|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **SCHOOL OF COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE** | | | | | **DEPARTMENT OF COMPUTER SCIENCE ENGINEERING** | | | | |
| **Program Name:** B. Tech | | | | **Assignment Type: Lab** | | | **Academic Year:**2025-2026 | | |
| **Course Coordinator Name** | | | | Venkataramana Veeramsetty | | | | | |
| **Instructor(s) Name** | | | | |  | | --- | | Dr. V. Venkataramana (Co-ordinator) | | Dr. T. Sampath Kumar | | Dr. Pramoda Patro | | Dr. Brij Kishor Tiwari | | Dr.J.Ravichander | | Dr. Mohammand Ali Shaik | | Dr. Anirodh Kumar | | Mr. S.Naresh Kumar | | Dr. RAJESH VELPULA | | Mr. Kundhan Kumar | | Ms. Ch.Rajitha | | Mr. M Prakash | | Mr. B.Raju | | Intern 1 (Dharma teja) | | Intern 2 (Sai Prasad) | | Intern 3 (Sowmya) | | NS\_2 ( Mounika) | | | | | | |
| **Course Code** | | | 24CS002PC215 | **Course Title** | | AI Assisted Coding | | | |
| **Year/Sem** | | | II/I | **Regulation** | | R24 | | | |
| **Date and Day**  **of Assignment** | | | Week6 - Monday | **Time(s)** | |  | | | |
| **Duration** | | | 2 Hours | **Applicable to**  **Batches** | |  | | | |
| **AssignmentNumber:11.1**(Present assignment number)/**24**(Total number of assignments) | | | | | | | | | |
| **NAME: P. HEMAN ROLL NO:2403A510F5**  **BATCH:06 DATE:15-09-2025** | | | | | | | | | |
|  | **Q.No.** | **Question** | | | | | | ***Expected Time***  ***to complete*** |  |
|  | 1 | **Lab 11 – Data Structures with AI: Implementing Fundamental Structures**  **Lab Objectives**   * Use AI to assist in designing and implementing fundamental data structures in Python. * Learn how to prompt AI for structure creation, optimization, and documentation. * Improve understanding of Lists, Stacks, Queues, Linked Lists, Trees, Graphs, and Hash Tables. * Enhance code quality with AI-generated comments and performance suggestions.   **Task Description #1 – Stack Implementation**  Task: Use AI to generate a Stack class with push, pop, peek, and is\_empty methods.  Sample Input Code:  class Stack:  pass  Expected Output:   * A functional stack implementation with all required methods and docstrings.       OUTPUT:  OBSERVATION:  In this experiment, a stack was created using Python. A stack follows the **Last In, First Out** rule, which means the last element added is the first one to be removed. The program allows us to add elements (push), remove elements (pop), see the top element (peek), and check if the stack is empty. By using the menu, we can easily test how the stack works.  **Task Description #2 – Queue Implementation**  Task: Use AI to implement a Queue using Python lists.  Sample Input Code:  class Queue:  pass  Expected Output:   * FIFO-based queue class with enqueue, dequeue, peek, and size methods.       OUTPUT:    OBSERVATION:  In this experiment, a queue was created using Python. A queue follows the **First In, First Out** rule, which means the first element added is the first one to be removed. The program allows us to add elements (enqueue), remove elements (dequeue), check the front element (peek), see the size, and display the queue. By using the menu, we can clearly see how the queue works in order  **Task Description #3 – Linked List**  Task: Use AI to generate a Singly Linked List with insert and display methods.  Sample Input Code:  class Node:  pass  class LinkedList:  pass  Expected Output:   * A working linked list implementation with clear method documentation.     OUTPUT:    OBSERVATION:  A singly linked list was implemented where each element (node) contains data and a reference to the next node. Elements can be inserted at the end, and all elements can be displayed in order. This structure allows dynamic memory allocation and easy insertion without shifting elements, unlike arrays.  **Task Description #4 – Binary Search Tree (BST)**  Task: Use AI to create a BST with insert and in-order traversal methods.  Sample Input Code:  class BST:  pass  Expected Output:   * BST implementation with recursive insert and traversal methods.     OUTPUT:    OBSERVATION:  A BST was created where each node has a left and right child. Elements are inserted in such a way that left child < parent < right child. In-order traversal displays elements in sorted order. This structure helps in efficient searching, insertion, and deletion compared to linear lists  **Task Description #5 – Hash Table**  Task: Use AI to implement a hash table with basic insert, search, and delete methods.  Sample Input Code:  class HashTable:  pass  Expected Output:   * Collision handling using chaining, with well-commented methods.   OUTPUT:    OBSERVATION:  A hash table was implemented using chaining to handle collisions. Elements can be inserted, searched, and deleted efficiently using a key. This structure provides fast access to data (average O(1) time) and is useful for scenarios requiring quick lookups  **Task Description #6 – Graph Representation**  Task: Use AI to implement a graph using an adjacency list.  Sample Input Code:  class Graph:  pass  Expected Output:   * Graph with methods to add vertices, add edges, and display connections.     OUTPUT:    Observation:  A graph was implemented using an adjacency list. Vertices and edges can be added, and all connections are displayed. This representation efficiently handles sparse graphs and is useful for modeling networks like road maps or social connections  **Task Description #7 – Priority Queue**  Task: Use AI to implement a priority queue using Python’s heapq module.  Sample Input Code:  class PriorityQueue:  pass  Expected Output:   * Implementation with enqueue (priority), dequeue (highest priority), and display methods.     OUTPUT:    OBSERVATION:  A priority queue was implemented using Python’s heapq module. Elements are enqueued with a priority, and the element with the highest priority is dequeued first. This structure is useful in scheduling and task management where some items need to be processed before others  **Task Description #8 – Deque**  Task: Use AI to implement a double-ended queue using collections.deque.  Sample Input Code:  class DequeDS:  pass  Expected Output:   * Insert and remove from both ends with docstrings.   **Task Description #9 – AI-Generated Data Structure Comparisons**  Task: Use AI to generate a comparison table of different data structures (stack, queue, linked list, etc.) including time complexities.  Sample Input Code:  # No code, prompt AI for a data structure comparison table  Expected Output:   * A markdown table with structure names, operations, and complexities.     OUTPUT:    OBSERBVATION:  Different data structures (stack, queue, linked list, BST, hash table, graph, deque, priority queue) were compared based on operations like insertion, deletion, search, and traversal. Each structure has its own advantages; for example, stacks and queues are simple, BSTs allow sorted access, and hash tables provide fast lookup  **Task Description #10 Real-Time Application Challenge – Choose the Right Data Structure**  **Scenario:** Your college wants to develop a Campus Resource Management System that handles:   1. Student Attendance Tracking – Daily log of students entering/exiting the campus. 2. Event Registration System – Manage participants in events with quick search and removal. 3. Library Book Borrowing – Keep track of available books and their due dates. 4. Bus Scheduling System – Maintain bus routes and stop connections. 5. Cafeteria Order Queue – Serve students in the order they arrive.   Student Task:   * For each feature, select the most appropriate data structure from the list below:   + Stack   + Queue   + Priority Queue   + Linked List   + Binary Search Tree (BST)   + Graph   + Hash Table   + Deque * Justify your choice in 2–3 sentences per feature. * Implement one selected feature as a working Python program with AI-assisted code generation.   Expected Output:   * A table mapping feature → chosen data structure → justification. * A functional Python program implementing the chosen feature with comments and docstrings.     OUTPUT:    ✅ Deliverables (For All Tasks)   1. AI-generated prompts for code and test case generation. 2. At least 3 assert test cases for each task. 3. AI-generated initial code and execution screenshots. 4. Analysis of whether code passes all tests. 5. Improved final version with inline comments and explanation. 6. Compiled report (Word/PDF) with prompts, test cases, assertions, code, and output.   Top of Form | | | | | | Week6 - Monday |  |