Maximum Average Subarray II

Given an array consisting of `n` integers, find the contiguous subarray whose **length is greater than or equal to** `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:
when length is 5, maximum average value is 10.8,
when length is 6, maximum average value is 9.16667.
Thus return 12.75.
```

Note:

1. 1 k n
2. Elements of the given array will be in range [-10,000, 10,000].
3. The answer with the calculation error less than 10^{-5} will be accepted.

Solution 1

Let d(x, y) be the density of segment [x, y], ie. `d(x, y) = (A[x]+...+A[y]) / (y-x+1)`. It can be computed quickly with prefix sums.

Now we refer to section 3 of *Kai-min Chung, Hsueh-I Lu - An Optimal Algorithm for the Maximum-Density Segment Problem. 2008.*

For each ending index j, the current interval for i under consideration, [0, j-K+1] (minus parts on the left we have already discarded), has been decomposed into *minimum* density segments of longest length [hull[i], hull[i+1]-1], and we discard these segments as appropriate. That is, for each i in increasing order, hull[i+1] is the largest index in [hull[i], j-K+1] so that [hull[i], hull[i+1]-1] has minimum density.

This is simply a lower hull of candidate points i, in a geometric interpretation where d(a, b) = the slope of the line segment `(a, P[a]) to (b+1, P[b+1])`. Then, we can prove that discarding components with lower density than our current candidate d(hull[0], j) must leave us with the highest density option remaining.

```
def findMaxAverage(self, A, K):
    N = len(A)
    P = [0]
    for x in A:
        P.append(P[-1] + x)

    def d(x, y):
        return (P[y+1] - P[x]) / float(y+1-x)

    hull = collections.deque()
    ans = float('-inf')

    for j in xrange(K-1, N):
        while len(hull) >= 2 and d(hull[-2], hull[-1]-1) >= d(hull[-2], j-K):
            hull.pop()
        hull.append(j-K + 1)
        while len(hull) >= 2 and d(hull[0], hull[1]-1) <= d(hull[0], j):
            hull.popleft()
        ans = max(ans, d(hull[0], j))

    return ans
```

written by awice original link here

Solution 2

(nums[i]+nums[i+1]+...+nums[j])/(j-i+1)>x
=>nums[i]+nums[i+1]+...+nums[j]>x*(j-i+1)
=>(nums[i]-x)+(nums[i+1]-x)+...+(nums[j]-x)>0

```
public class Solution {
    boolean check(int[] nums,int k,double x) //Check whether we can find a subarray whose average is bigger than x
    {
        int n=nums.length;
        double[] a=new double[n];
        for (int i=0;i<n;i++) a[i]=nums[i]-x; //Transfer to a[i], find whether there is a subarray whose sum is bigger than 0
        double now=0,last=0;
        for (int i=0;i<k;i++) now+=a[i];
        if (now>=0) return true;
        for (int i=k;i<n;i++)
        {
            now+=a[i];
            last+=a[i-k];
            if (last<0)
            {
                now-=last;
                last=0;
            }
            if (now>=0) return true;
        }
        return false;
    }
    public double findMaxAverage(int[] nums, int k) {
        double l=Integer.MIN_VALUE,r=Integer.MAX_VALUE;
        while (r-l>0.000004) //Binary search the answer
        {
            double mid=(l+r)/2;
            if (check(nums,k,mid)) l=mid; else r=mid;
        }
        return r;
    }
}
```

written by KakaHiguain original link here

Solution 3

We binary search on the answer. Let `P[i] = A[0] + A[1] + ... + A[i-1]`, the i-th prefix sum under A.

Let's focus our attention on `possible(x)`, a function that is true iff it is possible to have an average of at least `x`. Consider the elements `B = [a-x for a in A]` with corresponding prefix sum `Q[i] = P[i] - i*x` under B.

We want to know if there is some `>= K` length subarray in B with average at least zero. Suppose the subarray is `B[i] + B[i+1] + ... + B[j] = Q[j+1] - Q[i]`. To check whether this quantity is positive, for any `j`, and any `i <= j - K + 1`, we should check whether `Q[j+1] >= min_{i <= j-K+1} Q[i]`. Keeping a running minimum `m` of this array Q, we can check this in linear time.

Unfortunately, the time constraint on Python solutions is fairly tight, so we need another trick to avoid TLE. If a segment has the biggest average and we break it into two pieces, one of its pieces also has at least the same average. When the length is `>= 2*K`, we can split it into pieces of at least length `K`, with the largest such piece being less than length `2*K`.

Thus, we only need to check segments of length `K <= L < 2*K` to find an instance of the maximum average. When K is small, this admits an O(NK) solution that we use instead. Our solution in that case is identical to *Maximum Average Subarray I*, repeated K times.

```
def findMaxAverage(self, A, K):
    N = len(A)
    P = [0]
    for x in A:
        P.append(P[-1] + x)

    if K < 100:
        ans = float('-inf')
        for k in xrange(K, min(2*K, N+1)):
            best_sum = max(P[i+k] - P[i] for i in xrange(N-k+1))
            ans = max(ans, best_sum / float(k))
        return ans

    def possible(x):
        m = P[0]
        for i, v in enumerate(P):
            m = min(m, v-i*x)
            if i+K == len(P): break
            if P[i+K] - (i+K)*x >= m:
                return True
        return False

    lo, hi = min(A), max(A)
    while hi - lo > .00001:
        mi = (lo + hi) / 2.0
        if possible(mi):
            lo = mi
        else:
            hi = mi
    return lo
```

written by awice original link here

From Leetcoder.