

**Unit V: Trees**

**07 Hours**

Introduction, properties of trees, Binary Search Tree, Tree Traversal, Decision Tree, Prefix codes and Huffman coding, cut sets, Spanning Trees and Minimum Spanning Tree, Kruskal's and Prim's algorithms, The Max flow- Min Cut Theorem (Transport network).

**Exemplar/ Case Studies:** Algebraic Expression Tree, Tic-Tac-Toe Game Tree.

❖ **INTRODUCTION**

- A tree is a connected undirected graph with no simple circuits.
- Because a tree cannot have a simple circuit, a tree cannot contain multiple edges or loops. Therefore any tree must be a simple graph.

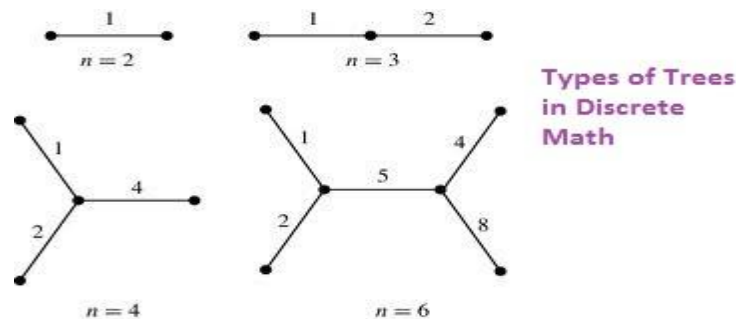


Figure 1: Type of Trees

- **Example 1:** Which of the graphs shown in Figure 2 are trees?

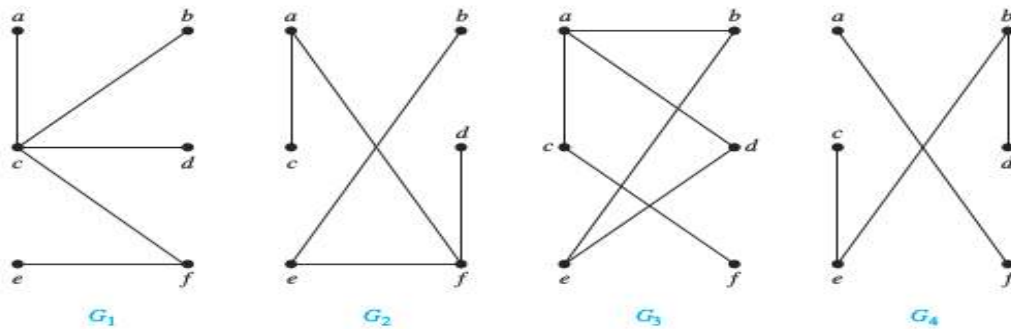


Figure 2: Examples of Trees and Graphs That Are Not Trees

Solution:  $G_1$  and  $G_2$  are trees, because both are connected graphs with no simple circuits.  $G_3$  is not a tree because  $e, b, a, d, e$  is a simple circuit in this graph. Finally,  $G_4$  is not a tree because it is not connected.

- Any connected graph that contains no simple circuits is a tree.
- What about graphs containing no simple circuits that are not necessarily connected?.
- These graphs are called **forests** and have the property that each of their connected components is a tree. Figure 3 displays a forest.
- A **forest** is composed of one tree or some disconnected trees. A forest is a disjoint union of trees.

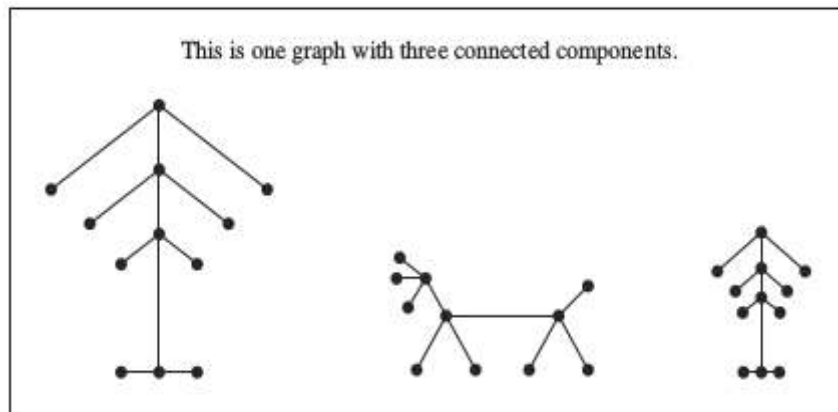


Figure 3: Example of Forest

➤ **Example 2:** Are the following graphs trees?

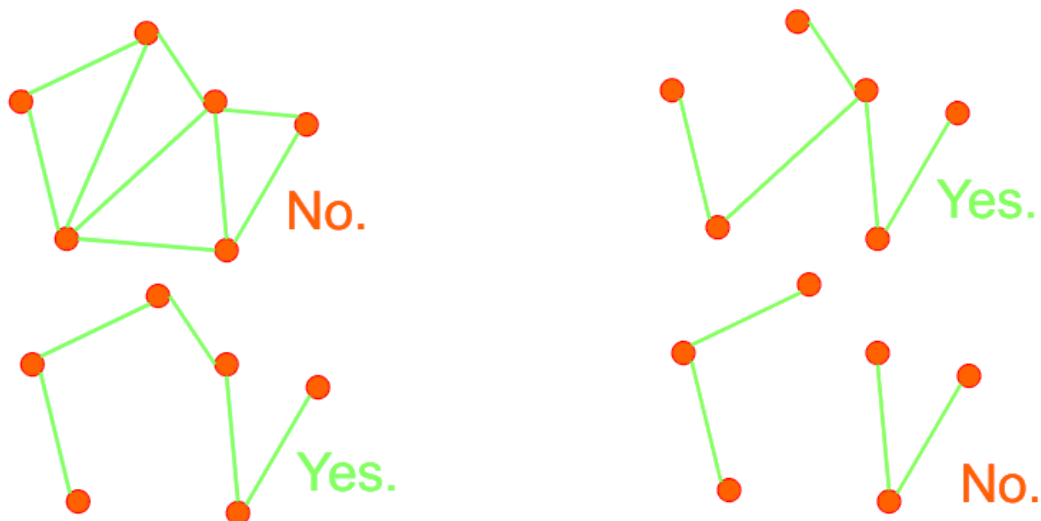


Figure 4: Example of Tree

➤ **Theorem 1 :** An undirected graph is a tree if and only if there is a unique simple path between any two of its vertices.

➤ **Theorem 2:** Every tree with  $n$  vertices has exactly  $n - 1$  edges.

❖ **Rooted Trees**

➤ A rooted tree is a tree in which one vertex has been designated as the root and every edge is directed away from the root.

❖ **Terminology for rooted trees**

➤ If  $v$  is a vertex in  $T$  other than the root, the parent of  $v$  is the unique vertex  $u$  such that there is a directed edge from  $u$  to  $v$ .

➤ If  $u$  is the **parent** of  $v$ ,  $v$  is called a **child** of  $u$ .

➤ Two Vertices with the same parent are called **siblings**.

➤ A vertex of a tree is called a **leaf** if it has no children. A **terminating vertex** (or a leaf) in a tree is a vertex of degree 1.

- Vertices that have children are called **internal vertices**. An **internal vertex** (or a branch vertex) in a tree is a vertex of degree greater than 1.
- **Vertices** are sometimes referred to as **nodes**, particularly when dealing with graph trees.
- The **ancestors** of a vertex other than the root are the vertices in the path from the root to this vertex, excluding the vertex itself and including the root (i.e., its parent, its parent's parent, and so on, until the root is reached).
- The **descendants** of a vertex  $v$  are those vertices that have  $v$  as an ancestor.
- If  $a$  is a vertex in a tree, the **subtree** with  $a$  as its root is the subgraph of the tree consisting of  $a$  and its descendants and all edges incident to these descendants.

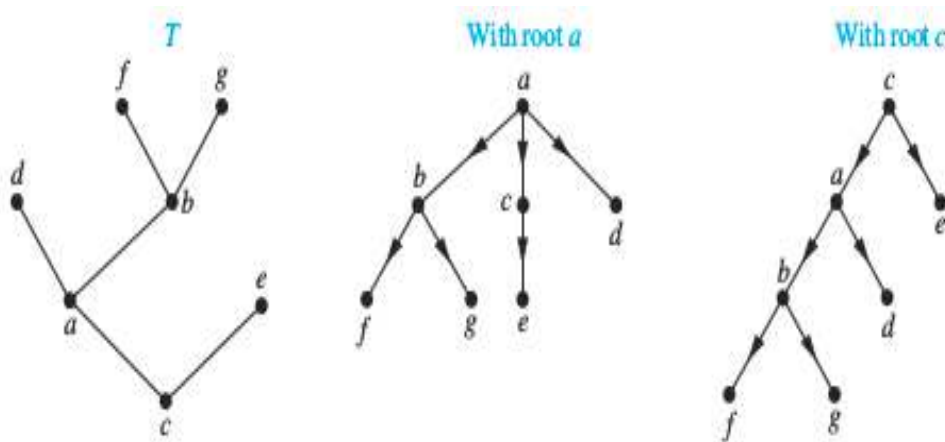


Figure 5: Tree and Rooted Trees Formed by Designating Two Different Roots.

- **Example 3:** In the rooted tree  $T$  (with root  $a$ ) shown in Figure 6, find the parent of  $c$ , the children of  $g$ , the siblings of  $h$ , all ancestors of  $e$ , all descendants of  $b$ , all internal vertices, and all leaves. What is the subtree rooted at  $g$ ?

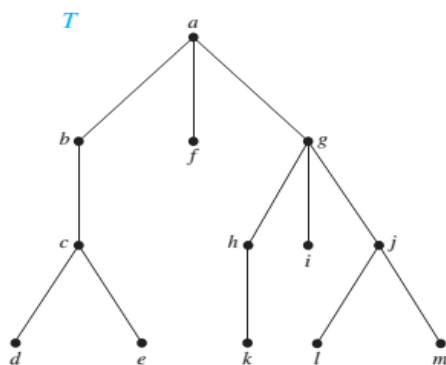


Figure 6: A Rooted Tree  $T$

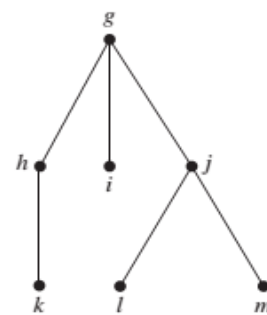


Figure 7: The Subtree Rooted at  $g$ .

Solution: The parent of  $c$  is  $b$ . The children of  $g$  are  $h$ ,  $i$ , and  $j$ . The siblings of  $h$  are  $i$  and  $j$ . The ancestors of  $e$  are  $c$ ,  $b$ , and  $a$ . The descendants of  $b$  are  $c$ ,  $d$ , and  $e$ . The internal vertices are  $a$ ,  $b$ ,  $c$ ,  $g$ ,  $h$ , and  $j$ . The leaves are  $d$ ,  $e$ ,  $f$ ,  $i$ ,  $k$ ,  $l$ , and  $m$ . The subtree rooted at  $g$  is shown in Figure 7.

- **Example 4:** Answer these questions about the rooted tree illustrated in Figure 8.
- a) Which vertex is the root?
  - b) Which vertices are internal?
  - c) Which vertices are leaves?
  - d) Which vertices are children of j?
  - e) Which vertex is the parent of h?
  - f) Which vertices are siblings of o?
  - g) Which vertices are ancestors of m?
  - h) Which vertices are descendants of b?

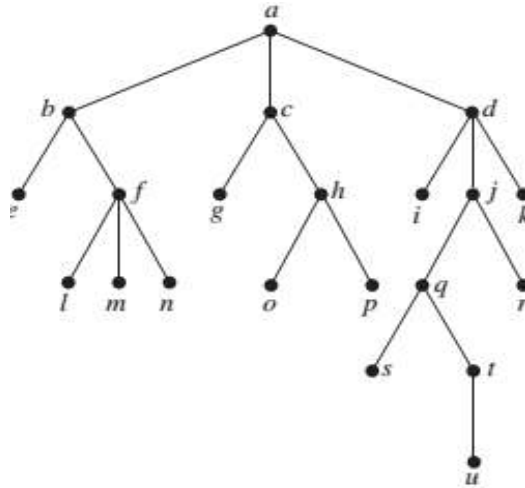


Figure 8: Rooted Tree

**Solution:** a) Vertex a is the root, since it is drawn at the top.

b) The internal vertices are the vertices with children, namely a, b, c, d, f, h, j, q, & t.

c) The leaves are the vertices without children, namely e, g, i, k, l, m, n, o, p, r, s, & u.

d) The children of j are the vertices adjacent to j and below j, namely q and r.

e) The parent of h is the vertex adjacent to h and above h, namely c.

f) Vertex o has only one sibling, namely p, which is the other child of o's parent, h.

g) The ancestors of m are all the vertices on the unique simple path from m back to the root, namely f, b, and a.

h) The descendants of b are e, f, l, m, and n.

- **Example 5:** What is the level/length of each vertex of the rooted tree in Figure 8?

Solution: We can easily determine the levels from the drawing. The root a is at level 0. The vertices in the row below a are at level 1, namely b, c, and d. The vertices below that, namely e through k (in alphabetical order), are at level 2. Similarly l through r are at level 3, s and t are at level 4, and u is at level 5.

❖ **M-ary Tree**

- A rooted tree is called an **m-ary tree** if every internal vertex has no more than m children. The tree is called a full m-ary tree if every internal vertex has exactly m children. An m-ary tree with  $m = 2$  is called a binary tree.

- **Example 6:** Are the rooted trees in Figure 9 full m-ary trees for positive integer m?

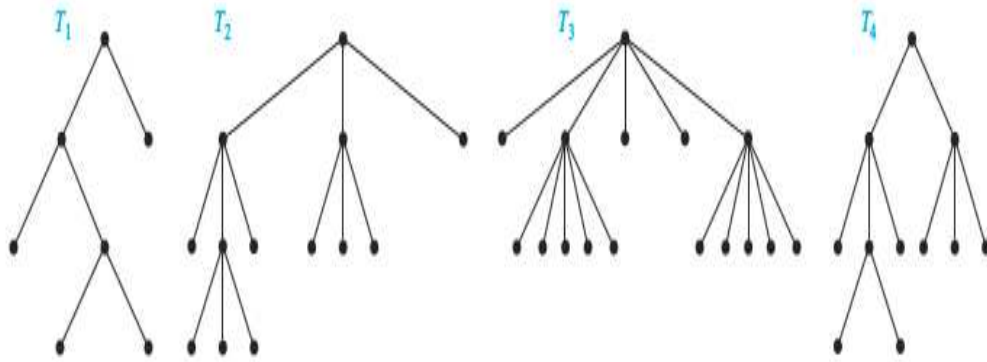


Figure 9: Rooted Trees

Solution:  $T_1$  is a full binary tree because each of its internal vertices has two children.  $T_2$  is a full 3-ary tree because each of its internal vertices has three children. In  $T_3$  each internal vertex has five children, so  $T_3$  is a full 5-ary tree.  $T_4$  is not a full m-ary tree for any m because some of its internal vertices have two children and others have three children.

### ❖ Ordered Rooted Tree

- An ordered rooted tree is a rooted tree where the children of each internal vertex are ordered. Ordered rooted trees are drawn so that the children of each internal vertex are shown in order from left to right.
- In an **ordered binary tree** (usually called just a binary tree), if an internal vertex has two children, the first child is called the **left child** and the second child is called the **right child**.
- The tree rooted at the left child of a vertex is called the left subtree of this vertex, and the tree rooted at the right child of a vertex is called the right subtree of the vertex.
- **Example :** What are the left and right children of d in the binary tree T shown in Figure 10(a) (where the order is that implied by the drawing)? What are the left and right subtrees of c?

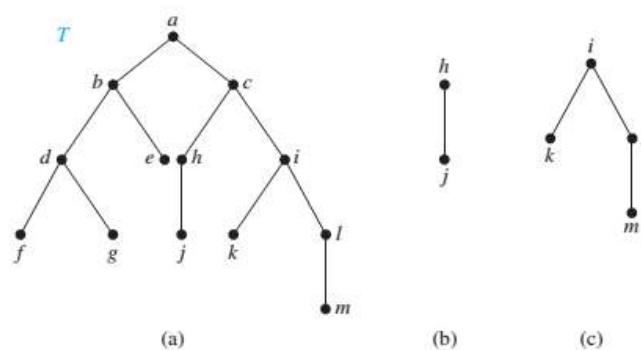


Figure 10: A Binary Tree T and Left and Right Subtrees of the Vertex c.

Solution: The left child of  $d$  is  $f$  and the right child is  $g$ . We show the left and right subtrees of  $c$  in Figures 10(b) and 10(c), respectively.

### ❖ Properties of Trees

- **Every tree with  $n$  vertices has exactly  $n - 1$  edges.**
- **A full  $m$ -ary tree with  $i$  internal vertices contains  $n = mi + 1$  vertices.**  
 Proof: Every vertex, except the root, is the child of an internal vertex. Because each of the  $i$  internal vertices has  $m$  children, there are  $mi$  vertices in the tree other than the root. Therefore, the tree contains  $n = mi + 1$  vertices.
- A full  $m$ -ary tree with
  - (i)  $n$  vertices has  $i = (n - 1)/m$  internal vertices and  $l = [(m - 1)n + 1] / m$  leaves,
  - (ii)  $i$  internal vertices has  $n = mi + 1$  vertices and  $l = (m - 1)i + 1$  leaves,
  - (iii)  $l$  leaves has  $n = (ml - 1)/(m - 1)$  vertices &  $i = (l - 1)/(m - 1)$  internal vertices.

### ❖ Prefix Code

- Consider the problem of using bit strings to encode the letters of the English alphabet (where no distinction is made between lowercase and uppercase letters). We can represent each letter with a bit string of length five, because there are only 26 letters and there are 32 bit strings of length five. The total number of bits used to encode data is five times the number of characters in the text when each character is encoded with five bits.
- Consider using bit strings of different lengths to encode letters. Letters that occur more frequently should be encoded using short bit strings, and longer bit strings should be used to encode rarely occurring letters. When letters are encoded using varying numbers of bits, some method must be used to determine where the bits for each character start and end. For instance, if  $e$  were encoded with 0,  $a$  with 1, and  $t$  with 01, then the bit string 0101 could correspond to  $eat$ ,  $tea$ ,  $eaea$ , or  $tt$ .
- One way to ensure that no bit string corresponds to more than one sequence of letters is to encode letters so that the bit string for a letter never occurs as the first part of the bit string for another letter.
- Codes with this property are called **prefix codes**. For instance, the encoding of  $e$  as 0,  $a$  as 10, and  $t$  as 11 is a prefix code.
- **A set of sequence is said to be prefix code if no sequence in the set is prefix of another. OR**
- **A code is called a prefix (free) code if no codeword is a prefix of another one.**

- A word can be recovered from the unique bit string that encodes its letters. For example, the string 10110 is the encoding of *ate*.
- To see this, note that the initial 1 does not represent a character, but 10 does represent *a* (and could not be the first part of the bit string of another letter). Then, the next 1 does not represent a character, but 11 does represent *t*. The final bit, 0, represents *e*.
- For Example: The set {01,10,11,000} is a prefix code. The set {1,00,01,000,0001} is not prefix code because the sequence 00 is prefix of the sequence 000 and 0001.
- A prefix code can be represented using a binary tree, where the characters are the labels of the leaves in the tree. The edges of the tree are labeled so that an edge leading to a left child is assigned a 0 and an edge leading to a right child is assigned a 1. The bit string used to encode a character is the sequence of labels of the edges in the unique path from the root to the leaf that has this character as its label.

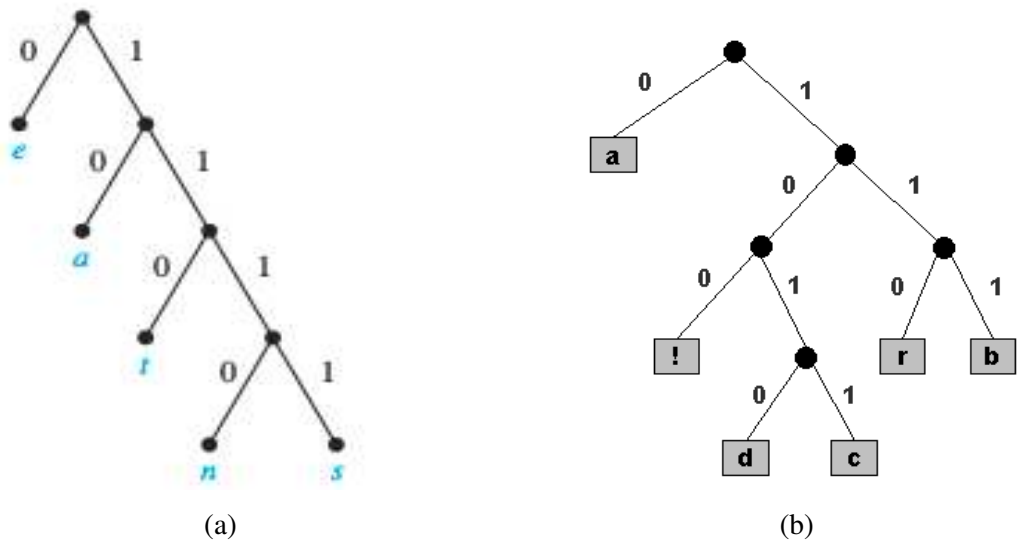


Figure 11: A Binary Tree with a Prefix Code

- For Figure 11(a) Represents the encoding of e by 0, a by 10, t by 110, n by 1110, and s by 1111.

- Bit string 11111011100 using the code in Figure 11(a) encode the message the “sane”

1111 10 1110 0  
s a n e

- For Figure 11(b) Represents the encoding of a by 0, b by 111, r by 110,c by 1011, d by 1010 and ! by 100.

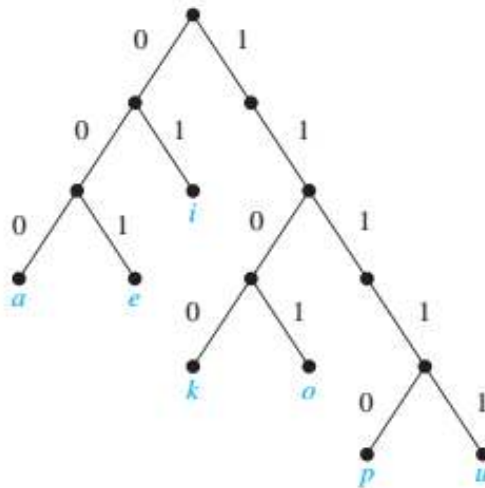
- For example, the bit string 0111110010110101001111100100 encodes the message "abracadabra!". The first 0 must encode 'a', then the next three 1's must encode 'b', then 110 must encode r, and so on as follows:

0 111 110 0 1011 0 1010 0 111 110 0 100  
a b r a c a d a b r a !

- **Example 7:** Which of these codes are prefix codes?
- a) a: 11,e: 00,t: 10,s:01                      b)a:0,e:1,t: 01,s: 001  
 c) a: 101,e: 11,t: 001,s: 011,n: 010      d)a: 010,e: 11,t: 011,s: 1011,n: 1001,i: 10101

Solution:

- a) This is a prefix code, since no code is the first part of another.  
 b) This is not a prefix code, since, for instance, the code for a is the first part of the code for t.  
 c) This is a prefix code, since no code is the first part of another.  
 d) This is a prefix code, since no code is the first part of another.
- **Example 8:** What are the codes for a, e, i, k, o, p, and u if the coding scheme is represented by this tree?



Solution: a :000 ; e:001, i:01; k:1100; o:1101; p:11110 and u:1111.

- **Example 9:** Construct the binary tree with prefix codes representing these coding schemes.
- a) a: 11,e:0,t: 101,s: 100  
 b) a:1,e: 01,t: 001,s: 0001,n: 00001  
 c) a: 1010,e:0,t: 11,s: 1011,n: 1001,i: 10001
- **Example 10:** Given the coding scheme a:001, b:0001, e:1, r:0000, s:0100, t:011, x: 01010, find the word represented by
- a) 01110100011.      b)0001110000.      c) 0100101010.      d)01100101010.

❖ **Optimal Tree**

- Let T be any full binary tree and  $w_1, w_2, w_3, \dots, w_t$  be the weights of the terminal vertices. Then the weight W of a binary tree is given by:

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



Where  $l_i$  is the length of the leaf  $i$  from the root of the tree. The fully binary tree is called optimal if its weight is minimum.

- Optimal tree are used to construct variable length codes, where the letters of alphabets are represented by binary digit.
- **Example 11:** Figure 12 show two fully binary tree  $T_1$  and  $T_2$ , with the weights of the leaves as 3,4 and 5. Show which tree is optimal tree.

Solution:

$$W(T_1) = 3*2 + 4*2 + 5*1 = 6 + 8 + 5 = 19$$

$$W(T_2) = 3*1 + 4*2 + 5*2 = 3 + 8 + 10 = 21$$

Here  $W(T_1) < W(T_2)$ , Thus  $T_1$  is optimal tree for weights 3,4, and 5.

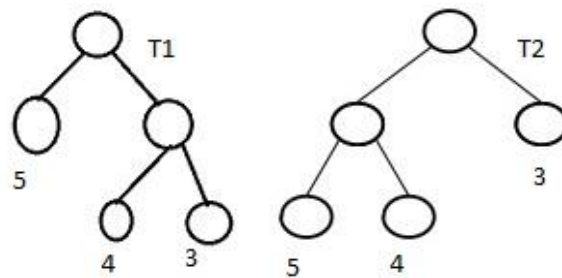


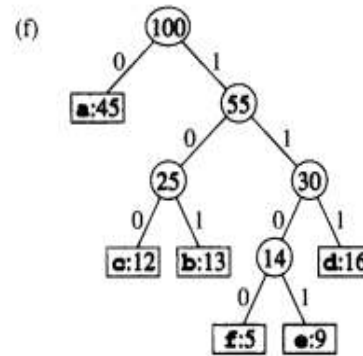
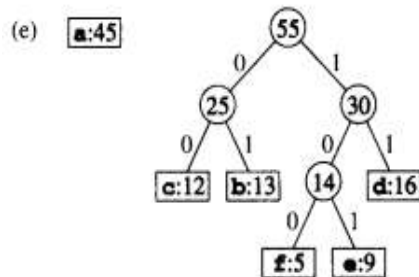
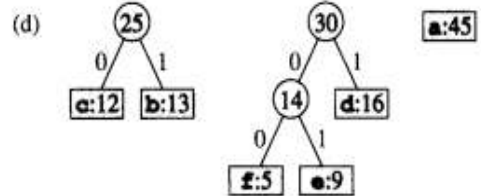
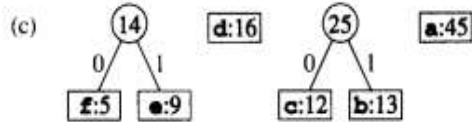
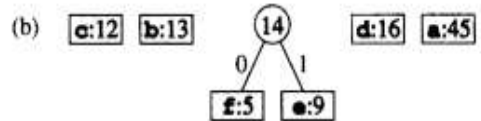
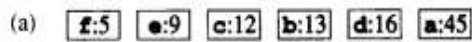
Figure 12: Fully Binary Tree with wights 3,4 and 5

### ❖ Optimal Prefix Code

- A binary prefix code obtained from optimal tree is called **optimal prefix code**.

### ❖ Huffman Algorithm to find optimal tree

- A Huffman code is a particular type of optimal prefix code that is commonly used for lossless data compression.
- **Given:** A set of symbols and their weights (usually proportional to probabilities).
- **Find:** A prefix-free binary code (a set of codewords) with minimum expected codeword length (equivalently, a tree with minimum weighted path length from the root).
- Let  $W_1, W_2, W_3, \dots, W_t$  be the weights of the leaves and it is required to construct an optimal binary tree.
  - ✓ **Step 1:** Arrange the weights of a tree in increasing order.
  - ✓ **Step 2:** Consider two leaves with minimum weights  $W_1$  and  $W_2$ . Replace two leaves and their parent by the leaf. Assign weight  $W_1 + W_2$  to this new leaf.
  - ✓ **Step 3:** Repeat step 2 for the weights  $W_1, W_2, W_3, \dots, W_t$  until no weight remains.
  - ✓ **Step 4:** The tree obtained in this method is an optimal tree for given weights and stop the procedure.
- **Example 12:** Construct Huffman code for the data: 45, 13, 12, 16, 9, 5



- **Example 13:** Construct Huffman code for the data: A: 0.08, B:0.10, C: 0.12, D: 0.15, E: 0.20, F: 0.35. What is the average number of bits used to encode a character?

Solution: (Draw tree)

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

$$3*0.08 + 3*0.10 + 3*0.12 + 3*0.15 + 2*0.20 + 2*0.35 = 2.45$$

- **Example 14:** Use Huffman coding to encode following symbols with given frequencies and find the average number of bits required to encode a character.

i. a: 0.20, b: 0.10, c: 0.15, d: 0.25, e: 0.30.

ii. A: 0.10, B: 0.25, C: 0.05, D: 0.15, E: 0.30, F: 0.07, G: 0.08.

- **Example 15:** For the following set of weights, construct optimal binary prefix code. For each weight in the set, give the corresponding code words.

i) 10,30,05,15, 20, 15, 05.

ii) 8, 9, 10, 11, 13, 15, 22.

- **Example 16:** A secondary storage media contains information in 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), 1st(35), zip(37).Construct the Huffman code of this.



- **Example 18:** How many edges must be removed from a connected graph with  $v$  vertices and  $e$  edges to produce a spanning tree?

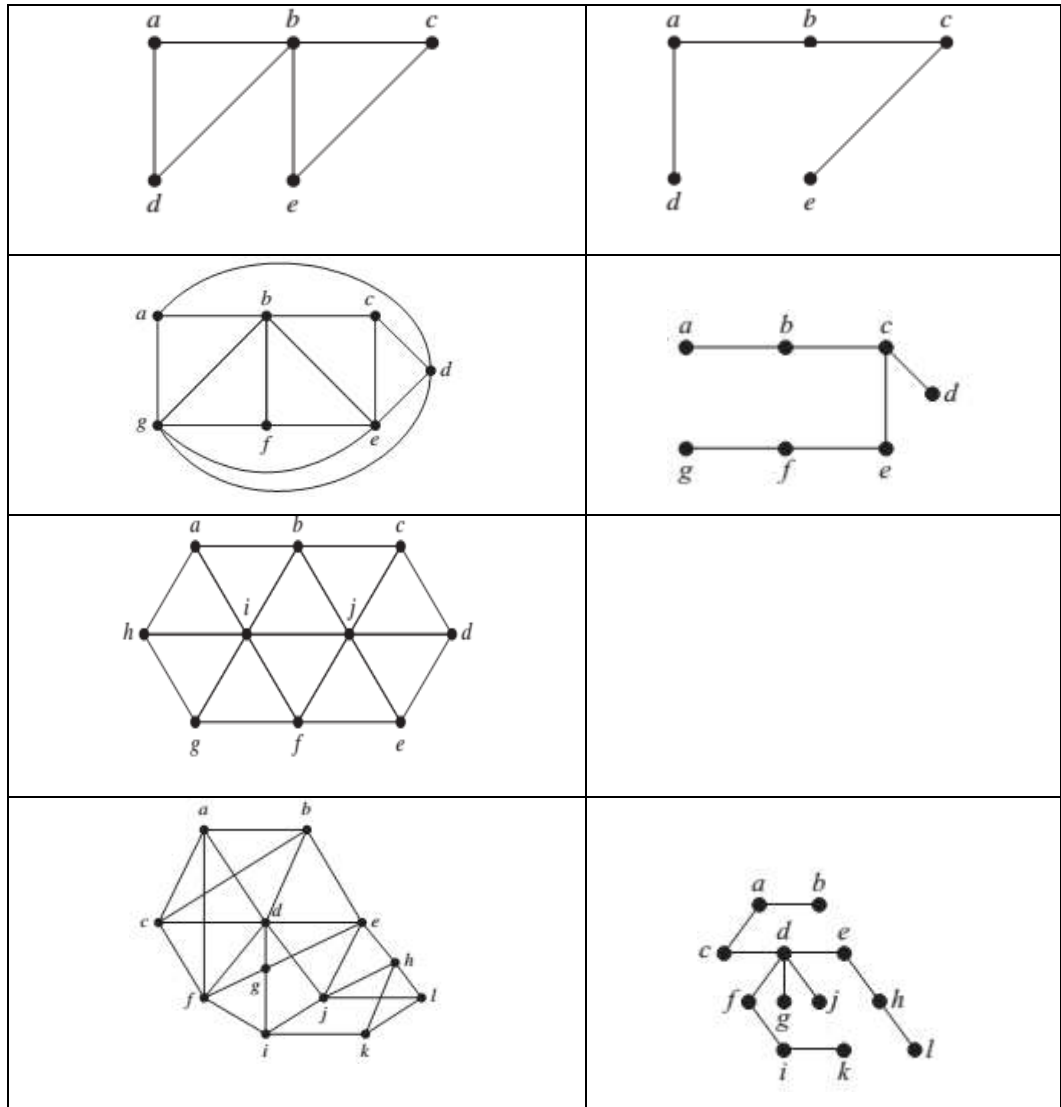
Solution:

The graph has  $e$  edges. The spanning tree has  $v - 1$  edges.

Therefore we need to remove :  $e - (v - 1)$  edges.

Hence Graph has :  $e - v + 1$  edges.

- **Example 19:** Find a spanning tree for each simple graph  $G$  shown in Figure below:

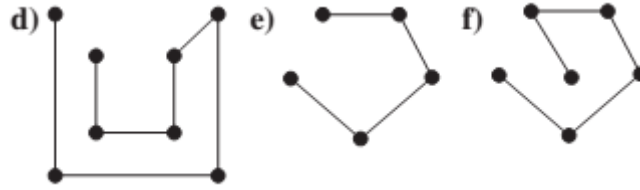


- **Example 20:** Find a spanning tree for each of these graphs.

a)  $K_5$    b)  $K_{4,4}$    c)  $K_{1,6}$    d)  $Q_3$    e)  $C_5$    f)  $W_5$

Solution:





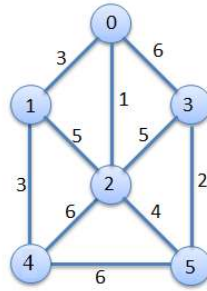
### ❖ Minimum Spanning Tree

- A minimum spanning tree in a connected weighted graph is a spanning tree that has the smallest possible sum of weights of its edges.
- A minimum spanning tree (MST) or minimum weight spanning tree is then a spanning tree with weight less than or equal to the weight of every other spanning tree.
- There are two algorithm to find minimum spanning tree.



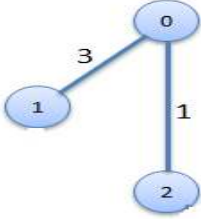
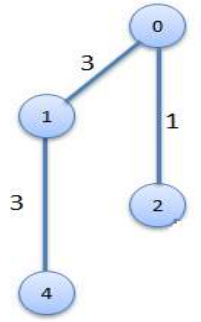
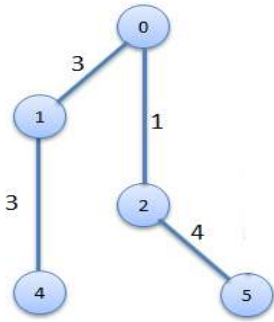
### ❖ Prim's Algorithm

- Prim's algorithm is a greedy algorithm that finds a minimum spanning tree for a connected weighted undirected graph.
- It finds a subset of the edges that forms a tree that includes every vertex, where the total weight of all the edges in the tree is minimized.
- This algorithm is directly based on the MST( minimum spanning tree) property.
- The algorithm may informally be described as performing the following steps:
  1. Initialize a tree with a single vertex, chosen arbitrarily from the graph.
  2. Grow the tree by one edge: of the edges that connect the tree to vertices not yet in the tree, find the minimum-weight edge, and transfer it to the tree.
  3. Repeat step 2 (until all vertices are in the tree).
- Following are details step of implementation of Prim's algorithm:
  - ✓ Let  $G(V,E)$  be connected weighted graph
  - ✓ **Step 1:** Select any vertex  $V_0$  in graph. Set  $T = \{V_0, \emptyset\}$
  - ✓ **Step 2:** Find edge  $e_i = (V_0, V_i)$  in  $E$  such that its one vertex is  $V_0 \in T$  and its weight is minimum. Therefore New Set  $T = \{\{V_0, V_1\}, \{e_i\}\}$
  - ✓ **Step 3:** Choose next edge  $e_k = (V_k, V_j)$  in such a way that it's one vertex  $V_k \in T$  and other vertex  $V_j \notin T$  and weight of  $e_k$  is as small as possible. Again join vertex  $V_j$  and edge  $e_k$  to  $T$ .
  - ✓ **Step 4:** Repeat step 3 until  $T$  contain all vertices of graph  $G$ . The result will be MST of graph  $G$ .

➤ **Example 21:** Using Prim's Algorithm Find MST for given graph.

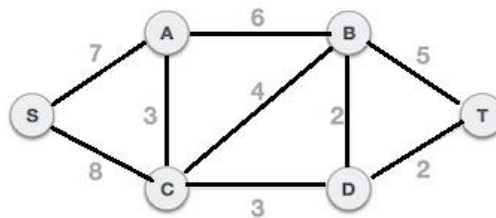


Solution: Procedure for finding Minimum Spanning Tree using Prim's Algorithm

<p><b>Step 1:</b> Select Vertex 0 as a starting vertex. Therefore <math>T = \{\{0\}, \emptyset\}</math></p>	
<p><b>Step 2:</b> Vertex 0 is adjacent to vertex 1, 2 and 3. Edge <math>(0, 1) = 3</math>, edge <math>(0, 2) = 1</math> and edge <math>(0, 3) = 6</math>. Select edge with minimum weight i.e. say <b>edge <math>e_1 = (0,2) = 1</math></b> Therefore <math>T = \{\{0, 2\}, e_1\}</math>.</p>	
<p><b>Step 3:</b> Now Vertex 0 is adjacent to vertex 1, and 3 Edge <math>(0, 1) = 3</math> and edge <math>(0, 3) = 6</math>. Vertex 2 is adjacent to 1, 3, 4, and 5 Edge <math>(2, 1) = 5</math>, edge <math>(2, 3) = 5</math>, edge <math>(2, 4) = 6</math> and edge <math>(2, 5) = 4</math> Select edge with minimum weight. i.e. say <b>edge <math>e_2 = (0, 1) = 3</math></b> Therefore <math>T = \{\{0, 2, 1\}, e_1, e_2\}</math>.</p>	
<p><b>Step 4:</b> Now Vertex 0 is adjacent to vertex 3 Edge <math>(0, 3) = 6</math>. Vertex 2 is adjacent to 3, 4, and 5 Edge <math>(2, 3) = 5</math>, edge <math>(2, 4) = 6</math> and edge <math>(2, 5) = 4</math> Vertex 1 is adjacent to vertex 4. Edge <math>(1, 4) = 3</math>. Select edge with minimum weight. i.e. say <b>edge <math>e_3 = (1,4) = 3</math></b> Therefore <math>T = \{\{0, 2, 1, 4\}, e_1, e_2, e_3\}</math>.</p>	
<p><b>Step 5:</b> Now Vertex 0 is adjacent to vertex 3 Edge <math>(0, 3) = 6</math>. Vertex 2 is adjacent to 3, and 5 Edge <math>(2, 3) = 5</math> and edge <math>(2, 5) = 4</math> Vertex 4 is adjacent to 5 Edge <math>(4, 5) = 6</math> Select edge with minimum weight. i.e. say <b>edge <math>e_4 = (2, 5) = 4</math></b> Therefore <math>T = \{\{0, 2, 1, 4, 5\}, e_1, e_2, e_3, e_4\}</math>.</p>	

<p><b>Step 5:</b> Now Vertex 0 is adjacent to vertex 3. Edge <math>(0, 3) = 6</math>.                  Vertex 2 is adjacent to 3. Edge <math>(2, 3) = 5</math>                  Vertex 5 is adjacent to 3. Edge <math>(5, 3) = 2</math>                  Select edge with minimum weight.                  i.e. say <b>edge <math>e_5 = (5, 3) = 2</math></b>                  Therefore <math>T = \{(0, 2, 1, 4, 5, 3), e_1, e_2, e_3, e_4, e_5\}</math>.</p>	
<p><b>Step 6:</b> All vertex are include in spanning tree, and graph obtained is the minimum spanning tree.                  Minimum Cost = <math>1 + 2 + 3 + 3 + 4 = 13</math></p>	

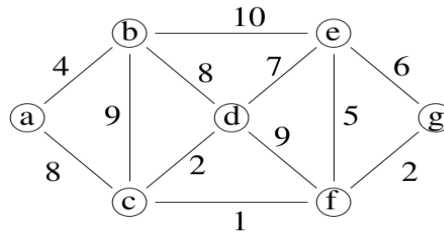
➤ **Example 22:** Using Prim's Algorithm Find MST for given graph.

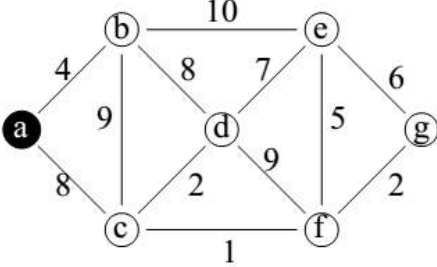
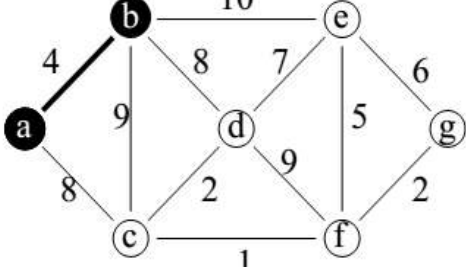
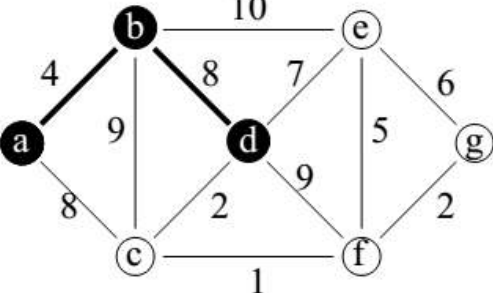
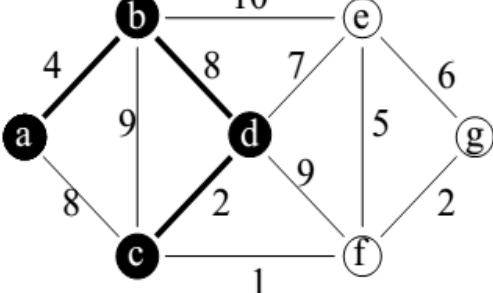
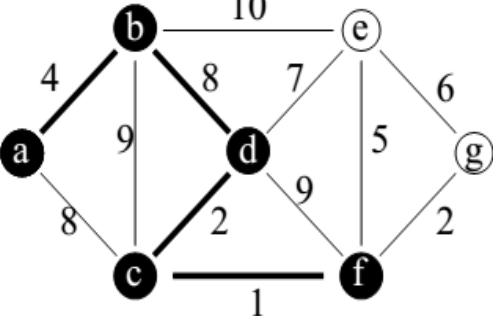
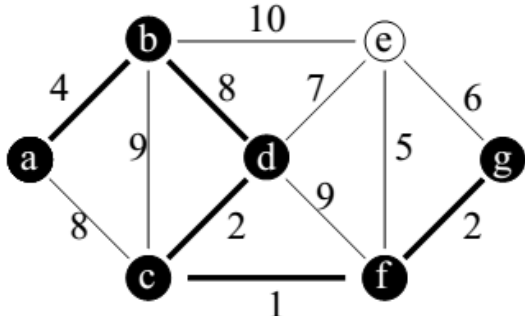
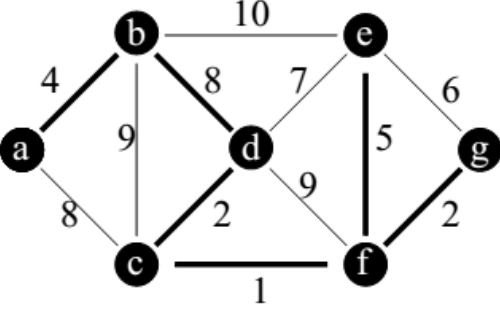


Solution:

<p>Step 1: Select vertex S as initial vertex and then edge SA is minimum so select SA.</p>	<p>Step 2: Edge AC is minimum so select AC</p>
<p>Step 3: Edge CD is minimum so select CD</p>	<p>Step 4: Edge DB &amp; DT is minimum so select both DB &amp; DT.</p>
<p>Step 5: All vertex are include in spanning tree, and graph obtained is the minimum spanning tree.                  Minimum Cost = <math>7 + 3 + 3 + 2 + 2 = 17</math></p>	<p>Spanning Tree:</p>

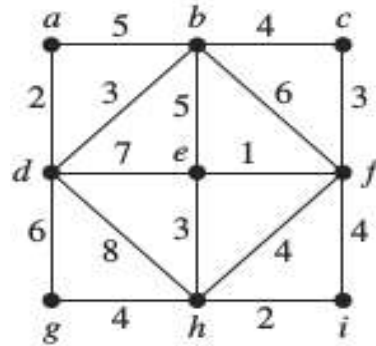
➤ **Example 23:** Using Prim's Algorithm Find MST for given graph.



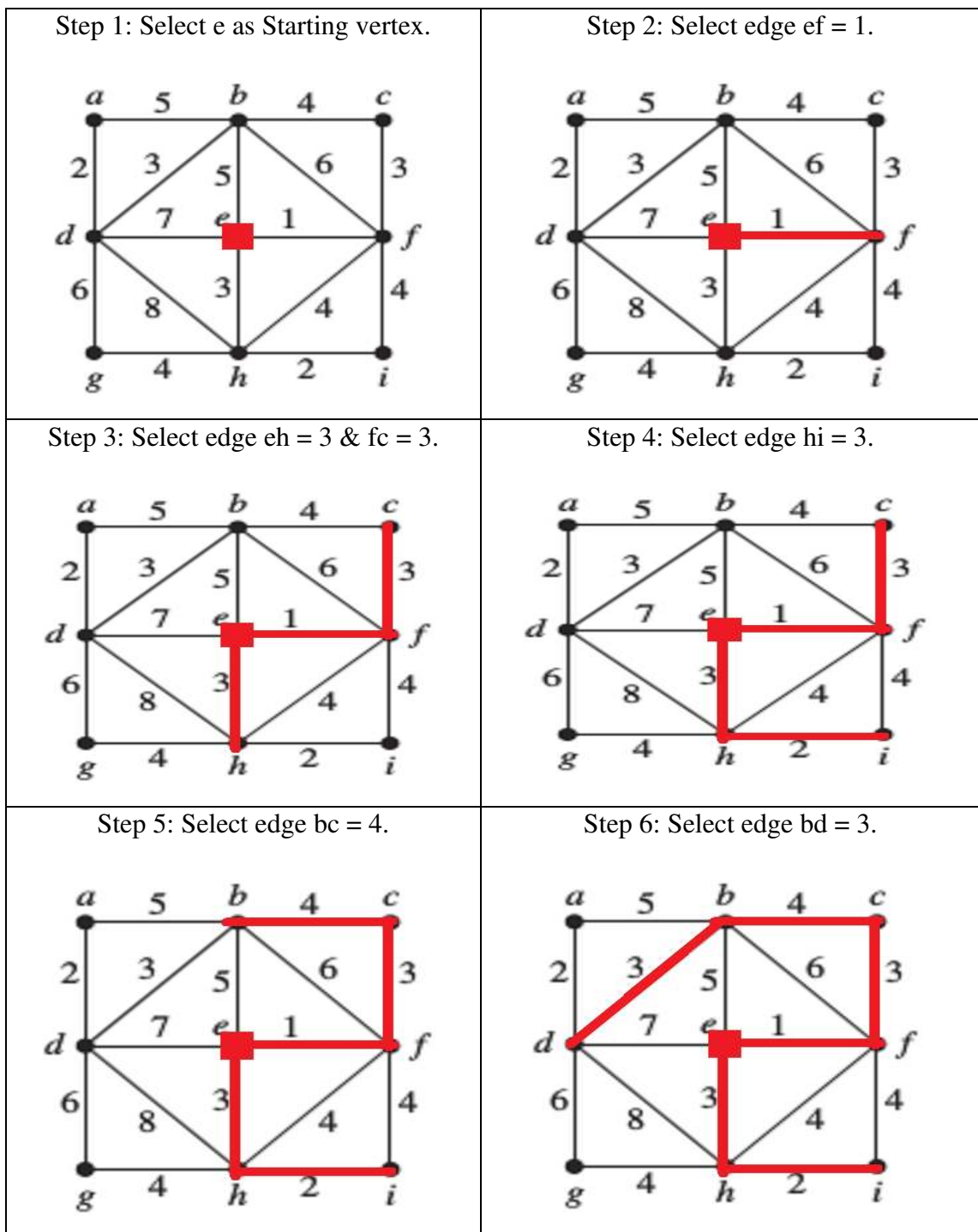
<p>Step 1: Select a as starting vertex.</p> 	<p>Step 2: Select edge ab = 04.</p> 
<p>Step 3: Select edge bd = 8.</p> 	<p>Step 4: Select edge dc = 2.</p> 
<p>Step 5: Select edge cf = 1.</p> 	<p>Step 6: Select edge fg = 2.</p> 
<p>Step 7: Select edge fe = 5.</p> 	<p>Step 8: All vertex are include in spanning tree, and graph obtained is the minimum spanning tree.</p> <p>Minimum Cost =  <math>4 + 8 + 2 + 1 + 2 + 5 = 22.</math></p>

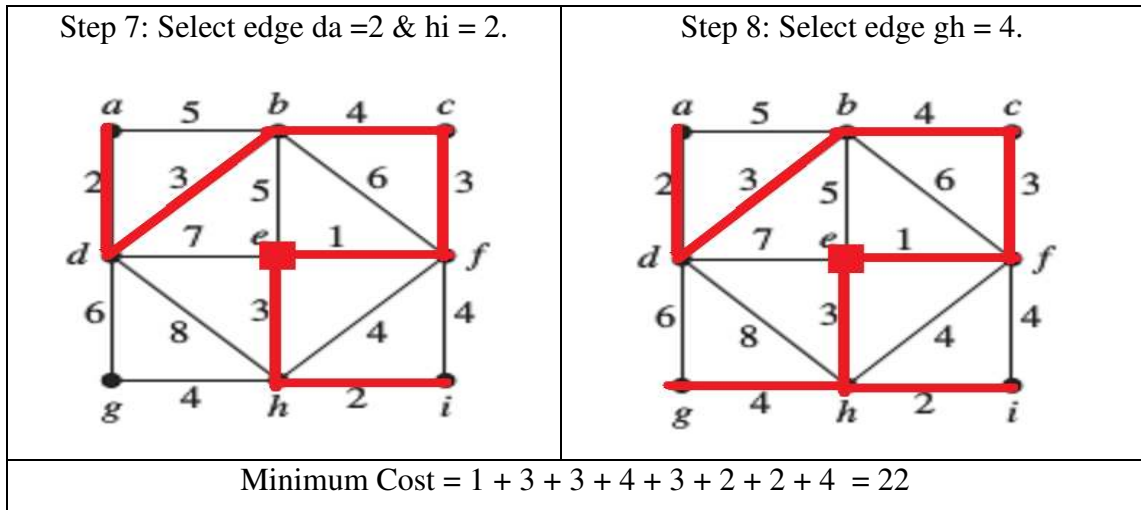


➤ **Example 24:** Using Prim's Algorithm Find MST for given graph.

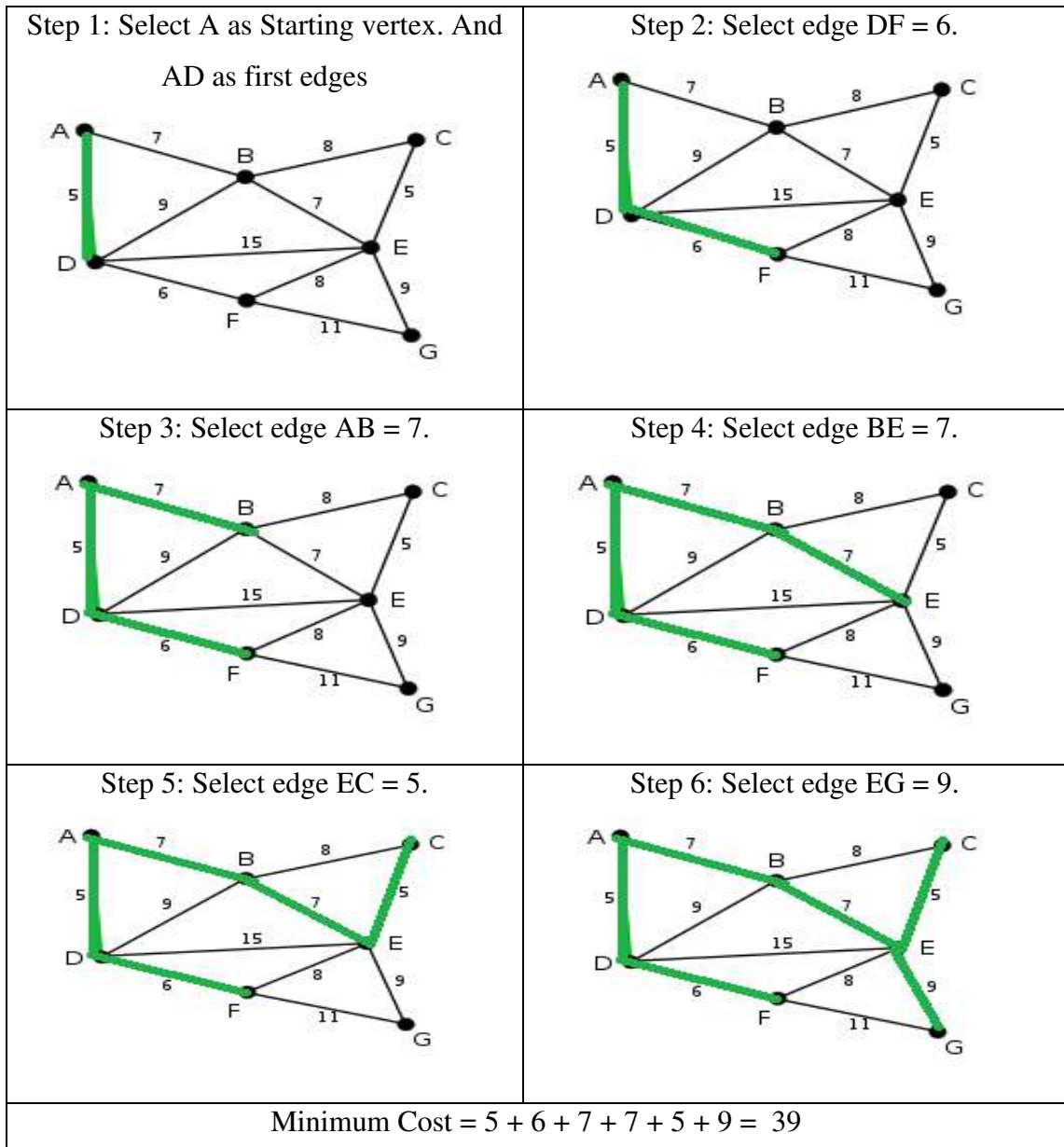


Selecting a as starting vertex also yeild same result and gives same edges.

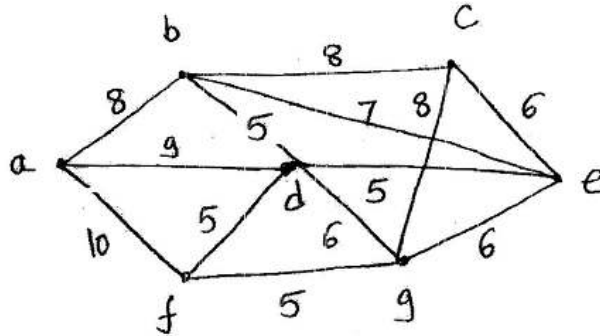




➤ **Example 25:** Obtain the minimum spanning tree using Prim's algorithm for the following graph. Obtain the total cost of minimum spanning tree.



- **Example 26:** Obtain the minimum spanning tree using Prim's algorithm for the following graph. Obtain the total cost of minimum spanning tree.

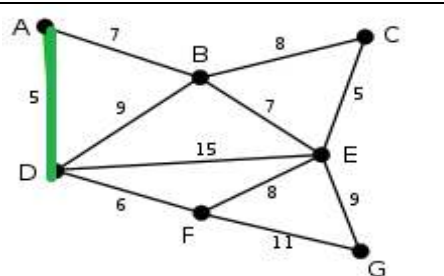


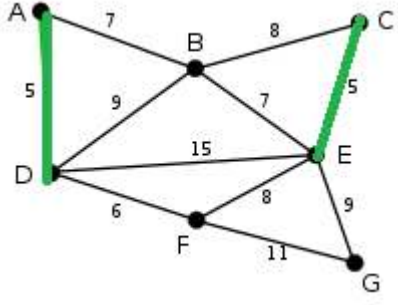
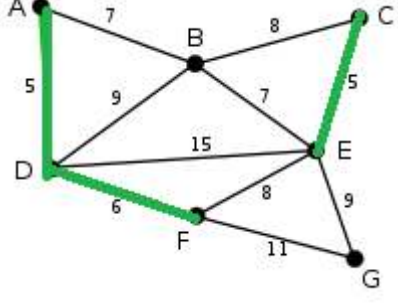
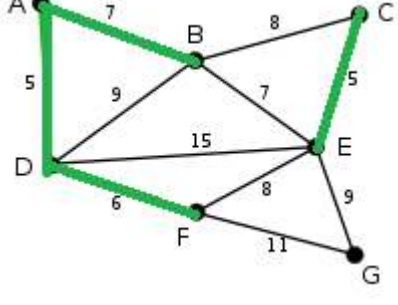
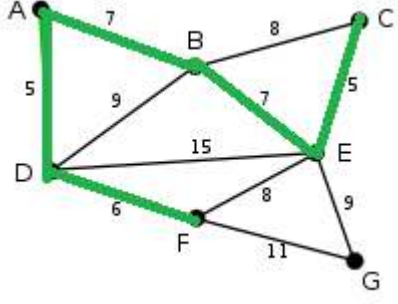
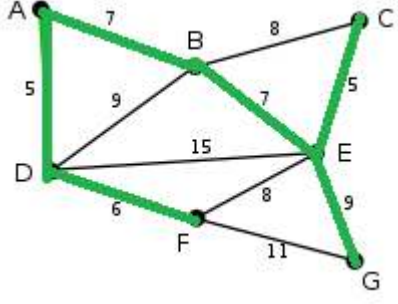
❖ **Kruskal's Algorithm**

- Kruskal's algorithm is a minimum-spanning-tree algorithm which finds an edge of the least possible weight that connects any two trees in the forest.
- It is a greedy algorithm in graph theory as it finds a minimum spanning tree for a connected weighted graph adding increasing cost arcs at each step.
- This means it finds a subset of the edges that forms a tree that includes every vertex, where the total weight of all the edges in the tree is minimized.
- If the graph is not connected, then it finds a minimum spanning forest.
- Following are details step of implementation of Kruskal's algorithm.
  - ✓ Let  $G(V,E)$  be weighted connected graph.
  - ✓ **Step 1:** Pick up the  $e_i$  of the Graph  $G$  such that its weights  $W(e_i)$  is minimum. (If there are more edges of the minimum weight then select all those edges which do not form circuit)
  - ✓ **Step 2:** If edges  $e_1, e_2, e_3, \dots, e_n$  have been chosen then pick an edge  $e_{n+1}$  such that
    - $e_{n+1} \neq e_i$  for  $i=1, 2, 3, \dots, n$
    - The edges  $e_1, e_2, e_3, \dots, e_n, e_{n+1}$  do not form circuit.
    - $W(e_{n+1})$  is as small as possible subject to condition (ii).
  - ✓ **Step 3:** Stop procedure when step two cannot be implemented further.
- **Example 27:** Using Kruskal algorithm Find MST for given graph.

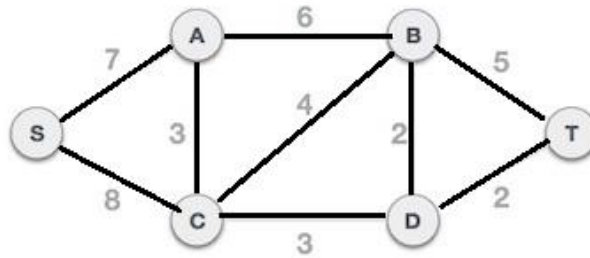
Solution:

1. AD and CE are the shortest edges, with length 5, and AD has been arbitrarily chosen, so it is highlighted.



<p>2. CE is now the shortest edge that does not form a cycle, with length 5, so it is highlighted as the second edge.</p>	
<p>3. The next edge, DF with length 6, is highlighted using much the same method.</p>	
<p>4. The next-shortest edges are AB and BE, both with length 7. AB is chosen arbitrarily, and is highlighted. The edge BD has been highlighted in red, because there already exists a path (in green) between B and D, so it would form a cycle (ABD) if it were chosen</p>	
<p>5. The process continues to highlight the next-smallest edge, BE with length 7. Many more edges are highlighted in red at this stage: BC because it would form the loop BCE, DE because it would form the loop DEBA, and FE because it would form FEBAD.</p>	
<p>6. Finally, the process finishes with the edge EG of length 9, and the minimum spanning tree is found.</p>	

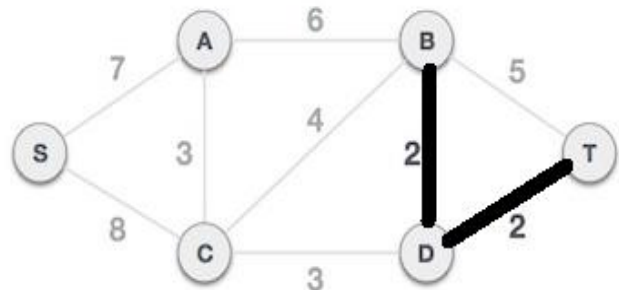
➤ **Example 28:** Using Krushal’s Algorithm Find MST for given graph.



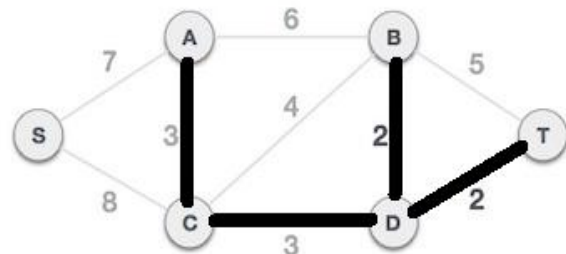
Solution: Arrange all edges in their increasing order of weight.

BD	DT	AC	CD	CB	BT	AB	SA	SC
2	2	3	3	4	5	6	7	8

Now we start adding edges to the graph beginning from the one which has the least weight. The least cost is 2 and edges involved are B,D and D,T. We add them. Adding them does not violate spanning tree properties, so we continue to our next edge selection.



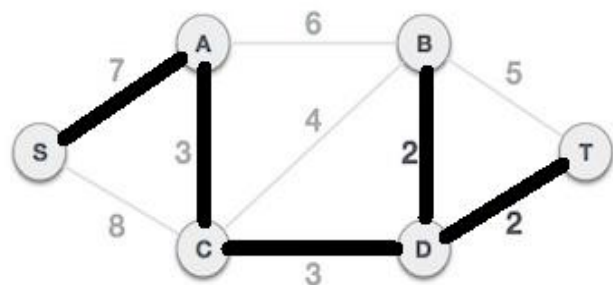
Next cost is 3, and associated edges are A,C and C,D. We add them again.



Next cost in the table is 4, and we observe that adding it will create a circuit in the graph. We ignore it. In the process we shall ignore/avoid all edges that create a circuit. We observe that edges with cost 5 and 6 also create circuits. We ignore them and move on.

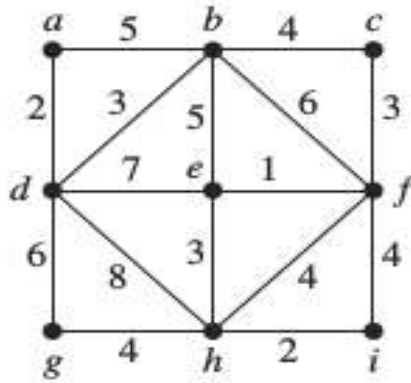
Now we are left with only one node to be added. Between the two least cost edges available 7 and 8, we shall add the edge with cost 7.

By adding edge S, A we have included all the nodes of the graph and we now have minimum cost spanning tree.



Minimum Cost Spanning Tree = 2 + 2 + 3 + 3 + 7 = 17.

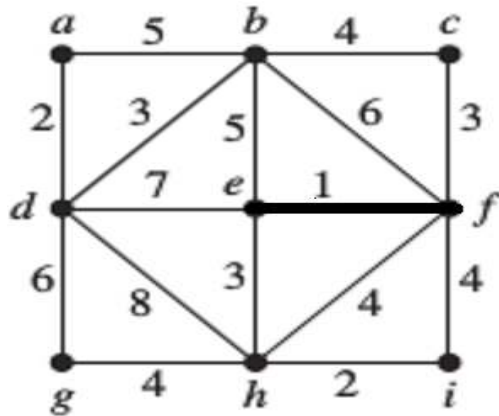
➤ **Example 29:** Using Krushal’s Algorithm Find MST for given graph.



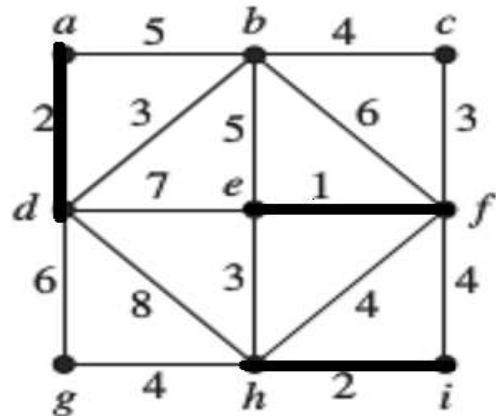
Solution: Arrange all edges in their increasing order of weight

ef	ad	hi	bd	cf	eh	bc	fi	gh	fh	be	ab	bf	dg	de	dh
1	2	2	3	3	3	4	4	4	4	5	5	6	6	7	8

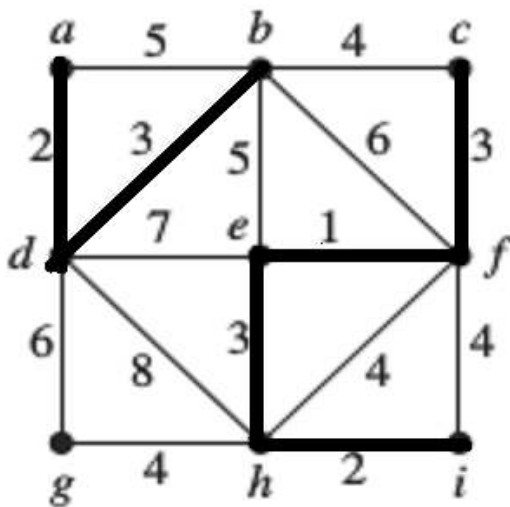
Step 1: Select edge ef = 1.



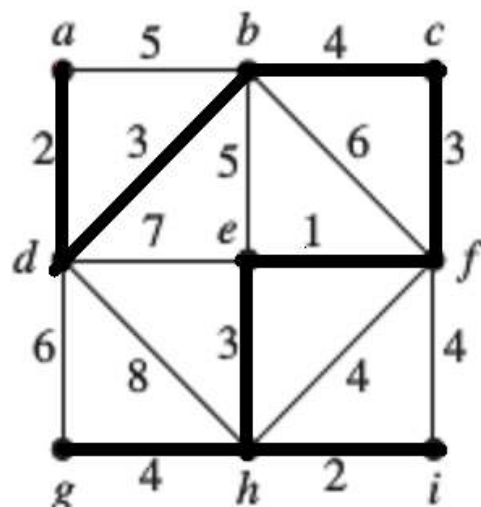
Step 2: Select edge ad & hi = 2.



Step 3: Select edge bd , eh & fc = 3.



Step 4: Select edge bc & gh = 4.

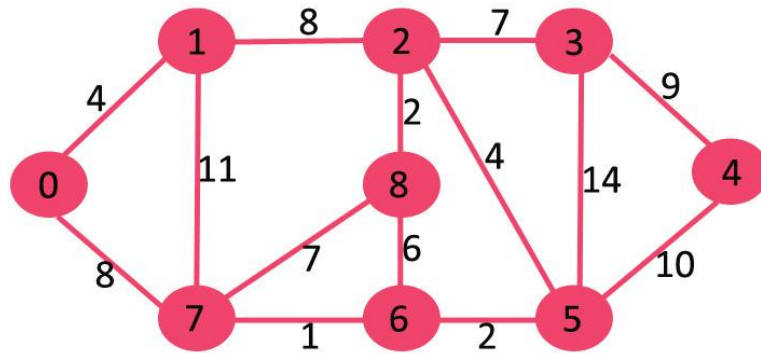


Other all edges form a circuit so we neglect it.

Minimum Cost = 1 + 2 + 2 + 3 + 3 + 3 + 4 + 4 = 22



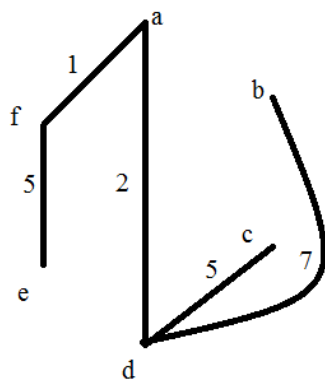
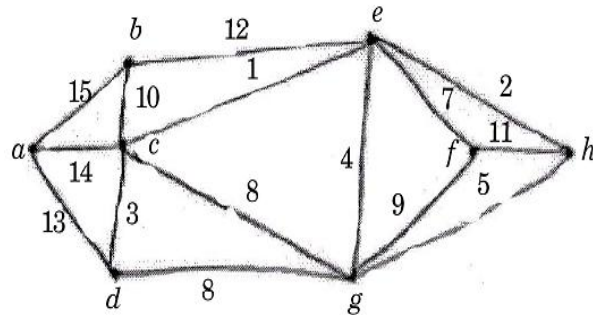
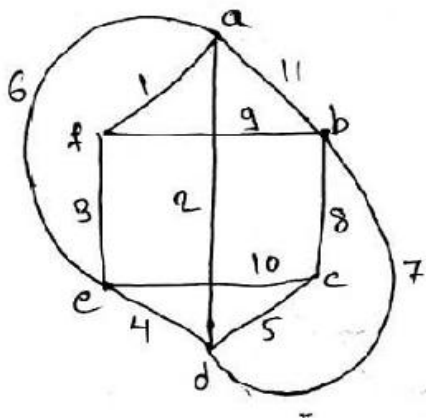
➤ **Example 30:** Using Prim's and Krushal's algorithm find MST for following graph



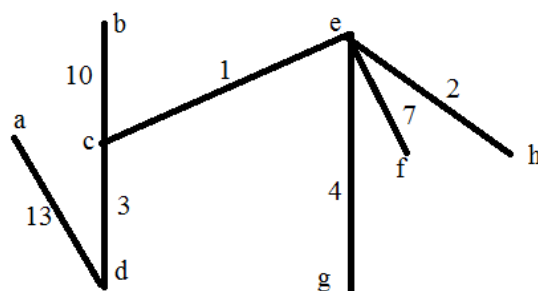
Prims Algorithm	Krushal Algorithm

<p>MST = 4 + 8 + 1 + 2 + 4 + 2 + 7 + 9 = 37</p>	

➤ **Example 31:** Find the minimum spanning tree of the given graph using Kruskal's algorithm



Weight = 16

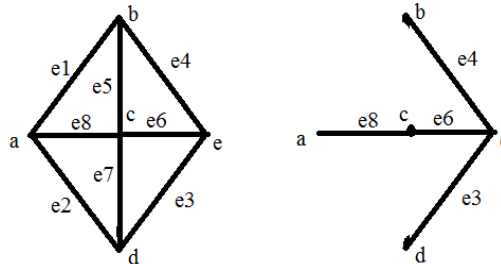


Weight = 40



❖ **Fundamental Circuit**

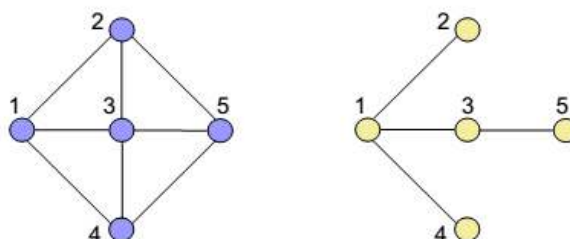
- For a Graph G and its associated spanning tree T, the fundamental circuit of a chord 'e' of T is given by the circuit formed due to addition of 'e' in T.
- In other words Let G be connected graph and Let T be spanning tree of G. Consider a chord of T i.e. an edge of G which is not in T. Then 'T + e' contain a unique circuit called as fundamental circuit.
- For different spanning tree of graph G, the fundamental circuit will be different.
- The number of fundamental circuit in any spanning tree is equal to the numbers of chords of that spanning tree.
- Hence if the connected graph G has v number of vertices and e number of edges then spanning tree T has (v-1) branches and ( e - v + 1 ) numbers of chords.
- Thus there will be ( e - v + 1 ) number of fundamental circuit in G with respect to the spanning tree T.
- **Example 32:** Find the fundamental circuit for the graph show in figure below w.r.t the spanning tree T.



Solution:

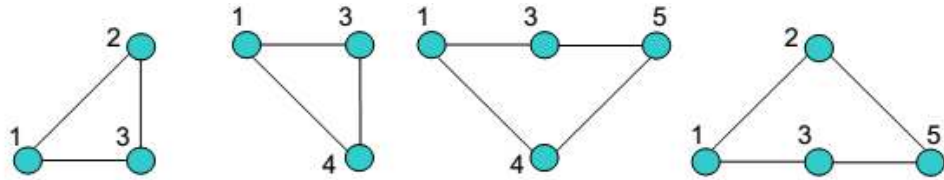
Chord	Fundamental Circuit
e1	{ e1, e4, e6, e8 }
e2	{ e2, e3, e6, e8 }
e5	{ e5, e4, e6 }
e7	{ e7, e3, e6 }

- **Example 33:** Find the fundamental circuit for the graph show in figure below w.r.t the spanning tree T.



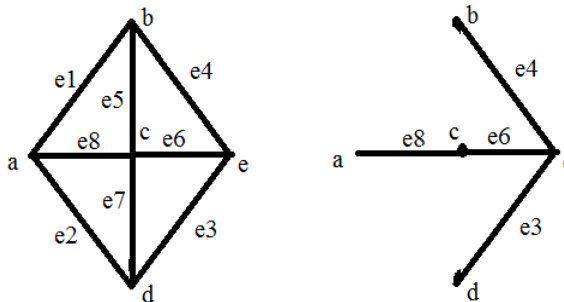
Solution:

Fundamental circuit for above graph G and spanning tree T are



❖ **Fundamental Cut-sets**

- Let T be the spanning tree of the connected graph G.
- The removal of any branch from a spanning tree, breaks the spanning tree into two trees i.e there is 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 fundamental cut set of the graph G with respect to the spanning tree T is called fundamental system of cut set.
- 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 spanning tree has (v-1) branches which is the same as the number of fundamental circuit.
- **Example 34:** Find the fundamental circuit for the graph show in figure below w.r.t the spanning tree T.

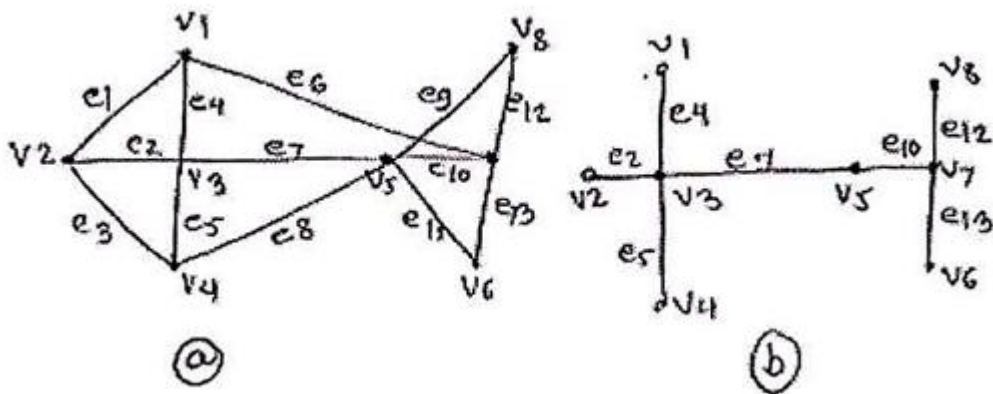


Solution:

Branch	Fundamental cutset
e4	{ e1, e5, e4 }
e3	{ e2, e7, e3 }
e6	{ e1, e5, e6, e7, e2 }
e8	{ e1, e8, e2 }

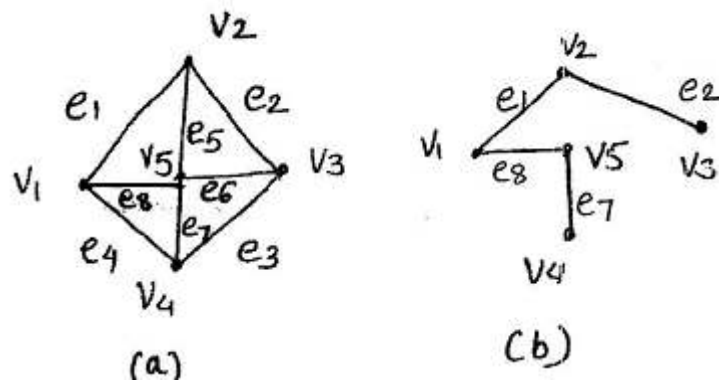
- **Example 35:** Find the fundamental system of cutset for the graph show in figure below w.r.t the spanning tree T.

Solution: The spanning tree T has 7 branches {e2,e4,e5,e7,e10,e12,e13}. Therefore its has 7 fundamental cutset.

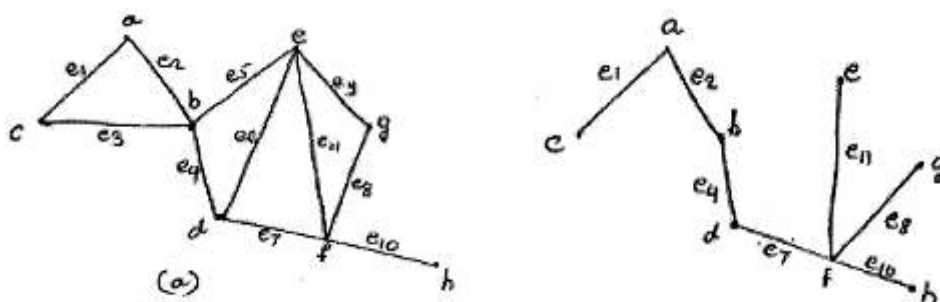


Branch	Fundamental cutset
e2	{e1,e2,e3}
e4	{e1,e4,e6}
e5	{e3,e5,e8}
e7	{e6,e7,e8}
e10	{e6,e9,e10,e11}
e12	{e9,e12}
e13	{e11,e13}

➤ **Example 36:** Find the fundamental system of cutset for the graph show in figure below w.r.t the spanning tree T.



➤ **Example 37:** Find the fundamental circuit and fundamental system of cutset for the graph show in figure below w.r.t the spanning tree T.



### ❖ Network Flow

- In graph theory, a **flow network** (also known as a **transportation network**) is a directed graph where each edge has a **capacity** and each edge receives a flow.
- The amount of flow on an edge cannot exceed the capacity of the edge.
- A directed graph is called a **network**, the vertices are called **nodes** and the edges are called **arcs**.
- A flow must satisfy the restriction that the amount of flow into a node equals the amount of flow out of it.
- It is a **source**, which has only outgoing flow, and **sink** which has only incoming flow.
- A network can be used to model traffic in a road system, circulation with demands, fluids in pipes, currents in an electrical circuit, or anything similar in which something travels through a network of nodes.
- Two Methods for finding Max Flow in a transport network
  - ✓ Ford Fulkerson Algorithm
  - ✓ Labeling Procedure Algorithm

#### Step of Labeling Procedure Algorithm

- Suppose a transport network with the capacity of each edge is given.
  1. Initially, assign the zero flow to each edge of the given network.  
Also assign the label  $(-, \infty)$  to the source  $s$ .
  2. Scan all those vertices which are adjacent to the source  $s$ .  
Suppose the vertex  $b$  is adjacent to  $s$  and  $C(s,b) > F(s,b)$  then label the vertex  $b$  as  $(a^+, \Delta b)$  where  $\Delta b = C(s,b) - F(s,b)$ .  
The vertex  $b$  is not labeled if  $C(s,b) = F(s,b)$ .
  3. Scan all those vertices which are adjacent to labeled vertices. Suppose the vertex  $q$  is adjacent to the labeled vertex  $b$ . The vertex  $q$  is labeled as  $(b^+, \Delta q)$ .  
where  $\Delta q = \min(\Delta b, [C(b,q) - F(b,q)])$  if  $[C(b,q) > F(b,q)]$ .  
The vertex  $q$  is not labeled if  $C(b,q) = F(b,q)$ .  
Also the vertex  $q$  is labeled  $(b^-, \Delta q)$ , where  $\Delta q = \min(\Delta b, F(b,q))$  if  $F(b,q) > 0$ .
  4. Repeat step 3 till we reach sink  $z$  vertex.
  5. If the sink  $z$  is labeled with the label  $(y^+, \Delta z)$  where  $y$  is some labeled vertex, then increase flow of the edge  $(y,z)$  from  $F(y,z)$  to  $F(y,z) + \Delta z$ .  
Note that the vertex  $y$  must be labeled either  $(q^+, \Delta y)$  or  $(q^-, \Delta y)$  with  $\Delta y$ .  
If  $y$  is labeled  $(q^+, \Delta y)$  then increase the flow in the edge  $(q,y)$  from  $F(q,y)$  to  $F(q,y) + \Delta y$ .  
On the other hand if  $y$  is labeled  $(q^-, \Delta y)$  then decrease the flow in the edge  $(y,q)$  from  $F(y,q)$  to  $F(y,q) - \Delta y$ .

This process is continued back to source  $s$  and the value of the flow in the transport network will be increased by amount  $\Delta z$ . Again start labelling procedure to further increase the value of the flow in the network i.e. step 3.

6. If the sink is not labeled, then denote all the labeled vertices by  $P$  and unlabeled vertices by  $P^c$ . Determine the capacity of the cut  $(P, P^c)$  which is the value of the maximal flow.

**\*\*\*\*\* THE END \*\*\*\*\***