小米 Online Judge 19年2月常规赛题解

本题解仅供大家参考，欢迎大佬们提供更为巧妙的方法~

小爱密码

题解

本题的递推式是乘积形式， 而n又很大, 无法用矩阵快速幂优化。注意到 $F(n)$ 的结果一定是 $A^{k_1} \times B^{k_2} \times C^{k_3 \times D}$ 的形式，于是我们可以使用矩阵快速幂分开加速递推 A, B, C 三个的指数。因为 $G(n)$ 是前缀积，所以相当于求 n 个 $F(x)$ 的对应指数之和，在递推矩阵中加一维记录前缀和即可。单次计算复杂度为 $\mathrm{O}(\log n)$ 。

标程

```
#include<stdio.h>
#include<string.h>
typedef long long ll;
struct node {
    ll a[4][4];
}orgin,res;
ll p,mod;
node mult(node x,node y) {
    node tmp;
    memset(tmp.a,0,sizeof(tmp.a));
    for(int i=0;i<4;i++)
        for(int j=0;j<4;j++)
            for(int k=0;k<4;k++){
                tmp.a[i][j] += x.a[i][k] * y.a[k][j] % p;
                tmp.a[i][j] = (tmp.a[i][j] + p) % p + p;
            }
    return tmp;
}
void ksm(ll x) {
    memset(res.a,0,sizeof(res.a));
    memset(orgin.a,0,sizeof(orgin.a));
    orgin.a[0][0] = orgin.a[0][1] = orgin.a[0][2] = orgin.a[0][3] = orgin.a[1][1] =
orgin.a[1][2] = orgin.a[1][3] = orgin.a[2][2] = orgin.a[2][3] = orgin.a[3][2] = res.a[0][0]
= res.a[1][1] = res.a[2][2] = res.a[3][3] = 1;
    while(x) {
        if(x&1) res = mult(res,orgin);
        x >>= 1;
        orgin = mult(orgin,orgin);
    }
}
ll Qmult(ll x,ll y) {
    ll ans = 1;
```

```
    while(y) {
        if(y&1) ans = ans * x % mod;
        y >>= 1;
        x = x * x % mod;
    }
    return ans;
}
ll phi(ll n) {
    ll res = n;
    for(ll i=2;i*i<=n;i++)
        if(n%i==0) {
            res -= res/i;
            while(n%i==0) n/=i;
        }
    if(n>1) res -= res/n;
    return res;
}
int check1(ll n,ll x) {
    ll A = 1,B = 1;
    for(int i=1;i<n;i++) {
        ll C = A + B;
        A = B;
        B = C;
        if(C >= x) return 1;
    }
    return 0;
}
int check2(ll n,ll x) {
    ll A = 1,B = 1,sum = 2;
    for(int i=1;i<n;i++) {
        ll C = A + B;
        sum += C;
        A = B;
        B = C;
        if(sum >= x) return 1;
    }
    return 0;
}
int check3(ll n,ll x) {
    ll A = 1,B = 1,sum = 2,G = 3;
    for(int i=1;i<n;i++) {
        ll C = A + B;
        sum += C;
        G += sum;
        A = B;
        B = C;
        if(G >= x) return 1;
    }
    return 0;
}
int main(){
    ll A,B,C,D,n;
    scanf("%lld%lld%lld%lld%lld%lld",&A,&B,&C,&D,&mod,&n);
```

```
    A%=mod,B%=mod,C%=mod;
    ll k = Qmult(C,D)%mod;
    if(n == 1) {
        printf("%09lld\n",A);
        return 0;
    }
    if(n == 2) {
        printf("%09lld\n",A*B%mod);
        return 0;
    }
    if(n == 3) {
        printf("%09lld\n",A*A%mod*B%mod*B%mod*k%mod);
        return 0;
    }
    p = phi(mod);
    ksm(n-2);
    ll fn = (res.a[2][2] + res.a[2][3]);
    ksm(n-3);
    ll sn = (res.a[1][1]*2 + res.a[1][2] + res.a[1][3]);
    ksm(n-4);
    ll gn = (res.a[0][0]*3 + res.a[0][1]*2 + res.a[0][2] + res.a[0][3]);
    ll ans = 1;
    if(check1(n-2,fn)) ans = ans%mod * Qmult(A,fn%p+p) % mod;
    else ans = ans%mod * Qmult(A,fn) % mod;
    if(check2(n-3,sn)) ans = ans%mod * Qmult(B,sn%p+p) % mod;
    else ans = ans%mod * Qmult(B,sn) % mod;
    if(check3(n-4,gn)) ans = ans%mod * Qmult(k,gn%p+p) % mod;
    else ans = ans%mod * Qmult(k,gn) % mod;
    printf("%09lld\n",ans);
    return 0;
}
```

> 本题由志愿者 Archangel 提供

Carryon 数数字

题解

对于一个十六进制来说，可以表示为$x=\sum_{i=0}^{k} a_i * 16^i$。对于每一个 $a_i * 16^i = a_i * (15+1)^i = a_i * \sum_{j=0}^{i} \binom{i}{j} * 15^j = a_i \pmod{15}$。

对于$l=r$的情况，$ans = l = \sum_{i=0}^{k} a_i \pmod{15}$，

对于$l+1=r$的情况，我们可以看做将$l \pmod{15}$连在$l+1$的前面，那么这个的答案显然是 $ans = l + \sum_{i=0}^{k} a_i = l + l + 1 = l + r \pmod{15}$，

那么数学归纳法可得，$ans = \sum_{i=l}^{r} i \pmod{15}$。因为要对15取模，对于数列$i \pmod{15}$会有 $\sum_{i=k}^{k+14} i = 0 \pmod{15}$。所以在求ans的时候，从$l$开始取连续(r-l+1)%15个数计算ans即可。

PS: $\binom{i}{j}$为组合数i取j，即C_i^j。

标程

```
#include <iostream>
#define ll long long int
using namespace std;
ll n, m;
int main(){
    while( cin >> n >> m ){
        ll ans = (m - n + 1) % 15;
        ll sum = 0;
        while(ans --){
            sum += n%15;
            n++;
        }
        sum %= 15;
        cout << sum << endl;
    }
    return 0;
}
```

> 本题由志愿者 carryon 提供

Logic Gatekeeper

题解一：cdp分治

显然用二维树状数组空间上是吃不消的。

如果把操作按点拆成4个，那么由于其中有两对点的x坐标相等，所以最终总会被放在一起，且操作有一部分相互抵消，那么不妨合并他们。把操作拆成两个，即询问(0,0)到(x,y)的，拆成修改左方的子矩阵。但是为了方便，这里需要把坐标都偏移一格。

我们考虑拆分区间，考虑跨立区间的贡献。

先对x轴进行排序，然后cdq的时候对时间进行排序。

按照x轴划分区间之后，左边的修改对右边的询问有影响，右边的修改对左边的询问也有贡献。

由于线段树的常数比较大，而且操作的次数很多，我们用树状数组维护y轴上的贡献，左边的数组修改的时候，应该改val*x即可，因为右边的询问x必定比左边大。而右边的改则只需要改val，询问时乘x，因为右边x比较大。

时间复杂度$O(q\log q\log m)$。

标程一

```
#include <algorithm>
#include <cstdio>
#define rg register
```

```
#define lowbit(x) ((x)&(-(x)))
using namespace std;
typedef long long ll;
const int maxn=1000010,maxm=100010,mod=998244353;
template <typename Tp> inline void getmin(Tp &x,Tp y){if(y<x) x=y;}
template <typename Tp> inline void getmax(Tp &x,Tp y){if(y>x) x=y;}
template <typename Tp> inline void read(Tp &x)
{
    x=0;int f=0;char ch=getchar();
    while(ch!='-'&&(ch<'0'||ch>'9')) ch=getchar();
    if(ch=='-') f=1,ch=getchar();
    while(ch>='0'&&ch<='9') x=x*10+ch-'0',ch=getchar();
    if(f) x=-x;
}
struct data{
    int tp,t,x,y,id,y2;
    bool operator < (const data &p){return x<p.x;}
}q[maxm<<2],tmp[maxm<<2];
int n,m,k,tot,qc,top,ans[maxm],res[maxm],id[maxm<<2];
inline int pls(int x,int y){return x+y>=mod?x+y-mod:x+y;}
inline int dec(int x,int y){return x-y<0?x-y+mod:x-y;}
struct BIT{
    int a[maxn],a2[maxn];
    void add(int p,int v)
    {
        for(int i=p;i<=m;i+=lowbit(i))
          a[i]=pls(a[i],v),a2[i]=pls(a2[i],(ll)p*v%mod);
    }
    int query(int p)
    {
        int res=0;
        for(int i=p;i;i-=lowbit(i)) res=pls(res,dec((ll)(p+1)*a[i]%mod,a2[i]));
        return res;
    }
}a,b;
int power(int x,int y)
{
    int res=1;
    for(;y;y>>=1,x=(ll)x*x%mod)
      if(y&1)
        res=(ll)res*x%mod;
    return res;
}
void input()
{
    int op,x,y,s,t,val;
    read(n);read(m);read(k);m++;tot=k<<1;
    for(rg int i=1;i<=k;i++)
    {
        read(op);read(x);read(y);read(s);read(t);
        x++;y++;s++;t++;
        if(op==1)
        {
```

```
            read(val);
            q[(i<<1)-1]=(data){0,i,s,t,val,y};
            q[i<<1]=(data){0,i,x-1,t,mod-val,y};
        }
        else
        {
            q[(i<<1)-1]=(data){1,i,s,t,++qc,y};
            q[i<<1]=(data){2,i,x-1,t,qc,y};
            res[qc]=power((ll)(s-x+1)*(t-y+1)%mod,mod-2);
        }
    }
    sort(q+1,q+tot+1);
}
void cdq(const int &l,const int &r)
{
    int mid=(l+r)>>1;
    if(l<mid) cdq(l,mid);
    if(mid+1<r) cdq(mid+1,r);
    rg int i=l,j=mid+1;top=0;
    while(i<=mid&&j<=r)
    {
        if(q[i].t<=q[j].t) tmp[++top]=q[i++],id[top]=0;
        else tmp[++top]=q[j++],id[top]=1;
    }
    while(i<=mid) tmp[++top]=q[i++],id[top]=0;
    while(j<=r) tmp[++top]=q[j++],id[top]=1;
    for(i=1;i<=top;i++)
    {
        if(id[i])//r
        {
            if(tmp[i].tp==1)
              ans[tmp[i].id]=pls(ans[tmp[i].id],dec(a.query(tmp[i].y),a.query(tmp[i].y2-
1)));
            else if(tmp[i].tp==2)
              ans[tmp[i].id]=dec(ans[tmp[i].id],dec(a.query(tmp[i].y),a.query(tmp[i].y2-
1)));
            else b.add(tmp[i].y2,tmp[i].id),b.add(tmp[i].y+1,mod-tmp[i].id);
        }
        else
        {
            if(tmp[i].tp==1)
              ans[tmp[i].id]=pls(ans[tmp[i].id],
(ll)tmp[i].x*dec(b.query(tmp[i].y),b.query(tmp[i].y2-1))%mod);
            else if(tmp[i].tp==2)
              ans[tmp[i].id]=dec(ans[tmp[i].id],
(ll)tmp[i].x*dec(b.query(tmp[i].y),b.query(tmp[i].y2-1))%mod);
            else a.add(tmp[i].y2,(ll)tmp[i].id*tmp[i].x%mod),a.add(tmp[i].y+1,(ll)(mod-
tmp[i].id)*tmp[i].x%mod);
        }
    }
    for(i=1;i<=top;i++)
    {
        if(!tmp[i].tp)
```

```
        {
            if(id[i]) b.add(tmp[i].y2,mod-tmp[i].id),b.add(tmp[i].y+1,tmp[i].id);
            else a.add(tmp[i].y2,mod-(ll)tmp[i].id*tmp[i].x%mod),a.add(tmp[i].y+1,
(ll)tmp[i].id*tmp[i].x%mod);
        }
        q[l+i-1]=tmp[i];
    }
}
int main()
{
    input();
    cdq(1,tot);
    for(rg int i=1;i<=qc;i++) printf("%lld\n",(ll)ans[i]*res[i]%mod);
    return 0;
}
```

题解二：KDtree

令A为二维差分数组，即$A_{i,j}$表示(i,j)到(n,m)这个区域加上的值。

若是询问$(1,1)$到x,y的矩阵的权值和，有：

$$\sum_{i=1}^{x}\sum_{j=1}^{y}A_{i,j}\times(x-i+1)\times(y-j+1)$$
$$=\sum_{i=1}^{x}\sum_{j=1}^{y}A_{i,j}\times(xy+x+y+1)-A_{i,j}\times i(y+1)-A_{i,j}\times j(x+1)+A_{i,j}\times ij$$

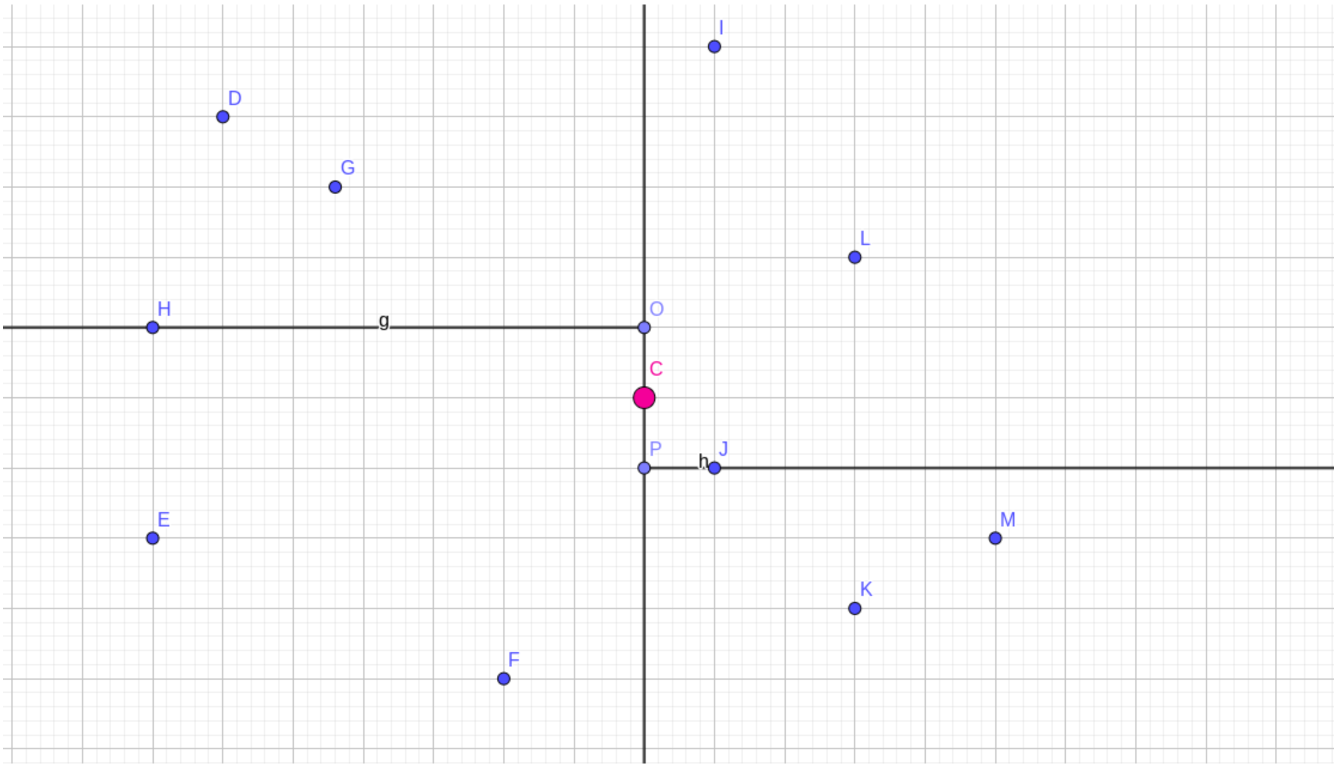
因此维护$A_{i,j},A_{i,j}\times i,A_{i,j}\times j,A_{i,j}\times ij$即可

搞4棵kd-tree分别维护

单次询问二维前缀和的复杂度是$O(log_{\frac{4}{3}}n)$的

怎么证明复杂度?

对于kd-tree的某个中心点（图中红色的那个点）：



它和它左右俩儿子把平面分成了4个区域，每个区域的点数都是一样的。

- 如果询问（的右下角）落在了左上方的区域内，那么其他三个区域都可以舍去
- 如果询问落在了左下方，那么右边的两个区域都能舍去
- 如果询问落在了右上方，那么至少右下方的区域可以舍去
- 如果询问落在了右下方，那么左上方的整个区域都包含在了询问内，可以直接返回。

（这是左边的线高一些的情况，右边的线高一些的情况也是类似的）

这样至少每次能去掉$\frac{1}{4}$的点，留下$\frac{3}{4}$的点。

这样子一次的复杂度就是$O(log_{\frac{4}{3}}n)$的了

总复杂度是$O(qlog_{\frac{4}{3}}q)$的，理论复杂度比$O(qlog_2^2q)$的要低

燃鹅。。。

如果你不做任何优化，在随机数据下你会发现你要跑大概30秒。。。

因为你不仅带了kd-tree的巨大常数，还带了整整一个16的大常数。。。

因此要卡常优化。。。

在这里提供这么几种优化方式：

- 保持树的平衡，类似替罪羊树的思想不断重构保证平衡，大概可以快一倍（如果数据不随机那么这一条是必须的）
- 还有一种更好的平衡方法，就是把操作离线，先把所有点都记录下来，最后一起建一颗kd-tree，后面的插入操作改为修改操作，比上一种平衡方式快一倍
- 还有最关键的，四颗kd-tree的结构是完全相同的，只有值不同。我们没必要每次都重新寻找点的位置，我们可以把四颗树合并成一颗，每个节点开4个值域，每次查询返回4个值。。。这样可以弄到一个$\frac{1}{4}$的常数！！！

然后还有`register`啊`inline`啊共用全局变量啊引用啊等等等等卡常。。。

标程二

```
#include <bits/stdc++.h>

#define NS (100005)
#define MOD (998244353)
#define REG register

using namespace std;

template <typename _Tp> inline void IN(_Tp& dig)
{
    REG char c; REG bool flag = 0; dig = 0;
    while (c = getchar(), !isdigit(c)) if (c == '-') flag = 1;
    while (isdigit(c)) dig = dig * 10 + c - '0', c = getchar();
    if (flag) dig = -dig;
}

inline int pls(REG int a, REG int b)
{
    return a + b >= MOD ? a + b - MOD : a + b;
}
inline int mns(REG int a, REG int b) {return a - b < 0 ? a - b + MOD : a - b;}
inline int mul(REG int a, REG int b) {return 1ll * a * b % MOD;}

bool D;

struct N
{
    int p[2], d[4], sum[4], mn[2], mx[2], s[2];
    inline bool operator < (const N oth) const
    {
        if (p[D] == oth.p[D]) return p[D ^ 1] < oth.p[D ^ 1];
        return p[D] < oth.p[D];
    }
} arr[NS << 2];

struct Q {int x, y, d[4];};

struct kd_tree
{
    int root, sz; N e[NS << 2]; Q o;
    kd_tree()
    {
        root = sz = 0;
        e[0].mn[0] = e[0].mn[1] = INT_MAX;
        e[0].mx[0] = e[0].mx[1] = INT_MIN;
    }
    inline void pup(REG int a)
    {
        for (REG int i = 0; i < 4; i += 1)
            e[a].sum[i] =
                pls(e[a].d[i], pls(e[e[a].s[0]].sum[i], e[e[a].s[1]].sum[i]));
```

```
    for (REG int i = 0; i < 2; i += 1)
    {
        e[a].mn[i] = min(e[a].p[i], min(e[e[a].s[0]].mn[i], e[e[a].s[1]].mn[i]));
        e[a].mx[i] = max(e[a].p[i], max(e[e[a].s[0]].mx[i], e[e[a].s[1]].mx[i]));
    }
}
inline void pup2(REG int a)
{
    for (REG int i = 0; i < 4; i += 1)
        e[a].sum[i] =
            pls(e[a].d[i], pls(e[e[a].s[0]].sum[i], e[e[a].s[1]].sum[i]));
}
int build(REG int l, REG int r, REG bool dv)
{
    if (l > r) return 0;
    D = dv;
    REG int a = ++sz, mid = (l + r) >> 1;
    nth_element(arr + l, arr + mid, arr + r + 1);
    e[a] = arr[mid];
    e[a].s[0] = build(l, mid - 1, dv ^ 1);
    e[a].s[1] = build(mid + 1, r, dv ^ 1);
    pup(a); return a;
}
void modify(REG int a, REG bool dv)
{
    if (e[a].p[0] == o.x && e[a].p[1] == o.y)
    {
        for (REG int i = 0; i < 4; i += 1) e[a].d[i] = pls(e[a].d[i], o.d[i]);
        pup2(a); return;
    }
    if (!dv)
    {
        if (e[a].p[0] == o.x)
        {
            if (o.y < e[a].p[1]) modify(e[a].s[0], dv ^ 1);
            else modify(e[a].s[1], dv ^ 1);
        }
        else
        {
            if (o.x < e[a].p[0]) modify(e[a].s[0], dv ^ 1);
            else modify(e[a].s[1], dv ^ 1);
        }
    }
    else
    {
        if (e[a].p[1] == o.y)
        {
            if (o.x < e[a].p[0]) modify(e[a].s[0], dv ^ 1);
            else modify(e[a].s[1], dv ^ 1);
        }
        else
        {
            if (o.y < e[a].p[1]) modify(e[a].s[0], dv ^ 1);
```

```
                else modify(e[a].s[1], dv ^ 1);
            }
        }
        pup2(a);
    }
    inline void modify() {modify(root, 0);}
    void query(REG int a)
    {
        if (!a || e[a].mn[0] > o.x || e[a].mn[1] > o.y) return;
        if (e[a].mx[0] <= o.x && e[a].mx[1] <= o.y)
        {
            for (REG int i = 0; i < 4; i += 1) o.d[i] = pls(o.d[i], e[a].sum[i]);
            return;
        }
        if (e[a].p[0] <= o.x && e[a].p[1] <= o.y)
            for (REG int i = 0; i < 4; i += 1) o.d[i] = pls(o.d[i], e[a].d[i]);
        query(e[a].s[0]), query(e[a].s[1]);
    }
    inline void query() {query(root);}
} kdt;

inline int qpow(REG int a, REG int b)
{
    REG int res = 1; a %= MOD;
    while (b)
    {
        if (b & 1) res = mul(res, a);
        a = mul(a, a), b >>= 1;
    }
    return res % MOD;
}

inline int Inv(REG int a) {return qpow(a, MOD - 2);}

int n, m, q, o[NS], a[NS], b[NS], c[NS], d[NS], k[NS];;

int main(int argc, char const* argv[])
{
    IN(n), IN(m), IN(q); REG int cnt = 0;
    for (REG int i = 1; i <= q; i += 1)
    {
        IN(o[i]), IN(a[i]), IN(b[i]), IN(c[i]), IN(d[i]);
        if (o[i] == 1)
        {
            IN(k[i]);
            arr[++cnt].p[0] = a[i], arr[cnt].p[1] = b[i];
            arr[++cnt].p[0] = c[i] + 1, arr[cnt].p[1] = d[i] + 1;
            arr[++cnt].p[0] = a[i], arr[cnt].p[1] = d[i] + 1;
            arr[++cnt].p[0] = c[i] + 1, arr[cnt].p[1] = b[i];
        }
    }
    sort(arr + 1, arr + 1 + cnt); REG int tmp = cnt; cnt = 0;
    for (REG int i = 1; i <= tmp; i += 1)
```

```
        if (arr[i].p[0] != arr[i - 1].p[0] || arr[i].p[1] != arr[i - 1].p[1])
            arr[++cnt] = arr[i];
    kdt.root = kdt.build(1, cnt, 0);
    for (REG int i = 1, res; i <= q; i += 1)
        if (o[i] == 1)
        {
            kdt.o.x = a[i], kdt.o.y = b[i];
            kdt.o.d[0] = k[i], kdt.o.d[1] = mul(a[i], k[i]);
            kdt.o.d[2] = mul(b[i], k[i]), kdt.o.d[3] = mul(mul(a[i], b[i]), k[i]);
            kdt.modify();
            kdt.o.x = c[i] + 1, kdt.o.y = d[i] + 1;
            kdt.o.d[0] = k[i], kdt.o.d[1] = mul(c[i] + 1, k[i]);
            kdt.o.d[2] = mul(d[i] + 1, k[i]);
            kdt.o.d[3] = mul(mul(c[i] + 1, d[i] + 1), k[i]);
            kdt.modify();
            kdt.o.x = a[i], kdt.o.y = d[i] + 1;
            kdt.o.d[0] = mns(0, k[i]), kdt.o.d[1] = mns(0, mul(a[i], k[i]));
            kdt.o.d[2] = mns(0, mul(d[i] + 1, k[i]));
            kdt.o.d[3] = mns(0, mul(mul(a[i], d[i] + 1), k[i]));
            kdt.modify();
            kdt.o.x = c[i] + 1, kdt.o.y = b[i];
            kdt.o.d[0] = mns(0, k[i]), kdt.o.d[1] = mns(0, mul(c[i] + 1, k[i]));
            kdt.o.d[2] = mns(0, mul(b[i], k[i]));
            kdt.o.d[3] = mns(0, mul(mul(c[i] + 1, b[i]), k[i]));
            kdt.modify();
        }
        else
        {
            memset(kdt.o.d, 0, sizeof(kdt.o.d));
            kdt.o.x = c[i], kdt.o.y = d[i], kdt.query();
            res = pls(kdt.o.d[3],
                mul(kdt.o.d[0], pls(pls(mul(c[i], d[i]), c[i]), d[i] + 1)));
            res = mns(res, mul(kdt.o.d[1], d[i] + 1));
            res = mns(res, mul(kdt.o.d[2], c[i] + 1));
            memset(kdt.o.d, 0, sizeof(kdt.o.d));
            kdt.o.x = a[i] - 1, kdt.o.y = b[i] - 1, kdt.query();
            res = pls(res, kdt.o.d[3]);
            res = pls(res,
                mul(kdt.o.d[0], pls(pls(mul(a[i] - 1, b[i] - 1), a[i]), b[i] - 1)));
            res = mns(res, mul(kdt.o.d[1], b[i]));
            res = mns(res, mul(kdt.o.d[2], a[i]));
            memset(kdt.o.d, 0, sizeof(kdt.o.d));
            kdt.o.x = a[i] - 1, kdt.o.y = d[i], kdt.query();
            res = mns(res, kdt.o.d[3]);
            res = mns(res,
                mul(kdt.o.d[0], pls(pls(mul(a[i] - 1, d[i]), a[i]), d[i])));
            res = pls(res, mul(kdt.o.d[1], d[i] + 1));
            res = pls(res, mul(kdt.o.d[2], a[i]));
            memset(kdt.o.d, 0, sizeof(kdt.o.d));
            kdt.o.x = c[i], kdt.o.y = b[i] - 1, kdt.query();
            res = mns(res, kdt.o.d[3]);
            res = mns(res,
                mul(kdt.o.d[0], pls(pls(mul(c[i], b[i] - 1), c[i]), b[i])));
```

```
            res = pls(res, mul(kdt.o.d[1], b[i]));
            res = pls(res, mul(kdt.o.d[2], c[i] + 1));
            printf("%d\n", mul(res, Inv(mul(c[i] - a[i] + 1, d[i] - b[i] + 1))));
        }
    return 0;
}
```

友情链接：Logic Gatekeeper 歌曲的[网易云地址]

> 本题由志愿者 Remmina和他的小伙伴们 提供