

C++ Snippets

Muhammad Samir Al-Sawalhy

Contents

March 1, 2024

1	Competitive programming template
2	Competitive programming template with multi-tests
3	Increase the stack memory limit
4	Read an array of length n from the stdin
5	Brute force primality test
6	Miller & rabin probabilistic primality test
7	Prime factorization in $O(\sqrt{n})$
8	Euler's totient theorem
9	Sieve's algorithm to mark numbers as primes and composites
10	Fast sieve's algorithm to calc minimum prime
11	Dijkstra's tsp algorithm
12	Mst (minimum spanning tree), kruskal's algorithm
13	Computational geometry stuff for competitive prgramming
14	Stl policy container (oset, omap)
15	Optimized segment tree with basic operations
16	Segment tree data structure
17	Persistent segtree
18	N-d binary indexed tree
19	Sparse table
20	Modular arithmetics stolen from jiangly
21	Modular combinations
22	Disjoint set union
23	Matrix exponentiation
24	Fast input scanner
25	Pascal triagle, useful for combinations
26	Description
27	Extended euclidian algorithm

28	Modular inverse for coprimes not only prime mod	9
29	Trie data structure	9
30	String hashing implementation (polynomial hashing)	9
31	Eulerian path/circuit in directed graphs	10
32	Eulerian path/circuit in undirected graphs	10
33	Mo's algorithm	11
34	Torjan's algorithm	11
35	Kmp string algorithm	11
36	Z algorithm for strings	11
37	Random utils	11
38	Least common ancestor using binary lifting	11
39	Least common ancestor using sparse table	12
40	Centroid decomposition of a tree	12

1 Competitive programming template

```
1 //
2 #include <bits/stdc++.h>
3 using namespace std;
4
5 #ifdef SAWALHY
6 #include "debug.hpp"
7 #else
8 #define debug(...) 0
9 #define debug_itr(...) 0
10 #define debug_bits(...) 0
11 #endif
12
13 #define ll long long
14 #define int long long
15 #define all(v) v.begin(), v.end()
16 #define rall(v) v.rbegin(), v.rend()
17 #define minit(v, x) v = min(v, x)
18 #define maxit(v, x) v = max(v, x)
19
20 int32_t main() {
21     ios_base::sync_with_stdio(false);
22     cin.tie(NULL), cout.tie(NULL);
23
24     ${0}
25
26     return 0;
27 }
```

2 Competitive programming template with multi-tests

```
8 //
9 #include <bits/stdc++.h>
10 using namespace std;
11
12 #ifdef SAWALHY
13 #include "debug.hpp"
14 #else
15 #define debug(...) 0
16 #define debug_itr(...) 0
17 #define debug_bits(...) 0
18 #endif
```

```

11 #endif
12
13 #define ll long long
14 #define int long long
15 #define all(v) v.begin(), v.end()
16 #define rall(v) v.rbegin(), v.rend()
17 #define minit(v, x) v = min(v, x)
18 #define maxit(v, x) v = max(v, x)
19
20 void solve() {
21     ${0}
22 }
23
24 int32_t main() {
25     ios_base::sync_with_stdio(false);
26     cin.tie(NULL), cout.tie(NULL);
27
28     int t;
29     cin >> t;
30     while (t--)
31         solve();
32
33     return 0;
34 }

```

3 Increase the stack memory limit

```

1 static void run_with_stack_size(void (*func)(void), size_t stsize) {
2     char *stack, *send;
3     stack = (char *)malloc(stsize);
4     send = stack + stsize - 16;
5     send = (char *)((uintptr_t)send / 16 * 16);
6     asm volatile(
7         "mov %%rsp, (%0)\n"
8         "mov %0, %%rsp\n"
9         :
10        : "r"(send));
11    func();
12    asm volatile("mov (%0), %%rsp\n" : : "r"(send));
13    free(stack);
14 }
15
16 int32_t main() {
17     run_with_stack_size(main_, 1024 * 1024 * 1024 / 2); // run with a 512 MB stack
18     return 0;
19 }

```

4 Read an array of length n from the stdin

```

1 int n;
2 cin >> n;
3 vector<int> a(n);
4 for (int i = 0; i < n; i++) {
5     cin >> a[i];
6 }

```

5 Brute force primality test

```

1 bool is_prime(ll n) {
2     if (n < 2) return false;
3     if (n == 2) return true;
4     if (n % 2 == 0) return false;
5     for (ll i = 3; i * i <= n; i += 2)
6         if (n % i == 0) return false;
7     return true;
8 }

```

6 Miller & rabin probabilistic primality test

```

1 bool miller_rabin_ptest(unsigned ll n, int k = 3) {
2     if (n < 2) return false;
3     if (n == 2) return true;
4
5     while (k--) {

```

```

6         unsigned ll a = 1LL * rand() * rand() % (n - 2) + 2; // [2 ... n-1]
7         unsigned ll r = 1;
8         for (unsigned ll p = n - 1; p; p >= 1) {
9             if (p & 1) r = r * a % n;
10            a = a * a % n;
11        }
12        if (r != 1) return false;
13    }
14
15    return true; // probably
16 }

```

7 Prime factorization in $O(\sqrt{n})$

```

1 map<ll, ll> primefacts(ll n) {
2     map<ll, ll> result;
3     int r = 0;
4
5     while (n % 2 == 0) {
6         r++;
7         n = n / 2;
8     }
9
10    if (r > 0)
11        result[2] = r;
12
13    int sqn = sqrt(n);
14    for (int i = 3; i <= sqn; i += 2) {
15        r = 0;
16        while (n % i == 0) {
17            r++;
18            n = n / i;
19        }
20        if (r > 0)
21            result[i] = r;
22    }
23
24    if (n > 2)
25        result[n] = 1;
26
27    return result;
28 }

```

8 Euler's totient theorem

```

1 std::vector<int> phi(${1:n} + 1);
2 std::iota(phi.begin(), phi.end(), 0);
3
4 for (int i = 1; i <= ${2:$1}; i++) {
5     for (int j = i < 1; j <= ${2:$1}; j += i)
6         phi[j] -= phi[i];
7 }

```

9 Sieve's algorithm to mark numbers as primes and composites

```

1 void sieve(vector<bool> &is_prime) {
2     is_prime[1] = false;
3     is_prime[0] = false;
4     int s = is_prime.size();
5     for (int i = 4; i < s; i += 2)
6         is_prime[i] = false;
7     for (int i = 3; i * i < s; i += 2) {
8         if (is_prime[i]) {
9             for (int j = i * i; j < s; j += i + i)
10                 is_prime[j] = false;
11         }
12     }
13 }

```

10 Fast sieve's algorithm to calc minimum prime

```

1 std::vector<int> minp, primes;
2 void sieve(int n) {
3     minp.assign(n + 1, 0);
4     primes.clear();
5
6     for (int i = 2; i <= n; i++) {
7         if (minp[i] == 0) {
8             minp[i] = i;
9             primes.push_back(i);
10        }
11
12        for (auto p : primes) {
13            if (i * p > n) {
14                break;
15            }
16            minp[i * p] = p;
17            if (p == minp[i]) {
18                break;
19            }
20        }
21    }
22 }

```

11 Dijkstra's tsp algorithm

```

1 long long dijkstra(int s, int e, vector<vector<pair<int, int>>> &adj) {
2     int n = adj.size();
3     vector<int> prev(n + 1);
4     vector<ll> dist(n + 1, 1e18);
5
6     typedef pair<ll, int> item;
7     priority_queue<item, deque<item>, greater<item>> qu;
8     qu.push({0, s});
9     dist[s] = 0;
10
11     while (!qu.empty()) {
12         auto [d, i] = qu.top();
13         qu.pop();
14
15         if (dist[i] < d) continue;
16         for (auto [j, D]: adj[i]) {
17             if (dist[j] > D + d) {
18                 prev[j] = i;
19                 dist[j] = D + d;
20                 qu.push({dist[j], j});
21             }
22         }
23     }
24
25     // for (int i = e; i != s; i = prev[i]);
26     return dist[e];
27 }

```

12 Mst (minimum spanning tree), kruskal's algorithm

```

1 struct Edge {
2     int from, to;
3     long long weight;
4     Edge(int from, int to, long long weight) : from(from), to(to), weight(weight) {}
5     bool operator<(Edge &e) { return weight < e.weight; }
6 };
7
8 pair<long long, vector<Edge>> mst_kruskal(vector<Edge> &edges, int n) {
9     DSU uf(n + 1);
10    double cost = 0;
11    vector<Edge> mst_edges;
12
13    sort(edges.rbegin(), edges.rend());
14
15    while (!edges.empty()) {
16        auto &e = edges.back();
17        edges.pop_back();
18        if (uf.uni(e.from, e.to)) {
19            cost += e.weight;
20            mst_edges.push_back(e);
21        }
22    }
23
24    if (mst_edges.size() != n - 1)
25        return {1e18, {}};

```

```

26
27     return {cost, mst_edges};
28 }

```

13 Computational geometry stuff for competitive programming

```

1 namespace Geometry
2 {
3
4     using T = long double;
5     const T EPS = 1e-8;
6     const double PI = acos(-1.0);
7
8     template<typename T, typename V>
9     int cmp(T a, V b) { return (a - b) < -EPS ? -1 : (a > EPS ? 1 : 0); }
10    template<typename T, typename V>
11    bool iseq(T a, V b) { return cmp(a, b) == 0; }
12    template<typename T>
13    bool iseq0(T a) { return cmp(a, 0) == 0; }
14    template<typename T, typename V>
15    bool islte(T a, V b) { return cmp(a, b) != 1; }
16    template<typename T, typename V>
17    bool isgte(T a, V b) { return cmp(a, b) != -1; }
18    template<typename T, typename V>
19    bool islt(T a, V b) { return cmp(a, b) == -1; }
20    template<typename T, typename V>
21    bool isgt(T a, V b) { return cmp(a, b) == 1; }
22    template<typename T>
23    int sign(T val) { return cmp(val, 0); }
24
25    enum PointState { OUT,
26                    IN,
27                    ON };
28
29    typedef struct Point {
30        T x, y;
31
32        Point() {}
33        Point(T _x, T _y) : x(_x), y(_y) {}
34        Point operator+(const Point &p) const { return Point(x + p.x, y + p.y); }
35        Point operator-(const Point &p) const { return Point(x - p.x, y - p.y); }
36        Point operator/(T denom) const { return Point(x / denom, y / denom); }
37        Point operator*(T scaler) const { return Point(x * scaler, y * scaler); }
38
39        T dot(const Point &p) const { return x * p.x + y * p.y; }
40        T cross(const Point &p) const { return x * p.y - y * p.x; }
41        T dot(const Point &a, const Point &b) const { return (a - *this).dot(b - *this); }
42        T cross(const Point &a, const Point &b) const { return (a - *this).cross(b - *this); }
43        T norm() const { return dot(*this); }
44
45        long double len() const { return sqrt1(dot(*this)); }
46        long double ang(bool pos = true) const {
47            auto a = atan2(y, x);
48            if (pos && a < 0) a += PI * 2;
49            return a;
50        }
51
52        Point rotate(const Point &p, long double a) { return (*this - p).rotate(a) + p; }
53        Point rotate(long double angle) {
54            auto l = len(), a = ang();
55            return Point(l * cos(a + angle), l * sin(a + angle));
56        }
57
58        bool operator==(const Point &p) const { return (*this - p).norm() <= EPS; }
59        bool operator!=(const Point &p) const { return !(*this == p); }
60        bool operator<(const Point &p) const { return x < p.x || (x == p.x && y < p.y); }
61        friend ostream &operator<<(ostream &os, const Point &p) { return os << '(' << p.x << ', ' << p.y <<
62            ')'; }
63        friend istream &operator>>(istream &is, Point &p) { return is >> p.x >> p.y; }
64    } pt;
65
66    int ccw(const pt &a, pt &b, pt &c) {
67        if (a == b) return (a == c ? 0 : +3); // same point or different
68        b = b - a, c = c - a;
69        if (sign(b.cross(c)) == +1) return +1; // "COUNTER_CLOCKWISE"
70        if (sign(b.cross(c)) == -1) return -1; // "CLOCKWISE"
71        if (sign(b.dot(c)) == -1) return +2; // "ON_RAY_b_a"
72        if (cmp(b.norm(), c.norm()) == -1) return -2; // "ON_RAY_a_b"
73        return 0; // "ON_SEGMENT"
74    }
75
76    bool colinear(const pt &a, const pt &b, const pt &c) {
77        return abs(ccw(a, b, c)) != 1;
78    }
79 }

```

```

79 pt slope(pt a, pt b, bool change_direction = true) {
80     assert(is_integral_v<T>);
81     long long dx = a.x - b.x;
82     long long dy = a.y - b.y;
83     if (dx == 0 && dy == 0) return pt(0, 0);
84     long long g = gcd(abs(dy), abs(dx));
85     dx /= g, dy /= g;
86     if (change_direction) {
87         if (dx < 0) dy *= -1, dx *= -1;
88         if (dx == 0) dy = abs(dy);
89     }
90     return pt(dx, dy);
91 }
92
93 struct Segment {
94     pt a, b;
95     Segment() {}
96     Segment(pt a, pt b) : a(a), b(b) {}
97     bool operator==(const Segment &s) const { return a == s.a ? b == s.b : a == s.b && b == s.a; };
98     friend istream &operator>>(istream &is, Segment &s) { return is >> s.a >> s.b; }
99     friend ostream &operator<<(ostream &os, const Segment &s) {
100         return os << "\"" << s.a << ", " << s.b << "\"";
101     }
102 };
103
104 struct Line : public Segment {
105     Line() {}
106     Line(pt a, pt b) : Segment(a, b) {}
107     bool operator==(const Line &l) const { return iseq0((a - b).cross(l.a - l.b)); };
108 };
109
110 struct Ray : public Segment {
111     Ray() {}
112     Ray(pt a, pt b) : Segment(a, b) {}
113     bool operator==(const Ray &r) const { return a == r.a && slope(a, b, false) == slope(r.a, r.b, false); };
114 };
115
116 struct Polygon {
117     int n;
118     vector<pt> verts;
119     Polygon() = default;
120     Polygon(int n) : n(n) { verts.resize(n); }
121     Polygon(vector<pt> &vert) : verts(vert), n(vert.size()) {}
122
123     T area2() const {
124         T a = 0;
125         for (int i = 2; i < n; i++)
126             a += verts[0].cross(verts[i], verts[i - 1]);
127         return abs(a);
128     }
129
130     long double area() const { return area2() / 2.0; };
131
132     void no_collinear() {
133         vector<pt> v;
134         for (int i = 0; i <= n; i++) {
135             while (v.size() > 1 && colinear(v.back(), v.end()[-2], verts[i % n]))
136                 v.pop_back();
137             v.push_back(verts[i % n]);
138         }
139         v.pop_back();
140         n = v.size();
141         verts = v;
142         assert(n > 2);
143     }
144
145     void ensure_ccw() {
146         start_bottom_left();
147         if (ccw(verts[0], verts[1], verts.back()) == -1)
148             reverse(verts.begin() + 1, verts.end());
149     }
150
151     void start_bottom_left() {
152         int pos = 0; // most left-bottom point
153         for (int i = 1; i < n; i++)
154             if (verts[i] < verts[pos])
155                 pos = i;
156         rotate(verts.begin(), verts.begin() + pos, verts.end());
157     }
158 };
159
160 bool parallel(const Line &a, const Line &b) { return (a.b - a.a).cross(b.b - b.a) == 0; }
161 bool orthogonal(const Line &a, const Line &b) { return (a.a - a.b).dot(b.a - b.b) == 0; }
162
163 bool intersect(const Line &l, const Line &m) { return !parallel(l, m); }
164
165 bool intersect(const pt &p, const Segment &s) { return ccw(s.a, s.b, p) == 0; }
166 bool intersect(const Segment &s, const pt &p) { return intersect(p, s); }
167
168 bool intersect(const pt &p, const Line &l) { return abs(ccw(l.a, l.b, p)) != 1; }
169 bool intersect(const Line &l, const pt &p) { return intersect(p, l); }

```

```

170
171 bool intersect(const Segment &s, const Line &l) { return ccw(l.a, l.b, s.a) * ccw(l.a, l.b, s.b) != 1; }
172
173 bool intersect(const Line &l, const Segment &s) { return intersect(s, l); }
174
175 bool intersect(const Segment &s, const Segment &t) { return ccw(s.a, s.b, t.a) * ccw(s.a, s.b, t.b) <= 0 && ccw(t.a, t.b, s.a) * ccw(t.a, t.b, s.b) <= 0; }
176
177 bool intersect(const Segment &s, const Ray &r) {
178     auto d1 = (s.a - s.b).cross(r.b - r.a),
179           d2 = (s.a - r.a).cross(r.b - r.a),
180           d3 = (s.a - s.b).cross(s.a - r.a);
181     if (abs(d1) <= EPS)
182         return r.a.cross(r.b, s.a) == 0 &&
183             (r.a.dot(r.b, s.a) >= 0 || r.a.dot(r.b, s.b) >= 0); // NOT BACK
184     return sign(d1) * sign(d2) >= 0 && sign(d1) * sign(d3) >= 0 && abs(d2) <= abs(d1);
185 }
186
187 bool intersect(const Ray &r, const Segment &s) { return intersect(s, r); }
188
189 bool intersection(pt a, pt b, pt c, pt d, pt &inter) {
190     assert(is_floating_point_v<T>);
191     long double d1 = (a - b).cross(d - c);
192     long double d2 = (a - c).cross(d - b);
193     if (fabs(d1) <= EPS) return false;
194     long double t1 = d2 / d1;
195     inter = a + (b - a) * t1;
196     return true;
197 }
198
199 template<typename T, typename V>
200 bool intersection(const T &l, const V &m, pt &inter) {
201     if (!intersect(l, m)) return false;
202     return intersection(l.a, l.b, m.a, m.b, inter);
203 }
204
205 // - NOTE: The polygon shouldn't have collinear points.
206 // - NOTE: First vertex should be the bottom-left, points in ccw order.
207 vector<pt> intersection(const Polygon &poly, const Line &line) {
208     int n = poly.n;
209     vector<pt> inter;
210     const vector<pt> &verts = poly.verts;
211
212     pt x;
213     for (int i = 1; i <= n; i++) {
214         int I = i % n, J = i - 1, K = (i - 2 + n) % n;
215         if (intersection(line, Segment(verts[I], verts[J]), x)) {
216             if (x == verts[I]) continue;
217             if (x != verts[J]) {
218                 inter.push_back(x);
219                 continue;
220             }
221             int dir1 = ccw(line.a, line.b, verts[I]);
222             int dir2 = ccw(line.a, line.b, verts[K]);
223             if (dir1 * dir2 == -1)
224                 // entering or leaving from a vertex
225                 inter.push_back(verts[J]);
226             // side (I, J) is on the line
227             bool isWideAngleI = islt(ccw(verts[I], verts[(I + 1) % n], verts[J]), 0);
228             bool isWideAngleJ = islt(ccw(verts[J], verts[I], verts[K]), 0);
229             if (isWideAngleI) inter.push_back(verts[I]);
230             if (isWideAngleJ) inter.push_back(verts[J]);
231             inter.push_back(verts[I]);
232             inter.push_back(verts[J]);
233         }
234     }
235
236     debug(inter);
237
238     // sort in one direction, as if you travel on the line
239     // in this direction and see the points one by one
240     // NOTE: points may NOT be exactly on the line due to precesion errors
241     sort(all(inter), [&l](pt l, pt r) {
242         return sign((line.b - line.a).dot(r - l)) == 1;
243     });
244
245     assert(inter.size() % 2 == 0);
246     return inter;
247 };
248
249 struct Circle {
250     pt c;
251     T r;
252
253     Circle() = default;
254     Circle(pt c, T r) : c(c), r(r) {}
255     Circle(const vector<pt> &p) {
256         if (p.size() == 1) c = p[0], r = 0;
257         else if (p.size() == 2) {
258             c = (p[0] + p[1]) / 2;
259             r = (p[0] - c).len();

```

```

260     } else {
261         assert(p.size() == 3);
262         *this = Circle(p[0], p[1], p[2]);
263     }
264 }
265
266 Circle(pt a, pt b, pt c) {
267     // if we have a cord in a circle,
268     // the perpendicular from the center will pass from the center
269     // so we simply solve for the intersection of two lines
270     auto ABmid = (a + b) / 2.0, BCmid = (b + c) / 2.0;
271     auto ABnorm = pt((a - b).y, -(a - b).x);
272     auto BCnorm = pt((b - c).y, -(b - c).x);
273     bool valid = intersection(
274         Line(ABmid, ABmid + ABnorm),
275         Line(BCmid, BCmid + BCnorm), this->c);
276     assert(valid); // unless at least two points are identical
277     r = (a - this->c).len();
278 }
279
280 friend ostream& operator<<(ostream& os, const Circle& c) {
281     return os << "c(" << c.c << ", " << c.r << ")";
282 }
283 };
284
285 PointState point_in_triangle(pt a, pt b, pt c, pt point) {
286     int x = ccw(a, b, point), y = ccw(b, c, point), z = ccw(c, a, point);
287     if (sign(x) == sign(y) && sign(y) == sign(z)) return IN;
288     if (x * y * z == 0) return ON;
289     return OUT;
290 }
291
292 PointState point_in_circle(const pt &p, const vector<pt> &cir) {
293     if (cir.size() == 0) return OUT;
294     auto c = Circle(cir);
295     if (isseq((p - c.c).norm(), c.r + c.r)) return ON;
296     if (islt((p - c.c).norm(), c.r + c.r)) return IN;
297     return OUT;
298 }
299
300 PointState point_in_polygon(const pt &p, const vector<pt> &polygon) {
301     int wn = 0, n = polygon.size();
302     for (int i = 0, j = 1; i < n; i++, j++, j %= n) {
303         if (ccw(polygon[j], polygon[i], p) == 0) return ON;
304         if ((p.y < polygon[j].y) != (p.y < polygon[i].y)) {
305             wn += polygon[j].y > polygon[i].y && ccw(p, polygon[i], polygon[j]) == 1;
306             wn -= polygon[j].y < polygon[i].y && ccw(p, polygon[j], polygon[i]) == 1;
307         }
308     }
309     return wn == 0 ? OUT : IN;
310 }
311
312 PointState ray_and_polygon(const Ray &r, const Polygon &polygon) {
313     // NOTE: Should be a good ray (a != b),
314     // and non-degenerate polygon with no duplicated points
315     int n = polygon.n;
316     PointState ans = OUT;
317     for (int i = 0, j = 1, k = 2; i < n; i++, j++, k++, j %= n, k %= n) {
318         if (!intersect(Segment(polygon.verts[i], polygon.verts[j]), r)) continue;
319         auto x = r.a.cross(r.b, polygon.verts[i]);
320         auto y = r.a.cross(r.b, polygon.verts[j]);
321         auto z = r.a.cross(r.b, polygon.verts[k]);
322         if (x == 0) ans = ON; // Maybe tangent
323         else if (y == 0) {
324             // (the ray splits an internal angle)
325             // Entering from a vertex
326             if (sign(x) + sign(z) == -1) return IN;
327         } else return IN; // Entering from an edge
328     }
329     return ans;
330 }
331
332 vector<pt> &sort_clock(vector<pt> &points, bool cw = false) {
333     int n = points.size();
334
335     // choose the pivot (most bottom-right point)
336     for (int i = 1; i < n; i++) {
337         auto &l = points[0], &r = points[i];
338         int cy = cmp(l.y, r.y), cx = cmp(l.x, r.x);
339         if (cy == 0 ? cx == -1 : cy == +1) swap(l, r);
340     }
341
342     // sorting with points[0] as pivot
343     sort(points.begin() + 1, points.end(),
344         [&](pt l, pt r) {
345             auto c = ccw(points[0], l, r);
346             int cx = cmp(l.x, r.x), cy = cmp(l.y, r.y);
347             // closer to bottom-right comes first
348             if (abs(c) != 1) return cy == 0 ? cx == 1 : cy == -1;
349             return cw ? c == -1 : c == 1;
350         });
351 }

```

```

352     return points;
353 }
354
355 // sort a convex polygon cw or ccw with the bottom-right as the pivot
356 vector<pt> &sort_convex(vector<pt> &points, bool cw = false) {
357     int n = points.size();
358
359     // choose the pivot (most bottom-right point)
360     for (int i = 1; i < n; i++) {
361         auto &l = points[0], &r = points[i];
362         int cy = cmp(l.y, r.y), cx = cmp(l.x, r.x);
363         if (cy == 0 ? cx == -1 : cy == +1) swap(l, r);
364     }
365
366     // sorting with points[0] as pivot
367     sort(points.begin() + 1, points.end(),
368         [&](pt l, pt r) {
369             auto c = ccw(points[0], l, r);
370             int cx = cmp(l.x, r.x), cy = cmp(l.y, r.y);
371
372             if (abs(c) != 1) { // collinear
373                 if (cw) return cy == 0 ? cx == 1 : cy == 1;
374                 else
375                     return cy == 0 ? cx == -1 : cy == -1;
376             }
377
378             return cw ? c == -1 : c == 1;
379         });
380
381     return points;
382 }
383
384 vector<pt> convexhull(vector<pt> &p, bool strict = false) {
385     int n = p.size(), k = 0, sgn = strict ? 0 : -1;
386     if (n <= 2) return p;
387     vector<pt> ch(2 + n); // CCW
388     auto cmp = [](pt x, pt y) { return (x.x != y.x ? x.x < y.x : x.y < y.y); };
389     sort(begin(p), end(p), cmp);
390     for (int i = 0; i < n; ch[k++] = p[i++]) // lower hull
391         while (k >= 2 && sign((ch[k - 1] - ch[k - 2]).cross(p[i] - ch[k - 2])) <= sgn) --k;
392     for (int i = n - 2, t = k + 1; i >= 0; ch[k++] = p[i--]) // upper hull
393         while (k >= t && sign((ch[k - 1] - ch[k - 2]).cross(p[i] - ch[k - 2])) <= sgn) --k;
394     ch.resize(k - 1);
395     return ch;
396 }
397
398 struct PointInConvex {
399     int n;
400     vector<pt> seq;
401     pt translation;
402
403     PointInConvex(vector<pt> &polygon) { prepare_convex_ccw(polygon); }
404
405     void prepare_convex_ccw(vector<pt> &points) {
406         // NOTE: the polygon should be strictly convex
407         n = points.size();
408         int pos = 0; // most left-bottom point
409         for (int i = 1; i < n; i++)
410             if (points[i] < points[pos])
411                 pos = i;
412         rotate(points.begin(), points.begin() + pos, points.end());
413
414         seq.resize(n);
415         for (int i = 0; i < n; i++)
416             seq[i] = points[(i + 1) % n] - points[0];
417         translation = points[0];
418     }
419
420     int check(pt point) {
421         point = point - translation;
422         if (intersect(point, Segment(pt(0, 0), seq[0]))) return 0;
423         if (seq.size() <= 2) return -1;
424
425         int l = 0, r = n - 1;
426         while (r - l > 1) {
427             int mid = (l + r) / 2;
428             if (sign(seq[mid].cross(point)) != -1)
429                 l = mid;
430             else
431                 r = mid;
432         }
433
434         int ok = point_in_triangle(seq[l], seq[l + 1], pt(0, 0), point);
435         if (ok == -1) return -1;
436         if (intersect(point, Segment(seq[l], seq[l + 1]))) return 0;
437         return 1;
438     }
439 };
440
441 struct Welzl {
442     vector<pt> points;
443     Welzl(vector<pt> &_points) : points(_points) {

```

```

444     shuffle(all(points), default_random_engine(time(NULL)));
445 }
446
447 Circle get_circle() { return Circle(go()); }
448 vector<pt> go(int i = 0, vector<pt> cir = {}) {
449     if (cir.size() == 3 || i == (int) points.size()) return cir;
450     auto new_cir = go(i + 1, cir);
451     if (point_in_circle(points[i], new_cir) != OUT)
452         return new_cir;
453     cir.push_back(points[i]);
454     return go(i + 1, cir);
455 }
456 };
457
458 }; // namespace Geometry
459
460 using namespace Geometry;

```

14 Stl policy container (oset, omap)

```

1 #include<ext/pb_ds/assoc_container.hpp>
2 #include<ext/pb_ds/tree_policy.hpp>
3 using namespace __gnu_pbds;
4 template<typename T>
5 using ordered_set = tree<T, null_type, std::less<T>, rb_tree_tag, tree_order_statistics_node_update>;

```

15 Optimized segment tree with basic operations

```

1 template<typename T = long long>
2 struct Sum {
3     T value;
4     Sum(T value = 0) : value(value) {}
5     Sum &operator+=(const Sum &other) { return value += other.value, *this; }
6     Sum &operator+(const Sum &other) const { return value + other.value; }
7 };
8
9 template<typename T = long long>
10 struct Max {
11     T value;
12     Max(T value = numeric_limits<T>::min() / 2) : value(value) {}
13     Max &operator+=(const Max &other) { return value = max(value, other.value), *this; }
14     Max &operator+(const Max &other) const { return Max(max(value, other.value)); }
15 };
16
17 template<typename T = long long>
18 struct Min {
19     T value;
20     Min(T value = numeric_limits<T>::max() / 2) : value(value) {}
21     Min &operator+=(const Min &other) { return value = min(value, other.value), *this; }
22     Min &operator+(const Min &other) const { return Min(min(value, other.value)); }
23 };
24
25 // source: https://codeforces.com/blog/entry/18051
26 template<typename T>
27 struct Segtree {
28     int n;
29     vector<T> tree;
30
31     Segtree() = default;
32     Segtree(int n) : n(n) {
33         tree.resize(n * 2);
34     }
35
36     void build() {
37         for (int i = n - 1; i > 0; --i)
38             tree[i] = tree[i << 1] + tree[i << 1 | 1];
39     }
40
41     void update(int i, T val) {
42         for (tree[i += n] = val; i > 1; i >= 1)
43             tree[i >> 1] = tree[i] + tree[i ^ 1];
44     }
45
46     auto query(int l, int r) {
47         T res;
48         for (l += n, r += n + 1; l < r; l >= 1, r >= 1) {
49             if (l & 1) res += tree[l++];
50             if (r & 1) res += tree[--r];
51         }
52         return res.value;
53     }
54 };

```

16 Segment tree data structure

```

1 struct Value;
2 struct Update;
3 struct Node;
4
5 // Replaceable by primitives (using Value = long long)
6 struct Value {
7     long long sum = 0, mn = 1e18, mx = -1e18;
8     Value() = default;
9     Value(ll value) { sum = mn = mx = value; }
10
11     Value &operator+=(const Value &other) {
12         sum += other.sum;
13         mn = min(mn, other.mn);
14         mx = max(mx, other.mx);
15         return *this;
16     }
17
18     Value &operator+(const Value &other) const {
19         return Value(*this) += other;
20     }
21 };
22
23 struct Update {
24     // NOTE: Sometime you need to split the update, in these cases
25     // you should include the range [a, b] of the update in the struct Update
26     Value value;
27     enum State {
28         idle,
29         relative,
30         forced
31     } state = idle;
32
33     Update() = default;
34     Update(Value value, State state = forced) : value(value), state(state){};
35
36     Update &operator+=(const Update &other) {
37         if (state == idle || other.state == forced) {
38             *this = other;
39         } else {
40             assert(other.state == relative);
41             value += other.value;
42         }
43         return *this;
44     }
45
46     void apply_on(Value &other, int cnt) const {
47         if (state == forced) other = value;
48         else other += value;
49         other.sum += value.sum * (cnt - 1);
50     }
51
52     Update get(const Node &node) const { return *this; }
53 };
54
55 struct Node {
56     int l = -1, r = -1; // [l, r]
57     Update up;
58     Value value;
59
60     Node() = default;
61     Node(int l, int r, const Value &value) : l(l), r(r), value(value){};
62
63     void update(const Update &up) { this->up += up; }
64
65     void apply_update() {
66         up.apply_on(value, r - l + 1);
67         up.state = Update::idle;
68     }
69 };
70
71 struct Segtree {
72     int n;
73     vector<Node> tree;
74
75     Segtree(int n) {
76         if ((n & (n - 1)) != 0)
77             n = 1 << (32 - __builtin_clz(n));
78         this->n = n;
79         tree.assign(n << 1, Node());
80         for (int i = n; i < n << 1; i++)
81             tree[i].l = tree[i].r = i - n;
82         for (int i = n - 1; i > 0; i--)
83             tree[i].l = tree[i << 1].l, tree[i].r = tree[i << 1 | 1].r;
84     }
85
86     Segtree(const vector<Value> &values) : Segtree(values.size()) {
87         for (int i = 0; i < (int) values.size(); i++)

```

```

88     tree[i + n].value = values[i];
89     build();
90 }
91
92 void build() {
93     for (int i = n - 1; i > 0; --i) pull(i);
94 }
95
96 inline Value query(int i) { return query(1, i, i); }
97 inline Value query(int l, int j) { return query(1, i, j); }
98 inline void update(int l, const Update &val) { update(1, i, i, val); }
99 inline void update(int l, int j, const Update &val) { update(1, i, j, val); }
100
101 private:
102 void pull(int i) {
103     tree[i].value = tree[i << 1].value + tree[i << 1 | 1].value;
104 }
105
106 void push(int i) {
107     if (tree[i].up.state != Update::idle) {
108         if (i < n) {
109             int l = i << 1, r = i << 1 | 1;
110             tree[l].update(tree[i].up.get(tree[l]));
111             tree[r].update(tree[i].up.get(tree[r]));
112         }
113         tree[i].apply_update();
114     }
115 }
116
117 Value query(int i, int l, int r) {
118     push(i);
119     if (tree[i].r < l || r < tree[i].l) return Value(); // default
120     if (l <= tree[i].l && tree[i].r <= r) return tree[i].value;
121     return query(i << 1, l, r) + query(i << 1 | 1, l, r);
122 }
123
124 void update(int i, int l, int r, const Update &up) {
125     push(i);
126     if (tree[i].r < l || r < tree[i].l) return;
127     if (l <= tree[i].l && tree[i].r <= r) {
128         tree[i].update(up);
129         push(i); // to apply the update
130         return;
131     }
132     update(i << 1, l, r, up.get(tree[i << 1]));
133     update(i << 1 | 1, l, r, up.get(tree[i << 1 | 1]));
134     pull(i);
135 }
136 };

```

17 Persistent segtree

```

1  const int MIN = -(1 << 30), MAX = (1 << 30) - 1;
2  const int DEFAULT = 0;
3
4  struct Node {
5      long long value = DEFAULT;
6      Node *l = nullptr, *r = nullptr;
7      Node() = default;
8      Node(Node *l, Node *r) : l(l), r(r) {
9          if (l) value += l->value;
10         if (r) value += r->value;
11     }
12 };
13
14 struct PersistentSegtree {
15     Node *root = nullptr;
16
17     Node *update(int x, long long value) {
18         return root = update(x, value, root, MIN, MAX);
19     }
20
21     long long query(int L, int R) {
22         return query(L, R, root, MIN, MAX);
23     }
24
25 private:
26     Node *update(int x, long long value, Node *node, int l, int r) {
27         if (l == r) {
28             assert(l == x);
29             Node *ret = new Node();
30             ret->value = value;
31             return ret;
32         }
33
34         int mid = l + (r - l) / 2;
35         Node *left = node ? node->l : nullptr;
36         Node *right = node ? node->r : nullptr;

```

```

37         if (x <= mid) return new Node(update(x, value, left, l, mid), right);
38         else return new Node(left, update(x, value, right, mid + 1, r));
39     }
40
41     long long query(int L, int R, Node *node, int l, int r) {
42         if (!node || L > r || R < l) return DEFAULT;
43         if (L <= l && r <= R) return node->value;
44         int mid = l + (r - l) / 2;
45         return query(L, R, node->l, l, mid) + query(L, R, node->r, mid + 1, r);
46     }
47 };

```

18 N-d binary indexed tree

```

1  template<class T, int... Ns>
2  struct BIT {
3      T val = 0;
4      void update(T v) { val += v; }
5      T query() { return val; }
6  };
7
8  template<class T, int N, int... Ns>
9  struct BIT<T, N, Ns...> {
10     BIT<T, Ns...> bit[N + 1];
11     template<typename... Args>
12     void update(int pos, Args... args) {
13         for (pos++; pos <= N; pos += (pos & -pos)) bit[pos].update(args...);
14     }
15     template<typename... Args>
16     T sum(int r, Args... args) {
17         T res = 0;
18         for (r++; r; r -= (r & -r)) res += bit[r].query(args...);
19         return res;
20     }
21     template<typename... Args>
22     T query(int l, int r, Args... args) {
23         return sum(r, args...) - sum(l - 1, args...);
24     }
25 }; // BIT<int,10,10> gives a 2D BIT

```

19 Sparse table

```

1  template<typename T, class CMP = function<T(const T &, const T &)>>
2  class SparseTable {
3  public:
4      int n;
5      vector<vector<T>>> sp;
6      CMP func;
7
8      void build(const vector<T> &a, const CMP &f) {
9          func = f;
10         n = static_cast<int>(a.size());
11         int max_log = 32 - __builtin_clz(n);
12         sp.resize(max_log);
13         sp[0] = a;
14         for (int j = 1; j < max_log; ++j) {
15             sp[j].resize(n - (1 << j) + 1);
16             for (int i = 0; i <= n - (1 << j); ++i) {
17                 sp[j][i] = func(sp[j - 1][i], sp[j - 1][i + (1 << (j - 1))]);
18             }
19         }
20     }
21
22     T query(int l, int r) const {
23         int lg = 32 - __builtin_clz(r - l + 1) - 1;
24         return func(sp[lg][l], sp[lg][r - (1 << lg) + 1]);
25     }
26 };

```

20 Modular arithmetics stolen from jiangly

```

1  template<typename T = void> // default
2  struct BiggerType {
3      typedef ll type;
4  };
5
6  template<> // for long long
7  struct BiggerType<ll> {
8      typedef __int128 type;
9  };

```

```

10 template<typename T, T mod, typename V = typename BiggerType<T>::type>
11 struct mint {
12 private:
13     inline T norm(T x) const {
14         if (x < 0) x += mod;
15         if (x >= mod) x -= mod;
16         return x;
17     }
18 public:
19     T x;
20     mint(T x = 0) : x(norm(x)) {}
21     mint(V x) : x(norm(x % mod)) {}
22     mint operator-() const { return mint(norm(mod - x)); }
23     mint inv() const {
24         assert(x != 0);
25         return power(mod - 2);
26     }
27     mint power(long long b) const {
28         mint res = 1, a = x;
29         for (; b; b >>= 1, a *= a) {
30             if (b & 1) res *= a;
31         }
32         return res;
33     }
34     mint &operator+=(const mint &rhs) {
35         x = (V) x + rhs.x % mod;
36         return *this;
37     }
38     mint &operator+=(const mint &rhs) {
39         x = norm(x + rhs.x);
40         return *this;
41     }
42     mint &operator-=(const mint &rhs) {
43         x = norm(x - rhs.x);
44         return *this;
45     }
46     mint &operator/=(const mint &rhs) { return *this *= rhs.inv(); }
47     friend mint operator*(const mint &lhs, const mint &rhs) {
48         mint res = lhs;
49         res *= rhs;
50         return res;
51     }
52     friend mint operator+(const mint &lhs, const mint &rhs) {
53         mint res = lhs;
54         res += rhs;
55         return res;
56     }
57     friend mint operator-(const mint &lhs, const mint &rhs) {
58         mint res = lhs;
59         res -= rhs;
60         return res;
61     }
62     friend mint operator/(const mint &lhs, const mint &rhs) {
63         mint res = lhs;
64         res /= rhs;
65         return res;
66     }
67     friend bool operator==(const mint &lhs, const mint &rhs) {
68         return lhs.x == rhs.x;
69     }
70     friend std::istream &operator>>(std::istream &is, mint &a) {
71         T v;
72         return is >> v, a = mint(v), is;
73     }
74     friend std::ostream &operator<<(std::ostream &os, const mint &a) {
75         return os << a.x;
76     }
77     friend mint max(mint a, mint b) {
78         return a.x > b.x ? a : b;
79     }
80     friend mint min(mint a, mint b) {
81         return a.x < b.x ? a : b;
82     }
83     }
84 };
85
86 // constexpr int MOD = 998244353;
87 constexpr int MOD = 1000000007;
88 using Z = mint<int32_t, MOD>;

```

21 Modular combinations

```

1 vector<Z> fact = {1};
2 vector<Z> fact_inv = {1};
3
4 void build_fact(int n = 1e6) {
5     while ((int) fact.size() < n + 1)

```

```

6     fact.push_back(fact.back() * (int) fact.size());
7     fact_inv.resize(fact.size());
8     fact_inv.back() = fact.back().inv();
9     for (int j = fact_inv.size() - 2; fact_inv[j].x == 0; j--)
10         fact_inv[j] = fact_inv[j + 1] * (j + 1);
11 }
12
13 Z ncr(int n, int r) {
14     if (r > n || r < 0) return 0;
15     if ((int) fact.size() < n + 1) build_fact(n);
16     return fact[n] * fact_inv[r] * fact_inv[n - r];
17 }

```

22 Disjoint set union

```

1 struct DSU {
2     vector<int> size, parent;
3     int forests;
4
5     DSU(int n) {
6         forests = n;
7         size.assign(n, 1);
8         parent.resize(n);
9         iota(all(parent), 0);
10    }
11
12    bool connected(int x, int y) { return find(x) == find(y); }
13
14    int find(int x) {
15        if (parent[x] == x) return x;
16        return parent[x] = find(parent[x]);
17    }
18
19    bool uni(int x, int y) {
20        x = find(x), y = find(y);
21        if (x == y) return false;
22        forests--;
23        parent[y] = x;
24        size[x] += size[y];
25        return true;
26    }
27 };

```

23 Matrix exponentiation

```

1 constexpr ll MOD = 1e9 + 7;
2
3 template<typename T = int, int mod = MOD>
4 struct matrix {
5     typedef vector<vector<T>> vv;
6     vv mat;
7     int n, m;
8
9     matrix() { n = 0, m = 0; }
10    matrix(vv mat) : mat(mat) { n = mat.size(), m = mat[0].size(); }
11    matrix(int n, int m, T ini = 0) : n(n), m(m) { mat = vv(n, vector<T>(m, ini)); }
12
13    matrix operator*(const matrix &other) const {
14        matrix mat = *this;
15        return mat *= other;
16    }
17
18    matrix operator+(const matrix &other) const {
19        matrix mat = *this;
20        return mat += other;
21    }
22
23    matrix operator-(const matrix &other) const {
24        matrix mat = *this;
25        return mat -= other;
26    }
27
28    matrix &operator+=(const matrix &other) {
29        assert(m == other.n);
30        vector<vector<T>> temp(n, vector<T>(other.m));
31        for (int i = 0; i < n; i++) {
32            for (int j = 0; j < other.m; j++) {
33                for (int k = 0; k < m; k++) {
34                    temp[i][j] = (temp[i][j] + 1LL * mat[i][k] * other.mat[k][j]) % mod;
35                }
36            }
37        }
38        mat = temp;

```



```

39     m = other.m;
40     return *this;
41 }
42
43 matrix &operator+=(const matrix &other) {
44     assert(m == other.m && n == other.n);
45     for (int i = 0; i < n; i++) {
46         for (int j = 0; j < m; j++)
47             mat[i][j] = (mat[i][j] + other.mat[i][j]) % mod + mod) % mod;
48     }
49     return *this;
50 }
51
52 matrix &operator-=(const matrix &other) {
53     assert(m == other.m && n == other.n);
54     for (int i = 0; i < n; i++) {
55         for (int j = 0; j < m; j++)
56             mat[i][j] = (mat[i][j] - other.mat[i][j]) % mod + mod) % mod;
57     }
58     return *this;
59 }
60
61 matrix power(ll p) {
62     assert(p >= 0);
63     matrix m = *this;
64     matrix res = identity(n);
65     for (; p >= 1; m *= m)
66         if (p & 1) res *= m;
67     return res;
68 }
69
70 static matrix identity(int size) {
71     matrix I = vv(size, vector<T>(size));
72     for (int i = 0; i < size; i++)
73         I.mat[i][i] = 1;
74     return I;
75 }
76 };

```

24 Fast input scanner

```

1 char in[1 << 24];
2 struct Scanner {
3     char const *o;
4     Scanner() : o(in) { load(); }
5     void load() { in[fread(in, 1, sizeof(in) - 4, stdin)] = 0; }
6     unsigned readInt() {
7         unsigned u = 0;
8         while (*o && *o <= 32)
9             ++o;
10        while (*o >= '0' && *o <= '9')
11            u = u * 10 + (*o++ - '0');
12        return u;
13    }
14 } sc;

```

25 Pascal triagle, useful for combinations

```

1 vector<vector<Z>> pascal;
2 void build_pascal(int d) {
3     pascal = {{1}};
4     while (d--) {
5         vector<Z> &lastrow = pascal.back();
6         int s = lastrow.size();
7         vector<Z> newrow(s + 1);
8         newrow.front() = 1;
9         newrow.back() = 1;
10        for (int i = 1; i < s; i++)
11            newrow[i] = lastrow[i] + lastrow[i - 1];
12        pascal.push_back(newrow);
13    }
14 }

```

26 Description

```

1 template<int base = 10>
2 class bigint {
3 public:
4     vector<int> digits;

```

```

5
6     bigint(unsigned ll value = 0) { set_value(value); }
7
8     bigint(string s) {
9         digits.resize(s.size());
10        for (int i = (int) s.size() - 1; i >= 0; i--) {
11            digits[i] = s[(int) s.size() - 1 - i] - '0';
12        }
13    }
14
15    template<typename RandomIt>
16    bigint(RandomIt begin, RandomIt end) {
17        digits.assign(begin, end);
18    }
19
20    void set_value(ll value) {
21        digits.clear();
22        while (value) {
23            digits.push_back(value % base);
24            value /= base;
25        }
26    }
27
28    int size() const { return digits.size(); }
29
30    void trim() {
31        while (digits.back() == 0 && digits.size() > 1)
32            digits.pop_back();
33    }
34
35    int &operator[](int i) { return digits[i]; }
36
37    int operator[](int i) const { return digits[i]; }
38
39    void operator+=(const bigint &rhs) {
40        vector<int> res(size() + rhs.size() + 1);
41        for (int i = 0; i < size(); i++) {
42            for (int j = 0; j < rhs.size(); j++) {
43                res[i + j] += digits[i] * rhs[j];
44            }
45        }
46        for (int i = 0; i < (int) res.size() - 1; i++) {
47            res[i + 1] += res[i] / base;
48            res[i] %= base;
49        }
50        digits = res;
51        trim();
52    }
53
54    void operator+=(const bigint &rhs) {
55        digits.resize(max(size(), rhs.size()) + 1);
56        int i;
57        for (i = 0; i < rhs.size(); i++) {
58            digits[i] += rhs[i];
59            if (digits[i] >= base) {
60                digits[i + 1] += digits[i] / base;
61                digits[i] %= base;
62            }
63        }
64        while (i < (int) digits.size() - 1 && digits[i] >= base) {
65            digits[i + 1] += digits[i] / base;
66            digits[i] %= base;
67        }
68        trim();
69    }
70
71    void operator%=(ll mod) {
72        ll p = 1;
73        ll res = 0;
74        for (int i = 0; i < size(); i++) {
75            res = (res + p * digits[i] % mod) % mod;
76            p = p * base % mod;
77        }
78        *this = res;
79    }
80
81    friend bool operator==(bigint &lhs, bigint &rhs) {
82        return lhs.digits == rhs.digits;
83    }
84
85    friend bool operator!=(bigint &lhs, bigint &rhs) {
86        return lhs.digits != rhs.digits;
87    }
88
89    friend bool operator<(bigint &lhs, bigint &rhs) {
90        if (lhs.size() != rhs.size())
91            return lhs.size() < rhs.size();
92        for (int i = lhs.size() - 1; i >= 0; i--) {
93            if (lhs[i] < rhs[i]) return true;
94            if (lhs[i] > rhs[i]) return false;
95        }
96        return false; // equal

```

```

97     }
98
99     friend ostream &operator<<(ostream &os, const bigint &bi) {
100         for (int i = bi.size() - 1; i >= 0; i--) os << bi[i];
101         return os;
102     }
103 };

```

27 Extended euclidian algorithm

```

1 // a * x + b * y = gcd(a, b)
2 pair<ll, ll> exgcd(ll a, ll b) {
3     if (!b) return {1, 0};
4     pair<int, int> p = exgcd(b, a % b);
5     return {p.second, p.first - (a / b) * p.second};
6 }

```

28 Modular inverse for coprimes not only prime mod

```

1 // source: https://codeforces.com/blog/entry/23365
2 // a and b must be co-prime. returns (1 / a) mod b.
3 ll mod_inv(ll a, ll b) {
4     return 1 < a ? b - mod_inv(b % a, a) * b / a : 1;
5 }

```

29 Trie data structure

```

1 struct Trie {
2     vector<vector<int>> child;
3     vector<int> count;
4
5     Trie() {
6         add_node();
7     }
8
9     int add_node() {
10        count.push_back(0);
11        child.push_back(vector<int>(26));
12        return count.size() - 1;
13    }
14
15    void insert(const string &s) {
16        int cur = 0; // root
17        for (auto c: s) {
18            if (child[cur][c - 'a'] == 0)
19                child[cur][c - 'a'] = add_node();
20            cur = child[cur][c - 'a'];
21            count[cur]++;
22        }
23    }
24 };

```

30 String hashing implementation (polynomial hashing)

```

1 struct HashValue;
2 constexpr int HASH_ITER = 1;
3 const ll M = (1LL << 61) - 1;
4 vector<HashValue> B;
5
6 struct HashValue {
7     long long val[HASH_ITER]{};
8     HashValue(long long v = 0) {
9         for (int i = 0; i < HASH_ITER; i++) val[i] = v;
10    }
11    HashValue &operator+=(const HashValue &a) {
12        for (int i = 0; i < HASH_ITER; i++) val[i] = (val[i] + a.val[i]) % M;
13        return *this;
14    }

```

```

15    HashValue &operator+=(const HashValue &a) {
16        for (int i = 0; i < HASH_ITER; i++) val[i] = (val[i] + a.val[i]) % M;
17        return *this;
18    }
19    HashValue &operator*=(const HashValue &a) {
20        for (int i = 0; i < HASH_ITER; i++) val[i] = ((__int128) val[i] * a.val[i]) % M;
21        return *this;
22    }
23    HashValue operator+(const HashValue &a) { return HashValue(*this) += a; }
24    HashValue operator-(const HashValue &a) { return HashValue(*this) -= a; }
25    HashValue operator*(const HashValue &a) { return HashValue(*this) *= a; }
26    bool operator<(const HashValue &a) const {
27        for (int i = 0; i < HASH_ITER; i++)
28            if (val[i] != a.val[i]) return val[i] < a.val[i];
29        return false;
30    }
31    bool operator==(const HashValue &a) const {
32        for (int i = 0; i < HASH_ITER; i++)
33            if (val[i] != a.val[i]) return false;
34        return true;
35    }
36 };
37
38 void setB(int n) {
39     if (B.size() == 0) {
40         HashValue v = 1;
41         B.push_back(v);
42         mt19937 rng(random_device{}());
43         for (int i = 0; i < HASH_ITER; i++)
44             v.val[i] = uniform_int_distribution<ll>(0, M - 1)(rng);
45         B.push_back(v);
46     }
47     while (B.size() <= n) B.push_back(B.back() * B[1]);
48 }
49
50 struct Hash {
51     vector<HashValue> p_hash;
52
53     Hash(const string &s) : p_hash(s.size() + 1) {
54         setB(s.size());
55         for (int i = 0; i < s.size(); i++)
56             p_hash[i + 1] = p_hash[i] * B[1] + s[i];
57     }
58
59     auto get_hash(int start, int end) {
60         return p_hash[end + 1] - p_hash[start] * B[end - start + 1];
61     }
62 };

```

31 Eulerian path/circuit in directed graphs

```

1 template<typename Edge>
2 class DirectedEulerian {
3 public:
4     int n, m;
5     vector<vector<pair<int, Edge>>> adj;
6
7     DirectedEulerian(int n, int m) : n(n), m(m) {
8         adj.assign(n, vector<pair<int, Edge>>());
9     }
10
11     void add_edge(int u, int v, Edge edge) {
12         adj[u].emplace_back(v, edge);
13     }
14
15     vector<Edge> path(bool circuit = false) {
16         vector<Edge> path;
17         int in = 0, out = 0;
18
19         calc_deg();
20         int start = -1, end = -1;
21         for (int i = 0; i < n; i++) {
22             if (indeg[i] > outdeg[i])
23                 in += indeg[i] - outdeg[i], end = i;
24             else if (indeg[i] < outdeg[i])
25                 out += outdeg[i] - indeg[i], start = i;
26         }
27
28         if (m == 0 || !((in == 0 && out == 0) || (in == 1 && out == 1 && !circuit))) {
29             return {};
30         }
31
32         if (start == -1) {
33             assert(end == -1);
34             for (int i = 0; i < n; i++) {
35                 if (outdeg[i] > 0) {
36                     start = end = i;
37                     break;

```

```

38     }
39 }
40 }
41
42 dfs(start, {}, path);
43
44 path.pop_back();
45 reverse(all(path));
46
47 return path;
48 }
49
50 private:
51 vector<int> indeg, outdeg;
52
53 void calc_deg() {
54     indeg.assign(n, 0);
55     outdeg.assign(n, 0);
56     for (int i = 0; i < n; i++) {
57         outdeg[i] = adj[i].size();
58         for (auto &j: adj[i]) indeg[j.first]++;
59     }
60 }
61
62 void dfs(int i, Edge e, vector<Edge> &path) {
63     while (outdeg[i] > 0)
64         outdeg[i]--, dfs(adj[i][outdeg[i]].first, adj[i][outdeg[i]].second, path);
65     path.push_back(e);
66 }
67 };

```

32 Eulerian path/circuit in undirected graphs

```

1 template<typename Edge>
2 class UndirectedEulerian {
3 public:
4     int n, m;
5     vector<vector<pair<int, Edge>>> adj; // NOTE: dont't add a self-edge twice
6
7     UndirectedEulerian(int n, int m) : n(n), m(m) {
8         adj.assign(n, vector<pair<int, Edge>>());
9     }
10
11     void add_edge(int u, int v, Edge edge) {
12         adj[u].emplace_back(v, edge);
13         adj[v].emplace_back(u, edge);
14     }
15
16     vector<Edge> path(bool circuit = false) {
17         vector<Edge> path;
18
19         cnt.clear();
20         calc_deg();
21         int start = -1, end = -1, odds = 0;
22         for (int i = 0; i < n; i++) {
23             if (deg[i] & 1) {
24                 odds++;
25                 if (!start)
26                     end = i;
27                 else
28                     start = i;
29             }
30         }
31
32         if (m == 0 || !(odds == 0 || (odds == 2 && !circuit))) {
33             return {};
34         }
35
36         if (start == -1) {
37             assert(end == -1);
38             for (int i = 0; i < n; i++) {
39                 if (deg[i] > 0) {
40                     start = end = i;
41                     break;
42                 }
43             }
44         }
45
46         dfs(start, -1, {}, path);
47
48         path.pop_back();
49         reverse(all(path));
50
51         return path;
52     }
53 }
54
55 private:

```

```

56 vector<int> deg;
57 map<pair<int, int>, int> cnt;
58
59 void calc_deg() {
60     deg.assign(n, 0);
61     for (int i = 0; i < n; i++) {
62         for (auto &j: adj[i]) {
63             deg[j.first]++;
64             if (i == j.first)
65                 deg[j.first]++;
66             if (i < j.first)
67                 cnt[{i, j.first}]++;
68         }
69     }
70 }
71
72 void dfs(int i, int p, Edge e, vector<Edge> &path) {
73     cnt[{min(i, p), max(i, p)}]--;
74     while (adj[i].size()) {
75         auto [j, E] = adj[i].back();
76         adj[i].pop_back();
77         if (cnt[{min(i, j), max(i, j)}] == 0) continue;
78         dfs(j, i, E, path);
79     }
80     path.push_back(e);
81 }
82 };

```

33 Mo's algorithm

```

1 int block_size;
2
3 struct MO {
4     struct Query {
5         int l, r, idx;
6         Query(int l, int r, int idx) : l(l), r(r), idx(idx) {}
7         bool operator<(const Query &q) const {
8             if (l / block_size != q.l / block_size)
9                 return pair(l, r) < pair(q.l, q.r);
10             return (l / block_size & 1) ? (r < q.r) : (r > q.r);
11         }
12     };
13
14     vector<int> arr;
15     vector<Query> queries;
16
17     MO(vector<int> &arr, vector<Query> &queries) : arr(arr), queries(queries) {}
18
19     int l = 0, r = -1;
20
21     void set_range(Query &q) {
22         // [l, r] inclusive
23         while (l > q.l) add(arr[--l]);
24         while (r < q.r) add(arr[++r]);
25         while (l < q.l) remove(arr[l++]);
26         while (r > q.r) remove(arr[r--]);
27     }
28
29     void add(int x) {
30     }
31
32     void remove(int x) {
33     }
34
35     int getans(Query &q) {
36     }
37
38     vector<int> ans() {
39         block_size = arr.size() / sqrt(queries.size()) + 1;
40         vector<int> ans(queries.size());
41         sort(all(queries));
42
43         l = queries.front().l, r = queries.front().l - 1;
44         for (auto &q: queries) {
45             set_range(q);
46             ans[q.idx] = getans(q);
47         }
48
49         return ans;
50     }
51 };

```

34 Torjan's algorithm

```

1 struct SCC {
2     int N, ID = 0, COMP = 0;
3     vector<vector<int>> adj;
4     vector<int> id, comp, st;
5
6     SCC(const vector<vector<int>> &adj) : adj(adj), N(adj.size()) {
7         id.resize(N), comp = vector<int>(N, -1);
8         go();
9     }
10
11     void go() {
12         for (int i = 0; i < N; i++)
13             if (!id[i]) dfs(i);
14     }
15
16     int dfs(int i) {
17         int low = id[i] = ++ID;
18         st.push_back(i);
19         for (int j: adj[i])
20             if (comp[j] == -1)
21                 // id[j] != 0 -> in stack, don't dfs
22                 low = min(low, id[j] ? dfs(j));
23         if (low == id[i]) {
24             COMP++;
25             for (int j = -1; j != i; j = st.back())
26                 comp[j] = st.back() = COMP, st.pop_back();
27         }
28         return low;
29     }
30 };

```

35 Kmp string algorithm

```

1 vector<int> KMP(const string &a, const string &b) {
2     // search for b in a
3     vector<int> ans;
4     int n = a.length(), m = b.length();
5     int b_table[n];
6     b_table[0] = 0;
7
8     for (int i = 1, k = 0; i < m; i++) {
9         while (k > 0 && b[k] != b[i])
10             k = b_table[k - 1];
11         k += b[i] == b[k];
12         b_table[i] = k;
13     }
14
15     for (int i = 0, k = 0; i < n; i++) {
16         while (k > 0 && b[k] != a[i])
17             k = b_table[k - 1];
18         k += b[k] == a[i];
19         if (k == m) {
20             k = b_table[k - 1];
21             ans.push_back(i - m + 1);
22         }
23     }
24
25     return ans;
26 }

```

36 Z algorithm for strings

```

1 vector<int> zfunction(string s) {
2     int n = s.size();
3
4     vector<int> z(n);
5     for (int i = 1, l = 1, r = 1; i < n; i++) {
6         if (i < r) z[i] = min(z[i - l], r - i);
7         while (i + z[i] < n && s[i + z[i]] == s[z[i]]) z[i]++;
8         if (i + z[i] > r) r = i + z[i], l = i;
9     }
10
11     return z;
12 }

```

37 Random utils

```

1 mt19937 rng = mt19937(random_device() {});
2

```

```

3 void seed(int s) { rng = mt19937(s); }
4
5 int rand_int(int x, int y) {
6     return uniform_int_distribution<int>(x, y)(rng);
7 }

```

38 Least common ancestor using binary lifting

```

1 struct LCA {
2     int n, LOG;
3     vector<int> depth;
4     vector<vector<int>> up, adj;
5
6     LCA(int n, int root = 0) : n(n), LOG(log2(n) + 1) {
7         adj.resize(n), depth.resize(n);
8         up.assign(n, vector<int>(LOG, root));
9     }
10
11     void add_edge(int u, int v) {
12         adj[u].push_back(v);
13         adj[v].push_back(u);
14     }
15
16     void dfs(int u, int p) {
17         for (auto v: adj[u]) {
18             if (v == p) continue;
19             up[v][0] = u;
20             depth[v] = depth[u] + 1;
21             dfs(v, u);
22         }
23     }
24
25     void build(int root = 0) {
26         dfs(root, root);
27         for (int k = 1; k < LOG; k++)
28             for (int u = 0; u < n; u++)
29                 up[u][k] = up[up[u][k - 1]][k - 1];
30     }
31
32     int query(int u, int v) const {
33         if (depth[u] < depth[v]) swap(u, v);
34         for (int k = LOG - 1; k >= 0; k--) {
35             if (depth[up[u][k]] >= depth[v]) {
36                 u = up[u][k];
37             }
38         }
39         if (u == v) return u;
40         for (int k = LOG - 1; k >= 0; k--) {
41             if (up[u][k] != up[v][k]) {
42                 u = up[u][k];
43                 v = up[v][k];
44             }
45         }
46         return up[u][0];
47     }
48 };

```

39 Least common ancestor using sparse table

```

1 struct LCA {
2     int n, LOG, _time;
3     vector<int> first, depth;
4     vector<vector<int>> adj, table;
5
6     LCA(int n) : n(n), LOG(log2(n) + 3) {
7         adj.resize(n), depth.resize(n), first.resize(n);
8         table.assign(LOG, vector<int>(2 * n));
9     }
10
11     void add_edge(int u, int v) {
12         adj[u].push_back(v);
13         adj[v].push_back(u);
14     }
15
16     void dfs(int u, int p) {
17         first[u] = _time;
18         table[0][_time++] = u;
19         for (auto v: adj[u]) {
20             if (v == p) continue;
21             depth[v] = depth[u] + 1;
22             dfs(v, u);
23             table[0][_time++] = u;
24         }
25     }
26 }

```

```

25     }
26
27     void build(int root = 0) {
28         _time = 0;
29         dfs(root, root);
30         assert(_time < 2 * n);
31         for (int i = 1; i < LOG; i++) {
32             for (int j = 0; j + (1 << i) <= 2 * n; j++) {
33                 if (depth[table[i - 1][j]] < depth[table[i - 1][j + (1 << (i - 1))]]) {
34                     table[i][j] = table[i - 1][j];
35                 } else {
36                     table[i][j] = table[i - 1][j + (1 << (i - 1))];
37                 }
38             }
39         }
40     }
41
42     int query(int u, int v) const {
43         u = first[u], v = first[v];
44         if (u == v) return table[0][u];
45         if (u > v) swap(u, v);
46         int lg = 31 - __builtin_clz(v - u + 1);
47         if (depth[table[lg][u]] < depth[table[lg][v - (1 << lg) + 1]]) {
48             return table[lg][u];
49         } else {
50             return table[lg][v - (1 << lg) + 1];
51         }
52     }
53 };

```

40 Centroid decomposition of a tree

```

1 struct Centroids {
2     vector<vector<int>>> edges;
3     vector<bool> removed;

```

```

4     vector<int> par;
5     vector<int> sz;
6     int n;
7
8     Centroids(int n) : n(n) {
9         edges.resize(n), removed.resize(n);
10        sz.resize(n), par.assign(n, -1);
11    }
12
13    void add_edge(int a, int b) {
14        edges[a].push_back(b);
15        edges[b].push_back(a);
16    }
17
18    void find_size(int v, int p = -1) {
19        sz[v] = 1;
20        for (int x: edges[v]) {
21            if (x == p || removed[x]) continue;
22            find_size(x, v), sz[v] += sz[x];
23        }
24    }
25
26    int find_centroid(int v, int p, int n) {
27        for (int x: edges[v]) {
28            if (x == p || removed[x]) continue;
29            if (sz[x] > n / 2) return find_centroid(x, v, n);
30        }
31        return v;
32    }
33
34    void build(int v = 0, int p = -1) {
35        find_size(v);
36        int c = find_centroid(v, -1, sz[v]);
37        removed[c] = true, par[c] = p;
38        for (int x: edges[c])
39            if (!removed[x]) build(x, c);
40    }
41 };

```