



Data Structure & Object-Oriented Programming LAB

COMPLEX ENGINEERING ACTIVITY CAB Booking System

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

SESSION:

Fall 25

DATE OF REPORT SUBMITTED:
25 December 2025

GRADE/POINTS:

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.

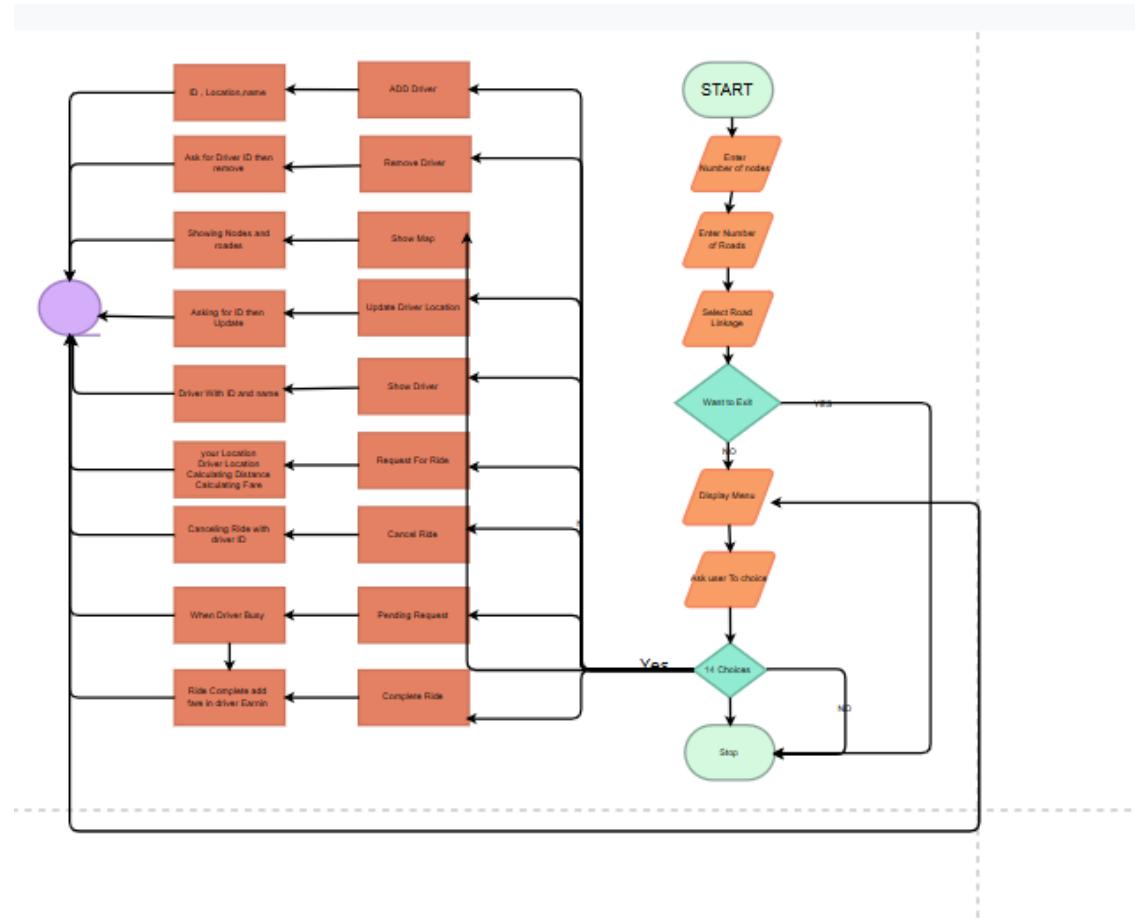
showRideHistory(): Displays all recorded rides.

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:

```

main.cpp X
1 #include <iostream>
2 #include <string>
3 #include <sstream>
4 using namespace std;
5
6 // ----- Simple Queue -----
7 class SimpleQueue
8 {
9 private:
10     static const int MAXN = 100;
11     int arr[MAXN];
12     int frontIdx;
13     int backIdx;
14     int sz;
15
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;
  
```

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

```
[ls] main.cpp x [ ]  
85     string status;  
86     RideRecord* next;  
87  
88     RideRecord(int r, int d, int s, int de, double dist, double f, string st)  
89     {  
90         rideID = r;  
91         driverID = d;  
92         srcNode = s;  
93         destNode = de;  
94         distance = dist;  
95         fare = f;  
96         status = st;  
97         next = nullptr;  
98     }  
99 }  
100  
101 class RideHistory  
102 {  
103 private:  
104     RideRecord* head;  
105 public:  
106     RideHistory()  
107     {  
108         head = nullptr;  
109     }  
110  
111     void add(int rideID, int driverID, int src, int dest, double dist, double fare, string status)  
112     {  
113         RideRecord* node = new RideRecord(rideID, driverID, src, dest, dist, fare, status);  
114         node->next = head;  
115         head = node;  
116     }  
117  
118     void printAll()  
119     {  
120         cout << "Ride History (most recent first):" << endl;  
121         RideRecord* cur = head;  
122         if (cur == NULL)  
123         {  
124             cout << " No rides yet." << endl;  
125             return;  
126         }  
127         while (cur != NULL)  
128         {  
129             cout << " Ride " << cur->rideID  
130             << " | Driver " << cur->driverID  
131             << " | " << cur->srcNode << " -> " << cur->destNode  
132             << " | dist: " << cur->distance  
133             << " | fare: " << cur->fare  
134             << " | " << cur->status << endl;  
135             cur = cur->next;  
136         }  
137     }  
138  
139     RideRecord* findById(int rideID)  
140     {  
141         RideRecord* cur = head;  
142         while (cur)  
143         {  
144             if (cur->rideID == rideID)  
145                 return cur;  
146             cur = cur->next;  
147         }  
148     }  
149 }
```

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

```

199
200
201     Driver* findRec(Driver* node, int id)
202     {
203         if (!node)
204             return nullptr;
205         if (id < node->driverID)
206             return node;
207         if (id > node->driverID)
208             return findRec(node->left, id);
209         return findRec(node->right, id);
210     }
211
212     Driver* deleteRec(Driver* node, int id)
213     {
214         if (!node)
215             return nullptr;
216         if (id < node->driverID)
217             node->left = deleteRec(node->left, id);
218         else if (id > node->driverID)
219             node->right = deleteRec(node->right, id);
220         else
221         {
222             if (!node->left)
223             {
224                 Driver* r = node->right;
225                 delete node;
226                 return r;
227             }
228             if (!node->right)
229             {
230                 Driver* l = node->left;
231                 delete node;
232                 return l;
233             }
234             Driver* succParent = node;
235             Driver* succ = node->right;
236             while (succ->left)
237             {
238                 succParent = succ;
239                 succ = succ->left;
240             }
241             node->driverID = succ->driverID;
242             node->name = succ->name;
243             node->currentNode = succ->currentNode;
244             node->available = succ->available;
245             node->earnings = succ->earnings;
246
247             if (succParent->left == succ)
248                 succParent->left = succ->right;
249             else
250                 succParent->right = succ->right;
251
252             delete succ;
253         }
254     }
255     return node;
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     void collectAvailable(Driver** arr, int& count)
314     {
315         count = 0;
316         collectAvailRec(root, arr, count);
317     }
318 }
319
320 // ----- Graph & Dijkstra -----
321 class Graph
322 {
323     private:
324         int n;
325         double adj[100][100];
326
327     public:
328         Graph(int nodes = 0)
329         {
330             n = nodes;
331             for (int i = 0; i < 100; i++)
332                 for (int j = 0; j < 100; j++)
333                     adj[i][j] = 1e18;
334         }
335
336         void resize(int nodes)
337         {
338             n = nodes;
339             for (int i = 0; i < 100; i++)
340                 for (int j = 0; j < 100; j++)
341                     adj[i][j] = 1e18;
342         }
343
344         void addEdge(int u, int v, double w)
345         {
346             if (u < 1 || v < 1 || u > n || v > n)
347             {
348                 cout << "Invalid node index" << endl;
349                 return;
350             }
351             adj[u][v] = w;
352             adj[v][u] = w;
353         }
354
355         int size()
356         {
357             return n;
358         }
359
360         void dijkstra(int source, double dist[100])
361         {
362             bool visited[100] = { false };
363             for (int i = 1; i <= n; i++)
364                 dist[i] = 1e18;
365             dist[source] = 0;
366
367             for (int count = 1; count <= n; count++)
368             {
369                 int u = -1;
370                 for (int i = 1; i <= n; i++)
371                     if (!visited[i] && (u == -1 || dist[i] < dist[u]))
372
[*] main.cpp x
SY: 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
386         void print()
387         {
388             cout << "City map adjacency matrix:" << endl;
389             for (int i = 1; i <= n; i++)
390             {
391                 cout << i << ": ";
392                 for (int j = 1; j <= n; j++)
393                     if (adj[i][j] < 1e17)
394                         cout << j << "(" << adj[i][j] << ")";
395                 cout << endl;
396             }
397         }
398     };
399
400 // ----- Cab Management -----
401 class CabSystem
402 {
403     private:
404         Graph graph;
405         DriverBST drivers;
406         RideHistory rideHistory;
407         int nextRideID;
408         double ratePerKm;
409         SimpleQueue pendingSrc;
410         SimpleQueue pendingDest;
411
412     public:
413         CabSystem()
414         {
415             graph.resize(0);
416             nextRideID = 1;
417             ratePerKm = 10.0;
418         }
419
420         void initGraph(int nodes)
421         {
422             graph.resize(nodes);
423         }
424
425         void addRoad(int u, int v, double w)
426         {
427             graph.addEdge(u, v, w);
428         }
429

```

```

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         {
484             if (availCount == 0)
485             {
486                 cout << "No available drivers. Added to pending queue." << endl;
487                 pendingSrc.enqueue(src);
488                 pendingDest.enqueue(dest);
489                 return;
490             }
491
492             Driver* chosen = nullptr;
493             double minDist = 1e18;
494
495             for (int i = 0; i < availCount; i++)
496                 if (dist[availArr[i]->currentNode] < minDist)
497                 {
498                     minDist = dist[availArr[i]->currentNode];
499                     chosen = availArr[i];
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         void completeRide(int rideID)
531         {
532             RideRecord* ride = rideHistory.findById(rideID);
533             if (!ride)
534             {
535                 cout << "Ride not found." << endl;
536                 return;
537             }
538             if (ride->status != "Ongoing")
539             {
540                 cout << "Ride not ongoing. Current status: " << ride->status << endl;
541                 return;
542             }
543
544             cout << "Ride completed successfully." << endl;
545             cout << "Driver ID: " << ride->driverID << endl;
546             cout << "Ride ID: " << ride->rideID << endl;
547             cout << "Start Node: " << ride->startNode << endl;
548             cout << "End Node: " << ride->endNode << endl;
549             cout << "Distance: " << ride->distance << endl;
550             cout << "Fare: " << ride->fare << endl;
551             cout << "Status: " << ride->status << endl;
552
553             cout << "Ride details saved to history." << endl;
554         }
555
556     }
557
558     void printRideHistory()
559     {
560         rideHistory.print();
561     }
562
563     void printPendingQueue()
564     {
565         pendingSrc.print();
566         pendingDest.print();
567     }
568
569     void printDriverStatus()
570     {
571         drivers.printStatus();
572     }
573
574     void printGraph()
575     {
576         graph.print();
577     }
578
579     void printDriverList()
580     {
581         drivers.printList();
582     }
583
584     void printRideList()
585     {
586         rideHistory.printList();
587     }
588
589     void printPendingList()
590     {
591         pendingSrc.printList();
592         pendingDest.printList();
593     }
594
595     void printDriverDetails()
596     {
597         drivers.printDetails();
598     }
599
600     void printRideDetails()
601     {
602         rideHistory.printDetails();
603     }
604
605     void printPendingDetails()
606     {
607         pendingSrc.printDetails();
608         pendingDest.printDetails();
609     }
610
611     void printDriverStatus()
612     {
613         drivers.printStatus();
614     }
615
616     void printGraph()
617     {
618         graph.print();
619     }
620
621     void printDriverList()
622     {
623         drivers.printList();
624     }
625
626     void printRideList()
627     {
628         rideHistory.printList();
629     }
630
631     void printPendingList()
632     {
633         pendingSrc.printList();
634         pendingDest.printList();
635     }
636
637     void printDriverDetails()
638     {
639         drivers.printDetails();
640     }
641
642     void printRideDetails()
643     {
644         rideHistory.printDetails();
645     }
646
647     void printPendingDetails()
648     {
649         pendingSrc.printDetails();
650         pendingDest.printDetails();
651     }
652
653     void printDriverStatus()
654     {
655         drivers.printStatus();
656     }
657
658     void printGraph()
659     {
660         graph.print();
661     }
662
663     void printDriverList()
664     {
665         drivers.printList();
666     }
667
668     void printRideList()
669     {
670         rideHistory.printList();
671     }
672
673     void printPendingList()
674     {
675         pendingSrc.printList();
676         pendingDest.printList();
677     }
678
679     void printDriverDetails()
680     {
681         drivers.printDetails();
682     }
683
684     void printRideDetails()
685     {
686         rideHistory.printDetails();
687     }
688
689     void printPendingDetails()
690     {
691         pendingSrc.printDetails();
692         pendingDest.printDetails();
693     }
694
695     void printDriverStatus()
696     {
697         drivers.printStatus();
698     }
699
700     void printGraph()
701     {
702         graph.print();
703     }
704
705     void printDriverList()
706     {
707         drivers.printList();
708     }
709
710     void printRideList()
711     {
712         rideHistory.printList();
713     }
714
715     void printPendingList()
716     {
717         pendingSrc.printList();
718         pendingDest.printList();
719     }
720
721     void printDriverDetails()
722     {
723         drivers.printDetails();
724     }
725
726     void printRideDetails()
727     {
728         rideHistory.printDetails();
729     }
730
731     void printPendingDetails()
732     {
733         pendingSrc.printDetails();
734         pendingDest.printDetails();
735     }
736
737     void printDriverStatus()
738     {
739         drivers.printStatus();
740     }
741
742     void printGraph()
743     {
744         graph.print();
745     }
746
747     void printDriverList()
748     {
749         drivers.printList();
750     }
751
752     void printRideList()
753     {
754         rideHistory.printList();
755     }
756
757     void printPendingList()
758     {
759         pendingSrc.printList();
760         pendingDest.printList();
761     }
762
763     void printDriverDetails()
764     {
765         drivers.printDetails();
766     }
767
768     void printRideDetails()
769     {
770         rideHistory.printDetails();
771     }
772
773     void printPendingDetails()
774     {
775         pendingSrc.printDetails();
776         pendingDest.printDetails();
777     }
778
779     void printDriverStatus()
780     {
781         drivers.printStatus();
782     }
783
784     void printGraph()
785     {
786         graph.print();
787     }
788
789     void printDriverList()
790     {
791         drivers.printList();
792     }
793
794     void printRideList()
795     {
796         rideHistory.printList();
797     }
798
799     void printPendingList()
800     {
801         pendingSrc.printList();
802         pendingDest.printList();
803     }
804
805     void printDriverDetails()
806     {
807         drivers.printDetails();
808     }
809
810     void printRideDetails()
811     {
812         rideHistory.printDetails();
813     }
814
815     void printPendingDetails()
816     {
817         pendingSrc.printDetails();
818         pendingDest.printDetails();
819     }
820
821     void printDriverStatus()
822     {
823         drivers.printStatus();
824     }
825
826     void printGraph()
827     {
828         graph.print();
829     }
830
831     void printDriverList()
832     {
833         drivers.printList();
834     }
835
836     void printRideList()
837     {
838         rideHistory.printList();
839     }
840
841     void printPendingList()
842     {
843         pendingSrc.printList();
844         pendingDest.printList();
845     }
846
847     void printDriverDetails()
848     {
849         drivers.printDetails();
850     }
851
852     void printRideDetails()
853     {
854         rideHistory.printDetails();
855     }
856
857     void printPendingDetails()
858     {
859         pendingSrc.printDetails();
860         pendingDest.printDetails();
861     }
862
863     void printDriverStatus()
864     {
865         drivers.printStatus();
866     }
867
868     void printGraph()
869     {
870         graph.print();
871     }
872
873     void printDriverList()
874     {
875         drivers.printList();
876     }
877
878     void printRideList()
879     {
880         rideHistory.printList();
881     }
882
883     void printPendingList()
884     {
885         pendingSrc.printList();
886         pendingDest.printList();
887     }
888
889     void printDriverDetails()
890     {
891         drivers.printDetails();
892     }
893
894     void printRideDetails()
895     {
896         rideHistory.printDetails();
897     }
898
899     void printPendingDetails()
900     {
901         pendingSrc.printDetails();
902         pendingDest.printDetails();
903     }
904
905     void printDriverStatus()
906     {
907         drivers.printStatus();
908     }
909
910     void printGraph()
911     {
912         graph.print();
913     }
914
915     void printDriverList()
916     {
917         drivers.printList();
918     }
919
920     void printRideList()
921     {
922         rideHistory.printList();
923     }
924
925     void printPendingList()
926     {
927         pendingSrc.printList();
928         pendingDest.printList();
929     }
930
931     void printDriverDetails()
932     {
933         drivers.printDetails();
934     }
935
936     void printRideDetails()
937     {
938         rideHistory.printDetails();
939     }
940
941     void printPendingDetails()
942     {
943         pendingSrc.printDetails();
944         pendingDest.printDetails();
945     }
946
947     void printDriverStatus()
948     {
949         drivers.printStatus();
950     }
951
952     void printGraph()
953     {
954         graph.print();
955     }
956
957     void printDriverList()
958     {
959         drivers.printList();
960     }
961
962     void printRideList()
963     {
964         rideHistory.printList();
965     }
966
967     void printPendingList()
968     {
969         pendingSrc.printList();
970         pendingDest.printList();
971     }
972
973     void printDriverDetails()
974     {
975         drivers.printDetails();
976     }
977
978     void printRideDetails()
979     {
980         rideHistory.printDetails();
981     }
982
983     void printPendingDetails()
984     {
985         pendingSrc.printDetails();
986         pendingDest.printDetails();
987     }
988
989     void printDriverStatus()
990     {
991         drivers.printStatus();
992     }
993
994     void printGraph()
995     {
996         graph.print();
997     }
998
999     void printDriverList()
1000    {
1001        drivers.printList();
1002    }
1003
1004    void printRideList()
1005    {
1006        rideHistory.printList();
1007    }
1008
1009    void printPendingList()
1010    {
1011        pendingSrc.printList();
1012        pendingDest.printList();
1013    }
1014
1015    void printDriverDetails()
1016    {
1017        drivers.printDetails();
1018    }
1019
1020    void printRideDetails()
1021    {
1022        rideHistory.printDetails();
1023    }
1024
1025    void printPendingDetails()
1026    {
1027        pendingSrc.printDetails();
1028        pendingDest.printDetails();
1029    }
1030
1031    void printDriverStatus()
1032    {
1033        drivers.printStatus();
1034    }
1035
1036    void printGraph()
1037    {
1038        graph.print();
1039    }
1040
1041    void printDriverList()
1042    {
1043        drivers.printList();
1044    }
1045
1046    void printRideList()
1047    {
1048        rideHistory.printList();
1049    }
1050
1051    void printPendingList()
1052    {
1053        pendingSrc.printList();
1054        pendingDest.printList();
1055    }
1056
1057    void printDriverDetails()
1058    {
1059        drivers.printDetails();
1060    }
1061
1062    void printRideDetails()
1063    {
1064        rideHistory.printDetails();
1065    }
1066
1067    void printPendingDetails()
1068    {
1069        pendingSrc.printDetails();
1070        pendingDest.printDetails();
1071    }
1072
1073    void printDriverStatus()
1074    {
1075        drivers.printStatus();
1076    }
1077
1078    void printGraph()
1079    {
1080        graph.print();
1081    }
1082
1083    void printDriverList()
1084    {
1085        drivers.printList();
1086    }
1087
1088    void printRideList()
1089    {
1090        rideHistory.printList();
1091    }
1092
1093    void printPendingList()
1094    {
1095        pendingSrc.printList();
1096        pendingDest.printList();
1097    }
1098
1099    void printDriverDetails()
1100    {
1101        drivers.printDetails();
1102    }
1103
1104    void printRideDetails()
1105    {
1106        rideHistory.printDetails();
1107    }
1108
1109    void printPendingDetails()
1110    {
1111        pendingSrc.printDetails();
1112        pendingDest.printDetails();
1113    }
1114
1115    void printDriverStatus()
1116    {
1117        drivers.printStatus();
1118    }
1119
1120    void printGraph()
1121    {
1122        graph.print();
1123    }
1124
1125    void printDriverList()
1126    {
1127        drivers.printList();
1128    }
1129
1130    void printRideList()
1131    {
1132        rideHistory.printList();
1133    }
1134
1135    void printPendingList()
1136    {
1137        pendingSrc.printList();
1138        pendingDest.printList();
1139    }
1140
1141    void printDriverDetails()
1142    {
1143        drivers.printDetails();
1144    }
1145
1146    void printRideDetails()
1147    {
1148        rideHistory.printDetails();
1149    }
1150
1151    void printPendingDetails()
1152    {
1153        pendingSrc.printDetails();
1154        pendingDest.printDetails();
1155    }
1156
1157    void printDriverStatus()
1158    {
1159        drivers.printStatus();
1160    }
1161
1162    void printGraph()
1163    {
1164        graph.print();
1165    }
1166
1167    void printDriverList()
1168    {
1169        drivers.printList();
1170    }
1171
1172    void printRideList()
1173    {
1174        rideHistory.printList();
1175    }
1176
1177    void printPendingList()
1178    {
1179        pendingSrc.printList();
1180        pendingDest.printList();
1181    }
1182
1183    void printDriverDetails()
1184    {
1185        drivers.printDetails();
1186    }
1187
1188    void printRideDetails()
1189    {
1190        rideHistory.printDetails();
1191    }
1192
1193    void printPendingDetails()
1194    {
1195        pendingSrc.printDetails();
1196        pendingDest.printDetails();
1197    }
1198
1199    void printDriverStatus()
1200    {
1201        drivers.printStatus();
1202    }
1203
1204    void printGraph()
1205    {
1206        graph.print();
1207    }
1208
1209    void printDriverList()
1210    {
1211        drivers.printList();
1212    }
1213
1214    void printRideList()
1215    {
1216        rideHistory.printList();
1217    }
1218
1219    void printPendingList()
1220    {
1221        pendingSrc.printList();
1222        pendingDest.printList();
1223    }
1224
1225    void printDriverDetails()
1226    {
1227        drivers.printDetails();
1228    }
1229
1230    void printRideDetails()
1231    {
1232        rideHistory.printDetails();
1233    }
1234
1235    void printPendingDetails()
1236    {
1237        pendingSrc.printDetails();
1238        pendingDest.printDetails();
1239    }
1240
1241    void printDriverStatus()
1242    {
1243        drivers.printStatus();
1244    }
1245
1246    void printGraph()
1247    {
1248        graph.print();
1249    }
1250
1251    void printDriverList()
1252    {
1253        drivers.printList();
1254    }
1255
1256    void printRideList()
1257    {
1258        rideHistory.printList();
1259    }
1260
1261    void printPendingList()
1262    {
1263        pendingSrc.printList();
1264        pendingDest.printList();
1265    }
1266
1267    void printDriverDetails()
1268    {
1269        drivers.printDetails();
1270    }
1271
1272    void printRideDetails()
1273    {
1274        rideHistory.printDetails();
1275    }
1276
1277    void printPendingDetails()
1278    {
1279        pendingSrc.printDetails();
1280        pendingDest.printDetails();
1281    }
1282
1283    void printDriverStatus()
1284    {
1285        drivers.printStatus();
1286    }
1287
1288    void printGraph()
1289    {
1290        graph.print();
1291    }
1292
1293    void printDriverList()
1294    {
1295        drivers.printList();
1296    }
1297
1298    void printRideList()
1299    {
1300        rideHistory.printList();
1301    }
1302
1303    void printPendingList()
1304    {
1305        pendingSrc.printList();
1306        pendingDest.printList();
1307    }
1308
1309    void printDriverDetails()
1310    {
1311        drivers.printDetails();
1312    }
1313
1314    void printRideDetails()
1315    {
1316        rideHistory.printDetails();
1317    }
1318
1319    void printPendingDetails()
1320    {
1321        pendingSrc.printDetails();
1322        pendingDest.printDetails();
1323    }
1324
1325    void printDriverStatus()
1326    {
1327        drivers.printStatus();
1328    }
1329
1330    void printGraph()
1331    {
1332        graph.print();
1333    }
1334
1335    void printDriverList()
1336    {
1337        drivers.printList();
1338    }
1339
1340    void printRideList()
1341    {
1342        rideHistory.printList();
1343    }
1344
1345    void printPendingList()
1346    {
1347        pendingSrc.printList();
1348        pendingDest.printList();
1349    }
1350
1351    void printDriverDetails()
1352    {
1353        drivers.printDetails();
1354    }
1355
1356    void printRideDetails()
1357    {
1358        rideHistory.printDetails();
1359    }
1360
1361    void printPendingDetails()
1362    {
1363        pendingSrc.printDetails();
1364        pendingDest.printDetails();
1365    }
1366
1367    void printDriverStatus()
1368    {
1369        drivers.printStatus();
1370    }
1371
1372    void printGraph()
1373    {
1374        graph.print();
1375    }
1376
1377    void printDriverList()
1378    {
1379        drivers.printList();
1380    }
1381
1382    void printRideList()
1383    {
1384        rideHistory.printList();
1385    }
1386
1387    void printPendingList()
1388    {
1389        pendingSrc.printList();
1390        pendingDest.printList();
1391    }
1392
1393    void printDriverDetails()
1394    {
1395        drivers.printDetails();
1396    }
1397
1398    void printRideDetails()
1399    {
1400        rideHistory.printDetails();
1401    }
1402
1403    void printPendingDetails()
1404    {
1405        pendingSrc.printDetails();
1406        pendingDest.printDetails();
1407    }
1408
1409    void printDriverStatus()
1410    {
1411        drivers.printStatus();
1412    }
1413
1414    void printGraph()
1415    {
1416        graph.print();
1417    }
1418
1419    void printDriverList()
1420    {
1421        drivers.printList();
1422    }
1423
1424    void printRideList()
1425    {
1426        rideHistory.printList();
1427    }
1428
1429    void printPendingList()
1430    {
1431        pendingSrc.printList();
1432        pendingDest.printList();
1433    }
1434
1435    void printDriverDetails()
1436    {
1437        drivers.printDetails();
1438    }
1439
1440    void printRideDetails()
1441    {
1442        rideHistory.printDetails();
1443    }
1444
1445    void printPendingDetails()
1446    {
1447        pendingSrc.printDetails();
1448        pendingDest.printDetails();
1449    }
1450
1451    void printDriverStatus()
1452    {
1453        drivers.printStatus();
1454    }
1455
1456    void printGraph()
1457    {
1458        graph.print();
1459    }
1460
1461    void printDriverList()
1462    {
1463        drivers.printList();
1464    }
1465
1466    void printRideList()
1467    {
1468        rideHistory.printList();
1469    }
1470
1471    void printPendingList()
1472    {
1473        pendingSrc.printList();
1474        pendingDest.printList();
1475    }
1476
1477    void printDriverDetails()
1478    {
1479        drivers.printDetails();
1480    }
1481
1482    void printRideDetails()
1483    {
1484        rideHistory.printDetails();
1485    }
1486
1487    void printPendingDetails()
1488    {
1489        pendingSrc.printDetails();
1490        pendingDest.printDetails();
1491    }
1492
1493    void printDriverStatus()
1494    {
1495        drivers.printStatus();
1496    }
1497
1498    void printGraph()
1499    {
1500        graph.print();
1501    }
1502
1503    void printDriverList()
1504    {
1505        drivers.printList();
1506    }
1507
1508    void printRideList()
1509    {
1510        rideHistory.printList();
1511    }
1512
1513    void printPendingList()
1514    {
1515        pendingSrc.printList();
1516        pendingDest.printList();
1517    }
1518
1519    void printDriverDetails()
1520    {
1521        drivers.printDetails();
1522    }
1523
1524    void printRideDetails()
1525    {
1526        rideHistory.printDetails();
1527    }
1528
1529    void printPendingDetails()
1530    {
1531        pendingSrc.printDetails();
1532        pendingDest.printDetails();
1533    }
1534
1535    void printDriverStatus()
1536    {
1537        drivers.printStatus();
1538    }
1539
1540    void printGraph()
1541    {
1542        graph.print();
1543    }
1544
1545    void printDriverList()
1546    {
1547        drivers.printList();
1548    }
1549
1550    void printRideList()
1551    {
1552        rideHistory.printList();
1553    }
1554
1555    void printPendingList()
1556    {
1557        pendingSrc.printList();
1558        pendingDest.printList();
1559    }
1560
1561    void printDriverDetails()
1562    {
1563        drivers.printDetails();
1564    }
1565
1566    void printRideDetails()
1567    {
1568        rideHistory.printDetails();
1569    }
1570
1571    void printPendingDetails()
1572    {
1573        pendingSrc.printDetails();
1574        pendingDest.printDetails();
1575    }
1576
1577    void printDriverStatus()
1578    {
1579        drivers.printStatus();
1580    }
1581
1582    void printGraph()
1583    {
1584        graph.print();
1585    }
1586
1587    void printDriverList()
1588    {
1589        drivers.printList();
1590    }
1591
1592    void printRideList()
1593    {
1594        rideHistory.printList();
1595    }
1596
1597    void printPendingList()
1598    {
1599        pendingSrc.printList();
1600        pendingDest.printList();
1601    }
1602
1603    void printDriverDetails()
1604    {
1605        drivers.printDetails();
1606    }
1607
1608    void printRideDetails()
1609    {
1610        rideHistory.printDetails();
1611    }
1612
1613    void printPendingDetails()
1614    {
1615        pendingSrc.printDetails();
1616        pendingDest.printDetails();
1617    }
1618
1619    void printDriverStatus()
1620    {
1621        drivers.printStatus();
1622    }
1623
1624    void printGraph()
1625    {
1626        graph.print();
1627    }
1628
1629    void printDriverList()
1630    {
1631        drivers.printList();
1632    }
1633
1634    void printRideList()
1635    {
1636        rideHistory.printList();
1637    }
1638
1639    void printPendingList()
1640    {
1641        pendingSrc.printList();
1642        pendingDest.printList();
1643    }
1644
1645    void printDriverDetails()
1646    {
1647        drivers.printDetails();
1648    }
1649
1650    void printRideDetails()
1651    {
1652        rideHistory.printDetails();
1653    }
1654
1655    void printPendingDetails()
1656    {
1657        pendingSrc.printDetails();
1658        pendingDest.printDetails();
1659    }
1660
1661    void printDriverStatus()
1662    {
1663        drivers.printStatus();
1664    }
1665
1666    void printGraph()
1667    {
1668        graph.print();
1669    }
1670
1671    void printDriverList()
1672    {
1673        drivers.printList();
1674    }
1675
1676    void printRideList()
1677    {
1678        rideHistory.printList();
1679    }
1680
1681    void printPendingList()
1682    {
1683        pendingSrc.printList();
1684        pendingDest.printList();
1685    }
1686
1687    void printDriverDetails()
1688    {
1689        drivers.printDetails();
1690    }
1691
1692    void printRideDetails()
1693    {
1694        rideHistory.printDetails();
1695    }
1696
1697    void printPendingDetails()
1698    {
1699        pendingSrc.printDetails();
1700        pendingDest.printDetails();
1701    }
1702
1703    void printDriverStatus()
1704    {
1705        drivers.printStatus();
1706    }
1707
1708    void printGraph()
1709    {
1710        graph.print();
1711    }
1712
1713    void printDriverList()
1714    {
1715        drivers.printList();
1716    }
1717
1718    void printRideList()
1719    {
1720        rideHistory.printList();
1721    }
1722
1723    void printPendingList()
1724    {
1725        pendingSrc.printList();
1726        pendingDest.printList();
1727    }
1728
1729    void printDriverDetails()
1730    {
1731        drivers.printDetails();
1732    }
1733
1734    void printRideDetails()
1735    {
1736        rideHistory.printDetails();
1737    }
1738
1739    void printPendingDetails()
1740    {
1741        pendingSrc.printDetails();
1742        pendingDest.printDetails();
1743    }
1744
1745    void printDriverStatus()
1746    {
1747        drivers.printStatus();
1748    }
1749
1750    void printGraph()
1751    {
1752        graph.print();
1753    }
1754
1755    void printDriverList()
1756    {
1757        drivers.printList();
1758    }
1759
1760    void printRideList()
1761    {
1762        rideHistory.printList();
1763    }
1764
1765    void printPendingList()
1766    {
1767        pendingSrc.printList();
1768        pendingDest.printList();
1769    }
1770
1771    void printDriverDetails()
1772    {
1773        drivers.printDetails();
1774    }
1775
1776    void printRideDetails()
1777    {
1778        rideHistory.printDetails();
1779    }
1780
1781    void printPendingDetails()
1782    {
1783        pendingSrc.printDetails();
1784        pendingDest.printDetails();
1785    }
1786
1787    void printDriverStatus()
1788    {
1789        drivers.printStatus();
1790    }
1791
1792    void printGraph()
1793    {
1794        graph.print();
1795    }
1796
1797    void printDriverList()
1798    {
1799        drivers.printList();
1800    }
1801
1802    void printRideList()
1803    {
1804        rideHistory.printList();
1805    }
1806
1807    void printPendingList()
1808    {
1809        pendingSrc.printList();
1810        pendingDest.printList();
1811    }
1812
1813    void printDriverDetails()
1814    {
1815        drivers.printDetails();
1816    }
1817
1818    void printRideDetails()
1819    {
1820        rideHistory.printDetails();
1821    }
1822
1823    void printPendingDetails()
1824    {
1825        pendingSrc.printDetails();
1826        pendingDest.printDetails();
1827    }
1828
1829    void printDriverStatus()
1830    {
1831        drivers.printStatus();
1832    }
1833
1834    void printGraph()
1835    {
1836        graph.print();
1837    }
1838
1839    void printDriverList()
1840    {
1841        drivers.printList();
1842    }
1843
1844    void printRideList()
1845    {
1846        rideHistory.printList();
1847    }
1848
1849    void printPendingList()
1850    {
1851        pendingSrc.printList();
1852        pendingDest.printList();
1853    }
1854
1855    void printDriverDetails()
1856    {
1857        drivers.printDetails();
1858    }
1859
1860    void printRideDetails()
1861    {
1862        rideHistory.printDetails();
1863    }
1864
1865    void printPendingDetails()
1866    {
1867        pendingSrc.printDetails();
1868        pendingDest.printDetails();
1869    }
1870
1871    void printDriverStatus()
1872    {
1873        drivers.printStatus();
1874    }
1875
1876    void printGraph()
1877    {
1878        graph.print();
1879    }
1880
1881    void printDriverList()
1882    {
1883        drivers.printList();
1884    }
1885
1886    void printRideList()
1887    {
1888        rideHistory.printList();
1889    }
1890
1891    void printPendingList()
1892    {
1893        pendingSrc.printList();
1894        pendingDest.printList();
1895    }
1896
1897    void printDriverDetails()
1898    {
1899        drivers.printDetails();
1900    }
1901
1902    void printRideDetails()
1903    {
1904        rideHistory.printDetails();
1905    }
1906
1907    void printPendingDetails()

```

```

[+] 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 void cancelRide(int rideID)
556 {
557     RideRecord* ride = rideHistory.findByID(rideID);
558     if (!ride)
559     {
560         cout << "Ride ID " << rideID << " not found." << endl;
561         return;
562     }
563     if (ride->status != "Ongoing")
564     {
565         cout << "Cannot cancel ride. Status: " << ride->status << endl;
566         return;
567     }
568
569     ride->status = "Cancelled";
570
571     Driver* d = drivers.find(ride->driverID);
572     if (d)
573         d->available = true;
574
575     cout << "Ride " << rideID << " has been cancelled." << endl;
576 }
577
578 void processPendingRequests()
579 {
580     int qsize = pendingSrc.size();
581     for (int i = 0; i < qsize; i++)
582     {
583         int src = pendingSrc.dequeue();
584         int dest = pendingDest.dequeue();
585         cout << "Trying pending request " << src << " -> " << dest << "..." << endl;
586         requestRide(src, dest);
587     }
588 }
589
590 void showRideHistory()
591 {
592     rideHistory.printAll();
593 }
594 }
595
596 // ----- Main -----
597
598 // ----- Main -----
599
600 int main()
601 {
602
603
604     CabSystem sys;
605     int nodes;
606     cout << "Initialize city map (number of nodes): ";
607     cin >> nodes;
608     sys.initGraph(nodes);
609
610     int m;
611     cout << "Create roads (enter m edges): ";
612     cin >> m;
613
614     for (int i = 0; i < m; i++)
615     {
616         int u, v;
617         double w;
618         cout << "Edge " << i + 1 << " u v w: ";
619         cin >> u >> v >> w;
620         sys.addRoad(u, v, w);
621     }
622
623     sys.setRate(10.0);
624
625     while (true)
626     {
627         cout << "\n--- Cab Management Menu ---" << endl;
628         cout << "1. Add Driver" << endl;
629         cout << "2. Remove Driver" << endl;
630         cout << "3. Update Driver Location" << endl;
631         cout << "4. Show Drivers" << endl;
632         cout << "5. Show Map" << endl;
633         cout << "6. Request Ride" << endl;
634         cout << "7. Process Pending Requests" << endl;
635         cout << "8. Show Ride History" << endl;
636         cout << "9. Cancel Ride" << endl;
637         cout << "10. Complete Ride" << endl;
638         cout << "11. Exit" << endl;
639         cout << "Choose: ";
640
641         int ch;
642         cin >> ch;
643
644         if (ch == 1)
645         {
646             int id, node;
647             string name;
648             cout << "Driver ID: ";
649             cin >> id;
650             cout << "Name: ";
651             cin.ignore();
652             getline(cin, name);
653             cout << "Node: ";
654             cin >> node;
655             sys.addDriver(id, name, node);
656         }
657     }
658 }

```

```

[+] main.cpp x
598
599
600 // ----- Main -----
601 int main()
602 {
603
604     CabSystem sys;
605     int nodes;
606     cout << "Initialize city map (number of nodes): ";
607     cin >> nodes;
608     sys.initGraph(nodes);
609
610     int m;
611     cout << "Create roads (enter m edges): ";
612     cin >> m;
613
614     for (int i = 0; i < m; i++)
615     {
616         int u, v;
617         double w;
618         cout << "Edge " << i + 1 << " u v w: ";
619         cin >> u >> v >> w;
620         sys.addRoad(u, v, w);
621     }
622
623     sys.setRate(10.0);
624
625     while (true)
626     {
627         cout << "\n--- Cab Management Menu ---" << endl;
628         cout << "1. Add Driver" << endl;
629         cout << "2. Remove Driver" << endl;
630         cout << "3. Update Driver Location" << endl;
631         cout << "4. Show Drivers" << endl;
632         cout << "5. Show Map" << endl;
633         cout << "6. Request Ride" << endl;
634         cout << "7. Process Pending Requests" << endl;
635         cout << "8. Show Ride History" << endl;
636         cout << "9. Cancel Ride" << endl;
637         cout << "10. Complete Ride" << endl;
638         cout << "11. Exit" << endl;
639         cout << "Choose: ";
640
641         int ch;
642         cin >> ch;
643
644         if (ch == 1)
645         {
646             int id, node;
647             string name;
648             cout << "Driver ID: ";
649             cin >> id;
650             cout << "Name: ";
651             cin.ignore();
652             getline(cin, name);
653             cout << "Node: ";
654             cin >> node;
655             sys.addDriver(id, name, node);
656         }
657     }
658 }

```

```
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\hp\Desktop\3 > + ^

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\b3 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\b3 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 + ^ 
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 + ^ 
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.