*************FFT+FWHT************

```
#define _USE_MATH_DEFINES
#define PI acos(-1)
#define int long long
#define MOD 1000000007
typedef vector<int>vi;
namespace FFT {
    typedef long long ll;
    typedef long double ld;
    struct base {
    typedef double T; T re, im;
    base() :re(0), im(0) {}
    base(T re) :re(re), im(0) {}
    base(T re, T im) :re(re), im(im) {}
    base operator + (const base& o) const{
    return base(re + o.re, im + o.im);}
    base operator - (const base& o) const{
    return base(re - o.re, im - o.im);}
    base operator * (const base& o) const{
    return base(re*o.re-im*o.im,re*o.im+im*o.re);}
    base operator * (ld k)const{return base(re*k,im*k);}
    base conj() const{ return base(re, -im);}
    };
    const int N= 20;/// log(actual size)+2
    const int MAXN= (1 << N)+5;
    base w[MAXN], f1[MAXN];
    int rev[MAXN];
    void build_rev(int k){
    static int rk= -1;
    if(k==rk)return; rk= k;
    int K= (1<<k);
    for(int i= 1; i<=K; i++){
        int j= rev[i - 1], t= k-1;
        while(t>=0 && ((j >> t)&1)){j^= 1 << t; --t;}
        if(t>=0){j^= 1 << t; --t;}rev[i]= j;
    }}
    void fft(base *a, int k){
    build_rev(k);
    int n= (1 << k);
    for(int i= 0; i<n; i++)if(rev[i]>i)swap(a[i], a[rev[i]]);
    for(int l= 2, ll= 1; l<= n; l+= l, ll+= ll){
        if(w[ll].re == 0 && w[ll].im == 0){
            ld angle= PI/ll;
            base ww(cosl(angle), sinl(angle));
            if(ll>1)for(int j= 0; j<ll; ++j){
                if(j&1)w[ll+j]= w[(ll+j)/2]*ww;
                else w[ll+j]= w[(ll+j)/2];
            }else w[ll] = base(1, 0);
        }
    for(int i= 0; i<n; i+= l)for(int j= 0; j < ll; j++){
        base v= a[i+j], u= a[i+j+ll]*w[ll+j];
        a[i+j]= v+u; a[i+j+ll]= v-u;

    }}}
   vi mul(const vi& a, const vi& b){
    int k= 1, ABsize= (int)(a.size())+(int)(b.size());
    while((1 << k) < ABsize) ++k;
    int n = (1 << k);
    for (int i = 0; i < n; i++) f1[i] = base(0, 0);
    int Asize=(int)(a.size());
    int Bsize=(int)(b.size());
    for(int i= 0; i<Asize; i++)f1[i]= f1[i]+base(a[i], 0);
    for(int i= 0; i<Bsize; i++)f1[i]= f1[i]+base(0, b[i]);
    fft(f1, k);
    for (int i = 0; i < 1 + n / 2; i++){
        base p= f1[i] + f1[(n - i) % n].conj();
        base _q= f1[(n - i) % n] - f1[i].conj();
        base q(_q.im, _q.re);
        f1[i] = (p * q) * 0.25;
        if (i > 0) f1[(n - i)] = f1[i].conj();
    }
    for(int i= 0; i<n; i++)f1[i]= f1[i].conj();
    fft(f1, k);vi r(ABsize);
    int Rsize=(int)(r.size());
    for(int i= 0; i<Rsize; i++)r[i]= round(f1[i].re/(double)n);
    /// IN case you only need distinct entities,
    /// NOT number of ways, uncomment the line below
    /// for(int i= 0; i<Rsize; i++)r[i]= min(r[i], 1ll);
    return r;}
}
inline int binpow(int a, int n){/// This is for FWHT
    int res= 1;
    while(n){
    if(n&1)res= 1LL*res*a % MOD;
    a= (1LL*a*a)%MOD;n>>= 1;
    }return res;
}
vi FWHT(vi x, bool inverse)
{
    for(int len= 1; 2*len<= x.size(); len<<= 1){
    for(int i= 0; i<x.size(); i+= 2 * len){
    for(int j= 0; j<len; j++){
    int u= x[i+j], v= x[i+len+j];
    x[i+j]= u+v;
    if(x[i+j]>=MOD)x[i+j]-= MOD;
    x[i+len+j]= u-v;
    if(x[i+len+j]<0)x[i+len+j]+= MOD;
    if(x[i+len+j]>=MOD)x[i+len+j]-= MOD;
    }}}
    if(inverse){
        int rev_n= binpow(x.size(), MOD-2);
        for(int i= 0; i<x.size(); i++)x[i]= 1LL*x[i]*rev_n % MOD;
    }return x;
}
int32_t main()
{
    /// FFT- all possible sum, FWHT- all possible Xorsum
    /// FWHT: Transform p, Transform q
    /// r= Multiplying p*q(point to point)
    /// inverse transform r
    return 0;
}
```

*******Linear Recurrence Solver(BM)*******

```
#define pb push_back
typedef long long ll;
#define SZ 233333
const int MOD=1e4+7;///or any prime
typedef vector<int>vi;
ll qp(ll a,ll b){
          ll x=1; a%=MOD;
          while(b){
    if(b&1)x=x*a%MOD;a=a*a%MOD; b>>=1;
    }return x;
}
namespace linear_seq{
inline vi BM(vi x){
    vi ls, rem;
    int lf, ld;
    for(int i=0;i<int(x.size());++i){
    ll t=0;
    for(int j=0;j<int(rem.size());++j)
    t=(t+x[i-j-1]*(ll)rem[j])%MOD;
    if((t-x[i])%MOD==0) continue;
    if(!rem.size()){
        rem.resize(i+1);
        lf=i; ld=(t-x[i])%MOD;
        continue;
    }
    ll k=-(x[i]-t)*qp(ld,MOD-2)%MOD;
    vi c(i-lf-1);c.pb(k);
    for(int j=0;j<int(ls.size());++j)c.pb(-ls[j]*k%MOD);
    if(c.size()<rem.size()) c.resize(rem.size());
    for(int j=0;j<int(rem.size());++j)
    c[j]=(c[j]+rem[j])%MOD;
    if(i-lf+(int)ls.size()>=(int)rem.size())
    ls=rem,lf=i,ld=(t-x[i])%MOD;
    rem=c;
    }
    for(int i=0;i<int(rem.size());++i)
    rem[i]=(rem[i]%MOD+MOD)%MOD;
    return rem;
}
int m;
ll a[SZ],h[SZ],t_[SZ],s[SZ],t[SZ];
inline void mul(ll*p,ll*q){
    for(int i=0;i<m+m;++i) t_[i]=0;
    for(int i=0;i<m;++i) if(p[i])
    for(int j=0;j<m;++j)
    t_[i+j]=(t_[i+j]+p[i]*q[j])%MOD;
    for(int i=m+m-1;i>=m;--i) if(t_[i])
    for(int j=m-1;~j;--j)
    t_[i-j-1]=(t_[i-j-1]+t_[i]*h[j])%MOD;
    for(int i=0;i<m;++i) p[i]=t_[i];
}
inline ll calc(ll K){
    for(int i=m;~i;--i)s[i]=t[i]=0;
    s[0]=1; if(m!=1) t[1]=1; else t[0]=h[0];
while(K){
        if(K&1) mul(s,t);
        mul(t,t); K>>=1;
    }
    ll su=0;
    for(int i=0;i<m;++i)su=(su+s[i]*a[i])%MOD;
    return (su%MOD+MOD)%MOD;
}
inline int work(vi x, ll n){
    if(n<int(x.size()))return x[n];
    vi v=BM(x);m=v.size();if(!m) return 0;
    for(int i=0;i<m;++i)h[i]=v[i],a[i]=x[i];
    return calc(n);
}};
int main(){
    Using namespace linear_seq;
    vi v;/// vector with some values
    int n;/// 0 - Indexed
    printf("%d\n", work(v, n));
         return 0;
}
```

*************Phi+PhiSum************

```
#define N 1000000
#define lg 25
int d[lg*N],nxt[lg*N],lst[N+2],phi[N+2];
void eulerTotient(){
    for(int i=2; i<=N; i++)phi[i]= i;
    for(int i=2; i<=N; i++)
    if(phi[i]==i){
    phi[i]--;
    for(int j=2*i; j<=N; j+=i)
    phi[j]/= i, phi[j]*= (i-1);
    }return;
}
long long phiSum[N+2];
void eulerTotientSum(){
    for(int i=2; i<=N; i++)lst[i]= i;
    for(int i=2, idx= N; i<=N; i++)
    for(int j=i; j<=N; j+=i){
    idx++;d[idx]= i;nxt[ lst[j] ]= idx;
    nxt[idx]= -1;lst[j]= idx;}
    phiSum[1]= 1;
    for(int j=2; j<=N; j++){
    phiSum[j]= (j*1ll*(j+1))/2;
    int now= nxt[j];
    while(now!=-1){
    int x= d[now];
    phiSum[j]-= phiSum[j/x]*1ll*x;
    now= nxt[now];
    }}return;
}
```

*******Geometry*******

```
#define ll long long
const double pi= 4*atan(1), eps= 1e-14;
inline int dcmp (double x){
if(fabs(x) < eps)return 0; else return x<0?-1:1;
}
double fix_acute(double th){
return th<-pi ?(th+2*pi):th>pi?(th-2*pi):th;
}
inline double getDistance(double x, double y){return
sqrt(x*x+y*y);}
inline double torad(double deg){return deg/180*pi;}
inline double toDeg(double rad){return (rad*180.0)/pi;}
struct Point{
double x, y;
Point (double x = 0, double y = 0): x(x), y(y) {}
bool operator == (const Point& u) const {
return dcmp(x - u.x) == 0 && dcmp(y - u.y)== 0;}
bool operator != (const Point& u)const{
return !(*this == u); }
bool operator < (const Point& u)const{
return dcmp(x - u.x)<0 || (dcmp(x-u.x)==0 && dcmp(y-u.y)<0);}
bool operator > (const Point& u)const { return u < *this; }
bool operator <= (const Point& u)const
{return *this < u || *this == u;}
bool operator >= (const Point& u)
const { return *this > u || *this == u; }
Point operator + (const Point& u){return Point(x+u.x, y+u.y);}
Point operator - (const Point& u){return Point(x-u.x, y-u.y);}
Point operator * (const double u){return Point(x*u, y*u);}
Point operator / (const double u){return Point(x/u, y/u);}
double operator * (const Point& u){return x*u.y-y*u.x;}
};
```

```
typedef Point Vector;
typedef vector<Point>Polygon;
struct Line{
double a, b, c;
Line(double a = 0, double b = 0, double c = 0): a(a), b(b), c(c){}
};
double modifiedatan2(Point a){
double ret= atan2(a.y, a.x);if(ret<0)ret+= 2*pi;return toDeg(ret);
}
double getDistance(Point a, Point b){
double x=a.x-b.x, y=a.y-b.y; return sqrt(x*x + y*y);
}
struct Segment{
Point a;Point b;Segment(){}
Segment(Point aa,Point bb) {a=aa,b=bb;}
};
struct DirLine{
Point p;Vector v;double ang;DirLine () {}
DirLine(Point p, Vector v): p(p), v(v){ang = atan2(v.y, v.x);}
bool operator < (const DirLine& u)const{return ang<u.ang;}
};
namespace Vectorial {
bool cmpX(const Point &a, const Point &b){return a.x<b.x;}
bool cmpY(const Point &a, const Point &b){return a.y<b.y;}
bool cmpAngle(const Point &a, const Point &b){
return modifiedatan2(a)<modifiedatan2(b);}
double getDot(Vector a, Vector b){return a.x*b.x+a.y*b.y;}
double getCross(Vector a, Vector b){return a.x*b.y-a.y*b.x;}
double getLength (Vector a){return sqrt(getDot(a, a)); }
double getPLength (Vector a){ return getDot(a, a); }
double getAngle (Vector u){ return atan2(u.y, u.x);}
double getSignedAngle(Vector a, Vector b){
return getAngle(b)-getAngle(a);
}
Vector rotate(Vector a, double rad){
return Vector(a.x*cos(rad)-a.y*sin(rad),
a.x*sin(rad)+a.y*cos(rad));
}
Vector ccw(Vector a, double co, double si){
return Vector(a.x*co-a.y*si, a.y*co+a.x*si);
}
Vector cw(Vector a, double co, double si){
return Vector(a.x*co+a.y*si, a.y*co-a.x*si);
}
Vector scale(Vector a, double s=1.0){return a/getLength(a)*s;}
Vector getNormal(Vector a){
double l= getLength(a);return Vector(-a.y/l, a.x/l);
}
};

namespace Linear{
using namespace Vectorial;
Line getLine(double x1, double y1, double x2, double y2){
return Line(y2-y1, x1-x2, y1*x2-x1*y2);
}
Line getLine (double a, double b, Point u){
return Line(a, -b, u.y * b - u.x * a);
}
bool getIntersection(Line p, Line q, Point& o){
if(fabs(p.a * q.b - q.a * p.b) < eps)return false;
o.x= (q.c*p.b-p.c*q.b)/(p.a*q.b-q.a*p.b);
o.y= (q.c*p.a-p.c*q.a)/(p.b*q.a-q.b*p.a);return true;
}
bool getIntersection(Point p,Vector v,Point q,Vector w,Point& o){
if(dcmp(getCross(v, w))==0)return false;Vector u= p-q;
double k= getCross(w, u)/getCross(v, w);o= p+v*k;return true;
}
double perpendicularProjection(Point p, Point a, Point b){
/// from point p to line(a, b)
Point edge= (b-a)/getLength(b-a);return getDot(edge, p-a);
}
double closestPairPoint(Point* P, int n){
typedef set<Point, bool(*)(const Point&, const Point&)>setType;
typedef setType::iterator setIT;setType s(&cmpY);
double ret= 1e20;sort(P, P+n, cmpX);
for(int i=1; i<n; i++)
if(P[i-1].x==P[i].x && P[i-1].y==P[i].y)return 0.0;
s.clear();for(int i=0; i<n; i++)s.insert(P[i]);
for(int i=0, idx=0; i<n; i++){
    Point it= P[i];
    while(it.x-P[idx].x>ret){s.erase(P[idx]);idx++;}
    Point low= Point(it.x, it.y-ret), high= Point(it.x, it.y+ret);
    setIT lowest= s.lower_bound(low);
    if(lowest!=s.end()){
    setIT highest= s.upper_bound(high);
    for(setIT now= lowest; now!= highest; now++){
    double cur= getDistance(*now, it);if(cur==0)continue;
    ret= min(ret, cur);
    }}s.insert(it);
    }return ret;
}
double getDistanceToLine(Point p, Point a, Point b){
return fabs(getCross(b-a, p-a)/getLength(b-a));}
double getDistanceToSegment(Point p, Point a, Point b){
    if(a==b)return getLength(p-a);
    Vector v1= b-a, v2= p-a, v3= p-b;
    if(dcmp(getDot(v1, v2))<0)return getLength(v2);
    else if(dcmp(getDot(v1, v3))>0)return getLength(v3);
    else return fabs(getCross(v1, v2)/getLength(v1));
}
double getDistanceSegToSeg(Point a,Point b,Point c,Point d){
    double Ans=min(getDistanceToSegment(a,c,d),
            getDistanceToSegment(b,c,d));
    return  Ans=min(Ans,getDistanceToSegment(c,a,b));
    Ans= min(Ans,getDistanceToSegment(d,a,b));
}
Point getPointToLine(Point p, Point a, Point b){
    Vector v= b-a; return a+v*(getDot(v, p-a)/getDot(v,v));
}
bool onSegment(Point p, Point a, Point b){
return !dcmp(getCross(a-p,b-p)) && dcmp(getDot(a-p,b-p))<=0;
}
bool haveIntersection(Point a1,Point a2,Point b1,Point b2){
    if(onSegment(a1,b1,b2))return true;
    if(onSegment(a2,b1,b2))return true;
    if(onSegment(b1,a1,a2))return true;
    if(onSegment(b2,a1,a2))return true;///Case of touching
    double c1=getCross(a2-a1,b1-a1),c2=getCross(a2-a1,b2-a1),
    c3=getCross(b2-b1, a1-b1),c4=getCross(b2-b1,a2-b1);
    return dcmp(c1)*dcmp(c2)<0 && dcmp(c3)*dcmp(c4)<0;
}
bool onLeft(DirLine l,Point p){return dcmp(l.v*(p-l.p))>=0;}
};
```

```
namespace Triangular{
   using namespace Vectorial;
   double getAngle (double a, double b, double c){
      return acos((a*a+b*b-c*c)/(2*a*b));
   }
   double getArea(double a, double b, double c){
      double s=(a+b+c)/2;return sqrt(s*(s-a)*(s-b)*(s-c));
   }
   double getArea(double a, double h){return a*h/2;}
   double getArea(Point a, Point b, Point c){
      return fabs(getCross(b-a, c-a))/2;
   }
   double getDirArea(Point a, Point b, Point c){
      return getCross(b-a, c-a)/2;
   }
   //ma/mb/mc = length of median from side a/b/c
   double getArea_(double ma,double mb,double mc){
      double s=(ma+mb+mc)/2;
      return 4/3.0*sqrt(s*(s-ma)*(s-mb)*(s-mc));
   }
   //ha/hb/hc = length of perpendicular from side a/b/c
   double get_Area(double ha,double hb,double hc){
      double H=(1/ha+1/hb+1/hc)/2;
      double A= 4*sqrt(H*(H-1/ha)*(H-1/hb)*(H-1/hc));
      return 1.0/A;
   }
   bool pointInTriangle(Point a, Point b, Point c, Point p){
      double s1=getArea(a,b,c),
      s2=getArea(p,b,c)+getArea(p,a,b)+getArea(p,c,a);
      return dcmp(s1 - s2) == 0;
   }
};

namespace Polygonal{
using namespace Linear;using namespace Triangular;
double getSignedArea(Point* p, int n){
   double ret = 0;
   for(int i=0; i<n-1; i++)
   ret+= (p[i]-p[0])*(p[i+1]-p[0]);return ret/2.0;
 }
long long pointsOnPolygon(Point* p, int n){
      long long ret= 0;for(int i=0; i<n; i++){
      Point a= p[(i+1)%n]-p[i];
      ll g= abs(__gcd((ll)a.x,(ll)a.y));ret+= g;
      }return ret;
   }
   int getConvexHull(Point* p, int n, Point* ch){
   sort(p, p + n);int m= 0;
for(int i=0; i<n; i++){
   while(m>1 &&
dcmp(getCross(ch[m-1]-ch[m-2],p[i]-ch[m-1]))<=0)
   m--;ch[m++] = p[i];
   }int k= m;
   for(int i= n-2; i>= 0; i--){
   while(m>k &&
dcmp(getCross(ch[m-1]-ch[m-2],p[i]-ch[m-2]))<=0)
   m--;ch[m++] = p[i];}
   if(n>1)m--;return m;
   }
```

```
double diameter(Point* p, int n, Point* ch){
   n= getConvexHull(p, n, ch);double ret= 0;
      for(int i=0, j=1; i<n; i++){
      if(i==j)j= (j+1)%n;
   while(getDistance(ch[i], ch[j])<getDistance(ch[i], ch[(j+1)%n]))
      j= (j+1)%n;ret= max(ret, getDistance(ch[i], ch[j]));
      }return ret;
   }
   Polygon maximumEnclosingTriangle(Point* p, int n)
   {
      Polygon ret;if(n<3)return ret;double res= 0.0;
      for(int i=0, j=1, k=2; i<n; i++){
      if(i==j)j= (j+1)%n;if(j==k)k= (k+1)%n;
      double area= getArea(p[i], p[j], p[k]);
      while(true){
      while(true){
      int nk= (k+1)%n;double narea= getArea(p[i], p[j], p[nk]);
      if(dcmp(narea-area)>=0)area= narea, k= nk;else break;
      }
      int nj= (j+1)%n;double narea= getArea(p[i], p[nj], p[k]);
      if(dcmp(narea-area)>=0)area= narea, j= nj;else break;
      }
      if(dcmp(area-res)>0)res= area, ret.clear(),
      ret.push_back(p[i]), ret.push_back(p[j]),
ret.push_back(p[k]);
      }return ret;
   }

pair<double,double>minimumEnclosingRectangle(Point *p,int
n,Point *ch){
   pair<double, double>ret= {1e9, 1e9};n= getConvexHull(p, n,
ch);
   if(n<3)return ret;for(int i=0; i<n; i++)p[i]= ch[i];
   int l=1, r=1, u=1;for(int i=0; i<n; i++){
   while(perpendicularProjection(p[(r+1)%n], p[i],
p[(i+1)%n])>perpendicularProjection(p[r%n], p[i],
p[(i+1)%n]))r++;
   while(u<r || getDistanceToLine(p[(u+1)%n], p[i],
p[(i+1)%n])>getDistanceToLine(p[u%n], p[i], p[(i+1)%n]))u++;
   while(l<u || perpendicularProjection(p[(l+1)%n], p[i],
p[(i+1)%n])<perpendicularProjection(p[l%n], p[i],
p[(i+1)%n]))l++;
   double w= perpendicularProjection(p[r%n], p[i],
p[(i+1)%n])-perpendicularProjection(p[l%n], p[i], p[(i+1)%n]);
   double h= getDistanceToLine(p[u%n], p[i], p[(i+1)%n]);
   ret.first= min(ret.first, w*h);
   ret.second= min(ret.second, 2.0*(w+h));
   }return ret;
}

void rotatingCalipers(Point *p, int n, vector<Segment>& sol) {
      sol.clear();int j = 1; p[n] = p[0];for (int i = 0; i < n; i++) {
      while (getCross(p[j+1]-p[i+1], p[i]-p[i+1]) >
getCross(p[j]-p[i+1], p[i]-p[i+1]))j = (j+1) % n;

sol.push_back(Segment(p[i],p[j]));sol.push_back(Segment(p[i +
1],p[j + 1]));
      }
   }
```

```
void rotatingCalipersGetRectangle(Point *p, int n, double& area,
double& perimeter){
        p[n] = p[0];int l = 1, r = 1, j = 1;area = perimeter = 1e20;
        for (int i=0; i<n; i++){
            Vector v = (p[i+1]-p[i]) / getLength(p[i+1]-p[i]);
            while (dcmp(getDot(v, p[r%n]-p[i]) - getDot(v,
p[(r+1)%n]-p[i])) < 0) r++;
            while (j < r || dcmp(getCross(v, p[j%n]-p[i]) -
getCross(v,p[(j+1)%n]-p[i])) < 0) j++;
            while (l < j || dcmp(getDot(v, p[l%n]-p[i]) - getDot(v,
p[(l+1)%n]-p[i])) > 0) l++;
            double w = getDot(v, p[r%n]-p[i])-getDot(v, p[l%n]-p[i]), h
= getDistanceToLine (p[j%n], p[i], p[i+1]);
            area = min(area, w * h);perimeter = min(perimeter, 2 * w
+ 2 * h);
        }
    }
    Polygon cutPolygon(Polygon u, Point a, Point b) {
        Polygon ret;int n = u.size();for (int i = 0; i < n; i++) {
            Point c = u[i], d = u[(i+1)%n];if (dcmp((b-a)*(c-a)) >= 0)
ret.push_back(c);
            if(dcmp((b-a)*(d-c))!= 0){Point t;getIntersection(a, b-a, c,
d-c, t);if(onSegment(t, c, d))ret.push_back(t);}
        }return ret;
    }
int halfPlaneIntersection(DirLine* li, int n, Point* poly) {
        sort(li, li + n);int first, last;
        Point* p = new Point[n];DirLine* q = new DirLine[n];
        q[first=last=0] = li[0];
        for(int i = 1; i < n; i++) {
            while (first < last && !onLeft(li[i], p[last-1])) last--;while
(first < last && !onLeft(li[i], p[first])) first++;
            q[++last] = li[i];
            if(dcmp(q[last].v * q[last-1].v) == 0) {last--;if
(onLeft(q[last], li[i].p)) q[last] = li[i];}
            if(first < last)getIntersection(q[last-1].p, q[last-1].v,
q[last].p, q[last].v, p[last-1]);
        }
        while (first < last && !onLeft(q[first], p[last-1])) last--;
        if (last - first <= 1) { delete [] p; delete [] q; return 0; }
        getIntersection(q[last].p, q[last].v, q[first].p, q[first].v,
p[last]);
        int m = 0;for (int i = first; i <= last; i++) poly[m++] =
p[i];delete [] p; delete [] q;
        return m;
    }
Polygon simplify (const Polygon& poly){
    Polygon ret;int n = poly.size();
    for(int i=0; i<n; i++){
        Point a = poly[i], b = poly[(i+1)%n], c = poly[(i+2)%n];
        if(dcmp((b-a)*(c-b)) != 0 && (ret.size() == 0 || b !=
ret[ret.size()-1]))ret.push_back(b);
        }return ret;
    }
Point ComputeCentroid(Point* p,int n){
        Point c(0, 0);double scale= 6.0*getSignedArea(p,n);
        for(int i=0; i<n; i++){
            int j=(i+1)%n;
            c=c+(p[i]+p[j])*(p[i].x*p[j].y-p[j].x*p[i].y);
        }return c/scale;
    }
    /// pt must be in ccw order with no three collinear points
    /// returns inside = 1, on = 0, outside = -1
    int pointInConvexPolygon(Point* pt,int n,Point p){
        assert(n>=3);int lo=1 , hi= n-1 ;
        while(hi-lo>1){
            int mid = (lo + hi) / 2;
            if(getCross(pt[mid]-pt[0],p-pt[0])>0)lo= mid;else hi=mid;
        }bool in = pointInTriangle(pt[0], pt[lo], pt[hi], p);
    if(!in) return -1;
    if(getCross(pt[lo] - pt[lo-1], p - pt[lo-1]) == 0) return 0;
    if(getCross(pt[hi] - pt[lo], p - pt[lo]) == 0) return 0;
    if(getCross(pt[hi] - pt[(hi+1)%n],p-pt[(hi+1)%n]) == 0)return 0;
        return 1;
    }
    /// Calculate [ACW, CW] tangent pair from an external point
    #define CW          -1
    #define ACW 1
    int direction(Point st, Point ed, Point q){
        return dcmp(getCross(ed-st, q-ed));
    }
    bool isGood(Point u, Point v, Point Q, int dir){
        return direction(Q, u, v)!= -dir;
    }
    Point better(Point u, Point v, Point Q, int dir){
        return direction(Q, u, v) == dir ? u : v;
    }
    Point tangents(Point* pt, Point Q, int dir, int lo, int hi){
        while(hi-lo>1){
            int mid= (lo+hi)/2;
            bool pvs= isGood(pt[mid], pt[mid - 1], Q, dir);
            bool nxt= isGood(pt[mid], pt[mid + 1], Q, dir);
            if(pvs && nxt) return pt[mid];
            if(!(pvs || nxt)){
                Point p1 = tangents(pt, Q, dir, mid+1, hi);
                Point p2 = tangents(pt, Q, dir, lo, mid - 1);
                return better(p1, p2, Q, dir);
            }
            if(!pvs){
                if(direction(Q, pt[mid], pt[lo])==dir)hi=mid-1;
                else if(better(pt[lo],pt[hi],Q,dir)==pt[lo])hi=mid-1;
                else lo = mid + 1;
            }
            if(!nxt){
                if(direction(Q, pt[mid], pt[lo]) == dir)  lo = mid + 1;
                else if(better(pt[lo], pt[hi], Q, dir) == pt[lo]) hi = mid - 1;
                else lo = mid + 1;
            }
        }
    Point ret=pt[lo];for(int
i=lo+1;i<=hi;i++)ret=better(ret,pt[i],Q,dir);
    return ret;
    }

    /// [ACW, CW] Tangent
    pair<Point, Point>get_tangents(Point* pt,int n,Point Q){
        Point acw_tan=tangents(pt, Q, ACW, 0, n-1);
        Point cw_tan=tangents(pt, Q, CW, 0, n-1);
        return make_pair(acw_tan, cw_tan);
    }
};
```

*******Suffix Array*******

```
#define mxn 1000006 /// Don't use N here as max size
#define ll long long
inline bool cmp(int *r, int a, int b, int l){return ((r[a]==r[b]) &&
(r[a+l]==r[b+l]));}
int wa[mxn], wb[mxn], wws[mxn], wv[mxn], rnk[mxn], lcp[mxn],
sa[mxn], dt[mxn], N;
/// ind - index of string, pos - position in that string
int ind[mxn], pos[mxn], sparse[20][mxn];
void DA(int *r, int *sa, int n, int m){
    int i, j, p, *x=wa, *y=wb, *t;
    for(i=0; i<m; i++) wws[i]=0;
    for(i=0; i<n; i++) wws[x[i]=r[i]]++;
    for(i=1; i<m; i++) wws[i]+=wws[i-1];
    for(i=n-1; i>=0; i--) sa[--wws[x[i]]]=i;
    for(j=1, p=1; p<n; j*=2, m=p){
        for(p=0, i=n-j; i<n; i++) y[p++]=i;
        for(i=0; i<n; i++)if(sa[i]>=j) y[p++]=sa[i]-j;
        for(i=0; i<n; i++)wv[i]=x[y[i]];
        for(i=0; i<m; i++)wws[i]=0;
        for(i=0; i<n; i++)wws[wv[i]]++;
        for(i=1; i<m; i++)wws[i]+=wws[i-1];
        for(i=n-1; i>=0; i--)sa[--wws[wv[i]]]=y[i];
        for(t=x, x=y, y=t, p=1, x[sa[0]]= 0, i=1; i<n; i++)
        x[sa[i]]= cmp(y, sa[i-1], sa[i], j)?p-1:p++;
    }return;
}
void cal_lcp(int *r, int *sa, int n){
    int i, j, k= 0;
    for(i=1; i<=n; i++) rnk[sa[i]]=i;
    for(i=0; i<n; lcp[rnk[i++]]=k)
    for(k?k--:0, j=sa[rnk[i]-1]; r[i+k]==r[j+k]; k++);
    return;
}
void suffix_array(char *A){
    for(int i= 0; i<=128; i++)
    wa[i]= wb[i]= wws[i]= wv[i]= rnk[i]= lcp[i]= sa[i]= dt[i]= 0;
    for(int i= 0; i<=N; i++){
    wa[i] = wb[i] = wws[i] = wv[i] = rnk[i] = lcp[i] = sa[i] = dt[i] = 0;
        if(i<N)dt[i]= A[i] ;
    }DA(dt, sa, N+1, 128);cal_lcp(dt, sa, N);
    for(int i=0; i<N; i++){
        /// transforming it into 0-based SA
        sa[i]= sa[i+1];lcp[i]= lcp[i+2];
        sparse[0][i]= lcp[i];/// throw away if not needed
        rnk[i]--;
    }
    for(int j=1; j<20; j++)
    for(int i=0; i+(1<<(j))<N; i++)
    sparse[j][i]= min(sparse[j-1][i], sparse[j-1][ i+(1<<(j-1)) ]);
    return;
}
int bsl(int i, int mid){
    for(int j=19; j>=0; j--)
    if(i-(1<<(j))>=0 && sparse[j][ i-(1<<(j)) ]>=mid)i-= (1<<(j));
    return i;
}
int bsr(int i, int mid){
    for(int j=19; j>=0; j--)
    if(i+(1<<(j))<N && sparse[j][i]>=mid)i+= (1<<(j));
    return i;
}
int find_lcp(int l, int r){
    int mn= N;
    for(int j=19; j>=0; j--)
    if(l+(1<<(j))<=r)mn= min(mn, sparse[j][l]), l+= (1<<(j));
    return mn;
}
char str[mxn], s[mxn];
int main(){
    int n;
    scanf("%s", str);N= strlen(str);
    for(int i=0; i<N; i++)ind[i]= 0, pos[i]= i;
    str[N]= '#', ind[N]= -1, N++;
    scanf("%d", &n);
    for(int i=1; i<=n; i++){
        scanf("%s", s);
        for(int j=0; s[j]; j++)
        str[N]= s[j], ind[N]= i, pos[N]= j, N++;
        str[N]= '#', ind[N]= -1, N++;
    }
    suffix_array(str);
    return 0;
}
```

******KMP*****

```
int fail[N];char str[N];
void kmp(int len){
    int now= -1;fail[0]= -1;
    for(int i=1; i<len; i++){
        while(now!=-1 && str[now+1]!=str[i])now= fail[now];
        if(str[now+1]==str[i])fail[i]= ++now;
        else fail[i]= now= -1;
    }return;
}/// Period= len-fail[n-1]-1, iff Period divides len
```

*****Aho Corasic******

```
int par[mxn], child[mxn][26], fail[mxn], now[55], sz[55], len[55],
val[mxn], cnt;
char txt[55][22];
void insrt(int n){
    cnt= 0;
    memset(now, 0, sizeof now);
    memset(len, 0, sizeof len);
    memset(child[0], -1, sizeof child[0]);
    queue<int>q;
    for(int i=0; i<n; i++){
        int l= txt[i][0]-'a';
        if(child[0][l]==-1){
        ++cnt;
        memset(child[cnt], -1, sizeof child[cnt]);
        child[0][l]= cnt;
        }
        now[i]= child[0][l];len[i]++;
        if(len[i]!=sz[i])q.push(i);
        else val[ now[i] ]= 1;
    }
    while(!q.empty()){
    int i= q.front();q.pop();
    int l= txt[i][ len[i] ]-'a';
    if(child[ now[i] ][l]==-1){
        ++cnt;
        memset(child[cnt], -1, sizeof child[cnt]);
        child[ now[i] ][l]= cnt;
        par[cnt]= now[i];
    }
    now[i]= child[ now[i] ][l];
    int x= fail[ par[ now[i] ] ];
    while(x && child[x][l]==-1)x= fail[x];
    if(child[x][l]!=-1)
    fail[ now[i] ]= child[x][l];
    else fail[ now[i] ]= 0;
    len[i]++;
    if(len[i]!=sz[i])q.push(i);
    else val[ now[i] ]= 1;
    }return;
}
void func(){/// call after marking endings
/// iteration can change, depends on how we mark
/// i.e. fail->me OR me->fail
    for(int i=1; i<=cnt; i++)val[i]|= val[ fail[i] ];return;
}
int traverse(int nw, int l){
/// traversing through the Aho-Corasic tree, l is letter
    if(child[nw][l]==-1){
        while(nw && child[nw][l]==-1)nw= fail[nw];
        if(child[nw][l]!=-1)nw= child[nw][l];
    }else nw= child[nw][l];return nw;
}
int main(){
    int n;scanf("%d", &n);
    for(int i=0; i<n; i++)
    scanf("%s", txt[i]), sz[i]= strlen(txt[i]);
    insrt(n);func();
    return 0;
}
```

******Palindromic Tree*******

```
#define mxn 100005
int child[mxn][26], len[mxn], fail[mxn], cnt, now;
char str[mxn];
void init(){
    cnt= now= 2;
    memset(child[1], -1, sizeof child[1]);
    memset(child[2], -1, sizeof child[2]);
    len[1]= -1, len[2]= 0;fail[1]= 1, fail[2]= 1;
    return;
}
void insrt(int p){ /// Insert ith character
    while(str[ p-len[now]-1 ]!=str[p])now= fail[now];
    int x= fail[now];
    while(str[ p-len[x]-1 ]!=str[p])x= fail[x];
    if(child[now][ str[p]-'a' ]==-1){
        child[now][ str[p]-'a' ]= ++cnt;
        memset(child[cnt], -1, sizeof child[cnt]);
        len[cnt]= len[now]+2;
        if(len[cnt]==1)fail[cnt]= 2;
        else fail[cnt]= child[x][ str[p]-'a' ];
    }now= child[now][ str[p]-'a' ];return;
}
int main(){
    scanf("%s", str+1);
    int res= 0, n= strlen(str+1);init();
    /// len[now] is longest palindrome ending at i
    for(int i=1; i<=n; i++)
    insrt(i), res= max(res, len[now]);
    /// (cnt-2) is total distinct palindrome
    return 0;
}
```

******SOS DP*****

```
for(int i=0; i<n; i++){
    for(int mask= 0; mask<(1<<(n)); mask++)/// sub->super
    if(mask&(1<<(i)))freq[mask]+= freq[mask^(1<<(i))];
    for(int mask= (1<<(n))-1; mask>0; mask--)/// super->sub
    if(mask&(1<<(i)))freq[mask^(1<<(i))]+= freq[mask];
}
```

*****Convex Hull Trick*****

```
#define ll long long
ll ara[200005];
vector<ll>m, b;
bool bad(int f1, int f2, int f3){
    return
(1.0*(b[f3]-b[f1])*(m[f1]-m[f2])>=1.0*(b[f2]-b[f1])*(m[f1]-m[f3]));
}
void add(ll _m, ll _b){
    m.push_back(_m);b.push_back(_b);int sz= m.size();
    while(sz>=3 && bad(sz-3, sz-2, sz-1)){
        ll t= m.back();m.pop_back();m.pop_back();m.push_back(t);
        t= b.back();b.pop_back();b.pop_back();b.push_back(t);
        sz--;
    }return;
}
ll eval(int i, ll x){return (m[i]*x + b[i]);}
ll query(ll x){
    int lo= 0, hi= m.size()-1;ll mx= LLONG_MIN;
    while(lo+5<hi){
        int m1= (lo+hi)/2;int m2= m1+1;
        ll y1= eval(m1, x);ll y2= eval(m2, x);
        if(y1>=y2)mx= y1, hi= m1;
        else mx= y2, lo= m2;
    }for(int i=lo; i<=hi; i++)mx= max(mx, eval(i, x));
    return mx;
}
```

******Mat Expo******

```
/// Set Identity & Base matrix before calling bigmat
ll a[N][N], b[N][N], temp[N][N], mat[N][N], id[N][N];
void mul(int n){
    for(int i=0; i<n; i++)for(int j=0; j<n; j++)temp[i][j]= 0;
    for(int i=0; i<2; i++)for(int j=0; j<2; j++)
    for(int k=0; k<2; k++)
    temp[i][j]+= a[i][k]*b[k][j], temp[i][j]%= mod;
    return;
}
void bigmat(ll p, int n){
    if(!p){
        for(int i=0; i<n; i++)for(int j=0; j<n; j++)
        temp[i][j]= id[i][j];
        return;
    }
    bigmat(p/2, n);
    for(int i=0; i<n; i++)for(int j=0; j<n; j++)
    a[i][j]= temp[i][j], b[i][j]= temp[i][j];
    mul(n);
    if(p&1ll){
        for(int i=0; i<n; i++)for(int j=0; j<n; j++)
        a[i][j]= temp[i][j], b[i][j]= mat[i][j];
        mul(n);
    }return;
}
```

******MAXFLOW(DINIC)******

```
#define N 250
struct Edge{int v, flow, C, rev;};
vector<Edge>adj[N];
int level[N], start[N], V;/// V - Total
nodes
void addEdge(int u, int v, int C){
    Edge a{v, 0, C, adj[v].size()};
    Edge b{u, 0, 0, adj[u].size()};

adj[u].push_back(a);adj[v].push_back(
b);
}
bool BFS(int s, int t){
    for(int i=0 ; i<V ; i++)level[i]=
-1*(i!=s);
    queue<int>q;q.push(s);
    while(!q.empty()){
        int u= q.front();q.pop();
        for(int i=0; i<adj[u].size(); i++){
            Edge e= adj[u][i];
            if(level[e.v]<0  && e.flow<e.C)
            level[e.v]= level[u]+1,
q.push(e.v);
        }
    }return !(level[t]<0);
}
int sendFlow(int u, int flow, int t){
    if(u==t)return flow;
    for(int i=start[u]; i<adj[u].size(); i++){
        Edge &e= adj[u][i];
        if(level[e.v]==level[u]+1 &&
e.flow<e.C){
            int curr_flow= min(flow,
e.C-e.flow);
            int temp_flow= sendFlow(e.v,
curr_flow, t);
            if(temp_flow>0){
                e.flow+= temp_flow;
                adj[e.v][e.rev].flow-=
temp_flow;
                return temp_flow;
            }
        }start[u]++;
    }return 0;
}
int DinicMaxflow(int s, int t){///source,
sink
    if(s==t)return -1;/// Invalid
    int total= 0;
    while(bool x= BFS(s, t) == true){
        memset(start, 0, sizeof start);
        while(int f= sendFlow(s,
INT_MAX, t))total += f;
    }return total;
}
```

*****MATCHING(Kuhn)*****

```
#define N 1003
int match[N], vis[N];
bool kuhn(int x, int c)
{   /// using c to check visited or not
    if(vis[x]==c)return 0;
    vis[x]= c;
    for(auto y:adj[x])
    if(match[y]==-1 || kuhn(match[y], c)){
        match[y]= x;
        return 1;
    }return 0;
```

*****MATCHING(HopCroft)******

```
#define N 100005
int n, rght[N], lft[N], vis[N], lvl[N];
vector<int>adj[N];
bool dfs(int x){
    vis[x]= 1;
    for(auto y:adj[x])
    if(lft[y]==-1 || (!vis[lft[y]] &&
    lvl[lft[y]]>lvl[x] && dfs(lft[y]))){
    rght[x]= y;lft[y]= x;return 1;
    }return 0;
}
int hopcroft(){
    memset(lft, -1, sizeof lft);
    memset(rght, -1, sizeof rght);
    int ret= 0;
    while(true){
        queue<int>q;memset(lvl,-1, sizeof
lvl);
        for(int i=1; i<=n; i++){
            if(rght[i]==-1)lvl[i]= 0, q.push(i);
            else lvl[i]= -1;
        }
        while(q.size()){
            int x= q.front(); q.pop();
            for(auto y:adj[x])
            if(lft[y]!=-1 && lvl[lft[y]]==-1)
            q.push(lft[y]), lvl[lft[y]]= lvl[x]+1;
        }
        memset(vis, 0, sizeof vis);
        int sum = 0;
        for(int i=1;i<=n;i++)
        if(rght[i]==-1 && dfs(i))sum++;
        if(!sum)break;
        ret+= sum;
    }for(int i=0; i<=n+2; i++)adj[i].clear();
    return ret;
}
```

*****Min Cost Max Flow*****

```
#define mxn 202
const int inf= 1e8;
typedef pair<int, int> pi;
struct edge{
    int from, to, flow, cst;edge(){};
    edge(int fr, int t, int fl, int c){
    from= fr, to= t, flow= fl, cst= c;}
};
vector<edge>e;
vector<int>adj[mxn];
int dis[mxn], par[mxn];
void addEdge(int v, int u, int f, int c){
    adj[v].push_back(e.size());
    e.push_back(edge(v, u, f, c));
    adj[u].push_back(e.size());
    e.push_back(edge(u, v, 0, -c));
    return;
}
void shortest_paths(int n, int s){
    bool inq[mxn];/// only for SPFA
    for(int i=0; i<n; i++)
    dis[i]=inf,par[i]= -1,inq[i]=0;dis[s]= 0;
    /// Bellman Ford
    /*for(int i=1; i<n; i++)
    for(int j=0; j<e.size(); j++)
    if(e[j].flow>0 &&
dis[e[j].to]>dis[e[j].from]+e[j].cst)
    dis[e[j].to]=
dis[e[j].from]+e[j].cst,par[e[j].to]= j;*/
    /// Dijkstra
    /*priority_queue<pi>pq;pq.push({0,
s});
    while(!pq.empty()){
    pi p= pq.top();pq.pop();
    if(-p.first!=dis[p.second])continue;
    int v= p.second;
    for(int i=0; i<adj[v].size(); i++){
        int id= adj[v][i];/// id of v->u edge
        int u= e[id].to;
        if(e[id].flow>0 &&
dis[u]>dis[v]+e[id].cst){
            par[u]= id;
            dis[u]= dis[v]+e[id].cst;
            pq.push({-dis[u], u});}
        }
    }*/
    /// SPFA
    queue<int>q;q.push(s);
    while(!q.empty()){
    int v= q.front();q.pop();
    inq[v]= 0;
    for(int i=0; i<adj[v].size(); i++){
        int id= adj[v][i];/// id of v->u edge
        int u= e[id].to;
        if(e[id].flow>0 &&
dis[u]>dis[v]+e[id].cst){
            par[u]= id;
            dis[u]= dis[v]+e[id].cst;
            if(!inq[u])inq[u]= 1, q.push(u);}
        }}return;
}
int min_cost_flow(int n, int need, int s,
int t){
    /// n nodes, need flows, source s,
target t
    int flow= 0, ret= 0;
    while(flow<need){
        shortest_paths(n, s);
        if(dis[t]==inf)return -1;
        int f= need-flow, cur= t;
        while(cur!=s){
            f= min(f, e[ par[cur] ].flow);
            cur= e[ par[cur] ].from;
        }
        flow+= f;ret+= f*dis[t];cur= t;
        while(cur!=s){
            e[ par[cur] ].flow-= f;
            e[ par[cur]^1 ].flow+= f;
            cur= e[ par[cur] ].from;
        }
    }if(flow<need)return -1;
    else return ret;}
```

Leading m digits of a*b*c*d*e*…..

ans= log(a)+log(b)+log(c)+log(d)+log(e)+...

ans= (ans-floor(val))*pow(10, m-1)

Derangement: D(n)= n*D(n-1)+(-1)^n

Burnside lemma/Group Theory:

```
/// Coloring n beads with k color
for(int i=0; i<n; i++){/// i rotations
    int g= __gcd(n, i);res+= bigmod(k, g);res%= mod;
}res= (res*bigmod(n, mod-2))%mod;/// then divide res by n
```

Binomial Coefficient property:

Sum of the squares: (nC0)^2+(nC1)^2+⋯+(nCn)^2=(2n)Cn

Weighted sum: 1(nC1)+2(nC2)+⋯+n(nCn)=n*2^(n−1)

Connection with the Fibonacci numbers:

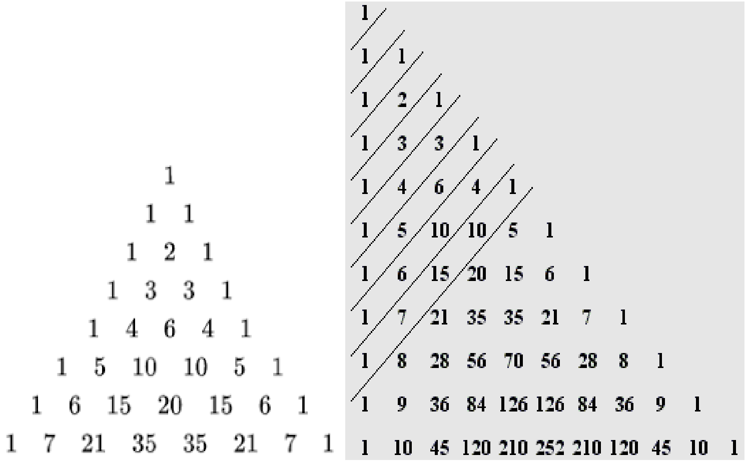
(nC0)+(n−1C1)+⋯+(n−kCk)+⋯+(0Cn)=Fib(n+1)

Sum over n and k: for(k=0; k<=m; k++)∑((n+k)Ck)=(n+m+1)Cm

---

n= m1+m2+m3+...+mk

Multinomial Coefficient: n!/(m1! * m2! * m3! * …. * mk!)



Number of spanning tree in Bipartite Graph:

G(X, Y)= X^(Y-1) * Y^(X-1)

Where X is number of nodes in 1st set, Y is in 2nd set.

---

2 SAT(Satisfiability):

Mark X as even & !X as odd

Add edge for each (X V Y) as: (!X->Y) and (!Y->X)

Do topological sort & create SCC

If X and !X id in same component then impossible to satisfy

Else keep X if (component[X]>component[!X])

Random Prime:

1500450271, 3267000013, 4093082899, 3628273133, 2860486313, 3367900313

Fast Headers:

```
#pragma GCC optimize("Ofast")
#pragma GCC
target("sse,sse2,sse3,ssse3,sse4,popcnt,abm,mmx,avx,tune=native") ///com error
#pragma GCC optimize("unroll-loops")
```

Algebra Formula: GCD((x^a)-1, (x^b)-1)= (x^GCD(a, b))-1

A-dominating Sequence:

n As, m Bs(n>=m)

Number of Dominating seq: (n+m)Cn - (n+m)C(n+1)

Catalan Number:

When in A-dominating seq n=m

(2n)Cn - (2n)C(n+1)

1,1,2,5,14,42,132,429,1430,4862,16796,58786,208012,742900

Urns & Balls/Stars & Bars:

Number of ways to put m balls in n urns: (n+m-1)C(m-1)

Random Function:

```
mt19937
rng(chrono::steady_clock::now().time_since_epoch().count());
/// for int64, use mt19937_64
int rand_func(int l, int r){
return uniform_int_distribution<int>(l, r) (rng);}
```

Shank's baby step giant step:

```
/// Given b, x and mod, find p such that b^p % mod = x;
/// where b and mod are relatively prime
/// Solution: Find p as (p*m + q), using divide and conquer
unordered_map<ll, ll>mp
int main(){
    ll b, x, m= 10001, mul= 1ll;/// square root of mod
    scanf("%lld %lld", &b, &x);
    for(ll i=0; i<m; i++){
        if(!mp[mul])mp[mul]= i+1;
        mul*= b;
        mul%= mod;
    }
    ll mm= 1ll;
    for(ll i=0; i<=m+1; i++){
        ll p= bigmod(mm, mod-2);p= (x*p)%mod;
        ll q= mp[p];
        if(q){
        p= i;q--;
        /// check whether result needs to be maximum or minimum
        /// and if it needs to be inside a certain given range or not !!
        cout<<(p*m + q)<<endl;/// minimum
        break;}mm*= mul;mm%= mod;
    }return 0;
}
```

---

n= p1^a1 * p2^a2 * p3^a3 * ….. * pk^ak

Number Of Divisor:

(a1+1)*(a2+1)*(a3+1)*....(ak+1)

Sum Of Divisor:

(p1^0+p1^1+...+p1^a1)*(p2^0+p2^1++...+p2^a2)*.....
*(pk^0+pk^1+...+pk^ak) **=**
(p1^(a1+1)-1)/(p1-1) * (p2^(a2+1)-1)/(p2-1) *....
*(pk^(ak+1)-1)/(pk-1)

Fibonacci Matrix:

{Fn, F(n-1)} = {{1, 1},{1, 0}}^(n-1) * {F1,F0 }

Persistent Segment Tree:

```
const int M = MM;
int  a[M], root[M];
int avail;
struct node{
   int l, r, val;
   node(){
      l = r = val = 0;
   }
} sum[M*40];
int update(int PreNode, int l, int r, int L, int
val){
   int NewNode = ++avail;
   if(l==r){
      sum[NewNode].val = val;
      ///sum[NewNode].val =
sum[PreNode].val + val;
      return NewNode;
   }
   int mid = (l+r)/2;
   if(L<=mid){
      sum[NewNode].r = sum[PreNode].r;
      sum[NewNode].l =
update(sum[PreNode].l,l,mid,L,val);
   }
   else{
      sum[NewNode].l = sum[PreNode].l;
      sum[NewNode].r =
update(sum[PreNode].r,mid+1,r,L,val);
   }
   sum[NewNode].val =
sum[sum[NewNode].l].val +
sum[sum[NewNode].r].val;
   return NewNode;
}
ll query(int n, int l, int r, int L, int R){
   if(l>R || r<L)return 0;
   if(l>=L && r<=R)return sum[n].val;
   int mid = (l+r)/2;

   int tot = query(sum[n].l,l,mid,L,R) +
query(sum[n].r,mid+1,r,L,R);
   return tot;
}
ll query(int n, int l, int r, int k){
   if(l==r) l;
   int mid = (l+r)/2;
   if(sum[sum[n].l].val>=k){
      return query(sum[n].l,l,mid,k);
   }
   else {
      return
query(sum[n].r,mid+1,r,k-sum[sum[n].l].val)
;
   }
}
```

LCA :

```
vi adj[MM];
int par[MM], sp[MM][22],  lvl[MM];
void dfs(int p, int x, int lev){
   par[x] = p;
   lvl[x] = lev;
   for(auto y:adj[x])if(p!=y){
      dfs(x, y, lev+1);
   }
}
void build(int n){
   for(int i=1;i<=n;i++){
      sp[i][0] = par[i];
   }
   for(int i=1;(1<<i)<=n ;i++){
      for(int j=1;j<=n;j++){
         sp[j][i] = sp[sp[j][i-1]][i-1];
      }
   }
}
int lca_of(int u, int v){
   if(lvl[u]>lvl[v])swap(u, v);
   for(int i=17;i>=0;i--){
      if(!sp[v][i])continue;
      if(lvl[sp[v][i]]>=lvl[u]){
         v = sp[v][i];
      }
   }
   if(u==v)return u;
   for(int i=17;i>=0;i--){
      if(sp[u][i]==sp[v][i])continue;
      u = sp[u][i]; v = sp[v][i];
   }
   return par[u];
}

ll distance_of(int u, int v){
   int lca = lca_of(u, v);
   return lvl[u] + lvl[v] - 2*lvl[lca];
}

int kth_parent_of(int v, int k){
   for(int i=17;i>=0;i--){
      if(!sp[v][i])continue;
      if((1<<i) <= k){
         v = sp[v][i];
         k -= (1<<i);
      }
   }
   return v;
}
```

Template: /*

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

using namespace std;

#define    PF(a)     printf("%d\n",(a))
#define    PFL(a)    printf("%lld\n",(a))
#define    SF(a)     scanf("%d",&a)
#define    SF2(a,b)  scanf("%d %d",&a, &b)
#define    SFL(a)    scanf("%lld",&a)
#define    SFL2(a,b) scanf("%lld %lld",&a,
&b)
#define    gc()      getchar()
#define    pb        push_back
#define    pc()      printf("Case %d: ",tt++)
#define    tc()      cout<<"Case "<<tt++<<":
"
#define    dbg(x)    cout << #x << " -> " <<
x << endl;
#define    MAX       2134567891
#define    MOD       1000000007
#define    MM        200005
#define    mem(a)    memset((a),0,sizeof
(a))
#define    SET(a)    memset((a),-1,sizeof
(a))
#define    output
freopen("output.txt","w",stdout);
#define    input
freopen("input.txt","r",stdin);
#define    I_O
ios_base::sync_with_stdio(0);
cin.tie(0);cout.tie(0)
#define    rep(a)    for(int i=0;i<(a);i++)
#define    REP(a)    for(int j=0;j<(a);j++)

mt19937
rng(chrono::steady_clock::now().time_sinc
e_epoch().count());

typedef long long ll;
typedef unsigned long long llu;
typedef pair < int , int > pi;
typedef pair < int , pi > pii;
typedef vector < int > vi; */
ll bigmod(ll a, ll b, ll c){
   if(b==0)return 1%c;ll
x=bigmod(a,b/2,c);x=(x*x)%c;
   if(b%2==1)x=(x*a)%c;return x;
}

ll poww(ll a, ll b){
   if(b==0)return 1;ll
x=poww(a,b/2);x=x*x;if(b%2==1)x=(x*a);re
turn x;
}
ll mod_inverse(ll a, ll mod){return
bigmod(a,mod-2,mod);}
ll LCM(ll a, ll b){ return a*b/ __gcd(a,b);}
int pr = 50000;
vi primelist;
bool a[MM*100];
void seive( ){
   int i,j,k=sqrt(pr);
   a[1]=1;
   primelist.pb(2);
   for(i=4;i<=pr;i+=2)a[i]=1;

for(i=3;i<=k;i+=2)if(!a[i])for(j=i*i;j<=pr;j+=2*i
)a[j]=1;
   for(i=3;i<=pr;i+=2)if(!a[i])primelist.pb(i);
}
int phi[MM];
void calculatePhi() {
  for (int i = 1; i < M; i++) {
    phi[i] = i;
  }
  for (int p = 2; p < M; p++) {
    if (phi[p] == p) { // p is a prime
      for (int k = p; k < M; k += p) {
        phi[k] -= phi[k] / p;
      }
    }
  }
}

ll fact_divs( ll n, ll p){
    ll cnt=0;while(p<=n){cnt += n/p;n /=
p;}return cnt;
}
int Set(int N,int pos){return N=N |
(1<<pos);}
int reset(int N,int pos){return N= N &
~(1<<pos);}
bool check(int N,int pos){return (bool)(N &
(1<<pos));}
```

Automata:

```
const int M = MM;
struct state{
    int len, link,fpos;
    bool isclone;
    map < char , int > next;
    vector < int > inv_link;
} st[2*M];
int sz, last;
int cnt[2*M];
bool terminal[2*M];
void initialize(){
    rep(sz+1){
        st[i].next.clear();
        st[i].inv_link.clear();
    }
    sz = last = 0; st[0].len = 0; st[0].link = -1;
}
void build_automata(char c){
    int cur = ++sz;
    cnt[cur] = 1;
    st[cur].len = st[last].len + 1;
    st[cur].isclone = 0;
    st[cur].fpos = st[cur].len;
    int p;
    for(p=last;p!=-1 && !st[p].next[c];
p=st[p].link){
        st[p].next[c] = cur;
    }
    if(p==-1){
        st[cur].link = 0;
    }
    else{
        int q = st[p].next[c];
        if(st[p].len+1==st[q].len){
            st[cur].link = q;
        }
        else{
            int clone = ++sz;
            cnt[clone] = 0;
            st[clone] = st[q];
            st[clone].len = st[p].len+1;
            st[clone].isclone = 1;
            for(;p!=-1 &&
st[p].next[c]==q;p=st[p].link){
                st[p].next[c] = clone;
            }
            st[q].link = st[cur].link = clone;
        }
    }
    last = cur;
}

void cal_terminal(){
    int now = last;
    while(now!=-1){
        terminal[now] = 1;
        now = st[now].link;
    }
}
char s[M];
vi v[M];
void cal_occuarence(int len){
    for(int i = sz;i>=1;i--){
        v[st[i].len].pb(i);
    }
    for(int i = len;i>=1;i--){
        for(auto x : v[i]){
            if(st[x].link==-1)continue;
            cnt[st[x].link] += cnt[x];
        }
        v[i].clear();
    }
}
ll max_match(int len){
    int now = 0;
    int mx = 0;
    rep(len){
        now = st[now].next[s[i]-'a'];
        if( terminal[now] && cnt[now]>=3 ){
            mx = max(mx,i+1);
        }
    }
    return mx;
}
void cal_inv_link(){
    for(int i=1;i<=sz;i++){
        st[st[i].link].inv_link.pb(i);
    }
}
vi res;
///here, now = last_pos of pattern;
void find_occurences(int now){
    if(!st[now].isclone)res.pb(st[now].fpos);
    for(auto x : st[now].inv_link){
        find_occurences(x);
    }
}
ll dp[M*2];
ll dist_sub(int n){
    if(dp[n]!=-1)return dp[n];
    ll tot = 1;
    for(auto it =
st[n].next.begin();it!=st[n].next.end();it++){
        if(it->second==0)continue;
        tot += dist_sub(it->second);
    }
    dp[n] = tot;
    return dp[n];
}
string lcs (string p) {
    int v = 0, l = 0, best = 0, bestpos = 0;
    for (int i = 0; i < p.size(); i++) {
        while (v && !st[v].next.count(p[i]-'a')) {
            v = st[v].link ;
            l = st[v].len ;
        }
        if (st[v].next.count(p[i]-'a')) {
            v = st[v].next[p[i]-'a'];
            l++;
        }
        if (l > best) {
            best = l;
            bestpos = i;
        }
    }
    return p.substr(bestpos - best + 1, best);
}

int lcp[M];
void cal_lcp_array(string s){
    int n = s.size();
    initialize();
    for(int i=n-1;i>=0;i--){
        build_automata(s[i]);
        lcp[i+1] = st[st[last].link].len;
    }
}
int main() {
    sf("%s",s);
    initialize();
    int len = strlen(s);
    rep(len)build_automata(s[i]-'a');
    cal_inv_link();
    find_occurences(st[0].next[0]);
    for(auto x:res)cout<<x<<endl;
    return 0;
}
```

BIT:

```
int tree[2*MM],n;
void update(ll idx, ll val){
    while(idx && idx <= n){
        tree[idx] += val;
        idx += idx & (-idx);
    }
}
ll query( ll idx ){
    ll sum = 0;
    while( idx > 0){
        sum += tree[idx];
        idx -= idx & (-idx);
    }
    return sum;
}
```

2D BIT (PBDS):

```
OST bit[N];
void insert(int x, int y){
        for(int i = x; i < N; i += i & -i){
                bit[i].insert(mp(y, x));
        }
}
void remove(int x, int y){
        for(int i = x; i < N; i += i & -i){
                bit[i].erase(mp(y, x));
        }
}
int query(int x, int y){
        int ans = 0;
        for(int i = x; i > 0; i -= i & -i){
                ans += bit[i].order_of_key(mp(y+1, 0));
        }
        return ans;
}
```

SOS DP: (submask)

```
  SET(f); int N = 22;
  for(int i = 0; i<n; ++i){
    f[a[i]] = a[i];
  }
  for(int i = 0; i < N; ++i) for(int mask = 0; mask < (1<<N); ++mask){
    if((mask & (1<<i)) && f[mask^(1<<i)]!=-1){
      f[mask] = f[mask^(1<<i)];
    }
  }
```

SOS DP: (supermask)

```
  int N = 20;
  for(int i=(1<<N)-1;i>=0;i--){
    f[i] = frq[i];
  }
  for(int i = 0; i < N; ++i) for(int mask = (1<<N)-1; mask >=0; --mask){
    if(!(bool)(mask & (1<<i))){
      f[mask] += f[mask^(1<<i)];
    }
  }
```

Sibling dp:

```
const int M = 205;
vi adj[M];
int cost[M][M],n,m;
int child[M], rt[M];
ll dp[M][M];
void find_sibling(int p, int x){
  bool flg = 0;
  int pre;
  for(auto y:adj[x]){
    if(y==p)continue;
    if(!flg){
      child[x] = y;
      flg = 1;
    }
    else{
      rt[pre] = y;
    }
    pre = y;
    find_sibling(x,y);
  }
}
ll dpcall(int p, int x, int k){
  if(x==-1)return 0;
  ll &ret = dp[x][k];
  if(~ret)return ret;
  ret = MAX;
  /* //node er moddhe achi...ekhn ami chinta korbo, ei node k ami koto diye nibo... */
  ll res = 1 + dpcall(x,child[x],m);
  ll ress = dpcall(p,rt[x],k);
  ret = min(ret, res+ress);
  ll rest = k - cost[p][x];
  for(int i=0; i<=rest; i++){
    ret = min(ret, dpcall(x,child[x],i) + dpcall(p,rt[x], rest-i));
  }
  return ret;

}
int main() {
  I_O;
  int t, tt=1,u,v,c;
  cin>>t;
  while(t--){
    SET(dp); SET(child); SET(rt); mem(cost);
    cin>>n>>m;
    rep(n-1){
      cin>>u>>v>>c;
      adj[u].pb(v); adj[v].pb(u);
      cost[u][v] = cost[v][u] = c;
    }
    cost[0][1] = cost[1][0] = 101;
    find_sibling(0,1);
    ll res = dpcall(0,1,0);
    tc();
    cout<<res<<endl;
    rep(n+2)adj[i].clear();
  }
  return 0;
}
```

Convex Hull Tricks:

```
const int M = 100005;
bool Q;
struct Line {
        mutable ll k, m, p;
        bool operator<(const Line& o) const {
                return Q ? p < o.p : k < o.k;
        }
};

struct LineContainer : multiset<Line> {
        const ll inf = LLONG_MAX;
        ll div(ll a, ll b) {
                return a / b - ((a ^ b) < 0 && a % b); }
        bool isect(iterator x, iterator y) {
                if (y == end()) { x->p = inf; return false; }
                if (x->k == y->k) x->p = x->m > y->m ? inf : -inf;
                else x->p = div(y->m - x->m, x->k - y->k);
                return x->p >= y->p;
        }
        void add(ll k, ll m) {
/*              auto z = insert({k, m, 0}), y = z++, x = y; /// for max query */
                auto z = insert({-k, -m, 0}), y = z++, x = y;/* // for min query*/
                while (isect(y, z)) z = erase(z);
                if (x != begin() && isect(--x, y)) isect(x, y = erase(y));
                while ((y = x) != begin() && (--x)->p >= y->p){
                        isect(x, erase(y));
                }
        }
        ll query(ll x) {
/*              if(empty()) return -1e18; /// for max query */
                if(empty()) return 1e18; /// for min query
                Q = 1; auto l = *lower_bound({0,0,x}); Q = 0;
/*              return (l.k * x + l.m); /// for max query */
                return -(l.k * x + l.m); /// for min query
        }
} lc;
```

Divide and Conquer Optimization:

```
const int M = 4005;
ll dp[2][M];
int a[M][M];
int sum[M][M];
ll cost_fun(int l, int r){
  return (sum[r][r] - sum[l-1][r] - sum[r][l-1] + sum[l-1][l-1])/2;
}
void dpcall(int xr, int l, int r, int optl, int optr){
  if (l > r) return;
  int mid = (l + r)/2;
  pair<ll, int> best = {1e18, -1};
  for (int k = optl; k <= min(mid,optr); k++) {
    best = min(best, {dp[xr^1][k-1] + cost_fun(k,mid), k});
  }
  dp[xr][mid] = best.first;
  int opt = best.second;
  dpcall(xr, l, mid - 1, optl, opt);
  dpcall(xr, mid + 1, r, opt, optr);
}
int main() {

  int n,k;
  SF2(n,k);
  for(ll i=1; i<=n; i++){
    for(int j=1;j<=n;j++){
      SF(a[i][j]);
      sum[i][j] = sum[i-1][j] + sum[i][j-1] - sum[i-1][j-1] + a[i][j];
    }
  }
  for(int i=1; i<=n; i++) dp[0][i] = 1e18;
  for(int i=1; i<=k; i++){
    dpcall(i%2, 1, n, 1, n);
  }
  PFL(dp[k%2][n]);
  return 0;
}
```

DSU on tree:

```
const int M = 1000005;
int sz[M], bigone[M], big[M], lvl[M];
vector < int > adj[M];
void find_size(int p, int x, int lv) {
    sz[x] = 1;
    lvl[x] = lv;
    int mx = -1;
    bigone[x] = -1;

    for(auto y : adj[x]){
        if(y == p) continue;

        find_size(x, y, lv+1);

        sz[x] += sz[y];
        if(sz[y] > mx){
            bigone[x] = y;
            mx = sz[y];
        }
    }
}
int mx, depth, frq[M], res[M];
void add(int p, int x, int val){
    if(val == 1){
        frq[lvl[x]]++;
        int now = frq[lvl[x]];
        if(now > mx){
            mx = now;
            depth = lvl[x];
        }
        else if(now == mx){
            depth = min(depth, lvl[x]);
        }
    }
    else {
        frq[lvl[x]]--;
    }
    for(auto y : adj[x]){
        if(y == p || big[y]) continue;
        add(x, y, val);
    }
}
void dfs(int p, int x, int keep){
    for(auto y : adj[x]){
        if(y == p || y == bigone[x]) continue;
        dfs(x, y, 0);
    }
    if(bigone[x] != -1){
        dfs(x, bigone[x], 1);
        big[bigone[x]] = 1;
    }
    add(p, x, 1);
    //result part
    res[x] = depth - lvl[x];
    if(bigone[x] != -1){
        big[bigone[x]] = 0; /*/// jodi keep == 0 hoy taile se bigchild soho sobai k muche dibe eksathe */
    }
    if(keep == 0) {
        add(p, x, -1);
        mx = depth = 0;
    }
}
```

Dynamic Connectivity:

```
const int M = 300005;
map < pi, int > mp;
vector < int >  qr;
vector < pi > vec[M*4];
int par[M], dsz[M], comp, res[M], vis[M];
stack < int > stq;
void init(int n){
    for(int i=1; i<=n; i++){
        par[i] = i;
        dsz[i] = 1;
    }
    comp = n;
    while(stq.size()) stq.pop();
}
ll unfn(int u){
    while(par[u] != u) u = par[u];
    return u;
}
void mergee(int x, int y){
    int u = unfn(x);
    int v = unfn(y);
    if( u == v ) return;
    --comp;
    if(dsz[u] > dsz[v]) swap(u, v);
    stq.push(u);
    dsz[v] += dsz[u];
    par[u] = v;
}
void rollback(int cur){
    while(stq.size() > cur){
        int u = stq.top(); stq.pop();
        dsz[par[u]] -= dsz[u];
        par[u] = u;
        ++comp;
    }
}
void update(int l, int r, int k, int L, int R, pi &p){
    if(l > R || r < L) return;
    if(l>=L && r<=R) {
        vec[k].pb(p);
        return;
    }
    int mid = (l + r)/2;
    update(l, mid, k*2, L, R, p);
    update(mid+1, r, k*2+1, L, R, p);
}
void dfs(int l, int r, int k){
    int sz = stq.size();
    for(auto x : vec[k]){
        mergee(x.first, x.second);
    }
    if(l == r){
        if(vis[l]) cout<<comp<<endl;
        rollback(sz);
        return;
    }
    int mid = (l + r)/2;
    dfs(l, mid, k*2);
    dfs(mid+1, r, k*2+1);
    rollback(sz);
}
int main(){
    I_O;
    int n, m;
    cin>>n>>m;
    init(n);
    for(int i=1; i<=m; i++){
        string ch;
        cin>>ch;
        if(ch == "?"){
            vis[i] = 1;
            continue;
        }
        int u, v; cin>>u>>v;
        if(u > v) swap(u, v);
        pi p = {u, v};
        if(ch == "+"){
            mp[p] = i;
        }
        else {
            update(1, m, 1, mp[p], i, p);
            mp.erase(p);
        }
    }
    for(auto x:mp){
        pi p = x.first;
        update(1, m, 1, x.second, m, p);
    }
    dfs(1, max(1, m), 1);
          return 0;
}
```

Dominator Tree:

```
const int M = 200005;
vector < pi > adj[M];
vector < int > dag[M], parent[M], dtree[M], toporder;
int subsz[M] ,sp[M][22],  lvl[M], vis[M];
ll cost[M];
void initialize(int n){
    for(int i=0; i<=n; i++){
        adj[i].clear();
        dag[i].clear();
        parent[i].clear();
        dtree[i].clear();
        vis[i] = lvl[i] = subsz[i] = 0;
        for(int j=0; j<=18; j++){
            sp[i][j] = 0;
        }
    }

    toporder.clear();
}
void dijkstra(int n, int src){
    for(int i=1; i<=n; i++) cost[i] = 1e18;
    cost[src] = 0;

    multiset < pair < ll, ll > > min_heap;
    min_heap.insert({cost[src], src});

    while(min_heap.size()){
        pair < ll, ll > p = *min_heap.begin();
        min_heap.erase(min_heap.find(p));

        int cur_node = p.second;
        ll dist = p.first;

        if(dist > cost[cur_node])continue;
        for(auto x : adj[cur_node]){
            int next_node = x.first;
            ll weight = x.second;

            if(cost[next_node] > cost[cur_node]
+ weight){
                cost[next_node] =
cost[cur_node] + weight;

min_heap.insert({cost[next_node],
next_node});
            }
        }
    }
}
void build_dag(int n){
    for(int i=1; i<=n; i++){
        if(cost[i] == 1e18) continue;
        for(auto x : adj[i]){
            int y = x.first;
            ll w = x.second;
            if((cost[i] + w) == cost[y]){
                dag[i].pb(y);
                parent[y].pb(i);
            }
        }
    }
}
void toposort(int x){
    vis[x] = 1;
    for(auto y : dag[x]) if(!vis[y]) toposort(y);
    toporder.pb(x);
}
void build_lca(int n, int p, int x){
    sp[x][0] = p;
    lvl[x] = lvl[p] + 1;
    for(int i=1; (1<<i)<=n; i++){
        sp[x][i] = sp[sp[x][i-1]][i-1];
    }
}
int lca_of(int u, int v){
    if(lvl[u] > lvl[v])swap(u, v);
    for(int i=17; i>=0; i--){
        if(!sp[v][i])continue;
        if(lvl[sp[v][i]] >= lvl[u]){
            v = sp[v][i];
        }
    }
    if(u==v)return u;
    for(int i=17;i>=0;i--){
        if(sp[u][i]==sp[v][i])continue;
        u = sp[u][i]; v = sp[v][i];
    }
    return sp[u][0];
}
```

```
void build_dominator_tree(int n){
    reverse(toporder.begin(),
toporder.end());
    for(auto x : toporder){
        int lca = -1;
        for(auto y : parent[x]){
            if(lca == -1){
                lca = y;
            }
            else {
                lca = lca_of(lca, y);
            }
        }
        if(lca == -1) continue;
        build_lca(n, lca, x);
        dtree[lca].pb(x);
    }
}
void dfs(int x){
    subsz[x] = 1;
    for(auto y : dtree[x]){
        dfs(y);
        subsz[x] += subsz[y];
    }
}
int main() {
    I_O;
    ll n, m, s;
    cin>>n>>m>>s;
    for(int i=1; i<=m; i++){
        int u, v, w;
        cin>>u>>v>>w;
        adj[u].pb({v, w});
        adj[v].pb({u, w});
    }
    dijkstra(n, s);
    build_dag(n);
    toposort(s);
    build_dominator_tree(n);
    dfs(s);
    ll res = 1;
    n = toporder.size();
    int ans = 0;
    for(auto x : dtree[s]){
        ans = max(ans, subsz[x]);

    }
    cout<<ans<<endl;

    return 0;
}
```

Articulation Point and Bridge:

```
const int M = MM;
vi adj[M];
int dtime[M], low[M], par[M], cnt;
int root, child;
int artpoint[M];

void articulate(int x){
            low[x] = dtime[x] = ++cnt;
            for(auto y:adj[x]){
                        if(!dtime[y]){
                                    par[y] = x;
                                    if(x == root)
child++;
                                    articulate(y);
                                    if(low[y] >=
dtime[x]) artpoint[x] = true;
                                    if(low[y] >
dtime[x]){

cout<<"Edge "<<x<<" & "<<y<<" is a
bridge."<<endl;
                                    }
                                    low[x]=min(low[x],
low[y]);
                        }
                        else if (y != par[x]){
                                    low[x] =
min(low[x], dtime[y]);
                        }
            }
}
int main(){
    I_O;
            int n, m, u, v;
            cin>>n>>m;
            for (int i=0; i<m; i++){
                        cin>>u>>v;
                        adj[u].pb(v);
                        adj[v].pb(u);
            }
            cnt=0;
            mem(dtime);
            mem(artpoint);
            for (int i=1; i<=n; i++){
                        if (!dtime[i]){
                                    root = i;
                                    child = 0;
                                    articulate(i);
                                    artpoint[root] =
(child > 1);
                        }
            }
            printf("Articulation points:\n");
            for (int i=1; i<=n; i++){
                        if (artpoint[i])
                                    cout<<"Vertex:
"<<i<<endl;
            }
            return 0;
}
```

SQRT decom:

```
struct node{
    ll l,r,id,sq;
} d[50010];
bool cmp( node a, node b ){
    if(a.sq==b.sq){
        if(a.sq&1) return a.r<b.r;
        else return b.r<a.r;
    }
    return a.sq<b.sq;
}
ll a[MM],freq[MM],sum[MM],res[MM];
int main(){
    int t, tt=1;
    SF(t);
    while(t--){
        int n,c,q;
        mem(sum);mem(freq);
        SF3(n,c,q);
        rep(n)SF(a[i+1]);
        ll sqr = sqrt(n);
        rep(q){
            SFL2(d[i].l,d[i].r);
            d[i].id = i;
            d[i].sq = d[i].l/sqr;
        }
        sort(d,d+q,cmp);
        ll rt = 0,lf = 1, mx = 0;
        rep(q){
            while(lf>d[i].l){
                lf--;
                sum[freq[a[lf]]]--;
                freq[a[lf]]++;
                sum[freq[a[lf]]]++;
                mx = max(mx,freq[a[lf]]);
            }
            while(rt<d[i].r){
                rt++;
                sum[freq[a[rt]]]--;
                freq[a[rt]]++;
                sum[freq[a[rt]]]++;
                mx = max(mx,freq[a[rt]]);
            }
            while(lf<d[i].l){
                sum[freq[a[lf]]]--;
                freq[a[lf]]--;
                sum[freq[a[lf]]]++;
                if(!sum[mx]) mx--;
                lf++;
            }
            while(rt>d[i].r){
                sum[freq[a[rt]]]--;
                freq[a[rt]]--;
                sum[freq[a[rt]]]++;
                if(!sum[mx]) mx--;
                rt--;
            }
            res[d[i].id] = mx;
        }
        pf("Case %d:\n",tt++);
        rep(q){
            PFL(res[i]);
        }
    }
    return 0;
}
```

MO’s & DSU rollback:

```
const int M = MM;
int par[M], sz[M], comp;
stack <int> stq;
inline void init(int n){
    for(int i=1; i<=n; i++){
        par[i] = i;
        sz[i] = i;
    }
    comp = n;
    while(stq.size()) stq.pop();
}
inline int unfn(int x, int flg) {
    if(par[x] == x) return x;
    if(flg) return unfn(par[x], flg);
    else return par[x] = unfn(par[x], flg);
}
inline void merge(int u, int v, int flg) {
    u = unfn(u, flg);
    v = unfn(v, flg);
    if(u == v) return;
    --comp;
    if(sz[u] > sz[v]) swap(u, v);
    par[u] = v;
    sz[v] += sz[u];
    if(flg) stq.push(u);
}
inline void rollback(int cur) {
    while(stq.size() > cur) {
        int u = stq.top(), v = par[u]; stq.pop();
        sz[v] -= sz[u];
        ++comp;
        par[u] = u;
    }
}
struct node{
    int l,r,id,sq;
} d[M];
inline bool cmp( node a, node b ){
    if(a.sq==b.sq){
        return (a.r<b.r);
    }
    return a.sq<b.sq;
}
int a[M],vis[M], res[M];
struct nodee{
    int u, v;
    nodee(){
        u = v = 0;
    }
} edge[M];
int main(){
    int t, tt=1;
    SF(t);
    while(t--){
        int n, m, q;
        SF3(n, m, q);
        for(int i=1; i<=m; i++){
            int u, v; SF2(u, v);
            edge[i].u = u;
            edge[i].v = v;
        }
        int sqr = 300;
        rep(q){
            SF2(d[i].l, d[i].r);
            d[i].id = i;
            d[i].sq = d[i].l/sqr;
        }
        sort(d,d+q,cmp);
        int rt = 0,lf = 1;
        int pre = -1;
        rep(q){
            int block = d[i].sq;
            if(block != pre){
                init(n);
                pre = block;
                rt = d[i].l;
            }
            while(rt<d[i].r){
                rt++;
                int cblock = rt / sqr;
                if(cblock <= block) continue;
                merge(edge[rt].u, edge[rt].v, 0);
            }
            int nxt = min(d[i].r, sqr*(block+1) - 1);

            int cur = stq.size();
            for(int j=d[i].l; j<=nxt; j++){
                merge(edge[j].u, edge[j].v, 1);
            }

            res[d[i].id] = comp;
            rollback(cur);
        }
        rep(q){
            PF(res[i]);
        }
    }
    return 0;
}
```

MO’s on tree:

```
const int M = MM;
vi adj[M];
int par[M], sp[M][22],  lvl[M], st[M], ed[M], id, a[2*M], b[M];
void dfs(int p, int x, int lev){
    st[x] = ++id;
    a[id] = x;
    par[x] = p;
    lvl[x] = lev;
    for(auto y:adj[x])if(p!=y){
        dfs(x, y, lev+1);
    }
    ed[x] = ++id;
    a[id] = x;
}
//LCA part
struct node{
    int l,r,id,sq,lca;
} d[M];
bool cmp( node a, node b ){
    if(a.sq==b.sq){
        if(a.sq&1) return a.r<b.r;
        else return b.r<a.r;
    }
    return a.sq<b.sq;
}
int frq[M], vis[M], res[M];
vector < ll > vec;
int main(){
    int n, m; SF2(n, m);
    int cnt = 0;
    for(int i=1;i<=n;i++){
        SF(b[i]);
        vec.pb(b[i]);
    }
    vec.pb(1e15);
    sort(vec.begin(), vec.end());
    for(int i=1;i<=n;i++){
        b[i] =
upper_bound(vec.begin(),vec.end(), b[i]) -
vec.begin();
    }
    rep(n-1){
        int u, v; SF2(u, v);
        adj[u].pb(v);
        adj[v].pb(u);
    }
    dfs(0, 1, 1);
    build(n);
    int sqr = sqrt(id);
    rep(m){
        SF2(d[i].l, d[i].r);
        if(st[d[i].l] > st[d[i].r]){
            swap(d[i].l, d[i].r);
        }
        d[i].id = i;
        int lcaa = lca_of(d[i].l, d[i].r);
        if(lcaa==d[i].l){
            d[i].l = st[d[i].l];
            d[i].r = st[d[i].r];
            d[i].lca = -1;
        }
        else {
            d[i].l = ed[d[i].l];
            d[i].r = st[d[i].r];
            d[i].lca = lcaa;
        }
        d[i].sq = d[i].l/sqr;

    }
    sort(d,d+m,cmp);
    ll rt = 0,lf = 1, ans = 0;
    rep(m){
        while(lf>d[i].l){
            lf--;
            if(vis[a[lf]]&1){
                vis[a[lf]]++;
                frq[b[a[lf]]]--;
                if(!frq[b[a[lf]]])ans--;
            }
            else {
                vis[a[lf]]++;
                frq[b[a[lf]]]++;
                if(frq[b[a[lf]]]==1)ans++;
            }
        }

        while(rt<d[i].r){
            rt++;
            if(vis[a[rt]]&1){
                vis[a[rt]]++;
                frq[b[a[rt]]]--;
                if(!frq[b[a[rt]]])ans--;
            }
            else {
                vis[a[rt]]++;
                frq[b[a[rt]]]++;
                if(frq[b[a[rt]]]==1)ans++;
            }
        }

        while(lf<d[i].l){
            if(vis[a[lf]]&1){
                vis[a[lf]]--;
                frq[b[a[lf]]]--;
                if(!frq[b[a[lf]]])ans--;
            }
            else {
                vis[a[lf]]--;
                frq[b[a[lf]]]++;
                if(frq[b[a[lf]]]==1)ans++;
            }
            lf++;
        }

        while(rt>d[i].r){
            if(vis[a[rt]]&1){
                vis[a[rt]]--;
                frq[b[a[rt]]]--;
                if(!frq[b[a[rt]]])ans--;
            }
            else {
                vis[a[rt]]--;
                frq[b[a[rt]]]++;
                if(frq[b[a[rt]]]==1)ans++;
            }
            rt--;
        }
        res[d[i].id] = ans;
        if(d[i].lca!=-1){
            int val = b[d[i].lca];
            if(!frq[val]) res[d[i].id]++;
        }
    }
    rep(m)PF(res[i]);

    return 0;
}
```

Centroid Decom:

```
const int M = 200005;
vector < int > adj[M];
int sbz[M], parent[M], vis[M], a[M];
void find_size(int p, int x){
    sbz[x] = 1;
    int sz = adj[x].size();
    for(int i = 0; i < sz; i++){
        int y = adj[x][i];
        if(p == y || vis[y])continue;
        find_size(x, y);
        sbz[x] += sbz[y];
    }
}
ll find_center(int p, int x, int l){
    int sz = adj[x].size();
    for(int i = 0; i < sz; i++) {
        int y = adj[x][i];
        if(y==p || vis[y])continue;
        if(sbz[y] > l/2) {
            return find_center(x,y,l);
        }
    }
    return x;
}
int frq[1<<21];
ll res[M];
void add(int p, int x, int mask, int val){
    frq[mask] += val;
    for(auto y : adj[x]){
        if(y == p || vis[y]) continue;
        add(x, y, mask ^ a[y], val);
    }
}
ll xtra;
ll dfs(int p, int x, int mask, int cmask){
    ll total = frq[mask ^ cmask];
    if(__builtin_popcount(mask) <= 1){
total++; xtra++;}
    for(int i = 0; i < 20; i++){
        int nmask = mask ^ (1<<i);
        total += frq[nmask ^ cmask];
    }
    for(auto y : adj[x]){
        if(y == p || vis[y]) continue;
        total += dfs(x, y, mask ^ a[y], cmask);
    }
    res[x] += total;
    return total;
}
void cal(int x){
    ll total = 0;
    for(auto y : adj[x]){
        if( vis[y] ) continue;
        add(x, y, a[y] ^ a[x], 1);
    }
    for(auto y : adj[x]){
        if( vis[y] ) continue;
        add(x, y, a[y] ^ a[x], -1);
        total += dfs(x, y, a[y] ^ a[x], a[x]);
        add(x, y, a[y] ^ a[x], 1);
    }
    for(auto y : adj[x]){
        if( vis[y] ) continue;
        add(x, y, a[y] ^ a[x], -1);
    }
    res[x] += (total + xtra) / 2;
    xtra = 0 ;
}
void decompose(int p, int x){
    find_size(p,x);
    x = find_center(p, x, sbz[x]);
    vis[x] = 1;
    cal( x );
    int sz = adj[x].size();
    for(int i = 0; i < sz; i++){
        int y = adj[x][i];
        if( vis[y] ) continue;
        decompose(x, y);
    }
}
string s;
int main() {
    I_O;
    int n, m;
    cin>>n;
    for(int i = 1; i < n; i++){
        int u, v; cin>>u>>v;
        adj[u].pb(v);
        adj[v].pb(u);
    }
    cin>>s;
    for(int i = 1; i <= n; i++) a[i] =
(1LL<<((ll)s[i-1] - 'a'));
    decompose(1, 1);
    for(int i = 1; i <= n; i++){
        cout<<res[i] + 1<<" ";
    }
    cout<<endl;
    return 0;
}
```

Segtree Polynomial Update:(Sup)

```
ll sum[M*4], prop[M*4], cnt[M*4], a[M];
void build(int l, int r, int k){
    if(l==r){
        sum[k] = a[l];
        return;
    }
    int mid = (l+r)/2;
    build(l, mid, k*2);
    build(mid+1, r, k*2+1);
    sum[k] = sum[k*2] + sum[k*2+1];
}
void propagate(int l, int r, int k){
    if(!cnt[k])return;
    if(l!=r){
        int mid = (l+r)/2;
        prop[k*2] += prop[k];
        prop[k*2+1] += prop[k] +
cnt[k]*((mid-l+1));
        cnt[k*2] += cnt[k];
        cnt[k*2+1] += cnt[k];
    }
    ll len = (r-l+1);
    sum[k] += prop[k]*len +
((len*(len+1LL))/2LL) * cnt[k];
    prop[k] = cnt[k] = 0;
}
void update(int l, int r, int k, int L, int R){
    propagate(l, r, k);
    if(l>R || r<L)return;
    if(l>=L && r<=R){
        prop[k] = l-L;
        cnt[k] = 1;
        propagate(l, r, k);
        return;
    }
    int mid = (l+r)/2;
    update(l, mid, k*2, L, R);
    update(mid+1, r, k*2+1, L, R);
    sum[k] = sum[k*2] + sum[k*2+1];
}
```

Strongly Connected Component

```
vi adj[M],adjj[M];
int vis[M],par[M];
stack < int > st;
void toposort(int x){
    vis[x] = 1;
    for(auto y : adj[x]){
        if(vis[y])continue;
        toposort(y);
    }
    st.push(x);
}
void scc(int x, int cmp){
    vis[x] = 1; par[x] = cmp;
    for(auto y : adjj[x]){
        if(vis[y])continue;
        scc(y,cmp);
    }
}
vi nadj[500];
int main() {
    I_O;
    int t,tt=1;
    cin>>t;
    while(t--){
        int n,m,x,y;
        cin>>n>>m;
        rep(m){
            cin>>x>>y;
            adj[x].pb(y);
            adjj[y].pb(x);
        }
        mem(vis);
        for(int i=1;i<=n;i++){
            if(vis[i])continue;
            toposort(i);
        }
        mem(vis);
        int cnt = 1;
        while(st.size()){
            x = st.top();st.pop();
            if(vis[x])continue;
            scc(x,cnt);
            cnt++;
        }
        for(int i=1;i<=n;i++){
            for(auto x:adj[i]){
                if(par[i]==par[x])continue;
                nadj[par[i]].pb(par[x]);
            }
        }
        for(int i=1;i<cnt;i++){
            cout<<i<<" -> ";
            for(auto x:nadj[i]){
                cout<<x<<" ";
            }
            cout<<endl;
        }
        for(int i=1;i<=n;i++){
            adj[i].clear(); adjj[i].clear();
nadj[i].clear();
        }
    }
    return 0;
}
```

Centroid (Supliment):

```
int msl[M][5], msr[M][5];
int nnode, lt[M],rt[M];
void make_binary(int p, int x, int l, int r)
{
    if(r-l+1==0)return;
    if(r-l+1==1){
        parent[adjj[p][r]] = x; lt[x] = adjj[p][r];
    }
    else if(r-l+1==2){
        parent[adjj[p][l]] = x; lt[x] = adjj[p][l];
        parent[adjj[p][r]] = x; rt[x] = adjj[p][r];
    }
    else if(r-l+1==3){
        ++nnode; lvl[nnode] = lvl[x];
        if(a[x] == 1) a[nnode] = 1;
        parent[nnode] = x; lt[x] = nnode;
        make_binary(p,nnode,l,l+1);
        parent[adjj[p][r]] = x; rt[x] = adjj[p][r];
    }
    else{
        int mid = (l+r)/2; ++nnode;
        lvl[nnode] = lvl[x]; parent[nnode] = x;
        if(a[x] == 1) a[nnode] = 1; lt[x] =
nnode;
        make_binary(p,nnode,l,mid);
++nnode;
        lvl[nnode] = lvl[x]; if(a[x] == 1)
a[nnode] = 1;
        parent[nnode] = x; rt[x] = nnode;
        make_binary(p,nnode,mid+1,r);
    }
}
void lift_to_binary(int x){
    int sz = adjj[x].size();
    make_binary(x,x,0,sz-1);
    for(auto y :adjj[x]){
        lift_to_binary(y);
    }
}
void dfs(int p, int x, int lev, int side){
    if(x==0)return;
    if(side==1){
        if(mp[lev][x] <= 2 && a[x] == 0) {
            msl[p][mp[lev][x]]++;
        }
    }
    else{
        if(mp[lev][x] <= 2 && a[x] == 0){
            msr[p][mp[lev][x]]++;
        }
    }
    dfs(p,lt[x],lev,side);
    dfs(p,rt[x],lev,side);
}
void traverse_binary_tree(int x){
    if(x==0)return;
    if(a[x] == 0) msl[x][0]++;
    if(a[x] == 0) msr[x][0]++;
    dfs(x,lt[x],lvl[x],1);
    dfs(x,rt[x],lvl[x],2);
    traverse_binary_tree(lt[x]);
    traverse_binary_tree(rt[x]);
}
nnode = n;
decompose(0,1,0);
lift_to_binary(adjj[0][0]);
traverse_binary_tree(adjj[0][0]);
```

Number Theory

```
----------------------------------------------------------------------
int co[MX+5], p[MX+5];
void phi()
{
    for(int i=1; i<=MX; i++) co[i]=i;
    for(int i=2; i<=MX; i++)
    {
        if(!p[i])
        {
            for(int j=i; j<=MX; p[j]=1, j+=i)
                co[j] = ( co[j] / i ) * (i-1);
        }
    }
}
----------------------------------------------------------------------
using u64 = uint64_t;
using u128 = __uint128_t;
u64 bigmod(u64 base, u64 p, u64 mod)
{
    u64 r=1;  base%=mod;
    while(p)
    {
        if(p&1) r=(u128) r*base % mod;
        base = (u128) base*base %mod, p>>=1;
    }
    return r;
}
bool isComposite(u64 n, u64 a, u64 d, int s)
{
    u64 x = bigmod(a,d,n);
    if(x==1 || x==n-1) return false;
    for(int r=1; r<s ; r++)
    {
        x=(u128) x*x %n;
        if(x==n-1) return false;
    }
    return true;
}
bool millerRobin(u64 n)
{
    if(n<4) return (n==2||n==3);
    int s=0; u64 d=n-1;
    while((d&1)==0) d>>=1, s++;
    int iter=10;
    for(int i=0; i<iter; i++)
    {
        u64 a=2+rand()%(n-3);
        if(isComposite(n,a,d,s)) return false;
    }
    return true;
}
----------------------------------------------------------------------
typedef long long    ll;
typedef __int128     ull;
typedef pair<ll,ll>  pL;
ull ext_gcd(ull A, ull B, ull *X, ull *Y)
{
    ull x,x1,x2,y,y1,y2,r,r1,r2,q;
    x1=0; y1=1, x2=1; y2=0;
    for(r2=A, r1=B; r1!=0; r2=r1, r1=r, x2=x1, x1=x,
y2=y1, y1=y)
        q=r2/r1, r=r2%r1, x=x2-q*x1, y=y2-q*y1;
    *X=x2; *Y=y2;  // coefficient of a and b
    return r2;     // gcd
}
bool linearDiophantine(ll a, ll b, ll c) // return X and
Y such that AX+BY+C=0;
{
    ll g = ext_gcd(abs(a), abs(b));
    if (c % g) return false;
    X *= c / g, Y *= c / g;
    if (a < 0) X = -X;
    if (b < 0) Y = -Y;
    return true;
}
pL getXY(ll k, ll A, ll B) // BezoutCoefficient/valid X,
Y
{
    ll gcd=__gcd(A,B);
    return {X+k*B/gcd,Y-k*A/gcd};
}

/** Works for coprime moduli only **/
/** Return {-1,-1} if invalid input.
    Otherwise, returns {x,L}, where x is the solution
unique to mod L
**/
pL CRT( vector<ll> A, vector<ll> M )
{
    if(A.size() != M.size()) return {-1,-1}; /** Invalid
input*/
    int n = A.size();  ull a1 = A[0], m1 = M[0];
    for ( int i = 1; i < n; i++ )
    {
        ull a2 = A[i], m2 = M[i];
        ull p, q;
        ext_gcd(m1, m2, &p, &q);
         ull x = (a1*m2*q + a2*m1*p) % (m1*m2);
        a1 = x, m1 = m1 * m2;
    }
    if (a1 < 0) a1 += m1;
    return {a1, m1};
}
/** Works for both non-coprime and coprime
moduli.
for better understanding code see the comments
of previous code */
pL CRT( vector<ll> A, vector<ll> M ) {
    if(A.size() != M.size()) return {-1,-1};  /** Invalid
input*/
    ll n = A.size();
    ull a1 = A[0], m1 = M[0];
    for ( ll i = 1; i < n; i++ ) {
        ull a2 = A[i], m2 = M[i];
        ull g = __gcd(m1, m2);
```

```
      if ( a1 % g != a2 % g ) return {-1,-1};
      ull p, q;
      ext_gcd(m1/g, m2/g, &p, &q);
      ull mod = (m1 / g) * m2;
      ull x = (a1*(m2/g)*q + a2*(m1/g)*p) % mod;
      a1 = x;
      if (a1 < 0) a1 += mod;
      m1 = mod;
   }
   return {a1, m1};
}
//Lucas Theorem
// can find NCR(n,r,p) where 1<=R<=N<=1e9 and p is a small prime number
map<pair< pair<ll,ll>, ll>, ll>mp; // for memorization
ll NCR(ll n, ll r, ll p)
{
   if(r<0 || r>n)      return 0;
   if(!r || r==n)      return 1;
   if(n>=p)            return (NCR(n/p,r/p,p)*NCR(n%p,r%p,p))%p;
   if(!mp[{{n,r},p}]) mp[{{n,r},p}]=(NCR(n-1,r-1,p)+NCR(n-1,r,p))%p;
   return mp[{{n,r},p}];
}
---------------------------------------------------------------------
```

DP

```
---------------------------------------------------------------------
vector<pair<int, int> > g[MX];
int child[MX],nxt[MX];
int cost[MX][MX],dp[MX][MX];
void findSibling(int x, int p)
{
   int parent,flag=1;
   for(auto u:g[x])
   {
      if(u==p) continue;
      if(flag) child[x]=u, flag=0;
      else nxt[parent]=u;
      parent=u, findSibling(u, x);
   }
}
int siblingDP(int x, int p, int k)
{
   if(x==-1)     return 0;
   if(~dp[x][k]) return dp[x][k];
   int mx=MX;
   int rs=1+siblingDP(child[x], x, m); // create a new subtree as the left child is root
   int rs1=siblingDP(nxt[x],p,k);
// calculate k preservation for siblings of same subtree
   mx=min(mx,rs+rs1);
   int rm=k-cost[x][p];
   // calculation for left child and siblings of same subtree where
   // preservation is distributed between these two
   for(int i=0;i<=rm;i++)
   {
      mx=min(mx,siblingDP(child[x],x,i) + siblingDP(nxt[x],p,rm-i));
   }
   return dp[x][k]=mx;
}
void CLEAR()
{
   mem(child,-1), mem(nxt,-1);
   mem(dp,-1), mem(cost,0);
   for(int i=0;i<=n;i++) g[i].clear();
}
---------------------------------------------------------------------
typedef long double float128;
const ll is_query = -(1LL<<62), inf = 1e18;
struct Line
{
   ll m, b;
   mutable function<const Line*()> succ;
   bool operator<(const Line& rhs) const
   {
      if (rhs.b != is_query) return m < rhs.m;
      const Line* s = succ();
      if (!s)  return 0;
      ll x = rhs.m;
      return b - s->b < (s->m - m) * x;
   }
};
struct HullDynamic : public multiset<Line> // // will maintain lower hull for minimum/maximum
{
   bool bad(iterator y)
   {
      auto z = next(y);
      if (y == begin())
      {
         if (z == end())       return 0;
         return y->m == z->m && y->b <= z->b;
      }
      auto x = prev(y);
      if (z == end()) return y->m == x->m && y->b <= x->b;
      return (float128)(x->b - y->b)*(z->m - y->m) >= (float128)(y->b - z->b)*(y->m - x->m);
   }
   void add_line(ll m, ll b)
   {
      //auto y = insert({ -m, -b }); // For minimum
      auto y = insert({ m, b });     // For maximum
      y->succ = [=] { return next(y) == end() ? 0 : &*next(y); };
      if (bad(y))
      {
         erase(y); return;
      }
      while (next(y) != end() && bad(next(y))) erase(next(y));
      while (y != begin() && bad(prev(y))) erase(prev(y));
```

```
    }

    ll getbest(ll x)
    {
        auto l = *lower_bound((Line)
        {
            x, is_query
        });
        //return -(l.m * x + l.b);  // For minimum
        return (l.m * x + l.b);     // For maximum
    }
} CHT;
---------------------------------------------------------------------
struct line
{
    long long a, b;
    double xleft; bool type;
    line(long long _a, long long _b)
    {
        a = _a, b = _b, type = 0;
    }
    bool operator < (const line &other) const
    {
        if(other.type)  return xleft < other.xleft;
        return a > other.a;
    }
};
double meet(line x, line y)
{
    return 1.0 * (y.b - x.b) / (x.a - y.a);
}
struct cht
{
    set < line > hull;
    cht()
    {
        hull.clear();
    }
    typedef set < line > :: iterator ite;
    bool hasleft(ite node)
    {
        return node != hull.begin();
    }
    bool hasright(ite node)
    {
        return node != prev(hull.end());
    }
    void updateborder(ite node)
    {
        if(hasright(node))
        {
            line temp = *next(node);
            hull.erase(temp);
            temp.xleft = meet(*node, temp);
            hull.insert(temp);
        }
        if(hasleft(node))
        {
            line temp = *node;
            temp.xleft = meet(*prev(node), temp);
            hull.erase(node);
            hull.insert(temp);
        }
        else
        {
            line temp = *node;
            hull.erase(node);
            temp.xleft = -1e18;
            hull.insert(temp);
        }
    }
    bool useless(line left, line middle, line right)
    {
        double x = meet(left, right);
        double y = x * middle.a + middle.b;
        double ly = left.a * x + left.b;
        return y > ly;
    }
    bool useless(ite node)
    {
        if(hasleft(node) && hasright(node))
        {
            return useless(*prev(node), *node,
*next(node));
        }
        return 0;
    }
    void add_line(long long a, long long b)
    {
        //line temp = line(-a, -b); // for maximum
        line temp = line(a, b); // for minimum
        auto it = hull.lower_bound(temp);
        if(it != hull.end() && it -> a == a)
        {
            if(it -> b > b) hull.erase(it);
            else return;
        }
        hull.insert(temp);
        it = hull.find(temp);
        if(useless(it))
        {
            hull.erase(it);
            return;
        }
        while(hasleft(it) && useless(prev(it)))
hull.erase(prev(it));
        while(hasright(it) && useless(next(it)))
hull.erase(next(it));
        updateborder(it);
    }
    long long getbest(long long x)
    {
        if(hull.empty()) return 1e18;
        line query(0, 0);
        query.xleft = x;
        query.type = 1;
    auto it = hull.lower_bound(query);
    it = prev(it);
    //return -(it -> a * x + it -> b); // for maximum
    return (it -> a * x + it -> b); // for minimum
  }
}T[4*MX];
void up(int p, int l, int h, int id, pL pr)
{
  T[p].add_line(pr.first,pr.second);
  if(l==h) return;
  int m=(l+h)/2;
  if(id<=m) up(2*p,l,m,id,pr);
  else      up(2*p+1,m+1,h,id,pr);
}
ll Q(int p, int l, int h, int x, int y, ll vl)
{
  if(l>y||h<x) return inf;
  if(l>=x && h<=y) return T[p].getbest(vl);
  int m=(l+h)/2;
  return min(Q(2*p,l,m,x,y,vl),Q(2*p+1,m+1,h,x,y,vl));
}
ll p[MX],a[MX],h[MX],dp[MX];
int main()
{
  int n; cin>>n;
  for(int i=1; i<=n; i++) cin>>p[i];
  for(int i=1; i<=n; i++) cin>>a[i];
  for(int i=1; i<=n; i++) cin>>h[i];
  dp[1]=a[1];
  up(1,1,n,p[1],{-2*h[1],dp[1]+h[1]*h[1]});
  for(int i=2; i<=n; i++)
  {
    dp[i]=a[i]+Q(1,1,n,1,p[i]-1,h[i])+(h[i]*h[i]);
    up(1,1,n,p[i],{-2*h[i],dp[i]+h[i]*h[i]});
  }
  cout<<dp[n]<<endl;
  return 0;
}
```

SOS DP

```
//O(n) memory
for(int i = 0; i<(1<<N); i++) dp[i] = A[i];
for(int i = 0; i < N; i++)
{
  for(int mask = 0; mask < (1<<N); mask++)
  {
    if(mask & (1<<i)) dp[mask] += dp[mask^(1<<i)];
  }
}
```

```
// n log (n)  memory
for(int mask = 0; mask < (1<<N); mask++) /* use
reverse loop for supermask */
  dp[mask][0] = A[mask];
  if(msk&1) dp[msk][0]+=f[msk^1]; /* use if(!(msk&1))
for supermask */
  for(int i = 1; i < N; ++i)
  {
    if(mask & (1<<i)) dp[mask][i] = dp[mask][i-1] +
dp[mask^(1<<i)][i-1];
    else dp[mask][i] = dp[mask][i-1];
  }
  F[mask] = dp[mask][N-1];
}
```

DS

```
struct HASH
{
  ll base=31, mod=1e9+7/*998244353*/, pw[MX],
H[MX], RH[MX], n, m;
  void generate_hash(string s)
  {
    pw[0]=1; n=s.size();
    for(int i=1; i<=n; i++) pw[i] = (pw[i-1] * base ) %
mod;
    for(int i=0; i<n; i++)  H[i+1]  = ( ( i ? ( H[i] * base ) :
0 ) + s[i]) % mod;
    for(int i=0; i<n; i++)  RH[i+1] = ( ( i ? (RH[i] * base
) : 0 ) + s[n-i-1]) % mod;
  }
  ll getHV(int i, int sz)
  {
    return (H[i+sz] - (H[i] * pw[sz]) % mod + mod) %
mod;
  }
  ll getRHV(int i, int sz)
  {
    return (RH[i+sz] - (RH[i] * pw[sz]) % mod + mod)
% mod;
  }
  ll deleteChar(int i)
  {
    ll h = getHV(i+1, n-i-1);
    if(i) h = ((getHV(0, i) * pw[n-i-1]) % mod +
h)%mod;
    return h;
  }
}HS;
```

```
struct PERSISTANT_TRIE
{
  int vl, in[2];
};
PERSISTANT_TRIE T[25*MX];
void add(int pre, int cur, int x)
{
  for(int i=20; i>=0; i--)
  {
    bool z=x&(1<<i);
    if(!T[cur].in[z])
    {
      T[cur].in[z]=++av;
T[T[cur].in[z]].vl=T[T[cur].in[z]].in[0]=T[T[cur].in[z]].in[1]
=0; //clear instantly
        }
        T[T[cur].in[z]].vl=1+T[T[pre].in[z]].vl;
        T[cur].in[1^z]=T[pre].in[1^z];
        cur=T[cur].in[z], pre=T[pre].in[z];
    }
}
int Q(int pre, int cur, int x)
{
    int mx=0;
    for(int i=20; i>=0; i--)
    {
        bool z=x&(1<<i);
        int d=T[T[cur].in[1^z]].vl-T[T[pre].in[1^z]].vl;
        if(d) mx|=(1<<i), cur=T[cur].in[1^z],
pre=T[pre].in[1^z];
        else cur=T[cur].in[z], pre=T[pre].in[z];
    }
    return mx;
}
----------------------------------------------------------------------
const int chr=26;
struct node
{
    int nxt[chr];
    node() {
        mem(nxt,-1);
    }
};
node T[MX];
int suffix[MX], indx,len, path[MX];
int val[MX],ed[MX];
//int E[MX], esf[MX];
//vector<int>v[MX];   // list of index where an macth is
ocurr for the i'th string
struct Aho_Corasick
{
    void init()
    {
        //mem(esf,0); // contain immediate previous suffix
which is an endpoint of a given string
        //mem(E,0);   // check if the vertex is a endpoint or
not
        len=indx=0;
        mem(T,0), mem(suffix,0), mem(val,0);
        T[indx]=node();
    }
    void insert(string s, int p) // s is pth string in
input
    {
        int now=0;
        for(int i=0; i<s.size(); i++)
        {
            int id=s[i]-'a';
            if(T[now].nxt[id]==-1) T[now].nxt[id]=++indx,
T[indx]=node();
            now=T[now].nxt[id];
        }
        ed[p]=now;
        //E[now]=p;
    }
    void reverse_link()
    {
        queue<int>q;
        for(int i=0; i<chr; i++)
        {
            if(T[0].nxt[i]!=-1) q.push(T[0].nxt[i]);
            else T[0].nxt[i]=0;
        }
        while(!q.empty())
        {
            int u=q.front(); q.pop();
            for(int i=0; i<chr; i++)
            {
                int v=T[u].nxt[i];
                if(v==-1)
                {
                    T[u].nxt[i]=T[suffix[u]].nxt[i]; continue;
                }
                suffix[v]=T[suffix[u]].nxt[i];
                q.push(v);
                path[len++]=v;
                //if(E[suffix[v]]) esf[v]=suffix[v];
                //else          esf[v]=esf[suffix[v]];
            }
        }
    }
    void search(string s)
    {
        int now=0;
        for(int i=0; i<s.size(); i++)
        {
            int id=s[i]-'a';
            now=T[now].nxt[id];
            val[now]++;
            //int nd=now;
            //while(nd>0)
            //{
            //    if(E[nd]) v[E[nd]].push_back(i+1);
            //    nd=esf[nd];
            //}
        }
        for(int i=len-1; i>=0; i--)
val[suffix[path[i]]]+=val[path[i]];
    }
}AC;


----------------------------------------------------------------------
```

```
struct treeNode
{
   int par, depth, sz, pos_segbase, heavy;
} node[N];

vector<int>adj[N];
int bar[N],T[2*N],a[N],heavy[N], av;
int dfs(int cur, int pre, int dpt)
{
   node[cur].par=pre;
   node[cur].depth=dpt;
   node[cur].heavy=-1;
   int s=1,h=0;
   for(int i=0; i<adj[cur].size(); i++)
   {
      if(adj[cur][i]!=pre)
      {
         int c=dfs(adj[cur][i],cur,dpt+1);
         if(c>h) h=c, node[cur].heavy=i;
         s+=c;
      }
   }
   return node[cur].sz=s;
}
int chainNo=0,chainHead[N],chainInd[N];
void hld(int cur, int pre)
{
   if(chainHead[chainNo]==-1)
chainHead[chainNo]=cur;
   chainInd[cur]=chainNo;
   bar[av]=a[cur];
   node[cur].pos_segbase=av++;
   int ind=node[cur].heavy;
   if(ind>=0) hld(adj[cur][ind],cur);
   for(int i=0; i<adj[cur].size(); i++)
   {
      if(ind!=i&&adj[cur][i]!=pre) chainNo++,
hld(adj[cur][i],cur);
   }
}
void build(int n)
{
   for(int i=0;i<n;i++) T[i+n]=bar[i];
else
         {
            ans +=
(RMQ(n,node[chainHead[chain_u]].pos_segbase,node
[u].pos_segbase));
            u = node[chainHead[chain_u]].par;
         }
      }
   }
   return ans;
}
void maxEdge(int u, int v,int n)
{
   int ans = chain_up(u,v,n);
```

```
   for(int i=n-1;i>0;i--) T[i]=T[i<<1]+T[i<<1|1];
}
void update(int n, int pos, int v)
{
   pos+=n;
   for(T[pos]=v,pos>>=1;pos>0;pos>>=1)
T[pos]=T[pos<<1]+T[pos<<1|1];
}
int RMQ(int n, int x, int y)
{
   y++, x+=n;y+=n;
   int s=0;
   for(;x<y;x>>=1,y>>=1)
   {
      if(x&1) s+=T[x++];
      if(y&1) s+=T[--y];
   }
   return s;
}

int chain_up(int u, int v, int n)
{
   int chain_u, chain_v, ans = 0;
   while (true)
   {
      chain_u = chainInd[u], chain_v = chainInd[v];
      if (chain_u==chain_v)
      {
         if(node[v].depth>node[u].depth) swap(u,v);
         ans +=
RMQ(n,node[v].pos_segbase,node[u].pos_segbase);
         break;
      }
      else
      {
         if (chain_u<chain_v)
         {
            ans +=
(RMQ(n,node[chainHead[chain_v]].pos_segbase,node
[v].pos_segbase));
            v=node[chainHead[chain_v]].par;
         }

   printf("%d\n", ans);
}
void Set(int n)
{
   av=0, chainNo=0;
   for(int i=0;i<=n;i++) chainHead[i]=-1, chainInd[i]=0,
adj[i].clear();
}

int main()
{
   int m=dfs(0,0,0);
   Set(n), hld(0,0), build(n);
   update(n,node[x].pos_segbase,y);
maxEdge(x,y,n);
    return 0;
}
------------------------------------------
struct PalindromicTree
{
    int len;
    int link;
    int num;    // cnnt of differennt palindrome
    int occur;  // cnt of same palindromes
    int nxt[26];
};
PalindromicTree T[MX];
int len;   // string length
string s;
int node;  // node 1 - root with len -1, node 2 - root with len 0
int suff;  // max suffix palindrome
int New()
{
    node++;
    T[node].len=T[node].link=0;
    T[node].num=T[node].occur=0;
    mem(T[node].nxt,0);
    return node;
}
bool add(int pos)
{
    int cur=suff, curlen=0;
    int let=s[pos]-'a';

    while(true)   //Finding maximum length palindromic suffix
    {
        curlen=T[cur].len;
        if(pos-1-curlen>=0 && s[pos-1-curlen]==s[pos]) break;
        cur=T[cur].link;
    }
    if(T[cur].nxt[let]) //Existing node
    {
        suff=T[cur].nxt[let];
        return false;
    }
    suff = New();
    T[node].len=T[cur].len+2;
    T[cur].nxt[let]=node;
    if(T[node].len==1) //Single character, connected with root
    {
        T[node].link=2;
        T[node].num=1;
        return true;
    }
    while(true)    //Finding suffix link
    {
        cur=T[cur].link;
        curlen=T[cur].len;
        if(pos-1-curlen>=0 && s[pos-1-curlen]==s[pos])
        {
            T[node].link=T[cur].nxt[let];
            break;
        }
    }
    T[node].num=1+T[T[node].link].num;
    return true;
}
void initTree()
{
    node=suff=2;
    T[1].len=-1, T[2].len=0;
    T[1].link=1, T[2].link=1;
    mem(T[1].nxt,0);
    mem(T[2].nxt,0);
}
ll totalpalindrome=0;
void buildTree()
{
    initTree();
    for(int i=0;i<len;i++)
    {
        add(i); T[suff].occur++;
        totalpalindrome+=T[suff].num;
    }
}
ll countPalindromes()
{
    ll cnt=0;
    for(int i=node;i>2;i--)
    {
        T[T[i].link].occur+=T[i].occur;
        cnt+=T[i].occur;
    }
    return cnt;
}
int main()
{


    cin>>s;
    len=s.size();
    buildTree();
    cout<<totalpalindrome<<endl;

    //cout<<countPalindromes()<<endl;
    return 0;
}
------------------------------------------
void mltple(ll a[2][2], ll b[2][2])
{
    ll ml[2][2];
    for(ll i=0; i<2; i++)
    {
        for(ll j=0; j<2; j++)
        {
            ml[i][j]=0;
            for(ll k=0; k<2; k++) ml[i][j]=(ml[i][j] + (a[i][k]*b[k][j] % m))%m;
        }
    }
    for(ll i=0; i<2; i++)
    {
        for(ll j=0; j<2; j++) a[i][j]=ml[i][j];
    }
}
void power(ll a[2][2],ll n)
{
    ll b[2][2]= {{1,1},{1,0}};
    if(n==1) return;
    else if(n%2==0) power(a,n/2), mltple(a,a);
    else if(n%2!=0) power(a,n-1), mltple(a,b);
}
ll matrixExpo(ll x, ll y, ll n)
{
    ll a[2][2]= {{1,1},{1,0}};
    power(a,n-1);
    return (y*a[0][0]+x*a[0][1])%m;
}
------------------------------------------
#include <ext/pb_ds/assoc_container.hpp>
using namespace __gnu_pbds;
typedef unsigned __int128 ull;
const int RANDOM = chrono::high_resolution_clock::now().time_since_epoch().count();
struct chash {
    int operator()(ull x) const { return (x^(x>>32)^(x>>64)^(x>>96)); }
    //int operator()(int x) const { return x ^ RANDOM; }
};
gp_hash_table<ull,int,chash> table[20];
------------------------------------------
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;
```

```
#define ordered_set tree<int,
null_type,less<int>,
rb_tree_tag,tree_order_statistics_
node_update>
/**PBDS Operations**/
ordered_set st;
int x=*st.find_by_order(4);
//find 4th element in orderd set
int y=st.order_of_key(4);
//number of element less than 4 in
orderd set
if (st.find(4) != st.end())
st.erase(st.find(4));
//delete 4 if exist in orderd set
```

Parallel BS

```
/*Descripion: given a connected
graph of n vertices and m edges.
edges are numbered from 1 to m.
given q query. each contains x, y,
z. for each query you have to tell
if two brother started visited the
graph by order of the given edge
than what will be the minimum
number of edge for which the
number of unique visited vertices
by both of them will be greater
then equal to z. idea: do binary
search for each query parrallelly.
Also need Datastructure (Here,
DSU) . Complexity: O(q * log(q) *
X). where X is dependent on the
problem and the data structures
used in it */int bos[MX], sz[MX],
x[MX], y[MX], z[MX], L[MX],
R[MX];
vector<int>check[MX];
vector<pi>edge;
int Boss(int x)
{
    if(bos[x]==x) return x;
    return bos[x]=Boss(bos[x]);
}
void connectEdge(int id)
{
    --id;
    int a=edge[id].ff, b=edge[id].ss;
    int p=Boss(a), q=Boss(b);
    if(p!=q) bos[q]=p, sz[p]+=sz[q];
}
int main()
{
    int n, m, q; cin>>n>>m;
    for(int i=1; i<=m; i++)
    {
        int a, b; cin>>a>>b;
        edge.push_back({a, b});
    }
    cin>>q;
    for(int i=1; i<=q; i++)
cin>>x[i]>>y[i]>>z[i];
    for(int i=1; i<=q; i++)  L[i]=1,
R[i]=m; //set lower and
upperbound for each query
    for(int i=1; i<=LG; i++)
    {
        for(int j=1; j<=n; j++) bos[j]=j,
sz[j]=1; //reset DSU
        for(int j=1; j<=m; j++)
check[j].clear(); // clear mid query
        for(int j=1; j<=q; j++)
//generated mid point for each
query
        {
            if(L[j]!=R[j])
            {
                int mid=(L[j]+R[j])/2;

check[mid].push_back(j); //insert
query index to its current mid
point
            }
        }
        for(int e=1; e<=m; e++)
//build DSU by insert edges one
by one
        {
            connectEdge(e);
            for(auto k:check[e])
//check validation and reset
lower/upperbound for each query
which current midpoint is e
            {
                int a=Boss(x[k]),
b=Boss(y[k]);
                if(a==b && sz[a]>=z[k]
|| a!=b && sz[a]+sz[b]>=z[k])
R[k]=e;
                else L[k]=e+1;
            }
        }
    }
    for(int i=1; i<=q; i++)
cout<<R[i]<<'\n';
    return 0;
}
```

Arithmetic Progression:

a, (a+d), (a+2*d), (a+3*d),.....

nth term: an= a1+(n-1)*d

Sum of first n terms:

S= (n/2)*(2*a + (n-1)*d)

Geometric Progression:

a, a*r, a*r^2, a*r^3,........

Sum of first n terms:

a*((r^n) - 1)/(r-1)

Gaussian Elimination:

```
/// Gaussian Elimination(Gauss
Jordan method): O(n*m*m)
/// Matrix dimension n x m and
(m+1)th is column vector
#define mxn 102
#define EPS 1e-8
double dp[mxn][mxn];

int Gauss(int n, int m){
    int col, row, mxr;
    for(col= row= 1; row<=n &&
col<=m; row++, col++){
        mxr= row;
        for(int i=row+1; i<=n; i++)

if(fabs(dp[i][col])>fabs(dp[mxr][col
]))mxr=i;
        if(mxr!= row)swap(dp[row],
dp[mxr]);
        if(fabs(dp[row][col])<EPS){
            row--;
            continue;
        }
        for(int i=1; i<=n; i++)
        if(i!= row &&
fabs(dp[i][col])>EPS){
            for(int j=m+1; j>=col; j--)
                dp[i][j]-=
(dp[row][j]/dp[row][col])*dp[i][col];
        }
    }
    row--;
    for(int i=row; i>=1; i--){
    for(int j=i+1; j<=row; j++)
        dp[i][m+1]-=
(dp[j][m+1]*dp[i][j]);
        dp[i][m+1]/= dp[i][i];
    }return row;/// returns rank of a
matrix!
}
int main()
{
    Gauss(100, 100);
    cout<<dp[1][101]<<endl;
    return 0;
}
```