Description | Editorial | Solutions (1.6K) | Submissions

1396. Design Underground System

Hint

Medium | 2.9K | 135

Companies

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 t_1 then checks out at time t_2, then $t_1 < t_2$. All events happen in chronological order.

Java | Auto

```java
class UndergroundSystem {

    Map<Integer, ArrivalInfo> arrivals;
    Map<String, double[]> total;

    public UndergroundSystem() {
        arrivals = new HashMap<>();
        total = new HashMap<>();

    }

    public void checkIn(int id, String stationName, int t) {
        arrivals.put(id, new ArrivalInfo(id, stationName, t));
    }

    public void checkOut(int id, String stationName, int t) {
        ArrivalInfo arrival = arrivals.get(id);
        String key = arrival.stationName +"_" + stationName;
        double[]pair = total.getOrDefault(key, new double[2]);
        int time = t-arrival.time;
```

Testcase | Result

Accepted Runtime: 24 ms

Case 1 | Case 2

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"]]
```

Console | Run | Submit

Description | Editorial | Solutions (7.9K) | Submissions

11. Container With Most Water

Hint

Medium | 24.3K | 1.3K

Companies

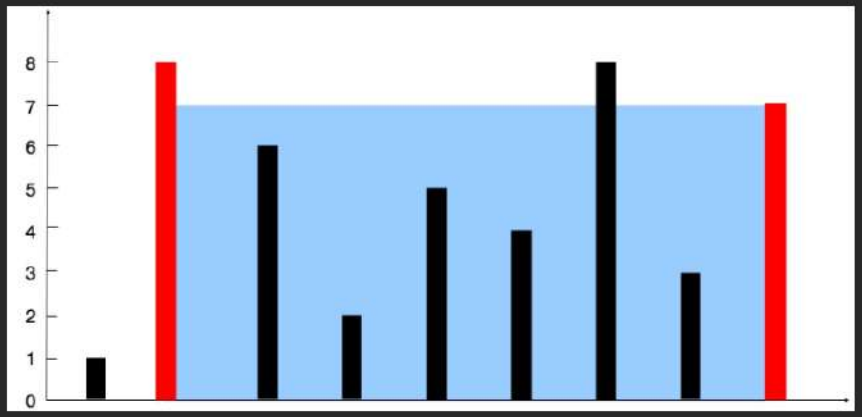
You are given an integer array `height` of length `n`. There are `n` vertical lines drawn such that the two endpoints of the `i^th` line are `(i, 0)` and `(i, height[i])`.

Find two lines that together with the x-axis form a container, such that the container contains the most water.

Return *the maximum amount of water a container can store.*

Notice that you may not slant the container.

Example 1:



Input: height = [1,8,6,2,5,4,8,3,7]
Output: 49
Explanation: The above vertical lines are represented by array [1,8,6,2,5,4,8,3,7]. In this case, the max area of water (blue section) the container can contain is 49.

Java | Auto

```java
class Solution {
    public int maxArea(int[] height) {
        int water = 0, left =0, right = height.length-1;

        while(left  < right){
            water = Math.max(water, Math.min(height[left], height[right]) * (right-left));
            if(height[left] > height[right]) right--;
            else left++;
        }
        return water;
    }
}
```

Testcase | Result

Accepted Runtime: 0 ms

Case 1 | Case 2

Input

height =
[1,8,6,2,5,4,8,3,7]

Output

49

Expected

49

Contribute a testcase

Console | Run | Submit