CU-Later Code Library

yangster67, ikaurov, serichaoo

May 21th 2024

Contents

Templates

Ken's template

```cpp
#include <bits/stdc++.h>
using namespace std;
#define all(v) (v).begin(), (v).end()
typedef long long ll;
typedef long double ld;
#define pb push_back
#define sz(x) (int)(x).size()
#define fi first
#define se second
#define endl '\n'
```

Kevin's template

```cpp
// paste Kaurov's Template, minus last line
typedef vector<int> vi;
typedef vector<ll> vll;
typedef pair<int, int> pii;
typedef pair<ll, ll> pll;
typedef pair<double, double> pdd;
const ld PI = acosl(-1);
const ll mod7 = 1e9 + 7;
const ll mod9 = 998244353;
const ll INF = 2*1024*1024*1023;
const char nl = '\n';
#define forn(i, n) for (int i = 0; i < int(n); i++)
#pragma GCC target("avx2,bmi,bmi2,lzcnt,popcnt")
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;
template<class T> using ordered_set = tree<T, null_type, less<T>, rb_tree_tag, tree_order_statistics_node_update>;
ll d, l, r, k, n, m, p, q, u, v, w, x, y, z;
string s, t;
vi d4x = {1, 0, -1, 0};
vi d4y = {0, 1, 0, -1};
vi d8x = {1, 0, -1, 0, 1, 1, -1, -1};
vi d8y = {0, 1, 0, -1, 1, -1, 1, -1};
mt19937 rng(chrono::steady_clock::now().time_since_epoch().count());


bool multiTest = 1;
void solve(int tt){
}

int main(){
  ios::sync_with_stdio(0);cin.tie(0);cout.tie(0);
  cout<<fixed<< setprecision(14);

  int t = 1;
  if (multiTest) cin >> t;
  forn(ii, t) solve(ii);
}
```

Geometry

```cpp
template<typename T>
struct TPoint{
  T x, y;
  int id;
  static constexpr T eps = static_cast<T>(1e-9);
  TPoint() : x(0), y(0), id(-1) {}
  TPoint(const T& x_, const T& y_) : x(x_), y(y_), id(-1) {}
  TPoint(const T& x_, const T& y_, const int id_) : x(x_), y(y_), id(id_) {}

  TPoint operator + (const TPoint& rhs) const {
    return TPoint(x + rhs.x, y + rhs.y);
  }
  TPoint operator - (const TPoint& rhs) const {
    return TPoint(x - rhs.x, y - rhs.y);
  }
  TPoint operator * (const T& rhs) const {
    return TPoint(x * rhs, y * rhs);
  }
  TPoint operator / (const T& rhs) const {
    return TPoint(x / rhs, y / rhs);
  }
  TPoint ort() const {
    return TPoint(-y, x);
  }
  T abs2() const {
    return x * x + y * y;
  }
};
template<typename T>
bool operator< (TPoint<T>& A, TPoint<T>& B){
  return make_pair(A.x, A.y) < make_pair(B.x, B.y);
}
template<typename T>
bool operator== (TPoint<T>& A, TPoint<T>& B){
  return abs(A.x - B.x) <= TPoint<T>::eps && abs(A.y - B.y) <= TPoint<T>::eps;
}
template<typename T>
struct TLine{
  T a, b, c;
  TLine() : a(0), b(0), c(0) {}
  TLine(const T& a_, const T& b_, const T& c_) : a(a_), b(b_), c(c_) {}
  TLine(const TPoint<T>& p1, const TPoint<T>& p2){
    a = p1.y - p2.y;
    b = p2.x - p1.x;
    c = -a * p1.x - b * p1.y;
  }
};
template<typename T>
T det(const T& a11, const T& a12, const T& a21, const T& a22){
  return a11 * a22 - a12 * a21;
}
template<typename T>
T sq(const T& a){
  return a * a;
}
template<typename T>
T smul(const TPoint<T>& a, const TPoint<T>& b){
  return a.x * b.x + a.y * b.y;
}
template<typename T>
T vmul(const TPoint<T>& a, const TPoint<T>& b){
  return det(a.x, a.y, b.x, b.y);
}
template<typename T>
bool parallel(const TLine<T>& l1, const TLine<T>& l2){
  return abs(vmul(TPoint<T>(l1.a, l1.b), TPoint<T>(l2.a, l2.b))) <= TPoint<T>::eps;
}
template<typename T>
bool equivalent(const TLine<T>& l1, const TLine<T>& l2){
  return parallel(l1, l2) &&
  abs(det(l1.b, l1.c, l2.b, l2.c)) <= TPoint<T>::eps &&
  abs(det(l1.a, l1.c, l2.a, l2.c)) <= TPoint<T>::eps;
}
template<typename T>
TPoint<T> intersection(const TLine<T>& l1, const TLine<T>& l2){
  return TPoint<T>(
    det(-l1.c, l1.b, -l2.c, l2.b) / det(l1.a, l1.b, l2.a, l2.b),
    det(l1.a, -l1.c, l2.a, -l2.c) / det(l1.a, l1.b, l2.a, l2.b)
  );
}
template<typename T>
int sign(const T& x){
  if (abs(x) <= TPoint<T>::eps) return 0;
  return x > 0? +1 : -1;
}
template<typename T>
T area(const vector<TPoint<T>>& pts){
  int n = sz(pts);
  T ans = 0;
  for (int i = 0; i < n; i++){
    ans += vmul(pts[i], pts[(i + 1) % n]);
  }
  return abs(ans) / 2;
}
template<typename T>
T dist_pp(const TPoint<T>& a, const TPoint<T>& b){
  return sqrt(sq(a.x - b.x) + sq(a.y - b.y));
}
template<typename T>
TLine<T> perp_line(const TLine<T>& l, const TPoint<T>& p){
  T na = -l.b, nb = l.a, nc = - na * p.x - nb * p.y;
  return TLine<T>(na, nb, nc);
}
template<typename T>
TPoint<T> projection(const TPoint<T>& p, const TLine<T>& l){
  return intersection(l, perp_line(l, p));
}
template<typename T>
T dist_pl(const TPoint<T>& p, const TLine<T>& l){
  return dist_pp(p, projection(p, l));
}
template<typename T>
struct TRay{
  TLine<T> l;
  TPoint<T> start, dirvec;
  TRay() : l(), start(), dirvec() {}
  TRay(const TPoint<T>& p1, const TPoint<T>& p2){
    l = TLine<T>(p1, p2);
    start = p1, dirvec = p2 - p1;
  }
};
template<typename T>
bool is_on_line(const TPoint<T>& p, const TLine<T>& l){
  return abs(l.a * p.x + l.b * p.y + l.c) <= TPoint<T>::eps;
}
template<typename T>
bool is_on_ray(const TPoint<T>& p, const TRay<T>& r){
  if (is_on_line(p, r.l)){
    return sign(smul(r.dirvec, TPoint<T>(p - r.start))) != -1;
  }
  else return false;
}
template<typename T>
bool is_on_seg(const TPoint<T>& P, const TPoint<T>& A, const TPoint<T>& B){
  return is_on_ray(P, TRay<T>(A, B)) && is_on_ray(P, TRay<T>(B, A));
}
template<typename T>
T dist_pr(const TPoint<T>& P, const TRay<T>& R){
  auto H = projection(P, R.l);
  return is_on_ray(H, R)? dist_pp(P, H) : dist_pp(P, R.start);
}
template<typename T>
T dist_ps(const TPoint<T>& P, const TPoint<T>& A, const TPoint<T>& B){
  auto H = projection(P, TLine<T>(A, B));
  if (is_on_seg(H, A, B)) return dist_pp(P, H);
  else return min(dist_pp(P, A), dist_pp(P, B));
}
template<typename T>
bool acw(const TPoint<T>& A, const TPoint<T>& B){
  T mul = vmul(A, B);
  return mul > 0 || abs(mul) <= TPoint<T>::eps;
}
template<typename T>
bool cw(const TPoint<T>& A, const TPoint<T>& B){
  T mul = vmul(A, B);
  return mul < 0 || abs(mul) <= TPoint<T>::eps;
}
template<typename T>
vector<TPoint<T>> convex_hull(vector<TPoint<T>> pts){
  sort(all(pts));
  pts.erase(unique(all(pts)), pts.end());
  vector<TPoint<T>> up, down;
  for (auto p : pts){
    while (sz(up) > 1 && acw(up.end()[-1] - up.end()[-2], p - up.end()[-2])) up.pop_back();
    while (sz(down) > 1 && cw(down.end()[-1] - down.end()[-2], p - down.end()[-2])) down.pop_back();
    up.pb(p), down.pb(p);
  }
  for (int i = sz(up) - 2; i >= 1; i--) down.pb(up[i]);
  return down;
}

template<typename T>
bool in_triangle(TPoint<T>& P, TPoint<T>& A, TPoint<T>& B, TPoint<T>& C){
  if (is_on_seg(P, A, B) || is_on_seg(P, B, C) || is_on_seg(P, C, A)) return true;
  return cw(P - A, B - A) == cw(P - B, C - B) &&
  cw(P - A, B - A) == cw(P - C, A - C);
}
template<typename T>
void prep_convex_poly(vector<TPoint<T>>& pts){
  rotate(pts.begin(), min_element(all(pts)), pts.end());
}
// 0 - Outside, 1 - Exclusively Inside, 2 - On the Border
template<typename T>
int in_convex_poly(TPoint<T>& p, vector<TPoint<T>>& pts){
  int n = sz(pts);
  if (!n) return 0;
  if (n <= 2) return is_on_seg(p, pts[0], pts.back());
  int l = 1, r = n - 1;
  while (r - l > 1){
    int mid = (l + r) / 2;
    if (acw(pts[mid] - pts[0], p - pts[0])) l = mid;
    else r = mid;
  }
  if (!in_triangle(p, pts[0], pts[l], pts[l + 1])) return 0;
  if (is_on_seg(p, pts[l], pts[l + 1]) ||
    is_on_seg(p, pts[0], pts.back()) ||
    is_on_seg(p, pts[0], pts[1])
  ) return 2;
  return 1;
}
// 0 - Outside, 1 - Exclusively Inside, 2 - On the Border
template<typename T>
int in_simple_poly(TPoint<T> p, vector<TPoint<T>>& pts){
  int n = sz(pts);
  bool res = 0;
  for (int i = 0; i < n; i++){
    auto a = pts[i], b = pts[(i + 1) % n];
    if (is_on_seg(p, a, b)) return 2;
    if (((a.y > p.y) - (b.y > p.y)) * vmul(b - p, a - p) > TPoint<T>::eps){
      res ^= 1;
    }
  }
  return res;
}
template<typename T>
void minkowski_rotate(vector<TPoint<T>>& P){
  int pos = 0;
  for (int i = 1; i < sz(P); i++){
    if (abs(P[i].y - P[pos].y) <= TPoint<T>::eps){
      if (P[i].x < P[pos].x) pos = i;
    }
    else if (P[i].y < P[pos].y) pos = i;
  }
  rotate(P.begin(), P.begin() + pos, P.end());
}
// P and Q are strictly convex, points given in counterclockwise order
template<typename T>
vector<TPoint<T>> minkowski_sum(vector<TPoint<T>> P, vector<TPoint<T>> Q){
  minkowski_rotate(P);
  minkowski_rotate(Q);
  P.pb(P[0]);
  Q.pb(Q[0]);
  vector<TPoint<T>> ans;
  int i = 0, j = 0;
  while (i < sz(P) - 1 || j < sz(Q) - 1){
    ans.pb(P[i] + Q[j]);
    T curmul;
    if (i == sz(P) - 1) curmul = -1;
    else if (j == sz(Q) - 1) curmul = +1;
    else curmul = vmul(P[i + 1] - P[i], Q[j + 1] - Q[j]);
    if (abs(curmul) < TPoint<T>::eps || curmul > 0) i++;
    if (abs(curmul) < TPoint<T>::eps || curmul < 0) j++;
  }
  return ans;
}
using Point = TPoint<ll>; using Line = TLine<ll>; using Ray = TRay<ll>; const ld PI = acos(-1);
```

Geometry

Basic stuff

```
using ll = long long;
using ld = long double;

constexpr auto eps = 1e-8;
const auto PI = acos(-1);
int sgn(ld x) { return (abs(x) <= eps) ? 0 : (x < 0 ? -1 : 1); }

struct Point {
  ld x = 0, y = 0;
  Point() = default;
  Point(ld _x, ld _y) : x(_x), y(_y) {}
  bool operator<(const Point &p) const { return !sgn(p.x - x) ? sgn(y - p.y) < 0 : x < p.x; }
  bool operator==(const Point &p) const { return !sgn(p.x - x) && !sgn(p.y - y); }
  Point operator+(const Point &p) const { return {x + p.x, y + p.y}; }
  Point operator-(const Point &p) const { return {x - p.x, y - p.y}; }
  Point operator*(ld a) const { return {x * a, y * a}; }
  Point operator/(ld a) const { return {x / a, y / a}; }
  auto operator*(const Point &p) const { return x * p.x + y * p.y; }  // dot
  auto operator^(const Point &p) const { return x * p.y - y * p.x; }  // cross
  friend auto &operator>>(istream &i, Point &p) { return i >> p.x >> p.y; }
  friend auto &operator<<(ostream &o, Point p) { return o << p.x << ' ' << p.y; }
};

struct Line {
  Point s = {0, 0}, e = {0, 0};
  Line() = default;
  Line(Point _s, Point _e) : s(_s), e(_e) {}
  friend auto &operator>>(istream &i, Line &l) { return i >> l.s >> l.e; }  // ((x1, y1), (x2, y2)
};

struct Segment : Line {
  using Line::Line;
};

struct Circle {
  Point o = {0, 0};
  ld r = 0;
  Circle() = default;
  Circle(Point _o, ld _r) : o(_o), r(_r) {}
};
```

```
auto dist2(const Point &a) { return a * a; }
auto dist2(const Point &a, const Point &b) { return dist2(a - b); }
```

```cpp
auto dist(const Point &a) { return sqrt(dist2(a)); }
auto dist(const Point &a, const Point &b) { return sqrt(dist2(a - b)); }
auto dist(const Point &a, const Line &l) { return abs((a - l.s) ^ (l.e - l.s)) / dist(l.s, l.e); }
auto dist(const Point &p, const Segment &l) {
  if (l.s == l.e) return dist(p, l.s);
  auto d = dist2(l.s, l.e), t = min(d, max((ld)0, (p - l.s) * (l.e - l.s)));
  return dist((p - l.s) * d, (l.e - l.s) * t) / d;
}
/* Needs is_intersect
auto dist(const Segment &l1, const Segment &l2) {
  if (is_intersect(l1, l2)) return (ld)0;
  return min({dist(l1.s, l2), dist(l1.e, l2), dist(l2.s, l1), dist(l2.e, l1)});
} */

Point perp(const Point &p) { return Point(-p.y, p.x); }

auto rad(const Point &p) { return atan2(p.y, p.x); }
```

Transformation

```cpp
Point project(const Point &p, const Line &l) {
  return l.s + ((l.e - l.s) * ((l.e - l.s) * (p - l.s))) / dist2(l.e - l.s);
}

Point reflect(const Point &p, const Line &l) {
  return project(p, l) * 2 - p;
}

Point dilate(const Point &p, ld scale_x = 1, ld scale_y = 1) { return Point(p.x * scale_x, p.y * scale_y); }
Line dilate(const Line &l, ld scale_x = 1, ld scale_y = 1) { return Line(dilate(l.s, scale_x, scale_y), dilate(l.e, scale_x, scale_y)); }
Segment dilate(const Segment &l, ld scale_x = 1, ld scale_y = 1) { return Segment(dilate(l.s, scale_x, scale_y), dilate(l.e, scale_x, scale_y)); }
vector<Point> dilate(const vector<Point> &p, ld scale_x = 1, ld scale_y = 1) {
  int n = p.size();
  vector<Point> res(n);
  for (int i = 0; i < n; i++)
    res[i] = dilate(p[i], scale_x, scale_y);
  return res;
}

Point rotate(const Point &p, ld a) { return Point(p.x * cos(a) - p.y * sin(a), p.x * sin(a) + p.y * cos(a)); }
Line rotate(const Line &l, ld a) { return Line(rotate(l.s, a), rotate(l.e, a)); }
Segment rotate(const Segment &l, ld a) { return Segment(rotate(l.s, a), rotate(l.e, a)); }
Circle rotate(const Circle &c, ld a) { return Circle(rotate(c.o, a), c.r); }
vector<Point> rotate(const vector<Point> &p, ld a) {
  int n = p.size();
  vector<Point> res(n);
  for (int i = 0; i < n; i++)
    res[i] = rotate(p[i], a);
  return res;
}

Point translate(const Point &p, ld dx = 0, ld dy = 0) { return Point(p.x + dx, p.y + dy); }
Line translate(const Line &l, ld dx = 0, ld dy = 0) { return Line(translate(l.s, dx, dy), translate(l.e, dx, dy)); }
Segment translate(const Segment &l, ld dx = 0, ld dy = 0) { return Segment(translate(l.s, dx, dy), translate(l.e, dx, dy)); }
Circle translate(const Circle &c, ld dx = 0, ld dy = 0) { return Circle(translate(c.o, dx, dy), c.r); }
vector<Point> translate(const vector<Point> &p, ld dx = 0, ld dy = 0) {
  int n = p.size();
  vector<Point> res(n);
  for (int i = 0; i < n; i++)
    res[i] = translate(p[i], dx, dy);
  return res;
}
```

Relation

```cpp
enum class Relation { SEPARATE, EX_TOUCH, OVERLAP, IN_TOUCH, INSIDE };
Relation get_relation(const Circle &a, const Circle &b) {
  auto c1c2 = dist(a.o, b.o);
  auto r1r2 = a.r + b.r, diff = abs(a.r - b.r);
  if (sgn(c1c2 - r1r2) > 0) return Relation::SEPARATE;
  if (sgn(c1c2 - r1r2) == 0) return Relation::EX_TOUCH;
  if (sgn(c1c2 - diff) > 0) return Relation::OVERLAP;
  if (sgn(c1c2 - diff) == 0) return Relation::IN_TOUCH;
  return Relation::INSIDE;
}

auto get_cos_from_triangle(ld a, ld b, ld c) { return (a * a + b * b - c * c) / (2.0 * a * b); }

bool on_line(const Line &l, const Point &p) { return !sgn((l.s - p) ^ (l.e - p)); }

bool on_segment(const Segment &l, const Point &p) {
  return !sgn((l.s - p) ^ (l.e - p)) && sgn((l.s - p) * (l.e - p)) <= 0;
}

bool on_segment2(const Segment &l, const Point &p) { // assume p on Line l
  if (l.s == p || l.e == p) return true;
  if (min(l.s, l.e) < p && p < max(l.s, l.e)) return true;
  return false;
}

bool is_parallel(const Line &a, const Line &b) { return !sgn((a.s - a.e) ^ (b.s - b.e)); }
bool is_orthogonal(const Line &a, const Line &b) { return !sgn((a.s - a.e) * (b.s - b.e)); }

int is_intersect(const Segment &a, const Segment &b) {
  auto d1 = sgn((a.e - a.s) ^ (b.s - a.s)), d2 = sgn((a.e - a.s) ^ (b.e - a.s));
  auto d3 = sgn((b.e - b.s) ^ (a.s - b.s)), d4 = sgn((b.e - b.s) ^ (a.e - b.s));
  if (d1 * d2 < 0 && d3 * d4 < 0) return 2;  // intersect at non-end point
  return (d1 == 0 && sgn((b.s - a.s) * (b.s - a.e)) <= 0) ||
         (d2 == 0 && sgn((b.e - a.s) * (b.e - a.e)) <= 0) ||
         (d3 == 0 && sgn((a.s - b.s) * (a.s - b.e)) <= 0) ||
         (d4 == 0 && sgn((a.e - b.s) * (a.e - b.e)) <= 0);
}

int is_intersect(const Line &a, const Segment &b) {
  auto d1 = sgn((a.e - a.s) ^ (b.s - a.s)), d2 = sgn((a.e - a.s) ^ (b.e - a.s));
  if (d1 * d2 < 0) return 2;  // intersect at non-end point
  return d1 == 0 || d2 == 0;
}

Point intersect(const Line &a, const Line &b) {
  auto u = a.e - a.s, v = b.e - b.s;
  auto t = ((b.s - a.s) ^ v) / (u ^ v);
  return a.s + u * t;
}

int is_intersect(const Circle &c, const Line &l) {
  auto d = dist(c.o, l);
  return sgn(d - c.r) < 0 ? 2 : !sgn(d - c.r);
}

vector<Point> intersect(const Circle &a, const Circle &b) {
  auto relation = get_relation(a, b);
  if (relation == Relation::INSIDE || relation == Relation::SEPARATE) return {};
  auto vec = b.o - a.o;
  auto d2 = dist2(vec);
  auto p = (d2 + a.r * a.r - b.r * b.r) / ((long double)2 * d2), h2 = a.r * a.r - p * p * d2;
  auto mid = a.o + vec * p, per = perp(vec) * sqrt(max((long double)0, h2) / d2);
  if (relation == Relation::OVERLAP)
    return {mid + per, mid - per};
  else
    return {mid};
}

vector<Point> intersect(const Circle &c, const Line &l) {
  if (!is_intersect(c, l)) return {};
  auto v = l.e - l.s, t = v / dist(v);
  Point a = l.s + t * ((c.o - l.s) * t);
  auto d = sqrt(max((ld)0, c.r * c.r - dist2(c.o, a)));
  if (!sgn(d)) return {a};
  return {a - t * d, a + t * d};
}

int in_poly(const vector<Point> &p, const Point &a) {
  int cnt = 0, n = (int)p.size();
  for (int i = 0; i < n; i++) {
    auto q = p[(i + 1) % n];
    if (on_segment(Segment(p[i], q), a)) return 1;  // on the edge of the polygon
    cnt ^= ((a.y < p[i].y) - (a.y < q.y)) * ((p[i] - a) ^ (q - a)) > 0;
  }
  return cnt ? 2 : 0;
}

int is_intersect(const vector<Point> &p, const Line &a) {
  // 1: touching, >=2: intersect count
  int cnt = 0, edge_cnt = 0, n = (int)p.size();
  for (int i = 0; i < n; i++) {
    auto q = p[(i + 1) % n];
    if (on_line(a, p[i]) && on_line(a, q)) return -1;  // infinity
    auto t = is_intersect(a, Segment(p[i], q));
    (t == 1) && edge_cnt++, (t == 2) && cnt++;
  }
  return cnt + edge_cnt / 2;
}

vector<Point> tangent(const Circle &c, const Point &p) {
  auto d = dist(c.o, p), l = c.r * c.r / d, h = sqrt(c.r * c.r - l * l);
  auto v = (p - c.o) / d;
  return {c.o + v * l + perp(v) * h, c.o + v * l - perp(v) * h};
}

Circle get_circumscribed(const Point &a, const Point &b, const Point &c) {
  Line u((a + b) / 2, ((a + b) / 2) + perp(b - a));
  Line v((b + c) / 2, ((b + c) / 2) + perp(c - b));
  auto o = intersect(u, v);
  return Circle(o, dist(o, a));
}

Circle get_inscribed(const Point &a, const Point &b, const Point &c) {
  auto l1 = dist(b - c), l2 = dist(c - a), l3 = dist(a - b);
  Point o = (a * l1 + b * l2 + c * l3) / (l1 + l2 + l3);
  return Circle(o, dist(o, Line(a, b)));
}

pair<ld, ld> get_centroid(const vector<Point> &p) {
  int n = (int)p.size();
  ld x = 0, y = 0, sum = 0;
  auto a = p[0], b = p[1];
  for (int i = 2; i < n; i++) {
    auto c = p[i];
    auto s = area({a, b, c});
    sum += s;
    x += s * (a.x + b.x + c.x);
    y += s * (a.y + b.y + c.y);
    swap(b, c);
  }
  return {x / (3 * sum), y / (3 * sum)};
}
```

Area

```
auto area(const vector<Point> &p) {
  int n = (int)p.size();
  long double area = 0;
  for (int i = 0; i < n; i++) area += p[i] ^ p[(i + 1) % n];
  return area / 2.0;
}

auto area(const Point &a, const Point &b, const Point &c) {
  return ((long double)((b - a) ^ (c - a))) / 2.0;
}

auto area2(const Point &a, const Point &b, const Point &c) { return (b - a) ^ (c - a); }

auto area_intersect(const Circle &c, const vector<Point> &ps) {
  int n = (int)ps.size();
  auto arg = [&](const Point &p, const Point &q) { return atan2(p ^ q, p * q); };
  auto tri = [&](const Point &p, const Point &q) {
    auto r2 = c.r * c.r / (long double)2;
    auto d = q - p;
    auto a = d * p / dist2(d), b = (dist2(p) - c.r * c.r) / dist2(d);
    long double det = a * a - b;
    if (sgn(det) <= 0) return arg(p, q) * r2;
    auto s = max((long double)0, -a - sqrt(det)), t = min((long double)1, -a + sqrt(det));
    if (sgn(t) < 0 || sgn(1 - s) <= 0) return arg(p, q) * r2;
    auto u = p + d * s, v = p + d * t;
    return arg(p, u) * r2 + (u ^ v) / 2 + arg(v, q) * r2;
  };
  long double sum = 0;
  for (int i = 0; i < n; i++) sum += tri(ps[i] - c.o, ps[(i + 1) % n] - c.o);
  return sum;
}

auto adaptive_simpson(ld _l, ld _r, function<ld(ld)> f) {
  auto simpson = [&](ld l, ld r) { return (r - l) * (f(l) + 4 * f((l + r) / 2) + f(r)) / 6; };
  function<ld(ld, ld, ld)> asr = [&](ld l, ld r, ld s) {
    auto mid = (l + r) / 2;
    auto left = simpson(l, mid), right = simpson(mid, r);
    if (!sgn(left + right - s)) return left + right;
    return asr(l, mid, left) + asr(mid, r, right);
  };
  return asr(_l, _r, simpson(_l, _r));
}

vector<Point> half_plane_intersect(vector<Line> &L) {
  int n = (int)L.size(), l = 0, r = 0;  // [left, right]
  sort(L.begin(), L.end(),
       [](const Line &a, const Line &b) { return rad(a.s - a.e) < rad(b.s - b.e); });
  vector<Point> p(n), res;
  vector<Line> q(n);
  q[0] = L[0];
  for (int i = 1; i < n; i++) {
    while (l < r && sgn((L[i].e - L[i].s) ^ (p[r - 1] - L[i].s)) <= 0) r--;
    while (l < r && sgn((L[i].e - L[i].s) ^ (p[l] - L[i].s)) <= 0) l++;
    q[++r] = L[i];
    if (sgn((q[r].e - q[r].s) ^ (q[r - 1].e - q[r - 1].s)) == 0) {
```

```cpp
      r--;
      if (sgn((q[r].e - q[r].s) ^ (L[i].s - q[r].s)) > 0) q[r] = L[i];
    }
    if (l < r) p[r - 1] = intersect(q[r - 1], q[r]);
  }
  while (l < r && sgn((q[l].e - q[l].s) ^ (p[r - 1] - q[l].s)) <= 0) r--;
  if (r - l <= 1) return {};
  p[r] = intersect(q[r], q[l]);
  return vector<Point>(p.begin() + l, p.begin() + r + 1);
}
```

Convex

```cpp
vector<Point> get_convex(vector<Point> &points, bool allow_collinear = false) {
  // strict, no repeat, two pass
  sort(points.begin(), points.end());
  points.erase(unique(points.begin(), points.end()), points.end());
  vector<Point> L, U;
  for (auto &t : points) {
    for (ll sz = L.size(); sz > 1 && (sgn((t - L[sz - 2]) ^ (L[sz - 1] - L[sz - 2])) >= 0);
         L.pop_back(), sz = L.size()) {
    }
    L.push_back(t);
  }
  for (auto &t : points) {
    for (ll sz = U.size(); sz > 1 && (sgn((t - U[sz - 2]) ^ (U[sz - 1] - U[sz - 2])) <= 0);
         U.pop_back(), sz = U.size()) {
    }
    U.push_back(t);
  }
  // contain repeats if all collinear, use a set to remove repeats
  if (allow_collinear) {
    for (int i = (int)U.size() - 2; i >= 1; i--) L.push_back(U[i]);
  } else {
    set<Point> st(L.begin(), L.end());
    for (int i = (int)U.size() - 2; i >= 1; i--) {
      if (st.count(U[i]) == 0) L.push_back(U[i]), st.insert(U[i]);
    }
  }
  return L;
}

vector<Point> get_convex2(vector<Point> &points, bool allow_collinear = false) {  // strict, no repeat, one pass
  nth_element(points.begin(), points.begin(), points.end());
  sort(points.begin() + 1, points.end(), [&](const Point &a, const Point &b) {
    int rad_diff = sgn((a - points[0]) ^ (b - points[0]));
    return !rad_diff ? (dist2(a - points[0]) < dist2(b - points[0])) : (rad_diff > 0);
  });
  if (allow_collinear) {
    int i = (int)points.size() - 1;
    while (i >= 0 && !sgn((points[i] - points[0]) ^ (points[i] - points.back()))) i--;
    reverse(points.begin() + i + 1, points.end());
  }
  vector<Point> hull;
  for (auto &t : points) {
    for (ll sz = hull.size();
         sz > 1 && (sgn((t - hull[sz - 2]) ^ (hull[sz - 1] - hull[sz - 2])) >= allow_collinear);
         hull.pop_back(), sz = hull.size()) {
    }
    hull.push_back(t);
  }
  return hull;
}

vector<Point> get_convex_safe(vector<Point> points, bool allow_collinear = false) {
  return get_convex(points, allow_collinear);
}

vector<Point> get_convex2_safe(vector<Point> points, bool allow_collinear = false) {
  return get_convex2(points, allow_collinear);
}

bool is_convex(const vector<Point> &p, bool allow_collinear = false) {
  int n = p.size();
  int lo = 1, hi = -1;
  for (int i = 0; i < n; i++) {
    int cur = sgn((p[(i + 2) % n] - p[(i + 1) % n]) ^ (p[(i + 1) % n] - p[i]));
    lo = min(lo, cur); hi = max(hi, cur);
  }
  return allow_collinear ? (hi - lo) < 2 : (lo == hi && lo);
}

auto rotating_calipers(const vector<Point> &hull) {
  // use get_convex2
  int n = (int)hull.size();  // return the square of longest dist
  assert(n > 1);
  if (n <= 2) return dist2(hull[0], hull[1]);
  ld res = 0;
  for (int i = 0, j = 2; i < n; i++) {
    auto d = hull[i], e = hull[(i + 1) % n];
    while (area2(d, e, hull[j]) < area2(d, e, hull[(j + 1) % n])) j = (j + 1) % n;
    res = max(res, max(dist2(d, hull[j]), dist2(e, hull[j])));
  }
  return res;
}

// Find polygon cut to the left of l
vector<Point> convex_cut(const vector<Point> &p, const Line &l) {
  int n = p.size();
  vector<Point> cut;
  for (int i = 0; i < n; i++) {
    auto a = p[i], b = p[(i + 1) % n];
    if (sgn((l.e - l.s) ^ (a - l.s)) >= 0)
      cut.push_back(a);
    if (sgn((l.e - l.s) ^ (a - l.s)) * sgn((l.e - l.s) ^ (b - l.s)) == -1)
      cut.push_back(intersect(Line(a, b), l));
  }
  return cut;
}

// Sort by angle in range [0, 2pi)
template <class RandomIt>
void polar_sort(RandomIt first, RandomIt last, Point origin = Point(0, 0)) {
  auto get_quad = [&](const Point& p) {
    Point diff = p - origin;
    if (diff.x > 0 && diff.y >= 0) return 1;
    if (diff.x <= 0 && diff.y > 0) return 2;
    if (diff.x < 0 && diff.y <= 0) return 3;
    return 4;
  };
  auto polar_cmp = [&](const Point& p1, const Point& p2) {
    int q1 = get_quad(p1), q2 = get_quad(p2);
    if (q1 != q2) return q1 < q2;
    return ((p1 - origin) ^ (p2 - origin)) > 0;
  };
  sort(first, last, polar_cmp);
}
```

Basic 3D

```cpp
using ll = long long;
using ld = long double;

constexpr auto eps = 1e-8;
const auto PI = acos(-1);
int sgn(ld x) { return (abs(x) <= eps) ? 0 : (x < 0 ? -1 : 1); }

struct Point3D {
  ld x = 0, y = 0, z = 0;
  Point3D() = default;
  Point3D(ld _x, ld _y, ld _z) : x(_x), y(_y), z(_z) {}
  bool operator<(const Point3D &p) const { return !sgn(p.x - x) ? (!sgn(p.y - y) ? sgn(p.z - z) < 0 : y < p.y) : x < p.x; }
  bool operator==(const Point3D &p) const { return !sgn(p.x - x) && !sgn(p.y - y) && !sgn(p.z - z); }
  Point3D operator+(const Point3D &p) const { return {x + p.x, y + p.y, z + p.z}; }
  Point3D operator-(const Point3D &p) const { return {x - p.x, y - p.y, z - p.z}; }
  Point3D operator*(ld a) const { return {x * a, y * a, z * a}; }
  Point3D operator/(ld a) const { return {x / a, y / a, z / a}; }
  auto operator*(const Point3D &p) const { return x * p.x + y * p.y + z * p.z; }  // dot
  Point3D operator^(const Point3D &p) const { return {y * p.z - z * p.y, z * p.x - x * p.z, x * p.y - y * p.x}; }  // cross
  friend auto &operator>>(istream &i, Point3D &p) { return i >> p.x >> p.y >> p.z; }
};

struct Line3D {
  Point3D s = {0, 0, 0}, e = {0, 0, 0};
  Line3D() = default;
  Line3D(Point3D _s, Point3D _e) : s(_s), e(_e) {}
};

struct Segment3D : Line3D {
  using Line3D::Line3D;
};

auto dist2(const Point3D &a) { return a * a; }
auto dist2(const Point3D &a, const Point3D &b) { return dist2(a - b); }
auto dist(const Point3D &a) { return sqrt(dist2(a)); }
auto dist(const Point3D &a, const Point3D &b) { return sqrt(dist2(a - b)); }
auto dist(const Point3D &a, const Line3D &l) { return dist((a - l.s) ^ (l.e - l.s)) / dist(l.s, l.e); }
auto dist(const Point3D &p, const Segment3D &l) {
  if (l.s == l.e) return dist(p, l.s);
  auto d = dist2(l.s, l.e), t = min(d, max((ld)0, (p - l.s) * (l.e - l.s)));
  return dist((p - l.s) * d, (l.e - l.s) * t) / d;
}
```

Miscellaneous

```cpp
tuple<int,int,ld> closest_pair(vector<Point> &p) {
  using Pt = pair<Point,int>;
  int n = p.size();
  assert(n > 1);
  vector<Pt> pts(n), buf;
  for (int i = 0; i < n; i++) pts[i] = {p[i], i};
  sort(pts.begin(), pts.end());
  buf.reserve(n);
  auto cmp_y = [](const Pt& p1, const Pt& p2) { return p1.first.y < p2.first.y; };
  function<tuple<int,int,ld>(int, int)> recurse = [&](int l, int r) -> tuple<int,int,ld> {
    int i = pts[l].second, j = pts[l + 1].second;
    ld d = dist(pts[l].first, pts[l + 1].first);
    if (r - l < 5) {
      for (int a = l; a < r; a++) for (int b = a + 1; b < r; b++) {
        ld cur = dist(pts[a].first, pts[b].first);
        if (cur < d) { i = pts[a].second; j = pts[b].second; d = cur; }
      }
      sort(pts.begin() + l, pts.begin() + r, cmp_y);
    }
    else {
      int mid = (l + r)/2;
      ld x = pts[mid].first.x;
      auto [li, lj, ldist] = recurse(l, mid);
      auto [ri, rj, rdist] = recurse(mid, r);
      if (ldist < rdist) { i = li; j = lj; d = ldist; }
      else { i = ri; j = rj; d = rdist; }
      inplace_merge(pts.begin() + l, pts.begin() + mid, pts.begin() + r, cmp_y);
      buf.clear();
      for (int a = l; a < r; a++) {
        if (abs(x - pts[a].first.x) >= d) continue;
        for (int b = buf.size() - 1; b >= 0; b--) {
          if (pts[a].first.y - buf[b].first.y >= d) break;
          ld cur = dist(pts[a].first, buf[b].first);
          if (cur < d) { i = pts[a].second; j = buf[b].second; d = cur; }
        }
        buf.push_back(pts[a]);
      }
    }
    return {i, j, d};
  };
  return recurse(0, n);
}

Line abc_to_line(ld a, ld b, ld c) {
  assert(!sgn(a) || !sgn(b));
  if(a == 0) return Line(Point(0, -c/b), Point(1, -c/b));
  if(b == 0) return Line(Point(-c/a, 0), Point(-c/a, 1));
  Point s(0, -c/b), e(1, (-c - a)/b), diff = e - s;
  return Line(s, s + diff/dist(diff));
}

tuple<ld,ld,ld> line_to_abc(const Line& l) {
  Point diff = l.e - l.s;
  return {-diff.y, diff.x, -(diff ^ l.s)};
}
```

Graph Theory

Max Flow

```cpp
struct Edge {
  int from, to, cap, remain;
};

struct Dinic {
  int n;
  vector<Edge> e;
  vector<vector<int>> g;
  vector<int> d, cur;
  Dinic(int _n) : n(_n), g(n), d(n), cur(n) {}
  void add_edge(int u, int v, int c) {
    g[u].push_back((int)e.size());
    e.push_back({u, v, c, c});
    g[v].push_back((int)e.size());
    e.push_back({v, u, 0, 0});
  }
  ll max_flow(int s, int t) {
    int inf = 1e9;
    auto bfs = [&]() {
      fill(d.begin(), d.end(), inf), fill(cur.begin(), cur.end(), 0);
      d[s] = 0;
      vector<int> q{s}, nq;
      for (int step = 1; q.size(); swap(q, nq), nq.clear(), step++) {
        for (auto& node : q) {
          for (auto& edge : g[node]) {
            int ne = e[edge].to;
            if (!e[edge].remain || d[ne] <= step) continue;
            d[ne] = step, nq.push_back(ne);
            if (ne == t) return true;
          }
        }
      }
      return false;
    };
    function<int(int, int)> find = [&](int node, int limit) {
      if (node == t || !limit) return limit;
      int flow = 0;
      for (int i = cur[node]; i < g[node].size(); i++) {
        cur[node] = i;
        int edge = g[node][i], oe = edge ^ 1, ne = e[edge].to;
        if (!e[edge].remain || d[ne] != d[node] + 1) continue;
        if (int temp = find(ne, min(limit - flow, e[edge].remain))) {
          e[edge].remain -= temp, e[oe].remain += temp, flow += temp;
        } else {
          d[ne] = -1;
        }
        if (flow == limit) break;
      }
      return flow;
    };
    ll res = 0;
    while (bfs())
      while (int flow = find(s, inf)) res += flow;
    return res;
  }
};
```

- USAGE

```cpp
int main() {
  int n, m, s, t;
  cin >> n >> m >> s >> t;
  Dinic dinic(n);
  for (int i = 0, u, v, c; i < m; i++) {
    cin >> u >> v >> c;
    dinic.add_edge(u - 1, v - 1, c);
  }
  cout << dinic.max_flow(s - 1, t - 1) << '\n';
}
```

PushRelabel Max-Flow (faster)

```cpp
// https://github.com/kth-competitive-programming/kactl/blob/main/content/graph/PushRelabel.h
#define rep(i, a, b) for (int i = a; i < (b); ++i)
#define all(x) begin(x), end(x)
#define sz(x) (int)(x).size()
typedef long long ll;
typedef pair<int, int> pii;
typedef vector<int> vi;

struct PushRelabel {
  struct Edge {
    int dest, back;
    ll f, c;
  };
  vector<vector<Edge>> g;
  vector<ll> ec;
  vector<Edge*> cur;
  vector<vi> hs;
  vi H;
  PushRelabel(int n) : g(n), ec(n), cur(n), hs(2 * n), H(n) {}

  void addEdge(int s, int t, ll cap, ll rcap = 0) {
    if (s == t) return;
    g[s].push_back({t, sz(g[t]), 0, cap});
    g[t].push_back({s, sz(g[s]) - 1, 0, rcap});
  }

  void addFlow(Edge& e, ll f) {
    Edge& back = g[e.dest][e.back];
    if (!ec[e.dest] && f) hs[H[e.dest]].push_back(e.dest);
    e.f += f;
    e.c -= f;
    ec[e.dest] += f;
    back.f -= f;
    back.c += f;
    ec[back.dest] -= f;
  }
  ll calc(int s, int t) {
    int v = sz(g);
    H[s] = v;
    ec[t] = 1;
    vi co(2 * v);
    co[0] = v - 1;
    rep(i, 0, v) cur[i] = g[i].data();
    for (Edge& e : g[s]) addFlow(e, e.c);

    for (int hi = 0;;) {
      while (hs[hi].empty())
        if (!hi--) return -ec[s];
      int u = hs[hi].back();
      hs[hi].pop_back();
      while (ec[u] > 0)  // discharge u
        if (cur[u] == g[u].data() + sz(g[u])) {
          H[u] = 1e9;
          for (Edge& e : g[u])
            if (e.c && H[u] > H[e.dest] + 1) H[u] = H[e.dest] + 1, cur[u] = &e;
          if (++co[H[u]], !--co[hi] && hi < v)
            rep(i, 0, v) if (hi < H[i] && H[i] < v)-- co[H[i]], H[i] = v + 1;
          hi = H[u];
        } else if (cur[u]->c && H[u] == H[cur[u]->dest] + 1)
          addFlow(*cur[u], min(ec[u], cur[u]->c));
        else
          ++cur[u];
    }
  }
  bool leftOfMinCut(int a) { return H[a] >= sz(g); }
};
```

Min-Cost Max-Flow

```cpp
class MCMF {
public:
  static constexpr int INF = 1e9;
  const int n;
  vector<tuple<int, int, int>> e;
  vector<vector<int>> g;
  vector<int> h, dis, pre;
  bool dijkstra(int s, int t) {
    dis.assign(n, INF);
    pre.assign(n, -1);
    priority_queue<pair<int, int>, vector<pair<int, int>>, greater<>> que;
    dis[s] = 0;
    que.emplace(0, s);
    while (!que.empty()) {
      auto [d, u] = que.top();
      que.pop();
      if (dis[u] != d) continue;
      for (int i : g[u]) {
        auto [v, f, c] = e[i];
        if (c > 0 && dis[v] > d + h[u] - h[v] + f) {
          dis[v] = d + h[u] - h[v] + f;
          pre[v] = i;
          que.emplace(dis[v], v);
        }
      }
    }
    return dis[t] != INF;
  }
  MCMF(int n) : n(n), g(n) {}
  void add_edge(int u, int v, int fee, int c) {
    g[u].push_back(e.size());
    e.emplace_back(v, fee, c);
    g[v].push_back(e.size());
    e.emplace_back(u, -fee, 0);
  }
  pair<ll, ll> max_flow(const int s, const int t) {
    int flow = 0, cost = 0;
    h.assign(n, 0);
    while (dijkstra(s, t)) {
      for (int i = 0; i < n; ++i) h[i] += dis[i];
      for (int i = t; i != s; i = get<0>(e[pre[i] ^ 1])) {
        --get<2>(e[pre[i]]);
        ++get<2>(e[pre[i] ^ 1]);
      }
      ++flow;
      cost += h[t];
    }
    return {flow, cost};
  }
};
```

Max Cost Feasible Flow

```cpp
struct Edge {
  int from, to, cap, remain, cost;
};

struct MCMF {
  int n;
  vector<Edge> e;
  vector<vector<int>> g;
  vector<ll> d, pre;
  MCMF(int _n) : n(_n), g(n), d(n), pre(n) {}
  void add_edge(int u, int v, int c, int w) {
    g[u].push_back((int)e.size());
    e.push_back({u, v, c, c, w});
    g[v].push_back((int)e.size());
    e.push_back({v, u, 0, 0, -w});
  }
  pair<ll, ll> max_flow(int s, int t) {
    ll inf = 1e18;
    auto spfa = [&]() {
      fill(d.begin(), d.end(), -inf); // important!
      vector<int> f(n), seen(n);
      d[s] = 0, f[s] = 1e9;
      vector<int> q{s}, nq;
      for (; q.size(); swap(q, nq), nq.clear()) {
        for (auto& node : q) {
          seen[node] = false;
          for (auto& edge : g[node]) {
            int ne = e[edge].to, cost = e[edge].cost;
            if (!e[edge].remain || d[ne] >= d[node] + cost) continue;
            d[ne] = d[node] + cost, pre[ne] = edge;
            f[ne] = min(e[edge].remain, f[node]);
            if (!seen[ne]) seen[ne] = true, nq.push_back(ne);
          }
        }
      }
      return f[t];
    };
    ll flow = 0, cost = 0;
    while (int temp = spfa()) {
      if (d[t] < 0) break; // important!
      flow += temp, cost += temp * d[t];
      for (ll i = t; i != s; i = e[pre[i]].from) {
        e[pre[i]].remain -= temp, e[pre[i] ^ 1].remain += temp;
      }
    }
    return {flow, cost};
  }
};
```

Heavy-Light Decomposition

```cpp
struct HeavyLight {
  int root = 0, n = 0;
  std::vector<int> parent, deep, hson, top, sz, dfn;
  HeavyLight(std::vector<std::vector<int>> &g, int _root)
      : root(_root), n(int(g.size())), parent(n), deep(n), hson(n, -1), top(n), sz(n), dfn(n, -1) {
    int cur = 0;
    std::function<int(int, int, int)> dfs = [&](int node, int fa, int dep) {
      deep[node] = dep, sz[node] = 1, parent[node] = fa;
      for (auto &ne : g[node]) {
        if (ne == fa) continue;
        sz[node] += dfs(ne, node, dep + 1);
        if (hson[node] == -1 || sz[ne] > sz[hson[node]]) hson[node] = ne;
      }
      return sz[node];
    };
    std::function<void(int, int)> dfs2 = [&](int node, int t) {
      top[node] = t, dfn[node] = cur++;
      if (hson[node] == -1) return;
      dfs2(hson[node], t);
      for (auto &ne : g[node]) {
        if (ne == parent[node] || ne == hson[node]) continue;
        dfs2(ne, ne);
      }
    };
    dfs(root, -1, 0), dfs2(root, root);
  }

  int lca(int x, int y) const {
    while (top[x] != top[y]) {
      if (deep[top[x]] < deep[top[y]]) swap(x, y);
      x = parent[top[x]];
    }
    return deep[x] < deep[y] ? x : y;
  }

  std::vector<std::array<int, 2>> get_dfn_path(int x, int y) const {
    std::array<std::vector<std::array<int, 2>>, 2> path;
    bool front = true;
    while (top[x] != top[y]) {
      if (deep[top[x]] > deep[top[y]]) swap(x, y), front = !front;
      path[front].push_back({dfn[top[y]], dfn[y] + 1});
      y = parent[top[y]];
    }
    if (deep[x] > deep[y]) swap(x, y), front = !front;

    path[front].push_back({dfn[x], dfn[y] + 1});
    std::reverse(path[1].begin(), path[1].end());
    for (const auto &[left, right] : path[1]) path[0].push_back({right, left});
    return path[0];
  }

  Node query_seg(int u, int v, const SegTree &seg) const {
    auto node = Node();
    for (const auto &[left, right] : get_dfn_path(u, v)) {
      if (left > right) {
        node = pull(node, rev(seg.query(right, left)));
      } else {
        node = pull(node, seg.query(left, right));
      }
    }
    return node;
  }
};
```

- USAGE:

```cpp
vector<ll> light(n);
SegTree heavy(n), form_parent(n);
// cin >> x >> y, x--, y--;
```

```cpp
int z = lca(x, y);
while (x != z) {
  if (dfn[top[x]] <= dfn[top[z]]) {
    // [dfn[z], dfn[x]), from heavy
    heavy.modify(dfn[z], dfn[x], 1);
    break;
  }
  // x -> top[x];
  heavy.modify(dfn[top[x]], dfn[x], 1);
  light[parent[top[x]]] += a[top[x]];
  x = parent[top[x]];
}
while (y != z) {
  if (dfn[top[y]] <= dfn[top[z]]) {
    // (dfn[z], dfn[y]], from heavy
    form_parent.modify(dfn[z] + 1, dfn[y] + 1, 1);
    break;
  }
  // y -> top[y];
  form_parent.modify(dfn[top[y]], dfn[y] + 1, 1);
  y = parent[top[y]];
}
```

General Unweight Graph Matching

- Complexity: $O(n^3)$ (?)

```cpp
struct BlossomMatch {
  int n;
  vector<vector<int>> e;
  BlossomMatch(int _n) : n(_n), e(_n) {}
  void add_edge(int u, int v) { e[u].push_back(v), e[v].push_back(u); }
  vector<int> find_matching() {
    vector<int> match(n, -1), vis(n), link(n), f(n), dep(n);
    function<int(int)> find = [&](int x) { return f[x] == x ? x : (f[x] = find(f[x])); };
    auto lca = [&](int u, int v) {
      u = find(u), v = find(v);
      while (u != v) {
        if (dep[u] < dep[v]) swap(u, v);
        u = find(link[match[u]]);
      }
      return u;
    };
    queue<int> que;
    auto blossom = [&](int u, int v, int p) {
      while (find(u) != p) {
        link[u] = v, v = match[u];
        if (vis[v] == 0) vis[v] = 1, que.push(v);
        f[u] = f[v] = p, u = link[v];
      }
    };
    // find an augmenting path starting from u and augment (if exist)
    auto augment = [&](int node) {
      while (!que.empty()) que.pop();
      iota(f.begin(), f.end(), 0);
      // vis = 0 corresponds to inner vertices, vis = 1 corresponds to outer vertices
      fill(vis.begin(), vis.end(), -1);
      que.push(node);
      vis[node] = 1, dep[node] = 0;
      while (!que.empty()) {
        int u = que.front();
        que.pop();
        for (auto v : e[u]) {
          if (vis[v] == -1) {
            vis[v] = 0, link[v] = u, dep[v] = dep[u] + 1;
            // found an augmenting path
            if (match[v] == -1) {
              for (int x = v, y = u, temp; y != -1; x = temp, y = x == -1 ? -1 : link[x]) {
                temp = match[y], match[x] = y, match[y] = x;
              }
              return;
            }
            vis[match[v]] = 1, dep[match[v]] = dep[u] + 2;
            que.push(match[v]);
          } else if (vis[v] == 1 && find(v) != find(u)) {
            // found a blossom
            int p = lca(u, v);
            blossom(u, v, p), blossom(v, u, p);
          }
        }
      }
    };
    // find a maximal matching greedily (decrease constant)
    auto greedy = [&]() {
      for (int u = 0; u < n; ++u) {
        if (match[u] != -1) continue;
        for (auto v : e[u]) {
          if (match[v] == -1) {
            match[u] = v, match[v] = u;
            break;
          }
        }
      }
    };
    greedy();
    for (int u = 0; u < n; ++u)
      if (match[u] == -1) augment(u);
    return match;
  }
};
```

Maximum Bipartite Matching

- Needs dinic, complexity $\approx O(n + m\sqrt{n})$

```cpp
struct BipartiteMatch {
  int l, r;
  Dinic dinic = Dinic(0);
  BipartiteMatch(int _l, int _r) : l(_l), r(_r) {
    dinic = Dinic(l + r + 2);
    for (int i = 1; i <= l; i++) dinic.add_edge(0, i, 1);
    for (int i = 1; i <= r; i++) dinic.add_edge(l + i, l + r + 1, 1);
  }
  void add_edge(int u, int v) { dinic.add_edge(u + 1, l + v + 1, 1); }
  ll max_matching() { return dinic.max_flow(0, l + r + 1); }
};
```

2-SAT and Strongly Connected Components

```cpp
void scc(vector<vector<int>>& g, int* idx) {
  int n = g.size(), ct = 0;
  int out[n];
  vector<int> ginv[n];
  memset(out, -1, sizeof out);
  memset(idx, -1, n * sizeof(int));
  function<void(int)> dfs = [&](int cur) {
    out[cur] = INT_MAX;
    for(int v : g[cur]) {
      ginv[v].push_back(cur);
      if(out[v] == -1) dfs(v);
    }
    ct++; out[cur] = ct;
  };
  vector<int> order;
  for(int i = 0; i < n; i++) {
    order.push_back(i);
    if(out[i] == -1) dfs(i);
  }
  sort(order.begin(), order.end(), [&](int& u, int& v) {
    return out[u] > out[v];
  });
  ct = 0;
  stack<int> s;
  auto dfs2 = [&](int start) {
    s.push(start);
    while(!s.empty()) {
      int cur = s.top();
      s.pop();
      idx[cur] = ct;
      for(int v : ginv[cur])
        if(idx[v] == -1) s.push(v);
    }
  };
  for(int v : order) {
    if(idx[v] == -1) {
      dfs2(v);
      ct++;
    }
  }
}

// 0 => impossible, 1 => possible
pair<int,vector<int>> sat2(int n, vector<pair<int,int>>& clauses) {
  vector<int> ans(n);
  vector<vector<int>> g(2*n + 1);
  for(auto [x, y] : clauses) {
    x = x < 0 ? -x + n : x;
    y = y < 0 ? -y + n : y;
    int nx = x <= n ? x + n : x - n;
    int ny = y <= n ? y + n : y - n;
    g[nx].push_back(y);
    g[ny].push_back(x);
  }
  int idx[2*n + 1];
  scc(g, idx);
  for(int i = 1; i <= n; i++) {
    if(idx[i] == idx[i + n]) return {0, {}};
    ans[i - 1] = idx[i + n] < idx[i];
  }
  return {1, ans};
}
```

Enumerating Triangles

- Complexity: $O(n + m\sqrt{m})$

```
void enumerate_triangles(vector<pair<int,int>>& edges, function<void(int,int,int)> f) {
  int n = 0;
  for(auto [u, v] : edges) n = max({n, u + 1, v + 1});
  vector<int> deg(n);
  vector<int> g[n];
  for(auto [u, v] : edges) {
    deg[u]++;
    deg[v]++;
  }
  for(auto [u, v] : edges) {
    if(u == v) continue;
    if(deg[u] > deg[v] || (deg[u] == deg[v] && u > v))
      swap(u, v);
    g[u].push_back(v);
  }
  vector<int> flag(n);
  for(int i = 0; i < n; i++) {
    for(int v : g[i]) flag[v] = 1;
    for(int v : g[i]) for(int u : g[v]) {
      if(flag[u]) f(i, v, u);
    }
    for(int v : g[i]) flag[v] = 0;
  }
}
```

Tarjan

- shrink all circles into points (2-edge-connected-component)

```
int cnt = 0, now = 0;
vector<ll> dfn(n, -1), low(n), belong(n, -1), stk;
function<void(ll, ll)> tarjan = [&](ll node, ll fa) {
  dfn[node] = low[node] = now++, stk.push_back(node);
  for (auto& ne : g[node]) {
    if (ne == fa) continue;
    if (dfn[ne] == -1) {
      tarjan(ne, node);
      low[node] = min(low[node], low[ne]);
    } else if (belong[ne] == -1) {
      low[node] = min(low[node], dfn[ne]);
    }
  }
  if (dfn[node] == low[node]) {
    while (true) {
      auto v = stk.back();
      belong[v] = cnt;
      stk.pop_back();
      if (v == node) break;
    }
    ++cnt;
  }
};
```

- 2-vertex-connected-component / Block forest

```
int cnt = 0, now = 0;
vector<vector<ll>> e1(n);
vector<ll> dfn(n, -1), low(n), stk;
function<void(ll)> tarjan = [&](ll node) {
  dfn[node] = low[node] = now++, stk.push_back(node);
  for (auto& ne : g[node]) {
    if (dfn[ne] == -1) {
      tarjan(ne);
      low[node] = min(low[node], low[ne]);
      if (low[ne] == dfn[node]) {
        e1.push_back({});
        while (true) {
          auto x = stk.back();
          stk.pop_back();
          e1[n + cnt].push_back(x);
          // e1[x].push_back(n + cnt); // undirected
          if (x == ne) break;
        }
        e1[node].push_back(n + cnt);
        // e1[n + cnt].push_back(node); // undirected
        cnt++;
      }
    } else {
      low[node] = min(low[node], dfn[ne]);
    }
  }
};
```

Kruskal reconstruct tree

```
int _n, m;
cin >> _n >> m; // _n: # of node, m: # of edge
int n = 2 * _n - 1; // root: n-1
vector<array<int, 3>> edges(m);
for (auto& [w, u, v] : edges) {
  cin >> u >> v >> w, u--, v--;
}
sort(edges.begin(), edges.end());
vector<int> p(n);
iota(p.begin(), p.end(), 0);
function<int(int)> find = [&](int x) { return p[x] == x ? x : (p[x] = find(p[x])); };
auto merge = [&](int x, int y) { p[find(x)] = find(y); };
vector<vector<int>> g(n);
vector<int> val(m);
val.reserve(n);
for (auto [w, u, v] : edges) {
  u = find(u), v = find(v);
  if (u == v) continue;
  val.push_back(w);
  int node = (int)val.size() - 1;
  g[node].push_back(u), g[node].push_back(v);
  merge(u, node), merge(v, node);
}
```

centroid decomposition

```
vector<char> res(n), seen(n), sz(n);
function<int(int, int)> get_size = [&](int node, int fa) {
  sz[node] = 1;
  for (auto& ne : g[node]) {
    if (ne == fa || seen[ne]) continue;
    sz[node] += get_size(ne, node);
  }
  return sz[node];
};
function<int(int, int, int)> find_centroid = [&](int node, int fa, int t) {
  for (auto& ne : g[node])
    if (ne != fa && !seen[ne] && sz[ne] > t / 2) return find_centroid(ne, node, t);
  return node;
};
function<void(int, char)> solve = [&](int node, char cur) {
  get_size(node, -1); auto c = find_centroid(node, -1, sz[node]);
  seen[c] = 1, res[c] = cur;
  for (auto& ne : g[c]) {
    if (seen[ne]) continue;
    solve(ne, char(cur + 1)); // we can pass c here to build tree
  }
};
```

virtual tree

```
map<int, vector<int>> gg; vector<int> stk{0};
auto add = [&](int x, int y) { gg[x].push_back(y), gg[y].push_back(x); };
for (int i = 0; i < k; i++) {
  if (a[i] != 0) {
    int p = lca(a[i], stk.back());
    if (p != stk.back()) {
      while (dfn[p] < dfn[stk[int(stk.size()) - 2]]) {
        add(stk.back(), stk[int(stk.size()) - 2]);
        stk.pop_back();
      }
      add(p, stk.back()), stk.pop_back();
      if (dfn[p] > dfn[stk.back()]) stk.push_back(p);
    }
    stk.push_back(a[i]);
  }
}
while (stk.size() > 1) {
  if (stk.back() != 0) {
    add(stk.back(), stk[int(stk.size()) - 2]);
    stk.pop_back();
  }
}
```

Math

Inverse

```
ll inv(ll a, ll m) { return a == 1 ? 1 : ((m - m / a) * inv(m % a, m) % m); }
// or
power(a, MOD - 2)
```

- USAGE: get factorial

```
vector<Z> f(MAX_N, 1), rf(MAX_N, 1);
for (int i = 2; i < MAX_N; i++) f[i] = f[i - 1] * i % MOD;
rf[MAX_N - 1] = power(f[MAX_N - 1], MOD - 2);
for (int i = MAX_N - 2; i > 1; i--) rf[i] = rf[i + 1] * (i + 1) % MOD;
```

```
auto binom = [&](ll n, ll r) -> Z {
  if (n < 0 || r < 0 || n < r) return 0;
  return f[n] * rf[n - r] * rf[r];
};
```

Mod Class

```
constexpr ll norm(ll x) { return (x % MOD + MOD) % MOD; }
template <typename T>
constexpr T power(T a, ll b, T res = 1) {
  for (; b; b /= 2, (a *= a) %= MOD)
    if (b & 1) (res *= a) %= MOD;
  return res;
}
struct Z {
  ll x;
  constexpr Z(ll _x = 0) : x(norm(_x)) {}
  // auto operator<=>(const Z &) const = default; // cpp20 only
  Z operator-() const { return Z(norm(MOD - x)); }
  Z inv() const { return power(*this, MOD - 2); }
  Z &operator*=(const Z &rhs) { return x = x * rhs.x % MOD, *this; }
  Z &operator+=(const Z &rhs) { return x = norm(x + rhs.x), *this; }
  Z &operator-=(const Z &rhs) { return x = norm(x - rhs.x), *this; }
  Z &operator/=(const Z &rhs) { return *this *= rhs.inv(); }
  Z &operator%=(const ll &rhs) { return x %= rhs, *this; }
  friend Z operator*(Z lhs, const Z &rhs) { return lhs *= rhs; }
  friend Z operator+(Z lhs, const Z &rhs) { return lhs += rhs; }
  friend Z operator-(Z lhs, const Z &rhs) { return lhs -= rhs; }
  friend Z operator/(Z lhs, const Z &rhs) { return lhs /= rhs; }
  friend Z operator%(Z lhs, const ll &rhs) { return lhs %= rhs; }
  friend auto &operator>>(istream &i, Z &z) { return i >> z.x; }
  friend auto &operator<<(ostream &o, const Z &z) { return o << z.x; }
};
```

- large mod (for NTT to do FFT in ll range without modulo)

```
constexpr i128 MOD = 9223372036737335297;
```

- fastest mod class! be careful with overflow, only use when the time limit is tight

```
constexpr int norm(int x) {
  if (x < 0) x += MOD;
  if (x >= MOD) x -= MOD;
  return x;
}
```

Combinatorics

```
const int NMAX = 3000010;
ll factorialcompute[NMAX];
ll invfactorialcompute[NMAX];
ll binpow(ll a, ll pow, ll mod) {
    if (pow <= 0)
        return 1;
    ll p = binpow(a, pow / 2, mod) % mod;
    p = (p * p) % mod;

    return (pow % 2 == 0) ? p : (a * p) % mod;
}
ll inverse(ll a, ll mod) {
    if (a == 1) return 1;
    return binpow(a, mod-2, mod);
}
```

```
ll combination(int a, int b, ll mod) {
    if ( a < b) return 0;
    ll cur = factorialcompute[a];
    cur *= invfactorialcompute[b];
    cur %= mod;
    cur *= invfactorialcompute[a - b];
    cur %= mod;
    return cur;
}
void precomputeFactorial() {
    factorialcompute[0] = 1;
    invfactorialcompute[0] = 1;
    for(int i = 1; i < NMAX; i++) {
        factorialcompute[i] = factorialcompute[i-1] * i;
        factorialcompute[i] %= MOD;
    }
    invfactorialcompute[NMAX-1] = inverse(factorialcompute[NMAX-1], MOD);
    for(int i = NMAX-2; i > -1; i--) {
        invfactorialcompute[i] = invfactorialcompute[i+1] * (i+1);
        invfactorialcompute[i] %= MOD;
    }
}
```

exgcd

```
array<ll, 3> exgcd(ll a, ll b) {
    if(!b) return {a, 1, 0};

    auto [g, x, y] = exgcd(b, a%b);
    return {g, y, x - a/b*y};
}
```

Factor/primes

```
vector<int> primes(0);
void gen_primes(int a) {
    vector<bool> is_prime(a+1, true);
    is_prime[0] = is_prime[1] = false;
    for(int i = 2; i * i <= a; i++) {
        if(is_prime[i]) {
            for(int j = i * i; j <= a; j += i) is_prime[j] = false;
        }
    }
    for(int i = 0; i <= a; i++) {
        if(is_prime[i]) primes.push_back(i);
    }
}
vector<ll> gen_factors_prime(ll a){
    vector<ll> factors;
    factors.push_back(1);
    if(a == 1) return factors;
    for(int z : primes) {
        if(z * z > a) {
            z = a;
        }
        int cnt = 0;
        while(a % z == 0) {
            cnt++;
            a /= z;
        }
        ll num = z;
        int size = factors.size();
        for(int i = 1; i <= cnt; i++) {
            for(int j = 0; j < size; j++) {
                factors.push_back(num * factors[j]);
            }
            num *= z;
        }
        if (a == 1) break;
    }
    return factors;
}
vector<int> get_primes(int num) {
    vector<int> curPrime;
    if(num == 1) return curPrime;
    for(int z : primes) {
        if(z * z > num) {
            curPrime.push_back(num);
            break;
        }
        if(num % z == 0) {
            curPrime.push_back(z);
            while(num % z == 0) num /= z;
        }
        if(num == 1) break;
    }
    return curPrime;
}
```

Cancer mod class

- Explanation: for some prime modulo p, maintains numbers of form p^x * y, where y is a nonzero remainder mod p
- Be careful with calling Cancer(x, y), it doesn't fix the input if y > p

```
struct Cancer {
  ll x; ll y;
  Cancer() : Cancer(0, 1) {}
  Cancer(ll _y) {
    x = 0, y = _y;
    while(y % MOD == 0) {
      y /= MOD;
      x++;
    }
  }
  Cancer(ll _x, ll _y) : x(_x), y(_y) {}
  Cancer inv() { return Cancer(-x, power(y, MOD - 2)); }
  Cancer operator*(const Cancer &c) { return Cancer(x + c.x, (y * c.y) % MOD); }
  Cancer operator*(ll m) {
    ll p = 0;
    while(m % MOD == 0) {
      m /= MOD;
      p++;
    }
    return Cancer(x + p, (m * y) % MOD);
  }
  friend auto &operator<<(ostream &o, Cancer c) { return o << c.x << ' ' << c.y; }
};
```

NTT, FFT, FWT

- ntt

```
void ntt(vector<Z>& a, int f) {
  int n = int(a.size());
  vector<Z> w(n);
  vector<int> rev(n);
  for (int i = 0; i < n; i++) rev[i] = (rev[i / 2] / 2) | ((i & 1) * (n / 2));
  for (int i = 0; i < n; i++) {
    if (i < rev[i]) swap(a[i], a[rev[i]]);
  }
  Z wn = power(f ? (MOD + 1) / 3 : 3, (MOD - 1) / n);
  w[0] = 1;
  for (int i = 1; i < n; i++) w[i] = w[i - 1] * wn;
  for (int mid = 1; mid < n; mid *= 2) {
    for (int i = 0; i < n; i += 2 * mid) {
      for (int j = 0; j < mid; j++) {
        Z x = a[i + j], y = a[i + j + mid] * w[n / (2 * mid) * j];
        a[i + j] = x + y, a[i + j + mid] = x - y;
      }
    }
  }
```

```cpp
  if (f) {
    Z iv = power(Z(n), MOD - 2);
    for (auto& x : a) x *= iv;
  }
}
```

- USAGE: Polynomial multiplication

```cpp
vector<Z> mul(vector<Z> a, vector<Z> b) {
  int n = 1, m = (int)a.size() + (int)b.size() - 1;
  while (n < m) n *= 2;
  a.resize(n), b.resize(n);
  ntt(a, 0), ntt(b, 0);
  for (int i = 0; i < n; i++) a[i] *= b[i];
  ntt(a, 1);
  a.resize(m);
  return a;
}
```

- FFT (should prefer NTT, only use this when input is not integer)

```cpp
const double PI = acos(-1);
auto mul = [&](const vector<double>& aa, const vector<double>& bb) {
  int n = (int)aa.size(), m = (int)bb.size(), bit = 1;
  while ((1 << bit) < n + m - 1) bit++;
  int len = 1 << bit;
  vector<complex<double>> a(len), b(len);
  vector<int> rev(len);
  for (int i = 0; i < n; i++) a[i].real(aa[i]);
  for (int i = 0; i < m; i++) b[i].real(bb[i]);
  for (int i = 0; i < len; i++) rev[i] = (rev[i >> 1] >> 1) | ((i & 1) << (bit - 1));
  auto fft = [&](vector<complex<double>>& p, int inv) {
    for (int i = 0; i < len; i++)
      if (i < rev[i]) swap(p[i], p[rev[i]]);
    for (int mid = 1; mid < len; mid *= 2) {
      auto w1 = complex<double>(cos(PI / mid), (inv ? -1 : 1) * sin(PI / mid));
      for (int i = 0; i < len; i += mid * 2) {
        auto wk = complex<double>(1, 0);
        for (int j = 0; j < mid; j++, wk = wk * w1) {
          auto x = p[i + j], y = wk * p[i + j + mid];
          p[i + j] = x + y, p[i + j + mid] = x - y;
        }
      }
    }
    if (inv == 1) {
      for (int i = 0; i < len; i++) p[i].real(p[i].real() / len);
    }
  };
  fft(a, 0), fft(b, 0);
  for (int i = 0; i < len; i++) a[i] = a[i] * b[i];
  fft(a, 1);
  a.resize(n + m - 1);
  vector<double> res(n + m - 1);
  for (int i = 0; i < n + m - 1; i++) res[i] = a[i].real();
  return res;
};
```

Polynomial Class

```cpp
using ll = long long;
constexpr ll MOD = 998244353;

ll norm(ll x) { return (x % MOD + MOD) % MOD; }
template <class T>
T power(T a, ll b, T res = 1) {
  for (; b; b /= 2, (a *= a) %= MOD)
    if (b & 1) (res *= a) %= MOD;
  return res;
}

struct Z {
  ll x;
  Z(ll _x = 0) : x(norm(_x)) {}
  // auto operator<=>(const Z &) const = default;
  Z operator-() const { return Z(norm(MOD - x)); }
  Z inv() const { return power(*this, MOD - 2); }
  Z &operator*=(const Z &rhs) { return x = x * rhs.x % MOD, *this; }
  Z &operator+=(const Z &rhs) { return x = norm(x + rhs.x), *this; }
  Z &operator-=(const Z &rhs) { return x = norm(x - rhs.x), *this; }
  Z &operator/=(const Z &rhs) { return *this *= rhs.inv(); }
  Z &operator%=(const ll &rhs) { return x %= rhs, *this; }
  friend Z operator*(Z lhs, const Z &rhs) { return lhs *= rhs; }
  friend Z operator+(Z lhs, const Z &rhs) { return lhs += rhs; }
  friend Z operator-(Z lhs, const Z &rhs) { return lhs -= rhs; }
  friend Z operator/(Z lhs, const Z &rhs) { return lhs /= rhs; }
  friend Z operator%(Z lhs, const ll &rhs) { return lhs %= rhs; }
  friend auto &operator>>(istream &i, Z &z) { return i >> z.x; }
  friend auto &operator<<(ostream &o, const Z &z) { return o << z.x; }
};

void ntt(vector<Z> &a, int f) {
  int n = (int)a.size();
  vector<Z> w(n);
  vector<int> rev(n);
  for (int i = 0; i < n; i++) rev[i] = (rev[i / 2] / 2) | ((i & 1) * (n / 2));
  for (int i = 0; i < n; i++)
    if (i < rev[i]) swap(a[i], a[rev[i]]);
  Z wn = power(ll(f ? (MOD + 1) / 3 : 3), (MOD - 1) / n);
  w[0] = 1;
  for (int i = 1; i < n; i++) w[i] = w[i - 1] * wn;
  for (int mid = 1; mid < n; mid *= 2) {
    for (int i = 0; i < n; i += 2 * mid) {
      for (int j = 0; j < mid; j++) {
        Z x = a[i + j], y = a[i + j + mid] * w[n / (2 * mid) * j];
        a[i + j] = x + y, a[i + j + mid] = x - y;
      }
    }
  }
  if (f) {
    Z iv = power(Z(n), MOD - 2);
    for (int i = 0; i < n; i++) a[i] *= iv;
  }
}

struct Poly {
  vector<Z> a;
  Poly() {}
  Poly(const vector<Z> &_a) : a(_a) {}
  int size() const { return (int)a.size(); }
  void resize(int n) { a.resize(n); }
  Z operator[](int idx) const {
    if (idx < 0 || idx >= size()) return 0;
    return a[idx];
  }
  Z &operator[](int idx) { return a[idx]; }
  Poly mulxk(int k) const {
    auto b = a;
    b.insert(b.begin(), k, 0);
    return Poly(b);
  }
  Poly modxk(int k) const { return Poly(vector<Z>(a.begin(), a.begin() + min(k, size()))); }
  Poly divxk(int k) const {
    if (size() <= k) return Poly();
    return Poly(vector<Z>(a.begin() + k, a.end()));
  }
  friend Poly operator+(const Poly &a, const Poly &b) {
    vector<Z> res(max(a.size(), b.size()));
    for (int i = 0; i < (int)res.size(); i++) res[i] = a[i] + b[i];
    return Poly(res);
  }
  friend Poly operator-(const Poly &a, const Poly &b) {
    vector<Z> res(max(a.size(), b.size()));
    for (int i = 0; i < (int)res.size(); i++) res[i] = a[i] - b[i];
    return Poly(res);
  }
  friend Poly operator*(Poly a, Poly b) {
    if (a.size() == 0 || b.size() == 0) return Poly();
    int n = 1, m = (int)a.size() + (int)b.size() - 1;
    while (n < m) n *= 2;
    a.resize(n), b.resize(n);
    ntt(a.a, 0), ntt(b.a, 0);
    for (int i = 0; i < n; i++) a[i] *= b[i];
    ntt(a.a, 1);
    a.resize(m);
    return a;
  }
  friend Poly operator*(Z a, Poly b) {
    for (int i = 0; i < (int)b.size(); i++) b[i] *= a;
    return b;
  }
  friend Poly operator*(Poly a, Z b) {
    for (int i = 0; i < (int)a.size(); i++) a[i] *= b;
    return a;
  }
  Poly &operator+=(Poly b) { return (*this) = (*this) + b; }
  Poly &operator-=(Poly b) { return (*this) = (*this) - b; }
  Poly &operator*=(Poly b) { return (*this) = (*this) * b; }
  Poly deriv() const {
    if (a.empty()) return Poly();
    vector<Z> res(size() - 1);
    for (int i = 0; i < size() - 1; ++i) res[i] = (i + 1) * a[i + 1];
    return Poly(res);
  }
  Poly integr() const {
    vector<Z> res(size() + 1);
    for (int i = 0; i < size(); ++i) res[i + 1] = a[i] / (i + 1);
    return Poly(res);
  }
  Poly inv(int m) const {
    Poly x({a[0].inv()});
    int k = 1;
    while (k < m) {
      k *= 2;
      x = (x * (Poly({2}) - modxk(k) * x)).modxk(k);
    }
    return x.modxk(m);
  }
  Poly log(int m) const { return (deriv() * inv(m)).integr().modxk(m); }
  Poly exp(int m) const {
    Poly x({1});
    int k = 1;
    while (k < m) {
      k *= 2;
      x = (x * (Poly({1}) - x.log(k) + modxk(k))).modxk(k);
    }
    return x.modxk(m);
  }
  Poly pow(int k, int m) const {
    int i = 0;
    while (i < size() && a[i].x == 0) i++;
    if (i == size() || 1LL * i * k >= m) {
      return Poly(vector<Z>(m));
    }
    Z v = a[i];
    auto f = divxk(i) * v.inv();
    return (f.log(m - i * k) * k).exp(m - i * k).mulxk(i * k) * power(v, k);
  }
  Poly sqrt(int m) const {
    Poly x({1});
    int k = 1;
    while (k < m) {
      k *= 2;
      x = (x + (modxk(k) * x.inv(k)).modxk(k)) * ((MOD + 1) / 2);
    }
    return x.modxk(m);
  }
  Poly mulT(Poly b) const {
    if (b.size() == 0) return Poly();
    int n = b.size();
    reverse(b.a.begin(), b.a.end());
    return ((*this) * b).divxk(n - 1);
  }
  Poly divmod(Poly b) const {
    auto n = size(), m = b.size();
    auto t = *this;
    reverse(t.a.begin(), t.a.end());
    reverse(b.a.begin(), b.a.end());
    Poly res = (t * b.inv(n)).modxk(n - m + 1);
    reverse(res.a.begin(), res.a.end());
    return res;
  }
  vector<Z> eval(vector<Z> x) const {
    if (size() == 0) return vector<Z>(x.size(), 0);
    const int n = max(int(x.size()), size());
    vector<Poly> q(4 * n);
    vector<Z> ans(x.size());
    x.resize(n);
    function<void(int, int, int)> build = [&](int p, int l, int r) {
      if (r - l == 1) {
        q[p] = Poly({1, -x[l]});
      } else {
        int m = (l + r) / 2;
        build(2 * p, l, m), build(2 * p + 1, m, r);
        q[p] = q[2 * p] * q[2 * p + 1];
      }
    };
    build(1, 0, n);
    auto work = [&](auto self, int p, int l, int r, const Poly &num) -> void {
      if (r - l == 1) {
        if (l < int(ans.size())) ans[l] = num[0];
      } else {
        int m = (l + r) / 2;
        self(self, 2 * p, l, m, num.mulT(q[2 * p + 1]).modxk(m - l));
        self(self, 2 * p + 1, m, r, num.mulT(q[2 * p]).modxk(r - m));
      }
    };
    work(work, 1, 0, n, mulT(q[1].inv(n)));
    return ans;
  }
};
```

Sieve

- linear sieve

```
vector<int> min_primes(MAX_N), primes;
primes.reserve(1e5);
for (int i = 2; i < MAX_N; i++) {
  if (!min_primes[i]) min_primes[i] = i, primes.push_back(i);
  for (auto& p : primes) {
    if (p * i >= MAX_N) break;
    min_primes[p * i] = p;
    if (i % p == 0) break;
  }
}
```

- mobius function

```
vector<int> min_p(MAX_N), mu(MAX_N), primes;
```

```
mu[1] = 1, primes.reserve(1e5);
for (int i = 2; I < MAX_N; i++) {
  if (min_p[i] == 0) {
    min_p[i] = i;
    primes.push_back(i);
    mu[i] = -1;
  }
  for (auto p : primes) {
    if (i * p >= MAX_N) break;
    min_p[i * p] = p;
    if (i % p == 0) {
      mu[i * p] = 0;
      break;
    }
    mu[i * p] = -mu[i];
  }
}
```

- Euler's totient function

```
vector<int> min_p(MAX_N), phi(MAX_N), primes;
phi[1] = 1, primes.reserve(1e5);
for (int i = 2; i < MAX_N; i++) {
  if (min_p[i] == 0) {
    min_p[i] = i;
    primes.push_back(i);
    phi[i] = i - 1;
  }
  for (auto p : primes) {
    if (i * p >= MAX_N) break;
    min_p[i * p] = p;
    if (i % p == 0) {
      phi[i * p] = phi[i] * p;
      break;
    }
    phi[i * p] = phi[i] * phi[p];
  }
}
```

Gaussian Elimination

```
bool is_0(Z v) { return v.x == 0; }
Z abs(Z v) { return v; }
bool is_0(double v) { return abs(v) < 1e-9; }

// 1 => unique solution, 0 => no solution, -1 => multiple solutions
template <typename T>
int gaussian_elimination(vector<vector<T>> &a, int limit) {
    if (a.empty() || a[0].empty()) return -1;
  int h = (int)a.size(), w = (int)a[0].size(), r = 0;
  for (int c = 0; c < limit; c++) {
    int id = -1;
    for (int i = r; i < h; i++) {
      if (!is_0(a[i][c]) && (id == -1 || abs(a[id][c]) < abs(a[i][c]))) {
        id = i;
      }
    }
    if (id == -1) continue;
    if (id > r) {
      swap(a[r], a[id]);
      for (int j = c; j < w; j++) a[id][j] = -a[id][j];
    }
    vector<int> nonzero;
    for (int j = c; j < w; j++) {
      if (!is_0(a[r][j])) nonzero.push_back(j);
    }
    T inv_a = 1 / a[r][c];
    for (int i = r + 1; i < h; i++) {
      if (is_0(a[i][c])) continue;
      T coeff = -a[i][c] * inv_a;
      for (int j : nonzero) a[i][j] += coeff * a[r][j];
    }
    ++r;
  }
  for (int row = h - 1; row >= 0; row--) {
    for (int c = 0; c < limit; c++) {
      if (!is_0(a[row][c])) {
        T inv_a = 1 / a[row][c];
        for (int i = row - 1; i >= 0; i--) {
          if (is_0(a[i][c])) continue;
          T coeff = -a[i][c] * inv_a;
          for (int j = c; j < w; j++) a[i][j] += coeff * a[row][j];
        }
        break;
      }
    }
  } // not-free variables: only it on its line
  for(int i = r; i < h; i++) if(!is_0(a[i][limit])) return 0;
  return (r == limit) ? 1 : -1;
}

template <typename T>
pair<int,vector<T>> solve_linear(vector<vector<T>> a, const vector<T> &b, int w) {
  int h = (int)a.size();
  for (int i = 0; i < h; i++) a[i].push_back(b[i]);
  int sol = gaussian_elimination(a, w);
  if(!sol) return {0, vector<T>()};
  vector<T> x(w, 0);
  for (int i = 0; i < h; i++) {
    for (int j = 0; j < w; j++) {
      if (!is_0(a[i][j])) {
        x[j] = a[i][w] / a[i][j];
        break;
      }
    }
  }
  return {sol, x};
}
```

is_prime

- (Miller–Rabin primality test)

```
i128 power(i128 a, i128 b, i128 MOD = 1, i128 res = 1) {
  for (; b; b /= 2, (a *= a) %= MOD)
    if (b & 1) (res *= a) %= MOD;
  return res;
}

bool is_prime(ll n) {
  if (n < 2) return false;
  static constexpr int A[] = {2, 3, 5, 7, 11, 13, 17, 19, 23};
  int s = __builtin_ctzll(n - 1);
  ll d = (n - 1) >> s;
  for (auto a : A) {
    if (a == n) return true;
    ll x = (ll)power(a, d, n);
    if (x == 1 || x == n - 1) continue;
    bool ok = false;
    for (int i = 0; i < s - 1; ++i) {
      x = ll((i128)x * x % n);  // potential overflow!
      if (x == n - 1) {
        ok = true;
        break;
      }
    }
    if (!ok) return false;
  }
  return true;
}
```

```
ll pollard_rho(ll x) {
  ll s = 0, t = 0, c = rng() % (x - 1) + 1;
  ll stp = 0, goal = 1, val = 1;
  for (goal = 1;; goal *= 2, s = t, val = 1) {
    for (stp = 1; stp <= goal; ++stp) {
      t = ll(((i128)t * t + c) % x);
      val = ll((i128)val * abs(t - s) % x);
      if ((stp % 127) == 0) {
        ll d = gcd(val, x);
        if (d > 1) return d;
      }
    }
    ll d = gcd(val, x);
    if (d > 1) return d;
  }
}

ll get_max_factor(ll _x) {
  ll max_factor = 0;
  function<void(ll)> fac = [&](ll x) {
    if (x <= max_factor || x < 2) return;
    if (is_prime(x)) {
      max_factor = max_factor > x ? max_factor : x;
      return;
    }
    ll p = x;
    while (p >= x) p = pollard_rho(x);
    while ((x % p) == 0) x /= p;
    fac(x), fac(p);
  };
  fac(_x);
  return max_factor;
}
```

Radix Sort

```
struct identity {
    template<typename T>
    T operator()(const T &x) const {
        return x;
    }
};
// A stable sort that sorts in passes of `bits_per_pass` bits at a time.
template<typename T, typename T_extract_key = identity>
void radix_sort(vector<T> &data, int bits_per_pass = 10, const T_extract_key &extract_key = identity()) {
    if (int64_t(data.size()) * (64 - __builtin_clzll(data.size())) < 2 * (1 << bits_per_pass)) {
        stable_sort(data.begin(), data.end(), [&](const T &a, const T &b) {
            return extract_key(a) < extract_key(b);
        });
        return;
    }

    using T_key = decltype(extract_key(data.front()));
    T_key minimum = numeric_limits<T_key>::max();
    for (T &x : data)
        minimum = min(minimum, extract_key(x));

    int max_bits = 0;
    for (T &x : data) {
        T_key key = extract_key(x);
        max_bits = max(max_bits, key == minimum ? 0 : 64 - __builtin_clzll(key - minimum));
    }
    int passes = max((max_bits + bits_per_pass / 2) / bits_per_pass, 1);
    if (64 - __builtin_clzll(data.size()) <= 1.5 * passes) {
        stable_sort(data.begin(), data.end(), [&](const T &a, const T &b) {
            return extract_key(a) < extract_key(b);
        });
        return;
    }
    vector<T> buffer(data.size());
    vector<int> counts;
    int bits_so_far = 0;

    for (int p = 0; p < passes; p++) {
        int bits = (max_bits + p) / passes;
        counts.assign(1 << bits, 0);
        for (T &x : data) {
            T_key key = T_key(extract_key(x) - minimum);
            counts[(key >> bits_so_far) & ((1 << bits) - 1)]++;
        }
        int count_sum = 0;
        for (int &count : counts) {
            int current = count;
            count = count_sum;
            count_sum += current;
        }
        for (T &x : data) {
            T_key key = T_key(extract_key(x) - minimum);
            int key_section = int((key >> bits_so_far) & ((1 << bits) - 1));
            buffer[counts[key_section]++] = x;
        }
        swap(data, buffer);
        bits_so_far += bits;
    }
}
```

- USAGE

```
radix_sort(edges, 10, [&](const edge &e) -> int { return abs(e.weight - x); });
```

lucas

```
ll lucas(ll n, ll m, ll p) {
  if (m == 0) return 1;
  return (binom(n % p, m % p, p) * lucas(n / p, m / p, p)) % p;
}
```

parity of n choose m

```
(n & m) == m <=> odd
```

sosdp

subset sum

```
auto f = a;
for (int i = 0; i < SZ; i++) {
  for (int mask = 0; mask < (1 << SZ); mask++) {
    if (mask & (1 << i)) f[mask] += f[mask ^ (1 << i)];
  }
}
```

prf

```
ll _h(ll x) { return x * x * x * 1241483 + 19278349; }
ll prf(ll x) { return _h(x & ((1 << 31) - 1)) + _h(x >> 31); }
```

String

AC Automaton

```
struct AC_automaton {
  int sz = 26;
  vector<vector<int>> e = {vector<int>(sz)};  // vector is faster than unordered_map
  vector<int> fail = {0}, end = {0};
  vector<int> fast = {0};  // closest end

  int insert(string& s) {
    int p = 0;
    for (auto c : s) {
      c -= 'a';
      if (!e[p][c]) {
        e.emplace_back(sz);
        fail.emplace_back();
        end.emplace_back();
        fast.emplace_back();
        e[p][c] = (int)e.size() - 1;
      }
      p = e[p][c];
    }
    end[p] += 1;
    return p;
  }

  void build() {
    queue<int> q;
    for (int i = 0; i < sz; i++)
      if (e[0][i]) q.push(e[0][i]);
    while (!q.empty()) {
      int p = q.front();
      q.pop();
      fast[p] = end[p] ? p : fast[fail[p]];
      for (int i = 0; i < sz; i++) {
        if (e[p][i]) {
          fail[e[p][i]] = e[fail[p]][i];
          q.push(e[p][i]);
        } else {
          e[p][i] = e[fail[p]][i];
        }
      }
    }
  }
};
```

KMP

- nex[i]: length of longest common prefix & suffix for pat[0..i]

```
vector<int> get_next(vector<int> &pat) {
  int m = (int)pat.size();
  vector<int> nex(m);
  for (int i = 1, j = 0; i < m; i++) {
    while (j && pat[j] != pat[i]) j = nex[j - 1];
    if (pat[j] == pat[i]) j++;
    nex[i] = j;
  }
  return nex;
}
```

- kmp match for txt and pat

```
auto nex = get_next(pat);
for (int i = 0, j = 0; i < n; i++) {
  while (j && pat[j] != txt[i]) j = nex[j - 1];
  if (pat[j] == txt[i]) j++;
  if (j == m) {
    // do what you want with the match
    // start index is `i - m + 1`
    j = nex[j - 1];
  }
}
```

Z function

- z[i]: length of longest common prefix of s and s[i:]

```
vector<int> z_function(string s) {
  int n = (int)s.size();
  vector<int> z(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[z[i]] == s[i + z[i]]) ++z[i];
    if (i + z[i] - 1 > r) l = i, r = i + z[i] - 1;
  }
  return z;
}
```

General Suffix Automaton

```
constexpr int SZ = 26;

struct GSAM {
  vector<vector<int>> e = {vector<int>(SZ)};  // the labeled edges from node i
  vector<int> parent = {-1};                  // the parent of i
  vector<int> length = {0};                   // the length of the longest string

  GSAM(int n) { e.reserve(2 * n), parent.reserve(2 * n), length.reserve(2 * n); };
  int extend(int c, int p) {  // character, last
    bool f = true;            // if already exist
    int r = 0;                // potential new node
    if (!e[p][c]) {           // only extend when not exist
      f = false;
      e.push_back(vector<int>(SZ));
      parent.push_back(0);
      length.push_back(length[p] + 1);
      r = (int)e.size() - 1;
      for (; ~p && !e[p][c]; p = parent[p]) e[p][c] = r;  // update parents
    }
    if (f || ~p) {
      int q = e[p][c];
      if (length[q] == length[p] + 1) {
        if (f) return q;
        parent[r] = q;
      } else {
        e.push_back(e[q]);
        parent.push_back(parent[q]);
        length.push_back(length[p] + 1);
        int qq = parent[q] = (int)e.size() - 1;
        for (; ~p && e[p][c] == q; p = parent[p]) e[p][c] = qq;
        if (f) return qq;
        parent[r] = qq;
      }
    }
    return r;
  }
};
```

- Topo sort on GSAM

```
ll sz = gsam.e.size();
vector<int> c(sz + 1);
vector<int> order(sz);
for (int i = 1; i < sz; i++) c[gsam.length[i]]++;
for (int i = 1; i < sz; i++) c[i] += c[i - 1];
for (int i = 1; i < sz; i++) order[c[gsam.length[i]]--] = i;
reverse(order.begin(), order.end()); // reverse so that large len to small
```

- can be used as an ordinary SAM
- USAGE (the number of distinct substring)

```
int main() {
  int n, last = 0;
  string s;
  cin >> n;
  auto a = GSAM();
  for (int i = 0; i < n; i++) {
    cin >> s;
    last = 0;  // reset last
    for (auto&& c : s) last = a.extend(c, last);
  }
  ll ans = 0;
  for (int i = 1; i < a.e.size(); i++) {
    ans += a.length[i] - a.length[a.parent[i]];
  }
  cout << ans << endl;
  return 0;
}
```

Manacher

```
string longest_palindrome(string& s) {
  // init "abc" -> "^$a#b#c$"
  vector<char> t{'^', '#'};
  for (char c : s) t.push_back(c), t.push_back('#');
  t.push_back('$');
  // manacher
  int n = t.size(), r = 0, c = 0;
  vector<int> p(n, 0);
  for (int i = 1; i < n - 1; i++) {
    if (i < r + c) p[i] = min(p[2 * c - i], r + c - i);
    while (t[i + p[i] + 1] == t[i - p[i] - 1]) p[i]++;
    if (i + p[i] > r + c) r = p[i], c = i;
  }
    // s[i] -> p[2 * i + 2] (even), p[2 * i + 2] (odd)
  // output answer
  int index = 0;
  for (int i = 0; i < n; i++)
    if (p[index] < p[i]) index = i;
  return s.substr((index - p[index]) / 2, p[index]);
}
```

Lyndon

- def: suf(s) > s

```
void duval(const string &s) {
  int n = (int)s.size();
  for (int i = 0; i < n;) {
    int j = i, k = i + 1;
    for (; j < n && s[j] <= s[k]; j++, k++)
      if (s[j] < s[k]) j = i - 1;

    while (i <= j) {
      // cout << s.substr(i, k - j) << '\n';
      i += k - j;
    }
  }
}
```

minimal representation

```
int k = 0, i = 0, j = 1;
while (k < n && i < n && j < n) {
  if (s[(i + k) % n] == s[(j + k) % n]) {
    k++;
  } else {
    s[(i + k) % n] > s[(j + k) % n] ? i = i + k + 1 : j = j + k + 1;
    if (i == j) i++;
    k = 0;
  }
}
i = min(i, j); // from 0
```

suffix array

```
vi classTable[21];
vector<int> suffix_array(string const& s) {
    forn(i, 21) classTable[i].clear();

    int n = s.size();
    const int alphabet = 256;
    vector<int> p(n), c(n), cnt(max(alphabet, n), 0);
    for (int i = 0; i < n; i++)
        cnt[s[i]]++;
    for (int i = 1; i < alphabet; i++)
        cnt[i] += cnt[i-1];
    for (int i = 0; i < n; i++)
        p[--cnt[s[i]]] = i;
    c[p[0]] = 0;
    int classes = 1;
    for (int i = 1; i < n; i++) {
        if (s[p[i]] != s[p[i-1]])
            classes++;
        c[p[i]] = classes - 1;
    }
    classTable[0] = c;
    vector<int> pn(n), cn(n);
    for (int h = 0; (1 << h) < n; ++h) {
        for (int i = 0; i < n; i++) {
            pn[i] = p[i] - (1 << h);
            if (pn[i] < 0)
                pn[i] += n;
        }
        fill(cnt.begin(), cnt.begin() + classes, 0);
        for (int i = 0; i < n; i++)
            cnt[c[pn[i]]]++;
        for (int i = 1; i < classes; i++)
            cnt[i] += cnt[i-1];
        for (int i = n-1; i >= 0; i--)
            p[--cnt[c[pn[i]]]] = pn[i];
        cn[p[0]] = 0;
        classes = 1;
        for (int i = 1; i < n; i++) {
            pair<int, int> cur = {c[p[i]], c[(p[i] + (1 << h)) % n]};
            pair<int, int> prev = {c[p[i-1]], c[(p[i-1] + (1 << h)) % n]};
            if (cur != prev)
                ++classes;
            cn[p[i]] = classes - 1;
        }
        c.swap(cn);
        classTable[h+1] = c;
    }
    return p;
}

int lcp(int a, int b) {
    int ans = 0;
    for(int i = 19; i >= 0; i--) {
        if(classTable[i].size() == 0) continue;
        if(classTable[i][a] == classTable[i][b]) {
            a += (1 << i);
            b += (1 << i);
            ans += (1 << i);
        }
    }
    return ans;
}
```