

C++ Snippets

Muhammad Samir Al-Sawalhy

Contents

August 12, 2023

1	Competitive programming template
2	Competitive programming template with multi-tests
3	Read an array of length n from the stdin
4	Brute force primality test
5	Miller & rabin probabilistic primality test
6	Prime factorization in $O(\sqrt{n})$
7	Euler's totient theorem
8	Sieve's algorithm to mark numbers as primes and composites
9	Dijkstra's tsp algorithm
10	Mst (minimum spanning tree), kruskal's algorithm
11	Computational geometry stuff for competitive programming
12	Stl policy container (oiset, omap)
13	Optimized segment tree with basic operations
14	Segment tree data structure
15	Modular arithmetics stolen from jiangly
16	Modular combinations
17	Disjoint set union
18	Matrix exponentiation
19	Fast input scanner
20	Pascal triangle, useful for combinations
21	Description
22	Modular inverse for coprimes not only prime mod
23	Trie data structure
24	String hashing implementation (polynomial hashing)
25	Eulerian path/circuit in directed graphs
26	Eulerian path/circuit in undirected graphs
27	Mo's algorithm

28	Torjan's algorithm	9
29	Kmp string algorithm	9
30	Random tree generator for stress testing	10
31	Random utils	10
32	Least common ancestor using binary lifting	10

1 Competitive programming template

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

2 Competitive programming template with multi-tests

```
1 //
2 #include <bits/stdc++.h>
3
4 using namespace std;
5
6 #ifdef SAWALHY
7 #include "debug.hpp"
8 #else
9 #define debug(...) 0
10 #define debug_itr(...) 0
11 #define debug_bits(...) 0
12 #endif
13
14 #define ll long long
15 #define int long long
16 #define all(v) v.begin(), v.end()
17
18 void solve() {
19     {}
20 }
21
22 int32_t main() {
23     ios_base::sync_with_stdio(false);
24     cin.tie(NULL), cout.tie(NULL);
25
26     int t;
27     cin >> t;
28     while (t--)
29         solve();
30
31     return 0;
32 }
```

3 Read an array of length n from the stdin

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

4 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 }
```

5 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    return true; // probably
15 }
16 }
```

6 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 sgn = sqrt(n);
14    for (int i = 3; i <= sgn; 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 }
```

7 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 }
```

8 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 }
```

9 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 }
```

10 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 }

```

11 Computational geometry stuff for competitive programming

```

1 namespace Geometry
2 {
3
4     using T = long long;
5     const T EPS = 0;
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    typedef struct Point {
26        T x, y;
27
28        Point() {}
29        Point(T _x, T _y) : x(_x), y(_y) {}
30        Point operator+(const Point &p) const { return Point(x + p.x, y + p.y); }
31        Point operator-(const Point &p) const { return Point(x - p.x, y - p.y); }
32        Point operator/(T denom) const { return Point(x / denom, y / denom); }
33        Point operator*(T scaler) const { return Point(x * scaler, y * scaler); }
34
35        T dot(const Point &p) const { return x * p.x + y * p.y; }
36        T cross(const Point &p) const { return x * p.y - y * p.x; }
37        T dot(const Point &a, const Point &b) const { return (a - *this).dot(b - *this); }
38        T cross(const Point &a, const Point &b) const { return (a - *this).cross(b - *this); }
39        T norm() const { return dot(*this); }
40
41        long double len() const { return sqrtl(dot(*this)); }
42        long double ang(bool pos = true) const {
43            auto a = atan2l(y, x);
44            if (pos && a < 0) a += PI * 2;
45            return a;
46        }
47
48        Point rotate(const Point &p, long double a) { return (*this - p).rotate(a) + p; }
49        Point rotate(long double angle) {
50            auto l = len(), a = ang();
51            return Point(l * cos(a + angle), l * sin(a + angle));
52        }
53
54        bool operator==(const Point &p) const { return (*this - p).norm() <= EPS; }
55        bool operator!=(const Point &p) const { return !(*this == p); }
56        bool operator<(const Point &p) const { return x < p.x || (x == p.x && y < p.y); }
57        friend ostream &operator<<(ostream &os, const Point &p) { return os << '(' << p.x << ', ' << p.y << ')'; }
58        friend istream &operator>>(istream &is, Point &p) { return is >> p.x >> p.y; }
59    } pt;
60
61    int ccw(const pt &a, pt b, pt c) {
62        if (a == b) return (a == c ? 0 : +3); // same point or different
63        b = b - a, c = c - a;
64        if (sign(b.cross(c)) == +1) return +1; // "COUNTER_CLOCKWISE"
65        if (sign(b.cross(c)) == -1) return -1; // "CLOCKWISE"
66        if (sign(b.dot(c)) == -1) return +2; // "ONLINE_BACK"
67        if (cmp(b.norm(), c.norm()) == -1) return -2; // "ONLINE_FRONT"
68        return 0; // "ON_SEGMENT"
69    }
70

```

```

71    pt slope(pt a, pt b, bool change_direction = true) {
72        assert(is_integral_v<T>);
73        long long dx = a.x - b.x;
74        long long dy = a.y - b.y;
75        if (dx == 0 && dy == 0) return pt(0, 0);
76        long long g = gcd(abs(dy), abs(dx));
77        dx /= g, dy /= g;
78        if (change_direction) {
79            if (dx < 0) dy *= -1, dx *= -1;
80            if (dx == 0) dy = abs(dy);
81        }
82        return pt(dx, dy);
83    }
84
85    struct Segment {
86        pt a, b;
87        Segment() {}
88        Segment(pt a, pt b) : a(a), b(b) {}
89        bool operator==(const Segment &s) const { return a == s.a ? b == s.b : a == s.b && b == s.a; };
90        friend istream &operator>>(istream &is, Segment &s) { return is >> s.a >> s.b; }
91        friend ostream &operator<<(ostream &os, const Segment &s) {
92            return os << "{" << s.a << ", " << s.b << "}";
93        }
94    };
95
96    struct Line : public Segment {
97        Line() {}
98        Line(pt a, pt b) : Segment(a, b) {}
99        bool operator==(const Line &l) const { return iseq0((a - b).cross(l.a - l.b)); };
100    };
101
102    struct Ray : public Segment {
103        Ray() {}
104        Ray(pt a, pt b) : Segment(a, b) {}
105        bool operator==(const Ray &r) const { return a == r.a && slope(a, b, false) == slope(r.a, r.b, false); };
106    };
107
108    struct Polygon {
109        int n;
110        vector<pt> vert;
111        Polygon() = default;
112        Polygon(int n) : n(n) { vert.resize(n); }
113        Polygon(vector<pt> &vert) : vert(vert), n(vert.size()) {}
114
115        T area2() const {
116            T a = 0;
117            for (int i = 2; i < n; i++)
118                a += vert[0].cross(vert[i], vert[i - 1]);
119            return abs(a);
120        }
121
122        long double area() const { return area2() / 2.0; };
123    };
124
125    bool parallel(const Line &a, const Line &b) { return (a.b - a.a).cross(b.b - b.a) == 0; }
126    bool orthogonal(const Line &a, const Line &b) { return (a.a - a.b).dot(b.a - b.b) == 0; }
127
128    bool intersect(const Line &l, const Line &m) { return !parallel(l, m); }
129    bool intersect(const pt &p, const Segment &s) { return ccw(s.a, s.b, p) == 0; }
130    bool intersect(const pt &p, const Line &l) { return abs(ccw(l.a, l.b, p)) != 1; }
131    bool intersect(const Segment &s, const Line &l) { return ccw(l.a, l.b, s.a) * ccw(l.a, l.b, s.b) != 1; }
132
133    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; }
134
135    bool intersect(const Segment &s, const Ray &r) {
136        auto d1 = (s.a - s.b).cross(r.b - r.a),
137            d2 = (s.a - r.a).cross(r.b - r.a),
138            d3 = (s.a - s.b).cross(s.a - r.a);
139        if (abs(d1) <= EPS)
140            return r.a.cross(r.b, s.a) == 0 &&
141                (r.a.dot(r.b, s.a) >= 0 || r.a.dot(r.b, s.b) >= 0); // NOT BACK
142        return sign(d1) * sign(d2) >= 0 && sign(d1) * sign(d3) >= 0 && abs(d2) <= abs(d1);
143    }
144
145    bool intersection(pt a, pt b, pt c, pt d, pt &inter) {
146        assert(is_floating_point_v<T>);
147        long double d1 = (a - b).cross(d - c);
148        long double d2 = (a - c).cross(d - b);
149        if (fabs(d1) <= EPS) return false;
150        long double t1 = d2 / d1;
151        inter = a + (b - a) * t1;
152        return true;
153    }
154
155    template<typename T, typename V>
156    bool intersection(const T &l, const V &m, pt &inter) {
157        if (!intersect(l, m)) return false;
158        return intersection(l.a, l.b, m.a, m.b, inter);
159    }

```

```

160 struct Circle {
161     pt c;
162     T r;
163
164     Circle() = default;
165     Circle(pt c, T r) : c(c), r(r) {}
166     Circle(const vector<pt> &p) {
167         if (p.size() == 1) c = p[0], r = 0;
168         else if (p.size() == 2) {
169             c = (p[0] + p[1]) / 2;
170             r = (p[0] - c).len();
171         } else {
172             assert(p.size() == 3);
173             *this = Circle(p[0], p[1], p[2]);
174         }
175     }
176
177     Circle(pt a, pt b, pt c) {
178         // if we have a cord in a circle,
179         // the perpendicular from the center will pass from the center
180         // so we simply solve for the intersection of two lines
181         auto ABmid = (a + b) / 2.0, BCmid = (b + c) / 2.0;
182         auto ABnorm = pt((a - b).y, -(a - b).x);
183         auto BCnorm = pt((b - c).y, -(b - c).x);
184         bool valid = intersection(
185             Line(ABmid, ABmid + ABnorm),
186             Line(BCmid, BCmid + BCnorm), this->c);
187         assert(valid); // unless at least two points are identical
188         r = (a - this->c).len();
189     }
190
191     friend bool intersect(const pt &p, const Circle &c) { return islte((p - c.c).norm(), c.r * c.r); }
192     friend ostream &operator<<(ostream &os, const Circle &c) {
193         return os << "c(" << c.c << ", " << c.r << ")";
194     }
195 };
196
197 int point_in_triangle(pt a, pt b, pt c, pt point) {
198     // point is on an edge or all are either 1 or -1
199     int x = ccw(a, b, point), y = ccw(b, c, point), z = ccw(c, a, point);
200     if (sign(x) == sign(y) && sign(y) == sign(z)) return 1;
201     if (x * y * z == 0) return 0;
202     return -1;
203 }
204
205 int point_in_circle(const pt &p, const vector<pt> &cir) {
206     if (cir.size() == 0) return -1;
207     auto c = Circle(cir);
208     if (iseq((p - c.c).norm(), c.r * c.r)) return 0;
209     if (intersect(p, c)) return 1;
210     return -1;
211 }
212
213 int point_in_polygon(const pt &p, const vector<pt> &polygon) {
214     int wn = 0, n = polygon.size();
215     for (int i = 0, j = 1; i < n; i++, j++, j %= n) {
216         if (ccw(polygon[j], polygon[i], p) == 0) return 0;
217         if ((p.y < polygon[j].y) != (p.y < polygon[i].y)) {
218             wn += polygon[j].y > polygon[i].y && ccw(p, polygon[i], polygon[j]) == 1;
219             wn -= polygon[j].y < polygon[i].y && ccw(p, polygon[j], polygon[i]) == 1;
220         }
221     }
222     return wn == 0 ? -1 : 1;
223 }
224
225 int ray_and_polygon(const Ray &r, const Polygon &polygon) {
226     // NOTE: Should be a good ray (a != b),
227     // and non-degenerate polygon with no duplicated points
228     int n = polygon.n, ans = -1;
229     for (int i = 0, j = 1, k = 2; i < n; i++, j++, k++, j %= n, k %= n) {
230         if (!intersect(Segment(polygon.vert[i], polygon.vert[j]), r)) continue;
231         auto x = r.a.cross(r.b, polygon.vert[i]);
232         auto y = r.a.cross(r.b, polygon.vert[j]);
233         auto z = r.a.cross(r.b, polygon.vert[k]);
234         if (x == 0) ans = 0; // Maybe tangent
235         else if (y == 0) {
236             // (the ray splits an internal angle)
237             // Entering from a vertex
238             if (sign(x) * sign(z) == -1) return 1;
239         } else return 1; // Entering from an edge
240     }
241     return ans;
242 }
243
244 vector<pt> &sort_clock(vector<pt> &points, bool cw = false) {
245     int n = points.size();
246
247     // choose the pivot (most bottom-right point)
248     for (int i = 1; i < n; i++) {
249         auto &l = points[0], &r = points[i];
250         int cy = cmp(l.y, r.y), cx = cmp(l.x, r.x);
251         if (cy == 0 ? cx == -1 : cy == +1) swap(l, r);
252     }
253
254     // sorting with points[0] as pivot
255     sort(points.begin() + 1, points.end(),
256         [&](pt l, pt r) {
257             auto c = ccw(points[0], l, r);
258             int cx = cmp(l.x, r.x), cy = cmp(l.y, r.y);
259             // closer to bottom-right comes first
260             if (abs(c) != 1) return cy == 0 ? cx == 1 : cy == -1;
261             return cw ? c == -1 : c == 1;
262         });
263
264     return points;
265 }
266
267 vector<pt> &sort_convex(vector<pt> &points, bool cw = false) {
268     int n = points.size();
269
270     // choose the pivot (most bottom-right point)
271     for (int i = 1; i < n; i++) {
272         auto &l = points[0], &r = points[i];
273         int cy = cmp(l.y, r.y), cx = cmp(l.x, r.x);
274         if (cy == 0 ? cx == -1 : cy == +1) swap(l, r);
275     }
276
277     // sorting with points[0] as pivot
278     sort(points.begin() + 1, points.end(),
279         [&](pt l, pt r) {
280             auto c = ccw(points[0], l, r);
281             int cx = cmp(l.x, r.x), cy = cmp(l.y, r.y);
282
283             if (abs(c) != 1) { // collinear
284                 if (cw) return cy == 0 ? cx == 1 : cy == -1;
285                 else
286                     return cy == 0 ? cx == -1 : cy == -1;
287             }
288
289             return cw ? c == -1 : c == 1;
290         });
291
292     return points;
293 }
294
295 vector<pt> convexhull(vector<pt> &p, bool strict = false) {
296     int n = p.size(), k = 0, sgn = strict ? 0 : -1;
297     if (n <= 2) return p;
298     vector<pt> ch(2 * n); // CCW
299     auto cmp = [](pt x, pt y) { return (x.x != y.x ? x.x < y.x : x.y < y.y); };
300     sort(begin(p), end(p), cmp);
301     for (int i = 0; i < n; ch[k++] = p[i++]) // lower hull
302     for (int i = n - 2, t = k + 1; i >= 0; ch[k++] = p[i--]) // upper hull
303     while (k >= t && sign((ch[k - 1] - ch[k - 2]).cross(p[i] - ch[k - 1])) <= sgn) --k;
304     while (k >= t && sign((ch[k - 1] - ch[k - 2]).cross(p[i] - ch[k - 1])) <= sgn) --k;
305     ch.resize(k - 1);
306     return ch;
307 }
308
309 struct PointInConvex {
310     int n;
311     vector<pt> seq;
312     pt translation;
313
314     PointInConvex(vector<pt> &polygon) { prepare_convex_ccw(polygon); }
315
316     void prepare_convex_ccw(vector<pt> &points) {
317         // NOTE: the polygon should be strictly convex
318         n = points.size();
319         int pos = 0; // most left-bottom point
320         for (int i = 1; i < n; i++)
321             if (points[i] < points[pos])
322                 pos = i;
323         rotate(points.begin(), points.begin() + pos, points.end());
324
325         seq.resize(n);
326         for (int i = 0; i < n; i++)
327             seq[i] = points[(i + 1) % n] - points[0];
328         translation = points[0];
329     }
330
331     int check(pt point) {
332         point = point - translation;
333         if (intersect(point, Segment(pt(0, 0), seq[0]))) return 0;
334         if (seq.size() <= 2) return -1;
335
336         int l = 0, r = n - 1;
337         while (r - l > 1) {
338             int mid = (l + r) / 2;
339             if (sign(seq[mid].cross(point)) != -1)
340                 l = mid;
341             else
342                 r = mid;
343         }
344     }

```

```

344     int ok = point_in_triangle(seq[l], seq[l + 1], pt(0, 0), point);
345     if (ok == -1) return -1;
346     if (intersect(point, Segment(seq[l], seq[l + 1]))) return 0;
347     return 1;
348 }
349 };
350
351 struct Welzl {
352     vector<pt> points;
353     Welzl(vector<pt> &points) : points(points) {
354         shuffle(all(points), default_random_engine(time(NULL)));
355     }
356 }
357
358 Circle get_circle() { return Circle(go()); }
359 vector<pt> go(int i = 0, vector<pt> cir = {}) {
360     if (cir.size() == 3 || i == (int) points.size()) return cir;
361     auto new_cir = go(i + 1, cir);
362     if (point_in_circle(points[i], new_cir) != -1)
363         return new_cir;
364     cir.push_back(points[i]);
365     return go(i + 1, cir);
366 }
367 };
368
369 }; // namespace Geometry
370
371 using namespace Geometry;

```

12 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>;

```

13 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 void relative_update(int i, T val) {
47     for (tree[i += n] += val; i > 1; i >>= 1)
48         tree[i >> 1] = tree[i] + tree[i ^ 1];
49 }
50
51 auto query(int l, int r) {
52     T res;
53     for (l += n, r += n + 1; l < r; l >>= 1, r >>= 1) {
54         if (l & 1) res += tree[l++];
55         if (r & 1) res += tree[--r];
56     }
57     return res.value;
58 }
59 };

```

14 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(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         int l = i << 1, r = i << 1 | 1;
108         if (tree[i].up.state != Update::idle) {
109             if (i < n) {
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 };

```

15 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
11 template<typename T, T mod, typename V = typename BiggerType<T>::type>
12 struct mint {
13 private:
14     inline T norm(T x) const {
15         if (x < 0) x += mod;
16         if (x >= mod) x -= mod;
17         return x;
18     }
19

```

```

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

```

16 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 }

```

17 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 };

```

18 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    }

```

```

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; 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 };

```

19 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 };

```

20 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 }

```

21 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==(const bigint &lhs, const bigint &rhs) {
82     return lhs.digits == rhs.digits;
83 }
84
85 friend bool operator!=(const bigint &lhs, const bigint &rhs) {
86     return lhs.digits != rhs.digits;
87 }
88
89 friend bool operator<(const bigint &lhs, const 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 };

```

22 Modular inverse for coprimes not only prime mod

```

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

```

23 Trie data structure

```

1 template<int MAX_SIZE = 26>
2 struct trie {
3     trie *child[MAX_SIZE];
4     int count = 0;
5     char value;
6     bool is_leaf = false;
7
8     trie() {
9         for (int i = 0; i < MAX_SIZE; i++)
10             child[i] = nullptr;
11     }
12
13     ~trie() {
14         for (int i = 0; i < MAX_SIZE; i++) {
15             if (child[i] == nullptr) continue;
16             delete child[i];
17         }
18     }
19
20     trie *insert(const char *str) {
21         count++;
22         if (*str == '\0') {
23             is_leaf = true;
24             return this;
25         }
26
27         int cur = *str - 'a';
28         if (child[cur] == nullptr) {
29             child[cur] = new trie();
30             child[cur]->value = *str;
31         }
32
33         return child[cur]->insert(str + 1);
34     }
35 };

```

24 String hashing implementation (polynomial hashing)

```

1 class hashed_string {
2 public:
3     // change M and B if you want
4     static const ll M = (1LL << 61) - 1;
5     static const ll B;
6
7 private:
8     // pow[i] contains P^i % M
9     static vector<mint<ll, M>> pow;
10    // hash of the prefixes
11    vector<mint<ll, M>> p_hash;
12
13 public:
14     hashed_string(const string &s) : p_hash(s.size() + 1) {
15         while (pow.size() < (int) s.size())
16             pow.push_back(pow.back() * B);
17         for (int i = 0; i < s.size(); i++)
18             p_hash[i + 1] = p_hash[i] * B + s[i];
19     }
20
21     auto get_hash(int start, int end) {
22         auto raw_val = p_hash[end + 1] - p_hash[start] * pow[end - start + 1];
23         return raw_val;
24     }
25 };
26

```



```

27 mt19937 rng((uint32_t) chrono::steady_clock::now().time_since_epoch().count());
28 vector<mint<ll, hashed_string::M>> hashed_string::pow = {1};
29 const ll hashed_string::B = uniform_int_distribution<ll>(0, M - 1)(rng);

```

25 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     DirectedEulerian(int n, int m, const vector<vector<pair<int, Edge>>> &adj) : adj(adj), n(n), m(m)
7     {}
8     vector<Edge> path(bool circuit = false) {
9         vector<Edge> path;
10        int in = 0, out = 0;
11
12        calc_deg();
13        int start = -1, end = -1;
14        for (int i = 0; i < n; i++) {
15            if (indeg[i] > outdeg[i])
16                in += indeg[i] - outdeg[i], end = i;
17            else if (indeg[i] < outdeg[i])
18                out += outdeg[i] - indeg[i], start = i;
19        }
20
21        if (m == 0 || !((in == 0 && out == 0) || (in == 1 && out == 1 && !circuit))) {
22            return {};
23        }
24
25        if (start == -1) {
26            assert(end == -1);
27            for (int i = 0; i < n; i++) {
28                if (outdeg[i] > 0) {
29                    start = end = i;
30                    break;
31                }
32            }
33        }
34
35        dfs(start, {}, path);
36
37        path.pop_back();
38        reverse(all(path));
39
40        return path;
41    }
42
43 private:
44     vector<int> indeg, outdeg;
45
46     void calc_deg() {
47         indeg.assign(n, 0);
48         outdeg.assign(n, 0);
49         for (int i = 0; i < n; i++) {
50             outdeg[i] = adj[i].size();
51             for (auto &j: adj[i]) indeg[j.first]++;
52         }
53     }
54
55     void dfs(int i, Edge e, vector<Edge> &path) {
56         while (outdeg[i] > 0)
57             outdeg[i]--, dfs(adj[i][outdeg[i]].first, adj[i][outdeg[i]].second, path);
58         path.push_back(e);
59     }
60 };

```

26 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: don't add a self-edge twice
6     UndirectedEulerian(int n, int m, const vector<vector<pair<int, Edge>>> &adj) : adj(adj), n(n), m(m)
7     {}
8     vector<Edge> path(bool circuit = false) {
9         vector<Edge> path;
10
11        cnt.clear();

```

```

12        calc_deg();
13        int start = -1, end = -1, odds = 0;
14        for (int i = 0; i < n; i++) {
15            if (deg[i] & 1) {
16                odds++;
17                if (~start)
18                    end = i;
19            else
20                start = i;
21        }
22
23        if (m == 0 || !(odds == 0 || (odds == 2 && !circuit))) {
24            return {};
25        }
26
27        if (start == -1) {
28            assert(end == -1);
29            for (int i = 0; i < n; i++) {
30                if (deg[i] > 0) {
31                    start = end = i;
32                    break;
33                }
34            }
35        }
36
37        dfs(start, -1, {}, path);
38
39        path.pop_back();
40        reverse(all(path));
41
42        return path;
43    }
44
45 private:
46     vector<int> deg;
47     map<pair<int, int>, int> cnt;
48
49     void calc_deg() {
50         deg.assign(n, 0);
51         for (int i = 0; i < n; i++) {
52             for (auto &j: adj[i]) {
53                 deg[j.first]++;
54                 if (i == j.first)
55                     deg[j.first]++;
56                 if (i < j.first)
57                     cnt[{i, j.first}]++;
58             }
59         }
60     }
61
62     void dfs(int i, int p, Edge e, vector<Edge> &path) {
63         cnt[{min(i, p), max(i, p)}]--;
64         while (adj[i].size()) {
65             auto [j, E] = adj[i].back();
66             adj[i].pop_back();
67             if (cnt[{min(i, j), max(i, j)}] == 0) continue;
68             dfs(j, i, E, path);
69         }
70         path.push_back(e);
71     }
72 }
73 };

```

27 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     return ans;
49 }
50
51 };

```

28 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;)
26                 comp[j = st.back()] = COMP, st.pop_back();
27         }
28         return low;
29     }
30 };

```

29 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     return ans;
25 }

```

30 Random tree generator for stress testing

```

1 vector<int> gen_tree_parents(int n, int root) {
2     // in a tree, each node other than the root
3     // has exactly one parent
4     assert(1 <= root && root <= n);
5     vector<int> parents(n + 1, -1);
6     vector<int> order(n + 1);
7     if (n == 1) return parents;
8     iota(all(order), 0);
9     shuffle(order.begin() + 1, order.end(),
10            default_random_engine(rand()));
11     swap(order[1], *find(all(order), root));
12     for (int i = 2; i <= n; i++)
13         parents[order[i]] = order[gen(1, i - 1)];
14     return parents;
15 }
16
17 vector<pair<int, int>> gen_tree(int n, int root = -1) {
18     if (root == -1) root = gen(1, n);
19     auto parents = gen_tree_parents(n, root);
20     vector<pair<int, int>> edges;
21     for (int i = 1; i <= n; i++) {
22         if (i == root) continue;
23         edges.emplace_back(i, parents[i]);
24     }
25     assert(edges.size() == n - 1);
26     return edges;
27 }

```

31 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 }

```

32 Least common ancestor using binary lifting

```

1 struct LCA {
2     int n, LOG;
3     vector<int> depth;
4     vector<vector<int>> parent, adj;
5
6     LCA() : n(0), LOG(-1e9) {}
7
8     LCA(const vector<vector<int>> &adj, int root = 0) : adj(adj), n(adj.size()), LOG(log2(n) + 1) {
9         depth.resize(n);
10        parent = vector<vector<int>>(n, vector<int>(LOG, root));
11        preprocess(root);
12    }
13
14    void dfs(int u, int p) {
15        for (auto v: adj[u]) {
16            if (v == p) continue;
17            parent[v][0] = u;
18            depth[v] = depth[u] + 1;
19            dfs(v, u);
20        }
21    }
22
23    void preprocess(int root) {
24        dfs(root, root);
25        for (int k = 1; k < LOG; k++)
26            for (int u = 0; u < n; u++)
27                parent[u][k] = parent[parent[u][k - 1]][k - 1];
28    }
29 }

```

```
30 int query(int u, int v) {
31     if (depth[u] < depth[v]) swap(u, v);
32     for (int k = LOG - 1; k >= 0; k--) {
33         if (depth[parent[u][k]] >= depth[v]) {
34             u = parent[u][k];
35         }
36     }
37     if (u == v) return u;
38     for (int k = LOG - 1; k >= 0; k--) {
```

```
39         if (parent[u][k] != parent[v][k]) {
40             u = parent[u][k];
41             v = parent[v][k];
42         }
43     }
44     return parent[u][0];
45 }
46 };
```