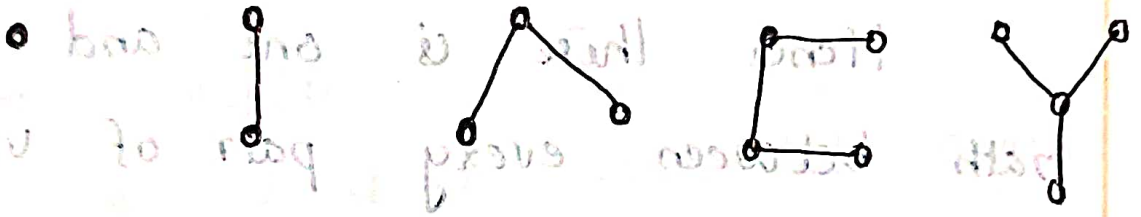
MODULE III: TREES AND GRAPH ALGORITHMS

- Trees - properties, pendant vertex
- Distance and centres in a tree
- Rooted and Binary Trees
- Counting trees
- Spanning trees
- Prim's Algorithm
- Kruskal's Algorithm
- Dijkstra's shortest path Algorithm
- Floyd - Warshall shortest path Algorithm

Trees: (T)

A tree is a connected graph without any circuits.

Trees with one, two, three and four vertices given below.



Note:

A tree has to be a simple graph that is having neither a self loop nor a parallel edge (because they both form circuits)

Some properties of Trees

Theorem 3-1

There is only one and only one path between every pair of vertices in a tree.

Proof:

Since T is a connected graph, there must exists at least one path between

every pair of vertices in T.

Now suppose that between two vertices a and b of T, there are two distinct paths. The union of these two paths will contain a circuit which contradicts that T is a tree.

Hence there is one and only one path between every pair of vertices in a tree T.

Theorem 3-2

If in a graph G there is one and only one path between every pair of vertices G is a tree.

Proof

Since there exists a path between every pair of vertices G is connected. Now to prove G is a tree we need to prove that G has no circuits.

A circuit in a graph (with two or more vertices) implies that there is at least one pair of vertices a & b such that there are two distinct paths between a and b. Since G has one and only one path between every pair of vertices, G can have no circuits.

So G is a connected acyclic graph. i, G is a tree.

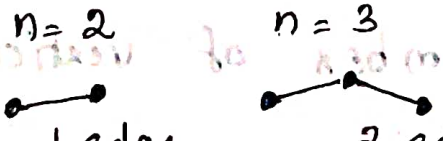
Theorem 3-3

A tree with n vertices has n-1 edges.

Proof

Let us prove the theorem by induction on number of vertices. for n = 1, 2 and 3 it is easy to see that the theorem is true.

i.



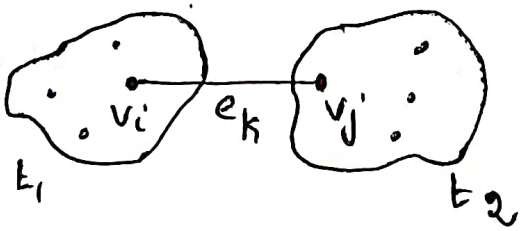
Assume that theorem holds for all trees with fewer than n vertices.

Let us now consider a tree with n vertices.

In T let e_k be an edge with end vertices v_i and v_j.

Since by theorem there is no other path between v_i and v_j except e_k.

Hence the deletion of e_k from T will disconnect the graph. Furthermore $T - e_k$ consists of exactly two components and each of these components is a tree.



Both these trees t_1 and t_2 have fewer than n vertices each and therefore by the induction hypothesis each contains one edge less than the number of

Thus $T - e_k$ consists of $n-2$ edges (and n vertices).

Hence T has exactly $n-1$ edges.

Theorem 3-4

Any connected graph with n vertices and $n-1$ edges is a tree.

Proof:

Let us assume the contradiction. Suppose G be a connected graph with n vertices and $n-1$ edges but not a tree. This means G contains at least one circuit.

Now remove an edge e from the circuit. Then $G-e$ is again a circuitless connected graph with $n-2$ edges. That is $G-e$ is a tree with n vertices & $n-2$ edges which contradicts the theorem that a graph with n vertices tree with n vertices has $n-1$ edges. Hence our assumption was wrong and G is a tree.

Theorem 3-5

A graph is a tree iff it is minimally connected.

Proof:

[Note: A graph is said to be minimally connected if removal of any edge from it disconnects the graph clearly minimally connected graph has no cycles]

Let G be a tree. i, there exists one and only one path between every pair of vertices and the removal of one edge from the path disconnects the graph. Hence G is minimally connected.

Conversely let G is a minimally connected graph. Which obviously means G has no cycles. Hence G is a tree.

Note:

We can also explain this as for a graph with n vertices we need a minimum of n-1 edges to make it connected. i, G is with n vertices & n-1 edges and hence a tree and vice versa.

Theorem 3-6

A graph with n vertices, n-1 edges and no circuits is connected.

Proof:

Suppose that there exists a circuitless graph G with n vertices and n-1 edges which is disconnected.

In that case G will consists of two or more circuitless components.

Without loss of generality, let G consists of two components g_1 and g_2

Add an edge e between a vertex v_1 in g_1 and v_2 in g_2. Since there was no path between v_1 and v_2 in G adding e did not create a circuit.

Thus G∪e is a circuitless, connected graph (ie, a tree) of n vertices and n edges., which is not possible. by theorem 3-3.

The results of the preceding 6 theorems can be summarized as follows,

A graph G with n vertices is called a tree if,

1. G is connected and is circuitless or
2. G is connected and has n-1 edges or
3. G is circuitless and has n-1 edges or
4. There is exactly one path between every pair of vertices in G or,
5. G is minimally connected graph.

All these five expressions are equivalent, definitions of a tree.

Theorem 3-7

In any tree (with two or more vertices) there are at least two pendant vertices.

Proof

In a tree of n vertices we have n-1 edges.

i.e., $\sum d(v) = 2(n-1) = 2n-2$

i.e., 2(n-1) degrees to be divided among n vertices.

Since no vertex can be of zero degree, we must have at least two vertices of degree one in a tree.

DISTANCE AND CENTERS IN A TREE

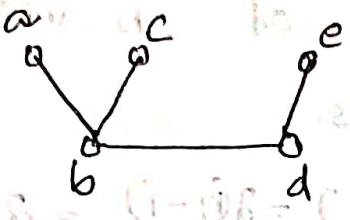
Distance in a Graph:

In a connected graph G the distance $d(v_i, v_j)$ between two of its vertices v_i and v_j is the length of the shortest path (i.e., number of edges in the shortest path) between them.

Note:

Since in a tree there is only one path between every two vertices distance is the length of the path between the vertices.

eg:



$d(a,b)=1$ $d(b,c)=1$ $d(c,d)=2$

$d(a,c)=2$ $d(b,d)=1$ $d(c,e)=3$

$d(a,d)=2$ $d(b,e)=2$

$d(a,e)=3$

Eccentricity of a vertex

The eccentricity $E(v)$ of a vertex v in a graph G is the distance from v to the vertex farthest from v in G.

i, $E(v) = \max_{v_i \in G} d(v, v_i)$

Center of a graph

A vertex with minimum eccentricity in graph G is called a center of G.

Eg: In the above graph,

$E(a) = 3$
$E(b) = 2$
$E(c) = 3$
$E(d) = 2$
$E(e) = 3$

Here vertices with minimum eccentricity are b & d
Hence b & d are centers of the graph.

Theorem 3-8

Every tree has either one or two centers.

Proof

The maximum distance, $\max d(v, v_i)$ from a given vertex v to any other vertex v_i occurs only when v_i is a pendant vertex.

Let T be a tree having more than two vertices. Then T must have two or more pendant vertices.

Delete all the pendant vertices from T. Then the resulting graph T' is still a tree.

The removal of pendant vertices from T will uniformly reduce the eccentricities of the remaining vertices by one.

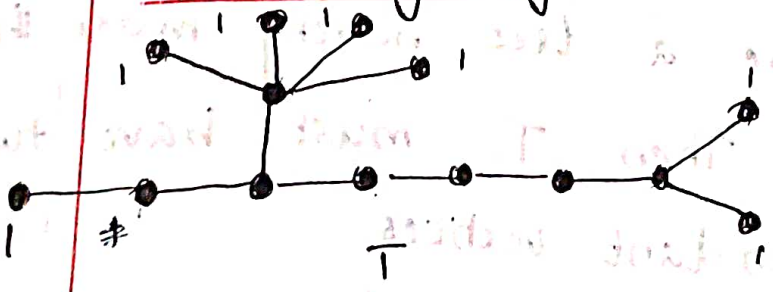
Therefore all vertices that T had as centers will still remain centers in T'.

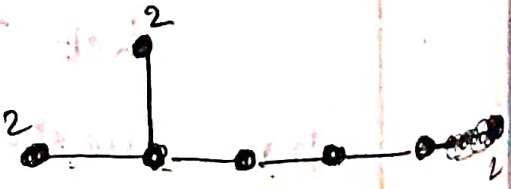
From T' we can again remove all pendant vertices and get another tree T''.

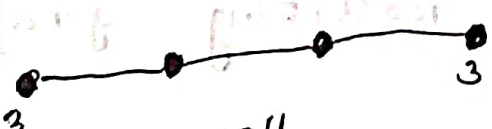
We continue this process until there is left either a vertex (which is the center of T) or an edge (whose end vertices are the two centers of T).

Hence the theorem

Illustrating eg:







centers of T

Note:

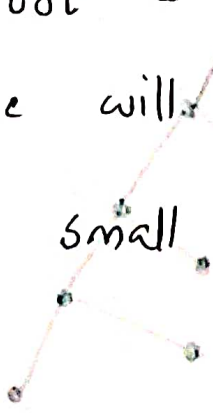
- The eccentricity of a center in a tree is defined as the radius of the tree
- The length of the longest path in T is defined as the diameter of the tree
- The radius of the tree is not necessarily half its diameter.

ROOTED AND BINARY TREES

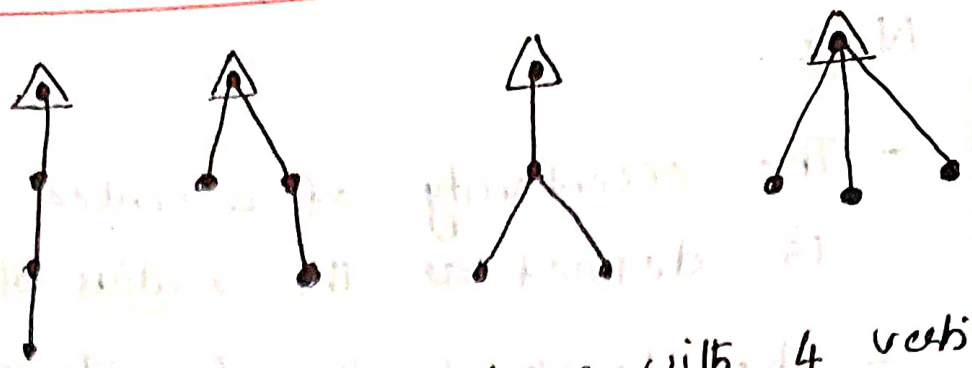
Rooted Tree

A tree in which one vertex is distinguished from all the others is called a rooted tree, and that vertex is called root of the tree.

In a diagram of a rooted tree, the root is generally marked distinctly. We will show the root enclosed in a small triangle.



Eg:



Rooted trees with 4 vertices.

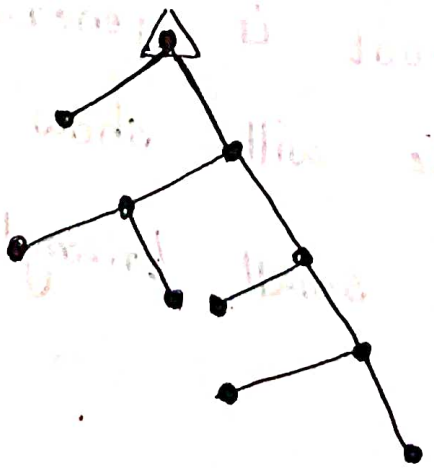
Binary Trees:

A binary tree is defined as a tree in which there is exactly one vertex of degree two, and each of the remaining vertices is of degree one or three.

Since the vertex of degree 2 is distinct from all the other vertices, this vertex serves as a root.

Thus every binary tree is a rooted tree.

Eg:



Properties of Binary Trees:

1. The number of vertices n in a binary tree is always odd.

In a binary tree exactly one vertex is of even degree (vertex with degree 2)

The remaining $n-1$ vertices are of odd degree (degree one or three)

Since by theorem that the number of odd vertices in a graph is even $n-1$ should be even. i, n is odd.

---

2. Let p be the number of pendant vertices in a Binary tree. T. Then $n-p-1$ is the number of vertices of degree 3. Therefore number of edges in T equals. For we have, $2e = \sum d(v_i)$ (tree, $e = n-1$)

i, $2(n-1) = p \times 1 + (n-p-1) \times 3 + 2$

$2n-2 = p + 3n - 3p - 3 + 2 = -2p + 3n - 1$

i, $2p = 3n - 1 - 2n + 2 = n + 1$

i, $p = \frac{n+1}{2}$

$\therefore$ No. of pendant vertices in T $= \frac{n+1}{2}$ //

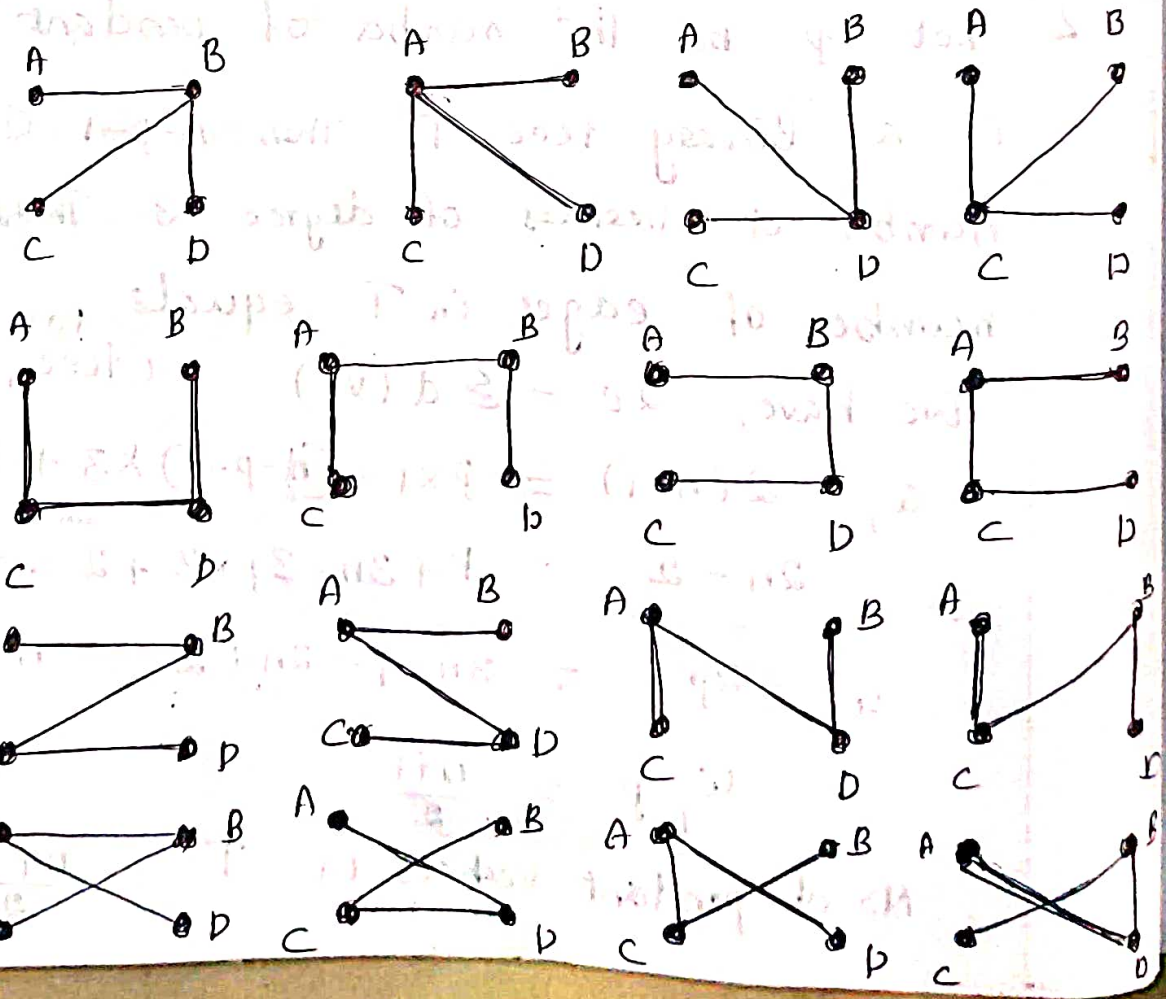
Counting Trees

Question is what is the number of different trees that one can construct with n distinct vertices.

labelled graph

A graph in which each vertex is assigned a unique name or label is called labelled graph otherwise unlabelled.

If the tree with 4 vertices is labelled we can count 16 trees. (eventhough some of them are isomorphic.



But if the vertices are unlabelled it can be verified that there are only two distinct trees with 4 vertices.

Result:

The number of labelled trees with n vertices $(n \geq 2)$ is n^{n-2}

Spanning Trees

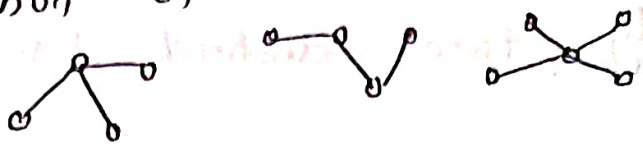
A tree T is said to be a spanning tree of a connected graph G if T is a subgraph of G and it contains all vertices of G.

Spanning tree is sometimes refered to as a skeleton or scaffolding of G.

Since spanning trees are the largest (with maximum number of edges) trees among all trees of G, it is also called maximal tree subgraph of G or maximal

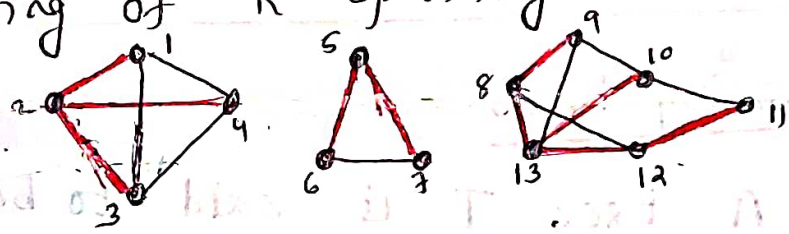
Forest

A collection of trees is called Forest

eg: 

Note:

Each component of a disconnected graph does have a spanning tree. Thus a disconnected graph with k components has a spanning forest consisting of k spanning trees.

Eg: 

Finding a Spanning Tree in a connected graph

If a graph G has no circuits it is its own spanning tree.

If G has a circuit delete an edge from the circuit. This will leave the graph still connected.

If there are more circuits repeat the operation till there remains no circuits. Hence it result in a connected circuit free graph containing all vertices in G.

Theorem 3-9

Every connected graph has at least one spanning tree.

The above explanation for finding the spanning tree in a graph can be used for proof.

Branch of a Tree

Any edge in a spanning tree is called a branch of of a tree.

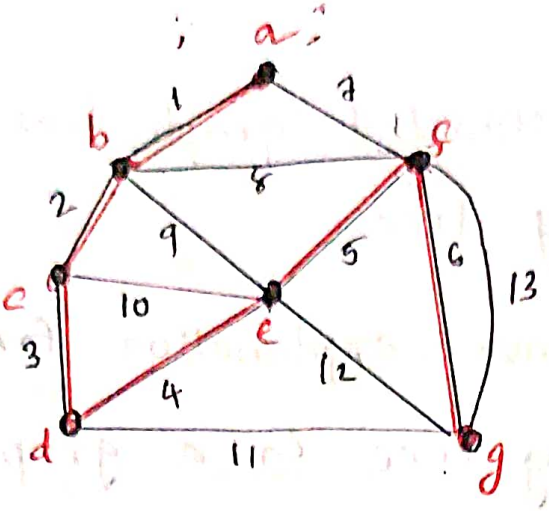
Chord

An edge of G that is not in a given spanning tree is called a chord.

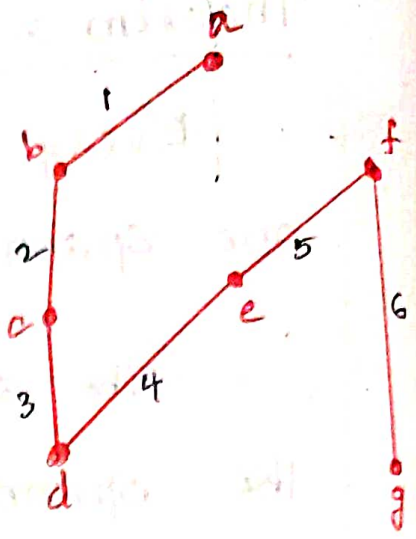
Notes:

- Branches and chords are defined only with respect to a spanning tree.
- An edge that is a branch of one spanning tree may be a chord with respect to another spanning tree.
- A connected graph G can be considered as the union of two subgraphs, $T \oplus \overline{T}$ where T is a spanning tree $\overline{T}$ is the compliment of T in G

Eg: Graph G



spanning tree



Here $\{1, 2, 3, 4, 5, 6\}$ are branches of the given spanning tree while edges $\{7, 8, 9, 10, 11, 12, 13\}$ are chords with respect to the spanning tree.

Theorem 3-10

With respect to any of its spanning trees, a connected graph of n vertices and e edges has n-1 tree branches and e-n+1 chords

Proof

Since the spanning tree is a tree with n vertices it has n-1 edges. (by theorem 3-3)
i, n-1 branches.

Also if T is a spanning tree $G = T \cup \bar{T}$ where $\bar{T}$ is the compliment of T in G.

The edges of $\bar{T}$ are chords with respect to the given spanning tree.

∴ The number of chords

$= e - (n-1)$

(total edges − no. of edges in T)

$= e - n + 1$

Note:

In a graph G with n vertices, e edges, and k components the numbers called rank and nullity are given by

rank, $r = n - k$

nullity $\quad k_e = e - n + k$

rank in G = number of branches of spanning tree of G

Nullity of G = number of chords in G

rank + nullity = numbers of edges in G

Spanning Trees in a Weighted graph

A spanning tree in a graph G is a minimal subgraph connecting all the vertices of G.

If graph G is a weighted graph, then the weight of a spanning tree T of G is defined as the sum of the weights of all the branches in T.

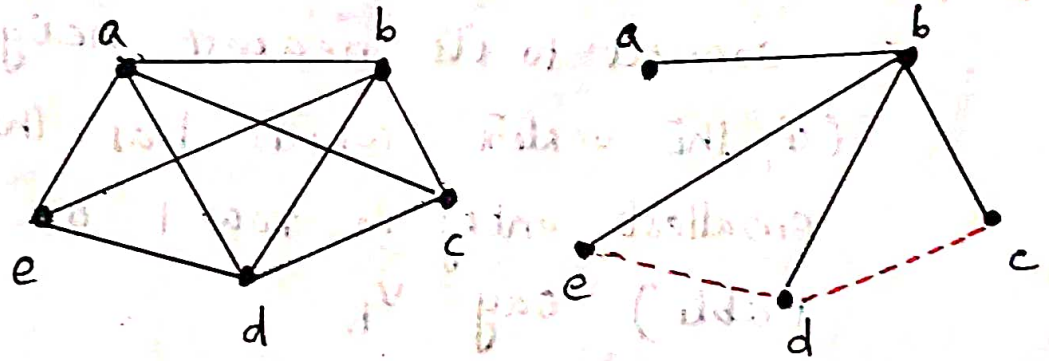
Shortest spanning Tree or Minimal Spanning Tree.

A spanning tree with the smallest weight in a weighted graph is called a minimal spanning tree or shortest-distance spanning tree.

Fundamental Circuits

Consider a spanning tree T in a connected graph G. Adding any one chord (an edge of G that is not in a given spanning tree) to T will create exactly one circuit. Such a circuit formed by adding a chord to a spanning tree is called a fundamental circuit.

Example: G: T.



Adding the chord (c,d) to the spanning tree T will form a fundamental circuit bcdb

Adding the chord (d,e) to the spanning tree T will creat a fundamental circuit bdeb

But adding both the chords will create a circuit bcdea also which is not a fundamental circuit.

Algorithms for shortest spanning Tree

I Prim's Algorithm

Let G be a weighted graph with n vertices.

Step I : Tabulate the given weights of the edges of G in an nxn table.

Step II : Set the weights of nonexistent edges as very large. (say ∞)

Step III : Start from a vertex v_1 and connect to its nearest neighbour (i, the vertex which has the smallest entry in row 1 of the table) say v_k

Step IV : Now consider v_1 & v_k as one subgraph and connect this subgraph to its closest neighbour (i, to a vertex other than v_1 & v_k that has the smallest entry among all entries in row 1 and row k). Let this vertex be v_i.

Step V: Now regard the tree with vertices v_1, v_k & v_i as one subgraph and continue this process untill all vertices have been connected by (n-1) edges.

Example:

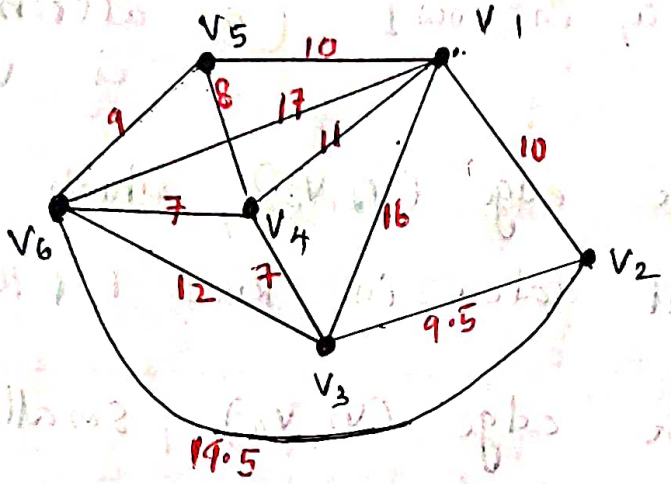


Table:

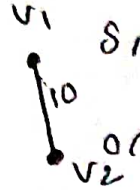
	V_1	V_2	V_3	V_4	V_5	V_6
V_1	—	10	16	11	10	17
V_2	10	—	9.5	∞	∞	19.5
V_3	16	9.5	—	7	∞	12
V_4	11	∞	7	—	8	7
V_5	10	∞	∞	8	—	9
V_6	17	19.5	12	7	9	—

Give step by step figures.

(Note: The entries in the table are symmetric with respect to the diagonal and the diagonal is empty)

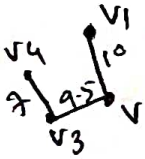
I: Start from a vertex say V_1.

II: Choose edge (V_1, V_2) which has the smallest weight among edges incident on V_1, as in Row I. (an alternate choice is V_1V_5)



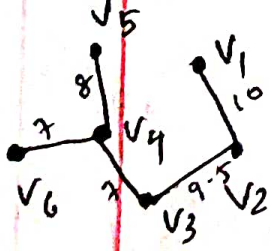
III: Choose edge (V_2, V_3), which has smallest value in Row 1 & Row 2

IV: Choose edge (V_3, V_4), smallest weight in Row 1, Row 2 and Row 3.



V: Choose edge (V_4, V_6), smallest weighted edge in Row 1, Row 2, Row 3 & Row 4

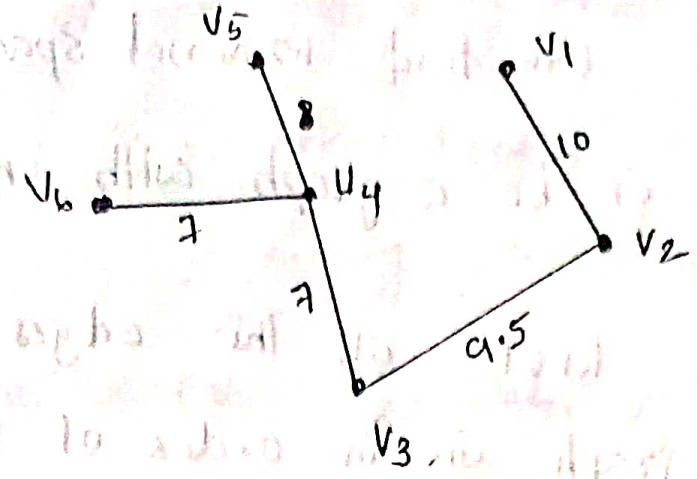
VI: Choose edge (V_4, V_5), smallest-weighted edge in Row 1, 2, 3, 4 & 5.



(Each step we have to select a edge which is not already selected)

VII: Stop the process as 5 edges are selected in the graph with 6 vertices.

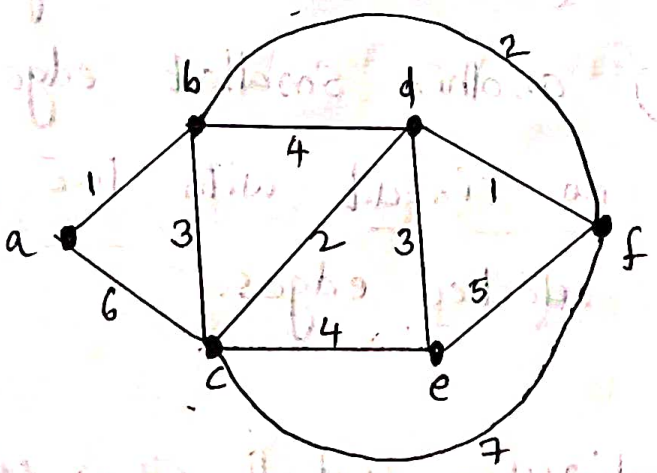
The spanning tree obtained is,



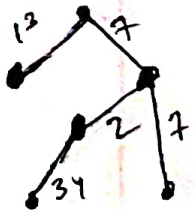
Total weight of the spanning tree

$= 10 + 9.5 + 7 + 7 + 8 = 41.5$, which is the smallest weight that corresponds to the minimal spanning tree.

Qn 1)

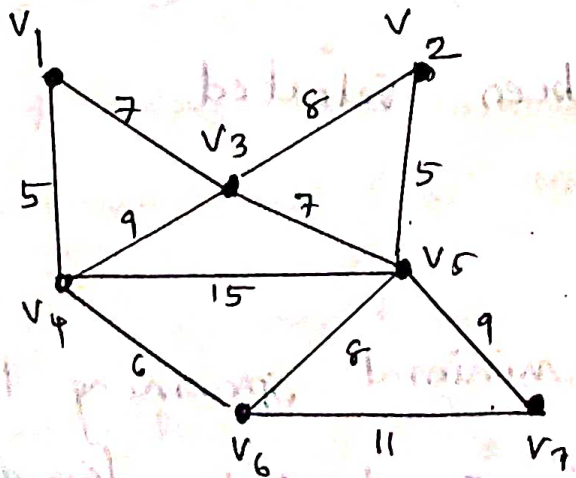


(HW)

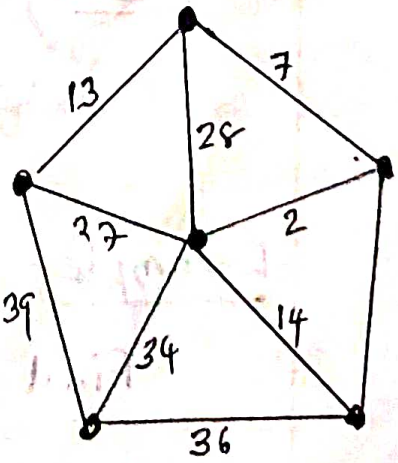


63

2)



(3)



II

Kruskal's Algorithm

(to find minimal spanning tree)

Let G be a graph with n vertices.

Step I : List all the edges of the graph G in order of nondecreasing weights.

Step II : Select the smallest edge of G

Step III : Now for each successive step select (from all remaining edges of G) another smallest edge, that makes no circuit with the previously selected edges.

Step IV : Continue until (n-1) edges have been selected.

Example:

Find the minimal spanning tree in the following graph using Kruskal's algorithm.

We will illustrate the same problem given as example in Prim's algorithm.

Step I: listing the edges in order of non decreasing weights we get,

$(v_4, v_3) \leq (v_4, v_6) \leq (v_4, v_5) \leq (v_5, v_6) \leq$
$(v_3, v_2) \leq (v_1, v_2) \leq (v_1, v_5) \leq (v_1, v_4) \leq$
$(v_3, v_6) \leq (v_1, v_3) \leq (v_1, v_6) \leq (v_2, v_6)$

$(7 \leq 7 \leq 8 \leq 9 \leq 9.5 \leq 10 \leq 10 \leq 11 \leq 12 \leq 16 \leq 17 \leq 19.5)$

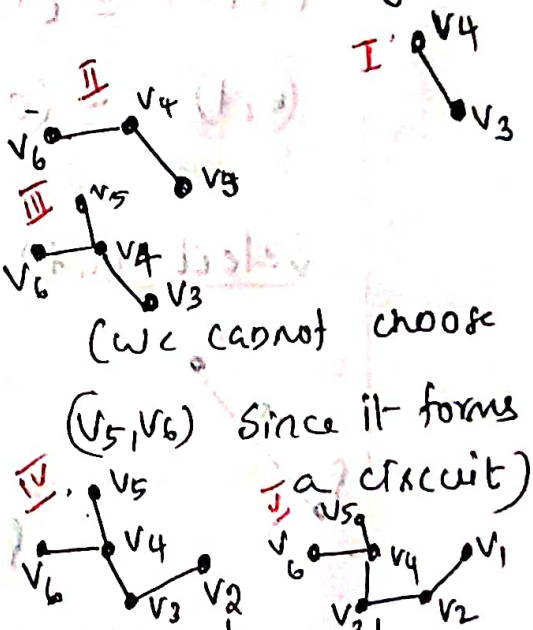
choose edge (v_4, v_3) – smallest weight

" (v_4, v_6) –

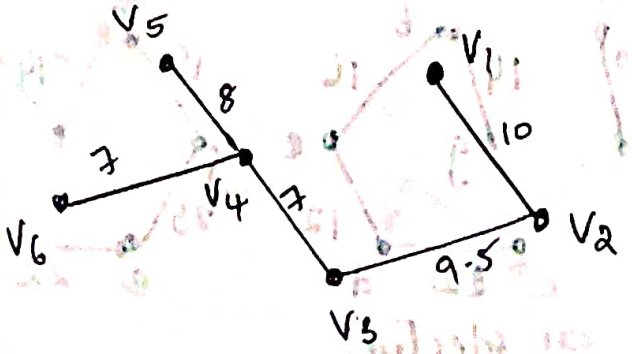
" (v_4, v_5)

" (v_3, v_2) (we cannot choose (v_5, v_6) since it forms a circuit)

" (v_1, v_2)

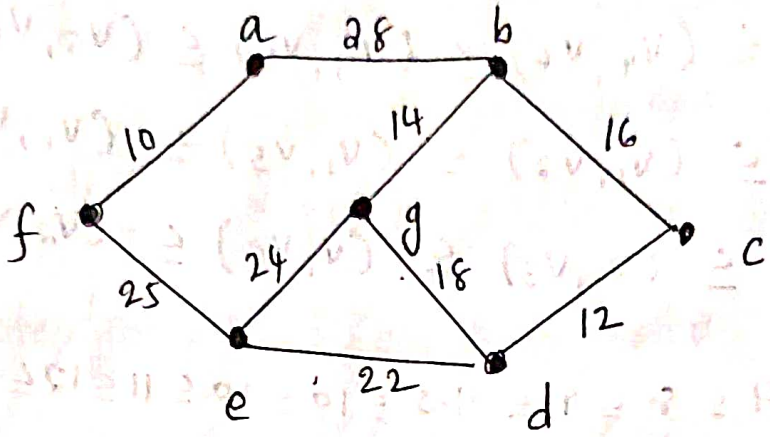


5 edges of smallest weights have been selected and the minimal spanning tree is



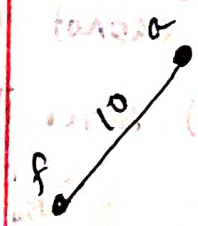
total weight =
$10 + 9.5 + 7 + 7 + 8 = 41.5$

Use kruskal's algorithm to find the shortest spanning tree in the following graph. Also use Prim's algorithm.

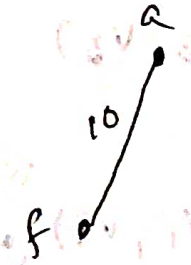


$(a,f) \leq (c,d) \leq (b,g) \leq (b,c) \leq (d,g) \leq (e,d) \leq (e,g) \leq (e,f) \leq (a,b)$

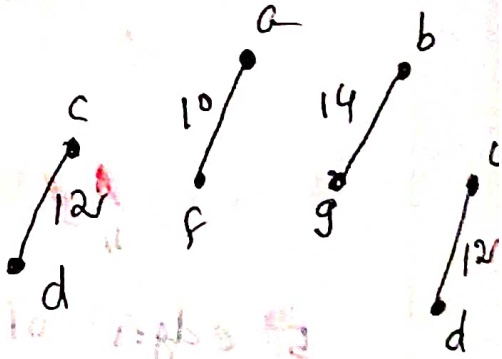
Select (a,f)



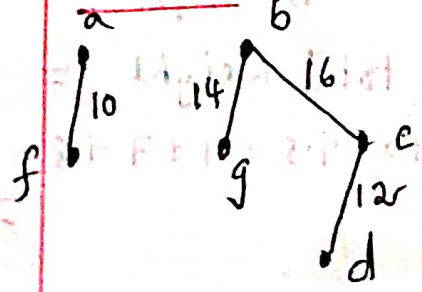
Select (c,d)



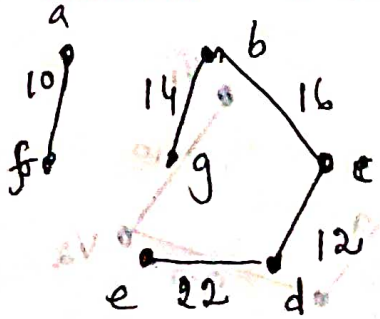
Select (b,g)



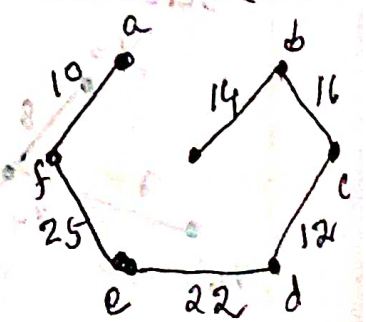
Select (b,c)



Select (e,d)

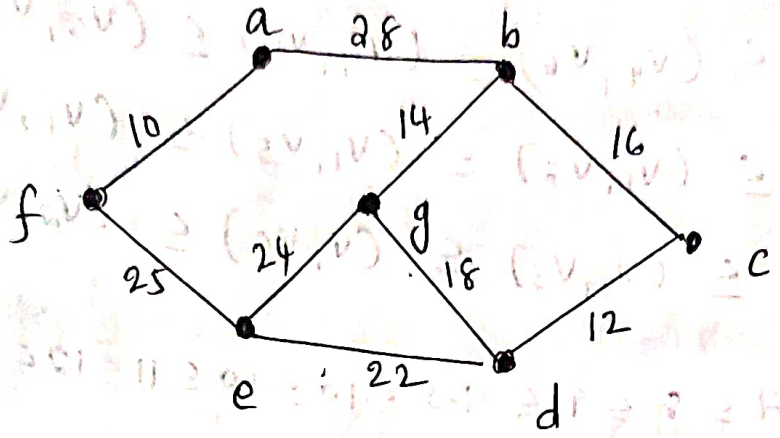


Select (e,f)



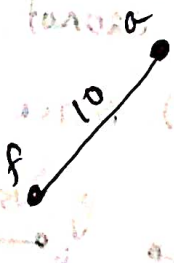
$n-1 = 7-1 = 6$ edges are selected.

Use kruskal's algorithm to find shortest spanning tree in the following graph. Also use Prim's algorithm.



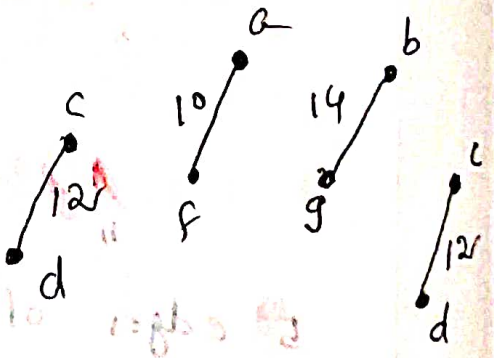
$(a,f) \leq (c,d) \leq (b,g) \leq (b,c) \leq (d,g) \leq (e,d) \leq (e,g) \leq (e,f) \leq (a,b)$

Select (a,f)

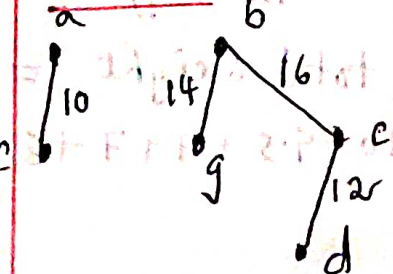


Select (c,d)

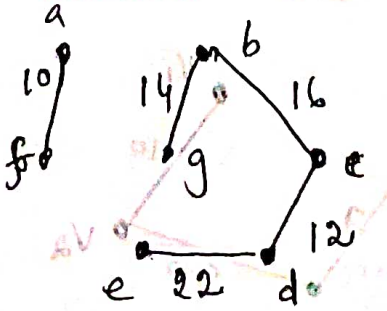
Select (b,g)



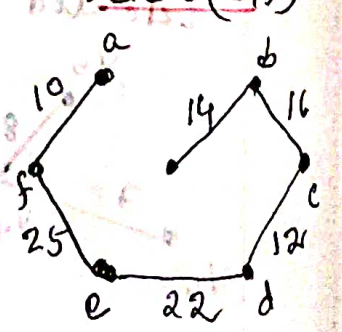
Select (b,c)



Select (e,d)

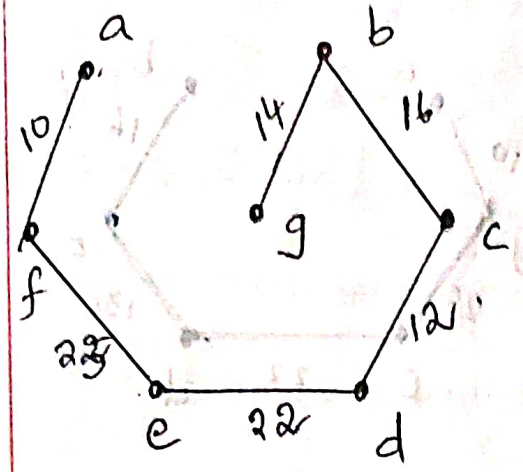


Select (e,f)



$n-1 = 7-1 = 6$ edges are selected.

Hence the spanning tree is,



Total weight (minimum weight) = 16+12+22+25+10+14

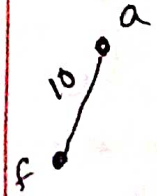
= 99

Prims Algorithm

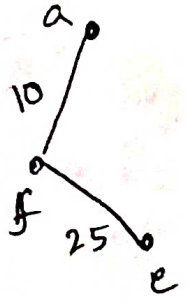
Table

	a	b	c	d	e	f	g
a	-	28	∞	∞	∞	10	∞
b	28	-	16	∞	∞	∞	14
c	∞	16	-	12	∞	∞	∞
d	∞	∞	12	-	22	∞	18
e	∞	∞	∞	22	-	25	24
f	10	∞	∞	∞	25	-	∞
g	∞	14	∞	18	24	∞	-

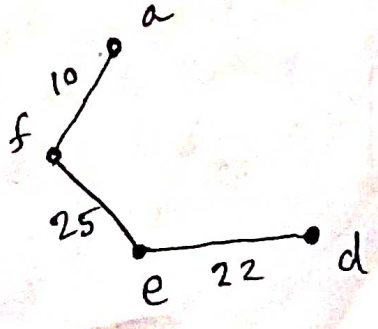
Select (a, f)



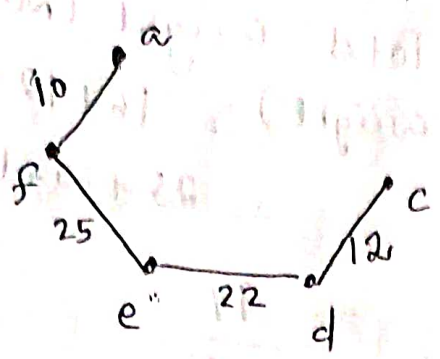
Select (f, e) (from row a & row f)



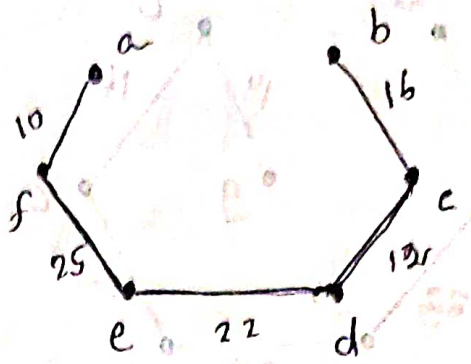
Select (e, d) (from row a row f & row e)



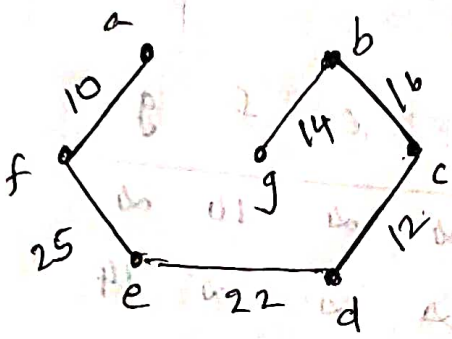
Select (d, c) from rows of a, f, e, d



Select c, b from rows of a, f, e, d & c

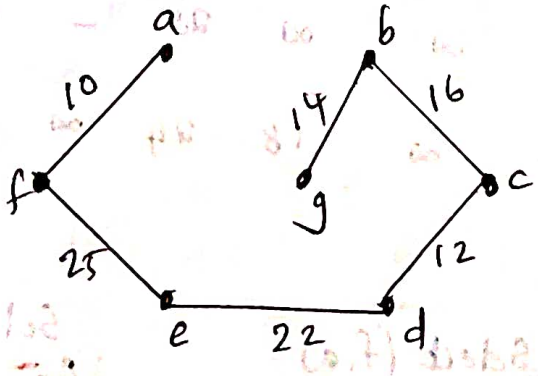


Select (b, g)



Selected all $n-1 = 7-1 = 6$ edges.

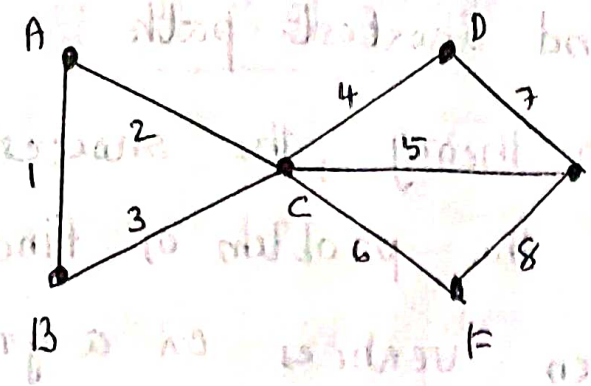
Hence the spanning tree with minimal weight is



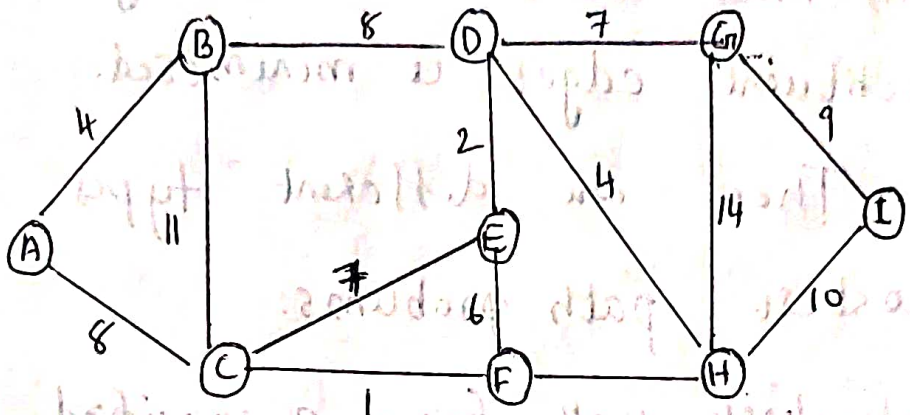
i, $10+25+16+14+12+22 = 99$

(HW)

1)



2)



Application of shortest spanning Tree.

Suppose that we are to connect n cities $v_1, v_2, \ldots v_n$ through a network of roads. The cost c_{ij} of building a direct road between v_i & v_j are given for all pairs of cities where road can be built. The problem is then to find the least expensive network that connects all n cities together. It is evident that this connected network be a tree. Thus the problem of connecting n cities with a least expensive network is the problem of finding a shortest spanning tree in a connected weighted graph of n vertices

Algorithms to find shortest path

In Graph theory, the shortest path problem is the problem of finding a path between vertices in a graph such that sum of the weights of its constituent edges is minimised.

There are different types of shortest path problems.

- shortest path from a specified vertex to another specified vertex — Dijkstras Algorithm (or from a specified vertex to every other vertices)
- Shortest path between all pairs of vertices - Floyd-Warshall Algorithm.

(From a vertex to every other vertices)

Dijkstra's Shortest path Algorithm

The problem of finding the shortest path from a specified vertex 's' to another specified vertex 't' can be stated as follows.

A simple weighted digraph G of n vertices is described by an n x n matrix $D = [d_{ij}]$, where,

d_{ij} = length (or distance or weight) of the directed edge from vertex i to vertex j, $d_{ij} \geq 0$

$d_{ii} = 0$

$d_{ij} = \infty$, if there is no edge from vertex i to vertex j

Step I :

Assign a permanent label '0' to the starting vertex s, and a temporary label ∞ to the remaining (n-1) vertices.

Step II :

Every vertex j that is not yet

permanently labelled, gets a new temporary label whose value is given by

min { old label of j, (old label of i + dij) }

where i is the latest vertex permanently labelled, in the previous iteration and dij is the direct distance between vertices i & j. If i and j are not joined by an edge then dij = ∞.

Step III

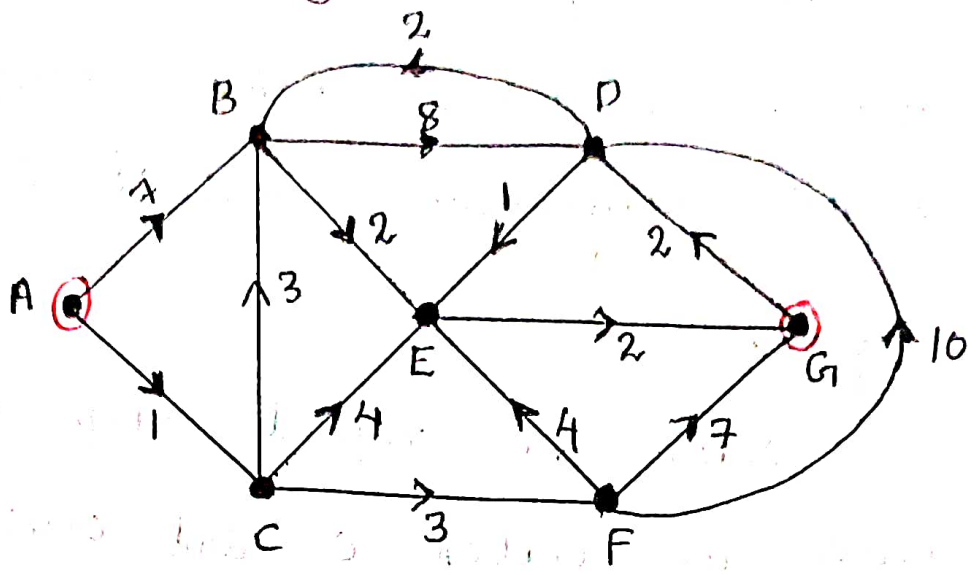
The smallest value among all temporary labels is found, and this becomes the permanent label of the corresponding vertex. In case of a tie select any one of the vertex for permanent labelling.

Step IV

Step II and III are repeated alternately until the destination vertex t gets a permanent label.

Problem:

1) Find a shortest path from vertex A to G in the following graph using Dijkstra's Algorithm.



Step I

Assign label '0' to vertex A and all the other vertices are labelled '∞'.

Step II

Now assign label (0+7 =7) 7 to vertex B & label 0+1 = 1 to the vertex C, which are adjacent to vertex A by a directed edge.

i, each vertex C & B are labeled by assigning min{old label of C/B, old label of C/B + distance of AC or AB correspondingly }

Step III

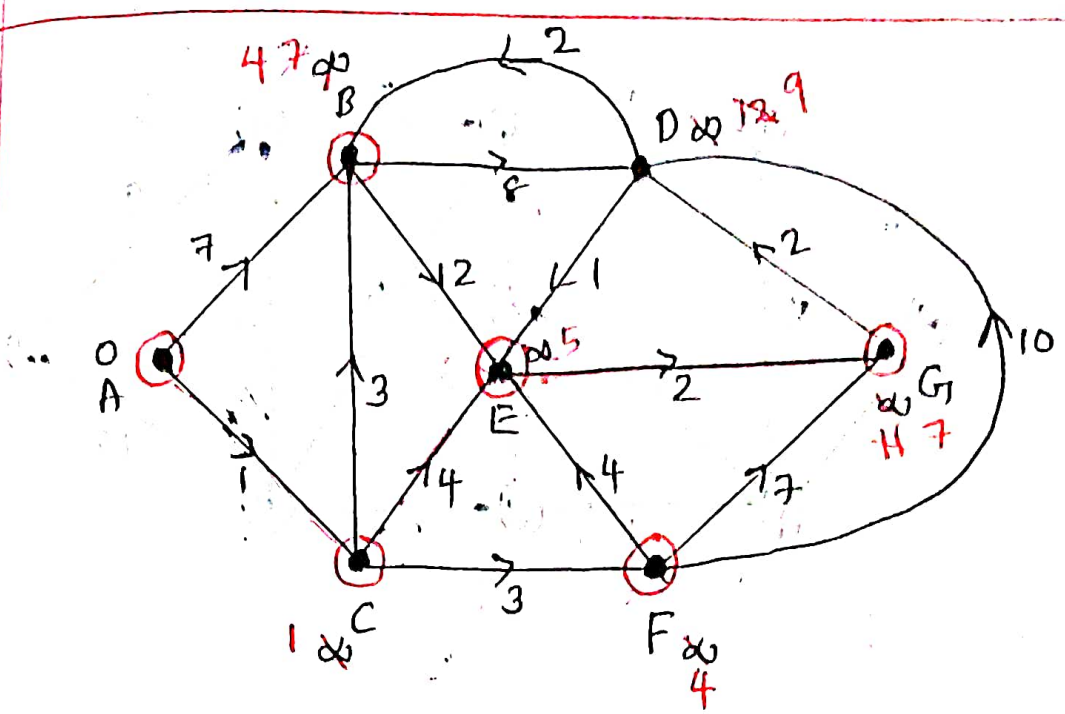
Now the smallest value among all the temporary labelled vertices is that of C (ie 1). Now assign '1' as the permanent label of C.

Step IV

Now we repeat step to from the vertex C and continue this till the vertex G is permanently labelled.

The corresponding labelling can be given by the following table.

A	B	C	D	E	F	G
0	∞	∞	∞	∞	∞	∞
0	7	1	∞	∞	∞	∞
0	4	1	∞	5	4	∞
0	4	1	12	5	4	∞
0	4	1	12	5	4	11
0	4	1	12	5	4	7
0	4	1	9	5	4	7

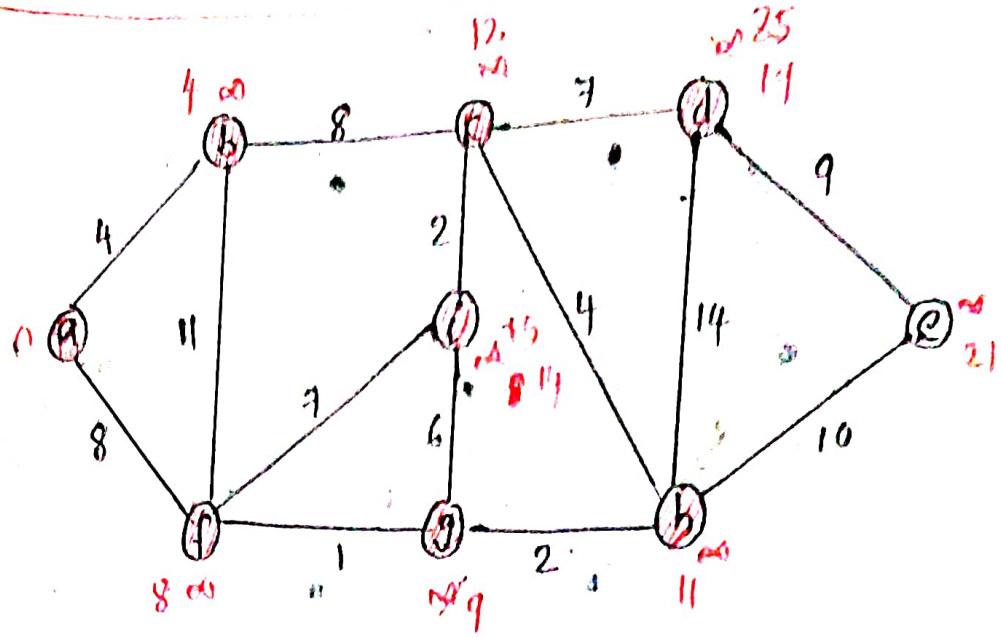


Hence the length of shortest path from A to G is 7

Notes:

- If the given digraph is not simple, it can be simplified by discarding all self-loops, and replacing every set of parallel edges by the shortest weighted edge among them.
- It is also applied to undirected graphs also.
- If the graph is not weighted assume $d_{ij} = 1$ (adjacency matrix becomes distance matrix).

(2)

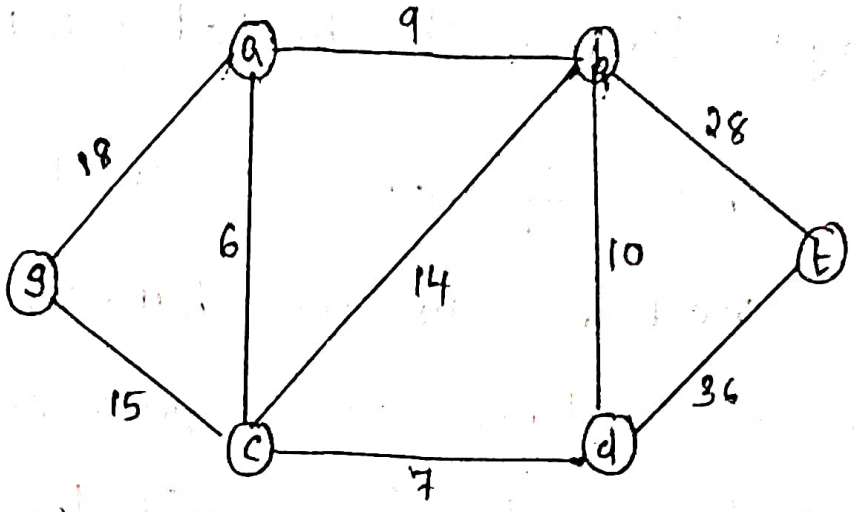


a	b	c	d	e	f	g	h	i
[0]	∞	∞	∞	∞	∞	∞	∞	∞
[0]	[4]	∞	∞	∞	8	∞	∞	∞
[0]	[4]	12	∞	∞	[8]	∞	∞	∞
[0]	[4]	12	∞	∞	[8]	[9]	∞	15
[0]	[4]	12	∞	∞	[8]	[9]	[11]	15
[0]	[4]	[12]	25	21	[8]	[9]	[11]	15
[0]	[4]	[12]	19	21	[8]	[9]	[11]	[14]
[0]	[4]	[12]	[19]	21	[8]	[9]	[11]	[14]
[0]	[4]	[12]	[19]	[21]	[8]	[9]	[11]	[14]

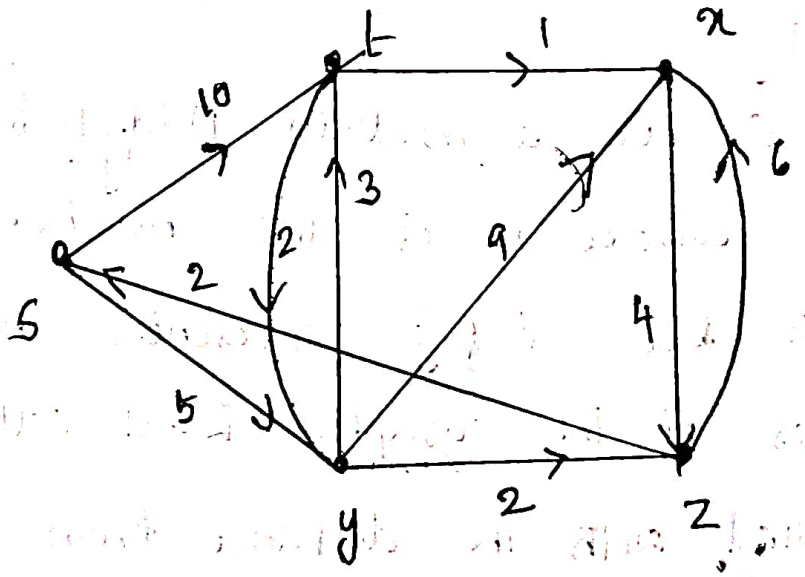
The last row gives the distance of shortest path from (a) to all the other vertices.

HW

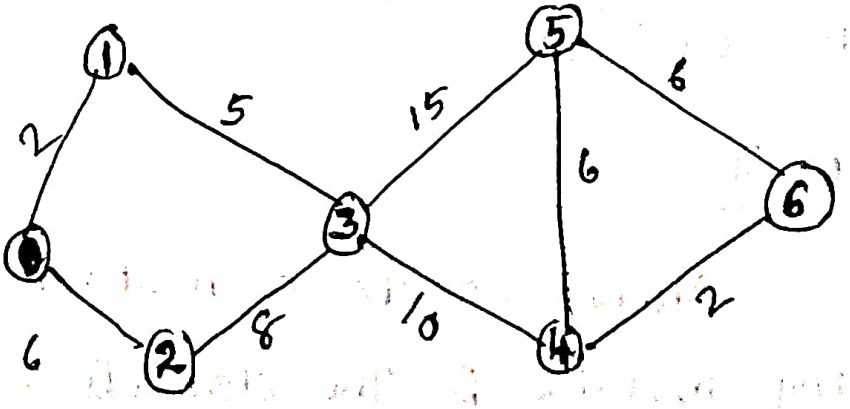
(3)


a
b
c
d
e
g
9
18
6
14
10
28
15
36
7

(4)


t
x
y
z
s
1
10
3
6
2
9
2
4
5
2
2

5)


1
2
3
4
5
6
2
5
15
6
6
6
8
10
2

Floyd - Warshall's shortest path Algorithm

This algorithm is used to find shortest distance between every pair of vertices in a given weighted directed graph. (in which the weights can be positive or negative.)

Step I

Create a matrix $D=[d_{ij}]$ of dimension $n \times n$ where n is the number of vertices. let i & j represents the vertices of the graph. Each cell dij is filled with the distance from ith vertex to the jth vertex. If there is no path from i to j the cell is left as '∞'

Step II:

Now create a matrix D_1 using matrix D. The elements in the first row and first column are left as

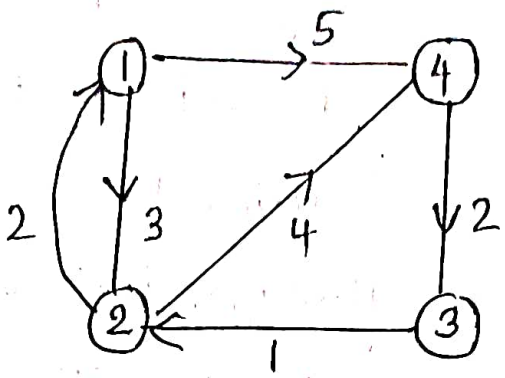
they are. The remaining cells are filled as follows. Let k be the intermediate vertex in the shortest path from source to destination. dij is filled with $(d_{ik} + d_{kj})$ if $d_{ij} > (d_{ik} + d_{kj})$ i.e., if the direct distance from the source to the destination is greater than the path through the vertex k then the cell is filled with $d_{ik} + d_{kj}$

Step III

Now create D_2 similarly from D'. The elements in the second column and second row are left as they are. Now proceed the steps as in step 2 with k be the second intermediate vertex.

Continue this process creating the matrices $D_1, D_2, \cdots D_n$ by considering all the vertices of the graph as intermediate vertices.

Q.) Find the shortest path between every pair of vertices in the following graph.



Step I

Let,

$$D = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 3 & \infty & 5 \\ 2 & 0 & \infty & 4 \\ \infty & 1 & 0 & \infty \\ \infty & \infty & 2 & 0 \end{bmatrix} \end{array}$$

Step 2

Create matrix D_1 from D.

Take (1) as the intermediate vertex

$$D_1 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 3 & \infty & 5 \\ 2 & 0 & \infty & 4 \\ \infty & 1 & 0 & \infty \\ \infty & \infty & 2 & 0 \end{bmatrix} \end{array}$$

Row 1 & colum 1 remains same

$d_{23} = \infty$, $d_{21} + d_{13} = 2 + \infty$

$d_{24} = 4$, $d_{21} + d_{14} = 2 + 5 = 7$

$d_{32} = 1$, $d_{31} + d_{12} = \infty + 3$

$d_{34} = \infty$, $d_{31} + d_{14} = \infty + 5$

$d_{42} = \infty$, $d_{41} + d_{12} = \infty + 3$

$d_{43} = 2$, $d_{41} + d_{13} = \infty + \infty$

Step 3:

Create matrix D_2 from D_1

Take ② as the intermediate vertex and row and column remains same.

$$D_2 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 3 & \infty & 5 \\ 2 & 0 & \infty & 4 \\ 3 & 1 & 0 & 5 \\ \infty & \infty & 2 & 0 \end{bmatrix} \end{array}$$

$d_{13} = \infty,\ d_{12} + d_{23} = 3 + \infty$

$d_{14} = 5,\ d_{12} + d_{24} = 3 + 4 = 7$ $(5 < 7)$

$d_{31} = \infty,\ d_{32} + d_{21} = 1 + 2 = 3$

$d_{34} = \infty\ d_{32} + d_{24} = 1 + 4 = 5$

$d_{41} = \infty,\ d_{42} + d_{21} = \infty + 2$

$d_{43} = 2,\ d_{42} + d_{23} = \infty + \infty$

Step 4:

Create matrix D_3 from D_2

Take ③ as intermediate vertex.

Row 3 & column 3 remains same.

$$D_3 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 3 & \infty & 5 \\ 2 & 0 & \infty & 4 \\ 3 & 1 & 0 & 5 \\ 5 & 3 & 2 & 0 \end{bmatrix} \end{array}$$

$d_{12} = 3,\ d_{13} + d_{32} = \infty + 1$

$d_{14} = 5,\ d_{13} + d_{34} = \infty + 5$

$d_{21} = 2,\ d_{23} + d_{31} = \infty + 3$

$d_{24} = 4,\ d_{23} + d_{34} = \infty + 5$

$d_{41} = \infty,\ d_{43} + d_{31} = 2 + 3 = 5$

$d_{42} = \infty,\ d_{43} + d_{32} = 2 + 1 = 3$

Step 4

Create matrix D_4 from D_3

Take vertex (4) as intermediate vertex

Row 4, Column 4 remains unchanged

$$D_4 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 3 & 7 & 5 \\ 2 & 0 & 6 & 4 \\ 3 & 1 & 0 & 5 \\ 5 & 3 & 2 & 0 \end{bmatrix} \end{array}$$

$d_{12} = 3$, $d_{14} + d_{42} = 5 + 3 = 8$

$d_{13} = \infty$ $d_{14} + d_{43} = 5 + 2 = 7$

$d_{21} = 2$, $d_{24} + d_{41} = 4 + 5 = 9$

$d_{23} = \infty$ $d_{24} + d_{43} = 4 + 2 = 6$

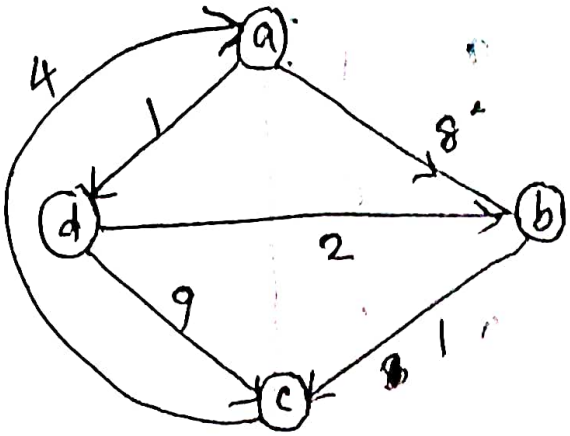
$d_{31} = 3$, $d_{34} + d_{41} = 5 + 5 = 10$

$d_{32} = 1$, $d_{34} + d_{42} = 5 + 3 = 8$

The entry in the ij^{th} cell gives the shortest distance from vertex i to vertex j.

Hence from D_4 we get the shortest distance ~~from~~ between every pair of vertices in the given graph.

(a)



$$D = \begin{array}{c} \\ a \\ b \\ c \\ d \end{array} \begin{array}{c} \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{bmatrix} 0 & 8 & \infty & 1 \\ \infty & 0 & 1 & \infty \\ 4 & \infty & 0 & \infty \\ \infty & 2 & 9 & 0 \end{bmatrix} \end{array}$$

$$D_1 = \begin{array}{c} \\ a \\ b \\ c \\ d \end{array} \begin{array}{c} \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{bmatrix} 0 & 8 & \infty & 1 \\ \infty & 0 & 1 & \infty \\ 4 & 12 & 0 & 5 \\ \infty & 2 & 9 & 0 \end{bmatrix} \end{array}$$

$(bc) = 1, \quad (ba) + (ac) = \infty$

$(bd) = \infty, \quad ba + ad = \infty$

$(cb) = \infty \quad (ca + ab) = 4 + 8 = 12$

$$D_2 = \begin{array}{c} \\ a \\ b \\ c \\ d \end{array} \begin{array}{c} \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{bmatrix} 0 & 8 & 9 & 1 \\ \infty & 0 & 1 & \infty \\ 4 & 12 & 0 & 5 \\ \infty & 2 & 9 & 0 \end{bmatrix} \end{array}$$

$$D_3 = \begin{array}{c} \\ a \\ b \\ c \\ d \end{array} \begin{array}{c} \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{bmatrix} 0 & 8 & 9 & 1 \\ 5 & 0 & 1 & 6 \\ 4 & 12 & 0 & 5 \\ 8 & 2 & 9 & 0 \end{bmatrix} \end{array}$$

$$D_4 = \begin{array}{c} \\ a \\ b \\ c \\ d \end{array} \begin{array}{c} \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{bmatrix} 0 & 3 & 9 & 1 \\ 5 & 0 & 1 & 6 \\ 4 & 7 & 0 & 5 \\ 8 & 2 & 9 & 0 \end{bmatrix} \end{array}$$

a d + d b

The matrix D gives the lengths of the shortest paths between every pair of vertices of the graph.