Maximum Average Subarray I

Given an array consisting of `n` integers, find the contiguous subarray of given length `k` that has the maximum average value. And you need to output the maximum average value.

Example 1:

```
Input: [1,12,-5,-6,50,3], k = 4
Output: 12.75
Explanation: Maximum average is (12-5-6+50)/4 = 51/4 = 12.75
```

Note:

1. 1 k n
2. Elements of the given array will be in the range [-10,000, 10,000].

Solution 1

```
public class Solution {
    public double findMaxAverage(int[] nums, int k) {
        long sum = 0;
        for (int i = 0; i < k; i++) sum += nums[i];
        long max = sum;

        for (int i = k; i < nums.length; i++) {
            sum += nums[i] - nums[i - k];
            max = Math.max(max, sum);
        }

        return max / 1.0 / k;
    }
}
```

written by shawngao original link here

Solution 2

Using prefix sums (where `sums[i]` is the sum of the first `i` numbers) to compute subarray sums.

```
def findMaxAverage(self, nums, k):
    sums = [0] + list(itertools.accumulate(nums))
    return max(map(operator.sub, sums[k:], sums)) / k
```

NumPy version (requires `import numpy as np`):

```
def findMaxAverage(self, nums, k):
    sums = np.cumsum([0] + nums)
    return int(max(sums[k:] - sums[:-k])) / k
```

written by StefanPochmann original link here

Solution 3

Java 8's reduce functional feature comes in handy when we have to go through an array and calculate a result out of its contents. This solution is a bit more convoluted than the straightforward approach of using a for loop, but its a good way to look at the reduce function for single pass array problems.

The trick in using the reduce function boils down to following:

1. As we look at every element in the array, we need to access two information,
    - the rolling sum
    - the max sum
      These are the two variables we can maintain as the accumulator, that gets passed down to each iteration.
2. When using Java's reduce and to reduce the integer array to anything other than the type of the array, we need to use the 3 parameter version of reduce.

which is

```
<U> U reduce(U identity,
         BiFunction<U,? super T,U> accumulator,
         BinaryOperator<U> combiner)
```

- identity refers to the initial value of the accumulator, in our case, we are going to accumulate both sum and maxSum in an integer[].
- accumulator, is a Function that takes the
    - the same integer array that gets passed down through each iteration which is the acc int[]
    - and returns the new integer[], which will contain the updated sum and the updated maxSum
- combiner is not used unless we have a parallel stream, so its of no significance in this context

```
public static double findMaxAverage(int[] nums, int k) {
        int sum = IntStream.range(0, k).map(i -> nums[i]).sum();
        return IntStream.range(k, nums.length).boxed().reduce(
                new int[] {sum - nums[0], sum},
                (arr, i) -> new int[] {arr[0] + nums[i] - nums[i-k+1], Math.max(arr
[1], arr[0] + nums[i])},
                (x, y) -> x)[1] / (double)k;

    }
```

written by johnyrufus16 original link here