Contents

计算几何

二维几何基础

```
1  struct point {
2    double x, y;
3    point(double x = 0, double y = 0) : x(x), y(y) {}
4    inline double length() const { return sqrt(x * x + y * y); }
5    inline double norm() const { return length(); }
6    inline double norm2() const { return x * x + y * y; }
7    inline point unit() const; // unitize
8    inline point negate() const { return point(-x, -y); }
9    inline point rot90() const { return point(-y, x); }  // counter - clockwise
10   inline point _rot90() const { return point(y, -x); }  // clockwise
11   inline point rotate(double theta) const {  // counter - clockwise
12     double c = cos(theta), s = sin(theta);
13     return point(x * c - y * s, x * s + y * c);
14   }
15 };
16 // Basic 2D operators (e.g. +, -, *, /) has been removed.
17 struct line {
18   point s, t;
19   line(point s = point(), point t = point()) : s(s), t(t) {}
20   inline double length() const { return dis(s, t); }
21 };
22 //线段交点
23 //注意如果两条线段是共线的且有交点，那么 intersect_judgement 确实会返回 true,
24 //但是 line_intersect 会求错，所以这种情况需要特判.
25 inline bool point_on_line(const point &a, const line &b) {
26   return sign(det(a - b.s, b.t - b.s)) == 0 && dot(b.s - a, b.t - a) < EPS;
27 }
28 inline bool two_side(const point &a, const point &b, const line &c) {
29   return sign(det(a - c.s, c.t - c.s)) * sign(det(b - c.s, c.t - c.s)) < 0;
30 }
31 inline bool intersect_judgement(const line &a, const line &b) {
32   if (point_on_line(b.s, a) || point_on_line(b.t, a)) return true;
33   if (point_on_line(a.s, b) || point_on_line(a.t, b)) return true;
34   return two_side(a.s, a.t, b) && two_side(b.s, b.t, a);
35 }
36 inline point line_intersect(const line &a, const line &b) {
```

```
  double s1 = det(a.t - a.s, b.s - a.s);
  double s2 = det(a.t - a.s, b.t - a.s);
  return (b.s * s2 - b.t * s1) / (s2 - s1);
}
//点到直线的距离
double point_to_line(const point &p, const line &l) {
  return fabs(det(l.t - l.s, p - l.s)) / dis(l.s, l.t);
}
inline double min_point_to_line(const point &a, const line &b) {
  if (dot(b.s - a, b.t - a) < EPS) return fabs(det(b.s - a, b.t - a) /
 ↪  b.length());
  return min(dis(a, b.s), dis(a, b.t));
}
//点在多边形内
bool in_polygon(const point &p, const vector<point> &poly) {
  int n = (int)poly.size();
  int counter = 0;
  for (int i = 0; i < n; ++i) {
    point a = poly[i], b = poly[(i + 1) % n];
    if (point_on_line(p, line(a, b))) return false;  // bounded excluded
    int x = sign(det(p - a, b - a));
    int y = sign(a.y - p.y);
    int z = sign(b.y - p.y);
    if (x > 0 && y <= 0 && z > 0) counter++;
    if (x < 0 && z <= 0 && y > 0) counter--;
  }
  return counter != 0;
}
//点到直线的投影
point project_to_line(const point &p, const line &l) {
  return l.s + (l.t - l.s) * (dot(p - l.s, l.t - l.s) / (l.t - l.s).norm2());
}
//圆类
struct circle {
  point center;
  double radius;
  circle(point center = point(), double radius = 0)
      : center(center), radius(radius) {}
};
inline bool operator==(const circle &a, const circle &b) {
  return a.center == b.center && fabs(a.radius - b.radius) < EPS;
}
inline bool operator!=(const circle &a, const circle &b) {
  return a.center != b.center || fabs(a.radius - b.radius) > EPS;
}
inline bool in_circle(const point &p, const circle &c) {
  return dis(p, c.center) < c.radius + EPS;
}
//圆的生成函数
circle make_circle(const point &a, const point &b) {
  return circle((a + b) / 2, dis(a, b) / 2);
}
circle make_circle(const point &a, const point &b, const point &c) {
  point center = circumcenter(a, b, c);
  return circle(center, dis(center, a));
}
//点到圆的切线
pair<line, line> tangent(const point &p, const circle &c) {
  circle a = make_circle(p, c.center);
  return make_pair(circle_intersect(a, c), circle_intersect(c, a));
}
//直线与圆的交点
```

```
//返回 AB 方向的第一个交点
point line_circle_intersect(const line &l, const circle &c) {
  double x = sqrt(sqr(c.radius) - sqr(point_to_line(c.center, l)));
  return project_to_line(c.center, l) + (l.s - l.t).unit() * x;
}
//圆与圆的交点
point circle_intersect(const circle &a, const circle &b) {  // get another point
  using circle_intersect(b, a) point r = (b.center - a.center).unit();
  double d = dis(a.center, b.center);
  double x = .5 * ((sqr(a.radius) - sqr(b.radius)) / d + d);
  double h = sqrt(sqr(a.radius) - sqr(x));
  return a.center + r * x + r.rot90() * h;
}
```

快速凸包

```
//水平序凸包
inline bool turn_left(const point &a, const point &b, const point &c) {
  return det(b - a, c - a) > EPS;
}
inline bool turn_right(const point &a, const point &b, const point &c) {
  return det(b - a, c - a) < -EPS;
}
inline vector<point> convex_hull(vector<point> a) {
  int n = (int)a.size(), cnt = 0;
  sort(a.begin(), a.end());
  vector<point> ret;
  for (int i = 0; i < n; ++i) {
    while (cnt > 1 && turn_left(ret[cnt - 2], a[i], ret[cnt - 1])) {
      --cnt;
      ret.pop_back();
    }
    ret.push_back(a[i]);
    ++cnt;
  }
  int fixed = cnt;
  for (int i = n - 1; i >= 0; --i) {
    while (cnt > fixed && turn_left(ret[cnt - 2], a[i], ret[cnt - 1])) {
      --cnt;
      ret.pop_back();
    }
    ret.push_back(a[i]);
    ++cnt;
  }
  // this algorithm will preserve the points which are collineation
  // the lowest point will occur twice , i.e. ret.front () == ret.back ()
  return ret;
}
```

半平面交

```
//半平面交
inline bool two_side(const point &a, const point &b, const line &c) {
  return sign(det(a - c.s, c.t - c.s)) * sign(det(b - c.s, c.t - c.s)) < 0;
}
vector<point> cut(const vector<point> &c, line p) {
  vector<point> ret;
  if (c.empty()) return ret;
  for (int i = 0; i < (int)c.size(); ++i) {
    int j = (i + 1) % (int)c.size();
    if (!turn_right(p.s, p.t, c[i])) ret.push_back(c[i]);
    if (two_side(c[i], c[j], p)) ret.push_back(line_intersubsection(p,
 ↪  line(c[i], c[j])));
```

```
  }
  return ret;
}
static const double BOUND = 1e5;
/*
convex .clear ();
convex . push_back ( point (-BOUND , -BOUND ));
convex . push_back ( point (BOUND , -BOUND ));
convex . push_back ( point (BOUND , -BOUND ));
convex . push_back ( point (BOUND , -BOUND ));
convex = cut(convex , line(point , point));
Judgement : convex . empty ();
*/
//高效半平面交
// plane[] 按照法向量 (逆时针 90 度) 极角排序, 去除平行半平面
inline bool turn_left(const line &l, const point &p) {
  return turn_left(l.s, l.t, p);
}
vector<line> half_plane_intersect(const vector<line> &h) {
  int fore = 0, rear = -1;
  vector<line> ret;
  for (int i = 0; i < (int)h.size(); ++i) {
    while (fore < rear && !turn_left(h[i], line_intersect(ret[rear - 1],
 ↪  ret[rear]))) --rear;
    while (fore < rear && !turn_left(h[i], line_intersect(ret[fore], ret[fore +
 ↪  1]))) ++fore;
    ++rear;
    ret.push_back(h[i]);
  }
  while (rear - fore > 1 && !turn_left(ret[fore], line_intersect(ret[rear - 1],
 ↪  ret[rear]))) --rear;
  while (rear - fore > 1 && !turn_left(ret[rear], line_intersect(ret[fore],
 ↪  ret[fore + 1]))) ++fore;
  if (rear - fore < 2) return vector<line>();
  return ret;
}
```

三角形的心

```
//三角形的内心
point incenter(const point &a, const point &b, const point &c) {
  double p = (a - b).length() + (b - c).length() + (c - a).length();
  return (a * (b - c).length() + b * (c - a).length() + c * (a - b).length()) /
        p;
}
//三角形的外心
point circumcenter(const point &a, const point &b, const point &c) {
  point p = b - a, q = c - a, s(dot(p, p) / 2, dot(q, q) / 2);
  double d = det(p, q);
  return a + point(det(s, point(p.y, q.y)), det(point(p.x, q.x), s)) / d;
}
//三角形的垂心
point orthocenter(const point &a, const point &b, const point &c) {
  return a + b + c - circumcenter(a, b, c) * 2.0;
}
```

圆与多边形面积交

```
double getSectorArea(const Point &a, const Point &b, const double &r) {
 ↪  // 求扇形面积
  double c = (2.0 * r * r - sqrdist(a, b)) / (2.0 * r * r);
  double alpha = acos(c);
  return r * r * alpha / 2.0;
}
std::pair<double, double> getSolution(const double &a, const double &b, const
 ↪  double &c) {
  double delta = b * b - 4.0 * a * c; // 求二次方程 ax^2 + bx + c = 0 的解
  if (dcmp(delta) < 0) return std::make_pair(0, 0);
  else return std::make_pair((-b - sqrt(delta)) / (2.0 * a), (-b + sqrt(delta)) /
 ↪  (2.0 * a));
}
std::pair<Point, Point> getIntersection(const Point &a, const Point &b, const
 ↪  double &r) {
  Point d = b - a; // 直线与圆的交点
  double A = dot(d, d);
  double B = 2.0 * dot(d, a);
  double C = dot(a, a) - r * r;
  std::pair<double, double> s = getSolution(A, B, C);
  return std::make_pair(a + d * s.first, a + d * s.second);
}
double getPointDist(const Point &a, const Point &b) { // 原点到线段 AB 的距离
  Point d = b - a;
  int sA = dcmp(dot(a, d)), sB = dcmp(dot(b, d));
  if (sA * sB <= 0) return det(a, b) / dist(a, b);
  else return std::min(dist(a), dist(b));
}
// a 和 b 和原点组成的三角形与半径为 r 的圆的交的面积
double getArea(const Point &a, const Point &b, const double &r) {
  double dA = dot(a, a), dB = dot(b, b), dC = getPointDist(a, b), ans = 0.0;
  if (dcmp(dA - r * r) <= 0 && dcmp(dB - r * r) <= 0) return det(a, b) / 2.0;
  Point tA = a / dist(a) * r;
  Point tB = b / dist(b) * r;
  if (dcmp(dC - r) > 0) return getSectorArea(tA, tB, r);
  std::pair<Point, Point> ret = getIntersection(a, b, r);
  if (dcmp(dA - r * r) > 0 && dcmp(dB - r * r) > 0) {
    ans += getSectorArea(tA, ret.first, r);
    ans += det(ret.first, ret.second) / 2.0;
    ans += getSectorArea(ret.second, tB, r);
    return ans;
  }
  if (dcmp(dA - r * r) > 0) return det(ret.first, b) / 2.0 + getSectorArea(tA,
 ↪  ret.first, r);
  else return det(a, ret.second) / 2.0 + getSectorArea(ret.second, tB, r);
}
double getArea(int n, Point *p, const Point &c, const double r)  {
  double ret = 0.0; // 求圆与多边形的交的主过程
  for (int i = 0; i < n; i++) {
    int sgn = dcmp(det(p[i] - c, p[(i + 1) % n] - c));
    if (sgn > 0) ret += getArea(p[i] - c, p[(i + 1) % n] - c, r);
    else ret -= getArea(p[(i + 1) % n] - c, p[i] - c, r);
  }
  return fabs(ret);
}
```

圆并求面积

注意事项: 复杂度 $\mathcal{O}(n^2 \log n)$

```
struct arc {
  double theta;
  int delta;
  point p;
  arc(){};
  arc(const double &theta, const point &p, int d) : theta(theta), p(p), delta(d)
 ↪  {}
};
```

```
vector<arc> vec;
vector<double> ans;
vector<point> center;
int cnt = 0;
inline bool operator<(const arc &a, const arc &b) {
  return a.theta + EPS < b.theta;
}
inline void psh(const double t1, const point p1, const double t2, const point p2) {
  if (t2 + EPS < t1) cnt++;
  vec.push_back(arc(t1, p1, 1));
  vec.push_back(arc(t2, p2, -1));
}
inline double cub(const double &x) { return x * x * x; }
inline void combine(int d, const double &area, const point &o) {
  if (sign(area) == 0) return;
  center[d] = (center[d] * ans[d] + o * area) * (1 / (ans[d] + area));
  ans[d] += area;
}
void area(vector<circle> &cir) {
  int n = cir.size();
  vector<bool> f;
  f.resize(n);
  vec.clear();
  cnt = 0;
  for (int i = 0; i < n; i++) {
    f[i] = true;
    for (int j = 0; j < n; j++)
      if (i != j) {
        if ((cir[i] == cir[j] && i < j) || (cir[i] != cir[j] && cir[i].radius < cir[j].radius + EPS && (cir[i].center - cir[j].center).length() < fabs(cir[i].radius - cir[j].radius) + EPS)) {
          f[i] = false;
          break;
        }
      }
  }
  int n1 = 0;
  for (int i = 0; i < n; i++) if (f[i]) cir[n1++] = cir[i];
  n = n1;
  ans.clear();
  center.clear();
  ans.resize(n + 1);
  center.resize(n + 1);
  point dvd;
  for (int i = 0; i < n; i++) {
    dvd = cir[i].center - point(cir[i].radius, 0);
    vec.clear();
    vec.push_back(arc(-PI, dvd, 1));
    cnt = 0;
    for (int j = 0; j < n; j++)
      if (j != i) {
        double d = (cir[j].center - cir[i].center).norm2();
        if (d < sqr(cir[j].radius - cir[i].radius) + EPS) {
          if (cir[i].radius + i * EPS < cir[j].radius + j * EPS)
            psh(-PI, dvd, PI, dvd);
        } else if (d + EPS < sqr(cir[j].radius + cir[i].radius)) {
          double lambda = 0.5 * (1 + (sqr(cir[i].radius) - sqr(cir[j].radius)) / d);
          point cp(cir[i].center + (cir[j].center - cir[i].center) * lambda);
          point nor((cir[j].center - cir[i].center)._rot90().unit() * (sqrt(sqr(cir[i].radius) - (cp - cir[i].center).norm2())));
          point frm(cp + nor);
          point to(cp - nor);
          psh(atan2((frm - cir[i].center).y, (frm - cir[i].center).x), frm, atan2((to - cir[i].center).y, (to - cir[i].center).x), to);
        }
      }
    sort(vec.begin() + 1, vec.end());
    vec.push_back(arc(PI, dvd, -1));
    for (int j = 0; j + 1 < vec.size(); j++) {
      cnt += vec[j].delta;
      double theta(vec[j + 1].theta - vec[j].theta);
      double area(sqr(cir[i].radius) * theta * 0.5);
      combine(cnt, area, cir[i].center + point(sin(vec[j + 1].theta) - sin(vec[j].theta), cos(vec[j].theta) - cos(vec[j + 1].theta)) * (1. / area / 3 * cub(cir[i].radius)));
      combine(cnt, -sqr(cir[i].radius) * sin(theta) * 0.5, (cir[i].center + vec[j].p + vec[j + 1].p) / 3.);
      combine(cnt, det(vec[j].p, vec[j + 1].p) * 0.5, (vec[j].p + vec[j + 1].p) / 3.);
    }
  }
}
```

最小覆盖圆

```
circle minimum_circle(vector<point> p) {
  circle ret;
  random_shuffle(p.begin(), p.end());
  for (int i = 0; i < (int)p.size(); ++i)
    if (!in_circle(p[i], ret)) {
      ret = circle(p[i], 0);
      for (int j = 0; j < i; ++j)
        if (!in_circle(p[j], ret)) {
          ret = make_circle(p[j], p[i]);
          for (int k = 0; k < j; ++k)
            if (!in_circle(p[k], ret)) ret = make_circle(p[i], p[j], p[k]);
        }
    }
  return ret;
}
```

三维几何基础

```
struct TPoint{
  double x, y, z;
  TPoint() {}
  TPoint(double x, double y, double z) : x(x), y(y), z(z) {}
  bool operator <(const TPoint &p)const {
    int dX = dcmp(x - p.x), dY = dcmp(y - p.y), dZ = dcmp(z - p.z);
    return dX < 0 || (dX == 0 && (dY < 0 || (dY == 0 && dZ < 0)));
  }
};
double sqrdist(const TPoint &a);
double sqrdist(const TPoint &a, const TPoint &b);
double dist(const TPoint &a);
double dist(const TPoint &a, const TPoint &b);
double dot(const TPoint &a, const TPoint &b);
TPoint det(const TPoint &a, const TPoint &b) {
  TPoint ret;
  ret.x = a.y * b.z - b.y * a.z;
  ret.y = a.z * b.x - b.z * a.x;
  ret.z = a.x * b.y - b.x * a.y;
  return ret;
}
```

```cpp
double detdot(const TPoint &a, const TPoint &b, const TPoint &c, const TPoint &d) {
  return dot(det(b - a, c - a), d - a);
}
```

三维凸包

```cpp
struct Triangle{ // Construction function removed.
  TPoint a, b, c;
  double getArea() {
    TPoint ret = det(b - a, c - a);
    return dist(ret) / 2.0;
  }
};
namespace Convex_Hull {
  struct Face{ // Construction function removed.
    int a, b, c;
    bool isOnConvex;
  };
  int nFace, left, right, whe[MAXN][MAXN];
  Face queue[MAXF], tmp[MAXF];
  bool isVisible(const std::vector<TPoint> &p, const Face &f, const TPoint &a) {
    return dcmp(detdot(p[f.a], p[f.b], p[f.c], a)) > 0;
  }
  bool init(std::vector<TPoint> &p) {
    bool check = false;
    for (int i = 1; i < (int)p.size(); i++) {
      if (dcmp(sqrdist(p[0], p[i]))) {
        std::swap(p[1], p[i]);
        check = true;
        break;
      }
    }
    if (!check) return false;
    check = false;
    for (int i = 2; i < (int)p.size(); i++) {
      if (dcmp(sqrdist(det(p[i] - p[0], p[1] - p[0])))) {
        std::swap(p[2], p[i]);
        check = true;
        break;
      }
    }
    if (!check) return false;
    check = false;
    for (int i = 3; i < (int)p.size(); i++) {
      if (dcmp(detdot(p[0], p[1], p[2], p[i]))) {
        std::swap(p[3], p[i]);
        check = true;
        break;
      }
    }
    if (!check) return false;
    for (int i = 0; i < (int)p.size(); i++)
      for (int j = 0; j < (int)p.size(); j++) {
        whe[i][j] = -1;
      }
    return true;
  }
  void pushface(const int &a, const int &b, const int &c) {
    nFace++;
    tmp[nFace] = Face(a, b, c);
    tmp[nFace].isOnConvex = true;
    whe[a][b] = nFace;
    whe[b][c] = nFace;
    whe[c][a] = nFace;
  }
  bool deal(const std::vector<TPoint> &p, const std::pair<int, int> &now, const TPoint &base) {
    int id = whe[now.second][now.first];
    if (!tmp[id].isOnConvex) return true;
    if (isVisible(p, tmp[id], base)) {
      queue[++right] = tmp[id];
      tmp[id].isOnConvex = false;
      return true;
    }
    return false;
  }
  std::vector<Triangle> getConvex(std::vector<TPoint> &p) {
    static std::vector<Triangle> ret;
    ret.clear();
    if (!init(p)) return ret;
    if (!isVisible(p, Face(0,1,2),p[3])) pushface(0,1,2); else pushface(0,2,1);
    if (!isVisible(p, Face(0,1,3),p[2])) pushface(0,1,3); else pushface(0,3,1);
    if (!isVisible(p, Face(0,2,3),p[1])) pushface(0,2,3); else pushface(0,3,2);
    if (!isVisible(p, Face(1,2,3),p[0])) pushface(1,2,3); else pushface(1,3,2);
    for (int a = 4; a < (int)p.size(); a++) {
      TPoint base = p[a];
      for (int i = 1; i <= nFace; i++) {
        if (tmp[i].isOnConvex && isVisible(p, tmp[i], base)) {
          left = 0, right = 0;
          queue[++right] = tmp[i];
          tmp[i].isOnConvex = false;
          while (left < right) {
            Face now = queue[++left];
            if (!deal(p,std::make_pair(now.a,now.b),base))
              pushface(now.a,now.b,a);
            if (!deal(p,std::make_pair(now.b,now.c),base))
              pushface(now.b,now.c,a);
            if (!deal(p,std::make_pair(now.c,now.a),base))
              pushface(now.c,now.a,a);
          }
          break;
        }
      }
    }
    for (int i = 1; i <= nFace; i++) {
      Face now = tmp[i];
      if (now.isOnConvex)ret.push_back(Triangle(p[now.a],p[now.b],p[now.c]));
    }
    return ret;
  }
};
// Usage
std::vector<TPoint> p;
std::vector<Triangle> answer;
answer = Convex_Hull::getConvex(p);
```

三维绕轴旋转

注意事项：以右手拇指为向量方向，逆时针绕轴（剩下四根手指方向）旋转 θ 角的右乘矩阵。

```cpp
Matrix getTrans(const double &a, const double &b, const double &c) {
    Matrix ret;
    ret.a[0][0] = 1; ret.a[0][1] = 0; ret.a[0][2] = 0; ret.a[0][3] = 0;
    ret.a[1][0] = 0; ret.a[1][1] = 1; ret.a[1][2] = 0; ret.a[1][3] = 0;
    ret.a[2][0] = 0; ret.a[2][1] = 0; ret.a[2][2] = 1; ret.a[2][3] = 0;
    ret.a[3][0] = a; ret.a[3][1] = b; ret.a[3][2] = c; ret.a[3][3] = 1;
```

```
    return ret;
}
Matrix getRotate(const double &a, const double &b, const double &c, const double &theta) {
    Matrix ret;
    ret.a[0][0] = a * a * (1 - cos(theta)) + cos(theta);
    ret.a[0][1] = a * b * (1 - cos(theta)) + c * sin(theta);
    ret.a[0][2] = a * c * (1 - cos(theta)) - b * sin(theta);
    ret.a[0][3] = 0;
    ret.a[1][0] = b * a * (1 - cos(theta)) - c * sin(theta);
    ret.a[1][1] = b * b * (1 - cos(theta)) + cos(theta);
    ret.a[1][2] = b * c * (1 - cos(theta)) + a * sin(theta);
    ret.a[1][3] = 0;
    ret.a[2][0] = c * a * (1 - cos(theta)) + b * sin(theta);
    ret.a[2][1] = c * b * (1 - cos(theta)) - a * sin(theta);
    ret.a[2][2] = c * c * (1 - cos(theta)) + cos(theta);
    ret.a[2][3] = 0;
    ret.a[3][0] = 0; ret.a[3][1] = 0; ret.a[3][2] = 0; ret.a[3][3] = 1;
    return ret;
}
Matrix getRotate(const double &ax, const double &ay, const double &az, const double &bx, const double &by, const double &bz, const double &theta) {
    double l = dist(Point(0, 0, 0), Point(bx, by, bz));
    Matrix ret = getTrans(-ax, -ay, -az);
    ret = ret * getRotate(bx / l, by / l, bz / l, theta);
    ret = ret * getTrans(ax, ay, az);
    return ret;
}
```

图论

点双连通分量

```
int n, m, x, y, ans1, ans2, tot1, tot2, flag, size, ind2, dfn[N], low[N], block[M], vis[N];
vector<int> a[N];
pair<int, int> stack[M];
void tarjan(int x, int p) { // 坚固的点双连通分量
  dfn[x] = low[x] = ++ind2;
  for (int i = 0; i < a[x].size(); ++i)
    if (dfn[x] > dfn[a[x][i]] && a[x][i] != p){
      stack[++size] = make_pair(x, a[x][i]);
      if (i == a[x].size() - 1 || a[x][i] != a[x][i + 1])
        if (!dfn[a[x][i]]){
          tarjan(a[x][i], x);
          low[x] = min(low[x], low[a[x][i]]);
          if (low[a[x][i]] >= dfn[x]){
            tot1 = tot2 = 0;
            ++flag;
            for (; ; ){
              if (block[stack[size].first] != flag) {
                ++tot1;
                block[stack[size].first] = flag;
              }
              if (block[stack[size].second] != flag) {
                ++tot1;
                block[stack[size].second] = flag;
              }
              if (stack[size].first == x && stack[size].second == a[x][i]) break;
              ++tot2; --size;
            }
            for (; stack[size].first == x && stack[size].second == a[x][i]; --size)
              ++tot2;
            if (tot2 < tot1) ans1 += tot2;
            if (tot2 > tot1) ans2 += tot2;
          }
        }
        else low[x] = min(low[x], dfn[a[x][i]]);
    }
}
int main(){
  for (; ; ){
    scanf("%d%d", &n, &m);
    if (n == 0 && m == 0) return 0;
    for (int i = 1; i <= n; ++i) {
      a[i].clear();
      dfn[i] = 0;
    }
    for (int i = 1; i <= m; ++i){
      scanf("%d%d",&x, &y);
      ++x, ++y;
      a[x].push_back(y);
      a[y].push_back(x);
    }
    for (int i = 1; i <= n; ++i)
      sort(a[i].begin(), a[i].end());
    ans1 = ans2 = ind2 = 0;
    for (int i = 1; i <= n; ++i)
      if (!dfn[i]) {
        size = 0;
        tarjan(i, 0);
      }
    printf("%d %d\n", ans1, ans2);
  }
  return 0;
}
```

Hopcoft-Karp 求最大匹配

```
int matchx[N], matchy[N], level[N];
bool dfs(int x) {
    for (int i = 0; i < (int)edge[x].size(); ++i) {
        int y = edge[x][i];
        int w = matchy[y];
        if (w == -1 || level[x] + 1 == level[w] && dfs(w)) {
            matchx[x] = y;
            matchy[y] = x;
            return true;
        }
    }
    level[x] = -1;
    return false;
}
int solve() {
    std::fill(matchx, matchx + n, -1);
    std::fill(matchy, matchy + m, -1);
    for (int answer = 0; ; ) {
        std::vector<int> queue;
        for (int i = 0; i < n; ++i) {
            if (matchx[i] == -1) {
                level[i] = 0;
                queue.push_back(i);
            } else level[i] = -1;
        }
        for (int head = 0; head < (int)queue.size(); ++head) {
            int x = queue[head];
```

```
            for (int i = 0; i < (int)edge[x].size(); ++i) {
                int y = edge[x][i];
                int w = matchy[y];
                if (w != -1 && level[w] < 0) {
                    level[w] = level[x] + 1;
                    queue.push_back(w);
                }
            }
        }
        int delta = 0;
        for (int i = 0; i < n; ++i) {
            if (matchx[i] == -1 && dfs(i)) delta++;
        }
        if (delta == 0) return answer;
        else answer += delta;
    }
}
```

KM 带权匹配

注意事项： 最小权完美匹配，复杂度为 $\mathcal{O}(|V|^3)$。

```
// w[nx][ny] means the weight of edges
// lx[nx], ly[ny], link[ny], visx[nx], visy[ny], slack[ny]
// If you want to find the minimum of this problem, just take the negatatives.
int DFS(int x) {
    visx[x] = 1;
    for (int y = 1;y <= ny;y ++){
        if (visy[y]) continue;
        int t = lx[x] + ly[y] - w[x][y];
        if (t == 0) {
            visy[y] = 1;
            if (link[y] == -1 || DFS(link[y])){
                link[y] = x;
                return 1;
            }
        }
        else slack[y] = min(slack[y], t);
    }
    return 0;
}
int KM(){
    int i,j;
    memset(link,-1,sizeof(link));
    memset(ly,0,sizeof(ly));
    for (i = 1; i <= nx; i++)
        for (j = 1, lx[i] = -inf; j <= ny; j++)
          lx[i] = max(lx[i],w[i][j]);
    for (int x = 1; x <= nx; x++){
        for (i = 1; i <= ny; i++) slack[i] = inf;
        while (true) {
            memset(visx, 0, sizeof(visx));
            memset(visy, 0, sizeof(visy));
            if (DFS(x)) break;
            int d = inf;
            for (i = 1; i <= ny;i++)
                if (!visy[i] && d > slack[i]) d = slack[i];
            for (i = 1; i <= nx; i++)
                if (visx[i]) lx[i] -= d;
            for (i = 1; i <= ny; i++)
                if (visy[i]) ly[i] += d;
                else slack[i] -= d;
        }
    }
    int res = 0;
    for (i = 1;i <= ny;i ++)
        if (link[i] > -1) res += w[link[i]][i];
    return res;
}
```

稀疏图最大流

注意事项： 适用于比较稀疏的一般图。

```
int Maxflow_Isap(int s,int t,int n) {
  std::fill(pre + 1, pre + n + 1, 0);
  std::fill(d + 1, d + n + 1, 0);
  std::fill(gap + 1, gap + n + 1, 0);
  for (int i = 1; i <= n; i++) cur[i] = h[i];
  gap[0] = n;
  int u = pre[s] = s, v, maxflow = 0;
  while (d[s] < n) {
    v = n + 1;
    for (int i = cur[u]; i; i = e[i].next)
    if (e[i].flow && d[u] == d[e[i].node] + 1) {
      v = e[i].node; cur[u]=i; break;
    }
    if (v <= n) {
      pre[v] = u; u = v;
      if (v == t) {
        int dflow = INF, p = t; u = s;
        while (p != s) {
          p = pre[p];
          dflow = std::min(dflow, e[cur[p]].flow);
        }
        maxflow += dflow; p = t;
        while (p != s) {
          p = pre[p];
          e[cur[p]].flow -= dflow;
          e[e[cur[p]].opp].flow += dflow;
        }
      }
    }
    else{
      int mindist = n + 1;
      for (int i = h[u]; i; i = e[i].next)
        if (e[i].flow && mindist > d[e[i].node]) {
          mindist = d[e[i].node]; cur[u] = i;
        }
      if (!--gap[d[u]]) return maxflow;
      gap[d[u] = mindist + 1]++; u = pre[u];
    }
  }
  return maxflow;
}
```

稠密图最大流

注意事项： 适用于二分图以及一些比较稠密的、增广路径比较短的图。

```
bool BFS() {
  int h = 0, t = 1;
  for (int i = 1; i <= n; i ++) d[i] = 0;
  d[que[1] = S] = 1;
  while (h != t) {
    int cur = que[++h];
    for (int p = head[cur]; p != 0; p = pre[p]) {
      if (len[p] == 0 || d[other[p]] != 0) continue;
      d[other[p]] = d[cur] + 1;
```

```
      if (other[p] == n) return 1;
      que[++t] = other[p];
    }
  }
  return 0;
}
int dinic(int x, int flow) {
  if (x == n) return flow;
  int tmp = flow;
  for (int p = last[x]; p != 0; p = pre[p]) {
    if (len[p] == 0 || d[other[p]] != d[x] + 1) continue;
    int res = dinic(other[p], min(tmp, len[p]));
    len[p] -= res; len[p ^ 1] += res;
    if (len[p]) last[x] = p;
    tmp -= res;
  }
  if (flow - tmp == 0) d[x] = 0;
  return flow - tmp;
}
for (int i = 1; i <= n; i ++) head[i] = 0; // Remember to init.
```

稠密图费用流

```
int S, T, totFlow, totCost, dis[N], slack[N], visit[N];
int modlable () {
    int delta = INF;
    for (int i = 1; i <= T; i++) {
        if (!visit[i] && slack[i] < delta) delta = slack[i];
        slack[i] = INF;
    }
    if (delta == INF) return 1;
    for (int i = 1; i <= T; i++)
        if (visit[i]) dis[i] += delta;
    return 0;
}
int dfs (int x, int flow) {
    if (x == T) {
        totFlow += flow;
        totCost += flow * (dis[S] - dis[T]);
        return flow;
    }
    visit[x] = 1;
    int left = flow;
    for (int i = e.last[x]; ~i; i = e.succ[i])
        if (e.cap[i] > 0 && !visit[e.other[i]]) {
            if (dis[e.other[i]] + e.cost[i] == dis[x]) {
                int delta = dfs(e.other[i], min (left, e.cap[i]));
                e.cap[i] -= delta; e.cap[i ^ 1] += delta;
                left -= delta;
                if (!left) { visit[x] = 0; return flow; }
            } else slack[e.other[i]] = min(slack[e.other[i]], dis[e.other[i]] +
 ↪  e.cost[i] - dis[x]);
        }
    return flow - left;
}
pair <int, int> minCost () {
    totFlow = 0; totCost = 0;
    fill (dis + 1, dis + T + 1, 0);
    do {
        do { fill (visit + 1, visit + T + 1, 0); }
        while (dfs (S, INF));
    } while (!modlable ());
    return make_pair (totFlow, totCost);
}
```

2-SAT 问题

```
int stamp, comps, top;
int dfn[N], low[N], comp[N], stack[N];
void add(int x, int a, int y, int b) {
    edge[x << 1 | a].push_back(y << 1 | b);
}
void tarjan(int x) {
    dfn[x] = low[x] = ++stamp;
    stack[top++] = x;
    for (int i = 0; i < (int)edge[x].size(); ++i) {
        int y = edge[x][i];
        if (!dfn[y]) {
            tarjan(y);
            low[x] = std::min(low[x], low[y]);
        } else if (!comp[y]) {
            low[x] = std::min(low[x], dfn[y]);
        }
    }
    if (low[x] == dfn[x]) {
        comps++;
        do {
            int y = stack[--top];
            comp[y] = comps;
        } while (stack[top] != x);
    }
}
bool solve() {
    int counter = n + n + 1;
    stamp = top = comps = 0;
    std::fill(dfn, dfn + counter, 0);
    std::fill(comp, comp + counter, 0);
    for (int i = 0; i < counter; ++i) {
        if (!dfn[i]) tarjan(i);
    }
    for (int i = 0; i < n; ++i) {
        if (comp[i << 1] == comp[i << 1 | 1]) return false;
        answer[i] = (comp[i << 1 | 1] < comp[i << 1]);
    }
    return true;
}
```

有根树的同构

```
const unsigned long long MAGIC = 4423;
unsigned long long magic[N];
std::pair<unsigned long long, int> hash[N];
void solve(int root) {
    magic[0] = 1;
    for (int i = 1; i <= n; ++i) {
        magic[i] = magic[i - 1] * MAGIC;
    }
    std::vector<int> queue;
    queue.push_back(root);
    for (int head = 0; head < (int)queue.size(); ++head) {
        int x = queue[head];
        for (int i = 0; i < (int)son[x].size(); ++i) {
            int y = son[x][i];
            queue.push_back(y);
        }
    }
    for (int index = n - 1; index >= 0; --index) {
        int x = queue[index];
```

```
        hash[x] = std::make_pair(0, 0);
        std::vector<std::pair<unsigned long long, int> > value;
        for (int i = 0; i < (int)son[x].size(); ++i) {
            int y = son[x][i];
            value.push_back(hash[y]);
        }
        std::sort(value.begin(), value.end());
        hash[x].first = hash[x].first * magic[1] + 37;
        hash[x].second++;
        for (int i = 0; i < (int)value.size(); ++i) {
            hash[x].first = hash[x].first * magic[value[i].second] +
↪   value[i].first;
            hash[x].second += value[i].second;
        }
        hash[x].first = hash[x].first * magic[1] + 41;
        hash[x].second++;
    }
}
```

Dominator Tree

```
#define foreach(A, x, it) for (int it = A.h[x]; it; it = A.e[it].next)
const int MAXN = 200001;
const int MAXM = 400001;
template<class T, int MAXN, int MAXM>
struct AdjList{
  struct Edge{
    T data;
    int next;
  }e[MAXM];
  int h[MAXN], t;
  void add(int x, const T &data) {
    t++; e[t] = (Edge){data, h[x]}; h[x] = t;
  }
  void drop(int x) {
    h[x] = e[h[x]].next;
  }
  T & operator [](const int &index) {
    return e[index].data;
  }
  void clear(int n) {
    std::fill(h + 1, h + n + 1, t = 0);
  }
};
// fa 是并查集的父亲，f 是树的父亲，sdom 是半必经点，idom 是必经点
// smin 是带权并查集的权值，pred 是前驱链表，succ 是后继链表
int dfn[MAXN], sdom[MAXN], idom[MAXN], id[MAXN], f[MAXN], fa[MAXN], smin[MAXN];
AdjList<int, MAXN, MAXM> pred, succ;
long long answer[MAXN];
void predfs(int x) {
  id[dfn[x] = ++stamp] = x;
  foreach(succ, x, i) {
    int y = succ[i];
    if (!dfn[y]) {
      f[y] = x;
      predfs(y);
    }
  }
}
int getfa(int x) {
  if (fa[x] == x) return x;
  int ret = getfa(fa[x]);
  if (dfn[sdom[smin[fa[x]]]] < dfn[sdom[smin[x]]]) {
    smin[x] = smin[fa[x]];
  }
  return fa[x] = ret;
}
void tarjan(int s) {
  static AdjList<int, MAXN, MAXN> tmp;
  stamp = tmp.t = 0;
  predfs(s);
  for (int i = 1; i <= stamp; i++) {
    fa[id[i]] = smin[id[i]] = id[i];
    tmp.h[id[i]] = idom[id[i]] = 0;
  }
  for (int o = stamp; o >= 1; o--) {
    int x = id[o];
    if (o != 1) {
      sdom[x] = f[x];
      foreach(pred, x, i) {
        int p = pred[i];
        if (!dfn[p]) continue;
        if (dfn[p] > dfn[x]) {
          getfa(p);
          p = sdom[smin[p]];
        }
        if (dfn[sdom[x]] > dfn[p]) sdom[x] = p;
      }
      tmp.add(sdom[x], x);
    }
    while (tmp.h[x] != 0) {
      int y = tmp[tmp.h[x]];
      tmp.drop(x);
      getfa(y);
      if (x != sdom[smin[y]]) {
        idom[y] = smin[y];
      } else {
        idom[y] = x;
      }
    }
    foreach(succ, x, i) {
      if (f[succ[i]] == x) {
        fa[succ[i]] = x;
      }
    }
  }
  idom[s] = s;
  for (int i = 2; i <= stamp; i++) {
    int x = id[i];
    if (idom[x] != sdom[x]) {
      idom[x] = idom[idom[x]];
    }
  }
}
```

无向图最小割

```
int node[N], dist[N];
bool visit[N];
int solve(int n) {
  int answer = INT_MAX;
  for (int i = 0; i < n; ++i) node[i] = i;
  while (n > 1) {
    int max = 1;
    for (int i = 0; i < n; ++i) {
      dist[node[i]] = graph[node[0]][node[i]];
```

```
      if (dist[node[i]] > dist[node[max]]) max = i;
    }
    int prev = 0;
    memset(visit, 0, sizeof(visit));
    visit[node[0]] = true;
    for (int i = 1; i < n; ++i) {
      if (i == n - 1) {
        answer = std::min(answer, dist[node[max]]);
        for (int k = 0; k < n; ++k) {
          graph[node[k]][node[prev]] = (graph[node[prev]][node[k]] +=
↪  graph[node[k]][node[max]]);
        }
        node[max] = node[--n];
      }
      visit[node[max]] = true;
      prev = max; max = -1;
      for (int j = 1; j < n; ++j) {
        if (!visit[node[j]]) {
          dist[node[j]] += graph[node[prev]][node[j]];
          if (max == -1 || dist[node[max]] < dist[node[j]]) {
            max = j;
          }
        }
      }
    }
  }
  return answer;
}
```

最大团搜索

```
// mc[i] 代表只用 i-n 号点的答案
// g 代表连通性
void dfs(int size) {
  int i, j, k;
  if (len[size] == 0) {
    if (size > ans) {
      ans = size;
      found = true;
    }
    return;
  }
  for (k = 0; k < len[size] && !found; k ++) {
    if (size + len[size] - k <= ans) break;
    i = list[size][k];
    if (size + mc[i] <= ans) break;
    for (j = k + 1, len[size + 1] = 0; j < len[size]; j ++)
      if (g[i][list[size][j]]) list[size + 1][len[size + 1] ++] = list[size][j];
    dfs(size + 1);
    if (found) {
      prin[size + 1] = i;
    }
  }
}
void work() {
  int i, j;
  mc[n] = ans = 1;
  ansi = 1;
  for (i = n - 1; i; i --) {
    found = false;
    len[1] = 0;
    for (j = i + 1; j <= n; j ++) if (g[i][j]) list[1][len[1]++] = j;
    dfs(1);
    mc[i] = ans;
    if (found) prin[1] = i;
  }
}
void print() {
  printf("%d\n", ans);
  for (int i = 1; i < ans; i ++) printf("%d ", prin[i]);
  printf("%d\n", prin[ans]);
}
```

极大团计数

```
bool g[N][N];
int ne[N], ce[N], list[N][N], ans;
void dfs(int size) {
  if (ans > 1000) return;
  int i, j, k, t, cnt, best = 0;
  bool bb;
  if (ne[size] == ce[size]) {
    if (ce[size] == 0) ++ans;
    return;
  }
  for (t = 0, i = 1; i <= ne[size]; ++i) {
    for (cnt = 0, j = ne[size] + 1; j <= ce[size]; ++j)
      if (!g[list[size][i]][list[size][j]]) ++cnt;
    if (t == 0 || cnt < best) t = i, best = cnt;
  }
  if (t && best <= 0) return;
  for (k = ne[size] + 1; k <= ce[size]; ++k) {
    if (t > 0) {
      for (i = k; i <= ce[size]; ++i)
        if (!g[list[size][t]][list[size][i]]) break;
      swap(list[size][k], list[size][i]);
    }
    i = list[size][k];
    ne[size + 1] = ce[size + 1] = 0;
    for (j = 1; j < k; ++j)
      if (g[i][list[size][j]])
        list[size + 1][++ne[size + 1]] = list[size][j];
    for (ce[size + 1] = ne[size + 1], j = k + 1; j <= ce[size]; ++j)
      if (g[i][list[size][j]]) list[size + 1][++ce[size + 1]] = list[size][j];
    dfs(size + 1);
    ++ne[size];
    --best;
    for (j = k + 1, cnt = 0; j <= ce[size]; ++j)
      if (!g[i][list[size][j]]) ++cnt;
    if (t == 0 || cnt < best) t = k, best = cnt;
    if (t && best <= 0) break;
  }
}
int main(){
  int n, m;
  while (scanf("%d%d", &n, &m) == 2) {
    for (int i = 1; i <= n; ++i)
      for (int j = 1; j <= n; ++j)
        g[i][j] = false;
    while (m--) {
      int x, y;
      scanf("%d%d", &x, &y);
      g[x][y] = g[y][x] = true;
    }
    ne[0] = 0;
    ce[0] = 0;
    for (int i = 1; i <= n; ++i)
```

```
      list[0][++ce[0]] = i;
    ans = 0;
    dfs(0);
    if (ans > 1000) puts("Too many maximal sets of friends.");
    else printf("%d\n", ans);
  }
  return 0;
}
```

最小树形图

```
int n, m, used[N], pass[N], eg[N], more, queue[N];
double g[N][N];
void combine(int id, double &sum) {
  int tot = 0, from, i, j, k;
  for (; id != 0 && !pass[id]; id = eg[id]) {
    queue[tot++] = id;
    pass[id] = 1;
  }
  for (from = 0; from < tot && queue[from] != id; from++);
  if  (from == tot) return;
  more = 1;
  for (i = from; i < tot; i++) {
    sum += g[eg[queue[i]]][queue[i]];
    if (i != from) {
      used[queue[i]] = 1;
      for (j = 1; j <= n; j++) if (!used[j]) {
        if (g[queue[i]][j] < g[id][j]) g[id][j] = g[queue[i]][j];
      }
    }
  }
  for (i = 1; i <= n; i++) if (!used[i] && i != id) {
    for (j = from; j < tot; j++) {
      k = queue[j];
      if (g[i][id] > g[i][k] - g[eg[k]][k]) g[i][id] = g[i][k] - g[eg[k]][k];
    }
  }
}
double mdst(int root) {
  int i, j, k;
  double sum = 0;
  memset(used, 0, sizeof(used));
  for (more = 1; more; ) {
    more = 0;
    memset(eg, 0, sizeof(eg));
    for (i = 1; i <= n; i++) if (!used[i] && i != root) {
      for (j = 1, k = 0; j <= n; j++) if (!used[j] && i != j)
        if (k == 0 || g[j][i] < g[k][i]) k = j;
      eg[i] = k;
    }
    memset(pass, 0, sizeof(pass));
    for (i = 1; i <= n; i++) if (!used[i] && !pass[i] && i != root) combine(i,
 ↪  sum);
  }
  for (i = 1; i <= n; i++) if (!used[i] && i != root) sum += g[eg[i]][i];
  return sum;
}
```

带花树

```
int match[N], belong[N], next[N], mark[N], visit[N];
std::vector<int> queue;
int find(int x) {
    if (belong[x] != x) belong[x] = find(belong[x]);
    return belong[x];
}
void merge(int x, int y) {
    x = find(x); y = find(y);
    if (x != y) belong[x] = y;
}
int lca(int x, int y) {
    static int stamp = 0;
    stamp++;
    while (true) {
        if (x != -1) {
            x = find(x);
            if (visit[x] == stamp) return x;
            visit[x] = stamp;
            if (match[x] != -1) x = next[match[x]];
            else x = -1;
        }
        std::swap(x, y);
    }
}
void group(int a, int p) {
    while (a != p) {
        int b = match[a], c = next[b];
        if (find(c) != p) next[c] = b;
        if (mark[b] == 2) {
            mark[b] = 1;
            queue.push_back(b);
        }
        if (mark[c] == 2) {
            mark[c] = 1;
            queue.push_back(c);
        }
        merge(a, b); merge(b, c);
        a = c;
    }
}
void augment(int source) {
    queue.clear();
    for (int i = 0; i < n; ++i) {
        next[i] = visit[i] = -1;
        belong[i] = i;
        mark[i] = 0;
    }
    mark[source] = 1;
    queue.push_back(source);
    for (int head = 0; head < (int)queue.size() && match[source] == -1; ++head) {
        int x = queue[head];
        for (int i = 0; i < (int)edge[x].size(); ++i) {
            int y = edge[x][i];
            if (match[x] == y || find(x) == find(y) || mark[y] == 2) {
                continue;
            }
            if (mark[y] == 1) {
                int r = lca(x, y);
                if (find(x) != r) next[x] = y;
                if (find(y) != r) next[y] = x;
                group(x, r); group(y, r);
            } else if (match[y] == -1) {
                next[y] = x;
                for (int u = y; u != -1; ) {
                    int v = next[u];
                    int mv = match[v];
                    match[v] = u;
```

```
                    match[u] = v;
                    u = mv;
                }
                break;
            } else {
                next[y] = x;
                mark[y] = 2;
                mark[match[y]] = 1;
                queue.push_back(match[y]);
            }
        }
    }
}
int solve() {
    std::fill(match, match + n, -1);
    for (int i = 0; i < n; ++i) {
        if (match[i] == -1) augment(i);
    }
    int answer = 0;
    for (int i = 0; i < n; ++i) {
        answer += (match[i] != -1);
    }
    return answer;
}
```

数论及代数

魔幻多项式

快速傅里叶变换

注意事项：请实现复数类 Complex，并注意快速傅里叶变换精度较差，建议使用快速数论变换。

```
int prepare(int n) {
  int len = 1;
  for (; len <= 2 * n; len <<= 1);
  for (int i = 0; i < len; i++) {
    e[0][i] = Complex(cos(2 * pi * i / len), sin(2 * pi * i / len));
    e[1][i] = Complex(cos(2 * pi * i / len), -sin(2 * pi * i / len));
  }
  return len;
}
void DFT(Complex *a, int n, int f) {
  for (int i = 0, j = 0; i < n; i++) {
    if (i > j) std::swap(a[i], a[j]);
    for (int t = n >> 1; (j ^= t) < t; t >>= 1);
  }
  for (int i = 2; i <= n; i <<= 1)
    for (int j = 0; j < n; j += i)
      for (int k = 0; k < (i >> 1); k++) {
        Complex A = a[j + k];
        Complex B = e[f][n / i * k] * a[j + k + (i >> 1)];
        a[j + k] = A + B;
        a[j + k + (i >> 1)] = A - B;
      }
  if (f == 1) {
    for (int i = 0; i < n; i++) a[i].a /= n;
  }
}
```

光速数论变换

注意事项：MOD 应该为一个特殊的质数 2^n+1 且 n 应该要足够大，PRT 为这个质数的原根。

```
// meminit(A, l, r) 是将数组 A 的 [l, r) 清 0。
// memcopy(target, source, l, r) 是将 source 的 [l, r) 复制到 target 的 [l, r)
#define meminit(A, l, r) memset(A + (l), 0, sizeof(*A) * ((r) - (l)))
#define memcopy(B, A, l, r) memcpy(B, A + (l), sizeof(*A) * ((r) - (l)))
void DFT(int *a, int n, int f) { // 封闭形式，常数小（10^7 跑 2.23 秒）
  for (register int i = 0, j = 0; i < n; i++) {
    if (i > j) std::swap(a[i], a[j]);
    for (register int t = n >> 1; (j ^= t) < t; t >>= 1);
  }
  for (register int i = 2; i <= n; i <<= 1) {
    static int exp[MAXN];
    exp[0] = 1; exp[1] = fpm(PRT, (MOD - 1) / i);
    if (f == 1) exp[1] = fpm(exp[1], MOD - 2);
    for (register int k = 2; k < (i >> 1); k++) {
      exp[k] = 1ll * exp[k - 1] * exp[1] % MOD;
    }
    for (register int j = 0; j < n; j += i) {
      for (register int k = 0; k < (i >> 1); k++) {
        register int &pA = a[j + k], &pB = a[j + k + (i >> 1)];
        register int A = pA, B = 1ll * pB * exp[k] % MOD;
        pA = (A + B) % MOD;
        pB = (A - B + MOD) % MOD;
      }
    }
  }
  if (f == 1) {
    register int rev = fpm(n, MOD - 2, MOD);
    for (register int i = 0; i < n; i++) {
      a[i] = 1ll * a[i] * rev % MOD;
    }
  }
}
// 在不写高精度的情况下合并 FFT 所得结果对 MOD 取模后的答案
// 值得注意的是，这个东西不能最后再合并，而是应该每做一次多项式乘法就 CRT 一次
int CRT(int *a) {
  static int x[3];
  for (int i = 0; i < 3; i++) {
    x[i] = a[i];
    for (int j = 0; j < i; j++) {
      int t = (x[i] - x[j] + FFT[i] -> MOD) % FFT[i] -> MOD;
      if (t < 0) t += FFT[i] -> MOD;
      x[i] = 1LL * t * inv[j][i] % FFT[i] -> MOD;
    }
  }
  int sum = 1, ret = x[0] % MOD;
  for (int i = 1; i < 3; i ++) {
    sum = 1LL * sum * FFT[i - 1] -> MOD % MOD;
    ret += 1LL * x[i] * sum % MOD;
    if(ret >= MOD) ret -= MOD;
  }
  return ret;
}
for (int i = 0; i < 3; i++)
  // inv 数组的预处理过程，inverse(x, p) 表示求 x 在 p 下逆元
  for (int j = 0; j < 3; j++)
    inv[i][j] = inverse(FFT[i] -> MOD, FFT[j] -> MOD);
```

牛顿迭代法

问题描述：给出多项式 $G(x)$，求解多项式 $F(x)$ 满足：$G(F(x)) \equiv 0 \pmod{x^n}$ 答案只需要精确到 $F(x) \bmod x^n$ 即可。

实现原理：考虑倍增，假设有：

$$G(F_t(x)) \equiv 0 \pmod{x^t}$$

对 $G(F_{t+1}(x))$ 在模 x^{2t} 意义下进行 Taylor 展开：

$$G(F_{t+1}(x)) \equiv G(F_t(x)) + \frac{G'(F_t(x))}{1!}(F_{t+1}(x) - F_t(x)) \pmod{x^{2t}}$$

那么就有：$F_{t+1}(x) \equiv F_t(x) - \frac{G(F_t(x))}{G'(F_t(x))} \pmod{x^{2t}}$

注意事项： $G(F(x))$ 的常数项系数必然为 0，这个可以作为求解的初始条件；

多项式求逆
原理： 令 $G(x) = x * A - 1$（其中 A 是一个多项式系数），根据牛顿迭代法有：

$$F_{t+1}(x) \equiv F_t(x) - \frac{F_t(x) * A(x) - 1}{A(x)} \equiv 2F_t(x) - F_t(x)^2 * A(x) \pmod{x^{2t}}$$

注意事项：

1. $F(x)$ 的常数项系数必然不为 0，否则没有逆元；
2. 复杂度是 $O(n \log n)$ 但是常数比较大（10^5 大概需要 0.3 秒左右）；
3. 传入的两个数组必须不同，但传入的次数界没有必要是 2 的次幂；

```
void getInv(int *a, int *b, int n) {
  static int tmp[MAXN];
  b[0] = fpm(a[0], MOD - 2, MOD);
  for (int c = 2, M = 1; c < (n << 1); c <<= 1) {
    for (; M <= 3 * (c - 1); M <<= 1);
    meminit(b, c, M); meminit(tmp, c, M);
    memcopy(tmp, a, 0, c);
    DFT(tmp, M, 0); DFT(b, M, 0);
    for (int i = 0; i < M; i++) {
      b[i] = 1ll * b[i] * (2ll - 1ll * tmp[i] * b[i] % MOD + MOD) % MOD;
    }
    DFT(b, M, 1);
    meminit(b, c, M);
  }
}
```

多项式取指数和对数
作用： 给出一个多项式 $A(x)$，求一个多项式 $F(x)$ 满足 $e^A(x) - F(x) \equiv 0 \pmod{x^n}$。
原理： 令 $G(x) = \ln x - A$（其中 A 是一个多项式系数），根据牛顿迭代法有：

$$F_{t+1}(x) \equiv F_t(x) - F_t(x)(\ln F_t(x) - A(x)) \pmod{x^{2t}}$$

求 $\ln F_t(x)$ 可以用先求导再积分的办法，即：$\ln A(x) = \int \frac{F'(x)}{F(x)} \mathrm{d}x$ 多项式的求导和积分可以在 $O(n)$ 的时间内完成，因此总复杂度为 $O(n \log n)$。
应用： 加速多项式快速幂。
注意事项：

1. 进行 log 的多项式必须保证常数项系数为 1，否则必须要先求出 $\log a[0]$ 是多少；
2. 传入的两个数组必须不同，但传入的次数界没有必要是 2 的次幂；
3. 常数比较大，10^5 的数据求指数和对数分别需要 0.37s 和 0.85s 左右的时间，注意这里 `memset` 几乎不占用时。

```
void getDiff(int *a, int *b, int n) { // 多项式取微分
  for (int i = 0; i + 1 < n; i++) b[i] = 1ll * (i + 1) * a[i + 1] % MOD;
  b[n - 1] = 0;
}
void getInt(int *a, int *b, int n) { // 多项式取积分，积分常数为 0
  static int inv[MAXN];
  inv[1] = 1; b[0] = 0;
  for (int i = 2; i < n; i++)  inv[i] = 1ll * (MOD - MOD / i) * inv[MOD % i] % MOD;
  for (int i = 1; i < n; i++) b[i] = 1ll * a[i - 1] * inv[i] % MOD;
}
void getLn(int *a, int *b, int n) {
  static int inv[MAXN], d[MAXN];
  int M = 1;
  for (; M <= 2 * (n - 1); M <<= 1);
  getInv(a, inv, n); getDiff(a, d, n);
  meminit(d, n, M); meminit(inv, n, M);
  DFT(d, M, 0); DFT(inv, M, 0);
  for (int i = 0; i < M; i++) d[i] = 1ll * d[i] * inv[i] % MOD;
  DFT(d, M, 1);
  getInt(d, b, n);
}
void getExp(int *a, int *b, int n) {
  static int ln[MAXN], tmp[MAXN];
  b[0] = 1;
  for (int c = 2, M = 1; c < (n << 1); c <<= 1) {
    for (; M <= 2 * (c - 1); M <<= 1);
    int bound = std::min(c, n);
    memcopy(tmp, a, 0, bound);
    meminit(tmp, bound, M); meminit(b, c, M);
    getLn(b, ln, c);
    meminit(ln, c, M);
    DFT(b, M, 0); DFT(tmp, M, 0); DFT(ln, M, 0);
    for (int i = 0; i < M; i++) b[i] = 1ll * b[i] * (1ll - ln[i] + tmp[i] + MOD)
 ↪  % MOD;
    DFT(b, M, 1);
    meminit(b, c, M);
  }
}
```

多项式除法
作用： 给出两个多项式 $A(x)$ 和 $B(x)$，求两个多项式 $D(x)$ 和 $R(x)$ 满足：

$$A(x) \equiv D(x)B(x) + R(x) \pmod{x^n}$$

注意事项：

1. 常数比较大概为 6 倍 FFT 的时间，即大约 10^5 的数据 0.07s 左右；
2. 传入两个多项式的次数界，没有必要是 2 的次幂，但是要保证除数多项式不为 0。

```
void divide(int n, int m, int *a, int *b, int *d, int *r) {
 ↪  // n、m 分别为多项式 A（被除数）和 B（除数）的次数界
  static int M, tA[MAXN], tB[MAXN], inv[MAXN], tD[MAXN];
  for (; n > 0 && a[n - 1] == 0; n--);
  for (; m > 0 && b[m - 1] == 0; m--);
  for (int i = 0; i < n; i++) tA[i] = a[n - i - 1];
  for (int i = 0; i < m; i++) tB[i] = b[m - i - 1];
  for (M = 1; M <= n - m + 1; M <<= 1);
  meminit(tB, m, M);
  getInv(tB, inv, M);
  for (M = 1; M <= 2 * (n - m + 1); M <<= 1);
  meminit(inv, n - m + 1, M); DFT(inv, M, 0);
  meminit(tA, n - m + 1, M); DFT(tA, M, 0);
  for (int i = 0; i < M; i++) d[i] = 1ll * inv[i] * tA[i] % MOD;
  DFT(d, M, 1);
  std::reverse(d, d + n - m + 1);
  for (M = 1; M <= n; M <<= 1);
  memcopy(tB, b, 0, m); meminit(tB, m, M);
  memcopy(tD, d, 0, n - m + 1); meminit(tD, n - m + 1, M);
```

```
  DFT(tD, M, 0); DFT(tB, M, 0);
  for (int i = 0; i < M; i++) r[i] = 1ll * tD[i] * tB[i] % MOD;
  DFT(r, M, 1);
  meminit(r, n, M);
  for (int i = 0; i < n; i++) r[i] = (a[i] - r[i] + MOD) % MOD;
}
```

线性递推数列求第 n 项

```
//已知 a_0, a_1, ..., a_{m-1}
//  a_n = c_0 * a_{n-m} + ... + c_{m-1} * a_{n-1}
//  求 a_n = v_0 * a_0 + v_1 * a_1 + ... + v_{m-1} * a_{m-1}
void linear_recurrence(long long n, int m, int a[], int c[], int p) {
  long long v[M] = {1 % p}, u[M << 1], msk = !!n;
  for(long long i(n); i > 1; i >>= 1) {
    msk <<= 1;
  }
  for(long long x(0); msk; msk >>= 1, x <<= 1) {
    fill_n(u, m << 1, 0);
    int b(!!(n & msk)); x |= b;
    if(x < m) u[x] = 1 % p;
    else {
      for(int i(0); i < m; i++) {
        for(int j(0), t(i + b); j < m; j++, t++) {
          u[t] = (u[t] + v[i] * v[j]) % p;
        }
      }
      for(int i((m << 1) - 1); i >= m; i--) {
        for(int j(0), t(i - m); j < m; j++, t++) {
          u[t] = (u[t] + c[j] * u[i]) % p;
        }
      }
    }
    copy(u, u + m, v);
  }
  //a[n] = v[0] * a[0] + v[1] * a[1] + ... + v[m - 1] * a[m - 1]
  for(int i(m); i < 2 * m; i++) {
    a[i] = 0;
    for(int j(0); j < m; j++) {
      a[i] = (a[i] + (long long)c[j] * a[i + j - m]) % p;
    }
  }
  for(int j(0); j < m; j++) {
    b[j] = 0;
    for(int i(0); i < m; i++) {
      b[j] = (b[j] + v[i] * a[i + j]) % p;
    }
  }
  for(int j(0); j < m; j++) a[j] = b[j];
}
```

线性规划

注意事项： 使用单纯形法求解：

$$max\{c_{1\times m}\cdot x_{m\times 1}\mid x_{m\times 1}\geq 0_{m\times 1}, a_{n\times m}\cdot x_{m\times 1}\leq b_{n\times 1}\}$$

```
std::vector<double> solve(const std::vector<std::vector<double> > &a,
              const std::vector<double> &b, const std::vector<double> &c) {
  int n = (int)a.size(), m = (int)a[0].size() + 1;
  std::vector<std::vector<double> > value(n + 2, std::vector<double>(m + 1));
  std::vector<int> index(n + m);
  int r = n, s = m - 1;
  for (int i = 0; i < n + m; ++i) index[i] = i;
  for (int i = 0; i < n; ++i) {
    for (int j = 0; j < m - 1; ++j) {
      value[i][j] = -a[i][j];
    }
    value[i][m - 1] = 1;
    value[i][m] = b[i];
    if (value[r][m] > value[i][m]) r = i;
  }
  for (int j = 0; j < m - 1; ++j) value[n][j] = c[j];
  value[n + 1][m - 1] = -1;
  for (double number; ; ) {
    if (r < n) {
      std::swap(index[s], index[r + m]);
      value[r][s] = 1 / value[r][s];
      for (int j = 0; j <= m; ++j) {
        if (j != s) value[r][j] *= -value[r][s];
      }
      for (int i = 0; i <= n + 1; ++i) {
        if (i != r) {
          for (int j = 0; j <= m; ++j) {
            if (j != s) value[i][j] += value[r][j] * value[i][s];
          }
          value[i][s] *= value[r][s];
        }
      }
    }
    r = s = -1;
    for (int j = 0; j < m; ++j) {
      if (s < 0 || index[s] > index[j]) {
        if (value[n + 1][j] > eps || value[n + 1][j] > -eps && value[n][j] > eps){
          s = j;
        }
      }
    }
    if (s < 0) break;
    for (int i = 0; i < n; ++i) {
      if (value[i][s] < -eps) {
        if (r < 0 || (number = value[r][m] / value[r][s] - value[i][m] /
 ↪  value[i][s]) < -eps || number < eps && index[r + m] > index[i + m]) {
          r = i;
        }
      }
    }
    if (r < 0) return std::vector<double>(); //  Solution is unbounded.
  }
  if (value[n + 1][m] < -eps) return std::vector<double>(); // No solution.
  std::vector<double> answer(m - 1);
  for (int i = m; i < n + m; ++i) {
    if (index[i] < m - 1) answer[index[i]] = value[i - m][m];
  }
  return answer;
}
```

中国剩余定理

注意事项： p_i 无需两两互质

```
bool solve(int n, std::pair<long long, long long> input[],
                  std::pair<long long, long long> &output) {
  output = std::make_pair(1, 1);
  for (int i = 0; i < n; ++i) {
    long long number, useless; // euclid(a, b, x, y)
    euclid(output.second, input[i].second, number, useless);
    long long divisor = std::__gcd(output.second, input[i].second);
```

```
    if ((input[i].first - output.first) % divisor) return false;
    number *= (input[i].first - output.first) / divisor;
    fix(number, input[i].second);
    output.first += output.second * number;
    output.second *= input[i].second / divisor;
    fix(output.first, output.second);
  }
  return true;
}
```

直线下整点个数

注意事项： 返回结果为：$\sum_{0 \le i < n} \lfloor \frac{a + b \cdot i}{m} \rfloor$ 即直线下整点个数。

```
long long solve(const long long &n, const long long &a,
                const long long &b, const long long &m) {
  if (b == 0) return n * (a / m);
  if (a >= m) return n * (a / m) + solve(n, a % m, b, m);
  if (b >= m) return (n - 1) * n / 2 * (b / m) + solve(n, a, b % m, m);
  return solve((a + b * n) / m, (a + b * n) % m, m, b);
}
```

闪电素数判定

```
const int BASE[12] = {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37};
bool check(const long long &prime, const long long &base) {
  long long number = prime - 1;
  for (; ~number & 1; number >>= 1);
  long long result = power_mod(base, number, prime);
  for (; number != prime - 1 && result != 1 && result != prime - 1; number <<= 1)
  ↪  {
    result = multiply_mod(result, result, prime);
  }
  return result == prime - 1 || (number & 1) == 1;
}
bool miller_rabin(const long long &number) {
  if (number < 2) return false;
  if (number < 4) return true;
  if (~number & 1) return false;
  for (int i = 0; i < 12 && BASE[i] < number; ++i) {
    if (!check(number, BASE[i])) {
      return false;
    }
  }
  return true;
}
```

闪电质因数分解

```
long long pollard_rho(const long long &number, const long long &seed) {
  long long x = rand() % (number - 1) + 1, y = x;
  for (int head = 1, tail = 2; ; ) {
    x = multiply_mod(x, x, number);
    x = add_mod(x, seed, number);
    if (x == y) return number;
    long long answer = std::__gcd(abs(x - y), number);
    if (answer > 1 && answer < number) return answer;
    if (++head == tail) {
      y = x;
      tail <<= 1;
    }
  }
}
void factorize(const long long &number, std::vector<long long> &divisor) {
  if (number > 1) {
    if (miller_rabin(number)) divisor.push_back(number);
    else {
      long long factor = number;
      for (; factor >= number; factor = pollard_rho(number, rand() % (number - 1)
      ↪  + 1));
      factorize(number / factor, divisor);
      factorize(factor, divisor);
    }
  }
}
```

自适应辛普森

```
double area(const double &left, const double &right) {
  double mid = (left + right) / 2;
  return (right - left) * (calc(left) + 4 * calc(mid) + calc(right)) / 6;
}
double simpson(const double &left, const double &right,
               const double &eps, const double &area_sum) {
  double mid = (left + right) / 2;
  double area_left = area(left, mid), area_right = area(mid, right);
  double area_total = area_left + area_right;
  if (std::abs(area_total - area_sum) < 15 * eps) {
    return area_total + (area_total - area_sum) / 15;
  }
  return simpson(left, mid, eps / 2, area_left) + simpson(mid, right, eps / 2,
  ↪  area_right);
}
double simpson(const double &left, const double &right, const double &eps) {
  return simpson(left, right, eps, area(left, right));
}
```

二次剩余

```
void calcH(int &t, int &h, const int p) {
  int tmp = p - 1; for (t = 0; (tmp & 1) == 0; tmp /= 2) t++; h = tmp;
}
// solve equation x^2 mod p = a
bool solve(int a, int p, int &x, int &y) {
  srand(19920225);
  if (p == 2) { x = y = 1; return true; }
  int p2 = p / 2, tmp = power(a, p2, p);
  if (tmp == p - 1) return false;
  if ((p + 1) % 4 == 0) {
    x = power(a, (p + 1) / 4, p); y = p - x; return true;
  } else {
    int t, h, b, pb; calcH(t, h, p);
    if (t >= 2) {
      do {b = rand() % (p - 2) + 2;
      } while (power(b, p / 2, p) != p - 1);
      pb = power(b, h, p);
    } int s = power(a, h / 2, p);
    for (int step = 2; step <= t; step++) {
      int ss = (((long long)(s * s) % p) * a) % p;
      for (int i = 0; i < t - step; i++) ss = ((long long)ss * ss) % p;
      if (ss + 1 == p) s = (s * pb) % p; pb = ((long long)pb * pb) % p;
    } x = ((long long)s * a) % p; y = p - x;
  } return true;
}
```

Pell 方程

```
ULL A,B,p[maxn],q[maxn],a[maxn],g[maxn],h[maxn];
int main() {
  for (int test=1, n;scanf("%d",&n) && n;++test) {
    printf("Case %d: ",test);
    if (fabs(sqrt(n)-floor(sqrt(n)+1e-7))<=1e-7)   {
      int a=(int)(floor(sqrt(n)+1e-7)); printf("%d %d\n",a,1);
    } else { // 求 x^2 - ny^2 = 1 的最小正整数根，n 不是完全平方数
      p[1]=q[0]=h[1]=1;p[0]=q[1]=g[1]=0;
      a[2]=(int)(floor(sqrt(n)+1e-7));
      for (int i=2;i;++i) {
        g[i]=-g[i-1]+a[i]*h[i-1]; h[i]=(n-sqr(g[i]))/h[i-1];
        a[i+1]=(g[i]+a[2])/h[i]; p[i]=a[i]*p[i-1]+p[i-2];
        q[i]=a[i]*q[i-1]+q[i-2];
        if (sqr((ULL)(p[i]))-n*sqr((ULL)(q[i]))==1){
          A=p[i];B=q[i];break;
        }
      }
      cout << A << ' ' << B <<endl;
    }
  }
}
```

原根相关

1. 模 m 有原根的充要条件：$m = 2, 4, p^a, 2p^a$，其中 p 是奇素数；
2. 求任意数 p 原根的方法：对 $\phi(p)$ 因式分解，即 $\phi(p) = p_1^{r_1} p_2^{r_2} \cdots p_k^{r_k}$，若恒成立：

$$g^{\frac{p-1}{g}} \neq 1 \pmod{p}$$

那么 g 就是 p 的原根。

3. 若模 m 有原根，那么它一共有 $\phi(\phi(m))$ 个原根。

字符串

广义后缀自动机

注意事项：空间是插入字符串总长度的 2 倍并请注意字符集大小。

```
void add(int x, int &last) {
  int lastnode = last;
  if (c[lastnode][x]) {
    int nownode = c[lastnode][x];
    if (l[nownode] == l[lastnode] + 1) last = nownode;
    else{
      int auxnode = ++size; l[auxnode] = l[lastnode] + 1;
      for (int i = 0; i < 26; i++) c[auxnode][i] = c[nownode][i];
      f[auxnode] = f[nownode]; f[nownode] = auxnode;
      for (; lastnode && c[lastnode][x] == nownode; lastnode = f[lastnode]) {
        c[lastnode][x] = auxnode;
      }
      last = auxnode;
    }
  } else { // Naive Suffix Automaton
    int newnode = ++size; l[newnode] = l[lastnode] + 1;
    for (; lastnode && !c[lastnode][x]; lastnode = f[lastnode]) c[lastnode][x] =
 ↪  newnode;
    if (!lastnode) f[newnode] = 1;
    else{
      int nownode = c[lastnode][x];
      if (l[lastnode] + 1 == l[nownode]) f[newnode] = nownode;
      else{
        int auxnode = ++size; l[auxnode] = l[lastnode] + 1;
        for (int i = 0; i < 26; i++) c[auxnode][i] = c[nownode][i];
        f[auxnode] = f[nownode]; f[nownode] = f[newnode] = auxnode;
        for (; lastnode && c[lastnode][x] == nownode; lastnode = f[lastnode]) {
          c[lastnode][x] = auxnode;
        }
      }
    }
    last = newnode;
  }
}
```

后缀数组

注意事项：$\mathcal{O}(n \log n)$ 倍增构造。

```
namespace suffix_array{
  int wa[MAXN], wb[MAXN], ws[MAXN], wv[MAXN];
  bool 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 *x = wa, *y = wb, *t;
    for (int i = 0; i < m; i++) ws[i] = 0;
    for (int i = 0; i < n; i++) ws[x[i] = r[i]]++;
    for (int i = 1; i < m; i++) ws[i] += ws[i - 1];
    for (int i = n - 1; i >= 0; i--) sa[--ws[x[i]]] = i;
    for (int i, j = 1, p = 1; p < n; j <<= 1, 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++) ws[i] = 0;
      for (i = 0; i < n; i++) ws[wv[i]]++;
      for (i = 1; i < m; i++) ws[i] += ws[i - 1];
      for (i = n - 1; i >= 0; i--) sa[--ws[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 getheight(int *r, int *sa, int *rk, int *h, int n) {
    for (int i = 1; i <= n; i++) rk[sa[i]] = i;
    for (int i = 0, j, k = 0; i < n; h[rk[i++]] = k)
      for (k ? k-- : 0, j = sa[rk[i] - 1]; r[i + k] == r[j + k]; k++);
  }
};
suffix_array::DA(r, sa, n + 1, CHAR_SET);  // Usage: 0-based
suffix_array::getheight(r, sa, rk, h, n);
```

回文自动机

注意事项：请注意字符集大小。

```
struct Palindromic_Tree{
  int nTree, nStr, last, c[MAXT][26], fail[MAXT], r[MAXN], l[MAXN], s[MAXN];
  int allocate(int len) {
    l[nTree] = len;
    r[nTree] = 0;
    fail[nTree] = 0;
    memset(c[nTree], 0, sizeof(c[nTree]));
    return nTree++;
  }
  void init() {
    nTree = nStr = 0;
    int newEven = allocate(0);
    int newOdd = allocate(-1);
    last = newEven;
```

```
    fail[newEven] = newOdd;
    fail[newOdd] = newEven;
    s[0] = -1;
  }
  void add(int x) {
    s[++nStr] = x;
    int nownode = last;
    while (s[nStr - l[nownode] - 1] != s[nStr]) nownode = fail[nownode];
    if (!c[nownode][x]) {
      int newnode = allocate(l[nownode] + 2), &newfail = fail[newnode];
      newfail = fail[nownode];
      while (s[nStr - l[newfail] - 1] != s[nStr]) newfail = fail[newfail];
      newfail = c[newfail][x];
      c[nownode][x] = newnode;
    }
    last = c[nownode][x];
    r[last]++;
  }
  void count() {
    for (int i = nTree - 1; i >= 0; i--) {
      r[fail[i]] += r[i];
    }
  }
}
```

Manacher

注意事项： 1-based 算法，请注意下标。

```
int manacher(char *text, int length, int *palindrome) {
  static char buffer[MAXN];
  for (int i = 1; i <= length; i++) {
    buffer[2 * i - 1] = text[i];
    if (i != 0) buffer[2 * i] = '#';
  }
  palindrome[1] = 1;
  for (int i = 2, j = 0; i <= 2 * length - 1; ++i) {
    if (j + palindrome[j] <= i) palindrome[i] = 0;
    else palindrome[i] = std::min(palindrome[(j << 1) - i], j + palindrome[j] -
 ↪  i);
    while (i - palindrome[i] >= 1 && i + palindrome[i] <= 2 * length - 1 &&
 ↪  buffer[i - palindrome[i]] == buffer[i + palindrome[i]]) {
      palindrome[i]++;
    }
    if (i + palindrome[i] > j + palindrome[j]) j = i;
  }
  int answer = 0;
  for (int i = 1; i < 2 * length; i++) {
    if (i & 1) answer = std::max(answer, 2 * (palindrome[i] - 1 >> 1) + 1);
    else answer = std::max(answer, 2 * (palindrome[i] >> 1));
  }
  return answer;
}
```

循环串的最小表示

注意事项： 0-Based 算法，请注意下标。

```
int minrep(const char *s) {
  int length = strlen(s), i = 0, j = 1, k = 0;
  while (i < length && j < length && k < length) {
    if (s[(i + k) % length] == s[(j + k) % length]) k++;
    else {
      if (s[(i + k) % length] > s[(j + k) % length]) i += k + 1;
      else j += k + 1;
      if (i == j) j++;
      k = 0;
    }
  }
  return std::min(i, j);
}
```

后缀树

注意事项：

1. 边上的字符区间是左闭右开区间；
2. 如果要建立关于多个串的后缀树，请用不同的分隔符，并且对于每个叶子结点，去掉和它父亲的连边上出现的第一个分隔符之后的所有字符；

```
const int MAXL = 100001; // The length of the string being inserted into the ST.
const int MAXD = 27;     // The size of the alphabet.
struct SuffixTree{
  int size, length, pCur, dCur, lCur, lBuf, text[MAXL];
  std::pair<int, int> suffix[MAXL];
  struct Node{
    int left, right, sLink, next[MAXD];
  }tree[MAXL * 2];
  int getLength(const int &rhs) {
    return tree[rhs].right ? tree[rhs].right - tree[rhs].left : length + 1 -
 ↪  tree[rhs].left;
  }
  void addLink(int &last, int node) {
    if (last != 0) tree[last].sLink = node;
    last = node;
  }
  int alloc(int left, int right = 0) {
    size++;
    memset(&tree[size], 0, sizeof(tree[size]));
    tree[size].left = left;
    tree[size].right = right;
    tree[size].sLink = 1;
    return size;
  }
  bool move(int node) {
    int length = getLength(node);
    if (lCur >= length) {
      lCur -= length;
      dCur += length;
      pCur = node;
      return true;
    }
    return false;
  }
  void init() {
    size = length = 0;
    lCur = dCur = lBuf = 0;
    pCur = alloc(0);
  }
  void extend(int x) {
    text[++length] = x;
    lBuf++;
    for (int last = 0; lBuf > 0; ) {
      if (lCur == 0) dCur = length;
      if (!tree[pCur].next[text[dCur]]) {
        int newleaf = alloc(length);
        tree[pCur].next[text[dCur]] = newleaf;
        suffix[length + 1 - lBuf] = std::make_pair(pCur, newleaf);
        addLink(last, pCur);
```

```
      } else {
        int nownode = tree[pCur].next[text[dCur]];
        if (move(nownode)) continue;
        if (text[tree[nownode].left + lCur] == x) {
          lCur++;
          addLink(last, pCur);
          break;
        }
        int newleaf = alloc(length), newnode = alloc(tree[nownode].left,
    ↪  tree[nownode].left + lCur);
        tree[nownode].left += lCur;
        tree[pCur].next[text[dCur]] = newnode;
        tree[newnode].next[x] = newleaf;
        tree[newnode].next[text[tree[nownode].left]] = nownode;
        suffix[length + 1 - lBuf] = std::make_pair(newnode, newleaf);
        addLink(last, newnode);
      }
      lBuf--;
      if (pCur == 1 && lCur > 0) lCur--, dCur++;
      else pCur = tree[pCur].sLink;
    }
  }
};
```

数据结构

树链剖分

点操作版本

```
void modify(int x, int y, int val) {
  int fx = t[x], fy = t[y];
  while (fx != fy) {
    if (d[fx] > d[fy]) {
      modify(1, 1, n, w[fx], w[x], val);
      x = f[fx]; fx = t[x];
    } else {
      modify(1, 1, n, w[fy], w[y], val);
      y = f[fy]; fy = t[y];
    }
  }
  if (d[x] < d[y]) modify(1, 1, n, w[x], w[y], val);
  else modify(1, 1, n, w[y], w[x], val);
}
Node query(int x, int y) {
  int fx = t[x], fy = t[y];
  Node left = Node(), right = Node();
  while (fx != fy) {
    if (d[fx] > d[fy]) {
      left = query(1, 1, n, w[fx], w[x]) + left;
      x = f[fx]; fx = t[x];
    } else {
      right = query(1, 1, n, w[fy], w[y]) + right;
      y = f[fy]; fy = t[y];
    }
  }
  if (d[x] < d[y]) {
    right = query(1, 1, n, w[x], w[y]) + right;
  } else {
    left = query(1, 1, n, w[y], w[x]) + left;
  }
  std::swap(left.lsum, left.rsum);
  return left + right;
}
```

边操作版本

```
void modify(int x, int y) {
  int fx = t[x], fy = t[y];
  while (fx != fy) {
    if (d[fx] > d[fy]) {
      modify(1, 1, n, w[fx], w[x]);
      x = f[fx]; fx = t[x];
    } else {
      modify(1, 1, n, w[fy], w[y]);
      y = f[fy]; fy = t[y];
    }
  }
  if (x != y) {
    if (d[x] < d[y]) modify(1, 1, n, w[z[x]], w[y]);
    else modify(1, 1, n, w[z[y]], w[x]);
  }
}
// TODO 边询问
```

Link Cut Tree

```
struct MsgNode{
  int leftColor, rightColor, answer;
  MsgNode() {
    leftColor = -1;
    rightColor = -1;
    answer = 0;
  }
  MsgNode(int c) {
    leftColor = rightColor = c;
    answer = 1;
  }
  MsgNode operator +(const MsgNode &p)const {
    if (answer == 0) return p;
    if (p.answer == 0) return *this;
    MsgNode ret;
    ret.leftColor = leftColor;
    ret.rightColor = p.rightColor;
    ret.answer = answer + p.answer - (rightColor == p.leftColor);
    return ret;
  }
}d[MAXN], g[MAXN];
int n, m, c[MAXN][2], f[MAXN], p[MAXN], s[MAXN], flag[MAXN];
bool r[MAXN];
void init(int x, int value) {
  d[x] = g[x] = MsgNode(value);
  c[x][0] = c[x][1] = 0;
  f[x] = p[x] = flag[x] = -1;
  s[x] = 1;
}
void update(int x) {
  s[x] = s[c[x][0]] + s[c[x][1]] + 1; g[x] = MsgNode();
  if (c[x][0 ^ r[x]]) g[x] = g[x] + g[c[x][0 ^ r[x]]];
  g[x] = g[x] + d[x];
  if (c[x][1 ^ r[x]]) g[x] = g[x] + g[c[x][1 ^ r[x]]];
}
void makesame(int x, int c) {
  flag[x] = c;
  d[x] = MsgNode(c);
  g[x] = MsgNode(c);
}
void pushdown(int x) {
  if (r[x]) {
```

```
    std::swap(c[x][0], c[x][1]);
    r[c[x][0]] ^= 1;
    r[c[x][1]] ^= 1;
    std::swap(g[c[x][0]].leftColor, g[c[x][0]].rightColor);
    std::swap(g[c[x][1]].leftColor, g[c[x][1]].rightColor);
    r[x] = false;
  }
  if (flag[x] != -1) {
    if (c[x][0]) makesame(c[x][0], flag[x]);
    if (c[x][1]) makesame(c[x][1], flag[x]);
    flag[x] = -1;
  }
}
void rotate(int x, int k) {
  pushdown(x); pushdown(c[x][k]);
  int y = c[x][k]; c[x][k] = c[y][k ^ 1]; c[y][k ^ 1] = x;
  if (f[x] != -1) c[f[x]][c[f[x]][1] == x] = y;
  f[y] = f[x]; f[x] = y; f[c[x][k]] = x; std::swap(p[x], p[y]);
  update(x); update(y);
}
void splay(int x, int s = -1) {
  pushdown(x);
  while (f[x] != s) {
    if (f[f[x]] != s) rotate(f[f[x]], (c[f[f[x]]][1] == f[x]) ^ r[f[f[x]]]);
    rotate(f[x], (c[f[x]][1] == x) ^ r[f[x]]);
  }
  update(x);
}
void access(int x) {
  int y = 0;
  while (x != -1) {
    splay(x); pushdown(x);
    f[c[x][1]] = -1; p[c[x][1]] = x;
    c[x][1] = y; f[y] = x; p[y] = -1;
    update(x); x = p[y = x];
  }
}
void setroot(int x) {
  access(x); splay(x); r[x] ^= 1;
  std::swap(g[x].leftColor, g[x].rightColor);
}
void link(int x, int y) {
  setroot(x); p[x] = y;
}
void cut(int x, int y) {
    access(x); splay(y, -1);
    if (p[y] == x) p[y] = -1;
    else {
      access(y); splay(x,-1);
      p[x] = -1;
    }
}
```

可持久化平衡树

```
int ran() {
  static int ret = 182381727;
  return (ret += (ret << 1) + 717271723) & (~0u >> 1);
}
int alloc(int node = 0) {
  size++;
  if (node) {
    c[size][0] = c[node][0];
    c[size][1] = c[node][1];
    s[size] = s[node];
    d[size] = d[node];
  }
  else{
    c[size][0] = c[size][1] = 0;
    s[size] = 1;
    d[size] = ' ';
  }
  return size;
}
void update(int x) {
  s[x] = 1;
  if (c[x][0]) s[x] += s[c[x][0]];
  if (c[x][1]) s[x] += s[c[x][1]];
}
int merge(const std::pair<int, int> &a) {
  if (!a.first) return a.second;
  if (!a.second) return a.first;
  if (ran() % (s[a.first] + s[a.second]) < s[a.first]) {
    int newnode = alloc(a.first);
    c[newnode][1] = merge(std::make_pair(c[newnode][1], a.second));
    update(newnode);
    return newnode;
  } else {
    int newnode = alloc(a.second);
    c[newnode][0] = merge(std::make_pair(a.first, c[newnode][0]));
    update(newnode);
    return newnode;
  }
}
std::pair<int, int> split(int x, int k) {
  if (!x || !k) return std::make_pair(0, x);
  int newnode = alloc(x);
  if (k <= s[c[x][0]]) {
    std::pair<int, int> ret = split(c[newnode][0], k);
    c[newnode][0] = ret.second;
    update(newnode);
    return std::make_pair(ret.first, newnode);
  } else {
    std::pair<int, int> ret = split(c[newnode][1], k - s[c[x][0]] - 1);
    c[newnode][1] = ret.first;
    update(newnode);
    return std::make_pair(newnode, ret.second);
  }
}
int build(int l, int r) {
  int newnode = alloc();
  d[newnode] = tmp[l + r >> 1];
  if (l <= (l + r >> 1) - 1) c[newnode][0] = build(l, (l + r >> 1) - 1);
  if ((l + r >> 1) + 1 <= r) c[newnode][1] = build((l + r >> 1) + 1, r);
  update(newnode);
  return newnode;
}
```

可持久化左偏树

```
class Node {
public:
  Node *left, *right;
  int key, dist;
  Node(int key) : left(NULL), right(NULL), key(key), dist(0) {}
  Node* update() {
    if (!left || (right && left->dist < right->dist)) std::swap(left, right);
```

```
    dist = right ? right->dist + 1 : 0;
    return this;
  }
};
Node* merge(Node *x, Node *y) {
  if (!x) return y;
  if (!y) return x;
  if (x->key < y->key) {
    x = new Node(*x);
    x->right = merge(x->right, y);
    return x->update();
  } else {
    y = new Node(*y);
    y->right = merge(x, y->right);
    return y->update();
  }
}
```

k-d Tree

```
struct Point{
  int data[MAXK], id;
}p[MAXN];
struct KdNode{
  int l, r;
  Point p, dmin, dmax;
  KdNode() {}
  KdNode(const Point &rhs) : l(0), r(0), p(rhs), dmin(rhs), dmax(rhs) {}
  inline void merge(const KdNode &rhs) {
    for (register int i = 0; i < k; i++) {
      dmin.data[i] = std::min(dmin.data[i], rhs.dmin.data[i]);
      dmax.data[i] = std::max(dmax.data[i], rhs.dmax.data[i]);
    }
  }
  inline long long getMinDist(const Point &rhs)const {
    register long long ret = 0;
    for (register int i = 0; i < k; i++) {
      if (dmin.data[i] <= rhs.data[i] && rhs.data[i] <= dmax.data[i]) continue;
      ret += std::min(1ll * (dmin.data[i] - rhs.data[i]) * (dmin.data[i] -
 ↪  rhs.data[i]),
          1ll * (dmax.data[i] - rhs.data[i]) * (dmax.data[i] - rhs.data[i]));
    }
    return ret;
  }
  long long getMaxDist(const Point &rhs) {
    long long ret = 0;
    for (register int i = 0; i < k; i++) {
      int tmp = std::max(std::abs(dmin.data[i] - rhs.data[i]),
          std::abs(dmax.data[i] - rhs.data[i]));
      ret += 1ll * tmp * tmp;
    }
    return ret;
  }
}tree[MAXN * 4];
struct Result{
  long long dist;
  Point d;
  Result() {}
  Result(const long long &dist, const Point &d) : dist(dist), d(d) {}
  bool operator >(const Result &rhs)const {
    return dist > rhs.dist || (dist == rhs.dist && d.id < rhs.d.id);
  }
  bool operator <(const Result &rhs)const {
    return dist < rhs.dist || (dist == rhs.dist && d.id > rhs.d.id);
  }
};
inline long long sqrdist(const Point &a, const Point &b) {
  register long long ret = 0;
  for (register int i = 0; i < k; i++) {
    ret += 1ll * (a.data[i] - b.data[i]) * (a.data[i] - b.data[i]);
  }
  return ret;
}
inline int alloc() {
  size++;
  tree[size].l = tree[size].r = 0;
  return size;
}
void build(const int &depth, int &rt, const int &l, const int &r) {
  if (l > r) return;
  register int middle = l + r >> 1;
  std::nth_element(p + l, p + middle, p + r + 1,
    [=](const Point &a, const Point &b){return a.data[depth] < b.data[depth];};
  tree[rt = alloc()] = KdNode(p[middle]);
  if (l == r) return;
  build((depth + 1) % k, tree[rt].l, l, middle - 1);
  build((depth + 1) % k, tree[rt].r, middle + 1, r);
  if (tree[rt].l) tree[rt].merge(tree[tree[rt].l]);
  if (tree[rt].r) tree[rt].merge(tree[tree[rt].r]);
}
std::priority_queue<Result, std::vector<Result>, std::greater<Result> > heap;
void getMinKth(const int &depth, const int &rt, const int &m, const Point &d) {
 ↪  // 求 K 近点
  Result tmp = Result(sqrdist(tree[rt].p, d), tree[rt].p);
  if ((int)heap.size() < m) {
    heap.push(tmp);
  } else if (tmp < heap.top()) {
    heap.pop();
    heap.push(tmp);
  }
  int x = tree[rt].l, y = tree[rt].r;
  if (x != 0 && y != 0 && sqrdist(d, tree[x].p) > sqrdist(d, tree[y].p))
 ↪  std::swap(x, y);
  if (x != 0 && ((int)heap.size() < m || tree[x].getMinDist(d) <
 ↪  heap.top().dist)) {
    getMinKth((depth + 1) % k, x, m, d);
  }
  if (y != 0 && ((int)heap.size() < m || tree[y].getMinDist(d) <
 ↪  heap.top().dist)) {
    getMinKth((depth + 1) % k, y, m, d);
  }
}
void getMaxKth(const int &depth, const int &rt, const int &m, const Point &d) {
 ↪  // 求 K 远点
  Result tmp = Result(sqrdist(tree[rt].p, d), tree[rt].p);
  if ((int)heap.size() < m) {
    heap.push(tmp);
  } else if (tmp > heap.top()) {
    heap.pop();
    heap.push(tmp);
  }
  int x = tree[rt].l, y = tree[rt].r;
  if (x != 0 && y != 0 && sqrdist(d, tree[x].p) < sqrdist(d, tree[y].p))
 ↪  std::swap(x, y);
```

```
  if (x != 0 && ((int)heap.size() < m || tree[x].getMaxDist(d) >=
  ↪  heap.top().dist)) { // 这里的 >= 是因为在距离相等的时候需要按照 id 排序
    getMaxKth((depth + 1) % k, x, m, d);
  }
  if (y != 0 && ((int)heap.size() < m || tree[y].getMaxDist(d) >=
  ↪  heap.top().dist)) {
    getMaxKth((depth + 1) % k, y, m, d);
  }
}
```

杂项算法

Dancing Links

```
// 精确覆盖
const int MAXD = 1120;
const int MAXN = 1000200;
int n, m, t, size;
int U[MAXN], D[MAXN], L[MAXN], R[MAXN], C[MAXN], Row[MAXN];
int H[MAXD], S[MAXD];
void init(int n, int m) {
  for(int i = 0; i <= m; ++i) {
    S[i] = 0, D[i] = U[i] = i;
    L[i+1] = i, R[i] = i + 1;
  }
  R[m] = 0, size = m;
  for(int i = 1; i <= n; ++i) H[i] = -1;
}
void link(int r, int c) {
  ++S[C[++size] = c];
  Row[size] = r;
  D[size] = D[c], U[D[c]] = size;
  U[size] = c, D[c] = size;
  if(H[r] < 0) H[r] = L[size] = R[size] = size;
  else {
    R[size] = R[H[r]], L[R[size]] = size;
    L[size] = H[r];
    R[H[r]] = size;
  }
}
void remove(int c) {
  R[L[c]] = R[c], L[R[c]] = L[c];
  for(int i = D[c]; i != c; i = D[i])
    for(int j = R[i]; j != i; j = R[j])
      U[D[j]] = U[j], D[U[j]] = D[j], -- S[C[j]];
}
void resume(int c) {
  R[L[c]] = L[R[c]] = c;
  for(int i = U[c]; i != c; i = U[i])
    for(int j = L[i]; j != i; j = L[j])
      U[D[j]] = D[U[j]] = j, ++S[C[j]];
}
int ans[MAXD], cnt;
bool dance(int k) {
  int i, j, tmp, c;
  if( !R[0] ) return 1;
  for(tmp = MAXD, i = R[0]; i; i = R[i])
    if(S[i] < tmp) tmp = S[c = i];
  remove(c);
  for(i = D[c]; i != c; i = D[i]) {
    ans[cnt++] = Row[i]; //用栈记录解
    for(j = R[i]; j != i; j = R[j]) remove(C[j]);
    if(dance(k + 1)) return 1;
    --cnt;
    for(j = L[i]; j != i; j = L[j]) resume(C[j]);
  }
  resume(c);
  return 0;
}
// 可重复覆盖
const int mxm = 15 * 15 + 10;
const int MAXD = 15 * 15 + 10;
const int MAXDode = MAXD * mxm;
const int INF = 0x3f3f3f3f;
//能不加的行尽量不加， 减少搜索时间
int size;
int U[MAXDode], D[MAXDode], R[MAXDode], L[MAXDode], Row[MAXDode],  Col[MAXDode];
int H[MAXD], S[mxm];
int ansd;
void init(int n, int m) {
  int i;
  for(i = 0; i <= m; ++i) {
    S[i] = 0, U[i] = D[i] = i;
    L[i] = i - 1, R[i] = i + 1;
  }
  R[m] = 0, L[0] = m, size = m;
  for(i = 1; i <= n; ++i) H[i] = -1;
}
void link(int r, int c) {
  ++S[Col[++size] = c];
  Row[size] = r; D[size] = D[c]; U[D[c]] = size; U[size] = c; D[c] = size;
  if(H[r] < 0) H[r] = L[size] = R[size] = size;
  else {
    R[size] = R[H[r]]; L[R[H[r]]] = size;
    L[size] = H[r]; R[H[r]] = size;
  }
}
void remove(int c) {
  for(int i = D[c]; i != c; i = D[i])
    L[R[i]] = L[i], R[L[i]] = R[i];
}
void resume(int c) {
  for(int i = U[c]; i != c; i = U[i])
    L[R[i]] = R[L[i]] = i;
}
bool vv[mxm];
int f() {
  int ret = 0, c, i, j;
  for(c = R[0]; c != 0; c = R[c]) vv[c] = 1;
  for(c = R[0]; c != 0; c = R[c])
    if(vv[c]) {
      ++ret, vv[c] = 0;
      for(i = D[c]; i != c; i = D[i])
        for(j = R[i]; j != i; j = R[j])
          vv[Col[j]] = 0;
    }
  return ret;
}
void dance(int d) {
  if(d + f() >= ansd) return;
  if(R[0] == 0) {
    if(d < ansd) ansd = d;
    return;
  }
  int c = R[0], i, j;
  for(i = R[0]; i; i = R[i])
```

```
    if(S[i] < S[c]) c = i;
  for(i = D[c]; i != c; i = D[i]) {
    remove(i);
    for(j = R[i]; j != i; j = R[j]) remove(j);
    dance(d + 1);
    for(j = L[i]; j != i; j = L[j]) resume(j);
    resume(i);
  }
}
```

日期公式

```
int zeller(int y, int m, int d) { // y 年 m 月 d 日是星期几
  if (m <= 2) y--, m += 12; int c = y / 100; y %= 100;
  int w = ((c >> 2) - (c << 1) + y + (y >> 2) + (13 * (m + 1) / 5) + d - 1) % 7;
  if (w < 0) w += 7; return w;
}
int getId(int y, int m, int d) { // y 年 m 月 d 日的日期编号
  if (m < 3) {y--; m += 12;}
  return 365 * y + y / 4 - y / 100 + y / 400 + (153 * m + 2) / 5 + d;
}
```

经纬度球面距离

```
double sphereDis(double lon1, double lat1, double lon2, double lat2, double R) {
  return R*acos(cos(lat1)*cos(lat2)*cos(lon1-lon2)+sin(lat1)*sin(lat2));
}
```

其他

Java Hints

```
import java.util.*;
import java.math.*;
import java.io.*;
public class Main{
  static class Task{
    void solve(int testId, InputReader cin, PrintWriter cout) {
      // Write down the code you want
    }
  };
  public static void main(String args[]) {
    InputStream inputStream = System.in;
    OutputStream outputStream = System.out;
    InputReader in = new InputReader(inputStream);
    PrintWriter out = new PrintWriter(outputStream);
    TaskA solver = new TaskA();
    solver.solve(1, in, out);
    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());
    }
  }
};
// Arrays
int a[];
.fill(a[,int fromIndex,int toIndex],val);|.sort(a[, int fromIndex, int toIndex])
// String
String s;
.charAt(int i);|compareTo(String)|compareToIgnoreCase ()|contains(String)|
length ()|substring(int l, int len)
// BigInteger
.abs()|.add()|bitLength()|subtract()|divide()|remainder()|divideAndRemainder()|
modPow(b, c)|pow(int) | multiply () | compareTo () |
gcd() | intValue () | longValue () | isProbablePrime(int c) (1 - 1/2^c) |
nextProbablePrime () | shiftLeft(int) | valueOf ()
// BigDecimal
.ROUND_CEILING | ROUND_DOWN_FLOOR | ROUND_HALF_DOWN | ROUND_HALF_EVEN |
 ↪  ROUND_HALF_UP | ROUND_UP
.divide(BigDecimal b, int scale , int round_mode) | doubleValue () |
 ↪  movePointLeft(int) | pow(int) |
setScale(int scale , int round_mode) | stripTrailingZeros ()
// StringBuilder
StringBuilder sb = new StringBuilder ();
sb.append(elem) | out.println(sb)
// TODO Java STL 的使用方法以及上面这些方法的检验
```

vimrc

```
set ruler
set number
set smartindent
set autoindent
set tabstop=4
set softtabstop=4
set shiftwidth=4
set hlsearch
set incsearch
set autoread
set backspace=2
set mouse=a
syntax on
nmap <C-A> ggVG
vmap <C-C> "+y
filetype plugin indent on
autocmd FileType cpp set cindent
autocmd FileType cpp map <F9> :w <CR> :!g++ % -o %< -g -std=c++11 -Wall -Wextra
 ↪  -Wconversion && size %< <CR>
autocmd FileType cpp map <C-F9> :!g++ % -o %< -std=c++11 -O2 && size %< <CR>
autocmd FileType cpp map <F8> :!time ./%< < %<.in <CR>
autocmd FileType cpp map <F5> :!time ./%< <CR>
map <F3> :vnew %<.in <CR>
map <F4> :!gedit % <CR>
```

常用结论

上下界网络流

$B(u,v)$ 表示边 (u,v) 流量的下界，$C(u,v)$ 表示边 (u,v) 流量的上界，$F(u,v)$ 表示边 (u,v) 的流量。设 $G(u,v) = F(u,v) - B(u,v)$，显然有：$0 \le G(u,v) \le C(u,v) - B(u,v)$

无源汇的上下界可行流

建立超级源点 S^* 和超级汇点 T^*，对于原图每条边 (u,v) 在新网络中连如下三条边：$S^* \to v$，容量为 $B(u,v)$；$u \to T^*$，容量为 $B(u,v)$；$u \to v$，容量为 $C(u,v) - B(u,v)$。最后求新网络的最大流，判断从超级源点 S^* 出发的边是否都满流即可，边 (u,v) 的最终解中的实际流量为 $G(u,v) + B(u,v)$。

有源汇的上下界可行流

从汇点 T 到源点 S 连一条上界为 ∞，下界为 0 的边。按照**无源汇的上下界可行流**一样做即可，流量即为 $T \to S$ 边上的流量。

有源汇的上下界最大流

1. 在**有源汇的上下界可行流**中，从汇点 T 到源点 S 的边改为连一条上界为 ∞，下届为 x 的边。x 满足二分性质，找到最大的 x 使得新网络存在**无源汇的上下界可行流**即为原图的最大流。
2. 从汇点 T 到源点 S 连一条上界为 ∞，下界为 0 的边，变成无源汇的网络。按照**无源汇的上下界可行流**的方法，建立超级源点 S^* 和超级汇点 T^*，求一遍 $S^* \to T^*$ 的最大流，再将从汇点 T 到源点 S 的这条边拆掉，求一次 $S \to T$ 的最大流即可。

有源汇的上下界最小流

1. 在**有源汇的上下界可行流**中，从汇点 T 到源点 S 的边改为连一条上界为 x，下界为 0 的边。x 满足二分性质，找到最小的 x 使得新网络存在**无源汇的上下界可行流**即为原图的最小流。
2. 按照**无源汇的上下界可行流**的方法，建立超级源点 S^* 与超级汇点 T^*，求一遍 $S^* \to T^*$ 的最大流，但是注意这一次不加上汇点 T 到源点 S 的这条边，即不使之改为无源汇的网络去求解。求完后，再加上那条汇点 T 到源点 S 上界 ∞ 的边。因为这条边下界为 0，所以 S^*，T^* 无影响，再直接求一次 $S^* \to T^*$ 的最大流。若超级源点 S^* 出发的边全部满流，则 $T \to S$ 边上的流量即为原图的最小流，否则无解。

上下界费用流

来源：BZOJ 3876 设汇 t，源 s，超级源 S，超级汇 T，本质是每条边的下界为 1，上界为 MAX，跑一遍有源汇的上下界最小费用最小流。（因为上界无穷大，所以只要满足所有下界的最小费用最小流）

1. 对每个点 x：从 x 到 t 连一条费用为 0，流量为 MAX 的边，表示可以任意停止当前的剧情（接下来的剧情从更优的路径去走，画个样例就知道了）
2. 对于每一条边权为 z 的边 $x \to y$：
   - 从 S 到 y 连一条流量为 1，费用为 z 的边，代表这条边至少要被走一次。
   - 从 x 到 y 连一条流量为 MAX，费用为 z 的边，代表这条边除了至少走的一次之外还可以随便走。
   - 从 x 到 T 连一条流量为 1，费用为 0 的边。（注意是每一条 $x->y$ 的边都连，或者你可以记下 x 的出边数 K_x，连一次流量为 K_x，费用为 0 的边）。

建完图后从 S 到 T 跑一遍费用流，即可。（当前跑出来的就是满足上下界的最小费用最小流了）

弦图相关

1. 团数 $\leq$ 色数, 弦图团数 $=$ 色数
2. 设 $next(v)$ 表示 $N(v)$ 中最前的点. 令 w* 表示所有满足 $A \in B$ 的 w 中最后的一个点, 判断 $v \cup N(v)$ 是否为极大团, 只需判断是否存在一个 w, 满足 $Next(w) = v$ 且 $|N(v)| + 1 \leq |N(w)|$ 即可.
3. 最小染色: 完美消除序列从后往前依次给每个点染色, 给每个点染上可以染的最小的颜色
4. 最大独立集: 完美消除序列从前往后能选就选
5. 弦图最大独立集数 $=$ 最小团覆盖数, 最小团覆盖: 设最大独立集为 $\{p_1, p_2, \ldots, p_t\}$，则 $\{p_1 \cup N(p_1), \ldots, p_t \cup N(p_t)\}$ 为最小团覆盖

Bernoulli 数

1. 初始化：$B_0(n) = 1$
2. 递推公式：$B_m(n) = n^m - \sum_{k=0}^{m-1} \binom{m}{k} \frac{B_k(n)}{m-k+1}$
3. 应用：$\sum_{k=1}^{n} k^m = \frac{1}{m+1} \sum_{k=0}^{m} \binom{m+1}{k} n^{m+1-k}$

常见错误

1. 数组或者变量类型开错，例如将 double 开成 int；
2. 函数忘记返回返回值；
3. 初始化数组没有初始化完全；
4. 对空间限制判断不足导致 MLE；

博弈游戏

巴什博奕

1. 只有一堆 n 个物品，两个人轮流从这堆物品中取物，规定每次至少取一个，最多取 m 个。最后取光者得胜。
2. 显然，如果 $n = m + 1$，那么由于一次最多只能取 m 个，所以，无论先取者拿走多少个，后取者都能够一次拿走剩余的物品，后者取胜。因此我们发现了如何取胜的法则：如果 $n = m + 1\ r + s$，(r 为任意自然数，$s \leq m$), 那么先取者要拿走 s 个物品，如果后取者拿走 $k(k \leq m)$ 个，那么先取者再拿走 $m + 1 - k$ 个，结果剩下 $(m + 1)(r - 1)$ 个，以后保持这样的取法，那么先取者肯定获胜。总之，要保持给对手留下 $(m + 1)$ 的倍数，就能最后获胜。

威佐夫博弈

1. 有两堆各若干个物品，两个人轮流从某一堆或同时从两堆中取同样多的物品，规定每次至少取一个，多者不限，最后取光者得胜。
2. 判断一个局势 (a, b) 为奇异局势（必败态）的方法：$a_k = [k(1 + \sqrt{5})/2]\ b_k = a_k + k$

阶梯博奕

1. 博弈在一列阶梯上进行，每个阶梯上放着自然数个点，两个人进行阶梯博弈，每一步则是将一个阶梯上的若干个点（至少一个）移到前面去，最后没有点可以移动的人输。
2. 解决方法：把所有奇数阶梯看成 N 堆石子，做 NIM。（把石子从奇数堆移动到偶数堆可以理解为拿走石子，就相当于几个奇数堆的石子在做 Nim）

图上删边游戏

链的删边游戏

1. 游戏规则：对于一条链，其中一个端点是根，两人轮流删边，脱离根的部分也算被删去，最后没边可删的人输。
2. 做法：$sg[i] = n - dist(i) - 1$（其中 n 表示总点数，$dist(i)$ 表示离根的距离）

树的删边游戏

1. 游戏规则：对于一棵有根树，两人轮流删边，脱离根的部分也算被删去，没边可删的人输。
2. 做法：叶子结点的 $sg = 0$，其他节点的 sg 等于儿子结点的 $sg + 1$ 的异或和。

局部连通图的删边游戏

1. 游戏规则：在一个局部连通图上，两人轮流删边，脱离根的部分也算被删去，没边可删的人输。局部连通图的构图规则是，在一棵基础树上加边得到，所有形成的环保证不共用边，且只与基础树有一个公共点。
2. 做法：去掉所有的偶环，将所有的奇环变为长度为 1 的链，然后做树的删边游戏。

常用数学公式

求和公式

1. $\sum_{k=1}^{n} (2k-1)^2 = \frac{n(4n^2-1)}{3}$
2. $\sum_{k=1}^{n} k^3 = [\frac{n(n+1)}{2}]^2$
3. $\sum_{k=1}^{n} (2k-1)^3 = n^2(2n^2-1)$
4. $\sum_{k=1}^{n} k^4 = \frac{n(n+1)(2n+1)(3n^2+3n-1)}{30}$
5. $\sum_{k=1}^{n} k^5 = \frac{n^2(n+1)^2(2n^2+2n-1)}{12}$
6. $\sum_{k=1}^{n} k(k+1) = \frac{n(n+1)(n+2)}{3}$
7. $\sum_{k=1}^{n} k(k+1)(k+2) = \frac{n(n+1)(n+2)(n+3)}{4}$
8. $\sum_{k=1}^{n} k(k+1)(k+2)(k+3) = \frac{n(n+1)(n+2)(n+3)(n+4)}{5}$
9. $\frac{1}{(1-x)^{n+1}} = \sum_{i=0}^{n} \binom{i+n}{i} x^i$
10. $\frac{1}{\sqrt{1-4x}} = \sum_{i=0}^{n} \binom{2i}{i} x^i$

斐波那契数列

1. $fib_0 = 0, fib_1 = 1, fib_n = fib_{n-1} + fib_{n-2}$
2. $fib_{n+2} \cdot fib_n - fib_{n+1}^2 = (-1)^{n+1}$
3. $fib_{-n} = (-1)^{n-1} fib_n$
4. $fib_{n+k} = fib_k \cdot fib_{n+1} + fib_{k-1} \cdot fib_n$
5. $gcd(fib_m, fib_n) = fib_{gcd(m,n)}$
6. $fib_m | fib_n^2 \Leftrightarrow n | fib_n | m$

错排公式

$$D_n = (n-1)(D_{n-2} - D_{n-1}) == n! \cdot (1 - \frac{1}{1!} + \frac{1}{2!} - \frac{1}{3!} + \ldots + \frac{(-1)^n}{n!})$$

莫比乌斯函数

$$\mu(n) = \begin{cases} 1 & 若n = 1 \\ (-1)^k & 若n无平方数因子，且n = p_1 p_2 \ldots p_k \\ 0 & 若n有大于1的平方数因数 \end{cases}$$

$$\sum_{d|n} \mu(d) = \begin{cases} 1 & 若n = 1 \\ 0 & 其他情况 \end{cases}$$

$$g(n) = \sum_{d|n} f(d) \Leftrightarrow f(n) = \sum_{d|n} \mu(d) g(\frac{n}{d})$$

$$g(x) = \sum_{n=1}^{[x]} f(\frac{x}{n}) \Leftrightarrow f(x) = \sum_{n=1}^{[x]} \mu(n) g(\frac{x}{n})$$

Burnside 引理

设 G 是一个有限群，作用在集合 X 上。对每个 g 属于 G，令 X^g 表示 X 中在 g 作用下的不动元素，轨道数（记作 $|X/G|$）为 $|X/G| = \frac{1}{|G|} \sum_{g \in G} |X^g|$.

五边形数定理

设 $p(n)$ 是 n 的拆分数，有 $p(n) = \sum_{k \in \mathbb{Z} \setminus \{0\}} (-1)^{k-1} p\left(n - \frac{k(3k-1)}{2}\right)$

树的计数

1. 有根树计数：$n+1$ 个结点的有根树的个数为 $a_{n+1} = \frac{\sum_{j=1}^{n} j \cdot a_j \cdot S_{n,j}}{n}$，其中，$S_{n,j} = \sum_{i=1}^{n/j} a_{n+1-ij} = S_{n-j,j} + a_{n+1-j}$
2. 无根树计数：当 n 为奇数时，n 个结点的无根树的个数为 $a_n - \sum_{i=1}^{n/2} a_i a_{n-i}$，当 n 为偶数时，n 个结点的无根树的个数为 $a_n - \sum_{i=1}^{n/2} a_i a_{n-i} + \frac{1}{2} a_{\frac{n}{2}} (a_{\frac{n}{2}} + 1)$
3. n 个结点的完全图的生成树个数为：n^{n-2}
4. 矩阵 - 树定理：图 G 由 n 个结点构成，设 $\boldsymbol{A}[G]$ 为图 G 的邻接矩阵、$\boldsymbol{D}[G]$ 为图 G 的度数矩阵，则图 G 的不同生成树的个数为 $\boldsymbol{C}[G] = \boldsymbol{D}[G] - \boldsymbol{A}[G]$ 的任意一个 $n-1$ 阶主子式的行列式值。

欧拉公式

平面图的顶点个数、边数和面的个数有如下关系：$V - E + F = C + 1$

其中，V 是顶点的数目，E 是边的数目，F 是面的数目，C 是组成图形的连通部分的数目。当图是单连通图的时候，公式简化为：$V - E + F = 2$

皮克定理

给定顶点坐标均是整点（或正方形格点）的简单多边形，其面积 A 和内部格点数目 i、边上格点数目 b 的关系：$A = i + \frac{b}{2} - 1$

牛顿恒等式

设

$$\prod_{i=1}^{n} (x - x_i) = a_n + a_{n-1} x + \cdots + a_1 x^{n-1} + a_0 x^n$$

$$p_k = \sum_{i=1}^{n} x_i^k$$

则

$$a_0 p_k + a_1 p_{k-1} + \cdots + a_{k-1} p_1 + k a_k = 0$$

特别地，对于

$$|\boldsymbol{A} - \lambda \boldsymbol{E}| = (-1)^n (a_n + a_{n-1} \lambda + \cdots + a_1 \lambda^{n-1} + a_0 \lambda^n)$$

有

$$p_k = \mathrm{Tr}(\boldsymbol{A}^k)$$

平面几何公式

三角形

1. 面积：$S = \frac{a \cdot H_a}{2} = \frac{ab \cdot sinC}{2} = \sqrt{p(p-a)(p-b)(p-c)} \left(\frac{a+b+c}{2}\right)$
2. 中线：$M_a = \frac{\sqrt{2(b^2+c^2)-a^2}}{2} = \frac{\sqrt{b^2+c^2+2bc \cdot cosA}}{2}$
3. 角平分线：$T_a = \frac{\sqrt{bc \cdot [(b+c)^2 - a^2]}}{b+c} = \frac{2bc}{b+c} cos\frac{A}{2}$
4. 高线：$H_a = bsinC = csinB = \sqrt{b^2 - (\frac{a^2+b^2-c^2}{2a})^2}$
5. 内切圆半径

$$r = \frac{S}{p} = \frac{arcsin\frac{B}{2} \cdot sin\frac{C}{2}}{sin\frac{B+C}{2}} = 4R \cdot sin\frac{A}{2} sin\frac{B}{2} sin\frac{C}{2}$$

$$= \sqrt{\frac{(p-a)(p-b)(p-c)}{p}} = p \cdot tan\frac{A}{2} tan\frac{B}{2} tan\frac{C}{2}$$

6. 外接圆半径：$R = \frac{abc}{4S} = \frac{a}{2sinA} = \frac{b}{2sinB} = \frac{c}{2sinC}$

四边形

D_1, D_2 为对角线，M 对角线中点连线，A 为对角线夹角，p 为半周长

1. $a^2 + b^2 + c^2 + d^2 = D_1^2 + D_2^2 + 4M^2$
2. $S = \frac{1}{2} D_1 D_2 sinA$
3. 对于圆内接四边形：$ac + bd = D_1 D_2$
4. 对于圆内接四边形：$S = \sqrt{(p-a)(p-b)(p-c)(p-d)}$

正 n 边形

R 为外接圆半径，r 为内切圆半径

1. 中心角：$A = \frac{2\pi}{n}$
2. 内角：$C = \frac{n-2}{n}\pi$
3. 边长：$a = 2\sqrt{R^2 - r^2} = 2R \cdot sin\frac{A}{2} = 2r \cdot tan\frac{A}{2}$
4. 面积：$S = \frac{nar}{2} = nr^2 \cdot tan\frac{A}{2} = \frac{nR^2}{2} \cdot sinA = \frac{na^2}{4 \cdot tan\frac{A}{2}}$

圆

1. 弧长：$l = rA$
2. 弦长：$a = 2\sqrt{2hr - h^2} = 2r \cdot sin\frac{A}{2}$
3. 弓形高：$h = r - \sqrt{r^2 - \frac{a^2}{4}} = r(1 - cos\frac{A}{2}) = \frac{1}{2} \cdot arctan\frac{A}{4}$
4. 扇形面积：$S_1 = \frac{rl}{2} = \frac{r^2 A}{2}$
5. 弓形面积：$S_2 = \frac{rl - a(r-h)}{2} = \frac{r^2}{2}(A - sinA)$

棱柱

1. 体积（A 为底面积，h 为高）：$V = Ah$
2. 侧面积（l 为棱长，p 为直截面周长）：$S = lp$
3. 全面积：$T = S + 2A$

棱锥

1. 体积（A 为底面积，h 为高）：$V = Ah$
2. 正棱锥侧面积（l 为棱长，p 为直截面周长）：$S = lp$
3. 正棱锥全面积：$T = S + 2A$

棱台

1. 体积（A_1, A_2 为上下底面积，h 为高）：$V = (A_1 + A_2 + \sqrt{A_1 A_2}) \cdot \dfrac{h}{3}$
2. 正棱台侧面积（p_1, p_2 为上下底面周长，l 为斜高）：$S = \dfrac{p_1 + p_2}{2} l$
3. 正棱台全面积：$T = S + A_1 + A_2$

圆柱

1. 侧面积：$S = 2\pi rh$
2. 全面积：$T = 2\pi r(h + r)$
3. 体积：$V = \pi r^2 h$

圆锥

1. 母线：$l = \sqrt{h^2 + r^2}$
2. 侧面积：$S = \pi r l$
3. 全面积：$T = \pi r(l + r)$
4. 体积：$V = \dfrac{\pi}{3} r^2 h$

圆台

1. 母线：$l = \sqrt{h^2 + (r_1 - r_2)^2}$
2. 侧面积：$S = \pi(r_1 + r_2)l$
3. 全面积：$T = \pi r_1(l + r_1) + \pi r_2(l + r_2)$
4. 体积：$V = \dfrac{\pi}{3}(r_1^2 + r_2^2 + r_1 r_2)h$

球

1. 全面积：$T = 4\pi r^2$
2. 体积：$V = \dfrac{4}{3}\pi r^3$

球台

1. 侧面积：$S = 2\pi rh$
2. 全面积：$T = \pi(2rh + r_1^2 + r_2^2)$
3. 体积：$V = \dfrac{\pi h[3(r_1^2 + r_2^2) + h^2]}{6}$

球扇形

1. 全面积（h 为球冠高，r_0 为球冠底面半径）：$T = \pi r(2h + r_0)$
2. 体积：$V = \dfrac{2}{3}\pi r^2 h$

立体几何公式

球面三角公式

设 a, b, c 是边长，A, B, C 是所对的二面角，有余弦定理

$$cosa = cosb \cdot cosc + sinb \cdot sinc \cdot cosA$$

正弦定理

$$\frac{sinA}{sina} = \frac{sinB}{sinb} = \frac{sinC}{sinc}$$

三角形面积是 $A + B + C - \pi$

四面体体积公式

U, V, W, u, v, w 是四面体的 6 条棱，U, V, W 构成三角形，$(U, u), (V, v), (W, w)$ 互为对棱，则

$$V = \frac{\sqrt{(s-2a)(s-2b)(s-2c)(s-2d)}}{192uvw}$$

其中

$$\begin{cases} a &= \sqrt{xYZ}, \\ b &= \sqrt{yZX}, \\ c &= \sqrt{zXY}, \\ d &= \sqrt{xyz}, \\ s &= a+b+c+d, \\ X &= (w-U+v)(U+v+w), \\ x &= (U-v+w)(v-w+U), \\ Y &= (u-V+w)(V+w+u), \\ y &= (V-w+u)(w-u+V), \\ Z &= (v-W+u)(W+u+v), \\ z &= (W-u+v)(u-v+W) \end{cases}$$

附录

NTT 素数及原根列表

Id	Primes	PRT	Id	Primes	PRT	Id	Primes	PRT
1	7340033	3	38	311427073	7	75	786432001	7
2	13631489	15	39	330301441	22	76	799014913	13
3	23068673	3	40	347078657	3	77	800063489	3
4	26214401	3	41	359661569	3	78	802160641	11
5	28311553	5	42	361758721	29	79	818937857	5
6	69206017	5	43	377487361	7	80	824180737	5
7	70254593	3	44	383778817	5	81	833617921	13
8	81788929	7	45	387973121	6	82	850395137	3
9	101711873	3	46	399507457	5	83	862978049	3
10	104857601	3	47	409993217	3	84	880803841	26
11	111149057	3	48	415236097	5	85	883949569	7
12	113246209	7	49	447741953	3	86	897581057	3
13	120586241	6	50	459276289	11	87	899678209	7
14	132120577	5	51	463470593	3	88	907018241	3
15	136314881	3	52	468713473	5	89	913309697	3
16	138412033	5	53	469762049	3	90	918552577	5
17	141557761	26	54	493879297	10	91	919601153	3
18	147849217	5	55	531628033	5	92	924844033	5
19	155189249	6	56	576716801	6	93	925892609	3
20	158334977	3	57	581959681	11	94	935329793	3
21	163577857	23	58	595591169	3	95	938475521	3
22	167772161	3	59	597688321	11	96	940572673	7
23	169869313	5	60	605028353	3	97	943718401	7
24	185597953	5	61	635437057	11	98	950009857	7
25	186646529	3	62	639631361	6	99	957349889	6
26	199229441	3	63	645922817	3	100	962592769	7
27	204472321	19	64	648019969	17	101	972029953	10
28	211812353	3	65	655360001	3	102	975175681	17
29	221249537	3	66	666894337	5	103	976224257	3
30	230686721	6	67	683671553	3	104	985661441	3
31	246415361	3	68	710934529	17	105	998244353	3
32	249561089	3	69	715128833	3	106	1004535809	3
33	257949697	5	70	718274561	3	107	1007681537	3
34	270532609	22	71	740294657	3	108	1012924417	5
35	274726913	3	72	745537537	5	109	1045430273	3
36	290455553	3	73	754974721	11	110	1051721729	6
37	305135617	5	74	770703361	11	111	1053818881	7