Unit V

Trees

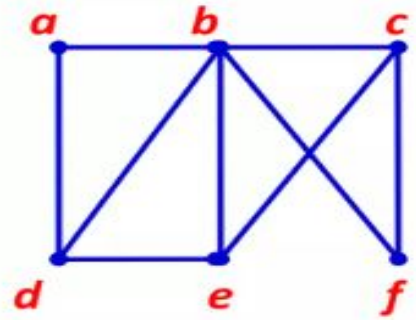
Topics

- 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).

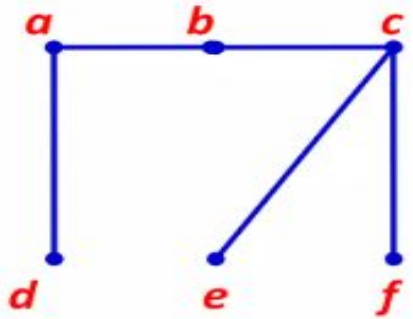
Introduction

- A **tree** is a connected undirected graph with no simple circuit.
- An undirected graph is a tree if and only if there is a unique simple path between any two of its vertices.
- THEOREM: A tree with n vertices has $n - 1$ edges.

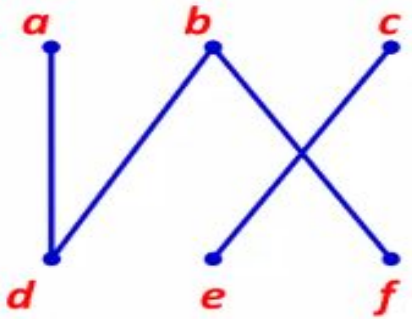
EXAMPLES



NOT A TREE



A TREE

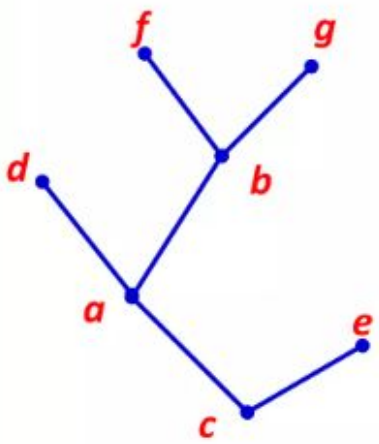


NOT A TREE

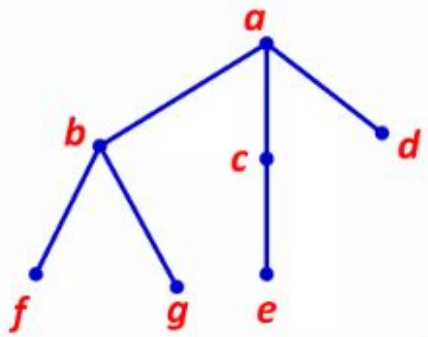
Introduction

- 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.
- Different choice of root produce different rooted tree

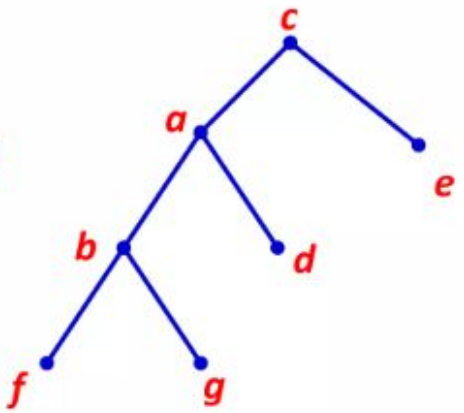
EXAMPLES



A TREE



A TREE with root *a*

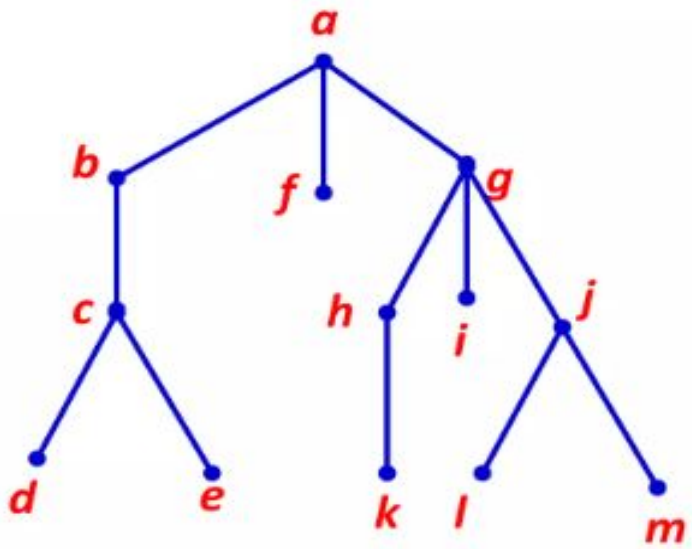


A TREE with root *c*

Properties of trees

- Parent – A vertex other than root is a parent if it has one or more children
  - The parent of *c* is *b*
- Children – If *A* is a vertex with successors *B* and *C*, then *B* and *C* are the children of *A*.
  - The children of *a* is *b*, *f* and *g*
- Siblings – Children with the same parent vertex.
  - *h*, *i* and *j* are siblings
- Level – the length of the unique path from the root to a vertex
  - Vertex *a* is at level 0
  - Vertices *d* and *e* is at level 3

EXAMPLES

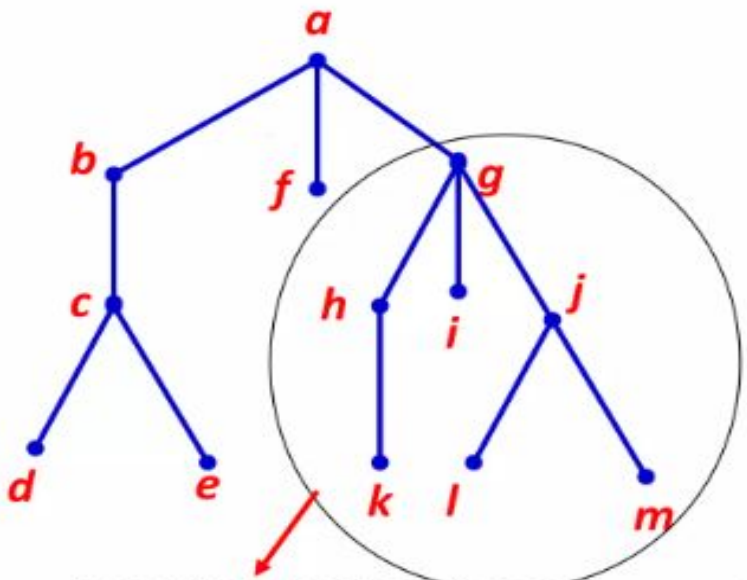


- Height – The maximum level of all the vertices
  - The height of this tree is 3.

Properties of trees

- Ancestor of a vertex (v) – the vertices in the path from the root to this vertex excluding this vertex.
  - The ancestors of *e* are *c*, *b* and *a*
- Descendent of a vertex (v) – vertices that have *v* as ancestor.
  - The descendants of *b* are *c*, *d* and *e*
- Leaf – A vertex with no children
  - The leaves are *d*, *e*, *f*, *i*, *k*, *l* and *m*
- Internal Vertices – vertices that have children
  - The internal vertices are *a*, *b*, *c*, *g*, *h* and *j*

EXAMPLES



A subtree rooted at *g*

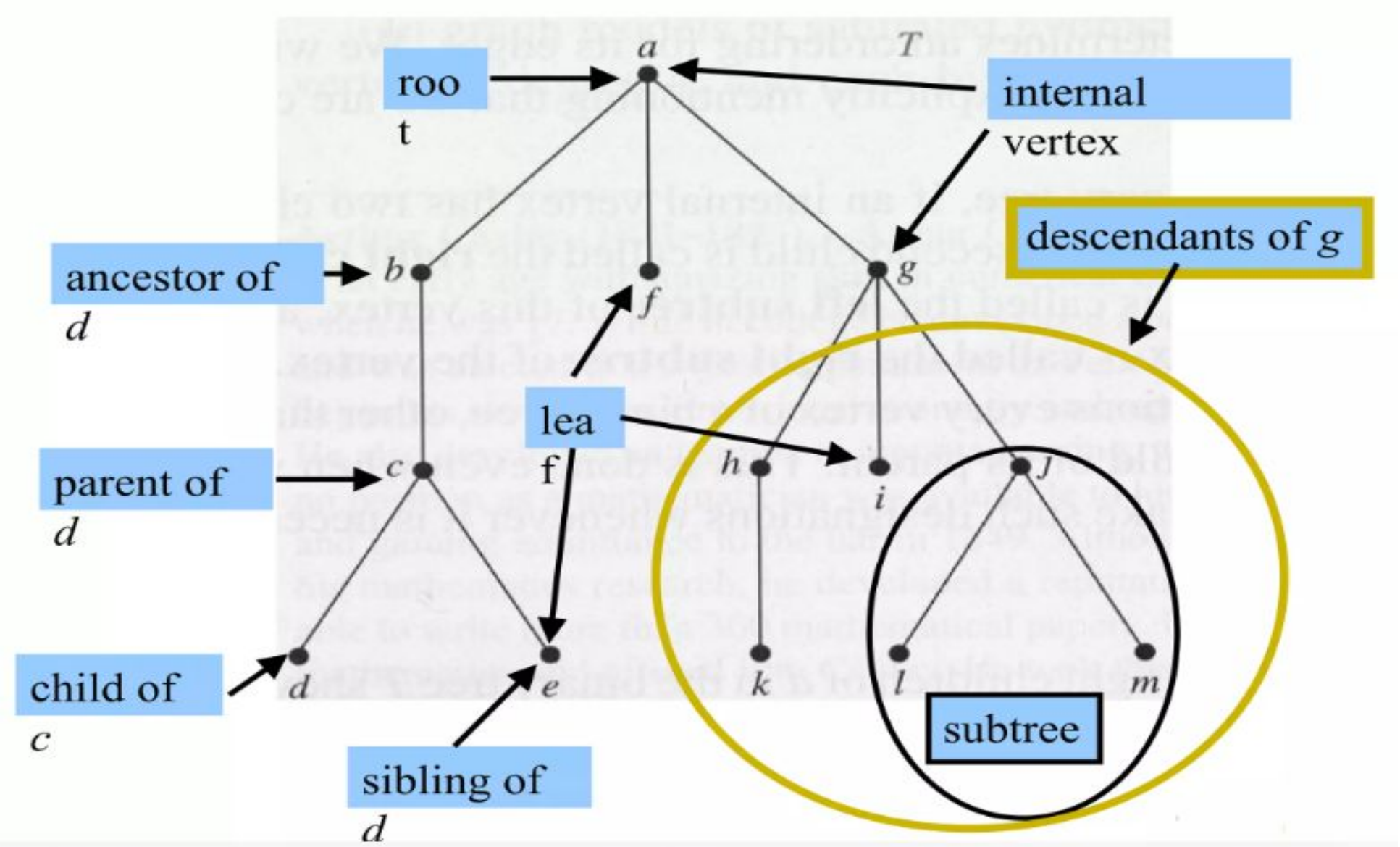
- Subtree – A subgraph of the tree consisting of a root and its descendent and all edges incident to these descendent.

Properties of trees

- Degree of node:Number of children /number of subtrees

- Depth of Node :Total number of edges from root node to a particular node
- Depth of Tree: Total number of edges from root node to a leaf node in the longest path

Properties of trees

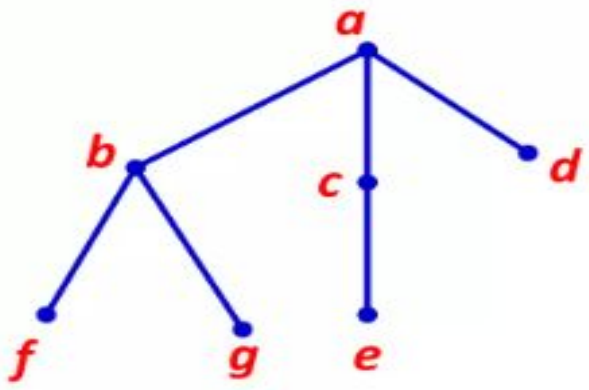


M-ary Tree

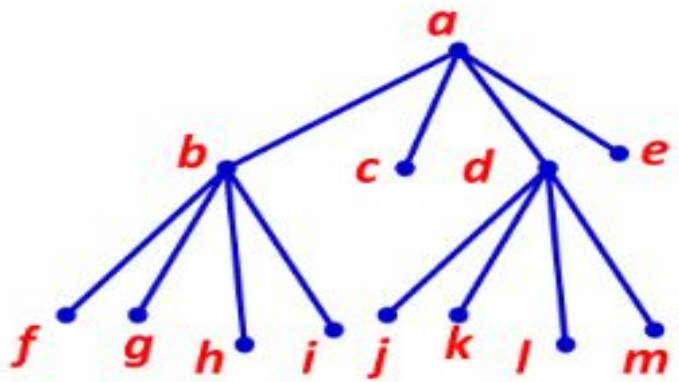
- **THEOREM 1**: A full *m*-ary tree with *i* internal vertices contains $n = mi + 1$ vertices

- A rooted tree is called an *m*-ary tree if every vertex has no more than *m* children.
- The tree is called a full *m*-ary tree if every internal vertex has exactly *m* children.
- A rooted *m*-ary tree is balanced if all leaves are at levels *h* or *h*-1.

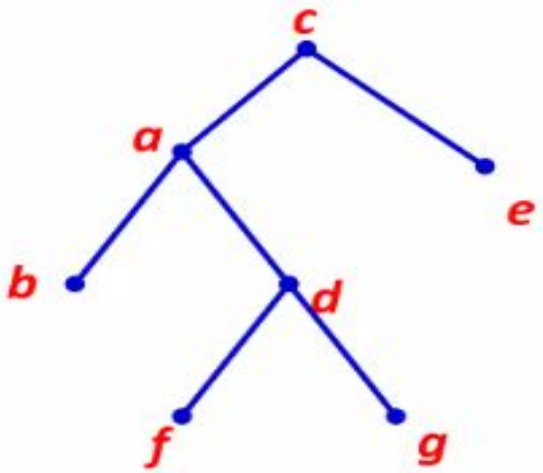
EXAMPLES



A 3-ary TREE



A full 4-ary TREE



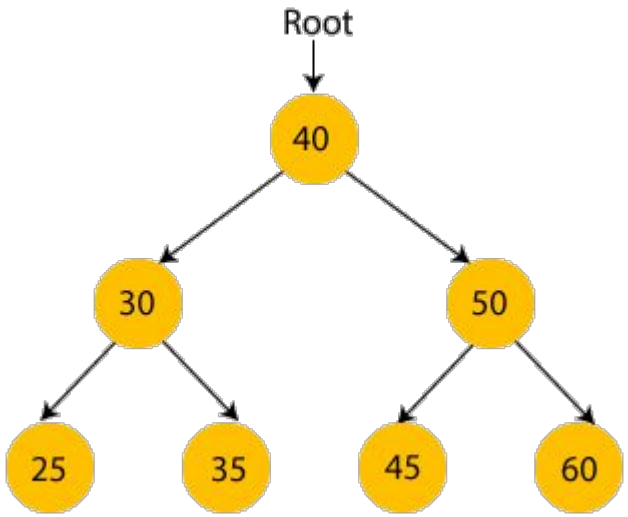
A full binary TREE

Applications of Tree

- Binary Search Tree
- Decision Tree
- Game Tree

Binary search tree

- **Binary Tree:**Each node has at most two children(left and Right)
- In **a Binary search tree(BST)** the value of left node must be smaller than the parent node, and the value of right node must be greater than the parent node. This rule is applied recursively to the left and right subtrees of the root.



Binary search tree

- Example of creating a binary search tree

45, 15, 79, 90, 10, 55, 12, 20, 50

- First, we have to insert **45** into the tree as the root of the tree.
- Then, read the next element; if it is smaller than the root node, insert it as the root of the left subtree, and move to the next element.
- Otherwise, if the element is larger than the root node, then insert it as the root of the right subtree.

Binary search tree

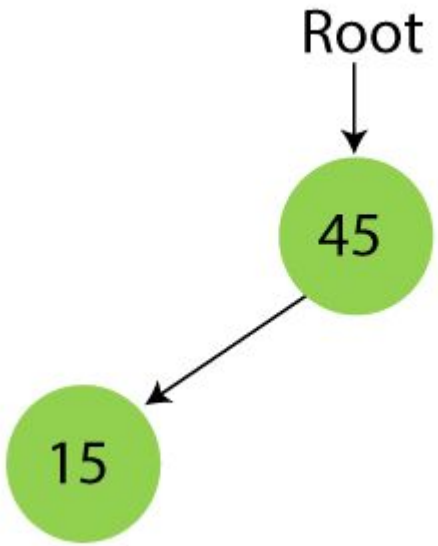
- Example of creating a binary search tree

45, 15, 79, 90, 10, 55, 12, 20, 50

Step 1 - Insert 45.

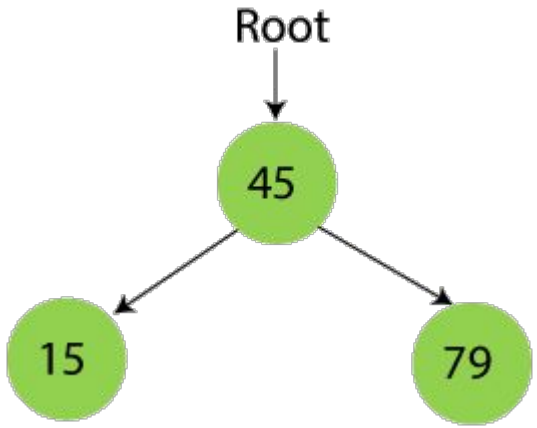
Step 2 - Insert 15.

15 is smaller than 45, so insert it as the root node of the left subtree.



Step 3 - Insert 79.

79 is greater than 45, so insert it as the root node of the right subtree.



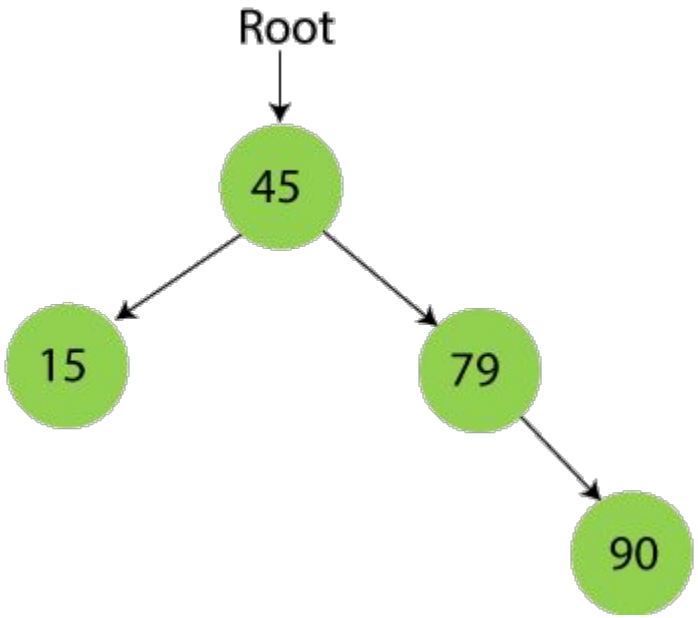
Binary search tree

- Example of creating a binary search tree

45, 15, 79, 90, 10, 55, 12, 20, 50

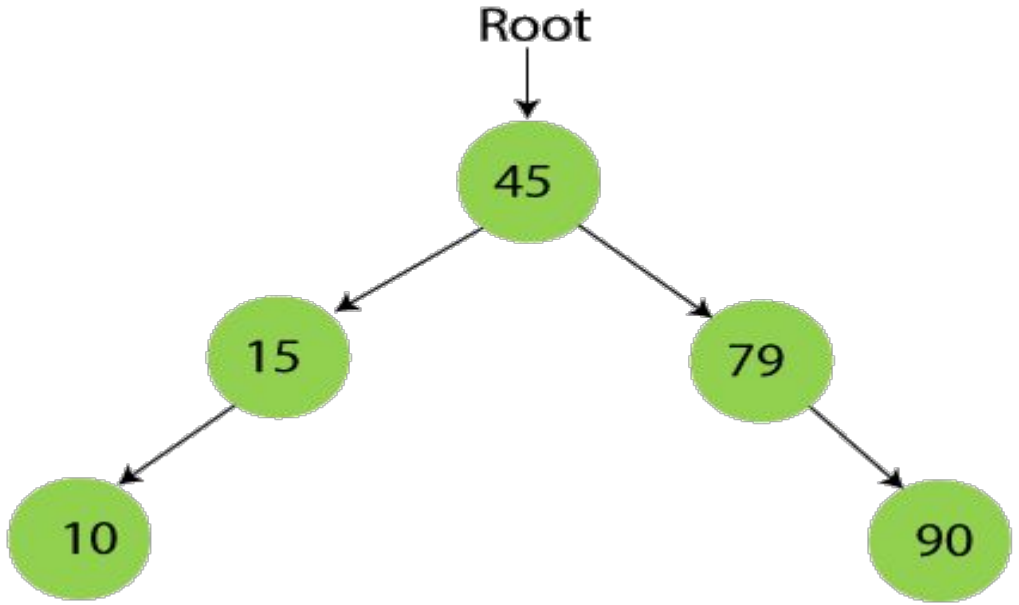
Step 4 - Insert 90

90 is greater than 45 and 79, so it will be inserted as the right subtree of 79



Step 5 - Insert 10.

10 is smaller than 45 and 15, so it will be inserted as a left subtree of 15.



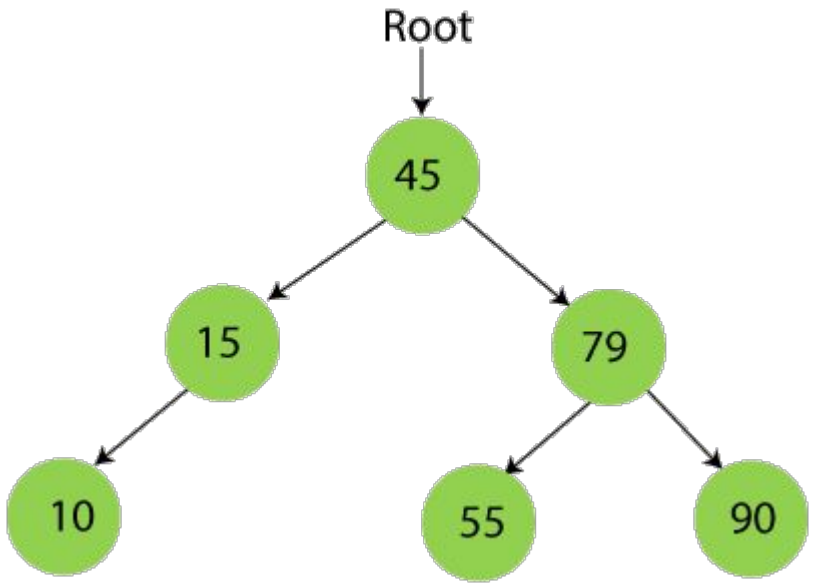
Binary search tree

- Example of creating a binary search tree

45, 15, 79, 90, 10, 55, 12, 20, 50

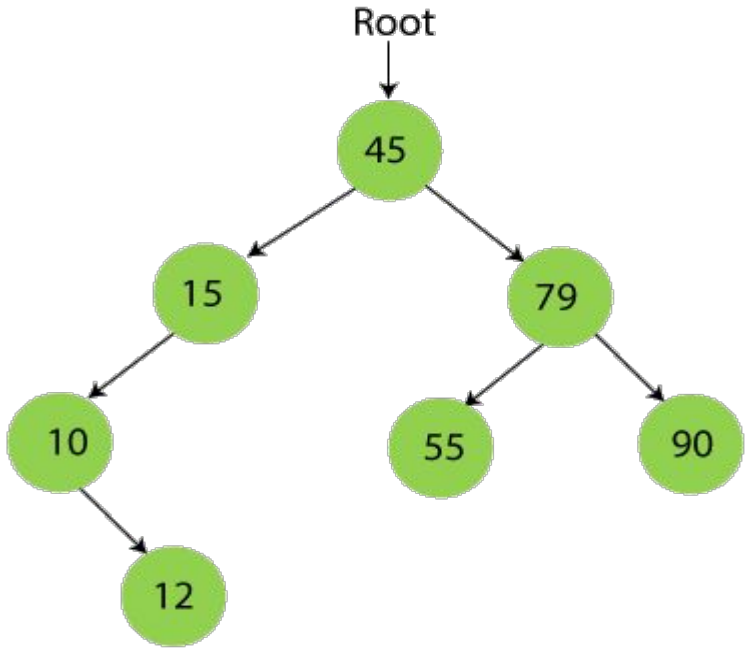
Step 6 - Insert 55

55 is larger than 45 and smaller than 79, so it will be inserted as the left subtree of 79.



Step 7 - Insert 12.

12 is smaller than 45 and 15 but greater than 10, so it will be inserted as the right subtree of 10



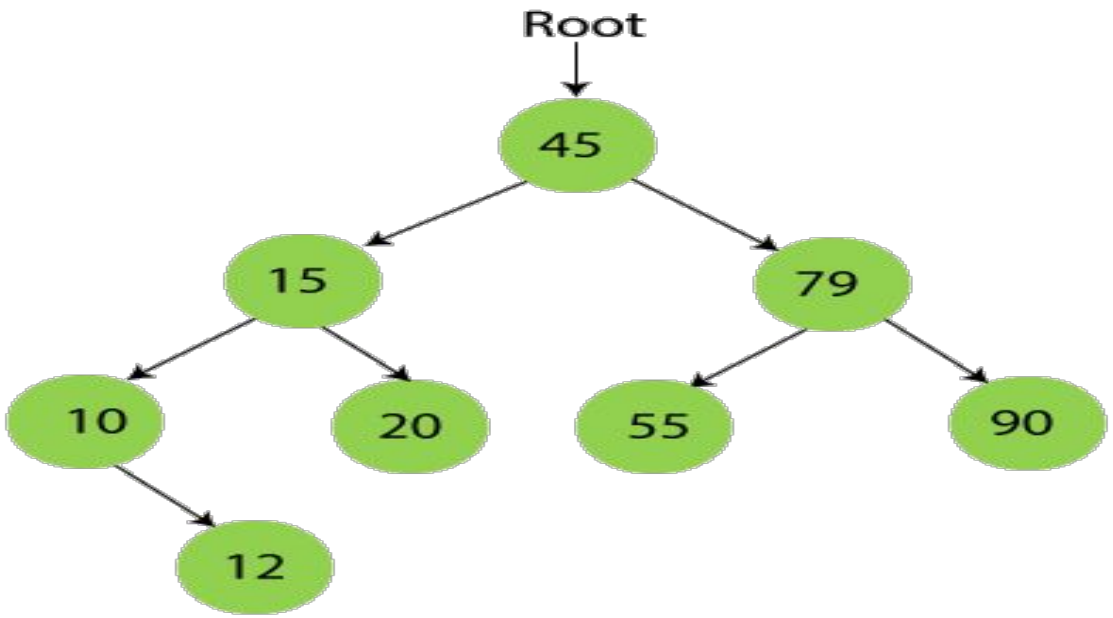
Binary search tree

- Example of creating a binary search tree

45, 15, 79, 90, 10, 55, 12, 20, 50

Step 8 - Insert 20.

20 is smaller than 45 but greater than 15, so it will be inserted as the right subtree of 15.



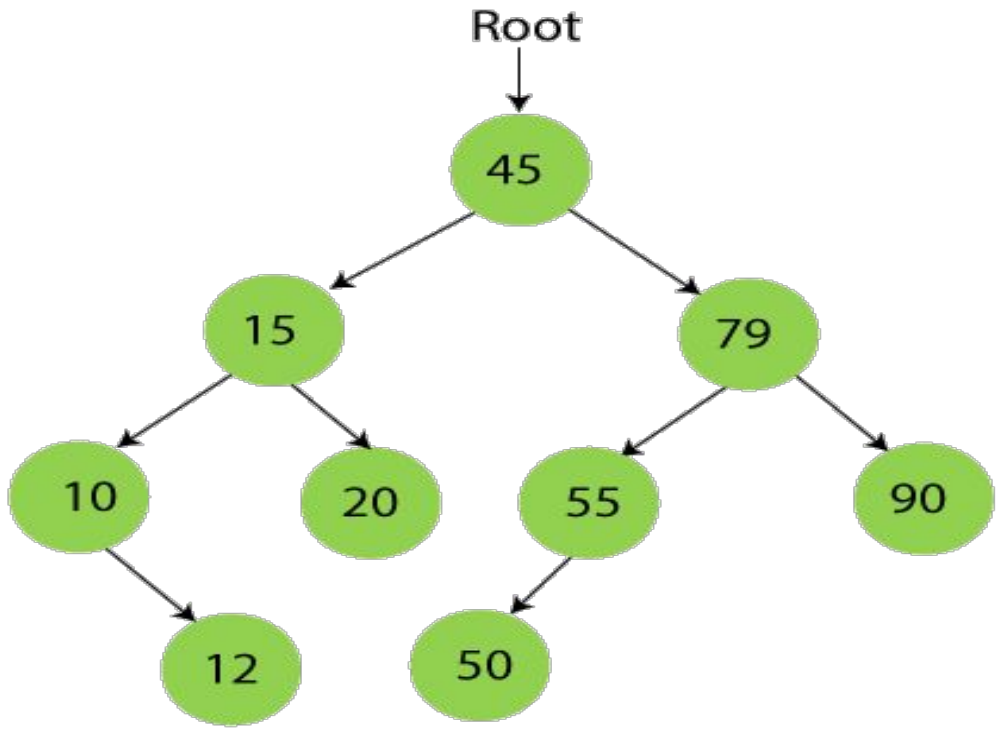
Binary search tree

- Example of creating a binary search tree

45, 15, 79, 90, 10, 55, 12, 20, 50

Step 9 - Insert 50.

50 is greater than 45 but smaller than 79 and 55. So, it will be inserted as a left subtree of 55.



Binary search tree(Assignment)

- Construct binary search tree

21,28,14,18,11,32,25,23,37,27,5,15,19,30,12,26,

Tree traversal

- Traversing or visiting each node of a tree

3 ways of Tree Traversal

- Preorder traversal: root- left- right
- Inorder traversal: left-root-right
- Postorder traversal: left-right-root

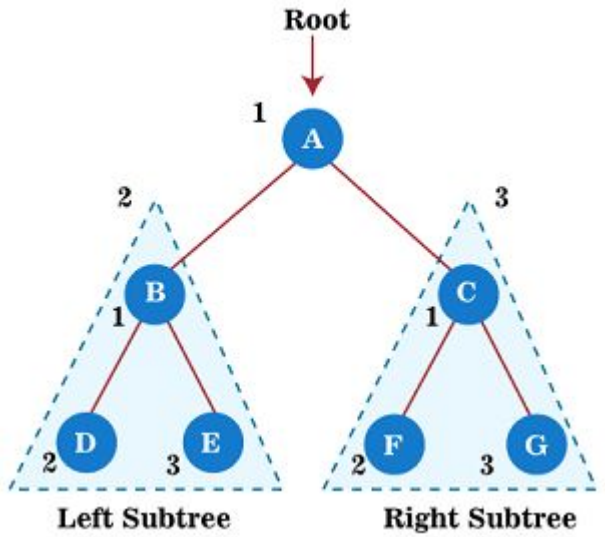
Tree traversal

- Preorder traversal: root- left- right
- First root node is visited after that the left subtree is traversed recursively, and finally, right subtree is recursively traversed.
- As the root node is traversed before (or pre) the left and right subtree, it is called preorder traversal.
- Each node is visited before both of its subtrees.

Step 1 - Visit the root node

Step 2 - Traverse the left subtree recursively.

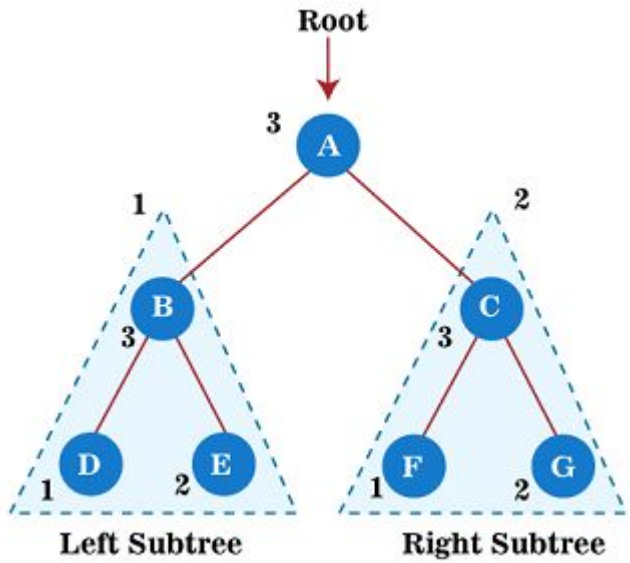
Step 3 - Traverse the right subtree recursively.



Preorder Traversal A → B → D → E → C → F → G

Tree traversal

- Postorder traversal: left-right –root
- Step 1 - Traverse the left subtree recursively.
- Step 2 - Traverse the right subtree recursi
- Step 3 - Visit the root node.

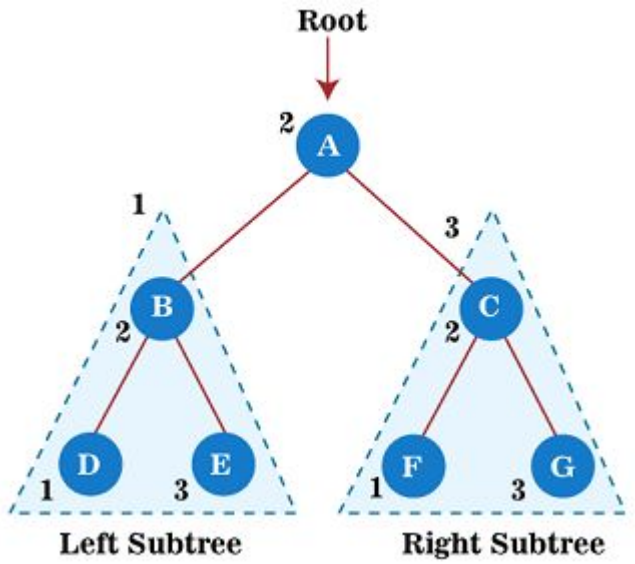


Postorder Traversal D → E → B → F → G → C → A

Tree traversal

Inorder traversal :left- root –right

- Step 1 - Traverse the left subtree recursively.
- Step 2 - Visit the root node.
- Step 3 - Traverse the right subtree recursively.



Inorder Traversal: D → B → E → A → F → C → G

Tree traversal

Construct a binary tree from given inorder and preorder traversals

- Inorder :b d f h k m p t v m
- Preoder: b f d k h v w t m

Arithmetic expressions

- Standard: *infix* form

  (A+B) * C – D/ E

- Fully parenthesized form (in-order & parenthesis):

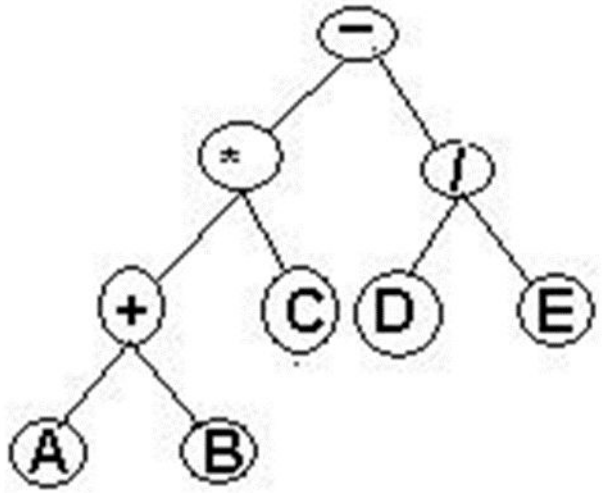
  (((A + B) * C) – (D / E))

- *Postfix* form (reverse Polish notation):

  A B + C * D E / -

- *Prefix* form (Polish notation):

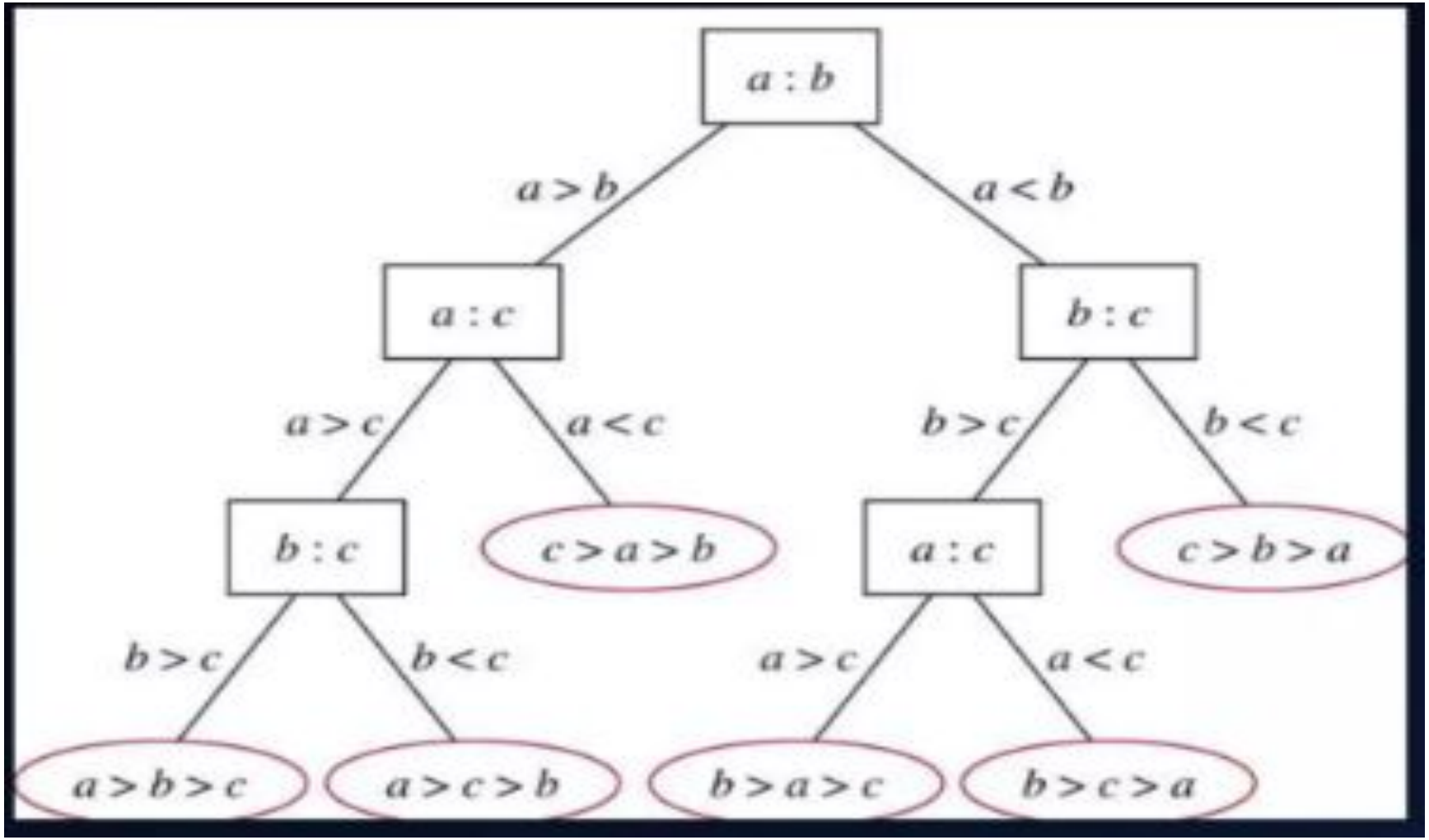
  - * + A B C / D E



Decision Tree

- **Decision trees** are a popular and powerful tool used in various fields such as machine learning, data mining, and statistics.
- **Flowchart-like structure** used to make decisions or predictions.
- It consists of…….
- **Nodes** representing decisions or tests on attributes
- **Branches** representing the outcome of these decisions
- **Leaf** nodes representing final outcomes or predictions.

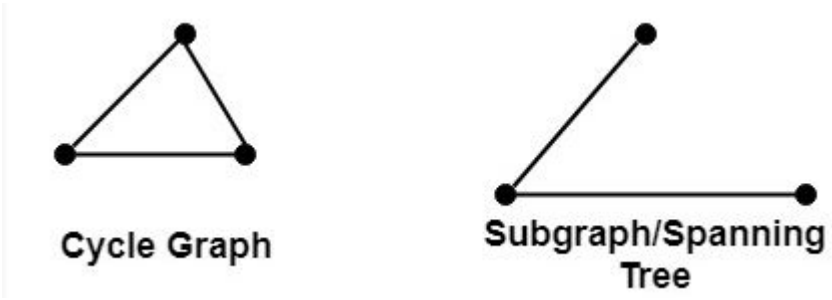
Decision Tree (Example)



Spanning Trees and Minimum Spanning Tree

- A spanning tree of a **connected undirected graph G**
- is a tree that minimally includes all of the vertices of G
- A graph may have many spanning trees.

(A subgraph T of a connected graph G is called spanning tree of G if T is a tree and T include all vertices of G.)



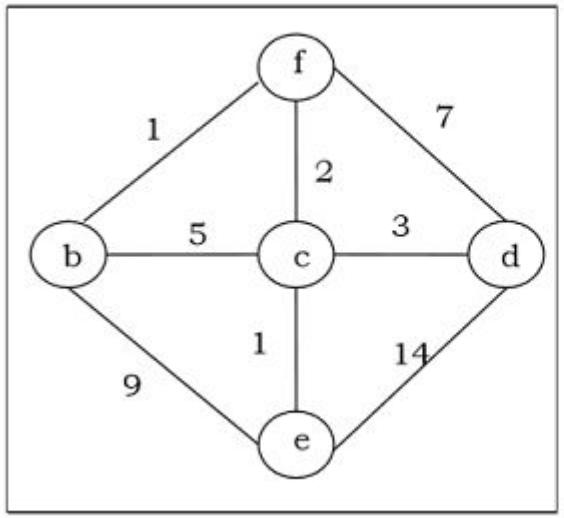
Spanning Trees and Minimum Spanning Tree

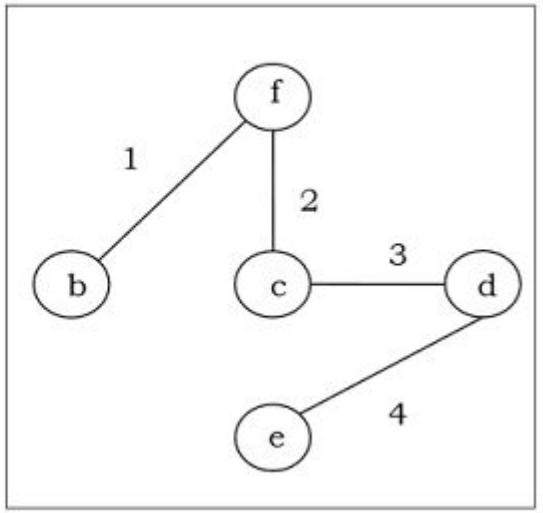
- Minimum Spanning Tree:

- Suppose G is a connected weight graph i.e., each edge of G is assigned a non-negative number called the weight of edge, then any spanning tree T of G is assigned a total weight obtained by adding the weight of the edge in T.

- **A minimum spanning tree of G is a tree whose total weight is as small as possible.**

Example





Kruskal‘s algorithms

- Greedy algorithm that finds a minimum spanning tree for a connected weighted graph.
- It finds a tree of that graph which includes every vertex and the total weight of all the edges in the tree is less than or equal to every possible spanning tree.

Algorithm

Step 1 – Arrange all the edges of the given graph $G(V, E)$ in ascending order as per their edge weight.

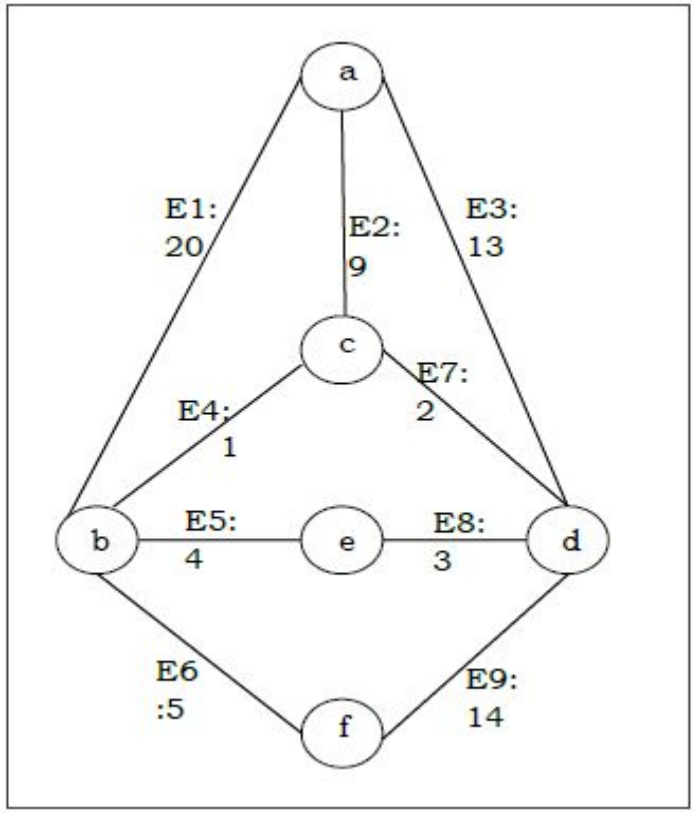
Step 2 – Choose the smallest weighted edge from the graph and check if it forms a cycle with the spanning tree formed so far.

Step 3 – If there is no cycle, include this edge to the spanning tree else discard it.

Step 4 – Repeat Step 2 and Step 3 until $(V - 1)$ number of edges are left in the spanning tree.

Kruskal‘s algorithm

- find minimum spanning tree for the following graph G using Kruskal’s algorithm.

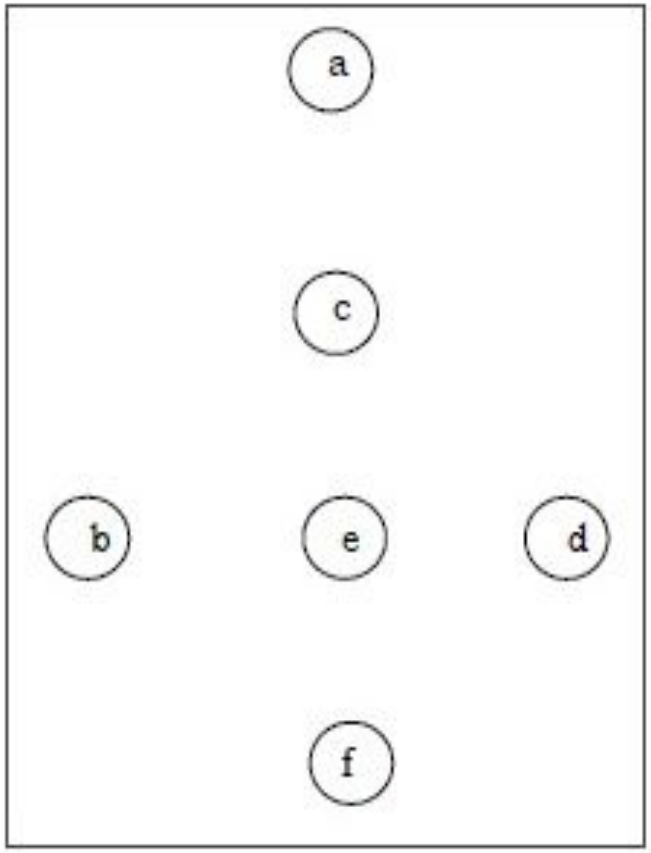


Edge No.	Vertex Pair	Edge Weight
E1	(a, b)	20
E2	(a, c)	9
E3	(a, d)	13
E4	(b, c)	1
E5	(b, e)	4
E6	(b, f)	5
E7	(c, d)	2
E8	(d, e)	3
E9	(d, f)	14

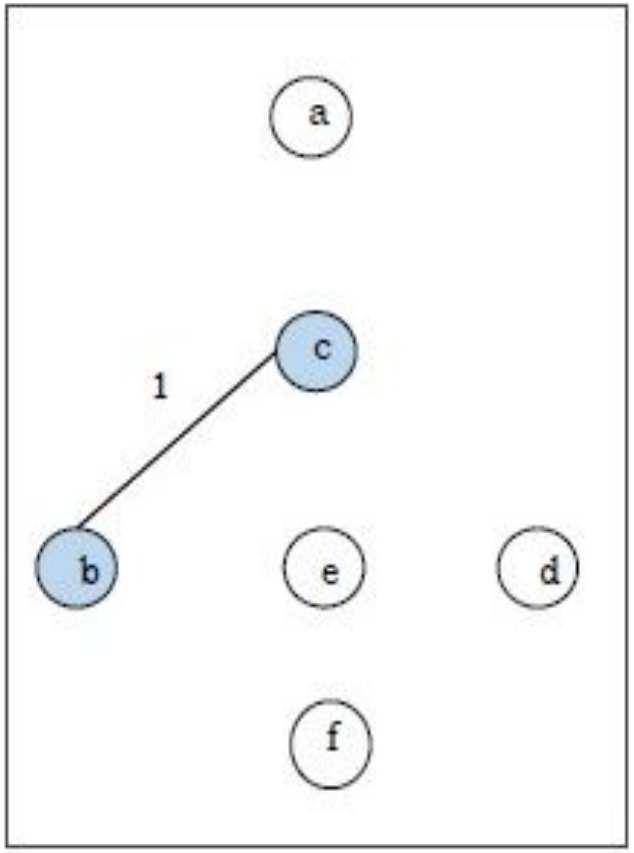
Edge No.	Vertex Pair	Edge Weight
E4	(b, c)	1
E7	(c, d)	2
E8	(d, e)	3
E5	(b, e)	4
E6	(b, f)	5
E2	(a, c)	9
E3	(a, d)	13
E9	(d, f)	14
E1	(a, b)	20

Rearrange the table in ascending order with respect to Edge weight

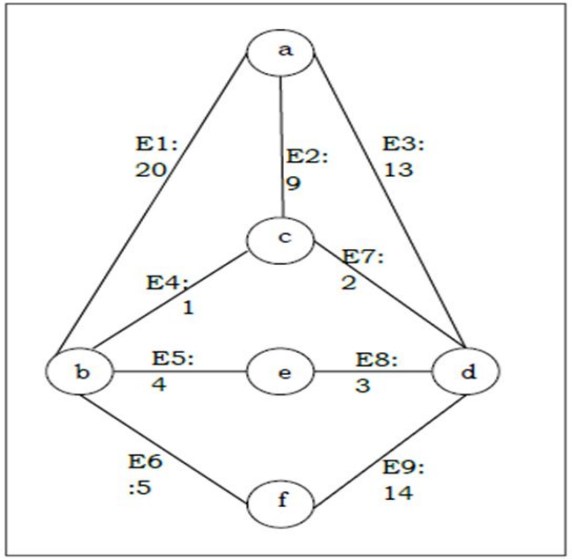
Kruskal‘s algorithm



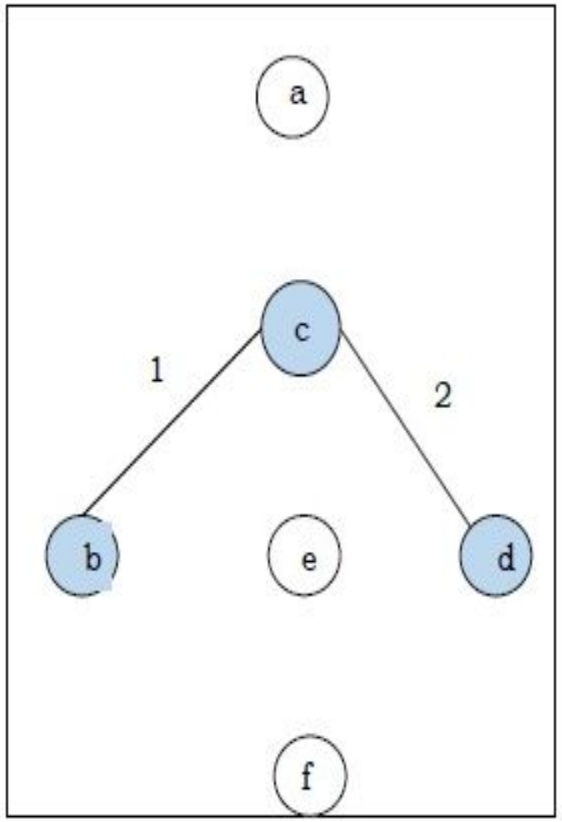
After adding vertices



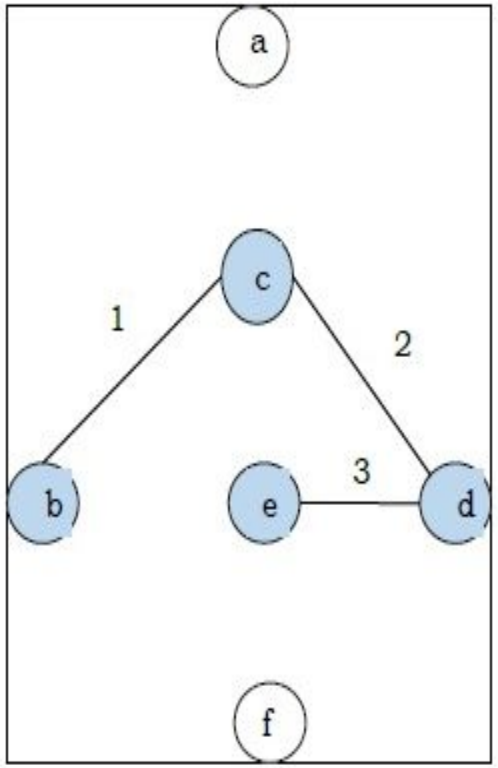
After adding edge E4



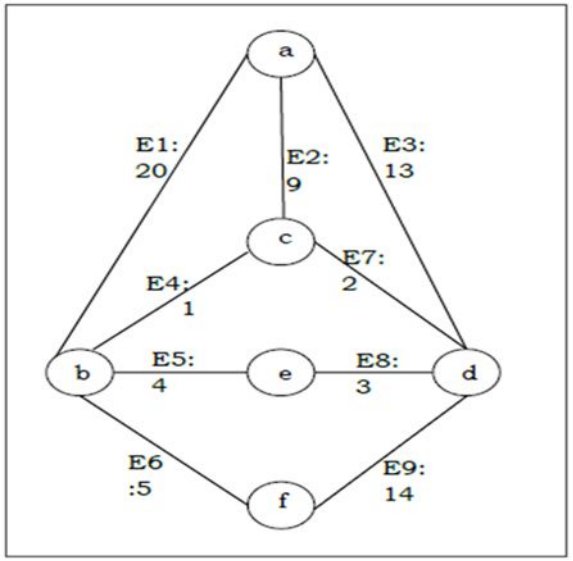
Kruskal‘s algorithm



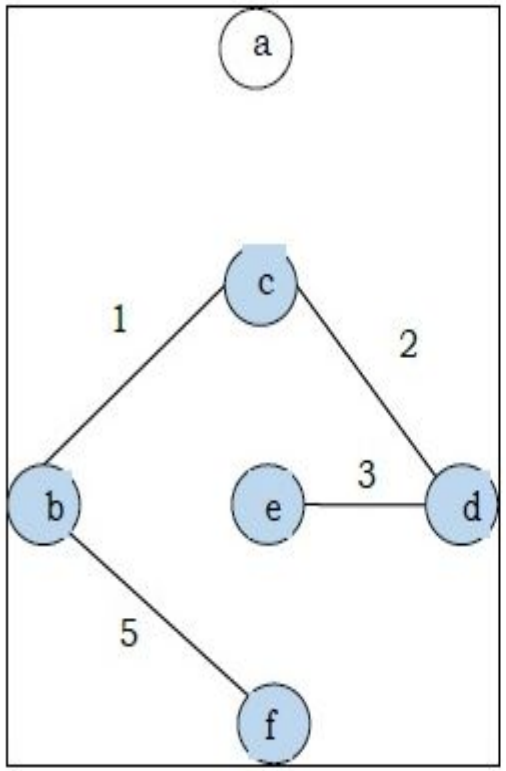
After adding edge E7



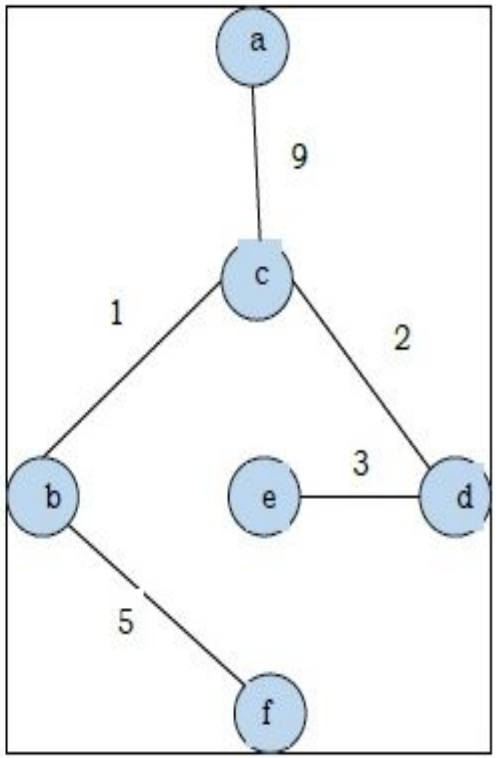
After adding edge E8



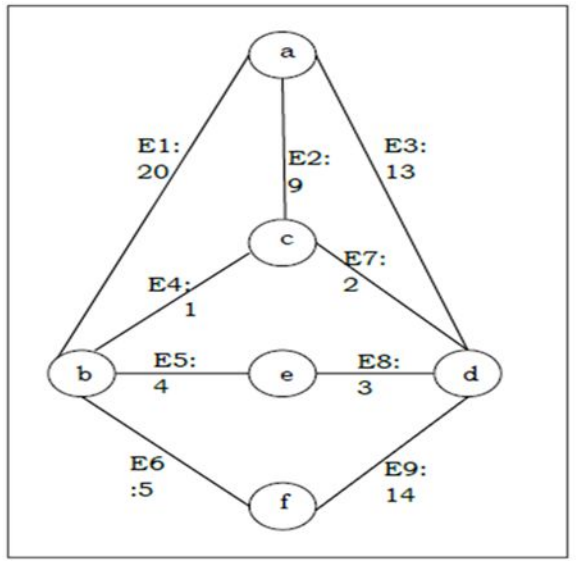
Kruskal‘s and Prim‘s algorithms



After adding edge E6
(don’t add E5 since it forms cycle)



After adding edge E2



minimal spanning tree and its total weight is $(1+2+3+5+9)=20.$

Prim‘s algorithm

- Discovered in 1930 by mathematicians, Vojtech Jarnik and Robert C. P
- Greedy algorithm that finds a minimum spanning tree for a connected weighted graph.
- It finds a tree of that graph which includes every vertex and the total weight of all the edges in the tree is less than or equal to every possible spanning tree.
- Prim’s algorithm is faster on dense graphs.

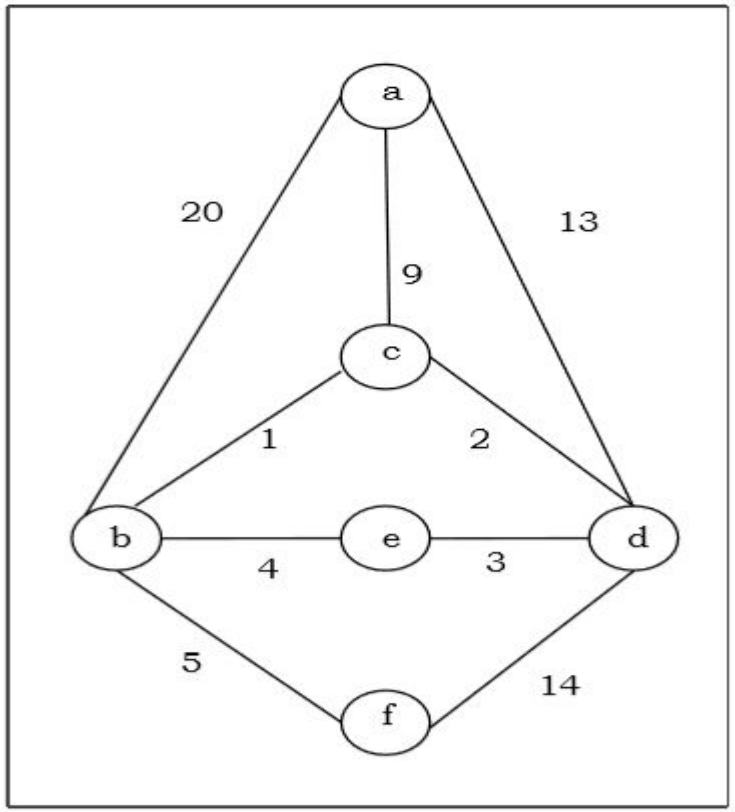
Prim‘s algorithm

Algorithm

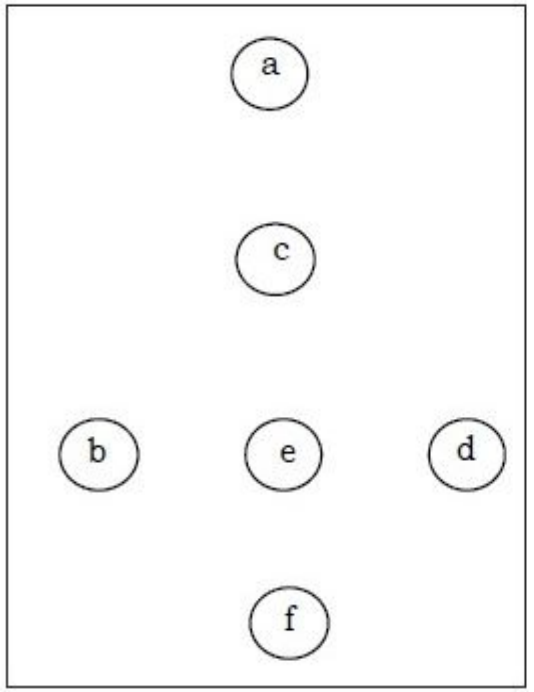
1. Initialize the minimal spanning tree with a single vertex, randomly chosen from the graph.
2. Repeat steps 3 and 4 until all the vertices are included in the tree.
3. Select an edge that connects the tree with a vertex not yet in the tree, so that the weight of the edge is minimal and inclusion of the edge does not form a cycle.
4. Add the selected edge and the vertex that it connects to the tree.

Prim‘s algorithm

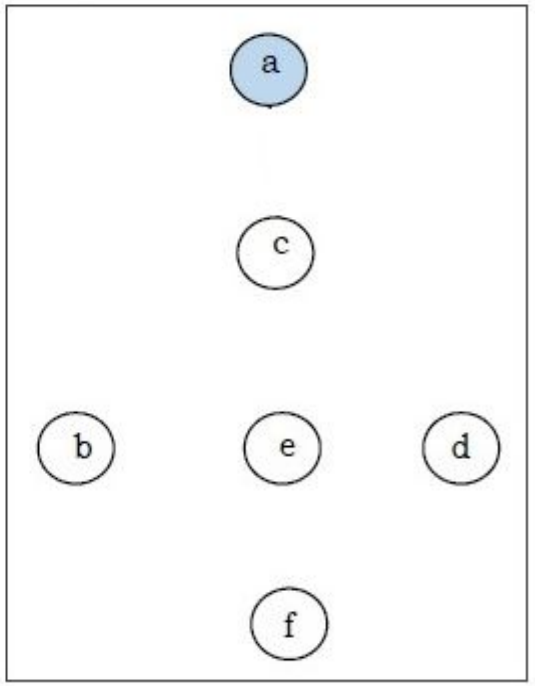
- Find minimum spanning tree for the following graph G using Prim’s algorithm.
-



start with the vertex ‘a’ and proceed.

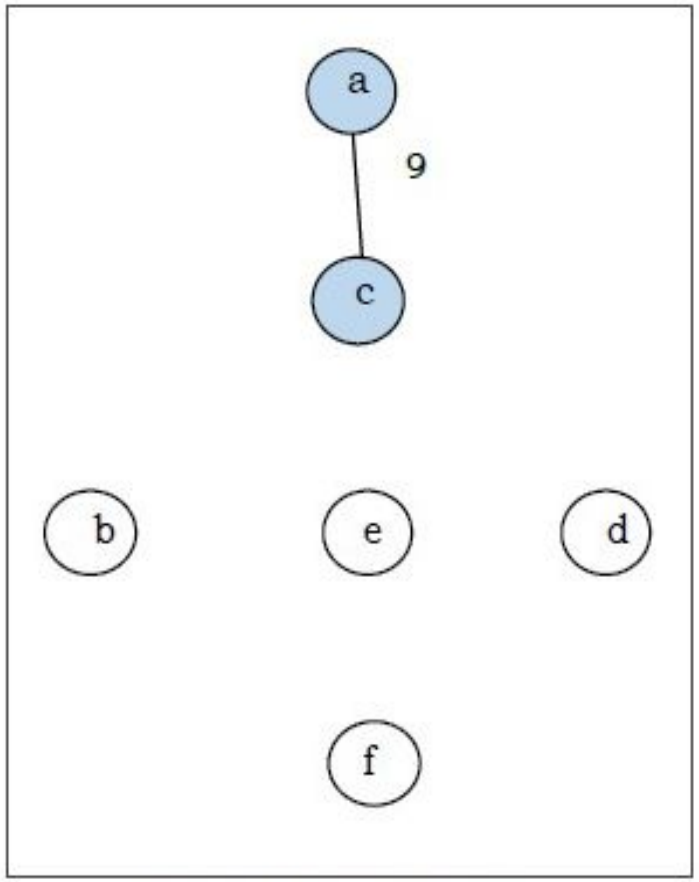


No vertices added

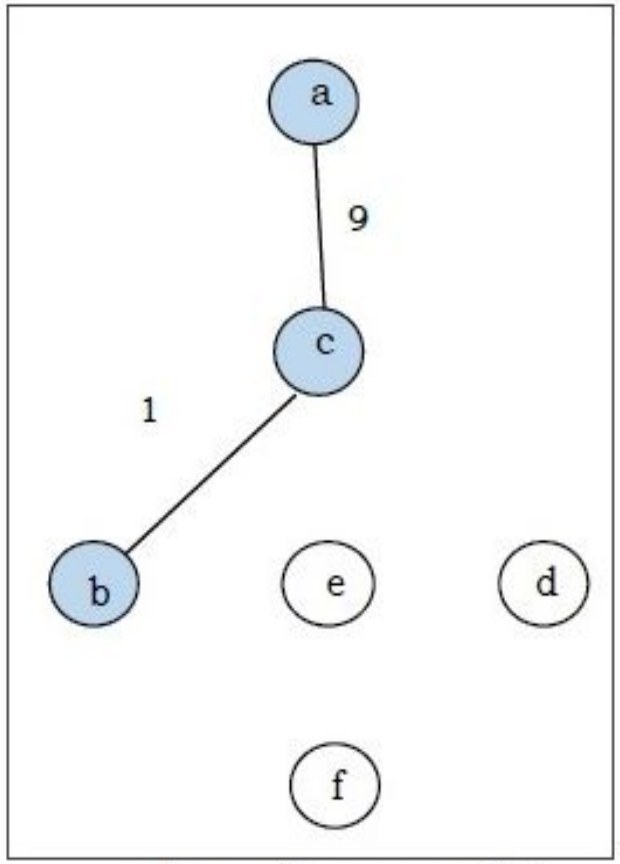


After adding vertex ‘a’

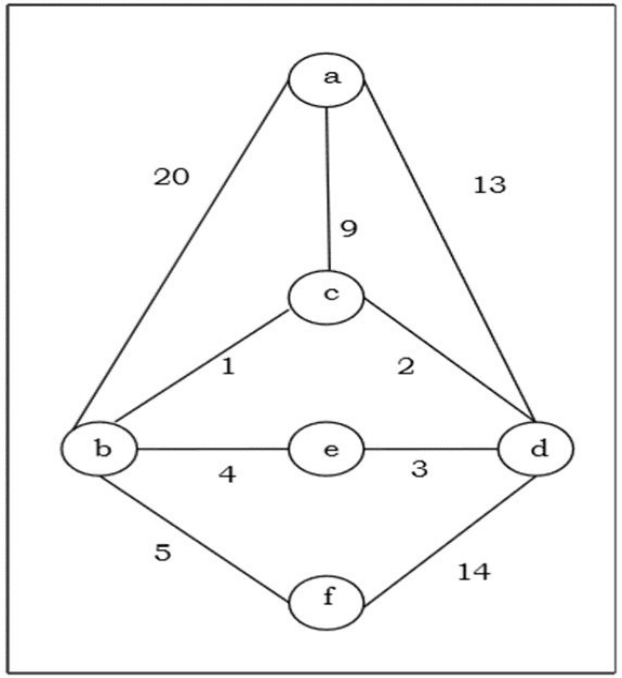
Prim‘s algorithm



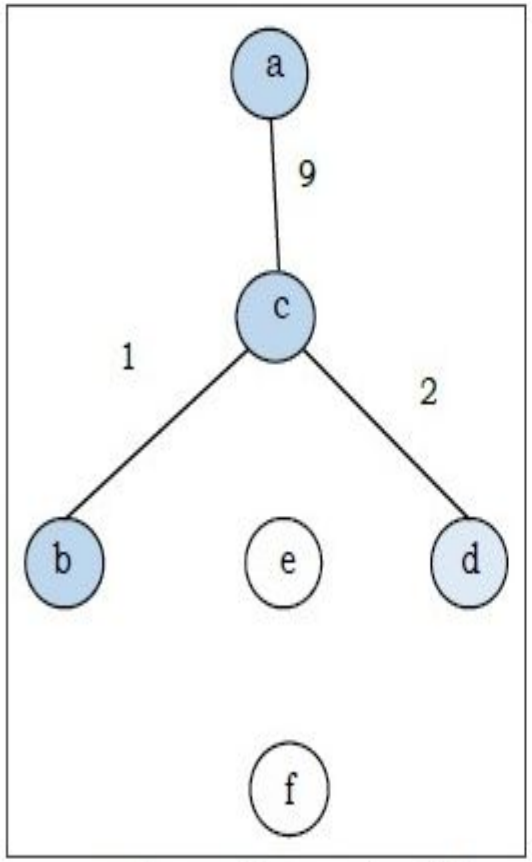
After adding vertex ‘c’



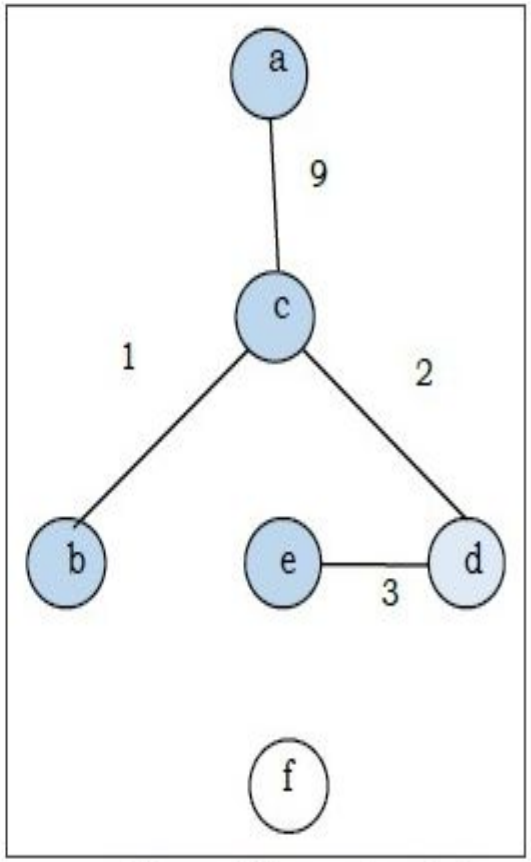
After adding vertex ‘b’



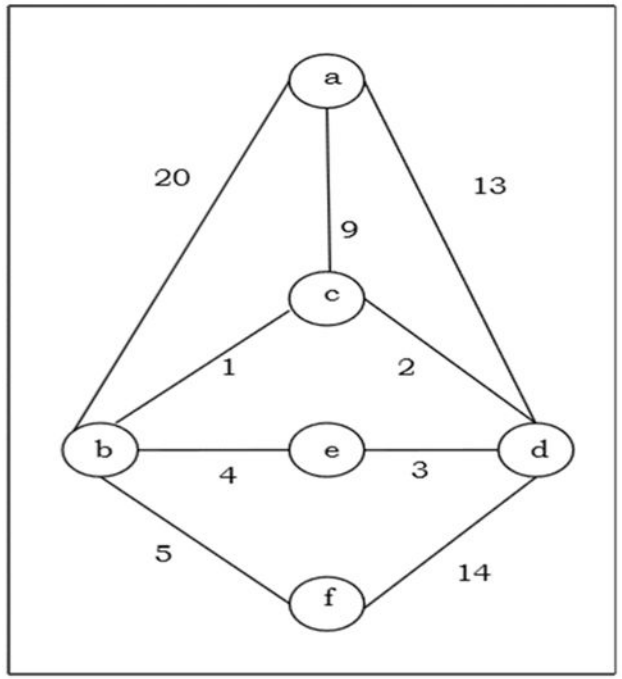
Prim‘s algorithm



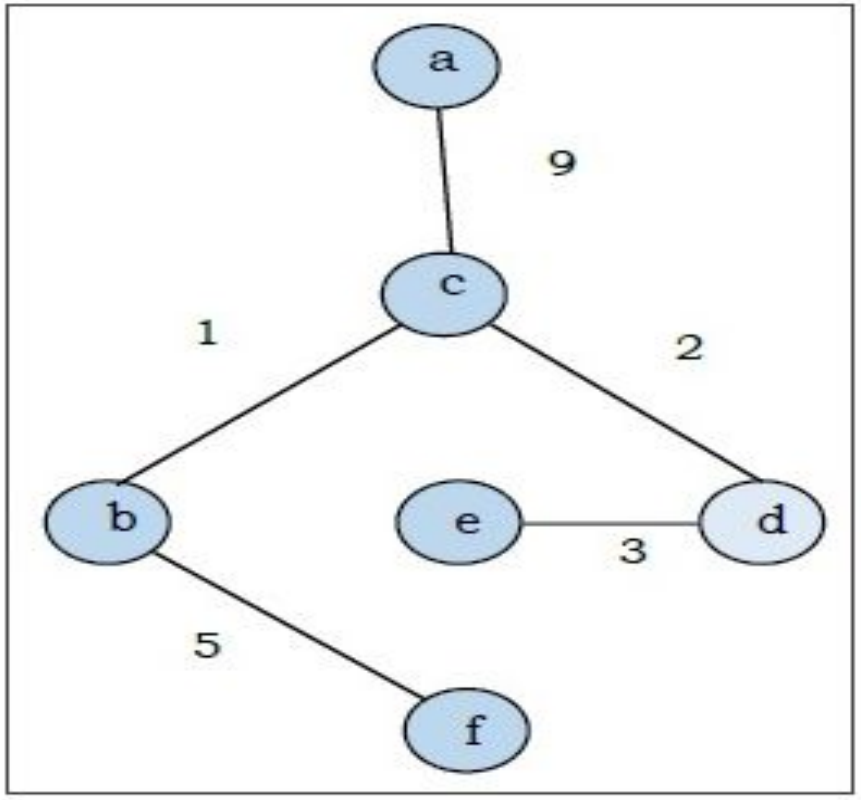
After adding vertex ‘d’



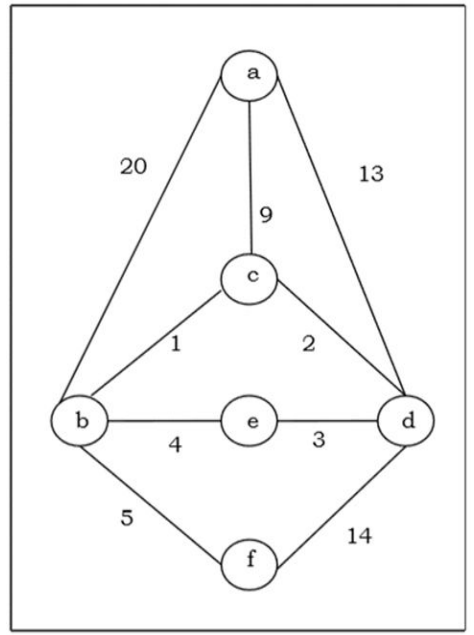
After adding vertex ‘e’



Prim‘s algorithm



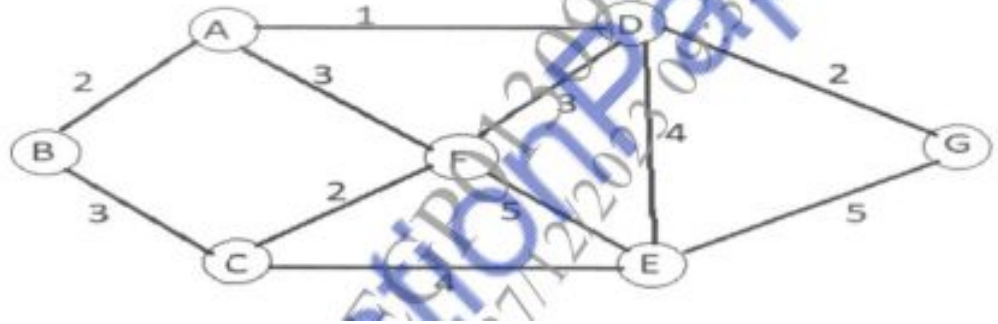
After adding vertex ‘f’



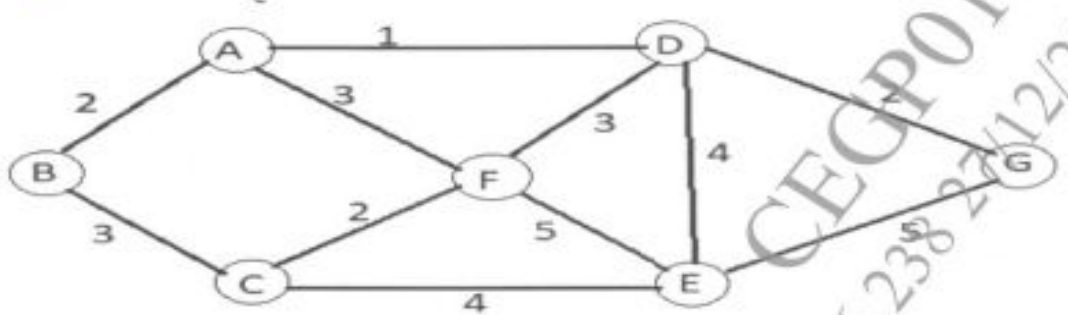
minimal spanning tree and its total weight is (1+2+3+5+9)=20

Questions

c) Find minimum spanning tree using Kruskals Algorithm. **[6]**

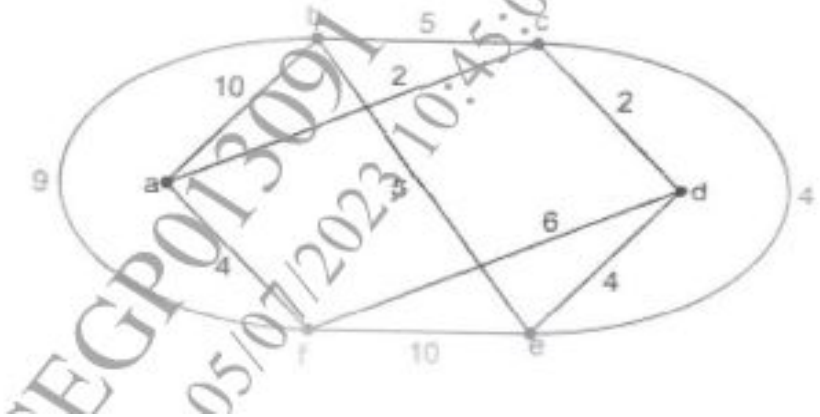


c) Find minimum Spanning tree using prims algorithm **[6]**



Questions

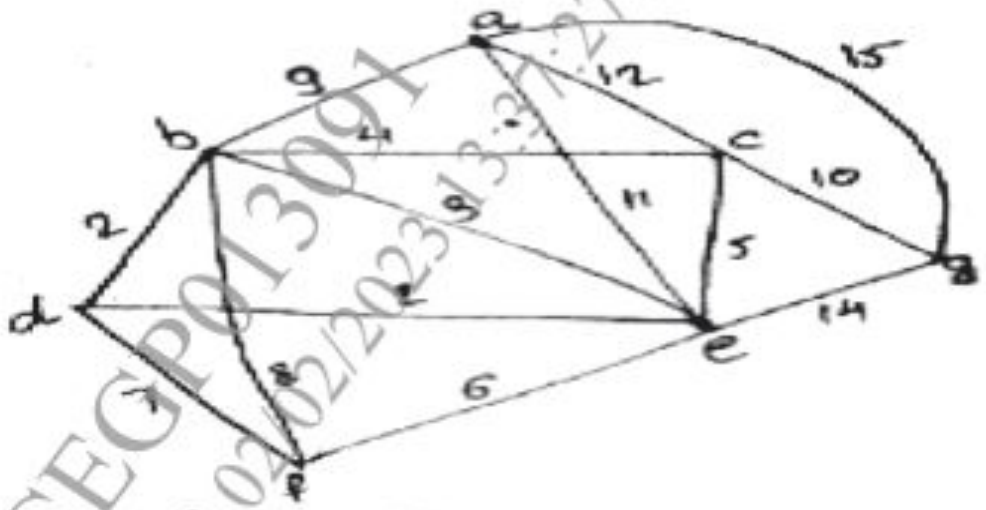
c) Use Kruskal's algorithm to find the minimum spanning tree for the connected weighted graph G as shown in fig. below **[6]**



c) What is Minimum Spanning tree? Explain briefly steps involved in finding MST in Prim's Algorithm? **[6]**

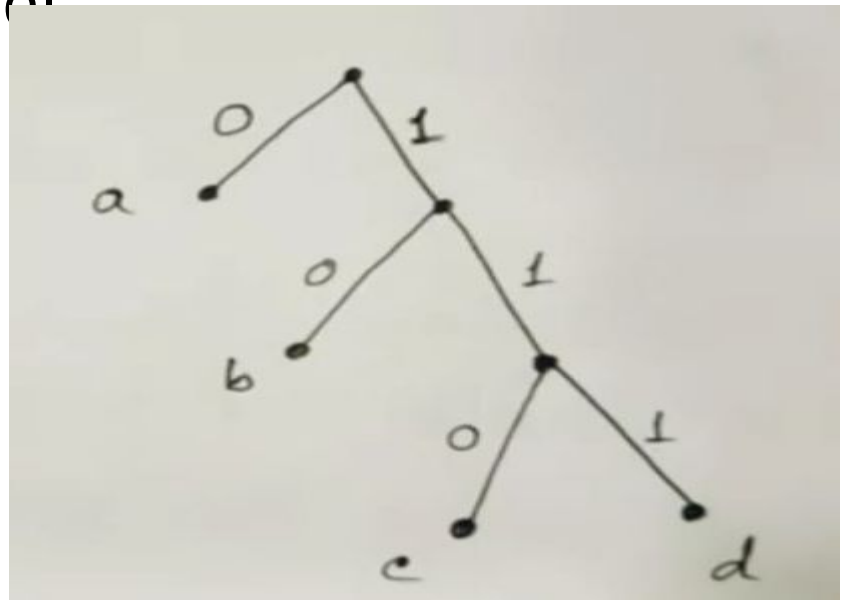
Questions

c) Give the stepwise construction of minimum spanning tree using Prims algorithm for the following graph. Obtain the total cost of minimum spanning tree. **[6]**



Prefix codes

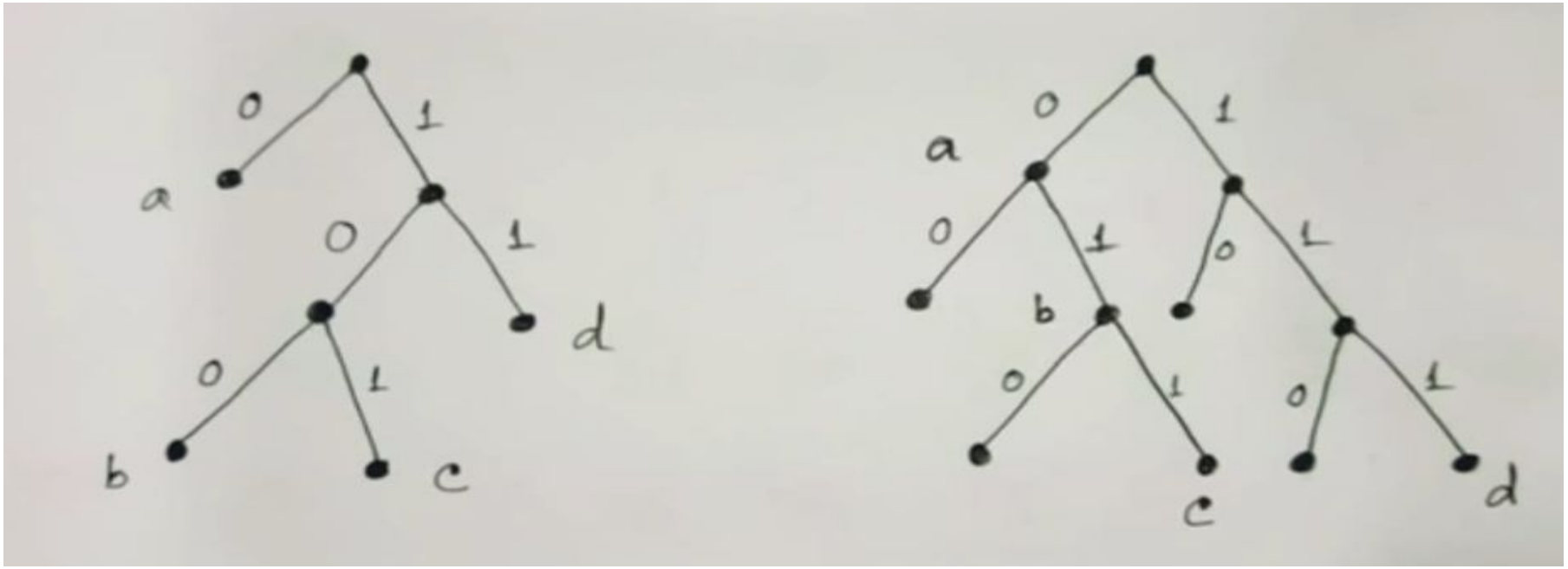
- A "prefix code" is a type of encoding mechanism ("code").
- **prefix code**, variable length the entire set of possible encoded values ("codewords") must not contain *any* values that start with any *other* value in the set.
- a=0
- b=10
- c-=110
- D=111



Prefix codes

- To verify given code is prefix not not
- EX: a=0 b=100 c=101 d=11

  a=0 b=01 c=011 d=111



Prefix Code

Not Prefix Code

Prefix codes

- Purpose

- To encode or Compress Data

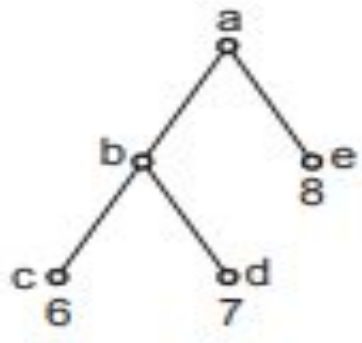
Optimal Tree

Let T be any full binary tree and $W_1, W_2, W_3, \ldots W_t$ be the weights of the leaves (terminal vertices) then the weight W of the full binary tree is given by

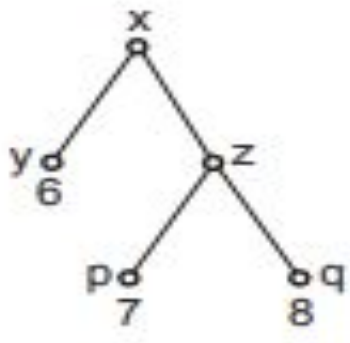
$$W(T) = \sum_{i=0}^{t} W_i l_i$$

Where $l_i = l(i)$ is the length of the path of the leaf i from the root of the tree. The full binary tree is called an optimal tree if its weight is minimum.

Optimal Tree



T_1



T_2

$$W(T_1) = 6 \times l(c) + 7 \times l(d) + 8 \times l(e)$$
$$= 6 \times 2 + 7 \times 2 + 8 \times 1$$
$$= 12 + 14 + 8 = 34$$

$$W(T_2) = 6 \times 1 + 7 \times 2 + 8 \times 2$$
$$= 6 + 14 + 16 = 36$$

$$W(T_1) < W(T_2)$$

T_1 is the optimal tree

Huffman coding

- Huffman coding is a lossless data compression algorithm
- The idea is to assign variable-length codes to input characters, lengths of the assigned codes are based on the frequencies of corresponding characters.
- The variable-length codes assigned to input characters are Prefix Codes
- To construct optimal prefix code is called **Huffman coding**.

Huffman coding

- ***Steps to build Huffman Tree***

1. Create a leaf node for each unique character and build a min heap of all leaf nodes (Min Heap is used as a priority queue. The value of frequency field is used to compare two nodes in min heap. Initially, the least frequent character is at root)
2. Extract two nodes with the minimum frequency from the min heap.
3. Create a new internal node with a frequency equal to the sum of the two nodes frequencies. Make the first extracted node as its left child and the other extracted node as its right child. Add this node to the min heap.
4. Repeat steps 2 and 3 until the heap contains only one node. The remaining node is the root node and the tree is complete.

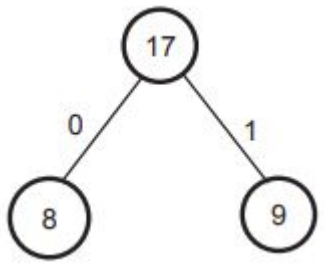
Huffman coding

8, 9, 12, 14, 16, 19.

Step 1 : Increasing order of weights :

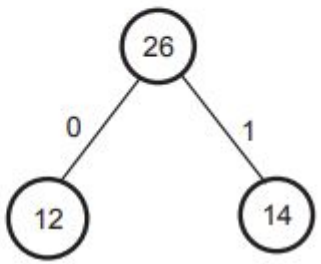
8, 9, 12, 14, 16, 19
↑ ↑

(↑ : First two smallest weights)

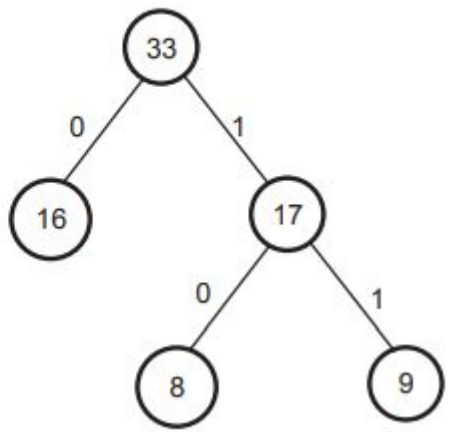


Step 2 : Rewrite sequence of weights in increasing order

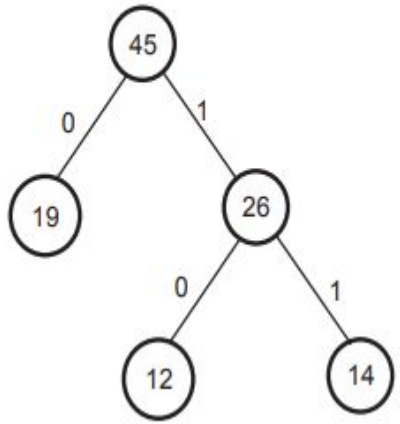
12, 14, 16, 17, 19
↑ ↑



Step 3 : Sequence : 16, 17, 19, 26,
↑ ↑

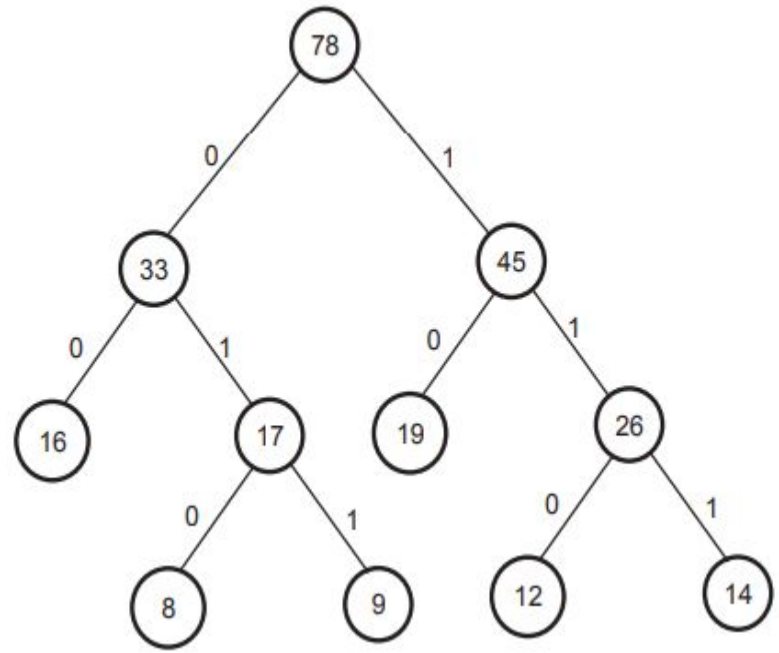


Step 4 : Sequence : 19, 26, 33,



Step 5 : Sequence : 33, 45

Symbols	Binary Prifix Codes
16	00
8	010
9	011
19	10
12	110
14	111



The Max flow- Min Cut Theorem (Transport network)

- A **transportation network** is a connected, weighted, directed graph with the following properties.
- There is one **source**, a vertex with no incoming edges. [the vertex has indegree 0.]
- There is one **sink**, a vertex with no outgoing edges. [the vertex has outdegree 0.]
- Each edge (u,v) is assigned a nonnegative weight Cuv called the **capacity** of that edge.
- **Cut:** is a set of edges whose removal divides a connected graph into two disjoint subsets.

The Max flow- Min Cut Theorem (Transport network)

-Cut in a transportation network G is a partition of the vertices of G into two sets S and T so that the source is in S and the sink is in T two

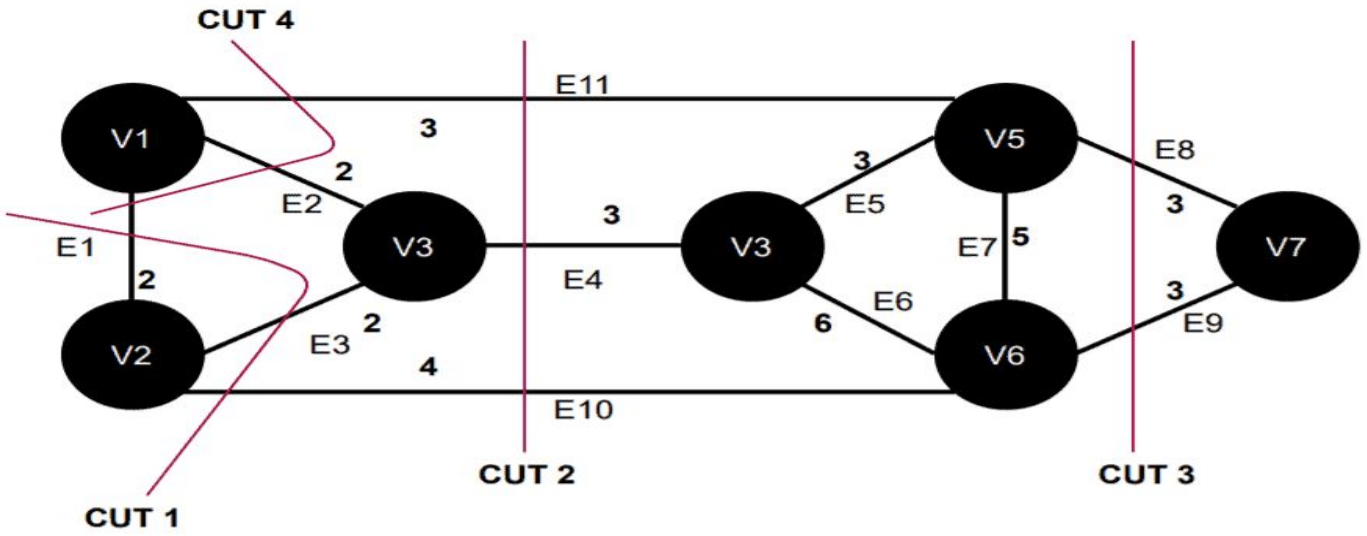
-Variations of a cut:

Maximum cut

Minimum cut

The Max flow- Min Cut Theorem (Transport network)

- Minimum cut of a weighted graph is defined as the minimum sum of weights of edges that, when removed from the graph, divide the graph into two sets.



The Max flow- Min Cut Theorem (Transport network)

Maximum flow is defined as the maximum amount of flow that the graph or network would allow to flow from the source node to its sink node.

Max-Flow Min-Cut Theorem

Theorem states that the maximum flow through any network from a given source to a given sink is exactly equal to the minimum sum of a cut

Let G=(X,A) be a flow network, ϕ a flow on G. The value of the maximum flow from the source S to the sink P is equal to the capacity of the minimum cut CT separating S and P

Find maximum flow in the transport network using labeling procedure. Determine the corresponding min-cut. **[6]**

For the following graph find different cut set and identify the max flow in given network? [6]