Design Hit Counter

Design a hit counter which counts the number of hits received in the past 5 minutes.

Each function accepts a timestamp parameter (in seconds granularity) and you may assume that calls are being made to the system in chronological order (ie, the timestamp is monotonically increasing). You may assume that the earliest timestamp starts at 1.

It is possible that several hits arrive roughly at the same time.

Example:

```
HitCounter counter = new HitCounter();

// hit at timestamp 1.
counter.hit(1);

// hit at timestamp 2.
counter.hit(2);

// hit at timestamp 3.
counter.hit(3);

// get hits at timestamp 4, should return 3.
counter.getHits(4);

// hit at timestamp 300.
counter.hit(300);

// get hits at timestamp 300, should return 4.
counter.getHits(300);

// get hits at timestamp 301, should return 3.
counter.getHits(301);
```

Follow up:
What if the number of hits per second could be very large? Does your design scale?

Credits:
Special thanks to @elmirap for adding this problem and creating all test cases.

Solution 1

O(s) s is total seconds in given time interval, in this case 300. basic ideal is using buckets. 1 bucket for every second because we only need to keep the recent hits info for 300 seconds. hit[] array is wrapped around by mod operation. Each hit bucket is associated with times[] bucket which record current time. If it is not current time, it means it is 300s or 600s... ago and need to reset to 1.

```
public class HitCounter {
    private int[] times;
    private int[] hits;
    /** Initialize your data structure here. */
    public HitCounter() {
        times = new int[300];
        hits = new int[300];
    }

    /** Record a hit.
        @param timestamp - The current timestamp (in seconds granularity). */
    public void hit(int timestamp) {
        int index = timestamp % 300;
        if (times[index] != timestamp) {
            times[index] = timestamp;
            hits[index] = 1;
        } else {
            hits[index]++;
        }
    }

    /** Return the number of hits in the past 5 minutes.
        @param timestamp - The current timestamp (in seconds granularity). */
    public int getHits(int timestamp) {
        int total = 0;
        for (int i = 0; i < 300; i++) {
            if (timestamp - times[i] < 300) {
                total += hits[i];
            }
        }
        return total;
    }
}
```

written by xuyirui original link here

Solution 2

In this problem, I use a queue to record the information of all the hits. Each time we call the function getHits( ), we have to delete the elements which hits beyond 5 mins (300). The result would be the length of the queue :)

```
public class HitCounter {
        Queue<Integer> q = null;
        /** Initialize your data structure here. */
        public HitCounter() {
            q = new LinkedList<Integer>();
        }

        /** Record a hit.
            @param timestamp - The current timestamp (in seconds granularity). */
        public void hit(int timestamp) {
            q.offer(timestamp);
        }

        /** Return the number of hits in the past 5 minutes.
            @param timestamp - The current timestamp (in seconds granularity). */
        public int getHits(int timestamp) {
            while(!q.isEmpty() && timestamp - q.peek() >= 300) {
                q.poll();
            }
            return q.size();
        }
    }
```

written by cyqz1993 original link here

Solution 3

This solution is based on the idea in this post: https://nuttynanaus.wordpress.com/2014/03/09/software-engineer-interview-questions/
There are two solutions, the first one we choose 1s as granularity, the other is full accuracy(see the post). We call move() before hit() and getHits(). move() will take time at most O(N), where N is the length of the array.

```
public class HitCounter {
    int N;
    int[] count;
    int lastPosition;
    int lastTime;
    int sum;

    /** Initialize your data structure here. */
    public HitCounter() {
        N = 300;
        count = new int[N];
        lastPosition = 0;
        lastTime = 0;
        sum = 0;
    }

    /** Record a hit.
        @param timestamp - The current timestamp (in seconds granularity). */
    public void hit(int timestamp) {
        move(timestamp);
        count[lastPosition]++;
        sum++;
    }

    /** Return the number of hits in the past 5 minutes.
        @param timestamp - The current timestamp (in seconds granularity). */
    public int getHits(int timestamp) {
        move(timestamp);
        return sum;
    }

    void move(int timestamp){
        int gap = Math.min(timestamp-lastTime, N);
        for(int i=0; i<gap;i++){
            lastPosition = (lastPosition+1)%N;
            sum -= count[lastPosition];
            count[lastPosition] = 0;
        }
        lastTime = timestamp;
    }
}
```

written by OrlandoChen0308 original link here