神秘物质（atom,2s,256MB)

【算法分析】

我们首先知道，把一个区间扩大，极差不会变小。

所有子区间的极差最大值就是整个区间的极差。

所有子区间的极差最小值就是区间内相邻两个数差的最小值。

那么我们用 Splay 维护子树内的三个信息：最大值、最小值、一个数和下一个数差的最小值。

插入和合并元素只要用 Splay 实现。

提取区间[L,R]只需要将 L-1 对应的结点旋到根，R+1 对应的结点旋到 L-1 下面，那么 R+1 的左子树就是这个区间，直接利用子树的根的信息回答询问即可。

实现的时候可以插入 0 和 n+1 两个虚拟结点。

假设 N,M 同阶，那么时间复杂度就是$O(N\log N)$。

【参考程序】

```
//陈煜翔
#include <bits/stdc++.h>

template <class T>
inline void read(T &x)
{
    static char ch;
    while (!isdigit(ch = getchar()));
    x = ch - '0';
    while (isdigit(ch = getchar()))
        x = x * 10 + ch - '0';
}

template <class T>
inline void putint(T x)
{
    static char buf[15], *tail = buf;
    if (!x) putchar('0');
    else
    {
        if (x < 0) x = ~x + 1, putchar('-');
        for (; x; x /= 10) *++tail = x % 10 + '0';
        for (; tail != buf; --tail) putchar(*tail);
    }
}

template <class T>
inline void tense(T &x, const T &y)
{
    if (x > y)
```

```
        x = y;
}

template <class T>
inline void relax(T &x, const T &y)
{
    if (x < y)
        x = y;
}

constintMaxN = 2e5 + 5;
constint INF = 0x3f3f3f3f;

struct node
{
    int fa, lc, rc, val, dlt, sze;
    int mind, minv, maxv;

    #define fa(x) tr[x].fa
    #define lc(x) tr[x].lc
    #define rc(x) tr[x].rc
    #define val(x) tr[x].val //val 表示结点 x 的权值
    #define dlt(x) tr[x].dlt //dlt 表示结点 x 和它在序列中下一个元素的差值
    #define sze(x) tr[x].sze//sze 表示子树大小
    #define mind(x) tr[x].mind //mind 表示子树中最小的相邻元素差值
    #define minv(x) tr[x].minv //minv 表示子树中最小的权值
    #define maxv(x) tr[x].maxv //maxv 表示子树中最大的权值
}tr[MaxN];

intrt, nT;
char s[10];
int n, m;
int a[MaxN];

inline bool which(int x)
{
    return rc(fa(x)) == x;
}

inline void upt(int x)//用 x 的左右儿子更新结点 x 的信息
{
    sze(x) = 1;
    mind(x) = dlt(x);
    minv(x) = maxv(x) = val(x);
```

```
    if (lc(x))
    {
        sze(x) += sze(lc(x));
        tense(mind(x), mind(lc(x)));
        tense(minv(x), minv(lc(x)));
        relax(maxv(x), maxv(lc(x)));
    }
    if (rc(x))
    {
        sze(x) += sze(rc(x));
        tense(mind(x), mind(rc(x)));
        tense(minv(x), minv(rc(x)));
        relax(maxv(x), maxv(rc(x)));
    }
}

inline void rotate(int x)
{
    int y = fa(x), z = fa(fa(x));
    int b = lc(y) == x ?rc(x) : lc(x);

    fa(x) = z, fa(y) = x;
    if (b) fa(b) = y;

    if (lc(z) == y)
        lc(z) = x;
    else
        rc(z) = x;

    if (lc(y) == x)
        rc(x) = y, lc(y) = b;
    else
        lc(x) = y, rc(y) = b;

    upt(y);
}

inline void Splay(int x, int tar) //将 x 旋转到 tar 下面
{
    while (fa(x) != tar)
    {
        if (fa(fa(x)) != tar)
            rotate(which(x) == which(fa(x)) ? fa(x) : x);
        rotate(x);
```

```
    }
    upt(x);
    if (!tar)
        rt = x;
}

inline void build(int&x, int l, int r) //刚开始的序列我们可以直接分治建树
{
    if (l > r) return;
    int mid = l + r >> 1;

    x = ++nT;

    val(x) = a[mid];
    dlt(x) = abs(a[mid] - a[mid + 1]);
    build(lc(x), l, mid - 1);
    build(rc(x), mid + 1, r);

    if (lc(x)) fa(lc(x)) = x;
    if (rc(x)) fa(rc(x)) = x;
    upt(x);
}

inline intquery_kth(int k)//查询目前在序列中编号为 k 的结点
{
    int x = rt;
    while (k)
    {
        if (sze(lc(x)) + 1 == k)
            return x;
        else if (sze(lc(x)) + 1 > k)
            x = lc(x);
        else
        {
            k -= sze(lc(x)) + 1;
            x = rc(x);
        }
    }
}

inline int split(int l, int r) //提取出区间[l,r]对应的子树
{
    int t1 = query_kth(l), t2 = query_kth(r + 2);
    Splay(t1, 0), Splay(t2, t1);
```

```
    return lc(t2);
}

intmain()
{
    freopen("atom.in", "r", stdin);
    freopen("atom.out", "w", stdout);

    mind(0) = minv(0) = INF;

    read(n), read(m);
    for (inti = 1; i<= n; ++i)
        read(a[i]);
    build(rt, 0, n + 1);

    for (inti = 1; i<= m; ++i)
    {
        scanf("%s", s + 1);
        int x, y;
        read(x), read(y);

        if (s[2] == 'e') //合并两个相邻结点
        {
            int u = split(x, x + 1), v = fa(u);
            val(u) = y;
            dlt(fa(v)) = abs(y - val(fa(v)));
            dlt(u) = abs(y - val(v));
            lc(u) = rc(u) = 0;

            upt(u);
            Splay(u, 0);
        }
        else if (s[1] == 'i')//插入结点
        {
            int u = split(x, x), v = fa(u);
            val(rc(u) = ++nT) = y;
            fa(nT) = u;

            dlt(u) = abs(y - val(u));
            dlt(nT) = abs(y - val(v));

            upt(nT);
            Splay(nT, 0);
```

```
        }
        else if (s[2] == 'a')//询问最大极差
        {
            int u = split(x, y);
            putint(maxv(u) - minv(u)), putchar('\n');
        }
        else //询问最小极差
        {
            int u = split(x, y - 1);
            putint(mind(u)), putchar('\n');
        }
    }

    return 0;
}
```

生产线(train, 4s, 1GB)

【解题报告】

1. 只有插入(I)操作和询问(Q)

将元素插入队列是最经典的平衡树操作，用 Treap 等平衡树实现即可维护队列。

接下来考虑如何求解询问的答案。首先是不仅要求某个区间最大的元素，当最大的元素有多个时还要求最中间的一个。为了解决这一问题，我们需要在平衡树的每个结点（代表的子树）上维护这个子树内数字的最大值及其数量。然后询问时，分两次遍历平衡树对应的区间，第一次求解区间内最大的数字及其个数 m，第二次寻找第$\left\lceil\frac{m}{2}\right\rceil$个最大数字的位置 x。

在求出最中间的最大数字的位置 x 后，我们考虑求解下式

$$\sum_{i=l}^{x-1}\begin{cases}x-i & if\ h_i > h_{i+1}\\ 0 & if\ h_i \le h_{i+1}\end{cases} + \sum_{i=x+1}^{r}\begin{cases}i-x & if\ h_i > h_{i-1}\\ 0 & if\ h_i \le h_{i-1}\end{cases}$$

可以看出以 x 为分界，左右两区间的求解是对称的，所以我们可以先考虑解决左侧的式子（另一侧的求解是类似的）：

$$\sum_{i=l}^{x-1}\begin{cases}x-i & if\ h_i > h_{i+1}\\ 0 & if\ h_i \le h_{i+1}\end{cases}$$

求解上式可以用类似于在线段树上维护等差数列类似的做法实现，具体是在平衡树中每个节点 node 维护（设该节点对应的子树表示的区间为[L, R]）：

$$cnt(node)=\sum_{i=L}^{R-1}\begin{cases}1 & if\ h_i>h_{i+1}\\0 & if\ h_i\le h_{i+1}\end{cases}$$

$$ans(node)=\sum_{i=L}^{R-1}\begin{cases}R-i & if\ h_i>h_{i+1}\\0 & if\ h_i\le h_{i+1}\end{cases}$$

可以看到$cnt(node)$即为满足$h_i>h_{i+1}$的个数，而$ans(node)$与题目所要求的答案的形式是相同的。现在我们通过这些值考虑节点 node（区间[L, R]）对询问[l, r]的贡献（由平衡树的作用，必然有整个子树都在询问区间内，另我们先考虑左边的式子，故有$l\le L\le R\le x\le r$）：

$$\sum_{i=L}^{R}\begin{cases}x-i & if\ h_i>h_{i+1}\\0 & if\ h_i\le h_{i+1}\end{cases}$$

$$=\sum_{i=L}^{R}\begin{cases}x-R+R-i & if\ h_i>h_{i+1}\\0 & if\ h_i\le h_{i+1}\end{cases}$$

$$=\sum_{i=L}^{R}\begin{cases}x-R & if\ h_i>h_{i+1}\\0 & if\ h_i\le h_{i+1}\end{cases}+\sum_{i=L}^{R}\begin{cases}R-i & if\ h_i>h_{i+1}\\0 & if\ h_i\le h_{i+1}\end{cases}$$

$$=(x-R)*\sum_{i=L}^{R-1}\begin{cases}1 & if\ h_i>h_{i+1}\\0 & if\ h_i\le h_{i+1}\end{cases}+\ ans(node)$$

$$=(x-R)*cnt(node)\ +\ ans(node)$$

由$cnt(node)$和$ans(node)$即可计算出区间[L, R]对答案的贡献了，同样的计算过程可以使用在平衡树中$ans(node)$的维护上。

由于右侧式子是对称的，所以只需改为在平衡树中维护$lans(node), rans(node)$、$lcnt(node), rcnt(node)$即可。

2. 有插入(I)操作、翻转(R)操作和询问(Q)

翻转操作可以使用 Splay 或函数式 Treap 等可以提取区间的平衡树来实现，每次翻转操作时将对应区间旋转成一棵子树，然后在该子树的根上打上翻转标记，并维护$lans(node), rans(node)$、$lcnt(node), rcnt(node)$的值。

【参考程序】

```
#include<cmath>
#include<vector>
#include<cstdio>
#include<cstring>
#include<algorithm>
```

```
#define rep(i,n) for (int i=0;i<n;i++)
#define For(i,n) for (int i=1;i<=n;i++)
#define FOR(i,a,b) for (int i=a;i<=b;i++)
#define FORD(i,a,b) for (int i=a;i>=b;i--)
#define MOD(x) if (x >= mo) x %= mo;
#define DELMOD(x) if (x >= mo) x -= mo;

using namespace std;
typedef long long LL;

const int mo = 1000000007;
const int TREE = 100000;
const int SZ = 50000;

int sz, root, now;
int fa[SZ], c[SZ][2];
short h[SZ], lmost[SZ], rmost[SZ], high[SZ];
LL s[SZ], cnt[SZ], lans[SZ], rans[SZ], lcnt[SZ], rcnt[SZ];
bool rev[SZ];

inline void Update(int x)
{
    int &l = c[x][0], &r = c[x][1];
    s[x] = s[l] + s[r] + 1;
    lmost[x] = rmost[x] = high[x] = h[x];
    cnt[x] = 1;
    lans[x] = rans[x] = lcnt[x] = rcnt[x] = 0;
    if (l)
    {
        lmost[x] = lmost[l];
        if (high[l] > high[x])
            high[x] = high[l], cnt[x] = cnt[l];
        else if (high[l] == high[x])
            cnt[x] += cnt[l];

        lans[x] = (lans[x] + lans[l] + (s[r] + 1) % mo * (lcnt[l] % mo)
) % mo;
        rans[x] += rans[l];
        DELMOD(rans[x]);

        lcnt[x] += lcnt[l];
        rcnt[x] += rcnt[l];

        if (rmost[l] > h[x])
```

```
        {
            lans[x] += s[r] + 1;
            MOD(lans[x]);
            lcnt[x]++;
        }

        else if (rmost[l] < h[x])
        {
            rans[x] += s[l];
            MOD(rans[x]);
            rcnt[x]++;
        }
    }

    if (r)
    {
        rmost[x] = rmost[r];
        if (high[r] > high[x])
            high[x] = high[r], cnt[x] = cnt[r];
        else if (high[r] == high[x])
            cnt[x] += cnt[r];

        lans[x] += lans[r];
        DELMOD(lans[x]);
        rans[x] = (rans[x] + rans[r] + rcnt[r] % mo * ((s[l] + 1) % mo)
) % mo;

        lcnt[x] += lcnt[r];
        rcnt[x] += rcnt[r];

        if (lmost[r] > h[x])
        {
            rans[x] += s[l] + 1;
            MOD(rans[x]);
            rcnt[x]++;
        }

        else if (lmost[r] < h[x])
        {
            lans[x] += s[r];
            MOD(lans[x]);
            lcnt[x]++;
        }
    }
```

```
}

inline void Reverse(int x)
{
    rev[x] = !rev[x];
    swap(c[x][0], c[x][1]);
    swap(lmost[x], rmost[x]);
    swap(lcnt[x], rcnt[x]);
    swap(lans[x], rans[x]);
}

inline void Down(int x)
{
    if (!rev[x]) return;
    rev[x] = false;
    if (c[x][0])
        Reverse(c[x][0]);
    if (c[x][1])
        Reverse(c[x][1]);
}

int FindMax(int x, int m)
{
    Down(x);
    int &l = c[x][0], &r = c[x][1];
    if (high[l] == high[now] && cnt[l] >= m)
        return FindMax(l, m);
    if (high[l] == high[now])
    {
        if (cnt[l] >= m)
            return FindMax(l, m);
        m -= cnt[l];
    }
    if (h[x] == high[now])
        m--;
    if (!m)
        return x;
    return FindMax(r, m);
}

inline int kth(int rk, int node = -1)
{
    int x;
    if (node == -1)
```

```
        x = root;
    else
        x = node;
    while (1) {
        Down(x);
        if (s[c[x][0]]+1==rk) return x;
        if (s[c[x][0]]+1>rk) x=c[x][0];
        else{
            rk-=s[c[x][0]]+1;
            x=c[x][1];
        }
    }
};

inline void Rotate(int &root,int x)
{
    int y,z,p,q;
    y=fa[x];z=fa[y];
    if (c[y][0]==x) p=0; else p=1;
    q=p^1;
    if (y==root) root=x;
    else if (c[z][0]==y) c[z][0]=x;
                else c[z][1]=x;
    fa[x]=z;fa[y]=x;fa[c[x][q]]=y;
    c[y][p]=c[x][q];c[x][q]=y;
    Update(y);Update(x);
}

inline void Splay(int &root,int x)
{
    int y,z;
    while (x!=root){
        y=fa[x];z=fa[y];
        if (y!=root)
          if ((c[y][0]==x)^(c[z][0]==y)) Rotate(root,x);
                                     else Rotate(root,y);
        Rotate(root,x);
    }
}

int main()
{
    freopen("train.in", "r", stdin);
    freopen("train.out", "w", stdout);
```

```
    int hh, l, r, x, pos, u, v, q;
    char cmd;
    sz = 2;
    c[1][1] = 2; fa[2] = 1;
    s[1] = 2; s[2] = 1;
    root = 1;
    scanf("%d", &q);
    rep(qq, q)
    {
        scanf(" %c", &cmd);
        switch(cmd)
        {
        case 'I':
            scanf("%d%d", &hh, &x);
            u = kth(x+1);
            Splay(root,u);
            sz++; h[sz] = hh;
            fa[sz] = u; fa[c[u][1]] = sz;
            c[sz][1] = c[u][1]; c[u][1] = sz;
            Update(sz); Update(u);
            break;
        case 'Q':
            scanf("%d%d", &l, &r);
            u=kth(l);v=kth(r+2);
            Splay(root,u);Splay(c[u][1],v);
            now = c[v][0];
            u = FindMax(now, (cnt[now] + 1) / 2);
            Splay(c[v][0], u);
            l = c[u][0], r = c[u][1];
            printf("%I64d\n", (lans[l] + lcnt[l] + rans[r] + rcnt[r])%m
o);
            break;
        case 'R':
            scanf("%d%d", &l, &r);
            u=kth(l);v=kth(r+2);
            Splay(root,u);Splay(c[u][1],v);
            Reverse(c[v][0]);
            break;
        }
    }
    return 0;
}
```

机器人问题（fittest，2s，512MB）

【算法分析】

算法一：

先不考虑提前破坏的情况，最优策略一定是按一定顺序逐个将敌方的机器人破坏，记每个机器人需要打的次数 $P_i = \left\lceil \frac{D_i}{ATK} \right\rceil$。

考虑两个敌方的机器人 i, j，若他们在决策中相邻，考虑将 i 放前面和 j 放前面的代价。

若将 i 放在 j 前面，则代价为 $(P_i - 1) \times (A_i + A_j) + P_j \times A_j$。

若将 j 放在 i 前面，则代价为 $(P_j - 1) \times (A_i + A_j) + P_i \times A_i$。

将 i 放前面更优当且仅当：

$$(P_i - 1) \times (A_i + A_j) + P_j \times A_j < (P_j - 1) \times (A_i + A_j) + P_i \times A_i$$

即：

$$P_i A_j < P_j A_i$$

即：

$$\frac{P_i}{A_i} < \frac{P_j}{A_j}$$

所以我们按 $\frac{P_i}{A_i}$ 为关键字从小到大排序，然后枚举提前破坏的两个机器人计算答案。

时间复杂度为 $O(n^2)$，期望得分 20 分。

算法二：

仍然将敌方的机器人按上面的关键字排序，现在我们假设提前破坏了两个机器人 $i, j(i < j)$，考虑减少的代价。

先考虑提前破坏一个机器人的收益，可以表示为：

$$S_i = A_i \times \left[\left(\sum_{j=1}^{i} P_j \right) - 1 \right] + P_i \times \sum_{j=i+1}^{n} A_j$$

那提前破坏两个机器人的收益就是 $S_i + S_j - P_i \times A_j (i < j)$，其中 $P_i \times A_j$ 是重复部分。

于是我们枚举 i，令收益 $f(i) = S_i + S_j - P_i \times A_j (j > i)$，将式子转化为：

$$\boldsymbol{S_j} = P_i \times \boldsymbol{A_j} - S_i + f(i)$$

这是一个显然的斜率优化模型，将每个 j 映射为平面上的点 (A_j, S_j)，问题就是求一个斜率为 P_i，并且过一个点 (A_j, S_j) 的直线的最大截距，对所有 j 维护一个斜率递减的上凸壳即可。

从大到小枚举 i，由于斜率和决策点的横坐标都没有单调性，所以要用 $Splay$ 维护凸壳，并在凸壳上二分斜率求得最优答案。时间复杂度 $O(n \log n)$。

【参考程序】

```
#include <cstdio>
#include <algorithm>
typedef long long  ll;
const  int maxn = 330000;
int ld[maxn], ra[maxn];
ll  s[maxn];
```

```
ll  ans, res;
int n, ATK;
struct  qs
{
    int a, p;
}   q[maxn];
bool    operator<(qs x, qs y) {return x.p * y.a < x.a * y.p;}
inline  void    renew(ll &x, ll y) {if (x < y) x = y;}
struct  cljsplay
{
    struct  node
    {
        ll  x, y, lx, ly;
        node    *p, *s[2];
        node() {p = s[0] = s[1] = 0;}
        bool    getlr() {return p->s[1] == this;}
        node    *link(int w, node *p) {s[w] = p; if (p) p->p = this; return
this;}
    }   *root;
    void    rot(node *p)
    {
        node *q = p->p->p;
        p->getlr()  ?  p->link(0,  p->p->link(1,  p->s[0]))  :  p->link(1,
p->p->link(0, p->s[1]));
        if (q) q->link(q->s[1] == p->p, p); else (root = p)->p = 0;
    }
    void    splay(node *p)//将结点 p 旋到根
    {
        while (p->p && p->p->p)
            p->getlr() == p->p->getlr() ? (rot(p->p), rot(p)) : (rot(p),
rot(p));
        if (p->p) rot(p);
    }
    void    splay(node *p, node *tar)//将结点 p 旋到 tar 的下面
    {
        while (p->p != tar && p->p->p != tar)
            p->getlr() == p->p->getlr() ? (rot(p->p), rot(p)) : (rot(p),
rot(p));
        if (p->p != tar) rot(p);
    }
    void    preset()
    {
        root = new node; root->x = 0; root->y = 0; root->lx = 0; root->ly
= -1;
```

```
    }
    node*   prev(node *p)
    {
        if (!p) return 0;
        splay(p);
        if (!p->s[0]) return 0;
        node *q = p->s[0];
        for (; q->s[1]; q=q->s[1]);
        splay(q);
        return q;
    }
    node*   succ(node *p)
    {
        if (!p) return 0;
        splay(p);
        if (!p->s[1]) return 0;
        node *q = p->s[1];
        for (; q->s[0]; q=q->s[0]);
        splay(q);
        return q;
    }
    void   insert(ll x, ll y)//插入一个点(x,y)
    {
        node *p = root, *p1, *p2;
        while (p)
        {
            p1 = p;
            p = p->s[p->x < x];
        }
        p = new node; p->x = x; p->y = y;
        p1->link(p1->x < x, p);
        splay(p);
        if ((p1 = prev(p)) && (p2 = succ(p)))
        {
            if ((p2->x - p->x) * (p1->y - p->y) >= (p2->y - p->y) * (p1->x
- p->x))
            {
                splay(p1);
                splay(p2, p1);
                p2->s[0] = 0;
                return;
            }
        }
        while (p2 = prev(p1 = prev(p)))
```

```
        {
            if ((p->x - p1->x) * (p2->y - p1->y) >= (p->y - p1->y) * (p2->x
- p1->x))
            {
                splay(p2);
                splay(p, p2);
                p->s[0] = 0;
            }   else    break;
        }
        if (p1 = prev(p)) p->lx = p1->x, p->ly = p1->y;
        while (p2 = succ(p1 = succ(p)))
        {
            if ((p->x - p1->x) * (p2->y - p1->y) >= (p->y - p1->y) * (p2->x
- p1->x))
            {
                splay(p2);
                splay(p, p2);
                p->s[0] = 0;
            }   else    break;
        }
        if (p1 = succ(p)) p1->lx = p->x, p1->ly = p->y;
    }
    ll  get(int k) //二分斜率
    {
        ll  res = 0;
        node *p = root;
        while (p)
        {
            renew(res, p->y - k * p->x);
            p = p->s[p->y - k * p->x >= p->ly - k * p->lx];
        }
        return res;
    }
}tool1;
int main()
{
    freopen("fittest.in", "r", stdin);
    freopen("fittest.out", "w", stdout);
    scanf("%d%d", &n, &ATK);
    for (int i=1; i<=n; i++)
    {
        scanf("%d%d", &q[i].a, &q[i].p);
        q[i].p = (q[i].p + ATK - 1) / ATK;
    }
```

```
    std::sort(q + 1, q + n + 1);//按 q[i].p/q[i].a 排序
    ld[0] = 0;
    for (int i=1; i<=n; i++) ld[i] = ld[i-1] + q[i].p;
    ra[n+1] = 0;
    for (int i=n; i; i--) ra[i] = ra[i+1] + q[i].a;
    for (int i=1; i<=n; i++)
        s[i] = (ld[i] - 1) * (ll)q[i].a + ra[i+1] * (ll)q[i].p;
    ans = 0;
    for (int i=1; i<=n; i++) ans += (ld[i] - 1) * (ll)q[i].a;
    res = 0;
    tool1.preset();
    for (int i=n-1; i; i--)
    {
        tool1.insert(q[i+1].a, s[i+1]);//插入决策点
        renew(res, s[i] + tool1.get(q[i].p));//在凸壳上二分
    }
    printf("%I64d\n", ans - res);
    return 0;
}
```