Yakimochi

Beijing U of Posts and Tel

9646516 x JinHaonan x LucidaLu

2019 年 5 月 31 日

目　录

1 Environment

1.1 .vimrc

```
set tabstop=4
set shiftwidth=4
set autoindent
set cindent
set number
syntax on
inoremap {<tab> {}<left><return><up><end><return>
imap ` <c-n>
inoremap <c-l> <up><end><space>{<down><end><cr>}<up><end>
inoremap <c-k> <up><end><bs><bs><down><down><esc>ddi<up><end>
set timeoutlen=150
set cino=l-s
set filetype=cpp
iab ll long long
iab opn freopen("input","r",stdin)
autocmd BufNewFile *.cpp 0r ~/template.cpp
```

1.2 .bashrc

```
alias g++='g++ -Wall -Wextra -g -fsanitize=undefined -std=c++14'
```

2 Mathematics

2.1 MillerRabinAndPollardRho

```
namespace MillerRabin {
    long long Mul(long long a,long long b,long long mo){
        long long tmp=a*b-(long long)((long double)a/mo*b+1e-8)*mo;
        return (tmp%mo+mo)%mo;
    }

    long long Pow(long long a,long long b,long long mo){
        long long res=1;
        for(;b;b>>=1,a=Mul(a,a,mo))if(b&1)res=Mul(res,a,mo);
        return res;
    }

    bool IsPrime(long long n){
        if(n==2)return 1;
        if(n<2||!(n&1))return 0;
        static const int P=9,num[P]={2,3,5,7,11,13,17,19,23};
        long long x=n-1;int t=0;
        for(;!(x&1);x>>=1)++t;
        for(int i=1;i<P;++i){
            long long a=num[i]%(n-1)+1,res=Pow(a%n,x,n),last=res;
            for(int j=1;j<=t;++j){
                res=Mul(res,res,n);
```

```
                if(res==1&&last!=1&&last!=n-1)return 0;
                last=res;
            }
            if(res!=1)return 0;
        }
        return 1;
    }
}

namespace PollardRho {
    using namespace MillerRabin;
    unsigned long long seed;

    long long Rand(long long mo){
        return (seed+=417934045419982028911)%mo;
    }

    long long F(long long x,long long c,long long mo){
        return (Mul(x,x,mo)+c)%mo;
    }

    long long gcd(long long a,long long b){
        return b?gcd(b,a%b):a;
    }

    long long Get(long long c,long long n){
        long long x=Rand(n),y=F(x,c,n),p=n;
        for(;x!=y&&(p==n||p==1);x=F(x,c,n),y=F(F(y,c,n),c,n))
            p=x>y?gcd(n,x-y):gcd(n,y-x);
        return p;
    }

    void Divide(long long n,long long p[]){
        if(n<2)return;
        if(IsPrime(n)){p[++*p]=n;return;}
        for(;;){
            long long tmp=Get(Rand(n-1)+1,n);
            if(tmp!=1&&tmp!=n){
                Divide(tmp,p);
                Divide(n/tmp,p);
                return;
            }
        }
    }
}
```

2.2 Simplex

```
namespace Simplex {//(<=)+(Maximize)
    const int XN=0,XM=0;
    const double eps=1e-5,inf=1e100;
```

```
    int sgn(double const &x) {
        return (x>-eps)-(x<eps);
    }

    int n,m;
    double a[XM][XN],b[XM],c[XN],v;

    void Pivot(int l,int e) {
        b[l]/=a[l][e];
        for(int i=1;i<=n;++i)
            if(i!=e) a[l][i]/=a[l][e];
        a[l][e]=1/a[l][e];
        for(int i=1;i<=m;++i)
            if(i!=l && sgn(a[i][e])) {
                b[i]-=a[i][e]*b[l];
                for(int j=1;j<=n;++j)
                    if(j!=e)
                        a[i][j]-=a[i][e]*a[l][j];
                a[i][e]*=-a[l][e];
            }
        v+=c[e]*b[l];
        for(int i=1;i<=n;++i)
            if(i!=e)
                c[i]-=c[e]*a[l][i];
        c[e]*=-a[l][e];
    }

    double Run() {
        for(int l,e;(e=std::find_if(c+1,c+1+n,[&](double const &x)->bool {
                        return sgn(x)>0;} )-c)!=n+1;) {
            double lim=inf;
            for(int i=1;i<=m;++i)
                if(IsPositive(a[i][e]) && Reduce(lim,b[i]/a[i][e]))
                    l=i;
            if(lim==inf)
                return inf;
            else
                Pivot(l,e);
        }
        return v;
    }
}
```

2.3 Gauss

```
typedef double Square[XN][XN];
void Gauss(Square A,int n) {
    for(int i=1;i<=n;++i) {
        int id=i;
        for(int j=i+1;j<=n;++j)
            if(abs(A[j][i])>abs(A[id][i]))
                id=j;
```

```
        std::swap_ranges(A[i]+1,A[i]+n+2,A[id]+1);
        for(int k=i+1;k<=n+1;++k)
            A[i][k]/=A[i][i];
        A[i][i]=1;
        for(int j=i+1;j<=n;++j) {
            for(int k=i+1;k<=n+1;++k)
                A[j][k]-=A[j][i]*A[i][k];
            A[j][i]=0;
        }
    }
    for(int i=n;i>=1;--i) {
        for(int j=i+1;j<=n;++j) {
            A[i][n+1]-=A[j][n+1]*A[i][j];
            A[i][j]=0;
        }
    }
}
```

2.4 Determinant

```
typedef int Square[XN][XN];
//Matrix-Tree 度数-邻接
int Determinant(Square a,int n) {
    for(int i=1;i<=n;++i)
        for(int j=1;j<=n;++j)
            ((a[i][j]%=P)+=P)%=P;
    int f=1;
    for(int i=1;i<=n;++i) {
        int &A=a[i][i];
        for(int j=i+1;j<=n;++j) {
            for(int &B=a[j][i];B;f=P-f) {
                int t=A/B;
                for(int k=1;k<=n;++k)
                    a[i][k]=Minus(a[i][k],Mul(a[j][k],t));
                std::swap_ranges(a[i]+1,a[i]+1+n,a[j]+1);
            }
        }
    }
    int res=f;
    for(int i=1;i<=n;++i)
        res=Mul(a[i][i],res);
    return res;
}
```

2.5 SimpsonFormula

```
typedef std::pair<double,double> Point;
double Simpson(Point const &l,Point const &r,Point const &mid) {
    return (r.first-l.first)/6*(l.second+r.second+4*mid.second);
}
```

```
double Int(Point const &l,Point const &r,Point const &mid,double const &s,double
↪  const &eps) {
    Point m1={(l.first+mid.first)/2,Fun((l.first+mid.first)/2)},
          m2={(mid.first+r.first)/2,Fun((mid.first+r.first)/2)};
    double s1=Simpson(l,mid,m1),s2=Simpson(mid,r,m2);
    if(eps<1e-8 && fabs(s1+s2-s)<15*eps)
        return s1+s2+(s1+s2-s)/15;
    else
        return Int(l,mid,m1,s1,eps/2)+Int(mid,r,m2,s2,eps/2);
}

double Int(Point const &l,Point const &r) {
    Point mid={(l.first+r.first)/2,Fun((l.first+r.first)/2)};
    return Int(l,r,mid,Simpson(l,r,mid),1e-4);
}
```

3 Geometry

3.1 BasicDefinations

```
const double eps=1e-10;

int sgn(double const &x) {
    return (x>-eps)-(x<eps);
}

double p2(double const &x) {
    return x*x;
}

struct Point {
    double x,y;

    double Length() const {
        return sqrt(x*x+y*y);
    }

    Point Normal() const {
        return {-y,x};
    }

    Point Unit() const {
        double len=Length();
        return Point{x/len,y/len};
    }

    friend Point operator +(const Point &a,const Point &b) {
        return {a.x+b.x,a.y+b.y};
    }

    friend Point operator -(const Point &a,const Point &b) {
```

```
        return {a.x-b.x,a.y-b.y};
    }

    friend Point operator *(const Point &a,const double &k) {
        return Point{a.x*k,a.y*k};
    }

    friend Point operator /(const Point &a,const double &k) {
        return Point{a.x/k,a.y/k};
    }

    friend double Inner(const Point &a,const Point &b) {
        return a.x*b.x+a.y*b.y;
    }

    friend double Outer(const Point &a,const Point &b) {
        return a.x*b.y-a.y*b.x;
    }
};

struct Line {
    Point p,v;
    double ang;
};

double Dist(const Point &a,const Point &b) {
    return (a-b).Length();
}

double Dist(const Point &a,Line const &l) {
    return fabs(Outer(a-l.p,l.v))/l.v.Length();
}

Point Cross(Line const &l1,Line const &l2) {
    double t=Outer(l2.v,l1.p-l2.p)/Outer(l1.v,l2.v);
    return l1.p+l1.v*t;
}
```

3.2 ConvexHull

```
int ConvexHull(Point p[],int n,Point hull[]) {
    static Point stack[XN*2];
    int top=0;
    std::sort(p+1,p+1+n,[](Point const &a,Point const &b) { return a.x<b.x ||
    ↪  (a.x==b.x && a.y<b.y); });
    for(int i=1;i<=n;++i) {
        while(top>=2 && sgn(Outer(stack[top]-stack[top-1],p[i]-stack[top]))<=0)
            top--;
        stack[++top]=p[i];
    }
    int k=top+1;
    for(int i=n-1;i;--i) {
```

```
        while(top>=k && sgn(Outer(stack[top]-stack[top-1],p[i]-stack[top]))<=0)
            top--;
        stack[++top]=p[i];
    }
    if(top!=1)
        top--;
    std::copy(stack+1,stack+1+top,hull+1);
    return top;
}
```

3.3 HalfPlaneIntersect

```
bool OnLeft(const Point &p,Line const &l) {
    return sgn(Outer(l.v,p-l.p))>0;
}

bool Paral(Line const &l1,Line const &l2) {
    return sgn(Outer(l1.v,l2.v))==0;
}

int Intersect(Line l[],int n,Line uni[]) {
    std::sort(l+1,l+1+n,[](Line const &a,Line const &b) { return a.ang<b.ang; });
    static Point Qp[XN];static Line Ql[XN];
    int head,tail;Ql[head=tail=1]=l[1];
    for(int i=2;i<=n;++i) {
        while(tail-head>=1 && !OnLeft(Qp[tail-1],l[i]))
            tail--;
        while(tail-head>=1 && !OnLeft(Qp[head],l[i]))
            head++;
        Ql[++tail]=l[i];
        if(Paral(Ql[tail-1],Ql[tail])){
            --tail;
            if(OnLeft(l[i].p,Ql[tail]))
                Ql[tail]=l[i];
        }
        if(tail-head>=1)
            Qp[tail-1]=Cross(Ql[tail-1],Ql[tail]);
    }
    while(tail-head>=1 && !OnLeft(Qp[tail-1],Ql[head]))
        tail--;
    if(tail-head>=1) {
        std::copy(Ql+head,Ql+tail+1,uni+1);
        return tail-head+1;
    } else
        return 0;
}
```

3.4 MinimalCoveringCircle

```
struct Circle {
    Point o;
```

```
    double r;
    Circle(Point o,double r):o(o),r(r) {}
};

Point CircleCenter(Point p1,Point p2,Point p3) {
    long double a1=p2.x-p1.x,b1=p2.y-p1.y,c1=(a1*a1+b1*b1)/2;
    long double a2=p3.x-p1.x,b2=p3.y-p1.y,c2=(a2*a2+b2*b2)/2;
    long double d=a1*b2-a2*b1;
    return {p1.x+(c1*b2-c2*b1)/d,p1.y+(a1*c2-a2*c1)/d};
}

Circle MinCoveringCircle(Point p[],int n) {
    std::random_shuffle(p+1,p+1+n);
    Point o=p[1];double r=0;
    for(int i=2;i<=n;i++)
        if(sgn(Dist(o,p[i])-r)>0) {
            o=p[i],r=0;
            for(int j=1;j<i;j++)
                if(sgn(Dist(o,p[j])-r)>0) {
                    o=(p[i]+p[j])/2;
                    r=Dist(o,p[i]);
                    for(int k=1;k<j;k++)
                        if(sgn(Dist(o,p[k])-r)>0) {
                            o=CircleCenter(p[i],p[j],p[k]);
                            r=Dist(o,p[k]);
                        }
                }
        }
    return Circle(o,r);
}
```

3.5 DiameterOfPointSet

```
double MaxDist(Point p[],int n) {
    //
    if(n==2) {
        return Dist(p[1],p[2]);
    } else {
        double res=0;
        for(int i=1,cp=2;i<=n;++i) {
            Line cl(p[i],p[i%n+1]-p[i]);
            while(Dist(p[cp],cl)<Dist(p[cp%n+1],cl))
                cp=cp%n+1;
            Enlarge(res,std::max(Dist(p[cp],p[i]),Dist(p[cp],p[i%n+1])));
        }
        return res;
    }
}
```

3.6 ShortestDistanceBetweenPoints

```
double DC(int L,int R) {
    if(L==R)
        return inf;
    else {
        int M=(L+R)/2;double x0=p[M].x;
        double h=std::min(DC(L,M),DC(M+1,R));
        static Point s1[XN],s2[XN],t[XN];
        int c1=0,c2=0;
        for(int i=L;i<=M;++i)
            if(x0-p[i].x<=h)
                s1[++c1]=p[i];
        for(int i=M+1;i<=R;++i)
            if(p[i].x-x0<=h)
                s2[++c2]=p[i];
        for(int p1=1,p2=1;p1<=c1;++p1) {
            while(p2<=c2 && s1[p1].y-s2[p2].y>h)
                ++p2;
            for(int i=p2;i<=c2 && s2[i].y<=s1[p1].y+h;++i)
                Reduce(h,Dist(s2[i],s1[p1]));
        }
        std::merge(p+L,p+M+1,p+M+1,p+R+1,t+L,[&](auto const &a,auto const
        ↪ &b)->bool {
            return a.y!=b.y?a.y<b.y:a.x<b.x;
        });
        std::copy(t+L,t+R+1,p+L);
        return h;
    }
}
```

4 Data Structures

4.1 Splay

```
double DC(int L,int R) {
    if(L==R)
        return inf;
    else {
        int M=(L+R)/2;double x0=p[M].x;
        double h=std::min(DC(L,M),DC(M+1,R));
        static Point s1[XN],s2[XN],t[XN];
        int c1=0,c2=0;
        for(int i=L;i<=M;++i)
            if(x0-p[i].x<=h)
                s1[++c1]=p[i];
        for(int i=M+1;i<=R;++i)
            if(p[i].x-x0<=h)
                s2[++c2]=p[i];
        for(int p1=1,p2=1;p1<=c1;++p1) {
            while(p2<=c2 && s1[p1].y-s2[p2].y>h)
                ++p2;
```

```
            for(int i=p2;i<=c2 && s2[i].y<=s1[p1].y+h;++i)
                Reduce(h,Dist(s2[i],s1[p1]));
        }
        std::merge(p+L,p+M+1,p+M+1,p+R+1,t+L,[&](auto const &a,auto const
        ↪ &b)->bool {
            return a.y!=b.y?a.y<b.y:a.x<b.x;
        });
        std::copy(t+L,t+R+1,p+L);
        return h;
    }
}
```

4.2 StaticEdge-BasedDC

```
double DC(int L,int R) {
    if(L==R)
        return inf;
    else {
        int M=(L+R)/2;double x0=p[M].x;
        double h=std::min(DC(L,M),DC(M+1,R));
        static Point s1[XN],s2[XN],t[XN];
        int c1=0,c2=0;
        for(int i=L;i<=M;++i)
            if(x0-p[i].x<=h)
                s1[++c1]=p[i];
        for(int i=M+1;i<=R;++i)
            if(p[i].x-x0<=h)
                s2[++c2]=p[i];
        for(int p1=1,p2=1;p1<=c1;++p1) {
            while(p2<=c2 && s1[p1].y-s2[p2].y>h)
                ++p2;
            for(int i=p2;i<=c2 && s2[i].y<=s1[p1].y+h;++i)
                Reduce(h,Dist(s2[i],s1[p1]));
        }
        std::merge(p+L,p+M+1,p+M+1,p+R+1,t+L,[&](auto const &a,auto const
        ↪ &b)->bool {
            return a.y!=b.y?a.y<b.y:a.x<b.x;
        });
        std::copy(t+L,t+R+1,p+L);
        return h;
    }
}
```

4.3 StaticVertex-BasedDC

```
double DC(int L,int R) {
    if(L==R)
        return inf;
    else {
        int M=(L+R)/2;double x0=p[M].x;
        double h=std::min(DC(L,M),DC(M+1,R));
```

```
        static Point s1[XN],s2[XN],t[XN];
        int c1=0,c2=0;
        for(int i=L;i<=M;++i)
            if(x0-p[i].x<=h)
                s1[++c1]=p[i];
        for(int i=M+1;i<=R;++i)
            if(p[i].x-x0<=h)
                s2[++c2]=p[i];
        for(int p1=1,p2=1;p1<=c1;++p1) {
            while(p2<=c2 && s1[p1].y-s2[p2].y>h)
                ++p2;
            for(int i=p2;i<=c2 && s2[i].y<=s1[p1].y+h;++i)
                Reduce(h,Dist(s2[i],s1[p1]));
        }
        std::merge(p+L,p+M+1,p+M+1,p+R+1,t+L,[&](auto const &a,auto const
        ↪  &b)->bool {
            return a.y!=b.y?a.y<b.y:a.x<b.x;
        });
        std::copy(t+L,t+R+1,p+L);
        return h;
    }
}
```

4.4 Treap

```
double DC(int L,int R) {
    if(L==R)
        return inf;
    else {
        int M=(L+R)/2;double x0=p[M].x;
        double h=std::min(DC(L,M),DC(M+1,R));
        static Point s1[XN],s2[XN],t[XN];
        int c1=0,c2=0;
        for(int i=L;i<=M;++i)
            if(x0-p[i].x<=h)
                s1[++c1]=p[i];
        for(int i=M+1;i<=R;++i)
            if(p[i].x-x0<=h)
                s2[++c2]=p[i];
        for(int p1=1,p2=1;p1<=c1;++p1) {
            while(p2<=c2 && s1[p1].y-s2[p2].y>h)
                ++p2;
            for(int i=p2;i<=c2 && s2[i].y<=s1[p1].y+h;++i)
                Reduce(h,Dist(s2[i],s1[p1]));
        }
        std::merge(p+L,p+M+1,p+M+1,p+R+1,t+L,[&](auto const &a,auto const
        ↪  &b)->bool {
            return a.y!=b.y?a.y<b.y:a.x<b.x;
        });
        std::copy(t+L,t+R+1,p+L);
        return h;
    }
```

```
}
```

4.5 VirtualTree

```
double DC(int L,int R) {
    if(L==R)
        return inf;
    else {
        int M=(L+R)/2;double x0=p[M].x;
        double h=std::min(DC(L,M),DC(M+1,R));
        static Point s1[XN],s2[XN],t[XN];
        int c1=0,c2=0;
        for(int i=L;i<=M;++i)
            if(x0-p[i].x<=h)
                s1[++c1]=p[i];
        for(int i=M+1;i<=R;++i)
            if(p[i].x-x0<=h)
                s2[++c2]=p[i];
        for(int p1=1,p2=1;p1<=c1;++p1) {
            while(p2<=c2 && s1[p1].y-s2[p2].y>h)
                ++p2;
            for(int i=p2;i<=c2 && s2[i].y<=s1[p1].y+h;++i)
                Reduce(h,Dist(s2[i],s1[p1]));
        }
        std::merge(p+L,p+M+1,p+M+1,p+R+1,t+L,[&](auto const &a,auto const
        ↪ &b)->bool {
            return a.y!=b.y?a.y<b.y:a.x<b.x;
        });
        std::copy(t+L,t+R+1,p+L);
        return h;
    }
}
```

4.6 Aho-CorasickAutomaton

```
double DC(int L,int R) {
    if(L==R)
        return inf;
    else {
        int M=(L+R)/2;double x0=p[M].x;
        double h=std::min(DC(L,M),DC(M+1,R));
        static Point s1[XN],s2[XN],t[XN];
        int c1=0,c2=0;
        for(int i=L;i<=M;++i)
            if(x0-p[i].x<=h)
                s1[++c1]=p[i];
        for(int i=M+1;i<=R;++i)
            if(p[i].x-x0<=h)
                s2[++c2]=p[i];
        for(int p1=1,p2=1;p1<=c1;++p1) {
            while(p2<=c2 && s1[p1].y-s2[p2].y>h)
```

```
                ++p2;
            for(int i=p2;i<=c2 && s2[i].y<=s1[p1].y+h;++i)
                Reduce(h,Dist(s2[i],s1[p1]));
        }
        std::merge(p+L,p+M+1,p+M+1,p+R+1,t+L,[&](auto const &a,auto const
        ↪ &b)->bool {
            return a.y!=b.y?a.y<b.y:a.x<b.x;
        });
        std::copy(t+L,t+R+1,p+L);
        return h;
    }
}
```

4.7 BitSet

```
double DC(int L,int R) {
    if(L==R)
        return inf;
    else {
        int M=(L+R)/2;double x0=p[M].x;
        double h=std::min(DC(L,M),DC(M+1,R));
        static Point s1[XN],s2[XN],t[XN];
        int c1=0,c2=0;
        for(int i=L;i<=M;++i)
            if(x0-p[i].x<=h)
                s1[++c1]=p[i];
        for(int i=M+1;i<=R;++i)
            if(p[i].x-x0<=h)
                s2[++c2]=p[i];
        for(int p1=1,p2=1;p1<=c1;++p1) {
            while(p2<=c2 && s1[p1].y-s2[p2].y>h)
                ++p2;
            for(int i=p2;i<=c2 && s2[i].y<=s1[p1].y+h;++i)
                Reduce(h,Dist(s2[i],s1[p1]));
        }
        std::merge(p+L,p+M+1,p+M+1,p+R+1,t+L,[&](auto const &a,auto const
        ↪ &b)->bool {
            return a.y!=b.y?a.y<b.y:a.x<b.x;
        });
        std::copy(t+L,t+R+1,p+L);
        return h;
    }
}
```

4.8 Block-DividedTree

```
double DC(int L,int R) {
    if(L==R)
        return inf;
    else {
        int M=(L+R)/2;double x0=p[M].x;
```

```
        double h=std::min(DC(L,M),DC(M+1,R));
        static Point s1[XN],s2[XN],t[XN];
        int c1=0,c2=0;
        for(int i=L;i<=M;++i)
            if(x0-p[i].x<=h)
                s1[++c1]=p[i];
        for(int i=M+1;i<=R;++i)
            if(p[i].x-x0<=h)
                s2[++c2]=p[i];
        for(int p1=1,p2=1;p1<=c1;++p1) {
            while(p2<=c2 && s1[p1].y-s2[p2].y>h)
                ++p2;
            for(int i=p2;i<=c2 && s2[i].y<=s1[p1].y+h;++i)
                Reduce(h,Dist(s2[i],s1[p1]));
        }
        std::merge(p+L,p+M+1,p+M+1,p+R+1,t+L,[&](auto const &a,auto const
        ↪ &b)->bool {
            return a.y!=b.y?a.y<b.y:a.x<b.x;
        });
        std::copy(t+L,t+R+1,p+L);
        return h;
    }
}
```

4.9 DynamicChain-BasedDC

```
double DC(int L,int R) {
    if(L==R)
        return inf;
    else {
        int M=(L+R)/2;double x0=p[M].x;
        double h=std::min(DC(L,M),DC(M+1,R));
        static Point s1[XN],s2[XN],t[XN];
        int c1=0,c2=0;
        for(int i=L;i<=M;++i)
            if(x0-p[i].x<=h)
                s1[++c1]=p[i];
        for(int i=M+1;i<=R;++i)
            if(p[i].x-x0<=h)
                s2[++c2]=p[i];
        for(int p1=1,p2=1;p1<=c1;++p1) {
            while(p2<=c2 && s1[p1].y-s2[p2].y>h)
                ++p2;
            for(int i=p2;i<=c2 && s2[i].y<=s1[p1].y+h;++i)
                Reduce(h,Dist(s2[i],s1[p1]));
        }
        std::merge(p+L,p+M+1,p+M+1,p+R+1,t+L,[&](auto const &a,auto const
        ↪ &b)->bool {
            return a.y!=b.y?a.y<b.y:a.x<b.x;
        });
        std::copy(t+L,t+R+1,p+L);
        return h;
```

```
    }
}
```

4.10 FHQTreap

```
double DC(int L,int R) {
    if(L==R)
        return inf;
    else {
        int M=(L+R)/2;double x0=p[M].x;
        double h=std::min(DC(L,M),DC(M+1,R));
        static Point s1[XN],s2[XN],t[XN];
        int c1=0,c2=0;
        for(int i=L;i<=M;++i)
            if(x0-p[i].x<=h)
                s1[++c1]=p[i];
        for(int i=M+1;i<=R;++i)
            if(p[i].x-x0<=h)
                s2[++c2]=p[i];
        for(int p1=1,p2=1;p1<=c1;++p1) {
            while(p2<=c2 && s1[p1].y-s2[p2].y>h)
                ++p2;
            for(int i=p2;i<=c2 && s2[i].y<=s1[p1].y+h;++i)
                Reduce(h,Dist(s2[i],s1[p1]));
        }
        std::merge(p+L,p+M+1,p+M+1,p+R+1,t+L,[&](auto const &a,auto const
        ↪ &b)->bool {
            return a.y!=b.y?a.y<b.y:a.x<b.x;
        });
        std::copy(t+L,t+R+1,p+L);
        return h;
    }
}
```

4.11 KDTree

```
double DC(int L,int R) {
    if(L==R)
        return inf;
    else {
        int M=(L+R)/2;double x0=p[M].x;
        double h=std::min(DC(L,M),DC(M+1,R));
        static Point s1[XN],s2[XN],t[XN];
        int c1=0,c2=0;
        for(int i=L;i<=M;++i)
            if(x0-p[i].x<=h)
                s1[++c1]=p[i];
        for(int i=M+1;i<=R;++i)
            if(p[i].x-x0<=h)
                s2[++c2]=p[i];
        for(int p1=1,p2=1;p1<=c1;++p1) {
```

```
            while(p2<=c2 && s1[p1].y-s2[p2].y>h)
                ++p2;
            for(int i=p2;i<=c2 && s2[i].y<=s1[p1].y+h;++i)
                Reduce(h,Dist(s2[i],s1[p1]));
        }
        std::merge(p+L,p+M+1,p+M+1,p+R+1,t+L,[&](auto const &a,auto const
        ↪  &b)->bool {
            return a.y!=b.y?a.y<b.y:a.x<b.x;
        });
        std::copy(t+L,t+R+1,p+L);
        return h;
    }
}
```

4.12 Leftist

```
double DC(int L,int R) {
    if(L==R)
        return inf;
    else {
        int M=(L+R)/2;double x0=p[M].x;
        double h=std::min(DC(L,M),DC(M+1,R));
        static Point s1[XN],s2[XN],t[XN];
        int c1=0,c2=0;
        for(int i=L;i<=M;++i)
            if(x0-p[i].x<=h)
                s1[++c1]=p[i];
        for(int i=M+1;i<=R;++i)
            if(p[i].x-x0<=h)
                s2[++c2]=p[i];
        for(int p1=1,p2=1;p1<=c1;++p1) {
            while(p2<=c2 && s1[p1].y-s2[p2].y>h)
                ++p2;
            for(int i=p2;i<=c2 && s2[i].y<=s1[p1].y+h;++i)
                Reduce(h,Dist(s2[i],s1[p1]));
        }
        std::merge(p+L,p+M+1,p+M+1,p+R+1,t+L,[&](auto const &a,auto const
        ↪  &b)->bool {
            return a.y!=b.y?a.y<b.y:a.x<b.x;
        });
        std::copy(t+L,t+R+1,p+L);
        return h;
    }
}
```

4.13 Link-CutTrees

```
double DC(int L,int R) {
    if(L==R)
        return inf;
    else {
```

```
        int M=(L+R)/2;double x0=p[M].x;
        double h=std::min(DC(L,M),DC(M+1,R));
        static Point s1[XN],s2[XN],t[XN];
        int c1=0,c2=0;
        for(int i=L;i<=M;++i)
            if(x0-p[i].x<=h)
                s1[++c1]=p[i];
        for(int i=M+1;i<=R;++i)
            if(p[i].x-x0<=h)
                s2[++c2]=p[i];
        for(int p1=1,p2=1;p1<=c1;++p1) {
            while(p2<=c2 && s1[p1].y-s2[p2].y>h)
                ++p2;
            for(int i=p2;i<=c2 && s2[i].y<=s1[p1].y+h;++i)
                Reduce(h,Dist(s2[i],s1[p1]));
        }
        std::merge(p+L,p+M+1,p+M+1,p+R+1,t+L,[&](auto const &a,auto const
        ↪ &b)->bool {
            return a.y!=b.y?a.y<b.y:a.x<b.x;
        });
        std::copy(t+L,t+R+1,p+L);
        return h;
    }
}
```

4.14 Scapegoat

```
double DC(int L,int R) {
    if(L==R)
        return inf;
    else {
        int M=(L+R)/2;double x0=p[M].x;
        double h=std::min(DC(L,M),DC(M+1,R));
        static Point s1[XN],s2[XN],t[XN];
        int c1=0,c2=0;
        for(int i=L;i<=M;++i)
            if(x0-p[i].x<=h)
                s1[++c1]=p[i];
        for(int i=M+1;i<=R;++i)
            if(p[i].x-x0<=h)
                s2[++c2]=p[i];
        for(int p1=1,p2=1;p1<=c1;++p1) {
            while(p2<=c2 && s1[p1].y-s2[p2].y>h)
                ++p2;
            for(int i=p2;i<=c2 && s2[i].y<=s1[p1].y+h;++i)
                Reduce(h,Dist(s2[i],s1[p1]));
        }
        std::merge(p+L,p+M+1,p+M+1,p+R+1,t+L,[&](auto const &a,auto const
        ↪ &b)->bool {
            return a.y!=b.y?a.y<b.y:a.x<b.x;
        });
        std::copy(t+L,t+R+1,p+L);
```

```
        return h;
    }
}
```

5 String

5.1 SuffixArray

```
struct SuffixArray {
    int sa[XN],rank[XN],height[XN],n;

    SuffixArray(const char *s):n(strlen(s+1)) {
        static int temp[2][XN],cnt[XN],*x=temp[0],*y=temp[1];
        int m=256;
        std::fill(cnt+1,cnt+1+m,0);
        for(int i=1;i<=n;++i) cnt[x[i]=s[i]]++;
        std::partial_sum(cnt+1,cnt+1+m,cnt+1);
        for(int i=n;i>=1;--i) sa[cnt[x[i]]--]=i;
        for(int len=1;len<n;len<<=1) {
            int p=0;
            for(int i=n-len+1;i<=n;++i) y[++p]=i;
            for(int i=1;i<=n;++i) if(sa[i]>len) y[++p]=sa[i]-len;
            std::fill(cnt+1,cnt+1+m,0);
            for(int i=1;i<=n;++i) cnt[x[i]]++;
            std::partial_sum(cnt+1,cnt+1+m,cnt+1);
            for(int i=n;i>=1;--i) sa[cnt[x[y[i]]]--]=y[i];
            std::swap(x,y);x[sa[1]]=p=1;
            for(int i=2;i<=n;++i)
                x[sa[i]]=y[sa[i-1]]==y[sa[i]]
                      &&
                       ↪  (sa[i-1]+len<=n?y[sa[i-1]+len]:0)==(sa[i]+len<=n?y[sa[i]+len]:0
            if((m=p)==n) break;
        }
        for(int i=1;i<=n;++i) rank[sa[i]]=i;
        for(int i=1,len=0;i<=n;++i)
            if(rank[i]!=1) {
                int j=sa[rank[i]-1];
                while(s[i+len]==s[j+len]) ++len;
                height[rank[i]]=len;
                if(len) len--;
            }
    }
};
```

5.2 SuffixAutomaton

```
struct SuffixAutomata {

    struct Node {
        std::map<int,Node*> son;
        Node *par;
```

```
        int maxRight;
        Node(int maxRight=0):par(0),maxRight(maxRight) {}
    }*root,*last;

    long long cnt;
    SuffixAutomata():root(new Node),last(root),cnt(0) {}

    void Extend(int x) {
        Node *p=last,*nx=new Node(last->maxRight+1);
        for(;p && !p->son[x];p->son[x]=nx,p=p->par);
        if(p==0) {
            nx->par=root;
        } else {
            Node *ox=p->son[x];
            if(p->maxRight+1==ox->maxRight) {
                nx->par=ox;
            } else {
                Node *o=new Node(*ox);
                o->maxRight=p->maxRight+1;
                ox->par=nx->par=o;
                for(;p && p->son[x]==ox;p->son[x]=o,p=p->par);
            }
        }
        cnt+=nx->maxRight-nx->par->maxRight;
        last=nx;
    }
};
```

5.3 SuffixBalancedTree

```
struct SuffixBalancedTree {
    static const double alpha=0.8;

    struct Node {
        Node *son[2];
        double l,r,tag;
        int size,ndct;
        bool exist;
        char ch;
        Node *next;

        Node(double l,double r,char ch,Node
        ↪  *next):l(l),r(r),tag((l+r)/2),size(1),ndct(1),exist(1),ch(ch),next(next)
        ↪  {
            son[0]=son[1]=null;
        }

        Node(void*) {
            size=ndct=exist=0;
            ch=0;
            tag=-1;
            son[0]=son[1]=0;
```

```
    }

    void Up() {
        ndct=son[0]->ndct+1+son[1]->ndct;
        size=son[0]->size+exist+son[1]->size;
    }

    bool Unbalanced() {
        return ndct*alpha<std::max(son[0]->ndct,son[1]->ndct);
    }
}*root;

std::stack<Node*> nodes;

static Node *null;

SuffixBalancedTree():root(null) {
    nodes.push(null);
}

Node *&Insert(Node *&pos,double l,double r,char ch,Node *next) {
    if(pos==null) {
        pos=new Node(l,r,ch,next);
        nodes.push(pos);
        return null;
    } else {
        Node *&goat=ch<pos->ch || (ch==pos->ch && next->tag<pos->next->tag)
                    ?Insert(pos->son[0],l,(l+r)/2,ch,next)
                    :Insert(pos->son[1],(l+r)/2,r,ch,next);
        pos->Up();
        return pos->Unbalanced()?pos:goat;
    }
}

static Node *Flatten(Node *pos,Node *app) {
    if(pos==null)
        return app;
    else {
        pos->son[1]=Flatten(pos->son[1],app);
        return Flatten(pos->son[0],pos);
    }
}

static std::pair<Node*,Node*> Rebuild(Node *begin,double l,double r,int n) {
    if(n==0) {
        return std::pair<Node*,Node*>(null,begin);
    } else {
        int mid=(1+n)/2;
        std::pair<Node*,Node*> left=Rebuild(begin,l,(l+r)/2,mid-1);
        Node *pos=left.second;
        std::pair<Node*,Node*> right=Rebuild(pos->son[1],(l+r)/2,r,n-mid);
        pos->son[0]=left.first;
```

```
        pos->son[1]=right.first;
        pos->l=l,pos->r=r,pos->tag=(l+r)/2;
        pos->Up();
        return std::pair<Node*,Node*>(pos,right.second);
    }
}

static void Rebuild(Node *&root) {
    Node *begin=Flatten(root,null);
    root=Rebuild(begin,root->l,root->r,root->ndct).first;
}

static void Delete(Node *pos,Node *del) {
    if(pos==del) {
        pos->exist=0;
        pos->Up();
    } else {
        Delete(pos->son[pos->tag<del->tag],del);
        pos->Up();
    }
}

int LessCount(const char *s) {
    int res=0;
    for(Node *pos=root;pos!=null;) {
        Node *p=pos;
        const char *c=s;
        while(p->ch==*c) {
            p=p->next;
            ++c;
        }
        if(p->ch<*c) {
            res+=pos->son[0]->size+pos->exist;
            pos=pos->son[1];
        } else
            pos=pos->son[0];
    }
    return res;
}

void Append(char ch) {
    Node *&goat=Insert(root,0,1,ch,nodes.top());
    if(goat!=null)
        Rebuild(goat);
}

void Pop() {
    Delete(root,nodes.top());
    nodes.pop();
}

int Count(char *s,int len) {
```

```
        s[len+1]=CHAR_MAX;
        int res=LessCount(s+1);
        s[len+1]=CHAR_MIN;
        res-=LessCount(s+1);
        //null's ch must satisfy CHAR_MIN < ch < ALL
        return res;
    }
};
const double SuffixBalancedTree::alpha;
SuffixBalancedTree::Node *SuffixBalancedTree::null=new
↪  SuffixBalancedTree::Node((void*)0);
```

5.4 exKMP

```
//nxt ± ºB[i..m] Bµ  ¹«¹²ǰ
//extend ± ºA[i..n] Bµ  ¹«¹²ǰ³¤¶
void exKMP(char *A,char *B,int nxt[],int extend[]) {
    int n=strlen(A+1),m=strlen(B+1),x=1;
    nxt[1]=m;
    for(;x<m&&B[x]==B[x+1];++x);
    nxt[2]=x-1;x=2;

    for(int i=3;i<=m;++i)
        if(i+nxt[i-x+1]-1<nxt[x]+x-1)nxt[i]=nxt[i-x+1];
        else{
            int j=nxt[x]+x-i+1;
            if(j<1)j=1;
            for(;j+i-1<=m&&B[j]==B[j+i-1];++j);
            nxt[i]=j-1;
            if(nxt[x]<=nxt[i])x=i;
        }

    x=1;
    for(;A[x]==B[x];++x);
    extend[1]=x-1;
    x=1;
    for(int i=2;i<=n;++i)
        if(i+nxt[i-x+1]-1<extend[x]+x-1)extend[i]=nxt[i-x+1];
        else{
            int j=extend[x]+x-i+1;
            if(j<1)j=1;
            for(;j+i-1<=n&&B[j]==A[j+i-1];++j);
            nxt[i]=j-1;
            if(nxt[x]<=nxt[i])x=i;
        }
}
```

5.5 Manacher

```
void Manacher(char *str,int rad[])//str  ´® ma É  º  iI ¸«´  ®µİ 벬 ¶
{
```

```
    int len=strlen(str+1),l=0;
    for(int i=1;i<=len;++i){
        s[++l]='$';
        s[++l]=str[i];
    }
    s[++l]='$';s[0]='#';//s Æ    º   ´®
    rad[1]=1;
    int R=1,ID=1;//R ĝ±ĵ¼«³¤»   ®µ   Ķ   IDIR¶ µĻ   ®µ
    for(int i=1;i<=l;++i){
        if(i<R)
            rad[i]=min(rad[2*ID-i],R-i+1);//2*ID-iIi ±ĵ  ¤»   ®       µ
        else
            rad[i]=1;
        for(;s[i+rad[i]]==s[i-rad[i]];++rad[i]);
        if(R<rad[i]+i-1){
            R=rad[i]+i-1;
            ID=i;
        }
    }
    //  ´®µ  »   ®Imax{rad[i]-1}
}
```

5.6 MinimumRepresentation

```
int MinimumRepresentation(int *a,int n) {
    ++a;
    int p1=0,p2=1,len=0;
    while(p1<n && p2<n && len<n) {
        if(a[(p1+len)%n]==a[(p2+len)%n])
            len++;
        else {
            (a[(p1+len)%n]>a[(p2+len)%n]?p1:p2)+=len+1;
            if(p1==p2)
                p2++;
            len=0;
        }
    }
    return std::min(p1,p2)+1;
}
```

6 Graph Theory

6.1 KthShortestPath

```
int MinimumRepresentation(int *a,int n) {
    ++a;
    int p1=0,p2=1,len=0;
    while(p1<n && p2<n && len<n) {
        if(a[(p1+len)%n]==a[(p2+len)%n])
            len++;
        else {
```

```
            (a[(p1+len)%n]>a[(p2+len)%n]?p1:p2)+=len+1;
            if(p1==p2)
                p2++;
            len=0;
        }
    }
    return std::min(p1,p2)+1;
}
```

6.2 MinCost-MaxFlow

```
int MinimumRepresentation(int *a,int n) {
    ++a;
    int p1=0,p2=1,len=0;
    while(p1<n && p2<n && len<n) {
        if(a[(p1+len)%n]==a[(p2+len)%n])
            len++;
        else {
            (a[(p1+len)%n]>a[(p2+len)%n]?p1:p2)+=len+1;
            if(p1==p2)
                p2++;
            len=0;
        }
    }
    return std::min(p1,p2)+1;
}
```

6.3 EdgeBiconnectedComponent

```
int MinimumRepresentation(int *a,int n) {
    ++a;
    int p1=0,p2=1,len=0;
    while(p1<n && p2<n && len<n) {
        if(a[(p1+len)%n]==a[(p2+len)%n])
            len++;
        else {
            (a[(p1+len)%n]>a[(p2+len)%n]?p1:p2)+=len+1;
            if(p1==p2)
                p2++;
            len=0;
        }
    }
    return std::min(p1,p2)+1;
}
```

6.4 VertexBiconnectedComponent

```
int MinimumRepresentation(int *a,int n) {
    ++a;
    int p1=0,p2=1,len=0;
    while(p1<n && p2<n && len<n) {
```

```
        if(a[(p1+len)%n]==a[(p2+len)%n])
            len++;
        else {
            (a[(p1+len)%n]>a[(p2+len)%n]?p1:p2)+=len+1;
            if(p1==p2)
                p2++;
            len=0;
        }
    }
    return std::min(p1,p2)+1;
}
```

6.5 StronglyConnectedComponent

```
int MinimumRepresentation(int *a,int n) {
    ++a;
    int p1=0,p2=1,len=0;
    while(p1<n && p2<n && len<n) {
        if(a[(p1+len)%n]==a[(p2+len)%n])
            len++;
        else {
            (a[(p1+len)%n]>a[(p2+len)%n]?p1:p2)+=len+1;
            if(p1==p2)
                p2++;
            len=0;
        }
    }
    return std::min(p1,p2)+1;
}
```

6.6 CutEdge

```
int MinimumRepresentation(int *a,int n) {
    ++a;
    int p1=0,p2=1,len=0;
    while(p1<n && p2<n && len<n) {
        if(a[(p1+len)%n]==a[(p2+len)%n])
            len++;
        else {
            (a[(p1+len)%n]>a[(p2+len)%n]?p1:p2)+=len+1;
            if(p1==p2)
                p2++;
            len=0;
        }
    }
    return std::min(p1,p2)+1;
}
```

6.7 CutVertex

```
int MinimumRepresentation(int *a,int n) {
    ++a;
    int p1=0,p2=1,len=0;
    while(p1<n && p2<n && len<n) {
        if(a[(p1+len)%n]==a[(p2+len)%n])
            len++;
        else {
            (a[(p1+len)%n]>a[(p2+len)%n]?p1:p2)+=len+1;
            if(p1==p2)
                p2++;
            len=0;
        }
    }
    return std::min(p1,p2)+1;
}
```

6.8 Dijkstra

```
int MinimumRepresentation(int *a,int n) {
    ++a;
    int p1=0,p2=1,len=0;
    while(p1<n && p2<n && len<n) {
        if(a[(p1+len)%n]==a[(p2+len)%n])
            len++;
        else {
            (a[(p1+len)%n]>a[(p2+len)%n]?p1:p2)+=len+1;
            if(p1==p2)
                p2++;
            len=0;
        }
    }
    return std::min(p1,p2)+1;
}
```

6.9 Hungary

```
int MinimumRepresentation(int *a,int n) {
    ++a;
    int p1=0,p2=1,len=0;
    while(p1<n && p2<n && len<n) {
        if(a[(p1+len)%n]==a[(p2+len)%n])
            len++;
        else {
            (a[(p1+len)%n]>a[(p2+len)%n]?p1:p2)+=len+1;
            if(p1==p2)
                p2++;
            len=0;
        }
    }
```

```
    return std::min(p1,p2)+1;
}
```

6.10 ISAP

```
int MinimumRepresentation(int *a,int n) {
    ++a;
    int p1=0,p2=1,len=0;
    while(p1<n && p2<n && len<n) {
        if(a[(p1+len)%n]==a[(p2+len)%n])
            len++;
        else {
            (a[(p1+len)%n]>a[(p2+len)%n]?p1:p2)+=len+1;
            if(p1==p2)
                p2++;
            len=0;
        }
    }
    return std::min(p1,p2)+1;
}
```

6.11 KuhnMunkres

```
int MinimumRepresentation(int *a,int n) {
    ++a;
    int p1=0,p2=1,len=0;
    while(p1<n && p2<n && len<n) {
        if(a[(p1+len)%n]==a[(p2+len)%n])
            len++;
        else {
            (a[(p1+len)%n]>a[(p2+len)%n]?p1:p2)+=len+1;
            if(p1==p2)
                p2++;
            len=0;
        }
    }
    return std::min(p1,p2)+1;
}
```

7 Number Theory

7.1 DiscreteLogarithm

```
int MinimumRepresentation(int *a,int n) {
    ++a;
    int p1=0,p2=1,len=0;
    while(p1<n && p2<n && len<n) {
        if(a[(p1+len)%n]==a[(p2+len)%n])
            len++;
        else {
```

```
            (a[(p1+len)%n]>a[(p2+len)%n]?p1:p2)+=len+1;
            if(p1==p2)
                p2++;
            len=0;
        }
    }
    return std::min(p1,p2)+1;
}
```

7.2 ExtendedLucas

```
int MinimumRepresentation(int *a,int n) {
    ++a;
    int p1=0,p2=1,len=0;
    while(p1<n && p2<n && len<n) {
        if(a[(p1+len)%n]==a[(p2+len)%n])
            len++;
        else {
            (a[(p1+len)%n]>a[(p2+len)%n]?p1:p2)+=len+1;
            if(p1==p2)
                p2++;
            len=0;
        }
    }
    return std::min(p1,p2)+1;
}
```

7.3 LucasTheorem

```
int MinimumRepresentation(int *a,int n) {
    ++a;
    int p1=0,p2=1,len=0;
    while(p1<n && p2<n && len<n) {
        if(a[(p1+len)%n]==a[(p2+len)%n])
            len++;
        else {
            (a[(p1+len)%n]>a[(p2+len)%n]?p1:p2)+=len+1;
            if(p1==p2)
                p2++;
            len=0;
        }
    }
    return std::min(p1,p2)+1;
}
```

7.4 Min25

```
int MinimumRepresentation(int *a,int n) {
    ++a;
    int p1=0,p2=1,len=0;
    while(p1<n && p2<n && len<n) {
```

```
        if(a[(p1+len)%n]==a[(p2+len)%n])
            len++;
        else {
            (a[(p1+len)%n]>a[(p2+len)%n]?p1:p2)+=len+1;
            if(p1==p2)
                p2++;
            len=0;
        }
    }
    return std::min(p1,p2)+1;
}
```

7.5 Polynomial

```
int MinimumRepresentation(int *a,int n) {
    ++a;
    int p1=0,p2=1,len=0;
    while(p1<n && p2<n && len<n) {
        if(a[(p1+len)%n]==a[(p2+len)%n])
            len++;
        else {
            (a[(p1+len)%n]>a[(p2+len)%n]?p1:p2)+=len+1;
            if(p1==p2)
                p2++;
            len=0;
        }
    }
    return std::min(p1,p2)+1;
}
```

8 Uncategorized

8.1 ArrayPointer

```
template <class T>
struct ArrayPointer {
    int id;

    ArrayPointer(T *x=0) {
        if(!x)
            id=-1;
        else
            a[id=cnt++]=*x;
    }

    T *operator ->() {
        return a+id;
    }

    T &operator *() {
        return a[id];
```

```
    }

    static T *a;
    static int cnt;
};

/*
template <> TypeName
↪  *ArrayPointer<TypeName>::a=(TypeName*)malloc(SIZE*sizeof(TypeName));
template <> int ArrayPointer<TypeName>::cnt=0;
overload operator_new
*/
```

8.2 SharedPointer

```
template <class T>
struct SharedPointer {
    T *ptr;
    int *cnt;

    void Release() {
        if(ptr && --*cnt==0) {
            delete ptr;
            delete cnt;
        }
    }

    SharedPointer():ptr(0),cnt(0) {}

    SharedPointer(T *p):ptr(0) {
        *this=p;
    }

    SharedPointer(SharedPointer const &other):ptr(0) {
        *this=other;
    }

    ~SharedPointer() {
        Release();
    }

    T *operator ->() {
        return ptr;
    }

    T &operator *() {
        return *ptr;
    }

    bool operator ==(SharedPointer const &other) const {
        return ptr==other.ptr;
    }
```

```
    bool operator !=(SharedPointer const &other) const {
        return ptr!=other.ptr;
    }

    SharedPointer &operator =(T *p) {
        Release();
        if(p) {
            ptr=p;
            (*(cnt=new int))=1;
        } else {
            ptr=0;
            cnt=0;
        }
        return *this;
    }

    SharedPointer &operator =(SharedPointer const &other) {
        Release();
        if(other.ptr) {
            ptr=other.ptr;
            (*(cnt=other.cnt))++;
        } else {
            ptr=0;
            cnt=0;
        }
        return *this;
    }
};
```

8.3 MoTree

```
//By SiriusRen
#include <cmath>
#include <cstdio>
#include <cstring>
#include <algorithm>
using namespace std;
const int N=100050;
int n,m,q,xx,yy,Block,block[N],cnt=1,fa[N][20],tot,top,cnt1,cnt2,op,num[N];
int first[N],next[N*2],v[N*2],C[N],stk[N],deep[N],V[N],W[N],last[N],vis[N];
typedef long long ll;ll Ans[N],ans;
struct Query{
    int l,r,lca,id,time;Query(){}
    Query(int ll,int rr,int zz,int ii,int tt){l=ll,r=rr,lca=zz,id=ii,time=tt;}
    friend bool operator<(Query a,Query b){
        if(block[a.l]==block[b.l]){
            if(block[a.r]==block[b.r])return a.time<b.time;
            return block[a.r]<block[b.r];
        }
        return block[a.l]<block[b.l];
    }
```

```
}query[N];
struct Change{
    int position,color,lastcolor;Change(){}
    Change(int pp,int cc,int ll){position=pp,color=cc,lastcolor=ll;}
}change[N];
void add(int x,int y){v[tot]=y,next[tot]=first[x],first[x]=tot++;}
void dfs(int x){
    for(int i=first[x];~i;i=next[i])if(v[i]!=fa[x][0])
        fa[v[i]][0]=x,deep[v[i]]=deep[x]+1,dfs(v[i]);
    stk[++top]=x;
    if(top==Block){
        for(int i=1;i<=top;i++)block[stk[i]]=cnt;
        top=0,cnt++;
    }
}
int lca(int x,int y){
    if(deep[x]<deep[y])swap(x,y);
    for(int i=19;i>=0;i--)if(deep[x]-(1<<i)>=deep[y])x=fa[x][i];
    if(x==y)return x;
    for(int i=19;i>=0;i--)if(fa[x][i]!=fa[y][i])x=fa[x][i],y=fa[y][i];
    return fa[x][0];
}
void reverse(int x){
    if(vis[x])ans-=(ll)V[C[x]]*W[num[C[x]]],num[C[x]]--;
    else num[C[x]]++,ans+=(ll)V[C[x]]*W[num[C[x]]];
    vis[x]^=1;
}
void change_color(int x,int y){
    if(vis[x])reverse(x),C[x]=y,reverse(x);
    else C[x]=y;
}
void work(int x,int y){
    while(x!=y){
        if(deep[x]<deep[y])swap(x,y);
        reverse(x),x=fa[x][0];
    }
}
int read(){
    char p=getchar();int x=0;
    while(p<'0'||p>'9')p=getchar();
    while(p>='0'&&p<='9')x=x*10+p-'0',p=getchar();
    return x;
}
int main(){
    memset(first,-1,sizeof(first));
    scanf("%d%d%d",&n,&m,&q);
    for(int i=1;i<=m;i++)V[i]=read();
    for(int i=1;i<=n;i++)W[i]=read();
    Block=pow(n,2.0/3.0)*0.5;
    for(int i=1;i<n;i++)xx=read(),yy=read(),add(xx,yy),add(yy,xx);
    deep[1]=1,dfs(1);
    for(int i=1;i<=top;i++)block[stk[i]]=cnt;
```

```
    for(int j=1;j<=19;j++)
        for(int i=1;i<=n;i++)
            fa[i][j]=fa[fa[i][j-1]][j-1];
    for(int i=1;i<=n;i++)C[i]=read(),last[i]=C[i];
    for(int i=1;i<=q;i++){
        op=read(),xx=read(),yy=read();
        if(op){
            if(block[xx]>block[yy])swap(xx,yy);
            query[++cnt1]=Query(xx,yy,lca(xx,yy),cnt1,cnt2);
        }
        else change[++cnt2]=Change(xx,yy,last[xx]),last[xx]=yy;
    }
    sort(query+1,query+1+cnt1);
    for(int i=1,T=0;i<=cnt1;i++){
        for(;T<query[i].time;T++)change_color(change[T+1].position,change[T+1].color);
        for(;T>query[i].time;T--)change_color(change[T].position,change[T].lastcolor);
        if(i==1)work(query[i].l,query[i].r);
        else work(query[i-1].l,query[i].l),work(query[i-1].r,query[i].r);
        reverse(query[i].lca),Ans[query[i].id]=ans,reverse(query[i].lca);
    }
    for(int i=1;i<=cnt1;i++)printf("%lld\n",Ans[i]);
}
```

8.4 MoSequence

```
//By SiriusRen
#include <cmath>
#include <cstdio>
#include <algorithm>
using namespace std;
const int N=1050000;
int n,m,a[N],cnt1,cnt2,Block,block[N],xx,yy,ans,sum[N],last[N],Ans[N];
char op[105];
struct Query{
    int L,R,time,id;
    Query(int LL,int RR,int TT,int II){
        L=LL,R=RR,time=TT,id=II;
    }Query(){}
}query[N];
struct Change{
    int position,color,lastcolor;
    Change(int II,int CC,int LL){
        position=II,color=CC,lastcolor=LL;
    }Change(){}
}change[N];
bool operator<(Query a,Query b){
    if(block[a.L]==block[b.L]){
        if(a.R!=b.R)return a.R<b.R;
        return a.time<b.time;
    }
```

```
    return block[a.L]<block[b.L];
}
void update(int x,int f){
    if(f==1){if(!sum[x])ans++;sum[x]++;}
    else if(f==-1){if(sum[x]==1)ans--;sum[x]--;}
}
int main(){
    scanf("%d%d",&n,&m);
    Block=(int)pow(n,2.0/3.0);
    for(int i=1;i<=n;i++)scanf("%d",&a[i]),last[i]=a[i],block[i]=(i-1)/Block+1;
    for(int i=1;i<=m;i++){
        scanf("%s%d%d",op,&xx,&yy);
        if(op[0]=='Q')query[++cnt1]=Query(xx,yy,cnt2,cnt1);
        else change[++cnt2]=Change(xx,yy,last[xx]),last[xx]=yy;
    }
    sort(query+1,query+1+cnt1);
    for(int L=1,R=0,i=1,T=0;i<=cnt1;i++){
        for(;T<query[i].time;T++){
            if(change[T+1].position>=L&&change[T+1].position<=R)
                update(a[change[T+1].position],-1),update(change[T+1].color,1);
            a[change[T+1].position]=change[T+1].color;
        }
        for(;T>query[i].time;T--){
            if(change[T].position>=L&&change[T].position<=R)
                update(a[change[T].position],-1),update(change[T].lastcolor,1);
            a[change[T].position]=change[T].lastcolor;
        }
        for(;R<query[i].R;R++)update(a[R+1],1);
        for(;R>query[i].R;R--)update(a[R],-1);
        for(;L<query[i].L;L++)update(a[L],-1);
        for(;L>query[i].L;L--)update(a[L-1],1);
        Ans[query[i].id]=ans;
    }
    for(int i=1;i<=cnt1;i++)printf("%d\n",Ans[i]);
}
```