Unit-4 Graph Theory

Basic Terminology-

A graph G consists of two sets:

(i) A non-empty set V whose elements are called vertices, nodes or point of G.

The set V(G) is called the vertex set of G.

(ii) A set E of edges such that each $e \in E$ associated with Ordered or unordered pairs of elements of V. The set E(G) is called edge set of G.

$$E(G) = \{(u,v) : u,v \in V(G)\}$$

$$E(G) \subseteq V(G) \times V(G)$$

Incident-

The edge e that joins the vertices u and v is said to be incident. on each of its end points u and v.

Adjacent Vertices-

Any pair of vertices that is connected by an edge in a graph is called adjacent vertices.

Isolated Vertex-

~~Any~~ In a graph a vertex that is not adjacent to another vertex is called an isolated vertex.

Undirected Graph-

An undirected graph G consists of a set V of vertices and a set E of edges such that each edge $e \in E$ is associated with an unordered pair of vertices.

Directed Graph -

A directed graph (or digraph) G consists of a set V of vertices and a set E of edges such that $e \in E$ is associated with an ordered pair of vertices. In other words, if each edge of the graph G has a direction then the graph is called directed graph.

Loop-

An edge of a graph that joins a vertex to itself is called a loop or self loop i.e, a loop is an edge (v_i, v_j) where $v_i = v_j$

Multiple or parallel edges -

In some directed as well as undirected graph, we may have contain pair of vertices joined by more than one edges, such edges are called multiple or parallel edges.

Two edges (v_i, v_j) and (v_f, v_r) are parallel edges if $v_i = v_f$ and $v_j = v_r$.

Note -

In case of directed edges, the two possible edges between a pair of vertices which are opposite in direction are considered distinct. So more than one directed edge in a particular direction in the case of a directed graph is considered parallel.

Finite / Infinite graph-

A graph $G(V, E)$ is said to finite if it has a finite number of vertices and finite number of edges; otherwise it is a infinite graph.

Order of graph-

If G is a finite, $|V(G)|$ denotes the number of vertices in G and is called the order of G.

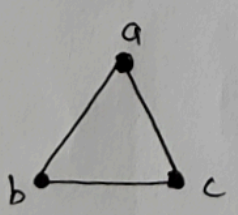
Size of graph-

If G is finite, $|E(G)|$ denotes the number of edges in G and is called the size of G.

Note-

Sometimes graph can be represent as of order n and size m an (n, m) graph.

Example-



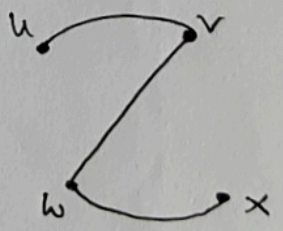
$V = \{a, b, c\}$

$E = \{(a,b), (a,c), (b,c)\}$

$|V(G)| = 3$

$|E(G)| = 3$

Adjacent vertices- a and b, a and c and c and b



$V = \{u, v, w, x\}$

$E = \{(u,v), (v,w), (w,x)\}$

$|V(G)| = 4$

$|E(G)| = 3$

Adjacent vertices - u and v, v and w and w and x

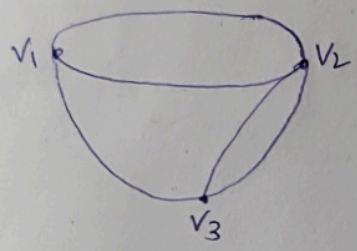
Non adjacent vertices - u and w, u and x, v and x.

Simple Graph-

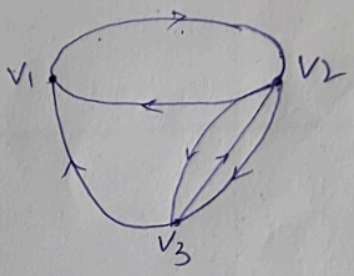
A graph which has neither loops nor multiple edges i.e, where each edge connects two distinct vertices and no two edges connect the same pair of vertices is called simple graph.

Multigraph-

Any graph which contains some multiple edges is called a multigraph. In a multigraph no loops are allowed.



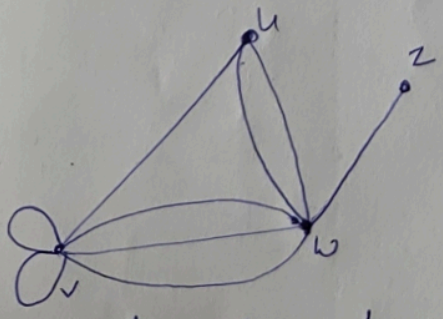
Undired Multigraph



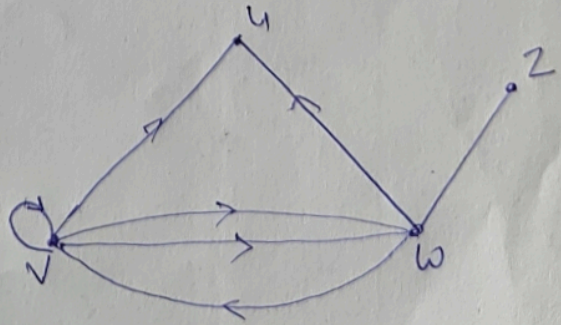
Directed Multigraph

Pseudograph-

A graph with self-loops and multiple edges is called a pseudograph.



Undirected pseudograph

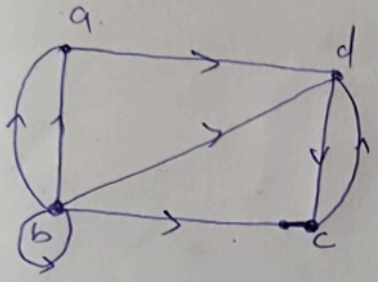


Directed pseudograph.

Note –

1. The sum of the indegree and out degree of a vertex is called the total degree of the vertex.
2. A vertex with zero indegree is called a source and a vertex with zero out degree is called a sink.

Eg



Indeg(a) = 2 Outdeg(a) = 1 Total degree = 8+
Indeg(b) = 1 Outdeg(b) = 5
Indeg(c) = 2 Outdeg(c) = 1
Indeg(d) = 3 Outdeg(d) = 1

The sum of the indegree and the sum of the out degree each equal to 8,

Walk –

A walk in a graph G is a finite alternating sequence.

$$v_0 - e_1 - v_1 - e_2 - v_2 - e_3 \cdots - e_n - v_n$$

of vertices and edges of the graph such that each edge e_i in the sequence joins vertices v_{i-1} and v_i. $1 \le i \le n$.

→ The end vertices v_0 and v_n are the end or terminal vertices of the walk.

→ The vertices $v_1, v_2 \cdots v_{n-1}$ are called its internal vertices.

→ A walk is open when the terminal vertices are distinct.

→ A walk is closed when the terminal vertices are same.

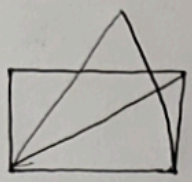
Note –

A walk may repeat both vertices and edges.

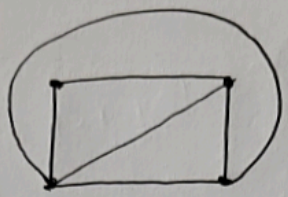
Planar Graph–

A graph is a planar graph if it is possible to represent it in the plane such that no two edges of the graph intersect except possibly at a vertex to which they are both incident.

Eg



Non-planar



planar

Eulers formula–

Theorem–

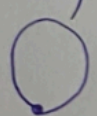
If a connected ~~pair~~ planar graph G has n vertices, e edges and r region, then $n-e+r=2$.

Proof: We prove the theorem by induction on e, number of edges of G.

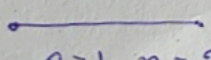
Basis of induction: If $e=0$, then G must have just have one vertex i.e, $n=1$ and one infinite region i.e $r=1$.

Then, $n-e+r=1-0+1=2$.

If $e=1$, then the number of vertices of G is either 1 or 2, the first possibility of occuring when the edge is a loop. These two possibilities give rise to two regions and one region respectively.



$e=1, n=1$



$e=1, n=2$

In the case of loop, $n-e+r = 1-1+2 = 2$ and in case of non-loop, $n-e+r = 2-1+1 = 2$.

Induction hypothesis :

Now, we suppose that the result is true for any connected plane graph G with e-1 edges.

Induction step :

We add one new edge K to G to form a connected Supergraph of G which is denoted by G+K. There are three possibilities.

(i) K is a loop, in which case a new region bounded by the loop is created ~~by~~ but the number of vertices remains unchanged.

(ii) K joins two distinct vertices of G, in which case one of the region of G is split into two, so that number of region is increased by 1, but the number of vertices remains unchanged.

(iii) K is incident with only one vertex of G on which case another vertex must be added, increasing the number of vertices by one, but leaving the number of regions unchanged.

If let n', e', and r' denotes the number of vertices, edges and regions in G and n, e and r denote the same in G+K. Then.

A walk

In case (i) $n-e+r = n'-(e'+1)+(r'+1) = n'-e'+r'$

In case (ii) $n-e+r = n'-(e'+1)+(r'+1) = n'-e'+r'$

In case (iii) $n-e+r = (n'+1)-(e'+1)+r' = n'-e'+r'$

But by our induction hypothesis, $n'-e'+r'=2$. Thus in each case $n-e+r=2$.

Hence, by mathematical induction the formula is true for all plane graph.

Euler Path-

Euler path is a path in a graph that visits every edge exactly once.

Euler Circuit-

Trail-

A walk is called a trail if all its edges are distinct. A trail is open or closed depends on whether its end vertices are distinct or not.

Circuit-

A closed trail is called a circuit.

Path-

A walk is called a path if all its vertices and edges are distinct.

Cycle-

A path in which only repeated vertex is the first vertex is called a cycle.

Euler Path-

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

Euler Circuit-

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

Note

→ An Euler path starts and ends at different vertices.

→ An Euler circuit starts and ends at the same vertex.

Euler Graph-

A connected graph G is called an Euler graph, if there is a closed trail which includes every edge of the graph G.

Hamiltonian Graph-

If there exists a closed walk in the connected graph that visits every vertex of the graph exactly once without repeating the edges, then such a graph is called as as hamiltonian graph.

Hamiltonian Circuit-

If there exists a walk in the connected graph that visits every vertex of the graph exactly once without repeating the edges and returns to the starting vertex, then such a walk is called as a Hamiltonian circuit.

Hamiltonian Path-

If there exists a walk in the connected graph that visits every vertex of the graph exactly once without repeating the edges, then such a walk is called as a Hamiltonian path.

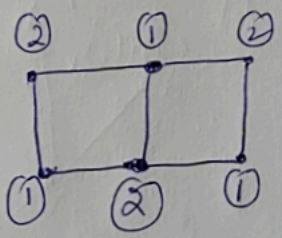
Connected Graph-

A pair of vertices in a graph is a connected pair if there is a path between them. A graph G is connected graph if every pair of vertices in G is a connected pair. Otherwise, it is a disconnected graph.

Chromatic Number-

The chromatic number of a graph G is the minimum number of colours needed for a proper colouring i.e, the minimum number of colours needed to assign colour to each vertex of G such that no two adjacent vertices are of same colour. It is denoted by $\chi(G)$.

Eg-



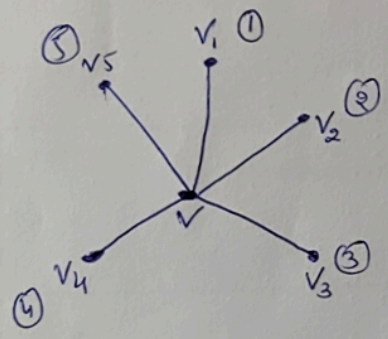
$\chi(G) = 2$.

Theorem:- (Five colour theorem)

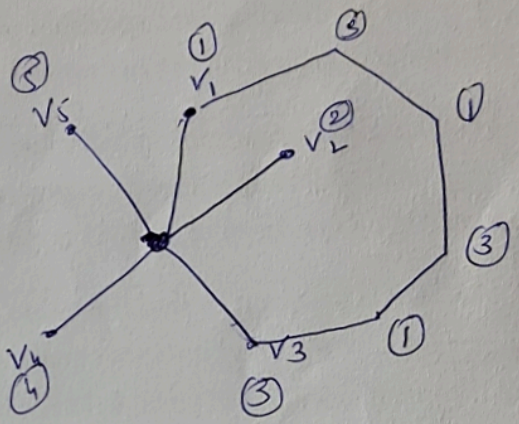
A planar Graph G is 5-Colourable.

Proof The proof is by induction on the number n of vertices G. If $n \leq 5$, then the theorem obviously holds, since the graph is 5-colourable. We assume that the theorem ~~fr to~~ holds for every planar graph with $n-1$ vertices, we shall prove that it is true for a planar graph with n vertices. Since G is a planar and is connected it must have a vertex v such that $\deg(v) \leq 5$.

1. Let G' be the graph obtained by deleting v from G. By the induction hypothesis, G' requires not more than 5 colours. When v has degree 1, 2, 3, 4 there is no difficulty, Since we can give to v one more colour. hence, it Sufficies to consider the case when G' has 5 colours with which its vertices are properly coloured. as shown in fig (a)



(a)



(b)

Suppose there is a path between the vertices v_1 and v_3 coloured alternatively with colours ① and 3 (shown in fig(b)) then, a Similar path between v_2 and v_4 cannot exist, since this path if it exists, will intersect the path between v_1 and v_3, contradicting that G is planar. Hence, we can interchange the colours of all vertices connected to v_2, giving colour 4 to v_2, while v_4 has still colour 4. Thus, we can colour v with colour 2, and hence the graph $G+v = G$ is 5-Colourable. Similarly, if we have assumed that no path exist between v_1 and v_3, we could colour v_3 with colour 1 and v_1 with colour 3. Thus, a planar graph with 5 vertices is 5-colourable.

Shortest Path

1. Dijkstra's Algorithm-

Input:- A connected weighted graph G. If G has self-loops, delete them. If G has parallel edges two vertices edges, delete all except that having the last weight.

Output: L(z), the length of shortest distance from a to z.

Step 1: Initially set the starting vertex a permanently with 0 i.e L(a)=0 and set $L(v)=\infty$ for all vertices $v \neq a$.

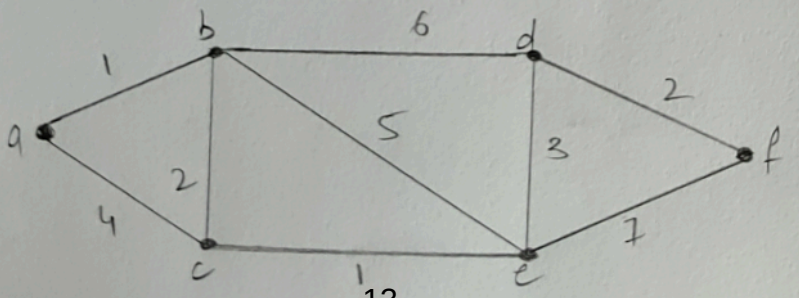
Step 2: Let u be a vertex in T for which L(u) is minimum and hence the ~~parameter~~ permanent label of u.

Step 3: If u=z Stop.

Step 4: For every edge e=(u,v), incident with u, if $v \in T$, change L(v) to min (L(v), L(u)+ w(e)).

Step 5: Change T to T-{u} and go to step 2.

Example - Apply Dijkstra's algorithm to the graph given below and find the shortest path from a to f.



Soln

a	b	c	d	e	f	Reasons
[0] ✓	∞	∞	∞	∞	∞	Starting vertex 'a' is permanently labelled 0, others are temporarily labelled ∞.
[0]	min(∞, 0+1)= [1] ✓	min(∞, 0+4)= 4	∞	∞	∞	There are two edges incident with a→ab and ac, b is permanently labelled 1 since it is minimum in the row.
[0]	[1]	min(4, 1+2)= [3]	min(∞, 1+6)= 7	min(∞, 1+5)=6	∞	There are three edges incident with b→bc, bd and be, c is permanently labelled 3 since it is minimum in this row.
[0]	[1]	[3]	7	min(6, 3+1)= [4] ✓	∞	There is one edge incident with c→ce. e is the permanently labelled 4 since it is minimum in this row.
[0]	[1]	[3]	min(7, 4+3)= [7] ✓	[4]	min(∞, 4+7)=11	There are two edges incident with e→ed and ef. d is permanently labelled 7.
[0]	[1]	[3]	[7]	[4]	min(11, 7+2)= [9]	There is one edge incident with d→df, f is permanently labelled 9. The algorithm stops because f is the final vertex.

Since the permanent label of final vertex (f) is 9, the shortest distance between a and f is 9.

Minimal Spanning Tree -

Let G be a connected weighted graph. The weight of a spanning tree of G is the sum of the weights of the edges. A minimal spanning tree of G is a spanning tree of G with minimum weight.

Algorithm for Minimal Spanning Trees -

1. Kuskal's Algorithm.

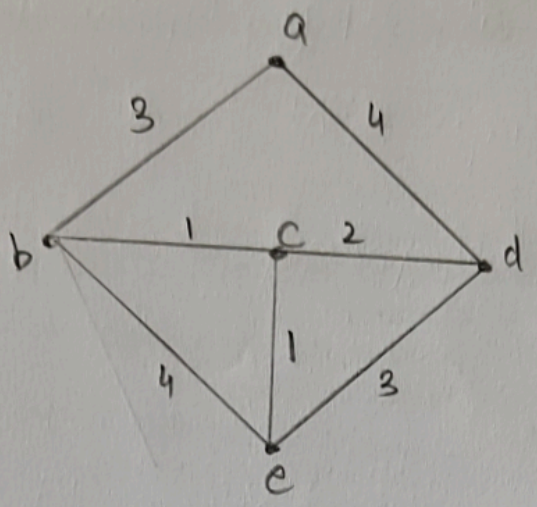
Step 1:- List all the edges (which do not form a loop) of G in non-decreasing order of their weights.

Step 2:- Select an edge of minimum weight (if more than one edge of minimum weight, arbitrarily choose one of them) This is the first edge of T.

Step 3:- At each stage, select an edge of minimum weight from all the remaining edge of G. if it does not form a circuit with the previously selected edges in T. Include the edge in T.

Step 4:- Repeat step 3 until n-1 edges have been selected when n is the number of vertices in G.

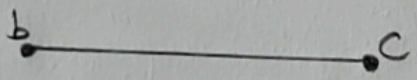
Q) Show how ~~Kur~~ Kruskal's algorithm find a minimal spanning tree of the graph.



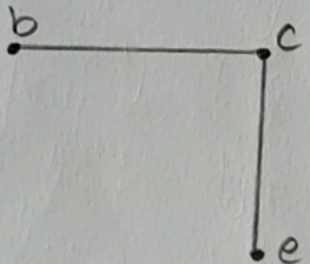
Soln Step1:
List the edges in non-decreasing order of their weights

Edge:	(b,c)	(c,e)	(c,d)	(a,b)	(e,d)	(a,d)	(b,e)
Weight:	1	1	2	3	3	4	4

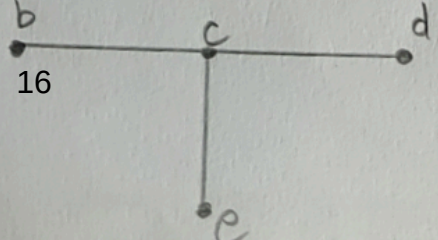
Step2:- Select the edge (b, c) since it has the smallest weight, include it in T.



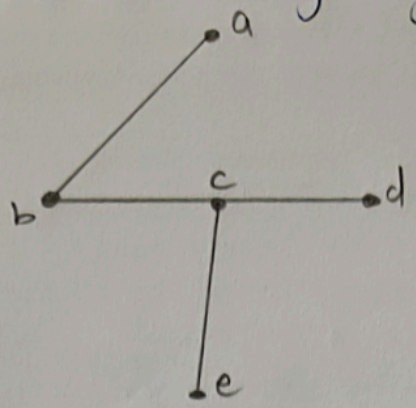
Step 3:- Select an edge with the next smallest weight (c,e) since it does not form circuit with the existing edges in T, so include it in T.



Step 4:- Select an edge with the next smallest weight (c,d) since it does not form circuit with the existing edge in T, so include it in T.



Step 5 :- Select an edge with the next smallest weight (a,b) since it does not form circuit with the existing edges in T, so include it in T.



Since G contains 5 vertices and we have chosen 4 edges, we stop the algorithm and the minimal spanning tree is produced.

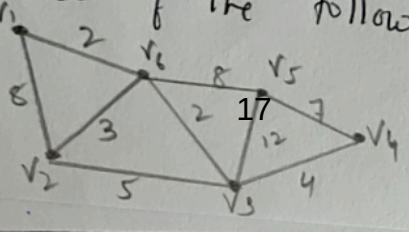
2. Prim's Algorithm-

Step 1:- Remove all self loop (if exist) and all parallel edges b/w two vertices except the one with minimum weight. Select any vertex in G. Among all the edges incident with the selected vertex, choose an edge of minimum weight. Include it in T.

Step 2:- At each stage, choose an edge of smallest weight joining a vertex already included in T and a vertex not yet included, if it doesn't form a circuit with the edges in T. Include it in T.

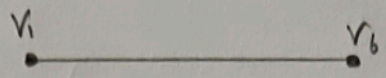
Step 3:- Repeat until all the vertices of G are included.

Q. Describe prim's algorithm and use this to find out the minimal spanning tree of the following graph.

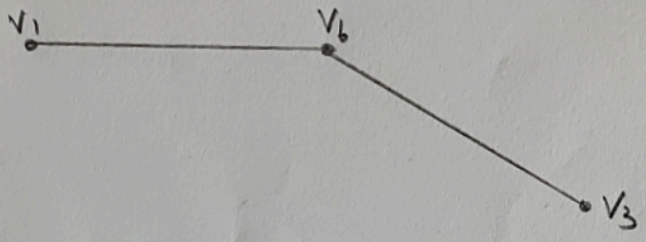


2+2+3+4+7+10

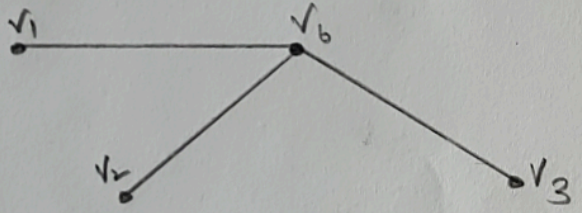
2 Soln Step 1:- We choose the vertex v_1 as the starting vertex. The edge with the smallest weight incident on v_1 is (v_1, v_6) as we choose the edge and incident it in T.(spanning tree)



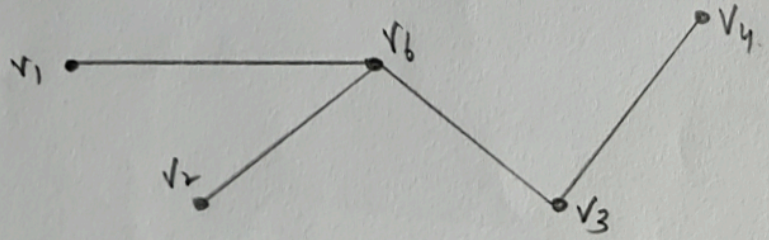
Step 2:- Now, $w(v_1, v_2) = 8$, $w(v_6, v_2) = 3$, $w(v_6, v_3) = 2$ and $w(v_6, v_5) = 8$. We choose the edge (v_6, v_3) since it is minimum and include it in T.



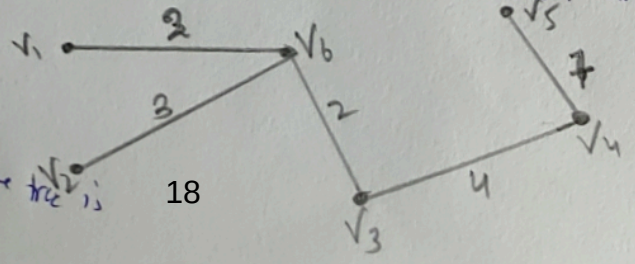
Step 3:- Again $w(v_1, v_2) = 8$, $w(v_6, v_2) = 3$, $w(v_6, v_5) = 8$, $w(v_3, v_2) = 5$, $w(v_3, v_4) = 4$ and $w(v_3, v_5) = 12$. We choose (v_6, v_2) since it is minimum and include in T.



Step 4:- Now among elligible edges, $w(v_6, v_5) = 8$, $w(v_3, v_5) = 12$, $w(v_3, v_4) = 4$ we choose (v_3, v_4) since it is minimum and include it in T.



Step 5:- Only the vertex v_5 is not yet include. Now $(v_6, v_5) = 8$, $w(v_3, v_5) = 12$ $w(v_4, v_5) = 7$. We choose (v_4, v_5) since it is minimum and include it in T.



The minimal spanning tree is obtained the total weight of the tree is
$2+2+3+4+7 = 18$.

Difference between Prim's and Kruskal's Algorithm-

In prim's algorithm at every step edges of minimum weight that are incident to a vertex, already in tree and not forming a cycle, are chosen.

In Kruskal's algorithm, edges of minimum weight that are not necessarily incident to a vertex already in the tree and not forming a cycle is chosen.

Spanning Tree-

A subgraph T of a connected graph $G(V, E)$ is called a spanning tree if

(i) T is a tree.

(ii) T includes every vertex of G i.e $|V| = |G|$.

→ If $|V| = n$ and $|E| = m$, then the spanning tree of G must have n vertices and hence $n-1$ edges.

Algorithm for Constructing Spanning Trees-

1. Breath-first Search (BFS)-

The idea of BFS is to visit all vertices on a given level before going into the next level until all are visited.

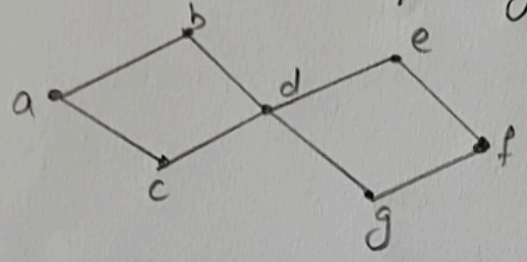
Procedure-

1. Arbitrarily choose a vertex and designate it as the root. Then add all edges incidents to this vertex, such that the addition of edges does not produce any cycle.
2. The new vertices added at this stage become the vertices at level 1 in the spanning tree, arbitrarily order them.
3. Next, for each vertex at level 1, visited in order, add each edge incident to this vertex to the tree as long as it does not produce any cycle.
4. Arbitrarily order the children of each vertex of at level 1. This produces the vertices at level 2 in the tree

5. Continue the same procedure until all the vertices in the tree have been added.

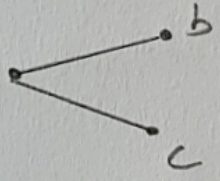
6. The procedure ends, since there are only a finite no. of edges in the graph.

Q Use BFS algorithm to find a spanning tree of graph G of given fig.

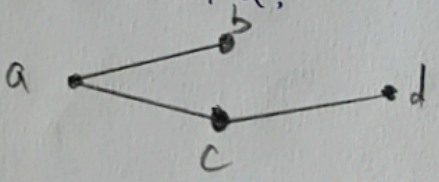


Soln (i) Choose the vertex 'a' to be the root a.

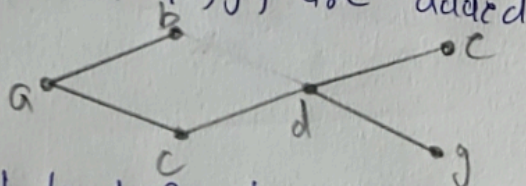
(ii) Add edges incident with all vertices adjacent to 'a', so that edges {a,b}, {a,c} are added. Two vertices b and c are level 1 in the tree.



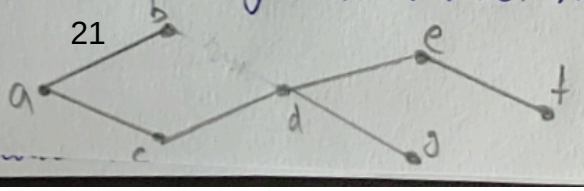
(iii) Add edges from these vertices at level 1 to adjacent vertices not already in the tree. Hence the edge {c,d} is added. The vertex d is in level 2.



(iv) Add edge from d in level 2 to adjacent vertices not already in the tree. The edge {d,e} and {d,g} are added. Hence e and g are in level 3.



(v) Add edge from e at level 3 to adjacent vertices not already in the tree and hence {e,f} is added.



connected. A spanning

Depth-first Search (DFS)-

An alternative to breath-first search is depth-first search which proceeds to successive levels in a tree at the earliest possible opportunity. DFS is also called back tracking.

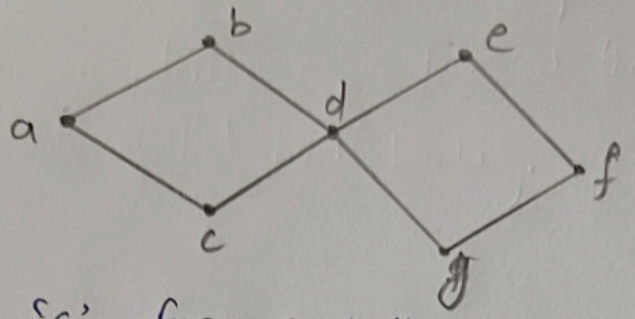
Procedure—

1. Arbitrarily choose a vertex from the vertices of the graph and designate it as the root.
2. Form a path starting at this vertex by successively adding edges as long as possible where each new edge is incident with the last vertex in the path without producing any cycle.
3. If the path goes through all vertices of the graph, the tree consisting of this path is a spanning tree.

   Otherwise, move back to the next ~~level~~ to last vertex in the path, and if possible form a new path starting at this vertex passing through vertices that were not already visited.
4. If this cannot be done, move back another vertex in the path, that is two vertices back in the path, and repeat.
5. Repeat this procedure, beginning at the last vertex visited, moving back up the path one vertex at a time, forming ~~a~~ new paths that are as long as possible until no more edges can be added.

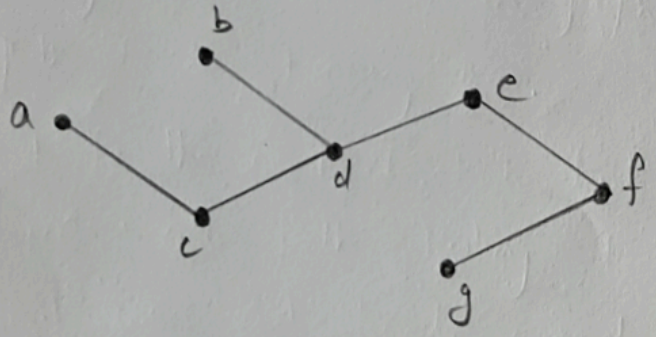
6. This process ends since the graph has a finite no. of edges and is connected. A spinning is produced.

6 Find a spanning tree of the graph using Depth-first search algorithm.



Soln Chose the vertex 'a'. Form a path by successively adding edges incident with vertices not already in the path as long as possible. This produces the path a-c-d-e-f-g.

Now back track to 'f'. There is no path beginning at f containing vertices not already visited. Similarly, after back track at 'e', there is no path. So more back track at 'd' and form the path d-b. This produces the required spanning tree.



Note -

(i) For DFS and BFS, discard all parallel edges and self loops from the given connected graph.

(ii) If the given connected graph G is directed graph then we construct the corresponding undirected graph.