Data Structure & Algorithm

Lab Project$_s$ Details

Instructor: Taha Ali **Total Marks:**10

Instruction:

- **Each Group Consist of 3 Members.**
- **Last Date of Project submission is January 31st, 2025.**

Note:

- **Demo will be taken individually.**

1. Smart City Waste Management System

Scenario:

A smart city requires an efficient waste collection system to ensure cleanliness and minimize operational costs. Your task is to develop a system that monitors waste levels in bins across the city and dynamically assigns collection trucks to optimize routes and schedules.

Features:

1. **Bin Monitoring**:
   - Use a **Heap** to prioritize bins based on their waste levels. Bins closer to full capacity will be serviced first.
2. **Route Optimization**:
   - Represent the city road network using **Graphs**. Implement **Dijkstra's Algorithm** to calculate the shortest route for collection trucks.
3. **Dynamic Updates**:
   - Handle real-time traffic updates and reroute collection trucks dynamically.
4. **Data Storage**:
   - Use a **HashMap** to associate bin IDs with their locations and waste levels.
5. **Scheduling**:
   - Implement **Queues** to manage truck dispatch schedules for efficient waste collection.

Outcome:

The system will reduce operational costs, improve waste collection efficiency, and contribute to a cleaner city.

---

2. Hospital Patient Management System

Scenario:

Hospitals often struggle with managing large volumes of patients, appointments, and emergency cases. Develop a system that automates patient records, appointment scheduling, and emergency handling to ensure better healthcare delivery.

Features:

1. **Patient Records**:
   - Use **Linked Lists** to maintain a history of patient visits for easy record retrieval.
2. **Appointment Scheduling**:
   - Use a **Priority Queue** to schedule appointments based on urgency or severity of the condition.
3. **Emergency Management**:
   - Use **Graphs** to find the fastest ambulance routes to hospitals in emergencies.
4. **Doctor Availability**:
   - Track doctors' schedules using **Stacks** to handle overlapping appointments.
5. **Reporting**:
   - Use a **Binary Search Tree (BST)** to generate reports on patient statistics, such as the most common conditions treated.

Outcome:

The system will streamline patient management, reduce wait times, and enhance emergency response capabilities.

---

3. E-Commerce Inventory Management System

Scenario:

With thousands of products and orders to manage, an e-commerce platform needs an efficient system for inventory tracking, order management, and personalized recommendations to improve customer satisfaction.

Features:

1. **Inventory Management**:
    - Use a **Trie** to allow quick searching of products by name or category.
2. **Order Tracking**:
    - Implement **Queues** to process orders in the sequence they are received.
3. **Product Recommendations**:
    - Use **Graphs** to recommend related or frequently bought-together products.
4. **Sales Reporting**:
    - Implement a **Segment Tree** to analyze and display sales trends over specific time periods.
5. **Stock Alerts**:
    - Use a **Heap** to monitor low-stock products and prioritize restocking.

Outcome:

The system will provide seamless inventory tracking, enhanced customer experience through recommendations, and accurate sales reporting.

---

4. Airline Reservation System

Scenario:

Managing flight reservations efficiently is crucial for airlines to ensure passenger satisfaction and operational smoothness. Design a system to handle bookings, cancellations, and seat allocations dynamically.

Features:

1. **Seat Allocation**:
    - Represent flight seating charts using **2D Arrays** for quick updates and access.
2. **Booking Management**:
    - Use a **Queue** to handle ticket booking requests in order.
3. **Cancellation Handling**:
    - Use a **Stack** to maintain a history of cancellations for audit and refund tracking.
4. **Flight Scheduling**:
    - Represent flight connections using **Graphs** and allow users to view optimal routes for connecting flights.
5. **Frequent Flyer Benefits**:

- Implement a **Priority Queue** to prioritize frequent flyers for upgrades and additional perks.

Outcome:

The system will improve reservation processes, enhance customer experience, and provide better flight management capabilities.

---

5. College Event Management System

Scenario:

Universities often host multiple events simultaneously, requiring a system to manage participant registrations, event schedules, and results efficiently.

Features:

1. **Participant Registration**:
   - Use **Linked Lists** to maintain a dynamic list of registered participants.
2. **Event Scheduling**:
   - Use a **Heap** to schedule events by priority or available time slots.
3. **Results Management**:
   - Use a **HashMap** to store event results and allow quick retrieval.
4. **Venue Allocation**:
   - Represent venues as nodes in a **Graph** and allocate them based on availability.
5. **Leaderboard**:
   - Implement sorting algorithms (e.g., MergeSort) to display leaderboards of top-performing participants.

Outcome:

The system will ensure smooth event organization and provide real-time updates for participants and organizers.

---

6. Ride-Sharing Application

Scenario:

Ride-sharing platforms require efficient driver-passenger matching, route optimization, and fare calculations. Develop a system that handles these aspects dynamically.

Features:

1. **Driver-Passenger Matching**:
   - Use a **Priority Queue** to match passengers with the closest available drivers.
2. **Route Optimization**:
   - Use **Dijkstra's Algorithm** to calculate the shortest route between pickup and drop-off points.
3. **Dynamic Pricing**:
   - Use **Heaps** to dynamically adjust fares during peak hours.
4. **Ride History**:
   - Use **Stacks** to store and retrieve past rides for users.
5. **City Mapping**:
   - Represent the city's roads as a **Graph** to optimize routes and handle traffic updates.

Outcome:

The system will provide fast, efficient, and cost-effective ride-sharing services.

---

7. Movie Recommendation System

Scenario:

Streaming platforms rely on personalized recommendations to enhance user experience. Develop a movie recommendation system that uses user preferences and viewing history to suggest relevant content.

Features:

1. **Movie Search**:
   - Use a **Trie** to allow efficient searching by title or genre.
2. **User Preferences**:
   - Represent user interests as nodes in a **Graph** and find similar users for collaborative recommendations.
3. **Viewing History**:
   - Use **Stacks** to store and retrieve recently watched movies.
4. **Trending Movies**:
   - Use a **Heap** to rank trending movies based on ratings and reviews.
5. **Recommendation Algorithm**:
   - Use **Dynamic Programming** to calculate recommendations based on viewing patterns.

Outcome:

The system will enhance user engagement and satisfaction by providing accurate recommendations.

---

8. Weather Prediction System

Scenario:

Weather prediction requires analyzing historical data and real-time sensor inputs. Design a system to provide accurate weather forecasts and alerts.

Features:

1. **Data Storage**:
   - Use a **HashMap** to store historical weather data for specific locations.
2. **Prediction Models**:
   - Use **Dynamic Programming** to calculate future weather trends.
3. **Real-Time Updates**:
   - Represent sensor data flow using **Graphs** for analysis.
4. **Priority Alerts**:
   - Use a **Priority Queue** to send severe weather alerts to affected regions.
5. **Trend Analysis**:
   - Use a **BST** to analyze and display temperature trends.

Outcome:

The system will provide accurate weather forecasts and timely alerts, improving public safety.

---

9. Social Network Platform

Scenario:

Develop a social networking platform where users can connect, share posts, and follow trends.

Features:

1. **User Connections**:
   - Represent user connections as a **Graph** to visualize and recommend friends.
2. **News Feed**:
   - Use a **Priority Queue** to display trending posts.

3. **Message History**:
   - Use **Stacks** to store chat history for quick access.
4. **Search Functionality**:
   - Use a **Trie** for efficient searching of users and groups.
5. **Recommendation Engine**:
   - Use **Heaps** to recommend friends or popular groups.

Outcome:

The platform will enhance user engagement and foster social connections.

---

10. Food Delivery Management System

Scenario:

Develop a food delivery system that connects customers with restaurants, manages orders, and optimizes delivery routes.

Features:

1. **Order Management**:
   - Use **Queues** to process orders in the sequence they are placed.
2. **Delivery Route Optimization**:
   - Use **Graphs** to model delivery routes and calculate the shortest paths.
3. **Customer Feedback**:
   - Use a **HashMap** to store and analyze customer reviews.
4. **Restaurant Search**:
   - Use a **Trie** to provide efficient restaurant search functionality.
5. **Driver Assignment**:
   - Use a **Heap** to assign delivery drivers based on proximity to the restaurant and customer.

Outcome:

The system will improve food delivery efficiency and enhance customer satisfaction.
