Team Note of Titanic

nguyenphong233, tudiv, tuntun

ICPC Vietnam Regional 2025

Contents

1 DataStructure

1.1 Convex Hull Trick (Stack, LineContainer)

```cpp
struct Line{ // call init() before use
  ll a, b, c; // y = ax + b, c = line index
  Line(ll a, ll b, ll c) : a(a), b(b), c(c) {}
  ll f(ll x){ return a * x + b; }
};
vector<Line> v; int pv;
void init(){ v.clear(); pv = 0; }
int chk(const Line &a, const Line &b, const Line &c) const {
  return (__int128_t)(a.b - b.b) * (b.a - c.a) <=
  (__int128_t)(c.b - b.b) * (b.a - a.a);
}
void insert(Line l){
  if(v.size() > pv && v.back().a == l.a){ // fix if min query
    if(l.b < v.back().b) l = v.back();    v.pop_back();
  }
  while(v.size() >= pv+2 && chk(v[v.size()-2], v.back(), l))
  v.pop_back();
  v.push_back(l);
}
p query(ll x){ // if min query, then v[pv].f(x) >=
v[pv+1].f(x)
  while(pv+1 < v.size() && v[pv].f(x) <= v[pv+1].f(x)) pv++;
  return {v[pv].f(x), v[pv].c};
}
///// line container start (max query) /////
struct Line {
  mutable ll k, m, p;
  bool operator<(const Line& o) const { return k < o.k; }
  bool operator<(ll x) const { return p < x; }
}; // (for doubles, use inf = 1/.0, div(a,b) = a/b)
struct LineContainer : multiset<Line, less<>> {
  static const ll inf = LLONG_MAX; // div: floor
  ll div(ll a, ll b) { return a / b - ((a ^ b) < 0 && a % b);
  }
  bool isect(iterator x, iterator y) {
    if (y == end()) return x->p = inf, 0;
    if (x->k == y->k) x->p = x->m > y->m ? inf : -inf;
    else x->p = div(y->m - x->m, x->k - y->k);
    return x->p >= y->p;
  }
  void add(ll k, ll m) {
    auto z = insert({k, m, 0}), y = z++, x = y;
    while (isect(y, z)) z = erase(z);
    if (x != begin() && isect(--x, y)) isect(x, y =
    erase(y));
    while ((y = x) != begin() && (--x)->p >= y->p) isect(x,
    erase(y));
  } ll query(ll x) { assert(!empty());
    auto l = *lower_bound(x); return l.k * x + l.m; }
};
```

1.2 Persistent Segment Tree

```cpp
struct PSTNode{ // call init(root[0], s, e) before use
  PSTNode *l, *r; int v; PSTNode(){ l = r = nullptr; v = 0; }
}; PSTNode *root[101010];
```

```
PST(){ memset(root, 0, sizeof root); } // constructor
void init(PSTNode *node, int s, int e){
  if(s == e) return; int m = s + e >> 1;
  node->l = new PSTNode; node->r = new PSTNode;
  init(node->l, s, m); init(node->r, m+1, e);
}
void update(PSTNode *prv, PSTNode *now, int s, int e, int x){
  if(s == e){ now->v = prv ? prv->v + 1 : 1; return; }
  int m = s + e >> 1;
  if(x <= m){
    now->l = new PSTNode; now->r = prv->r;
    update(prv->l, now->l, s, m, x);
  }
  else{
    now->r = new PSTNode; now->l = prv->l;
    update(prv->r, now->r, m+1, e, x);
  } now->v = (now->l?now->l->v:0) + (now->r?now->r->v:0);
}
int kth(PSTNode *prv, PSTNode *now, int s, int e, int k){
  if(s == e) return s;
  int m = s + e >> 1, diff = now->l->v - prv->l->v;
  if(k <= diff) return kth(prv->l, now->l, s, m, k);
  else return kth(prv->r, now->r, m+1, e, k-diff);
}

struct Vertex {
    Vertex *l, *r;
    int sum;

    Vertex(int val) : l(nullptr), r(nullptr), sum(val) {}
    Vertex(Vertex *l, Vertex *r) : l(l), r(r), sum(0) {
        if (l) sum += l->sum;
        if (r) sum += r->sum;
    }
};
Vertex* build(int a[], int tl, int tr) {
    if (tl == tr)
        return new Vertex(a[tl]);
    int tm = (tl + tr) / 2;
    return new Vertex(build(a, tl, tm), build(a, tm+1, tr));
}
int get_sum(Vertex* v, int tl, int tr, int l, int r) {
    if (l > r)
        return 0;
    if (l == tl && tr == r)
        return v->sum;
    int tm = (tl + tr) / 2;
    return get_sum(v->l, tl, tm, l, min(r, tm))
         + get_sum(v->r, tm+1, tr, max(l, tm+1), r);
}
Vertex* update(Vertex* v, int tl, int tr, int pos, int
new_val) {
    if (tl == tr)
        return new Vertex(new_val);
    int tm = (tl + tr) / 2;
    if (pos <= tm)
        return new Vertex(update(v->l, tl, tm, pos, new_val),
        v->r);
    else
        return new Vertex(v->l, update(v->r, tm+1, tr, pos,
        new_val));
}
int find_kth(Vertex* vl, Vertex *vr, int tl, int tr, int k) {
    if (tl == tr)
        return tl;
    int tm = (tl + tr) / 2, left_count = vr->l->sum -
    vl->l->sum;
    if (left_count >= k)
        return find_kth(vl->l, vr->l, tl, tm, k);
    return find_kth(vl->r, vr->r, tm+1, tr, k-left_count);
}
```

1.3 Lazy LiChao Tree

```
/* get_point(x) : get min(f(x)), O(log X)
range_min(l,r) get min(f(x)), l<=x<=r, O(log X)
insert(l,r,a,b) : insert f(x)=ax+b, l<=x<=r, O(log^2 X)
add(l,r,a,b) : add f(x)=ax+b, l<=x<=r, O(log^2 X)
WARNING: a != 0인 add가 없을 때만 range_min 가능 */
template<typename T, T LE, T RI, T INF=(long long)(4e18)>
struct LiChao{
  struct Node{
    int l, r; T a, b, mn, aa, bb;
    Node(){ l = r = 0; a = 0; b = mn = INF; aa = bb = 0; }
    void apply(){ mn += bb; a += aa; b += bb; aa = bb = 0; }
    void add_lazy(T _aa, T _bb){ aa += _aa; bb += _bb; }
    T f(T x) const { return a * x + b; }
  }; vector<Node> seg; LiChao() : seg(2) {}
  void make_child(int n){
    if(!seg[n].l) seg[n].l = seg.size(), seg.emplace_back();
    if(!seg[n].r) seg[n].r = seg.size(), seg.emplace_back();
  }
  void push(int node, T s, T e){
    if(seg[node].aa || seg[node].bb){
      if(s != e){
        make_child(node);
        seg[seg[node].l].add_lazy(seg[node].aa,
        seg[node].bb);
        seg[seg[node].r].add_lazy(seg[node].aa,
        seg[node].bb);
      } seg[node].apply();
    }
  }
  void insert(T l, T r, T a, T b, int node=1, T s=LE, T
  e=RI){
    if(r < s || e < l || l > r) return;
    make_child(node); push(node, s, e); T m = (s + e) >> 1;
    seg[node].mn=min({seg[node].mn,
    a*max(s,l)+b,a*min(e,r)+b});
    if(s < l || r < e){
      if(l <= m) insert(l, r, a, b, seg[node].l, s, m);
      if(m+1 <= r) insert(l, r, a, b, seg[node].r, m+1, e);
      return;
    }
    T &sa = seg[node].a, &sb = seg[node].b;
    if(a*s+b < sa*s+sb) swap(a, sa), swap(b, sb);
    if(a*e+b >= sa*e+sb) return;
    if(a*m+b < sa*m+sb){
      swap(a,sa); swap(b,sb);
      insert(l,r,a,b,seg[node].l,s,m);
    } else insert(l, r, a, b, seg[node].r, m+1, e);
  }
  void add(T l, T r, T a, T b, int node=1, T s=LE, T e=RI){
    if(r < s || e < l || l > r) return;
    make_child(node); push(node, s, e); T m = (s + e) >> 1;
    if(s < l || r < e){
      insert(s, m, seg[node].a, seg[node].b, seg[node].l, s,
      m);

      insert(m+1,e,seg[node].a,seg[node].b,seg[node].r,m+1,e);
      seg[node].a = 0; seg[node].b = seg[node].mn = INF;
      if(l <= m) add(l, r, a, b, seg[node].l, s, m);
      if(m+1 <= r) add(l, r, a, b, seg[node].r, m+1, e);
      seg[node].mn=min(seg[seg[node].l].mn,
      seg[seg[node].r].mn);
      return;
    } seg[node].add_lazy(a, b); push(node, s, e);
  }
  T get_point(T x, int node=1, T s=LE, T e=RI){
    if(node == 0) return INF; push(node, s, e);
    T m = (s + e) >> 1, res = seg[node].f(x);
    if(x <= m) return min(res, get_point(x, seg[node].l, s,
    m));
    else return min(res, get_point(x, seg[node].r, m+1, e));
  }
  T range_min(T l, T r, int node=1, T s=LE, T e=RI){
    if(node == 0 || r < s || e < l || l > r) return INF;
    push(node, s, e); T m = (s + e) >> 1;
    if(l <= s && e <= r) return seg[node].mn;
    return min({ seg[node].f(max(s,l)),
    seg[node].f(min(e,r)),
       range_min(l, r, seg[node].l, s, m),
       range_min(l, r, seg[node].r, m+1, e) });
  }
};
```

2 Geometry

2.1 $O(\log N)$ Point in Convex Polygon

```
bool Check(const vector<Point> &v, const Point &pt){
  if(CCW(v[0], v[1], pt) < 0) return false;
  int l = 1, r = v.size() - 1;
  while(l < r){
    int m = l + r + 1 >> 1;
    if(CCW(v[0], v[m], pt) >= 0) l = m; else r = m - 1;
  }
  if(l == v.size() - 1) return CCW(v[0], v.back(), pt) == 0
  && v[0] <= pt && pt <= v.back();
  return CCW(v[0], v[l], pt) >= 0 && CCW(v[l], v[l+1], pt) >=
  0 && CCW(v[l+1], v[0], pt) >= 0;
}
```

2.2 Segment Distance, Segment Reflect

```
double Proj(Point a, Point b, Point c){
```

```cpp
  ll t1 = (b - a) * (c - a), t2 = (a - b) * (c - b);
  if(t1 * t2 >= 0 && CCW(a, b, c) != 0)
    return abs(CCW(a, b, c)) / sqrt(Dist(a, b));
  else return 1e18; // INF
}
double SegmentDist(Point a[2], Point b[2]){
  double res = 1e18; // NOTE: need to check intersect
  for(int i=0; i<4; i++)
  res=min(res,sqrt(Dist(a[i/2],b[i%2])));
  for(int i=0; i<2; i++) res = min(res, Proj(a[0], a[1],
  b[i]));
  for(int i=0; i<2; i++) res = min(res, Proj(b[0], b[1],
  a[i]));
  return res;
}
P Reflect(P p1, P p2, P p3){ // line p1-p2, point p3
  auto [x1,y1] = p1; auto [x2,y2] = p2; auto [x3,y3] = p3;
  auto a = y1-y2, b = x2-x1, c = x1 * (y2-y1) + y1 * (x1-x2);
  auto d = a * y3 - b * x3;
  T x = -(a*c+b*d) / (a*a+b*b), y = (a*d-b*c) / (a*a+b*b);
  return 2 * P(x,y) - p3;
}
```

2.3 Tangent Series

```cpp
template<bool UPPER=true> // O(log N)
Point GetPoint(const vector<Point> &hull, real_t slope){
    auto chk = [slope](real_t dx, real_t dy){ return UPPER ?
    dy >= slope * dx : dy <= slope * dx; };
    int l = -1, r = hull.size() - 1;
    while(l + 1 < r){
        int m = (l + r) / 2;
        if(chk(hull[m+1].x - hull[m].x, hull[m+1].y -
        hull[m].y)) l = m; else r = m;
    }
    return hull[r];
}
int ConvexTangent(const vector<Point> &v, const Point &pt,
int up=1){ //given outer point, O(log N)
  auto sign = [&](ll c){ return c>0 ? up : c==0 ? 0 : -up; };
  auto local = [&](Point p, Point a, Point b, Point c){
    return sign(CCW(p, a, b)) <= 0 && sign(CCW(p, b, c)) >=
    0;
  }; // assert(v.size() >= 2);
  int n = v.size() - 1, s = 0, e = n, m;
  if(local(pt, v[1], v[0], v[n-1])) return 0;
  while(s + 1 < e){
    m = (s + e) / 2;
    if(local(pt, v[m-1], v[m], v[m+1])) return m;
    if(sign(CCW(pt, v[s], v[s+1])) < 0){ // up
      if(sign(CCW(pt, v[m], v[m+1])) > 0) e = m;
      else if(sign(CCW(pt, v[m], v[s])) > 0) s = m; else e =
      m;
    }
    else{ // down
      if(sign(CCW(pt, v[m], v[m+1])) < 0) s = m;
      else if(sign(CCW(pt, v[m], v[s])) < 0) s = m; else e =
      m;
    }
  }
  if(s && local(pt, v[s-1], v[s], v[s+1])) return s;
  if(e != n && local(pt, v[e-1], v[e], v[e+1])) return e;
  return -1;
}
int Closest(const vector<Point> &v, const Point &out, int
now){
  int prv = now > 0 ? now-1 : v.size()-1, nxt = now+1 <
  v.size() ? now+1 : 0, res = now;
  if(CCW(out, v[now], v[prv]) == 0 && Dist(out, v[res]) >
  Dist(out, v[prv])) res = prv;
  if(CCW(out, v[now], v[nxt]) == 0 && Dist(out, v[res]) >
  Dist(out, v[nxt])) res = nxt;
  return res; // if parallel, return closest point to out
} // int point_idx =  Closest(convex_hull, pt,
ConvexTangent(hull + hull[0], pt, +-1) % N);
/////////
double polar(pdd x){ return atan2(x.second, x.first); }
int tangent(circle &A, circle &B, pdd des[4]){ // return
angle
  int top = 0; // outer
  double d = size(A.O - B.O), a = polar(B.O - A.O), b = PI +
  a;
  double t = sq(d) - sq(A.r - B.r);
  if (t >= 0){
    t = sqrt(t); double p = atan2(B.r - A.r, t);
    des[top++] = pdd(a + p + PI / 2, b + p - PI / 2);
    des[top++] = pdd(a - p - PI / 2, b - p + PI / 2);
  }
  t = sq(d) - sq(A.r + B.r); // inner
  if (t >= 0){ t = sqrt(t);
    double p = atan2(B.r + A.r, t);
    des[top++] = pdd(a + p - PI / 2, b + p - PI / 2);
    des[top++] = pdd(a - p + PI / 2, b - p + PI / 2);
  }
  return top;
}
pair<T, T> CirclePointTangent(P o, double r, P p){
  T op=D1(p,o), u=atan2l(p.y-o.y, p.x-o.x), v=acosl(r/op);
  return {u + v, u - v};
} // COORD 1e4 EPS 1e-7 / COORD 1e3 EPS 1e-9 with circleLine
```

2.4 Intersect Series

```cpp
// 0: not intersect, -1: infinity, 4: intersect
// 1/2/3: intersect first/second/both segment corner
// flag, xp, xq, yp, yq : (xp / xq, yp / yq)
using T = __int128_t; // T <= O(COORD^3)
tuple<int,T,T,T,T> SegmentIntersect(P s1, P e1, P s2, P e2){
  if(!Intersect(s1, e1, s2, e2)) return {0, 0, 0, 0, 0};
  auto det = (e1 - s1) / (e2 - s2);
  if(!det){
      if(s1 > e1) swap(s1, e1);
      if(s2 > e2) swap(s2, e2);
      if(e1 == s2) return {3, e1.x, 1, e1.y, 1};
      if(e2 == s1) return {3, e2.x, 1, e2.y, 1};
      return {-1, 0, 0, 0, 0};
  }
  T p = (s2 - s1) / (e2 - s2), q = det, flag = 0;
  T xp = s1.x * q + (e1.x - s1.x) * p, xq = q;
  T yp = s1.y * q + (e1.y - s1.y) * p, yq = q;
  if(xp%xq || yp%yq) return {4,xp,xq,yp,yq};//gcd?
  //if(xq < 0) xp=-xp, xq=-xq; if(yq < 0) yp=-yp, yq=-yq
  //gcd?
  xp /= xq; yp /= yq;
  if(s1.x == xp && s1.y == yp) flag |= 1;
  if(e1.x == xp && e1.y == yp) flag |= 1;
  if(s2.x == xp && s2.y == yp) flag |= 2;
  if(e2.x == xp && e2.y == yp) flag |= 2;
  return {flag ? flag : 4, xp, 1, yp, 1};
}
P perp() const { return P(-y, x); }
#define arg(p, q) atan2(p.cross(q), p.dot(q))
bool circleIntersect(P a,P b,double r1,double r2,pair<P, P>*
out){
  if (a == b) { assert(r1 != r2); return false; }
  P vec = b-a; double d2 = vec.dist2(), sum = r1+r2, dif =
  r1-r2;
  double p = (d2 + r1*r1 - r2*r2)/(d2*2), h2 = r1*r1 -
  p*p*d2;
  if (sum*sum < d2 || dif*dif > d2) return false; // use EPS
  plz...
  P mid = a + vec*p, per = vec.perp() * sqrt(fmax(0, h2) /
  d2);
  *out = {mid + per, mid - per}; return true;
}
vector<P> circleLine(P c, double r, P a, P b) {
    P ab = b - a, p = a + ab * (c-a) * ab / D2(ab);
    T s = (b - a) / (c - a), h2 = r*r - s*s / D2(ab);
    if (abs(h2) < EPS) return {p}; if (h2 < 0) return {};
    P h = ab / D1(ab) * sqrtl(h2); return {p - h, p + h};
} // use circleLine if you use double...
int CircleLineIntersect(P o, T r, P p1, P p2, bool segment){
  P s = p1, d = p2 - p1; // line : s + kd, int support
  T a = d * d, b = (s - o) * d * 2, c = D2(s, o) - r * r;
  T det = b * b - 4 * a * c; // solve ak^2 + bk + c = 0, a >
  0
  if(!segment) return Sign(det) + 1;
  if(det <= 0) return det ? 0 : 0 <= -b && -b <= a + a;
  bool f11 = b <= 0 || b * b <= det;
  bool f21 = b <= 0 && b * b >= det;
  bool f12 = a+a+b >= 0 && det <= (a+a+b) * (a+a+b);
  bool f22 = a+a+b >= 0 || det >= (a+a+b) * (a+a+b);
  return (f11 && f12) + (f21 && f22);
} // do not use this if you want to use double...
double circlePoly(P c, double r, vector<P> ps){ // return
area
  auto tri = [&](P p, P q) { // ps must be ccw polygon
    auto r2 = r * r / 2; P d = q - p;
    auto a = d.dot(p)/d.dist2(), b =
    (p.dist2()-r*r)/d.dist2();
    auto det = a * a - b;
    if (det <= 0) return arg(p, q) * r2;
    auto s = max(0., -a-sqrt(det)), t = min(1.,
    -a+sqrt(det));
```

```
    if (t < 0 || 1 <= s) return arg(p, q) * r2;
    P u = p + d * s, v = p + d * t;
    return arg(p,u) * r2 + u.cross(v)/2 + arg(v,q) * r2;
  };
  auto sum = 0.0;
  rep(i,0,sz(ps)) sum += tri(ps[i] - c, ps[(i+1)%sz(ps)] -
  c);
  return sum;
}
// extrVertex: point of hull, max projection onto line
#define cmp(i,j)
sgn(dir.perp().cross(poly[(i)%n]-poly[(j)%n]))
#define extr(i) cmp(i + 1, i) >= 0 && cmp(i, i - 1 + n) < 0
int extrVertex(vector<P>& poly, P dir) {
  int n = sz(poly), lo = 0, hi = n;
  if (extr(0)) return 0;
  while (lo + 1 < hi) {
    int m = (lo + hi) / 2; if (extr(m)) return m;
    int ls = cmp(lo + 1, lo), ms = cmp(m + 1, m);
    (ls < ms || (ls == ms && ls == cmp(lo, m)) ? hi : lo) =
    m;
  }
  return lo;
}
//(-1,-1): no collision
//(i,-1): touch corner
//(i,i): along side (i,i+1)
//(i,j): cross (i,i+1)and(j,j+1)
//(i,i+1): cross corner i
// O(log n), ccw no colinear point convex polygon
// P perp() const { return P(-y, x); }
#define cmpL(i) sgn(a.cross(poly[i], b))
array<int, 2> lineHull(P a, P b, vector<P>& poly) { // O(log
N)
  int endA = extrVertex(poly, (a - b).perp());
  int endB = extrVertex(poly, (b - a).perp());
  if (cmpL(endA) < 0 || cmpL(endB) > 0) return {-1, -1};
  array<int, 2> res;
  rep(i,0,2) {
    int lo = endB, hi = endA, n = sz(poly);
    while ((lo + 1) % n != hi) {
      int m = ((lo + hi + (lo < hi ? 0 : n)) / 2) % n;
      (cmpL(m) == cmpL(endB) ? lo : hi) = m;
    }
    res[i] = (lo + !cmpL(hi)) % n;
    swap(endA, endB);
  }
  if (res[0] == res[1]) return {res[0], -1};
  if (!cmpL(res[0]) && !cmpL(res[1]))
    switch ((res[0] - res[1] + sz(poly) + 1) % sz(poly)) {
      case 0: return {res[0], res[0]};
      case 2: return {res[1], res[1]};
    }
  return res;
}
```

2.5 $O(N^2 \log N)$ Circles Area

```
ld NormAngle(ld v){while(v < -EPS) v += M_PI * 2;
  while(v > M_PI * 2 + EPS) v -= M_PI * 2;
  return v; }
ld TwoCircleUnion(const Circle &p, const Circle &q) {
  ld d = D1(p.o - q.o); if(d >= p.r+q.r-EPS) return 0;
  else if(d <= abs(p.r-q.r)+EPS) return
  pow(min(p.r,q.r),2)*PI;
  ld pc = (p.r*p.r + d*d - q.r*q.r) / (p.r*d*2), pa =
  acosl(pc);
  ld qc = (q.r*q.r + d*d - p.r*p.r) / (q.r*d*2), qa =
  acosl(qc);
  ld ps = p.r*p.r*pa - p.r*p.r*sin(pa*2)/2;
  ld qs = q.r*q.r*qa - q.r*q.r*sin(qa*2)/2;
  return ps + qs; }
ld TwoCircleIntersect(P p1, P p2, ld r1, ld r2){
  auto f = [](ld a, ld b, ld c){
    return acosl((a*a + b*b - c*c) / (2*a*b)); };
  ld d = D1(p1, p2); if(d + EPS > r1 + r2) return 0;
  if(d < abs(r1-r2) + EPS) return min(r1,r2)*min(r1,r2)*M_PI;
  ld t1 = f(r1, d, r2), t2 = f(r2, d, r1);
  return r1*r1*(t1-sinl(t1)*cosl(t1))
       + r2*r2*(t2-sinl(t2)*cosl(t2)); }
vector<pair<double, double>> CoverSegment(Cir a, Cir b) {
  double d = abs(a.o - b.o); vector<pair<double, double>>
  res;
  if(sign(a.r + b.r - d) == 0); /* skip */
  else if(d <= abs(a.r - b.r) + eps) {
    if (a.r < b.r) res.emplace_back(0, 2 * pi);
  } else if(d < abs(a.r + b.r) - eps) {
    double o = acos((a.r*a.r + d*d - b.r*b.r) / (2 * a.r *
    d));
    double z = NormAngle(atan2((b.o - a.o).y, (b.o -
    a.o).x));
    double l = NormAngle(z - o), r = NormAngle(z + o);
    if(l > r) res.emplace_back(l, 2*pi),
    res.emplace_back(0,r);
    else res.emplace_back(l, r);
  } return res;
}// circle should be identical
double CircleUnionArea(vector<Cir> c) {
  int n = c.size(); double a = 0, w;
  for (int i = 0; w = 0, i < n; ++i) {
    vector<pair<double, double>> s = {{2 * pi, 9}}, z;
    for (int j = 0; j < n; ++j) if (i != j) {
      z = CoverSegment(c[i], c[j]);
      for (auto &e : z) s.push_back(e); } /* for j */
    sort(s.begin(), s.end());
    auto F = [&] (double t) { return c[i].r * (c[i].r * t +
    c[i].o.x * sin(t) - c[i].o.y * cos(t)); };
    for (auto &e : s) {
      if (e.first > w) a += F(e.first) - F(w);
      w = max(w, e.second); } /* for e */
  } return a * 0.5; }
```

2.6 Segment In Polygon

```
// WARNING: C.push_back(C[0]) before call function
bool segment_in_polygon_non_strict(vector<P> &C, P s, P e){
    if(!pip(C, s) || !pip(C, e)) return false;
    if(s == e) return true; P d = e - s;
    vector<pair<frac,int>> v; auto g=raypoints(C, s, d, v);
    for(auto [fr,ev] : v){ // in(06) out(27)
        if(fr.first < 0 || g < fr) continue;
        if(ev == 4) return false; // pass outside corner
        if(fr < g && (ev == 2 || ev == 7)) return false;
        if(0 < fr.first && (ev == 0 || ev == 6)) return
        false;
    } return true;
}
```

2.7 Polygon Cut, Center, Union

```
// Returns the polygon on the left of line l
// *: dot product, ^: cross product
// l = p + d*t, l.q() = l + d
// doubled_signed_area(p,q,r) = (q-p) ^ (r-p)
template<class T> vector<point<T>> polygon_cut(const
vector<point<T>> &a, const line<T> &l){
  vector<point<T>> res;
  for(auto i = 0; i < (int)a.size(); ++ i){
    auto cur = a[i], prev = i ? a[i - 1] : a.back();
    bool side = doubled_signed_area(l.p, l.q(), cur) > 0;
    if(side != (doubled_signed_area(l.p, l.q(), prev) > 0))
      res.push_back(l.p + (cur - l.p ^ prev - cur) / (l.d ^
      prev - cur) * l.d);
    if(side) res.push_back(cur);
  }
  return res;
}
P polygonCenter(const vector<P>& v){ // center of mass
  P res(0, 0); double A = 0;
  for (int i = 0, j = sz(v) - 1; i < sz(v); j = i++) {
    res = res + (v[i] + v[j]) * v[j].cross(v[i]);
    A += v[j].cross(v[i]);
  } return res / A / 3;
}
// O(points^2), area of union of n polygon, ccw polygon
int sideOf(P s, P e, P p) { return sgn((e-s)/(p-s)); }
int sideOf(const P& s, const P& e, const P& p, double eps) {
  auto a = (e-s)/(p-s); auto l=D1(e-s) * eps;
  return (a > l) - (a < -l);
}
double rat(P a, P b) { return sgn(b.x) ? a.x/b.x : a.y/b.y; }
double polyUnion(vector<vector<P>>& poly) { // (points)^2
  double ret = 0;
  rep(i,0,sz(poly)) rep(v,0,sz(poly[i])) {
    P A = poly[i][v], B = poly[i][(v + 1) % sz(poly[i])];
    vector<pair<double, int>> segs = {{0, 0}, {1, 0}};
    rep(j,0,sz(poly)) if (i != j) { // START
      rep(u,0,sz(poly[j])) {
        P C = poly[j][u], D = poly[j][(u + 1) % sz(poly[j])];
        int sc = sideOf(A, B, C), sd = sideOf(A, B, D);
        if (sc != sd) {
          double sa = C.cross(D, A), sb = C.cross(D, B);
          if (min(sc, sd) < 0)
            segs.emplace_back(sa / (sa - sb), sgn(sc - sd));
        }
```

```
      else if (!sc && !sd && j<i && sgn((B-A).dot(D-C))>0){
        segs.emplace_back(rat(C - A, B - A), 1);
        segs.emplace_back(rat(D - A, B - A), -1);
    } /*else if*/ } /*rep u*/ } /*rep j*/ // END
    sort(all(segs));
    for (auto& s : segs) s.first = min(max(s.first, 0.0),
    1.0);
    double sum = 0; int cnt = segs[0].second;
    rep(j,1,sz(segs)) {
      if (!cnt) sum += segs[j].first - segs[j - 1].first;
      cnt += segs[j].second;
    }
    ret += A.cross(B) * sum;
  } return abs(ret) / 2;
}
```

2.8 Polycon Raycast

```
// ray A + kd and CCW polygon C, return events {k, event_id}
// 0: out->line / 1: in->line / 2: line->out / 3: line->in
// 4: pass corner outside / 5: pass corner inside / 6: out ->
in / 7: in -> out
// WARNING: C.push_back(C[0]) before use, ccw, no colinear
struct frac{
  ll first, second; frac(){}
  frac(ll a, ll b) : first(a), second(b) {
    if( b < 0 ) first = -a, second = -b; // operator cast
    int128
  } double v(){ return 1.*first/second; } // operator <,<=,==
}; // assert(d != P(0,0));
frac raypoints(const vector<P> &C, P A, P d,
vector<pair<frac, int>> &R){ vector<pair<frac, int>> L;
  auto g = gcd(abs(d.x), abs(d.y)); d.x /= g, d.y /= g;
  for(int i = 0; i+1 < C.size(); i++){ P v = C[i+1] - C[i];
    int a = sign(d/(C[i]-A)), b = sign(d/(C[i+1]-A));
    if(a == 0)L.emplace_back(frac(d*(C[i]-A)/size2(d), 1),
    b);
    if(b == 0)L.emplace_back(frac(d*(C[i+1]-A)/size2(d),
    1),a);
    if(a*b == -1) L.emplace_back(frac((A-C[i])/v, v/d), 6);
  } sort(L.begin(), L.end());
  for(int i = 0; i < L.size(); i++){
    // assert(i+2 >= L.size() || !(L[i].first ==
    L[i+2].first));

    if(i+1<L.size()&&L[i].first==L[i+1].first&&L[i].second!=6){
      int a = L[i].second, b = L[i+1].second;
      R.emplace_back(L[i++].first, a*b? a*b > 0?
      4:6:(1-a-b)/2);
    } /* end if */ else R.push_back(L[i]); } /* end for */
  int state = 0; // 0: out, 1: in, 2: line+ccw, 3: line+cw
  for(auto &[_,n] : R){
    if( n == 6 ) n ^= state, state ^= 1;
    else if( n == 4 ) n ^= state;
    else if( n == 0 ) n = state, state ^= 2;
    else if( n == 1 ) n = state^(state>>1), state ^= 3;
  } return frac(g, 1);
}
bool visible(const vector<P> &C, P A, P B){
  if( A == B ) return true;//return outside?
  char I[4] = "356", O[4] = "157";
  vector<pair<frac, int>> R; vector<frac> E;
  frac s = frac(0, 1), e = raypoints(C, A, B-A, R);
  for(auto [f,n] : R){
    if(*find(O, O+3, n+'0')) E.push_back(f);
    if(*find(I, I+3, n+'0')) E.push_back(f);
  }
  for(int j = 0; j < E.size(); j += 2) if( !(e <= E[j] ||
  E[j+1] <= s) ) return false;
  return true; }
```

2.9 2-SAT

```
int SZ; vector<vector<int>> G1, G2;
void Init(int n){ SZ = n; G1 = G2 =
vector<vector<int>>(SZ*2); }
int New(){
  for(int i=0;i<2;i++) G1.emplace_back(), G2.emplace_back();
  return SZ++; }
void AddEdge(int s, int e){ G1[s].push_back(e);
G2[e].push_back(s); }
// T(x) = x << 1, F(x) = x << 1 | 1, I(x) = x ^ 1
void AddCNF(int a, int b){ AddEdge(I(a), b); AddEdge(I(b),
a); }
void MostOne(vector<int> vec){ compress(vec);
  for(int i=0; i<vec.size(); i++){
    int now = New();
    AddEdge(vec[i], T(now)); AddEdge(F(now), I(vec[i]));
    if(i == 0) continue;
    AddEdge(T(now-1), T(now)); AddEdge(F(now), F(now-1));
    AddEdge(T(now-1), I(vec[i])); AddEdge(vec[i], F(now-1));
  }
}
```

2.10 Horn SAT

```
/* n : numer of variance
{}, 0 : x1 | {0, 1}, 2 : (x1 and x2) => x3, (-x1 or -x2 or
x3)
fail -> empty vector */
vector<int> HornSAT(int n, const vector<vector<int>> &cond,
const vector<int> &val){
  int m = cond.size(); vector<int> res(n), margin(m), stk;
  vector<vector<int>> gph(n);
  for(int i=0; i<m; i++){
    margin[i] = cond[i].size();
    if(cond[i].empty()) stk.push_back(i);
    for(auto j : cond[i]) gph[j].push_back(i);
  }
  while(!stk.empty()){
    int v = stk.back(), h = val[v]; stk.pop_back();
    if(h < 0) return vector<int>();
    if(res[h]) continue; res[h] = 1;
    for(auto i : gph[h]) if(!--margin[i]) stk.push_back(i);
  } return res;
}
```

2.11 Convex Hull

```
// ray A + kd and CCW polygon C, return events {k, event_id}
// 0: out->line / 1: in->line / 2: line->out / 3: line->in
// 4: pass corner outside / 5: pass corner inside / 6: out ->
in / 7: in -> out
// WARNING: C.push_back(C[0]) before use, ccw, no colinear
struct frac{
  ll first, second; frac(){}
  frac(ll a, ll b) : first(a), second(b) {
    if( b < 0 ) first = -a, second = -b; // operator cast
    int128
  } double v(){ return 1.*first/second; } // operator <,<=,==
}; // assert(d != P(0,0));
frac raypoints(const vector<P> &C, P A, P d,
vector<pair<frac, int>> &R){ vector<pair<frac, int>> L;
  auto g = gcd(abs(d.x), abs(d.y)); d.x /= g, d.y /= g;
  for(int i = 0; i+1 < C.size(); i++){ P v = C[i+1] - C[i];
    int a = sign(d/(C[i]-A)), b = sign(d/(C[i+1]-A));
    if(a == 0)L.emplace_back(frac(d*(C[i]-A)/size2(d), 1),
    b);
    if(b == 0)L.emplace_back(frac(d*(C[i+1]-A)/size2(d),
    1),a);
    if(a*b == -1) L.emplace_back(frac((A-C[i])/v, v/d), 6);
  } sort(L.begin(), L.end());
  for(int i = 0; i < L.size(); i++){
    // assert(i+2 >= L.size() || !(L[i].first ==
    L[i+2].first));

    if(i+1<L.size()&&L[i].first==L[i+1].first&&L[i].second!=6){
      int a = L[i].second, b = L[i+1].second;
      R.emplace_back(L[i++].first, a*b? a*b > 0?
      4:6:(1-a-b)/2);
    } /* end if */ else R.push_back(L[i]); } /* end for */
  int state = 0; // 0: out, 1: in, 2: line+ccw, 3: line+cw
  for(auto &[_,n] : R){
    if( n == 6 ) n ^= state, state ^= 1;
    else if( n == 4 ) n ^= state;
    else if( n == 0 ) n = state, state ^= 2;
    else if( n == 1 ) n = state^(state>>1), state ^= 3;
  } return frac(g, 1);
}
bool visible(const vector<P> &C, P A, P B){
  if( A == B ) return true;//return outside?
  char I[4] = "356", O[4] = "157";
  vector<pair<frac, int>> R; vector<frac> E;
  frac s = frac(0, 1), e = raypoints(C, A, B-A, R);
  for(auto [f,n] : R){
    if(*find(O, O+3, n+'0')) E.push_back(f);
    if(*find(I, I+3, n+'0')) E.push_back(f);
  }
  for(int j = 0; j < E.size(); j += 2) if( !(e <= E[j] ||
  E[j+1] <= s) ) return false;
  return true; }
```

2.12 Heavy Light Decomposition

```
struct HLD { // 0-based, remember to build
  int n, _id;
  vector <vector <int>> g;
```

```
  vector <int> dep, pa, tsz, ch, hd, id;
  void dfs(int v, int p) {
    dep[v] = ~p ? dep[p] + 1 : 0;
    pa[v] = p, tsz[v] = 1, ch[v] = -1;
    for (int u : g[v]) if (u != p) {
      dfs(u, v);
      if (ch[v] == -1 || tsz[ch[v]] < tsz[u])
        ch[v] = u;
      tsz[v] += tsz[u];
    }
  }
  void hld(int v, int p, int h) {
    hd[v] = h, id[v] = _id++;
    if (~ch[v]) hld(ch[v], v, h);
    for (int u : g[v]) if (u != p && u != ch[v])
      hld(u, v, u);
  }
  vector <pii> query(int u, int v) {
    vector <pii> ans;
    while (hd[u] != hd[v]) {
      if (dep[hd[u]] > dep[hd[v]]) swap(u, v);
      ans.emplace_back(id[hd[v]], id[v] + 1);
      v = pa[hd[v]];
    }
    if (dep[u] > dep[v]) swap(u, v);
    ans.emplace_back(id[u], id[v] + 1);
    return ans;
  }
  void build() {
    for (int i = 0; i < n; ++i) if (id[i] == -1)
      dfs(i, -1), hld(i, -1, i);
  }
  void add_edge(int u, int v) {
    g[u].pb(v), g[v].pb(u); }
  HLD (int _n) : n(_n), _id(0), g(n), dep(n), pa(n),
    tsz(n), ch(n), hd(n), id(n, -1) {}
};
```

2.13 Centroid Decomposition

```
struct CD { // 0-based, remember to build
  int n, lg; // pa, dep are centroid tree attributes
  vector <vector <int>> g, dis;
  vector <int> pa, tsz, dep, vis;
  void dfs1(int v, int p) {
    tsz[v] = 1;
    for (int u : g[v]) if (u != p && !vis[u])
      dfs1(u, v), tsz[v] += tsz[u];
  }
  int dfs2(int v, int p, int _n) {
    for (int u : g[v])
      if (u != p && !vis[u] && tsz[u] > _n / 2)
        return dfs2(u, v, _n);
    return v;
  }
  void dfs3(int v, int p, int d) {
    dis[v][d] = ~p ? dis[p][d] + 1 : 0;
    for (int u : g[v]) if (u != p && !vis[u])
      dfs3(u, v, d);
  }
  void cd(int v, int p, int d) {
    dfs1(v, -1), v = dfs2(v, -1, tsz[v]);
    vis[v] = true, pa[v] = p, dep[v] = d;
    dfs3(v, -1, d);
    for (int u : g[v]) if (!vis[u])
      cd(u, v, d + 1);
  }
  void build() { cd(0, -1, 0); }
  void add_edge(int u, int v) {
    g[u].pb(v), g[v].pb(u); }
  CD (int _n) : n(_n), lg(__lg(n) + 1), g(n),
    dis(n, vector <int>(lg)), pa(n), tsz(n),
    dep(n), vis(n) {}
};
```

2.14 SCC

```
struct SCC {
  int n, nscc, _id;
  vector <vector <int>> g;
  vector <int> dep, low, scc_id, stk;
  void dfs(int v) {
    dep[v] = low[v] = _id++, stk.pb(v);
    for (int u : g[v]) if (scc_id[u] == -1) {
      if (low[u] == -1) dfs(u);
      low[v] = min(low[v], low[u]);
    }
    if (low[v] == dep[v]) {
      int id = nscc++, x;
      do {
        x = stk.back(), stk.pop_back(), scc_id[x] = id;
      } while (x != v);
    }
  }
  void build() {
    for (int i = 0; i < n; ++i) if (low[i] == -1)
      dfs(i);
  }
  void add_edge(int u, int v) { g[u].pb(v); }
  SCC (int _n) : n(_n), nscc(0), _id(0), g(n), dep(n),
    low(n, -1), scc_id(n, -1), stk() {}
};
```

2.15 $O(3^{V/3})$ Maximal Clique

```
using B = bitset<128>; template<typename F> //0-based
void maximal_cliques(vector<B>&g,F f,B P=~B(),B X={},B R={}){
  if(!P.any()){ if(!X.any()) f(R); return; }
  auto q = (P|X)._Find_first(); auto c = P & ~g[q];
  for(int i=0; i<g.size(); i++) if(c[i]) {
    R[i] = 1; cliques(g, f, P&g[i], X&g[i], R);
    R[i]=P[i]=0; X[i] = 1; } // faster for sparse gph
} // undirected, self loop not allowed, O(3^{n/3})
B max_independent_set(vector<vector<int>> g){ //g=adj matrix
  int n = g.size(), i, j; vector<B> G(n); B res{};
  auto chk_mx = [&](B a){ if(a.count()>res.count()) res=a; };
  for(i=0; i<n; i++) for(int j=0; j<n; j++)
    if(i!=j && !g[i][j])G[i][j]=1;
  cliques(G, chk_mx); return res; }
```

2.16 $O(V \log V)$ Tree Isomorphism

```
struct Tree{ // (M1,M2)=(1e9+7, 1e9+9), P1,P2 = random int
array(sz >= N+2)
  int N; vector<vector<int>> G; vector<pair<int,int>> H;
  vector<int> S, C; // size,centroid
  Tree(int N) : N(N), G(N+2), H(N+2), S(N+2) {}
  void addEdge(int s, int e){ G[s].push_back(e);
  G[e].push_back(s); }
  int getCentroid(int v, int b=-1){
    S[v] = 1; // do not merge if-statements
    for(auto i : G[v]) if(i!=b) if(int now=getCentroid(i,v);
    now<=N/2) S[v]+=now; else break;
    if(N - S[v] <= N/2) C.push_back(v); return S[v] = S[v];
  }
  int init(){
    getCentroid(1); if(C.size() == 1) return C[0];
    int u = C[0], v = C[1], add = ++N;
    G[u].erase(find(G[u].begin(), G[u].end(), v));
    G[v].erase(find(G[v].begin(), G[v].end(), u));
    G[add].push_back(u); G[u].push_back(add);
    G[add].push_back(v); G[v].push_back(add);
    return add;
  }
  pair<int,int> build(const vector<ll> &P1, const vector<ll>
  &P2, int v, int b=-1){
    vector<pair<int,int>> ch; for(auto i : G[v]) if(i != b)
    ch.push_back(build(P1, P2, i, v));
    ll h1 = 0, h2 = 0; stable_sort(ch.begin(), ch.end());
    if(ch.empty()){ return {1, 1}; }
    for(int i=0; i<ch.size(); i++)
    h1=(h1+(ch[i].first^P1[P1.size()-1-i])*P1[i])%M1,
    h2=(h2+(ch[i].second^P2[P2.size()-1-i])*P2[i])%M2;
    return H[v] = {h1, h2};
  }
  int build(const vector<ll> &P1, const vector<ll> &P2){
    int rt = init(); build(P1, P2, rt); return rt;
  }
};
```

2.17 $O(E\sqrt{V})$ Bipartite Matching, Konig, Dilworth

```
struct HopcroftKarp{
  int n, m; vector<vector<int>> g;
  vector<int> dst, le, ri; vector<char> visit, track;
  HopcroftKarp(int n, int m) : n(n), m(m), g(n), dst(n),
  le(n, -1), ri(m, -1), visit(n), track(n+m) {}
  void add_edge(int s, int e){ g[s].push_back(e); }
  bool bfs(){ bool res = false; queue<int> que;
    fill(dst.begin(), dst.end(), 0);
    for(int i=0; i<n; i++)if(le[i] ==
    -1)que.push(i),dst[i]=1;
    while(!que.empty()){ int v = que.front(); que.pop();
      for(auto i : g[v]){
        if(ri[i] == -1) res = true;
        else
        if(!dst[ri[i]])dst[ri[i]]=dst[v]+1,que.push(ri[i]);
```

```
      }
    }
    return res;
  }
  bool dfs(int v){
    if(visit[v]) return false; visit[v] = 1;
    for(auto i : g[v]){
      if(ri[i] == -1 || !visit[ri[i]] && dst[ri[i]] == dst[v]
      + 1 && dfs(ri[i])){ le[v] = i; ri[i] = v; return true;
      }
    } return false;
  }
  int maximum_matching(){
    int res = 0; fill(all(le), -1); fill(all(ri), -1);
    while(bfs()){
      fill(visit.begin(), visit.end(), 0);
      for(int i=0; i<n; i++) if(le[i] == -1) res += dfs(i);
    } return res;
  }
  vector<pair<int,int>> maximum_matching_edges(){
    int matching = maximum_matching();
    vector<pair<int,int>> edges; edges.reserve(matching);
    for(int i=0; i<n; i++) if(le[i] != -1)
    edges.emplace_back(i, le[i]);
    return edges;
  }
  void dfs_track(int v){
    if(track[v]) return; track[v] = 1;
    for(auto i : g[v]) track[n+i] = 1, dfs_track(ri[i]);
  }
  tuple<vector<int>, vector<int>, int>
  minimum_vertex_cover(){
    int matching = maximum_matching(); vector<int> lv, rv;
    fill(track.begin(), track.end(), 0);
    for(int i=0; i<n; i++) if(le[i] == -1) dfs_track(i);
    for(int i=0; i<n; i++) if(!track[i]) lv.push_back(i);
    for(int i=0; i<m; i++) if(track[n+i]) rv.push_back(i);
    return {lv, rv, lv.size() + rv.size()}; //
    s(lv)+s(rv)=mat
  }
  tuple<vector<int>, vector<int>, int>
  maximum_independent_set(){
    auto [a,b,matching] = minimum_vertex_cover();
    vector<int> lv, rv; lv.reserve(n-a.size());
    rv.reserve(m-b.size());
    for(int i=0, j=0; i<n; i++){
      while(j < a.size() && a[j] < i) j++;
      if(j == a.size() || a[j] != i) lv.push_back(i);
    }
    for(int i=0, j=0; i<m; i++){
      while(j < b.size() && b[j] < i) j++;
      if(j == b.size() || b[j] != i) rv.push_back(i);
    } // s(lv)+s(rv)=n+m-mat
    return {lv, rv, lv.size() + rv.size()};
  }
  vector<vector<int>> minimum_path_cover(){ // n == m
    int matching = maximum_matching();
    vector<vector<int>> res; res.reserve(n - matching);
    fill(track.begin(), track.end(), 0);
    auto get_path = [&](int v) -> vector<int> {
      vector<int> path{v}; // ri[v] == -1
      while(le[v] != -1) path.push_back(v=le[v]);
      return path;
    };
    for(int i=0; i<n; i++) if(!track[n+i] && ri[i] == -1)
      res.push_back(get_path(i));
    return res; // sz(res) = n-mat
  }
  vector<int> maximum_anti_chain(){ // n = m
    auto [a,b,matching] = minimum_vertex_cover();
    vector<int> res; res.reserve(n - a.size() - b.size());
    for(int i=0, j=0, k=0; i<n; i++){
      while(j < a.size() && a[j] < i) j++;
      while(k < b.size() && b[k] < i) k++;
      if((j == a.size() || a[j] != i) && (k == b.size() ||
      b[k] != i)) res.push_back(i);
    } return res; // sz(res) = n-mat
  }
};
```

2.18 $O(V^2\sqrt{E})$ Push Relabel

```
template<typename flow_t> struct Edge {
  int u, v, r; flow_t c, f; Edge() = default;
  Edge(int u, int v, flow_t c, int r) : u(u), v(v), r(r),
  c(c), f(0) {}
};
template<typename flow_t, size_t _Sz> struct PushRelabel {
  using edge_t = Edge<flow_t>;
  int n, b, dist[_Sz], count[_Sz+1];
  flow_t excess[_Sz]; bool active[_Sz];
  vector<edge_t> g[_Sz]; vector<int> bucket[_Sz];
  void clear(){ for(int i=0; i<_Sz; i++) g[i].clear(); }
  void addEdge(int s, int e, flow_t x){
    g[s].emplace_back(s, e, x, (int)g[e].size());
    if(s == e) g[s].back().r++;
    g[e].emplace_back(e, s, 0, (int)g[s].size()-1);
  }
  void enqueue(int v){
    if(!active[v] && excess[v] > 0 && dist[v] < n){
      active[v] = true; bucket[dist[v]].push_back(v); b =
      max(b, dist[v]); }
  }
  void push(edge_t &e){
    flow_t fl = min(excess[e.u], e.c - e.f);
    if(dist[e.u] == dist[e.v] + 1 && fl > flow_t(0)){
      e.f += fl; g[e.v][e.r].f -= fl; excess[e.u] -= fl;
      excess[e.v] += fl; enqueue(e.v); }
  }
  void gap(int k){
    for(int i=0; i<n; i++){
      if(dist[i] >= k) count[dist[i]]--, dist[i] =
      max(dist[i], n), count[dist[i]]++;
      enqueue(i); }
  }
  void relabel(int v){
    count[dist[v]]--; dist[v] = n;
    for(const auto &e : g[v]) if(e.c - e.f > 0) dist[v] =
    min(dist[v], dist[e.v] + 1);
    count[dist[v]]++; enqueue(v);
  }
  void discharge(int v){
    for(auto &e : g[v]) if(excess[v] > 0) push(e); else
    break;
    if(excess[v] > 0) if(count[dist[v]] == 1) gap(dist[v]);
    else relabel(v);
  }
  flow_t maximumFlow(int _n, int s, int t){
    // memset dist, excess, count, active 0
    n = _n; b = 0; for(auto &e : g[s]) excess[s] += e.c;
    count[s] = n; enqueue(s); active[t] = true;
    while(b >= 0){
      if(bucket[b].empty()) b--;
      else{
        int v = bucket[b].back(); bucket[b].pop_back();
        active[v] = false; discharge(v);
      } /*else*/ } /*while*/ return excess[t];
  }
};
```

2.19 $O(V^2E)$ Dinic

```
template<typename FlowType, size_t _Sz, FlowType
_Inf=1'000'000'007>
struct Dinic{
  struct Edge{ int v, dual; FlowType c; };
  int Level[_Sz], Work[_Sz];
  vector<Edge> G[_Sz];
  void clear(){ for(int i=0; i<_Sz; i++) G[i].clear(); }
  void AddEdge(int s, int e, FlowType x){
    G[s].push_back({e, (int)G[e].size(), x});
    G[e].push_back({s, (int)G[s].size()-1, 0});
  }
  bool BFS(int S, int T){
    memset(Level, 0, sizeof Level);
    queue<int> Q; Q.push(S); Level[S] = 1;
    while(Q.size()){
      int v = Q.front(); Q.pop();
      for(const auto &i : G[v]){
        if(!Level[i.v] && i.c) Q.push(i.v), Level[i.v] =
        Level[v] + 1;
      }
    }
    return Level[T];
  }
  FlowType DFS(int v, int T, FlowType tot){
    if(v == T) return tot;
    for(int &_i=Work[v]; _i<G[v].size(); _i++){
      Edge &i = G[v][_i];
      if(Level[i.v] != Level[v] + 1 || !i.c) continue;
      FlowType fl = DFS(i.v, T, min(tot, i.c));
      if(!fl) continue;
      i.c -= fl; G[i.v][i.dual].c += fl;
      return fl;
```

```
    }
    return 0;
  }
  FlowType MaxFlow(int S, int T){
    FlowType ret = 0, tmp;
    while(BFS(S, T)){
      memset(Work, 0, sizeof Work);
      while((tmp = DFS(S, T, _Inf))) ret += tmp;
    }
    return ret;
  }
  tuple<FlowType, vector<int>, vector<int>> MinCut(int S, int
  T){
    FlowType fl = MaxFlow(S, T);
    vector<int> a, b;
    const int Bias = 1e9;
    queue<int> Q; Q.push(S); Level[S] += Bias;
    while(Q.size()){
      int v = Q.front(); Q.pop();
      for(const auto &i : G[v]){
        if(Level[i.v] < Bias) Q.push(i.v), Level[i.v] +=
        Bias;
      }
    }
    for(int i=0; i<_Sz; i++){
      if(Level[i]) a.push_back(i);
      else b.push_back(i);
    }
    return make_tuple(fl, a, b);
  }
};
```

2.20 Manhattan MST

```
void solve(int n) {
  init();
  vector<int> v(n), ds;
  for (int i = 0; i < n; ++i) {
    v[i] = i, ds.pb(x[i] - y[i]);
  }
  sort(ds.begin(), ds.end());
  ds.resize(unique(ds.begin(), ds.end()) - ds.begin());
  sort(v.begin(), v.end(), [&](int i, int j) { return x[i] ==
  x[j] ? y[i] > y[j] : x[i] > x[j]; });
  int j = 0;
  for (int i = 0; i < n; ++i) {
    int p = lower_bound(ds.begin(), ds.end(), x[v[i]] -
    y[v[i]]) - ds.begin() + 1;
    pair<int, int> q = query(p);
    // query return prefix minimum
    if (~q.second) add_edge(v[i], q.second);
    add(p, make_pair(x[v[i]] + y[v[i]], v[i]));
  }
}
void make_graph() {
  solve(n);
  for (int i = 0; i < n; ++i) swap(x[i], y[i]);
  solve(n);
  for (int i = 0; i < n; ++i) x[i] = -x[i];
  solve(n);
  for (int i = 0; i < n; ++i) swap(x[i], y[i]);
  solve(n);
}
```

2.21 $O(V^3)$ Hungarian Method

```
// C[j][w] = cost(j-th job, w-th worker), j <= w, O(J^2W)
// ret[i] = minimum cost to assign 0..i jobs to distinct
workers
template<typename T>bool ckmin(T &a, const T &b){return b<a ?
a=b,1 : 0;}
template<typename T>vector<T>Hungarian(const
vector<vector<T>>&C){
  const int J = C.size(), W = C[0].size(); assert(J <= W);
  vector<int> job(W+1, -1); //job[i] - i(worker) matched
  vector<T> ys(J), yt(W + 1), answers;//W-th worker is dummy
  const T inf = numeric_limits<T>::max();
  for(int j_cur=0; j_cur<J; j_cur++){
    int w_cur = W; job[w_cur] = j_cur;
    vector<T> min_to(W+1,inf);vector<int> prv(W+1,
    -1),in(W+1);
    while(job[w_cur] != -1){
      in[w_cur]=1; T delta=inf; int j = job[w_cur], w_next;
      for(int w=0; w<W; w++){ if(in[w] != 0) continue;
        if(ckmin(min_to[w], C[j][w]-ys[j]-yt[w]))
        prv[w]=w_cur;
        if(ckmin(delta, min_to[w])) w_next = w;
      }
      for(int w=0; w<=W; w++){
        if(in[w] == 0) min_to[w] -= delta;
        else ys[job[w]] += delta, yt[w] -= delta;
      } /*end for w*/ w_cur = w_next; } /* end while */
    for(int w; w_cur!=-1;
    w_cur=w)job[w_cur]=job[w=prv[w_cur]];
    answers.push_back(-yt[W]);
  } return answers; }
```

2.22 $O(V^3)$ Global Min Cut

```
template<typename T, T INF>// 0-based, adj matrix
pair<T, vector<int>> GetMinCut(vector<vector<T>> g){
  int n=g.size(); vector<int> use(n), cut, mn_cut; T mn=INF;
  for(int phase=n-1; phase>=0; phase--){
    vector<int> w=g[0], add=use; int k=0, prv;
    for(int i=0; i<phase; i++){ prv = k; k = -1;
      for(int j=1; j<n; j++) if(!add[j] && (k==-1 || w[j] >
      w[k])) k=j;
      if(i + 1 < phase){
        for(int j=0; j<n; j++) w[j] += g[k][j];
        add[k] = 1; continue; }
      for(int j=0; j<n; j++) g[prv][j] += g[k][j];
      for(int j=0; j<n; j++) g[j][prv] = g[prv][j];
      use[k] = 1; cut.push_back(k);
      if(w[k] < mn) mn_cut = cut, mn = w[k];
    }
  } return {mn, mn_cut};
}
```

2.23 Kuhn Algorithm $O(nm)$

```
int n, k;
vector<vector<int>> g;
vector<int> mt;
vector<bool> used;

bool try_kuhn(int v) {
    if (used[v])
        return false;
    used[v] = true;
    for (int to : g[v]) {
        if (mt[to] == -1 || try_kuhn(mt[to])) {
            mt[to] = v;
            return true;
        }
    }
    return false;
}
int main() {
    // ... reading the graph ...
    mt.assign(k, -1);
    vector<bool> used1(n, false);
    for (int v = 0; v < n; ++v) {
        for (int to : g[v]) {
            if (mt[to] == -1) {
                mt[to] = v;
                used1[v] = true;
                break;
            }
        }
    }
    for (int v = 0; v < n; ++v) {
        if (used1[v])
            continue;
        used.assign(n, false);
        try_kuhn(v);
    }

    for (int i = 0; i < k; ++i)
        if (mt[i] != -1)
            printf("%d %d\n", mt[i] + 1, i + 1);
}
```

2.24 $O(E \log V)$ Directed MST

```
using D = int; struct edge { int u, v; D w; };
vector<int> DirectedMST(vector<edge> &e, int n, int root){
  using T = pair<D, int>;  // 0-based, return index of edges
  using PQ = pair<priority_queue <T,vector<T>,greater<T>>,
  D>;
  auto push = [](PQ &pq, T v){
  pq.first.emplace(v.first-pq.second, v.second); };
  auto top = [](const PQ &pq) -> T {
    auto r = pq.first.top(); return {r.first + pq.second,
    r.second}; };
  auto join = [&push, &top](PQ &a, PQ &b) {
    if(a.first.size() < b.first.size()) swap(a, b);
```

```
    while(!b.first.empty()) push(a, top(b)), b.first.pop();
    };
  vector<PQ> h(n * 2);
  for(int i=0; i<e.size(); i++) push(h[e[i].v], {e[i].w, i});
  vector<int> a(n*2), v(n*2, -1), pa(n*2, -1), r(n*2);
  iota(a.begin(), a.end(), 0);
  auto o = [&](int x) { int y; for(y=x; a[y]!=y; y=a[y]);;;
    for(int ox=x; x!=y; ox=x) x = a[x], a[ox] = y;
    return y; };
  v[root] = n + 1; int pc = n;
  for(int i=0; i<n; i++) if(v[i] == -1) {
    for(int p=i; v[p]==-1 || v[p]==i; p=o(e[r[p]].u)){
      if(v[p] == i){ int q = p; p = pc++;
        do{ h[q].second = -h[q].first.top().first;
          join(h[pa[q]=a[q]=p], h[q]);
        }while((q=o(e[r[q]].u)) != p);
      } v[p] = i;
      while(!h[p].first.empty() && o(e[top(h[p]).second].u)
      == p) h[p].first.pop();
      r[p] = top(h[p]).second;
    }
  }
  vector<int> ans;
  for(int i=pc-1; i>=0; i--) if(i != root && v[i] != n) {
    for(int f=e[r[i]].v; f!=-1 && v[f]!=n; f=pa[f]) v[f] =
    n;;;
    ans.push_back(r[i]);
  }
  return ans;
}
```

2.25 MCMF CP Algorithm $O(FT)$

```
struct Edge
{
    int from, to, capacity, cost;
};

vector<vector<int>> adj, cost, capacity;

const int INF = 1e9;

void shortest_paths(int n, int v0, vector<int>& d,
vector<int>& p) {
    d.assign(n, INF);
    d[v0] = 0;
    vector<bool> inq(n, false);
    queue<int> q;
    q.push(v0);
    p.assign(n, -1);

    while (!q.empty()) {
        int u = q.front();
        q.pop();
        inq[u] = false;
        for (int v : adj[u]) {
            if (capacity[u][v] > 0 && d[v] > d[u] +
            cost[u][v]) {
                d[v] = d[u] + cost[u][v];
                p[v] = u;
                if (!inq[v]) {
                    inq[v] = true;
                    q.push(v);
                }
            }
        }
    }
}

int min_cost_flow(int N, vector<Edge> edges, int K, int s,
int t) {
    adj.assign(N, vector<int>());
    cost.assign(N, vector<int>(N, 0));
    capacity.assign(N, vector<int>(N, 0));
    for (Edge e : edges) {
        adj[e.from].push_back(e.to);
        adj[e.to].push_back(e.from);
        cost[e.from][e.to] = e.cost;
        cost[e.to][e.from] = -e.cost;
        capacity[e.from][e.to] = e.capacity;
    }

    int flow = 0;
    int cost = 0;
    vector<int> d, p;
    while (flow < K) {
        shortest_paths(N, s, d, p);
        if (d[t] == INF)
            break;

        // find max flow on that path
        int f = K - flow;
        int cur = t;
        while (cur != s) {
            f = min(f, capacity[p[cur]][cur]);
            cur = p[cur];
        }

        // apply flow
        flow += f;
        cost += f * d[t];
        cur = t;
        while (cur != s) {
            capacity[p[cur]][cur] -= f;
            capacity[cur][p[cur]] += f;
            cur = p[cur];
        }
    }

    if (flow < K)
        return -1;
    else
        return cost;
}
```

2.26 Dinic 2 - CP Algorithm

```
struct FlowEdge {
    int v, u;
    long long cap, flow = 0;
    FlowEdge(int v, int u, long long cap) : v(v), u(u),
    cap(cap) {}
};

struct Dinic {
    const long long flow_inf = 1e18;
    vector<FlowEdge> edges;
    vector<vector<int>> adj;
    int n, m = 0;
    int s, t;
    vector<int> level, ptr;
    queue<int> q;

    Dinic(int n, int s, int t) : n(n), s(s), t(t) {
        adj.resize(n);
        level.resize(n);
        ptr.resize(n);
    }

    void add_edge(int v, int u, long long cap) {
        edges.emplace_back(v, u, cap);
        edges.emplace_back(u, v, 0);
        adj[v].push_back(m);
        adj[u].push_back(m + 1);
        m += 2;
    }

    bool bfs() {
        while (!q.empty()) {
            int v = q.front();
            q.pop();
            for (int id : adj[v]) {
                if (edges[id].cap == edges[id].flow)
                    continue;
                if (level[edges[id].u] != -1)
                    continue;
                level[edges[id].u] = level[v] + 1;
                q.push(edges[id].u);
            }
        }
        return level[t] != -1;
    }

    long long dfs(int v, long long pushed) {
        if (pushed == 0)
            return 0;
        if (v == t)
            return pushed;
        for (int& cid = ptr[v]; cid < (int)adj[v].size();
        cid++) {
            int id = adj[v][cid];
            int u = edges[id].u;
            if (level[v] + 1 != level[u])
                continue;
```

```
            long long tr = dfs(u, min(pushed, edges[id].cap -
            edges[id].flow));
            if (tr == 0)
                continue;
            edges[id].flow += tr;
            edges[id ^ 1].flow -= tr;
            return tr;
        }
        return 0;
    }

    long long flow() {
        long long f = 0;
        while (true) {
            fill(level.begin(), level.end(), -1);
            level[s] = 0;
            q.push(s);
            if (!bfs())
                break;
            fill(ptr.begin(), ptr.end(), 0);
            while (long long pushed = dfs(s, flow_inf)) {
                f += pushed;
            }
        }
        return f;
    }
};
```

2.27 $O(V^3)$ General Matching

```
int N, M, R, Match[555], Par[555], Chk[555], Prv[555],
Vis[555];
vector<int> G[555]; // n 500 20ms
int Find(int x){return x == Par[x] ? x : Par[x] =
Find(Par[x]);}
int LCA(int u, int v){ static int cnt = 0;
  for(cnt++; Vis[u]!=cnt; swap(u, v)) if(u) Vis[u] = cnt, u =
  Find(Prv[Match[u]]);
  return u; }
void Blossom(int u, int v, int rt, queue<int> &q){
  for(; Find(u)!=rt; u=Prv[v]){
    Prv[u] = v; Par[u] = Par[v=Match[u]] = rt;
    if(Chk[v] & 1) q.push(v), Chk[v] = 2;
} }
bool Augment(int u){ // iota Par 0, fill Chk 0
  queue<int> Q; Q.push(u); Chk[u] = 2;
  while(!Q.empty()){ u = Q.front(); Q.pop();
    for(auto v : G[u]){
      if(Chk[v] == 0){
        Prv[v]=u; Chk[v]=1; Q.push(Match[v]);
        Chk[Match[v]]=2;
        if(!Match[v]){ for(; u; v=u) u = Match[Prv[v]],
        Match[Match[v]=Prv[v]] = v;;; return true; }
      }
      else if(Chk[v] == 2){ int l = LCA(u, v); Blossom(u, v,
      l, Q), Blossom(v, u, l, Q); }
  } /* for v */ } /* while */
  return 0; }
void Run(){ for(int i=1; i<=N; i++) if(!Match[i]) R +=
Augment(i); }
```

2.28 $O(V^3)$ Weighted General Matching

```
namespace weighted_blossom_tree{ // n 400 w 1e8 700ms, n 500 w
1e6 300ms
  #define d(x) (lab[x.u]+lab[x.v]-e[x.u][x.v].w*2)
  const int N=403*2; using ll = long long; using T = int; //
  sum of weight, single weight
  const T inf=numeric_limits<T>::max()>>1;
  struct Q{ int u, v; T w; } e[N][N]; vector<int> p[N];
  int n, m=0, id, h, t, lk[N], sl[N], st[N], f[N], b[N][N],
  s[N], ed[N], q[N]; T lab[N];
  void upd(int u, int v){ if (!sl[v] || d(e[u][v]) <
  d(e[sl[v]][v])) sl[v] = u; }
  void ss(int v){
    sl[v]=0; for(int u=1; u<=n; u++) if(e[u][v].w > 0 &&
    st[u] != v && !s[st[u]]) upd(u, v);
  }
  void ins(int u){ if(u <= n) q[++t] = u; else for(int v :
  p[u]) ins(v); }
  void mdf(int u, int w){ st[u]=w; if(u > n) for(int v :
  p[u]) mdf(v, w); }
  int gr(int u,int v){
    if ((v=find(p[u].begin(), p[u].end(), v) - p[u].begin())
    & 1){
      reverse(p[u].begin()+1, p[u].end()); return
      (int)p[u].size() - v;
    }
    return v; }
  void stm(int u, int v){
    lk[u] = e[u][v].v;
    if(u <= n) return; Q w = e[u][v];
    int x = b[u][w.u], y = gr(u,x);
    for(int i=0; i<y; i++) stm(p[u][i], p[u][i^1]);
    stm(x,v);rotate(p[u].begin(), p[u].begin()+y,
    p[u].end()); }
  void aug(int u, int v){
    int w = st[lk[u]]; stm(u, v); if (!w) return;
    stm(w, st[f[w]]); aug(st[f[w]], w); }
  int lca(int u, int v){
    for(++id; u|v; swap(u, v)){
      if(!u) continue; if(ed[u] == id) return u;
      ed[u] = id; if(u = st[lk[u]]) u = st[f[u]]; // not ==
    }
    return 0; }
  void add(int u, int a, int v){
    int x = n+1; while(x <= m && st[x]) x++;
    if(x > m) m++;
    lab[x] = s[x] = st[x] = 0; lk[x] = lk[a];
    p[x].clear(); p[x].push_back(a);
    for(int i=u, j; i!=a; i=st[f[j]]) p[x].push_back(i),
    p[x].push_back(j=st[lk[i]]), ins(j);
    reverse(p[x].begin()+1, p[x].end());
    for(int i=v, j; i!=a; i=st[f[j]]) p[x].push_back(i),
    p[x].push_back(j=st[lk[i]]), ins(j);
    mdf(x,x); for(int i=1; i<=m; i++) e[x][i].w=e[i][x].w=0;
    memset(b[x]+1, 0, n*sizeof b[0][0]);
    for (int u : p[x]){
      for(v=1; v<=m; v++) if(!e[x][v].w || d(e[u][v]) <
      d(e[x][v])) e[x][v] = e[u][v],e[v][x] = e[v][u];
      for(v=1; v<=n; v++) if(b[u][v]) b[x][v] = u;
    }
    ss(x); }
  void ex(int u){  // s[u] == 1
    for(int x : p[u]) mdf(x, x);
    int a = b[u][e[u][f[u]].u],r = gr(u, a);
    for(int i=0; i<r; i+=2){
      int x = p[u][i], y = p[u][i+1];
      f[x] = e[y][x].u; s[x] = 1; s[y] = 0; sl[x] = 0;
      ss(y);;;
      ins(y); }
    s[a] = 1; f[a] = f[u];
    for(int i=r+1;i<p[u].size();i++)s[p[u][i]]=-1,
    ss(p[u][i]);
    st[u] = 0; }
  bool on(const Q &e){
    int u=st[e.u], v=st[e.v], a;
    if(s[v] == -1) f[v] = e.u, s[v] = 1, a = st[lk[v]], sl[v]
    = sl[a] = s[a] = 0, ins(a);
    else if(!s[v]){
      a = lca(u, v); if(!a) return aug(u,v), aug(v,u), true;
      else add(u,a,v);
    }
    return false; }
  bool bfs(){
    memset(s+1, -1, m*sizeof s[0]); memset(sl+1, 0, m*sizeof
    sl[0]);
    h = 1; t = 0; for(int i=1; i<=m; i++) if(st[i] == i &&
    !lk[i]) f[i] = s[i] = 0, ins(i);
    if(h > t) return 0;
    while (true){
      while (h <= t){
        int u = q[h++];
        if (s[st[u]] != 1) for (int v=1; v<=n; v++) if
        (e[u][v].w > 0 && st[u] != st[v])
          if(d(e[u][v])) upd(u, st[v]); else if(on(e[u][v]))
          return true;
      }
      T x = inf;
      for(int i=n+1; i<=m; i++) if(st[i] == i && s[i] == 1) x
      = min(x, lab[i]>>1);
      for(int i=1; i<=m; i++) if(st[i] == i && sl[i] && s[i]
      != 1) x = min(x, d(e[sl[i]][i])>>s[i]+1);
      for(int i=1; i<=n; i++) if(~s[st[i]]) if((lab[i] +=
      (s[st[i]]*2-1)*x) <= 0) return false;
      for(int i=n+1 ;i<=m; i++) if(st[i] == i && ~s[st[i]])
      lab[i] += (2-s[st[i]]*4)*x;
      h = 1; t = 0;
      for(int i=1; i<=m; i++) if(st[i] == i && sl[i] &&
      st[sl[i]] != i && !d(e[sl[i]][i]) && on(e[sl[i]][i]))
      return true;
      for(int i=n+1; i<=m; i++) if(st[i] == i && s[i] == 1 &&
      !lab[i]) ex(i);
```

```
    }
    return 0; }
  template<typename TT> pair<int,ll> run(int N, const
  vector<tuple<int,int,TT>> &edges){ // 1-based
    memset(ed+1, 0, m*sizeof ed[0]); memset(lk+1, 0, m*sizeof
    lk[0]);
    n = m = N; id = 0; iota(st+1, st+n+1, 1); T wm = 0; ll r
    = 0;
    for(int i=1; i<=n; i++) for(int j=1; j<=n; j++) e[i][j] =
    {i,j,0};
    for(auto [u,v,w] : edges) wm = max(wm,
    e[v][u].w=e[u][v].w=max(e[u][v].w,(T)w));
    for(int i=1; i<=n; i++) p[i].clear();
    for(int i=1; i<=n; i++) for (int j=1; j<=n; j++) b[i][j]
    = i*(i==j);
    fill_n(lab+1, n, wm); int match = 0; while(bfs())
    match++;
    for(int i=1; i<=n; i++) if(lk[i]) r += e[i][lk[i]].w;
    return {match, r/2};
  }
  #undef d
} using weighted_blossom_tree::run,
weighted_blossom_tree::lk;
```

3 Math

3.1 Binary GCD, Extend GCD, CRT, Combination

```
ll binary_gcd(ll a, ll b){
  if(a == 0 || b == 0) return a + b;
  int az = __builtin_ctzll(a), bz = __builtin_ctzll(b);
  int shift = min(az, bz); b >>= bz;
  while(a){ a >>= az; ll diff = b-a;
    az = __builtin_ctz(diff); b = min(a, b); a = abs(diff);
  } return b << shift;
} // return [g,x,y] s.t. ax+by=gcd(a,b)=g
tuple<ll,ll,ll> ext_gcd(ll a, ll b){
  if(b == 0) return {a, 1, 0}; auto [g,x,y] = ext_gcd(b, a %
  b);
  return {g, y, x - a/b * y}; }
ll inv(ll a, ll m){ //return x when ax mod m = 1, fail -> -1
  auto [g,x,y] = ext_gcd(a, m); return g == 1 ? mod(x, m) :
  -1;}
void DivList(ll n){ // {n/1, n/2, ... , n/n}, size <= 2 sqrt
n
  for(ll i=1, j=1; i<=n; i=j+1) Report(i, j=n/(n/i), n/i); }
void Div2List(ll n){// n/(i^2), n^{3/4}
  for(ll i=1, j=1; i*i<=n; i=j+1){
    j = (ll)floorl(sqrtl(n/(n/(i*i)))); Report(i, j,
    n/(i*i));
} }//square free: sum_{i=1..sqrt n} mu(i)floor(n/(i^2))
pair<ll,ll> crt(ll a1, ll m1, ll a2, ll m2){
  ll g = gcd(m1, m2), m = m1 / g * m2;
  if((a2 - a1) % g) return {-1, -1};
  ll md = m2/g, s = mod((a2-a1)/g, m2/g);
  ll t = mod(get<1>(ext_gcd(m1/g%md, m2/g)), md);
  return { a1 + s * t % md * m1, m }; }
pair<ll,ll> crt(const vector<ll> &a, const vector<ll> &m){
  ll ra = a[0], rm = m[0];
  for(int i=1; i<m.size(); i++){
    auto [aa,mm] = crt(ra, rm, a[i], m[i]);
    if(mm == -1) return {-1, -1}; else tie(ra,rm) =
    tie(aa,mm);
  } return {ra, rm}; }
struct Lucas{ // init : O(P), query : O(log P)
  const size_t P; vector<ll> fac, inv;
  ll Pow(ll a, ll b){ /* return a^b mod P */ }
  Lucas(size_t P):P(P),fac(P),inv(P){ /* init fac, facinv */
  }
  ll small(ll n, ll r) const { /* n! / r! / (n-r)! */ }
  ll calc(ll n, ll r) const { if(n<r || n<0 || r<0) return 0;
    if(!n || !r || n == r) return 1;
    else return small(n%P, r%P) * calc(n/P, r/P) % P; }
};
template<ll p, ll e> struct CombinationPrimePower{
  vector<ll> val; ll m; // init : O(p^e), query : O(log p)
  CombinationPrimePower(){
    m=1; for(int i=0; i<e; i++) m *= p; val.resize(m);
    val[0]=1;
    for(int i=1; i<m; i++)val[i] = val[i-1] * (i%p ? i : 1) %
    m;
  }
  pair<ll,ll> factorial(int n){ if(n < p) return {0, val[n]};
    int k = n / p; auto v = factorial(k);
    int cnt = v.first + k, kp = n / m, rp = n % m;
    ll ret=v.second * Pow(val[m-1], kp%2, m) % m * val[rp] %
    m;
    return {cnt, ret}; }
  ll calc(int n, int r){ if(n < 0 || r < 0 || n < r) return
  0;
    auto v1=factorial(n), v2=factorial(r), v3=factorial(n-r);
    ll cnt = v1.first - v2.first - v3.first;
    ll ret = v1.second * inv(v2.second, m) % m *
    inv(v3.second, m) % m;
    if(cnt >= e) return 0;
    for(int i=1; i<=cnt; i++) ret = ret * p % m;
    return ret; }
};
```

3.2 Diophantine

```
// solutions to ax + by = c where x in [xlow, xhigh] and y in
[ylow, yhigh]
// cnt, leftsol, rightsol, gcd of a and b
template<class T> array<T, 6> solve_linear_diophantine(T a, T
b, T c, T xlow, T xhigh, T ylow, T yhigh){
    T g, x, y = euclid(a >= 0 ? a : -a, b >= 0 ? b : -b, x,
    y); array<T, 6> no_sol{0, 0, 0, 0, 0, g};
    if(c % g) return no_sol; x *= c / g, y *= c / g;
    if(a < 0) x = -x; if(b < 0) y = -y;
    a /= g, b /= g, c /= g;
    auto shift = [&](T &x, T &y, T a, T b, T cnt){ x += cnt *
    b, y -= cnt * a; };
    int sign_a = a > 0 ? 1 : -1, sign_b = b > 0 ? 1 : -1;
    shift(x, y, a, b, (xlow - x) / b);
    if(x < xlow) shift(x, y, a, b, sign_b);
    if(x > xhigh) return no_sol;
    T lx1 = x; shift(x, y, a, b, (xhigh - x) / b);
    if(x > xhigh) shift(x, y, a, b, -sign_b);
    T rx1 = x; shift(x, y, a, b, -(ylow - y) / a);
    if(y < ylow) shift(x, y, a, b, -sign_a);
    if(y > yhigh) return no_sol;
    T lx2 = x; shift(x, y, a, b, -(yhigh - y) / a);
    if(y > yhigh) shift(x, y, a, b, sign_a);
    T rx2 = x; if(lx2 > rx2) swap(lx2, rx2);
    T lx = max(lx1, lx2), rx = min(rx1, rx2);
    if(lx > rx) return no_sol;
    return {(rx - lx) / (b >= 0 ? b : -b) + 1, lx, (c - lx *
    a) / b, rx, (c - rx * a) / b, g};
}
```

3.3 FloorSum

```
// sum of floor((A*i+B)/M) over 0 <= i < N in O(log(N+M+A+B))
// Also, sum of i * floor((A*i+B)/M) and floor((A*i+B)/M)^2
template<class T, class U> // T must be able to hold arg^2
array<U, 3> weighted_floor_sum(T n, T m, T a, T b){
  array<U, 3> res{}; auto[qa,ra]=div(a,m);
  auto[qb,rb]=div(b,m);
  if(T n2 = (ra * n + rb) / m){
    auto prv=weighted_floor_sum<T,U>(n2, ra, m, m-rb-1);
    res[0] += U(n-1)*n2 - prv[0];
    res[1] += (U(n-1)*n*n2 - prv[0] - prv[2]) / 2;
    res[2] += U(n-1)*(n2-1)*n2 - 2*prv[1] + res[0];
  }
  res[2] += U(n-1)*n*(2*n-1)/6 * qa*qa + U(n)*qb*qb;
  res[2] += U(n-1)*n * qa*qb + 2*res[0]*qb + 2*res[1]*qa;
  res[0] += U(n-1)*n/2 * qa + U(n)*qb;
  res[1] += U(n-1)*n*(2*n-1)/6 * qa + U(n-1)*n/2 * qb;
  return res;
}
ll modsum(ull to, ll c, ll k, ll m){
  c = (c % m + m) % m; k = (k % m + m) % m;
  return to*c + k*sumsq(to) - m*divsum(to, c, k, m);
} // sum (ki+c)%m 0<=i<to, O(log m) large constant
```

3.4 XOR Basis (XOR Maximization)

```
vector<ll> basis; // ascending
for(int i=0; i<n; i++){ ll x; cin >> x;
for(int j=(int)basis.size()-1; j>=0; j--)
x=min(x,basis[j]^x);
if(x)basis.insert(lower_bound(basis.begin(),basis.end(),
x),x);
}//xor maximization, reverse -> for(auto
i:basis)r=max(r,r^i);
// minimization, return basis.back(), WARNING: x=0 => return
0
// choose 2k element => solve(a1^a2, a1^a3, a1^a4, ...)
```

3.5 $O(N^3 \log 1/\epsilon)$ Polynomial Equation

```
vector<double> poly_root(vector<double> p, double xmin,
double xmax){
  if(p.size() == 2){ return {-p[0] / p[1]}; }
  vector<double> ret, der(p.size()-1);
```

```
  for(int i=0; i<der.size(); i++) der[i] = p[i+1] * (i + 1);
  auto dr = poly_root(der, xmin, xmax);
  dr.push_back(xmin-1); dr.push_back(xmax+1);
  sort(dr.begin(), dr.end());
  for(int i=0; i+1<dr.size(); i++){
    double l = dr[i], h = dr[i+1]; bool sign = calc(p, l) >
    0;
    if (sign ^ (calc(p, h) > 0)){
      for(int it=0; it<60; it++){ // while(h-l > 1e-8)
        double m = (l + h) / 2, f = calc(p, m);
        if ((f <= 0) ^ sign) l = m; else h = m;
      }
      ret.push_back((l + h) / 2);
    }
  }
  return ret;
}
```

3.6 Gauss Jordan Elimination

```
template<typename T> // return {rref, rank, det, inv}
tuple<vector<vector<T>>, int, T, vector<vector<T>>>
Gauss(vector<vector<T>> a, bool square=true){ // n500 -400ms
  int n = a.size(), m = a[0].size(), rank = 0;//bitset
  4096-700
  vector<vector<T>> out(n, vector<T>(m, 0)); T det = T(1);
  for(int i=0; i<n; i++) if(square) out[i][i] = T(1);
  for(int i=0; i<m; i++){
    if(rank == n) break;
    if(IsZero(a[rank][i])){
      T mx = T(0); int idx = -1; // fucking precision error
      for(int j=rank+1; j<n; j++) if(mx < abs(a[j][i])) mx =
      abs(a[j][i]), idx = j;
      if(idx == -1 || IsZero(a[idx][i])){ det = 0; continue;
      }
      for(int k=0; k<m; k++){
        a[rank][k] = Add(a[rank][k], a[idx][k]);
        if(square)out[rank][k]=Add(out[rank][k],out[idx][k]);
      }
    }
    det = Mul(det, a[rank][i]);
    T coeff = Div(T(1), a[rank][i]);
    for(int j=0; j<m; j++) a[rank][j] = Mul(a[rank][j],
    coeff);
    for(int j=0; j<m; j++) if(square) out[rank][j] =
    Mul(out[rank][j], coeff);
    for(int j=0; j<n; j++){
      if(rank == j) continue;
      T t = a[j][i]; // Warning: [j][k], [rank][k]
      for(int k=0; k<m; k++) a[j][k] = Sub(a[j][k],
      Mul(a[rank][k], t));
      for(int k=0; k<m; k++) if(square) out[j][k] =
      Sub(out[j][k], Mul(out[rank][k], t));
    }
    rank++; // linear system: warning len(A) != len(A[0])
  } return {a, rank, det, out}; // linear system: get
  RREF(A|b)
} // 0 0 ... 0 b[i]: inconsistent, rank < len(A[0]): multiple
// get det(A) mod M, M can be composite number
// remove mod M -> get pure det(A) in integer
ll Det(vector<vector<ll>> a){//destroy matrix, n500 -400ms
  int n = a.size(); ll ans = 1;
  for(int i=0; i<n; i++){
    for(int j=i+1; j<n; j++){
      while(a[j][i] != 0){ // gcd step
        ll t = a[i][i] / a[j][i];
        if(t)for(int k=i;k<n;k++)
        a[i][k]=(a[i][k]-a[j][k]*t)%M;
        swap(a[i], a[j]); ans *= -1;
      }
    }
    ans = ans * a[i][i] % M; if(!ans) return 0;
  } return (ans + M) % M;
}
```

3.7 Berlekamp + Kitamasa

```
const int mod = 1e9+7; ll pw(ll a, ll b){/*a^b mod M*/}
vector<int> berlekamp_massey(vector<int> x){
  int n = x.size(),L=0,m=0; ll b=1; if(!n) return {};
  vector<int> C(n), B(n), T; C[0]=B[0]=1;
  for(int i=0; ++m && i<n; i++){ ll d = x[i] % mod;
    for(int j=1; j<=L; j++) d = (d + 1LL * C[j] * x[i-j]) %
    mod;
    if(!d) continue; T=C; ll c = d * pw(b, mod-2) % mod;
    for(int j=m; j<n; j++) C[j] = (C[j] - c * B[j-m]) % mod;
    if(2 * L <= i) L = i-L+1, B = T, b = d, m = 0;
  }
  C.resize(L+1); C.erase(C.begin());
  for(auto &i : C) i = (mod - i) % mod; return C;
} // O(NK + N log mod)
int get_nth(vector<int> rec, vector<int> dp, ll n){
  int m = rec.size(); vector<int> s(m), t(m); ll ret=0;
  s[0] = 1; if(m != 1) t[1] = 1; else t[0] = rec[0];
  auto mul = [&rec](vector<int> v, vector<int> w){
    int m = v.size(); vector<int> t(2*m);
    for(int j=0; j<m; j++) for(int k=0; k<m; k++){
      t[j+k] = (t[j+k] + 1LL * v[j] * w[k]) % mod;
    }
    for(int j=2*m-1; j>=m; j--) for(int k=1; k<=m; k++){
      t[j-k] = (t[j-k] + 1LL * t[j] * rec[k-1]) % mod;
    }
    t.resize(m); return t;
  };
  for(; n; n>>=1, t=mul(t,t)) if(n & 1) s=mul(s,t);
  for(int i=0; i<m; i++) ret += 1LL * s[i] * dp[i] % mod;
  return ret % mod;
} // O(N2 log X)
int guess_nth_term(vector<int> x, ll n){
  if(n < x.size()) return x[n];
  vector<int> v = berlekamp_massey(x);
  return v.empty() ? 0 : get_nth(v, x, n);
}
struct elem{int x, y, v;}; // A_(x, y) <- v, 0-based. no
duplicate please..
vector<int> get_min_poly(int n, vector<elem> M){
  // smallest poly P such that A^i = sum_{j < i} {A^j \times
  P_j}
  vector<int> rnd1, rnd2, gobs; mt19937 rng(0x14004);
  auto gen = [&rng](int lb, int ub){ return
  uniform_int_distribution<int>(lb, ub)(rng); };
  for(int i=0; i<n; i++) rnd1.push_back(gen(1, mod-1)),
  rnd2.push_back(gen(1, mod-1));
  for(int i=0; i<2*n+2; i++){ int tmp = 0;
    for(int j=0; j<n; j++) tmp = (tmp + 1LL * rnd2[j] *
    rnd1[j]) % mod;
    gobs.push_back(tmp); vector<int> nxt(n);
    for(auto &j : M) nxt[j.x] = (nxt[j.x] + 1LL * j.v *
    rnd1[j.y]) % mod;
    rnd1 = nxt;
  } auto v = berlekamp_massey(gobs);
  return vector<int>(v.rbegin(), v.rend());
}
ll det(int n, vector<elem> M){
  vector<int> rnd; mt19937 rng(0x14004);
  auto gen = [&rng](int lb, int ub){ return
  uniform_int_distribution<int>(lb, ub)(rng); };
  for(int i=0; i<n; i++) rnd.push_back(gen(1, mod-1));
  for(auto &i : M) i.v = 1LL * i.v * rnd[i.y] % mod;
  auto sol = get_min_poly(n, M)[0]; if(n % 2 == 0) sol = mod
  - sol;
  for(auto &i : rnd) sol = 1LL * sol * pw(i, mod-2) % mod;
  return sol;
}
```

3.8 Linear Sieve

```
// sp : smallest prime factor
// tau : number of divisors, sigma : sum of divisors
// phi : Euler's Totient Function
// mu : Mobius Function, 0 if n is not square-free(has a
squared prime factor), (-1)^k
// k is number of distinct prime factor

// e[i] : 소인수분해에서 i의 지수
vector<int> prime;
int sp[sz], e[sz], phi[sz], mu[sz], tau[sz], sigma[sz];
phi[1] = mu[1] = tau[1] = sigma[1] = 1;
for(int i=2; i<=n; i++){
  if(!sp[i]){
    prime.push_back(i);
    e[i] = 1; phi[i] = i-1; mu[i] = -1; tau[i] = 2; sigma[i]
    = i+1;
  }
  for(auto j : prime){
    if(i*j >= sz) break;
    sp[i*j] = j;
    if(i % j == 0){
      e[i*j] = e[i]+1; phi[i*j] = phi[i]*j; mu[i*j] = 0;
      tau[i*j] = tau[i]/e[i*j]*(e[i*j]+1);
      sigma[i*j] = sigma[i]*(j-1)/(pw(j, e[i*j])-1)*(pw(j,
      e[i*j]+1)-1)/(j-1);//overflow
      break;
    }
    e[i*j] = 1; phi[i*j] = phi[i] * phi[j]; mu[i*j] = mu[i] *
    mu[j];
```

```
    tau[i*j] = tau[i] * tau[j]; sigma[i*j] = sigma[i] *
    sigma[j];
  }
}
```

3.9 Xudyh Sieve

```
/* e(x) = [x==1], 1(x) = 1, id_k(x) = x^k
mu: mobius function, id(x) = x
phi: euler totient function
sigma_k: sum of k-th power of divisors
sigma = sigma_1, d = tau = sigma_0
sigma_k = id_k * 1  | sigma = id * 1
id_k = sigma_k * mu | id = sigma * mu
e = 1 * mu | d = 1 * 1 | 1 = d * mu
phi * 1 = id | phi = id * mu | sigma = phi * d
g = f * 1 iff f = g * mu */
template<class T, class F1, class F2, class F3>
struct xudyh_sieve{
  T th; // threshold, 2(single query) ~ 5 * MAXN^2/3
  F1 pf; F2 pg; F3 pfg;
  // prefix sum of f(up to th), g(easy to calc), f*g(easy to
  calc)
  unordered_map<T, T> mp; // f * g means dirichlet conv.
  xudyh_sieve(T th,F1 pf,F2 pg,F3
  pfg):th(th),pf(pf),pg(pg),pfg(pfg){}
  // Calculate the preix sum of a multiplicative f up to n
  T query(T n){ // O(n^2/3)
    if(n <= th) return pf(n); if(mp.count(n)) return mp[n];
    T res = pfg(n);
    for(T low = 2, high = 2; low <= n; low = high + 1){
      high = n / (n / low);
      res -= (pg(high) - pg(low - 1)) * query(n / low); //
      MOD
    }
    return mp[n] = res / pg(1);//Pow(pg(1),MOD-2)?
  }
};
```

3.10 Miller Rabin + Pollard Rho

```
// 32bit : 2, 7, 61 / ull MulMod, PowMod (cast __uint128_t)
// 64bit : 2, 325, 9375, 28178, 450775, 9780504, 1795265022
bool MillerRabin(ull n, ull a){
  if(a % n == 0) return true; int cnt = __builtin_ctzll(n -
  1);
  ull p=PowMod(a, n>>cnt, n); if(p==1 || p+1==n) return true;
  while(cnt--) if((p=MulMod(p,p,n)) == n - 1) return true;
  return false;
}
bool IsPrime(ll n){
  if(n <= 11) return hard_coding;
  if(n % 2 == 0 || ... 3 5 7 11) return false;
  for(int p : {comments}) if(!MillerRabin(n, p)) return
  false;
  return true;
}
ull Rho(ull n){
  ull x = 0, y = 0, t = 30, prd = 2, i = 1, q;
  auto f = [&](ull x) { return MulMod(x, x, n) + i; };
  while(t++ % 40 || __gcd(prd, n) == 1){
    if(x == y) x = ++i, y = f(x);
    if((q = MulMod(prd, max(x,y) - min(x,y), n))) prd = q;
    x = f(x), y = f(f(y));
  } return __gcd(prd, n);
}
vector<ull> Factorize(ull n){ // sort?
  if(n == 1) return {}; if(IsPrime(n)) return {n};
  auto x = Rho(n); auto l=Factorize(x), r=Factorize(n/x);
  l.insert(l.end(), r.begin(), r.end()); return l; }
```

3.11 Primitive Root, Discrete Log/Sqrt

```
ll PrimitiveRoot(ll p){ // order p-1
  vector<pair<ll,ll>> v = Factorize(p-1);
  for(ll r=1; ; r++){
    bool flag = true; // Warning: 64bit Pow
    for(auto [d,e] : v) if(PowMod(r, (p-1)/d, p) == 1){ flag
    = false; break; }
    if(flag) return r;
  }
}
// Given A, B, P, solve A^x === B mod P, return smallest
value
ll DiscreteLog(ll A, ll B, ll P){ // O(sqrt P) with hash set
  __gnu_pbds::gp_hash_table<ll,__gnu_pbds::null_type> st;
  ll t = ceil(sqrt(P)), k = 1; // use binary search?
  for(int i=0; i<t; i++) st.insert(k), k = k * A % P;
  ll inv = Pow(k, P-2, P);
  for(int i=0, s=1; i<t; i++, s=s*inv%P){
    ll x = B * s % P;
    if(st.find(x) == st.end()) continue;
    for(int j=0, f=1; j<t; j++, f=f*A%P){
      if(f == x) return i * t + j;
    }
  }
  return -1;
}
// Given A, P, solve X^2 === A mod P, return arbitrary
ll DiscreteSqrt(ll A, ll P){//O(log^2P), O(logP) in random
data
  if(A == 0) return 0;
  if(Pow(A, (P-1)/2, P) != 1) return -1;
  if(P % 4 == 3) return Pow(A, (P+1)/4, P);
  ll s = P - 1, n = 2, r = 0, m;
  while(~s & 1) r++, s >>= 1;
  while(Pow(n, (P-1)/2, P) != P-1) n++;
  ll x = Pow(A, (s+1)/2, P), b = Pow(A, s, P), g = Pow(n, s,
  P);
  for(;; r=m){
    ll t = b; for(m=0; m<r && t!=1; m++) t = t * t % P;
    if(!m) return x;
    ll gs = Pow(g, 1LL << (r-m-1), P);
    g = gs * gs % P; x = x * gs % P; b = b * g % P;
  }
}
```

3.12 FFT All

```
template<int M>
struct MINT{
  int v;
  MINT() : v(0) {}
  MINT(ll val){
    v = (-M <= val && val < M) ? val : val % M;
    if(v < 0) v += M;
  }
  // @TODO : pw, operator >> << == != + - * /
  friend MINT pw(MINT a, ll b){
    MINT ret= 1;
    while(b){
      if(b & 1) ret *= a;
      b >>= 1; a *= a;
    }
    return ret;
  }
  friend MINT inv(const MINT a) { return pw(a, M-2); }
};
namespace fft{
  using real_t = double; using cpx = complex<real_t>;
  void FFT(vector<cpx> &a, bool inv_fft = false){
    int N = a.size();
    vector<cpx> root(N/2);
    for(int i=1, j=0; i<N; i++){
      int bit = (N >> 1);
      while(j >= bit) j -= bit, bit >>= 1;
      j += bit;
      if(i < j) swap(a[i], a[j]);
    }
    real_t ang = 2 * acos(-1) / N * (inv_fft ? -1 : 1);
    for(int i=0; i<N/2; i++) root[i] = cpx(cos(ang * i),
    sin(ang * i));
    /*
    XOR Convolution : set roots[*] = 1.
    OR Convolution : set roots[*] = 1, and do following:
    if (!inv) a[j + k] = u + v, a[j + k + i/2] = u;
    else a[j + k] = v, a[j + k + i/2] = u - v;
    */
    for(int i=2; i<=N; i<<=1){
      int step = N / i;
      for(int j=0; j<N; j+=i) for(int k=0; k<i/2; k++){
        cpx u = a[j+k], v = a[j+k+i/2] * root[step * k];
        a[j+k] = u+v; a[j+k+i/2] = u-v;
      }
    }
    if(inv_fft) for(int i=0; i<N; i++) a[i] /= N; // skip for
    OR convolution.
  }
  vector<ll> multiply(const vector<ll> &_a, const vector<ll>
  &_b){
    vector<cpx> a(all(_a)), b(all(_b));
    int N = 2; while(N < a.size() + b.size()) N <<= 1;
    a.resize(N); b.resize(N);
    FFT(a); FFT(b);
    for(int i=0; i<N; i++) a[i] *= b[i];
    FFT(a, 1);
```

```
    vector<ll> ret(N);
    for(int i=0; i<N; i++) ret[i] = llround(a[i].real());
    return ret;
  }
  vector<ll> multiply_mod(const vector<ll> &a, const
  vector<ll> &b, const ull mod){
    int N = 2; while(N < a.size() + b.size()) N <<= 1;
    vector<cpx> v1(N), v2(N), r1(N), r2(N);
    for(int i=0; i<a.size(); i++) v1[i] = cpx(a[i] >> 15,
    a[i] & 32767);
    for(int i=0; i<b.size(); i++) v2[i] = cpx(b[i] >> 15,
    b[i] & 32767);
    FFT(v1); FFT(v2);
    for(int i=0; i<N; i++){
      int j = i ? N-i : i;
      cpx ans1 = (v1[i] + conj(v1[j])) * cpx(0.5, 0);
      cpx ans2 = (v1[i] - conj(v1[j])) * cpx(0, -0.5);
      cpx ans3 = (v2[i] + conj(v2[j])) * cpx(0.5, 0);
      cpx ans4 = (v2[i] - conj(v2[j])) * cpx(0, -0.5);
      r1[i] = (ans1 * ans3) + (ans1 * ans4) * cpx(0, 1);
      r2[i] = (ans2 * ans3) + (ans2 * ans4) * cpx(0, 1);
    }
    FFT(r1, true); FFT(r2, true);
    vector<ll> ret(N);
    for(int i=0; i<N; i++){
      ll av = llround(r1[i].real()) % mod;
      ll bv = ( llround(r1[i].imag()) + llround(r2[i].real())
      ) % mod;
      ll cv = llround(r2[i].imag()) % mod;
      ret[i] = (av << 30) + (bv << 15) + cv;
      ret[i] %= mod; ret[i] += mod; ret[i] %= mod;
    }
    return ret;
  }
  // 104,857,601   =  25 * 2^22 + 1, w = 3
  // 998,244,353   = 119 * 2^23 + 1, w = 3
  // 2,281,701,377 =  17 * 2^27 + 1, w = 3
  // 2,483,027,969 =  37 * 2^26 + 1, w = 3
  // 2,113,929,217 =  63 * 2^25 + 1, w = 5
  // 1,092,616,193 = 521 * 2^21 + 1, w = 3
  template<int W, int M>
  static void NTT(vector<MINT<M>> &f, bool inv_fft = false){
    using T = MINT<M>;
    int N = f.size();
    vector<T> root(N >> 1);
    for(int i=1, j=0; i<N; i++){
      int bit = N >> 1;
      while(j >= bit) j -= bit, bit >>= 1;
      j += bit;
      if(i < j) swap(f[i], f[j]);
    }
    T ang = pw(T(W), (M-1)/N); if(inv_fft) ang = inv(ang);
    root[0] = 1; for(int i=1; i<N>>1; i++) root[i] =
    root[i-1] * ang;
    for(int i=2; i<=N; i<<=1){
      int step = N / i;
      for(int j=0; j<N; j+=i) for(int k=0; k<i/2; k++){
        T u = f[j+k], v = f[j+k+(i>>1)] * root[k*step];
          f[j+k] = u + v; f[j+k+(i>>1)] = u - v;
        }
      }
      if(inv_fft){
        T rev = inv(T(N));
        for(int i=0; i<N; i++) f[i] *= rev;
      }
    }
    template<int W, int M>
    vector<MINT<M>> multiply_ntt(vector<MINT<M>> a,
    vector<MINT<M>> b){
      int N = 2; while(N < a.size() + b.size()) N <<= 1;
      a.resize(N); b.resize(N);
      NTT<W, M>(a); NTT<W, M>(b);
      for(int i=0; i<N; i++) a[i] *= b[i];
      NTT<W, M>(a, true);
      return a;
    }
}
template<int W, int M>
struct PolyMod{
  using T = MINT<M>;
  vector<T> a;
  PolyMod(){}
  PolyMod(T a0) : a(1, a0) { normalize(); }
  PolyMod(const vector<T> a) : a(a) { normalize(); }
  int size() const { return a.size(); }
  int deg() const { return a.size() - 1; }
  void normalize(){ while(a.size() && a.back() == T(0))
  a.pop_back(); }
  T operator [] (int idx) const { return a[idx]; }
  typename vector<T>::const_iterator begin() const { return
  a.begin(); }
  typename vector<T>::const_iterator end() const { return
  a.end(); }
  void push_back(const T val) { a.push_back(val); }
  void pop_back() { a.pop_back(); }
  T evaluate(T x) const {
    T ret = T(0);
    for(int i=deg(); i>=0; i--) ret = ret * x + a[i];
    return ret;
  }
  PolyMod reversed() const {
    vector<T> b = a;
    reverse(b.begin(), b.end());
    return b;
  }
  PolyMod trim(int n) const {
    return vector<T>(a.begin(), a.begin() + min(n, size()));
  }
  // @TODO : operator + - *(with scala) /(with scala)
  PolyMod inv(int n){
    PolyMod q(T(1) / a[0]);
    for(int i=1; i<n; i<<=1){
      PolyMod p = PolyMod(2) - q * trim(i * 2);
      q = (p * q).trim(i * 2);
    }
    return q.trim(n);
  }
  PolyMod operator *= (const PolyMod &b){
    *this = fft::multiply_ntt<W, M>(a, b.a);
    normalize(); return *this;
  }
  PolyMod operator /= (const PolyMod &b){
    if(deg() < b.deg()) return *this = PolyMod();
    int sz = deg() - b.deg() + 1;
    PolyMod ra = reversed().trim(sz), rb =
    b.reversed().trim(sz).inv(sz);
    *this = (ra * rb).trim(sz);
    for(int i=sz-size(); i; i--) push_back(T(0));
    reverse(all(a)); normalize();
    return *this;
  }
  PolyMod operator %= (const PolyMod &b){
    if(deg() < b.deg()) return *this;
    PolyMod tmp = *this; tmp /= b; tmp *= b;
    *this -= tmp; normalize();
    return *this;
  }
  PolyMod operator * (const PolyMod &b) const { return
  PolyMod(*this) *= b; }
  PolyMod operator / (const PolyMod &b) const { return
  PolyMod(*this) /= b; }
  PolyMod operator % (const PolyMod &b) const { return
  PolyMod(*this) %= b; }
};
using mint = MINT<998244353>;
using poly = PolyMod<3, 998244353>;
mint Kitamasa(poly c, poly a, ll n){
  poly d = vector<mint>{1};
  poly xn = vector<mint>{0, 1};
  poly f;
  for(int i=0; i<c.size(); i++) f.push_back(-c[i]);
  f.push_back(1);
  while(n){
    if(n & 1) d = d * xn % f;
    n >>= 1; xn = xn * xn % f;
  }
  mint ret = 0;
  for(int i=0; i<=a.deg(); i++) ret += a[i] * d[i];
  return ret;
}
```

4 String

4.1 KMP, Hash, Manacher, Z

```
vector<int> getFail(const container &pat){
  vector<int> fail(pat.size());
//match: pat[0..j] and pat[j-i..i] is equivalent
//ins/del: manipulate corresponding range to pattern starts
at 0
//        (insert/delete pat[i], manage pat[j-i..i])
  function<bool(int, int)> match = [&](int i, int j){ };
  function<void(int)> ins = [&](int i){ };
  function<void(int)> del = [&](int i){ };
```

```
  for(int i=1, j=0; i<pat.size(); i++){
    while(j && !match(i, j)){
      for(int s=i-j; s<i-fail[j-1]; s++) del(s);
      j = fail[j-1]; }
    if(match(i, j)) ins(i), fail[i] = ++j;
  } return fail;
}
vector<int> doKMP(const container &str, const container
&pat){
  vector<int> ret, fail = getFail(pat);
//match: pat[0..j] and str[j-i..i] is equivalent
//ins/del: manipulate corresponding range to pattern starts
at 0
//         (insert/delete str[i], manage str[j-i..i])
  function<bool(int, int)> match = [&](int i, int j){ };
  function<void(int)> ins = [&](int i){ };
  function<void(int)> del = [&](int i){ };
  for(int i=0, j=0, s; i<str.size(); i++){
    while(j && !match(i, j)){
      for(int s=i-j; s<i-fail[j-1]; s++) del(s);
      j = fail[j-1]; }
    if(match(i, j)){
      if(j+1 == pat.size()){
        ret.push_back(i-j); for(s=i-j;s<i-fail[j]+1;
        s++)del(s);
        j = fail[j];
      } else ++j;      ins(i);
    } } return ret;
}
// # a # b # a # a # b # a #
// 0 1 0 3 0 1 6 1 0 3 0 1 0
vector<int> Manacher(const string &inp){
  string s = "#"; for(auto c : inp) s += c, s += "#";
  int n = s.size(); vector<int> p(n);
  vector<pair<int,int>> range, maximal;
  auto make = [&](int l, int r) { return make_pair(l/2,
  (r-1)/2); };
  for(int i=0, k=-1, r=-1; i<n; i++){
    if(i <= r) p[i] = min(r-i, p[2*k-i]);
    while(i-p[i]-1 >= 0 && i+p[i]+1<n && s[i-p[i]-1] ==
    s[i+p[i]+1]){
      p[i]++; range.push_back(make(i-p[i], i+p[i]));
    } if(i+p[i] > r) r = i+p[i], k = i;
    if(p[i] != 0) maximal.push_back(make(i-p[i], i+p[i]));
  } // compress(range), range can contains O(1) dup.
  substr...
  return p; } // range: distinct palindrome(<= n)
//z[i]=match length of s[0,n-1] and s[i,n-1]
vector<int> Z(const string &s){
  int n = s.size(); vector<int> z(n); z[0] = n;
  for(int i=1, l=0, r=0; i<n; i++){
    if(i < r) z[i] = min(r-i-1, z[i-l]);
    while(i+z[i] < n && s[i+z[i]] == s[z[i]]) z[i]++;
    if(i+z[i] > r) r = i+z[i], l = i;
  } return z;
}
```

4.2 Aho-Corasick

```
struct AC { // remember to build_fail!!!
  int ch[N][C], to[N][C], fail[N], cnt[N], _id;
  // fail link tree: fail[i] -> i
  AC () { reset(); }
  int newnode() {
    fill_n(ch[_id], C, 0), fill_n(to[_id], C, 0);
    fail[_id] = cnt[_id] = 0; return _id++; }
  int insert(string s) {
    int now = 0;
    for (char c : s) {
      if (!ch[now][c - 'a'])
        ch[now][c - 'a'] = newnode();
      now = ch[now][c - 'a'];
    }
    cnt[now]++; return now;
  }
  void build_fail() {
    queue <int> q;
    for (int i = 0; i < C; ++i) if (ch[0][i])
      q.push(ch[0][i]), to[0][i] = ch[0][i];
    while (!q.empty()) {
      int v = q.front(); q.pop();
      for (int i = 0; i < C; ++i) {
        if (!ch[v][i]) to[v][i] = to[fail[v]][i];
        else {
          int u = ch[v][i], k = fail[v];
          while (k && !ch[k][i]) k = fail[k];
          if (ch[k][i]) k = ch[k][i];
          fail[u] = k, cnt[u] += cnt[k], to[v][i] = u;
          q.push(u);
        }
      }
    }
  }
  // int match(string &s) {
  //   int now = 0, ans = 0;
  //   for (char c : s) {
  //     now = to[now][c - 'a'];
  //     ans += cnt[now];
  //   }
  //   return ans;
  // }
  void reset() { _id = 0, newnode(); }
} ac;
```

4.3 Aho-Corasick 2

```
struct aho_corasick{
    struct node{
        int suffix_link = -1, exit_link = -1, nxt[128];
        vector<int> leaf;
        node() {fill(nxt, nxt+128, -1);}
    };
    vector<node> g = {node()};
    void insert_string(const string &s, int sidx){
        int p = 0;
        for (char c: s){
            if (g[p].nxt[c] == -1){
                g[p].nxt[c] = g.size();
                g.emplace_back();
            }
            p = g[p].nxt[c];
        }
        g[p].leaf.push_back(sidx);
    }
    void build_automaton(){
        for (deque<int> q = {0}; q.size(); q.pop_front()){
            int v = q.front(), suffix_link =
            g[v].suffix_link;
            if (v) g[v].exit_link =
            g[suffix_link].leaf.size() ? suffix_link :
            g[suffix_link].exit_link;
            for (int i=0; i<128; i++){
                int &nxt = g[v].nxt[i], nxt_sf = v ?
                g[suffix_link].nxt[i] : 0;
                if (nxt == -1) nxt = nxt_sf;
                else{
                    g[nxt].suffix_link = nxt_sf;
                    q.push_back(nxt);
                }
            }
        }
    }
    vector<int> get_sindex(int p){
        vector<int> a;
        for (int v = g[p].leaf.size() ? p : g[p].exit_link; v
        != -1; v = g[v].exit_link)
            for (int j: g[v].leaf)
                a.push_back(j);
        return a;
    }
};
```

4.4 $O(N \log N)$ SA + LCP

```
pair<vector<int>, vector<int>> SuffixArray(const string &s){
  int n = s.size(), m = max(n, 256);
  vector<int> sa(n), lcp(n), pos(n), tmp(n), cnt(m);
  auto counting_sort = [&](){
    fill(cnt.begin(), cnt.end(), 0);
    for(int i=0; i<n; i++) cnt[pos[i]]++;
    partial_sum(cnt.begin(), cnt.end(), cnt.begin());
    for(int i=n-1; i>=0; i--) sa[--cnt[pos[tmp[i]]]] =
    tmp[i];
  };
  for(int i=0; i<n; i++) sa[i] = i, pos[i] = s[i], tmp[i] =
  i;
  counting_sort();
  for(int k=1; ; k<<=1){ int p = 0;
    for(int i=n-k; i<n; i++) tmp[p++] = i;
    for(int i=0; i<n; i++) if(sa[i] >= k) tmp[p++] = sa[i] -
    k;
    counting_sort(); tmp[sa[0]] = 0;
    for(int i=1; i<n; i++){
      tmp[sa[i]] = tmp[sa[i-1]];
      if(sa[i-1]+k < n && sa[i]+k < n && pos[sa[i-1]] ==
      pos[sa[i]] && pos[sa[i-1]+k] == pos[sa[i]+k]) continue;
```

```
      tmp[sa[i]] += 1;
    }
    swap(pos, tmp); if(pos[sa.back()] + 1 == n) break;
  }
  for(int i=0, j=0; i<n; i++, j=max(j-1,0)){
    if(pos[i] == 0) continue;
    while(sa[pos[i]-1]+j < n && sa[pos[i]]+j < n &&
    s[sa[pos[i]-1]+j] == s[sa[pos[i]]+j]) j++;
    lcp[pos[i]] = j;
  } return {sa, lcp};
}
auto [SA,LCP] = SuffixArray(S); RMQ<int> rmq(LCP);
vector<int> Pos(N); for(int i=0; i<N; i++) Pos[SA[i]] = i;
auto get_lcp = [&](int a, int b){
    if(Pos[a] > Pos[b]) swap(a, b);
    return a == b ? (int)S.size() - a : rmq.query(Pos[a]+1,
    Pos[b]);
};
vector<pair<int,int>> can; // common substring {start, lcp}
vector<tuple<int,int,int>> valid; // valid substring [string,
end_l~end_r]
for(int i=1; i<N; i++){
  if(SA[i] < X && SA[i-1] > X) can.emplace_back(SA[i],
  LCP[i]);
  if(i+1 < N && SA[i] < X && SA[i+1] > X)
  can.emplace_back(SA[i], LCP[i+1]);
}
for(int i=0; i<can.size(); i++){
  int skip = i > 0 ? min({can[i-1].second, can[i].second,
  get_lcp(can[i-1].first, can[i].first)}) : 0;
  valid.emplace_back(can[i].first, can[i].first + skip,
  can[i].first + can[i].second - 1);
}
```

4.5 Suffix Automaton

```
template<typename T, size_t S, T init_val>
struct initialized_array : public array<T, S> {
  initialized_array(){ this->fill(init_val); }
};
template<class Char_Type, class Adjacency_Type>
struct suffix_automaton{
  // Begin States
  // len: length of the longest substring in the class
  // link: suffix link
  // firstpos: minimum value in the set endpos
  vector<int> len{0}, link{-1}, firstpos{-1},
  is_clone{false};
  vector<Adjacency_Type> next{{}};
  ll ans{0LL}; // 서로 다른 부분 문자열 개수
  // End States
  void set_link(int v, int lnk){
    if(link[v] != -1) ans -= len[v] - len[link[v]];
    link[v] = lnk;
    if(link[v] != -1) ans += len[v] - len[link[v]];
  }
  int new_state(int l, int sl, int fp, bool c, const
  Adjacency_Type &adj){
    int now = len.size(); len.push_back(l);
    link.push_back(-1);
    set_link(now, sl); firstpos.push_back(fp);
    is_clone.push_back(c); next.push_back(adj); return now;
  } int last = 0;
  void extend(const vector<Char_Type> &s){
    last = 0; for(auto c: s) extend(c); }
  void extend(Char_Type c){
    int cur = new_state(len[last] + 1, -1, len[last], false,
    {}), p = last;
    while(~p && !next[p][c]) next[p][c] = cur, p = link[p];
    if(!~p) set_link(cur, 0);
    else{
      int q = next[p][c];
      if(len[p] + 1 == len[q]) set_link(cur, q);
      else{
        int clone = new_state(len[p] + 1, link[q],
        firstpos[q], true, next[q]);
        while(~p && next[p][c] == q) next[p][c] = clone, p =
        link[p];
        set_link(cur, clone); set_link(q, clone);
      }
    }
    last = cur;
  }
  int size() const {  return (int)len.size(); } // # of
  states
}; suffix_automaton<int, initialized_array<int,26,0>> T;
// for(auto c : s) if((x=T.next[x][c]) == 0) return false;
```

4.6 Bitset LCS

```
#include <x86intrin.h>
template<size_t _Nw> void _M_do_sub(_Base_bitset<_Nw> &A,
const _Base_bitset<_Nw> &B){
  for(int i=0, c=0; i<_Nw; i++) c = _subborrow_u64(c,
  A._M_w[i], B._M_w[i], (ull*)&A._M_w[i]);
}
void _M_do_sub(_Base_bitset<1> &A, const _Base_bitset<1> &B){
A._M_w -= B._M_w; }
template<size_t _Nb> bitset<_Nb>& operator-=(bitset<_Nb> &A,
const bitset<_Nb> &B){
  _M_do_sub(A, B); return A;
}
template<size_t _Nb> inline bitset<_Nb> operator-(const
bitset<_Nb> &A, const bitset<_Nb> &B){
  bitset<_Nb> C(A); return C -= B;
}
char s[50050], t[50050];
int lcs(){ // O(NM/64)
  bitset<50050> dp, ch[26];
  int n = strlen(s), m = strlen(t);
  for(int i=0; i<m; i++) ch[t[i]-'A'].set(i);
  for(int i=0; i<n; i++){ auto x = dp | ch[s[i]-'A']; dp = dp
  - (dp ^ x) & x; }
  return dp.count();
}
```

4.7 Lyndon Factorization, Minimum Rotation

```
// link[i]: length of smallest suffix of s[0..i-1]
// factorization result: s[res[i]..res[i+1]-1]
vector<int> Lyndon(const string &s){
  int n = s.size(); vector<int> link(n);
  for(int i=0; i<n; ){
    int j=i+1, k=i; link[i] = 1;
    for(; j<n && s[k]<=s[j]; j++){
      if(s[j] == s[k]) link[j] = link[k], k++;
      else link[j] = j - i + 1, k = i;
    } for(; i<=k; i+=j-k);
  } vector<int> res;
  for(int i=n-1; i>=0; i-=link[i])
  res.push_back(i-link[i]+1);
  reverse(res.begin(), res.end()); return res;
}
// rotate(v.begin(), v.begin()+min_rotation(v), v.end());
template<typename T> int min_rotation(T s){ // O(N)
  int a = 0, N = s.size();
  for(int i=0; i<N; i++) s.push_back(s[i]);
  for(int b=0; b<N; b++) for(int k=0; k<N; k++){
    if(a+k == b || s[a+k] < s[b+k]){ b += max(0, k-1); break;
    }
    if(s[a+k] > s[b+k]){ a = b; break; }
  }
  return a;
}
```

4.8 All LCS

```
void AllLCS(const string &s, const string &t){
  vector<int> h(t.size()); iota(h.begin(), h.end(), 0);
  for(int i=0, v=-1; i<s.size(); i++, v=-1){
    for(int r=0; r<t.size(); r++){
      if(s[i] == t[r] || h[r] < v) swap(h[r], v);
      //LCS(s[0..i],t[l..r]) = r-l+1 - sum([h[x] >= l] | x <=
      r)
} /*for r*/ } /* for i */ } /* end*/
```

5 Misc

5.1 CMakeLists.txt

```
set(CMAKE_CXX_STANDARD 17)
set(CMAKE_CXX_FLAGS "-DLOCAL -lm -g -Wl,--stack,268435456")
add_compile_options(-Wall -Wextra -Winvalid-pch -Wfloat-equal
-Wno-sign-compare -Wno-misleading-indentation
-Wno-parentheses)
# add_compile_options(-O3 -mavx -mavx2 -mfma)
#pragma GCC optimize("Ofast,no-stack-protector")
#pragma GCC optimize("no-math-errno,unroll-loops")
#pragma GCC target("sse,sse2,sse3,ssse3,sse4")
#pragma GCC target("popcnt,abm,mmx,avx,arch=skylake")
__builtin_ia32_ldmxcsr(__builtin_ia32_stmxcsr()|0x8040)
```

5.2 Calendar

```
int f(int y,int m,int d){// 0: Sat, 1: Sun, ...
  if (m<=2) y--, m+=12; int c=y/100; y%=100;
  int w=((c>>2)-(c<<1)+y+(y>>2)+(13*(m+1)/5)+d-1)%7;
  if (w<0) w+=7; return w; }
```

5.3 Template

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
typedef pair<int, int> pii;
#define pb push_back
#define all(a) a.begin(), a.end()
#define sz(a) ((int)a.size())
#ifdef ABS
template <typename T>
ostream& operator << (ostream &o, vector <T> vec) {
    o << "{"; int f = 0;
    for (T i : vec) o << (f++ ? " " : "") << i;
    return o << "}"; }
void bug__(int c, auto ...a) {
    cerr << "\e[1;" << c << "m";
    (..., (cerr << a << " "));
    cerr << "\e[0m" << endl; }
#define bug_(c, x...) bug__(c, __LINE__, "[" + string(#x) +
"]", x)
#define bug(x...) bug_(32, x)
#define bugv(x...) bug_(36, vector(x))
#define safe bug_(33, "safe")
#else
#define bug(x...) void(0)
#define bugv(x...) void(0)
#define safe void(0)
#endif
const int mod = 998244353, N = 100000;

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

}
```

5.4 Stress Test

```
#!/usr/bin/env bash
g++ $1.cpp -o $1
g++ $2.cpp -o $2
g++ $3.cpp -o $3
for i in {1..100} ; do
  ./$3 > input.txt
  # st=$(date +%s%N)
  ./$1 < input.txt > output1.txt
  # echo "$((($(date +%s%N) - $st)/1000000))ms"
  ./$2 < input.txt > output2.txt
  if cmp --silent -- "output1.txt" "output2.txt" ; then
    continue
  fi
  echo Input:
  cat input.txt
  echo Your Output:
  cat output1.txt
  echo Correct Output:
  cat output2.txt
  exit 1
done
echo OK!
```

```
./stress.sh main good gen
```

5.5 Ternary Search

```
while(s + 3 <= e){
  T l = (s + s + e) / 3, r = (s + e + e) / 3;
  if(Check(l) > Check(r)) s = l; else e = r;
}// get minimum / when multiple answer, find minimum `s`
T mn = INF, idx = s;
for(T i=s; i<=e; i++) if(T now = Check(i); now < mn) mn =
now, idx = i;
```

5.6 Add/Mul Update, Range Sum Query

```
struct Lz{
  ll a, b; // constructor, clear(a = 1, b = 0)
  Lz& operator+=(const Lz &t); // a *= t.a, b = t.a * b + t.b
};
struct Ty{
  ll cnt, sum; // constructor cnt=1, sum=0
  Ty& operator += (const Ty &t); // cnt += t.cnt, sum +=
  t.sum
  Ty& operator += (const Lz &t); // sum= t.a * sum + cnt *
  t.b
};
```

5.7 $O(N \times \max W)$ Subset Sum (Fast Knapsack)

```
// O(N*maxW), maximize sumW <= t
int Knapsack(vector<int> w, int t){
  int a = 0, b = 0, x;
  while(b < w.size() && a + w[b] <= t) a += w[b++];
  if(b == w.size()) return a;
  int m = *max_element(w.begin(), w.end());
  vector<int> u, v(2*m, -1); v[a+m-t] = b;
  for(int i=b; (u=v,i<w.size()); i++){
    for(x=0; x<m; x++) v[x+w[i]] = max(v[x+w[i]], u[x]);
    for(x=2*m; --x>m; ) for(int j=max(0,u[x]); j<v[x]; j++)
    v[x-w[j]] = max(v[x-w[j]], j);
  } for(a=t; v[a+m-t]<0; a--);;; return a;
}
```

5.8 Monotone Queue Optimization

```
template<class T, bool GET_MAX = false> // D[i] = func_{0 <=
j < i} D[j] + cost(j, i)
pair<vector<T>, vector<int>> monotone_queue_dp(int n, const
vector<T> &init, auto cost){
  assert((int)init.size() == n + 1); // cost function ->
  auto, do not use std::function
  vector<T> dp = init; vector<int> prv(n+1);
  auto compare = [](T a, T b){ return GET_MAX ? a < b : a >
  b; };
  auto cross = [&](int i, int j){
    int l = j, r = n + 1;
    while(l < r){
      int m = (l + r + 1) / 2;
      if(compare(dp[i] + cost(i, m), dp[j] + cost(j, m))) r =
      m - 1; else l = m;
  } return l; };
  deque<int> q{0};
  for(int i=1; i<=n; i++){
    while(q.size() > 1 && compare(dp[q[0]] + cost(q[0], i),
    dp[q[1]] + cost(q[1], i))) q.pop_front();
    dp[i] = dp[q[0]] + cost(q[0], i); prv[i] = q[0];
    while(q.size() > 1 && cross(q[q.size()-2], q.back()) >=
    cross(q.back(), i)) q.pop_back();
    q.push_back(i);
  } /*for end*/ return {dp, prv}; }
```

5.9 Random, PBDS, Bit Trick, Bitset

```
mt19937
rd((unsigned)chrono::steady_clock::now().time_since_epoch().count
uniform_int_distribution<int> rnd_int(l, r); // rnd_int(rd)
uniform_real_distribution<double> rnd_real(0, 1);//
rnd_real(rd)
// ext/pb_ds/assoc_container.hpp, tree_policy.hpp, rope
// namespace __gnu_pbds (find_by_order, order_of_key)
// namespace __gnu_cxx (append(str), substr(l, r), at(idx))
template <typename T> using ordered_set = tree<T, null_type,
less<T>, rb_tree_tag, tree_order_statistics_node_update>;
bool next_combination(T &bit, int N){
  T x = bit & -bit, y = bit + x;
  bit = (((bit & ~y) / x) >> 1) | y;
  return (bit < (1LL << N)); }
long long next_perm(long long v){
  long long t = v | (v-1);
  return (t + 1) | (((~t & -~t) - 1) >> (__builtin_ctz(v) +
  1));
} // __builtin_clz/ctz/popcount
for(submask=mask; submask; submask=(submask-1)&mask);
for(supermask=mask; supermask<(1<<n);
supermask=(supermask+1)|mask);
int frq(int n, int i) { int j, r = 0; // # of digit i in [1,
n]
  for (j = 1; j <= n; j *= 10) if (n / j / 10 >= !i) r += (n
  / 10 / j - !i) * j + (n / j % 10 > i ? j : n / j % 10 == i
  ? n % j + 1 : 0);
  return r; }
bitset<17> bs; bs[1] = bs[7] = 1; assert(bs._Find_first() ==
1);
assert(bs._Find_next(0) == 1 && bs._Find_next(1) == 7);
assert(bs._Find_next(3) == 7 && bs._Find_next(7) == 17);
cout << bs._Find_next(7) << "\n";
template <int len = 1> // Arbitrary sized bitset
void solve(int n){ // solution using bitset<len>
  if(len < n){ solve<std::min(len*2, MAXLEN)>(n); return; } }
```

5.10 Fast I/O, Fast Div, Fast Mod

```
namespace io { // thanks to cgiosy
  const signed IS=1<<20; char I[IS+1],*J=I;
  inline void daer(){if(J>=I+IS-64){
    char*p=I;do*p++=*J++;
    while(J!=I+IS);p[read(0,p,I+IS-p)]=0;J=I;}}
  template<int N=10,typename T=int>inline T getu(){
    daer();T x=0;int k=0;do x=x*10+*J-'0';
    while(*++J>='0'&&++k<N);++J;return x;}
```

```
  template<int N=10,typename T=int>inline T geti(){
    daer();bool e=*J=='-';J+=e;return(e?-1:1)*getu<N,T>();}
  struct f{f(){I[read(0,I,IS)]=0;}}flu; };
struct FastMod{ // typedef __uint128_t L;
  ull b, m; FastMod(ull b) : b(b), m(ull((L(1) << 64) / b))
  {}
  ull reduce(ull a){ // can be proven that 0 <= r < 2*b
    ull q = (ull)((L(m) * a) >> 64), r = a - q * b;
    return r >= b ? r - b : r;
} };
inline pair<uint32_t, uint32_t> Div(uint64_t a, uint32_t b){
  if(__builtin_constant_p(b)) return {a/b, a%b};
  uint32_t lo=a, hi=a>>32;
  __asm__("div %2" : "+a,a" (lo), "+d,d" (hi) : "r,m" (b));
  return {lo, hi}; // BOJ 27505, q r < 2^32
} // divide 10M times in ~400ms
ull mulmod(ull a, ull b, ull M){ // ~2x faster than int128
  ll ret = a * b - M * ull(1.L / M * a * b);
  return ret + M * (ret < 0) - M * (ret >= (ll)M);
} // safe for 0 ≤ a,b < M < (1<<63) when long double is
80bit
```

5.11 DP Optimization

```
// Quadrangle Inequality : C(a, c)+C(b, d) ≤ C(a, d)+C(b, c)
// Monotonicity : C(b, c) ≤ C(a, d)
// CHT, DnC Opt(Quadrangle), Knuth(Quadrangle and
Monotonicity)
// Knuth: K[i][j-1] <= K[i][j] <= K[i+1][j]
// 1. Calculate D[i][i], K[i][i]
// 2. Calculate D[i][j], K[i][j] (i < j)
// Another: D[i][j] = min(D[i-1][k] + C[k+1][j]), C
quadrangle
// i=1..k  j=n..1  k=K[i-1,j]..K[i,j+1] update,
vnoi/icpc22_mn_c
```

5.12 Highly Composite Numbers, Large Prime

```
< 10^k            number     divisors   2 3 5 71113171923293137
--------------------------------------------------------------
1                      6            4   1 1
2                     60           12   2 1 1
3                    840           32   3 1 1 1
4                   7560           64   3 3 1 1
5                  83160          128   3 3 1 1 1
6                 720720          240   4 2 1 1 1 1
7                8648640          448   6 3 1 1 1 1
8               73513440          768   5 3 1 1 1 1 1
9              735134400         1344   6 3 2 1 1 1 1
10            6983776800         2304   5 3 2 1 1 1 1 1
11           97772875200         4032   6 3 2 2 1 1 1 1
12          963761198400         6720   6 4 2 1 1 1 1 1 1
13         9316358251200        10752   6 3 2 1 1 1 1 1 1 1
14        97821761637600        17280   5 4 2 2 1 1 1 1 1 1
15       866421317361600        26880   6 4 2 1 1 1 1 1 1 1 1
16      8086598962041600        41472   8 3 2 2 1 1 1 1 1 1 1
17     74801040398884800        64512   6 3 2 2 1 1 1 1 1 1 1 1
18    897612484786617600       103680   8 4 2 2 1 1 1 1 1 1 1 1

< 10^k     prime   # of prime        < 10^k                prime
-----------------------------------------------------------------
1              7            4            10           9999999967
2             97           25            11          99999999977
3            997          168            12         999999999989
4           9973         1229            13        9999999999971
5          99991         9592            14       99999999999973
6         999983        78498            15      999999999999989
7        9999991       664579            16     9999999999999937
8       99999989      5761455            17    99999999999999997
9      999999937     50847534            18   999999999999999989
```

5.13 Python Decimal

```python
from fractions import Fraction
from decimal import Decimal, getcontext
getcontext().prec = 250 # set precision
N, two, itwo = 200, Decimal(2), Decimal(0.5)
# sin(x) = sum (-1)^n x^(2n+1) / (2n+1)!
# cos(x) = sum (-1)^n x^(2n) / (2n)!
def angle(cosT):
  #given cos(theta) in decimal return theta
  for i in range(N): cosT=((cosT+1)/two)**itwo
  sinT = (1-cosT*cosT)**itwo
  return sinT*(2**N)
pi = angle(Decimal(-1))
```

6 Notes

6.1 Calculus, Newton's Method

- Implicit differentiation: Differentiate both sides of $f(x, y) = 0$ with respect to x, then solve for dy/dx.
- (Example) $\frac{d}{dx}(x^3) + \frac{d}{dx}(y^3) - 3\frac{d}{dx}(xy) = 3x^2 + 3y^2\frac{dy}{dx} - 3(y + x\frac{dy}{dx}) = 0$
- Derivative of the inverse function: $(f^{-1})'(x) = 1/f'(f^{-1}(x))$
- Newton–Raphson method: $x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$
- Substitution in integration: Let $x = g(t)$, then $\int f(x)\,dx = \int f(g(t))g'(t)\,dt$
- (Example) In $\int \frac{f'(x)}{f(x)}dx$, let $t = f(x)$, so $f'(x) = dt/dx$.
  Therefore, $\int \frac{f'(x)}{f(x)}dx = \int \frac{1}{t}\,dt = \ln|t| + C = \ln|f(x)| + C$
- Trigonometric substitution: For $\sqrt{a^2 - x^2}$, let $x = a\sin t$; For $\sqrt{a^2 + x^2}$, let $x = a\tan t$; Be careful with the range of t.
- Volume of a solid: If the cross-sectional area function $A(x)$ is continuous on $[a, b]$, then the volume is $\int_a^b A(x)\,dx$.
- (Disk method): If a continuous function $f(x)$ on $[a, b]$ satisfies $f(x) \geq 0$, the volume of the solid obtained by rotating the region bounded by $f(x)$, $x = a$, $x = b$, and the x-axis about the y-axis is $\int_a^b 2\pi x f(x)\,dx$.
- Arc length: If $f'(x)$ is continuous on $[a, b]$, the arc length from $x = a$ to $x = b$ is $\int_a^b \sqrt{1 + [f'(x)]^2}\,dx$.
- Surface area of revolution: When the curve is rotated about the x-axis, the surface area is $\int_a^b 2\pi f(x)\sqrt{1 + [f'(x)]^2}\,dx$.
- Green's theorem: $\oint_C (L\,dx + M\,dy) = \iint_D \left(\frac{\partial M}{\partial x} - \frac{\partial L}{\partial y}\right)dx\,dy$
- where C is positively oriented, piecewise smooth, simple, and closed; D is the region enclosed by C; L and M have continuous partial derivatives on D.

$f(x)$	$f'(x)$	$\int f(x)\,dx$
$\sin x$	$\cos x$	$-\cos x$
$\cos x$	$-\sin x$	$\sin x$
$\tan x$	$\sec^2 x = 1 + \tan^2 x$	$-\ln\lvert\cos x\rvert$
$\csc x$	$-\csc x \cot x$	$\ln\lvert\tan(x/2)\rvert$
$\sec x$	$\sec x \tan x$	$\ln\lvert\tan(x/2 + \pi/4)\rvert$
$\cot x$	$-\csc^2 x$	$\ln\lvert\sin x\rvert$
$\arcsin x$	$\frac{1}{\sqrt{1-x^2}}$	$x\arcsin x + \sqrt{1-x^2}$
$\arccos x$	$-\frac{1}{\sqrt{1-x^2}}$	$x\arccos x - \sqrt{1-x^2}$
$\arctan x$	$\frac{1}{1+x^2}$	$x\arctan x - \frac{\ln(x^2+1)}{2}$
$\csc^{-1} x$	$-\frac{1}{x\sqrt{x^2-1}}$	$x\csc^{-1} x + \cosh^{-1}\lvert x\rvert$
$\sec^{-1} x$	$\frac{1}{x\sqrt{x^2-1}}$	$x\sec^{-1} x - \cosh^{-1}\lvert x\rvert$
$\cot^{-1} x$	$-\frac{1}{1+x^2}$	$x\cot^{-1} x + \frac{\ln(x^2+1)}{2}$

6.2 Zeta/Mobius Transform

- Subset Zeta/Mobius Transform(n=sz=2^k,i*=2)
  - for i=1..n-1 for j=0..n-1 if(i and j) v[j] ±= v[i xor j]
- Superset Zeta/Mobius Transform(n=sz=2^k,i*=2)
  - for i=1..n-1 for j=0..n-1 if(i and j) v[i xor j] ±= v[j]
- Divisor Zeta/Mobius Transform(n=sz-1)
  - for p:Prime for i=1..n/p v[i*p] += v[i]
  - for p:Prime for i=n/p..1 v[i*p] -= v[i]
- Multiple Zeta/Mobius Transform(n=sz-1)
  - for p:Prime for i=n/p..1 v[i] += v[i*p]
  - for p:Prime for i=1..n/p v[i] -= v[i*p]
- AND Convolution: SupZeta(A), SupZeta(B), SupMobius(mul)
- OR Conv.: Subset, GCD Conv.: Multiple, LCM Conv.: Divisor
- AND/OR: 2^20 0.3s, Subset: 2^{20} 2.5s, GCD/LCM: 1e6 0.3s

6.3 Generating Function

- Arithmetic sequence: $(pn + q)x^n = p/(1 - x) + q/(1 - x)^2$
- Geometric sequence: $(rx)^n = (1 - rx)^{-1}$
- Combination: $C(m, n)x^n = (1 + x)^m$
- Combination with repetition: $C(m + n - 1, n)x^n = (1 - x)^{-m}$
- EGF of $f(n) = \sum_{k=0}^{n} k! \times S_2(n, k)$: $1/(2 - e^x)$
- Ordinary Generating Function (OGF) $A(x) = \sum_{i\geq 0} a_i x^i$
  $A(rx) \Rightarrow r^n a_n$, $xA(x)' \Rightarrow na_n$
  $A(x) + B(x) \Rightarrow a_n + b_n$, $A(x)B(x) \Rightarrow \sum_{i=0}^{n} a_i b_{n-i}$
  $A(x)^k \Rightarrow \sum_{i_1+i_2+\cdots+i_k=n} a_{i_1}a_{i_2}\ldots a_{i_k}$
  $\frac{A(x)}{1-x} \Rightarrow \sum_{i=0}^{n} a_i$
- Exponential Generating Function (EGF) $A(x) = \sum_{i\geq 0} \frac{a_i}{i!}x^i$
  $A(x) + B(x) \Rightarrow a_n + b_n$, $A(x)B(x) \Rightarrow \sum_{i=0}^{n} \binom{n}{i} a_i b_{n-i}$
  $A^{(k)}(x) \Rightarrow a_{n+k}$, $xA(x) \Rightarrow na_n$
  $A(x)^k \Rightarrow \sum_{i_1+i_2+\cdots+i_k=n} \binom{n}{i_1, i_2, \ldots, i_k} a_{i_1}a_{i_2}\ldots a_{i_k}$

6.4 Counting

- Bernoulli numbers
  $B_0 = 1, B_1^{\pm} = \pm\frac{1}{2}, B_2 = \frac{1}{6}, B_3 = 0$

$$\sum_{j=0}^{m}\binom{m+1}{j}B_j = 0, \text{ EGF is } B(x) = \frac{x}{e^x-1} = \sum_{n=0}^{\infty} B_n \frac{x^n}{n!}.$$

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

- Stirling numbers of the second kind Partitions of n distinct elements into exactly k groups.
  $S(n,k) = S(n-1,k-1) + kS(n-1,k), S(n,1) = S(n,n) = 1$
  $S(n,k) = \frac{1}{k!}\sum_{i=0}^{k}(-1)^{k-i}\binom{k}{i}i^n$
  $x^n = \sum_{i=0}^{n} S(n,i)(x)_i$
- Pentagonal number theorem
  $$\prod_{n=1}^{\infty}(1-x^n) = 1 + \sum_{k=1}^{\infty}(-1)^k\left(x^{k(3k+1)/2} + x^{k(3k-1)/2}\right)$$
- Catalan numbers
  $C_n^{(k)} = \frac{k+1}{n+k+1}\binom{2n+k}{n}$
  $C^{(k)}(x) = 1 + x[C^{(k)}(x)]^k$
- Eulerian numbers
  Number of permutations $\pi \in S_n$ in which exactly k elements are greater than the previous element. k j:s s.t. $\pi(j) > \pi(j+1)$, $k+1$ j:s s.t. $\pi(j) \geq j$, k j:s s.t. $\pi(j) > j$.
  $E(n,k) = (n-k)E(n-1,k-1) + (k+1)E(n-1,k)$
  $E(n,0) = E(n,n-1) = 1$
  $E(n,k) = \sum_{j=0}^{k}(-1)^j\binom{n+1}{j}(k+1-j)^n$

6.5 Faulhaber's Formula ($\sum_{k=1}^{n} k^c$)

- B_n: Bernoulli numbers
- Exponential generating function (EGF): $\frac{x}{e^x-1} = \frac{1}{(e^x-1)/x} = \sum_{n=0}^{\infty}\frac{B_n}{n!}x^n$
- General term: $B_n = \sum_{k=0}^{n}\frac{k!(-1)^k}{k+1}S_2(n,k)$
- Recurrence relation: $B_0 = 1$; $B_n = -\frac{1}{n+1}\sum_{r=0}^{n-1}\binom{n+1}{r}B_r$
- Power sum formula: $\sum_{k=1}^{n} k^c = \sum_{k=0}^{c}\frac{(-1)^k}{c+1}\binom{c+1}{k}B_k n^{c+1-k}$

6.6 About Graph Degree Sequence

- Simple undirected graph (Erdős–Gallai theorem): For a degree sequence $d_1 \geq \cdots \geq d_n$, the sum of all degrees is even **and** for all $1 \leq k \leq n$, $\sum_{i=1}^{k} d_i \leq k(k-1) + \sum_{i=k+1}^{n}\min(d_i,k)$.
- Simple bipartite graph (Gale–Ryser theorem): For degree sequences $a_1 \geq \cdots \geq a_n$ and b_i, sum(a) = sum(b) **and** for all $1 \leq k \leq n$, $\sum_{i=1}^{k} a_i \leq \sum_{i=1}^{n}\min(b_i,k)$.
- Simple directed graph (Fulkerson–Chen–Anstee theorem): For in-degree/out-degree pairs $(a_1,b_1),\cdots,(a_n,b_n)$ satisfying $a_1 \geq \cdots \geq a_n$, sum(a) = sum(b) **and** for all $1 \leq k \leq n$, $\sum_{i=1}^{k} a_i \leq \sum_{i=1}^{k}\min(b_i,k-1) + \sum_{i=k+1}^{n}\min(b_i,k)$.

6.7 Burnside, Grundy, Pick, Hall, Simpson, Area of Quadrangle, Fermat Point, Euler, Pythagorean

- Burnside's Lemma
  - Formula
  For a group $G = (X,A)$ with set X and action A, $|A||X/A| = \sum(|\text{Fixed points of a}|, \text{ for all } a \in A)$
  X/A is the set of equivalence classes (orbits) obtained by grouping elements of X that can be transformed into each other by the action.
  - Explanation
  Orbit: For a group and an action f, connect a,b with an edge if $f(a) = b$; each connected component is an orbit.
  Number of orbits = sum of (number of fixed points of each action g) divided by the number of actions.
  - Degrees of freedom cheat sheet
  n rotations: fixed points of rotation $i = \gcd(n,i)$
  n odd reflections: $(n+1)/2$ symmetry axes (fixed points)
  n even reflections: $n/2$ axes through vertices (fixed points $n/2+1$) + $n/2$ axes through edges (fixed points $n/2$)
- Algorithmic Games
  - Nim Game (last to take wins): XOR $= 0 \Rightarrow$ second player wins, otherwise first player wins.
  - Subtraction Game: if one can take up to k stones per turn, compute each pile mod $(k+1)$ and XOR the results.
  - Index-k Nim: one can choose up to k piles and remove any number from each; for each binary digit, sum bits across piles and take mod $(k+1)$; if all digits $\equiv 0$, second player wins, otherwise first wins.
  - Misère Nim: if all piles contain 1 stone $\Rightarrow$ odd $N \to$ second wins; otherwise, same rule as normal Nim (XOR $= 0 \to$ second wins).
- Pick's Theorem
  For a simple polygon with lattice vertices: $A = I + \frac{B}{2} - 1$, where I = number of interior lattice points, B = number of boundary lattice points, A = area.
- Hall's Marriage Theorem
  In a bipartite graph (L,R), a perfect matching covering all L exists iff for every subset $S \subseteq L$, $|S| \leq |N(S)|$, where $N(S)$ is the set of neighbors of S in R.
- Simpson's Rule (Integration)
  $S_n(f) = \frac{h}{3}[f(x_0) + f(x_n) + 4\sum f(x_{2i+1}) + 2\sum f(x_{2i})]$
  If $M = \max|f^{(4)}(x)|$, then the error bound is $E_n \leq \frac{M(b-a)}{180}h^4$.
- Brahmagupta's Formula
  For a cyclic quadrilateral with side lengths a,b,c,d: $S = \sqrt{(s-a)(s-b)(s-c)(s-d)}$, where $s = (a+b+c+d)/2$.
- Bretschneider's Formula
  For any quadrilateral with sides a,b,c,d and sum of opposite angles $= 2\theta$: $S = \sqrt{(s-a)(s-b)(s-c)(s-d) - abcd \times \cos^2\theta}$.
- Fermat Point
  The point minimizing the sum of distances to the three vertices of a triangle.
  If one angle $\geq 120°$, that vertex is the Fermat point. Otherwise, construct equilateral triangles on each side, connect the new vertices to the opposite original vertices — the intersection is the Fermat point.
  If all angles $< 120°$, minimal sum of distances $= \sqrt{(a^2+b^2+c^2+4\sqrt{3}S)/2}$, where S is the area.
- Euler's Theorem
  For coprime integers a,n: $a^{\phi(n)} \equiv 1 \pmod n$
  For all integers: $a^n \equiv a^{n-\phi(n)} \pmod n$
  If $m \geq \log_2 n$, then $a^m \equiv a^{m\%\phi(n)+\phi(n)} \pmod n$.
- $g^0 + g^1 + g^2 + \cdots g^{p-2} \equiv -1 \pmod p$ iff $g = 1$, otherwise 0.
- If $n \equiv 0 \pmod 2$, then $1^n + 2^n + \cdots + (n-1)^n \equiv 0 \pmod n$.
- Eulerian Numbers
  Number of permutations $\pi \in S_n$ in which exactly k elements are greater than the previous one.
  $E(n,k) = (n-k)E(n-1,k-1) + (k+1)E(n-1,k)$
  $E(n,0) = E(n,n-1) = 1$
  $E(n,k) = \sum_{j=0}^{k}(-1)^j\binom{n+1}{j}(k+1-j)^n$
- Pythagorean Triple
  Primitive triples (a,b,c) satisfying $a^2+b^2=c^2$ are generated by $(a,b,c) = (st, \frac{s^2-t^2}{2}, \frac{s^2+t^2}{2})$, where $\gcd(s,t) = 1$, $s > t$.

6.8 About Graph Minimum Cut

- A problem of minimizing cost by assigning N boolean variables $v_1,\cdots,v_n$ can be represented as a **minimum cut problem**, where nodes assigned `true` are connected to T, and those assigned `false` are connected to F.
  1. When v_i is `true`, a cost occurs $\Rightarrow$ add an edge from i to F with that cost.
  2. When v_i is `false`, a cost occurs $\Rightarrow$ add an edge from i to T with that cost.
  3. When v_i is `true` and v_j is `false`, a cost occurs $\Rightarrow$ add an edge from i to j with that cost.
  4. When $v_i \neq v_j$, a cost occurs $\Rightarrow$ add edges both $i \to j$ and $j \to i$ with that cost.
  5. If v_i being `true` implies v_j must also be `true`: add an infinite-capacity edge $i \to j$.
  6. If v_i being `false` implies v_j must also be `false`: add an infinite-capacity edge $j \to i$.
- If you combine rules (5) and (6) with the constraint that v_i and v_j must differ, the problem becomes equivalent to **MAX-2SAT**.
- **Maximum Density Subgraph** (NEERC'06H, BOJ 3611 "Team Difficulty"):
  - Use binary search on x to check whether there exists a subgraph with density $\geq x$.
  - Given a graph with N vertices, M edges, and degrees D_i.
  - For each edge, add bidirectional edges with capacity 1.
  - From the source to each vertex: add capacity M. From each vertex to the sink: add capacity $M - D_i + 2x$.
  - In the resulting min-cut, if there is at least one vertex connected to S, then a subgraph with density $\geq x$ exists — those vertices form that subgraph.
  - Use the condition `while (r - l ≥ 1.0 / (n * n))` to control precision; iterating too many times causes floating-point errors.

6.9 Matrix with Graph(Kirchhoff, Tutte, LGV)

- Kirchhoff's Theorem
  Denote L be a $n \times n$ matrix as the Laplacian matrix of graph G, where $L_{ii} = d(i)$, $L_{ij} = -c$ where c is the number of edge (i,j) in G.
  - The number of undirected spanning in G is $|\det(\tilde{L}_{11})|$.
  - The number of directed spanning tree rooted at r in G is $|\det(\tilde{L}_{rr})|$.
- Tutte's Matrix
  Let D be a $n \times n$ matrix, where $d_{ij} = x_{ij}$ (x_{ij} is chosen uniformly at random) if $i < j$ and $(i,j) \in E$, otherwise $d_{ij} = -d_{ji}$. $\frac{rank(D)}{2}$ is the maximum matching on G.

- Erdős–Gallai Theorem
  A sequence of non-negative integers $d_1 \geq d_2 \geq \ldots \geq d_n$ can be represented as the degree sequence of a finite simple graph on n vertices if and only if $d_1 + d_2 + \ldots + d_n$ is even and
  $$\sum_{i=1}^{k} d_i \leq k(k-1) + \sum_{i=k+1}^{n} \min(d_i, k)$$
  holds for all $1 \leq k \leq n$.
- Burnside's Lemma
  Let X be a set and G be a group that acts on X. For $g \in G$, denote by X^g the elements fixed by g:
  $$X^g = \{x \in X \mid gx \in X\}$$
  Then
  $$|X/G| = \frac{1}{|G|} \sum_{g \in G} |X^g|.$$
- Gale–Ryser theorem
  A pair of sequences of nonnegative integers $a_1 \geq \cdots \geq a_n$ and $b_1, \ldots, b_n$ is bigraphic if and only if $\sum_{i=1}^{n} a_i = \sum_{i=1}^{n} b_i$ and $\sum_{i=1}^{k} a_i \leq \sum_{i=1}^{n} \min(b_i, k)$ holds for every $1 \leq k \leq n$. Sequences a and b called bigraphic if there is a labeled simple bipartite graph such that a and b is the degree sequence of this bipartite graph.
- Fulkerson–Chen–Anstee theorem
  A sequence $(a_1, b_1), \ldots, (a_n, b_n)$ of nonnegative integer pairs with $a_1 \geq \cdots \geq a_n$ is digraphic if and only if $\sum_{i=1}^{n} a_i = \sum_{i=1}^{n} b_i$ and $\sum_{i=1}^{k} a_i \leq \sum_{i=1}^{k} \min(b_i, k-1) + \sum_{i=k+1}^{n} \min(b_i, k)$ holds for every $1 \leq k \leq n$. Sequences a and b called digraphic if there is a labeled simple directed graph such that each vertex v_i has indegree a_i and outdegree b_i.
- Pick's theorem
  For simple polygon, when points are all integer, we have $A = \#\{\text{lattice points in the interior}\} + \frac{\#\{\text{lattice points on the boundary}\}}{2} - 1$
- Spherical cap
  - A portion of a sphere cut off by a plane.
  - r: sphere radius, a: radius of the base of the cap, h: height of the cap, θ: $\arcsin(a/r)$.
  - Volume $= \pi h^2(3r-h)/3 = \pi h(3a^2+h^2)/6 = \pi r^3(2+\cos\theta)(1-\cos\theta)^2/3$.
  - Area $= 2\pi rh = \pi(a^2+h^2) = 2\pi r^2(1-\cos\theta)$.

6.10 About Graph Matching(Graph with $|V| \leq 500$)

- **Game on a Graph**: A token starts at vertex s. Players alternately move the token to an adjacent vertex; if a player cannot move, they lose.
  $\Leftrightarrow$ The second player wins if and only if there exists a maximum matching that does not include s.
- **Chinese Postman Problem**: Find a minimum-weight walk that visits every edge at least once.
  Run Floyd–Warshall to get all-pairs shortest paths, then collect all odd-degree vertices and find a minimum-weight perfect matching among them. (The number of odd vertices is always even.)
- **Unweighted Edge Cover**: Find the smallest (minimum cardinality) set of edges that covers all vertices.
  Result: $|V|-|M|$, where M is a maximum matching. There are no paths of length 3; the structure consists of multiple star graphs.
- **Weighted Edge Cover**: $\sum_{v \in V} w(v) - \sum_{(u,v) \in M} \big(w(u) + w(v) - d(u,v)\big)$, where $w(x)$ is the minimum weight of an edge incident to vertex x.
- **NEERC'18 B**: Each machine requires two workers to operate. For each machine, create two vertices and connect them with an edge; the answer is $|M| - |\text{machines}|$. It helps to think of each edge as contributing 1/2 to the answer.
- **Minimum Disjoint Cycle Cover**: Find a set of vertex-disjoint cycles (each of length ≥ 3) covering all vertices.
  Each vertex must be incident to exactly two different edges. While this might seem expressible as a flow, edges with capacity 2 can only carry 1 unit of flow — so a standard flow model fails. Instead, duplicate every vertex and edge (e.g. (v, v'), $(e_{i,u}, e_{i,v})$). For each edge $e = (u,v)$, connect e_u and e_v with an edge of weight w (similar to NEERC'18), and connect $(u, e_{i,u}), (u', e_{i,u}), (v, e_{i,v}), (v', e_{i,v})$ with zero-weight edges. A perfect matching exists $\Leftrightarrow$ a disjoint cycle cover exists. After finding a maximum-weight matching, subtract the total matching weight from the sum of all edge weights to get the result.
- **Two Matching**: Find a maximum-weight matching where each vertex can be incident to at most two edges.
  Each connected component must be either a single vertex, a path, or a cycle. Add zero-weight edges between every pair of distinct vertices and also add a zero-weight (v, v') edge — this turns the problem into the Disjoint Cycle Cover problem. A component with a single vertex can be treated as having a self-loop; for path components, it helps to think of connecting the two endpoints.