Contents

1. Geometry

1.1 二维几何基础操作

```
#define cp const point &
int turn (cp a, cp b, cp c) { return sgn(det(b-a, c-a)); }
vector <point> convex_hull (vector <point> a) {
 | sort (a.begin(), a.end()); // 小于号 (y, x) 字典序
 | a.erase(unique(a.begin(), a.end()), a.end());//必要时去重
 | int n = (int) a.size (), cnt = 0;
 | vector <point> ret;
 | for (int i = 0; i < n; i++) {
 |  | while (cnt > 1
 |  | && turn (ret[cnt - 2], ret[cnt - 1], a[i]) <= 0)
 |  |  | --cnt, ret.pop_back (); // 保留边界:    <  0
 |  | ++cnt, ret.push_back (a[i]); }
 | for (int i = n - 2, fixed = cnt; i >= 0; i--) {
 |  | while (cnt > fixed
 |  | && turn (ret[cnt - 2], ret[cnt - 1], a[i]) <= 0)
 |  |  | --cnt, ret.pop_back (); // 所有点共线边界会保留两次
 |  | ++cnt, ret.push_back (a[i]); }
 | if (n > 1) ret.pop_back (); // n <= 2 吗?
 | return ret; } // 小于号为 (y, x) 时边 [0, 2pi) 逆时针
```

```
vector <point> add (vector <point> a, vector <point> b) {
// size > 0, rotate(begin, min, end), 无重, 小于号 (y, x)
 | if (a.size() == 1 || b.size() == 1) {
 |  | vector <point> ret;
 |  | for (auto i : a) for (auto j : b) ret.push_back(i+j);
 |  | return ret; }
 | vector <point> x, y;
 | for (int i = 0; i < a.size(); i++)
 |  | x.push_back(a[(i + 1) % a.size()] - a[i]);
 | for (int i = 0; i < b.size(); i++)
 |  | y.push_back(b[(i + 1) % b.size()] - b[i]);
 | vector <point> ret (x.size() + y.size());
 | merge(x.begin(), x.end(), y.begin(), y.end(),
 |  |   ret.begin(), [](cp u, cp v) {
 |  | return half(u)-half(v) ? half(u) : det(u, v) > 0;});
 | point cur = a[0] + b[0];
 | for (auto &i : ret) swap(i, cur), cur = cur + i;
 | return ret; } // ret 可能共线, 但没有重点
```

```
struct point {
 | point rot(LD t) const { // 逆时针
 |  | return {x*cos(t) - y*sin(t), x*sin(t) + y*cos(t)};}
 | point rot90() const { return {-y, x}; }};
bool two_side(cp a, cp b, cl c) {
 | return turn(c.s, c.t, a) * turn(c.s, c.t, b) < 0; }
point line_inter(cl a, cl b) { // 直线交点
 | LD s1 = det(a.t - a.s, b.s - a.s);
 | LD s2 = det(a.t - a.s, b.t - a.s);
 | return (b.s * s2 - b.t * s1) / (s2 - s1); }
vector <point> cut (const vector<point> &c, line l) {
 | vector <point> ret; // 线切凸包
 | int n = (int) c.size(); if (!n) return ret;
 | for (int i = 0; i < n; i++) {
 |  | int j = (i + 1) % n;
 |  | if (turn (l.s, l.t, c[i]) >= 0) ret.push_back(c[i]);
 |  | if (two_side (c[i], c[j], l))
 |  |  | ret.push_back(line_inter(l, {c[i], c[j]})); }
 | return ret; }
bool pos (cp a,cl b){ // point_on_segment 点在线段上
 | return turn(b.s, b.t, a) == 0 // 在直线上
 | && sgn(dot(b.s - a, b.t - a)) <= 0;}
bool inter_judge(cl a,cl b) { // 线段判非严格交
```

```
 | if (pos (b.s, a) || pos (b.t, a)) return true;
 | if (pos (a.s, b) || pos (a.t, b)) return true;
 | return two_side (a.s, a.t, b)
 |     && two_side (b.s, b.t, a); }
point proj_to_line (cp a, cl b) { // 点在直线投影
 | point st = b.t - b.s;
 | return b.s + st * (dot(a - b.s, st) / dot(st, st));}
LD p2l (cp a, cl b) { // point_to_line
 | return abs(det(b.t-b.s, a-b.s)) / dis(b.s, b.t); }
LD p2s (cp a, cl b) { // point_to_segment 注意退化
 | if ( sgn(dot(b.s - a, b.t - b.s))
 |    * sgn(dot(b.t - a, b.t - b.s)) >= 0)
 |  | return min (dis (a, b.s), dis (a, b.t));
 | return p2l (a, b); }
bool point_on_ray (cp a, cl b) { // 点在射线上
 | return turn (b.s, b.t, a) == 0
 | && sgn(dot(a - b.s, b.t - b.s)) >= 0; }
bool ray_inter_judge(line a, line b) { // 射线判交
 | LD s1, s2; // can be LL
 | s1 = det(a.t - a.s, b.s - a.s);
 | s2 = det(a.t - a.s, b.t - a.s);
 | if (sgn(s1) == 0 && sgn(s2) == 0) {
 |  | return sgn(dot(a.t - a.s, b.s - a.s)) >= 0
 |  |     || sgn(dot(b.t - b.s, a.s - b.s)) >= 0; }
 | if (!sgn(s1 - s2) || sgn(s1) == sgn(s2 - s1)) return 0;
 | swap(a, b);
 | s1 = det(a.t - a.s, b.s - a.s);
 | s2 = det(a.t - a.s, b.t - a.s);
 | return sgn(s1) != sgn(s2 - s1); }
```

```
int half(cp a){return a.y > 0||(a.y == 0 && a.x > 0)?1:0;}
bool turn_left(cl a, cl b, cl c) {
  return turn(a.s, a.t, line_inter(b, c)) > 0; }
bool is_para(cl a, cl b){return!sgn(det(a.t-a.s,b.t-b.s));}
bool cmp(cl a, cl b) {
  int sign = half(a.t - a.s) - half(b.t - b.s);
  int dir = sgn(det(a.t - a.s, b.t - b.s));
  if (!dir && !sign) return turn(a.s, a.t, b.t) < 0;
  else return sign ? sign > 0 : dir > 0; }
vector <point> hpi(vector <line> h) { // 半平面交
  sort(h.begin(), h.end(), cmp);
  vector <line> q(h.size()); int l = 0, r = -1;
  for(auto &i : h) {
   while (l < r && !turn_left(i, q[r - 1], q[r])) --r;
   while (l < r && !turn_left(i, q[l], q[l + 1])) ++l;
   if (l <= r && is_para(i, q[r])) continue;
   q[++r] = i; }
  while (r - l > 1 && !turn_left(q[l], q[r - 1], q[r])) --r;
  while (r - l > 1 && !turn_left(q[r], q[l], q[l + 1])) ++l;
  if(r - l < 2) return {};
  vector <point> ret(r - l + 1);
  for(int i = l; i <= r; i++)
    ret[i - l] = line_inter(q[i], q[i == r ? l : i + 1]);
  return ret; }
// 空集会在队列里留下一个开区域；开区域会被判定为空集。
// 为了保证正确性，一定要加足够大的框，尽可能避免零面积区域。
// 实在需要零面积区域边缘，需要仔细考虑 turn_left 的实现。
```

1.2 整数半平面交

```
struct line : point {
 | LD z; // ax + by + c >= 0
 | line () {}
 | line (LD a, LD b, LD c): point(a, b), z(c) {}
 | line (cp a, cp b): point((b-a).rot90()), z(det(a, b)){}
 | LD operator () (cp a) const{return dot(a, *this) + z;}};
```

```
point line_inter (cl u, cl v) {
return point(det({u.z, u.y}, {v.z, v.y}),
             det({u.x, u.z}, {v.x, v.z}) ) / -det(u, v); }
LD dis (cl l, cp x = {0, 0}) { return l(x) / l.len(); }
bool is_para(cl x, cl y) { return !sgn(det(x, y)); }
LD det(cl a, cl b, cl c) {
 | return det(a,b)*c.z + det(b,c)*a.z + det(c,a)*b.z;}
int check(cl a, cl b, cl c) { // sgn left(a, inter(b, c))
 | return sgn(det(b, c, a)) * sgn(det(b, c)); }
bool turn_left(cl a, cl b, cl c){return check(a, b, c) >0;}
bool cmp (cl a, cl b) {
if (is_para(a, b) && dot(a, b) > 0) return dis(a) < dis(b);
return half(a) == half(b) ? sgn(det(a,b))>0 : half(b)>0;}
// 用以上函数替换 HPI 函数, 需要 half(point)
line perp(cl l) { return {l.y, -l.x, 0}; } // 垂直
line para(cl l, cp o) { // 过一点平行
    return {l.x, l.y, l.z - l(o)}; }
point proj(cp x, cl l) {return x - l * (l(x)/l.len2());}
point refl(cp x, cl l) {return x - l * (l(x)/l.len2())*2;}
bool is_perp(cl x, cl y) { return !sgn(dot(x, y)); }
LD area(cl a, cl b, cl c) { // 0 when error
 | LD d = det(a, b, c);
 | return d * d / (det(a, b) * det(b, c) * det(c, a)); }
vector <line> cut (const vector <line>& o, line l){
 | vector <line> ret; int n = (int) o.size();
 | for (int i = 0; i < n; i++) {
 |  | cl u = o[i], v = o[(i+1) % n], w = o[(i + 2) % n];
 |  | int va = check(l, u, v), vb = check(l, v, w);
 |  | if (va > 0 || vb > 0 || (va == 0 && vb == 0))
 |  |  | ret.push_back(v);
 |  | if (va >= 0 && vb < 0) ret.push_back(l);
 | } return ret; }
```

1.3 凸包询问：凸包内、切点、交点、最近点

```
int n; vector <point> a; // 可以封装成一个 struct
bool inside (cp u) { // 点在凸包内
 | int l = 1, r = n - 2;
 | while (l < r) {
 |  | int mid = (l + r + 1) / 2;
 |  | if (turn(a[0], a[mid], u) >= 0) l = mid;
 |  | else r = mid - 1; }
 | return turn (a[0], a[l], u) >= 0
 |     && turn (a[l], a[l + 1], u) >= 0
 |     && turn (a[l + 1], a[0], u) >= 0; }
int search (auto f) { // 凸包求极值, 需要 C++17
 | int l = 0, r = n - 1;
 | int d = f(a[r], a[l]) ? (swap(l, r), -1) : 1;
 | while (d * (r - l) > 1) {
 |  | int mid = (l + r) / 2;
 |  | if (f(a[mid], a[l]) && f(a[mid], a[mid - d])) l = mid;
 |  | else r = mid; } return l; }
pair<int, int> get_tan(cp u) { // 求切线
 return // 严格在凸包外; 需要边界上时, 特判 a[n-1] -> a[0]
 {search([&](cp x, cp y){return turn(u, y, x) > 0;}),
  search([&](cp x, cp y){return turn(u, x, y) > 0;})};}
point at (int i) { return a[i % n]; }
int inter (cp u, cp v, int l, int r) {
 | int sl = turn(u, v, at(l));
 | while (l + 1 < r) {
 |  | int m = (l + r) / 2;
 |  | if (sl == turn(u, v, at(m))) l = m;
 |  | else r = m; | } return l % n; }
bool get_inter(cp u, cp v, int &i, int &j) { // 求直线交点
 | int p0 = search([&](cp x, cp y){
 |  return det(v - u, x - u) < det(v - u, y - u);}),
 | p1 = search([&](cp x, cp y) {
```

```
  |  return det(v - u, x - u) > det(v - u, y - u);});
  | if (turn(u, v, a[p0]) * turn(u, v, a[p1]) < 0) {
  |  | if (p0 > p1) swap(p0, p1);
  |  | i = inter (u, v, p0, p1);
  |  | j = inter (u, v, p1, p0 + n);
  |  | return true; } else return false; }
LD near (cp u, int l, int r) {
  | if (l > r) r += n;
  | int sl = sgn (dot(u - at(l), at(l + 1) - at(l)));
  | LD ret = p2s (u, {at(l), at(l + 1)});
  | while (l + 1 < r) {
  |  | int m = (l + r) / 2;
  |  | if (sl == sgn (dot(u - at(m), at(m + 1) - at(m))))
  |  |  | l = m; else r = m; }
    return min(ret, p2s(u, {at(l), at(l + 1)})); }
LD get_near (cp u) { // 求凸包外点到凸包最近点
  | if (inside(u)) return 0;
  | auto [x, y] = get_tan(u);
  | return min(near(u, x, y), near(u, y, x)); }
```

1.4 点、线段在简单多边形内

```
bool point_in_polygon (cp u, const vector <point> & p) {
  | int n = (int) p.size (), cnt = 0;
  | for (int i = 0; i < n; ++i) {
  |  | point a = p[i], b = p[(i + 1) % n];
  |  | if (pos (u, {a, b})) return true;
  |  | int x = turn (a, u, b);
  |  | int y = sgn (a.y - u.y);
  |  | int z = sgn (b.y - u.y);
  |  | if (x > 0 && y <= 0 && z > 0) ++cnt;
  |  | if (x < 0 && z <= 0 && y > 0) --cnt; }
  | return cnt != 0; } // < 0 在逆时针多边形内; > 0 顺时针
bool in_polygon (cp u, cp v) {
  | // u, v in polygon; p contain those u, v on border
  | for (int i = 0; i < n; i++) {
  |  | int j = (i + 1) % n, k = (i + 2) % n;
  |  | cp ii = p[i], jj = p[j], kk = p[k];
  |  | if (inter_judge_strict({u, v}, {ii, jj})) return 0;
  |  | if (point_on_segment (jj, {u, v})) {
  |  |  | bool good = true, left = turn (ii, jj, kk) >= 0;
  |  |  | for (auto x : {u, v})
  |  |  |  | if (left)
  |  |  |  |  | good &=    turn(ii, jj, x) >= 0
  |  |  |  |  |         && turn(jj, kk, x) >= 0;
  |  |  |  | else
  |  |  |  |  | good &= !(   turn(jj, x, kk) > 0
  |  |  |  |  |           && turn(jj, ii, x) > 0);
  |  |  | if (!good) return 0;
  |  | } } return 1; }
LD get_far (int uid, int vid) {
  | // u -> v in polygon, check the ray u -> polygon
  | cp u = p[uid], v = p[vid];
  | LD far = 1e9;
  | for (int i = 0; i < n; i++) {
  |  | int j = (i + 1) % n, k = (i + 2) % n;
  |  | cp ii = p[i], jj = p[j], kk = p[k];
  |  | if (two_side (ii, jj, {u, v})) {
  |  |  | LD s1 = det(jj - ii, u - ii);
  |  |  | LD s2 = det(jj - ii, v - ii);
  |  |  | if (sgn(s1 - s2) && sgn(s1) != sgn(s2 - s1))
  |  |  |  | far = min(far,
  |  |  |  |           dis(u, line_inter({ii,jj},{u,v})));}
  |  | if (j != uid && point_on_ray (jj, {u, v})) {
  |  |  | bool good = turn(ii, jj, kk) <= 0;
  |  |  | for (auto x : {u - (jj - u), jj + (jj - u)})
  |  |  |  | good &= !(   turn(ii, jj, x) > 0
```

```
|  |  |  |           && turn(jj, kk, x) > 0);
|  |  | if (!good) far = min(far, dis(u, jj));
|  | } } return far; }
void work() {
    for (int i = 0; i < n; i++)
|  | for (int j = 0; j < n; j++) if (i != j) {
|  |  | if (!in_polygon(p[i], p[j])) continue;
|  |  | LD ret = get_far(i,j)+get_far(j,i)-dis(p[i],p[j]);
|  |  | ans = max(ans, ret); } }
```

1.5 $O(n^2)$ Ear Clipping 三角剖分

```
vector <array <int, 3>> tri;
void solve () {
| list <int> l;
| for (int k = 0; k < n; k++) l.push_back(k);
| auto check = [&](auto u, auto v, auto w) {
|  | if (turn(u, v, w) <= 0) return false;
|  | for (auto i : l)
|  |  | if (turn(u, v, p[i]) == 1 &&
|  |  |     turn(v, w, p[i]) == 1 &&
|  |  |     turn(w, u, p[i]) == 1) return false;
|  | return true; };
| for (auto it = l.begin(); l.size() > 3; ) {
|  | auto u = (it == l.begin() ? prev(l.end()) : prev(it));
|  | auto v = (next(it) == l.end() ? l.begin() : next(it));
|  | if (!check(p[*u], p[*it], p[*v])) it = v;
|  | else {
|  |  | tri.push_back({*u, *it, *v});
|  |  | l.erase(it);
|  |  | it = u; } }
| tri.push_back({l.front(), *next(l.begin()), l.back()}); }
```

1.6 单插入动态凸包

```
struct hull { // upper hull, left to right
set <point> a; LL tot;
hull () {tot = 0;}
LL calc(auto it) {
| auto u = it == a.begin() ? a.end() : prev(it);
| auto v = next(it);
| LL ret = 0;
| if (u != a.end()) ret += det(*u, *it);
| if (v != a.end()) ret += det(*it, *v);
| if (u != a.end() && v != a.end()) ret -= det(*u, *v);
| return ret; }
void insert (point p) {
| if (!a.size()) { a.insert (p); return; }
| auto it = a.lower_bound (p);
| bool out;
| if (it == a.begin()) out = (p < *it); // special case
| else if (it == a.end()) out = true;
| else out = turn(*prev(it), *it, p) > 0;
| if (!out) return;
| while (it != a.begin()) {
|  | auto o = prev(it);
|  | if (o == a.begin() || turn(*prev(o), *o, p) < 0) break;
|  | else erase(o); }
| while (it != a.end()) {
|  | auto o = next(it);
|  | if (o == a.end() || turn(p, *it, *o) < 0) break;
|  | else erase(it), it = o; }
| tot += calc(a.insert(p).first); }
void erase(auto it) { tot -= calc(it); a.erase(it); } };
```

1.7 圆

```
struct circle { point c; LD r;};
bool in_circle(cp a, const circle &b) {
 | return sgn(b.r - dis(b.c, a)) >= 0; }
circle make_circle(cp a, cp b) { // 以 a b 为直径的圆
 | point p = (a + b) / 2;
 | return {p, dis(a, p)}; }
circle make_circle(cp a, cp b, cp c) { // 三点共线 inf / nan
 | point bc = c - b, ca = a - c, ab = b - a;
 | point o = (b + c - bc.rot90()*dot(ca,ab)/det(ca,ab)) / 2;
 | return {o, dis(o, a)}; } // 检查上一行正负号
circle min_circle (vector <point> p) { // 最小覆盖圆
 circle ret({0, 0}, 0);
 shuffle (p.begin (), p.end (), rng);
 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[i], p[j]);
    for (int k = 0; k < j; k++) if (!in_circle(p[k], ret))
     ret = make_circle (p[i], p[j], p[k]);
  } } return ret; }
pair <point, point> line_circle_inter (cl a, circle c) {
 | LD d = p2l (c.c, a);
 | // 需要的话返回 vector <point>
 | /* if (sgn (d - R) >= 0) return {}; */
 | LD x = sqrt (sqr(c.r) - sqr(d)); // sqrt(max(0., ...))
 | return {
 |  | proj_to_line (c.c, a) + (a.s - a.t).unit() * x,
 |  | proj_to_line (c.c, a) - (a.s - a.t).unit() * x }; }
LD circle_inter_area (circle a, circle b) { // 圆面积交
 | LD 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) {
 |  | LD r = min (a.r, b.r);
 |  | return r * r * PI; }
 | LD 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 sqr(a.r)*t1 + sqr(b.r)*t2 - d*a.r*sin(t1);}
vector <point> circle_inter (circle a, circle b) { // 圆交点
 | if (a.c == b.c || sgn (dis (a.c, b.c) - a.r - b.r) > 0
 |  | || sgn (dis (a.c, b.c) - abs (a.r - b.r)) < 0)
 |  | return {};
 | point r = (b.c - a.c).unit();
 | LD d = dis (a.c, b.c);
 | LD x = ((sqr (a.r) - sqr (b.r)) / d + d) / 2;
 | LD h = sqrt (sqr (a.r) - sqr (x));
 | if (sgn (h) == 0) return {a.c + r * x};
 | return {a.c + r * x + r.rot90 () * h,
 |         a.c + r * x - r.rot90 () * h}; }
// 返回按照顺时针方向
vector <point> tangent (cp a, circle b) {
 | return circle_inter (make_circle (a, b.c), b); }
vector <line> extangent (circle a, circle b) {
 | vector <line> ret; // 未考虑两圆内切的一条外切线
 | if (sgn(dis(a.c, b.c)-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.len()).rot90();
 |  | ret.push_back({a.c + dir, b.c + dir});
 |  | ret.push_back({a.c - dir, b.c - dir});
 | } else {
 |  | point p = (b.c * a.r - a.c * b.r) / (a.r - b.r);
 |  | auto u = tangent(p, a), v = tangent(p, b);
 |  | if (u.size() == 2 && v.size() == 2) {
 |  |  | if (sgn(a.r-b.r) < 0)
```

```
 |  |  |  | swap(u[0], u[1]), swap(v[0], v[1]);
 |  |  | ret.push_back({u[0], v[0]});
 |  |  | ret.push_back({u[1], v[1]}); } }
 | return ret; }
vector <line> intangent(circle a, circle b) {
 | vector <line> ret; // 未考虑两圆外切的一条内切线
 | point p = (b.c * a.r + a.c * b.r) / (a.r + b.r);
 | vector u = tangent(p, a), v = tangent(p, b);
 | if (u.size() == 2 && v.size() == 2) {
 |  | ret.push_back({u[0], v[0]});
 |  | ret.push_back({u[1], v[1]}); } return ret; }
```

1.8 圆反演、阿波罗尼茨圆

所有关于两点 A, B 满足 $PA/PB = k$ 且不等于 1 的点 P 的轨迹是一个圆.

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. 两两相切的圆 r1, r2, r3, 求与他们都相切的圆 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. $r_4^{\pm} = \frac{r_1 r_2 r_3}{r_1 r_2 + r_1 r_3 + r_2 r_3 \pm 2\sqrt{r_1 r_2 r_3 (r_1 + r_2 + r_3)}}$

```
circle inv_c2c(point O, LD R, circle A) {
 | LD OA = dis(A.c, O);
 | LD RB = 0.5 * R * R * (1 / (OA - A.r) - 1 / (OA + A.r));
 | LD OB = OA * RB / A.r;
 | point B = O + (A.c - O) * (OB / OA);
 | return {B, RB};
}  // 点 O 在圆 A 外，求圆 A 的反演圆 B，R 是反演半径
circle inv_l2c(point O, LD R, line l) {
 | point P = proj_to_line(O, l);
 | LD d = dis(O, P);
 | LD RB = R * R / (2 * d);
 | point VB = (P - O) / d * RB;
 | return {O + VB, RB};
}  // 不过 O 点的直线反演为过 O 点的圆，R 是反演半径
line inv_c2l (point O, LD R, circle A) {
 | LD t = R * R / (2 * A.r);
 | point p = O + (A.c - O).unit() * t;
 | return {p, p + (O - p).rot90()};
}  // 过 O 点的圆反演为不过 O 点的直线，R 是反演半径
```

1.9 圆并

```
int C; circle c[MAXN]; LD area[MAXN];
struct event { // 如果需要边界而非面积，那么仔细考虑事件顺序
 point p; LD ang; int delta;
 bool operator <(const event &a){return ang < a.ang;}};
void addevent(cc a, cc b, vector<event> &e, int &cnt) {
 LD d2=dis2(a.c, b.c),dw=((a.r-b.r)*(a.r+b.r)/d2+1)/2,pw=
sqrt(max(0.,-(d2-sqr(a.r-b.r)*(d2-sqr(a.r+b.r))/sqr(2*d2));
 point d = b.c - a.c, p = d.rot(PI / 2),
```

```
  q0 = a.c + d * dw + p * pw,
  q1 = a.c + d * dw - p * pw;
 LD ang0 = atan2((q0 - a.c).y, (q0 - a.c).x),
    ang1 = atan2((q1 - a.c).y, (q1 - a.c).x);
 e.push_back({q1,ang1,1}); e.push_back({q0,ang0,-1});
 cnt += ang1 > ang0; }
bool issame(cc a, cc b) {
 return sgn(dis(a.c,b.c)) == 0 && sgn(a.r-b.r) == 0; }
bool overlap(cc a, cc b) {
 return sgn(a.r - b.r - dis(a.c, b.c)) >= 0; }
bool intersect(cc a, cc b) {
 return sgn(dis(a.c, b.c) - a.r - b.r) < 0; }
void solve() {
 fill(area, area + C + 2, 0);
 for(int i = 0; i < C; ++i) { int cnt = 1;
  vector<event> e;
  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], e, cnt);
  if(e.empty()) area[cnt] += PI * c[i].r * c[i].r;
  else {
   sort(e.begin(), e.end());
   e.push_back(e.front());
   for(int j = 0; j + 1 <(int)e.size(); ++j) {
    cnt += e[j].delta;
    area[cnt] += det(e[j].p,e[j + 1].p) / 2;
    LD ang = e[j + 1].ang - e[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 多边形与圆交

```
LD angle (cp u, cp v) {
 | return 2 * asin(dis(u.unit(), v.unit()) / 2); }
LD area(cp s, cp t, LD r) { // 2 * area
 | LD theta = angle(s, t);
 | LD dis = p2s ({0, 0}, {s, t});
 | if (sgn(dis - r) >= 0) return theta * r * r;
 | auto [u, v] = line_circle_inter({s, t}, {{0, 0}, r});
 | point lo = sgn(det(s, u)) >= 0 ? u : s;
 | point hi = sgn(det(v, t)) >= 0 ? v : t;
 | return det(lo, hi) + (theta - angle(lo, hi)) * r * r; }
LD solve(vector<point> &p, cc c) {
 | LD ret = 0;
 | for (int i = 0; i < (int) p.size (); ++i) {
 |  | auto u = p[i] - c.c;
 |  | auto v = p[(i + 1) % p.size()] - c.c;
 |  | int s = sgn(det(u, v));
 |  | if      (s > 0) ret += area (u, v, c.r);
 |  | else if (s < 0) ret -= area (v, u, c.r);
 | } return abs (ret) / 2; } //ret 在 p 逆时针时为正
```

1.11 球面基础, 经纬度球面距离

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: $S = ER^2$

```
// lontitude 经度范围: ±π, latitude 纬度范围: ±π/2
LD sphereDis(LD lon1, LD lat1, LD lon2, LD lat2, LD R) {
 | return R * acos(cos(lat1) * cos(lat2) * cos(lon1 - lon2) + sin(lat1) * sin(lat2)); }
```

1.12 圆上整点

```
vector <LL> solve(LL r) {
 | vector <LL> ret; // non-negative Y pos
 | ret.push_back(0);
 | LL l = 2 * r, s = sqrt(l);
 | for (LL d=1; d<=s; d++) if (l%d==0) {
 |  | LL lim=LL(sqrt(l/(2*d)));
 |  | for (LL a = 1; a <= lim; a++) {
 |  |  | LL b = sqrt(l/d-a*a);
 |  |  | if (a*a+b*b==l/d && __gcd(a,b)==1 && a!=b)
 |  |  |  | ret.push_back(d*a*b);
 |  | } if (d*d==l) break;
 |  | lim = sqrt(d/2);
 |  | for (LL a=1; a<=lim; a++) {
 |  |  | LL b = sqrt(d - a * a);
 |  |  | if (a*a+b*b==d && __gcd(a,b)==1 && a!=b)
 |  |  |  | ret.push_back(l/d*a*b);
 | } } return ret; }
```

1.13 三相之力

```
point incenter (cp a, cp b, cp 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 (cp a, cp b, cp 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, {p.y, q.y}), det({p.x, q.x}, s)) / d; }
point orthocenter (cp a, cp b, cp c) {
 | return a + b + c - circumcenter (a, b, c) * 2.0; }
point fermat_point (cp a, cp b, cp 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_inter ({a, b + (c - b).rot (sq3)}, {b, c + (a - c).rot (sq3)});
 | else mid = line_inter ({a, c + (b - c).rot (sq3)}, {c, b + (a - b).rot (sq3)});
 | return mid; } // minimize(|A-x|+|B-x|+|C-x|)
```

1.14 相关公式

1.14.1 Heron's Formula

$p = \frac{a+b+c}{2}, a \geq b \geq c,$

$S = \sqrt{p(p-a)(p-b)(p-c)},$

$S = \frac{1}{2}\sqrt{a^2c^2 - \left(\frac{a^2+c^2-b^2}{2}\right)^2}.$

1.14.2 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

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

1.14.3 三角形内心

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

1.14.4 三角形外心

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

1.14.5 三角形垂心

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

1.14.6 三角形偏心

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

内角的平分线和对边的两个外角平分线交点, 外切圆圆心. 剩余两点的同理.

1.14.7 三角形内接外接圆半径

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

1.14.8 Pick's Theorem 格点多边形面积

$S = I + \frac{B}{2} - 1$. I 内部点, B 边界点。

1.14.9 Euler's Formula 多面体与平面图的点、边、面

Convex polyhedron: $V - E + F = 2$.
Planar graph: $|F| = |E| - |V| + n + 1$, n : #(connected components).

1.14.10 三角公式

$\sin(a)+\sin(b)=2\sin(\frac{a+b}{2})\cos(\frac{a-b}{2})$
$\sin(a)-\sin(b)=2\cos(\frac{a+b}{2})\sin(\frac{a-b}{2})$
$\cos(a)+\cos(b)=2\cos(\frac{a+b}{2})\cos(\frac{a-b}{2})$
$\cos(a)-\cos(b)=-2\sin(\frac{a+b}{2})\sin(\frac{a-b}{2})$
$\sin(a\pm b)=\sin a\cos b\pm\cos a\sin b$
$\cos(a\pm b)=\cos a\cos b\mp\sin a\sin b$
$\tan(a\pm b)=\frac{\tan(a)\pm\tan(b)}{1\mp\tan(a)\tan(b)}$
$\tan(a)\pm\tan(b)=\frac{\sin(a\pm b)}{\cos(a)\cos(b)}$
$\sin(na)=n\cos^{n-1}a\sin a-\binom{n}{3}\cos^{n-3}a\sin^3a+\binom{n}{5}\cos^{n-5}a\sin^5a-\dots$
$\cos(na)=\cos^n a-\binom{n}{2}\cos^{n-2}a\sin^2a+\binom{n}{4}\cos^{n-4}a\sin^4a-\dots$

1.14.11 超球坐标系

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

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

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

1.14.13 立体角公式

ϕ：二面角

$$\Omega=(\phi_{ab}+\phi_{bc}+\phi_{ac})\ \text{rad}-\pi\ \text{sr}$$

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

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

1.14.14 常用体积公式

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

1.14.15 扇形与圆弧重心

扇形重心与圆心距离为 $\frac{4r\sin(\theta/2)}{3\theta}$,
圆弧重心与圆心距离为 $\frac{4r\sin^3(\theta/2)}{3(\theta-\sin(\theta))}$.

1.14.16 高维球体积

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

1.15 三维几何基础操作

```
/* 右手系逆时针绕轴旋转，(x,y,z)A = (x_new, y_new, z_new)
new[i] += old[j] * A[j][i] */
void calc(p3 n, LD cosw) {
 | LD sinw = sqrt(1 - cosw * cosw);
 | n = n.unit();
 | for (int i = 0; i < 3; i++) {
 | | int j = (i + 1) % 3, k = (j + 1) % 3;
 | | LD x = n[i], y = n[j], z = n[k];
 | | A[i][i] = (y * y + z * z) * cosw + x * x;
 | | A[i][j] = x * y * (1 - cosw) + z * sinw;
 | | A[i][k] = x * z * (1 - cosw) - y * sinw; } }
p3 cross (p3 a, p3 b) { return p3(
 a.y*b.z - a.z*b.y, a.z*b.x - a.x*b.z, a.x*b.y - a.y*b.x);}
LD mix(p3 a, p3 b, p3 c) { return dot(cross(a, b), c); }
struct l3 { p3 s, t; };
struct plane { // nor 为单位法向量，离原点距离 m
 | p3 nor; LD m;
 | plane(p3 r, p3 a) : nor(r.unit()), m(dot(nor, a)) {} };
// 除法注意除零；点到直线投影：与二维一致
p3 project_to_plane(p3 a, plane b) { // 点到平面投影
 | return a + b.nor * (b.m - dot(a, b.nor)); }
pair<p3, p3> l3_closest(l3 x, l3 y) { // 两直线最近点
 | LD a = dot(x.t - x.s, x.t - x.s);
 | LD b = dot(x.t - x.s, y.t - y.s);
 | LD e = dot(y.t - y.s, y.t - y.s);
 | LD d = a*e - b*b; p3 r = x.s - y.s;
 | LD c = dot(x.t - x.s, r), f = dot(y.t - y.s, r);
 | LD s = (b*f - c*e) / d, t = (a*f - c*b) / d;
 | return {x.s + (x.t - x.s)*s, y.s + (y.t - y.s)*t}; }
p3 intersect(plane a, l3 b) { // 直线与平面交点
 | LD t = dot(a.nor, a.nor*a.m - b.s)/dot(a.nor, b.t - b.s);
 | return b.s + (b.t - b.s) * t; }
// 平面与平面求交线
l3 intersect(plane a, plane b) {
 | p3 d = cross(a.nor, b.nor), d2 = cross(b.nor, d);
 | LD t = dot(d2, a.nor);
 | p3 s = d2 * (a.m - dot(b.nor*b.m, a.nor))/t + b.nor*b.m;
 | return {s, s + d}; }
// 三个平面求交点
p3 intersect(plane a, plane b, plane c) {
return intersect(a, intersect(b, c));
p3 c1 (a.nor.x, b.nor.x, c.nor.x);
p3 c2 (a.nor.y, b.nor.y, c.nor.y);
p3 c3 (a.nor.z, b.nor.z, c.nor.z);
p3 c4 (a.m, b.m, c.m);
return 1 / mix(c1, c2, c3) * p3(mix(c4, c2, c3), mix(c1, c4, c3), mix(c1, c2, c4)); }
```

1.16 三维凸包

```
vector <p3> p;
int mark[N][N], stp;
typedef array <int, 3> Face;
vector <Face> face;
```

```
LD volume (int a, int b, int c, int d) {
 | return mix (p[b] - p[a], p[c] - p[a], p[d] - p[a]); }
void ins(int a, int b, int c) {face.push_back({a, b, c});}
void add(int v) {
 | vector <Face> tmp; int a, b, c; stp++;
 | for (auto f : face) {
 |  | if (sgn(volume(v, f[0], f[1], f[2])) < 0) {
 |  |  | for (auto i : f) for (auto j : f)
 |  |  |  | mark[i][j] = stp; }
 |  | else {
 |  |  | tmp.push_back(f);}
 | } face = tmp;
 | for (int i = 0; i < (int) tmp.size(); i++) {
 |  | a = tmp[i][0], b = tmp[i][1], c = tmp[i][2];
 |  | if (mark[a][b] == stp) ins(b, a, v);
 |  | if (mark[b][c] == stp) ins(c, b, v);
 |  | if (mark[c][a] == stp) ins(a, c, v); } }
bool Find(int n) {
 | for (int i = 2; i < n; i++) {
 |  | p3 ndir = cross (p[0] - p[i], p[1] - p[i]);
 |  | if (ndir == p3(0,0,0)) continue;
 |  | swap(p[i], p[2]);
 |  | for (int j = i + 1; j < n; j++) {
 |  |  | if (sgn(volume(0, 1, 2, j)) != 0) {
 |  |  |  | swap(p[j], p[3]);
 |  |  |  | ins(0, 1, 2);
 |  |  |  | ins(0, 2, 1);
 |  |  |  | return 1;
 | } } } return 0; }
mt19937 rng;
bool solve() {
 | face.clear();
 | int n = (int) p.size();
 | shuffle(p.begin(), p.end(), rng);
 | if (!Find(n)) return 0;
 | for (int i = 3; i < n; i++) add(i);
 | return 1; }
```

1.17 最小覆盖球

```
vector<p3> b; Circle calc() {
 if(b.empty()) { return Circle(p3(0, 0, 0), 0);
 }else if(1 == b.size()) {return Circle(b[0], 0);
 }else if(2 == b.size()) {
  return Circle((b[0] + b[1]) / 2, (b[0] - b[1]).len() / 2);
 }else if(3 == b.size()) {
  LD r = dis(b[0], b[1]) * dis(b[1], b[2]) * dis(b[2], b[0]) / 2 / cross(b[0] - b[2], b[1] -
    ↪ b[2]).len();
  return Circle(intersect({b[1] - b[0], (b[1] + b[0]) / 2}, {b[2] - b[1], (b[2] + b[1]) / 2},
    ↪ {cross(b[1] - b[0], b[2] - b[0]), b[0]}), r);
 }else { p3 o(intersect({b[1] - b[0], (b[1] + b[0]) / 2}, {b[2] - b[0], (b[2] + b[0]) / 2}, {b[3] -
   ↪ b[0], (b[3] + b[0]) / 2})); return Circle(o, (o - b[0]).len()); } }
Circle miniBall(int n) {
 Circle res(calc());
 for(int i = 0; i < n; i++) if(!in_circle(a[i], res)) {
  b.push_back(a[i]); res = miniBall(i); b.pop_back();
  if (i) { p3 tmp = a[i]; memmove(a + 1, a, sizeof(p3) * i); a[0] = tmp; } }
 return res; }
```

2. Tree & Graph

2.1 树的重心

定义

- **定义一 (按子树大小)**: 找到一个点，删去这个点后剩下的所有子树的节点数都不超过总结点数的一半 ($\lfloor n/2 \rfloor$)，那么这个点就是树的重心。
- **定义二 (按最大子树)**: 找到一个点，其所有的子树中最大的子树节点数最少，那么这个点就是树的重心。

性质

- **数量**: 一棵树最少有一个重心，最多只有两个重心。当且仅当存在一条边，断开后形成两个大小均为 $n/2$ 的子树时，这条边连接的两个节点都是重心。
- **距离和最小**: 树中所有点到重心的距离之和是最小的。如果有两个重心，那么它们的距离和相等。
- **路径性**: 把两棵树通过一条边相连得到一棵新的树，那么新树的重心在连接原来两棵树的重心的路径上。
- **动态变化**: 在一棵树上添加或删除一个叶子，它的重心最多只移动一条边的距离。

2.2 树的直径

定义 树上的最长简单路径。(以下性质基于边权为正)

性质

- **中心性**: 如果一棵树有多条直径，那么它们必然相交于一点（这个点可能是某条边的中点）。
- **最远点**: 对于树上任意一点，距离其最远的点一定是某条直径的端点之一。
- **可合并性 (重要)**: 对于树上的两个点集 S 和 T，若 S 中最远点对是 (x, y)，T 中最远点对是 (u, v)，那么 $S \cup T$ 中的最远点对一定是 $\{x, y, u, v\}$ 这四个点中某两点组成的点对。

2.3 树链剖分

```cpp
/*
加完边记得调用 work() 初始化
*/
struct HLD {
    int n;
    std::vector<int> siz, top, dep, parent, in, out, seq;
    std::vector<std::vector<int>> adj;
    int cur;

    HLD() {}
    HLD(int n) { init(n); }
    void init(int n) {
        this->n = n;
        siz.resize(n);
        top.resize(n);
        dep.resize(n);
        parent.resize(n);
        in.resize(n);
        out.resize(n);
        seq.resize(n);
        cur = 0;
        adj.assign(n, {});
    }
    void addEdge(int u, int v) {
        adj[u].push_back(v);
        adj[v].push_back(u);
    }
    void work(int root = 0) {
```

```
        top[root] = root;
        dep[root] = 0;
        parent[root] = -1;
        dfs1(root);
        dfs2(root);
    }
    void dfs1(int u) {
        if (parent[u] != -1) {
            adj[u].erase(std::find(adj[u].begin(), adj[u].end(), parent[u]));
        }

        siz[u] = 1;
        for (auto &v : adj[u]) {
            parent[v] = u;
            dep[v] = dep[u] + 1;
            dfs1(v);
            siz[u] += siz[v];
            if (siz[v] > siz[adj[u][0]]) {
                std::swap(v, adj[u][0]);
            }
        }
    }
    void dfs2(int u) {
        in[u] = cur++;
        seq[in[u]] = u;
        for (auto v : adj[u]) {
            top[v] = v == adj[u][0] ? top[u] : v;
            dfs2(v);
        }
        out[u] = cur;
    }
    int lca(int u, int v) {
        while (top[u] != top[v]) {
            if (dep[top[u]] > dep[top[v]]) {
                u = parent[top[u]];
            } else {
                v = parent[top[v]];
            }
        }
        return dep[u] < dep[v] ? u : v;
    }

    int dist(int u, int v) { return dep[u] + dep[v] - 2 * dep[lca(u, v)]; }

    int jump(int u, int k) {
        if (dep[u] < k) {
            return -1;
        }

        int d = dep[u] - k;

        while (dep[top[u]] > d) {
            u = parent[top[u]];
        }

        return seq[in[u] - dep[u] + d];
    }

    bool isAncester(int u, int v) { return in[u] <= in[v] && in[v] < out[u]; }

    int rootedParent(int u, int v) {
        std::swap(u, v);
        if (u == v) {
            return u;
        }
        if (!isAncester(u, v)) {
            return parent[u];
```

```
        }
        auto it =
            std::upper_bound(adj[u].begin(), adj[u].end(), v,
                             [&](int x, int y) { return in[x] < in[y]; }) -
            1;
        return *it;
    }

    int rootedSize(int u, int v) {
        if (u == v) {
            return n;
        }
        if (!isAncester(v, u)) {
            return siz[v];
        }
        return n - siz[rootedParent(u, v)];
    }

    int rootedLca(int a, int b, int c) {
        return lca(a, b) ^ lca(b, c) ^ lca(c, a);
    }

    int intersection(int x, int y, int u, int v) {
        vector<int> t = {lca(x, u), lca(x, v), lca(y, u), lca(y, v)};
        sort(t.begin(), t.end());
        int l = lca(x, y), r = lca(u, v);
        if (dep[l] > dep[r]) swap(l, r);
        if (dep[t[0]] < dep[l] || dep[t[2]] < dep[r]) {
            return 0;
        }
        return 1 + dist(t[2], t[3]);
    }
};
```

2.4 prufer 与树的互相转化

```
vector<int> edges_to_prufer(const vector<pair<int, int>>& eg)  //[1,n], 定根为 n
{
    int n = eg.size() + 1, i, j, k;
    vector<int> fir(n + 1), nxt(2 * n + 1), e(2 * n + 1), rd(n + 1);
    int cnt = 0;
    for (auto [u, v] : eg) {
        e[++cnt] = v;
        nxt[cnt] = fir[u];
        fir[u] = cnt;
        ++rd[v];
        e[++cnt] = u;
        nxt[cnt] = fir[v];
        fir[v] = cnt;
        ++rd[u];
    }
    for (i = 1; i <= n; i++)
        if (rd[i] == 1) break;
    int u = i;
    vector<int> r;
    for (j = 1; j < n - 1; j++) {
        for (k = fir[u], u = rd[u] = 0; k; k = nxt[k])
            if (rd[e[k]]) {
                r.push_back(e[k]);
                if ((--rd[e[k]] == 1) && (e[k] < i)) u = e[k];
            }
        if (!u) {
            while (rd[i] != 1) ++i;
            u = i;
        }
    }
    return r;
```

```cpp
}
vector<pair<int, int>> prufer_to_edges(const vector<int>& p)  //[1,n], 定根为 n
{
    int n = p.size(), i, j, k;
    int m = n + 3;
    vector<int> cs(m);
    for (i = 0; i < n; i++) ++cs[p[i]];
    i = 0;
    while (cs[++i]);
    int u = i, v;
    vector<pair<int, int>> r;
    for (j = 0; j < n; j++) {
        cs[u] = 1e9;
        r.push_back({u, v = p[j]});
        if ((--cs[v] == 0) && (v < i)) u = v;
        if (v != u) {
            while (cs[i]) ++i;
            u = i;
        }
    }
    r.push_back({u, n + 2});
    return r;
}
```

2.5 树哈希

核心思想 将树的拓扑结构映射成一个数。

- 有根树比较好用的 hash 公式是 $f(p) = \sum_{s \in \text{son}(p)} \text{hash}(f(s)) + C$。
- 其中 hash 函数可以是随机数或者多项式。C 是常数，取 1 就行。
- 这样的 hash 公式利于换根，换根公式也容易推导：

$$g(p) = \begin{cases} f(p) & \text{p is root} \\ f(p) + \text{hash}(g(\text{fa}(p)) - \text{hash}(f(p))) & \text{others} \end{cases}$$

- 有根树的 hash 可以用根节点的 hash 值。
- 无根树的 hash 可以用所有有点作为根节点的 hash 值的异或和作为 hash 值。

复杂度分析

- 随机数 hash 使用了 map，复杂度为 $O(n \log n)$。

注意事项

- 防止被卡，可以更改 hash 函数，常数 C，取模底数（如模 998244353），此时无根树 hash 可以改为所有点 hash 值的加和。
- 使用前记得调用 calc(int root = 0)，否则会 assert。

```cpp
using ull = unsigned long long;

/**
 * @brief 树哈希类
 *
 * 用于计算有根树和无根树的哈希值。
 * 核心思想是通过 DFS 和换根 DP
 * 来为每个节点的子树以及以每个节点为根的整棵树生成一个唯一的哈希值。
```

```
 */
class tree_hash {
   private:
    // 静态成员，用于在类的所有实例中共享
    static map<ull, ull> mp;  // 哈希值映射，确保相同的子树结构得到相同的随机值
    static mt19937_64 rng;    // 64 位梅森旋转算法随机数生成器

    int n;                  // 树的节点数
    vector<vector<int>> g;  // 邻接表表示的树
    vector<ull> dp;         // dp[i] 存储以 i 为根的子树的哈希值
    vector<ull> rdp;        // rdp[i] 存储以 i 为整棵树的根时的哈希值
    ull hash_val;           // 整棵树的无根哈希值
    bool calced;            // 标记是否已执行计算

   public:
    /**
     * @brief 构造函数
     * @param ng 描述树结构的邻接表
     */
    tree_hash(const vector<vector<int>>& ng) {
        n = ng.size();
        g = ng;
        dp.resize(n, 0);
        rdp.resize(n, 0);
        hash_val = 0;
        calced = false;
    }

   private:
    /**
     * @brief 为一个子树哈希值生成一个唯一的随机映射值
     *
     * 使用 map 来记忆化，确保对于相同的输入 x，总是返回相同的随机数。
     * 这是为了保证同构的子树贡献的哈希值是相同的。
     * @param x 子树的哈希值
     * @return 映射后的唯一哈希值
     */
    static ull get(ull x) {
        if (mp.count(x)) return mp[x];
        return mp[x] = rng();
    }

    /**
     * @brief 第一次 DFS，计算每个节点的子树哈希值
     * @param p 当前节点
     * @param fa 父节点
     */
    void dfs(int p, int fa) {
        dp[p] = 1;  // 初始值，可以看作是节点自身的贡献
        for (auto s : g[p]) {
            if (s != fa) {
                dfs(s, p);
                // 将所有子节点的子树哈希值通过 get() 映射后累加
                dp[p] += get(dp[s]);
            }
        }
    }

    /**
     * @brief 第二次 DFS（换根 DP），计算以每个节点为根时整棵树的哈希值
     * @param p 当前节点
     * @param fa 父节点
     */
    void rdfs(int p, int fa) {
        if (fa == -1) {
            // 如果是根节点，其换根哈希值就是它的子树哈希值
            rdp[p] = dp[p];
```

```
        } else {
            // 否则，它的换根哈希值 = 它的子树哈希值 + 从父节点那边传来的哈希值
            // 父节点传来的值 = (父节点的换根哈希值 - 当前子树对父节点的贡献)
            ull parent_contribution = rdp[fa] - get(dp[p]);
            rdp[p] = dp[p] + get(parent_contribution);
        }

        for (auto s : g[p]) {
            if (s != fa) {
                rdfs(s, p);
            }
        }
    }

  public:
    /**
     * @brief 执行计算
     * @param root 任意指定的根节点，默认为 0
     */
    void calc(int root = 0) {
        dfs(root, -1);
        rdfs(root, -1);

        // 计算无根树哈希值，通常用所有节点的有根哈希值的异或和
        // 也可以用和，但异或可以避免顺序问题
        hash_val = 0;
        for (int i = 0; i < n; i++) {
            hash_val ^= rdp[i];
        }
        calced = true;
    }

    /**
     * @brief 获取以节点 p 为根的有根树哈希值
     * @param p 节点编号
     * @return 哈希值
     */
    ull rooted_hash(int p) {
        assert(calced && "calc() must be called before getting hash values.");
        return rdp[p];
    }

    /**
     * @brief 获取无根树的哈希值
     * @return 哈希值
     */
    ull unrooted_hash() {
        assert(calced && "calc() must be called before getting hash values.");
        return hash_val;
    }
};

// 初始化静态成员
mt19937_64 tree_hash::rng(
    chrono::steady_clock::now().time_since_epoch().count());
map<ull, ull> tree_hash::mp;
```

2.6 内向基环树

```
// 返回 p 所在连通块的一个环上点
int check(int p) {
    if (vis[p]) return p;
    vis[p] = 1;
    return check(ne[p]);
}

vector<int> cir;
```

```
int p = check(i);
int q = p;
while (1) {
    cir.push_back(q);
    if (ne[q] == p) break;
    q = ne[q];
}
```

2.7 Hopcroft-Karp $O(\sqrt{V}E)$

```
vector<int> E[N];  // 邻接表，E[u] 存储左侧顶点 u 连接的所有右侧顶点
vector<int> ml, mr;  // ml[i] 存储左侧顶点 i 匹配的右侧顶点，mr[i] 存储右侧顶点
                     // i 匹配的左侧顶点。0 表示未匹配。
vector<int> a, p;  // HK 算法内部使用的辅助数组，无需关心具体含义

/**
 * @brief 计算二分图的最大匹配 (Hopcroft-Karp 算法)
 * @param nl 左侧顶点的数量 (编号从 1 到 nl)
 * @param nr 右侧顶点的数量 (编号从 1 到 nr)
 */
void match(int nl, int nr) {
    ml.assign(nl + 1, 0);
    mr.assign(nr + 1, 0);
    while (true) {
        bool ok = false;
        a.assign(nl + 1, 0);
        p.assign(nl + 1, 0);
        static queue<int> q;
        while (!q.empty()) q.pop();  // 清空队列
        for (int i = 1; i <= nl; i++)
            if (!ml[i]) a[i] = p[i] = i, q.push(i);
        while (!q.empty()) {
            int x = q.front();
            q.pop();
            if (ml[a[x]]) continue;
            for (auto y : E[x]) {
                if (!mr[y]) {
                    for (ok = 1; y; x = p[x]) mr[y] = x, swap(ml[x], y);
                    break;
                } else if (!p[mr[y]])
                    q.push(y = mr[y]), p[y] = x, a[y] = a[x];
            }
            if (ok) break;  // 找到一条增广路后就跳出
        }
        if (!ok) break;  // 找不到更多增广路，算法结束
    }
}
```

2.8 Hungarian $O(VE/w)$

```
using B = bitset<N>;
B edge[N];  // 邻接矩阵的 bitset 形式。edge[i][j] 为 1 表示左侧顶点 i 和右侧顶点
            // j 之间有边
bool dfs(int x, B& unvis, vector<int>& match) {
    for (B z = edge[x];;) {
        z &= unvis;
        int y = z._Find_first();
        if (y == N) return 0;
        unvis.reset(y);
        if (!match[y] || dfs(match[y], unvis, match)) return match[y] = x, 1;
    }
}
// 返回 一个 vector，ret[i] 表示左侧顶点 i 匹配的右侧顶点。0 表示未匹配。
vector<int> match(int nl, int nr) {
    B unvis;
    unvis.set();
```

```
    vector<int> match(nr + 1), ret(nl + 1);
    for (int i = 1; i <= nl; ++i)
        if (dfs(i, unvis, match)) unvis.set();
    for (int i = 1; i <= nr; ++i) ret[match[i]] = i;
    return ret[0] = 0, ret;
}
```

2.9 Shuffle 一般图最大匹配 $O(VE)$

```
const int N = 1e3 + 10;
// 带随机化的、基于增广路搜索的启发式算法
// 该做法能在洛谷通过 70 % 的测试点 疑似没用

int n, m, vis[N];
int mat[N];  // 匹配数组。mat[i] = j 表示顶点 i 和顶点 j 匹配。若 mat[i] = 0，则
             // i 未匹配。
vector<int> E[N];
bool dfs(int tim, int x) {
    shuffle(E[x].begin(), E[x].end(), rng);
    vis[x] = tim;
    for (auto y : E[x]) {
        int z = mat[y];
        if (vis[z] == tim) continue;
        mat[x] = y, mat[y] = x, mat[z] = 0;
        if (!z || dfs(tim, z)) return true;
        mat[x] = 0, mat[y] = z, mat[z] = y;
    }
    return false;
}
int main() {  // 暗含二分图性质跑一次即可
    cin >> n >> m;
    while (m--) {
        int u, v;
        cin >> u >> v;
        E[u].push_back(v);
        E[v].push_back(u);
    }

    for (int _ = 0; _ < 10; _++) {
        fill(vis + 1, vis + n + 1, 0);
        for (int i = 1; i <= n; ++i)
            if (!mat[i]) dfs(i, i);
    }
    int res = 0;
    for (int i = 1; i <= n; i++) res += mat[i] > 0;
    cout << res / 2 << "\n";
    for (int i = 1; i <= n; i++) cout << mat[i] << " \n"[i == n];
    return 0;
}
```

2.10 KM 最大权匹配 $O(V^3)$

```
/*
此模板要求边权为正。
传入的邻接矩阵编号从 0 开始。
如果想求完美匹配，需要给每条边加上足够大的偏移量（ > n * 最大边权 便可）。
*/
template <class T>
struct MaxAssignment {
   public:
    T solve(int nx, int ny, std::vector<std::vector<T>> a) {
        assert(0 <= nx && nx <= ny);
        assert(int(a.size()) == nx);
        for (int i = 0; i < nx; ++i) {
            assert(int(a[i].size()) == ny);
            for (auto x : a[i]) assert(x >= 0);
```

```
        }

        auto update = [&](int x) {
            for (int y = 0; y < ny; ++y) {
                if (lx[x] + ly[y] - a[x][y] < slack[y]) {
                    slack[y] = lx[x] + ly[y] - a[x][y];
                    slackx[y] = x;
                }
            }
        };

        costs.resize(nx + 1);
        costs[0] = 0;
        lx.assign(nx, std::numeric_limits<T>::max());
        ly.assign(ny, 0);
        xy.assign(nx, -1);
        yx.assign(ny, -1);
        slackx.resize(ny);
        for (int cur = 0; cur < nx; ++cur) {
            std::queue<int> que;
            visx.assign(nx, false);
            visy.assign(ny, false);
            slack.assign(ny, std::numeric_limits<T>::max());
            p.assign(nx, -1);

            for (int x = 0; x < nx; ++x) {
                if (xy[x] == -1) {
                    que.push(x);
                    visx[x] = true;
                    update(x);
                }
            }

            int ex, ey;
            bool found = false;
            while (!found) {
                while (!que.empty() && !found) {
                    auto x = que.front();
                    que.pop();
                    for (int y = 0; y < ny; ++y) {
                        if (a[x][y] == lx[x] + ly[y] && !visy[y]) {
                            if (yx[y] == -1) {
                                ex = x;
                                ey = y;
                                found = true;
                                break;
                            }
                            que.push(yx[y]);
                            p[yx[y]] = x;
                            visy[y] = visx[yx[y]] = true;
                            update(yx[y]);
                        }
                    }
                }
                if (found) break;

                T delta = std::numeric_limits<T>::max();
                for (int y = 0; y < ny; ++y)
                    if (!visy[y]) delta = std::min(delta, slack[y]);
                for (int x = 0; x < nx; ++x)
                    if (visx[x]) lx[x] -= delta;
                for (int y = 0; y < ny; ++y) {
                    if (visy[y]) {
                        ly[y] += delta;
                    } else {
                        slack[y] -= delta;
                    }
```

```
                }
                for (int y = 0; y < ny; ++y) {
                    if (!visy[y] && slack[y] == 0) {
                        if (yx[y] == -1) {
                            ex = slackx[y];
                            ey = y;
                            found = true;
                            break;
                        }
                        que.push(yx[y]);
                        p[yx[y]] = slackx[y];
                        visy[y] = visx[yx[y]] = true;
                        update(yx[y]);
                    }
                }
            }

            costs[cur + 1] = costs[cur];
            for (int x = ex, y = ey, ty; x != -1; x = p[x], y = ty) {
                costs[cur + 1] += a[x][y];
                if (xy[x] != -1) costs[cur + 1] -= a[x][xy[x]];
                ty = xy[x];
                xy[x] = y;
                yx[y] = x;
            }
        }
        return costs[nx];
    }

    // 返回左部点的匹配点

    std::vector<int> assignment() { return xy; }

    // 返回所有顶点的顶标
    std::pair<std::vector<T>, std::vector<T>> labels() {
        return std::make_pair(lx, ly);
    }

    std::vector<T> weights() { return costs; }

   private:
    std::vector<T> lx, ly, slack, costs;
    std::vector<int> xy, yx, p, slackx;
    std::vector<bool> visx, visy;
};
```

2.11 有向图欧拉回路/路径

- **check 欧拉回路**: 所有点 `degree_in = degree_out`
- **check 欧拉路径**: 所有点 `degree_in = degree_out`, 或者存在一对点出入度相差 1, 其他点 `degree_in = degree_out`

```
/**
 * @brief 有向图欧拉路/回路求解类
 *
 * 使用 Hierholzer 算法（逐步插入回路法）寻找欧拉通路。
 * 核心思想是从一个节点开始 DFS，将经过的边删除，直到无法前进时将节点入栈。
 * 最终栈中逆序的节点序列即为一条欧拉路。
 */
class EulerGraphDir {
public:
    int n;                  // 图的节点数
    vector<vector<int>> g; // 邻接表
    vector<int> din, dout; // 节点的入度和出度
    vector<int> path;      // 存储找到的欧拉路

```

```
private:
    vector<int> cur; // 当前弧优化：记录每个节点当前访问到了哪条边

    /**
     * @brief Hierholzer 算法核心
     * @param u 当前节点
     */
    void dfs(int u) {
        // 使用引用和后置 ++ 实现当前弧优化，遍历 u 的所有出边
        for (int& i = cur[u]; i < g[u].size(); ) {
            int v = g[u][i];
            i++; // 移动到下一条边（这条边被" 删除"）
            dfs(v);
        }
        path.push_back(u); // 回溯时将节点加入路径
    }

public:
    /**
     * @brief 构造函数
     * @param n_ 节点数量
     */
    EulerGraphDir(int n_) : n(n_) {
        g.resize(n);
        din.assign(n, 0);
        dout.assign(n, 0);
        cur.assign(n, 0);
    }

    /**
     * @brief 添加一条有向边
     * @param u 起点
     * @param v 终点
     */
    void add_edge(int u, int v) {
        g[u].push_back(v);
        dout[u]++;
        din[v]++;
    }

    /**
     * @brief 检查是否存在欧拉回路
     * @return 如果所有点的入度都等于出度，则返回 true
     */
    bool has_euler_circuit() {
        for (int i = 0; i < n; i++) {
            if (din[i] != dout[i]) {
                return false;
            }
        }
        // 注意：还需要检查图的连通性，这里省略，假设图是连通的
        return true;
    }

    /**
     * @brief 检查是否存在欧拉路
     * @return 如果满足欧拉路条件，返回 true
     */
    bool has_euler_path() {
        int start_node = -1, end_node = -1;
        int odd_count = 0;
        for (int i = 0; i < n; i++) {
            if (din[i] != dout[i]) {
                odd_count++;
                if (dout[i] == din[i] + 1) {
                    if (start_node != -1) return false; // 超过一个起点
                    start_node = i;
```

```
                } else if (din[i] == dout[i] + 1) {
                    if (end_node != -1) return false; // 超过一个终点
                    end_node = i;
                } else {
                    return false; // 度数差不为 1
                }
            }
        }
        // 要么所有点入度 = 出度，要么恰好一个起点一个终点
        return odd_count == 0 || odd_count == 2;
    }

    /**
     * @brief 求解欧拉路/回路
     * @return 如果不存在则返回 false
     */
    bool solve() {
        int start_node = -1;
        int odd_out = 0, odd_in = 0;

        for (int i = 0; i < n; i++) {
            if (dout[i] > din[i]) {
                odd_out++;
                if (dout[i] - din[i] > 1) return false;
                start_node = i;
            } else if (din[i] > dout[i]) {
                odd_in++;
                if (din[i] - dout[i] > 1) return false;
            }
        }

        // 判断是否存在欧拉路或回路
        if (!((odd_out == 0 && odd_in == 0) || (odd_out == 1 && odd_in == 1))) {
            return false;
        }

        // 如果是欧拉回路，从第一个有出度的点开始
        if (start_node == -1) {
            for (int i = 0; i < n; i++) {
                if (dout[i] > 0) {
                    start_node = i;
                    break;
                }
            }
            // 如果是个空图或者只有孤立点，也算合法
            if (start_node == -1) return true;
        }

        dfs(start_node);
        reverse(path.begin(), path.end());

        // 检查是否所有边都被访问
        int edge_count = 0;
        for(int i = 0; i < n; ++i) edge_count += dout[i];
        return path.size() == edge_count + 1;
    }

    /**
     * @brief 对邻接表排序以获得字典序最小的欧拉路
     */
    void min_lexicographical() {
        for (auto& vec : g) {
            sort(vec.begin(), vec.end());
        }
    }
};
```

2.12 2-SAT, 强连通分量

```cpp
struct TwoSat {
    int n;
    vector<vector<int>> e;
    vector<int> ans;
    TwoSat(int n) : n(n), e(2 * n), ans(n) {}

    void add(int u, bool f, int v, bool g) {
        e[2 * u + !f].push_back(2 * v + g);
        e[2 * v + !g].push_back(2 * u + f);
    }

    bool work() {
        vector<int> id(2 * n, -1), dfn(2 * n, -1), low(2 * n, -1);
        vector<int> stk;
        int tot = 0, cnt = 0;
        auto tarjan = [&](auto self, int u) -> void {
            stk.push_back(u);
            dfn[u] = low[u] = tot++;
            for (auto v : e[u]) {
                if (dfn[v] == -1) {
                    self(self, v);
                    low[u] = min(low[u], low[v]);
                } else if (id[v] == -1) {
                    low[u] = min(low[u], dfn[v]);
                }
            }

            if (dfn[u] == low[u]) {
                int v;
                do {
                    v = stk.back();
                    stk.pop_back();
                    id[v] = cnt;
                } while (v != u);
                cnt++;
            }
        };

        // 改为反向遍历就是字典序最大。
        for (int i = 0; i < 2 * n; i++) {
            if (dfn[i] == -1) {
                tarjan(tarjan, i);
            }
        }
        // 输出字典序最小的可行解。
        for (int i = 0; i < n; i++) {
            if (id[2 * i] == id[2 * i + 1]) return 0;
            ans[i] = id[2 * i] > id[2 * i + 1];
        }
        // 如果需要判断某个变量的取值是否唯一，需要判断由 0 能否到达 1 之类的,o(n^2)
        return 1;
    }
}
```

2.13 Bitset Kosaraju

```cpp
bitset<N> e[N], re[N], vis;

// e 是原图的邻接矩阵，re 是反图的邻接矩阵
// e[i][j] = 1 表示存在一条从 i 到 j 的有向边
// re[j][i] = 1 同样表示 i 到 j 的边

vector<int> sta;
void dfs0(int x, bitset<N> e[]) {
    vis.reset(x);
```

```
    while (true) {
        int go = (e[x] & vis)._Find_first();
        if (go == N) break;
        dfs0(go, e);
    }
    sta.push_back(x);
}
// 返回一个二维向量，每个内层 vector 包含一个强连通分量的所有顶点
vector<vector<int>> solve() {  // re 需要连好反向边
    vis.set();
    for (int i = 1; i <= n; ++i)
        if (vis.test(i)) dfs0(i, e);
    vis.set();
    auto s = sta;
    vector<vector<int>> ret;
    for (int i = n - 1; i >= 0; --i)
        if (vis.test(s[i])) {
            sta.clear(), dfs0(s[i], re), ret.push_back(sta);
        }
    return ret;
}
```

2.14 边双连通分量

```
struct EBCC {
    int n;
    std::vector<std::vector<int>> adj;
    std::vector<int> stk;
    std::vector<int> dfn, low, bel;
    int cur, cnt;

    EBCC() {}
    EBCC(int n) { init(n); }

    void init(int n) {
        this->n = n;
        adj.assign(n, {});
        dfn.assign(n, -1);
        low.resize(n);
        bel.assign(n, -1);
        stk.clear();
        cur = cnt = 0;
    }

    void addEdge(int u, int v) {
        adj[u].push_back(v);
        adj[v].push_back(u);
    }

    void dfs(int x, int p) {
        dfn[x] = low[x] = cur++;
        stk.push_back(x);

        for (auto y : adj[x]) {
            if (y == p) {
                continue;
            }
            if (dfn[y] == -1) {
                dfs(y, x);
                low[x] = std::min(low[x], low[y]);
            } else if (bel[y] == -1 && dfn[y] < dfn[x]) {
                low[x] = std::min(low[x], dfn[y]);
            }
        }

        if (dfn[x] == low[x]) {
            int y;
```

```
            do {
                y = stk.back();
                bel[y] = cnt;
                stk.pop_back();
            } while (y != x);
            cnt++;
        }
    }

    std::vector<int> work() {
        dfs(0, -1);
        return bel;
    }

    struct Graph {
        int n;  // 缩点后新图的节点数量
        std::vector<std::pair<int, int>>
            edges;              // 缩点后新图的边集，可自行修改为邻接表
        std::vector<int> siz;   // 包含了多少个原始图的节点
        std::vector<int> cnte;  // 包含了多少条原始图的边
    };
    Graph compress() {
        Graph g;
        g.n = cnt;
        g.siz.resize(cnt);
        g.cnte.resize(cnt);
        for (int i = 0; i < n; i++) {
            g.siz[bel[i]]++;
            for (auto j : adj[i]) {
                if (bel[i] < bel[j]) {
                    // 因为添加的一定是割边，所以不存在重复。
                    g.edges.emplace_back(bel[i], bel[j]);
                } else if (i < j) {
                    g.cnte[bel[i]]++;
                }
            }
        }
        return g;
    }
};
```

2.15 点双连通分量

不可以传入自环，否则孤立点不会被统计，有时候要考虑特判孤立点。

```
/*
用于求解无向图的点双连通分量（V-BCCs），并能构建对应的圆方树。
圆方树是一个新图，其中包含两种类型的节点：
1. 方点：代表原始图中的一个点双连通分量。
2. 圆点：代表原始图中的一个割点。
树中的边连接一个圆点和一个方点，当且仅当该割点属于该点双连通分量。
 */
struct VBCC {
    int n;  // 原始图的节点数
    vector<vector<int>> adj;
    vector<vector<int>> col;  // 存储每个点双连通分量包含的节点
    vector<int> dfn, low;
    vector<int> stk;
    int cur, cnt;     // 时间戳计数器和点双连通分量计数器
    vector<int> cut;  // 标记每个节点是否为割点
    vector<int> bel;  // 映射：原始图节点 -> 新图节点编号

    VBCC(int n) { init(n); }

    void init(int n) {
        this->n = n;
        adj.assign(n, {});
```

```
        dfn.assign(n, -1);
        low.resize(n);
        col.assign(n, {});
        cut.assign(n, 0);
        bel.assign(n, -1);
        stk.clear();
        cnt = cur = 0;
    }

    void addEdge(int u, int v) {
        adj[u].push_back(v);
        adj[v].push_back(u);
    }

    void tarjan(int x, int root) {
        low[x] = dfn[x] = cur++;
        stk.push_back(x);
        if (x == root && !adj[x].size()) {
            col[cnt++].push_back(x);
            return;
        }

        int flg = 0;
        for (auto y : adj[x]) {
            if (dfn[y] == -1) {
                tarjan(y, root);

                low[x] = min(low[x], low[y]);

                if (dfn[x] <= low[y]) {
                    flg++;
                    if (x != root || flg > 1) {
                        cut[x] = 1;
                    }
                    int pre = 0;
                    do {
                        pre = stk.back();
                        col[cnt].push_back(pre);
                        stk.pop_back();
                    } while (pre != y);
                    col[cnt++].push_back(x);
                }
            } else {
                low[x] = min(low[x], dfn[y]);
            }
        }
    }

    void work() {
        for (int i = 0; i < n; i++) {
            if (dfn[i] == -1) {
                tarjan(i, i);
            }
        }
    }

    // 新图（圆方树）的结构体定义
    struct Graph {
        int n;                          // 新图的节点数量
        vector<pair<int, int>> edges;  // 新图的边集
    };

    Graph compress() {
        // 节点编号规则：
        // 0 到 cnt-1 是 " 方点", 代表 V-BCC
        // cnt 开始是 " 圆点", 代表割点
        int m = cnt;
```

```
        bel.assign(n, -1);

        // 为所有割点分配新编号
        for (int i = 0; i < n; i++) {
            if (cut[i]) {
                bel[i] = m++;
            }
        }

        Graph g;
        g.n = m;

        // 构建新图的边
        for (int i = 0; i < cnt; i++) {
            for (auto u : col[i]) {
                if (cut[u]) {
                    g.edges.emplace_back(i, bel[u]);
                }
            }
        }

        return g;
    }
};
```

2.16 Dominator Tree 支配树

```
vector<int> G[MAXN], R[MAXN], son[MAXN];
int ufs[MAXN]; // R 是反图, son 存的是 sdom 树上的儿子
int idom[MAXN], sdom[MAXN], anc[MAXN];
// anc: sdom 的 dfn 最小的祖先
int pr[MAXN], dfn[MAXN], id[MAXN], stamp;
int findufs(int x) { if (ufs[x] == x) return x;
 | int t = ufs[x]; ufs[x] = findufs(ufs[x]);
 | if (dfn[sdom[anc[x]]] > dfn[sdom[anc[t]]])
 |  | anc[x] = anc[t];
 | return ufs[x]; }
void dfs(int x) {
 | dfn[x] = ++stamp; id[stamp] = x; sdom[x] = x;
 | for (int y : G[x]) if (!dfn[y]) { pr[y] = x; dfs(y); } }
void get_dominator(int n) {
 | for (int i = 1; i <= n; i++) ufs[i] = anc[i] = i;
 | dfs(1);
 | for (int i = n; i > 1; i--) { int x = id[i];
 |  | for (int y : R[x]) if (dfn[y]) { findufs(y);
 |  |  | if (dfn[sdom[x]] > dfn[sdom[anc[y]]])
 |  |  |  | sdom[x] = sdom[anc[y]]; }
 |  | son[sdom[x]].push_back(x); ufs[x] = pr[x];
 |  | for (int u : son[pr[x]]) { findufs(u);
 |  |  | idom[u] = (sdom[u] == sdom[anc[u]] ?
 |  |  |  | pr[x] : anc[u]); }
 |  | son[pr[x]].clear(); }
 | for (int i = 2; i <= n; i++) { int x = id[i];
 |  | if (idom[x] != sdom[x]) idom[x] = idom[idom[x]];
 |  | son[idom[x]].push_back(x); } }
```

2.17 Dinic 最大流

复杂度证明思路 假设 dist 为残量网络上的距离. Dinic 一轮增广会找到一个极大的长度为 $\mathrm{dist}(s,t)$ 的增广路集合 blocking flow, 增广后 $\mathrm{dist}(s,t)$ 将会增大. 因此只有 $O(V)$ 轮; 如果一轮增广是 $O(VE)$ 的, 总复杂度是 $O(V^2E)$.

单位流量网络 在 0-1 流量图上 Dinic 有更好的性质.

- 复杂度为 $O(\min\{V^{2/3}, E^{1/2}\}E)$.
- $\mathrm{dist}(s,t) = d$, 残量网络上至多还存在 E/d 的流.
- 每个点只有一个入/出度时复杂度 $O(V^{1/2}E)$, 例如 Hopcroft-Karp.

```cpp
const ll INF = 1e9;
int nn, mm, s, t, cnt = 1, h[N], ans;
struct node {
    int v, nex;
    ll w;
} e[M];
void add(int u, int v, ll w) {
    e[++cnt] = {v, h[u], w};
    h[u] = cnt;
}
int d[N], now[N];
int bfs() {
    for (int i = 1; i <= nn; i++) d[i] = INF;
    queue<int> q;
    d[s] = 0;
    now[s] = h[s];
    q.push(s);
    while (q.size()) {
        int p = q.front();
        q.pop();
        for (int i = h[p]; i; i = e[i].nex) {
            int v = e[i].v;
            if (e[i].w > 0 && d[v] == INF) {
                d[v] = d[p] + 1;
                now[v] = h[v];
                q.push(v);
                if (v == t) return 1;
            }
        }
    }
    return 0;
}
ll dfs(int p, ll sum) {
    if (p == t) return sum;
    ll k = 0, flow = 0;
    for (int i = now[p]; i && sum; i = e[i].nex) {
        now[p] = i;
        ll v = e[i].v, w = e[i].w;
        if (w > 0 && d[v] == d[p] + 1) {
            k = dfs(v, min(sum, (ll)w));
            if (k == 0) d[v] = INF;
            e[i].w -= k;
            e[i ^ 1].w += k;
            sum -= k;
            flow += k;
        }
    }
    return flow;
}
void solve() {
    for (int i = 1; i <= nn; i++) h[i] = 0;
    ans = 0;
    cnt = 1;
    // add(u,v.w),add(v,u,0);
    while (bfs()) ans += dfs(s, INF);
}
```

```cpp
template <class T>
struct MaxFlow {
    struct _Edge {
        int to;
        T cap;
        _Edge(int to, T cap) : to(to), cap(cap) {}
    };

    int n;
```

```
    std::vector<_Edge> e;
    std::vector<std::vector<int>> g;
    std::vector<int> cur, h;

    MaxFlow() {}
    MaxFlow(int n) { init(n); }

    void init(int n) {
        this->n = n;
        e.clear();
        g.assign(n, {});
        cur.resize(n);
        h.resize(n);
    }

    bool bfs(int s, int t) {
        h.assign(n, -1);
        std::queue<int> que;
        h[s] = 0;
        que.push(s);
        while (!que.empty()) {
            const int u = que.front();
            que.pop();
            for (int i : g[u]) {
                auto [v, c] = e[i];
                if (c > 0 && h[v] == -1) {
                    h[v] = h[u] + 1;
                    if (v == t) {
                        return true;
                    }
                    que.push(v);
                }
            }
        }
        return false;
    }

    T dfs(int u, int t, T f) {
        if (u == t) {
            return f;
        }
        auto r = f;
        for (int &i = cur[u]; i < int(g[u].size()); ++i) {
            const int j = g[u][i];
            auto [v, c] = e[j];
            if (c > 0 && h[v] == h[u] + 1) {
                auto a = dfs(v, t, std::min(r, c));
                e[j].cap -= a;
                e[j ^ 1].cap += a;
                r -= a;
                if (r == 0) {
                    return f;
                }
            }
        }
        return f - r;
    }
    void addEdge(int u, int v, T c) {
        g[u].push_back(e.size());
        e.emplace_back(v, c);
        g[v].push_back(e.size());
        e.emplace_back(u, 0);
    }
    T flow(int s, int t) {
        T ans = 0;
        while (bfs(s, t)) {
            cur.assign(n, 0);
```

```
            ans += dfs(s, t, std::numeric_limits<T>::max());
        }
        return ans;
    }

    std::vector<bool> minCut() {
        std::vector<bool> c(n);
        for (int i = 0; i < n; i++) {
            c[i] = (h[i] != -1);
        }
        return c;
    }

    struct Edge {
        int from;
        int to;
        T cap;
        T flow;
    };
    std::vector<Edge> edges() {
        std::vector<Edge> a;
        for (int i = 0; i < e.size(); i += 2) {
            Edge x;
            x.from = e[i + 1].to;
            x.to = e[i].to;
            x.cap = e[i].cap + e[i + 1].cap;
            x.flow = e[i + 1].cap;
            a.push_back(x);
        }
        return a;
    }

    // 构建残留网络
    std::vector<vector<int>> build() {
        std::vector<std::vector<int>> g(n);
        // 在这个实现中，e[i] 和 e[i^1] 是一对相反的边。
        // e[i] 是一条从 e[i^1].to 到 e[i].to 的边。
        for (size_t i = 0; i < e.size(); i++) {
            // 如果一条边的剩余容量大于 0，那么它就存在于残留网络中
            if (e[i].cap > 0) {
                // 找到这条边的起点。
                // e[i] 的反向边是 e[i^1]，反向边的终点就是 e[i] 的起点。
                int from = e[i ^ 1].to;
                // 边的终点
                int to = e[i].to;
                g[from].push_back(to);
            }
        }
        // 返回构建好的残留网络邻接表
        return g;
    }
};
```

2.18 原始对偶费用流

```
bool bfs() {
    for (int i = S; i <= T; i++) cur[i] = head[i];  // S-T?
    for (int i = S; i <= T; i++) dep[i] = 0;        // S-T?
    dep[S] = 1;
    queue<int> q;
    q.push(S);
    while (!q.empty()) {
        int x = q.front();
        q.pop();
        for (int i = head[x]; i; i = e[i].nxt) {
            auto [nxt, v, w, f] = e[i];
            if (f && h[v] == h[x] + w && !dep[v]) {
```

```
                dep[v] = dep[x] + 1, q.push(v);
            }
        }
    }
    return !!dep[T];
}
int dfs(int x, int flow) {
    if (x == T) return flow;
    int used = 0, ret;
    for (int &i = cur[x]; i; i = e[i].nxt) {
        auto [nxt, v, w, f] = e[o];
        if (dep[v] == dep[x] + 1 && h[v] == h[x] + w && f &&
            (ret = dfs(d, min(flow - used, f)))) {
            e[i].f -= ret;
            e[i ^ 1].f += f;
            used += ret;
            if (flow == used) break;
        }
    }
    return used;
}
typedef pair<value, int> pii;  // Unusual!
pii solve() {                  // return {cost, flow}
    value cost = 0;
    int flow = 0;
    for (int i = S; i <= T; i++) h[i] = 0;  // S-T?
    for (bool first = true;;) {
        priority_queue<pii, vector<pii>, greater<pii>> q;
        for (int i = S; i <= T; i++) dis[i] = INF_value;  // S-T?
        dis[S] = 0;
        if (first) {
            // TODO: SSSP, Bellman-Ford / DP on DAG
            first = false;
        } else {
            q.push({0, S});
            while (!q.empty()) {
                auto [d, x] = q.top();
                q.pop();
                if (dis[x] != d) continue;
                for (int o = head[x]; o; o = e[o].nxt) {
                    value w = d + e[o].w + h[x] - h[e[o].v];
                    if (e[o].f > 0 && dis[e[o].v] > w) {
                        dis[e[o].v] = w;
                        q.push({w, e[o].v});
                    }
                }
            }
        }
        if (dis[T] >= INF_value) break;
        // 所有点必须可达，否则加 h[i] += min(dis[i], dis[T])
        for (int i = S; i <= T; i++) h[i] += dis[i];  // S-T?
        int f = 0;
        while (bfs()) f += dfs(S, INF_int);
        cost += f * h[T];
        flow += f;
    }
    return {cost, flow};
}
```

```
template <class T>
struct MinCostFlow {
    struct _Edge {
        int to;
        T cap;
        T cost;
        _Edge(int to_, T cap_, T cost_) : to(to_), cap(cap_), cost(cost_) {}
    };
```

```
    int n;
    std::vector<_Edge> e;
    std::vector<std::vector<int>> g;
    std::vector<T> h, dis;
    std::vector<int> pre;
    bool dijkstra(int s, int t) {
        dis.assign(n, std::numeric_limits<T>::max());
        pre.assign(n, -1);
        std::priority_queue<std::pair<T, int>, std::vector<std::pair<T, int>>,
                            std::greater<std::pair<T, int>>>
            que;
        dis[s] = 0;
        que.emplace(0, s);
        while (!que.empty()) {
            T d = que.top().first;
            int u = que.top().second;
            que.pop();
            if (dis[u] != d) {
                continue;
            }
            for (int i : g[u]) {
                int v = e[i].to;
                T cap = e[i].cap;
                T cost = e[i].cost;
                if (cap > 0 && dis[v] > d + h[u] - h[v] + cost) {
                    dis[v] = d + h[u] - h[v] + cost;
                    pre[v] = i;
                    que.emplace(dis[v], v);
                }
            }
        }
        return dis[t] != std::numeric_limits<T>::max();
    }
    MinCostFlow() {}
    MinCostFlow(int n_) { init(n_); }
    void init(int n_) {
        n = n_;
        e.clear();
        g.assign(n, {});
    }
    void addEdge(int u, int v, T cap, T cost) {
        g[u].push_back(e.size());
        e.emplace_back(v, cap, cost);
        g[v].push_back(e.size());
        e.emplace_back(u, 0, -cost);
    }
    std::pair<T, T> flow(int s, int t) {
        T flow = 0;
        T cost = 0;
        h.assign(n, 0);
        while (dijkstra(s, t)) {
            for (int i = 0; i < n; ++i) {
                h[i] += dis[i];
            }
            T aug = std::numeric_limits<int>::max();
            for (int i = t; i != s; i = e[pre[i] ^ 1].to) {
                aug = std::min(aug, e[pre[i]].cap);
            }
            for (int i = t; i != s; i = e[pre[i] ^ 1].to) {
                e[pre[i]].cap -= aug;
                e[pre[i] ^ 1].cap += aug;
            }
            flow += aug;
            cost += aug * h[t];
        }
        return std::make_pair(flow, cost);
    }
```

```
    struct Edge {
        int from;
        int to;
        T cap;
        T cost;
        T flow;
    };
    std::vector<Edge> edges() {
        std::vector<Edge> a;
        for (int i = 0; i < e.size(); i += 2) {
            Edge x;
            x.from = e[i + 1].to;
            x.to = e[i].to;
            x.cap = e[i].cap + e[i + 1].cap;
            x.cost = e[i].cost;
            x.flow = e[i + 1].cap;
            a.push_back(x);
        }
        return a;
    }
};
```

2.19 虚树

```
int dfn[N];
int h[N], m, a[N << 1], len;  // 存储关键点

bool cmp(int x, int y) {
    return dfn[x] < dfn[y];  // 按照 dfs 序排序
}

void build() {
    sort(h + 1, h + m + 1, cmp);  // 把关键点按照 dfs 序排序
    for (int i = 1; i < m; ++i) {
        a[++len] = h[i];
        a[++len] = lca(h[i], h[i + 1]);  // 插入 lca
    }
    a[++len] = h[m];
    sort(a + 1, a + len + 1, cmp);  // 把所有虚树上的点按照 dfs 序排序
    len = unique(a + 1, a + len + 1) - a - 1;  // 去重
    for (int i = 1, lc; i < len; ++i) {
        lc = lca(a[i], a[i + 1]);
        conn(lc, a[i + 1]);  // 连边 lc -> a[i+1]
    }
}
```

2.20 网络流总结

最小割集, 最小割必须边以及可行边

最小割集 从 S 出发, 在残余网络中 BFS 所有权值非 0 的边 (包括反向边), 得到点集 $\{S\}$, 另一集为 $\{V\}-\{S\}$.

最小割集必须点 残余网络中 S 直接连向的点必在 S 的割集中, 直接连向 T 的点必在 T 的割集中; 若这些点的并集为全集, 则最小割方案唯一.

最小割可行边 在残余网络中求强联通分量, 将强联通分量缩点后, 剩余的边即为最小割可行边, 同时这些边也必然满流.

最小割必须边 在残余网络中求强联通分量, 若 S 出发可到 u, T 出发可到 v, 等价于 $\mathrm{scc}_S = \mathrm{scc}_u$ 且 $\mathrm{scc}_T = \mathrm{scc}_v$, 则该边为必须边.

常见问题

最大权闭合子图 点权, 限制条件形如: 选择 A 则必须选择 B, 选择 B 则必须选择 C, D. 建图方式: B 向 A 连边, CD 向 B 连边. 求解: S 向正权点连边, 负权点向 T 连边, 其余边容量 ∞, 求最小割, 答案为 S 所在最小割集.

二次布尔型 (文理分科) n 个点分为两类, i 号点有 l_i 或 r_i 的代价, i, j 同属一侧分别获得 l_{ij} 或 r_{ij} 的代价, 问最小代价. $L \to i : (l_i + 1/2 \sum_j l_{ij}), i \to R : (r_i + 1/2 \sum_j r_{ij}), i \leftrightarrow j : 1/2(l_{ij} + r_{ij})$. 实现时边权乘 2 为整数, 求解后答案除 2

为整数. 图拆点可以看作二分图.

如果是二元限制是分类不同时有一个代价 d_{ij}，建图可以简化为 $L \to i : l_i, i \to R : r_i, i \leftrightarrow j : d_{ij}$. 经典例子: xor 最小值, 按位拆开建图.

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 $\deg_i = \text{in}_i - \text{out}_i$ 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.

二物流 水源 S_1, 水汇 T_1, 油源 S_2, 油汇 T_2, 每根管道流量共用. 求流量和最大. 建超级源 SS_1 汇 TT_1, 连边 $SS_1 \to S_1$,$SS_1 \to S_2$,$T_1 \to TT_1$,$T_2 \to TT_1$, 设最大流为 x_1. 建超级源 SS_2 汇 TT_2, 连边 $SS_2 \to S_1$, $SS_2 \to T_2$,$T_1 \to TT_2$, $S_2 \to TT_2$, 设最大流为 x_2. 则最大流中水流量 $\frac{x_1+x_2}{2}$, 油流量 $\frac{x_1-x_2}{2}$.

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 == $du_i > 0$) 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.

整数线性规划转费用流

首先将约束关系转化为所有变量下界为 0, 上界没有要求, 并满足一些等式, 每个变量在均在等式左边且出现恰好两次, 系数为 +1 和 −1, 优化目标为 $\max \sum v_i x_i$ 的形式. 将等式看做点, 等式 i 右边的值 b_i 若为正, 则 S 向 i 连边 $(b_i, 0)$, 否则 i 向 T 连边 $(-b_i, 0)$. 将变量看做边, 记变量 x_i 的上界为 m_i(无上界则 $m_i = inf$), 将 x_i 系数为 +1 的那个等式 u 向系数为 −1 的等式 v 连边 (m_i, v_i).

2.21 Gomory-Hu 无向图最小割树 $O(V^3E)$

每次随便找两个点 s, t 求在**原图**的最小割, 在最小割树上连 (s, t, w_{cut}), 递归对由割集隔开的部分继续做. 在得到的树上, 两点最小割即为树上瓶颈路. 实现时, 由于是随意找点, 可以写为分治的形式.

2.22 Stoer-Wagner 无向图最小割 $O(VE + V^2 \log V)$

```
const int N = 601;
int f[N], siz[N], G[N][N];
int getf(int x) {return f[x] == x ? x : f[x] = getf(f[x]);}
int dis[N], vis[N], bin[N];
int n, m;
int contract(int &s, int &t) {  // Find s,t
  | memset(vis, 0, sizeof(vis));
  | memset(dis, 0, sizeof(dis));
  | int i, j, k, mincut, maxc;
  | for (i = 1; i <= n; i++) {
  |   | k = -1; maxc = -1;
```

```
  |  | for (j = 1; j <= n; j++)
  |  |  | if (!bin[j] && !vis[j] && dis[j] > maxc) {
  |  |  |  | k = j;
  |  |  |  | maxc = dis[j]; }
  |  | if (k == -1) return mincut;
  |  | s = t; t = k; mincut = maxc; vis[k] = true;
  |  | for (j = 1; j <= n; j++)
  |  |  | if (!bin[j] && !vis[j]) dis[j] += G[k][j];
  | } return mincut; }
const int inf = 0x3f3f3f3f;
int solve() {
  | int mincut, i, j, s, t, ans;
  | for (mincut = inf, i = 1; i < n; i++) {
  |  | ans = contract(s, t);
  |  | bin[t] = true;
  |  | if (mincut > ans) mincut = ans;
  |  | if (mincut == 0) return 0;
  |  | for (j = 1; j <= n; j++)
  |  |  | if (!bin[j]) G[s][j] = (G[j][s] += G[j][t]);
  | } return mincut; }
int main() {
  | cin >> n >> m;
  | for (int i = 1; i <= n; ++i) f[i] = i, siz[i] = 1;
  | for (int i = 1, u, v, w; i <= m; ++i) {
  |  | cin >> u >> v >> w;
  |  | int fu = getf(u), fv = getf(v);
  |  | if (fu != fv) {
  |  |  | if (siz[fu] > siz[fv]) swap(fu, fv);
  |  |  | f[fu] = fv, siz[fv] += siz[fu]; }
  |  | G[u][v] += w, G[v][u] += w; }
  | cout << (siz[getf(1)] != n ? 0 : solve()); }
```

2.23 弦图

弦图的定义 连接环中不相邻的两个点的边称为弦. 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.

单纯点 一个点称为单纯点当 $\{v\} \cup A(v)$ 的导出子图为一个团. 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 $v_i, \cdots, v_n$ 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 从 n 到 1 的顺序依次给点标号. 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 最大的未标号的点进行标号. 用桶维护优先队列可以做到 $O(n+m)$.

弦图的判定 判定最大势算法输出是否合法即可. 如果依次判断是否构成团, 时间复杂度为 $O(nm)$. 考虑优化, 设 $v_{i+1}, \cdots, v_n$ 中所有与 v_i 相邻的点依次为 $N(v_i) = \{v_{j1}, \cdots, v_{jk}\}$. 只需判断 v_{j1} 是否与 $v_{j2}, \cdots, v_{jk}$ 相邻即可. 时间复杂度 $O(n+m)$.

弦图的染色 完美消除序列**从后往前**染色, 染上出度的 mex.

最大独立集 完美消除序列**从前往后**能选就选.

团数 最大团的点数. 一般图团数 ≤ 色数, 弦图团数 = 色数.

极大团 弦图的极大团一定为 $\{x\} \cup N(x)$.

最小团覆盖 用最少的团覆盖所有的点. 设最大独立集为 $\{p_1, \ldots, p_t\}$, 则 $\{p_1 \cup N(p_1), \ldots, p_t \cup N(p_t)\}$ 为最小团覆盖.

弦图 k 染色计数 $\prod_{v \in V} k - N(v) + 1$.

区间图 每个顶点代表一个区间, 有边当且仅当区间有交. 区间图是弦图, 一个完美消除序列是右端点排序.

```
vector <int> L[N];
int seq[N], lab[N], col[N], id[N], vis[N];
void mcs() {
  | for (int i = 0; i < n; i++) L[i].clear();
  | fill(lab + 1, lab + n + 1, 0);
```

```
 | fill(id + 1, id + n + 1, 0);
 | for (int i = 1; i <= n; i++) L[0].push_back(i);
 | int top = 0;
 | for (int k = n; k; k--) {
 |  | int x = -1;
 |  | for ( ; ; ) {
 |  |  | if (L[top].empty()) top --;
 |  |  | else {
 |  |  |  | x = L[top].back(), L[top].pop_back();
 |  |  |  | if (lab[x] == top) break;
 |  |  | }
 |  | }
 |  | seq[k] = x; id[x] = k;
 |  | for (auto v : E[x]) {
 |  |  | if (!id[v]) {
 |  |  |  | L[++lab[v]].push_back(v);
 |  |  |  | top = max(top, lab[v]);
 |  |  | } } } }
bool check() {
 | fill(vis + 1, vis + n + 1, 0);
 | for (int i = n; i; i--) {
 |  | int x = seq[i];
 |  | vector <int> to;
 |  | for (auto v : E[x])
 |  |  | if (id[v] > i) to.push_back(v);
 |  | if (to.empty()) continue;
 |  | int w = to.front();
 |  | for (auto v : to) if (id[v] < id[w]) w = v;
 |  | for (auto v : E[w]) vis[v] = i;
 |  | for (auto v : to)
 |  |  | if (v != w && vis[v] != i) return false;
 | } return true; }
void color() {
 | fill(vis + 1, vis + n + 1, 0);
 | for (int i = n; i; i--) {
 |  | int x = seq[i];
 |  | for (auto v : E[x]) vis[col[v]] = x;
 |  | for (int c = 1; !col[x]; c++)
 |  |  | if (vis[c] != x) col[x] = c;
 | } }
```

2.24 Minimum Mean Cycle 最小平均值环 $O(n^2)$

```
// 点标号为 1,2,···,n, 0 为虚拟源点向其他点连权值为 0 的单向边.
// f[i][v] : 从 0 到 v 恰好经过 i 条路的最短路
ll f[N][N] = {Inf}; int u[M], v[M], w[M]; f[0][0] = 0;
for(int i = 1; i <= n + 1; i ++)
 | for(int j = 0; j < m; j ++)
 |  | f[i][v[j]] = min(f[i][v[j]], f[i - 1][u[j]] + w[j]);
double ans = Inf;
for(int i = 1; i <= n; i ++) {
 | double t = -Inf;
 | for(int j = 1; j <= n; j ++)
 |  | t = max(t, (f[n][i] - f[j][i]) / (double)(n - j));
 | ans = min(t, ans); }
```

2.25 一般图最大匹配 - Blossom

```
int n, m, l, r, u, v, res, q[N], p[N], h[N], fa[N], col[N];
vector<int> G[N];
inline int find(int x) {return fa[x] == x ? x : fa[x] = find(fa[x]);}
int lca(int u, int v) {
 | static int fl[N], tim; ++tim;
 | while (fl[u] != tim) {
 |  | if (u) fl[u] = tim, u = find(p[h[u]]);
 |  | swap(u, v);
```

```
 | } return u; }
void blossom(int u, int v, int t) {
 | while (find(u) != t) {
 |  | p[u] = v, v = h[u], fa[u] = fa[v] = t;
 |  | if (col[v] == 1) col[v] = 2, q[++r] = v;
 |  | u = p[v]; } }
bool match(int rt) {
 | for (int i = 1; i <= n; i++) fa[i] = i, col[i] = 0;
 | l = r = 1, q[1] = rt, col[rt] = 2;
 | while (l <= r) {
 |  | u = q[l++];
 |  | for (int v : G[u]) if (!col[v]) {
 |  |  | col[v] = 1, col[h[v]] = 2, p[v] = u, q[++r] = h[v];
 |  |  | if (!h[v]) {
 |  |  |  | while (u) u = h[p[v]], h[v] = p[v], h[p[v]] = v, v = u;
 |  |  |  | return true;
 |  |  | }
 |  | } else if (col[v] == 2) {
 |  |  | int t = lca(u, v); blossom(u, v, t), blossom(v, u, t);
 |  | }
 | } return false;
} // for(int i=1;i<=n;i++)if(!h[i]&&match(i))++res;
```

2.26 图论结论

2.26.1 最小乘积问题原理

每个元素有两个权值 $\{x_i\}$ 和 $\{y_i\}$, 要求在某个限制下 (例如生成树, 二分图匹配) 使得 $\Sigma x \Sigma y$ 最小. 对于任意一种符合限制的选取方法, 记 $X = \Sigma x_i, Y = \Sigma y_i$, 可看做平面内一点 (X, Y). 答案必在下凸壳上, 找出该下凸壳所有点, 即可枚举获得最优答案. 可以递归求出此下凸壳所有点, 分别找出距 x, y 轴最近的两点 A, B, 分别对应于 $\Sigma y_i, \Sigma x_i$ 最小. 找出距离线段最远的点 C, 则 C 也在下凸壳上, C 点满足 $AB \times AC$ 最小, 也即

$$(X_B - X_A)Y_C + (Y_A - Y_B)X_C - (X_B - X_A)Y_A - (Y_B - Y_A)X_A$$

最小, 后两项均为常数, 因此将所以权值改成 $(X_B - X_A)y_i + (Y_B - Y_A)x_i$, 求同样问题 (例如最小生成树, 最小权匹配) 即可. 求出 C 点以后, 递归 AC, BC.

2.26.2 最小环

无向图最小环: 每次 floyd 到 k 时, 判断 1 到 $k-1$ 的每一个 i, j:

$$\text{ans} = \min\{\text{ans}, d(i, j) + G(i, k) + G(k, j)\}.$$

有向图最小环: 做完 floyd 后, $d(i, i)$ 即为经过 i 的最小环.

2.26.3 度序列的可图性

判断一个度序列是否可转化为简单图, 除了一种贪心构造的方法外, 下列方法更快速. EG 定理: 将度序列从大到小排序得到 $\{d_i\}$, 此序列可转化为简单图当且仅当 Σd_i 为偶数, 且对于任意的 $1 \le k \le n-1$ 满足 $\sum_{i=1}^{k} d_i \le k(k-1) + \sum_{i=k+1}^{n} \min(k, d_i)$.

2.26.4 切比雪夫距离与曼哈顿距离转化

曼哈顿转切比雪夫: $(x + y, x - y)$, 适用于一些每次只能向四联通的格子走一格的问题. 切比雪夫转曼哈顿: $(\frac{x+y}{2}, \frac{x-y}{2})$, 适用于统计距离.

2.26.5 树链的交

```
bool cmp(int a,int b){return dep[a]<dep[b];}
path merge(path u, path v){
 | int d[4], c[2];
 | if (!u.x||!v.x) return path(0, 0);
 | d[0]=lca(u.x,v.x); d[1]=lca(u.x,v.y);
```

```
| d[2]=lca(u.y,v.x); d[3]=lca(u.y,v.y);
| c[0]=lca(u.x,u.y); c[1]=lca(v.x,v.y);
| sort(d,d+4,cmp); sort(c,c+2,cmp);
| if (dep[c[0]] <= dep[d[0]] && dep[c[1]] <= dep[d[2]])
|  | return path(d[2],d[3]);
| else return path(0, 0); }
```

2.26.6 带修改 MST

维护少量修改的 MST (银川 21: 求有 16 个 'e1 or e2' 的限制条件的 MST)

找出必须边将修改边标 $-\infty$, 在 MST 上的其余边为必须边, 以此缩点.

找出无用边将修改边标 ∞, 不在 MST 上的其余边为无用边, 删除之.

假设修改边数为 k, 操作后图中最多剩下 $k+1$ 个点和 $2k$ 条边.

2.26.7 差分约束

$x_r - x_l \le c$:add(l, r, c) $x_r - x_l \ge c$:add(r, l, -c)

2.26.8 Segment Tree Beats

区间 min, 区间求和. 维护最大值 m, 严格次大值 s 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 $m \le x$ 时直接退出; 2, 当 $se < x < ma$ 时, 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 $se \ge x$ 时递归. 均摊 $O(\log^2 n)$.

2.26.9 二分图

最小点覆盖 = 最大匹配数. 独立集与覆盖集互补. 最小点覆盖构造方法: 对二分图流图求割集, 跨过的边指示最小点覆盖.

Hall 定理 $G=(X,Y,E), |M|=|X| \Leftrightarrow \forall S \subseteq X, |S| \le |A(S)|$.

2.26.10 稳定婚姻问题

男士按自己喜欢程度从高到底依次向每位女士求婚, 女士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. 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 $O(n^2)$.

2.26.11 竞赛图 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.

2.26.12 Ramsey Theorem R(3,3)=6, R(4,4)=18

6 个人中存在 3 人相互认识或者相互不认识.

2.26.13 树的计数 Prufer 序列

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 $n_1^{n_2-1} \times n_2^{n_1-1}$.

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: $(\sum c_i)^{m-2} \prod c_i$

2.26.14 有根树计数 1,1,2,4,9,20,48,115,286,719,1842,4766

无标号 $a_{n+1} = 1/n \sum_{k=1}^{n} (\sum_{d|k} d \cdot a(d)) \cdot a(n-k+1)$

2.26.15 无根树计数

n 是奇数时, 有 $a_n - \sum_i^{n/2} a_i a_{n-i}$ 种不同的无根树.

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

2.26.16 生成树计数 Kirchhoff's Matrix-Tree Theorem

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 $u \to v$ 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: $c = |K$ 的任意 1 个 $n-1$ 阶主子式 $|$

有向图外向树计数: $c = |$ 去掉根所在的那阶得到的主子式 $|$

2.26.17 有向图欧拉回路计数 BEST Theorem

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

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: $\mathrm{t_v}(G) = \mathrm{t_w}(G)$.

2.26.18 Tutte Matrix

Tutte matrix A of a graph $G = (V, E)$:

$$A_{ij} = \begin{cases} x_{ij} & \text{if } (i,j) \in E \text{ and } i < j \\ -x_{ij} & \text{if } (i,j) \in E \text{ and } i > j \\ 0 & \text{otherwise} \end{cases}$$

where x_{ij} are indeterminates. The determinant of this skew-symmetric matrix is then a polynomial (in the variables x_{ij}, $i < j$): this coincides with the square of the pfaffian of the matrix A and is non-zero (as a polynomial) if and only if a perfect matching exists.

2.26.19 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.

2.26.20 有向图无环定向, 色多项式

图的色多项式 $P_G(q)$ 对图 G 的 q-染色计数.

Triangle K_3 : $x(x-1)(x-2)$

Complete graph K_n : $x(x-1)(x-2)\cdots(x-(n-1))$

Tree with n vertices : $x(x-1)^{n-1}$

Cycle C_n : $(x-1)^n + (-1)^n(x-1)$

acyclic orientations of an n-vertex graph G is $(-1)^n P_G(-1)$.

2.26.21 拟阵交问题

拟阵定义: , S, T 是独立集, 则 S 子集是, 若 $|S| > |T|$, 则 S 能扩充 T. 最大带权拟阵交问题: 全集 U 中每个元素都有权值 w_i. 设**同一个**全集 U 上有两个满足拟阵性质的集族 $\mathcal{F}_1, \mathcal{F}_2$. 对于 $k = 1..|U|$, 分别求出一个集合 S, 满足 $S \in \mathcal{F}_1 \cap \mathcal{F}_2$ 且 $|S|$ 恰好为 k 的前提下, S 中元素权值和最小.

设集合大小为 k 时已经求出了答案 S. 现在希望求出集合大小为 $k+1$ 的答案. U 中所有元素分为两个集合: 当前答案集合 S, 和剩余集合 $T = U \backslash S$. 考虑 T 中的某个元素 x_i. 记 $A = \{x_i | S \cup \{x_i\} \in \mathcal{F}_1\}$, $B = \{x_i | S \cup \{x_i\} \in \mathcal{F}_2\}$. 如果 T 中某个元素 $x_i \notin A$, 说明 x_i 加进 S 中形成了某个" 环", 从而不满足 $\mathcal{F}_1$ 的限制. 考虑这个" 环" 上每个元素 y_j, 满足 $S \backslash \{y_j\} \cup \{x_i\} \in \mathcal{F}_1$, 将 y_j 向每个 x_i 连边. 如果 T 中某个元素 $x_i \notin B$, 同理找出 S 中每一个元素 y_j 使得 $S \backslash \{y_j\} \cup \{x_i\} \in \mathcal{F}_2$, 将 x_i 向 y_j 连边. 现在求出从 A 到 B 的多源多汇最短路, 权值在点上, 若点属于 T 则权值为正, 否则

属于 S, 权值为负. 最短路上每个 T 中的点放进 S, S 中的点放进 T, 则完成了一次增广. 由于每次增广路的起点和终点都在 T 中, 所以每次增广都会使得 $|S|$ 增加 1.

最大拟阵交问题可以去掉权值直接求增广路.

2.26.22 双极定向

```
//双极定向: 给定无向图和两个极点 s,t, 要求将每条边定向后成为 DAG, 使得 s 可达所有点, 所有点均可达 t
//topo 为定向后 DAG 的拓扑序, 边 (u,v) 定向为 u->v 当且仅当拓扑序中 u 在 v 的前面.
int n, dfn[N], low[N], stamp, p[N], preorder[N], topo[N];
bool fucked = 0, sign[N]; vector<int> G[N];
void dfs(int x, int fa, int s, int t){
 | dfn[x] = low[x] = ++stamp;
 | preorder[stamp] = x, p[x] = fa;
 | if (x == s) dfs(t, x, s, t);
 | for (int y : G[x]){
 |  | if (x == s && y == t) continue;
 |  | if (!dfn[y]){
 |  |  | if (x == s) fucked = true;
 |  |  | dfs(y, x, s, t);
 |  |  | low[x] = min(low[x], low[y]); }
 |  | else if (dfn[y] < dfn[x] && y != fa)
 |  |  | low[x] = min(low[x], dfn[y]); } }
bool bipolar_orientation(int s, int t){
 | G[s].push_back(t), G[t].push_back(s);
 | stamp = fucked = 0, dfs(s, s, s, t);
 | for (int i = 1; i <= n; i++)
 |  | if (i != s && (!dfn[i] || low[i] >= dfn[i]))
 |  |  | fucked = true;
 | if (fucked) return false;
 | sign[s] = 0;//memset sign[] is not necessary
 | int pre[n + 5], suf[n + 5]; // list
 | suf[0] = s; pre[s] = 0, suf[s] = t;
 | pre[t] = s, suf[t] = n + 1; pre[n + 1] = t;
 | for (int i = 3; i <= n; i++){
 |  | int v = preorder[i];
 |  | if (!sign[preorder[low[v]]]){ // insert before p[v]
 |  |  | int P = pre[p[v]];
 |  |  | pre[v] = P, suf[v] = p[v];
 |  |  | suf[P] = pre[p[v]] = v; }
 |  | else{ // insert after p[v]
 |  |  | int S = suf[p[v]];
 |  |  | pre[v] = p[v], suf[x] = S;
 |  |  | suf[p[v]] = pre[S] = v; }
 |  | sign[p[x]] = !sign[preorder[low[x]]]; }
 | for (int x = s, cnt = 0; x != n + 1; x = suf[x])
 |  | topo[++cnt] = x;
 | return true; }
```

2.26.23 图中的环

没有奇环的图是**二分图**, 没有偶环的图是**仙人掌**. 判定没有奇环仅用深度奇偶性判即可; 判定没有偶环的图需要记录覆盖次数判定是否存在奇环有交.

3. Data Structure

3.1 回滚并查集

```cpp
struct DSU {
    std::vector<int> siz;
    std::vector<int> f;
    std::vector<std::array<int, 2>> his;

    DSU() {}
    DSU(int n) { init(n); }

    void init(int n) {
        f.resize(n);
        std::iota(f.begin(), f.end(), 0);
        siz.assign(n, 1);
    }

    bool same(int x, int y) { return find(x) == find(y); }

    int find(int x) {
        while (f[x] != x) {
            x = f[x];
        }
        return x;
    }

    bool merge(int x, int y) {
        x = find(x);
        y = find(y);
        if (x == y) {
            return false;
        }
        if (siz[x] < siz[y]) {
            std::swap(x, y);
        }
        his.push_back({x, y});
        siz[x] += siz[y];
        f[y] = x;
        return true;
    }

    int time() { return his.size(); }

    void revert(int tm) {
        while (his.size() > tm) {
            auto [x, y] = his.back();
            his.pop_back();
            f[y] = y;
            siz[x] -= siz[y];
        }
    }
};
```

3.2 莫队合集

3.2.1 普通莫队

```cpp
struct Q {
    int l, r, id;
} q[N];
bool cmp(Q x, Q y) {
    if (pos[x.l] == pos[y.l])
        return x.r < y.r;
    else
        return pos[x.l] < pos[y.l];
}
```

```
sort(q + 1, q + 1 + m);
for (int i = 1, l = 1, r = 0; i <= m; ++i) {
    while (l > q[i].l) add(a[--l]);  // 左扩展
    while (r < q[i].r) add(a[++r]);  // 右扩展
    while (l < q[i].l) del(a[l++]);  // 左删除
    while (r > q[i].r) del(a[r--]);  // 右删除
}
```

3.2.2 树上莫队

```
// 预处理部分
void dfs2(int u, int t) {
    top[u] = t;
    a[++tim] = u, in[u] = tim;
    if (son[u]) dfs2(son[u], t);
    for (auto v : e[u]) {
        if (v == fa[u] || v == son[u]) continue;
        dfs2(v, v);
    }
    a[++tim] = u, out[u] = tim;
}

// 处理询问部分
int lc = lca(x, y);
q[++tot].t = t;
q[tot].id = tot;
if (in[x] > in[y]) swap(x, y);  // 先 x 后 y
if (lca == x) {                 // 直链情况
    q[tot].l = in[x];
    q[tot].r = in[y];
} else {  // 折链情况
    q[tot].l = out[x];
    q[tot].r = in[y];
    q[tot].lca = lc;  // 以便补偿
}
```

3.3 单点修改线段树

```
#include<bits/stdc++.h>

template <class Info>
struct SegmentTree {
    int n;

    std::vector<Info> info;
    SegmentTree() : n(0) {}
    SegmentTree(int n_, Info v_ = Info()) { init(n_, v_); }
    template <class T>
    SegmentTree(std::vector<T> init_) {
        init(init_);
    }
    void init(int n_, Info v_ = Info()) { init(std::vector(n_, v_)); }
    template <class T>
    void init(std::vector<T> init_) {
        n = init_.size();
        info.assign(4 << std::__lg(n), Info());
        std::function<void(int, int, int)> build = [&](int p, int l, int r) {
            if (r - l == 1) {
                info[p] = init_[l];
                return;
            }
            int m = (l + r) / 2;
            build(2 * p, l, m);
            build(2 * p + 1, m, r);
            pull(p);
        };
```

```
        build(1, 0, n);
    }
    void pull(int p) { info[p] = info[2 * p] + info[2 * p + 1]; }
    void modify(int p, int l, int r, int x, const Info &v) {
        if (r - l == 1) {
            info[p] = v;
            return;
        }
        int m = (l + r) / 2;
        if (x < m) {
            modify(2 * p, l, m, x, v);
        } else {
            modify(2 * p + 1, m, r, x, v);
        }
        pull(p);
    }
    void modify(int p, const Info &v) { modify(1, 0, n, p, v); }
    Info rangeQuery(int p, int l, int r, int x, int y) {
        if (l >= y || r <= x) {
            return Info();
        }
        if (l >= x && r <= y) {
            return info[p];
        }
        int m = (l + r) / 2;
        return rangeQuery(2 * p, l, m, x, y) +
               rangeQuery(2 * p + 1, m, r, x, y);
    }
    Info rangeQuery(int l, int r) { return rangeQuery(1, 0, n, l, r); }

    /**
     * @brief 在线段树上二分，查找区间 [x, y) 中第一个满足 pred 条件的元素位置。
     * * @tparam F 谓词函数类型，应当是 bool(const Info&)
     * @param pred 谓词函数，如果当前区间的合并信息满足条件，则返回 true
     * @return int 第一个满足条件的位置索引，如果不存在则返回 -1
     */
    template <class F>
    int findFirst(int p, int l, int r, int x, int y, F &&pred) {
        // 如果当前区间 [l, r) 与查询区间 [x, y) 没有交集，直接返回
        if (l >= y || r <= x) {
            return -1;
        }
        // 如果当前区间被查询区间完全包含，并且整个区间的信息都不满足谓词，
        // 说明这个区间内不可能有答案，进行剪枝
        if (l >= x && r <= y && !pred(info[p])) {
            return -1;
        }
        // 如果到达叶子节点，说明找到了一个满足条件的最小单位，返回其位置
        if (r - l == 1) {
            return l;
        }
        int m = (l + r) / 2;
        // 优先在左子树中查找，因为要找 " 第一个"
        int res = findFirst(2 * p, l, m, x, y, pred);
        // 如果左子树没找到，再去右子树中查找
        if (res == -1) {
            res = findFirst(2 * p + 1, m, r, x, y, pred);
        }
        return res;
    }
    template <class F>
    int findFirst(int l, int r, F &&pred) {
        return findFirst(1, 0, n, l, r, pred);
    }

    /**
     * @brief 在线段树上二分，查找区间 [x, y) 中最后一个满足 pred
```

```
     * 条件的元素位置。
     * * @tparam F 谓词函数类型，应当是 bool(const Info&)
     * @param pred 谓词函数，如果当前区间的合并信息满足条件，则返回 true
     * @return int 最后一个满足条件的位置索引，如果不存在则返回 -1
     */
    template <class F>
    int findLast(int p, int l, int r, int x, int y, F &&pred) {
        // 如果当前区间 [l, r) 与查询区间 [x, y) 没有交集，直接返回
        if (l >= y || r <= x) {
            return -1;
        }
        // 如果当前区间被查询区间完全包含，并且整个区间的信息都不满足谓词，
        // 说明这个区间内不可能有答案，进行剪枝
        if (l >= x && r <= y && !pred(info[p])) {
            return -1;
        }
        // 如果到达叶子节点，说明找到了一个满足条件的最小单位，返回其位置
        if (r - l == 1) {
            return l;
        }
        int m = (l + r) / 2;
        // 优先在右子树中查找，因为要找 " 最后一个"
        int res = findLast(2 * p + 1, m, r, x, y, pred);
        // 如果右子树没找到，再去左子树中查找
        if (res == -1) {
            res = findLast(2 * p, l, m, x, y, pred);
        }
        return res;
    }
    template <class F>
    int findLast(int l, int r, F &&pred) {
        return findLast(1, 0, n, l, r, pred);
    }
};

struct Info {
    int x = 0;
};

// 区间最大值
Info operator+(const Info &a, const Info &b) { return {std::max(a.x, b.x)}; }

int main() {

    std::vector<Info> a = {{1}, {5}, {2}, {8}, {6}, {7}, {4}, {9}};
    SegmentTree<Info> tr(a);

    // 在一个子区间 [0, 5) 中查找第一个值 >= 6 的元素
    int val = 6;
    int pos = tr.findFirst(0, 5, [&](const Info& v) {
        return v.x >= val;
    });

    return 0;
}
```

3.4 懒标记线段树

```
template <class Info, class Tag>
struct LazySegmentTree {
    int n;
    vector<Info> info;
    vector<Tag> tag;
    LazySegmentTree() : n(0) {}
    LazySegmentTree(int n_, Info v_ = Info()) {
        init(n_, v_);
    }  // 初始为同一种结点
```

```
    template <class T>
    LazySegmentTree(vector<T> init_) {
        init(init_);
    }  // 针对序列初始化

    void init(int n_, Info v_ = Info()) { init(vector(n_, v_)); }
    template <class T>
    void init(vector<T> init_) {
        n = init_.size();
        info.assign(4 << __lg(n), Info());
        tag.assign(4 << __lg(n), Tag());
        // 结点信息为左闭右开
        function<void(int, int, int)> build = [&](int p, int l, int r) {
            if (r - l == 1) {
                info[p] = init_[l];
                return;
            }
            int m = (l + r) / 2;
            build(2 * p, l, m), build(2 * p + 1, m, r);
            pull(p);
        };
        build(1, 0, n);
    }

    void pull(int p) {
        info[p] = info[2 * p] + info[2 * p + 1];
    }  // 向上合并结点
    void apply(int p, const Tag &v) {
        info[p].apply(v);
        tag[p].apply(v);
    }
    void push(int p) {
        apply(2 * p, tag[p]);
        apply(2 * p + 1, tag[p]);
        tag[p] = Tag();
    }  // 懒标记下传
    void modify(int p, int l, int r, int x, const Info &v) {
        if (r - l == 1) {
            info[p] = v;
            return;
        }
        int m = (l + r) / 2;
        push(p);
        if (x < m) {
            modify(2 * p, l, m, x, v);
        } else {
            modify(2 * p + 1, m, r, x, v);
        }
        pull(p);
    }
    void modify(int p, const Info &v) { modify(1, 0, n, p, v); }
    Info rangeQuery(int p, int l, int r, int x, int y) {
        if (l >= y || r <= x) {
            return Info();
        }
        if (l >= x && r <= y) {
            return info[p];
        }
        int m = (l + r) / 2;
        push(p);
        return rangeQuery(2 * p, l, m, x, y) +
               rangeQuery(2 * p + 1, m, r, x, y);
    }
    Info rangeQuery(int l, int r) { return rangeQuery(1, 0, n, l, r); }
    void rangeApply(int p, int l, int r, int x, int y, const Tag &v) {
        if (l >= y || r <= x) {
            return;
```

```
        }
        if (l >= x && r <= y) {
            apply(p, v);
            return;
        }
        int m = (l + r) / 2;
        push(p);
        rangeApply(2 * p, l, m, x, y, v);
        rangeApply(2 * p + 1, m, r, x, y, v);
        pull(p);
    }
    void rangeApply(int l, int r, const Tag &v) {
        return rangeApply(1, 0, n, l, r, v);
    }

    template <class F>
    int findFirst(int p, int l, int r, int x, int y, F &&pred) {
        if (l >= y || r <= x) {
            return -1;
        }
        if (l >= x && r <= y && !pred(info[p])) {
            return -1;
        }
        if (r - l == 1) {
            return l;
        }
        int m = (l + r) / 2;
        push(p);
        int res = findFirst(2 * p, l, m, x, y, pred);
        if (res == -1) {
            res = findFirst(2 * p + 1, m, r, x, y, pred);
        }
        return res;
    }
    template <class F>
    int findFirst(int l, int r, F &&pred) {
        return findFirst(1, 0, n, l, r, pred);
    }
    template <class F>
    int findLast(int p, int l, int r, int x, int y, F &&pred) {
        if (l >= y || r <= x) {
            return -1;
        }
        if (l >= x && r <= y && !pred(info[p])) {
            return -1;
        }
        if (r - l == 1) {
            return l;
        }
        int m = (l + r) / 2;
        push(p);
        int res = findLast(2 * p + 1, m, r, x, y, pred);
        if (res == -1) {
            res = findLast(2 * p, l, m, x, y, pred);
        }
        return res;
    }
    template <class F>
    int findLast(int l, int r, F &&pred) {
        return findLast(1, 0, n, l, r, pred);
    }
};

struct Tag {
    int x = 0;
    void apply(const Tag &t) & { x = std::max(x, t.x); }
};
```

```

struct Info {
    int x = 0;
    void apply(const Tag &t) & { x = std::max(x, t.x); }
};

Info operator+(const Info &a, const Info &b) { return {std::max(a.x, b.x)}; }
```

3.5 线段树合并与分裂

```
int ln[M], rn[M], tot, cnt[M], rt[N];
// 合并 左闭右开
int merge(int u, int v, int l, int r) {
    if (!u || !v) return u + v;
    if (l == r - 1) {
        cnt[u] += cnt[v];
        return u;
    }
    int mid = (l + r) / 2;
    ln[u] = merge(ln[u], ln[v], l, mid);
    rn[u] = merge(rn[u], rn[v], mid, r);
    cnt[u] = cnt[ln[u]] + cnt[rn[u]];
    return u;
}

// 这里需要和主席树区别开，不需要共享节点
void insert(int& u, int l, int r, int x) {
    if (!u) u = ++tot;
    cnt[u]++;
    if (l == r - 1) return;
    int mid = (l + r) / 2;
    if (x < mid)
        insert(ln[u], l, mid, x);
    else
        insert(rn[u], mid, r, x);
}

// op = 0 表示前 k 个给 x 后面的给 y
// op = 1 表示后 k 个给 x 前面的给 y
void split(int x, int& y, int k, int op) {
    if (!x) return;
    y = ++tot;
    if (!op) {
        int s = cnt[ln[x]];
        if (s >= k)
            split(ln[x], ln[y], k, op), swap(rn[x], rn[y]);
        else
            split(rn[x], rn[y], k - s, op);
        cnt[y] = cnt[x] - k, cnt[x] = k;
    } else {
        int s = cnt[rn[x]];
        if (s >= k)
            split(rn[x], rn[y], k, op), swap(ln[x], ln[y]);
        else
            split(ln[x], ln[y], k - s, op);
        cnt[y] = cnt[x] - k, cnt[x] = k;
    }
}
```

3.6 笛卡尔树

```
struct tree {
    int n, rt;
    vector<int> ln, rn;

    // 按照数组下标构成二叉搜索树
```

```
    // 按照 w 构成小根堆。
    void build(vector<int>& w) {
        this->n = w.size();
        ln.resize(n, -1), rn.resize(n, -1);

        vector<int> stk;
        for (int i = 0; i < n; i++) {
            int last = -1;
            // 如果要改成大根堆，只需将下面的 > 改为 <
            while (!stk.empty() && w[stk.back()] > w[i])
                last = stk.back(), stk.pop_back();
            if (!stk.empty()) rn[stk.back()] = i;
            if (last != -1) ln[i] = last;
            stk.push_back(i);
        }
        rt = stk[0];
    }
};
```

3.7 点分治

```
vector<int>e[N];
int vis[N], s[N], maxs[N] = {(int)1e9};
void getrt(int p, int fa, int siz, int &rt) {
    s[p] = 1;
    maxs[p] = 0;
    for (auto [v, c] : e[p]) {
        if (v == fa || vis[v]) continue;
        s[p] += s[v];
        maxs[p] = max(maxs[p], s[v]);
    }
    maxs[p] = max(maxs[p], siz - s[p]);
    if (maxs[p] < maxs[rt]) rt = p;
}
void cal(int p, int siz) {
    int rt = 0;
    getrt(p, 0, siz, rt);
    for (auto [v, c] : e[rt]) {
        if (vis[v]) continue;
        // 计算相关贡献
    }
    vis[rt] = 1;
    for (auto [v, c] : e[rt]) {
        if (vis[v]) continue;
        if (s[v] > s[rt]) s[v] = siz - s[rt];
        cal(v, s[v]);
    }
}
// 调用: cal(1,n);
```

3.8 树上启发式合并

```
// 轻重链剖分的性质保证每个点只会被加入和删除 log 次

struct HLD {
    vector<vector<int>> e;
    vector<int> sz, son;
    int hson;  // 不需要被添加的重子树
    HLD(int n) {
        e.resize(n + 1);
        sz.resize(n + 1);
        son.resize(n + 1);
    }

    void addEdge(int u, int v) {
        e[u].push_back(v);
```

```
        e[v].push_back(u);
    }

    void dfs1(int u, int f) {
        sz[u] = 1;
        for (auto v : e[u]) {
            if (v == f) continue;
            dfs1(v, u);
            sz[u] += sz[v];
            if (sz[v] > sz[son[u]]) son[u] = v;
        }
    }

    void add(int u, int f) {
        /*
        加点逻辑
        */
        for (auto v : e[u]) {
            if (v == f || v == hson) continue;
            add(v, u);
        }
    }

    void sub(int u, int f) {
        /*
        删点逻辑。
        */
        for (auto v : e[u]) {
            if (v == f) continue;
            sub(v, u);
        }
    }

    void dfs2(int u, int f, int op) {
        for (auto v : e[u]) {
            if (v == f || v == son[u]) continue;
            dfs2(v, u, 0);
        }
        if (son[u]) {
            dfs2(son[u], u, 1);
        }
        hson = son[u];
        add(u, f);

        /*
        获取该点的答案。
        */

        if (!op) {
            sub(u, f);
        }
    }

    void work(int rt = 1) {
        dfs1(rt, 0);
        dfs2(rt, 0, 0);
    }
};
```

3.9 $O(n \log n) - O(1)$ LCA

```
class Magic {
   public:
    int n;                                // 树中的节点数
    std::vector<std::vector<int>> adj;  // 树的邻接表
    std::vector<int> dep;                 // dep[i] 存储节点 i 的深度
    std::vector<int> pos;    // pos[i] 存储节点 i 在欧拉序中首次出现的位置
```

```
    std::vector<int> euler;  // 树的欧拉序
    int tot;                 // 欧拉序中的节点总数

  private:
    std::vector<std::vector<int>> st;  // 用于在欧拉序上进行 RMQ 的稀疏表

  public:
    /**
     * @brief 默认构造函数。
     */
    Magic() : n(0), tot(0) {}

    /**
     * @brief 构造函数，为大小为 n 的树初始化对象。
     * @param n 树中的节点数。
     */
    Magic(int n) { init(n); }

    /**
     * @brief 为大小为 n 的树初始化或重新初始化数据结构。
     * @param n 节点数。
     */
    void init(int n) {
        this->n = n;
        tot = 0;
        adj.assign(n, {});
        dep.resize(n);
        pos.resize(n);
        // 欧拉序最多有 2*n - 1 个元素
        euler.resize(n * 2);
    }

    void addEdge(int u, int v) {
        adj[u].push_back(v);
        adj[v].push_back(u);
    }

    /**
     * @brief 执行深度优先搜索以构建欧拉序和深度数组。
     * @param u 当前节点。
     * @param parent 当前节点的父节点。
     * @param d 当前节点的深度。
     */
    void dfs(int u, int parent, int d) {
        dep[u] = d;
        pos[u] = tot;
        euler[tot++] = u;
        for (int v : adj[u]) {
            if (v == parent) continue;
            dfs(v, u, d + 1);
            euler[tot++] = u;
        }
    }

    void work(int root = 0) {
        // 1. 使用 DFS 构建欧拉序和深度数组。
        tot = 0;
        dfs(root, -1, 0);

        // 2. 初始化并构建用于 RMQ 的稀疏表。
        // 表中存储节点索引，并根据它们的深度进行比较。
        int max_log = (tot > 0) ? __lg(tot) : 0;
        st.assign(max_log + 1, std::vector<int>(tot));

        st[0].assign(euler.begin(), euler.begin() + tot);

        for (int j = 1; j <= max_log; j++) {
```

```
            for (int i = 0; i + (1 << j) <= tot; i++) {
                int u1 = st[j - 1][i];
                int u2 = st[j - 1][i + (1 << (j - 1))];
                // 存储深度较小的节点。
                st[j][i] = (dep[u1] < dep[u2]) ? u1 : u2;
            }
        }
    }

    int lca(int u, int v) {
        int l = pos[u];
        int r = pos[v];
        if (l > r) {
            std::swap(l, r);
        }
        int k = __lg(r - l + 1);

        int u1 = st[k][l];
        int u2 = st[k][r - (1 << k) + 1];

        return (dep[u1] < dep[u2]) ? u1 : u2;
    }
};
```

3.10 LCT 动态树

```
// 记得初始化 mn; 维护虚子树: access link cut pushup
int fa[MX], ch[MX][2], w[MX], mn[MX], mark[MX];
int get(int x) {return x == ch[fa[x]][1];}
int nrt(int x) {return get(x) || x == ch[fa[x]][0];}
void pushup(int x) {
 | mn[x] = w[x];
 | if (lch(x)) mn[x] = min(mn[x], mn[lch(x)]);
 | if (rch(x)) mn[x] = min(mn[x], mn[rch(x)]); }
void rev(int x) {mark[x] ^= 1, swap(lch(x), rch(x));}
void pushdown(int x) {
 | if (mark[x]) {
 |  | if (lch(x)) rev(lch(x));
 |  | if (rch(x)) rev(rch(x));
 |  | mark[x] = false; } }
void rot(int x) {
 | int f = fa[x], gf = fa[f];
 | int which = get(x), W = ch[x][!which];
 | if (nrt(f)) ch[gf][ch[gf][1] == f] = x;
 | ch[x][!which] = f, ch[f][which] = W;
 | if (W) fa[W] = f;
 | fa[f] = x, fa[x] = gf;
 | pushup(f); }
void splay(int x) {
 | static int stk[MX];
 | int f = x, dep = 0; stk[++dep] = f;
 | while (nrt(f)) stk[++dep] = f = fa[f];
 | while (dep) pushdown(stk[dep--]);
 | while (nrt(x)) {
 |  | if (nrt(f = fa[x])) rot(get(x) == get(f) ? f : x);
 |  | rot(x);
 | } pushup(x); }
void access(int x) {
 | for(int y = 0 ; x ; x = fa[y = x])
 |  | splay(x), rch(x) = y, pushup(x); }
void makeroot(int x) {access(x), splay(x), rev(x);}
void split(int x, int y) {makeroot(x), access(y), splay(y);}
int findroot(int x) {
 | access(x), splay(x);
 | while (lch(x)) pushdown(x), x = lch(x);
 | return splay(x), x; | }
void link(int x, int y) {
```

```
  | makeroot(x);
  | if (findroot(y) != x) fa[x] = y; }
void cut(int x, int y) {
  | makeroot(x);
  | if (findroot(y) != x || fa[y] != x || lch(y)) return;
  | rch(x) = fa[y] = 0, pushup(x); }
```

3.11 可持久化 Treap

```
/* 不可持久化：把 copy(a, b) 换成 a = b，并且去除新建结点 */
const int MX = (2e5 + 233) * 18 * 8;
int vcnt;
struct node {
  int sz, ch[2], pri;
  int rev; LL sum; int val;
} tr[MX];
int newnode(int v) {
  static mt19937 rng(114514);
  int x = ++vcnt;
  tr[x].sz = 1;
  lch = rch = 0;
  tr[x].pri = rng();
  tr[x].sum = tr[x].val = v;
  tr[x].rev = false;
  return x; }
void copy(int x, int y) { tr[x] = tr[y]; }
int merge(int x, int y) {
  if (!x || !y) return x + y;
  int z = ++vcnt;
  if (tr[x].pri < tr[y].pri) {
    pushdown(x); copy(z, x);
    tr[z].ch[1] = merge(tr[z].ch[1], y);
  } else {
    pushdown(y); copy(z, y);
    tr[z].ch[0] = merge(x, tr[z].ch[0]); }
  pushup(z); return z; }
void split(int x, int dsz, int &r1, int &r2) {
  if (!x) {
    r1 = r2 = 0;
  } else {
    pushdown(x);
    if (tr[lch].sz + 1 <= dsz) {
      r1 = ++vcnt; copy(r1, x);
      split(tr[r1].ch[1], dsz - 1 - tr[lch].sz
        , tr[r1].ch[1], r2);
      pushup(r1);
    } else {
      r2 = ++vcnt; copy(r2, x);
      split(tr[r2].ch[0], dsz, r1, tr[r2].ch[0]);
      pushup(r2); } } }
```

3.12 PBDS 实现平衡树

```
#include <bits/extc++.h>
#include <bits/stdc++.h>
using namespace std;
using namespace __gnu_pbds;
struct node {
    int a, b;
};
struct cmp {
    bool operator()(const node& a, const node& b) const {
        if (a.a != b.a) return a.a < b.a;
        return a.b < b.b;
    }
};
```

```cpp
// 基于自定义类型的平衡树
tree<node,                                // 元素类型
     null_type,                           // 无映射值
     cmp,                                 // 比较策略
     rb_tree_tag,                         // 底层使用红黑树
     tree_order_statistics_node_update    // 支持排名更新
     >
    pbds;

// 基于 int 类型的平衡树
tree<int, null_type, std::less<int>, rb_tree_tag,
     tree_order_statistics_node_update>
    tr;
void solve() {
    // 插入
    pbds.insert({1, 1});
    pbds.insert({1, 2});
    pbds.insert({2, 2});

    // 访问
    for (auto [a, b] : pbds) cout << a << ' ' << b << endl;

    // 删除
    pbds.erase({1, 0});  // 无用
    pbds.erase({1, 1});

    // 获取元素的排名
    // 注意，返回值为 0base
    cout << pbds.order_of_key({2, 2}) + 1 << endl;  // 2
    cout
        << pbds.order_of_key({1, 0}) + 1
        << endl;  // 1
                  // (元素可以不在集合中，但仍然可以查找排名，此时可当作其在集合)

    // 查找排名第 k 小（注意，这里同样为 0base)
    auto no = pbds.find_by_order(2 - 1);
    cout << no->a << ' ' << no->b << endl;  //(2,2)

    // 存在性
    auto it = pbds.find({1, 0});
    cout << (it != pbds.end()) << endl;  // 0

    // 二分
    auto itt = pbds.lower_bound({2, 1});
    cout << itt->a << ' ' << itt->b << endl;  //(2,2)

    // 清空
    pbds.clear();
}
int main() {
    solve();
    return 0;
}
```

3.13 PBDS 实现可并堆

```cpp
#include <bits/extc++.h>
#include <bits/stdc++.h>

using namespace __gnu_pbds;
using namespace std;

template <typename T, typename Cmp = std::less<T>>
using heap = __gnu_pbds::priority_queue<T, Cmp>;

void solve() {
    // 创建一个小根堆（默认是大根堆，std::less<int> 使其成为最大堆）
```

```cpp
    // 使用 std::greater<int> 来创建一个最小堆
    heap<int, std::greater<int>> h1;

    // push: 插入元素
    h1.push(10);
    h1.push(5);
    h1.push(20);

    // 获取堆顶元素（最小值）
    cout << "h1 堆顶元素: " << h1.top() << endl;  // 5

    // 弹出堆顶元素
    h1.pop();
    cout << "h1 pop 后堆顶元素: " << h1.top() << endl;  // 10

    // 合并操作
    heap<int, std::greater<int>> h2;
    h2.push(8);
    h2.push(10);

    cout << " 合并前 h1 的大小: " << h1.size() << endl;  // 2
    cout << " 合并前 h2 的大小: " << h2.size() << endl;  // 2

    // join: 将 h2 合并到 h1 中。操作后，h2 会被清空
    // 该操作复杂度为 O(1)
    h1.join(h2);

    cout << " 合并后 h1 的大小: " << h1.size() << endl;  // 4
    cout << " 合并后 h2 的大小: " << h2.size() << endl;  // 0
    cout << " 合并后 h1 的堆顶: " << h1.top() << endl;   // 8

    // 遍历堆中元素
    // 注意：遍历顺序不保证有序
    for (int val : h1) {
        cout << val << " ";
    }
    cout << endl;

    // 修改元素
    // modify 需要一个指向元素的迭代器
    // AI 称 push 和 pop 某个元素不会使其他迭代器元素失效
    auto it = h1.push(12);  // push 会返回一个迭代器
    h1.push(30);

    cout << " 修改前堆顶: " << h1.top() << endl;  // 8

    // 将值为 12 的元素修改为 2
    h1.modify(it, 2);

    cout << " 修改后堆顶: " << h1.top() << endl;  // 2

    // 清空堆
    h1.clear();
}

int main() {
    solve();
    return 0;
}
```

3.14 手写 bitset

```cpp
struct Bitset {
    typedef unsigned int ui;
    typedef unsigned long long ll;
#define all(x) (x).begin(), (x).end()
    const static ll B = -1llu;
```

```
    vector<ll> a;
    int n;
    Bitset() {}
    Bitset(int _n) : n(_n), a(_n + 63 >> 6) {}
    bool operator[](int x) const {
        assert(x >= 0 && x < n);
        return a[x >> 6] >> (x & 63) & 1;
    }
    void set(int x, bool y) {
        assert(x >= 0 && x < n);
        a[x >> 6] = (a[x >> 6] & (B ^ 1 << (x & 63))) | ((ll)y << (x & 63));
    }
    void set(int x) {
        assert(x >= 0 && x < n);
        a[x >> 6] |= 1llu << (x & 63);
    }
    void set() {
        memset(a.data(), 0xff, a.size() * sizeof a[0]);
        a.back() &= (1llu << 1 + (n - 1 & 63)) - 1;
    }
    void reset(int x) { a[x >> 6] &= ~(1llu << (x & 63)); }
    void reset() { memset(a.data(), 0, a.size() * sizeof a[0]); }
    int count() const {
        int r = 0;
        for (ll x : a) r += __builtin_popcountll(x);
        return r;
    }
    Bitset &operator|=(const Bitset &o) {
        assert(n == o.n);
        for (int i = 0; i < a.size(); i++) a[i] |= o.a[i];
        return *this;
    }
    Bitset operator|(Bitset o) {
        o |= *this;
        return o;
    }
    Bitset &operator&=(const Bitset &o) {
        assert(n == o.n);
        for (int i = 0; i < a.size(); i++) a[i] &= o.a[i];
        return *this;
    }
    Bitset operator&(Bitset o) {
        o &= *this;
        return o;
    }
    Bitset &operator^=(const Bitset &o) {
        assert(n == o.n);
        for (int i = 0; i < a.size(); i++) a[i] ^= o.a[i];
        return *this;
    }
    Bitset operator^(Bitset o) {
        o ^= *this;
        return o;
    }
    Bitset operator~() const {
        auto r = *this;
        for (ll &x : r.a) x = ~x;
        return r;
    }
    Bitset &operator<<=(int x) {
        if (x >= n) {
            fill(all(a), 0);
            return *this;
        }
        assert(x >= 0);
        int y = x >> 6;
        x &= 63;
```

```
        for (int i = (int)a.size() - 1; i > y; i--)
            a[i] = a[i - y] << x | a[i - y - 1] >> 64 - x;
        a[y] = a[0] << x;
        memset(a.data(), 0, y * sizeof a[0]);
        // fill_n(a.begin(),y,0);
        a.back() &= (1llu << 1 + (n - 1 & 63)) - 1;
        return *this;
    }
    Bitset operator<<(int x) {
        auto r = *this;
        r <<= x;
        return r;
    }
    Bitset &operator>>=(int x) {
        if (x >= n) {
            fill(all(a), 0);
            return *this;
        }
        assert(x >= 0);
        int y = x >> 6, R = (int)a.size() - y - 1;
        x &= 63;
        for (int i = 0; i < R; i++)
            a[i] = a[i + y] >> x | a[i + y + 1] << 64 - x;
        a[R] = a.back() >> x;
        memset(a.data() + R + 1, y * sizeof a[0]);
        // fill(R+1+all(a),0);
        return *this;
    }
    Bitset operator>>(int x) {
        auto r = *this;
        r >>= x;
        return r;
    }
    void range_set(int l, int r)  //[l,r) to 1
    {
        if (l >> 6 == r >> 6) {
            a[l >> 6] |= (1llu << r - l) - 1 << (l & 63);
            return;
        }
        if (l & 63) {
            a[l >> 6] |= ~((1llu << (l & 63)) - 1);  //[l&63,64)
            l = (l >> 6) + 1 << 6;
        }
        if (r & 63) {
            a[r >> 6] |= (1llu << (r & 63)) - 1;
            r = (r >> 6) - 1 << 6;
        }
        memset(a.data() + (l >> 6), 0xff, (r - l >> 6) * sizeof a[0]);
    }
    void range_reset(int l, int r)  //[l,r) to 0
    {
        if (l >> 6 == r >> 6) {
            a[l >> 6] &= ~((1llu << r - l) - 1 << (l & 63));
            return;
        }
        if (l & 63) {
            a[l >> 6] &= (1llu << (l & 63)) - 1;  //[l&63,64)
            l = (l >> 6) + 1 << 6;
        }
        if (r & 63) {
            a[r >> 6] &= ~((1llu << (r & 63)) - 1);
            r = (r >> 6) - 1 << 6;
        }
        memset(a.data() + (l >> 6), 0, (r - l >> 6) * sizeof a[0]);
    }
    void range_set(int l, int r, bool x)  //[l,r)
    {
```

```
        if (x)
            range_set(l, r);
        else
            range_reset(l, r);
    }
};
```

4. String

4.1 最小表示法

```
int min_pos(vector<int> a) { // 0-based
 | int n = a.size(), i = 0, j = 1, k = 0;
 | while (i < n && j < n && k < n) {
 |  | auto u = a[(i + k) % n]; auto v = a[(j + k) % n];
 |  | int t = u > v ? 1 : (u < v ? -1 : 0);
 |  | if (t == 0) k++; else {
 |  |  | if (t > 0) i += k + 1; else j += k + 1;
 |  |  | if (i == j) j++;
 |  |  | k = 0; } } return min(i, j); }
```

4.2 Manacher

```
vector<int> manacher(string& s) {
 | //用 '#' 分割字符
    string t = "#";
    for (auto c : s) {
        t += c;
        t += '#';
    }
    int n = t.size();
    vector<int> r(n);//回文半径
    for (int i = 0, j = 0; i < n; i++) {
        if (2 * j - i >= 0 && j + r[j] > i) {
            r[i] = min(r[2 * j - i], j + r[j] - i);
        }
        while (i - r[i] >= 0 && i + r[i] < n && t[i - r[i]] == t[i + r[i]]) r[i]++;

        if (i + r[i] > j + r[j]) {
            j = i;
        }
    }
    return r;
}
```

4.3 KMP, exKMP

```
void kmp(char *s, int n) { // 1-based
 | fail[0] = fail[1] = 0;
 | for (int i = 1; i < n; i++) { int j = fail[i];
 |  | while (j && s[i + 1] != s[j + 1]) j = fail[j];
 |  | if (s[i + 1] == s[j + 1]) fail[i + 1] = j + 1;
 |  | else fail[i + 1] = 0; } }
void exkmp(char *s, int *a, int n) { // 1-based
 | int l = 0, r = 0; a[1] = n;
 | for (int i = 2; i <= n; i++) {
 |  | a[i] = i > r ? 0 : min(r - i + 1, a[i - l + 1]);
 |  | while (i+a[i] <= n && s[1+a[i]] == s[i+a[i]]) a[i]++;
 |  | if (i + a[i] - 1 > r) {l = i; r = i + a[i] - 1;}}}
```

4.4 AC 自动机

注意代码是以 0 为根的，如果要 1-base 的话要改一下没有儿子时的逻辑。

```
int ch[MAXN][26], fail[MAXN], q[MAXN], cnt = 0;
int insert(const char *c) { int x = 0; while (*c) {
 | if (!ch[x][*c - 'a']) ch[x][*c - 'a'] = ++cnt;
 | x = ch[x][*c++ - 'a']; } return x; }
void getfail() { int x, head = 0, tail = 0;
 | for (int c = 0; c < 26; c++) if (ch[0][c])
 |  |  | q[tail++] = ch[0][c];
```

```
| while (head != tail) { x = q[head++];
|  | for (int c = 0; c < 26; c++) { if (ch[x][c]) {
|  |  |  | fail[ch[x][c]] = ch[fail[x]][c];
|  |  |  | q[tail++] = ch[x][c];
|  |  | } else ch[x][c] = ch[fail[x]][c]; } } }
```

4.5 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 = s1s2sk, 其中 si 是 Lyndon 串, si ≥ si+1. 且这种分解方法是唯一的.
//后缀排序后, 排名的所有前缀最小值构成了 Ly 分解的左端点。
void mnsuf(char *s, int *mn, int n){ // 每个前缀的最小后缀
 | //1 - base , 求 Lyndon 分解去掉 mn 即可
    for(int i = 1; i <= n;)
        int j = i + 1, k = i; mn[i] = i;
        for(; j <= n && s[k] <= s[j]; j++){
            if(s[k] < s[j]) k = mn[j] = i;
                else mn[j] = mn[k] + j - k, k++;
        for(; i <= k; i += j - k) {} } } //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 + 1, mx + n + 1, 0); // 1 - base
    for(int i = 1; i <= n;){
        int j = i + 1, k = i; !mx[i] ? mx[i] = i : 0;
        for(; j <= n && s[k] >= s[j]; j++){
            !mx[j] ? mx[j] = i : 0;
            s[k] > s[j] ? k = i : k++; }
        for(; i <= k; i += j - k) {} } }
```

4.6 后缀自动机

```
struct SAM {
    static constexpr int M = 26;  // 字符集大小
    struct Node {
        int len;            // 该状态代表的最长字符串的长度
        int link;           // 后缀链接
        array<int, M> nxt;  // 转移边
        Node() : len{}, link{}, nxt{} {}
    };
    vector<Node> t;  // 状态节点数组
    int last = 1;    // 指向上一个完整字符串对应的状态节点

    SAM() { init(); }

    void init() {
        t.assign(2, Node());
        t[0].len = -1;  // 虚空节点，方便处理
        t[0].nxt.fill(1);
        // t[1] 是初始状态节点，代表空串""
    }

    int newNode() {
        t.push_back(Node());
        return t.size() - 1;
    }

    void extend(int x) {
        int cur = newNode();
        t[cur].len = t[last].len + 1;
        int p = last;
        while (p != 0 && t[p].nxt[x] == 0) {
            t[p].nxt[x] = cur;
            p = t[p].link;
        }
        int q = t[p].nxt[x];
        if (p == 0) {
```

```
            t[cur].link = 1;
        } else if (t[q].len == t[p].len + 1) {
            t[cur].link = q;
        } else {
            int clone = newNode();
            t[clone].link = t[q].link;
            t[clone].nxt = t[q].nxt;
            t[clone].len = t[p].len + 1;
            t[cur].link = clone;
            t[q].link = clone;
            while (p != 0 && t[p].nxt[x] == q) {
                t[p].nxt[x] = clone;
                p = t[p].link;
            }
        }
        last = cur;
        return;
    }

    int nxt(int p, int x) { return t[p].nxt[x]; }
    int link(int p) { return t[p].link; }
    int len(int p) { return t[p].len; }
    int size() { return t.size(); }

    static inline int num(int x) { return x - 'a'; }
    SAM(string &s) {
        init();
        build(s);
    }

    // 完整构建 SAM
    void build(string &s) {
        for (auto &c : s) {
            extend(num(c));
        }
        get_out_linktree();
    }

    // ---- 可选部分 ---- //
    vector<vector<int>> ot;  // out-link-tree, parent 树
    vector<int> endpos_size;

    // 构建 parent 树
    void get_out_linktree() {
        ot.assign(t.size(), {});
        for (int i = 2; i < t.size(); i++) {
            ot[t[i].link].push_back(i);
        }
    }

    // 计算每个状态的 endpos 集合大小
    void calc_endpos_size(string &s) {
        endpos_size.resize(t.size());
        int p = 1;
        for (auto c : s) {
            p = t[p].nxt[num(c)];
            endpos_size[p]++;
        }
        auto dfs = [&](auto &&self, int p) -> void {
            for (auto s : ot[p]) {
                self(self, s);
                endpos_size[p] += endpos_size[s];
            }
        };
        dfs(dfs, 1);
        endpos_size[1] = 1;
    }
```

```
};
```

4.7 后缀数组

```
struct SA {
    int n;
    std::vector<int> sa, rk, lc;
    /*
    sa[k] = 后缀排名为 k 的后缀起始位置（k = 0..n-1）
    rk[pos] = 后缀 s[pos..] 的排名（0..n-1）
    lc[i] = LCP(sa[i], sa[i+1])

    求任意两个后缀的 LCP：LCP(s[a, n], s[b, n])
    LCP(LCP(s[l], s[l + 1]) ... LCP(s[r - 1], s[r]))
    min(lc[l] ... lc[r-1]) = rmq(l, r-1)

    */
    SA(std::string s) {
        n = s.size();
        sa.resize(n);
        lc.resize(n - 1);
        rk.resize(n);
        std::iota(sa.begin(), sa.end(), 0);
        std::sort(sa.begin(), sa.end(),
                  [&](int a, int b) { return s[a] < s[b]; });
        rk[sa[0]] = 0;
        for (int i = 1; i < n; i++) {
            rk[sa[i]] = rk[sa[i - 1]] + (s[sa[i]] != s[sa[i - 1]]);
        }
        int k = 1;
        std::vector<int> tmp, cnt(n);
        tmp.reserve(n);
        while (rk[sa[n - 1]] < n - 1) {
            tmp.clear();
            for (int i = 0; i < k; i++) {
                tmp.push_back(n - k + i);
            }
            for (auto i : sa) {
                if (i >= k) {
                    tmp.push_back(i - k);
                }
            }
            std::fill(cnt.begin(), cnt.end(), 0);
            for (int i = 0; i < n; i++) {
                cnt[rk[i]]++;
            }
            for (int i = 1; i < n; i++) {
                cnt[i] += cnt[i - 1];
            }
            for (int i = n - 1; i >= 0; i--) {
                sa[--cnt[rk[tmp[i]]]] = tmp[i];
            }
            std::swap(rk, tmp);
            rk[sa[0]] = 0;
            for (int i = 1; i < n; i++) {
                rk[sa[i]] =
                    rk[sa[i - 1]] + (tmp[sa[i - 1]] < tmp[sa[i]] ||
                                     sa[i - 1] + k == n ||
                                     tmp[sa[i - 1] + k] < tmp[sa[i] + k]);
            }
            k *= 2;
        }
        for (int i = 0, j = 0; i < n; i++) {
            if (rk[i] == 0) {
                j = 0;
            } else {
                for (j -= j > 0; i + j < n && sa[rk[i] - 1] + j < n &&
```

```
                                s[i + j] == s[sa[rk[i] - 1] + j];) {
                    j++;
                }
                lc[rk[i] - 1] = j;
            }
        }
    }
};
```

4.8 Suffix Balanced Tree 后缀平衡树

```
// 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
```

4.9 广义在线 SAM

```
struct SAM{
int tot,fail[MM],len[MM],t[MM][26];
SAM(){tot=1;}
int insert(int c,int last){
 | if(t[last][c]){
 |  | int p=last,q=t[p][c];
 |  | if(len[p]+1==len[q])return q;
 |  | else { int nq=++tot;
 |  |  | fail[nq]=fail[q];fail[q]=nq;
 |  |  | len[nq]=len[p]+1;memcpy(t[nq],t[q],sizeof(t[q]));
 |  |  | for(;p && t[p][c]==q;p=fail[p])t[p][c]=nq;
 |  |  | //可以直接复制下面的代码。
 |  |  | return nq; } }
 | int p=last,np=++tot;
 | len[np]=len[p]+1;
 | for(;p && !t[p][c];p=fail[p])t[p][c]=np;
 | if(!p)fail[np]=1;
 | else {
 |  | int q=t[p][c];
 |  | if(len[q]==len[p]+1)fail[np]=q;
 |  | else { int nq=++tot;
 |  |  | fail[nq]=fail[q];fail[q]=nq;
 |  |  | len[nq]=len[p]+1;memcpy(t[nq],t[q],sizeof(t[q]));
 |  |  | for(;p && t[p][c]==q;p=fail[p])t[p][c]=nq;
 |  |  | fail[np]=nq; } }
 | return np; } }sam;
// scanf("%s",st+1);int slen=strlen(st+1);
// int last=1;
// for(int j=1;j<=slen;j++)last=sam.insert(st[j]-'a',last);
```

4.10 回文自动机

1 的子树是长为偶数的串, 0 的子树是长为奇数的. 1 代表空串, 0 没有意义.

```
constexpr int ALPHA_SIZE = 26;
```

```
struct PAM
{
    struct Node{
        array<int,ALPHA_SIZE>next;//可以换成 map
        int dep;//回文树上深度 / * 当前 * 以这个点结尾的回文后缀个数
        int len;//回文长度
        int cnt;//作为最长回文后缀的次数，调用 calc_cnt 后变成该回文子串数量
        int fail;//指向最长回文真后缀
        int trans;//长度小于等于自身一半的最长回文后缀
        Node():next{},dep{},len{},cnt{1},fail{},trans{}{}
    };

    static constexpr int odd_root = 0;
    static constexpr int even_root = 1;
    //odd root -> 0
    //even root -> 1

    vector<Node>t;
    int suff;
    string s;

    PAM()
    {
        init();
    }

    PAM(string &s)
    {
        init();
        for(auto ch : s){
            add(ch);
        }
    }

    void init()
    {
        t.assign(2,Node());
        t[0].len = -1;
        suff = 1;
        s.clear();
    }

    int newNode()
    {
        t.emplace_back();
        return t.size() - 1;
    }

    constexpr int num(const char& c)noexcept
    {
        return c - 'a';
    };

    bool add(char c)
    {
        s += c;
        int x = num(c);
        int cur = get_fail(suff);
        if(t[cur].next[x]){//exist
            suff = t[cur].next[x];
            t[suff].cnt++;
            return false;
        }

        int p = newNode();
        suff = p;//new longest palindrome suffix
        t[p].len = t[cur].len + 2;
```

```
        t[cur].next[x] = p;
        if(t[p].len == 1){//trans form odd_root
            t[p].fail = even_root;//even root
            t[p].trans = even_root;
            t[p].dep = 1;
            return true;
        }
        int f = get_fail(t[cur].fail);// find new fail begin at lps(cur)
        t[p].fail = t[f].next[x];
        int tf = get_trans(t[cur].trans, t[p].len);
        t[p].trans = t[tf].next[x];
        t[p].dep = t[t[p].fail].dep + 1;
        return true;
    }

    int get_fail(int p)
    {
        // if p == odd_root -> len = -1, ok
        int len = s.length() - 1;
        while(len - t[p].len - 1 < 0 || s[len] != s[len - t[p].len - 1])p = t[p].fail;
        return p;
    }

    int get_trans(int p, int plen)
    {
        int len = s.length() - 1;
        //(t[p].len + 2) * 2 < plen
        while(t[p].len * 2 > plen - 4 || len - t[p].len - 1 < 0 || s[len] != s[len - t[p].len -
          ↪ 1])p = t[p].fail;
        return p;
    }

    void calc_cnt()
    {
        for(int i = t.size() - 1;i > 1;i--){
            t[t[i].fail].cnt += t[i].cnt;
        }
    }
};
```

4.11 回文自动机 + dp

使用 slink 指针优化 dp.

```
//使用 slink 指针来优化转移为回文子串的 dp
constexpr int ALPHA_SIZE = 26;
struct PAM
{
    struct Node{
        array<int,ALPHA_SIZE>next;
        int dep;
        int len;
        int cnt;
        int fail;//lps
        int diff;//与 lps 的长度之差
        int slink;//指向第一个 diff 不等于自身的回文后缀
        Node():next{},dep{},len{},cnt{1},fail{},diff{},slink{}{}
    };

    static constexpr int odd_root = 0;
    static constexpr int even_root = 1;
    //odd root -> 0
    //even root -> 1

    vector<Node>t;
    int suff;
    string s;
```

```
    PAM()
    {
        init();
    }

    PAM(string &s)
    {
        init();
        for(auto ch : s){
            add(ch);
        }
    }

    void init()
    {
        t.assign(2,Node());
        t[0].len = -1;
        t[1].diff = 1;
        suff = 1;
        s.clear();
    }

    int newNode()
    {
        t.emplace_back();
        return t.size() - 1;
    }

    constexpr int num(const char& c)noexcept
    {
        return c - 'a';
    };

    bool add(char c)
    {
        s += c;
        int x = num(c);
        int cur = get_fail(suff);
        if(t[cur].next[x]){//exist
            suff = t[cur].next[x];
            t[suff].cnt++;
            return false;
        }

        int p = newNode();
        suff = p;//new longest palindrome suffix
        t[p].len = t[cur].len + 2;
        t[cur].next[x] = p;
        if(t[p].len == 1){//trans form odd_root
            t[p].fail = even_root;//even root
            t[p].dep = 1;
            t[p].diff = 1;
            t[p].slink = 1;//even_root
            return true;
        }
        int f = get_fail(t[cur].fail);// find new fail begin at lps(cur)
        t[p].fail = t[f].next[x];
        t[p].dep = t[t[p].fail].dep + 1;
        t[p].diff = t[p].len - t[t[p].fail].len;
        if(t[p].diff == t[t[p].fail].diff)t[p].slink = t[t[p].fail].slink;
        else t[p].slink = t[p].fail;
        return true;
    }

    int get_fail(int p)
    {
```

```
        // if p == odd_root -> len = -1, ok
        int len = s.length() - 1;
        while(len - t[p].len - 1 < 0 || s[len] != s[len - t[p].len - 1])p = t[p].fail;
        return p;
    }

};
```

4.12 Runs

```
struct Runs{
    int l,r,p;
};vector<Runs> run;
bool operator==(Runs x,Runs y){return x.l==y.l && x.r==y.r;}
int gl(int x,int y); // 求 S[1,x],S[1,y] 的最长公共后缀
int gr(int x,int y); // 求 S[x,n],S[y,n] 的最长公共前缀
//上面两个可以用 二分 + Hash 或者后缀数组实现。
bool getcmp(int x,int y){//S[x,n]<S[y,n]
 | int len=gr(x,y);
 | return st[x+len]<st[y+len];}
int ly[N];
void lyndon(bool type){//后缀排序法求 Lyndon
 | stack<PII> stk;stk.push({n,n});ly[n]=n;
 | for(int i=n-1;i>=1;i--){
 | | int now=i;
 | | while(!stk.empty() && getcmp(i,stk.top().first)!=type)
 | | | now=stk.top().second,stk.pop();
 | | ly[i]=now;
 | | stk.push({i,now});
 | } }
void getrun(){
 | for(int l=1;l<=n;l++){
 | | int r=ly[l],ll=l,rr=r;
 | | if(l!=1)ll-=gl(l-1,r);
 | | if(r!=n)rr+=gr(l,r+1);
 | | if(rr-ll+1>=2*(r-l+1))run.push_back({ll,rr,r-l+1}); } }
void solve(){
 | st[n + 1] = '\0'; run.clear();
 | init();//Hash 或者 SA 的启动
 | for(int op=0;op<=1;op++){//0 正常字典序,1 反序
 | | lyndon(op); getrun(); }
 | sort(run.begin(),run.end(),[](Runs x, Runs y){
        return x == y ? x.p < y.p : (x.l != y.l ? x.l < y.l : x.r < y.r);});
 | run.erase(unique(run.begin(),run.end()),run.end()); }
```

4.13 字符串 Hash

```
static constexpr u128 inv = []() {
 | u128 ret = P;
 | for (int i = 0; i < 6; i++) ret *= 2 - ret * P;
 | return ret; }();
constexpr u128 chk = u128(-1) / P;
bool check(i128 a, i128 b) {
 | if (a < b) swap(a, b);
 | return (a - b) * inv <= chk; }
```

4.14 String Conclusions

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.

Border 和周期

如果 r 是 S 的一个 border, 则 $|S| - r$ 是 S 的一个周期.

如果 p 和 q 都是 S 的周期, 且满足 $p + q \leq |S| + gcd(p, q)$, 则 $gcd(p, q)$ 也是一个周期.

字符串匹配与 Border

若字符串 S, T 满足 $2|S| \geq |T|$, 则 S 在 T 中所有匹配位置成等差数列.

若 S 的匹配次数大于 2, 则等差数列的周期恰好等于 S 的最小周期.

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 Border

回文串长度为 t 的后缀是一个回文后缀, 等价于 t 是该串的 border. 因此回文后缀的长度也可以划分成 $O(\log |S|)$ 段.

子串最小后缀

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

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.

ZJJ: SAM 处理手法

1. 基本子串结构: CLB 搞的那玩意。
2. 正反串 SAM 的基本联系: 一个子串出现的位置将会在两个 SAM 中同时得到映照。
3. SAM 上转成数点问题。
4. 线段树合并维护 endpos 集合。
5. 树剖保证到根的链上只涉及 log 次修改和查询。(区间 border)
6. LCT 保证到根的链只修改均摊 log 个不同的颜色段。(区间本质不同子串数量)

ZJJ: 字符串常见错误

1. 字符串算法变式记得判匹配位置超出字符串的情况，例如多组数据下的双端插入回文串，后缀数组多组。2. 警惕 char 运算中 'a' 和 '␣a' 的区别。3. char kmp[]

5. Math 数学

5.1 Long Long $O(1)$ 乘, Barrett

```
LL modmul(LL a, LL b, LL M) { // skip2004, M < 63bit
 | LL ret = a * b - M * LL(1.L * a / M * b + 0.5);
 | return ret < 0 ? ret + M : ret; }
ULL modmul(ULL a, ULL b, LL M) { // orz@CF, M in 63 bit
 | ULL c = (long double)a * b / M;
 | LL ret = LL(a * b - c * M) % LL(M); // must be signed
 | return ret < 0 ? ret + M : ret; }
// use int128 instead if M > 63 bit
struct DIV {
 | ULL p, ip;
 | void init (ULL _p) { p = _p; ip = -1llu / p; }
 | int mod (ULL x) { // x < 2 ^ 64
 |  | ULL q = ULL(((u128)ip * x) >> 64);
 |  | ULL r = x - q * p;
 |  | return int(r >= p ? r - p : r);
} }; // speedup only when mod is not const
```

5.2 exgcd, 逆元

假设我们已经找到了一组解 (p_0, q_0) 满足 $ap_0 + bq_0 = \gcd(a, b)$, 那么其他的解都满足

$$p = p_0 + \frac{b}{\gcd(p,q)} \times t \qquad q = q_0 - \frac{a}{\gcd(p,q)} \times t$$

其中 t 为任意整数.

```
ll exgcd(ll a, ll b, ll &x, ll &y) {
    if (b == 0) return x = 1, y = 0, a;
    ll t = exgcd(b, a % b, y, x);
    y -= a / b * x;
    return t;
}
ll inv(ll x, ll m) {
    ll a, b;
    exgcd(x, m, a, b);
    return (a % m + m) % m;
}

/**
 * @brief 求解 ax + by = c，返回 x 的最小非负整数解和对应的 y
 * @return 返回一个 pair<ll, ll>。如果方程有解，返回 {x, y}，其中 x
 * 是最小非负整数解；如果无解，返回 {-1, -1}。
 */
std::pair<ll, ll> exgcd(ll a, ll b, ll c) {
    assert(a || b);
    // 迭代基：如果 b=0，方程为 ax=c，解为 x=c/a, y=0 (y 可为任意值，取 0)
    if (!b) return {c / a, 0};

    ll d = std::gcd(a, b);
    if (c % d) return {-1, -1};  // c 不是 gcd(a,b) 的倍数，无整数解

    // p, q 保存原始的 a, b 值
    ll x = 1, x1 = 0, p = a, q = b, k;

    // 用于计算最后 x 的模数
    ll mod = std::abs(b / d);

    // 循环结束后，求得的 x 是满足 a*x + b*y = gcd(a,b) 的一个解
    while (b) {
        k = a / b;
        x -= k * x1;
        a -= k * b;
```

```
        std::swap(x, x1);
        std::swap(a, b);
    }

    // 将 ax + by = d 的解 x 扩展为 ax + by = c 的解
    // 通解为 x' = x * (c/d) + k * (b/d)，我们要求最小非负整数解
    // 所以先对 b/d 取模
    x = x * (c / d) % mod;

    // 保证 x 是最小非负整数
    if (x < 0) x += mod;

    // 根据求得的 x，计算出对应的 y
    return {x, (c - p * x) / q};
}
```

递推逆元: $\text{inv}(i) \equiv (P - P/i) \cdot \text{inv}(P \bmod i)$

5.3 CRT 中国剩余定理

```
namespace CRT {
pair<ll, ll> exgcd(ll a, ll b, ll c) {
    assert(a || b);
    if (!b) return {c / a, 0};
    ll d = gcd(a, b);
    if (c % d) return {-1, -1};
    ll x = 1, x1 = 0, p = a, q = b, k;
    b = abs(b);
    while (b) {
        k = a / b;
        x -= k * x1;
        a -= k * b;
        swap(x, x1);
        swap(a, b);
    }
    b = abs(q / d);
    x = x * (c / d) % b;
    if (x < 0) x += b;
    return {x, (c - p * x) / q};
}
struct Q {
    ll p, r;  // 0<=r<p
    Q operator+(const Q &o) const {
        if (p == 0 || o.p == 0) return {0, 0};
        auto [x, y] = exgcd(p, -o.p, r - o.r);
        if (x == -1 && y == -1) return {0, 0};
        ll q = lcm(p, o.p);
        return {q, ((r - x * p) % q + q) % q};
    }
};
};  // namespace CRT
using CRT::Q;
```

5.4 Miller Rabin, Pollard Rho

```
mt19937 rng(123);
#define rand() LL(rng() & LLONG_MAX)
const int BASE[] = {2, 7, 61};//int(7,3e9)
//{2,325,9375,28178,450775,9780504,1795265022}LL(37)
struct miller_rabin {
bool check (const LL &M, const LL &base) {
 | LL a = M - 1;
 | while (~a & 1) a >>= 1;
 | LL w = power (base, a, M); // power should use mul
 | for (; a != M - 1 && w != 1 && w != M - 1; a <<= 1)
 |  | w = mul (w, w, M);
 | return w == M - 1 || (a & 1) == 1; }
```

```
bool solve (const LL &a) {//O((3 or 7) · log n · mul)
 | if (a < 4) return a > 1;
 | if (~a & 1) return false;
 | for (int i = 0; i < sizeof(BASE)/4 && BASE[i] < a; ++i)
 |  | if (!check (a, BASE[i])) return false;
 | return true; } };
miller_rabin is_prime;
LL get_factor (LL a, LL seed) {//O(n^{1/4} · log n · mul)
 | LL x = rand () % (a - 1) + 1, y = x;
 | for (int head = 1, tail = 2; ; ) {
 |  | x = mul (x, x, a); x = (x + seed) % a;
 |  | if (x == y) return a;
 |  | LL ans = gcd (abs (x - y), a);
 |  | if (ans > 1 && ans < a) return ans;
 |  | if (++head == tail) { y = x; tail <<= 1; } } }
void factor (LL a, vector<LL> &d) {
 | if (a <= 1) return;
 | if (is_prime.solve (a)) d.push_back (a);
 | else {
 |  | LL f = a;
 |  | for (; f >= a; f = get_factor (a, rand() % (a - 1) + 1));
 |  | factor (a / f, d);
 |  | factor (f, d); } }
```

5.5 线性基

```
struct LB {
    const int BASE = 16;
    vector<int> d, p;
    int cnt, flag;
    LB() {
        d.assign(BASE, 0);
        p.assign(BASE, 0);
        cnt = flag = 0;
    }
    bool insert(int val, int pos) {
        for (int i = BASE - 1; i >= 0; i--) {
            if (val & (1ll << i)) {
                if (!d[i]) {
                    d[i] = val;
                    p[i] = pos;
                    return true;
                }
                if (pos > p[i]) swap(pos, p[i]), swap(val, d[i]);
                val ^= d[i];
            }
        }
        flag = 1;  // 可以异或出 0
        return false;
    }
    bool check(ll val) {  // 判断 val 是否能被异或得到
        for (int i = BASE - 1; i >= 0; i--) {
            if (val & (1ll << i)) {
                if (!d[i]) {
                    return false;
                }
                val ^= d[i];
            }
        }
        return true;
    }
    ll ask_max(ll res) {
        for (int i = BASE - 1; i >= 0; i--) {
            if ((res ^ d[i]) > res) res ^= d[i];
        }
        return res;
    }
```

```
    ll ask_min() {
        if (flag) return 0;  // 特判 0
        for (int i = 0; i <= BASE - 1; i++) {
            if (d[i]) return d[i];
        }
    }
    void rebuild() {  // guass 消元 保证只有该处有 1
        for (int i = BASE - 1; i >= 0; i--) {
            for (int j = i - 1; j >= 0; j--) {
                if (d[i] & (1ll << j)) d[i] ^= d[j];
            }
        }
        for (int i = 0; i <= BASE - 1; i++) {
            if (d[i]) p[cnt++] = d[i];
        }
    }
    ll kth(ll k) {      // 查询能被异或得到的第 k 小值, 如不存在则返回 -1
        if (flag) k--;  // 特判 0, 如果不需要 0, 直接删去
        if (!k) return 0;
        ll res = 0;
        if (k >= (1ll << cnt)) return -1;
        for (int i = BASE - 1; i >= 0; i--) {
            if (k & (1LL << i)) res ^= p[i];
        }
        return res;
    }
};
```

5.6 扩展卢卡斯

```
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), 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);}
```

5.7 阶乘取模

```
// n! mod p^q Time : O(pq^2 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}; }
```

5.8 类欧几里得直线下格点统计

```
// ∑_{i=0}^{n-1} ⌊(a+bi)/m⌋, n,m,a,b > 0
LL solve(LL n, LL a, LL b, LL m){
 | if (b == 0) return n * (a / m);
 | if (a >= m) return n * (a / m) + solve(n, a % m, b, m);
 | if (b >= m) return (n-1)*n/2*(b/m) + solve(n,a,b%m,m);
 | return solve((a + b * n) / m, (a + b * n) % m, m, b); }
```

5.9 万能欧几里德

```
Val work(LL P, LL R, LL Q, LL n, Val VU, Val VR) {
//(Px+R)/Q, 1<=x<=i, 经过整点先 U 再 R
 | if(!(((i128)n * P + R) / Q)) return ksm(VR, n);
 | if(P>=Q) return work(P%Q,R,Q,n, VU, ksm(VU, P/Q) * VR);
 | Val res; swap(VU,VR);
 | res = ksm(VU, (Q-R-1)/P)*VR;
 | LL m = ((i128)n * P + R) / Q;
 | res = res * work(Q, (Q-R-1)%P, P, m-1, VU, VR);
 | return res * ksm(VU, n - ((i128)m*Q - R - 1) / P); }
```

5.10 平方剩余

```
// 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; }
```

5.11 线性同余不等式

```
// Find the minimal non-negtive solutions for l ≤ d · x mod m ≤ r
// 0 ≤ d, l, r < m; l ≤ 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; }
```

5.12 原根

定义 使得 $a^x \bmod m = 1$ 的最小的 x, 记作 $\delta_m(a)$. 若 $a \equiv g^s \mod m$, 其中 g 为 m 的一个原根. 则虽然 s 随 g 的不同取值有所不同, 但是必然满足 $\delta_m(a) = \gcd(s, \varphi(m))$.

性质 $\delta_m(a^k) = \frac{\delta_m(a)}{\gcd(\delta_m(a),k)}$

k 次剩余 给定方程 $x^k \equiv a \mod m$, 求所有解. 若 k 与 $\varphi(m)$ 互质, 则可以直接求出 k 对 $\varphi(m)$ 的逆元. 否则, 将 k 拆成两部分, $k = uv$, 其中 $u \perp \varphi(m)$, $v | \varphi(m)$, 先求 $x^v \equiv a \mod m$, 则 $ans = x^{u^{-1}}$. 下面讨论 $k | \varphi(m)$ 的问题. 任取一原根 g, 对两侧取离散对数, 设 $x = g^s$, $a = g^t$, 其中 t 可以用 BSGS 求出, 则问题转化为求出所有的 s 满足 $ks \equiv t \mod \varphi(m)$, exgcd 即可求解, 显然有解的条件是 $k | \delta_m(a)$.

5.13 多项式

```
using ll = long long;
const int mod = 998244353;

int qmi(ll a, ll k) {
    ll res = 1;
```

```cpp
    while (k) {
        if (k & 1) res = res * a % mod;
        k >>= 1;
        a = a * a % mod;
    }
    return res;
}

namespace NTT {
const int g = 3;
vector<int> Omega(int L) {
    int wn = qmi(g, mod / L);
    vector<int> w(L);
    w[L >> 1] = 1;
    for (int i = L / 2 + 1; i <= L - 1; i++) w[i] = 1ll * w[i - 1] * wn % mod;
    for (int i = L / 2 - 1; i; i--) w[i] = w[i << 1];
    return w;
}
auto W = Omega(1 << 21);  // NOLINT
void DIF(int* a, int n) {
    for (int k = n >> 1; k; k >>= 1)
        for (int i = 0, y; i < n; i += k << 1)
            for (int j = 0; j < k; j++)
                y = a[i + j + k],
                a[i + j + k] = 1ll * (a[i + j] - y + mod) * W[k + j] % mod,
                a[i + j] = (y + a[i + j]) % mod;
}
void IDIT(int* a, int n) {
    for (int k = 1; k < n; k <<= 1)
        for (int i = 0, x, y; i < n; i += k << 1)
            for (int j = 0; j < k; j++)
                x = a[i + j], y = (1ll * a[i + j + k] * W[k + j]) % mod,
                a[i + j + k] = x - y < 0 ? x - y + mod : x - y,
                a[i + j] = (y + a[i + j]) % mod;
    int Inv = mod - (mod - 1) / n;
    for (int i = 0; i < n; i++) a[i] = 1ll * a[i] * Inv % mod;
    reverse(a + 1, a + n);
}
// 用普通数组实现的 NTT 蝶形变化
}  // namespace NTT
namespace Polynomial {
using Poly = std::vector<int>;

// mul/div int
Poly& operator*=(Poly& a, int b) {
    for (auto& x : a) x = 1ll * x * b % mod;
    return a;
}
Poly operator*(Poly a, int b) { return a *= b; }
Poly operator*(int a, Poly b) { return b * a; }
Poly& operator/=(Poly& a, int b) { return a *= qmi(b, mod - 2); }
Poly operator/(Poly a, int b) { return a /= b; }

// Poly add/sub
Poly& operator+=(Poly& a, Poly b) {
    a.resize(max(a.size(), b.size()));
    for (int i = 0; i < b.size(); i++) a[i] = (a[i] + b[i]) % mod;
    return a;
}
Poly operator+(Poly a, Poly b) { return a += b; }
Poly& operator-=(Poly& a, Poly b) {
    a.resize(max(a.size(), b.size()));
    for (int i = 0; i < b.size(); i++) a[i] = (a[i] - b[i] + mod) % mod;
    return a;
}
Poly operator-(Poly a, Poly b) { return a -= b; }

```

```cpp
// Poly mul
void DFT(Poly& a) { NTT::DIF(a.data(), a.size()); }
void IDFT(Poly& a) { NTT::IDIT(a.data(), a.size()); }
int norm(int n) {
    return 1 << (32 - __builtin_clz(n - 1));
}  // 返回大于等于 n 的最小 2 的整数次幂
void norm(Poly& a) {
    if (!a.empty()) a.resize(norm(a.size()), 0);
}  // 剩余的用 0 填
Poly& dot(Poly& a, Poly& b) {
    for (int i = 0; i < a.size(); i++) a[i] = 1ll * a[i] * b[i] % mod;
    return a;
}
Poly operator*(Poly a, Poly b) {
    int n = a.size() + b.size() - 1, L = norm(n);
    if (a.size() <= 8 || b.size() <= 8) {
        Poly c(n);
        for (int i = 0; i < a.size(); i++)
            for (int j = 0; j < b.size(); j++)
                c[i + j] = (c[i + j] + (ll)a[i] * b[j]) % mod;
        return c;
    }
    a.resize(L), b.resize(L);
    DFT(a), DFT(b), dot(a, b), IDFT(a);
    return a.resize(n), a;
}

// Poly inv 利用牛顿迭代递归实现的乘法逆
Poly Inv2k(Poly a) {  // a.size() = 2^k
    int n = a.size(), m = n >> 1;
    if (n == 1) return {qmi(a[0], mod - 2)};
    Poly b = Inv2k(Poly(a.begin(), a.begin() + m)), c = b;
    b.resize(n), DFT(a), DFT(b), dot(a, b), IDFT(a);
    for (int i = 0; i < n; i++) a[i] = i < m ? 0 : mod - a[i];
    DFT(a), dot(a, b), IDFT(a);
    return move(c.begin(), c.end(), a.begin()), a;
}
Poly Inv(Poly a) {
    int n = a.size();
    norm(a), a = Inv2k(a);
    return a.resize(n), a;
}

// Poly div/mod
Poly operator/(Poly a, Poly b) {
    int k = a.size() - b.size() + 1;
    if (k < 0) return {0};
    reverse(a.begin(), a.end());
    reverse(b.begin(), b.end());
    b.resize(k), a = a * Inv(b);
    a.resize(k), reverse(a.begin(), a.end());
    return a;
}
pair<Poly, Poly> operator%(Poly a, const Poly& b) {
    Poly c = a / b;
    a -= b * c, a.resize(b.size() - 1);
    return {c, a};
}

// 求导和积分
Poly deriv(Poly a) {
    for (int i = 1; i < a.size(); i++) a[i - 1] = 1ll * i * a[i] % mod;
    return a.pop_back(), a;
}

// 此处如果预处理逆元的话大概可以优化
Poly integ(Poly a) {
```

```
    a.push_back(0);
    for (int i = a.size() - 1; i; i--)
        a[i] = 1ll * a[i - 1] * qmi(i, mod - 2) % mod;
    return a[0] = 0, a;
}

// Poly 要求 a[0] = 1
Poly Ln(Poly a) {
    int n = a.size();
    a = deriv(a) * Inv(a);
    return a.resize(n - 1), integ(a);
}

// Poly exp
Poly Exp(Poly a) {
    int n = a.size(), k = norm(n);
    Poly b = {1}, c, d;
    a.resize(k);
    for (int L = 2; L <= k; L <<= 1) {
        d = b, b.resize(L), c = Ln(b), c.resize(L);
        for (int i = 0; i < L; i++)
            c[i] = a[i] - c[i] + (a[i] < c[i] ? mod : 0);
        c[0] = (c[0] + 1) % mod;
        DFT(b), DFT(c), dot(b, c), IDFT(b);
        move(d.begin(), d.end(), b.begin());
    }
    return b.resize(n), b;
}

Poly Pow(Poly& a, int b) { return Exp(Ln(a) * b); }  // a[0] = 1, 多项式快速幂

Poly Sqrt(Poly a) {
    int n = a.size(), k = norm(n);

    Poly b = {(int)(new Cipolla)->sqrt(a[0], mod)}, c;
    a.resize(k * 2, 0);
    for (int L = 2; L <= k; L <<= 1) {
        b.resize(2 * L, 0), c = Poly(a.begin(), a.begin() + L) * Inv(b);
        for (int i = 0; i <= 2 * L; i++)
            b[i] = 1ll * (b[i] + c[i]) * (mod + 1 >> 1) % mod;
    }
    return b.resize(n), b;
}

}  // namespace Polynomial
using namespace Polynomial;
```

5.14 FFT

```
using cp = complex<double>; const double PI = acos(-1.0);
vector<cp> omega[25]; // 单位根
// n 是 DFT 的最大长度，例如如果最多有两个长为 m 的多项式相乘，
// 或者求逆的长度为 m，那么 n 需要 >= 2m
void fft_init(int n) { // n = 2^k
 | for (int k = 2, d = 0; k <= n; k *= 2, d++) {
 |  | omega[d].resize(k + 1);
 |  | for (int i = 0; i <= k; i++) // polar 是用模和辐角求复数
 |  |  | omega[d][i] = polar(1.0, 2 * PI * i / k); } }
void fft(cp* a, int n, int t) {
 | for (int i = 1, j = 0; i < n - 1; i++) {
 |  | int k = n; do j ^= (k >>= 1); while (j < k);
 |  | if (i < j) swap(a[i], a[j]); }
 | for (int k = 1, d = 0; k < n; k *= 2, d++)
 |  | for (int i = 0; i < n; i += k * 2)
 |  |  | for (int j = 0; j < k; j++) {
 |  |  |  | cp w = omega[d][t > 0 ? j : k * 2 - j];
 |  |  |  | cp u = a[i + j], v = w * a[i + j + k];
```

```
    |   |   |   | a[i + j] = u + v; a[i + j + k] = u - v; }
    | if (t < 0) for (int i = 0; i < n; i++) a[i] /= n; }
```

5.15 NTT

```
vector<int> omega[25]; // 单位根
// n 是 DFT 的最大长度，例如如果最多有两个长为 m 的多项式相乘，
// 或者求逆的长度为 m，那么 n 需要 >= 2m
void ntt_init(int n) { // n = 2^k
  | for (int k = 2, d = 0; k <= n; k *= 2, d++) {
  |   | omega[d].resize(k + 1);
  |   | int wn = qpow(3, (p - 1) / k), tmp = 1;
  |   | for (int i = 0; i <= k; i++) { omega[d][i] = tmp;
  |   |   | tmp = (LL)tmp * wn % p; } } }
// 传入的数必须是 [0, p) 范围内，不能有负的
// 否则把 d == 16 改成 d % 8 == 0 之类，多取几次模
void ntt(int *c, int n, int tp) {
  | static ULL a[N];
  | for (int i = 0; i < n; i++) a[i] = c[i];
  | for (int i = 1, j = 0; i < n - 1; i++) {
  |   | int k = n; do j ^= (k >>= 1); while (j < k);
  |   | if (i < j) swap(a[i], a[j]); }
  | for (int k = 1, d = 0; k < n; k *= 2, d++) {
  |   | if (d == 16) for (int i = 0; i < n; i++) a[i] %= p;
  |   | for (int i = 0; i < n; i += k * 2)
  |   |   | for (int j = 0; j < k; j++) {
  |   |   |   | int w = omega[d][tp > 0 ? j : k * 2 - j];
  |   |   |   | ULL u = a[i + j], v = w * a[i + j + k] % p;
  |   |   |   | a[i + j] = u + v;
  |   |   |   | a[i + j + k] = u - v + p; } }
  | if (tp>0) {for (int i = 0; i < n; i++) c[i] = a[i] % p;}
  | else { int inv = qpow(n, p - 2);
  |   | for (int i = 0; i < n; i++) c[i] = a[i] * inv % p;}}
```

5.16 MTT 任意模数卷积

```
void dft(cp* a, cp* b, int n) { static cp c[MAXN];
  | for (int i = 0; i < n; i++)
  |   | c[i] = cp(a[i].real(), b[i].real());
  | fft(c, n, 1);
  | for (int i = 0; i < n; i++) { int j = (n - i) & (n - 1);
  |   | a[i] = (c[i] + conj(c[j])) * 0.5;
  |   | b[i] = (c[i] - conj(c[j])) * -0.5i; } }
void idft(cp* a, cp* b, int n) { static cp c[MAXN];
  | for (int i = 0; i < n; i++) c[i] = a[i] + 1i * b[i];
  | fft(c, n, -1);
  | for (int i = 0; i < n; i++) {
  |   | a[i] = c[i].real(); b[i] = c[i].imag(); } }
vector<int> multiply(const vector<int>& u,
  |   | const vector<int>& v, int mod) { // 任意模数卷积
  | static cp a[2][MAXN], b[2][MAXN], c[3][MAXN];
  | int base = ceil(sqrt(mod));
  | int n = (int)u.size(), m = (int)v.size();
  | int fft_n = 1; while (fft_n < n + m - 1) fft_n *= 2;
  | for (int i = 0; i < 2; i++) {
  |   | fill(a[i], a[i] + fft_n, 0);
  |   | fill(b[i], b[i] + fft_n, 0); }
  | for (int i = 0; i < 3; i++)
  |   | fill(c[i], c[i] + fft_n, 0);
  | for (int i = 0; i < n; i++) { // 一定要取模!
  |   | a[0][i] = (u[i] % mod) % base;
  |   | a[1][i] = (u[i] % mod) / base; }
  | for (int i = 0; i < m; i++) { // 一定要取模!
  |   | b[0][i] = (v[i] % mod) % base;
  |   | b[1][i] = (v[i] % mod) / base; }
  | dft(a[0], a[1], fft_n); dft(b[0], b[1], fft_n);
```

```
  | for (int i = 0; i < fft_n; i++) {
  |   | c[0][i] = a[0][i] * b[0][i];
  |   | c[1][i] = a[0][i] * b[1][i] + a[1][i] * b[0][i];
  |   | c[2][i] = a[1][i] * b[1][i]; }
  | fft(c[1], fft_n, -1); idft(c[0], c[2], fft_n);
  | int base2 = base * base % mod;
  | vector<int> ans(n + m - 1);
  | for (int i = 0; i < n + m - 1; i++)
  |   | ans[i] = ((LL)(c[0][i].real() + 0.5) +
  |   |   | (LL)(c[1][i].real() + 0.5) % mod * base +
  |   |   | (LL)(c[2][i].real() + 0.5) % mod * base2) % mod;
  | return ans; }
```

5.17 多项式运算

5.17.1 多项式求逆 开根 ln exp

```
using poly = vector<int>; // 用到 poly 的部分补成 2 ^ k
poly poly_calc(const poly& u, const poly& v, // 长度要相同
  | function<int(int, int)> op) { // 返回长度是两倍
  | static int a[MAXN], b[MAXN], c[MAXN];
  | int n = (int)u.size();
  | memcpy(a, u.data(), sizeof(int) * n);
  | fill(a + n, a + n * 2, 0);
  | memcpy(b, v.data(), sizeof(int) * n);
  | fill(b + n, b + n * 2, 0);
  | ntt(a, n * 2, 1); ntt(b, n * 2, 1);
  | for (int i = 0; i < n * 2; i++) c[i] = op(a[i], b[i]);
  | ntt(c, n * 2, -1); return poly(c, c + n * 2); }
poly poly_mul(const poly& u, const poly& v) { // 乘法
  | return poly_calc(u, v, [](int a, int b)
  |   | { return (LL)a * b % p; }); } // 返回长度是两倍
poly poly_inv(const poly& a) { // 求逆，返回长度不变
  | poly c{qpow(a[0], p - 2)}; // 常数项一般都是 1
  | for (int k = 2; k <= (int)a.size(); k *= 2) {
  |   | c.resize(k); poly b(a.begin(), a.begin() + k);
  |   | c = poly_calc(b, c, [](int bi, int ci) {
  |   |   | return ((2 - (LL)bi * ci) % p + p) * ci % p; });
  |   | memset(c.data() + k, 0, sizeof(int) * k); }
  | c.resize(a.size()); return c; }
poly poly_sqrt(const poly& a) { // 开根，返回长度不变
  | poly c{1}; // 常数项不是 1 的话要写二次剩余
  | for (int k = 2; k <= (int)a.size(); k *= 2) {
  |   | c.resize(k); poly b(a.begin(), a.begin() + k);
  |   | b = poly_mul(b, poly_inv(c));
  |   | for (int i = 0; i < k; i++) // inv_2 是 2 的逆元
  |   |   | c[i] = (LL)(c[i] + b[i]) * inv_2 % p; }
  | c.resize(a.size()); return c; }
poly poly_derivative(const poly& a) { poly c(a.size());
  | for (int i = 1; i < (int)a.size(); i++) // 求导
  |   | c[i - 1] = (LL)a[i] * i % p; return c; }
poly poly_integrate(const poly& a) { poly c(a.size());
  | for (int i = 1; i < (int)a.size(); i++) // 不定积分
  |   | c[i] = (LL)a[i - 1] * inv[i] % p; return c; }
poly poly_ln(const poly& a) { // ln，常数项非 0，返回长度不变
  | auto c = poly_mul(poly_derivative(a), poly_inv(a));
  | c.resize(a.size()); return poly_integrate(c); }
// exp，常数项必须是 0，返回长度不变
// 常数很大并且总代码很长，一般可以改用分治 FFT
// 依据：设 G(x) = exp F(x)，则 g_i = 1/i * sum_{k=1}^{i-1} g_{i-k} k f_k
poly poly_exp(const poly& a) { poly c{1};
  | for (int k = 2; k <= (int)a.size(); k *= 2) {
  |   | c.resize(k); auto b = poly_ln(c);
  |   | for (int i = 0; i < k; i++)
  |   |   | b[i] = (a[i] - b[i] + p) % p;
  |   | (++b[0]) %= p; c = poly_mul(b, c);
  |   | memset(c.data() + k, 0, sizeof(int) * k); }
```

```
51 | c.resize(a.size()); return c; }
```

5.17.2 多项式除法 取模

需要抄求逆。

```
poly poly_auto_mul(poly a, poly b) { // 自动判断长度的乘法
 | int res_len = (int)a.size() + (int)b.size() - 1;
 | int ntt_n = 1; while (ntt_n < res_len) ntt_n *= 2;
 | a.resize(ntt_n); b.resize(ntt_n);
 | ntt(a.data(), ntt_n, 1); ntt(b.data(), ntt_n, 1);
 | for (int i = 0; i < ntt_n; i++)
 | | a[i] = (LL)a[i] * b[i] % p;
 | ntt(a.data(), ntt_n, -1); a.resize(res_len); return a; }
// 多项式除法，a 和 b 长度可以任意
// 商的长度是 n - m + 1，余数的长度是 m - 1
poly poly_div(const poly& a, const poly& b) {
 | int n = (int)a.size(), m = (int)b.size();
 | if (n < m) return {};
 | int ntt_n = 1; while (ntt_n < n - m + 1) ntt_n *= 2;
 | poly f(ntt_n), g(ntt_n);
 | for (int i = 0; i < n - m + 1; i++) f[i] = a[n - i - 1];
 | for (int i = 0; i < m && i < n - m + 1; i++)
 | | g[i] = b[m - i - 1];
 | auto g_inv = poly_inv(g);
 | fill(g_inv.begin() + n - m + 1, g_inv.end(), 0);
 | auto c = poly_mul(f, g_inv); c.resize(n - m + 1);
 | reverse(c.begin(), c.end()); return c; }
// 多项式取模，a 和 b 长度可以任意，返回 (余数，商)
pair<poly, poly> poly_mod(const poly& a, const poly& b) {
 | int n = (int)a.size(), m = (int)b.size();
 | if (n < m) return {a, {}};
 | auto d = poly_div(a, b); auto c = poly_auto_mul(b, d);
 | poly r(m - 1);
 | for (int i = 0; i < m - 1; i++)
 | | r[i] = (a[i] - c[i] + p) % p;
 | return {r, d}; }
```

5.17.3 多点求值

需要抄取模。

```
struct poly_eval { poly f; vector<int> x; // 函数和询问点
 | vector<poly> gs; vector<int> ans; // gs 是预处理数组
 | poly_eval(poly f, vector<int> x) : f(f), x(x) {}
 | void pretreat(int l, int r, int o) { poly& g = gs[o];
 | | if (l == r) { g = poly{p - x[l], 1}; return; }
 | | int mid = (l + r) / 2; pretreat(l, mid, o * 2);
 | | pretreat(mid + 1, r, o * 2 + 1);
 | | if (o > 1)
 | | | g = poly_auto_mul(gs[o * 2], gs[o * 2 + 1]); }
 | void solve(int l, int r, int o, const poly& f) {
 | | if (l == r) { ans[l] = f[0]; return; }
 | | int mid = (l + r) / 2;
 | | solve(l, mid, o * 2, poly_mod(f, gs[o * 2]).first);
 | | solve(mid + 1, r, o * 2 + 1,
 | | | poly_mod(f, gs[o * 2 + 1]).first); }
 | vector<int> operator() () { // 包装好的接口
 | | int n = (int)f.size(), m = (int)x.size();
 | | if (m <= n) x.resize(m = n + 1);
 | | else if (n < m - 1) f.resize(n = m - 1);
 | | int bit_ceil = 1; while (bit_ceil < m) bit_ceil *= 2;
 | | ntt_init(bit_ceil * 2); // 注意这里 ntt_init 过了
 | | gs.resize(2 * bit_ceil + 1); pretreat(0, m - 1, 1);
 | | ans.resize(m); solve(0, m - 1, 1, f); return ans;} };
```

5.17.4 插值

牛顿插值 实现时可以用 k 次差分替代右边的式子，也可以卷积。

$$f(n)=\sum_{i=0}^{k}\binom{n}{i}r_i \iff r_i=\sum_{j=0}^{i}(-1)^{i-j}\binom{i}{j}f(j)$$

拉格朗日插值

$$f(x)=\sum_i f(x_i)\prod_{j\neq i}\frac{x-x_j}{x_i-x_j}$$

5.18 线性递推

$O(k^2\log n)$

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

$O(k\log k\log n)$ - **Bostan-Mori**

```
int bostan_mori(int k, poly a, poly b) {
 | int n = (int)a.size(); while (k) { poly c = b;
 | | for (int i = 1; i < n; i += 2) c[i] = (p - c[i]) % p;
 | | a = poly_mul(a, c); b = poly_mul(b, c);
 | | for (int i = 0; i < n; i++) {
 | | | a[i] = a[i * 2 + k % 2]; b[i] = b[i * 2]; }
 | | a.resize(n); b.resize(n); k /= 2; }
 | return (LL)a[0] * qpow(b[0], p - 2) % p; }
// a_n = sum_{i=1}^{m} f_i a_{n-i}  (f_0 = 0), f.size() = a.size() + 1
int linear_recurrance(int n, poly f, poly a) {
 | int m = (int)a.size(), ntt_n = 1;
 | while (ntt_n <= m) ntt_n *= 2; ntt_init(ntt_n * 2);
 | f.resize(ntt_n); a.resize(ntt_n); f[0] = 1;
 | for (int i = 1; i <= m; i++) f[i] = (p - f[i]) % p;
 | a = poly_mul(a, f); a.resize(ntt_n);
 | fill(a.data() + m, a.data() + ntt_n, 0);
 | return bostan_mori(n, a, f); }
```

$O(k \log k \log n)$ - **多项式取模**

需要抄前面的多项式取模。预处理 $O(k \log k \log n)$，固定 n 和系数只改变初始值的话，询问一次 $O(k)$。注意只询问一次的话不如 Bostan-Mori 快。

```
poly poly_power_mod(LL k, const poly& m) { // x^k mod m
 | poly ans{1}, a{0, 1}; while (k) { if (k & 1)
 | | | ans = poly_mod(poly_auto_mul(ans, a), m).first;
 | | a = poly_mod(poly_auto_mul(a, a), m).first; k /= 2; }
 | return ans; }
// a_n = sum_{i=1}^{m} c_i a_{n-i}  (c_0 = 0)
struct linear_recurrence { poly f; // f 是预处理结果
 | linear_recurrence(const poly& c, LL n) {
 | | assert(c[0] == 0); // c[0] 是没有用的
 | | int m = (int)c.size() - 1;
 | | int ntt_n = 1; while (ntt_n < m * 2) ntt_n *= 2;
 | | ntt_init(ntt_n); // 图省事就直接 ntt_init(1 << 18)
 | | poly t(m + 1); t[m] = 1;
 | | for (int i = 0; i < m; i++)t[i] = (p - c[m - i]) % p;
 | | f = poly_power_mod(n, t); }
 | int operator()(const vector<int>& a) { // 0~m-1 项初始值
 | | assert(a.size() == f.size()); int ans = 0;
 | | for (int i = 0; i < (int)a.size(); i++)
 | | | ans = (ans + (LL)f[i] * a[i]) % p;
 | | return ans; } };
```

5.19 Berlekamp-Massey 最小多项式

如果要求出一个次数为 k 的递推式，则输入的数列需要至少有 $2k$ 项。

返回的内容满足 $\sum_{j=0}^{m-1} a_{i-j}c_j = 0$，并且 $c_0 = 1$。

如果不加最后的处理的话，代码返回的结果会变成 $a_i = \sum_{j=0}^{m-1} c_{j-1}a_{i-j}$，有时候这样会方便接着跑递推，需要的话就删掉最后的处理。

```
vector<int> berlekamp_massey(const vector<int> &a) {
 | vector<int> v, last; // v is the answer, 0-based
 | int k = -1, delta = 0;
 | for (int i = 0; i < (int)a.size(); i++) { int tmp = 0;
 | | for (int j = 0; j < (int)v.size(); j++)
 | | | tmp = (tmp + (LL)a[i - j - 1] * v[j]) % p;
 | | if (a[i] == tmp) continue;
 | | if (k < 0) { k = i; delta = (a[i] - tmp + p) % p;
 | | | v = vector<int>(i + 1); continue; }
 | | vector<int> u = v;
 | | int val = (LL)(a[i] - tmp + p) *
 | | | qpow(delta, p - 2) % p;
 | | if (v.size() < last.size() + i - k)
 | | | v.resize(last.size() + i - k);
 | | (v[i - k - 1] += val) %= p;
 | | for (int j = 0; j < (int)last.size(); j++) {
 | | | v[i - k + j] = (v[i - k + j] -
 | | | | (LL)val * last[j]) % p;
 | | | if (v[i - k + j] < 0) v[i - k + j] += p; }
 | | if ((int)u.size() - i < (int)last.size() - k) {
 | | | last = u; k = i; delta = a[i] - tmp;
 | | | if (delta < 0) delta += p; } }
 | for (auto &x : v) x = (p - x) % p;
 | v.insert(v.begin(), 1); //一般是需要最小递推式的，处理一下
 | return v; } // ∀i, sum_{j=0}^{m} a_{i-j} v_j = 0
```

如果要求向量序列的递推式，就把每位乘一个随机权值 (或者说是乘一个随机行向量 v^T) 变成求数列递推式即可。如果是矩阵序列的话就随机一个行向量 u^T 和列向量 v，然后把矩阵变成 $u^T Av$ 的数列。

优化矩阵快速幂 DP 假设 f_i 有 n 维，先暴力求出 $f_{0\sim 2n-1}$，然后跑 Berlekamp-Massey，最后调用快速线性递推即可。

求矩阵最小多项式 矩阵 A 的最小多项式是次数最小的并且 $f(A)=0$ 的多项式 f。实际上最小多项式就是 $\{A^i\}$ 的最小递推式，所以直接调用 Berlekamp-Massey 就好了，显然它的次数不超过 n。

瓶颈在于求出 A^i，实际上我们只要处理 $A^i v$ 就行了，每次对向量做递推。

求稀疏矩阵的行列式 如果能求出特征多项式，则常数项乘上 $(-1)^n$ 就是行列式，但是最小多项式不一定就是特征多项式。

把 A 乘上一个随机对角阵 B，则 AB 的最小多项式有很大概率就是特征多项式，最后再除掉 $\det B$ 就行了。

求稀疏矩阵的秩 设 A 是一个 $n\times m$ 的矩阵, 首先随机一个 $n\times n$ 的对角阵 P 和一个 $m\times m$ 的对角阵 Q, 然后计算 $QAPA^TQ$ 的最小多项式即可。

实际上不用计算这个矩阵，因为求最小多项式时要用它乘一个向量，我们依次把这几个矩阵乘到向量里就行了。答案就是最小多项式除掉所有 x 因子后剩下的次数。

解稀疏方程组 $Ax=b$，其中 A 是一个 $n\times n$ 的**满秩**稀疏矩阵，b 和 x 是 $1\times n$ 的**列**向量，A,b 已知，需要解出 x。

做法: 显然 $x=A^{-1}b$. 如果我们能求出 $\{A^ib\}(i\geq 0)$ 的最小递推式 $\{r_{0\dots m-1}\}(m\leq n)$，那么就有结论

$$A^{-1}b=-\frac{1}{r_{m-1}}\sum_{i=0}^{m-2}A^ibr_{m-2-i}$$

因为 A 是稀疏矩阵，直接按定义递推出 $b\dots A^{2n-1}b$ 即可。

```
vector<int> solve_sparse_equations(const vector<tuple<int, int, int> > &A, const vector<int> &b) {
 | int n = (int)b.size(); // 0-based
 | vector<vector<int> > f({b});
 | for (int i = 1; i < 2 * n; i++) {
 | | vector<int> v(n); auto &u = f.back();
 | | for (auto [x, y, z] : A) // [x, y, value]
 | | | v[x] = (v[x] + (long long)u[y] * z) % p;
 | | f.push_back(v); }
 | vector<int> w(n); mt19937 gen;
 | for (auto &x : w)
 | | x = uniform_int_distribution<int>(1, p - 1)(gen);
 | vector<int> a(2 * n);
 | for (int i = 0; i < 2 * n; i++)
 | | for (int j = 0; j < n; j++)
 | | | a[i] = (a[i] + (long long)f[i][j] * w[j]) % p;
 | auto c = berlekamp_massey(a); int m = (int)c.size();
 | vector<int> ans(n);
 | for (int i = 0; i < m - 1; i++)
 | | for (int j = 0; j < n; j++)
 | | | ans[j] = (ans[j] +
 | | | | (long long)c[m - 2 - i] * f[i][j]) % p;
 | int inv = qpow(p - c[m - 1], p - 2);
 | for (int i = 0; i < n; i++)
 | | ans[i] = (long long)ans[i] * inv % p;
 | return ans; }
```

5.20 FWT

```
/*
And: (1,1 / 0,1) (1,-1 / 0,1) Or: (1,0 / 1,1) (1,0 / -1,1) Xor: (1,1 / 1,-1) (0.5,0.5 / 0.5,-0.5)
IFWT 的矩阵时 FWT 的逆，对于任意运算 ⊕，满足 FWT 的矩阵需要:
C[i][j] × C[i][k] = C[i][j ⊕ k]
对于不存在 FWT 矩阵的运算: 通过映射 01 变成另外一个可行的运算。*/
const LL XOR[2][2] = {{1, 1}, {1, M-1}};
const LL i2 = (M+1)/2, iXOR[2][2] = {{i2, i2}, {i2, M-i2}};
void FWT(LL f[], const LL C[2][2], int n) {
  for (int t = 1; t < n; t <<= 1) {
    for (int l = 0; l < n; l += t + t) {
      for (int i = 0; i < t; i++) {
        LL x = f[l + i], y = f[l + t + i];
        f[l + i]     = (C[0][0] * x + C[0][1] * y) % M;
        f[l + t + i] = (C[1][0] * x + C[1][1] * y) % M;
```

```
      } } } }
```

5.21 K 进制 FWT

```
// n : power of k, omega[i] : (primitive kth root) ^ i
void fwt(int* a, int k, int type) {
  static int tmp[K];
  for (int i = 1; i < n; i *= k)
    for (int j = 0, len = i * k; j < n; j += len)
      for (int low = 0; low < i; low++) {
        for (int t = 0; t < k; t++)
          tmp[t] = a[j + t * i + low];
        for (int t = 0; t < k; t++){
          int x = j + t * i + low;
          a[x] = 0;
          for (int y = 0; y < k; y++)
            a[x] = int(a[x] + 1ll * tmp[y] * omega[(k + type) * t * y % k] % MOD);
        }
      }
  if (type == -1)
    for (int i = 0, invn = inv(n); i < n; i++)
      a[i] = int(1ll * a[i] * invn % MOD); }
```

5.22 Simplex 单纯形

```
const LD eps = 1e-9, INF = 1e9; const int N = 105;
namespace Simplex {
int n, m, id[N], tp[N]; LD a[N][N];
void pivot(int r, int c) {
 | swap(id[r + n], id[c]);
 | LD t = -a[r][c]; a[r][c] = -1;
 | for (int i = 0; i <= n; i++) a[r][i] /= t;
 | for (int i = 0; i <= m; i++) if (a[i][c] && r != i) {
 |  | t = a[i][c]; a[i][c] = 0;
 |  | for (int j = 0; j <= n; j++) a[i][j] += t*a[r][j];}}
bool solve() {
 | for (int i = 1; i <= n; i++) id[i] = i;
 | for ( ; ; ) {
 |  | int i = 0, j = 0; LD w = -eps;
 |  | for (int k = 1; k <= m; k++)
 |  |  | if (a[k][0] < w || (a[k][0] < -eps && rand() & 1))
 |  |  |  | w = a[i = k][0];
 |  | if (!i) break;
 |  | for (int k = 1; k <= n; k++)
 |  |  | if (a[i][k] > eps) {j = k; break;}
 |  | if (!j) { printf("Infeasible"); return 0;}
 |  | pivot(i, j);}
 | for ( ; ; ) {
 |  | int i = 0, j = 0; LD w = eps, t;
 |  | for (int k = 1; k <= n; k++)
 |  |  | if (a[0][k] > w) w = a[0][j = k];
 |  | if (!j) break;
 |  | w = INF;
 |  | for (int k = 1; k <= m; k++)
 |  |  | if (a[k][j] < -eps && (t = -a[k][0]/a[k][j]) < w)
 |  |  |  | w = t, i = k;
 |  | if (!i) { printf("Unbounded"); return 0;}
 |  | pivot(i, j);}
 | return 1;}
LD ans() {return a[0][0];}
void output() {
 | for (int i = n + 1; i <= n + m; i++) tp[id[i]] = i - n;
 | for (int i = 1; i <=n; i++) printf("%.9lf ", tp[i] ? a[tp[i]][0] : 0);}
}using namespace Simplex;
int main() { int K; read(n); read(m); read(K);
for (int i = 1; i <= n; i++) {LD x; scanf("%lf", &x); a[0][i] = x;}
```

```
for (int i = 1; i <= m; i++) {LD x;
 | for (int j = 1; j <= n; j++) scanf("%lf", &x), a[i][j] = -x;
 | scanf("%lf", &x); a[i][0] = x;}
if (solve()) { printf("%.9lf\n", (LD)ans()); if (K) output();}}
// 标准型: maximize c^T x, subject to Ax ≤ b and x ≥ 0
// 对偶型: minimize b^T y, subject to A^T x ≥ c and y ≥ 0
```

5.23 高斯消元最小范数解

```
typedef vector <LD> vec; /* sum a[i][0..d] = 0 */
pair<vec,vector<vec>> gauss(vector<vec> &a, int n, int d) {
 | vector <int> pivot(d, -1);
 | for (int i = 0, o = 0; i < d; i++) {
 |  | int j = o; while (j < n && abs(a[j][i]) < eps) j++;
 |  | if (j == n) continue;
 |  | swap(a[j], a[o]); LD w = a[o][i];
 |  | for (int k = 0; k <= d; k++) a[o][k] /= w;
 |  | for (int x = 0; x < n; x++)
 |  |  | if (x != o && abs(a[x][i]) > eps) {
 |  |  |  | w = a[x][i];
 |  |  |  | for (int k = 0; k <= d; k++)
 |  |  |  |  | a[x][k] -= a[o][k] * w;   }
 |  | pivot[i] = o++;
 | } vec x0(d); vector <vec> t; int free = 0;
 | for (int i = 0; i < d; i++)
 |  | if (pivot[i] != -1) x0[i] = -a[pivot[i]][d];
 |  | else free ++;
 | for (int i = 0; i < d; i++) if (pivot[i] == -1) {
 |  | vec x(d); x[i] = -1;
 |  | for (int j = 0; j < d; j++)
 |  |  | if (pivot[j] != -1) x[j] = a[pivot[j]][i];
 |  | t.push_back(x);
 | } if (t.size()) {
 |  | vector <vec> f;
 |  | for (int u = 0; u < free; u++) {
 |  |  | vec x(free + 1);
 |  |  | for (int i = 0; i < free; i++)
 |  |  |  | for (int j = 0; j < d; j++)
 |  |  |  |  | x[i] += t[u][j] * t[i][j];
 |  |  | for (int j = 0; j < d; j++)
 |  |  |  | x[free] += t[u][j] * x0[j];
 |  |  | f.push_back(x);
 |  | }
 |  | auto [k, tt] = gauss(f, free, free);
 |  | assert (tt.size() == 0);
 |  | for (int x = 0; x < free; x++)
 |  |  | for (int i = 0; i < d; i++)
 |  |  |  | x0[i] += k[x] * t[x][i];
 | } return {x0, t}; }
```

5.24 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_0 x_k + n y_0 y_k
// y_{k+1} = x_0 y_k + y_0 x_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]}; }}
```

5.25 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]; }
```

5.26 自适应 Simpson

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

6. Tricks

6.1 线性预处理左边第 k 大

```cpp
int n, l[N][60], r[N][60], a[N], k;
/*
l[i][j] 表示 i 左边第 j 个 比 a[i] 大的数。
r[i][j] 同理
*/
vector<int> stk[60], tmp;
void init() {
    for (int i = 1; i <= n; i++) l[i][0] = r[i][0] = i;
    for (int i = n; i; i--) {
        for (int j = k; j; --j) {
            tmp.clear();
            while (!stk[j].empty() && a[i] > a[stk[j].back()]) {
                l[stk[j].back()][j] = i;
                tmp.push_back(stk[j].back());
                stk[j].pop_back();
            }
            if (j + 1 <= k)
                stk[j + 1].insert(stk[j + 1].end(), tmp.rbegin(), tmp.rend());
        }
        stk[1].push_back(i);
    }
    for (int i = 1; i <= k; i++) stk[i].clear();
    for (int i = 1; i <= n; i++) {
        for (int j = k; j; --j) {
            tmp.clear();
            while (!stk[j].empty() && a[i] > a[stk[j].back()]) {
                r[stk[j].back()][j] = i;
                tmp.push_back(stk[j].back());
                stk[j].pop_back();
            }
            if (j + 1 <= k)
                stk[j + 1].insert(stk[j + 1].end(), tmp.rbegin(), tmp.rend());
        }
        stk[1].push_back(i);
    }
}
```

7. Appendix

7.1 Formulas 公式表

7.1.1 Mobius Inversion

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

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

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

7.1.2 杜教筛

$$S_\varphi(n)=\frac{n(n+1)}{2}-\sum_{d=2}^{n}S_\varphi\left(\left\lfloor\frac{n}{d}\right\rfloor\right)$$

$$S_\mu(n)=1-\sum_{d=2}^{n}S_\mu\left(\left\lfloor\frac{n}{d}\right\rfloor\right)$$

7.1.3 降幂公式

$$a^k\equiv a^{k\bmod\varphi(p)+\varphi(p)},\ k\geq\varphi(p)$$

7.1.4 其他常用公式

$$\sum_{i=1}^{n}[(i,n)=1]\,i=n\frac{\varphi(n)+e(n)}{2}$$

$$\sum_{i=1}^{n}\sum_{j=1}^{i}[(i,j)=d]=S_\varphi\left(\left\lfloor\frac{n}{d}\right\rfloor\right)$$

$$\sum_{i=1}^{n}\sum_{j=1}^{m}[(i,j)=d]=\sum_{d|k}\mu\left(\frac{k}{d}\right)\left\lfloor\frac{n}{k}\right\rfloor\left\lfloor\frac{m}{k}\right\rfloor$$

$$\sum_{i=1}^{n}f(i)\sum_{j=1}^{\lfloor\frac{n}{i}\rfloor}g(j)=\sum_{i=1}^{n}g(i)\sum_{j=1}^{\lfloor\frac{n}{i}\rfloor}f(j)$$

7.1.5 单位根反演

$$\sum_{i=0}^{\lfloor\frac{n}{k}\rfloor}C_n^{ik}$$

引理

$$\frac{1}{k}\sum_{i=0}^{k-1}\omega_k^{in}=[k\mid n]$$

反演

$$\begin{aligned}Ans&=\sum_{i=0}^{n}C_n^i[k\mid i]\\&=\sum_{i=0}^{n}C_n^i(\frac{1}{k}\sum_{j=0}^{k-1}\omega_k^{ij})\\&=\frac{1}{k}\sum_{i=0}^{n}C_n^i\sum_{j=0}^{k-1}\omega_k^{ij}\\&=\frac{1}{k}\sum_{j=0}^{k-1}(\sum_{i=0}^{n}C_n^i(\omega_k^j)^i)\\&=\frac{1}{k}\sum_{j=0}^{k-1}(1+\omega_k^j)^n\end{aligned}$$

另, 如果要求的是 $[n\%k=t]$, 其实就是 $[k\mid(n-t)]$. 同理推式子即可.

7.1.6 Arithmetic Function

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

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

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

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

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

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

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

$J_k(n)$ 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.

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

$$\sum_{i=1}^{n}\sum_{j=1}^{n}[gcd(i,j)=1]ij=\sum_{i=1}^{n}i^2\varphi(i)$$

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

7.1.7 Binomial Coefficients

C	0	1	2	3	4	5	6	7	8	9	10
0	1										
1	1	1									
2	1	2	1								
3	1	3	3	1							
4	1	4	6	4	1						
5	1	5	10	10	5	1					
6	1	6	15	20	15	6	1				
7	1	7	21	35	35	21	7	1			
8	1	8	28	56	70	56	28	8	1		
9	1	9	36	84	126	126	84	36	9	1	
10	1	10	45	120	210	252	210	120	45	10	1

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

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

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

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

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

$$\binom{n}{k} = (-1)^k \binom{k-n-1}{k}, \sum_{k \le n} \binom{r+k}{k} = \binom{r+n+1}{n}$$
$$\binom{n_1 + \cdots + n_p}{m} = \sum_{k_1 + \cdots + k_p = m} \binom{n_1}{k_1} \cdots \binom{n_p}{k_p}$$

7.1.8 Fibonacci Numbers, Lucas Numbers

$$F(z) = \frac{z}{1 - z - z^2}$$
$$\hat{\phi} = \frac{1 - \sqrt{5}}{2}$$
$$\sum_{k=1}^{n} f_k = f_{n+2} - 1, \quad \sum_{k=1}^{n} f_k^2 = f_n f_{n+1}$$
$$\sum_{k=0}^{n} f_k f_{n-k} = \frac{1}{5}(n-1) f_n + \frac{2}{5} n f_{n-1}$$
$$\frac{f_{2n}}{f_n} = f_{n-1} + f_{n+1}$$
$$f_1 + 2f_2 + 3f_3 + \cdots + nf_n = nf_{n+2} - f_{n+3} + 2$$
$$\gcd(f_m, f_n) = f_{\gcd(m,n)}$$
$$f_n^2 + (-1)^n = f_{n+1} f_{n-1}$$
$$f_{n+k} = f_n f_{k+1} + f_{n-1} f_k$$
$$f_{2n+1} = f_n^2 + f_{n+1}^2$$
$$(-1)^k f_{n-k} = f_n f_{k-1} - f_{n-1} f_k$$

```python
def fib(n): # F(n), F(n + 1)
    if not n: return (0, 1)
    a, b = fib(n >> 1)
    c = a * (2 * b - a)
    d = a * a + b * b
    if n & 1:
        return (d, c + d)
    else:
        return (c, d)
```

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

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

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

除了 $n = 0, 4, 8, 16$, L_n 是素数, 则 n 是素数.

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

$$F_n = \frac{\phi^n - \hat{\phi}^n}{\sqrt{5}},\ L_n = \phi^n + \hat{\phi}^n$$

$$\frac{L_n + F_n\sqrt{5}}{2} = \left(\frac{1+\sqrt{5}}{2}\right)^n$$

7.1.9 Sum of Powers

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

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

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

7.1.10 Catalan Numbers 1, 1, 2, 5, 14, 42, 132, 429, 1430...

$$c_0 = 1, c_n = \sum_{i=0}^{n-1} c_i c_{n-1-i} = c_{n-1}\frac{4n-2}{n+1} = \frac{\binom{2n}{n}}{n+1} = \binom{2n}{n} - \binom{2n}{n-1}$$

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

Usage: n 对括号序列; n 个点满二叉树; $n \times n$ 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.

类卡特兰数 从 $(1,1)$ 出发走到 (n,m), 只能向右或者向上走, 不能越过 $y=x$ 这条线 (即保证 $x \geq y$), 合法方案数是 $C_{n+m-2}^{n} - C_{n+m-2}^{n-1}$.

7.1.11 Motzkin Numbers 1, 1, 2, 4, 9, 21, 51, 127, 323, 835...

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 $\sum_i^a k_i (1 \leq a \leq n)$ 非负且所有数总和为 0 的方案数.

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

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

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

7.1.12 Derangement 错排数 0, 1, 2, 9, 44, 265, 1854, 14833...

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

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

7.1.13 Bell Numbers 1, 1, 2, 5, 15, 52, 203, 877, 4140 ...

n 个元素集合划分的方案数.

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

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

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

7.1.14 Stirling Numbers

第一类 n 个元素集合分作 k 个**非空轮换**方案数.

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

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

n\k	0	1	2	3	4	5	6
0	1						
1	0	1					
2	0	1	1				
3	0	2	3	1			
4	0	6	11	6	1		
5	0	24	50	35	10	1	
6	0	120	274	225	85	15	1
7	0	720	1764	1624	735	175	21

$$\begin{bmatrix} n+1 \\ 2 \end{bmatrix} = n!H_n \text{ (see 7.1.16)}$$

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

$$x^{\overline{n}} = \sum_k \begin{bmatrix} n \\ k \end{bmatrix} x^k$$

For fixed k, EGF:

$$\sum_{n=0}^{\infty} \begin{bmatrix} n \\ k \end{bmatrix} \frac{x^n}{n!} = \frac{x^k}{k!} \left(\frac{\ln(1-x)}{x} \right)^k$$

第二类 把 n 个元素集合分作 k 个**非空子集**方案数.

$$\left\{ {n+1 \atop k} \right\} = k\left\{ {n \atop k} \right\} + \left\{ {n \atop k-1} \right\}$$

$$m! \left\{ {n \atop m} \right\} = \sum_k \binom{m}{k} k^n (-1)^{m-k}$$

n\k	0	1	2	3	4	5	6
0	1						
1	0	1					
2	0	1	1				
3	0	1	3	1			
4	0	1	7	6	1		
5	0	1	15	25	10	1	
6	0	1	31	90	65	15	1
7	0	1	63	301	350	140	21

$$\begin{aligned} x^n &= \sum_k \left\{ {n \atop k} \right\} x^{\underline{k}} \\ &= \sum_k \left\{ {n \atop k} \right\} (-1)^{n-k} x^{\overline{k}} \end{aligned}$$

For fixed k, EGF and OGF:

$$\sum_{n=0}^{\infty} \left\{ {n \atop k} \right\} \frac{x^n}{n!} = \frac{x^k}{k!} \left(\frac{e^x - 1}{x} \right)^k$$

$$\sum_{n=0}^{\infty} \left\{ {n \atop k} \right\} x^n = x^k \prod_{i=1}^{k} (1 - ix)^{-1}$$

7.1.15 Eulerian Numbers

n\k	0	1	2	3	4	5	6
1	1						
2	1	1					
3	1	4	1				
4	1	11	11	1			
5	1	26	66	26	1		
6	1	57	302	302	57	1	
7	1	120	1191	2416	1191	120	1

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

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

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

7.1.16 Harmonic Numbers, 1, 3/2, 11/6, 25/12, 137/60...

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

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

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

7.1.17 卡迈克尔函数

卡迈克尔函数表示模 m 剩余系下最大的阶, 即 $\lambda(m) = \max_{a \perp m} \delta_m(a)$.
容易看出, 若 $\lambda(m) = \varphi(m)$, 则 m 存在原根.
该函数可由下述方法计算: 分解质因数 $m = p_1^{\alpha_1} p_2^{\alpha_2} ... p_t^{\alpha_t}$. 则 $\lambda(m) = \text{lcm}(\lambda(p_1^{\alpha_1}), p_2^{\alpha_2}, ..., p_t^{\alpha_t})$. 其中对奇质数 p, $\lambda(p^\alpha) = (p-1)p^{\alpha-1}$.
对 2 的幂, $\lambda(2^k) = 2^{k-2}, s.t. k \geq 3$. $\lambda(4) = \lambda(2) = 2$.

7.1.18 求拆分数

```
def penta(k):
  return k*(3*k-1)//2
def compute_partition(goal):
  p = [1]
  for n in range(1,goal+1):
    p.append(0)
    for k in range(1,n+1):
      c = (-1)**(k+1)
      for t in [penta(k), penta(-k)]:
        if (n-t) >= 0:
          p[n] = p[n] + c*p[n-t]
  return p
```

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

记 $p(n)$ 表示 n 的拆分数,$f(n,k)$ 表示将 n 拆分且每种数字使用次数必须小于等于 k 的拆分数. 则

$$P(x)\Phi(x) = 1, F(x^k)\Phi(x) = 1$$

暴力拆开卷积, 可以得到将 1,-1,2,-2... 带入五边形数 $(-1)^k x^{k(3k-1)/2}$ 中, 由于小于 n 的五边形数只有 $\sqrt{n}$ 个, 可以 $O(n\sqrt{n})$ 计算答案:
$p(n) = p(n-1) + p(n-2) - p(n-5) - p(n-7) + \cdots,$
$f(n,k) = p(n) - p(n-k) - p(n-2k) + p(n-5k) + p(n-7k) - \cdots$

7.1.19 Bernoulli Numbers 1, 1/2, 1/6, 0, -1/30, 0, 1/42 ...

$$B(x) = \sum_{i\geq 0} \frac{B_i x^i}{i!} = \frac{x}{e^x - 1}$$

$$B_n = [n = 0] - \sum_{i=0}^{n-1} \binom{n}{i} \frac{B_i}{n-k+1}, \sum_{i=0}^{n} \binom{n+1}{i} B_i = 0$$

$$S_n(m) = \sum_{i=0}^{m-1} i^n = \sum_{i=0}^{n} \binom{n}{i} B_{n-i} \frac{m^{i+1}}{i+1}$$

$B_0 = 1,\ B_1 = -\frac{1}{2},\ B_4 = -\frac{1}{30},\ B_6 = \frac{1}{42},\ B_8 = -\frac{1}{30},\ \ldots.$
(除了 $B_1 = -\frac{1}{2}$ 以外, 伯努利数的奇数项都是 0.)
自然数幂次和关于次数的 EGF:

$$F(x) = \sum_{k=0}^{\infty} \frac{\sum_{i=0}^{n} i^k}{k!} x^k$$
$$= \sum_{i=0}^{n} e^{ix} = \frac{e^{(n+1)x-1}}{e^x - 1}$$

7.1.20 kMAX-MIN 反演

$$k\mathrm{MAX}(S) = \sum_{T\subset S, T\neq\emptyset} (-1)^{|T|-k} C_{|T|-1}^{k-1} \mathrm{MIN}(T)$$

代入 $k = 1$ 即为 MAX-MIN 反演:

$$\mathrm{MAX}(S) = \sum_{T\subset S, T\neq\emptyset} (-1)^{|T|-1} \mathrm{MIN}(T)$$

7.1.21 伍德伯里矩阵不等式

$$(A + UCV)^{-1} = A^{-1} - A^{-1}U(C^{-1} + VA^{-1}U)^{-1}VA^{-1}$$

该等式可以动态维护矩阵的逆, 令 $C = [1], U, V$ 分别为 $1 \times n$ 和 $n \times 1$ 的向量, 这样可以构造出 UCV 为只有某行或者某列不为 0 的矩阵, 一次修改复杂度为 $O(n^2)$.

7.1.22 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).

7.1.23 枚举勾股数 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.

7.1.24 四面体体积 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)

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

7.1.25 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.

7.1.26 常见博弈游戏

Nim-K 游戏 n 堆石子轮流拿, 每次最多可以拿 k 堆石子, 谁走最后一步输。结论: 把每一堆石子的 sg 值 (即石子数量) 二进制分解, 先手必败当且仅当每一位二进制位上 1 的个数是 (k+1) 的倍数.

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 n 个人 $0, \ldots, n-1$, 令 $f_{i,m}$ 为 i 个人报 m 的胜利者, 则 $f_{1,m} = 0, f_{i,m} = (f_{i-1,m} + m) \bmod i$.

阶梯 Nim 在一个阶梯上, 每次选一个台阶上任意个式子移到下一个台阶上, 不可移动者输. 结论: SG 值等于奇数层台阶上石子数的异或和. 对于树形结构也适用, 奇数层节点上所有石子数异或起来即可.

图上博弈 给定无向图, 先手从某点开始走, 只能走相邻且未走过的点, 无法移动者输. 对该图求最大匹配, 若某个点不一定在最大匹配中则先手必败, 否则先手必胜.

最大最小定理求纳什均衡点 在二人零和博弈中, 可以用以下方式求出一个纳什均衡点: 在博弈双方中**任选**一方, 求混合策略 $\mathbf{p}$ 使得对方选择任意一个**纯策略**时, 己方的最小收益最大 (等价于对方的最大收益最小). 据此可以求出双方在此局面下的最优期望得分, 分别等于己方最大的最小收益和对方最小的最大收益. 一般而言, 可以得到形如

$$\max_{\mathbf{p}} \min_{i} \sum_{p_j \in \mathbf{p}, p_j \geq 0, \sum p_j = 1} p_j w_{i,j}$$

的形式. 当 $\sum p_j w_{i,j}$ 可以表示成只与 i 有关的函数 $f(i)$ 时, 可以令初始时 $p_i = 0$, 不断调整 $\sum p_j w_{i,j}$ 最小的那个 i 的概率 p_i, 直至无法调整或者 $\sum p_j = 1$ 为止.

7.1.27 概率相关

$D(X) = E(X - E(X))^2 = E(X^2) - (E(X))^2, D(X+Y) = D(X) + D(Y),$

$D(aX) = a^2 D(X), E[x] = \sum_{i=1}^{\infty} P(X \geq i)$, m 个数的方差: $s^2 = \dfrac{\sum_{i=1}^{m} x_i^2}{m} - \overline{x}^2$

7.1.28 邻接矩阵行列式的意义

在无向图中取若干个环, 一种取法权值就是边权的乘积, 对行列式的贡献是 $(-1)^{\text{even}}$, 其中 even 是偶环的个数.

7.1.29 Others (某些近似数值公式在这里)

$$e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!} = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots$$

$$\sin(x) = \sum_{n=0}^{\infty} \frac{(-1)^n x^{2n+1}}{(2n+1)!} = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \cdots$$

$$\cos(x) = \sum_{n=0}^{\infty} \frac{(-1)^n x^{2n}}{(2n)!} = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \cdots$$

$$\ln(1+x) = \sum_{n=1}^{\infty} \frac{(-1)^{n+1} x^n}{n} = x - \frac{x^2}{2} + \frac{x^3}{3} - \cdots \quad \text{for} \quad -1 < x \leq 1$$

$$\arctan(x) = \sum_{n=0}^{\infty} \frac{(-1)^n x^{2n+1}}{2n+1} = x - \frac{x^3}{3} + \frac{x^5}{5} - \cdots \quad \text{for} \quad -1 \leq x \leq 1$$

$$(1+x)^k = \sum_{n=0}^{\infty} \binom{k}{n} x^n = 1 + kx + \frac{k(k-1)}{2!}x^2 + \frac{k(k-1)(k-2)}{3!}x^3 + \cdots$$

$$\arcsin(x) = x + \sum_{n=1}^{\infty} \frac{(2n-1)!!}{(2n)!!} \frac{x^{2n+1}}{2n+1}$$

$$\frac{\pi}{4} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \cdots, \frac{1}{\pi} = \frac{2\sqrt{2}}{9801} \sum_{k=0}^{\infty} \frac{(4k)!(1103 + 26390k)}{(k!)^4 396^{4k}}$$

$$\sum_{i=1}^{n} \frac{1}{n} \approx \ln\left(n + \frac{1}{2}\right) + \frac{1}{24(n+0.5)^2} + \Gamma, (\Gamma \approx 0.57721566490153286060 65)$$

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

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

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

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

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

$n \bmod 2 = 1$:

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

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, Integration Table 导数积分表

$(\frac{u}{v})' = \frac{u'v - uv'}{v^2}$
$(a^x)' = (\ln a)a^x$
$(\tan x)' = \sec^2 x$
$(\cot x)' = \csc^2 x$
$(\sec x)' = \tan x \sec x$
$(\csc x)' = -\cot x \csc x$
$(\arcsin x)' = \frac{1}{\sqrt{1-x^2}}$
$(\arccos x)' = -\frac{1}{\sqrt{1-x^2}}$
$(\arctan x)' = \frac{1}{1+x^2}$
$(\operatorname{arccot} x)' = -\frac{1}{1+x^2}$
$(\operatorname{arccsc} x)' = -\frac{1}{x\sqrt{1-x^2}}$
$(\operatorname{arcsec} x)' = \frac{1}{x\sqrt{1-x^2}}$
$(\tanh x)' = \operatorname{sech}^2 x$
$(\coth x)' = -\operatorname{csch}^2 x$
$(\operatorname{sech} x)' = -\operatorname{sech} x \tanh x$
$(\operatorname{csch} x)' = -\operatorname{csch} x \coth x$
$(\operatorname{arcsinh} x)' = \frac{1}{\sqrt{1+x^2}}$
$(\operatorname{arccosh} x)' = \frac{1}{\sqrt{x^2-1}}$
$(\operatorname{arctanh} x)' = \frac{1}{1-x^2}$
$(\operatorname{arccoth} x)' = \frac{1}{x^2-1}$
$(\operatorname{arccsch} x)' = -\frac{1}{|x|\sqrt{1+x^2}}$
$(\operatorname{arcsech} x)' = -\frac{1}{x\sqrt{1-x^2}}$

· $ax^2+bx+c(a>0)$

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

$$\int \frac{x}{ax^2+bx+c}\mathrm{d}x=\frac{1}{2a}\ln|ax^2+bx+c|-\frac{b}{2a}\int\frac{\mathrm{d}x}{ax^2+bx+c}$$

· $\sqrt{\pm ax^2+bx+c}(a>0)$

$$\int \frac{\mathrm{d}x}{\sqrt{ax^2+bx+c}}=\frac{1}{\sqrt{a}}\ln|2ax+b+2\sqrt{a}\sqrt{ax^2+bx+c}|+C$$

$$\int \sqrt{ax^2+bx+c}\mathrm{d}x=\frac{2ax+b}{4a}\sqrt{ax^2+bx+c}+\frac{4ac-b^2}{8\sqrt{a^3}}\ln|2ax+b+2\sqrt{a}\sqrt{ax^2+bx+c}|+C$$

$$\int \frac{x}{\sqrt{ax^2+bx+c}}\mathrm{d}x=\frac{1}{a}\sqrt{ax^2+bx+c}-\frac{b}{2\sqrt{a^3}}\ln|2ax+b+2\sqrt{a}\sqrt{ax^2+bx+c}|+C$$

$$\int \frac{\mathrm{d}x}{\sqrt{c+bx-ax^2}}=-\frac{1}{\sqrt{a}}\arcsin\frac{2ax-b}{\sqrt{b^2+4ac}}+C$$

$$\int \sqrt{c+bx-ax^2}\mathrm{d}x=\frac{2ax-b}{4a}\sqrt{c+bx-ax^2}+\frac{b^2+4ac}{8\sqrt{a^3}}\arcsin\frac{2ax-b}{\sqrt{b^2+4ac}}+C$$

$$\int \frac{x}{\sqrt{c+bx-ax^2}}\mathrm{d}x=-\frac{1}{a}\sqrt{c+bx-ax^2}+\frac{b}{2\sqrt{a^3}}\arcsin\frac{2ax-b}{\sqrt{b^2+4ac}}+C$$

· $\sqrt{\pm\frac{x-a}{x-b}}$ **或** $\sqrt{(x-a)(x-b)},(a<b)$

$$\int \frac{\mathrm{d}x}{\sqrt{(x-a)(b-x)}}=2\arcsin\sqrt{\frac{x-a}{b-x}}+C$$

$$\int \sqrt{(x-a)(b-x)}\mathrm{d}x=\frac{2x-a-b}{4}\sqrt{(x-a)(b-x)}+\frac{(b-a)^2}{4}\arcsin\sqrt{\frac{x-a}{b-x}}+C$$

三角函数的积分 (省略 $+C$)

$$\int \tan x\mathrm{d}x=-\ln|\cos x|$$
$$\int \cot x\mathrm{d}x=\ln|\sin x|$$
$$\int \sec x\mathrm{d}x=\ln\left|\tan\left(\frac{\pi}{4}+\frac{x}{2}\right)\right|$$
$$=\ln|\sec x+\tan x|$$
$$\int \csc x\mathrm{d}x=\ln\left|\tan\frac{x}{2}\right|$$
$$=\ln|\csc x-\cot x|$$

$$\int \sec^2 x\mathrm{d}x=\tan x$$
$$\int \csc^2 x\mathrm{d}x=-\cot x$$
$$\int \sec x\tan x\mathrm{d}x=\sec x$$
$$\int \csc x\cot x\mathrm{d}x=-\csc x$$
$$\int \sin^2 x\mathrm{d}x=\frac{x}{2}-\frac{1}{4}\sin 2x$$
$$\int \cos^2 x\mathrm{d}x=\frac{x}{2}+\frac{1}{4}\sin 2x$$

$$\int \sin^n x\mathrm{d}x=-\frac{1}{n}\sin^{n-1}x\cos x+\frac{n-1}{n}\int\sin^{n-2}x\mathrm{d}x$$
$$\int \cos^n x\mathrm{d}x=\frac{1}{n}\cos^{n-1}x\sin x+\frac{n-1}{n}\int\cos^{n-2}x\mathrm{d}x$$
$$\int \frac{\mathrm{d}x}{\sin^n x}=-\frac{1}{n-1}\frac{\cos x}{\sin^{n-1}x}+\frac{n-2}{n-1}\int\frac{\mathrm{d}x}{\sin^{n-2}x}$$
$$\int \frac{\mathrm{d}x}{\cos^n x}=\frac{1}{n-1}\frac{\sin x}{\cos^{n-1}x}+\frac{n-2}{n-1}\int\frac{\mathrm{d}x}{\cos^{n-2}x}$$

$$\begin{aligned}&\int\cos^m x\sin^n x\mathrm{d}x\\ =&\frac{1}{m+n}\cos^{m-1}x\sin^{n+1}x+\frac{m-1}{m+n}\int\cos^{m-2}x\sin^n x\mathrm{d}x\\ =&-\frac{1}{m+n}\cos^{m+1}x\sin^{n-1}x+\frac{n-1}{m+1}\int\cos^m x\sin^{n-2}x\mathrm{d}x\end{aligned}$$

$$\int \frac{\mathrm{d}x}{a+b\sin x}=\begin{cases}\frac{2}{\sqrt{a^2-b^2}}\arctan\frac{a\tan\frac{x}{2}+b}{\sqrt{a^2-b^2}} & (a^2>b^2)\\ \frac{1}{\sqrt{b^2-a^2}}\ln\left|\frac{a\tan\frac{x}{2}+b-\sqrt{b^2-a^2}}{a\tan\frac{x}{2}+b+\sqrt{b^2-a^2}}\right| & (a^2<b^2)\end{cases}$$

$$\int \frac{\mathrm{d}x}{a+b\cos x}=\begin{cases}\frac{2}{a+b}\sqrt{\frac{a+b}{a-b}}\arctan\left(\sqrt{\frac{a-b}{a+b}}\tan\frac{x}{2}\right) & (a^2>b^2)\\ \frac{1}{a+b}\sqrt{\frac{a+b}{a-b}}\ln\left|\frac{\tan\frac{x}{2}+\sqrt{\frac{a+b}{b-a}}}{\tan\frac{x}{2}-\sqrt{\frac{a+b}{b-a}}}\right| & (a^2<b^2)\end{cases}$$

$$\int \frac{\mathrm{d}x}{a^2\cos^2x+b^2\sin^2x}=\frac{1}{ab}\arctan\left(\frac{b}{a}\tan x\right)$$

$$\int \frac{\mathrm{d}x}{a^2\cos^2x-b^2\sin^2x}=\frac{1}{2ab}\ln\left|\frac{b\tan x+a}{b\tan x-a}\right|$$

$$\int x\sin ax\mathrm{d}x=\frac{1}{a^2}\sin ax-\frac{1}{a}x\cos ax$$
$$\int x^2\sin ax\mathrm{d}x=-\frac{1}{a}x^2\cos ax+\frac{2}{a^2}x\sin ax+\frac{2}{a^3}\cos ax$$
$$\int x\cos ax\mathrm{d}x=\frac{1}{a^2}\cos ax+\frac{1}{a}x\sin ax$$
$$\int x^2\cos ax\mathrm{d}x=\frac{1}{a}x^2\sin ax+\frac{2}{a^2}x\cos ax-\frac{2}{a^3}\sin ax$$

反三角函数的积分 (其中 $a>0$)

$$\int \arcsin\frac{x}{a}\mathrm{d}x=x\arcsin\frac{x}{a}+\sqrt{a^2-x^2}+C$$
$$\int x\arcsin\frac{x}{a}\mathrm{d}x=(\frac{x^2}{2}-\frac{a^2}{4})\arcsin\frac{x}{a}+\frac{x}{4}\sqrt{x^2-x^2}+C$$
$$\int x^2\arcsin\frac{x}{a}\mathrm{d}x=\frac{x^3}{3}\arcsin\frac{x}{a}+\frac{1}{9}(x^2+2a^2)\sqrt{a^2-x^2}+C$$
$$\int \arccos\frac{x}{a}\mathrm{d}x=x\,arccos\frac{x}{a}-\sqrt{a^2-x^2}+C$$
$$\int x\arccos\frac{x}{a}\mathrm{d}x=(\frac{x^2}{2}-\frac{a^2}{4})\arccos\frac{x}{a}-\frac{x}{4}\sqrt{a^2-x^2}+C$$
$$\int x^2\arccos\frac{x}{a}\mathrm{d}x=\frac{x^3}{3}\arccos\frac{x}{a}-\frac{1}{9}(x^2+2a^2)\sqrt{a^2-x^2}+C$$
$$\int \arctan\frac{x}{a}\mathrm{d}x=x\arctan\frac{x}{a}-\frac{a}{2}\ln(a^2+x^2)+C$$

$\int x \arctan \frac{x}{a} \mathrm{d}x = \frac{1}{2}(a^2 + x^2) \arctan \frac{x}{a} - \frac{a}{2}x + C$

$\int x^2 \arctan \frac{x}{a} \mathrm{d}x = \frac{x^3}{3} \arctan \frac{x}{a} - \frac{a}{6}x^2 + \frac{a^3}{6} \ln(a^2 + x^2) + C$

指数函数的积分

$\int a^x \mathrm{d}x = \frac{1}{\ln a} a^x + C$

$\int \mathrm{e}^{ax} \mathrm{d}x = \frac{1}{a} a^{ax} + C$

$\int x\mathrm{e}^{ax} \mathrm{d}x = \frac{1}{a^2}(ax - 1)a^{ax} + C$

$\int x^n \mathrm{e}^{ax} \mathrm{d}x = \frac{1}{a} x^n \mathrm{e}^{ax} - \frac{n}{a} \int x^{n-1} \mathrm{e}^{ax} \mathrm{d}x$

$\int x a^x \mathrm{d}x = \frac{x}{\ln a} a^x - \frac{1}{(\ln a)^2} a^x + C$

$\int x^n a^x \mathrm{d}x = \frac{1}{\ln a} x^n a^x - \frac{n}{\ln a} \int x^{n-1} a^x \mathrm{d}x$

$\int \mathrm{e}^{ax} \sin bx \mathrm{d}x = \frac{1}{a^2+b^2} \mathrm{e}^{ax}(a \sin bx - b \cos bx) + C$

$\int \mathrm{e}^{ax} \cos bx \mathrm{d}x = \frac{1}{a^2+b^2} \mathrm{e}^{ax}(b \sin bx + a \cos bx) + C$

$\int \mathrm{e}^{ax} \sin^n bx \mathrm{d}x = \frac{1}{a^2+b^2n^2} \mathrm{e}^{ax} \sin^{n-1} bx(a \sin bx - nb \cos bx) + \frac{n(n-1)b^2}{a^2+b^2n^2} \int \mathrm{e}^{ax} \sin^{n-2} bx \mathrm{d}x$

$\int \mathrm{e}^{ax} \cos^n bx \mathrm{d}x = \frac{1}{a^2+b^2n^2} \mathrm{e}^{ax} \cos^{n-1} bx(a \cos bx + nb \sin bx) + \frac{n(n-1)b^2}{a^2+b^2n^2} \int \mathrm{e}^{ax} \cos^{n-2} bx \mathrm{d}x$

对数函数的积分

$\int \ln x \mathrm{d}x = x \ln x - x + C$

$\int \frac{\mathrm{d}x}{x \ln x} = \ln \left| \ln x \right| + C$

$\int x^n \ln x \mathrm{d}x = \frac{1}{n+1} x^{n+1} (\ln x - \frac{1}{n+1}) + C$

$\int (\ln x)^n \mathrm{d}x = x(\ln x)^n - n \int (\ln x)^{n-1} \mathrm{d}x$

$\int x^m (\ln x)^n \mathrm{d}x = \frac{1}{m+1} x^{m+1} (\ln x)^n - \frac{n}{m+1} \int x^m (\ln x)^{n-1} \mathrm{d}x$

7.3 Python Hint

```python
def RandomAndList():
    import random
    random.normalvariate(0.5, 0.1)
    l = [str(i) for i in range(9)]
    sorted(l), min(l), max(l), len(l)
    random.shuffle(l)
    l.sort(key=lambda x:x ^ 1,reverse=True)
    from functools import cmp_to_key
    l.sort(key=cmp_to_key(lambda x, y:(y^1)-(x^1)))
    from itertools import *
    for i in product('ABCD', repeat=2):
        pass # AA AB AC AD BA BB BC BD CA CB CC CD DA DB DC DD
    for i in permutations('ABCD', repeat=2):
        pass # AB AC AD BA BC BD CA CB CD DA DB DC
    for i in combinations('ABCD', repeat=2):
        pass # AB AC AD BC BD CD
    for i in combinations_with_replacement('ABCD', repeat=2):
        pass # AA AB AC AD BB BC BD CC CD DD
def FractionOperation():
    from fractions import Fraction
    a.numerator, a.denominator, str(a)
    a = Fraction(0.233).limit_denominator(1000)
def DecimalOperation():
    from decimal import Decimal, getcontext, FloatOperation
    getcontext().prec = 100
    getcontext().rounding = getattr(decimal, 'ROUND_HALF_EVEN')
    # default; other: FLOOR, CELILING, DOWN, ...
    getcontext().traps[FloatOperation] = True
    Decimal((0, (1, 4, 1, 4), -3)) # 1.414
    a = Decimal(1<<31) / Decimal(100000)
    print(f"{a:.9f}") # 21474.83648
    print(a.sqrt(), a.ln(), a.log10(), a.exp(), a ** 2)
def Complex():
    a = 1-2j
    print(a.real, a.imag, a.conjugate())
def FastIO():
    import sys, atexit, io
    _INPUT_LINES = sys.stdin.read().splitlines()
    input = iter(_INPUT_LINES).__next__
    _OUTPUT_BUFFER = io.StringIO()
    sys.stdout = _OUTPUT_BUFFER
    @atexit.register
    def write():
        sys.__stdout__.write(_OUTPUT_BUFFER.getvalue())
```

8. Miscellany

8.1 WIN 运行脚本

```cpp
#include <bits/stdc++.h>
using namespace std;

int main(int argc, char* argv[]) {
    string s = argv[1];
    string cpp = s + ".cpp";
    string exe = s + ".exe";
    string out = s + ".out";
    string err = s + ".err";

    string cp = "g++ -std=c++20 -O2 " + cpp + " -o " + exe;

    if (system(cp.c_str()) != 0) {
        cerr << "Compile Failed!" << endl;
        return 1;
    } else {
        system("cls");
        cerr << "Compile Success!" << endl;
    }

    string run = exe;
    if (argc > 2) {
        run += " < " + string(argv[2]);
    }
    run += " > " + out + " 2> " + err;

    system(run.c_str());

    cout << "\n--- Output ---\n";
    system(("type " + out).c_str());
    cout << "\n--------------\n";

    cout << "\n--- Error ---\n";
    system(("type " + err).c_str());
    cout << "\n--------------\n";

    return 0;
}
/*
首先编译运行脚本，这只需要执行一次：g++ r.cpp -o r.exe
以文件内容作为输入：r a a.in（文件名可以随意指定）
手动输入：r a
*/
```

8.2 Zeller 日期公式

```cpp
// 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; }
//  y<0: 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); }
```

8.3 有理数二分: Stern-Brocot 树, Farey 序列

```python
def build(a, b, c, d, level = 1):
```

```
    x = a + c; y = b + d
    build(a, b, x, y, level + 1)
    build(x, y, c, d, level + 1)
build(0, 1, 1, 0) # 最简分数, Stern-Brocot
build(0, 1, 1, 1) # 最简真分数, Farey
```

8.4 黄金三分

```
constexpr LD R = (sqrt(5) - 1) / 2;
auto split = [](LD l, LD r) { return l + (r - l) * R; };
LD solve(LD a, LD c, auto f) {
 | LD b = split(a, c), bv = f(b);
 | for (int _ = T; _; _--) {
 |  | LD x = split(a, b), xv = f(x);
 |  | if (xv < bv) c = b, b = x, bv = xv; // 最小化, 注意符号
 |  | else a = c, c = x;
 | } return bv; }
```

8.5 DP 优化

8.5.1 四边形不等式

```
// a ≤ b ≤ c ≤ d : w(b,c) ≤ w(a,d), w(a,c) + w(b,d) ≤ w(a,d) + w(b,c)
for (int len = 2; len <= n; ++len) {
 | for (int l = 1, r = len; r <= n; ++l, ++r) {
 |  | f[l][r] = INF;
 |  | for (int k = m[l][r - 1]; k <= m[l + 1][r]; ++k) {
 |  |  | if (f[l][r] > f[l][k] + f[k + 1][r] + w(l, r)) {
 |  |  |  | f[l][r] = f[l][k] + f[k + 1][r] + w(l, r);
 |  |  |  | m[l][r] = k;
 | } } }
```

8.5.2 树形背包优化

限制: 必须取与根节点相连的一个连通块.

转化: 一个点的子树对应于 DFS 序中的一个区间. 则每个点的决策为, 取该点, 或者舍弃该点对应的区间. 从后往前 dp, 设 $f(i,v)$ 表示从后往前考虑到 i 号点, 总体积为 V 的最优价值, 设 i 号点对应的区间为 $[i, i+\text{size}_i-1]$, 转移为 $f(i,v)=\max\{f(i+1,V-v_i)+w_i, f(i+\text{size}_i, v)\}$.

如果要求任意连通块, 则点分治后转为指定根的连通块问题即可.

8.5.3 $O(n \cdot \max a_i)$ Subset Sum

```
int SubsetSum(vector <int> &a, int t) {
  int B = *max_element(a.begin(), a.end());
  int n = (int) a.size(), s = 0, i = 0;
  while (i < n && s + a[i] <= t) s += a[i ++];
  if (i == n) return s;
  vector <int> f(2 * B + 1, -1), pre (B + 1, -1);
  f[s - (t - B)] = i;
  for ( ; i < n; i++) {
    s += a[i];
    for (int d = 0; d <= B; d++) pre[d] = max(0, f[d + B]);
    for (int d = B; d >= 0; d--)
      f[d + a[i]] = max(f[d + a[i]], f[d]);
    for (int d = 2 * B; d > B; --d)
      for (int j = pre[d - B]; j < f[d]; j++)
        f[d - a[j]] = max(f[d - a[j]], j); }
  for (i = 0; i <= B; i++) if (f[B - i] >= 0) return t - i;}
```

8.6 Hash Table

```
template <class T,int P = 314159/*,451411,1141109,2119969*/>
struct hashmap {
ULL id[P]; T val[P];
int R[P]; // del: few clears
hashmap() {memset(id, -1, sizeof id);}
T get(const ULL &x) const {
 | for (int i = int(x % P), j = 1; ~id[i]; i = (i + j) % P, j = (j + 2) % P /*unroll if needed*/) {
 |  | if (id[i] == x) return val[i]; }
 | return 0; }
T& operator [] (const ULL &x) {
 | for (int i = int(x % P), j = 1;       ; i = (i + j) % P, j = (j + 2) % P) {
 |  | if (id[i] == x) return val[i];
 |  | else if (id[i] == -1llu) {
 |  |  | id[i] = x;
 |  |  | R[++R[0]] = i;  // del: few clears
 |  |  | return val[i]; } } }
void clear() { // del: few clears
for (int &x = R[0]; x; id[R[x]] = -1, val[R[x]] = 0, --x);
void fullclear() {memset(id, -1, sizeof id); R[0] = 0; } };
```

8.7 基数排序

```
const int SZ = 1 << 8; // almost always fit in L1 cache
void SORT(int a[], int c[], int n, int w) {
 | for(int i=0; i<SZ; i++) b[i] = 0;
 | for(int i=1; i<=n; i++) b[(a[i]>>w) & (SZ-1)]++;
 | for(int i=1; i<SZ; i++) b[i] += b[i - 1];
 | for(int i=n; i; i--) c[b[(a[i]>>w) & (SZ-1)]--] = a[i];}
void Sort(int *a, int n){
 | SORT(a, c, n, 0); SORT(c, a, n, 8);
 | SORT(a, c, n, 16); SORT(c, a, n, 24); }
```

8.8 Hacks: O3, 读入优化, Bitset, builtin

```
//fast = O3 + ffast-math + fallow-store-data-races
#pragma GCC optimize("Ofast")
#pragma GCC target("sse2,abm,fma,mmx,avx2,tune=native")
const int SZ = 1 << 16; int getc() {
 | static char buf[SZ], *ptr = buf, *top = buf;
 | if (ptr == top) {
 |  | ptr = buf, top = buf + fread(buf, 1, SZ, stdin);
 |  | if (top == buf) return -1; }
 | return *ptr++; }
idx=b._Find_first();idx!=b.size();idx=b._Find_next(idx);
struct HashFunc{size_t operator()(const KEY &key)const{}};
__builtin_uaddll_overflow(a, b, &c) // binary big int
void GospersHack(int k, int n) {
 | for (int s = (1 << k) - 1, c, r; s < (1 << n);
 | c = s & -s, r = s + c, s = (((r ^ s) >> 2) / c) | r); }
```

8.9 试机赛与纪律文件

- 检查身份证件: 护照、学生证、胸牌以及现场所需通行证。
- 确认什么东西能带进场。特别注意: 智能手表、金属（钥匙）等等。
- 测试鼠标、键盘、显示器和座椅。如果有问题, 立刻联系工作人员。
- 测试比赛提交方式。如果有 submit 命令, 确认如何使用。
- 设置自动保存，自动备份。
- 测试编译器版本。C++20 cin >> (s + 1); C++17 auto [x, y]: a; C++14 [](auto x, auto y); C++11 auto; bits/stdc++.h; pb_ds。

```
#include <ext/rope>
using namespace __gnu_cxx;
rope <int> R; R.insert(y, x); R[x]; R.erase(x, 1);
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;
tree <int, null_type, less<int>, rb_tree_tag,
```

```
tree_order_statistics_node_update> s;
s.insert(1); s.find_by_order(0); s.order_of_key(5);
```

- 测试 __int128, __float128, long double
- 测试代码长度限制，尝试触发 NO-OUTPUT, OUTPUT-LIMIT, RUN-ERROR。
- 测试 pragma 是否 CE。
- 测试 clar: 如果问不同类型的愚蠢的问题, 得到的回复是否不一样?
- 测试 -fsanitize address, undefined, define _GLIBCXX_DEBUG
- 测试 time 命令是否能显示内存占用。/usr/bin/time -v ./a.out
- 测试 clock() 是否能够正常工作; 测试本地性能与提交性能。

```
const int N = 1 << 20;
for (int T = 4; T; T--) {
    int a[N], b[N], c[N];
    for (int i = 0; i < N; i++) a[i] = i, b[i] = N - i;
    ntt_init(N); ntt(a, N, 1); ntt(b, N, 1);
    for (int i = 0; i < N; i++) c[i] = a[i] * b[i];
    ntt(c, N, -1); assert (c[0] == 0xad223d); }
// 机房: 190ms; (1s T = 21)
// QOJ: 340+-40ms; (1s T = 11.9)
// CF: 1050+-100ms; (1s T = 3.9)
```

```
import sys
sys.set_int_max_str_digits(0) # >= Python 3.11
T = 3
for _ in range(T):
    assert str(3 ** 100000 * 10 ** 100000)[:9] == "133497141"
# 机房: 590ms; (1s T = 5)
# QOJ: 201ms; (1s T = 16)
# CF Python3: 1100ms; (1s T = 2.9) Pypy3: 310ms; (1s T = 12)
```

提交前检查:

- 保存, 编译, 测过样例，测过边界 n=0/1
- 数组开到 n/2n/m/n+32, LL/int128
- 多测清空，调用初始化, 清空时数组大小, 没读入完就 break
- 取模取到正值，输入输出格式 (%Lf，%llu)
- 时间空间限制, 关闭流同步，时间卡开 Ofast

提交后检查:

- sum 多组输入: 应只用了输入内容 (memset TL, 错下标 WA), 是否正确撤销
- 输入是否保证顺序，保证是整数，保证三点共线
- int/LL 溢出, INF/-1 大小，浮点数 eps 和 = 0, __builtin_popcount**ll**
- 类似 pair <LL, int> x = pair <int, int>(); 不会报警告
- 离散化与二分: lower_bound upper_bound +-1, begin(), end()
- 自定义排序：排序方向，比较函数为小于，考虑坏点: 如叉积 (0, 0)
- 样例是否为对称/回文的? 考虑构造不对称的情况，正序逆序
- 复制过的代码对应位置正确修改
- 检查变量重名，变量复用
- 图: 边标号初始是否为 1, 单双向边, 反向边边权
- 几何: 共线, sqrt(-0.0), nan / inf, 重点, 除零 det/dot, 旋转方向, 求得的是否是所求

8.10 Constant Table 常数表

NTT 976224257 r=3 (20) 985661441 r=3 (22) 998244353 r=3 (23)

1004535809 r=3 (21) 1007681537 r=3 (20) 1012924417 r=5 (21)

1045430273 r=3 (20) 1051721729 r=6 (20) 1053818881 r=7 (20)

n	$\log_{10} n$	$n!$	$C(n, n/2)$	$\mathrm{LCM}(1 \dots n)$	P_n
2	0.30102999	2	2	2	2
3	0.47712125	6	3	6	3
4	0.60205999	24	6	12	5
5	0.69897000	120	10	60	7
6	0.77815125	720	20	60	11
7	0.84509804	5040	35	420	15
8	0.90308998	40320	70	840	22
9	0.95424251	362880	126	2520	30
10	1	3628800	252	2520	42
11	1.04139269	39916800	462	27720	56
12	1.07918125	479001600	924	27720	77
15	1.17609126	1.31e12	6435	360360	176
20	1.30103000	2.43e18	184756	232792560	627
25	1.39794001	1.55e25	5200300	26771144400	1958
30	1.47712125	2.65e32	155117520	1.444e14	5604
P_n	37338_{40}	204226_{50}	966467_{60}	190569292_{100}	$1e9_{114}$

$n \le$	10	100	1e3	1e4	1e5	1e6
$\max \omega(n)$	2	3	4	5	6	7
$\max d(n)$	4	12	32	64	128	240
$\pi(n)$	4	25	168	1229	9592	78498
$n \le$	1e7	1e8	1e9	1e10	1e11	1e12
$\max \omega(n)$	8	8	9	10	10	11
$\max d(n)$	448	768	1344	2304	4032	6720
$\pi(n)$	664579	5761455	5.08e7	4.55e8	4.12e9	3.7e10
$n \le$	1e13	1e14	1e15	1e16	1e17	1e18
$\max \omega(n)$	12	12	13	13	14	15
$\max d(n)$	10752	17280	26880	41472	64512	103680
$\pi(n)$	Prime number theorem: $\pi(x) \sim x/\log(x)$					

Vimrc, Bashrc

```
source $VIMRUNTIME/mswin.vim
behave mswin
set mouse=a ci ai si nu ts=4 sw=4 is hls backup undofile
color slate
map <F7> : ! make %<<CR>
map <F8> : ! time ./%< <CR>
```

```
export CXXFLAGS='-g -Wall -Wextra -Wconversion -Wshadow -std=gnu++20'
ulimit -s 1048576
```

VSCode: 打开自动保存

STL 积分/求和

1. $\int_0^1 t^{x-1}(1-t)^{y-1}dt = beta(x,y) = \frac{\Gamma(x)\Gamma(y)}{\Gamma(x+y)}$
2. $\int_0^{+\infty} t^{num-1}e^{-t}dt = tgamma(num) = e^{lgamma(num)} = \Gamma(num)$
3. $\int_0^{phi} \frac{d\theta}{\sqrt{1-k^2\sin^2\theta}} = ellint\_1(k,phi)$
4. $\int_0^{phi} \sqrt{1-k^2\sin^2\theta}d\theta = ellint\_2(k,phi)$
5. $\int_{num}^{+\infty} \frac{e^{-t}}{t}dt = -expint(-num)$
6. $\sum_{n=1}^{+\infty} n^{-num} = riemann\_zeta(num)$
7. $\frac{2}{\sqrt{\pi}}\int_0^{arg} e^{-t^2}dt = erf(arg)$