# Arondight's Standard Code Library*

*Shanghai Jiao Tong University*

Dated: August 1, 2017

*https://www.github.com/footoredo/Arondight

# 1 计算几何

```cpp
int sign(DB x) {
  return (x > eps) - (x < -eps);
}
DB msqrt(DB x) {
  return sign(x) > 0 ? sqrt(x) : 0;
}

struct Point {
  DB x, y;
  Point rotate(DB ang) const {  // 逆时针旋转 ang 弧度
    return Point(cos(ang) * x - sin(ang) * y,
        cos(ang) * y + sin(ang) * x);
  }
  Point turn90() const {  // 逆时针旋转 90 度
    return Point(-y, x);
  }
  Point unit() const {
    return *this / len();
  }
};
DB dot(const Point& a, const Point& b) {
  return a.x * b.x + a.y * b.y;
}
DB det(const Point& a, const Point& b) {
  return a.x * b.y - a.y * b.x;
}
#define cross(p1,p2,p3) ((p2.x-p1.x)*(p3.y-p1.y)-(p3.x-p1.x)*(p2.y-p1.y))
#define crossOp(p1,p2,p3) sign(cross(p1,p2,p3))
bool isLL(const Line& l1, const Line& l2, Point& p) {  // 直线与直线交点
  DB s1 = det(l2.b - l2.a, l1.a - l2.a),
     s2 = -det(l2.b - l2.a, l1.b - l2.a);
  if (!sign(s1 + s2)) return false;
  p = (l1.a * s2 + l1.b * s1) / (s1 + s2);
  return true;
}
bool onSeg(const Line& l, const Point& p) {  // 点在线段上
  return sign(det(p - l.a, l.b - l.a)) == 0 && sign(dot(p - l.a, p - l.b)) <= 0;
}
Point projection(const Line & l, const Point& p) {
  return l.a + (l.b - l.a) * (dot(p - l.a, l.b - l.a) / (l.b - l.a).len2());
}
DB disToLine(const Line& l, const Point& p) {  // 点到 * 直线 * 距离
  return fabs(det(p - l.a, l.b - l.a) / (l.b - l.a).len());
}
DB disToSeg(const Line& l, const Point& p) {  // 点到线段距离
  return sign(dot(p - l.a, l.b - l.a)) * sign(dot(p - l.b, l.a - l.b)) == 1 ?
   ↪ disToLine(l, p) : std::min((p - l.a).len(), (p - l.b).len());
}
// 圆与直线交点
bool isCL(Circle a, Line l, Point& p1, Point& p2) {
  DB x = dot(l.a - a.o, l.b - l.a),
     y = (l.b - l.a).len2(),
     d = x * x - y * ((l.a - a.o).len2() - a.r * a.r);
  if (sign(d) < 0) return false;
  Point p = l.a - ((l.b - l.a) * (x / y)), delta = (l.b - l.a) * (msqrt(d) / y);
  p1 = p + delta; p2 = p - delta;
  return true;
}
//圆与圆的交面积
DB areaCC(const Circle& c1, const Circle& c2) {
  DB d = (c1.o - c2.o).len();
  if (sign(d - (c1.r + c2.r)) >= 0) return 0;
  if (sign(d - std::abs(c1.r - c2.r)) <= 0) {
    DB r = std::min(c1.r, c2.r);
    return r * r * PI;
  }
  DB x = (d * d + c1.r * c1.r - c2.r * c2.r) / (2 * d),
    t1 = acos(x / c1.r), t2 = acos((d - x) / c2.r);
  return c1.r * c1.r * t1 + c2.r * c2.r * t2 - d * c1.r * sin(t1);
}
// 圆与圆交点
bool isCC(Circle a, Circle b, P& p1, P& p2) {
  DB s1 = (a.o - b.o).len();
  if (sign(s1 - a.r - b.r) > 0 || sign(s1 - std::abs(a.r - b.r)) < 0) return false;
  DB s2 = (a.r * a.r - b.r * b.r) / s1;
  DB aa = (s1 + s2) * 0.5, bb = (s1 - s2) * 0.5;
  P o = (b.o - a.o) * (aa / (aa + bb)) + a.o;
  P delta = (b.o - a.o).unit().turn90() * msqrt(a.r * a.r - aa * aa);
  p1 = o + delta, p2 = o - delta;
  return true;
}
// 求点到圆的切点，按关于点的顺时针方向返回两个点
bool tanCP(const Circle &c, const Point &p0, Point &p1, Point &p2) {
  double x = (p0 - c.o).len2(), d = x - c.r * c.r;
  if (d < eps) return false; // 点在圆上认为没有切点
  Point p = (p0 - c.o) * (c.r * c.r / x);
  Point delta = ((p0 - c.o) * (-c.r * sqrt(d) / x)).turn90();
  p1 = c.o + p + delta;
  p2 = c.o + p - delta;
  return true;
}
// 求圆到圆的外共切线，按关于 c1.o 的顺时针方向返回两条线
vector<Line> extanCC(const Circle &c1, const Circle &c2) {
  vector<Line> ret;
  if (sign(c1.r - c2.r) == 0) {
    Point dir = c2.o - c1.o;
    dir = (dir * (c1.r / dir.len())).turn90();
    ret.push_back(Line(c1.o + dir, c2.o + dir));
    ret.push_back(Line(c1.o - dir, c2.o - dir));
  } else {
    Point p = (c1.o * -c2.r + c2.o * c1.r) / (c1.r - c2.r);
    Point p1, p2, q1, q2;
    if (tanCP(c1, p, p1, p2) && tanCP(c2, p, q1, q2)) {
      if (c1.r < c2.r) swap(p1, p2), swap(q1, q2);
      ret.push_back(Line(p1, q1));
      ret.push_back(Line(p2, q2));
    }
```

```
  }
  return ret;
}
// 求圆到圆的内共切线，按关于 c1.o 的顺时针方向返回两条线
std::vector<Line> intanCC(const Circle &c1, const Circle &c2) {
  std::vector<Line> ret;
  Point p = (c1.o * c2.r + c2.o * c1.r) / (c1.r + c2.r);
  Point p1, p2, q1, q2;
  if (tanCP(c1, p, p1, p2) && tanCP(c2, p, q1, q2)) { // 两圆相切认为没有切线
    ret.push_back(Line(p1, q1));
    ret.push_back(Line(p2, q2));
  }
  return ret;
}
bool contain(vector<Point> polygon, Point p) { // 判断点 p 是否被多边形包含，包括落在边界上
  int ret = 0, n = polygon.size();
  for(int i = 0; i < n; ++ i) {
    Point u = polygon[i], v = polygon[(i + 1) % n];
    if (onSeg(Line(u, v), p)) return true;  // Here I guess.
    if (sign(u.y - v.y) <= 0) swap(u, v);
    if (sign(p.y - u.y) > 0 || sign(p.y - v.y) <= 0) continue;
    ret += sign(det(p, v, u)) > 0;
  }
  return ret & 1;
}
// 用半平面 (q1,q2) 的逆时针方向去切凸多边形
std::vector<Point> convexCut(const std::vector<Point>&ps, Point q1, Point q2) {
  std::vector<Point> qs; int n = ps.size();
  for (int i = 0; i < n; ++i) {
    Point p1 = ps[i], p2 = ps[(i + 1) % n];
    int d1 = crossOp(q1,q2,p1), d2 = crossOp(q1,q2,p2);
    if (d1 >= 0) qs.push_back(p1);
    if (d1 * d2 < 0) qs.push_back(isSS(p1, p2, q1, q2));
  }
  return qs;
}
// 求凸包
std::vector<Point> convexHull(std::vector<Point> ps) {
  int n = ps.size(); if (n <= 1) return ps;
  std::sort(ps.begin(), ps.end());
  std::vector<Point> qs;
  for (int i = 0; i < n; qs.push_back(ps[i ++]))
    while (qs.size() > 1 && sign(det(qs[qs.size() - 2], qs.back(), ps[i])) <= 0)
      qs.pop_back();
  for (int i = n - 2, t = qs.size(); i >= 0; qs.push_back(ps[i --]))
    while ((int)qs.size() > t && sign(det(qs[qs.size() - 2], qs.back(), ps[i])) <= 0)
      qs.pop_back();
  return qs;
```

## 1.1 凸包

```
// 凸包中的点按逆时针方向
struct Convex {
  int n;
  std::vector<Point> a, upper, lower;
  void make_shell(const std::vector<Point>& p,
      std::vector<Point>& shell) {  // p needs to be sorted.
    clear(shell); int n = p.size();
    for (int i = 0, j = 0; i < n; i++, j++) {
      for (; j >= 2 && sign(det(shell[j-1] - shell[j-2],
              p[i] - shell[j-2])) <= 0; --j) shell.pop_back();
      shell.push_back(p[i]);
    }
  }
  void make_convex() {
    std::sort(a.begin(), a.end());
    make_shell(a, lower);
    std::reverse(a.begin(), a.end());
    make_shell(a, upper);
    a = lower; a.pop_back();
    a.insert(a.end(), upper.begin(), upper.end());
    if ((int)a.size() >= 2) a.pop_back();
    n = a.size();
  }
  void init(const std::vector<Point>& _a) {
    clear(a); a = _a; n = a.size();
    make_convex();
  }
  void read(int _n) {  // Won't make convex.
    clear(a); n = _n; a.resize(n);
    for (int i = 0; i < n; i++)
      a[i].read();
  }
  std::pair<DB, int> get_tangent(
      const std::vector<Point>& convex, const Point& vec) {
    int l = 0, r = (int)convex.size() - 2;
    assert(r >= 0);
    for (; l + 1 < r; ) {
      int mid = (l + r) / 2;
      if (sign(det(convex[mid + 1] - convex[mid], vec)) > 0)
        r = mid;
      else l = mid;
    }
    return std::max(std::make_pair(det(vec, convex[r]), r),
        std::make_pair(det(vec, convex[0]), 0));
  }
  int binary_search(Point u, Point v, int l, int r) {
    int s1 = sign(det(v - u, a[l % n] - u));
    for (; l + 1 < r; ) {
      int mid = (l + r) / 2;
      int smid = sign(det(v - u, a[mid % n] - u));
      if (smid == s1) l = mid;
      else r = mid;
    }
    return l % n;
  }
  // 求凸包上和向量 vec 叉积最大的点，返回编号，共线的多个切点返回任意一个
  int get_tangent(Point vec) {
    std::pair<DB, int> ret = get_tangent(upper, vec);
    ret.second = (ret.second + (int)lower.size() - 1) % n;
    ret = std::max(ret, get_tangent(lower, vec));
```

```
    return ret.second;
  }
  // 求凸包和直线 u, v 的交点，如果不相交返回 false，如果有则是和 (i, next(i)) 的交点，交在点上不确定返回前后两条边其中之一
  bool get_intersection(Point u, Point v, int &i0, int &i1) {
    int p0 = get_tangent(u - v), p1 = get_tangent(v - u);
    if (sign(det(v - u, a[p0] - u)) * sign(det(v - u, a[p1] - u)) <= 0) {
      if (p0 > p1) std::swap(p0, p1);
      i0 = binary_search(u, v, p0, p1);
      i1 = binary_search(u, v, p1, p0 + n);
      return true;
    }
    else return false;
  }
};
```

### 1.2 三角形的心

```
Point inCenter(const Point &A, const Point &B, const Point &C) { // 内心
  double a = (B - C).len(), b = (C - A).len(), c = (A - B).len(),
    s = fabs(det(B - A, C - A)),
    r = s / p;
  return (A * a + B * b + C * c) / (a + b + c);
}
Point circumCenter(const Point &a, const Point &b, const Point &c) { // 外心
  Point bb = b - a, cc = c - a;
  double db = bb.len2(), dc = cc.len2(), d = 2 * det(bb, cc);
  return a - Point(bb.y * dc - cc.y * db, cc.x * db - bb.x * dc) / d;
}
Point othroCenter(const Point &a, const Point &b, const Point &c) { // 垂心
  Point ba = b - a, ca = c - a, bc = b - c;
  double Y = ba.y * ca.y * bc.y,
       A = ca.x * ba.y - ba.x * ca.y,
       x0 = (Y + ca.x * ba.y * b.x - ba.x * ca.y * c.x) / A,
       y0 = -ba.x * (x0 - c.x) / ba.y + ca.y;
  return Point(x0, y0);
}
```

### 1.3 半平面交

```
struct Point {
  int quad() const { return sign(y) == 1 || (sign(y) == 0 && sign(x) >= 0);}
};
struct Line {
  bool include(const Point &p) const { return sign(det(b - a, p - a)) > 0; }
  Line push() const{ // 将半平面向外推 eps
    const double eps = 1e-6;
    Point delta = (b - a).turn90().norm() * eps;
    return Line(a - delta, b - delta);
  }
};
bool sameDir(const Line &l0, const Line &l1) { return parallel(l0, l1) && sign(dot(l0.b - l0.a, l1.b - l1.a)) == 1; }
bool operator < (const Point &a, const Point &b) {
  if (a.quad() != b.quad()) {
    return a.quad() < b.quad();
  } else {
    return sign(det(a, b)) > 0;
  }
}
bool operator < (const Line &l0, const Line &l1) {
  if (sameDir(l0, l1)) {
    return l1.include(l0.a);
  } else {
    return (l0.b - l0.a) < (l1.b - l1.a);
  }
}
bool check(const Line &u, const Line &v, const Line &w) { return w.include(intersect(u, v)); }
vector<Point> intersection(vector<Line> &l) {
  sort(l.begin(), l.end());
  deque<Line> q;
  for (int i = 0; i < (int)l.size(); ++i) {
    if (i && sameDir(l[i], l[i - 1])) {
      continue;
    }
    while (q.size() > 1 && !check(q[q.size() - 2], q[q.size() - 1], l[i])) q.pop_back();
    while (q.size() > 1 && !check(q[1], q[0], l[i])) q.pop_front();
    q.push_back(l[i]);
  }
  while (q.size() > 2 && !check(q[q.size() - 2], q[q.size() - 1], q[0])) q.pop_back();
  while (q.size() > 2 && !check(q[1], q[0], q[q.size() - 1])) q.pop_front();
  vector<Point> ret;
  for (int i = 0; i < (int)q.size(); ++i) ret.push_back(intersect(q[i], q[(i + 1) % q.size()]));
  return ret;
}
```

### 1.4 圆交面积及重心

```
struct Event {
  Point p;
  double ang;
  int delta;
  Event (Point p = Point(0, 0), double ang = 0, double delta = 0) : p(p), ang(ang), delta(delta) {}
};
bool operator < (const Event &a, const Event &b) {
  return a.ang < b.ang;
}
void addEvent(const Circle &a, const Circle &b, vector<Event> &evt, int &cnt) {
  double d2 = (a.o - b.o).len2(),
       dRatio = ((a.r - b.r) * (a.r + b.r) / d2 + 1) / 2,
       pRatio = sqrt(-(d2 - sqr(a.r - b.r)) * (d2 - sqr(a.r + b.r)) / (d2 * d2 * 4));
  Point d = b.o - a.o, p = d.rotate(PI / 2),
      q0 = a.o + d * dRatio + p * pRatio,
      q1 = a.o + d * dRatio - p * pRatio;
  double ang0 = (q0 - a.o).ang(),
       ang1 = (q1 - a.o).ang();
  evt.push_back(Event(q1, ang1, 1));
  evt.push_back(Event(q0, ang0, -1));
```

```
  cnt += ang1 > ang0;
}
bool issame(const Circle &a, const Circle &b) { return sign((a.o - b.o).len()) == 0 &&
    ↪ sign(a.r - b.r) == 0; }
bool overlap(const Circle &a, const Circle &b) { return sign(a.r - b.r - (a.o -
    ↪ b.o).len()) >= 0; }
bool intersect(const Circle &a, const Circle &b) { return sign((a.o - b.o).len() - a.r -
    ↪ b.r) < 0; }
Circle c[N];
double area[N];  // area[k] -> area of intersections >= k.
Point centroid[N];
bool keep[N];
void add(int cnt, DB a, Point c) {
  area[cnt] += a;
  centroid[cnt] = centroid[cnt] + c * a;
}
void solve(int C) {
  for (int i = 1; i <= C; ++ i) {
        area[i] = 0;
        centroid[i] = Point(0, 0);
    }
  for (int i = 0; i < C; ++i) {
    int cnt = 1;
    vector<Event> evt;
    for (int j = 0; j < i; ++j) if (issame(c[i], c[j])) ++cnt;
    for (int j = 0; j < C; ++j) {
      if (j != i && !issame(c[i], c[j]) && overlap(c[j], c[i])) {
        ++cnt;
      }
    }
    for (int j = 0; j < C; ++j) {
      if (j != i && !overlap(c[j], c[i]) && !overlap(c[i], c[j]) && intersect(c[i],
    ↪ c[j])) {
        addEvent(c[i], c[j], evt, cnt);
      }
    }
    if (evt.size() == 0u) {
      add(cnt, PI * c[i].r * c[i].r, c[i].o);
    } else {
      sort(evt.begin(), evt.end());
      evt.push_back(evt.front());
      for (int j = 0; j + 1 < (int)evt.size(); ++j) {
        cnt += evt[j].delta;
        add(cnt, det(evt[j].p, evt[j + 1].p) / 2, (evt[j].p + evt[j + 1].p) / 3);
        double ang = evt[j + 1].ang - evt[j].ang;
        if (ang < 0) {
          ang += PI * 2;
        }
                if (sign(ang) == 0) continue;
                add(cnt, ang * c[i].r * c[i].r / 2, c[i].o +
                    Point(sin(ang1) - sin(ang0), -cos(ang1) + cos(ang0)) * (2 / (3 *
    ↪ ang) * c[i].r));
        add(cnt, -sin(ang) * c[i].r * c[i].r / 2, (c[i].o + evt[j].p + evt[j + 1].p) /
    ↪ 3);
      }
    }
    for (int i = 1; i <= C; ++ i)
    if (sign(area[i])) {
      centroid[i] = centroid[i] / area[i];
    }
}
```

### 1.5 三维向量绕轴旋转

```
// 三维绕轴旋转，大拇指指向 axis 向量方向，四指弯曲方向转 w 弧度
Point rotate(const Point& s, const Point& axis, DB w) {
  DB x = axis.x, y = axis.y, z = axis.z;
  DB s1 = x * x + y * y + z * z, ss1 = msqrt(s1),
     cosw = cos(w), sinw = sin(w);
  DB a[4][4];
  memset(a, 0, sizeof a);
  a[3][3] = 1;
  a[0][0] = ((y * y + z * z) * cosw + x * x) / s1;
  a[0][1] = x * y * (1 - cosw) / s1 + z * sinw / ss1;
  a[0][2] = x * z * (1 - cosw) / s1 - y * sinw / ss1;
  a[1][0] = x * y * (1 - cosw) / s1 - z * sinw / ss1;
  a[1][1] = ((x * x + z * z) * cosw + y * y) / s1;
  a[1][2] = y * z * (1 - cosw) / s1 + x * sinw / ss1;
  a[2][0] = x * z * (1 - cosw) / s1 + y * sinw / ss1;
  a[2][1] = y * z * (1 - cosw) / s1 - x * sinw / ss1;
  a[2][2] = ((x * x + y * y) * cos(w) + z * z) / s1;
  DB ans[4] = {0, 0, 0, 0}, c[4] = {s.x, s.y, s.z, 1};
  for (int i = 0; i < 4; ++ i)
    for (int j = 0; j < 4; ++ j)
      ans[i] += a[j][i] * c[j];
  return Point(ans[0], ans[1], ans[2]);
}
```

### 1.6 三维凸包

```
__inline P cross(const P& a, const P& b) {
  return P(
      a.y * b.z - a.z * b.y,
      a.z * b.x - a.x * b.z,
      a.x * b.y - a.y * b.x
        );
}

__inline DB mix(const P& a, const P& b, const P& c) {
  return dot(cross(a, b), c);
}

__inline DB volume(const P& a, const P& b, const P& c, const P& d) {
  return mix(b - a, c - a, d - a);
}

struct Face {
  int a, b, c;
  __inline Face() {}
  __inline Face(int _a, int _b, int _c):
```

```
    a(_a), b(_b), c(_c) {}
  __inline DB area() const {
    return 0.5 * cross(p[b] - p[a], p[c] - p[a]).len();
  }
  __inline P normal() const {
    return cross(p[b] - p[a], p[c] - p[a]).unit();
  }
  __inline DB dis(const P& p0) const {
    return dot(normal(), p0 - p[a]);
  }
};

std::vector<Face> face, tmp;  // Should be O(n).
int mark[N][N], Time, n;

__inline void add(int v) {
  ++ Time;
  clear(tmp);
  for (int i = 0; i < (int)face.size(); ++ i) {
    int a = face[i].a, b = face[i].b, c = face[i].c;
    if (sign(volume(p[v], p[a], p[b], p[c])) > 0) {
      mark[a][b] = mark[b][a] = mark[a][c] =
        mark[c][a] = mark[b][c] = mark[c][b] = Time;
    }
    else {
      tmp.push_back(face[i]);
    }
  }
  clear(face); face = tmp;
  for (int i = 0; i < (int)tmp.size(); ++ i) {
    int a = face[i].a, b = face[i].b, c = face[i].c;
    if (mark[a][b] == Time) face.emplace_back(v, b, a);
    if (mark[b][c] == Time) face.emplace_back(v, c, b);
    if (mark[c][a] == Time) face.emplace_back(v, a, c);
    assert(face.size() < 500u);
  }
}

void reorder() {
  for (int i = 2; i < n; ++ i) {
    P tmp = cross(p[i] - p[0], p[i] - p[1]);
    if (sign(tmp.len())) {
      std::swap(p[i], p[2]);
      for (int j = 3; j < n; ++ j)
        if (sign(volume(p[0], p[1], p[2], p[j]))) {
          std::swap(p[j], p[3]);
          return;
        }
    }
  }
}

void build_convex() {
  reorder();
  clear(face);
  face.emplace_back(0, 1, 2);
  face.emplace_back(0, 2, 1);
  for (int i = 3; i < n; ++ i)
    add(i);
}
```

# 2 数论

## 2.1 $O(m^2 \log n)$ 求线性递推数列第 n 项

Given $a_0, a_1, \ldots, a_{m-1}$
$a_n = c_0 \times a_{n-m} + \cdots + c_{m-1} \times a_{n-1}$
Solve for $a_n = v_0 \times a_0 + v_1 \times a_1 + \cdots + v_{m-1} \times a_{m-1}$

```
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];
  }
```

```
}
```

### 2.2 求逆元

```
void ex_gcd(long long a, long long b, long long &x, long long &y) {
  if (b == 0) {
    x = 1;
    y = 0;
    return;
  }
  long long xx, yy;
  ex_gcd(b, a % b, xx, yy);
  y = xx - a / b * yy;
  x = yy;
}

long long inv(long long x, long long MODN) {
  long long inv_x, y;
  ex_gcd(x, MODN, inv_x, y);
  return (inv_x % MODN + MODN) % MODN;
}
```

### 2.3 中国剩余定理

```
// 返回 (ans, M)，其中 ans 是模 M 意义下的解
std::pair<long long, long long> CRT(const std::vector<long long>& m, const
    ↪ std::vector<long long>& a) {
  long long M = 1, ans = 0;
  int n = m.size();
  for (int i = 0; i < n; i++) M *= m[i];
  for (int i = 0; i < n; i++) {
    ans = (ans + (M / m[i]) * a[i] % M * inv(M / m[i], m[i])) % M;  // 可能需要大整数相乘
    ↪ 取模
  }
  return std::make_pair(ans, M);
}
// 模数不互质的情况
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);  // fix 成正的
    output.first += output.second * number;
    output.second *= input[i].second / divisor;
    fix(output.first, output.second);
  }
  return true;
}
```

### 2.4 魔法 CRT

```
// MOD is the given module
// Do not depend on LL * LL % LL
inline int CRT(int *a) {
  static int x[N];
  for (int i = 0; i < N; i ++) {
    x[i] = a[i];
    for (int j = 0; j < i; j ++) {
      int t = (x[i] - x[j] + mod[i]) % mod[i];
      if (t < 0) t += mod[i];
      x[i] = 1LL * t * Inv[j][i] % mod[i];
    }
  }
  int sum = 1, ret = x[0] % MOD;
  for (int i = 1; i < N; i ++) {
    sum = 1LL * sum * mod[i - 1] % MOD;
    ret += 1LL * x[i] * sum % MOD;
    if (ret >= MOD) ret -= MOD;
  }
  return ret;
}
for (int i = 0; i < N; i ++)
  for (int j = i + 1; j < N; j ++) {
    Inv[i][j] = fpw(mod[i], mod[j] - 2, mod[j]);
  }
```

### 2.5 素性测试

```
int strong_pseudo_primetest(long long n,int base) {
    long long n2=n-1,res;
    int s=0;
    while(n2%2==0) n2>>=1,s++;
    res=powmod(base,n2,n);
    if((res==1)||(res==n-1)) return 1;
    s--;
    while(s>=0) {
        res=mulmod(res,res,n);
        if(res==n-1) return 1;
        s--;
    }
    return 0; // n is not a strong pseudo prime
}
int isprime(long long n) {
  static LL testNum[]={2,3,5,7,11,13,17,19,23,29,31,37};
  static LL lim[]={4,0,1373653LL,25326001LL,25000000000LL,2152302898747LL,
    ↪ 3474749660383LL,341550071728321LL,0,0,0,0};
  if(n<2||n==3215031751LL) return 0;
  for(int i=0;i<12;++i){
    if(n<lim[i]) return 1;
    if(strong_pseudo_primetest(n,testNum[i])==0) return 0;
  }
  return 1;
}
```

### 2.6 质因数分解

```
int ansn; LL ans[1000];
LL func(LL x,LL n){ return(mod_mul(x,x,n)+1)%n; }
LL Pollard(LL n){
  LL i,x,y,p;
  if(Rabin_Miller(n)) return n;
  if(!(n&1)) return 2;
  for(i=1;i<20;i++){
    x=i; y=func(x,n); p=gcd(y-x,n);
    while(p==1) {x=func(x,n); y=func(func(y,n),n); p=gcd((y-x+n)%n,n)%n;}
    if(p==0||p==n) continue;
    return p;
  }
}
void factor(LL n){
  LL x;
  x=Pollard(n);
  if(x==n){ ans[ansn++]=x; return; }
  factor(x), factor(n/x);
}
```

### 2.7 线下整点

```
// ∑_{i=0}^{n-1} ⌊(a+bi)/m⌋, n,m,a,b > 0
LL solve(LL n,LL a,LL b,LL 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);
}
```

### 2.8 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。

## 3 代数

### 3.1 快速傅里叶变换

```
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;
  }
}
```

### 3.2 快速数论变换

```
// 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);
```

### 3.3 自适应辛普森积分

```
namespace adaptive_simpson {
  template<typename function>
  inline double area(function f, const double &left, const double &right) {
    double mid = (left + right) / 2;
    return (right - left) * (f(left) + 4 * f(mid) + f(right)) / 6;
  }

  template<typename function>
  inline double simpson(function f, const double &left, const double &right, const
   ↪ double &eps, const double &area_sum) {
    double mid = (left + right) / 2;
    double area_left = area(f, left, mid);
    double area_right = area(f, mid, right);
    double area_total = area_left + area_right;
    if (fabs(area_total - area_sum) <= 15 * eps) {
      return area_total + (area_total - area_sum) / 15;
    }
    return simpson(f, left, right, eps / 2, area_left) + simpson(f, mid, right, eps / 2,
   ↪ area_right);
  }

  template<typename function>
  inline double simpson(function f, const double &left, const double &right, const
   ↪ double &eps) {
    return simpson(f, left, right, eps, area(f, left, right));
  }
}
```

### 3.4 单纯形

```
const double eps = 1e-8;
// max{c * x | Ax <= b, x >= 0} 的解，无解返回空的 vector，否则就是解.
vector<double> simplex(vector<vector<double> > &A, vector<double> b, vector<double> c) {
  int n = A.size(), m = A[0].size() + 1, r = n, s = m - 1;
  vector<vector<double> > D(n + 2, vector<double>(m + 1));
  vector<int> ix(n + m);
  for(int i = 0; i < n + m; i++)  {
    ix[i] = i;
  }
  for(int i = 0; i < n; i++) {
    for(int j = 0; j < m - 1; j++) {
      D[i][j] = -A[i][j];
    }
    D[i][m - 1] = 1;
    D[i][m] = b[i];
    if (D[r][m] > D[i][m]) {
      r = i;
    }
  }

  for(int j = 0; j < m - 1; j++) {
    D[n][j] = c[j];
  }
  D[n + 1][m - 1] = -1;
  for(double d; ;) {
    if (r < n) {
      swap(ix[s], ix[r + m]);
      D[r][s] = 1. / D[r][s];
      for(int j = 0; j <= m; j++) {
        if (j != s) {
          D[r][j] *= -D[r][s];
        }
      }
      for(int i = 0; i <= n + 1; i++) {
        if (i != r) {
          for(int j = 0; j <= m; j++) {
            if (j != s) {
              D[i][j] += D[r][j] * D[i][s];
            }
          }
          D[i][s] *= D[r][s];
        }
      }
    }
    r = -1, s = -1;
    for(int j = 0; j < m; j++) {
      if (s < 0 || ix[s] > ix[j]) {
        if (D[n + 1][j] > eps || D[n + 1][j] > -eps && D[n][j] > eps) {
          s = j;
        }
      }
    }
    if (s < 0) {
      break;
    }
    for(int i = 0; i < n; i++) {
      if (D[i][s] < -eps) {
        if (r < 0 || (d = D[r][m] / D[r][s] - D[i][m] / D[i][s]) < -eps
```

```
          || d < eps && ix[r + m] > ix[i + m]) {

          r = i;
        }
      }
    }

    if (r < 0) {
      return vector<double> ();
    }
  }
  if (D[n + 1][m] < -eps) {
    return vector<double> ();
  }

  vector<double> x(m - 1);
  for(int i = m; i < n + m; i++) {
    if (ix[i] < m - 1) {
      x[ix[i]] = D[i - m][m];
    }
  }
  return x;
}
```

# 4 字符串

## 4.1 后缀数组

```
const int MAXN = MAXL * 2 + 1;
int a[MAXN], x[MAXN], y[MAXN], c[MAXN], sa[MAXN], rank[MAXN], height[MAXN];
void calc_sa(int n) {
  int m = alphabet, k = 1;
  memset(c, 0, sizeof(*c) * (m + 1));
  for (int i = 1; i <= n; ++i) c[x[i] = a[i]]++;
  for (int i = 1; i <= m; ++i) c[i] += c[i - 1];
  for (int i = n; i; --i) sa[c[x[i]]--] = i;
  for (; k <= n; k <<= 1) {
    int tot = k;
    for (int i = n - k + 1; i <= n; ++i) y[i - n + k] = i;
    for (int i = 1; i <= n; ++i)
      if (sa[i] > k) y[++tot] = sa[i] - k;
    memset(c, 0, sizeof(*c) * (m + 1));
    for (int i = 1; i <= n; ++i) c[x[i]]++;
    for (int i = 1; i <= m; ++i) c[i] += c[i - 1];
    for (int i = n; i; --i) sa[c[x[y[i]]]--] = y[i];
    for (int i = 1; i <= n; ++i) y[i] = x[i];
    tot = 1; x[sa[1]] = 1;
    for (int i = 2; i <= n; ++i) {
      if (max(sa[i], sa[i - 1]) + k > n || y[sa[i]] != y[sa[i - 1]] || y[sa[i] + k] !=
    ↪ y[sa[i - 1] + k]) ++tot;
      x[sa[i]] = tot;
    }
    if (tot == n) break; else m = tot;
  }
}
void calc_height(int n) {
  for (int i = 1; i <= n; ++i) rank[sa[i]] = i;
  for (int i = 1; i <= n; ++i) {
    height[rank[i]] = max(0, height[rank[i - 1]] - 1);
    if (rank[i] == 1) continue;
    int j = sa[rank[i] - 1];
    while (max(i, j) + height[rank[i]] <= n && a[i + height[rank[i]]] == a[j +
    ↪ height[rank[i]]]) ++height[rank[i]];
  }
}
```

## 4.2 后缀自动机

```
static const int MAXL = MAXN * 2;  // MAXN is original length
static const int alphabet = 26;  // sometimes need changing
int l, last, cnt, trans[MAXL][alphabet], par[MAXL], sum[MAXL], seq[MAXL], mxl[MAXL],
  ↪ size[MAXL];  // mxl is maxlength, size is the size of right
char str[MAXL];
inline void init() {
  l = strlen(str + 1); cnt = last = 1;
  for (int i = 0; i <= l * 2; ++i) memset(trans[i], 0, sizeof(trans[i]));
  memset(par, 0, sizeof(*par) * (l * 2 + 1));
  memset(mxl, 0, sizeof(*mxl) * (l * 2 + 1));
  memset(size, 0, sizeof(*size) * (l * 2 + 1));
}
inline void extend(int pos, int c) {
  int p = last, np = last = ++cnt;
  mxl[np] = mxl[p] + 1; size[np] = 1;
  for (; p && !trans[p][c]; p = par[p]) trans[p][c] = np;
  if (!p) par[np] = 1;
  else {
    int q = trans[p][c];
    if (mxl[p] + 1 == mxl[q]) par[np] = q;
    else {
      int nq = ++cnt;
      mxl[nq] = mxl[p] + 1;
      memcpy(trans[nq], trans[q], sizeof(trans[nq]));
      par[nq] = par[q];
      par[np] = par[q] = nq;
      for (; trans[p][c] == q; p = par[p]) trans[p][c] = nq;
    }
  }
}
inline void buildsam() {
  for (int i = 1; i <= l; ++i) extend(i, str[i] - 'a');
  memset(sum, 0, sizeof(*sum) * (l * 2 + 1));
  for (int i = 1; i <= cnt; ++i) sum[mxl[i]]++;
  for (int i = 1; i <= l; ++i) sum[i] += sum[i - 1];
  for (int i = cnt; i; --i) seq[sum[mxl[i]]--] = i;
  for (int i = cnt; i; --i) size[par[seq[i]]] += size[seq[i]];
}
```

## 4.3 EX 后缀自动机

```
inline void add_node(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 = ++cnt; l[auxnode] = l[lastnode] + 1;
      for (int i = 0; i < alphabet; ++i) c[auxnode][i] = c[nownode][i];
      par[auxnode] = par[nownode]; par[nownode] = auxnode;
      for (; lastnode && c[lastnode][x] == nownode; lastnode = par[lastnode]) {
        c[lastnode][x] = auxnode;
      }
      last = auxnode;
    }
  } else {
    int newnode = ++cnt; l[newnode] = l[lastnode] + 1;
    for (; lastnode && !c[lastnode][x]; lastnode = par[lastnode]) c[lastnode][x] =
    ↪ newnode;
    if (!lastnode) par[newnode] = 1;
    else {
      int nownode = c[lastnode][x];
      if (l[lastnode] + 1 == l[nownode]) par[newnode] = nownode;
      else {
        int auxnode = ++cnt; l[auxnode] = l[lastnode] + 1;
        for (int i = 0; i < alphabet; ++i) c[auxnode][i] = c[nownode][i];
        par[auxnode] = par[nownode]; par[nownode] = par[newnode] = auxnode;
        for (; lastnode && c[lastnode][x] == nownode; lastnode = par[lastnode]) {
          c[lastnode][x] = auxnode;
        }
      }
    }
    last = newnode;
  }
}
```

### 4.4 后缀树

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；

### 4.5 回文自动机

```
int nT, nStr, last, c[MAXT][26], fail[MAXT], r[MAXN], l[MAXN], s[MAXN];
int allocate(int len) {
  l[nT] = len;
  r[nT] = 0;
  fail[nT] = 0;
  memset(c[nT], 0, sizeof(c[nT]));
  return nT++;
}
void init() {
  nT = nStr = 0;
  int newE = allocate(0);
  int newO = allocate(-1);
  last = newE;
  fail[newE] = newO;
  fail[newO] = newE;
  s[0] = -1;
}
void add(int x) {
  s[++nStr] = x;
  int now = last;
  while (s[nStr - l[now] - 1] != s[nStr]) now = fail[now];
  if (!c[now][x]) {
    int newnode = allocate(l[now] + 2), &newfail = fail[newnode];
    newfail = fail[now];
    while (s[nStr - l[newfail] - 1] != s[nStr]) newfail = fail[newfail];
    newfail = c[newfail][x];
    c[now][x] = newnode;
  }
  last = c[now][x];
  r[last]++;
}
void count() {
  for (int i = nT - 1; i >= 0; i--) {
    r[fail[i]] += r[i];
  }
}
```

## 5 数据结构

### 5.1 KD-Tree

```
long long norm(const long long &x) {
    //    For manhattan distance
    return std::abs(x);
    //    For euclid distance
    return x * x;
}

struct Point {
    int x, y, id;

    const int& operator [] (int index) const {
        if (index == 0) {
            return x;
        } else {
            return y;
        }
    }

    friend long long dist(const Point &a, const Point &b) {
        long long result = 0;
        for (int i = 0; i < 2; ++i) {
            result += norm(a[i] - b[i]);
        }
        return result;
    }
} point[N];

struct Rectangle {
    int min[2], max[2];
```

```
    Rectangle() {
        min[0] = min[1] = INT_MAX;  // sometimes int is not enough
        max[0] = max[1] = INT_MIN;
    }

    void add(const Point &p) {
        for (int i = 0; i < 2; ++i) {
            min[i] = std::min(min[i], p[i]);
            max[i] = std::max(max[i], p[i]);
        }
    }

    long long dist(const Point &p) {
        long long result = 0;
        for (int i = 0; i < 2; ++i) {
            //    For minimum distance
            result += norm(std::min(std::max(p[i], min[i]), max[i]) - p[i]);
            //    For maximum distance
            result += std::max(norm(max[i] - p[i]), norm(min[i] - p[i]));
        }
        return result;
    }
};

struct Node {
    Point seperator;
    Rectangle rectangle;
    int child[2];

    void reset(const Point &p) {
        seperator = p;
        rectangle = Rectangle();
        rectangle.add(p);
        child[0] = child[1] = 0;
    }
} tree[N << 1];

int size, pivot;

bool compare(const Point &a, const Point &b) {
    if (a[pivot] != b[pivot]) {
        return a[pivot] < b[pivot];
    }
    return a.id < b.id;
}

// 左閉右開: build(1, n + 1)
int build(int l, int r, int type = 1) {
    pivot = type;
    if (l >= r) {
        return 0;
    }
    int x = ++size;
    int mid = l + r >> 1;
    std::nth_element(point + l, point + mid, point + r, compare);
    tree[x].reset(point[mid]);
    for (int i = l; i < r; ++i) {
        tree[x].rectangle.add(point[i]);
    }
    tree[x].child[0] = build(l, mid, type ^ 1);
    tree[x].child[1] = build(mid + 1, r, type ^ 1);
    return x;
}

int insert(int x, const Point &p, int type = 1) {
    pivot = type;
    if (x == 0) {
        tree[++size].reset(p);
        return size;
    }
    tree[x].rectangle.add(p);
    if (compare(p, tree[x].seperator)) {
        tree[x].child[0] = insert(tree[x].child[0], p, type ^ 1);
    } else {
        tree[x].child[1] = insert(tree[x].child[1], p, type ^ 1);
    }
    return x;
}

// For minimum distance
// For maximum: 下面递归 query 时 0, 1 换顺序;< and >;min and max
void query(int x, const Point &p, std::pair<long long, int> &answer, int type = 1) {
    pivot = type;
    if (x == 0 || tree[x].rectangle.dist(p) > answer.first) {
        return;
    }
    answer = std::min(answer,
             std::make_pair(dist(tree[x].seperator, p), tree[x].seperator.id));
    if (compare(p, tree[x].seperator)) {
        query(tree[x].child[0], p, answer, type ^ 1);
        query(tree[x].child[1], p, answer, type ^ 1);
    } else {
        query(tree[x].child[1], p, answer, type ^ 1);
        query(tree[x].child[0], p, answer, type ^ 1);
    }
}

std::priority_queue<std::pair<long long, int> > answer;

void query(int x, const Point &p, int k, int type = 1) {
    pivot = type;
    if (x == 0 || (int)answer.size() == k && tree[x].rectangle.dist(p) >
   ↪ answer.top().first) {
        return;
    }
    answer.push(std::make_pair(dist(tree[x].seperator, p), tree[x].seperator.id));
    if ((int)answer.size() > k) {
        answer.pop();
    }
```

```
    if (compare(p, tree[x].seperator)) {
        query(tree[x].child[0], p, k, type ^ 1);
        query(tree[x].child[1], p, k, type ^ 1);
    } else {
        query(tree[x].child[1], p, k, type ^ 1);
        query(tree[x].child[0], p, k, type ^ 1);
    }
}
```

## 5.2 Treap

```
struct Node{
  int mn, key, size, tag;
  bool rev;
  Node* ch[2];
  Node(int mn, int key, int size): mn(mn), key(key), size(size), rev(0), tag(0){}
  void downtag();
  Node* update(){
    mn = min(ch[0] -> mn, min(key, ch[1] -> mn));
    size = ch[0] -> size + 1 + ch[1] -> size;
    return this;
  }
};
typedef pair<Node*, Node*> Pair;
Node *null, *root;
void Node::downtag(){
  if(rev){
    for(int i = 0; i < 2; i++)
      if(ch[i] != null){
        ch[i] -> rev ^= 1;
        swap(ch[i] -> ch[0], ch[i] -> ch[1]);
      }
    rev = 0;
  }
  if(tag){
    for(int i = 0; i < 2; i++)
      if(ch[i] != null){
        ch[i] -> key += tag;
        ch[i] -> mn += tag;
        ch[i] -> tag += tag;
      }
    tag = 0;
  }
}
int r(){
  static int s = 3023192386;
  return (s += (s << 3) + 1) & (~0u >> 1);
}
bool random(int x, int y){
  return r() % (x + y) < x;
}
Node* merge(Node *p, Node *q){
  if(p == null) return q;
  if(q == null) return p;
  p -> downtag();
  q -> downtag();
  if(random(p -> size, q -> size)){
    p -> ch[1] = merge(p -> ch[1], q);
    return p -> update();
  }else{
    q -> ch[0] = merge(p, q -> ch[0]);
    return q -> update();
  }
}
Pair split(Node *x, int n){
  if(x == null) return make_pair(null, null);
  x -> downtag();
  if(n <= x -> ch[0] -> size){
    Pair ret = split(x -> ch[0], n);
    x -> ch[0] = ret.second;
    return make_pair(ret.first, x -> update());
  }
  Pair ret = split(x -> ch[1], n - x -> ch[0] -> size - 1);
  x -> ch[1] = ret.first;
  return make_pair(x -> update(), ret.second);
}
pair<Node*, Pair> get_segment(int l, int r){
  Pair ret = split(root, l - 1);
  return make_pair(ret.first, split(ret.second, r - l + 1));
}
int main(){
  null = new Node(INF, INF, 0);
  null -> ch[0] = null -> ch[1] = null;
  root = null;
}
```

## 5.3 Link/cut Tree

```
inline void reverse(int x) {
  tr[x].rev ^= 1; swap(tr[x].c[0], tr[x].c[1]);
}

inline void rotate(int x, int k) {
  int y = tr[x].fa, z = tr[y].fa;
    tr[x].fa = z; tr[z].c[tr[z].c[1] == y] = x;
    tr[tr[x].c[k ^ 1]].fa = y; tr[y].c[k] = tr[x].c[k ^ 1];
    tr[x].c[k ^ 1] = y; tr[y].fa = x;
}

inline void splay(int x, int w) {
  int z = x; pushdown(x);
  while (tr[x].fa != w) {
    int y = tr[x].fa; z = tr[y].fa;
    if (z == w) {
      pushdown(z = y); pushdown(x);
      rotate(x, tr[y].c[1] == x);
      update(y); update(x);
    } else {
      pushdown(z); pushdown(y); pushdown(x);
      int t1 = tr[y].c[1] == x, t2 = tr[z].c[1] == y;
      if (t1 == t2) rotate(y, t2), rotate(x, t1);
```

```
      else rotate(x, t1), rotate(x, t2);
      update(z); update(y); update(x);
    }
  }
  update(x);
  if (x != z) par[x] = par[z], par[z] = 0;
}

inline void access(int x) {
  for (int y = 0; x; y = x, x = par[x]) {
    splay(x, 0);
    if (tr[x].c[1]) par[tr[x].c[1]] = x, tr[tr[x].c[1]].fa = 0;
    tr[x].c[1] = y; par[y] = 0; tr[y].fa = x; update(x);
  }
}

inline void makeroot(int x) {
  access(x); splay(x, 0); reverse(x);
}

inline void link(int x, int y) {
  makeroot(x); par[x] = y;
}

inline void cut(int x, int y) {
  access(x); splay(y, 0);
  if (par[y] != x) swap(x, y), access(x), splay(y, 0);
  par[y] = 0;
}

inline void split(int x, int y) {  // x will be the root of the tree
  makeroot(y); access(x); splay(x, 0);
}
```

### 5.4 树状数组查询第 k 小元素

```
inline void reverse(int x) {
  tr[x].rev ^= 1; swap(tr[x].c[0], tr[x].c[1]);
}

inline void rotate(int x, int k) {
  int y = tr[x].fa, z = tr[y].fa;
    tr[x].fa = z; tr[z].c[tr[z].c[1] == y] = x;
    tr[tr[x].c[k ^ 1]].fa = y; tr[y].c[k] = tr[x].c[k ^ 1];
    tr[x].c[k ^ 1] = y; tr[y].fa = x;
}

inline void splay(int x, int w) {
  int z = x; pushdown(x);
  while (tr[x].fa != w) {
    int y = tr[x].fa; z = tr[y].fa;
    if (z == w) {
      pushdown(z = y); pushdown(x);
      rotate(x, tr[y].c[1] == x);
      update(y); update(x);
    } else {
      pushdown(z); pushdown(y); pushdown(x);
      int t1 = tr[y].c[1] == x, t2 = tr[z].c[1] == y;
      if (t1 == t2) rotate(y, t2), rotate(x, t1);
      else rotate(x, t1), rotate(x, t2);
      update(z); update(y); update(x);
    }
  }
  update(x);
  if (x != z) par[x] = par[z], par[z] = 0;
}

inline void access(int x) {
  for (int y = 0; x; y = x, x = par[x]) {
    splay(x, 0);
    if (tr[x].c[1]) par[tr[x].c[1]] = x, tr[tr[x].c[1]].fa = 0;
    tr[x].c[1] = y; par[y] = 0; tr[y].fa = x; update(x);
  }
}

inline void makeroot(int x) {
  access(x); splay(x, 0); reverse(x);
}

inline void link(int x, int y) {
  makeroot(x); par[x] = y;
}

inline void cut(int x, int y) {
  access(x); splay(y, 0);
  if (par[y] != x) swap(x, y), access(x), splay(y, 0);
  par[y] = 0;
}

inline void split(int x, int y) {  // x will be the root of the tree
  makeroot(y); access(x); splay(x, 0);
}
```

## 6 图论

### 6.1 基础

```
struct Graph {  // Remember to call .init()!
    int e, nxt[M], v[M], adj[N], n;
    bool base;
    __inline void init(bool _base, int _n = 0) {
        assert(n < N);
        n = _n; base = _base;
        e = 0; memset(adj + base, -1, sizeof(*adj) * n);
    }
    __inline int new_node() {
        adj[n + base] = -1;
        assert(n + base + 1 < N);
        return n++ + base;
    }
    __inline void ins(int u0, int v0) {  // directional
```

```
        assert(u0 < n + base && v0 < n + base);
        v[e] = v0; nxt[e] = adj[u0]; adj[u0] = e++;
        assert(e < M);
    }
    __inline void bi_ins(int u0, int v0) {  // bi-directional
        ins(u0, v0); ins(v0, u0);
    }
};
```

## 6.2 KM

```
struct KM {
  // Truly O(n^3)
  // 邻接矩阵，不能连的边设为 -INF，求最小权匹配时边权取负，但不能连的还是 -INF，使用时先对 1 -> n 调用 hungary() ，再 get_ans() 求值
  int w[N][N];
  int lx[N], ly[N], match[N], way[N], slack[N];
  bool used[N];
  void init() {
    for (int i = 1; i <= n; i++) {
      match[i] = 0;
      lx[i] = 0;
      ly[i] = 0;
      way[i] = 0;
    }
  }
  void hungary(int x) {
    match[0] = x;
    int j0 = 0;
    for (int j = 0; j <= n; j++) {
      slack[j] = INF;
      used[j] = false;
    }

    do {
      used[j0] = true;
      int i0 = match[j0], delta = INF, j1 = 0;
      for (int j = 1; j <= n; j++) {
        if (used[j] == false) {
          int cur = -w[i0][j] - lx[i0] - ly[j];
          if (cur < slack[j]) {
            slack[j] = cur;
            way[j] = j0;
          }
          if (slack[j] < delta) {
            delta = slack[j];
            j1 = j;
          }
        }
      }
      for (int j = 0; j <= n; j++) {
        if (used[j]) {
          lx[match[j]] += delta;
          ly[j] -= delta;
        }
        else slack[j] -= delta;
      }
      j0 = j1;
    } while (match[j0] != 0);

    do {
      int j1 = way[j0];
      match[j0] = match[j1];
      j0 = j1;
    } while (j0);
  }

  int get_ans() {
    int sum = 0;
    for(int i = 1; i <= n; i++) {
      if (w[match[i]][i] == -INF) ; // 无解
      if (match[i] > 0) sum += w[match[i]][i];
    }
    return sum;
  }
} km;
```

## 6.3 点双连通分量

bcc.forest is a set of connected tree whose vertices are chequered with cut-vertex and BCC.

```
const bool BCC_VERTEX = 0, BCC_EDGE = 1;
struct BCC {  // N = N0 + M0. Remember to call init(&raw_graph).
  Graph *g, forest; // g is raw graph ptr.
  int dfn[N], DFN, low[N];
  int stack[N], top;
  int expand_to[N];    // Where edge i is expanded to in expaned graph.
  // Vertex i expaned to i.
  int compress_to[N];  // Where vertex i is compressed to.
  bool vertex_type[N], cut[N], compress_cut[N], branch[M];
  //std::vector<int> BCC_component[N];  // Cut vertex belongs to none.
  __inline void init(Graph *raw_graph) {
    g = raw_graph;
  }
  void DFS(int u, int pe) {
    dfn[u] = low[u] = ++DFN; cut[u] = false;
    if (!~g->adj[u]) {
      cut[u] = 1;
      compress_to[u] = forest.new_node();
      compress_cut[compress_to[u]] = 1;
    }
    for (int e = g->adj[u]; ~e; e = g->nxt[e]) {
      int v = g->v[e];
      if ((e ^ pe) > 1 && dfn[v] > 0 && dfn[v] < dfn[u]) {
        stack[top++] = e;
        low[u] = std::min(low[u], dfn[v]);
      }
      else if (!dfn[v]) {
        stack[top++] = e; branch[e] = 1;
        DFS(v, e);
        low[u] = std::min(low[v], low[u]);
```

```
        if (low[v] >= dfn[u]) {
          if (!cut[u]) {
            cut[u] = 1;
            compress_to[u] = forest.new_node();
            compress_cut[compress_to[u]] = 1;
          }
          int cc = forest.new_node();
          forest.bi_ins(compress_to[u], cc);
          compress_cut[cc] = 0;
          //BCC_component[cc].clear();
          do {
            int cur_e = stack[--top];
            compress_to[expand_to[cur_e]] = cc;
            compress_to[expand_to[cur_e^1]] = cc;
            if (branch[cur_e]) {
              int v = g->v[cur_e];
              if (cut[v])
                forest.bi_ins(cc, compress_to[v]);
              else {
                //BCC_component[cc].push_back(v);
                compress_to[v] = cc;
              }
            }
          } while (stack[top] != e);
        }
      }
    }
  }
  void solve() {
    forest.init(g->base);
    int n = g->n;
    for (int i = 0; i < g->e; i++) {
      expand_to[i] = g->new_node();
    }
    memset(branch, 0, sizeof(*branch) * g->e);
    memset(dfn + g->base, 0, sizeof(*dfn) * n); DFN = 0;
    for (int i = 0; i < n; i++)
      if (!dfn[i + g->base]) {
        top = 0;
        DFS(i + g->base, -1);
      }
  }
} bcc;

bcc.init(&raw_graph);
bcc.solve();
// Do something with bcc.forest ...
```

### 6.4 边双连通分量

```
struct BCC {
  Graph *g, forest;
  int dfn[N], low[N], stack[N], tot[N], belong[N], vis[N], top, dfs_clock;
  // tot[] is the size of each BCC, belong[] is the BCC that each node belongs to
  pair<int, int > ori[M]; // bridge in raw_graph(raw node)
  bool is_bridge[M];
  __inline void init(Graph *raw_graph) {
    g = raw_graph;
    memset(is_bridge, false, sizeof(*is_bridge) * g -> e);
    memset(vis + g -> base, 0, sizeof(*vis) * g -> n);
  }
  void tarjan(int u, int from) {
    dfn[u] = low[u] = ++dfs_clock; vis[u] = 1; stack[++top] = u;
    for (int p = g -> adj[u]; ~p; p = g -> nxt[p]) {
      if ((p ^ 1) == from) continue;
      int v = g -> v[p];
      if (vis[v]) {
        if (vis[v] == 1) low[u] = min(low[u], dfn[v]);
      } else {
        tarjan(v, p);
        low[u] = min(low[u], low[v]);
        if (low[v] > dfn[u]) is_bridge[p / 2] = true;
      }
    }
    if (dfn[u] != low[u]) return;
    tot[forest.new_node()] = 0;
    do {
      belong[stack[top]] = forest.n;
      vis[stack[top]] = 2;
      tot[forest.n]++;
      --top;
    } while (stack[top + 1] != u);
  }
  void solve() {
    forest.init(g -> base);
    int n = g -> n;
    for (int i = 0; i < n; ++i)
      if (!vis[i + g -> base]) {
        top = dfs_clock = 0;
        tarjan(i + g -> base, -1);
      }
    for (int i = 0; i < g -> e / 2; ++i)
      if (is_bridge[i]) {
        int e = forest.e;
        forest.bi_ins(belong[g -> v[i * 2]], belong[g -> v[i * 2 + 1]], g -> w[i * 2]);
        ori[e] = make_pair(g -> v[i * 2 + 1], g -> v[i * 2]);
        ori[e + 1] = make_pair(g -> v[i * 2], g -> v[i * 2 + 1]);
      }
  }
} bcc;
```

### 6.5 最小树形图

```
const int MAXN,INF;// INF >= sum( W_ij )
int from[MAXN + 10][MAXN * 2 + 10],n,m,edge[MAXN + 10][MAXN * 2 + 10];
int sel[MAXN * 2 + 10],fa[MAXN * 2 + 10],vis[MAXN * 2 + 10];
int getfa(int x){if(x == fa[x]) return x; return fa[x] = getfa(fa[x]);}
void liuzhu(){ // 1-base: root is 1, answer = (sel[i], i) for i in [2..n]
  fa[1] = 1;
  for(int i = 2; i <= n; ++i){
    sel[i] = 1; fa[i] = i;
```

```
    for(int j = 1; j <= n; ++j) if(fa[j] != i)
      if(from[j][i] = i, edge[sel[i]][i] > edge[j][i]) sel[i] = j;
  }
  int limit = n;
  while(1){
    int prelimit = limit; memset(vis, 0, sizeof(vis)); vis[1] = 1;
    for(int i = 2; i <= prelimit; ++i) if(fa[i] == i && !vis[i]){
      int j = i; while(!vis[j]) vis[j] = i, j = getfa(sel[j]);
      if(j == 1 || vis[j] != i) continue; vector<int> C; int k = j;
      do C.push_back(k), k = getfa(sel[k]); while(k != j);
      ++limit;
      for(int i = 1; i <= n; ++i){
        edge[i][limit] = INF, from[i][limit] = limit;
      }
      fa[limit] = vis[limit] = limit;
      for(int i = 0; i < int(C.size()); ++i){
        int x = C[i], fa[x] = limit;
        for(int j = 1; j <= n; ++j)
          if(edge[j][x] != INF && edge[j][limit] > edge[j][x] - edge[sel[x]][x]){
            edge[j][limit] = edge[j][x] - edge[sel[x]][x];
            from[j][limit] = x;
          }
      }
      for(int j=1;j<=n;++j) if(getfa(j)==limit) edge[j][limit] = INF;
      sel[limit] = 1;
      for(int j = 1; j <= n; ++j)
        if(edge[sel[limit]][limit] > edge[j][limit]) sel[limit] = j;
    }
    if(prelimit == limit) break;
  }
  for(int i = limit; i > 1; --i) sel[from[sel[i]][i]] = sel[i];
}
```

## 6.6 带花树

```
vector<int> link[maxn];
int n,match[maxn],Queue[maxn],head,tail;
int pred[maxn],base[maxn],start,finish,newbase;
bool InQueue[maxn],InBlossom[maxn];
void push(int u){ Queue[tail++]=u;InQueue[u]=true; }
int pop(){ return Queue[head++]; }
int FindCommonAncestor(int u,int v){
  bool InPath[maxn];
  for(int i=0;i<n;i++) InPath[i]=0;
  while(true){ u=base[u];InPath[u]=true;if(u==start) break;u=pred[match[u]]; }
  while(true){ v=base[v];if(InPath[v]) break;v=pred[match[v]]; }
  return v;
}
void ResetTrace(int u){
  int v;
  while(base[u]!=newbase){
    v=match[u];
    InBlossom[base[u]]=InBlossom[base[v]]=true;
    u=pred[v];
    if(base[u]!=newbase) pred[u]=v;
  }
}
void BlossomContract(int u,int v){
  newbase=FindCommonAncestor(u,v);
  for (int i=0;i<n;i++)
  InBlossom[i]=0;
  ResetTrace(u);ResetTrace(v);
  if(base[u]!=newbase) pred[u]=v;
  if(base[v]!=newbase) pred[v]=u;
  for(int i=0;i<n;++i)
  if(InBlossom[base[i]]){
    base[i]=newbase;
    if(!InQueue[i]) push(i);
  }
}
bool FindAugmentingPath(int u){
  bool found=false;
  for(int i=0;i<n;++i) pred[i]=-1,base[i]=i;
  for (int i=0;i<n;i++) InQueue[i]=0;
  start=u;finish=-1; head=tail=0; push(start);
  while(head<tail){
    int u=pop();
    for(int i=link[u].size()-1;i>=0;i--){
      int v=link[u][i];
      if(base[u]!=base[v]&&match[u]!=v)
        if(v==start||(match[v]>=0&&pred[match[v]]>=0))
          BlossomContract(u,v);
        else if(pred[v]==-1){
          pred[v]=u;
          if(match[v]>=0) push(match[v]);
          else{ finish=v; return true; }
        }
    }
  }
  return found;
}
void AugmentPath(){
  int u=finish,v,w;
  while(u>=0){ v=pred[u];w=match[v];match[v]=u;match[u]=v;u=w; }
}
void FindMaxMatching(){
  for(int i=0;i<n;++i) match[i]=-1;
  for(int i=0;i<n;++i) if(match[i]==-1) if(FindAugmentingPath(i)) AugmentPath();
}
```

## 6.7 Dominator Tree

```
vector<int> prec[N], succ[N];
vector<int> ord;
int stamp, vis[N];
int num[N];
int fa[N];
void dfs(int u) {
  vis[u] = stamp;
  num[u] = ord.size();
  ord.push_back(u);
```

```
  for (int i = 0; i < (int)succ[u].size(); ++i) {
    int v = succ[u][i];
    if (vis[v] != stamp) {
      fa[v] = u;
      dfs(v);
    }
  }
}
int fs[N], mins[N], dom[N], sem[N];
int find(int u) {
  if (u != fs[u]) {
    int v = fs[u];
    fs[u] = find(fs[u]);
    if (mins[v] != -1 && num[sem[mins[v]]] < num[sem[mins[u]]]) {
      mins[u] = mins[v];
    }
  }
  return fs[u];
}
void merge(int u, int v) { fs[u] = v; }
vector<int> buf[N];
int buf2[N];
void mark(int source) {
  ord.clear();
  ++stamp;
  dfs(source);
  for (int i = 0; i < (int)ord.size(); ++i) {
    int u = ord[i];
    fs[u] = u, mins[u] = -1, buf2[u] = -1;
  }
  for (int i = (int)ord.size() - 1; i > 0; --i) {
    int u = ord[i], p = fa[u];
    sem[u] = p;
    for (int j = 0; j < (int)prec[u].size(); ++j) {
      int v = prec[u][j];
      if (use[v] != stamp) continue;
      if (num[v] > num[u]) {
        find(v); v = sem[mins[v]];
      }
      if (num[v] < num[sem[u]]) {
        sem[u] = v;
      }
    }
    buf[sem[u]].push_back(u);
    mins[u] = u;
    merge(u, p);
    while (buf[p].size()) {
      int v = buf[p].back();
      buf[p].pop_back();
      find(v);
      if (sem[v] == sem[mins[v]]) {
        dom[v] = sem[v];
      } else {
        buf2[v] = mins[v];
      }
    }
  }
  dom[ord[0]] = ord[0];
  for (int i = 0; i < (int)ord.size(); ++i) {
    int u = ord[i];
    if (~buf2[u]) {
      dom[u] = dom[buf2[u]];
    }
  }
}
```

### 6.8 无向图最小割

```
int cost[maxn][maxn],seq[maxn],len[maxn],n,m,pop,ans;
bool used[maxn];
void Init(){
  int i,j,a,b,c;
  for(i=0;i<n;i++) for(j=0;j<n;j++) cost[i][j]=0;
  for(i=0;i<m;i++){
    scanf("%d %d %d",&a,&b,&c); cost[a][b]+=c; cost[b][a]+=c;
  }
  pop=n; for(i=0;i<n;i++) seq[i]=i;
}
void Work(){
  ans=inf; int i,j,k,l,mm,sum,pk;
  while(pop > 1){
    for(i=1;i<pop;i++) used[seq[i]]=0; used[seq[0]]=1;
    for(i=1;i<pop;i++) len[seq[i]]=cost[seq[0]][seq[i]];
    pk=0; mm=-inf; k=-1;
    for(i=1;i<pop;i++) if(len[seq[i]] > mm){ mm=len[seq[i]]; k=i; }
    for(i=1;i<pop;i++){
      used[seq[l=k]]=1;
      if(i==pop-2) pk=k;
      if(i==pop-1) break;
      mm=-inf;
      for(j=1;j<pop;j++) if(!used[seq[j]])
        if((len[seq[j]]+=cost[seq[l]][seq[j]]) > mm)
          mm=len[seq[j]], k=j;
    }
    sum=0;
    for(i=0;i<pop;i++) if(i != k) sum+=cost[seq[k]][seq[i]];
    ans=min(ans,sum);
    for(i=0;i<pop;i++)
      cost[seq[k]][seq[i]]=cost[seq[i]][seq[k]]+=cost[seq[pk]][seq[i]];
    seq[pk]=seq[--pop];
  }
  printf("%d\n",ans);
}
```

### 6.9 重口味费用流

```
int S, T, totFlow, totCost;

int 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]]) {
            int y = e.other[i];
            if (dis[y] + e.cost[i] == dis[x]) {
                int delta = dfs (y, 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[y] = min (slack[y], dis[y] + 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);
}
```

### 6.10 2-SAT

```
int S, T, totFlow, totCost;

int 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]]) {
            int y = e.other[i];
            if (dis[y] + e.cost[i] == dis[x]) {
                int delta = dfs (y, 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[y] = min (slack[y], dis[y] + 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);
}
```

## 7 其他

### 7.1 Dancing Links

```
struct Node {
  Node *l, *r, *u, *d, *col;
  int size, line_no;
  Node() {
    size = 0; line_no = -1;
    l = r = u = d = col = NULL;
  }
} *root;
```

```

void cover(Node *c) {
  c->l->r = c->r; c->r->l = c->l;
  for (Node *u = c->d; u != c; u = u->d)
    for (Node *v = u->r; v != u; v = v->r) {
      v->d->u = v->u;
      v->u->d = v->d;
      -- v->col->size;
    }
}

void uncover(Node *c) {
  for (Node *u = c->u; u != c; u = u->u) {
    for (Node *v = u->l; v != u; v = v->l) {
      ++ v->col->size;
      v->u->d = v;
      v->d->u = v;
    }
  }
  c->l->r = c; c->r->l = c;
}

std::vector<int> answer;
bool search(int k) {
  if (root->r == root) return true;
  Node *r = NULL;
  for (Node *u = root->r; u != root; u = u->r)
    if (r == NULL || u->size < r->size)
      r = u;
  if (r == NULL || r->size == 0) return false;
  else {
    cover(r);
    bool succ = false;
    for (Node *u = r->d; u != r && !succ; u = u->d) {
      answer.push_back(u->line_no);
      for (Node *v = u->r; v != u; v = v->r)  // Cover row
        cover(v->col);
      succ |= search(k + 1);
      for (Node *v = u->l; v != u; v = v->l)
        uncover(v->col);
      if (!succ) answer.pop_back();
    }
    uncover(r);
    return succ;
  }
}

bool entry[CR][CC];
Node *who[CR][CC];
int cr, cc;

void construct() {
  root = new Node();
  Node *last = root;
  for (int i = 0; i < cc; ++ i) {
    Node *u = new Node();
    last->r = u; u->l = last;
    Node *v = u; u->line_no = i;
    last = u;
    for (int j = 0; j < cr; ++ j)
      if (entry[j][i]) {
        ++ u->size;
        Node *cur = new Node();
        who[j][i] = cur;
        cur->line_no = j;
        cur->col = u;
        cur->u = v; v->d = cur;
        v = cur;
      }
    v->d = u; u->u = v;
  }
  last->r = root; root->l = last;
  for (int j = 0; j < cr; ++ j) {
    Node *last = NULL;
    for (int i = cc - 1; i >= 0; -- i)
      if (entry[j][i]) {
        last = who[j][i];
        break;
      }
    for (int i = 0; i < cc; ++ i)
      if (entry[j][i]) {
        last->r = who[j][i];
        who[j][i]->l = last;
        last = who[j][i];
      }
  }
}

void destruct() {
  for (Node *u = root->r; u != root; ) {
    for (Node *v = u->d; v != u; ) {
      Node *nxt = v->d;
      delete(v);
      v = nxt;
    }
    Node *nxt = u->r;
    delete(u); u = nxt;
  }
  delete root;
}
```

## 7.2 蔡勒公式

0 for Sunday. Day and month is 1-based.

```
int zeller(int y,int m,int d) {
  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);
}
```

# 8 技巧

## 8.1 真正的释放 STL 容器内存空间

```cpp
template <typename T>
__inline void clear(T& container) {
  container.clear();  // 或者删除了一堆元素
  T(container).swap(container);
}
```

## 8.2 无敌的大整数相乘取模

Time complexity $O(1)$.

```cpp
// 需要保证 x 和 y 非负
long long mult(long long x, long long y, long long MODN) {
  long long t = (x * y - (long long)((long double)x / MODN * y + 1e-3) * MODN) % MODN;
  return t < 0 ? t + MODN : t;
}
```

## 8.3 无敌的读入优化

```cpp
// getchar() 读入优化 << 关同步 cin << 此优化
// 用 isdigit() 会小幅变慢
// 返回 false 表示读到文件尾
namespace Reader {
    const int L = (1 << 15) + 5;
    char buffer[L], *S, *T;
    __inline bool getchar(char &ch) {
        if (S == T) {
            T = (S = buffer) + fread(buffer, 1, L, stdin);
            if (S == T) {
        ch = EOF;
        return false;
      }
        }
    ch = *S++;
    return true;
    }
    __inline bool getint(int &x) {
    char ch; bool neg = 0;
    for (; getchar(ch) && (ch < '0' || ch > '9'); ) neg ^= ch == '-';
    if (ch == EOF) return false;
    x = ch - '0';
    for (; getchar(ch), ch >= '0' && ch <= '9'; )
      x = x * 10 + ch - '0';
    if (neg) x = -x;
    return true;
    }
}
```

## 8.4 梅森旋转算法

High quality pseudorandom number generator, twice as efficient as rand() with -O2. C++11 required.

```cpp
#include <random>

int main() {
    std::mt19937 g(seed);  // std::mt19937_64
    std::cout << g() << std::endl;
}
```

# 9 提示

## 9.1 控制 cout 输出实数精度

```cpp
std::cout << std::fixed << std::setprecision(5);
```

## 9.2 vimrc

```
set nu
set sw=4
set sts=4
set ts=4
syntax on
set cindent
```

## 9.3 让 make 支持 c++11

In .bashrc or whatever:

```
export CXXFLAGS='-std=c++11 -Wall'
```

## 9.4 tuple 相关

```cpp
mytuple = std::make_tuple (10, 2.6, 'a');           // packing values into tuple
std::tie (myint, std::ignore, mychar) = mytuple;    // unpacking tuple into variables
std::get<I>(mytuple) = 20;
std::cout << std::get<I>(mytuple) << std::endl;     // get the Ith(const) element
```

## 9.5 线性规划转对偶

$$\begin{array}{l}\text{maximize } \mathbf{c}^T\mathbf{x} \\ \text{subject to } \mathbf{Ax} \le \mathbf{b}, \mathbf{x} \ge 0\end{array} \iff \begin{array}{l}\text{minimize } \mathbf{y}^T\mathbf{b} \\ \text{subject to } \mathbf{y}^T\mathbf{A} \ge \mathbf{c}^T, \mathbf{y} \ge 0\end{array}$$

## 9.6 32-bit/64-bit 随机素数

| 32-bit | 64-bit |
|---|---|
| 73550053 | 1249292846855685773 |
| 148898719 | 1701750434419805569 |
| 189560747 | 3605499878424114901 |
| 459874703 | 5648316673387803781 |
| 1202316001 | 6125342570814357977 |
| 1431183547 | 6215155308775851301 |
| 1438011109 | 6294606778040623451 |
| 1538762023 | 6347330550446020547 |
| 1557944263 | 7429632924303725207 |
| 1981315913 | 8524720079480389849 |

## 9.7 NTT 素数及其原根

| Prime | Primitive root |
|---|---|
| 1053818881 | 7 |
| 1051721729 | 6 |
| 1045430273 | 3 |
| 1012924417 | 5 |
| 1007681537 | 3 |

## 9.8 Java Hints

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

public class Main {
  static int get(char c) {
    if (c <= '9')
      return c - '0';
    else if (c <= 'Z')
      return c - 'A' + 10;
    else
      return c - 'a' + 36;
  }
  static char get(int x) {
    if (x <= 9)
      return (char)(x + '0');
    else if (x <= 35)
      return (char)(x - 10 + 'A');
    else
      return (char)(x - 36 + 'a');
  }
  static BigInteger get(String s, BigInteger x) {
    BigInteger ans = BigInteger.valueOf(0), now = BigInteger.valueOf(1);
    for (int i = s.length() - 1; i >= 0; i--) {
      ans = ans.add(now.multiply(BigInteger.valueOf(get(s.charAt(i)))));
      now = now.multiply(x);
    }
    return ans;
  }
  public static void main(String [] args) {
    Scanner cin = new Scanner(new BufferedInputStream(System.in));
    for (; ; ) {
      BigInteger x = cin.nextBigInteger();
      if (x.compareTo(BigInteger.valueOf(0)) == 0)
        break;
      String s = cin.next(), t = cin.next(), r = "";
      BigInteger ans = get(s, x).mod(get(t, x));
      if (ans.compareTo(BigInteger.valueOf(0)) == 0)
        r = "0";
      for (; ans.compareTo(BigInteger.valueOf(0)) > 0;) {
        r = get(ans.mod(x).intValue()) + r;
        ans = ans.divide(x);
      }
      System.out.println(r);
    }
  }
}

// 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)
```