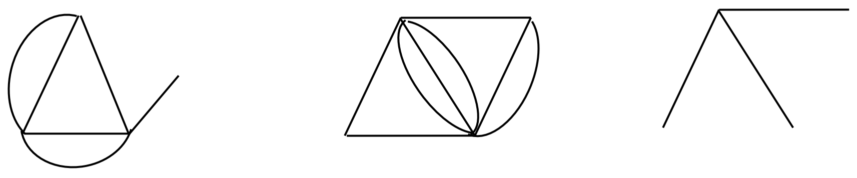
V(G) or V = {a, b, c, d, e} and set of edges.

E(G) or E= { {a, b}, {a, c}, {b, c},{b, d},{c, d}, {c, e}, {d, e}}

Multi graph

In a graph, two vertices are either adjacent or they are not, that is , two vertices are joined by one edge or no edges. A multigraph M consist of a finite non empty set V of vertices and a set of E of edges where every two vertices of M are joined by finite number of edges.

- If two or more edges join the same pair of vertices, then these edges are called parallel edges.
- In multi graph muliple edge occurs.
- In a pseudograph, an edge is also permitted to join a vertex to itself. Such an edge is called a loop.

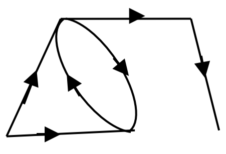


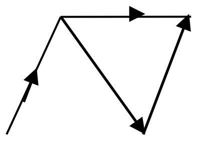
M_1 M_2 M_3

Every graph is a multigraph

Directed graph (Digraph)

A directed graph (V, E) consists of set of V of vertices, a set E of edges that are ordered pairs of elements of V. In this graph, loop is allowed but no two vertices can have multiple edges in same direction





D_1 D_2

D_2 is oriented graph

Directed Mutligraph

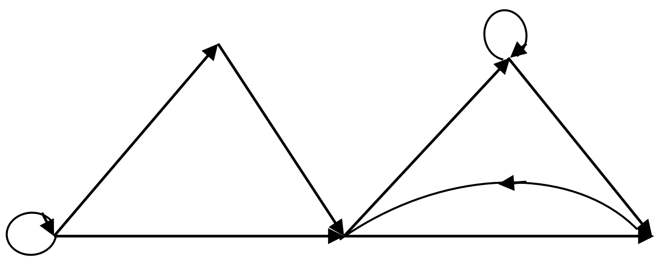


Fig: Directed Mutligraph

Graph Terminologies

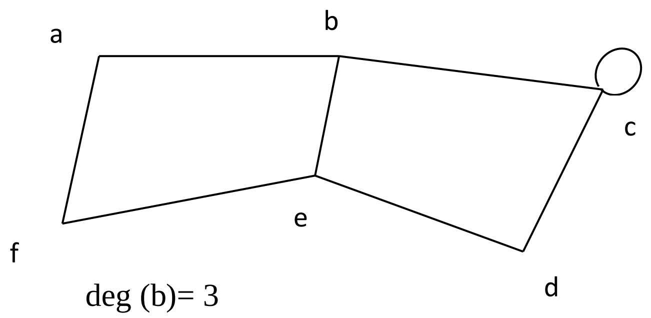
Adjacent : Two vertices u, v in an undirected graph G are called adjacent or neighbor if {u,v} is an edge.

The edge e is called incident with the vertices u and v if e={u, v}

Degree of a vertex :

The degree of a vertex is undirected graph is a number of edges incident with it, except a loop at a vertex. Degree vertex v is denoted by deg(v). A vertex of degree zero is called isolated vertex and are with degree one is called pendant vertex.

Loop in a vertex counts twice to the degree. e.g.: find the degrees.



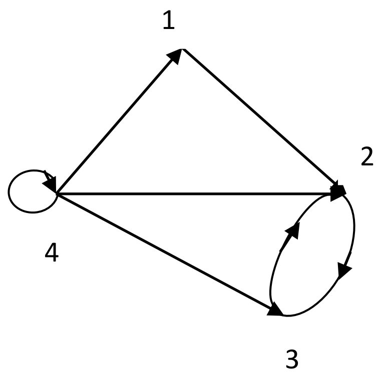
deg (a)=2	deg (b)= 3
deg (c) =4	deg (d) = 2
deg (e) = 3	deg (f) = 2

Let (u, v) be an edge representing edge of a directed graph G, u is called adjacent to v and v is called adjacent from u. The vertex v is called initial vertex and the vertex v is called terminal or end vertex. Loop had same initial and terminal vertex.

In directed graph the in-degree of a vertex v, denoted by $deg^{-}(v)$, is a number of edges, that have v as their terminal vertex. The out-degree of a vertex v, denoted by $deg^{+}(v)$, is the number of edges that have v as their initial vertex.

Loop at a vertex adds up both in-degree and out-degree to one or more than calculated in-degree and out-degree.

- Find in-degree and out-degree of each vertex



In-degrees:

$deg^{-}(1) = 2$

$deg^{-}(2) = 3$

$deg^{-}(3) = 2$

$deg^{-}(4) = 1$

Out-degrees:

$deg^{+}(1) = 1$

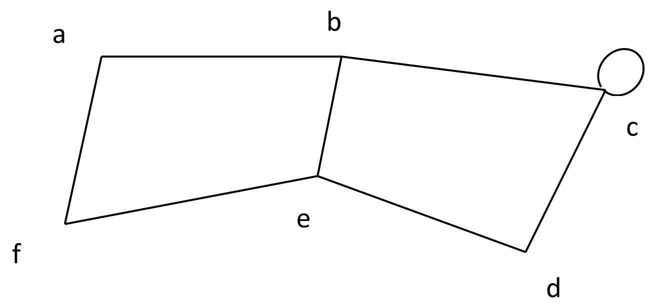
$\deg^+(2) = 1$

$\deg^+(3) = 1$

$\deg^+(4) = 4$

Theorem 1 : The Handshaking Theorem

Let G=(V,E) be an undirected graph with a edges Then 2e = $_v$deg (v)



Proof :

deg(a) = deg(f)= deg(d)=2

deg(b) = deg(e)=3

deg(c) = 4

We have

LHS = 2*e

= 2× 8

=16

RHS =$\sum_{v \in V} \deg(v)$

=2+ 2+ 2 + 3 + 3 + 4

=16

Therefore , LHS= RHS

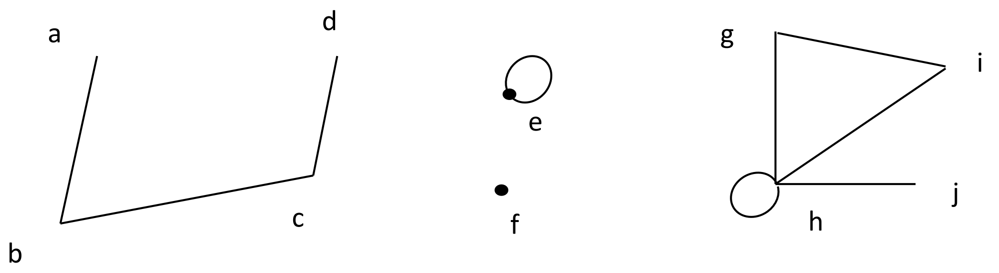
Theorem 2: An undirected graph has an even number of vertices of odd degree.

- Let V_1 and V_2 be set of vertices of even and odd degrees respectively. In an undirected graph G = (V, E)
  We have,

$$2e = \sum_{v\in V} \deg(v) = \sum_{v\in V1} \deg(v) + \sum_{v\in V2} \deg(v)$$

From equality above, we can say the left part is even i.e. 2e is even, the sum of deg (v) for $v \in V_1$ is even since every vertices has even degree. So for the left hand to be even sum of deg (v) for $v \in V_2$ must be even. Since all vertices in the set V_2 have odd degree the number of such vertices must be even for the sum to be even.

Example: Which vertices in following graph are isolated which are pendant and what is maximum degree? What types of graph is it?



– Vertex f is isolated, vertices a, d and j are pendant. Maximum degree is deg (h) = 5.

This graph is pseudograph (undirected loops)

Example : Determine the number of its edges and sum of the degrees of all its vertices.

- There are 9 edges and sum of degrees is 18.

Theorem 3:

The sum of the in-degrees of all vertices in a digraph = The sum of the out-degrees = The number of edges.

Let G= (V, E) be a graph with directed edges,

Then,

$$\sum_{v\in V}(\deg^{-}(v))= \sum_{v\in V}(\deg^{+} (v))= | E |$$

Complete graph

- The complete graph of n vertices is the simple graph that contains exactly one edge between each pair of distinct vertices.

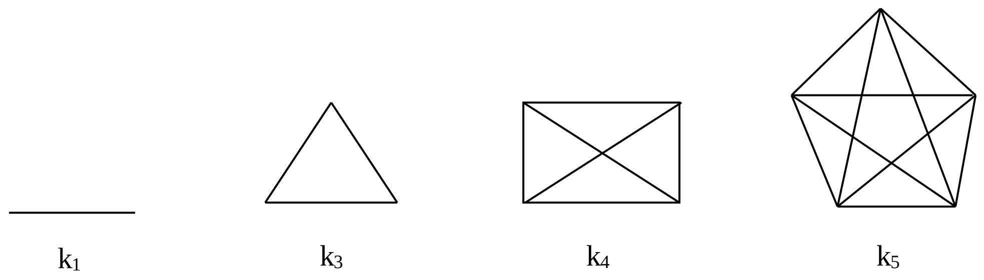
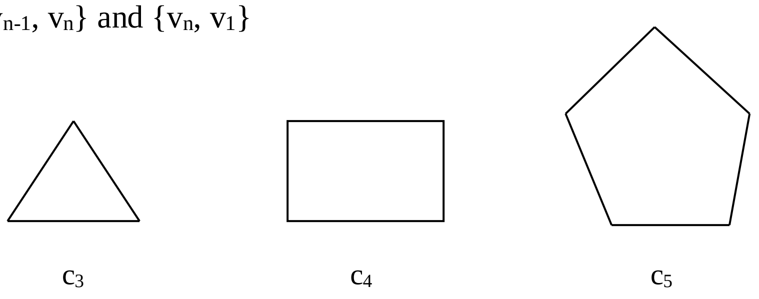


Fig: Complete graph

- A complete graph of order n is denoted by k_n. Since every distinct vertices of k_n are joined by an edge, the number of pairs of vertices in k_n is $\binom{n}{2}$ the size of k_n is $\binom{n}{2} = \frac{n(n-1)}{2}$

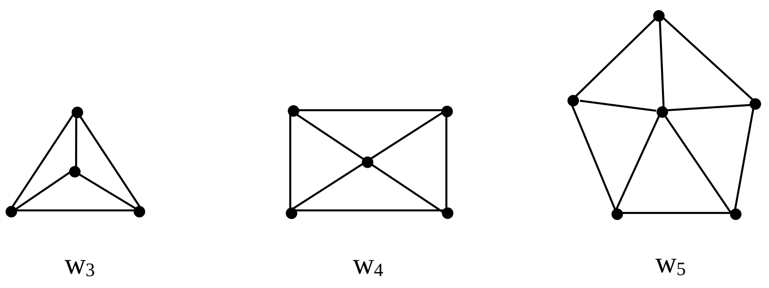
Cycle:

The cycle C_n, $n \geq 3$, consists of n vertices $v_1, v_2, \ldots\ldots\ldots\ldots\ldots\ldots$ v_n and edges $\{v_1, v_2\}$, $\{v_2, v_3\}$, $\ldots\ldots\ldots\ldots\ldots\ldots \{v_{n-1}, v_n\}$ and $\{v_n, v_1\}$



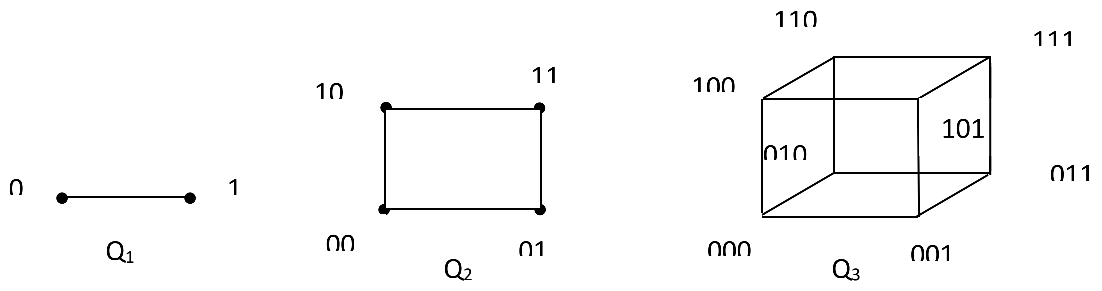
Wheel :

When a new vertex is added to a cycle C_n and this new vertex is connected to each of the n vertices in C_n, we obtain a wheel W_n.



n-cubes

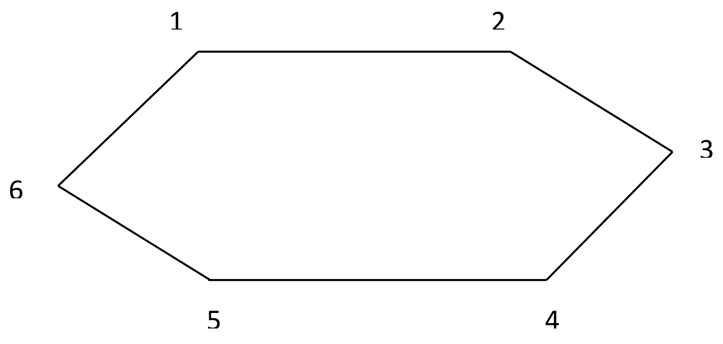
The n-cube, denoted by Qn, is the graph that has vertices representing the 2^n bit string of length of n. Two vertices are adjacent if and only if the bit strings that they represent differ in exactly one bit position.



Bipartite Graph

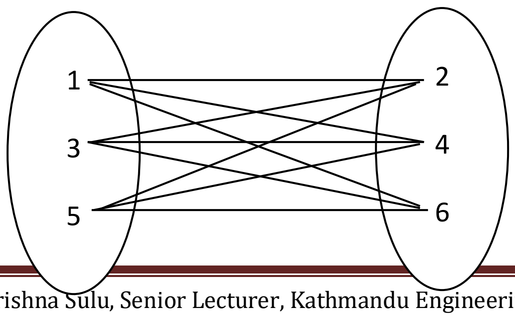
A simple graph G is bipartite if its vertex set can be partitioned into two disjoint subsets V_1 and V_2 such that every edge in the graph connects a vertex from the set V_1 to the vertex of set V_2 . No two vertices of the same set are connected by an edge

E.g.: Is C_6 Bipartite.

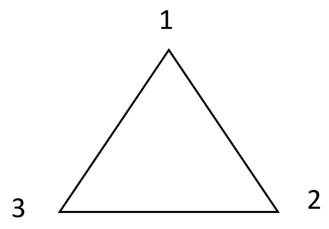


Its vertex set can be partitioned into two sets V_1={1,3,5} and V_2 ={2, 4,6}

every edge of C_6 connects a vertex in V_1 with a vertex in V_2.



Q) *Is K_3 Bipatite?*

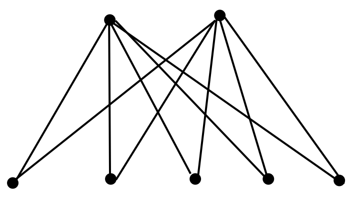


=No

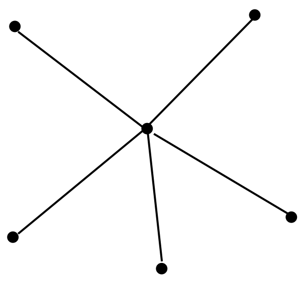
Complete Bipartite Graph

The complete bipartite graph $K_{m,n}$ is the graph where the vertex set is partitioned into two subsets of m and n vertices, respectively. In this graph, there is an edge between two vertices if and only if two vertices are in different subsets of vertices.

E.g. : $K_{2,5}$



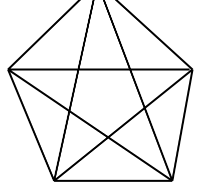
E.g. : $K_{1,5}$



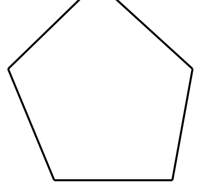
Operations on Graphs

– <u>Sub graph</u>

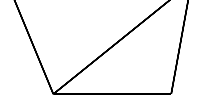
A subgraph of a graph G = (V, E) is a graph H = (W, F), where wV and F E.



k_5



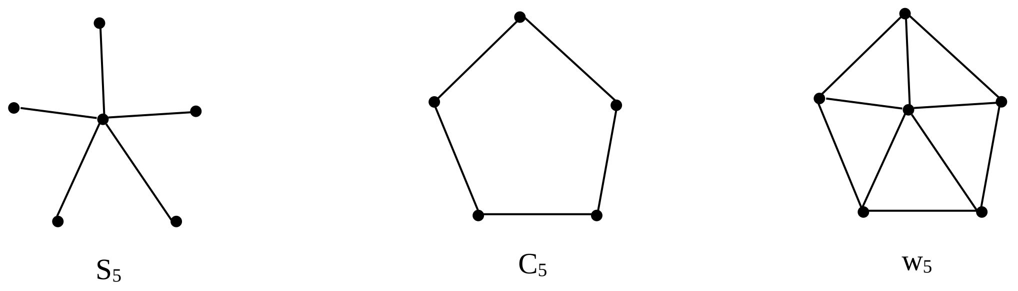
c_5



sub graph of K_5

Union

The union of two simple graphs $G_1=(V_1, E_1)$ and $G2 = (V_2, E_2)$ is the simple graph with vertex set $V=(V_1UV_2)$ and set $E=(E_1UE_2)$. The union is denoted by G_1UG_2



$S_5UC_5 = W_5$

Representing graphs

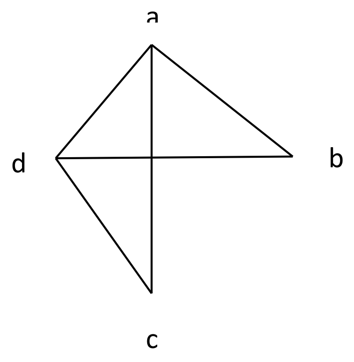
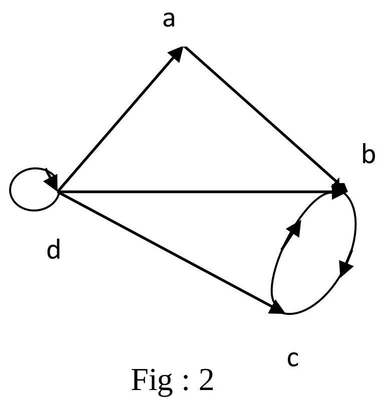


Fig : 1

Edge list for simple graph

Vertex	**Adjacent vertices**
a	b, c, d
b	a, d
c	a, d
d	a, b, c

Edge list for Directed graph



Initial Vertex	End vertex
a	b
b	c
c	b
d	a, b, c, d

Fig : 2

Adjacency Matrix:

Given a simple graph G=(V, E) with |V|=n of the vertices of the graph are listed in some arbitrary order like V_1, V_2 , ………………V_n, the adjacency matrix A of G, with respect to the order of vertices is n-by-n zero-one matrix ($A=[a_{ij}]$) with the condition

$$a_{ij}=\begin{cases} 1 \; if(Vi, Vj) \; is \; an \; edge \; of \; G \\ 0 \qquad\qquad otherwise \end{cases}$$

Adjacency matrix for undirected graph is symmetric, in case of pseudograph or multigraph the representation is similar but the matrix here is not zero-one matrix rather the $(i, j)^{th}$ entry of matrix contains the no. of edges appearing between the pair of vertices.

In case of directed graph, we can extend the same concept as in undirected graph as below

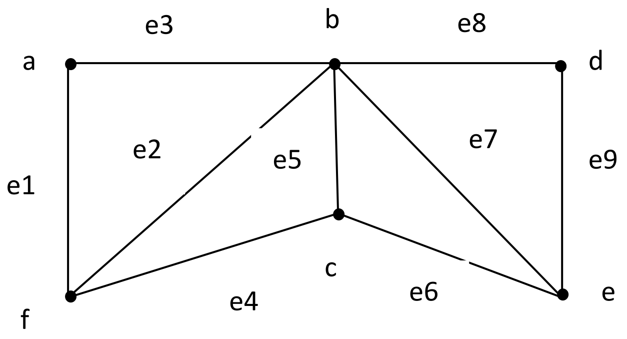
$$a_{ij}=\begin{cases} 1 \; if(Vi, Vj) \; is \; an \; edge \; of \; G \\ 0 \qquad\qquad otherwise \end{cases}$$

Incidence Matrix

Given an undirected graph G = (V, E). Assume that vertices of the graph V_1, V_2 ……………. V_n and the edges of graph are e_1, e_2 ……………e_m . The incidence matrix of a graph with respect to the above ordering of V and E is n by m matrix. Where

$$m_{ij}=\begin{cases} 1 \text{ when edge, is incident with V1} \\ \quad 0 \qquad\qquad otherwise \end{cases}$$

When the graph is not simple then graph can be represented by using incidence matrix where multiple edges corresponds to two different columns with exactly same entries. Loops are represented with column with only one entry.



The order of vertices be a, b, c, d, e, f, and edges order be e1, e2, e3, e4, e5, e6, e7, e8, e9

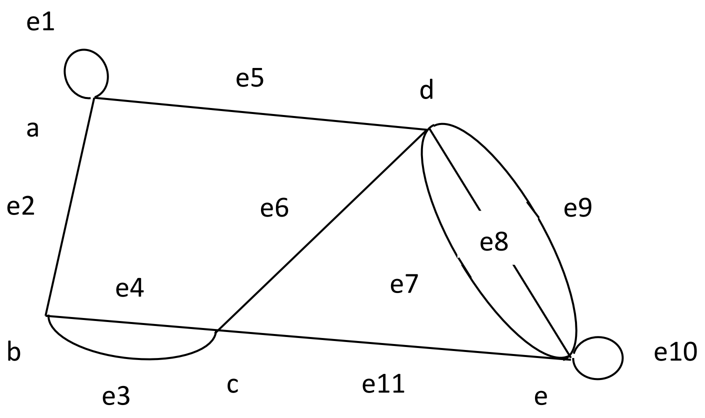
Adjacency Matrix:

$$\begin{array}{c|cccccc} & a & b & c & d & e & f \\ \hline a & 0 & 1 & 0 & 0 & 0 & 1 \\ b & 1 & 0 & 1 & 1 & 1 & 1 \\ c & 0 & 1 & 0 & 0 & 1 & 1 \\ d & 0 & 1 & 0 & 0 & 1 & 0 \\ e & 0 & 1 & 1 & 1 & 0 & 0 \\ f & 1 & 1 & 1 & 0 & 0 & 0 \end{array}$$

Incidence matrix

$$\begin{array}{c|ccccccccc} & e_1 & e_2 & e_3 & e_4 & e_5 & e_6 & e_7 & e_8 & e_9 \\ \hline a & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ b & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 \\ c & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ d & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ e & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ f & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \end{array}$$

2



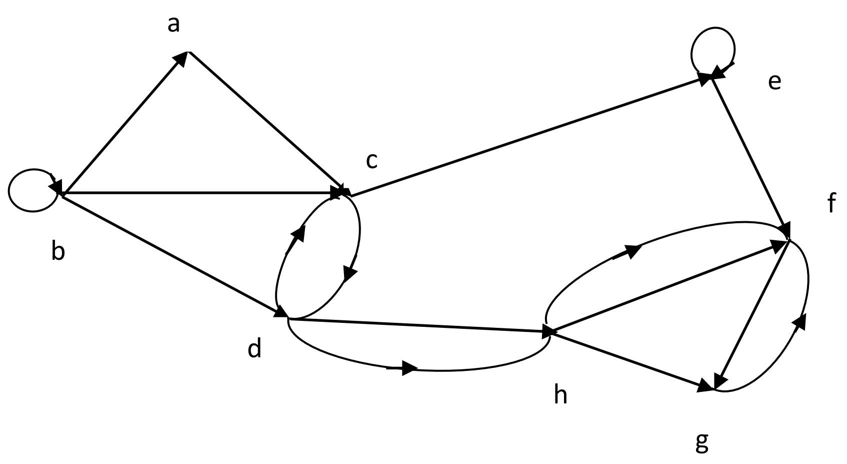
Adjacency matrix

$$\begin{pmatrix} 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 2 & 0 & 0 \\ 0 & 2 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 3 \\ 0 & 0 & 1 & 3 & 1 \end{pmatrix}$$

Incidence matrix

$$\begin{array}{c|ccccccccccc} & e_1 & e_2 & e_3 & e_4 & e_5 & e_6 & e_7 & e_8 & e_9 & e_{10} & e_{11} \\ \hline a & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ b & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ c & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ d & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ e & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{array}$$

3



Adjacency matrix

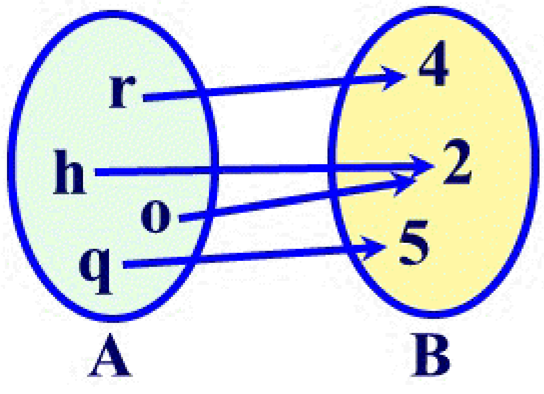
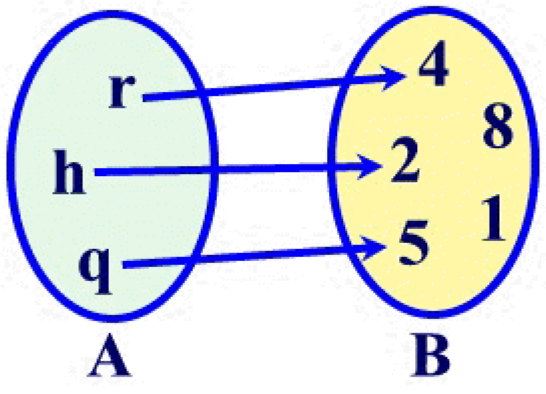
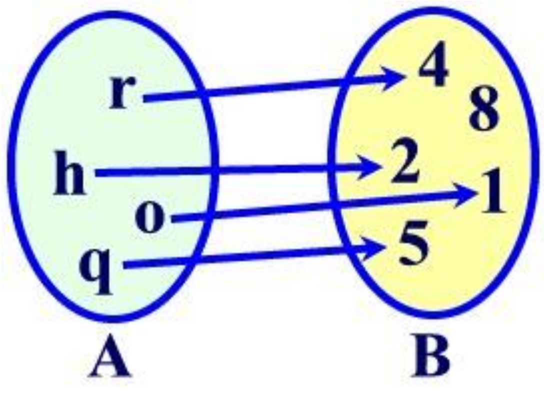
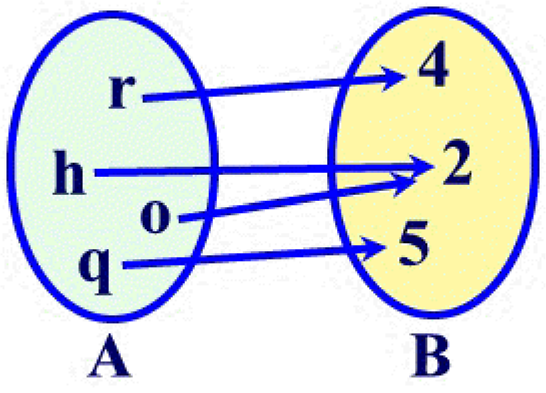
$$
\begin{array}{c|cccccccc|}
 & a & b & c & d & e & f & g & h \\
a & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
b & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\
c & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\
d & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 2 \\
e & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\
f & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
g & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
h & 0 & 0 & 0 & 0 & 0 & 2 & 1 & 0
\end{array}
$$

Isomorphism of Graphs

Two simple graphs $G_1=(V_1, E_1)$ and $G_2=(V_2, E_2)$ are isomorphic if there is 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 a and b in $V_{1.}$ Such a function is called isomorphism.

A function *f* from **A** to **B** is called **one-to-one** (or 1-1) if whenever *f* (*a*) = *f* (*b*) then *a* = *b*. No element of **B** is the image of more than one element in **A.**

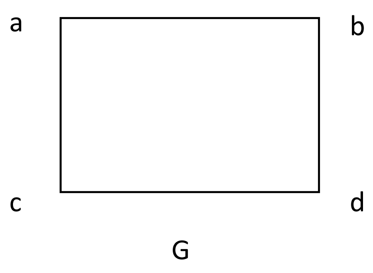
In a one-to-one function, given any *y* there is only one *x* that can be paired with the given *y*. Such functions are referred to as injective.

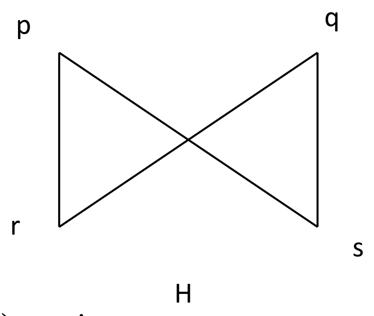
 **"Onto"** (all elements in B are used)	 **NOT "Onto"** (the 8 and 1 in Set B are not used)
 **"One-to-One"**	 **NOT "One-to-One"**

A function *f* from **A** to **B** is called **onto** if for all *b* in **B** there is an *a* in **A** such that *f* (*a*) = *b*. All elements in **B** are used.

Such functions are referred to as **surjective**.

E.g. : G= (V, E) and H(W, F)



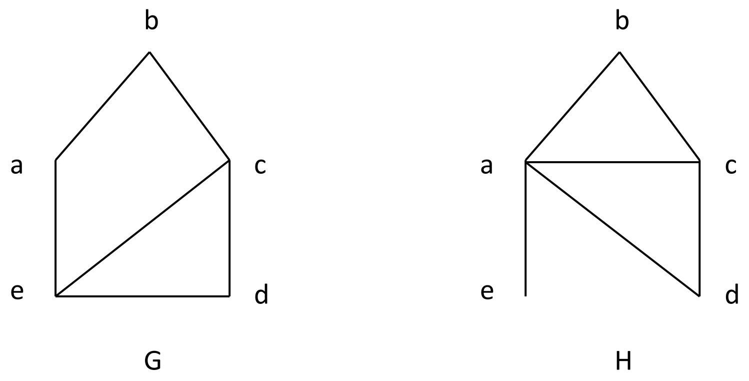


The function f with f(a) = p, f(b) = s, f(c) = r, f(d) =q is a one to one correspondence between V and W.

We can show two simple graphs are not isomorphic by showing they don't share a property is called invariant. These properties are described as follows:

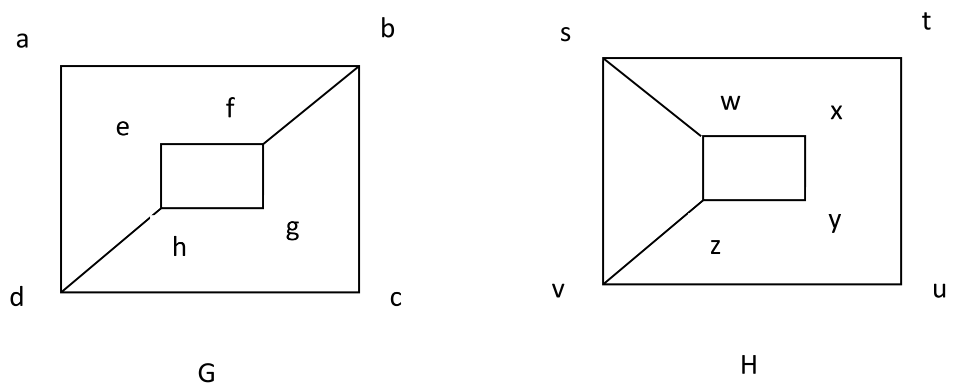
1) Isomorphic simple graphs must have the same number of vertices.
2) Isomorphic simple graphs must have same number of edges.
3) The degree of vertices must be same because the number of edges from the vertex is determined by degree.

Example 1 : show that the graphs displayed below are not isomorphic

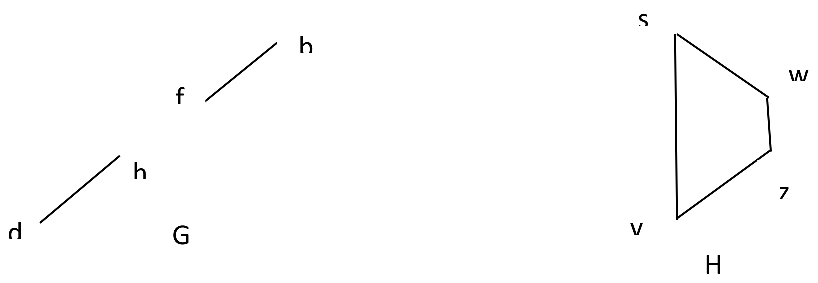


– Both graphs have 5 vertices and six edges. But H has a vertex of degree one, namely e, whereas G has no vertices of degree one. Hence G and H are not isomorphic.

Example 2 :

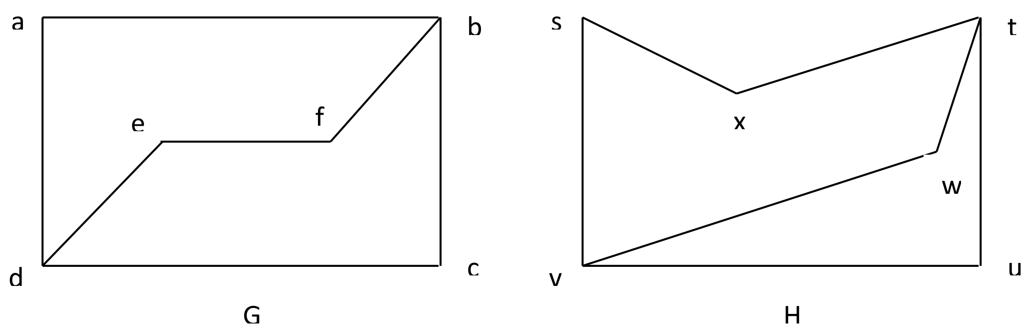


– Both graphs have eight vertices and ten edges. Also, they both have four vertices of degree two and four vertices of degree three. However, G and H are not isomorphic. Since, deg(a)=2 in G, a must correspondence to either t, u, x or y in H because these are the vertices of degree two in H. Here, each of these four vertices in H is adjacent to another vertex of degree two in H, which is not true for a in G.
– Another way to see that two graphs are not isomorphic is to note that subgraphs formed by connecting the edges from the vertex with same degree in both the graphs are not isomorphic. For e.g.: subgraphs of G and H made up of vertices of degree three and the edges connecting them are not isomorphic.



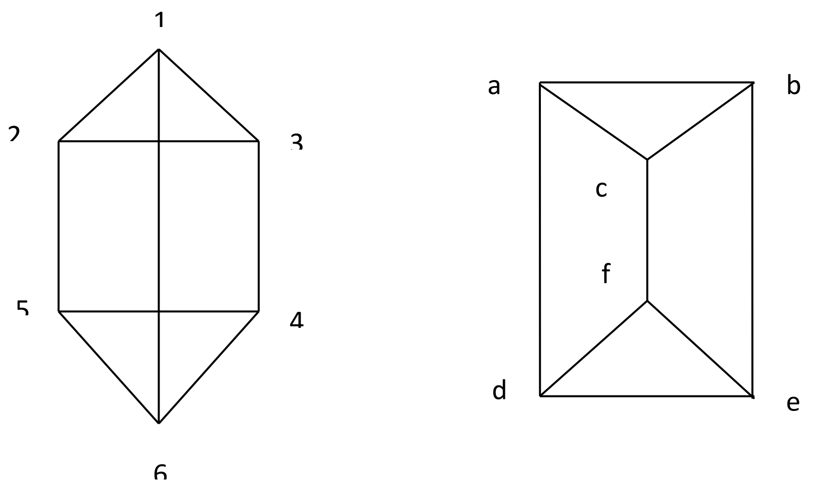
– To show isomorphism of graphs, we can also use adjacency matrix. For this, adjacency matrices of two graphs are same.

Example : 3 Determine whether the graphs are isomorphic



– Both graphs have six vertices and seven edges. Both have four vertices of degree two and two vertices of degree three. Also, Subgraphs are isomorphic. Here, we cannot say these graphs are isomorphic or not. For this, we can use adjacency matrix with the order of vertices a, b, c, d, e, f and w, t, u, v, s, x. If we can see similar matrices after drawing adjacency matrices. Hence, these graphs are isomorphic.

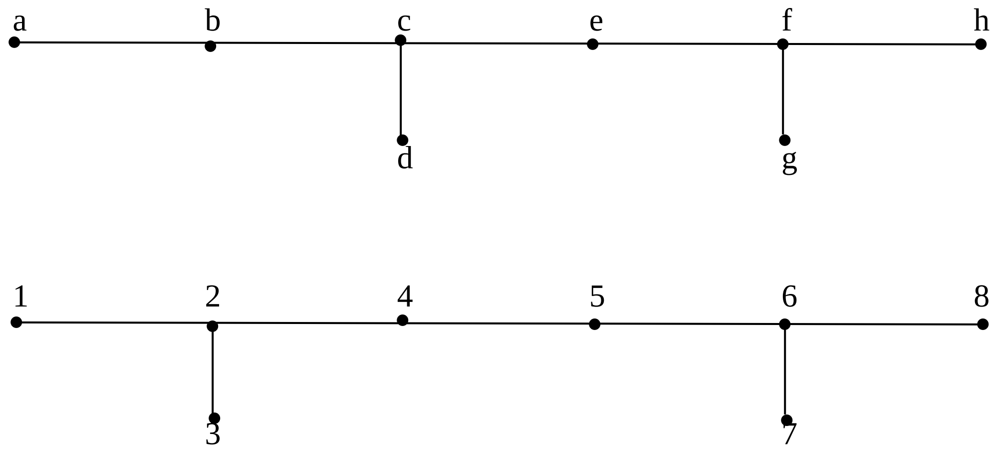
Example 4 : Determine whether given two graphs are isomorphic or not?



– In above graphs, no.of vertices is same, no. of edges are same and all vetices in both the graphs have degree 3.

Since the invariants agree on both the graphs, we can try out to find the function that is isomorphism. Take the sequence of vertices from the first graph as 1, 2, 3, 4, 5 and 6. Now, define f(1) = c, f(2)=a here, there is adjacency preservation since we have {1,2}as and edge in the first graph where as {f(1), f(2)}={c,a} is an edge in second graph. Similarly, we can assign f(3)=b, f(4)=e, f(5)=d, f(6)=f. Since we found one to one correspondence between verticesof two graphs preserving the adjacency, above graphs are isomorphic. Here, the adjacency matyrices of two graphs in which vertices are ordered in terms of function i.e, 1, 2, 3, 4, 5 and 6 for first graph and c, a, b, e, d and f in the second graph are same.

one to one correspondence between verticesof two graphs preserving the adjacency, above graphs are isomorphic. Here, the adjacency matyrices of two graphs in which vertices are ordered in terms of function i.e, 1, 2, 3, 4, 5 and 6 for first graph and c, a, b, e, d and f in the second graph are same.**Example 5 :**



- Both graphs have 8 vertices, 7 edges in both graphs two vertices have degree 3, 4 vertices have degree 1 and remaining 2 vertices have degree 2. Since, the invariants agree in both the graphs, we can continue to get the function such that it is isomorphism.

  However, in case of first graph, the subgraph containing vertex with degree 3 is not isomorphic with any graph formed by connecting edges with vertex 2 or 6 (both of degree 3). Hence, two graphs are not isomorphic.

Walk, Paths, Circuit

Walk:

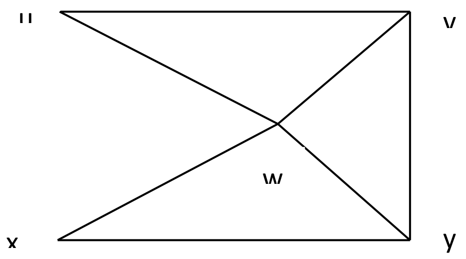
A walk is defined as a finite alternative sequence of vertices and edges of the form.

$v_i\ e_j,\ v_{i+1}\ e_{j+1},\ v_{i+2}\ e_{j+2}\ldots\ldots\ldots\ldots\ldots e_k\ v_m.$

Which begins and ends with vertices, such that

i) each edge in the sequence is incident on the vertices preceding and following it in the sequence.

A walk that begins and ends at the same vertex is called a closed walk . A walk that is not closed is an open walk. If u be initial vertex, v be terminal vertex. If u=v, then walk w is closed and u≠v it is open . As we move from one vertex of w to next, we are actually encountering or traversing edges of G, Possibly traversing some edges of G more than once. The number of edges encountered in a walk (including multiple occurrence of an edge) is called the length of the walk.



w: x, y, w, y, v, w is therefore a walk. Length of walk is the no. of edges. Here 5 is the length.

A walk of length 0 is a trivial walk.

– We define a u-v **trail** in a graph G to be a u-v walk in which **no edge** is traversal more than once.

Since w:x, y, w, y, v, w is not an x-w trail as the edge wy is repeated.

T: u, w, y, x, w, v is a u-v trail.

Trail T repeats the vertex w.

– A u-v walk is a graph in which **no vertices** are repeated in u-v **path.**

P : u, w, y, v is a u-v **path.**

If no vertex in a walk is repeated then no edge is repeated either. Hence, every path is a trail.

- A circuit in graph G is a closed trail of length 3 or more. Hence, a circuit begins and ends at the same vertex but repeats no edges. In a circuit, vertices can be repeated in addition to the first and last.

For example:

c : y, w, u, v, w, x, y

e: x, y, w, u, v, w, x

c: w, x, y, w, u, v, w

is a circuit.

Cycle

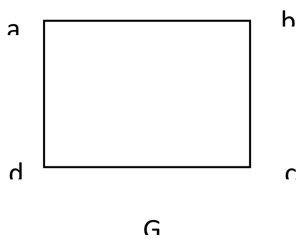
A circuit that repeats no vertex except for the first and last is a cycle. A K-cycle is a cycle of length K. A 3-cycle is also referred to as a triangle. A cycle of odd length is called odd cycle, a cycle of even length is called as even cycle.

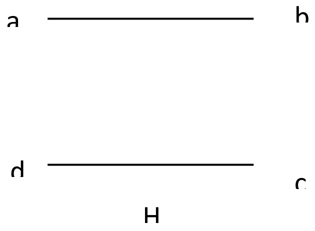
For example:

C' = x, y, v, w, x is a cycle.

Connectedness

- An undirected graph is called connected if there is a path between every pair of distinct vertices of the graph.
- There is a simple path between every pair of distinct vertices of a connected undirected graph.



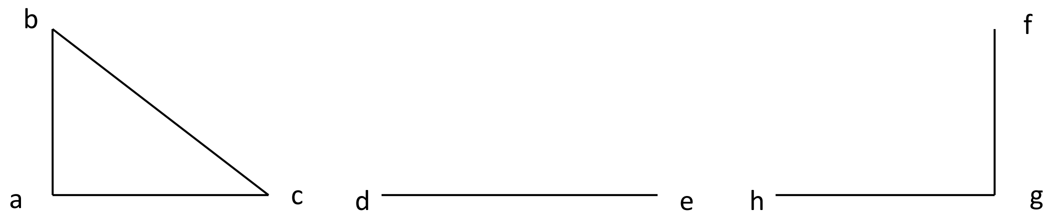


Graph G is connected but graph H is not connected.

- A graph that is not connected is the union of more than one connected graphs that do not share the common vertex. These disjoint connected sub graphs are called connected components of a graph.

For example:

What are the connected components of the following graph?



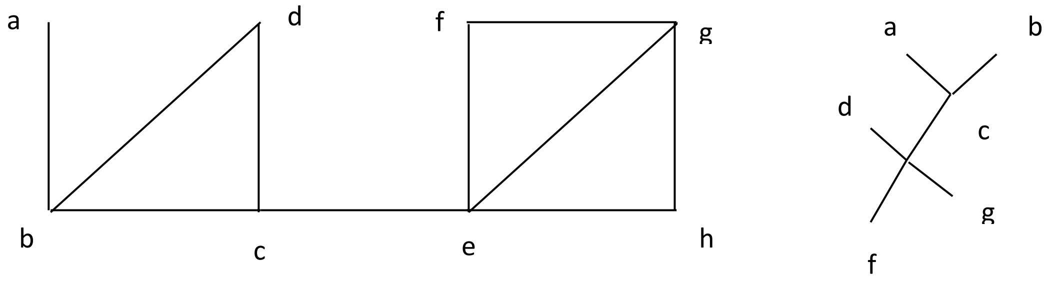
connected components are {a, b, c,}, {d, e}, {f, g, h}

Cut Edges and Vertices

If one can remove a vertex (and all incident edges) and produce a graph with more connected components, the vertex is called a cut vertex.

If removal of an edge creates more connected component, the edge is called a cut edge or bridge.

Q. Find the cut vertices and cut edges in the following graph:



Cut vertices : c and e

Cut edge ; {c, e}

Vertex V is cut vertex of G iff deg v>2

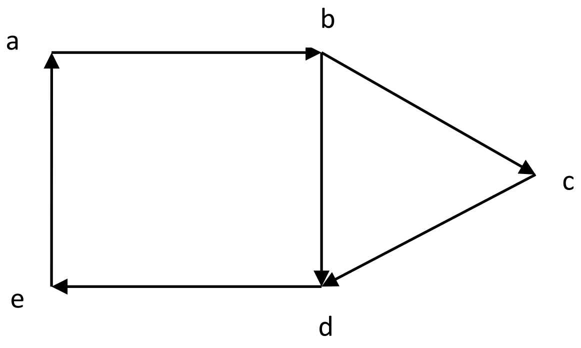
Connectedness in Directed graphs

A directed graph is strongly connected if there is a directed path between every pair of vertices.

A directed graph is weakly connected if there is a path between every pair of vertices in the underlying undirected graph.

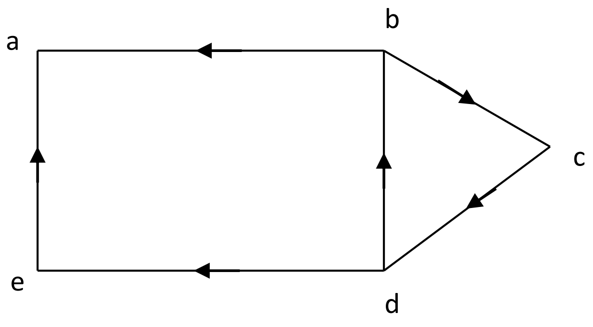
For example:

Q.N. Is the following graph strongly connected?



The graph is strongly connected because there is a directed path between every pair of vertices. If a directed graph is strongly connected, then it must also be weakly connected.

Q2. Is the following graph strongly connected?



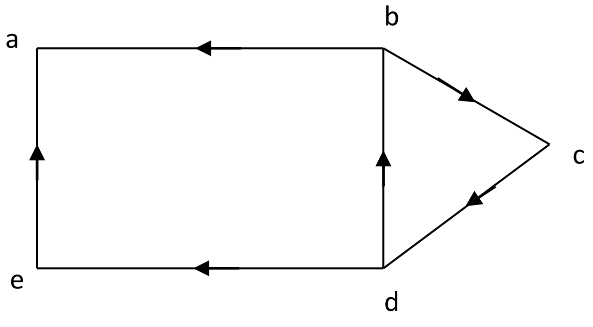
This graph is not strongly connected because there is no directed path between a and b, a and e etc.

However, it is weakly connected (imagine the graph as an undirected graph)

The subgraphs of a directed graph of G that are strongly connected but not contained in larger strongly connected subgraphs (the maximal strongly connected subgraphs) are called the strongly connected components or strong components of G.

For example:

What are strongly connected components of following graphs:



This graph has three strongly connected components

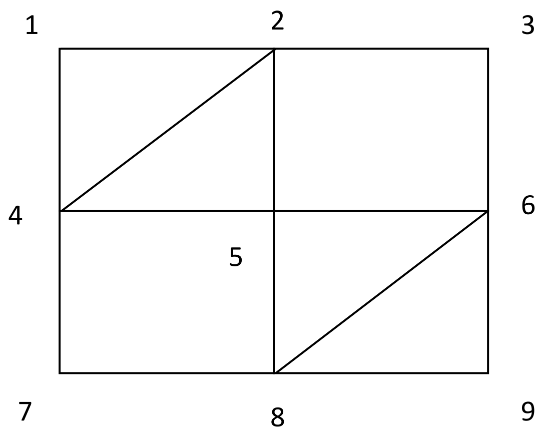
- The vertex a
- The vertex e
- The graph consists of
  V= { b, c, d} and
  E={{b,c}, {c,d}, {d, b}}

Euler path and circuits

A circuit C in a graph G is called Eulerian circuits if C contains every edge of G. Since no edge is repeated in a circuit, every edge appears exactly once in Eulerian Circuits.

A connected graph that contains as Eulerian circuit is called Eulerian graph.

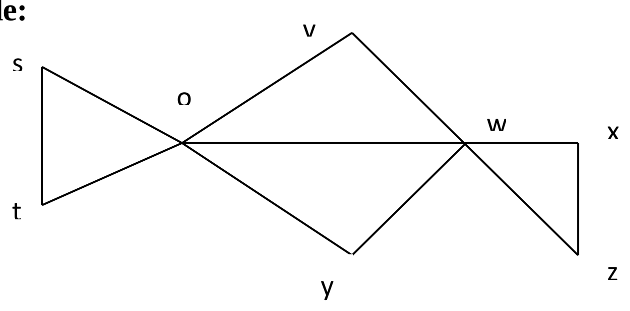
For example:



Euler circuit is 1, 2, 3, 6, 9, 8, 7, 4, 5, 8, 6, 5, 2

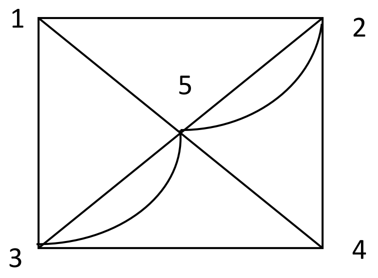
For connected graph G, we refer to an open trail that contains every edge of G as an Eulerian trail

For example:



T= u, s, t, u, v, w, y, u, w, x, z, w

An euler path is a path using every edge the graph exactly once.



No Euler circuit exist

Necessary and sufficient conditions

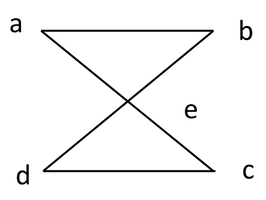
A connected mutligraph has a euler circuit iff each of its vertices has an even degree.

A connected multigraph has a eulerian trail iff exactly two vertices of G have odd degree .

For example:

Q. Which of the following graphs is a eularian circuit?

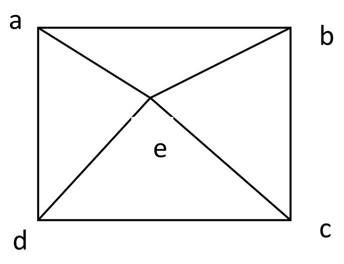
1.



Yes

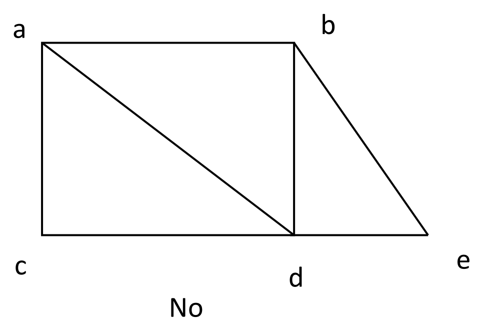
(a, e, c, d, e, b, a)

2.



No

3.

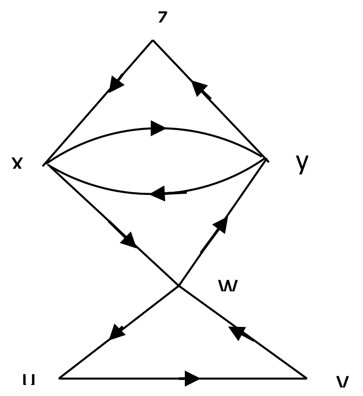


No

Euler trail : (a, c, d, e, b, d, a, b)

Eulerian for Directed graph

A non trivial connected digraph D is eulerian if and only if od V = id v for every vertex

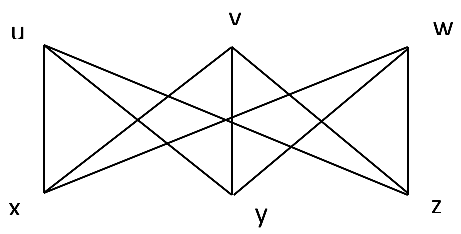


C: u, v, w, y, z, x, y, x, w, u, is eulerian circuit.

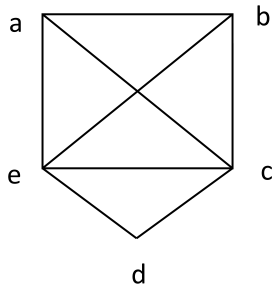
Hamilton Paths and Circuits

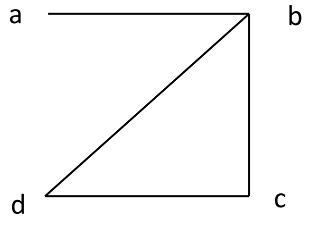
u

A cycle in graph G that contains every vertex of G is called a Hamiltonian cycle. Certainly a graph G (n≥ 3) is Hamiltonian path in G. If a graph contains Hamiltonian cycle, it contains Hamiltonian path.



c: u, x, v, y, w, z, u is a Hamiltonian cycle of G





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

Hamilton circuit

For Hamilton circuit

No vertex of degree 1

If a node has degree 2, then both edges incident to it must be in any Hamilton circuit incident to it must be in any Hamilton circuit.

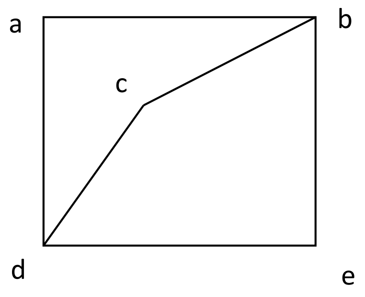
No smaller circuits contained in any Hamilton circuit (the start/end point of any smaller circuit would have to be visited twice).

A sufficient condition

Let G be a connected simple graph with a vertices with n$\geq$3.

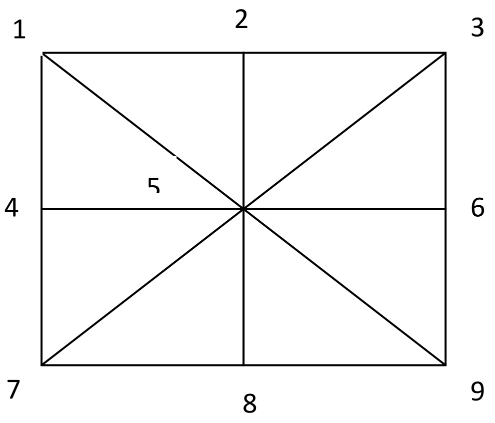
G has Hamilton circuit if the degree of each vertex is $\geq$ n/2.

For example 1:



In this graph, there is no hamilton circuit since node has degree 2 and both edges from it must be in Hamilton circuit which is not possible one of the hamilton path is a, b, c, d, e.

Example 2 :



In graph, we can have hamilton circuit, the circuit can be 1, 2, 3, 5, 6, 9, 8, 7, 4, 1. Since, there is circuit. We can have path also.

Dirac's theorem:

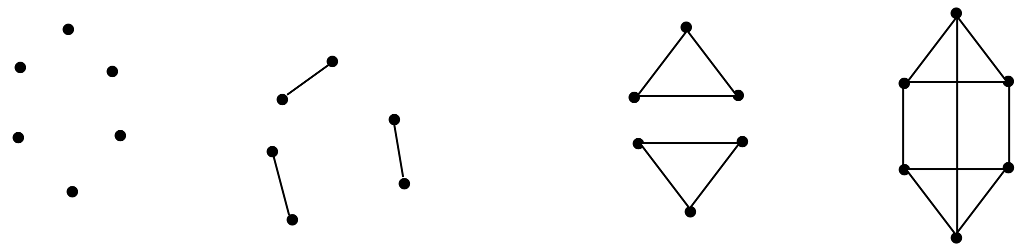
If G is a simple graph with n vertices with n≥3 such that the degree of every vertex in G is at least $\frac{n}{2}$, then G has a hamilton circuit.

Ore's Theorem:

If G is a simple graph with n vertices with ≥3 such that deg (u) + deg(v)≥n for every pair of non adjacent vertices u and v in G, then G has a Hamilton path.

Regular Graphs:

In graph theory, a regular graph is a graph where each vertex has the same number of neighbors i.e., every vertex has the same degree. A regular graph with vertices of degree k is called a k-regular graph or regular graph of degree k. Regular graph of degree at most 2 are easy to classify. A o-regular graph consists of disconnected vertices a 1-regular graph consists of disconnected edges, and a 2-regular graph consists of disconnected cycles. A 3-regular graph is known as cubic graph.



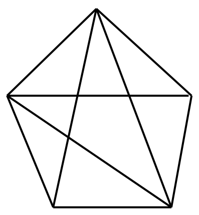
o-regular graph 1- regular graph 2-regular graph 3-regular graph

Planner graphs

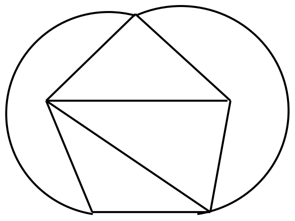
A graph is called a planar if it can be drawn in the plane without any edges crossing such drawing is called a planar representation of the graph.

For example:

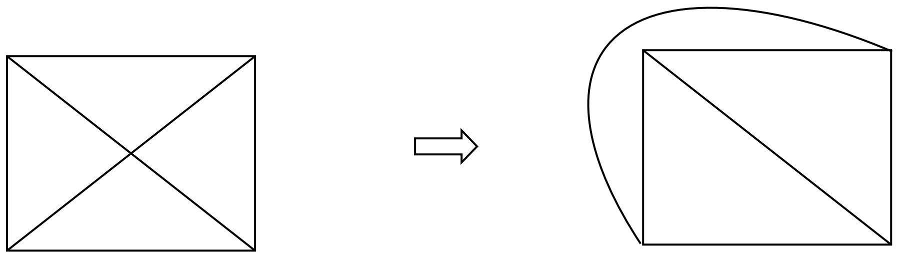
Draw the graph below as planar representation of the graph.



Solution,



2.

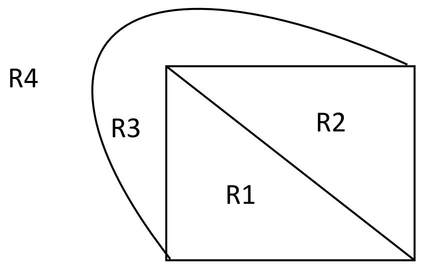


Not all graphs are planar

It may be difficult to show that graph is non-planner. We would have to show that there is no way to draw the graph without any edges crossing.

Regions

Euler showed that all planar representations of a graph split the plane into the same number of regions, including on unbounded region.



Any planar representation of K3,3 V_1 and V_2 connected to V_4 and V_5.

The four edges $\{V_1, V_4\},\{V_4, V_2\},\{V_2, V_5\},\{V_5, V_1\}$ form a closed curve that splits the plane into two regions R_1 and R_2.

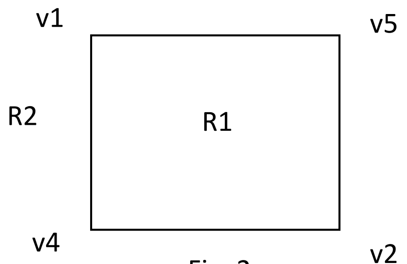


Fig. 2

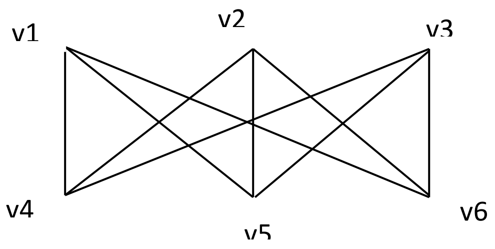
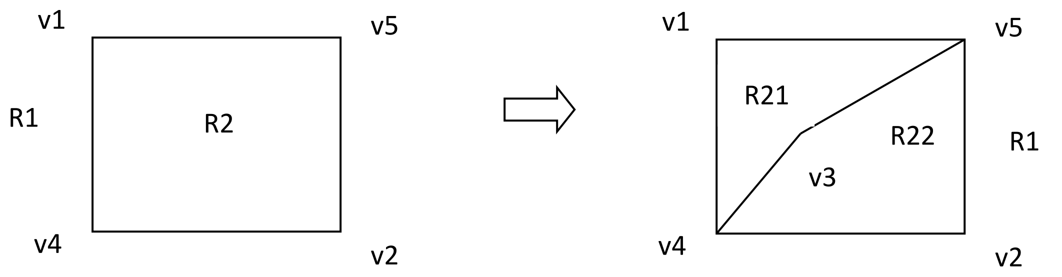


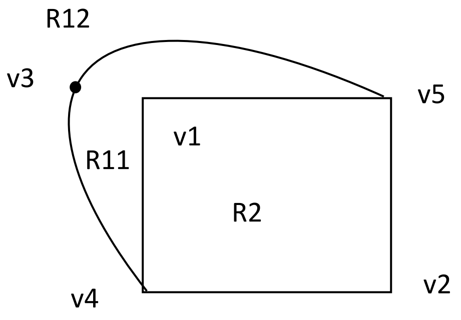
Fig. 1

Next, V_3 must be either in R_1 or R_2. Assume V_3 is in R_2. Then edges (V_3, V_4) and (V_3, V_5) separate R_2 into two sub regions R_{21} and R_{22}.



Now, there is no way to place vertex V_6 without forcing a crossing.

- → If V_6 is in R_1, then (V_6, V_3) must cross an edge.
- → If V_6 is in R_{21}, then {V_6, V_2} must cross an edge.
- → If V_6 is in R_{22}, then {V_6, V_1} must cross an edge.
- → Alternatively assume V_3 in R1. Then the edge (V_3, V_4) and (V_3, V_4) and (V_3, V_5) separate R_1 into two sub-regions R_{11} and R_{12}.



Now there is no way to place vertex V_6 without forcing a crossing.

- → If V_6 is in R_2 , then {V_6, V_3} must cross an edge.
- → If V_6 is in R_{11}, then {V_6, V_2} must cross an edge.
- → If V_6 is in $R_{12,}$ then {$V_{6,}$ V_1} must cross an edge.

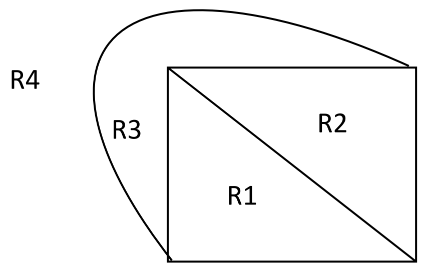
Consequently, the graph $K_{3,3}$ must be non- planar .

Euler devised a formula for expressing the relationship between the number of vertices, edges and regions of a planar graph.

- → These may help to determine if a graph can be planar or not.

Euler's Formula:

$\rightarrow$ Let G be a connected planar simple graph with e edges and v vertices . Let r be the number of regions in a planar representation of G. Then r=e-v + 2



no. of edge, e = 6

no. of vertices, v=4

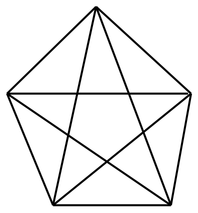
no. of regions, r = e – v + 2

= 4

Corollary 1: If G is a connected planar simple graph with e edges and v vertices where $v \geq 3$, then $e \leq 3v-6$

For example:

Is K_5 planar?



k_5

⇨ K_5 has 5 vertices and 10 edges.

Here , $V \geq 3$

So, if K_5 is planar, it must be true that

$e \leq 3v-6$

$3v-6 = 3\times5- 6 = 15-6= 9$

Thus, e must be less than or equal to 9

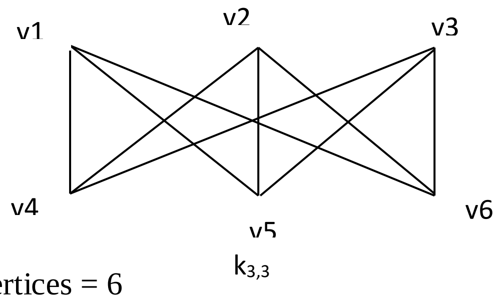
But e = 10

Therefore K_5 is non-planar.

Corollary 2 : If G is a connected planar simple graph, the G has a vertex of degree not exceeding 5.

Corollary 3: If a connected planar simple graph has a edges and v vertices with $v \geq 3$ and no circuits of length 3, then $e \leq 2V - 4$

For example: Is $K_{3,3}$ planar?



$K_{3,3}$ has vertices = 6

$K_{3,3}$ has edges = 9

Here, $v \geq 3$ and there is no circuits of length 3, If $K_{3,3}$ were planar, then $e \leq 2v - 4$ would be true.

$2V - 4$

$= 2\times6 - 4$

$= 8$

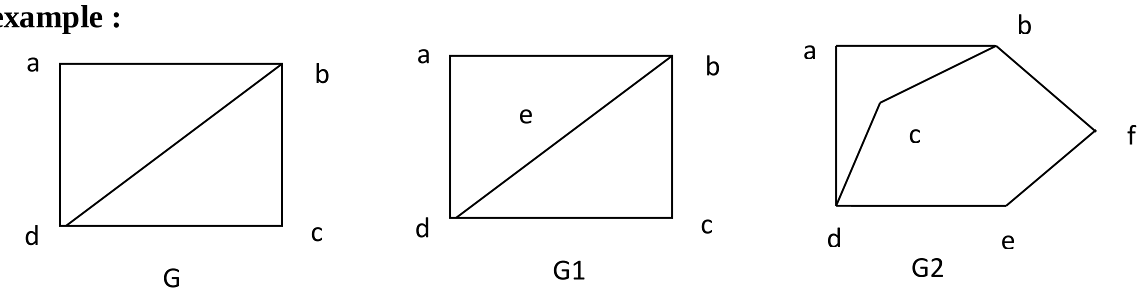
So, e must be ≤ 8

But e = 9

Therefore $K_{3,3}$ is a non-planar.

⇨ If a graph is planar, so will be any graph obtained by removing an edge {u,v} and adding a new vertex w together with edges {u,w} and {w,v}. such an operation is called an elementary sub-division. The graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are called homeomorphic if they can be obtained from the same graph by a sequence of elementary sub division.

For example :

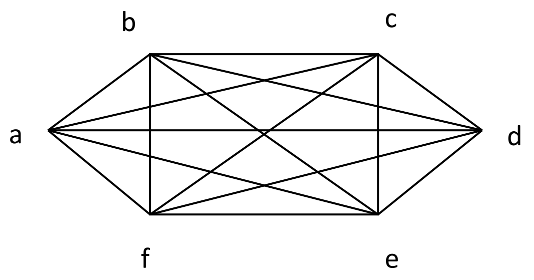


Graphs G_1 and G_2 are homeomorphic to the graph

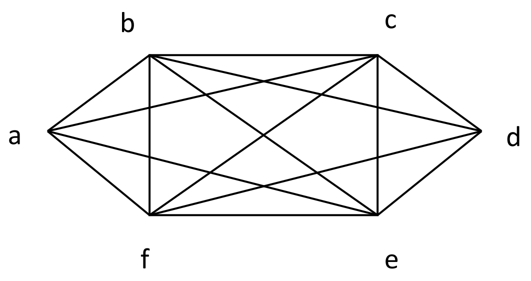
Kuratowski's Theorem (Planarity Testing Algorithm)

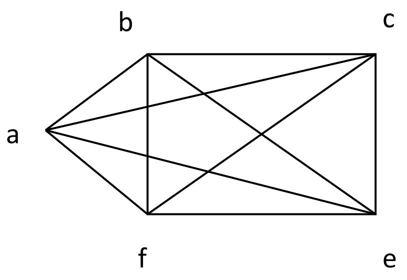
A graph is non- planar if and only if it contains a sub graph homeomorphic to $K_{3,3}$ or K_5.

For example:



Solution





Above graph is homegraphic to K_5, the given graph is not planar.

Weighted Graphs:

Graphs that have a number assigned to each edge (u, v) has a weight (u,v). Each weight is a real number.

Weight can represent distance, cost, time, capacity etc.

The length of a path in a weighted graph is the sum of the weights on the edges.

Dijkstra's Algorithm finds the shortest path between two vertices.

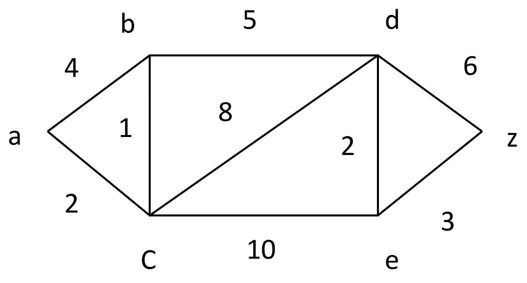
A shortest –path Algorithm

There are several algorithm that find a shortest path between two vertices in a weighted graph.

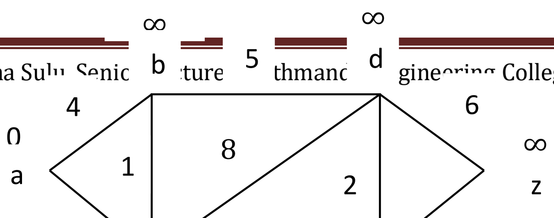
Dijkstra's Algorithm

```
Procedure Dijkstra (G: weighted connected simple graph with all weights positive)

G has vertices $a=v_0, v_1,\ldots\ldots\ldots\ldots, v_n=2$ and weighs $(v_i, v_j)$ is not an edge in G.

For i=1 to x

$L(v_i) = \infty$

$L(a) = 0$

$s :: = \emptyset$

{ the labels are now initialized so that the label of a is 0 and all other labels are $\infty$ and s is an empty set}

While $z \notin s$

Begin

u: a vertex not in s with L(u) minima

s: s {u}

for all vertices u not in S.

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 vertices not in s}

End {L(~~Z)~~ ={ length of  a shorter path from a to z}
```

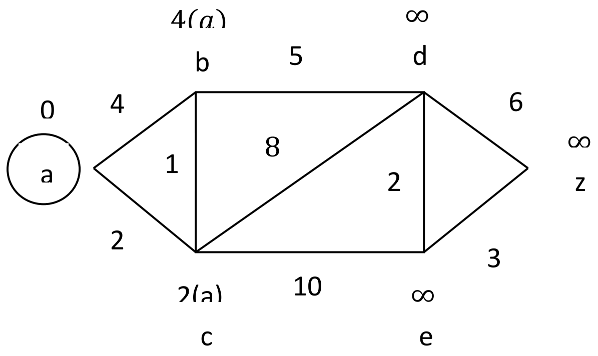
QN.1. Use Dijkstra's algorithm to find the shortest path from the vertices a to z in the following weighted graph given below.



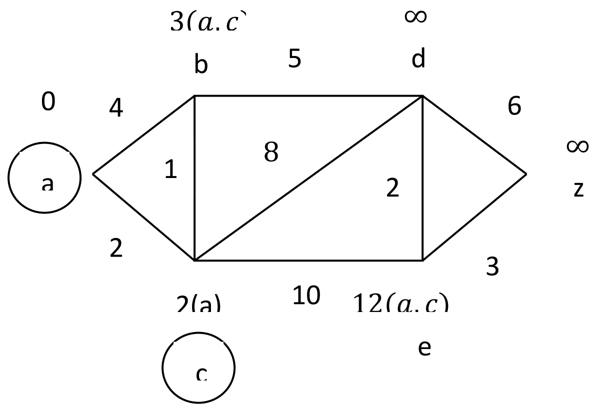
1)



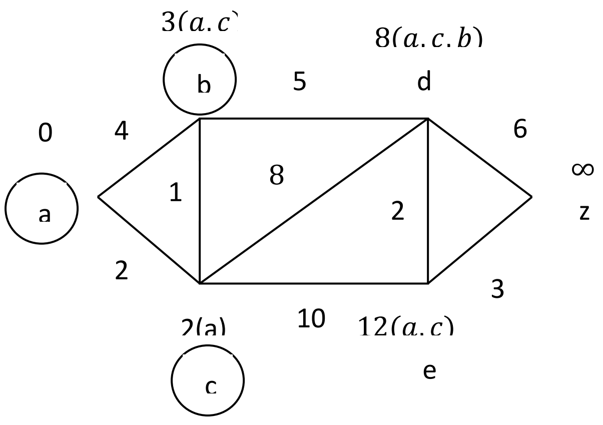
2)


4(a)
∞
b
5
d
0
4
6
a
1
8
2
∞
z
2
3
2(a)
10
∞
c
e

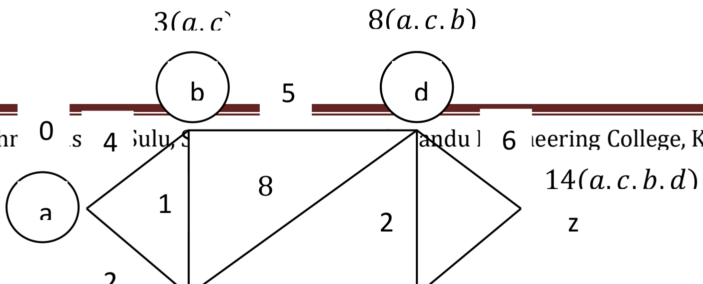
3)


3(a.c)
∞
b
5
d
0
4
6
a
1
8
2
∞
z
2
3
2(a)
10
12(a.c)
c
e

4)

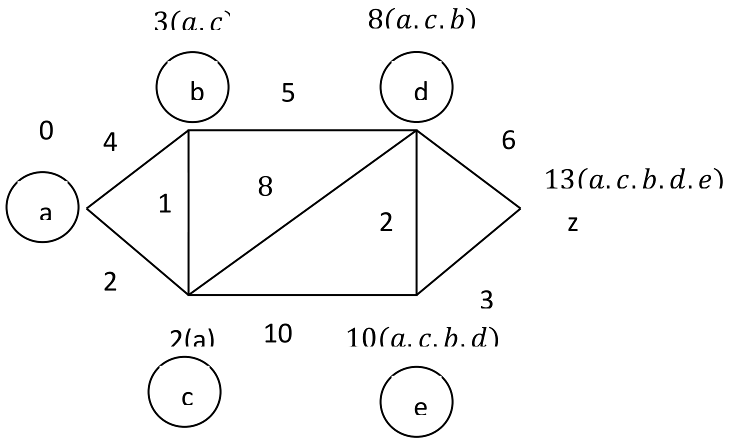

3(a.c)
8(a.c.b)
b
5
d
0
4
6
a
1
8
2
∞
z
2
3
2(a)
10
12(a.c)
c
e

5)

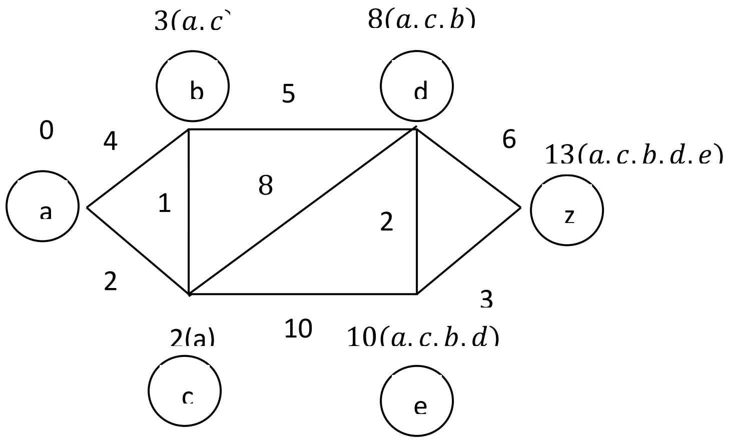

3(a.c)
8(a.c.b)
b
5
d
0
4
6
14(a.c.b.d)
a
1
8
2
z
2

10

6)



7)



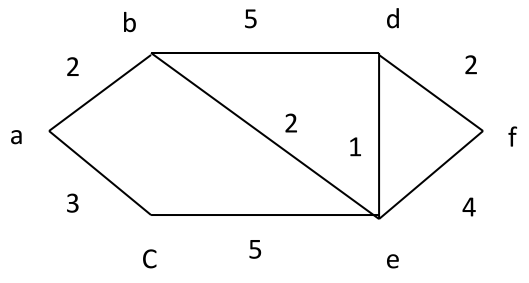
w(u,v)	a	b	c	d	e	z

a	0	4	2	∞	∞	∞
b	4	0	1	5	∞	∞
c	2	1	0	8	10	∞
d	∞	5	8	0	2	6
e	∞	∞	10	2	0	3
2	∞	∞	∞	6	3	0

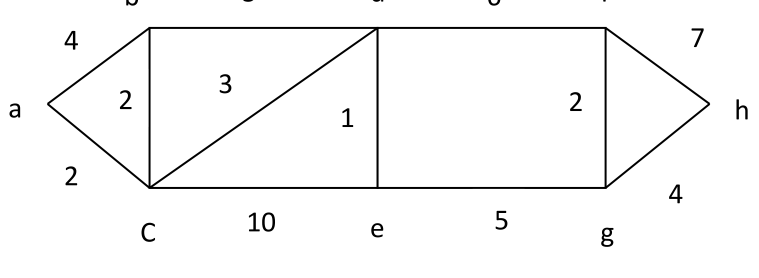
s	ø	{a}	{a,c}	{a,c,b}	{a,c,b,d}	{a,c,b,d,e}	{a,c,b,d,e,z}
L(a)	0	0	0	0	0	0	0
L(b)	∞	4	3	3	3	3	3
L(c)	∞	2	2	2	2	2	2
L(d)	∞	∞	10	8	8	8	8
L(e)	∞	∞	12	12	10	10	10
L(f)	∞	∞	∞	∞	14	13	13

Hence, the shortest path from a to z has length L(z)=13

Q 2)



Q 3)



The travelling salesman problem

The travelling salesman is one of the classical problem in computer science.

A travelling salesman wants to visit a number of cities and then return to his strarring point of course he wants to save time and energy. So he wants to determine the shortest cycle for his trip.

We can represent the cities and the distances between them by a weighted, complete, undirected graph.

The problem then is to find the shortest cycle of minimal total weight that visits each vertex exactly one.

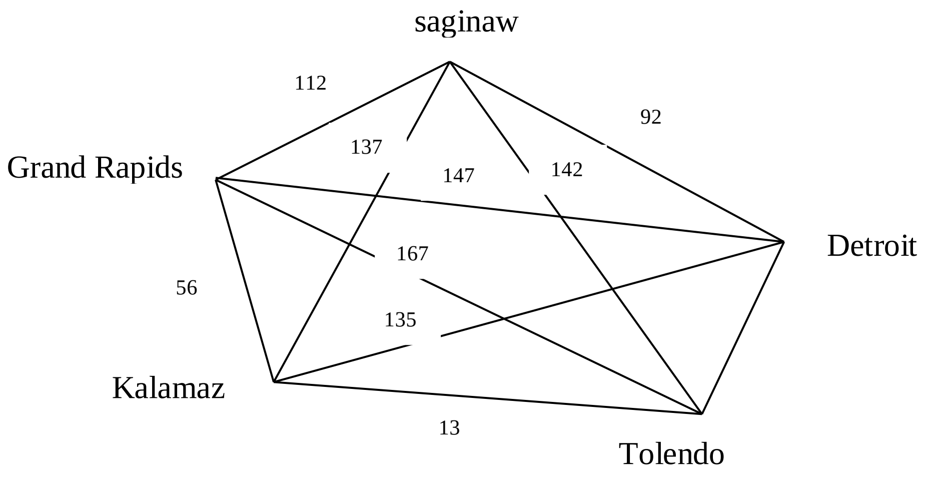
Importance

Variety of scheduling application can be solved as a traveling salesman problem.

Examples:

Ordering drill position on a drill press.

School bus routing



Route

- Detroit, Toledo, Grand- Rapids-saginaw-kala-mazoo- Detroit 610
- Detroit Toledo-Grand Rapids-kalamazoo-saginav-Detroit 546
- Detroit Toledo-Grand Rapids-kalamazoo-saginaw-Grand-Raphids-Detroit 588
- Detroit-Toledo-kalamazoo-Grand Rapids-saginaw-Detroit 458
- Detroit-Toledo-saginaw-kalamazoo-Grand Rapids-Detroit 540
- Detroit-Toledo-saginaw-Grand Rapids-kalamazoo-Detroit- 504
- Detroit-saginaw-Tolede-Grand Rapids-kalamazoo-Detroit-598
- Detroit-saginaw-Toledo-kalamazoo-Grand Rapids-Detroit-576
- Detroit-saginaw-kalamazoo-Toledo-Grand Rapids-Detroit-682
- Detroit-saginaw-Grand Rapids-Toledo-kalamazoo-Detroit-646

- Detroit-GrandRapids-saginaw-Toledo-kalamazoo-Detroit-670
- Detroit-GrandRapids-Toledo-saginaw-kalamazoo-Detroit- 728

Tree

A tree is a connected undirected simple graph with no simple circuit. A tree is a particular type of graph.

Properties

1. There is a unique simple path between any 2 of its vertices.
2. No loops
3. No multiple edges

For example:

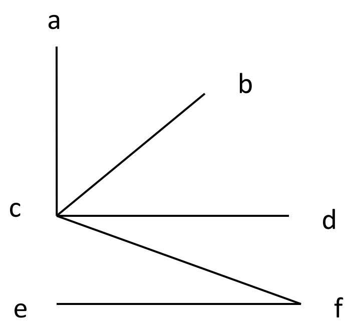


Fig: G1

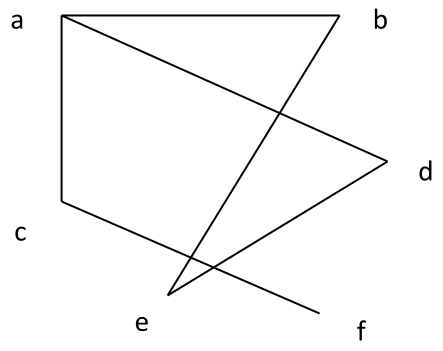


Fig: G2

G1: It is tree because it is connected graph with no simple circuit.

G2: It is not a free because there is a cycle a, b, e, d, a

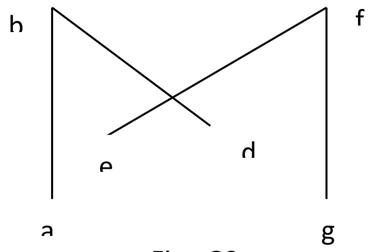


Fig : G3

An undirected graph having no simple circuit and is not connected is called forest. The forest has each of its connected components an tree G3 displays a forest.

Theorem1:

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

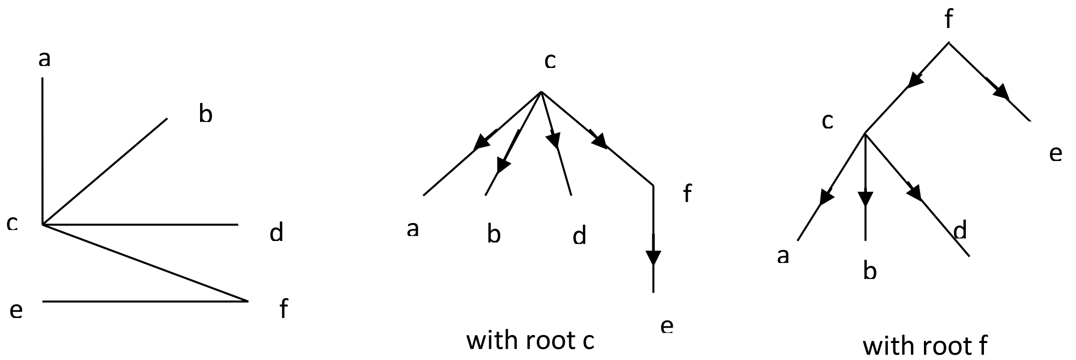
Proof:

Assume that T is a tree. Since T is a tree. It is a connected simple graph with no simple circuits. Let x and y be two vertices of T. We know that every connected graph has a simple path between every pair of vertices. So, there is a simple path from x to y. This path must be unique because, if the path between x and y is not unique then there is another path between x and y that uses edges different from the path, then reversing the path i.e, going to from x to y from first path and going from y to x through second path forms a circuit. This is a contradiction that is a tree. Hence, there is a unique simple path between any two vertices of a tree.

Rooted (Directed) Tree

In many applications of trees, a particular vertex of a tree is designated as the root. 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.

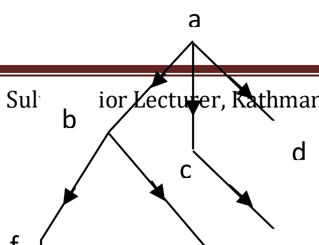
We can change an unrooted true into a rooted by choosing any vertex as the root. The tree in which root is defined produces a directed graph.



with root c with root f

- Root :Vertex with in degree O.
- In a rooted tree, if V is the vertex in T other than root, then the **Parent** of V is a vertex of u in T such that there is directed edge from u to v.
- In this scenario v is called **child** of u.
- Vertices with same parents are called **Siblings**
- All the vertices that appear in the path from root to some vertex v in T, including root are called **ancestors** of V.
- The **descendents** of a vertex V are those vertices that have V as their ancestor.
- All the vertices that have children are called internal vertices (root is also an internal vertex if the tree has more than one vertices.

For example:



Parent : b is parent of f and g.

Child : g and f are children of b

Siblings: f and g are siblings.

Ancestors : Ancestors of g are b, a.

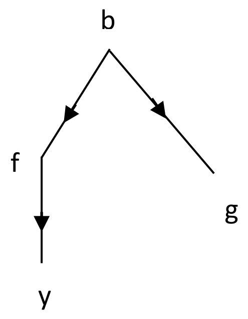
Descendents : Descendents of b : f, g, y

Leaf : Vertices with no children (y, g, e, d)

Internal vertices : a, b, c, f

Sub tree : Subgraphs consisting of V and its descentants and their incident edges

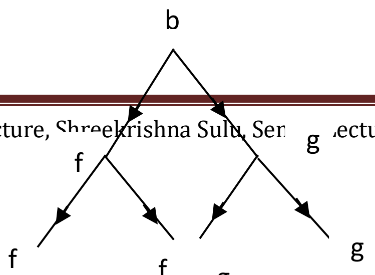
subtree rooted at b:

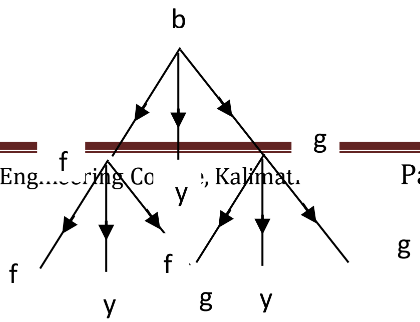


- Level (of v) is length of unique path from root to v (level of root = 0, level of b =1, level of g=2)
- Height is the maximum of vertices level (height : 3)

m-ary trees

- A rooted tree is called m-ary if every internal vertex has no more than m children.
- It is full m-ary if every internal vertex has exactly m children.
- A 2-ary tree is called a binary tree.





Full binary tree Full 3-ary tree

Ordered Rooted Tree

- An rooted tree where the children of each interval node are ordered.
- In ordered binary tree, we can define :
  - left child, right child
  - left subtree, right subtree
- For m-ary trees with m>2, we can use terms like "left most", "right most" etc.

Properties of trees:

Theorems 1 : A tree with n vertices has n-1 edges.

Proof :

Basis step: When n=1, a tree with n=1 vertex has no edge. It follows that theorem is true for n=1.

Inductive hypothesis :_Assume that the tree with K vertices has K-1 edges, where k is positive integer.

Inductive steps: Suppose that a tree T has K+1 vertices and that V is a leaf of T. Removing vertex V and the associated edge from T produces a tree T1 with K vertices, since the resulting graph is still connected and has no simple circuit. By the induction hypothesis, T1 has k-1 edges. Hence T has K edges since it has one more edge than the edge connecting V to its parent.

Theorem 2 : A full m-ary tree with i-internal vertices contains n=mi +1 vertices.

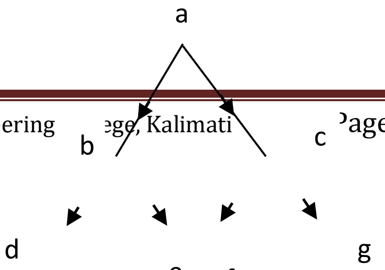
A level(depth) of a vertex v in a rooted tree is the length of the unique path form the root to vertex. The level of the root is zero. The height of the rooted tree is the length of the longest path from the root to any vertex. A rooted m-ary tree of height h is balanced if all leaves are at levels h or h-1.

For example:

Vertices = 13

Internal vertices (i) = 6

Leaves L = 7



13 = 6×2+1

3) A full m-ary tree with

i) n vertices has I=(n-1)/m internal vertices and

L = ((m-1) * n+1)/m leaves

ii) I internal vertices has n=m*I + 1 vertices

and L=(m-1)*I + 1 leaves

iii) L leaves has n =(m*1-1)/(m-1) vertices and

I= (L-1)/(m-1) internal vertices.

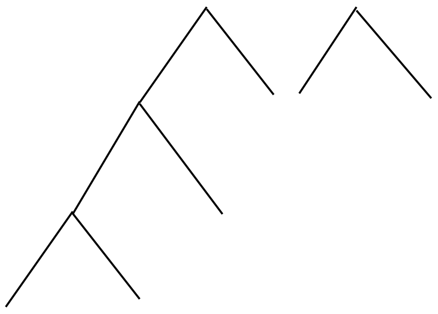


Fig: I

For example:

m=2, n=13, I=6, L=7 from fig (I)

i. I=(n-1)/m = (13-1)/2 = 6

and L =((m-1)*n+1)/m = ((2-1)*13 + 1)/2 = 7

ii. n= m*I + 1 = 2*6+1=13

and L =(m-1) *1 = (2-1)*6+1=7

iii. n=(m*L-1)/(m-1)=(2*7-1)/(2-1)=13

and I = (L-1)/(m-1) = (7-1)/(2-1)=6

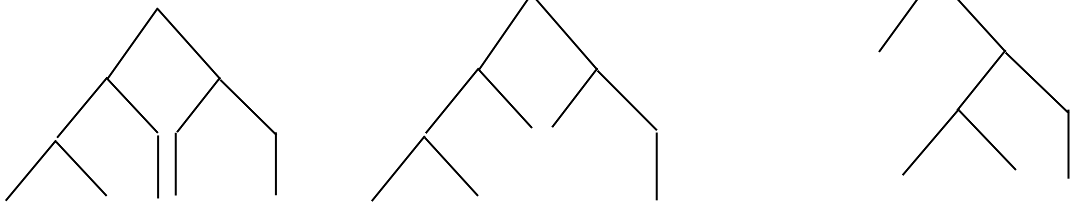
The level of a vertex in a rooted tree is the length of the path from the root to the vertex (level of the root is O)

The height of the rooted tree is the maximum of the levels of vertices (length of the longest path from the root to any vertex).

Balanced Tree:

A rooted m-ary tree of height h is balanced if all leaves are at levels h or h-1

For example



Applications of trees:

- How should items in a list be stored so that an item can be easily located? For this, we use the concept of binary search trees.
- What series of decisions should be made to find an object with a certain property in a collection of objects of a certain type? For this, we use the concepts of decision trees.
- How should a set of characters be efficiency coded by bit. Strings? Here, we use the concept of the prefix codes.

Binary Search Trees:

Searching for items in a list is one of the most important task that a rises in computer science. Our primary goal is to implement a searching algorithm that finds items efficiently when the items are totally ordered. This can be accomplished through the used binary search tree.

A binary search tree (BST) is a binary tree in which each child of a vertex is designated as a right or left child, no vertex has more than one right child or left child and each vertex is labeled with key, which is one of the items. Further more vertices are assigned keys so that the key of a vertex is both larger than the keys of all vertices in its left sub tree and smaller than keys of all vertices in its right subtree.

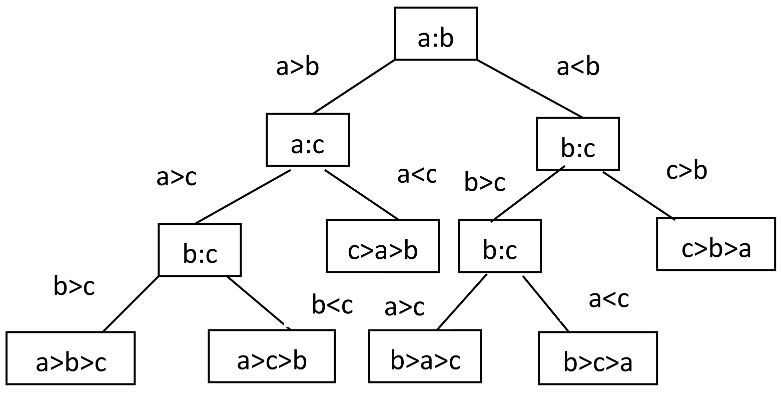
We use recursive procedure to from a binary search tree for a list of items. We start with a tree containing just one vertex, namely, the root. The first item in the list is assigned as the key of the root. To add a new item, first compare it with the keys of vertices already in the tree, starting at the root and moving to the left if the item is less than the key of respective vertex if this vertex has a left child or moving to the right if the item is greater than the key of the respective vertex if this vertex has a right child. When the item is less than respective vertex and this vertex has no left child, then a new vertex with this item as its key is inserted as a new left child. Similarly, when the item is greater than the respective vertex and this vertex has no right child, then a new vertex with this item as its key is inserted as a new right child.

For example:

Form a BST for the words mathematics, physics, geography, zoology, metrology, geology, psychology and chemistry.

Decision Trees:

Rooted trees can be used to model problems in which a series of decision leads to a solution. For instance, a binary search tree can be used to locate items based on a series of comparisions, where each comparison tell us whether we have located the item or whether we should go left or right in a substance. A rooted tree in which each internal vertex corresponds to a decision with a subtree at these vertices for each possible outcome of these vertices for each possible outcome of the decision is called a decision tree. The possible solution of the problem correspond to the paths to the leaves of the rooted tree.



A decision tree for sorting three distinct elements.

Prefix codes:

Consider a problem using bit string to encode the letters of the English alphabet. We can represent each letter with a bit string of length five as there are only 26 letters and there are 32 bit strings of length five.

Consider using bit strings of different length to encode letters. Letters that occur more frequently should be encoded using short bit strings. When letters are encoded using varying number of bits, some method must be used to determine where the bits of a each character start and end.

For instance, if we were encoded e with o, a with 1 and t with 01, then the bit string 0110 could represent to eat or tae or bit string 0101 could correspond to eat, tea, eaea or tt.

One way to ensure that no bit string corresponds to more than one sequence of letters is to encode letters so that the bit string for a letter never occurs as the first part of the bit string for another letter codes with this property are called prefix codes.

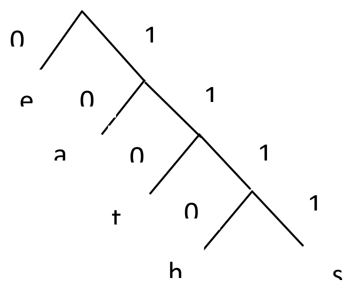
For example:

Encoding of e as o, a as 10 and t as 11 is a prefix code. A word can be recovered from unique bit string that encodes its letters.

For example: 10110 is the encoding of ate.

A prefix code can be represented using binary tree, where the characters are the labels of the leaves of tree. The edge of tree are labeled so that an edge leading to a left child is assigned a o and an edge 1.

We construct prefix code from any binary tree where the left edge at each internal vertex is labeled by 0 and right edge by 1 and leaves are labeled by characters.



The tree in the figure represents encoding of e by o., a by 10 t by 110, m by 1110 and s by 1111.

For example: 11111011100

s a n e

Huffman coding

Here is an algorithm that takes input the frequencies of symbols in a string and produces as output a prefix code that encodes the string using fewest possible bits. This algorithm is called Huftman coding. It is used in data compression. The subject devoted to reducing no. of bits required to represent information.

Procedure

F: Forest of n rooted trees.

While F is no a tree.

begin:

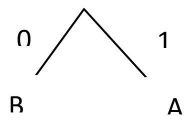
Replace the rooted trees T and T' of least weight from F with W(T)$\geq$ W(T') with a tree having a new root that has T as its left subtree and T' as its right subtree. Label the new edge to T with O and new edge to T' with 1.

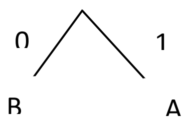
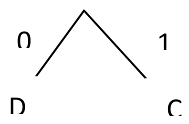
Again W(T) + W(T') as weight of new tree end.

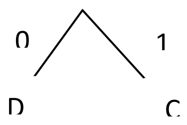
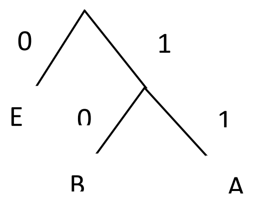
Use Huffman coding to encode following symbols with the frequencies listed what is the average no. of bits used to encode a character.

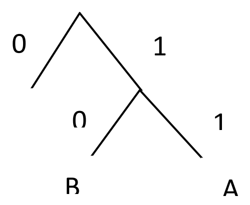
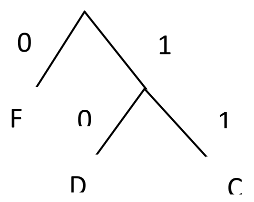
A=0.08, B=0.10, C=0.12, D=0.15, E=0.20, F=0.35

0.08	0.10	0.12	0.15	0.20	0.35
A	B	C	D	E	F initial Forest

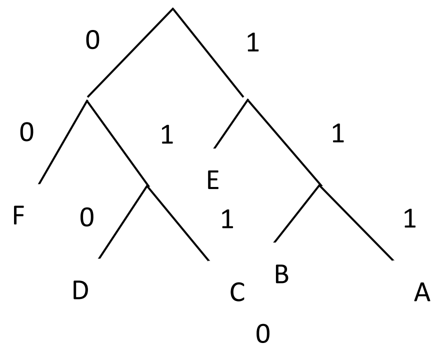
0.12	0.15	0.18	0.20	0.35	
C	D		E	F	Step 1

0.18	0.20	0.27	0.35	
	E		F	Step 2

0.27	0.35	0.38	
	F		Step 3

0.38	0.62	
		Step 4

1



From the graph, we can encode A by 111, B by 110, C by 011, D by 010, E by 10 , and F by 00.

The average number of bits used to encode a symbol using this encoding is

$P_k l_k$ (P_k : Probability, l_k : length)

$=0.08\times3 + 0.10\times3+0.12\times3 + 0.15\times3+0.2\times2+0.35\times2$

$=2.45$

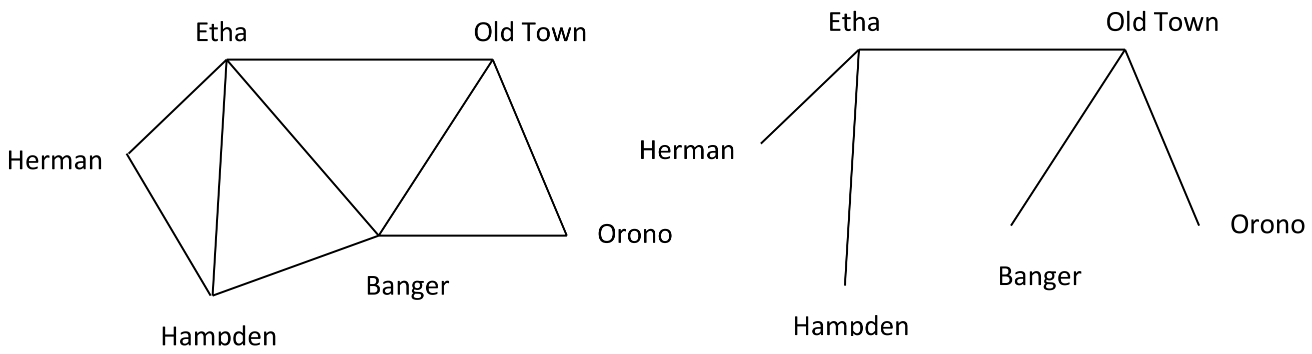
Spanning Trees:

Let G be a simple graph. A spanning tree is a sub graph of G that is a tree containing every vertex of G.

Consider a system of roads in Maine. The only way the roads can be kept open in the winter is by frequently plowing them. The highway department wants to plow the fewest roads so that there will always be cleared roads connecting any two towns. At least 5 roads must be plowed to ensure that there is a path between any two towns. Fig (b) represents one such set of roads. Note that the subgraph representing these roads in a tree because it is connected and contains six vertices and five edges.

This problem was solved with a connected subgraph with the minimum number of edges containing all vertices of the original simple graph such a graph must be a tree.

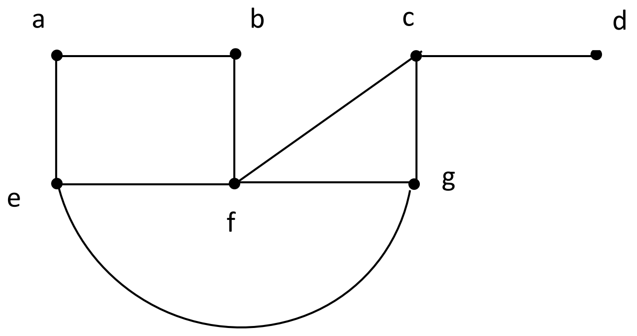
Example: 1

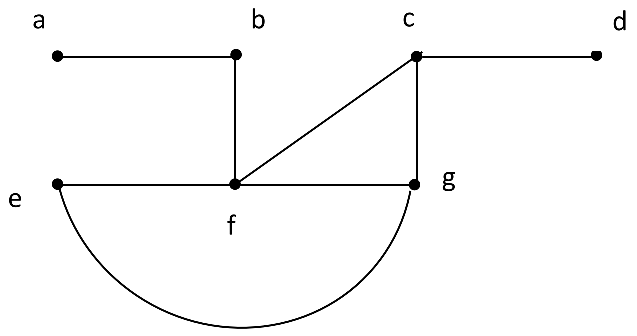


(a) (b)

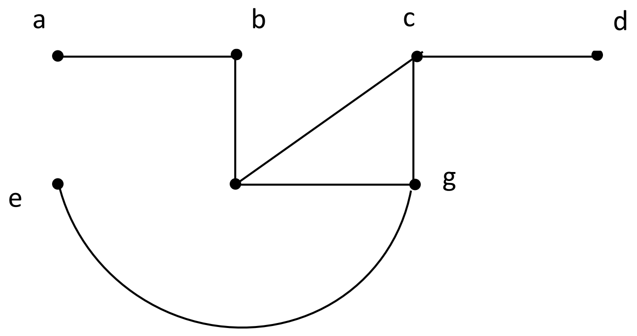
Fig (a) : Road system and (b) sets of roads to plow

Example: 2

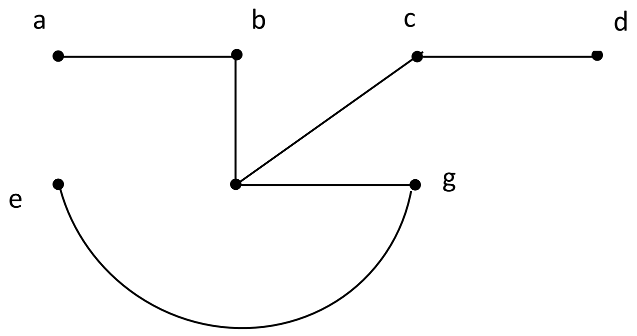




{a, e}



{e, f}



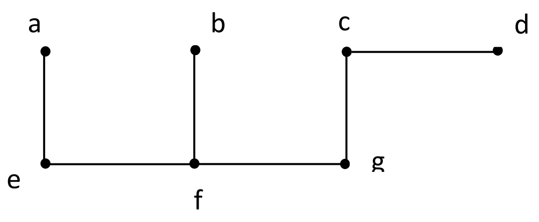
{c, g}

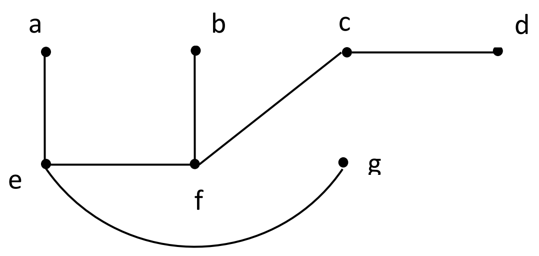
Fig : Producing spanning tree for G by removing edges that form simple circuits.

Theorem 1: A simple graph is connected if and only if it has a spanning tree.

Proof:

Firstly let a simple graph G has a spanning Tyree T. T contains every vertex of G. Therefore, there is a path T between any two of its vertices. Because T is a subgraph of G, there is a path in G between any two of its vertices. Hence, G is connected. Now suppose G is connected, If G is not a free, it must contain a simple circuit. Remove an edge from one of those simple circuits. The resulting graph has one fewer edge but still contains all the vertices of G is connected.





spanning trees are important in data networking such as IP multicasting.

The proof by theorem 1 gives an algorithm to finding spanning trees by removing edges from simple circuits. This algorithm is insufficient because it requires that simple circuits be identified.

We can build spanning tree for a connected simple graph using depth first search.

For this,

Firstly, We will arbitrarily choose a vertex of the graph as a root.

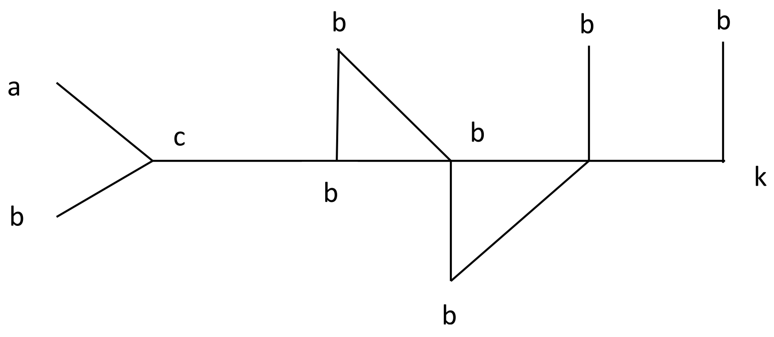
Form a path starting at this vertex by successively adding vertices and edges, where each new edges is incident with last vertex in path and a vertex not already in the path.

If path goes through all vertices of graph, the tree consisting of this path is spanning tree.

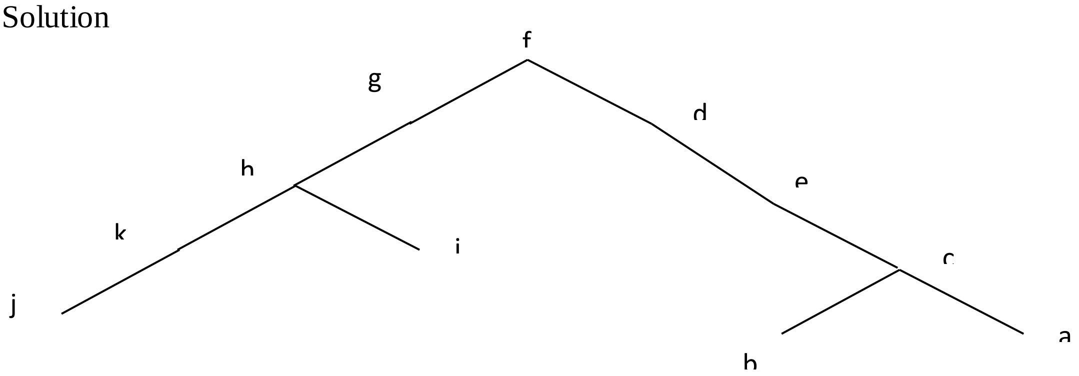
If path does not go through all vertices, more vertices and edges must be added.

Move back to the next to last vertex in the path and if possible, form a new path starting all this vertex passing through vertices that were not already visited. If this can not be done, move back another vertex in the path, that is two vertices back in the path and try again.

Example:

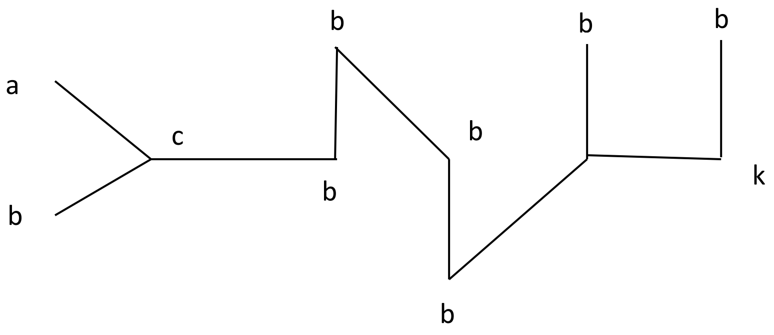


Solution



The edges selected by depth first search of a graph are called free edges.

All other edges of the graph must connect a vertex to an ancestor or descendant of this vertex in the tree. These edges are called back edges.



Depth first search is also called back tracking Algorithm

Procedure DFS (G: connected graph with vertices V_1, V_2, ………..V_n)

T: tree consisting only of the vertex V_1 visit (V_1)

Procedure visit (Vertex of G)

for each vertex w adjacent to and not yet in T.

begin,

add vertex w and edge {v, w} to T visit (w)

end.

Depth first search can be used to find paths and circuits in a graph, can be used to determine the connected components of a graph and to find cut vertices of a connected graph.

Breadth-First Search:

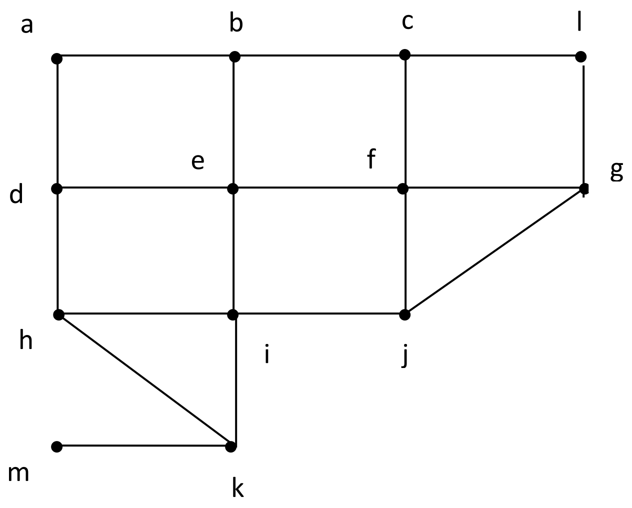
Choose a root to the vertex from the vertices of the graph.

Then, add all edges incident to this vertex. The new vertices added at this stage become the vertices at level 1 in a spanning tree.

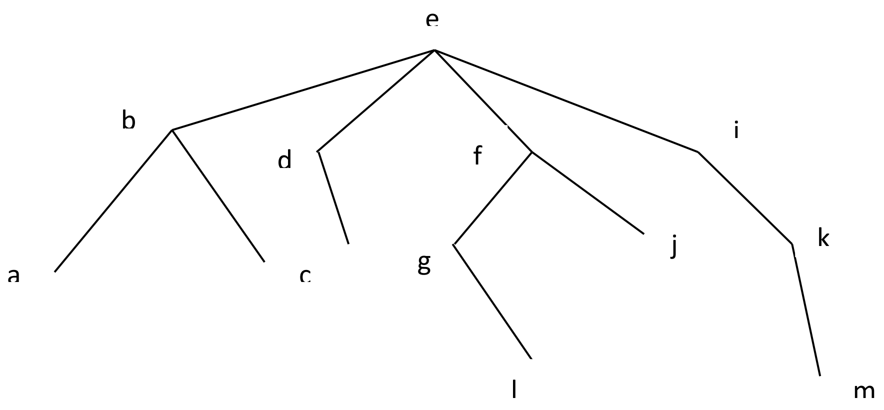
Arbitrarily order them.

For each vertex at level 1, visited in order, add each edge incident to this vertex to the tree as long as it doesnot produce a simple circuits.

Example:

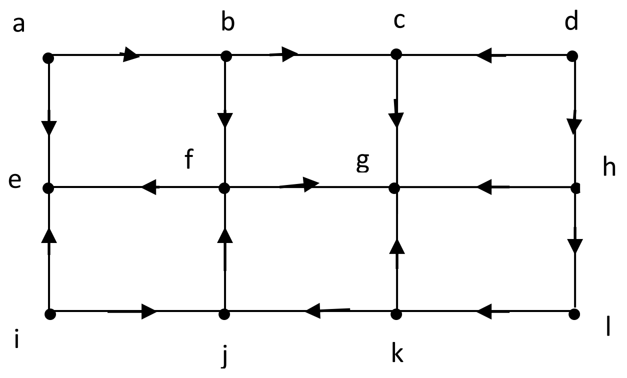


Solution

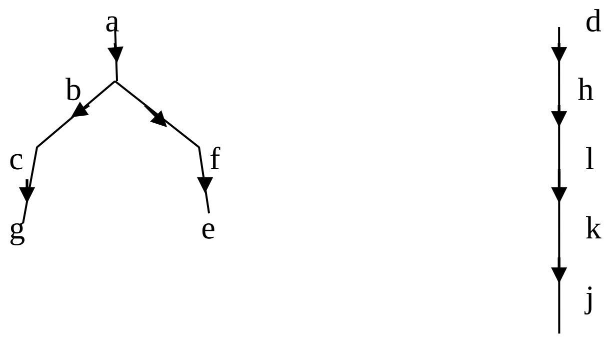


Depth first search in Directed Graphs

In directed graphs, there can be spanning forest.



Solution:



Minimum spanning trees:

A minimum spanning tree in a connected weighted graph is a spanning tree that has the smallest possible sum of weights of its edges.

We will use 2 algorithms for constructing minimum spanning trees.

Prim's Algorithm

The algorithm was given by Robert Prim in 1957.

To carry Prism's algorithm

Begin by choosing any edge with smallest weight, putting it into the spanning tree.

Successively add to the tree edges of minimum weight that are incident to a vertex already in the tree and not forming a simple circuit with those edges already in the tree. Stop when n-1 edges have been added.

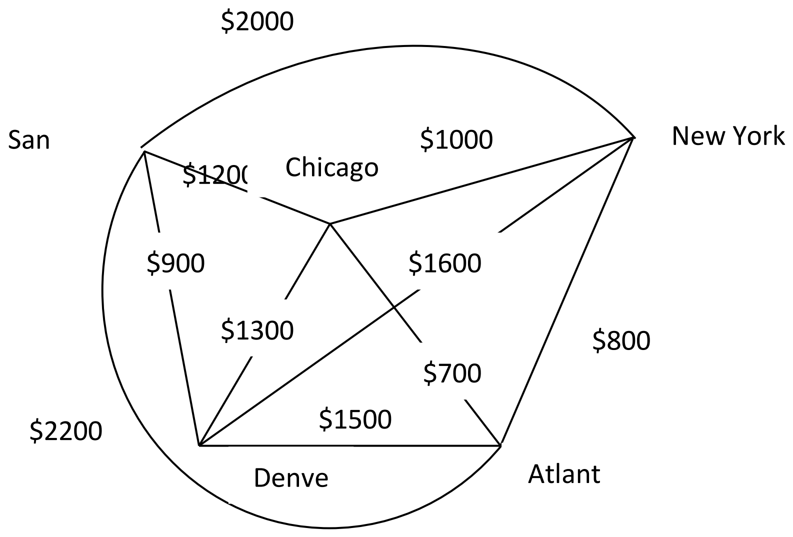
{Chicago, Atlanta}= $700

{Atlanta, New York}= $800

{Chicago, San Fransico}= $1200

{San Fransico, Denver} = $900

Total $3600



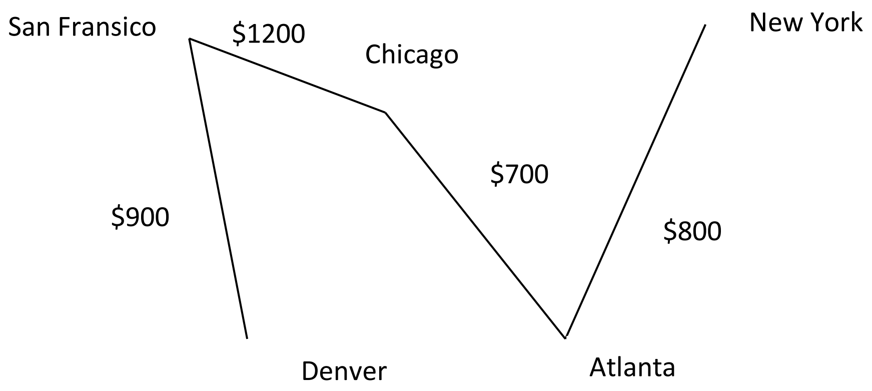


Fig : Minimum Spanning tree

Example

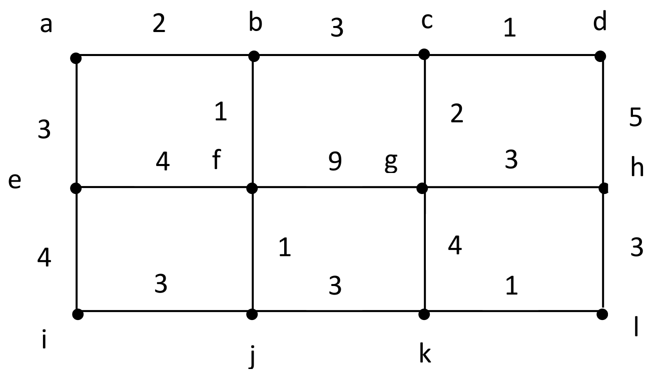


Fig : II

{b, f}= 1

{f, i}=2

{a, b} =2

{a, e}=3

{i,j}=3

{f,g}=3

{c, g}=2

{c, d}=1

{g, h}=3

{h, l}=3

{k, l}=1

$\overline{}$

24

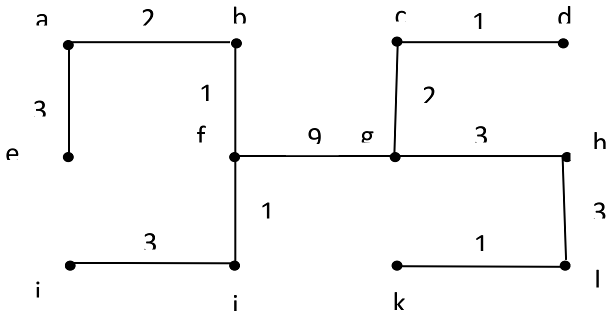


Fig : minimum spanning tree

Krushal's Algorithm

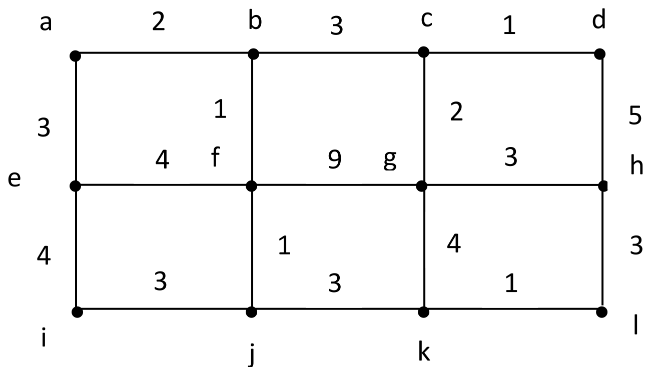
This algorithm is given by Jospeh Krushal

For this algorithm,

Firstly choose an edge in the graph with minimum weight.

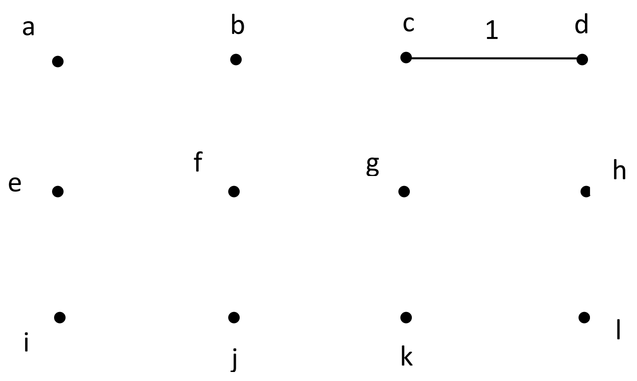
Successively add edge with minimum weight that do not form a simple circuit with those edges already choosen. Stop after n-1 edges have been selected.

For Fig II v



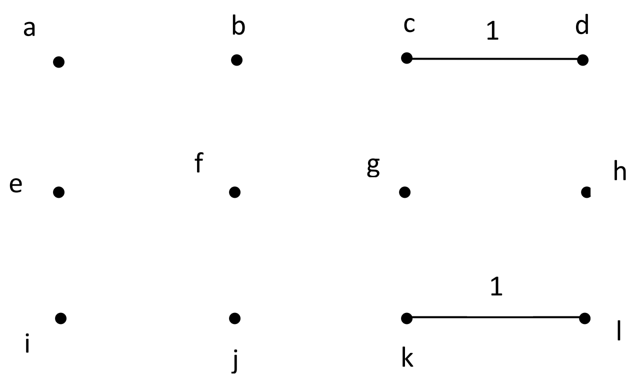
Solution

(a)



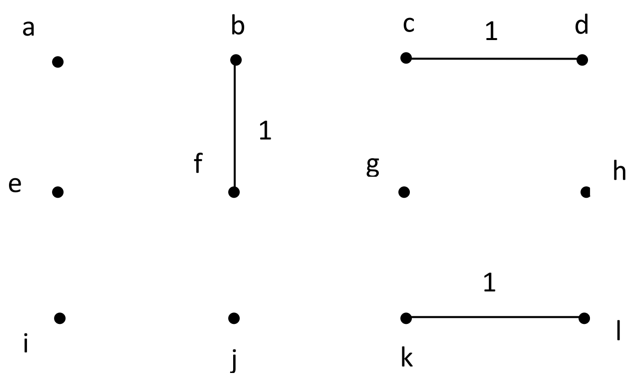
{c, d}= 1

(b)



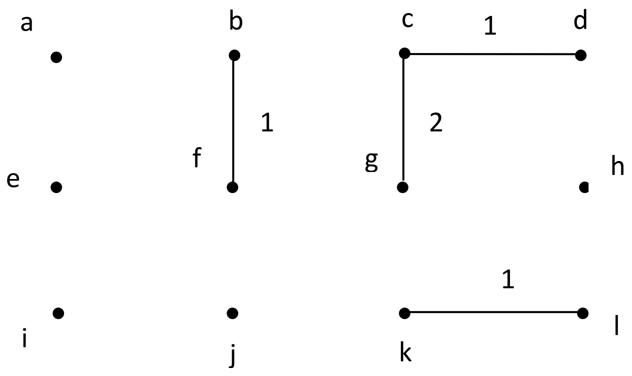
{k,l}=1

(c)



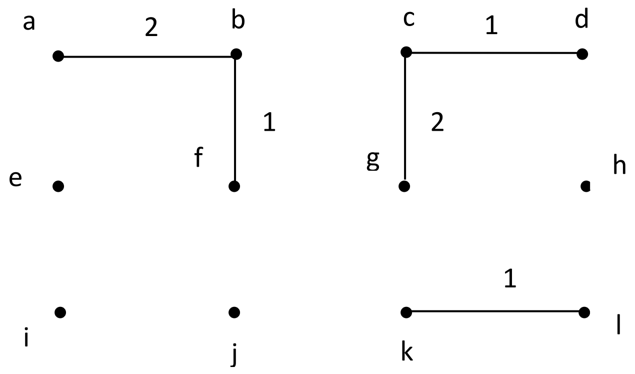
{b,f}=1

(d)



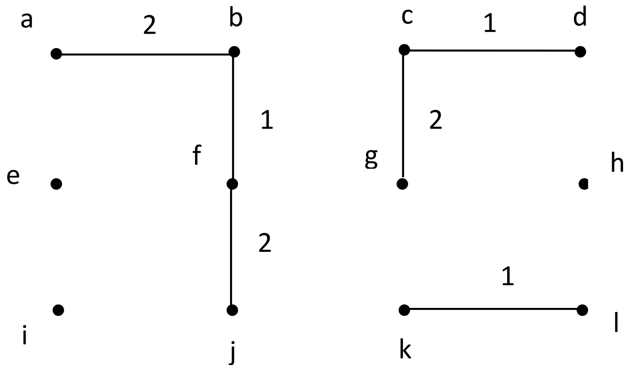
{c, g}=2

(e)



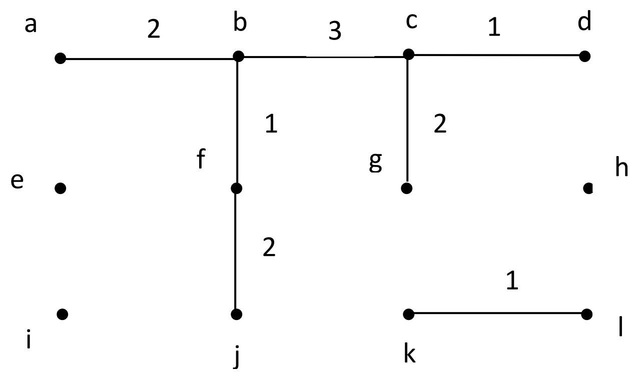
{a,b}=2

(f)



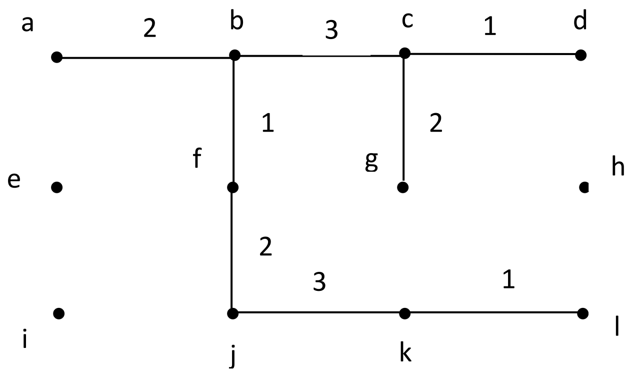
{f, j}=2

(f)



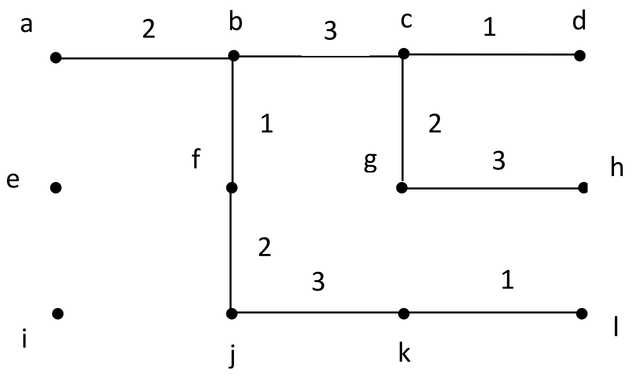
{b,c}=3

(g)



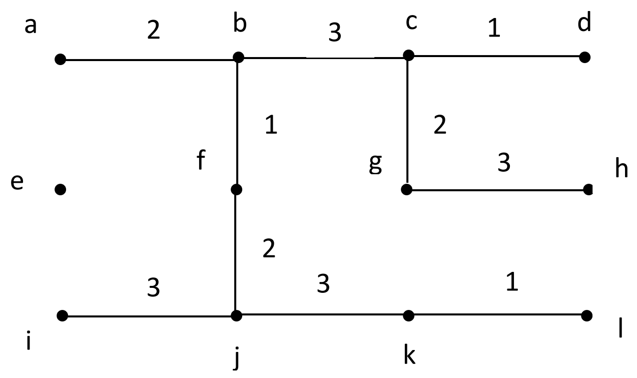
{j, k}=3

(h)



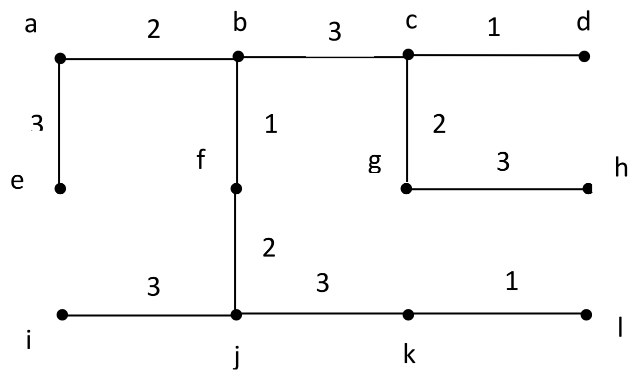
{g, h}=3

(i)



{i,j}=3

(j)

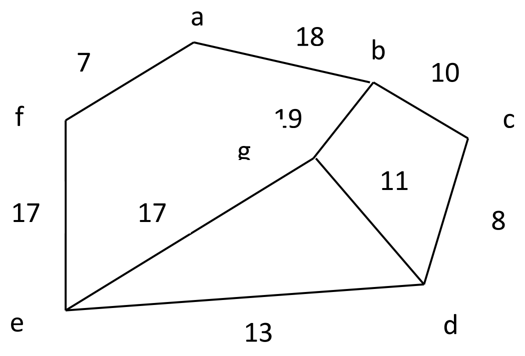


{a,e}=3

Total =24

minimum spanning tree

Q. Graph given below, find the minimum spanning tree



Graph coloring

A coloring of a simple graph is the assignment of a color to each vertex of the graph so that no two adjacent vertices are assigned the same color.

Smallest number of colors must be used instead of just assigning every region its own color.

A chromatic number of a graph is the least no. of colour needed for coloring graph.

4 color Map Theorem:

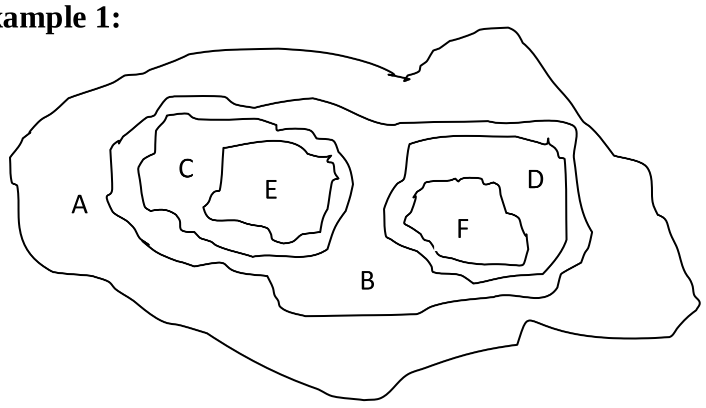
The chromatic number of a plannar graph is no greater than four.

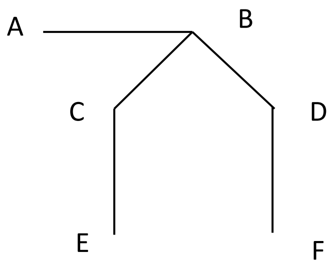
It can be shown that any two dimensional map can be painted using four color in such a way that adjacent regions are different colors.

Dual graph

Each map in a plane can be represented by a graph .Each region is represented by a vertex. Edges connect to vertices if the regions represented by these vertices have a common border.Two regions that touch at only one point are not considered adjacent.

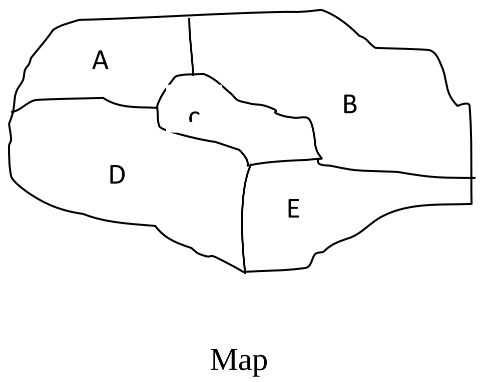
Example 1:

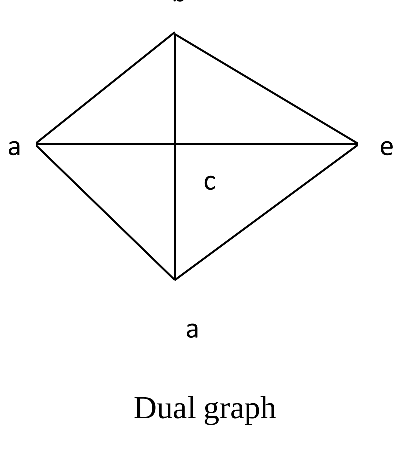




Dual graph

Example 2:

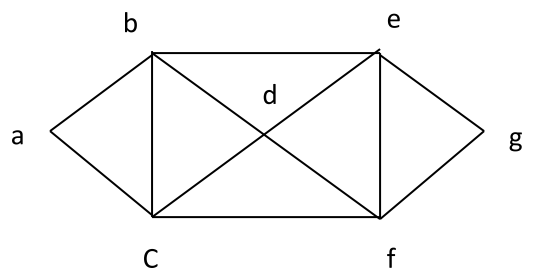




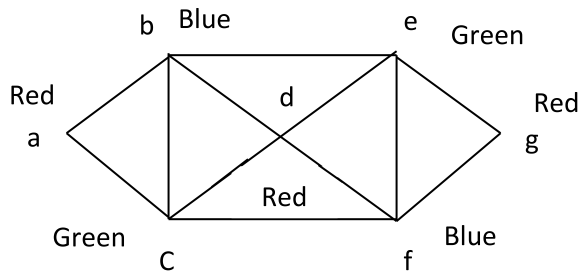
Map

Dual graph

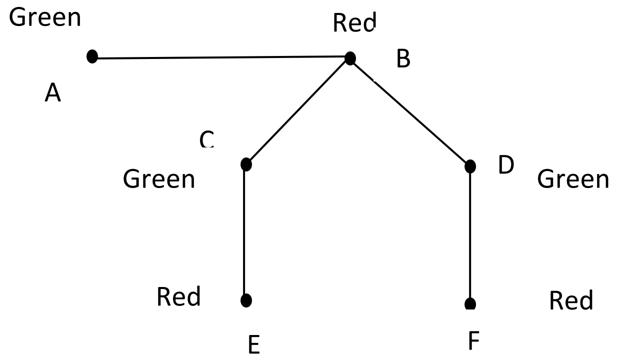
Q1. What is the chromatic no. of graph.



Chromatic no. must be at least 3 since a, b, c must be assigned different colors

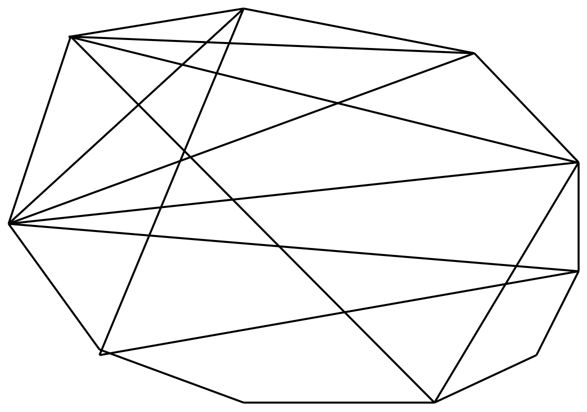


QN.2 **What is the chromatic no. of the fig(A)**



We can color the graph with at most 2 colors as shown in the graph red and Green.

Q.N.3. Find the chromatic no. of a graph.



Max flow and Min cut

- In network or transport network the flow is the amount of commodity (number of car in a road, gallon of oil in pipe, bits of formation in channel etc.) transported from one place to another for an instant of time.
- The maximum amount of-low is called the **capacity** of the line arc or edge.
- The flow f(e)=f(v,w) and the capacity c(e)=c(v,w) are non-negative real numbers.
- $f(e)\leq c(e)$.
- The unique starting vertex that has in degree=0 is called **source** $s\in V$
- The unique ending vertex rat has out degree=0 is called **sink**, $t\in V$
- In a network there is one source and one sink. Any other vertex : ca.ied intermediate vertex.
- **Flow conservation :** For any intermediate vertex, total flow into x equals to the flow out of x .
  $\sum_{w\in V} f(w,x)=\sum_{v\in V} f(x,v)$

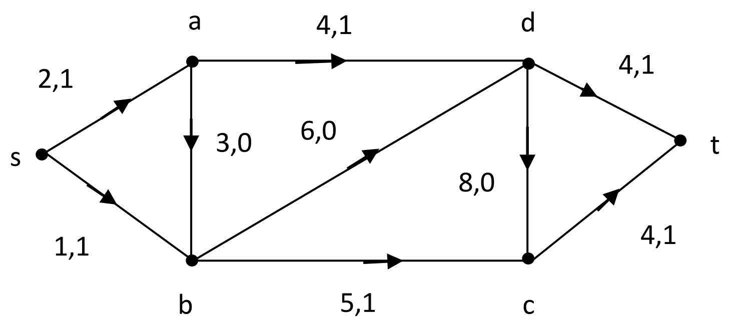
- Sum of flow from the source = Sum of flow to the sink.
- **Saturated flow:** The flow along an edge e(v, w) is said to be saturated if f(e)=c(e).
- If f(e)<c(e), then the flow is unsaturated, then residual capacity or slack is s(e)=c(e)-f(e).

<u>Flow augmenting path</u>

- The edge e_i is called forward edge if: is directed to x_j to x_{i+1} and transportation flow from x_i to $x_{,+1}$, it is backward edge if transportation flow from x_{i+1} to x_i .
- If a flow f is given by path $p:s=x_{1,}x_2 \ldots x_{k-1,}x_k=t$ is called a flow augmenting path (i)every forward edge of path has excess capacity that is f(e)<c(e). (ii) every backward edge has f(e)>0.

Max flow

- The flow that achieves the largest possible value is called the maximum flow or maximal flow in a network.
- To calculate max flow calculate the unused capacity c(e)-f(e) for each forward arc and f(e) in each backward arc.
- In a f- augmenting path from source to sink, for the forward edge increase the flow with the minimum slack(unused capacity) followed in the path.
- For backward edge in the f- augmenting path from source to sink, decrease the flow with the minimum slack.
  Δ =min(c(e)-f(e))
  f*=f+Δ (for forward edge)
  f*=f-Δ (for backward edge)



<u>Max flow</u>

Here the unused capacity=slack in f- augmenting path s, a, d, t are

S(s,a)=2-l=l
S(a,d)=4-l=l
S(d,t)=4-l=3
The minimum value is 1 in above slack value. So, now add the value 1 in the flow of s, a, d, t.
f(s,a)=l+l=2
f(a,d)=l+l=2
f(d,t)=l+l=2

Now s(s,a)=2-2=0. So, there is not necessity for further calculation of slack f-augmenting path containing s(s,a) edge.(remember for f-augmenting path (c(e)-f(e))>0).

- Again, in s(s,b)=1-1=0, So, there is no necessity for further calculation of slack in the augmented path containing edge (s,b)
- The max flow = flow from source s=2+1=3

Min cut

- In min cut we calculate the capacity by separating the vertices into two subsets p and p'
- p contains s only or s and other vertices which are not in p'.

- p' contains t only or t and other vertices which are not in p.
- If there are 4 vertices other than source and sink, then there are 2^4=16 s-t cuts.
- The capacity of a cut denoted by c(p, p') is defined to be the s-m of thecapacities of those edges directed from the vertices in set p to the vertices in p'.
- If the edge is backward edge put the value of the backward edge=0.
- A cut is called minimum cut if its capacity not exceed the capacity of any other cut of the network.

In above network,
If p={s} and p'={a,b,c,d,t} then c(p,p')=c(s,a)+c(s,b)=2+l=3.
If p={s,a} and p'={b,c,d,t} then c(p,p')=c(s,b)+c(a,b)+c(a,d)=l+3+4=8.
Similarly,
Table for possible s-t cut

P	P'	Capacity c(p,p')
{s}	{a,b,c„d,t}	3
{s,a}	{b,c,d,t}	8
{s,b}	{a,c,d,t}	13
{s,c}	{a,b,d,t}	7
{s,d}	{a,b,c,t}	15
{s,a,b}	{c,d,t}	15
{s,a,c}	{b,d,t}	20
{s„a,d}	{b/C,t}	16
{s,b,c}	{a,d,t}	12
{s,b,d}	{a,c,t}	19
{s,c,d}	{a,b,t}	11
{s,a,b,c}	{d/t}	10
{s,a,b,d}	{c,t}	17
{s,a,c,d}	{b,t}	12
{s,b,c,d}	{a,t}	10
{s,a,b,c,d}	{t}	8

Min cut =3

Hence, in the above example max flow=min cut=3.