Discrete Mathematics
Code - PCC - CS401
Stream - CSE/IT

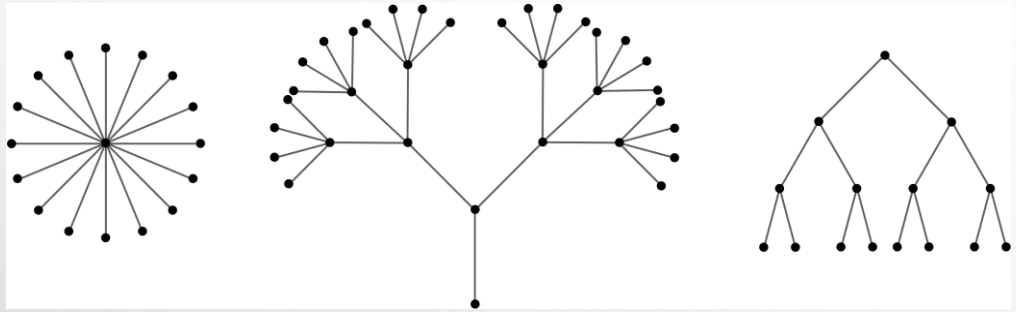
Prepared by - Subhra Sarkar

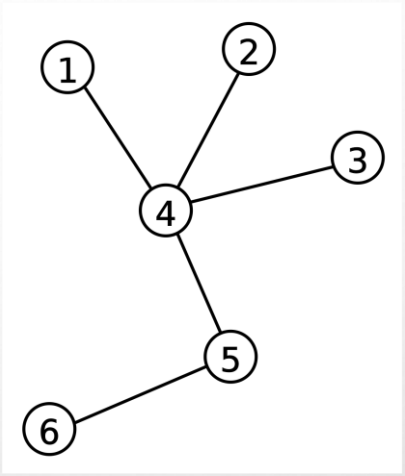
Module – V
Graph Theory contd…

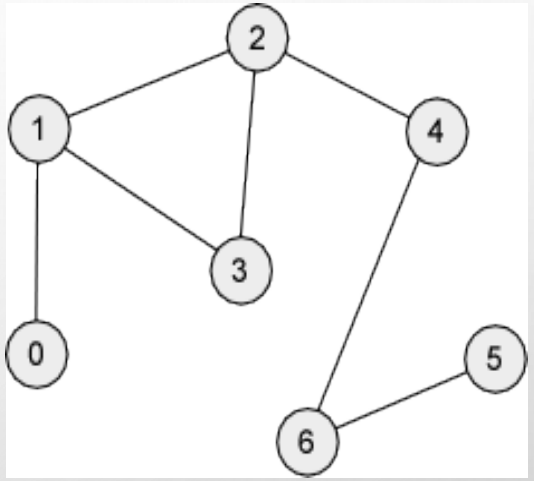
- Rooted Trees
- Trees & Sorting
- Weighted Trees & Prefix Codes
- Bi – connected component
- Articulation Points
- Shortest distances
- Binary Tree
- Binary Search Trees
- Huffman Coding
- Depth First Search
- Breadth First Search
- Dijkstra’s Algorithm

Tree

- A tree is a connected graph with no simple circuit. A collection of disjoint trees is called, quite appropriately, a forest.







- Properties of Tree –

➢ There is a unique path between every two vertices in a tree.

➢ The number of vertices is one more than the number of edges in a tree.

➢ A tree with two or more vertices has at least two leaves.

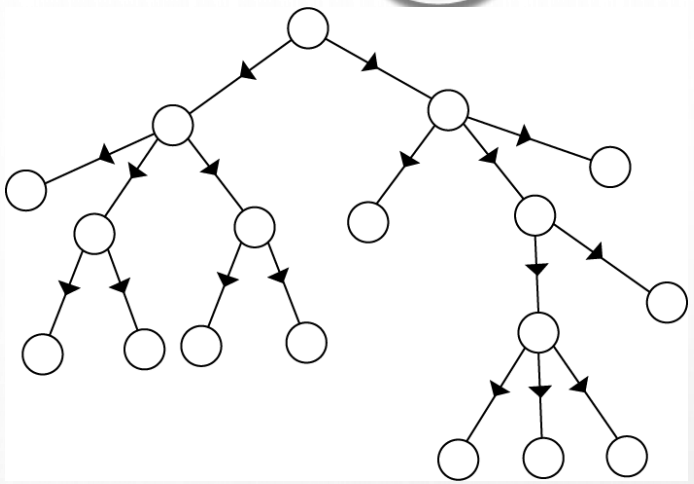
Equivalent definitions of trees – A graph is a tree if and only if

1. There is a unique path between every pair of vertices in the graph.

2. It is connected and $e = n - 1$.

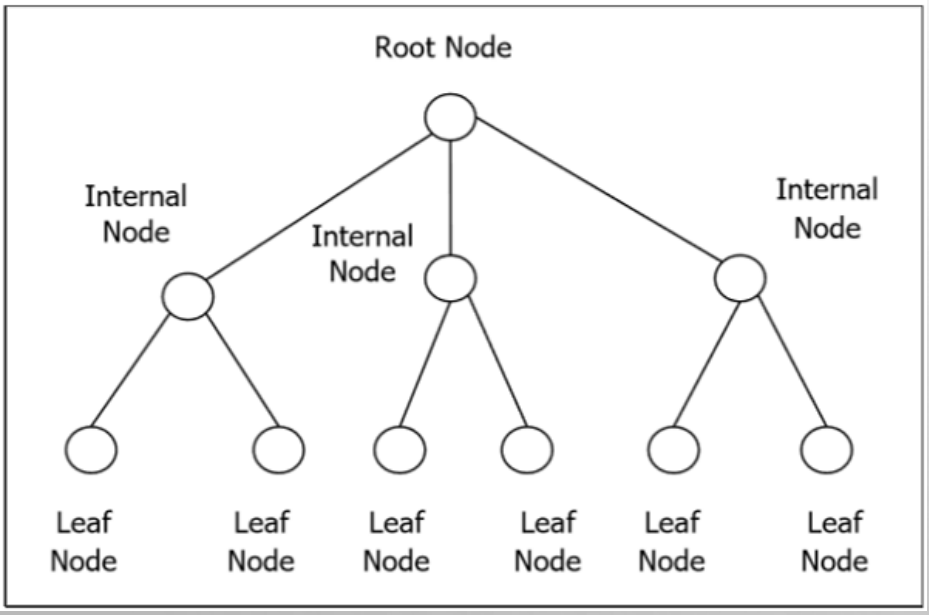
3. It has no circuits with $e = n - 1$.

Rooted Trees



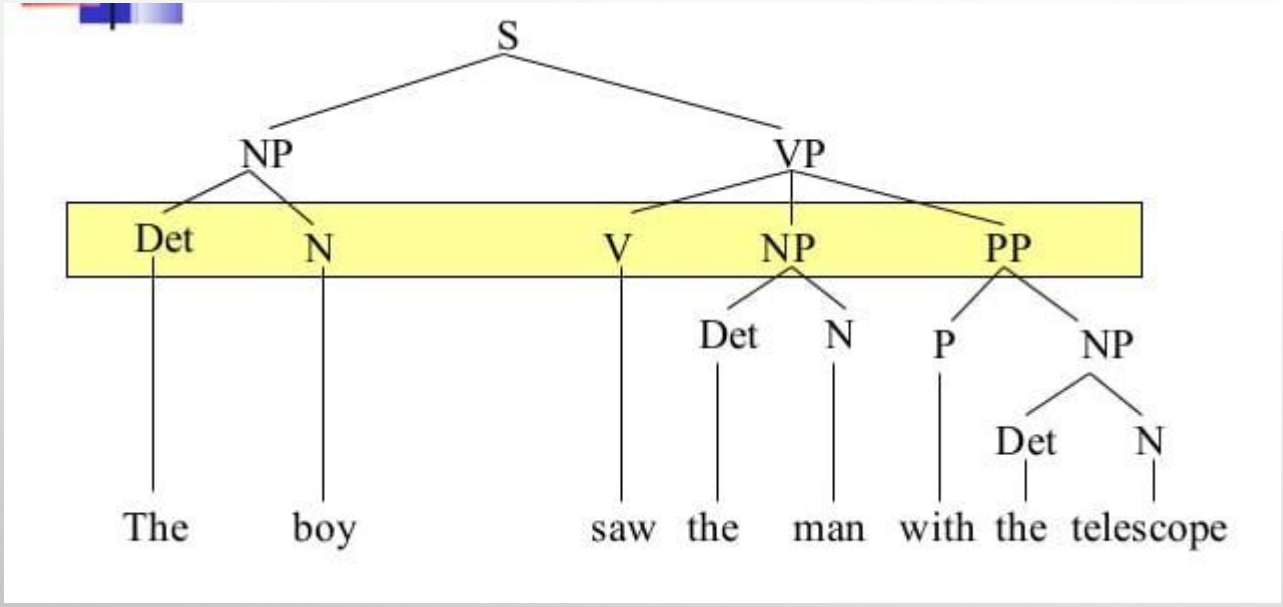
- A directed graph is said to be a directed tree if it becomes a tree when the directions of the edges are ignored.

- A directed tree is called a rooted tree if there is exactly one vertex whose incoming degree is 0 and incoming degree of all other vertices are 1.

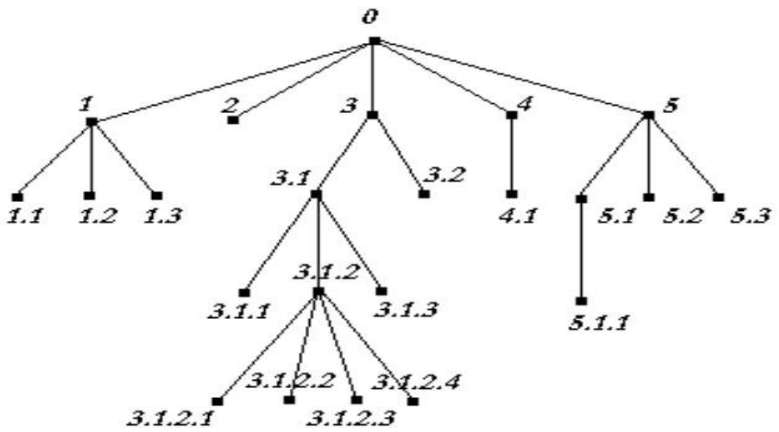


- The vertex with incoming degree 0 is called the root.
- A vertex whose outgoing degree is 0 is called a leaf or a Terminal node.
- A vertex whose outgoing degree is non-zero is called a branch node or an internal node.

➢ Ordered Tree – An ordered tree is a rooted tre with the edges incident from each branch node labeled with integers 1,2,….



Universal Address System of an Ordered Rooted Tree



3

- An ordered tree in which every branch node has at most m sons is called an m- ary tree.
- An m-ary tree is said to be full if every internal vertex has exactly m sons.

❖ An important class of m-ary trees is binary tree. For binary trees, we often refer to the left subtree and right subtree of the node.

Binary Tree

- If the outdegree of every node is less than or equal to 2, in a directed tree than the tree is called a binary tree.

- Basic Terminology –

  - Left Child: The node to the left of the root is called its left child.
  - Right Child: The node to the right of the root is called its right child.
  - Parent: A node having a left child or right child or both are called the parent of the nodes.
  - Siblings: Two nodes having the same parent are called siblings.
  - Leaf: A node with no children is called a leaf. The number of leaves in a binary tree can vary from one (minimum) to half the number of vertices (maximum) in a tree.
  - Descendant: A node is called descendant of another node if it is the child of the node or child of some other descendant of that node. All the nodes in the tree are descendants of the root.

- Left Subtree: The subtree whose root is the left child of some node is called the left subtree of that node.
- Right Subtree: The subtree whose root is the right child of some node is called the right subtree of that node.
- Level of a Node: The level of a node is its distance from the root. The level of root is defined as zero. The level of all other nodes is one more than its parent node. The maximum number of nodes at any level N is 2^N.

Path Length in Rooted trees

❑ Result – A full m - ary tree with

i. i internal vertices has n = mi + 1 vertices and l = (m-1)i + 1 leaves.
ii. n vertices has i = (n-1)/m internal vertices and l = $[(m-1)n+1]/m$ leaves.
iii. l leaves has n = (ml – 1)/(m-1) vertices and i = (l - 1)/(m - 1) internal vertices.

❑ Result – There are at most m^h leaves in an m-ary tree of height h.

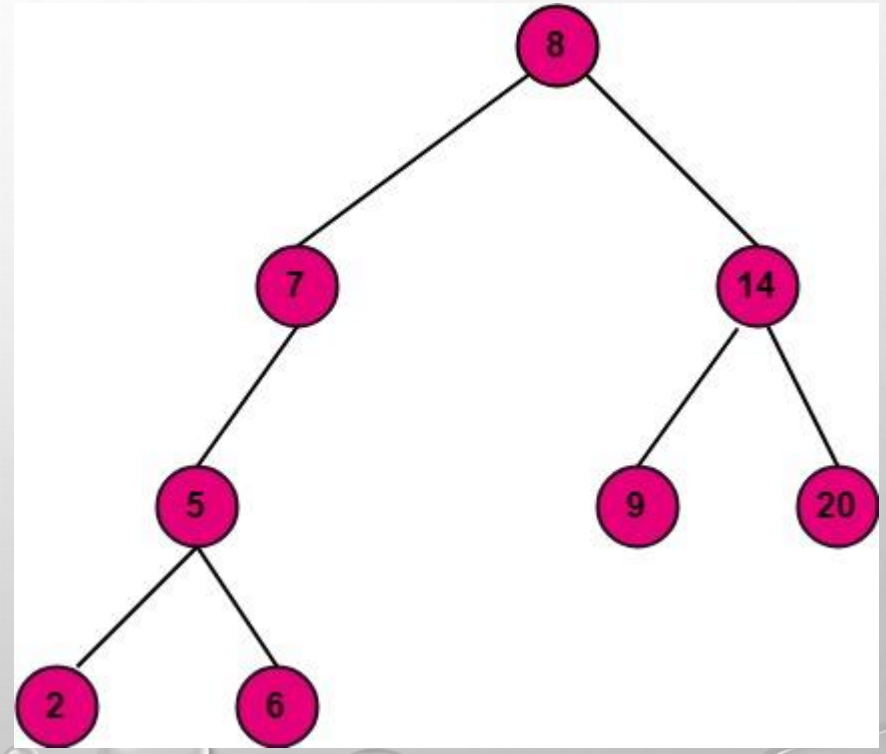
❑ Corollary – If an m-ary tree of height h has l leaves, then $h \geq \lceil \log_m^l \rceil$. If the m-ary tree is full and balanced, then $h = \lceil \log_m^l \rceil$.

- (ceiling function $\lceil x \rceil$ *is the smallest integer greater than or equal to* x).
- Balanced – all leaves are at levels h or h – 1.

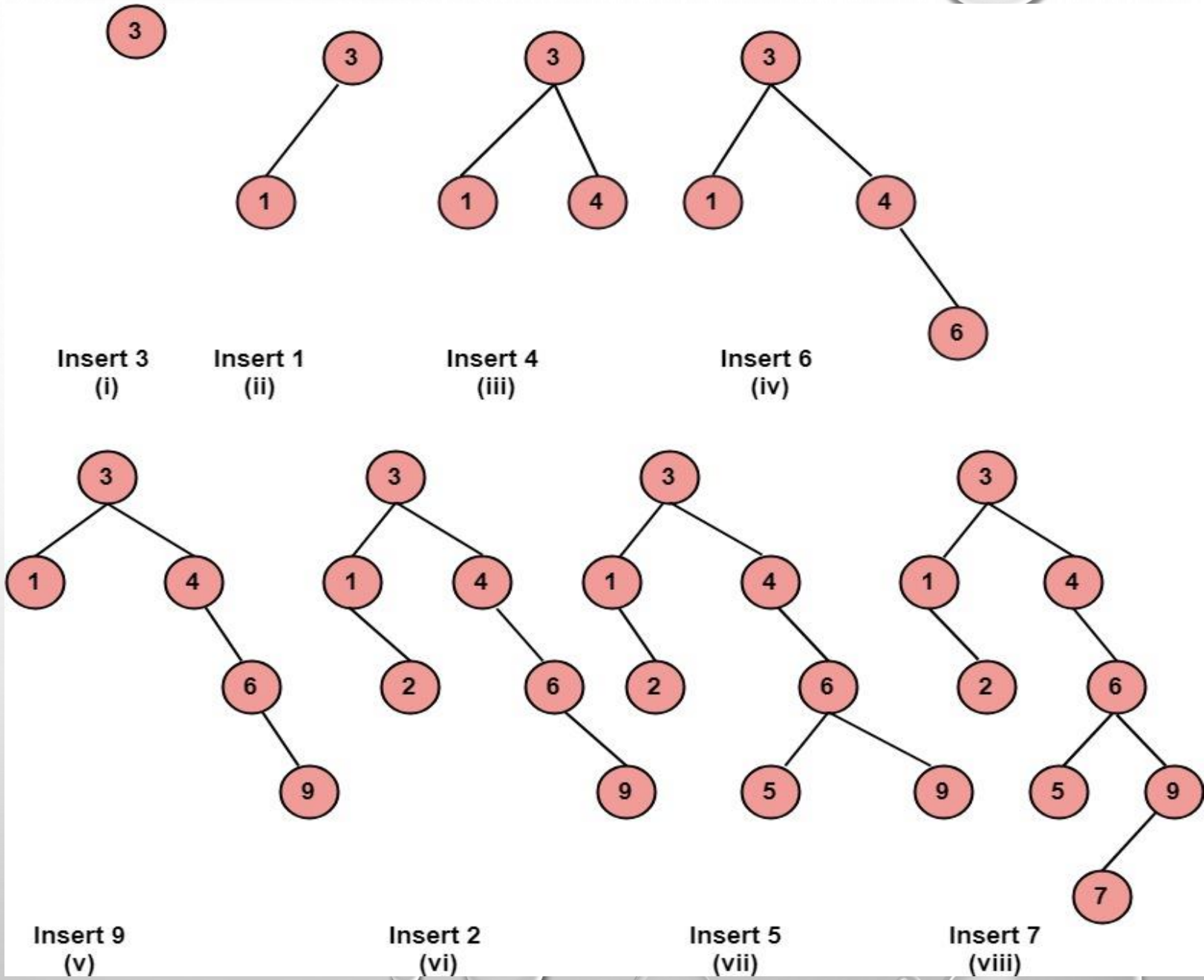
Binary Search Trees

- Binary search trees have the property that the node to the left contains a smaller value than the node pointing to it and the node to the right contains a larger value than the node pointing to it.

- ✓ It is not necessary that a node in a 'Binary Search Tree' point to the nodes whose value immediately precede and follow it



Example: Show the binary search tree after inserting 3, 1, 4, 6, 9, 2, 5, 7 into an initially empty binary search tree.



Prefix Codes

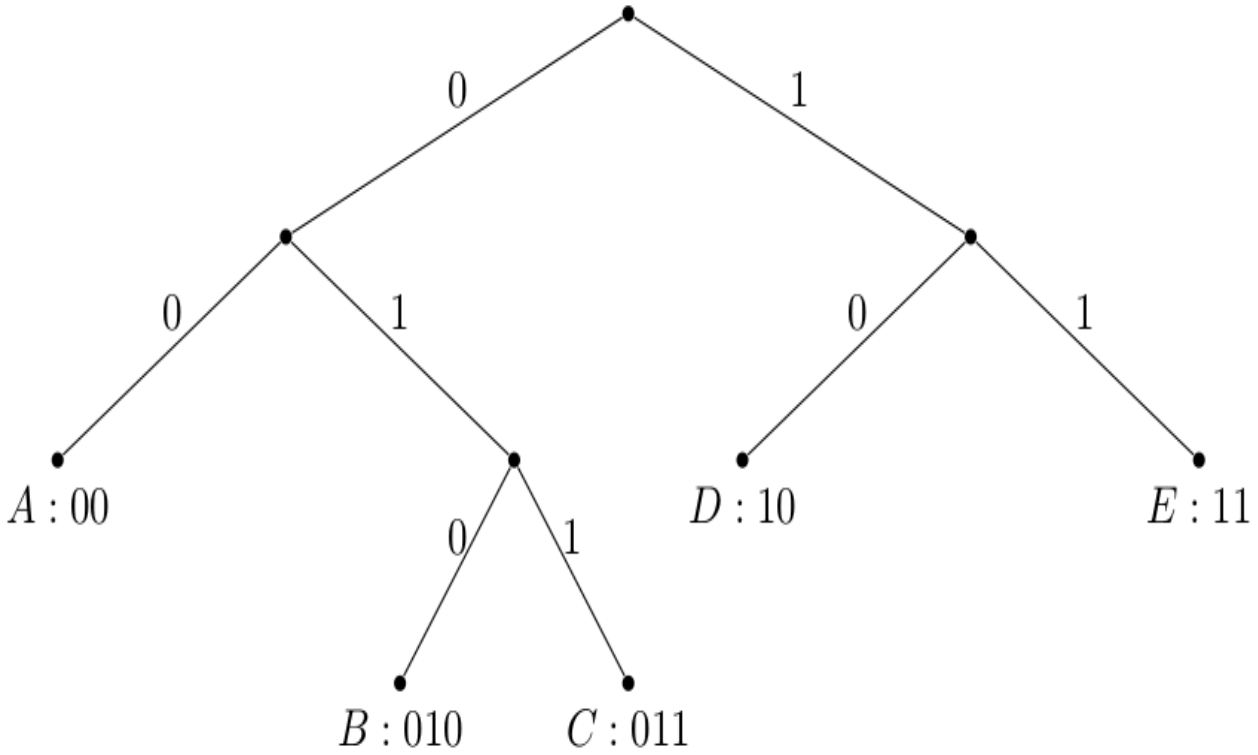
- A prefix code is a type of code system (typically a variable-length code) distinguished by its possession of the “prefix property”, which requires that there is no whole code word in the system that is a prefix (initial segment) of any other code word in the system. “

- Example – Consider the one to one binary code system {A: 00, B: 010, C: 011, D: 10, E: 11}
  A was coded as 00, while none of the codes of B, C, D, and E are started with 00. Using the same way, we check B, C, D, and E, and confirm that it is a prefix code system.

- If we change the above system a little – {A: 00, B: 010, C: 001, # 011 -> 001 , D: 10, E: 11}
  The new system is still one-to-one correspondence. However, it is no-longer a prefix code system, because the code of A 00 was shown as the prefix in the code of C which is 001.

- ✓ Note – We can always obtain a binary prefix code directly from a binary tree.

- For a given binary tree, we label the two edges incident from each branch node with 0 and 1. Assign to each leaf a sequence of 0's and 1's which is the sequence of labels of the edges in the path from the root to that leaf.



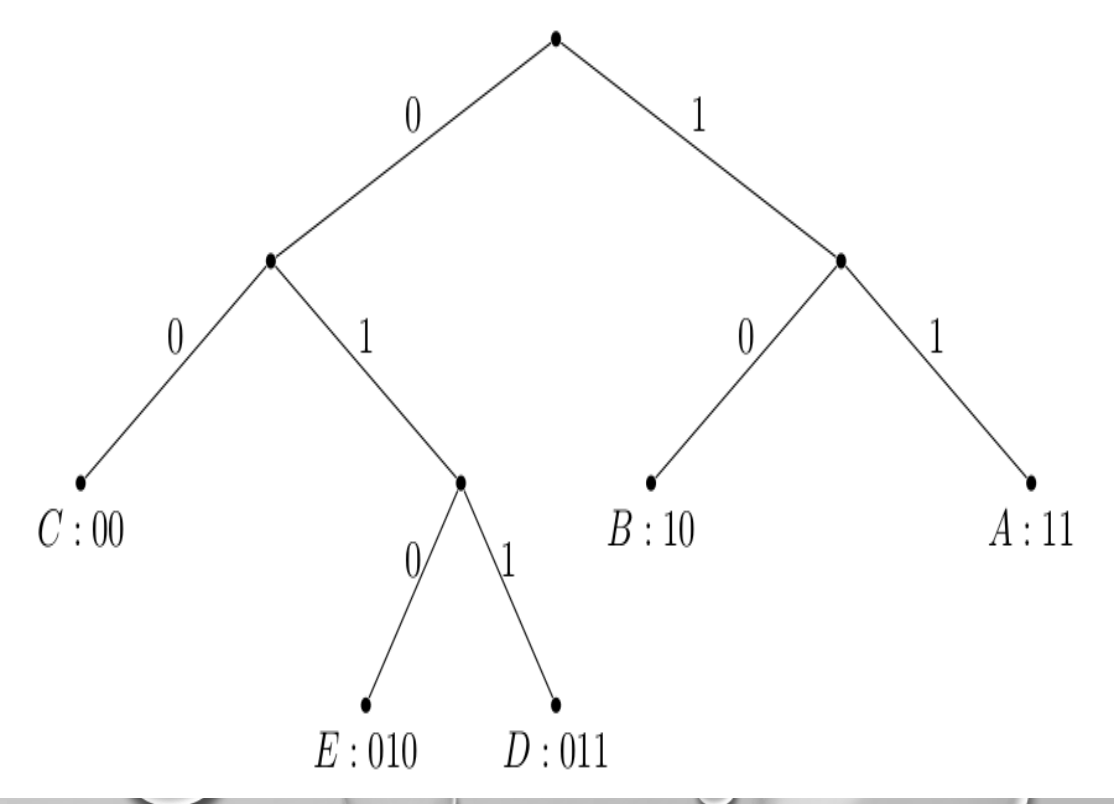
- ✓ Conversely, corresponding to a prefix code, we always obtain a binary tree, with the two edges incident from each of the branch nodes labeled with 0 and 1, such that the sequence of 0's and 1's assigned to the leaves are the sequences in the code.

Huffman Coding Tree

- Huffman coding tree is a prefix code tree. The specialty of Huffman tree compared to an ordinary prefix code tree is that it minimizes the probability of weighted mean of code length in the system.

- For example, we have a system “AAAAAABBBBCCCDDE”, the probabilities of letters are P(A) =1/3, P(B) = 4/15, P(C) = 1/5, P(D) = 2/15, P(D) = 2/15, P(E) = 1/15.

- Codes and Compression - Suppose that we have a 10,000 character data file that we wish to store. A binary code encodes each character as a binary string or codeword. We would like to find a binary code that encodes the file using as few bits as possible, i.e., compresses it as much as possible.
- In a *fixed-length code* each codeword has the same length. In a *variable-length code* codewords may have different lengths.



	a	b	c	d	e	f
Freq in '000s	45	13	12	16	9	5
a fixed-length	000	001	010	011	100	101
a variable-length	0	101	100	111	1101	1100

- ✓ The fixed length-code requires (45 + 13 + 12 + 16 + 9 + 5)* 3 = 300 bits to store the file.
- ✓ The variable-length code uses only (45*1 + 13*3 + 12*3 + 16*3 + 9*4 + 5*4) = 224 bits

Almost a 25% saving, lot of space..!!

- ❑ Huffman developed a nice greedy algorithm for producing a minimum cost (optimum) prefix code. The code that it produces is called a *Huffman code* .

- ***Huffman Tree gives that the prefix code we obtain is the optimum (lowest cost) prefix code.***

Problem Formulation

- Given a set of alphabets A = { a_1, a_2, … , a_n } with frequency distribution f(a_i). Find a binary prefix code C for A that minimizes the number of bits, i.e. minimize $B(C) = \sum_1^n f(a_i)L(c(a_i))$, which is needed to encode a message of $\sum_1^n f(a_i)$ characters, where c(a_i) is the codeword for encoding a_i, and L(c(a_i)) is the length of the codeword c(a_i).

  - Note that d(a_i), the depth of leaf a_i in tree T, is equal to the length, L(c(ai)) of the codeword in code C associated with that leaf. The sum $\sum$ f(a_i)d(a_i) is the *weighted external path length* of tree T

- Hence, the Huffman encoding problem is equivalent to the minimum-weight external path length problem: Given weights f(a_1),…., f(a_n), find a binary tree T with n leaves labelled a_1,…, a_n that has minimum weighted external path length.

Step 1: Pick two letters x, y from alphabet A with the smallest frequencies and create a subtree that has these two characters as leaves (greedy idea)

- Label the root of this subtree as z.

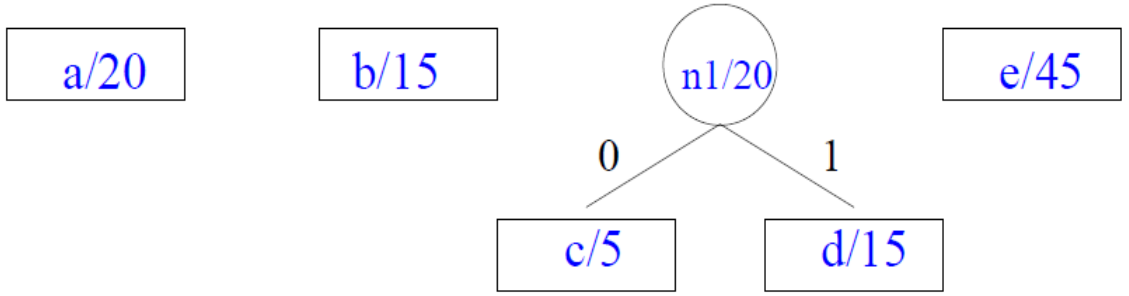
Step 2: Set frequency f(z) = f(x) + f(y). Remove x, y and add z creating new alphabet A' = A U {Z}- {x,y}. Therefore, |A'| = |A| - 1.

- Repeat this procedure, called *merge*, with new alphabet A' until an alphabet with only one symbol is left.

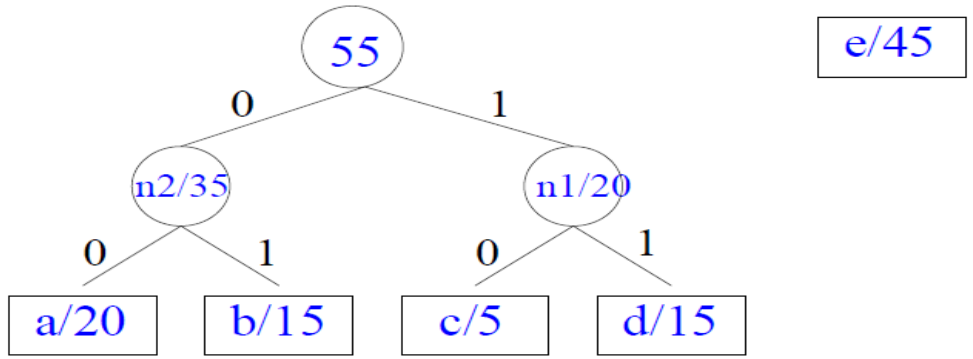
The resulting tree is the Huffman code.

Let $A = \{a/20, b/15, c/5, d/15, e/45\}$ be the alphabet and its frequency distribution.

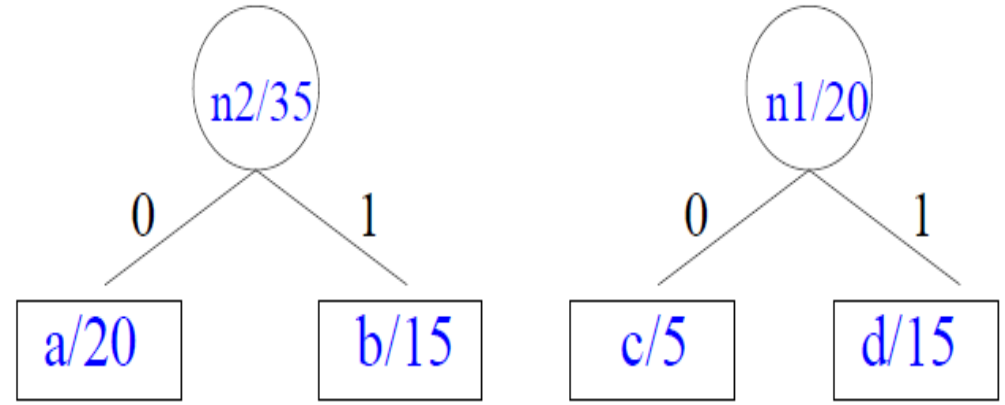
In the first step Huffman coding merges c and d.



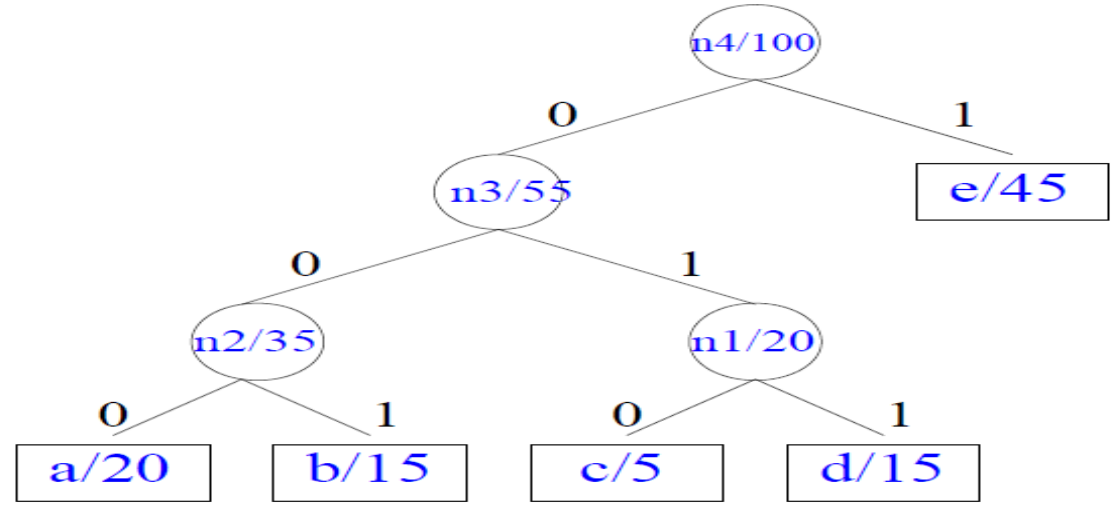
Alphabet is now $A_1 = \{a/20, b/15, n1/20, e/45\}$.



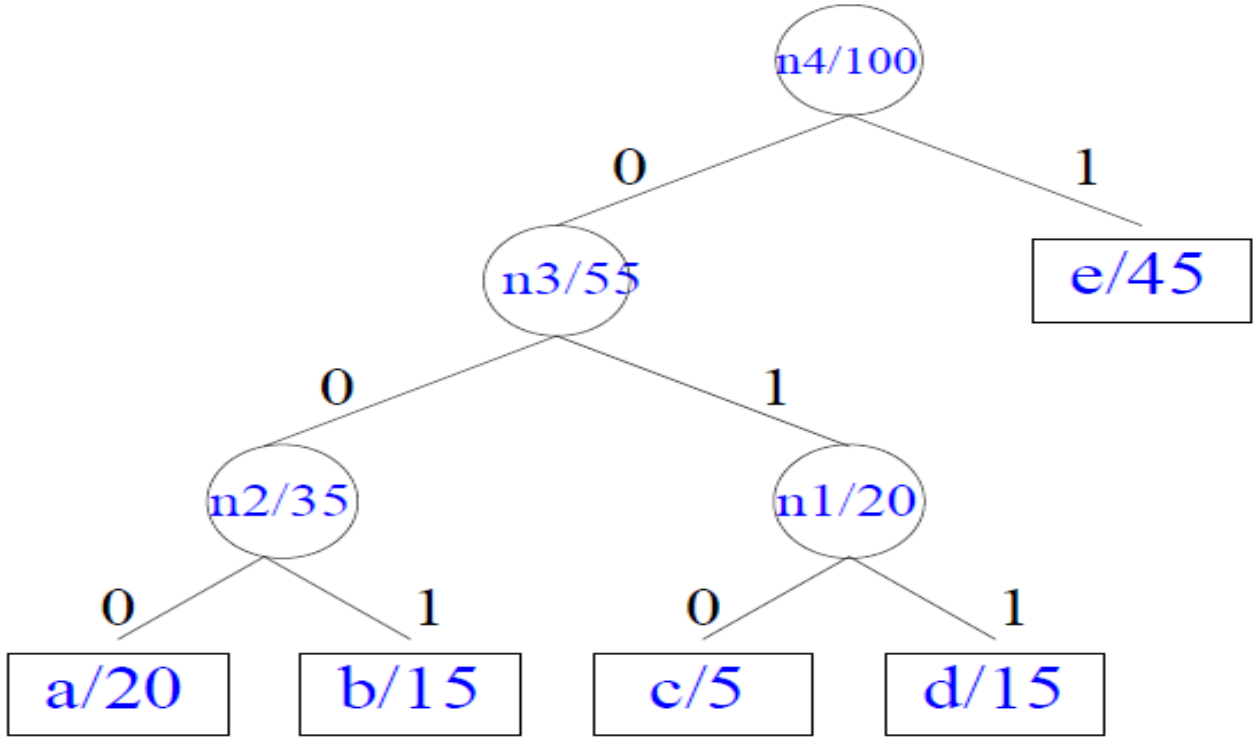
New alphabet is $A_3 = \{n3/55, e/45\}$.



New alphabet is $A_2 = \{n2/35, n1/20, e/45\}$.



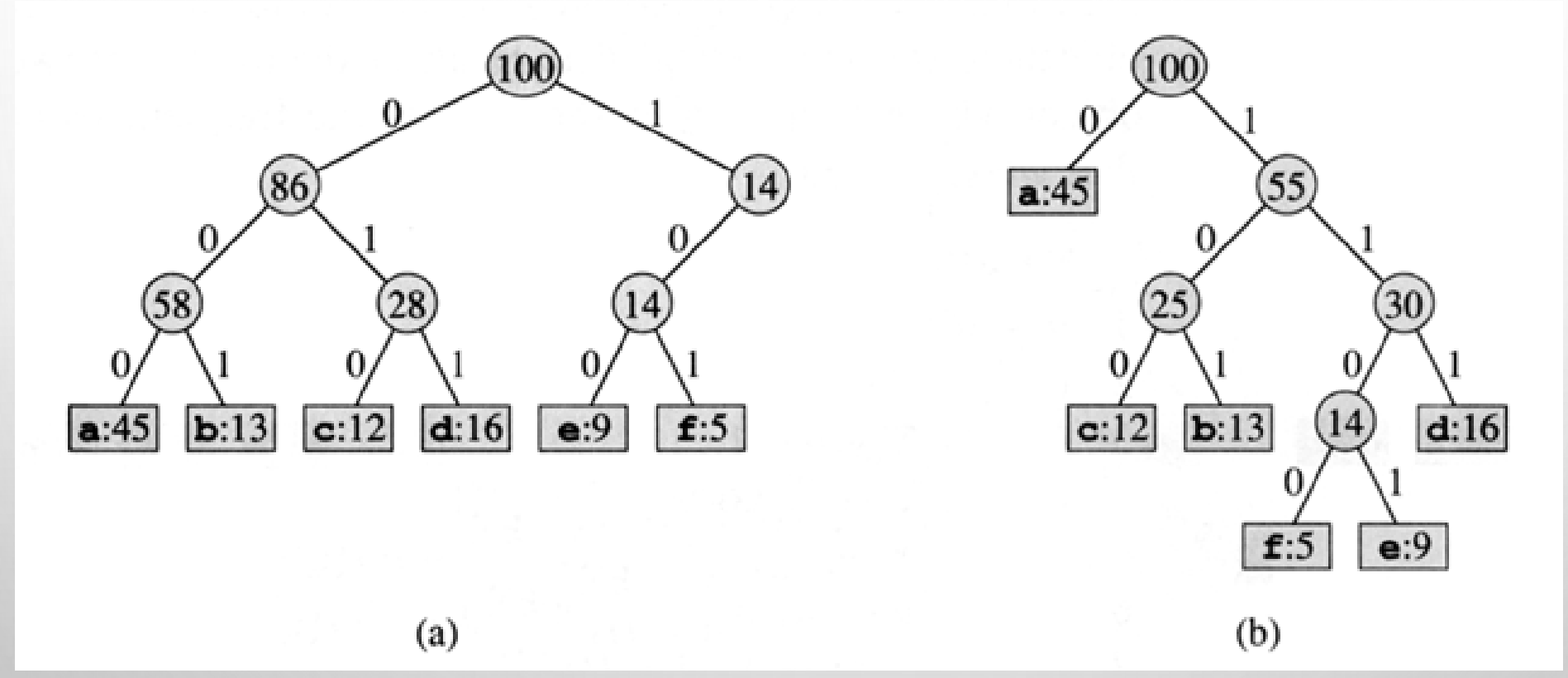
Huffman code is obtained from the Huffman tree.



Huffman code is

$a = 000, b = 001, c = 010, d = 011, e = 1.$

This is the optimum (minimum-cost) prefix code for this distribution.


100
0
1
86
14
0
1
0
58
28
14
0
1
0
1
0
1
a:45
b:13
c:12
d:16
e:9
f:5
(a)
100
0
1
a:45
55
0
1
25
30
0
1
0
1
c:12
b:13
14
d:16
0
1
f:5
e:9
(b)

Traversals of a Binary Tree

- Traversing a tree means visiting each node in a specified order. In computer science, tree traversal is a form of graph traversal and refers to the process of ***visiting*** /checking /updating ***each node*** in a tree data structure ***exactly once***.

- Linear data structures like arrays, stacks, queues, and linked list have only one way to read the data. But a hierarchical data structure, like a tree, can be traversed in different ways.

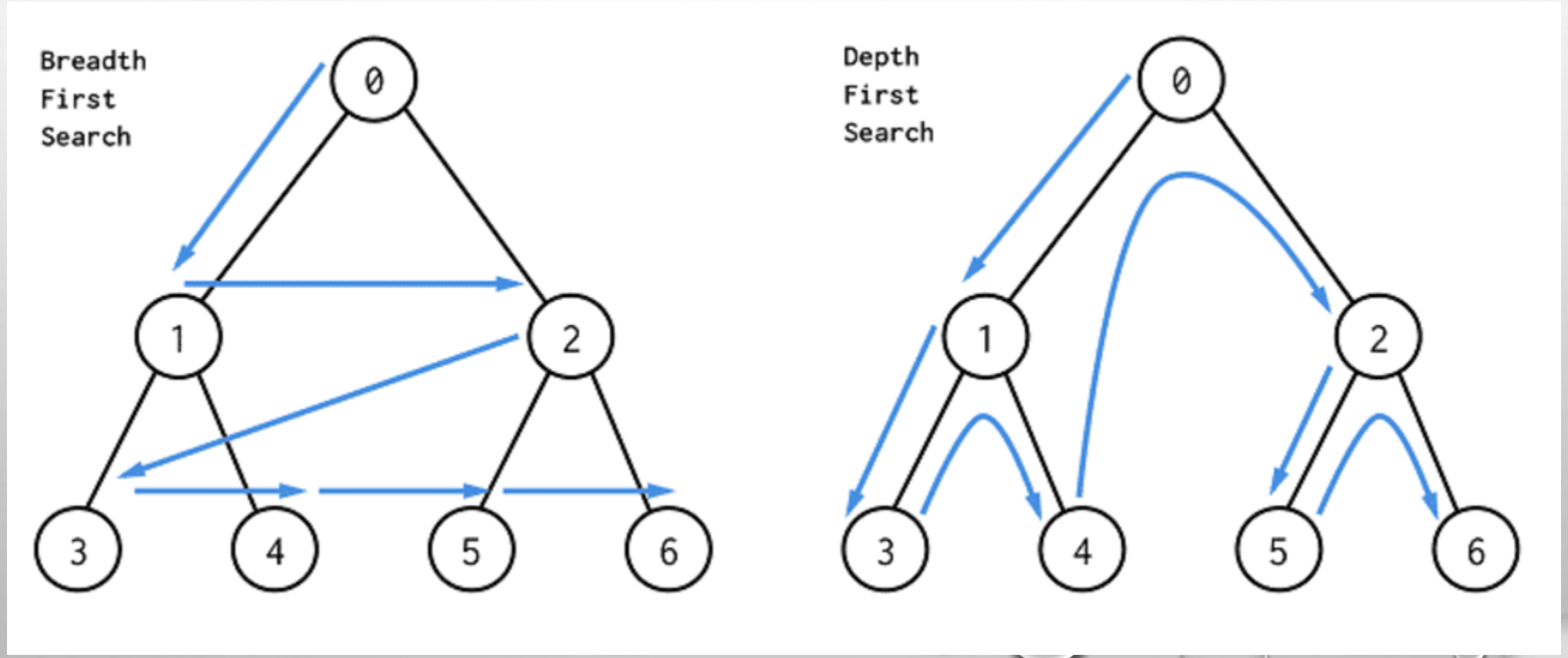
- Such traversals are classified by the order in which the nodes are visited. The algorithms are described for a binary tree, but they may be generalized to other trees as well

- There are generally two types of traversal – 1. Breadth First Traversal & 2. Depth First Traversal.

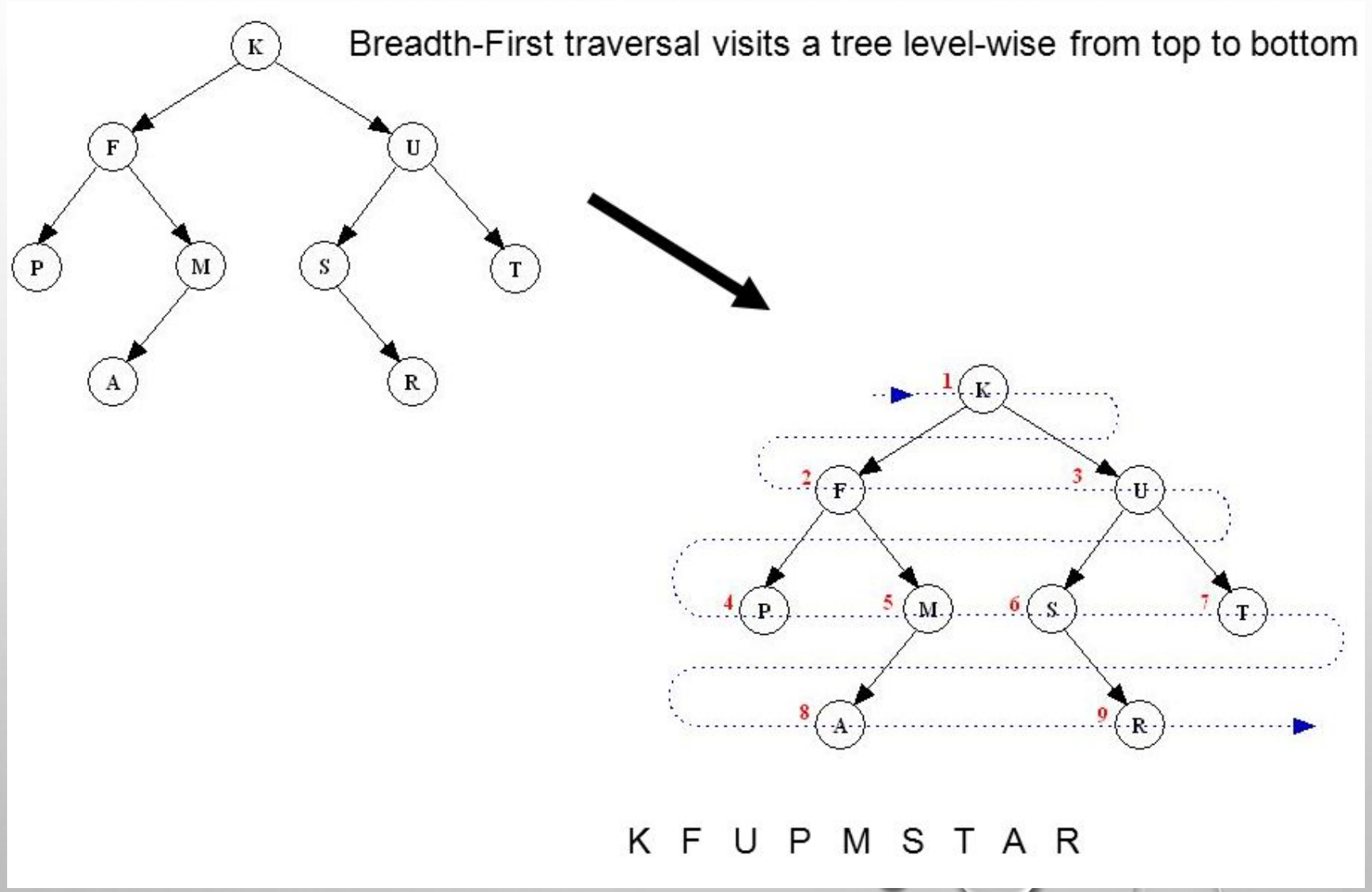
- There are three variants for depth first traverse a tree – Pre-order, In-order, and Post-order.

Tree Traversal

- Visiting a vertex
- Exploration of vertex



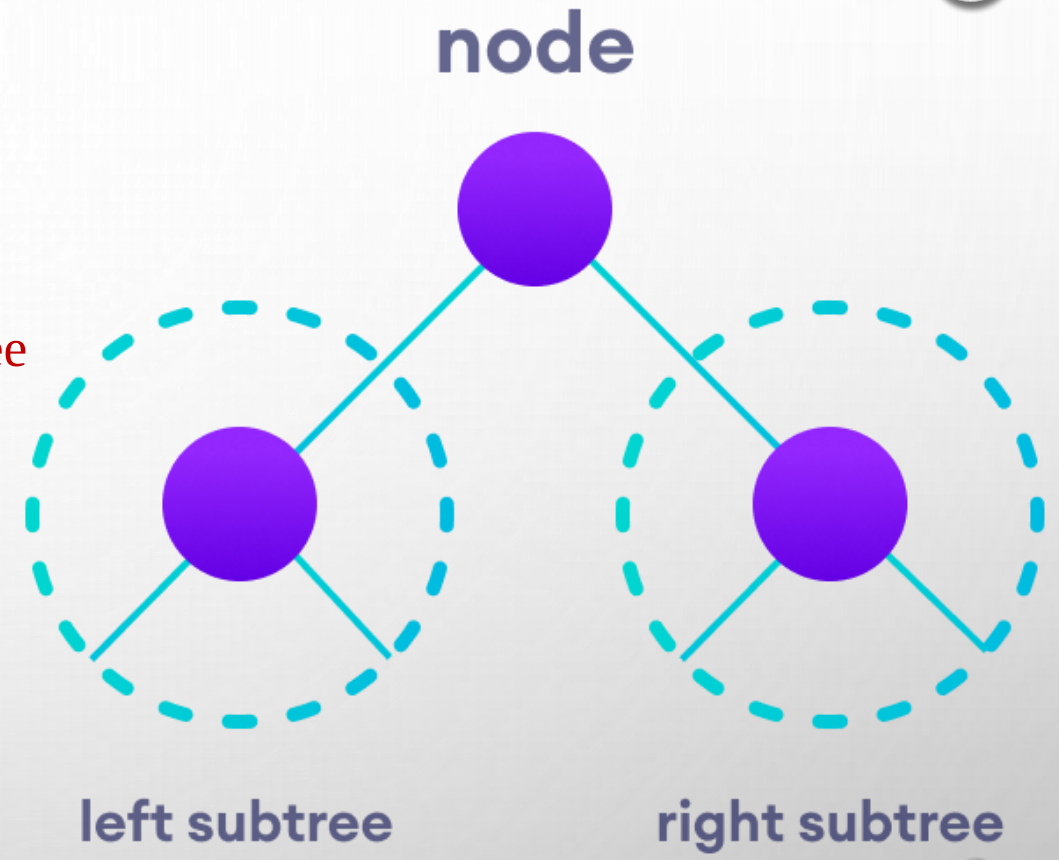
Breadth First Search



Depth First Search

- Visit all the nodes in the left subtree, visit the root node and visit all the nodes in the right subtree as well.
- Depending on the order in which we do this, there can be three types of traversal.

- In-order Traversal
- Pre-order Traversal
- Post-order Traversal



Depth First Search

- In-order – ***Left → Root → Right***
- The output of in-order traversal of this tree will be −

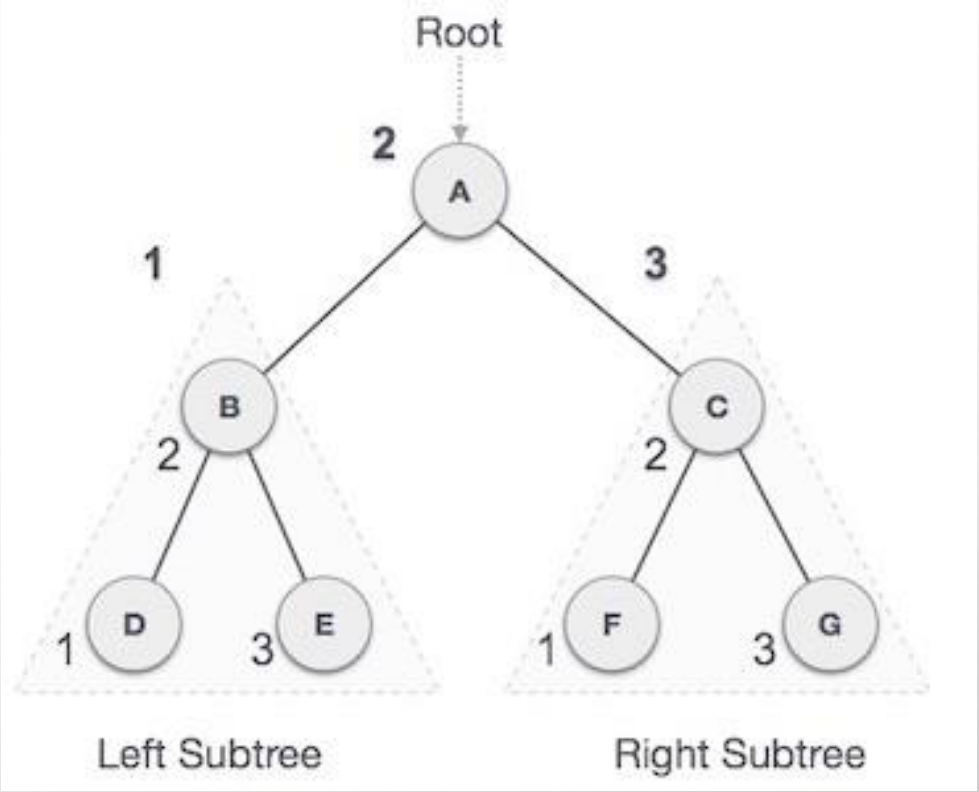
$$D \rightarrow B \rightarrow E \rightarrow A \rightarrow F \rightarrow C \rightarrow G$$

- Pre-order – ***Root → Left → Right***
- The output of pre-order traversal of this tree will be −

$$A \rightarrow B \rightarrow D \rightarrow E \rightarrow C \rightarrow F \rightarrow G$$

- Post-order – ***Left → Right → Root***
- The output of post-order traversal of this tree will be −

$$D \rightarrow E \rightarrow B \rightarrow F \rightarrow G \rightarrow C \rightarrow A$$

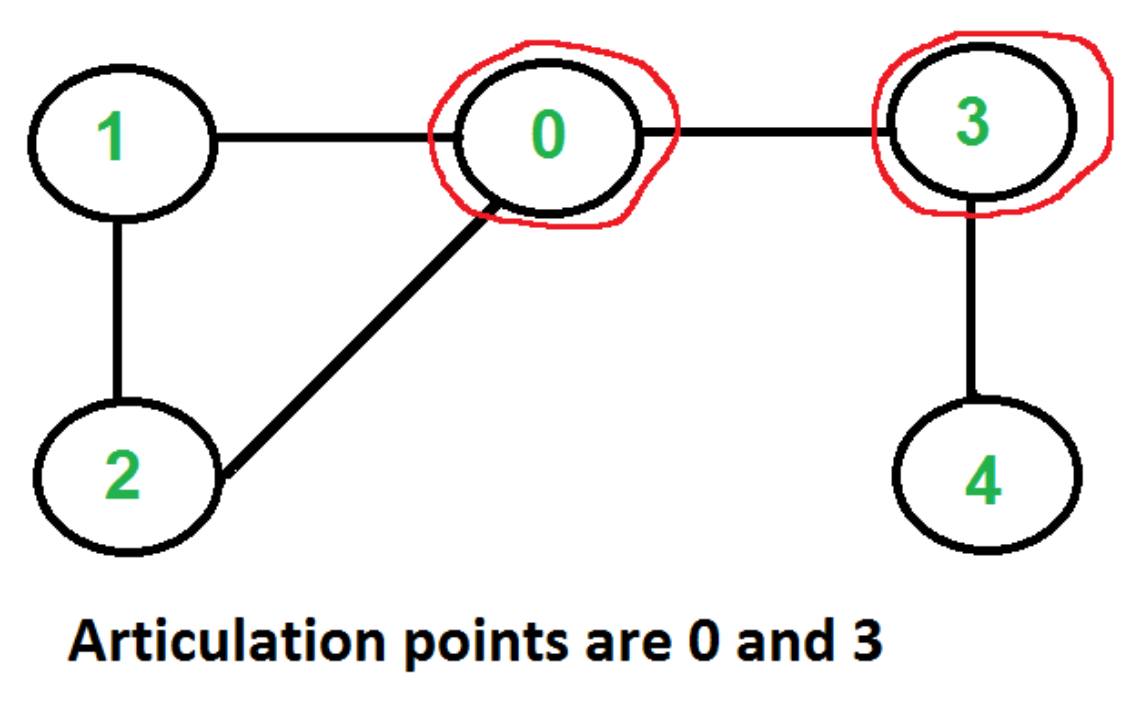


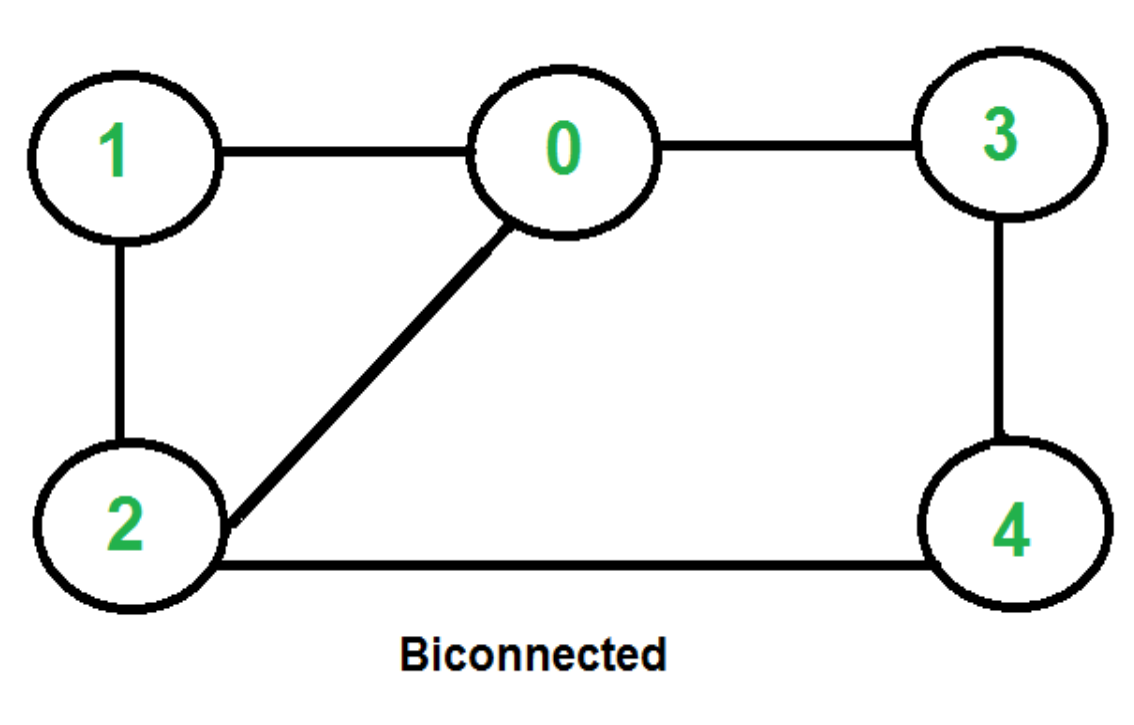
Articulation Points & Bi – Connected Graph

- Let G = (V, E) be a connected, undirected graph. Then

- An ***articulation point or Cut Vertex*** is a vertex v of G such that the deletion of v, together with all edges incident on v, produces a graph, G', that has at least two connected components.

- A ***biconnected graph*** is a connected graph that has no articulation points





Method of Finding all Articulation Points

- **Brute-force approach**: one by one remove all vertices and see if removal of a vertex causes the graph to disconnect:

```
for every vertex v, do
{
   remove v from graph
  Check if the graph remains connected (use BFS or DFS)
  If
  graph is disconnected, add v to AP list
 else
  add v back to the graph
}
```

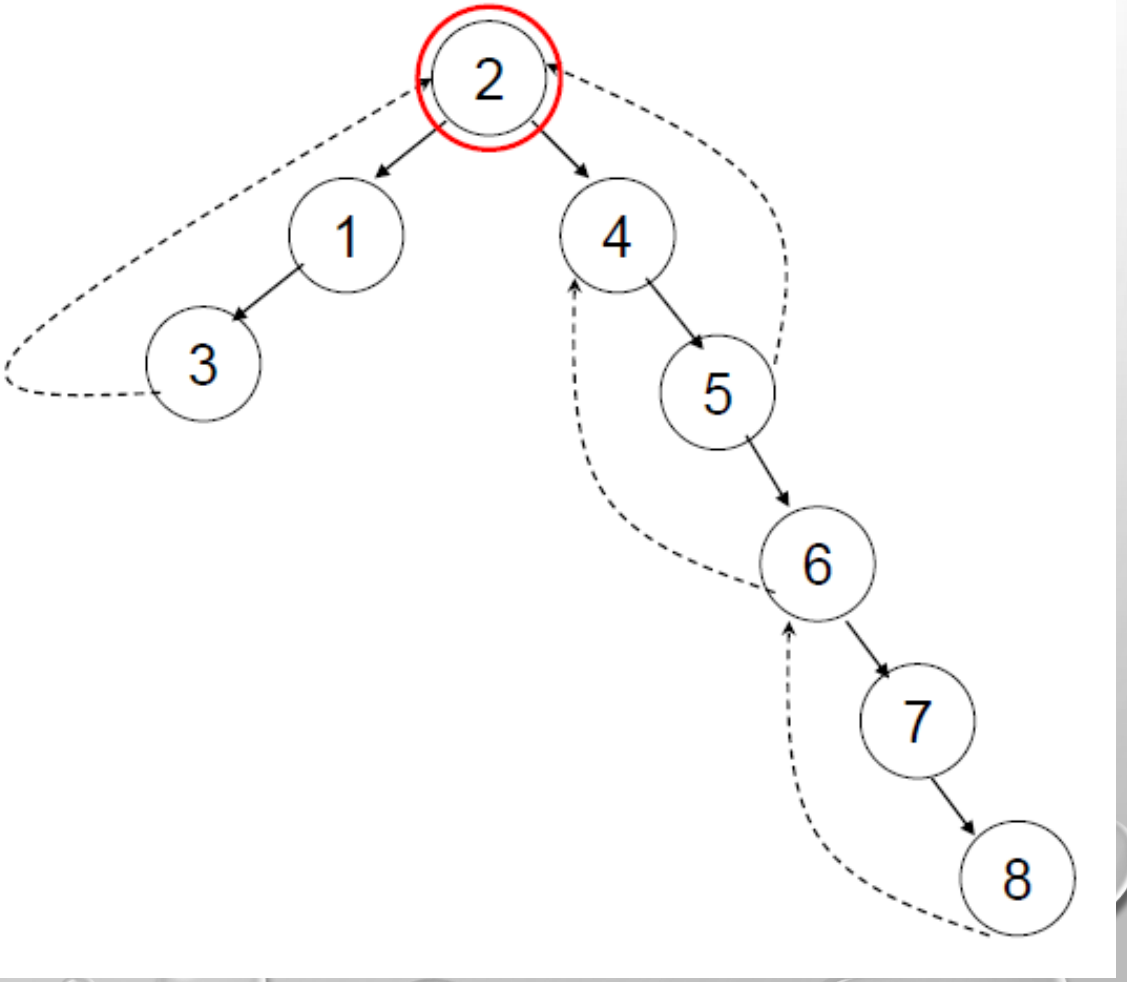
Method of Finding all Articulation Points

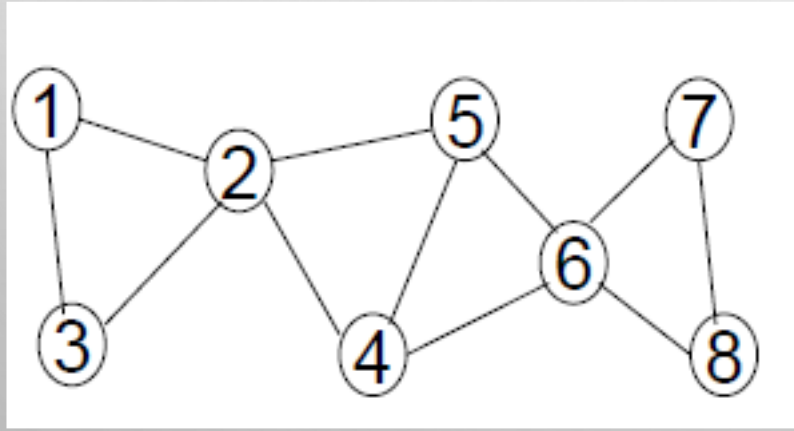
- **DFS- based-approach**: We have following properties:

1. The root of a DFS-tree is an articulation point if and only if it has at least two children.

2. A non root vertex v of a DFS-tree is an articulation point of G if and only if has a child s such that there is no back edge from s or any descendant of s to a proper ancestor of v.

3. Leaves of a DFS-tree are never articulation points





Bridge or Cut - Edge

❑ **A *bridge or Cut Edge* of G is an edge whose removal disconnects G.**

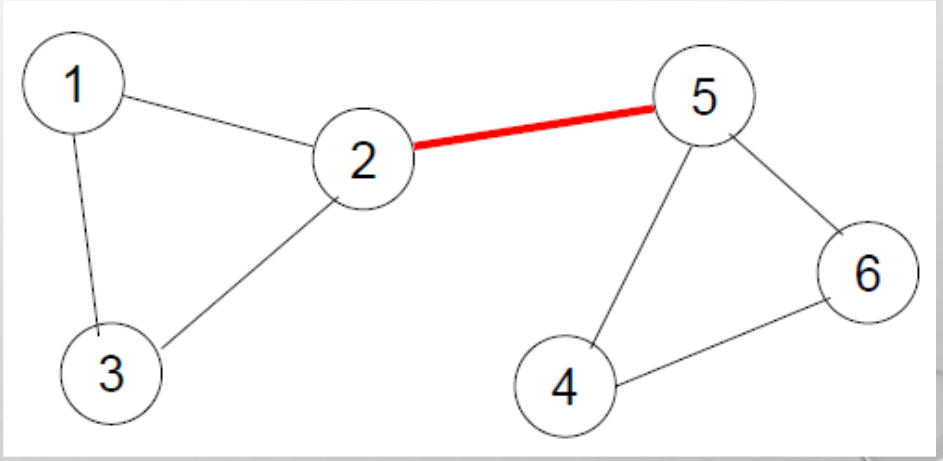
❑ An edge of G is a bridge if and only if it does not lie on any simple cycle of G.

- **Brute-force approach**: one by one remove all edges and see if removal of an edge causes the graph to disconnect.

- **DFS- approach**:

✓ If some vertex u has a back edge pointing to it, then no edge below u in the DFS tree can be a bridge. The reason is that each back edge gives us a cycle, and no edge that is a member of a cycle can be a bridge.

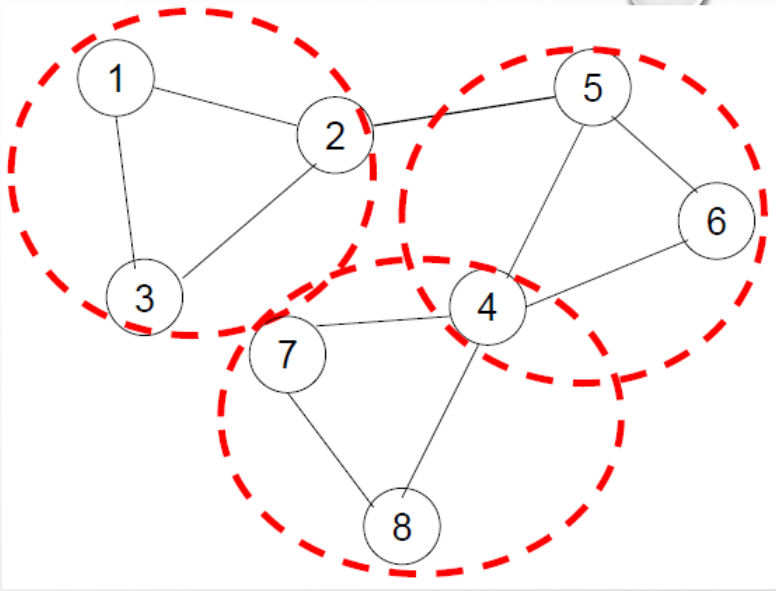
✓ If we have a vertex v whose parent in the DFS tree is u, and no ancestor of v has a back edge pointing to it, then (u, v) is a bridge.

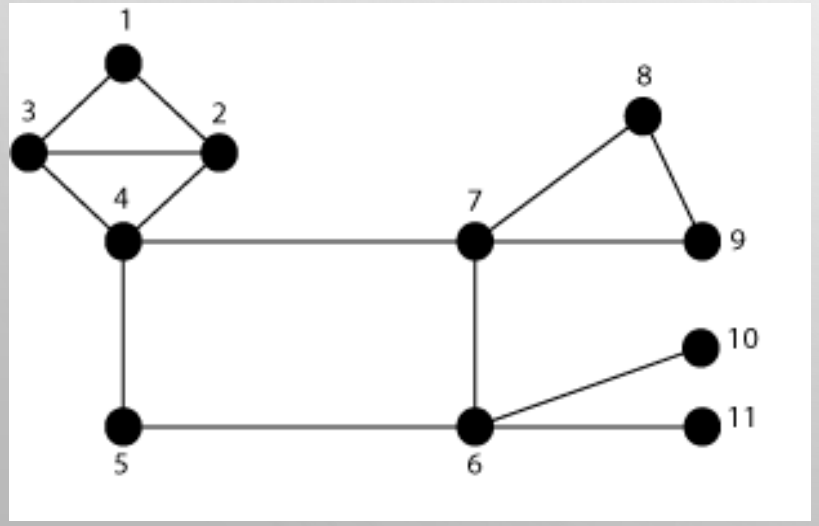


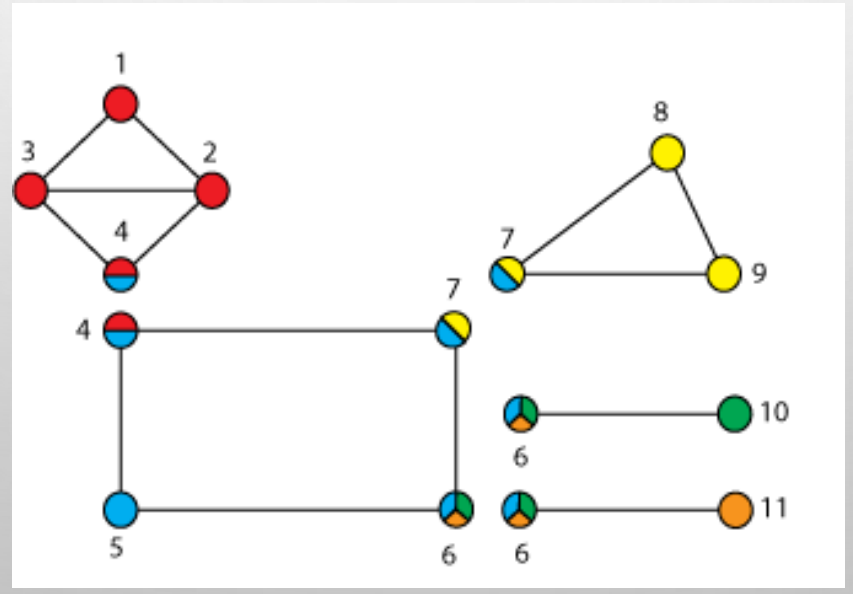
Bi – Connected Component

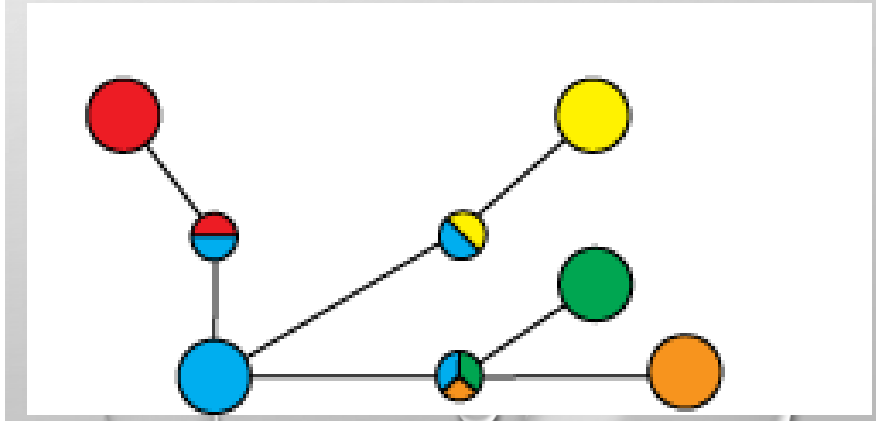
- A *biconnected component* of a connected undirected graph is a *maximal biconnected subgraph*, *H*, of G. By maximal, we mean that G contains no other subgraph that is both biconnected and properly contains *H*.

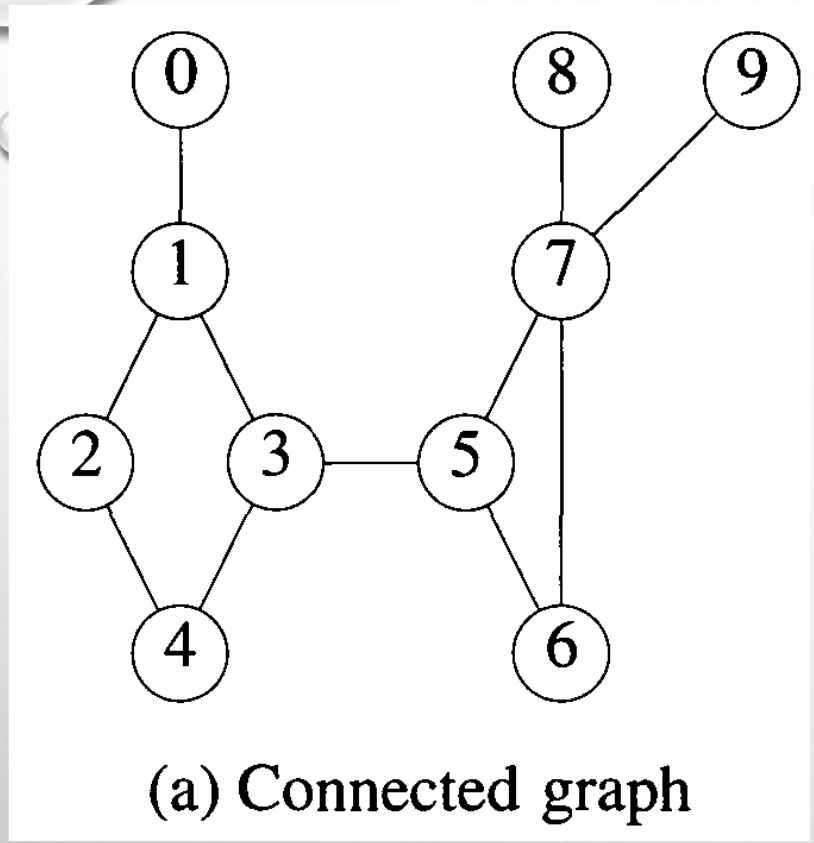
- A ***Bi - connected component*** of G is a maximal set of edges such that any two edges in the set lie on a common simple cycle.



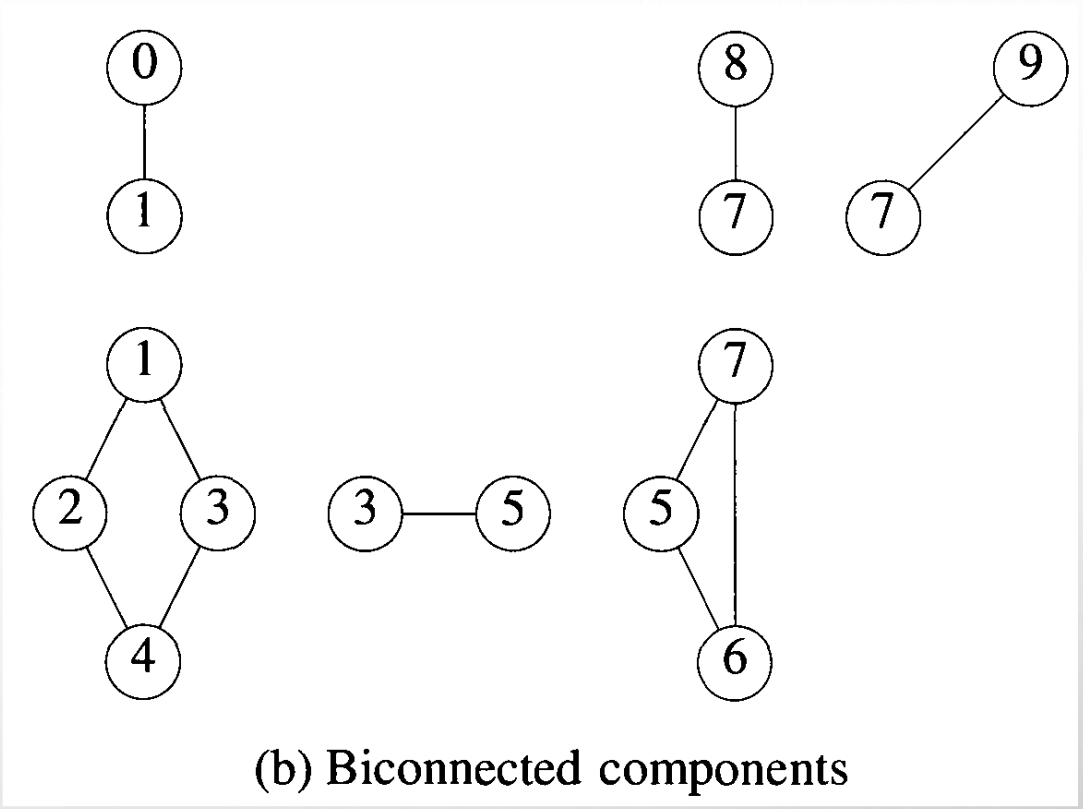








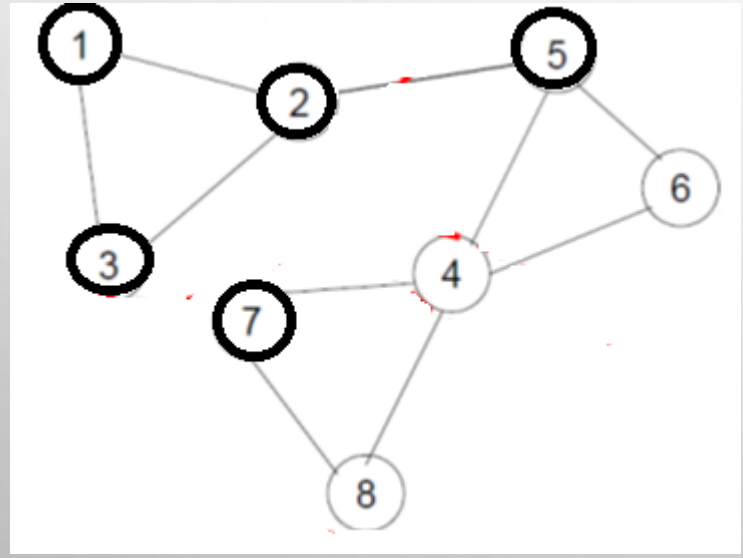
(a) Connected graph

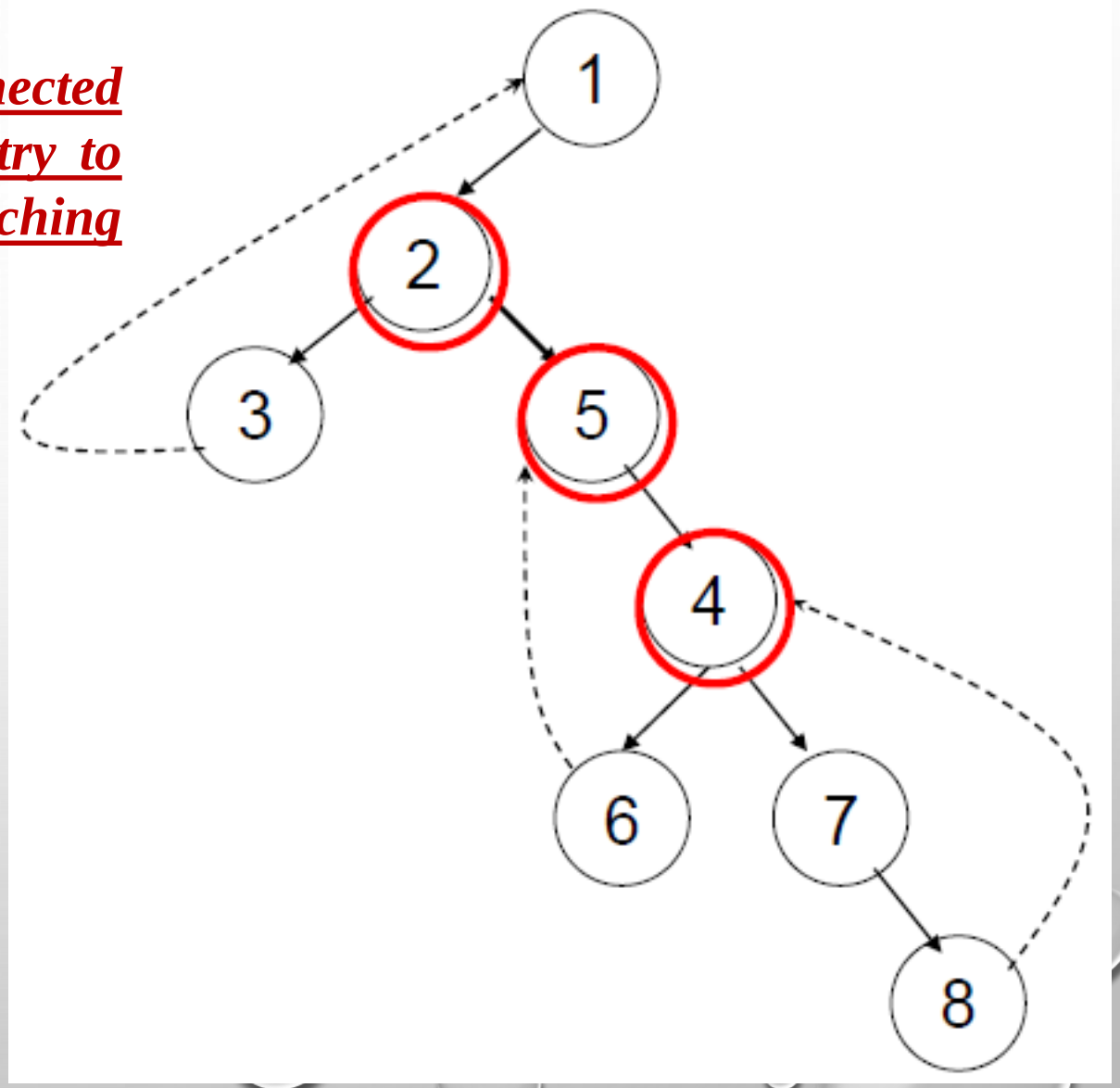


(b) Biconnected components

- Two biconnected components of the same graph have no more than one vertex in common. This means that no edge can be in two or more biconnected components of a graph. Hence, the biconnected components of G partition the edges of G.

- ***The common vertex of several biconnected components is an articulation point - We will try to identify the biconnected components while searching for articulation points***





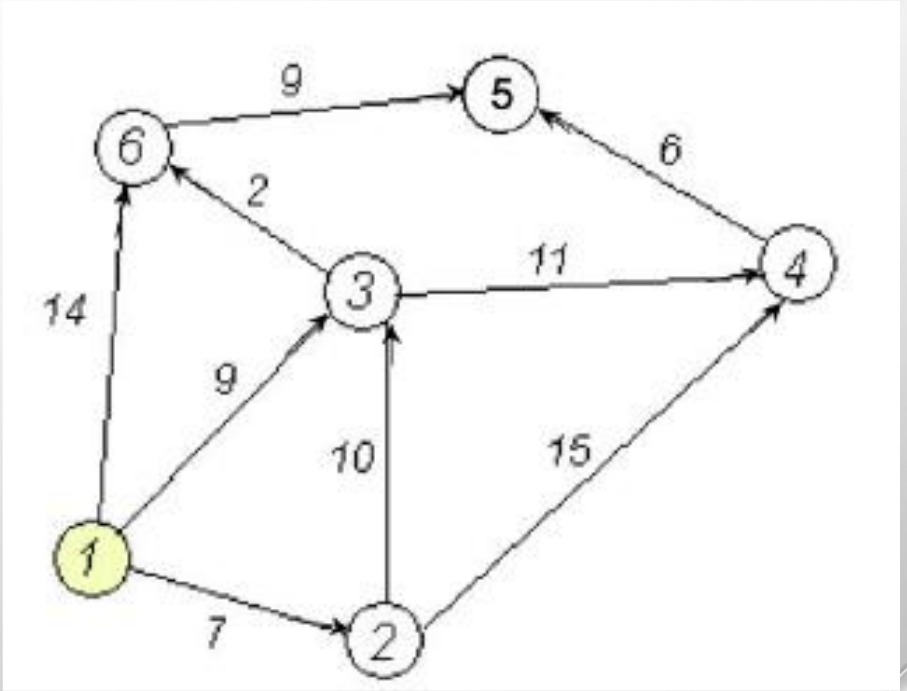
Dijkstra's algorithm – Shortest Path Algorithm

- Suppose we want to find a shortest path from a given node s to other nodes in a network. Dijkstra's algorithm solves such a problem. It finds the shortest path from a given node s to all other nodes in the network. Node s is called a starting node or an initial node

- Notations -

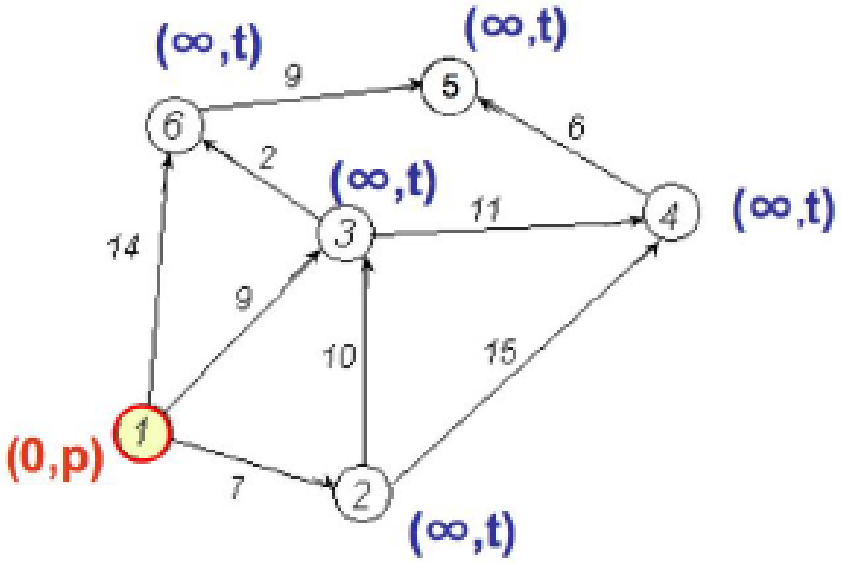
- d - distance value of a node
- p or t - status label of a node, where p stand for permanent and t stands for temporary
- c_{ij} is the cost of traversing link (i, j) as given by the problem

The state of a node n is the ordered pair of its distance value d_n and its status label.

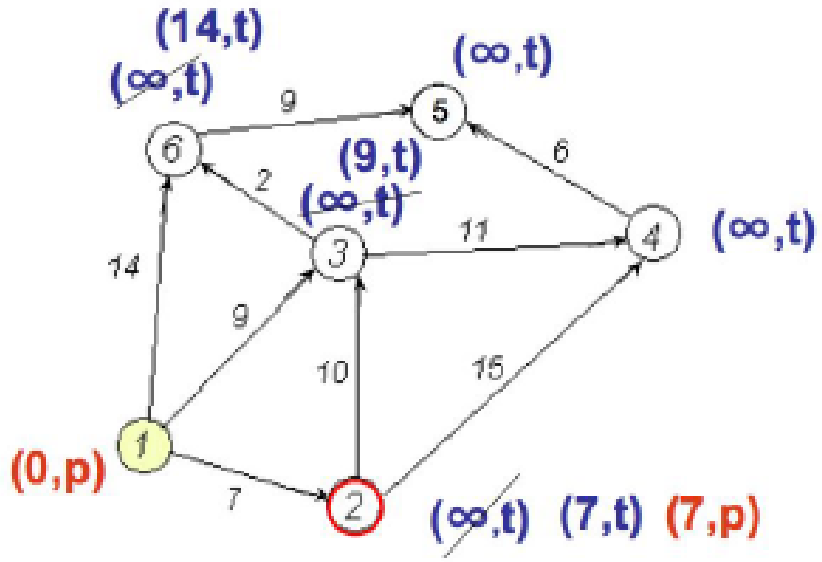


- Dijkstra's Algorithm Example : We want to find the shortest path from node 1 to all other nodes using Dijkstra's algorithm.

- Node 1 is designated as the current node
- The state of node 1 is $(0, p)$
- Every other node has state (∞, t)



- Nodes 2, 3,and 6 can be reached from the current node 1
- Update distance values for these nodes

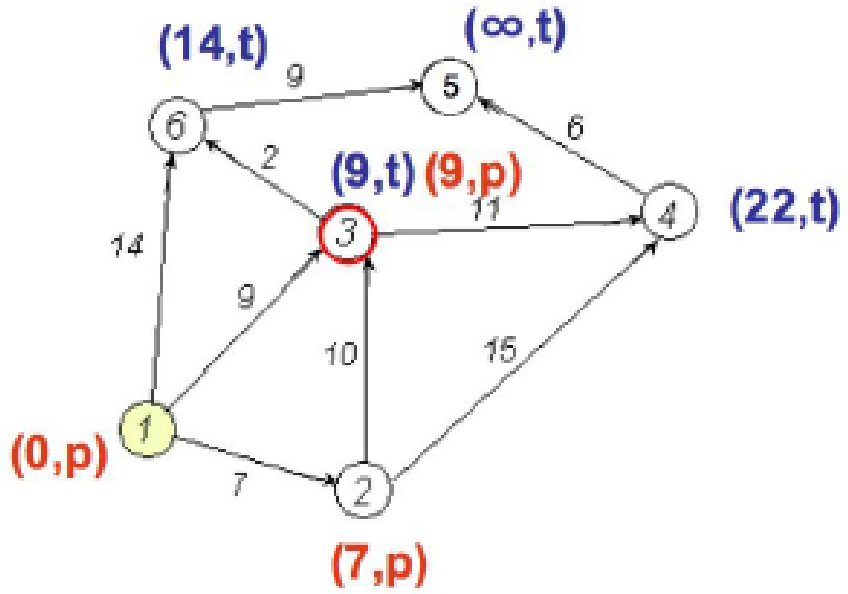


$$d_2 = \min\{\infty, 0 + 7\} = 7$$
$$d_3 = \min\{\infty, 0 + 9\} = 9$$
$$d_6 = \min\{\infty, 0 + 14\} = 14$$

- Now, among the nodes 2, 3, and 6, node 2 has the smallest distance value
- The status label of node 2 changes to permanent, so its state is $(7, p)$, while the status of 3 and 6 remains temporary
- Node 2 becomes the current node

- Nodes 3 and 4 can be reached from the current node 2
- Update distance values for these nodes

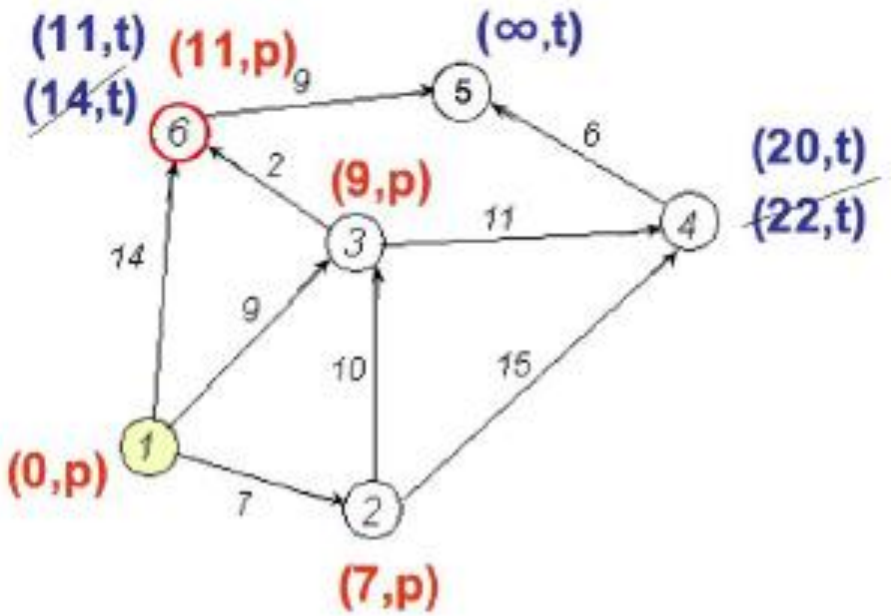
$$d_3 = \min\{9, 7 + 10\} = 9$$
$$d_6 = \min\{\infty, 7 + 15\} = 22$$



- Now, between the nodes 3 and 4 node 3 has the smallest distance value
- The status label of node 3 changes to permanent, while the status of 6 remains temporary
- Node 3 becomes the current node

- Nodes 6 and 4 can be reached from the current node 3
- Update distance values for them

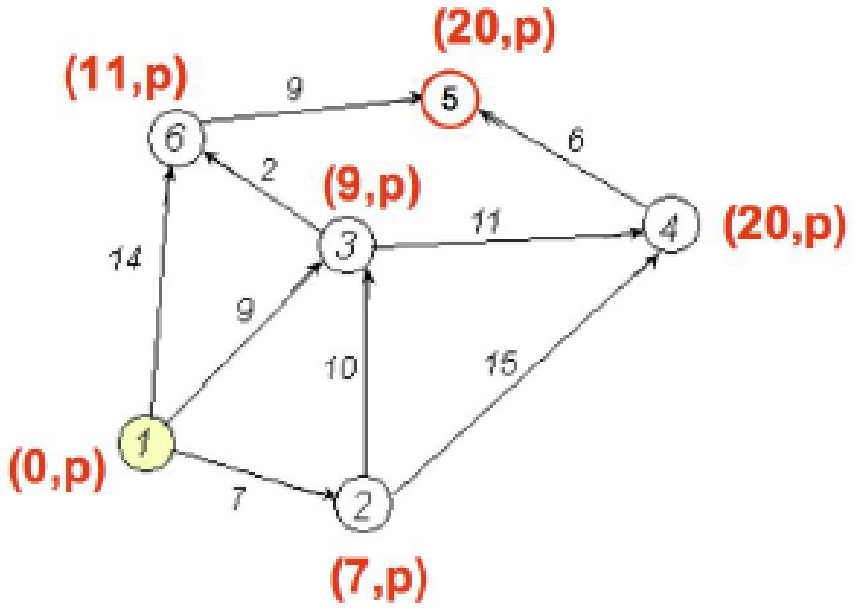
$$d_4 = \min\{22, 9 + 11\} = 20$$
$$d_6 = \min\{14, 9 + 2\} = 11$$



- Now, between the nodes 6 and 4 node 6 has the smallest distance value
- The status label of node 6 changes to permanent, while the status of 4 remains temporary
- Node 6 becomes the current node

- Node 5 can be reached from the current node 6
- Update distance value for node 5

$$d_5 = \min\{\infty, 11 + 9\} = 20$$



- Now, node 5 is the only candidate, so its status changes to permanent

- Node 5 becomes the current node

From node 5 we cannot reach any other node. Hence, node 4 gets permanently labeled and we are done.

Thank You