GRAPH THEORY

Prepared by

Dr. Raju Dutta
Assistant Professor
Department of Mathematics
Sushil Kar College
Champahati
South 24 Parganas

Subject Name: DSE-A
Subject Code: MTMG DSE-A
Total No. of Lectures: 60
Contact Hours: 60
Credit:5+1
Prepared by Dr. Raju Dutta, Assistant Professor, Department of Mathematics, Sushil Kar College, Champahati, South 24 Parganas.

Module No	Number of Lecture required	Topic	Application	Reference Book
Semester V/VI	10	Graphs, Digraphs, Weighted Graphs	They include, study of molecules, construction of bonds in chemistry and the study of atoms. Graph theoretical concepts are widely used in Operations Research. It is also used in modeling transport networks, activity networks and theory of games.	N.Deo, Graph Theory, Prentice- Hall of India
	10	Connect and disconnect graph, complement of graph,	This concept is very vital to apply any theorem in real life situation for example to go or send any point to another definitely need to know is there any path exists or not.	D.B.West, Introduction to Graph Theory, Prentice- Hall of India
	5	Walk, path, circuits, Euler graph	Graph theory is applied to the problems in engineering, such that circuit theory, network, connect path etc.	V.K.Balakrishnan, Graph Theory, Schaum's Outline, TMH
	3	Cut-sets, Cut-vertices, Matrix representation of graph, adjacency and incidence matrix of a graph	• To forms a disconnected graph • Converting a Complex Diagram/System To a Fault Tree • An adjacency matrix is a way of representing an n vertex graph • The Adjacency Matrix data structure implements the Graph interface	Kar, Engineering Mathematics II, TMH
	6	Graph isomorphis	• Civil engineering, city planning, building interior planning	Das & Pal, Engineering Mathematics,

		m, Bipartite graph	• A subgraph isomorphism algorithm and its application to biochemical data • Determination of Isomorphism and Its Applications for Arbitrary Graphs Based on Circuit Simulation • Bipartite graphs are extensively used in modern coding theory, especially to decode codewords rece ived from the channel • Bipartite graph is very much applicable for chromatic number, perfect graph, matching problems.	UN Dhar Publishers
	10	Tree, Binary tree,	• Applications of tree data structure • Binary Search Tree - Used in many search applications where data is constantly entering/leaving, such as the map and set objects in many languages' libraries. • Binary Space Partition - Used in almost every 3D video game to determine what objects need to be rendered. • Binary Tries - Used in almost every high-bandwidth router for storing router-tables. • Hash Trees - used in p2p programs and specialized image-signatures in which a hash needs to be verified, but the whole file is not available. • Heaps - Used in implementing efficient priority-queues, which in turn are used for	N.Deo, Graph Theory, Prentice- Hall of India

			scheduling processes in many operating systems, Quality-of-Service in routers, and A* (path-finding algorithm used in AI applications, including robotics and video games). Also used in heap-sort. • Huffman Coding Tree (Chip Uni) - used in compression algorithms, such as those used by the .jpeg and .mp3 file-formats. • GGM Trees - Used in cryptographic applications to generate a tree of pseudo-random numbers. • Syntax Tree - Constructed by compilers and (implicitly) calculators to parse expressions. • Treap - Randomized data structure used in wireless networking and memory allocation. • T-tree - Though most databases use some form of B-tree to store data on the drive, databases which keep all (most) their data in memory often use T-trees to do so.	
	4	Minimal spanning tree, Properties of trees.	The traveling salesman problem, the shortest spanning tree in a weighted graph, obtaining an optimal match of jobs and men and locating the shortest path between two vertices in a graph.	D.B.West, Introduction to Graph Theory, Prentice- Hall of India
	3	Dijkstra's Algorithm for shortest path problem	• Telephone Network • A travel agent requests software for making an agenda of flights for clients	N.Deo, Graph Theory, Prentice- Hall of India

			• to designate a file server in a local area network	
	4	Determination of minimal spanning tree using Kruska's Algorithm	• Studied for ways to lay out electrical networks in a way that minimizes the total cost of the wiring. • if you have a large LAN with many switches, finding a minimum spanning tree will be vital to ensure that only a minimum number of packets will be transmitted across the network. • One of the pratical applications of minimal spanning tree, I can think of is connecting different offices of the same company with least cost.	D.B.West, Introduction to Graph Theory, Prentice- Hall of India
	5	Determination of minimal spanning tree using Prim's Algorithm	• Studied for ways to lay out electrical networks in a way that minimizes the total cost of the wiring. • if you have a large LAN with many switches, finding a minimum spanning tree will be vital to ensure that only a minimum number of packets will be transmitted across the network. • One of the pratical applications of minimal spanning tree, I can think of is connecting different offices of the same company with least cost.	V.K.Balakrishnan, Graph Theory, Schaum's Outline, TMH

Subject Code: MTMG DSE-A

Graph Theory

Total Number of Lectures: 60L

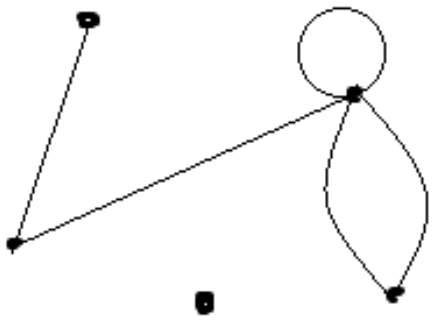
Topic 1. GRAPHS, DIGRAPHS, WEIGHTED GRAPHS

1. **Graph**

1.1 Definition: A graph G is a pair of sets (V,E) where V is the set of vertices and E is the set of edges. E is a multiset, in other words, its elements can occur more than once so that every element has a multiplicity. Often, we label the vertices with letters (for example: a, b, c . . . or v_1, v_2, . . .) or numbers 1, 2, . . . Throughout this lecture material, we will label the elements of V in this way.

Conceptually, a graph is formed by vertices and edges connecting the vertices.

Example1.



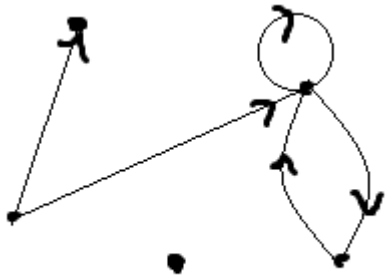
1.2 Directed Graphs

Definition: A directed graph or digraph G is a graph in which each edge $e=(v_i,v_j)$has a direction from its initial vertex v_i to its terminal vertex v_j.

In a digraph G(V,E), each edge e is associated with an ordered pair of vertices

Intuitively, a directed graph or digraph is formed by vertices connected by directed edges or arcs.

Example2.

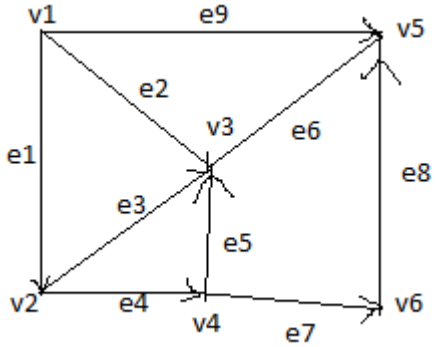


Formally, a digraph is a pair (V, E), where V is the vertex set and E is the edge set in graphs. The difference is that now the elements of E are ordered pairs: the arc from vertex u to vertex v is written as (u, v) and the other pair (v, u) is the opposite direction arc. We also have to keep track of the multiplicity of the arc (direction of a loop is irrelevant). We can pretty much use the same notions and results for digraphs. However:

1. Vertex u is the initial vertex and vertex v is the terminal vertex of the arc (u, v). We also say that the arc is incident out of u and incident into v.
2. The out-degree of the vertex v is the number of edges out of it and the in-degree of v is the number of edges going into it.
3. In the directed walk (trail, path or circuit),
$$v_{i_0}, e_{j_1}, v_{i_1}, e_{j_2}, \ldots, e_{j_k}, v_{i_k}$$
v_{i_ℓ} is the initial vertex and $v_{i_\ell - 1}$ is the terminal vertex of the arc e_{j_ℓ} . .
4. When we treat the graph (V, E) as a usual undirected graph, it is the underlying undirected graph of the digraph G = (V, E), denoted G_U.
5. Digraph G is connected if G_U is connected. The components of G are the directed subgraphs of G that correspond to the components of G_U. The vertices of G are connected if they are connected in G_U. Other notions for undirected graphs can be used for digraphs as well by dealing with the underlying undirected graph.
6. Vertices u and v are strongly connected if there is a directed u– v path and also a directed v– u path in G.
7. Digraph G is strongly connected if every pair of vertices is strongly connected. By convention, the trivial graph is strongly connected.
8. A strongly connected component H of the digraph G is a directed subgraph of G (not a null graph) such that H is strongly connected, but if we add any vertices or arcs to it, then it is not strongly connected anymore.

Every vertex of the digraph G belongs to one strongly connected component of G. However, an arc does not necessarily belong to any strongly connected component of G.

Example3. *For the digraph* G



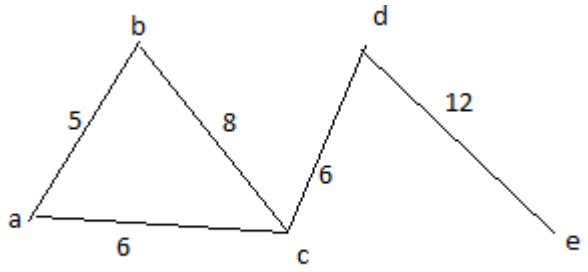
the strongly connected components are ({v_1}, ∅), ({v_2, v_3, v_4}, {e_3, e_4, e_5}), ({v_5}, ∅) *and* ({v_6}, ∅).

The *condensed graph* G_C of the digraph G is obtained by contracting all the arcs in every strongly connected component.

1.3 Waited Graph

In many applications, each edge of a graph has an associated numerical value, called a weight. Usually, the edge weights are nonnegative integers. Weighted graphs may be either directed or undirected.

Example 5: Weighted graphs where edges are assigned some numeric value.

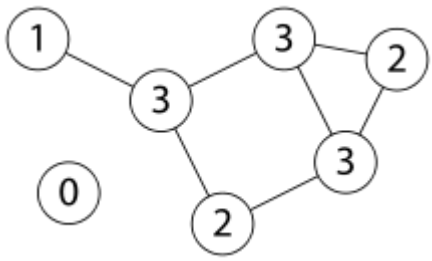


Topic 2. CONNECT AND DISCONNECT GRAPH, COMPLEMENT OF GRAPH

2.1 Connected Graph and Disconnected Graph

A graph is **connected** when there is a path between every pair of vertices. In a connected graph, there are no **unreachable** vertices. A graph that is not connected is **disconnected**. A graph with just one vertex is connected. An edgeless graph with two or more vertices is disconnected.

Example 1: Connected Graph and Disconnected Graph

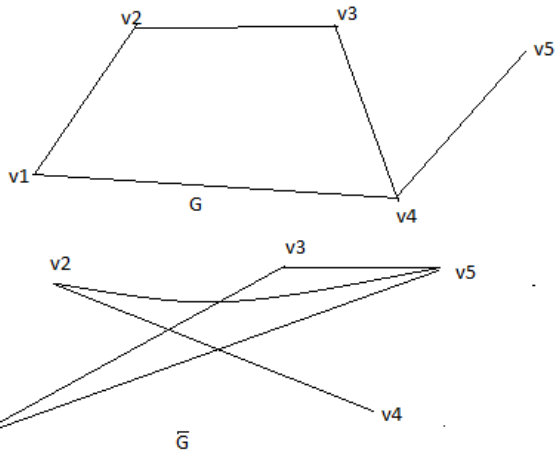


With vertex 0 this graph is disconnected, the rest of the graph is connected

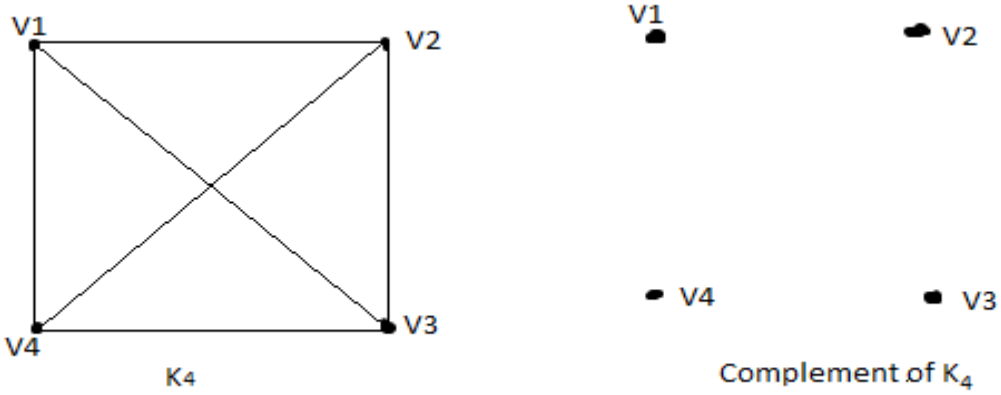
2.2 Complement of Graph

The *complement* of the simple graph G = (V, E) is the simple graph G = (V, E), where the edges in E are exactly the edges not in G.

Example2: complement of graph G is $\overline{G}$



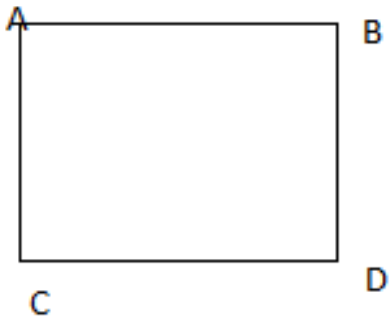
Example3. *The complement of the complete graph* K_n *is the empty graph with* n *vertices.*



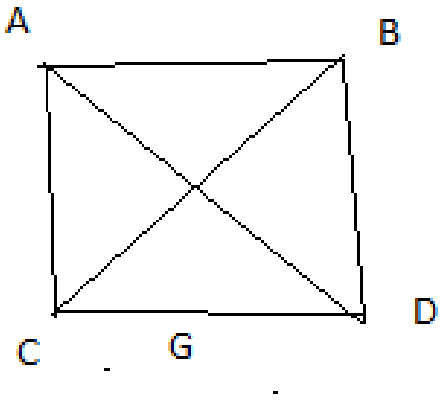
2.3 Regular Graph

In graph theory, a **regular graph** is a graph where each vertex has the same number of neighbors; i.e. every vertex has the samedegree or valency. A regular directed graph must also satisfy the stronger condition that the indegree and outdegree of each vertex are equal to each other. A regular graph with vertices of degree *k* is called a ***k*-regular graph** or regular graph of degree *k*.

Example5: The following represents a 2-regular graph, since every vertex has same degree 2



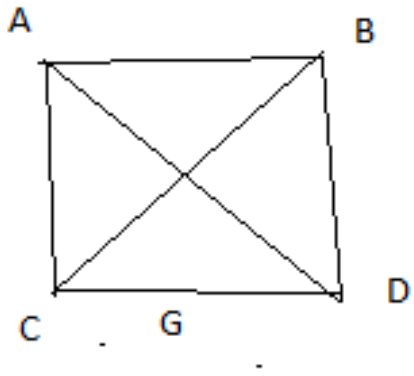
Example6: The following represents a 3-regular graph, since every vertex has same degree 3



2.4 Complete Graph

In the mathematical field of graph theory, a **complete graph** is a simple undirected graph in which every pair of distinct vertices is connected by a unique edge. A **complete digraph** is a directed graph in which every pair of distinct vertices is connected by a pair of unique edges (one in each direction).

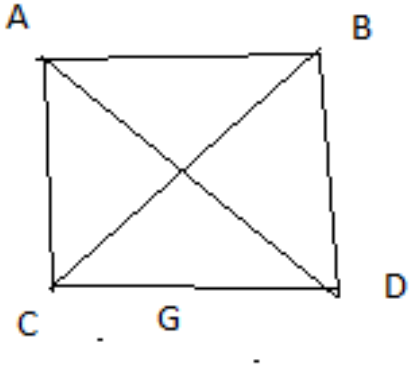
Example7: The following represents a Complete graph



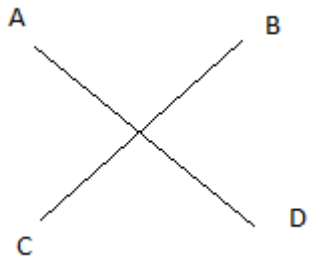
2.5 Subgraph

A graph G' whose graph vertices and graph edges form subsets of the graph vertices and graph edges of a given graph G. If G' is a subgraph of G, then$G$ is said to be a supergraph of G'.

Example 8: The following represents a graph G



Example 9: The following represents a subgraph G'



G'

Topic 3. WALK, PATH, CIRCUITS, EULER GRAPH, THEOREMS RELATED TO GRAPH THEORY

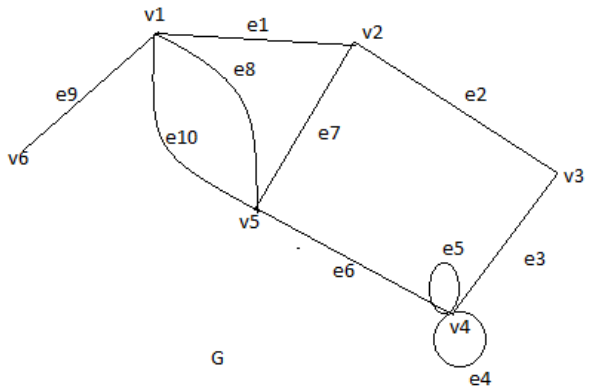
3.1 Walk

A *walk* in the graph G = (V, E) is a finite sequence of the form

$$v_{i_0}, e_{j_1}, v_{i_1}, e_{j_2}, \ldots, e_{j_k}, v_{i_k},$$

which consists of alternating vertices and edges of G. The walk starts at a vertex. Vertices $v_{i_{t-1}}$ and v_{i_t} are end vertices of e_{j_t} (t = 1, . . . , k). v_{i_0} is the *initial vertex* and v_{i_k} is the *terminal vertex*. k is the *length* of the walk. A zero length walk is just a single vertex v_{i_0} . It is allowed to visit a vertex or go through an edge more than once. A walk is *open* if $v_{i_0} \neq v_{i_k}$. Otherwise it is *closed*.

Example 1. *In the graph*



the walk

$$v_2, e_7, v_5, e_8, v_1, e_8, v_5, e_6, v_4, e_5, v_4, e_5, v_4$$

is open. On the other hand, the walk

$$v_4, e_5, v_4, e_3, v_3, e_2, v_2, e_7, v_5, e_6, v_4$$

is closed.

A walk is a ***trail*** if any edge is traversed at most once. Then, the number of times that the vertex pair u, v can appear as consecutive vertices in a trail is at most the number of parallel edges connecting u and v.

Example 2. *(Continuing from the previous example) The walk in the graph*

$$v_1, e_8, v_5, e_9, v_1, e_1, v_2, e_7, v_5, e_6, v_4, e_5, v_4, e_4, v_4$$

is a trail.

A trail is a *path* if any vertex is visited at most once except possibly the initial and terminal vertices when they are the same. A closed path is a ***circuit***. For simplicity, we will assume in the future that a circuit is not empty, i.e. its length ≥ 1. We identify the paths and circuits with the subgraphs induced by their edges.

Example3. *(Continuing from the previous example) The walk*

$$v_2, e_7, v_5, e_6, v_4, e_3, v_3$$

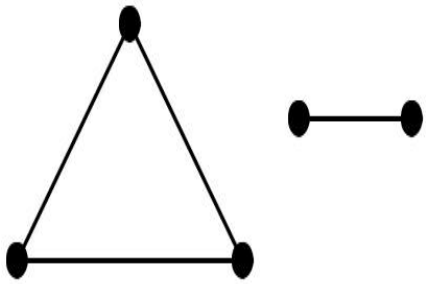
is a path and the walk

$$v_2, e_7, v_5, e_6, v_4, e_3, v_3, e_2, v_2$$

is a circuit.

The walk starting at u and ending at v is called an u– v *walk*. u and v are *connected* if there is a u– v walk in the graph (then there is also a u– v path!). If u and v are connected and v and w are connected, then u and w are also connected, i.e. if there is a u– v walk and a v– w walk, then there is also a u– w walk. A graph is *connected* if all the vertices are connected to each other. (A trivial graph is connected by convention.)

Example4. *The graph is not connected.*



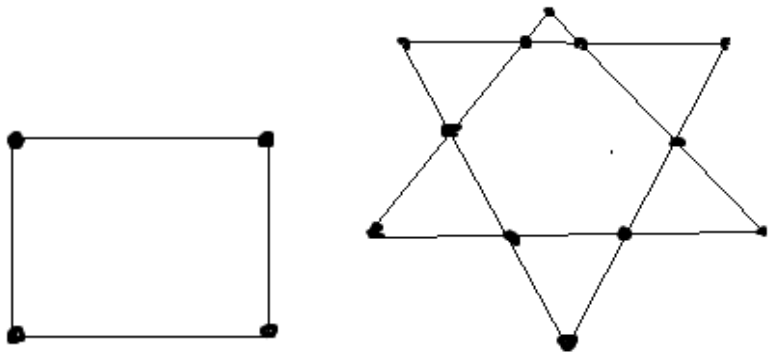
3.2 Euler Graph

If some closed walk in a graph contains all the edges of the graph, then the walk is called an Euler line and the graph is called Euler graph. Euler graph does not have any isolated vertices and therefore connected.

An Euler path is a path that uses every edge of a graph exactly once. An Euler circuit is a circuit that uses every edge of a graph exactly once.

An Euler path starts and ends at different vertices. An Euler circuit starts and ends at the same vertex.

Example 5: Euler Graph



Theorem3.3:(Handshaking Theorem) The sum of degrees of all vertices in a graph G is twice the number of edges in the graph, i.e, the sum of degrees of all vertices in a graph G is always even.

Symbolically, in a graph G(V,E) with $|E| = e$ **number of edges, we have**

$$\sum_{v \in V} d(v) = 2e$$

Proof: We prove the result by induction on e, the number of edges in G.
For e=1, the result is obvious.
Now let G(V,E) be a graph of size e and the theorem is true for any graph of size <e.
Let uv be an edge in G and let $G'(V',E')$be the graph obtained by deleting the edge uv from G .
So $G'(V',E')$ is a graph of size<e.
Therefore, by induction hypothesis $\sum_{v \in V'} d(v) = 2e'$ where $e' = |E'| = e - 1$
Now if we add the edge uv to G', then the sum of the degrees of the vertices is increased by 2, so that $\sum_{v \in V} d(v) = \sum_{v \in V'} d(v) + 2 = 2e' + 2 = 2(e' + 1) = 2e$
Hence the result is proved.

Theorem 3.4: The number of odd degree vertices in a graph is always even.

Proof: Let V and W be the set of vertices of odd degree and even degree respectively. Then by handshaking theorem the sum of degrees of odd degree vertices and even degree vertices of the graph G is equal to twice of edges of the graph i.e.

$$\sum_{v_i \in V} d(v_i) + \sum_{v_i \in W} d(v_i) = 2e$$

$$\sum_{v_i \in V} d(v_i) = 2e - \sum_{v_i \in W} d(v_i)$$

Since $\sum_{v_i \in W} d(v_i)$ is even, therefore $\sum_{v_i \in V} d(v_i)$ is also even, i.e, the number of odd degree vertices in a graph is always even.

Theorem3.5: The maximum number of edges in a simple graph with n vertices is n(n-1)/2

Proof: Let G be a simple graph having n number of vertices and e number of edges.
Then, by handshaking theorem $\sum_{v \in V} d(v) = 2e$
or, $d(v_1)+d(v_2)+\ldots+d(v_n)=2e$
We know that the maximum degree of each vertex in the graph G is (n-1)
Therefore
$d(v_1)=d(v_2)=\ldots=d(v_n)=n-1$
and
(n-1)+ (n-1)+…+ (n-1) (adding n times)=2e
n(n-1)=2e
e=n(n-1)/2

Topic 4. THEOREMS, CUT-SETS, CUT-VERTICES, MATRIX REPRESENTATION OF GRAPH, ADJACENCY AND INCIDENCE MATRIX OF A GRAPH

Theorem 4.1: The number of edges connected with a vertex of a simple graph of n vertices cannot exceed (n-1).

Proof: Let G be a simple graph. A simple graph G has no self-loop and parallel edges.

Let v be any vertex of the graph G. Since G is a simple there is no self-loop and parallel edge. The number of vertices of the graph G is n, so v can be adjacent to at most all the remaining (n-1) vertices of G.

Therefore maximum of edges connected with v in (n-1) i.e, the number of edges of a simple graph of n vertices cannot exceed (n-1).

Theorem 4.2: A complete graph with n vertices consists of n(n-1)/2 number of edges.

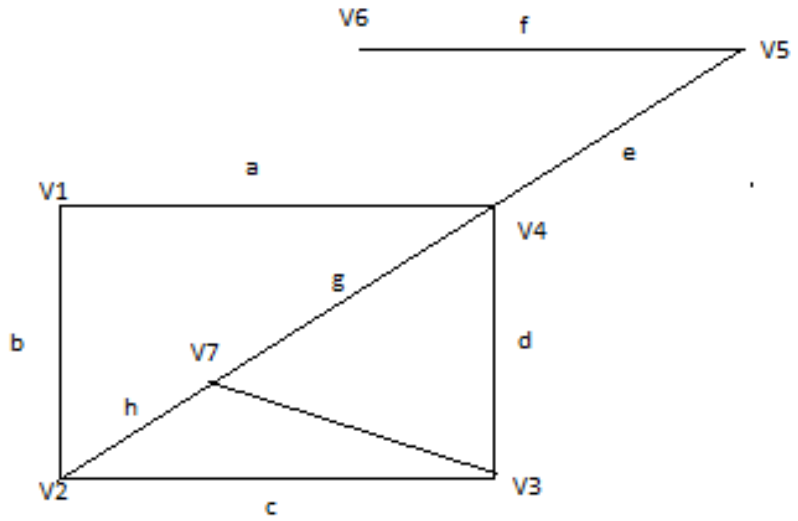
Proof: A simple graph in which there is exactly one edge between each pair of distinct vertices is called a complete graph.

Let G be a complete graph with n vertices and e edges. Since the complete graph has no loop and parallel edges, the number of adjacent vertices of every vertex is (n-1). Hence the total degree of the vertices is (n-1)+ (n-1)+…+ (n-1) (adding n times)=n(n-1)
By handshaking theorem the sum of degrees of all vertices in a graph G twice the number of edges in the graph i.e.
2e=n(n-1)
or, e=n(n-1)/2
Therefore a complete graph with n vertices consists n(n-1)/2 number of edges.

4.3 Cut Set

Let G(V, E) be a connected graph. A cut set for G is defined to be the smallest set of edges such that removal of the set disconnects the graph but removal of any proper subset of this set leaves a connected subgraph of G.

Example1. The cut sets in the following graph



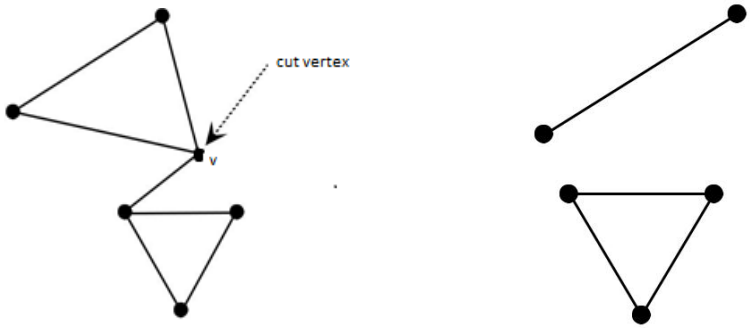
Here some of the cut sets are {a,b}, {f}, {e,f}, {a,g,d}, {a,g,i,c}, etc.

Here {a,g,i,d}is not a cut set, though if we remove the edges the graph becomes disconnected. Because one of its proper subset {a,s,d} is a cut set

4.4. Cut Vertex

Let G(V, E) be any connected graph. A cut vertex for G is a vertex v such that G-{v} has more components other than G or become disconnected.

The subgraph is obtained by deleting the vertex v along with the edges incident to it



G: $G - v$:

(*Note!* Generally, the only vertex of a trivial graph is not a cut vertex, neither is an isolated vertex.)

A graph is *separable* if it is not connected or if there exists at least one cut vertex in the graph. Otherwise, the graph is *nonseparable*.

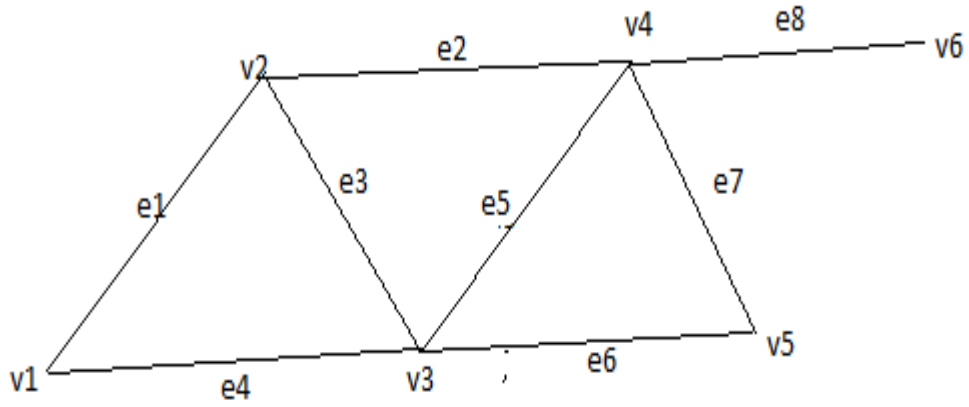
Therefore a *cut set* of the connected graph G = (V, E) is an edge set $F \subseteq E$ such that

1. $G - F$ (remove the edges of F one by one) is not connected, and
2. $G - H$ is connected whenever $H \subset F$.

Theorem 4.4. *If* F *is a cut set of the connected graph* G*, then* G − F *has two components.*

Proof. Let F = {$e_1, \ldots, e_k$}. The graph G − {$e_1, \ldots, e_{k-1}$} is connected (and so is G if k = 1) by condition #2. When we remove the edges from the connected graph, we get at most two components. □

Example1. *In the graph*



{e_1, e_4}, {e_6, e_7}, {e_1, e_2, e_3}, {e_8}, {e_3, e_4, e_5, e_6}, {e_2, e_5, e_7}, {e_2, e_5, e_6} *and* {e_2, e_3, e_4} *are cut sets. Are there other cut sets?*

In a graph G = (V, E), a pair of subsets V_1 and V_2 of V satisfying

$$V = V_1 \cup V_2 , \quad V_1 \cap V_2 = \emptyset , \quad V_1 \neq \emptyset , \quad V_2 \neq \emptyset,$$

is called a *cut* (or a *partition*) of G, denoted $\langle V_1, V_2 \rangle$. Usually, the cuts $\langle V_1, V_2 \rangle$ and $\langle V_2, V_1 \rangle$ are considered to be the same.

Example2. *(Continuing from the previous example)* $\langle \{v_1, v_2, v_3\}, \{v_4, v_5, v_6\} \rangle$ *is a cut.*

We can also think of a cut as an edge set:

cut $\langle V_1, V_2 \rangle$ = {those edges with one end vertex in V_1 and the other end vertex in V_2}.

(*Note!* This edge set does not define V_1 and V_2 uniquely so we can not use this for the definition of a cut.)

Using the previous definitions and concepts, we can easily pr ove the following:

1. The cut $\langle V_1, V_2 \rangle$ of a connected graph G (considered as an edge set) is a cut set if and only if the subgraphs induced by V_1 and V_2 are connected, i.e. G − $\langle V_1, V_2 \rangle$ has two components.
2. If F is a cut set of the connected graph G and V_1 and V_2 are the vertex sets of the two components of G − F , then $\langle V_1, V_2 \rangle$ is a cut and F = $\langle V_1, V_2 \rangle$.
3. If v is a vertex of a connected (nontrivial) graph G = (V, E), then $\langle \{v\}, V - \{v\} \rangle$ is a cut of G. It follows that the cut is a cut set if the subgraph (i.e. G − v) induced by V − {v} is connected, i.e. if v is *not* a cut vertex.

4.5 Matrix Representation of Graphs

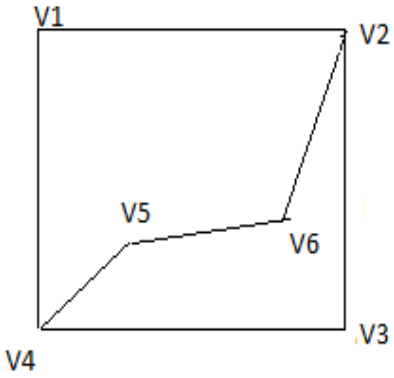
4.5.1 The *adjacency matrix* Representation of Simple Connected Graph

Let G be a simple Graph (i.e. having no parallel edges and self loops) with n vertices $v_1, v_2, v_3, \ldots, v_n$ then the *adjacency matrix* of G is given by a n x n matrix

$A(G)=(a_{ij})_{nxn}$

Where $a_{ij} = \begin{cases} 1; \text{ when } v_i v_j \text{ is an edge of } G \\ 0; \text{if there is no edge between } v_i \text{ and } v_j \text{of } G \end{cases}$

Example3: Find the adjacency matrix of the following graph



Here the number of vertices is 6. The adjacency matrix is

$$A(G) = \begin{bmatrix} & V1 & V2 & V3 & V4 & V5 & V6 \\ V1 & 0 & 1 & 0 & 1 & 0 & 0 \\ V2 & 1 & 0 & 1 & 0 & 0 & 1 \\ V3 & 0 & 1 & 0 & 1 & 0 & 0 \\ V4 & 1 & 0 & 1 & 0 & 1 & 0 \\ V5 & 0 & 0 & 0 & 1 & 0 & 1 \\ V6 & 0 & 1 & 0 & 0 & 1 & 0 \end{bmatrix}$$

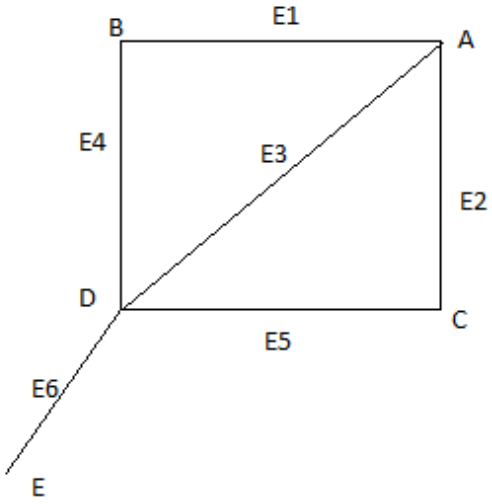
4.5.2 The *Incidence matrix* Representation of Simple Connected Graph

Let G be a simple Graph (i.e. having no parallel edges and self loops) with n vertices $v_1, v_2, v_3, \ldots, v_n$, and m edges $e_1, e_2, \ldots, e_m$ then the ***Incidence matrix*** of G is given by a n x m matrix

$I(G)=(a_{ij})_{nxm}$

Where $a_{ij} = \begin{cases} 1; \text{ when } e_j \text{ is incident on } v_i \text{ of } G \\ 0; \text{if there is no edge } e_j \text{ incident on } v_i \text{ of } G \end{cases}$

Example 4: Find the ***Incidence matrix*** of the following graph



Here we have 5 vertices and 6 edges, so the incidence matrix is of order 5 x 6 and is given by

$$\mathbf{I(G)=}\begin{bmatrix} & E1 & E2 & E3 & E4 & E5 & E6 \\ A & 0 & 1 & 0 & 1 & 0 & 0 \\ B & 1 & 0 & 1 & 0 & 0 & 1 \\ C & 0 & 1 & 0 & 1 & 0 & 0 \\ D & 1 & 0 & 1 & 0 & 1 & 0 \\ E & 0 & 0 & 0 & 1 & 0 & 1 \end{bmatrix}$$

Topic 5. GRAPH ISOMORPHISM, BIPARTITE GRAPH

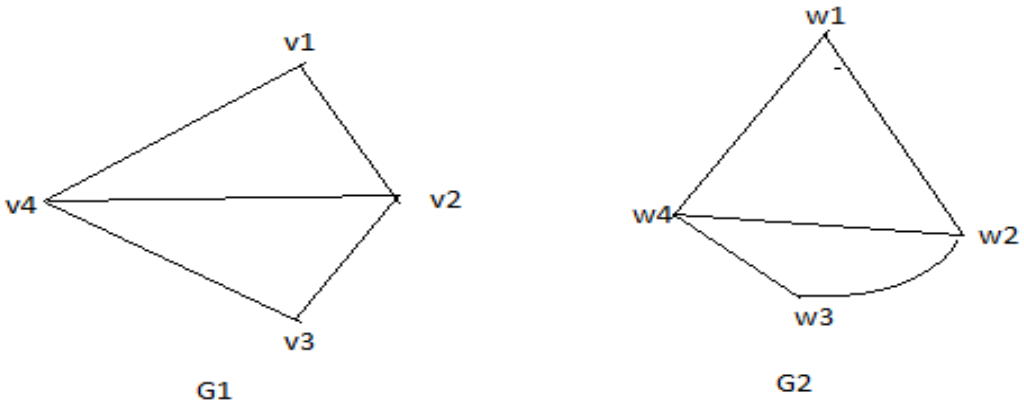
5.1 Isomorphism

Two graphs G_1 and G_2 are isomorphic if there is a function $f : V(G_1) \to V(G_2)$ from the vertices of $G_1$to the vertices of G_2 such that

1. f is one to one
2. f is onto, and
3. for each pair of vertex u and v of G_1, $\{u,v\} \in E(G_1)$ if and only if $\{f(u), f(v)\} \in E(G_2)$

The two graphs shown below are isomorphic, despite their different looking drawings.

Example1: Examine weather the following two graphs are isomorphic



Solution: Both the graphs G_1 and G_2 have 4 vertices and 5 edges

Also in the G_1 and G_2 there are 2 vertices of degree 3 and 2 vertices of degree 2.

Hence the necessary condition of isomorphism are satisfied

The adjacency matrices of the two graphs are

$$A(G_1) = \begin{bmatrix} & v1 & v2 & v3 & v4 \\ v1 & 0 & 1 & 0 & 1 \\ v2 & 1 & 0 & 1 & 1 \\ v3 & 0 & 1 & 0 & 1 \\ v4 & 1 & 1 & 1 & 0 \end{bmatrix} \qquad A(G_2) = \begin{bmatrix} & w1 & w2 & w3 & w4 \\ w1 & 0 & 1 & 0 & 1 \\ w2 & 1 & 0 & 1 & 1 \\ w3 & 0 & 1 & 0 & 1 \\ w4 & 1 & 1 & 1 & 0 \end{bmatrix}$$

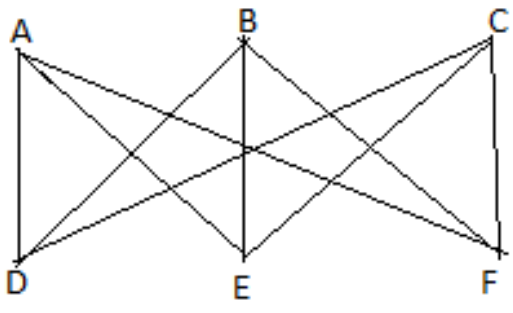
Since $A(G_1)=A(G_2)$

Therefore the two graphs are isomorphic.

5.2 Bipartite

In the mathematical field of graph theory, a **bipartite graph** (or **bigraph**) is a graph whose vertices can be divided into two disjoint sets U and V (that is,U and V are each independent sets) such that every edge connects a vertex in U to one inV . Vertex sets U and V are usually called the *parts* of the graph.

Example2: The following represents a complete bipartite graph

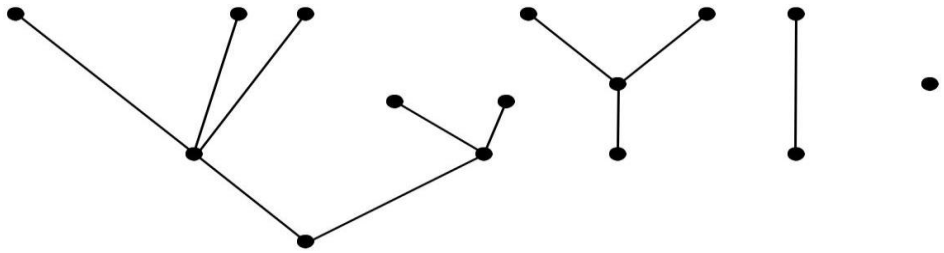


Topic 6. TREE, BINARY TREE, THEOREM

6.1 Trees and Forests

A forest is a circuitless graph. A tree is a connected forest. A subforest is a subgraph of a forest. A connected subgraph of a tree is a subtree. Generally speaking, a subforest (respectively subtree) of a graph is its subgraph, which is also a forest (respectively tree).

Example1. Four trees which together form a forest:



Theorem 6.2: Prove that if there is one and only one path between every pair of vertices in a graph G then G is tree.

Proof: Both the way we can prove the theorem, First let us consider G be a tree, then from the definition G is connected and so there exists a connected path in two vertices in G. The path must be unique otherwise if there exists more than one edge between two vertices then it will form a cycle which is a contradiction that G is a tree, so there exists unique path in between the vertices.

Conversely let G be a connected graph such that two vertices are connected by unique path. To prove G is tree we have to show there is no cycle in G. If possible let G contains a cycle then between two vertices there exists two paths which contradicts that G is connected by unique path.

Theorem 6.3: Prove that a tree with n vertices has (n−1) edges.

Proof: We prove the result by using induction on n, the number of vertices. The result is obviously true for n = 1, 2 and 3. Let the result be true for all trees with fewer than n vertices. Let T be a tree with n vertices and let e be an edge with end vertices u and v. So the only path between u and v is e. Therefore deletion of e from T disconnects T. Now, T − e consists of exactly two components T1 and T2 say, and as there were no cycles to begin with, each component is a tree. Let n1 and n2 be the number of vertices in T1 and T2 respectively, so that n1 + n2 = n. Also, n1 < n and n2 < n. Thus, by induction hypothesis, number of edges in T1 and T2 are respectively n1 −1 and n2 −1. Hence the number of edges in T = n1 −1+n2 −1+1 = n1 +n2 −1 = n−1.

Theorem 6.4: Prove that a connected graph with n vertices and (n−1) edges is a tree.

Proof: Let G be a connected graph with n vertices and n − 1 edges. We show that G contains no cycles. Assume to the contrary that G contains cycles. Remove an edge from a cycle so that the resulting graph is again connected. Continue this process of removing one edge from one cycle at a time till the resulting graph H is a tree. As H has n vertices, so number of edges in H is n−1. Now, the number of edges in G is greater than the number of edges in H. So n−1 > n−1, which is not possible. Hence, G has no cycles and therefore is a tree.

Topic 7. MINIMAL SPANNING TREE, PROPERTIES OF TREES

Theorem 7.1: Prove that the number of vertices in a binary tree is always odd

Proof: Let T be the binary tree with n vertices. From definition we know that in a binary tree one vertex is of degree 2 and remaining vertices are of degree either one of three. So the number of odd degree vertices in T is n-1. Again we know, the number of odd degree vertices is always even, so n-1 must be even. It is possible only when n is odd, Hence the theorem.

Theorem 7.2: The number of leaves (pendent vertices) in a binary tree with n vertices is given by (n+1)/2.

Proof Let K be a binary tree with n vertices. Let us consider r be the number of pendant vertices in the graph K. Then there are (n−r) internal vertices in the graph K and (n−r−1) vertices of degree three. Thus the number of edges in K = [3(n−r−1) + (2+r)]/2. But the number of edges in K is (n−1). Hence, [3(n−r−1) + (2+r)]/2 = n−1, so that r = (n+1)/2.

7.3 spanning tree

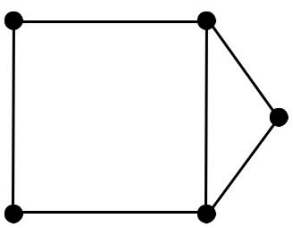
A spanning tree of a connected graph is a subtree that includes all the vertices of that graph.

7.4 Minimum spanning tree

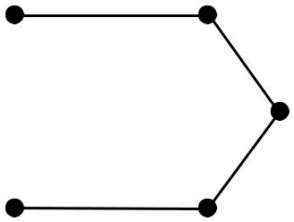
A minimum spanning tree is a spanning tree of a connected, undirected graph. It connects all the vertices together with the minimal total weighting for its edges.

A single graph can have many different spanning trees. We can also assign a *weight* to each edge, which is a number representing how unfavorable it is, and use this to assign a weight to a spanning tree by computing the sum of the weights of the edges in that spanning tree. A minimum spanning tree (MST) or minimum weight spanning tree is then a spanning tree with weight less than or equal to the weight of every other spanning tree. More generally, any undirected graph (not necessarily connected) has a minimum spanning forest, which is a union of minimum spanning trees for its connected components.

Example1.



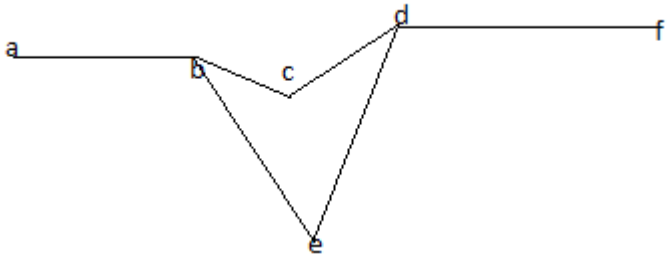
G:



spanning tree:

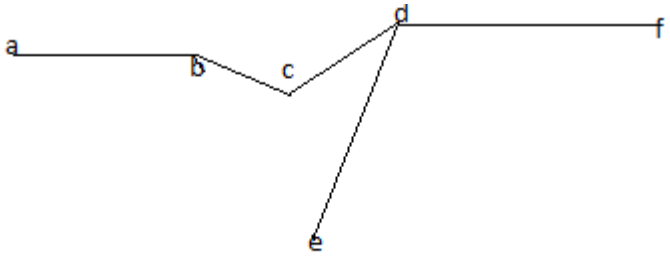
The edges of a spanning tree are called *branches* and the edges of the corresponding cospanning tree are called *links* or *chords*.

Example2: Construct spanning tree from the following graph.

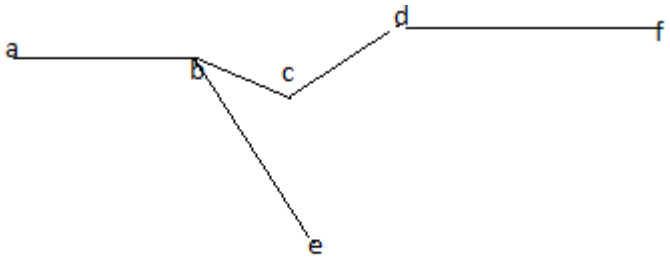


Solution:

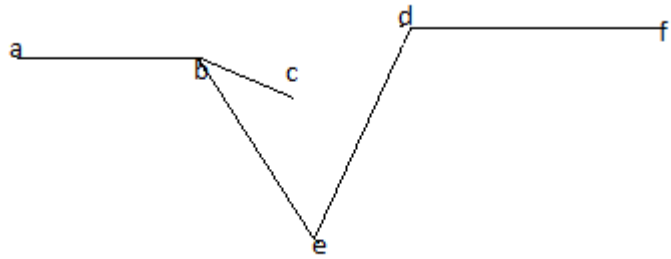
The spanning trees are given as follows:



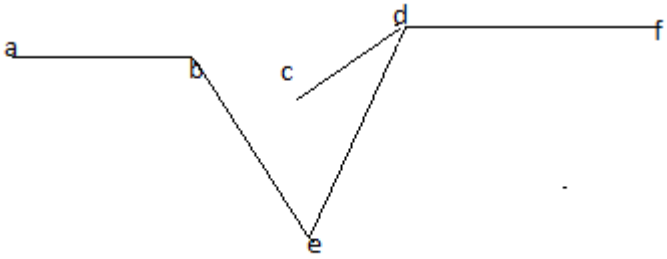
Spanning Tree 1



Spanning Tree 2



Spanning Tree 3



Spanning Tree 4

Topic 8. DIJKSTRA'S ALGORITHM FOR SHORTEST PATH PROBLEM

8.1 Dijkstra's Algorithm

A simple weighted graph G (V, E, w) with n vertices is described by an n x n matrix $W= (w_{ij})_{nxn}$

$$w_{ij} = \begin{cases} \text{weight (or distance or cost) of the edge from i to j} \\ 0 \\ \infty, \text{ if there is no edge from vertex i to j} \end{cases}$$

If the graph is not simple i.e. if the graph contains any loop, then discard it. Also if G contains parallel edges between any two vertices, discard all except the edge having the least weight.

Let us consider that we are to find out the shortest path from a specified vertex s to another specified vertex t.

At each stage in the algorithm some vertices have permanent labels and others have temporary labels. Labels of a vertex v is denoted by L(v).

Assign first, the permanent label 0 to the starting vertex s, i.e. L(s)=0 and a temporary label ∞ to the remaining vertices. Subsequently in each iteration another vertex gets a permanent label.

Step1: Every vertex j that is not yet permanently labelled gets a new temporary label whose value is given by

$L(j)=\min\{oldL(j), (oldL(i)+w_{ij})\}$

where i is the latest vertex permanently labelled in the last iteration and w_{ij} is the direct distance between the vertices i and j. If i and j are not joined by an edge then $w_{ij}=\infty$.

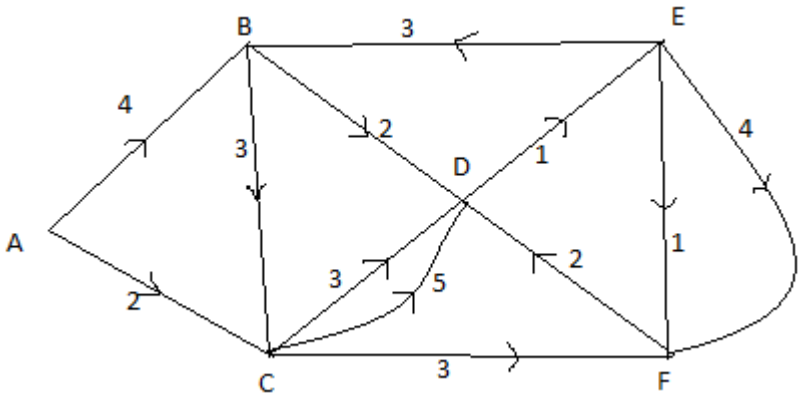
Step2: The smallest value among all the temporary labels is marked and this is the permanent label of the corresponding vertex. In case of tie, select any one for permanent labelling.

Step3: Step1 and Step2 are repeated alternately until the destination vertex t gets a permanent label.

Every stage of labelling will be displayed in the table. In the table permanent label of every vertices will be shown enclosed in a square.

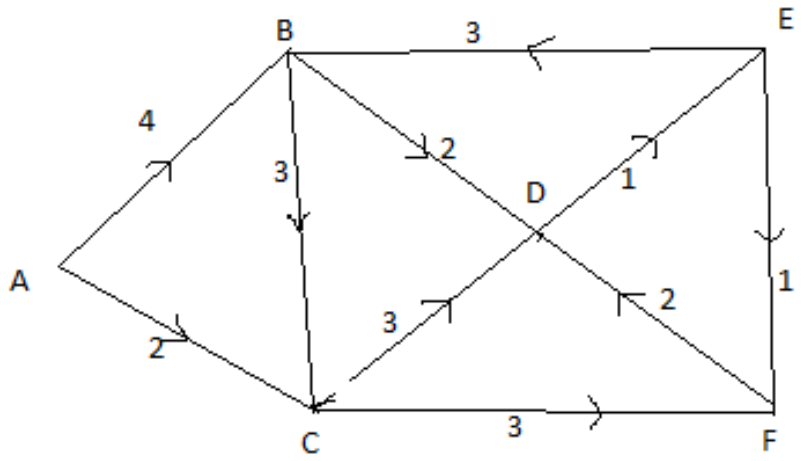
Here the shortest distance of the destination vertex t is found by its value of the permanent label. The shortest path will be formed by backtracking technique from the table.

Example1: Find the shortest path and distance from A to E using Dijkstra's Algorithm.



The given graph is not simple weighted. So, we make it simple

First we delete the edge CD of weight 5, because 3 is the minimum weight. We also delete the edge EF of weight 4 for same reason. Then the graph becomes simple and is given by



The weighted table W is formed on the basis

$$w_{ij} = \begin{cases} \text{weight (or distance or cost) of the edge from i to j} \\ 0 \\ \infty, \text{ if there is no edge from vertex i to j} \end{cases}$$

And is given by the following

	A	B	C	D	E	F
A	0	4	2	∞	∞	∞
B	∞	0	3	2	∞	∞
C	∞	∞	0	3	∞	3
D	∞	∞	∞	0	1	∞
E	∞	3	∞	∞	0	1
F	∞	∞	∞	2	∞	0

Here we are to find the shortest path from the vertex A to the vertex E. So we are to start our computation by assigning permanent label 0 to the vertex A, i.e. L(A)=0 and other temporary label ∞ to all others. Permanent label is shown in a square. Now at the every stage we compute temporary labels for all the vertices except those that already have permanent labels, and only some of them will get permanent labels. We continue this process until the destination vertex E gets a permanent label.

The computation is shown in the following table:

	A	B	C	D	E	F	
Step1	0	∞	∞	∞	∞	∞	**:** A got the permanent label 0 and all others have temporary label ∞
Step2	0	4	2	∞	∞	∞	**:** Calculation of temporary labels and 2 is the minimum among all
Step3	0	4	2	∞	∞	∞	: C got the permanent label
Step4	0	4	2	5	∞	5	: Calculation of temporary labels and 4 is the minimum among all
Step5	0	4	2	5	∞	5	: B got the permanent label
Step6	0	4	2	5	∞	5	: Calculation of temporary labels and 4 is the minimum among all
Step7	0	4	2	5	∞	5	: There is a tie, so we select D arbitrarily. D got the permanent label
Step8	0	4	2	5	6	5	: Calculation of temporary labels and 4 is the minimum among all
Step9	0	4	2	5	6	5	: F got the permanent label
Step10	0	4	2	5	6	5	: Calculation of temporary labels and 6 is the only label
Step11	0	4	2	5	6	5	: Destination vertex E got the permanent label

Now we apply backtracking technique for finding shortest path. Starting from the permanent label E (from step 11) we traverse back and see that in step 7, it is changed and at that step D has got the permanent label. So, we move to D. Repeating the same things we see that in step 3, the label of D is changed and at that step C has got permanent label. So we move to C. Now if we apply the same technique, then in step 1 the label of C is changed and at that step source vertex A has got the permanent label. So we reach at the source vertex and stop the process.

Hence the shortest path is given by

A→C→D→E

5.2 Reachability: Warshall's Algorithm

We only deal with directed graphs in this section. The results also hold for ”undirected” graphs if we interpret an edge as a pair of arcs in opposite directions.

Problem. *We are given an adjacency matrix of the digraph* G = (V, E). *We are to construct the* reachability matrix **R** = (r_{ij}) *of* G, *where*

$$r_{ij} = \begin{cases} 1 \text{ if } G \text{ has a directed } v^i - v^j \text{ path} \\ 0 \text{ otherwise.} \end{cases}$$

(Note that V = {$v_1, \ldots, v_n$}.*) In particular, we should note that if* $r_{ii} = 1$, *then* v_i *is in a directed circuit.*

Warshall's Algorithm constructs a series of n × n matrices $\mathbf{E}_1, \ldots, \mathbf{E}_n$ where

1. elements of $\mathbf{E}_i$ are either zero or one.
2. $\mathbf{E}_i \leq \mathbf{E}_{i+1}$ (i = 0, . . . , n − 1) (comparison is done element by element).
3. $\mathbf{E}_0$ is obtained from the adjacency matrix **D** by replacing the positive elements with ones.
4. $\mathbf{E}_n = \mathbf{R}$.

5.5 The Lightest Path: Floyd's Algorithm

Problem. *We are to find the lighest path from vertex* u *to vertex* v *(*6= u*) in a digraph or to show that there is no such path when the arcs of the digraph have been assigned arbitrary weights. Note that the weight of a directed path is the sum of the weights of the arcs traversed.*

Obviously, we can assume there are no loops or parallel arcs. Otherwise, we simply remove the loops and choose the arc with the lowest weight out of the parallel arcs. Floyd's Algorithm only works for digraphs. We write the weight of (x, y) as α(x, y) and construct the *weight matrix* **W** = (w_{ij}) where

$$w_{ij} = \begin{cases} \alpha(v_i, v_j) \text{ if there is an arc } (v_i, v_j) \\ \infty \text{ otherwise.} \end{cases}$$

(Once again, $V = \{v_1, \ldots, v_n\}$ is the vertex set of the digraph.) Floyd's Algorithm is similar to Warshall's Algorithm. It only works if the digraph has no negative cycles, i.e. no directed circuit in the digraph has a negative weight. In this case, the lightest directed path is the lightest directed walk.

Floyd's Algorithm constructs a sequence of matrices $\mathbf{W}_0, \mathbf{W}_1, \ldots, \mathbf{W}_n$ where $\mathbf{W}_0 = \mathbf{W}$

and

$(\mathbf{W}_k)_{ij}$ = weight of the lightest directed v_i– v_j path,

where there are only vertices $v_1, \ldots, v_k$ on the path besides v_i and v_j

(= ∞ if there is no such path).

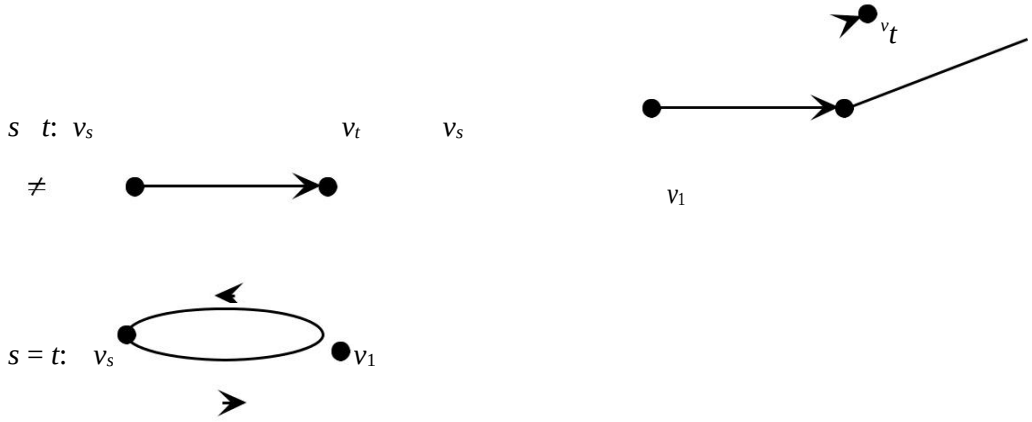
Statement. *When* $\mathbf{W}_k$ *is computed from* $\mathbf{W}_{k-1}$ *by the formula*

$$(\mathbf{W}_k)_{st} = \text{MIN}\{(\mathbf{W}_{k-1})_{st}, (\mathbf{W}_{k-1})_{sk} + (\mathbf{W}_{k-1})_{kt}\},$$

then we get the previously mentioned sequence of weight matrices. If the digraph has negative cycles, then the sequence is correct up to the point when one of the diagonal elements turns negative for the first time.

Proof. We use induction on k.

Induction Basis: k = 1. Since the digraph is loopless, the diagonal elements of $\mathbf{W}_0$ can only be ∞ and the lightest directed path (if there is one) is one of the following, and the statement is obvious:



Induction Hypothesis: The statement is true for $k < \ell$. ($\ell \geq 2$)
Induction Statement: The statement is true for $k = \ell$.

Induction Statement Proof: The diagonal elements of $\mathbf{W}_{\ell-1}$ have to be nonnegative (∞ is permitted) for us to get this k. Let us consider the case where s 6= t. (The case s = t is analogous.) We have five cases:

- Vertex v_ℓ is on the lightest directed path but it is not v_s or v_t, i.e. ℓ 6= s, t. Let us consider the directed subpath from v_s to v_ℓ whose vertices other than v_s and v_ℓ are in $\{v_1, \ldots, v_{\ell-1}\}$. Suppose the lightest directed v_s– v_ℓ path of this kind has common vertices with the directed subpath from v_ℓ to v_t other than v_ℓ itself, e.g. v_p. The directed v_s– v_p– v_ℓ– v_t walk we get would be lighter than

the original directed v_s– v_t path. By removing cycles, we would get a directed v_s– v_t path that would be lighter and would only contain

√

v_s as well as v_t and the vertices $v_1, \ldots, v_\ell$ ( ). (We have to remember that weights of cycles are not negative!) Therefore, the directed subpath from v_s to v_ℓ is the lightest di-rected v_s– v_ℓ path which contains the vertices $v_1, \ldots, v_{\ell-1}$ as well as v_s and v_ℓ. Similarly, the directed subpath from v_ℓ to v_t is the lightest directed v_ℓ– v_t path which contains the vertices $v_1, \ldots, v_{\ell-1}$ as well as v_t and v_ℓ. Now, we use the Induction Hypothesis:

$$(\mathbf{W}^{\ell})_{st} < (\mathbf{W}^{\ell-1})_{st}$$

(check the special case $(\mathbf{W}_{\ell-1})_{st} = \infty$) and

$$(\mathbf{W}^{\ell})_{st} = (\mathbf{W}^{\ell-1})_{s\ell} + (\mathbf{W}^{\ell-1})_{\ell t}.$$

- The directed v_s– v_t path with the lowest weight exists and $v_\ell = v_s$. By the Induction Hypothesis, $(\mathbf{W}^{\ell})_{st} = (\mathbf{W}_{\ell-1})_{st}$ and

$$(\mathbf{W}^{\ell-1})_{s\ell} + (\mathbf{W}^{\ell-1})_{\ell t} = (\mathbf{W}^{\ell-1})_{\ell\ell} + (\mathbf{W}^{\ell-1})_{\ell t} \geq (\mathbf{W}^{\ell-1})_{\ell t} = (\mathbf{W}^{\ell-1})_{st},$$

since $(\mathbf{W}_{\ell-1})_{\ell\ell} \geq 0$ (possibly $= \infty$).

- The directed v_s– v_t path exists and $v_\ell = v_t$. By the Induction Hypothesis, $(\mathbf{W}^{\ell})_{st} = (\mathbf{W}_{\ell-1})_{st}$ and

$$(\mathbf{W}^{\ell-1})_{s\ell} + (\mathbf{W}^{\ell-1})_{\ell t} = (\mathbf{W}^{\ell-1})_{s\ell} + (\mathbf{W}^{\ell-1})_{\ell\ell} \geq (\mathbf{W}^{\ell-1})_{s\ell} = (\mathbf{W}^{\ell-1})_{st},$$

since $(\mathbf{W}_{\ell-1})_{\ell\ell} \geq 0$ (possibly $= \infty$).

- The lightest directed v_s– v_t path exists but v_ℓ is not on the path. Now, we construct the lightest directed v_s– v_ℓ path and the lightest v_ℓ– v_t path which, in addition to the end ver-tices, contain only vertices $v_1, \ldots, v_{\ell-1}$, if it is possible. By combining these two paths, we get a directed v_s– v_t walk. By removing possible cycles from this walk, we get an as light or even lighter v_s– v_t path, which only contains vertices $v_1, \ldots, v_\ell$ as well as v_s and v_t. (We have to remember that weights of cycles are not negative!) Therefore, this is a case where

$$(\mathbf{W}^{\ell-1})_{s\ell} + (\mathbf{W}^{\ell-1})_{\ell t} \geq (\mathbf{W}^{\ell-1})_{st}$$

and the equation in the statement gives the right result. If there is no directed v_s– v_ℓ path or v_ℓ– v_t path, then it is obvious.

- The lightest directed v_s– v_t path does not exist. Then, $(\mathbf{W}^{\ell})_{st} = \infty$ and $(\mathbf{W}_{\ell-1})_{st} = \infty$. On the other hand, at least one of the elements $(\mathbf{W}_{\ell-1})_{s\ell}$ or $(\mathbf{W}_{\ell-1})_{\ell t}$ is $= \infty$ because otherwise we would get a directed v_s– v_t path by combining the v_s– v_ℓ path with the v_ℓ– v_t

√ □

path as well as removing all possible cycles ( ).

Floyd's Algorithm also constructs another sequence of matrices $\mathbf{Z}_0, \ldots, \mathbf{Z}_n$ in which we store the lightest directed paths in the following form

$$(\mathbf{Z}_k)_{ij} = \begin{cases} \ell \text{ where } v_\ell \text{ is the vertex following } v_i \text{ on the lightest directed} \\ v_i - v_i \text{ path containing only vertices } v_i \text{ and } v_i \text{ as well as } v_1, \ldots, v_k \\ \text{(if such a path exists)} \\ 0 \text{ otherwise.} \end{cases}$$

Obviously,

$$(\mathbf{Z}_0)_{ij} = \begin{cases} j \text{ if } (\mathbf{W})_{ij} \neq \infty \\ 0 \text{ otherwise.} \end{cases}$$

The matrix $\mathbf{Z}_k$ ($k \geq 1$) of the sequence can be obtained from the matrix $\mathbf{Z}_{k-1}$ by

$$(\mathbf{Z}_k)_{ij} = \begin{cases} (\mathbf{Z}_{k-1})_{ik} \text{ if } (\mathbf{W}_{k-1})_{ik} + (\mathbf{W}_{k-1})_{kj} < (\mathbf{W}_{k-1})_{ij} \\ (\mathbf{Z}_{k-1})_{ij} \text{ otherwise,} \end{cases}$$

so the sequence can be constructed with the sequence $\mathbf{W}_0, \mathbf{W}_1, \ldots, \mathbf{W}_n$ at the same time. Finally, Floyd's Algorithm is presented in the following pseudocode. We have added a part to test if there are negative elements on the diagonal and the construction of the $\mathbf{Z}_0, \ldots, \mathbf{Z}_n$ sequence of matrices.

Topic 9. DETERMINATION OF MINIMAL SPANNING TREE USING KRUSKAL'S ALGORITHM

9.1 Determination of minimal spanning tree

9.1.1 Kruskal's algorithm

Input: The provided Graph

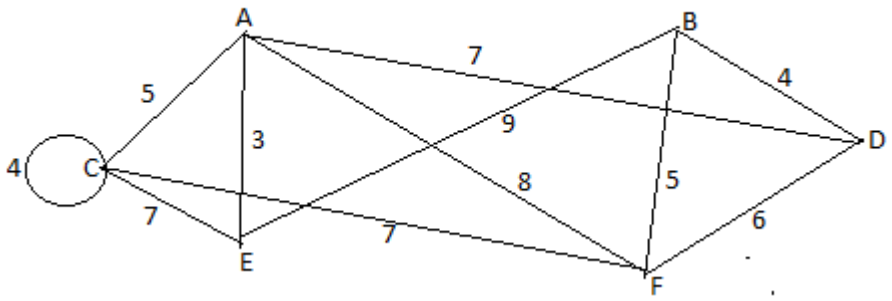
Step1: Arrange all the edges of G, except the loops in the order of non-decreasing weights

Step 2: Select the first minimum edge of the list of edges.

Step 3: Add the next edge of smallest weights to the previous one which does not make any cycle or select second minimum edge of the list of edges.

Step 4: Repeat Step 2 & 3 until n-1 edges are selected.

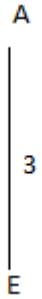
Example1: Using Kruskal's Algorithm find the minimal spanning tree of the following graph



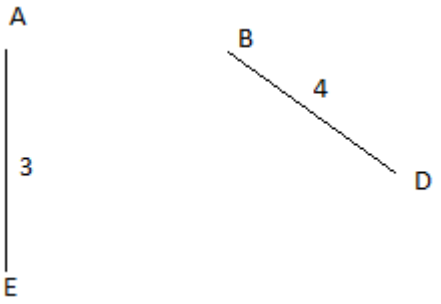
First we select all the edges of G, except the loops in the order of non-decreasing weights and write it in the following form:

Edges:	AE	BD	AC	BF	DF	CE	CF	AD	AF	BE
Weights:	3	4	5	5	6	7	7	7	8	9

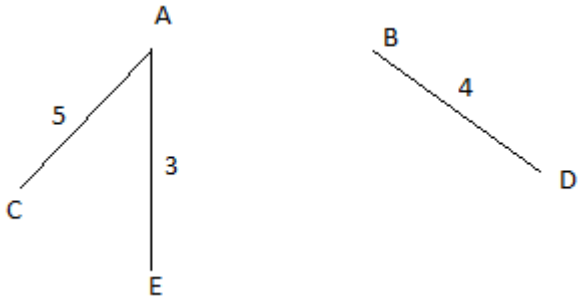
Step1: First select edge AE from the list, since it has the minimum weight



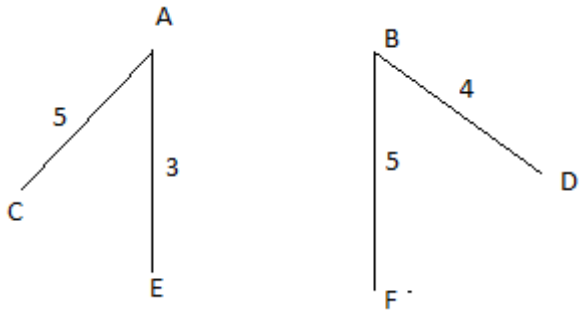
Step2: The next edge of smallest weight is BD. We can add it to the previous one because it does not form any cycle.



Step3: The next edge of smallest weight is AC. We can add it to the previous one because it does not form any cycle.

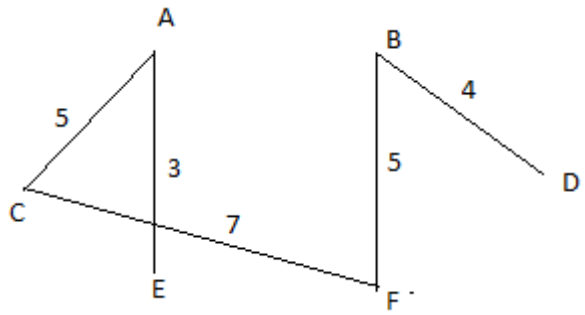


Step4: The next edge of smallest weight is BF. We can add it to the previous one because it does not form any cycle.

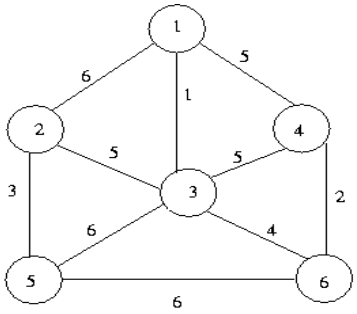


Step5: The next edge of smallest weight is DF. We discard it because it result in a cycle. Also we discard CE due to the same reason.

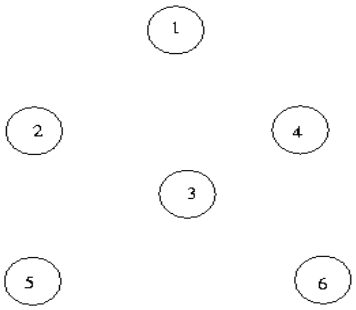
Step6: The next edge of smallest weight is CF. We can add it to the previous one because it does not form any cycle.



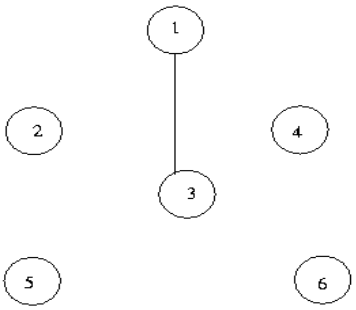
Example2: An illustration of Kruskal's algorithm



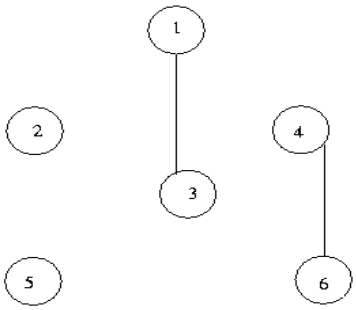
Edges:	12	13	14	23	25	34	35	36	46	56
Weights:	6	1	5	5	3	5	6	4	2	6



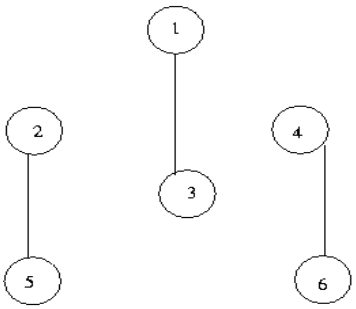
Initial Configuration



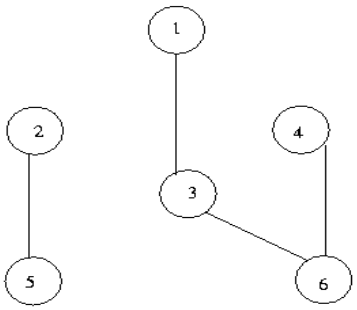
step1. choose (1,3)



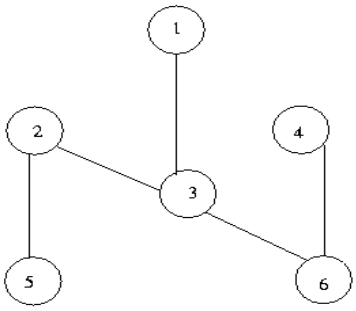
step2. choose (4,6)



step3. choose (2,5)

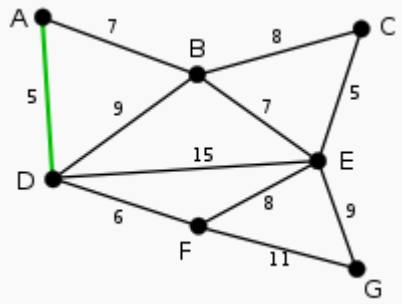
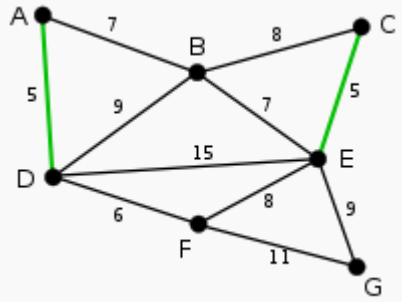
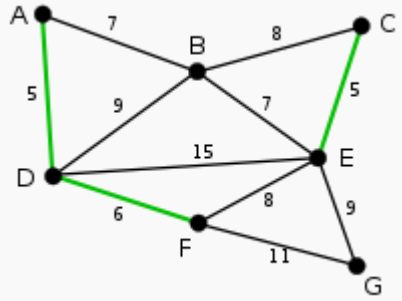
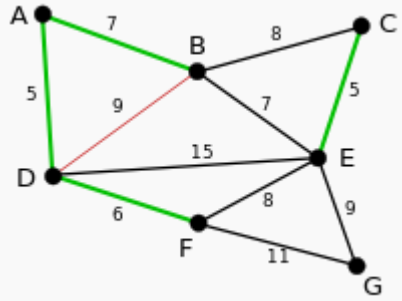


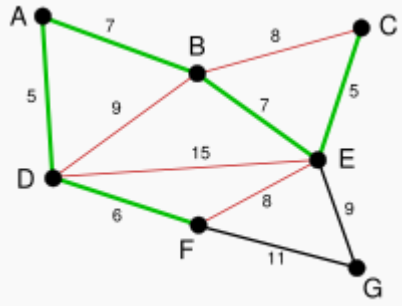
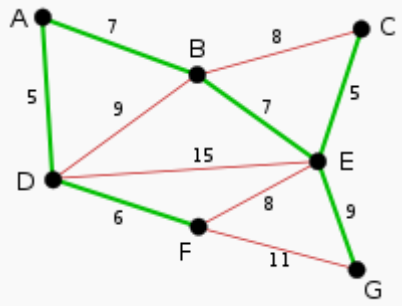
step4. choose (3,6)



step5. choose (2,3)

Example3: An illustration of Kruskal's algorithm

Image	Description
	**AD** and **CE** are the shortest edges, with length 5, and **AD** has been arbitrarily chosen, so it is highlighted.
	**CE** is now the shortest edge that does not form a cycle, with length 5, so it is highlighted as the second edge.
	The next edge, **DF** with length 6, is highlighted using much the same method.
	The next-shortest edges are **AB** and **BE**, both with length 7. **AB** is chosen arbitrarily, and is highlighted. The edge **BD** has been highlighted in red, because there already exists a path (in green) between **B** and **D**, so it would form a cycle (**ABD**) if it were chosen.

	The process continues to highlight the next-smallest edge, **BE** with length 7. Many more edges are highlighted in red at this stage: **BC**because it would form the loop **BCE**, **DE** because it would form the loop **DEBA**, and **FE** because it would form **FEBAD**.
	Finally, the process finishes with the edge **EG** of length 9, and the minimum spanning tree is found.

Topic 10. DETERMINATION OF MINIMAL SPANNING TREE USING PRIM'S ALGORITHM

10.1 Prim's algorithm

First we remove all the self-loop and parallel edges if exists from the given graph G except the edge with the minimum weight for any pair of vertices.

Next label the n vertices by $v_1, v_2, \ldots, v_n$. Weights of the edges are tabulated in an n x n table, of which ij^{th} eliment w_{ij} is given by the following rule

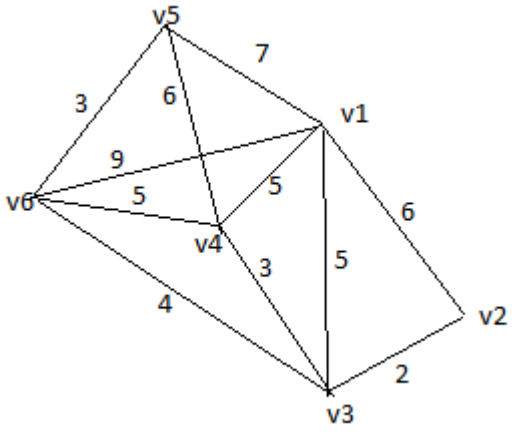
w_{ij}=weight of the edge between v_i and v_j

$w_{ij} = \infty$, if there is no direct edge between v_i and v_j

In n x n table we replace the diagonal values and put '-'and it is to be noted that the entries in the table are symmetric w.r.t its diagonal.

Now start from the vertex v_1.Connect it to the nearest adjacent vertex, i.e. the vertex for which there is the smallest entry in the row1 of the table. Suppose the vertex is v_i. Now consider the edge containing $v_1$and v_i as one subgraph and connect this subgraph to its nearest neighbour i.e the vertex other than $v_1$and v_i for which there is the smallest entry in the row 1and row i of the table. Suppose the vertex is v_k. Next consider the tree with the vertices v_1,v_i and v_k as one subgraph and continue the process of connecting until all the n vertices are connect by n-1 edges. If there is a tie for selecting the smallest entry in any row then we choose arbitrarily.

Example1: Using Prim's Algorithm find the minimal spanning tree of the following graph:

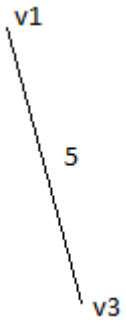


Here we have 6 vertices. So the minimal spanning tree will be with 6-1=5 edges.

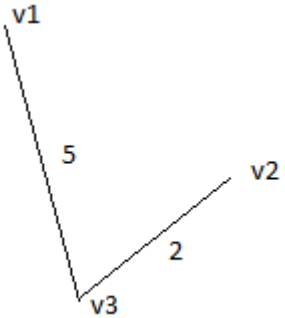
First we make the weight table in the following manner

	v1	v2	v3	v4	v5	v6
v1	–	6	5	5	7	9
v2	6	–	2	∞	∞	∞
v3	5	2	–	3	∞	4
v4	5	∞	3	–	6	5
v5	7	∞	∞	6	–	3
v6	9	∞	4	5	3	–

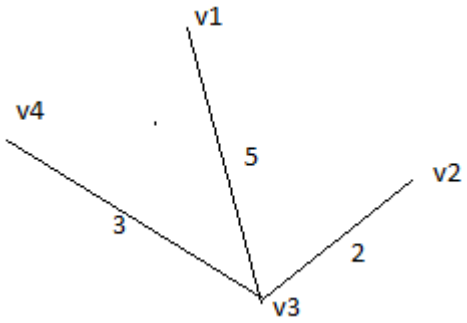
Step1: We start from the vertex v_1 and the smallest entry in the row 1 is 5 for both (v_1,v_3) and (v_1, v_4). We select (v_1, v_3) arbitrarily which results in the following subgraph:



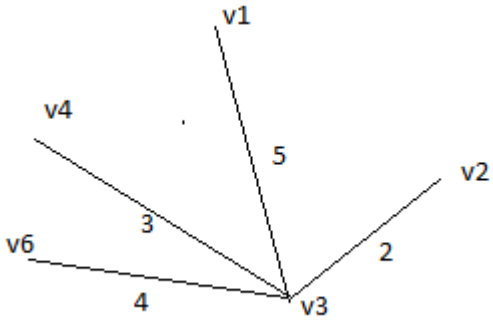
Step2: Then the smallest entry in the row1 and row3 is 2 for (v_3, v_2). We connect it to the above subgraph which results in the following:



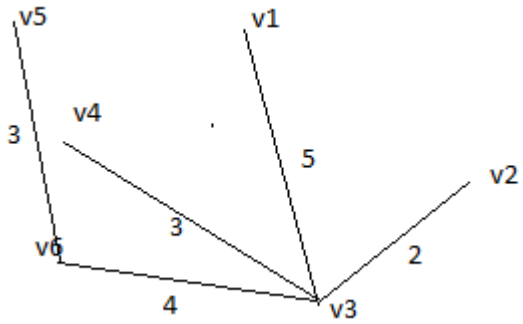
Step3: Then the smallest entry in the row1 and row3 is 3 for (v_3, v_4). We connect it to the above subgraph which results in the following:



Step4: Then the smallest entry in the row1, row3, row2 and row4 is 4 for (v_3, v_6). We connect it to the above subgraph which results in the following:



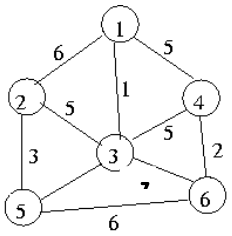
Step5: Then the smallest entry in the row1, row3, row2, row4 and row6 is 3 for (v_6, v_5). We connect it to the above subgraph which results in the following:



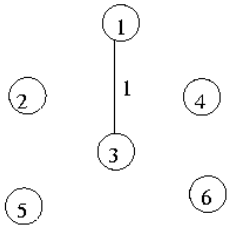
Science the number of vertices in the given graph is 6 and the subgraph in the last step contains 5(=6-1) edges, the required minimal spanning tree is given by the step 5.

Weight of the minimal spanning tree=5+2+3+4+3=17

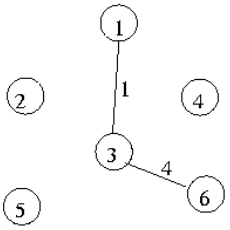
Example2: An example graph for illustrating Prim's algorithm



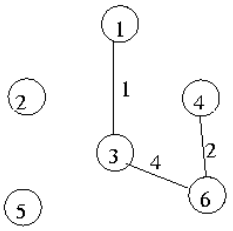
	1	2	3	4	5	6
1	–	6	1	5	∞	∞
2	6	–	5	∞	3	∞
3	1	5	–	5	7	4
4	5	∞	5	–	∞	2
5	∞	3	7	∞	–	6
6	∞	∞	7	2	6	–



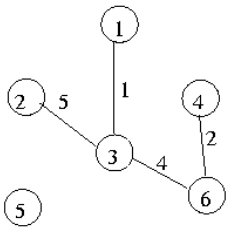
Iteration 1. U = {1}



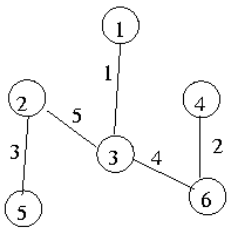
Iteration 2. U = {1,3}



Iteration 3. U = {1,3,6}



Iteration 4. U = {1,3,6,4}



Iteration 5. U = {1,3,6,4,2}

Weight of the minimal spanning tree=3+5+1+4+2=15

MULTIPLE CHOICE TYPE QUESTIONS ON MODULE

Choose the correct answer:

1. If the number of vertices and edges excluding self-loops is same then the incidence matrix of the graph is
   a) Symmetric b) identical c) square d) null

2. For a simple graph with 7 vertices and 8 edges if the 3rd row contains four 1 then

a) degree of $v_3=4$ b) degree of $v_1=4$ c) degree of $v_3=3$ d) degree of $v_3=1$

3. The degree of an isolated vertex is
a) 0 b) 1 c) 2 d) 3

4. A self-loop cannot be included in a
a) walk b) circuit c) trial d) path

5. Adjacency matrix of a graph
a) Symmetric b) Skew-Symmetric c) singular d) none of these

6. A minimally connected graph cannot have a
a) circuit b) component c) even vertex d) pendant vertex

7. A binary tree has exactly
a) two vertex of degree two b) one vertex of degree two
c) one vertex of degree one d) one vertex of degree three

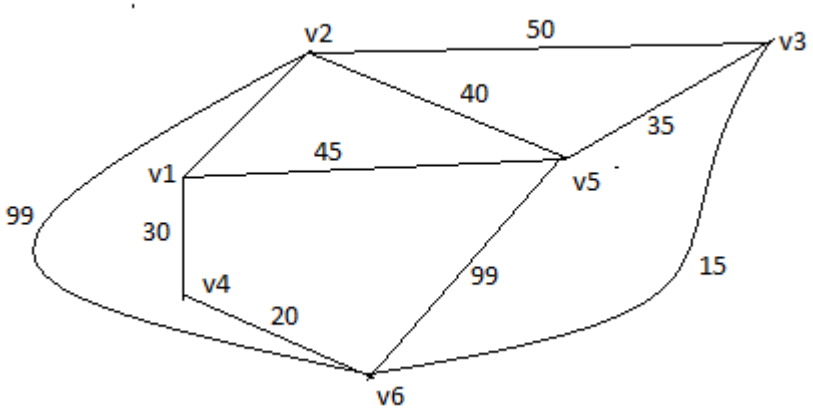
8. The minimum number of pendent vertices in a tree with five vertices is
a) 1 b) 2 c) 3 d) 4

9. A Tree is always a
a) Self-complement graph b) Euler Graph c) Simple graph
d) Hamiltonian graph

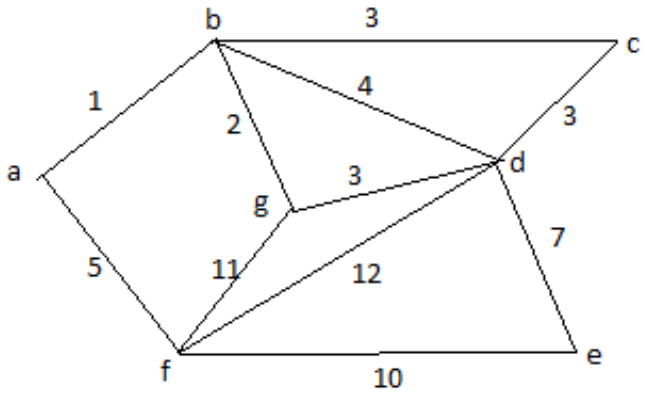
10. The number of shortest paths between two vertices in a graph
a) is always only one b) may be more than one c) is always at least one
d) none of these

PRACTICE PROBLEMS ON DESCRIPTIVE TYPE QUESTION OF MODULE

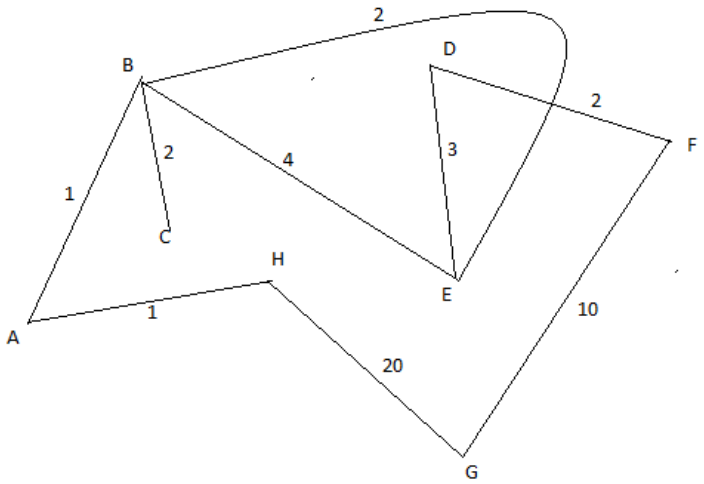
1. Suppose G is a non-directed graph with 12 edges. If G has 6 vertices each of degree 3 and the rest have degree less than 3, find the minimum number of vertices G can have.
2. Let G be a graph with n vertices and edges, then prove that G has a vertex of degree k such that k ≥ 2e/n.
3. If a simple regular graph has n vertices and 24 edges, find all possible values of n.
4. Find by Prim's Algorithm the minimum spanning tree of the following graph.



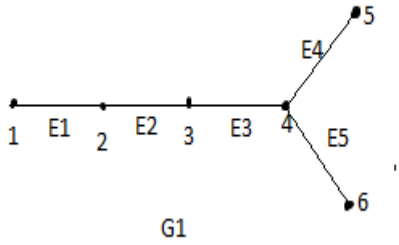
5. Find by Kruskal's Algorithm the minimum spanning tree of the following graph.



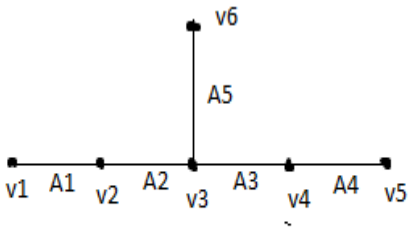
6. 5. Apply Dijkstra's Algorithm to determine a shortest path from A to G in the of the following graph.



7. Examine if the two graphs are isomorphic



G1

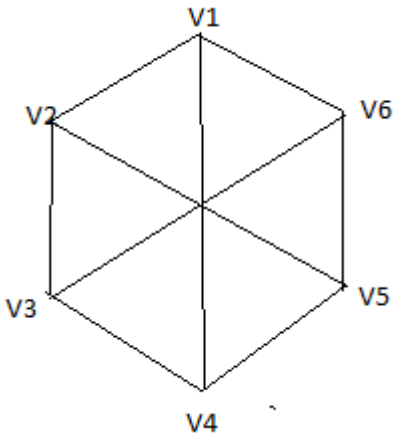


G2

8. Draw the graph whose incidence matrix is given by

$$\begin{bmatrix} 0 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

9. Find the complement of the graph



10. Find all the spanning trees in the following graph