Yakimochi

Beijing U of Posts and Tel

9646516 x JinHaonan x LucidaLu

May 31, 2019

Contents

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 Miller Rabin and Pollard Rho

```
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+=4179340454199820289ll)%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 Simpson Formula

```
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 Basic Definations

```
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 Convex Hull

```
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 Half Plane Intersect

```
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 Minimal Covering Circle

```
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 Diameter of Point Set

```
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 Shortest Distance Between Points

```
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

```
struct Splay {
    struct Node {
        Node *fa,*son[2];
        int size;

        Node(void*):size(0) {
            fa=son[0]=son[1]=0;
        }

        Node():size(1) {
            fa=son[0]=son[1]=null;
        }

        void Adopt(Node *s,int d) {
            if(s!=null)
                s->fa=this;
            son[d]=s;
        }

        bool Type() {
            return fa->son[1]==this;
        }


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

        void Down() {

        }
    }*root,*nil[2];

    static Node *null;

    Splay(int *a,int n) {
        root=nil[0]=new Node(0);
        nil[0]->Adopt(nil[1]=new Node(0),1);
        nil[1]->Adopt(Build(a,1,n),0);
        SplayTo(nil[1],null);
    }

    void Insert(int p,int a[],int n) {
        Node *newt=Build(a,1,n),*pl=Kth(p),*pr=Kth(p+1);
        SplayTo(pl,null);
        SplayTo(pr,pl);
        pr->Adopt(newt,0);
        SplayTo(newt,null);
    }

    void Delete(int l,int r) {
        Node *pos=Split(l,r);
        pos->fa->son[pos->Type()]=null;
        SplayTo(pos->fa,null);
        Recycle(pos);
    }

    static void Recycle(Node *pos) {
        if(pos->son[0]!=null)
            Recycle(pos->son[0]);
        if(pos->son[1]!=null)
            Recycle(pos->son[1]);
        delete pos;
    }

    static Node *Build(int *a,int l,int r) {
        if(l>r)
            return null;
        int mid=(l+r)/2;
        Node *pos=new Node(a[mid]);
        pos->Adopt(Build(a,l,mid-1),0);
        pos->Adopt(Build(a,mid+1,r),1);
        pos->Up();
        return pos;
    }

    static void Trans(Node *pos) {
        Node *fa=pos->fa,*grand=fa->fa;
        fa->Down();pos->Down();
        int d=pos->Type();
        if(grand!=null)
            grand->son[fa->Type()]=pos;
        pos->fa=grand;
        fa->Adopt(pos->son[!d],d);pos->Adopt(fa,!d);
        fa->Up();
    }

    void SplayTo(Node *pos,Node *goal) {
        for(;pos->fa!=goal;Trans(pos))
            if(pos->fa->fa!=goal)
                Trans(pos->Type()==pos->fa->Type()?pos->fa:pos);
        pos->Up();
        if(goal==null)
            root=pos;
    }

    Node *Kth(int k) {
        Node *pos=root;int x;
        ++k;
        while(k) {
            pos->Down();
            if((x=pos->son[0]->size+1)==k) {
                SplayTo(pos,null);
                return pos;
            } else if(k<x)
                pos=pos->son[0];
            else {
                k-=x;
                pos=pos->son[1];
            }
        }
        return 0;
    }

    Node *Split(int l,int r) {//·µ»     ½
        Node *pl=Kth(l-1),*pr=Kth(r+1);
        SplayTo(pl,null);SplayTo(pr,pl);
        return pr->son[0];
    }
};
Splay::Node *Splay::null=new Splay::Node((void*)0);
```

4.2 Static Edge-Based DC

```
namespace StaticEdgeBasedDC {
    int vtc,n;

    struct Edge {
        int to,v;
        Edge *pre,*rev;
        bool ban;

        Edge(int to,int v,Edge *pre):to(to),v(v),pre(pre),ban(0) {}

    }*G[XN],*oG[XN];

    void AddEdge(Edge *G[],int x,int y,int v) {
        G[x]=new Edge(y,v,G[x]);
        G[y]=new Edge(x,v,G[y]);
        G[x]->rev=G[y];
        G[y]->rev=G[x];
    }

    void Rebuild(int pos,int fa) {
        int cur=pos,cnt=0;
        for(Edge *e=oG[pos];e;e=e->pre)
            if(e->to!=fa) {
                int u=e->to;
                if(++cnt==2) {
                    cnt=0;
                    AddEdge(G,cur,vtc,0);
                    cur=vtc;
                }
                AddEdge(G,cur,u,1);
                Rebuild(u,pos);
            }
    }

    int size[XN];

    int GetSize(int pos,int fa) {
        size[pos]=1;
        for(Edge *e=G[pos];e;e=e->pre)
            if(!e->ban && e->to!=fa) {
                int u=e->to;
                size[pos]+=GetSize(u,pos);
            }
        return size[pos];
    }

    std::pair<int,Edge*> Bridge(int pos,int fa,int tol) {
        std::pair<int,Edge*> res=std::pair<int,Edge*>(INF,0);
        for(Edge *e=G[pos];e;e=e->pre)
            if(!e->ban && e->to!=fa) {
                int u=e->to;
                Reduce(res,std::min(Bridge(u,pos,tol),std::pair<int,Edge*>(std::max(size[
            }
        return res;
    }

    long long DC(Edge *brg) {
        if(!brg)
            return 0;
        else {
            brg->ban=brg->rev->ban=1;
            int x=brg->to,y=brg->rev->to;
            long long res=Calc();
            Enlarge(res,std::max(DC(Bridge(x,0,GetSize(x,0)).second),DC(Bridge(y,0,GetSiz
            return res;
        }
    }

    long long Run() {
        Rebuild(1,0);
        return DC(Bridge(1,0,GetSize(1,0)).second);
    }
}
```

4.3 Static Vertex-Based DC

```
namespace StaticVertexBasedDC {
    bool ud[XN];
    int size[XN];

    int GetSize(int pos,int fa) {
        size[pos]=1;
        for(Edge *e=G[pos];e;e=e->pre) {
            int u=e->to;
            if(!ud[u] && u!=fa)
                size[pos]+=GetSize(u,pos);
        }
        return size[pos];
    }

    int Centre(int pos,int fa,int const &tol) {
        static int f[XN]={INF};
        int res=0,mxs=0;
        for(Edge *e=G[pos];e;e=e->pre) {
            int u=e->to;
            if(!ud[u] && u!=fa) {
                int t=Centre(u,pos,tol);
                if(f[t]<f[res])
                    res=t;
                Enlarge(mxs,size[u]);
            }
        }
        f[pos]=std::max(mxs,tol-size[pos]);
        return f[pos]<f[res]?pos:res;
    }


    void DC(int pos) {
        ud[pos]=1;
        for(Edge *e=G[pos];e;e=e->pre) {
            int u=e->to;
            if(!ud[u]) {

            }
        }

        for(Edge *e=G[pos];e;e=e->pre) {
            int u=e->to;
            if(!ud[u])
                DC(Centre(u,0,GetSize(u,0)));
        }

    }
}
```

4.4 Treap

```
struct Treap {
    static const int P=1e9+7;

    struct Node {
        Node *son[2];
        int key,v,cnt,size;

        Node(int v):key(rand()%P),v(v),cnt(1),size(1) {
            son[0]=son[1]=null;
        }

        Node(void*):key(P),cnt(0),size(0) {
            son[0]=son[1]=0;
        }

        int MinID() {
            return son[0]->key>son[1]->key;
        }

        void Up() {
            size=son[0]->size+cnt+son[1]->size;
        }
    }*root;

    static Node *null;

    Treap():root(null) {}
```

```
static void Trans(Node *&pos,int d) {
    Node *s=pos->son[d];
    pos->son[d]=s->son[!d];
    s->son[!d]=pos;
    pos->Up(),s->Up();
    pos=s;
}

static int Adjust(Node *&pos) {
    int d=pos->MinID();
    if(pos->key>pos->son[d]->key) {
        Trans(pos,d);
        return !d;
    } else
        return -1;
}

static void Insert(Node *&pos,int v) {
    if(pos==null)
        pos=new Node(v);
    else if(pos->v==v) {
        pos->cnt++;
        pos->Up();
    } else {
        Insert(pos->son[pos->v<v],v);
        pos->Up();
        Adjust(pos);
    }
}

static void Remove(Node *&pos) {
    if(pos->son[0]==null && pos->son[1]==null)
        pos=null;
    else {
        int p=Adjust(pos);
        Remove(pos->son[p]);
        pos->Up();
    }
}

static void Delete(Node *&pos,int v) {
    if(pos->v==v) {
        if(--pos->cnt==0) {
            pos->key=P;
            Remove(pos);
        } else
            pos->Up();
    } else {
        Delete(pos->son[pos->v<v],v);
        pos->Up();
    }
```

```
}

void Insert(int x) {
    Insert(root,x);
}

void Delete(int x) {
    Delete(root,x);
}

int Rank(int v) {
    int res=0;
    for(Node *pos=root;pos!=null;)
        if(pos->v<v) {
            res+=pos->cnt+pos->son[0]->size;
            pos=pos->son[1];
        } else
            pos=pos->son[0];
    return ++res;
}

int Kth(int k) {
    for(Node *pos=root;pos!=null;)
        if(pos->son[0]->size+1<=k && k<=pos->son[0]->size+pos->cnt)
            return pos->v;
        else if(k<=pos->son[0]->size)
            pos=pos->son[0];
        else {
            k-=pos->son[0]->size+pos->cnt;
            pos=pos->son[1];
        }
    throw;
}

int Pred(int v) {
    int res;
    for(Node *pos=root;pos!=null;) {
        if(pos->v<v) {
            res=pos->v;
            pos=pos->son[1];
        } else
            pos=pos->son[0];
    }
    return res;
}

int Succ(int v) {
    int res;
    for(Node *pos=root;pos!=null;) {
        if(pos->v>v) {
            res=pos->v;
            pos=pos->son[0];
            } else
                pos=pos->son[1];
        }
        return res;
    }
};
Treap::Node *Treap::null=new Treap::Node((void*)0);
```

4.5 Virtual Tree

```
namespace VirtualTree {
    struct Graph {
        struct Edge {
            int to,v;
            Edge *pre;

            Edge(int to,int v,Edge *pre):to(to),v(v),pre(pre) {}

        }*G[XN],*pool,*mem;

        int us[XN],T;

        Graph():pool((Edge*)malloc(XN*2*sizeof(Edge))),mem(pool) {}

        void Check(int x) {
            if(us[x]!=T) {
                us[x]=T;
                G[x]=0;
            }
        }

        Edge *&operator [](int x) {
            Check(x);
            return G[x];
        }

        void operator ()(int x,int y,int c=1) {
            Check(x);
            G[x]=new(mem++) Edge(y,c,G[x]);
        }

        void Reset() {
            mem=pool;
            ++T;
        }
    }R;

    void Build(int h[],int hc) {
        std::sort(h+1,h+1+hc,[](int a,int b)->bool { return dfn[a]<dfn[b]; });
        static int stack[XN],top;
        stack[top=0]=0;R.Reset();
        for(int i=1;i<=hc;++i) {
            for(int lca=LCA(stack[top],h[i]);stack[top]!=lca;) {
                if(dep[lca]>dep[stack[top]])
                    stack[++top]=lca;
                else {
                    R(dep[stack[top-1]]>dep[lca]?stack[top-1]:lca,stack[top]);
                    top--;
                }
            }
            stack[++top]=h[i];
        }
        for(;top;top--)
            R(stack[top-1],stack[top]);
    }
}
```

4.6 Aho-Corasick Automaton

```
struct AhoCorasickAutomaton {
    struct Node {
        Node *son[26],*fail,*last;
        int cnt;

        Node() {
            memset(son,0,sizeof(son));
            fail=last=0;
            cnt=0;
        }

    }*root;

    void Insert(char *s) {
        Node *pos=root;
        for(;*s;++s) {
            int c=*s-'a';
            if(!pos->son[c])
                pos->son[c]=new Node;
            pos=pos->son[c];
        }
        pos->cnt++;
    }

    void Build() {
        root->fail=root->last=root;
        std::queue<Node*> Q;
        for(int c=0;c<26;++c)
            if(root->son[c]) {
                root->son[c]->fail=root->son[c]->last=root;
                Q.push(root->son[c]);
            } else root->son[c]=root;
        while(!Q.empty()) {
            Node *cur=Q.front();Q.pop();
            for(int c=0;c<26;++c)
```

```
                if(cur->son[c]) {
                    Node *u=cur->son[c];
                    u->fail=cur->fail->son[c];
                    u->last=u->fail->cnt?u->fail:u->fail->last;
                    Q.push(u);
                } else cur->son[c]=cur->fail->son[c];
        }
    }

    int Calc(Node *pos) {
        int res=0;
        while(pos->cnt) {
            res+=pos->cnt;
            pos->cnt=0;
            pos=pos->last;
        }
        return res;
    }

    int Match(char *s) {
        Node *pos=root;
        int res=0;
        for(;*s;++s) {
            int c=*s-'a';
            pos=pos->son[c];
            if(pos->cnt)
                res+=Calc(pos);
        }
        return res;
    }
};
```

4.7 BitSet

```
struct BitSet {
    unsigned int64 s[(XV>>6)+1];
    int maxI;

    BitSet(int v):maxI(v>>6) {
        memset(s,0,sizeof(s));
    }

    void Set(unsigned int pos,bool val) {
        val==0?(s[pos>>6]&=~(1ull<<(pos&63))):(s[pos>>6]|=1ull<<(pos&63));
    }

    bool Test(unsigned int pos) const {
        return s[pos>>6]>>(pos&63)&1;
    }
};
```

4.8 Block-Divided Tree

```
int cnt=0,B,anc[N],stk[N],top=0,id[N];
void dfs(int u,int fa){
    for(int i=head[u],v,re=top;i;i=e[i].nxt)
        if((v=e[i].to)!=fa){
            dfs(v,u);
            if(top-re>=B){
                for(++cnt;top!=re;--top)id[stk[top]]=cnt;
                anc[cnt]=u;
            }
        }
    stk[++top]=u;
}
void divide(){
    dfs(1,0);
    //cnt 块的数量
    //anc 每块的根
    //id 每个点属于哪个块
    //B  B<= 每个块的的大小 <=3B
    if(top){
        if(!cnt)anc[++cnt]=1;
        for(;top;--top)id[stk[top]]=cnt;
    }
}
```

4.9 Dynamic Chain-Based DC

```
namespace DynamicChainBasedDC {
    struct Part {
        int top;
        Part(int top):top(top) {}
    }*cn[XN];

    int dfs[XN],dc,lbd[XN],rbd[XN],dep[XN],sz[XN],prefer[XN],fa[XN];
    void DFS(int pos) {
        int mxs=0;
        sz[pos]=1;
        for(Edge *e=G[pos];e;e=e->pre) {
            int u=e->to;
            fa[u]=pos;dep[u]=dep[pos]+1;
            DFS(u);
            sz[pos]+=sz[u];
            if(Enlarge(mxs,sz[u]))
                prefer[pos]=u;
        }
    }

    void Assign(int pos,int rt) {
        dfs[++dc]=pos;
        lbd[pos]=dc;
        cn[pos]=cn[rt]?cn[rt]:new Part(pos);
        if(prefer[pos]) {
            Assign(prefer[pos],rt);
            for(Edge *e=G[pos];e;e=e->pre)
                if(e->to!=prefer[pos])
                    Assign(e->to,e->to);
        }
        rbd[pos]=dc;
    }

    void Divide() {
        DFS(1);
        cn[1]=new Part(1);
        Assign(1,1);
    }

    void Path(int p1,int p2) {
        while(cn[p1]!=cn[p2]) {
            if(dep[cn[p1]->top]<dep[cn[p2]->top])
                std::swap(p1,p2);
            //p1~cht[p1]
            p1=fa[cn[p1]->top];
        }
        if(lbd[p1]>lbd[p2])
            std::swap(p1,p2);
        //p1~p2
    }
}
```

4.10 FHQTreap

```
struct Treap {
    struct Node {
        Node *son[2];
        int v,add,size;
        long long sum;

        Node(int v):v(v),add(0),size(1),sum(v) {
            son[0]=son[1]=null;
        }

        Node(void*):v(0),add(0),size(0),sum(0) {
            son[0]=son[1]=this;
        }

        void Add(int d) {
            add+=d;
            sum+=(long long)d*size;
            v+=d;
        }

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

        void Down() {
            if(add) {
                if(son[0]!=null)
                    (son[0]=new Node(*son[0]))->Add(add);
                if(son[1]!=null)
                    (son[1]=new Node(*son[1]))->Add(add);
                add=0;
            }
        }
    }*root;

    static Node *null;

    Treap(int a[],int n):root(Build(a,1,n)) {}

    static Node *Build(int a[],int L,int R) {
        if(L>R)
            return null;
        else {
            int M=(L+R)/2;
            Node *pos=new Node(a[M]);
            pos->son[0]=Build(a,L,M-1);
            pos->son[1]=Build(a,M+1,R);
            pos->Up();
            return pos;
        }
    }

    static std::pair<Node*,Node*> Split(Node *pos,int k) {
        if(k==0)
            return std::pair<Node*,Node*>(null,pos);
        else if(k==pos->size)
            return std::pair<Node*,Node*>(pos,null);
        else {
            (pos=new Node(*pos))->Down();
            std::pair<Node*,Node*> res;
            if(k<=pos->son[0]->size) {
                res=Split(pos->son[0],k);
                pos->son[0]=res.second;
                pos->Up();
                res.second=pos;
            } else {
                res=Split(pos->son[1],k-pos->son[0]->size-1);
                pos->son[1]=res.first;
                pos->Up();
                res.first=pos;
            }
            return res;
        }
    }

    static Node *Merge(Node *p1,Node *p2) {
        if(p1==null || p2==null)
            return p1==null?p2:p1;
        else {
            Node *pos;
            if(rand()%(p1->size+p2->size)+1<=p1->size) {
                (pos=new Node(*p1))->Down();
                pos->son[1]=Merge(pos->son[1],p2);
                pos->Up();
            } else {
                (pos=new Node(*p2))->Down();
                pos->son[0]=Merge(p1,pos->son[0]);
                pos->Up();
            }
            return pos;
        }
    }

    struct Triple {
        Node *L,*M,*R;
    };

    static Triple Split(Node *root,int l,int r) {
        std::pair<Node*,Node*> x=Split(root,r),y=Split(x.first,l-1);
        return {y.first,y.second,x.second};
    }

    static Node *Merge(Triple t) {
        return Merge(t.L,Merge(t.M,t.R));
    }
};
Treap::Node *Treap::null=new Treap::Node((void*)0);
```

4.11 KDTree

```
struct Point {
    int d[XD];
};

long long Dist(Point const &a,Point const &b) {

}

struct KDTree {
    struct Node {
        Node *son[2];
        Point p,min,max;

        Node(Point p):p(p),min(p),max(p) {
            son[0]=son[1]=null;
        }

        Node(void*):min(),max() {
            son[0]=son[1]=0;
        }

        void Up() {
            for(int i=0;i<k;++i) {
                min.d[i]=std::min(p.d[i],std::min(son[0]->min.d[i],son[1]->min.d[i]));
                max.d[i]=std::max(p.d[i],std::max(son[0]->max.d[i],son[1]->max.d[i]));
            }
        }

        long long Dist(Point const &q) {

        }
    }*root;

    static Node *null;

    KDTree(Point p[],int n):root(Build(p,1,n,0)) {}

    Node *Build(Point p[],int L,int R,int d) {
        if(L>R)
            return null;
        else {
            struct Compare {
                int d;
                Compare(int d):d(d) {}

                bool operator ()(Point const &a,Point const &b) {
                    return a.d[d]<b.d[d];
                }
            };
            int M=(L+R)/2;
            std::nth_element(p+L,p+M,p+R+1,Compare(d));
            Node *pos=new Node(p[M]);
            pos->son[0]=Build(p,L,M-1,(d+1)%k);
            pos->son[1]=Build(p,M+1,R,(d+1)%k);
            pos->Up();
            return pos;
        }
    }

    long long Query(Point p) {
        long long res;
        Query(root,p,res);
        return res;
    }

    void Query(Node *pos,Point p,long long &res) {
        if(pos==null)
```

```
            return;
        else {
            Reduce(res,Dist(pos->p,p));
            if(pos->son[0]->Dist(p)<res)
                Query(pos->son[0],p,res);
            if(pos->son[1]->Dist(p)<res)
                Query(pos->son[1],p,res);
        }
    }
};
KDTree::Node *KDTree::null=new KDTree::Node((void*)0);
```

4.12 Leftist

```
struct Leftist {
    struct Node {
        Node *son[2];
        int v;
        int dist;

        Node(int const &v):v(v),dist(1) {
            son[0]=son[1]=null;
        }

        Node(void*):v(INF),dist(0) {
            son[0]=son[1]=0;
        }

        void Maintain() {
            if(son[0]->dist<son[1]->dist)
                std::swap(son[0],son[1]);
            dist=son[1]->dist+1;
        }
    }*root;

    static Node *null;

    Leftist():root(null) {}

    static Node *Merge(Node *p1,Node *p2) {
        if(p1==null || p2==null)
            return p1==null?p2:p1;
        else {
            if(p1->v>p2->v)
                std::swap(p1,p2);
            p1->son[1]=Merge(p1->son[1],p2);
            p1->Maintain();
            return p1;
        }
    }

    void Swallow(Leftist &other) {
        root=Merge(root,other.root);
        other.root=null;
    }

    void Push(int v) {
        root=Merge(root,new Node(v));
    }

    int Pop() {
        int res=root->v;
        root=Merge(root->son[0],root->son[1]);
        return res;
    }
};
Leftist::Node *Leftist::null=new Leftist::Node((void*)0);
```

4.13 Link-Cut Trees

```
class LinkCutTrees {
public:
    LinkCutTrees() {}

    void Link(int id1,int id2) {
        Link(node[id1],node[id2]);
    }

    void Cut(int id1,int id2) {
        Cut(node[id1],node[id2]);
    }
private:
    struct Node {

        Node *son[2],*fa;
        bool rev;

        Node(void*):rev(0) {
            son[0]=son[1]=fa=0;
        }

        Node():rev(0) {
            son[0]=son[1]=fa=null;
        }

        int Type() {
            return fa->son[1]==this;
        }

        bool isRoot() {
            return fa->son[0]!=this && fa->son[1]!=this;
        }

        void Adopt(Node *s,int d) {
```

```
        son[d]=s;
        if(s!=null)
            s->fa=this;
    }

    void Reverse() {
        rev^=1;
        std::swap(son[0],son[1]);
    }

    void Up() {

    }

    void Down() {
        if(rev) {
            if(son[0]!=null)
                son[0]->Reverse();
            if(son[1]!=null)
                son[1]->Reverse();
            rev=0;
        }
    }

}node*[];

static Node *null;

static void Trans(Node *pos) {
    Node *f=pos->fa,*g=f->fa;
    f->Down();pos->Down();
    int d=pos->Type();
    if(!f->isRoot())
        g->son[f->Type()]=pos;
    pos->fa=g;
    f->Adopt(pos->son[!d],d);f->Up();
    pos->Adopt(f,!d);
}

static void Splay(Node *pos) {
    pos->Down();
    for(Node *fa;!pos->isRoot();Trans(pos))
        if(!(fa=pos->fa)->isRoot())
            Trans(pos->Type()==fa->Type()?fa:pos);
    pos->Up();
}

static void Access(Node *pos) {
    for(Node *pred=null;pos!=null;pred=pos,pos=pos->fa) {
        Splay(pos);
        pos->son[1]=pred;
        pos->Up();
        }
    }

    static void Expose(Node *pos) {
        Access(pos);
        Splay(pos);
    }

    static Node *FindRoot(Node *pos) {
        Expose(pos);
        while(pos->son[0]!=null) {
            pos->Down();
            pos=pos->son[0];
        }
        return pos;
    }

    static void MakeRoot(Node *pos) {
        Expose(pos);
        pos->Reverse();
    }

    static void Cut(Node *p1,Node *p2) {
        MakeRoot(p1);
        Expose(p2);
        p2->son[0]=p1->fa=null;
        p2->Up();
    }

    static void Link(Node *p1,Node *p2) {
        MakeRoot(p1);
        p1->fa=p2;
    }
};
LinkCutTrees::Node *LinkCutTrees::null=new LinkCutTrees::Node((void*)0);
```

4.14 Scapegoat

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

    struct Node {
        Node *son[2];
        int v,cnt,size,ndct;

        Node(int v):v(v),cnt(1),size(1),ndct(1) {
            son[0]=son[1]=null;
        }

        Node(void*):size(0),ndct(0) {
            son[0]=son[1]=this;
        }
        void Up() {
            size=son[0]->size+cnt+son[1]->size;
            ndct=son[0]->ndct+1+son[1]->ndct;
        }

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

    static Node *null;

    Scapegoat():root(null) {}

    static Node *&Insert(Node *&pos,int v) {
        if(pos==null) {
            pos=new Node(v);
            return null;
        } else if(pos->v==v) {
            pos->cnt++;
            pos->Up();
            return null;
        } else {
            Node *&goat=Insert(pos->son[pos->v<v],v);
            pos->Up();
            return pos->Unbalanced()?pos:goat;
        }
    }

    static void Delete(Node *pos,int v) {
        if(pos==null)
            return;
        else if(pos->v==v) {
            pos->cnt--;
            pos->Up();
        } else {
            Delete(pos->son[pos->v<v],v);
            pos->Up();
        }
    }

    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,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,mid-1);
            Node *pos=left.second;
            std::pair<Node*,Node*> right=Rebuild(pos->son[1],n-mid);
            pos->son[0]=left.first;
            pos->son[1]=right.first;
            pos->Up();
            return std::pair<Node*,Node*>(pos,right.second);
        }
    }

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

    void Insert(int v) {
        Node *&goat=Insert(root,v);
        if(goat!=null)
            Rebuild(goat);
    }

    void Delete(int v) {
        Delete(root,v);
    }

    int Rank(int v) {
        int res=0;
        for(Node *pos=root;pos!=null;) {
            if(pos->v<v) {
                res+=pos->son[0]->size+pos->cnt;
                pos=pos->son[1];
            } else
                pos=pos->son[0];
        }
        return ++res;
    }

    int Kth(int k) {
        for(Node *pos=root;;) {
            int le=pos->son[0]->size+pos->cnt;
            if(k<=le) {
                if(pos->son[0]->size+1<=k && k<=le && pos->cnt)
                    return pos->v;
                else
                    pos=pos->son[0];
            } else {
                k-=le;
                pos=pos->son[1];
            }
        }
        throw;
    }

    int Pred(int v) {
        return Kth(Rank(v)-1);
    }

    int Succ(int v) {
        return Kth(Rank(v+1));
    }
};
const double Scapegoat::alpha;
Scapegoat::Node *Scapegoat::null=new Scapegoat::Node((void*)0);
```

5 String

5.1 Suffix Array

```
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 Suffix Automaton

```
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 Suffix Balanced Tree

```
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 Extended KMP

```
//nxt ± ᵓB[i..m] Bµ  ¹«¹²ǰ
//extend ± ᵓA[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  ´® maǼ   º  iI  ļ«´   ®µİ 볤 ¶
{
    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 Ǽ   º  ´®
    rad[1]=1;
    int R=1,ID=1;//R ǵ±ǰ¼«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 ±ǰ  ¤»   ®      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 Minimum Representation

```
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 Kth Shortest Path with AStar

```
#include <queue>
#include <cstdio>
#include <cstring>
using namespace std;
#define N 100200
int n,m,xx[N],yy[N],zz[N],tot,first[1005],next[N],v[N],w[N],s,e,k,h[1005],vis[1005];
void add(int x,int y,int z){w[tot]=z,v[tot]=y,next[tot]=first[x],first[x]=tot++;}
struct Node{int now,h,g;}jy;
priority_queue<Node>pq;
bool operator < (Node a,Node b){return a.g+a.h>b.g+b.h;}
void Dijkstra(){
    memset(h,0x3f,sizeof(h));
    h[e]=0,jy.now=e;
    pq.push(jy);
    while(!pq.empty()){
        Node t=pq.top();pq.pop();
        if(!vis[t.now])vis[t.now]=1;
        else continue;
        for(int i=first[t.now];~i;i=next[i])
            if(!vis[v[i]]&&h[v[i]]>h[t.now]+w[i]){
                h[v[i]]=h[t.now]+w[i];
                jy.now=v[i];jy.g=h[v[i]];
                pq.push(jy);
            }
    }
}
int A_star(){
    memset(vis,0,sizeof(vis));
    jy.now=s;jy.g=0;jy.h=h[s];
    pq.push(jy);
    while(!pq.empty()){
        Node t=pq.top();pq.pop();
        vis[t.now]++;
        if(vis[t.now]>k)continue;
        if(vis[e]==k)return t.g;
        for(int i=first[t.now];~i;i=next[i]){
            jy.now=v[i],jy.g=t.g+w[i],jy.h=h[jy.now];
            pq.push(jy);
        }
    }
    return -1;
}

int main(){
    memset(first,-1,sizeof(first));
    scanf("%d%d",&n,&m);
    for(int i=1;i<=m;i++)
        scanf("%d%d%d",&xx[i],&yy[i],&zz[i]),add(yy[i],xx[i],zz[i]);
    scanf("%d%d%d",&s,&e,&k);
    if(s==e)k++;
    Dijkstra();
    tot=0,memset(first,-1,sizeof(first));
    for(int i=1;i<=m;i++)add(xx[i],yy[i],zz[i]);
    printf("%d\n",A_star());
}
```

6.2 Min Cost-Max Flow

```
struct Edge {
    int to,cap,v,cost;
    Edge *rev,*pre;

    Edge(int to,int cap,int cost,Edge
     ↪  *pre):to(to),cap(cap),v(0),cost(cost),rev(0),pre(pre) {}
}*G[XN],*preArc[XN];
int sp[XN];
int Aug(int s,int t) {
    int d=INF;
    for(int pos=t;pos!=s;pos=preArc[pos]->rev->to)
        Reduce(d,preArc[pos]->cap-preArc[pos]->v);
    for(int pos=t;pos!=s;pos=preArc[pos]->rev->to) {
        preArc[pos]->v+=d;
        preArc[pos]->rev->v-=d;
    }
    return d;
}

bool Sp(int s,int t,int n) {
    static int Q[XN];
    static bool inq[XN];
    int *end=Q+n,*head=Q,*tail=Q;//Q+n!
    std::fill(sp+1,sp+1+n,INF);
    sp[s]=0;*tail++=s;inq[s]=1;
    while(head!=tail) {
        int pos=*head;
        inq[pos]=0;//inq!!!
        head=head==end?Q:head+1;
        for(Edge *e=G[pos];e;e=e->pre)
            if(e->cap>e->v) {
                int u=e->to;
                if(Reduce(sp[u],sp[pos]+e->cost)) {
                    preArc[u]=e;
                    if(!inq[u]) {
                        inq[u]=1;
                        if(sp[u]<sp[*head]) {
                            head=head==Q?end:head-1;
                            *head=u;
                        } else {
                            *tail=u;
                            tail=tail==end?Q:tail+1;
                        }
                    }
                }
            }
    }
    return sp[t]!=INF;
}

int MCMF(int s,int t,int n) {
    int cost=0,flow=0;
    while(Sp(s,t,n)) {
        int d=Aug(s,t);
        cost+=d*sp[t];
        flow+=d;
    }
    return cost;
}
```

6.3 Edge Biconnected Component

```
#include<bits/stdc++.h>
using namespace std;
const int N=100;
struct edge{int to,nxt,flag;}e[N*2];
int head[N],n,ecnt=1;
int low[N],dfn[N],bccno[N],dfs_clock,bcc_cnt;
vector<pair<int,int> >bridge;
vector<int>bcc[N];
void addedge(int u,int v){
    e[++ecnt]=(edge){v,head[u],0};head[u]=ecnt;
    e[++ecnt]=(edge){u,head[v],0};head[v]=ecnt;
}
void dfs(int u,int fa){
    dfn[u]=low[u]=++dfs_clock;
    for(int i=head[u],v;i;i=e[i].nxt)
        if(!dfn[v=e[i].to]){
            dfs(v,u);
            low[u]=min(low[u],low[v]);
            if(low[v]>dfn[u]){
                bridge.push_back(make_pair(u,v));
                e[i].flag=e[i^1].flag=1;
            }
        }else if(dfn[v]<dfn[u]&&v!=fa){
            low[u]=min(low[u],dfn[v]);
        }
}
void dfs_(int u){
    bccno[u]=bcc_cnt;
    bcc[bcc_cnt].push_back(u);
    for(int i=head[u];i;i=e[i].nxt)
        if(!e[i].flag){
            dfs_(e[i].to);
        }
}
void tarjan(){
    for(int i=1;i<=n;++i)
        if(!dfn[i])dfs(i,-1);
    for(int i=1;i<=n;++i)
        if(!bccno[i]){
            ++bcc_cnt;
            dfs_(i);
        }
}
int main(){
    scanf("%d",&n);
    for(int i=1;i<=n;++i){
        int u,v;
        scanf("%d %d",&u,&v);
        addedge(u,v);
    }
}
```

6.4 Vertex Biconnected Component

```
#include<bits/stdc++.h>
using namespace std;
const int N=100;
struct edge{int to,nxt;}e[N*2];
int head[N],n,ecnt;
int dfn[N],low[N],bccno[N],dfs_clock,bcc_cnt;
bool iscut[N];
vector<int>bcc[N];
stack<pair<int,int> >stk;
void addedge(int u,int v){
    e[++ecnt]=(edge){v,head[u]};head[u]=ecnt;
    e[++ecnt]=(edge){u,head[v]};head[v]=ecnt;
}
void dfs(int u,int fa){
    low[u]=dfn[u]=++dfs_clock;
    int child=0;
    for(int i=head[u],v;i;i=e[i].nxt){
        if(!dfn[v=e[i].to]){
            stk.push(make_pair(u,v));
            ++child;
            dfs(v,u);
            low[u]=min(low[u],low[v]);
            if(low[v]>=dfn[u]){
                iscut[u]=1;
                bcc[++bcc_cnt].clear();
                for(;;){
                    pair<int,int>x=stk.top();stk.pop();
                    if(bccno[x.first]!=bcc_cnt){
                        bcc[bcc_cnt].push_back(x.first);
                        bccno[x.first]=bcc_cnt;
                    }
                    if(bccno[x.second]!=bcc_cnt){
                        bcc[bcc_cnt].push_back(x.second);
                        bccno[x.second]=bcc_cnt;
                    }
                    if(x.first==u&&x.second==v)break;
                }
            }
        }else if(dfn[v]<dfn[u]&&v!=fa){
            stk.push(make_pair(u,v));
            low[u]=min(low[u],dfn[v]);
        }
    }
    if(fa<0&&child==1)iscut[u]=0;
}
void tarjan(){
    for(int i=1;i<=n;++i)
        if(!dfn[i])dfs(i,-1);
}
int main(){
    scanf("%d",&n);
    for(int i=1;i<=n;++i){
        int u,v;
        scanf("%d %d",&u,&v);
        addedge(u,v);
    }
}
```

6.5 Strongly Connected Component

```
#include<bits/stdc++.h>
using namespace std;
const int N=100;
struct edge{int to,nxt;}e[N];
int head[N],n,ecnt;
int low[N],dfn[N],stk[N],sccno[N],size[N],top,dfs_clock,scc_cnt;
bool instk[N];
void addedge(int u,int v){
    e[++ecnt]=(edge){v,head[u]};head[u]=ecnt;
}
void dfs(int u){
    dfn[u]=low[u]=++dfs_clock;
    stk[++top]=u;instk[u]=1;
    for(int i=head[u],v;i;i=e[i].nxt)
        if(!dfn[v=e[i].to]){
            dfs(v);
            low[u]=min(low[u],low[v]);
        }else if(instk[v]){
            low[u]=min(low[u],dfn[v]);
        }
    if(dfn[u]==low[u])
        for(++scc_cnt;;){
            int x=stk[top--];
            instk[x]=0;
            sccno[x]=scc_cnt;
            ++size[scc_cnt];
            if(x==u)break;
        }
}
void tarjan(){
    for(int i=1;i<=n;++i)
        if(!dfn[i])dfs(i);
}
int main(){
    scanf("%d",&n);
    for(int i=1;i<=n;++i){
        int u,v;
        scanf("%d %d",&u,&v);
        addedge(u,v);
    }
}
```

6.6 Cut Edge

```
#include<bits/stdc++.h>
using namespace std;
const int N=100;
struct edge{int to,nxt;}e[N*2];
int head[N],n,ecnt;
int low[N],dfn[N],dfs_clock;
vector<pair<int,int> >bridge;
void addedge(int u,int v){
    e[++ecnt]=(edge){v,head[u]};head[u]=ecnt;
    e[++ecnt]=(edge){u,head[v]};head[v]=ecnt;
}
void dfs(int u,int fa){
    dfn[u]=low[u]=++dfs_clock;
    for(int i=head[u],v;i;i=e[i].nxt)
        if(!dfn[v=e[i].to]){
            dfs(v,u);
            low[u]=min(low[u],low[v]);
            if(low[v]>dfn[u]){
                bridge.push_back(make_pair(u,v));
            }
        }else if(dfn[v]<dfn[u]&&v!=fa){
            low[u]=min(low[u],dfn[v]);
        }
}
void tarjan(){
    for(int i=1;i<=n;++i)
        if(!dfn[i])dfs(i,-1);
}
int main(){
    scanf("%d",&n);
    for(int i=1;i<=n;++i){
        int u,v;
        scanf("%d %d",&u,&v);
        addedge(u,v);
    }
}
```

6.7 Cut Vertex

```
#include<bits/stdc++.h>
using namespace std;
const int N=100;
struct edge{int to,nxt;}e[N*2];
int head[N],n,ecnt;
int low[N],dfn[N],dfs_clock;
bool iscut[N];
void addedge(int u,int v){
    e[++ecnt]=(edge){v,head[u]};head[u]=ecnt;
    e[++ecnt]=(edge){u,head[v]};head[v]=ecnt;
}
void dfs(int u,int fa){
    dfn[u]=low[u]=++dfs_clock;
    int child=0;
    for(int i=head[u],v;i;i=e[i].nxt)
        if(!dfn[v=e[i].to]){
            dfs(v,u);
            ++child;
            low[u]=min(low[u],low[v]);
            if(low[v]>=dfn[u]){
                iscut[u]=1;
            }
        }else if(dfn[v]<dfn[u]&&v!=fa){
            low[u]=min(low[u],dfn[v]);
        }
    if(fa<0&&child==1){
        iscut[u]=0;
    }
}
void tarjan(){
    for(int i=1;i<=n;++i)
        if(!dfn[i])dfs(i,-1);
}
int main(){
    scanf("%d",&n);
    for(int i=1;i<=n;++i){
        int u,v;
        scanf("%d %d",&u,&v);
        addedge(u,v);
    }
}
```

6.8 Dijkstra

```
#include <ext/pb_ds/priority_queue.hpp>

typedef __gnu_pbds::priority_queue<std::pair<long
 ↪  long,int>,std::greater<std::pair<long long,int>
 ↪  >,__gnu_pbds::pairing_heap_tag> Heap;
long long Dijkstra(int s,int t) {
    static long long sp[XN];
    static Heap::point_iterator ref[XN];
    Heap Q;
    memset(sp,31,sizeof(sp));
    sp[s]=0;
    Q.push(std::make_pair(0,s));
    while(!Q.empty()) {
        int pos=Q.top().second;Q.pop();
        for(Edge *e=G[pos];e;e=e->pre) {
            int u=e->to;
            if(Reduce(sp[u],sp[pos]+e->v)) {
                if(ref[u]!=0)
                    Q.modify(ref[u],std::make_pair(sp[u],u));
                else
                    ref[u]=Q.push(std::make_pair(sp[u],u));
            }
        }
    }
    return sp[t];
}
```

6.9 Hungary

```
namespace Hungary{
    const int N=500+10;
    int nx,ny;
    bool vis[N],w[N][N];
    int boy[N],girl[N];
    int dfs(int x){
        for(int y=1;y<=ny;++y)
            if(w[x][y]&&!vis[y]){
                vis[y]=1;
                if(!boy[y]||dfs(boy[y])){
                    girl[x]=y,boy[y]=x;
                    return 1;
                }
            }
        return 0;
    }
    int run(){
        int res=0;
        for(int x=1;x<=nx;++x)
            if(!girl[x]){
                memset(vis,0,sizeof (bool)*(ny+1));
                res+=dfs(x);
            }
        return res;
    }
}
```

6.10 ISAP

```
const int INF=1e9,XN=200+11;

struct Edge {
    int to,cap,v;
    Edge *rev,*pre;

    Edge(int to,int cap,Edge *pre):to(to),cap(cap),v(0),rev(0),pre(pre) {}

    void *operator new(size_t flag) {
        static Edge *Pool=(Edge*)malloc((XN<<1)*sizeof(Edge)),*Me;
        return flag?Me++:(Me=Pool);
    }
}*G[XN],*preArc[XN];

int Aug(int t) {
    int d=INF;
    for(int pos=t;preArc[pos];pos=preArc[pos]->rev->to)
        Reduce(d,preArc[pos]->cap-preArc[pos]->v);
    for(int pos=t;preArc[pos];pos=preArc[pos]->rev->to) {
        preArc[pos]->v+=d;
        preArc[pos]->rev->v-=d;
    }
    return d;
}

int ISAP(int s,int t,int n) {
    static int num[XN],d[XN];
    static Edge *cArc[XN];
    std::fill(num+1,num+n,0);
    std::fill(d+1,d+1+n,0);
    std::copy(G+1,G+1+n,cArc+1);
    num[0]=n;preArc[s]=0;
    int flow=0;
    for(int pos=s;d[s]<n;) {
        if(pos==t) {
            flow+=Aug(t);
            pos=s;
        }
        bool adv=0;
        for(Edge *&e=cArc[pos];e;e=e->pre) {
            int u=e->to;
            if(e->cap>e->v && d[u]+1==d[pos]) {
                adv=1;
                preArc[pos=u]=e;
                break;
            }
        }
        if(!adv) {
            if(--num[d[pos]]==0)
                break;
            d[pos]=n;
            for(Edge *e=cArc[pos]=G[pos];e;e=e->pre)
                if(e->cap>e->v)
                    Reduce(d[pos],d[e->to]+1);
            num[d[pos]]++;
            if(pos!=s)
                pos=preArc[pos]->rev->to;//cArc
        }
    }
    return flow;
}
```

6.11 Kuhn-Munkres

```
namespace KM {
    using namespace std;
    const int N=400+10;
    const int oo=2e9+10;
    int n,boy[N],girl[N],slack[N],pre[N],q[N],lx[N],ly[N],w[N][N];
    bool visx[N],visy[N];
    void aug(int y){//翻转匹配边和非匹配边，使匹配点对+1
        for(int x,z;y;y=z){
            x=pre[y],z=girl[x];//pre为增广路径的上一个点
            girl[x]=y,boy[y]=x;
        }//girl[x]为男生x的伴侣，boy[y]为女生y的伴侣
    }
    void bfs(int s){
        memset(visx,0,sizeof (bool)*(n+1));
        memset(visy,0,sizeof (bool)*(n+1));
        for(int i=1;i<=n;++i)slack[i]=oo;
        int h=0,t=1;q[0]=s;
        for(;;){
            for(;h!=t;){
                int x=q[h++];
                visx[x]=1;
                for(int y=1;y<=n;++y)
                    if(!visy[y]){
                        if(lx[x]+ly[y]==w[x][y]){
                            pre[y]=x;
                            if(!boy[y]){
                                aug(y);
                                return;//找到完备匹配
                            }else{
                                visy[y]=1;
                                q[t++]=boy[y];
                            }
                        }else if(lx[x]+ly[y]-w[x][y]<slack[y]){
                            pre[y]=x;
                            slack[y]=lx[x]+ly[y]-w[x][y];//更新 slack
                        }
                    }
            }
            int d=oo;
            for(int y=1;y<=n;++y)
                if(!visy[y])d=min(d,slack[y]);
            for(int i=1;i<=n;++i){
                if(visx[i])lx[i]-=d;
                if(visy[i])ly[i]+=d;else slack[i]-=d;//松弛操作
            }
            for(int y=1;y<=n;++y){
                if(!visy[y]&&!slack[y]){
                    if(!boy[y]){
                        aug(y);
                        return;
                    }else{
                        visy[y]=1;
                        q[t++]=boy[y];//松弛之后加入新的点
                    }
                }
            }
        }
    }
    long long run(int nx,int ny){//nx 为男生数量,ny 为女生数量
        n=max(nx,ny);//补足人数
        for(int i=1;i<=n;++i)
            for(int j=1;j<=n;++j)
                lx[i]=max(lx[i],w[i][j]);//lx,ly 为点标,w 为边权
        for(int i=1;i<=n;++i)bfs(i);
        long long res=0;
        for(int i=1;i<=n;++i)res+=lx[i]+ly[i];
        return res;
        //w[i][girl[i]]?girl[i]:0
    }
}

int main() {

}
```

7 Number Theory

7.1 Discrete Logarithm with BSGS

```
int BSGS(int y,int z,int P) {
    if(y%P) {
        std::unordered_map<int,int> S;
        int B=sqrt(P)+0.5;
        long long zyi=z;
        for(int i=0;i<=B;i++,(zyi*=y)%=P)
            if(!S.count(zyi))
                S[zyi]=i;
        int yb=Pow(y,B,P);
        long long ybi=yb;
        for(int i=1;i<=B;i++) {
            if(S.count(ybi))
                return B*i-S[ybi];
            (ybi*=yb)%=P;
        }
    }
    return -1;
}
```

7.2 Extended Lucas

```
int Exgcd(int a,int b,long long &x,long long &y) {
    if(!b) {
        x=1,y=0;
        return a;
    } else {
        int d=Exgcd(b,a%b,x,y);
        long long t=y;y=x-(a/b)*y,x=t;
        return d;
    }
}

int Inverse(int a,int n) {
    long long x,y;
    int d=Exgcd(a,n,x,y);
    assert(d==1);
    return (x%n+n)%n;
}

int Pow(long long base,long long v,int P) {
    long long res=1;
    for(;v;v>>=1,(base*=base)%=P)
        if(v&1)
            (res*=base)%=P;
    return res;
}

struct Lucas {
    struct Divisor {
        int p,t,pt,tM;
        std::vector<int> table;

        Divisor(int p,int t,int pt,int tM):p(p),t(t),pt(pt),tM(tM),table(pt) {
            table[0]=1;
            for(int i=1;i<pt;++i)//0?
                table[i]=i%p==0?table[i-1]:(long long)table[i-1]*i%pt;
        }

        int Calc(long long n) {
            if(n<p)//pt..
                return table[n];
            else
                return (long long)Calc(n/p)*Pow(table[pt-1],n/pt,pt)%pt*table[n%pt]%pt;
        }

        long long CalcTimes(long long x) {
            long long res=0;
            for(;x;x/=p)
                res+=x/p;
            return res;
        }

        long long Solve(long long n,long long m) {
            long long times=CalcTimes(n)-CalcTimes(m)-CalcTimes(n-m);
            if(times>=t)
                return 0;
            else
                return (long long)Pow(p,times,pt) *Calc(n)%pt *Inverse((long long)Calc(m)*Calc(n-m)%pt,pt)%pt *tM;
        }
    };

    int P;
    std::vector<Divisor> ps;

    Lucas(int P):P(P) {
        for(int d=2,x=P;x!=1;d=(long long)d*d<=P?d+1:x)
            if(x%d==0) {
                int t=0,pt=1;
                do {
                    ++t;pt*=d;
                    x/=d;
                } while(x%d==0);
                ps.push_back(Divisor(d,t,pt,(long long)Inverse(P/pt,pt)*(P/pt)%P));
            }
    }

    int operator ()(long long n,long long m) {
        long long res=0;
        for(Divisor &d : ps)
            (res+=d.Solve(n,m))%=P;
        return res;
    }
};
```

7.3 Lucas Theorem

```
int Lucas(int n,int m) {
    int res=1;
    while(n && m) {
        (res*=C(n%P,m%P))%=P;
        n/=P,m/=P;
    }
    return res;
}
```

7.4 Min25

```
const int N=1e5,XN=N+11;

int prime[XN*2],pcnt;
void Prep() {
    static bool notPrime[XN*2];
    for(int i=2;i<=N*2;++i) {
        if(!notPrime[i])
            prime[++pcnt]=i;
        for(int j=1;j<=pcnt && i*prime[j]<=N*2;++j) {
            notPrime[i*prime[j]]=1;
            if(i%prime[j]==0)
                break;
        }
    }
}

namespace Min25 {
    typedef unsigned long long ans_t;

    std::function<ans_t(int,int)> F;

    long long n;
    int lim,psz;

    struct Identifier {
        int id[2][XN],cnt;
        int &operator [](long long x) {
            int &res=x<=lim?id[0][x]:id[1][n/x];
            if(res==0)
                res=++cnt;
            return res;
        }
    }id;

    ans_t g[XN*2],fps[XN];

    ans_t H(long long n,int m) {
        if(n<=1 || m>psz)
            return 0;
        ans_t res=g[id[n]]-fps[m-1];
        for(int i=m;i<=psz && (long long)prime[i]*prime[i]<=n;++i) {
            long long pt=prime[i],pt1=pt*prime[i];
            for(int t=1;pt1<=n;++t,pt=pt1,pt1*=prime[i])
                res+=F(prime[i],t)*H(n/pt,i+1)+F(prime[i],t+1);
        }
        return res;
    }

    ans_t Solve(long long n,std::function<ans_t(int,int)>
    ↪   F,std::function<ans_t(long long)> gInit) {
        static long long kp[XN*2];
        int kpc=0;
        lim=sqrt(n)+0.5,psz=std::upper_bound(prime+1,prime+1+pcnt,lim)-prime;
        for(int i=id.cnt=0;i<=lim;++i)
            id.id[0][i]=id.id[1][i]=0;
        Min25::F=F;
        Min25::n=n;
        for(long long l=1,r;l<=n;l=r+1) {
            r=n/(n/l);
            g[id[kp[++kpc]=n/l]]=gInit(n/l);
        }
        for(int i=1;i<=psz;++i)
            fps[i]=fps[i-1]+F(prime[i],1);
        for(int j=1;j<=psz;++j)
            for(int i=1;i<=kpc && (long long)prime[j]*prime[j]<=kp[i];++i)
                g[id[kp[i]]]-=F(prime[j],1)*(g[id[kp[i]/prime[j]]]-fps[j-1]);
        return H(n,1);
    }
}
```

7.5 Polynomial

```
const int XN=1<<18;//Make2(n)*2?

namespace Polynomial {

    const int P=998244353;

    int Add(int x,int const &y) {
        return (x+=y)>=P?x-P:x;
    }

    int Minus(int x,int const &y) {
        return (x-=y)<0?x+P:x;
    }

    int Mul(long long x,int const &y) {
        return x*y%P;
    }

    int Pow(long long base,int v) {
```

```
    long long res;
    for(res=1;v;v>>=1,(base*=base)%=P)
        if(v&1)
            (res*=base)%=P;
    return res;
}

int Inverse(int x,int P=Polynomial::P) {
    return Pow(x,P-2);
}

int Make2(int x) {
    return 1<<((32-__builtin_clz(x))+((x&(-x))!=x));
}

void NTT(int a[],int n,int op) {
    for(int i=1,j=n>>1;i<n-1;++i) {
        if(i<j)
            std::swap(a[i],a[j]);
        int k=n>>1;
        while(k<=j) {
            j-=k;
            k>>=1;
        }
        j+=k;
    }
    for(int len=2;len<=n;len<<=1) {
        int rt=Pow(3,(P-1)/len);
        for(int i=0;i<n;i+=len) {
            int w=1;
            for(int j=i;j<i+len/2;++j) {
                int u=a[j],t=Mul(a[j+len/2],w);
                a[j]=Add(u,t),a[j+len/2]=Minus(u,t);
                w=Mul(w,rt);
            }
        }
    }
    if(op==-1) {
        std::reverse(a+1,a+n);
        int in=Inverse(n);
        for(int i=0;i<n;++i)
            a[i]=Mul(a[i],in);
    }
}

int Mul(int A[],int An,int B[],int Bn,int R[]) {
    static int a[XN],b[XN];
    int n=Make2(An+Bn-1);
    for(int i=0;i<n;++i) {
        a[i]=i<An?A[i]:0;
        b[i]=i<Bn?B[i]:0;
    }
    NTT(a,n,1);NTT(b,n,1);
    for(int i=0;i<n;++i)
        a[i]=Mul(a[i],b[i]);
    NTT(a,n,-1);
    std::copy(a,a+An+Bn-1,R);
    return An+Bn-1;
}

void Inverse(int A[],int An,int R[]) {
    int n=Make2(An);
    static int inv[XN],a[XN];
    inv[0]=Inverse(A[0]);
    for(int len=2;len<=n;len*=2) {
        for(int i=0;i<len*2;++i) {
            inv[i]=i<len/2?inv[i]:0;
            a[i]=i<std::min(An,len)?A[i]:0;
        }
        NTT(inv,len*2,1);NTT(a,len*2,1);
        for(int i=0;i<len*2;++i)
            inv[i]=Mul(inv[i],Minus(2,Mul(a[i],inv[i])));
        NTT(inv,len*2,-1);
    }
    std::copy(inv,inv+An,R);
}

void Differentiate(int A[],int n,int R[]) {
    for(int i=0;i<n-1;++i)
        R[i]=Mul(A[i+1],i+1);
    R[n-1]=0;
}

void Integrate(int A[],int n,int R[]) {
    for(int i=n-1;i>=1;--i)
        R[i]=Mul(A[i-1],Inverse(i));
    R[0]=0;
}

void SquareRoot(int A[],int An,int R[]) {
    int n=Make2(An);
    static int irt[XN],rt[XN],a[XN];
    assert(A[0]==1);rt[0]=1;
    int i2=Inverse(2);
    for(int len=2;len<=n;len*=2) {
        std::fill(rt+len/2,rt+len,0);
        Inverse(rt,len,irt);
        for(int i=0;i<len;++i) {
            a[i]=i<std::min(An,len)?A[i]:0;
            rt[i]=i<len/2?rt[i]:0;
            irt[i]=i<len/2?irt[i]:0;
        }
        NTT(irt,len,1);NTT(rt,len,1);NTT(a,len,1);
        for(int i=0;i<len;++i)
```

```
                rt[i]=Mul(i2,Add(Mul(a[i],irt[i]),rt[i]));
            NTT(rt,len,-1);
        }
        std::copy(rt,rt+An,R);
    }

    void Logarithm(int A[],int An,int R[]) {
        static int a[XN],b[XN];
        Differentiate(A,An,b);
        std::copy(A,A+An,a);
        Inverse(a,An,a);
        Mul(a,An,b,An,b);
        Integrate(b,An,b);
        std::copy(b,b+An,R);
    }

    void Exponent(int A[],int An,int R[]) {
        static int a[XN],b[XN],c[XN];
        int n=Make2(An);
        b[0]=1;
        for(int len=2;len<=n;len*=2) {
            std::fill(b+len/2,b+len,0);
            Logarithm(b,len,c);
            for(int i=0;i<len*2;++i) {
                a[i]=i<std::min(An,len)?A[i]:0;
                b[i]=i<len/2?b[i]:0;
                c[i]=i<len?c[i]:0;
            }
            NTT(a,len*2,1);NTT(b,len*2,1);NTT(c,len*2,1);
            for(int i=0;i<len*2;++i)
                b[i]=Mul(b[i],Add(Minus(1,c[i]),a[i]));
            NTT(b,len*2,-1);
        }
        std::copy(b,b+An,R);
    }

    void Pow(int A[],int An,int v,int R[]) {
        static int a[XN];
        int A0=A[0],i0=Inverse(A[0]);
        for(int i=0;i<An;++i)
            a[i]=Mul(i0,A[i]);
        Logarithm(a,An,a);
        for(int i=0;i<An;++i)
            a[i]=Mul(a[i],v);
        Exponent(a,An,a);
        int k=Pow(A0,v);
        for(int i=0;i<An;++i)
            R[i]=Mul(a[i],k);
    }
}
```

8 Uncategorized

8.1 Array Pointer

```
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 Shared Pointer

```
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 Mo Tree

```cpp
//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 Mo Sequence

```
//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]);
}
```