灯笼

Tag： 树状数组 、 sort 、 前缀和 、 思维

这个问题的本质是一个双重限制条件的一维数点问题，但是只要逐步拆解限制条件，并不是一道很难的题目。

用到的trick比较多，比如使用前缀和的差值表示一段区间，利用单调性预处理限制区间等。

题目的主体部分其实只用一个离线的树状数组就解决了。

30pt

直接讲最快的O(N^2)的算法，我们考虑题目中的两个限制条件

1. 权值和大于

2. 种类数小于

因为数字有正有负，1不具备单调性，但是1可以用前缀和来维护，换句话说就是给定l，r，可以O(1)的复杂度知道sum(l,r)。

对于限制2，当区间左端点l固定时，随着r端点向右扩展，包括颜色的种类数一定是单调递增的。所以对于每一个l，直接向右滑动，直到发现颜色的种类数多余M就break即可。

维护颜色可以使用一个桶数组维护颜色出现的次数，当发现桶数组中某个颜色出现的次数从0变为1时，就记录cnt++。

对于桶数组的声明方式，在c语言下有一个小trick，就是比如你想用数组的负数下标，可以这样。

int base[MAXN\*2];

int \*a=base+MAXN;

这样就相当于开了一个a数组，并且a数组的有效下标是左闭右开区间。

另30pt

我这里分的3个另10pt其实都是提示性的测试点，大家打比赛的时候注意子任务subtask除了部分得分的作用以外，有一些是具有引导性的，但是也不一定都是正向的引导，有些就比较偏，还是需要自己斟酌一下。

1.另10pt M=N

当M=N时，限制2不生效，题目转化成求区间和大于X的区间数目。

因为区间和可能很大，不好维护（维护起来要离散化），我们转换思路，首先构造一个结构体{前缀和,下标}，然后按照前缀和的顺序排序。

换句话说就是先求的前缀和数组，然后把前缀和数组绑定下标后，按照前缀和的大小进行排序。

这样排序后，右侧的前缀和的值一定大于左侧，取原数组的一段区间可以视为是在前缀和数组中去取两个元素。

问题转化成给一个有序数组，任取两个元素的差值大于X。

你发现转化后的问题，假设左边取的数字下标为j，右侧取的数字下标为i，如果[j,i]是符合条件的，则[j,i+1],[j-1,i]必定也符合条件（因为数组从左到右递增）

所以使用一个滑动窗口的写法，枚举i，然后while移动j。讲符合条件的区间端点用树状数组在原数组中标出，然后直接计算即可。

2.另10pt X=-1e9

当x=-1e9时，限制1不生效，题目转化为求区间元素种类数小于M的区间数目。

这个其实就将暴力改成一个划窗的枚举方式就好了，也就是枚举固定的右端点i，这个时候如果发现种类数较多，就把左端点j向右移动一下即可。

3.另10pt like>=0

这个是作为提示用的测试点，和 100pt 的做法的区别仅在不用对前缀和进行sort，因为每个数都>=0，那么前缀和肯定是单调增的，实际上是一个起提示作用的点。（提示你需要处理的前缀和必须单调递增）。

100pt

100pt 就是将上面的几个sub task合起来做一下。

首先你想一下对于1、2这两个限制条件，它们有没有联合产生的新影响。

显然是没有的，那就说明，这两个限制条件可以单独拆解，单独做。

首先按照X=-1e9的思路，预处理出每一个i，最左边延伸到最小的合法的j是谁，我们记录limit[x]表示，右端点为x时，在不考虑限制1的情况下，合法解左端点的最小值。

然后接下来就按照M=N这个subtask里面去做，前缀和，sort，用树状数组标记符合条件的下标。

区别在于原本是求一个前缀：

ans+=qt(id[i]-1)

现在不能全求，必须限制解的下标在limit的范围内，所以做一个前缀和的差分容斥即可，改为

if(id[i])ans+=qt(id[i]-1)-qt(limit[id[i]]-1);

例：n=7,m=3,x=5

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 原数组a[i] | 0 | -2 | 5 | -1 | 3 | 5 | -1 | 3 |
| 前缀和sum[i] | 0 | -2 | 3 | 2 | 5 | 10 | 9 | 12 |
| 将前缀和排序后第i名的原序号id[i] | 1 | 0 | 3 | 2 | 4 | 6 | 5 | 7 |
| 原序列中第i个数据最左延伸至limit[i] | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| 按前缀和排序后的顺序sum[id[i]] | -2 | 0 | 2 | 3 | 5 | 9 | 10 | 12 |

**i=3**

**j=0**

当i=3，j=0时，sum[id[i]]-sum[id[j]]>=x，说明出现了一个可能的区间[id[j]+1,id[i]]，可以在树状数组中标记id[j]+1，因为是按sum[id[i]]从小到大的顺序扫描i的，以后出现的i对应的sum[id[i]]一定更大，以id[j]+1为左端点的区间和也必然是满足x限制的。

这个题的每一个sub task都不是很难，但是综合起来还是需要一定的综合能力。

时间复杂度O(NlogN) ，空间复杂度O(N) 。

#include<bits/stdc++.h>

using namespace std;

const int MAXN=100005;

int a[MAXN],base[MAXN],sum[MAXN],id[MAXN],limit[MAXN],bit[MAXN],C,n,k,x,p;

int \*cnt=base+50000;

long long ans;

void ct(int x)

{

 x++;

 while(x<=n+1)bit[x]++,x+=(x&-x);

}

int qt(int x)

{

 int ret=0;

 x++;

 while(x)ret+=bit[x],x-=(x&-x);

 return ret;

}

int cal\_diff()

{

 int ret=0;

 for(int i=1;i<=n;++i)

 {

 ret+=a[i]>=x;

 }

 return ret;

}

int main()

{

 scanf("%d %d %d",&n,&k,&x);

 for(int i=1;i<=n;++i)

 {

 scanf("%d",&a[i]);

 sum[i]=sum[i-1]+a[i];

 C+=(cnt[a[i]]++)==0;

 while(C>k)C-=(--cnt[a[++p]])==0;

 limit[i]=p;

 id[i]=i;

 }

 sort(id,id+1+n,[](const int &x,const int &y){

 return sum[x]<sum[y];

 });

 p=-1;

 for(int i=0;i<=n;++i)

 {

 while(p<n&&sum[id[i]]-sum[id[p+1]]>=x)ct(id[++p]);

 if(id[i])ans+=qt(id[i]-1)-qt(limit[id[i]]-1);

 }

 printf("%lld\n",ans\*2-cal\_diff());

 return 0;

}