|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **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: Gundu Meghana**  **Roll No: 2403A510C1**  **Batch:04** | | | | | | | | | |
|  | **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.     **Prompt#1:**   * Write a python code that has a stack class with push, pop, peek and is\_empty methods using dynamic input.   **Code#1:**      **Observation#1:**   * This Python code defines a Stack class with standard stack operations (push, pop, peek, is\_empty) and demonstrates its use with a dynamic input loop where the user can interactively add elements to the stack, remove them, view the top element, check if the stack is empty, or quit the program, including basic error handling for invalid push commands.   **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.   **Prompt#2:**   * Write a python code that has a queue using python lists using dynamic input.   **Code#2:**      **Observation#2:**   * This Python code defines a Queue class using a list, implementing standard queue operations (enqueue, dequeue, is\_empty, size), and includes a dynamic input loop that allows users to interactively add elements to the front of the queue (enqueue), remove elements from the back (dequeue), check if the queue is empty, get its size, or exit the program, with basic error handling for the enqueue command.   **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.   **Prompt#3:**   * Write a python code that has a singly linked list with insert and display methods using dynamic input.   **Code#3:**      **Observation#3:**   * This Python code defines a Node class for the elements of a singly linked list and a SinglyLinkedList class that includes methods to insert a new node at a specified index and display the elements of the list; it also features a dynamic input loop allowing users to interactively add nodes at specific positions and print the current list, incorporating error handling for invalid insert commands and out-of-bounds indices during insertion.   **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.   **Prompt#4:**   * Write a python code that creates a BST with insert and in-order transversal methods using dynmaic input.   **Code#4:**        **Observation#4:**  **🡪** This Python code defines a Node class for the elements of a Binary Search Tree and a BinarySearchTree class with methods to insert data while maintaining the BST property and perform an in\_order\_traversal to print the elements in sorted order; it includes a dynamic input loop that allows users to interactively add integer values to the tree and display the tree's contents using in-order traversal, with basic error handling for invalid insert commands.  **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.   **Prompt#5:**   * Write a python code that contains a hash table with basic insert, search and delete methods using dynamic input.   **Code#5:**    **Observation#5:**   * This Python code implements a HashTable using a list of lists to handle collisions via chaining, providing methods for inserting key-value pairs (updating the value if the key exists), searching for a key's value, and deleting a key-value pair; it includes a dynamic input loop for interactive use of these operations and displaying the table's contents, along with basic error handling for incorrect command formats.   **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.   **Prompt#6:**   * Write a python code that gives a graph using an adjacency list using dynamic input.   **Code#6:**    **Observation#6:**   * This Python code defines a Graph class using an adjacency list (a dictionary where keys are vertices and values are lists of their neighbors) to represent an undirected graph, providing methods to add\_edge between two vertices (creating vertices if they don't exist) and display the graph's adjacency list; it includes a dynamic input loop for interactively adding edges and visualizing the graph's structure, with basic error handling for invalid add\_edge commands.   **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.   **Prompt#7:**   * Write a python code that has a priority queue with enqueue, dequeue and display methods using Python's heapq module using dynamic input.   **Code#7:**      **Observation#7:**   * This Python code defines a PriorityQueue class that uses the heapq module to maintain a min-heap, effectively simulating a max-heap for priority queue behavior by storing negated priorities; it provides methods to enqueue items with a given integer priority, dequeue the item with the highest priority, peek at the highest priority item, check if the queue is\_empty, get its size, and display the internal representation of the heap, all managed through a dynamic input loop with basic error handling for the enqueue command.   **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.   **Prompt#8**:   * Write a python code that implements a double-ended queue using collections.deque using dynamic input.   **Code#8:**        **Observation#8:**   * This Python code implements a Deque (double-ended queue) class by wrapping the efficient collections.deque object, providing methods to append and appendleft elements, pop and popleft elements, peek at the rightmost and peekleft at the leftmost elements, check if the deque is\_empty, get its size, and display its current contents; the code includes a dynamic input loop for interactive use of these operations with basic error handling for invalid commands.   **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.   **Prompt#9:**   * Write a python code that generates a comparison table of different data structures(stack, queue, linked list etc..) including time complexities using dynamic input.   **Code#9:**  **Observation#9:**   * This markdown cell provides a comparison table summarizing various data structures, including Stack, Queue (both list-based and using collections.deque), Singly Linked List, Binary Search Tree, Hash Table, Priority Queue, and Deque, listing common operations for each along with their typical average and worst-case time complexities, and notes that complexities can differ based on implementation.   **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.   **Prompt#10:**   * Choose the best data structure for each campus feature (attendance, events, library, buses, cafeteria), justify briefly, and implement one (e.g., Cafeteria Queue with dynamic input).     **Code#10:**        **Observation#10:**   * This Python code defines a CafeteriaQueue class that simulates an order queue using collections.deque for efficient FIFO operations, providing methods to add\_order by appending to the right, serve\_order by popping from the left, peek\_next to view the next order, check if the queue is\_empty, get the size of the queue, and display the current orders; the code includes a dynamic input loop for interactive use with commands like add, serve, peek, size, and display, along with basic command validation and informative output messages.   ✅ 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 |  |