Template

Y

November 2, 2019

目录

1 Basic

1.1 .vimrc

```
set nu ai ci si mouse=a ts=2 sts=2 sw=2
nmap<F2> : vs %<.in <CR>
nmap<F3> : !gedit % <CR>
nmap<F8> : !time ./%< < %<.in <CR>
nmap<F9> : :w <CR> :!g++ % -o %< -O2 -g -std=c++11 -Wall <CR>
nmap<F10> : :w <CR> :make %< <CR>
```

1.2 head

```
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;
typedef double db;
typedef pair<int,int> pii;
typedef vector<int> vi;
#define dd(x) cout << #x << "=" << x << ","
#define de(x) cout << #x << "=" << x << endl
#define rep(i,a,b) for(int i=(a);i<(b);++i)
#define per(i,a,b) for(int i=(b-1);i>=a;--i)
#define all(x) (x).begin(),(x).end()
#define sz(x) (int)(x).size()
#define mp make_pair
#define pb push_back
#define fi first
#define se second
#define endl "\n"
#define rk(x) upper_bound(all(V) , x) - V.begin()
#define lowbit(x) x&(-x)
#define inf 0x3f3f3f3f
const int N = 101010;
const int M = 1e9+7;

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


   return 0;
}
```

1.3 stl

```
#include<bits/stdc++.h>
using namespace std;
int main(){
   int num[6]={1,2,4,7,15,34},x=7;
   sort(num,num+6);// 从小到大排序
   lower_bound(num,num+6,x);// 第一个大等于的指针x
   upper_bound(num,num+6,x);// 第一个大于的指针x
   sort(num,num+6,greater<int>());// 从大到小排序
   lower_bound(num,num+6,x,greater<int>());// 第一个小等于的指针x
   upper_bound(num,num+6,x,greater<int>());// 第一个小于的指针x
   return 0;
}
```

2 DataStructure

2.1 LCARMQ

```
// N is 2 size of tree , id of nodes start from 1
struct LCARMQ{
   static const int N = 101010 << 1;
   int a[20][N] , lft[N] , dep[N] , lg[N] , L;
   int rmin(int x,int y){return dep[x] < dep[y] ? x : y;}
   void add(int x){ a[0][L++] = x;}
   void dfs(int c,int fa,const vi g[]){
      lft[c]=L;add(c);
      for(auto t : g[c]) if(t!=fa) dep[t]=dep[c]+1,dfs(t,c,g),add(c);
   }
   void Build(const vi g[]){
      L = 0;dfs(1,0,g);dep[0] = -1;
      rep(i,2,L) lg[i]=lg[i>>1]+1;
      rep(i,1,20){
         int lim = L+1-(1<<i);
         rep(j,0,lim) a[i][j] = rmin(a[i-1][j] , a[i-1][j+(1<<i>>1)]);
      }
   }
   int lca(int x,int y){
      x = lft[x] , y = lft[y];
      if(x > y) swap(x , y);
      int i = lg[y-x+1];
      return rmin(a[i][x] , a[i][y+1-(1<<i)]);
   }
};
```

2.2 ST

```
// [0,n)
struct ST{
   static const int N = 101010;
   int a[20][N], lg[N];
   void build(int *v, int n){
      rep(i, 2, n + 1) lg[i] = lg[i >> 1] + 1;
      rep(i, 0, n) a[0][i] = v[i];
      rep(i, 1, lg[n] + 1) rep(j, 0, n - (1 << i) + 1) {
         a[i][j] = max(a[i - 1][j], a[i - 1][j + (1 << i >> 1)]);
      }
   }
   int qry(int l, int r){
      if(l > r) swap(l, r);
      int i = lg[r - l + 1];
      return max(a[i][l] , a[i][r + 1 - (1 << i)]);
```

```
  }
};
```

3 Geo

3.1 基础点、向量

```
struct P {
   int quad() const { return sign(y) > 0 || (sign(y) == 0 && sign(x) >= 0); }
   P rot90() { return P(-y, x); }
   P rot(db a) { return P(cos(a) * x - sin(a) * y, cos(a) * y + sin(a) * x); }
   P norm() { return *this / len(); }
};
db rad(P p1, P p2) { return atan2l(det(p1, p2), dot(p1, p2)); } // p1 与 p2 的夹角，有方向
bool cmp(const pii &a, const pii &b) { // 级角排序
   int o = a > pii(0, 0), t = b > pii(0, 0);
   if(o != t) return o < t;
   return det(a, b) > 0;
}
// 【点集中最近点对】
namespace NearestPoints { // sz(A) <= 1e5
   db solve(int l, int r, vector<P> &p) {
      if(l == r) return 1e100;
      int m = l + r >> 1;
      db Xm = p[m].x, lim = min(solve(l, m, p), solve(m + 1, r, p));
      inplace_merge(p.begin() + l, p.begin() + m + 1, p.begin() + r + 1, [&](P a, P b){
      return a.y < b.y;});
      vector<P> V;
      rep(i, l, r + 1) if(fabs(p[i].x - Xm) <= lim) V.pb(p[i]);
      rep(i, 0, sz(V)) rep(j, i + 1, sz(V)) {
         if(fabs(V[j].y - V[i].y) >= lim) break;
         T dis = (V[i] - V[j]).len();
         lim = min(lim, dis);
      }
      return lim;
   }
   db solve(vector<P> A) {
      sort(all(A), [&](P a, P b){return a.x < b.x;});
      return solve(0, sz(A) - 1, A);
   }
}
// 【最小圆覆盖】
C Mincir(P *p,int n){
   random_shuffle(p , p + n);
   P o = p[0];db r = 0;
   rep(i,1,n) {
      if(sgn(abs(o-p[i])-r) <= 0) continue;
      o = p[i] , r = 0;
      rep(j,0,i) {
         if(sgn(abs(o-p[j])-r) <= 0) continue;
         o = (p[i] + p[j]) / 2 , r = abs(o-p[j]);
         rep(k,0,j) {
            if(sgn(abs(o-p[k])-r) <= 0) continue;
            o = outC(p[i],p[j],p[k]) , r = abs(o-p[k]);
      }}}
   return C(o,r);
}
// 【费马点】
// sqrt((a ^ 2 + b ^ 2 + c ^ 2 + 4 * sqrt(3) * area) / 2)
// 如果有重点，大于 2 的直接用模拟退火法
P fermat(vector<P> p) {
  int n = sz(p); assert(n);
  if(n == 1) return p[0];
  if(n == 2) return (p[0] + p[1]) / 2;
  if(n == 3) {
    db a[3];
    rep(i, 0, 3) a[i] = (p[(i + 2) % 3] - p[(i + 1) % 3]).len();
    rep(i, 0, 3) {
      int j = (i + 1) % 3, k = (i + 2) % 3;
      if(sign(a[i] * a[i] - a[j] * a[j] - a[k] * a[k] - a[j] * a[k]) >= 0) return p[i];
    }
    if(det(p[0], p[1], p[2]) < 0) swap(p[1], p[2]);
    P q1 = (p[2] - p[0]).rot(pi / 3) + p[0];
    P q2 = (p[0] - p[1]).rot(pi / 3) + p[1];
    return isLL(L(q1, p[1]), L(q2, p[2]));
  }
  auto Rand = [&] () { return rand() % 10000 / 5000 * pi; };
  P ans(0, 0); rep(i, 0, n) ans = ans + p[i]; ans = ans / n;
  db len = 0; rep(i, 0, n) len += (ans - p[i]).len();
  db t = 10000; // modify
  while(t > eps) {
    db ang = Rand();
    P np(ans.x + t * sin(ang), ans.y + t * cos(ang));
    db k = 0; rep(i, 0, n) k += (np - p[i]).len();
    if(sign(len - k) > 0) ans = np, len = k;
    t *= 0.999;
  }
  return ans;
}
```

3.2 线段、直线、曲线

```
// 【直线交点】
P isLL(L l1, L l2) {
   db s1 = det(l2.b - l2.a, l1.a - l2.a);
   db s2 = -det(l2.b - l2.a, l1.b - l2.a);
   return (l1.a * s2 + l1.b * s1) / (s1 + s2);
}
P isLL(L l, db a, db b, db c) { // ax + by + c = 0
   db u = a * l.a.x + b * l.a.y + c;
   db v = -(a * l.b.x + b * l.b.y + c);
   return (l.a * v + l.b * u) / (u + v);
}
P isLL(db a0, db b0, db c0, db a1, db b1, db c1) {
   db d = a0 * b1 - a1 * b0;
   return P(b0 * c1 - b1 * c0, a1 * c0 - a0 * c1) / d;
}
// 【线相交判定】
```

```
bool isSSr(const L &a, const L &b){
   db c1 = det(a.t - a.s, b.s - a.s), c2 = det(a.t - a.s, b.t - a.s);
   db c3 = det(b.t - b.s, a.s - b.s), c4 = det(b.t - b.s, a.t - b.s);
   return sign(c1) * sign(c2) < 0 && sign(c3) * sign(c4) < 0;
}
bool isSS(L a,L b){
   db c1 = det(a.t - a.s, b.s - a.s), c2 = det(a.t - a.s, b.t - a.s);
   db c3 = det(b.t - b.s, a.s - b.s), c4 = det(b.t - b.s, a.t - b.s);
   return sign(c1) * sign(c2) <= 0 && sign(c3) * sign(c4) <= 0 &&
      sign(max(a.s.x, a.t.x) - min(b.s.x, b.t.x)) >= 0 &&
      sign(max(b.s.x, b.t.x) - min(a.s.x, a.t.x)) >= 0 &&
      sign(max(a.s.y, a.t.y) - min(b.s.y, b.t.y)) >= 0 &&
      sign(max(b.s.y, b.t.y) - min(a.s.y, a.t.y)) >= 0;
}
bool isLS(P a1, P a2, P b1, P b2) { // 判断直线线段是否相交（端点也算）
   db c1 = det(a2 - a1, b1 - a1), c2 = det(a2 - a1, b2 - a1);
   return sign(c1) * sign(c2) <= 0;
}
// 【点到线距离】
db disToL(L l, P p) {
  return fabs(det(l.a, p, l.b) / (l.b - l.a).len());
}
db disToS(L l, P p) {
  return sign(dot(l.a, p, l.b)) * sign(dot(l.b, p, l.a)) == 1 ? disToL(l, p) : min((p -
  l.a).len(), (p - l.b).len());
}
// 【线到线距离】
db disSS(L a, L b){
   if(isSS(a, b)) return 0;
   return min(min(disToSeg(b, a.s), disToSeg(b, a.t)), min(disToSeg(a, b.s), disToSeg(a,
    b.t)));
}
```

3.3 凸包

```
// 【求凸包】
vector<P> convexHull(vector<P> ps) {
  int n = sz(ps); if(n <= 1) return ps;
  sort(all(ps)); vector<P> qs;
  for(int i = 0; i < n; qs.pb(ps[i++])) {
    while(sz(qs) > 1 && sign(det(qs[sz(qs) - 2], qs.back(), ps[i])) <= 0) qs.pop_back();
  }
  for(int i = n - 2, t = sz(qs); i >= 0; qs.pb(ps[i--])) {
    while(sz(qs) > t && sign(det(qs[sz(qs) - 2], qs.back(), ps[i])) <= 0) qs.pop_back();
  }
  qs.pop_back(); return qs;
}
// 【凸包最远点对】
db diameter(vector<P> A) {
   int n = sz(A);
   if(n <= 1) return 0;
   int l = 0, r = 0;
   rep(i, 1, n) (A[i] < A[l]) && (l = i), (A[r] < A[i]) && (r = i);
   db res = (A[l]-A[r]).len();
   int i = l, j = r;
   do (++(det(A[(i + 1) % n]- A[i], A[(j + 1) % n] - A[j]) >= 0 ? j : i)) %= n,
      res = max(res, (A[i] - A[j]).len());
   while(i != l || j != r);
   return res;
}
// 【动态凸包】
// O(nlogn)
// 插入点，询问点在不在凸包内部（包括边界）
namespace DCH {
   map<int, P> h1, h2;
   bool ao(P a, P b, P c) {
      // 包括边界：小等于
      return (b.y - a.y) * 1ll * (c.x - b.x) <= (c.y - b.y) * 1ll * (b.x - a.x);
   }
   bool in(map<int, P> &h, P p) {
      if(!sz(h)) return 0;
      if(p.x < h.begin()->se.x || p.x > h.rbegin()->se.x) return 0;
      auto l = h.lower_bound(p.x);
      if(p.x == l->se.x) return p.y <= l->se.y;
      auto r = l--;
      return ao(l->se, p, r->se);
   }
   void ins(map<int, P> &h, P p) {
      if(in(h, p)) return ;
      h[p.x] = p;
      auto pos = h.find(p.x);
      while(1) {
         auto l = pos; if(l == h.begin()) break; --l;
         auto ll = l; if(ll == h.begin()) break; --ll;
         if(ao(ll->se, l->se, p)) h.erase(l); else break;
      }
      while(1) {
         auto r = pos; r++; if(r == h.end()) break;
         auto rr = r; rr++; if(rr == h.end()) break;
         if(ao(p, r->se, rr->se)) h.erase(r); else break;
      }
   }
   void ins(int x, int y) { ins(h1, P(x, y)); ins(h2, P(x, -y)); }
   bool in(int x, int y) { return in(h1, P(x, y)) && in(h2, P(x, -y)); }
}
// 【凸包交】
namespace ConvecIntersection{ // ?
   const int N = 1005;
   struct Rec {
      P d[10];int dn;// d[dn] = d[0]
      P operator [] (const int&n) {return d[n];}
   }r[N];
   typedef pair<db,int> pdi;
   int n;pdi res[1000005];
   db getLoc(P a,P b,P p){
      if(sgn(b.x - a.x)) return (p.x - a.x) / (b.x - a.x);
      return (p.y - a.y) / (b.y - a.y);
   }
   db work() {
      db rt=0;
    rep(i,0,n) rep(j,0,r[i].dn){
      int sz=0;
      res[sz++] = pdi(0,0);res[sz++] = pdi(1,0);
      rep(t,0,n) {
        if(t == i) continue;
        rep(g,0,r[t].dn) {
          int du = sgn((r[i][j+1] - r[i][j]) / (r[t][g] - r[i][j]));
          int dv = sgn((r[i][j+1] - r[i][j]) / (r[t][g+1] - r[i][j]));
          if(!du && !dv) {
            if(sgn((r[i][j+1] - r[i][j]) * (r[t][g+1] - r[t][g])) < 0 || i < t){
              res[sz++] = pdi(getLoc(r[i][j] , r[i][j+1] , r[t][g]) , 1);
              res[sz++] = pdi(getLoc(r[i][j] , r[i][j+1] , r[t][g+1]) , -1);
            }} else {
              db s1 = (r[i][j] - r[t][g]) / (r[t][g+1] - r[t][g]);
              db s2 = (r[t][g+1] - r[t][g]) / (r[i][j+1] - r[t][g]);
              if(du >= 0 && dv < 0) res[sz++] = pdi(s1 / (s1 + s2) , 1);
              else if(du < 0 && dv >= 0) res[sz++] = pdi(s1 / (s1 + s2) , -1);
            }}}
      sort(res , res + sz);
      int cnt = 0; --sz;
      rep(t,0,sz) {
        cnt += res[t].se;
        if(cnt == 0 && sgn(res[t].fi - res[t+1].fi)) {
          db a = res[t].fi;
          if(a < 0) a = 0; if(a > 1) break;
          db b = res[t+1].fi;
          if(b < 0) continue; if(b > 1) b = 1;
          rt += ((r[i][j+1] - r[i][j]) * a + r[i][j]) / ((r[i][j+1]-r[i][j]) * b +
          r[i][j]);
        }}}
    return rt / 2;}}
```

3.4 三角形

```
// 【心】
P outC(P A, P B, P C) { // 外心
  P b = B - A, c = C - A;
  db dB = b.len2(), dC = c.len2(), d = 2 * det(b, c);
  return A - P(b.y * dC - c.y * dB, c.x * dB - b.x * dC) / d;
}
P baryC(P p[], int n) { // 重心
  P fz(0, 0); db fm = 0;
  rep(i, 1, n - 1) {
    db t = det(p[0], p[i], p[i + 1]);
    fm += t;
    fz = fz + (p[0] + p[i] + p[i + 1]) * t / 3;
  }
  return fz / fm;
}
```

3.5 多边形

```
// 【平面图欧拉定理】 V + F - E = 2
// 【简单多边形求面积交】
```

```
db polyInter(vector<P> &p, vector<P> &q) {
  int n = sz(p), m = sz(q);
  if(n < 3 || m < 3) return 0;
  // if(area(p) < 0) reverse(all(p));
  // if(area(q) < 0) reverse(all(q));
  db ans = 0;
  rep(i, 1, n - 1) {
    P p1 = p[i], p2 = p[i + 1];
    bool f1 = 0;
    if(det(p[0], p1, p2) < 0) swap(p1, p2), f1 = 1;
    rep(j, 1, m - 1) {
      P q1 = q[j], q2 = q[j + 1];
      bool f2 = 0;
      if(det(q[0], q1, q2) < 0) swap(q1, q2), f2 = 1;
      vector<P> ps({p[0], p1, p2});
      convexCut(ps, L(q[0], q1));
      convexCut(ps, L(q1, q2));
      convexCut(ps, L(q2, q[0]));
      db res = f1 == f2 ? area(ps) : -area(ps);
      ans += res;
    }
  }
  return fabs(ans);
}
```

3.6 圆

```
// 【两圆关系】
// 注意相等关系
// 相离4: 外切3: 相交2: 内切1: 内含0:
int relCC(C A, C B) { // 两圆关系
  db dis = (A.o - B.o).len();
  if(sign(dis - (A.r + B.r)) == 1) return 4;
  if(sign(dis - (A.r + B.r)) == 0) return 3;
  if(sign(dis - fabs(A.r - B.r)) == 1) return 2;
  if(sign(dis - fabs(A.r - B.r)) == 0) return 1;
  return 0;
}
// 【点圆切点】
bool tanCP(O c, P p0, P &p1, P &p2) {
  db x = (p0 - c.o).len2(), d = x - c.r * c.r;
  if(d < eps) return 0;
  P p = (p0 - c.o) * (c.r * c.r / x);
  P det = ((p0 - c.o) * (-c.r * sqrt(d) / x)).rot90();
  p1 = c.o + p + det;
  p2 = c.o + p - det;
  return 1;
}
// 【圆圆切点】
vector<P> tanCC(const C &c1, const C &c2) {
  vector<P> res;
  db dis = (c1.o - c2.o).len();
  if(sign(dis - (c1.r + c2.r)) == 0) {
    res.pb(c1.o + (c2.o - c1.o) * c1.r / (c1.r + c2.r));
  }
```

```
   if(sign(dis - fabs(c1.r - c2.r) == 0)) {
      res.pb(c1.o + (c2.o - c1.o) * c1.r / (c1.r - c2.r));
   }
   return res;
}
// 【直线和圆求交】
bool isCL(O a, L l, P &p1, P &p2) {
  db x = dot(l.a - a.o, l.b - l.a);
  db y = (l.b - l.a).len2();
  db d = x * x - y * ((l.a - a.o).len2() - a.r * a.r);
  if(sign(d) < 0) return 0;
  d = max(d, 0.);
  P p = l.a - ((l.b - l.a) * (x / y)), det = (l.b - l.a) * (sqrt(d) / y);
  p1 = p - det, p2 = p + det; // dir : l.a -> l.b
  return 1;
}
// 【圆与三角形交面积】
db areaCT(db r,P s,P t) { // 需要除 2
   P p1, p2;
   bool f = isCL(C(P(0, 0), r), L(s, t), p1, p2);
   if(!f) return r * r * rad(s, t);
   bool b1 = sign(s.len2() - r * r) == 1 , b2 = sign(t.len2() - r * r) == 1;
   if(b1 && b2) {
      if(sign(dot(s - p1, t - p1)) <= 0 && sign(dot(s - p2, t - p2) <= 0))
         return r * r * (rad(s, p1) + rad(p2, t)) + det(p1, p2);
      else return r * r * rad(s, t);
   } else if(b1) return r * r * rad(s, p1) + det(p1, t);
   else if(b2) return r * r * rad(p2, t) + det(s, p2);
   return det(s, t);
}
// 【圆与多边形交面积】
db areaCPoly(C c, vector<P> p) {
   int n = sz(p);
   db ans = 0;
   rep(i, 0, n) {
      P u = p[i], v = p[(i + 1) % n];
      ans += areaCT(c.r, u - c.o, v - c.o);
   }
   return fabs(ans) / 2;
}
// 【圆交】
namespace CircleIntersection{ // ?
   struct E{
      P p;T ang;int delta;
      E(){} E(P p,T ang,int delta):p(p),ang(ang),delta(delta){}
      bool operator < (const E&b) const {return ang<b.ang;}
   };
   bool overlap(C a,C b) {return sgn(a.r-b.r-abs(a.o-b.o))>=0;}
   void solve(C *c,int n,T *ans) {
      memset(ans , 0 , sizeof(T) * (n + 1));
      rep(i,0,n) {
         int cnt=1;
         vector<E> evt;
         rep(j,0,i) if(c[i]==c[j]) cnt++;
         rep(j,0,n) if(j!=i&&!(c[i]==c[j])&&overlap(c[j],c[i])) cnt++;
         rep(j,0,n) if(j!=i){
            vector<P> pts=insCC(c[i],c[j]);
            if(sz(pts)) {
               T a[2];
               rep(j,0,2) a[j]=(pts[j]-c[i].o).arg();
               evt.pb(E(pts[0],a[0],1));
               evt.pb(E(pts[1],a[1],-1));
               cnt += a[0] > a[1];
            }
         }
         if(!sz(evt)) ans[cnt] += pi*c[i].r*c[i].r;
         else{
            sort(all(evt));
            evt.pb(evt.front());
            rep(j,0,sz(evt)-1) {
               cnt+=evt[j].delta;
               ans[cnt] += evt[j].p / evt[j+1].p / 2;
               db ang = evt[j + 1].ang - evt[j].ang;
               if(ang < 0) ang += pi * 2;
               ans[cnt] += ang * c[i].r * c[i].r / 2 - sin(ang) * c[i].r * c[i].r / 2;
            }}}}}
```

3.7 3D

```
// 【最小球覆盖】
P3 MinSphere(vector<P3> p) {
  int n = sz(p); assert(n);
  db t = 1; P3 ans(0, 0, 0);
  rep(i, 0, n) ans = ans + p[i]; ans = ans / n;
  while(t > eps) {
    int j = -1; db ret = -1;
    rep(i, 0, n) {
      db tmp = (p[i] - ans).len();
      if(ret < tmp) ret = tmp, j = i;
    }
    ans = ans + (p[j] - ans) * t;
    t *= 0.999;
  }
  return ans;
}
// 【三维向量变换】
struct Mat {
  db a[4][4];
  void set() { rep(i, 0, 4) rep(j, 0, 4) a[i][j] = 0; }
  void e() { rep(i, 0, 4) a[i][i] = 1; }
  Mat operator * (const Mat &c) {
    Mat r; r.set();
    rep(i, 0, 4) rep(j, 0, 4) rep(k, 0, 4) r.a[i][j] += a[i][k] * c.a[k][j];
    return r;
  }
};
Mat kpow(Mat a, int b) {
  Mat r; r.set(); r.e();
  while(b) {
    if(b & 1) r = r * a;
    a = a * a;
    b >>= 1;
  }
  return r;
}
Mat translate(db tx, db ty, db tz) { // 平移，以下矩阵均为左乘
  db p[4][4] = {
    1, 0, 0, tx,
    0, 1, 0, ty,
    0, 0, 1, tz,
    0, 0, 0, 1};
  Mat r; rep(i, 0, 4) rep(j, 0, 4) r.a[i][j] = p[i][j]; return r;
}
Mat scale(db a, db b, db c) { // 缩放
  db p[4][4] = {
    a, 0, 0, 0,
    0, b, 0, 0,
    0, 0, c, 0,
    0, 0, 0, 1};
  Mat r; rep(i, 0, 4) rep(j, 0, 4) r.a[i][j] = p[i][j]; return r;
}
Mat rotate(P3 s, db a) { // 绕 s 为轴旋转 a 度，右手方向
  db l = s.len(), x = s.x / l, y = s.y / l, z = s.z / l, si = sin(a), co = cos(a);
  db p[4][4] = {
    co + (1 - co) * x * x, (1 - co) * x * y - si * z, (1 - co) * x * z + si * y, 0,
    (1 - co) * y * x + si * z, co + (1 - co) * y * y, (1 - co) * y * z - si * x, 0,
    (1 - co) * z * x - si * y, (1 - co) * z * y + si * x, co + (1 - co) * z * z, 0,
    0, 0, 0, 1};
  Mat r; rep(i, 0, 4) rep(j, 0, 4) r.a[i][j] = p[i][j]; return r;
}
```

3.8 1、基础点、向量

```
struct P {
   int quad() const { return sign(y) > 0 || (sign(y) == 0 && sign(x) >= 0); }
   P rot90() { return P(-y, x); }
   P rot(db a) { return P(cos(a) * x - sin(a) * y, cos(a) * y + sin(a) * x); }
   P norm() { return *this / len(); }
};
db rad(P p1, P p2) { return atan2l(det(p1, p2), dot(p1, p2)); } // p1 与 p2 的夹角，有方
向
bool cmp(const pii &a, const pii &b) { // 级角排序
   int o = a > pii(0, 0), t = b > pii(0, 0);
   if(o != t) return o < t;
   return det(a, b) > 0;
}
// 【点集中最近点对】
namespace NearestPoints { // sz(A) <= 1e5
   db solve(int l, int r, vector<P> &p) {
      if(l == r) return 1e100;
      int m = l + r >> 1;
      db Xm = p[m].x, lim = min(solve(l, m, p), solve(m + 1, r, p));
      inplace_merge(p.begin() + l, p.begin() + m + 1, p.begin() + r + 1, [&](P a, P b){
      return a.y < b.y;});
      vector<P> V;
      rep(i, l, r + 1) if(fabs(p[i].x - Xm) <= lim) V.pb(p[i]);
      rep(i, 0, sz(V)) rep(j, i + 1, sz(V)) {
         if(fabs(V[j].y - V[i].y) >= lim) break;
         T dis = (V[i] - V[j]).len();
         lim = min(lim, dis);
      }
      return lim;
   }
   db solve(vector<P> A) {
      sort(all(A), [&](P a, P b){return a.x < b.x;});
      return solve(0, sz(A) - 1, A);
   }
}
// 【最小圆覆盖】
C Mincir(P *p,int n){
   random_shuffle(p , p + n);
   P o = p[0];db r = 0;
   rep(i,1,n) {
      if(sgn(abs(o-p[i])-r) <= 0) continue;
      o = p[i] , r = 0;
      rep(j,0,i) {
         if(sgn(abs(o-p[j])-r) <= 0) continue;
         o = (p[i] + p[j]) / 2 , r = abs(o-p[j]);
         rep(k,0,j) {
            if(sgn(abs(o-p[k])-r) <= 0) continue;
            o = outC(p[i],p[j],p[k]) , r = abs(o-p[k]);
         }}}
   return C(o,r);
}
// 【费马点】
// sqrt((a ^ 2 + b ^ 2 + c ^ 2 + 4 * sqrt(3) * area) / 2)
// 如果有重点，大于 2 的直接用模拟退火法
P fermat(vector<P> p) {
  int n = sz(p); assert(n);
  if(n == 1) return p[0];
  if(n == 2) return (p[0] + p[1]) / 2;
  if(n == 3) {
    db a[3];
    rep(i, 0, 3) a[i] = (p[(i + 2) % 3] - p[(i + 1) % 3]).len();
    rep(i, 0, 3) {
      int j = (i + 1) % 3, k = (i + 2) % 3;
      if(sign(a[i] * a[i] - a[j] * a[j] - a[k] * a[k] - a[j] * a[k]) >= 0) return p[i];
    }
    if(det(p[0], p[1], p[2]) < 0) swap(p[1], p[2]);
    P q1 = (p[2] - p[0]).rot(pi / 3) + p[0];
    P q2 = (p[0] - p[1]).rot(pi / 3) + p[1];
    return isLL(L(q1, p[1]), L(q2, p[2]));
  }
  auto Rand = [&] () { return rand() % 10000 / 5000 * pi; };
  P ans(0, 0); rep(i, 0, n) ans = ans + p[i]; ans = ans / n;
  db len = 0; rep(i, 0, n) len += (ans - p[i]).len();
  db t = 10000; // modify
  while(t > eps) {
    db ang = Rand();
    P np(ans.x + t * sin(ang), ans.y + t * cos(ang));
    db k = 0; rep(i, 0, n) k += (np - p[i]).len();
    if(sign(len - k) > 0) ans = np, len = k;
    t *= 0.999;
  }
  return ans;
}
```

3.9 2、线段、直线、曲线

```
// 【直线交点】
P isLL(L l1, L l2) {
   db s1 = det(l2.b - l2.a, l1.a - l2.a);
   db s2 = -det(l2.b - l2.a, l1.b - l2.a);
   return (l1.a * s2 + l1.b * s1) / (s1 + s2);
}
P isLL(L l, db a, db b, db c) { // ax + by + c = 0
   db u = a * l.a.x + b * l.a.y + c;
   db v = -(a * l.b.x + b * l.b.y + c);
   return (l.a * v + l.b * u) / (u + v);
}
P isLL(db a0, db b0, db c0, db a1, db b1, db c1) {
   db d = a0 * b1 - a1 * b0;
   return P(b0 * c1 - b1 * c0, a1 * c0 - a0 * c1) / d;
}
// 【线相交判定】
bool isSSr(const L &a, const L &b){
   db c1 = det(a.t - a.s, b.s - a.s), c2 = det(a.t - a.s, b.t - a.s);
   db c3 = det(b.t - b.s, a.s - b.s), c4 = det(b.t - b.s, a.t - b.s);
   return sign(c1) * sign(c2) < 0 && sign(c3) * sign(c4) < 0;
}
bool isSS(L a,L b){
   db c1 = det(a.t - a.s, b.s - a.s), c2 = det(a.t - a.s, b.t - a.s);
   db c3 = det(b.t - b.s, a.s - b.s), c4 = det(b.t - b.s, a.t - b.s);
   return sign(c1) * sign(c2) <= 0 && sign(c3) * sign(c4) <= 0 &&
      sign(max(a.s.x, a.t.x) - min(b.s.x, b.t.x)) >= 0 &&
      sign(max(b.s.x, b.t.x) - min(a.s.x, a.t.x)) >= 0 &&
      sign(max(a.s.y, a.t.y) - min(b.s.y, b.t.y)) >= 0 &&
      sign(max(b.s.y, b.t.y) - min(a.s.y, a.t.y)) >= 0;
}
bool isLS(P a1, P a2, P b1, P b2) { // 判断直线线段是否相交（端点也算）
   db c1 = det(a2 - a1, b1 - a1), c2 = det(a2 - a1, b2 - a1);
   return sign(c1) * sign(c2) <= 0;
}
// 【点到线距离】
db disToL(L l, P p) {
  return fabs(det(l.a, p, l.b) / (l.b - l.a).len());
}
db disToS(L l, P p) {
  return sign(dot(l.a, p, l.b)) * sign(dot(l.b, p, l.a)) == 1 ? disToL(l, p) : min((p -
  l.a).len(), (p - l.b).len());
}
// 【线到线距离】
db disSS(L a, L b){
   if(isSS(a, b)) return 0;
   return min(min(disToSeg(b, a.s), disToSeg(b, a.t)), min(disToSeg(a, b.s), disToSeg(a,
    b.t)));
}
```

3.10 3、凸包

```
// 【求凸包】
vector<P> convexHull(vector<P> ps) {
  int n = sz(ps); if(n <= 1) return ps;
  sort(all(ps)); vector<P> qs;
  for(int i = 0; i < n; qs.pb(ps[i++])) {
    while(sz(qs) > 1 && sign(det(qs[sz(qs) - 2], qs.back(), ps[i])) <= 0) qs.pop_back();
  }
  for(int i = n - 2, t = sz(qs); i >= 0; qs.pb(ps[i--])) {
    while(sz(qs) > t && sign(det(qs[sz(qs) - 2], qs.back(), ps[i])) <= 0) qs.pop_back();
  }
  qs.pop_back(); return qs;
}
// 【凸包最远点对】
db diameter(vector<P> A) {
   int n = sz(A);
   if(n <= 1) return 0;
   int l = 0, r = 0;
   rep(i, 1, n) (A[i] < A[l]) && (l = i), (A[r] < A[i]) && (r = i);
   db res = (A[l]-A[r]).len();
   int i = l, j = r;
   do (++(det(A[(i + 1) % n]- A[i], A[(j + 1) % n] - A[j]) >= 0 ? j : i)) %= n,
      res = max(res, (A[i] - A[j]).len());
   while(i != l || j != r);
   return res;
}
// 【动态凸包】
// O(nlogn)
// 插入点，询问点在不在凸包内部（包括边界）
namespace DCH {
   map<int, P> h1, h2;
   bool ao(P a, P b, P c) {
      // 包括边界：小等于
      return (b.y - a.y) * 1ll * (c.x - b.x) <= (c.y - b.y) * 1ll * (b.x - a.x);
   }
   bool in(map<int, P> &h, P p) {
      if(!sz(h)) return 0;
      if(p.x < h.begin()->se.x || p.x > h.rbegin()->se.x) return 0;
      auto l = h.lower_bound(p.x);
      if(p.x == l->se.x) return p.y <= l->se.y;
      auto r = l--;
      return ao(l->se, p, r->se);
   }
   void ins(map<int, P> &h, P p) {
      if(in(h, p)) return ;
      h[p.x] = p;
      auto pos = h.find(p.x);
      while(1) {
         auto l = pos; if(l == h.begin()) break; --l;
         auto ll = l; if(ll == h.begin()) break; --ll;
         if(ao(ll->se, l->se, p)) h.erase(l); else break;
      }
      while(1) {
         auto r = pos; r++; if(r == h.end()) break;
         auto rr = r; rr++; if(rr == h.end()) break;
         if(ao(p, r->se, rr->se)) h.erase(r); else break;
      }
   }
   void ins(int x, int y) { ins(h1, P(x, y)); ins(h2, P(x, -y)); }
   bool in(int x, int y) { return in(h1, P(x, y)) && in(h2, P(x, -y)); }
}
// 【凸包交】
namespace ConvecIntersection{ // ?
   const int N = 1005;
   struct Rec {
      P d[10];int dn;// d[dn] = d[0]
      P operator [] (const int&n) {return d[n];}
   }r[N];
   typedef pair<db,int> pdi;
   int n;pdi res[1000005];
   db getLoc(P a,P b,P p){
      if(sgn(b.x - a.x)) return (p.x - a.x) / (b.x - a.x);
      return (p.y - a.y) / (b.y - a.y);
   }
   db work() {
      db rt=0;
      rep(i,0,n) rep(j,0,r[i].dn){
         int sz=0;
         res[sz++] = pdi(0,0);res[sz++] = pdi(1,0);
         rep(t,0,n) {
            if(t == i) continue;
            rep(g,0,r[t].dn) {
               int du = sgn((r[i][j+1] - r[i][j]) / (r[t][g] - r[i][j]));
               int dv = sgn((r[i][j+1] - r[i][j]) / (r[t][g+1] - r[i][j]));
               if(!du && !dv) {
                  if(sgn((r[i][j+1] - r[i][j]) * (r[t][g+1] - r[t][g])) < 0 || i < t){
                     res[sz++] = pdi(getLoc(r[i][j] , r[i][j+1] , r[t][g]) , 1);
                     res[sz++] = pdi(getLoc(r[i][j] , r[i][j+1] , r[t][g+1]) , -1);
                  }} else {
                     db s1 = (r[i][j] - r[t][g]) / (r[t][g+1] - r[t][g]);
                     db s2 = (r[t][g+1] - r[t][g]) / (r[i][j+1] - r[t][g]);
                     if(du >= 0 && dv < 0) res[sz++] = pdi(s1 / (s1 + s2) , 1);
                     else if(du < 0 && dv >= 0) res[sz++] = pdi(s1 / (s1 + s2) , -1);
                  }}}
         sort(res , res + sz);
         int cnt = 0; --sz;
         rep(t,0,sz) {
            cnt += res[t].se;
            if(cnt == 0 && sgn(res[t].fi - res[t+1].fi)) {
               db a = res[t].fi;
               if(a < 0) a = 0; if(a > 1) break;
               db b = res[t+1].fi;
               if(b < 0) continue; if(b > 1) b = 1;
               rt += ((r[i][j+1] - r[i][j]) * a + r[i][j]) / ((r[i][j+1]-r[i][j]) * b +
               r[i][j]);
            }}}
      return rt / 2;}}
```

3.11 4、三角形

```
// 【心】
P outC(P A, P B, P C) { // 外心
   P b = B - A, c = C - A;
   db dB = b.len2(), dC = c.len2(), d = 2 * det(b, c);
   return A - P(b.y * dC - c.y * dB, c.x * dB - b.x * dC) / d;
}
P baryC(P p[], int n) { // 重心
   P fz(0, 0); db fm = 0;
   rep(i, 1, n - 1) {
      db t = det(p[0], p[i], p[i + 1]);
      fm += t;
      fz = fz + (p[0] + p[i] + p[i + 1]) * t / 3;
   }
   return fz / fm;
}
```

3.12 5、多边形

```
// 【平面图欧拉定理】 V + F - E = 2
// 【简单多边形求面积交】
db polyInter(vector<P> &p, vector<P> &q) {
   int n = sz(p), m = sz(q);
   if(n < 3 || m < 3) return 0;
   // if(area(p) < 0) reverse(all(p));
   // if(area(q) < 0) reverse(all(q));
   db ans = 0;
   rep(i, 1, n - 1) {
      P p1 = p[i], p2 = p[i + 1];
      bool f1 = 0;
      if(det(p[0], p1, p2) < 0) swap(p1, p2), f1 = 1;
      rep(j, 1, m - 1) {
         P q1 = q[j], q2 = q[j + 1];
         bool f2 = 0;
         if(det(q[0], q1, q2) < 0) swap(q1, q2), f2 = 1;
         vector<P> ps({p[0], p1, p2});
         convexCut(ps, L(q[0], q1));
         convexCut(ps, L(q1, q2));
         convexCut(ps, L(q2, q[0]));
         db res = f1 == f2 ? area(ps) : -area(ps);
         ans += res;
      }
   }
   return fabs(ans);
}
```

3.13 6、圆

```
// 【两圆关系】
// 注意相等关系
// 相离4: 外切3: 相交2: 内切1: 内含0:
int relCC(C A, C B) { // 两圆关系
   db dis = (A.o - B.o).len();
   if(sign(dis - (A.r + B.r)) == 1) return 4;
   if(sign(dis - (A.r + B.r)) == 0) return 3;
   if(sign(dis - fabs(A.r - B.r)) == 1) return 2;
   if(sign(dis - fabs(A.r - B.r)) == 0) return 1;
   return 0;
}
// 【点圆切点】
bool tanCP(O c, P p0, P &p1, P &p2) {
   db x = (p0 - c.o).len2(), d = x - c.r * c.r;
   if(d < eps) return 0;
   P p = (p0 - c.o) * (c.r * c.r / x);
   P det = ((p0 - c.o) * (-c.r * sqrt(d) / x)).rot90();
   p1 = c.o + p + det;
   p2 = c.o + p - det;
   return 1;
}
// 【圆圆切点】
vector<P> tanCC(const C &c1, const C &c2) {
   vector<P> res;
   db dis = (c1.o - c2.o).len();
   if(sign(dis - (c1.r + c2.r)) == 0) {
      res.pb(c1.o + (c2.o - c1.o) * c1.r / (c1.r + c2.r));
   }
   if(sign(dis - fabs(c1.r - c2.r) == 0)) {
      res.pb(c1.o + (c2.o - c1.o) * c1.r / (c1.r - c2.r));
   }
   return res;
}
// 【直线和圆求交】
bool isCL(O a, L l, P &p1, P &p2) {
  db x = dot(l.a - a.o, l.b - l.a);
  db y = (l.b - l.a).len2();
  db d = x * x - y * ((l.a - a.o).len2() - a.r * a.r);
  if(sign(d) < 0) return 0;
  d = max(d, 0.);
  P p = l.a - ((l.b - l.a) * (x / y)), det = (l.b - l.a) * (sqrt(d) / y);
  p1 = p - det, p2 = p + det; // dir : l.a -> l.b
  return 1;
}
// 【圆与三角形交面积】
db areaCT(db r,P s,P t) { // 需要除 2
   P p1, p2;
   bool f = isCL(C(P(0, 0), r), L(s, t), p1, p2);
   if(!f) return r * r * rad(s, t);
   bool b1 = sign(s.len2() - r * r) == 1 , b2 = sign(t.len2() - r * r) == 1;
   if(b1 && b2) {
      if(sign(dot(s - p1, t - p1)) <= 0 && sign(dot(s - p2, t - p2) <= 0))
         return r * r * (rad(s, p1) + rad(p2, t)) + det(p1, p2);
      else return r * r * rad(s, t);
   } else if(b1) return r * r * rad(s, p1) + det(p1, t);
   else if(b2) return r * r * rad(p2, t) + det(s, p2);
   return det(s, t);
}
// 【圆与多边形交面积】
db areaCPoly(C c, vector<P> p) {
   int n = sz(p);
   db ans = 0;
   rep(i, 0, n) {
      P u = p[i], v = p[(i + 1) % n];
      ans += areaCT(c.r, u - c.o, v - c.o);
   }
   return fabs(ans) / 2;
}
// 【圆交】
namespace CircleIntersection{ // ?
   struct E{
      P p;T ang;int delta;
      E(){} E(P p,T ang,int delta):p(p),ang(ang),delta(delta){}
      bool operator < (const E&b) const {return ang<b.ang;}
   };
   bool overlap(C a,C b) {return sgn(a.r-b.r-abs(a.o-b.o))>=0;}
   void solve(C *c,int n,T *ans) {
      memset(ans , 0 , sizeof(T) * (n + 1));
      rep(i,0,n) {
         int cnt=1;
         vector<E> evt;
         rep(j,0,i) if(c[i]==c[j]) cnt++;
         rep(j,0,n) if(j!=i&&!(c[i]==c[j])&&overlap(c[j],c[i])) cnt++;
         rep(j,0,n) if(j!=i){
            vector<P> pts=insCC(c[i],c[j]);
            if(sz(pts)) {
               T a[2];
               rep(j,0,2) a[j]=(pts[j]-c[i].o).arg();
               evt.pb(E(pts[0],a[0],1));
               evt.pb(E(pts[1],a[1],-1));
               cnt += a[0] > a[1];
            }
         }
         if(!sz(evt)) ans[cnt] += pi*c[i].r*c[i].r;
         else{
            sort(all(evt));
            evt.pb(evt.front());
            rep(j,0,sz(evt)-1) {
               cnt+=evt[j].delta;
               ans[cnt] += evt[j].p / evt[j+1].p / 2;
               db ang = evt[j + 1].ang - evt[j].ang;
               if(ang < 0) ang += pi * 2;
               ans[cnt] += ang * c[i].r * c[i].r / 2 - sin(ang) * c[i].r * c[i].r / 2;
            }}}}}
```

3.14 7、3D

```
// 【最小球覆盖】
P3 MinSphere(vector<P3> p) {
  int n = sz(p); assert(n);
  db t = 1; P3 ans(0, 0, 0);
  rep(i, 0, n) ans = ans + p[i]; ans = ans / n;
  while(t > eps) {
    int j = -1; db ret = -1;
    rep(i, 0, n) {
      db tmp = (p[i] - ans).len();
      if(ret < tmp) ret = tmp, j = i;
    }
    ans = ans + (p[j] - ans) * t;
    t *= 0.999;
  }
  return ans;
}
// 【三维向量变换】
struct Mat {
  db a[4][4];
  void set() { rep(i, 0, 4) rep(j, 0, 4) a[i][j] = 0; }
  void e() { rep(i, 0, 4) a[i][i] = 1; }
  Mat operator * (const Mat &c) {
    Mat r; r.set();
    rep(i, 0, 4) rep(j, 0, 4) rep(k, 0, 4) r.a[i][j] += a[i][k] * c.a[k][j];
    return r;
  }
};
Mat kpow(Mat a, int b) {
  Mat r; r.set(); r.e();
  while(b) {
    if(b & 1) r = r * a;
    a = a * a;
    b >>= 1;
  }
  return r;
}
Mat translate(db tx, db ty, db tz) { // 平移，以下矩阵均为左乘
  db p[4][4] = {
    1, 0, 0, tx,
    0, 1, 0, ty,
    0, 0, 1, tz,
    0, 0, 0, 1};
  Mat r; rep(i, 0, 4) rep(j, 0, 4) r.a[i][j] = p[i][j]; return r;
}
Mat scale(db a, db b, db c) { // 缩放
  db p[4][4] = {
    a, 0, 0, 0,
    0, b, 0, 0,
    0, 0, c, 0,
    0, 0, 0, 1};
  Mat r; rep(i, 0, 4) rep(j, 0, 4) r.a[i][j] = p[i][j]; return r;
}
Mat rotate(P3 s, db a) { // 绕 s 为轴旋转 a 度，右手方向
  db l = s.len(), x = s.x / l, y = s.y / l, z = s.z / l, si = sin(a), co = cos(a);
  db p[4][4] = {
    co + (1 - co) * x * x, (1 - co) * x * y - si * z, (1 - co) * x * z + si * y, 0,
    (1 - co) * y * x + si * z, co + (1 - co) * y * y, (1 - co) * y * z - si * x, 0,
    (1 - co) * z * x - si * y, (1 - co) * z * y + si * x, co + (1 - co) * z * z, 0,
    0, 0, 0, 1};
  Mat r; rep(i, 0, 4) rep(j, 0, 4) r.a[i][j] = p[i][j]; return r;
}
```

3.15 HalfPlane_n2

```
// l: a->b 逆时针方向
void convexCut(vector<P> &p, L l) {
   vector<P> q;
   rep(i, 0, sz(p)) {
      P p1 = p[i], p2 = p[(i + 1) % sz(p)];
      int d1 = sign(det(l.a, l.b, p1));
      int d2 = sign(det(l.a, l.b, p2));
      if(d1 >= 0) q.pb(p1);
      if(d1 * d2 < 0) q.pb(isLL(L(p1, p2), l));
   } p = q;
}
// ax + by + c >= 0
void convexCut(vector<P> &p, db a, db b, db c) {
   vector<P> q;
   rep(i, 0, sz(p)) {
      P p1 = p[i], p2 = p[(i + 1) % sz(p)];
      int d1 = sign(a * p1.x + b * p1.y + c);
      int d2 = sign(a * p2.x + b * p2.y + c);
      if(d1 >= 0) q.pb(p1);
      if(d1 * d2 < 0) q.pb(isLL(L(p1, p2), a, b, c));
   } p = q;
}
```

3.16 HalfPlane_nlogn

```
struct P {
   int quad() const { return sign(y) > 0 || (sign(y) == 0 && sign(x) >= 0); }
};
struct L {
   // ax + by + c >= 0, (a != 0 || b != 0)
   L(db a, db b, db c) {
      if(sign(a)==0) {
         this->a=P(0,-c/b);this->b=P(sign(b),-c/b);
      } else if(sign(b)==0) {
         this->a=P(-c/a,0);this->b=P(-c/a,-sign(a));
      } else {
         if(sign(c)!=0) {
            int x=sign(c)*sign(det(P(-c/a,0), P(0,-c/b)));
            if(x==1) this->a=P(-c/a,0),this->b=P(0,-c/b);
            else this->a=P(0,-c/b),this->b=P(-c/a,0);
         } else {
            this->a=P(0,0);this->b=P(sign(b),sign(b)*(-a/b));
         }
      }
   }
   bool includer(const P &p) const { return sign(det(b - a, p - a)) > 0; }
   bool include(const P &p) const { return sign(det(b - a, p - a)) >= 0; }
   // 向内（右手方向）推
   L push(db len) {
      P det = (b - a).rot90().norm() * len;
      return L(a + det, b + det);
   }
};
bool sameDir(L l0, L l1) {
   P a = l0.a - l0.b, b = l1.a - l1.b;
   return sign(det(a, b)) == 0 && sign(dot(a, b)) == 1;
}
bool operator < (const P &a,  const P &b) {
   if(a.quad() != b.quad()) return a.quad() < b.quad();
   return sign(det(a, b)) > 0;
}
bool operator < (const L &l0, const L &l1) {
   if(sameDir(l0, l1)) return l1.includer(l0.a);
   return (l0.b - l0.a) < (l1.b - l1.a);
}
bool check(L u, L v, L w) { return w.include(isLL(u, v)); }
deque<L> halfPlane(vector<L> l) {
   sort(all(l)); deque<L> q;
   rep(i, 0, sz(l)) {
      if(i && sameDir(l[i], l[i - 1])) continue;
      while(sz(q) > 1 && !check(q[sz(q) - 2], q.back(), l[i])) q.pop_back();
      while(sz(q) > 1 && !check(q[1], q[0], l[i])) q.pop_front();
      q.pb(l[i]);
   }
   while(sz(q) > 2 && !check(q[sz(q) - 2], q.back(), q[0])) q.pop_back();
   while(sz(q) > 2 && !check(q[1], q[0], q.back())) q.pop_front();
   return q;
}
```

3.17 MaxAreaPoly

```
ld solve_poly(vi &S) {
   assert(sz(S) > 0);
   int sum = 0, hi = S[0];
   vi vals;
   rep(i, 1, sz(S)) {
      int cur = S[i];
      if (cur > hi) swap(cur, hi);
      sum += cur;
      vals.pb(cur);
   }
   if (sum <= hi) return 0;
   auto getAngle = [&](ld D) -> ld{
      ld tot = 0;
      for (int l : vals) tot += 2 * asin(ld(l) / ld(D));
      return tot;
   };
   bool isReflex = (getAngle(hi) < PI);
   auto tooSmall = [&](ld D) {
      ld ang = getAngle(D);
      ld hiAng = 2 * asin(ld(hi) / ld(D));
      if (isReflex) return ang < hiAng;
      else return ang + hiAng >= 2 * PI;
   };
   ld mi = hi, ma = hi + 1;
   int numExpand = 0;
   while (tooSmall(ma)) numExpand++, ma += (ma - mi);
   rep(tim, 0, 50 + numExpand) {
      ld md = mi + (ma - mi) / 2;
      if (tooSmall(md)) mi = md;
      else ma = md;
   }
   ld D = mi, area = 0;
   for (int l : vals) area += ld(l) * sqrt(ld(D) * ld(D) - ld(l) * ld(l)) / 4;
   ld hiArea = ld(hi) * sqrt(ld(D) * ld(D) - ld(hi) * ld(hi)) / 4;
   if (isReflex) area -= hiArea;
   else area += hiArea;
   return area;
}
```

3.18 MaxAreaTri

```
// O(n ^ 2)
void maxAreaTri(P *p, int n, P &a, P &b, P &c) {
   int i = 0, j = 1, k = 2;
   a = p[i], b = p[j], c = p[k];
   T res = area(a, b, c), cur = res, tmp;
   do {
      while(1) {
         while(cur <= (tmp = area(p[i], p[j], p[(k + 1) % n]))) (++k) %= n, cur = tmp;
         if(cur <= (tmp = area(p[i], p[(j + 1) % n], p[k]))) (++j) %= n, cur = tmp;
         else break;
      }
      if(cur > res) a = p[i], b = p[j], c = p[k], res = cur;
      (++i) %= n;
      if(i == j) (++j) %= n;
      if(j == k) (++k) %= n;
      cur = area(p[i], p[j], p[k]);
   } while(i);
}
```

3.19 MinAreaTri

```
// 无重点、三点共线
// O(n^2log_2n)
struct P { int x, y, ind, u, v; };
namespace MinAreaTri {
   const int N = 2020;
   const ll inf = 4e18;
   int n, m, pos[N];
   P p[N], l[N * N];
   bool cmp(const P &x, const P &y) { return det(x, y) < 0; }
   void solve() {
      sort(p + 1, p + 1 + n);
      rep(i, 1, n + 1) p[i].ind = i, pos[i] = i;
      m = 0; rep(i, 1, n + 1) rep(j, i + 1, n + 1) {
         l[++m] = p[i] - p[j];
         if(l[m].x < 0) l[m].x *= -1, l[m].y *= -1;
         else if(l[m].x == 0 && l[m].y < 0) l[m].y *= -1;
         l[m].u = i, l[m].v = j;
      }
      sort(l + 1, l + 1 + m, cmp);
      mi = inf, ma = 0;
      rep(i, 1, m + 1) {
         int u = l[i].u, v = l[i].v;
         int pu = pos[u], pv = pos[v];
         if(pu > pv) swap(u, v), swap(pu, pv);
         if(pu == 1 || pv == n) continue;
         mi = min(mi, area(p[pu - 1], p[pu], p[pv + 1], p[v]));
         ma = max(ma, area(p[1], p[pu], p[n], p[v]));
         swap(p[pu], p[pv]);
         swap(pos[u], pos[v]);
      }
      cout << mi << " " << ma << endl;
   }
}
```

3.20 凹四边形计数

```
const int N = 1010;
int n; P p[N], q[N]; ll s[N];
namespace CNT {
   bool gao(P a) { return a.y > 0 || (a.y == 0 && a.x >= 0); }
   bool cmp(P a, P b) {
      bool o = gao(a), t = gao(b);
      if(o != t) return o > t;
      return det(a, b) > 0;
   }
   void solve(int u, ll &ans) {
      rep(i, 1, n + 1) q[i] = p[i]; swap(q[1], q[u]);
      rep(i, 2, n + 1) q[i] = q[i] - p[u];
      sort(q + 2, q + n + 1, cmp);
      int k = n; while(k >= 2 && q[k].y <= 0) --k;
      int j = k, cnt = 0;
      per(i, k + 1, n + 1) {
         while(j >= 2 && det(q[j], q[i]) > 0) --j, ++cnt;
         s[i] = s[i + 1] + cnt;
      }
      int c = j = k + 1;
      rep(i, 2, k + 1) {
         while(c <= n && det(q[i], q[c]) > 0) ++c;
         while(j <= n && det(q[i], q[j]) >= 0) ++j;
         ans += s[j] + (n - j + 1) * 1ll * (c - k - 1);
      }
   }
   ll solve() {
      ll ans = 0; rep(i, 1, n + 1) solve(i, ans);
      return ans;
   }
}
```

3.21 平面图转对偶图

```
struct Planar {
   static const int N = 101010, M = 101010;
   // ps id starts from 0
   vector<P> ps;
   // cnte id starts from 0
   int cnte, ne[M];
   bool vis[M];
   // u -> (v, cnte)
   vector<pii> g[N];
   pii E[M];
   vector<db> areas;

   void init() {
      rep(i, 0, sz(ps)) g[i].clear();
      fill_n(vis, cnte, false);
      ps.clear(); cnte = 0;
      areas.clear();
   }
   void adde(int u, int v) {
      g[u].pb(mp(v, cnte));
      E[cnte++] = mp(u, v);
      g[v].pb(mp(u, cnte));
      E[cnte++] = mp(v, u);
   }
   int V;
   bool cmp(const pii &i, const pii &j) {
      P a = ps[i.fi] - ps[V], b = ps[j.fi] - ps[V];
      int o = P(0, 0) < a, t = P(0, 0) < b;
      if(o != t) return o < t;
      return det(a, b) > 0;
   }
   void go(int e) {
      db res = 0;
      while(!vis[e]) {
         res += det(ps[E[e].se], ps[E[e].fi]); vis[e] = 1;
         e = ne[e ^ 1];
      }
      if(res > 0) areas.pb(res / 2);
   }
   void solve(const vector<P> &_ps, const vector<pii> &es) {
      init(); ps = _ps;
      for(auto e : es) adde(e.fi, e.se);
      rep(i, 0, sz(ps)) {
         V = i; sort(all(g[i]), cmp);
         rep(j, 0, sz(g[i])) {
            ne[g[i][j].se] = g[i][(j + 1) % sz(g[i])].se;
         }
      }
      rep(i, 0, cnte) if(!vis[i]) go(i);
   }
};
```

3.22 旋转卡壳

```
// 凸包都是顺时针给出
// 【凸包直径】点 - 点
T diameter(vector<P> ps) {
   n = sz(ps); T ans = 0;
   if(n <= 1) return 0;
   if(n == 2) return (ps[1] - ps[0]).len();
   rep(i, 0, n) {
      P t = ps[i] - ps[(i + 1) % n];
      while(det(t, ps[(p + 1) % n] - ps[p]) > 0) (++p) %= n;
      ans = max(ans, (ps[i] - ps[p]).len());
      ans = max(ans, (ps[(i + 1) % n] - ps[p]).len());
   }
   return ans;
}
// 【凸包宽度】点 - 边
// 【凸包间的最大距离】点 - 点
// 【凸包间的最小距离】
T solve(P p[], int n, P q[], int m) {
   int o = 0, t = 0; T ans = inf;
   rep(i, 1, n) if(p[i].y > p[o].y) o = i;
   rep(i, 1, m) if(q[i].y < q[t].y) t = i;
   rep(i, 0, n) {
      P a = p[(o + 1) % n] - p[o]; db tmp;
      while((tmp = det(a, q[(t + 1) % m] - q[t])) < 0) (++t) %= m;
      if(sign(tmp)) ans = min(ans, disToSeg(L(p[o], p[(o + 1) % n]), q[t]));
      else ans = min(ans, disSS(L(p[o], p[(o + 1) % n]), L(q[t], q[(t + 1) % m])));
      (++o) %= n;
   }
   return ans;
}
T work(P p[], int n, P q[], int m) {
   return min(solve(p, n, q, m), solve(q, m, p, n));
}
// 【凸包最小面积外接矩形】
T solve(vector<P> ps) {
   int n = sz(ps); T ans = 1e18;
   int p = 1, l = 1, r;
   rep(i, 0, n) {
      P t = ps[i] - ps[(i + 1) % n];
      while(det(t, ps[(p + 1) % n] - ps[p]) > 0) (++p) %= n;
      while(dot(t, ps[(l + 1) % n] - ps[l]) < 0) (++l) %= n;
      r = (p + 1) % n;
      while(dot(t, ps[(r + 1) % n] - ps[r]) > 0) (++r) %= n;
      ll et = abs(det(ps[p], ps[i], ps[(i + 1) % n]));
      ll ot = abs(dot(t, ps[l] - ps[r]));
      ans = min(ans, (db)et * ot / t.len2());
   }
   return ans;
}
// 【凸包最小周长外接矩形】
```

4 Graph

4.1 矩阵 -树定理（生成树计数）

```
/*生成树计数问题
n 个节点的无向图 G ，求一个包含 n-1 条边的边集使得边集的边构成一颗树，问这样的边集的数量度数矩阵
 D n*的矩阵n Di,i=的度数i Di,j=0(i!=j)邻接矩阵
 A n*的矩阵n Ai,j=(与有边相连ij)?1:0基尔霍夫矩阵
 M M=D-A Mi,i=的度数i Mi,j=(与有边相连ij)?-1:0(i!=j)矩阵树定理对于图，它的基尔霍夫矩阵的每个
 代数余子式相等，且等于的生成树的数目。
 GMG*/
```

5 Java

5.1 IO

```
package mytest;
//提交评测前删除package
import java.io.*;
import java.util.*;
import java.math.*;

public class Main {
    BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));
    PrintWriter writer = new PrintWriter(System.out);
    StringTokenizer tokenizer = null;
    void solve()throws Exception{
      List<String> mylist1 = new ArrayList<>();
      List<String> mylist2 = new LinkedList<>();
      List<String> mylist3 = new Vector<>();
      Vector<String> vec = new Vector<>();
      Queue<String> que = new LinkedList<>();
      Stack<String> sta = new Stack<>();
      Set<String> myset = new HashSet<>();
      Set<String> myset2 = new TreeSet<>();
      Map<String, Integer> mymap = new HashMap<>();
      Map<String, Integer> mymap2 = new TreeMap<>();
    }
    void run()throws Exception{
      try {
         while (true) {
            solve();
         }
      }catch(Exception e) {}
      finally {
         reader.close();
         writer.close();
      }
    }
    String next()throws Exception{
      for(;tokenizer == null || !tokenizer.hasMoreTokens();){
         tokenizer = new StringTokenizer(reader.readLine());
      }
    return tokenizer.nextToken();
  }
  int nextInt()throws Exception{
    return Integer.parseInt(next());
  }
  double nextDouble()throws Exception{
    return Double.parseDouble(next());
  }
  BigInteger nextBigInteger()throws Exception{
    return new BigInteger(next());
  }
  public static void main(String args[])throws Exception{
    (new Main()).run();
  }
}
```

6 Math

6.1 FFT

```
const double PI = acos(-1.0);
const int _M = N, _N = N;
template <class V>
struct FT {
   struct cp { double x, y; } tmp[_M * 2 + 5];
   friend cp operator + (cp &a, cp &b) { return cp{ a.x + b.x,a.y + b.y }; }
   friend cp operator - (cp &a, cp &b) { return cp{ a.x - b.x,a.y - b.y }; }
   friend cp operator * (cp &a, cp &b) { return cp{ a.x*b.x - a.y*b.y,a.x*b.y + a.y*b.x
   }; }
   cp get(double x) { return cp{ cos(x),sin(x) }; }
   vector <cp> aa, bb;
   void FFT(vector<cp> &a, int n, int op) {
      for (int i = (n >> 1), j = 1; j < n; j++) {
         if (i < j) swap(a[i], a[j]);
         int k; for (k = (n >> 1); k&i; i ^= k, k >>= 1); i ^= k;
      }
      for (int m = 2; m <= n; m <<= 1) {
         cp w = get(2 * PI*op / m); tmp[0] = cp{ 1,0 };
         for (int j = 1; j < (m >> 1); j++) tmp[j] = tmp[j - 1] * w;
         for (int i = 0; i < n; i += m)
            for (int j = i; j < i + (m >> 1); j++) {
               cp u = a[j], v = a[j + (m >> 1)] * tmp[j - i];
               a[j] = u + v, a[j + (m >> 1)] = u - v;
            }
      }
      if (op == -1) rep(i, 0, n) a[i] = cp{ a[i].x / n,a[i].y / n };
   }
   vector<V> multiply(vector<V> A, vector<V> B, int op = 0) {
      if (op) reverse(all(A));
      int lena = A.size(), lenb = B.size(), len = 1;
      while (len < lena + lenb) len <<= 1;
      aa = vector<cp>(len), bb = vector<cp>(len);
      rep(i, 0, lena) aa[i] = cp{ (double)A[i],0 };
      rep(i, 0, lenb) bb[i] = cp{ (double)B[i],0 };
      FFT(aa, len, 1), FFT(bb, len, 1);
      rep(i, 0, len) aa[i] = aa[i] * bb[i];
      FFT(aa, len, -1); A.clear();
      if (!op) rep(i, 0, len) A.pb((ll)(aa[i].x + 0.5)); else
         rep(i, lena - 1, lena + lenb - 2 + 1) A.pb((ll)(aa[i].x + 0.5));
      return A;
   }
};
FT<ll> fft;
```

6.2 NTT

```
const int M = 1 << 17 << 1;
int a[M], b[M];

struct NTT{
   static const int G = 3, P = 1004535809; //P = C*2^k + 1
   int N, na, nb, w[2][M], rev[M];
   ll kpow(ll a, int b){
      ll c = 1;
      for (; b; b >>= 1,a = a * a % P) if (b & 1) c = c * a %P;
      return c;
   }
   void FFT(int *a, int f){
      rep(i, 0, N) if (i < rev[i]) swap(a[i], a[rev[i]]);
      for (int i = 1; i < N; i <<= 1)
         for (int j = 0, t = N / (i << 1); j < N; j += i << 1)
            for (int k = 0, l = 0, x, y; k < i; k++, l += t)
               x = (ll) w[f][l] * a[j+k+i] % P, y = a[j+k], a[j+k] = (y+x) % P, a[j+k+i]
                = (y-x+P) % P;
      if (f) for (int i = 0, x = kpow(N, P-2); i < N; i++) a[i] = (ll)a[i] * x % P;
   }
   void work(){
      int d = __builtin_ctz(N);
      w[0][0] = w[1][0] = 1;
      for (int i = 1, x = kpow(G, (P-1) / N), y = kpow(x, P-2); i < N; i++) {
         rev[i] = (rev[i>>1] >> 1) | ((i&1) << (d-1));
         w[0][i] = (ll)x * w[0][i-1] % P, w[1][i] = (ll) y * w[1][i-1] % P;
      }
   }
   void doit(int *a, int *b, int na, int nb){ // [0, na)
      for (N = 1; N < na + nb - 1; N <<= 1);
      rep(i, na, N) a[i] = 0;
      rep(i, nb, N) b[i] = 0;
      work(), FFT(a,0), FFT(b,0);
      rep(i, 0, N) a[i] = (ll)a[i] * b[i] % P;
      FFT(a, 1);
      //rep(i, 0, N) cout << a[i] << endl;
   }
} ntt;
```

6.3 欧拉函数

```
/*欧拉公式
```

```
Euler(n)=n/(1–1/p1)(1–1/p2)…()求
[1,n中与–1]互质的数的个数n
*/

//求单个欧拉函数值
int euler(int n){
   int ans = 1,i;
   for (i = 2; i * i <= n; i++){
      if (n % i == 0){
         n /= i;
         ans *= i – 1;
         while (n % i == 0){
            n /= i;
            ans *= i;
         }
      }
   }
   if (n > 1) ans *= n – 1;
   return ans;
}
//筛素数法求1..欧拉函数值n
```

6.4 线性筛素数

```
void sift_prime(bool notprime[],int N){
   vector<int> prime;
   memset(notprime,false,sizeof(bool)*N);
   notprime[0] = notprime[1] = 1;
   for (int i = 2; i < N; ++i){
      if (!notprime[i]) prime.push_back(i);
      for (int j = 0; i * prime[j] <= N && j < prime.size(); ++j){
         notprime[i * prime[j]] = 1;
         if (i % prime[j] == 0) break; //speed up linear time
      }
   }
}
```

6.5 高斯消元

```
int guass(int n){
   int ans=1,t,tmp;
   for (int i=0; i<n; i++){
      for (int j=i+1; j<n; j++){
         while (mat[j][i]){
            t=mat[i][i]/mat[j][i];
            for (int k=0; k<n; k++){
               tmp=mat[i][k];
               tmp–=t*mat[j][k];
               mat[i][k]=tmp;
            }
            for (int k=0; k<n; k++) swap(mat[i][k],mat[j][k]);
            ans=–ans;
         }
      }
         if (mat[i][i]==0) return 0;
         ans*=mat[i][i];
   }
   return ans;
}
```