Data Structures and Algorithm Design

Course title & Code	Credits	Credit distribution of the course			Eligibility criteria	Pre-requisite of the course (if any)
		Lecture	Tutorial	Practical/ Practice		
Data Structures and Algorithm Design	4	3	0	1	Class XII Pass	NA

COURSE OBJECTIVE

- Understand and implement fundamental data structures (stacks, queues, linked lists, trees, and graphs).
- Learn algorithm design techniques like divide and conquer, dynamic programming, and greedy algorithms.
- Analyze the efficiency of algorithms using asymptotic notations (Big-O, Omega, Theta).
- Apply data structures and algorithms to solve real-world problems like polynomial manipulation and the knapsack problem.
- Develop problem-solving skills and optimize solutions for efficiency.

COURSE OUTCOME

Upon successful completion of this course, the student will be able to:

- Use and manipulate various data structures, including stacks, queues, linked lists, and trees, effectively.
- Use asymptotic notations (Big-O, Omega, Theta) to analyze and evaluate the time and space complexity of algorithms.
- Use advanced algorithm design techniques such as dynamic programming and greedy algorithms to solve computational problems.
- Use tree and graph algorithms (e.g., traversal, shortest path, and minimum spanning tree) to solve real-world problems.
- Use critical thinking and problem-solving skills to design, optimize, and implement efficient computational solutions.

SYLLABUS

Unit I: Linear Data Structures **(9 Hours)**

Stack- Definition, operations, and applications, evaluating arithmetic expressions, Other applications of stacks

Queue- Types and applications, Simple Queue, Circular Queue, Double-Ended Queue (Deque)

Linked Lists- Singly Linked List, Circularly Linked List, Doubly Linked List, Applications of Linked Lists-Polynomial Manipulation

Unit II: Non-Linear Tree Structures **(12 Hours)**

Introduction to Trees: Definition and Terminology (Root, Parent, Child, Sibling, Leaf, Depth, Height),Properties of Trees, Binary Trees, Structure and traversals of trees (in-order, pre-order, post-order), Applications of Binary Trees, Binary Search Tree, Balanced Trees- AVL Tree, B-Tree Insertion and Deletion

Unit III: Algorithm Design and Analysis **(12 Hours)**

Asymptotic Notations (Big-O, Omega, Theta), Insertion Sort Time Complexity Analysis,Divide and Conquer Approach- Quick Sort, Linear Search vs Binary Search, Greedy Algorithms-Knapsack Problem, Dynamic Programming-Longest Common Subsequence

Unit IV: Graphs **(12 Hours)**

Graph Representations-Adjacency Matrix and Adjacency List, Graph Traversals-Depth-First Search (DFS), Breadth-First Search (BFS), Single Source Shortest Path Algorithms-Dijkstra's Algorithm, Bellman-Ford Algorithm,Minimum Spanning Tree - Prim's Algorithm, Kruskal's Algorithm

REFERENCES

1. "Introduction to Algorithms" by Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein
2. E. Horowitz, S.Sahni and Dinesh Mehta, "Fundamentals of Data structures in C++",University Press, 2007
3. E. Horowitz, S. Sahni and S. Rajasekaran, "Computer Algorithms/C++",Second Edition, University Press, 2007

PRACTICAL COMPONENT (IF ANY)

Use Python for practical labs for Machine Learning.

LIST OF PRACTICALS (30 Hours)

1. Create a stack and implement the basic operations (push, pop, peek) using arrays and linked lists.
2. Create a program to evaluate arithmetic expressions using a stack.
3. Create a simple queue using an array. Then, modify it to make a circular queue and a double-ended queue (Deque) with operations like adding and removing from both ends.
4. Create a doubly linked list and perform basic operations like adding, removing, and displaying elements.
5. Create a linked list to represent a polynomial and create operations to add, subtract, and multiply polynomials.
6. Create a binary tree and perform tree traversals: in-order, pre-order, and post-order (ways to visit and print all the nodes).
7. Create an AVL tree (a self-balancing tree) and implement insertions and deletions while ensuring the tree remains balanced.
8. Implement different sorting algorithms: insertion sort, quick sort, and merge sort, and analyze how fast they run with different input sizes.
9. Implement the Knapsack Problem.