Team notebook

November 14, 2019

Contents

1 Bits and Bytes

1.1 all subsets(3^n)

```
/// iterating over all subsets : O(3^n)
for (int mask = 0; mask < (1<<n); mask++){
       F[mask] = A[0];
    /// iterate over all the subsets of the mask
    for(int i = mask; i > 0; i = (i-1) & mask){
       F[mask] += A[i];
    }
}
```

1.2 bit manipulation

```
#include <bits/stdc++.h>
using namespace std;

bitset <100> bs ;

int main () {
    /// lowest set bit of a 32-bit integer : example :
       __builtin_ctz(100100) = 2 . exception : builtin_ctz(0) is
       undefined
    ///long long x = 8589934591 ;
    ///cout << __builtin_ctz(x) << endl ;
    /// lowest set bit of a 64-bit integer : example :
       __builtin_ctzll(100100) = 2 . exception :
       builtin_ctzll(0) is undefined
    ///__builtin_ctzll
    /// number of set bits : __builtin_popcount(x)
    /// number of set bits : __builtin_popcountll(x)

    bs[15] = 1 ;
    bs[43] = 1 ;
    bs[12] = 1 ;
    cout << bs._Find_first() << endl ; /// first set bit :
       prints 12 /// returns bs.size() if no set bit
    cout << bs._Find_next(15) << endl ; /// set bit strictly
       next to 15 . prints 43 /// if no set bit after idx = (15)
       returns bs.size()
    cout << bs.count() << endl ;
    //bs.set() ;
    //bs.reset() ;
    bs.flip() ;
    cout << bs.count() << endl ;
}
```

1.3 bitset kundu

```
typedef unsigned int UI;

struct Bitset {
    const static int K = 32, X = 31;

    vector<UI> bs;
    int N;

    Bitset(int n) {
        N = n/K+1;
        bs.resize(N);
    }

    bool get(int i) {
        return bs[i>>5] & (1U<<(i&X));
    }

    void set(int i) {
        bs[i>>5] |= (1U<<(i&X));
    }

    void And(const Bitset &b) {
        for (int i=0; i<N; i++)
            bs[i] &= b.bs[i];
    }

    int count() {
        int ans = 0;
        for (int i=0; i<N; i++)
            ans += __builtin_popcount(bs[i]);
        return ans;
    }
};
```

1.4 substring check with Bitset

```
/// How many times a string occur in another one using Bitset
#include <bits/stdc++.h>
using namespace std;
const int N = 100005;
char s[N], t[N];
bitset <N> bs[26], cur;

int main () {
    //freopen ("in.txt", "r" , stdin) ;
    scanf ("%s" , s+1);
    int n = strlen(s+1);
    for (int i = 1; i <= n; i++) bs[s[i]-'a'][i] = 1;
    int q;
    cin >> q;
    while (q--) {
        int type;
        scanf ("%d", &type);
        if (type == 1) {
            int idx;
            scanf ("%d", &idx);
            scanf("%s",t);
            bs[s[idx]-'a'][idx] = 0;
            s[idx] = t[0];
            bs[s[idx]-'a'][idx] = 1;
        }
        else {
            int l, r;
            scanf ("%d %d %s" , &l, &r, t+1);
            int m = strlen(t+1);
            if (r-l+1 < m) {
                printf("0\n");
                continue;
            }
```

```
            cur.reset();
            cur = ~cur;
            for (int i = 1 ; i <= m ; i++) {
                cur &= (bs[t[i]-'a']>>(i-1)) ;
            }
            cur >>= l;
            int ans = cur.count();
            cur >>= (r-l-m+2);
            ans -= cur.count();
            printf ("%d\n",ans);
        }
    }
}
```

2 Geometry

2.1 2DGeo

```
struct PT {
  ld x,y;
  PT() {}
  PT(ld x,ld y) : x(x), y(y) {}
  PT(const PT &p) : x(p.x), y(p.y) {}
};
PT operator +(PT a,PT b) {
  return PT(a.x+b.x,a.y+b.y);
}
PT operator -(PT a,PT b) {
  return PT(a.x-b.x,a.y-b.y);
}
PT operator *(PT a,ld b) {
  return PT(a.x*b,a.y*b);
}
PT operator /(PT a,ld b){
  return PT(a.x/b,a.y/b);
}
ld operator *(PT a,PT b){ //dot
    return a.x*b.x+a.y*b.y;
}
ld operator ^(PT a,PT b){ //cross
    return a.x*b.y-a.y*b.x;
}

struct Line {
  PT p, v;ld ang;Line() {}
  ld a,b,c; // ax+by+c=0
  Line(PT p,PT v):p(p),v(v){
    ang=atan2(v.y,v.x);
    PT q = p+v;
    if( dcmp(q.x-p.x) == 0 ) {
      a = 1; b = 0; c = -p.x;
    }
    else{
      ld m = (q.y-p.y)/(q.x-p.x);
      a = m; b = -1, c = p.y - m*p.x;
    }
  }
  Line(ld a_,ld b_,ld c_){
    a = a_,b = b_,c = c_;
    v = Point(-b,a) ;
    if (dcmp(a) == 0) p = PT(0,-c/b);
    else p = PT(-c/a,0);
  }
  double val(PT q) { return a*q.x + b*q.y + c} ;
  bool operator < (const Line & L) const {return ang<L.ang;}
  PT point(ld t) { return p+v*t;}
};
int dcmp(ld x) {
```

```
    if (fabs(x)<eps) return 0; return x<0 ? -1 : 1;
}

/// determine if lines from a to b and c to d are parallel or
    collinear
bool LinesParallel(PT a, PT b, PT c, PT d) {
  return fabs((b-a)^(c-d)) < eps;
}
bool LinesCollinear(PT a, PT b, PT c, PT d) {
  return LinesParallel(a, b, c, d) &&
  fabs((a-b)^(a-c)) < eps && fabs((c-d)^(c-a)) < eps;
}
/// intersection of line ab and cd
PT ComputeLineIntersection(PT a, PT b, PT c, PT d) {
  b=b-a;d=c-d;c=c-a;
  assert((b*b) > EPS && (d*d) > EPS);
  return a + b*(c^d)/(b^d);
}
/// projection of point p on line AB
PT GetLineProjection(PT p,PT A,PT B) {
  PT v=B-A;
  return A+v*(v*(p-A))/(v*v);
}
///distance from point p to line AB
ld DistanceToLine(PT p,PT A,PT B) {
  PT v1 = B-A, v2 = p-A;
  return fabs(v1^v2)/len(v1);
}
///checks whether segment AB and segment CD intersects
bool SegmentsIntersect(PT a, PT b, PT c, PT d) {
  if (LinesCollinear(a, b, c, d)) {
    if (dist2(a, c) < eps || dist2(a, d) < eps ||
        dist2(b, c) < eps || dist2(b, d) < eps) return true;
    if ((c-a)*(c-b) > 0 && (d-a)*(d-b) > 0 && (c-b)*(d-b) > 0)
      return false;
    return true;
  }
  if (((d-a)^(b-a))*((c-a)^(b-a)) > 0) return false;
  if (((a-c)^(d-c))*((b-c)^(d-c)) > 0) return false;
  return true;
}
/// project point c onto line segment AB
PT ProjectPointSegment(PT a, PT b, PT c) {

  ld r = (b-a)*(b-a);
  if (fabs(r) < EPS) return a;
  r = (c-a)*(b-a)/r;
  if (r < 0) return a;
  if (r > 1) return b;
  return a + (b-a)*r;
}
///determine if p is on segment ab
//bool OnSegment(PT p,PT a,PT b) {
//  return dcmp((a-p)^(b-p)) == 0 && dcmp((a-p)*(b-p)) < 0;
//}
///distance from point p to segment AB
ld DistanceToSegment(PT p,PT A,PT B) {
  if (A==B) return len(p-A);
  PT v1 = B-A, v2 = p-A, v3 = p - B;
  if (dcmp(v1*v2)<0) return len(v2);
  else if (dcmp(v1*v3)>0 ) return len(v3);
  else return fabs(v1^v2) / len(v1);
}
double segment_segment_distance(PT a, PT b, PT p, PT q) {
    double d1 = dist(a, ProjectPointSegment(p, q, a)), d2 =
        dist(b, ProjectPointSegment(p, q, b)), d3 = dist(p,
        ProjectPointSegment(a, b, p)), d4 = dist(q,
        ProjectPointSegment(a ,b, q));
    return min(min(d1,d2),min(d3,d4));
}
```

```
/// Circle structure
struct CR {
  PT c; ld r,x,y;
  CR(PT c,ld r):c(c),r(r),x(c.x),y(c.y) {}
  PT point(ld rad) {
    return PT(c.x+cos(rad)*r,c.y+sin(rad)*r);
  }
};
int getLineCircleIntersection(Line L, CR cir, vector<PT> &sol) {
  if ( dcmp(DistanceToLine(cir.c,L.p,L.p+L.v)-cir.r)==0) {
    PT A=GetLineProjection(cir.c,L.p,L.p+L.v);
    sol.push_back(A);
    return 1;
  }
  ld a = L.v.x, b = L.p.x - cir.c.x, c = L.v.y, d= L.p.y -
     cir.c.y;
  ld e = a*a+c*c, f = 2*(a*b + c*d), g = b*b+d*d-cir.r*cir.r;
  ld delta = f*f - 4*e*g,t1,t2;
  if (dcmp(delta)<0) return 0;
  else if (dcmp(delta)==0) {
    t1 = t2 = -f / (2*e);
    sol.push_back(L.point(t1));
    return 1;
  }
  t1 = (-f - sqrt(delta)) / (2*e);
  sol.push_back(L.point(t1));
  t2 = (-f + sqrt(delta)) / (2*e);
  sol.push_back(L.point(t2));
  return 2;
}
ld angle(PT v) {
  return atan2(v.y,v.x);
}
int getCircleCircleIntersection(CR C1,CR C2, vector<PT>& sol) {
  ld d = len(C1.c-C2.c);
  if (dcmp(d)==0){
    if (dcmp(C1.r - C2.r)==0) return -1; //same circle
    return 0; //concentric circle
  }
  if (dcmp(C1.r+C2.r-d)<0) return 0; //no intersection, outside
  if (dcmp(fabs(C1.r-C2.r)-d)>0) return 0; //no intersection,
     inside
  ld a = angle(C2.c-C1.c);
  ld da = acos((C1.r*C1.r+d*d - C2.r*C2.r)/ (2*C1.r*d));
  PT p1 = C1.point(a-da), p2 = C1.point(a+da);
  sol.push_back(p1);if (p1==p2) return 1;
  sol.push_back(p2);
  return 2;
}
//tangent from p to circle c,returns dir vec from p to c
int getTangents(PT p,CR c, vector<PT> &sol){
  PT u= c.c-p;
  ld dist = len(u);
  if (dist<c.r) return 0;
  else if (dcmp(dist-c.r)==0){
    sol.push_back(RotateCCW(u,PI/2));
    return 1;
  }
  else{
    ld ang = asin(c.r / dist);
    sol.push_back(RotateCCW(u,-ang));
    sol.push_back(RotateCCW(u,ang));
    return 2;
  }
}
//tangent from p to circle c
//returns points on circle that touches the tangent
int getTangentsPoint(PT p,CR c, vector<PT> &point){
  PT u= c.c-p;ld dist = len(u);
```

```
  if (dist<c.r) return 0;
  else if (dcmp(dist-c.r)==0) {
    point.push_back(p);return 1;
  }
  else {
    PT v;ld ang = asin(c.r / dist);v = RotateCCW(u,-ang);
    point.push_back(GetLineProjection(c.c,p,p+v));
    v = RotateCCW(u, ang);
    point.push_back(GetLineProjection(c.c,p,p+v));return 2;
  }
}
//common tangent of two circle A and B; return the point on
//circles the tangent touchesai-bi is a common tangent
int getTangents(CR A,CR B, vector<PT> &a, vector<PT> &b) {
  int cnt = 0;
  if (A.r<B.r) {
    swap(A,B),swap(a,b);
  }
  ld 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);
  ld rdiff = A.r-B.r; ld rsum = A.r+B.r;
  if (d2 < rdiff*rdiff) return 0;
  ld base = atan2(B.y-A.y,B.x-A.x);
  if (d2 == 0 && A.r == B.r) return -1;
  if (dcmp(d2-rdiff*rdiff)==0) {
    a.push_back(A.point(base)); b.push_back(B.point(base));
    return 1;
  }
  ld ang = acos((A.r-B.r)/sqrt(d2));
  a.push_back(A.point(base+ang));
  b.push_back(B.point(base+ang));
  a.push_back(A.point(base-ang));
  b.push_back(B.point(base-ang));
  if (dcmp(d2-rsum*rsum)) {
    a.push_back(A.point(base));
    b.push_back(B.point(base+PI));
  }
  else if (dcmp(d2-rsum*rsum)==1) {
    ld ang = acos((A.r+B.r)/sqrt(d2));
    a.push_back(A.point(base+ang));
    b.push_back(B.point(PI+base+ang));
    a.push_back(A.point(base-ang));
    b.push_back(B.point(PI+base-ang));
  }
  return (int)a.size();
}
/// pori_britto
CR CircumscribedCircle(PT p1,PT p2,PT p3){
  ld Bx = p2.x-p1.x, By= p2.y-p1.y;
  ld Cx = p3.x-p1.x, Cy= p3.y-p1.y,D = 2*(Bx*Cy-By*Cx);
  ld cx = (Cy*(Bx*Bx+By*By)-By*(Cx*Cx+Cy*Cy))/D + p1.x;
  ld cy = (Bx*(Cx*Cx+Cy*Cy)-Cx*(Bx*Bx+By*By))/D + p1.y;
  PT p = PT(cx,cy); return CR(p,len(p1-p));
}
/// ontor_britto
CR InscribedCircle(PT p1,PT p2,PT p3) {
  ld a = len(p2-p3),b = len(p3-p1),c = len(p1-p2);
  PT p = (p1*a+p2*b+p3*c)/(a+b+c);
  return CR(p,DistanceToLine(p,p1,p2));
}
ld radToPositive(ld rad){
  if (dcmp(rad)<0) rad=ceil(-rad/PI)*PI+rad;
  if (dcmp(rad-PI)>=0) rad-=floor(rad/PI)*PI;
  return rad;
}
PT normalUnit(PT A){
  ld L = len(A);return PT(-A.y/L, A.x/L);
}
Line LineTranslation(Line l, PT v){
  l.p = l.p+v;return l;
}
```

```
/// sol contains the center of these circles
void CircleThroughAPointAndTangentToALineWithRadius(PT p,Line
    l,ld r,vector<PT>& sol) {
  PT e = normalUnit(l.v);
  Line l1=LineTranslation(l,e*r),l2=LineTranslation(l,-e*r);
  getLineCircleIntersection(l1,CR(p,r), sol);
  getLineCircleIntersection(l2,CR(p,r), sol);
}
/// sol contains the center of these circles
void CircleTangentToTwoLinesWithRadius(Line l1,Line l2, ld r,
    vector<PT>& sol) {
  PT e1 = normalUnit(l1.v), e2 = normalUnit(l2.v);
  Line
      L1[2]={LineTranslation(l1,e1*r),LineTranslation(l1,e1*(-r))},
  L2[2]={LineTranslation(l2,e2*r),LineTranslation(l2,-e2*r)};
  for( int i = 0; i < 2; i++ ) {
    for( int j = 0; j < 2; j++ ) {
      sol.push_back(ComputeLineIntersection(L1[i].p,L1[i].v,L2[j].p,L2[j].v));
    }
  }
}
/// sol contains the center of these circles
void CircleTangentToTwoDisjointCirclesWithRadius(CR c1,CR c2,
    ld r, vector<PT>& sol) {
  c1.r+=r;c2.r+=r;getCircleCircleIntersection(c1,c2,sol);
}

int isPointInPolygon(PT p, vector<PT> &poly)
{
    int wn=0;
    int n=poly.size();
    for(int i = 0; i < n; i++)
    {
        if (OnSegment(p,poly[i],poly[(i+1)%n])) return -1; //on
            edge
        int k=dcmp((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; //inside
    return 0; //outside
}
/// Point in simple polygon,1:on/inside;0:strictly out
bool isPointOnPolygon(PT q,vector<PT> &p) {
  int n = p.size() , fl = 0;
  for(int i = 0 ; i < n ; i++) {
    if (fabs((q-p[i])^(p[i+1]-p[i])) < eps and
        (p[i]-q)*(p[i+1]-q) < eps ) {
      return true;
    }
  }
  for(int i = 0 ; i < n ; i++) {
    PT a = p[i] , b = p[i+1] ;
    if (fabs(a.x-b.x) < eps) continue ;
    if (a.x > b.x) swap(a,b);
    if (q.x < a.x-eps or q.x > b.x-eps) continue ;
    if (( (q-a)^(b-a) ) > 0.0) fl ^= 1 ;
  }
  return fl;
}
/// returns 1 if CCW or collinear , returns 0 if CW
bool CCW(PT a, PT b, PT c) {
  ld area=a.x*b.y+b.x*c.y+c.x*a.y-b.x*a.y-c.x*b.y-a.x*c.y;
  return (dcmp(area) >= 0);
}
/// returns 1 if p is on or inside triangle(a,b,c)
bool PointsInTriangle (PT a, PT b, PT c, PT p) {
  int d1 = dcmp((b-a)^(p-b)), d2 = dcmp((c-b)^(p-c)),
```

```
    d3 = dcmp((a-c)^(p-a));
  return !(((d1 < 0) || (d2 < 0) || (d3 < 0)) &&
       ((d1 > 0) || (d2 > 0) || (d3 > 0)));
}
/// cut a convex polygon by a line
vector<PT> cut(vector<PT> &polygon, Line l, int sign) {
    vector<PT> np ;
    int sz = polygon.size();
    for(int i = 0 ; i < sz ; i++) {
        PT p = polygon[i] , q = polygon[(i+1)%sz];
        if (dcmp(l.val(p))*sign >= 0) {
            np.push_back(p);
        }
        ld na = l.val(p), nb = l.val(q);
        if (na*nb < 0.0) {
          np.push_back(a + (b-a)*(na/(na-nb))) ;
        }
    }
    return np ;
}
///diameter of a convex polygon p
ld rotating_calipers(vector<PT> p)
{
  int q = 1, n = p.size();ld ans = 0;
  for( int i = 0; i < n; i++) {
    while(triArea2(p[i],p[(i+1)%n],p[(q+1)%n]) >
        triArea2(p[i],p[(i+1)%n],p[q] ))
      q=(q+1)%n;
    ans = max( ans, (ld)max(len(p[i]- p[q]), len(p[(i+1)%n]
        -p[q] ) ) );
  }
  return ans;
}
///minimum area rectangle for convex polygon
ld rec_rotating_calipers(PT *p,int n)
{
  int q=1;ld ans1=1e15,ans2=1e15;int l=0,r=0;
  for( int i = 0; i < n; i++ ) {
    while(dcmp(triArea2(p[i],p[(i+1)%n],p[(q+1)%n])-triArea2(p[i],p[(i+
        )) >0)
      q=(q+1)%n;
    while (dcmp((p[(i+1)%n]-p[i])*(p[(r+1)%n]-p[r]))>0 )
      r=(r+1)%n;
    if (!i) l = q;
    while (dcmp((p[(i+1)%n]-p[i])*(p[(l+1)%n]-p[l]))<0 )
      l=(l+1)%n;
    ld d = len(p[(i+1)%n]-p[i]);
    ld h = triArea2(p[i],p[(i+1)%n],p[q] )/d;
    ld w
        =(((p[(i+1)%n]-p[i])*(p[r]-p[i]))-((p[(i+1)%n]-p[i])*(p[l]-p[i]
    ans1 = min(ans1, 2*(h+w)), ans2 = min(ans2,h*w);
  }
}

/*tangent lines to a convex polygon from a point outside*/
#define CW     -1
#define ACW     1
int direction(pii st, pii ed, pii q) {
  LL xp = (LL) (ed.xx - st.xx) * (q.yy - ed.yy) - (LL) (ed.yy -
      st.yy) * (q.xx - ed.xx);
  if(!xp) return 0; if(xp > 0) return ACW;
  return CW;
}
bool isGood(pii u, pii v, pii Q, int dir) {
  return (direction(Q, u, v) != -dir);
}
pii better(pii u, pii v, pii Q, int dir) {
  if(direction(Q, u, v) == dir) return u;
  return v;
}
```

```
pii tangents(vector<pii> &hull, pii Q, int dir, int lo, int hi) {
  int mid;
  while(hi - lo + 1 > 2) {
    mid = (lo + hi)/2;
    bool pvs = isGood(hull[mid], hull[mid - 1], Q, dir);
    bool nxt = isGood(hull[mid], hull[mid + 1], Q, dir);
    if(pvs && nxt) return hull[mid];
    if(!(pvs || nxt)) {
      pii p1 = tangents(hull, Q, dir, mid+1, hi);
      pii p2 = tangents(hull, Q, dir, lo, mid - 1);
      return better(p1, p2, Q, dir);
    }
    if(!pvs) {
      if(direction(Q, hull[mid], hull[lo]) == dir) hi = mid - 1;
      else if(better(hull[lo], hull[hi], Q, dir) == hull[lo]) hi
          = mid - 1;
      else lo = mid + 1;
    }
    if(!nxt) {
      if(direction(Q, hull[mid], hull[lo]) == dir) lo = mid + 1;
      else if(better(hull[lo], hull[hi], Q, dir) == hull[lo]) hi
          = mid - 1;
      else lo = mid + 1;
    }
  }
  pii ret = hull[lo];
  for(int i = lo + 1; i <= hi; i++) ret = better(ret, hull[i],
      Q, dir);
  return ret;
}
/// returns two point of convex polygon that is tangent
pair< pii, pii> get_tangents(vector<pii> &polygon, pii Q) {
  pii acw_tan = tangents(polygon, Q, ACW, 0, (int)
      polygon.size() - 1);
  pii cw_tan = tangents(polygon, Q, CW, 0, (int) polygon.size()
      - 1);
  return make_pair(acw_tan, cw_tan);
}
```

2.2 3dGeo

```
#include<bits/stdc++.h>
using namespace std;
const double eps=1e-10;

struct PT {
  double x, y, z;
  PT() {}
  PT(double x, double y, double z) : x(x), y(y), z(z) {}
  PT(const PT &p) : x(p.x), y(p.y), z(p.z) {}
};
PT operator +(PT a,PT b){
  return PT(a.x+b.x,a.y+b.y, a.z+b.z);
}
PT operator -(PT a,PT b){
  return PT(a.x-b.x,a.y-b.y, a.z-b.z);
}
PT operator *(PT a,double b){
  return PT(a.x*b,a.y*b, a.z*b);
}
PT operator /(PT a,double b){
  return PT(a.x/b,a.y/b, a.z/b);
}
double operator *(PT a,PT b) {
  return a.x*b.x+a.y*b.y+a.z*b.z;
}
PT operator ^(PT a,PT b){
  return PT(a.y*b.z-a.z*b.y,a.z*b.x-a.x*b.z,a.x*b.y-a.y*b.x);
```

```
}
int dcmp(double x){
  if (abs(x)<eps) return 0;
  return x<0 ? -1 : 1;
}
bool operator <(const PT &a,const PT &b){
  return make_pair(make_pair(a.x,a.y), a.z) <
         make_pair(make_pair(b.x,b.y), b.z);
}
bool operator==(const PT &a,const PT &b){
return dcmp(a.x-b.x)==0&&dcmp(a.y-b.y)==0&&dcmp(a.z-b.z)==0;
}
double len(PT a){
  return sqrt(a*a);
}
double dist(PT a, PT b){
  return sqrt((a-b)*(a-b));
}
double dist2(PT a, PT b){
  return ((a-b)*(a-b));
}
PT reversePT(PT a){
  return a*(-1);
}
///Angle between two vector
double angleRad( PT a, PT b ){
  return acos( max(-1.0, min(1.0, (a*b)/(len(a)*len(b)))) );
}
///small angle between two vector
double smallAngle( PT a, PT b ){
   return acos( min(abs(a*b)/len(a)/len(b), 1.0) );
}
///u + dt
struct Line{
  PT d, u;
  Line(PT d,PT u):d(d),u(u){}
  PT point(double t){
    return u + d*t;
  }
};
///ax + by + cz = d
struct Plane{
    double a,b,c,d;PT n;
    Plane(){}
    Plane(PT p, PT q, PT r ) :
      Plane( (q-p)^(r-p), ((q-p)^(r-p))*(p) ) {}
    ///normal in direction of p,q,r
    Plane(double a, double b, double c, double d) :
      a(a), b(b), c(c), d(d), n(PT(a,b,c)){}
    Plane(PT n, double d) :
      n(n), a(n.x), b(n.y), c(n.z), d(d) {}
    Plane(const Plane &p) :
      n(p.n), d(p.d), a(p.a), b(p.b), c(p.c) {}
};
///returns 0 if t is on plane p
///returns 1/-1 if t is on positive/negative side of normal
int side( Plane &p, PT a ){
  return dcmp(p.a*a.x + p.b*a.y + p.c*a.z - p.d);
}



///translate all point on a plane with respect to t
Plane Translate( Plane &p, PT t ){
  return Plane( p.n, p.d + p.n*t );
}
///rotate d to the left with respect to normal in plane p
PT rotateCCW90(Plane p, PT d){
  return (p.n^d);
}
```

```
PT unitVector( PT v ){
  return v/len(v);
}
///rotate d to the right with respect to normal in plane p
PT rotateCW90(Plane p, PT d){
  return (d^p.n);
}
///shift plane up(dist>0)/down(dist<0) to distance dist
Plane ShiftUpDown( Plane &p, double dist ){
  return Plane( p.n, p.d + dist*len(p.n) );
}
///returns 0 if t is on plane of a,b,c
///returns 1/-1 if t is on positive/negative side of a,b,c
int orientPointPlane( PT a, PT b, PT c, PT t ){
  double v = ( (b-a)^(c^a) )*(t-a);
  return dcmp(v);
}
///projection of point q on plane p
PT projectPointPlane( Plane &p, PT q ){
  return PT( q + p.n*((p.d- p.n*q)/(p.n*p.n)) );
}
///reflection of point q on plane p
PT reflectPointPlane( Plane &p, PT q ){
  return PT( q + p.n*(2.0*((p.d- p.n*q)/(p.n*p.n))) );
}
///assuming a is the center, ab is new x axis
vector<PT> convert3Dto2D( PT a, PT b, PT c, vector<PT>pt ){
  PT n = (b-a)^(c-a),dx = unitVector(b-a),
    dy = unitVector(n^(b-a));
  vector<PT>newpt;
  for( int i = 0; i < pt.size(); i++ )
    newpt.push_back( PT( dx*(pt[i]-a), dy*(pt[i]-a), 0 ) );
  return newpt;
}
double distancePointLine( Line l, PT p ){
  return len(l.d^( p-l.u ))/len(l.d);
}
PT projectPointLine( Line l, PT p ){
  return PT( l.u + l.d*(( (p-l.u)*(l.d) )/(l.d*l.d)) );
}
PT reflectPointLine( Line l, PT p ){
  return PT( projectPointLine(l,p)*2.0 - p );
}
///undefined if line and plane is parallel ie( p.b*l.d = 0 )
PT intersectionLinePlane( Line &l, Plane &p ){
  double k = (p.d - (p.n*l.u))/(p.n*l.d);
  return PT(l.u + l.d*k);
}
Line intersectioPlanePLane( Plane &p1, Plane &p2 ){
  PT d = p1.n^p2.n;
  return Line(d, ((p2.n*p1.d - p1.n*p2.d)^d)/(d*d));
}
double distanceLineLine( Line &l1, Line &l2 ){
  PT d = l1.d^l2.d;
  if( dcmp(len(d))==0 ) return distancePointLine(l1, l2.u);
  return abs( (l2.u-l1.u)*d )/len(d);
}
PT closestPointOnL1fromL2( Line &l1, Line &l2 ){
  PT n = l1.d^l2.d, n3 = l2.d^n;
  ///p is the plane including line l2 and n
  Plane p = Plane( n3, n3*l2.u );
  return intersectionLinePlane( l1, p );
}
///2 planes are parallel if crs product of their normal is 0
///2 planes are parallel if dot product of their normal is 0
///angle between two lines is angle between direction vector
double smallAngleBetweenTwoPlane( Plane p1, Plane p2 ){
  return smallAngle(p1.n, p2.n);
}
double angleBetweenTwoPlane( Plane p1, Plane p2 ){
```

```
  return angleRad(p1.n, p2.n);
}
double smallAngleBetweenPlaneLine( Plane &p1, Line &l1 ){
  return acos(-1.0) - smallAngle(p1.n, l1.d);
}
double tri_area( PT a, PT b, PT c ){
  return 0.5*len((b-a)^(c-a));
}
struct Face{
  PT a, b, c;
  Face(){}
  Face(PT a, PT b, PT c) : a(a), b(b), c(c) {}
  Face( const Face &f ) : a(f.a), b(f.b), c(f.c) {}
};
///phi = longitude, lamda = lattitude
struct Sphere{
  PT cen; double r;
  Sphere(){}
  Sphere( const Sphere &s ) : cen(s.cen), r(s.r) {}
  Sphere( PT cen, double r ) : cen(cen), r(r) {}
  PT convert( double phi, double lamda ){
    return PT( r*cos(phi)*cos(lamda),r*cos(phi)*sin(lamda),
             r*sin(phi) );
  }
};
double surfaceArea( vector<Face> &vec){
  double s = 0;
  for( int i = 0; i < vec.size(); i++ )
    s = s + len((vec[i].b-vec[i].a)^(vec[i].c-vec[i].a));
  return s*0.5;
}
double ploy hedronVolume( vector<Face> &vec ){
  if( vec.size() == 0 ) return 0;
  PT reff = vec[0].a; double vol = 0;
  for( int i = 1; i < vec.size(); i++ ) {
    PT ar = (vec[i].b-vec[i].a)^(vec[i].c - vec[i].a);
    vol += abs( ar*(reff-vec[i].a) );
  }
  return vol/6.0;
}
vector<PT> intersectionLineSphere(PT cen, double r, Line l){
  vector<PT>vec;
  double h2 = r*r - distancePointLine(l, cen)*
                    distancePointLine(l, cen);
  if( dcmp(h2) < 0 ) return vec;
  if( dcmp(h2) == 0 ){
    vec.push_back( projectPointLine(l, cen) );
    return vec;
  }
  PT v = projectPointLine(l, cen);
  PT h = l.d*sqrt(h2)/len(l.d);
  vec.push_back(v+h); vec.push_back(v-h);
  return vec;
}
/// let's consider the case of a spherical triangle ABC.
///It's area is given by r2(a + b + c - pi) where r is
///the radius of the sphere and a; b; c are the amplitudes
///of the three interior angles of ABC
bool InsideATriangle (PT A , PT B , PT C , PT P) {
  if (abs(tri_area(A,B,P) + tri_area(A,C,P) +
      tri_area(B,C,P) - tri_area(A,B,C)) < eps) return 1 ;
  return 0 ;
}
///project point c onto line segment through a and b
PT projectPointSegment(PT a, PT b, PT c){
  double r = (b-a)*(b-a);
  if(abs(r) < eps) return a;
  r = ( (c-a)*(b-a) ) / r;
  if (r < 0) return a; if (r > 1) return b;
  return a + (b-a)*r;
```

```
}
///compute distance from c to segment between a and b
double distancePointSegment(PT a, PT b, PT c){
  return dist(c, projectPointSegment(a, b, c));
}
///Minimum distance from Point P on a triangle with vertices
    A,B,C
double PointDistanceOn3dTriangle(PT A, PT B, PT C, PT P){
  Plane ABC = Plane(A,B,C); PT P_ = projectPointPlane(ABC,P);
  double ret = 1e19 ;
  if (InsideATriangle(A,B,C,P_))
      ret = min(ret, dist(P,P_)) ;
  ret = min(ret, distancePointSegment(A,B,P)) ;
  ret = min(ret, distancePointSegment(B,C,P)) ;
  ret = min(ret, distancePointSegment(A,C,P)) ;
  return ret ;
}
vector<Face> Convex3dHull(vector<PT> &V) {
  vector <Face> Faces ;
  for (int i = 0 ; i < V.size() ; i++) {
    for (int j = i+1 ; j < V.size() ; j++) {
      for (int k = j+1 ; k < V.size() ; k++) {
        if (tri_area(V[i],V[j],V[k]) < eps)
          continue ;
        bool up = 0 , down = 0 ;
        PT AB = V[j]-V[i] , AC = V[k]-V[i] ;
        PT normal = AB^AC ;
        for (int l = 0 ; l < V.size() ; l++) {
          if (l == i or l == j or l == k)
            continue ;
          if (abs(normal*(V[l]-V[i])) < eps) {
            if ( abs( ( tri_area(V[i],V[j],V[l]) +
                      tri_area(V[i],V[k],V[l]) +
                      tri_area(V[j],V[k],V[l]) -
                      tri_area(V[j],V[k],V[i]) ) ) < eps ){
              up = down = 1 ;
              break ;
            }
          }
          else if (normal*(V[l]-V[i]) < 0)
            down = 1 ;
          else
            up = 1 ;
        }
        if (up == 0 or down == 0) {
          Face temp;
          temp.a = V[i], temp.b = V[j] , temp.c = V[k] ;
          Faces.push_back(temp) ;
        }
      }
    }
  }
  return Faces ;
}

double greatCirclePointDistance( Sphere s, double phi1,
              double lamda1, double phi2, double lamda2){
  PT p1 = s.convert( phi1, lamda1 );
  PT p2 = s.convert( phi2, lamda2 );
  //always takes into account smallest distance
  return angleRad( p1-s.cen, p2-s.cen )*s.r;
}

double greatCircleArea( Sphere s, double phi1, double lamda1,
    double phi2, double lamda2, double phi3, double lamda3 )
{
  PT p1 = s.convert( phi1, lamda1 ),
  p2 = s.convert( phi2, lamda2 ),p3 = s.convert(phi3,lamda3);
  double a = angleBetweenTwoPlane( Plane(s.cen, p1, p2),
                                   Plane(s.cen, p1, p3) );
```

```
  double b = angleBetweenTwoPlane( Plane(s.cen, p2, p3),
                                   Plane(s.cen, p2, p1) );
  double c = angleBetweenTwoPlane( Plane(s.cen, p3, p1),
                                   Plane(s.cen, p3, p2) );
  return s.r*s.r*( a+b+c-acos(-1.0) );
}
```

2.3 3dGeoTemplate

```
#include <bits/stdc++.h>
using namespace std ;

typedef double ld ;

const ld eps = 1e-12 ;

struct Point {
    ld x , y , z ;
    Point() {}
    Point(ld x_,ld y_,ld z_) {
        x = x_ , y = y_ , z = z_ ;
    }
    Point operator + (Point const &P) {
        return Point(x+P.x,y+P.y,z+P.z) ;
    }
    Point operator - (Point const &P) {
        return Point(x-P.x,y-P.y,z-P.z) ;
    }
    ld operator * (Point const &P) {
        return (x*P.x+y*P.y+z*P.z) ;
    }
    Point operator ^ (Point const &P) {
        return Point(y*P.z-z*P.y,z*P.x-x*P.z,x*P.y-y*P.x) ;
    }
    Point operator / (ld d) {
        return Point(x/d,y/d,z/d) ;
    }
    Point operator * (ld d) {
        return Point(x*d,y*d,z*d) ;
    }
};

ld dist2 (Point A , Point B) {
    return (B-A)*(B-A) ;
}
ld dist(Point A , Point B) {
    return sqrt(dist2(A,B)) ;
}
ld length(Point A) {
    return sqrt(A*A) ;
}

ld tri_area(Point A , Point B , Point C) {
    return length( (B-A)^(C-A) ) /2.0 ;
}

// project point c onto line segment through a and b
Point ProjectPointSegment(Point a, Point b, Point c) {
    double r = (b-a)*(b-a);
    if (fabs(r) < eps) return a;
    r = ( (c-a)*(b-a) ) / r;
    if (r < 0) return a;
    if (r > 1) return b;
    return a + (b-a)*r;
}

// compute distance from c to segment between a and b
double DistancePointSegment(Point a, Point b, Point c) {
    return sqrt(dist2(c, ProjectPointSegment(a, b, c)));
```

```
}

// A , B , C are three points on the plane , calculates P's
   Projected Point on the plane
Point ProjectedPointOnAPlane (Point A , Point B , Point C ,
   Point P) {
   Point AB = B-A , AC = C-A ;
   Point n = AB^AC ;
   Point P_ = P + n * ( (n*(A-P))/(n*n) ) ;
   return P_ ;
}

bool InsideATriangle (Point A , Point B , Point C , Point P) {
   if (abs(tri_area(A,B,P) + tri_area(A,C,P) + tri_area(B,C,P)
      - tri_area(A,B,C)) < eps) return 1 ;
   return 0 ;
}

// Minimum distance from Point P on a triangle with vertices
   A,B,C

ld PointDistanceOn3dTriangle(Point A,Point B,Point C,Point P) {
   Point P_ = ProjectedPointOnAPlane(A,B,C,P) ;
   ld ret = 1e19 ;
   if (InsideATriangle(A,B,C,P_)) {
      ret = min(ret,dist(P,P_)) ;
   }
   ret = min(ret,DistancePointSegment(A,B,P)) ;
   ret = min(ret,DistancePointSegment(B,C,P)) ;
   ret = min(ret,DistancePointSegment(A,C,P)) ;
   return ret ;
}

struct face {
   Point a , b , c ;
};

vector<face> Convex3dHull(vector<Point> &V) {
   vector <face> Faces ;
   for (int i = 0 ; i < V.size() ; i++) {
      for (int j = i+1 ; j < V.size() ; j++) {
         for (int k = j+1 ; k < V.size() ; k++) {
            if (tri_area(V[i],V[j],V[k]) < eps) {
               continue ;
            }
            bool up = 0 , down = 0 ;
            Point AB = V[j]-V[i] , AC = V[k]-V[i] ;
            Point normal = AB^AC ;
            for (int l = 0 ; l < V.size() ; l++) {
               if (l == i or l == j or l == k) {
                  continue ;
               }
               if (abs(normal*(V[l]-V[i])) < eps) {
                  if ( abs( (tri_area(V[i],V[j],V[l]) +
                     tri_area(V[i],V[k],V[l]) +
                     tri_area(V[j],V[k],V[l]) -
                     tri_area(V[j],V[k],V[i]) ) ) < eps ) {
                     up = down = 1 ;
                     break ;
                  }
               }
               else if (normal*(V[l]-V[i]) < 0) {
                  down = 1 ;
               }
               else {
                  up = 1 ;
               }
            }
            if (up == 0 or down == 0) {
               face temp ;
```

```
                temp.a = V[i] , temp.b = V[j] , temp.c = V[k] ;
                Faces.push_back(temp) ;
            }
        }
    }
    return Faces ;
}

// takes spherical co-ordinate (r,phi,theta) , phi is the radian
    angle in XY plane (longitude)

Point Spherical2Cartesian (Point A) {
    return Point(A.x*cos(A.y)*cos(A.z) , A.x*sin(A.y)*cos(A.z) ,
        A.x*sin(A.z)) ;
}

double GeoDistance(Point A , Point B) {
    double R = A.x ;
    A = Spherical2Cartesian(A) ;
    B = Spherical2Cartesian(B) ;
    double d = dist(A,B) ;
    double angleInCenter = 2.0*asin(d/(2.0*R)) ;
    return R*angleInCenter ;
}


int main () {
    //freopen ("in.txt" , "r" , stdin) ;
    int tc ; scanf("%d" , &tc) ;
    while (tc--) {
        int n ; scanf("%d" , &n) ;
        vector <Point> S1 , S2 ;
        for (int i = 1 ; i <= n ; i++) {
            Point P ;
            scanf("%lf %lf %lf" , &P.x , &P.y , &P.z) ;
            S1.push_back(P) ;
        }
        ld surf = 0 ;
        vector <face> Faces1 = Convex3dHull(S1) ;
        for(int i = 0 ; i < Faces1.size() ; i++) {
            surf += tri_area(Faces1[i].a,Faces1[i].b,Faces1[i].c)
                ;
        }
        int m ; scanf("%d",&m) ;
        for (int i = 1 ; i <= m ; i++) {
            Point P ;
            scanf("%lf %lf %lf" , &P.x , &P.y , &P.z) ;
            S2.push_back(P) ;
        }
        vector <face> Faces2 = Convex3dHull(S2) ;
        for(int i = 0 ; i < Faces2.size() ; i++) {
            surf += tri_area(Faces2[i].a,Faces2[i].b,Faces2[i].c)
                ;
        }
        ld di = 1e19 ;
        for (int i = 0 ; i < n ; i++) {
            for (int j = 0 ; j < Faces2.size() ; j++) {
                di =
                    min(di,PointDistanceOn3dTriangle(Faces2[j].a,Faces2[j
                    ;
            }
        }
        for (int i = 0 ; i < m ; i++) {
            for (int j = 0 ; j < Faces1.size() ; j++) {
                di =
                    min(di,PointDistanceOn3dTriangle(Faces1[j].a,Faces1[j
                    ;
            }
        }
```

```
        surf += di ;
        printf ("%.12f\n" , surf) ;
    }
    return 0 ;
}
```

2.4 AllAboutHull

```
/// All About Convex Hull....
struct Point{
    bool operator < (const Point &p) const {
        return make_pair(x,y) < make_pair(p.x,p.y) ;
    }
    bool operator > (const Point &p) const {
        return make_pair(x,y) > make_pair(p.x,p.y) ;
    }
};
struct ConvexHull {
        vector<Point> hull, lower, upper;
        int n;
        /// builds convex hull of a set of points
        bool ccw(Point p,Point q,Point r) {
        return ((q-p)^(r-q)) > 0 ;
    }
  ll cross(Point p, Point q, Point r) {
    return (q-p)^(r-q);
  }
  Point LineLineIntersection(Point p1, Point p2, Point q1, Point
      q2) {
    ll a1 = cross(q1, q2, p1), a2 = -cross(q1, q2, p2);
    return (p1 * a2 + p2 * a1) / (a1 + a2);
  }

  void init(vector<Point> &poly) {
    hull.clear() ; lower.clear() ; upper.clear() ;
    sort(poly.begin(),poly.end()) ;
    for(int i = 0 ; i < poly.size() ; i++) {
      while(lower.size() >= 2 and
          !ccw(lower[lower.size()-2],lower.back(),poly[i])) {
          lower.pop_back() ;
      }
      lower.push_back(poly[i]) ;
    }
    for(int i = (int)poly.size()-1 ; i >= 0 ; i--) {
      while(upper.size() >= 2 and
          !ccw(upper[upper.size()-2],upper.back(),poly[i])) {
        upper.pop_back() ;
      }
      upper.push_back(poly[i]) ;
    }
    hull = lower ;
    for(int i = 1 ; i + 1 < upper.size() ; i++)
        hull.push_back(upper[i]) ;
    n = hull.size();
  }
  int sign(ll x) {
    if (x < 0) return -1 ;
    return x > 0 ;
  }
  int crossOp(Point p, Point q, Point r) {
    ll c = (q-p)^(r-q) ;
    if (c < 0) return -1 ;
    return (c > 0) ;
  }
  /// tests if Point p is inside or on the convex polygon
  /// if Point p is on any side a,b is the index of two endpoint
      of the segment
  bool contain(Point p,int&a,int&b){
```

```
        if(p.x < lower[0].x || p.x > lower.back().x)
            return 0;
int id = lower_bound(lower.begin(),
    lower.end(),Point(p.x,-INF)) - lower.begin();
if(lower[id].x == p.x){
            if(lower[id].y > p.y) return 0;
        } else {
            if(crossOp(lower[id-1],lower[id],p) < 0)
                return 0;
            if(crossOp(lower[id-1],lower[id],p) == 0){
    a = id - 1; b = id;
    return 1;
            }
        }
id = lower_bound(upper.begin(),
    upper.end(),Point(p.x,INF),greater<Point>()) -
    upper.begin();
        if(upper[id].x == p.x){
            if(upper[id].y < p.y) return 0;
        } else {
            if(crossOp(upper[id-1],upper[id],p) < 0)
                return 0;
            if(crossOp(upper[id-1],upper[id],p) == 0) {
    a = id - 1 + lower.size() - 1;
    b = id + lower.size() - 1;
    return 1;
            }
        }
        return 1;
    }
    int find(vector<Point>&vec, Point dir){
        int l = 0 , r = vec.size();
        while(l+5<r){
            int L = (l*2+r)/3, R = (l+r*2)/3;
            if(vec[L]*dir > vec[R]*dir)
                r=R;
            else
                l=L;
        }
        int ret = l;
        for(int k = l+1; k < r; k++) if(vec[k]*dir >
            vec[ret]*dir) ret = k;
        return ret;
    }
    /// if there are rays coming from infinite distance in
        dir direction, the furthest Point of the hull is
        returned
    int findFarest(Point dir){
        if(sign(dir.y) > 0 || sign(dir.y) == 0 &&
            sign(dir.x) > 0){
            return ( (int)lower.size()-1 +
                find(upper,dir)) % n;
        } else {
            return find(lower,dir);
        }
    }
    Point get(int l,int r,Point p1,Point p2){
        int sl = crossOp(p1,p2,hull[l%n]);
        while(l+1<r){
            int m = (l+r)>>1;
            if(crossOp(p1,p2,hull[m%n]) == sl)
                l = m;
            else
                r = m;
        }
        return
            LineLineIntersection(p1,p2,hull[l%n],hull[(l+1)%n]);
    }
//Intersection between a line and a convex polygon. O(log(n))
// touching the hull does not count as intersection
```

```
vector<Point> Line_Hull_Intersection(Point p1, Point p2){
        int X = findFarest((p2-p1).rot90());
        int Y = findFarest((p1-p2).rot90());
        if(X > Y) swap(X,Y);
        if(crossOp(p1,p2,hull[X]) * crossOp(p1,p2,hull[Y])
            < 0){
                return {get(X,Y,p1,p2),get(Y,X+n,p1,p2)};
        } else {
                return {};
        }
}
void update_tangent(Point p, int id, int&a,int&b){
        if(crossOp(p,hull[a],hull[id]) > 0) a = id;
        if(crossOp(p,hull[b],hull[id]) < 0) b = id;
}
void binary_search(int l,int r,Point p,int&a,int&b){
        if(l==r) return;
        update_tangent(p,l%n,a,b);
        int sl = crossOp(p,hull[l%n],hull[(l+1)%n]);
        while(l+1<r){
                int m = l+r>>1;
                if(crossOp(p,hull[m%n],hull[(m+1)%n]) ==
                    sl)
                        l=m;
                else
                        r=m;
        }
        update_tangent(p,r%n,a,b);
}
void get_tangent(Point p,int&a,int&b){
        if(contain(p,a,b)) {
                return ;
        }
        a = b = 0;
                int id = lower_bound(lower.begin(), lower.end(),p)
                    - lower.begin();
                binary_search(0,id,p,a,b);
                binary_search(id,lower.size(),p,a,b);
                id = lower_bound(upper.begin(),
                    upper.end(),p,greater<Point>()) -
                    upper.begin();
                binary_search((int)lower.size() - 1, (int)
                    lower.size() - 1 + id,p,a,b);
                binary_search((int) lower.size() - 1 + id,(int)
                    lower.size() - 1 + upper.size(),p,a,b);
        }
};
```

2.5 CircleUnionArea

```
/// Circle Union Area
struct Point {
    double x,y ;
    Point(double a=0.0,double b=0.0) {x=a,y=b;}
    Point operator+(const Point &a)const {return
        Point(x+a.x,y+a.y);}
    Point operator-(const Point &a)const {return
        Point(x-a.x,y-a.y);}
    Point operator*(const double &a)const {return
        Point(x*a,y*a);}
    Point operator/(const double &a)const {return
        Point(x/a,y/a);}
    double operator*(const Point &a)const {return x*a.y-y*a.x;}
    double operator/(const Point &a)const {return sqrt(
        (a.x-x)*(a.x-x)+(a.y-y)*(a.y-y) );}
}po[N];
double r[N] ;
const double eps = 1e-7 ;
```

```
const double pi = acos(-1.0) ;
int sgn(double x) {
    return fabs(x)<eps?0:(x>0.0?1:-1) ;
}
pair<double,bool> ARG[2*N] ;
double cir_union(Point c[],double r[],int n) {
    double sum = 0.0 , sum1 = 0.0 ,d,p1,p2,p3 ;
    for(int i = 0 ; i < n ; i++) {
        bool f = 1 ;
        for(int j = 0 ; f&&j<n ; j++) {
            if (i!=j and sgn(r[j]-r[i]-c[i]/c[j])!=-1 ) f=0;
        }
        if(!f) swap(r[i],r[--n]),swap(c[i--],c[n]) ;
    }
    for(int i = 0 ; i < n ; i++) {
        int k = 0 , cnt = 0 ;
        for(int j = 0 ; j < n ; j++) {
            if(i!=j and sgn((d=c[i]/c[j])-r[i]-r[j])<=0) {
                p3 = acos((r[i]*r[i]+d*d-r[j]*r[j])/(2.0*r[i]*d)) ;
                p2 = atan2(c[j].y-c[i].y,c[j].x-c[i].x) ;
                p1 = p2-p3 ;
                p2 = p2+p3 ;
                if(sgn(p1+pi)==-1) p1+=2*pi,cnt++;
                if(sgn(p2-pi)==1) p2-=2*pi,cnt++;
                ARG[k++] = make_pair(p1,0);
                ARG[k++] = make_pair(p2,1);
            }
        }
        if(k) {
            sort(ARG,ARG+k) ;
            p1 = ARG[k-1].first-2*pi;
            p3 = r[i]*r[i] ;
            for(int j = 0 ; j < k ; j++) {
                p2 = ARG[j].first;
                if(cnt==0) {
                    sum += (p2-p1-sin(p2-p1))*p3 ;
                    sum1 +=
                        (c[i]+Point(cos(p1),sin(p1))*r[i])*(c[i]+Point(co
                }
                p1 = p2 ;
                ARG[j].second ? cnt--:cnt++;
            }
        }
        else {
            sum += 2*pi*r[i]*r[i] ;
        }
    }
    return (sum+fabs(sum1))*0.5 ;
}
```

2.6 SimpsonIntegration

```
// We divide the integration segment[a;b] into 2n equal parts
// number of steps (already multiplied by 2)
const int N = 1000 * 1000;
double simpson_integration(double a, double b){
    double h = (b - a) / N;
    double s = f(a) + f(b); // a = x_0 and b = x_2n
    for (int i = 1; i <= N - 1; ++i) {
        double x = a + h * i;
        s += f(x) * ((i & 1) ? 4 : 2);
    }
    s *= h / 3;
    return s;
}
```

2.7 circle polygon intersection area

```
/// maximum convex polygon and circle intersection area
/// given polygon and circle's radius
/// n*log^2(n)

#include <bits/stdc++.h>
using namespace std;

typedef double ld;
const ld eps = 1e-12, PI = acos(-1);

int dcmp(ld x) {
    if (fabs(x)<eps) return 0; return x < 0 ? -1 : 1;
}

int sign(ld x) {
    if(x < 0) return -1;
    return 1;
}

struct PT {
  ld x,y;
  PT() {}
  PT(ld x,ld y) : x(x), y(y) {}
  PT(const PT &p) : x(p.x), y(p.y) {}
  ld len() {
    return sqrt(x*x+y*y);
  }
};
PT operator +(PT a,PT b) {
  return PT(a.x+b.x,a.y+b.y);
}
PT operator -(PT a,PT b) {
  return PT(a.x-b.x,a.y-b.y);
}
PT operator *(PT a,ld b) {
  return PT(a.x*b,a.y*b);
}
PT operator /(PT a,ld b){
  return PT(a.x/b,a.y/b);
}
ld operator *(PT a,PT b){ //dot
    return a.x*b.x+a.y*b.y;
}
ld operator ^(PT a,PT b){ //cross
    return a.x*b.y-a.y*b.x;
}

ld len(PT p) {
    return sqrt(p*p);
}

ld sqr(ld x) {
    return x*x;
}
ld det(PT a, PT b) {
    return a^b;
}
ld dot(PT a, PT b) {
    return a*b;
}

ld clamp(ld x, ld l, ld r) {
    if(x < l) return l;
    if(x > r) return r;
    return x;
}

double areaCT(PT pa, PT pb, double r) {
```

```
        if (pa.len() < pb.len()) swap(pa, pb);
        if (dcmp(pb.len()) == 0) return 0;
        double a = pb.len(), b = pa.len(), c = (pb - pa).len();
        double sinB = fabs(det(pb, pb - pa) / a / c),
               cosB = dot(pb, pb - pa) / a / c,
               sinC = fabs(det(pa, pb) / a / b),
               cosC = dot(pa, pb) / a / b;
        double B = atan2(sinB, cosB), C = atan2(sinC, cosC);
        double S = 0.;
        if (a > r) {
               S = C / 2 * r * r;
               double h = a * b * sinC / c;
               if (h < r && B < PI / 2) {
                      S -= (acos(h / r) * r * r - h * sqrt(r * r
                          - h * h));
               }
        } else if (b > r) {
               double theta = PI - B - asin(clamp(sinB / r *
                   a,-1,+1));
               S = a * r * sin(theta) / 2 + (C - theta) / 2 * r *
                   r;
        } else {
               S = sinC * a * b / 2;
        }
        return S;
}

ld poly_cross(vector<PT> P, PT cen, ld r) {
    int n = P.size();
    ld ans = 0;
    for(int i = 0; i < n; i++) {
        ld cur_area = fabs(areaCT(P[i]-cen, P[(i+1)%n]-cen,
            r))*sign(det(P[i]-cen,P[(i+1)%n]-cen));
        ans += cur_area;
    }
    return ans;
}


/// intersection of line ab and cd
PT ComputeLineIntersection(PT a, PT b, PT c, PT d) {
  b=b-a;d=c-d;c=c-a;
  assert((b*b) > eps && (d*d) > eps);
  return a + b*(c^d)/(b^d);
}

const ld inf = 1e100;
int n;
ld Rad;

ld f(double X, vector<PT> P) {
    int n = P.size();
    double U = -1e100, D = 1e100;
    for(int i = 0; i < n; i++) {
        ld x1 = P[i].x, x2 = P[(i+1)%n].x;
        if(dcmp(x1-X) == 0 && dcmp(x1-x2) == 0) {
            U = max(U, P[i].y);
            U = max(U, P[(i+1)%n].y);
            D = min(D, P[i].y);
            D = min(D, P[(i+1)%n].y);
            continue;
        }
        if(x1 > x2) swap(x1, x2);
        if(X >= x1 && X <= x2) {
            PT ints = ComputeLineIntersection(P[i], P[(i+1)%n],
                PT(X,0), PT(X,1));
            U = max(U, ints.y);
            D = min(D, ints.y);
        }
    }
```

```
ld lo = D, hi = 220;
for(int itr = 1; itr <= 100; itr++) {
    ld mid = (lo+hi)/2.0;
    if(poly_cross(P, PT(X,mid),Rad) < eps) {
        hi = mid;
    }
    else {
        lo = mid;
    }
}
ld upper = lo;

lo = -200, hi = U;
for(int itr = 1; itr <= 100; itr++) {
    ld mid = (lo+hi)/2.0;
    if(poly_cross(P, PT(X,mid),Rad) < eps) {
        lo = mid;
    }
    else {
        hi = mid;
    }
}
ld lower = lo;
ld ans = 0;
for(int itr = 1; itr <= 100; itr++) {
    ld m1 = (lower+lower+upper)/3.0, m2 =
        (lower+upper+upper)/3.0;
    ld ar1 = poly_cross(P, PT(X,m1),Rad), ar2 = poly_cross(P,
        PT(X,m2), Rad);
    ans = max(ans, ar1);
    ans = max(ans, ar2);
    if(ar1 < ar2) lower = m1;
    else upper = m2;
}
    return ans;
}




int main() {

    cout << setprecision(12) << fixed;
    cin >> n >> Rad;
    vector<PT> P(n);
    ld L = inf, R = -inf;
    for(int i = 0; i < n; i++) {
        cin >> P[i].x >> P[i].y;
        L = min(L, P[i].x);
        R = max(R, P[i].x);
    }
    double ans = 0;
    for(int itr = 1; itr <= 100; itr++) {
        ld m1 = (L+L+R)/3.0, m2 = (L+R+R)/3.0;
        ld ar1 = f(m1, P), ar2 = f(m2, P);
        ans = max(ans, ar1);
        ans = max(ans, ar2);
        if(ar1 < ar2) L = m1;
        else R = m2;
    }
    cout << ans << endl;
}
```

2.8 common area of two circle

```
#include <bits/stdc++.h>
using namespace std;
```

```
typedef long long ll;


typedef double ld;

ld eps = 1e-9;


int dcmp(ld x){
    if(fabs(x) < eps) return 0;
    if(x < 0) return -1;
    return 1;
}

struct PT {
  double x,y;
  PT() {}
  PT(double x,double y) : x(x), y(y) {}
  PT(const PT &p) : x(p.x), y(p.y) {}
  bool operator == (const PT &p) const {
    return dcmp(x-p.x)==0 && dcmp(y-p.y)==0;
  }
};
PT operator +(PT a,PT b) {
  return PT(a.x+b.x,a.y+b.y);
}
PT operator -(PT a,PT b) {
  return PT(a.x-b.x,a.y-b.y);
}
PT operator *(PT a,double b) {
  return PT(a.x*b,a.y*b);
}
PT operator /(PT a,double b){
  return PT(a.x/b,a.y/b);
}
```

```
double operator *(PT a,PT b){ //dot
    return a.x*b.x+a.y*b.y;
}
double operator ^(PT a,PT b){ //cross
    return a.x*b.y-a.y*b.x;
}

const double pi = acos(-1.0);

double len(PT p) {
    return sqrt(p*p);
}

double angleRad( PT a, PT b ){
  return acos( max(-1.0, min(1.0, (a*b)/(len(a)*len(b)))) );
}
ld angle(PT v) {
  return atan2(v.y,v.x);
}
int getCircleCircleIntersection(PT C1,double r1, PT C2, double
    r2, vector<PT>& sol) {
  ld d = len(C1-C2);
  if (dcmp(d)==0){
    if (dcmp(r1 - r2)==0) return -1; //same circle
    return 0; //concentric circle
  }
  if (dcmp(r1+r2-d)<0) return 0; //no intersection, outside
  if (dcmp(fabs(r1-r2)-d)>0) return 0; //no intersection, inside
  ld a = angle(C2-C1);
  ld da = acos((r1*r1+d*d - r2*r2)/ (2*r1*d));
  PT p1 = PT(C1.x+cos(a-da)*r1,C1.y+sin(a-da)*r1);
  PT p2 = PT(C1.x+cos(a+da)*r1,C1.y+sin(a+da)*r1);
  sol.push_back(p1);if (p1==p2) return 1;
  sol.push_back(p2);
  return 2;
```

```
}

double circle_portion(PT c, double r, PT ints, PT c2) {
    double ang = angleRad(c2-c,ints-c);
    if(ang <= pi/2.0) {
        return r*r*(ang-sin(ang)*cos(ang));
    }
    return r*r*(ang + sin(pi-ang)*cos(pi-ang));
}

double common_area_between_two_circle(PT c1, double r1, PT c2,
    double r2) {
    if(r1 < r2) swap(c1,c2), swap(r1,r2);
    double distance_between_centers = sqrt(((c1-c2)*(c1-c2)));
    if(distance_between_centers >= r1+r2) return 0.0;
    if(distance_between_centers <= abs(r1-r2)) return pi*r2*r2;
    vector<PT> ints;
    getCircleCircleIntersection(c1,r1,c2,r2,ints);
    assert(ints.size() == 2);
    return circle_portion(c1,r1,ints[0],c2) +
           circle_portion(c2,r2,ints[0],c1);
}




int main() {

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

    cout << setprecision(12) << fixed;
    int tc, caseno = 1;
    cin >> tc;
    while(tc--) {
        PT c1, c2;
        double r1, r2;
        cin >> c1.x >> c1.y >> r1;
        cin >> c2.x >> c2.y >> r2;
        double ans =
            max(0.0,common_area_between_two_circle(c1,r1,c2,r2));
        cout << "Case " << caseno++ << ": " << ans << endl;
    }

    return 0;
}
```

2.9 geo library vovuh

```
/// https://codeforces.com/gym/101205/submission/23409200
#include <bits/stdc++.h>
#define MP make_pair
#define PB push_back
#define int long long
#define st first
#define nd second
#define rd third
#define FOR(i, a, b) for(int i =(a); i <=(b); ++i)
#define RE(i, n) FOR(i, 1, n)
#define FORD(i, a, b) for(int i = (a); i >= (b); --i)
#define REP(i, n) for(int i = 0;i <(n); ++i)
#define VAR(v, i) __typeof(i) v=(i)
#define FORE(i, c) for(VAR(i, (c).begin()); i != (c).end(); ++i)
#define ALL(x) (x).begin(), (x).end()
#define SZ(x) ((int)(x).size())
using namespace std;
template<typename TH> void _dbg(const char* sdbg, TH h) {
    cerr<<sdbg<<"="<<h<<"\n"; }
template<typename TH, typename... TA> void _dbg(const char*
    sdbg, TH h, TA... t) {
```

```
  while(*sdbg != ',')cerr<<*sdbg++; cerr<<"="<<h<<",";
      _dbg(sdbg+1, t...);
}
#ifdef LOCAL
#define debug(...) _dbg(#__VA_ARGS__, __VA_ARGS__)
#define debugv(x) {{cerr <<#x <<" = "; FORE(itt, (x)) cerr
    <<*itt <<", "; cerr <<"\n"; }}
#else
#define debug(...) (__VA_ARGS__)
#define debugv(x)
#define cerr if(0)cout
#endif
#define make(type, x) type x; cin>>x;
#define make2(type, x, y) type x, y; cin>>x>>y;
#define make3(type, x, y, z) type x, y, z; cin>>x>>y>>z;
#define make4(type, x, y, z, t) type x, y, z, t; cin>>x>>y>>z>>t;
#define next ____next
#define prev ____prev
#define left ____left
#define hash ____hash
typedef long long ll;
typedef pair<int, int> PII;
typedef pair<ll, ll> PLL;
typedef vector<int> VI;
typedef vector<VI> VVI;
typedef vector<ll> VLL;
typedef vector<pair<int, int> > VPII;
typedef vector<pair<ll, ll> > VPLL;

template<class C> void mini(C&a4, C b4){a4=min(a4, b4); }
template<class C> void maxi(C&a4, C b4){a4=max(a4, b4); }
template<class T1, class T2>
ostream& operator<< (ostream &out, pair<T1, T2> pair) { return
    out << "(" << pair.first << ", " << pair.second << ")";}
```

```
template<class A, class B, class C> struct Triple { A first; B
    second; C third;
  bool operator<(const Triple& t) const { if (st != t.st) return
      st < t.st; if (nd != t.nd) return nd < t.nd; return rd <
      t.rd; } };
template<class T> void ResizeVec(T&, vector<int>) {}
template<class T> void ResizeVec(vector<T>& vec, vector<int> sz)
    {
  vec.resize(sz[0]); sz.erase(sz.begin()); if (sz.empty()) {
      return; }
  for (T& v : vec) { ResizeVec(v, sz); }
}
typedef Triple<int, int, int> TIII;
template<class A, class B, class C>
ostream& operator<< (ostream &out, Triple<A, B, C> t) { return
    out << "(" << t.st << ", " << t.nd << ", " << t.rd << ")"; }
template<class T> ostream& operator<<(ostream& out, vector<T>
    vec) { out<<"("; for (auto& v: vec) out<<v<<", "; return
    out<<")"; }


typedef long double LD;

const LD kEps = 1e-9;
const LD kPi = 2 * acos(0);
LD Sq(LD x) {
  return x * x;
}
struct Point {
  LD x, y;
  Point() {}
  Point(LD a, LD b) : x(a), y(b) {}
  Point(const Point& a) : x(a.x), y(a.y) {}
  void operator=(const Point& a) { x = a.x; y = a.y; }
```

```
Point operator+(const Point& a) const { Point p(x + a.x, y +
    a.y); return p; }
Point operator-(const Point& a) const { Point p(x - a.x, y -
    a.y); return p; }
Point operator*(LD a) const { Point p(x * a, y * a); return p;
    }
Point operator/(LD a) const { assert(a > kEps); Point p(x / a,
    y / a); return p; }
Point& operator+=(const Point& a) { x += a.x; y += a.y; return
    *this; }
Point& operator-=(const Point& a) { x -= a.x; y -= a.y; return
    *this; }
Point& operator*=(LD a) { x *= a; y *= a; return *this;}
Point& operator/=(LD a) { assert(a > kEps); x /= a; y /= a;
    return *this; }

bool IsZero() const {
  return abs(x) < kEps && abs(y) < kEps;
}
bool operator==(const Point& a) const {
  return (*this - a).IsZero();
}
LD CrossProd(const Point& a) const {
  return x * a.y - y * a.x;
}
LD CrossProd(Point a, Point b) const {
  a -= *this;
  b -= *this;
  return a.CrossProd(b);
}
LD DotProd(const Point& a) const {
  return x * a.x + y * a.y;
}
LD Norm() const {
  return sqrt(Sq(x) + Sq(y));
}
void NormalizeSelf() {
  *this /= Norm();
}
Point Normalize() {
  Point res(*this);
  res.NormalizeSelf();
  return res;
}
LD Dist(const Point& a) const {
  return (*this - a).Norm();
}
LD Angle() const {
  return atan2(y, x);
}
void RotateSelf(LD angle) {
  LD c = cos(angle);
  LD s = sin(angle);
  LD nx = x * c - y * s;
  LD ny = y * c + x * s;
  y = ny;
  x = nx;
}
Point Rotate(LD angle) const {
  Point res(*this);
  res.RotateSelf(angle);
  return res;
}
static bool LexCmp(const Point& a, const Point& b) {
  if (abs(a.x - b.x) > kEps) {
    return a.x < b.x;
  }
  return a.y < b.y;
}
LD SqNorm() {
```

```
    return x * x + y * y;
  }
  friend ostream& operator<<(ostream& out, Point m);
};

ostream& operator<<(ostream& out, Point p) {
  out << "(" << p.x << ", " << p.y << ")";
  return out;
}

struct Circle {
  Point center;
  LD r;
  Circle(LD x, LD y, LD rad) {
    center = Point(x, y);
    r = rad;
  }
  Circle(const Point& a, LD rad) : center(a), r(rad) {}
  LD Area() const {
    return kPi * Sq(r);
  }
  LD Perimeter() const {
    return 2 * kPi * r;
  }
  LD Diameter() const {
    return 2 * r;
  }
  Point RotateRightMost(LD ang) const {
    return center + Point{r * cos(ang), r * sin(ang)};
  }
  bool operator==(const Circle& c) const {
    return center == c.center && abs(r - c.r) < kEps;
  }
};
```

```
struct Line {
  Point p[2];
  bool is_seg;
  Line(Point a, Point b, bool is_seg_ = false) {
    p[0] = a;
    p[1] = b;
    is_seg = is_seg_;
  }
  Line() {
  }
  Point& operator[](int a) {
    return p[a];
  }
  Point NormalVector() {
    Point perp = p[1] - p[0];
    perp.RotateSelf(kPi / 2);
    perp.NormalizeSelf();
    return perp;
  }

  // (A, B, C) such that A^2 + B^2 = 1, (A, B) > (0, 0)
  vector<LD> LineEqNormLD() { // seems ok
    LD A = p[1].y - p[0].y;
    LD B = p[0].x - p[1].x;
    LD C = -(A * p[0].x + B * p[0].y);
    assert(abs(A * p[1].x + B * p[1].y + C) < kEps);
    LD norm = sqrt(Sq(A) + Sq(B));
    vector<LD> res{A, B, C};
    for (auto& x : res) { x /= norm; }
    if (A < -kEps || (abs(A) < kEps && B < -kEps)) {
      for (auto& x : res) { x *= -1; }
    }
    return res;
  }
```

```
  // assumes that coordinates are integers!
  vector<int> LineEqNormInt() { // seems ok
    int A = round(p[1].y - p[0].y);
    int B = round(p[0].x - p[1].x);
    int C = -(A * p[0].x + B * p[0].y);
    int gcd = abs(__gcd(A, __gcd(B, C)));
    vector<int> res{A, B, C};
    for (auto& x : res) { x /= gcd; }
    if (A < 0 || (A == 0 && B < 0)) {
      for (auto& x : res) { x *= -1; }
    }
    return res;
  }
};

struct Utils {
  // 0, 1, 2 or 3 pts. In case of 3 pts it means they are equal
  static vector<Point> InterCircleCircle(Circle a, Circle b) {
    if (a.r + kEps < b.r) {
      swap(a, b);
    }
    if (a == b) {
      return vector<Point>{a.RotateRightMost(0),
          a.RotateRightMost(2 * kPi / 3),
          a.RotateRightMost(4 * kPi / 3)};
    }
    Point diff = b.center - a.center;
    LD dis = diff.Norm();
    LD ang = diff.Angle();
    LD longest = max(max(a.r, b.r), dis);
    LD per = a.r + b.r + dis;
    if (2 * longest > per + kEps) {
      return vector<Point>();
    }
    if (abs(2 * longest - per) < 2 * kEps) {
      return vector<Point>{a.RotateRightMost(ang)};
    }
    LD ang_dev = acos((Sq(a.r) + Sq(dis) - Sq(b.r)) / (2 * a.r *
        dis));
    return vector<Point>{a.RotateRightMost(ang - ang_dev),
        a.RotateRightMost(ang + ang_dev)};
  }

  static vector<Point> InterLineLine(Line& a, Line& b) { //
      working fine
    Point vec_a = a[1] - a[0];
    Point vec_b1 = b[1] - a[0];
    Point vec_b0 = b[0] - a[0];
    LD tr_area = vec_b1.CrossProd(vec_b0);
    LD quad_area = vec_b1.CrossProd(vec_a) +
        vec_a.CrossProd(vec_b0);
    if (abs(quad_area) < kEps) { // parallel or coinciding
      if (PtBelongToLine(b, a[0])) {
        return {a[0], a[1]};
      } else {
        return {};
      }
    }
    return {a[0] + vec_a * (tr_area / quad_area)};
  }

  static Point ProjPointToLine(Point p, Line l) { ///Tested
    Point diff = l[1] - l[0];
    return l[0] + diff * (diff.DotProd(p - l[0]) /
        diff.DotProd(diff));
  }

  static Point ReflectPtWRTLine(Point p, Line l) {
    Point proj = ProjPointToLine(p, l);
    return proj * 2 - p;
```

```
}

static vector<Point> InterCircleLine(Circle c, Line l) { ///
    Tested here:
    http://codeforces.com/gym/100554/submission/10197624
  Point proj = ProjPointToLine(c.center, l);
  LD dis_proj = c.center.Dist(proj);
  if (dis_proj > c.r + kEps) { return vector<Point>(); }
  LD a = sqrt(Sq(c.r) - Sq(dis_proj));
  Point dir = l[1] - l[0];
  LD dir_norm = dir.Norm();
  vector<Point> cands{proj + dir * (a / dir_norm), proj - dir
      * (a / dir_norm)};
  if (cands[0].Dist(cands[1]) < kEps) { return
      vector<Point>{proj}; }
  return cands;
}

static bool PtBelongToLine(Line l, Point p) {
  return abs(l[0].CrossProd(l[1], p)) < kEps;
}

static bool PtBelongToSeg(Line l, Point p) { // seems ok
  return abs(p.Dist(l[0]) + p.Dist(l[1]) - l[0].Dist(l[1])) <
      kEps;
}

static vector<Point> InterCircleSeg(Circle c, Line l) {
    //seems ok
  vector<Point> from_line = InterCircleLine(c, l);
  vector<Point> res;
  for (auto p : from_line) {
    if (PtBelongToSeg(l, p)) { res.PB(p); }
  }
  return res;
}

static vector<Point> TangencyPtsToCircle(Circle c, Point p) {
    // seems ok
  LD d = c.center.Dist(p);
  if (d < c.r - kEps) { return {}; }
  if (d < c.r + kEps) { return {p}; }
  LD from_cent = (p - c.center).Angle();
  LD ang_dev = acos(c.r / d);
  return {c.RotateRightMost(from_cent - ang_dev),
      c.RotateRightMost(from_cent + ang_dev)};
}

// outer and inner tangents tested only locally (however I
    believe that rigorously)
static vector<Line> OuterTangents(Circle c1, Circle c2) {
  if (c1 == c2) { return {}; } // is it surely best choice?
  if (c1.r < c2.r) { swap(c1, c2); }
  if (c2.r + c1.center.Dist(c2.center) < c1.r - kEps) { return
      {}; }
  if (abs(c1.r - c2.r) < kEps) {
    Point diff = c2.center - c1.center;
    Point R = diff.Rotate(kPi / 2) * (c1.r / diff.Norm());
    return {{c1.center + R, c2.center + R}, {c1.center - R,
        c2.center - R}};
  }
  Point I = c1.center + (c2.center - c1.center) * (c1.r /
      (c1.r - c2.r));
  if (c2.r + c1.center.Dist(c2.center) < c1.r + kEps) {
    return {{I, I + (c2.center - c1.center).Rotate(kPi / 2)}};
  }
  vector<Point> to1 = TangencyPtsToCircle(c1, I);
  vector<Point> to2 = TangencyPtsToCircle(c2, I);
  vector<Line> res{{to1[0], to2[0]}, {to1[1], to2[1]}};
  assert(Utils::PtBelongToLine(res[0], I));
```

```
  assert(Utils::PtBelongToLine(res[1], I));
  return res;
}

// unfortunately big part of code is same as in previous
    function
// can be joined when putting appropriate signs in few places
// however those ifs differ a bit hence it may not be good idea
// to necessarily join them
static vector<Line> InnerTangents(Circle c1, Circle c2) {
  if (c1 == c2) { return {}; } // this time surely best choice
  if (c1.r < c2.r) { swap(c1, c2); }
  LD d = c1.center.Dist(c2.center);
  if (d < c1.r + c2.r - kEps) { return {}; }
  Point I = c1.center + (c2.center - c1.center) * (c1.r /
      (c1.r + c2.r));
  if (d < c1.r + c2.r + kEps) {
    return {{I, I + (c2.center - c1.center).Rotate(kPi / 2)}};
  }
  vector<Point> to1 = TangencyPtsToCircle(c1, I);
  vector<Point> to2 = TangencyPtsToCircle(c2, I);
  vector<Line> res{{to1[0], to2[0]}, {to1[1], to2[1]}};
  assert(Utils::PtBelongToLine(res[0], I));
  assert(Utils::PtBelongToLine(res[1], I));
  return res;
}

static bool AreParallel(Line l1, Line l2) { // seems ok
  return abs(l1[0].CrossProd(l2[0], l1[1]) -
      l1[0].CrossProd(l2[1], l1[1])) < kEps;
}

// returns a vector of points such that their convex hull is
    intersection of those segments
// SZ(res) == 0 => empty intersection, SZ(res) == 1 =>
    intersection is a point, SZ(res) == 2 => intersection is a
    segment
static vector<Point> InterSegs(Line l1, Line l2) { // seems ok
  if (!Point::LexCmp(l1[0], l1[1])) { swap(l1[0], l1[1]); }
  if (!Point::LexCmp(l2[0], l2[1])) { swap(l2[0], l2[1]); }
  if (AreParallel(l1, l2)) {
    if (!PtBelongToLine(l1, l2[0])) { return vector<Point>(); }
    vector<Point> ends(2);
    for (int tr = 0; tr < 2; tr++) {
      if (Point::LexCmp(l1[tr], l2[tr]) ^ tr) {
        ends[tr] = l2[tr];
      } else {
        ends[tr] = l1[tr];
      }
    }
    if ((ends[1] - ends[0]).IsZero()) {
      ends.pop_back();
    }
    if (SZ(ends) == 2 && Point::LexCmp(ends[1], ends[0])) {
        return vector<Point>(); }
    return ends;
  } else {
    vector<Point> p = InterLineLine(l1, l2);
    if (PtBelongToSeg(l1, p[0]) && PtBelongToSeg(l2, p[0])) {
        return p; }
    return vector<Point>();
  }
}

static LD Angle(Point P, Point Q, Point R) { // angle PQR
  LD ang2 = (P - Q).Angle();
  LD ang1 = (R - Q).Angle();
  LD ans = ang1 - ang2;
  if (ans < kEps) {
```

```
    ans += 2 * kPi;
  }
  return ans;
}

// tested here:
    http://codeforces.com/contest/600/submission/14961583
// DON'T change anything as this will lead to precision errors
// don't know why, but this is the only version which works
    precisely even for very mean cases
static LD DiskInterArea(Circle c1, Circle c2) { // tested
    here:
    http://opentrains.snarknews.info/~ejudge/team.cgi?contest_id=006254
    problem I
  if (c1.r < c2.r) {
    swap(c1, c2);
  }
  LD d = c1.center.Dist(c2.center);
  if (c1.r + c2.r < d + kEps) {
    return 0;
  }
  if (c1.r - c2.r > d - kEps) {
    return kPi * Sq(c2.r);
  }
  LD alfa = acos((Sq(d) + Sq(c1.r) - Sq(c2.r)) / (2 * d *
      c1.r));
  LD beta = acos((Sq(d) + Sq(c2.r) - Sq(c1.r)) / (2 * d *
      c2.r));
  return alfa * Sq(c1.r) + beta * Sq(c2.r) - sin(2 * alfa) *
      Sq(c1.r) / 2 - sin(2 * beta) * Sq(c2.r) / 2;
}

static Line RadAxis(Circle c1, Circle c2) {
  LD d = c1.center.Dist(c2.center);
  LD a = (Sq(c1.r) - Sq(c2.r) + Sq(d)) / (2 * d);
    Point Q = c1.center + (c2.center - c1.center) * (a / d);
    Point R = Q + (c2.center - c1.center).Rotate(kPi / 2);
    return Line(Q, R);
  }
};

struct Polygon {
  vector<Point> pts;
  Polygon(vector<Point> pts_) : pts(pts_) {}
  Polygon() : Polygon(vector<Point>()) {}
  void Add(Point p) {
    pts.push_back(p);
  }
  // positive for counterclockwise
  double Area() {
    double area = 0;
    for (int i = 0; i < SZ(pts); i++) {
      area += pts[i].CrossProd(pts[(i + 1) % SZ(pts)]);
    }
    area /= 2;
    return area;
  }
  void OrientCounterclockwise() {
    if (Area() < 0) {
      reverse(pts.begin(), pts.end());
    }
  }
  int next(int a) {
    if (a + 1 < SZ(pts)) {
      return a + 1;
    }
    return 0;
  }
  pair<int, int> FurthestPair() { // tested here:
      http://codeforces.com/contest/333/submission/11058065
```

```
  MakeConvexHull();
  OrientCounterclockwise();
  int furth = 1;
  pair<int, int> best_pair = make_pair(0, 0);
  double best_dis = 0;
  for (int i = 0; i < SZ(pts); i++) {
    Point side = pts[next(i)] - pts[i];
    while (side.CrossProd(pts[furth] - pts[i]) <
        side.CrossProd(pts[next(furth)] - pts[i])) {
      furth = next(furth);
    }
    vector<int> vec{i, next(i)};
    for (auto ind : vec) {
      if (pts[ind].Dist(pts[furth]) > best_dis) {
        best_pair = make_pair(ind, furth);
        best_dis = pts[ind].Dist(pts[furth]);
      }
    }
    cerr<<"Furthest from: "<<pts[i]<<"-"<<pts[next(i)]<<" is
        "<<pts[furth]<<endl;
  }
  return best_pair;
}
void MakeConvexHull() { // tested everywhere
    http://codeforces.com/contest/333/submission/11058065
  vector<Point> one_way_hull[2];
  sort(pts.begin(), pts.end(), Point::LexCmp);
  for (int dir = -1; dir <= 1; dir += 2) {
    int hull_num = (dir + 1) / 2;
    auto& H = one_way_hull[hull_num];
    one_way_hull[hull_num].push_back(pts[0]);
    if (SZ(pts) > 1) {
      H.push_back(pts[1]);
    }
    for (int i = 2; i < SZ(pts); i++) {
      while (SZ(H) >= 2 &&
          dir * (pts[i] - H[SZ(H) - 2]).CrossProd(H.back() -
              H[SZ(H) - 2]) > -kEps) {
        H.pop_back();
      }
      H.push_back(pts[i]);
    }
  }
  pts.clear();
  for (auto p : one_way_hull[1]) {
    pts.push_back(p);
  }
  for (int i = SZ(one_way_hull[0]) - 2; i >= 1; i--) {
    pts.push_back(one_way_hull[0][i]);
  }
}

// without sides
vector<vector<bool>> InsideDiagonalsMatrix() { // tested here:
    http://codeforces.com/contest/438/submission/11063385
  int n = pts.size();
  vector<vector<bool>> res(n, vector<bool>(n));
  for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
      Line diag(pts[i], pts[j]);
      if (i == j || abs(i - j) == 1 || abs(i - j) == n - 1) {
          continue; }
      res[i][j] = 1;
      for (int k = 0; k < n; k++) {
        int kk = next(k);
        Line side(pts[k], pts[kk]);
        if (k == i || k == j || kk == i || kk == j) { continue;
            }
        vector<Point> inter = Utils::InterSegs(diag, side);
        if (SZ(inter)) { res[i][j] = 0; }
```

```
      }
      int act = next(i);
      LD areas[2] = {0, 0};
      int passed_j = 0;
      while (act != i) {
        passed_j |= (act == j);
        areas[passed_j] += pts[i].CrossProd(pts[act],
            pts[next(act)]);
        act = next(act);
      }
      if (areas[0] * areas[1] < kEps) {
        res[i][j] = 0;
      }
    }
  }
  return res;
 }
};

// CLIP START
bool InUpper(Point a) {
 if (abs(a.y) > kEps) {
  return a.y > 0;
 }
 return a.x > 0;
}

bool angle_cmp(const Point a, const Point b) {
 bool u = InUpper(a);
 bool v = InUpper(b);
 return u!=v ? u : a.CrossProd(b)>0;
}

/**
 * @brief a+(b-a)*f \in c+lin(d-c)
 * @returns f
 */
LD cross(Point a, Point b, Point c, Point d) {
 return (d - c).CrossProd(a - c) / (d - c).CrossProd(a - b);
}

struct ClipLine { // valid side is on left
 ClipLine(Point A, Point B) : al(A), bl(B), a(A), b(B) {};
 Point al,bl; // original line points
 mutable Point a,b; // actual intersection points
 Point dir() const { return bl - al; }
 bool operator<(const ClipLine& l) const { return
    angle_cmp(dir(),l.dir()); }
 Point cross(const ClipLine& l) {
  return al + (bl - al) * ::cross(al, bl, l.al, l.bl);
 }
 bool left(Point p) {
  return (bl - al).CrossProd(p - al) > 0;
 }
};

struct Clip {
 Clip(LD r) : area(4*r*r) {
  Point a{-r,-r}, b{r,-r}, c{r,r}, d{-r,r};
  lines = {ClipLine(a,b), ClipLine(b,c), ClipLine(c,d),
     ClipLine(d,a)};
 }

 void insert(Line l) { insert(ClipLine(l[0], l[1])); }

 void insert(ClipLine l) {
  assert(abs(l.dir().SqNorm()) > kEps);
  find(l);
  while (size() && !l.left(it->a) && !l.left(it->b)) {
     erase(); }
```

```
  if (size()) {
    while (prev(), size() && !l.left(it->a) && !l.left(it->b))
       { erase(); }
  }
  if (size() && (!l.left(it->a) || !l.left(it->b))) {
    l.a = l.cross(*it);
    area -= l.a.CrossProd(it->b)*.5; it->b = l.a; next();
    l.b = l.cross(*it);
    if ((l.a-l.b).SqNorm() < kEps) {
      l.b = l.a;
    }
    area -= it->a.CrossProd(l.b) * .5;
    it->a = l.b;
    if (!(l.a - l.b).IsZero()) {
      area += l.a.CrossProd(l.b)*.5;
      lines.insert(l);
    }
  }
  //assert(l.dir().SqNorm()>1e-13);
}

void find(const ClipLine &l) {
  it = lines.lower_bound(l);
  if (it == lines.end()) { it = lines.begin(); }
}

void recalculate() {
  area = 0; for (const ClipLine &l : lines) area +=
     l.a.CrossProd(l.b);
  area *= .5;
}

int size() { return lines.size(); }
void next() { if(++it==lines.end()) it = lines.begin(); }
void prev() { if(it==lines.begin()) it = lines.end(); --it; }
  void erase() {
     assert(it!=lines.end());
     area -= it->a.CrossProd(it->b)*.5;
     it = lines.erase(it);
     if(it==lines.end()) it = lines.begin();
  }
  typename set<ClipLine>::iterator it;
  set<ClipLine> lines;
  LD area;
};
// CLIP ENDS

// CENTERS BEGIN
Point Bary(Point A, Point B, Point C, LD a, LD b, LD c) {
    return (A * a + B * b + C * c) / (a + b + c);
}

Point Centroid(Point A, Point B, Point C) {
    return Bary(A, B, C, 1, 1, 1);
}

Point Circumcenter(Point A, Point B, Point C) {
    LD a = (B - C).SqNorm(), b = (C - A).SqNorm(), c = (A -
        B).SqNorm();
    return Bary(A, B, C, a * (b + c - a), b * (c + a - b), c *
        (a + b - c));
}

Point Incenter(Point A, Point B, Point C) {
    return Bary(A, B, C, (B - C).Norm(), (A - C).Norm(), (A -
        B).Norm());
}

Point Orthocenter(Point A, Point B, Point C) {
```

```
    LD a = (B - C).SqNorm(), b = (C - A).SqNorm(), c = (A -
        B).SqNorm();
    return Bary(A, B, C, (a+b-c)*(c+a-b), (b+c-a)*(a+b-c),
        (c+a-b)*(b+c-a));
}

Point Excenter(Point A, Point B, Point C) { // opposite to A
    LD a = (B - C).Norm(), b = (A - C).Norm(), c = (A -
        B).Norm();
    return Bary(A, B, C, -a, b, c);
}


struct Point3 {
  LD x, y, z;
  Point3 operator+(Point3 a) { Point3 p{x + a.x, y + a.y, z +
      a.z}; return p; }
  Point3 operator-(Point3 a) { Point3 p{x - a.x, y - a.y, z -
      a.z}; return p; }
  Point3 operator*(LD a) { Point3 p{x * a, y * a, z * a}; return
      p; }
  Point3 operator/(LD a) { assert(a > kEps); Point3 p{x / a, y /
      a, z / a}; return p; }
  Point3& operator+=(Point3 a) { x += a.x; y += a.y; z += a.z;
      return *this; }
  Point3& operator-=(Point3 a) { x -= a.x; y -= a.y; z -= a.z;
      return *this; }
  Point3& operator*=(LD a) { x *= a; y *= a; z *= a; return
      *this;}
  Point3& operator/=(LD a) { assert(a > kEps); x /= a; y /= a; z
      /= a; return *this; }

  LD& operator[](int a) {
    if (a == 0) { return x; }
    if (a == 1) { return y; }
    if (a == 2) { return z; }
    assert(false);
  }
  bool IsZero() {
    return abs(x) < kEps && abs(y) < kEps && abs(z) < kEps;
  }
  bool operator==(Point3 a) {
    return (*this - a).IsZero();
  }
  LD DotProd(Point3 a) {
    return x * a.x + y * a.y + z * a.z;
  }
  LD Norm() {
    return sqrt(x * x + y * y + z * z);
  }
  LD SqNorm() {
    return x * x + y * y + z * z;
  }
  void NormalizeSelf() {
    *this /= Norm();
  }
  Point3 Normalize() {
    Point3 res(*this);
    res.NormalizeSelf();
    return res;
  }
  LD Dis(Point3 a) {
    return (*this - a).Norm();
  }
  pair<LD, LD> SphericalAngles() {
    return {atan2(z, sqrt(x * x + y * y)), atan2(y, x)};
  }
  LD Area(Point3 p) {
    return Norm() * p.Norm() * sin(Angle(p)) / 2;
  }
```

```
  LD Angle(Point3 p) {
    LD a = Norm();
    LD b = p.Norm();
    LD c = Dis(p);
    return acos((a * a + b * b - c * c) / (2 * a * b));
  }
  static LD Angle(Point3 p, Point3 q) {
    return p.Angle(q);
  }
  Point3 CrossProd(Point3 p) {
    Point3 q(*this);
    return {q[1] * p[2] - q[2] * p[1], q[2] * p[0] - q[0] *
        p[2], q[0] * p[1] - q[1] * p[0]};
  }
  static bool LexCmp(Point3& a, const Point3& b) {
    if (abs(a.x - b.x) > kEps) { return a.x < b.x; }
    if (abs(a.y - b.y) > kEps) { return a.y < b.y; }
    return a.z < b.z;
  }

  friend ostream& operator<<(ostream& out, Point3 m);
};

ostream& operator<<(ostream& out, Point3 p) {
  out << "(" << p.x << ", " << p.y << ", " << p.z << ")";
  return out;
}

struct Line3 {
  Point3 p[2];
  Point3& operator[](int a) { return p[a]; }
  friend ostream& operator<<(ostream& out, Line3 m);
};

ostream& operator<<(ostream& out, Line3 l) {
  out << l[0] << " - " << l[1];
  return out;
}

struct Plane {
  Point3 p[3];
  Point3& operator[](int a) { return p[a]; }
  Point3 GetNormal() {
    Point3 cross = (p[1] - p[0]).CrossProd(p[2] - p[0]);
    return cross.Normalize();
  }
  void GetPlaneEq(LD& A, LD& B, LD& C, LD& D) {
    Point3 normal = GetNormal();
    A = normal[0];
    B = normal[1];
    C = normal[2];

    D = normal.DotProd(p[0]);
    assert(abs(D - normal.DotProd(p[1])) < kEps);
    assert(abs(D - normal.DotProd(p[2])) < kEps);
  }
  vector<Point3> GetOrtonormalBase() {
    Point3 normal = GetNormal();
    Point3 cand = {-normal.y, normal.x, 0};
    if (abs(cand.x) < kEps && abs(cand.y) < kEps) {
      cand = {0, -normal.z, normal.y};
    }
    cand.NormalizeSelf();
    Point3 third = Plane{Point3{0, 0, 0}, normal,
        cand}.GetNormal();
    assert(abs(normal.DotProd(cand)) < kEps &&
        abs(normal.DotProd(third)) < kEps &&
        abs(cand.DotProd(third)) < kEps);
    return {normal, cand, third};
  }
```

```
};


struct Circle3 {
  Plane pl;
  Point3 cent;
  LD r;
  friend ostream& operator<<(ostream& out, Circle3 m);
};

ostream& operator<<(ostream& out, Circle3 c) {
  out << "pl: (" << c.pl[0] << ", " << c.pl[1] << ", " <<
      c.pl[2] << "), cent: " << c.cent << " r: " << c.r << "\n";
  return out;
}

struct Sphere {
  Point3 cent;
  LD r;
};

struct Utils3 {
  static bool Lines3Equal(Line3 p, Line3 l) {
    return Utils3::PtBelongToLine3(p[0], l) &&
        Utils3::PtBelongToLine3(p[1], l);
  }
  //angle PQR
  static LD Angle(Point3 P, Point3 Q, Point3 R) {
    return (P - Q).Angle(R - Q);
  }
  static Point3 ProjPtToLine3(Point3 p, Line3 l) { // ok
    Point3 diff = l[1] - l[0];
    diff.NormalizeSelf();
    return l[0] + diff * (p - l[0]).DotProd(diff);
  }
  static LD DisPtLine3(Point3 p, Line3 l) { // ok
//    LD area = Area(p, l[0], l[1]);
//     LD dis1 = 2 * area / l[0].Dis(l[1]);
    LD dis2 = p.Dis(ProjPtToLine3(p, l));
//     assert(abs(dis1 - dis2) < kEps);
    return dis2;
  }
  static LD DisPtPlane(Point3 p, Plane pl) {
    Point3 normal = pl.GetNormal();
    return abs(normal.DotProd(p - pl[0]));
  }
  static Point3 ProjPtToPlane(Point3 p, Plane pl) {
    Point3 normal = pl.GetNormal();
    return p - normal * normal.DotProd(p - pl[0]);
  }
  static bool PtBelongToPlane(Point3 p, Plane pl) {
    return DisPtPlane(p, pl) < kEps;
  }
  static Point PlanePtTo2D(Plane pl, Point3 p) { // ok
    assert(PtBelongToPlane(p, pl));
    vector<Point3> base = pl.GetOrtonormalBase();
    Point3 control{0, 0, 0};
    REP (tr, 3) {
      control += base[tr] * p.DotProd(base[tr]);
    }
    assert(PtBelongToPlane(pl[0] + base[1], pl));
    assert(PtBelongToPlane(pl[0] + base[2], pl));
    assert((p - control).IsZero());
    return {p.DotProd(base[1]), p.DotProd(base[2])};
  }
  static Line PlaneLineTo2D(Plane pl, Line3 l) {
    return {PlanePtTo2D(pl, l[0]), PlanePtTo2D(pl, l[1])};
  }
  static Point3 PlanePtTo3D(Plane pl, Point p) { // ok
    vector<Point3> base = pl.GetOrtonormalBase();
```

```
    return base[0] * base[0].DotProd(pl[0]) + base[1] * p.x +
        base[2] * p.y;
  }
  static Line3 PlaneLineTo3D(Plane pl, Line l) {
    return Line3{PlanePtTo3D(pl, l[0]), PlanePtTo3D(pl, l[1])};
  }


  static Line3 ProjLineToPlane(Line3 l, Plane pl) { // ok
    return Line3{ProjPtToPlane(l[0], pl), ProjPtToPlane(l[1],
        pl)};
  }
  static LD DisLineLine(Line3 l, Line3 k) { // ok
    Plane together {l[0], l[1], l[0] + k[1] - k[0]}; // parallel
        FIXME
    Line3 proj = ProjLineToPlane(k, together);
    Point3 inter = (Utils3::InterLineLine(l, proj))[0];
    Point3 on_k_inter = k[0] + inter - proj[0];
    return inter.Dis(on_k_inter);
  }
//
  static bool PtBelongToLine3(Point3 p, Line3 l) {
    return DisPtLine3(p, l) < kEps;
  }
  static bool Line3BelongToPlane(Line3 l, Plane pl) {
    return PtBelongToPlane(l[0], pl) && PtBelongToPlane(l[1],
        pl);
  }
  static LD Det(Point3 a, Point3 b, Point3 d) { // ok
    Point3 pts[3] = {a, b, d};
    LD res = 0;
    for (int sign : {-1, 1}) {
      REP (st_col, 3) {
        int c = st_col;
        LD prod = 1;
        REP (r, 3) {
          prod *= pts[r][c];
          c = (c + sign + 3) % 3;
        }
        res += sign * prod;
      }
    }
    return res;
  }
  static LD Area(Point3 p, Point3 q, Point3 r) { // ok
    q -= p;
    r -= p;
    return q.Area(r);
  }
  static Point3 PtFromSphericalAng(LD alpha, LD beta) { // ok
    return {cos(alpha) * cos(beta), cos(alpha) * sin(beta),
        sin(alpha)};
  }
  static vector<Point3> InterLineLine(Line3 k, Line3 l) {
    if (Lines3Equal(k, l)) { return {k[0], k[1]}; }
    if (PtBelongToLine3(l[0], k)) { return {l[0]}; }
    Plane pl{l[0], k[0], k[1]};
    if (!PtBelongToPlane(l[1], pl)) { return {}; }
    Line k2 = PlaneLineTo2D(pl, k);
    Line l2 = PlaneLineTo2D(pl, l);
    vector<Point> inter = Utils::InterLineLine(k2, l2);
    vector<Point3> res;
    for (auto P : inter) { res.PB(PlanePtTo3D(pl, P)); }
    return res;
  }

  static Plane ParallelPlane(Plane pl, Point3 A) { // plane
      parallel to pl going through A
    Point3 diff = A - ProjPtToPlane(A, pl);
    return Plane{pl[0] + diff, pl[1] + diff, pl[2] + diff};
```

```
  }

  // image of B in rotation wrt line passing through origin s.t.
     A1->A2
  // implemented in more general case with similarity instead of
     rotation
  static Point3 RotateAccordingly(Point3 A1, Point3 A2, Point3
     B1) { // ok
    Plane pl{A1, A2, {0, 0, 0}};
    Point A12 = PlanePtTo2D(pl, A1);
    Point A22 = PlanePtTo2D(pl, A2);
    complex<LD> rat = complex<LD>(A22.x, A22.y) /
       complex<LD>(A12.x, A12.y);
    Plane plb = ParallelPlane(pl, B1);
    Point B2 = PlanePtTo2D(plb, B1);
    complex<LD> Brot = rat * complex<LD>(B2.x, B2.y);
    return PlanePtTo3D(plb, {Brot.real(), Brot.imag()});
  }

//  static vector<Point3> InterCoplanarCircleCircle(Circle c1,
   Circle c2) {
//    //assert(c1.pl == c2.pl);
//
//  }

  static vector<Circle3> InterSpherePlane(Sphere s, Plane pl) {
     // ok
    Point3 proj = ProjPtToPlane(s.cent, pl);
    LD dis = s.cent.Dis(proj);
    if (dis > s.r + kEps) {
      //return {{{}, {}, {}}, {}, -1};
      return {};
    }
    if (dis > s.r - kEps) {
      //return {{{}, {}, {}}, proj, 0};
      return {{pl, proj, 0}}; // is it best choice?
    }
    return {{pl, proj, sqrt(s.r * s.r - dis * dis)}};
  }

  static bool PtBelongToSphere(Sphere s, Point3 p) {
    return abs(s.r - s.cent.Dis(p)) < kEps;
  }

  static LD DisOnSphere(Sphere sph, Point3 A, Point3 B) {
    assert(PtBelongToSphere(sph, A));
    assert(PtBelongToSphere(sph, B));
    LD ang = Angle(A, sph.cent, B);
    return ang * sph.r;
  }
};

struct PointS {
  LD lat, lon;
  PointS(LD latt, LD lonn) { lat = latt; lon = lonn; }
  Point3 toEucl() {
    return Point3{cos(lat) * cos(lon), cos(lat) * sin(lon),
       sin(lat)};
  }
  PointS(Point3 p) {
    p.NormalizeSelf();
    lat = asin(p.z);
    lon = acos(p.y / cos(lat));
  }
};

LD DistS(Point3 a, Point3 b) {
       return atan2l(b.CrossProd(a).Norm(), a.DotProd(b));
}
```

```
struct CircleS {
        Point3 o; // center of circle on sphere
        LD r; // arc len
        LD area() const { return 2 * kPi * (1 - cos(r)); }
};

CircleS From3(Point3 a, Point3 b, Point3 c) { // any three
    different points
        int tmp = 1;
        if ((a - b).Norm() > (c - b).Norm()) { swap(a, c); tmp =
            -tmp; }
        if ((b - c).Norm() > (a - c).Norm()) { swap(a, b); tmp =
            -tmp; }
        Point3 v = (c - b).CrossProd(b - a); v = v * (tmp /
            v.Norm());
        return CircleS{v, DistS(a,v)};
}

CircleS From2(Point3 a, Point3 b) { // neither the same nor the
    opposite
        Point3 mid = (a + b) / 2;
        mid = mid / mid.Norm();
        return From3(a, mid, b);
}

LD Angle(Point3 A, Point3 B, Point3 C) { //angle at A, no two
    points opposite
        LD a = B.DotProd(C);
        LD b = C.DotProd(A);
        LD c = A.DotProd(A);
        return acos((b - a * c) / sqrt((1 - Sq(a)) * (1 -
            Sq(c))));
}
```

```
LD TriangleArea(Point3 A, Point3 B, Point3 C) { // no two poins
    opposite
        LD a = Angle(C, A, B);
        LD b = Angle(A, B, C);
        LD c = Angle(B, C, A);
        return a + b + c - kPi;
}

vector<Point3> IntersectionS(CircleS c1, CircleS c2) { // what
    about c1==c2 case?
        Point3 n = c2.o.CrossProd(c1.o), w = c2.o * cos(c1.r) -
            c1.o * cos(c2.r);
        LD d = n.SqNorm();
  if (d < kEps) {
    cerr<<"parallel circles?\n";
    return {};
  }
  LD a = w.SqNorm() / d;
        vector<Point3> res;
        if (a >= 1 + kEps) { return res; }
        Point3 u = n.CrossProd(w) / d;
        if (a > 1 - kEps) {
    res.PB(u);
          return res;
        }
        LD h = sqrt((1 - a) / d);
        res.PB(u + n * h);
        res.PB(u - n * h);
        return res;
}

bool PtBelongToSeg(Point3 A, Point3 B, Point3 C) { // A belong
    to BC
  return abs(DistS(A, B) + DistS(A, C) - DistS(B, C)) < kEps;
}
```

```
const int kInf = 1e9;
const int kEarthRad = 6370;
typedef pair<LD, LD> PLD;
struct Sol {
       void Test(int cnt) {
    int air_num;
    LD rad;
    if (!(cin>>air_num>>rad)) { exit(0); }
    cout<<"Case "<<cnt<<":\n";
    rad /= kEarthRad;
    vector<Point3> airports(air_num + 2);
    vector<CircleS> caps(air_num + 2);
    vector<Point3> interesting(air_num + 1);
    RE (i, air_num) {
      LD lon, lat;
      cin>>lon>>lat;
      lon *= kPi / 180;
      lat *= kPi / 180;
      airports[i] = (PointS{(LD)lat, (LD)lon}).toEucl();
      interesting[i] = airports[i];
      caps[i] = CircleS{airports[i], rad};
    }
    RE (i, air_num) {
      RE (j, i - 1) {
        vector<Point3> inters = IntersectionS(caps[i], caps[j]);
        debug(caps[i].o, caps[j].o);
        for (auto p : inters) {
          interesting.PB(p);
          debug(interesting.back());
        }
      }
    }
    int all = SZ(interesting) - 1;
    vector<vector<LD>> tru_dis(all + 2, vector<LD>(all + 2));
    RE (i, all) {
      RE (j, all) {
        if (i != j) { tru_dis[i][j] = kInf; }
      }
    }
    debug(all);
    RE (i, all) {
      RE (j, i - 1) {
        if ((interesting[i] + interesting[j]).IsZero() ||
            (interesting[i] - interesting[j]).IsZero()) {
          continue; }
        CircleS seg = From2(interesting[i], interesting[j]);

        vector<PLD> good_intervals;
        bool good_whole = false;
        LD seg_len = DistS(interesting[i], interesting[j]);
        RE (c, air_num) {
          vector<Point3> inters = IntersectionS(seg, caps[c]);
          if (SZ(inters) < 2) { continue; }
          bool bel0 = PtBelongToSeg(inters[0], interesting[i],
              interesting[j]);
          bool bel1 = PtBelongToSeg(inters[1], interesting[i],
              interesting[j]);
          if (!bel0 && !bel1) {
            if (DistS(interesting[i], airports[c]) < rad + kEps) {
              good_whole = true;
              break;
            }
          } else if (bel0 && bel1) {
            LD d0 = DistS(inters[0], interesting[i]);
            LD d1 = DistS(inters[1], interesting[i]);
            good_intervals.PB({min(d0, d1), max(d0, d1)});
          } else {
            if (bel1) {
              swap(inters[0], inters[1]);
            }
```

```
      LD d0 = DistS(inters[0], interesting[i]);
      LD aidis = DistS(interesting[i], airports[c]);
      if (aidis < rad - kEps || (abs(aidis - rad) < kEps &&
          DistS(interesting[j], airports[c]) > rad + kEps)) {
        good_intervals.PB({0, d0});
      } else {
        good_intervals.PB({d0, seg_len});
      }
    }
  }
  if (!good_whole) {
    if (!good_intervals.empty()) {
      debug(j, i, good_intervals, seg_len);
      good_intervals.PB({0, 0});
      good_intervals.PB({seg_len, seg_len});
      sort(ALL(good_intervals));
      LD rightmost = good_intervals[0].nd;
      bool fail = false;
      for (auto interval : good_intervals) {
        if (interval.st > rightmost + kEps) {
          fail = true;
        }
        maxi(rightmost, interval.nd);
      }
      if (!fail) {
        good_whole = true;
      }
    }
  }
  if (good_whole) {
    debug(i, j);
    tru_dis[i][j] = tru_dis[j][i] = seg_len;
  }
 }
}
```

```
RE (k, all) {
 RE (i, all) {
  RE (j, all) {
   mini(tru_dis[i][j], tru_dis[i][k] + tru_dis[k][j]);
  }
 }
}
debug(tru_dis);
int q;
cin>>q;
RE (_, q) {
 vector<vector<LD>> air_dis(air_num + 2, vector<LD>(air_num
     + 2));
 int s, t;
 LD c;
 cin>>s>>t>>c;
 c /= kEarthRad;
 debug(c);
 RE (i, air_num) {
  RE (j, air_num) {
   if (i == j) { continue; }
   if (tru_dis[i][j] < c + kEps) {
    air_dis[i][j] = tru_dis[i][j];
   } else {
    air_dis[i][j] = kInf;
   }
  }
 }
 RE (k, air_num) {
  RE (i, air_num) {
   RE (j, air_num) {
    mini(air_dis[i][j], air_dis[i][k] + air_dis[k][j]);
   }
  }
 }
```

```
      if (air_dis[s][t] > kInf / 2) {
        cout<<"impossible\n";
      } else {
        cout<<air_dis[s][t] * kEarthRad<<"\n";
      }
    }


  }

};

#undef int
int main() {
#define int long long

  ios_base::sync_with_stdio(0);
  cout << fixed << setprecision(3);
  cerr << fixed << setprecision(3);
  cin.tie(0);
  //double beg_clock = 1.0 * clock() / CLOCKS_PER_SEC;

  int cnt = 0;
  while (1) {
    cnt++;
    Sol sol;
    sol.Test(cnt);
  }

  return 0;
}
```

2.10 integration gaussian quadrature and spline interpolation

```
/**
 * Problem: dog (aka Parabellum).
 * Correct solution (must be OK).
 * Author: stgatilov
 *
 * Incrementally integrates time over angle with 3-node Guass
    quadrature (saving time values)
 * Time for angle is interpolated with cubic spline (Hermite).
 * After period reduction, angle from time is determined with
    binary search.
 *
 * OK on current tests with maxError = 1e-7 in checker.
 * Time: O(N + N log(eps))   N - number of nodes
 */

//#pragma comment(linker, "/STACK:20000000")
#include <vector>
#include <list>
#include <map>
#include <set>
#include <deque>
#include <stack>
#include <bitset>
#include <algorithm>
#include <functional>
#include <numeric>
#include <utility>
#include <sstream>
#include <iostream>
#include <iomanip>
#include <cstdio>
#include <cstring>
```

```
#include <cassert>
#include <cstdlib>
#include <ctime>
#include <cstdint>
#include <unordered_set>
#include <cinttypes>
#include <climits>

#define _USE_MATH_DEFINES
#include <math.h>

using namespace std;
typedef long long int64;
#ifdef HOME
    #define E(c) cerr<<#c
    #define Eo(x) cerr<<#x<<" = "<<(x)<<endl
    #define Ef(...) fprintf(stderr, __VA_ARGS__)
#else
    #define E(c) ((void)0)
    #define Eo(x) ((void)0)
    #define Ef(...) ((void)0)
#endif


struct Point {
    double x, y;

    Point() : x(0), y(0) {}
    Point(double _x, double _y) : x(_x), y(_y) {}

    Point operator- (const Point &b) {
        return Point(x - b.x, y - b.y);
    }
    Point operator+ (const Point &b) {
        return Point(x + b.x, y + b.y);
    }
    Point operator* (double coef) {
        return Point(x * coef, y * coef);
    }
};

double dot(const Point &a, const Point &b) {
    return a.x * b.x + a.y * b.y;
}
double cross(const Point &a, const Point &b) {
    return a.x * b.y - a.y * b.x;
}
double len(const Point &a) {
    return sqrt(dot(a, a));
}
Point polar(double ang) {
    return Point(cos(ang), sin(ang));
}

Point ostapPos, kislaPos;
Point ostapVel;
double kislaSpeed;
double radius, startAngle;

double pi = acos(-1.0);

double CalcAngularSpeed(double deltaAngle) {
    double polarAngle = deltaAngle + (pi/2.0+ startAngle);
    Point dir = polar(polarAngle);

    double qb2 = dot(ostapVel, dir);
    double qc = dot(ostapVel, ostapVel) - (kislaSpeed *
        kislaSpeed);
    double qd2 = qb2*qb2 - qc;
    assert(qd2 > 0.0);
```

```
    double sol = sqrt(qd2) - qb2;

    double angSp = sol / radius;
    return angSp;
}
double CalcDtDa(double deltaAngle) {
    return 1.0 / CalcAngularSpeed(deltaAngle);
}

const int NODES = 100<<10;
double timeVal[NODES + 1], timeDer[NODES + 1];

//const int GAUSS_CNT = 3;
//const double GAUSS_NODES[] = {-sqrt(3.0/5.0), 0.0,
    sqrt(3.0/5.0)};
//const double GAUSS_WEIGHTS[] = {5.0/9.0, 8.0/9.0, 5.0/9.0};
//

const int GAUSS_CNT = 4;
const double GAUSS_NODES[] = {-sqrt(3.0/7.0 +
    (2.0/7.0)*sqrt(6.0/5.0)),-sqrt(3.0/7.0 -
    (2.0/7.0)*sqrt(6.0/5.0)), sqrt(3.0/7.0 -
    (2.0/7.0)*sqrt(6.0/5.0)),sqrt(3.0/7.0 +
    (2.0/7.0)*sqrt(6.0/5.0))};
const double GAUSS_WEIGHTS[] =
    {(18.0-sqrt(30.0))/36.0,(18.0+sqrt(30.0))/36.0,(18.0+sqrt(30.0))/36.0,(18.0-sqrt(30.0))/36.0};

const double STEP = 2*pi / NODES;
void IntegrateTime() {
    timeVal[0] = 0;
    timeDer[0] = CalcDtDa(0);
    for (int i = 0; i < NODES; i++) {
        double integ = 0.0;
        for (int j = 0; j < GAUSS_CNT; j++) {
            double node = STEP * (i + (GAUSS_NODES[j] + 1.0) *
                0.5);
            double wgt = GAUSS_WEIGHTS[j] * 0.5;
            integ += wgt * CalcDtDa(node);
        }
        timeVal[i+1] = timeVal[i] + integ * STEP;
        timeDer[i+1] = CalcDtDa(STEP * (i+1));
    }
}

double InterpolateTime(double deltaAngle) {
    double param = deltaAngle * (1.0 / STEP);
    int cell = int(param);
    cell = max(cell, 0);
    cell = min(cell, NODES - 1);
    double t = param - cell;

    double ctrl[4][4];
    ctrl[0][0] = timeVal[cell+0];
    ctrl[0][3] = timeVal[cell+1];
    ctrl[0][1] = ctrl[0][0] + timeDer[cell+0] * STEP / 3.0;
    ctrl[0][2] = ctrl[0][3] - timeDer[cell+1] * STEP / 3.0;
    for (int i = 1; i <= 3; i++)
        for (int j = 0; j <= 3-i; j++)
            ctrl[i][j] = ctrl[i-1][j] * (1-t) + ctrl[i-1][j+1] *
                t;
    double interp = ctrl[3][0];

    return interp;
}

int q;
vector<double> timeQueries;

int main(int argc, char **argv) {
```

```
//    freopen("input.txt", "r", stdin);
//    freopen("output.txt", "w", stdout);
    ios::sync_with_stdio(0);
    cin.tie(0);
    cout << setprecision(10) << fixed;
    cin >> ostapPos.x >> ostapPos.y;
    cin >> kislaPos.x >> kislaPos.y;
    cin >> ostapVel.x >> ostapVel.y;
    cin >> kislaSpeed;
    cin >> q;
    for (int i = 0; i < q; i++) {
        double t;
        cin >> t;
        timeQueries.push_back(t);
    }

    Point startVec = kislaPos - ostapPos;
    radius = len(startVec);
    startAngle = atan2(startVec.y, startVec.x);

    IntegrateTime();
    double period = timeVal[NODES];
    Eo(period);

    for (int i = 0; i < q; i++) {
        double qt = timeQueries[i];
        int laps = int(qt / period);
        double lapRem = qt - laps * period;

        double left = 0.0;
        double right = 2 * pi;
        static const int TIMES = 60;
        for (int z = 0; z < TIMES; z++) {
            double middle = (left + right) / 2;
            if (middle == left || middle == right)
                break;
            if (InterpolateTime(middle) < lapRem)
                left = middle;
            else
                right = middle;
        }

        double polarAngle = startAngle + (left + right) * 0.5;
        Point ostapAt = ostapPos + ostapVel * qt;
        Point kislaAt = ostapAt + polar(polarAngle) * radius;
        cout << kislaAt.x << " " << kislaAt.y << "\n";
    }
    return 0;
}
```

2.11 max common area of two convex objects

```
#include <bits/stdc++.h>
using namespace std;

typedef long long ll;

struct Lattice{
    ll x , y ;
    Lattice(ll x_,ll y_) {x=x_,y=y_;}
    Lattice() {}

    Lattice operator + (const Lattice &p) {
        return {x+p.x,y+p.y} ;
    }
    Lattice operator - (const Lattice &p) {
        return {x-p.x,y-p.y} ;
    }
    ll operator * (const Lattice &p) {
```

```
        return x*p.x+y*p.y ;
    }
    Lattice operator * (const ll t) {
        return Lattice(t*x,t*y);
    }
    Lattice operator / (const ll t) {
        return Lattice(x/t,y/t);
    }
    ll operator ^ (const Lattice &p) {
        return x*p.y-y*p.x ;
    }
    bool operator < (const Lattice &p) const {
        return make_pair(x,y) < make_pair(p.x,p.y) ;
    }
    bool operator > (const Lattice &p) const {
        return make_pair(x,y) > make_pair(p.x,p.y) ;
    }
    Lattice rot90() {
        return Lattice(-y,x) ;
    }
};

bool onSegment(Lattice p,Lattice q,Lattice a) {
    return ( (p-q)^(q-a) ) == 0 and ((p-a)*(q-a)) <= 0 ;
}

int sign (ll x) {
    if (x < 0) return -1;
    return x > 0;
}

bool intersect(Lattice a,Lattice b,Lattice p,Lattice q) {
    if ( (sign( ((b-a)^(p-a)) ) * sign( ((b-a)^(q-a)) )) == -1
        and (sign( ((q-p)^(a-p)) ) * sign( ((q-p)^(b-p)))) == -1)
        {
        return 1;
    }
    if (onSegment(a,b,p)) return 1;
    if (onSegment(a,b,q)) return 1;
    if (onSegment(p,q,a)) return 1;
    if (onSegment(p,q,b)) return 1;
    return 0;
}

bool intersect(vector<Lattice> &A,vector<Lattice> &B,Lattice
    dir) {
    int n = A.size() , m = B.size();
    for(int i = 0; i < n; i++) {
        Lattice p = A[i];
        Lattice q = p + dir*100000 ;
        for(int j = 0; j < m; j++) {
            Lattice a = B[j] , b = B[(j+1)%m];
            if (intersect(a,b,p,q)) {
                return 1;
            }
        }
    }
    return 0;
}


const double eps = 1e-9;


int dcmp(double x) {
    if (fabs(x) < eps) return 0;
    if (x < 0.0) return -1;
    return 1;
}
```

```
struct Point {
    double x , y ;
    Point() {}
    Point(double x_,double y_) {x=x_,y=y_;}
    Point operator + (const Point &p) {
        return {x+p.x,y+p.y} ;
    }
    Point operator - (const Point &p) {
        return {x-p.x,y-p.y} ;
    }
    Point operator * (const double &t) {
        return {x*t,y*t} ;
    }
    double operator * (const Point &p) {
        return x*p.x + y*p.y ;
    }
    double operator ^ (const Point &p) {
        return x*p.y - y*p.x ;
    }
};

bool ccw (Lattice p,Lattice q,Lattice r) {
    return ( (q-p)^(r-p) ) > 0;
}

void ConvexHull(vector<Lattice> &poly) {
    sort(poly.begin(),poly.end()) ;
    vector<Lattice> hull,lower,upper ;
    for(int i = 0 ; i < poly.size() ; i++) {
        while(lower.size() >= 2 and
            !ccw(lower[lower.size()-2],lower.back(),poly[i])) {
            lower.pop_back() ;
        }
        lower.push_back(poly[i]) ;
    }
    for(int i = poly.size()-1 ; i >= 0 ; i--) {
        while(upper.size() >= 2 and
            !ccw(upper[upper.size()-2],upper.back(),poly[i])) {
            upper.pop_back() ;
        }
        upper.push_back(poly[i]) ;
    }
    hull = lower ;
    for(int i = 1 ; i + 1 < upper.size() ; i++)
        hull.push_back(upper[i]) ;
    poly = hull ;
}

double Abs(Point p) {
    return sqrt(p*p);
}

const double magic = 100000.0;

bool onSegment(Point p,Point q,Point a) {
    return dcmp((p-q)^(q-a)) == 0 and dcmp( ((p-a)*(q-a)) ) <= 0
        ;
}

void intersect(Point a,Point b,Point p,Point q,vector<double>
    &res) {
    Point dir = (b-a)*(1.0/magic) ;
    if ( (dcmp( ((b-a)^(p-a)) ) * dcmp( ((b-a)^(q-a)) )) == -1
        and (dcmp( ((q-p)^(a-p)) ) * dcmp( ((q-p)^(b-p)) )) == -1)
        {
        double na = (q-p)^(a-p) , nb = (q-p)^(b-p);
        double t = magic*na/(na-nb) ;
        if (t > 0.0) res.push_back(t);
    }
    else {
```

```
        vector<Point> ends;
        if (onSegment(a,b,p)) ends.push_back(p);
        if (onSegment(a,b,q)) ends.push_back(q);
        if (onSegment(p,q,a)) ends.push_back(a);
        if (onSegment(p,q,b)) ends.push_back(b);

        for(int i = 0 ; i < ends.size() ; i++) {
            Point tp = ends[i];
            if (dcmp(dir.x) != 0) {
                double t = (tp.x-a.x)/dir.x ;
                if (t > 0.0) res.push_back(t);
            }
            else {
                double t = (tp.y-a.y)/dir.y ;
                if (t > 0.0)res.push_back(t);
            }
        }
    }
}

void solve(vector<double> &events,vector<Point> &A,vector<Point>
    &B,Point dir) {
    int n = A.size() , m = B.size() ;
    double vel = Abs(dir);
    for(int i = 0 ; i < n ; i++) {
        Point a = A[i] , b = a + dir*magic ;
        for(int j = 0 ; j < m ; j++) {
            Point p = B[j] , q = B[(j+1)%m];
            intersect(a,b,p,q,events);
        }
    }
}

void intersect(vector<Point> &V,Point a,Point b,Point p,Point q)
    {
    double na = (a-p)^(q-p) , nb = (b-p)^(q-p) ;
    if (na*nb < 0.0) {
        V.push_back(a + (b-a)*(na/(na-nb))) ;
    }
}


void cut(vector<Point> &polygon, Point a , Point b) {
    vector<Point> np ;
    int sz = polygon.size();
    for(int i = 0 ; i < sz ; i++) {
        Point p = polygon[i] , q = polygon[(i+1)%sz];
        if (dcmp((b-a)^(p-a)) >= 0) {
            np.push_back(p);
        }
        intersect(np,p,q,a,b);
    }
    polygon = np ;
}

double calc(vector <Point> &A,vector<Point> &B) {
    int m = B.size();
    for(int i = 0 ; i < m ; i++) {
        Point p = B[i] , q = B[(i+1)%m];
        cut(A,p,q);
    }
    if (A.size() == 0) return -1e9;
    if (A.size() < 3) return 0;
    double area = 0.0 ;
    int n = A.size() ;
    for(int i = 0 ; i < n; i++) {
        area += (A[i]^A[(i+1)%n]);
    }
    return 0.5*area;
}
```

```
double area(vector<Point> A , vector<Point> &B,Point dir) {
    for(int i = 0 ; i < A.size() ; i++) A[i] = A[i] + dir ;
    return calc(A,B);
}

int main() {

    //freopen ("in.txt" , "r" , stdin);
    int n ;
    while(cin >> n) {
        vector<Lattice> A(n);
        for(int i = 0; i < n; i++) {
            cin >> A[i].x >> A[i].y;
        }
        Lattice da;
        cin >> da.x >> da.y;
        int m; cin >> m;
        vector<Lattice> B(m);
        for(int i = 0; i < m; i++) cin >> B[i].x >> B[i].y;
        Lattice db;
        cin >> db.x >> db.y;
        if (da.x == db.x and da.y == db.y) {
            cout << "never" << endl;
            continue ;
        }
        ConvexHull(A); n = A.size();
        ConvexHull(B); m = B.size();
        if( intersect(A,B,da-db) or intersect(B,A,db-da) ) {
            vector<Point> C(n), D(m);
            for(int i = 0 ; i < n ; i++) C[i].x = A[i].x , C[i].y
                = A[i].y ;
            for(int i = 0 ; i < m ; i++) D[i].x = B[i].x , D[i].y
                = B[i].y ;
            Point dirA = Point(da.x,da.y);
            Point dirB = Point(db.x,db.y);
            vector <double> events;
            solve(events,C,D,(dirA-dirB));
            solve(events,D,C,(dirB-dirA));
            events.push_back(0.0) ;
            events.push_back(magic) ;
            sort(events.begin(),events.end()) ;
            Point dir = dirA-dirB;
            double ans = -1e8 , Time = 0.0 ;
            for(int i = 1 ; i < events.size() ; i++) {
                double lo = events[i-1] , hi = events[i] ;
                for(int it = 1 ; it <= 50 ; it++) {
                    double m1 = (2.0*lo + hi)/3.0 , m2 = (lo +
                        2.0*hi)/3.0;
                    double a1 = area(C,D,dir*m1) , a2 =
                        area(C,D,dir*m2) ;
                    if (dcmp(a1-a2) == 0) {
                        hi = m2 ;
                    }
                    else if (a1 > a2) {
                        hi = m2 ;
                    }
                    else {
                        lo = m1 ;
                    }
                }
                double cur = area(C,D,dir*lo) ;
                if (dcmp(cur-ans) == 1) {
                    ans = cur;
                    Time = lo;
                }
            }
            cout << setprecision(12) << Time << endl ;
        }
```

```
        else {
            cout << "never" << endl;
        }
    }
    return 0;
}
```

2.12 minimum enclosing circle 3dconvexhull

```
#include <iostream>
#include <ctime>
#include <fstream>
#include <cmath>
#include <cstring>
#include <cassert>
#include <cstdio>
#include <algorithm>
#include <iomanip>
#include <vector>
#include <stack>
#include <queue>
#include <set>
#include <map>
#include <complex>
#include <utility>
#include <cctype>
#include <list>
#include <bitset>
#include <unordered_set>
#include <unordered_map>

using namespace std;

#define FORALL(i,a,b) for(int i=(a);i<=(b);++i)
#define FOR(i,n) for(int i=0;i<(n);++i)
#define FORB(i,a,b) for(int i=(a);i>=(b);--i)

typedef long long ll;
typedef long double ld;

typedef pair<ll,int> plli;
typedef pair<int,int> pii;
typedef map<int,int> mii;

#define pb push_back
#define mp make_pair

#define EPS (1e-8)
#define MAXN 1005
#define sign(x) (((x)>EPS)-((x)<(-EPS)))

const ld PI = atan2(0,-1);

// 3d vector (can degenerate to 2d when z=0)
#define T ld
struct vec {
  T x,y,z;   //coordinates/data
  vec(T xx, T yy, T zz=0.){ x=xx;y=yy;z=zz; }
  vec() { x=y=z=0;}

  // vector ops
  vec& operator=(const vec& b) { x=b.x; y=b.y; z=b.z; return
      *this; }
  vec operator+(const vec& b) const { return vec(x+b.x, y+b.y,
      z+b.z); }
  vec operator-(const vec& b) const { return vec(x-b.x, y-b.y,
      z-b.z); }
  T operator*(const vec& b) const { return x*b.x + y*b.y +
      z*b.z; }
```

```
  vec operator^(const vec& b) const { return vec(y*b.z - z*b.y,
                                      z*b.x - x*b.z,
                                      x*b.y - y*b.x); }
  // scalar mult
  vec operator*(T k) const { return vec(x*k,y*k,z*k); }
  vec operator/(T k) const { return vec(x/k,y/k,z/k); }
  vec operator-() const { return vec(-x,-y,-z); } // negation

  T sqlen() const { return (*this) * (*this); }

  bool operator<(const vec& other) const {
    if (x < other.x) return true;
    if (x > other.x) return false;
    if (y < other.y) return true;
    if (y > other.y) return false;
    if (z < other.z) return true;
    if (z > other.z) return false;
    return false;
  }
};
vec operator*(T k, vec v) { return v*k; }
ostream& operator<<(ostream& out, const vec& v) {
  return out << "(" << v.x << "," << v.y << "," << v.z <<")";
}
#undef T

#define INSIDE (-1)
#define ON (0)
#define OUTSIDE (1)

typedef vector<vec> edge;
typedef vector<vec> face;
typedef vector<face> hull;

bool eq(ld a, ld b) {
  return abs(b-a) <= EPS;
}

ld len(vec a) {
  return sqrtl(a.sqlen());
}

int side(vec a, vec b, vec c, vec x) {
  vec norm = (b-a) ^ (c-a);
  vec me = x-a;
  return sign(me * norm);
}

bool is_colinear(vec a, vec b, vec c) {
  vec u = b-a, v = c-a;
  vec w = u^v;
  return eq(w.sqlen(),0);
}

vec projection(vec a, vec b, vec c, vec x) {
  if (side(a,b,c,x) == ON) return x;
  vec norm = (b-a) ^ (c-a);
  vec ans = x - norm * ((norm * (x-a)) / (norm * norm));
  assert(side(a,b,c,ans) == ON);
  return ans;
}

/// O(n^2log(n))

hull find_hull(vec* P, int N) {
  random_shuffle(P, P+N);

  // Find 4 non-degenerate points (make a tetrahedron)
  FORALL(j,2,N-1) if (!is_colinear(P[0],P[1],P[j])) { swap(P[j],
      P[2]); break; }
```

```
FORALL(j,3,N-1) if (side(P[0],P[1],P[2],P[j]) != 0) {
    swap(P[j], P[3]); break; }

// Canonicalize them
if (side(P[0],P[1],P[2],P[3]) == OUTSIDE) swap(P[0], P[1]);
assert(side(P[0],P[1],P[2],P[3]) == INSIDE);
assert(side(P[0],P[3],P[1],P[2]) == INSIDE);
assert(side(P[0],P[2],P[3],P[1]) == INSIDE);
assert(side(P[3],P[2],P[1],P[0]) == INSIDE);

hull H{
  {P[0],P[1],P[2]},
  {P[0],P[3],P[1]},
  {P[0],P[2],P[3]},
  {P[3],P[2],P[1]}
};

auto make_degrees = [&](const hull& H) {
  map<edge,int> ans;
  for (const auto & f : H) {
    assert(f.size() == 3);
    FOR(i,3) {
      vec a = f[i];
      vec b = f[(i+1)%3];
      ans[{a,b}]++;
    }
  }

  return ans;
};

// incrementally add points
FORALL(j,4,N-1) {
  hull H2; H2.reserve(H.size());
  vector<face> plane;
  for (const auto & f : H) {
    int s = side(f[0],f[1],f[2],P[j]);
    if (s == INSIDE || s == ON) H2.pb(f);
  }

  // For any edge that now only has 1 incident face (it's
     other face deleted)
  // add a new face with this vertex and that edge.
  map<edge, int> D = make_degrees(H2);
  const auto tmp = H2;
  for (const auto & f : tmp) {
    assert(f.size() == 3);
    FOR(i,3) {
      vec a = f[i];
      vec b = f[(i+1)%3];
      int d = D[{a,b}] + D[{b,a}];
      assert(d == 1 || d == 2);
      if (d==1) {
        // add a new face
        H2.pb({a, P[j], b});
      }
    }
  }

  H = H2;
}

// sanity check that this is at least mostly a hull :)
for (const auto & f : H) {
  FOR(i,N) {
    int s = side(f[0],f[1],f[2],P[i]);
    assert(s == INSIDE || s == ON);
  }
}
```

```
  // sanity check that this figure is closed
  map<edge, int> D = make_degrees(H);
  for (const auto & f : H) {
    assert(f.size() == 3);
    FOR(i,3) {
      vec a = f[i];
      vec b = f[(i+1)%3];
      int d = D[{a,b}] + D[{b,a}];
      assert(d == 2);
    }
  }

  return H;
}

// line stuff
bool on (vec a, vec b, vec x) {
  return eq(len(x-a) + len(x-b), len(a-b));
}

// find the intersection point of ab with cd
vec isect(vec a, vec b, vec c, vec d) {
  vec u = (b-a), v = (d-c), z = (c-a);
  vec vz = v^z, vu = v^u;
  ld s = len(vz) / len(vu) * sign(vz*vu);
  return a + u*s;
}

typedef pair<vec, ld> circle_t;

bool in_circle(const vec& v, const circle_t& C) {
  return len(v - C.first) <= C.second + EPS;
}

circle_t better(circle_t A, circle_t B) {
  if (A.second < B.second) return A;
  return B;
}

circle_t find_circle(vec a) { return {a, 0}; }
circle_t find_circle(vec a, vec b) { return { (a+b)/2, len(a-b)
    / 2 }; }
circle_t find_circle(vec a, vec b, vec c, bool force_on = false)
    {
  vec u = (b-a), v = (c-a);
  vec norm = u ^ v;
  vec uperp = u^norm, vperp = v^norm;
  vec ab = (a+b)/2, ac = (a+c)/2;

  if (is_colinear(a,b,c)) {
    if (on(a,b,c)) return { (a+b)/2, len(a-b) / 2 };
    if (on(a,c,b)) return { (a+c)/2, len(a-c) / 2 };
    if (on(c,b,a)) return { (c+b)/2, len(c-b) / 2 };
    assert(false);
  }

  vec ans = isect(ab, ab + uperp, ac, ac + vperp);
  assert(eq(len(ans-a), len(ans-b)));
  assert(eq(len(ans-a), len(ans-c)));

  circle_t C = { ans, (len(ans-a) + len(ans-b) + len(ans-c)) /
     3.01 };
  assert(in_circle(a, C) && eq(len(ans-a), C.second));
  assert(in_circle(b, C) && eq(len(ans-b), C.second));
  assert(in_circle(c, C) && eq(len(ans-c), C.second));

  if (force_on) return C;

  circle_t C_ab = find_circle(a,b);
```

```
  circle_t C_bc = find_circle(b,c);
  circle_t C_ac = find_circle(a,c);

  if (in_circle(c, C_ab)) C = better(C, C_ab);
  if (in_circle(a, C_bc)) C = better(C, C_bc);
  if (in_circle(b, C_ac)) C = better(C, C_ac);

  assert(in_circle(a,C));
  assert(in_circle(b,C));
  assert(in_circle(c,C));

  return C;
}

/// minimum enclosing circle- expected runtime- O(N)

// Find circle of N points. K of them (the last K) are guaranteed
// to be on the boundary.
circle_t find_circle(vec* P, int N, int K) {
  if (K >= 3) {
    assert(K == 3);
    assert(!is_colinear(P[N-1],P[N-2],P[N-3]));
    assert(side(P[N-1],P[N-2],P[N-3],P[0]) == ON);
    auto C = find_circle(P[N-1],P[N-2],P[N-3],true);
    return C;
  }

  if (N == 1) return find_circle(P[0]);
  if (N == 2) return find_circle(P[0],P[1]);
  assert(K < N);

  // pick a random point, remove it, recurse.
  // with very high probability, that recursed circle is the
     optimal circle
  // if not, we just try again (and this point is added to the K
     set)
  int i = rand()%(N-K);

  swap(P[i], P[N-1-K]); swap(P[N-1-K], P[N-1]); // hack: avoid
     deleting back K
  auto C = find_circle(P, N-1, K);
  swap(P[N-1-K], P[N-1]); swap(P[i], P[N-1-K]);

  if (in_circle(P[i],C)) return C;

  // Didn't work, that's fine. Add it to our K-set ("boundary
     set") and try again
  swap(P[i], P[N-1-K]);
  C = find_circle(P, N, K+1);
  swap(P[i], P[N-1-K]);

  return C;
}

vec P[MAXN];
vec F[MAXN];
vec tmp[MAXN];
int main() {
  int N,x,y,z;
  scanf("%d",&N);
  FOR(i,N) {
    scanf("%d%d%d",&x,&y,&z);
    P[i] = vec(x,y,z);
  }

  // Find the convex hull
  hull H = find_hull(P,N);
```

```
  // Now try all cylinders with bases aligned with faces of the
     hull
  ld ans = 1000000000000000.;
  for (const auto & f : H) {
    assert(f.size() == 3);
    vec norm = (f[1] - f[0]) ^ (f[2] - f[0]);

    ld height = 0;
    FOR(i,N) height = max(height, abs((P[i] - f[0])*norm));
    height /= len(norm);

    FOR(i,N) F[i] = projection(f[0],f[1],f[2],P[i]);
    auto C = find_circle(F,N,0);
    ld r = C.second;

    FOR(i,N) assert(in_circle(F[i], C));  // sanity check,
       everyone should be in the circle
    ans = min(ans, PI * r * r * height);
  }

  cout << fixed << setprecision(8) << ans << endl;
}
```

2.13 monotone chain

```
#include <bits/stdc++.h>
using namespace std;

typedef long long ll;

struct point {
    ll x, y;
    bool operator < (const point &P) {
        return make_pair(x, y) < make_pair(P.x, P.y);
    }
    bool operator != (const point &P) {
        return make_pair(x, y) != make_pair(P.x, P.y);
    }
};

bool CCW (point a , point b , point c) {
    long long area = ( a.x - b.x ) * ( b.y - c.y ) - ( b.x - c.x
       ) * ( a.y - b.y );
    if (area > 0) return 1; /// change to >= 0 if redundant
       points are also required
    return 0;
}

void monotone_chain (vector <point> &p, vector <point> &hull) {
    hull.clear();
    sort (p.begin() , p.end());
    vector <point> L, U;
    for(int i = 0 ; i < p.size() ; i++) {
        while(L.size() >= 2 and !CCW(L[L.size()-2] , L.back() ,
           p[i])) {
            L.pop_back();
        }
        L.push_back(p[i]);
    }
    for(int i = (int)p.size()-1 ; i >= 0 ; i--) {
        while(U.size() >= 2 and !CCW(U[U.size()-2] , U.back() ,
           p[i])) {
            U.pop_back();
        }
        U.push_back(p[i]);
    }
    for(int i = 0; i < L.size(); i++) hull.push_back(L[i]);
    if (L.back() != U[0]) hull.push_back(U[0]);
    for(int i = 1; i + 1 < U.size(); i++) hull.push_back(U[i]);
```

```
    if (U.back()!=L[0]) hull.push_back(U.back());
}

int main () {
    int n;
    cin >> n;
    vector <point> v(n);
    for(int i = 0; i < n; i++) {
        cin >> v[i].x >> v[i].y;
    }
    vector <point> hull;
    monotone_chain(v, hull);
    for(int i = 0; i < hull.size(); i++) {
        cout << hull[i].x << " " << hull[i].y << endl;
    }
    return 0 ;
}
```

2.14 new 3d geo

```
#include<bits/stdc++.h>
using namespace std;
const double eps=1e-10;

struct PT {
  double x, y, z;
  PT() {}
  PT(double x, double y, double z) : x(x), y(y), z(z) {}
  PT(const PT &p) : x(p.x), y(p.y), z(p.z) {}
};
PT operator +(PT a,PT b){
  return PT(a.x+b.x,a.y+b.y, a.z+b.z);
}
PT operator -(PT a,PT b){
  return PT(a.x-b.x,a.y-b.y, a.z-b.z);
}
PT operator *(PT a,double b){
  return PT(a.x*b,a.y*b, a.z*b);
}
PT operator /(PT a,double b){
  return PT(a.x/b,a.y/b, a.z/b);
}
double operator *(PT a,PT b) {
  return a.x*b.x+a.y*b.y+a.z*b.z;
}
PT operator ^(PT a,PT b){
  return PT(a.y*b.z-a.z*b.y,a.z*b.x-a.x*b.z,a.x*b.y-a.y*b.x);
}
int dcmp(double x){
  if (abs(x)<eps) return 0;
  return x<0 ? -1 : 1;
}
bool operator <(const PT &a,const PT &b){
  return make_pair(make_pair(a.x,a.y), a.z) <
         make_pair(make_pair(b.x,b.y), b.z);
}
bool operator==(const PT &a,const PT &b){
return dcmp(a.x-b.x)==0&&dcmp(a.y-b.y)==0&&dcmp(a.z-b.z)==0;
}
double len(PT a){
  return sqrt(a*a);
}
double dist(PT a, PT b){
  return sqrt((a-b)*(a-b));
}
double dist2(PT a, PT b){
  return ((a-b)*(a-b));
}
PT reversePT(PT a){
```

```
  return a*(-1);
}
///Angle between two vector
double angleRad( PT a, PT b ){
  return acos( max(-1.0, min(1.0, (a*b)/(len(a)*len(b)))) );
}
///small angle between two vector
double smallAngle( PT a, PT b ){
   return acos( min(abs(a*b)/len(a)/len(b), 1.0) );
}
///u + dt
struct Line{
  PT d, u;
  Line(PT d,PT u):d(d),u(u){}
  PT point(double t){
    return u + d*t;
  }
};
///ax + by + cz = d
struct Plane{
    double a,b,c,d;PT n;
    Plane(){}
    Plane(PT p, PT q, PT r ) :
      Plane( (q-p)^(r-p), ((q-p)^(r-p))*(p) ) {}
    ///normal in direction of p,q,r
    Plane(double a, double b, double c, double d) :
      a(a), b(b), c(c), d(d), n(PT(a,b,c)){}
    Plane(PT n, double d) :
      n(n), a(n.x), b(n.y), c(n.z), d(d) {}
    Plane(const Plane &p) :
      n(p.n), d(p.d), a(p.a), b(p.b), c(p.c) {}
};
///returns 0 if t is on plane p
///returns 1/-1 if t is on positive/negative side of normal
int side( Plane &p, PT a ){
  return dcmp(p.a*a.x + p.b*a.y + p.c*a.z - p.d);
}



///translate all point on a plane with respect to t
Plane Translate( Plane &p, PT t ){
  return Plane( p.n, p.d + p.n*t );
}
///rotate d to the left with respect to normal in plane p
PT rotateCCW90(Plane p, PT d){
  return (p.n^d);
}
PT unitVector( PT v ){
  return v/len(v);
}
///rotate d to the right with respect to normal in plane p
PT rotateCW90(Plane p, PT d){
  return (d^p.n);
}
///shift plane up(dist>0)/down(dist<0) to distance dist
Plane ShiftUpDown( Plane &p, double dist ){
  return Plane( p.n, p.d + dist*len(p.n) );
}
///returns 0 if t is on plane of a,b,c
///returns 1/-1 if t is on positive/negative side of a,b,c
int orientPointPlane( PT a, PT b, PT c, PT t ){
  double v = ( (b-a)^(c^a) )*(t-a);
  return dcmp(v);
}
///projection of point q on plane p
PT projectPointPlane( Plane &p, PT q ){
  return PT( q + p.n*((p.d- p.n*q)/(p.n*p.n)) );
}
///reflection of point q on plane p
```

```
PT reflectPointPlane( Plane &p, PT q ){
  return PT( q + p.n*(2.0*((p.d- p.n*q)/(p.n*p.n))) );
}
///assuming a is the center, ab is new x axis
vector<PT> convert3Dto2D( PT a, PT b, PT c, vector<PT>pt ){
  PT n = (b-a)^(c-a),dx = unitVector(b-a),
    dy = unitVector(n^(b-a));
  vector<PT>newpt;
  for( int i = 0; i < pt.size(); i++ )
    newpt.push_back( PT( dx*(pt[i]-a), dy*(pt[i]-a), 0 ) );
  return newpt;
}
double distancePointLine( Line l, PT p ){
  return len(l.d^( p-l.u ))/len(l.d);
}
PT projectPointLine( Line l, PT p ){
  return PT( l.u + l.d*(( (p-l.u)*(l.d) )/(l.d*l.d)) );
}
PT reflectPointLine( Line l, PT p ){
  return PT( projectPointLine(l,p)*2.0 - p );
}
///undefined if line and plane is parallel ie( p.b*l.d = 0 )
PT intersectionLinePlane( Line &l, Plane &p ){
  double k = (p.d - (p.n*l.u))/(p.n*l.d);
  return PT(l.u + l.d*k);
}
Line intersectioPlanePLane( Plane &p1, Plane &p2 ){
  PT d = p1.n^p2.n;
  return Line(d, ((p2.n*p1.d - p1.n*p2.d)^d)/(d*d));
}
double distanceLineLine( Line &l1, Line &l2 ){
  PT d = l1.d^l2.d;
  if( dcmp(len(d))==0 ) return distancePointLine(l1, l2.u);
  return abs( (l2.u-l1.u)*d )/len(d);
}
```

```
PT closestPointOnL1fromL2( Line &l1, Line &l2 ){
  PT n = l1.d^l2.d, n3 = l2.d^n;
  ///p is the plane including line l2 and n
  Plane p = Plane( n3, n3*l2.u );
  return intersectionLinePlane( l1, p );
}
///2 planes are parallel if crs product of their normal is 0
///2 planes are parallel if dot product of their normal is 0
///angle between two lines is angle between direction vector
double smallAngleBetweenTwoPlane( Plane p1, Plane p2 ){
  return smallAngle(p1.n, p2.n);
}
double angleBetweenTwoPlane( Plane p1, Plane p2 ){
  return angleRad(p1.n, p2.n);
}
double smallAngleBetweenPlaneLine( Plane &p1, Line &l1 ){
  return acos(-1.0) - smallAngle(p1.n, l1.d);
}

bool intersectionLineSegmentSphere(PT cen, double r, Line l){
  double h2 = r*r - distancePointLine(l, cen)*
                    distancePointLine(l, cen);
  if( dcmp(h2) < 0 ) return 0;
  if( dcmp(h2) == 0 ){
    return OnSegment(projectPointLine(l, cen), l.a, l.b);
  }
  PT v = projectPointLine(l, cen);
  PT h = l.d*sqrt(h2)/len(l.d);
  return OnSegment(v+h,l.a,l.b)&&OnSegment(v-h,l.a,l.b);
}

double angleRad( PT a, PT b ){
  return acos( max(-1.0, min(1.0, (a*b)/(len(a)*len(b)))) );
}
// returns 0 if on any end
```

```
bool OnSegment(PT p,PT a,PT b) {
  return dcmp(len((a-p)^(b-p))) == 0 && dcmp((a-p)*(b-p)) < 0;
}

double tri_area( PT a, PT b, PT c ){
  return 0.5*len((b-a)^(c-a));
}
struct Face{
  PT a, b, c;
  Face(){}
  Face(PT a, PT b, PT c) : a(a), b(b), c(c) {}
  Face( const Face &f ) : a(f.a), b(f.b), c(f.c) {}
};
///phi = longitude, lamda = lattitude
struct Sphere{
  PT cen; double r;
  Sphere(){}
  Sphere( const Sphere &s ) : cen(s.cen), r(s.r) {}
  Sphere( PT cen, double r ) : cen(cen), r(r) {}
  PT convert( double phi, double lamda ){
    return PT( r*cos(phi)*cos(lamda),r*cos(phi)*sin(lamda),
             r*sin(phi) );
  }
};
double surfaceArea( vector<Face> &vec){
  double s = 0;
  for( int i = 0; i < vec.size(); i++ )
    s = s + len((vec[i].b-vec[i].a)^(vec[i].c-vec[i].a));
  return s*0.5;
}
double ployhedronVolume( vector<Face> &vec ){
  if( vec.size() == 0 ) return 0;
  PT reff = vec[0].a; double vol = 0;
  for( int i = 1; i < vec.size(); i++ ) {
    PT ar = (vec[i].b-vec[i].a)^(vec[i].c - vec[i].a);
    vol += abs( ar*(reff-vec[i].a) );
  }
  return vol/6.0;
}
vector<PT> intersectionLineSphere(PT cen, double r, Line l){
  vector<PT>vec;
  double h2 = r*r - distancePointLine(l, cen)*
                    distancePointLine(l, cen);
  if( dcmp(h2) < 0 ) return vec;
  if( dcmp(h2) == 0 ){
    vec.push_back( projectPointLine(l, cen) );
    return vec;
  }
  PT v = projectPointLine(l, cen);
  PT h = l.d*sqrt(h2)/len(l.d);
  vec.push_back(v+h); vec.push_back(v-h);
  return vec;
}
/// let's consider the case of a spherical triangle ABC.
///It's area is given by r2(a + b + c - pi) where r is
///the radius of the sphere and a; b; c are the amplitudes
///of the three interior angles of ABC
bool InsideATriangle (PT A , PT B , PT C , PT P) {
  if (abs(tri_area(A,B,P) + tri_area(A,C,P) +
      tri_area(B,C,P) - tri_area(A,B,C)) < eps) return 1 ;
  return 0 ;
}
///project point c onto line segment through a and b
PT projectPointSegment(PT a, PT b, PT c){
  double r = (b-a)*(b-a);
  if(abs(r) < eps) return a;
  r = ( (c-a)*(b-a) ) / r;
  if (r < 0) return a; if (r > 1) return b;
  return a + (b-a)*r;
```

```
}
///compute distance from c to segment between a and b
double distancePointSegment(PT a, PT b, PT c){
  return dist(c, projectPointSegment(a, b, c));
}
///Minimum distance from Point P on a triangle with vertices
    A,B,C
double PointDistanceOn3dTriangle(PT A, PT B, PT C, PT P){
  Plane ABC = Plane(A,B,C); PT P_ = projectPointPlane(ABC,P);
  double ret = 1e19 ;
  if (InsideATriangle(A,B,C,P_))
      ret = min(ret, dist(P,P_)) ;
  ret = min(ret, distancePointSegment(A,B,P)) ;
  ret = min(ret, distancePointSegment(B,C,P)) ;
  ret = min(ret, distancePointSegment(A,C,P)) ;
  return ret ;
}
vector<Face> Convex3dHull(vector<PT> &V) {
  vector <Face> Faces ;
  for (int i = 0 ; i < V.size() ; i++) {
    for (int j = i+1 ; j < V.size() ; j++) {
      for (int k = j+1 ; k < V.size() ; k++) {
        if (tri_area(V[i],V[j],V[k]) < eps)
          continue ;
        bool up = 0 , down = 0 ;
        PT AB = V[j]-V[i] , AC = V[k]-V[i] ;
        PT normal = AB^AC ;
        for (int l = 0 ; l < V.size() ; l++) {
          if (l == i or l == j or l == k)
            continue ;
          if (abs(normal*(V[l]-V[i])) < eps) {
            if ( abs( ( tri_area(V[i],V[j],V[l]) +
                      tri_area(V[i],V[k],V[l]) +
                      tri_area(V[j],V[k],V[l]) -
                      tri_area(V[j],V[k],V[i]) ) ) < eps ){
              up = down = 1 ;
              break ;
            }
          }
          else if (normal*(V[l]-V[i]) < 0)
            down = 1 ;
          else
            up = 1 ;
        }
        if (up == 0 or down == 0) {
          Face temp;
          temp.a = V[i], temp.b = V[j] , temp.c = V[k] ;
          Faces.push_back(temp) ;
        }
      }
    }
  }
  return Faces ;
}

double greatCirclePointDistance( Sphere s, double phi1,
              double lamda1, double phi2, double lamda2){
  PT p1 = s.convert( phi1, lamda1 );
  PT p2 = s.convert( phi2, lamda2 );
  //always takes into account smallest distance
  return angleRad( p1-s.cen, p2-s.cen )*s.r;
}

double greatCircleArea( Sphere s, double phi1, double lamda1,
    double phi2, double lamda2, double phi3, double lamda3 )
{
  PT p1 = s.convert( phi1, lamda1 ),
  p2 = s.convert( phi2, lamda2 ),p3 = s.convert(phi3,lamda3);
  double a = angleBetweenTwoPlane( Plane(s.cen, p1, p2),
                                   Plane(s.cen, p1, p3) );
```

```
  double b = angleBetweenTwoPlane( Plane(s.cen, p2, p3),
                                   Plane(s.cen, p2, p1) );
  double c = angleBetweenTwoPlane( Plane(s.cen, p3, p1),
                                   Plane(s.cen, p3, p2) );
  return s.r*s.r*( a+b+c-acos(-1.0) );
}
```

2.15 point rotation trick

```
/// point rotation trick

#include <bits/stdc++.h>
using namespace std;
typedef long long lint;
typedef pair<lint, lint> pi;
const int mod = 1e9 + 7;

struct pnt{
    int x, y, idx;
    bool operator<(const pnt &p)const{
        return pi(x, y) < pi(p.x, p.y);
    }
}a[5005];

struct line{
    int dx, dy, i1, i2;
};

vector<line> v;
int n, rev[5005];
lint p, q;
lint ccw(pnt a, pnt b, pnt c){
    int dx1 = b.x - a.x;
    int dy1 = b.y - a.y;
    int dx2 = c.x - a.x;
    int dy2 = c.y - a.y;
    return abs(1ll * dx1 * dy2 - 1ll * dy1 * dx2);
}

long long ans = 0;

void solve(int c1, int c2, lint l){
    ans = max(ans, ccw(a[c1], a[c2], a[0]));
    ans = max(ans, ccw(a[c1], a[c2], a[n-1]));
}

int main(){
    cin >> n;
    for(int i=0; i<n; i++){
        cin >> a[i].x >> a[i].y;
    }
    sort(a, a+n);
    for(int i=0; i < n; i++){
        a[i].idx = i;
        rev[i] = i;
    }
    for(int i=0; i<n; i++){
        for(int j=i+1; j<n; j++){
            v.push_back({a[j].x - a[i].x, a[j].y - a[i].y,
                a[i].idx, a[j].idx});
        }
    }
    sort(v.begin(), v.end(), [&](const line &a, const line &b){
        lint cw = 1ll * a.dx * b.dy - 1ll * b.dx * a.dy;
        if(cw != 0) return cw > 0;
        return pi(a.i1, a.i2) < pi(b.i1, b.i2);
    });
    lint ret = 0;
    for(int i=0; i<v.size(); i++){
```

```
        int c1 = rev[v[i].i1], c2 = rev[v[i].i2];
        if(c1 > c2) swap(c1, c2);
        solve(c1, c2, p);
        swap(a[c1], a[c2]);
        swap(rev[v[i].i1], rev[v[i].i2]);
    }
    cout << setprecision(12) << fixed << ans/2.0 << endl;
}
```

2.16 union of circle

```
/// union of N <= 50 circle and their individual area
///
    https://codeforces.com/gym/100941/attachments/download/4189/20082009-summer-petrozavodsk-camp-petrozavodsk-su-wx-contest-en.pdf
#include <bits/stdc++.h>
using namespace std;

#define fore(i, l, r) for(int i = int(l); i < int(r); ++i)
#define forn(i, n) fore(i, 0, n)
#define fori(i, l, r) fore(i, l, int(r) + 1)

#define ifor(i, l, r) for(int i = int(r); i >= int(l); --i)
#define efor(i, l, r) ifor(i, l, int(r) - 1)
#define nfor(i, n) efor(i, 0, n)

#define sz(v) int((v).size())
#define all(v) (v).begin(), (v).end()
#define pb push_back
#define mp make_pair
#define x first
#define y second

template<typename T> inline T abs(T a){ return ((a < 0) ? -a :
    a); }

template<typename T> inline T sqr(T a){ return a * a; }

template <typename T1, typename T2>
ostream& operator <<(ostream& out, const pair<T1, T2>& p) {
        return out << "(" << p.x << ", " << p.y << ")";
}

typedef long long li;
typedef long double ld;

#ifdef KUVIMAN
#define LLD "%lld"
#else
#define LLD "%I64d"
#endif

bool solve(int);

ld getTime() {
        return clock() / ld(CLOCKS_PER_SEC);
}

int main() {
#ifdef KUVIMAN
        assert(freopen("input.txt", "r", stdin));
#else
        assert(freopen("runes.in", "r", stdin));
        assert(freopen("runes.out", "w", stdout));
#endif

        cout << fixed << setprecision(10);
        cerr << fixed << setprecision(3);

#ifdef KUVIMAN
        ld st = getTime();
```

```
#endif
	int test = 0;
	while (solve(test)) {
		++test;
#ifdef KUVIMAN
		ld ct = getTime();
		cerr << " == TIME : " << ct - st << " == " << endl;
		st = ct;
#endif
	}
	return 0;
}

// == TEMPLATE END ==

template <typename T1, typename T2>
auto max(const T1& a, const T2& b) -> decltype(true ? a : b) {
	return a > b ? a : b;
}
template <typename T1, typename T2>
auto min(const T1& a, const T2& b) -> decltype(true ? a : b) {
	return a < b ? a : b;
}

const ld EPS = 1e-8;
bool eqEps(ld a, ld b) {
	return abs(a - b) < EPS;
}

typedef pair<ld, ld> pt;
pt operator -(const pt& a, const pt& b) {
	return pt(a.x - b.x, a.y - b.y);
}
pt operator +(const pt& a, const pt& b) {
	return pt(a.x + b.x, a.y + b.y);
}
pt operator *(const pt& v, const ld& k) {
	return pt(v.x * k, v.y * k);
}
pt operator *(const ld& k, const pt& v) {
	return pt(k * v.x, k * v.y);
}
pt operator /(const pt& v, const ld& k) {
	return pt(v.x / k, v.y / k);
}
ld dot(const pt& a, const pt& b) {
	return a.x * b.x + a.y * b.y;
}
ld cross(const pt& a, const pt& b) {
	return a.x * b.y - a.y * b.x;
}
ld len2(const pt& v) {
	return dot(v, v);
}
ld len(const pt& v) {
	return sqrtl(len2(v));
}
pt rotate(const pt& v, const ld& sn, const ld& cs) {
	return pt(v.x * cs - v.y * sn, v.x * sn + v.y * cs);
}
pt norm(const pt& v) {
	return v / len(v);
}

vector<pt> intersect(const pair<pt,ld>& a, const pair<pt,ld>& b)
   {
	ld d = len(b.x - a.x);
	if (d < EPS)
		return vector<pt>();
	ld cs = (sqr(d) + sqr(a.y) - sqr(b.y)) / (2 * d * a.y);
```

```
	if (cs < -1 - EPS || cs > 1 + EPS)
		return vector<pt>();
	ld sn = sqrtl(max(0, 1 - sqr(cs)));
	vector<pt> res;
	pt v = norm(b.x - a.x) * a.y;
	res.pb(a.x + rotate(v, sn, cs));
	res.pb(a.x + rotate(v, -sn, cs));
	return res;
}

struct Event {
	ld y1, y2;
	bool down;
	pt c;
	ld r;
	Event(ld x1, ld x2, const pair<pt,ld>& c, bool down) {
		y1 = sqrtl(max(0, sqr(c.y) - sqr(x1 - c.x.x)));
		y2 = sqrtl(max(0, sqr(c.y) - sqr(x2 - c.x.x)));
		if (down) {
			y1 = -y1;
			y2 = -y2;
		}
		y1 += c.x.y;
		y2 += c.x.y;
		this->down = down;
		this->c = c.x;
		r = c.y;
	}
	Event(ld y) : y1(y), y2(y) {}
};

ld area(ld x1, ld x2, const Event& e) {
	ld res = (x2 - x1) * (e.y1 + e.y2) / 2;
	ld cs = dot(norm(pt(x1, e.y1) - e.c), norm(pt(x2, e.y2) -
	    e.c));
	cs = max(-1, min(1, cs));
	ld sector = sqr(e.r) * acosl(cs) / 2 - abs(cross(pt(x1,
	    e.y1) - e.c, pt(x2, e.y2) - e.c)) / 2;
	if (e.down)
		res -= sector;
	else
		res += sector;
	return res;
}

bool operator <(const Event& a, const Event& b) {
	if (abs(a.y1 - b.y1) > EPS)
		return a.y1 < b.y1;
	if (abs(a.y2 - b.y2) > EPS)
		return a.y2 < b.y2;
	if (a.down != b.down)
		return a.down;
	return (a.r > b.r) != a.down;
}
bool cmpY1(const Event& a, const Event& b) {
	return a.y1 < b.y1;
}
bool cmpY2(const Event& a, const Event& b) {
	return a.y2 < b.y2;
}

typedef pair<int,int> pti;
map<pti,pti> dsu;
pti root(pti v) {
	if (dsu.count(v) == 0)
		dsu[v] = v;
	if (dsu[v] == v) return v;
	return dsu[v] = root(dsu[v]);
}
```

```
bool solve(int) {
	int n;
	if (scanf("%d", &n) != 1)
		return false;
	vector<pair<pt,ld>> circles(n);
	forn(i, n) {
		int x, y, r;
		assert(scanf("%d%d%d", &x, &y, &r) == 3);
		circles[i] = mp(pt(x, y), r);
	}
	sort(all(circles));
	circles.erase(unique(all(circles)), circles.end());

	circles.pb(mp(pt(0, 0), 1e4));

	vector<ld> xs;
	for (auto c1 : circles) {
		xs.pb(c1.x.x - c1.y);
		xs.pb(c1.x.x + c1.y);
		for (auto c2 : circles) {
			auto inters = intersect(c1, c2);
			for (auto p : inters)
				xs.pb(p.x);
		}
	}
	sort(all(xs));
	xs.erase(unique(all(xs), eqEps), xs.end());

	vector<vector<Event>> evs(sz(xs) - 1);
	forn(i, sz(evs)) {
		ld x1 = xs[i], x2 = xs[i + 1];
		for (auto c : circles) {
			if (min(x1, x2) + EPS >= c.x.x - c.y &&
			    max(x1, x2) <= c.x.x + c.y + EPS) {
				evs[i].pb(Event(x1, x2, c, true));
				evs[i].pb(Event(x1, x2, c, false));
			}
		}
		sort(all(evs[i]));
	}

	dsu.clear();
	forn(i, sz(evs) - 1) {
		vector<ld> ys;
		for (auto e : evs[i])
			ys.pb(e.y2);
		for (auto e : evs[i + 1])
			ys.pb(e.y1);
		sort(all(ys));
		ys.erase(unique(all(ys), eqEps), ys.end());

		forn(j, sz(ys) - 1) {
			ld y = (ys[j] + ys[j + 1]) / 2;
			int id1 = int(upper_bound(all(evs[i]),
			    Event(y), cmpY2) - evs[i].begin());
			int id2 = int(upper_bound(all(evs[i + 1]),
			    Event(y), cmpY1) - evs[i + 1].begin());
			pti a = root(pti(i, id1));
			pti b = root(pti(i + 1, id2));
			if (a == b)
				continue;
			if (rand() & 1)
				swap(a, b);
			dsu[a] = b;
//			cerr << "MERGE " << a << ' ' << b << endl;
		}
	}

	map<pti,ld> ansq;
	forn(i, sz(evs)) {
```

```
                ld x1 = xs[i], x2 = xs[i + 1];
                ld prev = area(x1, x2, evs[i][0]);
                fore(j, 1, sz(evs[i])) {
                        ld cur = area(x1, x2, evs[i][j]);
//                      cerr << i << ' ' << j << " = " << cur -
    prev << endl;
                        ansq[root(pti(i, j))] += cur - prev;
                        prev = cur;
                }
        }

        vector<ld> ans;
        for (auto it : ansq) {
                if (it.x == root(pt(0, 1)))
                        continue;
                if (it.y < 1e-4)
                        continue;
                ans.pb(it.y);
        }

        sort(all(ans));

        printf("%d\n", sz(ans));
        forn(i, sz(ans)) {
                if (i) putchar(' ');
                printf("%.10f", double(ans[i]));
        }
        puts("");

        return true;
}
```

2.17 voronoi diagram

```
#include <bits/stdc++.h>
using namespace std;

typedef long long ll;

const int N = 1e6 + 100;

const double eps = 1e-9, pi = acos(-1);

struct PT {
    double x, y;
    PT() {}
    PT(double x_, double y_) {
        x = x_, y = y_;
    }
    PT operator + (const PT &p) const {
        return PT(x+p.x, y+p.y);
    }
    PT operator * (const double t) const {
        return PT(x*t, y*t);
    }
    PT operator / (const double t) const {
        return PT(x/t, y/t);
    }
    double operator * (const PT &p) const {
        return p.x*x + p.y*y;
    }
    PT operator - (const PT &p) const {
        return PT(x-p.x, y-p.y);
    }
    double operator ^ (const PT &p) {
        return x*p.y-y*p.x;
    }
    bool operator < (const PT &p) const {
```

```
        return make_pair(x, y) < make_pair(p.x, p.y);
    }
    PT rot90() {
        return PT(-y, x);
    }
    bool operator == (const PT &p) const {
        return fabs(x-p.x) < eps && fabs(y-p.y) < eps;
    }
};

double len(PT p) {
    return sqrt(p*p);
}

int dcmp(double x) {
    if(fabs(x) < eps) return 0;
    if(x < 0.0) return -1;
    return 1;
}

double ccw(PT p, PT q, PT r) {
    return (q-p)^(r-q);
}

// ax + by = c

struct line{
        double a, b, c;
        PT u, d;
        line(double a, double b,double c):a(a), b(b), c(c) {
        // careful that u, d is not updated here.
        }
    line(PT u_, PT d_) {
        u = u_, d = d_; // counter clockwise direction is the
            region
        a = d.y, b = -d.x, c = -u.y*d.x + u.x*d.y; // ax + by <= c
    }
        bool operator < (const line &l)const{
                bool flag1 = make_pair(a, b) > make_pair(0.0, 0.0);
                bool flag2 = make_pair(l.a, l.b) > make_pair(0.0,
                    0.0);
                if(flag1 != flag2) return flag1 > flag2;
                long double t = ccw(PT(0, 0), PT(a, b), PT(l.a,
                    l.b));
                return dcmp(t) == 0 ? c*hypot(l.a, l.b) < l.c *
                    hypot(a, b):t>0;
        }
        PT slope() {
        return PT(a, b);
    }
};

PT cross(line a, line b){
        double det = a.a * b.b - b.a * a.b;
        return PT((a.c * b.b - a.b * b.c) / det,
            (a.a * b.c - a.c * b.a) / det);
}

bool bad(line a, line b, line c){
        if(ccw(PT(0, 0), a.slope(), b.slope()) <= 0) return false;
        PT crs = cross(a, b);
        return crs.x * c.a + crs.y * c.b >= c.c;
}

// ax + by <= c;

bool hpi(vector<line> v, vector<PT> &solution){
        sort(v.begin(), v.end());
        deque<line> dq;
        for(auto &i : v) {
```

```
        if(!dq.empty() && !dcmp(ccw(PT(0, 0), dq.back().slope(),
            i.slope()))) continue;
        while(dq.size() >= 2 && bad(dq[dq.size()-2], dq.back(),
            i)) dq.pop_back();
                while(dq.size() >= 2 && bad(i,dq[0],dq[1]))
                    dq.pop_front();
                dq.push_back(i);
        }
        while(dq.size() > 2 && bad(dq[dq.size()-2], dq.back(),
            dq[0])) dq.pop_back();
        while(dq.size() > 2 && bad(dq.back(),dq[0],dq[1]))
            dq.pop_front();
        vector<PT> tmp;
        for(int i=0; i< dq.size(); i++){
                line cur = dq[i], nxt = dq[(i+1)%dq.size()];
                if(ccw(PT(0, 0), cur.slope(), nxt.slope()) <= eps)
                    return false;
                tmp.push_back(cross(cur, nxt));
        }
        solution = tmp;
        return true;
}

int main() {

    // freopen("in.txt", "r", stdin);
    ios::sync_with_stdio(0);
    cin.tie(0);
    cout << setprecision(12) << fixed;
    int n;
    cin >> n;
    vector<PT> P(n);
    for(int i = 0; i < n; i++) {
        cin >> P[i].x >> P[i].y;
    }
    double R = 1e9;
    vector<vector<PT>> voronoi_diagram;
    for(int i = 0; i < n; i++) {
        vector<line> lines;
        lines.push_back(line(1,0,R)); // x <= R
        lines.push_back(line(-1,0,R)); // x >= -R => -x <= R
        lines.push_back(line(0,1,R)); // y <= R
        lines.push_back(line(0,-1,R)); // y >= -R => -y <= R
        for(int j = 0; j < n; j++) {
            if(P[i] == P[j]) continue;
            PT u = (P[i]+P[j])*0.5;
            PT dir = P[j]-P[i];
            PT dir_90 = dir.rot90();
            PT v = u + dir_90;
            double a = dir_90.y, b = -dir_90.x, c = -u.y*dir_90.x
                + u.x*dir_90.y;
            lines.push_back(line(a,b,c));
        }
        vector<PT> polygon;
        hpi(lines, polygon);
        voronoi_diagram.push_back(polygon);
    }

    return 0;
}
```

3 Graph

3.1 BlockCutTree

```
namespace BCT
{
const int mx = 100005 ; //max(numberofedge , numberofnode )
```

```
bool isCutPoint[mx] ; int n , m ;
int low[mx] , pre[mx] , cnt2vcc , used[mx] ;
vector <int> biComp[mx] ;
struct Edge{
    int v , id ;
};
vector <Edge> g[mx] ;vector <int> bridges ; //for bridge
stack <int> stk ;

void init(int _n, int _m){
  n = _n ; m = _m ;
  for(int i=1 ; i<=n ; i++) g[i].clear(),biComp[i].clear();
  bridges.clear() ; /* for bridge */ }
void addEdge( int u, int v, int id ){
     g[u].pb( {v,id} ) ; g[v].pb({u,id}) ; }
void makeComponent( int edgeId ){
  ++cnt2vcc ;
  while( stk.size() != 0 ){
      biComp[cnt2vcc].pb( stk .top() ) ;
      if( stk.top() == edgeId ) { stk.pop() ; break ; }
      stk.pop() ;
    }
}
int dfs(int u, int par ,int edgeId ,int &cnt) {
  if( !used[edgeId] && edgeId !=0 ) {
      used[ edgeId ] = true ; stk.push(edgeId) ;
  }
  if( pre[u]!=-1 ) {
    low[par] = min( low[par] , pre[u] ) ;
    return low[par] ;
  }
//   printf("node-> %d par: %d edgeId: %d\n",u,par,edgeId) ;
    pre[u] = ++cnt ; low[u] = pre[u] ;
    int i ; bool hasChild = false ;
    for(i=0 ; i<g[u].size() ; i++) {
      if( g[u][i].id == edgeId ) continue ;
      int v = g[u][i].v ;
      if( dfs( v, u , g[u][i].id , cnt ) < 0 ) {
        low[u] = min( low[u] , low[v] ) ;
        if( low[ v ] == pre[ v ] ) {
          bridges.pb(g[u][i].id) ;
        }
      if( par==0 ? hasChild : low[v]>=pre[u] ) {
          isCutPoint[u] = true ;
          makeComponent(g[u][i].id) ;
      }
        hasChild = true ;
    }
  }

  if( par==0 && stk.size() != 0 ){ makeComponent(-1) ; }
  return -1 ;
  }

  int find2VCC() {
    int i , j ;
    int cnt = 0 ;
    for(i=1 ; i<=m ; i++) used[i] = false ;
    for(i=1 ; i<=n ; i++) {
      isCutPoint[i] = false ; pre[i] = -1 ;
    }
    cnt2vcc = 0 ;
    for(i=1; i<=n ; i++){
      if( pre[i]==-1 ) dfs(i,0,0,cnt) ;
    }
  }
}
{
    BCT::init(n,m) ;
    BCT::addEdge(u,v,i) ;
```

```
    BCT::find2VCC() ;
    int cntVcc = BCT::cnt2vcc ;
}
```

3.2 Blossom

```
const int MAXN = 2020 + 1;
struct GM  // 1-based Vertex index
{
  int vis[MAXN], par[MAXN], orig[MAXN], match[MAXN],
  aux[MAXN], t, N;
  vector<int> conn[MAXN]; queue<int> Q;
  void addEdge(int u, int v) {
    conn[u].push_back(v); conn[v].push_back(u);
  }
  void init(int n) {
    N = n; t = 0;
    for(int i=0; i<=n; ++i) {
      conn[i].clear(); match[i] = aux[i] = par[i] = 0;
    }
  }
  void augment(int u, int v) {
    int pv = v, nv;
    do {
      pv = par[v]; nv = match[pv]; match[v] = pv;
      match[pv] = v; v = nv;
    }
    while(u != pv);
  }

  int lca(int v, int w)
  {
    ++t;
    while(true) {
      if(v) {
        if(aux[v] == t) return v;
        aux[v] = t; v = orig[par[match[v]]];
      }
      swap(v, w);
    }
  }
  void blossom(int v, int w, int a) {
    while(orig[v] != a) {
      par[v] = w; w = match[v];
      if(vis[w] == 1) Q.push(w), vis[w] = 0;
      orig[v] = orig[w] = a; v = par[w];
    }
  }
  bool bfs(int u)
  {
    fill(vis+1, vis+1+N, -1);
    iota(orig + 1, orig + N + 1, 1); Q = queue<int> ();
    Q.push(u); vis[u] = 0;
    while(!Q.empty()) {
      int v = Q.front(); Q.pop();
      for(int x: conn[v])
      {
          if(vis[x] == -1) {
            par[x] = v; vis[x] = 1;
            if(!match[x]) return augment(u, x), true;
            Q.push(match[x]); vis[match[x]] = 0;
          }
          else if(vis[x] == 0 && orig[v] != orig[x])
          {
              int a = lca(orig[v], orig[x]);
              blossom(x, v, a); blossom(v, x, a);
          }
      }
    }
```

```
    return false;
  }
  int Match()
  {
    int ans = 0;
//find random matching (not necessary, constant improvement)
    vector<int> V(N-1); iota(V.begin(), V.end(), 1);
    shuffle(V.begin(), V.end(), mt19937(0x94949));
    for(auto x: V) if(!match[x]){
      for(auto y: conn[x]) if(!match[y]){
        match[x] = y, match[y] = x; ++ans;
        break;
      }
    }
    for(int i=1; i<=N; ++i) if(!match[i] && bfs(i)) ++ans;
    return ans;
  }
};
```

3.3 DinicMaxflow

```
#define MAXN 30010
#define clr(ar) memset(ar, 0, sizeof(ar))
/*Dinic's algorithm for directed graphs (0 based index for
graphs). For undirected graphs, just add two directed edges*/
const long long INF = (~0ULL) >> 1;
namespace flow{
  struct Edge{
  int u, v; long long cap, flow;
  Edge(){}
    Edge(int a, int b, long long c, long long f){
      u = a, v = b, cap = c, flow = f;
    }
  };
vector <int> adj[MAXN]; vector <struct Edge> E;
int n, s, t, ptr[MAXN], len[MAXN], dis[MAXN], Q[MAXN];
inline void init(int nodes, int source, int sink){
  clr(len); E.clear();
  n = nodes, s = source, t = sink;
  for (int i = 0; i < MAXN; i++) adj[i].clear();
}
  /// Adds a directed edge with capacity c
inline void addEdge(int a, int b, long long c){
  adj[a].push_back(E.size());
  E.push_back(Edge(a, b, c, 0));
  len[a]++; adj[b].push_back(E.size());
  E.push_back(Edge(b, a, 0, 0)); len[b]++;
}

inline bool bfs(){
  int i, j, k, id, f = 0, l = 0;
  memset(dis, -1, sizeof(dis[0]) * n);
  dis[s] = 0, Q[l++] = s;
  while (f < l && dis[t] == -1){
    i = Q[f++];
    for (k = 0; k < len[i]; k++){
      id = adj[i][k];
      if (dis[E[id].v] == -1 && E[id].flow < E[id].cap){
        Q[l++] = E[id].v; dis[E[id].v] = dis[i] + 1;
      }
    }
  }
  return (dis[t] != -1);
}
long long dfs(int i, long long f){
  if (i == t || !f) return f;
  while (ptr[i] < len[i]){
    int id = adj[i][ptr[i]];
    if (dis[E[id].v] == dis[i] + 1){
```

```
      long long x = dfs(E[id].v, min(f, E[id].cap - E[id].flow));
      if (x){
        E[id].flow += x, E[id ^ 1].flow -= x;
        return x;
      }
    }
    ptr[i]++;
  }
  return 0;
}

long long dinic(){
  long long res = 0;
  while (bfs()){
    memset(ptr, 0, n * sizeof(ptr[0]));
    while (long long f = dfs(s, INF)) {
      res += f;
    }
  }
  return res;
}
};
```

3.4 DirectedMst

```
struct Edge{
    int u, v, w;
    Edge(){}
    Edge(int a, int b, int c){ u = a, v = b, w = c;}
};
/// Directed minimum spanning tree in O(n * m)
/// Constructs a rooted tree of minimum total weight from the
    root node
/// Returns -1 if no solution from root
int directed_MST(int n, vector <Edge> E, int root){
    const int INF = (1 << 30) - 30;
    int i, j, k, l, x, y, res = 0;
    vector <int> cost(n), parent(n), label(n), comp(n);
    for (; ;){
        for (i = 0; i < n; i++) cost[i] = INF;
        for (auto e: E){
            if (e.u != e.v && cost[e.v] > e.w){
                cost[e.v] = e.w;
                parent[e.v] = e.u;
            }
        }
        cost[root] = 0;
        for (i = 0; i < n && cost[i] != INF; i++){};
        if (i != n) return -1; /// No solution
        for (i = 0, k = 0; i < n; i++) res += cost[i];
        for (i = 0; i < n; i++) label[i] = comp[i] = -1;
        for (i = 0; i < n; i++){
            for (x = i; x != root && comp[x] == -1; x =
                parent[x]) comp[x] = i;
            if (x != root && comp[x] == i){
                for (k++; label[x] == -1; x = parent[x]) label[x]
                    = k - 1;
            }
        }
        if (k == 0) break;
        for (i = 0; i < n; i++){
            if (label[i] == -1) label[i] = k++;
        }
        for (auto &e: E){
            x = label[e.u], y = label[e.v];
            if (x != y) e.w -= cost[e.v];
            e.u = x, e.v = y;
        }
        root = label[root], n = k;
```

```
    }
    return res;
}
```

3.5 Dominator Tree (Zawad)

```
#include<bits/stdc++.h>
using namespace std;
typedef vector<int> VI;
typedef vector<VI> VVI;

struct ChudirBhai
{
    int n;
    VVI g, tree, rg, bucket;
    VI sdom, par, dom, dsu, label;
    VI arr, rev;
    int T;

    ChudirBhai(int n): n(n), g(n+1), tree(n+1), rg(n+1),
        bucket(n+1),
                    sdom(n+1), par(n+1), dom(n+1), dsu(n+1),
                        label(n+1),
                    arr(n+1), rev(n+1), T(0)
    {
        for(int i = 1; i <= n; i++) sdom[i] = dom[i] = dsu[i] =
            label[i] = i;
    }

    void addEdge(int u, int v) { g[u].push_back(v); }

    void dfs0(int u)
    {
        T++; arr[u] = T, rev[T] = u;
        label[T] = T, sdom[T] = T, dsu[T] = T;

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

            if(!arr[w]) dfs0(w), par[arr[w]] = arr[u];
            rg[arr[w]].push_back(arr[u]);
        }
    }

    int Find(int u, int x = 0)
    {
        if(u == dsu[u]) return x? -1: u;

        int v = Find(dsu[u], x+1);
        if(v < 0) return u;

        if(sdom[label[dsu[u]]] < sdom[label[u]]) label[u] =
            label[dsu[u]];
        dsu[u] = v;

        return x? v: label[u];
    }

    void Union(int u, int v) { dsu[v] = u; }

    VVI buildAndGetTree(int s)
    {
        dfs0(s);

        for(int i = n; i >= 1; i--)
        {
            for(int j = 0; j < rg[i].size(); j++)
                sdom[i] = min(sdom[i], sdom[Find(rg[i][j])]);
```

```
            if(i > 1) bucket[sdom[i]].push_back(i);

            for(int j = 0; j < bucket[i].size(); j++)
            {
                int w = bucket[i][j], v = Find(w);

                if(sdom[v] == sdom[w]) dom[w] = sdom[w];
                else dom[w] = v;
            }

            if(i > 1) Union(par[i], i);
        }

        for(int i = 2; i <= n; i++)
        {
            if(dom[i] != sdom[i]) dom[i] = dom[dom[i]];

            tree[rev[i]].push_back(rev[dom[i]]);
            tree[rev[dom[i]]].push_back(rev[i]);
        }

        return tree;
    }

};

int main()
{
    int n, m;

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

    ChudirBhai bhai(n);

    for(int i = 1; i <= m; i++)
    {
        int u, v;
        scanf("%d %d", &u, &v);

        bhai.addEdge(u, v);
    }

    VVI tree = bhai.buildAndGetTree(1);

    for(int i = 1; i <= n; i++)
    {
        printf("%d: ", i);
        for(int j = 0; j < tree[i].size(); j++) printf("%d ",
            tree[i][j]);
        printf("\n");
    }

    return 0;
}
```

3.6 Euler Circuit For Directed Graph

```
#include <bits/stdc++.h>
using namespace std;

/// Hierholzer's algorithm, finds euler circuit in O(E)

const int N = 1e5 + 100;
vector<int> g[N];
int adj_used[N];
vector<int> circuit;

void euler(int u) {
```

```
    while(adj_used[u] < g[u].size()) {
        int v = g[u][adj_used[u]++];
        euler(v);
    }
    circuit.push_back(u);
}

void find_circuit(int n) {
    /// handles self-loop
    for(int i = 1; i <= n; i++) {
        int cnt = 0;
        for(int j = 0; j < adj[i].size(); j++) {
            if(adj[i][j] == i) {
                swap(adj[i][j], adj[i][cnt++]);
            }
        }
    }
    euler(1);
    reverse(circuit.begin(), circuit.end());
}
int main() {
    ios::sync_with_stdio(0);
    cin.tie(0);
    int n, m;
    cin >> n >> m;
    for(int i = 1; i <= m; i++) {
        int u, v;
        cin >> u >> v;
        g[u].push_back(v);
        g[v].push_back(u);
    }
    find_circuit(n);
    return 0;
}
```

3.7 Euler Tour (Undirected) Kundu

```
#include<bits/stdc++.h>
using namespace std;

struct Edge {
    int u, v;
    int other(int x) {return u^v^x;}
};

struct Eulerpath {
    int n;
    vector<Edge> edges;
    vector<bool> vis;
    vector<vector<int>> adj;

    Eulerpath(int nn) : n(nn), adj(nn) {}
    void addEdge(int u, int v) {
        edges.push_back({u, v});
        vis.push_back(0);
        int id = edges.size()-1;
        adj[u].push_back(id);
        adj[v].push_back(id);
    }

    bool getTour(vector<int> &path) {
        int u = 0, cnt=0;
        for (int i=0; i<n; i++)
            if(adj[i].size()%2) {
                u = i;
                ++cnt;
            }

        if (cnt !=0 && cnt != 2) return false;
```

```
        stack<int> st;
        path.clear();

        while (true) {
            while (adj[u].size() && vis[adj[u].back()] == 1)
                adj[u].pop_back();

            if (adj[u].empty()) {
                path.push_back(u);
                if (st.empty()) break;
                u = st.top();
                st.pop();
            }
            else {
                st.push(u);
                int id = adj[u].back();
                vis[id] = 1;
                u = edges[id].other(u);
            }
        }

        for (int i=0; i<edges.size(); i++)
            if (!vis[i])
                return false;
        return true;
    }
};


int main() {
    int n, m;
    cin>>n>>m;

    Eulerpath solver(n);
    while (m--) {
        int u, v;
        cin>>u>>v;
        solver.addEdge(u, v);
    }

    vector<int> v;
    bool b = solver.getTour(v);
    if (!b) cout<<"None"<<endl;
    else for (int x: v) cout<<x<<" ";
}
```

3.8 EulerPath(Arghya)

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

//#include <ext/pb_ds/assoc_container.hpp>
//#include <ext/pb_ds/tree_policy.hpp>

#define sf scanf
#define pf printf
#define pb push_back
#define mp make_pair
#define IN freopen("testing.txt","r",stdin)
#define OUT freopen("output.txt","w",stdout)
#define FOR(i,a,b) for(i=a ; i<=b ; i++)
#define DBG pf("Hi\n")
#define INF 1000000000
#define i64 long long int
#define eps (1e-8)
#define xx first
#define yy second
#define ln 17
#define off 2
```

```cpp
//using namespace __gnu_pbds;
using namespace std ;

typedef pair<int, int> pi ;
//typedef tree< pi, null_type, less<pi>, rb_tree_tag,
    tree_order_statistics_node_update> ordered_set;

const i64 mod = 1000000007LL ;

#define maxn 500005

/*
     *os.find_by_order(k) -> returns the k'th smallest element
        (indexing starts from 0)
      os.order_of_key(v) -> returns how many elements are
         strictly smaller than v
*/

int c[maxn] , d[maxn] ;

map< int , multiset<int> > g ;
map <int,int> vis ;


void dfs1(int u)
{
    vis[u] = 1 ;
    for( auto v : g[u] ) if( vis.find(v) == vis.end() ) dfs1(v) ;
}

///--------Euler path printing-----------///
//just call dfs2 with the node you want to start your path
//at first you need to make sure, the graph is connected and
    euler path exist
vector <int> ans ;

void dfs2(int u)
{
    while( (int)g[u].size() !=0 )
    {
        int v = *g[u].begin() ;
        g[u].erase( g[u].find(v) ) ;
        g[v].erase( g[v].find(u) ) ;
        dfs2(v) ;
    }
    ans.pb(u) ;
}
///--------Euler path printing-----------///


int main()
{
    int n ;

    scanf("%d",&n) ;

    for(int i=1 ; i<n ; i++) scanf("%d",&c[i]) ;
    for(int i=1 ; i<n ; i++) scanf("%d",&d[i]) ;

    for(int i=1 ; i<n ; i++)
    {
        if( c[i] > d[i] )
        {
            printf("-1\n") ;
            return 0 ;
        }
        g[ c[i] ].insert( d[i] ) ;
        g[ d[i] ].insert( c[i] ) ;
    }

    int src = c[1] , cnt = 0 ;
```

```
    for( auto it : g )
    {
        if( (int)it.second.size() & 1 )
        {
            cnt++ ;
            src = it.first ;
        }
    }

    dfs1( src ) ;

    if( vis.size() != g.size() || ( cnt!=0 && cnt!=2 ) )
    {
        printf("-1\n") ;
        return 0 ;
    }

    //call for printing euler path
    dfs2(src) ;

    for(int i=0 ; i<ans.size() ; i++)
    {
        printf("%d",ans[i]) ;
        if( i == (int)ans.size() - 1 ) printf("\n") ;
        else printf(" ") ;
    }

    return 0 ;
}
```

3.9 HopcroftKarp

```
struct Hopcroft_Karp { /// // N = left node + right node
const int NIL=0,INF=(1<<28),match[N],dist[N],n,m;
vector <int> G[N] ;
void init(int lft , int rgt) {
    n = lft , rgt = m ;
    for (int i = 0 ; i <= n+m+1 ; i++) G[i].clear() ;
}
void addEdge(int u , int v){ //u = left node from 1 to n
    G[u].push_back(v+n) ;//v = right node 1 to m
}
bool bfs(){
    queue <int> Q;
    for(int i = 1; i <= n ;i++) {
        if(match[i]==NIL) dist[i] = 0,Q.push(i);
        else dist[i] = INF;
    }
    dist[NIL] = INF;
    while(!Q.empty()) {
        int u = Q.front(); Q.pop();
        if(u!=NIL) {
            for(int i = 0; i < G[u].size(); i++) {
                int v = G[u][i];
                if(dist[match[v]]==INF) {
                    dist[match[v]] = dist[u] + 1;
                    Q.push(match[v]);
                }
            }
        }
    }
    return (dist[NIL]!=INF);
}
bool dfs(int u) {
    if(u!=NIL) {
        for(int i = 0; i < G[u].size() ; i++) {
            int v = G[u][i] ;
            if(dist[match[v]] == dist[u]+1) {
```

```
            if(dfs(match[v])) {
                match[v] = u;
                match[u] = v;
                return true;
            }
        }
    }
    dist[u] = INF;
    return false;
  }
  return true;
}

int hopcroft_karp() {
  memset( dist, 0, sizeof dist );
  memset( match, 0, sizeof match );
  int matching = 0 ;
  while(bfs()) {
    for(int i = 1 ; i <= n; i++) {
      if(match[i]==NIL && dfs(i)) {
        matching++ ;
      }
    }
  }
  return matching;
}
void VertexCover(vector<int>&color){///1: in min cover
  hopcroft_karp();
  vector< vector<int> > g(R+L+1) ; queue <int> Q;
  vector <int> vis(L+R+1,0) ;
  for(int u = 1 ; u <= L ; u++) {
    if (match[u]==0) Q.push(u) , vis[u] = 1;
    for(int i = 0 ; i < G[u].size(); i++) {
      int v = G[u][i] ;
      if (match[u] == v) g[v].push_back(u);
      else g[u].push_back(v);
    }
  }
  while(Q.size()) {
    int u = Q.front() ; Q.pop();
    for(int i = 0 ; i < g[u].size() ; i++) {
      int v = g[u][i] ;
      if (vis[v] == 0) vis[v] = 1 , Q.push(v);
    }
  }
  color.resize(R+L+1) ;
  for(int i = 1 ; i <= L ; i++) color[i] = (!vis[i]);
  for(int i = L+1 ; i <= L+R ; i++) color[i] = vis[i];
 }
};
/// call init() , then addEdge , then hopcroft_karp()
```

3.10 Hungarian Algorithm

```
#include <bits/stdtr1c++.h>

#define MAX 666
#define MAXIMIZE +1
#define MINIMIZE -1

#define inf (~0U >> 1)
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
#define dbg(x) cout << #x << " = " << x << endl
#define ran(a, b) ((((rand() << 15) ^ rand()) % ((b) - (a) + 1))
    + (a))

using namespace std;
```

```
namespace wm{ /// hash = 581023
    bool visited[MAX];
    int U[MAX], V[MAX], P[MAX], way[MAX], minv[MAX], match[MAX],
        ar[MAX][MAX];

    /// n = number of row and m = number of columns in 1 based,
        flag = MAXIMIZE or MINIMIZE
    /// match[i] contains the column to which row i is matched
    int hungarian(int n, int m, int mat[MAX][MAX], int flag){
        clr(U), clr(V), clr(P), clr(ar), clr(way);

        for (int i = 1; i <= n; i++){
            for (int j = 1; j <= m; j++){
                ar[i][j] = mat[i][j];
                if (flag == MAXIMIZE) ar[i][j] = -ar[i][j];
            }
        }
        if (n > m) m = n;

        int i, j, a, b, c, d, r, w;
        for (i = 1; i <= n; i++){
            P[0] = i, b = 0;
            for (j = 0; j <= m; j++) minv[j] = inf, visited[j] =
                false;

            do{
                visited[b] = true;
                a = P[b], d = 0, w = inf;

                for (j = 1; j <= m; j++){
                    if (!visited[j]){
                        r = ar[a][j] - U[a] - V[j];
                        if (r < minv[j]) minv[j] = r, way[j] = b;
                        if (minv[j] < w) w = minv[j], d = j;
                    }
                }

                for (j = 0; j <= m; j++){
                    if (visited[j]) U[P[j]] += w, V[j] -= w;
                    else minv[j] -= w;
                }
                b = d;
            } while (P[b] != 0);

            do{
                d = way[b];
                P[b] = P[d], b = d;
            } while (b != 0);
        }
        for (j = 1; j <= m; j++) match[P[j]] = j;

        return (flag == MINIMIZE) ? -V[0] : V[0];
    }
}

int main(){
}
```

3.11 Mincost Maxflow

```
#include<stdio.h>
#include<cstring>
#include<cstdlib>
#include<algorithm>
#include<vector>
#include<map>
#include<set>
#include<cmath>
#include<iostream>
```

```cpp
#include<assert.h>
#include<queue>
#include<string>

#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
#define dbg(x) cout << #x << " = " << x << endl
#define ran(a, b) ((((rand() << 15) ^ rand()) % ((b) - (a) + 1))
    + (a))

using namespace std;

/// Min-cost Max-flow using SPFA (Shortest Path Faster Algorithm)
/// 0 Based indexed for directed weighted graphs (for undirected
    graphs, just add two directed edges)

namespace mcmf{
    const int MAX = 1000010;
    const long long INF = 1LL << 60;

    long long cap[MAX], flow[MAX], cost[MAX], dis[MAX];
    int n, m, s, t, Q[10000010], adj[MAX], link[MAX], last[MAX],
        from[MAX], visited[MAX];

    void init(int nodes, int source, int sink){
        m = 0, n = nodes, s = source, t = sink;
        for (int i = 0; i <= n; i++) last[i] = -1;
    }

    void addEdge(int u, int v, long long c, long long w){
        adj[m] = v, cap[m] = c, flow[m] = 0, cost[m] = +w,
            link[m] = last[u], last[u] = m++;
        adj[m] = u, cap[m] = 0, flow[m] = 0, cost[m] = -w,
            link[m] = last[v], last[v] = m++;
    }

    bool spfa(){
        int i, j, x, f = 0, l = 0;
        for (i = 0; i <= n; i++) visited[i] = 0, dis[i] = INF;

        dis[s] = 0, Q[l++] = s;
        while (f < l){
            i = Q[f++];
            for (j = last[i]; j != -1; j = link[j]){
                if (flow[j] < cap[j]){
                    x = adj[j];
                    if (dis[x] > dis[i] + cost[j]){
                        dis[x] = dis[i] + cost[j], from[x] = j;
                        if (!visited[x]){
                            visited[x] = 1;
                            if (f && rand() & 7) Q[--f] = x;
                            else Q[l++] = x;
                        }
                    }
                }
            }
            visited[i] = 0;
        }
        return (dis[t] != INF);
    }

    pair <long long, long long> solve(){
        int i, j;
        long long mincost = 0, maxflow = 0;

        while (spfa()){
            long long aug = INF;
            for (i = t, j = from[i]; i != s; i = adj[j ^ 1], j =
                from[i]){
                aug = min(aug, cap[j] - flow[j]);
```

```
            }
            for (i = t, j = from[i]; i != s; i = adj[j ^ 1], j =
                from[i]){
                flow[j] += aug, flow[j ^ 1] -= aug;
            }
            maxflow += aug, mincost += aug * dis[t];
        }
        return make_pair(mincost, maxflow);
    }
}


int main(){

}
```

3.12 SCC+2SAT(arghya)

```
/*
2-sat
at first take a graph of size 2*n( for each variable, two
nodes ). for each clause of type ( a or b ), add two directed
edge !a-->b and !b-->a. if both x_i and !x_i is in same
connected component for some i, then this equations are
unsatisfiable . Otherwise there is a solution. Assume that f
is satisfiable. Now we want to give values to each variable
in order to satisfy f. It can be done with a topological sort
of vertices of the graph we made. If !x_i is after x_i in
topological sort, x_i should be FALSE. It should be TRUE
otherwise. say we have equation with three variable x1,x2,x3.
( x1 or !x2 ) and ( x2 or x3 ) = 1. so we add , x1,x2,x3 and
x4(as !x1) , x5(!x2) and x6(!x3) . Add edge x4-->x2 , x2-->x1
, x5-->x3 , x6-->x2.
you need to pass an array to the function findSCC, in which
result will be returned every node will be given a number,for
nodes of a single connected component the number will be same
this number representing nodes will be topologically sorted*/
class SCC{
public:
  vector <int> *g1 , *g2 ; int maxNode , *vis1 , *vis2 ;
  stack <int> st ;
  SCC(int MaxNode){
    maxNode = MaxNode ; vis1 = new int[maxNode+2] ;
    vis2 = new int[maxNode+2] ;
    g1 = new vector<int>[maxNode+2] ;
    g2 = new vector<int>[maxNode+2] ;
  }
  void addEdge(int u, int v) { g1[u].pb(v) ; g2[v].pb(u) ; }
  void dfs1(int u){
    if(vis1[u]==1) return ; vis1[u] = 1 ;
    for(int i=0 ; i<g1[u].size() ; i++) dfs1(g1[u][i]) ;
    st.push(u) ; return ;
  }
  void dfs2(int u, int cnt , int *ans){
    if(vis2[u]==1) return ; vis2[u] = 1 ;
    for(int i=0;i<g2[u].size();i++) dfs2(g2[u][i],cnt,ans) ;
      ans[u] = cnt ;
    }
    int findSCC( int *ans )
    {
      for(int i=1 ; i<=maxNode ; i++) vis1[i] = 0 ;
      for(int i=1 ; i<=maxNode ; i++){
        if(vis1[i]==0) dfs1(i) ;
      }
      int cnt = 0 ;
      for(int i=1 ; i<=maxNode ; i++) vis2[i] = 0 ;
      while( !st.empty() ) {
        int u = st.top() ;
        if(vis2[u]==0) { ++cnt ; dfs2( u , cnt , ans ) ; }
```

```
      st.pop() ;
    }
    for(int i=1 ; i<=maxNode ; i++) {
      g1[i].clear() ; g2[i].clear() ;
    }
    delete vis1 ; delete vis2 ; return cnt ;
  }
};
```

3.13 StronglyConnectedComponent

```
#include <bits/stdc++.h>
using namespace std;

// 1-based
namespace SCC {
    vector<vector<int>> g, rg, scc, sccg;
    vector<int> nodes, vis, who;
    int comp, n;
    void init(int n_) {
        n = n_;
        g.assign(n+1,{});
        rg.assign(n+1,{});
        vis.assign(n+1,0);
        nodes.clear();
        who.assign(n+1,0);
        comp = 0;
    }
    void addEdge(int u, int v) {
        g[u].push_back(v);
        rg[v].push_back(u);
    }
    void dfs1(int u) {
        vis[u] = 1;
        for(int v : g[u]) {
            if(!vis[v]) dfs1(v);
        }
        nodes.push_back(u);
    }
    void dfs2(int u) {
        who[u] = comp;
        for(int v: rg[u]) {
            if (!who[v]) dfs2(v);
        }
    }
    void SCC() {
        for(int i = 1; i <= n; i++) {
            if(!vis[i]) dfs1(i);
        }
        reverse(nodes.begin(), nodes.end());
        for(int u: nodes) {
            if (!who[u]) {
                ++comp;
                dfs2(u);
            }
        }
        scc.assign(comp+1,{});
        for(int i = 1; i <= n; i++) {
            scc[who[i]].push_back(i);
        }
        sccg.assign(comp + 1,{});
        for(int u = 1; u <= n; u++) {
            for(int v : g[u]) {
                if (who[u] != who[v]) {
                    sccg[who[u]].push_back(who[v]);
                }
            }
        }
        for(int i = 1; i <= comp; i++) {
```

```
            sort(sccg[i].begin(), sccg[i].end());
            sccg[i].erase(unique(sccg[i].begin(), sccg[i].end()),
                sccg[i].end());
        }
    }
}

int main() {
    int n, m;
    cin >> n >> m;
    SCC::init(n);
    for(int i = 1; i <= m; i++) {
        int u, v;
        cin >> u >> v;
        SCC::addEdge(u, v);
    }
    SCC::SCC();
}
```

3.14 diameter of tree

```
/// diameter of a tree
#include <bits/stdc++.h>
using namespace std;

typedef long long ll;

const int N = 1e6 + 100;

int n;
vector<int> G[N];
int far, far_node;
vector<int> dia_nodes;
int diameter;
int st, en;

void dfs1(int u, int p = -1, int d = 1) {
    if(d > far) {
        far = d;
        far_node = u;
    }
    for(int v : G[u]) {
        if(v != p) {
            dfs1(v, u, d + 1);
        }
    }
}
int dfs2(int u, int p = -1) {
    int fl = 0;
    if(u == en) {
        dia_nodes.push_back(u);
        return 1;
    }
    for(int v : G[u]) {
        if(v != p) {
            fl |= dfs2(v, u);
        }
    }
    if(fl) dia_nodes.push_back(u);
    return fl;
}

void find_diameter() {
    dia_nodes.clear();
    far = 0;
    dfs1(1);
    st = far_node;
    far = 0;
    dfs1(st);
```

```
    en = far_node;
    diameter = far;
    dfs2(st);
}


int main() {
    ios::sync_with_stdio(0);
    cin.tie(0);
    cin >> n;
    for(int i = 1; i < n; i++) {
        int u, v;
        cin >> u >> v;
        G[u].push_back(v);
        G[v].push_back(u);
    }
    find_diameter();
    return 0;
}
```

3.15 dominator for dag

```
/// Dominator Tree
#include <bits/stdc++.h>
using namespace std ;
#define mp make_pair
const int N = 1e5 + 5 , M = 5e5 + 5 , inf = 1e9 , LOG = 17 ;
int n , m ; // # of nodes and edges
vector < pair<int,int> > adj[N] ; // stores main graph
int a[M] , b[M] , W[M] ; // stores main graph
int dist[N] ; // distance calculated with dijkstra

vector <int> dag[N] ; // stores the DAG
vector <int> reverseDag[N] ; // stores reverse of DAG
stack <int> sorted ; // nodes in topSorted order of DAG
int vis[N] ; // used to topSort the nodes in DAG

vector <int> g[N] ; // stores dominator tree /// keeps edges in
    both way
int parent[N][LOG] ; // parents for dominator tree
int depth[N] ; // level of domiantor tree

int subTree[N] ;

void dijkstra (int src) {
    for (int i = 0 ; i <= n ; i++) {
        dist[i] = inf ;
    }
    dist[src] = 0 ;
    priority_queue < pair<int,int> > pq ;
    pq.push(mp(0,src)) ;
    while (!pq.empty()) {
        int u = pq.top().second ;
        int cost = -pq.top().first ;
        pq.pop() ;
        if (dist[u] < cost) {
            continue ;
        }
        for (int i = 0 ; i < adj[u].size() ; i++) {
            int v = adj[u][i].first , w = adj[u][i].second ;
            if (dist[u] + w < dist[v]) {
                dist[v] = dist[u] + w ;
                pq.push(mp(-dist[v],v)) ;
            }
        }
    }
}

void makeDag() {
```

```
    for (int i = 0 ; i <= n ; i++) {
        dag[i].clear() ;
        reverseDag[i].clear() ;
    }
    for (int i = 1 ; i <= m ; i++) {
        int u = a[i] , v = b[i] , w = W[i];
        if (dist[u] >= inf) continue ; // handles disconnected
            graph
        if (dist[u] + w == dist[v]) {
            dag[u].push_back(v) ;
            reverseDag[v].push_back(u) ;
        }
        else if (dist[v] + w == dist[u]) {
            dag[v].push_back(u) ;
            reverseDag[u].push_back(v) ;
        }
    }
}

void dfs (int u) {
    vis[u] = 1 ;
    for (int i = 0 ; i < dag[u].size() ; i++) {
        int v = dag[u][i] ;
        if (!vis[v]) {
            dfs(v) ;
        }
    }
    sorted.push(u) ;
}

void topSort () {
    memset (vis , 0 , sizeof vis) ;
    while (!sorted.empty()) sorted.pop() ;
    for (int i = 1 ; i <= n ; i++) {
        if (dist[i] >= inf) continue; // handles disconnected
            graph
        if (!vis[i]) {
            dfs(i) ;
        }
    }
}


int LCA (int u , int v) {
    if (depth[u] < depth[v]) { // u nichey
        swap(u,v) ;
    }
    int diff = depth[u] - depth[v] ;
    for (int i = LOG-1 ; i >= 0 ; i--) {
        if (diff >= (1<<i)) {
            diff -= (1<<i) ;
            u = parent[u][i] ;
        }
    }
    if (u == v) return u ;
    for (int i = LOG-1 ; i >= 0 ; i--) {
        if (parent[u][i] != parent[v][i]) {
            u = parent[u][i] ;
            v = parent[v][i] ;
        }
    }
    return parent[v][0] ;
}


void dominator (int src) {
    for (int i = 0 ; i <= n ; i++) {
        for (int k = 0 ; k < LOG ; k++) {
            parent[i][k] = -1 ;
```

```
        }
    }
    for (int i = 0 ; i <= n ; i++) {
        g[i].clear() ;
    }
    while (!sorted.empty()) {
        int u = sorted.top() ;
        sorted.pop() ;
        if (u == src) {
            continue ;
        }
        int par = reverseDag[u].back() ;
        for (int i = 0 ; i < reverseDag[u].size() ; i++) {
            int v = reverseDag[u][i] ;
            par = LCA(par,v) ;
        }
        g[u].push_back(par) ;
        g[par].push_back(u) ;
        parent[u][0] = par ;
        depth[u] = depth[par] + 1 ;
        for (int k = 1 ; k < LOG ; k++) {
            if (parent[u][k-1] == -1) {
                break ;
            }
            parent[u][k] = parent[parent[u][k-1]][k-1] ;
        }
    }
}

void solve (int u , int p) {
    subTree[u] = 1 ;
    for (int i = 0 ;i < g[u].size() ; i++) {
        int v = g[u][i] ;
        if (v == p) continue ;
        solve(v , u) ;
        subTree[u] += subTree[v] ;
    }
}

int src = 1 ;

int main () {
    //freopen ("in.txt", "r", stdin) ;
    int tc , caseno = 1 ;
    scanf ("%d" , &tc) ;
    while (tc--) {
        scanf ("%d %d" , &n , &m) ;
        for (int i = 0 ; i <= n ; i++) {
            adj[i].clear() ;
        }
        for (int i = 1 ; i <= m ; i++) {
            int u , v , w ;
            scanf ("%d %d %d" , &u , &v , &w) ;
            a[i] = ++u , b[i] = ++v , W[i] = w ;
            adj[u].push_back(mp(v,w)) ;
            adj[v].push_back(mp(u,w)) ;
        }
        dijkstra(src) ;
        makeDag() ;

        topSort() ;
        dominator(src) ;
        solve(src,-1) ;
        printf ("Case %d:\n" , caseno++) ;
        int Q ;
        scanf ("%d" , &Q) ;
        while (Q--) {
            int k ;
            scanf ("%d" , &k) ;
            int lca = -1 ;
```

```
            for (int i = 1 ; i <= k ; i++) {
                int u ; scanf ("%d" , &u) ;
                u++ ;
                if (dist[u] >= inf) {
                    continue ;
                }
                if (lca == -1) lca = u ;
                else lca = LCA(lca,u) ;
            }
            if (lca == -1) {
                printf ("0\n") ;
            }
            else {
                printf ("%d %d\n" , depth[lca] + 1 , subTree[lca])
                    ;
            }
        }
    }
}
```

3.16 dominator for general

```
namespace DominatorTree{
/*
Dominator Tree for General Graph ,Tr[u] stores all the
immediate children of node u (does not store the parent) in
the dominator tree. at first initialize with number of nodes.
then add edges(directed edges). call buildDominatorTree(r) ,
where r is the root. then just call dominator(u,v) to check
if v is u's dominator it returns false in case either u or v
is not connected to the root
*/
const int N = 202400;
vector <int> G[N] , pred[N] , dom[N] , Tr[N] , idom[N] , cnt;
int old[N] , dfn[N] , up[N] , f[N] , semi[N] , g[N] ;
int n , m , Time , st[N] , en[N] ;

void init(int _n) {
  for (int i = 0 ; i < N ; i++){
      G[i].clear() , pred[i].clear() , dom[i].clear() ,
      Tr[i].clear() ;
  }
  memset (old , 0 , sizeof old) ;
  memset (dfn , 0 , sizeof dfn) ;
  memset (f , 0 , sizeof f) ;
  memset (up , 0 , sizeof up) ;
  memset (old , 0 , sizeof old) ;
  memset (g , 0 , sizeof g) ;
  memset (idom , 0 , sizeof idom) ;
  memset (st , -1 , sizeof st) ;
  memset (en , -1 , sizeof en) ;
  n = _n ; cnt = 0 ; Time = 0 ;
}

void addEdge(int u , int v){ return G[u].push_back(v) ; }

void dfs(int u){
        old[dfn[u]=++cnt] = u ;
        semi[cnt] = g[cnt] = f[cnt] = cnt;
        for(int v : G[u]){
                if(!dfn[v]){
                        dfs(v);
                        up[dfn[v]] = dfn[u];
                }
                pred[dfn[v]].push_back(dfn[u]);
        }
}

int ff(int x) {
```

```
        if(x == f[x]) return x;
        int y = ff(f[x]) ;
        if(semi[g[x]] > semi[g[f[x]]])
                g[x] = g[f[x]];
        return f[x] = y;
}

void dfs1(int u)
{
    Time++ ;
    st[u] = Time ;
    for(int i=0 ; i<Tr[u].size() ; i++)
    {
        dfs1( Tr[u][i] ) ; //par is not stored in Tr[u]
    }
    Time++ ;
    en[u] = Time ;
}

void buildDominatorTree(int r){
        dfs(r);
        for(int y = cnt ; y >= 2 ; y--){
                for(int z : pred[y]) {
                        ff(z);
                        semi[y]=min(semi[y],semi[g[z]]);
                }
                dom[semi[y]].push_back(y);
                int x=f[y]=up[y];
                for(int z:dom[x]){
                        ff(z);
                        idom[z]=semi[g[z]]<x? g[z]:x;
                }
                dom[x].clear();
        }
        for(int y = 2 ; y <= cnt ; ++y){
                if(idom[y]!=semi[y])
                        idom[y]=idom[idom[y]];
                dom[idom[y]].push_back(y);
        }
        idom[r] = 0 ;
        for (int i = 1 ; i <= n ; i++) {
        for (int j = 0 ; j < dom[i].size() ; j++) {
            Tr[old[i]].push_back(old[dom[i][j]]) ;
        }
        }
        dfs1(r) ;
    }
    bool dominator( int u,int v )
    {
        //returns true if v is u's dominator
        if(st[u]==-1 || st[v]==-1) return false ;//if u or v is
            not connected to the root
        if( st[u] >= st[v] && st[u]<= en[v] ) return true ;
        return false ;
    }
}
```

3.17 two sat

```
/// copied from tanzir bhai
#include <bits/stdc++.h>
using namespace std;

typedef long long ll;

const int N = 1e5 + 100;
const int MAX = 1e5 + 100;

struct twoSat{
```

```
int n;
vector<int> E[MAX*2+10], V, Rev[MAX*2+10], sortedNodes;
bool state[MAX*2+10], vis[MAX*2+10];
int compId[MAX*2+10];

void init(int _n) {
    n = _n;
    for(int i = 0; i<= 2*n; i++) E[i].clear(), Rev[i].clear();
    V.clear();
    sortedNodes.clear();
    memset(state, 0, sizeof state);
}

int actual(int a) {
    if(a < 0) return n - a;
    else return a;
}

inline int neg(int a) {
    if(a > n) return a-n;
    else return n+a;
}

void dfs(int node) {
    vis[node] = true;
    for(auto v: E[node]) {
        if(!vis[v])
            dfs(v);
    }
    V.push_back(node);
}

void dfsRev(int node, int id) {
    sortedNodes.push_back(node);
    vis[node] = true;
    compId[node] = id;
    for(auto v: Rev[node]) {
        if(!vis[v])
            dfsRev(v, id);
    }
}

void buildSCC() {
    V.clear();
    mem(vis,0);
    for(int i = 1; i<=2*n; i++) {
        if(!vis[i]) dfs(i);
    }
    mem(vis,0);
    reverse(V.begin(), V.end());
    int cnt = 0;
    for(auto u: V) {
        if(!vis[u]) cnt++,dfsRev(u, cnt);
    }
}

bool topologicalOrder(int a, int b) {
    return compId[a] < compId[b];
}
bool satisfy() {
    buildSCC();
    /// if leader of i and -i is the same, then they are in
       the same component
    /// 2-sat is impossible, return 0
    for(int i = 1; i<=n; i++) {
        if(compId[i]==compId[i+n]) return 0;
    }
    /// topologically sort the components
```

```
        /// start from the back end of topologically sorted order
            and try to give everyone true state in that component
        /// if someone's opposite has true state, then let him
            have false state.
        for(int i = (int)sortedNodes.size()-1; i>=0; i--) {
            int u = sortedNodes[i];
            if( state[neg(u)] == 0)
                state[u]=1;
        }
        return 1;
    }

    void addEdge(int u, int v) {
        u = actual(u);
        v = actual(v);
        E[u].pb(v);
        Rev[v].pb(u);
    }
    void addOr(int u, int v) {
        addEdge(-u, v);
        addEdge(-v, u);
    }
    void forceTrue(int u) {
        addEdge(-u, u);
    }
    void forceFalse(int u) {
        addEdge(u,-u);
    }
    void addOriginalImplication(int u, int v) {
        addOr(-u,v);
    }
};

int main() {
}
```

4 Math

4.1 AND

```
struct ANDconvolution {
    ///poww(a,b,m) returns (a^b)%m
    ///XOR
    void WHtransform(vector<long long>&P,bool inverse=0 ) {
        for (int len = 1; 2 * len <= P.size(); len <<= 1) {
            for(int i = 0; i < P.size(); i += 2 * len) {
                for (int j = 0; j < len; j++) {
                    long long u = P[i + j];
                    long long v = P[i + len + j];
                    P[i + j] = u + v;//% mod
                    P[i + len + j] = u - v; //% mod
                }
            }
        }

        if (inverse) {
            //long long inv = poww(P.size(), mod-2, mod);
            for (int i = 0; i < P.size(); i++)
                P[i] = (P[i]/P.size()) ;
                //in case whole operation is done on modulo
        }
    }
    ///ORtransform
    void ANDtransform(vector<long long>&vec,bool inverse=0) {
        for(int len = 1; 2 * len <= vec.size(); len <<= 1) {
            for(int i = 0; i < vec.size(); i += 2 * len) {
                for(int j = 0; j < len; j++) {
```

```
                long long u = vec[i + j];
                long long v = vec[i + len + j];
                if( !inverse ) {
                    //AND
                    vec[i + j] = v;
                    vec[i + len + j] = (u + v);//%mod;
                    //OR
                    vec[i + j] = u + v;
                    vec[i + len + j] = u;//%mod;
                }
                else {
                    //AND
                    vec[i + j] = (-u + v);//%mod;
                    vec[i + len + j] = u;
                    //OR
                    vec[i + j] = v;//%mod;
                    vec[i + len + j] = u - v;
                }
            }
        }
    }
}
//input: two vector denoting coefficient of a polynomial
//output: a vector denoting their multiplication x^a*x^b
//= x^(a operation(and, or, xor) b)
vector<long long> multiply( vector<long long> v1,
                            vector<long long>v2 ) {
    int d = 1, dd = max( v1.size(), v2.size() );
    while(d<dd) d*=2;
    v1.resize(d, 0); v2.resize(d, 0);
    vector<long long>res(d, 0);
    ANDtransform(v1, 0); ANDtransform(v2, 0);
    for( int i = 0; i < d; i++ ) res[i] = v1[i]*v2[i];
    ANDtransform(res, 1);
    return res;
}
//input: two vector denoting coefficient of a polynomial
//output: a vector denoting (poly)^n
vector<long long> multiply( vector<long long>v1, int n )
{
    int d = 1, dd = v1.size();
    while(d<dd) d*=2;
    v1.resize(d, 0); vector<long long>res(d, 0);
    ANDtransform(v1, 0);
    for( int i = 0; i < d; i++ )
      res[i] = poww( v1[i], n, mod );
    ANDtransform(res, 1);
    return res;
}

};
```

4.2 Burnside Lemma

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

using namespace std ;
typedef pair <int,int> pii ;
const long long N = 1e6 +5 , md = 1e9 + 7 ;

long long BigMod(long long b , long long p) {
    if (p == 0) return 1 ;
    if (p == 1) return b ;
    long long t = BigMod(b,p/2) ;
    t = (t*t) ;
    if (p&1) return (t*b) ;
    return t ;
}
```

```
long long rot (long long n , long long c) { /// Sum of Fix(g)
    where g is a component of rotation set
    long long ans = 0 ;
    for (long long i = 0 ; i < n ; i++) {
        long long g = __gcd(n,i) ;
        ans = ans + BigMod(c,g) ;
    }
    return ans ;
}

long long reflection (long long n , long long c) {
    if (n&1) {
        return (n*BigMod(c,(n+1)/2)) ;
    }
    else {
        long long side = ((n/2)*BigMod(c,n/2)) ;
        long long vertex = ((n/2)*BigMod(c,n/2+1)) ;
        return (side + vertex) ;
    }
}

long long necklace(long long n,long long c) {
    long long ans = rot(n,c)/n ;
    return ans;
}

long long bracelet(long long n , long long c) {
    long long ans = rot(n,c) + reflection(n,c) ;
    ans = ans/(2*n);
    return ans ;
}

int main () {
    long long n ,c ;
    while (cin >> n >> c) {
        printf ("%lld %lld\n" , necklace(n,c) , bracelet(n,c)) ;
    }
    return 0;
}
```

4.3 ChineseRemainderTheorem

```
#include<bits/stdc++.h>
using namespace std;
const int N = 20;
intl GCD(intl a, intl b){}
intl LCM(intl a, intl b){}
inline long long normalize(long long x, long long mod) {
  x %= mod; if (x < 0) x += mod; return x;
}
struct GCD_type { long long x, y, d; };
GCD_type ex_GCD(long long a, long long b) {
    if (b == 0) return {1, 0, a};
    GCD_type pom = ex_GCD(b, a % b);
    return {pom.y, pom.x - a / b * pom.y, pom.d};
}
int testCases, t;
long long a[N], n[N], ans, lcm;
int main() {
    cin >> t;
    for(int i = 1; i <= t; i++)
      cin >> a[i] >> n[i], normalize(a[i], n[i]);
    ans = a[1]; lcm = n[1];
    for(int i = 2; i <= t; i++)
    {
        auto pom = ex_GCD(lcm, n[i]);
        int x1 = pom.x;
        int d = pom.d;
        if((a[i] - ans) % d != 0)
```

```
        return cerr << "No solutions" << endl, 0;
      ans = normalize(ans+x1*(a[i]-ans)/d%(n[i]/d)*lcm,
                      lcm*n[i]/d);
      lcm = LCM(lcm, n[i]);
    }
    cout << ans << " " << lcm << endl;
}
```

4.4 DiaphantineEquation

```
int gcd(int a, int b, int &x, int &y) {
    if (a == 0) {
        x = 0; y = 1;
        return b;
    }
    int x1, y1;
    int d = gcd(b%a, a, x1, y1);
    x = y1 - (b / a) * x1;
    y = x1;
    return d;
}
bool find_any_solution(int a, int b, int c,
                       int &x0, int &y0, int &g){
    g = gcd(abs(a), abs(b), x0, y0);
    if (c % g) return false;
    x0 *= c / g; y0 *= c / g;
    if (a < 0) x0 = -x0;
    if (b < 0) y0 = -y0;
    return true;
}
void shift_solution(int & x, int & y, int a, int b, int cnt){
    x += cnt * b; y -= cnt * a;
}
int find_all_solutions (int a, int b, int c,
                        int minx, int maxx, int miny, int maxy) {
    int x, y, g;
    if (! find_any_solution (a, b, c, x, y, g)) return 0;
    a /= g; b /= g;
    int sign_a = a>0 ? +1 : -1, sign_b = b>0 ? +1 : -1;
    shift_solution (x, y, a, b, (minx - x) / b);
    if (x < minx) shift_solution (x, y, a, b, sign_b);
    if (x > maxx) return 0;
    int lx1 = x;
    shift_solution (x, y, a, b, (maxx - x) / b);
    if (x > maxx) shift_solution (x, y, a, b, -sign_b);
    int rx1 = x;
    shift_solution (x, y, a, b, - (miny - y) / a);
    if (y < miny) shift_solution (x, y, a, b, -sign_a);
    if (y > maxy) return 0;
    int lx2 = x;
    shift_solution (x, y, a, b, - (maxy - y) / a);
    if (y > maxy) shift_solution (x, y, a, b, sign_a);
    int rx2 = x;
    if (lx2 > rx2) swap (lx2, rx2);
    int lx = max (lx1, lx2), rx = min (rx1, rx2);
    if (lx > rx) return 0;
    int sol = (rx - lx) / abs(b) + 1;
    for( int i = 0; i < sol; i++ )
      cout<<lx+i*abs(b)<<endl;
    return sol;
}
int main()
{
  cout << find_all_solutions(2,-3,-6,-10, 10, -10,10)<<endl;
}
```

4.5 Gaussian Elimination Extended

```
#include <bits/stdtr1c++.h>

#define EPS 1e-9
#define MAXROW 512
#define MAXCOL 512
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
#define dbg(x) cout << #x << " = " << x << endl
#define ran(a, b) ((((rand() << 15) ^ rand()) % ((b) - (a) + 1))
    + (a))

using namespace std;

/***

Gauss-Jordan Elimination

n = number of linear equations
m = number of variables
ar[i][m] = right-hand side value of constants

For instance, the system of linear equations becomes:

2x + y -z = 8    -----> (i)
-3x -y + 2z = -11 -----> (ii)
-2x + y + 2z = -3 -----> (iii)

n = 3 (x, y, z), m = 3 (i, ii, iii)
ar[0] = {2, 1, -1, 8} -----> (i)
ar[1] = {-3, -1, 2, -11} -----> (ii)
ar[2] = {-2, 1, 2, -3} -----> (iii)


Returns -1 when there is no solution
Otherwise returns the number of independent variables (0 for an
    unique solution)
Contains a solution in the vector res on successful completion
Note that the array is modified in the process

Notes:
For solving problems on graphs with probability/expectation,
    make sure the graph
is connected and a single component. If not, then re-number the
    vertex and solve
for each connected component separately.

***/

int gauss(int n, int m, double ar[MAXROW][MAXCOL],
    vector<double>& res){ /// hash = 835176
    res.assign(m, 0);
    vector <int> pos(m, -1);
    int i, j, k, l, p, free_var = 0;

    for (j = 0, i = 0; j < m && i < n; j++){
        for (k = i, p = i; k < n; k++){
            if (abs(ar[k][j]) > abs(ar[p][j])) p = k;
        }

        if (abs(ar[p][j]) > EPS){
            pos[j] = i;
            for (l = j; l <= m; l++) swap(ar[p][l], ar[i][l]);

            for (k = 0; k < n; k++){
                if (k != i){
                    double x = ar[k][j] / ar[i][j];
                    for (l = j; l <= m; l++) ar[k][l] -= (ar[i][l]
                        * x);
                }
```

```
            }
            i++;
        }
    }

    for (i = 0; i < m; i++){
        if (pos[i] == -1) free_var++;
        else res[i] = ar[pos[i]][m] / ar[pos[i]][i];
    }

    for (i = 0; i < n; i++) {
        double val = 0.0;
        for (j = 0; j < m; j++) val += (res[j] * ar[i][j]);
        if (abs(val - ar[i][m]) > EPS) return -1;
    }

    return free_var;
}

int main(){

}
```

4.6 Gaussian Elimination in GF(2)

```
#include <bits/stdtr1c++.h>

#define MAXROW 630
#define MAXCOL 630
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
#define dbg(x) cout << #x << " = " << x << endl
#define ran(a, b) ((((rand() << 15) ^ rand()) % ((b) - (a) + 1))
    + (a))

using namespace std;

/***

Gauss-Jordan Elimination in Galois Field, GF(2)

n = number of linear equations
m = number of variables
ar[i][m] = right-hand side value of constants

For instance, the system of linear equations (note not in GF(2))
    becomes:

2x + y -z = 8    -----> (i)
-3x -y + 2z = -11 -----> (ii)
-2x + y + 2z = -3 -----> (iii)

n = 3 (x, y, z), m = 3 (i, ii, iii)
ar[0] = {2, 1, -1, 8} -----> (i)
ar[1] = {-3, -1, 2, -11} -----> (ii)
ar[2] = {-2, 1, 2, -3} -----> (iii)


Returns -1 when there is no solution
Otherwise returns the number of independent variables (0 for an
    unique solution)
Contains a solution in the bit vector res on successful
    completion
Note that the array is modified in the process

***/

int gauss(int n, int m, bitset <MAXCOL> ar[MAXROW], bitset
    <MAXCOL>& res){ /// hash = 169721
```

```
    res.reset();
    vector <int> pos(m, -1);
    int i, j, k, l, v, p, free_var = 0;

    for (j = 0, i = 0; j < m && i < n; j++){
        for (k = i, p = i; k < n; k++){
            if (ar[k][j]){
                p = k;
                break;
            }
        }

        if (ar[p][j]){
            pos[j] = i;
            swap(ar[p], ar[i]);

            for (k = 0; k < n; k++){
                if (k != i && ar[k][j]) ar[k] ^= ar[i];
            }
            i++;
        }
    }

    for (i = 0; i < m; i++){
        if (pos[i] == -1) free_var++;
        else res[i] = ar[pos[i]][m];
    }

    for (i = 0; i < n; i++) {
        for (j = 0, v = 0; j < m; j++) v ^= (res[j] & ar[i][j]);
        if (v != ar[i][m]) return -1;
    }
    return free_var;
}
```

```
int main(){

}
```

4.7 Gaussian Elimination in Modular Field

```
#include <bits/stdtr1c++.h>

#define MAXROW 1010
#define MAXCOL 1010
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
#define dbg(x) cout << #x << " = " << x << endl
#define ran(a, b) ((((rand() << 15) ^ rand()) % ((b) - (a) + 1))
    + (a))

using namespace std;

int expo(int a, int b, int MOD){
    int res = 1;

    while (b){
        if (b & 1) res = (long long)res * a % MOD;
        a = (long long)a * a % MOD;
        b >>= 1;
    }
    return res;
}

/// Gaussian elimination in field MOD (MOD should be a prime)
int gauss(int n, int m, int MOD, int ar[MAXROW][MAXCOL],
    vector<int>& res){
    res.assign(m, 0);
    vector <int> pos(m, -1);
```

```
int i, j, k, l, p, d, free_var = 0;
const long long MODSQ = (long long)MOD * MOD;

for (j = 0, i = 0; j < m && i < n; j++){
    for (k = i, p = i; k < n; k++){
        if (ar[k][j] > ar[p][j]) p = k;
    }

    if (ar[p][j]){
        pos[j] = i;
        for (l = j; l <= m; l++) swap(ar[p][l], ar[i][l]);

        d = expo(ar[i][j], MOD - 2, MOD);
        for (k = 0; k < n && d; k++){
            if (k != i && ar[k][j]){
                int x = ((long long)ar[k][j] * d) % MOD;
                for (l = j; l <= m && x; l++){
                    if (ar[i][l]) ar[k][l] = (MODSQ + ar[k][l]
                        - ((long long)ar[i][l] * x)) % MOD;
                }
            }
        }
        i++;
    }
}

for (i = 0; i < m; i++){
    if (pos[i] == -1) free_var++;
    else res[i] = ((long long)ar[pos[i]][m] *
        expo(ar[pos[i]][i], MOD - 2, MOD)) % MOD;
}

for (i = 0; i < n; i++) {
    long long val = 0;
        for (j = 0; j < m; j++) val = (val + ((long long)res[j] *
            ar[i][j])) % MOD;
        if (val != ar[i][m]) return -1;
    }
    return free_var;
}

int main(){

}
```

4.8 NT

```
#include <bits/stdc++.h>
using namespace std;

typedef long long ll;

const int N = 1e6 + 100;
int mod = 1e9 + 7;


namespace NT {
    int fact[N], inv[N];

    int bm(int b, int p, int m) {
        if(p == 0) return 1%m;
        int t = bm(b,p/2,m);
        t = (1ll*t*t)%m;
        if(p&1) return 1ll*t*b%m;
        return t;
    }

    int C(int n, int r) {
```

```
        if(n < 0 or r < 0 or r > n) return 0;
        int ret = 1ll*fact[n]*inv[r]%mod;
        ret = 1ll*ret*inv[n-r]%mod;
        return ret;
    }

    void init() {
        fact[0] = 1;
        for(int i = 1; i < N; i++) {
            fact[i] = 1ll*fact[i-1]*i%mod;
        }
        inv[N-1] = bm(fact[N-1], mod-2, mod);
        for(int i = N-2; i >= 0; i--) {
            inv[i] = 1ll*inv[i+1]*(i+1)%mod;
        }
    }

    bool composite[N];
    vector<int> prime;

    void sieve() {
        composite[1] = 1;
        for(int i = 2; i < N; i++) {
            if(!composite[i]) prime.push_back(i);
            for(int j = 0; j < prime.size() && prime[j]*i < N;
                j++) {
                composite[prime[j] * i] = 1;
                if(i%prime[j] == 0) break;
            }
        }
    }
}


int ans[N];
```

```
int main() {
    ios::sync_with_stdio(0);
    cin.tie(0);
    NT::init();
    return 0;
}
```

4.9 NTT

```
struct NTT {
    vector<int>A, B, w[2], rev;
    int P, M, G;
    NTT( int mod ) {
        P = mod; G = 3;
    }
    int Pow(int a, int b) {
        int res=1;
        for (;b; b>>=1,a=a*1LL*a%P) if (b&1) res=res*1LL*a%P;
        return res;
    }
    void init( int n ) {
        for (M=1; M<n; M<<=1);
        M<<=1;
        A.resize(M); B.resize(M);
        w[0].resize(M); w[1].resize(M);
        rev.resize(M);
        for (int i=0; i<M; i++) {
            int x=i, &y=rev[i];
            y=0;
            for (int k=1; k<M; k<<=1,x>>=1)
                (y<<=1)|=x&1;
        }
        int x=Pow(G,(P-1)/M),y=Pow(x,P-2);
```

```
    w[0][0]=w[1][0]=1;
    for (int i=1; i<M; i++) {
        w[0][i]=w[0][i-1]*1LL*x%P;
        w[1][i]=w[1][i-1]*1LL*y%P;
    }
}
void ntransform(vector<int> &a, int f) {
    for (int i=0; i<M; i++)
        if (i<rev[i]) swap(a[i],a[rev[i]]);
    for (int i=1; i<M; i<<=1)
        for (int j=0,t=M/(i<<1); j<M; j+=i<<1)
            for (int k=0,l=0; k<i; k++,l+=t) {
                int x=a[j+k+i]*1ll*w[f][l]%P;
                int y=a[j+k];
                a[j+k+i]=y-x<0?y-x+P:y-x;
                a[j+k]=y+x>=P?y+x-P:y+x;
            }
    if (f) {
        int x=Pow(M,P-2);
        for (int i=0; i<M; i++) a[i]=a[i]*1ll*x%P;
    }
}

void multiply( vector<int> &X, vector<int> &Y,
               vector<int> &res) {
    init(max(X.size(), Y.size()));
    for( int i = 0; i < M; i++ ) A[i] = B[i] = 0;
    for( int i = 0; i < X.size(); i++ ) A[i] = X[i];
    for( int i = 0; i < Y.size(); i++ ) B[i] = Y[i];
    ntransform(A,0); ntransform(B,0);
    res.clear(); res.resize(M);
    for (int i=0; i<M; i++)
        res[i]=A[i]*1LL*B[i]%P;
    ntransform(res,1);
}
```

```
};
int main() {
    NTT ntt(998244353);
    vector<int>A{0, 2, 0, 1, 2}, B{1, 0, 0, 0, -1, 0, -5};
    ntt.multiply(A,B,A); //A = A*B
}
```

4.10 Pollard Rho (kundu)

```
/**
Range: 10^18 (tested), should be okay up to 2^63-1
miller_rabin(n)
    returns 1 if prime, 0 otherwise
    Magic bases:
        n < 4,759,123,141      3 :  2, 7, 61
        n < 1,122,004,669,633  4 :  2, 13, 23, 1662803
        n < 3,474,749,660,383  6 :  2, 3, 5, 7, 11, 13
        n < 2^64               7 : 2, 325, 9375, 28178, 450775,
            9780504, 1795265022
    Identifies 70000 18 digit primes in 1 second on Toph
pollard_rho(n):
    If n is prime, returns n
    Otherwise returns a proper divisor of n
    Able to factorize ~120 18 digit semiprimes in 1 second on
        Toph
    Able to factorize ~700 15 digit semiprimes in 1 second on
        Toph
Note: for factorizing large number, do trial division upto
      cubic root and then call pollard rho once.
*/


#include<bits/stdc++.h>
#define LL long long
```

```
using namespace std;

LL mult(LL a, LL b, LL mod) {
    assert(b < mod && a < mod);
    long double x = a;
    uint64_t c = x * b / mod;
    int64_t r = (int64_t)(a * b - c * mod) % (int64_t)mod;
    return r < 0 ? r + mod : r;
}

LL power(LL x, LL p, LL mod){
        LL s=1, m=x;
        while(p) {
                if(p&1) s = mult(s, m, mod);
                p>>=1;
                m = mult(m, m, mod);
        }
        return s;
}

bool witness(LL a, LL n, LL u, int t){
        LL x = power(a,u,n);
        for(int i=0; i<t; i++) {
                LL nx = mult(x, x, n);
                if (nx==1 && x!=1 && x!=n-1) return 1;
                x = nx;
        }
        return x!=1;
}

vector<LL> bases = {2, 325, 9375, 28178, 450775, 9780504,
    1795265022};
bool miller_rabin(LL n) {
        if (n<2)   return 0;
        if (n%2==0)  return n==2;
        LL u = n-1;
        int t = 0;
        while(u%2==0)  u/=2, t++; // n-1 = u*2^t

        for (LL v: bases) {
                LL a = v%(n-1) + 1;
                if(witness(a, n, u, t)) return 0;
    }
        return 1;
}

LL gcd(LL u, LL v) {
    if (u == 0) return v;
    if (v == 0) return u;
    int shift = __builtin_ctzll(u | v);
    u >>= __builtin_ctzll(u);
    do {
        v >>= __builtin_ctz(v);
        if (u > v) swap(u, v);
        v = v - u;
    } while (v);
    return u << shift;
}

mt19937_64
    rng(chrono::steady_clock::now().time_since_epoch().count());
LL pollard_rho(LL n) {
    if (n==1)              return 1;
    if (n%2==0)            return 2;
        if (miller_rabin(n))  return n;

    while (true) {
        LL x = uniform_int_distribution<LL>(1, n-1)(rng);
        LL y = 2, res = 1;
```

```
        for (int sz=2; res == 1; sz*=2) {
            for (int i=0; i<sz && res<=1; i++) {
                x = mult(x, x, n) + 1;
                res = gcd(abs(x-y), n);
            }
            y = x;
        }
        if (res!=0 && res!=n) return res;
    }
}


///Solves UVA - 11476
const int MX = 2.2e5+7;
vector<int> primes;
bool isp[MX];

void sieve() {
    fill(isp+2, isp+MX, 1);
    for (int i=2; i<MX; i++)
        if (isp[i]) {
            primes.push_back(i);
            for (int j=2*i; j<MX; j+=i)
                isp[j] = 0;
        }
}

vector<LL> factorize(LL x) {
    vector<LL> ans;
    for (int p: primes) {
        if (1LL*p*p*p > x) break;
        while (x%p==0) {
            x/=p;
            ans.push_back(p);
        }
    }
    if (x > 1) {
        LL z = pollard_rho(x);
        ans.push_back(z);
        if (z < x) ans.push_back(x/z);
    }
    return ans;
}

int main()
{
//   freopen("in.txt", "r", stdin);
//   freopen("out-mine.txt", "w", stdout);
    sieve();
    int t;
    cin>>t;

    while (t--) {
        long long x;
        if (!(cin>>x))  break;
        vector<LL> ans = factorize(x);
        sort(ans.begin(), ans.end());

        vector<pair<LL, int>> ff;
        for (LL x: ans) {
            if (ff.size() && ff.back().first == x)
                ff.back().second++;
            else    ff.push_back({x, 1});
        }
        cout<<x<<" =";
        bool first = true;
        for (auto pr: ff) {
            if (!first) cout<<" *";
            first = false;
            cout<<" "<<pr.first;
```

```
            if (pr.second > 1)cout<<"^"<<pr.second;
        }
        cout<<endl;
    }
}
```

4.11 PolynomialInterpolation

```
/*given n points (x0, y0), (x1, y1) ... (xn, yn) find f(x):
  f(x) = a0 + a1(x-x0)(x-x1) + a2(x-x0)(x-x1)(x-x2) + ....
         an(x-x0)(x-x1)(x-x2)...(x-x{n-1})
  now define p[xk] = yk;
  p[x{k-1},xk] = (p[xk] - p[x{k-1}])/(xk-x{k-1})
  p[x{k-2},x{k-1},xk ] = (p[x{k-1}, xk]-p[x{k-2}, x{k-2}])/
                         (xk - x{k-2})
  so f(x) = p[x0] + p[x0,x1](x-x0)(x-x1) + p[x0,x1,x2]*
            (x-x0)(x-x1)(x-x2)+..+p[x0,x1....xn]*(...) */
```

4.12 PolynomialRoots

```
double cal(const vector<double> &coef, double x) {
  double e = 1, s = 0;
  for (int i = 0; i < coef.size(); ++i) s += coef[i]*e,e*=x;
  return s;
}

double find(const vector<double> &coef,
            double l, double r, int sl, int sr) {
  int sl = dblcmp(cal(coef, l)), sr = dblcmp(cal(coef, r));
  if (sl == 0) return l;
  if (sr == 0) return r;
  for (int tt = 0; tt < 100 && r - l > eps; ++tt) {
    double mid = (l + r) / 2;
    int smid = dblcmp(cal(coef, mid));
    if (smid == 0) return mid;
    if (sl * smid < 0) r = mid;
    else l = mid;
  }
  return (l + r) / 2;
}

vector<double> rec(const vector<double>&coef,int n){//c[n]==1
  vector<double> ret;
  if (n == 1) {
    ret.push_back(-coef[0]);
    return ret;
  }
  vector<double> dcoef(n);
  for (int i = 0; i < n; ++i) dcoef[i]=coef[i+1]*(i+1)/n;
  double b = 2;
  for (int i = 0; i <= n; ++i)
    b = max(b, 2 * pow(fabs(coef[i]), 1.0 / (n - i)));
  vector<double> droot = rec(dcoef, n - 1);
  droot.insert(droot.begin(), -b);
  droot.push_back(b);
  for (int i = 0; i + 1 < droot.size(); ++i) {
    int sl = dblcmp(cal(coef, droot[i])),
        sr = dblcmp(cal(coef, droot[i + 1]));
    if (sl * sr > 0) continue;
    ret.push_back(find(coef,droot[i],droot[i+1], sl, sr));
  }
  return ret;
}

// solve c[0]+c[1]*x+c[2]*x^2+...+c[n]*x^n==0
vector<double> solve(vector<double> coef) {
  int n = coef.size() - 1;
```

```
  while (coef.back() == 0) coef.pop_back(), --n;
  for (int i = 0; i <= n; ++i) coef[i] /= coef[n];
  return rec(coef, n);
}
```

4.13 Shanks

```
#include <bits/stdc++.h>
using namespace std ;
const long long N = 105 , md = 100000007 , INF = md ;

long long bigmod (long long b ,long long p , long long M) {
    if (p == 0) return 1 ;
    if (p == 1) return b%M ;
    if (p&1) return (b*bigmod(b,p-1,M))%M ;
    long long t = bigmod(b,p/2,M) ;
    return (t*t)%M ;
}

long long Shanks(long long a , long long b , long long M) { ///
    a^x = b (mod M)
    map <long long,long long> vals ;
    long long n = (long long)sqrt(M+.0) + 1 ;
    for (long long p = n ; p >= 1 ; p--) {
        vals[bigmod(a,(n*p)%M,M)] = p ;
    }
    long long ans = INF ;
    for (long long q = 0 ; q <= n ; q++) {
        long long cur = b*bigmod(a,q,M) ;
        cur %= M ;
        if (vals[cur]) {
            ans = min(ans,vals[cur]*n-q) ;
        }
    }
    return ans ;
}
```

4.14 Sieve factorize

```
#include <bits/stdc++.h>
using namespace std;
const int N = 1e5 + 5;
typedef vector <pair<int,int>> vii;

int np[N], phi[N];
vector<int> primes;

void sieve(int n) {
    for (int i = 1; i <= n; i++) phi[i] = i;
    np[1] = 1;
    for (int i = 2; i <= n; i++) {
        if (np[i]) continue;
        primes.push_back(i);
        phi[i] = i-1;
        for (int j = 2*i; j <= n; j+=i) {
            np[j] = 1;
            phi[j] = (phi[j]/i)*(i-1);
        }
    }
}

vii factorize (int x) {
    vii v;
    for (int i = 0; primes[i]*primes[i] <= x; i++) {
        int k = 0;
        while (x%primes[i] == 0) {
            k++;
            x /= primes[i];
```

```
        }
        if (k) v.push_back(make_pair(primes[i],k));
    }
    if (x > 1) v.push_back(make_pair(x,1));
    return v;
}

int main () {
    int n = 1e5;
    sieve(n);
}
```

4.15 combinatorics

```
#include <bits/stdc++.h>
using namespace std;

typedef long long ll;

const int N = 2e6 + 100;
int mod = 1e9 + 7;


namespace Combi {
    int fact[N], inv[N];

    int bm(int b, int p, int m) {
        if(p == 0) return 1%m;
        int t = bm(b,p/2,m);
        t = (1ll*t*t)%m;
        if(p&1) return 1ll*t*b%m;
        return t;
    }
    int C(int n, int r) {
        if(n < 0 or r < 0 or r > n) return 0;
        int ret = 1ll*fact[n]*inv[r]%mod;
        ret = 1ll*ret*inv[n-r]%mod;
        return ret;
    }
    // X1 + X2 + ... + Xvar = Sum
    int no_of_eqns(int var, int sum) {
        return C(sum+var-1,var-1); // Xi >= 0
        // return C(sum-1,var-1); // Xi > 0
    }
    void init() {
        fact[0] = 1;
        for(int i = 1; i < N; i++) {
            fact[i] = 1ll*fact[i-1]*i%mod;
        }
        inv[N-1] = bm(fact[N-1], mod-2, mod);
        for(int i = N-2; i >= 0; i--) {
            inv[i] = 1ll*inv[i+1]*(i+1)%mod;
        }
    }
}

int main() {
    ios::sync_with_stdio(0);
    cin.tie(0);
    Combi::init();

    return 0;
}
```

4.16 congruence

```
/// This is a collection of useful code for solving problems that
```

```
/// involve modular linear equations. Note that all of the
/// algorithms described here work on NON-NEGATIVE INTEGERS.
/// Source: Stanford Notebook (modified)

#include <bits/stdc++.h>
#define LL long long
using namespace std;
typedef pair<LL, LL> PLL;


/// Computes gcd(a,b)
/// Range: LL

LL gcd(LL u, LL v) {
    if (u == 0) return v;
    if (v == 0) return u;
    int shift = __builtin_ctzll(u | v);
    u >>= __builtin_ctzll(u);
    do {
        v >>= __builtin_ctzll(v);
        if (u > v) swap(u, v);
        v = v - u;
    } while (v);
    return u << shift;
}

/// computes lcm(a,b)
/// Range: int
LL lcm(LL a, LL b) {
        return (a/gcd(a, b))*b;
}

/// (a^b) mod m via successive squaring
/// Range: int
LL power(LL a, LL b, LL m) {
    a = (a%m+m)%m;
        LL ans = 1;
        while (b) {
                if (b & 1) ans = (ans*a)%m;
                a = (a*a)%m;
                b >>= 1;
        }
        return ans;
}

/// returns g = gcd(a, b); finds x, y such that d = ax + by
/// Range: int (tested on CF 982E :( )
LL egcd(LL a, LL b, LL &x, LL &y) {
        LL xx = y = 0;
        LL yy = x = 1;
        while (b) {
                LL q = a/b;
                LL t = b; b = a%b; a = t;
                t = xx; xx = x-q*xx; x = t;
                t = yy; yy = y-q*yy; y = t;
        }
        return a;
}

/// finds all solutions to ax = b (mod m)
/// Range: int (not tested)
vector<LL> SolveCongruence(LL a, LL b, LL m) {
        LL x, y;
        vector<LL> ans;
        LL g = egcd(a, m, x, y);
        if (b%g == 0) {
                x = (x*(b/g))%m;
                if (x<0)   x+=m;
                for (LL i=0; i<g; i++) {
                        ans.push_back(x);
```

```
                        x = (x+m/g)%m;
                }
        }
        return ans;
}

/// Computes b such that ab = 1 (mod m), returns -1 on failure
/// Range: int
LL inverse(LL a, LL m) {
        LL x, y;
        LL g = egcd(a, m, x, y);
        if (g > 1) return -1;
        return (x%m+m)%m;
}

/// Chinese remainder theorem (special case):
/// find z such that z % m1 = r1, z % m2 = r2.
/// Here, z is unique modulo M = lcm(m1, m2).
/// Return (z, M). On failure, M = -1.
/// Range: int (tested on CF 982E :( )
PLL CRT(LL m1, LL r1, LL m2, LL r2) {
        LL s, t;
        LL g = egcd(m1, m2, s, t);
        if (r1%g != r2%g) return PLL(0, -1);
        LL M = m1*m2;
        LL ss = ((s*r2)%m2)*m1;
        LL tt = ((t*r1)%m1)*m2;
        LL ans = ((ss+tt)%M+M)%M;
        return PLL(ans/g, M/g);
}


/// Chinese remainder theorem:
/// find z such that z % m[i] = r[i] for all i.
/// The solution is unique modulo M = lcm(m[i]).
/// Return (z, M). On failure, M = -1.
/// Note that we do not require the mod values to be co-prime.
/// Range: int (if LCM fits in LL)
PLL CRT(const vector<LL> &m, const vector<LL> &r) {
        PLL ans = PLL(r[0], m[0]);
        for (LL i = 1; i < m.size(); i++) {
                ans = CRT(ans.second, ans.first, m[i], r[i]);
                if (ans.second == -1) break;
        }
        return ans;
}

/// computes x and y such that ax + by = c
/// returns whether the solution exists
/// Range: int
bool LinearDiophantine(LL a, LL b, LL c, LL &x, LL &y) {
        if (!a && !b) {
                if (c) return false;
                x = y = 0;
                return true;
        }
        if (!a) {
                if (c%b) return false;
                x = 0; y = c/b;
                return true;
        }
        if (!b) {
                if (c%a) return false;
                x = c/a; y = 0;
                return true;
        }
        LL g = gcd(a, b);
        if (c%g) return false;
        x = c/g * inverse(a/g, b/g);
        y = (c-a*x)/b;
```

```
        return true;
}

int main() {
        // expected: 2
        cout << gcd(14, 30) << endl;

        // expected: 2 -2 1
        LL x, y;
        LL g = egcd(14, 30, x, y);
        cout << g << " " << x << " " << y << endl;

        // expected: 95 45
    vector<long long> sols = SolveCongruence(14, 30, 100);
        for (LL i = 0; i < sols.size(); i++) cout << sols[i] << "
            ";
        cout << endl;

        // expected: 8
        cout << inverse(8, 9) << endl;

        // expected: 23 105
        //           11 12
        PLL ans = CRT({3,5,7}, {2,3,2});
        cout << ans.first << " " << ans.second << endl;
        ans = CRT({4,6}, {3,5});
        cout << ans.first << " " << ans.second << endl;

        // expected: 5 -15
        if (!LinearDiophantine(7, 2, 5, x, y)) cout << "ERROR" <<
            endl;
        else cout << x << " " << y << endl;
        return 0;
}
```

4.17 discreteRoot

```
// This program finds all numbers x such that x^k = a (mod n)
//powmod(a,b,p) returns (a^b)%p
// Finds the primitive root modulo p
int generator(int p) {
    vector<int> fact;
    int phi = p-1, n = phi;
    for (int i = 2; i * i <= n; ++i) {
        if (n % i == 0) {
            fact.push_back(i);
            while (n % i == 0) n /= i;
        }
    }
    if (n > 1) fact.push_back(n);
    for (int res = 2; res <= p; ++res) {
        bool ok = true;
        for (int factor : fact) {
            if (powmod(res, phi / factor, p) == 1) {
                ok = false; break;
            }
        }
        if (ok) return res;
    }
    return -1;
}

void solve( int n, int k, int a ) {
    int g = generator(n); //g is a primitive root
    // (g^y)^k = a mod n -> (g^k)^y = a mod n; now find y
    // Baby-step giant-step discrete logarithm algorithm
    // a^(n*sq-q) = b mod n
    int sq = (int) sqrt (n + .0) + 1;
    vector<pair<int, int>> dec(sq);
    for (int i = 1; i <= sq; ++i)
```

```
        dec[i-1] = {powmod(g, (i*sq*k) % (n - 1), n), i};
    sort(dec.begin(), dec.end()); int any_ans = -1;
    for (int i = 0; i < sq; ++i) {
        int my = powmod(g, i * k % (n - 1), n) * a % n;
        auto it = lower_bound(dec.begin(), dec.end(),
                              make_pair(my, 0));
        if (it != dec.end() && it->first == my) {
            any_ans = it->second * sq - i;
            break;
        }
    }
    if (any_ans == -1){ puts("0"); return 0; }
    // Print all possible answers
    int delta = (n-1) / gcd(k, n-1); vector<int> ans;
    for (int cur = any_ans % delta; cur < n-1; cur += delta)
        ans.push_back(powmod(g, cur, n));
    sort(ans.begin(), ans.end()); printf("%d\n", ans.size());
    for (int answer : ans) printf("%d ", answer);
}

int main() {
    int n, k, a;
    scanf("%d %d %d", &n, &k, &a);
    if (a == 0) puts("1\n0"), return 0;
    solve( n, k, a );
}
```

4.18 fft

```
/// FFT
struct FFT {
    struct node {
        double x,y;
        node() {}
        node(double a, double b): x(a), y(b) {}
        node operator + (const node &a) const {return
            node(this->x+a.x,this->y+a.y);}
        node operator - (const node a) const {return
            node(this->x-a.x,this->y-a.y);}
        node operator * (const node a) const {return
            node(this->x*a.x-this->y*a.y,this->x*a.y+a.x*this->y);}
    };

    int M;
    vector<node> A, B, w[2];
    vector<int>rev;
    long double pi;
    FFT() {
        pi = 3.1415926535897932384;
    }
    void init(int n) {
        M = 1;
        while(M < n) M <<= 1;
        M <<= 1;
        A.resize(M);
        B.resize(M);
        w[0].resize(M);
        w[1].resize(M);
        rev.resize(M);
        for (int i=0; i<M; i++) {
            int j=i,y=0;
            for (int x=1; x<M; x<<=1,j>>=1) (y<<=1)+=j&1;
            rev[i]=y;
        }
        for (int i=0; i<M; i++) {
            w[0][i] = node( cos(2*pi*i/M),sin(2*pi*i/M) );
            w[1][i] = node( cos(2*pi*i/M),-sin(2*pi*i/M) );
        }
    }
```

```
void ftransform( vector<node> &A, int p ) {
    for (int i=0; i<M; i++)
        if (i<rev[i])
            swap(A[i],A[rev[i]]);
    for (int i=1; i<M; i<<=1)
        for (int j=0,t=M/(i<<1); j<M; j+=i<<1)
            for (int k=0,l=0; k<i; k++,l+=t) {
                node x=w[p][l]*A[i+j+k];
                node y=A[j+k];
                A[j+k]=y+x;
                A[j+k+i]=y-x;
            }
    if (p)
        for (int i=0; i<M; i++)
            A[i].x/=M;
}
/// multiply P*Q and keeps the result in res
///degree of P is n and degree of Q is m
///P, Q is given in standard power form, in increasing
void multiply( vector<int> &P, vector<int> &Q, vector<int>
    &res) {
    init( max(P.size(),Q.size()) );
    for( int i = 0; i < M; i++ )
        A[i].x = A[i].y = B[i].x = B[i].y = 0;
    for( int i = 0; i < P.size(); i++ )
        A[i].x = P[i];
    for( int i = 0; i < Q.size(); i++ )
        B[i].x = Q[i];
    ftransform(A,0);
    ftransform(B,0);
    for (int k=0; k<M; k++)
        A[k] = A[k]*B[k];
    ftransform(A,1);
    res.resize(M);
    for( int i = 0; i < M; i++ )
            res[i] = round(A[i].x);
    }
};

int main() {

    return 0;
}
```

4.19 floor$_sum$

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <algorithm>
// Problem       : Mod Mod Mod (NAIPC 2019)
// Author        : Darcy Best
// Expected Result : AC
// Complexity    : O(log(p+q)) per test case

// I will instead compute:
//  (p + 2p + ... + np) - q * ([p/q] + [2p/q] + ... + [np/q])

#include <iostream>
using namespace std;

// Computes [p/q] + [2p/q] + ... + [np/q]
long long sum_of_floors(long long p,long long q,long long n){
  if(p == 0 || n == 0) return 0;

  if(n >= q)
    return p*(n/q)*(n+1) - (n/q)*((n/q)*p*q + p + q - 1)/2 +
        sum_of_floors(p,q,n%q);
```

```
  if(p >= q)
    return (p/q)*n*(n+1)/2 + sum_of_floors(p%q,q,n);

  return (n*p/q) * n - sum_of_floors(q,p,n*p/q);
}

long long gcd(long long a, long long b){
  return b == 0 ? a : gcd(b, a % b);
}

int main(){
  int C; cin >> C;

  while(C--){
    long long p,q,n; cin >> p >> q >> n;
    long long g = gcd(p,q);

    cout << (p * n * (n+1) / 2) - q * sum_of_floors(p/g, q/g, n)
        << endl;
  }
}
```

4.20 joseph

```
int joseph (int n , int k) {
    if (n==1) return 0 ;
    return (joseph(n-1,k) + k )%n ;
}
```

4.21 number theory

```
#include <bits/stdc++.h>
using namespace std;

typedef long long ll;

const int N = 1e6 + 100;
int mod = 1e9 + 7;


namespace NT {
    int fact[N], inv[N];

    int bm(int b, int p, int m) {
        if(p == 0) return 1%m;
        int t = bm(b,p/2,m);
        t = (1ll*t*t)%m;
        if(p&1) return 1ll*t*b%m;
        return t;
    }

    int C(int n, int r) {
        if(n < 0 or r < 0 or r > n) return 0;
        int ret = 1ll*fact[n]*inv[r]%mod;
        ret = 1ll*ret*inv[n-r]%mod;
        return ret;
    }

    void init() {
        fact[0] = 1;
        for(int i = 1; i < N; i++) {
            fact[i] = 1ll*fact[i-1]*i%mod;
        }
        inv[N-1] = bm(fact[N-1], mod-2, mod);
        for(int i = N-2; i >= 0; i--) {
            inv[i] = 1ll*inv[i+1]*(i+1)%mod;
        }
```

```
    }

    bool composite[N];
    vector<int> prime;

    void sieve() {
        composite[1] = 1;
        for(int i = 2; i < N; i++) {
            if(!composite[i]) prime.push_back(i);
            for(int j = 0; j < prime.size() && prime[j]*i < N;
                j++) {
                composite[prime[j] * i] = 1;
                if(i%prime[j] == 0) break;
            }
        }
    }
}


int ans[N];

int main() {
    ios::sync_with_stdio(0);
    cin.tie(0);
    NT::init();
    return 0;
}
```

4.22 pollard Rho

```
#include <bits/stdc++.h>
using namespace std;

bool mark[50000];
long long prime[10000], Pt = 0;
const long long LIM = LLONG_MAX, mod = 1e9 + 7;

void sieve() {
    prime[Pt++] = 2;
    int n = 47000;
    for(long long i = 3; i <= n; i++) {
        if(mark[i] == 0) {
            prime[Pt++] = i;
            for(long long j = i * i; j <= n; j += i) {
                mark[j] = 1;
            }
        }
    }
}

long long mul(long long a, long long b, long long m){
    long long x, res;
    if (a < b) swap(a, b);
    if (!b) return 0;
    if (a < (LIM / b)) return ((a * b) % m);
    res = 0, x = (a % m);
    while (b){
        if (b & 1){
            res = res + x;
            if (res >= m) res -= m;
        }
        b >>= 1;
        x <<= 1;
        if (x >= m) x -= m;
    }
    return res;
}

long long expo(long long x, long long n, long long m){
```

```
    long long res = 1;

    while (n){
        if (n & 1) res = mul(res, x, m);
        x = mul(x, x, m);
        n >>= 1;
    }

    return (res % m);
}

bool isPrime(long long n) {
    if (n == 2) return 1;
    if (n%2 == 0) return 0;
    long long d = n-1;
    while(d%2 == 0) d >>= 1;
    int test[] = {2,3,5,7,11,13,17,19,23};
    for(int i = 0; i < 9; i++) {
        long long x = test[i]%(n-2), temp = d;
        if (x < 2) x += 2;
        long long a = expo(x, d, n);
        while(temp != n-1 && a != 1 && a != n-1) {
            a = mul(a, a, n);
            temp <<= 1;
        }
        if (a != n-1 && (temp&1) == 0) return 0;
    }
    return 1;
}

long long pollard_rho(long long n, long long c) {
    long long x = 2, y = 2, i = 1, k = 2, d;
    while(true) {
        x = (mul(x, x, n) + c);
        if (x >= n) x -= n;
        d = __gcd(abs(x-y),n);
        if (d > 1) return d;
        if (++i == k) {
            y = x, k <<= 1;
        }
    }
    return n;
}

void llfactorize(long long n, vector<long long> &f) {
    if (n == 1) return;
    if (n < 1e9) {
        for(int i = 0; prime[i]* prime[i] <= n; i++) {
            while(n%prime[i] == 0) {
                f.push_back(prime[i]);
                n /= prime[i];
            }
        }
        if (n != 1) f.push_back(n);
        return;
    }
    if (isPrime(n)) {
        f.push_back(n);
        return;
    }
    long long d = n;
    for(int i = 2; d == n; i++){
        d = pollard_rho(n, i);
    }
    llfactorize(d, f);
    llfactorize(n/d, f);
}

void factorize(long long n, vector<pair<long long,long long>>
    &ans) {
```

```
    vector<long long> v;
    llfactorize(n, v);
    sort(v.begin(), v.end());
    long long a = v[0], b = 1;
    for(int i = 1; i < v.size(); i++) {
        if (v[i] == v[i-1] ) b++;
        else {
            ans.push_back({a,b});
            a = v[i];
            b = 1;
        }
    }
    ans.push_back({a,b});
}


int main() {

    sieve();
    while(true) {
        long long n;
        cin >> n;
        vector<pair<long long,long long>> f;
        factorize(n, f);
        for(int i = 0; i < f.size(); i++) {
            cout << f[i].first << " " << f[i].second << endl;
        }
    }
    return 0;
}
```

4.23 primeCountingTrick

```
#define maxn 1000000
i64 Lo[maxn+5] , Hi[maxn+5] ;
void primeCount( i64 N )
{
  i64 i , j , k , l , m ; i64 s = sqrt(N+0.0) + 1 ;
  for(i=1 ; i<=s ; i++) Lo[i] = i-1 ;
  for(i=1 ; i<=s ; i++) Hi[i] = (N/i) - 1 ;
  for(i=2 ; i<=s ; i++) {
    if( Lo[i] == Lo[i-1] ) continue ;
    i64 isq = i*i , lim = N/isq ;
    // we need , ( N/j ) >= i*i => j <= ( N/(i*i) )
    for( j=1 ; j<=lim && j<=s ; j++ ) {
      if(i*j>s) Hi[j] = Hi[j] - ( Lo[N/(i*j)] - Lo[i-1] ) ;
      else Hi[j] = Hi[j] - ( Hi[i*j] - Lo[i-1] ) ;
    }
    for( j=s ; j>=isq ; j-- ){ // j >= i*i
      Lo[j] = Lo[j] - ( Lo[j/i] - Lo[i-1] ) ;
    }
  }
    return ;
}
```

4.24 xor basis

```
#include <bits/stdc++.h>
using namespace std;

/// not tested yet

typedef long long ll;

const int N = 1e6 + 100;
const int lg = 18;

vector<int> xor_basis(vector <int> a) {
```

```
    int n = a.size();
    int row = 0;
    vector<int> basis;
    vector<bool> colm(lg, 0);
    for(int k = lg-1; k >= 0; k--) {
        for(int i = row; i < n; i++) {
            if((a[i]>>k)&1) {
                swap(a[i], a[row]);
                break;
            }
        }
        if(row < n && (a[row]>>k)&1) {
            basis.push_back(a[row]);
            colm[k] = 1;
            for(int i = row + 1; i < n; i++) {
                if((a[i]>>k)&1) a[i] ^= a[row];
            }
            row++;
        }
    }
    return basis;
}


int main() {
    ios::sync_with_stdio(0);
    cin.tie(0);
    int n;
    cin >> n;
    vector<int> a(n);
    for(int i = 0; i < n; i++) {
        cin >> a[i];
    }
    xor_basis(a);
    return 0;
}
```

5 Miscellaneous

5.1 Berlekamp-Massey

```
/// not tested
#include <bits/stdc++.h>
using namespace std;
#define pb push_back
typedef long long ll;
#define SZ 233333
const int MOD=1e9+7; //or any prime
ll qp(ll a,ll b)
{
        ll x=1; a%=MOD;
        while(b)
        {
                if(b&1) x=x*a%MOD;
                a=a*a%MOD; b>>=1;
        }
        return x;
}
namespace linear_seq {
inline vector<int> BM(vector<int> x)
{
        //ls: (shortest) relation sequence (after filling zeroes)
           so far
        //cur: current relation sequence
        vector<int> ls,cur;
        //lf: the position of ls (t')
        //ld: delta of ls (v')
        int lf,ld;
```

```
	for(int i=0;i<int(x.size());++i)
	{
		ll t=0;
		//evaluate at position i
		for(int j=0;j<int(cur.size());++j)
			t=(t+x[i-j-1]*(ll)cur[j])%MOD;
		if((t-x[i])%MOD==0) continue; //good so far
		//first non-zero position
		if(!cur.size())
		{
			cur.resize(i+1);
			lf=i; ld=(t-x[i])%MOD;
			continue;
		}
		//cur=cur-c/ld*(x[i]-t)
		ll k=-(x[i]-t)*qp(ld,MOD-2)%MOD/*1/ld*/;
		vector<int> c(i-lf-1); //add zeroes in front
		c.pb(k);
		for(int j=0;j<int(ls.size());++j)
			c.pb(-ls[j]*k%MOD);
		if(c.size()<cur.size()) c.resize(cur.size());
		for(int j=0;j<int(cur.size());++j)
			c[j]=(c[j]+cur[j])%MOD;
		//if cur is better than ls, change ls to cur
		if(i-lf+(int)ls.size()>=(int)cur.size())
			ls=cur,lf=i,ld=(t-x[i])%MOD;
		cur=c;
	}
	for(int i=0;i<int(cur.size());++i)
		cur[i]=(cur[i]%MOD+MOD)%MOD;
	return cur;
}
int m; //length of recurrence
//a: first terms
//h: relation
ll a[SZ],h[SZ],t_[SZ],s[SZ],t[SZ];
//calculate p*q mod f
inline void mull(ll*p,ll*q)
{
	for(int i=0;i<m+m;++i) t_[i]=0;
	for(int i=0;i<m;++i) if(p[i])
		for(int j=0;j<m;++j)
			t_[i+j]=(t_[i+j]+p[i]*q[j])%MOD;
	for(int i=m+m-1;i>=m;--i) if(t_[i])
		//miuns t_[i]x^{i-m}(x^m-\sum_{j=0}^{m-1}
		   x^{m-j-1}h_j)
		for(int j=m-1;~j;--j)
			t_[i-j-1]=(t_[i-j-1]+t_[i]*h[j])%MOD;
	for(int i=0;i<m;++i) p[i]=t_[i];
}
inline ll calc(ll K)
{
	for(int i=m;~i;--i)
		s[i]=t[i]=0;
	//init
	s[0]=1; if(m!=1) t[1]=1; else t[0]=h[0];
	//binary-exponentiation
	while(K)
	{
		if(K&1) mull(s,t);
		mull(t,t); K>>=1;
	}
	ll su=0;
	for(int i=0;i<m;++i) su=(su+s[i]*a[i])%MOD;
	return (su%MOD+MOD)%MOD;
}
inline int work(vector<int> x,ll n)
{
	if(n<int(x.size())) return x[n];
	vector<int> v=BM(x); m=v.size(); if(!m) return 0;
```

```
        for(int i=0;i<m;++i) h[i]=v[i],a[i]=x[i];
        return calc(n);
}
}
using linear_seq::work;
int main()
{
        cout<<work({1,1,2,3,5,8,13,21},10)<<"\n";
}
```

5.2 FastIO

```
public class Main {
  public static void main(String[] args) {
    InputStream inputStream = System.in;
    OutputStream outputStream = System.out;
    InputReader in = new InputReader(inputStream);
    PrintWriter out = new PrintWriter(outputStream);
    int n = in.nextInt();
    long l = in.nextLong();
    out.println(n);
    out.println(l);
    out.println("done");
    out.close();
  }
  static class InputReader {
    public BufferedReader reader;
    public StringTokenizer tokenizer;
    public InputReader(InputStream stream) {
      reader = new BufferedReader(new InputStreamReader(stream),
          32768);
      tokenizer = null;
    }
    public String next() {
      while (tokenizer == null || !tokenizer.hasMoreTokens()) {
        try {
          tokenizer = new StringTokenizer(reader.readLine());
        } catch (IOException e) {
          throw new RuntimeException(e);
        }
      }
      return tokenizer.nextToken();
    }

    public int nextInt() {
      return Integer.parseInt(next());
    }
    public long nextLong(){
      return Long.parseLong(next());
    }


  }
}
```

5.3 GrayCode

```
long long gray_code(long long x){
    return x ^ (x >> 1);
}
long long inverse_gray_code(long long x){
    long long h = 1, res = 0;
    do{ if (x & 1) res ^= h;
        x >>= 1, h = (h << 1) + 1;
    } while (x);
    return res;
}
```

5.4 LIS

```
/// Longest Increasing Subsequence
#include <bits/stdc++.h>
using namespace std ;
const int N = 1e6;
int a[N] , LIS[N] ; /// to find decreasing Sub , simply multiply
    the elements by -1

int main () {
    int n ;
    for (int i = 1 ; i <= n ; i++) {
        scanf ("%d" , &a[i]) ;
    }
    vector <int> v ;
    for (int i = 1 ; i <= n ; i++) {
        int x = a[i] ;
        vector<int>::iterator it = upper_bound(v.begin(),
            v.end(), x) ; /// change it to lower_bound for
            strictly increasing subsequence
        if (it == v.end()) v.push_back(x);
        else *it = x;
        LIS[i] = upper_bound(v.begin(), v.end(), x) - v.begin() ;
            /// change it to lower_bound for strictly increasing
            subsequence
    }
}
```

5.5 Miscellenous

```
/// Random
mt19937
    rng(chrono::steady_clock::now().time_since_epoch().count());
shuffle(V.begin(), V.end(), rng); int x = rng();
/// bit manipulation
number of leading zeros: __builtin_clz(x)
number of trailing zeros: __builtin_ctz(x)
number of set bits : __builtin_popcountll(x)
bitset : bs._Find_first(),bs._Find_next(15)
///subset(3^n)
for(int i = mask; i > 0; i = ((i-1) & mask))
/// ordered set
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;
typedef tree < int, null_mapped_type ,less<int>,rb_tree_tag,
tree_order_statistics_node_update > ordered_set;
find_by_order:returns an iterator to k-th largest element
(counting from zero) , order_of_key : returns the number of
items in a set that are strictly smaller than our item.
/// 2D Partial Sum : update (x1,y1) to (x2,y2) +x
a[x1][y1]+=x;a[x1][y2+1]-=x;a[x2+1][y1]-=x;a[x2+1][y2+1]+=x;
reconstruction: a[x][y] += a[x-1][y]+a[x][y-1]-a[x-1][y-1]
/// __int128:
__int128 x = 1e12; x = x * x + 1000;
while(x) {res.pb(x%10 + '0'); x/= 10;}
/// split a string by space
string str = "abc def gh" , buf; stringstream ss(str);
while(ss >> buf) cout << buf << endl;
/// ntt mod :
998244353 = 119 * 2^23 + 1 , primitive root = 3
985661441 = 235 * 2^22 + 1 , primitive root = 3
1012924417 = 483 * 2^21 + 1 , primitive root = 5
/// MO on tree
case-1: lca(u,v) == u , [ST(u),ST(v)]
```

```
case-2: otherwise, [EN(u),ST(v)] + [ST(lca), ST (lca)]
///
```

5.6 Mo Order fast

```
#include<bits/stdc++.h>
using namespace std;

/// complexity O(nsqrt(q))

inline int64_t gilbertOrder(int x, int y, int pow, int rotate) {
	if (pow == 0) {
		return 0;
	}
	int hpow = 1 << (pow-1);
	int seg = (x < hpow) ? (
		(y < hpow) ? 0 : 3
	) : (
		(y < hpow) ? 1 : 2
	);
	seg = (seg + rotate) & 3;
	const int rotateDelta[4] = {3, 0, 0, 1};
	int nx = x & (x ^ hpow), ny = y & (y ^ hpow);
	int nrot = (rotate + rotateDelta[seg]) & 3;
	int64_t subSquareSize = int64_t(1) << (2*pow - 2);
	int64_t ans = seg * subSquareSize;
	int64_t add = gilbertOrder(nx, ny, pow-1, nrot);
	ans += (seg == 1 || seg == 2) ? add : (subSquareSize -
	    add - 1);
	return ans;
}

struct Query {
	int l, r, idx;
	int64_t ord;

	inline void calcOrder() {
		ord = gilbertOrder(l, r, 21, 0); // 2^k >= n, you
		    can use k = 20 for most cases
	}
};

inline bool operator<(const Query &a, const Query &b) {
	return a.ord < b.ord;
}


int main() {

}
```

5.7 Simplex

```
/* O(n^2m) , n = no of var , m = no of inequalities
 * Algorithm : Simplex ( Linear Programming )
 * Author : Simon Lo
 * Note: Simplex algorithm on augmented matrix a of dimension
    (m+1)x(n+1)
 * returns 1 if feasible, 0 if not feasible, -1 if unbounded
 * returns solution in b[] in original var order, max(f) in ret
 * form: maximize sum_j(a_mj*x_j)-a_mn s.t. sum_j(a_ij*x_j)<=a_in
 * in standard form.
 * To convert into standard form:
 * 1. if exists equality constraint, then replace by both >= and
    <=
 * 2. if variable x doesn't have nonnegativity constraint, then
    replace by
 * difference of 2 variables like x1-x2, where x1>=0, x2>=0
```

```
 * 3. for a>=b constraints, convert to -a<=-b
 * note: watch out for -0.0 in the solution, algorithm may cycle
 * EPS = 1e-7 may give wrong answer, 1e-10 is better
 */

#include<bits/stdc++.h>
using namespace std;
#define MAXM 10005
#define MAXN 105
#define INF 1000000007
#define EPS (1e-12)
#define si(a) scanf("%d",&a)
#define PI acos(-1)
#define f first
#define s second
#define mp(a,b) make_pair(a,b)

struct Simplex {
    void pivot( int m,int n,double A[MAXM+7][MAXN+7],int *B,int
        *N,int r,int c ) {
        int i,j;
        swap( N[c],B[r] );
        A[r][c] = 1/A[r][c];
        for( j=0; j<=n; j++ ) if( j!=c ) A[r][j] *= A[r][c];
        for( i=0; i<=m; i++ ) {
            if( i!=r ) {
                for( j=0; j<=n; j++ ) if( j!=c ) A[i][j] -=
                    A[i][c]*A[r][j];
                A[i][c] = -A[i][c]*A[r][c];
            }
        }
    }

    int feasible( int m,int n,double A[MAXM+7][MAXN+7],int
        *B,int *N ) {
        int r,c,i;
        double p,v;
        while( 1 ) {
            for( p=INF,i=0; i<m; i++ ) if( A[i][n]<p ) p =
                A[r=i][n];
            if( p > -EPS ) return 1;
            for( p=0,i=0; i<n; i++ ) if( A[r][i]<p ) p =
                A[r][c=i];
            if( p > -EPS ) return 0;
            p = A[r][n]/A[r][c];
            for( i=r+1; i<m; i++ ) {
                if( A[i][c] > EPS ) {
                    v = A[i][n]/A[i][c];
                    if( v<p ) r=i,p=v;
                }
            }
            pivot( m,n,A,B,N,r,c );
        }
    }

    int simplex( int m,int n,double A[MAXM+7][MAXN+7],double
        *b,double &Ret ) {
        int B[MAXM*MAXN+7],N[MAXM*MAXN+7],r,c,i;
        double p,v;
        for( i=0; i<n; i++ ) N[i] = i;
        for( i=0; i<m; i++ ) B[i] = n+i;
        if( !feasible( m,n,A,B,N ) ) return 0;
        while( 1 ) {
            for( p=0,i=0; i<n; i++ ) if( A[m][i] > p ) p =
                A[m][c=i];
            if( p<EPS ) {
                for( i=0; i<n; i++ ) if( N[i]<n ) b[N[i]] = 0;
                for( i=0; i<m; i++ ) if( B[i]<n ) b[B[i]] =
                    A[i][n];
                Ret = -A[m][n];
```

```
            return 1;
        }
        for( p=INF,i=0; i<m; i++ ) {
            if( A[i][c] > EPS ) {
                v = A[i][n]/A[i][c];
                if( v<p ) p = v,r = i;
            }
        }
        if( p==INF ) return -1;
        pivot( m,n,A,B,N,r,c );
    }
  }
}S;

typedef pair<int,int> pii ;
pii P[105];
double mat[MAXM+7][MAXN+7];

int dist(int i,int j) {
    return (P[i].first-P[j].first)*(P[i].first-P[j].first) +
        (P[i].second-P[j].second)*(P[i].second-P[j].second) ;
}

int main() {
    //freopen ("in.txt","r",stdin) ;
    int n ; scanf("%d",&n);
    for(int i = 0 ; i < n ; i++) {
        cin >> P[i].first >> P[i].second ;
    }
    int eq = 0 ;
    for(int i = 0 ; i < n ; i++) {
        for(int j = 0 ; j < n ; j++) {
            if (i == j) continue ;
            mat[eq][i] = 1.0 ;
            mat[eq][j] = 1.0 ;
            mat[eq][n] = sqrt(dist(i,j)) ;
            eq++ ;
        }
    }
    for(int i = 0 ; i < n ; i++) mat[eq][i] = 1.0 ;
    double res[MAXN+7] , ans ;
    int t = S.simplex(eq,n,mat,res,ans) ;
    ans *= 2.0*acos(-1.0) ;
    printf ("%.12f\n" , ans) ;
}
```

5.8 closestpairpoints

```
/// Closest Pair Point
#include <bits/stdc++.h>
using namespace std ;

typedef pair<long long,long long> point ;

const int N = 1e5 + 5 ;
int a[N] ;

struct ClosestPairPoints {
    vector < point > points ;
    const long long INF = 1e18 ;

    void init() {
        points.clear() ;
    }
    void addPoint(point P) {
        points.push_back(P) ;
    }
    long long Dist (int i , int j) {
        long long dx = points[i].first - points[j].first ;
```

```
    long long dy = points[i].second - points[j].second ;
    return dx*dx + dy*dy ;
}
long long DivideAndConq(int l , int r) {
    if (r - l + 1 <= 3) {
        long long ans = INF;
        for (int i = l ; i <= r ; i++) {
            for (int j = i+1 ; j <= r ; j++) {
                ans = min(ans , Dist(i,j)) ;
            }
        }
        return ans ;
    }
    int mid = (l + r) / 2 ;
    long long ans =
        min(DivideAndConq(l,mid),DivideAndConq(mid+1,r)) ;
    vector < pair<long long,long long> > between ;
    for (int i = l ; i <= r ; i++) {
        long long dx = points[mid].first - points[i].first ;
        if (dx*dx < ans) {
            between.push_back({points[i].second,points[i].first})
                ;
        }
    }
    sort(between.begin(),between.end()) ;
    for (int i = 0 ; i < between.size() ; i++) {
        for (int j = i + 1 ; j < between.size() ; j++) {
            long long dy = between[i].first - between[j].first
                ;
            if (dy*dy >= ans) {
                break ;
            }
            long long dx = between[i].second -
                between[j].second ;
            ans = min(ans , dx*dx + dy*dy) ;
        }
    }
    return ans ;
}

long long ClosestPair() {
    sort(points.begin(),points.end()) ;
    return DivideAndConq(0,(int)points.size()-1) ;
}
};


int main() {
    int n ;
    cin >> n ;
    ClosestPairPoints T ;
    T.init() ;
    for (int i = 1 ; i <= n ; i++) {
        point P ;
        cin >> P.first >> P.second ;
        T.addPoint(P) ;
    }
    printf ("%lld\n",T.ClosestPair()) ;
}
```

5.9 dancing link

```
/// Dancing Link

#include <bits/stdc++.h>
using namespace std;

class DLX {
    public:
```

```
static const int MAXCL = 256;
    static const int MAXDL = 131072 + 256;
struct DancingLinks {
    int left, right, up, down, ch;
    int id;
} DL[MAXDL];
int s[MAXCL], o[MAXCL], head, size;
void remove(int c) {
    DL[DL[c].right].left = DL[c].left;
    DL[DL[c].left].right = DL[c].right;
    int i, j;
    for (i = DL[c].down; i != c; i = DL[i].down) {
        for (j = DL[i].right; j != i; j = DL[j].right) {
            DL[DL[j].down].up = DL[j].up;
            DL[DL[j].up].down = DL[j].down;
            s[DL[j].ch]--;
        }
    }
}
void resume(int c) {
    int i, j;
    for (i = DL[c].down; i != c; i = DL[i].down) {
        for (j = DL[i].left; j != i; j = DL[j].left) {
            DL[DL[j].down].up = j;
            DL[DL[j].up].down = j;
            s[DL[j].ch]++;
        }
    }
    DL[DL[c].right].left = c;
    DL[DL[c].left].right = c;
}
int ret[128];
void dfs(int dep, int remain) {
    ret[dep + remain] = 1;
    if (DL[head].right == head)
        return;
    int tmp = 0, c, i, j;
    // special: pick the maximum options column
    for (i = DL[head].right; i != head; i = DL[i].right)
        if (s[i] > tmp)
            tmp = s[i], c = i;
    // not always exists exact cover
    if (tmp == 0)
            return ;
    remove(c);
    for (i = DL[c].down; i != c; i = DL[i].down) {
        for (j = DL[i].right; j != i; j = DL[j].right)
            remove(DL[j].ch);
        dfs(dep+1, remain - DL[i].id);
        for (j = DL[i].left; j != i; j = DL[j].left)
            resume(DL[j].ch);
    }
    resume(c);
}
int newNode(int u, int d, int l, int r) {
    DL[size].up = u, DL[size].down = d;
    DL[size].left = l, DL[size].right = r;
    DL[u].down = DL[d].up = DL[l].right = DL[r].left = size;
    assert(size < MAXDL);
    return size++;
}
void newRow(int r[], int rn, int id) {
    int i, j, h;
    for (i = 0; i < rn; i++) {
        DL[size].ch = r[i], s[r[i]]++;
        DL[size].id = id; // extra data
        if (i) {
            j = newNode(DL[DL[r[i]].ch].up, DL[r[i]].ch,
                DL[h].left, h);
        } else {
```

```
            h = newNode(DL[DL[r[i]].ch].up, DL[r[i]].ch, size,
                size);
        }
    }
}
void init(int c) {// total column
    size = 0;
    head = newNode(0, 0, 0, 0);
    for (int i = 1; i <= c; i++) {
        newNode(i, i, DL[head].left, head);
        DL[i].ch = i, s[i] = 0;
    }
}
} DLX;

int main() {
    int cases = 0;
    int a, b, c, m;
    while (scanf("%d %d %d %d", &a, &b, &c, &m) == 4 && a) {
        int8_t g[20][20][20] = {};
        int8_t ig[20][20][20] = {}, cg[20][20][20] = {};
        for (int i = 0; i < m; i++) {
            int x, y, z;
            scanf("%d %d %d", &x, &y, &z);
            x--, y--, z--;
            g[x][y][z] = -1;
        }

        int col = 0, tot = 0;
        for (int i = 0; i < a; i++) {
            for (int j = 0; j < b; j++) {
                for (int k = 0; k < c; k++) {
                    if (g[i][j][k] == 0)
                        tot++;
                }
            }
        }

        for (int i = 0; i < a; i++) {
            for (int j = 0; j < b; j++) {
                for (int k = 0; k < c; k++) {
                    if (g[i][j][k] == -1)
                        continue;
                    int8_t t = 1;
                    if (i && j && k) {
                        t = g[i-1][j-1][k-1];
                        t = min(t,
                            g[i-1][j-1][k]);
                        t = min(t,
                            g[i-1][j][k-1]);
                        t = min(t,
                            g[i-1][j][k]);
                        t = min(t,
                            g[i][j-1][k-1]);
                        t = min(t,
                            g[i][j-1][k]);
                        t = min(t,
                            g[i][j][k-1]);
                        if (t < 0)
                            t = 1;
                        else
                            t++;
                    }
                    g[i][j][k] = t;
                    if (t > 1) {
                        for (int p = 0; p <
                            t; p++) {
                            for (int q =
                                0; q < t;
                                q++) {
```

```
                                        for
                                            (int
                                            r =
                                            0;
                                            r <
                                            t;
                                            r++)
                                                cg[i-p][j-q][k-r]
                                                    =
                                                    1;
                                    }
                                }
                            }
                        }
                    }
                }
            }
        }
    }
}

for (int i = 0; i < a; i++) {
        for (int j = 0; j < b; j++) {
                for (int k = 0; k < c; k++) {
                        if (cg[i][j][k] == 1)
                                ig[i][j][k] = ++col;
                }
        }
}
assert(col < 128);
DLX.init(col);

for (int i = 0; i < a; i++) {
        for (int j = 0; j < b; j++) {
                for (int k = 0; k < c; k++) {
                        if (g[i][j][k] < 2)
                                continue;
                        int t = g[i][j][k];
                        static int row[128];
                        for (int it = t; it >= 2;
                            it--) {
                                int row_cnt = 0;
                                for (int p = 0; p <
                                    it; p++) {
                                        for (int q =
                                            0; q < it;
                                            q++) {
                                                for
                                                    (int
                                                    r =
                                                    0;
                                                    r <
                                                    it;
                                                    r++)
                                                        row[rc
                                                            =
                                                            ig
                                        }
                                }
                                DLX.newRow(row,
                                    row_cnt,
                                    it*it*it);
                        }
                }
        }
}
memset(DLX.ret, 0, sizeof(DLX.ret));
DLX.dfs(0, tot);p09
printf("Case %d:", ++cases);
for (int i = 1; i <= tot; i++) {
        if (DLX.ret[i])
                printf(" %d", i);
}
puts("");
```

```
            fflush(stdout);
        }
        return 0;
}
```

5.10 mat expo

```
#include <bits/stdc++.h>
using namespace std;

int mod = 1e9 + 7;

vector<vector<int>> multiply(vector<vector<int>> &a,
    vector<vector<int>> &b) {
    int dim = a.size();
    vector<vector<int>> c(dim, vector<int>(dim, 0));
    for(int i = 0; i < dim; i++) {
        for(int j = 0; j < dim; j++) {
            for(int k = 0; k < dim; k++) {
                c[i][j] += 1ll*a[i][k]*b[k][j]%mod;
                c[i][j] %= mod;
            }
        }
    }
    return c;
}

vector<vector<int>> bigmod(long long p, vector<vector<int>>
    &base) {
    if(p == 0) {
        int dim = (int)base.size();
        vector<vector<int>> unity(dim, vector<int>(dim, 0));
        for(int i = 0; i < dim; i++) unity[i][i] = 1;
        return unity;
    }
    if(p&1) {
        vector<vector<int>> ret = bigmod(p-1,base);
        return multiply(ret, base);
    }
    vector<vector<int>> ret = bigmod(p/2,base);
    return multiply(ret,ret);
}

int main() {

}
```

5.11 ordered set

```
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>

using namespace std;
using namespace __gnu_pbds;
typedef
tree < int, null_type ,less<int>,
rb_tree_tag,
tree_order_statistics_node_update > ordered_set;

/*
    Operations :
    find_by_order : returns an iterator to the k-th
    largest element (counting from zero).
    order_of_key : returns the number of items in
    a set that are strictly smaller than our item.
    # to use as a multiset just insert a pair<int,int> with
        unique second element, change
```

```
    int to pair<int,int> in typedef part
*/


ordered_set Set;


int main() {

    Set.insert(5);
    Set.insert(2);
    Set.insert(6);
    Set.insert(5);
    cout << Set.order_of_key(2) << endl;
    cout << Set.size() << endl;
    auto it = Set.find_by_order(2); /// iterator to the 3th
       number
    cout << *it << endl;
    return 0;
}
```

5.12 stern brocot tree(copied)

```
#include<bits/stdc++.h>
using namespace std;

#define x first
#define y second

/// finds a fraction x/y with minimum y, such that L <= (x/y) < R
/// https://codeforces.com/blog/entry/50244

pair<long long, long long> solve(long double const&L, long
    double const&R){
    pair<long long, long long> l(0, 1), r(1, 1);
    if(L==0.0) return l; // corner case
    for(;;){
        pair<int, int> m(l.x+r.x, l.y+r.y);
        if(m.x<L*m.y){// move to the right;
            long long kl=1, kr=1;
            while(l.x+kr*r.x <= L*(l.y+kr*r.y))kr*=2;//
               exponential search
            while(kl!=kr){
                long long km = (kl+kr)/2;
                if(l.x+km*r.x < L*(l.y+km*r.y)) kl=km+1;
                else kr=km;
            }
            l = make_pair(l.x+(kl-1)*r.x, l.y+(kl-1)*r.y);
        } else if(m.x>=R*m.y){//move to the left
            long long kl=1, kr=1;
            while(r.x+kr*l.x>=R*(r.y+kr*l.y))kr*=2;// exponential
               search
            while(kl!=kr){
                long long km = (kl+kr)/2;
                if(r.x+km*l.x>=R*(r.y+km*l.y)) kl = km+1;
                else kr = km;
            }
            r = make_pair(r.x+(kl-1)*l.x, r.y+(kl-1)*l.y);
        } else {
            return m;
        }
    }
}


int main() {
    double L = 0.1, R = 0.5;
    pair<long long,long long> res = solve(L,R);
    cout << res.x << " " << res.y << endl;
```

```
}
```

5.13 xor mst

```
#include <bits/stdc++.h>
using namespace std;

const int N = 2e5 + 4;
const int LOG = 30;

int a[N], n;
int par[N];
int totComp;
long long MST = 0;

vector<int> comp[N];
vector<int> mst[N];

void addEdge(int u,int v) {
    mst[u].push_back(v);
    mst[v].push_back(u);
    --totComp;
    MST = MST + (long long) (a[u] ^ a[v]);
}

int to[N*LOG + 100][2];
int cnt[N*LOG + 100];
int totNodes = 1;

void add(int x, int idx) {
    int cur = 1;
    for(int k = 30 ; k >= 0 ; k--) {
        cnt[cur]++;
        int b = ((x>>k)&1);
        if (!to[cur][b]) to[cur][b] = ++totNodes;
        cur = to[cur][b];
    }
    cnt[cur]++;
    to[cur][0] = idx;
}

void Erase(int x) {
    int cur = 1;
    for(int k = 30 ; k >= 0 ; k--) {
        cnt[cur]--;
        int b = ((x>>k)&1);
        if (!to[cur][b]) assert(1 == 0), to[cur][b] = ++totNodes;
        cur = to[cur][b];
    }
    cnt[cur]--;
}

int query(int x) {
    int cur = 1;
    for(int k = 30 ; k >= 0 ; k--) {
        int b = ((x>>k)&1);
        if (to[cur][b] && cnt[to[cur][b]]) {
            cur = to[cur][b];
        } else {
            cur = to[cur][b^1];
        }
    }
    return to[cur][0];
}

int parent(int u) {
    if (par[u] == u) return u;
    return par[u] = parent(par[u]);
}
```

```
int main() {
    scanf("%d",&n);
    map <int,int> ocr;
    totComp = n;
    for(int i = 1; i <= n; i++) {
        scanf("%d",&a[i]);
        if (ocr.find(a[i]) == ocr.end()) {
            ocr[a[i]] = i;
            comp[i].push_back(i);
            par[i] = i;
            add(a[i],i);
        }
        else {
            int u = ocr[a[i]];
            par[i] = u;
            addEdge(i,u);
        }
    }

    while(totComp > 1) {
        set < pair<int,int> > edge;
        for(int i = 1; i <= n; i++) {
            if (comp[i].size() == 0) continue;
            for(int u : comp[i]) {
                Erase(a[u]);
            }
            int xorMin = 2e9, U, V;
            for(int u : comp[i]) {
                int v = query(a[u]);
                int xorCur = a[u] ^ a[v];
                if (xorCur < xorMin) {
                    xorMin = xorCur;
                    U = u, V = v;
                }
            }
            if(U > V) swap(U,V);
            edge.insert({U,V});
            for(int u : comp[i]) {
                add(a[u],u);
            }
        }

        for(pair<int,int> curEdge : edge) {
            int u = curEdge.first , v = curEdge.second;
            int pu = parent(u) , pv = parent(v);
            if(pu == pv) continue ;
            addEdge(u,v);
            if (comp[pu].size() < comp[pv].size()) swap(pu,pv);
            for(int node : comp[pv]) {
                par[node] = pu;
                comp[pu].push_back(node);
            }
            comp[pv].clear();
        }
    }

    cout << MST << endl;
}
```

6 String Algorithms

6.1 Palindromic Tree

```
/// Palindromic Tree
#include <bits/stdc++.h>
using namespace std;
const int N = 1e5 + 5;
```

```
struct node {
    int next[26] , len , sufflink , num ;
    /// len : length of the palindrome in node
    /// num : depth of the suffix link
};

int len;
char str[N];
node tree[N];
int sz;           /// node 1 - root with len -1, node 2 - root
    with len 0
int suff;          /// max suffix palindrome
long long ans;

bool addLetter(int pos) {
    int cur = suff, curlen = 0;
    int let = str[pos] - 'a';

    while (true) {
        curlen = tree[cur].len;
        if (pos - 1 - curlen >= 0 && str[pos - 1 - curlen] ==
            str[pos])
            break;
        cur = tree[cur].sufflink;
    }
    if (tree[cur].next[let]) {
        suff = tree[cur].next[let];
        return false;
    }

    suff = ++sz;
    tree[sz].len = tree[cur].len + 2;
    tree[cur].next[let] = sz;
    if (tree[sz].len == 1) {
        tree[sz].sufflink = 2;
        tree[sz].num = 1;
        return true;
    }

    while (true) {
        cur = tree[cur].sufflink;
        curlen = tree[cur].len;
        if (pos - 1 - curlen >= 0 && str[pos - 1 - curlen] ==
            str[pos]) {
            tree[sz].sufflink = tree[cur].next[let];
            break;
        }
    }

    tree[sz].num = 1 + tree[tree[sz].sufflink].num;

    return true;
}

void initTree() {
    memset (tree , 0 , sizeof tree) ;
    sz = 2; suff = 2;
    tree[1].len = -1; tree[1].sufflink = 1;
    tree[2].len = 0; tree[2].sufflink = 1;
}

int main() {
    scanf ("%s" , &str) ;
    len = strlen(str);
    initTree();
    for (int i = 0; i < len; i++) {
        ans += addLetter(i);
        printf ("%d " , ans) ;
```

```
    }

    return 0;
}
```

6.2 Suffix Automata(Arghya)

```
class SuffixAutomaton{
public:
  struct state{
      int edge[27] , len , link , cnt[2] ;
  };

  state *st ; int sz , last ;
  SuffixAutomaton( string &s , int k ) {
    int l = s.length() ; int i , j ;
    st = new state[l*2] ;
    st[0].link = -1 ; st[0].len = 0 ; sz = 1 ; last = 0 ;
    for(i=0 ; i<27 ; i++) st[0].edge[i] = -1 ;

  for(i=0 ; i<l ; i++) {
    int cur = sz++ ;
    for(j=0 ; j<27 ; j++) st[cur].edge[j] = -1 ;
    st[cur].len = st[last].len+1 ;
    int p = last , c = s[i]-'a' ;
    while( p!=-1 && st[p].edge[c] == -1 ) {
      st[p].edge[c] = cur ;
      p = st[p].link ;
    }
    if( p == -1 ) st[cur].link = 0 ;
    else{
      int q = st[p].edge[c] ;
      if( st[p].len+1 == st[ q ].len ) st[cur].link = q ;
      else{
        int clone = sz++ ;
        for(j=0;j<27;j++) st[clone].edge[j] = st[q].edge[j] ;
        st[clone].len = st[p].len+1 ;
        st[clone].link = st[q].link ;
        while( p!=-1 && st[p].edge[c] == q ) {
          st[p].edge[c] = clone ; p = st[p].link ;
        }
        st[q].link = st[cur].link = clone ;
      }
    }
    last = cur ;
  }
}
  ~SuffixAutomaton() {
    delete []st ;
  }
};
```

6.3 Suffix$_{A}$*utomata*

```
#include <bits/stdc++.h>
using namespace std;

/**
0 is the root-(null string)
advance without memoization is applicable for
if you traverse through a
string (no tree) which is usually the case

Application:
# Number of Occ of each state:
    initialize each state(except the clones) with cnt[state] = 1
    loop in decreasing order of len[state], and update:
        cnt[link[state]] += cnt[state]
```

```
    Small to Large Approach might be used to actually know the
        positions rather than the count

# First Occ of each state:
    for new state: firstpos(cur) = len(cur)-1
    for cloned state: firstpos(clone) = firstpos(q)

# Longest Common Substring:
    Build Automata for one string, iterate over other's all
        suffix
**/


const int ALPHA = 26;

namespace SuffixAutomata {
    vector<vector<int>> to, next_state;
    vector<int> link, len;
    int n, sz, cur;

    void add(int c) {
        int p = cur;
        cur = ++sz;
        len[cur] = len[p] + 1;
        while (to[p][c] == 0) {
            to[p][c] = cur;
            p = link[p];
        }
        if (to[p][c] == cur) {
            link[cur] = 0;
            return;
        }
        int q = to[p][c];
        if (len[q] == len[p] + 1) {
            link[cur] = q;
            return;
        }
        int cl = ++sz;
        to[cl] = to[q];
        link[cl] = link[q];
        len[cl] = len[p] + 1;
        link[cur] = link[q] = cl;
        while (to[p][c] == q) {
            to[p][c] = cl;
            p = link[p];
        }
    }
    /** advance with memoization **/
    int advance(int state, int c) {
        if(next_state[state][c] != -1) return
            next_state[state][c];
        int nstate;
        if(to[state][c]) nstate = to[state][c];
        else if(state) nstate = advance(link[state], c);
        else nstate = state;
        return next_state[state][c] = nstate;
    }
    /** advance without memoization **/
    /**
    int advance(int state, int c) {
        while (state and !to[state][c]) state = link[state];
        if (to[state][c]) state = to[state][c];
        return state;
    }
    **/
    void build(string &s) {
        cur = sz = 0;
        n = s.size();
        to.assign(2*n+1, vector<int> (ALPHA, 0)); /// null state
            + 2*n
```

```
        next_state.assign(2*n+1, vector<int> (ALPHA, -1));
        link.assign(2*n+1, 0);
        len.assign(2*n+1, 0);
        for(int i = 0; i < n; i++) {
            add(s[i]-'a'); /// change if CAPITAL Letters are
                needed
        }
    }
}

int main () {
    string s;
    cin >> s;
    SuffixAutomata::build(s);
    return 0;
}
```

6.4 aho$_c$*orasick*

```
/// lightoj Aho Corasick Solution :
/// Given n distinct patterns and 1 main string (str) ,
    calculates the number
/// times each pattern occurs in the main string

#include <bits/stdc++.h>
using namespace std;

const int N = 250000 + 1000, ALPHA = 26;

struct AC {
    struct state {
        int to[ALPHA], depth, suffLink, par, parLet, cnt,
            nxt[ALPHA];
    }states[N];
vector<int> suffix_tree[N];
int tot_nodes;
void init() {
    for(int i = 0; i < N; i++) suffix_tree[i].clear();
    memset(states, 0, sizeof states); /// be careful if it
        gets tle for too much memset
    tot_nodes = 1;
}
int add_string(string &str) {
    int cur = 1;
    for(int i = 0; i < str.size(); i++) {
        int c = str[i]-'a';
        if(!states[cur].to[c]) {
            states[cur].to[c] = ++tot_nodes;
            states[tot_nodes].par = cur;
            states[tot_nodes].depth = states[cur].depth + 1;
            states[tot_nodes].parLet = c;
        }
        cur = states[cur].to[c];
    }
    return cur;
}
void push_links() {
    queue <int> que;
    que.push(1);
    while (!que.empty()) {
        int node = que.front();
        que.pop();
        if (states[node].depth <= 1) states[node].suffLink =
            1;
        else {
            int cur = states[states[node].par].suffLink;
            int parLet = states[node].parLet;
            while (cur > 1 and !states[cur].to[parLet]) {
                cur = states[cur].suffLink;
```

```
            }
            if (states[cur].to[parLet]) {
                cur = states[cur].to[parLet];
            }
            states[node].suffLink = cur;
        }
        if (node != 1)
            suffix_tree[states[node].suffLink].push_back(node);
            /// creates suffix link tree
        for (int i = 0 ; i < ALPHA; i++) {
            if (states[node].to[i]) {
                que.push(states[node].to[i]);
            }
        }
    }
}
int next_state(int from, int c) {
    if(states[from].nxt[c]) return states[from].nxt[c];
    int cur = from;
    while(cur > 1 and !states[cur].to[c]) cur =
        states[cur].suffLink;
    if(states[cur].to[c]) cur = states[cur].to[c];
    return states[from].nxt[c] = cur;
}
void dfs(int u) {
    for(int v : suffix_tree[u]) {
        dfs(v);
        states[u].cnt += states[v].cnt;
    }
}
}aho;


int main () {
    //freopen ("in.txt" , "r" , stdin);
    int tc, caseno = 1;
    cin >> tc;
    while(tc--) {
        aho.init();
        int n;
        cin >> n;
        string T;
        cin >> T;
        vector<int> node_id(n + 1, 0);
        for(int i = 1; i <= n; i++) {
            string s;
            cin >> s;
            node_id[i] = aho.add_string(s);
        }
        aho.push_links();
        int cur = 1;
        for(int i = 0; i < T.size(); i++) {
            cur = aho.next_state(cur, T[i]-'a');
            aho.states[cur].cnt++;
        }
        aho.dfs(1);
        cout << "Case " << caseno++ << ":\n";
        for(int i = 1; i <= n; i++) {
            cout << aho.states[node_id[i]].cnt << "\n";
        }
    }
}
```

6.5 kmp

```
vector<int> prefix_function (string s) {
        int n = (int) s.length(); vector<int> pi (n);
        for (int i=1; i<n; ++i) {
```

```
            int j = pi[i-1];
            while (j > 0 && s[i] != s[j]) j = pi[j-1];
            if (s[i] == s[j]) ++j;
            pi[i] = j;
        }
        return pi;
}
```

6.6 suffix$_{a}rray_{l}atest$

```
#include <bits/stdc++.h>
using namespace std ;

/**

sa[i] = i'th suffix, i from 0 to n-1
everything is in 0'th base
lcp[i] = lcp of (i-1)th and ith suffix, i from 0 to n-1
adjust the alpha, usually for string ALPHA = 128 (max ascii
    value)
notice if clearing may cause tle
remove range_lcp_init() if not required

**/

const int N = 2e4 + 5;
const int ALPHA = 128, LOG = 20;

struct SuffixArray {
    int sa[N],data[N],rnk[N],height[N],n;
    int wa[N],wb[N],wws[N],wv[N];
    int lg[N], rmq[N][LOG], rev_sa[N];
    int cmp(int *r,int a,int b,int l){
        return (r[a]==r[b]) && (r[a+l]==r[b+l]);
}
void DA(int *r,int *sa,int n,int m){
    int i,j,p,*x=wa,*y=wb,*t;
    for(i=0;i<m;i++) wws[i]=0;
    for(i=0;i<n;i++) wws[x[i]=r[i]]++;
    for(i=1;i<m;i++) wws[i]+=wws[i-1];
    for(i=n-1;i>=0;i--) sa[--wws[x[i]]]=i;
    for(j=1,p=1;p<n;j*=2,m=p) {
        for(p=0,i=n-j;i<n;i++) y[p++]=i;
        for(i=0;i<n;i++) if(sa[i]>=j) y[p++]=sa[i]-j;
        for(i=0;i<n;i++) wv[i]=x[y[i]];
        for(i=0;i<m;i++) wws[i]=0;
        for(i=0;i<n;i++) wws[wv[i]]++;
        for(i=1;i<m;i++) wws[i]+=wws[i-1];
        for(i=n-1;i>=0;i--) sa[--wws[wv[i]]]=y[i];
        for(t=x,x=y,y=t,p=1,x[sa[0]]=0,i=1;i<n;i++)
            x[sa[i]]=cmp(y,sa[i-1],sa[i],j)?p-1:p++;
    }
}
void calheight(int *r,int *sa,int n){
    int i,j,k=0;
    for(i=1;i<=n;i++) rnk[sa[i]]=i;
    for(i=0;i<n;height[rnk[i++]]=k)
        for(k?k--:0,j=sa[rnk[i]-1];r[i+k]==r[j+k];k++);
}
void suffix_array (string &A) {
    n = A.size();
    for(int i=0;i<max(n+5,ALPHA);i++)
        sa[i]=data[i]=rnk[i]=height[i]=wa[i]=wb[i]=wws[i]=wv[i]=0;
    for (int i = 0; i < n; i++) data[i] = A[i];
    DA(data,sa,n+1,ALPHA);
    calheight(data,sa,n);
    for(int i = 0;i < n; i++) sa[i] = sa[i+1], height[i] =
        height[i+1], rev_sa[sa[i]] = i;
    range_lcp_init();
```

```
    }

    /** LCP for range : build of rmq table **/
    void range_lcp_init() {
        for(int i = 0; i < n; i++) rmq[i][0] = height[i];
        for(int j = 1; j < LOG; j++) {
            for(int i = 0; i < n; i++) {
                if (i+(1<<j)-1 < n) rmq[i][j] =
                    min(rmq[i][j-1],rmq[i+(1<<(j-1))][j-1]);
                else break;
            }
        }
        lg[0] = lg[1] = 0;
        for(int i = 2; i <= n; i++) {
            lg[i] = lg[i/2] + 1;
        }
    }

    /** lcp between l'th to r'th suffix **/
    int query_lcp(int l, int r) {
        assert(l <= r);
        assert(l>=0 && l<n && r>=0 && r<n);
        if(l == r) return n-sa[l];
        l++;
        int k = lg[r-l+1];
        return min(rmq[l][k],rmq[r-(1<<k)+1][k]);
    }
}SA;

int main () {

    return 0;
}
```

7 data structure

7.1 1D bit

```
#include <bits/stdc++.h>
using namespace std;
/// 1D BIT

struct BIT {
    int n;
    vector<int> tree;
    void init(int n_) {
        n = n_;
        tree.assign(n+1,0);
    }
    void upd (int idx, int val) {
        while (idx <= n) {
            tree[idx] += val ;
            idx += idx & (-idx) ;
        }
    }
    int query (int idx) {
        int sum = 0 ;
        while (idx > 0) {
            sum += tree[idx] ;
            idx -= idx & (-idx) ;
        }
        return sum ;
    }
    int Sum(int i, int j) {
        return query(j)-query(i-1);
    }
};

int main() {
```

```
}
```

7.2 2D BIT range update+range query

```
const int mx = 1002,my = 1002;
long long bit[4][mx][my];
void update( int x, int y, int val, int i ) {
    int y1;
    while( x<=mx ) {
        y1=y;
        while( y1<=my ) {
            bit[i][x][y1] += val;
            y1 += (y1&-y1);
        }
        x += (x&-x);
    }
}
long long query( int x, int y, int i ) {
    long long ans=0; int y1;
    while( x>0 ) {
        y1 = y;
        while( y1>0 ) {
            ans += bit[i][x][y1];
            y1 -= (y1&-y1);
        }
        x -= (x&-x);
    }
    return ans;
}
// add value k from (x1,y1) to (x2,y2) inclusive
void add( int x1, int y1, int x2, int y2, int k )
{
    update(x1,y1,k,0);
    update(x1,y2+1,-k,0);
    update(x2+1,y1,-k,0);
    update(x2+1,y2+1,k,0);
    update(x1,y1,k*(1-y1),1);
    update(x1,y2+1,k*y2,1);
    update(x2+1,y1,k*(y1-1),1);
    update(x2+1,y2+1,-y2*k,1);
    update(x1,y1,k*(1-x1),2);
    update(x1,y2+1,k*(x1-1),2);
    update(x2+1,y1,k*x2,2);
    update(x2+1,y2+1,-x2*k,2);
    update(x1,y1,(x1-1)*(y1-1)*k,3);
    update(x1,y2+1,-y2*(x1-1)*k,3);
    update(x2+1,y1,-x2*(y1-1)*k,3);
    update(x2+1,y2+1,x2*y2*k,3);
}
// get value from (x1,y1) to (x2,y2) inclusive
long long get( int x1, int y1, int x2, int y2 )
{
  intl v1=query(x2,y2,0)*x2*y2+query(x2,y2,1)*x2+
          query(x2,y2,2)*y2+query(x2,y2,3);
  intl v2=query(x2,y1-1,0)*x2*(y1-1)+ query(x2,y1-1,1)*x2+
          query(x2,y1-1,2)*(y1-1)+query(x2,y1-1,3);
  intl v3=query(x1-1,y2,0)*(x1-1)*y2+query(x1-1,y2,1)*(x1-1)+
          query(x1-1,y2,2)*y2+query(x1-1,y2,3);
  intl v4=query(x1-1,y1-1,0)*(x1-1)*(y1-1)+
          query(x1-1,y1-1,1)*(x1-1)+
          query(x1-1,y1-1,2)*(y1-1)+query(x1-1,y1-1,3);
  intl ans=v1-v2-v3+v4;
  return ans;
}
```

7.3 2D bit

```
/// 2D BIT
/// careful with if sum exceeds int limit
#include <bits/stdc++.h>
using namespace std ;

const int N = 1e3 + 5;
int BIT[N][N];

void upd (int idx, int idy, int n, int val) { /// n = maximum
    row and col
    int temp = idy;
    while (idx <= n) {
        idy = temp;
        while (idy <= n) {
            BIT[idx][idy] += val;
            idy += idy & (-idy);
        }
        idx += idx & (-idx);
    }
}
int query (int idx, int idy) {
    int sum = 0, temp = idy;
    while (idx > 0) {
        idy = temp;
        while (idy > 0) {
            sum += BIT[idx][idy];
            idy -= idy & (-idy);
        }
        idx -= idx & (-idx);
    }
    return sum;
}

int SUM (int x1, int y1, int x2, int y2) {
    return query(x2,y2)-query(x1-1,y2) -
        query(x2,y1-1)+query(x1-1,y1-1);
}

int main() {

    return 0;
}
```

7.4 2D partial sum

```
/// 2D Partial SUM
/// x1,y1,x2,y2 >= 1 : this must be true

#include <bits/stdc++.h>
using namespace std;
int a[1001][101];

int main () {
    while (true) {
        int x1 , y1 , x2 , y2 , x ;
        cin >> x1 >> y1 >> x2 >> y2 >> x;
        if (x1 == 0) break;
        a[x1][y1] += x;
        a[x1][y2+1] -= x;
        a[x2+1][y1] -= x;
        a[x2+1][y2+1] += x;
    }
    for(int i = 1; i <= 5; i++) {
        for (int j = 1; j <= 5; j++) {
            a[i][j] += a[i][j-1] + a[i-1][j] - a[i-1][j-1] ;
            cout << a[i][j] << " " ;
        }
        cout << endl ;
```

```
    }
    return 0;
}
```

7.5 HLD (solaiman)

```
/*
    Heavy-Light Decomposition
    1. flat[] (0-indexed) has the flattened array of the tree
       according
       to the decomposition into chains
    2. flatIdx[] is the reverse map of flat[]
    3. There will be O(logN) segments of chains between node u &
       v.
    4. getChainSegments(u, v) calculates these in O(logN) time
    5. dfs(u, p) & HLD(u, p) are essential. The rest of the
       functions
       are auxiliary
*/

#include<bits/stdc++.h>
using namespace std;
typedef pair<int,int>PII;
const int MAXN = 500007;
const int LOGN = 20;
vector<int>edg[MAXN];

int sbtr[MAXN], lvl[MAXN], pr[MAXN][LOGN];
int chainIdx[MAXN], chainHead[MAXN], flatIdx[MAXN], flat[MAXN];
int chainCnt, flatCnt;

void dfs(int u, int p)
{
    lvl[u] = lvl[p] + 1;
    pr[u][0] = p;
    for (int k = 1; k < LOGN; k++) {
        pr[u][k] = pr[pr[u][k-1]][k-1];
    }

    sbtr[u] = 1;
    for (int v : edg[u]) {
        if (v==p) continue;
        dfs(v, u);
        sbtr[u] += sbtr[v];
    }
}

/// auxiliary function
int getLCA(int u, int v)
{
    if (lvl[u] < lvl[v]) swap(u, v);
    for (int k = LOGN-1; k >= 0; k--) {
        if (lvl[u]-(1<<k) >= lvl[v]) {
            u = pr[u][k];
        }
    }
    if (u==v) return u;
    for (int k = LOGN-1; k >= 0; k--) {
        if (pr[u][k] != pr[v][k]) {
            u = pr[u][k];
            v = pr[v][k];
        }
    }
    return pr[u][0];
}

void HLD(int u, int p)
{
```

```
    chainIdx[u] = chainCnt;
    flatIdx[u] = flatCnt;
    flat[flatCnt] = u;
    flatCnt++;

    int biggie = -1, mx = 0;
    for (int v : edg[u]) {
        if (v==p) continue;
        if (mx < sbtr[v]) {
            mx = sbtr[v];
            biggie = v;
        }
    }
    if (biggie==-1) return;

    HLD(biggie, u);
    for (int v : edg[u]) {
        if (v==p||v==biggie) continue;
        chainCnt++;
        chainHead[chainCnt] = v;
        HLD(v, u);
    }
}

/// upSegments(l, u, vp) add segments for (l, u] to vp vector
/// provided l is an ancestor of u
void upSegments(int l, int u, vector<PII>&vp)
{
    while (chainIdx[l] != chainIdx[u]) {
        int uhead = chainHead[chainIdx[u]];
        vp.push_back(PII(flatIdx[uhead], flatIdx[u]));
        u = pr[uhead][0];
    }
    if (l!=u) {
        vp.push_back(PII(flatIdx[l]+1, flatIdx[u]));
    }
}

vector<PII>getChainSegments(int u, int v)
{
    int l = getLCA(u, v);
    vector<PII>rt;
    rt.push_back(PII(flatIdx[l], flatIdx[l]));
    if (u==v) return rt;
    upSegments(l, u, rt);
    upSegments(l, v, rt);
    return rt;
}

PII getSubtreeSegment(int u) {
    return PII(flatIdx[u], flatIdx[u]+sbtr[u]-1);
}

void performHLD(int root)
{
    dfs(root, 0);
    chainCnt = 0;
    flatCnt = 0;
    chainHead[0] = root;
    HLD(root, 0);
}

int main()
{
    int n;
    cin >> n;

    for (int i = 1; i < n; i++) {
        int u, v;
        cin >> u >> v;
```

```
        edg[u].push_back(v);
        edg[v].push_back(u);
    }

    performHLD(1);

    for (int i = 0; i < n; i++) cout << flat[i] << ' ';


    return 0;
}
```

7.6 HLD

```
/// HLD from ARGHYA, yet to get used to

#include <bits/stdc++.h>

#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>

#define sf scanf
#define pf printf
#define pb push_back
#define mp make_pair
#define IN freopen("transposition-115.txt","r",stdin)
#define OUT freopen("dichromatic.out","w",stdout)
#define FOR(i,a,b) for(i=a ; i<=b ; i++)
#define DBG pf("Hi\n")
#define INF 2000000000
#define i64 long long int
#define eps (1e-8)
#define xx first
#define yy second
#define sq(x) ((x)*(x))

using namespace __gnu_pbds;
using namespace std ;

#define maxn (1<<18)+5
#define mod 1000000007LL

typedef pair<i64,i64> pii ;
typedef long long int T ;

struct edge{
    int u , v , c , id ;
    bool operator<(const edge other)const{ return c < other.c ; }
};

namespace MST{
    int par[maxn] ;
    int findPar(int u)
    {
        if( par[u] != u ) return par[u] = findPar( par[u] ) ;
        return u ;
    }

    void findMST( int n , vector < edge > &e )
    {
        //after this function , e[0] to e[n-2] will contain the
           treeEdges
        //and other would be non-tree Edges

        sort( e.begin() , e.end() ) ;
        for(int i=1 ; i<=n ; i++) par[i] = i ;

        vector <edge> treeEdge , otherEdge ;
```

```
        for(int i=0 ; i< e.size() ; i++ )
        {
            int u = e[i].u , v = e[i].v ;
            int pu = findPar(u) , pv = findPar(v) ;
            if( pu==pv ) otherEdge.pb(e[i]) ;
            else{
                par[pu] = pv ;
                treeEdge.pb(e[i]) ;
            }
        }
        e.clear() ;
        for(int i=0 ; i<treeEdge.size() ; i++) e.pb( treeEdge[i]
            ) ;
        for(int i=0 ; i<otherEdge.size() ; i++) e.pb(
            otherEdge[i] ) ;
    }
}

int tr[2*maxn] , lazy[2*maxn] , ara[maxn] ;

void relax(int cn, int b, int e)
{
    tr[cn] = min( tr[cn] , lazy[cn] ) ;
    if( b!=e )
    {
        int lc = cn<<1 , rc = lc+1 , m = (b+e)>>1 ;
        lazy[lc] = min( lazy[cn] , lazy[lc] ) ;
        lazy[rc] = min( lazy[cn] , lazy[rc] ) ;
    }
    lazy[cn] = INF ;
}

void update(int cn, int b , int e, int l, int r,int val)
{
    if( lazy[cn] != INF ) relax(cn,b,e) ;
    if( e < l || b > r ) return ;
    if( l<=b && e<=r )
    {
        lazy[cn] = val ;
        relax(cn,b,e) ;
        return ;
    }
    int lc = cn<<1 , rc = lc+1 , m = (b+e)>>1 ;
    update(lc,b,m,l,r,val) ; update(rc,m+1,e,l,r,val) ;
    tr[cn] = max(tr[lc],tr[rc]) ;
}

int query(int cn, int b, int e, int l, int r)
{
    if(lazy[cn]!= INF ) relax(cn,b,e) ;
    if( e < l || b > r ) return -INF;
    if( l<=b && e<=r ) return tr[cn] ;
    int lc = cn<<1 , rc = lc+1 , m = (b+e)>>1 ;
    return max( query(lc,b,m,l,r) , query(rc,m+1,e,l,r) ) ;
}

void build(int cn, int b, int e)
{
    lazy[cn] = INF ;
    if( b==e )
    {
        tr[cn] = ara[b] ;
        return ;
    }
    int lc = cn<<1 , rc = lc+1 , m = (b+e)>>1 ;
    build(lc,b,m) ; build(rc,m+1,e) ;
    tr[cn] = max(tr[lc],tr[rc]) ;
}

namespace hld{
```

```
int in[maxn] , out[maxn] , sub[maxn] , t = 1 , nxt[maxn] ,
    depth[maxn], par[maxn] , n ;
vector <int> g[maxn] ;
void init(int _n)
{
    n = _n ;
    for(int i=0 ; i<=n ; i++) g[i].clear() ;
}

void addEdge(int u, int v)
{
    g[u].pb(v) ;
    g[v].pb(u) ;
}

void dfsSZ(int u)
{
    sub[u] = 1 ;

    for(int i=0 ; i<g[u].size() ; i++)
    {
        int v = g[u][i] ;
        for(int j=0 ; j<g[v].size() ; j++)
        {
            if( g[v][j] == u )
            {
                g[v].erase( g[v].begin() + j ) ;
                break ;
            }
        }
        dfsSZ(v) ;
        sub[u] += sub[v] ;
        if( sub[v] > sub[ g[u][0] ] ) swap( g[u][0] , g[u][i]
            ) ;
    }
}

void dfsHLD(int u)
{
    in[u] = ++t ;
    for(int i=0 ; i<g[u].size() ; i++)
    {
        int v = g[u][i] ;
        par[v] = u ;
        depth[v] = depth[u] + 1 ;
        if( i==0 ) nxt[v] = nxt[u] ;
        else nxt[v] = v ;

        dfsHLD(v) ;
    }
    out[u] = t ;
}

void preprocess(int root)
{
    dfsSZ(root) ;
    t = 0 ; nxt[root] = root ;
    depth[root] = 1 ;
    dfsHLD(root) ;
}

int hldQuery( int u , int v )
{
    int ans = -INF ;
    while( nxt[u] != nxt[v] )
    {
        if( depth[ nxt[u] ] < depth[ nxt[v] ] )
        {
            ans = max( ans , query(1,1,n, in[ nxt[v] ] , in[v]
                ) ) ;
```

```
            // do you thing here ( from in[v] to in[ nxt[v] ] )
            v = par[ nxt[v] ] ;
        }
        else{
            ans = max( ans , query(1,1,n, in[ nxt[u] ] , in[u]
                ) ) ;
            // do your thing here ( from in[u] to in[ nxt[u] ]
                )
            u = par[ nxt[u] ] ;
        }
    }
    int lc ;
    if( depth[u] > depth[v] ) swap(u,v) ;
    lc = u ;

    //here lc is the lca
    //if you are working on node , not on edge, then
        update/query upto u also
    //otherwise update/query from in[u]+1 to in[v]

    ans = max( ans , query(1,1,n,in[u]+1,in[v]) ) ;

    return ans ;
}

void hldUpdate( int u , int v , int val )
{
    while( nxt[u] != nxt[v] )
    {
        if( depth[ nxt[u] ] < depth[ nxt[v] ] )
        {
            update(1,1,n,in[ nxt[v] ] , in[v] , val ) ;
            // do you thing here ( from in[v] to in[ nxt[v] ] )
            v = par[ nxt[v] ] ;
        }
        else{
            update(1,1,n,in[ nxt[u] ] , in[u] , val ) ;
            // do your thing here ( from in[u] to in[ nxt[u] ]
                )
            u = par[ nxt[u] ] ;
        }
    }
    int lc ;
    if( depth[u] > depth[v] ) swap(u,v) ;
    lc = u ;

    //here lc is the lca
    //if you are working on node , not on edge, then
        update/query upto u also
    //otherwise update/query from in[u]+1 to in[v]

    update(1,1,n,in[u]+1,in[v],val) ;

    return ;
}

}

int ans[maxn] ;

int main()
{
    int n , m ;

    vector < edge > e ;

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

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

```
    int u ,v , c ;
    scanf("%d %d %d",&u,&v,&c) ;
    e.pb( {u,v,c,i} ) ;
}

MST::findMST( n , e ) ;

hld::init(n) ;

for(int i=0 ; i<n-1 ; i++)
{
    hld::addEdge( e[i].u , e[i].v ) ;
}

hld::preprocess(1) ;

for(int i=0 ; i<n-1 ; i++)
{
    int u = e[i].u , v = e[i].v , c= e[i].c ;

    if( hld::depth[u] > hld::depth[v] ) swap(u,v) ;

    ara[ hld::in[v] ] = e[i].c ;
}

ara[ hld::in[1] ] = INF ;
build(1,1,n) ;

for(int i=n-1 ; i<m ; i++)
{
    int u = e[i].u , v = e[i].v , c= e[i].c ;

    ans[ e[i].id ] = hld::hldQuery(u,v) - 1 ;
}
```

```
    for(int i=1 ; i<=n ; i++) ara[i] = INF ;
    build(1,1,n) ;

    for(int i=n-1 ; i<m ; i++)
    {
        int u = e[i].u , v = e[i].v , c= e[i].c ;
        hld::hldUpdate(u,v,c) ;
    }

    for(int i=0 ; i<n-1 ; i++)
    {
        int u = e[i].u , v = e[i].v , c= e[i].c ;
        int res = hld::hldQuery(u,v) ;
        if(res==INF) ans[ e[i].id ] = -1 ;
        else ans[ e[i].id ] = res-1 ;
    }

    for(int i=1 ; i<=m ; i++) printf("%d ",ans[i]) ;
    printf("\n") ;

    return 0 ;
}
```

7.7 LCA

```
#include <bits/stdc++.h>
using namespace std;

const int N = 1e5 + 5 , LOG = 20 ;
int par[N][LOG] , depth[N] ;
vector <int> g[N] ;

void dfs (int u , int p , int lvl) {
    par[u][0] = p ;
```

```
    depth[u] = lvl ;
    for (int i = 0 ; i < g[u].size() ; i++) {
        int v = g[u][i] ;
        if (v == p) continue ;
        dfs (v,u,lvl+1) ;
    }
}


void init(int root , int n) {
    dfs(root,-1,1) ;
    for (int k = 1 ; k < LOG ;k++) {
        for (int i = 1 ; i <= n ; i++) {
            if (par[i][k-1] != -1)
                par[i][k] = par[par[i][k-1]][k-1] ;
            else
                par[i][k] = -1 ;
        }
    }
}


int lca (int u , int v) {
    if (depth[u] < depth[v]) { /// u nichey
        swap(u,v) ;
    }
    int diff = depth[u] - depth[v] ;
    for (int i = LOG-1 ; i >= 0 ; i--) {
        if (diff >= (1<<i)) {
            diff -= (1<<i) ;
            u = par[u][i] ;
        }
    }
    if (u == v) return u ;
    for (int i = LOG-1 ; i >= 0 ; i--) {
        if (par[u][i] != par[v][i]) {
            u = par[u][i] ;
            v = par[v][i] ;
        }
    }
    return par[v][0] ;
}

int kth_par(int u , int k) {
    for (int i = LOG-1 ; i >= 0 ; i--) {
        if (k >= (1<<i)) {
            k -= (1<<i) ;
            u = par[u][i] ;
        }
        if (u == -1) return u ;
    }
    return u ;
}


int main () {

    return 0 ;
}
```

7.8 Lca min max cost on edges

```
/// cost on Edge , finds max and min
#include <bits/stdc++.h>
using namespace std ;

const int N = 1e5 + 5 , LOG = 20 ;
const long long inf = 1e18 ;
```

```
struct LCA {
    vector < pair<int,long long> > G[N] ;
    int par[N][LOG] , depth[N] ;
    long long mx[N][LOG] , mn[N][LOG] ;
    void init() {
        for(int i = 1 ; i < N ; i++) G[i].clear() ;
    }
    void addEdge(int u , int v , long long w) { /// adds
        directed edge
        G[u].push_back(make_pair(v,w)) ;
    }
    void lca_init(int root) {
        memset(par,-1,sizeof par) ;
        dfs(root,-1,0,1) ;
    }
    void dfs (int u , int p , long long w , int l) {
        par[u][0] = p ;
        if (u==1) mx[u][0] = -inf , mn[u][0] = inf;
        else mx[u][0] = mn[u][0] = w ;
        depth[u] = l ;
        for (int k = 1 ; k < LOG ; k++) {
            if (par[u][k-1] != -1) {
                par[u][k] = par[par[u][k-1]][k-1];
                mx[u][k] = max(mx[u][k-1],mx[par[u][k-1]][k-1]) ;
                mn[u][k] = min(mn[u][k-1],mn[par[u][k-1]][k-1]) ;
            }
            else {
                mx[u][k] = mx[u][k-1] ;
                mn[u][k] = mn[u][k-1] ;
            }
        }
        for (int i = 0 ; i < G[u].size() ; i++) {
            int v = G[u][i].first ;
            long long w_ = G[u][i].second ;
            if (v != p) {
                dfs(v,u,w_,l+1) ;
            }
        }
    }

    pair<long long,long long> solve (int u , int v) {
        if (depth[u] < depth[v]) {
            swap(u,v) ;
        }
        long long maxi = -inf , mini = inf ;
        int diff = depth[u] - depth[v] ;
        for (int i = LOG-1 ; i >= 0 ; i--) {
            if (diff >= (1<<i)) {
                diff -= (1<<i) ;
                maxi = max(maxi,mx[u][i]) ;
                mini = min(mini,mn[u][i]) ;
                u = par[u][i] ;
            }
        }
        if (u == v) return make_pair(mini,maxi) ;
        for (int i = LOG-1 ; i >= 0 ; i--) {
            if (par[u][i] != par[v][i]) {
                mini = min(mini,mn[u][i]) ;
                maxi = max(maxi,mx[u][i]) ;
                u = par[u][i] ;

                mini = min(mini,mn[v][i]) ;
                maxi = max(maxi,mx[v][i]) ;
                v = par[v][i] ;
            }
        }
        maxi = max(maxi,mx[u][0]) ;
        maxi = max(maxi,mx[v][0]) ;
        mini = min(mini,mn[u][0]) ;
        mini = min(mini,mn[v][0]) ;
```

```
        return make_pair(mini,maxi) ;
    }
}Tr;

int main () {
    int tc , caseno = 1;
    cin >> tc ;
    while (tc--) {
        Tr.init() ;
        int n ;
        scanf("%d",&n) ;
        for (int i = 1 ; i < n; i++) {
            int u , v ; long long w ;
            scanf("%d %d %lld",&u , &v , &w) ;
            Tr.addEdge(u,v,w) ;
            Tr.addEdge(v,u,w) ;
        }
        Tr.lca_init(1) ;
        int q ;
        cin >> q ;
        printf ("Case %d:\n",caseno++) ;
        while (q--) {
            int u , v ; scanf("%d %d",&u,&v) ;
            pair<long long,long long> ans = Tr.solve(u,v) ;
            printf ("%lld %lld\n",ans.first,ans.second);
        }
    }
}
```

7.9 Mo on Tree

```
/* Bubble Cup X - Finals [ Online Mirror ] */
/* I - Dating */
/* Mo on trees */

#include <bits/stdc++.h>
using namespace std;

const int N = 2e5 + 5 , BLOCK = 320 , LOG = 20;
int n , a[N] , taken[N] , node[N] , depth[N] , mof[N] , f[N] ,
    st[N] , en[N] , par[N][LOG] ;
long long res[N] , cnt[N][2] , ans ;
vector <int> g[N] ;
int Timer = 0 ;
struct Query {
    int u , v , id , l , r;
    bool operator <(const Query &p) const{
                int a = l/BLOCK, b = p.l/BLOCK;
                return a == b ? r < p.r : a < b;
        }
}Q[N];

void dfs (int u , int p , int lvl) {
    par[u][0] = p ;
    depth[u] = lvl ;
    st[u] = ++Timer ;
    node[Timer] = u ;
    for (auto v:g[u]) {
        if (v == p) continue ;
        dfs(v,u,lvl+1) ;
    }
    en[u] = ++Timer ;
    node[Timer] = u ;
}

void init(int n) {
    for (int k = 1 ; k < LOG ;k++) {
        for (int i = 1 ; i <= n ; i++) {
            if (par[i][k-1] != -1)
```

```
                par[i][k] = par[par[i][k-1]][k-1] ;
            else
                par[i][k] = -1 ;
        }
    }
}


int lca (int u , int v) {
    if (depth[u] < depth[v]) {
        swap(u,v) ;
    }
    int diff = depth[u] - depth[v] ;
    for (int i = LOG-1 ; i >= 0 ; i--) {
        if (diff >= (1<<i)) {
            diff -= (1<<i) ;
            u = par[u][i] ;
        }
    }
    if (u == v) return u ;
    for (int i = LOG-1 ; i >= 0 ; i--) {
        if (par[u][i] != par[v][i]) {
            u = par[u][i] ;
            v = par[v][i] ;
        }
    }
    return par[v][0] ;
}



int compress() {
    vector <int> v ; map<int,int> Map ;
    for (int i = 1 ; i <= n ; i++) {
        v.push_back(f[i]) ;
    }
    sort (v.begin(),v.end()) ;
    v.erase(unique(v.begin(),v.end()),v.end()) ;
    for (int i = 0 ; i < v.size() ; i++) {
        Map[v[i]] = i+1 ;
    }
    for (int i = 1 ; i <= n ; i++) {
        f[i] = Map[f[i]] ;
    }
    return (int)v.size() ;
}

void add (int t) {
    int u = node[t] ;
    if (taken[u]) {
        cnt[f[u]][mof[u]]-- ;
        ans -= cnt[f[u]][mof[u]^1] ;
    }
    else {
        cnt[f[u]][mof[u]]++;
        ans += cnt[f[u]][mof[u]^1] ;
    }
    taken[u] ^= 1 ;
}

/// del function lage na
void del (int t) {
    int u = node[t] ;
    if (taken[u]) {
        cnt[f[u]][mof[u]]-- ;
        ans -= cnt[f[u]][mof[u]^1] ;
    }
    else {
        cnt[f[u]][mof[u]]++ ;
        ans += cnt[f[u]][mof[u]^1] ;
```

```
    }
    taken[u] ^= 1 ;
}
int main () {
   // freopen ("in.txt" , "r" , stdin) ;
    scanf ("%d" , &n) ;
    for (int i = 1 ; i <= n ; i++) {
        scanf ("%d" , &mof[i]) ;
    }
    for (int i = 1 ; i <= n ; i++) {
        scanf ("%d" , &f[i]) ;
    }
    compress() ;
    for (int i = 1 ; i < n ; i++) {
        int u , v ;
        scanf ("%d %d" , &u , &v) ;
        g[u].push_back(v) ; g[v].push_back(u) ;
    }
    dfs(1,-1,1) ;
    init(n) ;
    int q ; scanf ("%d" , &q) ;
    for (int i = 1 ; i <= q; i++) {
        int u , v ;
        scanf ("%d %d" , &u , &v) ;
        if (st[u] > st[v]) swap(u,v) ;
        if (st[u] <= st[v] and en[u] >= en[v]) {
            Q[i] = {u,v,i,st[u],st[v]} ;
        }
        else {
            Q[i] = {u,v,i,en[u],st[v]} ;
        }
    }
    sort (Q+1,Q+q+1) ;
    int L = 1 , R = 0 ;
    for (int i = 1 ; i <= q ; i++) {
        int u = Q[i].u , v = Q[i].v ;
        int ql = Q[i].l , qr = Q[i].r ;
        while (L < ql) del(L++) ;
        while (L > ql) add(--L) ;
        while (R > qr) del(R--) ;
        while (R < qr) add(++R) ;
        int anc = lca(u,v) ;
        res[Q[i].id] = ans ;
        if (anc != u and anc != v) {
            res[Q[i].id] += cnt[f[anc]][mof[anc]^1] ;
        }
    }
    for (int i = 1 ; i <= q ; i++) {
        printf ("%lld\n" , res[i]) ;
    }
    return 0 ;
}
```

7.10 Mo's Algorithm

```
/// Mo's Algorithm Template
#include <bits/stdc++.h>
using namespace std;

typedef long long ll;

const int N = 3e5 + 100, BLOCK = 600;

struct Query {
    int l , r , id;
    bool operator <(const Query &p) const{
                int a = l/BLOCK, b = p.l/BLOCK;
                return a == b ? r < p.r : a < b;
        }
```

```
}Q[N];

void add (int idx) {}
void del (int idx) {}

int main() {
    ios::sync_with_stdio(0);
    cin.tie(0);
    int q; cin >> q;
    for (int i = 1 ; i <= q ; i++) {
        cin >> Q[i].l >> Q[i].r;
        Q[i].id = i;
    }
    sort (Q+1,Q+1+q);
    int l = 1, r = 0;
    for (int i = 1 ; i <= q ; i++) {
        int ql = Q[i].l, qr = Q[i].r, id = Q[i].id;
        while (l < ql) del(l++);
        while (l > ql) add(--l);
        while (r > qr) del(r--);
        while (r < qr) add(++r);
    }
    return 0;
}
```

7.11 Persistent Segment Tree

```
#include <bits/stdc++.h>
using namespace std;

/// not tested, persistent segtree for range sum

const int N = 1e5 + 5, LOG = 20;
int a[N], root[N];

struct node {
    int sum, l, r;
}node[N*LOG];

int tot_nodes = 0;

int upd(int cn, int b, int e, int i, int val) {
    int cur = ++tot_nodes;
    if (b == e) {
        node[cur].sum = node[cn].sum + val;
        return cur;
    }
    int mid = (b+e)/2;
    if (i <= mid) {
        node[cur].l = upd(node[cn].l, b, mid, i, val);
        node[cur].r = node[cn].r;
    }
    else {
        node[cur].r = upd(node[cn].r, mid+1, e, i, val) ;
        node[cur].l = node[cn].l;
    }
    node[cur].sum = node[node[cur].l].sum +
        node[node[cur].r].sum ;
    return cur;
}

int query(int cn , int b , int e , int i, int j) {
    if (b > j or e < i or !cn) return 0 ;
    if (b >= i and e <= j) {
        return node[cn].sum;
    }
    int mid = (b+e)/2;
    return query(node[cn].l,b,mid,i,j) +
        query(node[cn].r,mid+1,e,i,j);
```

```
}

int main () {

    return 0 ;

    /// FOR Creating a New version , root[i] =
        upd(root[i-1],1,n,idx,val) ;
}
```

7.12 RMQ(1D)

```
#include <bits/stdc++.h>
using namespace std;

const int N = 1e5 + 100, M = 5005, LOG = 20;

int rmq[N][LOG];
int n, a[N];
int lg[N];

void preprocess() {
    for(int i = 1; i <= n; i++) rmq[i][0] = a[i] ;
    for(int j = 1; j < LOG; j++) {
        for(int i = 1; i <= n; i++) {
            if (i+(1<<j)-1<=n) {
                rmq[i][j] =
                    max(rmq[i][j-1],rmq[i+(1<<(j-1))][j-1]) ;
            }
            else break ;
        }
    }
    for(int i = 2; i < N; i++) {
        lg[i] = lg[i/2] + 1;
    }
}

int query (int i , int j) {
    int k = lg[j-i+1];
    int ans = max(rmq[i][k],rmq[j-(1<<k)+1][k]);
    return ans;
}
```

7.13 RMQ(2D)

```
#include <bits/stdc++.h>
using namespace std ;

const int ln = 9 ;
int rmq[1005][1005][10][10];
int n , m , a[1005][1005],Log[1050]; /// a is the given matrix

void preprocess () {
    FOR(i,0,1004) {
        int j = 0 ;
        while(1<<(j+1)<=i) j++;
        Log[i] = j ;
    }
    FOR(i,0,ln) {
        FOR(j,0,ln) {
            FOR(x,1,n) {
                if (x+(1<<i)-1>n) break;
                FOR(y,1,m) {
                    if (y+(1<<j)-1>m) break;
                    if (i==0 and j==0) {
                        rmq[x][y][0][0] = a[x][y] ;
                    }
                    else if (i==0) {
```

```
                int yh = y + (1<<(j-1)) ;
                rmq[x][y][0][j] =
                    max(rmq[x][y][0][j-1],rmq[x][yh][0][j-1])
                    ;
            }
            else if (j==0) {
                int xh = x + (1<<(i-1)) ;
                rmq[x][y][i][0] =
                    max(rmq[x][y][i-1][0],rmq[xh][y][i-1][0])
                    ;
            }
            else {
                int xh = x + (1<<(i-1)) , yh = y +
                    (1<<(j-1)) ;
                rmq[x][y][i][j] =
                    max(rmq[x][y][i][j-1],rmq[x][yh][i][j-1])
                    ;
            }
        }
    }
}

int query(int x1,int y1,int x2,int y2) {   /// gives maximum of
    matrix with upper left point (x1,y1) and lower right point
    (x2,y2)
    int lx = x2-x1+1 , ly = y2-y1+1 , kx = 0 , ky = 0;
    kx = Log[lx] , ky = Log[ly] ;
    x2 = x2+1-(1<<kx) , y2 = y2+1-(1<<ky) ;
    return
        max({rmq[x1][y1][kx][ky],rmq[x1][y2][kx][ky],rmq[x2][y1][kx][ky],rmq[x2][y2][kx][ky]})
        ;
}
```

```
int main () {

    //IN ;

    cin >> n >> m ;
    FOR(i,1,n) FOR(j,1,m) si(a[i][j]) ;

    return 0 ;
}
```

7.14 centroid decomposition template

```
/// centroid decomposition template

#include <bits/stdc++.h>
using namespace std;

const int N = 2e5 + 100;

vector <int> g[N];
int n;
int child[N];
bool done[N];

void dfs_size(int u, int par) {
    child[u] = 1;
    for (int v: g[u]) {
        if (done[v] or v == par) continue;
        dfs_size(v, u);
        child[u] += child[v];
    }
}

int dfs_find_centroid(int u, int par, int sz) {
```

```
    for (int v: g[u]) {
        if (!done[v] and v != par and child[v] > sz) {
            return dfs_find_centroid(v,u,sz);
        }
    }
    return u;
}

void solve (int u) {
    /// problem specific things to do
}


void dfs_decompose(int u) {
    dfs_size(u, -1);
    int centroid = dfs_find_centroid(u, -1, child[u]/2);
    solve(centroid);
    done[centroid] = 1;
    for (int v : g[centroid]) {
        if (!done[v]) dfs_decompose(v);
    }
}


int main () {
    ios::sync_with_stdio(0);
    cin.tie(0);

    int n;
    cin >> n;
    for(int i = 1; i < n; i++) {
        int u, v;
        cin >> u >> v;
        g[u].push_back(v);
        g[v].push_back(u);
    }
    dfs_decompose(1);
}
```

7.15 lcaO(1) arghya

```
#define MAX 100010
#define LOG 18
namespace LCA{
  i64 sum[MAX] ; int st[MAX],en[MAX],lg[MAX],par[MAX],a[MAX],
  id[MAX],dp[LOG][MAX] ;
  vector <int> weight[MAX] , g[MAX]; int n , r , Time , cur ;
  void init(int nodes, int root){
    n = nodes, r = root, lg[0] = lg[1] = 0;
    for(int i = 2; i <= n; i++) lg[i] = lg[i >> 1] + 1;
    for(int i=0;i<= n;i++) g[i].clear(), weight[i].clear();
  }
  void addEdge(int u, int v, int w){
    g[u].push_back(v), weight[u].push_back(w);
    g[v].push_back(u), weight[v].push_back(w);
  }
  int lca(int u, int v){
    if( en[u] > en[v] )swap(u,v) ;
    if( st[v] <= st[u] && en[u] <= en[v] ) return v ;
    int l = lg[id[v] - id[u] + 1] ;
    int p1 = id[u] , p2 = id[v] - (1<<l) + 1 ;
    if(sum[dp[l][p1]]<sum[dp[l][p2]]) return par[dp[l][p1]] ;
    else return par[ dp[l][p2] ] ;
  }
  i64 dis( int u ,int v ){
    int l = lca(u,v) ;
    return (sum[u] + sum[v] - ( sum[l]<<1LL )) ;
  }
  void dfs(int u, int p , i64 curSum){
```

```
    st[u] = ++Time ; par[u] = p ; sum[u] = curSum ;
    for(int i=0 ; i<g[u].size() ; i++){
      if( g[u][i]==p ) continue ;
      dfs( g[u][i] ,u,curSum+weight[u][i]) ;
    }
    en[u] = ++Time ; a[++cur] = u ; id[u] = cur ;
  }
  void build(){
    cur = Time = 0 ; dfs( r , r , 0 );
    for(int i=1 ; i<=n ; i++) dp[0][i] = a[i] ;
    for(int l=0 ; l<LOG-1 ; l++) {
      for(int i=1 ; i<=n ; i++) {
        dp[l+1][i] = dp[l][i] ;
        if( (1<<l)+i <= n && sum[dp[l][i+(1<<l)]] <
           sum[dp[l][i]]) dp[l+1][i] = dp[l][ i+(1<<l)] ;
      }
    }
  }
}
```

7.16 link cut tree (solaiman)

```
/*
 Petar 'PetarV' Velickovic
 Data Structure: Link/cut Tree
 Source:
    https://github.com/PetarV-/Algorithms/blob/master/Data%20Structures/Link-cut%20Tree.cpp
*/

#include <bits/stdc++.h>
using namespace std;

/*
The link/cut tree data structure enables us to efficiently
    handle a dynamic forest of trees.
It does so by storing a decomposition of the forest into
    "preferred paths", where a path is
preferred to another when it has been more recently accessed.
Each preferred path is stored in a splay tree which is keyed by
    depth.
The tree supports the following operations:
    - make_tree(v): create a singleton tree containing the node
        v
    - find_root(v): find the root of the tree containing v
    - link(v, w):  connect v to w
                   (precondition: v is root of its own tree,
                   and v and w are not in the same tree!)
    - cut(v):      cut v off from its parent
    - path(v):     access the path from the root of v's tree to v
                   (in order to e.g. perform an aggregate query
                       on that path)
More complex operations and queries are possible that require
    the data structure
to be augmented with additional data. Here I will demonstrate
    the LCA(p, q)
(lowest common ancestor of p and q) operation.
Complexity:   O(1) for make_tree
              O(log n) amortized for all other operations
*/



const int MAXN = 100007;

struct LinkCutTree {
    struct Node {
        int L, R, P, lazyFlip;
        int PP;
```

```
};

Node LCT[MAXN];

void normalize(int u) {
    assert(u != -1);
    if (LCT[u].L != -1) LCT[LCT[u].L].P = u;
    if (LCT[u].R != -1) LCT[LCT[u].R].P = u;
    // (+ update sum of subtree elements etc. if wanted
}

// set v as p's left child
void setLeftChild(int p, int v) {
    LCT[p].L = v;
    normalize(p);
}
// set v as p's right child
void setRightChild(int p, int v) {
    LCT[p].R = v;
    normalize(p);
}

void pushLazy(int u) {
    if (!LCT[u].lazyFlip) return;
    swap(LCT[u].L, LCT[u].R);
    LCT[u].lazyFlip = 0;
    if (LCT[u].L != -1) LCT[LCT[u].L].lazyFlip ^= 1;
    if (LCT[u].R != -1) LCT[LCT[u].R].lazyFlip ^= 1;
}

void make_tree(int v) {
    LCT[v].L = LCT[v].R = LCT[v].P = LCT[v].PP = -1;
    LCT[v].lazyFlip = 0;
}
```

```
    void rotate(int v) {
        if (LCT[v].P == -1) return;
        int p = LCT[v].P;
        int g = LCT[p].P;
        if (LCT[p].L == v) {
            setLeftChild(p, LCT[v].R);
//            LCT[p].L = LCT[v].R;
//            if (LCT[v].R != -1) {
//                LCT[LCT[v].R].P = p;
//            }
            setRightChild(v, p);
//            LCT[v].R = p;
//            LCT[p].P = v;
        } else {
            setRightChild(p, LCT[v].L);
//            LCT[p].R = LCT[v].L;
//            if (LCT[v].L != -1) {
//                LCT[LCT[v].L].P = p;
//            }
            setLeftChild(v, p);
//            LCT[v].L = p;
//            LCT[p].P = v;
        }

        LCT[v].P = g;
        if (g != -1) {
            if (LCT[g].L == p) {
                setLeftChild(g, v);
//                LCT[g].L = v;
            } else {
                setRightChild(g, v);
//                LCT[g].R = v;
            }
        }
```

```
        // must preserve path-pointer!
        // (this only has an effect when g is -1)
        LCT[v].PP = LCT[p].PP;
        LCT[p].PP = -1;
    }

    void pushEmAll(int v) {
        if (LCT[v].P != -1) pushEmAll(LCT[v].P);
        pushLazy(v);
    }

    void splay(int v) {
//      cout << "splay " << v << endl;
        pushEmAll(v);
        while (LCT[v].P != -1) {
            int p = LCT[v].P;
            int g = LCT[p].P;
            if (g == -1) { // zig
                rotate(v);
            } else if ((LCT[p].L == v) == (LCT[g].L == p)) { //
                zig-zig
                rotate(p);
                rotate(v);
            } else { // zig-zag
                rotate(v);
                rotate(v);
            }
        }
    }

    /// returns v if v is in the root auxiliary tree
    /// otherwise returns the topmost unpreferred edge's parent
    int access(int v) {
        splay(v); // now v is root of its aux. tree
        if (LCT[v].R != -1) {
            LCT[LCT[v].R].PP = v;
            LCT[LCT[v].R].P = -1;
            setRightChild(v, -1);
//            LCT[v].R = -1;
        }

        int ret = v;
        while (LCT[v].PP != -1) {
            int w = LCT[v].PP;
            splay(w);
            if (LCT[w].PP == -1) ret = w;
            if (LCT[w].R != -1) {
                LCT[LCT[w].R].PP = w;
                LCT[LCT[w].R].P = -1;
            }
            LCT[v].PP = -1; /// ** missed ** Do we really need
                this?
            setRightChild(w, v);
//            LCT[w].R = v;
//            LCT[v].P = w;
            splay(v);
        }
        return ret;
    }

    int find_root(int v) {
        access(v);
        int ret = v;
        while (LCT[ret].L != -1) {
            ret = LCT[ret].L;
            pushLazy(ret);
        }
        access(ret);
        return ret;
    }
```

```
    /// make w, parent of v where v is a root
    void link(int v, int w) {// attach v's root to w
        access(w);
        // the root can only have right children in
        // its splay tree, so no need to check
        setLeftChild(v, w);
//        LCT[v].L = w;
//        LCT[w].P = v;
        LCT[w].PP = -1;
    }

    void cut(int v) {
        access(v);
        if (LCT[v].L != -1) {
            LCT[LCT[v].L].P = -1;
            LCT[LCT[v].L].PP = -1;
            setLeftChild(v, -1);
//            LCT[v].L = -1;
        }
    }

    void make_root(int v) {
        access(v);
        int l = LCT[v].L;
        if (l != -1) {
            setLeftChild(v, -1);
            LCT[l].P = -1;
            LCT[l].PP = v;
            LCT[l].lazyFlip ^= 1;
        }
    }

    bool isConnected(int p, int q) {
        return find_root(p)==find_root(q);
    }

    /// assuming p and q is in the same tree
    int LCA(int p, int q) {
        access(p);
        return access(q);
    }
};


int main()
{
    // This is the code I used for the problem Dynamic LCA
        (DYNALCA)
    // on Sphere Online Judge (SPOJ)

    ios::sync_with_stdio(false);
    cin.tie(0);

    int n, m;
    cin >> n >> m;

    LinkCutTree lct;
    for (int i = 1; i <= n; i++) lct.make_tree(i);

    while (m--) {
        string cmd;
        cin >> cmd;
        if (cmd == "link") {
            int p, q;
            cin >> p >> q;
            lct.link(p, q);
        } else if (cmd == "cut") {
            int p;
            cin >> p;
```

```
            lct.cut(p);
        } else if (cmd == "lca") {
            int p, q;
            cin >> p >> q;
            cout << lct.LCA(p, q) << "\n";
        }
    }

    return 0;
}
```

7.17 link cut tree

```
/// not tested yet
#include <bits/stdc++.h>
using namespace std;

const int MAXN = 4e5 + 10;

template<typename T>
ostream& operator<<(ostream& os, vector<T> v) {
  os << "[";
  for (T &i: v) os << i << ",";
  os << "]";
  return os;
}

template<class TH> void _dbg(const char *sdbg, TH h){
    cout<<sdbg<<'='<<h<<endl; }
template<class TH, class... TA> void _dbg(const char *sdbg, TH
    h, TA... a) {
  while(*sdbg!=',')cout<<*sdbg++;
  cout<<'='<<h<<','; _dbg(sdbg+1, a...);
}

#ifdef __DJKIM613
#define debug(...) _dbg(#__VA_ARGS__, __VA_ARGS__)
#else
#define debug(...) (__VA_ARGS__)
#endif

struct query{
    int node, time, type;
    query(){}
    query(int node, int time, int type) : node(node),
        time(time), type(type) {}
};

long long sqrt(long long x){
    return x * x;
}

struct LCT{
    int N;
    vector<int> P;
    vector<vector<int>> CH;
    vector<long long> val1, val2;
    vector<long long> rval1, rval2;

    LCT(){}
    LCT(int N) : N(N){
        P.resize(N);  CH.resize(N);
        for(int i = 0 ; i < N ; i++) CH[i].resize(2);
        val1.resize(N, 1); val2.resize(N, 0); val1[0] = 0;
        rval1.resize(N, 0);  rval2.resize(N, 0);
    }

    void push_up(int x){
        val1[x] = val1[CH[x][0]] + val1[CH[x][1]] + rval1[x] + 1;
```

```
    val2[x] = sqrt(val1[CH[x][0]]) + sqrt(val1[CH[x][1]]) +
        rval2[x];
}

bool is_root(int x){
    return (CH[P[x]][0] != x && CH[P[x]][1] != x);
}

void rotate(int x){
    int y = P[x], z = P[y], w = (CH[y][1] == x);
    if(!is_root(y)) CH[z][CH[z][1] == y] = x;
    CH[y][w] = CH[x][w ^ 1]; P[CH[x][w ^ 1]] = y;
    CH[x][w ^ 1] = y; P[y] = x;  P[x] = z;
    push_up(y); push_up(x);
}

void splay(int x){
    while(!is_root(x)){
        int y = P[x], z = P[y];
        if(!is_root(y)) ((CH[z][0] == y) == (CH[y][0] == x))
            ? rotate(y) : rotate(x);
        rotate(x);
    }
}

void access(int x){
    for(int t = 0 ; x ; t = x, x = P[x]){
        splay(x);
        rval1[x] += val1[CH[x][1]]; rval2[x] +=
            sqrt(val1[CH[x][1]]);
        CH[x][1] = t;
        rval1[x] -= val1[CH[x][1]]; rval2[x] -=
            sqrt(val1[CH[x][1]]);
        push_up(x);
    }
    }

    void link(int x, int p){
        access(p); splay(p);  splay(x);
        CH[p][1] = x;  P[x] = p;
        push_up(p);
    }

    void cut(int x, int p){
        access(p); splay(p);  splay(x);
        rval1[p] -= val1[x]; rval2[p] -= sqrt(val1[x]); P[x] = 0;
        push_up(p);
    }

    int find_root(int x){
        access(x); splay(x);
        while(CH[x][0]) x = CH[x][0];
        return x;
    }
};

int N, M;
int C[MAXN];
vector<query> Q[MAXN];
vector<int> ADJ[MAXN];
int P[MAXN];
long long ans[MAXN];
void dfs(int here){
    for(int there : ADJ[here]) if(there != P[here]) P[there] =
        here, dfs(there);
}

int main(void){
    ios_base::sync_with_stdio(false);
    cin.tie(nullptr);  cout.tie(nullptr);
```

```
cin >> N >> M;
for(int i = 1 ; i <= N ; i++) cin >> C[i];

for(int i = 1 ; i < N ; i++) {
    int x, y;
    cin >> x >> y;
    ADJ[x].push_back(y);   ADJ[y].push_back(x);
}

dfs(1); P[1] = N + 1;
LCT lct = LCT(N + 2);
for(int i = 1 ; i <= N ; i++) lct.link(i, P[i]);
for(int i = 1 ; i <= N ; i++) Q[C[i]].push_back(query(i, 0,
    0));
for(int i = 1 ; i <= M ; i++){
    int x, c;
    cin >> x >> c;
    Q[C[x]].push_back(query(x, i, 1));
    Q[C[x] = c].push_back(query(x, i, 0));
}

for(int i = 1 ; i <= N ; i++) Q[C[i]].push_back(query(i, M +
    1, 1));

for(int i = 1 ; i <= N ; i++) {
    for(auto q : Q[i]){
        int node = q.node, t = q.time, ff =
            lct.find_root(P[node]);
        if(q.type){
            lct.splay(ff); lct.splay(node);
            ans[t] += lct.val2[ff] + lct.val2[node];
            //debug("[+]", ff, t, lct.val2[ff] +
                lct.val2[node]);
            lct.link(node, P[node]); lct.splay(ff);
            ans[t] -= lct.val2[ff];
            //debug("[++]", ff, lct.val2[ff]);

        }
        else{
            lct.splay(ff); ans[t] += lct.val2[ff];
            //debug("[-]", ff, t, lct.val2[ff]);
            lct.cut(node, P[node]);
            lct.splay(ff); lct.splay(node); ans[t] -=
                lct.val2[ff] + lct.val2[node];
            //debug("[--]", lct.val2[ff] + lct.val2[node]);
        }
    }
    //cout << '\n';
}

for(int i = 1 ; i <= M ; i++) ans[i] += ans[i - 1];
for(int i = 0 ; i <= M ; i++) cout << ans[i] << '\n';
return 0;
}
```

7.18 segment tree Max with index

```
#include <bits/stdc++.h>
using namespace std;

typedef long long ll;

const int N = 2e5 + 100, inf = 1e9 + 100;
int tree[4*N];
int a[N];

/// max in a range - keeps index
/// initiate a[0] with -inf
```

```
void build (int cn,int b,int e) {
    if (b == e) {
        tree[cn] = b;
        return ;
    }
    int lc = 2*cn , rc = lc+1 , mid = (b+e)/2;
    build(lc,b,mid);
    build(rc,mid+1,e) ;
    if(a[tree[lc]] > a[tree[rc]]) tree[cn] = tree[lc];
    else tree[cn] = tree[rc];
}

int query (int cn , int b , int e , int i ,int j) {
    if (b > j or e < i) return 0;
    if (b >= i and e <= j) {
        return tree[cn];
    }
    int lc = 2*cn , rc = lc + 1 , mid = (b+e)/2;
    int idx1 = query(lc, b, mid, i, j), idx2 = query(rc, mid+1,
        e, i, j);
    if(a[idx1] > a[idx2]) return idx1;
    return idx2;
}

int main() {
    ios::sync_with_stdio(0);
    int n;
    cin >> n;
    for(int i = 1; i <= n; i++) {
        cin >> a[i];
    }
    a[0] = -inf;
    build(1,1,n);
}
```

7.19 segtree pt upd range max

```
#include <bits/stdc++.h>
using namespace std;

const int N = 1e5 + 5, inf = 2e9;
int tree[4*N], a[N]; /// be careful with overflow

void build (int cn,int b,int e) {
    if (b == e) {
        tree[cn] = a[b];
        return ;
    }
    int lc = 2*cn, rc = lc+1, mid = (b+e)/2;
    build(lc,b,mid);
    build(rc,mid+1,e);
    tree[cn] = max(tree[lc],tree[rc]);
}

void upd (int cn , int b , int e , int i, int add) {
    if (b > i or e < i) {
        return;
    }
    int lc = 2*cn , rc = lc + 1 , mid = (b+e)/2;
    if (b >= i and e <= i) {
        tree[cn] += add;
        return;
    }
    upd(lc,b,mid,i,add);
    upd(rc,mid+1,e,i,add);
    tree[cn] = max(tree[lc],tree[rc]);
}

int query (int cn , int b , int e , int i ,int j) {
    if (b > j or e < i) return -inf;
```

```
    if (b >= i and e <= j) {
        return tree[cn];
    }
    int lc = 2*cn, rc = lc + 1, mid = (b+e)/2;
    return max(query(lc,b,mid,i,j), query(rc,mid+1,e,i,j));
}


int main () {

    return 0 ;
}
```

7.20 segtree pt upd range min

```
#include <bits/stdc++.h>
using namespace std;

const int N = 1e6 + 100;

int tr[4*N], a[N]; /// be careful with overflow
int inf = 2e9;

void build (int cn , int b , int e) {
    if (b == e) {
        tr[cn] = a[b];
        return;
    }
    int lc = 2*cn, rc = lc+1, mid = (b+e)/2;
    build(lc,b,mid);
    build(rc,mid+1,e);
    tr[cn] = min(tr[lc],tr[rc]);
}

void upd (int cn , int b , int e , int i, int x) {
    if (b > i or e < i) {
        return;
    }
    int lc = 2*cn, rc = lc + 1, mid = (b+e)/2;
    if (b >= i and e <= i) {
        tr[cn] = x;
        return;
    }
    upd(lc,b,mid,i,x);
    upd(rc,mid+1,e,i,x);
    tr[cn] = min(tr[lc],tr[rc]);
}


int query (int cn , int b , int e , int i ,int j) {
    if (b > j or e < i) return inf;
    if (b >= i and e <= j) {
        return tr[cn];
    }
    int lc = 2*cn, rc = lc + 1, mid = (b+e)/2;
    return min(query(lc,b,mid,i,j),query(rc,mid+1,e,i,j));
}

int main() {
    return 0;
}
```

7.21 segtree pt upd range sum

```
#include <bits/stdc++.h>
using namespace std;
```

```
const int N = 1e5 + 5;
int tree[4*N] , a[N]; /// be careful with overflow

void build (int cn,int b,int e) {
    if (b == e) {
        tree[cn] = a[b];
        return;
    }
    int lc = 2*cn, rc = lc+1, mid = (b+e)/2;
    build(lc,b,mid);
    build(rc,mid+1,e);
    tree[cn] = tree[lc] + tree[rc];
}

void upd (int cn , int b , int e , int i , int add) {
    if (b > i or e < i) {
        return;
    }
    int lc = 2*cn , rc = lc + 1 , mid = (b+e)/2 ;
    if (b >= i and e <= i) {
        tree[cn] += add;
        return ;
    }
    upd(lc,b,mid,i,add);
    upd(rc,mid+1,e,i,add);
    tree[cn] = tree[lc] + tree[rc];
}


int query (int cn , int b , int e , int i ,int j) {
    if (b > j or e < i) return 0;
    if (b >= i and e <= j) {
        return tree[cn];
    }
    int lc = 2*cn , rc = lc + 1 , mid = (b+e)/2;
    return query(lc,b,mid,i,j) + query(rc,mid+1,e,i,j);
}


int main () {

    return 0 ;
}
```

7.22 segtree range add range max

```
#include <bits/stdc++.h>
using namespace std;

typedef long long ll;

const int N = 1e6 + 100;


/**
    1) Add X in a range
    2) query for max in range

**/

long long inf = 1e18;
long long tr[4*N], lz[4*N];

void propagate(int u, int st, int en) {
    if (!lz[u]) return;
    tr[u] += lz[u];
    if (st!=en) {
        lz[2*u] += lz[u];
        lz[2*u+1] += lz[u];
```

```
    }
    lz[u] = 0;
}

void update(int u, int st, int en, int l, int r, long long x) {
    propagate(u, st, en);
    if (r<st || en<l) return;
    else if (l<=st && en<=r) {
        lz[u] += x;
        propagate(u, st, en);
    }
    else {
        int mid = (st+en)/2;
        update(2*u, st, mid, l, r, x);
        update(2*u+1, mid+1, en, l, r, x);
        tr[u] = max(tr[2*u], tr[2*u+1]);
    }
}

long long query(int u, int st, int en, int l, int r) {
    propagate(u, st, en);
    if (r<st || en<l) return -inf;
    else if (l<=st && en<=r) return tr[u];
    else {
        int mid = (st+en)/2;
        return max(query(2*u, st, mid, l, r), query(2*u+1, mid+1,
            en, l, r));
    }
}

int main() {

}
```

7.23 segtree range add range min

```
#include <bits/stdc++.h>
using namespace std;

typedef long long ll;

const int N = 1e6 + 100;


/**
    1) Add X in a range
    2) query for min in a range

**/

long long inf = 1e18;
long long tr[4*N], lz[4*N];

void propagate(int u, int st, int en) {
    if (!lz[u]) return;
    tr[u] += lz[u];
    if (st!=en) {
        lz[2*u] += lz[u];
        lz[2*u+1] += lz[u];
    }
    lz[u] = 0;
}

void update(int u, int st, int en, int l, int r, long long x) {
    propagate(u, st, en);
    if (r<st || en<l) return;
    else if (l<=st && en<=r) {
        lz[u] += x;
        propagate(u, st, en);
```

```
    }
    else {
        int mid = (st+en)/2;
        update(2*u, st, mid, l, r, x);
        update(2*u+1, mid+1, en, l, r, x);
        tr[u] = min(tr[2*u], tr[2*u+1]);
    }
}

long long query(int u, int st, int en, int l, int r) {
    propagate(u, st, en);
    if (r<st || en<l) return inf;
    else if (l<=st && en<=r) return tr[u];
    else {
        int mid = (st+en)/2;
        return min(query(2*u, st, mid, l, r), query(2*u+1, mid+1,
            en, l, r));
    }
}

int main() {

}
```

7.24 segtree range add range sum

```
#include <bits/stdc++.h>
using namespace std;

/*
    range update : Add X to l to r
    range query : sum l to r
*/

const int N = 1e5 + 5;

int tree[4*N], lazy[4*N], a[N];

void init() {
    memset(tree,0,sizeof tree);
    memset(lazy,0,sizeof lazy);
}

void build (int cn,int b,int e) {
    if (b == e) {
        tree[cn] = a[b];
        return;
    }
    int lc = 2*cn, rc = lc+1, mid = (b+e)/2;
    build(lc,b,mid);
    build(rc,mid+1,e);
    tree[cn] = tree[lc] + tree[rc];
}

void relax (int cn, int b, int e) {
    if (lazy[cn]) {
        tree[cn] += (e-b+1)*lazy[cn] ;
        if (b != e) {
            lazy[2*cn] += lazy[cn] ;
            lazy[2*cn + 1] += lazy[cn] ;
        }
        lazy[cn] = 0;
    }
}

void upd (int cn, int b, int e, int i, int j, int add) {
    relax(cn,b,e);
    if (b > j or e < i) {
        return;
```

```
    }
    int lc = 2*cn , rc = lc + 1 , mid = (b+e)/2;
    if (b >= i and e <= j) {
        lazy[cn] += add;
        relax(cn,b,e);
        return;
    }
    upd(lc,b,mid,i,j,add);
    upd(rc,mid+1,e,i,j,add);
    tree[cn] = tree[lc] + tree[rc];
}

int query (int cn, int b, int e, int i, int j) {
    relax(cn,b,e);
    if (b > j or e < i) return 0;
    if (b >= i and e <= j) {
        return tree[cn];
    }
    int lc = 2*cn, rc = lc + 1, mid = (b+e)/2;
    return query(lc,b,mid,i,j) + query(rc,mid+1,e,i,j);
}

int main () {

    return 0;
}
```

8 dp

8.1 cht offline linear

```
/**
Linear Convex Hull Trick
Requirement:
    Minimum:
        M increasing, x decreasing, useless(s-1, s-2, s-3)
        M decreasing, x increasing, useless(s-3, s-2, s-1)
    Maximum:
        M increasing, x increasing, useless(s-3, s-2, s-1)
        M decreasing, x decreasing, useless(s-1, s-2, s-3)
**/


#include<bits/stdc++.h>
using namespace std;
typedef long long LL;

struct CHT {
    vector<LL> M;
    vector<LL> C;
    int ptr = 0;

    ///Use double comp if M,C is LL range
    bool useless(int l1, int l2, int l3) {
        return (C[l3]-C[l1])*(M[l1]-M[l2]) <=
            (C[l2]-C[l1])*(M[l1]-M[l3]);
    }

    LL f(int id, LL x) {
        return M[id]*x+C[id];
    }

    void add(LL m, LL c) {
        M.push_back(m);
        C.push_back(c);
        int s = M.size();

        while (s >= 3 && useless(s-3, s-2, s-1)) {
```

```
            M.erase(M.end()-2);
            C.erase(C.end()-2);
            s--;
        }
    }

    LL query(LL x) {
        if (ptr >= M.size()) ptr = M.size()-1;
        while (ptr < M.size()-1 && f(ptr, x) > f(ptr+1, x))
            ptr++; /// change > to < for maximum
        return f(ptr, x);
    }
};


///Solves SPOJ Acquire
const int N = 1e5+7;
LL dp[N];

typedef pair<LL, LL> PLL;
int main() {
    ios::sync_with_stdio(false);
    cin.tie(0);

    int n;
    cin>>n;

    vector<PLL> v(n), t;

    for (int i=0; i<n; i++) cin>>v[i].first>>v[i].second;
    sort(v.begin(), v.end());

    for (int i=0; i<n; i++) {
        while (t.size() && t.back().second <= v[i].second)
            t.pop_back();
        t.push_back(v[i]);
    }

    n = t.size();
    dp[0] = 0;

    CHT cht;
    for (int i=1; i<=n; i++) {
        cht.add(t[i-1].second, dp[i-1]);
        dp[i] = cht.query(t[i-1].first);
    }

    cout<<dp[n]<<endl;
}
```

8.2 connected component dp

```
/// Problem: Given an array A, find all permutations of A such
    that,
/// |A1-A2| + |A2-A3| + ... <= L , n <= 100, L <= 1000, Ai <=
    1000
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
const int INF = 2147483647;

int arr[1010];
int n, l;

int mem[110][1010][2][110][2];
int mod = 1000000007;

ll dp(int at, int curl, int kl, int k, int kr) {
```

```
// kl = 1 if there is a segment connected to the left
    border, 0 otherwise
// kr = 1 if there is a segment connected to the right
    border, 0 otherwise
// k is the number of segments in the middle

int nxtl = curl;
if (at > 0) {
    // add the penalty from the last element:
    nxtl += (kl+kr+2*k)*abs(arr[at]-arr[at-1]);
}

if (nxtl > l) return 0;
if (k < 0) return 0;

if (at == n-1) {
    return k == 0 ? 1 : 0;
}
if (mem[at][curl][kl][k][kr] != -1)
    return mem[at][curl][kl][k][kr];

ll res = 0;

res += dp(at+1, nxtl, 1, k, kr); // connect to left segment
res += dp(at+1, nxtl, 1, k-1, kr)*k; // connect to left
    segment, and join to some middle segment

res += dp(at+1, nxtl, kl, k, 1); // connect to right segment
res += dp(at+1, nxtl, kl, k-1, 1)*k; // connect to right
    segment, and join to some middle segment

res += dp(at+1, nxtl, kl, k+1, kr); // new segment
res += dp(at+1, nxtl, kl, k, kr)*k*2; // connect to some
    middle segment
    res += dp(at+1, nxtl, kl, k-1, kr)*k*(k-1); // join two
        middle segments

    return mem[at][curl][kl][k][kr] = res % mod;
}

int main() {
    memset(mem,-1,sizeof(mem));
    cin >> n >> l;
    for(int i = 0; i < n; i++) cin >> arr[i];
    sort(arr,arr+n);
    cout << dp(0, 0, 0, 0, 0) << endl;
    return 0;
}
```

8.3 convaxhulltrick

```
#define ll long long

//convex hull for maximizing
//in case of minimization, just insert(-m,-c) and negate the
    result for query

bool Q;
struct Line {
        mutable ll m, c, p;
        bool operator<(const Line& o) const {
                return Q ? p < o.p : m < o.m;
        }
};

struct LineContainer : multiset<Line> {
        // (for doubles, use inf = 1/.0, div(a,b) = a/b)
        const ll inf = LLONG_MAX;
```

```
    ll div(ll a, ll b) { // floored division
        return a / b - ((a ^ b) < 0 && a % b); }
    bool isect(iterator x, iterator y) {
        if (y == end()) { x->p = inf; return false; }
        if (x->m == y->m) x->p = x->c > y->c ? inf : -inf;
        else x->p = div(y->c - x->c, x->m - y->m);
        return x->p >= y->p;
    }
    void addLine(ll m, ll c) {
        auto z = insert({m, c, 0}), y = z++, x = y;
        while (isect(y, z)) z = erase(z);
        if (x != begin() && isect(--x, y)) isect(x, y =
            erase(y));
        while ((y = x) != begin() && (--x)->p >= y->p)
            isect(x, erase(y));
    }
    ll query(ll x) {
        assert(!empty());
        Q = 1; auto l = *lower_bound({0,0,x}); Q = 0;
        return l.m * x + l.c;
    }
    bool isEmpty(){ return (empty()) ; }
    void Clear()
    {
        clear() ;
    }
}ch;
```

8.4 digit dp template

```
#include <bits/stdc++.h>
using namespace std;

typedef long long ll;

int totNodes = 1;

struct state {
    int to[20] ;
    int depth ;
    int suffLink ;
    int par ;
    int parLet ;
    int cnt ;
    int nxt[20] ;
}states[205];

int add (string &str, int val) {

    int cur = 1 ; // root with a empty string
    int len = str.size();
    for (int i = 0 ; i < len ; i++) {
        char c = str[i] ;
        if (!states[cur].to[c-'a']) {
            states[cur].to[c-'a'] = ++totNodes ;
            states[totNodes].par = cur ;
            states[totNodes].depth = states[cur].depth + 1 ;
            states[totNodes].parLet = c-'a' ;
        }
        cur = states[cur].to[c-'a'] ;
    }
    states[cur].cnt += val;
    return cur ;
}

vector<int> g[205];

void dfs(int u) {
```

```
    for(int v : g[u]) {
        states[v].cnt += states[u].cnt;
        dfs(v);
    }
}

void pushLinks() {
    queue <int> Q ; Q.push(1) ;
    while (!Q.empty()) {
        int node = Q.front() ;
        Q.pop() ;
        if (states[node].depth <= 1) {
            states[node].suffLink = 1 ;
        }
        else {
            int cur = states[states[node].par].suffLink ;
            int parLet = states[node].parLet ;
            while (cur > 1 and !states[cur].to[parLet]) {
                cur = states[cur].suffLink ;
            }
            if (states[cur].to[parLet]) {
                cur = states[cur].to[parLet] ;
            }
            states[node].suffLink = cur;
        }
        for (int i = 0 ; i < 26 ; i++) {
            if (states[node].to[i]) {
                Q.push(states[node].to[i]) ;
            }
        }
    }
    for(int i = 2; i <= totNodes; i++)
        g[states[i].suffLink].push_back(i);
    dfs(1);
}
```

```
int Next (int from , char ch) {
    if (states[from].nxt[ch-'a']) return
        states[from].nxt[ch-'a'] ;
    int cur = from ;
    int c = ch - 'a' ;
    while (cur > 1 and !states[cur].to[c]) {
        cur = states[cur].suffLink ;
    }
    if (states[cur].to[c]) {
        cur = states[cur].to[c] ;
    }
    return states[from].nxt[ch-'a'] = cur ;
}

string input_string(bool flag) {
    int k;
    cin >> k;
    string ret;
    if(flag) ret.assign(200-k, 'a');
    for(int i = 0; i < k; i++) {
        int x;
        cin >> x;
        ret.push_back('a' + x);
    }
    return ret;
}

int n, base, limit;
int mod = 1e9 + 7;
int dp[201][201][501][2][2][2];
string R, L;
```

```
void add(int &x, int y) {
    x += y;
    if(x>=mod) x-=mod;
}

int call(int pos, int state, int sum, bool start, bool
    alreadyLarge, bool alreadySmall) {
    if(sum > limit) return 0;
    if(pos == 200) return 1;
    if(dp[pos][state][sum][start][alreadyLarge][alreadySmall] !=
        -1) return
        dp[pos][state][sum][start][alreadyLarge][alreadySmall];
    int ret = 0;
    for(int i = 0; i < base; i++) {
        if(alreadyLarge == false && i < L[pos]-'a') continue;
        if(alreadySmall == false && i > R[pos]-'a') continue;
        int npos = pos + 1;
        bool nstart = start|(i>0);
        bool nalreadyLarge = alreadyLarge|(i>L[pos]-'a');
        bool nalreadySmall = alreadySmall|(i<R[pos]-'a');
        int nstate = state;
        if(nstart) nstate = Next(nstate, 'a' + i);
        int nsum = sum + states[nstate].cnt;
        add(ret, call(npos, nstate, nsum, nstart, nalreadyLarge,
            nalreadySmall));
    }
    dp[pos][state][sum][start][alreadyLarge][alreadySmall] = ret;
    return ret;
}

int main() {
    ios::sync_with_stdio(0);
    cin.tie(0);
    memset(states, 0, sizeof states);
    cin >> n >> base >> limit;
    L = input_string(1);
    R = input_string(1);
    memset(dp, -1, sizeof dp);
    for(int i = 1; i <= n; i++) {
        string s = input_string(0);
        int x;
        cin >> x;
        int state = add(s, x);
    }
    pushLinks();
    cout << call(0,1,0,0,0,0) << endl;
    return 0;
}
```

8.5 dp divide and conquer

```
#include <bits/stdc++.h>
using namespace std;
#define int long long
const int N = 5005 ;
int n , K;
int a[N];
int dp[N][N] ,cost[N][N] ;

void solve(int i , int l , int r , int ql , int qr){
    if(r < l){
        return;
    }
    int mid = (l + r)/2 ;
    int idx = -1;
    int val = 0 ;
    for(int j = ql ; j <= min(qr , mid) ; ++j){
        int tmp = dp[i - 1][j - 1] + cost[j][mid];
```

```
        if(tmp >= val){
            val = tmp;
            idx = j;
        }
    }
    dp[i][mid] = val;
    solve(i , l , mid - 1 , ql , idx);
    solve(i , mid + 1 , r , idx , qr);
}

int main(){
    int tc ; cin >> tc ;
    while (tc--) {
        scanf ("%lld %lld" , &n , &K);
        for (int i = 1 ; i <= n ; i++) {
            scanf ("%lld" , &a[i]) ;
            dp[1][i] = dp[1][i-1]|a[i] ;
        }
        for (int i = 1 ; i <= n ; i++) {
            for (int j = i ; j <= n ; j++) {
                cost[i][j] = cost[i][j-1]|a[j] ;
            }
        }
        for (int k = 2; k <= K ; k++) {
            solve(k,1,n,1,n) ;
        }
        printf ("%lld\n" , dp[K][n]) ;
    }
}
```

8.6 knuth opt (kundu)

```
/**
You have to cut a wood stick into pieces. The most affordable
    company, The Analog Cutting Machinery,
Inc. (ACM), charges money according to the length of the stick
    being cut. Their procedure of work
requires that they only make one cut at a time.
It is easy to notice that different selections in the order of
    cutting can led to different prices. For
example, consider a stick of length 10 meters that has to be cut
    at 2, 4 and 7 meters from one end.
There are several choices. One can be cutting first at 2, then
    at 4, then at 7. This leads to a price of 10
+ 8 + 6 = 24 because the first stick was of 10 meters, the
    resulting of 8 and the last one of 6. Another
choice could be cutting at 4, then at 2, then at 7. This would
    lead to a price of 10 + 4 + 6 = 20, which
is a better price.
Your boss trusts your computer abilities to find out the minimum
    cost for cutting a given stick.
*//

#include <bits/stdc++.h>
#define LL long long
using namespace std;

const int N=1000+7;
LL a[N];
LL dp[N][N];
int opt[N][N];

const long long INF=1e15;


//solves ZOJ 2860

int main ()
```

```
{
    ios::sync_with_stdio(false);
    int l, n;

    while (cin>>l>>n)
    {
        a[0] = 0;
        a[++n] = l;

        for (int i=1; i<n; i++)
            cin>>a[i];

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

        for (int len = 2; len<=n; len++)
            for (int l=0; l+len<=n; l++)
            {
                int r=l+len;
                int optl = opt[l][r-1];
                int optr = opt[l+1][r];
                dp[l][r] = INF;

                for (int i=optl; i<=optr; i++)
                {
                    LL cost = dp[l][i] + dp[i][r] + (a[r] - a[l]);
                    if (cost < dp[l][r])
                        dp[l][r] = cost,
                        opt[l][r] = i;
                }
            }

        cout<<dp[0][n]<<endl;
    }
}
```