Team Notebook

BUET_BloodHound(Bangladesh University of Engineering and Technology)

March 26, 2019

Contents

1 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;
}
```

2 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;
}
///intersection area of triangle((0,0), A, B) and circle
     with center cen, radius r
ld Tri_cross_Cir(PT A,PT B,PT cen,ld r){
  ld a,b,c,x,y,s=((A-cen)^(B-cen))*0.5;
  a=len(B-cen);b=len(A-cen);c=len(A-B);
  if(a<=r&&b<=r)return s;
  else if(a<r&&b>=r) {
    x=((A-B)*(cen-B)+sqrt(c* c*r*r-sqr((A-B)^(cen-B))))/c;
    return asin(s*(c-x)*2.0/c/b/r)*r*r*0.5+s*x/c;
  }
  else if(a>=r&&b<r) {
    y=((B-A)*(cen-A)+sqrt(c*c*r*r-sqr((B-A)^(cen-A))))/c;
    return asin(s*(c-y)*2.0/c/a/r)*r*r*0.5+s*y/c;
  }
  else {
    if(fabs(2.0*s)>=r*c||(B-A)*(cen-A)<=0||(A-B)*(cen-B)<=0)
         {
      if((A-cen)*(B-cen)<0) {
        if(((A-cen)^(B-cen))<0)
          return (-pi-asin(s*2.0/a/b))*r*r*0.5;
        else return (pi-asin(s*2.0/a/b))*r*r*0.5;
      }
      else return asin(s*2/a/b)*r*r*0.5;
    }
    else {
      x=((A-B)*(cen-B)+sqrt(c*c*r*r-sqr((A-B)^(cen-B))))/c;
      y=((B-A)*(cen-A)+sqrt(c*c*r*r-sqr((B-A)^(cen-A))))/c;
```

```
    return (asin(s*(1-x/c)*2/r/b)+asin(s*(1-y/c)*2/r/a))*r*
        r*0.5+s*((y+x)/c-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);
}
/// 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+1)%n],p[q] )) >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])))/d;
    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);
}
```

3 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 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) );
}
```

4 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;
    }

};
```

5 AhoCorasick

```
class ahoCorasick{
public:
  int sigma , maxSize , curSize ; int **to , *link ;
  vector <int> *endedHere ;
/* 0 based trie, 0 is the root, curSize denote the number of
states now, 0 ,1,....,curSize-1 sigma = size of alphabet ,
link[i] points to the max suffix of string ended at i'th
     node
endedHere[i] = the strings ended in the i'th node of trie
to[i][j] = 0, if there is no such state, otherwise it
     denotes
the state to go from i, after adding j'th symbol g[i][....]
contains the node whose suffix link is to i, that means
link[ g[i][j] ] = i*/

ahoCorasick( int _sigma , int _maxSize ){
  sigma = _sigma ; maxSize = _maxSize ;
  to = new int*[maxSize+2] ;
  for(int i=0 ; i<maxSize+2 ; i++) {
      to[i] = new int[sigma] ;
      for(int j=0 ; j<sigma ; j++) to[i][j] = 0 ;
  }
  link = new int[maxSize+2] ; curSize = 1 ;
  endedHere = new vector<int>[maxSize+2] ;
}
void addString(string const& S ,int idx ){
  int cur = 0 ;
  for( auto ch : S ){
  if( to[cur][ ch-'a' ]==0 ) to[cur][ch-'a'] = curSize++;
      cur = to[cur][ ch-'a' ] ;
  }
  endedHere[cur].pb(idx) ;
}
void findSuffixLink(){
  int i ; queue < int > q ;
  for(i=0 ; i<sigma ; i++){
      if( to[0][i] != 0 ){ link[ to[0][i] ] = 0 ;
        q.push(to[0][i]) ; g[0].pb( to[0][i] ) ;
      }
  }
  while(!q.empty()){
    int state = q.front() ; q.pop() ;
    for(int ch = 0 ; ch<sigma ; ch++){
      if( to[state][ch]!=0 ){
        int failure = link[state] ;
        while( failure != 0 && to[failure][ch]==0 ){
          failure = link[failure] ;
        }
        link[ to[state][ch] ] = to[ failure ][ch] ;
        g[ to[ failure ][ch] ].pb(to[state][ch]);
        q.push( to[state][ch] ) ;
      }
    }
  }
}
int findNextState( int state , int input){
  while( state!=0 && to[ state ][input]==0 ){
    state = link[state] ;
  }
  return to[ state ][input] ;
}
void searchWord( string const& S ){
  int cur = 0 ;
  for(int i=0; i<S.size(); i++){
    char ch = S[i] ; cur = findNextState(cur,ch-'a') ;
// found the state where machine would come after S[0...i]
    for(int j=0 ; j<wasHere[cur].size() ; j++){
      pos[wasHere[cur][j]].pb( i ) ;
    }
  }
  return ;
}
~ahoCorasick(){
  for(int i=0 ; i<maxSize+2 ; i++) {
    delete to[i] ; endedHere[i].clear() ;
  }
  delete to ; delete endedHere ; delete link ;
}
};
```

6 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);
 }
};
```

7 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 ;
}
```

8 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;
  }
};
```

9 CentroidDecomposition

```
struct CentroidDecomposition{
  vector< vector<int> > g;
  vector <int> done, child, parent;
  int n;
  void init(int n_){
    n = n_; g.assign(n+1,{});done.assign(n+1,0);
    child.assign(n+1,0); parent.assign(n+1,0);
  }
  void addEdge(int u,int v){
    g[u].push_back(v); g[v].push_back(u);
  }
  void dfsSZ (int u, int par) {
      child[u] = 1;
      for (int v : g[u]) {
          if (done[v] or v == par) continue ;
          dfsSZ(v,u); child[u] += child[v] ;
      }
  }
  int dfsFC(int u, int par, int sz) {
    for (int v : g[u]) {
      if (!done[v] and v != par and child[v] > sz)
          return dfsFC(v,u,sz) ;
    }
    return u ;
  }
  void dfsCD(int u, int par = 0) {
      dfsSZ(u,0);
      int centroid = dfsFC(u,0,child[u]/2) ;
      parent[centroid] = par;
      /// solve(centroid); ///modify this accordingly
      done[centroid] = 1;
      for (int v : g[centroid]) {
          if (!done[v]) {
              dfsCD(v,centroid) ;
          }
      }
  }
}
```

10 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;
}
```

11 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(cos(p2),sin(p2))*r[i]);
            }
            p1 = p2 ;
            ARG[j].second ? cnt--:cnt++;
        }
    }
    else {
        sum += 2*pi*r[i]*r[i] ;
    }
  }
  return (sum+fabs(sum1))*0.5 ;
}
```

12 ConvexHullTrick

```
#define ll long long
bool Q;
struct Line {
 mutable ll k, m, p;
 bool operator<(const Line& o) const {
  return Q ? p < o.p : k < o.k;
 }
};
struct LineContainer : multiset<Line> {
 // (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->k == y->k) x->p = x->m > y->m ? inf : -inf;
  else x->p = div(y->m - x->m, x->k - y->k);
  return x->p >= y->p;
 }
 void addLine(ll k, ll m) {
  auto z = insert({k, m, 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.k * x + l.m;
 }
 bool isEmpty(){ return (empty()) ; }
 void Clear() { clear() ; }
}ch;
```

13 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;
}
```

14 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;
 }
}
```

15 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;
}
```

16 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) //(A[i].x + 0.5);
    }
};
```

17 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());
    }


  }
}
```

18 Gaussian Elimination

```
///n = eqn (0,..,n-1) , m = var(0...m-1) , ar[i][m] = RHS
///-1: no soln , 0: unique,> 0 : free var..array is modified
int gauss(int n,int m,double ar[5][5],vector<double>& res){
    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]);
            /// double eqn starts
            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);
                }
            }
            /// double eqn ends
            /// mod eqn begins
            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;
                    }
                }
            }
            /// mod eqn ends
            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 gauss(int n, int m, bitset <MAXCOL> ar[MAXROW], bitset <
    MAXCOL>& res){
    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;
}
```

19 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;
}
```

20 HLD

```
const int M = 34567;
vector<int>edge[M];
class LCA {
    void setLCA( int vertex, int root = 1 )
    //par is the parent array we get from traversing the tree
    //d is the distance array to all vertex from root
    //creates binary power parent of tree
    void create( int *par, int *d = 0 )
    //finds lca of vertex u and v
    int find_lca( int u, int v )
    //find distance between u and v
    int distance( int u, int v )
};
int level[M],
    sz[M], //size of subtree of 'u'
    parent[M], //parent of vertex 'u'
    chainNo[M], //In what chain is the vertex 'u' in
    chainLen[M], //Length of a 'chain'
    posChain[M], //position of 'u' in a chain-1 based
    chainHead[M], //Head of a 'chain'
    Index[M]; //position of vertex 'u' in mapped array

void dfs( int u, int p )
{
    sz[u] = 1;
    for( int i = 0; i < edge[u].size(); i++ ) {
        int v = edge[u][i];
        if( v != p ) {
            level[v] = level[u] + 1; parent[v] = u;
            dfs(v,u); sz[u] += sz[v];
        }
    }
}
int chain, cur;
void create_hld( int u ) {
    Index[u] = cur++;
    chainNo[u] = chain;
    chainLen[chain]++; //memset it for multiple test case
    posChain[u] = chainLen[chain];
    int maxSize = -1, maxV = -1;
    for( int i = 0; i < edge[u].size(); i++ ) {
        int v = edge[u][i];
        if( v == parent[u] ) continue;
        if( maxSize < sz[v] ) {
            maxV = v;
            maxSize = sz[v];
        }
    }
    if( maxV != -1 )
        create_hld( maxV );
    for( int i = 0; i < edge[u].size(); i++ ) {
        int v = edge[u][i];
        if( v == parent[u] || v == maxV ) continue;
        chain++;
        chainHead[chain] = v;
        create_hld(v);
    }
}

int data[M]; // value of mapped vertex
int tree[4*M];
void init_tree( int cn, int s, int e )
void update_tree( int cn, int s, int e, int x, int v )
int query_tree( int cn, int s, int e, int x, int y )

int n;
LCA lca;

void update( int x, int v ) {
    data[ Index[x] ] = v;
    update_tree(1, 1, n, Index[x], v);
}

int query_up( int x, int p ) {
    int ans = 0;
    while(1) {
        if( chainNo[x] == chainNo[p] ) {
            ans += query_tree( 1, 1, n, Index[p], Index[x] );
            break;
        }
        int l = Index[x] - posChain[x] + 1, r = Index[x];
        ans += query_tree( 1, 1, n, l, r );
        x = parent[chainHead[ chainNo[x] ]];
    }
    return ans;
}
int query( int x, int y ) {

    int p = lca.find_lca(x, y);
    int a = query_up( x, p ), b = query_up( y, p );
    return a+b-data[ Index[p] ];
}
int temp[M];
int main() {
        cin >> n;
        for( int i =1; i <= n; i++ ) //for multi test case
            edge[i].clear();
        memset( chainLen, 0, sizeof chainLen );
        for( int i = 1; i <= n; i++ )
            cin >> temp[i]; // value of vertex i
        for( int i = 1, a, b; i < n; i++ ) {
            cin >> a >> b; //input was 0 based vertex
            a++; b++;
            edge[a].pb(b); edge[b].pb(a);
        }
        level[1] = 0;
        dfs(1,-1);
        chain = 1;
```

```
        cur = 1;
        create_hld(1);
        lca.setLCA(n, 1);
        lca.create(parent, level);
        for( int i = 1; i <= n; i++ ) {
            data[ Index[i] ] = temp[i];
        }
        init_tree(1, 1, n);
    }
    return 0;
}
```

21 HalfPlaneIntersection

```
typedef pair<long double, long double> pi;
bool z(long double x){ return fabs(x) < eps; }
struct line{
 ld a, b, c;
 line(ld a,ld b,ld c):a(a), b(b), c(c) {}
 bool operator<(const line &l)const{
  bool flag1 = pi(a, b) > pi(0, 0);
  bool flag2 = pi(l.a, l.b) > pi(0, 0);
  if(flag1 != flag2) return flag1 > flag2;
  long double t = ccw(pi(0, 0), pi(a, b), pi(l.a, l.b));
  return z(t) ? c*hypot(l.a, l.b) < l.c * hypot(a, b):t>0;
 }
 pi slope(){ return pi(a, b); }
};
pi cross(line a, line b){
 long double det = a.a * b.b - b.a * a.b;
 return pi((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(pi(0, 0), a.slope(), b.slope()) <= 0) return false;
 pi crs = cross(a, b);
 return crs.first * c.a + crs.second * c.b >= c.c;
}
// ax + by <= c;
bool solve(vector<line> v, vector<pi> &solution){
 sort(v.begin(), v.end());
 deque<line> dq;
 for(auto &i : v){
  if(!dq.empty() &&
    z(ccw(pi(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<pi> tmp;
 for(int i=0; i<dq.size(); i++){
  line cur = dq[i], nxt = dq[(i+1)%dq.size()];
  if(ccw(pi(0, 0), cur.slope(), nxt.slope()) <= eps)
     return false;
  tmp.push_back(cross(cur, nxt));
  cout << tmp.back().first <<" "<< tmp.back().second<<endl;
 }
 solution = tmp;
 return true;
}
```

22 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()
```

23 Hungarian Algorithm

```
namespace wm {
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 = 1-MAXIMIZE or 0-MINIMIZE
/// match[i] contains the column to which row i is matched
int hungarian(int n, int m, int mat[MAX][MAX],int flag = 0){
  memset(U, 0, sizeof(U)); memset(V, 0, sizeof(V));
  memset(P, 0, sizeof(P)); memset(ar, 0, sizeof(ar));
  memset(way, 0, sizeof(way));
    int inf = 1e9;
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= m; j++) {
            ar[i][j] = mat[i][j];
            if (flag) 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]=0;
      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) ? -V[0] : V[0];
}
}
```

24 Mincost Maxflow

```
namespace mcmf{
const int MAX = 31000; const long long INF = 1LL << 60;
long long cap[MAX], flow[MAX], cost[MAX], dis[MAX];
int n, m, s, t, Q[MAX*10], 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);
}
}
```

25 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)]
///
```

26 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
}
```

27 PollardRho

```
//expo(a,b,m) returns (a^b)%m
//prime contains all primes upto 47000, Pt = number of prime
intl prime[10000], Pt = 0;
const intl LIM = LLONG_MAX, mod = 1000000007;
intl mul( intl a, intl b, intl mod) {
    intl x, res;
    if (a < b) swap(a, b);
    if (!b) return 0;
    if (a < (LIM / b)) return ((a * b) % mod);
    res = 0, x = (a % mod), b %= mod;
    while (b) {
        if (b & 1) {
            res = res + x;
            if (res >= mod) res -= mod;
        }
        b >>= 1; x <<= 1;
        if (x >= mod) x -= mod;
    }
    return res%mod;
}
int isPrime( intl n ) {
    if( n == 2 ) return 1;
    if( n % 2 == 0 ) return 0;
    intl 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++ ) {
        intl x = test[i]%(n-2), temp = d;
        if(x < 2) x += 2;
        intl 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;
}
intl pollard_rho(intl n, intl c) {
    intl 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(intl n, vector<intl> &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 ;
    }
    intl d = n;
    for (int i = 2; d == n; i++) d = pollard_rho(n, i);
    llfactorize(d, f);
    llfactorize(n/d, f);
}
void factorize( intl n, vector< pair<intl,intl> > &ans ) {
    vector<intl> v;
    llfactorize(n, v);
    sort( v.begin(), v.end() );
    intl a = v[0], b = 1;
    for( int i = 1; i < (int)v.size(); i++ ) {
        if( v[i] == v[i-1] ) b++;
        else {
            ans.pb( make_pair(a,b) );
            a = v[i]; b = 1;
        }
    }
    ans.pb( make_pair(a,b) );
}
int main(){
    vector< pair<intl, intl> >ans;
    factorize( n, ans );
    return 0;
}
```

28 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]*(...) */
```

29 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);
}
```

30 RMQ(2D)

```
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) {
    int lx=x2-x1+1,ly=y2-y1+1,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]}) ;
}
```

31 SCC+2SAT

```
/*
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 ;
    }
};
```

32 Simplex

```
/*** O(n^2*m) , n = no of var , m = no of inequalities
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.
1.if exists equality constraint,then replace by both >=and<=
2.if var x doesn't have nonnegativity constraint,then
replace by difference of 2 var like x1-x2,where x1>=0, x2>=0
3. for a>=b constraints, convert to -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 );
        }
    }
};
```

33 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;
}
```

34 Suffix Automata

```
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 ;
  }
};
```

35 SuffixArray

```
/// sa = 1-based suffix array,height = 1-based
/// height[i] = lcp between i'th and (i-1)'th suffix
struct SuffixArray {
    int sa[N],data[N],rnk[N],height[N],n;
    int wa[N],wb[N],wws[N],wv[N];
    void init() {
        for (int i=1;i<N;i++) sa[i]=data[i]=rnk[i]=height[i]=
             wa[i]=wb[i]=wws[i]=wv[i]=0 ;
    }
    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 (char *A) {
        n = strlen(A) ;init();
        for (int i = 0 ; i < n ; i++) {
            data[i] = A[i] ;
        }
        DA(data,sa,n+1,128); calheight(data,sa,n);
    }
};
```

36 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 );
}
```

37 dominator

```
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 ;
    }
}
```

38 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;
}
```

39 lca

```
#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)] ;
    }
  }
}
```

40 persistentSegmentTree

```
/*in this persistent segment tree every time we can add to
value of some node and we can answer query value of index l
to index r add in the interval i to j */
class persistentSegTree{
public:
  i64 *Tree ; int *left , *right , *root ; int cnt , n ;
  persistentSegTree(int _n) {
    n = _n ; Tree = new i64[21*(n+2)] ;
    left = new int[21*(n+2)] ; right = new int[21*(n+2)] ;
    root = new int[n+2] ; cnt = 1 ; root[0] = build(1,n) ;
  }
  int build(int b ,int e){
    int cur = cnt++ ;
    if(b==e){
      Tree[cur] = 0 ;
      return cur ;
    }
    int m = (b+e)/2 ;
    left[cur] = build(b,m) ; right[cur] = build(m+1,e) ;
    Tree[cur] = Tree[left[cur]] + Tree[right[cur]] ;
    return cur ;
  }
  int update(int contemporary,int b, int e,int idx,i64 val){
    if( idx < b || idx > e ) return contemporary ;
    int cur = cnt++ ;
    if(b==e){
      Tree[cur] = Tree[contemporary] + val ;
      return cur ;
    }
    int m = (b+e)/2 ;
    left[ cur ] = update(left[contemporary],b,m,idx,val) ;
    right[ cur ] = update(right[contemporary],m+1,e,idx,val);
    Tree[cur] = Tree[left[cur]] + Tree[right[cur]] ;
    return cur ;
  }

  i64 query(int i, int j ,int b, int e , int l, int r) {
    if(l>r) return 0 ;
    if( b>r || e<l ) return 0 ;
    if( l<=b && e<=r ) return Tree[j]-Tree[i] ;
    int m = (b+e)/2 ;
    i64 r1 = query(left[i],left[j],b,m,l,r) ;
    i64 r2 = query(right[i],right[j],m+1,e,l,r) ;
    return r1+r2 ;
  }

  i64 Query(int i, int j, int l ,int r) {
    if( j<0 || i>n || i>j || l>r ) return 0LL ;
    return query( root[i-1] , root[j] , 1 , n , l , r ) ;
  }

  ~persistentSegTree() {
    delete root; delete left; delete right; delete Tree;
  }
};
```

41 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 ;
}
```