Lab 11 – Data Structures with AI: Implementing Fundamental Structures

2303A51760

Batch-28

Task 1 – Stack Implementation

Code:

```
class Stack:
    def __init__(self):
        self.items = []
    def push(self, item):
        self.items.append(item)
    def pop(self):
        if not self.is_empty():
            return self.items.pop()
        return "Stack is empty"
    def peek(self):
        if not self.is_empty():
            return self.items[-1]
        return "Stack is empty"
    def is_empty(self):
        return len(self.items) == 0
```

Task 2 – Queue Implementation

Code:

```
class Queue:
    def __init__(self):
        self.items = []
    def enqueue(self, item):
        self.items.append(item)
    def dequeue(self):
        if self.items:
            return self.items.pop(0)
        return "Queue is empty"
    def peek(self):
        if self.items:
            return self.items[0]
        return "Queue is empty"
    def size(self):
        return len(self.items)
```

Task 3 – Singly Linked List

Code:

```
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None
class LinkedList:
    def __init__(self):
        self.head = None
    def insert(self, data):
        new_node = Node(data)
        if not self.head:
            self.head = new_node
            return
        temp = self.head
        while temp.next:
            temp = temp.next
        temp.next = new_node

    def display(self):
        temp = self.head
        while temp:
            print(temp.data, end=" -> ")
            temp = temp.next
        print("None")
```

Task 4 – Binary Search Tree (BST)

Code:

```
class BST:
    def __init__(self, key):
        self.key = key
        self.left = None
        self.right = None
    def insert(self, value):
        if value < self.key:
            if self.left is None:
                self.left = BST(value)
            else:
                self.left.insert(value)
        else:
            if self.right is None:
                self.right = BST(value)
            else:
                self.right.insert(value)
    def inorder(self):
        if self.left:
            self.left.inorder()
        print(self.key, end=" ")
        if self.right:
            self.right.inorder()
```

🔹 **Task 5 – Hash Table (Chaining)**

Code:

```
class HashTable:
    def __init__(self, size=10):
        self.size = size
        self.table = [[] for _ in range(size)]
    def _hash(self, key):
        return hash(key) % self.size
    def insert(self, key, value):
        index = self._hash(key)
        self.table[index].append((key, value))
    def search(self, key):
        index = self._hash(key)
        for k, v in self.table[index]:
            if k == key:
                return v
        return "Key not found"
    def delete(self, key):
        index = self._hash(key)
        for i, (k, v) in enumerate(self.table[index]):
            if k == key:
                del self.table[index][i]
                return "Deleted"
        return "Key not found"
```

Task 6 – Graph (Adjacency List)

Code:

```
class Graph:
    def __init__(self):
        self.graph = {}
    def add_vertex(self, vertex):
        if vertex not in self.graph:
            self.graph[vertex] = []
    def add_edge(self, v1, v2):
        self.add_vertex(v1)
        self.add_vertex(v2)
        self.graph[v1].append(v2)
        self.graph[v2].append(v1)
    def display(self):
        for vertex in self.graph:
            print(vertex, "->", self.graph[vertex])
```

Task 7 – Priority Queue

Code:

```
import heapq
class PriorityQueue:
    def __init__(self):
        self.heap = []
    def enqueue(self, priority, item):
        heapq.heappush(self.heap, (priority, item))
    def dequeue(self):
        if self.heap:
            return heapq.heappop(self.heap)
        return "Queue is empty"
    def display(self):
        print(self.heap)
```

Task 8 – Deque

Code:

```
from collections import deque
```

```
class DequeDS:
    def __init__(self):
        self.deque = deque()
    def insert_front(self, item):
        self.deque.appendleft(item)
    def insert_rear(self, item):
        self.deque.append(item)
    def remove_front(self):
        return self.deque.popleft()
    def remove_rear(self):
        return self.deque.pop()


)
```

Task 9 – Campus Resource Management System

Feature → Data Structure Mapping

Feature	**Data Structure**	**Justification**
Student Attendance Tracking	Hash Table	Fast lookup by student ID. O(1) average time complexity.
Event Registration	Linked List	Dynamic insertion and removal of participants.
Library Book Borrowing	BST	Books can be organized by ID for sorted access.

Feature	Data Structure	Justification
Bus Scheduling	Graph	Routes and stops form connected networks.
Cafeteria Order Queue	Queue	Orders must be served in FIFO order.

Implemented Feature: Cafeteria Order Queue

```
class CafeteriaQueue:
    """Manage student food orders using Queue."""

    def __init__(self):
        self.orders = []

    def place_order(self, student_name):
        self.orders.append(student_name)
        print(f"{student_name} placed order.")

    def serve_order(self):
        if self.orders:
            served = self.orders.pop(0)
            print(f"Serving {served}")
        else:
            print("No orders to serve.")

    def display_orders(self):
        print("Current Orders:", self.orders)


# Example
cq = CafeteriaQueue()
```

```
cq.place_order("Rahul")
cq.place_order("Anita")
cq.serve_order()
cq.display_orders()
```

Task 10 – Smart E-Commerce Platform

Feature → Data Structure Mapping

Feature	**Data Structure**	**Justification**
Shopping Cart	Linked List	Dynamic addition/removal of products.
Order Processing	Queue	Orders processed in FIFO order.
Top-Selling Products	Priority Queue	Ranked by highest sales.
Product Search	Hash Table	Fast product lookup by ID.
Delivery Route Planning	Graph	Warehouses and locations form network paths.

Implemented Feature: Product Search (Hash Table)

```
class ProductSearch:
    """Product lookup system using Hash Table."""

    def __init__(self):
        self.products = {}

    def add_product(self, product_id, name):
        self.products[product_id] = name
    def search_product(self, product_id):
        return self.products.get(product_id, "Product not found")


# Example
ps = ProductSearch()
ps.add_product(101, "Laptop")
ps.add_product(102, "Mobile")

print(ps.search_product(101))
print(ps.search_product(200))
```