Moving Average from Data Stream

Given a stream of integers and a window size, calculate the moving average of all integers in the sliding window.

For example,

```
MovingAverage m = new MovingAverage(3);
m.next(1) = 1
m.next(10) = (1 + 10) / 2
m.next(3) = (1 + 10 + 3) / 3
m.next(5) = (10 + 3 + 5) / 3
```

Solution 1

```
import collections

class MovingAverage(object):

    def __init__(self, size):
        """
        Initialize your data structure here.
        :type size: int
        """
        self.queue = collections.deque(maxlen=size)


    def next(self, val):
        """
        :type val: int
        :rtype: float
        """
        queue = self.queue
        queue.append(val)
        return float(sum(queue))/len(queue)



# Your MovingAverage object will be instantiated and called as such:
# obj = MovingAverage(size)
# param_1 = obj.next(val)
```

written by microleo720 original link here

Solution 2

Essentially, we just need to keep track of the sum of the current window as we go. This prevents an O(n) traversal over the Queue as we add new numbers to get the new average. If we need to evict then we just subtract that number off of our sum and divide by the size.

```
public class MovingAverage {
private double previousSum = 0.0;
private int maxSize;
private Queue<Integer> currentWindow;

public MovingAverage(int size) {
    currentWindow = new LinkedList<Integer>();
    maxSize = size;
}

public double next(int val) {
    if (currentWindow.size() == maxSize)
    {
        previousSum -= currentWindow.remove();
    }

    previousSum += val;
    currentWindow.add(val);
    return previousSum / currentWindow.size();
}}
```

written by bdwalker original link here

Solution 3

```
public class MovingAverage {

    Deque<Integer> dq;
    int size;
    int sum;
    public MovingAverage(int size) {
        dq = new LinkedList<>();
        this.size = size;
        this.sum = 0;
    }

    public double next(int val) {
        if (dq.size() < size) {
            sum += val;
            dq.addLast(val);
            return (double) (sum / dq.size());
        } else {
            int temp = dq.pollFirst();
            sum -= temp;
            dq.addLast(val);
            sum += val;
            return (double) (sum / size);
        }
    }

}
```

written by publicstatic2 original link here