----------**MinCostMaxFlow**----------

```
struct Arc
{
   int dest,cost,rest;
   Arc *rev,*next;
   Arc(){};
   Arc(int dest,int cost,int rest,Arc *next):
      dest(dest),cost(cost),rest(rest),next(next){};
}Npool[maxE],*Nptr(Npool);

Arc *adj[maxV],*preE[maxV];
Queue<int,maxV>q;
int dis[maxV],preV[maxV];
int mincost(0),maxflow(0);
bool v[maxV];

bool spfa()
{
   memset(dis,inf,sizeof(dis));
   memset(v,0,sizeof(v));
   while(!q.empty()) q.pop();
   dis[S]=0; v[S]=1; q.push(S);
   while(!q.empty())
   {
      int id=q.front();
      q.pop(); v[id]=0;
      for(Arc *p=adj[id];p;p=p->next)
      {
         if(p->rest>0)
         {
            int temp=dis[id]+p->cost;
            if(dis[p->dest]>temp)
            {
               dis[p->dest]=temp;
               preV[p->dest]=id; preE[p->dest]=p;
               if(!v[p->dest]) v[p->dest]=1, q.push(p->dest);
            }
         }
      }
   }
   return dis[T]!=inf;
}
void aug()
{
   int tflow=inf;
   for(int i=T;i!=S;i=preV[i])
      upd_min(tflow,preE[i]->rest);
   for(int i=T;i!=S;i=preV[i])
   {
      preE[i]->rest-=tflow;
      preE[i]->rev->rest+=tflow;
   }
   maxflow+=tflow;
   mincost+=dis[T]*tflow;
   return ;
}
void MinCostMaxFlow()
{
   while(spfa()) aug();
   return ;
}
```

----------**dinic**----------

```
bool bfs()
{
    memset(v,0,sizeof(v));    memset(dis,0xff,sizeof(dis));
    dis[S]=13;    q.clear();    q.push(S);
    while(!q.empty())
    {
        int id=q.front();    q.pop();
        for(Arc *p=adj[id];p;p=p->next)
        {
            int j=p->dest;
            if(p->rest && dis[j]==-1)
            {
                dis[j]=dis[id]+1;
                if(j==T)return true;
                q.push(j);
            }     }     } return false;    }
int aug(int i,int m=inf)
{
    if(i==T)return m;
    int ret(0);
    for(Arc *&p=cur[i];p;p=p->next)
    {
        int j=p->dest;
        if(p->rest && !v[j] && dis[i]==dis[j]-1)
            if(int a=aug(j,min(m-ret,p->rest)))
            {
                ret+=a;    p->rest-=a;    p->rev->rest+=a;
                if(ret==m)return ret;
            }
    }
    if(ret==0)dis[i]=-1;
    v[i]=1;
    return ret;
}
void MaxFlow()
{
    flow=0;
    while(bfs())
    {
        for(int i=0;i<maxV;i++)    cur[i]=adj[i];
        flow+=aug(S);
    }     }
```

----------**Hungary**----------

```
bool find(int id)
{
    for(Arc *p=adj[id];p;p=p->next)
    {
        int j=p->dest;
        if(!used[j])
        {
            used[j]=1;
            if(ilink[j]==-1 || find(ilink[j])){ ilink[j]=i; return true; }
        }
    } return false;
}
```

----------**tarjan**----------

```
int low[maxN],dfn[maxN],Belong[maxN];
bool v[maxN];
void tarjan(int id)
{
    dfn[id]=low[id]=++ind;
```

```
    v[id]=true; s.push(id);
    int tt;
    for(Arc* p=adj[id];p;p=p->next)
    {
        tt=p->dest;
        if(!dfn[tt])
        {
            tarjan(tt);
            if(low[tt]<low[id]) low[id]=low[tt];
        }
        else if(v[tt] && dfn[tt]<low[id]) low[id]=dfn[tt];
    }
    if(dfn[id]==low[id])
    {
        Bcnt++;    tt=0;
        while(tt!=id)
        {
            tt=s.top();
            s.pop(); v[tt]=false;
            Belong[tt]=Bcnt;
        }
    }
    return ;
}
void solve()
{
    for(int i=1;i<=N;i++)    if(!dfn[i])    tarjan(i);
    return ;
}
```

----------KMP----------

```
for(int i=N;i>=1;i--) n[i]=n[i-1];
for(int i=M;i>=1;i--) m[i]=m[i-1];
int ind=0; p[1]=0;

for(int i=2;i<=M;i++){
    while(ind>0 && m[ind+1]!=m[i])ind=p[ind];
    if(m[ind+1]==m[i])ind++;
    p[i]=ind;
}
ind=0;
int ans=0;
for(int i=1;i<=N;i++)
{
    while(ind>0 && m[ind+1]!=n[i])ind=p[ind];
    if(m[ind+1]==n[i])ind++;
    if(ind==M)
    {
        ans++;
        ind=p[ind];
    }
}
```

----------manacher----------

```
void manacher()
{
    L=strlen(s); int N=2*L+2;
    t[0]='$';t[1]='#';
    for(int i=L-1;i>=0;i--) t[i*2+2]=s[i] , t[i*2+3]='#';
    t[L*2+2]=0;
    int ind=0;
    for(int i=1;i<N;i++)
    {
        if(i>=ind+p[ind]) p[i]=1;
```

```
        else p[i]=min(p[ind*2-i],p[ind]-i+ind);
        for( ; t[i+p[i]]==t[i-p[i]]; p[i]++);
        if(p[i]+i>ind+p[ind])ind=i;
    }   }
----------suffix array----------
int SA[maxL],R[maxL],tmp[maxL],freq[maxL],height[maxL];
int cmp_S(const void* a, const void* b){return S[*(int*)a]-S[*(int*)b];}
void get_SA()
{
    int tie_n;
    for(int i=0;i<L;++i) SA[i]=i;
    qsort(SA,L,sizeof(int),cmp_S);
    tie_n=0;
    for(int i=0;i<L;++i)
    {
        int a=SA[i],pa=SA[i-1];
        if(!i || S[a]!=S[pa])R[a]=i+1;
        else R[a]=R[pa], ++tie_n;
    }
    for(int k=1;k<L && tie_n;k<<=1)
    {
        int p=0;
        for(int i=L-k;i<L;++i)tmp[p++]=i;
        for(int i=0;i<L;++i) if(SA[i]>=k) tmp[p++]=SA[i]-k;
        memset(freq, 0, (L+1)*sizeof(int));
        for(int i=0;i<L;++i)++freq[R[i]];
        for(int i=1;i<=L;++i)freq[i]+=freq[i-1];
        for(int i=L-1;i>=0;--i)SA[--freq[R[tmp[i]]]]=tmp[i];
        memcpy(tmp, R, L*sizeof(int));
        tie_n=0;
        for(int i=0;i<L;++i)
        {
            int a=SA[i], pa=SA[i-1];
            int b=a+k, pb=pa+k;
            if(!i || tmp[a]!=tmp[pa] || b>=L || pb>=L || tmp[b]!=tmp[pb]) R[a]=i+1;
            else R[a]=R[pa], ++tie_n;
        }
    }
}
void get_height()
{
    int h=0;
    for(int i=0;i<L;++i)
    {
        if(R[i]==1)continue;
        int pi=SA[R[i]-2];
        for(h>0?--h:0;S[i+h]==S[pi+h];++h);
        height[R[i]]=h;
    }
    height[1]=0;
}
----------AC-automation----------
struct Node
{
    bool hit;
    Node *child[26], *sfx;
    Node(): hit(0), sfx(0) { memset(child, 0, sizeof(child)); }
    Node* get_child(int i)
    {
        Node* p;
        for(p=this;p!=root && !p->child[i];p=p->sfx);
        return p->child[i]?p->child[i]:root;
```

```
    }
}Npool[maxNode], *Nptr, *root;
void dict_insert(char* str)
{
    Node* p=root;

    for(;*str;++str)
    {
        int i=c2i(*str);
        if(!p->child[i]) p->child[i] = new (Nptr++) Node();
        p=p->child[i];
    }
    p->hit=true;
}
void dict_get_sfx()
{
    static queue<Node*> Q;
    while(!Q.empty())Q.pop(); root->sfx=root;
    for(int i=0;i<26;++i) if(root->child[i])
    {
        root->child[i]->sfx=root;
        Q.push(root->child[i]);
    }
    while(!Q.empty())
    {
        Node* p=Q.front(); Q.pop();
        p->hit|=p->sfx->hit;
        for(int i=0;i<26;++i) if(p->child[i])
        {
            p->child[i]->sfx=p->sfx->get_child(i);
            Q.push(p->child[i]);
        }
    }
}
```

```
----------BIT----------
for(int i=0;i<=5000;i++) lowbit[i]=i&(-i);
int query(int n)
{
    int ret=0;
    while(n>0) {ret+=s[n];    n-=lowbit[n];    }
    return ret;
}
void add(int n)
{
    while(n<=5000) { s[n]++; n+=lowbit[n]; }
    return ;
}
----------inverse ----------
void pre()    // Mod  素数
{
    inv[0]=inv[1]=1;
    for(int i=2;i<Mod;i++)
        inv[i]=((Mod-Mod/i)*inv[Mod%i])%Mod;
}
```

----------Geometry by clj----------

```
#include <climits>
#include <numeric>
#include <cmath>
#include <vector>

#define foreach(e,x) for(__typeof(x.begin()) e=x.begin();e!=x.end();++e)
#define REP(i,n) for(int i=0;i<n;++i)

const double EPS = 1e-8;
inline int sign(double a)
{
    return a < -EPS ? -1 : a > EPS;
}

struct Point
{
    double x, y;
    Point() {};
    Point(double _x, double _y) :x(_x), y(_y) {};
    Point operator+(const Point&p) const {
        return Point(x + p.x, y + p.y);
    }
    Point operator-(const Point&p) const {
        return Point(x - p.x, y - p.y);
    }
    Point operator*(double d) const {
        return Point(x * d, y * d);
    }
    Point operator/(double d) const {
        return Point(x / d, y / d);
    }
    bool operator<(const Point&p) const {
        int c = sign(x - p.x);
        if (c) return c == -1;
        return sign(y - p.y) == -1;
    }
    double dot(const Point&p) const {
        return x * p.x + y * p.y;
    }
    double det(const Point&p) const {
        return x * p.y - y * p.x;
    }
    double alpha() const {
        return atan2(y, x);
    }
    double distTo(const Point&p) const {
        double dx = x - p.x, dy = y - p.y;
        return hypot(dx, dy);
    }
    double alphaTo(const Point&p) const {
        double dx = x - p.x, dy = y - p.y;
        return atan2(dy, dx);
    }
    void read() {
        scanf("%lf%lf", &x, &y);
    }
    double abs() {
        return hypot(x, y);
    }
    double abs2() {
        return x * x + y * y;
```

```
    }
    void write() {
        cout << "(" << x << "," << y << ")" << endl;
    }
};
#define cross(p1,p2,p3) ((p2.x-p1.x)*(p3.y-p1.y)-(p3.x-p1.x)*(p2.y-p1.y))
#define crossOp(p1,p2,p3) sign(cross(p1,p2,p3))

Point isSS(Point p1, Point p2, Point q1, Point q2) {
    double a1 = cross(q1,q2,p1), a2 = -cross(q1,q2,p2);
    return (p1 * a2 + p2 * a1) / (a1 + a2);
}

vector<Point> convexCut(const vector<Point>&ps, Point q1, Point q2) {
    vector<Point> qs;
    int n = ps.size();
    for (int i = 0; i < n; ++i) {
        Point p1 = ps[i], p2 = ps[(i + 1) % n];
        int d1 = crossOp(q1,q2,p1), d2 = crossOp(q1,q2,p2);
        if (d1 >= 0)
        qs.push_back(p1);
        if (d1 * d2 < 0)
        qs.push_back(isSS(p1, p2, q1, q2));
    }
    return qs;
}

double calcArea(const vector<Point>&ps) {
    int n = ps.size();
    double ret = 0;
    for (int i = 0; i < n; ++i) {
        ret += ps[i].det(ps[(i + 1) % n]);
    }
    return ret / 2;
}

vector<Point> convexHull(vector<Point> ps) {
    int n = ps.size();
    if (n <= 1) return ps;
    sort(ps.begin(), ps.end());
    vector<Point> qs;
    for (int i = 0; i < n; qs.push_back(ps[i++])) {
        while (qs.size() > 1 && crossOp(qs[qs.size()-2],qs.back(),ps[i]) <= 0)
        qs.pop_back();
    }
    for (int i = n - 2, t = qs.size(); i >= 0; qs.push_back(ps[i--])) {
        while ((int)qs.size() > t && crossOp(qs[qs.size()-2],qs.back(),ps[i]) <= 0)
        qs.pop_back();
    }
    qs.pop_back();
    return qs;
}

double convexDiameter(const vector<Point>&ps) {
    int n = ps.size();
    int is = 0, js = 0;
    for (int i = 1; i < n; ++i) {
        if (ps[i].x > ps[is].x)
        is = i;
        if (ps[i].x < ps[js].x)
        js = i;
    }
```

```
    double maxd = ps[is].distTo(ps[js]);
    int i = is, j = js;
    do {
    if ((ps[(i + 1) % n] - ps[i]).det(ps[(j + 1) % n] - ps[j]) >= 0)
        (++j) %= n;
    else
        (++i) %= n;
    maxd = max(maxd, ps[i].distTo(ps[j]));
    }while (i != is || j != js);
    return maxd;
};
```

----------HalfPlaneIntersection by clj----------

```
#define foreach(e,x) for(__typeof(x.begin()) e=x.begin();e!=x.end();++e)
struct Point

Point isSS(Point p1, Point p2, Point q1, Point q2)
struct Border {
        Point p1, p2;
        long double alpha;
        void setAlpha() {alpha = atan2(p2.y - p1.y, p2.x - p1.x); }
        void read() {p1.read();p2.read();setAlpha();}
};
int n;
const int MAX_N_BORDER = 20000 + 10;
const long double LARGE = 10000;
Border border[MAX_N_BORDER];

bool operator<(const Border&a, const Border&b) {
        int c = sign(a.alpha - b.alpha);
        if (c != 0)  return c == 1;
        return crossOp(b.p1,b.p2,a.p1) >= 0;
}

bool operator==(const Border&a, const Border&b) {
        return sign(a.alpha - b.alpha) == 0;
}
void add(long double x, long double y, long double nx, long double ny) {
        border[n].p1 = Point(x, y);
        border[n].p2 = Point(nx, ny);
        border[n].setAlpha();
        n++;
}

Point isBorder(const Border&a, const Border&b) {
        return isSS(a.p1, a.p2, b.p1, b.p2);
}
Border que[MAX_N_BORDER];
int qh, qt;
bool check(const Border&a, const Border&b, const Border&me) {
        Point is = isBorder(a, b);
        return crossOp(me.p1,me.p2,is) > 0;
}
void convexIntersection() {
        qh = qt = 0;
        sort(border, border + n);
        n = unique(border, border + n) - border;
        for (int i = 0; i < n; ++i) {
                Border cur = border[i];
                while (qh + 1 < qt && !check(que[qt - 2], que[qt - 1], cur)) --qt;
                while (qh + 1 < qt && !check(que[qh], que[qh + 1], cur)) ++qh;
                que[qt++] = cur;
        }
```

```
	while (qh + 1 < qt && !check(que[qt - 2], que[qt - 1], que[qh])) --qt;
	while (qh + 1 < qt && !check(que[qh], que[qh + 1], que[qt - 1])) ++qh;
}
void calcArea() {
	static Point ps[MAX_N_BORDER];
	int cnt = 0;
	if (qt - qh <= 2) {puts("0.0");return;}
	for (int i = qh; i < qt; ++i) {
		int next = i + 1 == qt ? qh : i + 1;
		ps[cnt++] = isBorder(que[i], que[next]);
	}
	long double area = 0;
	for (int i = 0; i < cnt; ++i) {area += ps[i].det(ps[(i + 1) % cnt]);}
	area /= 2;
	area = fabsl(area);
	cout.setf(ios::fixed);
	cout.precision(1);
	cout << area << endl;
}
int main() {
	cin >> n;
	for (int i = 0; i < n; ++i) {border[i].read();}
	add(0, 0, LARGE, 0);
	add(LARGE, 0, LARGE, LARGE);
	add(LARGE, LARGE, 0, LARGE);
	add(0, LARGE, 0, 0);
	convexIntersection();
	calcArea();
}
```

----------Geo 3D by clj----------

```
#include <climits>
#include <numeric>
#define foreach(e,x) for(__typeof(x.begin()) e=x.begin();e!=x.end();++e)
#define REP(i,n) for(int i=0;i<n;++i)
using namespace std;

const int MAX_N_POINTS = 10000 + 10;
const int MAX_N_FACES = 10000 + 10;
const double EPS = 1e-8;

int sign(double x) {  return x < -EPS ? -1 : x > EPS;}
struct Point {
	double x, y, z;
	void read() {
		scanf("%lf%lf%lf", &x, &y, &z);
	}
	Point() {
	}
	Point(double _x, double _y, double _z) : x(_x), y(_y), z(_z) {}

	Point operator+(Point p) {
		return Point(x + p.x, y + p.y, z + p.z);
	}
	Point operator-(Point p) {
		return Point(x - p.x, y - p.y, z - p.z);
	}
	Point operator*(double f) {
		return Point(x * f, y * f, z * f);
	}
	Point operator/(double f) {
```

```
        return Point(x / f, y / f, z / f);
    }
    double dot(Point p) {
        return x * p.x + y * p.y + z * p.z;
    }
    Point det(Point p) {
        return Point(y * p.z - z * p.y, z * p.x - x * p.z, x * p.y - y * p.x);
    }
    double abs() {
        return sqrt(abs2());
    }
    double abs2() {
        return x * x + y * y + z * z;
    }
    Point norm() {
        return *this / abs();
    }
    void write() {
        cout << x << " " << y << " " << z << endl;
    }
};
//a plane (*-p) dot o ==0
//get a plane form p1,p2,p3
void set(Point p1, Point p2, Point p3, Point&p, Point&o) {
    o = (p2 - p1).det(p3 - p1).norm();
    p = p1;
}

double disFP(Point p, Point o, Point q) { //plane point
    return (q - p).dot(o);
}

double disLP(Point p1, Point p2, Point q) { //line point
    return (p2 - p1).det(q - p1).abs() / (p2 - p1).abs();
}

double disLL(Point p1, Point p2, Point q1, Point q2) {
    Point p = q1 - p1;
    Point u = p2 - p1;
    Point v = q2 - q1;
    double d = u.abs2() * v.abs2() - u.dot(v) * u.dot(v);
    if (abs(d) < EPS) return disLP(q1, q2, p1);
    double s = (p.dot(u) * v.abs2() - p.dot(v) * u.dot(v)) / d;
    return disLP(q1, q2, p1 + u * s);
}

Point isFL(Point p, Point o, Point q1, Point q2) {
    double a = o.dot(q2 - p);
    double b = o.dot(q1 - p);
    double d = a - b;
    if (abs(d) < EPS)
        throw "none";
    return (q1 * a - q2 * b) / d;
}

vector<Point> isFF(Point p1, Point o1, Point p2, Point o2) {
    Point e = o1.det(o2);
    Point v = o1.det(e);
    double d = o2.dot(v);
    if (abs(d) < EPS)
        throw "none";
    Point q = p1 + (v * (o2.dot(p2 - p1)) / d);
```

```
    vector<Point> ret;
    ret.push_back(q);
    ret.push_back(q + e);
    return ret;
}
```

----------FFT by clj----------

```
#include <algorithm>
#include <complex>
#include <vector>
#include <cmath>
#define foreach(e,x) for(__typeof(x.begin()) e=x.begin();e!=x.end();++e)
typedef complex<double> Comp;

const Comp I(0, 1);

const int MAX_N = 1 << 20;
Comp tmp[MAX_N];
void DFT(Comp*a, int n, int rev) {
  if (n == 1) return;
  for (int i = 0; i < n; ++i) tmp[i] = a[i];
  for (int i = 0; i < n; ++i) {
    if (i & 1) a[n / 2 + i / 2] = tmp[i];
    else a[i / 2] = tmp[i];
  }
  Comp*a0 = a, *a1 = a + n / 2;
  DFT(a0, n / 2, rev);
  DFT(a1, n / 2, rev);
  Comp cur(1, 0);
  double alpha = 2 * M_PI / n * rev;
  Comp step = exp(I * alpha);
  for (int k = 0; k < n / 2; ++k) {
    tmp[k] = a0[k] + cur * a1[k];
    tmp[k + n / 2] = a0[k] - cur * a1[k];
    cur *= step;
  }
  for (int i = 0; i < n; ++i) {
    a[i] = tmp[i];
  }
}
int main() {
  static Comp a[1 << 20] = { }, b[1 << 20] = { };
  int n = 1 << 20;
  DFT(a, n, 1);
  DFT(b, n, 1);
  for (int i = 0; i < n; ++i) a[i] *= b[i];
  DFT(a, n, -1);
  for (int i = 0; i < n; ++i) a[i] /= n;
}
```

----------manacher by clj----------

```
//len[i] means the max palindrome length centered i/2
void palindrome(char cs[], int len[], int n) {
  for (int i = 0; i < n * 2; ++i)    len[i] = 0;
  for (int i = 0, j = 0, k; i < n * 2; i += k, j = max(j - k, 0))
  {
    while (i - j >= 0 && i + j + 1 < n * 2 && cs[(i - j) / 2] == cs[(i + j + 1) / 2]) j++;
    len[i] = j;
    for (k = 1; i - k >= 0 && j - k >= 0 && len[i - k] != j - k; k++)
      len[i + k] = min(len[i - k], j - k);
  }
}
```

```
----------SAP by clj----------
#define foreach(e,x) for(__typeof(x.begin()) e=x.begin();e!=x.end();++e)
#define REP(i,n) for(int i=0;i<n;++i)

template<class Flow>
struct Maxflow
{
   static const Flow INF = ~0U >> 1; //should change with type
//    static const Flow INF = numeric_limits<Flow>::max();
   struct Edge
   {
      int t;          Flow c;    Edge*n, *r;
      Edge(int _t, Flow _c, Edge*_n) : t(_t), c(_c), n(_n) {}
   };
   vector<Edge*> E;
   int addV()
   {
      E.push_back((Edge*) 0);
      return E.size() - 1;
   }
   void clear()
   {
      E.clear();
   }
   Edge* makeEdge(int s, int t, Flow c)
   {
      return E[s] = new Edge(t, c, E[s]);
   }
   void addEdge(int s, int t, Flow c)
   {
      Edge*e1 = makeEdge(s, t, c), *e2 = makeEdge(t, s, 0);
      e1->r = e2, e2->r = e1;
   }
   int calcMaxFlow(int vs, int vt)
   {
      int nV = E.size();
      Flow totalFlow = 0;

      vector<Flow> am(nV, 0);
      vector<int> h(nV, 0), cnt(nV + 1, 0);
      vector<Edge*> prev(nV, (Edge*) 0), cur(nV, (Edge*) 0);
      cnt[0] = nV;

      int u = vs;
      Edge*e;
      am[u] = INF;
      while (h[vs] < nV)
      {
         for (e = cur[u]; e; e = e->n)
            if (e->c > 0 && h[u] == h[e->t] + 1)
         break;
      if (e)
            {
         int v = e->t;
         cur[u] = prev[v] = e;
         am[v] = min(am[u], e->c);
         u = v;
         if (u == vt)
               {
            Flow by = am[u];
            while (u != vs)
                  {
```

```
                prev[u]->c -= by;
                prev[u]->r->c += by;
                u = prev[u]->r->t;
            }
            totalFlow += by;
            am[u] = INF;
        }
    }
    else
    {
        if (!--cnt[h[u]])break;
        h[u] = nV;
        for (e = E[u]; e; e = e->n)
            if (e->c > 0 && h[e->t] + 1 < h[u])
            {
                h[u] = h[e->t] + 1;
                cur[u] = e;
            }
            ++cnt[h[u]];
            if (u != vs)
                u = prev[u]->r->t;
        }
    }
    return totalFlow;
}
```

----------LCT by clj----------

```
//Dynamic Tree
typedef long long int64;
const int MOD = 51061;
const int MAX_N = int(1e5) + 10;
struct Mark {
    int64 add, mul; //x*mul+add
    Mark(int64 add, int64 mul) {this->add = add;this->mul = mul;}
    Mark() { mul = 1; add = 0;      }
    bool isId() {return mul == 1 && add == 0;}
};
Mark operator*(Mark a, Mark b) {
    return Mark((a.add * b.mul + b.add) % MOD, a.mul * b.mul % MOD);
}
struct Node {
    Node*p, *ch[2];
    bool rev;
    Mark m;
    int64 sum, val;
    int size;
    bool isRoot;
    Node*fa;
    Node() {sum = 0;isRoot = 0;size = 0;}
    void sc(Node*c, int d) {  ch[d] = c; c->p = this;}
    bool d() { return this == p->ch[1];  }
    void upd() {
        sum = (val + ch[0]->sum + ch[1]->sum) % MOD;
        size = 1 + ch[0]->size + ch[1]->size;
    }
    void apply(Mark a) {
        m = m * a;
        sum = (sum * a.mul + a.add * size) % MOD;
        val = (val * a.mul + a.add) % MOD;      }
    void revIt() {   rev ^= 1;  swap(ch[0], ch[1]);  }
    void relax();
    void setRoot(Node*f);
} Tnull, *null = &Tnull;
```

```
void Node::setRoot(Node*f) {
	fa = f;
	isRoot = true;
	p = null;
}
void Node::relax() {
	if (!m.isId()) {
		REP(i,2)	if (ch[i] != null)ch[i]->apply(m);
		m = Mark();
	}
	if (rev) {
		REP(i,2)	if (ch[i] != null)	ch[i]->revIt();
		rev = 0;
	}
}
Node mem[MAX_N], *C = mem;
Node*make(int v) {
	C->sum = C->val = v;
	C->rev = 0; C->m = Mark();
	C->ch[0] = C->ch[1] = null;
	C->isRoot = true;
	C->p = null;	C->fa = null;
	return C++;
}
void rot(Node*t) {
	Node*p = t->p;
	p->relax();	t->relax();
	bool d = t->d();
	p->p->sc(t, p->d());	p->sc(t->ch[!d], d);
	t->sc(p, !d);
	p->upd();
	if (p->isRoot) {
		p->isRoot = false;	t->isRoot = true;
		t->fa = p->fa;
	}
}
void pushTo(Node*t) {
	static Node*stk[MAX_N];
	int top = 0;
	while (t != null) {stk[top++] = t;t = t->p; }
	for (int i = top - 1; i >= 0; --i)	stk[i]->relax();
}

void splay(Node*u, Node*f = null) {
	pushTo(u);
	while (u->p != f) {
	  if (u->p->p == f)	rot(u);
		else	u->d() == u->p->d() ? (rot(u->p), rot(u)) : (rot(u), rot(u));
	}
	u->upd();
}
Node*v[MAX_N];
vector<int> E[MAX_N];
int n, nQ;
int que[MAX_N], fa[MAX_N], qh = 0, qt = 0;

void bfs() {
	que[qt++] = 0;	fa[0] = -1;
	while (qh < qt) {
		int u = que[qh++];
		for (vector<int>::iterator e = E[u].begin(); e != E[u].end(); ++e)
```

```
                if (*e != fa[u])  fa[*e] = u, v[*e]->fa = v[u], que[qt++] = *e;
        }
}

Node* expose(Node*u) {
        Node*v;
        for (v = null; u != null; v = u, u = u->fa) {
                splay(u);
                u->ch[1]->setRoot(u);
                u->sc(v, 1);
                v->fa = u;
        }
        return v;
}
void makeRoot(Node*u) {
        expose(u);    splay(u);     u->revIt();
}
void addEdge(Node*u, Node*v) {
        makeRoot(v);  v->fa = u;
}
void delEdge(Node*u, Node*v) {
        makeRoot(u);          expose(v);
        splay(u);             u->sc(null, 1);
        u->upd();             v->fa = null;
        v->isRoot = true;     v->p = null;
}
void markPath(Node*u, Node*v, Mark m) {
        makeRoot(u);  expose(v);
        splay(v);          v->apply(m);
}
int queryPath(Node*u, Node*v) {
        makeRoot(u);          expose(v);       splay(v);
        return v->sum;
}
int main() {
        scanf("%d%d", &n, &nQ);
        REP(i,n-1) {
                int u, v;    scanf("%d%d", &u, &v);
                --u, --v;
                E[u].push_back(v);          E[v].push_back(u);
        }
        REP(i,n)           v[i] = make(1);
        bfs();
        REP(i,nQ) {
                char cmd; scanf(" ");
                scanf("%c", &cmd);
                int i, j;scanf("%d%d", &i, &j);
                Node*u = ::v[--i], *v = ::v[--j];
                if (cmd == '+') {
                        int c;scanf("%d", &c);
                        markPath(u, v, Mark(c, 1));
                } else if (cmd == '*') {
                        int c;       scanf("%d", &c);
                        markPath(u, v, Mark(0, c));
                } else if (cmd == '/') {
                printf("%d\n", queryPath(u, v));
                } else {
                        int k, l;     scanf("%d%d", &k, &l);
                        delEdge(u, v);  addEdge(::v[--k], ::v[--l]);
                }
        }
}
```

```
----------DLX by kuangbin----------
const int N = 9; //3*3 数独
const int MaxN = N*N*N + 10;
const int MaxM = N*N*4 + 10;
const int maxnode = MaxN*4 + MaxM + 10;
char g[MaxN];
struct DLX
{
    int n,m,size;
    int U[maxnode],D[maxnode],R[maxnode],L[maxnode],
        Row[maxnode],Col[maxnode];
    int H[MaxN],S[MaxM];
    int ansd,ans[MaxN];
    void init(int _n,int _m)
    {
        n = _n;  m = _m;
        for(int i = 0;i <= m;i++)
        {
            S[i] = 0;
            U[i] = D[i] = i;
            L[i] = i-1;  R[i] = i+1;
        }
        R[m] = 0; L[0] = m;
        size = m;
        for(int i = 1;i <= n;i++)H[i] = -1;
    }
    void Link(int r,int c)
    {
        ++S[Col[++size]=c];
        Row[size] = r;
        D[size] = D[c];      U[D[c]] = size;
        U[size] = c;         D[c] = size;
        if(H[r] < 0)H[r] = L[size] = R[size] = size;
        else
        {
            R[size] = R[H[r]];   L[R[H[r]]] = size;
            L[size] = H[r];      R[H[r]] = size;
        }
    }
    void remove(int c)
    {
        L[R[c]] = L[c];   R[L[c]] = R[c];
        for(int i = D[c];i != c;i = D[i])
            for(int j = R[i];j != i;j = R[j])
            {
                U[D[j]] = U[j]; D[U[j]] = D[j];
                --S[Col[j]];
            }
    }
    void resume(int c)
    {
        for(int i = U[c];i != c;i = U[i])
            for(int j = L[i];j != i;j = L[j])
                ++S[Col[U[D[j]]=D[U[j]]=j]];
        L[R[c]] = R[L[c]] = c;
    }
    bool Dance(int d)
    {
        if(R[0] == 0)
        {
            for(int i = 0;i < d;i++)g[(ans[i]-1)/9] = (ans[i]-1)%9 + '1';
            for(int i = 0;i < N*N;i++)printf("%c",g[i]);
```

```
            printf("\n");
            return true;
        }
        int c = R[0];
        for(int i = R[0];i != 0;i = R[i])
            if(S[i] < S[c])
                c = i;
        remove(c);
        for(int i = D[c];i != c;i = D[i])
        {
            ans[d] = Row[i];
            for(int j = R[i];j != i;j = R[j])remove(Col[j]);
            if(Dance(d+1))return true;
            for(int j = L[i];j != i;j = L[j])resume(Col[j]);
        }
        resume(c);
        return false;
    }
};
void place(int &r,int &c1,int &c2,int &c3,int &c4,int i,int j,int k)
{
    r = (i*N+j)*N + k; c1 = i*N+j+1; c2 = N*N+i*N+k;
    c3 = N*N*2+j*N+k; c4 = N*N*3+((i/3)*3+(j/3))*N+k;
}
DLX dlx;
int main() {
    while(scanf("%s",g) == 1) {
        if(strcmp(g,"end") == 0)break;
        dlx.init(N*N*N,N*N*4);
        int r,c1,c2,c3,c4;
        for(int i = 0;i < N;i++)
            for(int j = 0;j < N;j++)
                for(int k = 1;k <= N;k++)
                    if(g[i*N+j] == '.' || g[i*N+j] == '0'+k) {
                        place(r,c1,c2,c3,c4,i,j,k);
                        dlx.Link(r,c1);  dlx.Link(r,c2);
                        dlx.Link(r,c3);  dlx.Link(r,c4);
                    }
        dlx.Dance(0);
    }
    return 0;
}
```

----------Java----------

```
import java.io.BufferedInputStream;
import java.util.Scanner;
import java.math.*;

public class Main
{
	public static void main(String[] args)
	{
		Scanner cin = new Scanner(new BufferedInputStream(System.in));
		int T = cin.nextInt();
		BigInteger a, b;
		for (int count = 1; count <= T; count++) {
			a = cin.nextBigInteger();
			b = cin.nextBigInteger();
			System.out.println("Case "+count+":");
			System.out.print(a+" + "+b+" = ");
			System.out.println(a.add(b));
			if (count < T) System.out.println();
		}
		cin.close();
	}
}
```

----------Vimrc----------

```
e $VIM\.vimrc
set nu
set cin sw=2
map <F9> :w<return> :!g++ -Wall % -o %< <return>:!./%< <return>
map <M-a> i//<ESC>h
map <M-d> xx
map <M-j> i<space><space><ESC>
```

----------BigInt----------

```
const int BI_n=100;
struct BigInt
{
    private:
        int a[BI_n];
        mutable int n;
        inline void trunc()const{for(--n;n>=0 && !a[n];--n);++n;}
        void carry(int start=0) {
            int i, tmp=0;
            for(i=start;i<n || tmp;++i)    {
                tmp+=a[i]; a[i]=tmp%10000; tmp/=10000;
            }
            if(i>=n) n=i;
        }
    public:
        BigInt(int x=0){
            memset(a, 0, sizeof(a));
            for(n=0;x;++n) a[n]=x%10000, x/=10000;
        }
        BigInt(const string& x) {
            memset(a, 0, sizeof(a)); n=0;
            int block=0, p10=1;
            for(int l=(int)x.size(), i=l-1;i>=0;--i) {
                block+=(x[i]-'0')*p10;    p10*=10;
                if(p10==10000) {
                    a[n++]=block;
                    block=0; p10=1;
                }
            }
            if(block) a[n++]=block;
```

```
    }
    void output() {
        printf("%d",a[n-1]);
        for(int i=n-2;i>=0;--i) printf("%04d",a[i]);
        printf("\n"); return ;
    }
    friend ostream& operator<<(ostream& out, BigInt x) {
        if(x.n==0) return out<<"0";
        out<<x.a[x.n-1];
        for(int i=x.n-2;i>=0;--i) out.fill('0'), out.width(4), out<<x.a[i];
        return out;
    }
    friend BigInt operator+(const BigInt& x, const BigInt& y) {
        BigInt ret;
        ret.n=max(x.n, y.n)+1;
        int tmp=0;
        for(int i=0;i<ret.n;++i)    {
            if(i<x.n) tmp+=x.a[i];
            if(i<y.n) tmp+=y.a[i];
            ret.a[i]=tmp%10000;    tmp/=10000;
        }
        ret.trunc();
        return ret;
    }
    friend BigInt operator-(const BigInt& x, const BigInt& y) {
        BigInt ret;    ret.n=x.n;
        int tmp=0, borrow=0;
        for(int i=0;i<x.n;++i) {
            tmp=x.a[i]-y.a[i]-borrow;
            if(tmp<0) ret.a[i]=tmp+10000, borrow=1;
            else          ret.a[i]=tmp,    borrow=0;
        }
        ret.trunc();
        return ret;
    }
    friend BigInt operator*(const BigInt& x, const BigInt& y) {
        BigInt ret;
        ret.n=x.n+y.n;
        for(int i=0;i<x.n;++i)
            for(int j=0;j<y.n;++j)    {
                ret.a[i+j]+=x.a[i]*y.a[j];
                if(ret.a[i+j]>100000000) ret.carry(i+j);
            }
        ret.carry(0);
        ret.trunc();
        return ret;
    }
};
```

----------**Segment-Tree**----------
-
```
struct Node
{
   int st,ed;
   long long col;
   int label;
   Node *left,*right;
   Node(){};
   Node(int st,int ed):st(st),ed(ed),label(0),left(NULL),right(NULL){};
   void update()
   {
      col=0ll;
      if(left)    col=left->col;
      if(right) col|=right->col;
      return ;
   }
   void downlabel()
   {
      if(left)    left->label=label, left->col=col;
      if(right) right->label=label, right->col=col;
      label=0;
   }
}Npool[maxNode],*Nptr,*root;
void Build(int st,int ed,Node *&p=root)
{
   p=new (Nptr++) Node(st,ed);
   p->col=2;
   if(ed-st==1)return ;
   int mid=(st+ed)>>1;
   Build(st,mid,p->left),Build(mid,ed,p->right);
   return ;
}
void Modify(int st,int ed,int c,Node *&p=root)
{
   if(p->label)p->downlabel();
   if(st==p->st && ed==p->ed)
   {
      p->label=c;
      p->col=(1ll)<<(c-1);
      return;
   }
   int mid=(p->st+p->ed)>>1;
   if(mid<=st) Modify(st,ed,c,p->right);
   else if(mid>=ed) Modify(st,ed,c,p->left);
   else Modify(st,mid,c,p->left),Modify(mid,ed,c,p->right);
   p->update();
}

long long Query(int st,int ed,Node *&p=root)
{
   if(p->label) p->downlabel();
   if(p->st==st && p->ed==ed) return p->col;
   int mid=(p->st+p->ed)>>1;
   if(mid<=st) return Query(st,ed,p->right);
   else if(mid>=ed) return Query(st,ed,p->left);
   else return Query(st,mid,p->left)|Query(mid,ed,p->right);
}
```

----------Hash----------

```
unsigned int BKDRHash(char *str)
{
    unsigned int seed = 131; // 31 131 1313 13131 131313 etc..
    unsigned int hash = 0;
    while (*str) hash = hash * seed + (*str++);
    return (hash & 0x7FFFFFFF);
}
unsigned int APHash(char *str)
{
    unsigned int hash = 0;
    for (int I;i=0; *str; i++)
    {
        if ((i & 1) == 0) hash ^= ((hash << 7) ^ (*str++) ^ (hash >> 3));
        else hash ^= (~((hash << 11) ^ (*str++) ^ (hash >> 5)));
    }
    return (hash & 0x7FFFFFFF);
}
unsigned int DJBHash(char *str)
{
    unsigned int hash = 5381;
    while (*str) hash += (hash << 5) + (*str++);
    return (hash & 0x7FFFFFFF);
}
unsigned int JSHash(char *str)
{
    unsigned int hash = 1315423911;
    while (*str)    hash ^= ((hash << 5) + (*str++) + (hash >> 2));
    return (hash & 0x7FFFFFFF);
}
```

----------locale----------

isspace	Check if character is a white-space
isprint	Check if character is printable
iscntrl	Check if character is a control character
isupper	Check if character is uppercase letter
islower	Check if character is lowercase letter
isalpha	Check if character is alphabetic
isdigit	Check if character is decimal digit
ispunct	Check if character is a punctuation character
isxdigit	Check if character is hexadecimal digit
isalnum	Check if character is alphanumeric
isgraph	Check if character has graphical representation
isblank	Check if character is blank

----------math----------

cbrt(x)	Returns the *cubic root* of *x*.
hypot(x,y)	Compute hypotenuse—(x^2+y^2)
ceil	Round up value
floor	Round down value
round	Round to nearest
lround	Round to nearest and cast to long integer
llround	Round to nearest and cast to long long integer
rint	Round to integral value
lrint	Round and cast to long integer
llrint	Round and cast to long long integer
nearbyint	Round to nearby integral value

trunc Rounds *x* toward zero, returning the nearest integral value that is not larger in magnitude than *x*.