|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **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 - Thursday | **Time(s)** | |  | | | |
| **Duration** | | | 2 Hours | **Applicable to**  **Batches** | |  | | | |
| **AssignmentNumber:11.1**(Present assignment number)/**24**(Total number of assignments) | | | | | | | | | |
|  | **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 1: Implementing a Stack (LIFO)**   * **Task**: Use AI to help implement a **Stack** class in Python with the following operations: push(), pop(), peek(), and is\_empty(). * **Instructions**:   + Ask AI to generate code skeleton with docstrings.   + Test stack operations using sample data.   + Request AI to suggest optimizations or alternative implementations (e.g., using collections.deque).   **Expected Output**:  A working Stack class with proper methods, Google-style docstrings, and inline comments for tricky parts  PROMPT: generate a python code for implement the stacks with operations like push(), pop(),peek() and is\_empty()      OUTPUT:  **Task 2: Queue Implementation with Performance Review**   * **Task**: Implement a **Queue** with enqueue(), dequeue(), and is\_empty() methods. * **Instructions**:   + First, implement using Python lists.   + Then, ask AI to review performance and suggest a more efficient implementation (using collections.deque). * **Expected Output**:   Two versions of a queue: one with lists and one optimized with deque, plus an AI-generated performance comparison PROMPT: generate a python code for queue with enqueu(), dequeue(), and is\_empty(). .  OUTPUT:  **Task 3: Singly Linked List with Traversal**   * **Task**: Implement a **Singly Linked List** with operations: insert\_at\_end(), delete\_value(), and traverse(). * **Instructions**:   + Start with a simple class-based implementation (Node, LinkedList).   + Use AI to generate inline comments explaining pointer updates (which are non-trivial).   + Ask AI to suggest test cases to validate all operations. * **Expected Output**:   A functional linked list implementation with clear comments explaining the logic of insertions and deletions. PROMPT: generate a python code for singly linked list with operations insert\_at\_end(), delete\_value(), and traverse()              OUTPUT:  **Task 4: Binary Search Tree (BST)**   * **Task**: Implement a **Binary Search Tree** with methods for insert(), search(), and inorder\_traversal(). * **Instructions**:   + Provide AI with a partially written Node and BST class.   + Ask AI to complete missing methods and add docstrings.   + Test with a list of integers and compare outputs of search() for present vs absent elements. * **Expected Output**:   A BST class with clean implementation, meaningful docstrings, and correct traversal output. PROMPT: generate a python code for binary search tree using commands like insert(), search(), inorder\_traversal()       OUTPUT:  **Task 5: Graph Representation and BFS/DFS Traversal**   * **Task**: Implement a **Graph** using an adjacency list, with traversal methods BFS() and DFS(). * **Instructions**:   + Start with an adjacency list dictionary.   + Ask AI to generate BFS and DFS implementations with inline comments.   + Compare recursive vs iterative DFS if suggested by AI. * **Expected Output**:   A graph implementation with BFS and DFS traversal methods, with AI-generated comments explaining traversal steps.  PROMPT:genrate a python code for graph using commands like adjacency list with traversal methods BFS() and DFS()  OUTPUT:  Top of Form | | | | | | Week6 - Thursday |  |