<https://leetcode.com/problems/design-underground-system/>

An underground railway system is keeping track of customer travel times between different stations. They are using this data to calculate the average time it takes to travel from one station to another.

Implement the UndergroundSystem class:

void checkIn(int id, string stationName, int t)

A customer with a card ID equal to id, checks in at the station stationName at time t.

A customer can only be checked into one place at a time.

void checkOut(int id, string stationName, int t)

A customer with a card ID equal to id, checks out from the station stationName at time t.

double getAverageTime(string startStation, string endStation)

Returns the average time it takes to travel from startStation to endStation.

The average time is computed from all the previous traveling times from startStation to endStation that happened **directly**, meaning a check in at startStation followed by a check out from endStation.

The time it takes to travel from startStation to endStation **may be different** from the time it takes to travel from endStation to startStation.

There will be at least one customer that has traveled from startStation to endStation before getAverageTime is called.

You may assume all calls to the checkIn and checkOut methods are consistent. If a customer checks in at time t1 then checks out at time t2, then t1 < t2. All events happen in chronological order.

**Example 1:**

Input

["UndergroundSystem","checkIn","checkIn","checkIn","checkOut","checkOut","checkOut","getAverageTime","getAverageTime","checkIn","getAverageTime","checkOut","getAverageTime"]

[[],[45,"Leyton",3],[32,"Paradise",8],[27,"Leyton",10],[45,"Waterloo",15],[27,"Waterloo",20],[32,"Cambridge",22],["Paradise","Cambridge"],["Leyton","Waterloo"],[10,"Leyton",24],["Leyton","Waterloo"],[10,"Waterloo",38],["Leyton","Waterloo"]]

Output

[null,null,null,null,null,null,null,14.00000,11.00000,null,11.00000,null,12.00000]

Explanation

UndergroundSystem undergroundSystem = new UndergroundSystem();

undergroundSystem.checkIn(45, "Leyton", 3);

undergroundSystem.checkIn(32, "Paradise", 8);

undergroundSystem.checkIn(27, "Leyton", 10);

undergroundSystem.checkOut(45, "Waterloo", 15); // Customer 45 "Leyton" -> "Waterloo" in 15-3 = 12

undergroundSystem.checkOut(27, "Waterloo", 20); // Customer 27 "Leyton" -> "Waterloo" in 20-10 = 10

undergroundSystem.checkOut(32, "Cambridge", 22); // Customer 32 "Paradise" -> "Cambridge" in 22-8 = 14

undergroundSystem.getAverageTime("Paradise", "Cambridge"); // return 14.00000. One trip "Paradise" -> "Cambridge", (14) / 1 = 14

undergroundSystem.getAverageTime("Leyton", "Waterloo"); // return 11.00000. Two trips "Leyton" -> "Waterloo", (10 + 12) / 2 = 11

undergroundSystem.checkIn(10, "Leyton", 24);

undergroundSystem.getAverageTime("Leyton", "Waterloo"); // return 11.00000

undergroundSystem.checkOut(10, "Waterloo", 38); // Customer 10 "Leyton" -> "Waterloo" in 38-24 = 14

undergroundSystem.getAverageTime("Leyton", "Waterloo"); // return 12.00000. Three trips "Leyton" -> "Waterloo", (10 + 12 + 14) / 3 = 12

**Example 2:**

Input

["UndergroundSystem","checkIn","checkOut","getAverageTime","checkIn","checkOut","getAverageTime","checkIn","checkOut","getAverageTime"]

[[],[10,"Leyton",3],[10,"Paradise",8],["Leyton","Paradise"],[5,"Leyton",10],[5,"Paradise",16],["Leyton","Paradise"],[2,"Leyton",21],[2,"Paradise",30],["Leyton","Paradise"]]

Output

[null,null,null,5.00000,null,null,5.50000,null,null,6.66667]

Explanation

UndergroundSystem undergroundSystem = new UndergroundSystem();

undergroundSystem.checkIn(10, "Leyton", 3);

undergroundSystem.checkOut(10, "Paradise", 8); // Customer 10 "Leyton" -> "Paradise" in 8-3 = 5

undergroundSystem.getAverageTime("Leyton", "Paradise"); // return 5.00000, (5) / 1 = 5

undergroundSystem.checkIn(5, "Leyton", 10);

undergroundSystem.checkOut(5, "Paradise", 16); // Customer 5 "Leyton" -> "Paradise" in 16-10 = 6

undergroundSystem.getAverageTime("Leyton", "Paradise"); // return 5.50000, (5 + 6) / 2 = 5.5

undergroundSystem.checkIn(2, "Leyton", 21);

undergroundSystem.checkOut(2, "Paradise", 30); // Customer 2 "Leyton" -> "Paradise" in 30-21 = 9

undergroundSystem.getAverageTime("Leyton", "Paradise"); // return 6.66667, (5 + 6 + 9) / 3 = 6.66667

**Constraints:**

* 1 <= id, t <= 106
* 1 <= stationName.length, startStation.length, endStation.length <= 10
* All strings consist of uppercase and lowercase English letters and digits.
* There will be at most 2 \* 104 calls **in total** to checkIn, checkOut, and getAverageTime.
* Answers within 10-5 of the actual value will be accepted.

**Attempt 1: 2023-02-07**

**Solution 1:  Hash Table (30 min)**

class UndergroundSystem {

Map<Integer, Passenger> passengerArrivals;

Map<String, Route> routeAverage;

public UndergroundSystem() {

passengerArrivals = new HashMap<Integer, Passenger>();

routeAverage = new HashMap<String, Route>();

}

public void checkIn(int id, String stationName, int t) {

passengerArrivals.put(id, new Passenger(id, stationName, t));

}

public void checkOut(int id, String stationName, int t) {

Passenger p = passengerArrivals.get(id);

int duration = t - p.t;

String key = p.stationName + "\_" + stationName;

Route r = routeAverage.getOrDefault(key, new Route(0, 0));

routeAverage.put(key, new Route(r.totalTime + duration, r.trips + 1));

}

public double getAverageTime(String startStation, String endStation) {

String key = startStation + "\_" + endStation;

Route r = routeAverage.get(key);

return (double)r.totalTime / r.trips;

}

}

class Passenger {

int id;

String stationName;

int t;

public Passenger(int id, String stationName, int t) {

this.id = id;

this.stationName = stationName;

this.t = t;

}

}

class Route {

int totalTime;

int trips;

public Route(int totalTime, int trips) {

this.totalTime = totalTime;

this.trips = trips;

}

}

/\*\*

\* Your UndergroundSystem object will be instantiated and called as such:

\* UndergroundSystem obj = new UndergroundSystem();

\* obj.checkIn(id,stationName,t);

\* obj.checkOut(id,stationName,t);

\* double param\_3 = obj.getAverageTime(startStation,endStation);

\*/

Time Complexity:O(1) as we are using HashMap, anything we are doing inserting, removing taking O(1)

Space Complexity:O(N + M) where N is no. of passengersArrivals we have & M is no. of routeAverage we have

**Refer to**

<https://leetcode.com/problems/design-underground-system/solutions/1976674/explanation/>

We, need to Design an Underground System,**This system will do 3 thing's for us :-**

* Check customer into a station
* Check customer out of a station
* Average travel b/w station

**checkIn(int id, String stationName, int t)**

here **id** is a unique identification to customer & **stationName** must check customer name into station at time **t**

**checkOut(int id, String stationName, int t)**

we have to reach to our final station

**getAverageTime(String startStation, String endStation)**

calculate avergae time b/w these 2 stations

Now we have to keep Track of our customer checkedIn & checkedOut unique Id's using our map

So, we just create a map called passengersArrivals

Map<id, (id, station, t)> passengersArrivals;

Now we have to keep Track of all previous station & once again we gonna use map for that

Take 2 station's :- A , B

& we have a comma b/w them. That'll be the unique key b/w travel of those 2 stations.

So, in our map we'll have:-

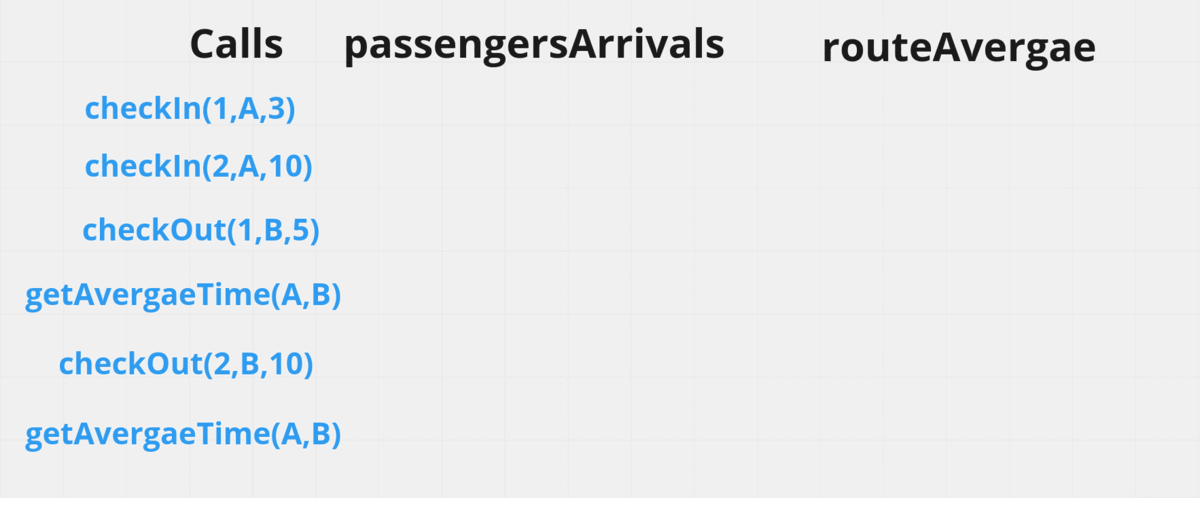
map<names, (total, count)> routeAverage;

here total is :- sum of all the travel that has happen b/w these 2 stations

here count is :- amount of customer that travel b/w these 2 stations

Easily compute the avergae in any time by simple doing "total / count"

**Let's Understood it visually,**



class UndergroundSystem {

class passenger{

public int id;

public String stationName;

public int time;

public passenger(int id, String stationName, int time){

this.id = id;

this.stationName = stationName;

this.time = time;

}

}

class route{

public double total = 0;

public int count = 0;

public void update(int difference){

count++;

total += difference;

}

public double getAvg(){

return total / count;

}

}

public Map<Integer, passenger> passengersArrivals;

public Map<String, route> routeAverage;

public UndergroundSystem() {

passengersArrivals = new HashMap<>();

routeAverage = new HashMap<>();

}

public void checkIn(int id, String stationName, int t) {

passengersArrivals.put(id, new passenger(id, stationName, t));

}

public final String DELIMETER = ",";

public void checkOut(int id, String stationName, int t) {

passenger arrivePassenger = passengersArrivals.get(id);

passengersArrivals.remove(id);

int difference = t - arrivePassenger.time;

String key = arrivePassenger.stationName + DELIMETER + stationName;

route average = routeAverage.containsKey(key) ? routeAverage.get(key) : new route();

average.update(difference);

routeAverage.put(key, average);

}

public double getAverageTime(String startStation, String endStation) {

String key = startStation + DELIMETER + endStation;

return routeAverage.get(key).getAvg();

}

}

ANALYSIS :-

**Time Complexity :-** BigO(1) as we are using HashMap, anything we are doing inserting, removing taking O(1)

**Space Complexity :-** BigO(N + M) where N is no. of passengersArrivals we have & M is no. of routeAverage we have