Nightfall

For Manual/Intelligence

August 6, 2018

Contents

1. Geometry

1.1 二维几何基础操作

```cpp
bool point_on_segment(const point &a,const line &b){
    return sgn (det (a - b.s, b.t - b.s)) == 0 && sgn (dot (b.s - a, b.t - a)) <= 0; }
bool two_side(const point &a,const point &b,const line &c) {
    return sgn (det (a - c.s, c.t - c.s)) * sgn (det (b - c.s, c.t - c.s)) < 0; }
bool intersect_judgment(const line &a,const line &b) {
    if (point_on_segment (b.s, a) || point_on_segment (b.t, a)) return true;
    if (point_on_segment (a.s, b) || point_on_segment (a.t, b)) return true;
    return two_side (a.s, a.t, b) && two_side (b.s, b.t, a); }
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); }
double point_to_line (const point &a, const line &b) {
    return fabs (det (b.t-b.s, a-b.s)) / dis (b.s, b.t); }
point project_to_line (const point &a, const line &b) {
    return b.s + (b.t - b.s) * (dot (a - b.s, b.t - b.s) / (b.t - b.s).norm2 ()); }
double point_to_segment (const point &a, const line &b) {
    if (sgn (dot (b.s - a, b.t - b.s)) * sgn (dot (b.t - a, b.t - b.s)) <= 0)
        return fabs (det (b.t - b.s, a - b.s)) / dis (b.s, b.t);
    return std::min (dis (a, b.s), dis (a, b.t)); }
bool in_polygon (const point &p, const std::vector <point> & po) {
    int n = (int) po.size (); int counter = 0;
    for (int i = 0; i < n; ++i) {
        point a = po[i], b = po[(i + 1) % n];
        if (point_on_segment (p, line (a, b))) return true;
        int x = sgn (det (p - a, b - a)), y = sgn (a.y - p.y), z = sgn (b.y - p.y);
    if (x > 0 && y <= 0 && z > 0) ++counter;
    if (x < 0 && z <= 0 && y > 0) --counter; }
  return counter != 0; }
std::vector <point> line_circle_intersect (const line &a, const circle &b) {
  if (cmp (point_to_line (b.c, a), b.r) > 0) return std::vector <point> ();
  double x = sqrt (sqr (b.r) - sqr (point_to_line (b.c, a)));
  return std::vector <point> ({project_to_line (b.c, a) + (a.s - a.t).unit () * x, project_to_line (b.c, a) - (a.s - a.t).unit () * x}); }
double circle_intersect_area (const circle &a, const circle &b) {
  double d = dis (a.c, b.c);
  if (sgn (d - (a.r + b.r)) >= 0) return 0;
  if (sgn (d - abs(a.r - b.r)) <= 0) {
    double r = std::min (a.r, b.r);
    return r * r * PI; }
  double x = (d * d + a.r * a.r - b.r * b.r) / (2 * d),
         t1 = acos (min (1., max (-1., x / a.r))),
         t2 = acos (min (1., max (-1., (d - x) / b.r)));
  return a.r * a.r * t1 + b.r * b.r * t2 - d * a.r * sin (t1); }
std::vector <point> circle_intersect (const circle &a, const circle &b) {
  if (a.c == b.c || cmp (dis (a.c, b.c), a.r + b.r) > 0 || cmp (dis (a.c, b.c), std::abs (a.r - b.r)) < 0)
    return std::vector <point> ();
  point r = (b.c - a.c).unit ();
  double d = dis (a.c, b.c);
  double x = ((sqr (a.r) - sqr (b.r)) / d + d) / 2;
  double h = sqrt (sqr (a.r) - sqr (x));
  if (sgn (h) == 0) return std::vector <point> ({a.c + r * x});
  return std::vector <point> ({a.c + r * x + r.rot90 () * h, a.c + r * x - r.rot90 () * h}); }
// 返回按照顺时针方向
std::vector <point> tangent (const point &a, const circle &b) {
  circle p = make_circle (a, b.c);
  return circle_intersect (p, b); }
std::vector <line> extangent (const circle &a, const circle &b) {
  std::vector <line> ret;
  if (cmp (dis (a.c, b.c), std::abs (a.r - b.r)) <= 0) return ret;
  if (sgn (a.r - b.r) == 0) {
    point dir = b.c - a.c;
    dir = (dir * a.r / dir.norm ()).rot90 ();
    ret.push_back (line (a.c + dir, b.c + dir));
    ret.push_back (line (a.c - dir, b.c - dir));
  } else {
    point p = (b.c * a.r - a.c * b.r) / (a.r - b.r);
    std::vector pp = tangent (p, a), qq = tangent (p, b);
    if (pp.size () == 2 && qq.size () == 2) {
      if (cmp (a.r, b.r) < 0) std::swap (pp[0], pp[1]), std::swap (qq[0], qq[1]);
      ret.push_back(line (pp[0], qq[0]));
      ret.push_back(line (pp[1], qq[1])); } }
  return ret; }
std::vector <line> intangent (const circle &a, const circle &b) {
  point p = (b.c * a.r + a.c * b.r) / (a.r + b.r);
  std::vector pp = tangent (p, a), qq = tangent (p, b);
  if (pp.size () == 2 && qq.size () == 2) {
    ret.push_back (line (pp[0], qq[0]));
    ret.push_back (line (pp[1], qq[1])); }
  return ret; }
```

1.2 直线半平面交

```
std::vector <point> cut (const std::vector<point> &c, line p) {
  std::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_left (p.s, p.t, c[i])) ret.push_back (c[i]);
    if (two_side (c[i], c[j], p))
      ret.push_back (line_intersect (p, line (c[i], c[j]))); }
  return ret; }
bool turn_left (const line &l, const point &p) {
  return turn_left (l.s, l.t, p); }
std::vector <point> half_plane_intersect (std::vector <line> h) {
  typedef std::pair <double, line> polar;
  std::vector <polar> g;
  g.resize (h.size ());
  for (int i = 0; i < (int) h.size (); ++i) {
    point v = h[i].t - h[i].s;
    g[i] = std::make_pair (atan2 (v.y, v.x), h[i]); }
  sort (g.begin (), g.end (), [] (const polar &a, const polar &b) {
    if (cmp (a.first, b.first) == 0)
      return sgn (det (a.second.t - a.second.s, b.second.t - a.second.s)) < 0;
    else
      return cmp (a.first, b.first) < 0; });
  h.resize (std::unique (g.begin (), g.end (), [] (const polar &a, const polar &b) { return cmp (a.first, b.first) == 0; }) - g.begin ());
  for (int i = 0; i < (int) h.size (); ++i)
    h[i] = g[i].second;
  int fore = 0, rear = -1;
  std::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; ret.pop_back (); }
    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; ret.pop_back (); }
  while (rear - fore > 1 && !turn_left (ret[rear], line_intersect (ret[fore], ret[fore + 1])))
    ++fore;
  if (rear - fore < 2) return std::vector <point> ();
  std::vector <point> ans;
  ans.resize (ret.size ());
  for (int i = 0; i < (int) ret.size (); ++i)
    ans[i] = line_intersect (ret[i], ret[(i + 1) % ret.size ()]);
  return ans; }
```

1.3 凸包

```
bool turn_left (const point &a, const point &b, const point &c) {
  return sgn (det (b - a, c - a)) >= 0; }
std::vector <point> convex_hull (std::vector <point> a) {
  int n = (int) a.size (), cnt = 0;
  std::sort (a.begin (), a.end ());
  std::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;
```

```cpp
  for (int i = n - 2; 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; }
  ret.pop_back (); return ret; }
```

1.4 直线与凸包交点

```cpp
// a 是顺时针凸包，i1 为 x 最小的点，j1 为 x 最大的点 需保证 j1 > i1
// n 是凸包上的点数，a 需复制多份或写循环数组类
int lowerBound(int le, int ri, const P & dir) {
  while (le < ri) {
    int mid((le + ri) / 2);
    if (sign((a[mid + 1] - a[mid]) * dir) <= 0) {
      le = mid + 1;
    } else ri = mid; }
  return le; }
int boundLower(int le, int ri, const P & s, const P & t) {
  while (le < ri) {
    int mid((le + ri + 1) / 2);
    if (sign((a[mid] - s) * (t - s)) <= 0) {
      le = mid;
    } else ri = mid - 1; }
  return le; }
void calc(P s, P t) {
  if(t < s) swap(t, s);
  int i3(lowerBound(i1, j1, t - s)); // 和上凸包的切点
  int j3(lowerBound(j1, i1 + n, s - t)); // 和下凸包的切点
  int i4(boundLower(i3, j3, s, t)); // 如果有交则是右侧的交点，与 a[i4]~a[i4+1] 相交 要判断是否有交的话 就手动 check 一下
  int j4(boundLower(j3, i3 + n, t, s)); // 如果有交左侧的交点，与 a[j4]~a[j4+1] 相交
   // 返回的下标不一定在 [0 ~ n-1] 内
}
```

1.5 点到凸包切线

```cpp
typedef vector<vector<P>> Convex;
#define sz(x) ((int) x.size())
int lb(P x, const vector<P> & v, int le, int ri, int sg) {
  if (le > ri) le = ri;
  int s(le), t(ri);
  while (le != ri) {
    int mid((le + ri) / 2);
    if (sign(det(v[mid] - x, v[mid + 1] - v[mid])) == sg)
      le = mid + 1; else ri = mid; }
  return le; } // le 即为下标，按需返回
// v[0] 为顺时针上凸壳，v[1] 为顺时针下凸壳，均允许起始两个点横坐标相同
// 返回值为真代表严格在凸包外，顺时针旋转在 d1 方向先碰到凸包
bool getTan(P x, const Convex & v, int & d1, int & d2) {
  if (x.x < v[0][0].x) {
    d1 = lb(x, v[0], 0, sz(v[0]) - 1, 1);
    d2 = lb(x, v[1], 0, sz(v[1]) - 1, -1) + (int) v[0].size() - 1;
    return true;
  } else if(x.x > v[0].back().x) {
    d1 = lb(x, v[1], 0, sz(v[1]) - 1, 1) + (int) v[0].size() - 1;
    d2 = lb(x, v[0], 0, sz(v[0]) - 1, -1);
    return true;
  } else {
    for(int d(0); d < 2; d++) {
      int id(lower_bound(v[d].begin(), v[d].end(), x, [&](const P & a, const P & b) { return d == 0 ? a < b : b < a; }) - v[d].begin());
      if (id && (id == sz(v[d]) || det(v[d][id - 1] - x, v[d][id] - x) > 0)) {
        d1 = lb(x, v[d], id, sz(v[d]) - 1, 1);
        d2 = lb(x, v[d], 0, id, -1);
        if (d) {
          d1 += (int) v[0].size() - 1;
          d2 += (int) v[0].size() - 1; }
        return true; } } }
  return false; }
```

1.6 闵可夫斯基和

```cpp
// cv[0..1] 为两个顺时针凸包，其中起点等于终点，求出的闵可夫斯基和不一定是严格凸包
int i[2] = {0, 0}, len[2] = {(int)cv[0].size() - 1, (int)cv[1].size() - 1};
vector<point> mnk;
mnk.push_back(cv[0][0] + cv[1][0]);
do {
  int d = (det(cv[0][i[0] + 1] - cv[0][i[0]], cv[1][i[1] + 1] - cv[1][i[1]]) >= 0);
  mnk.push_back(cv[d][i[d] + 1] - cv[d][i[d]] + mnk.back());
  i[d] = (i[d] + 1) % len[d];
} while(i[0] || i[1]);
```

1.7 Delaunay 三角剖分

```cpp
/* Delaunay Triangulation 随机增量算法 :
节点数至少为点数的 6 倍，空间消耗较大注意计算内存使用
建图的过程在 build 中，注意初始化内存池和初始三角形的坐标范围(Triangulation::LOTS)
Triangulation::find 返回包含某点的三角形
Triangulation::add_point 将某点加入三角剖分
某个 Triangle 在三角剖分中当且仅当它的 has_child 为 0
如果要找到三角形 u 的邻域，则枚举它的所有 u.edge[i].tri，该条边的两个点为 u.p[(i+1)%3], u.p[(i+2)%3] */
const int N = 100000 + 5, MAX_TRIS = N * 6;
const double EPSILON = 1e-6, PI = acos(-1.0);
struct Point {
  double x,y; Point():x(0),y(0){}
    Point(double x, double y):x(x),y(y){}
  bool operator ==(Point const& that)const {return x==that.x&&y==that.y;} };
inline double sqr(double x) { return x*x; }
double dist_sqr(Point const& a, Point const& b){return sqr(a.x-b.x)+sqr(a.y-b.y);}
bool in_circumcircle(Point const& p1, Point const& p2, Point const& p3, Point const& p4) {
  double u11 = p1.x-p4.x, u21 = p2.x-p4.x, u31 = p3.x-p4.x;
  double u12 = p1.y-p4.y, u22 = p2.y-p4.y, u32 = p3.y-p4.y;
  double u13 = sqr(p1.x)-sqr(p4.x) + sqr(p1.y) - sqr(p4.y);
  double u23 = sqr(p2.x)-sqr(p4.x) + sqr(p2.y) - sqr(p4.y);
  double u33 = sqr(p3.x)-sqr(p4.x) + sqr(p3.y) - sqr(p4.y);
  double det = -u13*u22*u31 + u12*u23*u31 + u13*u21*u32 - u11*u23*u32 - u12*u21*u33 + u11*u22*u33;
  return det > EPSILON; }
double side(Point const& a, Point const& b, Point const& p) { return (b.x-a.x)*(p.y-a.y) - (b.y-a.y)*(p.x-a.x);}
typedef int SideRef; struct Triangle; typedef Triangle* TriangleRef;
struct Edge {
  TriangleRef tri; SideRef side; Edge() : tri(0), side(0) {}
  Edge(TriangleRef tri, SideRef side) : tri(tri), side(side) {} };
struct Triangle {
  Point p[3];Edge edge[3];TriangleRef child[3]; Triangle(){}
  Triangle(Point const& p0,Point const& p1,Point const& p2){
    p[0] = p0; p[1] = p1; p[2] = p2;
        child[0] = child[1] = child[2] = 0; }
  bool has_child() const { return child[0] != 0; }
  int num_child() const {
    return child[0] == 0 ? 0
      : child[1] == 0 ? 1
      : child[2] == 0 ? 2 : 3; }
  bool contains(Point const& q) const {
    double a=side(p[0],p[1],q), b=side(p[1],p[2],q), c=side(p[2],p[0],q);
    return a >= -EPSILON && b >= -EPSILON && c >= -EPSILON; }
} triange_pool[MAX_TRIS], *tot_triangles;
void set_edge(Edge a, Edge b) {
  if (a.tri) a.tri->edge[a.side] = b;
  if (b.tri) b.tri->edge[b.side] = a; }
class Triangulation {
  public:
    Triangulation() {
      const double LOTS = 1e6;
      the_root = new(tot_triangles++) Triangle (Point(-LOTS,-LOTS), Point(+LOTS,-LOTS), Point(0,+LOTS)); }
    TriangleRef find(Point p) const { return find(the_root,p); }
    void add_point(Point const& p) { add_point(find(the_root,p),p); }
  private:
    TriangleRef the_root;
    static TriangleRef find(TriangleRef root,Point const& p){
      for( ; ; ) {
        if (!root->has_child()) return root;
        else for (int i = 0; i < 3 && root->child[i] ; ++i)
            if (root->child[i]->contains(p))
              {root = root->child[i]; break;} } }
    void add_point(TriangleRef root, Point const& p) {
      TriangleRef tab,tbc,tca;
      tab = new(tot_triangles++) Triangle(root->p[0], root->p[1], p);
      tbc = new(tot_triangles++) Triangle(root->p[1], root->p[2], p);
      tca = new(tot_triangles++) Triangle(root->p[2], root->p[0], p);
      set_edge(Edge(tab,0),Edge(tbc,1)); set_edge(Edge(tbc,0),Edge(tca,1));
      set_edge(Edge(tca,0),Edge(tab,1)); set_edge(Edge(tab,2),root->edge[2]);
      set_edge(Edge(tbc,2),root->edge[0]); set_edge(Edge(tca,2),root->edge[1]);
      root->child[0]=tab;root->child[1]=tbc;root->child[2]=tca;
      flip(tab,2); flip(tbc,2); flip(tca,2); }
    void flip(TriangleRef tri, SideRef pi) {
      TriangleRef trj = tri->edge[pi].tri; int pj = tri->edge[pi].side;
      if(!trj || !in_circumcircle(tri->p[0],tri->p[1],tri->p[2],trj->p[pj])) return;
      TriangleRef trk = new(tot_triangles++) Triangle(tri->p[(pi+1)%3], trj->p[pj], tri->p[pi]);
      TriangleRef trl = new(tot_triangles++) Triangle(trj->p[(pj+1)%3], tri->p[pi], trj->p[pj]);
      set_edge(Edge(trk,0), Edge(trl,0));
      set_edge(Edge(trk,1), tri->edge[(pi+2)%3]); set_edge(Edge(trk,2), trj->edge[(pj+1)%3]);
      set_edge(Edge(trl,1), trj->edge[(pj+2)%3]); set_edge(Edge(trl,2), tri->edge[(pi+1)%3]);
      tri->child[0]=trk; tri->child[1]=trl; tri->child[2]=0;
      trj->child[0]=trk; trj->child[1]=trl; trj->child[2]=0;
      flip(trk,1); flip(trk,2); flip(trl,1); flip(trl,2); } };
int n; Point ps[N];
void build(){
  tot_triangles = triange_pool; cin >> n;
  for(int i = 0; i < n; ++ i) scanf("%lf%lf",&ps[i].x,&ps[i].y);
  random_shuffle(ps, ps + n); Triangulation tri;
  for(int i = 0; i < n; ++ i) tri.add_point(ps[i]); }
//The Euclidean minimum spanning tree of a set of points is a subset of the Delaunay triangulation of the same points.
//Connecting the centers of the circumcircles produces the Voronoi diagram.
//No point in P is inside the circumcircle of any triangle.
//Maximize the minimum angle of all the angles of the triangles.
```

1.8 三角形 与 费马点

```cpp
point incenter (const point &a, const point &b, const point &c) {
    double p = dis (a, b) + dis (b, c) + dis (c, a);
    return (a * dis (b, c) + b * dis (c, a) + c * dis (a, b)) / p; }
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; }
point orthocenter (const point &a, const point &b, const point &c) {
    return a + b + c - circumcenter (a, b, c) * 2.0; }
point fermat_point (const point &a, const point &b, const point &c) {
    if (a == b) return a; if (b == c) return b;
    if (c == a) return c;
    double ab = dis (a, b), bc = dis (b, c), ca = dis (c, a);
    double cosa = dot (b - a, c - a) / ab / ca;
    double cosb = dot (a - b, c - b) / ab / bc;
    double cosc = dot (b - c, a - c) / ca / bc;
    double sq3 = PI / 3.0; point mid;
    if (sgn (cosa + 0.5) < 0) mid = a;
    else if (sgn (cosb + 0.5) < 0) mid = b;
    else if (sgn (cosc + 0.5) < 0) mid = c;
    else if (sgn (det (b - a, c - a)) < 0)
        mid = line_intersect (line (a, b + (c - b).rot (sq3)), line (b, c + (a - c).rot (sq3)));
    else
        mid = line_intersect (line (a, c + (b - c).rot (sq3)), line (c, b + (a - b).rot (sq3)));
    return mid; }
```

1.9 圆并

```cpp
int C; circle c[MAXN]; double area[MAXN];
struct event {
    point p; double ang; int delta;
    event (point p = point (), double ang = 0, int delta = 0) : p(p), ang(ang), delta(delta) {}
    bool operator < (const event &a) { return ang < a.ang; } };
void addevent(const circle &a, const circle &b, std::vector<event> &evt, int &cnt) {
    double d2 = (a.c - b.c).norm2(), dRatio = ((a.r - b.r) * (a.r + b.r) / d2 + 1) / 2,
        pRatio = sqrt (std::max (0., -(d2 - sqr(a.r - b.r)) * (d2 - sqr(a.r + b.r)) / (d2 * d2 * 4)));
    point d = b.c - a.c, p = d.rot(PI / 2),
        q0 = a.c + d * dRatio + p * pRatio,
        q1 = a.c + d * dRatio - p * pRatio;
    double ang0 = atan2 ((q0 - a.c).y, (q0 - a.c).x), ang1 = atan2 ((q1 - a.c).x, (q1 - a.c).y);
    evt.emplace_back(q1,ang1,1); evt.emplace_back(q0,ang0,-1);
    cnt += ang1 > ang0; }
bool issame(const circle &a, const circle &b) {
    return sgn((a.c-b.c).norm()) == 0 && sgn(a.r-b.r) == 0; }
bool overlap(const circle &a, const circle &b) {
    return sgn(a.r - b.r - (a.c - b.c).norm()) >= 0; }
bool intersect(const circle &a, const circle &b) {
    return sgn((a.c - b.c).norm() - a.r - b.r) < 0; }
void solve() {
    std::fill (area, area + C + 2, 0);
    for (int i = 0; i < C; ++i) { int cnt = 1;
        std::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.empty()) area[cnt] += PI * c[i].r * c[i].r;
        else {
            std::sort(evt.begin(), evt.end());
            evt.push_back(evt.front());
            for (int j = 0; j + 1 < (int)evt.size(); ++j) {
                cnt += evt[j].delta;
                area[cnt] += det(evt[j].p,evt[j + 1].p) / 2;
                double ang = evt[j + 1].ang - evt[j].ang;
                if (ang < 0) ang += PI * 2;
                area[cnt] += ang * c[i].r * c[i].r / 2 - sin(ang) * c[i].r * c[i].r / 2; } } } }
```

1.10 最小覆盖圆

```
circle minimum_circle (std::vector <point> p) { circle ret;
    std::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; }
```

1.11 多边形与圆交

```
double sector_area (const point &a, const point &b, const double &r) {
    double c = (2.0 * r * r - (a - b).norm2 ()) / (2.0 * r * r);
    double al = acos (c);
    return r * r * al / 2.0; }
double area(const point &a,const point &b,const double &r) {
    double dA = dot (a, a), dB = dot (b, b), dC = point_to_segment (point (), line (a, b)), ans = 0.0;
    if (sgn (dA - r * r) <= 0 && sgn (dB - r * r) <= 0) return det (a, b) / 2.0;
    point tA = a.unit () * r;
    point tB = b.unit () * r;
    if (sgn (dC - r) > 0) return sector_area (tA, tB, r);
    std::pair <point, point> ret = line_circle_intersect (line (a, b), circle (point (), r));
    if (sgn (dA - r * r) > 0 && sgn (dB - r * r) > 0) {
        ans += sector_area (tA, ret.first, r);
        ans += det (ret.first, ret.second) / 2.0;
        ans += sector_area (ret.second, tB, r);
        return ans; }
    if (sgn (dA - r * r) > 0)
        return det (ret.first, b) / 2.0 + sector_area (tA, ret.first, r);
    else
        return det (a, ret.second) / 2.0 + sector_area (ret.second, tB, r); }
double solve(const std::vector<point> &p, const circle &c) {
    double ret = 0.0;
    for (int i = 0; i < (int) p.size (); ++i) {
        int s = sgn (det (p[i] - c.c, p[ (i + 1) % p.size ()] - c.c));
        if (s > 0)
            ret += area (p[i] - c.c, p[ (i + 1) % p.size ()] - c.c, c.r);
        else
            ret -= area (p[ (i + 1) % p.size ()] - c.c, p[i] - c.c, c.r); }
    return fabs (ret); }
```

1.12 阿波罗尼茨圆

```
硬币问题：易知两两相切的圆半径为 r1, r2, r3, 求与他们都相切的圆的半径 r4
分母取负号，答案再取绝对值，为外切圆半径
分母取正号为内切圆半径
// $r_4^{\pm} = \frac{r_1 r_2 r_3}{r_1 r_2 + r_1 r_3 + r_2 r_3 \pm 2\sqrt{r_1 r_2 r_3 (r_1 + r_2 + r_3)}}$
```

1.13 圆幂 圆反演 根轴

圆幂：半径为 R 的圆 O，任意一点 P 到 O 的幂为 $h = OP^2 - R^2$
圆幂定理：过 P 的直线交圆在 A 和 B 两点，则 $PA \cdot PB = |h|$
根轴：到两圆等幂点的轨迹是一条垂直于连心线的直线
反演：已知一圆 C，圆心为 O，半径为 r，如果 P与 P'在过圆心 O的直线上，且 $OP \cdot OP' = r^2$，则称 P与 P'关于 O互为反演．一般 C取单位圆．
反演的性质：
不过反演中心的直线反形是过反演中心的圆，反之亦然．
不过反演中心的圆，它的反形是一个不过反演中心的圆．
两条直线在交点 A的夹角，等于它们的反形在相应点 A'的夹角，但方向相反．
两个相交圆周在交点 A的夹角等于它们的反形在相应点 A'的夹角，但方向相反．
直线和圆周在交点 A的夹角等于它们的反演图形在相应点 A'的夹角，但方向相反．
正交圆反形也正交．相切圆反形也相切，当切点为反演中心时，反形为两条平行线．

1.14 三维绕轴旋转 三维基础操作

```
/* 大拇指指向x轴正方向时□4指弯曲由y轴正方向指向z轴正方向
 大拇指沿着原点到点(x, y, z)的向量□4指弯曲方向旋转w度 */
/* (x, y, z) * A = (x_new, y_new, z_new), 行向量右乘转移矩阵 */
void calc(D x, D y, D z, D w) { // 三维绕轴旋转
    w = w * pi / 180;
    memset(a, 0, sizeof(a));
    s1 = x * x + y * y + z * z;
    a[0][0] = ((y*y+z*z)*cos(w)+x*x)/s1; a[0][1] = x*y*(1-cos(w))/s1+z*sin(w)/sqrt(s1); a[0][2] = x*z*(1-cos(w))/s1-y*sin(w)/sqrt(s1);
    a[1][0] = x*y*(1-cos(w))/s1-z*sin(w)/sqrt(s1); a[1][1] = ((x*x+z*z)*cos(w)+y*y)/s1; a[1][2] = y*z*(1-cos(w))/s1+x*sin(w)/sqrt(s1);
    a[2][0] = x*z*(1-cos(w))/s1+y*sin(w)/sqrt(s1); a[2][1] = y*z*(1-cos(w))/s1-x*sin(w)/sqrt(s1); a[2][2] = ((x*x+y*y)*cos(w)+z*z)/s1;
}
point3D cross (const point3D & a, const point3D & b) {
    return point3D(a.y * b.z - a.z * b.y, a.z * b.x - a.x * b.z, a.x * b.y - a.y * b.x); }
double mix(point3D a, point3D b, point3D c) {
    return dot(cross(a, b), c); }
struct Line { point3D s, t; };
struct Plane { // nor 为单位法向量, 离原点距离 m
    point3D nor; double m;
    Plane(point3D r, point3D a) : nor(r){
        nor = 1 / r.len() * r;
        m = dot(nor, a); } };
// 以下函数注意除以0的情况
// 点到平面投影
point3D project_to_plane(point3D a, Plane b) {
return a + (b.m - dot(a, b.nor)) * b.nor; }
// 点到直线投影
point3D project_to_line(point3D a, Line b) {
return b.s + dot(a - b.s, b.t - b.s) / dot(b.t - b.s, b.t - b.s) * (b.t - b.s); }
// 直线与直线最近点
pair<point3D, point3D> closest_two_lines(Line x, Line y) {
double a = dot(x.t - x.s, x.t - x.s);
double b = dot(x.t - x.s, y.t - y.s);
double e = dot(y.t - y.s, y.t - y.s);
double d = a*e - b*b; point3D r = x.s - y.s;
double c = dot(x.t - x.s, r), f = dot(y.t - y.s, r);
double s = (b*f - c*e) / d, t = (a*f - c*b) / d;
return {x.s + s*(x.t - x.s), y.s + t*(y.t - y.s)}; }
// 直线与平面交点
point3D intersect(Plane a, Line b) {
double t = dot(a.nor, a.m * a.nor - b.s) / dot(a.nor, b.t - b.s);
```

```cpp
return b.s + t * (b.t - b.s); }
// 平面与平面求交线
Line intersect(Plane a, Plane b) {
point3D d=cross(a.nor,b.nor), d2=cross(b.nor,d);
double t = dot(d2, a.nor);
point3D s = 1 / t * (a.m - dot(b.m * b.nor, a.nor)) * d2 + b.m * b.nor;
return (Line) {s, s + d}; }
// 三个平面求交点
point3D intersect(Plane a, Plane b, Plane c) {
return intersect(a, intersect(b, c));
point3D c1 (a.nor.x, b.nor.x, c.nor.x);
point3D c2 (a.nor.y, b.nor.y, c.nor.y);
point3D c3 (a.nor.z, b.nor.z, c.nor.z);
point3D c4 (a.m, b.m, c.m);
return 1 / mix(c1, c2, c3) * point3D(mix(c4, c2, c3), mix(c1, c4, c3), mix(c1, c2, c4)); }
```

1.15 最小覆盖球

```cpp
vector<point3D> vec;
Circle calc() {
    if(vec.empty()) { return Circle(point3D(0, 0, 0), 0);
    }else if(1 == (int)vec.size()) {return Circle(vec[0], 0);
    }else if(2 == (int)vec.size()) {
        return Circle(0.5 * (vec[0] + vec[1]), 0.5 * (vec[0] - vec[1]).len());
    }else if(3 == (int)vec.size()) {
        double r = (vec[0] - vec[1]).len() * (vec[1] - vec[2]).len() * (vec[2] - vec[0]).len() / 2 / fabs(cross(vec[0] - vec[2], vec[1] - vec[2]).len());
        Plane ppp1 = Plane(vec[1] - vec[0], 0.5 * (vec[1] + vec[0]));
        return Circle(intersect(Plane(vec[1] - vec[0], 0.5 * (vec[1] + vec[0])), Plane(vec[2] - vec[1], 0.5 * (vec[2] + vec[1])), Plane(cross(vec[1] - vec[0], vec[2] - vec[0]), vec[0])), r);
    }else {
        point3D o(intersect(Plane(vec[1] - vec[0], 0.5 * (vec[1] + vec[0])), Plane(vec[2] - vec[0], 0.5 * (vec[2] + vec[0])), Plane(vec[3] - vec[0], 0.5 * (vec[3] + vec[0]))));
        return Circle(o, (o - vec[0]).len()); } }
Circle miniBall(int n) {
    Circle res(calc());
    for(int i(0); i < n; i++) {
        if(!in_circle(a[i], res)) { vec.push_back(a[i]);
            res = miniBall(i); vec.pop_back();
            if(i) { point3D tmp(a[i]);
                memmove(a + 1, a, sizeof(point3D) * i);
                a[0] = tmp; } } }
    return res; }
int main() {
    int n; scanf("%d", &n);
    for(int i(0); i < n; i++) a[i].scan();
    sort(a, a + n); n = unique(a, a + n) - a;
    vec.clear(); random_shuffle(a, a + n);
    printf("%.10f\n", miniBall(n).r); }
```

1.16 球面基础

球面距离：连接球面两点的大圆劣弧（所有曲线中最短）
球面角：球面两个大圆弧所在半平面形成的二面角
球面凸多边形：把一个球面多边形任意一边向两方无限延长成大圆，其余边都在此大圆的同旁.
球面角盈E：球面凸n边形的内角和与$(n-2)\pi$的差
离北极夹角θ，距离h的球冠：$S = 2\pi Rh = 2\pi R^2(1-\cos\theta)$，$V = \frac{\pi h^2}{3}(3R-h)$
球面凸n边形面积：$S = ER^2$

1.17 经纬度球面距离

```cpp
// lontitude 经度范围：±π，latitude 纬度范围：±π/2
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)); }
```

1.18 最近点对

```cpp
double solve(point *p, int l, int r) { // 左闭右开 返回距离平方
    if(l + 1 >= r) return INF;
    int m = (l + r) / 2; double mx = p[m].x; vector <point> v;
    double ret = min(solve(p, l, m), solve(p, m, r));
    for(int i = l; i < r; i ++)
        if(sqr(p[i].x - mx) < ret) v.push_back(p[i]);
    sort(v.begin(), v.end(), by_y);
    for(int i = 0; i < v.size(); i ++)
        for(int j = i + 1; j < v.size(); j ++) {
            if(sqr(v[i].y - v[j].y) > ret) break;
            ret = min(ret, (v[i] - v[j]).len2()); }
    return ret; } // 需先对p[]按x进行排序
```

1.19 三维凸包

```cpp
int mark[N][N], cnt;
D mix(const Point & a, const Point & b, const Point & c) { return a.dot(b.cross(c)); }
double volume(int a, int b, int c, int d) { return mix(info[b] - info[a], info[c] - info[a], info[d] - info[a]); }
typedef array<int, 3> Face; vector<Face> face;
inline void insert(int a, int b, int c) { face.push_back({a, b, c}); }
void add(int v) {
    vector<Face> tmp; int a, b, c; cnt++;
    for(auto f : face)
        if(sign(volume(v, f[0], f[1], f[2])) < 0)
            for(int i : f) for(int j : f) mark[i][j] = cnt;
        else tmp.push_back(f);
    face = tmp;
    for(int i(0); i < (int)tmp.size(); i++) {
        a = face[i][0]; b = face[i][1]; c = face[i][2];
        if(mark[a][b] == cnt) insert(b, a, v);
        if(mark[b][c] == cnt) insert(c, b, v);
        if(mark[c][a] == cnt) insert(a, c, v); } }
int Find(int n) {
    for(int i(2); i < n; i++) {
        Point ndir=(info[0]-info[i]).cross(info[1]-info[i]);
        if(ndir==Point(0,0,0))continue;swap(info[i],info[2]);
        for(int j = i + 1; j < n; j++) if(sign(volume(0, 1, 2, j)) != 0) {
            swap(info[j], info[3]); insert(0, 1, 2), insert(0, 2, 1); return 1; } } }
int main() {
    int n; scanf("%d", &n);
    for(int i(0); i < n; i++) info[i].scan();
    random_shuffle(info, info + n);
    Find(n);
    for(int i = 3; i < n; i++) add(i); }
```

1.20 长方体表面两点最短距离

```cpp
int r;
void turn(int i, int j, int x, int y, int z,int x0, int y0, int L, int W, int H) {
    if (z==0) { int R = x*x+y*y; if (R<r) r=R;
    } else {
        if(i>=0 && i< 2) turn(i+1, j, x0+L+z, y, x0+L-x, x0+L, y0, H, W, L);
        if(j>=0 && j< 2) turn(i, j+1, x, y0+W+z, y0+W-y, x0, y0+W, L, H, W);
        if(i<=0 && i>-2) turn(i-1, j, x0-z, y, x-x0, x0-H, y0, H, W, L);
        if(j<=0 && j>-2) turn(i, j-1, x, y0-z, y-y0, x0, y0-H, L, H, W);
    } }
int main(){
    int L, H, W, x1, y1, z1, x2, y2, z2;
cin >> L >> W >> H >> x1 >> y1 >> z1 >> x2 >> y2 >> z2;
if (z1!=0 && z1!=H) if (y1==0 || y1==W)
    swap(y1,z1), std::swap(y2,z2), std::swap(W,H);
else swap(x1,z1), std::swap(x2,z2), std::swap(L,H);
if (z1==H) z1=0, z2=H-z2;
r=0x3fffffff;
turn(0,0,x2-x1,y2-y1,z2,-x1,-y1,L,W,H);
cout<<r<<endl; }
```

1.21 相关公式

1.21.1 Heron's Formula

$$S = \sqrt{p(p-a)(p-b)(p-c)}$$
$$p = \frac{a+b+c}{2}$$

1.21.2 四面体内接球球心

假设s_i是第i个顶点相对面的面积 则有

$$\begin{cases} x &= \dfrac{s_1x_1+s_2x_2+s_3x_3+s_4x_4}{s_1+s_2+s_3+s_4} \\ y &= \dfrac{s_1y_1+s_2y_2+s_3y_3+s_4y_4}{s_1+s_2+s_3+s_4} \\ z &= \dfrac{s_1z_1+s_2z_2+s_3z_3+s_4z_4}{s_1+s_2+s_3+s_4} \end{cases}$$

体积可以使用1/6混合积求，内接球半径为

$$r = \frac{3V}{s_1+s_2+s_3+s_4}$$

1.21.3 三角形内心

$$\frac{a\vec{A}+b\vec{B}+c\vec{C}}{a+b+c}$$

1.21.4 三角形外心

$$\frac{\vec{A}+\vec{B}-\frac{\overrightarrow{BC}\cdot\overrightarrow{CA}}{\overrightarrow{AB}\times\overrightarrow{BC}}\overrightarrow{AB}^T}{2}$$

1.21.5 三角形垂心

$$\vec{H} = 3\vec{G} - 2\vec{O}$$

1.21.6 三角形偏心

$$\frac{-a\vec{A}+b\vec{B}+c\vec{C}}{-a+b+c}$$

剩余两点的同理.

1.21.7 三角形内接外接圆半径

$$r = \frac{2S}{a+b+c},\ R = \frac{abc}{4S}$$

1.21.8 Pick's Theorem

$$S = I + \frac{B}{2} - 1$$

S is the area of lattice polygon, I is the number of lattice interior points, and B is the number of lattice boundary points.

1.21.9 Euler's Formula

For convex polyhedron: $V - E + F = 2$.
For planar graph: $|F| = |E| - |V| + n + 1$, n denotes the number of connected components.

1.22 三角公式

$$\sin(a\pm b) = \sin a\cos b \pm \cos a \sin b$$
$$\cos(a\pm b) = \cos a\cos b \mp \sin a \sin b$$
$$\tan(a\pm b) = \frac{\tan(a)\pm\tan(b)}{1\mp\tan(a)\tan(b)}$$
$$\tan(a)\pm\tan(b) = \frac{\sin(a\pm b)}{\cos(a)\cos(b)}$$
$$\sin(a)+\sin(b) = 2\sin(\frac{a+b}{2})\cos(\frac{a-b}{2})$$
$$\sin(a)-\sin(b) = 2\cos(\frac{a+b}{2})\sin(\frac{a-b}{2})$$
$$\cos(a)+\cos(b) = 2\cos(\frac{a+b}{2})\cos(\frac{a-b}{2})$$
$$\cos(a)-\cos(b) = -2\sin(\frac{a+b}{2})\sin(\frac{a-b}{2})$$

$$\sin(na) = n\cos^{n-1}a\sin a - \binom{n}{3}\cos^{n-3}a\sin^3 a + \binom{n}{5}\cos^{n-5}a\sin^5 a - \dots$$

$$\cos(na) = \cos^n a - \binom{n}{2}\cos^{n-2}a\sin^2 a + \binom{n}{4}\cos^{n-4}a\sin^4 a - \dots$$

1.22.1 超球坐标系

$$\begin{aligned} x_1 &= r\cos(\phi_1) \\ x_2 &= r\sin(\phi_1)\cos(\phi_2) \\ \cdots \\ x_{n-1} &= r\sin(\phi_1)\cdots\sin(\phi_{n-2})\cos(\phi_{n-1}) \\ x_n &= r\sin(\phi_1)\cdots\sin(\phi_{n-2})\sin(\phi_{n-1}) \\ \phi_{n-1} &\in [0,2\pi] \\ \forall i = 1..n-1\phi_i &\in [0,\pi] \end{aligned}$$

1.22.2 三维旋转公式

绕着$(0,0,0)-(ux,uy,uz)$旋转θ, (ux,uy,uz) 是单位向量

$$R = \begin{matrix} \cos\theta+u_x^2(1-\cos\theta) & u_xu_y(1-\cos\theta)-u_z\sin\theta & u_xu_z(1-\cos\theta)+u_y\sin\theta \\ u_yu_x(1-\cos\theta)+u_z\sin\theta & \cos\theta+u_y^2(1-\cos\theta) & u_yu_z(1-\cos\theta)-u_x\sin\theta \\ u_zu_x(1-\cos\theta)-u_y\sin\theta & u_zu_y(1-\cos\theta)+u_x\sin\theta & \cos\theta+u_z^2(1-\cos\theta) \end{matrix}.$$

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = R\begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

1.22.3 立体角公式

$$\phi\text{：二面角}$$
$$\Omega = (\phi_{ab}+\phi_{bc}+\phi_{ac})\ \mathrm{rad} - \pi\,\mathrm{sr}$$

$$\tan\left(\frac{1}{2}\Omega/\mathrm{rad}\right) = \frac{\left|\vec{a}\ \vec{b}\ \vec{c}\right|}{abc+\left(\vec{a}\cdot\vec{b}\right)c+(\vec{a}\cdot\vec{c})b+\left(\vec{b}\cdot\vec{c}\right)a}$$

$$\theta_s = \frac{\theta_a+\theta_b+\theta_c}{2}$$

1.22.4 常用体积公式

- Pyramid $V = \frac{1}{3}Sh$.
- Sphere $V = \frac{4}{3}\pi R^3$.
- Frustum $V = \frac{1}{3}h(S_1+\sqrt{S_1S_2}+S_2)$.
- Ellipsoid $V = \frac{4}{3}\pi abc$.

1.22.5 高维球体积

$$V_2 = \pi R^2,\ S_2 = 2\pi R$$
$$V_3 = \frac{4}{3}\pi R^3,\ S_3 = 4\pi R^2$$
$$V_4 = \frac{1}{2}\pi^2R^4,\ S_4 = 2\pi^2R^3$$
$$\text{Generally}, V_n = \frac{2\pi}{n}V_{n-2},\ S_{n-1} = \frac{2\pi}{n-2}S_{n-3}$$
$$\text{Where}, S_0 = 2,\ V_1 = 2,\ S_1 = 2\pi,\ V_2 = \pi$$

2. Data Structure

2.1 KD 树

```
//P-节点类  d[2]-坐标 mi[2]-最小坐标 mx[2]-最大坐标 l,r-左右儿
  ↪ 子 s-和  v-值 lz-懒惰标记 sz-子树大小
int rt,X0,X1,Y0,Y1,X,Y,O,an;
struct P{int d[2],mi[2],mx[2],l,r,s,v,lz,sz;}T[111111];
bool cmp(P a,P b) {
    return a.d[O]<b.d[O]||a.d[O]==b.d[O]&&a.d[O^1]<b.d[O^1];}
void ADD(int k,int z) {
    if(!k)return;T[k].v+=z;T[k].s+=T[k].sz*z;T[k].lz+=z;}
void pd(int k) { if(T[k].lz)
    ADD(T[k].l,T[k].lz),ADD(T[k].r,T[k].lz),T[k].lz=0;}
int bd(int l,int r,int o){//将[l,r]建树, 维度是o, 返回[l,r]的
  ↪ 根
    O=o;int k=l+r>>1,i;nth_element(T+l+1,T+k+1,T+r+1,cmp);
    for(i=0;i<2;i++)T[k].mi[i]=T[k].mx[i]=T[k].d[i];
    T[k].lz=T[k].v=T[k].s=0;
    if(l<k)for(O=T[k].l=bd(l,k-1,o^1),i=0;i<2;i++)
        Min(T[k].mi[i],T[O].mi[i]), Max(T[k].mx[i],T[O].mx[i]);
    if(k<r)for(O=T[k].r=bd(k+1,r,o^1),i=0;i<2;i++)
        Min(T[k].mi[i],T[O].mi[i]), Max(T[k].mx[i],T[O].mx[i]);
    T[k].sz=T[T[k].l].sz+T[T[k].r].sz+1;return k;
}void add(int k,int z){//范围+z, 范围为[X0,X1][Y0,Y1]
    if(X0>T[k].mx[0] || X1<T[k].mi[0] || Y0>T[k].mx[1] ||
      ↪ Y1<T[k].mi[1]) return;
    if(X0<=T[k].mi[0] && X1>=T[k].mx[0] && Y0<=T[k].mi[1] &&
      ↪ Y1>=T[k].mx[1]) {ADD(k,z);return;}
    if(X0<=T[k].d[0] && T[k].d[0]<=X1 && Y0<=T[k].d[1] &&
      ↪ T[k].d[1]<=Y1) T[k].v+=z;
    pd(k);add(T[k].l,z);add(T[k].r,z);
    T[k].s=T[k].v+T[T[k].l].s+T[T[k].r].s;
}int qu(int k){//询问范围和
    if(X0>T[k].mx[0] || X1<T[k].mi[0] || Y0>T[k].mx[1] ||
      ↪ Y1<T[k].mi[1]) return 0;
    if(X0<=T[k].mi[0] && X1>=T[k].mx[0] && Y0<=T[k].mi[1] &&
      ↪ Y1>=T[k].mx[1]) return T[k].s;
    int an=0;
```

```cpp
    if(X0<=T[k].d[0] && T[k].d[0]<=X1 && Y0<=T[k].d[1] && T[k].d[1]<=Y1) an=T[k].v;
    pd(k); return an+qu(T[k].l)+qu(T[k].r);
}//外部将坐标存入T[i].d[2]后调用rt=bd(1,n,0)
int dis(P a){return max(a.mi[0]-X,0) + max(X-a.mx[0],0) + max(a.mi[1]-Y,0) + max(Y-a.mx[1],0);}
void fdmin(int k,int o){//找离当前点(X,Y)最近的点
    int d=abs(T[k].d[0]-X)+abs(T[k].d[1]-Y),dl=inf,dr=inf;
    an=min(an,d);
    if(T[k].l)dl=dis(T[T[k].l]);
    if(T[k].r)dr=dis(T[T[k].r]);
    if(dl<dr){if(dl<an)fdmin(T[k].l,o^1); if(dr<an)fdmin(T[k].r,o^1);
    }else{if(dr<an)fdmin(T[k].r,o^1); if(dl<an)fdmin(T[k].l,o^1);}}
```

2.2 LCT 动态树

```cpp
bool ir(int x){return c[F[x]][0]!=x&&c[F[x]][1]!=x;}
void rev(int x){swap(c[x][0],c[x][1]);rv[x]^=1;}
void pd(int x){if(rv[x])rev(c[x][0]),rev(c[x][1]),rv[x]=0;}
void R(int x){
    int y=F[x],k=c[y][0]==x;F[c[y][!k]=c[x][k]]=y;
    F[x]=F[y];if(!ir(y))c[F[y]][c[F[y]][1]==y]=x;
    F[c[x][k]=y]=x;ps(y);
}void dw(int x){if(!ir(x))dw(F[x]);pd(x);}
void sy(int x){for(dw(x);!ir(x);R(x))if(!ir(F[x]))
R(c[F[x]][0]==x^c[F[F[x]]][0]==F[x]?x:F[x]);ps(x);}
int fd(int x){for(acs(x),sy(x);c[x][0];x=c[x][0]);return x;}
void acs(int x){for(o=0;x;c[x][1]=o,ps(x),o=x,x=F[x])sy(x);}
void mrt(int x){acs(x);sy(x);rev(x);}
// 无向图 lk, ct, qu
void lk(int x,int y){mrt(x);F[x]=y;}
void ct(int x,int y){mrt(x);acs(y);sy(y);c[y][0]=F[x]=0;}
int qu(int x,int y){mrt(x);acs(y);sy(y);return V[y];}
// 有向图 lk, ct, qu
void lk(int x,int y){acs(x);sy(x);F[x]=y;}
void ct(int x){acs(x);sy(x);F[c[x][0]]=0;c[x][0]=0;ps(x);}
int qu(int x){acs(x);sy(x);return V[x];}
void add(int x,int y,int z){mrt(x);acs(y);sy(y);ADD(y,z);}
```

2.3 可持久化平衡树

```cpp
int Copy(int x){// 可持久化
    id++;sz[id]=sz[x];L[id]=L[x];R[id]=R[x];
    v[id]=v[x];return id;
}int merge(int x,int y){
    // 合并 x 和 y 两颗子树，可持久化到 z 中
    if(!x||!y)return x+y;int z;
    int o=rand()%(sz[x]+sz[y]);// 注意 rand 上限
    if(o<sz[x])z=Copy(y),L[z]=merge(x,L[y]);
    else z=Copy(x),R[z]=merge(R[x],y);
    ps(z);return z;
}void split(int x,int&y,int&z,int k){
    // 将 x 分成 y 和 z 两颗子树，y 的大小为 k
    y=z=0;if(!x)return;
    if(sz[L[x]]>=k)z=Copy(x),split(L[x],y,L[z],k),ps(z);
    else y=Copy(x),split(R[x],R[y],z,k-sz[L[x]]-1),ps(y);}
```

2.4 动态斜率优化

```cpp
const LL is_query = - (1LL << 62);
struct Line { LL k, b; // kx + b
mutable function<const Line*()> succ;
bool operator < (const Line &rhs) const {
    if (rhs.b != is_query) return k < rhs.k;
    const Line* s = succ();
    if (!s) return 0; LL x = rhs.k;
    // 根据范围和有无 __int128 调整 是否使用 1.0L* 下同
    return 1.0L * b - s->b < 1.0L * (s->k - k) * x; } };
struct HullDynamic : public multiset<Line> {
bool bad(iterator y) {    // upper hull for maximum
    auto z = next(y); if (y == begin()) {
        if (z == end()) return 0;
        return y->k == z->k && y->b <= z->b; }
    auto x = prev(y);
    if (z == end()) return y->k == x->k && y->b <= x->b;
    return 1.0L * (x->b - y->b) * (z->k - y->k) >= 1.0L * (y->b - z->b) * (y->k - x->k); }
void insert_line(LL k, LL b) {
    auto y = insert({k, b});
    y->succ = [=]{return next(y) == end() ? 0 : &*next(y);};
    if (bad(y)) { erase(y); return; }
    while (next(y) != end() && bad(next(y))) erase(next(y));
    while (y != begin() && bad(prev(y))) erase(prev(y)); }
LL eval(LL x) { Line u = {x, is_query};
    auto l = *lower_bound(u); return l.k * x + l.b; }};
```

3. Tree & Graph

3.1 Stoer-Wagner 无向图最小割(树)

```cpp
int d[N];bool v[N],g[N];
int get(int&s,int&t){
    CL(d);CL(v);int i,j,k,an,mx;
    fr(i,1,n){  k=mx=-1;
        fr(j,1,n)if(!g[j]&&!v[j]&&d[j]>mx)k=j,mx=d[j];
        if(k==-1)return an;
        s=t;t=k;an=mx;v[k]=1;
        fr(j,1,n)if(!g[j]&&!v[j])d[j]+=w[k][j];
    }return an;}
int mincut(int n,int w[N][N]){
//n 为点数，w[i][j] 为 i 到 j 的流量，返回无向图所有点对最小割之和
    int ans=0,i,j,s,t,x,y,z;
    fr(i,1,n-1){
        ans=min(ans,get(s,t));
        g[t]=1;if(!ans)break;
        fr(j,1,n)if(!g[j])w[s][j]=(w[j][s]+=w[j][t]);
    }return ans;}
// 无向图最小割树
void fz(int l,int r){// 左闭右闭，分治建图
    if(l==r)return;S=a[l];T=a[r];
    reset();// 将所有边权复原
    flow(S,T);// 做网络流
    dfs(S);// 找割集，v[x]=1 属于 S 集，否则属于 T 集
    ADD(S,T,fl);// 在最小割树中建边
    L=l,R=r;fr(i,l,r) if(v[a[i]])q[L++]=a[i]; else q[R--]=a[i];
    fr(i,l,r)a[i]=q[i];fz(l,L-1);fz(R+1,r);}
```

3.2 KM 最大权匹配

```cpp
LL KM(int n,LL w[N][N]){ // n 为点，w 为边权  不存在的边权开 -n*(|maxv|+1),inf 设为 3n*(|maxv|+1)
    static LL lx[N],ly[N],slk[N];
    static int lk[N],pre[N];static bool vy[N];
    LL ans=0;int x,py,d,p;
    fr(i,1,n)fr(j,1,n)lx[i]=max(lx[i],w[i][j]);
    fr(i,1,n){
        fr(j,1,n)slk[j]=inf,vy[j]=0;
        for(lk[py=0]=i;lk[py];py=p){
            vy[py]=1;d=inf;x=lk[py];
            fr(y,1,n)if(!vy[y]){
                if(lx[x]+ly[y]-w[x][y]<slk[y])
                    slk[y]=lx[x]+ly[y]-w[x][y],pre[y]=py;
                if(slk[y]<d)d=slk[y],p=y;
            }fr(y,0,n) if(vy[y])lx[lk[y]]-=d,ly[y]+=d; else slk[y]-=d;
        }for(;py;py=pre[py])lk[py]=lk[pre[py]];}
    fr(i,1,n)ans+=lx[i]+ly[i];
    return ans;}//lk[] 为与右部点连的左部点
```

3.3 Hopcroft-Karp $O(\sqrt{N}M)$ 最大匹配

```cpp
// 左侧N个点，右侧K个点 ,1-base，初始化将matx[],maty[]都置为0
int N, K, que[N], dx[N], dy[N], matx[N], maty[N];
int BFS() { int flag = 0, qt = 0, qh = 0;
    for(int i = 1; i <= K; ++ i) dy[i] = 0;
    for(int i = 1; i <= N; ++ i) { dx[i] = 0;
        if (! matx[i]) que[qt ++] = i; }
    while (qh < qt) { int u = que[qh ++];
        for(Edge *e = E[u]; e; e = e->n) {
            if (! dy[e->t]) { dy[e->t] = dx[u] + 1;
                if (! maty[e->t]) flag = true; else {
                    dx[maty[e->t]] = dx[u] + 2;
                    que[qt ++] = maty[e->t]; } } } }
    return flag; }
int DFS(int u) {
    for(Edge *e = E[u]; e; e = e->n) {
        if (dy[e->t] == dx[u] + 1) { dy[e->t] = 0;
            if (! maty[e->t] || DFS(maty[e->t])) {
                matx[u] = e->t; maty[e->t] = u; return true; }}}
    return false; }
void Hopcroft() {
while (BFS()) for(int i=1; i<=N; ++i) if (!matx[i]) DFS(i);}
```

3.4 Blossom 带花树

```cpp
struct blossom{//mat[i] 为 i 号点的匹配点
int n,m,h,t,W,tot,fir[N],la[M],ne[M],F[N];
int mat[N],pre[N],tp[N],q[N],vs[N];
void in(int x,int y){n=x;m=y;W=tot=0;CL(fir);}
void ins(x,y){}// 初始化 n 个点 m 条边的图
int fd(int x){return F[x]?F[x]=fd(F[x]):x;}
int lca(int u,int v){
    for(++W;;u=pre[mat[u]],swap(u,v))
        if(vs[u=fd(u)]==W)return u;else vs[u]=u?W:0;
}void aug(int u,int v){for (int w;u;v=pre[u=w]) w=mat[v],mat[mat[u]=v]=u;}
void blo(int u,int v,int f){
    for(int w;fd(u)^f;u=pre[v=w])
        pre[u]=v, F[u]?0:F[u]=f, F[w=mat[u]]?0:F[w]=f, tp[w]^1?0:tp[q[++t]=w]=-1;
}int bfs(int u){
    int x,E,i;CL(tp);CL(F);
    for(--tp[q[h=t=1]=u];h<=t;u=q[++h])
    for(i=fir[u];i;i=ne[i])if(!tp[E=la[i]]){
        if(!mat[E])return aug(E,u),1;
        pre[E]=u,++tp[E],--tp[q[++t]=mat[E]];
    }else if(tp[E]^1&&fd(u)^fd(E)) blo(u,E,x=lca(u,E)),blo(E,u,x);
    return 0;}
int solve(){// 返回答案
    int i,an=0;fr(i,1,n)mat[i]?0:an+=bfs(i);
    return an;}
}G;
```

3.5 Cactus 仙人掌

```cpp
// 求边仙人掌长度为len的简单路径条数ans[len]
const int N = 5005, MOD = 1e9 + 7; vector<int> G[N];
int f[N][N], dfn[N], tot, low[N], n, sz[N];
vector<int> getcycle(int x, int root) {
    vector<int> cyc; // 获得root->x为第一条边的环
    cyc.push_back(root); cyc.push_back(x);
    while (true) { int nex = -1;
        for (int u : G[cyc.back()]) {
            if (low[u] != dfn[root] || cyc[cyc.size() - 2] == u || u == root) continue;
            nex = u; break; }
        if (nex == -1) break; cyc.push_back(nex); }
    return cyc; }
int walk(int x, int root) { // 沿着root->x dp一圈 获得从root出发往子树走所有长度为len的简单路径条数f[root][len]
    vector<int> cycle = getcycle(x, root);
    int szm = 0; // 合并路径时只for到最长长度, 保证复杂度
    for (int i = 1; i < (int) cycle.size(); ++ i) {
        int u = cycle[i], l1 = i, l2 = (int) cycle.size() - i;
        for (int k = 0; k < sz[u]; ++ k) {
            if (k + l1 < n) (f[root][k + l1] += f[u][k]) %= MOD;
            if (k + l2 < n) (f[root][k + l2] += f[u][k]) %= MOD;
            szm = max(szm, k + l1); szm = max(szm, k + l2); } }
    return szm; }
int g[N], msz, ans[N], par[N];
void walk2(int x, int root) { // 统计最浅点在这个环上的路径
    vector<int> cycle = getcycle(x, root);
    for (int i = 1; i < (int) cycle.size(); ++ i) {
        int u = cycle[i]; // 出发点为u往下走
        int l1 = i, l2 = (int) cycle.size() - i;
        for (int k = 0; k < sz[u]; ++ k) {
            for (int l = 0; l < msz; ++ l) {
                //g[l]为root之前子树伸出去长l路径条数 双向*2
                if (l + k + l1 < n)
                    (ans[l+k+l1]+=2LL*g[l]*f[u][k]%MOD)%=MOD;
                if (l + k + l2 < n)
                    (ans[l+k+l2]+=2LL*g[l]*f[u][k]%MOD)%=MOD;}}}
    for (int i = 1; i < (int) cycle.size(); ++ i) {
        int u = cycle[i], l1 = i, l2 = (int) cycle.size() - i;
        for (int k = 0; k < sz[u]; ++ k) { // 更新g[l]
            if (k + l1 < n) (g[k + l1] += f[u][k]) %= MOD;
            if (k + l2 < n) (g[k + l2] += f[u][k]) %= MOD;
            msz = max(msz, k+l1+1); msz = max(msz, k+l2+1); } }
    for (int i = 1; i < (int) cycle.size(); ++ i) {
        for (int j = 1; j < (int) cycle.size(); ++ j) {
            if (i == j) continue;
            // 环上u和v往下走出两条路径的合并
            int u = cycle[i], v = cycle[j];
            int l1 = abs(i-j), l2 = (int)cycle.size()-l1;
            for (int k = 0; k < sz[u]; ++ k) {
                for (int l = 0; l < sz[v]; ++ l) {
                    if (l+k+l1<n) (ans[l+k+l1] += 1LL * f[v][l]*f[u][k] % MOD) %= MOD;
                    if (l+k+l2<n) (ans[l+k+l2] += 1LL * f[v][l]*f[u][k] % MOD) %= MOD;}}}}
    msz = min(msz, n); }
void dfs(int x, int p = 0) {
    low[x] = dfn[x] = ++ tot; par[x] = p;
    for (int u : G[x]) if (u != p) {
        if (dfn[u]) low[x] = min(low[x], dfn[u]);
            else { dfs(u, x); low[x] = min(low[x], low[u]); } }
    f[x][0] = 1; sz[x] = 1;
    for (int u : G[x]) if (u != p) {
        if (low[u] == dfn[x] && par[u] != x) {
            // 用环边f[x]
            sz[x]=max(sz[x], walk(u,x)+1); sz[x]=min(sz[x],n); }
        else if (low[u] > dfn[x]) {
            // 树边更新f[x]
            sz[x]=max(sz[x], sz[u] + 2); sz[x]=min(sz[x], n);
            for (int k = 0; k < sz[u]; ++ k)
                (f[x][k + 1] += f[u][k]) %= MOD; } }
    // 子树背包
    for (int i = 0; i < n; ++ i) g[i] = 0;
    g[0] = 1; ans[0] ++; msz = 1;
    for (int u : G[x]) { if (u == p) continue;
        if (low[u]==dfn[x] && par[u]!=x) walk2(u, x); // 环边
        else if (low[u] > dfn[x]) { // 树边
            for (int k = 0; k < sz[u]; ++ k) {
                for (int l = 0; l < msz; ++ l) {
                    if (k + l + 1 < n) (ans[k + l + 1] += 2LL * f[u][k] * g[l] % MOD) %= MOD; } }
            for (int k = 0; k < sz[u]; ++ k)
                (g[k + 1] += f[u][k]) %= MOD;
            msz = max(msz, sz[u] + 1); msz = min(msz, n); } } }
```

3.6 Tarjan 连通分量

```cpp
// 点双联通分量
void ins_new(int x,int y){}//点双树的边
struct Tarjan{//n为点数□V[]中存每条边
    int n,id,t,low[N],dfn[N],q[N];vector<int>V[N];void ins(x,y){}
    void in(int o){//初始化□标号从n+1开始
        id=0;scc=o;fr(i,1,o)V[i].clear(),dfn[i]=low[i]=0;
    }void tj(int x,int fa){
        dfn[x]=low[x]=++id;q[++t]=x;int i,y,o;
        for(i=0;i<V[x].size();i++)if(!dfn[y=V[x][i]]){
```

```cpp
      tj(y,x);low[x]=min(low[x],low[y]);
      if(low[y]>=dfn[x])for(ins_new(x,++scc),o=0;y!=o;)
        to[o=q[t--]]=scc,ins_new(scc,o);
    }else if(y!=fa)low[x]=min(low[x],dfn[y]);
  }
  void work(){for(int i=1;i<=n;i++)if(!dfn[i])tj(i,0);}
}G;
// 强连通分量
void tj(int x){
  int i,y,o=0,E=V[x].size();
  low[x]=dfn[x]=++id;is[x]=1;q[++t]=x;
  for(i=0;i<E;i++)if(!dfn[y=V[x][i]])tj(y),low[x]=min(low[x],low[y]);
    else if(is[y])low[x]=min(low[x],dfn[y]);
  if(dfn[x]==low[x])
    for(scc++;o!=x;o=q[t--],bl[o]=scc,is[o]=0);}
```

3.7 Dominator Tree 支配树

```cpp
struct Dominator_Tree{
  //n为点数，s为起点，e[]中记录每条边
  int n,s,cnt;int dfn[N],id[N],pa[N],semi[N],idom[N],p[N],mn[N];
  vector<int>e[N],dom[N],be[N];
  void ins(x,y){e[x].pb(y);}
  void dfs(int x){//先得到DFS树
    dfn[x]=++cnt;id[cnt]=x;
    for(auto i:e[x]){
      if(!dfn[i])dfs(i),pa[dfn[i]]=dfn[x];
      be[dfn[i]].push_back(dfn[x]);
    }}
  int get(int x){//带权并查集
    if(p[x]!=p[p[x]]){
      if(semi[mn[x]]>semi[get(p[x])]) mn[x]=get(p[x]);
      p[x]=p[p[x]];
    }return mn[x];
  }void LT(){//求出semi和idom得到支配树
    for(int i=cnt;i>1;i--){
      for(auto j:be[i]) semi[i]=min(semi[i],semi[get(j)]);
      dom[semi[i]].push_back(i); int x=p[i]=pa[i];
      for(auto j:dom[x])idom[j]=(semi[get(j)]<x?get(j):x);
      dom[x].clear();
    }fr(i,2,cnt){
      if(idom[i]!=semi[i])idom[i]=idom[idom[i]];
      dom[id[idom[i]]].push_back(id[i]);
    }
  }void build(){//建立支配树
    fr(i,1,n)dfn[i]=0,dom[i].clear(),be[i].clear(),p[i]=mn[i]=semi[i]=i;
    cnt=0;dfs(s);LT();
  }
}G;
```

3.8 朱刘算法 最小树形图 (含lazy tag可并堆)

```cpp
using Val = long long;
#define nil mem
struct Node { Node *l,*r; int dist;int x,y;Val val,laz; }
mem[M] = {{nil, nil, -1}}; int sz = 0;
#define NEW(arg...) (new(mem + ++ sz)Node{nil,nil,0,arg})
void add(Node *x, Val o) {if(x!=nil){x->val+=o, x->laz+=o;}}
void down(Node *x){add(x->l,x->laz);add(x->r,x->laz);x->laz=0;}
Node *merge(Node *x, Node *y) {
  if (x == nil) return y; if (y == nil) return x;
  if (y->val < x->val) swap(x, y); //smalltop heap
  down(x); x->r = merge(x->r, y);
  if (x->l->dist < x->r->dist) swap(x->l, x->r);
  x->dist = x->r->dist + 1; return x; }
Node *pop(Node *x){down(x); return merge(x->l, x->r);}
struct DSU { int f[N]; void clear(int n) {
    for (int i=0; i<=n; ++i) f[i]=i; }
  int fd(int x) { if (f[x]==x) return x;
    return f[x]=fd(f[x]); }
  int& operator[](int x) {return f[fd(x)];}};
DSU W, S; Node *H[N], *pe[N];
vector<pair<int, int>> G[N]; int dist[N], pa[N];
// addedge(x, y, w) : NEW(x, y, w, 0)
Val chuliu(int s, int n) { // O(ElogE)
  for (int i = 1; i <= n; ++ i) G[i].clear();
  Val re=0; W.clear(n); S.clear(n); int rid=0;
  fill(H, H + n + 1, (Node*) nil);
  for (auto i = mem + 1; i <= mem + sz; ++ i)
    H[i->y] = merge(i, H[i->y]);
  for (int i = 1; i <= n; ++ i) if (i != s)
    for (;;) {
      auto in = H[S[i]]; H[S[i]] = pop(H[S[i]]);
      if (in == nil) return INF; // no solution
      if (S[in -> x] == S[i]) continue;
      re += in->val; pe[S[i]] = in;
      // if (in->x == s) true root = in->y
      add(H[S[i]], -in->val);
      if (W[in->x]!=W[i]) {W[in->x]=W[i];break;}
      G[in -> x].push_back({in->y,++rid});
      for (int j=S[in->x]; j!=S[i]; j=S[pe[j]->x]) {
        G[pe[j]->x].push_back({pe[j]->y, rid});
        H[j] = merge(H[S[i]], H[j]); S[i]=S[j]; }}
  ++ rid; for (int i=1; i<=n; ++ i) if(i!=s && S[i]==i)
    G[pe[i]->x].push_back({pe[i]->y, rid}); return re;}
void makeSol(int s, int n) {
  fill(dist, dist + n + 1, n + 1); pa[s] = 0;
  for (multiset<pair<int, int>> h = {{0,s}}; !h.empty();){
    int x=h.begin()->second;
    h.erase(h.begin()); dist[x]=0;
    for (auto i : G[x]) if (i.second < dist[i.first]) {
      h.erase({dist[i.first], i.first});
      h.insert({dist[i.first] = i.second, i.first});
      pa[i.first] = x; }}}
```

3.9 最大团

```cpp
// DN超级快最大团，建议 n <= 150
typedef bool BB[N]; struct Maxclique {
const BB *e; int pk, level; const float Tlimit;
struct Vertex { int i, d; Vertex(int i) : i(i), d(0) {}};
typedef vector<Vertex> Vertices; Vertices V;
typedef vector<int> ColorClass; ColorClass QMAX, Q;
vector<ColorClass> C;
static bool desc_degree(const Vertex &vi,const Vertex &vj)
{ return vi.d > vj.d; }
void init_colors(Vertices &v) {
  const int max_degree = v[0].d;
  for (int i = 0; i < (int)v.size(); i++)
    v[i].d = min(i, max_degree) + 1; }
void set_degrees(Vertices &v) {
  for (int i = 0, j; i < (int)v.size(); i++)
    for (v[i].d = j = 0; j < (int)v.size(); j++)
      v[i].d += e[v[i].i][v[j].i]; }
struct StepCount{ int i1, i2; StepCount(): i1(0),i2(0){}};
vector<StepCount> S;
bool cut1(const int pi, const ColorClass &A) {
  for (int i = 0; i < (int)A.size(); i++)
    if (e[pi][A[i]]) return true; return false; }
void cut2(const Vertices &A, Vertices & B) {
  for (int i = 0; i < (int)A.size() - 1; i++)
    if (e[A.back().i][A[i].i]) B.push_back(A[i].i); }
void color_sort(Vertices & R) { int j=0, maxno=1;
  int min_k=max((int)QMAX.size()-(int)Q.size()+1,1);
  C[1].clear(), C[2].clear();
  for (int i = 0; i < (int)R.size(); i++) {
    int pi = R[i].i, k = 1; while (cut1(pi, C[k])) k++;
    if (k > maxno) maxno = k, C[maxno + 1].clear();
    C[k].push_back(pi); if (k < min_k) R[j++].i = pi; }
  if (j > 0) R[j - 1].d = 0;
  for (int k = min_k; k <= maxno; k++)
    for (int i = 0; i < (int)C[k].size(); i++)
      R[j].i = C[k][i], R[j++].d = k; }
void expand_dyn(Vertices &R) {
  S[level].i1 = S[level].i1 + S[level-1].i1 - S[level].i2;
  S[level].i2 = S[level - 1].i1;
  while ((int)R.size()) {
    if ((int)Q.size() + R.back().d > (int)QMAX.size()) {
      Q.push_back(R.back().i); Vertices Rp; cut2(R, Rp);
      if ((int)Rp.size()) {
        if((float)S[level].i1/++pk<Tlimit)degree_sort(Rp);
        color_sort(Rp); S[level].i1++, level++;
        expand_dyn(Rp); level--;
      } else if ((int)Q.size() > (int)QMAX.size()) QMAX=Q;
      Q.pop_back(); } else return; R.pop_back(); }}
void mcqdyn(int *maxclique, int &sz) {
  set_degrees(V); sort(V.begin(), V.end(), desc_degree);
  init_colors(V);
  for (int i=0; i<(int)V.size()+1; i++) S[i].i1=S[i].i2=0;
  expand_dyn(V); sz = (int)QMAX.size();
 for(int i=0;i<(int)QMAX.size();i++)maxclique[i]=QMAX[i];}
void degree_sort(Vertices & R) {
  set_degrees(R); sort(R.begin(), R.end(), desc_degree); }
Maxclique(const BB *conn,const int sz,const float tt=.025)
  : pk(0), level(1), Tlimit(tt){
  for(int i = 0; i < sz; i++) V.push_back(Vertex(i));
  e = conn, C.resize(sz + 1), S.resize(sz + 1); }};
BB e[N]; int ans, sol[N]; for (...) e[x][y]=e[y][x]=true;
Maxclique mc(e, n); mc.mcqdyn(sol, ans); // 全部0下标
for (int i = 0; i < ans; ++i) cout << sol[i] << endl;
```

3.10 极大团计数

```
// 0下标，需删除自环(即确保$E_{ii} = false$，补图要特别注意)
// 极大团计数，最坏情况O(3^(n/3))
ll ans; ull E[64]; #define bit(i) (1ULL << (i))
void dfs(ull P, ull X, ull R) { // 不需要方案时可去掉R相关语句
  if (!P && !X) { ++ans; sol.pb(R); return; }
  ull Q = P & ~E[__builtin_ctzll(P | X)];
  for (int i; i = __builtin_ctzll(Q), Q; Q &= ~bit(i)) {
    dfs(P & E[i], X & E[i], R | bit(i));
    P &= ~bit(i), X |= bit(i); }}
ans = 0; dfs(n == 64 ? ~0ULL : bit(n) - 1, 0, 0);
```

3.11 Dinic 最大流

```
template <int MAXN = 100000, int MAXM = 100000>
struct flow_edge_list { int size;
int begin[MAXN], dest[MAXM], next[MAXM],flow[MAXM],inv[MAXM];
void clear(int n) { size=0; std::fill(begin, begin+n, -1); }
flow_edge_list (int n = MAXN) { clear (n); }
void add_edge (int u, int v, int f) {
  dest[size] = v; next[size] = begin[u]; flow[size] = f;
  inv[size] = size + 1; begin[u] = size++;
  dest[size] = u; next[size] = begin[v]; flow[size] = 0;
  inv[size] = size - 1; begin[v] = size++; } };
template <int MAXN = 1000, int MAXM = 100000>
struct dinic { int n, s, t; int d[MAXN], w[MAXN], q[MAXN];
int bfs (flow_edge_list <MAXN, MAXM> &e) {
  for (int i = 0; i < n; ++i) d[i] = -1; int l, r;
  q[l = r = 0] = s, d[s] = 0;
  for (; l <= r; l ++)
    for (int k = e.begin[q[l]]; k > -1; k = e.next[k])
      if (d[e.dest[k]] == -1 && e.flow[k] > 0)
        d[e.dest[k]] = d[q[l]]+1, q[++r] = e.dest[k];
  return d[t] > -1 ? 1 : 0; }
int dfs (flow_edge_list <MAXN, MAXM> &e, int u, int ext) {
  if (u == t) return ext; int k = w[u], ret = 0;
  for (; k>-1; k=e.next[k], w[u] = k) { if (ext==0) break;
    if (d[e.dest[k]] == d[u] + 1 && e.flow[k] > 0) {
      int flow=dfs(e, e.dest[k], std::min(e.flow[k],ext));
      if (flow > 0) {
        e.flow[k] -= flow, e.flow[e.inv[k]] += flow;
        ret += flow, ext -= flow; } } }
  if (k == -1) d[u] = -1; return ret; }
int solve(flow_edge_list<MAXN, MAXM> &e,int n,int s,int t) {
  int ans = 0; dinic::n = n; dinic::s = s; dinic::t = t;
  while (bfs (e)) {
    for (int i = 0; i < n; ++i) w[i] = e.begin[i];
    ans += dfs (e, s, INF); } return ans; } };
```

3.12 Dijkstra 费用流

```
const LL INF = 1e18; struct Edge { LL f, c; int to, r; };
vector<Edge> G[N]; LL d[N]; bool fst = true;
int S, T, prv[N], prp[N], cur[N], vst[N];
bool SPFA(int S) {
  if(fst) { fst = 0;
    // 此处为第一次求最短路，可 Dij 就和下面一样，不可就 SPFA 或根据图性质 DP
    return d[T] != INF; }
  fill(d + 1, d + 1 + T, INF); // 此处为 Dij
  priority_queue<pair<LL,int>> pq; pq.push({0,S}); d[S]=0;
  while(1) {
    while(!pq.empty() && -pq.top().first != d[pq.top().second]) pq.pop();
    if(pq.empty()) break;
    int v(pq.top().second); pq.pop(); int cnt(0);
    for (Edge e : G[v]) {
      if (e.f && d[e.to] > d[v] + e.c) {
        d[e.to] = d[v] + e.c; prv[e.to] = v;
        prp[e.to]=cnt; pq.push({-d[e.to], e.to}); }
      cnt++; } }
  return d[T] != INF; }
LL aug(int v, LL flow) {
  if(v == T) return flow; vst[v] = 1; LL flow1(flow);
  for(int & i(cur[v]); i < (int)G[v].size(); i++) {
    Edge & e = G[v][i];
    if(e.f && d[v] + e.c == d[e.to] && !vst[e.to]) {
      LL flow1(aug(e.to, min(flow, e.f)));
      flow-=flow1; e.f-=flow1; G[e.to][e.r].f += flow1; }
    if(flow == 0) { vst[v] = 0; return flow1 - flow; } }
  return flow1 - flow; }
LL mcmf() { LL ans = 0, sT = 0;
  while (SPFA(S)) { sT += d[T]; // 多路增广
    for(int i(1); i <= T; i++) cur[i] = 0, vst[i] = 0;
    ans += sT * aug(S, INF);
    for(int i(1); i <= T; i++)
      for(auto & e : G[i]) e.c += d[i] - d[e.to];}
  return ans; }
void add(int u, int v, int f, int c) {
  G[u].push_back({f, c, v, (int) G[v].size()});
  G[v].push_back({0, -c, u, (int) G[u].size() - 1}); }
int main() { // 初始化 S, T, T 编号最大, 1base
  // add(x, y, cap, cost)
  LL ans = mcmf(); }
```

3.13 完美消除序列 弦图判定与最小染色

```
//id[i]为点i的标号，seq[i]为标号为i的点，G[]存图
int q[N],label[N],id[N],vis[N],seq[N],c[N]; vector<int>G[N];
struct P{int lab,u;bool operator<(const P&a) const {return lab<a.lab;}};
void mcs(){//MCS算法求标号序列,优先队列做到O(mlgn)
  int i,j,u,v;CL(id);CL(label);
  CL(seq);priority_queue<P>Q;
  fr(i,1,n)Q.push(P{0,i}); //label_i表示第i个点与多少个已标号的点相邻
  dr(i,n,1){
    for(;id[Q.top().u];)Q.pop(); //每次选label_i最大的未标号的点标号
    u=Q.top().u;Q.pop();id[u]=i;
    for(j=0;j<G[u].size();j++)if(v=G[u][j],!id[v])label[v]++,Q.push(P{label[v],v});
  }fr(i,1,n)seq[id[i]]=i;
}bool ok(){//O(m)判断是否是弦图
  int i,j,t,u,v,w;CL(vis);
  dr(i,n,1){
    u=seq[i];t=0;//标号从小到大找点
    for(j=0;j<G[u].size();j++)
      if(v=G[u][j],id[v]>id[u])q[++t]=v;
```

```
        if(!t)continue;w=q[1];//找标号大于它的点中最小的
        fr(j,1,t)if(id[q[j]]<id[w])w=q[j];
        for(j=0;j<G[w].size();j++)vis[G[w][j]]=i;
        fr(j,1,t)if(q[j]!=w)if(vis[q[j]]!=i)return 0;
    }return 1;
int setcolor(){//弦图最小染色 团数=染色数
    int an=0,i,j,u,v;CL(vis);CL(c);
    for(i=n;i;i--){
        u=seq[i];
        for(j=0;j<G[u].size();j++)vis[c[G[u][j]]]=i;
        for(j=1;vis[j]==i;j++);//找最小的没出现的颜色
        c[u]=j;an=max(an,j);
    }return an;
}mcs();puts(ok()?"YES":"NO");printf("%d\n",setcolor());
```

3.14 欧拉回路

```
int x,y,t,tot,fir[N],d1[N],d2[N],q[M],la[M],ne[M],va[M];bool v[M];
void ins(int x,int y,int z){/*ADDEDGE*/d1[x]++;d2[y]++;}
void dfs(int x){
    for(int i=fir[x];i;i=fir[x]){
        for(;i&&v[abs(va[i])];i=ne[i]);fir[x]=i; // 将已经走过的边删去
        if(i)v[abs(va[i])]=1,dfs(la[i]),q[++t]=va[i];}// 走第一个未走过的边
}void Eular(p,n,m){
//p=1 是无向图，p=2 是有向图  n 是点数，m 是边数
    fr(i,1,m)if(rd(x,y),ins(x,y,i),p==1)ins(y,x,-i);
    fr(i,1,n)if(p==1&&d1[i]%2||p==2&&d1[i]^d2[i])
        return puts("NO"),0;// 不合法情况
    dfs(la[1]); // 从一条边开始找
    if(t<m)return puts("NO"),0; // 没有欧拉回路
    for(puts("YES");t;printf("%d ",q[t--]));} // 输出方案，正数是正向边，负数是反向边
```

3.15 K短路

```
int F[N],FF[N];namespace Left_Tree{//可持久化左偏树
    struct P{int l,r,h,v,x,y;}Tr[N*40];int RT[N],num;
    //l和r是左右儿子,h是高度,v是数值,x和y是在图中的两点
    int New(P o){Tr[++num]=o;return num;}
    void start(){num=0;Tr[0].l=Tr[0].r=Tr[0].h=0;Tr[0].v=inf;}
    int mg(int x,int y){
        if(!x)return y;
        if(Tr[x].v>Tr[y].v)swap(x,y);
        int o=New(Tr[x]);Tr[o].r=mg(Tr[o].r,y);
        if(Tr[Tr[o].l].h<Tr[Tr[o].r].h)swap(Tr[o].l,Tr[o].r);
        Tr[o].h=Tr[Tr[o].r].h+1;return o;}
    void add(int&k,int v,int x,int y){
        int o=New(Tr[0]);
        Tr[o].v=v;Tr[o].x=x;Tr[o].y=y;
        k=mg(k,o);    }}
using namespace Left_Tree;
struct SPFA{//SPFA，这里要记录路径
    void in(){tot=0;CL(fir);}void ins(x,y,z){}
    void work(int S,int n){//F[]求最短路从哪个点来□FF[]记最短路从哪条边来
}}A;
struct Kshort{
int tot,n,m,S,T,k,fir[N],va[M],la[M],ne[M];bool v[N];
struct P{
    int x,y,z;P(){}P(int x,int y,int z):x(x),y(y),z(z){}
    bool operator<(P a)const{return a.z<z;}};
priority_queue<P>Q;void in(){tot=0;CL(fir);}
void ins(x,y,z){}
void init(){//将图读入
    int i,x,y,z;in();A.in();start();rd(n,m)
    fr(i,1,m)rd(x,y,z),A.ins(y,x,z),ins(x,y,z);
    rd(S,T,k);if(S==T)k++;//注意起点终点相同的情况
    A.work(T,n);}//A是反向边
void dfs(int x){
    if(v[x])return;v[x]=1;if(F[x])RT[x]=RT[F[x]];
    for(int i=fir[x],y;i;i=ne[i])if(y=la[i],A.d[y]!=inf&&FF[x]!=i)
        add(RT[x],A.d[y]-A.d[x]+va[i],x,y);
    for(int i=A.fir[x];i;i=A.ne[i])if(F[A.la[i]]==x)dfs(A.la[i]);}
int work(){//返回答案，没有返回-1
    int i,x;dfs(T);
    if(!--k)return A.d[S]==inf?-1:A.d[S];
    P u,w;if(RT[S])Q.push(P(S,RT[S],A.d[S]+Tr[RT[S]].v));
    for(;k--;){
        if(Q.empty())return -1;u=Q.top();Q.pop();
        if(x=mg(Tr[u.y].l,Tr[u.y].r))
            Q.push(P(u.x,x,Tr[x].v-Tr[u.y].v+u.z));
        if(RT[x=Tr[u.y].y])Q.push(P(x,RT[x],u.z+Tr[RT[x]].v));}
    return u.z;}}G;
```

3.16 平面图转对偶图

```
int to[N],v[N],q[N];LL Area;
struct P{int x,y;double o;}p[N],e[N];
struct cmp{bool operator()(int a,int b){return e[a].o<e[b].o;}};
LL operator*(P a,P b){return 1ll*a.x*b.y-1ll*a.y*b.x;}
set<int,cmp>S[N];set<int,cmp>::iterator it;
void ADD(int x,int y){}//再新图中加边 不能有重复点 可以有多余边
void Planet(n,m){ //n为点数，m为边数，p[i].x/y为n个点坐标
    int j,x,y,cnt=0;
    fr(i,0,m-1){
        rd(x);rd(y); //第i条边连接第x个点和第y个点
        e[i<<1]=P{x,y, atan2(p[y].x-p[x].x,p[y].y-p[x].y)}; //加双向边，支持双向边权
        e[i<<1|1]=P{y,x, atan2(p[x].x-p[y].x,p[x].y-p[y].y)};
    }m*=2;
    fr(i,0,m-1)S[e[i].x].insert(i);
    fr(i,0,m-1)if(!v[i]){
        for(q[t=1]=j=i;;q[++t]=j=*it){ //按极角排序选最少转动角度找出下一条边
            it=S[e[j].y].upper_bound(j^1);
            if(it==S[e[j].y].end())it=S[e[j].y].begin();
            if(*it==i)break;
        }Area=0;fr(j,1,t) Area+=p[e[q[j]].x]*p[e[q[j]].y],v[q[j]]=1;
        if(Area>0){cnt++;fr(j,1,t)to[q[j]]=cnt;} //面积大于0是有限区域
    }fr(i,0,m-1)ADD(to[i],to[i^1]);} //若这条边有边权，在e[i].z中记录 // 如果to[i]为0不一定是外平面，要判在一个最小的多边形内
```

3.17 树同构

```
// O(1) 求逆 时间复杂度 O(n) MOD 需要是质数
#define fors(i) for (auto i : e[x]) if (i != p)
int ra[N]; void prepare() {
    for (int i = 0; i < N; ++ i) ra[i] = rand() % MOD;}
struct Sub {
    vector<int> s; int d1, d2, H1, H2;
    Sub() {d1 = d2 = 0; s.clear();}
    void add(int d, int v) { s.push_back(v);
        if (d>d1) d2=d1, d1=d; else if (d>d2) d2=d; }
    int hash() { H1 = H2 = 1; for (int i : s) {
            H1 = (ll) H1 * (ra[d1]+i) % MOD;
            H2 = (ll) H2 * (ra[d2]+i) % MOD; } return H1;}
    pii del(int d, int v) { if (d==d1)
        return {d2+1, (ll)H2*reverse(ra[d2]+v) % MOD};
        return {d1+1, (ll)H1*reverse(ra[d1]+v) % MOD};}};
pii U[N]; int A[N]; Sub tree[N]; //A[x]为以x为根的哈希值
void dfsD(int x, int p) { tree[x] = Sub();
    fors(i) { dfsD(i, x);
        tree[x].add(tree[i].d1 + 1, tree[i].H1); }
    tree[x].hash(); }
void dfsU(int x, int p) {
    if (p) tree[x].add(U[x].first, U[x].second);
    A[x] = tree[x].hash();
    fors(i){U[i]=tree[x].del(tree[i].d1+1,tree[i].H1);
        dfsU(i, x); } }
```

3.18 虚树

```cpp
rd(m);v[1]=a[1]=1,t=++m; //选定m个关键点, 1号点直接加入虚树
fr(i,2,m)rd(x),v[a[i]]=V[a[i]]=1; //v[]表示是选的点或LCA后的点, V[]表示选的点
for(sort(a+1,a+m+1,cmp),i=1;i<m;i++) //新加必要关键点
    if(!v[x=lca(a[i],a[i+1])])v[a[++t]=x]=1;
for(m=t,sort(a+1,a+m+1,cmp),ed=0,q[t=1]=1,i=2; i<=m; ins(q[t],a[i]),q[++t]=a[i++])
    for(;st[a[i]]<st[q[t]]||en[a[i]]>en[q[t]];t--); //再排一遍序, 单调栈建虚数
fr(i,1,m)v[a[i]]=V[a[i]]=fir[a[i]]=0; //求解答案后还原
```

3.19 动态最小生成树

```cpp
int n, m, q; // O((m+q) log q)
struct EdgeInfo {
int u, v, w, l, r; EdgeInfo(int u,int v,int w,int l,int r) :
    u(u),v(v),w(w),l(l),r(r){} EdgeInfo() {} };
long long ans[N]; int find(int f[], int u) {
    return f[u] == u ? u : f[u] = find(f, f[u]); }
bool join(int f[], int u, int v){ u=find(f,u), v=find(f, v);
    if (u == v) return false; return f[u] = v, true; }
void dfs(int l, int r, int n, const vector<EdgeInfo> &list, long long base) {
    if (list.empty())
        { for (int i=l; i<=r; i++) ans[l]=base; return ; }
    static vector<EdgeInfo> all,part;all.clear();part.clear();
    for (auto &e : list) if (e.l <= l && r <= e.r) {
            all.push_back(e);
        } else if (l <= e.r && e.l <= r) {part.push_back(e);}
    static int f[N], color[N], id[N]; // Contraction
    for (int i = 0; i < n; i++) f[i] = color[i] = i;
    for (auto &e : part) join(f, e.u, e.v);
    for (auto &e : all) if (join(f, e.u, e.v)) {
        join(color, e.u, e.v); base += e.w; }
    if (l == r) { ans[l] = base; return ; }
    for (int i = 0; i < n; i++) id[i] = -1;
    int tot = 0; for (int u = 0; u < n; u++) {
        int v = find(color, u); if (id[v] == -1) id[v] = tot++;
        id[u] = id[v]; } int m = 0; // Reduction
    for (int i = 0; i < tot; i++) f[i] = i;
    for (auto &e : part) { e.u = id[find(color, e.u)];
        e.v = id[find(color, e.v)]; }
    for (auto &e : all) {
        e.u = id[find(color, e.u)], e.v = id[find(color, e.v)];
        if (e.u == e.v) continue; assert(e.u<tot && e.v<tot);
        if (join(f, e.u, e.v)) all[m++] = e; }
    all.resize(m); vector<EdgeInfo> new_list;
    for (int i=0, j=0; i < part.size() || j < all.size(); ) {
        if(i<part.size()&&(j==all.size()||all[j].w>part[i].w)){
            new_list.push_back(part[i++]);
        } else { new_list.push_back(all[j++]); } }
    int mid = (l + r) / 2; dfs(l, mid, tot, new_list, base);
    dfs(mid + 1, r, tot, new_list, base); }
int main() { scanf("%d %d %d", &n, &m, &q);
    vector<pair<int, int> > memo; static int u[N], v[N], w[N];
    for (int i = 0; i < m; i++) {
        scanf("%d %d %d", &u[i], &v[i], &w[i]);
        --u[i], --v[i]; memo.push_back({0, w[i]}); }
    vector<EdgeInfo> info; // 把第 k 条边权值改为 d
    for (int i = 0; i < q; i++) {
        int k, d; scanf("%d %d", &k, &d); --k;
        if (memo[k].first < i) {
            info.push_back({u[k], v[k], memo[k].second, memo[k].first, i - 1});
        } memo[k] = {i, d}; }
    for (int i = 0; i < m; i++) {
        info.push_back({u[i], v[i], memo[i].second, memo[i].first, q - 1}); }
    sort(info.begin(), info.end(), [&](const EdgeInfo &a, const EdgeInfo &b) { return a.w < b.w; });
    dfs(0, q - 1, n, info, 0);
    for (int i = 0; i < q; i++) printf("%lld\n", ans[i]); }
```

3.20 图论知识

3.20.1 LCT常见应用

动态维护边双 可以通过LCT来解决一类动态边双连通分量问题．即静态的询问可以用边双连通分量来解决，而树有加边等操作的问题．

把一个边双连通分量缩到LCT的一个点中，然后在LCT上求出答案．缩点的方法为加边时判断两点的连通性，如果已经联通则把两点在目前LCT路径上的点都缩成一个点．

动态维护基环森林 通过LCT可以动态维护基环森林，即每个点有且仅有一个出度的图．有修改操作，即改变某个点的出边．对于每颗基环森林记录一个点为根，并把环上额外的一条边单独记出，剩下的边用LCT维护．一般使用有向LCT 维护．

修改时分以下几种情况讨论：1，修改的点是根，如果改的父亲在同一个联通块中，直接改额外边，否则删去额外边，在LCT上加边．2，修改的点不是根，那么把这个点和其父亲的联系切除．如果该点和根在一个环上，那么把多的那条边加到LCT上．最后如果改的那个父亲和修改的点在一个联通块中，记录额外边，否则LCT上加边．具体见例题．

3.20.2 LCT解决子树询问

通过记录轻边信息可以快速地维护出整颗LCT的一些值．如子树和，子树最大值等．在Access时要进行虚实边切换，这时减去实边的贡献，并加上新加虚边的贡献即可．有时需要套用数据结构，如Set来维护最值等问题．

模板：1，$x \to y$链$+z$；2，$x \to y$链变为z；3，在以x为根的树对y子树的点权求和；4，$x \to y$ 链取max；5，$x \to y$链求和；6，连接$x\ y$；7，断开$x\ y$．保证操作合法．

V单点值；sz平衡树的SZ；mv链上最大；S链上和；sm 区间相同标记；lz区间加标记；B 虚边之和；ST子树信息和；SM子树$+$链上信息和． 更新时，$S[x] = S[c[x][0]] + S[c[x][1]] + V[x]$，$ST[x] = B[x] + ST[c[x][0]] + ST[c[x][1]]$，$SM[x] = S[x] + ST[x]$．

3.20.3 差分约束

若要使得所有量两两的值最接近，则将如果将源点到各点的距离初始化为0．若要使得某一变量与其余变量的差最大，则将源点到各点的距离初始化为∞，其中之一为0．若求最小方案则跑最长路，否则跑最短路．

3.20.4 斯坦纳树

在一个无向带权图$G = (V, E)$中，将指定的k个点连通的一颗树称为斯坦纳树，边权总和最小的斯坦纳树称为最小斯坦纳树．

我们可以用DP+SPFA的方法求解斯坦纳树． 用$F_{i,state}$表示以i为根，指定集合中的点的联通状态为$state$的生成树的最小总权值，有两种转移方程．

第一种，通过两个子集合并进行转移，即$F_{i,state} = min(F_{i,subset1} + F_{i,subset2})$，这一部分使用DP完成．

第二种，在当前的联通状态下，对该联通状态进行松弛操作，即$F_{i,state} = min(F_{i,state}, F_{j,state} + w(i,j))$，这一部分使用SPFA完成．

时间复杂度$O(V * 3^k + cE * 2^k)$，c为SPFA复杂度中的常数．

3.20.5 李超线段树

李超线段树可以动态在添加若干条线段或直线$(a_i, b_i) \to (a_j, b_j)$，每次求$[l, r]$上最上面的那条线段的值．思想是让线段树中一个节点只对应一条直线，如果在这个区间加入一条直线，那么分类讨论．如果新加的这条直线在左右两端都比原来的更优，则替换原来的直线，将原来的直线扔掉．如果左右两端都比原来的劣，将这条直线扔掉．如果一段比原来的优，一段比原来的劣，那么判断一下两条线的交点，判断哪条直线可以完全覆盖一段一半的区间，把它保留，另一条直线下传到另一半区间．时间复杂度$O(nlgn)$．

3.20.6 吉如一线段树

吉如一线段树能解决一类区间和某个数取最大或最小，区间求和的问题．以区间取最小值为例，在线段树的每一个节点额外维护区间中的最大值ma，严格次大值se以及最大值个数t．现在假设我们要让区间$[L, R]$对x取min，先在线段树中定位若干个节点，对于每个节点分三种情况讨论：1，当$ma \le x$时，显然这一次修改不会对这个节点产生影响，直接退出；2，当$se < x < ma$时，显然这一次修改只会影响到所有最大值，所以把num加上$t * (x - ma)$，把ma更新为x，打上标记退出；3，当$se \ge x$时，无法直接更新着一个节点的信息，对当前节点的左儿子和右儿子递归处理．单次操作均摊复杂度$O(lg^2 n)$．

3.20.7 二分图最大匹配

最大独立集 最小覆盖点集 最小路径覆盖 最大独立集指求一个二分图中最大的一个点集，使得该点集内的点互不相连．最大独立集=总顶点数-最大匹配数．最小覆盖点集指用最少的点，使所有的边至少和一个点有关联．最小覆盖点集=最大匹配数．最小路径覆盖指一个DAG图G中用最少的路径使得所有点都被经过．最小路径覆盖=总点数-最大匹配数（拆点构图）．最大独立集S与最小覆盖集T互补．构造方法：1.做最大匹配，没有匹配的空闲点$u \in S$ 2.如果$u \in S$那么u的邻点必然属于T 3.如果一对匹配的点中有一个属于T那么另外一个属于S 4.还不能确定的，把左子图的放入S，右子图放入T．

二分图最大匹配关键点 关键点指的是一定在最大匹配中的点．由于二分图左右两侧是对称的，我们只考虑找左侧的关键点．先求任意一个最大匹配，然后给二分图定向：匹配边从右到左，非匹配边从左到右，从左侧每个不在最大匹配中的点出发DFS，给到达的那些点打上标记，最终左侧每个没有标记的匹配点即将爱为关键点．时间复杂度$O(n + m)$．

Hall定理 二分图$G = (X, Y, E)$有完备匹配的充要条件是：对于X的任意一个子集S都满足$|S|leq|A(S)|$，$A(S)$是Y的子集，是S的邻集．邻集的定义是与S有边的点集．

3.20.8 稳定婚姻问题

有n位男士和n位女士，每个人都对每个异性有一个不同的喜欢程度，现在使得每人恰好有一个异性配偶．如果男士u和女士v不是配偶但喜欢对方的程度都大于喜欢各自当前的配偶，则称他们为一个不稳定对．稳定婚姻问题是为了找出一个不含不稳定对的方案．

稳定婚姻问题的经典算法为求婚-拒绝算法，即男士按自己喜欢程度从高到底依次向每位女士求婚，直到有一个接受他．女士遇到比当前配偶更差的男士时拒绝他，遇到更喜欢的男士时就接受他，并抛弃以前的配偶．被抛弃的男士继续按照列表向剩下的女士依次求婚，直到所有人都有配偶．如果算法一定能得到一个匹配，而且这个匹配一定是稳定的．时间复杂度$O(n^2)$．

3.20.9 最大流和最小割

常见建模方法 拆点；黑白染色；流量正无穷表示冲突；缩点；数据结构优化建图；最小割 每个变元拉一条S到T的链，割在哪里表示取值，相互连边表示依赖关系；先把收益拿下，在考虑冲突与代价的影响．

判断一条边是否可能/一定在最小割中 令G'为残量网络G在强联通分量缩点之后的图．那么一定在最小割中的边(u, v)：(u, v)满流，且在G'中$u = S, v = T$；可能在最小割方案中的边(u, v)：(u, v)满流，或(u, v)满流，且在G'中$u \neq v$．

混合图欧拉回路 把无向边随便定向，计算每个点的入度和出度，如果有某个点出入度之差$deg_i = in_i - out_i$为奇数，肯定不存在欧拉回路．对于$deg_i > 0$的点，连接边$(i, T, deg_i/2)$;对于$deg_i < 0$的点，连接边$(S, i, -deg_i/2)$．最后检查是否满流即可．

3.20.10 一些网络流建图

无源汇有上下界可行流

每条边(u, v) 有一个上界容量$C_{u,v}$和下界容量$B_{u,v}$，我们让下界变为0,上界变为$C_{u,v}-B_{u,v}$，但这样做流量不守恒．建立超级源点SS和超级汇点TT，用du_i来记录每个节点的流量情况，$du_i = \sum B_{j,i} - \sum B_{i,j}$，添加一些附加弧．当$du_i > 0$时，连边$(SS, i, du_i)$;当$du_i < 0$时，连边$(i, TT, -du_i)$．最后对$(SS, TT)$求一次最大流即可，当所有附加边全部满流时(即$maxflow == \quad du_i > 0$)时有可行解．

有源汇有上下界最大可行流

建立超级源点SS和超级汇点TT，首先判断是否存在可行流，用无源汇有上下界可行流的方法判断．增设一条从T到S没有下界容量为无穷的边，那么原图就变成了一个无源汇有上下界可行流问题．同样地建图后，对(SS, TT)进行一次最大流，判断是否有可行解．

如果有可行解，删除超级源点SS和超级汇点TT，并删去T 到S的这条边，再对(S, T)进行一次最大流，此时得到的$maxflow$即为有源汇有上下界最大可行流．

有源汇有上下界最小可行流

建立超级源点SS和超级汇点TT，和无源汇有上下界可行流一样新增一些边，然后从SS到TT跑最大流．接着加上边(T, S, ∞)，再从SS到TT跑一遍最大流．

如果所有新增边都是满的，则存在可行流，此时T到S这条边的流量即为最小可行流．

有上下界费用流

如果求无源汇有上下界最小费用可行流或有源汇有上下界最小费用最大可行流，用1.6.3.1/1.6.3.2 的构图方法，给边加上费用即可．

求有源汇有上下界最小费用最小可行流，要先用1.6.3.3的方法建图，先求出一个保证必要边满流情况下的最小费用．如果费用全部非负，那么这时的费用就是答案．如果费用有负数，那么流多了可能更好，继续做从S到T的流量任意的最小费用流，加上原来的费用就是答案．

费用流消负环 新建超级源SS汇TT，对于所有流量非空的负权边e，先流满(ans+=e.f*e.c，e.rev.f+=e.f，e.f=0)，再连边SS→e.to，e.from→TT，流量均为e.f(>0)，费用均为0．再连边T→S流量∞费用0．此时没有负环了．做一遍SS到TT的最小费用最大流，将费用累加ans，拆掉T→S的那条边 (此边的流量为残量网络中S→T的流量)．此时负环已消，再继续跑最小费用最大流．

二物流 水源S1，水汇T1，油源S2，油汇T2，每根管道流量共用．求流量和最大．
建超级源SS1汇TT1，连边SS1→S1,SS1→S2,T1→TT1,T2→TT1，设最大流为x1．
建超级源SS2汇TT2，连边SS2→S1,SS2→T2,T1→TT2,S2→TT2，设最大流为x2．
则最大流中水流量$\frac{x1+x2}{2}$，油流量$\frac{x1-x2}{2}$．

3.20.11 割点于割边

割点 割边 一个点u是割点，当且仅当：1．u为非树根且有树边(u, v)满足$dfn_u \leq low_v$；2．u为树根且有多于一个的子树．一条无向边(u, v)是桥，当且仅当(u, v)是树边，且满足$dfn_u < low_v$．

3.20.12 2-SAT

如果选A就必须选B就从A向B连一条边，如果两个只能选一个的条件在同一个强连通分量中就不合法．输出可行方案可以比较X和X'的bl的大小，大的选X'．建图优化一般考虑前后缀的合并．

3.20.13 弦图

定义 我们称连接环中不相邻的两个点的边为弦．一个无向图称为弦图，当图中任意长度都大于3 的环都至少有一个弦．弦图的每一个诱导子图一定是弦图．

单纯点 设$N(v)$表示与点v相邻的点集．一个点称为单纯点当$v+N(v)$的诱导子图为一个团．引理：任何一个弦图都至少有一个单纯点，不是完全图的弦图至少有两个不相邻的单纯点．

完美消除序列 一个序列$v_1, v_2, ..., v_n$满足v_i在$v_i, v_{i+1}, ..., v_n$的诱导子图中为一个单纯点．一个无向图是弦图当且仅当它有一个完美消除序列．

最大势算法 最大势算法能判断一个图是否是弦图．从n到1的顺序依次给点标号 标号为i 的点出现在完美消除序列的第i个．设$label_i$表示第i个点与多少个已标号的点相邻，每次选择$label_i$最大的未标号的点进行标号．

然后判断这个序列是否为完美序列．如果依次判断$v_{i+1}, ..., .v_n$ 中所有与v_i相邻的点是否构成一个团，时间复杂度为$O(nm)$．考虑优化，设$v_{i+1}, ..., v_n$中所有与v_i 相邻的点依次为$v_{j1},...,v_{jk}$．只需判断v_{j1}是否与$v_{j2},...,v_{jk}$相邻即可．时间复杂度$O(n + m)$．

弦图的染色 按照完美消除序列中的点倒着给图中的点贪心染尽可能最小的颜色，这样一定能用最少的颜色数给图中所有点染色．弦图的团数=染色数．

最大独立集 完美消除序列从前往后能选就选．最大独立集=最小团覆盖．

3.20.14 三元环

有一种简单的写法，对于每条无向边(u, v)，如果$deg_u < deg_v$，那么连有向边(u, v)，否则连有向边(v, u)(注意度数相等以点标号为第二关键字判断)．然后枚举每个点x，假设x 是三元环中度数最小的点，然后暴力往后面枚两条边找到点y，判断是否有边(x, y)即可．可以证明，这样的时间复杂度也是为$O(m\sqrt{m})$的．

3.20.15 图同构

令$F_t(i) = (F_{t-1}(i)*A+\sum_{i\to j} F_{t-1}(j)*B+\sum_{j\to i} F_{t-1}(j)*C + D*(i-a))modP$，枚举点$a$，迭代$K$次后求得的就是$a$点所对应的$hash$值，其中$K, A, B, C, D, P$为$hash$参数，可自选．

3.20.16 竞赛图存在 Landau's Theorem

n个点竞赛图点按出度按升序排序，前i个点的出度之和不小于$\frac{i(i-1)}{2}$，度数总和等于$\frac{n(n-1)}{2}$．否则可以用优先队列构造出方案．

3.20.17 Ramsey Theorem

6个人中至少存在3人相互认识或者相互不认识．$R(3, 3) = 6, R(4, 4) = 18$

3.20.18 树的计数 Prufer序列

树和其prufer编码一一对应，一颗n个点的树，其prufer编码长度为$n - 2$，且度数为d_i 的点在prufer 编码中出现$d_i - 1$次．

由树得到序列：总共需要$n - 2$步，第i步在当前的树中寻找具有最小标号的叶子节点，将与其相连的点的标号设为Prufer序列的第i个元素p_i，并将此叶子节点从树中删除，直到最后得到一个长度为$n - 2$的Prufer 序列和一个只有两个节点的树．

由序列得到树：先将所有点的度赋初值为1，然后加上它的编号在Prufer序列中出现的次数，得到每个点的度；执行$n - 2$步，第i步选取具有最小标号的度为1的点u与$v = p_i$ 相连，得到树中的一条边，并将u和v 的度减一．最后再把剩下的两个度为1的点连边，加入到树中．

相关结论：n个点完全图，每个点度数依次为$d_1,d_2,...,d_n$，这样生成树的棵树为：$\frac{(n-2)!}{(d_1-1)!(d_2-1)!...(d_n-1)!}$．

左边有n_1个点，右边有n_2个点的完全二分图的生成树棵树为$n_1^{n_2-1} + n_2^{n_1-1}$．

m个连通块，每个连通块有c_i个点，把他们全部连通的生成树方案数：$(\sum c_i)^{m-2} \prod c_i$

3.20.19 有根树的计数

首先，令$S_{n,j} = \sum_{1 \leq j \leq n/j}$；于是$n + 1$个结点的有根树的总数为$a_{n+1} = \frac{\sum_{j=1}^{n} ja_jS_{n-j}}{n}$．注：$a_1 = 1, a_2 = 1, a_3 = 2, a_4 = 4, a_5 = 9, a_6 = 20, a_9 = 286, a_11 = 1842$．

3.20.20 无根树的计数

当n是奇数时，有$a_n - \sum_{i}^{n/2} a_ia_{n-i}$种不同的无根树．

当n时偶数时，有$a_n - \sum_{i}^{n/2} a_ia_{n-i} + \frac{1}{2}a_{n/2}(a_{n/2} + 1)$种不同的无根树．

3.20.21 生成树计数 Kirchhoff's Matrix-Tree Thoerem

Kirchhoff Matrix $T = Deg - A$, Deg是度数对角阵, A是邻接矩阵. 无向图度数矩阵是每个点度数; 有向图度数矩阵是每个点入度.
邻接矩阵$A[u][v]$表示$u->v$边个数, 重边按照边数计算, 自环不计入度数.
无向图生成树计数: $c = |K\text{的任意}1\text{个}n-1\text{阶主子式}|$
有向图外向树计数: $c = |\text{去掉根所在的那阶得到的主子式}|$

3.20.22 有向图欧拉回路计数 BEST Thoerem

$$\mathrm{ec}(G) = t_w(G)\prod_{v\in V}(\deg(v)-1)!$$

其中deg为入度(欧拉图中等于出度), $t_w(G)$为以w为根的外向树的个数. 相关计算参考生成树计数.
欧拉连通图中任意两点外向树个数相同: $\mathrm{t_v}(G) = \mathrm{t_w}(G)$.

3.20.23 Edmonds Matrix

Edmonds matrix A of a balanced ($|U| = |V|$) bipartite graph $G = (U, V, E)$:

$$A_{ij} = \begin{cases} x_{ij} & (u_i, v_j) \in E \\ 0 & (u_i, v_j) \notin E \end{cases}$$

where the x_{ij} are indeterminates. G有完美匹配当且仅当关于x_{ij}的多项式$det(A_{ij})$不恒为0. 完美匹配的个数等于多项式中单项式的个数.

3.20.24 Count Acyclic Orientations

The chromatic polynomial is a function $P_G(q)$ that counts the number of q-colorings of G.
Triangle K_3 : $t(t-1)(t-2)$
Complete graph K_n : $t(t-1)(t-2)\cdots(t-(n-1))$
Tree with n vertices : $t(t-1)^{n-1}$
Cycle C_n : $(t-1)^n + (-1)^n(t-1)$
The number of acyclic orientations of an n-vertex graph G is $(-1)^n P_G(-1)$, where $P_G(q)$ is the chromatic polynomial of the graph G.

4. String

4.1 KMP 扩展 KMP 循环最小表示

```
void kmp(string A,int*p){//A 为模式串, p 为失配数组
    int n=A.length(),i=1,j=0;
    for(CL(p);i<n;i++){
        for(;j&&A[j]^A[i];j=p[j-1]);
        if(A[i]==A[j])j++;
        p[i]=j;}
}int gans(string A,string B,int*p){
//B 为标准串, A 为待匹配串, p 为失配数组
    int n=B.length(),m=A.length(),j=0;
    fr(i,0,m-1){
        for(;j&&B[j]^A[i];j=p[j-1]);
        if(B[j]==A[i])j++;
        if(j==n)an++,j=p[j-1];
    } return an; }
void exkmp(char *s, int *a, int n) {
// 如果想求一个字符串相对另外一个字符串的最长公共前缀, 可以把他
  ↪ 们拼接起来从而求得
    a[0] = n; int p = 0, r = 0;
    for (int i = 1; i < n; ++i) {
        a[i] = (r > i) ? min(r - i, a[i - p]) : 0;
        while (i + a[i] < n && s[i + a[i]] == s[a[i]]) ++a[i];
        if (r < i + a[i]) r = i + a[i], p = i;
}}string mini(string A){// 最小表示法, 返回最小表示
    string AN;AN.clear();
    int i=0,j=1,k=0,x,n=A.length();
    for(;j<n&&k<n;){
        x=A[(i+k)%n]-A[(j+k)%n];
        x?x>0?i=max(j,i+k+1),j=i+1 :j=j+k+1,k=0:k++;
    }for(j=0;j<n;j++)AN=AN+A[(i+j)%n];
    return AN;}
```

4.2 Manacher

```
// 这段代码仅仅处理奇回文□使用时请往字符串中间加入 # 来使用
for(int i = 1, j = 0; i != (n << 1) - 1; ++i){
    int p=i>>1, q = i - p, r = ((j + 1) >> 1) + l[j] - 1;
    l[i] = r < q ? 0 : min(r - q + 1, l[(j << 1) - i]);
    while (p - l[i] != -1 && q + l[i] != n
        && s[p - l[i]] == s[q + l[i]]) l[i]++;
    if(q + l[i] - 1 > r) j=i;
```

```
    a += l[i];
}
```

4.3 Aho-Corasick Automation AC 自动机

```
for(i=0;i<26;i++)c[0][i]=1;//root为1
for(q[t=1]=1;h<t;)for(x=q[++h],i=0;i<26;i++){
    y=c[x][i];if(!y){c[x][i]=c[F[x]][i];continue;}
    for(k=F[x];!c[k][i];k=F[k]);
    F[y]=c[k][i];dg[y]|=dg[F[y]];q[++t]=y;}
```

4.4 Lydon Word Decomposition

```
//满足s的最小后缀等于s本身的串s称为Lyndon串.
//等价于: s是它自己的所有循环移位中唯一最小的一个.
//任意字符串s可以分解为s = s_1 s_2 ... s_k,其中s_i是Lyndon串,
  ↪ s_i ≥ s_{i+1}.且这种分解方法是唯一的.
void mnsuf(char *s, int *mn, int n) { // 每个前缀的最小后缀
    for (int i = 0; i < n; ) {
        int j = i, k = i + 1; mn[i] = i;
        for (; k < n && s[j] <= s[k]; ++ k)
            if (s[j] == s[k]) mn[k] = mn[j] + k - j, ++j;
                else mn[k] = j = i;
        for (; i <= j; i += k - j) {} } } // lyn+=s[i..i+k-j-1]
void mxsuf(char *s, int *mx, int n) { // 每个前缀的最大后缀
    fill(mx, mx + n, -1);
    for (int i = 0; i < n; ) {
        int j = i, k = i + 1; if (mx[i] == -1) mx[i] = i;
        for (; k < n && s[j] >= s[k]; ++k) {
            j = s[j] == s[k] ? j + 1 : i;
            if (mx[k] == -1) mx[k] = i; }
        for (; i <= j; i += k - j) {} } }
```

4.5 Suffix Array 后缀数组

```
//不用倍长数组 算height结尾要补-1 string和rank都是0-base
int rk[N], height[N], sa[N];
int cmp(int *x,int a,int b,int d){
    return x[a]==x[b]&&x[a+d]==x[b+d]; }
void doubling(int *a,int n,int m){
    static int sRank[N],tmpA[N],tmpB[N];
    int *x=tmpA,*y=tmpB;
    for(int i=0;i<m;++i) sRank[i]=0;
    for(int i=0;i<n;++i) ++sRank[x[i]=a[i]];
    for(int i=1;i<m;++i) sRank[i]+=sRank[i-1];
    for(int i=n-1;i>=0;--i) sa[--sRank[x[i]]]=i;
    for(int d=1,p=0;p<n;m=p,d<<=1){
        p=0; for(int i=n-d;i<n;++i) y[p++]=i;
        for(int i=0;i<n;++i) if(sa[i]>=d) y[p++]=sa[i]-d;
        for(int i=0;i<m;++i) sRank[i]=0;
        for(int i=0;i<n;++i) ++sRank[x[i]];
        for(int i=1;i<m;++i) sRank[i]+=sRank[i-1];
        for(int i=n-1;i>=0;--i) sa[--sRank[x[y[i]]]]=y[i];
        swap(x,y); x[sa[0]]=0; p=1; y[n] = -1;
        for(int i=1;i<n;++i)
          ↪ x[sa[i]]=cmp(y,sa[i],sa[i-1],d)?p-1:p++; } }
void calcHeight(int *a, int n){
    for(int i=0;i<n;++i) rk[sa[i]]=i;
    int cur=0; for(int i=0;i<n;++i)
    if(rk[i]) {
        if(cur) cur--;
        for(;a[i+cur]==a[sa[rk[i]-1]+cur];++cur);
        height[rk[i]]=cur; } }
```

4.6 Suffix Automation 后缀自动机

```
struct SAM{
int np,id,st[N],F[N],c[N][26],b[N],bl[N],cnt[N],pos[N];
void in(){CLEARALL;np=id=1;}//必须调用
void add(int x,int z){
    int p=np,q,nq;np=c[p][x];
    if(np&&st[np]==st[p]+1)return;
    st[np=++id]=st[p]+1;pos[np]=z;
    for(;p&&!c[p][x];p=F[p])c[p][x]=np;
    if(!p)F[np]=1;else if(st[p]+1==st[q=c[p][x]])F[np]=q;else{
        if(st[np]==st[p]+1)
            nq=np,st[nq]=st[p]+1,F[nq]=F[q],F[q]=nq;
```

```cpp
        else st[nq=++id]=st[p]+1,pos[nq]=pos[q],F[nq]=F[q],F[q]=F[np]=nq;
        for(memcpy(c[nq],c[q],sizeof c[q]);p&&c[p][x]==q;p=F[p])c[p][x]=nq;
    }}
void work(){//计数□这里的cnt[]为出现次数□可以视情况更改
    int i;LL an=0;
    for(i=1;i<=id;i++)b[st[i]]++;
    for(i=1;i<=id;i++)b[i]+=b[i-1];
    for(i=1;i<=id;i++)bl[b[st[i]]--]=i;
    for(i=id;i;i--)cnt[F[bl[i]]]+=cnt[bl[i]];}
void get(string S){//在后缀自动机上走□L为当前长度□p为当前位置
    for(int p=1,L=0,i=0;i<S.length();i++){
        if(c[p][x=S[i]-'a'])p=c[p][x],L++;else{
            for(;!c[p][x]&&p;p=F[p]);
            if(!p)p=1,L=0;else L=st[p]+1,p=c[p][x];
        }
    }}}A;
```

4.7 Suffix Balanced Tree 后缀平衡树

```cpp
// 后缀平衡树每次在字符串开头添加或删除字符，考虑在当前字符串 S 前插入一个字符 c，那么相当于在后缀平衡树中插入一个新的后缀 cS，简单的话可以使用预处理哈希二分 LCP 判断两个后缀的大小作 cmp，直接写 set，时间复杂度 O(nlg^2n)．为了方便可以把字符反过来做
// 例题 : 加一个字符或删一个字符，同时询问不同子串个数
struct cmp{
    bool operator()(int a,int b){
        int p=lcp(a,b);//注意这里是后面加□lcp是反过来的
        if(a==p)return 0;if(b==p)return 1;
        return s[a-p]<s[b-p];}
};set<int,cmp>S;set<int,cmp>::iterator il,ir;
void del(){S.erase(L--);}//在后面删字符
void add(char ch){//在后面加字符
    s[++L]=ch;mx=0;il=ir=S.lower_bound(L);
    if(il!=S.begin())mx=max(mx,lcp(L,*--il));
    if(ir!=S.end())mx=max(mx,lcp(L,*ir));
    an[L]=an[L-1]+L-mx;S.insert(L);
}
LL getan(){printf("%lld\n",an[L]);}//询问不同子串个数
```

4.8 Total LCS 子串对目标求 LCS

```cpp
const int N = 2005; int H[N][N], V[N][N]; char s[N], t[N];
int main() { gets(s + 1); gets(t + 1);
    int n = (int) strlen(s + 1), m = (int) strlen(t + 1);
    for (int i = 1; i <= m; ++ i) H[0][i] = i;
    for (int i = 1; i <= n; ++ i) {
        for (int j = 1; j <= m; ++ j) {
            if (s[i] == t[j]) {
                H[i][j] = V[i][j - 1]; V[i][j] = H[i - 1][j];
            } else {
                H[i][j] = max(H[i-1][j], V[i][j-1]);
                V[i][j] = min(H[i-1][j], V[i][j-1]); } } }
    for (int i = 1; i <= m; ++ i) { int ans = 0;
        for (int j = i; j <= m; ++ j) { ans += H[n][j] < i;
            printf("%d%c", ans, " \n"[j == m]); } } }
```

4.9 Palindrome Automation 回文自动机

```cpp
char s[N];int id,cnt[N],F[N],num[N],L[N],c[N][26],f[N],_f[N],pre[N],df[N],sk[N];
void add(int z,int n){ //注意这里的n是目前添加的字符数
    for(;s[n-L[p]-1]!=s[n];)p=F[p]; //失配后找一个尽量最长的
    if(!c[p][z]){ //这个回文串没有出现过，出现了新的本质不同的回文串
        int q=++id,k=F[p]; //新建节点
        for(L[q]=L[p]+1;s[n-L[k]-1]!=s[n];k=F[k]);//失败指针
        F[q]=c[k][z];c[p][z]=q;num[q]=num[F[q]]+1;
        df[q]=L[q]-L[F[q]];
        sk[q]=(df[q]==df[F[q]]?sk[F[q]]:F[q]);
    }p=c[p][z];cnt[p]++; //统计该类回文出现次数，这里的p即为last
}void init(){
    id=1;F[0]=F[1]=1;L[1]=-1; //一开始两个节点，0表示偶数长度串的根，1表示奇数长度串的根
    scanf("%s",s+1);n=strlen(s+1);_f[0]=1;
    for(int i=1;i<=n;i++){
        add(s[i]-'a',i);f[i]=1e9;
        for(int x=p;x;x=sk[x]){ //这里是求最小回文分割次数
            _f[x]=i-L[sk[x]]-df[x]; //先更新_f[x]，再用_f[x]更新f[x]
            if(df[F[x]]==df[x]&&f[_f[x]]>f[_f[F[x]]])
                _f[x]=_f[F[x]];
            if(f[i]>f[_f[x]]+1)
                f[i]=f[_f[x]]+1,pre[i]=_f[x];
        }}for(i=id;i;i--)cnt[F[i]]+=cnt[i]; }
```

4.10 Tips

4.10.1 双回文串

如果$s = x_1x_2 = y_1y_2 = z_1z_2, |x_1| < |y_1| < |z_1|, x_2, y_1, y_2, z_1$是回文串，则$x_1$和$z_2$也是回文串．

4.10.2 Border 的结构

字符串s的所有不小于$|s|/2$的border长度组成一个等差数列．
字符串s的所有 border 按长度排序后可分成$O(\log|s|)$段，每段是一个等差数列．回文串的回文后缀同时也是它的border．

4.10.3 子串最小后缀

设$s[p..n]$是$s[i..n]$, $(l \le i \le r)$中最小者，则minsuf(l, r)等于$s[p..r]$的最短非空 border．$\mathrm{minsuf(l, r)} = \min\{s[p..r], \mathrm{minsuf(r} - 2^k + \mathrm{1, r)}\}$, $(2^k < r{-}l+1 \le 2^{k+1})$.

4.10.4 子串最大后缀

从左往右扫，用set维护后缀的字典序递减的单调队列，并在对应时刻添加”小于事件”点以便在之后修改队列；查询直接在set里lower_bound.

5. Math 数学

5.1 CRT 中国剩余定理

```cpp
bool crt_merge(LL a1, LL m1, LL a2, LL m2, LL &A, LL &M) {
LL c = a2 - a1, d = __gcd(m1, m2); //合并两个模方程
if(c % d) return 0; // gcd(m1, m2) | (a2 - a1)时才有解
c = (c % m2 + m2) % m2; c /= d; m1 /= d; m2 /= d;
c = c * inv(m1 % m2, m2) % m2; //0逆元可任意值
M = m1*m2*d; A = (c *m1 %M *d %M +a1) % M; return 1;}//有解
```

5.2 扩展卢卡斯

```cpp
int l,a[33],p[33],P[33];
U fac(int k,LL n){// 求 n! mod pk^tk, 返回值 U{ 不包含 pk 的值 ,pk 出现的次数 }
    if (!n)return U{1,0};LL x=n/p[k],y=n/P[k],ans=1;int i;
    if(y){// 求出循环节的答案
        for(i=2;i<P[k];i++)if(i%p[k])ans=ans*i%P[k];
        ans=Pw(ans,y,P[k]);
    }for(i=y*P[k];i<=n;i++) if(i%p[k])ans=ans*i%M;// 求零散部分
    U z=fac(k,x);return U{ans*z.x%M,x+z.z};
}LL get(int k,LL n,LL m){// 求 C(n,m) mod pk^tk
    U a=fac(k,n),b=fac(k,m),c=fac(k,n-m);// 分三部分求解
    return Pw(p[k],a.z-b.z-c.z,P[k])*a.x%P[k]*inv(b.x,P[k])%P[k]*inv(c.x,P[k])%P[k];
}LL CRT(){// CRT 合并答案
    LL d,w,y,x,ans=0;
    fr(i,1,l)w=M/P[i],exgcd(w,P[i],x,y),ans=(ans+w*x%M*a[i])%M;
    return (ans+M)%M;
}LL C(LL n,LL m){// 求 C(n,m)
    fr(i,1,l)a[i]=get(i,n,m);
    return CRT();
}LL exLucas(LL n,LL m,int M){
    int jj=M,i; // 求 C(n,m)mod M,M=prod(pi^ki), 时间 O(pi^kilg^2n)
    for(i=2;i*i<=jj;i++)if(jj%i==0) for(p[++l]=i,P[l]=1;jj%i==0;P[l]*=p[l])jj/=i;
    if(jj>1)l++,p[l]=P[l]=jj;
    return C(n,m);}
```

5.3 Factorial Mod 阶乘取模

```cpp
// n! mod p^q Time : $O(pq^2 \frac{\log^2 n}{\log p})$
// Output : {a, b} means a*p^b
using Val=unsigned long long; //Val 需要 mod p^q 意义下 + *
typedef vector<Val> poly;
poly polymul(const poly &a,const poly &b){
  int n = (int) a.size(); poly c (n, Val(0));
  for (int i = 0; i < n; ++ i) {
    for (int j = 0; i + j < n; ++ j) {
      c[i + j] = c[i + j] + a[i] * b[j]; } }
  return c; } Val choo[70][70];
poly polyshift(const poly &a, Val delta) {
  int n = (int) a.size(); poly res (n, Val(0));
  for (int i = 0; i < n; ++ i) { Val d = 1;
    for (int j = 0; j <= i; ++ j) {
      res[i - j] = res[i - j]+a[i]*choo[i][j]*d;
      d = d * delta; } } return res; }
void prepare(int q) {
  for (int i = 0; i < q; ++ i) { choo[i][0] = Val(1);
    for (int j = 1; j <= i; ++ j)
      choo[i][j]=choo[i-1][j-1]+choo[i-1][j]; } }
pair<Val, LL> fact(LL n, LL p, LL q) { Val ans = 1;
  for (int r = 1; r < p; ++ r) {
    poly x (q, Val(0)), res (q, Val(0));
    res[0] = 1; LL _res = 0; x[0] = r; LL _x = 0;
    if (q > 1) x[1] = p, _x = 1; LL m = (n - r + p) / p;
    while (m) { if (m & 1) {
        res=polymul(res,polyshift(x,_res)); _res+=_x; }
      m >>= 1; x = polymul(x, polyshift(x, _x)); _x+=_x; }
    ans = ans * res[0]; }
  LL cnt = n / p; if (n >= p) { auto tmp=fact(n / p, p, q);
    ans = ans * tmp.first; cnt += tmp.second; }
  return {ans, cnt}; }
```

5.4 平方剩余

```cpp
// x^2=a (mod p),0 <=a<p,返回 true or false 代表是否存在解
// p必须是质数，若是多个单次质数的乘积，可以分别求解再用CRT合并
// 复杂度为 O(log n)
void multiply(ll &c, ll &d, ll a, ll b, ll w) {
  int cc = (a * c + b * d % MOD * w) % MOD;
  int dd = (a * d + b * c) % MOD; c = cc, d = dd; }
bool solve(int n, int &x) {
  if (n==0) return x=0,true; if (MOD==2) return x=1,true;
  if (power(n, MOD / 2, MOD) == MOD - 1) return false;
  ll c = 1, d = 0, b = 1, a, w;
  // finding a such that a^2 - n is not a square
  do { a = rand() % MOD; w = (a * a - n + MOD) % MOD;
    if (w == 0) return x = a, true;
  } while (power(w, MOD / 2, MOD) != MOD - 1);
  for (int times = (MOD + 1) / 2; times; times >>= 1) {
    if (times & 1) multiply(c, d, a, b, w);
    multiply(a, b, a, b, w); }
  // x = (a + sqrt(w)) ^ ((p + 1) / 2)
  return x = c, true; }
```

5.5 Baby-step Giant-step BSGS 离散对数

```cpp
LL inv(LL a,LL n){LL x,y;exgcd(a,n,x,y);return(x+n)%n;}
LL bsgs(LL a,LL b,LL n){// 在 (a,n)=1 时求最小的 x 使得 a^x mod n=b
  LL m=sqrt(n+0.5),e=1,i;map<LL,LL>mp;mp[1]=0;
  for(i=1;i<m;i++)if(!mp.count(e=e*a%n))mp[e]=i;
  e=e*a%n;e=inv(e,n);// e=a^m, 求出其逆元后放到等式右边
  for(i=0;i<m;b=b*e%n,i++)if(mp.count(b))return i*m+mp[b];
  return -1;// 无解
}LL exbsgs(LL a,LL b,LL n){// 求最小的 x 使 a^x mod n=b
  LL V,k=0,d,e=1;
  for(;(d=gcd(a,n))!=1;){
    if(b%d)return b==1?0:-1;// 如果 (a,n)=1, 要么 x=0&b=1, 要么无解
    k++;n=n/d;b=b/d;e=e*a/d%n;
    if(e==b)return k; }// 特判
  V=bsgs(a,b*inv(e,n)%n,n);return ~V?V+k:V;}// 有解返回 V+k
```

5.6 线性同余不等式

```cpp
// Find the minimal non-negtive solutions for $l \le d \cdot x \bmod m \le r$
// $0 \le d, l, r < m; l \le r, O(\log n)$
LL cal(LL m, LL d, LL l, LL r) {
  if (l==0) return 0; if (d==0) return MXL; // 无解
  if (d * 2 > m) return cal(m, m - d, m - r, m - l);
  if ((l - 1) / d < r / d) return (l - 1) / d + 1;
  LL k = cal(d, (-m % d + d) % d, l % d, r % d);
  return k==MXL ? MXL : (k*m + l - 1)/d+1;}// 无解 2
// return all x satisfying l1<=x<=r1 and l2<=(x*mul+add)%LIM<=r2
// here LIM = 2^32 so we use UI instead of "%".
// $O(\log p + \#solutions)$
struct Jump { UI val, step;
  Jump(UI val, UI step) : val(val), step(step) { }
  Jump operator + (const Jump & b) const {
    return Jump(val + b.val, step + b.step); }
  Jump operator - (const Jump & b) const {
    return Jump(val - b.val, step + b.step); }};
inline Jump operator * (UI x, const Jump & a) {
  return Jump(x * a.val, x * a.step); }
vector<UI> solve(UI l1, UI r1, UI l2, UI r2, pair<UI,UI> muladd) {
  UI mul = muladd.first, add = muladd.second, w = r2 - l2;
  Jump up(mul, 1), dn(-mul, 1); UI s(l1 * mul + add);
  Jump lo(r2 - s, 0), hi(s - l2, 0);
  function<void(Jump&, Jump&)> sub=[&](Jump& a, Jump& b){
    if (a.val > w) {
      UI t(((LL)a.val-max(0LL, w+1LL-b.val)) / b.val);
      a = a - t * b; } };
  sub(lo, up), sub(hi, dn);
  while (up.val > w || dn.val > w) {
    sub(up, dn); sub(lo, up);
    sub(dn, up); sub(hi, dn); }
  assert(up.val + dn.val > w); vector<UI> res;
  Jump bg(s + mul * min(lo.step, hi.step), min(lo.step, hi.step));
  while (bg.step <= r1 - l1) {
    if (l2 <= bg.val && bg.val <= r2)
      res.push_back(bg.step + l1);
    if (l2 <= bg.val-dn.val && bg.val-dn.val <= r2) {
      bg = bg - dn;
    } else bg = bg + up; }
  return res; }
```

5.7 Miller Rabin And Pollard Rho 启发式分解

```cpp
// Miller Rabin : bool miller_rabin::solve (const LL &) : tests whether a certain integer is prime.
typedef long long LL; struct miller_rabin {
int BASE[12] = {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37};
bool check (const LL &prime, const LL &base) {
  LL number = prime - 1;
  for (; ~number & 1; number >>= 1);
  LL result = llfpm (base, number, prime);
  for (; number != prime - 1 && result != 1 && result != prime - 1; number <<= 1)
    result = mul_mod (result, result, prime);
  return result == prime - 1 || (number & 1) == 1; }
bool solve (const LL &number) { // is prime
  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; } };
miller_rabin is_prime; const LL threshold = 13E9;
LL factorize (LL number, LL seed) {
  LL x = rand() % (number - 1) + 1, y = x;
  for (int head = 1, tail = 2; ; ) {
    x = mul_mod (x, x, number); x = (x + seed) % number;
```

```cpp
        if (x == y) return number;
        LL answer = gcd (abs (x - y), number);
        if (answer > 1 && answer < number) return answer;
        if (++head == tail) { y = x; tail <<= 1; } } }
void search (LL number, std::vector<LL> &divisor) {
    if (number <= 1) return;
    if (is_prime.solve (number)) divisor.push_back (number);
    else {
        LL factor = number;
        for (; factor >= number; factor = factorize (number, rand () % (number - 1) + 1));
        search (number / factor, divisor);
        search (factor, divisor); } }
```

5.8 杜教筛

```cpp
LL P(LL n){// 求欧拉函数前缀和
    if(n<M)return phi[n];//M=n^(2/3),phi[n] 为预处理前缀和
    LL x=1ll*n*(n+1)/2,i=2,la;
    for(;i<=n;i=la+1)la=n/(n/i),x-=P(n/i)*(la-i+1);
    return x;}
LL U(LL n){// 求莫比乌斯函数前缀和
    if(n<M)return u[n];// 预处理的前缀和
    LL x=1,i=2,la;
    for(;i<=n;i=la+1)la=n/(n/i),x-=U(n/i)*(la-i+1);
    return x;}
```

5.9 Extended Eratosthenes Sieve 扩展挨氏筛

```cpp
// 一般积性函数前缀和 要求：f(p) 为多项式
struct poly { LL a[2]; poly() {} int size() const {return 2;}
    poly(LL x, LL y) {a[0] = x; a[1] = y;} };
poly operator * (poly a, int p) {
    return poly(a.a[0], a.a[1] * p); }
poly operator - (const poly &a, const poly &b){
    return poly(a.a[0]-b.a[0], a.a[1]-b.a[1]);}
poly sum_fp(LL l, LL r) { // f(p) = 1 + p
    return poly(r-l+1, (l+r) * (r-l+1) / 2);}
LL fpk(LL p, LL k) { // f(p^k) = \sum_0^k (p^i)
    LL res = 0, q = 1;
    for (int i = 0; i <= k; ++ i) { res += q; q *= p; }
    return res; }
LL Value(poly p) { return p.a[0] + p.a[1]; }
LL n; int m; vector<poly> A, B; vector<int> P;
//need w = ⌊n/k⌋, about O(w^0.7)
LL calc(LL w, int id, LL f) {
    LL T = w>m ? Value(B[n/w]) : Value(A[w]);
    if (id) T -= Value(A[P[id - 1]]); LL ret = T * f;
    for (int i = id; i < P.size(); ++ i) {
        int p = P[i], e = 1; LL q = (LL) p*p; if (q>w) break;
        ret += calc(w/p, i+1, f * fpk(p, 1));
        while (1) {
            ++ e; LL f2 = f * fpk(p, e); ret+=f2; LL qq = q*p;
            if (qq <= w) {
                ret += calc(w/q, i+1, f2); q = qq;
            } else break; } }
    return ret; }
void prepare(LL N) { // about O(n^0.67)
    n = N; m = (int) sqrt(n + .5L);
    A.resize(m + 1); B.resize(m + 1);
    P.clear(); vector<int> isp; isp.resize(m + 1, 1);
    for (int i = 1; i <= m; ++ i) {
        A[i] = sum_fp(2, i); B[i] = sum_fp(2, n / i); }
    for (int p = 2; p <= m; ++ p) {
        if (isp[p]) P.push_back(p);
        for (int j : P) { if (j * p > m) break;
            isp[j * p] = 0; if (j % p == 0) break; }
        if (!isp[p]) continue;
        poly d = A[p - 1]; LL p2 = (LL) p * p;
        int to = (int) min(n / p2, (LL) m);
        for (int i = 1; i <= m / p; ++ i)
            B[i] = B[i] - (B[i * p] - d) * p;
        for (int i = m / p + 1; i <= to; ++ i)
            B[i] = B[i] - (A[n / p / i] - d) * p;
        for (int i = m; i >= p2; -- i)
            A[i] = A[i] - (A[i / p] - d) * p; } }
/* main() : prepare(n); LL ans = calc(n, 0, 1); */
```

5.10 原根

```cpp
bool ok(LL g,LL p){// 验证 g 是否是 p 的原根
    for(int i=1;i<=t;i++)if(Pw(g,(p-1)/q[i],p)==1)return 0;
    return 1;
}LL primitive_root(LL p){// 求 p 的原根
    LL i,n=p-1,g=1;t=0;
    for(i=2;i*i<=n;i++)if(n%i==0)for(q[++t]=i;n%i==0;n/=i);
    if(n!=1)q[++t]=n;
    for(;;g++)if(ok(g,p))return g;}
//当gcd(a,m)=1时，使a^x ≡ 1(modm)成立最小正整数x称为a对于模m的阶，记ord_m(a).
//阶的性质：a^n ≡ 1(mod m)充要条件是ord_m(a)|n，可推出ord_m(a)|ψ(m).
//当ord_m(g)=ψ(m)时，则称g是模m的一个原根，则g^0,g^1,...,g^{ψ(m)-1}覆盖了m以内所有与m互质的数.
//并不是所有数都存在原根，模意义下存在原根充要条件：m=2,4,p^k,2p^k. 其中为p奇素数，k为正整数.
```

5.11 FFT NTT FWT 多项式操作

```cpp
// double 精度对10^9+7 取模最多可以做到2^20
// FFT(Reverse(FFT(a,N)),N)=Na, Reverse是a_0不变,1到N-1反过来
const int MOD = 1e9 + 7; const double PI = acos(-1);
typedef complex<double> Complex;
const int MAXN = 262144, L = 15, MASK = (1<<L) - 1;
Complex w[MAXN];
void FFTInit(int N) {
    for (int i = 0; i < N; ++i)
        w[i] = Complex(cos(2*i*PI/N), sin(2*i*PI/N));}
void FFT(Complex p[], int n) { FFTInit(N);
    for (int i = 1, j = 0; i < n - 1; ++i) {
        for (int s = n; j ^= s >>= 1, ~j & s;);
        if (i < j) swap(p[i], p[j]); }
    for (int d = 0; (1 << d) < n; ++d) {
        int m = 1 << d, m2 = m * 2, rm = n >> (d+1);
        for (int i = 0; i < n; i += m2) {
            for (int j = 0; j < m; ++j) {
                Complex &p1 = p[i+j+m], &p2 = p[i+j];
                Complex t = w[rm * j] * p1;
                p1 = p2 - t, p2 = p2 + t; } } } }
void FFT_inv(Complex p[],int n) {
    FFT(p,n); reverse(p + 1, p + n);
    for (int i = 0; i < n; ++ i) p[i] /= n; }
Complex A[MAXN], B[MAXN], C[MAXN], D[MAXN];
void mul(int *a, int *b, int N) {
    for (int i = 0; i < N; ++i) {
        A[i] = Complex(a[i] >> L, a[i] & MASK);
        B[i] = Complex(b[i] >> L, b[i] & MASK); }
    FFT(A, N), FFT(B, N);
    for (int i = 0; i < N; ++i) { int j = (N - i) % N;
        Complex da = (A[i]-conj(A[j]))*Complex(0, -0.5),
                db = (A[i]+conj(A[j]))*Complex(0.5, 0),
                dc = (B[i]-conj(B[j]))*Complex(0, -0.5),
                dd = (B[i]+conj(B[j]))*Complex(0.5, 0);
        C[j] = da * dd + da * dc * Complex(0, 1);
        D[j] = db * dd + db * dc * Complex(0, 1); }
    FFT(C, N), FFT(D, N);
    for (int i = 0; i < N; ++i) {
        LL  da = (LL)(C[i].imag() / N + 0.5) % MOD,
            db = (LL)(C[i].real() / N + 0.5) % MOD,
            dc = (LL)(D[i].imag() / N + 0.5) % MOD,
            dd = (LL)(D[i].real() / N + 0.5) % MOD;
        a[i]=((dd << (L*2)) + ((db+dc) << L) + da) %MOD;}}
// 4179340454199820289LL (4e18) 原根=3 两倍不会爆 LL
// 2013265921 原根=31 两倍平方不会爆 LL
// 998244353 原根=3 // 1004535809 原根=3 // 469762049 原根=3
void NTT(int *a,int n,int f=1){
    int i,j,k,m,u,v,w,wm;
    for(i=n>>1,j=1,k;j<n-1;j++){
```

```
    if(i>j)swap(a[i],a[j]);
    for(k=n>>1;k<=i;k>>=1)i^=k;i^=k;
  }for(m=2;m<=n;m<<=1)
    for(i=0,wm=Pw(G,f==1?(M-1)/m:(M-1)/m*(m-1),M);i<n;i+=m)
      for(j=i,w=1;j<i+(m>>1);j++){
        u=a[j],v=1ll*w*a[j+(m>>1)]%M;
        if((a[j]=u+v)>=M)a[j]-=M;
        if((a[j+(m>>1)]=u-v)<0) a[j+(m>>1)]+=M;
        w=1ll*w*wm%M;
      }
  if(f==-1)for(w=Pw(n,M-2,M),i=0;i<n;i++)a[i]=1ll*a[i]*w%M;
}
void FWT(int w){//w=1为正运算，w=0为逆运算
  int i,j,k,x,y;
  for(i=1;i<N;i*=2)for(j=0;j<N;j+=i*2){
    for(k=0;k<i;k++) {
    x=a[j+k],y=a[i+j+k];
    if(w){
      //xor a[j+k]=x+y a[i+j+k]=x-y
      //and a[j+k]=x+y
      //or  a[i+j+k]=x+y
    }else{
      //xor a[j+k]=(x+y)/2 a[i+j+k]=(x-y)/2
      //and a[j+k]=x-y
      //or  a[i+j+k]=y-x
    } } } }
```

5.11.1 多项式牛顿法

已知函数$G(x)$，求多项式$F(x) \mod x^n$满足方程$G(F(x)) \equiv 0 \mod x^n$.

当$n=1$时，有$G(F(x)) \equiv 0 \mod x$，根据实际情况（逆元，二次剩余）求解．假设已经求出了$G(F_0(x)) \equiv 0 \mod x^n$，考虑扩展到$\mod x^{2n}$下：将$G(F(x))$在$F_0(x)$点泰勒展开，有

$$G(F(x)) = G(F_0(x)) + \frac{G'(F_0(x))}{1!}(F(x)-F_0(x)) + \cdots$$

因为$F(x)$和$F_0(x)$的最后n项相同，所以$(F(x)-F_0(x))^2$的最低的非0项次数大于$2n$，经过推导可以得到

$$F(x) \equiv F_0(x) - \frac{G(F_0(x))}{G'(F_0(x))} \mod x^{2n}$$

应用（以下复杂度均为$O(n\log n)$）：

多项式求逆（给定$A(x)$，求$A(x)B(x) \equiv 1 \mod x^n$）：构造方程$A(x)B(x)-1 \equiv 0 \mod x^n$，初始解$G_{invA}(B(x)) \equiv A[0]^{-1} \mod x$，递推式$F(x) \equiv 2F_0(x) - A(x)F_0^2(x) \mod x^{2n}$

多项式开方（给定$A(x)$，求$B^2(x) \equiv A(x) \mod x^n$）：初始解$G_{sqrtA}(B(x)) \equiv \sqrt{A[0]} \mod x$，递推式$F(x) \equiv \frac{F_0^2(x)+A(x)}{2F_0(x)} \mod x^{2n}$

多项式对数（给定常数项为1的$A(x)$，$B(x) \equiv \ln A(x)$）：对x求导得$(\ln A(x))' = \frac{A'(x)}{A(x)}$，使用多项式求逆，再积分回去$\ln A(x) \equiv \int \frac{A'(x)}{A(x)}$

多项式指数（给定常数项为0 的$A(x)$，求$B(x) \equiv e^{A(X)}$）：初始解$G_{expA}(B(x)) \equiv 1$，递推式$F(x) \equiv F_0(x)(1-\ln F_0(x)+A(x))$

多项式任意幂次（给定$A(x)$，求$B(x) \equiv A^k(x), k \in Q$）：$A^k(x) \equiv e^{k\ln A(x)}$

```
void Inv(int*A,int*B,int n){ //注意数组大小2n
//多项式求逆，B = A^{-1},n需为2的幂次
  static int C[N];B[0]=Pw(A[0],M-2,M);B[1]=0; //n=1时B[0] = A[0]^{-1}
  for(int m=2,i;m<=n;m<<=1){//递归转递推
    for(i=0;i<m;i++)C[i]=A[i];
    for(i=m;i<2*m;i++)C[i]=B[i]=0; //在模x^m意义下超过m次均为0
    NTT(C,m*2);NTT(B,m*2);
          //g(x) = g_0(x)(2 − f(x)g_0(x))( mod x^n)
    for(i=0;i<m*2;i++)
          B[i]=1ll*B[i]*(2-1ll*B[i]*C[i]%M+M)%M;
    NTT(B,m*2,-1);for(i=m;i<m*2;i++)B[i]=0;}
}
void Sqrt(int*A,int*B,int n){//多项式开根，B=sqrt(A)，n为2的幂次
  static int D[N],IB[N];
  B[0]=1;B[1]=0;//n=1时根据题意或二次剩余求解
  int I2=Pw(2,M-2,M),m,i;
  for(m=2;m<=n;m<<=1){//递归转递推
    for(i=0;i<m;i++)D[i]=A[i];
    for(i=m;i<2*m;i++)D[i]=B[i]=0;
    NTT(D,m*2);Inv(B,IB,m);NTT(IB,m*2);NTT(B,m*2);
    for(i=0;i<m*2;i++)
          B[i]=(1ll*B[i]*I2+1ll*I2*D[i]%M*IB[i])%M;
    NTT(B,m*2,-1);for(i=m;i<m*2;i++)B[i]=0;
  }
}
// 多项式除法：给定n次多项式A(x)和m ≤ n次多项式B(x)，求出D(x)，R(x)满足A(x) = D(x)B(x) + R(x)，并且degD ≤ n − m，degR < m，复杂度O(n log n)，常用于线性递推将2k项系数拍回k项时的优化：本质是将2k项的多项式除以k项零化多项式得到的余数
void Div(int *a, int n, int *b, int m, int *d, int *r) {
  // 注意这里n和m为多项式长度，注意需要4倍空间
  static int A[MAXN], B[MAXN]; while (!b[m - 1]) m --;
  int p = 1, t = n - m + 1; while (p < t << 1) p <<= 1;
  fill(A, A+p, 0); reverse_copy(b, b+m, A); Inv(A, B, p);
  fill(B+t, B+p, 0); NTT(B, p); reverse_copy(a, a+n, A);
  fill(A + t, A + p, 0); NTT(A, p);
  for (int i = 0; i < p; ++i) A[i] = 1LL*A[i]*B[i] % M;
  NTT(A, p, -1); reverse(A,A+t); copy(A,A+t,d); //lenD<=t
  for (p = 1; p < n; p <<= 1);
  fill(A + t, A + p, 0); NTT(A, p); copy(b, b + m, B);
  fill(B + m, B + p, 0); NTT(B, p);
  for (int i = 0; i < p; ++i) A[i] = 1LL*A[i]*B[i] % M;
  NTT(A, p, -1);
  for (int i = 0; i < m; ++i) r[i] = (a[i]-A[i]+M) % M;
  fill(r+m, r+p, 0); assert(r[m-1] == 0); } //lenR < m
```

5.11.2 多点求值 与 快速插值

多点求值：给出$F(x)$和$x_1,\cdots,x_n$，求$F(x_1),\cdots,F(x_n)$．考虑分治，设$L(x)=\prod_{i=1}^{\lfloor n/2\rfloor}(x-x_i)$，$R(x)=\prod_{i=\lfloor n/2\rfloor+1}^{n}(x-x_i)$，那么对于$1 \le i \le \lfloor n/2\rfloor$有$F(x_i)=(F \mod L)(x_i)$，对于$\lfloor n/2\rfloor < i \le n$有$F(x_i)=(F \mod R)(x_i)$．复杂度$O(n\log^2 n)$．

快速插值：给出n个x_i与y_i，求一个$n-1$次多项式满足$F(x_i)=y_i$．考虑拉格朗日插值：$F(x)=\sum_{i=1}^{n}\frac{\prod_{i\neq j}(x-x_j)}{\prod_{i\neq j}(x_i-x_j)}y_i$．对每个$i$先求出$\prod_{i\neq j}(x_i-x_j)$．设$M(x)=\prod_{i=1}^{n}(x-x_i)$，那么想要的是$\frac{M(x)}{x-x_i}$．取$x=x_i$时，上下都为0，使用洛必达法则，则原式化为$M'(x)$．使用分治算出$M(x)$，使用多点求值算出每个$\prod_{i\neq j}(x_i-x_j)=M'(x_i)$．设$\frac{y_i}{\prod_{i\neq j}(x_i-x_j)}=v_i$，现在要求出$\sum_{i=1}^{n}v_i\prod_{i\neq j}(x-x_j)$．使用分治：设$L(x)=\prod_{i=1}^{\lfloor n/2\rfloor}(x-x_i)$，$R(x)=\prod_{i=\lfloor n/2\rfloor+1}^{n}(x-x_i)$，则原式化为：$\left(\sum_{i=1}^{\lfloor n/2\rfloor}v_i\prod_{i\neq j,j\le\lfloor n/2\rfloor}(x-x_j)\right)R(x)+\left(\sum_{i=\lfloor n/2\rfloor+1}^{n}v_i\prod_{i\neq j,j>\lfloor n/2\rfloor}(x-x_j)\right)L(x)$，递归计算．复杂度$O(n\log^2 n)$．

5.12 多项式插值

```
// 拉格朗日插值 n 为点数，x 为要求的 f(x)，已知 f(X[i])=Y[i]
D lagrange(int n,D x,D X[],D Y[]){
  int i,j;D ans,v;fr(i,n){
    v=1;fr(j,n)if(i!=j)v*=(x-X[j])/(X[i]-X[j]);
    ans+=v*Y[i];}
  return ans;}
// 牛顿插值，给出 f(X[i])=Y[i]),i=0...n-1
void pre(){//O(n^2) 预处理
  fr(i,0,n-1)f[i][0]=Y[i];
  fr(i,1,n-1)fr(j,1,i) f[i][j]=(f[i][j-1]-f[i-1][j-1])/(X[i]-X[i-j]); }
D getfx(D x){//O(n) 询问单点值
  D an=f[0][0],v=1;
```

```
    fr(i,1,n-1)v*=(x-X[i-1]),an+=f[i][i]*v;
    return an;}
```

5.13 Simplex 单纯形

```
double an[N*2];int q[N*2];
void pivot(int l,int e){//转轴操作
    int i,j;a[l][e]=1/a[l][e];b[l]*=a[l][e];
    fr(i,1,n)if(i!=e)a[l][i]*=a[l][e];
    fr(i,1,m)if(i!=l&&fabs(a[i][e])>eps){
        b[i]-=a[i][e]*b[l];
        fr(j,1,n)if(j!=e)a[i][j]-=a[i][e]*a[l][j];
        a[i][e]*=-a[l][e];
    }v+=c[e]*b[l];
    fr(i,1,n)if(i!=e)c[i]-=c[e]*a[l][i];
    c[e]*=-a[l][e];swap(q[e],q[l+n]);//记录方案
}void pre(){//特殊处理b_i<0的情况，没有这种情况可不写
    for(;;){
        dr(i,m,1)if(b[i]<-eps)break;
        if(!i)break;
        dr(j,n,1)if(a[i][j]<-eps)break;
        if(!j)return puts("Infeasible"),0;//无解
        pivot(i,j);}
}//n为变量数，m为约束数，令目标函数$F=\sum c_jx_j$最大，第i条约束
  ↪ 为$\sum a_{ij}x_j <= b_i$
int main(int n,int m,double c[N],double b[M],double a[M][N]){
    //输出答案，并输出此时$x_1,x_2,...,x_n$的值
    double v,p;int l,e;fr(i,1,n)q[i]=i;
    for(pre();;pivot(l,e)){//寻找更优解
        dr(i,n,1)if(c[i]>eps)break;//选择系数大于0的
        if(!i)break;//找到最优解
        p=inf;e=i;fr(i,1,m)if(a[i][e]>eps&&p>b[i]/a[i][e])
            p=b[i]/a[i][e],l=i;
        if(p==inf)return puts("Unbounded"),0; //找不到可行的基本
          ↪ 变量，最优解无穷大
    }
    printf("%.8f",v);//输出答案
    fr(i,n+1,n+m)an[q[i]]=b[i-n];
    fr(i,1,n)printf("%.8lf ",an[i]);}//输出方案
// 标准型：maximize $\mathbf{c^Tx}$, subject to $\mathbf{Ax}\le\mathbf{b}$ and $\mathbf{x}\ge\mathbf{0}$
// 对偶型：minimize $\mathbf{b^Ty}$, subject to $\mathbf{A^Tx}\ge\mathbf{c}$ and $\mathbf{y}\ge\mathbf{0}$
```

5.14 线性递推

```
// Complexity: init O(n^2log) query O(n^2logk)
// Requirement: const LOG const MOD
// Example: In: {1, 3} {2, 1} an = 2an-1 + an-2
//          Out: calc(3) = 7
typedef vector<int> poly;
struct LinearRec {
    int n; poly first, trans; vector<poly> bin;
poly add(poly &a, poly &b) {
    poly res(n * 2 + 1, 0);
    // 不要每次新开 vector, 可以使用矩阵乘法优化
    for (int i = 0; i <= n; ++i) {
        for (int j = 0; j <= n; ++j) {
            (res[i+j]+=(LL)a[i] * b[j] % MOD) %= MOD;
    for (int i = 2 * n; i > n; --i) {
        for (int j = 0; j < n; ++j) {
            (res[i-1-j]+=(LL)res[i]*trans[j]%MOD) %=MOD;}
        res[i] = 0; }
    res.erase(res.begin() + n + 1, res.end());
    return res; }
LinearRec(poly &first, poly &trans): first(first),
  ↪ trans(trans) {
    n = first.size(); poly a(n + 1, 0); a[1] = 1;
    bin.push_back(a); for (int i = 1; i < LOG; ++i)
        bin.push_back(add(bin[i - 1], bin[i - 1])); }
int calc(int k) { poly a(n + 1, 0); a[0] = 1;
    for (int i = 0; i < LOG; ++i)
        if (k >> i & 1) a = add(a, bin[i]);
    int ret = 0; for (int i = 0; i < n; ++i)
        if ((ret += (LL)a[i + 1] * first[i] % MOD) >= MOD)
            ret -= MOD;
```

```
    return ret; }};
```

5.15 Berlekamp-Massey 最小多项式

```
// Complexity: O(n^2) Requirement: const MOD, inverse(int)
// Input: the first elements of the sequence
// Output: the recursive equation of the given sequence
// Example In: {1, 1, 2, 3}
// Example Out: {1, 1000000006, 1000000006} (MOD = 1e9+7)
struct Poly { vector<int> a; Poly() { a.clear(); }
    Poly(vector<int> &a): a(a) {}
    int length() const { return a.size(); }
    Poly move(int d) { vector<int> na(d, 0);
        na.insert(na.end(), a.begin(), a.end());
        return Poly(na); }
    int calc(vector<int> &d, int pos) { int ret = 0;
        for (int i = 0; i < (int)a.size(); ++i) {
            if ((ret+=(LL)d[pos - i]*a[i]%MOD) >= MOD) {
                ret -= MOD; }}
        return ret; }
    Poly operator - (const Poly &b) {
        vector<int> na(max(this->length(), b.length()));
        for (int i = 0; i < (int)na.size(); ++i) {
            int aa = i < this->length() ? this->a[i] : 0,
                bb = i < b.length() ? b.a[i] : 0;
            na[i] = (aa + MOD - bb) % MOD; }
        return Poly(na); } };
Poly operator * (const int &c, const Poly &p) {
    vector<int> na(p.length());
    for (int i = 0; i < (int)na.size(); ++i) {
        na[i] = (LL)c * p.a[i] % MOD; }
    return na; }
vector<int> solve(vector<int> a) {
    int n = a.size(); Poly s, b;
    s.a.push_back(1), b.a.push_back(1);
    for (int i = 0, j = -1, ld = 1; i < n; ++i) {
        int d = s.calc(a, i); if (d) {
            if ((s.length() - 1) * 2 <= i) {
                Poly ob = b; b = s;
                s=s-(LL)d*inverse(ld)%MOD*ob.move(i - j);
                j = i; ld = d;
            } else {
                s=s-(LL)d*inverse(ld)%MOD*b.move(i-j);}}}
    //Caution: s.a might be shorter than expected
    return s.a; }
/* 求行列式 -> 求特征多项式 : det(A)=(-1)^nPA(0)
 求矩阵或向量列最小多项式 : 随机投影成数列
 如果最小多项式里面有 x 的因子，那么行列式为 0，否则
 随机乘上对角阵 D, det(A)=det(AD)/det(D)*/
```

5.16 Pell 方程

```
// $x^2 - n*y^2 = 1$ 最小正整数根，n 为完全平方数时无解
// $x_{k+1} = x_0x_k + ny_0y_k$
// $y_{k+1} = x_0y_k + y_0x_k$
pair<LL, LL> peLL(LL n) {
    static LL p[N], q[N], g[N], h[N], a[N];
    p[1] = q[0] = h[1] = 1; p[0] = q[1] = g[1] = 0;
    a[2] = (LL)(floor(sqrtl(n) + 1e-7L));
    for(int i = 2; ; i ++) {
        g[i] = -g[i - 1] + a[i] * h[i - 1];
        h[i] = (n - g[i] * 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(p[i] * p[i] - n * q[i] * q[i] == 1)
            return {p[i], q[i]}; }}
```

5.17 解一元三次方程

```
double a(p[3]), b(p[2]), c(p[1]), d(p[0]);
double k(b / a), m(c / a), n(d / a);
double p(-k * k / 3. + m);
double q(2. * k * k * k / 27 - k * m / 3. + n);
```

```
Complex omega[3] = {Complex(1, 0), Complex(-0.5, 0.5 * sqrt(3)), Complex(-0.5, -0.5 * sqrt(3))};
Complex r1, r2; double delta(q * q / 4 + p * p * p / 27);
if (delta > 0) {
  r1 = cubrt(-q / 2. + sqrt(delta));
  r2 = cubrt(-q / 2. - sqrt(delta));
} else {
  r1 = pow(-q / 2. + pow(Complex(delta), 0.5), 1. / 3);
  r2 = pow(-q / 2. - pow(Complex(delta), 0.5), 1. / 3); }
for(int _(0); _ < 3; _++) {
  Complex x = -k/3. + r1*omega[_] + r2*omega[_* 2 % 3]; }
```

5.18 自适应 Simpson

```
// Adaptive Simpson's method : double simpson::solve (double (*f) (double), double l, double r, double eps) : integrates f over (l, r) with error eps.
struct simpson {
double area (double (*f) (double), double l, double r) {
  double m = l + (r - l) / 2;
  return (f (l) + 4 * f (m) + f (r)) * (r - l) / 6;
}
double solve (double (*f) (double), double l, double r, double eps, double a) {
  double m = l + (r - l) / 2;
  double left = area (f, l, m), right = area (f, m, r);
  if (fabs (left + right - a) <= 15 * eps) return left + right + (left + right - a) / 15.0;
  return solve (f, l, m, eps / 2, left) + solve (f, m, r, eps / 2, right);
}
double solve (double (*f) (double), double l, double r, double eps) {
  return solve (f, l, r, eps, area (f, l, r));
}};
```

5.19 Schreier-Sims 求排列群的阶

```
struct Perm{
  vector<int> P; Perm() {} Perm(int n) { P.resize(n); }
  Perm inv() const{ Perm ret(P.size());
    for(int i = 0; i < int(P.size()); ++i) ret.P[P[i]]=i;
    return ret; }
  int &operator [](const int &dn){ return P[dn]; }
  void resize(const size_t &sz){ P.resize(sz); }
  size_t size()const{ return P.size(); }
  const int &operator[](const int &dn)const{return P[dn];}};
Perm operator *(const Perm &a, const Perm &b){
  Perm ret(a.size());
  for(int i = 0; i < (int)a.size(); ++i) ret[i] = b[a[i]];
  return ret; }
typedef vector<Perm> Bucket; typedef vector<int> Table;
typedef pair<int,int> PII; int n, m;
vector<Bucket> buckets, bucketsInv;vector<Table> lookupTable;
int fastFilter(const Perm &g, bool addToGroup = true) {
  int n = buckets.size(); Perm p(g);
  for(int i = 0; i < n; ++i){
    int res = lookupTable[i][p[i]];
    if(res == -1) {
      if(addToGroup) { buckets[i].push_back(p);
        bucketsInv[i].push_back(p.inv());
        lookupTable[i][p[i]]=(int)buckets[i].size()-1; }
      return i; }
    p = p * bucketsInv[i][res]; }
  return -1; }
long long calcTotalSize() { long long ret = 1;
  for(int i = 0; i < n; ++i) ret *= buckets[i].size();
  return ret; }
bool inGroup(const Perm &g){return fastFilter(g,false)==-1;}
void solve(const Bucket &gen,int _n) {// m perm[0..n - 1]s
  n = _n, m = gen.size(); { //clear all
    vector<Bucket> _b(n); swap(buckets, _b);
    vector<Bucket> _I(n); swap(bucketsInv, _I);
    vector<Table> _T(n); swap(lookupTable, _T); }
  for(int i = 0; i < n; ++i){
    lookupTable[i].resize(n);
    fill(lookupTable[i].begin(),lookupTable[i].end(),-1);}
  Perm id(n); for(int i = 0; i < n; ++i) id[i] = i;
  for(int i = 0; i < n; ++i){
    buckets[i].push_back(id);bucketsInv[i].push_back(id);
    lookupTable[i][i] = 0; }
  for(int i = 0; i < m; ++i) fastFilter(gen[i]);
  queue<pair<PII,PII> > toUpdate;
  for(int i = 0; i < n; ++i)
    for(int j = i; j < n; ++j)
      for(int k = 0; k < (int) buckets[i].size(); ++k)
        for(int l = 0; l < (int) buckets[j].size(); ++l)
          toUpdate.push(make_pair(PII(i,k), PII(j,l)));
  while(!toUpdate.empty()){
    PII a=toUpdate.front().first;
    PII b=toUpdate.front().second; toUpdate.pop();
    int res = fastFilter(buckets[a.first][a.second] * buckets[b.first][b.second]);
    if(res==-1) continue;
    PII newPair(res, (int)buckets[res].size() - 1);
    for(int i = 0; i < n; ++i)
      for(int j = 0; j < (int)buckets[i].size(); ++j){
        if(i <= res)
          toUpdate.push(make_pair(PII(i, j), newPair));
        if(res <= i)
          toUpdate.push(make_pair(newPair,PII(i,j)));}}}
```

5.20 类欧几里得 直线下格点统计

// $\sum_{i=0}^{n-1}\lfloor\frac{a+bi}{m}\rfloor$, $n, m, a, b > 0$

```
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); }
```

6. Miscellany

6.1 Zeller 日期公式

```
// weekday=(id+1)%7;{Sun=0,Mon=1,...}   getId(1, 1, 1) = 0
int getId(int y, int m, int d) {
  if (m < 3) { y --; m += 12; }
  return 365 * y + y / 4 - y / 100 + y / 400 + (153 * (m - 3) + 2) / 5 + d - 307; }
//  当y小于0时，将除法改为向下取整后即可保证正确性，或统一加400的倍数年
auto date(int id) {
  int x=id+1789995, n, i, j, y, m, d;
  n = 4 * x / 146097; x -= (146097 * n + 3) / 4;
  i = (4000 * (x + 1)) / 1461001; x -= 1461 * i / 4 - 31;
  j = 80 * x / 2447; d = x - 2447 * j / 80; x = j / 11;
  m = j + 2 - 12 * x; y = 100 * (n - 49) + i + x;
  return make_tuple(y, m, d); }
```

6.2 Dancing Links 精确覆盖搜索

```
//main: prepare(C); for(i行)insert({为1的列},C); search(0);
struct node{
  node *left,*right,*up,*down,*col; int row,cnt;
}*head,*col[MAXC],Node[MAXNODE],*ans[MAXNODE];
int totNode, ansNode;
void insert(const std::vector<int> &V,int rownum){
  std::vector<node*> N;
  for(int i=0;i<int(V.size());++i){
    node* now=Node+(totNode++); now->row=rownum;
    now->col=now->up=col[V[i]], now->down=col[V[i]]->down;
    now->up->down=now, now->down->up=now;
    now->col->cnt++; N.push_back(now); }
  for(int i=0;i<int(V.size());++i)
    N[i]->right=N[(i+1)%V.size()], N[i]->left=N[(i-1+V.size())%V.size()]; }
void Remove(node *x){
  x->left->right=x->right, x->right->left=x->left;
```

```
    for(node *i=x->down;i!=x;i=i->down)
        for(node *j=i->right;j!=i;j=j->right)
            j->up->down=j->down, j->down->up=j->up, --(j->col-
              ↪ >cnt); }
void Resume(node *x){
    for(node *i=x->up;i!=x;i=i->up)
        for(node *j=i->left;j!=i;j=j->left)
            j->up->down=j->down->up=j, ++(j->col->cnt);
    x->left->right=x, x->right->left=x; }
bool search(int tot){
    if(head->right==head) return ansNode = tot, true;
    node *choose=NULL;
    for(node *i=head->right;i!=head;i=i->right){
        if(choose==NULL||choose->cnt>i->cnt) choose=i;
        if(choose->cnt<2) break; }
    Remove(choose);
    for(node *i=choose->down;i!=choose;i=i->down){
        for(node *j=i->right;j!=i;j=j->right) Remove(j->col);
        ans[tot]=i;
        if(search(tot+1)) return true;
        ans[tot]=NULL;
        for(node *j=i->left;j!=i;j=j->left) Resume(j->col); }
    Resume(choose); return false; }
void prepare(int totC){
    head=Node+totC;
    for(int i=0;i<totC;++i) col[i]=Node+i;
    totNode=totC+1; ansNode = 0;
    for(int i=0;i<=totC;++i){
        (Node+i)->right=Node+(i+1)%(totC+1);
        (Node+i)->left=Node+(i+totC)%(totC+1);
        (Node+i)->up=(Node+i)->down=Node+i;
        (Node+i)->cnt=0; } }
```

6.3 Java Template

```
import java.io.*; import java.util.*; import java.math.*;
public class Main {
static class solver { public void run(Scanner cin,
  ↪ PrintStream out) {} }
public static void main(String[] args) {
// Fast Reader & Big Numbers
InputReader in = new InputReader(System.in);
PrintWriter out = new PrintWriter(System.out);
BigInteger c = in.nextInt();
out.println(c.toString(8)); out.close(); // as Oct
BigDecimal d = new BigDecimal(10.0);
// d=d.divide(num, length, BigDecimal.ROUND_HALF_UP)
d.setScale(2, BigDecimal.ROUND_FLOOR); // 用于输出
System.out.printf("%.6f\n", 1.23); // C 格式
BigInteger num = BigInteger.valueOf(6);
num.isProbablePrime(10); // 1 - 1 / 2 ^ certainty
BigInteger rev = num.modPow(BigInteger.valueOf(-1),
  ↪ BigInteger.valueOf(25)); // rev=6^-1mod25 要互质
num = num.nextProbablePrime(); num.intValue();
Scanner cin=new Scanner(System.in);//SimpleReader
int n = cin.nextInt();
int [] a = new int [n]; // 初值未定义
// Random rand.nextInt(N) [0,N)
Arrays.sort(a, 5, 10, cmp); // sort(a+5, a+10)
ArrayList<Long> list = new ArrayList(); // vector
// .add(val) .add(pos, val) .remove(pos)
Comparator cmp=new Comparator<Long>(){ // 自定义逆序
  @Override public int compare(Long o1, Long o2) {
  /* o1 < o2 ? 1 :( o1 > o2  ? -1 : 0) */ } };
// Collections. shuffle(list) sort(list, cmp)
Long [] tmp = list.toArray(new Long [0]);
// list.get(pos) list.size()
Map<Integer,String> m = new HashMap<Integer,String>();
//m.put(key,val) get(key) containsKey(key) remove(key)
for (Map.Entry<Integer,String> entry:m.entrySet()); //
  ↪ entry.getKey() getValue()
Set<String> s = new HashSet<String>(); // TreeSet
//s.add(val)contains(val)remove(val);for(var : s)
solver Task=new solver();Task.run(cin,System.out);
PriorityQueue<Integer> Q=new PriorityQueue<Integer>();
// Q. offer(val) poll() peek() size()
// Read / Write a file : FileWriter FileReader PrintStream
} static class InputReader { // Fast Reader
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 { String line = reader.readLine();
      /*line == null ? end of file*/
      tokenizer = new StringTokenizer(line);
    } catch (IOException e) {
      throw new RuntimeException(e); }
  } return tokenizer.nextToken(); }
public BigInteger nextInt() {
  //return Long.parseLong(next()); Double Integer
  return new BigInteger(next(), 16); // as Hex
} } }
```

6.4 Vimrc

```
set ru nu ts=4 sts=4 sw=4 si sm hls is ar bs=2 mouse=a
syntax on
nm <F3> :vsplit %<.in <CR>
nm <F4> :!gedit % <CR>
au BufEnter *.cpp set cin
au BufEnter *.cpp nm <F5> :!time ./%< <CR>|
\nm <F7> :!gdb ./%< <CR>|nm <F8> :!time ./%< < %<.in <CR>|
\nm <F9> :!g++ % -o %< -g -std=gnu++14 -O2 && size %< <CR>
au BufEnter *.java nm <F5> :!time java %< <CR>|
\nm <F8> :!time java %< < %<.in <CR>|nm <F9> :!javac % <CR>
```

6.5 读入优化相关

```
#define __ __attribute__ ((optimize ("-O3")))
#define _ __ __inline __attribute__ ((__gnu_inline__,
  ↪ __always_inline__, __artificial__))
__inline int next_uint () {
    const int SIZE = 110000; static char buf[SIZE + 1]; static
      ↪ int p = SIZE;
    register int ans = 0, f = 1, sgn = 1;
    while ((p < SIZE || (p = 0, buf[fread (buf, 1, SIZE,
      ↪ stdin)] = 0, buf[0])) && (isdigit (buf[p]) && (ans =
      ↪ ans * 10 + buf[p] - '0', f = 0, 1) || f /* && (buf[p]
      ↪ == '-' && (sgn = 0), 1) */ )) ++p;
    return sgn ? ans : -ans; }
bitset._Find_first();bitset._Find_next(idx);
struct HashFunc{size_t operator()(const KEY &key)const{}};
```

7. Appendix

7.1 Formulas 公式表

7.1.1 Mobius Inversion

$$F(n)=\sum_{d|n}f(d)\Rightarrow f(n)=\sum_{d|n}\mu(d)F(\frac{n}{d})$$

$$F(n)=\sum_{n|d}f(d)\Rightarrow f(n)=\sum_{n|d}\mu(\frac{d}{n})F(d)$$

$$[x=1]=\sum_{d|x}\mu(d),\quad x=\sum_{d|x}\mu(d)$$

7.1.2 Arithmetic Function

$$(p-1)!\equiv -1(mod\ p)$$

$$a>1,m,n>0,\ \text{then}\,\gcd(a^m-1,a^n-1)=a^{\gcd(n,m)}-1$$

$$a>b,\gcd(a,b)=1,\ \text{then}\,\gcd(a^m-b^m,a^n-b^n)=a^{\gcd(m,n)}-b^{\gcd(m,n)}$$

$$\prod_{k=1,gcd(k,m)=1}^{m}k\equiv\begin{cases}-1 & \mod\ m,\ m=4,p^q,2p^q\\ 1 & \mod\ m,\ \text{otherwise}\end{cases}$$

$$\sigma_k(n)=\sum_{d|n}d^k=\prod_{i=1}^{\omega(n)}\frac{p_i^{(a_i+1)k}-1}{p_i^k-1}$$

$$J_k(n) = n^k \prod_{p|n}(1-\frac{1}{p^k})$$

$J_k(n)$ is the number of k-tuples of positive integers all less than or equal to n that form a coprime $(k+1)$-tuple together with n.

$$\sum_{\delta|n} J_k(\delta) = n^k$$

$$\sum_{\delta|n} \delta^s J_r(\delta) J_s(\frac{n}{\delta}) = J_{r+s}(n)$$

$$\sum_{\delta|n} \varphi(\delta) d(\frac{n}{\delta}) = \sigma(n),\ \sum_{\delta|n} |\mu(\delta)| = 2^{\omega(n)}$$

$$\sum_{\delta|n} 2^{\omega(\delta)} = d(n^2),\ \sum_{\delta|n} d(\delta^2) = d^2(n)$$

$$\sum_{\delta|n} d(\frac{n}{\delta}) 2^{\omega(\delta)} = d^2(n),\ \sum_{\delta|n} \frac{\mu(\delta)}{\delta} = \frac{\varphi(n)}{n}$$

$$\sum_{\delta|n} \frac{\mu(\delta)}{\varphi(\delta)} = d(n),\ \sum_{\delta|n} \frac{\mu^2(\delta)}{\varphi(\delta)} = \frac{n}{\varphi(n)}$$

$$n|\varphi(a^n-1)$$

$$\sum_{\substack{1\le k\le n \\ \gcd(k,n)=1}} f(\gcd(k-1,n)) = \varphi(n) \sum_{d|n} \frac{(\mu * f)(d)}{\varphi(d)}$$

$$\varphi(\mathrm{lcm}(m,n))\varphi(\gcd(m,n)) = \varphi(m)\varphi(n)$$

$$\sum_{\delta|n} d^3(\delta) = (\sum_{\delta|n} d(\delta))^2$$

$$d(uv) = \sum_{\delta|\gcd(u,v)} \mu(\delta) d(\frac{u}{\delta}) d(\frac{v}{\delta})$$

$$\sigma_k(u)\sigma_k(v) = \sum_{\delta|\gcd(u,v)} \delta^k \sigma_k(\frac{uv}{\delta^2})$$

$$\mu(n) = \sum_{k=1}^{n} [\gcd(k,n)=1] \cos 2\pi\frac{k}{n}$$

$$\varphi(n) = \sum_{k=1}^{n} [\gcd(k,n)=1] = \sum_{k=1}^{n} \gcd(k,n) \cos 2\pi\frac{k}{n}$$

$$\begin{cases} S(n) = \sum_{k=1}^{n} (f*g)(k) \\ \sum_{k=1}^{n} S(\lfloor\frac{n}{k}\rfloor) = \sum_{i=1}^{n} f(i) \sum_{j=1}^{\lfloor\frac{n}{i}\rfloor} (g*1)(j) \end{cases}$$

$$\begin{cases} S(n) = \sum_{k=1}^{n} (f\cdot g)(k), g \text{ completely multiplicative} \\ \sum_{k=1}^{n} S(\lfloor\frac{n}{k}\rfloor) g(k) = \sum_{k=1}^{n} (f*1)(k) g(k) \end{cases}$$

7.1.3 Binomial Coefficients

$$\binom{n}{k} = (-1)^k \binom{k-n-1}{k},\ \sum_{k\le n} \binom{r+k}{k} = \binom{r+n+1}{n}$$

$$\sum_{k=0}^{n} \binom{k}{m} = \binom{n+1}{m+1}$$

$$\sqrt{1+z} = 1 + \sum_{k=1}^{\infty} \frac{(-1)^{k-1}}{k\times 2^{2k-1}} \binom{2k-2}{k-1} z^k$$

$$\sum_{k=0}^{r} \binom{r-k}{m} \binom{s+k}{n} = \binom{r+s+1}{m+n+1}$$

$$C_{n,m} = \binom{n+m}{m} - \binom{n+m}{m-1}, n \ge m$$

$$\binom{n}{k} \equiv [n\&k=k] \pmod 2$$

$$\binom{n_1+\cdots+n_p}{m} = \sum_{k_1+\cdots+k_p=m} \binom{n_1}{k_1} \cdots \binom{n_p}{k_p}$$

7.1.4 Fibonacci Numbers

$$F(z) = \frac{z}{1-z-z^2}$$

$$f_n = \frac{\phi^n - \hat{\phi}^n}{\sqrt{5}}, \phi = \frac{1+\sqrt{5}}{2}, \hat{\phi} = \frac{1-\sqrt{5}}{2}$$

$$\sum_{k=1}^{n} f_k = f_{n+2} - 1,\ \sum_{k=1}^{n} f_k^2 = f_n f_{n+1}$$

$$\sum_{k=0}^{n} f_k f_{n-k} = \frac{1}{5}(n-1)f_n + \frac{2}{5} n f_{n-1}$$

$$\frac{f_{2n}}{f_n} = f_{n-1} + f_{n+1}$$

$$f_1 + 2f_2 + 3f_3 + \cdots + nf_n = nf_{n+2} - f_{n+3} + 2]$$

$$\gcd(f_m, f_n) = f_{\gcd(m,n)}$$

$$f_n^2 + (-1)^n = f_{n+1} f_{n-1}$$

$$f_{n+k} = f_n f_{k+1} + f_{n-1} f_k$$

$$f_{2n+1} = f_n^2 + f_{n+1}^2$$

$$(-1)^k f_{n-k} = f_n f_{k-1} - f_{n-1} f_k$$

$$\text{Modulo } f_n, f_{mn+r} \equiv \begin{cases} f_r, & m \bmod 4 = 0; \\ (-1)^{r+1} f_{n-r}, & m \bmod 4 = 1; \\ (-1)^n f_r, & m \bmod 4 = 2; \\ (-1)^{r+1+n} f_{n-r}, & m \bmod 4 = 3. \end{cases}$$

7.1.5 Lucas Numbers

$$L_0 = 2, L_1 = 1, L_n = L_{n-1} + L_{n-2} = (\frac{1+\sqrt{5}}{2})^n + \frac{1-\sqrt{5}}{2})^n$$

$$L(x) = \frac{2-x}{1-x-x^2}$$

7.1.6 Catlan Numbers

$$c_1 = 1, c_n = \sum_{i=0}^{n-1} c_i c_{n-1-i} = c_{n-1}\frac{4n-2}{n+1} = \frac{\binom{2n}{n}}{n+1} = \binom{2n}{n} - \binom{2n}{n-1}$$

$$c(x) = \frac{1-\sqrt{1-4x}}{2x}$$

Usage: n对括号序列; n个点满二叉树; $n*n$的方格左下到右上不过对角线方案数; 凸$n+2$边形三角形分割数; n个数的出栈方案数; $2n$个顶点连接, 线段两两不交的方案数.

7.1.7 Stirling Cycle Numbers

把n个元素集合分作k个非空环方案数.

$s(n,0) = 0, s(n,n) = 1, s(n+1,k) = s(n,k-1) - nS(n,k)$

$$s(n,k) = (-1)^{n-k} \begin{bmatrix} n \\ k \end{bmatrix}$$

$$\begin{bmatrix} n+1 \\ k \end{bmatrix} = n \begin{bmatrix} n \\ k \end{bmatrix} + \begin{bmatrix} n \\ k-1 \end{bmatrix},\ \begin{bmatrix} n+1 \\ 2 \end{bmatrix} = n! H_n$$

$$x^{\underline{n}} = \sum_k \begin{bmatrix} n \\ k \end{bmatrix} (-1)^{n-k} x^k,\ x^{\overline{n}} = \sum_k \begin{bmatrix} n \\ k \end{bmatrix} x^k$$

7.1.8 Stirling Subset Numbers

把n个元素集合分作k个非空子集方案数.

$$\begin{Bmatrix} n+1 \\ k \end{Bmatrix} = k \begin{Bmatrix} n \\ k \end{Bmatrix} + \begin{Bmatrix} n \\ k-1 \end{Bmatrix}$$

$$x^n = \sum_k \begin{Bmatrix} n \\ k \end{Bmatrix} x^{\underline{k}} = \sum_k \begin{Bmatrix} n \\ k \end{Bmatrix} (-1)^{n-k} x^{\overline{k}}$$

$$m! \begin{Bmatrix} n \\ m \end{Bmatrix} = \sum_k \binom{m}{k} k^n (-1)^{m-k}$$

For fixed k, generating functions :

$$\sum_{n=0}^{\infty} \begin{Bmatrix} n \\ k \end{Bmatrix} x^{n-k} = \prod_{r=1}^{k} \frac{1}{1-rx}$$

7.1.9 Motzkin Numbers

圆上n点间画不相交弦的方案数．选n个数$k_1, k_2, ..., k_n \in \{-1, 0, 1\}$ 保证$\sum_i^a k_i (1 \le a \le n)$非负且所有数总和为0的方案数．

$$M_{n+1} = M_n + \sum_i^{n-1} M_i M_{n-1-i} = \frac{(2n+3)M_n + 3nM_{n-1}}{n+3}$$

$$M_n = \sum_{i=0}^{\lfloor \frac{n}{2} \rfloor} \binom{n}{2k} Catlan(k)$$

$$M(X) = \frac{1 - x - \sqrt{1 - 2x - 3x^2}}{2x^2}$$

7.1.10 Eulerian Numbers

$$\left\langle {n \atop k} \right\rangle = (k+1)\left\langle {n-1 \atop k} \right\rangle + (n-k)\left\langle {n-1 \atop k-1} \right\rangle$$

$$x^n = \sum_k \left\langle {n \atop k} \right\rangle \binom{x+k}{n}$$

$$\left\langle {n \atop m} \right\rangle = \sum_{k=0}^{m} \binom{n+1}{k} (m+1-k)^n (-1)^k$$

7.1.11 Harmonic Numbers

$$\sum_{k=1}^{n} H_k = (n+1)H_n - n$$

$$\sum_{k=1}^{n} kH_k = \frac{n(n+1)}{2} H_n - \frac{n(n-1)}{4}$$

$$\sum_{k=1}^{n} \binom{k}{m} H_k = \binom{n+1}{m+1} (H_{n+1} - \frac{1}{m+1})$$

7.1.12 Pentagonal Number Theorem

$$\prod_{n=1}^{\infty} (1 - x^n) = \sum_{n=-\infty}^{\infty} (-1)^k x^{k(3k-1)/2}$$

$$p(n) = p(n-1) + p(n-2) - p(n-5) - p(n-7) + \cdots$$

$$f(n,k) = p(n) - p(n-k) - p(n-2k) + p(n-5k) + p(n-7k) - \cdots$$

7.1.13 Bell Numbers

n个元素集合划分的方案数．

$$B_n = \sum_{k=1}^{n} \left\{ {n \atop k} \right\}, \quad B_{n+1} = \sum_{k=0}^{n} \binom{n}{k} B_k$$

$$B_{p^m+n} \equiv mB_n + B_{n+1} \pmod p$$

$$B(x) = \sum_{n=0}^{\infty} \frac{B_n}{n!} x^n = e^{e^x - 1}$$

7.1.14 Bernoulli Numbers

$$B_n = 1 - \sum_{k=0}^{n-1} \binom{n}{k} \frac{B_k}{n-k+1}$$

$$G(x) = \sum_{k=0}^{\infty} \frac{B_k}{k!} x^k = \frac{1}{\sum_{k=0}^{\infty} \frac{x^k}{(k+1)!}}$$

$$\sum_{k=1}^{n} k^m = \frac{1}{m+1} \sum_{k=0}^{m} \binom{m+1}{k} B_k n^{m-k+1}$$

7.1.15 Sum of Powers

$$\sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}, \quad \sum_{i=1}^{n} i^3 = (\frac{n(n+1)}{2})^2$$

$$\sum_{i=1}^{n} i^4 = \frac{n(n+1)(2n+1)(3n^2+3n-1)}{30}$$

$$\sum_{i=1}^{n} i^5 = \frac{n^2(n+1)^2(2n^2+2n-1)}{12}$$

7.1.16 Sum of Squares

$r_k(n)$表示用k个平方数组成n的方案数．假设:

$$n = 2^{a_0} p_1^{2a_1} \cdots p_r^{2a_r} q_1 b_1 \cdots q_s b_s$$

其中$p_i \equiv 3 \mod 4$, $q_i \equiv 1 \mod 4$, 那么

$$r_2(n) = \begin{cases} 0 & \text{if any } a_i \text{ is a half-integer} \\ 4\prod_1^r (b_i + 1) & \text{if all } a_i \text{ are integers} \end{cases}$$

$r_3(n) > 0$当且仅当n不满足$4^a(8b+7)$的形式 (a, b为整数).

7.1.17 Pythagorean Triple

枚举$x^2 + y^2 = z^2$的三元组: 可令$x = m^2 - n^2, y = 2mn, z = m^2 + n^2$, 枚举$m$和$n$即可$O(n)$枚举勾股数．判断素勾股数方法: m, n至少一个为偶数并且m, n互质, 那么x, y, z就是素勾股数．

7.1.18 Tetrahedron Volume

If U, V, W, u, v, w are lengths of edges of the tetrahedron (first three form a triangle; u opposite to U and so on)

$$V = \frac{\sqrt{4u^2v^2w^2 - \sum_{cyc} u^2(v^2 + w^2 - U^2)^2 + \prod_{cyc} (v^2 + w^2 - U^2)}}{12}$$

7.1.19 杨氏矩阵 与 钩子公式

满足: 格子(i, j)没有元素, 则它右边和上边相邻格子也没有元素; 格子(i, j)有元素$a[i][j]$, 则它右边和上边相邻格子要么没有元素, 要么有元素且比$a[i][j]$大.

计数: $F_1 = 1, F_2 = 2, F_n = F_{n-1} + (n-1)F_{n-2}$, $F(x) = e^{x + \frac{x^2}{2}}$
钩子公式: 对于给定形状λ, 不同杨氏矩阵的个数为:

$$d_\lambda = \frac{n!}{\prod h_\lambda(i, j)}$$

$h_\lambda(i, j)$表示该格子右边和上边的格子数量加1.

7.1.20 重心

半径为 r , 圆心角为 θ 的扇形重心与圆心的距离为 $\frac{4r\sin(\theta/2)}{3\theta}$
半径为 r , 圆心角为 θ 的圆弧重心与圆心的距离为 $\frac{4r\sin^3(\theta/2)}{3(\theta - \sin(\theta))}$

7.1.21 常见游戏

Anti-Nim游戏　n堆石子轮流拿 谁走最后一步输 结论 先手胜当且仅当1．所有堆石子数都为1且游戏的SG值为0 (即有偶数个孤单堆-每堆只有1个石子数) 2．存在某堆石子数大于1且游戏的SG值不为0.

斐波那契博弈　有一堆物品, 两人轮流取物品, 先手最少取一个, 至多无上限, 但不能把物品取完, 之后每次取的物品数不能超过上一次取的物品数的二倍且至少为一件, 取走最后一件物品的人获胜．结论: 先手胜当且仅当物品数n不是斐波那契数.

威佐夫博弈　有两堆石子, 博弈双方每次可以取一堆石子中的任意个, 不能不取, 或者取两堆石子中的相同个．先取完者赢．结论: 求出两堆石子A和B的差值C, 如果$\left\lfloor C * \frac{\sqrt{5}+1}{2} \right\rfloor = min(A, B)$那么后手赢, 否则先手赢.

约瑟夫环　令n个人标号为$0, 1, 2, ..., n-1$, 令$f_{i,m}$表示i个人报m胜利者的编号, 则$f_{1,m} = 0, f_{i,m} = (f_{i-1,m} + m) mod\ i$.

7.1.22 错排公式

$$D_1 = 0, D_2 = 1, D_n = n!(\frac{1}{0!} - \frac{1}{1!} + \frac{1}{2!} - \frac{1}{3!} + ... + \frac{(-1)^n}{n!})$$

$$D_n = (n-1)(D_{n-1} + D_{n-2})$$

7.1.23 概率相关

对于随机变量X, 期望用$E(X)$表示, 方差用$D(X)$表示, 则$D(X) = E(X - E(X))^2 = E(X^2) - (E(X))^2, D(X+Y) = D(X) + D(Y) D(aX) = a^2 D(X)$

$$E[x] = \sum_{i=1}^{\infty} P(X \ge i)$$

7.1.24 常用泰勒展开

$$f(x) = \frac{f(x_0)}{0!} + \frac{f'(x_0)}{1!}(x - x_0) + \frac{f''(x_0)}{2!}(x - x_0)^2 + \cdots$$

$$\frac{1}{(1-x)^n} = 1 + \binom{n}{1} x + \binom{n+1}{2} x^2 + \binom{n+2}{3} x^3 + \cdots$$

$$e^x = 1 + x + \frac{x^2}{2!} + \cdots + \frac{x^i}{i!}$$

7.1.25 Others (某些近似数值公式在这里)

$$S_j = \sum_{k=1}^{n} x_k^j$$

$$h_m = \sum_{1 \le j_1 < \cdots < j_m \le n} x_{j_1} \cdots x_{j_m}, \quad H_m = \sum_{1 \le j_1 \le \cdots \le j_m \le n} x_{j_1} \cdots x_{j_m}$$

$$h_n = \frac{1}{n} \sum_{k=1}^{n} (-1)^{k+1} S_k h_{n-k}$$

$$H_n = \frac{1}{n} \sum_{k=1}^{n} S_k H_{n-k}$$

$$\sum_{k=0}^{n} kc^k = \frac{nc^{n+2} - (n+1)c^{n+1} + c}{(c-1)^2}$$

$$\sum_{i=1}^{n} = ln(n) + \Gamma, (\Gamma \approx 0.5772156649015328606065120 9)$$

$$n! = \sqrt{2\pi n}(\frac{n}{e})^n(1+\frac{1}{12n}+\frac{1}{288n^2}+O(\frac{1}{n^3}))$$

$$\max\{x_a-x_b, y_a-y_b, z_a-z_b\} - \min\{x_a-x_b, y_a-y_b, z_a-z_b\}$$
$$=\frac{1}{2}\sum_{cyc}|(x_a-y_a)-(x_b-y_b)|$$

$$(a+b)(b+c)(c+a) = \frac{(a+b+c)^3-a^3-b^3-c^3}{3}$$

$$a^3+b^3 = (a+b)(a^2-ab+b^2), a^3-b^3 = (a-b)(a^2+ab+b^2)$$

$$a^n+b^n = (a+b)(a^{n-1}-a^{n-2}b+a^{n-3}b^2-...-ab^{n-2}+b^{n-1})(n \; mod \; 2 = 1)$$

划分问题：n个$k-1$维向量最多把k维空间分为$\sum_{i=0}^{k} C_n^i$份.

7.2 Calculus Table 导数表

$(\frac{u}{v})' = \frac{u'v-uv'}{v^2}$
$(a^x)' = (\ln a)a^x$
$(\tan x)' = \sec^2 x$
$(\cot x)' = \csc^2 x$
$(\sec x)' = \tan x \sec x$
$(\csc x)' = -\cot x \csc x$
$(\arcsin x)' = \frac{1}{\sqrt{1-x^2}}$
$(\arccos x)' = -\frac{1}{\sqrt{1-x^2}}$
$(\arctan x)' = \frac{1}{1+x^2}$
$(\text{arccot}\, x)' = -\frac{1}{1+x^2}$
$(\text{arccsc}\, x)' = -\frac{1}{x\sqrt{1-x^2}}$
$(\text{arcsec}\, x)' = \frac{1}{x\sqrt{1-x^2}}$

$(\tanh x)' = \text{sech}^2 x$
$(\coth x)' = -\text{csch}^2 x$
$(\text{sech}\, x)' = -\text{sech}\, x \tanh x$
$(\text{csch}\, x)' = -\text{csch}\, x \coth x$
$(\text{arcsinh}\, x)' = \frac{1}{\sqrt{1+x^2}}$
$(\text{arccosh}\, x)' = \frac{1}{\sqrt{x^2-1}}$
$(\text{arctanh}\, x)' = \frac{1}{1-x^2}$
$(\text{arccoth}\, x)' = \frac{1}{x^2-1}$
$(\text{arccsch}\, x)' = -\frac{1}{|x|\sqrt{1+x^2}}$
$(\text{arcsech}\, x)' = -\frac{1}{x\sqrt{1-x^2}}$

7.3 Integration Table 积分表

7.3.1 $ax^2+bx+c(a>0)$

1. $\int \frac{dx}{ax^2+bx+c} = \begin{cases} \frac{2}{\sqrt{4ac-b^2}} \arctan \frac{2ax+b}{\sqrt{4ac-b^2}} + C & (b^2 < 4ac) \\ \frac{1}{\sqrt{b^2-4ac}} \ln \left| \frac{2ax+b-\sqrt{b^2-4ac}}{2ax+b+\sqrt{b^2-4ac}} \right| + C & (b^2 > 4ac) \end{cases}$
2. $\int \frac{x}{ax^2+bx+c} dx = \frac{1}{2a} \ln|ax^2+bx+c| - \frac{b}{2a} \int \frac{dx}{ax^2+bx+c}$

7.3.2 $\sqrt{\pm ax^2+bx+c}(a>0)$

1. $\int \frac{dx}{\sqrt{ax^2+bx+c}} = \frac{1}{\sqrt{a}} \ln|2ax+b+2\sqrt{a}\sqrt{ax^2+bx+c}| + C$
2. $\int \sqrt{ax^2+bx+c}dx = \frac{2ax+b}{4a}\sqrt{ax^2+bx+c} + \frac{4ac-b^2}{8\sqrt{a^3}} \ln|2ax+ b+2\sqrt{a}\sqrt{ax^2+bx+c}| + C$
3. $\int \frac{x}{\sqrt{ax^2+bx+c}}dx = \frac{1}{a}\sqrt{ax^2+bx+c} - \frac{b}{2\sqrt{a^3}} \ln|2ax+b+ 2\sqrt{a}\sqrt{ax^2+bx+c}| + C$
4. $\int \frac{dx}{\sqrt{c+bx-ax^2}} = -\frac{1}{\sqrt{a}} \arcsin \frac{2ax-b}{\sqrt{b^2+4ac}} + C$
5. $\int \sqrt{c+bx-ax^2}dx = \frac{2ax-b}{4a}\sqrt{c+bx-ax^2} + \frac{b^2+4ac}{8\sqrt{a^3}} \arcsin \frac{2ax-b}{\sqrt{b^2+4ac}} + C$
6. $\int \frac{x}{\sqrt{c+bx-ax^2}}dx = -\frac{1}{a}\sqrt{c+bx-ax^2} + \frac{b}{2\sqrt{a^3}} \arcsin \frac{2ax-b}{\sqrt{b^2+4ac}} + C$

7.3.3 $\sqrt{\pm\frac{x-a}{x-b}}$或$\sqrt{(x-a)(x-b)}$

1. $\int \frac{dx}{\sqrt{(x-a)(b-x)}} = 2\arcsin\sqrt{\frac{x-a}{b-x}} + C \; (a<b)$
2. $$\int \sqrt{(x-a)(b-x)}dx = \frac{2x-a-b}{4}\sqrt{(x-a)(b-x)} + \frac{(b-a)^2}{4} \arcsin\sqrt{\frac{x-a}{b-x}} + C, (a<b) \quad (1)$$

7.3.4 三角函数的积分

1. $\int \tan x dx = -\ln|\cos x| + C$
2. $\int \cot x dx = \ln|\sin x| + C$
3. $\int \sec x dx = \ln\left|\tan\left(\frac{\pi}{4}+\frac{x}{2}\right)\right| + C = \ln|\sec x + \tan x| + C$
4. $\int \csc x dx = \ln\left|\tan\frac{x}{2}\right| + C = \ln|\csc x - \cot x| + C$
5. $\int \sec^2 x dx = \tan x + C$
6. $\int \csc^2 x dx = -\cot x + C$
7. $\int \sec x \tan x dx = \sec x + C$
8. $\int \csc x \cot x dx = -\csc x + C$
9. $\int \sin^2 x dx = \frac{x}{2} - \frac{1}{4}\sin 2x + C$
10. $\int \cos^2 x dx = \frac{x}{2} + \frac{1}{4}\sin 2x + C$
11. $\int \sin^n x dx = -\frac{1}{n}\sin^{n-1} x \cos x + \frac{n-1}{n}\int \sin^{n-2} x dx$
12. $\int \cos^n x dx = \frac{1}{n}\cos^{n-1} x \sin x + \frac{n-1}{n}\int \cos^{n-2} x dx$
13. $\int \frac{dx}{\sin^n x} = -\frac{1}{n-1}\frac{\cos x}{\sin^{n-1} x} + \frac{n-2}{n-1}\int \frac{dx}{\sin^{n-2} x}$
14. $\int \frac{dx}{\cos^n x} = \frac{1}{n-1}\frac{\sin x}{\cos^{n-1} x} + \frac{n-2}{n-1}\int \frac{dx}{\cos^{n-2} x}$
15. $$\int \cos^m x \sin^n x dx$$
$$= \frac{1}{m+n}\cos^{m-1} x \sin^{n+1} x + \frac{m-1}{m+n}\int \cos^{m-2} x \sin^n x dx$$
$$= -\frac{1}{m+n}\cos^{m+1} x \sin^{n-1} x + \frac{n-1}{m+1}\int \cos^m x \sin^{n-2} x dx$$
16. $\int \frac{dx}{a+b\sin x} = \begin{cases} \frac{2}{\sqrt{a^2-b^2}} \arctan \frac{a\tan\frac{x}{2}+b}{\sqrt{a^2-b^2}} + C & (a^2>b^2) \\ \frac{1}{\sqrt{b^2-a^2}} \ln\left|\frac{a\tan\frac{x}{2}+b-\sqrt{b^2-a^2}}{a\tan\frac{x}{2}+b+\sqrt{b^2-a^2}}\right| + C & (a^2<b^2) \end{cases}$
17. $\int \frac{dx}{a+b\cos x} = \begin{cases} \frac{2}{a+b}\sqrt{\frac{a+b}{a-b}} \arctan\left(\sqrt{\frac{a-b}{a+b}}\tan\frac{x}{2}\right) + C & (a^2>b^2) \\ \frac{1}{a+b}\sqrt{\frac{a+b}{a-b}} \ln\left|\frac{\tan\frac{x}{2}+\sqrt{\frac{a+b}{b-a}}}{\tan\frac{x}{2}-\sqrt{\frac{a+b}{b-a}}}\right| + C & (a^2<b^2) \end{cases}$
18. $\int \frac{dx}{a^2\cos^2 x+b^2\sin^2 x} = \frac{1}{ab}\arctan\left(\frac{b}{a}\tan x\right) + C$
19. $\int \frac{dx}{a^2\cos^2 x-b^2\sin^2 x} = \frac{1}{2ab}\ln\left|\frac{b\tan x+a}{b\tan x-a}\right| + C$
20. $\int x\sin ax dx = \frac{1}{a^2}\sin ax - \frac{1}{a}x\cos ax + C$
21. $\int x^2\sin ax dx = -\frac{1}{a}x^2\cos ax + \frac{2}{a^2}x\sin ax + \frac{2}{a^3}\cos ax + C$
22. $\int x\cos ax dx = \frac{1}{a^2}\cos ax + \frac{1}{a}x\sin ax + C$
23. $\int x^2\cos ax dx = \frac{1}{a}x^2\sin ax + \frac{2}{a^2}x\cos ax - \frac{2}{a^3}\sin ax + C$

7.3.5 反三角函数的积分(其中 $a>0$)

1. $\int \arcsin\frac{x}{a}dx = x\arcsin\frac{x}{a} + \sqrt{a^2-x^2} + C$
2. $\int x\arcsin\frac{x}{a}dx = (\frac{x^2}{2}-\frac{a^2}{4})\arcsin\frac{x}{a} + \frac{x}{4}\sqrt{x^2-x^2} + C$
3. $\int x^2\arcsin\frac{x}{a}dx = \frac{x^3}{3}\arcsin\frac{x}{a} + \frac{1}{9}(x^2+2a^2)\sqrt{a^2-x^2} + C$
4. $\int \arccos\frac{x}{a}dx = x \; arccos\frac{x}{a} - \sqrt{a^2-x^2} + C$
5. $\int x\arccos\frac{x}{a}dx = (\frac{x^2}{2}-\frac{a^2}{4})\arccos\frac{x}{a} - \frac{x}{4}\sqrt{a^2-x^2} + C$
6. $\int x^2\arccos\frac{x}{a}dx = \frac{x^3}{3}\arccos\frac{x}{a} - \frac{1}{9}(x^2+2a^2)\sqrt{a^2-x^2} + C$
7. $\int \arctan\frac{x}{a}dx = x\arctan\frac{x}{a} - \frac{a}{2}\ln(a^2+x^2) + C$
8. $\int x\arctan\frac{x}{a}dx = \frac{1}{2}(a^2+x^2)\arctan\frac{x}{a} - \frac{a}{2}x + C$
9. $\int x^2\arctan\frac{x}{a}dx = \frac{x^3}{3}\arctan\frac{x}{a} - \frac{a}{6}x^2 + \frac{a^3}{6}\ln(a^2+x^2) + C$

7.3.6 指数函数的积分

1. $\int a^x dx = \frac{1}{\ln a}a^x + C$
2. $\int e^{ax} dx = \frac{1}{a}a^{ax} + C$
3. $\int xe^{ax} dx = \frac{1}{a^2}(ax-1)a^{ax} + C$
4. $\int x^n e^{ax} dx = \frac{1}{a}x^n e^{ax} - \frac{n}{a}\int x^{n-1}e^{ax}dx$
5. $\int xa^x dx = \frac{x}{\ln a}a^x - \frac{1}{(\ln a)^2}a^x + C$
6. $\int x^n a^x dx = \frac{1}{\ln a}x^n a^x - \frac{n}{\ln a}\int x^{n-1}a^x dx$
7. $\int e^{ax}\sin bx dx = \frac{1}{a^2+b^2}e^{ax}(a\sin bx - b\cos bx) + C$
8. $\int e^{ax}\cos bx dx = \frac{1}{a^2+b^2}e^{ax}(b\sin bx + a\cos bx) + C$
9. $\int e^{ax}\sin^n bx dx = \frac{1}{a^2+b^2n^2}e^{ax}\sin^{n-1} bx(a\sin bx - nb\cos bx) + \frac{n(n-1)b^2}{a^2+b^2n^2}\int e^{ax}\sin^{n-2} bx dx$
10. $\int e^{ax}\cos^n bx dx = \frac{1}{a^2+b^2n^2}e^{ax}\cos^{n-1} bx(a\cos bx + nb\sin bx) + \frac{n(n-1)b^2}{a^2+b^2n^2}\int e^{ax}\cos^{n-2} bx dx$

7.3.7 对数函数的积分

1. $\int \ln x dx = x\ln x - x + C$
2. $\int \frac{dx}{x\ln x} = \ln\big|\ln x\big| + C$
3. $\int x^n \ln x dx = \frac{1}{n+1}x^{n+1}(\ln x - \frac{1}{n+1}) + C$
4. $\int (\ln x)^n dx = x(\ln x)^n - n\int(\ln x)^{n-1}dx$
5. $\int x^m(\ln x)^n dx = \frac{1}{m+1}x^{m+1}(\ln x)^n - \frac{n}{m+1}\int x^m(\ln x)^{n-1}dx$