Solution: The graph shown in Fig. 6.183 can be redrawn as planar graph as follows:

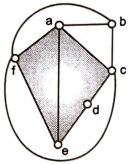


Fig. 6.183

The planar representation of the graph in Fig. 6.184 is shown below:

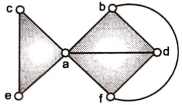


Fig. 6.184

Example 2: *How many edges must a planar graph have if it has 7 regions and 5 nodes. Draw one such graph.*

Solution: According to Euler's formula, in a planar graph $v - e + r = 2$.

where v, e, r are the number of vertices, edges and faces in a planar graph.

Here $v = 5$, $r = 7$, $e = ?$

Since the graph is a planar graph, therefore $5 - e + 7 = 2$

$\Rightarrow \quad e = 10$

Hence, the given graph must have 10 edges. This graph is shown below in Fig. 6.185.

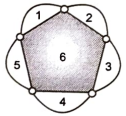


Fig. 6.185

Example 3: *Determine the number of regions defined by a connected planar graph with 6 nodes and 10 edges. Draw a simple and a non simple graph.*

Solution: Given $v = 6$, $e = 10$.

Hence by Euler's formula for a planar graph

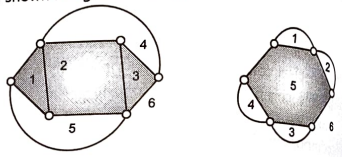
$v - e + r = 2$ where, r is the number of faces.

$\Rightarrow \quad 6 - 10 + r = 2$

$\Rightarrow \quad r = 6$

Hence the graph should have 6 faces.

Simple and a multiple graphs with $v = 6$, $e = 10$ and $r = 6$ are shown in Fig. 6.186 below.



Simple Graph **Multiple Graph**

Fig. 6.186

Example 4: *A connected planar graph has nine vertices having degrees 2, 2, 2, 3, 3, 3, 4, 4 and 5.*

How many edges are there? How many faces are there?

Solution: By Handshaking lemma

$$2e = \sum_{i=1}^{n} d(v_i) = \text{total degree}$$

where e is the number of edges

$\Rightarrow \quad 2e = 2 + 2 + 2 + 3 + 3 + 3 + 4 + 4 + 5$

$2e = 28 \Rightarrow e = 14$

Now, by Euler's formula

$v - e + r = 2$

$\Rightarrow \quad 9 - 14 + r = 2$

$\Rightarrow \quad r = 7$

Hence, there are 14 edges and 7 faces in the graph.

Example 5: *Show that a simple planar graph has a vertex of degree 5 or less.*

Solution: Let $G = (V, E)$ be a simple planar graph, where $|V| = v$, $|E| = e$.

If the number of vertices is less than or equal to 6, then there exists a vertex of degree 5 or less in the graph.

Suppose $|V| > 6$ and there is no vertex of degree ≤ 5.

Hence, by Handshaking lemma

$$2e = \sum_{i=1}^{v} d(v_i) \geq 6v$$

$\Rightarrow \quad 2e \geq 6v$

Therefore,

$e \leq (3v - 6)$

$\Rightarrow \quad 2e \leq 2(3v - 6)$

$\Rightarrow \quad 2e \leq 6v - 12$

$\Rightarrow \quad 6v \leq 2e \leq 6v - 12$

$\Rightarrow \quad 6v \leq 6v - 12$

$\Rightarrow \quad 0 \leq -12$

which is impossible.

Hence in a simple planar graph, there exists a vertex of degree 5 or less.

Example 6: Three utility problem: *There are three homes H_1, H_2 and H_3, each to be connected to each of the three utilities: Water (W), Gas (G) and Electricity (E) by means of conduits. Is it possible to make such connections without any crossovers of the conduits?*

Solution: The problem can be represented by a graph shown in Fig. 6.187, the conduits are shown as edges while the houses and utility supply centers are vertices.

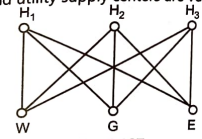
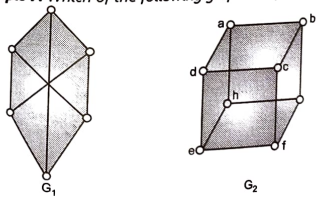


Fig. 6.187

The above graph is a complete bipartite graph $K_{3,3}$ which is a non-planar graph. Hence it is not possible to draw edges without crossing over.

Therefore it is **not** possible to make the connection without any crossovers of the conduits.

Example 7: *Which of the following graphs are planar?*



G_1 G_2

Fig. 6.188

Solution: The graph G_1 can be drawn as follows:

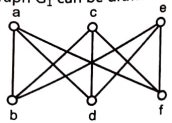


Fig. 6.189

The above graph is a Kuratowski's second graph $K_{3,3}$ which is non-planar.

Hence, the given graph G_1 is non-planar.

Consider G_2, it can be redrawn as follows:

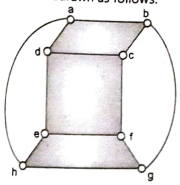


Fig. 6.190

The above graph is a planar graph because edges do not cross each other. Hence, the given graph G_2 is a planar graph.

Example 8: *Draw a planar representation of each graph if possible.*

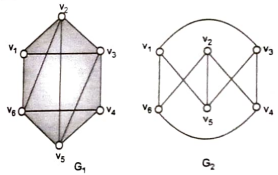


Fig. 6.191

Solution: The planar representation of G_1 is shown in Fig. 6.192.

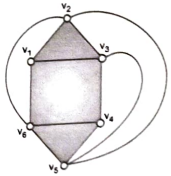


Fig. 6.192

The planar representation of G_2 is shown as follows:

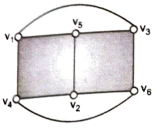


Fig. 6.193

Example 9: *Determine which of the graphs of Fig. 6.194 represents Eulerian circuit, Hamiltonian circuit, Bipartite graph and planar graph. Justify your answer in each case.*

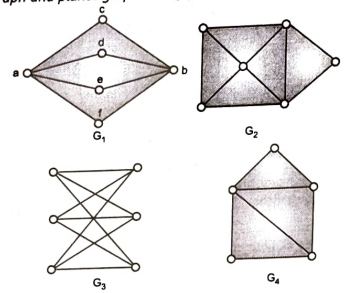


Fig. 6.194

Solution: G_1 is a bipartite graph in which the set of vertices V can be partitioned into two sets V_1 and V_2, where V_1 = {a, b} and V_2 = {c, d, e, f}

Such that $V_1 \cup V_2 = V$ and $V_1 \cap V_2 = \phi$ also each edge of G is from V_1 to V_2. The other representation of the graph is

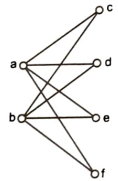


Fig. 6.195

Also, it is a complete bipartite graph $K_{2,4}$. Also the degree of each vertex in G_1 is even hence it contains an Eulerian circuit a c b d a e b f a. G_1 does not contain any Hamiltonian circuit. It is a planar graph since no two edges are intersecting each other.

Consider G_2

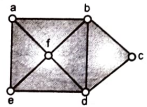


Fig. 6.196

It is not a bipartite graph because we cannot partition its vertex set into two parts V_1 and V_2 not having edges among themselves.

It contains a Hamiltonian circuit a b c d f e a.

Since the degree of each vertex is not even hence it does not contain an Eulerian circuit. But it contains an Eulerian path because there exists exactly two vertices of odd degree. An Eulerian path is a b c d f b d e f a e.

It is a planar graph because edge crossing is not there in the graph G_2

Consider G_3

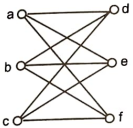


Fig. 6.197

It is a bipartite graph. Divide its vertex set V = {a, b, c, d, e, f} into two parts V_1 and V_2 such that V_1 = {a, b, c}, V_2 = {d, e, f} and each edge in G_3 is from V_1 to V_2.

In fact, it is a complete bipartite $K_{3,3}$ graph because each vertex in V_1 is joined to each vertex of V_2.

Since the degree of each vertex is not even, it does not have an Eulerian circuit.

As we know $K_{m,n}$ has a Hamiltonian circuit if m = n. Hence $G_3 = K_{3,3}$ has a Hamiltonian circuit given by a d b e c f a.

$K_{3,3}$ is known as Kuratowski's second graph which is given by planar graph.

Consider G_4

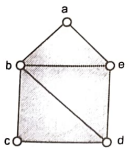


Fig. 6.198

It is not a bipartite graph.

Degree of each vertex is not even $\Rightarrow$ there is no Eulerian circuit.

But it has an Eulerian path because it contains exactly two vertices d and e of odd degree. An Eulerian path is d b a e b c d e.

It contains a Hamiltonian circuit a b c d e a.

It is a planar graph.

Example 10: *Show that in a connected planar linear graph with 6 vertices and 12 edges, each of the regions is bounded by 3 edges.*

Solution: According to Euler's theorem for planar graph, v - e + r = 2, where v is the number of vertices, e is the total number of edges and r is the number of regions in the given graph.

Here, v = 6 and e = 12.

Therefore, we have

$$6 - 12 + r = 2$$
$$r = 8$$

Hence, the given graph contains 8 regions. If each region is bounded by 3 edges then

$$2e = 3r$$

i.e. $2 \times 12 = 3 \times 8$

$$24 = 24, \text{ which is true.}$$

Hence, each region of the given graph is bounded by 3 edges.

Example 11: *Identify whether the graphs given are planar or not.*

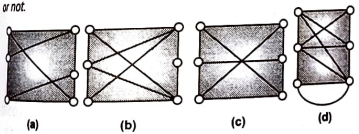


Fig. 6.199

Solution: (a) The graph shown in Fig. 6.199 (a) is planar. Its planar graph is

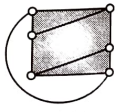
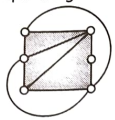


Fig. 6.200

(b) The planar graph of Fig. 6.199 (b) is as follows:



(c) The graph shown in Fig. 6.199 (c) is non-planar because it is Kuratowski's graph $k_{3,3}$, which is non-planar.

The graph shown in Fig. 6.199 (c) can be redrawn as follows:

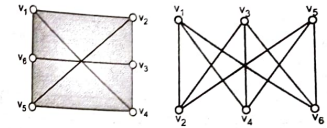


Fig. 6.202

(d) The graph shown in Fig. 6.199 (d) is planar as shown below:

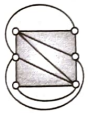


Fig. 6.203

Example 12: *How many colours are required to colour $K_{m,n}$? Why?*

Solution : $K_{m,n}$ is a bipartile graph with vertex set V which can be partitioned into two disjoint subsets V_1 and V_2 such that $V_1 \cap V_2 = \phi$, $V_1 \cup V_2 = V$ and each edge of $K_{m,n}$ joins a vertex of V_1 to a vertex from V_2. Since vertices of V_1 are not connected, therefore, all vertices of V_1 can be coloured using one colour. Similarly all vertices of V_2 can be coloured using another colour.

Thus, minimum 2 colours are required to colour the graph $K_{m,n}$.

Hence, $K_{m,n}$ graph is 2-chromatic.

Example 13: *State whether the following statements are TRUE or FALSE.*

(i) A graph is planar if and only if it contains a subgraph homeomorphic to $K_{3,3}$ or K_5.

(ii) Graph G is called as tree if it is connected acyclic graph.

(iii) A graph with one vertex and no edge is trivial graph.

(iv) The maximum degree of any vertex in a simple graph with n vertices is n+1.

(v) The chromatic number of an n vertex simple connected graph which does not contain any odd length cycle is 2. Assume n > 2.

Solution :

(i) False (A graph is planar if and only if it does not contain a subgraph homeomorphic to $K_{3,3}$ or K_5)

(ii) True

(iii) True

(iv) False (The maximum degree of any vertex in a simple graph with n vertices is n-1)

(v) True

EXERCISE – 6.2

1. Find the edge connectivity of the graph shown in Fig. 6.204.

Fig. 6.204

2. Determine the maximal edge connectivity of a connected graph with 5 nodes and 8 edges. Draw a graph in which the connectivity is met and one in which it is not.
3. Define a connected graph.

   In the following graph in Fig. 6.205, what will happen if we remove the vertices v_1 and v_5? Draw the graph after removal.

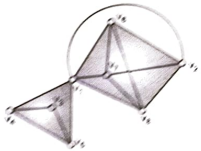


Fig. 6.205

4. Explain edge connectivity and vertex connectivity with examples. What is the relation between them?
5. Is the graph given in Fig. 6.205 is a separable graph? Justify your answer.
6. Define isomorphism. Find whether any edge in the graph G in Fig. 6.205 is an isthmus or not.
7. Write Dijkstra's shortest path algorithm to obtain a shortest path between two vertices in the graph.

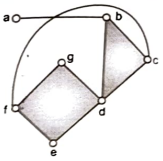


Fig. 6.206

8. Apply shortest path algorithm to determine a shortest path between a and z in the graphs shown below:

(i)

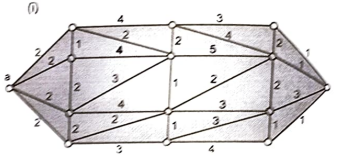


Fig. 6.207

(ii)

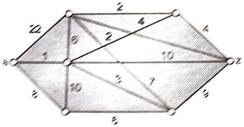


Fig. 6.208

(iii) **(Dec. 2014)**

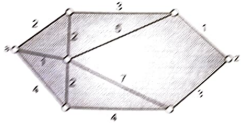


Fig. 6.209

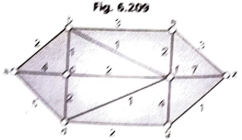


Fig. 6.210

9. Show that none of the following graphs contains a Hamiltonian circuit.

(i) (ii)

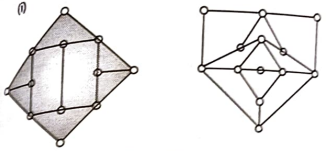


Fig. 6.211

10. Find an Eulerian circuit in the following graphs:

(i) (ii)

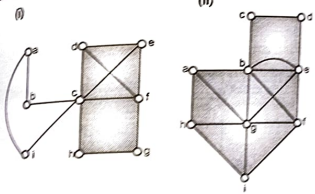


Fig. 6.212

11. Define the following terms:
    (i) Eulerian graph
    (ii) Hamiltonian graph
    (iii) Eulerian path
    (iv) Planar graph
    (v) Hamiltonian circuit
12. Determine whether following has a Hamiltonian circuit or not:

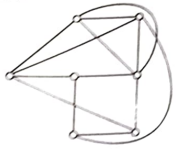


Fig. 6.213

13. State Euler's formula for a planar graph.

    Draw 2 non-isomorphic simple planar graphs with 6 nodes and 9 edges.
14. Draw a simple planar graph with 6 nodes and 10 edges.
15. Consider the graph as shown in Fig. 6.214. Find all the subgraphs when each vertex is deleted. Does any subgraph have any cut points?

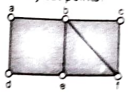


Fig. 6.214

16. Discuss with example relationship between complete bipartite graph and hamiltonian path.
17. Define planar graph. Let G be a connected planar graph with p vertices and q edges where $p \geq 3$ then show that $q \leq 3p - 6$.
18. Define and derive Euler's formula for the planar graph.

ANSWERS 6.2

1. Edge connectivity 4
2. Maximal edge connectivity $= \left[\frac{16}{5}\right] = 3$

   When the edge connectivity is met.

α

Fig. 6.215

When the edge connectivity is not met

β

Fig. 6.216

3. The graph will become disconnected and it will have 2 components.

γ δ

Fig. 6.217

5. Yes it is a separable graph.
   Removal of v_1 disconnects the graph.
6. The edge (a, b) is an isthmus.
8. (i) Shortest path is 8. (ii) Shortest path for a to z is
9. (iii) Shortest path is 6.
10. (i) The vertex d has degree 3, hence no Eulerian circuit.
    (ii) vertex a has degree 3, hence no Eulerian circuit.
12. No Hamiltonian circuit.
13.

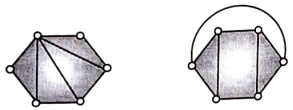


Fig. 6.218

14.

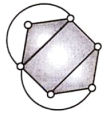


Fig. 6.219

15. All subgraphs of the given graph when each vertex is deleted are shown below:
    (i) when a is deleted:

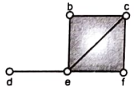


Fig. 6.220

e is a cut point.

(ii) when b is deleted:

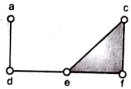


Fig. 6.221

e is a cut point.

(iii) when c is deleted:

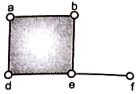


Fig. 6.222

e is a cut point.

(iv) when d is deleted

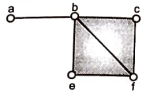


Fig. 6.223

b is a cut point.

(v) when e is deleted:

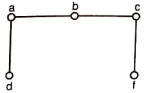


Fig. 6.224

Now, b is a cut point.

(vi) when f is deleted:

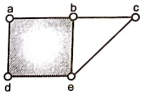


Fig. 6.225

No cut point is there.

POINTS TO REMEMBER

- A graph that has neither self loops nor parallel edges is called a **Simple Graph**, otherwise, it is called a **Multiple Graph**.
- Let G be a graph with vertex set V and edge set E. If each edge or each vertex or both are associated with some positive real numbers then the graph is called a **Weighted Graph**.
- A graph with finite number of vertices as well as finite number of edges is called a **Finite Graph**, otherwise, it is an infinite graph.
- A graph G = (V, E) is called a labeled graph if its edges are labeled with some name or data.

- Two vertices v_1 and v_2 of a graph G are said to be **adjacent** to each other if they are the end vertices of the same edge.
- The number of edges incident on a vertex v_i with self loop counted twice, is called the degree of the vertex v_i. It is denoted by $d(v_i)$
- A vertex with degree zero is called an isolated vertex. More formally, a vertex having no incident edge is an isolated vertex. A vertex of degree 1 is called a pendant vertex.
- Consider a graph G with e number of edges and n number of vertices. Since each edge contributes two degrees, the sum of the degrees of all vertices in G is twice the number of edges in G. i.e.

$$\sum_{n=1}^{n} d(v_i) = 2e$$

This is called **Handshaking Lemma**.

- Given a graph G with n vertices, e edges and no self loops. The incidence matrix $X(G) = [x_{ij}]$ of the graph G is an n × e matrix where,

$x_{ij} = 1$; if j^{th} edge e_j is incident on i^{th} vertex v_i

$= 0$; otherwise

Here n rows correspond to n vertices and e columns correspond to e edges.

- **Directed graphs or Digraphs:** A **directed graph** or **digraph D** is an ordered pair (V, A) where V is a non empty set of elements called vertices and A is the set of ordered pair of elements called directed edges or arcs. In other words, we can say that if the each edge of the graph G has a direction then the graph is called **Directed Graph** or **digraph**.
- The indegree of a vertex u of digraph D is defined as the **number of arcs which are incident into u** and is denoted by $\overrightarrow{d(u)}$. Similarly, the **outdegree of** a vertex u is defined as the **number of arcs which are incident out of u** and is denoted by $\overleftarrow{d(u)}$
- **Null Graph:** If the edge set of any graph with n vertices is an empty set, then the graph is known as **null graph**. It is denoted by N_n.
- **Complete Graph:**Let G be a simple graph on n vertices. If the degree of each vertex is (n – 1), then the graph G is called a **complete graph.** More generally, if every pair of vertices is adjacent in any simple graph, then the graph is said to be a **complete graph**.
- **Bipartite Graph:** Let G be a graph with vertex set V and edge set E, then G is called a **bipartite graph** if its vertex set V can be partitioned into two disjoint subsets say V_1 and V_2 such that $V_1 \cup V_2 = V$ and $V_1 \cap V_2 = \phi$ and also each edge of G joins a vertex of V_1 to a vertex of V_2.
- **Complete Bipartite Graph:** A bipartite graph is called a **complete bipartite graph if each** vertex of V_1 is joined to every vertex of V_2 by a unique edge.
- **Subgraph:** Let G = (V, E) be any given graph. Then the graph G' = (V', E') is called a **subgraph** of G if V' ⊆ V and E' ⊆ E. In H_1, H_2 and H_4 subgraphs of G but H_3 is not a subgraph of G because it contains the edge e which is not there in the graph G.
- **Edge Disjoint Subgraphs:** Two subgraphs H_1 and H_2 of the graph G are said to be **edge disjoint subgraphs** of a graph G if there is **no edge** common between H_1 and H_2 (but may have vertex common)
- **Vertex Disjoint Subgraphs:** Two subgraphs H_1 and H_2 are said to be **vertex disjoint subgraphs** of a graph G if there is **no vertex** common between them (i.e. they do not have common edges also)
- **Spanning Subgraph:** Let G = (V, E) be any graph. Then G' is said to be the **spanning subgraph** of the graph G if its vertex set V' is **equal to** the vertex set V of G.
- **Null Subgraph:** A subgraph H of a graph G is called a null subgraph if its vertex set is same as the vertex set of G and its edge set is empty set. i.e. it does not contain any edges. A null subgraph is constructed from a graph G by deleting all the edges of G.
- **Complement of a Graph:** Let G be a simple graph. Then the **complement of G** denoted by $\overline{G}$ is the graph whose vertex set is the same as the vertex set of G and in which two vertices are adjacent if and only if they are **not** adjacent in G.

- **A graph is said to be self complementary if it is isomorphic to its complement.**
- A path in a graph G is called a **simple path** if the edges do **not** repeat in the path.
- A path is said to be an **elementary** path if vertices do **not** repeat in the path.
- A circuit in the graph G is said to be a **simple circuit** if it does not include the same **edge** twice.
- A circuit is said to be an **elementary circuit** if it does not meet the same **vertex** twice (except for first and last vertex)
- A graph is said to be a **connected graph** if there exists a **path** between every pair of vertices, otherwise the graph is **disconnected**.
- **The vertex connectivity K (G)** of a simple connected graph G is defined as the **smallest number of vertices whose removal disconnects the graph.**
- A path is called an **Eulerian Path** if every edge of the graph G appears exactly once in the path.
- A circuit in a connected graph G is called a **Hamiltonian Circuit** if it contains every **vertex of G exactly once** (except the first and the last vertex)
- A graph is said to be a **planar graph** if it can be drawn on the plane **with no intersecting edges** i.e. a graph is a planar graph if it is drawn on a plane such that no edges cross each other.
- A planar representation of a graph divides the plane into regions (also called **windows, faces** and **meshes**) **A region is characterized by the set of edges forming its boundary.**
- A proper colouring of graph G is an assignment of colours to its vertices so that no two adjacent vertices have the same colour.

◈◈◈

Unit V : Trees

CHAPTER 7
TREES

7.1 INTRODUCTION

Trees are special type of connected graphs which appear as natural inverted trees. Trees were discovered by Kirchhoff in 1847 while investigating the electrical networks. In nineteenth century, Sir Arthur Cayley used trees to study the structure of hydrocarbons. Now-a-days, trees play an important role in many fields like computer science, chemistry, operations research etc. Trees are often used to specify hierarchical relationships. For example, an organization chart of a corporation can be represented by tree.

The most common example of tree is a 'family tree' where the children are grouped under their parents in the tree. In computer science, trees are useful in searching, storing, organizing and relating data in data base and analysis of algorithms as it easily accommodates the creation and deletion of nodes.

7.2 DEFINITION AND PROPERTIES OF TREES

A **"tree"** is a simple connected graph without any circuits.

Fig. 7.1 shows trees with one, two, three and four vertices.

Fig. 7.1

The graph in Fig. 7.2 however is not a tree because it contains a cycle abca. Observe that if we remove the edge cb, we get a graph which is a tree.

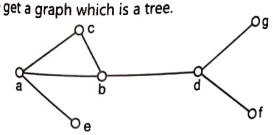


Fig. 7.2

Trees appear in numerous instances. The genealogy of a family is often represented by means of a tree (called a family tree). A river with its tributaries and subtributaries (sub tributaries) can be represented by a tree. The sorting of mail according to zip code is also done according to tree called **Decision Tree**.

It is shown in Fig. 7.3.

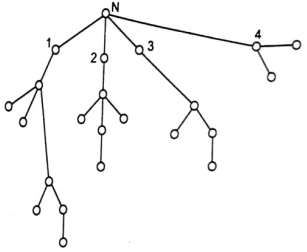


Fig. 7.3

A collection of disjoint trees is known as a **Forest**. A vertex of degree 1 in a tree is called a **Leaf** or a **Terminal Node** and a vertex of degree greater than one is called a **Branch Node** or an **Internal Node**. For example in the following Fig. 7.4, the vertices a, h, i etc. are leaves and vertices b, c, d, e are branch nodes.

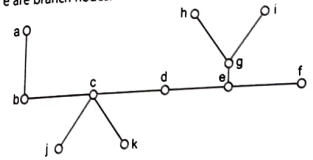


Fig. 7.4

A tree which is defined as non-cyclic connected graph, can be defined in terms of number of edges and vertices in the given graph.

Theorem :

G is a tree if and only if there exists a **unique** path between every pair of vertices of G.

Proof:

We note first that existence of a path between every pair of vertices assures that the graph G is connected.

Moreover, the graph G cannot contain a circuit if these paths are unique, since the existence of a circuit implies that there are two paths between a certain pair of vertices.

Thus, we conclude that a graph in which there is a unique path between every pair of vertices is a tree.

Conversely, if the graph G is a tree, it is connected and hence, there must be at least one path between every pair of vertices in G.

Now suppose that there are two distinct paths between vertices a and b of G.

The union of these two paths will contain a circuit and then, G cannot be a tree which is a contradiction.

Therefore if G is a tree then it has a unique path between every pair of vertices of G.

Theorem :

G is a tree if and only if G is connected and has exactly (n – 1) edges, where n is the number of vertices in G.

Proof:

Given G is a connected graph with the total number of edges e = (n – 1).

For tree, it is suffices to prove that it is non cyclic.

Suppose G contains a circuit C.

Let p denote the number of vertices in the circuit C which is equal to the number of edges in C also.

By the connectivity of G, we can say that every vertex of G that is not in C must be connected to the vertices in C.

Now each edge of G that is not in C can connect only one additional vertex to the vertices in C.

There are (n – p) vertices of G which are not in the circuit C.

Thus G must contain at least (n – p) edges that are not in C.

Hence, the total number of edges e in G is given by $e \geq p + (n - p) = n$ which is a contradiction to the hypothesis that e = n – 1.

It follows that G does not contain a circuit and is, therefore a tree.

Conversely, if G is a tree then by definition it is connected and non cyclic.

We have to prove that the number of edges in G are e = (n – 1).

We will prove this result by induction on n.

For n = 1 the result is obvious.

Let n > 1, consider G – e', for any edge e' of G.

Since a tree is a circuit less graph hence, G – e' is a disconnected graph with two components G_1 and G_2.

Now G_1 and G_2 are connected and contain no cycles since they are both subgraphs of G which is non cyclic.

Let G_i contains v_i vertices and e_i edges. for i = 1, 2.

By induction

$$e_i = n_i - 1, \quad i = 1, 2$$

Therefore, the number of edges in G is given by

$$e = e_1 + e_2 + 1$$

$$e = (n_1 - 1) + (n_2 - 1) + 1$$

$$e = n_1 + n_2 - 1$$

$$\Rightarrow \quad e = n - 1$$

Hence a tree contains (n – 1) number of edges.

The results of the preceding theorems can be summarized by saying that the following are equivalent definitions of a tree.

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

G is connected and a circuit less graph.

(ii) G is connected and has (n – 1) edges.

(iii) G is circuit less and has (n – 1) edges.

(iv) There is exactly one path between every pair of vertices in G.

7.3 CENTRE OF A TREE

As we have seen in the previous section that each of the tree has several pendant vertices. Consider the following tree shown in Fig. 7.5. It has 5 pendant vertices.

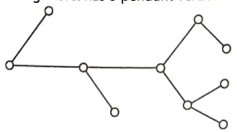


Fig. 7.5

for pendant vertices, we have an important result which says that **in any tree, there are at least two pendant vertices.** The reason is that in a tree of n vertices, we have (n – 1) edges and hence the total degree 2 (n – 1) has to be divided among n vertices. Since no vertex can be zero degree, we must have at least two vertices of degree one in a tree.

7.3.1 Eccentricity of a Vertex

We know that the distance between two vertices in a connected graph is a length of the shortest path between them.

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.

In the following Fig. 7.6.

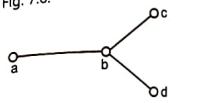


Fig. 7.6

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

A vertex with the **minimum eccentricity** is called a **centre** of G. In the above Fig. 7.6 the vertex b is the centre of that tree.

Every tree has either one or two centres. Consider the following tree in Fig. 7.7.

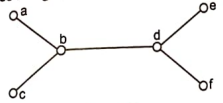


Fig. 7.7

Here E(a) = E(c) = E(e) = E(f) = 3

and E(b) = E(d) = 2

Hence this tree has 2 **centres** b and d.

7.3.2 Cut Points

As we have defined earlier, the cut points or cut vertices are those vertices whose removal disconnects the graph. **In any tree, all the vertices except pendant vertices (degree 1) are cut vertices.** For instance, in the following tree shown in Fig. 7.8, c, d, g are cut points.

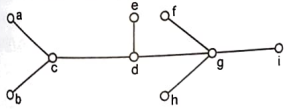


Fig. 7.8

SOLVED EXAMPLES

Example 1: *Draw all non-isomorphic trees on 5 vertices.*

Solution: All non-isomorphic trees on 5 vertices are shown below.

Fig. 7.9

Example 2: *Draw a tree with 6 nodes, exactly 3 of which have degree 1.*

Solution: A tree with 6 vertices which contains 3 pendant vertices is given by

Fig. 7.10

Example 3: *Show that it is possible to draw a tree with 10 vertices which has vertices either of degree 1 or of degree 3. Draw the tree. Is it possible to draw the same type of tree with 11 vertices ?*

Solution: Given the tree has 10 vertices. Therefore it has 9 edges. Suppose there are x number of vertices of degree 1 and y number of vertices of degree 3 in the tree.

Hence, x + y = 10.

Also, by Handshaking lemma ... (1)

$$2e = d(v_i)$$

$$\Rightarrow \quad 2 \times 9 = x + 3y$$

$$\Rightarrow \quad 18 = x + 3y \quad \ldots (2)$$

Solving equations (1) and (2) simultaneously, we get,

x = 6 and y = 4

That is there exist 6 vertices of degree 1 and 4 vertices of degree 3 in the tree of 10 vertices. One such tree is shown below.

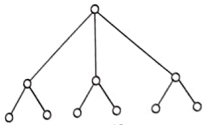


Fig. 7.11

For the second part, consider a tree with 11 vertices and 10 edges which has x vertices of degree 1 and y vertices of degree 3.

Hence $x + y = 11$

Also, by Handshaking lemma $2 \times 10 = x + 3y$

By above equations, we get

$$x = \frac{13}{2} \text{ and } y = \frac{9}{2} \text{ which is impossible.}$$

Therefore, it is not possible to draw a tree with 11 vertices which has vertices of degree 1 or 3.

Example 4: *Is it possible to draw a tree with five vertices having degrees 1, 1, 2, 2, 4 ?*

Solution: Since the tree has 5 vertices hence, it has 4 edges. Now given the vertices of tree are having degrees 1, 1, 2, 2, 4 i.e. the total degree of the tree = 10.

By Handshaking lemma $2e = d(v_i)$

where e is the number of edges in the graph.

$\Rightarrow \quad 2e = 10$

$\Rightarrow \quad e = 5$

which is a contradiction to the statement that the tree has 4 edges. Hence the tree with given degrees of vertices does not exist.

Example 5: *Which trees are complete bipartite graphs ?*

Solution: Suppose T is a tree which is a complete bipartite graph. Let $T = k_{m,n}$ then the total number of vertices in T is $(m + n)$. Hence the tree T contains $(m + n - 1)$ number of edges. But the graph $k_{m,n}$ has (mn) number of edges.

Therefore

$m + n - 1 = mn$

$\Rightarrow \quad (m - 1)(1 - n) = 0$

$\Rightarrow \quad m = 1 \text{ or } n = 1$

This means T is either $k_{1,n}$ or $k_{m,1}$ i.e. T is a star.

Example 6: *Find the centre of the following tree in Fig. 7.12.*

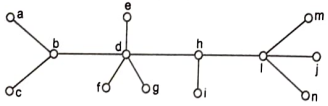


Fig. 7.12

Solution: The centre of the tree is a vertex with minimum eccentricity where eccentricity of the vertex is the distance of it from the farthest vertex.

The eccentricity of the different vertices of the tree in Fig. 7.12 are given as follows.

e (a) = 5, e (b) = 4, e (c) = 5, e (d) = 3, e (e) = 4, e (f) = 4, e (g) = 4, e (h) = 3, e (i) = 4

e (l) = 4, e (m) = 5, e (n) = 5, e (j) = 5.

The vertices with minimum eccentricity are d and h. Hence d and h are the **centres** of the tree.

7.4 ROOTED AND BINARY TREES

- Many of the applications of graph theory, particularly in computer science, use a certain kind of tree, called a rooted tree. These trees are used as models for the structures of file directories. Some of the other important uses of rooted trees include the representation and sorting of data, the representation of algebraic expressions etc.
- A tree in which one vertex (called the root) is distinguished from all the other vertices is known as **rooted tree**. Trees without any root are called **free trees** or **simply trees**. A vertex of degree one which is not a root is called a leaf or external node and all the vertices (including the roots) that are not leaves are called interior nodes. For instance in Fig. 7.13 a vertex N is a root for all the trees shown.

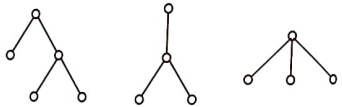


Fig. 7.13

- The other example of rooted tree is sorting of mail according to the zip code. Consider the Fig. 7.13. The point N from where all the mail goes out is distinguished from the rest of the vertices. Hence N can be considered as the root of the tree.
- A rooted tree is often used to specify hierarchical relationships. A example of such a tree, which is an organization chart of a corporation is given in the following Fig. 7.14.

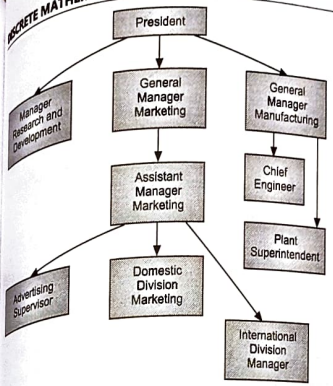


Fig. 7.14

- A directed tree (a tree with directions) is called a **rooted tree** if there is exactly one vertex whose incoming degree is **zero** and all other vertices have incoming degree **one**. The vertex with incoming degree 0 is known as the **root** of the tree. In Fig. 7.15 the vertex a is the root of the directed tree.

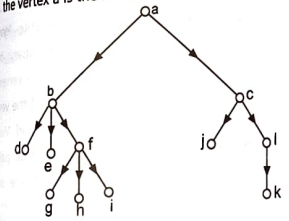


Fig. 7.15

- In a directed rooted tree, a vertex whose outgoing degree is 0 is called a **leaf** and a vertex whose outgoing degree is non zero is called a **Branch Node** or an **Internal Node**. In Fig. 7.15 the vertices b, c and f are branch nodes and the vertices d, e, g are leaves.

Now we are in position to define the level or depth of a vertex in a rooted tree.

- A vertex x in a rooted tree is said to be at **level n or depth n** if there is a path of length n from the root to the vertex x. The **height** of node y is the length of the longest path from y to a leaf. The **height** of the tree is the maximum of the levels or depth of its vertices.

Height of the tree is also called the **depth** of the tree. In the following Fig. 7.16, root a is at depth 0, vertices b, c and d are at depth 1 or level 1, vertices e and f are at depth 2 or level 2 and vertex g is at depth 3 or level 3. The height of the tree is 3.

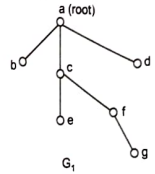


Fig. 7.16

- In a rooted tree, if the level of a vertex y in greater than the level of the vertex x, then we say **y is below x.** If y is below x and there is an edge from x to y, then we say y is the **son** of x. Also, x is said to be the **father** of y. Two vertices are said to be **brothers** if they are sons of the same vertex. If $P = (x, v_1, v_2 \ldots\ldots, v_{n-1}, y)$ is a path from x to y, then y is called a **descendant** of x. Also, x is said to be an ancestor of y. These terms clearly indicate that family trees are indeed rooted trees. To understand these terms, consider rooted tree G_1 in Fig. 7.16. The vertex e is below the vertex c. Hence e is the son of c or c is the father of e. Also e and f are the sons of c. Therefore the vertices e and f are brothers. Also, the vertices c and g are connected by the sequence of edges and g is below c. Therefore, g is a descendant of c or we can say c is an ancestor of g.
- The degree of a tree is the maximum degree of the nodes in the tree. In Fig. 7.16, the degree of the tree is 3.
- A **forest** is a set of disjoint trees. If we remove the root of a tree, we get a forest i.e. set of disjoint trees.

In another words, if the root and corresponding edges connecting the nodes (sons) to the root are deleted from a tree, we obtain a set of disjoint trees. This set of disjoint trees is called a **forest**.

- An **ordered tree** is a tree for which the order of the children of each node considered, is important. Thus, in ordered tree, the children of a node are having some order.

Consider following ordered trees T_1 and T_2. The order of children 2 and 3 is different in both the trees. Hence these two trees are different. Thus $T_1 \neq T_2$

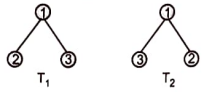


Fig. 7.17

We present now the definitions of subtree and m-ary tree in a rotated tree.

In a rotated tree T, a vertex x, together with all its descendants, is called the **subtree of T rooted at x.** Following Fig. 7.18 shows the rooted tree T, and its subtree T' rooted at b.

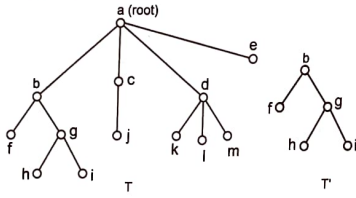


Fig. 7.18

A rooted tree in which every interior node has atmost m sons is called an m-ary tree.

An m-ary tree is said to be **regular m-ary tree** or **full m-ary tree** if every branch node has **exactly** m sons. For example, in the following Fig. 7.19 G_1 is a 2- ary tree or binary tree but it is not a regular binary tree. G_2 is a regular 3 - ary tree or a ternary tree.

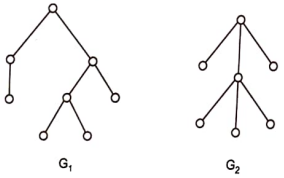


Fig. 7.19

In a regular m-ary tree, the relation between the interior nodes and the total number of nodes is given by following theorem.

Theorem :

A regular m-ary tree with i interior nodes has (mi + 1) nodes at all.

Proof:

Suppose the given regular m-ary tree has n number of vertices, out of which there are i interior vertices.

Clearly, the number of leaves or sons in the tree is t = (n – i).

Since the given graph is a regular m-ary tree hence each of the i interior nodes has m sons and thus the regular m-ary tree will have total (mi) sons.

But root is not a son.

Therefore the given tree has a total of (mi + 1) number of vertices.

Hence $\boxed{n = mi + 1}$

7.4.1 Binary Trees

- Binary trees form an important class of rooted trees. They are extensively used in the sorting procedure, binary identification problems and variable length binary codes. As mentioned earlier, in binary tree, every internal vertex has atmost 2 sons. The vertex to the left of the root is called its left child and the vertex to the right of the root is called its right child. Vertex having left child or right child or both is called parent of both the children. Two vertices having the same parent are called siblings.
- A vertex with no children is a leaf. In binary three no vertex or node can have more than two subtrees. The subtree whose root is the left child of some vertex is called the left subtree of that vertex. Similarly, if the root of the subtree is the right child of the vertex, then the subtree is the right subtree of the vertex. A binary tree is a full or regular binary tree if each internal vertex has **exactly** 2 sons i.e. in a full binary tree there is exactly one vertex of degree 2 and each of the remaining vertices is of degree one or three. The vertex of degree two serves as a root. Fig. 7.20 shows a regular binary tree with root a.

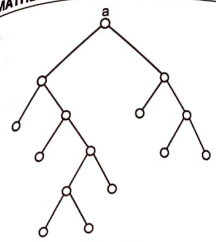


Fig. 7.20

- If 'h' is the height of a binary tree then maximum number of leaves in the binary tree are 2^h and maximum number of nodes are 2^{h+1} -1.
- We now define a very important characteristic of a binary tree called its balance factor. The **balance factor** of a binary tree is the difference in height between its left and right subtrees. If H_L and H_R are the heights of left and right subtrees respectively then the balance factor B of the tree is given by

$$B = H_L - H_R$$

- Consider the binary tree and its left and right subtree shown in Fig. 7.21.

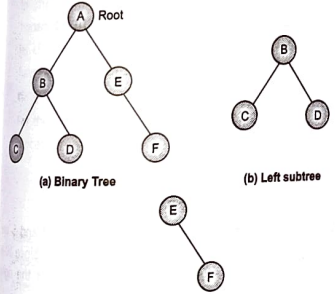


Fig. 7.21

- The balance factor of binary tree (a) is 0.
- A tree is balanced if its balance factor is zero and its subtrees are also balanced.

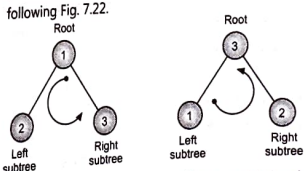
In the next section, traversing of binary trees using depth-first and breadth-first methods is explained.

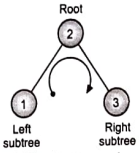
7.4.2 Binary Tree Traversal

- To recover the information stored in the binary tree, it is required to visit all the vertices in the tree. The process in which all the nodes of the tree are visited, is called tree traversal.
- Traversing means processing all the nodes of the tree. A binary tree traversal requires that each node of the tree to be processed once and only once in a predetermined sequence. Depth-first and Breadth-first are two general methods for traversing the binary tree. These are described below.

Depth-First Traversal:

- In this method, the processing proceeds vertically from the root through one child to the most distant descendent of that first child before processing a second child. In other words, we process all of the descendents of a child before going to the next child. There are three standard methods for this, namely preorder traversal, post order traversal and inorder traversal. These traversal algorithms visit every node of the tree in a particular order.
- In the **pre order traversal** the root node is processed first, followed by the left subtree and then the right subtree (**Root, Left, Right**). It processes the node **BEFORE** processing either subtree.
- In the **post order** traversal, it processes first the left-subtree then the right subtree and then, in last the root of the tree (**Left, Right, Root**) . The node is processed **AFTER** visiting both its subtrees.
- The **inorder traversal** processes the left subtree, then the root and finally the right subtree (**Left, Root, Right**). The meaning of prefix "in" is that the root is processed in between the subtrees. It means the node is processed **IN between** the processing of the left subtree and processing of the right subtree.
- In all cases, it is assumed that the left subtree is visited before the right subtree.
- The above three standard traversals are shown in the following Fig. 7.22.





(c) Inorder Traversal

Fig. 7.22: Binary tree traversal

- To understand the concept of preorder, postorder and inorder traversal, consider the following binary tree in Fig. 7.23.

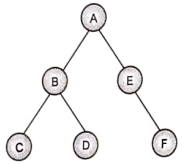
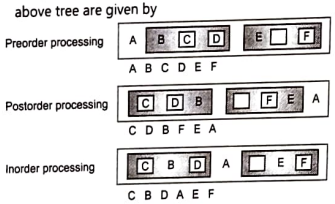


Fig. 7.23

- The preorder, postorder and inorder traversals of the above tree are given by



- Thus for given binary tree, we can find preorder, postorder and inorder traversals corresponding to that tree.
- Also, we can construct the binary tree if preorder and inorder traversals of binary tree are given. The root is found from preorder traversal and left and right subtrees are found from inorder traversal. The process for finding the root and left and right subtrees of required binary tree is explained below:

  Suppose the preorder traversal: 7, 4, 2, 5, 10, 12

  and inorder traversal: 2, 4, 5, 7, 10, 12 are given for the binary tree which is to be constructed .
- Since for preorder traversal, the processing order is **Root, Left, Right**, as explained earlier, we conclude that the first element of preorder traversal from left denotes the root of the tree. Hence 7 is the root of the tree. The left and right subtrees of root 7 will be found from inorder traversal. As in inorder traversal, the processing order is **Left, Root, Right**, the elements 2, 4, 5 left to root 7 in inorder traversal will be in left subtree. The elements 10,12 right to root 7 in inorder traversal will be in right subtree.

  That is, in inorder traversal: 2, 4, 5, **7**, 10, 12

  Left , Root, Right

  The root 7 and It's two subtrees having elements 2, 4, 5 in left subtree and 10, 12 in right subtree are shown below:

Fig. 7.24

Now, we go to next element after 7 in preorder traversal which is element 4 and 4 is in left subtree, hence 4 is root of the left subtree.

Again, for finding left and right subtrees of this new root 4, we consider the sequence 2, 4, 5 in inorder traversal. The element left to 4 is 2 which will be in left subtree rooted at 4. The element right to 4 is 5 which will be in right subtree rooted at 4.

Inorder traversal: 2, **4**, 5, **7**, 10, 12

Left , Root, Right

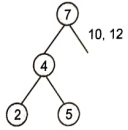


Fig. 7.25

Now, in preorder traversal, after 4, elements 2 and 5 are already processed, so we go to next element 10. Since 10 is in right subtree of root 7, hence 10 is root of the right subtree.

Again, for finding left and right subtrees of this new root 10, we consider the sequence 10, 12 in inorder traversal as before 10, all elements 2, 4, 5 and 7 are already processed. There is no element left to 10, hence there is no left subtree rooted at 10. The element right to 10 is 12 which will be in right subtree rooted at 10.

Inorder traversal: 2, **4**, 5, **7**, **10**, 12

Root, Right

The required binary tree is given by

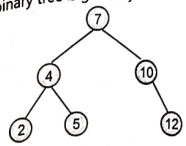


Fig. 7.26

It should be noted that in above procedure, the root is to be found from preorder traversal and left and right subtrees are to be found using inorder traversal. Also, in preorder traversal, we have to move from **left to right** for finding next root of the subtree.

The same procedure has to be followed if postorder and inorder traversals are given for any binary tree. The root has to be found out from postorder traversal and left and right subtrees are to be found out using inorder traversal. For postorder traversal, the processing order is **Left, Right, Root,** as explained earlier, we conclude that the last element of postorder traversal (or first element from right) is the root of the tree. Also, in postorder traversal, we have to move from **right to left** for finding next root of the subtree.

Breadth-First Traversal:

- In breadth-first traversal, the processing proceeds horizontally from the root of all of its children, then to its children's children and so on until all the nodes have been processed. In other words, each level completely processed before the next level is started.

  For the binary tree shown in Fig. 7.27 the Breadth-first process is shown below.

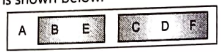


Fig. 7.27

Hence ABECDF will be the processing order for breadth-first traversal.

- After defining binary tree traversal techniques, we now move to one of the important application of binary tree, an expression tree, which we define in next section.

7.4.3 Binary Expression Tree

Algebraic expression can be conveniently expressed by its expression tree used in most of the compilers. An expression tree is a binary tree with the following properties:

1. Each leaf is an operand.
2. The root and interval nodes are operators.

Subtrees are sub expressions, with the root being an operator.

Here we consider only standard arithmetic operators +, –, *, /. An expression having operator can be decomposed into

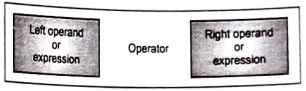


Fig. 7.28

For example, the expression (a + (b c)) – (d (e f)) can be represented as the tree shown in Fig. 7.29.

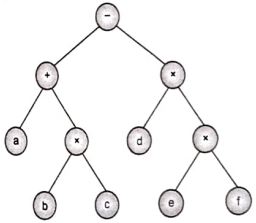


Fig. 7.29

7.4.4 Conversion of General Tree to Binary Tree

- As we know that for general tree, each node can have as many children as it is necessary to satisfy its requirements. For certain applications, it is required to process general trees. It is considerably easier to represent binary trees in programs than it is to represent general trees. In this section, the method to convert general tree to binary tree is explained.
- Consider the tree shown in Fig. 7.30 (a). To convert it into a binary tree, we first identify the branch from the parent to its first or leftmost child. These branches from each parent become left pointers in the binary tree. They are shown in Fig. 7.30 (b). Then we connect sibling, starting with the leftmost or first child, using a branch for each sibling to its right sibling. These branches are shown in Fig. 7.30 (c) and they are the right pointers in the binary tree.

- Now, remove all unneeded branches from the parent to its children. The resulting binary tree is shown in Fig. 7.30 (d). This binary tree is thus obtained by connecting together all siblings of a node and deleting all links from a node to its children except for the link to its leftmost or first child. This binary tree is formed using leftmost or first-child-next-right-sibling relationship.

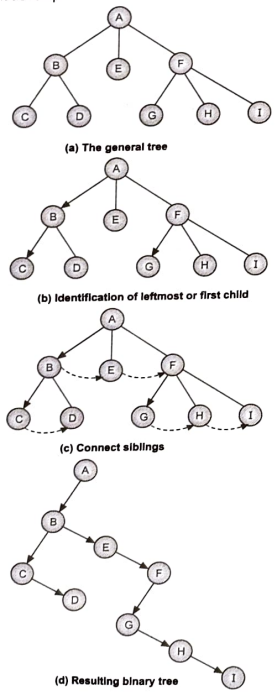


Fig. 7.30

7.4.5 Full Binary Tree and Complete Binary Tree

A binary tree is a full or regular binary tree if each internal vertex has **exactly** 2 sons. Let T be a binary tree with height greater than zero and with root 'a'. Deleting the root 'a' and its two incident edges produces two disjoint binary trees, whose roots are level 1 vertices of T. These disjoint binary trees are called **Left subtree** and **Right subtree** of the root 'a'. The roots of these subtrees are called the left son and the right son of 'a' and the edges which are deleted are called left branch and right branch of 'a' respectively. Consider the full binary tree with the root 'a' in Fig. 7.31. T_1 is the left subtree with root 'b' and T_2 is the right subtree with root 'c' of T respectively. e1 is the left branch and e2 is the right branch of T.

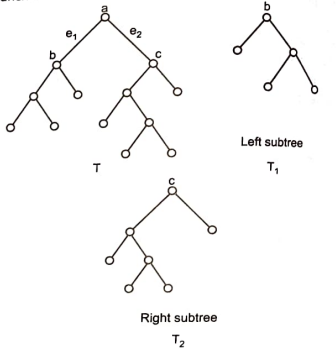


Fig. 7.31

- A **complete binary** tree is a proper binary tree where all leaves have the same level or depth.

The simplest example of a complete binary tree is a single elimination tournament. In the graph of single elimination tournament which is a complete binary tree, the contestants are represented by leaves and the winners by the internal nodes. Eventually, there is a single winner at the root. If the number of contestants is not a power of 2, then some contestants receive byes. The following Fig. 7.32 shows a single elimination tournament with 8 contestants.

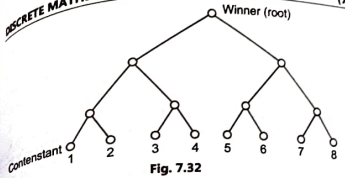


Fig. 7.32

In a complete binary tree, the number of nodes at depth 'd' is 2^d.

If 'd' is the height of a complete binary tree then number of leaves in the complete binary tree is 2^d and total number of nodes are $2^{d+1}-1$.

In Fig. 7.32 which is a complete binary tree, the root of the tree is at depth 0. Hence number of nodes at depth 0 is $2^0=1$ which is a root node itself.

Similarly at depth 1, the number of nodes are $2^1=2$.

At depth 2, the number of nodes are $2^2=4$.

At depth 3, the number of nodes are $2^3=8$.

Here the depth or height of the tree is 3. Therefore, the number of leaves is $2^3=8$. The total number of nodes are $2^{3+1}-1=15$.

7.4.6 Binary Search Tree (BST)

In many **search applications** where data is constantly entering or leaving, a particular kind of a binary tree is used, called a Binary Search Tree (BST). It provides the efficient way of data sorting, searching and retrieving.

A BST is a binary tree where nodes are ordered in the following way:

- each node is associated with one key.(also known as data)
- the keys in the left subtree are less then the key in its parent node, in short L < P.
- the keys in the right subtree are greater the key in its parent node, in short P < R.

It may be noted that in BST, each subtree is itself a binary search tree.

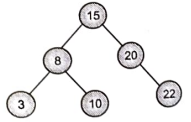


Fig. 7.33

- In the above tree, all nodes in the left subtree of 15 have keys < 15 while all nodes in the right subtree have keys > 15. Because both the left and right subtrees of a BST are again search trees; the above definition is recursively applied to all internal nodes.
- Now we formulate the problem of searching for an item in an ordered list which arise many a time in our daily life. To find a person's telephone number in a telephone directory, to find the record of an employee in a company are some of the examples of a searching problem.
- Consider the problem of searching an item x in the ordered list $k_1, k_2 \ldots k_n$ where $k_1, k_2 \ldots k_n$ are given items called **keys**. Assume $k_1 < k_2 < k_3 \ldots < k_n$. Our problem is to find whether the given item x is equal to one of the keys or whether x falls between k_i and k_{i+1} for some i. For this, we define a search tree for the keys $k_1, k_2 \ldots k_n$ to be a full binary tree with n branch nodes are labelled $k_1, k_2 \ldots k_n$ and the leaves are labelled $k_0, k_1, k_2 \ldots k_n$. For the branch node with the label k_i, its left subtree contains only vertices with labels < k_i and its right subtree contains only vertices with labels ≥ k_i. It is convenient to take the middle of the key from $k_1, k_2 \ldots k_n$ as a root of the tree. The search tree constructed in this way is known as **binary search** tree.

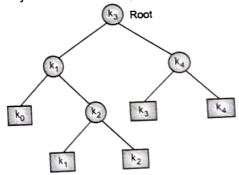


Fig. 7.34

- In the search procedure, we compare a given item x with the label of the root of the tree. If x is equal to the label of the root, the search is completed. If x is less than the label of the root then we shall compare x with the **left son** of the root, and if x is larger than the label of the root, we shall compare x with the **right son** of the root. Such comparisons continue for successive branch nodes until either x matches a key or a leaf is reached.

- Clearly, if a leaf labelled k_j is reached, it means that x is larger than the key k_j but less than the key k_{j+1}. If the leaf k_0 is reached, it means that x is less than k_1 and if the leaf k_n is reached, it indicates that x is larger than k_n.
- For example, consider the keys k_1, k_2, k_3 and k_4 where $k_1 < k_2 < k_3 < k_4$ we construct a binary search tree with the branch nodes k_1, k_2, k_3 and k_4 and with the terminal nodes k_0, k_1, k_2, k_3 and k_4. With k_3 as a root, the binary search tree for the given keys is shown in Fig. 7.34 where circles denote the branch vertices and boxes denote the leaves.
- Now, suppose the keys k_1, k_2, k_3, k_4 represent the items CD, EH, GI and PR respectively and the item x = DD is to be searched in the given keys. The search steps according to Fig. 7.34 will be as follows:

  Step 1: Compare DD with k_3 = GI.

  Step 2: Since DD is less than GI, go to left son of k_3 and compare DD with k_1 = CD.

  Step 3: Since DD is larger than CD, go to right son of k_1 and compare DD with k_2 = EH.

  Step 4: Since DD is less than EH, go to left son of k_2. In this way, the leaf labelled k_1 = CD is reached.
- Thus, we conclude that our given item DD is larger than k_1 = CD and less than k_2 = EH.
- Hence a binary search tree is a binary tree in which all items in the left subtree are less than the parent node, all items in the right subtree are greater than or equal to the parent node and each subtree is itself a binary search tree. Fig. 7.35 shows a binary search tree for key item k.

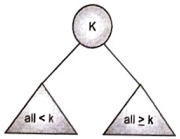


Fig. 7.35: Binary search tree

- Traversing of binary search trees using preorder, post order and inorder traversals are defined as it is defined for binary trees in article 7.4.1. These traversal algorithms visit every node of the BST in a particular order.
- **Inorder traversal** technique lists the nodes of binary search tree in **ascending order**. Hence Inorder traversal yields sorted list of nodes.
- In binary search tree, any item can be inserted or deleted.
- To insert a given item 'k' in a given binary search tree, first compare the item k with the root node. If the item k > root node, proceed to the right child and it becomes the root node for the right subtree. If item k < root node, proceed to the left child. Now, if item k is greater than the node, then the item k is inserted as the right child and if item k is less than the node, then the item k is inserted as the left-child.
- From above, it is clear that the insertion procedure is quite similar to searching. Starting from the root, we recursively go down the tree searching for a location in a BST to insert a new node.

  To explain the procedure, consider the example of inserting in order the integers 20, 3, 10, 22, 15, 35, 9, 2. The resulting binary search tree is given in Fig. 7.36.

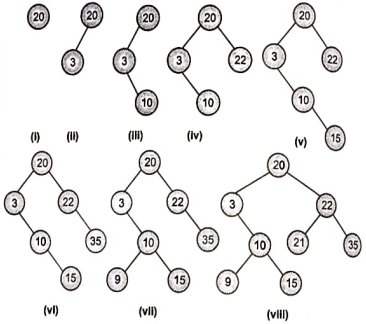


Fig. 7.36

- For deleting any item k from the binary search tree, we have to consider the number of children of the deleted node (k). If deleted node (k) has no children, then replace the node with null. If deleted node (k) has only one child, then replace the value of deleted node with the only child.
- If deleted node has two children, then replace the deleted node with the node that is closest in the value to the deleted node. To find the closest value, we move once to the left and then to the right as far as possible. This node is called immediate predecessor. Now replace the value of deleted node with immediate predecessor and then delete the replaced node by using previous cases. In another words, we find the largest node in the deleted node's left subtree as an immediate predecessor.

To understand the above procedure, consider the following binary search tree in Fig. 7.37.

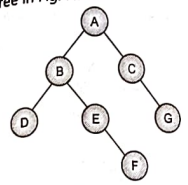


Fig. 7.37

After deleting the node D, the binary search tree will be

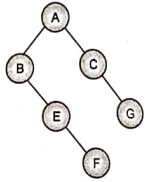


Fig. 7.38

If we delete the node C which has one child, the binary search tree after deletion of C is

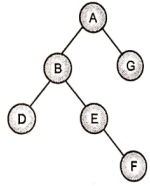


Fig. 7.39

The binary search tree obtained after deletion of root A which has two children is given by

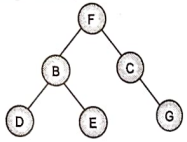


Fig. 7.40

7.4.7 Decision Tree

Another important application of rooted tree is a decision tree which is used to model sorting algorithms and to determine an estimate for the worst-case complexity of these algorithms.

- A rooted tree in which each vertex represents a test question and each descending edge from that vertex represents a possible answer to that question, is called a decision tree.
- Thus in a decision tree, each internal vertex corresponds to a decision, with a subtree at these vertices for each possible outcome of the decision.
- The leaves of the decision tree represent the possible solution or outcomes of the problem.

Consider the example of sorting three distinct numbers A, B and C. For sorting, we can pick any two of the three given numbers, and compare which is larger. For three elements A, B and C, there are 3! permutations possible, that is, there are 3!=6 possible orderings of these three numbers. The sorting algorithm based on binary comparisons can be represented by a binary decision tree in which each internal vertex represents a comparison of two elements. Each leaf represents one of the 3! permutations of 3 elements A, B and C.

For sorting three distinct numbers A, B and C, first we compare two elements say A and B. This will represent the root of the binary decision tree. There are two possibilities, A > B or A < B. These will be represented by two descending edges from the root. When A>B, we will compare A and C to find which element is larger. This is denoted by the internal vertex of decision tree. Two descending edges from this vertex will be A > C and A < C. If A < C then we reach to the leaf C > A > B which is one of the possible outcome. If A > C then again we compare elements B and C represented by the internal vertex of the tree. If B > C then we come to the leaf A > B > C and if B<C then we reach to leaf A > C > B. Similarly when A<B, we compare B and C and then compare A and C to get leaves B > A > C, B > C > A and C > B > A. The decision tree for sorting three distinct numbers A, B and C is shown in following figure.

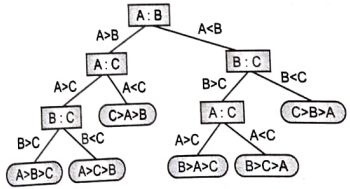


Fig. 7.41

SOLVED EXAMPLES

Example 1: *Draw all rooted tree with four nodes.*

Solution: Each rooted tree with four vertices rooted at 'a' will be isomorphic to one of the following:

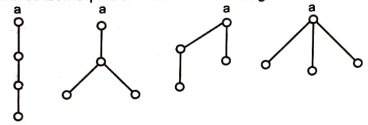


Fig. 7.42

Example 2: *Draw all full binary trees with 7 nodes.*

Solution: Each full binary tree will be isomorphic to one of the following:

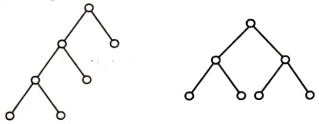


Fig. 7.43

Example 3: *Consider the rooted tree shown in Fig. 7.44.*

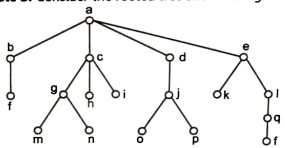


Fig. 7.44

(i) Find the father of c and of h.

(ii) Find the ancestors of f and of j.

(iii) Find the sons of d and of e.

(iv) Find the descendants of d.

(v) How many terminal vertices are there ?

(vi) How many internal vertices are there ?

(vii) Draw the subtree rooted at e.

Solution: (i) The father of c is a and the father of h is c.

(ii) The ancestors of f are b and a. Also the ancestors of j are d and a.

(iii) The son of d is j and the sons of e are k and l.

(iv) The descendant of d are j, o and p.

(v) There are 9 terminal vertices.

(vi) There are 9 internal vertices.

(vii) The subtree rooted at e is shown below:

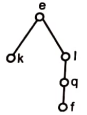


Fig. 7.45

Example 4: *For the following rooted tree, draw the diagram with the root v (degree four vertex) at the top and find out whether it is a full m-ary tree for some 'm' or not.*

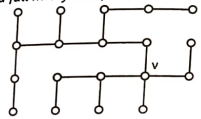


Fig. 7.46

Solution: With the root v (degree four vertex) at the top, the given rooted tree can be drawn as shown in Fig. 7.47.

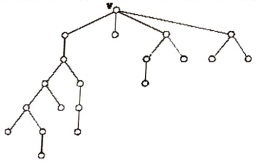


Fig. 7.47

It is a 4-ary tree (because its one internal node v has 4 sons) but it is *not* a full 4-ary tree because each internal node does not have 4 sons exactly.

Example 5: *What is the total number of nodes in a full binary tree with 20 leaves ?*

Solution: Let n represent the total number of nodes in a full binary tree. Then numbers of internal nodes i = (n – 20).

Also, the total number of nodes in a full m-ary tree is given by n = mi + 1.

In full binary tree m = 2.

$\Rightarrow \quad n = 2i + 1$

$\Rightarrow \quad n = 2(n - 20) + 1$

$\Rightarrow \quad n = 2n - 40 + 1$

$\Rightarrow \quad n = 39$

Hence a full binary tree with 20 leaves has total 39 nodes.

Example 6: *Does there exist a ternary tree with exactly 21 nodes ?*

Solution: Here the given tree is a ternary tree $\Rightarrow m = 3$, also $n = 21$.

Hence,

$n = mi + 1$

$\Rightarrow \quad 21 = 3i + 1$

$\Rightarrow \quad 20 = 3i$

The solution to the above equation is not an integer. Thus there does not exist any ternary tree with 21 nodes.

Example 7: *If there are 60 contestants in a single elimination tournament, how many matches are played ?*

Solution: A single elimination tournament can be represented by a full binary tree, in which contestant represent terminal vertices and winners, the internal vertices. According to the problem, there are 60 contestants (leaves). Let i be the number of internal vertices. Hence the total number of vertices

$n = 60 + i.$

Now, by theorem 8.3.1

$(60 + i) = 2i + 1$

$\Rightarrow \quad i = 59$

Hence 59 matches are played in the tournament.

Example 8: *Show that the total number of vertices in a full binary tree is always odd.*

Solution: We know that in a full binary tree, there is exactly one vertex of degree even (root) and the remaining (n – 1) vertices are of odd degrees. We know that the number of vertices of odd degree in any graph is always even. Thus (n – 1) is even. Hence n is odd.

Example 9: *Find the preorder, postorder and inorder traversal of the following tree:* ***(Oct. 2017)***

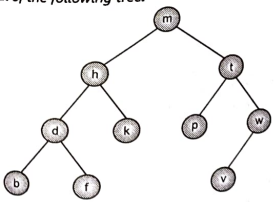
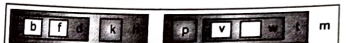


Fig. 7.48

Solution: The preorder, postorder and inorder traversals are as follows:

Preorder traversal:

Postorder Traversal:



Inorder Traversal:

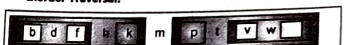


Fig. 7.49

Example 10: *Determine the preorder, post order and inorder traversal of the binary tree shown in Fig. 7.50.*

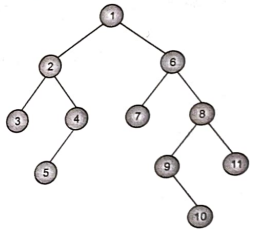
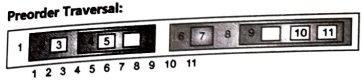


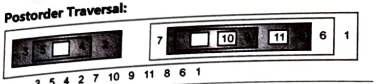
Fig. 7.50

Solution: The preorder, postorder and inorder traversals for the given tree are given below.

Preorder Traversal:



Postorder Traversal:



Inorder Traversal:

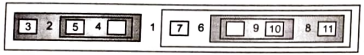


Fig. 7.51

Example 11: *Write and evaluate the expression tree shown below.*

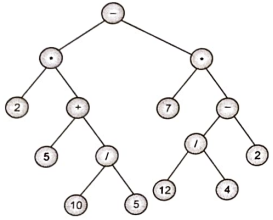


Fig. 7.52

Solution: The algebraic expression given by the expression tree is as follows:

$$\Big(\big((2) * \big((5) + ((10)/(5))\big)\big) - \big((7) * \big(((12)/(4)) - 2\big)\big)\Big)$$

It's value is (14) – (7) = 7.

Example 12: *Construct the labeled tree of the following algebraic expression:*

$$\big(((x + y) * z)/3\big) + (19 + (x * x))$$

Solution: The expression tree for the given algebraic expression is given below:

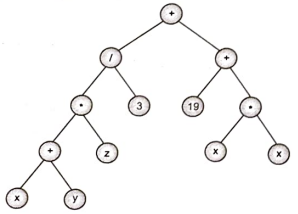


Fig. 7.53

Example 13: *Convert the following tree into binary tree:*

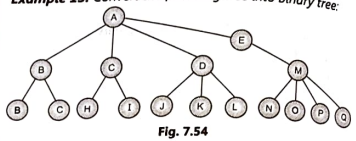
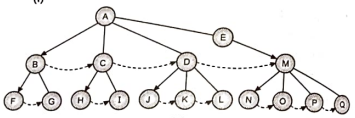


Fig. 7.54

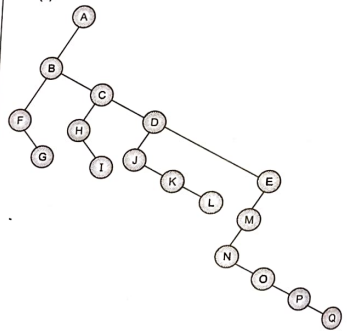
Solution: The steps involve in convert the given tree into a binary tree are as follows:

(i)



(a)

(ii)



(b)

Fig. 7.55

The binary tree is shown in Fig. 7.55 (b).

Example 14: *A binary tree T has 9 nodes. Draw the diagram of T if preorder and inorder traversals of tree are as follows:*

Preorder : G B Q A C P D E R

Inorder : Q B C A G P E D R

Solution: For constructing binary tree with given preorder and inorder traversals, the root is to be found from preorder traversal and left and right subtrees are to be found from inorder traversal. For preorder traversal, the processing order is **Root, Left, Right**. The first element of preorder traversal from left denotes the root of the tree. Hence **G** is the root of the tree.

The left and right subtrees of root G will be found from inorder traversal. As in inorder traversal, the processing order is **Left, Root, Right**, the elements left to root G in inorder traversal Q, B, C, A will be in left subtree. The elements right to root G in inorder traversal P, E, D, R will be in right subtree.

The next element in preorder traversal is B and B is in left subtree, so **B** will be the root of left subtree. Proceeding in the similar way, all the left and right subtrees with their roots can be found out. The required binary tree is shown as follows:

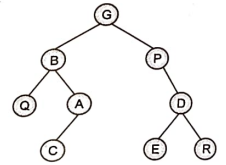


Fig. 7.56

Example 15: *Consider the algebraic expression*

$$E = (2x + y) * (5a - b)^3$$

(i) Find and draw tree corresponding to above expression.

(ii) Write all three traversals for the tree so formed.

Solution: (i) An expression tree is a binary tree where each leaf is an operand, the root and interval nodes are operators and subtrees are sub expressions, with the root being an operator. For the given expression, the required tree is drawn below:

(In following figure, the circle next to + circle should be filled with symbol ↑)

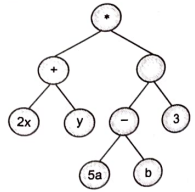


Fig. 7.57

(ii) All three traversals Preorder, Postorder and Inorder traversals for the expression tree are given below:

Preorder Traversal: * + 2x y ↑ – 5a b 3

Postorder Traversal: 2x y + 5a b – 3 ↑ *

Inorder Traversal: 2x + y * 5a – b ↑ 3

Example 16: *A telephone network is established among 100 people. Information received by the first person is passed along to the 99 others as follows; the first person calls exactly 3 people, and each of these people calls 3 others, and so on until there are no others to call. If each call takes 5 minutes, how long does it take for a message to be relayed from the first person to receive the message to everyone else ? How many people make no calls ?*

Solution: The given situation can be represented by a tree with a total of 100 nodes, where each node represents a person in the telephone network. Also, since each person calls exactly 3 people, hence each node has 3 sons. Therefore, the given tree is a full ternary tree. If there are i internal nodes then according to theorem.

$$100 = 3i + 1$$

$$\Rightarrow \quad i = 33$$

i.e. there are 33 interior nodes and thus there are 100 – 33 = 67 terminal nodes (leaves). Therefore 67 persons do not make any call. Now, the first person which is serving as the root will send the message to exactly 3 persons. Also each call takes 5 minutes. Hence after 5 minutes, 1 + 3 = 4 people have received the message. Each of these 3 people will call again 3 people. After 10 minutes, the total number of persons getting the message is 1 + 3 + 9 = 13. Continuing in this way, after 15 minutes 1 + 3 + 9 + 27 = 40 and after 20 minutes 1 + 3 + 9 + 27 + 81 = 121 persons could get the message. But we have only 100 nodes. Thus it takes 20 minutes for the message to be relayed to everyone.

Example 17: *Find the number of leaves in a full binary tree with n vertices.*

Solution: In a full binary tree, the number of internal nodes i is given by

$$n = 2i + 1$$

$$\Rightarrow \quad i = \left(\frac{n-1}{2}\right)$$

Hence the number of leaves $t = (n - i)$

$$t = n - \left(\frac{n-1}{2}\right)$$

$$t = \frac{n+1}{2}$$

Hence a full binary tree contains $\left(\frac{n+1}{2}\right)$ number of leaves.

Example 18: *19 lamps are to be connected to a single electrical outlet, using extension chords, each of which has 4 outlets. Find the number of extension chords needed and draw the corresponding tree.*

Solution: Represent lamps by the leaves and the extension chords by the branch nodes of the tree. Since there are 19 lamps to be connected and each extension chords has 4 outlets, this means the tree has 19 leaves and each branch node has 4 sons. Hence the tree is a full 4-ary (quaternary) tree with 19 leaves.

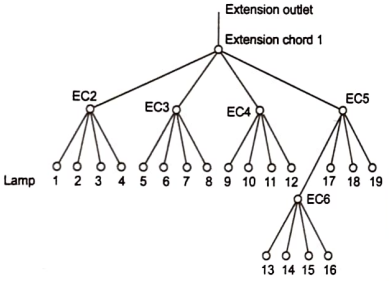


Fig. 7.58

Now, total number of vertices in a full 4-ary tree n = 4i + 1, where i is the number of branch nodes.

But n = 19 + i

Hence 19 + i = 4i + 1

⇒ i = 6

Hence, 6 extension chords are required to connect 19 lamps with a single outlet. The required tree is shown below in Fig. 7.58.

Example 19: *Consider the tree as shown in Fig. 7.59 (i) Which of the vertices are cut points ? (ii) Find all the vertices at level three, if the vertex picked as a root is U and W.*

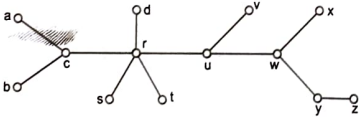


Fig. 7.59

Solution: (i) We know that every vertex of degree more than one is a cut point in a tree, hence in the given tree, c, r, u, w, y are cut points.

(ii) Consider the given tree with root as U. Since the level of the vertex is its distance from the root, the vertices of level 3 are a, b and z.

If we take w as a root of the tree in Fig. 7.59 then, the vertices of level 3 are d, s, t, c.

Example 20: *A binary search tree generated by inserting integers in order 50, 15, 62, 5, 20, 58, 91, 3, 8, 37, 60, 24. Determine the number of nodes in left and right.*

Solution: The steps involved in drawing the resultant binary search tree in Fig. 7.60.

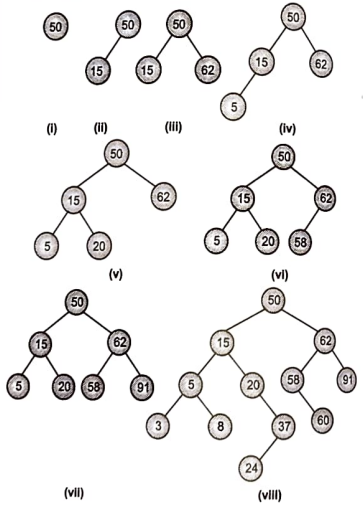


Fig. 7.60

Example 21 : *Given a sequence of numbers:*

11, 6, 8, 19, 4, 10, 5, 17, 43, 49, 31

Draw a binary search tree by inserting the above numbers from left to right. Also draw the resulted tree after removal of 11.

Solution: The resultant binary search tree with given sequence of numbers is given below:

(Modify following two graphs- Draw the small shaded circles as drawn in Fig 7.61. Also lines should be thin not bold and thick. Give figure numbers also.)

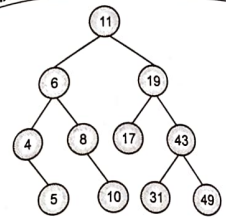


Fig. 7.61

The resultant binary search tree after removal of 11 is shown below:

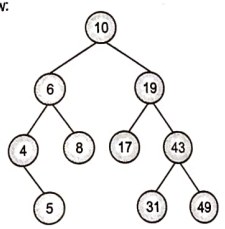


Fig. 7.62

Example 22: *Construct a Binary search Tree by inserting the following sequence of numbers 10, 12, 5, 4, 20, 8, 7, 15, and 13. Also Find Preorder, inorder and Postorder traversal of Binary Search Tree.* ***(May 2018)***

Solution: The resultant binary search tree for given sequence of numbers *10, 12, 5, 4, 20, 8, 7, 15, and 13* is given below:

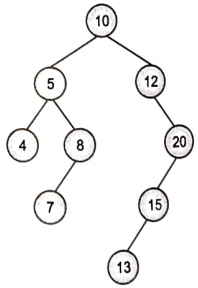


Fig. 7.63

The preorder, postorder and inorder traversal of above Binary Search Tree are as follows:

Preorder traversal: 10, 5, 4, 8, 7, 12, 20 ,15, 13

Postorder traversal: 4, 7, 8, 5, 13, 15, 20, 12, 10

Inorder traversal: 4, 5, 7, 8, 10, 12, 13, 15, 20

Example 23: *Consider binary search tree T given below.*

(i) Suppose nodes 20, 55 and 88 are added one after the other to T. Find the final tree.

(ii) Suppose nodes 22, 25 and 75 are deleted one after the other from T. Find the final tree.

[Show steps and cases]

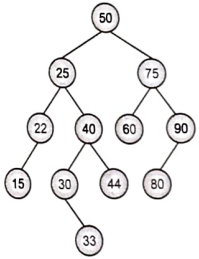


Fig. 7.64

Solution: (i) After insertion of nodes 20, 55 and 88 in the given binary search tree, the final tree will be as follows:

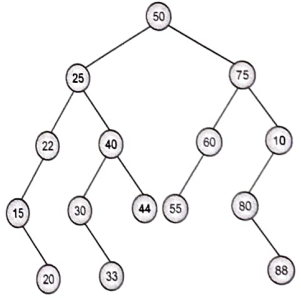


Fig. 7.65

(ii) Given nodes 22, 25 and 75 are deleted one after the other from T. First the node 22 is deleted from T. Since node 22 has only one child (node 15), replace the value of deleted node 22 with the only child node 15. The tree with node 22 deleted and replaced by node 15 is shown below:

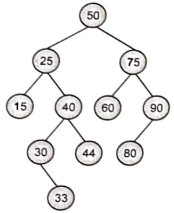


Fig. 7.66

Next, the node 25 is deleted from above tree. The node 25 has two children. The largest node in the deleted node's left subtree is 15.

Replace the value of deleted node 25 with node 15. The tree with node 25 deleted and replaced by node 15 is shown below:

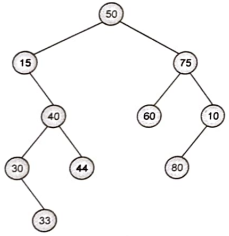


Fig. 7.67

Next, the node 75 is deleted from above tree. The node 75 has two children. The largest node in the deleted node's left subtree is 60.

Replace the value of deleted node 75 with node 60. The tree with node 75 deleted and replaced by node 60 is shown below:

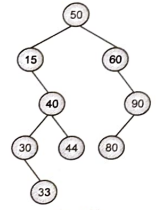


Fig. 7.68

This is the final tree with nodes 22, 25 and 75 deleted one after the other from original tree T.

7.5 PREFIX CODES

A set of sequences is said to be a **prefix code** if no sequences in the set is a prefix of another sequence in the set. For example, the set {01, 10, 11, 000, 001} is a prefix code. The set {1, 00, 01, 000, 0001} is **not** a prefix code because the sequence 00 is a prefix of the sequence 0001.

To obtain a prefix code, we use a full binary tree. For a given full binary tree, we label the two branches incident from each internal node with 0 and 1. For the left branch, we assign 0 and for the right branch we assign 1.

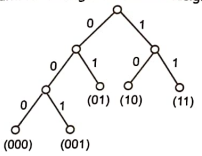


Fig. 7.69

Assign to each leaf of sequence of 0's and 1's which is the sequence of labels of the edges in the path from the root to that leaf. For example, Fig. 7.69 shows a full binary tree and the sequences assigned to its leaves.

Prefix code for the above Fig. 7.69 is {000, 001, 01, 10, 11}. It is clear that the set of sequences assigned to the leaves in any full binary tree is a prefix code.

7.5.1 Optimal Tree

Let T be any full binary tree and let $w_1, w_2, \ldots, w_t$ be the weights of the terminal vertices (leaves). Then, the weight W of the binary tree is given by

$$W = \sum_{i=1}^{t} w_i l_i$$

where l_i is the length of the path of the leaf i from the root of the tree. The full binary tree is called an **Optimal Tree** if its weight is **minimum**.

For instance, suppose 3, 4 and 5 are the weights of the leaves in a full binary tree. Fig. 7.70 shows two such binary trees T_1 and T_2.

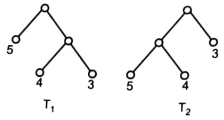


Fig. 7.70

The weight $w(T_1)$ of the tree T_1 in Fig. 7.70 is

$$w(T_1) = 3 \times 2 + 4 \times 2 + 5 \times 1$$
$$= 6 + 8 + 5$$
$$= 19$$

while the weight w (T_2) of the tree t_2 in Fig. 7.70 is

$$w(T_2) = 3 \times 1 + 4 \times 2 + 5 \times 2$$
$$= 3 + 8 + 10$$
$$= 21$$

Here $w(T_1) < w(T_2)$.

Thus T_1 is an optimal tree for the weights 3, 4 and 5.

Optimal trees are used in constructing variable length binary codes, where the letters of the alphabet (A, B, C ..... Z) are represented by binary digits. Since different letters have different frequencies of occurrence (frequencies are interpreted as weights $W_1, W_2 \ldots W_{26}$), a binary tree with minimum weighted path length (optimal tree) corresponds to a binary code of minimum cost.

7.5.2 Huffman Algorithm to find an Optimal Tree

Let $w_1, w_2, \ldots, w_t$ be the weights of the leaves and it is required to construct an optimal binary tree. The following algorithm gives the required optimal binary tree.

Step 1: Arrange the weights in increasing order.

Step 2: Consider two leaves with the minimum weights w_1 and w_2. Replace these two leaves and their father by a leaf and assign to this new leaf the weight $w_1 + w_2$.

Step 3: Repeat the step 2 for the weights ($w_1 + w_2$), w_3, w_4 ... w_t until no weight remains.

Step 4: The tree obtained in this way is an optimal tree for given weights, and stop.

7.5.3 Optimal Prefix Code

A binary prefix code obtained from a optimal tree is called an **optimal prefix code**. For example, consider an optimal tree for the weights 2, 6, 7 and 9 in Fig. 7.71.

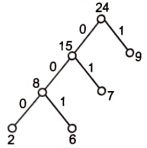


Fig. 7.71

The optimal prefix code for the weights 2, 6, 7 and 9 is given by

{000, 001, 01, 1}

SOLVED EXAMPLES

Example 1: *Obtain the prefix code of the following full binary tree in Fig. 7.72.*

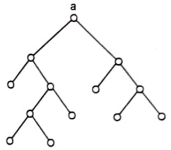


Fig. 7.72

Solution: Given 'a' is the root of the full binary tree. Assign 0 to the left son and 1 to the right son of each branch node of the tree.

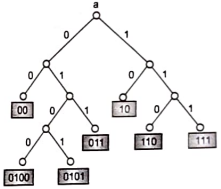
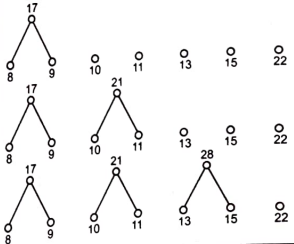


Fig. 7.73

The code words for the leaves obtained in this way are shown in the boxes. Hence, the prefix code for the given tree is {00, 10, 011, 110, 111, 0100, 0101}.

Example 2: Construct an optimal tree for the weights 8, 9, 10, 11, 13, 15, 22. Find the weight of the optimal tree.

Solution: The construction of an optimal tree for the weights 8, 9, 10, 11, 13, 15 and 22 is shown below.



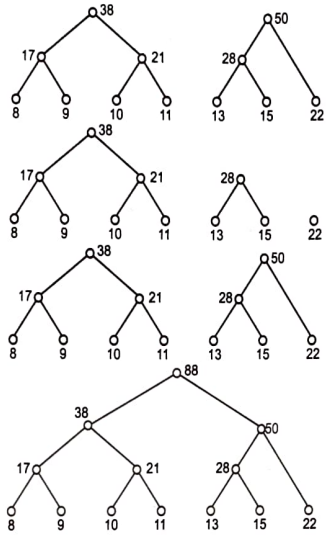


Fig. 7.74

The weight of the tree = 8 × 3 + 9 × 3 + 10 × 3 + 11 × 3 + 13 × 3 + 15 × 3 + 22 × 2 = 242.

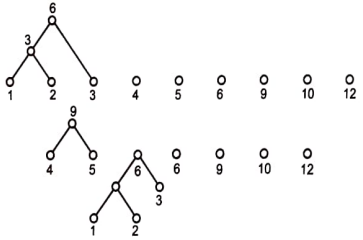
Example 3: *For each of the following sets of weights construct an optimal binary prefix code. For each weight in the set give the corresponding code word:*

(i) 1, 2, 3, 4, 5, 6, 9, 10, 12

(ii) 10, 11, 14, 16, 18, 21.

(iii) 5, 7, 8, 15, 35, 40.

Solution: Optimal binary prefix code for data 1, 2, 3, 4, 5, 6, 9, 10, 12 is as shown below.



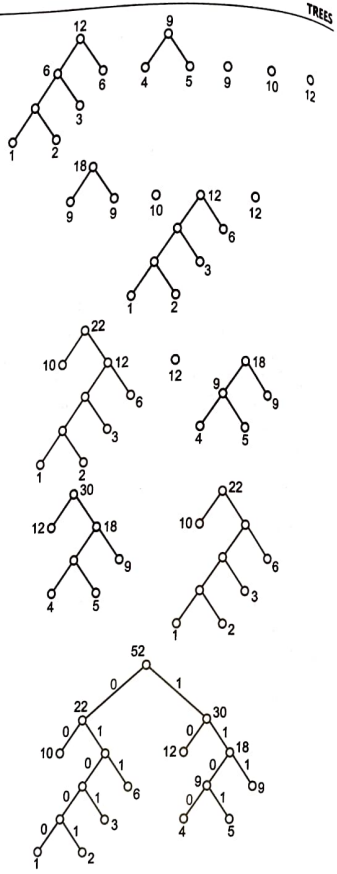


Fig. 7.75

The code words for the leaves are

1	01000
2	01001
3	0101
4	1100
5	1101
6	011
9	111
10	00
12	10

(ii) The optimal binary prefix code for 10, 11, 14, 16, 18, 21 is given as follows:

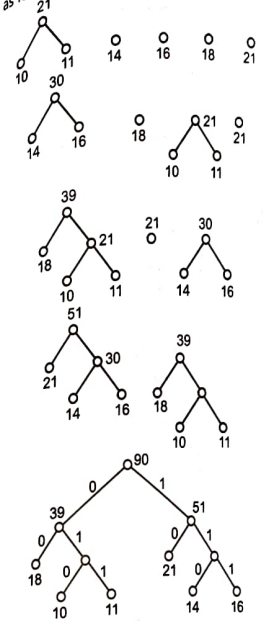
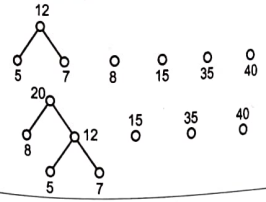


Fig. 7.76

The code words for leaves are

18	00
10	010
11	011
21	10
14	110
16	111

(iii) For the weight 5, 7, 8, 15, 35, 40, the optimal binary prefix code is given by following tree:



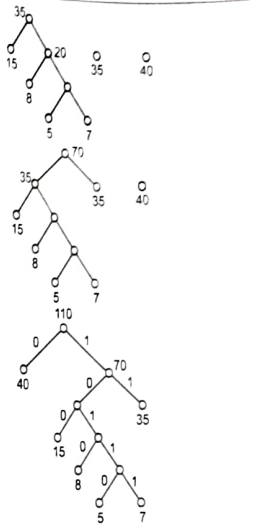


Fig. 7.77

The code words are

40	0
15	100
8	1010
5	10110
7	10111
35	11

Example 4: *For the following sets of weights, construct optimal binary prefix code. For each weight in the set, give the corresponding code words: 8, 9, 12, 14, 16, 19.*

Solution: Optimal binary prefix code for weights 8, 9, 12, 14, 16, 19 is as shown in following figure:

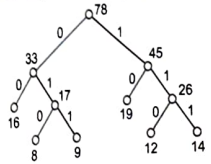


Fig. 7.78

Code word for 8	010	Code word for 14	111
Code word for 9	011	Code word for 16	00
Code word for 12	110	Code word for 19	10

Example 5: *A secondary storage media contains information of files with different formats. The frequency of different types of files is as follows:*

Exe (20), bin (75), bat (20), jpeg (85), dat (51), doc (32), sys (26), c (19), cpp (25), bmp (30), avi (24), prj (29), lst (35), zip (37).

Construct the Huffman code for this.

Solution: The steps involved in constructing the Huffman code for the given frequencies are shown below:

(i)

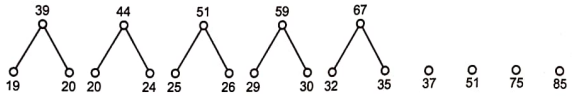


Fig. 7.79

(ii)

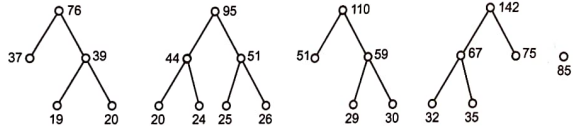


Fig. 7.80

(iii)

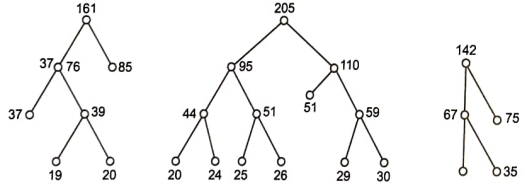


Fig. 7.81

(iv)

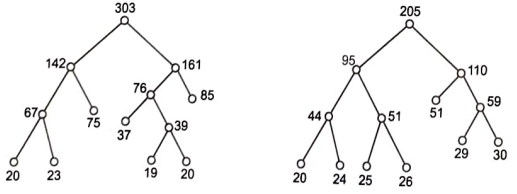


Fig. 7.82

(v) The optimal tree is

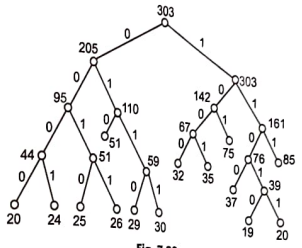


Fig. 7.83

The Huffman code for frequency

20	0000
24	0001
25	0010
26	0011
51	010
29	0110
30	0111
32	1000
35	1001
75	101
37	1100
19	11010
20	11011
85	111

Example 6: *Suppose data items A, B, C, D, E, F, G occur with the following probability distribution:*

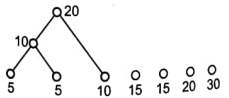
Data Item	A	B	C	D	E	F	G
Probability	10	30	05	15	20	15	05

Construct a Huffman code for the data. What is the minimum weighted path length ?

(April 2019, Oct. 2018)

Solution: The steps involved in constructing the Huffman code for the given data are as follows:

(i)



(ii)

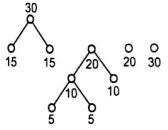


Fig. 7.84

(iii) (iv)

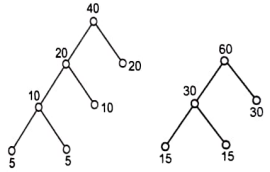


Fig. 7.85

(iv)

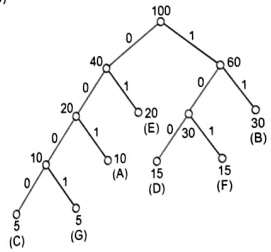


Fig. 7.86

The Huffman code for the given data is

05	0000 and 0001
10	001
15	100 and 101
20	01
30	11

The minimum weight of the tree is

$2(5 \times 4) + 10 \times 3 + 20 \times 2 + 2(15 \times 3) + 30 \times 2 = 260$

The minimum weight path length for the vertices in optimal tree is

A	3
B	2
C and G	4
D and F	3
E	2

Example 7: *State whether the given set is a prefix code. Justify {000, 001, 01,10, 11}*

Solution: The given set which contains 5 elements will be a prefix code if we can construct a full binary tree with five leaves.

Let n be the total number of vertices and i be the total number of interior vertices in a full binary tree with five leaves.

then, $n = i + 5 \Rightarrow i = n\_5$

Also, $n = 2(n-5) + 1 \Rightarrow n = 9$

Therefore, $i = 4$

Hence, the full binary tree with five leaves will have total a vertices and 4 interior nodes. One such full binary tree is

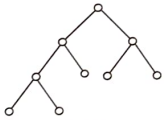


Fig. 7.87

Now assign 0 to the left branch and 1 to the right branch we get

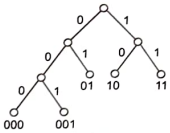


Fig. 7.88

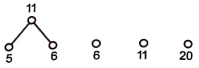
Since we can construct the full binary tree with leaves as prefix codes. The given set is in prefix code.

Example 8 : *For the following set of weights construct optimal binary prefix code.*

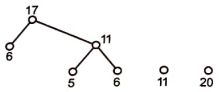
α	5
β	6
γ	6
δ	11
	20

Solution : The steps involved in constructing the optimal binary prefix code are as follows :

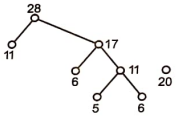
(i)



(ii)



(iii)



(iv)

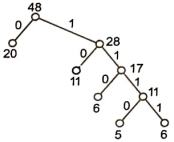


Fig. 7.89

The Huffman code for given data as :

α (5)	1110
β (6)	1111
γ (6)	110
δ (11)	10
(20)	0

7.6 SPANNING TREES

In this section, we study the definition of spanning tree and the algorithms to find a minimum spanning tree in a connected graph.

Let G be any connected graph. A **spanning subgraph** of G which is a tree is called a **Spanning Tree** of G, i.e. a subgraph of G, which is a tree and contains all the vertices of G is called a spanning tree of G.

It is obvious that there can be many spanning trees for a given connected graph. For instance, consider the graph G in Fig. 7.90, two spanning trees of G are T_1 and T_2.

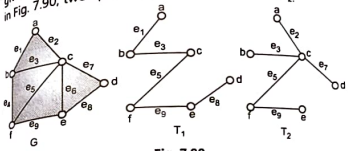


Fig. 7.90

Consider the spanning tree T_1 of G in Fig. 7.90. The edges of spanning tree T_1 are called **branches** of T_1. The edges of G which are **not** in T_1 are called **chords** of T_1. For example, the branches of T_1 are e_1, e_3, e_5, e_8 and e_9 and the chords of T_1 are e_2, e_4, e_6 and e_7. Similarly for the spanning tree T_2 of G in Fig. 7.90 the branches of T_2 are e_2, e_3, e_5, e_7 and e_8 and the chords of T_2 are e_1, e_4, e_6 and e_8. This shows that branches and chords corresponding to different spanning trees are different.

Suppose the connected graph G contains v number of vertices and e number of edges. Therefore, its spanning tree will contain same number of vertices v and (v – 1) number of edges. Hence, there are total (v – 1) branches in any spanning tree of G. Thus, the number of chords will be given by e – (v – 1) = e – v + 1.

7.6.1 Minimum Spanning Tree

- A spanning tree of a **weighted connected** graph G is called a **minimum spanning tree** if its weight is minimum. Minimum spanning trees are useful in building a network of roads or railway lines connecting a numbers of cities. Suppose there are n cities v_1 v_2 ... v_n and suppose the cost c_{ij} of constructing a direct link between the cities v_i and v_j are known. The connector problem is to design a network that connects all the n stations such that the total cost of construction is least. This problem can be represented by a weighted connected graph as follows.
- Represent each city as a vertex v_i of a weighted graph G and the cost c_{ij} of building the road as the weight of the edge (v_i, v_j) joining the vertices v_i and v_j. Then the connector problem reduces to find a spanning tree with the minimum weight in a weighted connected graph G.
- Two algorithms, Prim's algorithm and Kruskal's algorithm are discussed here to find the minimum spanning tree.

7.6.2 Prim's Algorithm

Let G = (V, E) be a connected weighted graph. Construct a minimal spanning tree T for G inductively as follows:

Step 1: Take a vertex v_0 in the graph G.

Set $T = \{\{v_0\}, \Phi\}$

Step 2: Find the edge $e_1 = (v_0, v_1)$ in E such that its one end vertex v_0 is in T and its weight is minimum.

Adjoin the vertex v_1 and the edge e_1 to $T_T = \{\{v_0, v_1\}, \{e_1\}\}$.

Step 3: Choose the next edge $e_i = (v_i, v_j)$ in such a way that its one end vertex v_i is in T and the other end vertex v_j is not in T (i.e. e_i should not form the circuit with the edges in T) and the weight of the edge e_i is as small as possible. Adjoin the edge e_i and vertex v_j to T.

Step 4: Repeat the step 3 until T contains all the vertices of G. The set T will give the minimum spanning tree of the graph G.

Another algorithm due to Kruskal (1956) is described here to find a minimum spanning tree in a weighted connected graph G.

7.6.3 Kruskal Algorithm

Let G = (V, E) be a weighted connected graph.

Step 1: Pick up an edge e_i of G such that its weight w (e_i) is minimum.

Step 2 : If edges $e_1, e_2 \ldots e_k$ have been chosen, then pick an edge e_{k+1} such that $e_{k+1} \neq e_i$ for any i = 1, 2...k

(ii)The edges $e_1, e_2 \ldots e_k, e_{k+1}$ do not form the circuit.

(iii) The weight w (e_{k+1}) is as small as possible subject to condition (ii).

Step 3: Stop, when (step 2) cannot be implemented.

Kruskal algorithm is very similar to Prim's algorithm. In Prim's algorithm, a single tree keeps on growing until it becomes the minimum spanning tree whereas Kruskal's algorithm "grows" a forest of trees. In both the algorithms, in each iteration, minimum weight edge is added which does not cause a cycle which shows that both algorithms are greedy algorithms.

Few examples which show the working of Prim's and Kruskal's algorithms (for minimal spanning tree) are illustrated here.

SOLVED EXAMPLES

Example 1: *Find all the spanning trees for the following graph.*

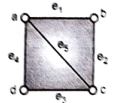


Fig. 7.91 (a)

Solution: Following are the different spanning trees of the given graph.

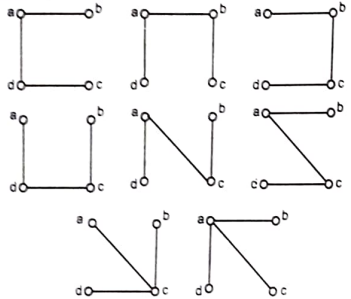


Fig. 7.91 (b)

Example 2: *Find the minimal spanning tree for the following graph by using Prim's algorithm.*

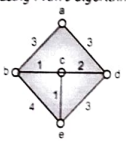
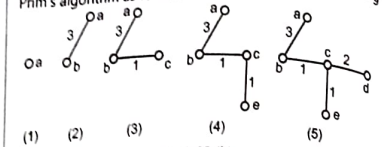


Fig. 7.92 (a)

Solution: Starting from the vertex a, the minimum spanning tree of the given graph can be obtained by using Prim's algorithm as follows:



(1) (2) (3) (4) (5)

Fig. 7.92 (b)

The graph T obtained in step 5 is a minimum spanning tree.

Its weight is 7.

Example 3: *Use Prim's algorithm to construct a minimal spanning tree for the weighted graph in Fig. 7.93 starting from the vertex a. Repeat the process starting from the vertex b. Verify that both trees have the same weight.*

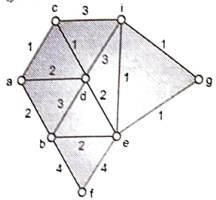
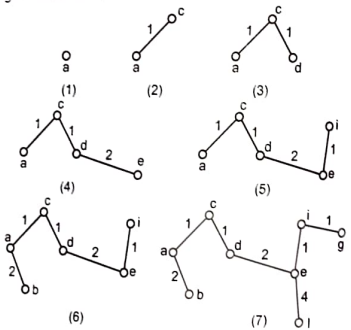


Fig. 7.93

Solution: With the help of Prim's algorithm, the minimum spanning tree of the graph in Fig. 7.94, starting from a is given as follows:



(1) (2) (3) (4) (5) (6) (7)

Fig. 7.94

The total weight of the minimum spanning tree is 12.

Now, use the same procedure, starting from the vertex b.

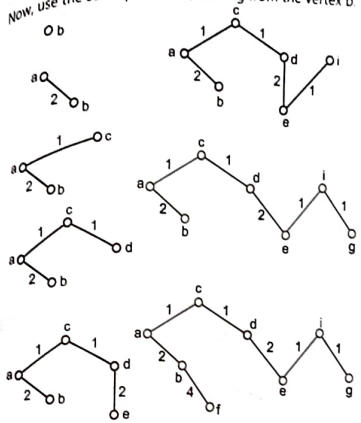


Fig. 7.95

The total weight of the above minimum spanning tree is also 12. Hence both the trees obtained in Fig. 7.94 and Fig. 7.95 have the same weight.

Example 4: *Use Kruskal algorithm to find a minimum spanning tree for the graph shown in Fig. 7.96.*

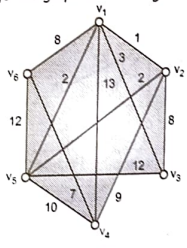
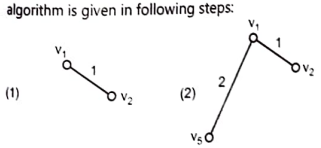
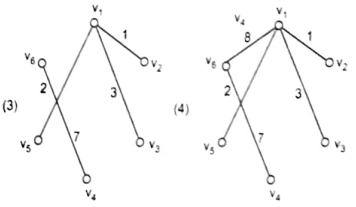


Fig. 7.96

Solution: The minimum spanning tree by using Kruskal's algorithm is given in following steps:



(1) (2)



(3) (4)

Fig. 7.97

Now at this stage we cannot take any edge from G because otherwise edges will form a circuit. Hence the graph in Fig. 7.97 is a minimal spanning tree.

Example 5: *Determine a minimum spanning tree for the graph shown below.*

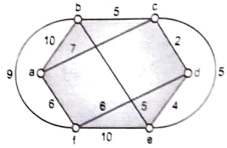


Fig. 7.98

Solution: Using Prim's algorithm we obtain a minimum spanning tree for the given graph as follows:

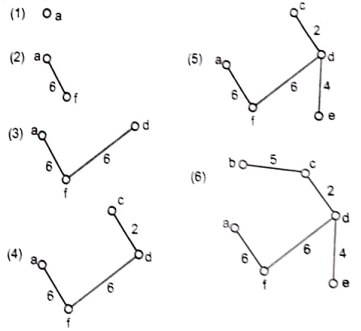


Fig. 7.99

The above tree T is a minimum spanning tree for the given graph.

Its weight is 23.

Example 6: *Find the minimum spanning tree for Fig. 7.100.*

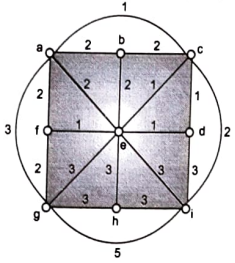


Fig. 7.100

Solution: With the help of Prim's algorithm, the minimum spanning tree of the graph shown in Fig. 7.100 can be obtained as follows:

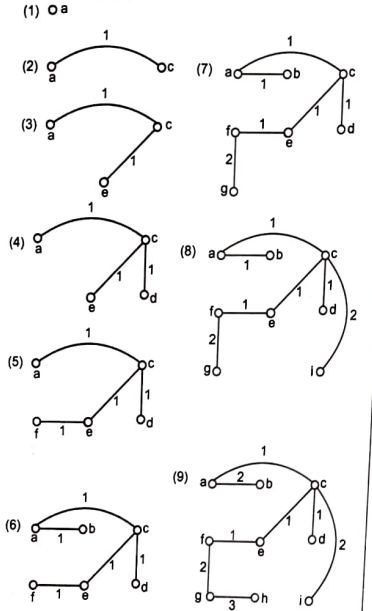


Fig. 7.101

The graph T shown in Fig. 7.101 is a minimum spanning tree. Its weight is 13.

Example 7: *Obtain the minimum spanning tree for the graph shown in Fig. 7.102. Obtain the total cost of minimum spanning tree.*

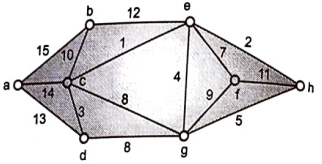


Fig. 7.102

Solution: Using *Kruskal* algorithm, the minimum spanning tree is obtained as follows:

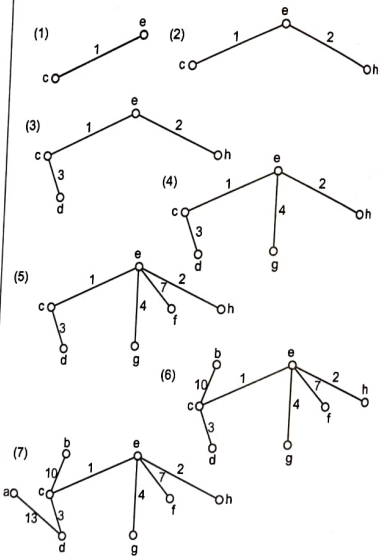


Fig. 7.103

The graph T is a minimum spanning tree of the given graph. Its total cost is 40.

Example 8: *Give the stepwise construction of minimum spanning tree for the following graph using Kruskal's algorithm.*

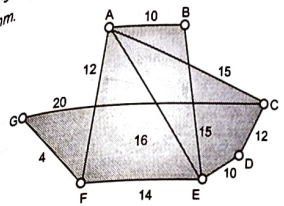


Fig. 7.104

Solution: Using Kruskal's algorithm, the minimum spanning tree can be obtained as follows:

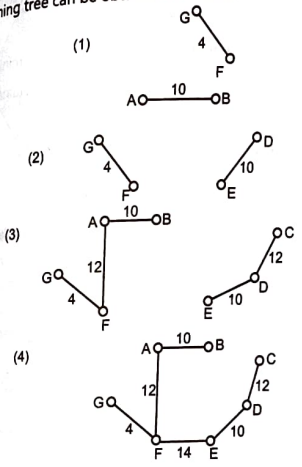


Fig.7.105

Example 9: *Use Kauskal's algorithm to find minimum spanning tree for the graph shown in following Fig. 7.106.* **(Dec. 2007)**

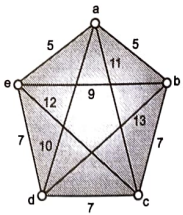


Fig. 7.106

Solution: The minimum spanning tree by using Krushal's algorithm is as follows:

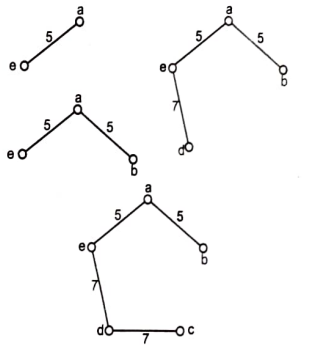


Fig. 7.107

The above tree T is a minimum spanning tree for the given graph. Its weight is 24.

Example 10: *Use Prim's algorithm to find the minimum spanning tree for graph given below.*

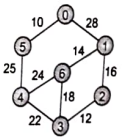


Fig. 7.108

Solution: Using Prim's algorithm, the minimum spanning tree can be obtained as shown in following steps:

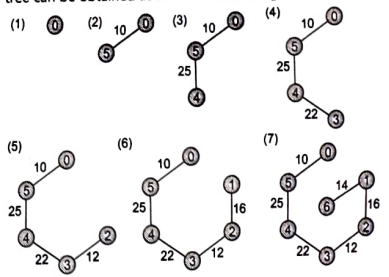


Fig. 7.109

The tree T is a minimum spanning tree with weight 99.

Example 11: *State whether the following statements are TRUE or FALSE.*

- *A spanning tree of graph G includes all the vertices of G.*
- *Preorder traversal of Binary Search Tree yields sorted list of nodes.*
- *Warshall's algorithm is used to find minimum spanning tree of a graph.*
- *The number of leaf nodes in a complete binary tree of depth d is 2^d.*
- *A tree with n nodes has n/2 edges.*
- *Post order traversal techniques lists the nodes of binary search tree in ascending order.*

Solution:

- True
- False (Inorder traversal of Binary Search Tree yields sorted list of nodes)
- False (Prim's or Kruskal's algorithm is used to find minimum spanning tree of a graph)
- True
- False (A tree with n nodes has n-1 edges)
- False (Inorder traversal techniques lists the nodes of binary search tree in ascending order.)

Example 12: *Find minimum spanning tree for the graph shown below using Kauskal's algorithm.* ***(Oct. 2018)***

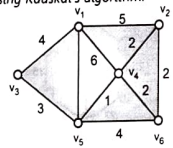


Fig. 7.110

Solution: The minimum spanning tree using Krushal's algorithm is as follows:

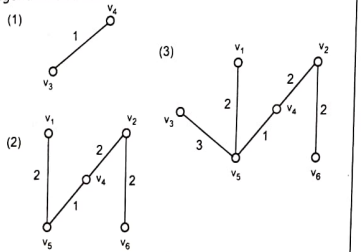


Fig. 7.111

The graph shown above is a minimum spanning tree. Its weight is 10.

Example 13: *Give the stepwise construction of minimum spanning tree using Prim's algorithm for the following graph. Obtain the total cost of minimum spanning tree.* ***(April 2019)***

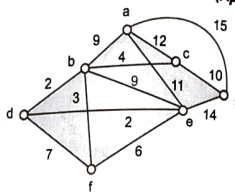


Fig. 7.112

Solution: With the help of Prim's algorithm, the minimum spanning tree of the given graph can be obtained as follows:

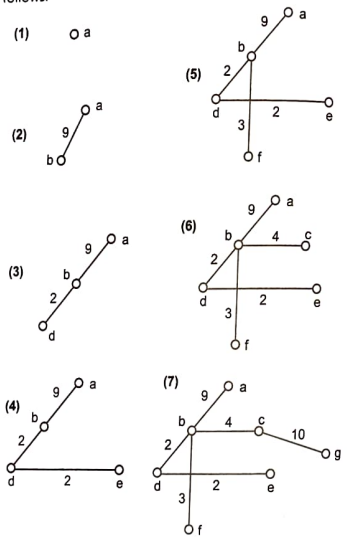


Fig. 7.113

The graph shown above is a minimum spanning tree.

The total cost of minimum spanning tree is 30.

7.7 FUNDAMENTAL CIRCUITS AND FUNDAMENTAL CUT SETS

In this section, we describe the inter relationship between spanning trees, cycles and cut sets.

Let G be a connected graph and let T be a spanning tree of G. Consider a **chord** of T i.e. an edge e of G which is not in T. Then T + e contains a unique circuit called **fundamental circuit** of G with respect to T. A fundamental circuit in a graph G is always with respect to a spanning tree of G. For different spanning trees of the graph G, the fundamental circuits will be different. For example, in Fig. 7.114, T_1 and T_2 are two spanning trees of a connected graph G.

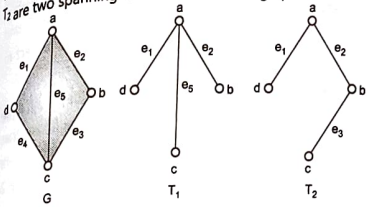


Fig. 7.114

For T_1, the edge e_3 and e_4 of G are chords. If the chord e_3 is added to T_1, then we obtain a circuit {e_2, e_5, e_3}. This is a fundamental circuit corresponding to e_3. Similarly corresponding to e_4, the fundamental circuit of G in T_1 is {e_1, e_5, e_4}.

Now consider the spanning tree T_2 of G. The chords of T_2 are e_4 and e_5. For e_4, the fundamental circuit is {e_1, e_2, e_3, e_4} and for e_5, {e_2, e_3, e_5} is a fundamental circuit. From above, it is clear that different spanning trees have different fundamental circuits. The number of fundamental circuit in any spanning tree is equal to the number of chords of that spanning tree. Hence, if the connected graph G has v number of vertices and e number of edges then the spanning tree T has (v – 1) branches and (e – v + 1) number of chords. Thus, there will be **(e – v + 1)** number of fundamental circuits in G with respect to the spanning tree T.

Fundamental Cut Set

The cut sets related to the spanning tree of a connected graph are defined below.

Let T be the spanning tree of the connected graph G. Since the removal of any branch from a spanning tree breaks the spanning tree into two trees, we say that corresponding to a branch in a spanning tree, there is a division of the vertices in the graph into two subsets corresponding to the vertices in the two trees. It follows that for every branch in a spanning tree there is a corresponding cut set called fundamental cut set. The set of all fundamental cut sets of the graph G with respect to the spanning tree T is called a fundamental system of cut set. For example, for the following graph G shown in Fig. 7.115. The removal of the branch e_1 from the spanning tree T divides the vertices of T into two subsets {a, f, d} and {b, c}. Hence corresponding to the branch e_1, the fundamental cut set of G with respect to T is {e_1, e_3, e_6}. That is, by removing e_1, e_3, e_6 edges from the graph G, the graph will become disconnected and will have two components with vertex set {a, f, d} and {b, c} respectively.

Similarly, for the branches e_2, e_4 and e_5, the fundamental cut sets are {e_2, e_3, e_6}, {e_4, e_3} and {e_5, e_6, e_3}.

From the above discussion, it is clear that each fundamental cut set contains **exactly one branch** of the spanning tree.

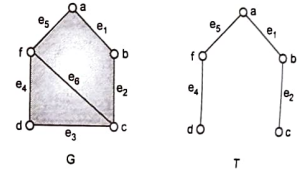


Fig. 7.115

The number of fundamental cut sets is equal to the number of branches in the spanning tree. If the connected graph G has v number of vertices and e number of edges, then the spanning tree has (v – 1) branches which is the same as the number of fundamental cut sets.

Now we present a close relationship between the fundamental circuits and the fundamental cut sets relative to a spanning tree.

Theorem :
A cut-set and any spanning tree must have at least one edge in common.
Proof:
Suppose there is a cut set that has no common edge with a spanning tree.
Then, the removal of the cut set will leave the spanning tree intact.

This means that the removal of the cut set will not separate the graph into two components, which is in contradiction to the definition of a cut set.

Hence, there must be at least one edge common between the cut set and the spanning tree of the graph.

Theorem :

A circuit and the complement of any spanning tree must have at least one edge in common.

Proof:

We know that the complement of a graph G is a graph whose vertex set is same as the vertex set of G and which contains the edges which are **not** in G.

Thus, the complement of a spanning tree will contain the edges which are not in the spanning tree.

If there is a circuit that has no common edge with the complement of a spanning tree, the circuit should be contained in the spanning tree.

However, this is impossible as a tree cannot contain a circuit.

Hence, a circuit and the complement of the spanning tree must have at least one edge in common.

Theorem :

For a given spanning tree, let D = {e_1, e_2, e_3 ... e_k} be a fundamental cut-set in which e_1 is a branch and e_2, e_3 ... e_k are chords of the spanning tree. Then, e_1 is contained in the fundamental circuits corresponding to e_i for i = 2, 3 ... k. Moreover, e_1 is not contained in any other fundamental circuits.

Proof:

Let C be the fundamental circuit corresponding to the chord e_2.

Since D also contains e_2, hence, e_2 is common in both C and D.

Now, C contains branches of the spanning tree and the chord e_2 and D contains the chords e_2, e_3 ... e_k and the branch e_1.

Also, every circuit has an even number of edges in common with every cut-set.

Therefore C and D will have an even number of edges

SOLVED EXAMPLES

Example 1: *How many fundamental cut sets and fundamental circuits are there in a graph G (with respect to any spanning tree) with 8 vertices and 11 edges ?*

Solution: Since the graph G contains 8 vertices, hence, its spanning tree T will contain 7 branches. We know that corresponding to each branch of the spanning tree T, there exists a unique fundamental cut set. Hence, there are 7 fundamental cut-sets of G with respect to T.

Also, the given graph has 11 edges. Thus, the total number of chords is 11 – 7 = 4. Since corresponding to each chord, there is a unique fundamental circuit, hence, there are 8 fundamental circuits in G with respect to T.

Example 2: *Find the fundamental circuits of the following graph G with respect to the given spanning tree T shown in following Fig. 7.116 below.*

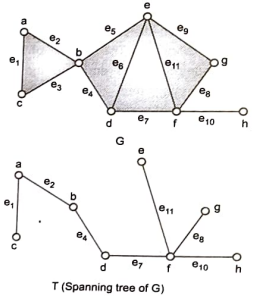


Fig. 7.116

Solution: The branches of T are e_1, e_2, e_4, e_7, e_8, e_{10} and e_{11}. The chords of G are e_3, e_5, e_6 and e_9. Hence, there will be four fundamental circuits. Corresponding to e_3, the fundamental circuit is {e_1, e_2, e_3}.

Corresponding to the chords e_5, e_6 and e_9 the fundamental circuits are

{e_4, e_7, e_{11}, e_5}, {e_7, e_{11}, e_6} and {e_{11}, e_8, e_9} respectively.

Example 3: *Find the fundamental cut-sets of the following graph G with respect to the given spanning tree T.*

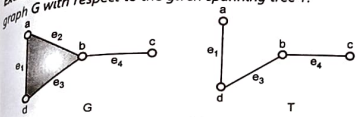


Fig. 7.117

Solution: Corresponding to each branch of the spanning tree, there exists a unique fundamental cut set. Here the spanning tree T has 3 branches e_1, e_3, e_4. Hence the graph G has 3 fundamental cut-sets with respect to T.

For e_1, {e_1, e_2} is a fundamental cut-set.

For e_3 and e_4, {e_2, e_3} and {e_4} are fundamental cut sets respectively.

Example 4: *Find the fundamental system of cut set for the graph G shown in Fig. 7.118 with respect to the spanning tree T.*

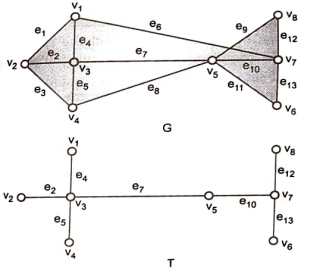


Fig. 7.118

Solution: Here the spanning tree has 7 branches and corresponding to each branch there is a fundamental cut set. Hence there are 7 fundamental cut-set of G with respect to T. Different fundamental cut sets are given below:

Branch Name	Fundamental Cut Set
e_2	{e_2, e_1, e_3}
e_4	{e_4, e_1, e_6}
e_5	{e_5, e_3, e_9}
e_7	{e_7, e_6, e_8}
e_{10}	{e_{10}, e_6, e_9, e_{11}}
e_{12}	{e_{12}, e_9}
e_{13}	{e_{13}, e_{11}}

Example 5: *Determine all possible spanning tress of the given graph shown in Fig. 7.119. Consider any spanning tree, find its fundamental system of cut-sets.*

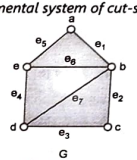
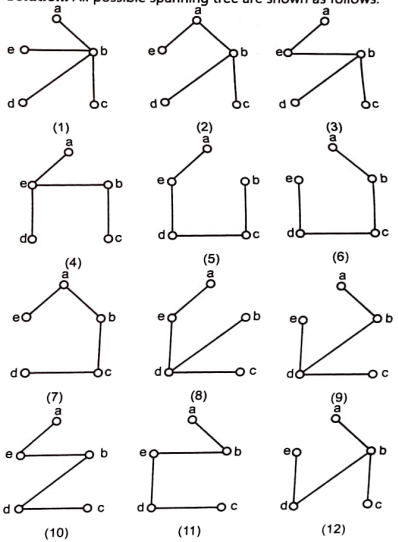


Fig. 7.119 (a)

Solution: All possible spanning tree are shown as follows:



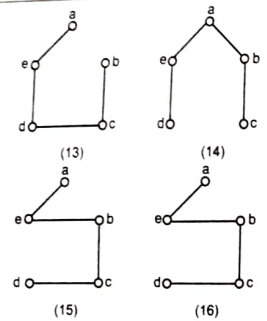


Fig. 7.119 (b)

Now, consider the following spanning tree T of the graph G in Fig. 7.120.

The fundamental cut-sets with respect to T are as follows:

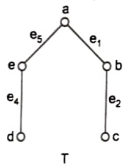


Fig. 7.120

Branch	Fundamental cut-set
e_1	{e1, e3, e6, e7}
e_2	{e2, e3}
e_4	{e4, e3, e7}
e_5	{e5, e6, e7, e3}

Example 6: *Determine all possible cut-sets of the graph G in a Fig. 7.121. Construct a spanning tree of the graph G and find its fundamental system of cut-sets.*

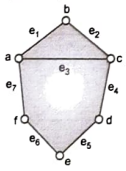


Fig. 7.121

Solution: A cut-set is a set of minimum number of edges where removal disconnects the graph. Following are the possible cut-sets of G –

(1) {e1, e2}, (2) {e2, e3, e4}, (3) {e4, e5} (4) {e5, e6}, (5) {e6, e7}, (6) {e7, e3, e1}, (7) {e2, e3, e5}, (8) {e1, e3, e5}, (9) {e2, e3, e6}, (10) {e1, e3, e6}, (11) {e4, e6}, (12) {e7, e4}, (13){e7, e5}, (14) {e2, e3, e7}, (15) {e1, e3, e4}.

Now consider the following spanning tree in Fig. 7.122.

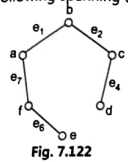


Fig. 7.122

The fundamental cut-sets are

Branch	Fundamental cut-set
e_1	{e1, e3, e5}
e_2	{e2, e3, e5}
e_4	{e4, e5}
e_6	{e6, e5}
e_7	{e7, e5}

Example 7: *Prove that the complement of a spanning tree does not contain a cut-set and that the complement of a cut-set does not contain a spanning tree.*

Solution: Suppose the complement of a spanning tree contains a cut-set. This means, the spanning tree does not contain the cut-set. Hence there is no edge common between the cut-set and the spanning tree. But this is the contradiction to the theorem that there is at least one edge common between a cut-set and the spanning tree. Therefore the complement of a spanning tree does not contain a cut-set.

For the second part, again assume that the complement of a cut-set contains the edges of spanning tree. This means there is no edge common to the cut-set and the spanning tree, which is the contradiction to the theorem 8.6.1. Hence the complement of a cut-set does not contain a spanning tree.

Example 8: *Draw the fundamental cut-sets and union of edge disjoint fundamental cut-sets of the graph G with respect to spinning trees T_1 and T_2 given below.*

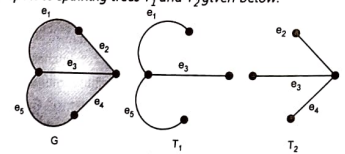
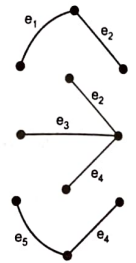


Fig. 7.123

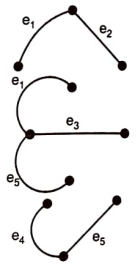
Solution: The fundamental cut-sets with respect to the spanning tree T_1 are as follows:

Branch	Fundamental cut-set
e_1	$\{e_1, e_2\}$
e_3	$\{e_2, e_3, e_4\}$
e_5	$\{e_4, e_5\}$



For the spanning tree T_2,

Branch	Fundamental cut-set
e_1	$\{e_1, e_2\}$
e_3	$\{e_1, e_3, e_5\}$
e_4	$\{e_4, e_5\}$



The union of edges disjoint fundamental cut-sets is Fig. 7.124 {e1, e2, e4, e5}

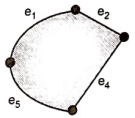


Fig. 7.124

Example 9: *For the Fig. 7.125 shown, give fundamental circuit.*

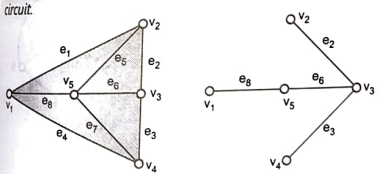


Fig. 7.125

Solution: Since there are four chords e1, e4, e5, e7, there will be four fundamental circuits corresponding to each chord.

Chord	Fundamental Circuit
e_1	{e1, e2, e6, e8}
e_4	{e4, e8, e6, e3}
e_5	{e5, e2, e6}
e_7	{e7, e3, e6}

Example 10: *For the graph drawn in Fig. 7.126 (a) and spanning tree in Fig. 7.126 (b), determine fundamental system of cut sets and fundamental circuits.*

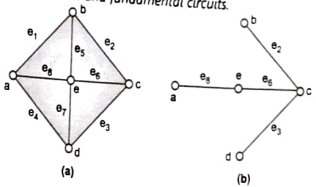


Fig. 7.126

Solution: Since there are 4 branches in the spanning tree (b), therefore there will be four fundamental cut sets corresponding to each branch.

Branch	Fundamental Cutset
e_2	{e1, e5, e2}
e_3	{e4, e7, e3}
e_6	{e1, e5, e7, e4, e6}
e_8	{e1, e4, e8}

Since the spanning tree contains 4 branches, the remaining 4 edges in the graph are chords, namely e1, e4, e5, e7. Corresponding to each of the chord, there is a circuit called fundamental circuit. They are given as follows:

Chord	Fundamental Circuit z
e_1	$\{e_1, e_2, e_6, e_8\}$
e_4	$\{e_4, e_8, e_6, e_3\}$
e_5	$\{e_5, e_2, e_6\}$
e_7	$\{e_7, e_6, e_3\}$

EXERCISE 7.1

1. Draw all non isomorphic trees with 4 and 6 vertices.
2. Draw a tree with 16 vertices which has either degree 1 or degree 3. Can you draw such a tree with 15 vertices.
3. Is it possible to draw a tree with six vertices having degrees 1, 1, 1, 1, 3, 3. If yes, draw the tree.

4. (a) Define with examples –
   (i) Rooted tree
   (ii) m-ary tree **(April 2018)**
   (iii) full binary tree
   (iv) level of vertex the rooted tree
   (v) Subtree and regular tree
   (vi) Regular m-ary tree
   (vii) Binary Search Tree **(April 2019)**
   (viii) Spanning Tree **(April 2019), (April 2018)**
   (ix) Level and Height of a Tree **(April 2019),(Oct.2018)**
   (x) Eccentricity of a vertex **(Oct.2018)**
   (xi) Properties of Trees **(April 2018)**
   (xii) Decision Tree **(Oct.2017)**

   (b) Find the cut points of the following tree. Also find the level of each vertex.

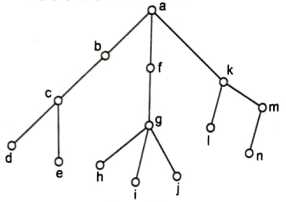


Fig. 7.127

5. Define the height of the tree. Find the height of the tree shown in Fig. 7.127 in the above problem 4.
6. Draw a full binary tree with 5 terminal vertices and 4 internal vertices.
7. (a) Explain the following terms in a rooted tree –
   (i) son (ii) parent (iii) subtree
   (iv) ancestor (v) descendant.

   (b) Answer the following questions for the tree in Fig. 7.128.
   (i) Find the parents of f and of i.
   (ii) Find the ancestors of c and d.
   (iii) Find the descendant of e and of d.
   (iv) Draw the subtree rooted at c.

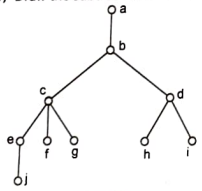


Fig. 7.128

8. Find whether the following tree is a full m-ary tree for some m or not. Redraw the tree with the root 'v' on the top.

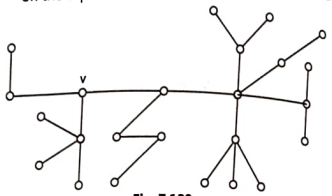


Fig. 7.129

9. Determine the total number of nodes on a full binary tree with 5 interior nodes.
10. Determine the number of leaves on a full 4-ary tree with 20 interior nodes.
11. Is there a full 4-ary tree with 100 nodes ?
12. Define an optimal binary prefix code.
13. For the following set of weights, construct an optimal binary prefix code. For each weight in the set, give the corresponding code word.
    (i) 2, 3, 5, 7, 9, 13. (ii) 8, 9, 10, 11, 13, 15, 22.
14. Define a spanning tree in a connected graph. When is it called minimum spanning tree.
15. Explain Kruskal algorithm to find a minimum spanning tree in a weighted connected graph. **(April 2018), (Oct. 2017)**
16. Obtain the minimum spanning tree for the following graphs. Find the total cost of the minimum spanning tree.

    (i)

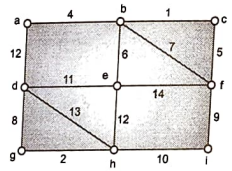


Fig. 7.130

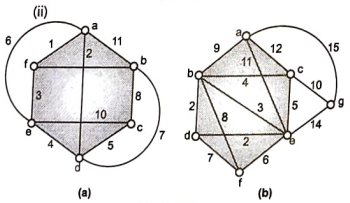


Fig. 7.131

17. Define the terms cut-set, fundamental system of cut-sets and fundamental circuits.
18. Determine all possible cut-sets of the following graph shown in Fig. 7.132. Construct a spanning tree of the graph G and find its fundamental system of cut-sets.

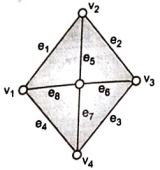


Fig. 7.132

19. Find the fundamental system of cut-set for the following graph G with respect to the spanning tree T given in Fig. 7.133.

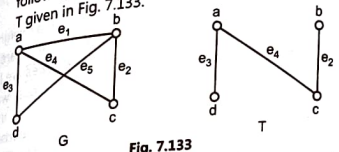


Fig. 7.133

20. For a given spanning tree, let C = {e1, e2, e3, ... ek} be a fundamental circuit in which e1 is a chord and e2, e3 ... ek are branches of the spanning tree. Prove that, e1 is contained in the fundamental cut-sets corresponding to ei for i = 2, 3, ... k. Moreover e1 is not contained in any other fundamental cut-set.
21. Prove that a regular m-ary tree with i-interior nodes has $(m_i + 1)$ nodes at all.
22. Write a short note on "Binary Search tree and its applications".
23. Define prefix code. Justify whether given set of codes are prefix codes or not.
    (i) {000, 001, 01, 10, 11}, (ii) {1, 00, 01, 000, 0001}.
24. Define: (i) Forest, (ii) Height of tree, (iii) Ordered tree, (iv) Properties of tree.

ANSWER - 7.1

1. Non-isomorphic trees on 6 vertices are:

Fig. 7.134

Non-isomorphic trees on 4 vertices are:

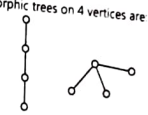


Fig. 7.135

2. Tree with 16 vertices of degree 1 or 3.

Fig. 7.136

It is not possible to draw a tree with 15 nodes which has degree 1 or 3.

3. Yes.

Fig. 7.137

4. The vertices of degree other than one are cut vertices. Levels of vertices of T are given below:
   $l(b) = l(f) = l(k) = 1$
   $l(c) = l(g) = l(l) = l(m) = 2$
   $l(d) = l(e) = l(h) = l(i) = l(j) = l(n) = 3.$
5. height = 3
6.

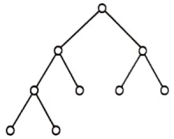


Fig. 7.138

7. (i) c and d
   (ii) a and b
   (iii) f and (h, i)
   (iv) The subtree rooted at c is

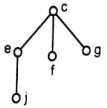


Fig. 7.139

8. No, it is not a full m-ary tree.
9. 11 10. 61
11. No, the integer solution to the equation 100 = 4i + 1 does not exist.
12. (i) Code word for 2 → 0000
Code word for 3 → 0001
Code word for 5 → 001
Code word for 7 → 10
Code word for 9 → 11
Code word for 13 → 01

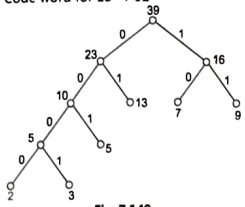


Fig. 7.140

(ii)

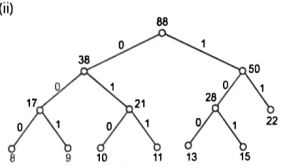


Fig. 7.141

The prefix code for the weights 8, 9, 10, 11, 13, 15 and 22 is given by {000, 001, 010, 011, 100, 101, 11}

16. (i)

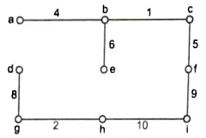


Fig. 7.142

Weight = 45

(ii)

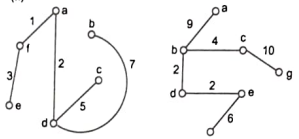


Fig. 7.143

Weight = 18 Weight = 33

18. All possible cut-sets are {e1, e4, e8}, {e1, e2, e5}, {e2, e3, e6}, {e3, e4, e7}, {e5, e6, e7, e8}, {e4, e8, e5, e2}, {e1, e5, e6, e3}, {e4, e6, e7, e2}, {e1, e8, e7, e3} {e1, e8, e7, e6, e2}, {e2, e5, e8, e7, e3}, {e4, e8, e5, e6, e3}, {e1, e5, e6, e7, e4}
19. {e5, e3}, {e4, e1, e5}, {e1, e5, e2}.
23. (i) Yes (ii) No, the sequence 00 is a prefix of the sequence 0001.

7.8 TRANSPORT NETWORK

In this section, we will see how a weighted directed graph can be used as a model for a network of pipelines through which some commodity is transported from one place to another.

- A network represents a model for transportation of a commodity from its production centre to its market through communication routes such as roads, pipelines, cables, etc. It is important to know the maximum rate of flow that is possible from one place to another place in the network.
- This type of network is represented by a weighted connected directed graph in which the vertices are the places (stations) in the network and the edges are lines through which the given commodity (oil, gas, water, cars, etc.) flow.
- The weight, a real positive number, associated with each edge represents the capacity of the line, that is, the maximum amount of flow possible per unit time. The graph shown in Fig. 7.144 represents a flow network containing 6 stations and 10 lines. The capacity of each line is also indicated in the Fig. 7.144

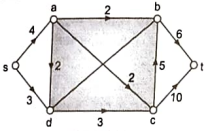


Fig. 7.144

- In the network, the vertex with no incoming edge is called a **source** s and the vertex with no outgoing edge is called a **sink** t. It is assumed that at each intermediate vertex, the total rate of commodity entering is equal to the rate of commodity leaving the vertex i.e. there is no accumulation or generation of the commodity at any vertex along the way. Further, the flow along any edge cannot exceed the capacity of that edge.

Now we define the transport network and maximal flow.

- **Transport Network:** A simple, connected, weighted, digraph G is called a transport (or flow) network if the weight associated with every directed edge in G is a non-negative number. In a transport network this number represents the capacity of the edge and is designated as C (i, j) for the edge directed from the vertex i to the vertex j.
- **Maximal Flow:** In a given transport network G, a flow is an assignment of non-negative number f (i, j) to every directed edge (i, j) (edge from the vertex i to the vertex j) such that the following conditions are satisfied:

(i) For every directed edge (i, j) in G,
$f(i, j) \leq c(i, j)$

(ii) There is a specified vertex s in G, called the **source**, for which $\sum_i f(s, i) - \sum_i f(i, s) = w$

where the summations are taken over all the vertices in G. Quantity w is called the **value of the flow.**

(iii) There is another specified vertex t in G, called the **sink** for which, $\sum_i f(t, i) - \sum_i (i, t) = -w$

(iv) All other vertices are called intermediate vertices and for each intermediate vertex j,
$\sum_i f(i, j) = \sum_k f(j, k)$, $j \neq s, t$

- It is clear from condition (i) that the flow through any edge does not exceed its capacity. Conditions (ii) and (iii) state that the net flow out of source and the net flow into the sink is w. According to the condition (iv), the amount of material flowing into a vertex must be equal to the amount of material flowing out of the vertex except at the source and the sink. For a given flow, an edge (i, j) is said to be **saturated** if f (i, j) = c (i, j) and is said to be **unsaturated** if f (i, j) < c (i, j).
- A set of flows f (i, j) for all (i, j) in G is called a flow pattern. A flow pattern that maximizes the quantity w is called a maximal flow pattern. There may be more than one maximum flow in a transport network.
- **Cut and the capacity of a cut:** Ignoring the directions of edges in a transport network, consider a cut-set with respect to the vertices s and t, that is, a cut set which separates the source s from the sink t. Such a set of edges in a transport network is called a **cut**. The notation (P, $\bar{P}$) is used to denote a cut that partitions the vertices into two subsets P and , where the subset P contains the source s and the subset $\bar{P}$ contains the sink t. The **capacity of a cut** denoted by C (P, $\bar{P}$) is defined to be the sum of the capacities of those edges directed from the vertices in the set P to the vertices in $\bar{P}$, that is, $C(P, \bar{P}) = \sum_{i \in P} \sum_{j \in \bar{P}} c(i, j)$

- For example, in Fig. 7.145, the dashed line identifies a cut that separates the vertices P = {s, b} from the subset of vertices $\bar{P}$ = {c, d, t}. The capacity of this cut is 5 + 1 + 5 + 3 = 14.
- Now, we give the statements of two theorems which give an upper bound on the values of flows in a transport network.

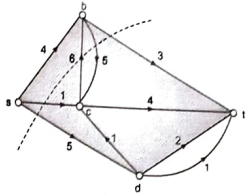


Fig. 7.145

Theorem: The value of the flow w in a given transport network is less than or equal to the capacity of any cut in the network.

(Max - Flow Min - Cut Theorem):

In a given transport network G, the maximum value of a flow from the source s to the sink t is equal to the minimum value of the capacities of all the cuts in G that separate the source s from the sink t.

For example, consider the transport network of Fig. 7.144. It has eight cuts that separate s from it. These cuts and their capacities are given in the following table.

Table.7.1

Vertex set P	Capacity C (P,)
{s}	10
{s,b}	14
{s, c}	19
{s, d}	9
{s, b, c}	12
{s, b, d}	13
{s, c, d}	17
{s, b, c, d}	10

The cut with minimum capacity among these is P = {s, d} and $\bar{P}$ = {b, c, t}. The maximum flow possible in s to t in the network is therefore w = 9 units.

We will now present an algorithm for constructing a maximum flow in a transport network. According to the theorem, the value of the flow in a transport network is less than or equal to the capacity of any cut. Therefore, we can always construct a flow, the value of which is equal to the capacity of a cut. This flow is a maximal flow in the given network. To find the maximal flow, we use the labeling procedure in the given network.