Previous Next

346. Moving Average from Data Stream

Oct. 6, 2019 | 19.5K views

Average Rating: 4.32 (22 votes)

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

Example:

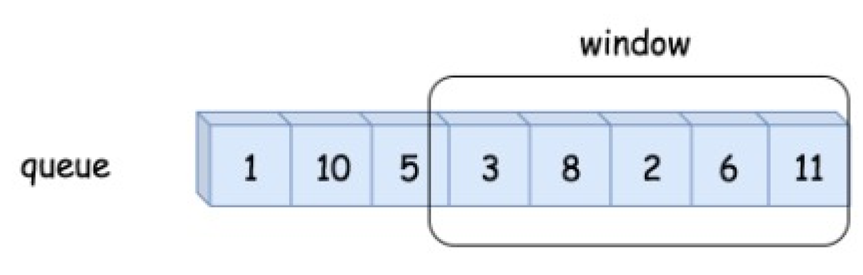
```
MovingAverage m = new MovingAverage(3);
m.next(1) = 1
m.next(10) = (1 + 10) / 2
m.next(3) = (1 + 10 + 3) / 3
m.next(5) = (10 + 3 + 5) / 3
```

Solution

Approach 1: Array or List

Intuition

Following the description of the problem, we could simply keep track of all the incoming values with the data structure of *Array* or *List*. Then from the data structure, later we retrieve the necessary elements to calculate the average.



Algorithm

- First, we initialize a variable `queue` to store the values from the data stream, and the variable `n` for the size of the moving window.
- At each invocation of `next(val)`, we first append the value to the queue. We then retrieve the last `n` values from the queue, in order to calculate the average.

Java | Python | Copy

```python
class MovingAverage:
    def __init__(self, size: int):
        self.size = size
        self.queue = []

    def next(self, val: int) -> float:
        size, queue = self.size, self.queue
        queue.append(val)
        # calculate the sum of the moving window
        window_sum = sum(queue[-size:])

        return window_sum / min(len(queue), size)
```

Complexity

- Time Complexity: $\mathcal{O}(N)$ where N is the size of the moving window, since we need to retrieve N elements from the queue at each invocation of `next(val)` function.
- Space Complexity: $\mathcal{O}(M)$, where M is the length of the queue which would grow at each invocation of the `next(val)` function.

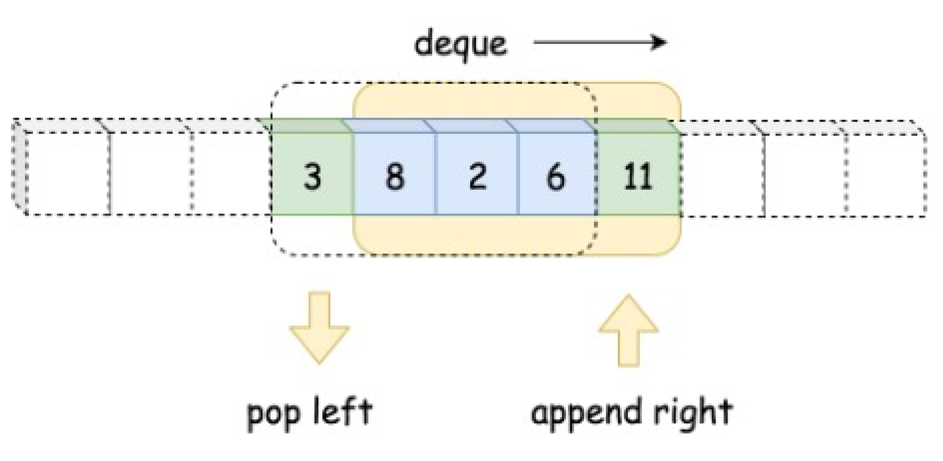
Approach 2: Double-ended Queue

Intuition

We could do better than the first approach in both time and space complexity.

> First of all, one might notice that we do not need to keep all values from the data stream, but rather the last `n` values which falls into the moving window.

By definition of the moving window, at each step, we add a new element to the window, and at the same time we remove the oldest element from the window. Here, we could apply a data structure called *double-ended queue* (*a.k.a* deque) to implement the moving window, which would have the constant time complexity ($\mathcal{O}(1)$) to add or remove an element from both its ends. With the deque, we could reduce the space complexity down to $\mathcal{O}(N)$ where N is the size of the moving window.



> Secondly, to calculate the sum, we do not need to reiterate the elements in the moving window.

We could keep the sum of the previous moving window, then in order to obtain the sum of the new moving window, we simply add the new element and deduce the oldest element. With this measure, we then can reduce the time complexity to constant.

Algorithm

Here is the definition of the *deque* from Python. We have similar implementation of deque in other programming languages such as Java.

> Deques are a generalization of stacks and queues (the name is pronounced `deck` and is short for `double-ended queue`). Deques support thread-safe, memory efficient appends and pops from either side of the deque with approximately the same O(1) performance in either direction.

Follow the intuition, we replace the queue with the *deque* and add a new variable `window_sum` in order to calculate the sum of moving window in constant time.

Java | Python | Copy

```python
from collections import deque
class MovingAverage:
    def __init__(self, size: int):
        self.size = size
        self.queue = deque()
        # number of elements seen so far
        self.window_sum = 0
        self.count = 0

    def next(self, val: int) -> float:
        self.count += 1
        # calculate the new sum by shifting the window
        self.queue.append(val)
        tail = self.queue.popleft() if self.count > self.size else 0

        self.window_sum = self.window_sum - tail + val

        return self.window_sum / min(self.size, self.count)
```

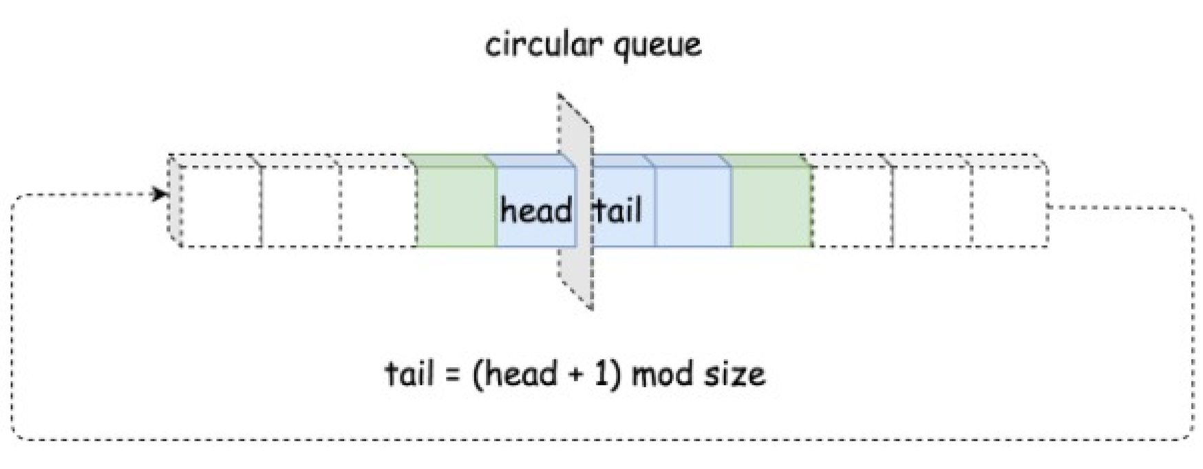
Complexity

- Time Complexity: $\mathcal{O}(1)$, as we explained in intuition.
- Space Complexity: $\mathcal{O}(N)$, where N is the size of the moving window.

Approach 3: Circular Queue with Array

Intuition

Other than the *deque* data structure, one could also apply another fun data structure called `circular queue`, which is basically a queue with the circular shape.



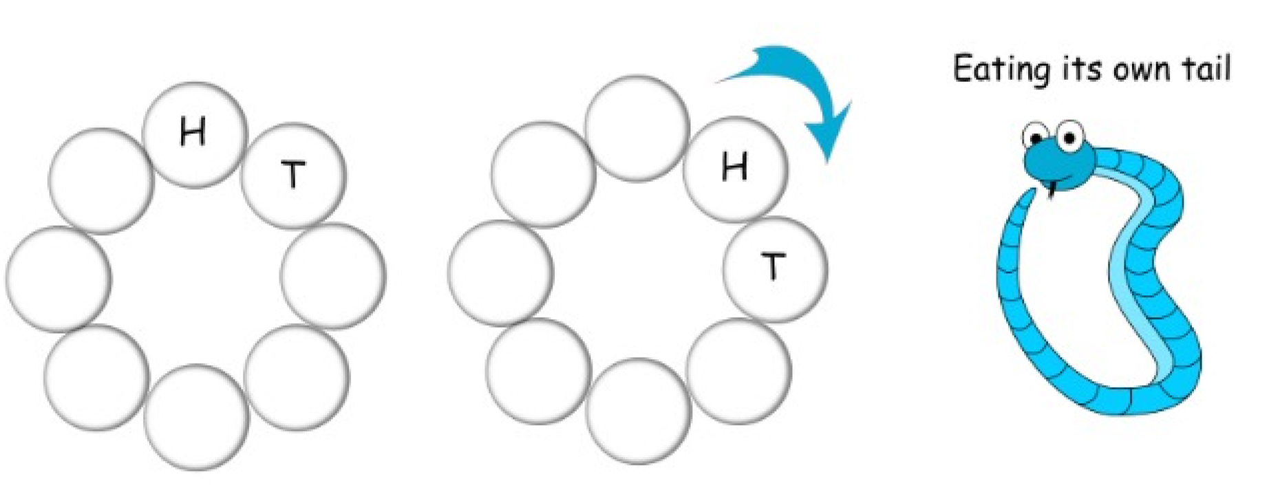
- The major advantage of circular queue is that by adding a new element to a full circular queue, it automatically discards the oldest element. Unlike deque, we do not need to explicitly remove the oldest element.
- Another advantage of circular queue is that a single index suffices to keep track of both ends of the queue, unlike deque where we have to keep a pointer for each end.

Algorithm

No need to resort to any library, one could easily implement a circular queue with a fixed-size array. The key to the implementation is the correlation between the index of `head` and `tail` elements, which we could summarize in the following formula:

$$\text{tail} = (\text{head} + 1) \quad \text{mod size}$$

In other words, the `tail` element is right next to the `head` element. Once we move the head forward, we would overwrite the previous tail element.



Java | Python | Copy

```python
class MovingAverage:
    def __init__(self, size: int):
        self.size = size
        self.queue = [0] * self.size
        self.head = self.window_sum = 0
        # number of elements seen so far
        self.count = 0

    def next(self, val: int) -> float:
        self.count += 1
        # calculate the new sum by shifting the window
        tail = (self.head + 1) % self.size
        self.window_sum = self.window_sum - self.queue[tail] + val
        # move on to the next head
        self.head = (self.head + 1) % self.size
        self.queue[self.head] = val
        return self.window_sum / min(self.size, self.count)
```

Complexity

- Time Complexity: $\mathcal{O}(1)$, as we can see that there is no loop in the `next(val)` function.
- Space Complexity: $\mathcal{O}(N)$, where N is the size of the circular queue.

Analysis written by @liaison and @andvary

Rate this article:

Previous Next

Comments: 8 Sort By

Type comment here... (Markdown is supported)

Preview Post

acoolguy ★ 33 November 16, 2019 6:28 AM Report

I would mention a queue instead of a Deque here in the explanation. We aren't really using both the stack and queue aspects of this data structure. Instead we are simply using the deque as a queue.

25 Share Reply

SHOW 1 REPLY

fliess ★ 3 October 9, 2019 12:37 PM

I think in Java solution of "Approach 2: Double-ended Queue" it would be clearer to call the result of `queue.poll()` as **head** (as per javadoc) rather than ~~tail~~. Just to avoid confusion among those, who are not sure how java Deque does work internally. Minor though.

3 Share Reply

SHOW 1 REPLY

sh0gg0th ★ 16 April 17, 2020 7:51 AM

We don't need to keep a separate `count` variable, `len(self.queue)` has a time complexity of O(1).

1 Share Reply

Xyzzy123 ★ 33 March 21, 2020 1:15 PM

Here's a simple approach in Python. It uses a queue (specifically a deque that's limited to queue operations) to store the most-recent numbers in the sliding window. It calculates the sum(queue) / len(queue) at each pass.

This isn't quite as efficient as Approach 2, since it needs to sum the k numbers in the queue during each

Read More

1 Share Reply

SHOW 2 REPLIES

Sithis ★ 9531 October 7, 2019 5:00 AM Report

```
class MovingAverage {
    private final Queue<Integer> window;
    private final int maxSize;
    private double sum = 0.0;
```

Read More

1 Share Reply

SHOW 1 REPLY

roireshef ★ 4 April 14, 2020 6:07 PM

Python implementation, O(1) runtime complexity with O(N) space complexity, beats 98%:

```
from queue import deque
class MovingAverage:
    def __init__(self, size: int):
```

Read More

0 Share Reply

shangl ★ 5 December 15, 2019 8:31 PM

My solution using the moving average formula:

```
double next(int val) {

    // a b c d e f g
```

Read More

0 Share Reply

thegreenquaga ★ 16 November 3, 2019 2:52 AM Report

Simple Java Solution with a fixed sized array.

```
class MovingAverage {

    int[] arr;
```

Read More

0 Share Reply