Python solution with detailed explanation

gabbu 1589 Last Edit: September 24, 2018 4:09 AM 3.2K VIEWS

52

Solutions

Design Hit Counter https://leetcode.com/problems/design-hit-counter/

Deque Based Solution

- Use deque as container. Deque stores the value of the timestamp.
- hit: append the timestamp to the deque. O(1)
- get_hit: O(N). During get_hit, pop out all elements from the left of the queue which do not serve any purpose and are stale.Return the length of the deque.
- How do we find the elements which are stale? Condition we want to use:" timestamp-t >= 300"
- Example: timestamp = 301 and t = 1. Valid Range: [1 to 300], [2 to 301], [3,303]. So 301-1 >= 300. Hence 1 should be popped since it doesnt belong to range 2 to 301.
- Poor scalability of the solution since we store all timestamps.

```
from collections import deque
class HitCounter(object):
    def __init__(self):
        """
        Initialize your data structure here.
        """
        self.counter = deque()

    def hit(self, timestamp):
        """
        Record a hit.
        @param timestamp - The current timestamp (in seconds granularity).
        :type timestamp: int
        :rtype: void
        """
        self.counter.append(timestamp)

    def getHits(self, timestamp):
        """
        Return the number of hits in the past 5 minutes.
        @param timestamp - The current timestamp (in seconds granularity).
        :type timestamp: int
        :rtype: int
        """
        while self.counter and timestamp -self.counter[0] >= 300:
            self.counter.popleft()
        return len(self.counter)
```

Dictionary based Solution

- The above solution doesnt scale since we add an element to the deque for every hit.
- Use a dictionary instead and prune the dictionary.
- Atleast in the dictionary solution, same timestamps will not be repeated. But the dictionary can grow unbounded.

```
from collections import defaultdict
class HitCounter(object):
    def __init__(self):
        """
        Initialize your data structure here.
        """
        self.counter = defaultdict(int)

    def get(self, timestamp):
        return self.prune(timestamp)

    def hit(self, timestamp):
        """
        Record a hit.
        @param timestamp - The current timestamp (in seconds granularity).
        :type timestamp: int
        :rtype: void
        """
        self.counter[timestamp] = self.counter[timestamp] + 1

    def getHits(self, timestamp):
        """
        Return the number of hits in the past 5 minutes.
        @param timestamp - The current timestamp (in seconds granularity).
        :type timestamp: int
        :rtype: int
        """
        cnt = 0
        for k in self.counter.keys():
            if timestamp - k >= 300:
                del self.counter[k]
            else:
                cnt = cnt + self.counter[k]
        return cnt
```

Optimized and Scalable Solution

- The third solution creates an array of 300 elements. Every element of the array comprises of {frequency, timestamp}.
- Timestamp 1 maps to index 0. Timestamp 100 maps to index 99.
- Use modulo mathematics to update it. hit: O(1). get_hit: O(300). This solution will scale perfectly!

```
class HitCounter(object):
    def __init__(self):
        """
        Initialize your data structure here.
        """
        self.counter = [[0,i+1] for i in range(300)]
        return

    def hit(self, timestamp):
        """
        Record a hit.
        @param timestamp - The current timestamp (in seconds granularity).
        :type timestamp: int
        :rtype: void
        """
        # ts = 301 means (301-1)%300
        idx = int((timestamp - 1)%300)
        if self.counter[idx][1] == timestamp:
            self.counter[idx][0] += 1
        else:
            self.counter[idx][0] = 1
            self.counter[idx][1] = timestamp

    def getHits(self, timestamp):
        """
        Return the number of hits in the past 5 minutes.
        @param timestamp - The current timestamp (in seconds granularity).
        :type timestamp: int
        :rtype: int
        """
        cnt = 0
        for x in self.counter:
            c,t = x[0],x[1]
            if timestamp - t < 300:
                cnt += c
        return cnt
```

https://discuss.leetcode.com/topic/48752/simple-java-solution-with-explanation/9