#include<cstdio>

#include<cmath>

#include<cstring>

#include<cstdlib>

#include<iostream>

#include<algorithm>

#include<vector>

#include<queue>

#include<map>

#include<complex>

using namespace std;

typedef long long ll;

typedef complex<double> point;

double eps=1e-8;

const double PI=acos(-1.0);

double det(point a,point b){ return (conj(a)\*b).imag();}

//叉积，顺时针为负数，逆时针为正数|a|×|b|×sin(ang)

double dot(point a,point b){ return (a\*conj(b)).real();}

//点积|a|×|b|×cos(ang)

inline int sgn(double n){ return abs(n)<eps?0:(n<0?-1:1);}

point rotate (point u,double a )//坐标逆时针旋转a度

{

return point ( u.real() \* cos ( a ) -u.imag() \* sin ( a ) ,u.imag() \* cos ( a ) + u.real() \* sin ( a ) ) ;

}

point convertPoint(point u,point v,point c)

{//a\*u+b\*v=c,返回(a,b),u,v不平行

return point(det(c,v)/det(u,v),det(c,u)/det(v,u));

}

int cross(point s,point a,point b){

point u=a-s;

point v=b-s;

if(det(u,v)>0) return 1;//在线段的左方

if(det(u,v)<0) return -1;//在线段的右方

if(dot(u,v)<=0) return 0;//在线段上

if(abs(u)>abs(v)) return 2;//在a到b的延长线上

return -2;//在b到a的延长线上

}

struct line: public vector<point>

{

line(){}

line(point a,point b){ push\_back(a),push\_back(b);}

};

point vec(point a){ return a/abs(a);}

point vec(const line &a){ return a[1]-a[0];}

bool IsInterSS(const line &u,const line &v)//判断线段交

{

if(sgn(det(vec(u),vec(v)))==0)//平行

return false;

if(sgn(cross(u[0],u[1],v[0])\*cross(u[0],u[1],v[1]))<=0 && sgn(cross(v[0],v[1],u[0])\*cross(v[0],v[1],u[1]))<=0)//相交

return true;

return false;

}

point InterLL(const line &u,const line &v)//直线与直线的交

{

double a=det(vec(u),vec(v));

double b=det(vec(u),u[1]-v[0]);

if(sgn(a)==0) { // parallel

return v[0]; // sameline

}

return v[0]+b/a\*vec(v);

}

double area(const point &a,const point &b,const point &c)//三角形面积

{

return abs(det(a-c,b-c)/2);

}

#define MAXN 1001000

int top;

point q[MAXN],res[MAXN];

bool cmp(const point &a,const point &b)

{

if(sgn(det(a-res[0],b-res[0]))==0)

return sgn(norm(a-res[0])-norm(b-res[0]))<0;

return sgn(det(a-res[0],b-res[0]))>0;

}

bool getContex(int n)//可以判断边上的点

{

top=0;

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

{

if( sgn((res[0]-res[i]).imag())>0

|| (sgn((res[0]-res[i]).imag())==0 &&

sgn((res[0]-res[i]).real())>0) )

swap(res[i],res[0]);

}

sort(res+1,res+n,cmp);

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

{

while(top>1 && sgn(det(res[i]-q[top-1],q[top-1]-q[top-2]))>=0)//加等号就没边上的点

top--;

q[top++]=res[i];

}

if(n<3 || sgn(det(q[top-1]-res[0],res[1]-res[0]))==0) return false;

// point last=q[--top];//加入边上的点

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

// {

// if(det(last-res[0],res[i]-res[0])==0)>0) q[top++]=res[i];

// }

return true;

}

double rotating\_calipers(point p[],int n)//凸包的最大距离

{

double ans=0;

point v;

int cur=1;

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

{

v=p[i]-p[(i+1)%n];

while(sgn(det(v,p[(cur+1)%n]-p[cur]))<0)

cur=(cur+1)%n;

ans=max(ans,max(norm(p[i]-p[cur]),norm(p[(i+1)%n]-p[(cur+1)%n])));

}

return ans;

}

point CrossSP(const line &v,const point &u)//点到直线距离返回垂足

{

if(v[1]==v[0]) return v[0];

if(sgn(dot(vec(v),u-v[0]))<0) return v[0];//删两行是直线

if(sgn(dot(vec(v),u-v[1]))>0) return v[1];//只删这行是射线

double a=dot(vec(v),u-v[0]);

return v[0]+a/norm(vec(v))\*vec(v);

}

double minDisSS(const line &u,const line &v,bool flag=false)

{//flag=true这行是线段v到直线u的最短距离

double ans=min(abs(v[1]-CrossSP(u,v[1])),abs(v[0]-CrossSP(u,v[0])));

if(flag) return ans;

return min(ans,minDisSS(v,u,true));

}

double rotating\_calipers(point p[],int np,point q[],int nq,bool flag=false)

{

int sp=0,sq=0;

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

if(sgn((p[i]-p[sp]).imag())<0)

sp=i;

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

if(sgn((q[i]-q[sq]).imag())<0)

sq=i;

double tmp;

double ans=abs(p[sp]-q[sq]);

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

{

while(sgn(tmp=det(p[(sp+1)%np]-p[sp],q[sq]-q[(sq+1)%nq]))<0)

sq=(sq+1)%nq;

if(sgn(tmp)==0)//线段平行的时候

ans=min(ans,minDisSS(line(p[(sp+1)%np],p[sp]),line(q[(sq+1)%nq],q[sq])));

else ans=min(ans,abs(q[sq]-CrossSP(line(p[sp],p[(sp+1)%np]),q[sq])));

sp=(sp+1)%np;

}

if(flag) return ans;

return min(ans,rotating\_calipers(q,nq,p,np,!flag));

}

point getCenter(point pt[],int n)//求多边形重心，三角形外接圆心

{

double sum=0,area;

point res=point(0,0);

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

{

area=det(pt[i],pt[(i+1)%n]);

res=res+(pt[i]+pt[(i+1)%n])/3.0\*area;

sum+=area;

}

res=res/sum;

return res;

}

//向量（a，b）的左边的垂线为（-b，a），右边的垂线为（b，-a）

double norm(point a){return (a\*conj(a)).real();}//库函数

struct circle{

point c;

double r;

circle(){}

circle(point c,double r):c(c),r(r){}

};

//返回过圆外一点圆的切线

int CutlineCP(const circle &u,const point &p,line res[])//返回切线的条数

{

point v=p-u.c;

if(sgn(norm(v)-u.r\*u.r)<0) return 0;//点在圆内

else if(sgn(norm(v)-u.r\*u.r)==0){

res[0]=line(p,p+point(-v.imag(),v.real()));

return 1;//点在圆上，只有一条切线

}

double r=u.r/sqrt(norm(v)-u.r\*u.r);

res[0]=line(point((v.real()\*r\*r-v.imag()\*r)/(1+r\*r),(v.imag()\*r\*r+v.real()\*r)/(1+r\*r))+u.c,p);//在从圆心到一点的向量的左边

res[1]=line(point((v.real()\*r\*r+v.imag()\*r)/(1+r\*r),(v.imag()\*r\*r-v.real()\*r)/(1+r\*r))+u.c,p);//在从圆心到一点的向量的右边

return 2;

}

line MiddleLine(const line &u,int flag=1)//线段的中位线

{//默认u向量左边

point p=(double)flag\*point(-vec(u).imag(),vec(u).real());

point middle=(u[1]+u[0])/2.0;

return line(middle,middle+p);

}

typedef vector<point> polygon;

circle TriangleOuterCircle(const polygon &u)//三角形的外切圆

{

point center=InterLL(MiddleLine(line(u[0],u[1])),MiddleLine(line(u[1],u[2])));

return circle(center,abs(center-u[0]));

}

int InterCC(circle u,circle v,point &ret1,point &ret2)//判断两个圆相交返回交点数量

{

point d=u.c-v.c;

double r1=u.r;

double r2=v.r;

if(sgn(r1+r2-abs(d))>=0&&sgn(abs(r1-r2)-abs(d))<=0)

{

point mid=d/abs(d)\*(norm(d)+r2\*r2-r1\*r1)/(2\*r2\*abs(d))\*r2+v.c;

point p=point(d.imag(),-d.real());

if(abs(p)!=0)p=p/abs(p)\*sqrt(r2\*r2-norm(mid-v.c));

ret1=mid+p;

ret2=mid-p;

if(sgn(r1+r2-abs(d))==0||sgn(abs(r1-r2)-abs(d))==0) return 1;

return 2;

}

return 0;

}

int InterCL(const circle &u,const line &v,point &ret1,point ret2)

{//直线与圆的交点

point d=CrossSP(v,u.c);

int k=sgn(abs(d-u.c)-u.r);

if(k>0)return 0;

if(k==0){

ret1=d;

return 1;

}

double h=sqrt(u.r\*u.r-norm(d-u.c));

point p=point((d-u.c).imag(),-(d-u.c).real());

p=vec(p);

ret1=d+p\*h;

ret2=d-p\*h;

return 2;

}

bool IsInterPS(const line &u,point v)//点和线段相交

{

if(sgn(det(u[0]-v,u[1]-v))==0 &&

sgn(v.real()-min(u[0].real(),u[1].real()))>=0 && sgn(v.real()-max(u[0].real(),u[1].real()))<=0 &&

sgn(v.imag()-min(u[0].imag(),u[1].imag()))>=0 && sgn(v.imag()-max(u[0].imag(),u[1].imag()))<=0 )

return true;

return false;

}

bool IsPInPolygon(point u,point p[],int n)//判断点在多边形内部

{

int cnt=0;

point far;

if(sgn(u.real())!=0)far=point(0,u.imag());

else far=point(1,u.imag());

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

{

int j=(i+1)%n;

line v=line(p[i],p[j]);

if(IsInterPS(v,u))return true;

int d1=sgn(det(far-u,p[i]-u));

int d2=sgn(det(far-u,p[j]-u));

int d3=sgn(det(p[j]-p[i],u-p[j]));

int d4=sgn(det(p[j]-p[i],far-p[j]));

if(d1\*d2<0 && d3\*d4 <0) cnt++;

else if( (d1\*d2==0&&d3\*d4<0) &&

sgn(u.imag()-max(p[i].imag(),p[j].imag()))==0 )

cnt++;

}

if(cnt&1)return true;

return false;

}

int cmp(const point &a,const point &b)//整点的极角排序，平行的按X轴排序

{

if(a.imag()\*b.imag()<0)

return a.imag()<b.imag();

if(det(a,b)==0) return a.real()<b.real();

return det(a,b)>0;

}

void getCrossCircle(point p[],int n,point &s,double &r)//最小圆覆盖

{

double step=inf;

r=inf;

s=point(0,0);

while(step>eps)

{

int t=0;

double maxDis=abs(p[t]-s);

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

{

if(maxDis<abs(p[i]-s))

{

maxDis=abs(p[i]-s);

t=i;

}

}

r=min(r,maxDis);

s=s+(p[t]-s)/maxDis\*step;

step\*=0.93;

}

}

//下面为圆交多边形的板子

double outer(point a,point b,point c){

return det(a-c,b-c);

}

double inner(point a,point b,point c){

return dot(a-c,b-c);

}

double sqr(double x){return x\*x;}

double calc(point a,point b,point c,double r)

{

double A,B,C,x,y,tS;

A=abs(b-c);

B=abs(a-c);

C=abs(b-a);

if(A<r&&B<r)

return outer(a,b,c)/2;

else if(A<r&&B>=r )

{

x=(inner(a,c,b)+sqrt(sqr(r)\*sqr(C)-sqr(outer(a,c,b))))/C;

tS=outer(a,b,c)/2;

return asin(tS\*(1-x/C)\*2/r/B)\*sqr(r)/2+tS\*x/C;

}

else if(A>=r&&B<r)

{

y=(inner(b,c,a)+sqrt(sqr(r)\*sqr(C)-sqr(outer(b,c,a))))/C;

tS=outer(a,b,c)/2;

return asin(tS\*(1-y/C)\*2/r/A)\*sqr(r)/2+tS\*y/C;

}

else if ( fabs ( outer (a ,b , c ) ) >= r \* C || inner (b ,c , a ) <=0|| inner (a ,c , b ) <=0)

{

if(inner(a,b,c)<0)

{

if(outer(a,b,c)<0)

return (-PI-asin(outer(a,b,c)/A/B))\*sqr(r)/2;

else return (PI-asin(outer(a,b,c)/A/B))\*sqr(r)/2;

}

else return asin(outer(a,b,c)/A/B)\*sqr(r)/2;

}

else{

x=(inner(a,c,b)+sqrt(sqr(r)\*sqr(C)-sqr(outer(a,c,b))))/C;

y=(inner(b,c,a)+sqrt(sqr(r)\*sqr(C)-sqr(outer(b,c,a))))/C;

tS=outer(a,b,c)/2;

return (asin(tS\*(1-x/C)\*2/r/B)+asin(tS\*(1-y/C)\*2/r/A))\*sqr(r)/2+tS\*((y+x)/C-1);

}

}

//计算一般多边形与圆的交面积(将多边形划分为三角形,然后有向三角形与圆求有向面积交

double solve(point p[],int n,point o,double r)

{

double res=0,sum;

point tri[3];

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

{

tri[0]=p[0];

tri[1]=p[i];

tri[2]=p[i+1];

sum=0;

for(int j=0;j<3;j++)

sum+=calc(tri[j],tri[(j+1)%3],o,r);

//sum为三角形与圆交的有向面积

res+=sum;

}

return abs(res);

}

//下面为半平面交

#include<cstdio>

#include<cmath>

#include<cstring>

#include<cstdlib>

#include<iostream>

#include<algorithm>

#include<vector>

#include<queue>

#include<map>

#include<complex>

#include<ctime>

using namespace std;

#define MAXN 1100

const double eps=1e-8;

const double PI=acos(-1.0);

typedef complex<double> point;

inline int sgn(double n){ return abs(n)<eps?0:(n<0?-1:1);}

double det(point a,point b){return (conj(a)\*b).imag();}//顺时针为负数，逆时针为正数

double dot(point a,point b){return (a\*conj(b)).real();}

struct line: public vector<point>

{

double k;

line(){}

line(point a,point b)

{

push\_back(a),push\_back(b);

k=atan2((b-a).imag(),(b-a).real());

}

};

point vec(point a){return a/abs(a);}

point vec(const line &a){return a[1]-a[0];}

point InterLL(const line &u,const line &v)//直线与直线的交

{

double a=det(vec(u),vec(v));

double b=det(vec(u),u[1]-v[0]);

if(sgn(a)==0) { // parallel

return v[0]; // sameline

}

return v[0]+b/a\*vec(v);

}

bool HPIcmp(const line &a,const line &b)

{

if(sgn(a.k-b.k)!=0) return a.k<b.k;

return sgn(det(a[0]-b[0],vec(b)))<0;

}

line Q[MAXN];

int HPI(line l[],int n,point res[])

{

sort(l,l+n,HPIcmp);

int tot=1;

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

if(sgn(l[i].k-l[i-1].k)!=0) l[tot++]=l[i];

int head=0,tail=1;

Q[0]=l[0],Q[1]=l[1];

int resn=0;

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

{

if(sgn(det(vec(Q[tail]),vec(Q[tail-1])))==0

|| sgn(det(vec(Q[head]),vec(Q[head+1])))==0)

return -1;

while(head<tail && sgn(det(InterLL(Q[tail],Q[tail-1])-l[i][0],vec(l[i])))>0)

tail--;

while(head<tail && sgn(det(InterLL(Q[head],Q[head+1])-l[i][0],vec(l[i])))>0)

head++;

Q[++tail]=l[i];

}

while(head<tail && sgn(det(InterLL(Q[tail],Q[tail-1])-Q[head][0],vec(Q[head])))>0)

tail--;

while(head<tail && sgn(det(InterLL(Q[head],Q[head+1])-Q[tail][0],vec(Q[tail])))>0)

head++;

if(tail<=head)return 0;

for(int i=head;i<tail;i++)

res[resn++]=InterLL(Q[i],Q[i+1]);

if(head<tail-1)

res[resn++]=InterLL(Q[head],Q[tail]);

return resn;

}

//下面为3维平面模版

#include<cstdio>

#include<cmath>

#include<cstring>

#include<cstdlib>

#include<iostream>

#include<algorithm>

#include<vector>

#include<queue>

#include<map>

#include<complex>

#include<set>

#include<assert.h>

#include<bitset>

using namespace std;

const double eps=1e-10;

int sgn(double n){return fabs(n)<eps?0:n>0?1:-1;}

struct spt{

double x,y,z;

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

spt operator + (const spt &a)const{return spt(x+a.x,y+a.y,z+a.z);}

spt operator - (const spt &a)const{return spt(x-a.x,y-a.y,z-a.z);}

double operator \* (const spt &a)const{return x\*a.x+y\*a.y+z\*a.z;}

spt operator \* (const double &a)const{return spt(x\*a,y\*a,z\*a);}

spt operator / (const double &a)const{return spt(x/a,y/a,z/a);}

spt operator ^ (const spt &a)const{return spt(y\*a.z-z\*a.y,z\*a.x-x\*a.z,x\*a.y-y\*a.x);}

void in(){scanf("%lf%lf%lf",&x,&y,&z);}

void out(){printf("%lf %lf %lf\n",x,y,z);}

}O;

struct line: public vector<spt>{

line(){};

line(spt a,spt b){

push\_back(a),push\_back(b);

}

};

typedef line face;

spt vec(const line &a){return a[1]-a[0];}

double norm(spt a){return sqrt(a\*a);}

bool isFL(face &u,line &v,spt &ret)//线面相交，ret是交点

{

double a=vec(u)\*(v[1]-u[0]),b=vec(u)\*(v[0]-u[0]);

if(sgn(a-b)==0)return false;

ret=(v[0]\*a-v[1]\*b)/(a-b);

return true;

}

bool isFF(face &u,face &v,line &ret)//面面相交，ret是相交直线

{

spt k=vec(u)^vec(v);

if(sgn(norm(k))==0) return false;

spt p=vec(u)^k;

p=u[0]+p\*( vec(v)\*(v[0]-u[0]))/(vec(v)\*p);

ret=line(p,p+k);

return true;

}

double disLP(line &u,spt &v,spt &ret)//点到直线的距离，ret是垂足

{

ret=u[0]+vec(u)\*( vec(u)\*(v-u[0])/norm(vec(u)))/norm(vec(u));

return fabs(vec(u)\*(v-u[0])/norm(vec(u))) ;

}

double disLL(line &u,line &v,spt &ret)//直线之间的距离，ret为u上的垂足

{

spt tmp=vec(u)^vec(v);

if(sgn(norm(tmp)==0))

return disLP(u,v[0],ret);

assert(isFL(face(v[0],(vec(u)^vec(v))^vec(v)),u,ret)==0);

return fabs(tmp\*(u[0]-v[0])/norm(tmp));

}

const double PI=acos(-1.0);//两个球的体积并

double CircleCombination(spt u,double ur,spt v,double vr)

{

double d,R,r,p,l,H,h,res;

R=max(ur,vr),r=min(ur,vr);

d=norm(u-v);

res=PI\*(R\*R\*R+r\*r\*r)\*4/3;

if(R+r>d)

{

if(R-r<=d)

{

p=(R+r+d)/2;

l=sqrt(p\*(p-R)\*(p-r)\*(p-d))\*2/d;

H=sqrt(R\*R-l\*l);

h=sqrt(r\*r-l\*l);

res-=PI\*(R\*R\*R\*2/3-R\*R\*H+H\*H\*H/3);

if(R\*R-r\*r<=d\*d)

res-=PI\*(r\*r\*r\*2/3-r\*r\*h+h\*h\*h/3);

else res-=PI\*r\*r\*r\*4/3-PI\*(r\*r\*r\*2/3-r\*r\*h+h\*h\*h/3);

}

else res-=PI\*r\*r\*r\*4/3;

}

return res;

}

树链剖分的模板：

#include<cstdio>

#include<cstring>

#include<cmath>

#include<algorithm>

#include<vector>

#include<queue>

#include<iostream>

using namespace std;

#define MAXN 100100

#define MOD 1000000007

typedef long long ll;

int n,m;

vector<int> G[MAXN];

int top[MAXN],siz[MAXN],son[MAXN],par[MAXN],depth[MAXN],out[MAXN],in[MAXN];

ll w[MAXN];

int tot;

void dfs1(int u,int pre)

{

par[u]=pre;

siz[u]=1;

int tp=0;

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

{

int v=G[u][i];

if(v!=pre)

{

dfs1(v,u);

siz[u]+=siz[v];

if(siz[v]>siz[tp])

tp=v;

}

}

if(tp) son[u]=tp;

}

void dfs2(int u,int deep,int high)

{

depth[u]=deep;

top[u]=high;

in[u]=++tot;

//printf("%d %d\n",u,in[u]);

if(son[u]>0)

dfs2(son[u],deep+1,high);

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

{

int v=G[u][i];

if(v!=son[u]&&v!=par[u])

{

dfs2(v,deep+1,v);

}

}

out[u]=tot;

}

void update(int x,ll val,ll \*sum)

{

while(x<=n)

{

sum[x]+=val;

sum[x]%=MOD;

x+=x&(-x);

}

}

ll getsum(int x,ll \*sum)

{

ll ret=0;

while(x>0)

{

ret+=sum[x];

ret%=MOD;

x-=x&(-x);

}

return ret;

}

ll sum\_all[MAXN],sum\_double[MAXN];

void init()

{

siz[0]=0;

tot=son[0]=0;

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

{

G[i].clear();

son[i]=-1;

sum\_all[i]=0;

sum\_double[i]=0;

w[i]=0;

}

}

void change(int x,ll val)

{

int u=x;

u=top[u];

while(par[u]>0)

{

ll all=getsum(out[u],sum\_all)-getsum(in[u]-1,sum\_all);

ll update\_val=(val-w[x])\*(val-w[x])%MOD+2\*all\*(val-w[x])%MOD;

update(in[par[u]],update\_val%MOD,sum\_double);

u=top[par[u]];

}

update(in[x],-w[x],sum\_all);

update(in[x],val,sum\_all);

w[x]=val;

}

int inqury(int x,int y)

{

ll ret=0;

int pre=0;

while(top[x]!=top[y])

{

if(depth[top[x]]<depth[top[y]]) swap(x,y);

ret+=getsum(in[x],sum\_double)-getsum(in[top[x]]-1,sum\_double);

if(son[x]>0)

{

ll all=(getsum(out[son[x]],sum\_all)-getsum(in[son[x]]-1,sum\_all))%MOD;

ret+=all\*all;

ret%=MOD;

}

if(pre>0)

{

ll all=(getsum(out[pre],sum\_all)-getsum(in[pre]-1,sum\_all))%MOD;

ret-=all\*all;

ret%=MOD;

}

pre=top[x];

x=par[top[x]];

}

if(depth[x]>depth[y]) swap(x,y);

ret+=getsum(in[y],sum\_double)-getsum(in[x]-1,sum\_double);

ret%=MOD;

if(son[y]>0)

{

ll all=(getsum(out[son[y]],sum\_all)-getsum(in[son[y]]-1,sum\_all))%MOD;

ret+=all\*all;

ret%=MOD;

}

if(par[x]>0)

{

ll all=(getsum(out[1],sum\_all)-getsum(out[x],sum\_all)+getsum(in[x]-1,sum\_all))%MOD;

ret+=all\*all;

ret%=MOD;

}

if(pre>0)

{

ll all=(getsum(out[pre],sum\_all)-getsum(in[pre]-1,sum\_all))%MOD;

ret-=all\*all;

ret%=MOD;

}

return ret;

}

int main()

{

//freopen("in","r",stdin);

while(~scanf("%d%d",&n,&m))

{

init();

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

scanf("%I64d",&w[i]);

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

{

int a,b;

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

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

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

}

dfs1(1,-1);

dfs2(1,1,1);

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

{

ll tp=w[i];

w[i]=0;

change(i,tp);

}

//printf("%d\*\*\*\n",getsum(in[5],sum\_double)-getsum(in[5]-1,sum\_double));

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

{

int kind;

ll all=getsum(out[1],sum\_all);

scanf("%d",&kind);

if(kind==1)

{

int u;

ll val;

scanf("%d%I64d",&u,&val);

change(u,val);

}

else{

int x,y;

scanf("%d%d",&x,&y);

printf("%d\n",((all\*all%MOD-inqury(x,y))%MOD+MOD)%MOD);

}

}

}

}

//极角旋转

#include<cstdio>

#include<cmath>

#include<cstring>

#include<cstdlib>

#include<iostream>

#include<algorithm>

#include<vector>

#include<queue>

#include<map>

#include<complex>

#include<list>

#include<set>

#include<assert.h>

#include<bitset>

#define lson l,mid,rt<<1

#define rson mid+1,r,rt<<1|1

using namespace std;

typedef long long ll;

typedef complex<ll> point;

ll det(point a,point b){return (conj(a)\*b).imag();}

ll dot(point a,point b){return (a\*conj(b)).real();}

void out(point a){printf("point : %lld %lld\n",a.real(),a.imag());}

#define MAXN 3000

#define eps 1e-10

point sav[MAXN],tmp[MAXN];

int cmp(const point &a,const point &b)

{

if(a.imag()\*b.imag()<0)

return a.imag()<b.imag();

if(det(a,b)==0) return a.real()<b.real();

return det(a,b)>0;

}

int main()

{

freopen("in.txt","r",stdin);

int n;

while(~scanf("%d",&n))

{

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

{

int a,b;

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

sav[i]=point(a,b);

}

ll ans=1ll\*n\*(n-1)\*(n-2)/6;

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

{

int cnt=0;

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

if(i!=o)tmp[cnt++]=sav[i]-sav[o];

sort(tmp,tmp+cnt,cmp);

int l=0,r=0,next;

ll ret=0;

// cout<<o<<endl;

// out(sav[o]);

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

{

while(dot(tmp[i],tmp[r])<0) r=(r-1+cnt)%cnt;

while(dot(tmp[i],tmp[l])<0) l=(l+1)%cnt;

for(next=(r+1)%cnt;dot(tmp[i],tmp[next])>=0;next=(r+1)%cnt)

{

ll t=det(tmp[r],tmp[next]);

if(t>0 || (t==0&&tmp[next].real()>tmp[r].real()) ) r=next;

else break;

}

for(next=(l-1+cnt)%cnt;dot(tmp[i],tmp[next])>=0;next=(l-1+cnt)%cnt)

{

ll t=det(tmp[l],tmp[next]);

if(t<0 || (t==0&&tmp[next].real()<tmp[l].real()) ) l=next;

else break;

}

// cout<<"i: "<<i<<endl;

// out(tmp[i]);

// cout<<l<<" "<<r<<endl;

ret+=cnt-(((r-l)%cnt+cnt)%cnt+1);

}

// cout<<"ret: "<<ret<<endl<<endl;

ans-=ret/2;

}

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

}

}

//求最小球覆盖

int max\_dis(spt p[],int n,spt s)

{

int res=0;

double maxDis=0,tmp;

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

{

tmp=norm(p[i]-s);

if(maxDis<tmp)

{

maxDis=tmp;

res=i;

}

}

return res;

}

void getCenter(spt p[],int n,spt &s,double &r)

{

s=spt(0,0,0);//可以先取平均值

r=100000;

double step=100000;//选取最大坐标范围

while(step>eps)

{

int t=max\_dis(p,n,s);

double tmp=norm(p[t]-s);

if(r>tmp) r=tmp;

s=s+(p[t]-s)/tmp\*step;

step\*=0.9993;//系数的选取根据具体精度调整

}

}

//三维凸包

# include<iostream>

# include <cstdio>

# include <cstring>

# include <cmath>

# include <algorithm>

# define N 505

# define eps 1e-8

using namespace std ;

struct Point

{

double x ,y , z ;

Point () {}

Point ( double px , double py , double pz ) : x ( px ) ,y ( py ) ,z ( pz ) {}

Point operator - ( const Point p )

{

return Point (x - p .x ,y - p .y ,z - p . z ) ;

}

Point operator + ( const Point p )

{

return Point (x + p .x ,y + p .y ,z + p . z ) ;

}

Point operator / ( const double p )

{

return Point (x / p ,y / p ,z / p ) ;

}

Point operator \* ( const Point p )

{

return Point ( y \* p .z - z \* p .y , z \* p .x - x \* p .z , x \* p .y - y \* p . x ) ;

}

double operator ^ ( const Point p )

{

return x \* p . x + y \* p . y + z \* p . z ;

}

double len()

{

return sqrt(x\*x+y\*y+z\*z);

}

};

struct ConvexPolygon3D

{

struct Face

{

int a ,b , c ;

bool flag ;

};

int n ;

Point pt [ N ];

int tri\_num ;

Face face [8\* N ];

int g [ N ][ N ];

double veclen ( const Point & p )

{

return sqrt ( p . x \* p . x + p . y \* p . y + p . z \* p . z ) ;

}

Point cross ( const Point &a , const Point &b , const Point & c )

{

return Point (( b .y - a . y ) \*( c .z - a . z ) -( b .z - a . z ) \*( c .y - a . y ) , -(( b .x - a . x ) \*( c .

z - a . z ) -( b .z - a . z ) \*( c .x - a . x ) ) ,( b .x - a . x ) \*( c .y - a . y ) -( b .y - a . y ) \*( c .x - a .

x));

}

double tri\_area ( Point a , Point b , Point c )

{

return veclen (( a - c ) \*( b - c ) ) /2;

}

double tetrahedron\_volume ( Point a , Point b , Point c , Point d )

{

return (( b - a ) \*( c - a ) ^( d - a ) ) /6;

}

double dlcmp ( Point &p , Face & f )

{

Point m = pt [ f . b ] - pt [ f . a ];

Point n = pt [ f . c ] - pt [ f . a ];

Point t =p - pt [ f . a ];

return ( m \* n ) ^ t ;

}

void deal ( int a , int b , int p )

{

int f = g [ a ][ b ];

Face add ;

if ( face [ f ]. flag )

{

if ( dlcmp ( pt [ p ] , face [ f ]) > eps )

dfs (p , f ) ;

else

{

add . a = b ;

add . b = a ;

add . c = p ;

add . flag =1;

g [ p ][ b ]= g [ a ][ p ]= g [ b ][ a ]= tri\_num ;

face [ tri\_num ++]= add ;

}

}

}

void dfs ( int p , int now )

{

face [ now ]. flag =0;

deal ( face [ now ]. b , face [ now ]. a , p ) ;

deal ( face [ now ]. c , face [ now ]. b , p ) ;

deal ( face [ now ]. a , face [ now ]. c , p ) ;

}

bool same ( int s , int t )

{

Point & a = pt [ face [ s ]. a ];

Point & b = pt [ face [ s ]. b ];

Point & c = pt [ face [ s ]. c ];

bool res = fabs ( tetrahedron\_volume (a ,b ,c , pt [ face [ t ]. a ]) ) < eps &&

fabs ( tetrahedron\_volume (a ,b ,c , pt [ face [ t ]. b ]) ) < eps &&

fabs ( tetrahedron\_volume(a ,b ,c , pt [ face [ t ]. c ]) ) < eps ;

return res ;

}

void solve ()

{

int i ,j , tmp ;

Face add ;

bool flag ;

tri\_num =0;

if (n <4)

return ;

flag = true ;

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

if ( veclen (( pt [0] - pt [1]) \*( pt [1] - pt [ i ]) ) > eps )

{

swap ( pt [2] , pt [ i ]) ;

flag = false ;

break ;

}

if ( flag )

return ;

flag = true ;

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

if ( fabs (( pt [0] - pt [1]) \*( pt [1] - pt [2]) ^( pt [0] - pt [ i ]) ) > eps )

{

swap ( pt [3] , pt [ i ]) ;

flag = false ;

break ;

}

if ( flag )

return ;

flag = true ;

for ( i =3; i < n ; i ++)

if ( veclen ( pt [0] - pt [ i ]) > eps )

{

swap ( pt [1] , pt [ i ]) ;

flag = false ;

break ;

}

if ( flag )

return ;

for ( i =0; i <4; i ++)

{

add . a =( i +1) %4;

add . b =( i +2) %4;

add . c =( i +3) %4;

add . flag = true ;

if ( dlcmp ( pt [ i ] , add ) >0)

swap ( add .b , add . c ) ;

g [ add . a ][ add . b ]= g [ add . b ][ add . c ]= g [ add . c ][ add . a ]= tri\_num ;

face [ tri\_num ++]= add ;

}

for ( i =4; i < n ; i ++)

for ( j =0; j < tri\_num ; j ++)

if ( face [ j ]. flag && dlcmp ( pt [ i ] , face [ j ]) > eps )

{

dfs (i , j ) ;

break ;

}

tmp = tri\_num ;

for ( i = tri\_num =0; i < tmp ; i ++)

if ( face [ i ]. flag )

face [ tri\_num ++]= face [ i ];

}

double area ()

{

double res =0;

if ( n ==3)

{

Point p = cross ( pt [0] , pt [1] , pt [2]) ;

res = veclen ( p ) /2;

}

else

{

for ( int i =0; i < tri\_num ; i ++)

res += tri\_area ( pt [ face [ i ]. a ] , pt [ face [ i ]. b ] , pt [ face [ i ]. c ]) ;

}

return res ;

}

double volume ()

{

double res =0;

Point tmp (0 ,0 ,0) ;

for ( int i =0; i < tri\_num ; i ++)

res += tetrahedron\_volume( tmp , pt [ face [ i ]. a ] , pt [ face [ i ]. b ] , pt [ face [

i ]. c ]) ;

return fabs ( res ) ;

}

Point get\_center () //凸包重心

{

Point res (0 ,0 ,0) ,o (0 ,0 ,0) ,p ;

double sum , vol ;

int i ;

sum =0;

for ( i =0; i < tri\_num ; i ++)

{

vol = tetrahedron\_volume (o , pt [ face [ i ].a ] , pt [ face [ i ].b ] , pt [ face[ i ].c ]) ;

sum += vol ;

p =( pt [ face [ i ].a ]+ pt [ face [ i ].b ]+ pt [ face [ i ].c ]) /4;

p . x \*= vol ;

p . y \*= vol ;

p . z \*= vol ;

res = res + p ;

}

res = res / sum ;

return res ;

}

int triangle\_num ()

{

return tri\_num ;

}

int polygon\_num ()

{

int i ,j , res , flag ;

res =0;

for ( i =0; i < tri\_num ; i ++)

{

flag =1;

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

if ( same (i , j ) )

{

flag =0;

break ;

}

res += flag ;

}

return res ;

}

};

ConvexPolygon3D hull ;

//点p到平面abc的距离

double dis\_point\_face ( Point p , Point a , Point b , Point c )

{

Point vec =( b - a ) \*( c - a ) ;

Point t =a - p ;

double tmp =( vec ^ t ) /( vec . len () \* t . len () ) ;

return fabs ( t . len () \* tmp ) ;

}

int main ()

{

int i ;

while ( scanf ("% d " ,& hull . n ) != EOF )

{

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

scanf ("% lf % lf % lf " ,& hull . pt [ i ]. x ,& hull . pt [ i ]. y ,& hull . pt [ i ]. z ) ;

hull . solve () ;

printf ("%.3 f \ n " , hull . area () ) ;

}

return 0;

}