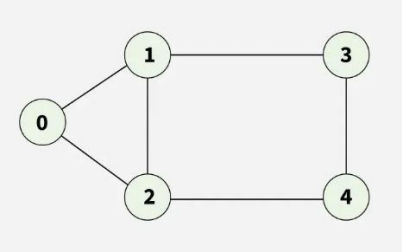
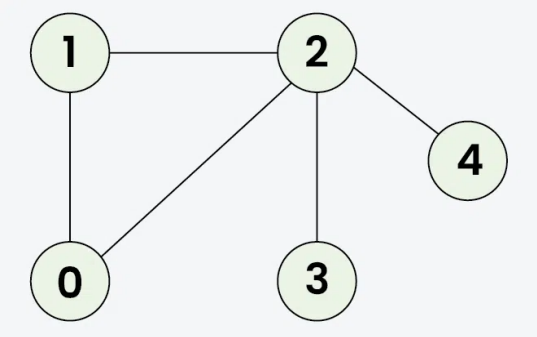
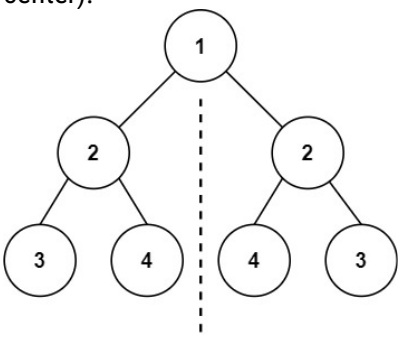
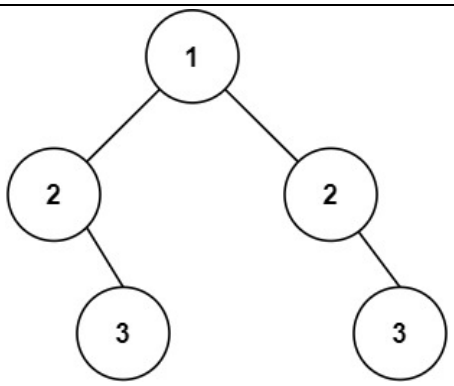
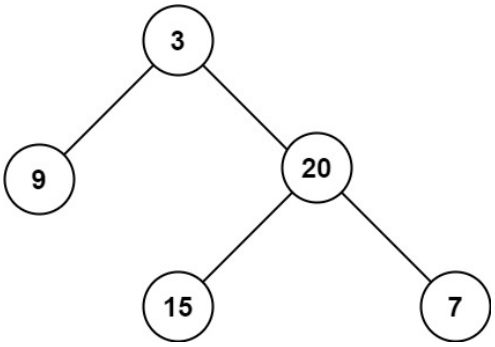
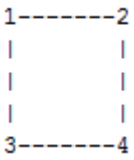
Program	B. Tech	Semester-5
Type of Course	Open Elective	
Prerequisite	Any programming language, Data Structure	
Course Objective	To understand the basics concept of Graph theory and implement a relative Programs.	

Teaching Scheme				Examination Scheme				
Lecture	Tutorial	Lab	Credit	Theory Marks		Practical Marks		Total Marks
				SEE	CIA	SEE	CIA	
3	0	2	4	70	30	25	25	150

1	Perform a following Matrix problem. 1. Take an input into 2-D array and display it on the screen 2. Take two 2-D array and perform addition and multiplication between them 3. Take 2-D array and perform a transportation of the matrix with in the same matrix.
2	Prob.1 Given the adjacency list and number of vertices and edges of a graph, the task is to represent the adjacency list for a directed graph. input: V = 3, edges[][]= {{0, 1}, {1, 2} {2, 0}} Output: 0 -> 1 1 -> 2 2 -> 0 input: V = 4, edges[][] = {{0, 1}, {1, 2}, {1, 3}, {2, 3}, {3, 0}} Output: 0 -> 1 1 -> 2 3 2 -> 3 3 -> 0 Prob. 2 Perform a Breadth First Search (BFS) traversal starting from vertex 0, visiting vertices from left to right according to the adjacency list, and return a list containing the BFS traversal of the graph. Input: adj[][] = [[1,2], [0,2,3], [0,1,4], [1,4], [2,3]]  Output: [0, 1, 2, 3, 4] Explanation: Starting from 0, the BFS traversal will follow these steps: Visit 0 ⟶ Output: [0] Visit 1 (first neighbor of 0) ⟶ Output: [0, 1] Visit 2 (next neighbor of 0) ⟶ Output: [0, 1, 2] Visit 3 (next neighbor of 1) ⟶ Output: [0, 1, 2, 3] Visit 4 (neighbor of 2) ⟶ Final Output: [0, 1, 2, 3, 4]

3	Prob. 1 Perform a Depth First Search (DFS) traversal starting from vertex 0, visiting vertices from left to right according to the adjacency list, and return a list containing the BFS traversal of the graph. Input: adj = [[1, 2], [0, 2], [0, 1, 3, 4], [2], [2]]  Output: [0 1 2 3 4] Explanation: The source vertex s is 0. We visit it first, and then we visit an adjacent. Start at 0: Mark as visited. Output: 0 Move to 1: Mark as visited. Output: 1 Move to 2: Mark as visited. Output: 2 Move to 3: Mark as visited. Output: 3 (backtrack to 2) Move to 4: Mark as visited. Output: 4 (backtrack to 2, then backtrack to 1, then to 0)
4	Prob. 1 In a social network of N people, some of them are directly connected as friends. If person A is a friend of person B, and person B is a friend of person C, then A, B, and C are all part of the same friend circle (a connected component in graph terms). You are given an undirected graph represented by an adjacency matrix of size N x N, where matrix[i][j] = 1 indicates a direct friendship between person i and person j. Your task is to determine the total number of friend circles. Input: 4 // total N people 1 1 0 0 // adj. Matrix 1 1 0 0 0 0 1 1 0 0 1 1 Output: 2
5	Prob. 1 Given the root of a binary tree, check whether it is a mirror of itself (i.e., symmetric around its center).  Input: root = [1,2,2,3,4,4,3] Output: true

	 Input: root = [1,2,2,null,3,null,3] Output: false
6	Prob. 1 Given the root of a binary tree, return its maximum depth. A binary tree's maximum depth is the number of nodes along the longest path from the root node down to the farthest leaf node.  Input: root = [3,9,20,null,null,15,7] Output: 3 Prob. 2 Identify all pendent (leaf) vertices in a given tree.
7	Prob. 1 Implement Kruskal's algorithm to find the Minimum Spanning Tree
8	Prob. 1 Given a undirected Graph of N vertices 1 to N and M edges in form of 2D array arr[][] whose every row consists of two numbers X and Y which denotes that there is an edge between X and Y, the task is to write program to create Adjacency Matrix of the given Graph. Input: N = 5, M = 4, arr[][] = { { 1, 2 }, { 2, 3 }, { 4, 5 }, { 1, 5 } } Output: [0 1 0 0 1, 1 0 1 0 0, 0 1 0 0 0, 0 0 0 0 1, 1 0 0 1 0]
9	Prob. 1 Given a undirected Graph of N vertices 1 to N and M edges in form of 2D array arr[][] whose every row consists of two numbers X and Y which denotes that there is an edge between X and Y, now write a program to create Laplacian matrix. The Laplacian matrix (also known as graph Laplacian, admittance matrix, Kirchhoff matrix or discrete Laplacian) is a matrix representation of a graph. To find the Laplacian matrix first, find adjacency matrix and degree matrix of a graph as the formula for the Laplacian matrix is as follows: Laplacian matrix = Degree matrix – Adjacency matrix Input: N = 5, M = 4, arr[][] = { { 1, 2 }, { 2, 3 }, { 4, 5 }, { 1, 5 } }

	Output: [2 -1 0 0 -1, -1 2 -1 0 0, 0 -1 1 0 0, 0 0 0 1 -1, -1 0 0 -1 2]
10	Prob. 1 Write a program to find a Dominant set of a Graph. (Undirected Graph) Dominant Set is a set of vertices S, such that for every vertex in the graph, it is an adjacent vertex to at least one of the vertex in the set S. Input: N = 4, M = 4, arr[][] = {{ 1, 2 }, { 1, 3 }, { 3, 4 }, { 2, 4 }}  Possible Dominant Sets are: S = {1,3} or {1,2} or {1,4} and many more.
11	Prob.1 Implement a solution to find a shortest path in a network (Dijkstra's Algorithm). Given: A weighted graph and a source vertex Find: The shortest distance from the source to all other vertices. Hint: Mark all distances as infinity except the source (distance = 0). Use a priority queue or greedy selection to always choose the closest unvisited node. Update distances to neighbors. Repeat until all nodes are visited.
12	Prob. 1 Implement a solution for the following problem using Greedy method. Objective: We want to minimize the cost while maximizing throughput. Problem Details: Cost of assigning Virtual Machine (VM) to task: The cost matrix is given (you could use a random matrix or a matrix based on the specifics of the cloud system). VM capabilities: Each VM has certain available resources (CPU, memory). (Take an input from user) Task requirements: Each task needs a certain amount of CPU and memory. (Take an input from user)
13	Prob. 1 Find and print all cut-vertices (articulation points) in an undirected graph.
14	Prob. 1 Given a bipartite graph with parts X and Y, check if Hall's condition holds.
15	Prob. 1 Degree Distribution Visualization: Plot and visualize degree distribution and eigenvector centrality.