# *Second Year of Computer Engineering (2019 Course)*

# (210241): Discrete Mathematics

| Teaching Scheme | Credit Scheme | Examination Scheme and Marks |
|---|---|---|
| Lecture: 03 Hours/Week | 03 | Mid_Semester (TH): 30 Marks<br>End_Semester (TH): 70 Marks |

| Marks weightage per unit for examination | | | | | | |
|---|---|---|---|---|---|---|
| **Unit Number** | **I** | **II** | **III** | **IV** | **V** | **VI** |
| Mid_Semester | 15 | 15 | - | - | - | - |
| End_Semester | - | - | 18 | 17 | 18 | 17 |

**Prerequisites: Basic Mathematics**

# Course Objectives

*To introduce several Discrete Mathematical Structures found to be serving as tools even today in the development of theoretical computer science.*

1. To introduce students to understand, explain, and apply the foundational mathematical concepts at the core of computer science.

2. To understand use of set, function and relation models to understand practical examples, and interpret the associated operations and terminologies in context.

3. To acquire knowledge of logic and proof techniques to expand mathematical maturity.

4. To learn the fundamental counting principle, permutations, and combinations.

5. To study how to model problem using graph and tree.

6. To learn how abstract algebra is used in coding theory.

# Course Outcomes

*On completion of the course, learner will be able to –*

**CO1:** Formulate problems precisely, solve the problems, apply formal proof techniques, and explain the reasoning clearly.

**CO2**: Apply appropriate mathematical concepts and skills to solve problems in both familiar and unfamiliar situations including those in real-life contexts.

**CO3:** Design and analyze real world engineering problems by applying set theory, propositional logic and to construct proofs using mathematical induction.

**CO4:** Specify, manipulate and apply equivalence relations; construct and use functions and apply these concepts to solve new problems.

**CO5:** Calculate numbers of possible outcomes using permutations and combinations; to model and analyze computational processes using combinatorics.

**CO6:** Model and solve computing problem using tree and graph and solve problems using appropriate algorithms.

**CO7**: Analyze the properties of binary operations, apply abstract algebra in coding theory and evaluate the algebraic structures.

# Learning Resources

❖ **Text Books:**


1. C. L. Liu, "Elements of Discrete Mathematics"||, TMH, ISBN 10:0-07-066913-9.2.
2. N. Biggs, "Discrete Mathematics", 3rd Ed, Oxford University Press, ISBN 0 –19-850717–8.


❖ **Reference Books:**


1. Kenneth H. Rosen, "Discrete Mathematics and its Applications"||, Tata McGraw-Hill, ISBN 978-0-07-288008-3
2. Bernard Kolman, Robert C. Busby and Sharon Ross, "Discrete Mathematical Structures"||, Prentice-Hall of India /Pearson, ISBN: 0132078457, 9780132078450.
3. Narsingh Deo, "Graph with application to Engineering and Computer Science", Prentice Hall of India, 1990, 0 –87692 –145 –4.
4. Eric Gossett, "Discrete Mathematical Structures with Proofs", Wiley India Ltd, ISBN:978-81-265-2758-8.
5. Sriram P.and Steven S., "Computational Discrete Mathematics", Cambridge University Press, ISBN 13: 978-0-521-73311-3.

# Unit V

# *Trees*

Duration: (07 Hours)

Mapping of Course Outcomes: CO1,CO2,CO6

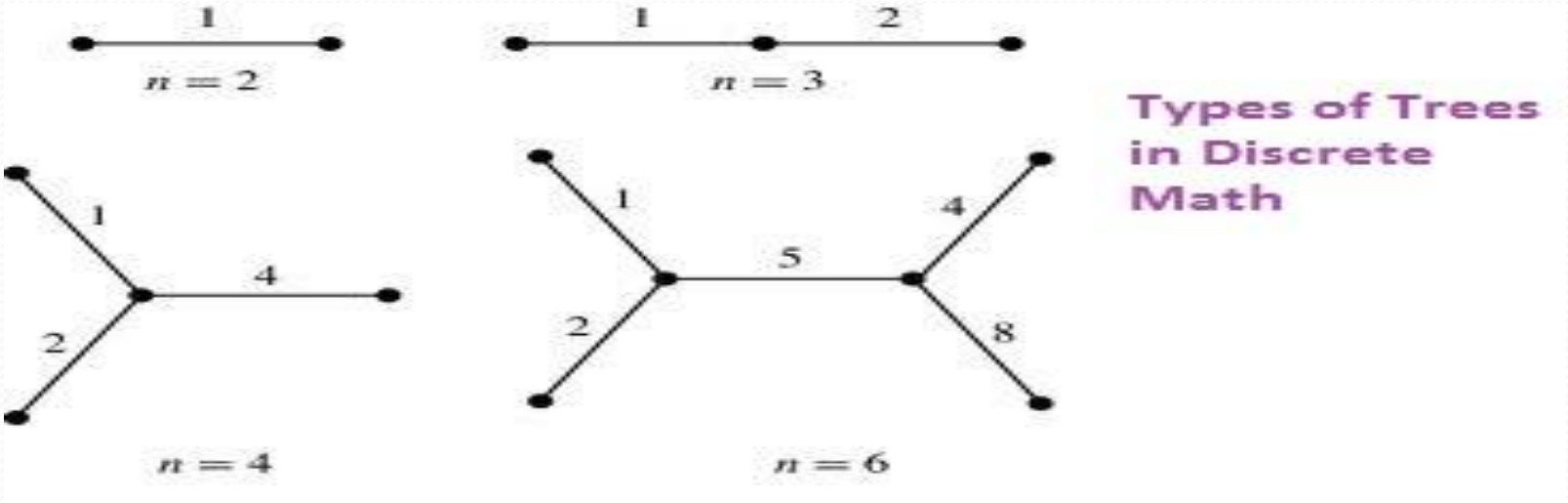
# Unit-IV: Contents

- 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.




- **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.

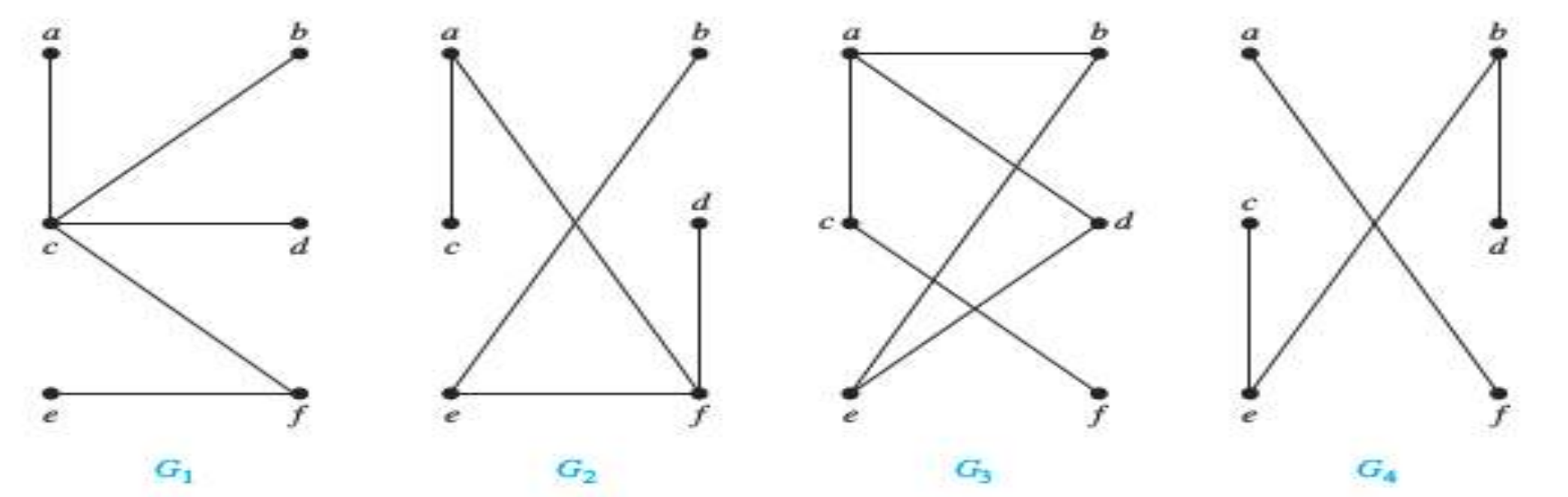
# Introduction

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

  **Solution:** G1 and G2 are trees, because both are connected graphs with no simple circuits.

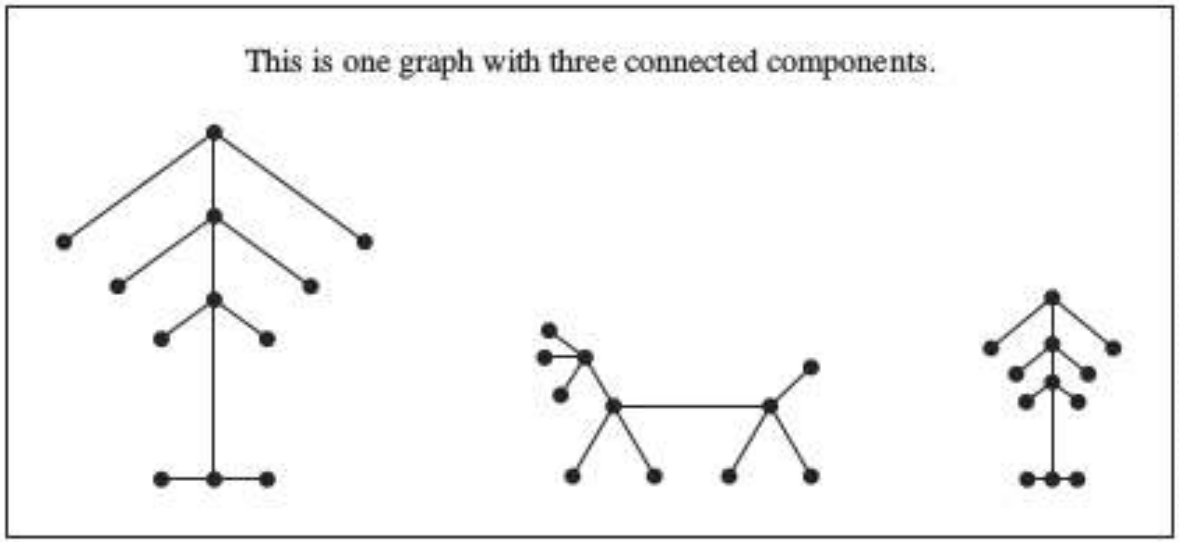
  G3 is not a tree because e, b, a, d, e is a simple circuit in this graph.

  Finally, G4 is not a tree because it is not connected



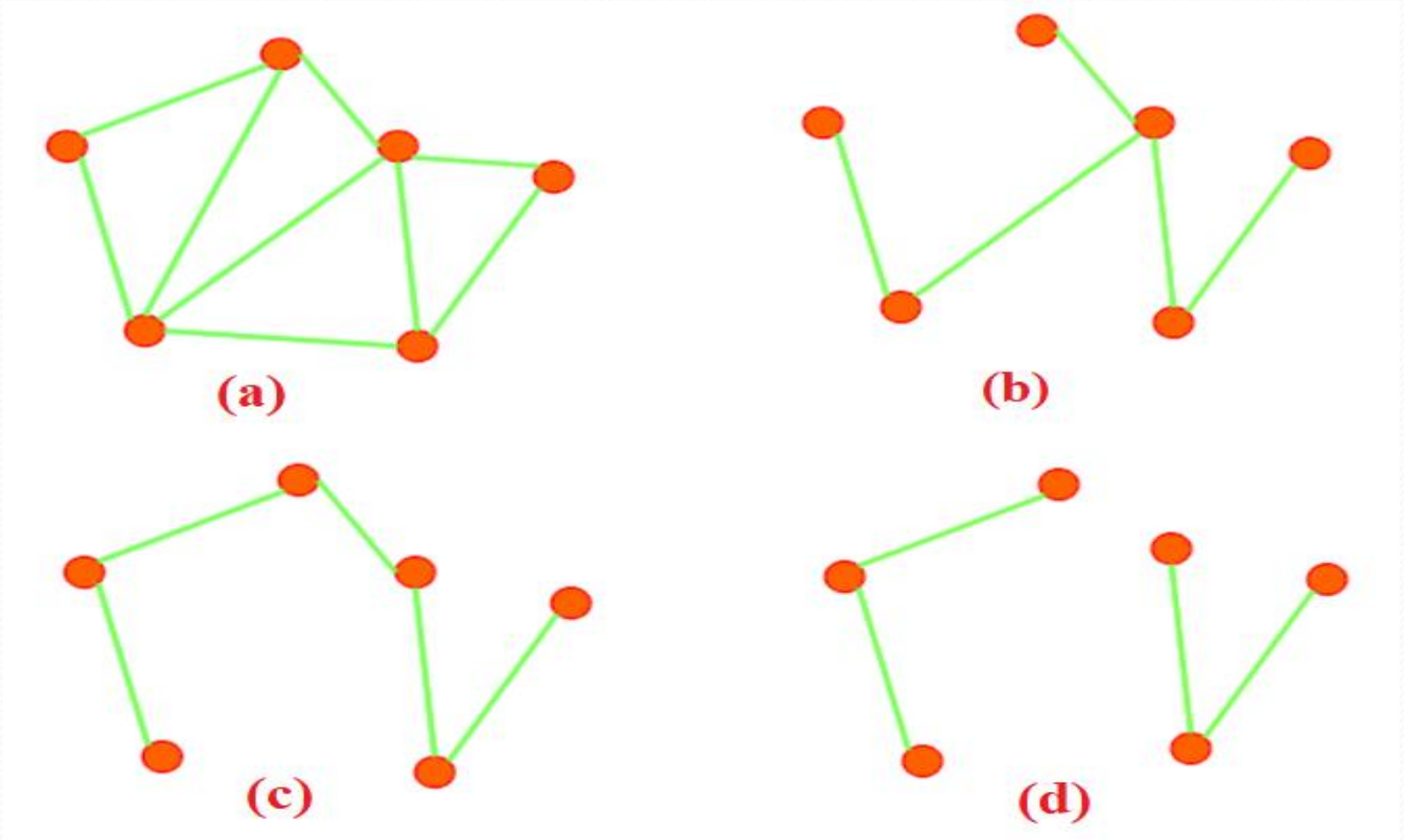
# Introduction

- 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.
- A forest is composed of one tree or some disconnected trees. A forest is a disjoint union of trees.



# Introduction

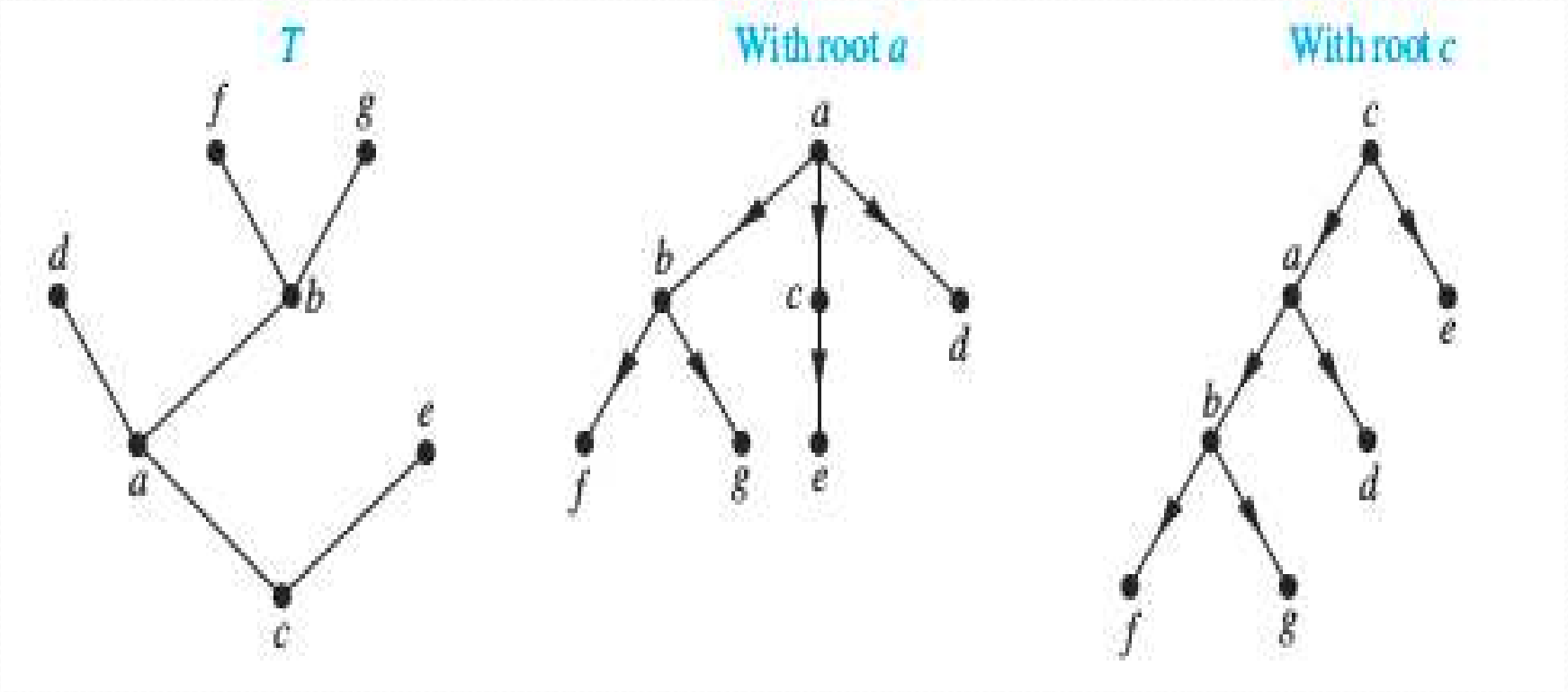
❖ **Example 2:** Are the following graphs trees?




Solution: Graph (a) and (d) are not tree while (b) & (c) tree

# 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.



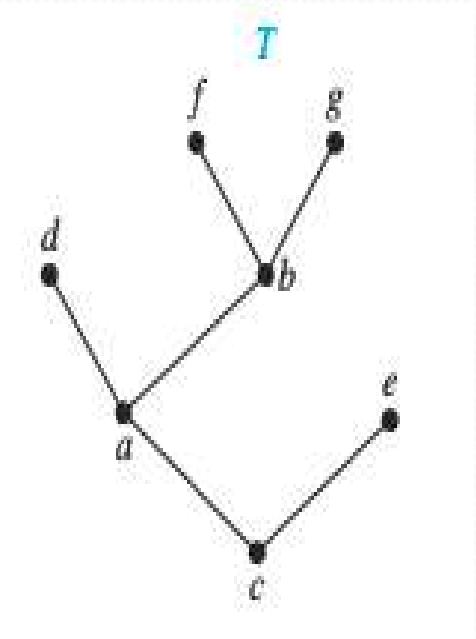
# Terminology for rooted trees

- 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.

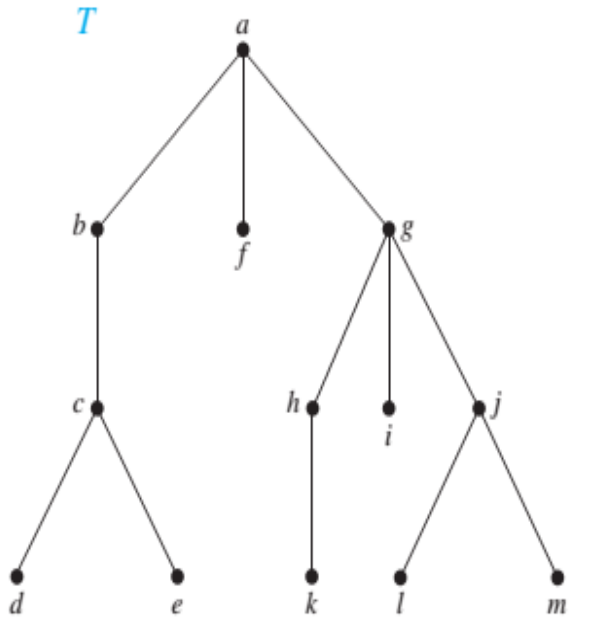
# Terminology for rooted trees

- 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.



# Terminology for rooted trees

❖ **Example 3:** In the rooted tree T(with root a) shown in Figure , 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?



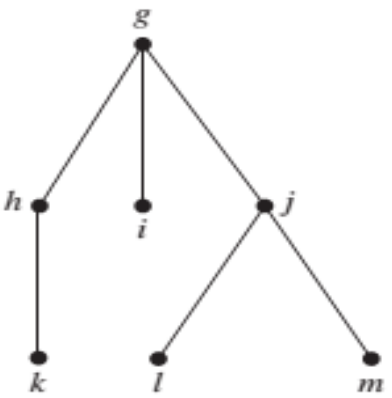

Solution: The parent of c is b.

The children of g are h, i, and j.

The siblings of h are i and j.

The ancestors of e are c, b, & a.

The descendants of b are c, d, & e.




The internal vertices are a, b, c, g, h, & j. The leaves are d, e, f, i, k, l, and m.

The subtree rooted at g is shown in Figure.

# Terminology for rooted trees

❖ **Example 4:** Answer these questions about the rooted tree illustrated in Figure.

a) Which vertex is the root? Vertex a

b) Which vertices are internal? a, b, c, d, f, h , j , q, & t.

c) Which vertices are children of j? q and r

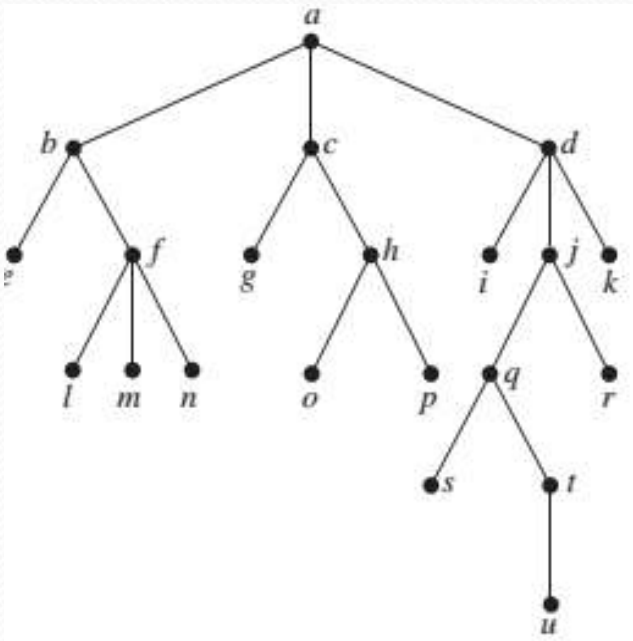
d) Which vertex is the parent of h? c

e) Which vertices are siblings of o? p

f) Which vertices are ancestors of m? f, b, and a

g) Which vertices are descendants of b? e, f, l, m, and n

h) Which vertices are leaves? e, g, i, k, l, m, n, o, p, r, s, & u.



# Terminology for rooted trees

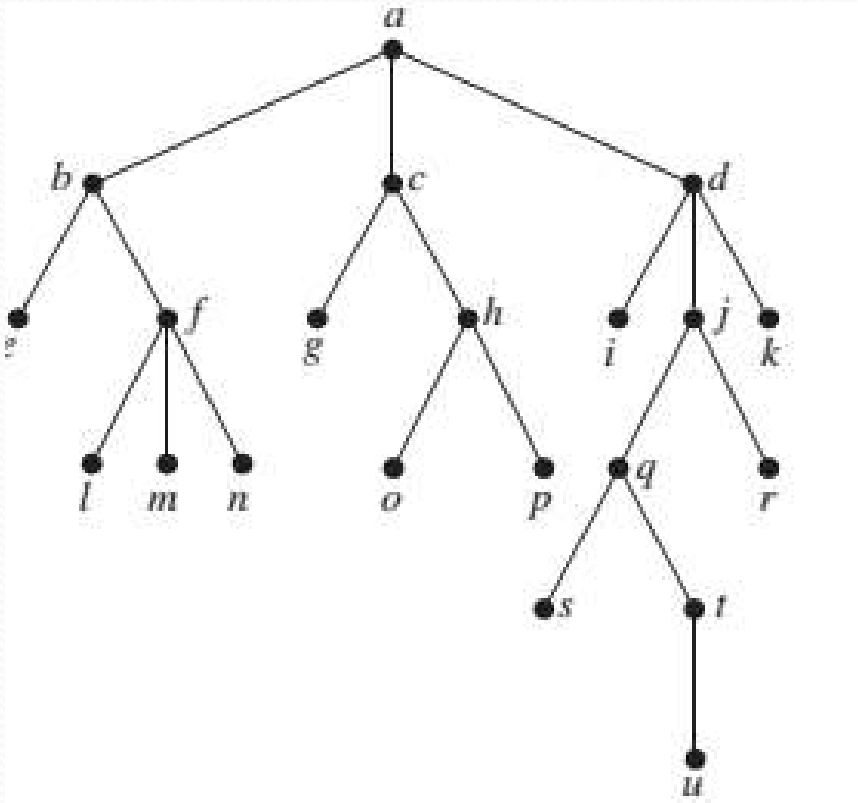
❖ **Example 5:** What is the level/length of each vertex of the rooted tree in Figure ?

**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.

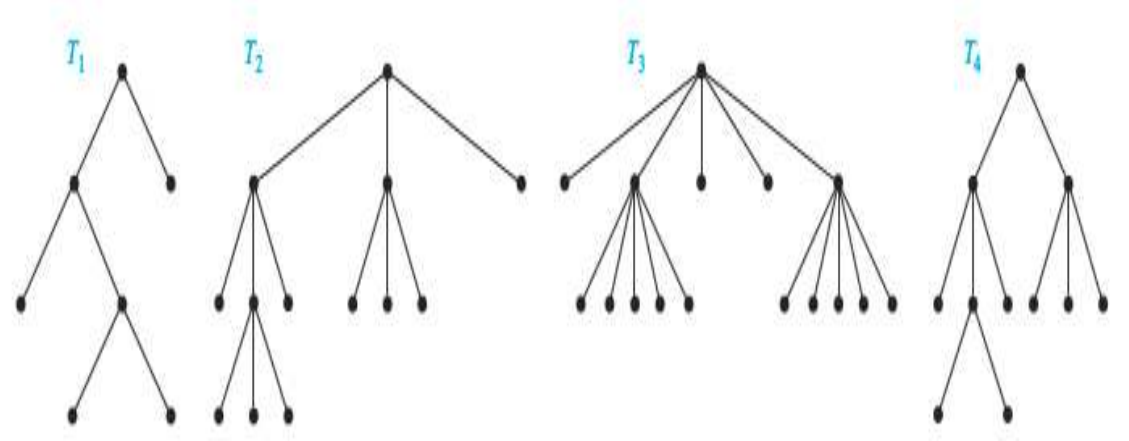
The vertices below that, namely e through k (in alphabetical order), are at level 2. Similarly l through r are at level 3, s and t are at level 4, and u is at level 5.



# 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 8 full m-ary trees for some positive integer m?

  **Solution:** T1 is a full binary tree because each of its internal vertices has two children. T2 is a full 3-ary tree because each of its internal vertices has three children. In T3 each internal vertex has five children, so T3 is a full 5-ary tree. T4 is not a full m-ary tree for any m because some of its internal vertices have two children and others have three children.



# Properties of M-ary Trees

❖ **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.*

# Prefix Code

- 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.

# Prefix Code

- 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.

  **i. e. the string 10110 is the encoding of ate**

- **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.

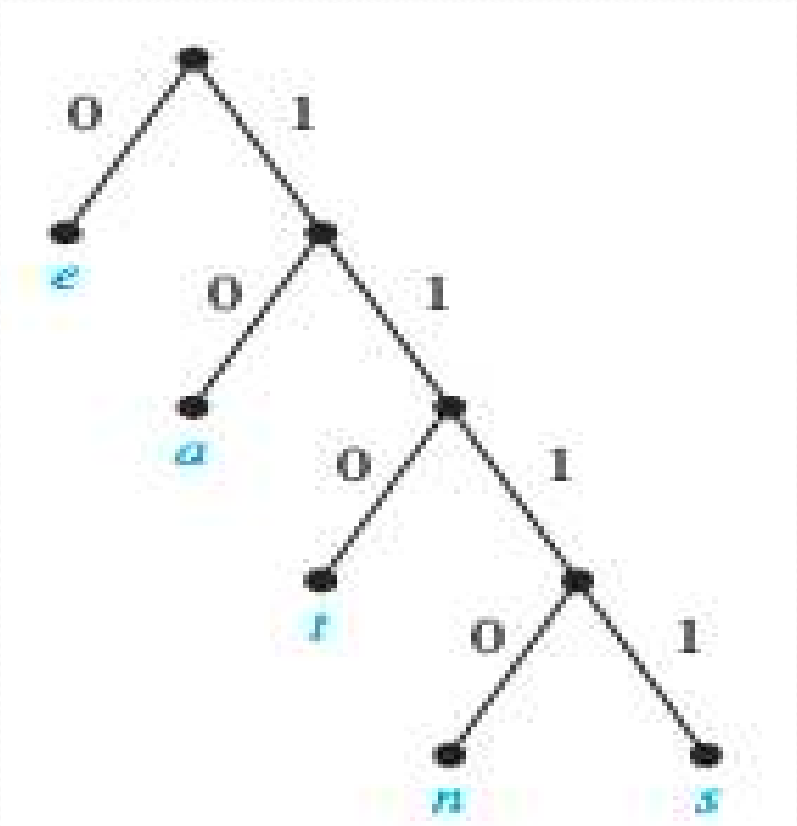
- A prefix code can be represented using a binary tree, where the characters are the labels of the leaves in the tree. The edges of the tree are labeled so that an edge leading to a left child is assigned a 0 and an edge leading to a right child is assigned a 1.
- The bit string used to encode a character 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.

# Prefix Code

- In figure the encoding is represents as:

e by 0, a by 10, t by 110, n by 1110, and

s by 1111.




- Bit string 11111011100 using the code in figure encode the message the

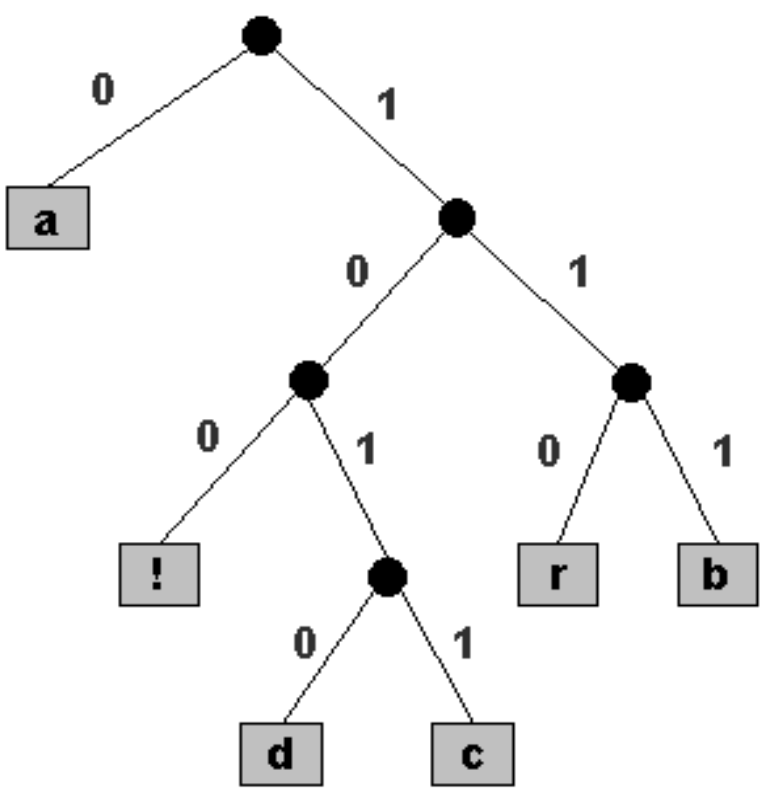
" sane "

| 1111 | 10 | 1110 | 0 |
|---|---|---|---|
| s | a | n | e |

# Prefix Code

- In figure the encoding is represents as:

a by 0, b by 111, r by 110, c by 1011, d by 1010 and ! by 100.




- The bit string

0111110010110101001111100100

encodes the message

" abracadabra! "

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

# Prefix Code

❖ **Example 7:** Which of these codes are prefix codes?

➢ **i) a: 11, e: 00, t: 10, s:01**

✓ This is a prefix code, since no code is the first part of another.

➢ **ii) a:0, e:1, t: 01, s: 001**

✓ This is not a prefix code, since, for instance, the code for a is the first part of the code for t.

➢ **iii) a: 101, e: 11, t: 001, s: 011, n: 010**

✓ This is a prefix code, since no code is the first part of another.

➢ **iv) a: 010, e: 11, t: 011, s: 1011, n: 1001, i: 10101**

✓ This is a prefix code, since no code is the first part of another.

# Prefix Code

❖ **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;

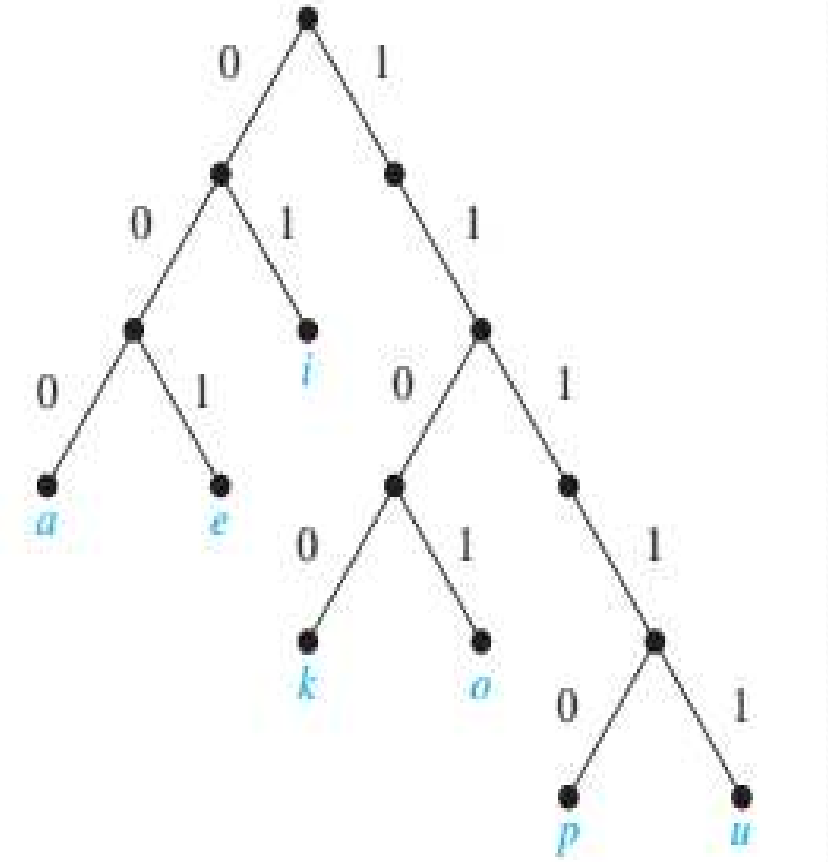
i: 01;

k : 1100;

o : 1101;

p : 11110 and

u : 11111.



# Prefix Code-Homework

❖ **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 w1,w2,w3,….wt be the weights of the terminal vertices. Then the weight W of a binary tree is given by:

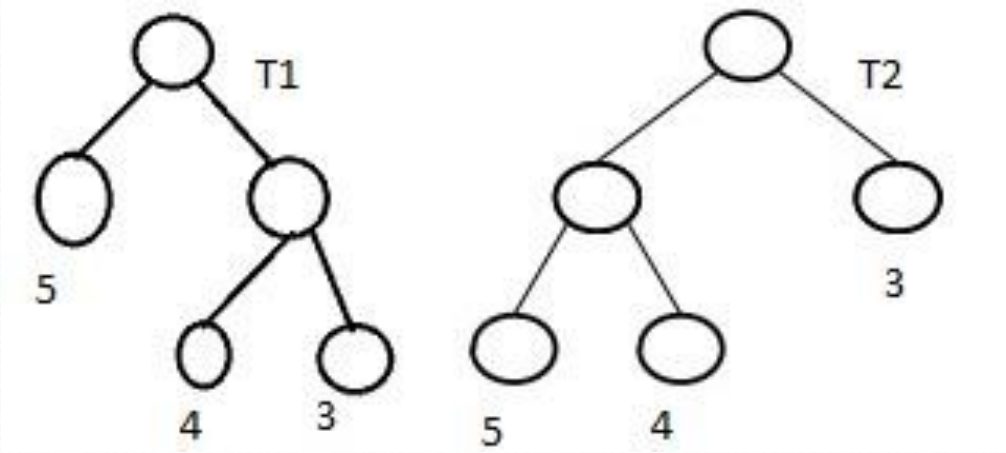
$$W(T) = \sum_{i=1}^{t} wili$$

Where li 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.

# Optimal Tree

❖ **Example 11:** Figure show two fully binary tree T1 and T2, with the weights of the leaves as 3,4 and 5. Show which tree is optimal tree.




**Solution:**

W(T1) = 3*2 + 4*2 + 5*1 = 6 + 8 + 5 = 19

W(T2) = 3*1 + 4*2 + 5*2 = 2 + 8 + 10 = 21

Here W(T1) < W(T2), Thus T1 is optimal tree for weights 3,4, and 5.

# Optimal Prefix Code-Huffman Algorithm

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

- **Huffman Algorithm to find optimal tree::**

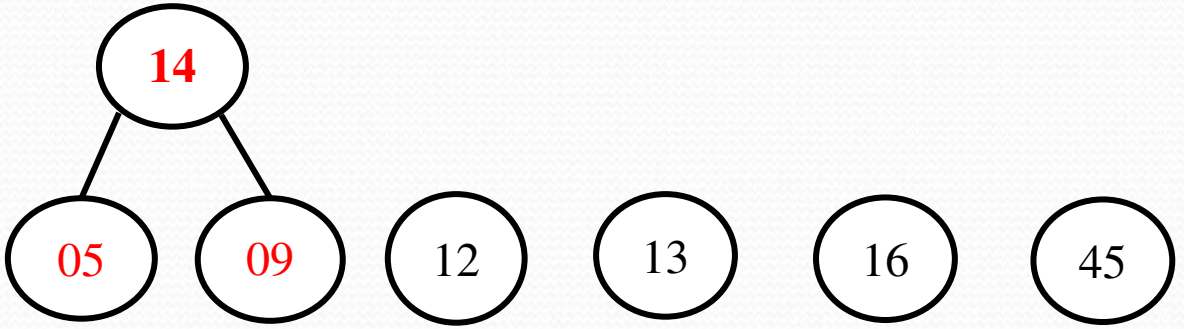
- A Huffman code is a particular type of optimal prefix code that is commonly used for lossless data compression.
- **Given:** A set of symbols and their weights (usually proportional to probabilities).
- **Find:** A prefix-free binary code (a set of codewords) with minimum expected codeword length (equivalently, a tree with minimum weighted path length from the root).
- Let W1,W2,W3,….,Wt 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 W1 and W2. Replace two leaves and their parent by the leaf. Assign weight W1 + W2 to this new leaf.
- ✓ **Step 3:** Repeat step 2 for the weights W1,W2,W3,….,Wt until no weight remains.
- ✓ **Step 4:** The tree obtained in this method is an optimal tree for given weights and stop the procedure.

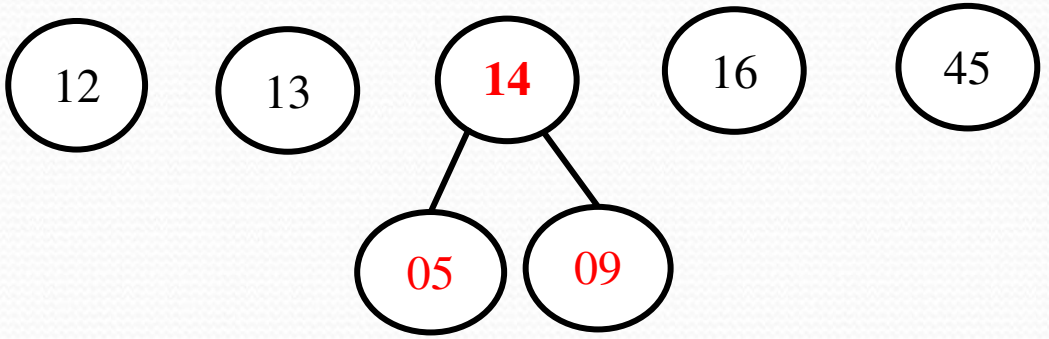
# Optimal Prefix Code-Huffman Algorithm

❖ **Example 12:** Construct Huffman code for the data: 45, 13, 12, 16, 9, 5.

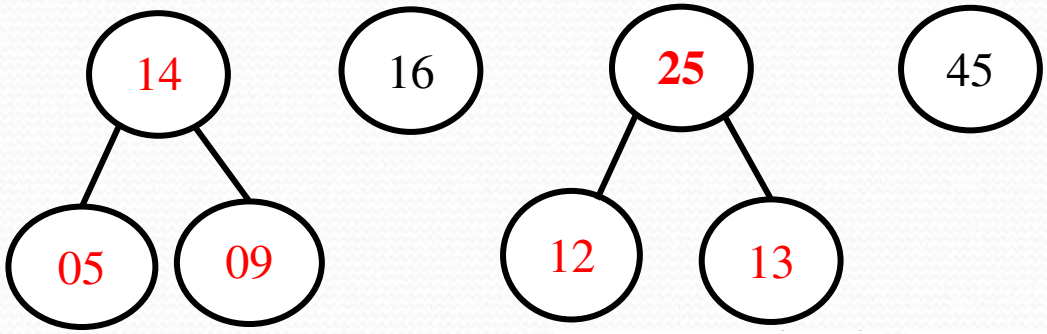
**Step 1:**



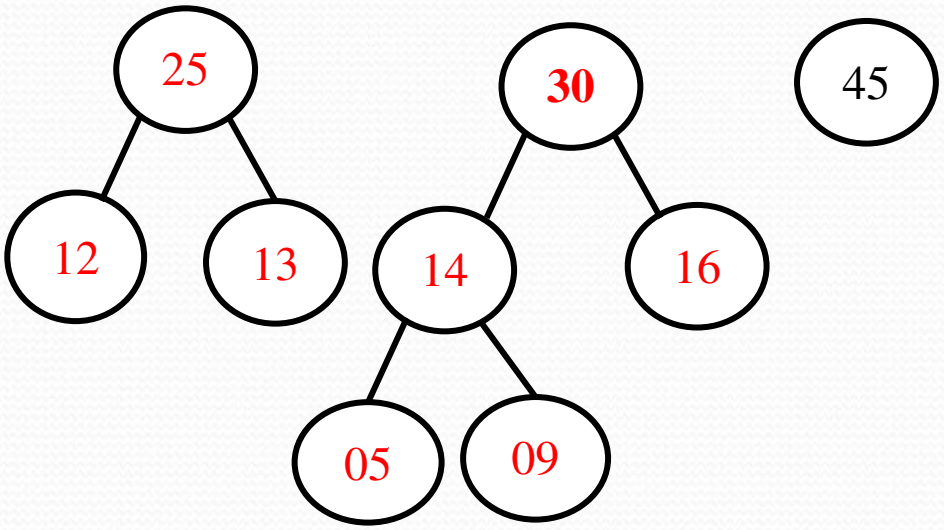

**Step 2:**



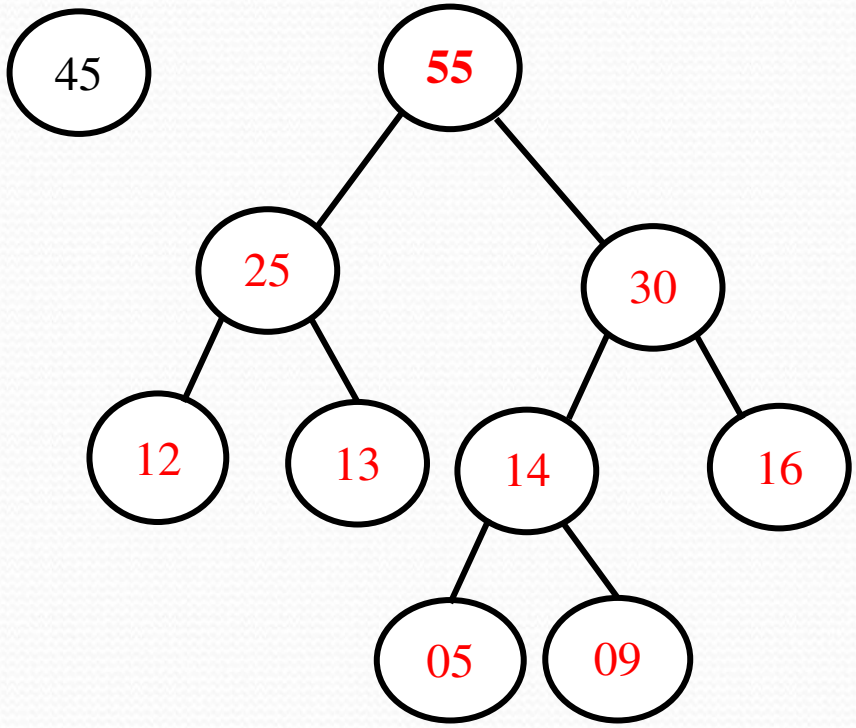

**Step 3:**



# Optimal Prefix Code-Huffman Algorithm



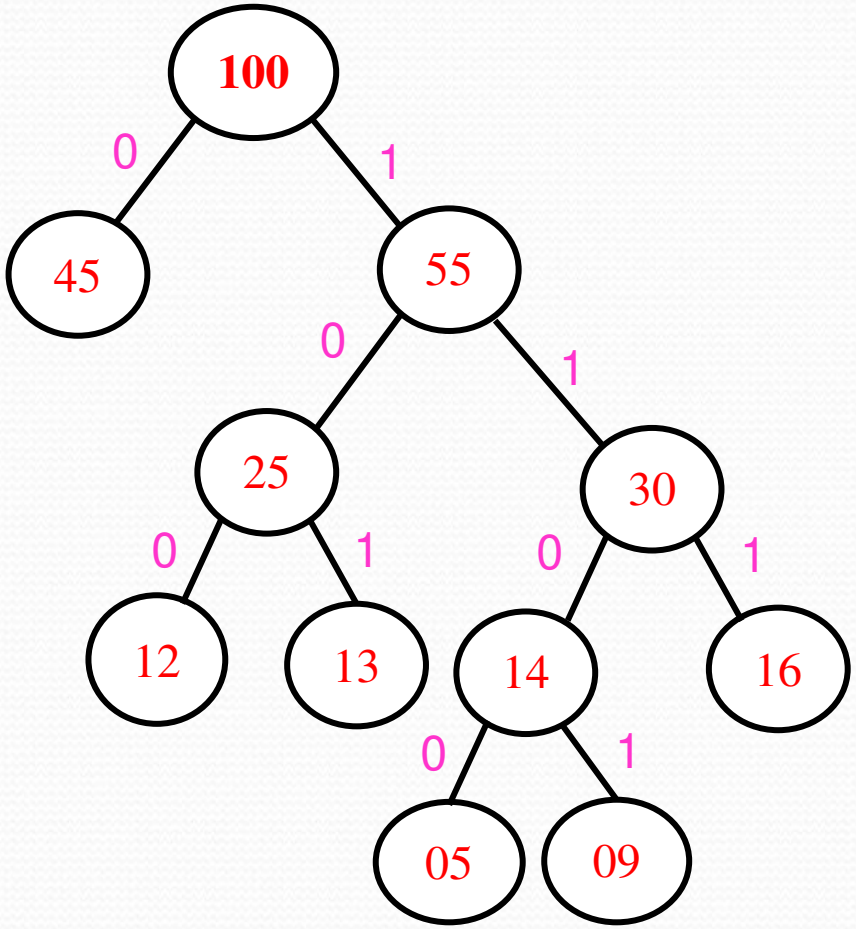

**Step 4:**




**Step 5:**

# Optimal Prefix Code-Huffman Algorithm

**Step 6:**




05=1100

09=1101

12=100

13=101

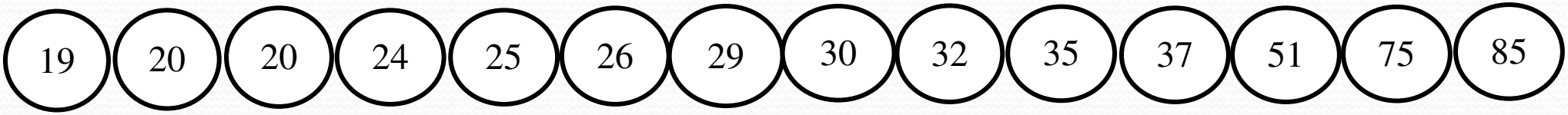
16=111

45=0

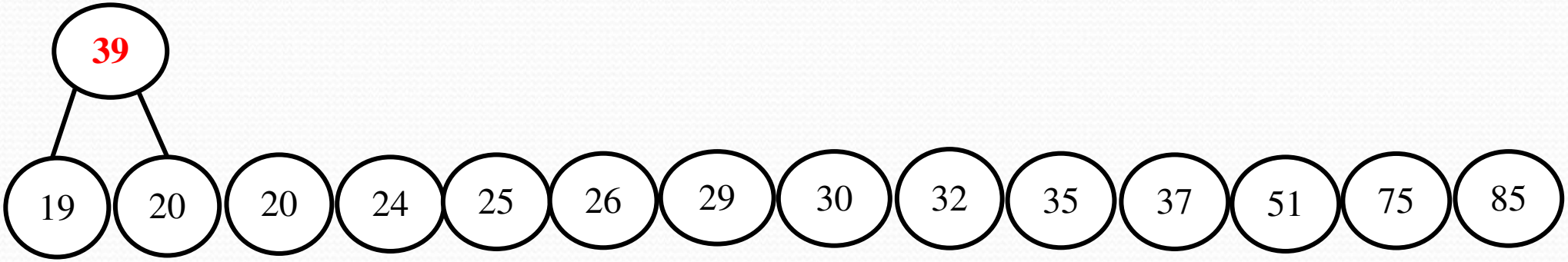
# Optimal Prefix Code-Huffman Algorithm

❖ **Example 13:** 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.

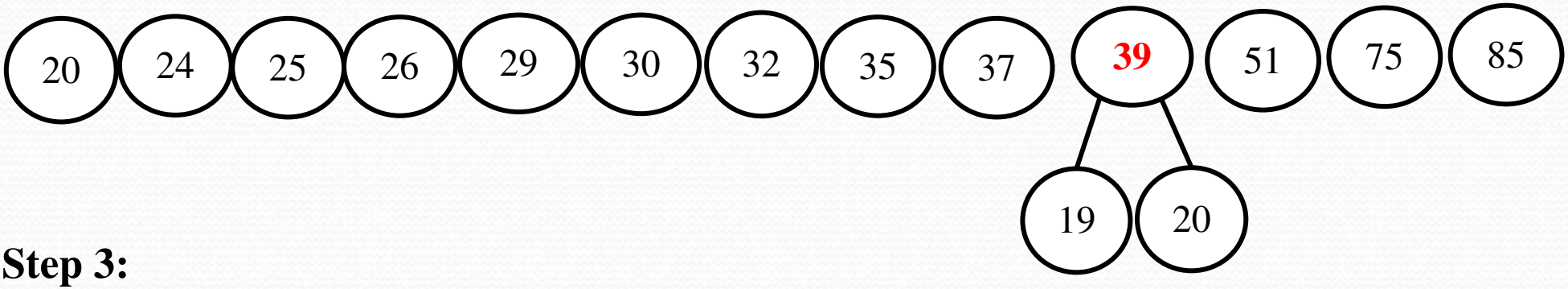
**Solution: Step 1:**



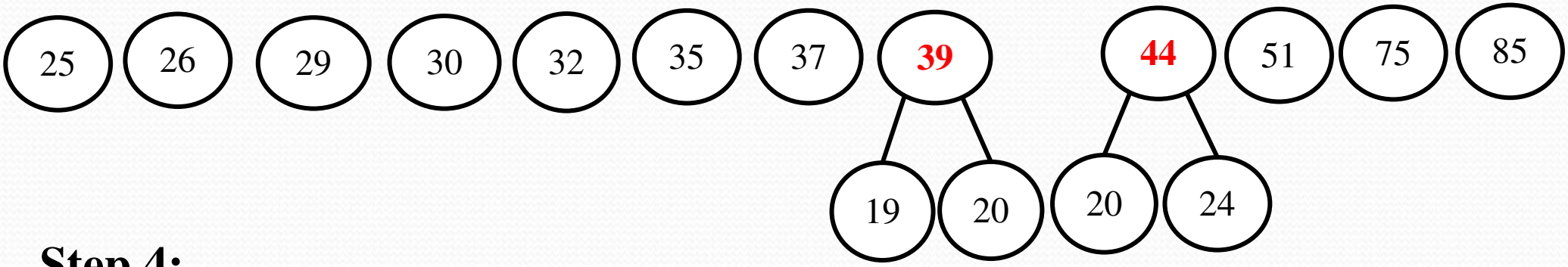

**Step 2:**



# Optimal Prefix Code-Huffman Algorithm

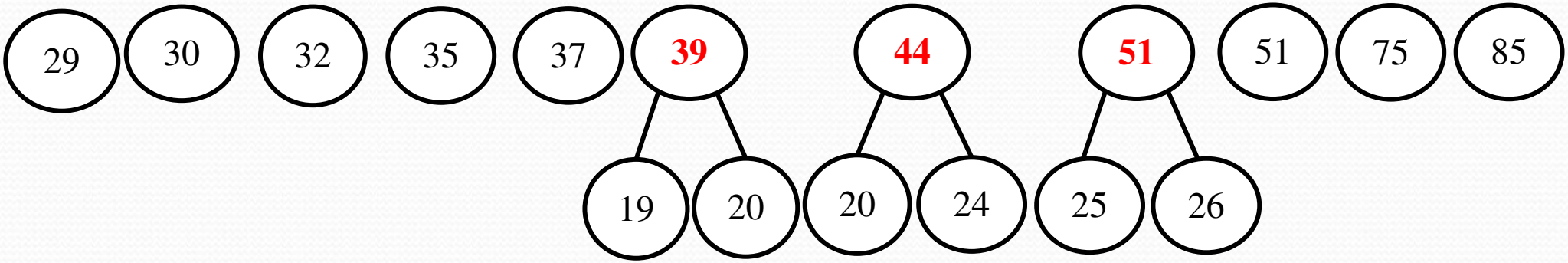



**Step 3:**

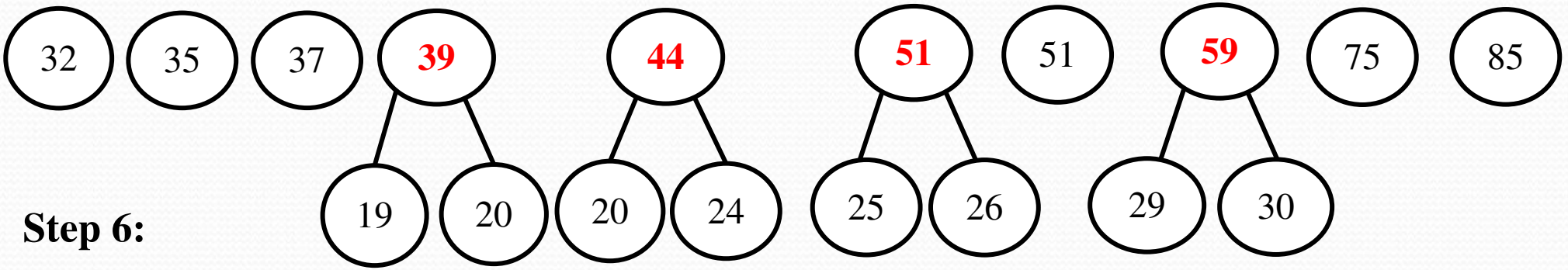



**Step 4:**

# Optimal Prefix Code-Huffman Algorithm

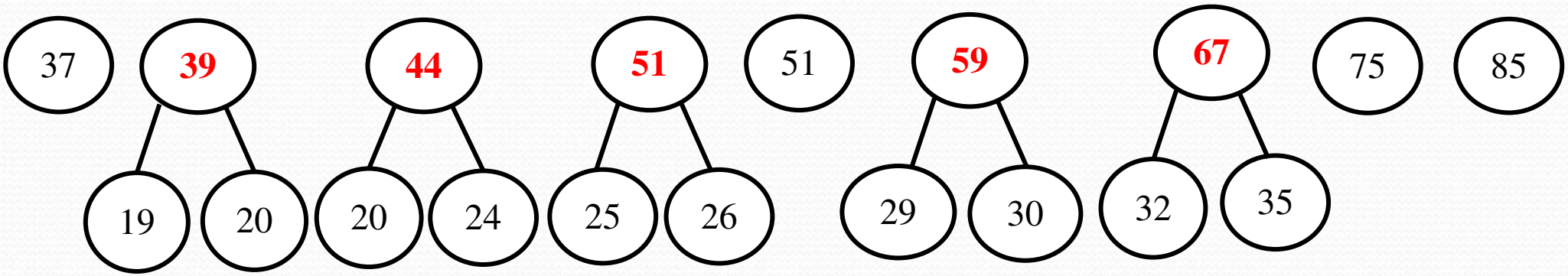



**Step 5:**

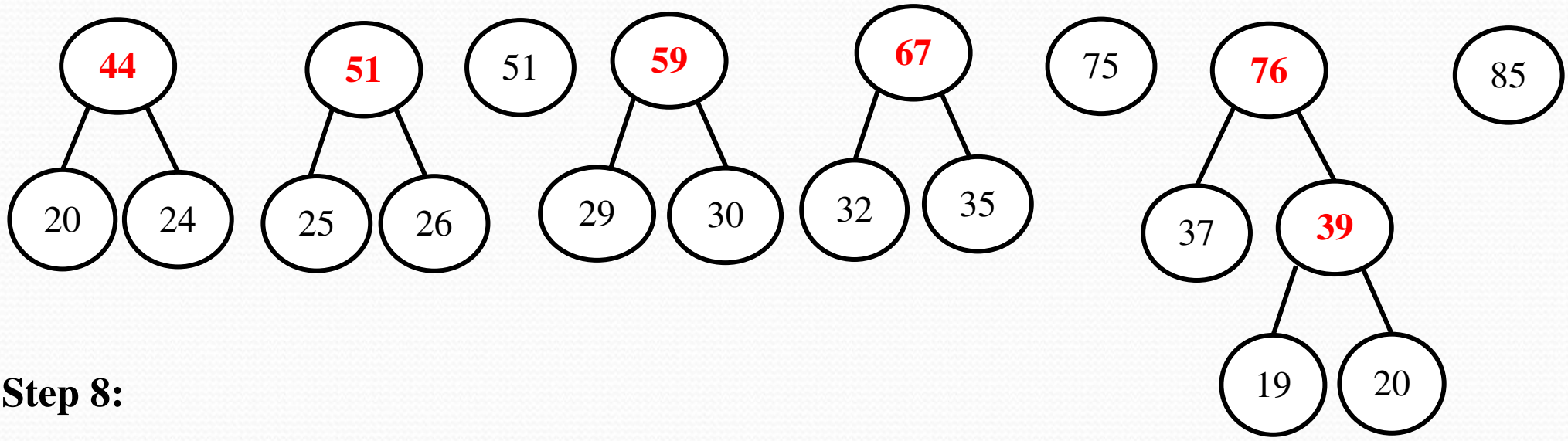



**Step 6:**

# Optimal Prefix Code-Huffman Algorithm

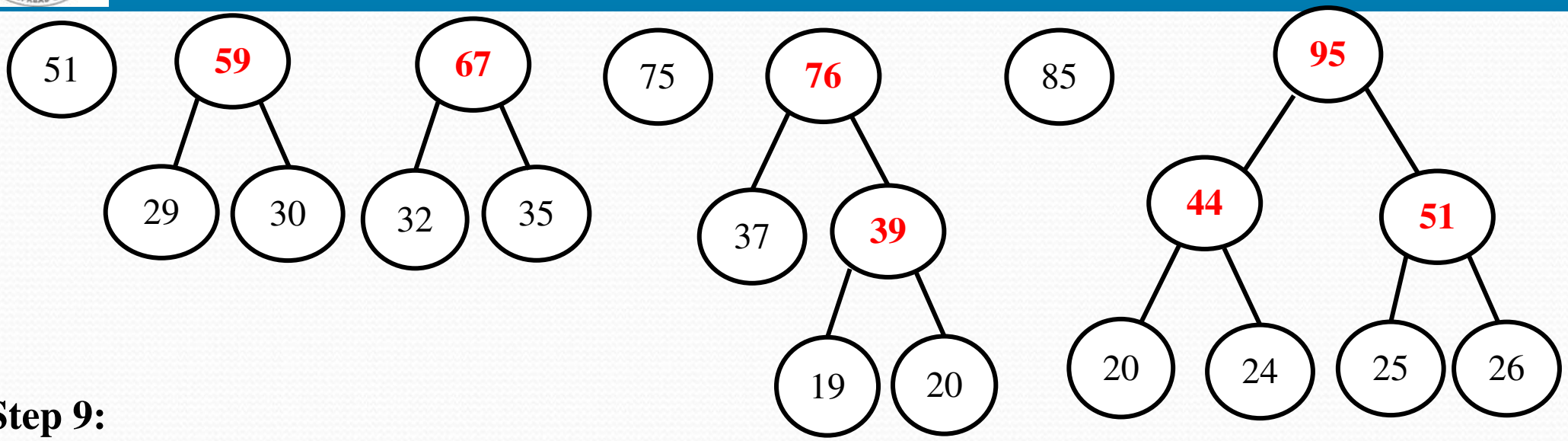



**Step 7:**

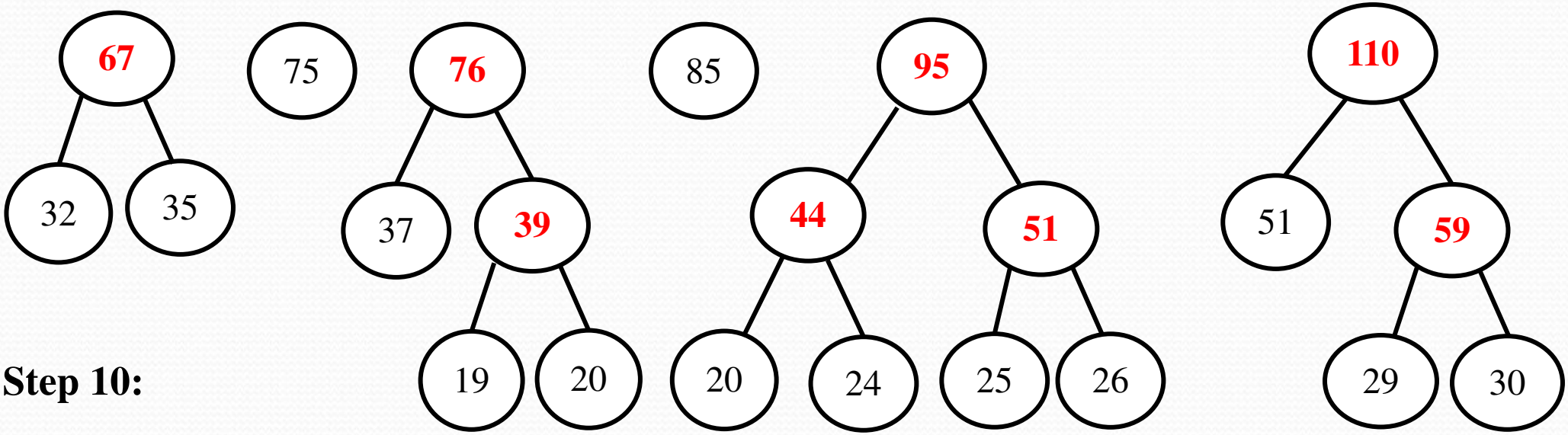



**Step 8:**

# Optimal Prefix Code-Huffman Algorithm

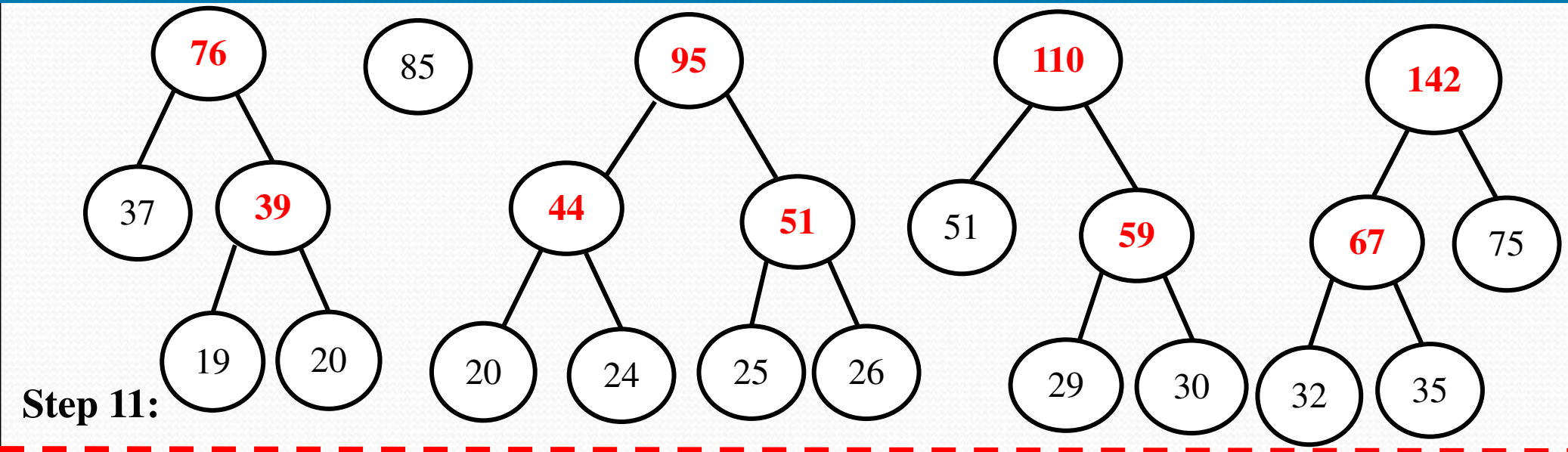


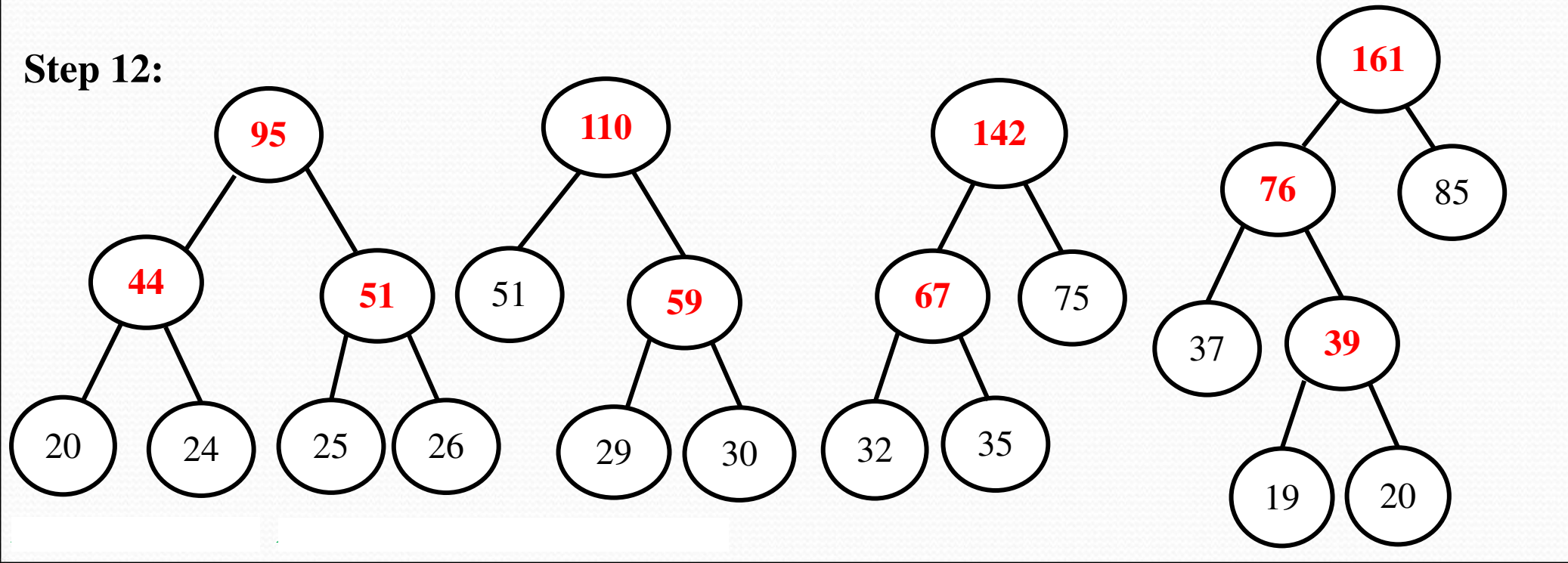

**Step 9:**




**Step 10:**

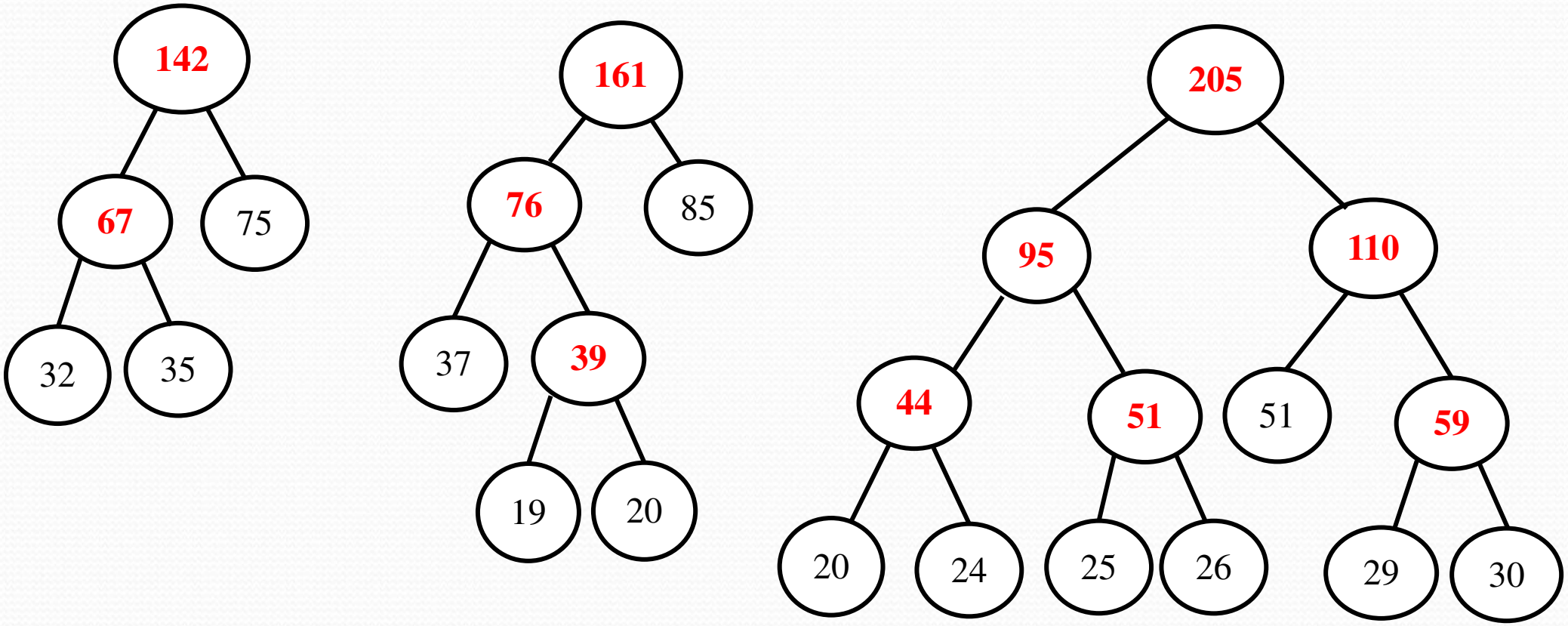
# Optimal Prefix Code-Huffman Algorithm



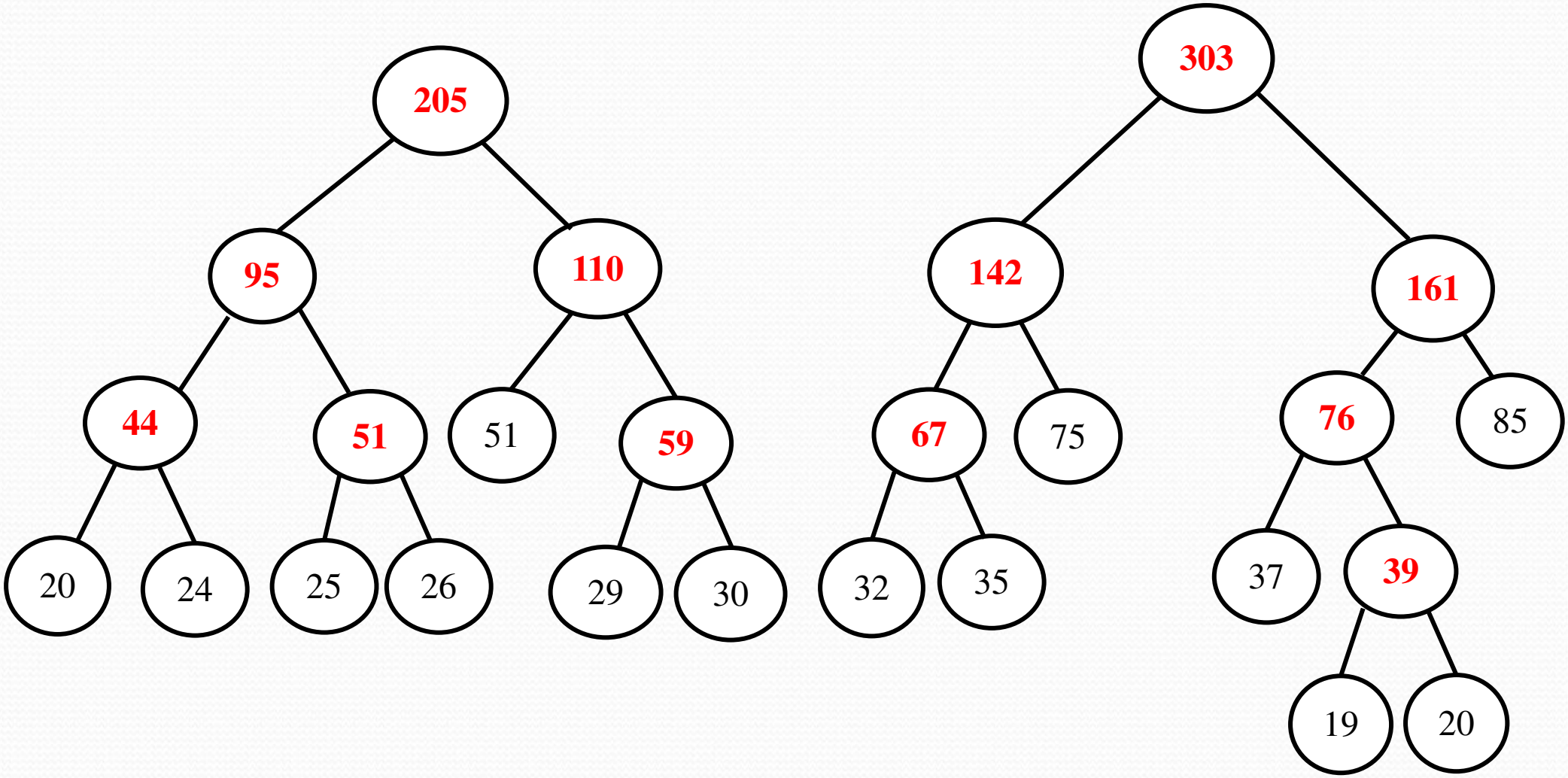
# Optimal Prefix Code-Huffman Algorithm

**Step 13:**



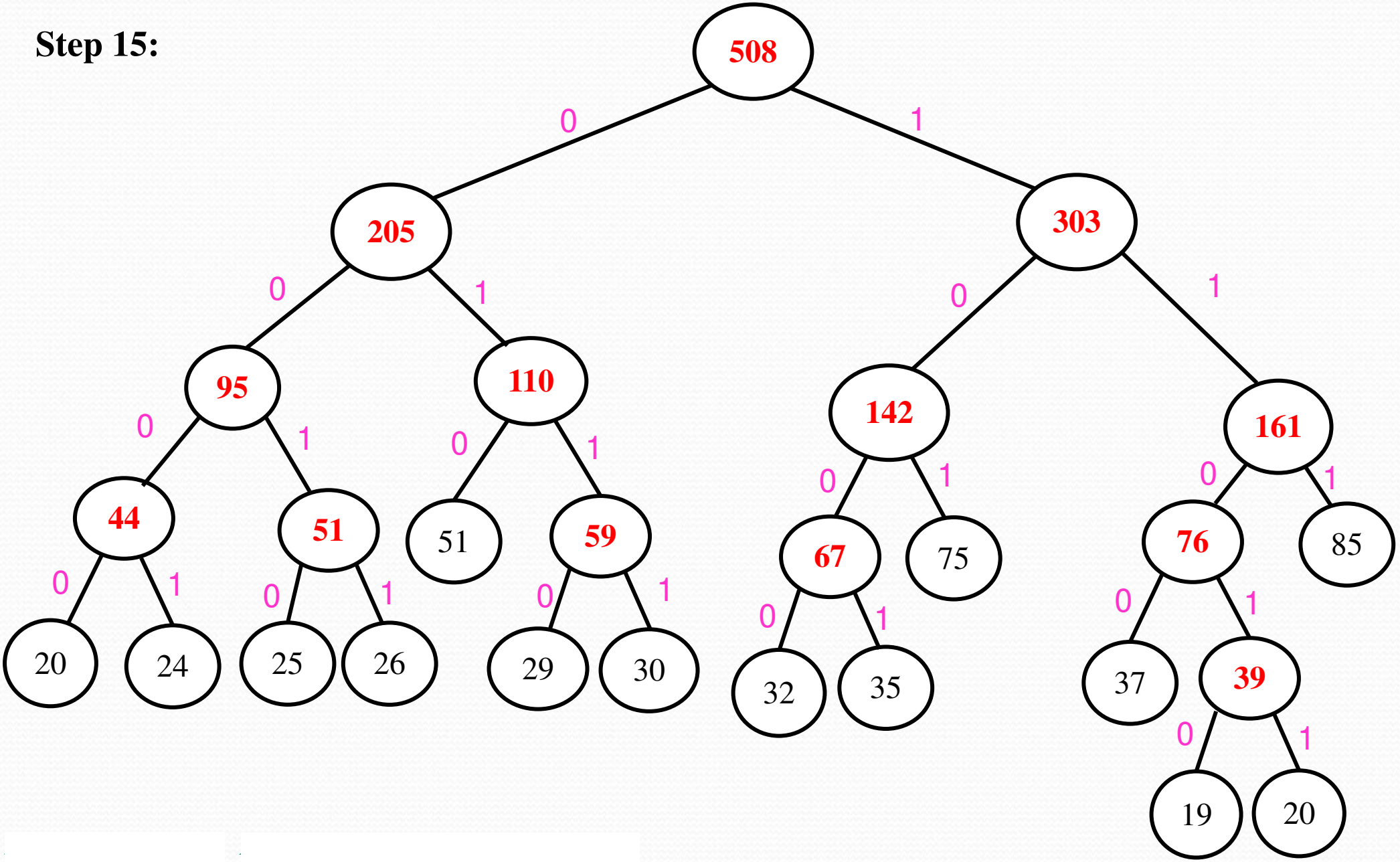
# Optimal Prefix Code-Huffman Algorithm

**Step 14:**



# Optimal Prefix Code-Huffman Algorithm

**Step 15:**



# Optimal Prefix Code-Huffman Algorithm

❖ **Example 14:** 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**

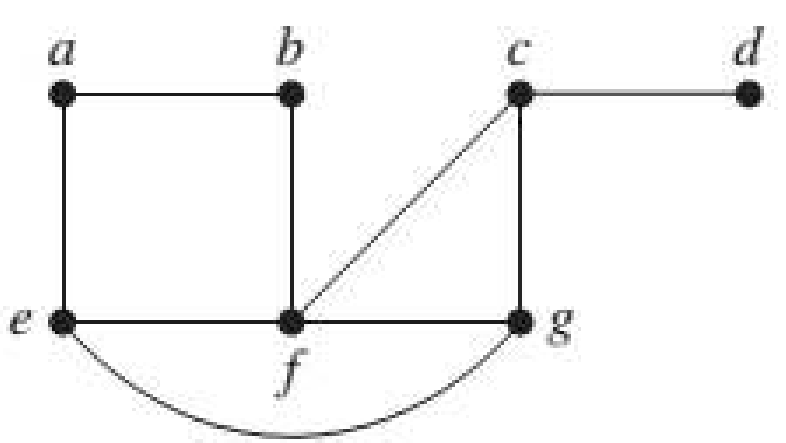
# Spanning Tree

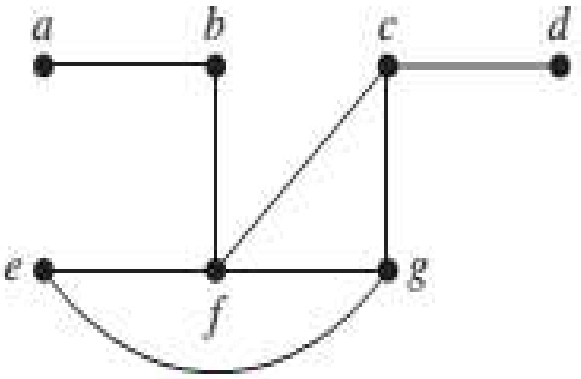
- Let G be a simple graph. A spanning tree of G is a subgraph of G that is a tree containing every vertex of G.
- In general, a graph may have several spanning trees, but a graph that is not connected will not contain a spanning tree
- A simple graph with a spanning tree must be connected, because there is a path in the spanning tree between any two vertices.
- The converse is also true; that is, every connected simple graph has a spanning tree.
- **Theorem:** A simple graph is connected if and only if it has a spanning tree.

# Spanning Tree

❖ **Example 17:** Find a spanning tree of the simple graph G shown in Figure.

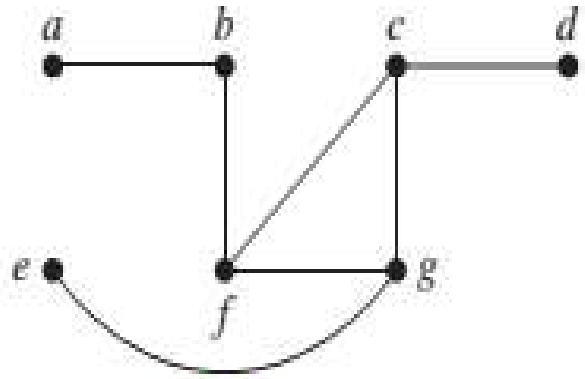
**Solution: Fig c below is spanning tree T.**



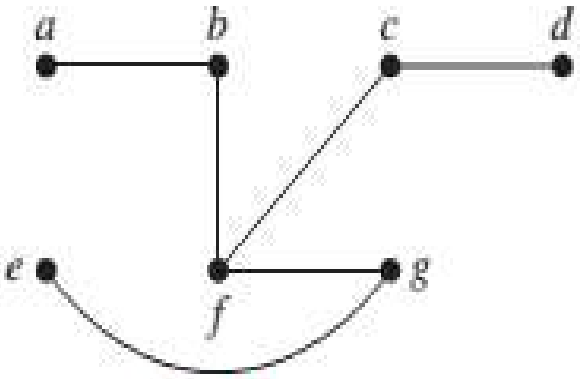

Edge removed: $\{a, e\}$

(a)




$\{e, f\}$

(b)



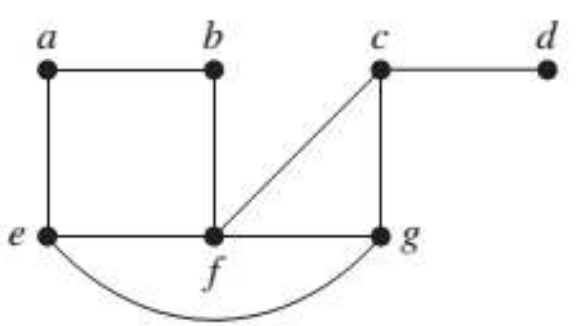

$\{c, g\}$

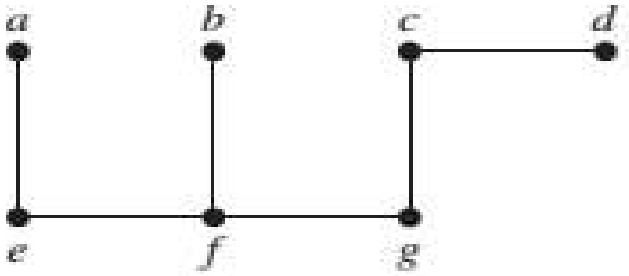
(c)

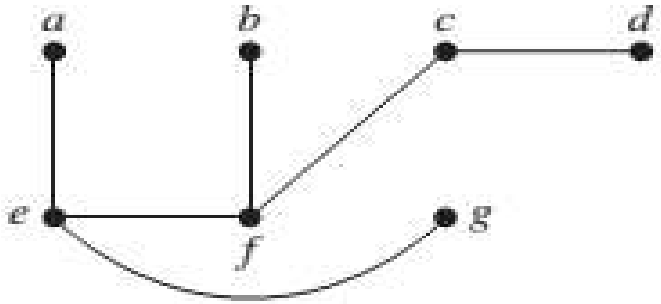
# Spanning Tree

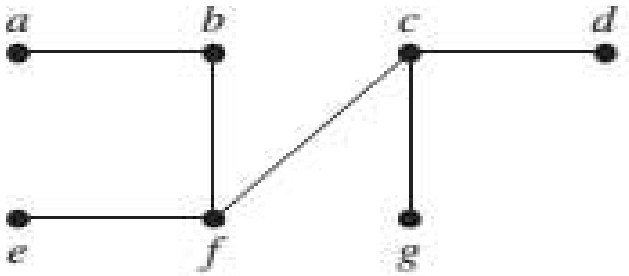
❖ **Example 17:** Find a spanning tree of the simple graph G shown in Figure.

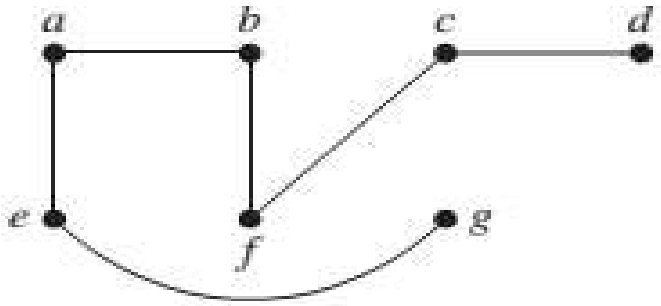
**Solution:**



# Spanning Tree

❖ **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.

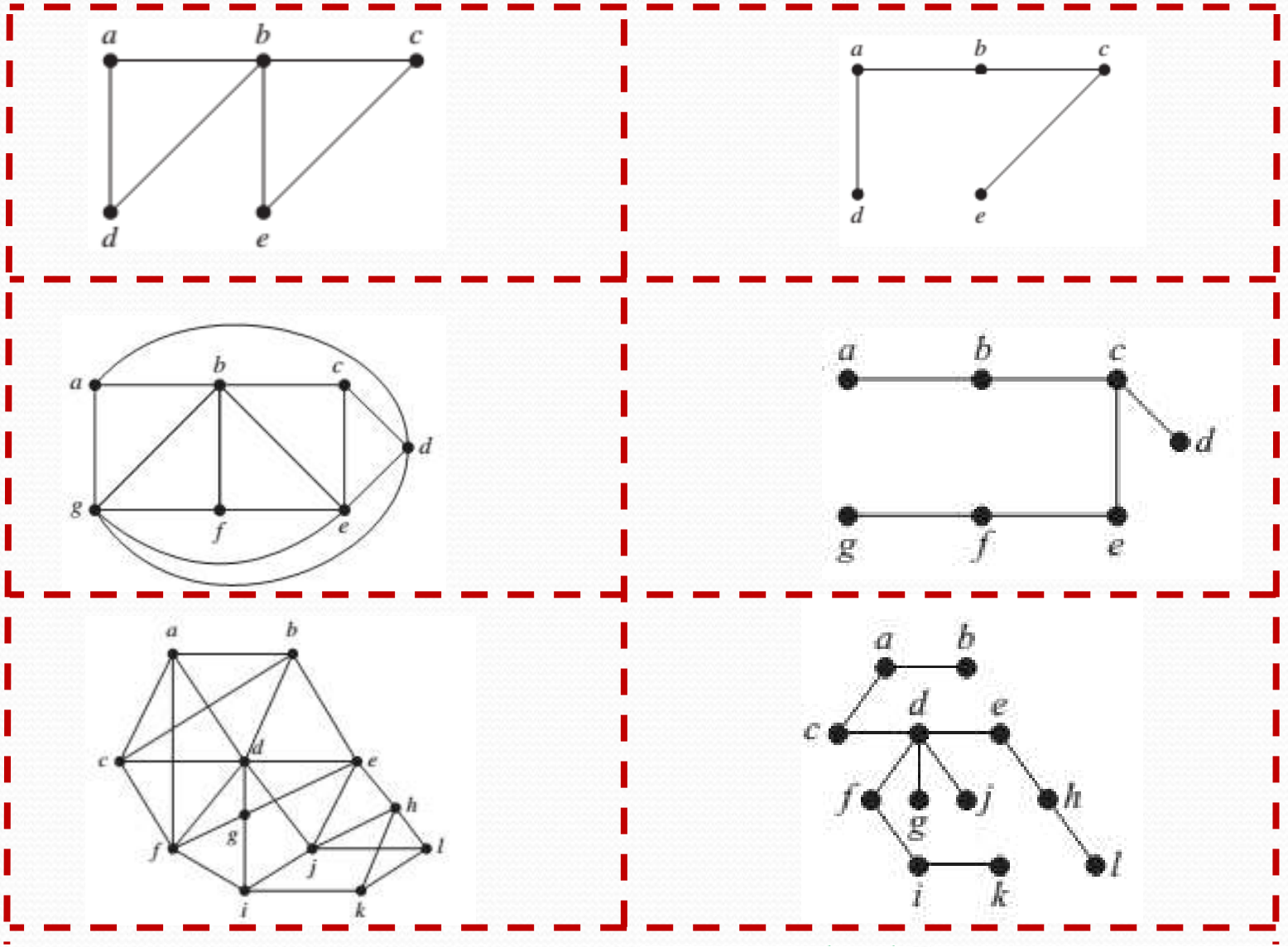
Hence Graph has : e – v + 1 edges

❖ **Example 19:** Find a spanning tree for each of these graphs.

a) K5 b)K4,4 c) K1,6 d)Q3 e) C5 f) W5

# Spanning Tree

❖ **Example 20:** Find a spanning tree for each simple graph G shown in Fig:



# 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
  - Krushal's Algorithm

# 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 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).

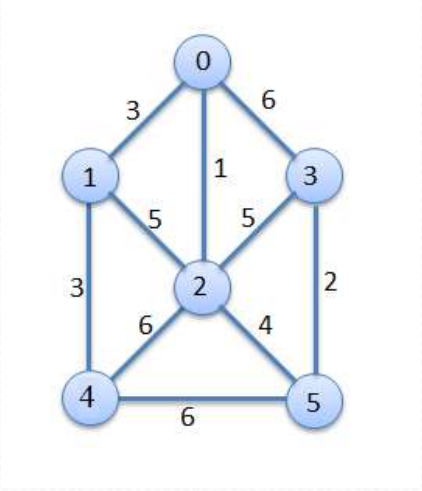
# Prim's Algorithm

- Following are details step of implementation of Prim's algorithm:
  - Let G(V,E) be connected weighted graph
  - **Step 1:** Select any vertex V0 in graph. Set T={ V0, ∅}
  - **Step 2:** Find edge ei= (V0, Vi) in E such that its one vertex is V0 ∈ T and its weight is minimum. Therefore New Set T={{ V0, V1},{ei}}
  - **Step 3:** Choose next edge ek = (Vk, Vj) in such a way that it's one vertex Vk ∈ T and other vertex Vj ∉ T and weight of ek is as small as possible. Again join vetex Vj and edge ek to T.
  - **Step 4**: Repeat step 3 until T contain all vertices of graph G. The result wil be MST of graph G.

# Prim's Algorithm

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

**Solution:**




❖ **Step 1:** Select Vertex 0 as a starting vertex. Therefore T = {{0},∅}


❖ **Step 2:** Vertex 0 is adjacent to vertex 1, 2 and 3.

Edge (0, 1) = 3, edge (0, 2) = 1 and edge (0, 3) = 6.

Select edge with minimum weight.

i.e. say edge e1 = (0,2) = 1 Therefore T = {{0, 2}, e1}.

# Prim's Algorithm

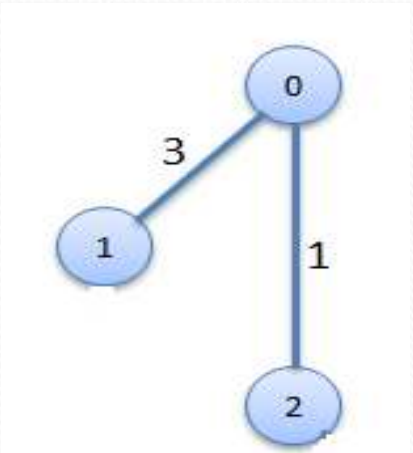
- **Step 3:** Now Vertex 0 is adjacent to vertex 1, and 3. Edge (0, 1) = 3 and edge (0, 3) = 6.

  Vertex 2 is adjacent to 1, 3, 4, and 5

  Edge (2, 1) = 5, edge (2, 3) = 5, edge (2, 4) = 6 and edge (2, 5 ) = 4

  Select edge with minimum weight. i.e. say edge e2 = (0, 1) = 3.

  Therefore T = {{0, 2, 1}, e1, e2}.




- **Step 4:** Now Vertex 0 is adjacent to vertex 3. Edge (0, 3) = 6.

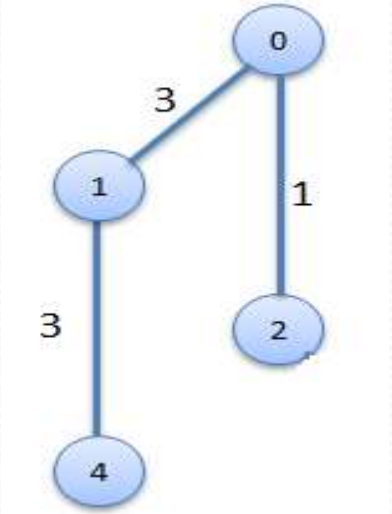
  Vertex 2 is adjacent to 3, 4, and 5

  Edge (2, 3) = 5 ,edge (2, 4) = 6 and edge (2, 5 ) = 4

  Vertex 1 is adjacent to vertex 4. Edge (1, 4) = 3.

  Select edge with minimum weight. i.e. say edge e3 = (1, 4) = 3.

  Therefore T = {{0, 2, 1, 4}, e1, e2, e3}.



# Prim's Algorithm

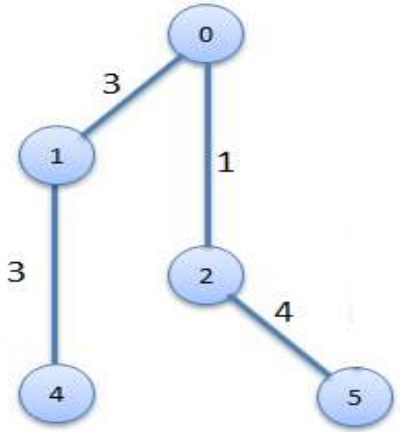
- **Step 5:** Now Vertex 0 is adjacent to vertex 3 Edge (0, 3) = 6.

  Vertex 2 is adjacent to 3, and 5 Edge (2, 3) = 5 and edge (2, 5 ) = 4.

  Vertex 4 is adjacent to 5 Edge (4, 5 ) = 6.

  Select edge with minimum weight. i.e. say edge e4 = (2, 5 ) = 4

  Therefore T = {{0, 2, 1, 4, 5}, e1, e2, e3, e4}.



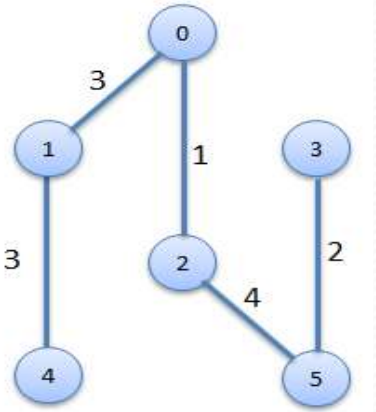

- **Step 6:** Now Vertex 0 is adjacent to vertex 3. Edge (0, 3) = 6.

  Vertex 2 is adjacent to 3. Edge (2, 3) = 5

  Vertex 5 is adjacent to 3. Edge (5, 3 ) = 2

  Select edge with minimum weight. i.e. say edge e5 = (5, 3 ) = 2

  Therefore T = {{0, 2, 1, 4, 5, 3}, e1, e2, e3, e4, e5}.




- **Step 7:** All vertex are include in spanning tree, and graph obtained is the minimum spanning tree. **Minimum Cost = 1 + 2 + 3 + 3 + 4 = 13.**

# Krushal'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.

# Krushal's Algorithm

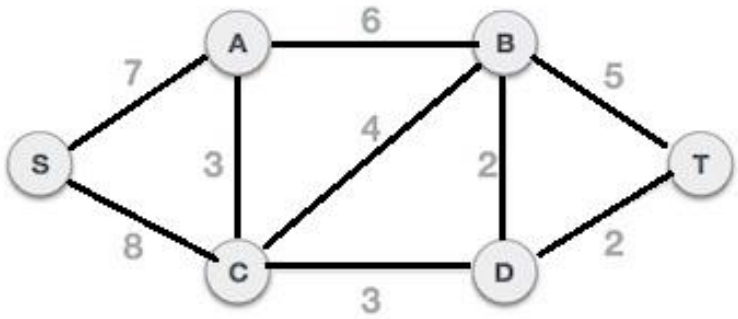
- Following are details step of implementation of Krushal's algorithm.
- Let G(V,E) be weighted connected graph.
- **Step 1:** Pick up the $e_i$ of the Graph G such that its weights W(e,i) is minimum.(If there are more edges of the minimum weight then select all those edges which do not form circuit)
- **Step 2:** If edges e1,e2,e3,…..en have been chosen then pick an edge $e_{n+1}$ such that:
  i. $e_{n+1} \neq e_i$ for i=1,2,3,…,n
  ii. The edges $e_1,e_2,e_3,\ldots..e_n\ e_{n+1}$ do not form circuit.
  iii. $W(e_{n+1})$ is as small as possible subject to condition (ii).
- **Step 3:** Stop procedure when step two cannot be implemented further.

# Krushal's Algorithm

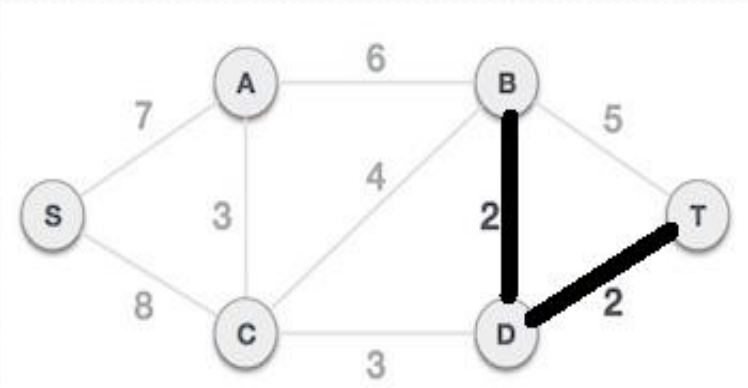
❖ **Example 22**: Using Krushal algorithm Find MST for given graph.

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

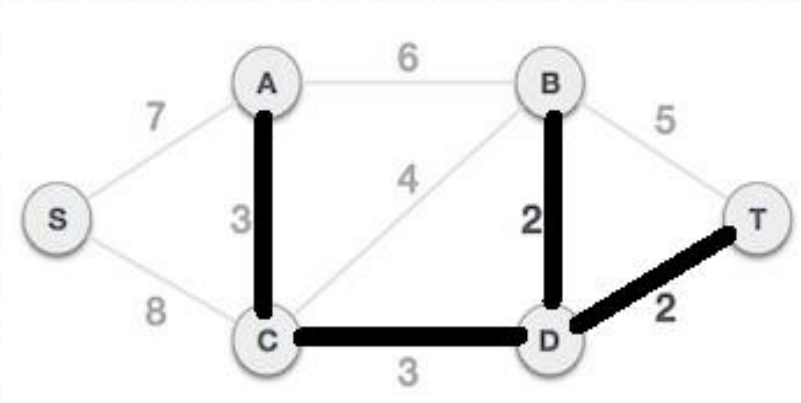
| BD | DT | AC | CD | CB | BT | AB | SA | SC |
|---|---|---|---|---|---|---|---|---|
| 2 | 2 | 3 | 3 | 4 | 5 | 6 | 7 | 8 |




**Step 1:** Now we start adding edges 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.




**Step 2:** Next cost is 3, and associated edges are A,C and C,D. We add them again.

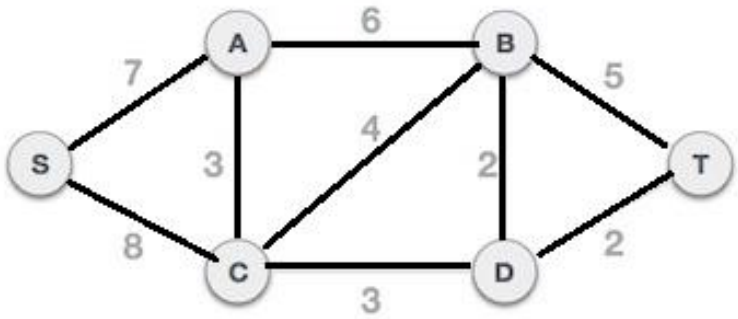


# Krushal's Algorithm

❖ **Example 22**: Using Krushal algorithm Find MST for given graph.

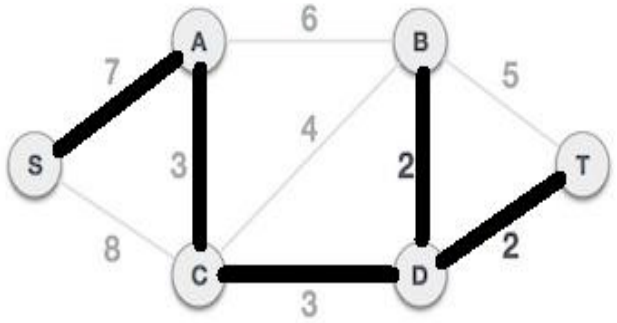
**Solution:** Arrange all edges in their increasing order of weight.

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




**Step 3:** 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.

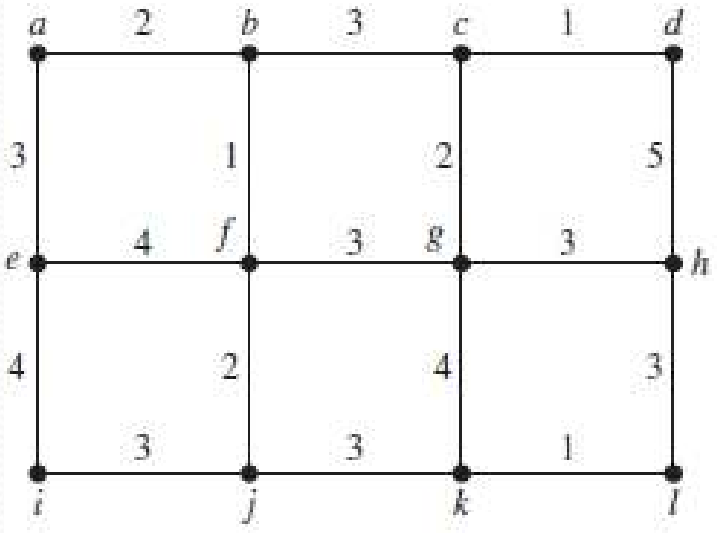
**Step 4:** Now we are left with only one node to be added. Between the two least cost edges available 7 and 8, we shall add the edge with cost 7.
By adding edge S, A we have included all the nodes of the graph and we now have minimum cost spanning tree.

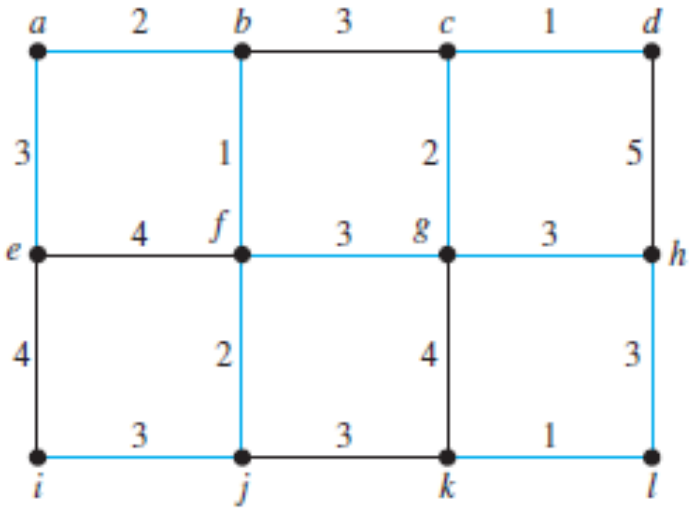



**Minimum Cost Spanning Tree = 2 + 2 + 3 + 3 + 7 = 17.**

# MST Example-Prim's algorithm

❖ Use Prim's algorithm to find a minimum spanning tree in the graph shown in Figure.



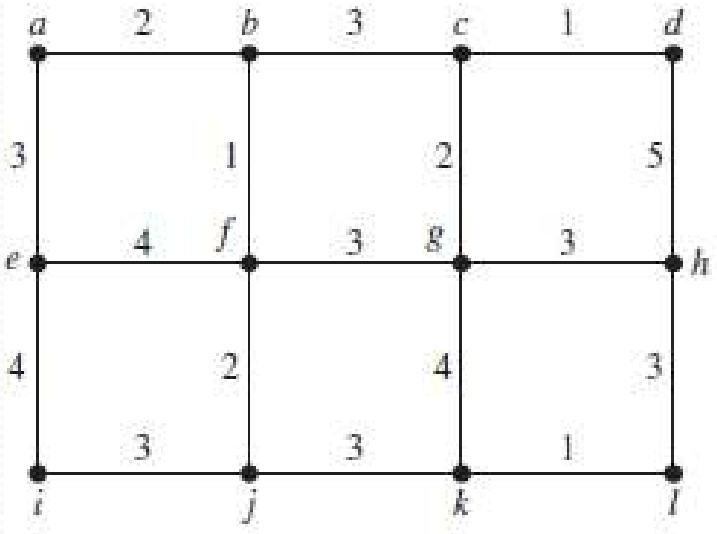



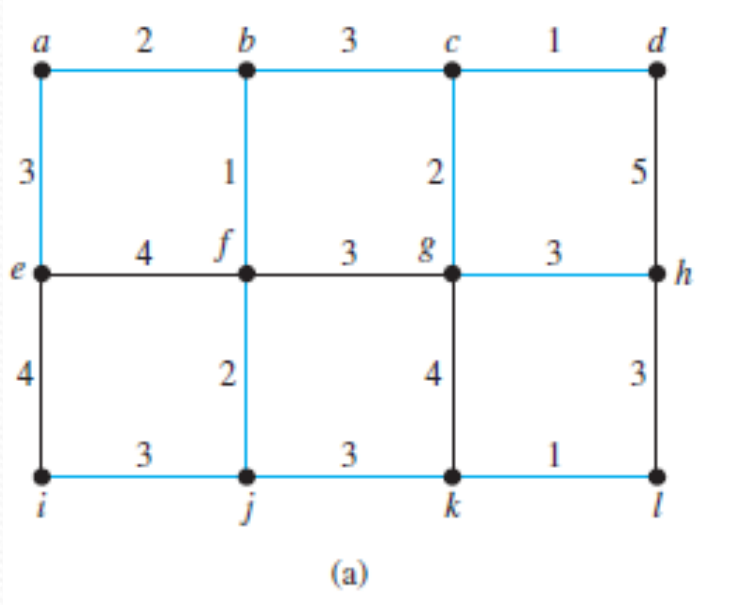

(a)

| Choice | Edge | Weight |
|---|---|---|
| 1 | b, e | 1 |
| 2 | a, b | 2 |
| 3 | f, j | 2 |
| 4 | a, e | 3 |
| 5 | i, j | 3 |
| 6 | f, g | 3 |
| 7 | c, g | 2 |
| 8 | c, d | 1 |
| 9 | g, h | 3 |
| 10 | h, l | 3 |
| 11 | k, l | 1 |
| **Total** | | **24** |

# MST Example- Krushal's algorithm

- Use Krushal's algorithm to find a minimum spanning tree in the graph shown in Figure.




(a)

| Choice | Edge | Weight |
|---|---|---|
| 1 | c, d | 1 |
| 2 | k, l | 1 |
| 3 | b, f | 1 |
| 4 | c, g | 2 |
| 5 | a, b | 2 |
| 6 | f, j | 2 |
| 7 | b, c | 3 |
| 8 | j, k | 3 |
| 9 | g, h | 3 |
| 10 | i, j | 3 |
| 11 | a, e | 3 |
| **Total** | | **24** |

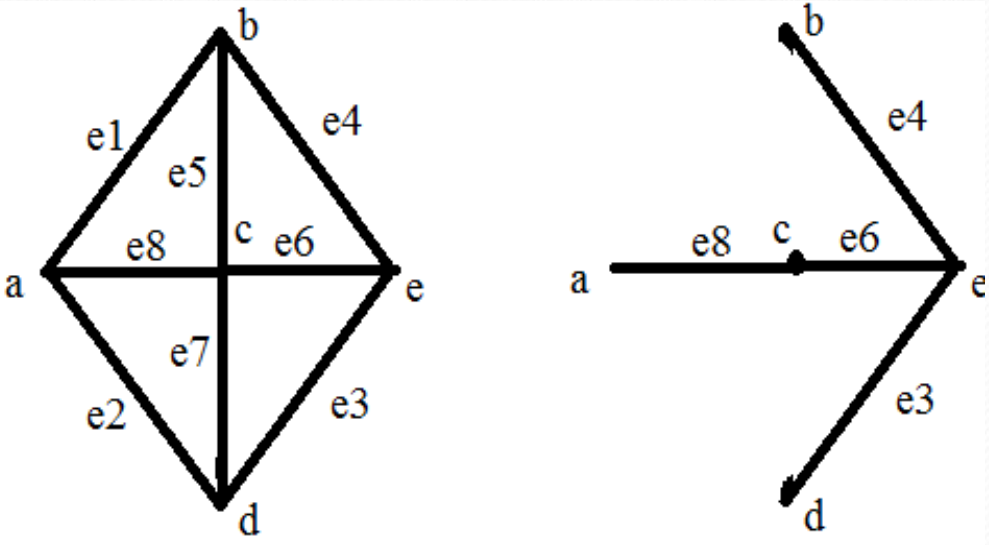
# Fundamental Circuit

- 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.

# Fundamental Circuit

❖ **Example 23:** 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 } |

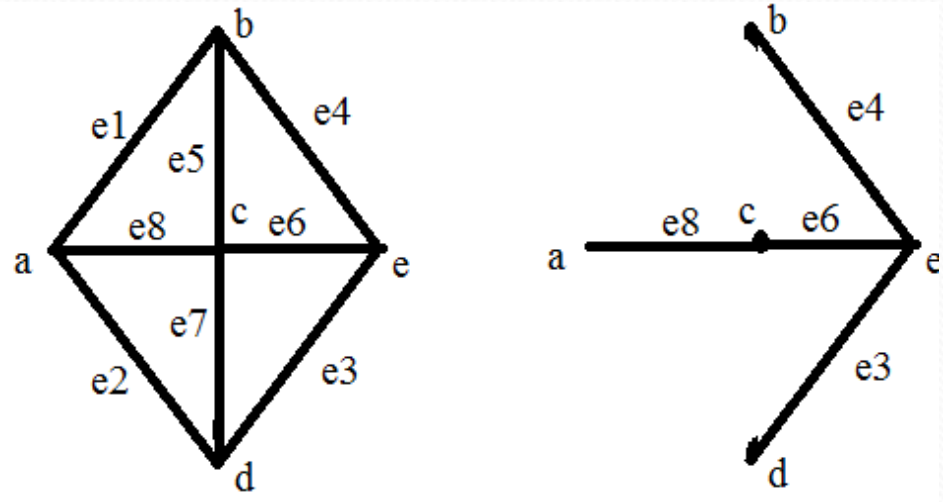
# Fundamental Cut-sets

- 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.

# Fundamental Cut-sets

❖ **Example 24:** Find the fundamental circuit for the graph show in figure below w.r.t the spanning tree T.

Solution:



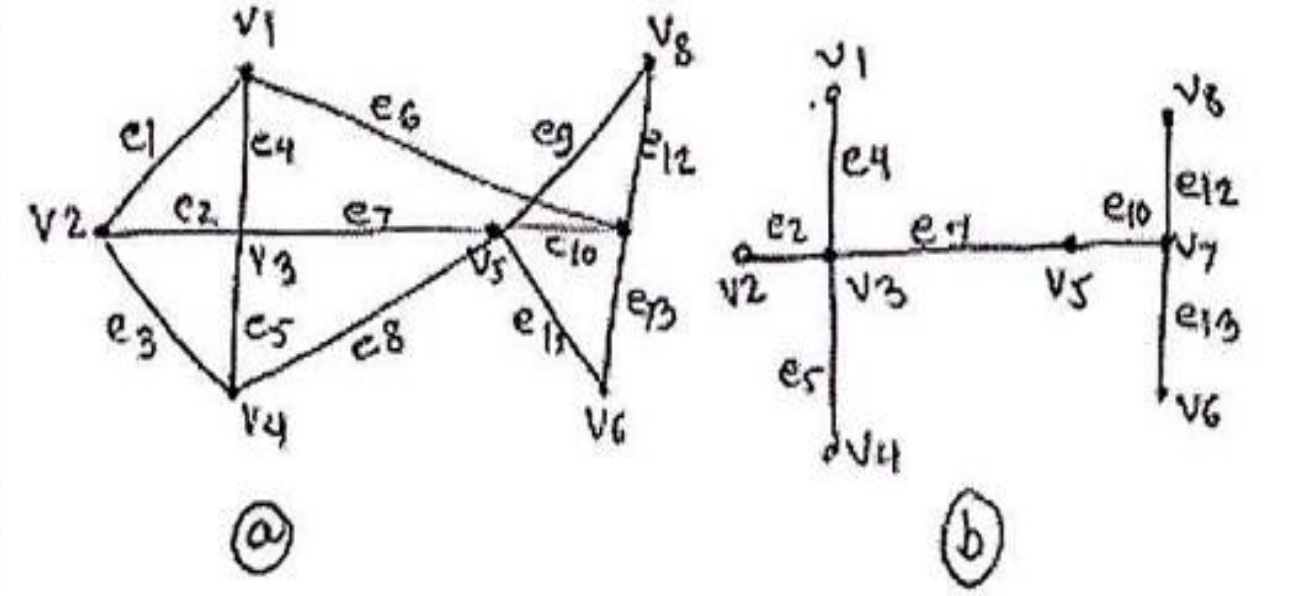

| Branch | Fundamental cutset |
|---|---|
| e4 | { e1, e5, e4 } |
| e3 | { e2, e7, e3 } |
| e6 | { e1, e5, e6, e7, e2 } |
| e8 | {e1, e8, e2 } |

# Fundamental Cut-sets

**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} |