



Data Structure & Object-Oriented Programming LAB

COMPLEX ENGINEERING ACTIVITY

CAB Booking System

<p>Name: Waleed Asif Syed Anis Shah Jibran khan Adnan Ali Roll no: 2420532,242073, 242039, 242108 Class: BEMTS-F-24-A Submitted to: Engr. Sidrish Ehsan</p>	
<p>SESSION: Fall 25</p>	
<p>DATE OF REPORT SUBMITTED: 25 December 2025</p>	<p>GRADE/POINTS:</p>

Table of Contents

1.	Introduction:	3
2.	Project Working.....	3
1.	Driver Management using Binary Search Tree (BST)	3
2.	Ride Requests and Ride History Management using Linked List	3
3.	Shortest Path Computation using Graph and Dijkstra's Algorithm	3
4.	Pending Ride Management using Queue (FIFO).....	4
5.	Integrated System Operation.....	4
3.	Implementation Details	4
1.	SimpleQueue.....	4
2.	RideHistory and RideRecord (Linked List)	4
3.	Driver and DriverBST.....	5
4.	Graph and Shortest Path Computation.....	5
5.	CabSystem (Main Controller).....	5
4.	Flow Chart.....	6
5.	Code:	6
6.	Output:.....	14
7.	Learning Outcomes	17
8.	Conclusion.....	17

CAB BOOKING SYSTEM

1. Introduction:

This project implements a Cab Management System using C++ and fundamental data structures. The system simulates how ride-hailing platforms manage drivers, customers, and city maps. It uses a graph-based city map to calculate shortest paths, a binary search tree (BST) to store drivers, a queue to manage pending ride requests, and a linked list to maintain ride history.

The system assigns the nearest available driver to a ride request using Dijkstra's shortest path algorithm, calculates fare based on distance, and supports ride completion, cancellation, and history tracking through a menu-driven interface.

The system enables users to book rides via a computer or smartphone, automatically assigning a nearby available driver and calculating the route and fare. In this project, the system models a city map of intersections (nodes) and roads (weighted edges), and handles user requests for rides, driver management, and ride history. The motivation is to simulate core functions of a ride-sharing service: adding/removing drivers, requesting and completing rides, and tracking history. This involves efficient data management and algorithms to ensure quick driver lookup, shortest-path computation, and orderly handling of pending requests. A GeeksforGeeks description highlights such a system's features – users enter pickup/drop-off locations, the system finds an available driver, shows estimated fare, and maintains trip history.

2. Project Working

The cab management system integrates multiple fundamental data structures and algorithms to simulate real-world ride-hailing operations. Its core functionalities are described below.

1. Driver Management using Binary Search Tree (BST)

Driver information is maintained using a **Binary Search Tree (BST)**, where each node is uniquely identified by a **driverID**. This structure enables efficient insertion, deletion, searching, and ordered traversal of driver records. Each driver node stores essential attributes, including driver ID, name, current location (**currentNode**), availability status, and total earnings.

The **CabSystem::addDriver** function inserts a new driver into the BST by invoking **DriverBST::insert**, while driver removal is handled through **DriverBST::remove**. To display all registered drivers, the system performs an **in-order traversal** of the BST, which outputs drivers in ascending order of driver ID. This ordered structure ensures organized and readable presentation of driver data.

2. Ride Requests and Ride History Management using Linked List

All ride-related records—whether ongoing, completed, or cancelled—are stored in a **singly linked list** implemented by the **RideHistory** class. The linked list structure allows dynamic growth of ride records and supports constant-time insertion.

Whenever a new ride is created, a **RideRecord** node containing the ride ID, driver ID, source node, destination node, travel distance, fare, and ride status is created and inserted at the head of the linked list using the **RideHistory::add** method. This approach ensures **O(1)** insertion complexity and maintains the ride history in reverse chronological order, with the most recent ride displayed first.

Users can view all ride records through the “Show Ride History” menu option, which traverses the linked list sequentially and displays each stored ride. The linked list is well-suited for this use case because the number of rides is not fixed and efficient insertion is required.

3. Shortest Path Computation using Graph and Dijkstra's Algorithm

The city map is modelled as a **weighted undirected graph**, implemented using an **adjacency matrix**. Nodes represent locations, and weighted edges represent roads with associated distances.

When a ride request is made from a source node to a destination node, the system computes the shortest distances from the source to all other nodes using **Dijkstra's algorithm**. These distances are used to determine

the nearest available driver by comparing the shortest-path distance between the source node and each available driver's current location.

Once a driver is selected, the shortest-path distance from the source to the destination is retrieved and used to calculate the fare based on a fixed per-kilometer rate. Dijkstra's algorithm is appropriate for this scenario because all edge weights are non-negative and single-source shortest paths are required.

The Graph class encapsulates the adjacency matrix representation and provides a `dijkstra()` function to compute shortest-path distances efficiently.

4. Pending Ride Management using Queue (FIFO)

If no suitable driver is available or reachable at the time of a ride request, the request is temporarily stored in a **First-In-First-Out (FIFO) queue**. The system uses two parallel queues to store the source and destination nodes of pending ride requests.

When drivers become available, the "Process Pending Requests" option dequeues each request in the order it was received and attempts to assign a driver again. The FIFO queue ensures fairness by servicing earlier requests before newer ones and provides a simple and effective mechanism for managing delayed ride assignments.

5. Integrated System Operation

These components operate cohesively to manage the complete ride lifecycle. When a ride is requested, the system computes shortest paths using the graph, identifies available drivers through an in-order traversal of the BST, and selects the nearest driver. The ride is then logged in the linked list, and the driver's availability is updated accordingly.

Upon ride completion or cancellation, the ride status is updated in the linked list, and the corresponding driver's availability and earnings are adjusted in the BST. Through the combined use of a BST, linked list, graph, and queue, the system efficiently supports all major operations of a cab management platform.

3. Implementation Details

The C++ implementation of the Cab Management System is organized around several key classes and data structures, each responsible for a specific subsystem of the application.

1. SimpleQueue

The SimpleQueue class implements a **fixed-size, array-based queue** to manage pending ride requests. It provides standard queue operations such as `enqueue(int)`, `dequeue()`, `empty()`, and `full()`, and follows **First-In-First-Out (FIFO)** behavior. Two parallel instances of this queue are used to store the source and destination nodes of pending ride requests. This design ensures that ride requests are processed in the order they are received.

2. RideHistory and RideRecord (Linked List)

The RideHistory class implements a **singly linked list** composed of RideRecord nodes. Each RideRecord stores complete ride information, including ride ID, driver ID, source node, destination node, travel distance, fare, ride status, and a pointer to the next record.

New ride records are added using the `RideHistory::add` method, which inserts each new node at the head of the list. This approach provides **constant-time insertion** and maintains the history in reverse chronological order. The `printAll` method traverses the list from the head to display all stored rides, while the `findByID(int)` method performs a linear search to locate a specific ride for completion or cancellation. The linked list structure is well-suited for this use case due to its dynamic size and efficient insertion characteristics.

3. Driver and DriverBST

The Driver class represents an individual driver as a node in a **Binary Search Tree (BST)**. Each driver node contains the driver ID, name, current location, availability status, total earnings, and pointers to left and right child nodes.

The DriverBST class manages all driver-related operations, including insertion, searching, deletion, and traversal. Drivers are inserted into the BST based on their unique driver ID, preserving the BST ordering property. The find(int) and remove(int) methods use recursive search logic to efficiently locate or delete drivers.

To display drivers, the inorderPrint() method performs an **in-order traversal**, outputting drivers sorted by driver ID. Additionally, the collectAvailable() method traverses the BST and collects pointers to available drivers into an array, which is later used to determine the nearest driver during ride assignment.

4. Graph and Shortest Path Computation

The Graph class represents the city road network using a **weighted adjacency matrix**. The matrix is initialized with a large sentinel value to indicate the absence of direct edges between nodes. Roads are added using the addEdge(u, v, w) method, which creates an undirected connection between two nodes with a specified weight. The graph size is managed dynamically using the resize() function. The print() method outputs the adjacency matrix in a readable format, displaying all connected edges and their weights.

Shortest-path computation is handled by the dijkstra(int source, double dist[]) method. This method implements the classic $O(V^2)$ version of Dijkstra's algorithm, which is suitable for the relatively small graph size used in this project. The algorithm computes the minimum distance from the source node to all other nodes and stores the results in the dist array.

5. CabSystem (Main Controller)

The CabSystem class serves as the central controller that integrates all system components. It aggregates instances of DriverBST, RideHistory, Graph, and two SimpleQueue objects, along with configuration parameters such as nextRideID and ratePerKm.

The class provides high-level methods corresponding to user menu operations, including:

initGraph(int nodes): Initializes the city map with a given number of nodes.

addRoad(int u, int v, double w): Adds a road between two nodes in the graph.

setRate(double r): Sets the fare rate per unit distance.

addDriver(int id, string name, int node): Inserts a new driver into the BST.

removeDriver(int id): Removes a driver from the BST.

updateDriverLocation(int id, int node): Updates a driver's current location.

printDrivers(): Displays all drivers using in-order BST traversal.

printMap(): Displays the city road network.

requestRide(int src, int dest): Validates input, computes shortest paths, selects the nearest available driver, updates driver status, calculates fare, records the ride in the linked list, and handles pending requests if necessary.

processPendingRequests(): Dequeues and reprocesses pending ride requests.

completeRide(int rideID): Marks a ride as completed and updates driver location and earnings.

cancelRide(int rideID): Cancels an ongoing ride and restores driver availability.

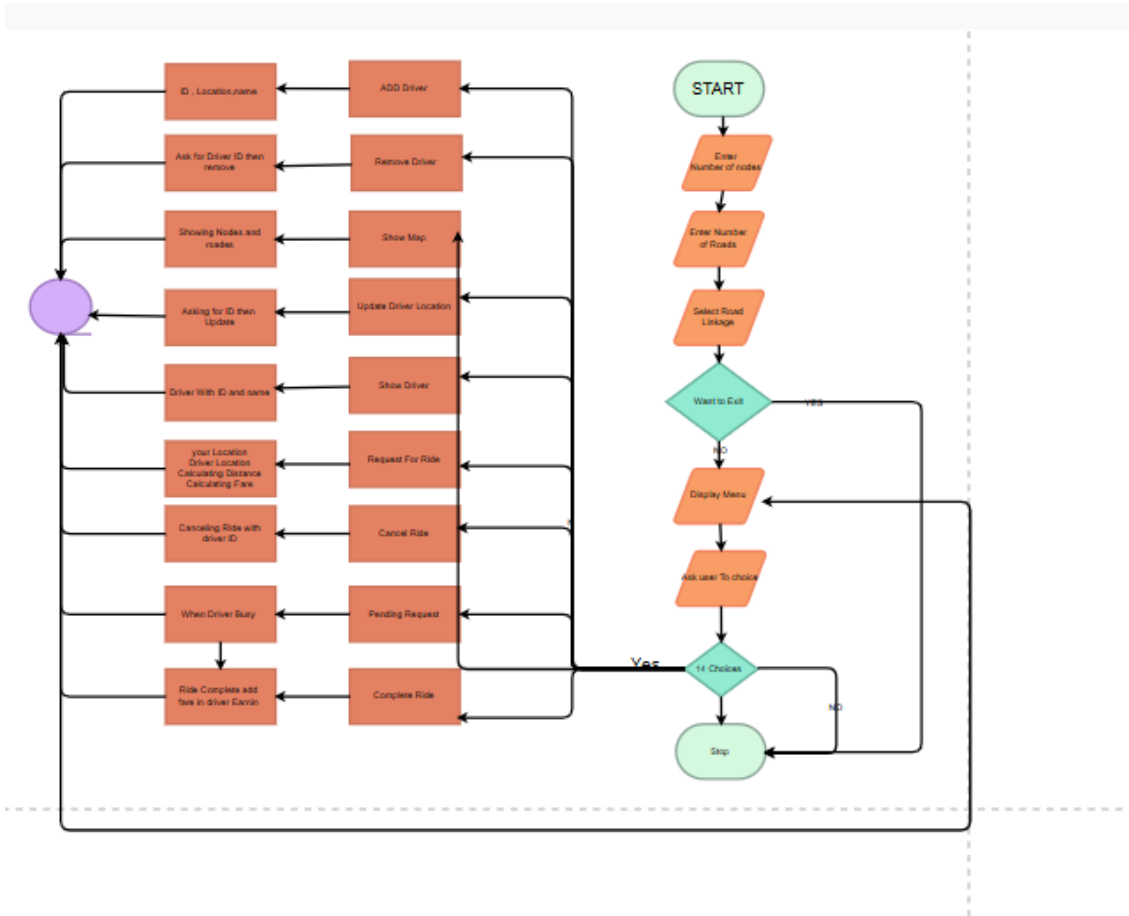
showRideHistory(): Displays all recorded rides.

6. Class Interaction and System Design

The system follows **object-oriented design principles**, where each class encapsulates a specific data structure or subsystem. The CabSystem class coordinates interactions between the graph (route computation), the driver BST (driver selection and management), the linked list (ride history), and the queue (pending rides).

For example, when a ride is requested, the CabSystem invokes the graph to compute shortest paths, queries the BST to identify available drivers, records the ride in the linked list, and updates the relevant driver's state. This modular design improves clarity, maintainability, and scalability of the system.

4. Flow Chart



5. Code:

```

1 #include <iostream>
2 #include <string>
3 #include <sstream>
4 using namespace std;
5
6 // ----- Simple Queue -----
7
8 class SimpleQueue
9 {
10 private:
11     static const int MAXN = 100;
12     int arr[MAXN];
13     int frontIdx;
14     int backIdx;
15     int sz;
16 public:
17     SimpleQueue()
18     {
19         frontIdx = 0;
20         backIdx = -1;
21         sz = 0;
22     }
23
24     bool empty()
25     {
26         return sz == 0;
27     }
28
29     bool full()
30     {
31         return sz == MAXN;
32     }
33
34     void enqueue(int value)
35     {
36         if (full())
37         {
38             cout << "Queue is full!" << endl;
39         }
40     }
41
42     void dequeue()
43     {
44         if (empty())
45         {
46             cout << "Queue is empty!" << endl;
47         }
48     }
49
50     int front()
51     {
52         return arr[frontIdx];
53     }
54
55     int back()
56     {
57         return arr[backIdx];
58     }
59
60     int size()
61     {
62         return sz;
63     }
64
65     void reset()
66     {
67         frontIdx = 0;
68         backIdx = -1;
69         sz = 0;
70     }
71
72     ~SimpleQueue()
73     {
74         // Destructor
75     }
76 };
77
78 int main()
79 {
80     SimpleQueue q;
81     q.enqueue(10);
82     q.enqueue(20);
83     q.enqueue(30);
84     q.enqueue(40);
85     q.enqueue(50);
86     q.enqueue(60);
87     q.enqueue(70);
88     q.enqueue(80);
89     q.enqueue(90);
90     q.enqueue(100);
91     q.enqueue(110);
92     q.enqueue(120);
93     q.enqueue(130);
94     q.enqueue(140);
95     q.enqueue(150);
96     q.enqueue(160);
97     q.enqueue(170);
98     q.enqueue(180);
99     q.enqueue(190);
100    q.enqueue(200);
101    q.enqueue(210);
102    q.enqueue(220);
103    q.enqueue(230);
104    q.enqueue(240);
105    q.enqueue(250);
106    q.enqueue(260);
107    q.enqueue(270);
108    q.enqueue(280);
109    q.enqueue(290);
110    q.enqueue(300);
111    q.enqueue(310);
112    q.enqueue(320);
113    q.enqueue(330);
114    q.enqueue(340);
115    q.enqueue(350);
116    q.enqueue(360);
117    q.enqueue(370);
118    q.enqueue(380);
119    q.enqueue(390);
120    q.enqueue(400);
121    q.enqueue(410);
122    q.enqueue(420);
123    q.enqueue(430);
124    q.enqueue(440);
125    q.enqueue(450);
126    q.enqueue(460);
127    q.enqueue(470);
128    q.enqueue(480);
129    q.enqueue(490);
130    q.enqueue(500);
131    q.enqueue(510);
132    q.enqueue(520);
133    q.enqueue(530);
134    q.enqueue(540);
135    q.enqueue(550);
136    q.enqueue(560);
137    q.enqueue(570);
138    q.enqueue(580);
139    q.enqueue(590);
140    q.enqueue(600);
141    q.enqueue(610);
142    q.enqueue(620);
143    q.enqueue(630);
144    q.enqueue(640);
145    q.enqueue(650);
146    q.enqueue(660);
147    q.enqueue(670);
148    q.enqueue(680);
149    q.enqueue(690);
150    q.enqueue(700);
151    q.enqueue(710);
152    q.enqueue(720);
153    q.enqueue(730);
154    q.enqueue(740);
155    q.enqueue(750);
156    q.enqueue(760);
157    q.enqueue(770);
158    q.enqueue(780);
159    q.enqueue(790);
160    q.enqueue(800);
161    q.enqueue(810);
162    q.enqueue(820);
163    q.enqueue(830);
164    q.enqueue(840);
165    q.enqueue(850);
166    q.enqueue(860);
167    q.enqueue(870);
168    q.enqueue(880);
169    q.enqueue(890);
170    q.enqueue(900);
171    q.enqueue(910);
172    q.enqueue(920);
173    q.enqueue(930);
174    q.enqueue(940);
175    q.enqueue(950);
176    q.enqueue(960);
177    q.enqueue(970);
178    q.enqueue(980);
179    q.enqueue(990);
180    q.enqueue(1000);
181    q.enqueue(1010);
182    q.enqueue(1020);
183    q.enqueue(1030);
184    q.enqueue(1040);
185    q.enqueue(1050);
186    q.enqueue(1060);
187    q.enqueue(1070);
188    q.enqueue(1080);
189    q.enqueue(1090);
190    q.enqueue(1100);
191    q.enqueue(1110);
192    q.enqueue(1120);
193    q.enqueue(1130);
194    q.enqueue(1140);
195    q.enqueue(1150);
196    q.enqueue(1160);
197    q.enqueue(1170);
198    q.enqueue(1180);
199    q.enqueue(1190);
200    q.enqueue(1200);
201    q.enqueue(1210);
202    q.enqueue(1220);
203    q.enqueue(1230);
204    q.enqueue(1240);
205    q.enqueue(1250);
206    q.enqueue(1260);
207    q.enqueue(1270);
208    q.enqueue(1280);
209    q.enqueue(1290);
210    q.enqueue(1300);
211    q.enqueue(1310);
212    q.enqueue(1320);
213    q.enqueue(1330);
214    q.enqueue(1340);
215    q.enqueue(1350);
216    q.enqueue(1360);
217    q.enqueue(1370);
218    q.enqueue(1380);
219    q.enqueue(1390);
220    q.enqueue(1400);
221    q.enqueue(1410);
222    q.enqueue(1420);
223    q.enqueue(1430);
224    q.enqueue(1440);
225    q.enqueue(1450);
226    q.enqueue(1460);
227    q.enqueue(1470);
228    q.enqueue(1480);
229    q.enqueue(1490);
230    q.enqueue(1500);
231    q.enqueue(1510);
232    q.enqueue(1520);
233    q.enqueue(1530);
234    q.enqueue(1540);
235    q.enqueue(1550);
236    q.enqueue(1560);
237    q.enqueue(1570);
238    q.enqueue(1580);
239    q.enqueue(1590);
240    q.enqueue(1600);
241    q.enqueue(1610);
242    q.enqueue(1620);
243    q.enqueue(1630);
244    q.enqueue(1640);
245    q.enqueue(1650);
246    q.enqueue(1660);
247    q.enqueue(1670);
248    q.enqueue(1680);
249    q.enqueue(1690);
250    q.enqueue(1700);
251    q.enqueue(1710);
252    q.enqueue(1720);
253    q.enqueue(1730);
254    q.enqueue(1740);
255    q.enqueue(1750);
256    q.enqueue(1760);
257    q.enqueue(1770);
258    q.enqueue(1780);
259    q.enqueue(1790);
260    q.enqueue(1800);
261    q.enqueue(1810);
262    q.enqueue(1820);
263    q.enqueue(1830);
264    q.enqueue(1840);
265    q.enqueue(1850);
266    q.enqueue(1860);
267    q.enqueue(1870);
268    q.enqueue(1880);
269    q.enqueue(1890);
270    q.enqueue(1900);
271    q.enqueue(1910);
272    q.enqueue(1920);
273    q.enqueue(1930);
274    q.enqueue(1940);
275    q.enqueue(1950);
276    q.enqueue(1960);
277    q.enqueue(1970);
278    q.enqueue(1980);
279    q.enqueue(1990);
280    q.enqueue(2000);
281    q.enqueue(2010);
282    q.enqueue(2020);
283    q.enqueue(2030);
284    q.enqueue(2040);
285    q.enqueue(2050);
286    q.enqueue(2060);
287    q.enqueue(2070);
288    q.enqueue(2080);
289    q.enqueue(2090);
290    q.enqueue(2100);
291    q.enqueue(2110);
292    q.enqueue(2120);
293    q.enqueue(2130);
294    q.enqueue(2140);
295    q.enqueue(2150);
296    q.enqueue(2160);
297    q.enqueue(2170);
298    q.enqueue(2180);
299    q.enqueue(2190);
300    q.enqueue(2200);
301    q.enqueue(2210);
302    q.enqueue(2220);
303    q.enqueue(2230);
304    q.enqueue(2240);
305    q.enqueue(2250);
306    q.enqueue(2260);
307    q.enqueue(2270);
308    q.enqueue(2280);
309    q.enqueue(2290);
310    q.enqueue(2300);
311    q.enqueue(2310);
312    q.enqueue(2320);
313    q.enqueue(2330);
314    q.enqueue(2340);
315    q.enqueue(2350);
316    q.enqueue(2360);
317    q.enqueue(2370);
318    q.enqueue(2380);
319    q.enqueue(2390);
320    q.enqueue(2400);
321    q.enqueue(2410);
322    q.enqueue(2420);
323    q.enqueue(2430);
324    q.enqueue(2440);
325    q.enqueue(2450);
326    q.enqueue(2460);
327    q.enqueue(2470);
328    q.enqueue(2480);
329    q.enqueue(2490);
330    q.enqueue(2500);
331    q.enqueue(2510);
332    q.enqueue(2520);
333    q.enqueue(2530);
334    q.enqueue(2540);
335    q.enqueue(2550);
336    q.enqueue(2560);
337    q.enqueue(2570);
338    q.enqueue(2580);
339    q.enqueue(2590);
340    q.enqueue(2600);
341    q.enqueue(2610);
342    q.enqueue(2620);
343    q.enqueue(2630);
344    q.enqueue(2640);
345    q.enqueue(2650);
346    q.enqueue(2660);
347    q.enqueue(2670);
348    q.enqueue(2680);
349    q.enqueue(2690);
350    q.enqueue(2700);
351    q.enqueue(2710);
352    q.enqueue(2720);
353    q.enqueue(2730);
354    q.enqueue(2740);
355    q.enqueue(2750);
356    q.enqueue(2760);
357    q.enqueue(2770);
358    q.enqueue(2780);
359    q.enqueue(2790);
360    q.enqueue(2800);
361    q.enqueue(2810);
362    q.enqueue(2820);
363    q.enqueue(2830);
364    q.enqueue(2840);
365    q.enqueue(2850);
366    q.enqueue(2860);
367    q.enqueue(2870);
368    q.enqueue(2880);
369    q.enqueue(2890);
370    q.enqueue(2900);
371    q.enqueue(2910);
372    q.enqueue(2920);
373    q.enqueue(2930);
374    q.enqueue(2940);
375    q.enqueue(2950);
376    q.enqueue(2960);
377    q.enqueue(2970);
378    q.enqueue(2980);
379    q.enqueue(2990);
380    q.enqueue(3000);
381    q.enqueue(3010);
382    q.enqueue(3020);
383    q.enqueue(3030);
384    q.enqueue(3040);
385    q.enqueue(3050);
386    q.enqueue(3060);
387    q.enqueue(3070);
388    q.enqueue(3080);
389    q.enqueue(3090);
390    q.enqueue(3100);
391    q.enqueue(3110);
392    q.enqueue(3120);
393    q.enqueue(3130);
394    q.enqueue(3140);
395    q.enqueue(3150);
396    q.enqueue(3160);
397    q.enqueue(3170);
398    q.enqueue(3180);
399    q.enqueue(3190);
400    q.enqueue(3200);
401    q.enqueue(3210);
402    q.enqueue(3220);
403    q.enqueue(3230);
404    q.enqueue(3240);
405    q.enqueue(3250);
406    q.enqueue(3260);
407    q.enqueue(3270);
408    q.enqueue(3280);
409    q.enqueue(3290);
410    q.enqueue(3300);
411    q.enqueue(3310);
412    q.enqueue(3320);
413    q.enqueue(3330);
414    q.enqueue(3340);
415    q.enqueue(3350);
416    q.enqueue(3360);
417    q.enqueue(3370);
418    q.enqueue(3380);
419    q.enqueue(3390);
420    q.enqueue(3400);
421    q.enqueue(3410);
422    q.enqueue(3420);
423    q.enqueue(3430);
424    q.enqueue(3440);
425    q.enqueue(3450);
426    q.enqueue(3460);
427    q.enqueue(3470);
428    q.enqueue(3480);
429    q.enqueue(3490);
430    q.enqueue(3500);
431    q.enqueue(3510);
432    q.enqueue(3520);
433    q.enqueue(3530);
434    q.enqueue(3540);
435    q.enqueue(3550);
436    q.enqueue(3560);
437    q.enqueue(3570);
438    q.enqueue(3580);
439    q.enqueue(3590);
440    q.enqueue(3600);
441    q.enqueue(3610);
442    q.enqueue(3620);
443    q.enqueue(3630);
444    q.enqueue(3640);
445    q.enqueue(3650);
446    q.enqueue(3660);
447    q.enqueue(3670);
448    q.enqueue(3680);
449    q.enqueue(3690);
450    q.enqueue(3700);
451    q.enqueue(3710);
452    q.enqueue(3720);
453    q.enqueue(3730);
454    q.enqueue(3740);
455    q.enqueue(3750);
456    q.enqueue(3760);
457    q.enqueue(3770);
458    q.enqueue(3780);
459    q.enqueue(3790);
460    q.enqueue(3800);
461    q.enqueue(3810);
462    q.enqueue(3820);
463    q.enqueue(3830);
464    q.enqueue(3840);
465    q.enqueue(3850);
466    q.enqueue(3860);
467    q.enqueue(3870);
468    q.enqueue(3880);
469    q.enqueue(3890);
470    q.enqueue(3900);
471    q.enqueue(3910);
472    q.enqueue(3920);
473    q.enqueue(3930);
474    q.enqueue(3940);
475    q.enqueue(3950);
476    q.enqueue(3960);
477    q.enqueue(3970);
478    q.enqueue(3980);
479    q.enqueue(3990);
480    q.enqueue(4000);
481    q.enqueue(4010);
482    q.enqueue(4020);
483    q.enqueue(4030);
484    q.enqueue(4040);
485    q.enqueue(4050);
486    q.enqueue(4060);
487    q.enqueue(4070);
488    q.enqueue(4080);
489    q.enqueue(4090);
490    q.enqueue(4100);
491    q.enqueue(4110);
492    q.enqueue(4120);
493    q.enqueue(4130);
494    q.enqueue(4140);
495    q.enqueue(4150);
496    q.enqueue(4160);
497    q.enqueue(4170);
498    q.enqueue(4180);
499    q.enqueue(4190);
500    q.enqueue(4200);
501    q.enqueue(4210);
502    q.enqueue(4220);
503    q.enqueue(4230);
504    q.enqueue(4240);
505    q.enqueue(4250);
506    q.enqueue(4260);
507    q.enqueue(4270);
508    q.enqueue(4280);
509    q.enqueue(4290);
510    q.enqueue(4300);
511    q.enqueue(4310);
512    q.enqueue(4320);
513    q.enqueue(4330);
514    q.enqueue(4340);
515    q.enqueue(4350);
516    q.enqueue(4360);
517    q.enqueue(4370);
518    q.enqueue(4380);
519    q.enqueue(4390);
520    q.enqueue(4400);
521    q.enqueue(4410);
522    q.enqueue(4420);
523    q.enqueue(4430);
524    q.enqueue(4440);
525    q.enqueue(4450);
526    q.enqueue(4460);
527    q.enqueue(4470);
528    q.enqueue(4480);
529    q.enqueue(4490);
530    q.enqueue(4500);
531    q.enqueue(4510);
532    q.enqueue(4520);
533    q.enqueue(4530);
534    q.enqueue(4540);
535    q.enqueue(4550);
536    q.enqueue(4560);
537    q.enqueue(4570);
538    q.enqueue(4580);
539    q.enqueue(4590);
540    q.enqueue(4600);
541    q.enqueue(4610);
542    q.enqueue(4620);
543    q.enqueue(4630);
544    q.enqueue(4640);
545    q.enqueue(4650);
546    q.enqueue(4660);
547    q.enqueue(4670);
548    q.enqueue(4680);
549    q.enqueue(4690);
550    q.enqueue(4700);
551    q.enqueue(4710);
552    q.enqueue(4720);
553    q.enqueue(4730);
554    q.enqueue(4740);
555    q.enqueue(4750);
556    q.enqueue(4760);
557    q.enqueue(4770);
558    q.enqueue(4780);
559    q.enqueue(4790);
560    q.enqueue(4800);
561    q.enqueue(4810);
562    q.enqueue(4820);
563    q.enqueue(4830);
564    q.enqueue(4840);
565    q.enqueue(4850);
566    q.enqueue(4860);
567    q.enqueue(4870);
568    q.enqueue(4880);
569    q.enqueue(4890);
570    q.enqueue(4900);
571    q.enqueue(4910);
572    q.enqueue(4920);
573    q.enqueue(4930);
574    q.enqueue(4940);
575    q.enqueue(4950);
576    q.enqueue(4960);
577    q.enqueue(4970);
578    q.enqueue(4980);
579    q.enqueue(4990);
580    q.enqueue(5000);
581    q.enqueue(5010);
582    q.enqueue(5020);
583    q.enqueue(5030);
584    q.enqueue(5040);
585    q.enqueue(5050);
586    q.enqueue(5060);
587    q.enqueue(5070);
588    q.enqueue(5080);
589    q.enqueue(5090);
590    q.enqueue(5100);
591    q.enqueue(5110);
592    q.enqueue(5120);
593    q.enqueue(5130);
594    q.enqueue(5140);
595    q.enqueue(5150);
596    q.enqueue(5160);
597    q.enqueue(5170);
598    q.enqueue(5180);
599    q.enqueue(5190);
600    q.enqueue(5200);
601    q.enqueue(5210);
602    q.enqueue(5220);
603    q.enqueue(5230);
604    q.enqueue(5240);
605    q.enqueue(5250);
606    q.enqueue(5260);
607    q.enqueue(5270);
608    q.enqueue(5280);
609    q.enqueue(5290);
610    q.enqueue(5300);
611    q.enqueue(5310);
612    q.enqueue(5320);
613    q.enqueue(5330);
614    q.enqueue(5340);
615    q.enqueue(5350);
616    q.enqueue(5360);
617    q.enqueue(5370);
618    q.enqueue(5380);
619    q.enqueue(5390);
620    q.enqueue(5400);
621    q.enqueue(5410);
622    q.enqueue(5420);
623    q.enqueue(5430);
624    q.enqueue(5440);
625    q.enqueue(5450);
626    q.enqueue(5460);
627    q.enqueue(5470);
628    q.enqueue(5480);
629    q.enqueue(5490);
630    q.enqueue(5500);
631    q.enqueue(5510);
632    q.enqueue(5520);
633    q.enqueue(5530);
634    q.enqueue(5540);
635    q.enqueue(5550);
636    q.enqueue(5560);
637    q.enqueue(5570);
638    q.enqueue(5580);
639    q.enqueue(5590);
640    q.enqueue(5600);
641    q.enqueue(5610);
642    q.enqueue(5620);
643    q.enqueue(5630);
644    q.enqueue(5640);
645    q.enqueue(5650);
646    q.enqueue(5660);
647    q.enqueue(5670);
648    q.enqueue(5680);
649    q.enqueue(5690);
650    q.enqueue(5700);
651    q.enqueue(5710);
652    q.enqueue(5720);
653    q.enqueue(5730);
654    q.enqueue(5740);
655    q.enqueue(5750);
656    q.enqueue(5760);
657    q.enqueue(5770);
658    q.enqueue(5780);
659    q.enqueue(5790);
660    q.enqueue(5800);
661    q.enqueue(5810);
662    q.enqueue(5820);
663    q.enqueue(5830);
664    q.enqueue(5840);
665    q.enqueue(5850);
666    q.enqueue(5860);
667    q.enqueue(5870);
668    q.enqueue(5880);
669    q.enqueue(5890);
670    q.enqueue(5900);
671    q.enqueue(5910);
672    q.enqueue(5920);
673    q.enqueue(5930);
674    q.enqueue(5940);
675    q.enqueue(5950);
676    q.enqueue(5960);
677    q.enqueue(5970);
678    q.enqueue(5980);
679    q.enqueue(5990);
680    q.enqueue(6000);
681    q.enqueue(6010);
682    q.enqueue(6020);
683    q.enqueue(6030);
684    q.enqueue(6040);
685    q.enqueue(6050);
686    q.enqueue(6060);
687    q.enqueue(6070);
688    q.enqueue(6080);
689    q.enqueue(6090);
690    q.enqueue(6100);
691    q.enqueue(6110);
692    q.enqueue(6120);
693    q.enqueue(6130);
694    q.enqueue(6140);
695    q.enqueue(6150);
696    q.enqueue(6160);
697    q.enqueue(6170);
698    q.enqueue(6180);
699    q.enqueue(6190);
700    q.enqueue(6200);
701    q.enqueue(6210);
702    q.enqueue(6220);
703    q.enqueue(6230);
704    q.enqueue(6240);
705    q.enqueue(6250);
706    q.enqueue(6260);
707    q.enqueue(6270);
708    q.enqueue(6280);
709    q.enqueue(6290);
710    q.enqueue(6300);
711    q.enqueue(6310);
712    q.enqueue(6320);
713    q.enqueue(6330);
714    q.enqueue(6340);
715    q.enqueue(6350);
716    q.enqueue(6360);
717    q.enqueue(6370);
718    q.enqueue(6380);
719    q.enqueue(6390);
720    q.enqueue(6400);
721    q.enqueue(6410);
722    q.enqueue(6420);
723    q.enqueue(6430);
724    q.enqueue(6440);
725    q.enqueue(6450);
726    q.enqueue(6460);
727    q.enqueue(6470);
728    q.enqueue(6480);
729    q.enqueue(6490);
730    q.enqueue(6500);
731    q.enqueue(6510);
732    q.enqueue(6520);
733    q.enqueue(6530);
734    q.enqueue(6540);
735    q.enqueue(6550);
736    q.enqueue(6560);
737    q.enqueue(6570);
738    q.enqueue(6580);
739    q.enqueue(6590);
740    q.enqueue(6600);
741    q.enqueue(6610);
742    q.enqueue(6620);
743    q.enqueue(6630);
744    q.enqueue(6640);
745    q.enqueue(6650);
746    q.enqueue(6660);
747    q.enqueue(6670);
748    q.enqueue(6680);
749    q.enqueue(6690);
750    q.enqueue(6700);
751    q.enqueue(6710);
752    q.enqueue(6720);
753    q.enqueue(6730);
754    q.enqueue(6740);
755    q.enqueue(6750);
756    q.enqueue(6760);
757    q.enqueue(6770);
758    q.enqueue(6780);
759    q.enqueue(6790);
760    q.enqueue(6800);
761    q.enqueue(6810);
762    q.enqueue(6820);
763    q.enqueue(6830);
764    q.enqueue(6840);
765    q.enqueue(6850);
766    q.enqueue(6860);
767    q.enqueue(6870);
768    q.enqueue(6880);
769    q.enqueue(6890);
770    q.enqueue(6900);
771    q.enqueue(6910);
772    q.enqueue(6920);
773    q.enqueue(6930);
774    q.enqueue(6940);
775    q.enqueue(6950);
776    q.enqueue(6960);
777    q.enqueue(6970);
778    q.enqueue(6980);
779    q.enqueue(6990);
780    q.enqueue(7000);
781    q.enqueue(7010);
782    q.enqueue(7020);
783    q.enqueue(7030);
784    q.enqueue(7040);
785    q.enqueue(7050);
786    q.enqueue(7060);
787    q.enqueue(7070);
788    q.enqueue(7080);
789    q.enqueue(7090);
790    q.enqueue(7100);
791    q.enqueue(7110);
792    q.enqueue(7120);
793    q.enqueue(7130);
794    q.enqueue(7140);
795    q.enqueue(7150);
796    q.enqueue(7160);
797    q.enqueue(7170);
798    q.enqueue(7180);
799    q.enqueue(7190);
800    q.enqueue(7200);
801    q.enqueue(7210);
802    q.enqueue(7220);
803    q.enqueue(7230);
804    q.enqueue(7240);
805    q.enqueue(7250);
806    q.enqueue(7260);
807    q.enqueue(7270);
808    q.enqueue(7280);
809    q.enqueue(7290);
810    q.enqueue(7300);
811    q.enqueue(7310);
812    q.enqueue(7320);
813    q.enqueue(7330);
814    q.enqueue(7340);
815    q.enqueue(7350);
816    q.enqueue(7360);
817    q.enqueue(7370);
818    q.enqueue(7380);
819    q.enqueue(7390);
820    q.enqueue(7400);
821    q.enqueue(7410);
822    q.enqueue(7420);
823    q.enqueue(7430);
824    q.enqueue(7440);
825    q.enqueue(7450);
826    q.enqueue(7460);
827    q.enqueue(7470);
828    q.enqueue(7480);
829    q.enqueue(7490);
830    q.enqueue(7500);
831    q.enqueue(7510);
832    q.enqueue(7520);
833    q.enqueue(7530);
834    q.enqueue(7540);
835    q.enqueue(7550);
836    q.enqueue(7560);
837    q.enqueue(7570);
838    q.enqueue(7580);
839    q.enqueue(7590);
840    q.enqueue(7600);
841    q.enqueue(7610);
842    q.enqueue(7620);
843    q.enqueue(7630);
844    q.enqueue(7640);
845    q.enqueue(7650);
846    q.enqueue(7660);
847    q.enqueue(7670);
848    q.enqueue(7680);
849    q.enqueue(7690);
850    q.enqueue(7700);
851    q.enqueue(7710);
852    q.enqueue(7720);
853    q.enqueue(7730);
854    q.enqueue(7740);
855    q.enqueue(7750);
856    q.enqueue(7760);
857    q.enqueue(7770);
858    q.enqueue(7780);
859    q.enqueue(7790);
860    q.enqueue(7800);
861    q.enqueue(7810);
862    q.enqueue(7820);
863    q.enqueue(7830);
864    q.enqueue(7840);
865    q.enqueue(7850);
866    q.enqueue(7860);
867    q.enqueue(7870);
868    q.enqueue(7880);
869    q.enqueue(7890);
870    q.enqueue(7900);
871    q.enqueue(7910);
872    q.enqueue(7920);
873    q.enqueue(7930);
874    q.enqueue(7940);
875    q.enqueue(7950);
876    q.enqueue(7960);
877    q.enqueue(7970);
878    q.enqueue(7980);
879    q.enqueue(7990);
880    q.enqueue(8000);
881    q.enqueue(8010);
882    q.enqueue(8020);
883    q.enqueue(8030);
884    q.enqueue(8040);
885    q.enqueue(8050);
886    q.enqueue(8060);
887    q.enqueue(8070);
888    q.enqueue(8080);
889    q.enqueue(8090);
890    q.enqueue(8100);
891    q.enqueue(8110);
892    q.enqueue(8120);
893    q.enqueue(8130);
894    q.enqueue(8140);
895    q.enqueue(8150);
896    q.enqueue(8160);
897    q.enqueue(8170);
898    q.enqueue(8180);
899    q.enqueue(8190);
900    q.enqueue(8200);
901    q.enqueue(8210);
902    q.enqueue(8220);
903    q.enqueue(8230);
904    q.enqueue(8240);
905    q.enqueue(8250);
906    q.enqueue(8260);
907    q.enqueue(8270);
908    q.enqueue(8280);
909    q.enqueue(8290);
910    q.enqueue(8300);
911    q.enqueue(8310);
912    q.enqueue(8320);
913    q.enqueue(8330);
914    q.enqueue(8340);
915    q.enqueue(8350);
916    q.enqueue(8360);
917    q.enqueue(8370);
918    q.enqueue(8380);
919    q.enqueue(8390);
920    q.enqueue(8400);
921    q.enqueue(8410);
922   
```

```

28
29
30 bool full()
31 {
32     return sz == MAXN;
33 }
34
35 void enqueue(int value)
36 {
37     if (full())
38     {
39         cout << "Queue is full!" << endl;
40         return;
41     }
42     backIdx = (backIdx + 1) % MAXN;
43     arr[backIdx] = value;
44     sz++;
45 }
46
47 int dequeue()
48 {
49     if (empty())
50     {
51         cout << "Queue is empty!" << endl;
52         return -1;
53     }
54     int value = arr[frontIdx];
55     frontIdx = (frontIdx + 1) % MAXN;
56     sz--;
57     return value;
58 }
59
60 int front()
61 {
62     if (empty())
63     {
64         cout << "Queue is empty!" << endl;
65         return -1;
66     }
67     return arr[frontIdx];
68 }
69
70 int size()
71 {
72     return sz;
73 }
74
75 // ----- Ride History -----
76 class RideRecord
77 {
78 public:
79     int rideID;
80     int driverID;
81     int srcNode;
82     int destNode;
83     double distance;
84     double fare;
85     string status;
86     RideRecord* next;
87

```

```

85
86 string status;
87 RideRecord* next;
88
89 RideRecord(int r, int d, int s, int de, double dist, double f, string st)
90 {
91     rideID = r;
92     driverID = d;
93     srcNode = s;
94     destNode = de;
95     distance = dist;
96     fare = f;
97     status = st;
98     next = nullptr;
99 }
100
101 class RideHistory
102 {
103 private:
104     RideRecord* head;
105
106 public:
107     RideHistory()
108     {
109         head = nullptr;
110     }
111
112     void add(int rideID, int driverID, int src, int dest, double dist, double fare, string status)
113     {
114         RideRecord* node = new RideRecord(rideID, driverID, src, dest, dist, fare, status);
115         node->next = head;
116         head = node;
117     }
118
119     void printAll()
120     {
121         cout << "Ride History (most recent first):" << endl;
122         RideRecord* cur = head;
123         if (cur == NULL)
124         {
125             cout << " No rides yet." << endl;
126             return;
127         }
128         while (cur != NULL)
129         {
130             cout << " Ride " << cur->rideID
131                  << " | Driver " << cur->driverID
132                  << " | " << cur->srcNode << " -> " << cur->destNode
133                  << " | dist: " << cur->distance
134                  << " | fare: " << cur->fare
135                  << " | " << cur->status << endl;
136             cur = cur->next;
137         }
138     }
139
140     RideRecord* findByID(int rideID)
141     {
142         RideRecord* cur = head;
143         while (cur)
144         {

```

```

142     RideRecord* cur = head;
143     while (cur)
144     {
145         if (cur->rideID == rideID)
146             return cur;
147         cur = cur->next;
148     }
149     return nullptr;
150 }
151 };
152
153 // ----- Driver BST -----
154 class Driver
155 {
156 public:
157     int driverID;
158     string name;
159     int currentNode;
160     bool available;
161     double earnings;
162     Driver* left;
163     Driver* right;
164
165     Driver(int id = 0, string n = "", int node = 0, bool avail = true)
166     {
167         driverID = id;
168         name = n;
169         currentNode = node;
170         available = avail;
171         earnings = 0;
172         left = nullptr;
173         right = nullptr;
174     }
175 };
176
177 class DriverBST
178 {
179 private:
180     Driver* root;
181
182     Driver* insertRec(Driver* node, Driver* toIns)
183     {
184         if (!node)
185             return toIns;
186         if (toIns->driverID < node->driverID)
187             node->left = insertRec(node->left, toIns);
188         else if (toIns->driverID > node->driverID)
189             node->right = insertRec(node->right, toIns);
190         else
191         {
192             node->name = toIns->name;
193             node->currentNode = toIns->currentNode;
194             node->available = toIns->available;
195             node->earnings = toIns->earnings;
196         }
197         return node;
198     }
199
200     Driver* findRec(Driver* node, int id)
201     {

```



```

199 Driver* findRec(Driver* node, int id)
200 {
201     if (!node)
202         return nullptr;
203     if (id == node->driverID)
204         return node;
205     if (id < node->driverID)
206         return findRec(node->left, id);
207     return findRec(node->right, id);
208 }
209
210 Driver* deleteRec(Driver* node, int id)
211 {
212     if (!node)
213         return nullptr;
214     if (id < node->driverID)
215         node->left = deleteRec(node->left, id);
216     else if (id > node->driverID)
217         node->right = deleteRec(node->right, id);
218     else
219     {
220         if (!node->left)
221         {
222             Driver* r = node->right;
223             delete node;
224             return r;
225         }
226         if (!node->right)
227         {
228             Driver* l = node->left;
229             delete node;
230             return l;
231         }
232         Driver* succParent = node;
233         Driver* succ = node->right;
234         while (succ->left)
235         {
236             succParent = succ;
237             succ = succ->left;
238         }
239         node->driverID = succ->driverID;
240         node->name = succ->name;
241         node->currentNode = succ->currentNode;
242         node->available = succ->available;
243         node->earnings = succ->earnings;
244         if (succParent->left == succ)
245             succParent->left = succ->right;
246         else
247             succParent->right = succ->right;
248         delete succ;
249     }
250     return node;
251 }
252 void inorderRec(Driver* node)
253

```

```

256 }
257
258 void inorderRec(Driver* node)
259 {
260     if (!node)
261         return;
262     inorderRec(node->left);
263     cout << " ID: " << node->driverID
264         << " | Name: " << node->name
265         << " | Node: " << node->currentNode
266         << " | " << (node->available ? "Available" : "Busy")
267         << " | Earnings: " << node->earnings << endl;
268     inorderRec(node->right);
269 }
270
271 void collectAvailRec(Driver* node, Driver** arr, int& idx)
272 {
273     if (!node)
274         return;
275     collectAvailRec(node->left, arr, idx);
276     if (node->available)
277         arr[idx++] = node;
278     collectAvailRec(node->right, arr, idx);
279 }
280
281 public:
282 DriverBST()
283 {
284     root = nullptr;
285 }
286
287 void insert(int id, string name, int node, bool avail = true)
288 {
289     Driver* d = new Driver(id, name, node, avail);
290     root = insertRec(root, d);
291 }
292
293 Driver* find(int id)
294 {
295     return findRec(root, id);
296 }
297
298 void remove(int id)
299 {
300     root = deleteRec(root, id);
301 }
302
303 void inorderPrint()
304 {
305     if (!root)
306     {
307         cout << "No Drivers." << endl;
308         return;
309     }
310     cout << "Drivers (in-order by ID):" << endl;
311     inorderRec(root);
312 }
313
314 void collectAvailable(Driver** arr, int& count)
315 {

```

```
[*]main.cpp x
Y: 313
314 void collectAvailable(Driver** arr, int& count)
315 {
316     count = 0;
317     collectAvailRec(root, arr, count);
318 }
319 };
320
321 // ----- Graph & Dijkstra -----
322 class Graph
323 {
324 private:
325     int n;
326     double adj[100][100];
327
328 public:
329     Graph(int nodes = 0)
330     {
331         n = nodes;
332         for (int i = 0; i < 100; i++)
333             for (int j = 0; j < 100; j++)
334                 adj[i][j] = 1e18;
335     }
336
337     void resize(int nodes)
338     {
339         n = nodes;
340         for (int i = 0; i < 100; i++)
341             for (int j = 0; j < 100; j++)
342                 adj[i][j] = 1e18;
343     }
344
345     void addEdge(int u, int v, double w)
346     {
347         if (u < 1 || v < 1 || u > n || v > n)
348         {
349             cout << "Invalid node index" << endl;
350             return;
351         }
352         adj[u][v] = w;
353         adj[v][u] = w;
354     }
355
356     int size()
357     {
358         return n;
359     }
360
361     void dijkstra(int source, double dist[100])
362     {
363         bool visited[100] = { false };
364         for (int i = 1; i <= n; i++)
365             dist[i] = 1e18;
366         dist[source] = 0;
367
368         for (int count = 1; count <= n; count++)
369         {
370             int u = -1;
371             for (int i = 1; i <= n; i++)
372                 if (!visited[i] && (u == -1 || dist[i] < dist[u]))
373                     u = i;
374
375             if (dist[u] == 1e18)
376                 break;
377             visited[u] = true;
378
379             for (int v = 1; v <= n; v++)
380                 if (adj[u][v] < 1e17)
381                     if (dist[v] > dist[u] + adj[u][v])
382                         dist[v] = dist[u] + adj[u][v];
383         }
384
385         void print()
386         {
387             cout << "City map adjacency matrix:" << endl;
388             for (int i = 1; i <= n; i++)
389             {
390                 cout << i << ": ";
391                 for (int j = 1; j <= n; j++)
392                     if (adj[i][j] < 1e17)
393                         cout << j << " " << adj[i][j] << " ";
394                 cout << endl;
395             }
396         }
397     };
398
399 // ----- Cab Management -----
400 class CabSystem
401 {
402 private:
403     Graph graph;
404     DriverBST drivers;
405     RideHistory rideHistory;
406     int nextRideID;
407     double ratePerKm;
408     SimpleQueue pendingSrc;
409     SimpleQueue pendingDest;
410
411 public:
412     CabSystem()
413     {
414         graph.resize(0);
415         nextRideID = 1;
416         ratePerKm = 10.0;
417     }
418
419     void initGraph(int nodes)
420     {
421         graph.resize(nodes);
422     }
423
424     void addRoad(int u, int v, double w)
425     {
426         graph.addEdge(u, v, w);
427     }
428 }
429
sources Compile Log Debug Find Results Console
```

```

424
425 void addRoad(int u, int v, double w)
426 {
427     graph.addEdge(u, v, w);
428 }
429
430 void setRate(double r)
431 {
432     ratePerKm = r;
433 }
434
435 void addDriver(int id, string name, int node)
436 {
437     drivers.insert(id, name, node, true);
438 }
439
440 void removeDriver(int id)
441 {
442     drivers.remove(id);
443 }
444
445 void updateDriverLocation(int id, int node)
446 {
447     Driver* d = drivers.find(id);
448     if (!d)
449     {
450         cout << "Driver not found." << endl;
451         return;
452     }
453     d->currentNode = node;
454 }
455
456 void printDrivers()
457 {
458     drivers.inorderPrint();
459 }
460
461 void printMap()
462 {
463     graph.print();
464 }
465
466 void requestRide(int src, int dest)
467 {
468     if (src < 1 || dest < 1 || src > graph.size() || dest > graph.size())
469     {
470         cout << "Invalid source/destination nodes." << endl;
471         return;
472     }
473
474     cout << "Processing ride from " << src << " to " << dest << "." << endl;
475     double dist[100];
476     graph.dijkstra(src, dist);
477
478     Driver* availArr[100];
479     int availCount;
480     drivers.collectAvailable(availArr, availCount);
481
482     if (availCount == 0)
483     {

```

```

481
482     if (availCount == 0)
483     {
484         cout << "No available drivers. Added to pending queue." << endl;
485         pendingSrc.enqueue(src);
486         pendingDest.enqueue(dest);
487         return;
488     }
489
490     Driver* chosen = nullptr;
491     double minDist = 1e18;
492
493     for (int i = 0; i < availCount; i++)
494     {
495         if (dist[availArr[i]->currentNode] < minDist)
496         {
497             minDist = dist[availArr[i]->currentNode];
498             chosen = availArr[i];
499         }
500     }
501
502     if (!chosen || minDist > 1e17)
503     {
504         cout << "No reachable available drivers. Added to pending queue." << endl;
505         pendingSrc.enqueue(src);
506         pendingDest.enqueue(dest);
507         return;
508     }
509
510     chosen->available = false;
511     graph.dijkstra(src, dist);
512     double tripDist = dist[dest];
513
514     if (tripDist > 1e17)
515     {
516         cout << "Destination unreachable. Cancelling." << endl;
517         chosen->available = true;
518         return;
519     }
520
521     double fare = tripDist * ratePerKm;
522     int rideID = nextRideID++;
523
524     // Ride starts as ongoing
525     rideHistory.add(rideID, chosen->driverID, src, dest, tripDist, fare, "Ongoing");
526
527     cout << "Assigned Driver " << chosen->driverID << " (" << chosen->name << ")." << endl;
528     cout << "Estimated distance: " << tripDist << " km, fare: " << fare << endl;
529 }
530
531 void completeRide(int rideID)
532 {
533     RideRecord* ride = rideHistory.findByID(rideID);
534     if (!ride)
535     {
536         cout << "Ride not found." << endl;
537         return;
538     }
539     if (ride->status != "Ongoing")
540     {
541         cout << "Ride not ongoing. Current status: " << ride->status << endl;
542         return;
543     }

```

```

[*]main.cpp x
541     }
542
543     ride->status = "Completed";
544
545     Driver* d = drivers.find(ride->driverID);
546     if (d)
547     {
548         d->currentNode = ride->destNode;
549         d->earnings += ride->fare;
550         d->available = true;
551     }
552
553     cout << "Ride " << rideID << " completed." << endl;
554 }
555
556 void cancelRide(int rideID)
557 {
558     RideRecord* ride = rideHistory.findByID(rideID);
559     if (!ride)
560     {
561         cout << "Ride ID " << rideID << " not found." << endl;
562         return;
563     }
564     if (ride->status != "Ongoing")
565     {
566         cout << "Cannot cancel ride. Status: " << ride->status << endl;
567         return;
568     }
569
570     ride->status = "Cancelled";
571
572     Driver* d = drivers.find(ride->driverID);
573     if (d)
574         d->available = true;
575
576     cout << "Ride " << rideID << " has been cancelled." << endl;
577 }
578
579 void processPendingRequests()
580 {
581     int qsize = pendingSrc.size();
582     for (int i = 0; i < qsize; i++)
583     {
584         int src = pendingSrc.dequeue();
585         int dest = pendingDest.dequeue();
586         cout << "Trying pending request " << src << " -> " << dest << "..." << endl;
587         requestRide(src, dest);
588     }
589 }
590
591 void showRideHistory()
592 {
593     rideHistory.printAll();
594 }
595 };
596
597 // ----- Menu -----
598
599 // ----- Main -----
600

```

```

[*]main.cpp x
598
599 // ----- Main -----
600 int main() {
601
602     CabSystem sys;
603     int nodes;
604     cout << "Initialize city map (number of nodes): ";
605     cin >> nodes;
606     sys.initGraph(nodes);
607
608     int m;
609     cout << "Create roads (enter m edges): ";
610     cin >> m;
611
612     for (int i = 0; i < m; i++)
613     {
614         int u, v;
615         double w;
616         cout << "Edge " << i + 1 << " u v w: ";
617         cin >> u >> v >> w;
618         sys.addRoad(u, v, w);
619     }
620
621     sys.setRate(10.0);
622
623     while (true)
624     {
625         cout << "\n--- Cab Management Menu ---" << endl;
626         cout << "1. Add Driver" << endl;
627         cout << "2. Remove Driver" << endl;
628         cout << "3. Update Driver Location" << endl;
629         cout << "4. Show Drivers" << endl;
630         cout << "5. Show Map" << endl;
631         cout << "6. Request Ride" << endl;
632         cout << "7. Process Pending Requests" << endl;
633         cout << "8. Show Ride History" << endl;
634         cout << "9. Cancel Ride" << endl;
635         cout << "10. Complete Ride" << endl;
636         cout << "11. Exit" << endl;
637         cout << "Choose: ";
638
639         int ch;
640         cin >> ch;
641
642         if (ch == 1)
643         {
644             int id, node;
645             string name;
646             cout << "Driver ID: ";
647             cin >> id;
648             cout << "Name: ";
649             cin.ignore();
650             getline(cin, name);
651             cout << "Node: ";
652             cin >> node;
653             sys.addDriver(id, name, node);
654         }
655     }
656 }
657

```

```

651         cout << "Name: ";
652         cin.ignore();
653         getline(cin, name);
654         cout << "Node: ";
655         cin >> node;
656         sys.addDriver(id, name, node);
657     }
658     else if (ch == 2)
659     {
660         int id;
661         cout << "Driver ID to remove: ";
662         cin >> id;
663         sys.removeDriver(id);
664     }
665     else if (ch == 3)
666     {
667         int id, node;
668         cout << "Driver ID: ";
669         cin >> id;
670         cout << "New node: ";
671         cin >> node;
672         sys.updateDriverLocation(id, node);
673     }
674     else if (ch == 4)
675         sys.printDrivers();
676     else if (ch == 5)
677         sys.printMap();
678     else if (ch == 6)
679     {
680         int s, d;
681         cout << "Source node: ";
682         cin >> s;
683         cout << "Destination node: ";
684         cin >> d;
685         sys.requestRide(s, d);
686     }
687     else if (ch == 7)
688         sys.processPendingRequests();
689     else if (ch == 8)
690         sys.showRideHistory();
691     else if (ch == 9)
692     {
693         int rideID;
694         cout << "Ride ID to cancel: ";
695         cin >> rideID;
696         sys.cancelRide(rideID);
697     }
698     else if (ch == 10)
699     {
700         int rideID;
701         cout << "Ride ID to complete: ";
702         cin >> rideID;
703         sys.completeRide(rideID);
704     }
705     else
706         break;
707 }
708 }
709
710

```

6. Output:

```
C:\Users\hnp\Desktop\3 x + v
Initialize city map (number of nodes): 4
Create roads (enter m edges): 3
Edge 1 u v w: 1 2 5
Edge 2 u v w: 2 3 2
Edge 3 u v w: 3 4 3

--- Cab Management Menu ---
1. Add Driver
2. Remove Driver
3. Update Driver Location
4. Show Drivers
5. Show Map
6. Request Ride
7. Process Pending Requests
8. Show Ride History
9. Cancel Ride
10. Complete Ride
11. Exit
Choose: 1
Driver ID: 123
Name: Anis
Node: 1

--- Cab Management Menu ---
1. Add Driver
2. Remove Driver
3. Update Driver Location
4. Show Drivers
5. Show Map
6. Request Ride
7. Process Pending Requests
8. Show Ride History
9. Cancel Ride
10. Complete Ride
11. Exit
Choose: 1
Driver ID: 232
Name: Waleed
Node: 3

--- Cab Management Menu ---
1. Add Driver
```

C:\Users\hp\Desktop\3 x + v

--- Cab Management Menu ---

1. Add Driver
2. Remove Driver
3. Update Driver Location
4. Show Drivers
5. Show Map
6. Request Ride
7. Process Pending Requests
8. Show Ride History
9. Cancel Ride
10. Complete Ride
11. Exit

Choose: 2

Driver ID to remove: 123

--- Cab Management Menu ---

1. Add Driver
2. Remove Driver
3. Update Driver Location
4. Show Drivers
5. Show Map
6. Request Ride
7. Process Pending Requests
8. Show Ride History
9. Cancel Ride
10. Complete Ride
11. Exit

Choose: 3

Driver ID: 232

New node: 1

--- Cab Management Menu ---

1. Add Driver
2. Remove Driver
3. Update Driver Location
4. Show Drivers
5. Show Map
6. Request Ride
7. Process Pending Requests
8. Show Ride History
9. Cancel Ride

C:\Users\hp\Desktop\3 x + v

8. Show Ride History

9. Cancel Ride

10. Complete Ride

11. Exit

Choose: 4

Drivers (in-order by ID):

ID: 232 | Name: Waleed | Node: 1 | Available | Earnings: 0

--- Cab Management Menu ---

1. Add Driver
2. Remove Driver
3. Update Driver Location
4. Show Drivers
5. Show Map
6. Request Ride
7. Process Pending Requests
8. Show Ride History
9. Cancel Ride
10. Complete Ride
11. Exit

Choose: 5

City map adjacency matrix:

1: 2(5)

2: 1(5) 3(2)

3: 2(2) 4(3)

4: 3(3)

--- Cab Management Menu ---

1. Add Driver
2. Remove Driver
3. Update Driver Location
4. Show Drivers
5. Show Map
6. Request Ride
7. Process Pending Requests
8. Show Ride History
9. Cancel Ride
10. Complete Ride
11. Exit

Choose: 6

Source node: 1

Destination node: 3

```

C:\Users\hp\Desktop\3 x + v
Destination node: 3
Processing ride from 1 to 3...
Assigned Driver 232 (Waleed).
Estimated distance: 7 km, fare: 70

--- Cab Management Menu ---
1. Add Driver
2. Remove Driver
3. Update Driver Location
4. Show Drivers
5. Show Map
6. Request Ride
7. Process Pending Requests
8. Show Ride History
9. Cancel Ride
10. Complete Ride
11. Exit
Choose: 7

--- Cab Management Menu ---
1. Add Driver
2. Remove Driver
3. Update Driver Location
4. Show Drivers
5. Show Map
6. Request Ride
7. Process Pending Requests
8. Show Ride History
9. Cancel Ride
10. Complete Ride
11. Exit
Choose: 8
Ride History (most recent first):
  Ride 1 | Driver 232 | 1 -> 3 | dist: 7 | fare: 70 | Ongoing

--- Cab Management Menu ---
1. Add Driver
2. Remove Driver
3. Update Driver Location
4. Show Drivers
5. Show Map
6. Request Ride

```

```

C:\Users\hp\Desktop\3 x + v
7. Process Pending Requests
8. Show Ride History
9. Cancel Ride
10. Complete Ride
11. Exit
Choose: 8
Ride History (most recent first):
  Ride 1 | Driver 232 | 1 -> 3 | dist: 7 | fare: 70 | Ongoing

--- Cab Management Menu ---
1. Add Driver
2. Remove Driver
3. Update Driver Location
4. Show Drivers
5. Show Map
6. Request Ride
7. Process Pending Requests
8. Show Ride History
9. Cancel Ride
10. Complete Ride
11. Exit
Choose: 10
Ride ID to complete: 1
Ride 1 completed.

--- Cab Management Menu ---
1. Add Driver
2. Remove Driver
3. Update Driver Location
4. Show Drivers
5. Show Map
6. Request Ride
7. Process Pending Requests
8. Show Ride History
9. Cancel Ride
10. Complete Ride
11. Exit
Choose: 11

-----
Process exited after 178.6 seconds with return value 0
Press any key to continue . . .

```


7. Learning Outcomes

After working on this project, I gain experience in:

- **Object-Oriented Design:** Defining classes (CabSystem, DriverBST, RideHistory, Graph) and their interactions. Designing methods for specific operations (adding drivers, handling rides).
- **Data Structures:** Implementing a BST for driver management, a linked list for ride history, a queue for pending requests, and a graph (adjacency matrix) for the city map. Understanding when and why each structure is appropriate (e.g. BST for sorted access linked list for dynamic records, queue for FIFO order).
- **Algorithms:** Applying Dijkstra's algorithm to find shortest paths in the graph. This reinforces knowledge of graph algorithms and their implementation in C++.
- **C++ Programming:** Using pointers, dynamic memory, and class encapsulation. (The code builds custom data structures rather than relying on STL, reinforcing how these structures work under the hood.)
- **Problem Solving:** Integrating multiple components to fulfill complex requirements (driver matching, request queuing, route finding). Developing logic to handle edge cases (no available drivers, unreachable destinations, ride cancellation).
- **Debugging and Testing:** Running sample scenarios, checking menu driven I/O, and verifying outputs (as shown in the example).

Overall, the Cab Booking System project provides practical experience with key computer science concepts (data structures, algorithms) and C++ programming techniques in a realistic application setting.

8. Conclusion

The Cab Booking System project successfully demonstrates the integration of object-oriented programming and fundamental data structures in solving a real-world problem. By modeling a city map using graphs, managing drivers with a binary search tree, handling ride requests through queues, and recording trip details via linked lists, the project reflects a comprehensive understanding of data structures and algorithmic logic in C++.

Through this project, we not only built a functional ride-booking simulation but also gained practical experience in designing interconnected systems, implementing efficient algorithms like Dijkstra's for shortest path calculation, and applying concepts of encapsulation, modularity, and dynamic data handling. This system can serve as a foundational framework for more advanced transport or logistics applications, potentially expandable to include file storage, real-time driver tracking, or user accounts.

Overall, the project highlights the power of C++ for structured problem-solving and lays a strong foundation for further exploration into complex system development.