|  |
| --- |
| V 0.2 |
| SchrodingerVirgin |
| --------ACM模板 |

|  |
| --- |
| 2015-5-20 |

目录

* 计算几何. . . . . . . . . . . . . . . . . . 2
* 数学. . . . . . . . . . . . . . . . . . . . . . 10
* 数据结构. . . . . . . . . . . . . . . . . . 21
* 图论. . . . . . . . . . . . . . . . . . . . . . 29
* DP. . . . . . . . . . . . . . . . . . . . . . . . 34
* 附录. . . . . . . . . . . . . . . . . . . . . . 39

1. 计算几何

* 二维

#include<cstdio>

#include<cstring>

#include<vector>

#include<algorithm>

#include<cmath>

using namespace std;

const double eps = 1e-10;

const double PI = acos(-1.0);

int dcmp(double x)

{

if(fabs(x) < eps) return 0;

else return x > 0? 1:-1;

}

//////////////////////////////点和直线部分//////////////////////////////////////

struct Point

{

double x,y;

Point(double x = 0,double y = 0):x(x),y(y) {}

};

typedef Point Vector;

Vector operator + (Vector A,Vector B) { return Vector(A.x+B.x,A.y+B.y); }

Vector operator - (Point A,Point B) { return Vector(A.x-B.x,A.y-B.y); }

Vector operator \* (Vector A,double p) { return Vector(A.x\*p,A.y\*p); }

Vector operator / (Vector A,double p) { return Vector(A.x/p,A.y/p); }

bool operator == (const Point& A,const Point& B) { return dcmp(A.x-B.x) == 0 && dcmp(A.y-B.y) == 0; }

bool operator < (const Point& A,const Point& B) { return A.x < B.x || (A.x == B.x && A.y < B.y); }

double Dist(Point A,Point B)

{

return sqrt((A.x-B.x)\*(A.x-B.x) + (A.y-B.y)\*(A.y-B.y));

}

double Dot(Vector A,Vector B) { return A.x\*B.x + A.y\*B.y; }

double Cross(Vector A,Vector B) { return A.x\*B.y - A.y\*B.x; }

double Length(Vector A) { return sqrt(Dot(A,A)); }

double Angle(Vector A,Vector B) { return acos(Dot(A,B) / Length(A) / Length(B)); } //A,B所成角

double Angle(Vector A) { return atan2(A.y,A.x); } //向量与x轴正方向所成角

double Area2(Point A,Point B,Point C) { return Cross(B-A,C-A); } //求AB,AC向量所成平行四边形面积

Vector Normal(Vector A) //求A向量单位法相量　向左侧

{

double L = Length(A);

return Vector(-A.y/L,A.x/L);

}

Vector Rotate(Vector A,double rad)

{

return Vector(A.x\*cos(rad)-A.y\*sin(rad),A.x\*sin(rad)+A.y\*cos(rad));

}

Point TwoLinePoint(Point P,Vector v,Point Q,Vector w) //求P+tv和Q+tw的交点

{

Vector u = P-Q;

double t = Cross(w,u)/Cross(v,w);

return P+v\*t;

}

double PointToLine(Point P,Point A,Point B) //点P到AB直线距离

{

Vector v1 = B-A,v2 = P-A;

return fabs(Cross(v1,v2))/Length(v1);

}

double PointToSegment(Point P,Point A,Point B) //点P到线段AB距离

{

if(A == B) return Length(P-A);

Vector v = B-A,v1 = P-A,v2 = P-B;

if(dcmp(Dot(v,v1)) < 0) return Length(v1);

else if(dcmp(Dot(v,v2)) > 0) return Length(v2);

else return fabs(Cross(v,v1)) / Length(v);

}

Point LineProjection(Point P,Point A,Point B) //点P在直线AB上的投影点

{

Vector v = B-A;

return A+v\*(Dot(v,P-A) / Dot(v,v));

}

bool IsSegmentProperIntersection(Point a1,Point a2,Point b1,Point b2) //判断２线段是否相交（规范相交：只有１个交点且交点不在线段端点）

{

double c1 = Cross(a2-a1,b1-a1),c2 = Cross(a2-a1,b2-a1),

c3 = Cross(b2-b1,a1-b1),c4 = Cross(b2-b1,a2-b1);

return dcmp(c1)\*dcmp(c2) < 0 && dcmp(c3)\*dcmp(c4) < 0;

}

bool IsPointOnSegment(Point P,Point A,Point B) //判断点是否在线段上

{

return dcmp(Cross(A-P,B-P)) == 0 && dcmp(Dot(A-P,B-P)) < 0; //如果允许在端点上就改成 <=

}

bool IsSegmentInproperIntersection(Point a1,Point a2,Point b1,Point b2) //判断２线段是否相交(非规范）

{

if(IsPointOnSegment(a1,b1,b2) || IsPointOnSegment(a2,b1,b2) ||

IsPointOnSegment(b1,a1,a2) || IsPointOnSegment(b2,a1,a2))

return true;

return IsSegmentProperIntersection(a1,a2,b1,b2);

}

double PolygonArea(const vector<Point>& P)//求多边形面积

{

int n = P.size();

double area = 0;

for(int i = 1;i < n-1;++i)

area += Area2(P[0],P[i],P[i+1]);

return area/2;

}

///////////////////////////////////直线和圆///////////////////////////////////////////

struct Line

{

Point p;

Vector v;

Line(Point p,Vector v): p(p),v(v) {}

Point getPoint(double t) { return p+v\*t; } //获得直线上一点

Line move(double d) { return Line(p+Normal(v)\*d,v); } //直线沿法相量平移d

};

struct Circle

{

Point c;

double r;

Circle(Point c,double r):c(c),r(r) {}

Point getPoint(double rad) //获得圆上一点(rad为逆时针角)

{

return Point(c.x+cos(rad)\*r,c.y\*sin(rad)\*r);

}

};

int LineCirclePoint(Line L,Circle C,vector<Point>& res) //求直线与圆的交点，返回交点数

{

double a = L.v.x,b = L.p.x-C.c.x,c = L.v.y,d = L.p.y-C.c.y;

double e = a\*a+c\*c,f = 2\*(a\*b + c\*d),g = b\*b+d\*d-C.r\*C.r;

double delta = f\*f-4\*e\*g;

double t1,t2;

if(dcmp(delta) < 0) return 0;

if(dcmp(delta) == 0)

{

t1 = t2 = -f / (2\*e);

res.push\_back(L.getPoint(t1));

return 1;

}

t1 = (-f - sqrt(delta)) / (2\*e);

res.push\_back(L.getPoint(t1));

t2 = (-f + sqrt(delta)) / (2\*e);

res.push\_back(L.getPoint(t2));

return 2;

}

int TwoCirclePoint(Circle C1,Circle C2,vector<Point>& res)

//求２圆交点，返回交点数（特别的－１表示２圆重合，无穷个交点)

{

double d = Length(C1.c-C2.c);

if(dcmp(d) == 0)

{

if(dcmp(C1.r-C2.r) == 0) return -1;

return 0;

}

if(dcmp(C1.r + C2.r - d) < 0) return 0;

if(dcmp(fabs(C1.r-C2.r) - d) > 0) return 0;

double a = Angle(C2.c-C1.c);

double da = acos((C1.r\*C1.r+d\*d-C2.r\*C2.r) / (2\*C1.r\*d));

Point p1 = C1.getPoint(a-da),p2 = C1.getPoint(a+da);

res.push\_back(p1);

if(p1 == p2) return 1;

res.push\_back(p2);

return 2;

}

int PointCircleTangents(Point p,Circle C,vector<Line>& res) //过顶点球圆的切线，返回切线条数

{

Vector u = C.c - p;

double dist = Length(u);

if(dist < C.r) return 0;

else if(dcmp(dist - C.r) == 0)

{

res.push\_back(Line(p,Rotate(u,PI/2)));

return 1;

}

else

{

double ang = asin(C.r/dist);

res.push\_back(Line(p,Rotate(u,-ang)));

res.push\_back(Line(p,Rotate(u,+ang)));

return 2;

}

}

int TwoCircleTangents(Circle A,Circle B,vector<Point>& a,vector<Point>& b)

//返回两圆的切线条数, -1表示无穷条切线, a,b分别是i条切线分别在A,B上的切点

{

int cnt = 0;

if(A.r < B.r) { swap(A,B); swap(a,b); }

int d2 = (A.c.x-B.c.x)\*(A.c.x-B.c.x) + (A.c.y-B.c.y)\*(A.c.y-B.c.y);

int rdiff = A.r-B.r;

int rsum = A.r+B.r;

if(d2 < rdiff\*rdiff) return 0;

double base = atan2(B.c.y-A.c.y,B.c.x-A.c.x);

if(d2 == 0 && A.r == B.r) return -1;

if(d2 == rdiff\*rdiff)

{

a.push\_back(A.getPoint(base)) ; b.push\_back(B.getPoint(base)); cnt++;

return 1;

}

double ang = acos((A.r-B.r) / sqrt(d2));

a.push\_back(A.getPoint(base+ang)); b.push\_back(B.getPoint(base+ang)); cnt++;

a.push\_back(A.getPoint(base-ang)); b.push\_back(B.getPoint(base-ang)); cnt++;

if(d2 == rsum\*rsum

{

a.push\_back(A.getPoint(base)); b.push\_back(B.getPoint(PI+base)); cnt++;

}

else if(d2 > rsum\*rsum)

{

double ang = acos((A.r+B.r) / sqrt(d2));

a.push\_back(A.getPoint(base+ang)); b.push\_back(B.getPoint(PI+base+ang)); cnt++;

a.push\_back(A.getPoint(base-ang)); b.push\_back(B.getPoint(PI+base-ang)); cnt++;

}

return cnt;

}

/////////////////////////点和多边形///////////////////////////////////

int IsPointInPolygon(Point p,vector<Point>& poly) //判断点p是否在多边形内　-1表示在边界上，1在内部，０在外部

{

int wn = 0;

int n=poly.size();

for(int i = 0; i < n; i++)

{

if(IsPointOnSegment(p, poly[i], poly[(i+1)%n])) return -1;

int k = dcmp(Cross(poly[(i+1)%n]-poly[i], p-poly[i]));

int d1 = dcmp(poly[i].y-p.y);

int d2 = dcmp(poly[(i+1)%n].y-p.y);

if(k > 0 && d1 <= 0 && d2 > 0) wn++;

if(k < 0 && d2 <= 0 && d1 > 0) wn--;

}

if (wn != 0) return 1;

return 0;

}

/////////////////////点群///////////////////////////

//计算凸包

//输入点顺序不会被破坏

//如果希望在凸包的边上有点，把２个<=改成< ?

//精度较高时候使用dcmp比较

vector<Point> p;

Point res[maxn];

vector<Point> ConvexHull(vector<Point>& p) //最后res中的凸包点 按逆时针顺序(好像是的)..

{

// 预处理，删除重复点

sort(p.begin(), p.end());

p.erase(unique(p.begin(), p.end()), p.end());

int n = p.size();

int m = 0;

vector<Point> ch(n+1);

for(int i = 0;i < n;++i)

{

while(m > 1 && dcmp(Cross(ch[m-1]-ch[m-2],p[i]-ch[m-2])) <= 0) m--;

ch[m++] = p[i];

}

int k = m;

for(int i = n-2;i >= 0;--i)

{

while(m > k && dcmp(Cross(ch[m-1]-ch[m-2],p[i]-ch[m-2])) <= 0) m--;

ch[m++] = p[i];

}

if(n > 1) m--;

ch.resize(m);

return ch;

}

//旋转卡壳求最大２点距离

double RotatingCalipers(vector<Point>& p)

{

// vector<Point> p(points);

ConvexHull(p);

int n = p.size();

if(n == 1) return 0;

if(n == 2) return Dist(p[0], p[1]);

p.push\_back(p[0]); // 免得取模

double ans = 0;

for(int u = 0, v = 1; u < n; u++) {

// 一条直线贴住边p[u]-p[u+1]

for(;;)

{

// 当Area(p[u], p[u+1], p[v+1]) <= Area(p[u], p[u+1], p[v])时停止旋转

// 即Cross(p[u+1]-p[u], p[v+1]-p[u]) - Cross(p[u+1]-p[u], p[v]-p[u]) <= 0

// 根据Cross(A,B) - Cross(A,C) = Cross(A,B-C)

// 化简得Cross(p[u+1]-p[u], p[v+1]-p[v]) <= 0

double diff = Cross(p[u+1]-p[u],p[v+1]-p[v]);

if(dcmp(diff) <= 0)

{

ans = max(ans,Dist(p[u],p[v]));

if(dcmp(diff) == 0) ans = max(ans,Dist(p[u],p[v+1]));

break;

}

}

v = (v + 1) % n;

}

}

return ans;

}

* 三维

#include<cstdio>

#include<cstring>

#include<cmath>

#include<ctime>

#include<cstdlib>

#include<vector>

using namespace std;

const double eps = 1e-10;

struct Point3

{

double x,y,z;

Point3(double x = 0,double y = 0,double z = 0):x(x),y(y),z(z) {}

};

typedef Point3 Vector3;

Vector3 operator + (Vector3 A,Vector3 B)

{

return Vector3(A.x+B.x,A.y+B.y,A.z+B.z);

}

Vector3 operator - (Point3 A,Point3 B)

{

return Vector3(A.x-B.x,A.y-B.y,A.z-B.z);

}

Vector3 operator \* (Vector3 A,double p)

{

return Vector3(A.x\*p,A.y\*p,A.z\*p);

}

Vector3 operator / (Vector3 A,double p)

{

return Vector3(A.x/p,A.y/p,A.z/p);

}

double Dot(Vector3 A,Vector3 B) { return A.x\*B.x+A.y\*B.y+A.z\*B.z; }

double Length(Vector3 A) { return sqrt(Dot(A,A)); }

double Angle(Vector3 A,Vector3 B) { return acos(Dot(A,B) / Length(A) / Length(B)); }

double dcmp(double x)

{

if(fabs(x) < eps) return 0;

else return x<0?-1:1;

}

bool operator == (const Point3& a,const Point3& b)

{

return dcmp(a.x-b.x) == 0 && dcmp(a.y-b.y) == 0 && dcmp(a.z-b.z);

}

double PointToPlane(Point3 p,Point3 p0,Vector3 n) //p到面(p0,n)的距离,n需为单位向量

{

return fabs(Dot(p-p0,n));

}

Point3 PlaneProjection(Point3 p,Point3 p0,Vector3 n) //p在(p0,n)的投影，n须为单位向量

{

return p-n\*Dot(p-p0,n);

}

Point3 LinePlanePoint(Point3 p1,Point3 p2,Point3 p0,Vector3 n) //p1-p2和(p0,n)的交点（假定存在）

{

Vector3 v = p2-p1;

double t = (Dot(n,p0-p1) / Dot(n,p2-p1));

return p1+v\*t;

}

Vector3 Cross(Vector3 A,Vector3 B)

{

return Vector3(A.y\*B.z-A.z\*B.y,A.z\*B.x-A.x\*B.z,A.x\*B.y-A.y\*B.x);

}

double Area2(Point3 A,Point3 B,Point3 C)

{

return Length(Cross(B-A,C-A));

}

bool IsPointInTriangle(Point3 P,Point3 P0,Point3 P1,Point3 P2)

//判断P是否在三角形P0P1P2中(假设P在平面P0,P1,P2上)

{

double area1 = Area2(P,P0,P1);

double area2 = Area2(P,P1,P2);

double area3 = Area2(P,P2,P0);

return dcmp(area1+area2+area3 - Area2(P0,P1,P2)) == 0;

}

bool IsSegmentTriangleIntersection(Point3 P0,Point3 P1,Point3 P2,Point3 A,Point3 B,Point3& P)

//判断三角形P0P1P2是否和线段AB相交

{

Vector3 n = Cross(P1-P0,P2-P0);

if(dcmp(Dot(n,B-A)) == 0) return false;

else

{

double t = Dot(n,P0-A) / Dot(n,B-A);

if(dcmp(t) < 0 || dcmp(t-1) > 0) return false;

P = A + (B-A)\*t;

return IsPointInTriangle(P,P0,P1,P2);

}

}

double PointToLine(Point3 P,Point3 A,Point3 B) //P到AB距离

{

Vector3 v1 = B-A,v2 = P-A;

return Length(Cross(v1,v2) / Length(v1));

}

double PointToSegment(Point3 P,Point3 A,Point3 B)//P到线段AB距离

{

if(A == B) return Length(P-A);

Vector3 v1 = B-A,v2 = P-A,v3 = P-B;

if(dcmp(Dot(v1,v2)) < 0) return Length(v2);

else if(dcmp(Dot(v1,v3)) > 0) return Length(v3);

else return Length(Cross(v1,v2)) / Length(v1);

}

double Volume6(Point3 A,Point3 B,Point3 C,Point3 D) //四面体ABCD有向面积的６倍

{

return Dot(D-A,Cross(B-A,C-A));

}

1. 数学

* 数论

#include<cstdio>

#include<cstring>

#include<cmath>

#include<algorithm>

#include<map>

using namespace std;

typedef long long LL;

const double eps = 1e-6;

const int maxn = 10000;

//V + F-2 = E 顶点数＋面数-2 = 边数

//斐波那且O(1) = ( ( (1+sqrt(5))/2 )^n - ( (1-sqrt(5))/2 )^n ) / sqrt(5)

LL gcd(LL a,LL b)

{

return !b? a : gcd(b,a%b);

}

LL lcm(LL a,LL b)

{

return a/gcd(a,b)\*b;

}

/////////////////////////////////////组合数//////////////////////////////////

LL C[maxn+10][maxn+10];

void GetAllC(int n) //计算所有C

{

for(int i = 0;i <= n;++i)

{

C[i][0] = 1;

for(int j = 1;j <= i;++j) C[i][j] = C[i-1][j] + C[i-1][j-1];

}

}

LL GetC(LL n,LL m)

{

m = m < (n-m)? m : (n-m);

LL res = 1;

for(int i = 1;i <= m;++i)

res = res\*(n-i+1)/i;

return res;

}

//利用计算C(n,m) % p,p是素数的

LL GetC(LL n,LL m)

{

if(n < m) return 0;

if(m > n-m) m = n-m;

LL s1 = 1,s2 = 1;

for(LL i = 0;i < m;++i)

{

s1 = (s1\*(n-i))%mod;

s2 = (s2\*(i+1))%mod;

}

return s1\*pow\_mod(s2,mod-2)%mod;　//b在p为素数下的逆为b^(p-2)%p

}

//当C(n,m)非常大时候，需要利用lucas计算C(n,m)%p

LL lucas(LL n,LL m)

{

if(m == 0) return 1;

return GetC(n%mod,m%mod)\*lucas(n/mod,m/mod)%mod;

}

////////////////////////////////素数//////////////////////////////////////////

bool isprm[maxn+10]; //0表示是prm

int mu[maxn]; //莫比乌斯反演系数

int prm[maxn];

int pn;

void GetPrm(int n) //获取素数表

{

memset(isprm,0,sizeof(isprm));

pn = 0;

for(LL i = 2;i <= n;++i)

if(!isprm[i])

{

prm[pn++] = i;

for(LL j = i\*i;j <= n;j += i) isprm[j] = 1;

}

}

int fac\_cnt[maxn];

void factorize(int num) //唯一分解定理分解num

{

memset(fac\_cnt,0,sizeof(fac\_cnt));

for(int i = 0;i < pn;++i)

{

while(num % prm[i] == 0)

{

fac\_cnt[i]++;

num /= prm[i];

}

if(num == 1)

break;

}

}

int cal(int pr,int n) //计算N!中分解到pr的幂

{

int res = 0;

while(n)

n /= pr,res += n;

return res;

}

LL phi[maxn];

void GetAllPhi(int n)

{

for(int i = 2; i <= n; i++) phi[i] = 0;

phi[1] = 1;

for(int i = 2; i <= n; i++) if(!phi[i])

for(int j = i; j <= n; j += i)

{

if(!phi[j]) phi[j] = j;

phi[j] = phi[j] / i \* (i-1);

}

}

LL GetPhi(int n)

{

int m = (int) sqrt(n+0.5);

LL ans = n;

for(int i = 2; i <= m; i++)

if(n % i == 0)

{

ans = ans / i \* (i-1);

while(n % i == 0) n /= i;

}

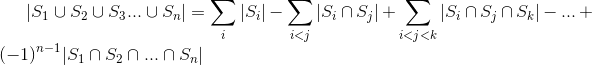
if(n > 1) ans = ans / n \* (n-1);

return ans;

}

一般设U中元素有n种不同性质，第i种性质成为Pi满足Pi的元素组成集合Si，那么

满足P1,P2..,Pn 中至少一个性质的U中元素的个数是:



容斥原理



//莫比乌斯

int sum[maxn];

void GetMobius(int n)

{

pn = 0;

memset(isprm,0,sizeof(isprm));

mu[1] = 1;

for(LL i = 2;i <= n;++i)

{

if(!isprm[i])

{

prm[pn++] = i;

mu[i] = -1;

}

for(LL j = 0;j < pn && i\*prm[j] <= n;++j)

{

isprm[i\*prm[j]] = 1;

if(i%prm[j]) mu[i\*prm[j]] = -mu[i];

else

{

mu[i\*prm[j]] = 0;

break;

}

}

}

for(int i = 1;i <= n;++i)

sum[i] = sum[i-1] + mu[i];

}

LL solve(int n,int m) //求1 <= x <= n,1 <= y <= m中 gcd(x,y) == 1的个数

{

LL res = 0;

if(n > m) swap(n,m);

for(int i = 1,last = 0;i <= n;i = last+1)

{

last = min(n/(n/i),m/(m/i));

res += (LL)(n/i)\*(m/i)\*(sum[last]-sum[i-1]);

}

return res;

}

/////////////////////////////模////////////////////////////////////////

//long long 的乘法

LL Mod(LL x)

{

if(x >= mod) return x-mod;

return x;

}

LL mul(LL a,LL b)

{

LL res = 0;

while(b)

{

if(b & 1) res = MOD(res+a);

a = MOD(a+a);

b >>= 1;

}

return res;

}

////////////////

LL mul\_mod(LL a,LL b,LL m)

{

return a\*b%m;

}

LL pow\_mod(LL a,LL p,LL m)

{

LL res = 1;

while(p)

{

if(p&1) res = res\*a%m;

a = a\*a%m;

p >>= 1;

}

return res;

}

void exgcd(LL a,LL b,LL& d,LL &x,LL &y); //前向声明

// 求模m下a的逆. 需要a,m互素，否则返回-1

LL inv(LL a,LL m)

{

LL d,x,y;

exgcd(a,m,d,x,y);

return d==1 ? (x+m) % m : -1;

}

//////////////////////////////////解方程组//////////////////////////////////////////////

// ax + by = gcd(a,b) 解(x,y) g=gcd(a,b)

void exgcd(LL a,LL b,LL& d,LL &x,LL &y)

{

if(!b) d = a,x = 1,y = 0;

else { exgcd(b,a%b,d,y,x); y -= (a/b)\*x; }

}

//　ax + by = c 有解　　==> c % g = 0

// (x\*c/g,y\*c/g) ==> (x0,y0)

// a'= a/g b'= b/g

// 解 (x0+kb',y0-ka') k为任意整数

bool exgcd\_res(LL& a,LL& b,LL c,LL& x0,LL& y0)

{

LL d;

exgcd(a,b,d,x0,y0);

if(c % d) return false;

else

{

a=a%d;

b=b%d;

x0=x0/d\*c;

y0=y0/d\*c;

return true;

}

}

//求解线性模方程ax = b(mod n),解为x,x+m0,x+2\*m0......

bool line\_mod(LL a,LL b,LL m,LL& x,LL& m0)

{

LL d,y;

exgcd(a,m,d,x,y);

if(b % d) return false;

else

{

x = x\*b/d;

m0 = m/d;

x = ((x%m0)+m0)%m0;//x 为最小正数解

return true;

}

}

//中国剩余定理解线性模方程组x = a[i](mod m[i])

LL china(LL n, LL a[], LL m[]) //m都互质

{

LL M = 1, d, y, x = 0;

for(int i = 0; i < n; i++) M \*= m[i];

for(int i = 0; i < n; i++)

{

LL w = M / m[i];

exgcd(m[i], w, d, d, y);

x = (x + y\*w\*a[i]) % M;

}

return (x+M) % M;

}

/\*我们先可以先找两个同余方程 设通解为N,N=r1(mod(m1)),N=r2(mod(m2)),

显然可以化为k1\*m1+r1=k2\*m2+r2;--->k1\*m1+(-k2\*m2)=r2-r1;

设a=m1,b=m2,x=k1,y=(-k2),c=r2-r1方程可写为ax+by=c;

由欧几里得解得x即可,那么将x化为原方程的最小正整数解，(x\*(c/d)%(b/d)+(b/d))%(b/d);

这里看不懂的去看解模线性方程。那么这个x就是原方程的最小整数解。

所以N=a\*（x+n\*（b/d））+r1====N=(a\*b/d)\*n+(a\*x+r1),

这里只有n为未知数所以又是一个N=(a\*x+r1)(mod(a\*b/d))的式子，

然后只要不断的将两个式变成一个式子，最后就能解出这个方程组的解。 \*/

LL china(LL n,LL A[],LL M[]) //m不互质的情况，求最小x

{

LL a = M[0],c1 = A[0];

for(int i = 1;i < n;++i)

{

LL b = M[i],c2 = A[i];

LL x,y,d;

exgcd(a,b,d,x,y);

LL c = c2-c1;

if(c % d) return -1;

LL b1 = b/d;

x = ( (c/d\*x)%b1 + b1)%b1;

c1 = a\*x+c1;

a = a\*b1;

}

if(c1 == 0) //当余数都为0,那么取最小公倍数

{

c1 = 1;

for(int i = 0;i < n;++i)

c1 = c1\*M[i]/gcd(c1,M[i]);

}

return c1;

}

//解模方程a^x = b mod(n) 无解返回-1 （对于n是素数的情况)

LL log\_mod(LL a,LL b,LL n)

{

LL m = (LL)sqrt(n+0.5);

LL v = inv(pow\_mod(a,m,n),n);

LL e = 1;

map<LL,LL> x;

x[1] = 0;

for(int i = 1;i < m;++i)

{

e = mul\_mod(e,a,n);

if(!x.count(e)) x[e] = i;

}

for(int i = 0;i < m;++i)

{

if(x.count(b)) return i\*m+x[b];

b = mul\_mod(b,v,n);

}

return -1;

}

//扩展Baby step Gaint step (对于任意n）

/\*

方法：

初始d=1,c=0,i=0;

1.令g=gcd(a,n),若g==1则执行下一步。否则由于a^x=k\*n+b;(k为某一整数),则(a/g)\*a^k=k\*(n/g)+b/g,(b/g为整除，若不成立则无解)

令n=n/g，d=d\*a/g，b=b/g,c++则d\*a^(x-c)=b(mod n),接着重复1步骤。

2.通过1步骤后，保证了a和d都与n互质，方程转换为d\*a^(x-c)=b(mod n)。由于a和n互质，所以由欧拉定理a^phi(n)==1(mod n),(a,n互质)

可知，phi(n)<=n,a^0==1(mod n),所以构成循环，且循环节不大于n。从而推出如果存在解，则必定1<=x<n。(在此基础上我们就可以用

Baby Step Giant Step方法了)

3.令m=ceil(sqrt(n)),则m\*m>=n。用哈希表存储a^0,a^1,..,a^(m-1)，接着判断1~m\*m-1中是否存在解。

4.为了减少时间，所以用哈希表缩减复杂度。分成m次遍历，每次遍历a^m长度。由于a和d都与n互质，所以gcd(d,n)=1，

所以用拓展的欧几里德定理求得d\*x+n\*y=gcd(d,n)=1,的一组整数解(x,y)。则d\*x+n\*y=1-->d\*x%n=(d\*x+n\*y)%n=1-->d\*(x\*b)%n=((d\*x)%n\*b%n)%n=b。

所以若x\*b在哈希表中存在，值为k，则a^k\*d=b(mod n),答案就是ans=k+c+i\*m。如果不存在，则令d=d\*a^m,i++后遍历下一个a^m，直到遍历a^0到a^(m-1)还未找到，则说明不解并退出。

\*/

LL log\_mod (LL a,LL b,LL n,LL c,LL d)

{

LL m,v,e=1,i,x,y,dd;

m=(LL)(sqrt(n+0.5)); //x=i\*m+j

map<LL,LL>f;

f[1]=m;

for(i=1;i<m;i++) //建哈希表，保存a^0,a^1,...,a^m-1

{

e=(e\*a)%n;

if(!f[e])f[e]=i;

}

e=(e\*a)%n;//e=a^m

for(i=0;i<m;i++)//每次增加m次方，遍历所有1<=f<=n

{

exgcd(d,n,dd,x,y);//d\*x+n\*y=1-->(d\*x)%n=1-->d\*(x\*b)%n==b

x=(x\*b%n+n)%n;

if(f[x])

{

LL num=f[x];

f.clear();//需要清空，不然会爆内存

return c+i\*m+(num==m?0:num);

}

d=(d\*e)%n;

}

return -1;

}

int main()

{

LL a,b,n;

while(scanf("%lld %lld %lld",&a,&n,&b) == 3)

{

if(b >= n)

{

puts("Orz,I can’t find D!");

continue;

}

if(b == 0)

{

printf("0");

continue;

}

LL ans = 0,c = 0,d = 1,t;

while((t = gcd(a,n)) != 1)

{

if(b % t) { ans = -1; break; }

c++;

n /= t;

b /= t;

d = d\*a/t%n;

if(d == b) { ans = c;break; }

}

if(ans != 0)

{

if(ans == -1) puts("Orz,I can’t find D!");

else printf("%lld\n",ans);

}

else

{

ans = log\_mod(a,b,n,c,d);

if(ans == -1) puts("Orz,I can’t find D!");

else printf("%lld\n",ans);

}

}

return 0;

}

///////////////////////矩阵////////////////////////////

typedef LL Matrix[maxn][maxn];

void gauss\_jordan(Matrix A,int n)

{

for(int i = 0;i < n;++i) {

int r = i;

for(int j = i+1;j < n;++j)

if(fabs(A[j][i]) > fabs(A[r][i])) r = j;

if(fabs(A[r][i]) < eps) continue;

if(r != i) for(int j = 0;j <= n;++j) swap(A[r][j],A[i][j]);

for(int k = 0;k < n;++k)

if(k != i)

for(int j = n;j >= i;--j)

A[k][j] -= A[k][i]/A[i][i] \* A[i][j];

}

}

int sz; //矩阵大小

LL mod; //模

void matrix\_mul(Matrix A,Matrix B,Matrix res)

{

Matrix C;

memset(C,0,sizeof(C));

for(int i = 0;i < sz;++i)

for(int j = 0;j < sz;++j)

for(int k = 0;k < sz;++k)

C[i][j] = (C[i][j] + A[i][k] \* B[k][j]) % mod;

memcpy(res,C,sizeof(C));

}

void matrix\_pow(Matrix A,LL n,Matrix res)

{

Matrix a,r;

memcpy(a,A,sizeof(a));

memset(r,0,sizeof(r));

for(int i = 0;i < sz;++i) r[i][i] = 1;

while(n){

if(n&1) matrix\_mul(r,a,r);

n >>= 1;

matrix\_mul(a,a,a);

}

memcpy(res,r,sizeof(r));

}

//求逆矩阵．．．．．will done

//计算a^1+a^2+a^3...a^n; 注意这里Matrix需要写成结构体形式

Matrix sum(Matrix a,LL n)

{

if(n == 1)

return a;

if(n&1)

return (a^n) + sum(a,n-1);

else

return sum(a,n/2) \* ((a^(n/2))+1);

}

////////////////////////////博弈论///////////////////////////////////

int SG[maxn];

int mex(int x)

{

int& ans = SG[x];

if(ans != -1) return ans;

bool vis[maxn]; //注意这里必须是局部数组，否则会被下一状态更改

memset(vis,0,sizeof(vis));

for(int i = 0;i < 10 && save[i] <= x;++i) //save[i]记录可以改变的数量

{

int t = x-save[i];

SG[t] = mex(t);

vis[SG[t]] = 1;

}

for(int i = 0;;i++)

if(!vis[i]) return ans = i;

}

递推:

约瑟夫: 

///////////////////////////奇奇怪怪的东西////////////////////////////

//母函数

memset(c1,0,sizeof(c1));

memset(c2,0,sizeof(c2));

memset(num,0,sizeof(num));

int sum = 0;

for(int i = 0;i < n;++i)

{

scanf("%d",&A[i]);

sum += A[i];

num[i] = 1;

}

for(int i = 0;i <= num[0]\*A[0];i += A[0])

c1[i] = 1;

for(int i = 1;i < n;++i)

{

for(int j = 0;j <= sum;++j)

for(int k = 0;k <= num[i]\*A[i] && j + k <= sum; k += A[i])

{

c2[j+k] += c1[j];

c2[abs(j-k)] += c1[j];

}

for(int j = 0;j <= sum;++j)

{

c1[j] = c2[j];

c2[j] = 0;

}

}

//数值积分

//F为被积分函数

double simpson(double a,double b)

{

double c = a+(b-a)/2;

return (F(a)+4\*F(c)+F(b))\*(b-a)/6;

}

double asr(double a,double b,double eps,double A)

{

double c = a + (b-a)/2;

double L = simpson(a,c), R = simpson(c,b);

if(fabs(L+R-A) <= 15\*eps) return L+R+(L+R-A)/15.0;

return asr(a,c,eps/2,L) + asr(c,b,eps/2,R);

}

double asr(double a,double b,double eps)

{

return asr(a,b,eps,simpson(a,b));

}

1. 数据结构

* 纯数据结构

//树状数组

void add(int x,int v)

{

for(int i = x;i < maxm;i += lowbit(i))

sum[i] += v;

}

int query(int x)

{

int res = 0;

for(int i = x;i > 0;i -= lowbit(i))

res += sum[i];

return res;

}

//二维

int sum(int x,int y)

{

int sum = 0;

for(int i = x;i > 0;i -= lowbit(i))

for(int j = y;j > 0;j -= lowbit(j))

sum += C[i][j];

return sum;

}

void add(int x,int y,int val)

{

for(int i = x;i < maxn;i += lowbit(i))

for(int j = y;j < maxn; j += lowbit(j))

C[i][j] += val;

}

/\*使用时先给a１－aｎ调用add,查询用sum\*/

//RMQ

//一维 0-----n-1

void RMQ\_init(const vector<int>& A)

{

int n = A.size();

for(int i = 0;i < n;++i)

d[i][0] = A[i];

for(int j = 1;(1 << j) <= n;++j)

for(int i = 0;i+(1<<j)-1 < n;++i)

d[i][j] = max(d[i][j-1],d[i+(1<<(j-1))][j-1]);

}

int query(int L,int R)

{

int k = 0;

while((1<<(k+1)) <= R-L+1) k++;

return max(d[L][k],d[R-(1<<k)+1][k]);

}

//二维 1---n

void RMQ\_init()

{

for(int r = 1;r <= R;++r)

for(int c = 1;c <= C;++c)

d[r][c][0][0] = A[r][c];

for(int i = 0;i <= power[R];++i)

for(int j = 0;j <= power[C];++j)

{

if(i == 0 && j == 0) continue;

int limitR = R+1-(1<<i);

int limitC = C+1-(1<<j);

for(int r = 1;r <= limitR;++r)

for(int c = 1;c <= limitC;++c)

{

if(i == 0)

d[r][c][i][j] = max(d[r][c][i][j-1],d[r][c+(1<<(j-1))][i][j-1]);

else

d[r][c][i][j] = max(d[r][c][i-1][j],d[r+(1<<(i-1))][c][i-1][j]);

}

}

}

int query(int r1,int c1,int r2,int c2)

{

int res = -1;

int k1 = power[r2-r1+1],k2 = power[c2-c1+1];

res = max(res,d[r1][c1][k1][k2]);

res = max(res,d[r1][c2-(1<<k2)+1][k1][k2]);

res = max(res,d[r2-(1<<k1)+1][c1][k1][k2]);

res = max(res,d[r2-(1<<k1)+1][c2-(1<<k2)+1][k1][k2]);

return res;

}

//线段树

//单点修改

const int maxn = 50000+10;

int sum[maxn<<2];

void pushup(int u)

{

sum[u] = sum[u << 1] + sum[u << 1 | 1];

}

void build(int L,int R,int u) //自底向上建树

{

if(L == R)

{

scanf("%d",&sum[u]);

return;

}

int m = (L + R) >> 1;

build(L,m,u << 1);

build(m+1,R,u << 1 | 1);

pushup(u);

}

void update(int p,int v,int L,int R,int u)

{

if(L == R)

{

sum[u] += v;

return;

}

int m = (L+R) >> 1;

if(p <= m) update(p,v,L,m,u << 1);

else update(p,v,m+1,R,u << 1 | 1);

pushup(u);

}

int query(int qL,int qR,int L,int R,int u)

{

if(qL <= L && R <= qR) return sum[u];

int m = (L+R) >> 1;

int res = 0;

if(qL <= m) res += query(qL,qR,L,m,u << 1);

if(qR > m) res += query(qL,qR,m+1,R,u << 1 | 1);

return res;

}

* 字符串

//KMP

void getFail(char \*P,int f[]) //失配函数: 如果P[j] 能匹配到P[i],那么f[j] = i+1

{

int m = strlen(P);

f[0] = 0,f[1] = 0;

for(int i = 1;i < m;++i)

{

int j = f[i];

while(j && P[i] != P[j]) j = f[j];

f[i+1] = P[i] == P[j] ?j+1:0;

}

}

void find(char\* T,char \*P,int f[])

{

int n = strlen(T),m = strlen(P);

getFail(P,f);

int j = 0;

for(int i = 0;i < n;++i)

{

while(j && P[j] != T[i]) j = F[j];

if(P[j] == T[i]) j++;

if(j == m) printf("%d\n",i-m+1);

}

}

//Trie

const int maxnode = 4000\*100+10;

const int sigma\_size = 26;

struct Trie

{

int ch[maxnode][sigma\_size];

int val[maxnode];

int sz;

void clear() { sz = 1;memset(ch[0],0,sizeof(ch[0])); }

int idx(char c) { return c - 'a'; }

void insert(const char \*s,int v)

{

int u = 0,n = strlen(s);

for(int i = 0;i < n;++i)

{

int c = idx(s[i]);

if(!ch[u][c])

{

memset(ch[sz],0,sizeof(ch[sz]));

val[sz] = 0;

ch[u][c] = sz++;

}

u = ch[u][c];

}

val[u] = v;

}

void find\_prefixes(const char\* s,int len,vector<int>& ans)

{

int u = 0;

for(int i = 0;i < len;++i)

{

if(s[i] == 0) break;

int c = idx(s[i]);

if(!ch[u][c]) break;

u = ch[u][c];

if(val[u] != 0) ans.push\_back(val[u]);

}

}

};

//后缀数组

int s[maxn];

int sa[maxn],t[maxn],t2[maxn],c[maxn]; //sa[i]是排序后第i个在原字符串中开始的下标

int srank[maxn],height[maxn]; //srank[i]是原字符串下标i所对应的排序后的名次

int n;

int ans;

void build\_sa(int m)

{

int \*x = t,\*y = t2;

for(int i = 0;i < m;++i) c[i] = 0;

for(int i = 0;i < n;++i) c[x[i] = s[i]]++;

for(int i = 0;i < m;++i) c[i] += c[i-1];

for(int i = n-1;i >= 0;--i) sa[--c[x[i]]] = i;

for(int k = 1;k <= n;k <<= 1)

{

int p = 0;

for(int i = n-k;i < n;++i) y[p++] = i;

for(int i = 0;i < n;++i) if(sa[i] >= k) y[p++] = sa[i]-k;

for(int i = 0;i < m;++i) c[i] = 0;

for(int i = 0;i < n;++i) c[x[y[i]]]++;

for(int i = 0;i < m;++i) c[i] += c[i-1];

for(int i = n-1;i >= 0;--i) sa[--c[x[y[i]]]] = y[i];

swap(x,y);

p = 1,x[sa[0]] = 0;

for(int i = 1;i < n;++i)

x[sa[i]] = y[sa[i-1]] == y[sa[i]] && y[sa[i-1]+k] == y[sa[i]+k] ? p-1:p++;

if(p >= n) break;

m = p;

}

}

void getHeight()

{

int k = 0;

for(int i = 0;i < n;++i) srank[sa[i]] = i;

height[srank[0]] = 0;

for(int i = 0;i < n;++i)

{

if(k) k--;

int j = sa[srank[i]-1];

while(s[i+k] == s[j+k]) k++;

height[srank[i]] = k;

}

}

scanf("%s",buf);

n = strlen(save);

// memset(height,0,sizeof(height));

for(int i = 0;i < n;++i) s[i] = save[i]-'a'+1; //原始字符数组（最后一个字符应必须是0，而前面的字符必须非0）

s[n++] = 0;

build\_sa(27);

getHeight();

//Hash

#include<bits/stdc++.h>

using namespace std;

typedef long long LL;

typedef unsigned long long ULL;

const int BASE1 = 313;

const int BASE2 = 31;

const LL MOD = 1000\*1000\*1000+7;

const int N = 600000+100;

LL pw1[N];

LL pw2[N];

set<pair<LL,LL> > was[N];

LL geth1(char\* s)

{

ULL hs = 0; //此处高能，防止溢出

while(\*s)

hs = (hs \* BASE1 + \*s++ - 'a' + 1)%MOD;

return (LL)hs;

}

LL geth2(char\* s)

{

ULL hs = 0;

while(\*s)

hs = (hs \* BASE2 + \*s++ - 'a' + 1)%MOD;

return (LL)hs;

}

int main()

{

// freopen("./test.txt","r",stdin);

pw1[0] = pw2[0] = 1;

for(int i = 1;i < N;++i)

{

pw1[i] = pw1[i-1]\*BASE1 % MOD;

pw2[i] = pw2[i-1]\*BASE2 % MOD;

}

int n,m;

scanf("%d %d",&n,&m);

char s[N];

for(int i = 0;i < n;++i)

{

scanf("%s",s);

was[strlen(s)].insert(make\_pair(geth1(s),geth2(s)));

}

for(int i = 0;i < m;++i)

{

scanf("%s",s);

int len = strlen(s);

LL h1 = geth1(s);

LL h2 = geth2(s);

bool ok = false;

for(int j = 0;j < len && !ok;++j)

for(int c = 'a';c <= 'c' && !ok;++c)

{

if(c == s[j]) continue;

LL nh1 = ((h1 + pw1[len - 1 - j] \* (c - s[j])) % MOD + MOD) % MOD; //替换s[j]后的hash

LL nh2 = ((h2 + pw2[len - 1 - j] \* (c - s[j])) % MOD + MOD) % MOD;

if(was[len].find(make\_pair(nh1,nh2)) != was[len].end())

ok = true;

}

if(ok)

puts("YES");

else

puts("NO");

}

return 0;

}

//Hash容错表

void init()

{

memset(head,-1,sizeof(head));

}

int modhash(int x)

{

int sum = 0;

for(int i = 0;i < 6;++i)

sum += save[x][i];

return sum%hashsize;

}

bool insert(int x)

{

int h = modhash(x);

int u = head[h];

while(u != -1)

{

if(judge(u,x)) return false;

u = nxt[u];

}

nxt[x] = head[h];

head[h] = x;

return true;

}

1. 图论

//拓扑排序

#include<cstdio>

#include<cstring>

#include<vector>

#include<algorithm>

using namespace std;

const int maxn = 1000;

bool G[maxn][maxn];

int indegree[maxn];

int n,m;

vector<int> res;

bool toposort(){

res.clear();

bool unfinish = true;;

while(unfinish)

{

unfinish = false;

for(int i = 1;i <= n;++i)

if(indegree[i] == 0)

{

unfinish = true;

indegree[i]--;

res.push\_back(i);

for(int j = 1;j <= n;++j) if(i != j && G[i][j]) indegree[j]--;

break;

}

}

return res.size() == n;

}

int main()

{

while(scanf("%d%d",&n,&m) == 2) {

memset(G,0,sizeof(G));

memset(indegree,0,sizeof(indegree));

for(int i = 0;i < m;++i){

int a,b;

scanf("%d%d",&a,&b);

if(!G[a][b]){

G[a][b] = 1;

indegree[b]++;

}

}

if(!toposort(){{

puts("NO");

break;

}

for(int i = 0;i < n;++i) printf("%d%c",res[i],(i == n-1)?'\n':' ');

}

return 0;

}

//Prim

memset(vis,0,sizeof(vis));

memset(d,0,sizeof(d));

vis[0] = 1;

for(int i = 0;i < n;++i)

d[i] = cal(0,i);

int res = 0;

for(int i = 1;i < n;++i)

{

int Min = INF,v;

for(int j = 1;j < n;++j)

if(!vis[j] && d[j] < Min)

Min = d[j],v = j;

vis[v] = 1;

res += Min;

for(int j = 0;j < n;++j)

{

int t = cal(v,j);

if(!vis[j] && d[j] > t)

d[j] = t;

}

}

//欧拉回路

#include<bits/stdc++.h>

using namespace std;

const int maxn = 100;

int n,G[maxn][maxn],degree[maxn];

void init()

{

memset(degree,0,sizeof(degree));

memset(G,0,sizeof(G));

}

struct Edge

{

int from,to;

Edge(int from = 0,int to = 0):from(from),to(to) {}

};

vector<Edge> res;

int back\_judge;

void dfs(int u){

for(int i = 0;i < n;++i)

if(G[u][i])

{

G[u][i]--,G[i][u]--;

dfs(i);

back\_judge++;

res.push\_back(Edge(u,i));

}

}

bool solve()

{

int cnt = 0,u = -1;

for(int i = 0;i < n;++i)

{

if(degree[i] & 1){

cnt++;

u=i;

}

else if(degree[i] && u==-1) u=i;

}

if(cnt>2) //如果是欧拉回路的问题，只需要改成cnt != 0

{

printf("No solution\n");

return false;

}

back\_judge = 0;

dfs(u);

if(back\_judge < n)

{

printf("No solution\n");

return false;

}

return true;

}

//哈密顿

#include<cstdio>

#include<algorithm>

using namespace std;

const int maxn = 1000+10;

int G[maxn][maxn];

int Prev[maxn],Next[maxn];

int n;

int main()

{

scanf("%d",&n);

for(int i = 0;i < n;++i)

{

int t;

while(1)

{

scanf("%d",&t);

G[i][t-1] = 1;

char ch = getchar();

if(ch == '\n' || ch == EOF) break;

}

}

for(int i = 0;i < n;++i) Next[i] = (i+1)%n;

for(int i = 0;i < n;++i) Prev[i] = (i-1+n)%n;

bool inok = true;

while(inok)

{

inok = false;

int x = 0;

while(1)

{

if(G[x][Next[x]] == 0)

{

inok = true;

int y = 0;

while(1){

int u = x,v = Next[x];

int p = y,q = Next[y];

if(v != p && u != p && u != q)

{

if(G[u][p] && G[v][q])

{

Next[u] = Prev[u],Prev[u] = p;

Next[v] = Next[v],Prev[v] = q;

Prev[p] = Prev[p],Next[p] = u;

Prev[q] = Next[q],Next[q] = v;

for(int z = Next[u];z != q;z = Next[z])

swap(Next[z],Prev[z]);

break;

}

}

y = Next[y];

if(y == 0) break;

}

break;

}

x = Next[x];

if(x == 0) break;

}

}

int x = 0;

while(1)

{

printf("%d ",x+1);

x = Next[x];

if(x == 0) break;

}

puts("1");

return 0;

}

//LCA(Tarjan 离线)

#include<cstdio>

#include<vector>

#include<cstring>

using namespace std;

const int maxn = 10000;

vector<int> G[maxn],que[maxn];

int n;

int fa[maxn],num[maxn],ans[maxn],indegree[maxn];

int vis[maxn];

void init()

{

for(int i = 1;i <= n;++i)

{

G[i].clear();

que[i].clear();

num[i] = 1;

fa[i] = i;

}

memset(vis,0,sizeof(vis));

memset(indegree,0,sizeof(indegree));

memset(ans,0,sizeof(ans));

}

int findfa(int x)

{

return fa[x] == x?x: fa[x] = findfa(fa[x]);

}

void add(int x,int y)

{

int a = findfa(x);

int b = findfa(y);

if(a != b)

{

fa[b] = a;

}

}

void LCA(int u) {

ans[u] = u;

for(int i = 0; i < G[u].size();++i)

{

LCA(G[u][i]);

add(u,G[u][i]);

ans[findfa(G[u][i])] = u;

}

vis[u] = 1;

for(int i = 0;i < que[u].size();++i)

if(vis[que[u][i]]) {

printf("%d\n",ans[findfa(que[u][i])]);

return;

}

}

int main() {

int kase;

scanf("%d",&kase);

while(kase--)

{

scanf("%d",&n);

init();

int a,b;

for(int i = 0;i < n-1;++i){

scanf("%d %d",&a,&b);

if(a != b){

G[a].push\_back(b);

indegree[b]++;

}

}

scanf("%d %d",&a,&b);

que[a].push\_back(b);

que[b].push\_back(a);

int u;

for(u = 1;u <= n;++u)

if(indegree[u] == 0) break;

LCA(u);

}

return 0;

}

//二分图

//二分图最小点覆盖数 = 最大匹配数

bool vis[maxn];

vector<int> G[maxn];

int match[maxn],n;

bool dfs(int u){

for(int i = 0; i < G[u].size();++i){

int v = G[u][i];

if(!vis[v]){

vis[v] = 1;

int t = match[v]; match[v] = u;

if(t == -1 || dfs(t)) return true;

match[v] = t;

}

}

return false;

}

int solve(){

int ans = 0;

memset(match,-1,sizeof(match));

for(int i = 0;i < n;++i){

memset(vis,0,sizeof(vis));

if(dfs(i)) ans++;

}

return ans;

}

* DP
* 序列DP

1. LIS(最长上升子序列)



int LIS(int n){

for(int i=1;i<=n;++i) g[i]=INF;

a[n]=INF-1;

for(int i=0;i<=n;++i){

int k=lower\_bound(g+1,g+n+1,a[i]) - g; // find in g[1..n],not find k=1; 若非严格递增，find中a[i]改为a[i]+1

d[i]=k;

g[k]=a[i]; //entrue a[i]<=g[k]

}

return d[n]-1;

}

1. LCS(最长公共子序列)



1. LCIS(最长公共上升子序列)

int LCIS(int n,int m)

{

memset(f,0,sizeof(f));

for(int i=0;i<n;++i){

int k=0,p=-1;

for(int j=0;j<m;++j){

if(a[i]>b[j])

if(f[j]>k)

k=f[p=j];

if(a[i]==b[j])

if(f[j] < k+1)

f[j]=k+1,prev[j]=p;

}

}

int k=0,p=-1;

for(int i=0;i<m;++i)

if(f[i]>k) k=f[p=i];

for(int i=k-1;i>=0;--i)

{

lcis[i]=b[p];

p=prev[p];

}

return k;

}

1. SCS

找到一个最短的字串，其子序列包含了所有给定字串

|SCS|=|S1|+|S2|-|LCS|

1. 最大连续(和)子序列

int last=0,res=-INF,p1=0,p2=0;

for(int i=0;i<n;++i){

last=max(0,last)+a[i];

//res=max(res,last);

if(res<last){

p1=i;

res=last;

}

}

int sum=0;

for(int j=p1;i<n;++j){

sum+=a[j];

if(sum==res){

p2=j;

break;

}

}

* 区间DP

1. 回文子序列

最长



统计



1. 最优三角剖分

将一个n个顶点的凸多边形中用n-3条互不相交的对角线将其划分为n-2个三角形，使w(i,j,k)(每一个三角形的权函数)最大

定义d(i,j)为子多边形i,i+1,…,j (i<j) 的最优解,则中间存在一个三角形 i-k-j (i<k<j)

最后的解为 d(0,n-1)



1. 最优矩阵链乘

矩阵相乘规则:两个矩阵,n×m,m×p相乘的运算量为nmp

给出一个n个矩阵组成的序列,选择一种相乘顺序使总运算量尽量小,第i个矩阵表示为 m[i-1]×m[i]

f(i,j) 表示矩阵i到j相乘所需的最小运算量



* 集合DP

1. 最优配对问题

空间里有n个点P0,P1,P2…,Pn-1,把他们分为n-2对，使所有点对中两点距离之和尽量小。

S表示已选点(下标)的集合,d(S)表示S中的点的最优解



d[0]=0;

for(int S=1;S<(1<<n);++S){

int i,j;

d[S]=INF;

for(i=0;i<n;++i)

if(S&(1<<i)) break;

for(int j=i+1;i<n;++j)

if(S&(1<<j)) d[S]=min(d[S],dist(i,j)+d[S^(1<<i)^(1<<j)]);

}

1. TSP

经典的NPC难题。

n个城市，两两之间均有道路相连，求一条经过每个城市一次且仅一次，最后回到起点的路线，使得经过道路总长度最短，城市编号0~n-1。

不妨设0为起点，d(i,S)表示目前在城市i,还需访问S中城市各一次回到0的最短长度。

d(i,S)=Min(d(j,S-{j})+dist(i,j) j \in S)

边界为d(i,0)=dist(0,i),答案为d(0,1,2,3,…,n−1)

for(int i=0;i<n;++i) d[i][0]=dist(0,i);

for(int i=0;i<n;++i){

for(int S=1;S<(1<<n);++S){

if(S&(1<<i)) continue;

d[i][S]=INF;

for(int j=0;j<n;++j) if(S&(1<<j))

d[i][S]=min(d[i][S],dist(i,j)+d[j][S^(1<<j)])

}

}

1. 图染色

一个无向图G，用最少的颜色将图染色，使得相邻结点颜色不同

d(S)表示把结点集S染色所需的最少颜色

d(S)=(d(S−S′)+1|S′⊂S)且S′内无相邻结点(可以染成同一种颜色的结点集)

d[0]=0;

for(int S=1;S<(1<<n);++S){

d[S]=INF;

for(int S0=S; S0; S0=(S0-1)&S)

if(no\_edges\_inside(S0)) d[S]=min(d[S],d[S-S0]+1);

}

* 树形DP

1. 重心

对于一棵n个结点的无根树，选择一个结点作为根结点，使其最大子树的结点最少

* d(i)表示以i为根的子树结点个数

d(i)=∑j∈s(i) d(j)+1

Ans=Min(Max(d(j)),n−d(i)))

* 只需要dfs一次,遍历每个结点，将无根树转化为有根树即可

1. 最长路径(最远点对)

对于一棵n个结点的无根树，找出一对点使距离最远,也称树的直径

* d(i)表示以i为根结点到叶子的最大距离

d(i)=Max(d(u)+d(v))+2

* u,v分别为最远和次远叶结点所在子树(两次dfs可求)

也可以不用dp，这里补充下树的直径的一些特性:

1. 任取一个结点u，用dfs/bfs求出u的最远结点v，再用一次dfs求出v的最远结点w，则路径(v,w)为最远点对
2. 树上任意一点的最远距离是其树的直径的两端点较远的一个

结合上面特性可以O(n)求出每个结点的最远点(三次bfs):

#include<bits/stdc++.h>

using namespace std;

const int maxn=10000 + 10;

int n;

bool vis[maxn];

vector<int> G[maxn];

vector<int> dis[maxn];

void init(){

for(int i=1;i<=n;++i) G[i].clear();

for(int i=1;i<=n;++i) dis[i].clear();

}

int bfs(int d[],int k){

memset(vis,0,sizeof(vis));

queue<int> q;

int res=k,mx=0;

q.push(k);

d[k]=0;

vis[k]=1;

while(!q.empty()){

int u=q.front();q.pop();

for(int i=0;i<G[u].size();++i){

int v=G[u][i];

if(vis[v]) continue;

vis[v]=1;

d[v]=d[u]+dis[u][i];

q.push(v);

if (d[v]>mx){

mx=d[v];

res=v;

}

}

}

return res;

}

int main()

{

int v,w;

while(scanf("%d",&n)==1){

init();

for(int u=2;u<=n;++u){ //2~n的结点，u,v,w,　u-v连通，权值为w

scanf("%d %d",&v,&w);

G[u].push\_back(v);

G[v].push\_back(u);

dis[u].push\_back(w);

dis[v].push\_back(w);

}

int ans1[maxn],ans2[maxn];

int u=bfs(ans1,1);

int v=bfs(ans1,u);

bfs(ans2,v);

for(int i=1;i<=n;++i) printf("%d\n",max(ans1[i],ans2[i]));

}

return 0;

}

* 附录

//分数

typedef long long LL;

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

LL lcm(LL a,LL b) { return a / gcd(a,b)\*b; }

struct Rat

{

LL a, b;

Rat(LL a=0):a(a),b(1) { }

Rat(LL x, LL y):a(x),b(y) {

if(b < 0) a = -a, b = -b;

LL d = gcd(a, b); if(d < 0) d = -d;

a /= d; b /= d;

}

};

Rat operator + (const Rat& A,const Rat& B)

{

LL x = lcm(A.b,B.b);

return Rat(A.a\*(x/A.b) + B.a\*(x/B.b),x);

}

Rat operator - (const Rat& A,const Rat& B) { return A + Rat(-B.a,B.b); }

Rat operator \* (const Rat& A,const Rat& B)

{

LL t1 = gcd(A.a,B.b),t2 = gcd(A.b,B.a);

return Rat((A.a/t1)\*(B.a/t2),(A.b/t2)\*(B.b/t1));

}

// BigInteger

#include<string>

#include<iostream>

#include<iosfwd>

#include<cmath>

#include<cstring>

#include<stdlib.h>

#include<stdio.h>

#include<cstring>

#define MAX\_L 2005 //最大长度，可以修改

using namespace std;

class bign

{

public:

int len, s[MAX\_L];//数的长度，记录数组

//构造函数

bign();

bign(const char\*);

bign(int);

bool sign;//符号 1正数 0负数

string toStr() const;//转化为字符串，主要是便于输出

friend istream& operator>>(istream &,bign &);//重载输入流

friend ostream& operator<<(ostream &,bign &);//重载输出流

//重载复制

bign operator=(const char\*);

bign operator=(int);

bign operator=(const string);

//重载各种比较

bool operator>(const bign &) const;

bool operator>=(const bign &) const;

bool operator<(const bign &) const;

bool operator<=(const bign &) const;

bool operator==(const bign &) const;

bool operator!=(const bign &) const;

//重载四则运算

bign operator+(const bign &) const;

bign operator++();

bign operator++(int);

bign operator+=(const bign&);

bign operator-(const bign &) const;

bign operator--();

bign operator--(int);

bign operator-=(const bign&);

bign operator\*(const bign &)const;

bign operator\*(const int num)const;

bign operator\*=(const bign&);

bign operator/(const bign&)const;

bign operator/=(const bign&);

//四则运算的衍生运算

bign operator%(const bign&)const;//取模（余数）

bign factorial()const;//阶乘

bign Sqrt()const;//整数开根（向下取整）

bign pow(const bign&)const;//次方

//一些乱乱的函数

void clean();

~bign();

};

#define max(a,b) a>b ? a : b

#define min(a,b) a<b ? a : b

bign::bign()

{

memset(s, 0, sizeof(s));

len = 1;

sign = 1;

}

bign::bign(const char \*num)

{

\*this = num;

}

bign::bign(int num)

{

\*this = num;

}

string bign::toStr() const

{

string res;

res = "";

for (int i = 0; i < len; i++)

res = (char)(s[i] + '0') + res;

if (res == "")

res = "0";

if (!sign&&res != "0")

res = "-" + res;

return res;

}

istream &operator>>(istream &in, bign &num)

{

string str;

in>>str;

num=str;

return in;

}

ostream &operator<<(ostream &out, bign &num)

{

out<<num.toStr();

return out;

}

bign bign::operator=(const char \*num)

{

memset(s, 0, sizeof(s));

char a[MAX\_L] = "";

if (num[0] != '-')

strcpy(a, num);

else

for (int i = 1; i < strlen(num); i++)

a[i - 1] = num[i];

sign = !(num[0] == '-');

len = strlen(a);

for (int i = 0; i < strlen(a); i++)

s[i] = a[len - i - 1] - 48;

return \*this;

}

bign bign::operator=(int num)

{

if (num < 0)

sign = 0, num = -num;

else

sign = 1;

char temp[MAX\_L];

sprintf(temp, "%d", num);

\*this = temp;

return \*this;

}

bign bign::operator=(const string num)

{

const char \*tmp;

tmp = num.c\_str();

\*this = tmp;

return \*this;

}

bool bign::operator<(const bign &num) const

{

if (sign^num.sign)

return num.sign;

if (len != num.len)

return len < num.len;

for (int i = len - 1; i >= 0; i--)

if (s[i] != num.s[i])

return sign ? (s[i] < num.s[i]) : (!(s[i] < num.s[i]));

return !sign;

}

bool bign::operator>(const bign&num)const

{

return num < \*this;

}

bool bign::operator<=(const bign&num)const

{

return !(\*this>num);

}

bool bign::operator>=(const bign&num)const

{

return !(\*this<num);

}

bool bign::operator!=(const bign&num)const

{

return \*this > num || \*this < num;

}

bool bign::operator==(const bign&num)const

{

return !(num != \*this);

}

bign bign::operator+(const bign &num) const

{

if (sign^num.sign)

{

bign tmp = sign ? num : \*this;

tmp.sign = 1;

return sign ? \*this - tmp : num - tmp;

}

bign result;

result.len = 0;

int temp = 0;

for (int i = 0; temp || i < (max(len, num.len)); i++)

{

int t = s[i] + num.s[i] + temp;

result.s[result.len++] = t % 10;

temp = t / 10;

}

result.sign = sign;

return result;

}

bign bign::operator++()

{

\*this = \*this + 1;

return \*this;

}

bign bign::operator++(int)

{

bign old = \*this;

++(\*this);

return old;

}

bign bign::operator+=(const bign &num)

{

\*this = \*this + num;

return \*this;

}

bign bign::operator-(const bign &num) const

{

bign b=num,a=\*this;

if (!num.sign && !sign)

{

b.sign=1;

a.sign=1;

return b-a;

}

if (!b.sign)

{

b.sign=1;

return a+b;

}

if (!a.sign)

{

a.sign=1;

b=bign(0)-(a+b);

return b;

}

if (a<b)

{

bign c=(b-a);

c.sign=false;

return c;

}

bign result;

result.len = 0;

for (int i = 0, g = 0; i < a.len; i++)

{

int x = a.s[i] - g;

if (i < b.len) x -= b.s[i];

if (x >= 0) g = 0;

else

{

g = 1;

x += 10;

}

result.s[result.len++] = x;

}

result.clean();

return result;

}

bign bign::operator \* (const bign &num)const

{

bign result;

result.len = len + num.len;

for (int i = 0; i < len; i++)

for (int j = 0; j < num.len; j++)

result.s[i + j] += s[i] \* num.s[j];

for (int i = 0; i < result.len; i++)

{

result.s[i + 1] += result.s[i] / 10;

result.s[i] %= 10;

}

result.clean();

result.sign = !(sign^num.sign);

return result;

}

bign bign::operator\*(const int num)const

{

bign x = num;

bign z = \*this;

return x\*z;

}

bign bign::operator\*=(const bign&num)

{

\*this = \*this \* num;

return \*this;

}

bign bign::operator /(const bign&num)const

{

bign ans;

ans.len = len - num.len + 1;

if (ans.len < 0)

{

ans.len = 1;

return ans;

}

bign divisor = \*this, divid = num;

divisor.sign = divid.sign = 1;

int k = ans.len - 1;

int j = len - 1;

while (k >= 0)

{

while (divisor.s[j] == 0) j--;

if (k > j) k = j;

char z[MAX\_L];

memset(z, 0, sizeof(z));

for (int i = j; i >= k; i--)

z[j - i] = divisor.s[i] + '0';

bign dividend = z;

if (dividend < divid) { k--; continue; }

int key = 0;

while (divid\*key <= dividend) key++;

key--;

ans.s[k] = key;

bign temp = divid\*key;

for (int i = 0; i < k; i++)

temp = temp \* 10;

divisor = divisor - temp;

k--;

}

ans.clean();

ans.sign = !(sign^num.sign);

return ans;

}

bign bign::operator/=(const bign&num)

{

\*this = \*this / num;

return \*this;

}

bign bign::operator%(const bign& num)const

{

bign a = \*this, b = num;

a.sign = b.sign = 1;

bign result, temp = a / b\*b;

result = a - temp;

result.sign = sign;

return result;

}

bign bign::pow(const bign& num)const

{

bign result = 1;

for (bign i = 0; i < num; i++)

result = result\*(\*this);

return result;

}

bign bign::factorial()const

{

bign result = 1;

for (bign i = 1; i <= \*this; i++)

result \*= i;

return result;

}

void bign::clean()

{

if (len == 0) len++;

while (len > 1 && s[len - 1] == '\0')

len--;

}

bign bign::Sqrt()const

{

if(\*this<0)return -1;

if(\*this<=1)return \*this;

bign l=0,r=\*this,mid;

while(r-l>1)

{

mid=(l+r)/2;

if(mid\*mid>\*this)

r=mid;

else

l=mid;

}

return l;

}

bign::~bign()

{

}

bign num0,num1,res;

int main()

{

cin>>num0>>num1;

res=num0+num1;

cout<<res<<endl;

num0=5;

num1="71";

res=num0-num1;

cout<<res<<endl;

res=num0.Sqrt();

cout<<res<<endl;

res=num0.pow(5);

cout<<res<<endl;

return 0;

}

//BigDecimal

#include <iostream>

#include <cstdio>

#include <cstdlib>

#include <cstring>

#include <queue>

#define INF 1E9

using namespace std;

struct BigNum

{

int len;

int num[10000];

int point;

BigNum()

{

len=1;

point=0;

memset(num,0,sizeof(num));

}

};

bool input(BigNum &a)//输入

{

string s;

if(cin>>s)

{

memset(a.num,0,sizeof(a.num));

int t=0,i;

a.len=s.size();

a.point=0;

for(i=s.size()-1;i>=0;i--)

{

if(s[i]=='.'){a.len--;a.point=t;continue;}

a.num[t++]=s[i]-'0';

}

return 1;

}

else return 0;

}

void output(BigNum &a)//输出

{

int i,j=0,flag;

for(i=0;i<a.point&&a.num[i]==0;i++);

flag=i;

if(a.point==a.len)

{

if(flag==a.point){cout<<"0"<<endl;return;}

else cout<<".";

}

// cout<<a.point<<" "<<flag<<endl;

for(i=a.len-1;i>=0;i--)

{

cout<<a.num[i];

if(i==flag)break;

if(i==a.point)cout<<".";

}

cout<<endl;

}

BigNum Mul(BigNum &a, BigNum &b)

{

int i, j, len = 0;

BigNum c;

for(i = 0; i < a.len; i++)

for(j = 0; j < b.len; j++)

{

c.num[i+j] += (a.num[i]\*b.num[j]);

if(c.num[i+j] >= 10)

{

c.num[i+j+1] += (int)c.num[i+j]/10;

c.num[i+j] %= 10;

}

}

c.point=a.point+b.point;

len = a.len+b.len;

while(c.num[len-1] == 0 && len > 1&&len>c.point) len--;

if(c.num[len]) len++;

c.len = len;

return c;

}

BigNum a;

int b;

int main()

{

while(input(a)&&~scanf("%d",&b))

{

BigNum ans;

if(b==0){cout<<1<<endl;continue;}

ans.num[0]=1;

while(b--)

{

ans=Mul(ans,a);

}

output(ans);

}

}

//FFT 快速傅里叶变换

#include <iostream>

#include <string.h>

#include <stdio.h>

#include <math.h>

using namespace std;

const int N = 500005;

const double PI = acos(-1.0);

struct Virt

{

double r, i;

Virt(double r = 0.0,double i = 0.0)

{

this->r = r;

this->i = i;

}

Virt operator + (const Virt &x)

{

return Virt(r + x.r, i + x.i);

}

Virt operator - (const Virt &x)

{

return Virt(r - x.r, i - x.i);

}

Virt operator \* (const Virt &x)

{

return Virt(r \* x.r - i \* x.i, i \* x.r + r \* x.i);

}

};

//雷德算法--倒位序

void Rader(Virt F[], int len)

{

int j = len >> 1;

for(int i=1; i<len-1; i++)

{

if(i < j) swap(F[i], F[j]);

int k = len >> 1;

while(j >= k)

{

j -= k;

k >>= 1;

}

if(j < k) j += k;

}

}

//FFT实现

void FFT(Virt F[], int len, int on)

{

Rader(F, len);

for(int h=2; h<=len; h<<=1) //分治后计算长度为h的DFT

{

Virt wn(cos(-on\*2\*PI/h), sin(-on\*2\*PI/h)); //单位复根e^(2\*PI/m)用欧拉公式展开

for(int j=0; j<len; j+=h)

{

Virt w(1,0); //旋转因子

for(int k=j; k<j+h/2; k++)

{

Virt u = F[k];

Virt t = w \* F[k + h / 2];

F[k] = u + t; //蝴蝶合并操作

F[k + h / 2] = u - t;

w = w \* wn; //更新旋转因子

}

}

}

if(on == -1)

for(int i=0; i<len; i++)

F[i].r /= len;

}

//求卷积

void Conv(Virt a[],Virt b[],int len)

{

FFT(a,len,1);

FFT(b,len,1);

for(int i=0; i<len; i++)

a[i] = a[i]\*b[i];

FFT(a,len,-1);

}

char str1[N],str2[N];

Virt va[N],vb[N];

int result[N];

int len;

void Init(char str1[],char str2[])

{

int len1 = strlen(str1);

int len2 = strlen(str2);

len = 1;

while(len < 2\*len1 || len < 2\*len2) len <<= 1;

int i;

for(i=0; i<len1; i++)

{

va[i].r = str1[len1-i-1] - '0';

va[i].i = 0.0;

}

while(i < len)

{

va[i].r = va[i].i = 0.0;

i++;

}

for(i=0; i<len2; i++)

{

vb[i].r = str2[len2-i-1] - '0';

vb[i].i = 0.0;

}

while(i < len)

{

vb[i].r = vb[i].i = 0.0;

i++;

}

}

void Work()

{

Conv(va,vb,len);

for(int i=0; i<len; i++)

result[i] = va[i].r+0.5;

}

void Export()

{

for(int i=0; i<len; i++)

{

result[i+1] += result[i]/10;

result[i] %= 10;

}

int high = 0;

for(int i=len-1; i>=0; i--)

{

if(result[i])

{

high = i;

break;

}

}

for(int i=high; i>=0; i--)

printf("%d",result[i]);

puts("");

}

int main()

{

while(~scanf("%s%s",str1,str2))

{

Init(str1,str2);

Work();

Export();

}

return 0;

}

//Java FastIO

import java.io.\*;

import java.util.\*;

import java.math.\*;

class Main{

static class Reader{

static BufferedReader reader;

static StringTokenizer tokenizer;

static void init(InputStream input){

reader=new BufferedReader(new InputStreamReader(input));

tokenizer= new StringTokenizer("");

}

static String next() throws IOException {

String s;

while(!tokenizer.hasMoreTokens()){

if((s=reader.readLine())==null) return null;

tokenizer = new StringTokenizer(s);

}

return tokenizer.nextToken();

}

static Integer nextInt() throws IOException {

String s=next();

if(s==null) return null;

return Integer.parseInt(s);

}

static Double nextDouble() throws IOException {

String s=next();

if (s==null) return null;

return Double.parseDouble(s);

}

}

public static void main(String[] args) throws IOException{

//InputStream f=new FileInputStream("/home/alex/Desktop/test.txt");

//Reader.init(f);

Reader.init(System.in);

//Scanner in= new Scanner(System.in);

int n;

while(true){

String s=Reader.next();

if(s==null) break;

R=new BigDecimal(s);

R=R.stripTrailingZeros();

//System.out.println(R.toString());

n=Reader.nextInt();

R=R.pow(n);

String S=R.toPlainString();

if(S.charAt(0)=='0') S=S.substring(1,S.length());

System.out.println(S);

}

}

}

* **并查集小整理**

1. **1.获取该连通分量的个数**

只需要在2个连通分量merge的时候，使fa[a] = b , sum[b] += sum[a]即可

查询x的节点所在连通分量中元素个数，即为 sum[findfa(x)]

1. **2. 获取到根节点的距离**

首先需要保证每个根节点在成为子节点后不会再成为新的根节点(即： 每次merge时候，要保证新的根节点的权值为0） ,merge过程只需要 fa[a] = b,并且使a的权值变成新的权值,

findfa这个过程：

int findfa(int x)

{

if(x == fa[x]) return x;

int t = findfa(fa[x]);

sum[x] += sum(fa[x]);

return fa[x] = t;

}

在下一次查询时，给a的后继节点都加上a的新权值

1. **3. 如hdu2818**

注意到，每次merge, a的后继节点应该加上b中连通分量的个数,所以需要综合运用1,2

**4. 并查集建立虚根**

HDU 3234

将所有确定的点都连通到虚根n上， 点i的权值表示i到

root的异或值， merge的时候,//我们保证根节点在一般情况下为0, 当加入节点时 a--x 需要xor res[x] b -- y需要 xor[y] 并且x -- y需要xor v 那么 a -- b需要xor res[x] xor res[y] xor v ，之后b的后继就可以进行更新

HDU2473

对并查集的删除操作(从连通分量中去除该点），通过建立虚根的方式，使用replace 数组

**5.并查集解决最小费用**

**例如Kruskal**

* **图论小结**

1. **最短路:**

1. K短路问题 ：首先在终点做一次Dijkstra,然后从起点开始使用a\_star进行扩展　（这里g(n)即为走过的路程,f(n)为当前点到终点的最短距离....实际上，可以只使用g(n)进行扩展，即再次Dijkstra ) ---------------------例题 POJ2449

2. 求最短路和次短路　在ｕｐｄａｔｅ时有４中状态

而且在HeapNdoe中要添加一个mode 记录当前是最短路还是次短路,同样done数组也许要添加１维

1. d[e.to][0] > d[u][mode] + e.w //更新e.to的最短路 2. d[e,to][0] == d[u][mode] + e.w //更新e.to最短路状态 3. d[e.to][1] > d[u][mode] + e.w //更新e.to的次短路 4. d[e.to][1] == d[u][mode] + e.w //更新e.to的次短路状态

3.最优比率正环(　正环一般都转化成负环 )

使用0,1规划 设ans = (F[1] + F[2] + F[3] + … F[n] ) / ( W[1] + W[2] + W[3] + … W[n] )

转化一下( F[1] – ans\*W[1] ) + ( F[2] – ans\*W[2] ) ….不难发现，当这个个式子的值 > 0时，ans非最优答案，还有更好的　　我们再转化一下(  ans\*W[1] – F[1] ) + ( ans\*W[2] - F[2] ) …...

那么当这个式子小于0,时还有更好的，　那么每次我们只需要找到使左边最小值，就可以知道ans是否是最优解（二分求最优解）　　所以只需要将边权值转化成 ans\*W[i] – F[i] 用bellman-ford判断是否存在负环即可 ---------- POJ3621

**图中判环**

1. 在无向图判环：并查集

2. 在有向图判环：　拓扑排序（如果不能，就是有环）

3.　有向图＋无向图: 先并查集判断，如果此时不存咋，就将一个连通分量缩成１个点，再来拓扑排序

4. 无向图中是否有奇数环: 非二分图必有奇数环

5. 无向图中是否有偶数环: 分离出所有的双连通分量,然后分别检测是否有偶环

(1) 如果是单环，直接判断即可

(2)如果是２个缠绕(贡献至少一条边）的环，如果这２个都是奇数环，那么可以通过去除中间的公共边，使得变成偶数环，所以这种情况必定存在偶数环

莫比乌斯反演一般都是来解决gcd(x,y) == a个数的问题

１．常规的求gcd(x,y) == a, 1 <= x <= n,1 <= y <= m

令f(i) 为　gcd(x,y) == i的个数

令g(i) 为　gcd(x,y) % i == 0的个数，那么g(i) = n/i;

除一下，即求gcd(x,y) == 1, 1<= x <= n/a, 1 <= y <= m/a的个数

那么就是:

for(int i = 1;i < = maxn;i++)

res += mu[i]\*n/i\*m/i;

2. 变种的求gcd(Ax,Ay) == a, 在数列中的个数

令f(i) 为　gcd(Ax,Ay) == i的个数

令g(i) 为　gcd(Ax,Ay) % i == 0的个数，那么g(i) = n/i;

首先先对数列中数进行处理，求出数列中能把A[i]整除的数的个数cnt[A[i]],这即为g(A[i])

然后类似的

for(int i = a;i <= maxn;i += a)

res += mu[i/a] \* cnt[i]\*cnt[i] //注意这里mu[i/a],类似前一中情况，而后面的cnt[i] 对应前面的n/i