

# CU-Later Code Library

yangster67, ikaurov, serichao

May 21th 2024

# Contents

## Templates

Ken's template . . . . .	1
Kevin's template . . . . .	1

## Geometry

### Geometry

Basic stuff . . . . .	3
Transformation . . . . .	4
Relation . . . . .	4
Area . . . . .	5
Convex . . . . .	6
Basic 3D . . . . .	7
Miscellaneous . . . . .	7

### Graph Theory

Max Flow . . . . .	7
PushRelabel Max-Flow (faster) . . . . .	8
Min-Cost Max-Flow . . . . .	8
Max Cost Feasible Flow . . . . .	9
Heavy-Light Decomposition . . . . .	9
General Unweight Graph Matching . . . . .	10
Maximum Bipartite Matching . . . . .	10
2-SAT and Strongly Connected Components . . . . .	10
Enumerating Triangles . . . . .	11
Tarjan . . . . .	11
Kruskal reconstruct tree . . . . .	11
centroid decomposition . . . . .	12
virtual tree . . . . .	12

### Math

Inverse . . . . .	12
Mod Class . . . . .	12
Combinatorics . . . . .	12
exgcd . . . . .	13
Factor/primes . . . . .	13
Cancer mod class . . . . .	13
NTT, FFT, FWT . . . . .	13
Polynomial Class . . . . .	14
Sieve . . . . .	15
Gaussian Elimination . . . . .	16
is_prime . . . . .	16
Radix Sort . . . . .	17
lucas . . . . .	17
parity of n choose m . . . . .	17
sosdp . . . . .	17
prf . . . . .	17

### String

AC Automaton . . . . .	17
KMP . . . . .	18
Z function . . . . .	18
General Suffix Automaton . . . . .	18
Manacher . . . . .	18
Lyndon . . . . .	19
minimal representation . . . . .	19
suffix array . . . . .	19

# Templates

## Ken's template

```
1 #include <bits/stdc++.h>
2 using namespace std;
3 #define all(v) (v).begin(), (v).end()
4 typedef long long ll;
5 typedef long double ld;
6 #define pb push_back
7 #define sz(x) (int)(x).size()
8 #define fi first
9 #define se second
10 #define endl '\n'
```

## Kevin's template

```
1 // paste Kaurov's Template, minus last line
2 typedef vector<int> vi;
3 typedef vector<ll> vll;
4 typedef pair<int, int> pii;
5 typedef pair<ll, ll> pll;
6 typedef pair<double, double> pdd;
7 const ld PI = acosl(-1);
8 const ll mod7 = 1e9 + 7;
9 const ll mod9 = 998244353;
10 const ll INF = 2*1024*1024*1023;
11 const char nl = '\n';
12 #define forn(i, n) for (int i = 0; i < int(n); i++)
13 #pragma GCC target("avx2,bmi,bmi2,lzcnt,popcnt")
14 #include <ext/pb_ds/assoc_container.hpp>
15 #include <ext/pb_ds/tree_policy.hpp>
16 using namespace __gnu_pbds;
17 template<class T> using ordered_set = tree<T, null_type,
18     ↪ less<T>, rb_tree_tag, tree_order_statistics_node_update>;
19 string s, t;
20 vi d4x = {1, 0, -1, 0};
21 vi d4y = {0, 1, 0, -1};
22 vi d8x = {1, 0, -1, 0, 1, 1, -1, -1};
23 vi d8y = {0, 1, 0, -1, 1, -1, 1, -1};
24 mt19937
25     ↪ rng(chrono::steady_clock::now().time_since_epoch().count());
26
27 bool multiTest = 1;
28 void solve(int tt){
29 }
30
31 int main(){
32     ios::sync_with_stdio(0);cin.tie(0);cout.tie(0);
33     cout<<fixed<< setprecision(14);
34
35     int t = 1;
36     if (multiTest) cin >> t;
37     forn(ii, t) solve(ii);
38 }
```

## Geometry

```
1 template<typename T>
2 struct TPoint{
3     T x, y;
4     int id;
5     static constexpr T eps = static_cast<T>(1e-9);
6     TPoint() : x(0), y(0), id(-1) {}
7     TPoint(const T& x_, const T& y_) : x(x_), y(y_), id(-1) {}
8     TPoint(const T& x_, const T& y_, const int id_) : x(x_),
9         ↪ y(y_), id(id_) {}
10
11     TPoint operator + (const TPoint& rhs) const {
12         return TPoint(x + rhs.x, y + rhs.y);
13     }
14     TPoint operator - (const TPoint& rhs) const {
15         return TPoint(x - rhs.x, y - rhs.y);
16     }
17 }
```

```

15     }
16     TPoint operator * (const T& rhs) const {
17         return TPoint(x * rhs, y * rhs);
18     }
19     TPoint operator / (const T& rhs) const {
20         return TPoint(x / rhs, y / rhs);
21     }
22     TPoint ort() const {
23         return TPoint(-y, x);
24     }
25     T abs2() const {
26         return x * x + y * y;
27     }
28 };
29 template<typename T>
30 bool operator< (TPoint<T>& A, TPoint<T>& B){
31     return make_pair(A.x, A.y) < make_pair(B.x, B.y);
32 }
33 template<typename T>
34 bool operator==(TPoint<T>& A, TPoint<T>& B){
35     return abs(A.x - B.x) <= TPoint<T>::eps && abs(A.y - B.y) <=
        ↪ TPoint<T>::eps;
36 }
37 template<typename T>
38 struct TLine{
39     T a, b, c;
40     TLine() : a(0), b(0), c(0) {}
41     TLine(const T& a_, const T& b_, const T& c_) : a(a_), b(b_),
        ↪ c(c_) {}
42     TLine(const TPoint<T>& p1, const TPoint<T>& p2){
43         a = p1.y - p2.y;
44         b = p2.x - p1.x;
45         c = -a * p1.x - b * p1.y;
46     }
47 };
48 template<typename T>
49 T det(const T& a11, const T& a12, const T& a21, const T& a22){
50     return a11 * a22 - a12 * a21;
51 }
52 template<typename T>
53 T sq(const T& a){
54     return a * a;
55 }
56 template<typename T>
57 T smul(const TPoint<T>& a, const TPoint<T>& b){
58     return a.x * b.x + a.y * b.y;
59 }
60 template<typename T>
61 T vmul(const TPoint<T>& a, const TPoint<T>& b){
62     return det(a.x, a.y, b.x, b.y);
63 }
64 template<typename T>
65 bool parallel(const TLine<T>& l1, const TLine<T>& l2){
66     return abs(vmul(TPoint<T>(l1.a, l1.b), TPoint<T>(l2.a,
        ↪ l2.b))) <= TPoint<T>::eps;
67 }
68 template<typename T>
69 bool equivalent(const TLine<T>& l1, const TLine<T>& l2){
70     return parallel(l1, l2) &&
71         abs(det(l1.b, l1.c, l2.b, l2.c)) <= TPoint<T>::eps &&
72         abs(det(l1.a, l1.c, l2.a, l2.c)) <= TPoint<T>::eps;
73 }
74 template<typename T>
75 TPoint<T> intersection(const TLine<T>& l1, const TLine<T>&
        ↪ l2){
76     return TPoint<T>(
77         det(-l1.c, l1.b, -l2.c, l2.b) / det(l1.a, l1.b, l2.a,
        ↪ l2.b),
78         det(l1.a, -l1.c, l2.a, -l2.c) / det(l1.a, l1.b, l2.a,
        ↪ l2.b)
79     );
80 }
81 template<typename T>
82 int sign(const T& x){
83     if (abs(x) <= TPoint<T>::eps) return 0;
84     return x > 0? +1 : -1;
85 }
86 template<typename T>
87 T area(const vector<TPoint<T>>& pts){
88     int n = sz(pts);
89     T ans = 0;
90     for (int i = 0; i < n; i++){
91         ans += vmul(pts[i], pts[(i + 1) % n]);
92     }
93     return abs(ans) / 2;
94 }
95 template<typename T>
96 T dist_pp(const TPoint<T>& a, const TPoint<T>& b){
97     return sqrt(sq(a.x - b.x) + sq(a.y - b.y));
98 }
99 template<typename T>
100 TLine<T> perp_line(const TLine<T>& l, const TPoint<T>& p){
101     T na = -l.b, nb = l.a, nc = - na * p.x - nb * p.y;
102     return TLine<T>(na, nb, nc);
103 }
104 template<typename T>
105 TPoint<T> projection(const TPoint<T>& p, const TLine<T>& l){
106     return intersection(l, perp_line(l, p));
107 }
108 template<typename T>
109 T dist_pl(const TPoint<T>& p, const TLine<T>& l){
110     return dist_pp(p, projection(p, l));
111 }
112 template<typename T>
113 struct TRay{
114     TLine<T> l;
115     TPoint<T> start, dirvec;
116     TRay() : l(), start(), dirvec() {}
117     TRay(const TPoint<T>& p1, const TPoint<T>& p2){
118         l = TLine<T>(p1, p2);
119         start = p1, dirvec = p2 - p1;
120     }
121 };
122 template<typename T>
123 bool is_on_line(const TPoint<T>& p, const TLine<T>& l){
124     return abs(l.a * p.x + l.b * p.y + l.c) <= TPoint<T>::eps;
125 }
126 template<typename T>
127 bool is_on_ray(const TPoint<T>& p, const TRay<T>& r){
128     if (is_on_line(p, r.l)){
129         return sign(smul(r.dirvec, TPoint<T>(p - r.start))) != -1;
130     }
131     else return false;
132 }
133 template<typename T>
134 bool is_on_seg(const TPoint<T>& P, const TPoint<T>& A, const
        ↪ TPoint<T>& B){
135     return is_on_ray(P, TRay<T>(A, B)) && is_on_ray(P,
        ↪ TRay<T>(B, A));
136 }
137 template<typename T>
138 T dist_pr(const TPoint<T>& P, const TRay<T>& R){
139     auto H = projection(P, R.l);
140     return is_on_ray(H, R)? dist_pp(P, H) : dist_pp(P, R.start);
141 }
142 template<typename T>
143 T dist_ps(const TPoint<T>& P, const TPoint<T>& A, const
        ↪ TPoint<T>& B){
144     auto H = projection(P, TLine<T>(A, B));
145     if (is_on_seg(H, A, B)) return dist_pp(P, H);
146     else return min(dist_pp(P, A), dist_pp(P, B));
147 }
148 template<typename T>
149 bool acw(const TPoint<T>& A, const TPoint<T>& B){
150     T mul = vmul(A, B);
151     return mul > 0 || abs(mul) <= TPoint<T>::eps;
152 }
153 template<typename T>
154 bool cw(const TPoint<T>& A, const TPoint<T>& B){
155     T mul = vmul(A, B);
156     return mul < 0 || abs(mul) <= TPoint<T>::eps;
157 }
158 template<typename T>
159 vector<TPoint<T>> convex_hull(vector<TPoint<T>> pts){

```

```

160     sort(all(pts));
161     pts.erase(unique(all(pts)), pts.end());
162     vector<TPoint<T>> up, down;
163     for (auto p : pts){
164         while (sz(up) > 1 && acw(up.end()[-1] - up.end()[-2], p -
↪ up.end()[-2])) up.pop_back();
165         while (sz(down) > 1 && cw(down.end()[-1] - down.end()[-2],
↪ p - down.end()[-2])) down.pop_back();
166         up.pb(p), down.pb(p);
167     }
168     for (int i = sz(up) - 2; i >= 1; i--) down.pb(up[i]);
169     return down;
170 }
171
172 template<typename T>
173 bool in_triangle(TPoint<T>& P, TPoint<T>& A, TPoint<T>& B,
↪ TPoint<T>& C){
174     if (is_on_seg(P, A, B) || is_on_seg(P, B, C) || is_on_seg(P,
↪ C, A)) return true;
175     return cw(P - A, B - A) == cw(P - B, C - B) &&
176     cw(P - A, B - A) == cw(P - C, A - C);
177 }
178 template<typename T>
179 void prep_convex_poly(vector<TPoint<T>>& pts){
180     rotate(pts.begin(), min_element(all(pts)), pts.end());
181 }
182 // 0 - Outside, 1 - Exclusively Inside, 2 - On the Border
183 template<typename T>
184 int in_convex_poly(TPoint<T>& p, vector<TPoint<T>>& pts){
185     int n = sz(pts);
186     if (!n) return 0;
187     if (n <= 2) return is_on_seg(p, pts[0], pts.back());
188     int l = 1, r = n - 1;
189     while (r - l > 1){
190         int mid = (l + r) / 2;
191         if (acw(pts[mid] - pts[0], p - pts[0])) l = mid;
192         else r = mid;
193     }
194     if (!in_triangle(p, pts[0], pts[l], pts[l + 1])) return 0;
195     if (is_on_seg(p, pts[l], pts[l + 1]) ||
196         is_on_seg(p, pts[0], pts.back()) ||
197         is_on_seg(p, pts[0], pts[l])
198     ) return 2;
199     return 1;
200 }
201 // 0 - Outside, 1 - Exclusively Inside, 2 - On the Border
202 template<typename T>
203 int in_simple_poly(TPoint<T> p, vector<TPoint<T>>& pts){
204     int n = sz(pts);
205     bool res = 0;
206     for (int i = 0; i < n; i++){
207         auto a = pts[i], b = pts[(i + 1) % n];
208         if (is_on_seg(p, a, b)) return 2;
209         if (((a.y > p.y) - (b.y > p.y)) * vmul(b - p, a - p) >
↪ TPoint<T>::eps){
210             res ^= 1;
211         }
212     }
213     return res;
214 }
215 template<typename T>
216 void minkowski_rotate(vector<TPoint<T>>& P){
217     int pos = 0;
218     for (int i = 1; i < sz(P); i++){
219         if (abs(P[i].y - P[pos].y) <= TPoint<T>::eps){
220             if (P[i].x < P[pos].x) pos = i;
221         }
222         else if (P[i].y < P[pos].y) pos = i;
223     }
224     rotate(P.begin(), P.begin() + pos, P.end());
225 }
226 // P and Q are strictly convex, points given in
↪ counterclockwise order
227 template<typename T>
228 vector<TPoint<T>> minkowski_sum(vector<TPoint<T>> P,
↪ vector<TPoint<T>> Q){
229     minkowski_rotate(P);

```

```

230     minkowski_rotate(Q);
231     P.pb(P[0]);
232     Q.pb(Q[0]);
233     vector<TPoint<T>> ans;
234     int i = 0, j = 0;
235     while (i < sz(P) - 1 || j < sz(Q) - 1){
236         ans.pb(P[i] + Q[j]);
237         T curmul;
238         if (i == sz(P) - 1) curmul = -1;
239         else if (j == sz(Q) - 1) curmul = +1;
240         else curmul = vmul(P[i + 1] - P[i], Q[j + 1] - Q[j]);
241         if (abs(curmul) < TPoint<T>::eps || curmul > 0) i++;
242         if (abs(curmul) < TPoint<T>::eps || curmul < 0) j++;
243     }
244     return ans;
245 }
246 using Point = TPoint<ll>; using Line = TLine<ll>; using Ray =
↪ TRay<ll>; const ld PI = acos(-1);

Geometry

Basic stuff

1 using ll = long long;
2 using ld = long double;
3
4 constexpr auto eps = 1e-8;
5 const auto PI = acos(-1);
6 int sgn(ld x) { return (abs(x) <= eps) ? 0 : (x < 0 ? -1 : 1);
↪ }
7
8 struct Point {
9     ld x = 0, y = 0;
10    Point() = default;
11    Point(ld _x, ld _y) : x(_x), y(_y) {}
12    bool operator<(const Point &p) const { return !sgn(p.x - x)
↪ ? sgn(y - p.y) < 0 : x < p.x; }
13    bool operator==(const Point &p) const { return !sgn(p.x - x)
↪ && !sgn(p.y - y); }
14    Point operator+(const Point &p) const { return {x + p.x, y +
↪ p.y}; }
15    Point operator-(const Point &p) const { return {x - p.x, y -
↪ p.y}; }
16    Point operator*(ld a) const { return {x * a, y * a}; }
17    Point operator/(ld a) const { return {x / a, y / a}; }
18    auto operator*(const Point &p) const { return x * p.x + y *
↪ p.y; } // dot
19    auto operator^(const Point &p) const { return x * p.y - y *
↪ p.x; } // cross
20    friend auto &operator>>(istream &i, Point &p) { return i >>
↪ p.x >> p.y; }
21    friend auto &operator<<(ostream &o, Point p) { return o <<
↪ p.x << ' ' << p.y; }
22 };
23
24 struct Line {
25     Point s = {0, 0}, e = {0, 0};
26     Line() = default;
27     Line(Point _s, Point _e) : s(_s), e(_e) {}
28     friend auto &operator>>(istream &i, Line &l) { return i >>
↪ l.s >> l.e; } // ((x1, y1), (x2, y2))
29 };
30
31 struct Segment : Line {
32     using Line::Line;
33 };
34
35 struct Circle {
36     Point o = {0, 0};
37     ld r = 0;
38     Circle() = default;
39     Circle(Point _o, ld _r) : o(_o), r(_r) {}
40 };
41
42 auto dist2(const Point &a) { return a * a; }
43 auto dist2(const Point &a, const Point &b) { return dist2(a -
↪ b); }

```

```

3 auto dist(const Point &a) { return sqrt(dist2(a)); }
4 auto dist(const Point &a, const Point &b) { return
  ↳ sqrt(dist2(a - b)); }
5 auto dist(const Point &a, const Line &l) { return abs((a -
  ↳ l.s) ^ (l.e - l.s)) / dist(l.s, l.e); }
6 auto dist(const Point &p, const Segment &l) {
7   if (l.s == l.e) return dist(p, l.s);
8   auto d = dist2(l.s, l.e), t = min(d, max((ld)0, (p - l.s) *
  ↳ (l.e - l.s)));
9   return dist((p - l.s) * d, (l.e - l.s) * t) / d;
10 }
11 /* Needs is_intersect */
12 auto dist(const Segment &l1, const Segment &l2) {
13   if (is_intersect(l1, l2)) return (ld)0;
14   return min({dist(l1.s, l2), dist(l1.e, l2), dist(l2.s, l1),
  ↳ dist(l2.e, l1)});
15 } */
16
17 Point perp(const Point &p) { return Point(-p.y, p.x); }
18
19 auto rad(const Point &p) { return atan2(p.y, p.x); }

```

## Transformation

```

1 Point project(const Point &p, const Line &l) {
2   return l.s + ((l.e - l.s) * ((l.e - l.s) * (p - l.s))) /
  ↳ dist2(l.e - l.s);
3 }
4
5 Point reflect(const Point &p, const Line &l) {
6   return project(p, l) * 2 - p;
7 }
8
9 Point dilate(const Point &p, ld scale_x = 1, ld scale_y = 1) {
10   ↳ return Point(p.x * scale_x, p.y * scale_y); }
11 Line dilate(const Line &l, ld scale_x = 1, ld scale_y = 1) {
12   ↳ return Line(dilate(l.s, scale_x, scale_y), dilate(l.e,
  ↳ scale_x, scale_y)); }
13 Segment dilate(const Segment &l, ld scale_x = 1, ld scale_y =
14   ↳ 1) { return Segment(dilate(l.s, scale_x, scale_y),
  ↳ dilate(l.e, scale_x, scale_y)); }
15 vector<Point> dilate(const vector<Point> &p, ld scale_x = 1,
16   ↳ ld scale_y = 1) {
17   int n = p.size();
18   vector<Point> res(n);
19   for (int i = 0; i < n; i++)
20     res[i] = dilate(p[i], scale_x, scale_y);
21   return res;
22 }
23
24 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)); }
25 Line rotate(const Line &l, ld a) { return Line(rotate(l.s, a),
  ↳ rotate(l.e, a)); }
26 Segment rotate(const Segment &l, ld a) { return
  ↳ Segment(rotate(l.s, a), rotate(l.e, a)); }
27 Circle rotate(const Circle &c, ld a) { return
  ↳ Circle(rotate(c.o, a), c.r); }
28 vector<Point> rotate(const vector<Point> &p, ld a) {
29   int n = p.size();
30   vector<Point> res(n);
31   for (int i = 0; i < n; i++)
32     res[i] = rotate(p[i], a);
33   return res;
34 }
35
36 Point translate(const Point &p, ld dx = 0, ld dy = 0) { return
  ↳ Point(p.x + dx, p.y + dy); }
37 Line translate(const Line &l, ld dx = 0, ld dy = 0) { return
  ↳ Line(translate(l.s, dx, dy), translate(l.e, dx, dy)); }
38 Segment translate(const Segment &l, ld dx = 0, ld dy = 0) {
39   ↳ return Segment(translate(l.s, dx, dy), translate(l.e, dx,
  ↳ dy)); }
40 Circle translate(const Circle &c, ld dx = 0, ld dy = 0) {
41   ↳ return Circle(translate(c.o, dx, dy), c.r); }
42 vector<Point> translate(const vector<Point> &p, ld dx = 0, ld
43   ↳ dy = 0) {

```

```

37   int n = p.size();
38   vector<Point> res(n);
39   for (int i = 0; i < n; i++)
40     res[i] = translate(p[i], dx, dy);
41   return res;
42 }

```

## Relation

```

1 enum class Relation { SEPARATE, EX_TOUCH, OVERLAP, IN_TOUCH,
  ↳ INSIDE };
2 Relation get_relation(const Circle &a, const Circle &b) {
3   auto c1c2 = dist(a.o, b.o);
4   auto r1r2 = a.r + b.r, diff = abs(a.r - b.r);
5   if (sgn(c1c2 - r1r2) > 0) return Relation::SEPARATE;
6   if (sgn(c1c2 - r1r2) == 0) return Relation::EX_TOUCH;
7   if (sgn(c1c2 - diff) > 0) return Relation::OVERLAP;
8   if (sgn(c1c2 - diff) == 0) return Relation::IN_TOUCH;
9   return Relation::INSIDE;
10 }
11
12 auto get_cos_from_triangle(ld a, ld b, ld c) { return (a * a +
  ↳ b * b - c * c) / (2.0 * a * b); }
13
14 bool on_line(const Line &l, const Point &p) { return !sgn((l.s
  ↳ - p) ^ (l.e - p)); }
15
16 bool on_segment(const Segment &l, const Point &p) {
17   return !sgn((l.s - p) ^ (l.e - p)) && sgn((l.s - p) * (l.e -
  ↳ p)) <= 0;
18 }
19
20 bool on_segment2(const Segment &l, const Point &p) { // assume
21   ↳ p on Line l
22   if (l.s == p || l.e == p) return true;
23   if (min(l.s, l.e) < p && p < max(l.s, l.e)) return true;
24   return false;
25 }
26
27 bool is_parallel(const Line &a, const Line &b) { return
  ↳ !sgn((a.s - a.e) ^ (b.s - b.e)); }
28 bool is_orthogonal(const Line &a, const Line &b) { return
  ↳ !sgn((a.s - a.e) * (b.s - b.e)); }
29
30 int is_intersect(const Segment &a, const Segment &b) {
31   auto d1 = sgn((a.e - a.s) ^ (b.s - a.s)), d2 = sgn((a.e -
  ↳ a.s) ^ (b.e - a.s));
32   auto d3 = sgn((b.e - b.s) ^ (a.s - b.s)), d4 = sgn((b.e -
  ↳ b.s) ^ (a.e - b.s));
33   if (d1 * d2 < 0 && d3 * d4 < 0) return 2; // intersect at
34   ↳ non-end point
35   return (d1 == 0 && sgn((b.s - a.s) * (b.s - a.e)) <= 0) ||
36     (d2 == 0 && sgn((b.e - a.s) * (b.e - a.e)) <= 0) ||
37     (d3 == 0 && sgn((a.s - b.s) * (a.s - b.e)) <= 0) ||
38     (d4 == 0 && sgn((a.e - b.s) * (a.e - b.e)) <= 0);
39 }
40
41 int is_intersect(const Line &a, const Segment &b) {
42   auto d1 = sgn((a.e - a.s) ^ (b.s - a.s)), d2 = sgn((a.e -
  ↳ a.s) ^ (b.e - a.s));
43   if (d1 * d2 < 0) return 2; // intersect at non-end point
44   return d1 == 0 || d2 == 0;
45 }
46
47 Point intersect(const Line &a, const Line &b) {
48   auto u = a.e - a.s, v = b.e - b.s;
49   auto t = ((b.s - a.s) ^ v) / (u ^ v);
50   return a.s + u * t;
51 }
52
53 int is_intersect(const Circle &c, const Line &l) {
54   auto d = dist(c.o, l);
55   return sgn(d - c.r) < 0 ? 2 : !sgn(d - c.r);
56 }
57
58 vector<Point> intersect(const Circle &a, const Circle &b) {
59   auto relation = get_relation(a, b);

```

```

58     if (relation == Relation::INSIDE || relation ==
↳ Relation::SEPARATE) return {};
59     auto vec = b.o - a.o;
60     auto d2 = dist2(vec);
61     auto p = (d2 + a.r * a.r - b.r * b.r) / ((long double)2 *
↳ d2), h2 = a.r * a.r - p * p * d2;
62     auto mid = a.o + vec * p, per = perp(vec) * sqrt(max((long
↳ double)0, h2) / d2);
63     if (relation == Relation::OVERLAP)
64         return {mid + per, mid - per};
65     else
66         return {mid};
67 }
68
69 vector<Point> intersect(const Circle &c, const Line &l) {
70     if (!is_intersect(c, l)) return {};
71     auto v = l.e - l.s, t = v / dist(v);
72     Point a = l.s + t * ((c.o - l.s) * t);
73     auto d = sqrt(max((ld)0, c.r * c.r - dist2(c.o, a)));
74     if (!sgn(d)) return {a};
75     return {a - t * d, a + t * d};
76 }
77
78 int in_poly(const vector<Point> &p, const Point &a) {
79     int cnt = 0, n = (int)p.size();
80     for (int i = 0; i < n; i++) {
81         auto q = p[(i + 1) % n];
82         if (on_segment(Segment(p[i], q), a)) return 1; // on the
↳ edge of the polygon
83         cnt ^= ((a.y < p[i].y) - (a.y < q.y)) * ((p[i] - a) ^ (q -
↳ a)) > 0;
84     }
85     return cnt ? 2 : 0;
86 }
87
88 int is_intersect(const vector<Point> &p, const Line &a) {
89     // 1: touching, >=2: intersect count
90     int cnt = 0, edge_cnt = 0, n = (int)p.size();
91     for (int i = 0; i < n; i++) {
92         auto q = p[(i + 1) % n];
93         if (on_line(a, p[i]) && on_line(a, q)) return -1; //
↳ infinity
94         auto t = is_intersect(a, Segment(p[i], q));
95         (t == 1) && edge_cnt++, (t == 2) && cnt++;
96     }
97     return cnt + edge_cnt / 2;
98 }
99
100 vector<Point> tangent(const Circle &c, const Point &p) {
101     auto d = dist(c.o, p), l = c.r * c.r / d, h = sqrt(c.r * c.r
↳ - l * l);
102     auto v = (p - c.o) / d;
103     return {c.o + v * l + perp(v) * h, c.o + v * l - perp(v) *
↳ h};
104 }
105
106 Circle get_circumscribed(const Point &a, const Point &b, const
↳ Point &c) {
107     Line u((a + b) / 2, ((a + b) / 2) + perp(b - a));
108     Line v((b + c) / 2, ((b + c) / 2) + perp(c - b));
109     auto o = intersect(u, v);
110     return Circle(o, dist(o, a));
111 }
112
113 Circle get_inscribed(const Point &a, const Point &b, const
↳ Point &c) {
114     auto l1 = dist(b - c), l2 = dist(c - a), l3 = dist(a - b);
115     Point o = (a * l1 + b * l2 + c * l3) / (l1 + l2 + l3);
116     return Circle(o, dist(o, Line(a, b)));
117 }
118
119 pair<ld, ld> get_centroid(const vector<Point> &p) {
120     int n = (int)p.size();
121     ld x = 0, y = 0, sum = 0;
122     auto a = p[0], b = p[1];
123     for (int i = 2; i < n; i++) {
124         auto c = p[i];

```

```

125         auto s = area({a, b, c});
126         sum += s;
127         x += s * (a.x + b.x + c.x);
128         y += s * (a.y + b.y + c.y);
129         swap(b, c);
130     }
131     return {x / (3 * sum), y / (3 * sum)};
132 }
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999

```



```

56     r--;
57     if (sgn((q[r].e - q[r].s) ^ (L[i].s - q[r].s)) > 0) q[r]
↪ = L[i];
58 }
59 if (1 < r) p[r - 1] = intersect(q[r - 1], q[r]);
60 }
61 while (1 < r && sgn((q[l].e - q[l].s) ^ (p[r - 1] - q[l].s))
↪ <= 0) r--;
62 if (r - 1 <= 1) return {};
63 p[r] = intersect(q[r], q[l]);
64 return vector<Point>(p.begin() + 1, p.begin() + r + 1);
65 }

```

## Convex

```

1 vector<Point> get_convex(vector<Point> &points, bool
↪ allow_collinear = false) {
2     // strict, no repeat, two pass
3     sort(points.begin(), points.end());
4     points.erase(unique(points.begin(), points.end()),
↪ points.end());
5     vector<Point> L, U;
6     for (auto &t : points) {
7         for (ll sz = L.size(); sz > 1 && (sgn((t - L[sz - 2]) ^
↪ (L[sz - 1] - L[sz - 2])) >= 0);
8             L.pop_back(), sz = L.size()) {
9         }
10        L.push_back(t);
11    }
12    for (auto &t : points) {
13        for (ll sz = U.size(); sz > 1 && (sgn((t - U[sz - 2]) ^
↪ (U[sz - 1] - U[sz - 2])) <= 0);
14            U.pop_back(), sz = U.size()) {
15        }
16        U.push_back(t);
17    }
18    // contain repeats if all collinear, use a set to remove
↪ repeats
19    if (allow_collinear) {
20        for (int i = (int)U.size() - 2; i >= 1; i--)
↪ L.push_back(U[i]);
21    } else {
22        set<Point> st(L.begin(), L.end());
23        for (int i = (int)U.size() - 2; i >= 1; i--) {
24            if (st.count(U[i]) == 0) L.push_back(U[i]),
↪ st.insert(U[i]);
25        }
26    }
27    return L;
28 }
29
30 vector<Point> get_convex2(vector<Point> &points, bool
↪ allow_collinear = false) { // strict, no repeat, one pass
31     nth_element(points.begin(), points.begin(), points.end());
32     sort(points.begin() + 1, points.end(), [&](const Point &a,
↪ const Point &b) {
33         int rad_diff = sgn((a - points[0]) ^ (b - points[0]));
34         return !rad_diff ? (dist2(a - points[0]) < dist2(b -
↪ points[0])) : (rad_diff > 0);
35     });
36     if (allow_collinear) {
37         int i = (int)points.size() - 1;
38         while (i >= 0 && !sgn((points[i] - points[0]) ^ (points[i]
↪ - points.back())) i--;
39         reverse(points.begin() + i + 1, points.end());
40     }
41     vector<Point> hull;
42     for (auto &t : points) {
43         for (ll sz = hull.size();
44             sz > 1 && (sgn((t - hull[sz - 2]) ^ (hull[sz - 1] -
↪ hull[sz - 2])) >= allow_collinear);
45             hull.pop_back(), sz = hull.size()) {
46         }
47        hull.push_back(t);
48    }
49    return hull;
50 }

```

```

51 vector<Point> get_convex_safe(vector<Point> points, bool
↪ allow_collinear = false) {
52     return get_convex(points, allow_collinear);
53 }
54
55 vector<Point> get_convex2_safe(vector<Point> points, bool
↪ allow_collinear = false) {
56     return get_convex2(points, allow_collinear);
57 }
58
59 bool is_convex(const vector<Point> &p, bool allow_collinear =
↪ false) {
60     int n = p.size();
61     int lo = 1, hi = -1;
62     for (int i = 0; i < n; i++) {
63         int cur = sgn((p[(i + 2) % n] - p[(i + 1) % n]) ^ (p[(i +
↪ 1) % n] - p[i]));
64         lo = min(lo, cur); hi = max(hi, cur);
65     }
66     return allow_collinear ? (hi - lo) < 2 : (lo == hi && lo);
67 }
68
69 auto rotating_calipers(const vector<Point> &hull) {
70     // use get_convex2
71     int n = (int)hull.size(); // return the square of longest
↪ dist
72     assert(n > 1);
73     if (n <= 2) return dist2(hull[0], hull[1]);
74     ld res = 0;
75     for (int i = 0, j = 2; i < n; i++) {
76         auto d = hull[i], e = hull[(i + 1) % n];
77         while (area2(d, e, hull[j]) < area2(d, e, hull[(j + 1) %
↪ n])) j = (j + 1) % n;
78         res = max(res, max(dist2(d, hull[j]), dist2(e, hull[j])));
79     }
80     return res;
81 }
82
83 // Find polygon cut to the left of l
84 vector<Point> convex_cut(const vector<Point> &p, const Line
↪ &l) {
85     int n = p.size();
86     vector<Point> cut;
87     for (int i = 0; i < n; i++) {
88         auto a = p[i], b = p[(i + 1) % n];
89         if (sgn((l.e - l.s) ^ (a - l.s)) >= 0)
90             cut.push_back(a);
91         if (sgn((l.e - l.s) ^ (a - l.s)) * sgn((l.e - l.s) ^ (b -
↪ l.s)) == -1)
92             cut.push_back(intersect(Line(a, b), l));
93     }
94     return cut;
95 }
96
97 // Sort by angle in range [0, 2pi)
98 template <class RandomIt>
99 void polar_sort(RandomIt first, RandomIt last, Point origin =
↪ Point(0, 0)) {
100     auto get_quad = [&](const Point& p) {
101         Point diff = p - origin;
102         if (diff.x > 0 && diff.y >= 0) return 1;
103         if (diff.x <= 0 && diff.y > 0) return 2;
104         if (diff.x < 0 && diff.y <= 0) return 3;
105         return 4;
106     };
107     auto polar_cmp = [&](const Point& p1, const Point& p2) {
108         int q1 = get_quad(p1), q2 = get_quad(p2);
109         if (q1 != q2) return q1 < q2;
110         return ((p1 - origin) ^ (p2 - origin)) > 0;
111     };
112     sort(first, last, polar_cmp);
113 }
114

```

## Basic 3D

```
1 using ll = long long;
2 using ld = long double;
3
4 constexpr auto eps = 1e-8;
5 const auto PI = acos(-1);
6 int sgn(ld x) { return (abs(x) <= eps) ? 0 : (x < 0 ? -1 : 1);
  ↪ }
7
8 struct Point3D {
9     ld x = 0, y = 0, z = 0;
10     Point3D() = default;
11     Point3D(ld _x, ld _y, ld _z) : x(_x), y(_y), z(_z) {}
12     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; }
13     bool operator==(const Point3D &p) const { return !sgn(p.x -
  ↪ x) && !sgn(p.y - y) && !sgn(p.z - z); }
14     Point3D operator+(const Point3D &p) const { return {x + p.x,
  ↪ y + p.y, z + p.z}; }
15     Point3D operator-(const Point3D &p) const { return {x - p.x,
  ↪ y - p.y, z - p.z}; }
16     Point3D operator*(ld a) const { return {x * a, y * a, z *
  ↪ a}; }
17     Point3D operator/(ld a) const { return {x / a, y / a, z /
  ↪ a}; }
18     auto operator*(const Point3D &p) const { return x * p.x + y
  ↪ * p.y + z * p.z; } // dot
19     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
20     friend auto &operator>>(istream &i, Point3D &p) { return i
  ↪ >> p.x >> p.y >> p.z; }
21 };
22
23 struct Line3D {
24     Point3D s = {0, 0, 0}, e = {0, 0, 0};
25     Line3D() = default;
26     Line3D(Point3D _s, Point3D _e) : s(_s), e(_e) {}
27 };
28
29 struct Segment3D : Line3D {
30     using Line3D::Line3D;
31 };
32
33 auto dist2(const Point3D &a) { return a * a; }
34 auto dist2(const Point3D &a, const Point3D &b) { return
  ↪ dist2(a - b); }
35 auto dist(const Point3D &a) { return sqrt(dist2(a)); }
36 auto dist(const Point3D &a, const Point3D &b) { return
  ↪ sqrt(dist2(a - b)); }
37 auto dist(const Point3D &a, const Line3D &l) { return dist((a
  ↪ - l.s) ^ (l.e - l.s)) / dist(l.s, l.e); }
38 auto dist(const Point3D &p, const Segment3D &l) {
39     if (l.s == l.e) return dist(p, l.s);
40     auto d = dist2(l.s, l.e), t = min(d, max((ld)0, (p - l.s) *
  ↪ (l.e - l.s)));
41     return dist((p - l.s) * d, (l.e - l.s) * t) / d;
42 }
```

## Miscellaneous

```
1 tuple<int,int,ld> closest_pair(vector<Point> &p) {
2     using Pt = pair<Point,int>;
3     int n = p.size();
4     assert(n > 1);
5     vector<Pt> pts(n), buf;
6     for (int i = 0; i < n; i++) pts[i] = {p[i], i};
7     sort(pts.begin(), pts.end());
8     buf.reserve(n);
9     auto cmp_y = [&](const Pt& p1, const Pt& p2) { return
  ↪ p1.first.y < p2.first.y; };
10     function<tuple<int,int,ld>(int, int)> recurse = [&](int l,
  ↪ int r) -> tuple<int,int,ld> {
11         int i = pts[l].second, j = pts[l + 1].second;
12         ld d = dist(pts[l].first, pts[l + 1].first);
```

```
13         if (r - l < 5) {
14             for (int a = l; a < r; a++) for (int b = a + 1; b < r;
  ↪ b++) {
15                 ld cur = dist(pts[a].first, pts[b].first);
16                 if (cur < d) { i = pts[a].second; j = pts[b].second; d
  ↪ = cur; }
17             }
18             sort(pts.begin() + l, pts.begin() + r, cmp_y);
19         }
20         else {
21             int mid = (l + r) / 2;
22             ld x = pts[mid].first.x;
23             auto [li, lj, ldist] = recurse(l, mid);
24             auto [ri, rj, rdist] = recurse(mid, r);
25             if (ldist < rdist) { i = li; j = lj; d = ldist; }
26             else { i = ri; j = rj; d = rdist; }
27             inplace_merge(pts.begin() + l, pts.begin() + mid,
  ↪ pts.begin() + r, cmp_y);
28             buf.clear();
29             for (int a = l; a < r; a++) {
30                 if (abs(x - pts[a].first.x) >= d) continue;
31                 for (int b = buf.size() - 1; b >= 0; b--) {
32                     if (pts[a].first.y - buf[b].first.y >= d) break;
33                     ld cur = dist(pts[a].first, buf[b].first);
34                     if (cur < d) { i = pts[a].second; j = buf[b].second;
  ↪ d = cur; }
35                 }
36                 buf.push_back(pts[a]);
37             }
38             return {i, j, d};
39         }
40     };
41     return recurse(0, n);
42 }
43
44 Line abc_to_line(ld a, ld b, ld c) {
45     assert(!sgn(a) || !sgn(b));
46     if (a == 0) return Line(Point(0, -c/b), Point(1, -c/b));
47     if (b == 0) return Line(Point(-c/a, 0), Point(-c/a, 1));
48     Point s(0, -c/b), e(1, (-c - a)/b), diff = e - s;
49     return Line(s, s + diff/dist(diff));
50 }
51
52 tuple<ld,ld,ld> line_to_abc(const Line& l) {
53     Point diff = l.e - l.s;
54     return {-diff.y, diff.x, -(diff ^ l.s)};
55 }
```

## Graph Theory

### Max Flow

```
1 struct Edge {
2     int from, to, cap, remain;
3 };
4
5 struct Dinic {
6     int n;
7     vector<Edge> e;
8     vector<vector<int>> g;
9     vector<int> d, cur;
10     Dinic(int _n) : n(_n), g(n), d(n), cur(n) {}
11     void add_edge(int u, int v, int c) {
12         g[u].push_back((int)e.size());
13         e.push_back({u, v, c, c});
14         g[v].push_back((int)e.size());
15         e.push_back({v, u, 0, 0});
16     }
17     ll max_flow(int s, int t) {
18         int inf = 1e9;
19         auto bfs = [&]() {
20             fill(d.begin(), d.end(), inf), fill(cur.begin(),
  ↪ cur.end(), 0);
21             d[s] = 0;
22             vector<int> q{s}, nq;
```



```

23     for (int step = 1; q.size(); swap(q, nq), nq.clear(),
↳ step++) {
24         for (auto& node : q) {
25             for (auto& edge : g[node]) {
26                 int ne = e[edge].to;
27                 if (!e[edge].remain || d[ne] <= step) continue;
28                 d[ne] = step, nq.push_back(ne);
29                 if (ne == t) return true;
30             }
31         }
32     }
33     return false;
34 };
35 function<int(int, int)> find = [&](int node, int limit) {
36     if (node == t || !limit) return limit;
37     int flow = 0;
38     for (int i = cur[node]; i < g[node].size(); i++) {
39         cur[node] = i;
40         int edge = g[node][i], oe = edge ^ 1, ne = e[edge].to;
41         if (!e[edge].remain || d[ne] != d[node] + 1) continue;
42         if (int temp = find(ne, min(limit - flow,
↳ e[edge].remain))) {
43             e[edge].remain -= temp, e[oe].remain += temp, flow
↳ += temp;
44         } else {
45             d[ne] = -1;
46         }
47         if (flow == limit) break;
48     }
49     return flow;
50 };
51 ll res = 0;
52 while (bfs())
53     while (int flow = find(s, inf)) res += flow;
54 return res;
55 }
56 };

```

## • USAGE

```

1 int main() {
2     int n, m, s, t;
3     cin >> n >> m >> s >> t;
4     Dinic dinic(n);
5     for (int i = 0, u, v, c; i < m; i++) {
6         cin >> u >> v >> c;
7         dinic.add_edge(u - 1, v - 1, c);
8     }
9     cout << dinic.max_flow(s - 1, t - 1) << '\n';
10 }

```

## PushRelabel Max-Flow (faster)

```

1 //
↳ https://github.com/kth-competitive-programming/kactl/blob/main/content/flow/push_relabel.cpp
2 #define rep(i, a, b) for (int i = a; i < (b); ++i)
3 #define all(x) begin(x), end(x)
4 #define sz(x) (int)(x).size()
5 typedef long long ll;
6 typedef pair<int, int> pii;
7 typedef vector<int> vi;
8
9 struct PushRelabel {
10     struct Edge {
11         int dest, back;
12         ll f, c;
13     };
14     vector<vector<Edge>> g;
15     vector<ll> ec;
16     vector<Edge*> cur;
17     vector<vi> hs;
18     vi H;
19     PushRelabel(int n) : g(n), ec(n), cur(n), hs(2 * n), H(n) {}
20
21     void addEdge(int s, int t, ll cap, ll rcap = 0) {
22         if (s == t) return;
23         g[s].push_back({t, sz(g[t]), 0, cap});

```

```

24         g[t].push_back({s, sz(g[s]) - 1, 0, rcap});
25     }
26
27     void addFlow(Edge& e, ll f) {
28         Edge& back = g[e.dest][e.back];
29         if (!ec[e.dest] && f) hs[H[e.dest]].push_back(e.dest);
30         e.f += f;
31         e.c -= f;
32         ec[e.dest] += f;
33         back.f -= f;
34         back.c += f;
35         ec[back.dest] -= f;
36     }
37
38     ll calc(int s, int t) {
39         int v = sz(g);
40         H[s] = v;
41         ec[t] = 1;
42         vi co(2 * v);
43         co[0] = v - 1;
44         rep(i, 0, v) cur[i] = g[i].data();
45         for (Edge& e : g[s]) addFlow(e, e.c);
46
47         for (int hi = 0;;) {
48             while (hs[hi].empty())
49                 if (!hi--) return -ec[s];
50             int u = hs[hi].back();
51             hs[hi].pop_back();
52             while (ec[u] > 0) // discharge u
53                 if (cur[u] == g[u].data() + sz(g[u])) {
54                     H[u] = 1e9;
55                     for (Edge& e : g[u])
56                         if (e.c && H[u] > H[e.dest] + 1) H[u] = H[e.dest]
↳ + 1, cur[u] = &e;
57                     if (++co[H[u]], !--co[hi] && hi < v)
58                         rep(i, 0, v) if (hi < H[i] && H[i] < v)--
↳ co[H[i]], H[i] = v + 1;
59                     hi = H[u];
60                 } else if (cur[u]->c && H[u] == H[cur[u]->dest] + 1)
61                     addFlow(*cur[u], min(ec[u], cur[u]->c));
62                 else
63                     ++cur[u];
64             }
65         }
66         bool leftOfMinCut(int a) { return H[a] >= sz(g); }

```

## Min-Cost Max-Flow

```

1 class MCMF {
2 public:
3     static constexpr int INF = 1e9;
4     const int n;
5     vector<tuple<int, int, int>> e;
6     vector<vector<int>> g;
7     vector<int> dis, pre;
8     bool dijkstra(int s, int t) {
9         dis.assign(n, INF);
10        pre.assign(n, -1);
11        priority_queue<pair<int, int>, vector<pair<int, int>>,
↳ greater<>> que;
12        dis[s] = 0;
13        que.emplace(0, s);
14        while (!que.empty()) {
15            auto [d, u] = que.top();
16            que.pop();
17            if (dis[u] != d) continue;
18            for (int i : g[u]) {
19                auto [v, f, c] = e[i];
20                if (c > 0 && dis[v] > d + h[u] - h[v] + f) {
21                    dis[v] = d + h[u] - h[v] + f;
22                    pre[v] = i;
23                    que.emplace(dis[v], v);
24                }
25            }
26        }
27        return dis[t] != INF;
28    }

```

```

29 MCMF(int n) : n(n), g(n) {}
30 void add_edge(int u, int v, int fee, int c) {
31     g[u].push_back(e.size());
32     e.emplace_back(v, fee, c);
33     g[v].push_back(e.size());
34     e.emplace_back(u, -fee, 0);
35 }
36 pair<ll, ll> max_flow(const int s, const int t) {
37     int flow = 0, cost = 0;
38     h.assign(n, 0);
39     while (dijkstra(s, t)) {
40         for (int i = 0; i < n; ++i) h[i] += dis[i];
41         for (int i = t; i != s; i = get<0>(e[pre[i] ^ 1])) {
42             --get<2>(e[pre[i]]);
43             ++get<2>(e[pre[i] ^ 1]);
44         }
45         ++flow;
46         cost += h[t];
47     }
48     return {flow, cost};
49 }
50 };

```

## Max Cost Feasible Flow

```

1 struct Edge {
2     int from, to, cap, remain, cost;
3 };
4
5 struct MCMF {
6     int n;
7     vector<Edge> e;
8     vector<vector<int>> g;
9     vector<ll> d, pre;
10    MCMF(int _n) : n(_n), g(n), d(n), pre(n) {}
11    void add_edge(int u, int v, int c, int w) {
12        g[u].push_back((int)e.size());
13        e.push_back({u, v, c, c, w});
14        g[v].push_back((int)e.size());
15        e.push_back({v, u, 0, 0, -w});
16    }
17    pair<ll, ll> max_flow(int s, int t) {
18        ll inf = 1e18;
19        auto spfa = [&]() {
20            fill(d.begin(), d.end(), -inf); // important!
21            vector<int> f(n), seen(n);
22            d[s] = 0, f[s] = 1e9;
23            vector<int> q{s}, nq;
24            for (; q.size(); swap(q, nq), nq.clear()) {
25                for (auto& node : q) {
26                    seen[node] = false;
27                    for (auto& edge : g[node]) {
28                        int ne = e[edge].to, cost = e[edge].cost;
29                        if (!e[edge].remain || d[ne] >= d[node] + cost)
30                            continue;
31                        d[ne] = d[node] + cost, pre[ne] = edge;
32                        f[ne] = min(e[edge].remain, f[node]);
33                        if (!seen[ne]) seen[ne] = true, nq.push_back(ne);
34                    }
35                }
36                return f[t];
37            };
38            ll flow = 0, cost = 0;
39            while (int temp = spfa()) {
40                if (d[t] < 0) break; // important!
41                flow += temp, cost += temp * d[t];
42                for (ll i = t; i != s; i = e[pre[i]].from) {
43                    e[pre[i]].remain -= temp, e[pre[i] ^ 1].remain +=
44                    temp;
45                }
46            }
47            return {flow, cost};
48        };

```

## Heavy-Light Decomposition

```

1 struct HeavyLight {
2     int root = 0, n = 0;
3     std::vector<int> parent, deep, hson, top, sz, dfn;
4     HeavyLight(std::vector<std::vector<int>> &g, int _root)
5         : root(_root), n((int)g.size()), parent(n), deep(n),
6         hson(n, -1), top(n), sz(n), dfn(n, -1) {
7         int cur = 0;
8         std::function<int(int, int, int)> dfs = [&](int node, int
9         fa, int dep) {
10             deep[node] = dep, sz[node] = 1, parent[node] = fa;
11             for (auto &ne : g[node]) {
12                 if (ne == fa) continue;
13                 sz[node] += dfs(ne, node, dep + 1);
14                 if (hson[node] == -1 || sz[ne] > sz[hson[node]])
15                     hson[node] = ne;
16             }
17             return sz[node];
18         };
19         std::function<void(int, int)> dfs2 = [&](int node, int t)
20         {
21             top[node] = t, dfn[node] = cur++;
22             if (hson[node] == -1) return;
23             dfs2(hson[node], t);
24             for (auto &ne : g[node]) {
25                 if (ne == parent[node] || ne == hson[node]) continue;
26                 dfs2(ne, ne);
27             }
28         };
29         dfs(root, -1, 0), dfs2(root, root);
30     }
31
32     int lca(int x, int y) const {
33         while (top[x] != top[y]) {
34             if (deep[top[x]] < deep[top[y]]) swap(x, y);
35             x = parent[top[x]];
36         }
37         return deep[x] < deep[y] ? x : y;
38     }
39
40     std::vector<std::array<int, 2>> get_dfn_path(int x, int y)
41     {
42         const std::array<std::vector<std::array<int, 2>>, 2> path;
43         bool front = true;
44         while (top[x] != top[y]) {
45             if (deep[top[x]] > deep[top[y]]) swap(x, y), front =
46             !front;
47             path[front].push_back({dfn[top[y]], dfn[y] + 1});
48             y = parent[top[y]];
49         }
50         if (deep[x] > deep[y]) swap(x, y), front = !front;
51         path[front].push_back({dfn[x], dfn[y] + 1});
52         std::reverse(path[1].begin(), path[1].end());
53         for (const auto &[left, right] : path[1])
54             path[0].push_back({right, left});
55         return path[0];
56     }
57
58     Node query_seg(int u, int v, const SegTree &seg) const {
59         auto node = Node();
60         for (const auto &[left, right] : get_dfn_path(u, v)) {
61             if (left > right) {
62                 node = pull(node, rev(seg.query(right, left)));
63             } else {
64                 node = pull(node, seg.query(left, right));
65             }
66         }
67         return node;
68     }
69 };

```

### • USAGE:

```

1 vector<ll> light(n);
2 SegTree heavy(n), form_parent(n);
3 // cin >> x >> y, x--, y--;

```

```

4  int z = lca(x, y);
5  while (x != z) {
6      if (dfn[top[x]] <= dfn[top[z]]) {
7          // [dfn[z], dfn[x]], from heavy
8          heavy.modify(dfn[z], dfn[x], 1);
9          break;
10     }
11     // x -> top[x];
12     heavy.modify(dfn[top[x]], dfn[x], 1);
13     light[parent[top[x]]] += a[top[x]];
14     x = parent[top[x]];
15 }
16 while (y != z) {
17     if (dfn[top[y]] <= dfn[top[z]]) {
18         // [dfn[z], dfn[y]], from heavy
19         form_parent.modify(dfn[z] + 1, dfn[y] + 1, 1);
20         break;
21     }
22     // y -> top[y];
23     form_parent.modify(dfn[top[y]], dfn[y] + 1, 1);
24     y = parent[top[y]];
25 }

```

## General Unweight Graph Matching

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

```

1  struct BlossomMatch {
2      int n;
3      vector<vector<int>> e;
4      BlossomMatch(int _n) : n(_n), e(_n) {}
5      void add_edge(int u, int v) { e[u].push_back(v),
6      ↪ e[v].push_back(u); }
7      vector<int> find_matching() {
8          vector<int> match(n, -1), vis(n), link(n), f(n), dep(n);
9          function<int(int)> find = [&](int x) { return f[x] == x ?
10          ↪ x : (f[x] = find(f[x])); };
11          auto lca = [&](int u, int v) {
12              u = find(u), v = find(v);
13              while (u != v) {
14                  if (dep[u] < dep[v]) swap(u, v);
15                  u = find(link[match[u]]);
16              }
17              return u;
18          };
19          queue<int> que;
20          auto blossom = [&](int u, int v, int p) {
21              while (find(u) != p) {
22                  link[u] = v, v = match[u];
23                  if (vis[v] == 0) vis[v] = 1, que.push(v);
24                  f[u] = f[v] = p, u = link[v];
25              }
26          };
27          // find an augmenting path starting from u and augment (if
28          ↪ exist)
29          auto augment = [&](int node) {
30              while (!que.empty()) que.pop();
31              iota(f.begin(), f.end(), 0);
32              // vis = 0 corresponds to inner vertices, vis = 1
33          ↪ corresponds to outer vertices
34              fill(vis.begin(), vis.end(), -1);
35              que.push(node);
36              vis[node] = 1, dep[node] = 0;
37              while (!que.empty()) {
38                  int u = que.front();
39                  que.pop();
40                  for (auto v : e[u]) {
41                      if (vis[v] == -1) {
42                          vis[v] = 0, link[v] = u, dep[v] = dep[u] + 1;
43                          // found an augmenting path
44                          if (match[v] == -1) {
45                              for (int x = v, y = u, temp; y != -1; x = temp,
46                              ↪ y = x == -1 ? -1 : link[x]) {
47                                  temp = match[y], match[x] = y, match[y] = x;
48                              }
49                              return;
50                          }
51                      }
52                  }
53              }
54          };
55      };
56  };

```

```

46         vis[match[v]] = 1, dep[match[v]] = dep[u] + 2;
47         que.push(match[v]);
48     } else if (vis[v] == 1 && find(v) != find(u)) {
49         // found a blossom
50         int p = lca(u, v);
51         blossom(u, v, p), blossom(v, u, p);
52     }
53 }
54 }
55 };
56 // find a maximal matching greedily (decrease constant)
57 auto greedy = [&]() {
58     for (int u = 0; u < n; ++u) {
59         if (match[u] != -1) continue;
60         for (auto v : e[u]) {
61             if (match[v] == -1) {
62                 match[u] = v, match[v] = u;
63                 break;
64             }
65         }
66     }
67 };
68 greedy();
69 for (int u = 0; u < n; ++u)
70     if (match[u] == -1) augment(u);
71 return match;
72 }
73 };

```

## Maximum Bipartite Matching

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

```

1  struct BipartiteMatch {
2      int l, r;
3      Dinic dinic = Dinic(0);
4      BipartiteMatch(int _l, int _r) : l(_l), r(_r) {
5          dinic = Dinic(l + r + 2);
6          for (int i = 1; i <= l; i++) dinic.add_edge(0, i, 1);
7          for (int i = 1; i <= r; i++) dinic.add_edge(l + i, l + r +
8          ↪ 1, 1);
9      }
10     void add_edge(int u, int v) { dinic.add_edge(u + 1, l + v +
11     ↪ 1, 1); }
12     ll max_matching() { return dinic.max_flow(0, l + r + 1); }
13 };

```

## 2-SAT and Strongly Connected Components

```

1  void scc(vector<vector<int>>& g, int* idx) {
2      int n = g.size(), ct = 0;
3      int out[n];
4      vector<int> ginv[n];
5      memset(out, -1, sizeof out);
6      memset(idx, -1, n * sizeof(int));
7      function<void(int)> dfs = [&](int cur) {
8          out[cur] = INT_MAX;
9          for (int v : g[cur]) {
10             ginv[v].push_back(cur);
11             if (out[v] == -1) dfs(v);
12         }
13         ct++; out[cur] = ct;
14     };
15     vector<int> order;
16     for (int i = 0; i < n; i++) {
17         order.push_back(i);
18         if (out[i] == -1) dfs(i);
19     }
20     sort(order.begin(), order.end(), [&](int& u, int& v) {
21         return out[u] > out[v];
22     });
23     ct = 0;
24     stack<int> s;
25     auto dfs2 = [&](int start) {
26         s.push(start);
27         while (!s.empty()) {

```

```

28     int cur = s.top();
29     s.pop();
30     idx[cur] = ct;
31     for(int v : ginv[cur])
32         if(idx[v] == -1) s.push(v);
33 }
34 };
35 for(int v : order) {
36     if(idx[v] == -1) {
37         dfs2(v);
38         ct++;
39     }
40 }
41 }
42
43 // 0 => impossible, 1 => possible
44 pair<int, vector<int>> sat2(int n, vector<pair<int, int>>&
45     ↪ clauses) {
46     vector<int> ans(n);
47     vector<vector<int>> g(2*n + 1);
48     for(auto [x, y] : clauses) {
49         x = x < 0 ? -x + n : x;
50         y = y < 0 ? -y + n : y;
51         int nx = x <= n ? x + n : x - n;
52         int ny = y <= n ? y + n : y - n;
53         g[nx].push_back(y);
54         g[ny].push_back(x);
55     }
56     int idx[2*n + 1];
57     scc(g, idx);
58     for(int i = 1; i <= n; i++) {
59         if(idx[i] == idx[i + n]) return {0, {}};
60         ans[i - 1] = idx[i + n] < idx[i];
61     }
62     return {1, ans};
63 }

```

## Enumerating Triangles

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

```

1 void enumerate_triangles(vector<pair<int, int>>& edges,
2     ↪ function<void(int, int, int)> f) {
3     int n = 0;
4     for(auto [u, v] : edges) n = max({n, u + 1, v + 1});
5     vector<int> deg(n);
6     vector<int> g[n];
7     for(auto [u, v] : edges) {
8         deg[u]++;
9         deg[v]++;
10    }
11    for(auto [u, v] : edges) {
12        if(u == v) continue;
13        if(deg[u] > deg[v] || (deg[u] == deg[v] && u > v))
14            swap(u, v);
15        g[u].push_back(v);
16    }
17    vector<int> flag(n);
18    for(int i = 0; i < n; i++) {
19        for(int v : g[i]) flag[v] = 1;
20        for(int v : g[i]) for(int u : g[v]) {
21            if(flag[u]) f(i, v, u);
22        }
23        for(int v : g[i]) flag[v] = 0;
24    }
25 }

```

## Tarjan

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

```

1 int cnt = 0, now = 0;
2 vector<ll> dfn(n, -1), low(n), belong(n, -1), stk;
3 function<void(ll, ll)> tarjan = [&](ll node, ll fa) {
4     dfn[node] = low[node] = now++, stk.push_back(node);

```

```

5     for (auto& ne : g[node]) {
6         if (ne == fa) continue;
7         if (dfn[ne] == -1) {
8             tarjan(ne, node);
9             low[node] = min(low[node], low[ne]);
10        } else if (belong[ne] == -1) {
11            low[node] = min(low[node], dfn[ne]);
12        }
13    }
14    if (dfn[node] == low[node]) {
15        while (true) {
16            auto v = stk.back();
17            belong[v] = cnt;
18            stk.pop_back();
19            if (v == node) break;
20        }
21        ++cnt;
22    }
23 };

```

- 2-vertex-connected-component / Block forest

```

1 int cnt = 0, now = 0;
2 vector<vector<ll>> e1(n);
3 vector<ll> dfn(n, -1), low(n), stk;
4 function<void(ll)> tarjan = [&](ll node) {
5     dfn[node] = low[node] = now++, stk.push_back(node);
6     for (auto& ne : g[node]) {
7         if (dfn[ne] == -1) {
8             tarjan(ne);
9             low[node] = min(low[node], low[ne]);
10            if (low[ne] == dfn[ne]) {
11                e1.push_back({});
12                while (true) {
13                    auto x = stk.back();
14                    stk.pop_back();
15                    e1[n + cnt].push_back(x);
16                    // e1[x].push_back(n + cnt); // undirected
17                    if (x == ne) break;
18                }
19                e1[node].push_back(n + cnt);
20                // e1[n + cnt].push_back(node); // undirected
21                cnt++;
22            }
23        } else {
24            low[node] = min(low[node], dfn[ne]);
25        }
26    }
27 };

```

## Kruskal reconstruct tree

```

1 int _n, m;
2 cin >> _n >> m; // _n: # of node, m: # of edge
3 int n = 2 * _n - 1; // root: n-1
4 vector<array<int, 3>> edges(m);
5 for (auto& [w, u, v] : edges) {
6     cin >> u >> v >> w, u--, v--;
7 }
8 sort(edges.begin(), edges.end());
9 vector<int> p(n);
10 iota(p.begin(), p.end(), 0);
11 function<int(int)> find = [&](int x) { return p[x] == x ? x :
12     ↪ (p[x] = find(p[x])); };
13 auto merge = [&](int x, int y) { p[find(x)] = find(y); };
14 vector<vector<int>> g(n);
15 vector<int> val(m);
16 val.reserve(n);
17 for (auto [w, u, v] : edges) {
18     u = find(u), v = find(v);
19     if (u == v) continue;
20     val.push_back(w);
21     int node = (int)val.size() - 1;
22     g[node].push_back(u), g[node].push_back(v);
23     merge(u, node), merge(v, node);
24 }

```

## centroid decomposition

```
1 vector<char> res(n), seen(n), sz(n);
2 function<int(int, int)> get_size = [&](int node, int fa) {
3     sz[node] = 1;
4     for (auto& ne : g[node]) {
5         if (ne == fa || seen[ne]) continue;
6         sz[node] += get_size(ne, node);
7     }
8     return sz[node];
9 };
10 function<int(int, int, int)> find_centroid = [&](int node, int
    ↪ fa, int t) {
11     for (auto& ne : g[node])
12         if (ne != fa && !seen[ne] && sz[ne] > t / 2) return
    ↪ find_centroid(ne, node, t);
13     return node;
14 };
15 function<void(int, char)> solve = [&](int node, char cur) {
16     get_size(node, -1); auto c = find_centroid(node, -1,
    ↪ sz[node]);
17     seen[c] = 1, res[c] = cur;
18     for (auto& ne : g[c]) {
19         if (seen[ne]) continue;
20         solve(ne, char(cur + 1)); // we can pass c here to build
    ↪ tree
21     }
22 };
```

## 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

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

- USAGE: get factorial

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

```
5 auto binom = [&](ll n, ll r) -> Z {
6     if (n < 0 || r < 0 || n < r) return 0;
7     return f[n] * rf[n - r] * rf[r];
8 };
```

## Mod Class

```
1 constexpr ll norm(ll x) { return (x % MOD + MOD) % MOD; }
2 template <typename T>
3 constexpr T power(T a, ll b, T res = 1) {
4     for (; b; b /= 2, (a *= a) %= MOD)
5         if (b & 1) (res *= a) %= MOD;
6     return res;
7 }
8 struct Z {
9     ll x;
10     constexpr Z(ll _x = 0) : x(norm(_x)) {}
11     // auto operator<=>(const Z &) const = default; // cpp20
    ↪ only
12     Z operator-() const { return Z(norm(MOD - x)); }
13     Z inv() const { return power(*this, MOD - 2); }
14     Z &operator*=(const Z &rhs) { return x = x * rhs.x % MOD,
    ↪ *this; }
15     Z &operator+=(const Z &rhs) { return x = norm(x + rhs.x),
    ↪ *this; }
16     Z &operator-=(const Z &rhs) { return x = norm(x - rhs.x),
    ↪ *this; }
17     Z &operator/=(const Z &rhs) { return *this *= rhs.inv(); }
18     Z &operator%=(const ll &rhs) { return x %= rhs, *this; }
19     friend Z operator*(Z lhs, const Z &rhs) { return lhs * rhs;
    ↪ }
20     friend Z operator+(Z lhs, const Z &rhs) { return lhs += rhs;
21     friend Z operator-(Z lhs, const Z &rhs) { return lhs -= rhs;
    ↪ }
22     friend Z operator/(Z lhs, const Z &rhs) { return lhs /= rhs;
    ↪ }
23     friend Z operator%(Z lhs, const ll &rhs) { return lhs %=
    ↪ rhs; }
24     friend auto &operator>>(istream &i, Z &z) { return i >> z.x;
    ↪ }
25     friend auto &operator<<(ostream &o, const Z &z) { return o
    ↪ << z.x; }
26 };
```

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

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

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

```
1 constexpr int norm(int x) {
2     if (x < 0) x += MOD;
3     if (x >= MOD) x -= MOD;
4     return x;
5 }
```

## Combinatorics

```
1 const int NMAX = 3000010;
2 ll factorialcompute[NMAX];
3 ll invfactorialcompute[NMAX];
4 ll binpow(ll a, ll pow, ll mod) {
5     if (pow <= 0)
6         return 1;
7     ll p = binpow(a, pow / 2, mod) % mod;
8     p = (p * p) % mod;
9
10    return (pow % 2 == 0) ? p : (a * p) % mod;
11 }
12 ll inverse(ll a, ll mod) {
13     if (a == 1) return 1;
14     return binpow(a, mod-2, mod);
15 }
```

```

16 ll combination(int a, int b, ll mod) {
17     if (a < b) return 0;
18     ll cur = factorialcompute[a];
19     cur *= invfactorialcompute[b];
20     cur %= mod;
21     cur *= invfactorialcompute[a - b];
22     cur %= mod;
23     return cur;
24 }
25 void precomputeFactorial() {
26     factorialcompute[0] = 1;
27     invfactorialcompute[0] = 1;
28     for(int i = 1; i < NMAX; i++) {
29         factorialcompute[i] = factorialcompute[i-1] * i;
30         factorialcompute[i] %= MOD;
31     }
32     invfactorialcompute[NMAX-1] =
33     ↪ inverse(factorialcompute[NMAX-1], MOD);
34     for(int i = NMAX-2; i > -1; i--) {
35         invfactorialcompute[i] = invfactorialcompute[i+1] *
36         ↪ (i+1);
37         invfactorialcompute[i] %= MOD;
38     }
39 }

```

## exgcd

```

1 array<ll, 3> exgcd(ll a, ll b) {
2     if(!b) return {a, 1, 0};
3
4     auto [g, x, y] = exgcd(b, a%b);
5     return {g, y, x - a/b*y};
6 }

```

## Factor/primes

```

1 vector<int> primes(0);
2 void gen_primes(int a) {
3     vector<bool> is_prime(a+1, true);
4     is_prime[0] = is_prime[1] = false;
5     for(int i = 2; i * i <= a; i++) {
6         if(is_prime[i]) {
7             for(int j = i * i; j <= a; j += i) is_prime[j] =
8             ↪ false;
9         }
10        for(int i = 0; i <= a; i++) {
11            if(is_prime[i]) primes.push_back(i);
12        }
13    }
14    vector<ll> gen_factors_prime(ll a){
15        vector<ll> factors;
16        factors.push_back(1);
17        if(a == 1) return factors;
18        for(int z : primes) {
19            if(z * z > a) {
20                z = a;
21            }
22            int cnt = 0;
23            while(a % z == 0) {
24                cnt++;
25                a /= z;
26            }
27            ll num = z;
28            int size = factors.size();
29            for(int i = 1; i <= cnt; i++) {
30                for(int j = 0; j < size; j++) {
31                    factors.push_back(num * factors[j]);
32                }
33                num *= z;
34            }
35            if (a == 1) break;
36        }
37        return factors;
38    }
39    vector<int> get_primes(int num) {

```

```

40        vector<int> curPrime;
41        if(num == 1) return curPrime;
42        for(int z : primes) {
43            if(z * z > num) {
44                curPrime.push_back(num);
45                break;
46            }
47            if(num % z == 0) {
48                curPrime.push_back(z);
49                while(num % z == 0) num /= z;
50            }
51            if(num == 1) break;
52        }
53        return curPrime;
54    }

```

## 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  $\text{Cancer}(x, y)$ , it doesn't fix the input if  $y > p$

```

1 struct Cancer {
2     ll x; ll y;
3     Cancer() : Cancer(0, 1) {}
4     Cancer(ll _y) {
5         x = 0, y = _y;
6         while(y % MOD == 0) {
7             y /= MOD;
8             x++;
9         }
10    }
11    Cancer(ll _x, ll _y) : x(_x), y(_y) {}
12    Cancer inv() { return Cancer(-x, power(y, MOD - 2)); }
13    Cancer operator*(const Cancer &c) { return Cancer(x + c.x,
14    ↪ (y * c.y) % MOD); }
15    Cancer operator*(ll m) {
16        ll p = 0;
17        while(m % MOD == 0) {
18            m /= MOD;
19            p++;
20        }
21        return Cancer(x + p, (m * y) % MOD);
22    }
23    friend auto &operator<<(ostream &o, Cancer c) { return o <<
24    ↪ c.x << ' ' << c.y; }
25 };

```

## NTT, FFT, FWT

- ntt

```

1 void ntt(vector<Z>& a, int f) {
2     int n = int(a.size());
3     vector<Z> w(n);
4     vector<int> rev(n);
5     for (int i = 0; i < n; i++) rev[i] = (rev[i / 2] / 2) | ((i
6     ↪ & 1) * (n / 2));
7     for (int i = 0; i < n; i++) {
8         if (i < rev[i]) swap(a[i], a[rev[i]]);
9     }
10    Z wn = power(f ? (MOD + 1) / 3 : 3, (MOD - 1) / n);
11    w[0] = 1;
12    for (int i = 1; i < n; i++) w[i] = w[i - 1] * wn;
13    for (int mid = 1; mid < n; mid *= 2) {
14        for (int i = 0; i < n; i += 2 * mid) {
15            for (int j = 0; j < mid; j++) {
16                Z x = a[i + j], y = a[i + j + mid] * w[n / (2 * mid) *
17                ↪ j];
18                a[i + j] = x + y, a[i + j + mid] = x - y;
19            }
20        }
21    }
22 }

```



```

20 if (f) {
21     Z iv = power(Z(n), MOD - 2);
22     for (auto& x : a) x *= iv;
23 }
24 }

```

### • USAGE: Polynomial multiplication

```

1 vector<Z> mul(vector<Z> a, vector<Z> b) {
2     int n = 1, m = (int)a.size() + (int)b.size() - 1;
3     while (n < m) n *= 2;
4     a.resize(n), b.resize(n);
5     ntt(a, 0), ntt(b, 0);
6     for (int i = 0; i < n; i++) a[i] *= b[i];
7     ntt(a, 1);
8     a.resize(m);
9     return a;
10 }

```

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

```

1 const double PI = acos(-1);
2 auto mul = [&](const vector<double>& aa, const vector<double>&
↳ bb) {
3     int n = (int)aa.size(), m = (int)bb.size(), bit = 1;
4     while ((1 << bit) < n + m - 1) bit++;
5     int len = 1 << bit;
6     vector<complex<double>> a(len), b(len);
7     vector<int> rev(len);
8     for (int i = 0; i < n; i++) a[i].real(aa[i]);
9     for (int i = 0; i < m; i++) b[i].real(bb[i]);
10    for (int i = 0; i < len; i++) rev[i] = (rev[i >> 1] >> 1) |
↳ ((i & 1) << (bit - 1));
11    auto fft = [&](vector<complex<double>>& p, int inv) {
12        for (int i = 0; i < len; i++)
13            if (i < rev[i]) swap(p[i], p[rev[i]]);
14        for (int mid = 1; mid < len; mid *= 2) {
15            auto w1 = complex<double>(cos(PI / mid), (inv ? -1 : 1)
↳ sin(PI / mid));
16            for (int i = 0; i < len; i += mid * 2) {
17                auto wk = complex<double>(1, 0);
18                for (int j = 0; j < mid; j++, wk = wk * w1) {
19                    auto x = p[i + j], y = wk * p[i + j + mid];
20                    p[i + j] = x + y, p[i + j + mid] = x - y;
21                }
22            }
23        }
24        if (inv == 1) {
25            for (int i = 0; i < len; i++) p[i].real(p[i].real() /
↳ len);
26        }
27    };
28    fft(a, 0), fft(b, 0);
29    for (int i = 0; i < len; i++) a[i] = a[i] * b[i];
30    fft(a, 1);
31    a.resize(n + m - 1);
32    vector<double> res(n + m - 1);
33    for (int i = 0; i < n + m - 1; i++) res[i] = a[i].real();
34    return res;
35 };

```

## Polynomial Class

```

1 using ll = long long;
2 constexpr ll MOD = 998244353;
3
4 ll norm(ll x) { return (x % MOD + MOD) % MOD; }
5 template <class T>
6 T power(T a, ll b, T res = 1) {
7     for (; b; b /= 2, (a *= a) %= MOD)
8         if (b & 1) (res *= a) %= MOD;
9     return res;
10 }
11
12 struct Z {
13     ll x;

```

```

14     Z(ll _x = 0) : x(norm(_x)) {}
15     // auto operator<=>(const Z &) const = default;
16     Z operator-(const Z &) const { return Z(norm(MOD - x)); }
17     Z inv() const { return power(*this, MOD - 2); }
18     Z &operator*=(const Z &rhs) { return x = x * rhs.x % MOD,
↳ *this; }
19     Z &operator+=(const Z &rhs) { return x = norm(x + rhs.x),
↳ *this; }
20     Z &operator-=(const Z &rhs) { return x = norm(x - rhs.x),
↳ *this; }
21     Z &operator/=(const Z &rhs) { return *this *= rhs.inv(); }
22     Z &operator%=(const ll &rhs) { return x %= rhs, *this; }
23     friend Z operator*(Z lhs, const Z &rhs) { return lhs * rhs;
↳ }
24     friend Z operator+(Z lhs, const Z &rhs) { return lhs + rhs;
↳ }
25     friend Z operator-(Z lhs, const Z &rhs) { return lhs - rhs;
↳ }
26     friend Z operator/(Z lhs, const Z &rhs) { return lhs / rhs;
↳ }
27     friend Z operator%(Z lhs, const ll &rhs) { return lhs %=
↳ rhs; }
28     friend auto &operator>>(istream &i, Z &z) { return i >> z.x;
↳ }
29     friend auto &operator<<(ostream &o, const Z &z) { return o
↳ << z.x; }
30 };
31
32 void ntt(vector<Z> &a, int f) {
33     int n = (int)a.size();
34     vector<Z> w(n);
35     vector<int> rev(n);
36     for (int i = 0; i < n; i++) rev[i] = (rev[i / 2] / 2) | ((i
↳ & 1) * (n / 2));
37     for (int i = 0; i < n; i++)
38         if (i < rev[i]) swap(a[i], a[rev[i]]);
39     Z wn = power(ll(f ? (MOD + 1) / 3 : 3), (MOD - 1) / n);
40     w[0] = 1;
41     for (int i = 1; i < n; i++) w[i] = w[i - 1] * wn;
42     for (int mid = 1; mid < n; mid *= 2) {
43         for (int i = 0; i < n; i += 2 * mid) {
44             for (int j = 0; j < mid; j++) {
45                 Z x = a[i + j], y = a[i + j + mid] * w[n / (2 * mid) *
↳ j];
46                 a[i + j] = x + y, a[i + j + mid] = x - y;
47             }
48         }
49     }
50     if (f) {
51         Z iv = power(Z(n), MOD - 2);
52         for (int i = 0; i < n; i++) a[i] *= iv;
53     }
54 }
55
56 struct Poly {
57     vector<Z> a;
58     Poly() {}
59     Poly(const vector<Z> &a) : a(a) {}
60     int size() const { return (int)a.size(); }
61     void resize(int n) { a.resize(n); }
62     Z operator[](int idx) const {
63         if (idx < 0 || idx >= size()) return 0;
64         return a[idx];
65     }
66     Z &operator[](int idx) { return a[idx]; }
67     Poly mulxk(int k) const {
68         auto b = a;
69         b.insert(b.begin(), k, 0);
70         return Poly(b);
71     }
72     Poly modxk(int k) const { return Poly(vector<Z>(a.begin(),
↳ a.begin() + min(k, size()))); }
73     Poly divxk(int k) const {
74         if (size() <= k) return Poly();
75         return Poly(vector<Z>(a.begin() + k, a.end()));
76     }
77     friend Poly operator+(const Poly &a, const Poly &b) {

```

```

78     vector<Z> res(max(a.size(), b.size()));
79     for (int i = 0; i < (int)res.size(); i++) res[i] = a[i] +
↪ b[i];
80     return Poly(res);
81 }
82 friend Poly operator-(const Poly &a, const Poly &b) {
83     vector<Z> res(max(a.size(), b.size()));
84     for (int i = 0; i < (int)res.size(); i++) res[i] = a[i] -
↪ b[i];
85     return Poly(res);
86 }
87 friend Poly operator*(Poly a, Poly b) {
88     if (a.size() == 0 || b.size() == 0) return Poly();
89     int n = 1, m = (int)a.size() + (int)b.size() - 1;
90     while (n < m) n *= 2;
91     a.resize(n), b.resize(n);
92     ntt(a.a, 0), ntt(b.a, 0);
93     for (int i = 0; i < n; i++) a[i] *= b[i];
94     ntt(a.a, 1);
95     a.resize(m);
96     return a;
97 }
98 friend Poly operator*(Z a, Poly b) {
99     for (int i = 0; i < (int)b.size(); i++) b[i] *= a;
100     return b;
101 }
102 friend Poly operator*(Poly a, Z b) {
103     for (int i = 0; i < (int)a.size(); i++) a[i] *= b;
104     return a;
105 }
106 Poly &operator+=(Poly b) { return (*this) = (*this) + b; }
107 Poly &operator-=(Poly b) { return (*this) = (*this) - b; }
108 Poly &operator*=(Poly b) { return (*this) = (*this) * b; }
109 Poly deriv() const {
110     if (a.empty()) return Poly();
111     vector<Z> res(size() - 1);
112     for (int i = 0; i < size() - 1; ++i) res[i] = (i + 1) *
↪ a[i + 1];
113     return Poly(res);
114 }
115 Poly integr() const {
116     vector<Z> res(size() + 1);
117     for (int i = 0; i < size(); ++i) res[i + 1] = a[i] / (i +
↪ 1);
118     return Poly(res);
119 }
120 Poly inv(int m) const {
121     Poly x({a[0].inv()});
122     int k = 1;
123     while (k < m) {
124         k *= 2;
125         x = (x * (Poly({2}) - modxx(k) * x)).modxx(k);
126     }
127     return x.modxx(m);
128 }
129 Poly log(int m) const { return (deriv() *
↪ inv(m)).integr().modxx(m); }
130 Poly exp(int m) const {
131     Poly x({1});
132     int k = 1;
133     while (k < m) {
134         k *= 2;
135         x = (x * (Poly({1}) - x.log(k) + modxx(k))).modxx(k);
136     }
137     return x.modxx(m);
138 }
139 Poly pow(int k, int m) const {
140     int i = 0;
141     while (i < size() && a[i].x == 0) i++;
142     if (i == size() || 1LL * i * k >= m) {
143         return Poly(vector<Z>(m));
144     }
145     Z v = a[i];
146     auto f = divxx(i) * v.inv();
147     return (f.log(m - i * k) * k).exp(m - i * k).mulxx(i * k)
↪ * power(v, k);
148 }

```

```

149 Poly sqrt(int m) const {
150     Poly x({1});
151     int k = 1;
152     while (k < m) {
153         k *= 2;
154         x = (x + (modxx(k) * x.inv(k)).modxx(k)) * ((MOD + 1) /
↪ 2);
155     }
156     return x.modxx(m);
157 }
158 Poly mult(Poly b) const {
159     if (b.size() == 0) return Poly();
160     int n = b.size();
161     reverse(b.a.begin(), b.a.end());
162     return ((*this) * b).divxx(n - 1);
163 }
164 Poly divmod(Poly b) const {
165     auto n = size(), m = b.size();
166     auto t = *this;
167     reverse(t.a.begin(), t.a.end());
168     reverse(b.a.begin(), b.a.end());
169     Poly res = (t * b.inv(n)).modxx(n - m + 1);
170     reverse(res.a.begin(), res.a.end());
171     return res;
172 }
173 vector<Z> eval(vector<Z> x) const {
174     if (size() == 0) return vector<Z>(x.size(), 0);
175     const int n = max(int(x.size()), size());
176     vector<Poly> q(4 * n);
177     vector<Z> ans(x.size());
178     x.resize(n);
179     function<void(int, int, int)> build = [&](int p, int l,
↪ int r) {
180         if (r - l == 1) {
181             q[p] = Poly({1, -x[l]});
182         } else {
183             int m = (l + r) / 2;
184             build(2 * p, l, m), build(2 * p + 1, m, r);
185             q[p] = q[2 * p] * q[2 * p + 1];
186         }
187     };
188     build(1, 0, n);
189     auto work = [&](auto self, int p, int l, int r, const Poly
↪ &num) -> void {
190         if (r - l == 1) {
191             if (l < int(ans.size())) ans[l] = num[0];
192         } else {
193             int m = (l + r) / 2;
194             self(self, 2 * p, l, m, num.mulT(q[2 * p + 1]).modxx(m
↪ - 1));
195             self(self, 2 * p + 1, m, r, num.mulT(q[2 * p]).modxx(r
↪ - m));
196         }
197     };
198     work(work, 1, 0, n, mulT(q[1].inv(n)));
199     return ans;
200 }
201 };

```

## Sieve

### • linear sieve

```

1 vector<int> min_primes(MAX_N), primes;
2 primes.reserve(1e5);
3 for (int i = 2; i < MAX_N; i++) {
4     if (!min_primes[i]) min_primes[i] = i, primes.push_back(i);
5     for (auto& p : primes) {
6         if (p * i >= MAX_N) break;
7         min_primes[p * i] = p;
8         if (i % p == 0) break;
9     }
10 }

```

### • mobius function

```

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

```

```

2 mu[1] = 1, primes.reserve(1e5);
3 for (int i = 2; i < MAX_N; i++) {
4     if (min_p[i] == 0) {
5         min_p[i] = i;
6         primes.push_back(i);
7         mu[i] = -1;
8     }
9     for (auto p : primes) {
10        if (i * p >= MAX_N) break;
11        min_p[i * p] = p;
12        if (i % p == 0) {
13            mu[i * p] = 0;
14            break;
15        }
16        mu[i * p] = -mu[i];
17    }
18 }

```

- Euler's totient function

```

1 vector<int> min_p(MAX_N), phi(MAX_N), primes;
2 phi[1] = 1, primes.reserve(1e5);
3 for (int i = 2; i < MAX_N; i++) {
4     if (min_p[i] == 0) {
5         min_p[i] = i;
6         primes.push_back(i);
7         phi[i] = i - 1;
8     }
9     for (auto p : primes) {
10        if (i * p >= MAX_N) break;
11        min_p[i * p] = p;
12        if (i % p == 0) {
13            phi[i * p] = phi[i] * p;
14            break;
15        }
16        phi[i * p] = phi[i] * phi[p];
17    }
18 }

```

## Gaussian Elimination

```

1 bool is_0(Z v) { return v.x == 0; }
2 Z abs(Z v) { return v; }
3 bool is_0(double v) { return abs(v) < 1e-9; }
4
5 // 1 => unique solution, 0 => no solution, -1 => multiple
6 // solutions
7 template <typename T>
8 int gaussian_elimination(vector<vector<T>> &a, int limit) {
9     if (a.empty() || a[0].empty()) return -1;
10    int h = (int)a.size(), w = (int)a[0].size(), r = 0;
11    for (int c = 0; c < limit; c++) {
12        int id = -1;
13        for (int i = r; i < h; i++) {
14            if (!is_0(a[i][c]) && (id == -1 || abs(a[id][c]) <
15            abs(a[i][c]))) {
16                id = i;
17            }
18        }
19        if (id == -1) continue;
20        if (id > r) {
21            swap(a[r], a[id]);
22            for (int j = c; j < w; j++) a[id][j] = -a[id][j];
23        }
24        vector<int> nonzero;
25        for (int j = c; j < w; j++) {
26            if (!is_0(a[r][j])) nonzero.push_back(j);
27        }
28        T inv_a = 1 / a[r][c];
29        for (int i = r + 1; i < h; i++) {
30            if (is_0(a[i][c])) continue;
31            T coeff = -a[i][c] * inv_a;
32            for (int j : nonzero) a[i][j] += coeff * a[r][j];
33            ++r;
34        }
35        for (int row = h - 1; row >= 0; row--) {

```

```

35        for (int c = 0; c < limit; c++) {
36            if (!is_0(a[row][c])) {
37                T inv_a = 1 / a[row][c];
38                for (int i = row - 1; i >= 0; i--) {
39                    if (is_0(a[i][c])) continue;
40                    T coeff = -a[i][c] * inv_a;
41                    for (int j = c; j < w; j++) a[i][j] += coeff *
42                    a[row][j];
43                }
44                break;
45            }
46        } // not-free variables: only it on its line
47        for (int i = r; i < h; i++) if (!is_0(a[i][limit])) return 0;
48        return (r == limit) ? 1 : -1;
49    }
50
51 template <typename T>
52 pair<int, vector<T>> solve_linear(vector<vector<T>> a, const
53 vector<T> &b, int w) {
54     int h = (int)a.size();
55     for (int i = 0; i < h; i++) a[i].push_back(b[i]);
56     int sol = gaussian_elimination(a, w);
57     if (!sol) return {0, vector<T>()};
58     vector<T> x(w, 0);
59     for (int i = 0; i < h; i++) {
60         for (int j = 0; j < w; j++) {
61             if (!is_0(a[i][j])) {
62                 x[j] = a[i][w] / a[i][j];
63                 break;
64             }
65         }
66     }
67     return {sol, x};

```

## is\_prime

- (Miller–Rabin primality test)

```

1 int128 power(int128 a, int128 b, int128 MOD = 1, int128 res = 1) {
2     for (; b /= 2, (a *= a) %= MOD)
3         if (b & 1) (res *= a) %= MOD;
4     return res;
5 }
6
7 bool is_prime(ll n) {
8     if (n < 2) return false;
9     static constexpr int A[] = {2, 3, 5, 7, 11, 13, 17, 19, 23};
10    int s = __builtin_ctzll(n - 1);
11    ll d = (n - 1) >> s;
12    for (auto a : A) {
13        if (a == n) return true;
14        ll x = (ll)power(a, d, n);
15        if (x == 1 || x == n - 1) continue;
16        bool ok = false;
17        for (int i = 0; i < s - 1; ++i) {
18            x = ll((i128)x * x % n); // potential overflow!
19            if (x == n - 1) {
20                ok = true;
21                break;
22            }
23        }
24        if (!ok) return false;
25    }
26    return true;
27 }
28
29 ll pollard_rho(ll x) {
30     ll s = 0, t = 0, c = rng() % (x - 1) + 1;
31     ll stp = 0, goal = 1, val = 1;
32     for (goal = 1;; goal *= 2, s = t, val = 1) {
33         for (stp = 1; stp <= goal; ++stp) {
34             t = ll(((i128)t * t + c) % x);
35             val = ll(((i128)val * abs(t - s) % x);
36             if ((stp % 127) == 0) {
37                 ll d = gcd(val, x);

```

```

10         if (d > 1) return d;
11     }
12 }
13 ll d = gcd(val, x);
14 if (d > 1) return d;
15 }
16 }
17
18 ll get_max_factor(ll _x) {
19     ll max_factor = 0;
20     function<void(ll)> fac = [&](ll x) {
21         if (x <= max_factor || x < 2) return;
22         if (is_prime(x)) {
23             max_factor = max_factor > x ? max_factor : x;
24             return;
25         }
26         ll p = x;
27         while (p >= x) p = pollard_rho(x);
28         while ((x % p) == 0) x /= p;
29         fac(x), fac(p);
30     };
31     fac(_x);
32     return max_factor;
33 }

```

## Radix Sort

```

1 struct identity {
2     template<typename T>
3     T operator()(const T &x) const {
4         return x;
5     }
6 };
7 // A stable sort that sorts in passes of `bits_per_pass` bits
8 // at a time.
9 template<typename T, typename T_extract_key = identity>
10 void radix_sort(vector<T> &data, int bits_per_pass = 10, const
11     T_extract_key &extract_key = identity()) {
12     if (int64_t(data.size()) * (64 -
13     __builtin_clzll(data.size())) < 2 * (1 << bits_per_pass))
14     {
15         stable_sort(data.begin(), data.end(), [&](const T &a,
16     const T &b) {
17             return extract_key(a) < extract_key(b);
18         });
19         return;
20     }
21
22     using T_key = decltype(extract_key(data.front()));
23     T_key minimum = numeric_limits<T_key>::max();
24     for (T &x : data)
25         minimum = min(minimum, extract_key(x));
26
27     int max_bits = 0;
28     for (T &x : data) {
29         T_key key = extract_key(x);
30         max_bits = max(max_bits, key == minimum ? 0 : 64 -
31     __builtin_clzll(key - minimum));
32     }
33
34     int passes = max((max_bits + bits_per_pass / 2) /
35     bits_per_pass, 1);
36     if (64 - __builtin_clzll(data.size()) <= 1.5 * passes) {
37         stable_sort(data.begin(), data.end(), [&](const T &a,
38     const T &b) {
39             return extract_key(a) < extract_key(b);
40         });
41         return;
42     }
43
44     vector<T> buffer(data.size());
45     vector<int> counts;
46     int bits_so_far = 0;
47
48     for (int p = 0; p < passes; p++) {
49         int bits = (max_bits + p) / passes;
50         counts.assign(1 << bits, 0);
51         for (T &x : data) {
52             T_key key = T_key(extract_key(x) - minimum);

```

```

43         counts[(key >> bits_so_far) & ((1 << bits) -
44     1)]++;
45     }
46     int count_sum = 0;
47     for (int &count : counts) {
48         int current = count;
49         count = count_sum;
50         count_sum += current;
51     }
52     for (T &x : data) {
53         T_key key = T_key(extract_key(x) - minimum);
54         int key_section = int((key >> bits_so_far) & ((1
55     << bits) - 1));
56         buffer[counts[key_section]++] = x;
57     }
58     swap(data, buffer);
59     bits_so_far += bits;
60 }

```

## • USAGE

```

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

```

## lucas

```

1 ll lucas(ll n, ll m, ll p) {
2     if (m == 0) return 1;
3     return (binom(n % p, m % p, p) * lucas(n / p, m / p, p)) %
4     p;
5 }

```

## parity of n choose m

```

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

```

## sosdp

### subset sum

```

1 auto f = a;
2 for (int i = 0; i < SZ; i++) {
3     for (int mask = 0; mask < (1 << SZ); mask++) {
4         if (mask & (1 << i)) f[mask] += f[mask ^ (1 << i)];
5     }
6 }

```

## prf

```

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

```

## String

## AC Automaton

```

1 struct AC_automaton {
2     int sz = 26;
3     vector<vector<int>> e = {vector<int>(sz)}; // vector is
4     // faster than unordered_map
5     vector<int> fail = {0}, end = {0};
6     vector<int> fast = {0}; // closest end
7
8     int insert(string& s) {
9         int p = 0;
10        for (auto c : s) {
11            c -= 'a';
12            if (!e[p][c]) {
13                e.emplace_back(sz);
14                fail.emplace_back();
15                end.emplace_back();
16                fast.emplace_back();
17                e[p][c] = (int)e.size() - 1;

```

```

17     }
18     p = e[p][c];
19 }
20 end[p] += 1;
21 return p;
22 }
23
24 void build() {
25     queue<int> q;
26     for (int i = 0; i < sz; i++)
27         if (e[0][i]) q.push(e[0][i]);
28     while (!q.empty()) {
29         int p = q.front();
30         q.pop();
31         fast[p] = end[p] ? p : fast[fail[p]];
32         for (int i = 0; i < sz; i++) {
33             if (e[p][i]) {
34                 fail[e[p][i]] = e[fail[p]][i];
35                 q.push(e[p][i]);
36             } else {
37                 e[p][i] = e[fail[p]][i];
38             }
39         }
40     }
41 }
42 };

```

## KMP

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

```

1 vector<int> get_next(vector<int> &pat) {
2     int m = (int)pat.size();
3     vector<int> nex(m);
4     for (int i = 1, j = 0; i < m; i++) {
5         while (j && pat[j] != pat[i]) j = nex[j - 1];
6         if (pat[j] == pat[i]) j++;
7         nex[i] = j;
8     }
9     return nex;
10 }

```

- kmp match for txt and pat

```

1 auto nex = get_next(pat);
2 for (int i = 0, j = 0; i < n; i++) {
3     while (j && pat[j] != txt[i]) j = nex[j - 1];
4     if (pat[j] == txt[i]) j++;
5     if (j == m) {
6         // do what you want with the match
7         // start index is `i - m + 1`
8         j = nex[j - 1];
9     }
10 }

```

## Z function

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

```

1 vector<int> z_function(string s) {
2     int n = (int)s.size();
3     vector<int> z(n);
4     for (int i = 1, l = 0, r = 0; i < n; ++i) {
5         if (i <= r) z[i] = min(r - i + 1, z[i - l]);
6         while (i + z[i] < n && s[z[i]] == s[i + z[i]]) ++z[i];
7         if (i + z[i] - 1 > r) l = i, r = i + z[i] - 1;
8     }
9     return z;
10 }

```

## General Suffix Automaton

```

1 constexpr int SZ = 26;
2
3 struct GSAM {

```

```

4     vector<vector<int>> e = {vector<int>(SZ)}; // the labeled
    ↪ edges from node i
5     vector<int> parent = {-1}; // the parent of
    ↪ i
6     vector<int> length = {0}; // the length of
    ↪ the longest string
7
8     GSAM(int n) { e.reserve(2 * n), parent.reserve(2 * n),
    ↪ length.reserve(2 * n); };
9     int extend(int c, int p) { // character, last
10         bool f = true; // if already exist
11         int r = 0; // potential new node
12         if (!e[p][c]) { // only extend when not exist
13             f = false;
14             e.push_back(vector<int>(SZ));
15             parent.push_back(0);
16             length.push_back(length[p] + 1);
17             r = (int)e.size() - 1;
18             for (; ~p && !e[p][c]; p = parent[p]) e[p][c] = r; //
    ↪ update parents
19         }
20         if (f || ~p) {
21             int q = e[p][c];
22             if (length[q] == length[p] + 1) {
23                 if (f) return q;
24                 parent[r] = q;
25             } else {
26                 e.push_back(e[q]);
27                 parent.push_back(parent[q]);
28                 length.push_back(length[p] + 1);
29                 int qq = parent[q] = (int)e.size() - 1;
30                 for (; ~p && e[p][c] == q; p = parent[p]) e[p][c] =
    ↪ qq;
31                 if (f) return qq;
32                 parent[r] = qq;
33             }
34         }
35         return r;
36     }
37 };

```

- Topo sort on GSAM

```

1 ll sz = gsam.e.size();
2 vector<int> c(sz + 1);
3 vector<int> order(sz);
4 for (int i = 1; i < sz; i++) c[gsam.length[i]]++;
5 for (int i = 1; i < sz; i++) c[i] += c[i - 1];
6 for (int i = 1; i < sz; i++) order[c[gsam.length[i]]--] = i;
7 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)

```

1 int main() {
2     int n, last = 0;
3     string s;
4     cin >> n;
5     auto a = GSAM();
6     for (int i = 0; i < n; i++) {
7         cin >> s;
8         last = 0; // reset last
9         for (auto&& c : s) last = a.extend(c, last);
10    }
11    ll ans = 0;
12    for (int i = 1; i < a.e.size(); i++) {
13        ans += a.length[i] - a.length[a.parent[i]];
14    }
15    cout << ans << endl;
16    return 0;
17 }

```

## Manacher

```

1 string longest_palindrome(string& s) {
2     // init "abc" -> "~$a#b#c$"

```

```

3  vector<char> t{'^', '#'};
4  for (char c : s) t.push_back(c), t.push_back('#');
5  t.push_back('$');
6  // manacher
7  int n = t.size(), r = 0, c = 0;
8  vector<int> p(n, 0);
9  for (int i = 1; i < n - 1; i++) {
10     if (i < r + c) p[i] = min(p[2 * c - i], r + c - i);
11     while (t[i + p[i] + 1] == t[i - p[i] - 1]) p[i]++;
12     if (i + p[i] > r + c) r = p[i], c = i;
13 }
14 // s[i] -> p[2 * i + 2] (even), p[2 * i + 2] (odd)
15 // output answer
16 int index = 0;
17 for (int i = 0; i < n; i++)
18     if (p[index] < p[i]) index = i;
19 return s.substr((index - p[index]) / 2, p[index]);
20 }

```

## Lyndon

- def:  $\text{suf}(s) > s$

```

1  void duval(const string &s) {
2      int n = (int)s.size();
3      for (int i = 0; i < n; i++) {
4          int j = i, k = i + 1;
5          for (; j < n && s[j] <= s[k]; j++, k++)
6              if (s[j] < s[k]) j = i - 1;
7
8          while (i <= j) {
9              // cout << s.substr(i, k - j) << '\n';
10             i += k - j;
11         }
12     }
13 }

```

## minimal representation

```

1  int k = 0, i = 0, j = 1;
2  while (k < n && i < n && j < n) {
3      if (s[(i + k) % n] == s[(j + k) % n]) {
4          k++;
5      } else {
6          s[(i + k) % n] > s[(j + k) % n] ? i = i + k + 1 : j = j +
↪ k + 1;
7          if (i == j) i++;
8          k = 0;
9      }
10 }
11 i = min(i, j); // from 0

```

## suffix array

```

1  vi classTable[21];
2  vector<int> suffix_array(string const& s) {
3      forn(i, 21) classTable[i].clear();
4
5      int n = s.size();
6      const int alphabet = 256;
7      vector<int> p(n), c(n), cnt(max(alphabet, n), 0);
8      for (int i = 0; i < n; i++)
9          cnt[s[i]]++;
10     for (int i = 1; i < alphabet; i++)
11         cnt[i] += cnt[i-1];
12     for (int i = 0; i < n; i++)
13         p[--cnt[s[i]]] = i;
14     c[p[0]] = 0;
15     int classes = 1;
16     for (int i = 1; i < n; i++) {
17         if (s[p[i]] != s[p[i-1]])
18             classes++;
19         c[p[i]] = classes - 1;
20     }
21     classTable[0] = c;

```

```

22     vector<int> pn(n), cn(n);
23     for (int h = 0; (1 << h) < n; ++h) {
24         for (int i = 0; i < n; i++) {
25             pn[i] = p[i] - (1 << h);
26             if (pn[i] < 0)
27                 pn[i] += n;
28         }
29         fill(cnt.begin(), cnt.begin() + classes, 0);
30         for (int i = 0; i < n; i++)
31             cnt[c[pn[i]]]++;
32         for (int i = 1; i < classes; i++)
33             cnt[i] += cnt[i-1];
34         for (int i = n-1; i >= 0; i--)
35             p[--cnt[c[pn[i]]]] = pn[i];
36         cn[p[0]] = 0;
37         classes = 1;
38         for (int i = 1; i < n; i++) {
39             pair<int, int> cur = {c[p[i]], c[(p[i] + (1 << h))
↪ % n]};
40             pair<int, int> prev = {c[p[i-1]], c[(p[i-1] + (1
↪ << h)) % n]};
41             if (cur != prev)
42                 ++classes;
43             cn[p[i]] = classes - 1;
44         }
45         c.swap(cn);
46         classTable[h+1] = c;
47     }
48     return p;
49 }
50
51 int lcp(int a, int b) {
52     int ans = 0;
53     for (int i = 19; i >= 0; i--) {
54         if (classTable[i].size() == 0) continue;
55         if (classTable[i][a] == classTable[i][b]) {
56             a += (1 << i);
57             b += (1 << i);
58             ans += (1 << i);
59         }
60     }
61     return ans;
62 }

```