T2 - Interactive 题解

题目简述

题目大意：
给定一个长度为 N 的整数序列 a 和一个目标值 k。定义一个区间是「有互动性」的，当且仅当该区间的**最大子段和** $\geq k$。
现有 Q 次询问，每次给定 l，求序列中有多少个**长度小于** l 的区间是有互动性的。

- $N, Q \leq 2 \times 10^5$。
- 序列元素 a_i 可正可负。

The Key：

1. **单调性**：如果一个区间是有互动性的，那么任何包含它的更大区间也一定是有互动性的。
2. **贡献转化**：统计“长度小于 l”的区间数量，可以转化为对每种长度的区间进行计数，最后做前缀和。

子任务

Subtask 1：$N, Q \leq 10^3$

由于规模较小，我们可以枚举所有的区间 $[i, j]$，用 $O(N)$ 暴力或 $O(1)$ 的动态规划计算每个区间的最大子段和。
预处理出长度为 len 的互动区间数量 $cnt[len]$，询问时输出 $\sum_{i=1}^{l-1} cnt[i]$。

时间复杂度：$\mathcal{O}(N^2 + Q)$

Subtask 2 & 3：特殊性质

- $a_i \geq 0$：最大子段和就是区间全体和。可以使用双指针（滑动窗口）维护。
- $k = 1$：最大子段和 ≥ 1 等价于区间内至少存在一个正数。

Subtask 4：Q = 1

由于只有一次询问 l，在通过双指针找到每个左端点 i 对应的最小有效右端点 j 后，直接判断哪些区间满足长度 $j-i+1<l$，并统计总数即可。

正解

> 根据**单调性**，对于每一个确定的左端点 i，一定存在一个最小的右端点 j，使得区间 $[i,j]$ 是有互动性的。
> 一旦 $[i,j]$ 满足条件，那么所有以 i 为左端点且右端点 $\geq j$ 的区间（即 $[i,j],[i,j+1],\ldots,[i,N]$）全都是有互动性的。
>
> 这些满足条件的区间长度分别为：$(j-i+1),(j-i+2),\ldots,(N-i+1)$

利用双指针维护左端点 l 和右端点 r。我们需要快速查询当前窗口 $[l,r]$ 的最大子段和。使用**线段树**维护区间最大子段和。

当窗口不满足 $\geq k$ 时，右移 r；否则尝试右移 l。对于每个 l，我们找到了最小的 r。这意味着长度在区间 $[r-l+1,N-l+1]$ 之间的所有长度贡献都要 $+1$。这是一个**区间加**操作，可以使用**差分数组** `diff` 来维护。

对差分数组做第一次前缀和，得到 `cnt[len]`：长度为 `len` 的互动区间总数。

对 `cnt[len]` 做第二次前缀和，得到 `sum[len]`：长度**小于等于** `len` 的互动区间总数。

询问长度小于 l，即输出 `sum[l - 1]`。

时间复杂度：$\mathcal{O}(N\log N+Q)$

```cpp
//官方 std
#include <bits/stdc++.h>
#define uwu return 0;
using namespace std;

const int SIZE = 2e5 + 5;

struct node {
    long long l_max, r_max, ans, sum;
    node() : l_max(-1e18), r_max(-1e18), ans(-1e18), sum(0) {};
    node(long long v) {
        long long val = max(v, 0LL);
        l_max = r_max = ans = v;
```

```
        sum = v;
    }
} SEGTree[4 * SIZE];

node join(node n1, node n2) {
    node res;
    res.sum = n1.sum + n2.sum;
    res.l_max = max(n1.l_max, n1.sum + n2.l_max);
    res.r_max = max(n2.r_max, n2.sum + n1.r_max);
    res.ans = max({n1.ans, n2.ans, n1.r_max + n2.l_max});
    return res;
}

void build(int L, int R, int id, const vector<int>& vec) {
    if (L == R) {
        SEGTree[id] = node(vec[L]);
        return;
    }
    int M = (L + R) / 2;
    build(L, M, 2 * id, vec);
    build(M + 1, R, 2 * id + 1, vec);
    SEGTree[id] = join(SEGTree[2 * id], SEGTree[2 * id + 1]);
}

node query(int ql, int qr, int L, int R, int id) {
    if (ql ≤ L && R ≤ qr) return SEGTree[id];
    int M = (L + R) / 2;
    if (qr ≤ M) return query(ql, qr, L, M, 2 * id);
    if (ql ≥ M + 1) return query(ql, qr, M + 1, R, 2 * id + 1);
    return join(query(ql, M, L, M, 2 * id), query(M + 1, qr, M +
1, R, 2 * id + 1));
}

long long diff[SIZE];

int main() {
    cin.tie(0), ios::sync_with_stdio(0);
    int N; long long k;
    cin >> N >> k;
    vector<int> vec(N);
    for (int &i : vec) cin >> i;

    build(0, N - 1, 1, vec);
```

```
56
57      int ptr_r = 0;
58      for (int ptr_l = 0; ptr_l < N; ptr_l++) {
59          ptr_r = max(ptr_r, ptr_l);
60          while (ptr_r < N && query(ptr_l, ptr_r, 0, N - 1, 1).ans
    < k) {
61              ptr_r++;
62          }
63
64          if (ptr_r < N) {
65              diff[ptr_r - ptr_l + 1]++;
66              diff[N - ptr_l + 1]--;
67          }
68      }
69
70      for (int i = 1; i ≤ N; i++) diff[i] += diff[i - 1];
71      for (int i = 1; i ≤ N; i++) diff[i] += diff[i - 1];
72
73      int Q; cin >> Q;
74      while (Q--) {
75          int l; cin >> l;
76          if (l ≤ 1) cout << 0 << "\n";
77          else cout << diff[min(l - 1, N)] << "\n";
78      }
79
80      uwu;
81  }
```