近似平均数

almost

【题意简述】

定义 n 个数 $X_1, X_2, ..., X_n (n>1)$ 的近似平均数为 $\frac{\sum_{i=1}^{n} Xi}{n-1}$

对于给出的长度为 N 的一个序列 A，要求回答 Q 个询问。

每个询问会给出 $L, R (1<= L < R <= N)$，请找出 a 与 b $(L <= a < b <= R)$ 使得 $A_a, A_{a+1}, \ldots, A_b$ 的近似平均数最大。

$N <= 10^5, Q <= 3\times 10^4$

【算法分析】

算法一：

对于每一次询问，枚举 a 与 b，通过预处理的前缀和计算 $A_a \sim A_b$ 的近似平均数更新答案。

时间复杂度： $O(QN^2)$

期望得分： $10 \sim 15$

算法二：

由于一些测试点的 N 比较小，我们可以事先暴力计算出每一组 (a,b) 的答案，计为 $f[a][b]$，再通过动态规划得到每一组 (L,R) 的答案 g[L][R]，转移方程式为 g[i][j]=max(g[i+1][j],g[i][j-1],f[i][j])，该算法可以 $O(1)$ 回答所有询问，但需要 N^2 的储存空间。

时间复杂度： $O(N^2+Q)$

期望得分： $25 \sim 35$

算法三：

在算法一中，为了求得最优的 (a,b) 我们要枚举它们两个量，这一点似乎是很不优的，

重复查询了很多已知的信息，我们有没有办法减少枚举其中的一个呢？答案是肯定的，这也正是这个问题的关键所在，当左界（或右界）确定的时候，我们可以使用高效的办法在某个范围内查询出最优的右界（或左界）。

我们设 A 序列的前缀和序列为 S，即 $S_i=\sum_{j=1}^{i}Aj$，把 (i,S_i) 看成一个二维坐标系中的点，当左界 a 确定时，我们在 $[x,y]$ 的范围内寻找最优的 b，实际上也就是寻找最优的 (b,S_b) 使得 (a,S_{a-1}) 与它的连线斜率最大。这样一来问题就不那么难办了，如果能够事先建立起 $[x,y]$ 内的一个凸壳，我们就可以直接用二分的方法查询最优的右界了。

综上我们得到了一个较为高效且所需空间很少的算法，对于一组询问 (L,R)，从 R 开始到 L 为止不断加入其对应的点，并维护凸包以支持查询操作，用所有查询中得到的最优值作为答案。

时间复杂度： $O(QN\log N)$

期望得分： $35\sim 45$

算法四：

我们尝试将算法二与算法三融合起来。

在算法二中，单次询问的回答时间可以达到 $O(1)$，之所以这么快是因为我们做了很多预处理操作，花了大量的空间来存储这些预处理计算出的值，但随着问题规模的增大，我们再也无法花费如此多的时间预处理出所有的东西，也没有足够的空间将计算出的东西保存下来。但是仔细想想，我们真的需要事先知道那么多东西吗？还是可以通过适量地增加查询的时间来得到一个完美地平衡呢？

对，分块在这里派上用场了，我们把整个序列 A 均匀地分成 T 段，每一段的长度 $Len=\frac{N}{T}$。以每两段的分界点为起点，分别往左和往右使用算法三，我们可以求得所需要预处理出的 $f[i][j]$，表示 $(L,R)=(i,jLen)$ 或 $(jLen,i)$ 时的答案。

对于一次查询 (L,R)，如果 L 与 R 分在了一块，我们可以直接用算法三处理，否则我们可以用预处理的 f 解决一大部分问题。若最优的 (a,b) 满足 $a>\left\lceil\frac{L}{Len}\right\rceil Len$，那么可以通过 $f[R][\left\lceil\frac{L}{Len}\right\rceil]$ 得到答案；若最优的 (a,b) 满足 $b<\left\lfloor\frac{R}{Len}\right\rfloor Len$，那么可以通过 $f[L][\left\lfloor\frac{R}{Len}\right\rfloor]$ 得到答案，即 $Ans=M\text{ax}\{f[R][\left\lceil\frac{L}{Len}\right\rceil],f[L][\left\lfloor\frac{R}{Len}\right\rfloor]\}$。但是当然不可忽

视的是，当最优的(a,b)满足 a 与 L 分在一块并且 b 与 R 分在一块时，我们还没有纳入计算，事实上解决方法也很简单，再利用算法三建立出$\left\lfloor \frac{R}{Len} \right\rfloor Len \sim R$的凸壳，枚举$L \sim \left\lceil \frac{L}{Len} \right\rceil Len$的每个位置作为左界求出最优右界更新答案就行了。

最后，我们来考虑一下算法的复杂度，预处理部分由于分出了 T 块，所以总共要往左右做 T 次算法三，复杂度为$O(TN\log N)$；而询问部分由于每一块的长度$Len=\frac{N}{T}$，因此复杂度为$O(\frac{NQ\log N}{T})$。算法总的复杂度为$O((\frac{NQ}{T}+TN)\log N)$。当$\frac{NQ}{T}=TN$，即$T=\sqrt{Q}$时，复杂度取到最小值$O(N\sqrt{Q}\log N)$。

时间复杂度：$O(N\sqrt{Q}\log N)$

期望得分：$90\sim100$

【附】

almost.cpp
C++ Source File
4 KB