

泡泡猿 ACM 模板

Rand0w & REXWIND & Dallby

2021 年 10 月 5 日



目录

1	头文件	1
1.1	头文件 (Rand0w)	1
1.2	头文件 (REXWind)	3
1.3	头文件 (Dallby)	3
2	数据结构	3
2.1	扫描线	3
3	数论	5
3.1	欧拉筛	5
3.2	Exgcd	5
3.3	ExCRT 扩展中国剩余定理	6
3.4	线性求逆元	6
3.5	多项式	6
3.5.1	FFT 快速傅里叶变换	6
3.5.2	NTT 快速数论变换	7
3.5.3	MTT 任意模数 FFT	7
3.6	组合数	9
3.7	矩阵快速幂	9
3.8	高斯消元	10
3.9	欧拉降幂	11
4	计算几何	11
4.1	三点求圆心	11
4.2	欧拉降幂	11
4.3	拉格朗日插值	11
5	数据结构	13
5.1	并查集系列	13
5.1.1	普通并查集	13
5.1.2	按秩合并并查集	13
5.1.3	可持久化并查集	14
5.1.4	ETT 维护动态图连通性	15
5.2	平衡树系列	15
5.2.1	fhq_treap	15
6	字符串	17
6.1	KMP	17
6.2	AC 自动机	17
6.3	FFT 解决字符串匹配问题	19
6.4	字符串哈希	20
6.5	后缀数组 SA+LCP	20
6.6	后缀自动机 SAM	21
7	其他	22
7.1	ST 表求 RMQ	22
7.2	莫队	23
7.3	带修莫队	24
8	STL 等小技巧	25
8.1	集合 set	25
8.2	快读快写 (短)	25
8.3	GCD(压行)	26
8.4	计时	26

1 头文件

1.1 头文件 (Rand0w)

```
1 #include <bits/stdc++.h>
2 // #include <bits/extc++.h>
3 // using namespace __gnu_pbds;
4 // using namespace __gnu_cxx;
5 using namespace std;
6 #pragma optimize(2)
7 // #pragma GCC optimize("Ofast,no-stack-protector")
8 // #pragma GCC target("sse,sse2,sse3,ssse3,sse4,popcnt,abm,mmx,avx,avx2,tune=native")
9 #define rbset(T) tree<T,null_type,less<T>,rb_tree_tag,tree_order_statistics_node_update>
10 const int inf = 0x7FFFFFFF;
11 typedef long long ll;
12 typedef double db;
13 typedef long double ld;
14 template<class T>inline void MAX(T &x,T y){if(y>x)x=y;}
15 template<class T>inline void MIN(T &x,T y){if(y<x)x=y;}
16 namespace FastIO
17 {
18 char buf[1 << 21], buf2[1 << 21], a[20], *p1 = buf, *p2 = buf, hh = '\n';
19 int p, p3 = -1;
20 void read() {}
21 void print() {}
22 inline int getc()
23 {
24 return p1 == p2 && (p2 = (p1 = buf) + fread(buf, 1, 1 << 21, stdin), p1 == p2) ? EOF : *p1++;
25 }
26 inline void flush()
27 {
28 fwrite(buf2, 1, p3 + 1, stdout), p3 = -1;
29 }
30 template <typename T, typename... T2>
31 inline void read(T &x, T2 &... oth)
32 {
33 int f = 0; x = 0; char ch = getc();
34 while (!isdigit(ch)){if (ch == '-')f = 1; ch = getc();}
35 while (isdigit(ch)){x = x * 10 + ch - 48; ch = getc();}
36 x = f ? -x : x; read(oth...);
37 }
38 template <typename T, typename... T2>
39 inline void print(T x, T2... oth)
40 {
41 if (p3 > 1 << 20) flush();
42 if (x < 0) buf2[++p3] = 45, x = -x;
43 do{a[++p] = x % 10 + 48;} while (x /= 10);
44 do{buf2[++p3] = a[p];} while (--p);
45 buf2[++p3] = hh;
46 print(oth...);
47 }
48 } // namespace FastIO
49 #define read FastIO::read
50 #define print FastIO::print
```

```
51 #define flush FastIO::flush
52 #define spt fixed<<setprecision
53 #define endl1 '\n'
54 #define mul(a,b,mod) (__int128)(a)*(b)%(mod)
55 #define pii(a,b) pair<a,b>
56 #define pow powmod
57 #define X first
58 #define Y second
59 #define lowbit(x) (x&-x)
60 #define MP make_pair
61 #define pb push_back
62 #define pt putchar
63 #define yx_queue priority_queue
64 #define lson(pos) (pos<<1)
65 #define rson(pos) (pos<<1|1)
66 #define y1 code_by_Rand0w
67 #define yn A_muban_for_ACM
68 #define j1 it_is_just_an_eastegg
69 #define lr hope_you_will_be_happy_to_see_this
70 #define int long long
71 #define rep(i, a, n) for (register int i = a; i <= n; ++i)
72 #define per(i, a, n) for (register int i = n; i >= a; --i)
73 const ll llinf = 4223372036854775851;
74 const ll mod = (0 ? 1000000007 : 998244353);
75 ll pow(ll a,ll b,ll md=mod) {ll res=1;a%=md; assert(b>=0); for(;b;b>>=1){if(b&1)res=mul(res,a,md);a=mul(a,a,md);}return res;}
76 const ll mod2 = 999998639;
77 const int m1 = 998244353;
78 const int m2 = 1000001011;
79 const int pr=233;
80 const double eps = 1e-7;
81 const int maxm= 1;
82 const int maxn = 510000;
83 void work()
84 {
85
86 }
87 signed main()
88 {
89     #ifndef ONLINE_JUDGE
90         //freopen("in.txt","r",stdin);
91         //freopen("out.txt","w",stdout);
92     #endif
93     //std::ios::sync_with_stdio(false);
94     //cin.tie(NULL);
95     int t = 1;
96     //cin>>t;
97     for(int i=1;i<=t;i++){
98         //cout<<"Case #"<<i<<": "<<endl1;
99         work();
100     }
101     return 0;
102 }
```

1.2 头文件 (REXWind)

```
1 #include<iostream>
2 #include<cstring>
3 #include<cstdio>
4 #include<algorithm>
5 #include<vector>
6 #include<map>
7 #include<queue>
8 #include<cmath>
9 using namespace std;
10
11 template<class T>inline void read(T &x){
12     x=0;char o,f=1;
13     while(o=getchar(),o<48)if(o==45)f=-f;
14     do x=(x<<3)+(x<<1)+(o^48);while(o=getchar(),o>47);x*=f;}
15 int cancel_sync=(ios::sync_with_stdio(0),cin.tie(0),0);
16 #define ll long long
17 #define ull unsigned long long
18 #define rep(i,a,b) for(int i=(a);i<=(b);i++)
19 #define repb(i,a,b) for(int i=(a);i>=b;i--)
20 #define mkp make_pair
21 #define ft first
22 #define sd second
23 #define log(x) (31-__builtin_clz(x))
24 #define INF 0x3f3f3f3f
25 typedef pair<int,int> pii;
26 typedef pair<ll,ll> pll;
27 ll gcd(ll a,ll b){ while(b^=a^=b^=a%=b); return a; }
28 // #define INF 0x7fffffff
29
30 void solve(){
31
32 }
33
34 int main(){
35     int z;
36     cin>>z;
37     while(z--) solve();
38 }
```

1.3 头文件 (Dallby)

```
1 #include<bits/stdc++.h>
2 cout<<"hello<<endl;
```

2 数据结构

2.1 扫描线

扫描线是离散化后，使用类似权值线段树来维护每个截面上的线段长度。

通过把二维平面上的四边形拆分成入边和出边两段，在遇到边的时候对对应的区间进行区间加/减即可。

每个节点上需要维护被完全覆盖的次数和实际长度。

```

1  #define ls (x<<1)
2  #define rs (x<<1|1)//这种方法感觉还挺好的
3
4  int cancel_sync=(ios::sync_with_stdio(0),cin.tie(0),0);
5  const int MAXN = 2e5+5;//这里要开n的两倍
6  //线结构体
7  struct Line{
8      ll l,r,h;
9      int qz;//记录位置和权值
10     bool operator < (Line &rhs){
11         return h < rhs.h;
12     }
13 }line[MAXN];
14 int n;
15 ll x1,y1,x2,y2;
16 ll X[MAXN];
17 //线段树
18 struct Segt{
19     int l,r;//是X的下标,即离散化后的
20     int sum;//sum是被完全覆盖的次数
21     ll len;//len是区间内被盖住的长度
22     //因为每次查询都是查询根节点,所以这边不需要懒惰标记
23 }t[MAXN<<3];//一个边有两个点,所以这里要开8倍
24 void build(int x,int l,int r){
25     t[x].l = l;t[x].r = r;
26     t[x].len = t[x].sum = 0;
27     if(l==r) return;//到了叶子节点
28     int mid = (l+r)>>1;
29     build(ls,l,mid);
30     build(rs,mid+1,r);
31 }
32 void push_up(int x){
33     int l = t[x].l,r = t[x].r;
34     if(t[x].sum) t[x].len = X[r+1]-X[l];//x的区间是X[l]到X[r+1]-1
35     else t[x].len = t[ls].len + t[rs].len;//合并儿子的信息
36 }
37 void update(int x,int L,int R,int v){//这里的LR存的是实际值
38     //这里如果是线段L,R,线段树上是L到R-1的部分维护
39     int l = t[x].l,r = t[x].r;
40     if(X[r+1]<=L || R<=X[l]) return;//加等于,不然会搞到无辜的线
41     if(L<=X[l]&&X[r+1]<=R){
42         t[x].sum += v;//修改覆盖次数
43         push_up(x);
44         return;
45     }
46     update(ls,L,R,v);
47     update(rs,L,R,v);
48     push_up(x);
49 }
50 int main(){
51     cin>>n;
52     rep(i,1,n){
53         cin>>x1>>y1>>x2>>y2;
54         X[2*i-1] = x1,X[2*i] = x2;//一会儿离散化要用的,这里存实际值

```

```

55     line[2*i-1] = Line{x1,x2,y1,1}; //开始的线
56     line[2*i] = Line{x1,x2,y2,-1}; //结束的线
57 }
58 n<=1; //line的数量是四边形数量的2倍
59 sort(line+1,line+1+n);
60 sort(X+1,X+1+n);
61 int tot = unique(X+1,X+1+n)-(X+1); //去除重复相邻元素,并且tot记录总数
62 build(1,1,tot-1); //为什么是tot-1?
63 //因为线段树只需要维护X[1]到X[tot]-1这一段的,实际长度是向右贴的
64 ll res = 0;
65 rep(i,1,n-1){ //每次高度是line[i+1].h-line[i].h,所以是到n-1就行
66     update(1,line[i].l,line[i].r,line[i].qz); //扫描线加入线段树
67     res += t[1].len*(line[i+1].h-line[i].h);
68 }
69 cout<<res<<endl;
70 }

```

3 数论

3.1 欧拉筛

$O(n)$ 筛素数

```

1 int primes[maxn+5],tail;
2 bool is_prime[maxn+5];
3 void euler(){
4     is_prime[1] = 1;
5     for (int i = 2; i < maxn; i++)
6     {
7         if (!is_prime[i])
8             primes[++tail]=i;
9         for (int j = 0; j < primes.size() && i * primes[j] < maxn; j++)
10        {
11            is_prime[i * primes[j]] = 1;
12            if ((i % primes[j]) == 0)
13                break;
14        }
15    }
16 }

```

3.2 Exgcd

求出 $ax + by = \gcd(a, b)$ 的一组可行解 $O(\log n)$

```

1 void Exgcd(ll a,ll b,ll &d,ll &x,ll &y){
2     if(!b){d=a;x=1;y=0;}
3     else{Exgcd(b,a%b,d,y,x);y-=x*(a/b);}
4 }

```

3.3 ExCRT 扩展中国剩余定理

$$\text{求解同余方程组} \begin{cases} x \% b_1 \equiv a_1 \\ x \% b_2 \equiv a_2 \\ \vdots \\ x \% b_n \equiv a_n \end{cases}$$

```

1 int exCRT(int a[],int b[],int n){
2     int lc=1;
3     for(int i=1;i<=n;i++){
4         lc=lcm(lc,a[i]);
5         for(int i=1;i<n;i++){
6             int p,q,g;
7             g=exgcd(a[i],a[i+1],p,q);
8             int k=(b[i+1]-b[i])/g;
9             q=-q;p*=k;q*=k;
10            b[i+1]=a[i]*p%lc+b[i];
11            b[i+1]%=lc;
12            a[i+1]=lcm(a[i],a[i+1]);
13        }
14        return (b[n]%lc+lc)%lc;
15    }

```

3.4 线性求逆元

```

1 void init(int p){
2     inv[1] = 1;
3     for(int i=2;i<=n;i++){
4         inv[i] = (ll)(p-p/i)*inv[p%i]%p;
5     }
6 }

```

3.5 多项式

3.5.1 FFT 快速傅里叶变换

```

1 const int SIZE=(1<<21)+5;
2 const double PI=acos(-1);
3 struct CP{
4     double x,y;
5     CP(double x=0,double y=0):x(x),y(y){}
6     CP operator +(const CP &A)const{return CP(x+A.x,y+A.y);}
7     CP operator -(const CP &A)const{return CP(x-A.x,y-A.y);}
8     CP operator *(const CP &A)const{return CP(x*A.x-y*A.y,x*A.y+y*A.x);}
9 };
10 int limit,rev[SIZE];
11 void DFT(CP *F,int op){
12     for(int i=0;i<limit;i++)if(i<rev[i])swap(F[i],F[rev[i]]);
13     for(int mid=1;mid<limit;mid<=1){
14         CP wn(cos(PI/mid),op*sin(PI/mid));
15         for(int len=mid<<1,k=0;k<limit;k+=len){
16             CP w(1,0);
17             for(int i=k;i<k+mid;i++){

```



```

18         CP tmp=w*F[i+mid];
19         F[i+mid]=F[i]-tmp;
20         F[i]=F[i]+tmp;
21         w=w*wn;
22     }
23 }
24 }
25 if(op==-1)for(int i=0;i<limit;i++)F[i].x/=limit;
26 }
27 void FFT(int n,int m,CP *F,CP *G){
28     for(limit=1;limit<=n+m;limit<<=1);
29     for(int i=0;i<limit;i++)rev[i]=(rev[i>>1]>>1)|((i&1)?limit>>1:0);
30     DFT(F,1),DFT(G,1);
31     for(int i=0;i<limit;i++)F[i]=F[i]*G[i];
32     DFT(F,-1);
33 }

```

3.5.2 NTT 快速数论变换

```

1 const int SIZE=(1<<21)+5;
2 int limit,rev[SIZE];
3 void DFT(ll *f, int op) {
4     const ll G = 3;
5     for(int i=0; i<limit; ++i) if(i<rev[i]) swap(f[i],f[rev[i]]);
6     for(int len=2; len<=limit; len<<=1) {
7         ll w1=pow(pow(G,(mod-1)/len),~op?1:mod-2);
8         for(int l=0, hf=len>>1; l<limit; l+=len) {
9             ll w=1;
10            for(int i=l; i<l+hf; ++i) {
11                ll tp=w*f[i+hf]%mod;
12                f[i+hf]=(f[i]-tp+mod)%mod;
13                f[i]=(f[i]+tp)%mod;
14                w=w*w1%mod;
15            }
16        }
17    }
18    if(op==-1) for(int i=0, inv=pow(limit,mod-2); i<limit; ++i) f[i]=f[i]*inv%mod;
19 }
20 void NTT(int n,int m,int *F,int *G){
21     for(limit=1;limit<=n+m;limit<<=1);
22     for(int i=0;i<limit;i++)rev[i]=(rev[i>>1]>>1)|((i&1)?limit>>1:0);
23     DFT(F,1),DFT(G,1);
24     for(int i=0;i<limit;i++)F[i]=F[i]*G[i];
25     DFT(F,-1);
26 }

```

3.5.3 MTT 任意模数 FFT

FFT 版常数巨大，慎用。

```

1 struct MTT{
2     static const int N=1<<20;
3     struct cp{
4         long double a,b;

```

```

5     cp(){a=0,b=0;}
6     cp(const long double &a,const long double &b):a(a),b(b){}
7     cp operator+(const cp &t)const{return cp(a+t.a,b+t.b);}
8     cp operator-(const cp &t)const{return cp(a-t.a,b-t.b);}
9     cp operator*(const cp &t)const{return cp(a*t.a-b*t.b,a*t.b+b*t.a);}
10    cp conj()const{return cp(a,-b);}
11
12    };
13    cp wn(int n,int f){
14        static const long double pi=acos(-1.0);
15        return cp(cos(pi/n),f*sin(pi/n));
16    }
17    int g[N];
18    void dft(cp a[],int n,int f){
19        for(int i=0;i<n;i++)if(i>g[i])swap(a[i],a[g[i]]);
20        for(int i=1;i<n;i<=1){
21            cp w=wn(i,f);
22            for(int j=0;j<n;j+=i<=1){
23                cp e(1,0);
24                for(int k=0;k<i;e=e*w,k++){
25                    cp x=a[j+k],y=a[j+k+i]*e;
26                    a[j+k]=x+y,a[j+k+i]=x-y;
27                }
28            }
29        }
30        if(f==-1){
31            cp Inv(1.0/n,0);
32            for(int i=0;i<n;i++)a[i]=a[i]*Inv;
33        }
34    }
35    cp a[N],b[N],Aa[N],Ab[N],Ba[N],Bb[N];
36    vector<ll> conv_mod(const vector<ll> &u,const vector<ll> &v,ll mod){ // 任意模数fft
37        const int n=(int)u.size()-1,m=(int)v.size()-1,M=sqrt(mod)+1;
38        const int k=32-__builtin_clz(n+m+1),s=1<<k;
39        g[0]=0; for(int i=1;i<s;i++)g[i]=(g[i/2]/2)|(((i&1)<<(k-1)));
40        for(int i=0;i<s;i++){
41            a[i]=i<=n?cp(u[i]%mod%M,u[i]%mod/M):cp();
42            b[i]=i<=m?cp(v[i]%mod%M,v[i]%mod/M):cp();
43        }
44        dft(a,s,1); dft(b,s,1);
45        for(int i=0;i<s;i++){
46            int j=(s-i)%s;
47            cp t1=(a[i]+a[j].conj())*cp(0.5,0);
48            cp t2=(a[i]-a[j].conj())*cp(0,-0.5);
49            cp t3=(b[i]+b[j].conj())*cp(0.5,0);
50            cp t4=(b[i]-b[j].conj())*cp(0,-0.5);
51            Aa[i]=t1*t3,Ab[i]=t1*t4,Ba[i]=t2*t3,Bb[i]=t2*t4;
52        }
53        for(int i=0;i<s;i++){
54            a[i]=Aa[i]+Ab[i]*cp(0,1);
55            b[i]=Ba[i]+Bb[i]*cp(0,1);
56        }
57        dft(a,s,-1); dft(b,s,-1);
58        vector<ll> ans;
59        for(int i=0;i<n+m+1;i++){
60            ll t1=llround(a[i].a)%mod;

```

```

60         ll t2=llround(a[i].b)%mod;
61         ll t3=llround(b[i].a)%mod;
62         ll t4=llround(b[i].b)%mod;
63         ans.push_back((t1+(t2+t3)*M%mod+t4*M*M)%mod);
64     }
65     return ans;
66 }
67 }mtt;

```

3.6 组合数

预处理阶乘，并通过逆元实现相除

```

1  ll jc[MAXN];
2  ll qpow(ll d,ll c){//快速幂
3      ll res = 1;
4      while(c){
5          if(c&1) res=res*d%med;
6          d=d*d%med;c>>=1;
7      }return res;
8  }
9  inline ll niyuan(ll x){return qpow(x,med-2);}
10 void initjc(){//初始化阶乘
11     jc[0] = 1;
12     rep(i,1,MAXN-1) jc[i] = jc[i-1]*i%med;
13 }
14 inline int C(int n,int m){//n是下面的
15     if(n<m) return 0;
16     return jc[n]*niyuan(jc[n-m])%med*niyuan(jc[m])%med;
17 }
18 int main(){
19     initjc();
20     int n,m;
21     while(cin>>n>>m) cout<<C(n,m)<<endl;
22 }

```

3.7 矩阵快速幂

```

1 struct Matrix{
2     ll a[MAXN][MAXN];
3     Matrix(ll x=0){
4         for(int i=0;i<n;i++){
5             for(int j=0;j<n;j++){
6                 a[i][j]=x*(i==j);
7             }
8         }
9     }
10    Matrix operator *(const Matrix &b)const{//通过重载运算符实现矩阵乘法
11        Matrix res(0);
12        for(int i=0;i<n;i++){
13            for(int j=0;j<n;j++){
14                for(int k=0;k<n;k++){
15                    ll &ma = res.a[i][j];
16                    ma = (ma+a[i][k]*b.a[k][j])%mod;

```

```
17         }
18     }
19 }
20 return res;
21 }
22 };
23 Matrix qpow(Matrix d,ll m){//底数和幂次数
24     Matrix res(1);//构造E单位矩阵
25     while(m){
26         if(m&1)
27             res=res*d;
28         d=d*d;
29         m>>=1;
30     }
31     return res;
32 }
```

3.8 高斯消元

$O(n^3)$ 复杂度，需要用 double 存储。

```
1 double date[110][110];
2 bool guass(int n){
3     for(int i=1;i<=n;i++){
4         int mix=-1;
5         for(int j=i;j<=n;j++){
6             if(date[j][i]!=0){
7                 mix=j;break;
8             }
9         }
10        if(mix==-1)
11            return false;
12        if(mix!=i)
13            for(int j=1;j<=n+1;j++)
14                swap(date[mix][j],date[i][j]);
15        double t=date[i][i];
16        for(int j=i;j<=n+1;j++){
17            date[i][j]=date[i][j]/t;
18        }
19        for(int j=1;j<=n;j++){
20            if(date[j][i]==0||j==i)
21                continue;
22            double g=date[j][i]/date[i][i];
23            for(int k=1;k<=n+1;k++)
24                date[j][k]-=date[i][k]*g;
25        }
26    }
27    return true;
28 }
```

3.9 欧拉降幂

$$a^b \equiv \begin{cases} a^{b \% \phi(p)}, & \gcd(a, p) = 1 \\ a^b, & \gcd(a, p) \neq 1, b < \phi(p) \pmod{p} \\ a^{b \% \phi(p) + \phi(p)}, & \gcd(a, p) \neq 1, b \geq \phi(p) \end{cases}$$

4 计算几何

4.1 三点求圆心

```

1 struct point{
2     double x;
3     double y;
4 };
5
6 point cal(point a, point b, point c){
7     double x1 = a.x; double y1 = a.y;
8     double x2 = b.x; double y2 = b.y;
9     double x3 = c.x; double y3 = c.y;
10    double a1 = 2*(x2-x1); double a2 = 2*(x3-x2);
11    double b1 = 2*(y2-y1); double b2 = 2*(y3-y2);
12    double c1 = x2*x2 + y2*y2 - x1*x1 - y1*y1;
13    double c2 = x3*x3 + y3*y3 - x2*x2 - y2*y2;
14    double rx = (c1*b2 - c2*b1)/(a1*b2 - a2*b1);
15    double ry = (c2*a1 - c1*a2)/(a1*b2 - a2*b1);
16    return point{rx, ry};
17 }
```

4.2 欧拉降幂

$$a^b \equiv \begin{cases} a^{b \% \phi(p)}, & \gcd(a, p) = 1 \\ a^b, & \gcd(a, p) \neq 1, b < \phi(p) \pmod{p} \\ a^{b \% \phi(p) + \phi(p)}, & \gcd(a, p) \neq 1, b \geq \phi(p) \end{cases}$$

4.3 拉格朗日插值

```

1 namespace polysum {
2 #define rep(i,a,n) for (int i=a;i<n;i++)
3 #define per(i,a,n) for (int i=n-1;i>=a;i--)
4 const int D = 1010000; ///可能需要用到的最高次
5 LL a[D], f[D], g[D], p[D], p1[D], p2[D], b[D], h[D][2], C[D];
6 LL powmod(LL a, LL b) {
7     LL res = 1;
8     a %= mod;
9     assert(b >= 0);
10
11     for (; b; b >>= 1) {
12         if (b & 1)
13             res = res * a % mod;
14
15         a = a * a % mod;
```

```

16     }
17
18     return res;
19 }
20
21 ///函数用途: 给出数列的 (d+1) 项, 其中d为最高次方项
22 ///求出数列的第n项, 数组下标从0开始
23 LL calcn(int d, LL *a, LL n) { /// a[0].. a[d] a[n]
24     if (n <= d)
25         return a[n];
26
27     p1[0] = p2[0] = 1;
28     rep(i, 0, d + 1) {
29         LL t = (n - i + mod) % mod;
30         p1[i + 1] = p1[i] * t % mod;
31     }
32     rep(i, 0, d + 1) {
33         LL t = (n - d + i + mod) % mod;
34         p2[i + 1] = p2[i] * t % mod;
35     }
36     LL ans = 0;
37     rep(i, 0, d + 1) {
38         LL t = g[i] * g[d - i] % mod * p1[i] % mod * p2[d - i] % mod * a[i] % mod;
39
40         if ((d - i) & 1)
41             ans = (ans - t + mod) % mod;
42         else
43             ans = (ans + t) % mod;
44     }
45     return ans;
46 }
47 void init(int M) {///用到的最高次
48     f[0] = f[1] = g[0] = g[1] = 1;
49     rep(i, 2, M + 5) f[i] = f[i - 1] * i % mod;
50     g[M + 4] = powmod(f[M + 4], mod - 2);
51     per(i, 1, M + 4) g[i] = g[i + 1] * (i + 1) % mod; ///费马小定理筛逆元
52 }
53
54 ///函数用途: 给出数列的 (m+1) 项, 其中m为最高次方
55 ///求出数列的前 (n-1) 项的和 (从第0项开始)
56 LL polysum(LL m, LL *a, LL n) { /// a[0].. a[m] \sum_{i=0}^{n-1} a[i]
57     for (int i = 0; i <= m; i++)
58         b[i] = a[i];
59
60     ///前n项和, 其最高次幂加1
61     b[m + 1] = calcn(m, b, m + 1);
62     rep(i, 1, m + 2) b[i] = (b[i - 1] + b[i]) % mod;
63     return calcn(m + 1, b, n - 1);
64 }
65 LL qpolysum(LL R, LL n, LL *a, LL m) { /// a[0].. a[m] \sum_{i=0}^{n-1} a[i]*R^i
66     if (R == 1)
67         return polysum(n, a, m);
68
69     a[m + 1] = calcn(m, a, m + 1);
70     LL r = powmod(R, mod - 2), p3 = 0, p4 = 0, c, ans;

```

```

71     h[0][0] = 0;
72     h[0][1] = 1;
73     rep(i, 1, m + 2) {
74         h[i][0] = (h[i - 1][0] + a[i - 1]) * r % mod;
75         h[i][1] = h[i - 1][1] * r % mod;
76     }
77     rep(i, 0, m + 2) {
78         LL t = g[i] * g[m + 1 - i] % mod;
79
80         if (i & 1)
81             p3 = ((p3 - h[i][0] * t) % mod + mod) % mod, p4 = ((p4 - h[i][1] * t) % mod + mod) % mod;
82         else
83             p3 = (p3 + h[i][0] * t) % mod, p4 = (p4 + h[i][1] * t) % mod;
84     }
85     c = powmod(p4, mod - 2) * (mod - p3) % mod;
86     rep(i, 0, m + 2) h[i][0] = (h[i][0] + h[i][1] * c) % mod;
87     rep(i, 0, m + 2) C[i] = h[i][0];
88     ans = (calcn(m, C, n) * powmod(R, n) - c) % mod;
89
90     if (ans < 0)
91         ans += mod;
92
93     return ans;
94 }
95 }

```

5 数据结构

5.1 并查集系列

5.1.1 普通并查集

带路径压缩, $O(1)$ 复杂度

```

1  int fa[maxn];
2  int find(int x){if(fa[x]^x)return fa[x]=find(fa[x]);return x;}
3  void merge(int a,int b){fa[find(a)]=find(b);}

```

5.1.2 按秩合并并查集

```

1  int fa[maxn];
2  int dep[maxn];
3  int find(int x){int now=x; while(fa[now]^now)now=fa[now];return now;}
4  void merge(int a,int b){
5      int l=find(a),r=find(b);
6      if(l==r) return;
7      if(dep[l]>dep[r])swap(l,r);
8      fa[l]=r;
9      dep[r]+=dep[l]==dep[r];
10 }

```

5.1.3 可持久化并查集

```
1 struct chair_man_tree{
2     struct node{
3         int lson,rson;
4     }tree[maxn<<5];
5     int tail=0;
6     int tail2=0;
7     int fa[maxn<<2];
8     int depth[maxn<<2];
9     inline int getnew(int pos){
10         tree[++tail]=tree[pos];
11         return tail;
12     }
13     int build(int l,int r){
14
15         if(l==r){
16             fa[++tail2]=l;
17             depth[tail2]=1;
18             return tail2;
19         }
20         int now=tail++;
21         int mid=(l+r)>>1;
22         tree[now].lson=build(l,mid);
23         tree[now].rson=build(mid+1,r);
24         return now;
25     }
26     int query(int pos,int l,int r,int qr){
27         if(l==r)
28             return pos;
29         int mid=(l+r)>>1;
30         if(qr<=mid)
31             return query(tree[pos].lson,l,mid,qr);
32         else return query(tree[pos].rson,mid+1,r,qr);
33     }
34     int update(int pos,int l,int r,int qr,int val){
35         if(l==r){
36             depth[++tail2]=depth[pos];
37             fa[tail2]=val;
38             return tail2;
39         }
40         int now=getnew(pos);
41         int mid=(l+r)>>1;
42         if(mid>=qr)
43             tree[now].lson=update(tree[now].lson,l,mid,qr,val);
44         else tree[now].rson=update(tree[now].rson,mid+1,r,qr,val);
45         return now;
46     }
47     int add(int pos,int l,int r,int qr){
48         if(l==r){
49             depth[++tail2]=depth[pos]+1;
50             fa[tail2]=fa[pos];
51             return tail2;
52         }
53         int now=getnew(pos);
```



```
54     int mid=(l+r)>>1;
55     if(mid>=qr)
56         tree[now].lson=add(tree[now].lson,l,mid,qr);
57     else tree[now].rson=add(tree[now].rson,mid+1,r,qr);
58     return now;
59 }
60 int getfa(int root,int qr){
61     int t=fa[query(root,1,n,qr)];
62     if(qr==t)
63         return qr;
64     else return getfa(root,t);
65 }
66 }t;
```

5.1.4 ETT 维护动态图连通性

待补

5.2 平衡树系列

5.2.1 fhq_treap

无旋 treap，可持久化，常数大

```
1 mt19937 rnd(514114);
2 struct fhq_treap{
3     struct node{
4         int l, r;
5         int val, key;
6         int size;
7     } fhq[maxn];
8     int cnt, root;
9     inline int newnode(int val){
10         fhq[++cnt].val = val;
11         fhq[cnt].key = rnd();
12         fhq[cnt].size = 1;
13         fhq[cnt].l = fhq[cnt].r = 0;
14         return cnt;
15     }
16     inline void pushup(int now){
17         fhq[now].size = fhq[fhq[now].l].size + fhq[fhq[now].r].size + 1;
18     }
19     void split(int now, int val, int &x, int &y){
20         if (!now){
21             x = y = 0;
22             return;
23         }
24         else if (fhq[now].val <= val){
25             x = now;
26             split(fhq[now].r, val, fhq[now].r, y);
27         }
28         else{
29             y = now;
30             split(fhq[now].l, val, x, fhq[now].l);
31         }
```

```
32     pushup(now);
33 }
34 int merge(int x, int y){
35     if (!x || !y)
36         return x + y;
37     if (fhq[x].key > fhq[y].key){
38         fhq[x].r = merge(fhq[x].r, y);
39         pushup(x);
40         return x;
41     }else{
42         fhq[y].l = merge(x, fhq[y].l);
43         pushup(y);
44         return y;
45     }
46 }
47 inline void insert(int val){
48     int x, y;
49     split(root, val, x, y);
50     root = merge(merge(x, newnode(val)), y);
51 }
52 inline void del(int val){
53     int x, y, z;
54     split(root, val - 1, x, y);
55     split(y, val, y, z);
56     y = merge(fhq[y].l, fhq[y].r);
57     root = merge(merge(x, y), z);
58 }
59 inline int getrk(int num){
60     int x, y;
61     split(root, num - 1, x, y);
62     int ans = fhq[x].size + 1;
63     root = merge(x, y);
64     return ans;
65 }
66 inline int getnum(int rank){
67     int now=root;
68     while(now)
69     {
70         if(fhq[fhq[now].l].size+1==rank)
71             break;
72         else if(fhq[fhq[now].l].size>=rank)
73             now=fhq[now].l;
74         else{
75             rank-=fhq[fhq[now].l].size+1;
76             now=fhq[now].r;
77         }
78     }
79     return fhq[now].val;
80 }
81 inline int pre(int val){
82     int x, y, ans;
83     split(root, val - 1, x, y);
84     int t = x;
85     while (fhq[t].r)
86         t = fhq[t].r;
```

```
87     ans = fhq[t].val;
88     root = merge(x, y);
89     return ans;
90 }
91 inline int aft(int val){
92     int x, y, ans;
93     split(root, val, x, y);
94     int t = y;
95     while (fhq[t].l)
96         t = fhq[t].l;
97     ans = fhq[t].val;
98     root = merge(x, y);
99     return ans;
100 }
101 } tree;
```

6 字符串

6.1 KMP

```
1 const int MAXN = 2e6+5;
2 int pi[MAXN]; //MAXN记得开大一点,因为这里要存到m+n+1长度的
3 vector<int> res; //储存答案
4
5 void getpi(const string &s){ //求s的前缀函数
6     pi[0]=0;
7     int j=0;
8     rep(i,1,s.length()-1){
9         while(j>0&&s[i]!=s[j]) j=pi[j-1]; //找到合适且最长的j
10        if(s[i]==s[j])j++; //能成功匹配的情况
11        pi[i]=j;
12    }
13 }
14
15 void kmp(string s,string t){ //在主串t中找模式串s
16     getpi(s+'#'+t);
17     int n=(int)s.length(),m=(int)t.length();
18     rep(i,n+1,m+n+1-1)
19         if(pi[i]==n) res.push_back(i-2*s.size()); //i-2n计算得左端点
20 }
```

6.2 AC 自动机

```
1 const int MAXN = 1e5+5;
2 int jdbh[MAXN]; //记录第i个模式串对应的节点编号
3 int cntcx[MAXN]; //记录第i个模式串出现的次数
4 inline int idx(char c){return c-'a';}
5 struct Node{
6     int son[26],flag,fail; //cnt记录次数,flag记录编号
7     void clr(){
8         memset(son,0,sizeof(son));
9         flag=0;
```

```
10     }
11 }trie[MAXN*10];
12 int n,cntt;//cntt记录总点数
13 string s,ms[166];
14 int maxx;
15 queue<int>q;
16 inline void insert(string &s,int num){
17     int siz = s.size(),v,u=1;
18     rep(i,0,siz-1){
19         v = idx(s[i]);
20         if(!trie[u].son[v]){trie[u].son[v] = ++cntt;trie[cntt].clr();}
21         u = trie[u].son[v];
22     }
23     trie[u].flag = num;//标记为单词,flag记录编号
24     //保证每个模式串只出现一次
25     cntcx[num] = 0;
26     jdbh[num] = u;//记录当前单词对应的节点编号
27 }
28 inline void getfail(){
29     rep(i,0,25) trie[0].son[i] = 1;
30     trie[0].flag = 0;
31     q.push(1);
32     trie[1].fail = 0;
33     int u,v,ufail;
34     while(!q.empty()){
35         u = q.front();q.pop();
36         rep(i,0,25){
37             v = trie[u].son[i];
38             ufail = trie[u].fail;
39             if(!v){trie[u].son[i]=trie[ufail].son[i];continue;}//画好一条跳fail的路
40             trie[v].fail = trie[ufail].son[i];
41             q.push(v);
42         }
43     }
44 }
45 inline void query(string &s){
46     int siz = s.size(),u = 1,v,k;
47     rep(i,0,siz-1){
48         v = idx(s[i]);
49         k = trie[u].son[v];
50         while(k){
51             if(trie[k].flag){
52                 cntcx[trie[k].flag]++;//计数
53                 maxx = max(maxx,cntcx[trie[k].flag]);
54             }
55             k = trie[k].fail;//跳fail
56         }
57         u = trie[u].son[v];//这一句其实也有跳fail的功能,很精妙
58     }
59 }
60 inline void solve(){
61     cntt = 1;
62     trie[0].clr();
63     trie[1].clr();
64     rep(i,1,n){
```

```

65     cin>>ms[i];
66     insert(ms[i],i);
67 }
68 getfail();
69 cin>>s;
70 maxx = 0;
71 query(s);
72 cout<<maxx<<endl;
73 rep(i,1,n){
74     if(cntcx[i]==maxx) cout<<ms[i]<<endl;
75 }
76 }

```

6.3 FFT 解决字符串匹配问题

可以用来解决含有通配符的字符串匹配问题定义匹配函数

$$(x, y) = (A_x - B_x)^2$$

如果两个字符相同，则满足 $C(x, y) = 0$

定义模式串和文本串 x 位置对齐时候的完全匹配函数为

$$P(x) = \sum C(i, x + i)$$

模式串在位置 x 上匹配时, $p(x) = 0$

通过将模式串 reverse 后卷积，可以快速处理每个位置 x 上的完全匹配函数 $P(x)$ 同理，如果包含通配符，则设通配符的值为 0，可以构造损失函数

$$C(x, y) = (A_x - B_x)^2 \cdot A_x \cdot B_x = A_x^3 B_x + A_x B_x^3 - 2A_x^2 B_x^2$$

通过三次 FFT 即可求得每个位置上的 $P(x)$

以下是用 FFT 解决普通字符串匹配问题的代码

即实现 KMP 的功能，复杂度较高，为 $O(n \log n)$

```

1 void solve(){
2     limit = 1,l=0;
3     cin>>n>>m;
4     cin>>s1>>s2;
5     rep(i,0,n-1) B[i].x = s1[i]-'a'+1;
6     rep(i,0,m-1) A[i].x = s2[i]-'a'+1;
7     double T = 0;
8     //T = sigma A[i]^A[i] i=0~m-1
9     rep(i,0,m-1) T += A[i].x*A[i].x;
10    //f[x] = sigma B[i]^B[i] i=0~x
11    f[0] = B[0].x*B[0].x;
12    rep(i,1,n-1) f[i] = f[i-1]+B[i].x*B[i].x;
13    //g[x] = S[i]*B[j] i+j==x
14    reverse(A,A+m);//S = A.reverse
15    //FFT预处理
16    while(limit<=n+m-2) limit<<=1,l++;
17    rep(i,0,limit-1)
18        r[i]= ( r[i>>1]>>1 )| ( (i&1)<<(l-1) );
19
20    FFT(A,1);FFT(B,1);
21    rep(i,0,limit) A[i]=A[i]*B[i];
22    FFT(A,-1);
23    rep(i,0,n-1) g[i] = (int)(A[i].x/limit+0.5);//四舍五入

```

```

24
25 //T + f(x) - f(x-m) - 2g(x);
26 double tmp;
27 rep(x,m-1,n-1){
28     tmp = T+f[x]-2*g[x];
29     if(x!=m-1) tmp -= f[x-m];
30     //cout<<tmp<<' ';
31     if(fabs(tmp)<eps) cout<<x-(m-1)+1<<endl;//输出匹配上的位置
32 }
33 cout<<endl;
34 }

```

6.4 字符串哈希

快速取子串哈希值

```

1 const int b = 131;//推荐的base, 可以选其他质数
2 void init(int n){//初始化
3     pw[0] = 1;
4     for (int i = 1; i <= n; i ++ ) {
5         h[i] = h[i-1]*b + str[i];//做每个前缀的哈希值
6         pw[i] = pw[i-1]*b;//预处理b^k的值
7     }
8 }
9 // 计算子串 str[l ~ r] 的哈希值
10 ull get(int l, int r) {
11     return h[r] - h[l-1]*pw[r-l+1];
12 }

```

6.5 后缀数组 SA+LCP

LCP(i,j) 后缀 i 和后缀 j 的最长公共前缀

```

1 int n,m;
2 string s;
3 int rk[MAXN],sa[MAXN],c[MAXN],rk2[MAXN];
4 //sa[i]存排名i的原始编号 rk[i]存编号i的排名 第二关键字rk2
5 inline void get_SA(){
6     rep(i,1,n) ++c[rk[i]=s[i]];//基数排序
7     rep(i,2,m) c[i] += c[i-1];
8     //c做前缀和, 可以知道每个关键字的排名最低在哪里
9     repb(i,n,1) sa[c[rk[i]]--] = i;//记录每个排名的原编号
10
11     for(int w=1;w<=n;w<=1){//倍增
12         int num = 0;
13         rep(i,n-w+1,n) rk2[++num] = i;//没有第二关键字的排在前面
14         rep(i,1,n) if(sa[i]>w) rk2[++num] = sa[i]-w;
15         //编号sa[i]大于w的才能作为编号sa[i]-w的第二关键字
16         rep(i,1,m) c[i] = 0;
17         rep(i,1,n) ++c[rk[i]];
18         rep(i,2,m) c[i] += c[i-1];
19         repb(i,n,1) sa[c[rk[rk2[i]]]--]=rk2[i],rk2[i]=0;
20         //同一个桶中按照第二关键字排序
21         swap(rk,rk2);
22         //这时候的rk2是这次排序用到的上一轮的rk,要计算出新的rk给下一轮排序

```

```

23
24     rk[sa[1]]=1,num=1;
25     rep(i,2,n)
26         rk[sa[i]] = (rk2[sa[i]]==rk2[sa[i-1]]&&rk2[sa[i]+w]==rk2[sa[i-1]+w])?num:++num;
27     //下一次排名的第一关键字,相同的两个元素排名也相同
28     if(num==n) break;//rk都唯一时,排序结束
29     m=num;
30 }
31 }
32 int height[MAXN];
33 inline void get_height(){
34     int k = 0,j;
35     rep(i,1,n) rk[sa[i]] = i;
36     rep(i,1,n){
37         if(rk[i]==1) continue;//第一名往前没有前缀
38         if(k) k--;//h[i]>=h[i-1]-1 即height[rk[i]]>=height[rk[i-1]]-1
39         j = sa[rk[i]-1];//找排在rk[i]前面的
40         while(j+k<=n&&i+k<=n&&s[i+k]==s[j+k]) ++k;//逐字符比较
41         //因为每次k只会-1,故++k最多只会加2n次
42         height[rk[i]] = k;
43     }
44 }
45 inline void solve(){
46     cin>>s;
47     s = ' '+s;
48     n = s.size()-1,m = 122;//m为字符个数'z'=122
49     get_SA();
50     rep(i,1,n) cout<<sa[i]<<' ';
51     cout<<endl;
52 }

```

6.6 后缀自动机 SAM

```

1 struct state{
2     int len,link;
3     map<char,int> nxt;//也可以用数组,空间换时间
4 };
5 state sta[MAXN<<1];//状态数需要设定为两倍
6 int sz,last;//sz为自动机大小
7 inline void init_SAM(){
8     sta[0].len = 0;sta[0].link = -1;//虚拟状态t0
9     sz = 1;
10    last = 0;
11 }
12 int cnt[MAXN<<1];
13 void SAM_extend(char c){
14     int cur = sz++;
15     cnt[cur] = 1;
16     sta[cur].len = sta[last].len+1;
17     int p = last;
18     //沿着last的link添加到c的转移,直到找到已经有c转移的状态p
19     while(p!=-1&&!sta[p].nxt.count(c)){
20         sta[p].nxt[c] = cur;
21         p = sta[p].link;

```

```

22     }
23     if(p==-1) sta[cur].link = 0; //情况1,没有符合的p
24     else{
25         int q = sta[p].nxt[c];
26         if(sta[q].len==sta[p].len+1) //情况2,稳定的转移(lenq=lenp+1,前面没有增加)
27             sta[cur].link = q;
28         else{ //情况3,把q的lenp+1的部分拿出来(clone),p到clone的转移是稳定的
29             int clone = sz++;
30             cnt[clone] = 0;
31             sta[clone].len = sta[p].len+1;
32             sta[clone].nxt = sta[q].nxt;
33             sta[clone].link = sta[q].link;
34             while(p!=-1 && sta[p].nxt[c]==q){ //把向q的转移指向clone
35                 sta[p].nxt[c]=clone;
36                 p=sta[p].link;
37             }
38             sta[q].link = sta[cur].link = clone; //clone是q的后缀,故linkq=clone
39         }
40     }
41     last = cur; //sta[last]包含目前处理的整个前缀!
42 }
43 string s;
44 vector<int> e[MAXN<<1];
45 void dfs(int now){
46     for(auto to:e[now]){
47         dfs(to);
48         cnt[now] += cnt[to];
49     }
50 }
51 inline void solve(){
52     cin>>s;
53     init_SAM();
54     int siz = s.size();
55     rep(i,0,siz-1) SAM_extend(s[i]);
56     rep(i,1,sz-1) e[sta[i].link].push_back(i); //link边反过来构造树
57     dfs(0);
58     ll maxx = 0;
59     rep(i,1,sz-1)
60         if(cnt[i]!=1) maxx = max(maxx,1ll*cnt[i]*sta[i].len);
61     cout<<maxx<<endl;
62 }
63 int main(){
64     solve();
65 }
66 //P3804 【模板】后缀自动机 (SAM)
67 //https://www.luogu.com.cn/problem/P3804

```

7 其他

7.1 ST 表求 RMQ

$O(n \log n)$ 预处理, $O(1)$ 查询

```
1 #define log(x) (31-__builtin_clz(x)) //谢谢hjt
```



```

2  const int MAXN = 1e5+10;
3  const int LOGN = log(MAXN)/log(2)+5; //这里要开大一点,之前因为没开大翻车了
4
5  int M[MAXN][LOGN];
6  int a[MAXN];
7  int z,m,n;
8
9  void init(){//初始化,复杂度O(nlogn)
10     for(int i=1;i<=n;i++) M[i][0]=i; //长度为1的区间最值是自己
11     for(int j=1;j<=LOGN;j++){
12         for(int i=1;i<=n-(1<<j)+1;i++){
13             if(a[M[i][j-1]]<a[M[i+(1<<(j-1))][j-1]]) M[i][j] = M[i][j-1]; //这里以最小值为例
14             else M[i][j] = M[i+(1<<j-1)][j-1];
15         }
16     }
17 }
18
19 int query(int l,int r){
20     int k = log(r-l+1)/log(2); //向下取整
21     if(a[M[l][k]]<a[M[r-(1<<k)+1][k]]) return M[l][k];
22     else return M[r-(1<<k)+1][k];
23 }

```

7.2 莫队

```

1  int cnt[MAXN]; //记录数字在区间[l,r]内出现的次数
2  int pos[MAXN],a[MAXN];
3  ll ans[MAXN];
4  int n,m,k,res;
5  struct Q{
6      int l,r,k; //k记录原来的编号
7      friend bool operator < (Q x,Q y){ //同一个分块内r小的排前面;不同分块则按分块靠前的
8          return pos[x.l]==pos[y.l]?x.r<y.r:pos[x.l]<pos[y.l];
9          //return (pos[a.l]^pos[b.l])?pos[a.l]<pos[b.l]:((pos[a.l]&1)?a.r<b.r:a.r>b.r);
10         //这条第一个和==是一样的,后面的是对于左端点在同一奇数块的区间,右端点按升序排列,反之降序
11     }
12 }q[MAXN];
13
14 void Add(int pos){
15     res -= cnt[a[pos]]*cnt[a[pos]];
16     cnt[a[pos]]++;
17     res += cnt[a[pos]]*cnt[a[pos]];
18 }
19 void Sub(int pos){
20     res -= cnt[a[pos]]*cnt[a[pos]];
21     cnt[a[pos]]--;
22     res += cnt[a[pos]]*cnt[a[pos]];
23 }
24 int main(){
25     cin>>n>>m>>k; //k为数字范围
26     memset(cnt,0,sizeof(cnt));
27     int siz = sqrt(n); //每个分块的大小
28     rep(i,1,n){
29         cin>>a[i];

```

```

30     pos[i] = i/siz;//分块
31 }
32 rep(i,1,m){
33     cin>>q[i].l>>q[i].r;
34     q[i].k = i;//记录原来的编号,用于打乱顺序后的还原
35 }
36 sort(q+1,q+1+m);
37 res = 0;//初始化res
38 int l = 1,r = 0;//当前知道的区间
39 //因为是闭区间,如果是[1,1]的话则一开始就包含一个元素了
40 rep(i,1,m){//莫队的核心,注意加减的顺序
41     while(q[i].l<l) Add(--l);
42     while(q[i].l>l) Sub(l++);
43     while(q[i].r<r) Sub(r--);
44     while(q[i].r>r) Add(++r);
45     ans[q[i].k] = res;
46 }
47 rep(i,1,m) cout<<ans[i]<<endl;
48 }

```

7.3 带修莫队

```

1  int a[MAXN],b[MAXN];//a读入一开始的序列,b记录修改后的
2  int pos[MAXN];//分块
3  int cq,cr;//统计查询修改次数
4  int R[MAXN][3];//0记位置,1记原本的值,2记修改后的值
5  ll res;
6  int ans[MAXN];//记录结果
7  int n,m;
8  void Add(int x){if(cnt[x]==0)res++;cnt[x]++;};//带修莫队的add和sub有区别
9  void Sub(int x){if(cnt[x]==1)res--;cnt[x]--;}
10 struct Q{
11     int l,r,k,t;
12     friend bool operator < (Q a,Q b){
13         return (pos[a.l]^pos[b.l])?pos[a.l]<pos[b.l]:((pos[a.r]^pos[b.r])?a.r<b.r:a.t<b.t);
14         //增加第三关键字,询问的先后顺序,用t或者k应该都行
15     }
16 }q[MAXN];
17 int main(){
18     cin>>n>>m;
19     cq = cr = 0;
20     int siz = pow(n,2.0/3.0);//这么分块最好,别问
21     rep(i,1,n){
22         cin>>a[i];
23         b[i]=a[i];
24         pos[i] = i/siz;
25     }
26     char hc;
27     rep(i,1,m){//读入修改和询问
28         cin>>hc;
29         if(hc=='Q'){
30             cin>>q[cq].l>>q[cq].r;
31             q[cq].k=cq;q[cq].t=cr;//注意这时候R[cr]还是没有的,这次询问是在R[cr-1]之后的
32             cq++;

```

```

33     }
34     else{
35         cin>>R[cr][0]>>R[cr][2];
36         R[cr][1] = b[R[cr][0]];
37         b[R[cr][0]] = R[cr][2]; //在b数组中记录更改
38         cr++;
39     }
40 }
41 sort(q,q+cq);
42 int l=1,r=0,sjc=0; //时间戳
43 res = 0;
44 rep(i,0,cq-1){
45     while(sjc<q[i].t){
46         if(l<=R[sjc][0]&&R[sjc][0]<=r) //判断修改是否在该区间内
47             Sub(R[sjc][1]),Add(R[sjc][2]);
48         a[R[sjc][0]] = R[sjc][2]; //在a上也进行更改
49         sjc++;
50     }
51     while(sjc>q[i].t){
52         sjc--;
53         if(l<=R[sjc][0]&&R[sjc][0]<=r) //判断修改是否在该区间内
54             Sub(R[sjc][2]),Add(R[sjc][1]);
55         a[R[sjc][0]] = R[sjc][1]; //在a上也进行更改
56     }
57     while(l>q[i].l) Add(a[--l]);
58     while(l<q[i].l) Sub(a[l++]);
59     while(r<q[i].r) Add(a[++r]);
60     while(r>q[i].r) Sub(a[r--]);
61     ans[q[i].k] = res;
62 }
63 rep(i,0,cq-1) cout<<ans[i]<<endl;
64 }
    
```

8 STL 等小技巧

8.1 集合 set

还可以通过 `lower_bound` 和 `upper_bound` 返回迭代器来找前驱, 后继

```

1 //并交集
2 vector<int> ANS;
3 set_union(s1.begin(),s1.end(),s2.begin(),s2.end(),inserter(ANS,ANS.begin())); //set_intersection()
4
5 //通过迭代器遍历集合
6 set<char>::iterator iter = temp1.begin();
7 while (iter!=temp1.end()){
8     cout<<*iter;
9     iter++;
10 }
    
```

8.2 快读快写 (短)

```
1 template<class T>inline void read(T &x){x=0;char o,f=1;while(o=getchar(),o<48)if(o==45)f=-f;do x=(x<<3)+(x<<1)+(o^48);while(o=getchar(),o>47);x*=f;}
2 template<class T>
3 void wt(T x){//快写
4     if(x < 0) putchar('-'), x = -x;
5     if(x >= 10) wt(x / 10);
6     putchar('0' + x % 10);
7 }
```

8.3 GCD(压行)

```
1 ll gcd(ll a,ll b){ while(b^=a^=b^=a%=b); return a; }
```

8.4 计时

```
1 inline double run_time(){
2     return 1.0*clock()/CLOCKS_PER_SEC;
3 }
```