《人间应又雪》解题报告

福建师范大学附属中学 黄佳旭

1 简要题意

给定长度为 n 的数组 $a_{1\sim n}$ 和常数 c，保证 $0\le a_i\le m$ 且 a_i 为整数。

每次操作可以选择一个 $x\in[1,n]$，令 $a_x:=a_x-c$，然后从以下两种选择中选取一种：

- $\forall 1\le i\le x,a_i:=a_i-1$。
- $\forall x\le i\le n,a_i:=a_i-1$。

每次操作后 a_i 对 0 取 $\max$。

求最少进行多少次操作，可以使得 $\forall 1\le i\le n,a_i=0$。

每个测试点有多组测试数据。

2 数据范围

对于 100% 的数据，$1\le T\le 10^5$，$1\le n,m\le 5\times 10^5$，$\sum n,\sum m\le 10^6$，$0\le a_i\le m$，$0\le c\le 5\times 10^5$。

子任务编号	n	m	特殊限制	分值	子任务依赖
1	$\le 5\times 10^5$	$\le 5\times 10^5$	$c=0$	2	
2	$\le 5\times 10^5$	≤ 2	无	3	
3	≤ 5	≤ 5	$T\le 10$	5	
4	≤ 50	≤ 50	$\sum n,\sum m\le 200$	10	3
5	≤ 300	≤ 300	$\sum n,\sum m\le 600$	10	4
6	≤ 2000	≤ 2000	$\sum n,\sum m\le 4000$	10	5
7	$\le 5\times 10^4$	$\le 5\times 10^4$	$c\le 20,\sum n,\sum m\le 10^5$	20	
8	$\le 5\times 10^4$	$\le 5\times 10^4$	$\sum n,\sum m\le 10^5$	15	$6,7$
9	$\le 5\times 10^5$	$\le 5\times 10^5$	$c\le 20$	10	$1,7$
10	$\le 5\times 10^5$	$\le 5\times 10^5$	无	15	$2,8,9$

3 解题过程

3.1 算法一

对于 $c=0$ 的情况，每次操作贪心的让所有 a_i 减 1，答案就是 a_i 的最大值。

时间复杂度 $O(n)$，期望通过子任务 1。

3.2 算法二

通过算法一可以得知答案不超过 m，对于 $m \leq 2$ 的情况，我们只需要判断是否可以用 0 次或 1 次操作解决。

答案为 0 即初始 a_i 全为 0，答案为 1 我们可以枚举操作的位置和类型，容易判断是否可行。

时间复杂度 $O(n)$，期望通过子任务 2。

3.3 算法三

对于 n, m 很小的情况，多次随机化取最优解，或者搜索剪枝，期望通过子任务 3。

3.4 算法四

为了方便，**下面我们将 令** $\forall 1 \leq i \leq x, a_i := a_i - 1$ **的操作 称为向左的操作，反之称为向右的操作。**

考虑一个暴力的 DP，记 $f_{i,j,k}$ 表示考虑了前 i 个数，有 j 个向右的操作，目前 $1 \sim i$ 中 a 的最大值为 k，这种情况下最小的操作数。

转移枚举这个位置进行了 x 个向左的操作和 y 个向右的操作，简单讨论一下即可。

参考代码如下：

```
for(int i=1;i<=n;++i){
    for(int j=0;j<=m;++j)for(int k=0;k<=m;++k)if(f[i-1][j][k]<inf){
        for(int l=0;l<=j;++l)for(int o=0;o+k<=m;++o){
            int v=max(0,a[i]-(l+o)*c-l-o-k); //v 表示 a[i] 当前操作后的值
            U(f[i][max(j-l,v)][k+o],f[i-1][j][k]+l+o);
        }
    }
}
```

时间复杂度 $O(nm^4)$，常数较小，期望通过子任务 $3, 4$。

3.5 算法五

可以发现，**最优情况下一定存在一个** p，**满足所有向左的操作的** x **都** $\geq p$，**所有向右的操作的** x **都** $\leq p$。否则说明存在向左的操作 x_1 和向右的操作 x_2，$x_1 < x_2$，那我们可以交换 x_1 和 x_2 的方向，这样是不劣的。

因此可以枚举 i, j, k，表示 $p = i$，有 k 个向左的操作，j 个向右的操作，判断是否可行。

考虑假设已经确定了有 k 个向左的操作，那么我们要怎么安排向右的操作？

可以贪心的考虑，认为每个数都已经减去了 k，然后从 $1 \sim n$ 依次确定每个位置的操作。

设 $f_{i,k}$ 表示钦定有 k 个向左的操作，那么清除 $a_1 \sim a_{i-1}$ 要多少个向右的操作，$g_{i,j}$ 表示钦定有 j 个向右的操作，那么清除 $a_{i+1} \sim a_n$ 要多少个向左的操作，这是容易 $O(nm)$ 递推求出的。

参考代码如下：

```
for(int ad=0;ad<=m;++ad){
    int x=ad;
    for(int j=1;j<=n;++j){
        f[j][ad]=x-ad;
        if(a[j]>x){
            int t=(a[j]-x-1)/(c+1)+1;
```

```
            x+=t;
        }
    }
    x=ad;
    for(int j=n;j;--j){
        g[j][ad]=x-ad;
        if(a[j]>x){
            int t=(a[j]-x-1)/(c+1)+1;
            x+=t;
        }
    }
}
```

那么 (i,j,k) 合法，当且仅当，$f_{i,k}\le j,g_{i,j}\le k,j+k+(j+k-f_{i,k}-g_{i,j})\times c\ge a_i$。

因为我们有 j 个向右的操作，而已经用了 $f_{i,k}$ 个向右的操作，所以我们会在 i 上用 $j-f_{i,k}$ 个向右的操作，向左的操作同理。

时间复杂度 $O(nm^2)$ 瓶颈在于枚举 i,j,k，期望通过子任务 5。

可以发现，随着向右的操作数不断增加，向左操作的需求会不断减小。

那么我们在枚举 i,j 的时候，可以用指针维护最小的 k。

参考代码如下：

```
for(int i=1;i<=n;++i){
    for(int j=0,k=m;j<=m;++j){
        while(k&&check(i,j,k-1))--k;
        if(check(i,j,k))ans=min(ans,j+k);
    }
}
```

时间复杂度 $O(nm)$，期望通过子任务 6。

3.6 算法六

考虑二分答案，假设用 t 次操作，判断是否可行。

二分答案的好处是，对于一个 j，可以确定出 $k=t-j$。

那么对于一对确定的 j 和 k，我们要判断是否可行。

那么这时候我们就不要枚举分界点 i 了，我们可以求出 pl_j 表示有 $k=t-j$ 个向左的操作，用 j 个向右的操作能清空 $a_1\sim a_{pl_j-1}$，还剩下 cl_j 个向右的操作，注意这意味着 a_{pl_j} 无法用剩下的 cl_j 个操作清空，同样类似的定义 pr_k 和 cr_k，表示用 k 个向左的操作能清空 $a_{pr_k+1}\sim a_n$，还剩下 cr_k 个向左的操作。

如果 $pl_j<pr_k$，那么意味着 a_{pl_j} 和 a_{pr_k} 是无法清空的，不可行。

如果 $pl_j>pr_k$，说明每个 a 都至少被清空一次了，一定可行。

如果 $pl_j=pr_k$，我们还要考虑 a_{pl_j} 是否能被清空，那么需要 $j+k+(cl_j+cr_k)\times c\ge a_{pl_j}$。

这样每次二分后枚举 j 判断是否可行，暴力 $O(n)$ 求出 pl_j,cl_j,pr_k,cr_k。

时间复杂度 $O(nm\log m)$，期望通过子任务 6。

3.7 算法七

算法六的瓶颈在于求出所有的 pl_j, cl_j 和 pr_k, cr_k，考虑能否较快地求出。

因为 pl_j 和 pr_k 是对称的，我们仅需要考虑求出 pl_j 和 cl_j，翻转序列 a 后用同样的方法即可求出 pr_k, cr_k。

初始时 k 个向左的操作令所有 a_i 减 k，相当于是“画了条分界线”，表示 $a_i \le k$ 的 a_i 都被清空了。

那么如果遇到一个 $a_i > k$，**我们就要用一次向右的操作，令 $a_i := a_i - c$，同时分界线 $+1$**。

定义 $f_{i,k}$ 表示初始分界线为 k，到 i 位置分界线变为 $f_{i,k}$。

如果**某一位置 i 分界线 $> t$ 了**，说明我们的 j 次操作已经不够用了，我们就可以求出 $pl_j = i, cl_j = t - f_{i-1,k}$。

考虑 $f_{i-1,k}$ 到 $f_{i,k}$ 是怎么变化的，如果 $a_i \le f_{i-1,k}$ 则 $f_{i,k} = f_{i-1,k}$，否则 $f_{i,k} = f_{i-1,k} + \lceil \frac{a_i - f_{i-1,k}}{c+1} \rceil$。

我们将 f_i 看做关于 k 的函数来整体维护，可以发现满足 $f_i(k) \le f_i(k+1) \le f_i(k) + 1$，那么我们可以按照如下方式维护：

- 如果 $f_i(k_1) = f_i(k_2)$，直接将 k_1 和 k_2 合并。
- 如果 $f_i(k) > t$，求出 pl_{t-k} 和 cl_{t-k} 并将 k 删除。

那么相当于我们维护了一些点 $b_1, b_2, \cdots, b_u$，满足：

- $f_i(b_u) = t$。
- $\forall 1 \le x < u, f_i(b_{x+1}) = f_i(b_x) + 1$。

注意因为有合并操作这里每个点实际上代表了一个区间的值。

那么加入了一个 a_i，如果 $a_i \le t$，相当于从 $f_i(b_x) = a_i$ 的 b_x 开始，每隔 c 个点合并相邻的 2 个点。

对于 $a_i > t$ 的情况也是类似的，只是末尾的一部分点会被删除。

每次修改时暴力重构发生变化的部分，因为每隔 c 个点就会合并一次，点数少一，所以暴力重构的时间复杂度是 $O(mc)$。

总时间复杂度 $O((n + mc)\log m)$，期望通过子任务 7。

修改操作中删除 $> t$ 点是取出末尾的一段点求答案，容易线性维护。

合并操作需要支持删除一个点，查询当前序列中的第 k 个点，容易使用数据结构维护。

标程使用的是常数较小的树状数组二分，时间复杂度 $O(n \log m + m \log^2 m)$，期望通过子任务 8。

3.8 算法八

注意到算法七的瓶颈在于**求出每个位置 i 需要合并的点，这只与数组 a 有关，而与二分的答案 t 无关**。t 只与取出末尾的点有关。

所以考虑在二分前先进行预处理，模拟一遍修改过程，求出每个位置 i 会合并的点对，二分答案时直接遍历点对，如果还存在就合并。

预处理使用暴力重构是 $O(mc)$ 的，时间复杂度 $O(mc + (n+m)\log m)$，期望通过子任务 9。

使用数据结构维护，时间复杂度 $O((n+m)\log m)$，期望通过子任务 10。

参考代码实现：

```cpp
#include<bits/stdc++.h>
using namespace std;
const int N=5e5+5;
int T,tid,n,m,c,a[N],pl[N],cl[N],pr[N],cr[N];
int fa[N],tr[N],nx[N],tot;
bool dl[N];
vector<pair<int,int> >ml[N],mr[N];
inline void upd(int x){
    for(x=m-x+1;x<=m;x+=x&-x)--tr[x];
}
inline int qryk(int k){
    int x=0,v=0;
    for(int i=18;~i;--i)if(x+(1<<i)<=m&&v+tr[x+(1<<i)]<k)
        x+=1<<i,v+=tr[x];
    return m-x;
}
void calc(int t,vector<pair<int,int> >*g){
    nx[t+1]=t;
    for(int i=t+1;i<=m;++i)dl[i]=true;
    for(int i=0;i<=t;++i)fa[i]=i,nx[i]=i-1,dl[i]=false;
    for(int i=1;i<=n;++i){
        int nv=t;
        for(int j=nx[t+1];~j&&(t-nv)*c+t<a[i];j=nx[j]){
            pl[j]=i,cl[j]=t-nv,--nv,dl[j]=true;
            nx[t+1]=nx[j];
        }
        for(auto [u,v]:g[i])if(!dl[u]&&!dl[v])
            nx[u]=nx[v],fa[v]=u;
    }
    for(int i=nx[t+1];~i;i=nx[i])pl[i]=n+1;
    for(int i=t;~i;--i)pl[i]=pl[fa[i]],cl[i]=cl[fa[i]];
}
void Calc(vector<pair<int,int> >*g){
    for(int i=1;i<=m;++i)tr[i]=(i&-i);
    tot=m+1;
    for(int i=0;i<=m+1;++i)nx[i]=i-1;
    for(int i=1;i<=n;++i){
        int p=m-a[i]+1;
        while(p<tot){
            int u=qryk(p),v=nx[u];
            nx[u]=nx[v];
            g[i].emplace_back(u,v);
            --tot,upd(v);
            p+=c;
        }
    }
}
bool check(int t){
    reverse(a+1,a+n+1);
    calc(t,mr);
    for(int i=0;i<=t;++i)pr[i]=n-pl[i]+1,cr[i]=cl[i];
    reverse(a+1,a+n+1);
    calc(t,ml);
    for(int i=0;i<=t;++i){
        if(pl[i]>pr[t-i])return true;
        if(pl[i]<pr[t-i])continue;
```

```
        if(t+(cl[i]+cr[t-i])*c>=a[pl[i]])return true;
    }
    return false;
}
void init(){
    for(int i=1;i<=n;++i)ml[i].clear(),mr[i].clear();
    reverse(a+1,a+n+1);
    Calc(mr);
    reverse(a+1,a+n+1);
    Calc(ml);
}
void solve(){
    scanf("%d%d%d",&n,&m,&c);
    for(int i=1;i<=n;++i)scanf("%d",&a[i]);
    init();
    int l=0,r=m;
    while(l<r){
        int mid=l+r>>1;
        if(check(mid))r=mid;
        else l=mid+1;
    }
    printf("%d\n",l);
}
int main(){
    scanf("%d%d",&T,&tid);
    while(T--)solve();
    return 0;
}
```

4 参考资料

本题的灵感来源于 CF1876G Clubstep (https://codeforces.com/problemset/problem/1876/G)。

出题过程中与集训队员魏忠浩讨论过。