C++ Snippets

Muhammad Samir Al-Sawalhy

March 1, 2024

Contents

1 Competitive programming template

```cpp
//
#include <bits/stdc++.h>
using namespace std;

#ifdef SAWALHY
#include "debug.hpp"
#else
#define debug(...) 0
#define debug_itr(...) 0
#define debug_bits(...) 0
#endif

#define ll long long
#define int long long
#define all(v) v.begin(), v.end()
#define rall(v) v.rbegin(), v.rend()
#define minit(v, x) v = min(v, x)
#define maxit(v, x) v = max(v, x)

int32_t main() {
    ios_base::sync_with_stdio(false);
    cin.tie(NULL), cout.tie(NULL);

    ${0}

    return 0;
}
```

2 Competitive programming template with multi-tests

```cpp
//
#include <bits/stdc++.h>
using namespace std;

#ifdef SAWALHY
#include "debug.hpp"
#else
#define debug(...) 0
#define debug_itr(...) 0
#define debug_bits(...) 0
```

```cpp
#endif

#define ll long long
#define int long long
#define all(v) v.begin(), v.end()
#define rall(v) v.rbegin(), v.rend()
#define minit(v, x) v = min(v, x)
#define maxit(v, x) v = max(v, x)

void solve() {
    ${0}
}

int32_t main() {
    ios_base::sync_with_stdio(false);
    cin.tie(NULL), cout.tie(NULL);

    int t;
    cin >> t;
    while (t--)
        solve();

    return 0;
}
```

3 Increase the stack memory limit

```cpp
static void run_with_stack_size(void (*func)(void), size_t stsize) {
    char *stack, *send;
    stack = (char *)malloc(stsize);
    send = stack + stsize - 16;
    send = (char *)((uintptr_t)send / 16 * 16);
    asm volatile(
        "mov %%rsp, (%0)\n"
        "mov %0, %%rsp\n"
        :
        : "r"(send));
    func();
    asm volatile("mov (%0), %%rsp\n" : : "r"(send));
    free(stack);
}

int32_t main() {
    run_with_stack_size(main_, 1024 * 1024 * 1024 / 2); // run with a 512 MB stack
    return 0;
}
```

4 Read an array of length n from the stdin

```cpp
int n;
cin >> n;
vector<int> a(n);
for (int i = 0; i < n; i++) {
    cin >> a[i];
}
```

5 Brute force primality test

```cpp
bool is_prime(ll n) {
    if (n < 2) return false;
    if (n == 2) return true;
    if (n % 2 == 0) return false;
    for (ll i = 3; i * i <= n; i += 2)
        if (n % i == 0) return false;
    return true;
}
```

6 Miller & rabin probabilistic primality test

```cpp
bool miller_rabin_ptest(unsigned ll n, int k = 3) {
    if (n < 2) return false;
    if (n == 2) return true;

    while (k--) {
        unsigned ll a = 1LL * rand() * rand() % (n - 2) + 2;// [2 ... n-1]
        unsigned ll r = 1;
        for (unsigned ll p = n - 1; p; p >>= 1) {
            if (p & 1) r = r * a % n;
            a = a * a % n;
        }
        if (r != 1) return false;
    }

    return true;// probably
}
```

7 Prime factorization in o(sqrt(n))

```cpp
map<ll, ll> primefacts(ll n) {
    map<ll, ll> result;
    int r = 0;

    while (n % 2 == 0) {
        r++;
        n = n / 2;
    }

    if (r > 0)
        result[2] = r;

    int sqn = sqrt(n);
    for (int i = 3; i <= sqn; i += 2) {
        r = 0;
        while (n % i == 0) {
            r++;
            n = n / i;
        }
        if (r > 0)
            result[i] = r;
    }

    if (n > 2)
        result[n] = 1;

    return result;
}
```

8 Euler's totient theorm

```cpp
std::vector<int> phi(${1:n} + 1);
std::iota(phi.begin(), phi.end(), 0);

for (int i = 1; i <= ${2:$1}; i++) {
    for (int j = i << 1; j <= ${2:$1}; j += i)
        phi[j] -= phi[i];
}
```

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

```cpp
void sieve(vector<bool> &is_prime) {
    is_prime[1] = false;
    is_prime[0] = false;
    int s = is_prime.size();
    for (int i = 4; i < s; i += 2)
        is_prime[i] = false;
    for (int i = 3; i * i < s; i += 2) {
        if (is_prime[i]) {
            for (int j = i * i; j < s; j += i + i)
                is_prime[j] = false;
        }
    }
}
```

10 Fast sieve's algorithm to calc minimum prime

```
std::vector<int> minp, primes;
void sieve(int n) {
    minp.assign(n + 1, 0);
    primes.clear();

    for (int i = 2; i <= n; i++) {
        if (minp[i] == 0) {
            minp[i] = i;
            primes.push_back(i);
        }

        for (auto p : primes) {
            if (i * p > n) {
                break;
            }
            minp[i * p] = p;
            if (p == minp[i]) {
                break;
            }
        }
    }
}
```

11 Dijkstra's tsp algorithm

```
long long dijkstra(int s, int e, vector<vector<pair<int, int>>> &adj) {
    int n = adj.size();
    vector<int> prev(n + 1);
    vector<ll> dist(n + 1, 1e18);

    typedef pair<ll, int> item;
    priority_queue<item, deque<item>, greater<item>> qu;
    qu.push({0, s});
    dist[s] = 0;

    while (!qu.empty()) {
        auto [d, i] = qu.top();
        qu.pop();

        if (dist[i] < d) continue;
        for (auto [j, D]: adj[i]) {
            if (dist[j] > D + d) {
                prev[j] = i;
                dist[j] = D + d;
                qu.push({dist[j], j});
            }
        }
    }

    // for (int i = e; i != s; i = prev[i]);
    return dist[e];
}
```

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

```
struct Edge {
    int from, to;
    long long weight;
    Edge(int from, int to, long long weight) : from(from), to(to), weight(weight) {}
    bool operator<(Edge &e) { return weight < e.weight; }
};

pair<long long, vector<Edge>> mst_kruskal(vector<Edge> &edges, int n) {
    DSU uf(n + 1);
    double cost = 0;
    vector<Edge> mst_edges;

    sort(edges.rbegin(), edges.rend());

    while (!edges.empty()) {
        auto &e = edges.back();
        edges.pop_back();
        if (uf.uni(e.from, e.to)) {
            cost += e.weight;
            mst_edges.push_back(e);
        }
    };

    if (mst_edges.size() != n - 1)
        return {1e18, {}};

    return {cost, mst_edges};
}
```

13 Computational geometry stuff for competitive prgramming

```
namespace Geometry
{

using T = long double;
const T EPS = 1e-8;
const double PI = acos(-1.0);

template<typename T, typename V>
int cmp(T a, V b) { return (a -= b) < -EPS ? -1 : (a > EPS ? 1 : 0); }
template<typename T, typename V>
bool iseq(T a, V b) { return cmp(a, b) == 0; }
template<typename T>
bool iseq0(T a) { return cmp(a, 0) == 0; }
template<typename T, typename V>
bool islte(T a, V b) { return cmp(a, b) != 1; }
template<typename T, typename V>
bool isgte(T a, V b) { return cmp(a, b) != -1; }
template<typename T, typename V>
bool islt(T a, V b) { return cmp(a, b) == -1; }
template<typename T, typename V>
bool isgt(T a, V b) { return cmp(a, b) == 1; }
template<typename T>
int sign(T val) { return cmp(val, 0); }

enum PointState { OUT,
                  IN,
                  ON };

typedef struct Point {
    T x, y;

    Point() {}
    Point(T _x, T _y) : x(_x), y(_y) {}
    Point operator+(const Point &p) const { return Point(x + p.x, y + p.y); }
    Point operator-(const Point &p) const { return Point(x - p.x, y - p.y); }
    Point operator/(T denom) const { return Point(x / denom, y / denom); }
    Point operator*(T scaler) const { return Point(x * scaler, y * scaler); }

    T dot(const Point &p) const { return x * p.x + y * p.y; }
    T cross(const Point &p) const { return x * p.y - y * p.x; }
    T dot(const Point &a, const Point &b) const { return (a - *this).dot(b - *this); }
    T cross(const Point &a, const Point &b) const { return (a - *this).cross(b - *this); }
    T norm() const { return dot(*this); }

    long double len() const { return sqrtl(dot(*this)); }
    long double ang(bool pos = true) const {
        auto a = atan2l(y, x);
        if (pos && a < 0) a += PI * 2;
        return a;
    }

    Point rotate(const Point &p, long double a) { return (*this - p).rotate(a) + p; }
    Point rotate(long double angle) {
        auto l = len(), a = ang();
        return Point(l * cos(a + angle), l * sin(a + angle));
    }

    bool operator==(const Point &p) const { return (*this - p).norm() <= EPS; }
    bool operator!=(const Point &p) const { return !(*this == p); }
    bool operator<(const Point &p) const { return x < p.x || (x == p.x && y < p.y); }
    friend ostream &operator<<(ostream &os, const Point &p) { return os << '(' << p.x << ',' << p.y <<
            ')'; }
    friend istream &operator>>(istream &is, Point &p) { return is >> p.x >> p.y; }
} pt;

int ccw(const pt &a, pt b, pt c) {
    if (a == b) return (a == c ? 0 : +3); // same point or different
    b = b - a, c = c - a;
    if (sign(b.cross(c)) == +1) return +1;        // "COUNTER_CLOCKWISE"
    if (sign(b.cross(c)) == -1) return -1;        // "CLOCKWISE"
    if (sign(b.dot(c)) == -1) return +2;          // "ON_RAY_b_a)"
    if (cmp(b.norm(), c.norm()) == -1) return -2; // "ON_RAY_a_b"
    return 0;                                     // "ON_SEGMENT"
}

bool colinear(const pt &a, const pt &b, const pt &c) {
    return abs(ccw(a, b, c)) != 1;
}

```

```
pt slope(pt a, pt b, bool change_direction = true) {
    assert(is_integral_v<T>);
    long long dx = a.x - b.x;
    long long dy = a.y - b.y;
    if (dx == 0 && dy == 0) return pt(0, 0);
    long long g = gcd(abs(dy), abs(dy));
    dx /= g, dy /= g;
    if (change_direction) {
        if (dx < 0) dy *= -1, dx *= -1;
        if (dx == 0) dy = abs(dy);
    }
    return pt(dx, dy);
}

struct Segment {
    pt a, b;
    Segment() {}
    Segment(pt a, pt b) : a(a), b(b) {}
    bool operator==(const Segment &s) const { return a == s.a ? b == s.b : a == s.b && b == s.a; };
    friend istream &operator>>(istream &is, Segment &s) { return is >> s.a >> s.b; }
    friend ostream &operator<<(ostream &os, const Segment &s) {
        return os << "{" << s.a << ", " << s.b << "}";
    }
};

struct Line : public Segment {
    Line() {}
    Line(pt a, pt b) : Segment(a, b) {}
    bool operator==(const Line &l) const { return iseq0((a - b).cross(l.a - l.b)); };
};

struct Ray : public Segment {
    Ray() {}
    Ray(pt a, pt b) : Segment(a, b) {}
    bool operator==(const Ray &r) const { return a == r.a && slope(a, b, false) == slope(r.a, r.b,
          false); };
};

struct Polygon {
    int n;
    vector<pt> verts;
    Polygon() = default;
    Polygon(int n) : n(n) { verts.resize(n); }
    Polygon(vector<pt> &vert) : verts(vert), n(vert.size()) {}

    T area2() const {
        T a = 0;
        for (int i = 2; i < n; i++)
            a += verts[0].cross(verts[i], verts[i - 1]);
        return abs(a);
    }

    long double area() const { return area2() / 2.0; };

    void no_collinear() {
        vector<pt> v;
        for (int i = 0; i <= n; i++) {
            while (v.size() > 1 && colinear(v.back(), v.end()[-2], verts[i % n]))
                v.pop_back();
            v.push_back(verts[i % n]);
        }
        v.pop_back();
        n = v.size();
        verts = v;
        assert(n > 2);
    }

    void ensure_ccw() {
        start_bottom_left();
        if (ccw(verts[0], verts[1], verts.back()) == -1)
            reverse(verts.begin() + 1, verts.end());
    }

    void start_bottom_left() {
        int pos = 0; // most left-bottom point
        for (int i = 1; i < n; i++)
            if (verts[i] < verts[pos])
                pos = i;
        rotate(verts.begin(), verts.begin() + pos, verts.end());
    }
};

bool parallel(const Line &a, const Line &b) { return (a.b - a.a).cross(b.b - b.a) == 0; }
bool orthogonal(const Line &a, const Line &b) { return (a.a - a.b).dot(b.a - b.b) == 0; }

bool intersect(const Line &l, const Line &m) { return !parallel(l, m); }

bool intersect(const pt &p, const Segment &s) { return ccw(s.a, s.b, p) == 0; }
bool intersect(const Segment &s, const pt &p) { return intersect(p, s); }

bool intersect(const pt &p, const Line &l) { return abs(ccw(l.a, l.b, p)) != 1; }
bool intersect(const Line &l, const pt &p) { return intersect(p, l); }

bool intersect(const Segment &s, const Line &l) { return ccw(l.a, l.b, s.a) * ccw(l.a, l.b, s.b) != 1;
      }
bool intersect(const Line &l, const Segment &s) { return intersect(s, l); }

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; }

bool intersect(const Segment &s, const Ray &r) {
    auto d1 = (s.a - s.b).cross(r.b - r.a),
         d2 = (s.a - r.a).cross(r.b - r.a),
         d3 = (s.a - s.b).cross(s.a - r.a);
    if (abs(d1) <= EPS)
        return r.a.cross(r.b, s.a) == 0 &&
               (r.a.dot(r.b, s.a) >= 0 || r.a.dot(r.b, s.b) >= 0); // NOT BACK
    return sign(d1) * sign(d2) >= 0 && sign(d1) * sign(d3) >= 0 && abs(d2) <= abs(d1);
}

bool intersect(const Ray &r, const Segment &s) { return intersect(s, r); }

bool intersection(pt a, pt b, pt c, pt d, pt &inter) {
    assert(is_floating_point_v<T>);
    long double d1 = (a - b).cross(d - c);
    long double d2 = (a - c).cross(d - c);
    if (fabs(d1) <= EPS) return false;
    long double t1 = d2 / d1;
    inter = a + (b - a) * t1;
    return true;
}

template<typename T, typename V>
bool intersection(const T &l, const V &m, pt &inter) {
    if (!intersect(l, m)) return false;
    return intersection(l.a, l.b, m.a, m.b, inter);
}

// - NOTE: The polygon shouldn't have collinear points.
// - NOTE: First vertex should be the bottom-left, points in ccw order.
vector<pt> intersection(const Polygon &poly, const Line &line) {
    int n = poly.n;
    vector<pt> inter;
    const vector<pt> &verts = poly.verts;

    pt x;
    for (int i = 1; i <= n; i++) {
        int I = i % n, J = i - 1, K = (i - 2 + n) % n;
        if (intersection(line, Segment(verts[I], verts[J]), x)) {
            if (x == verts[I]) continue;
            if (x != verts[J]) {
                inter.push_back(x);
                continue;
            }
            int dir1 = ccw(line.a, line.b, verts[I]);
            int dir2 = ccw(line.a, line.b, verts[K]);
            if (dir1 * dir2 == -1)
                // entering or leaving from a vertex
                inter.push_back(verts[J]);
        } else if (abs(ccw(line.a, line.b, verts[J])) != 1) {
            // side (I, J) is on the line
            bool isWideAngleI = islt(ccw(verts[I], verts[((I + 1) % n)], verts[J]), 0);
            bool isWideAngleJ = islt(ccw(verts[J], verts[I], verts[K]), 0);
            if (isWideAngleI) inter.push_back(verts[I]);
            if (isWideAngleJ) inter.push_back(verts[J]);
            inter.push_back(verts[I]);
            inter.push_back(verts[J]);
        }
    }

    debug(inter);

    // sort in one direction, as if you travel on the line
    // in this direction and see the points one by one
    // NOTE: points may NOT be eaxctly on the line due to precesion errors
    sort(all(inter), [&](pt l, pt r) {
        return sign((line.b - line.a).dot(r - l)) == 1;
    });

    assert(inter.size() % 2 == 0);
    return inter;
};

struct Circle {
    pt c;
    T r;

    Circle() = default;
    Circle(pt c, T r) : c(c), r(r) {}
    Circle(const vector<pt> &p) {
        if (p.size() == 1) c = p[0], r = 0;
        else if (p.size() == 2) {
            c = (p[0] + p[1]) / 2;
            r = (p[0] - c).len();
        } else {
            assert(p.size() == 3);
            *this = Circle(p[0], p[1], p[2]);
        }
    }

    Circle(pt a, pt b, pt c) {
        // if we have a cord in a circle,
        // the perpendicular from the center will pass from the center
        // so we simply solve for the interection of two lines
        auto ABmid = (a + b) / 2.0, BCmid = (b + c) / 2.0;
        auto ABnorm = pt((a - b).y, -(a - b).x);
        auto BCnorm = pt((b - c).y, -(b - c).x);
        bool valid = intersection(
                Line(ABmid, ABmid + ABnorm),
                Line(BCmid, BCmid + BCnorm), this->c);
        assert(valid); // unless at least two points are identical
        r = (a - this->c).len();
    }

    friend ostream &operator<<(ostream &os, const Circle &c) {
        return os << "c{" << c.c << ", " << c.r << "}";
    }
};

PointState point_in_triangle(pt a, pt b, pt c, pt point) {
    int x = ccw(a, b, point), y = ccw(b, c, point), z = ccw(c, a, point);
    if (sign(x) == sign(y) && sign(y) == sign(z)) return IN;
    if (x * y * z == 0) return ON;
    return OUT;
}

PointState point_in_circle(const pt &p, const vector<pt> &cir) {
    if (cir.size() == 0) return OUT;
    auto c = Circle(cir);
    if (iseq((p - c.c).norm(), c.r * c.r)) return ON;
    if (islt((p - c.c).norm(), c.r * c.r)) return IN;
    return OUT;
}

PointState point_in_polygon(const pt &p, const vector<pt> &polygon) {
    int wn = 0, n = polygon.size();
    for (int i = 0, j = 1; i < n; i++, j++, j %= n) {
        if (ccw(polygon[j], polygon[i], p) == 0) return ON;
        if ((p.y < polygon[j].y) != (p.y < polygon[i].y)) {
            wn += polygon[j].y > polygon[i].y && ccw(p, polygon[i], polygon[j]) == 1;
            wn -= polygon[j].y < polygon[i].y && ccw(p, polygon[j], polygon[i]) == 1;
        }
    }
    return wn == 0 ? OUT : IN;
}

PointState ray_and_polygon(const Ray &r, const Polygon &polygon) {
    // NOTE: Should be a good ray (a != b),
    // and non-degenerate polygon with no duplicated points
    int n = polygon.n;
    PointState ans = OUT;
    for (int i = 0, j = 1, k = 2; i < n; i++, j++, k++, j %= n, k %= n) {
        if (!intersect(Segment(polygon.verts[i], polygon.verts[j]), r)) continue;
        auto x = r.a.cross(r.b, polygon.verts[i]);
        auto y = r.a.cross(r.b, polygon.verts[j]);
        auto z = r.a.cross(r.b, polygon.verts[k]);
        if (x == 0) ans = ON; // Maybe tangent
        else if (y == 0) {
            // (the ray splits an internal angle)
            // Entering from a vertex
            if (sign(x) * sign(z) == -1) return IN;
        } else return IN; // Entering from an edge
    }
    return ans;
}

vector<pt> &sort_clock(vector<pt> &points, bool cw = false) {
    int n = points.size();

    // choose the pivot (most bottom-right point)
    for (int i = 1; i < n; i++) {
        auto &l = points[0], &r = points[i];
        int cy = cmp(l.y, r.y), cx = cmp(l.x, r.x);
        if (cy == 0 ? cx == -1 : cy == +1) swap(l, r);
    }

    // sorting with points[0] as pivot
    sort(points.begin() + 1, points.end(),
         [&](pt l, pt r) {
             auto c = ccw(points[0], l, r);
             int cx = cmp(l.x, r.x), cy = cmp(l.y, r.y);
             // closer to bottom-right comes first
             if (abs(c) != 1) return cy == 0 ? cx == 1 : cy == -1;
             return cw ? c == -1 : c == 1;
         });

    return points;
}

// sort a convex polygon cw or ccw with the bottom-right as the pivot
vector<pt> &sort_convex(vector<pt> &points, bool cw = false) {
    int n = points.size();

    // choose the pivot (most bottom-right point)
    for (int i = 1; i < n; i++) {
        auto &l = points[0], &r = points[i];
        int cy = cmp(l.y, r.y), cx = cmp(l.x, r.x);
        if (cy == 0 ? cx == -1 : cy == +1) swap(l, r);
    }

    // sorting with points[0] as pivot
    sort(points.begin() + 1, points.end(),
         [&](pt l, pt r) {
             auto c = ccw(points[0], l, r);
             int cx = cmp(l.x, r.x), cy = cmp(l.y, r.y);

             if (abs(c) != 1) { // collinear
                 if (cw) return cy == 0 ? cx == 1 : cy == 1;
                 else
                     return cy == 0 ? cx == -1 : cy == -1;
             }

             return cw ? c == -1 : c == 1;
         });

    return points;
}

vector<pt> convexhull(vector<pt> &p, bool strict = false) {
    int n = p.size(), k = 0, sgn = strict ? 0 : -1;
    if (n <= 2) return p;
    vector<pt> ch(2 * n); // CCW
    auto cmp = [](pt x, pt y) { return (x.x != y.x ? x.x < y.x : x.y < y.y); };
    sort(begin(p), end(p), cmp);
    for (int i = 0; i < n; ch[k++] = p[i++]) // lower hull
        while (k >= 2 && sign((ch[k - 1] - ch[k - 2]).cross(p[i] - ch[k - 1])) <= sgn) --k;
    for (int i = n - 2, t = k + 1; i >= 0; ch[k++] = p[i--]) // upper hull
        while (k >= t && sign((ch[k - 1] - ch[k - 2]).cross(p[i] - ch[k - 1])) <= sgn) --k;
    ch.resize(k - 1);
    return ch;
}

struct PointInConvex {
    int n;
    vector<pt> seq;
    pt translation;

    PointInConvex(vector<pt> polygon) { prepare_convex_ccw(polygon); }

    void prepare_convex_ccw(vector<pt> &points) {
        // NOTE: the polygon should be strictly convex
        n = points.size();
        int pos = 0; // most left-bottom point
        for (int i = 1; i < n; i++)
            if (points[i] < points[pos])
                pos = i;
        rotate(points.begin(), points.begin() + pos, points.end());

        seq.resize(n);
        for (int i = 0; i < n; i++)
            seq[i] = points[(i + 1) % n] - points[0];
        translation = points[0];
    }

    int check(pt point) {
        point = point - translation;
        if (intersect(point, Segment(pt(0, 0), seq[0]))) return 0;
        if (seq.size() <= 2) return -1;

        int l = 0, r = n - 1;
        while (r - l > 1) {
            int mid = (l + r) / 2;
            if (sign(seq[mid].cross(point)) != -1)
                l = mid;
            else
                r = mid;
        }

        int ok = point_in_triangle(seq[l], seq[l + 1], pt(0, 0), point);
        if (ok == -1) return -1;
        if (intersect(point, Segment(seq[l], seq[l + 1]))) return 0;
        return 1;
    }
};

struct Welzl {
    vector<pt> points;
    Welzl(vector<pt> &_points) : points(_points) {
```

```cpp
        shuffle(all(points), default_random_engine(time(NULL)));
    }

    Circle get_circle() { return Circle(go()); }
    vector<pt> go(int i = 0, vector<pt> cir = {}) {
        if (cir.size() == 3 || i == (int) points.size()) return cir;
        auto new_cir = go(i + 1, cir);
        if (point_in_circle(points[i], new_cir) != OUT)
            return new_cir;
        cir.push_back(points[i]);
        return go(i + 1, cir);
    }
};

}; // namespace Geometry

using namespace Geometry;
```

14 Stl policy container (oset, omap)

```cpp
#include<ext/pb_ds/assoc_container.hpp>
#include<ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;
template<typename T>
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

```cpp
template<typename T = long long>
struct Sum {
    T value;
    Sum(T value = 0) : value(value) {}
    Sum &operator+=(const Sum &other) { return value += other.value, *this; }
    Sum operator+(const Sum &other) const { return value + other.value; }
};

template<typename T = long long>
struct Max {
    T value;
    Max(T value = numeric_limits<T>::min() / 2) : value(value) {}
    Max &operator+=(const Max &other) { return value = max(value, other.value), *this; }
    Max operator+(const Max &other) const { return Max(max(value, other.value)); }
};

template<typename T = long long>
struct Min {
    T value;
    Min(T value = numeric_limits<T>::max() / 2) : value(value) {}
    Min &operator+=(const Min &other) { return value = min(value, other.value), *this; }
    Min operator+(const Min &other) const { return Min(min(value, other.value)); }
};

// source: https://codeforces.com/blog/entry/18051
template<typename T>
struct Segtree {
    int n;
    vector<T> tree;

    Segtree() = default;
    Segtree(int n) : n(n) {
        tree.resize(n * 2);
    }

    void build() {
        for (int i = n - 1; i > 0; --i)
            tree[i] = tree[i << 1] + tree[i << 1 | 1];
    }

    void update(int i, T val) {
        for (tree[i += n] = val; i > 1; i >>= 1)
            tree[i >> 1] = tree[i] + tree[i ^ 1];
    }

    auto query(int l, int r) {
        T res;
        for (l += n, r += n + 1; l < r; l >>= 1, r >>= 1) {
            if (l & 1) res += tree[l++];
            if (r & 1) res += tree[--r];
        }
        return res.value;
    }
};
```

16 Segment tree data structure

```cpp
struct Value;
struct Update;
struct Node;

// Replaceable by primitives (using Value = long long)
struct Value {
    long long sum = 0, mn = 1e18, mx = -1e18;
    Value() = default;
    Value(ll value) { sum = mn = mx = value; }

    Value &operator+=(const Value &other) {
        sum += other.sum;
        mn = min(mn, other.mn);
        mx = max(mx, other.mx);
        return *this;
    }

    Value operator+(const Value &other) const {
        return Value(*this) += other;
    }
};

struct Update {
    // NOTE: Sometime you need to split the update, in these cases
    // you should include the range [a, b] of the update in the struct Update
    Value value;
    enum State {
        idle,
        relative,
        forced
    } state = idle;

    Update() = default;
    Update(Value value, State state = forced) : value(value), state(state){};

    Update &operator+=(const Update &other) {
        if (state == idle || other.state == forced) {
            *this = other;
        } else {
            assert(other.state == relative);
            value += other.value;
        }
        return *this;
    }

    void apply_on(Value &other, int cnt) const {
        if (state == forced) other = value;
        else other += value;
        other.sum += value.sum * (cnt - 1);
    }

    Update get(const Node &node) const { return *this; }
};

struct Node {
    int l = -1, r = -1; // [l, r]
    Update up;
    Value value;

    Node() = default;
    Node(int l, int r, const Value &value) : l(l), r(r), value(value){};

    void update(const Update &up) { this->up += up; }

    void apply_update() {
        up.apply_on(value, r - l + 1);
        up.state = Update::idle;
    }
};

struct Segtree {
    int n;
    vector<Node> tree;

    Segtree(int n) {
        if ((n & (n - 1)) != 0)
            n = 1 << (32 - __builtin_clz(n));
        this->n = n;
        tree.assign(n << 1, Node());
        for (int i = n; i < n << 1; i++)
            tree[i].l = tree[i].r = i - n;
        for (int i = n - 1; i > 0; i--)
            tree[i].l = tree[i << 1].l, tree[i].r = tree[i << 1 | 1].r;
    }

    Segtree(const vector<Value> &values) : Segtree(values.size()) {
        for (int i = 0; i < (int) values.size(); i++)
```

```cpp
            tree[i + n].value = values[i];
        build();
    }

    void build() {
        for (int i = n - 1; i > 0; --i) pull(i);
    }

    inline Value query(int i) { return query(1, i, i); }
    inline Value query(int i, int j) { return query(1, i, j); }
    inline void update(int i, const Update &val) { update(1, i, i, val); }
    inline void update(int i, int j, const Update &val) { update(1, i, j, val); }

private:
    void pull(int i) {
        tree[i].value = tree[i << 1].value + tree[i << 1 | 1].value;
    }

    void push(int i) {
        if (tree[i].up.state != Update::idle) {
            if (i < n) {
                int l = i << 1, r = i << 1 | 1;
                tree[l].update(tree[i].up.get(tree[l]));
                tree[r].update(tree[i].up.get(tree[r]));
            }
            tree[i].apply_update();
        }
    }

    Value query(int i, int l, int r) {
        push(i);
        if (tree[i].r < l || r < tree[i].l) return Value(); // default
        if (l <= tree[i].l && tree[i].r <= r) return tree[i].value;
        return query(i << 1, l, r) + query(i << 1 | 1, l, r);
    }

    void update(int i, int l, int r, const Update &up) {
        push(i);
        if (tree[i].r < l || r < tree[i].l) return;
        if (l <= tree[i].l && tree[i].r <= r) {
            tree[i].update(up);
            push(i); // to apply the update
            return;
        }
        update(i << 1, l, r, up.get(tree[i << 1]));
        update(i << 1 | 1, l, r, up.get(tree[i << 1 | 1]));
        pull(i);
    }
};
```

17 Persistent segtree

```cpp
const int MIN = -(1 << 30), MAX = (1 << 30) - 1;
const int DEFAULT = 0;

struct Node {
    long long value = DEFAULT;
    Node *l = nullptr, *r = nullptr;
    Node() = default;
    Node(Node *l, Node *r) : l(l), r(r) {
        if (l) value += l->value;
        if (r) value += r->value;
    }
};

struct PersistentSegtree {
    Node *root = nullptr;

    Node *update(int x, long long value) {
        return root = update(x, value, root, MIN, MAX);
    }

    long long query(int L, int R) {
        return query(L, R, root, MIN, MAX);
    }

private:
    Node *update(int x, long long value, Node *node, int l, int r) {
        if (l == r) {
            assert(l == x);
            Node *ret = new Node();
            ret->value = value;
            return ret;
        }

        int mid = l + (r - l) / 2;
        Node *left = node ? node->l : nullptr;
        Node *right = node ? node->r : nullptr;
        if (x <= mid) return new Node(update(x, value, left, l, mid), right);
        else return new Node(left, update(x, value, right, mid + 1, r));
    }

    long long query(int L, int R, Node *node, int l, int r) {
        if (!node || L > r || R < l) return DEFAULT;
        if (L <= l && r <= R) return node->value;
        int mid = l + (r - l) / 2;
        return query(L, R, node->l, l, mid) + query(L, R, node->r, mid + 1, r);
    }
};
```

18 N-d binary indexed tree

```cpp
template<class T, int... Ns>
struct BIT {
    T val = 0;
    void update(T v) { val += v; }
    T query() { return val; }
};

template<class T, int N, int... Ns>
struct BIT<T, N, Ns...> {
    BIT<T, Ns...> bit[N + 1];
    template<typename... Args>
    void update(int pos, Args... args) {
        for (pos++; pos <= N; pos += (pos & -pos)) bit[pos].update(args...);
    }
    template<typename... Args>
    T sum(int r, Args... args) {
        T res = 0;
        for (r++; r; r -= (r & -r)) res += bit[r].query(args...);
        return res;
    }
    template<typename... Args>
    T query(int l, int r, Args... args) {
        return sum(r, args...) - sum(l - 1, args...);
    }
}; // BIT<int,10,10> gives a 2D BIT
```

19 Sparse table

```cpp
template<typename T, class CMP = function<T(const T &, const T &)>>
class SparseTable {
public:
    int n;
    vector<vector<T>> sp;
    CMP func;

    void build(const vector<T> &a, const CMP &f) {
        func = f;
        n = static_cast<int>(a.size());
        int max_log = 32 - __builtin_clz(n);
        sp.resize(max_log);
        sp[0] = a;
        for (int j = 1; j < max_log; ++j) {
            sp[j].resize(n - (1 << j) + 1);
            for (int i = 0; i <= n - (1 << j); ++i) {
                sp[j][i] = func(sp[j - 1][i], sp[j - 1][i + (1 << (j - 1))]);
            }
        }
    }

    T query(int l, int r) const {
        int lg = 32 - __builtin_clz(r - l + 1) - 1;
        return func(sp[lg][l], sp[lg][r - (1 << lg) + 1]);
    }
};
```

20 Modular arithmetics stolen from jiangly

```cpp
template<typename T = void> // default
struct BiggerType {
    typedef ll type;
};

template<> // for long long
struct BiggerType<ll> {
    typedef __int128 type;
};
```

```

template<typename T, T mod, typename V = typename BiggerType<T>::type>
struct mint {
private:
    inline T norm(T x) const {
        if (x < 0) x += mod;
        if (x >= mod) x -= mod;
        return x;
    }

public:
    T x;
    mint(T x = 0) : x(norm(x)) {}
    mint(V x) : x(norm(x % mod)) {}
    mint operator-() const { return mint(norm(mod - x)); }
    mint inv() const {
        assert(x != 0);
        return power(mod - 2);
    }
    mint power(long long b) const {
        mint res = 1, a = x;
        for (; b; b >>= 1, a *= a) {
            if (b & 1) res *= a;
        }
        return res;
    }
    mint &operator*=(const mint &rhs) {
        x = (V) x * rhs.x % mod;
        return *this;
    }
    mint &operator+=(const mint &rhs) {
        x = norm(x + rhs.x);
        return *this;
    }
    mint &operator-=(const mint &rhs) {
        x = norm(x - rhs.x);
        return *this;
    }
    mint &operator/=(const mint &rhs) { return *this *= rhs.inv(); }
    friend mint operator*(const mint &lhs, const mint &rhs) {
        mint res = lhs;
        res *= rhs;
        return res;
    }
    friend mint operator+(const mint &lhs, const mint &rhs) {
        mint res = lhs;
        res += rhs;
        return res;
    }
    friend mint operator-(const mint &lhs, const mint &rhs) {
        mint res = lhs;
        res -= rhs;
        return res;
    }
    friend mint operator/(const mint &lhs, const mint &rhs) {
        mint res = lhs;
        res /= rhs;
        return res;
    }
    friend bool operator==(const mint &lhs, const mint &rhs) {
        return lhs.x == rhs.x;
    }
    friend std::istream &operator>>(std::istream &is, mint &a) {
        T v;
        return is >> v, a = mint(v), is;
    }
    friend std::ostream &operator<<(std::ostream &os, const mint &a) {
        return os << a.x;
    }
    friend mint max(mint a, mint b) {
        return a.x > b.x ? a : b;
    }
    friend mint min(mint a, mint b) {
        return a.x < b.x ? a : b;
    }
};

// constexpr int MOD = 998244353;
constexpr int MOD = 1000000007;
using Z = mint<int32_t, MOD>;
```

21 Modular combinations

```
vector<Z> fact = {1};
vector<Z> fact_inv = {1};

void build_fact(int n = 1e6) {
    while ((int) fact.size() < n + 1)
        fact.push_back(fact.back() * (int) fact.size());
    fact_inv.resize(fact.size());
    fact_inv.back() = fact.back().inv();
    for (int j = fact_inv.size() - 2; fact_inv[j].x == 0; j--)
        fact_inv[j] = fact_inv[j + 1] * (j + 1);
}

Z ncr(int n, int r) {
    if (r > n || r < 0) return 0;
    if ((int) fact.size() < n + 1) build_fact(n);
    return fact[n] * fact_inv[r] * fact_inv[n - r];
}
```

22 Disjoint set union

```
struct DSU {
    vector<int> size, parent;
    int forests;

    DSU(int n) {
        forests = n;
        size.assign(n, 1);
        parent.resize(n);
        iota(all(parent), 0);
    }

    bool connected(int x, int y) { return find(x) == find(y); }

    int find(int x) {
        if (parent[x] == x) return x;
        return parent[x] = find(parent[x]);
    }

    bool uni(int x, int y) {
        x = find(x), y = find(y);
        if (x == y) return false;
        forests--;
        parent[y] = x;
        size[x] += size[y];
        return true;
    }
};
```

23 Matrix exponentiation

```
constexpr ll MOD = 1e9 + 7;

template<typename T = int, int mod = MOD>
struct matrix {
    typedef vector<vector<T>> vv;
    vv mat;
    int n, m;

    matrix() { n = 0, m = 0; }
    matrix(vv mat) : mat(mat) { n = mat.size(), m = mat[0].size(); }
    matrix(int n, int m, T ini = 0) : n(n), m(m) { mat = vv(n, vector<T>(m, ini)); }

    matrix operator*(const matrix &other) const {
        matrix mat = *this;
        return mat *= other;
    }

    matrix operator+(const matrix &other) const {
        matrix mat = *this;
        return mat += other;
    }

    matrix operator-(const matrix &other) const {
        matrix mat = *this;
        return mat -= other;
    }

    matrix &operator*=(const matrix &other) {
        assert(m == other.n);
        vector<vector<T>> temp(n, vector<T>(other.m));
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < other.m; j++) {
                for (int k = 0; k < m; k++) {
                    temp[i][j] = (temp[i][j] + 1LL * mat[i][k] * other.mat[k][j]) % mod;
                }
            }
        }
        mat = temp;
        m = other.m;
        return *this;
    }

    matrix &operator+=(const matrix &other) {
        assert(m == other.m && n == other.n);
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < m; j++)
                mat[i][j] = ((mat[i][j] + other.mat[i][j]) % mod + mod) % mod;
        }
        return *this;
    }

    matrix &operator-=(const matrix &other) {
        assert(m == other.m && n == other.n);
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < m; j++)
                mat[i][j] = ((mat[i][j] - other.mat[i][j]) % mod + mod) % mod;
        }
        return *this;
    }

    matrix power(ll p) {
        assert(p >= 0);
        matrix m = *this;
        matrix res = identity(n);
        for (; p; p >>= 1, m *= m)
            if (p & 1) res *= m;
        return res;
    }

    static matrix identity(int size) {
        matrix I = vv(size, vector<T>(size));
        for (int i = 0; i < size; i++)
            I.mat[i][i] = 1;
        return I;
    }
};
```

24 Fast input scanner

```
char in[1 << 24];
struct Scanner {
    char const *o;
    Scanner() : o(in) { load(); }
    void load() { in[fread(in, 1, sizeof(in) - 4, stdin)] = 0; }
    unsigned readInt() {
        unsigned u = 0;
        while (*o && *o <= 32)
            ++o;
        while (*o >= ’0’ && *o <= ’9’)
            u = u * 10 + (*o++ - ’0’);
        return u;
    }
} sc;
```

25 Pascal triagle, useful for combinations

```
vector<vector<Z>> pascal;
void build_pascal(int d) {
    pascal = {{1}};
    while (d--) {
        vector<Z> &lastrow = pascal.back();
        int s = lastrow.size();
        vector<Z> newrow(s + 1);
        newrow.front() = 1;
        newrow.back() = 1;
        for (int i = 1; i < s; i++)
            newrow[i] = lastrow[i] + lastrow[i - 1];
        pascal.push_back(newrow);
    }
}
```

26 Description

```
template<int base = 10>
class bigint {
public:
    vector<int> digits;

    bigint(unsigned ll value = 0) { set_value(value); }

    bigint(string s) {
        digits.resize(s.size());
        for (int i = (int) s.size() - 1; i >= 0; i--) {
            digits[i] = s[(int) s.size() - 1 - i] - ’0’;
        }
    }

    template<typename RandomIt>
    bigint(RandomIt begin, RandomIt end) {
        digits.assign(begin, end);
    }

    void set_value(ll value) {
        digits.clear();
        while (value) {
            digits.push_back(value % base);
            value /= base;
        }
    }

    int size() const { return digits.size(); }

    void trim() {
        while (digits.back() == 0 && digits.size() > 1)
            digits.pop_back();
    }

    int &operator[](int i) { return digits[i]; }

    int operator[](int i) const { return digits[i]; }

    void operator*=(const bigint &rhs) {
        vector<int> res(size() + rhs.size() + 1);
        for (int i = 0; i < size(); i++) {
            for (int j = 0; j < rhs.size(); j++) {
                res[i + j] += digits[i] * rhs[j];
            }
        }
        for (int i = 0; i < (int) res.size() - 1; i++) {
            res[i + 1] += res[i] / base;
            res[i] %= base;
        }
        digits = res;
        trim();
    }

    void operator+=(const bigint &rhs) {
        digits.resize(max(size(), rhs.size()) + 1);
        int i;
        for (i = 0; i < rhs.size(); i++) {
            digits[i] += rhs[i];
            if (digits[i] >= base) {
                digits[i + 1] += digits[i] / base;
                digits[i] %= base;
            }
        }
        while (i < (int) digits.size() - 1 && digits[i] >= base) {
            digits[i + 1] = digits[i] / base;
            digits[i] %= base;
        }
        trim();
    }

    void operator%=(ll mod) {
        ll p = 1;
        ll res = 0;
        for (int i = 0; i < size(); i++) {
            res = (res + p * digits[i] % mod) % mod;
            p = p * base % mod;
        }
        *this = res;
    }

    friend bool operator==(bigint &lhs, bigint &rhs) {
        return lhs.digits == rhs.digits;
    }

    friend bool operator!=(bigint &lhs, bigint &rhs) {
        return lhs.digits != rhs.digits;
    }

    friend bool operator<(bigint &lhs, bigint &rhs) {
        if (lhs.size() != rhs.size())
            return lhs.size() < rhs.size();
        for (int i = lhs.size() - 1; i >= 0; i--) {
            if (lhs[i] < rhs[i]) return true;
            if (lhs[i] > rhs[i]) return false;
        }
        return false;// equal
    }

    friend ostream &operator<<(ostream &os, const bigint &bi) {
        for (int i = bi.size() - 1; i >= 0; i--) os << bi[i];
        return os;
    }
};
```

27 Extended euclidian algorithm

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

28 Modular inverse for coprimes not only prime mod

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

29 Trie data structure

```
struct Trie {
    vector<vector<int>> child;
    vector<int> count;

    Trie() {
        add_node();
    }

    int add_node() {
        count.push_back(0);
        child.push_back(vector<int>(26));
        return count.size() - 1;
    }

    void insert(const string &s) {
        int cur = 0; // root
        for (auto c: s) {
            if (child[cur][c - ’a’] == 0)
                child[cur][c - ’a’] = add_node();
            cur = child[cur][c - ’a’];
            count[cur]++;
        }
    }
};
```

30 String hashing implementation (polynomial hashing)

```
struct HashValue;
constexpr int HASH_ITER = 1;
const ll M = (1LL << 61) - 1;
vector<HashValue> B;

struct HashValue {
    long long val[HASH_ITER]{};
    HashValue(long long v = 0) {
        for (int i = 0; i < HASH_ITER; i++) val[i] = v;
    }
    HashValue &operator-=(const HashValue &a) {
        for (int i = 0; i < HASH_ITER; i++) val[i] = (val[i] - a.val[i] + M) % M;
        return *this;
    }
    HashValue &operator+=(const HashValue &a) {
        for (int i = 0; i < HASH_ITER; i++) val[i] = (val[i] + a.val[i]) % M;
        return *this;
    }
    HashValue &operator*=(const HashValue &a) {
        for (int i = 0; i < HASH_ITER; i++) val[i] = ((__int128) val[i] * a.val[i]) % M;
        return *this;
    }
    HashValue operator+(const HashValue &a) { return HashValue(*this) += a; }
    HashValue operator-(const HashValue &a) { return HashValue(*this) -= a; }
    HashValue operator*(const HashValue &a) { return HashValue(*this) *= a; }
    bool operator<(const HashValue &a) const {
        for (int i = 0; i < HASH_ITER; i++)
            if (val[i] != a.val[i]) return val[i] < a.val[i];
        return false;
    }
    bool operator==(const HashValue &a) const {
        for (int i = 0; i < HASH_ITER; i++)
            if (val[i] != a.val[i]) return false;
        return true;
    }
};

void setB(int n) {
    if (B.size() == 0) {
        HashValue v = 1;
        B.push_back(v);
        mt19937 rng(random_device{}());
        for (int i = 0; i < HASH_ITER; i++)
            v.val[i] = uniform_int_distribution<ll>(0, M - 1)(rng);
        B.push_back(v);
    }
    while (B.size() <= n) B.push_back(B.back() * B[1]);
}

struct Hash {
    vector<HashValue> p_hash;

    Hash(const string &s) : p_hash(s.size() + 1) {
        setB(s.size());
        for (int i = 0; i < s.size(); i++)
            p_hash[i + 1] = p_hash[i] * B[1] + s[i];
    }

    auto get_hash(int start, int end) {
        return p_hash[end + 1] - p_hash[start] * B[end - start + 1];
    }
};
```

31 Eulerian path/circuit in directed graphs

```
template<typename Edge>
class DirectedEulerian {
public:
    int n, m;
    vector<vector<pair<int, Edge>>> adj;

    DirectedEulerian(int n, int m) : n(n), m(m) {
        adj.assign(n, vector<pair<int, Edge>>());
    }

    void add_edge(int u, int v, Edge edge) {
        adj[u].emplace_back(v, edge);
    }

    vector<Edge> path(bool circuit = false) {
        vector<Edge> path;
        int in = 0, out = 0;

        calc_deg();
        int start = -1, end = -1;
        for (int i = 0; i < n; i++) {
            if (indeg[i] > outdeg[i])
                in += indeg[i] - outdeg[i], end = i;
            else if (indeg[i] < outdeg[i])
                out += outdeg[i] - indeg[i], start = i;
        }

        if (m == 0 || !((in == 0 && out == 0) || (in == 1 && out == 1 && !circuit))) {
            return {};
        }

        if (start == -1) {
            assert(end == -1);
            for (int i = 0; i < n; i++) {
                if (outdeg[i] > 0) {
                    start = end = i;
                    break;
                }
            }
        }

        dfs(start, {}, path);

        path.pop_back();
        reverse(all(path));

        return path;
    }

private:
    vector<int> indeg, outdeg;

    void calc_deg() {
        indeg.assign(n, 0);
        outdeg.assign(n, 0);
        for (int i = 0; i < n; i++) {
            outdeg[i] = adj[i].size();
            for (auto &j: adj[i]) indeg[j.first]++;
        }
    }

    void dfs(int i, Edge e, vector<Edge> &path) {
        while (outdeg[i] > 0)
            outdeg[i]--, dfs(adj[i][outdeg[i]].first, adj[i][outdeg[i]].second, path);
        path.push_back(e);
    }
};
```

32 Eulerian path/circuit in undirected graphs

```
template<typename Edge>
class UndirectedEulerian {
public:
    int n, m;
    vector<vector<pair<int, Edge>>> adj; // NOTE: dont’t add a self-edge twice

    UndirectedEulerian(int n, int m) : n(n), m(m) {
        adj.assign(n, vector<pair<int, Edge>>());
    }

    void add_edge(int u, int v, Edge edge) {
        adj[u].emplace_back(v, edge);
        adj[v].emplace_back(u, edge);
    }


    vector<Edge> path(bool circuit = false) {
        vector<Edge> path;

        cnt.clear();
        calc_deg();
        int start = -1, end = -1, odds = 0;
        for (int i = 0; i < n; i++) {
            if (deg[i] & 1) {
                odds++;
                if (˜start)
                    end = i;
                else
                    start = i;
            }
        }

        if (m == 0 || !(odds == 0 || (odds == 2 && !circuit))) {
            return {};
        }

        if (start == -1) {
            assert(end == -1);
            for (int i = 0; i < n; i++) {
                if (deg[i] > 0) {
                    start = end = i;
                    break;
                }
            }
        }

        dfs(start, -1, {}, path);

        path.pop_back();
        reverse(all(path));

        return path;
    }

private:
    vector<int> deg;
    map<pair<int, int>, int> cnt;

    void calc_deg() {
        deg.assign(n, 0);
        for (int i = 0; i < n; i++) {
            for (auto &j: adj[i]) {
                deg[j.first]++;
                if (i == j.first)
                    deg[j.first]++;
                if (i <= j.first)
                    cnt[{i, j.first}]++;
            }
        }
    }

    void dfs(int i, int p, Edge e, vector<Edge> &path) {
        cnt[{min(i, p), max(i, p)}]--;
        while (adj[i].size()) {
            auto [j, E] = adj[i].back();
            adj[i].pop_back();
            if (cnt[{min(i, j), max(i, j)}] == 0) continue;
            dfs(j, i, E, path);
        }
        path.push_back(e);
    }
};
```

33 Mo’s algorithm

```
int block_size;

struct MO {
    struct Query {
        int l, r, idx;
        Query(int l, int r, int idx) : l(l), r(r), idx(idx) {}
        bool operator<(const Query &q) const {
            if (l / block_size != q.l / block_size)
                return pair(l, r) < pair(q.l, q.r);
            return (l / block_size & 1) ? (r < q.r) : (r > q.r);
        }
    };

    vector<int> arr;
    vector<Query> queries;

    MO(vector<int> &arr, vector<Query> &queries) : arr(arr), queries(queries) {}

    int l = 0, r = -1;

    void set_range(Query &q) {
        // [l, r] inclusive
        while (l > q.l) add(arr[--l]);
        while (r < q.r) add(arr[++r]);
        while (l < q.l) remove(arr[l++]);
        while (r > q.r) remove(arr[r--]);
    }

    void add(int x) {
    }

    void remove(int x) {
    }

    int getans(Query &q) {
    }

    vector<int> ans() {
        block_size = arr.size() / sqrt(queries.size()) + 1;
        vector<int> ans(queries.size());
        sort(all(queries));

        l = queries.front().l, r = queries.front().l - 1;
        for (auto &q: queries) {
            set_range(q);
            ans[q.idx] = getans(q);
        }

        return ans;
    }
};
```

34 Torjan’s algorithm

```cpp
struct SCC {
    int N, ID = 0, COMP = 0;
    vector<vector<int>> adj;
    vector<int> id, comp, st;

    SCC(const vector<vector<int>> &adj) : adj(adj), N(adj.size()) {
        id.resize(N), comp = vector<int>(N, -1);
        go();
    }

    void go() {
        for (int i = 0; i < N; i++)
            if (!id[i]) dfs(i);
    }

    int dfs(int i) {
        int low = id[i] = ++ID;
        st.push_back(i);
        for (int j: adj[i])
            if (comp[j] == -1)
                // id[j] != 0 -> in stack, don't dfs
                low = min(low, id[j] ?: dfs(j));
        if (low == id[i]) {
            COMP++;
            for (int j = -1; j != i;)
                comp[j = st.back()] = COMP, st.pop_back();
        }
        return low;
    }
};
```

35 Kmp string algorithm

```cpp
vector<int> KMP(const string &a, const string &b) {
    // search for b in a
    vector<int> ans;
    int n = a.length(), m = b.length();
    int b_table[n];
    b_table[0] = 0;

    for (int i = 1, k = 0; i < m; i++) {
        while (k > 0 && b[k] != b[i])
            k = b_table[k - 1];
        k += b[i] == b[k];
        b_table[i] = k;
    }

    for (int i = 0, k = 0; i < n; i++) {
        while (k > 0 && b[k] != a[i])
            k = b_table[k - 1];
        k += b[k] == a[i];
        if (k == m) {
            k = b_table[k - 1];
            ans.push_back(i - m + 1);
        }
    }

    return ans;
}
```

36 Z algorithm for strings

```cpp
vector<int> zfunction(string s) {
    int n = s.size();

    vector<int> z(n);
    for (int i = 1, l = 1, r = 1; i < n; i++) {
        if (i < r) z[i] = min(z[i - l], r - i);
        while (i + z[i] < n && s[i + z[i]] == s[z[i]]) z[i]++;
        if (i + z[i] > r) r = i + z[i], l = i;
    }

    return z;
}
```

37 Random utils

```cpp
mt19937 rng = mt19937(random_device()());

void seed(int s) { rng = mt19937(s); }

int rand_int(int x, int y) {
    return uniform_int_distribution<int>(x, y)(rng);
}
```

38 Least common ancestor using binary lifting

```cpp
struct LCA {
    int n, LOG;
    vector<int> depth;
    vector<vector<int>> up, adj;

    LCA(int n, int root = 0) : n(n), LOG(log2(n) + 1) {
        adj.resize(n), depth.resize(n);
        up.assign(n, vector<int>(LOG, root));
    }

    void add_edge(int u, int v) {
        adj[u].push_back(v);
        adj[v].push_back(u);
    }

    void dfs(int u, int p) {
        for (auto v: adj[u]) {
            if (v == p) continue;
            up[v][0] = u;
            depth[v] = depth[u] + 1;
            dfs(v, u);
        }
    }

    void build(int root = 0) {
        dfs(root, root);
        for (int k = 1; k < LOG; k++)
            for (int u = 0; u < n; u++)
                up[u][k] = up[up[u][k - 1]][k - 1];
    }

    int query(int u, int v) const {
        if (depth[u] < depth[v]) swap(u, v);
        for (int k = LOG - 1; k >= 0; k--) {
            if (depth[up[u][k]] >= depth[v]) {
                u = up[u][k];
            }
        }
        if (u == v) return u;
        for (int k = LOG - 1; k >= 0; k--) {
            if (up[u][k] != up[v][k]) {
                u = up[u][k];
                v = up[v][k];
            }
        }
        return up[u][0];
    }
};
```

39 Least common ancestor using sparse table

```cpp
struct LCA {
    int n, LOG, _time;
    vector<int> first, depth;
    vector<vector<int>> adj, table;

    LCA(int n) : n(n), LOG(log2(n) + 3) {
        adj.resize(n), depth.resize(n), first.resize(n);
        table.assign(LOG, vector<int>(2 * n));
    }

    void add_edge(int u, int v) {
        adj[u].push_back(v);
        adj[v].push_back(u);
    }

    void dfs(int u, int p) {
        first[u] = _time;
        table[0][_time++] = u;
        for (auto v: adj[u]) {
            if (v == p) continue;
            depth[v] = depth[u] + 1;
            dfs(v, u);
            table[0][_time++] = u;
        }
    }

    void build(int root = 0) {
        _time = 0;
        dfs(root, root);
        assert(_time < 2 * n);
        for (int i = 1; i < LOG; i++) {
            for (int j = 0; j + (1 << i) <= 2 * n; j++) {
                if (depth[table[i - 1][j]] < depth[table[i - 1][j + (1 << (i - 1))]]) {
                    table[i][j] = table[i - 1][j];
                } else {
                    table[i][j] = table[i - 1][j + (1 << (i - 1))];
                }
            }
        }
    }

    int query(int u, int v) const {
        u = first[u], v = first[v];
        if (u == v) return table[0][u];
        if (u > v) swap(u, v);
        int lg = 31 - __builtin_clz(v - u + 1);
        if (depth[table[lg][u]] < depth[table[lg][v - (1 << lg) + 1]]) {
            return table[lg][u];
        } else {
            return table[lg][v - (1 << lg) + 1];
        }
    }
};
```

40 Centroid decomposition of a tree

```
struct Centroids {
    vector<vector<int>> edges;
    vector<bool> removed;
    vector<int> par;
    vector<int> sz;
    int n;

    Centroids(int n) : n(n) {
        edges.resize(n), removed.resize(n);
        sz.resize(n), par.assign(n, -1);
    }

    void add_edge(int a, int b) {
        edges[a].push_back(b);
        edges[b].push_back(a);
    }

    void find_size(int v, int p = -1) {
        sz[v] = 1;
        for (int x: edges[v]) {
            if (x == p || removed[x]) continue;
            find_size(x, v), sz[v] += sz[x];
        }
    }

    int find_centroid(int v, int p, int n) {
        for (int x: edges[v]) {
            if (x == p || removed[x]) continue;
            if (sz[x] > n / 2) return find_centroid(x, v, n);
        }
        return v;
    }

    void build(int v = 0, int p = -1) {
        find_size(v);
        int c = find_centroid(v, -1, sz[v]);
        removed[c] = true, par[c] = p;
        for (int x: edges[c])
            if (!removed[x]) build(x, c);
    }
};
```