3829. Design Ride Sharing System

Solved

Medium Topics Hint

A ride sharing system manages ride requests from riders and availability from drivers. Riders request rides, and drivers become available over time. The system should match riders and drivers in the order they arrive.

Implement the `RideSharingSystem` class:

- `RideSharingSystem()` Initializes the system.
- `void addRider(int riderId)` Adds a new rider with the given `riderId`.
- `void addDriver(int driverId)` Adds a new driver with the given `driverId`.
- `int[] matchDriverWithRider()` Matches the **earliest** available driver with the **earliest** waiting rider and removes both of them from the system. Returns an integer array of size 2 where `result = [driverId, riderId]` if a match is made. If no match is available, returns `[-1, -1]`.
- `void cancelRider(int riderId)` Cancels the ride request of the rider with the given `riderId` **if the rider exists** and has **not** yet been matched.

Example 1:

> **Input:**
> ["RideSharingSystem", "addRider", "addDriver", "addRider", "matchDriverWithRider", "addDriver", "cancelRider", "matchDriverWithRider", "matchDriverWithRider"]
> [[], [3], [2], [1], [], [5], [3], [], []]
>
> **Output:**
> [null, null, null, null, [2, 3], null, null, [5, 1], [-1, -1]]
>
> **Explanation**
>
> RideSharingSystem rideSharingSystem = new RideSharingSystem(); // Initializes the system
> rideSharingSystem.addRider(3); // rider 3 joins the queue
> rideSharingSystem.addDriver(2); // driver 2 joins the queue
> rideSharingSystem.addRider(1); // rider 1 joins the queue
> rideSharingSystem.matchDriverWithRider(); // returns [2, 3]
> rideSharingSystem.addDriver(5); // driver 5 becomes available
> rideSharingSystem.cancelRider(3); // rider 3 is already matched, cancel has no effect
> rideSharingSystem.matchDriverWithRider(); // returns [5, 1]
> rideSharingSystem.matchDriverWithRider(); // returns [-1, -1]

Example 2:

> **Input:**
> ["RideSharingSystem", "addRider", "addDriver", "addDriver", "matchDriverWithRider", "addRider", "cancelRider", "matchDriverWithRider"]
> [[], [8], [8], [6], [], [2], [2], []]
>
> **Output:**
> [null, null, null, null, [8, 8], null, null, [-1, -1]]
>
> **Explanation**
>
> RideSharingSystem rideSharingSystem = new RideSharingSystem(); // Initializes the system
> rideSharingSystem.addRider(8); // rider 8 joins the queue
> rideSharingSystem.addDriver(8); // driver 8 joins the queue
> rideSharingSystem.addDriver(6); // driver 6 joins the queue
> rideSharingSystem.matchDriverWithRider(); // returns [8, 8]
> rideSharingSystem.addRider(2); // rider 2 joins the queue
> rideSharingSystem.cancelRider(2); // rider 2 cancels
> rideSharingSystem.matchDriverWithRider(); // returns [-1, -1]

Constraints:

- `1 <= riderId, driverId <= 1000`
- Each `riderId` is **unique** among riders and is added at most **once**.
- Each `driverId` is **unique** among drivers and is added at most **once**.
- At most 1000 calls will be made in **total** to `addRider`, `addDriver`, `matchDriverWithRider`, and `cancelRider`.

Seen this question in a real interview before? 1/5

Yes No

Accepted 22,854/36.3K | Acceptance Rate 63.0%

Topics

Hint 1

Hint 2

Hint 3

Discussion (20)