BSC. (HONS.) COMPUTER SCIENCE

DISCIPLINE SPECIFIC CORE COURSE -7 (DSC-7) : Data Structures

CREDIT DISTRIBUTION, ELIGIBILITY AND PRE-REQUISITES OF THE COURSE

Course title & Code	Credits	Credit distribution of the course			Eligibility criteria	Pre-requisite of the course (if any)
		Lecture	Tutorial	Practical/ Practice		
DSC07 Data Structures	**4**	**3**	**0**	**1**	Pass in Class XII	DSC-01/DSC04

Learning Objectives

The course aims at developing the ability to use basic data structures like arrays, stacks, queues, lists, and trees to solve problems. C++ is chosen as the language to implement the implementation of these data structures.

Learning outcomes

On successful completion of the course, students will be able to:

- Compare two functions for their rates of growth.
- Understand abstract specification of data-structures and their implementation.
- Compute time and space complexity of operations on a data-structure.
- Identify the appropriate data structure(s) for a given application and understand the trade-offs involved in terms of time and space complexity.
- Apply recursive techniques to solve problems.

SYLLABUS OF DSC-7

Unit 1 (9 hours)

Growth of Functions, Recurrence Relations: Functions used in analysis, asymptotic notations, asymptotic analysis, solving recurrences using recursion trees, Master Theorem.

Unit 2 (16 hours)

Arrays, Linked Lists, Stacks, Queues: Arrays: array operations, applications, two-dimensional arrays, dynamic allocation of arrays; Linked Lists: singly linked lists, doubly linked lists, circularly linked lists, Stacks: stack as an ADT, implementing stacks using arrays, implementing stacks using linked lists, applications of stacks; Queues: queue as an ADT,

implementing queues using arrays, implementing queues using linked lists,. Time complexity analysis.

Unit 3 (5 hours)

Recursion: Recursive functions, linear recursion, binary recursion.

Unit 4 (6 hours)

Trees, Binary Trees: Trees: definition and properties, tree traversal algorithms and their time complexity analysis; binary trees: definition and properties, traversal of binary trees and their time complexity analysis.

Unit 5 (7 hours)

Binary Search Trees, Balanced Search Trees: Binary Search Trees: insert, delete, search operations, time complexity analysis of these operations; Balanced Search Trees: insert, search operations, time complexity analysis of these operations. Time complexity analysis.

Unit 6 (2 hours)

Binary Heap: Binary Heaps: heaps, heap operations.

Essential/recommended readings

1. Goodrich, M.T., Tamassia, R., & Mount, D., *Data Structures and Algorithms Analysis in C++,* 2nd edition, Wiley, 2011.
2. Cormen, T.H., Leiserson, C.E., Rivest, R. L., Stein C. *Introduction to Algorithms*, 4th edition, Prentice Hall of India, 2022.

Additional references

1. Sahni, S. *Data Structures, Algorithms and applications in C++*, 2nd edition, Universities Press, 2011.
2. Langsam Y., Augenstein, M. J., & Tanenbaum, A. M. *Data Structures Using C and C++*, Pearson, 2009.

Practical List (If any): (30 Hours)

Practical exercises such as

1. Write a program to implement singly linked list as an ADT that supports the following operations:
   (i) Insert an element x at the beginning of the singly linked list
   (ii) Insert an element x at i^{th} position in the singly linked list
   (iii) Remove an element from the beginning of the singly linked list
   (iv) Remove an element from i^{th} position in the singly link
   (v) Search for an element x in the singly linked list and return its pointer
   (vi) Concatenate two singly linked lists

2. Write a program to implement doubly linked list as an ADT that supports the following operations:
   (i) Insert an element x at the beginning of the doubly linked list
   (ii) Insert an element x at i^{th} position in the doubly linked list
   (iii) Insert an element x at the end of the doubly linked list
   (iv) Remove an element from the beginning of the doubly linked list
   (v) Remove an element from i^{th} position in the doubly linked list.
   (vi) Remove an element from the end of the doubly linked list
   (vii) Search for an element x in the doubly linked list and return its pointer
   (viii) Concatenate two doubly linked lists

3. Write a program to implement circular linked list as an ADT which supports the following operations:
   (i) Insert an element x at the front of the circularly linked list
   (ii) Insert an element x after an element y in the circularly linked list
   (iii) Insert an element x at the back of the circularly linked list
   (iv) Remove an element from the back of the circularly linked list
   (v) Remove an element from the front of the circularly linked list
   (vi) Remove the element x from the circularly linked list
   (vii) Search for an element x in the circularly linked list and return its pointer
   (viii) Concatenate two circularly linked lists

4. Implement a stack as an ADT using Arrays.

5. Implement a stack as an ADT using the Linked List ADT.

6. Write a program to evaluate a prefix/postfix expression using stacks.

7. Implement Queue as an ADT using the circular Arrays.

8. Implement Queue as an ADT using the Circular Linked List ADT.

9. Write a program to implement Binary Search Tree as an ADT which supports the following operations:
   (i) Insert an element x
   (ii) Delete an element x
   (iii) Search for an element x in the BST and change its value to y and then place the node with value y at its appropriate position in the BST
   (iv) Display the elements of the BST in preorder, inorder, and postorder traversal
   (v) Display the elements of the BST in level-by-level traversal
   (vi) Display the height of the BST

10. Write a program to implement a balanced search tree as an ADT.