Télécom Paris

Télécom 1

Mahdi Nasser, Léopold Bernard, Pacôme Fromager

2025-11-20

1 Mathematics 1

2 Data Structures 4

3 Graphs 9

4 Trees 13

5 Strings 15

6 Geometry 19

7 Miscellaneous 22

Mathematics (1)

CRT.h

578582, 56 lines

```
struct Congruence {
    ll a, m;
};
// m_i pairwise coprime
ll CRT(vector<Congruence> const& congruences) {
    ll M = 1;
    for (auto const& congruence : congruences) {
        M *= congruence.m;
    }

    ll sol = 0;
    for (auto const& congruence : congruences) {
        ll a_i = congruence.a;
        ll M_i = M / congruence.m;
        ll N_i = mod_inv(M_i, congruence.m);
        sol = (sol + a_i * M_i % M * N_i) % M;
    }
    return sol;
}
bool handle_prime(ll m_new, int i, ll a, unordered_map<ll,
     Congruence>& mp) {
    ll a_new = a % m_new;
    if (mp.count(i)) {
        auto [a_old, m_old] = mp[i];
        ll m_min = min(m_old, m_new);
        if (a_old % m_min != a_new % m_min) return 0;
        if (m_new > m_old) mp[i] = {a_new, m_new};

    } else {
        mp[i] = {a_new, m_new};
    }
    return 1;
}
// K = lcm(m_i)
ll CRT_general(vector<Congruence> const& congruences) {
    unordered_map<ll, Congruence> mp;
    for (auto [a, m] : congruences) {
        for (ll i = 2; i * i <= m; i++) {
            ll m_new = 1;
            while (m % i == 0) {
                m_new *= i;
                m /= i;
            }
            if (m_new > 1) {
                if (!handle_prime(m_new, i, a, mp)) return -1;
            }
        }
        if (m != 1) {
            if (!handle_prime(m, m, a, mp)) return -1;
        }
    }
    vector<Congruence> cong_new;
    for (auto [k, c] : mp) {
        cong_new.push_back(c);
    }
    return CRT(cong_new);
}
```

Diophantine.h

c393c4, 74 lines

```
int gcd(int a, int b, int& x, int& y) {
    x = 1, y = 0;
    int x1 = 0, y1 = 1, a1 = a, b1 = b;
    while (b1) {
        int q = a1 / b1;
        tie(x, x1) = make_tuple(x1, x - q * x1);
        tie(y, y1) = make_tuple(y1, y - q * y1);
        tie(a1, b1) = make_tuple(b1, a1 - q * b1);
    }
    return a1;
}
bool find_any_solution(int a, int b, int c, int& x0, int& y0,
     int& g) {
    g = gcd(abs(a), abs(b), x0, y0);
    if (c % g) {
        return false;
    }

    x0 *= c / g;
    y0 *= c / g;
    if (a < 0) x0 = -x0;
    if (b < 0) y0 = -y0;
    return true;
}

void shift_solution(int& x, int& y, int a, int b, int cnt) {
    x += cnt * b;
    y -= cnt * a;
}

// case when a * b > 0
// generate sols through x = lx + k * b / g until x = rx, //
     find corresponding y using ax + by = c
int find_all_solutions(int a, int b, int c, int minx, int maxx,
      int miny, int maxy) {
    int x, y, g;
    if (!find_any_solution(a, b, c, x, y, g))
        return 0;
    a /= g;
    b /= g;

    int sign_a = a > 0 ? +1 : -1;
    int sign_b = b > 0 ? +1 : -1;

    shift_solution(x, y, a, b, (minx - x) / b);
    if (x < minx)
        shift_solution(x, y, a, b, sign_b);
    if (x > maxx)
        return 0;
    int lx1 = x;

    shift_solution(x, y, a, b, (maxx - x) / b);
    if (x > maxx)
        shift_solution(x, y, a, b, -sign_b);
    int rx1 = x;

    shift_solution(x, y, a, b, -(miny - y) / a);
    if (y < miny)
        shift_solution(x, y, a, b, -sign_a);
    if (y > maxy)
        return 0;
    int lx2 = x;

    shift_solution(x, y, a, b, -(maxy - y) / a);
    if (y > maxy)
        shift_solution(x, y, a, b, sign_a);
    int rx2 = x;

    if (lx2 > rx2)
        swap(lx2, rx2);
    int lx = max(lx1, lx2);
    int rx = min(rx1, rx2);

    if (lx > rx)
        return 0;
    return (rx - lx) / abs(b) + 1;
}
```

FFT.h

7311de, 55 lines

```
using cd = complex<double>;
const double PI = acos(-1);

void fft(vector<cd>& a, bool invert) {
    int n = a.size();

    for (int i = 1, j = 0; i < n; i++) {
        int bit = n >> 1;
        for (; j & bit; bit >>= 1)
            j ^= bit;
        j ^= bit;

        if (i < j)
            swap(a[i], a[j]);
    }

    for (int len = 2; len <= n; len <<= 1) {
        double ang = 2 * PI / len * (invert ? -1 : 1);
        cd wlen(cos(ang), sin(ang));
        for (int i = 0; i < n; i += len) {
            cd w(1);
            for (int j = 0; j < len / 2; j++) {
                cd u = a[i + j], v = a[i + j + len / 2] * w;
                a[i + j] = u + v;
                a[i + j + len / 2] = u - v;
                w *= wlen;
            }
        }
    }

    if (invert) {
        for (cd& x : a)
            x /= n;
    }
}

vll multiply(vi const& a, vi const& b) {
    vector<cd> fa(a.begin(), a.end()), fb(b.begin(), b.end());
    int n = 1;
    while (n < a.size() + b.size())
        n <<= 1;
    fa.resize(n);
    fb.resize(n);
```

```
    fft(fa, false);
    fft(fb, false);
    for (int i = 0; i < n; i++)
        fa[i] *= fb[i];
    fft(fa, true);

    vll result(n);
    for (int i = 0; i < n; i++)
        result[i] = round(fa[i].real());
    return result;
}
```

IntegerPartitions.h

dffb49, 24 lines

```
// O(n^2)
for (int v = 1; v < N; v++) {
    for (int s = v; s < N; s++) {
        dp[s] += dp[s - v];
        dp[s] %= M;
    }
}
// O(n*sqrt(n))
dp[0] = dp[1] = 1;
for (int i = 2; i < N; i++) {
    dp[i] = 0;
    for (int j = 1; i > (ll)j * (3 * j - 1) / 2; j++) {
        int x = i - j * (3 * j - 1) / 2;
        int y = i - j * (3 * j + 1) / 2;
        if (j & 1) {
            dp[i] += dp[x];
            if (y > 0) dp[i] += dp[y];
        } else {
            dp[i] -= dp[x];
            if (y > 0) dp[i] -= dp[y];
        }
        dp[i] = (dp[i] + M) % M;
    }
}
```

Matrix.h

770b17, 135 lines

```
template <class T>
struct Matrix {
    vector<vector<T>> A;
    int n;
    bool e;
    Matrix(int n_, bool e = 0) : A(n_, vector<T>(n_)), n(n_), e
        (e) {}
    T& operator()(int i, int j) { return A[i][j]; }
    Matrix operator*(Matrix oth) {
        if (e) return oth;
        if (oth.e) return *this;
        Matrix ans(n);
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                for (int k = 0; k < n; k++) {
                    ans(i, j) += A[i][k] * oth(k, j) % M;
                    ans(i, j) %= M;
                }
            }
        }
        return ans;
    }
};
// Solving System of Linear Equations
const double EPS = 1e-9;
const int INF = 2;
// O(min(n, m) * nm)
int gauss(vector<vector<double>> a, vector<double>& ans) {
    int n = (int)a.size();
    int m = (int)a[0].size() - 1;

    vi where(m, -1);
    for (int col = 0, row = 0; col < m && row < n; ++col) {
        int sel = row;
        for (int i = row; i < n; ++i)
            if (abs(a[i][col]) > abs(a[sel][col]))
                sel = i;
        if (abs(a[sel][col]) < EPS)
            continue;
        for (int i = col; i <= m; ++i)
            swap(a[sel][i], a[row][i]);
        where[col] = row;

        for (int i = 0; i < n; ++i)
            if (i != row) {
                double c = a[i][col] / a[row][col];
                for (int j = col; j <= m; ++j)
                    a[i][j] -= a[row][j] * c;
            }
        ++row;
    }

    ans.assign(m, 0);
    for (int i = 0; i < m; ++i)
        if (where[i] != -1)
            ans[i] = a[where[i]][m] / a[where[i]][i];
    for (int i = 0; i < n; ++i) {
        double sum = 0;
        for (int j = 0; j < m; ++j)
            sum += ans[j] * a[i][j];
        if (abs(sum - a[i][m]) > EPS)
            return 0;
    }

    for (int i = 0; i < m; ++i)
        if (where[i] == -1)
            return INF;
    return 1;
}
// General Matrix Operations
struct Matrix {
    int n, m;
    vector<vector<double>> v;

    Matrix() {}
    Matrix(const vector<vector<double>>& _v) : n(_v.size()), m(
        _v[0].size()), v(_v) {}

    Matrix operator*(const Matrix& b) const {
        assert(m == b.n);
        Matrix res;
        res.n = n, res.m = b.m;
        for (int i = 0; i < res.n; i++) {
            for (int j = 0; j < res.m; j++) {
                for (int k = 0; k < m; k++) {
                    res.v[i][j] += v[i][k] * b.v[k][j];
                    res.v[i][j] %= M;
                }
            }
        }
        return res;
    }
    double determinant() {  // O(n^3)
        double det = 1;
        for (int i = 0; i < n; ++i) {
            int k = i;
            for (int j = i + 1; j < n; ++j)
                if (abs(v[j][i]) > abs(v[k][i]))
                    k = j;
            if (abs(v[k][i]) < EPS) {
                det = 0;
                break;
            }
            swap(v[i], v[k]);
            if (i != k)
                det = -det;
            det *= v[i][i];
            for (int j = i + 1; j < n; ++j)
                v[i][j] /= v[i][i];
            for (int j = 0; j < n; ++j)
                if (j != i && abs(v[j][i]) > EPS)
                    for (int k = i + 1; k < n; ++k)
                        v[j][k] -= v[i][k] * v[j][i];
        }
    }

    int rank() {  // O(n^3)
        int n = v.size();
        int m = v[0].size();

        int rank = 0;
        vector<bool> row_selected(n, false);
        for (int i = 0; i < m; ++i) {
            int j;
            for (j = 0; j < n; ++j) {
                if (!row_selected[j] && abs(v[j][i]) > EPS)
                    break;
            }

            if (j != n) {
                ++rank;
                row_selected[j] = true;
                for (int p = i + 1; p < m; ++p)
                    v[j][p] /= v[j][i];
                for (int k = 0; k < n; ++k) {
                    if (k != j && abs(v[k][i]) > EPS) {
                        for (int p = i + 1; p < m; ++p)
```

MillerRabin.h

a3c812, 45 lines

```
ll mult(ll a, ll b, ll mod) {
    return (__int128)a * b % mod;
}
ll mod_pow(ll b, ll p, ll mod) {
    ll ans = 1;
    while (p) {
        if (p & 1) ans = mult(ans, b, mod);
        b = mult(b, b, mod);
        p >>= 1;
    }
    return ans;
}
bool is_prime(ll n, int trials = 5) {
    if (n == 2) return 1;
    if (n < 2 || !(n & 1)) return 0;

    ll q = n - 1;
    int k = 0, b = 0;
    while (!(q & 1)) {
        q >>= 1;
        k++;
    }

    for (int a : {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37})
        {
        // ll a = rand() % (n - 4) + 2;
```

```
        if (n == a) {
            return 1;  //
        }

        ll t = mod_pow(a, q, n);
        bool ok = (t == 1) || (t == n - 1);
        if (ok) continue;

        for (int i = 1; i < k; i++) {
            t = mult(t, t, n);
            if (t == n - 1) {
                ok = 1;
                break;
            }
        }
        if (!ok)
            return 0;
    }
    return 1;
}
```

NTT.h

07128a, 146 lines

```
// Arbitrary Mod
typedef complex<double> cd;
void fft(vector<cd>& a) {
    int n = size(a), L = 31 - __builtin_clz(n);
    static vector<complex<long double>> R(2, 1);
    static vector<cd> rt(2, 1);  // (^ 10% faster if double)
    for (static int k = 2; k < n; k *= 2) {
        R.resize(n);
        rt.resize(n);
        auto x = polar(1.0L, acos(-1.0L) / k);
        for (int i = k; i < 2 * k; i++) rt[i] = R[i] = i & 1 ?
             R[i / 2] * x : R[i / 2];
    }
    vi rev(n);
    for (int i = 0; i < n; i++) rev[i] = (rev[i / 2] | (i & 1)
         << L) / 2;
    for (int i = 0; i < n; i++)
        if (i < rev[i]) swap(a[i], a[rev[i]]);

    for (int k = 1; k < n; k *= 2)
        for (int i = 0; i < n; i += 2 * k)
            for (int j = 0; j < k; j++) {
                auto x = (double*)&rt[j + k], y = (double*)&a[i
                     + j + k];
                cd z(x[0] * y[0] - x[1] * y[1], x[0] * y[1] + x
                     [1] * y[0]);
                a[i + j + k] = a[i + j] - z;
                a[i + j] += z;
            }
}

typedef vector<ll> vll;
template <int mod>
vll convMod(const vll& a, const vll& b) {
    if (a.empty() || b.empty()) return {};
    vll res(size(a) + size(b) - 1);
    int B = 32 - __builtin_clz(size(res)), n = 1 << B, cut =
         int(sqrt(mod));
    vector<cd> L(n), R(n), outs(n), outl(n);
    for (int i = 0; i < size(a); i++) L[i] = cd((int)a[i] / cut
         , (int)a[i] % cut);
    for (int i = 0; i < size(b); i++) R[i] = cd((int)b[i] / cut
         , (int)b[i] % cut);
    fft(L), fft(R);
    for (int i = 0; i < n; i++) {
        int j = -i & (n - 1);
        outl[j] = (L[i] + conj(L[j])) * R[i] / (2.0 * n);
        outs[j] = (L[i] - conj(L[j])) * R[i] / (2.0 * n) / 1i;
    }
    fft(outl), fft(outs);
    for (int i = 0; i < size(res); i++) {
        ll av = ll(real(outl[i]) + .5), cv = ll(imag(outs[i]) +
             .5);
        ll bv = ll(imag(outl[i]) + .5) + ll(real(outs[i]) + .5)
             ;
        res[i] = ((av % mod * cut + bv) % mod * cut + cv) % mod
             ;
    }
    return res;
}
// Mod of the form p=c2^k+1
/*
Already Precomputed Constants
MOD = 998244353     root = 3     inv_root = 332748118     mod_pow
     = 23     odd_factor = 119
MOD = 132120577     root = 5     inv_root = 52848231      mod_pow
     = 21     odd_factor = 63
MOD = 7340033       root = 3     inv_root = 2446678       mod_pow
     = 20     odd_factor = 7
MOD = 786433        root = 10    inv_root = 235930        mod_pow
     = 18     odd_factor = 3;
*/

const ll mod = 132120577;
const ll root = 5;
const ll inv_root = 52848231;
const int mod_p = 21;
const ll odd_factor = 63;

void NTT(vector<ll>& a, const ll& cur_root) {
    int n = (int)a.size();
    for (int i = 1, j = 0; i < n; i++) {
        int bit = n >> 1;
        for (; j & bit; bit >>= 1) j ^= bit;
        if (i < (j ^= bit)) {
            swap(a[i], a[j]);
        }
    }

    for (int len = 2; len <= n; len <<= 1) {
        ll Wn = cur_root;
        for (int i = len; i < n; i <<= 1)
            Wn = (Wn * Wn) % mod;

        for (int i = 0; i < n; i += len) {
            ll W = 1;
            for (int j = 0; j < (len >> 1); j++, W = (W * Wn) %
                 mod) {
                ll even = a[i + j], odd = (W * a[i + j + (len
                     >> 1)]) % mod;
                a[i + j + (len >> 1)] = (even - odd + mod) %
                     mod;
                a[i + j] = (even + odd) % mod;
            }
        }
    }
}

vector<ll> poly_mod_mult(vector<ll>& a, vector<ll>& b) {
    vector<ll> A(a.begin(), a.end()), B(b.begin(), b.end());
    int N = (int)a.size(), M = (int)b.size();
    int n = 1, logN = 0;
    while (n < (N + M)) {
        n <<= 1;
        logN++;
    }
    A.resize(n, 0);
    B.resize(n, 0);

    // Getting roots with cycle exactly n
    ll norm_factor = odd_factor * mod_pow(2, mod_p - logN);
    ll cur_root = mod_pow(root, norm_factor);
    ll cur_inv_root = mod_pow(inv_root, norm_factor);

    NTT(A, cur_root);
    NTT(B, cur_root);
    for (int i = 0; i < n; i++) A[i] = (A[i] * B[i]) % mod;

    NTT(A, cur_inv_root);
    ll invN = mod_inv(n);
    for (auto& x : A) x = (x * invN) % mod;

    return A;
}

void generate_constants(const ll& MOD) {
    auto is_primitve_root = [&](ll x) {
        ll cur = x, N = 1;
        while (++N != MOD - 1) {
            cur = (cur * x) % MOD;
            if (cur == 1) {
                return false;
            }
        }
        return true;
    };
    int r = 2;
    while (!is_primitve_root(r)) {
        r++;
    }
    int p = 0;
    ll cur = MOD - 1, rInv = mod_inv(r);
    while ((cur & 1) == 0) {
        p++;
        cur >>= 1;
    }

    cout << "Prime: " << MOD << '\n';
    cout << "Primitive Root: " << r << '\n';
    cout << "Inverse of Primitive Root: " << rInv << '\n';
    cout << "Highest Power of 2 in (Prime-1): " << p << ' ' <<
         (1ll << p) << '\n';
    cout << "Highest Odd factor of (Prime-1): " << (MOD - 1) /
         (1ll << p) << '\n';
}
```

PollardRho.h

0289df, 26 lines

```
ll rho(ll n, ll c) {
    ll x = 2, y = 2, i = 1, k = 2, d;
    while (true) {
        x = mult(x, x, n) + c;
        if (x >= n) x -= n;
        d = gcd(x - y, n);
        if (d > 1) return d;
        if (++i == k) y = x, k <<= 1;
    }
    return n;
}
void pollard_rho(ll n, vll& f) {  // O(n^1/4)
    if (n == 1)
        return;

    if (is_prime(n)) {
```

```
        f.push_back(n);
        return;
    }
    ll d = n;
    for (int i = 2; d == n; i++) {
        d = rho(n, i);
    }
    pollard_rho(d, f);
    pollard_rho(n / d, f);
}
```

Sieve.h

7c93e2, 53 lines

```
void sieve() {
    // primes <= N
    vector<bool> is_prime(N + 1, true);
    is_prime[0] = is_prime[1] = false;
    for (int i = 2; i * i <= N; i++) {
        if (is_prime[i]) {
            for (int j = i * i; j <= N; j += i)
                is_prime[j] = false;
        }
    }
}
// useful for O(logn) factorization where n <= 1e7
int spf[N];
void linear_sieve() {
    vi pr;

    for (int i = 2; i < N; ++i) {
        if (spf[i] == 0) {
            spf[i] = i;
            pr.push_back(i);
        }
        for (int j = 0; i * pr[j] < N; ++j) {
            spf[i * pr[j]] = pr[j];
            if (pr[j] == spf[i]) {
                break;
            }
        }
    }
}

vii factorize(int x) {
    vii fac;
    while (x > 1) {
        int p = spf[x];
        int j = 0;
        while (x % p == 0) {
            x /= p;
            j++;
        }
        fac.push_back({p, j});
    }
    return fac;
}
// without 1
void gen_divs(const vii& fac, int i, int c, int x, vi& divs) {
    if (i == fac.size()) return;
    auto [p, j] = fac[i];
    if (c < j) {
        divs.push_back(x * p);
        gen_divs(fac, i, c + 1, x * p, divs);
    }
    gen_divs(fac, i + 1, 0, x, divs);
}
```

Totient.h

64ce84, 26 lines

```
int phi(int n) {
    int result = n;
    for (int i = 2; i * i <= n; i++) {
        if (n % i == 0) {
            while (n % i == 0)
                n /= i;
            result -= result / i;
        }
    }
    if (n > 1)
        result -= result / n;
    return result;
}

void phi_1_to_n(int n) {
    vector<int> phi(n + 1);
    for (int i = 0; i <= n; i++)
        phi[i] = i;

    for (int i = 2; i <= n; i++) {
        if (phi[i] == i) {
            for (int j = i; j <= n; j += i)
                phi[j] -= phi[j] / i;
        }
    }
}
```

Data Structures (2)

DSUrb.h

cabcff, 38 lines

```
struct DSU {
    struct save {
        int x, y, hx, hy, cc;
    };

    vector<int> h, p;
    int cc;
    stack<save> history;

    DSU(int n) : h(n), p(n), cc(n) {
        iota(all(p), 0);
    }

    int find(int x) { return x == p[x] ? x : find(p[x]); }
    bool join(int x, int y) {
        int px = find(x), py = find(y);
        if (px == py) {
            return false;
        } else {
            if (h[px] < h[py]) swap(px, py);

            history.push({px, py, h[px], h[py], cc});

            if (h[px] == h[py]) h[px]++;
            cc--;
            p[py] = px;

            return true;
        }
    }

    void rollback() {
        auto [x, y, hx, hy, c] = history.top();
        history.pop();
        p[x] = x, p[y] = y, h[x] = hx, h[y] = hy;
        cc = c;
    }
};
```

ExplicitTreap.h

18bf9e, 109 lines

```
#define f first
#define s second

using pt = struct tnode *;
struct tnode {
    int pri, val;
    pt c[2];  // essential

    int sz;
    ll sum;          // for range queries
    bool flip = 0;  // lazy update
    tnode(int _val) {
        pri = rand();
        sum = val = _val;
        sz = 1;
        c[0] = c[1] = nullptr;
    }
    ~tnode() {
        delete c[0];
        delete c[1];
    }
};

int size(pt x) { return x ? x->sz : 0; }
ll sum(pt x) { return x ? x->sum : 0; }

pt push(pt x) {  // lazy propagation
    if (!x || !x->flip) return x;

    swap(x->c[0], x->c[1]);
    x->flip = 0;

    if (x->c[0]) x->c[0]->flip ^= 1;
    if (x->c[1]) x->c[1]->flip ^= 1;

    return x;
}

pt pull(pt x) {
    pt a = x->c[0], b = x->c[1];
    // assert(!x->flip); push(a), push(b);
    x->sz = 1 + size(a) + size(b);
    x->sum = (x->val + sum(a) + sum(b)) % M;
    return x;
}

void tour(pt x, vi &v) {  // print values of nodes,
    if (!x) return;       // inorder traversal
    // push(x);
    tour(x->c[0], v);
    v.push_back(x->val);
    tour(x->c[1], v);
}

pair<pt, pt> split(pt t, int v) {  // >= v goes to the right
    if (!t) return {t, t};
    // push(t);
    if (t->val >= v) {
        auto p = split(t->c[0], v);
        t->c[0] = p.s;
        return {p.f, pull(t)};
    } else {
        auto p = split(t->c[1], v);
        t->c[1] = p.f;
```

```
        return {pull(t), p.s};
    }
}
pair<pt, pt> splitsz(pt t, int sz) {  // sz nodes go to left
    if (!t) return {t, t};
    // push(t);
    if (size(t->c[0]) >= sz) {
        auto p = splitsz(t->c[0], sz);
        t->c[0] = p.s;
        return {p.f, pull(t)};
    } else {
        auto p = splitsz(t->c[1], sz - size(t->c[0]) - 1);
        t->c[1] = p.f;
        return {pull(t), p.s};
    }
}
pt merge(pt l, pt r) {  // keys in l < keys in r
    if (!l || !r) return l ?: r;
    // push(l), push(r);
    pt t;
    if (l->pri > r->pri)
        l->c[1] = merge(l->c[1], r), t = l;
    else
        r->c[0] = merge(l, r->c[0]), t = r;
    return pull(t);
}
pt insert(pt x, int v) {
    auto a = split(x, v), b = split(a.s, v + 1);
    return merge(a.f, merge(new tnode(v), b.s));
}
pt erase(pt x, int v) {
    auto a = split(x, v), b = split(a.s, v + 1);
    return merge(a.f, b.s);
}
ll query(pt t, int l, int r, int i, int j) {
    if (!t || r < i || l > j) {
        return 0;
    }
    if (i <= l && r <= j) {
        return t->sum;
    }
    int m = size(t->c[0]) + l;
    ll add = (i <= m && m <= j ? t->val : 0);
    return (add + query(t->c[0], l, m - 1, i, j) + query(t->c
        [1], m + 1, r, i, j)) % M;
}
```

ImplicitTreap.h

400e4e, 94 lines

```
struct Node {
    int val;
    ll sum;
    bool rev;
    int weight, size;
    Node *left, *right;
    Node(int x) : val(x), weight(rand()), size(1), left(NULL),
        right(NULL), rev(0), sum(x) {}
};
inline int size(Node* node) { return node ? node->size : 0; }
inline ll sum(Node* node) { return node ? node->sum : 0; }

void pull(Node* node) {
    node->sum = node->val + sum(node->left) + sum(node->right);
    node->size = 1 + size(node->left) + size(node->right);
}
void push(Node*& node) {
    if (!node) return;
    if (node->rev) {
        node->rev = 0;
        swap(node->left, node->right);
        if (node->left) node->left->rev ^= 1;
        if (node->right) node->right->rev ^= 1;
    }
}
void split(Node* node, Node*& left, Node*& right, int p) {
    push(node);
    if (!node) {
        left = right = NULL;
        return;
    }
    if (size(node->left) <= p) {
        split(node->right, node->right, right, p - size(node->
            left) - 1);
        left = node;
    } else {
        split(node->left, left, node->left, p);
        right = node;
    }
    pull(node);
}
void merge(Node*& node, Node* left, Node* right) {
    push(left), push(right);
    if (left == NULL) {
        node = right;
        return;
    }
    if (right == NULL) {
        node = left;
        return;
    }
    if (left->weight < right->weight) {
        merge(left->right, left->right, right);
        node = left;
    } else {
        merge(right->left, left, right->left);
        node = right;
    }
    pull(node);
}
void reverse(Node* node, int l, int r) {
    Node *t1, *t2, *t3;
    split(node, t1, t2, r);
    split(t1, t1, t3, l - 1);
    t3->rev ^= 1;
    merge(node, t1, t3);
    merge(node, node, t2);
}
ll query(Node* node, int l, int r, int i, int j) {
    push(node);
    if (!node || r < i || l > j) {
        return 0;
    }
    if (i <= l && r <= j) {
        return node->sum;
    }
    int m = size(node->left) + l;
    int add = (i <= m && m <= j ? node->val : 0);
    return add + query(node->left, l, m - 1, i, j) + query(node
        ->right, m + 1, r, i, j);
}

ostream& operator<<(ostream& os, Node* node) {
    if (!node) return os;
    push(node);
    os << node->left;
    os << node->val;
    os << node->right;
    return os;
}
```

```
}
// main
Node* root = NULL;
for (int i = 0; i < n; i++) {
    int x;
    cin >> x;
    merge(root, root, new Node(x));
}
```

LazySparseSegTree.h

d8f92b, 40 lines

```
struct node {
    int sum = 0, lazy = 0, l, r;
    node *left, *right;
    node(int l, int r) : l(l), r(r), left(nullptr), right(
        nullptr) {}

    void extend() {
        if (!left && l != r) {
            int m = l + r >> 1;
            left = new node(l, m);
            right = new node(m + 1, r);
        }
    }

    void push() {  // lazy only for children; used when
        children are created when necessary
        if (!lazy || l == r || !left) return;
        left->lazy = lazy, right->lazy = lazy;
        int m = l + r >> 1;
        left->sum = (m - l + 1) * lazy, right->sum = (r - m) *
            lazy;
        lazy = 0;
    }

    void update(int i, int j, int x) {  // set
        if (i <= l && r <= j) {
            lazy = x;
            sum = (r - l + 1) * x;
            return;
        }
        if (l > j || r < i) return;
        extend(), push();
        left->update(i, j, x);
        right->update(i, j, x);
        sum = left->sum + right->sum;
    }
    int query(int i, int j) {  // sum
        if (i <= l && r <= j) return sum;
        if (j < l || r < i) return 0;
        extend(), push();
        return left->query(i, j) + right->query(i, j);
    }
};
```

LineContainer.h

Description: Container where you can add lines of the form kx+m, and query maximum values at points x.

8ec1c7, 32 lines

```
struct Line {
    mutable ll k, m, p;
    bool operator<(const Line &o) const { return k < o.k; }
    bool operator<(ll x) const { return p < x; }
};
struct LineContainer : multiset<Line, less<>> {
    // (for doubles, use inf = 1/.0, div(a,b) = a/b)
    static const ll inf = LLONG_MAX;
    ll div(ll a, ll b) {  // floored division
        return a / b - ((a ^ b) < 0 && a % b);
```

```
    }
    bool isect(iterator x, iterator y) {
        if (y == end()) return x->p = inf, 0;
        if (x->k == y->k)
            x->p = x->m > y->m ? inf : -inf;
        else
            x->p = div(y->m - x->m, x->k - y->k);
        return x->p >= y->p;
    }
    void add(ll k, ll m) {
        auto z = insert({k, m, 0}), y = z++, x = y;
        while (isect(y, z)) z = erase(z);
        if (x != begin() && isect(--x, y)) isect(x, y = erase(y
            ));
        while ((y = x) != begin() && (--x)->p >= y->p)
            isect(x, erase(y));
    }
    ll query(ll x) {
        assert(!empty());
        auto l = *lower_bound(x);
        return l.k * x + l.m;
    }
};
```

Mo.h

c07cb8, 29 lines

```
const int S = 200;
struct Mo {
    const vi &A;
    int L, R;
    int ans;

    Mo(vi &A) : A(A), L(0), R(-1), ans(0) {}

    int process(int l, int r) {
        while (R < r) insert(A[++R]);
        while (R > r) erase(A[R--]);
        while (L < l) erase(A[L++]);
        while (L > l) insert(A[--L]);
        return ans;
    }
    void insert(int x) {
    }
    void erase(int x) {
    }
};

struct Query {
    int l, r, i;
    bool operator<(Query q) {
        int b1 = l / S, b2 = q.l / S;
        if (b1 == b2) return r < q.r;
        return b1 < b2;
    }
};
```

PersSegTree.h

8f0a79, 67 lines

```
const int e = 0;
struct node {
    ll sum;
    node *left, *right;
    node(int x) : left(nullptr), right(nullptr), sum(x) {}
    node(node *l, node *r) : left(l), right(r), sum(e) {
        if (left) sum += l->sum;
        if (right) sum += r->sum;
    }
};
struct PersSegTree {
    int n;
    vector<node *> version;

    PersSegTree(int n_) : n(n_) {
        vi A(n, e);
        version.push_back(build(A, 0, n - 1));
    }

    PersSegTree(vi &A) {
        n = A.size();
        version.push_back(build(A, 0, n - 1));
    }

    node *build(vi &A, int l, int r) {
        if (l == r) {
            return new node(A[l]);
        }
        int m = l + r >> 1;
        return new node(build(A, l, m), build(A, m + 1, r));
    }

    ll query(node *v, int l, int r, int i, int j) {
        if (i <= l && r <= j) {
            return v->sum;
        }
        if (r < i || l > j) {
            return e;
        }
        int m = l + r >> 1;
        return query(v->left, l, m, i, j) + query(v->right, m +
            1, r, i, j);
    }
    node *update(node *v, int l, int r, int p, int x) {
        if (l == r) {
            return new node(x);
        }

        int m = l + r >> 1;
        if (p <= m)
            return new node(update(v->left, l, m, p, x), v->
                right);
        else
            return new node(v->left, update(v->right, m + 1, r,
                p, x));
    }

    ll query(int i, int j, int k = -1) {
        if (k == -1) k = version.size() - 1;
        return query(version[k], 0, n - 1, i, j);
    }
    void update(int p, int x, int k = -1) {
        if (k == -1) k = version.size() - 1;
        version[k] = update(version[k], 0, n - 1, p, x);
    }
    void copy(int k = -1) {
        if (k == -1) k = version.size() - 1;
        version.push_back(version[k]);
    }
};
```

QueryTree.h

d083b5, 41 lines

```
struct query {
    int u, v;
    bool joined = 0;
    query(int u_, int v_) : u(u_), v(v_) {}
};

struct QueryTree {
    DSU dsu;
    vector<vector<query>> tree;
    vi ans;

    QueryTree(int n, int q) : dsu(n), tree(4 * q + 5), ans(q) {
        }

    void insert(int p, int l, int r, int i, int j, query qr) {
        if (i <= l && r <= j) {
            tree[p].push_back(qr);
            return;
        }
        if (l > j || r < i) return;
        int m = l + r >> 1;
        insert(p << 1, l, m, i, j, qr);
        insert(p << 1 | 1, m + 1, r, i, j, qr);
    }

    void dfs(int p, int l, int r) {
        for (query& qr : tree[p]) {
            qr.joined = dsu.join(qr.u, qr.v);
        }

        if (l == r) {
            ans[l] = dsu.cc;
        } else {
            int m = l + r >> 1;
            dfs(p << 1, l, m);
            dfs(p << 1 | 1, m + 1, r);
        }
        for (query qr : tree[p]) {
            if (qr.joined) dsu.rollback();
        }
    }
}
```

RUPQ.h

326d69, 27 lines

```
template <class T>
struct BIT {
    int n;
    vector<T> bit;

    BIT(int n) : n(n), bit(n + 1) {}

    // dont use even for pt update
    void add(int i, T val) {
        i++;
        for (; i <= n; i += i & -i) {
            bit[i] += val;
        }
    }
    void range_add(int l, int r, T val) {
        add(l, val);
        if (r + 1 < n)
            add(r + 1, -val);
    }
    T pt_query(int i) {
        i++;
        T ret = 0;
        for (; i > 0; i -= i & -i)
            ret += bit[i];
        return ret;
    }
};
```

SegTree2D.h

558d38, 113 lines

```
template <class T>
struct SegTree2D {
    vector<vector<T>> tree;
    vector<vi>& A;
    int n, m, ni, mi;

    const T e = 0;
    T merge(T a, T b) { return a + b; }

    SegTree2D(vector<vi>& A_) : A(A_) {
        n = A.size(), m = A[0].size();
        ni = n, mi = m;
        while (__builtin_popcount(n) != 1) n++;
        while (__builtin_popcount(m) != 1) m++;

        tree.assign(2 * n, vector<T>(2 * m, e));

        build_x(1, 0, n - 1);
    }
    void build_y(int px, int lx, int rx, int py, int ly, int ry
        ) {
        if (ly == ry) {
            if (lx == rx) {
                if (lx >= ni || ly >= mi)
                    tree[px][py] = e;
                else
                    tree[px][py] = A[lx][ly];
            } else {
                tree[px][py] = merge(tree[px << 1][py], tree[px
                    << 1 | 1][py]);
            }
        } else {
            int my = ly + ry >> 1;
            build_y(px, lx, rx, py << 1, ly, my);
            build_y(px, lx, rx, py << 1 | 1, my + 1, ry);
            tree[px][py] = merge(tree[px][py << 1], tree[px][py
                << 1 | 1]);
        }
    }
    void build_x(int px, int lx, int rx) {
        if (lx != rx) {
            int mx = lx + rx >> 1;
            build_x(px << 1, lx, mx);
            build_x(px << 1 | 1, mx + 1, rx);
        }
        build_y(px, lx, rx, 1, 0, m - 1);
    }

    T query_y(int px, int ly, int ry) {
        T ra = e, rb = e;
        for (ly += m, ry += m + 1; ly < ry; ly >>= 1, ry >>= 1)
            {
            if (ly & 1) ra = merge(ra, tree[px][ly++]);
            if (ry & 1) rb = merge(rb, tree[px][--ry]);
        }
        return merge(ra, rb);
    }
    T query(int lx, int rx, int ly, int ry) {
        T ra = e, rb = e;
        for (lx += n, rx += n + 1; lx < rx; lx >>= 1, rx >>= 1)
            {
            if (lx & 1) ra = merge(ra, query_y(lx++, ly, ry));
            if (rx & 1) rb = merge(rb, query_y(--rx, ly, ry));
        }
        return merge(ra, rb);
    }
    /*T query_y(int px, int py, int ly, int ry, int i2, int j2)
        {
        if (i2 <= ly && ry <= j2) {
            return tree[px][py];
        }
        if (ry < i2 || ly > j2) {
            return e;
        }
        int my = ly + ry >> 1;
        return merge(query_y(px, py << 1, ly, my, i2, j2),
            query_y(px, py << 1 | 1, my + 1, ry, i2, j2));
    }
    T query_x(int px, int lx, int rx, int i1, int j1, int i2,
        int j2) {
        if (i1 <= lx && rx <= j1) {
            return query_y(px, 1, 0, m - 1, i2, j2);
        }
        if (rx < i1 || lx > j1) {
            return e;
        }
        int mx = lx + rx >> 1;
        return merge(query_x(px << 1, lx, mx, i1, j1, i2, j2),
            query_x(px << 1 | 1, mx + 1, rx, i1, j1, i2, j2));
    }*/

    void update_y(int px, int lx, int rx, int py, int ly, int
        ry, int x, int y, int v) {
        if (ly == ry) {
            if (lx == rx) {
                tree[px][py] = v;
            } else {
                tree[px][py] = tree[px << 1][py] + tree[px << 1
                    | 1][py];
            }
        } else {
            int my = ly + ry >> 1;
            if (y <= my)
                update_y(px, lx, rx, py << 1, ly, my, x, y, v);
            else
                update_y(px, lx, rx, py << 1 | 1, my + 1, ry, x
                    , y, v);
            tree[px][py] = merge(tree[px][py << 1], tree[px][py
                << 1 | 1]);
        }
    }
    void update_x(int px, int lx, int rx, int x, int y, T v) {
        if (lx != rx) {
            int mx = lx + rx >> 1;
            if (x <= mx)
                update_x(px << 1, lx, mx, x, y, v);
            else
                update_x(px << 1 | 1, mx + 1, rx, x, y, v);
        }
        update_y(px, lx, rx, 1, 0, n - 1, x, y, v);
    }

    // T query(int x1, int x2, int y1, int y2) { return query_x
        (1, 0, n - 1, x1, x2, y1, y2); }
    void update(int x, int y, T v) { update_x(1, 0, n - 1, x, y
        , v); }
};
```

SparseSegTree.h

50398d, 31 lines

```
struct node {
    int val = 2e9, l, r;
    node *left, *right;
    node(int l, int r) : l(l), r(r), left(nullptr), right(
        nullptr) {}
    void update(int p, int x) {  // set
        if (l == p && r == p) {
            val = x;
            return;
        }
        int m = l + r >> 1;
        if (p <= m) {
            if (!left) left = new node(l, m);
            left->update(p, x);
        } else {
            if (!right) right = new node(m + 1, r);
            right->update(p, x);
        }
        val = 2e9;  // important
        if (left) ckmin(val, left->val);
        if (right) ckmin(val, right->val);
    }
    int query(int i, int j) {  // min
        if (i <= l && r <= j) return val;
        if (j < l || r < i) return 2e9;
        int res = 2e9;
        if (left) ckmin(res, left->query(i, j));
        if (right) ckmin(res, right->query(i, j));
        return res;
    }
};
```

SparseTable.h

8d6156, 22 lines

```
template <class T>
struct SparseTable {
    vector<vector<T>> st;
    int log, n;

    T op(T& a, T& b) { return min(a, b); }
    SparseTable(vector<T>& A) {
        n = A.size();
        log = 32 - __builtin_clz(n);
        st.assign(n, vector<T>(log));
        for (int i = 0; i < n; i++) st[i][0] = A[i];
        for (int k = 1; k < log; k++) {
            for (int i = 0; i + (1 << k) - 1 < n; i++) {
                st[i][k] = op(st[i][k - 1], st[i + (1 << (k -
                    1))][k - 1]);
            }
        }
    }
    T query(int l, int r) {
        int k = 31 - __builtin_clz(r - l + 1);
        return op(st[l][k], st[r - (1 << k) + 1][k]);
    }
};
```

SplayTree.h

df37c8, 182 lines

```
const int MX = 1e6 + 5, M = 998244353;
struct node {
    int p, ch[2], sz;
    long long val, sum, lazyb, lazyc;
    bool flip;
    node(long long x = 0) : ch(), val(x), sum(x), flip(0), p(0)
        , sz(0), lazyb(1), lazyc(0) {}
} t[MX];
int id = 1;

void push(int u) {
    if (!u) return;
    if (t[u].lazyb != 1 || t[u].lazyc) {
```

```
        t[u].val = (t[u].lazyb * t[u].val + t[u].lazyc) % M;
        t[u].sum = (t[u].lazyb * t[u].sum + t[u].lazyc * t[u].
             sz) % M;

        for (int i = 0; i < 2; i++) {
            int v = t[u].ch[i];
            if (v) {
                t[v].lazyb = t[v].lazyb * t[u].lazyb % M;
                t[v].lazyc = (t[v].lazyc * t[u].lazyb + t[u].
                     lazyc) % M;
            }
        }

        t[u].lazyb = 1;
        t[u].lazyc = 0;
    }
    if (t[u].flip) {
        swap(t[u].ch[0], t[u].ch[1]);
        t[t[u].ch[0]].flip ^= 1;
        t[t[u].ch[1]].flip ^= 1;
        t[u].flip = 0;
    }
}

void pull(int u) {
    int l = t[u].ch[0], r = t[u].ch[1];
    push(l), push(r);
    t[u].sz = t[l].sz + t[r].sz + 1;
    t[u].sum = (t[l].sum + t[r].sum + t[u].val) % M;
}

void attach(int u, int v, int d) {
    t[u].ch[d] = v;
    t[v].p = u;
    pull(u);
}

int build(vector<int>& A, int l, int r) {
    if (l > r) return 0;
    int m = (l + r) / 2, u = id++;
    t[u] = node(A[m]);
    attach(u, build(A, l, m - 1), 0);
    attach(u, build(A, m + 1, r), 1);
    return u;
}

int dir(int u) {
    int p = t[u].p;
    return t[p].ch[0] == u ? 0 : (t[p].ch[1] == u ? 1 : -1);
}

void rotate(int u) {
    int v = t[u].p, w = t[v].p;
    int du = dir(u), dv = dir(v);

    attach(v, t[u].ch[!du], du);
    attach(u, v, !du);

    if (~dv) {
        attach(w, u, dv);
    } else {
        t[u].p = w;
    }
}

void splay(int u) {
    while (true) {
        int v = t[u].p, w = t[v].p;
        int du = dir(u), dv = dir(v);
        push(w), push(v), push(u);
        if (du == -1) return;
        if (~dv) rotate(du == dv ? v : u);
        rotate(u);
    }
}

int find(int u, int i) {
    assert(u);
    push(u);
    int rk = t[t[u].ch[0]].sz;

    if (i < rk) {
        return find(t[u].ch[0], i);
    } else if (i > rk) {
        return find(t[u].ch[1], i - rk - 1);
    } else {
        splay(u);
        return u;
    }
}

int find_max(int u) {
    push(u);
    while (t[u].ch[1]) {
        u = t[u].ch[1];
        push(u);
    }
    splay(u);
    return u;
}

int find_min(int u) {
    push(u);
    while (t[u].ch[0]) {
        u = t[u].ch[0];
        push(u);
    }
    splay(u);
    return u;
}

int merge(int l, int r) {
    push(l), push(r);
    if (!l || !r) return l ? l : r;
    l = find_max(l);
    attach(l, r, 1);
    return l;
}

pair<int, int> split(int u, int i) {
    assert(i < t[u].sz);
    if (i == -1) {
        return {0, u};
    }
    u = find(u, i);
    int v = t[u].ch[1];
    t[u].ch[1] = t[v].p = 0;
    pull(u);
    return {u, v};
}

int insert(int u, int i, int val) {
    int v = id++;
    t[v] = node(val);

    auto [l, r] = split(u, i - 1);
    attach(v, l, 0);
    attach(v, r, 1);
    return v;
}

int erase(int u, int i) {
    u = find(u, i);
    int l = t[u].ch[0], r = t[u].ch[1];
    t[l].p = t[r].p = 0;
    return merge(l, r);
}

int reverse(int u, int l, int r) {
    auto [a, b] = split(u, l - 1);
    auto [c, d] = split(b, r - l);
    t[c].flip ^= 1;
    return merge(merge(a, c), d);
}

int update(int u, int l, int r, int x, int y) {
    auto [a, b] = split(u, l - 1);
    auto [c, d] = split(b, r - l);
    t[c].lazyb = x;
    t[c].lazyc = y;
    return merge(merge(a, c), d);
}

pair<int, int> query(int u, int l, int r) {
    auto [a, b] = split(u, l - 1);
    auto [c, d] = split(b, r - l);
    push(c);
    int ans = t[c].sum;
    u = merge(merge(a, c), d);
    return {u, ans};
}
```

Sqrt.h

86a79e, 41 lines

```
struct Sqrt {
    int sz, B, n;
    vector<vector<>> blocks;
    vi A;
    Sqrt(vi &A_) : A(A_) {
        n = A.size();
        sz = sqrt(n);
        B = (n + sz - 1) / sz;
        blocks.resize(B);
        for (int b = 0; b < B; b++) build_block(b);
    }
    void build_block(int b) {
        int l = b * sz, r = min(n, (b + 1) * sz);
        blocks[b].clear();
        for (int i = l; i < r; i++) blocks[b].push_back();
    }
    ll query_block(int b, int x) {
    }
    ll query(int l, int r, int x) {
        ll ans = 1;
        int bl = l / sz, br = r / sz;
        if (bl == br) {
            for (int i = l; i <= r; i++) {
            }
        } else {
            for (int i = l; i < sz * (bl + 1); i++) {
            }
            for (int b = bl + 1; b < br; b++) {
                query_block(b, x);
            }
            for (int i = br * sz; i <= r; i++) {
            }
```

```
        }
        return ans;
    }
    void update(int p, int x) {
        A[p] = x;
        int b = p / sz;
        build_block(b);  // if all block needs to be rebuilt
    }
};
```

Graphs (3)

ArticulationPoints.h

126691, 34 lines

```
vector<bool> art;
vector<int> low, num;
int timer;
void dfs(int u, int p) {
    low[u] = num[u] = timer++;
    int children = 0;
    for (int v : AL[u]) {
        if (num[v] == -1) {
            dfs(v, u);
            low[u] = min(low[u], low[v]);
            children++;
            if (low[v] >= num[u] && p != -1) {
                art[u] = 1;
            }
        } else if (p != v) {
            low[u] = min(low[u], num[v]);
        }
    }
    if (p == -1 && children > 1) {
        art[u] = 1;
    }
}

void art_points() {
    low.assign(n, -1);
    num.assign(n, -1);
    art.assign(n, 0);
    timer = 0;
    for (int u = 0; u < n; u++) {
        if (num[u] == -1) {
            dfs(u, -1);
        }
    }
}
```

BCCs.h

a89b1c, 62 lines

```
struct BCC {
    vector<vector<int>>&AL, comps;
    vector<int> num, low, art, id;
    int n, timer, c;
    stack<int> stk;
    BCC(vector<vector<int>>& AL_) : AL(AL_) {
        n = AL.size();
        num.resize(n);
        low.resize(n);
        art.resize(n);
        id.resize(n);
        timer = 0, c = 0;
        dfs(0, -1);
    }

    void dfs(int u, int p) {
        num[u] = low[u] = ++timer;
        stk.push(u);
        for (int v : AL[u]) {
            if (v == p) continue;
            if (num[v]) {
                low[u] = min(low[u], num[v]);
            } else {
                dfs(v, u);
                low[u] = min(low[u], low[v]);
                if (low[v] >= num[u]) {
                    comps.push_back({u});
                    art[u] = num[u] > 1 || num[v] > 2;
                    while (comps[c].back() != v) {
                        comps[c].push_back(stk.top());
                        stk.pop();
                    }
                    c++;
                }
            }
        }
    }
    // block-cut tree
    vector<vector<int>> get_tree() {
        vector<vector<int>> t;
        int i = 0;
        for (int u = 0; u < n; u++) {
            if (art[u]) {
                id[u] = i++;
                t.push_back({});
            }
        }
        for (auto& comp : comps) {
            int cur = i++;
            t.push_back({});
            for (int u : comp) {
                if (art[u]) {
                    t[id[u]].push_back(cur);
                    t[cur].push_back(id[u]);
                } else {
                    id[u] = cur;
                }
            }
        }
        return t;
    }
};
```

Bridges.h

14f73a, 29 lines

```
vector<int> num, low;
vector<bool> bridge;
int timer;
void dfs(int u, int p) {
    num[u] = low[u] = ++timer;
    for (auto [v, i] : AL[u]) {
        if (!num[v]) {
            dfs(v, u);
            if (low[v] > num[u]) {
                bridge[i] = 1;
            }
            low[u] = min(low[u], low[v]);
        } else if (v != p) {
            low[u] = min(low[u], num[v]);
        }
    }
}
void bridges() {
    bridge.assign(m, 0);
    num.assign(n, -1);
    low.assign(n, -1);
    timer = 0;

    for (int u = 0; u < n; u++) {
        if (num[u] == -1) {
            dfs(u, -1);
        }
    }
}
```

Dinic.h

Time: $\mathcal{O}\left(V^2E\right)$

12d9b2, 67 lines

```
// A unit network is a network in which for any vertex except
// s and t either incoming or outgoing edge is unique and has
// unit capacity. This case: O(E * sqrt(V))
// In a more generic settings when all edges have unit
// capacities, but the number of incoming and outgoing edges
// is unbounded: O(E * sqrt(E)), O(E * V2/3)

using edge = array<ll, 3>;
const ll INF = 1e18;
struct max_flow {
    int n;
    vector<edge> EL;
    vector<vi> AL;
    vi d, last;
    vii p;
    bool bfs(int s, int t) {
        d.assign(n, -1);
        d[s] = 0;
        queue<int> q({s});
        p.assign(n, {-1, -1});
        while (!q.empty()) {
            int u = q.front();
            q.pop();
            if (u == t) break;
            for (auto& id : AL[u]) {
                auto& [v, cap, flow] = EL[id];
                if (cap - flow > 0 && d[v] == -1)
                    d[v] = d[u] + 1, q.push(v), p[v] = {u, id};
            }
        }
        return d[t] != -1;
    }
    ll dfs(int u, int t, ll f = INF) {
        if ((u == t) || (f == 0)) return f;
        for (int& i = last[u]; i < AL[u].size(); i++) {
            auto& [v, cap, flow] = EL[AL[u][i]];
            if (d[v] != d[u] + 1) continue;
            if (ll pushed = dfs(v, t, min(f, cap - flow))) {
                flow += pushed;
                auto& rflow = EL[AL[u][i] ^ 1][2];
                rflow -= pushed;
                return pushed;
            }
        }
        return 0;
    }

    max_flow(int n_) : n(n_) {
        EL.clear();
        AL.assign(n, vi());
    }
    void add_edge(int u, int v, ll w, bool directed = true) {
        if (u == v) return;
        EL.push_back({v, w, 0});
        AL[u].push_back(EL.size() - 1);
        EL.push_back({u, directed ? 0 : w, 0});
        AL[v].push_back(EL.size() - 1);
```

```
    }
    ll dinic(int s, int t) {
        ll mf = 0;
        while (bfs(s, t)) {
            last.assign(n, 0);
            while (ll f = dfs(s, t)) mf += f;
        }
        return mf;
    }
};
```

DomTree.h

47656a, 85 lines

```
void solve() {
    int n, m;
    cin >> n >> m;
    vector<vi> AL(n);
    vector<vi> ALr(n);
    for (int i = 0; i < m; i++) {
        int a, b;
        cin >> a >> b;
        AL[--a].push_back(--b);
        ALr[b].push_back(a);
    }

    vi visit_order;
    vi visit_time(n, 1e9);
    vi visit_parent(n, -1);
    vector<bool> visited(n, false);
    auto dfs = [&](auto self, int at, int prev) {
        if (visited[at]) {
            return;
        }
        visited[at] = true;

        visit_time[at] = visit_order.size();
        visit_order.push_back(at);
        visit_parent[at] = prev;
        for (int next : AL[at]) {
            self(self, next, at);
        }
    };
    dfs(dfs, 0, -1);

    vi dsu_parent(n, -1);
    vi dsu_min(n, 1e9);
    auto find = [&](auto self, int x) {
        if (dsu_parent[x] == -1) {
            return x;
        }
        if (dsu_parent[dsu_parent[x]] != -1) {
            int parent_res = self(self, dsu_parent[x]);
            if (visit_time[parent_res] < visit_time[dsu_min[x
                ]]) {
                dsu_min[x] = parent_res;
            }
            dsu_parent[x] = dsu_parent[dsu_parent[x]];
        }

        return dsu_min[x];
    };

    vi sdom(n, -1);
    for (int i = visit_order.size() - 1; i > 0; i--) {
        int c = visit_order[i];
        for (int from : ALr[c]) {
            int find_res = find(find, from);
            if (sdom[c] == -1 || visit_time[find_res] <
                visit_time[sdom[c]]) {
                sdom[c] = find_res;
            }
        }

        dsu_parent[c] = visit_parent[c];
        dsu_min[c] = sdom[c];
    }

    vii sdom_times(n, {1e9, -1});
    for (int c : visit_order) {
        if (c != 0) {
            sdom_times[c] = {visit_time[sdom[c]], c};
        }
    }
    // LCA only straight path and don't include last node in
        min
    LCA lca(visit_parent, sdom_times);

    vi rdom(n, -1);
    for (int c : visit_order) {
        if (c != 0) {
            rdom[c] = lca.get(sdom[c], c).second;
        }
    }

    vi idom(n, -1);
    for (int c : visit_order) {
        if (c != 0) {
            idom[c] = rdom[c] == c ? sdom[c] : idom[rdom[c]];
        }
    }
}
```

EulerTour.h

f50744, 31 lines

```
const int N = 1e5;
vector<pair<int, int>> AL[N];
bool seen[N];      // remove if directed
vector<int> path;  // reversed (important if path not cycle)
void dfs(int u) {
    while (!AL[u].empty()) {
        auto [v, i] = AL[u].back();
        AL[u].pop_back();
        if (seen[i]) continue;
        seen[i] = 1;
        dfs(v);
    }
    path.push_back(u);  // u&1 for de bruijn
}
string debruijn_seq(int n) {
    if (n == 1) {
        cout << "01";
        return;
    }
    int sz = (1 << (n - 1));
    for (int u = 0; u < sz; u++) {
        int v = (u & ((1 << (n - 2)) - 1)) << 1;
        AL[u].push_back(v);
        AL[u].push_back(v ^ 1);
    }
    dfs(0);
    reverse(path.begin(), path.end());
    n -= 2;
    while (n-- > 0) cout << 0;
    for (int x : path) cout << x;
}
```

FloydWarshall.h

d69558, 15 lines

```
for (int i = 0; i < V; ++i)
    for (int j = 0; j < V; ++j)
        p[i][j] = i;
for (int k = 0; k < V; ++k)
    for (int i = 0; i < V; ++i)
        for (int j = 0; j < V; ++j)
            if (AM[i][k] + AM[k][j] < AM[i][j]) {
                AM[i][j] = AM[i][k] + AM[k][j];
                p[i][j] = p[k][j];
            }

void print_path(int i, int j) {
    if (i != j) print_path(i, p[i][j]);
    printf(" %d", v);
}
```

LCT.h

751e7d, 160 lines

```
typedef struct snode* sn;
struct snode {
    sn p, c[2];     // parent, children
    sn extra;       // extra cycle node for "The Applicant"
    bool flip = 0;  // subtree flipped or not
    int sz;
    ll val, sum;       // value in node, # nodes in current
        splay tree
    ll sub, vsub = 0;  // vsub stores sum of virtual children

    snode(int _val) : val(_val) {
        p = c[0] = c[1] = extra = NULL;
        calc();
    }

    friend int getSz(sn x) { return x ? x->sz : 0; }
    friend ll getSub(sn x) { return x ? x->sub : 0; }
    friend ll getSum(sn x) { return x ? x->sum : 0; }

    void prop() {  // lazy prop
        if (!flip) return;
        swap(c[0], c[1]);
        flip = 0;
        for (int i = 0; i < 2; i++)
            if (c[i]) c[i]->flip ^= 1;
    }
    void calc() {  // recalc vals
        for (int i = 0; i < 2; i++)
            if (c[i]) c[i]->prop();
        sz = 1 + getSz(c[0]) + getSz(c[1]);
        sub = val + getSub(c[0]) + getSub(c[1]) + vsub;
        sum = val + getSum(c[0]) + getSum(c[1]);
    }

    int dir() {
        if (!p) return -2;
        for (int i = 0; i < 2; i++)
            if (p->c[i] == this) return i;
        return -1;  // p is path—parent pointer
    }  // -> not in current splay tree
    // test if root of current splay tree
    bool isRoot() { return dir() < 0; }
    friend void setLink(sn x, sn y, int d) {
        if (y) y->p = x;
        if (d >= 0) x->c[d] = y;
    }
    void rot() {  // assume p and p->p propagated
        assert(!isRoot());
        int x = dir();
        sn pa = p;
```

```
    setLink(pa->p, this, pa->dir());
    setLink(pa, c[x ^ 1], x);
    setLink(this, pa, x ^ 1);
    pa->calc();
}
void splay() {
    while (!isRoot() && !p->isRoot()) {
        p->p->prop(), p->prop(), prop();
        dir() == p->dir() ? p->rot() : rot();
        rot();
    }
    if (!isRoot()) p->prop(), prop(), rot();
    prop();
    calc();
}
sn fbo(int b) {  // find by order
    prop();
    int z = getSz(c[0]);  // of splay tree
    if (b == z) {
        splay();
        return this;
    }
    return b < z ? c[0]->fbo(b) : c[1]->fbo(b - z - 1);
}

void access() {  // bring this to top of tree, propagate
    for (sn v = this, pre = NULL; v; v = v->p) {
        v->splay();  // now switch virtual children
        if (pre) v->vsub -= pre->sub;
        if (v->c[1]) v->vsub += v->c[1]->sub;
        v->c[1] = pre;
        v->calc();
        pre = v;
    }
    splay();
    assert(!c[1]);  // right subtree is empty
}
void makeRoot() {
    access();
    flip ^= 1;
    access();
    assert(!c[0] && !c[1]);
}

friend sn lca(sn x, sn y) {
    if (x == y) return x;
    x->access(), y->access();
    if (!x->p) return NULL;
    x->splay();
    return x->p ?: x;  // y was below x in latter case
}  // access at y did not affect x -> not connected
friend bool connected(sn x, sn y) { return lca(x, y); }
// # nodes above
int distRoot() {
    access();
    return getSz(c[0]);
}
sn getRoot() {  // get root of LCT component
    access();
    sn a = this;
    while (a->c[0]) a = a->c[0], a->prop();
    a->access();
    return a;
}
sn getPar(int b) {  // get b-th parent on path to root
    access();
    b = getSz(c[0]) - b;
    assert(b >= 0);
    return fbo(b);
}  // can also get min, max on path to root, etc
ll query() {
    access();
    return sum;
}
ll sub_query(sn p) {
    p->makeRoot();
    access();
    return vsub + val;
}

void add(int v) {
    access();
    val += v;
    calc();
}
friend void link(sn x, sn y, bool force = 0) {
    assert(!connected(x, y));
    if (force)
        y->makeRoot();  // make x par of y
    else {
        y->access();
        assert(!y->c[0]);
    }
    x->access();
    setLink(y, x, 0);
    y->calc();
}
friend void cut(sn y) {  // cut y from its parent
    y->access();
    assert(y->c[0]);
    y->c[0]->p = NULL;
    y->c[0] = NULL;
    y->calc();
}
friend void cut(sn x, sn y) {  // if x, y adj in tree
    x->makeRoot();
    y->access();
    assert(y->c[0] == x && !x->c[0] && !x->c[1]);
    cut(y);
}
};
```

MCBM.h

Time: $\mathcal{O}(kE)$, k = $\mathcal{O}(V)$

69a80e, 34 lines

```
//
vi match, vis;

int aug(int L) {
    if (vis[L]) return 0;
    vis[L] = 1;
    for (auto& R : AL[L])
        if ((match[R] == -1) || aug(match[R])) {
            match[R] = L;
            return 1;
        }
    return 0;
}

unordered_set<int> freeV;
for (int L = 0; L < nleft; ++L) freeV.insert(L);  // initial
    assumption
match.assign(nleft + nright, -1);
int MCBM = 0;

for (int L = 0; L < nleft; ++L) {  // O(V+E)
    vi candidates;
    for (int& R : AL[L])
        if (match[R] == -1) candidates.push_back(R);
    if (!candidates.empty()) {
        ++MCBM;
        freeV.erase(L);
        int a = rand() % (int)candidates.size();
        match[candidates[a]] = L;
    }
}
for (auto& f : freeV) {
    vis.assign(nleft, 0);
    MCBM += aug(f);
}
```

MCMF.h

d92d98, 80 lines

```
struct MCMF {
    using F = ll;
    using C = ll;  // flow type, cost type
    struct Edge {
        int u, v;
        F flo, cap;
        C cost;
    };

    int N;
    vector<C> p, dist;
    vi pre;
    vector<Edge> eds;
    vector<vi> adj;

    MCMF(int _N) {
        N = _N;
        p.resize(N), dist.resize(N), pre.resize(N), adj.resize(
            N);
    }
    void add(int u, int v, F cap, C cost) {
        assert(cap >= 0);
        adj[u].push_back(size(eds));
        eds.push_back({u, v, 0, cap, cost});
        adj[v].push_back(size(eds));
        eds.push_back({v, u, 0, 0, -cost});
    }
    // use asserts, don't try smth dumb
    bool path(int s, int t) {  // find lowest cost path to send
        flow through
        const C inf = numeric_limits<C>::max();
        for (int i = 0; i < N; i++)
            dist[i] = inf;
        using T = pair<C, int>;
        priority_queue<T, vector<T>, greater<T>> todo;
        todo.push({dist[s] = 0, s});
        while (size(todo)) {  // Dijkstra
            T x = todo.top();
            todo.pop();
            if (x.first > dist[x.second]) continue;
            for (auto e : adj[x.second]) {
                const Edge& E = eds[e];  // all weights should
                    be non-negative
                if (E.flo < E.cap && ckmin(dist[E.v], (x.first
                    + E.cost + p[x.second] - p[E.v])))
                    pre[E.v] = e, todo.push({dist[E.v], E.v});
            }
        }  // if costs are doubles, add some EPS so you
        // don't traverse ~0-weight cycle repeatedly
        return dist[t] != inf;  // return flow
    }
    pair<F, C> calc(int s, int t) {
        assert(s != t);
        for (int i = 0; i < N; i++)
```

```
            for (int e = 0; e < size(eds); e++) {
                const Edge& E = eds[e];  // Bellman–Ford
                if (E.cap) ckmin(p[E.v], p[eds[e ^ 1].v] + E.
                    cost);
            }
        F totFlow = 0;
        C totCost = 0;
        while (path(s, t)) {  // p -> potentials for Dijkstra
            for (int i = 0; i < N; i++)
                p[i] += dist[i];  // don't matter for
                    unreachable nodes
            F df = numeric_limits<F>::max();
            for (int x = t; x != s; x = eds[pre[x] ^ 1].v) {
                const Edge& E = eds[pre[x]];
                ckmin(df, E.cap - E.flo);
            }
            totFlow += df;
            totCost += (p[t] - p[s]) * df;
            for (int x = t; x != s; x = eds[pre[x] ^ 1].v)
                eds[pre[x]].flo += df, eds[pre[x] ^ 1].flo -=
                    df;
        }  // get max flow you can send along path
        return {totFlow, totCost};
    }
    vii get_edges() {
        vii res;
        for (auto e : eds) {
            if (e.flo > 0 && e.cost)
                res.push_back({e.u, e.v});
        }
        return res;
    }
};
```

MinPathCover.h

c8220d, 14 lines

```
vi idl(n), idr(n);
int nleft = 0, nright = 0;
for (int u = 0; u < n; u++) {
    if (out[u]) idl[u] = nleft++;
}
for (int u = 0; u < n; u++) {
    if (in[u]) idr[u] = nleft + nright++;
}
for (int u = 0; u < n; u++) {
    for (int v : ALt[u]) {
        AL[idl[u]].push_back(idr[v]);
    }
}
// n - MCBM
```

SCCs.h

a29829, 36 lines

```
int timer, scc_cnt;
vector<int> num, low, vis, id;
stack<int> stk;
void dfs(int u) {
    low[u] = num[u] = timer;
    timer++;
    stk.push(u);
    vis[u] = 1;
    for (auto v : AL[u]) {
        if (num[v] == -1)
            dfs(v);
        if (vis[v])
            low[u] = min(low[u], low[v]);
    }
    if (low[u] == num[u]) {
        ++scc_cnt;
        while (1) {
            int v = stk.top();
            stk.pop();
            id[v] = u;
            vis[v] = 0;
            if (u == v) break;
        }
    }
}
void sccs() {
    id.assign(n, -1);
    num.assign(n, -1);
    low.assign(n, 0);
    vis.assign(n, 0);
    while (!stk.empty()) stk.pop();
    timer = scc_cnt = 0;
    for (int u = 0; u < n; ++u)
        if (num[u] == -1)
            dfs(u);
}
```

SPFA.h

cac3c6, 36 lines

```
if (mn_edge >= 0) {
    cout << mn_edge << "\n";
    return;
}
queue<int> q;
vll dist(n, 1e18);
vi freq(n, 0);
vector<bool> in_queue(n, 0);

for (int i = 0; i < n; i++) {
    q.push(i);
    dist[i] = 0;
    in_queue[i] = 1;
}
ll ans = LONG_LONG_MAX;
bool neg_cycle = false;
while (!q.empty() && !neg_cycle) {
    int u = q.front();
    q.pop();
    in_queue[u] = 0;
    for (auto& [v, w] : AL[u]) {
        if (dist[u] + w >= dist[v]) continue;
        dist[v] = dist[u] + w;
        ans = min(ans, dist[v]);
        freq[v]++;

        if (!in_queue[v]) {
            q.push(v);
            in_queue[v] = 1;
        }
        if (freq[v] > n) {
            neg_cycle = true;
            break;
        }
    }
}
```

TwoSAT.h

Time: $\mathcal{O}(n+m)$

234772, 46 lines

```
struct TwoSAT {
    vector<vi> AL;
    vector<bool> ans;
    TwoSAT(int n) : AL(2 * n), ans(n) {}

    void add_disj(int a, int b, bool na, bool nb) {
        int u = 2 * a ^ na, v = 2 * b ^ nb;
        AL[u ^ 1].push_back(v);
        AL[v ^ 1].push_back(u);
    }
    bool solve() {
        int n = AL.size();
        vi in_stk(n), low(n), num(n, -1), id(n);
        int scc = 0, ctr = 0;
        stack<int> stk;
        auto tarjan = [&](auto self, int u) -> void {
            low[u] = num[u] = ctr++;
            in_stk[u] = 1;
            stk.push(u);
            for (int v : AL[u]) {
                if (num[v] == -1) self(self, v);
                if (in_stk[v]) ckmin(low[u], low[v]);
            }

            if (low[u] == num[u]) {
                scc++;
                while (1) {
                    int v = stk.top();
                    stk.pop();
                    id[v] = -scc;
                    in_stk[v] = 0;
                    if (v == u) break;
                }
            }
        };

        for (int u = 0; u < n; u++) {
            if (num[u] == -1) tarjan(tarjan, u);
        }
        for (int u = 0; u < n; u += 2) {
            if (id[u] == id[u ^ 1]) return false;
            ans[u / 2] = id[u ^ 1] < id[u];
        }
        return true;
    }
};
```

GlobalMinCut.h

Description: Find a global minimum cut in an undirected graph, as represented by an adjacency matrix.
Time: $\mathcal{O}(V^3)$

8b0e19, 21 lines

```
pair<int, vi> globalMinCut(vector<vi> mat) {
    pair<int, vi> best = {INT_MAX, {}};
    int n = sz(mat);
    vector<vi> co(n);
    rep(i, 0, n) co[i] = {i};
    rep(ph, 1, n) {
        vi w = mat[0];
        size_t s = 0, t = 0;
        rep(it, 0, n - ph) {  // O(V^2) => O(E log V) with prio
            . queue
            w[t] = INT_MIN;
            s = t, t = max_element(all(w)) - w.begin();
            rep(i, 0, n) w[i] += mat[t][i];
        }
        best = min(best, {w[t] - mat[t][t], co[t]});
        co[s].insert(co[s].end(), all(co[t]));
        rep(i, 0, n) mat[s][i] += mat[t][i];
        rep(i, 0, n) mat[i][s] = mat[s][i];
        mat[0][t] = INT_MIN;
    }
    return best;
}
```

hopcroftKarp.h

Description: Fast bipartite matching algorithm. Graph g should be a list of neighbors of the left partition, and $btoa$ should be a vector full of -1's of the same size as the right partition. Returns the size of the matching. $btoa[i]$ will be the match for vertex i on the right side, or -1 if it's not matched.
Usage: `vi btoa(m, -1); hopcroftKarp(g, btoa);`
Time: $\mathcal{O}\left(\sqrt{V}E\right)$

f612e4, 42 lines

```
bool dfs(int a, int L, vector<vi>& g, vi& btoa, vi& A, vi& B) {
    if (A[a] != L) return 0;
    A[a] = -1;
    for (int b : g[a])
        if (B[b] == L + 1) {
            B[b] = 0;
            if (btoa[b] == -1 || dfs(btoa[b], L + 1, g, btoa, A
                 , B)) return btoa[b] = a, 1;
        }
    return 0;
}

int hopcroftKarp(vector<vi>& g, vi& btoa) {
    int res = 0;
    vi A(g.size()), B(btoa.size()), cur, next;
    for (;;) {
        fill(all(A), 0);
        fill(all(B), 0);
        cur.clear();
        for (int a : btoa)
            if (a != -1) A[a] = -1;
        rep(a, 0, sz(g)) if (A[a] == 0) cur.push_back(a);
        for (int lay = 1;; lay++) {
            bool islast = 0;
            next.clear();
            for (int a : cur)
                for (int b : g[a]) {
                    if (btoa[b] == -1) {
                        B[b] = lay;
                        islast = 1;
                    } else if (btoa[b] != a && !B[b]) {
                        B[b] = lay;
                        next.push_back(btoa[b]);
                    }
                }
            if (islast) break;
            if (next.empty()) return res;
            for (int a : next) A[a] = lay;
            cur.swap(next);
        }
        rep(a, 0, sz(g)) res += dfs(a, 0, g, btoa, A, B);
    }
}
```

GomoryHu.h

Description: Given a list of edges representing an undirected flow graph, returns edges of the Gomory-Hu tree. The max flow between any pair of vertices is given by minimum edge weight along the Gomory-Hu tree path.
Time: $\mathcal{O}(V)$ Flow Computations

"PushRelabel.h" 0418b3, 12 lines

```
typedef array<ll, 3> Edge;
vector<Edge> gomoryHu(int N, vector<Edge> ed) {
    vector<Edge> tree;
    vi par(N);
    rep(i, 1, N) {
        PushRelabel D(N);  // Dinic also works
        for (Edge t : ed) D.addEdge(t[0], t[1], t[2], t[2]);
        tree.push_back({i, par[i], D.calc(i, par[i])});
        rep(j, i + 1, N) if (par[j] == par[i] && D.leftOfMinCut
             (j)) par[j] = i;
    }
    return tree;
}
```

WeightedMatching.h

Description: Given a weighted bipartite graph, matches every node on the left with a node on the right such that no nodes are in two matchings and the sum of the edge weights is minimal. Takes cost[N][M], where cost[i][j] = cost for L[i] to be matched with R[j] and returns (min cost, match), where L[i] is matched with R[match[i]]. Negate costs for max cost. Requires $N \leq M$.
Time: $\mathcal{O}\left(N^2M\right)$

1e0fe9, 33 lines

```
pair<int, vi> hungarian(const vector<vi>& a) {
    if (a.empty()) return {0, {}};
    int n = sz(a) + 1, m = sz(a[0]) + 1;
    vi u(n), v(m), p(m), ans(n - 1);
    rep(i, 1, n) {
        p[0] = i;
        int j0 = 0;  // add "dummy" worker 0
        vi dist(m, INT_MAX), pre(m, -1);
        vector<bool> done(m + 1);
        do {  // dijkstra
            done[j0] = true;
            int i0 = p[j0], j1, delta = INT_MAX;
            rep(j, 1, m) if (!done[j]) {
                auto cur = a[i0 - 1][j - 1] - u[i0] - v[j];
                if (cur < dist[j]) dist[j] = cur, pre[j] = j0;
                if (dist[j] < delta) delta = dist[j], j1 = j;
            }
            rep(j, 0, m) {
                if (done[j])
                    u[p[j]] += delta, v[j] -= delta;
                else
                    dist[j] -= delta;
            }
            j0 = j1;
        } while (p[j0]);
        while (j0) {  // update alternating path
            int j1 = pre[j0];
            p[j0] = p[j1], j0 = j1;
        }
    }
    rep(j, 1, m) if (p[j]) ans[p[j] - 1] = j - 1;
    return {-v[0], ans};  // min cost
}
```

MinimumVertexCover.h

Description: Finds a minimum vertex cover in a bipartite graph. The size is the same as the size of a maximum matching, and the complement is a maximum independent set.

"DFSMatching.h" da4196, 23 lines

```
vi cover(vector<vi>& g, int n, int m) {
    vi match(m, -1);
    int res = dfsMatching(g, match);
    vector<bool> lfound(n, true), seen(m);
    for (int it : match)
        if (it != -1) lfound[it] = false;
    vi q, cover;
    rep(i, 0, n) if (lfound[i]) q.push_back(i);
    while (!q.empty()) {
        int i = q.back();
        q.pop_back();
        lfound[i] = 1;
        for (int e : g[i])
            if (!seen[e] && match[e] != -1) {
                seen[e] = true;
                q.push_back(match[e]);
            }
    }
    rep(i, 0, n) if (!lfound[i]) cover.push_back(i);
    rep(i, 0, m) if (seen[i]) cover.push_back(n + i);
    assert(sz(cover) == res);
    return cover;
}
```

EdgeColoring.h

Description: Given a simple, undirected graph with max degree D, computes a $(D + 1)$-coloring of the edges such that no neighboring edges share a color. (D-coloring is NP-hard, but can be done for bipartite graphs by repeated matchings of max-degree nodes.)
Time: $\mathcal{O}(NM)$

e210e2, 28 lines

```
vi edgeColoring(int N, vector<pii> eds) {
    vi cc(N + 1), ret(sz(eds)), fan(N), free(N), loc;
    for (pii e : eds) ++cc[e.first], ++cc[e.second];
    int u, v, ncols = *max_element(all(cc)) + 1;
    vector<vi> adj(N, vi(ncols, -1));
    for (pii e : eds) {
        tie(u, v) = e;
        fan[0] = v;
        loc.assign(ncols, 0);
        int at = u, end = u, d, c = free[u], ind = 0, i = 0;
        while (d = free[v], !loc[d] && (v = adj[u][d]) != -1)
            loc[d] = ++ind, cc[ind] = d, fan[ind] = v;
        cc[loc[d]] = c;
        for (int cd = d; at != -1; cd ^= c ^ d, at = adj[at][cd
            ]) swap(adj[at][cd], adj[end = at][cd ^ c ^ d]);
        while (adj[fan[i]][d] != -1) {
            int left = fan[i], right = fan[++i], e = cc[i];
            adj[u][e] = left;
            adj[left][e] = u;
            adj[right][e] = -1;
            free[right] = e;
        }
        adj[u][d] = fan[i];
        adj[fan[i]][d] = u;
        for (int y : {fan[0], u, end})
            for (int& z = free[y] = 0; adj[y][z] != -1; z++);
    }
    rep(i, 0, sz(eds)) for (tie(u, v) = eds[i]; adj[u][ret[i]]
        != v;)++ ret[i];
    return ret;
}
```

Trees (4)

LCA.h

55b19a, 59 lines

```
struct LCA {
    vector<vi>&AL, up, dp;
    vi d;
    int n, log;

    LCA(vector<vi>& AL_) : AL(AL_) {
        n = AL.size();
        d.resize(n);
        log = log2(n) + 2;
        up.assign(n, vi(log));
        dp.assign(n, vi(log));
        for (int u = 0; u < n; u++)
            // init dp[u][0]
            ;
        dfs(0, -1);
    }

    int merge(int a, int b) { return max(a, b); }
```

```
    void dfs(int u, int p) {
        for (int v : AL[u]) {
            if (v == p) continue;
            d[v] = d[u] + 1;
            up[v][0] = u;
            for (int j = 1; j < log; j++) {
                dp[v][j] = merge(dp[v][j - 1], dp[up[v][j -
                    1]][j - 1]);
                up[v][j] = up[up[v][j - 1]][j - 1];
            }
            dfs(v, u);
        }
    }

    int get(int u, int v) {
        if (d[u] < d[v]) swap(u, v);
        int k = d[u] - d[v];
        int res = 0;
        for (int j = 0; j < log; j++) {
            if (k & (1 << j)) {
                res = merge(res, dp[u][j]);
                u = up[u][j];
            }
        }
        if (u == v) {
            res = merge(res, dp[u][0]);
            return res;
        }
        for (int j = log - 1; j >= 0; j--) {
            if (up[u][j] != up[v][j]) {
                res = merge(res, dp[u][j]);
                res = merge(res, dp[v][j]);
                u = up[u][j];
                v = up[v][j];
            }
        }
        res = merge(res, dp[u][1]);
        res = merge(res, dp[v][0]);
        return res;
    }
};
```

CentroidDecomp.h

753c8c, 32 lines

```
int get_sizes(int u, int p = -1) {
    sz[u] = 1;
    for (int v : AL[u]) {
        if (v == p || removed[v]) {
            continue;
        }
        sz[u] += get_sizes(v, u);
    }
    return sz[u];
}

int get_centroid(int u, int tree_size, int p = -1) {
    for (int v : AL[u]) {
        if (v == p || removed[v]) {
            continue;
        }
        if (sz[v] * 2 > tree_size) {
            return get_centroid(v, tree_size, u);
        }
    }
    return u;
}

void centroid_decomp(int u) {
    int centroid = get_centroid(u, get_sizes(u));
    // do something
    removed[centroid] = true;
    for (int v : AL[centroid]) {
        if (removed[v]) continue;
        centroid_decomp(v);
    }
}
```

ChildQueries.h

0700ce, 12 lines

```
// tin[u] -> tout[u] doesn't include u
void dfs(int u, int p) {
    tin[u] = t, chin[u] = t;
    for (int v : AL[u]) {
        if (v != p) id[v] = t++;
    }
    chout[u] = t - 1;
    for (int v : AL[u]) {
        if (v != p) dfs(v, u);
    }
    tout[u] = t - 1;
}
```

HLD.h

e661e8, 62 lines

```
template <class T>
struct HLD {
    vector<vi>& AL;
    vi sz, par, d, id, top;
    SegTree<T> st;
    int ct;

    HLD(vector<vi>& AL_) : AL(AL_), st(AL_.size()) {
        int n = AL.size();
        sz.assign(n, 0);
        par.assign(n, -1);
        d.resize(n);
        id.assign(n, -1);
        top.assign(n, -1);
        ct = -1;
        dfs(0, -1);
        dfs_hld(0, -1, 0);
    }

    void dfs(int u, int p) {
        sz[u] = 1;
        for (int v : AL[u]) {
            if (v == p) continue;
            d[v] = d[u] + 1;
            par[v] = u;
            dfs(v, u);
            sz[u] += sz[v];
        }
    }
    void dfs_hld(int u, int p, int t) {
        id[u] = ++ct;
        top[u] = t;
        int h_ch = -1, h_sz = -1;
        for (int v : AL[u]) {
            if (v == p) continue;
            if (sz[v] > h_sz) {
                h_ch = v;
                h_sz = sz[v];
            }
        }
        if (h_ch == -1) return;
        dfs_hld(h_ch, u, t);
        for (int v : AL[u]) {
            if (v == p || v == h_ch) continue;
            dfs_hld(v, u, v);
        }
    }

    T query(int u, int v) {
        T ans = 0;
        while (top[u] != top[v]) {
            if (d[top[u]] < d[top[v]]) swap(u, v);
            ans = st.merge(ans, st.query(id[top[u]], id[u]));
            u = par[top[u]];
        }
        if (d[u] > d[v]) swap(u, v);
        ans = st.merge(ans, st.query(id[u], id[v]));
        return ans;
    }

    void update(int p, int x) { st.update(id[p], x); }
};
```

PathQueriesInv.h

321cdf, 24 lines

```
// pt updates
int timer;

void dfs(int u, int p) {
    in[u] = ++timer;
    for (int v : AL[u]) {
        if (v == p) continue;
        dfs(v, u);
    }
    out[u] = ++timer;
}
// main
dfs(0, -1);
for (int i = 0; i < n; i++) {
    st.update(in[i], a[i]);
    st.update(out[i], -a[i]);
}

st.update(in[i], x);
st.update(out[i], -x);
a[i] = x;

int anc = lca(u, v);
st.query(in[anc], in[u]) + st.query(in[anc], in[v]) - a[anc];
```

SmallLarge.h

efe04c, 13 lines

```
set<int> dfs(int u, int p) {
    set<int> st{a[u]};
    for (int v : AL[u]) {
        if (v == p) continue;
        set<int> cur = dfs(v, u);
        if (cur.size() > st.size()) {
            swap(cur, st);
        }
        for (int x : cur) st.insert(x);
    }
    // process st here to compute ans[u]. Offline queries
    return st;
}
```

VirtualTree.h

9f280e, 19 lines

```
bool sort_tin(const int& a, const int& b) { return tin[a] < tin
    [b]; }

vector<int> vtree(const vector<int>& key) {
    if (key.empty()) return {};
```

```
    vector<int> res = key;
    sort(res.begin(), res.end(), sort_tin);
    for (int i = 1; i < (int)key.size(); i++) {
        res.push_back(lca(key[i - 1], key[i]));
    }
    sort(res.begin(), res.end(), sort_tin);
    res.erase(unique(res.begin(), res.end()), res.end());
    for (int v : res) {
        vadj[v].clear();
    }
    for (int i = 1; i < (int)res.size(); i++) {
        vadj[lca(res[i - 1], res[i])].push_back(res[i]);
    }
    return res;
}
```

KRT.h

1afc4a, 15 lines

```
// id is the next unused id to assign to any new node created
// par is the standard definition of dsu parent
// adj will store the KRT
int id, par[2 * MAXN];
vector<int> adj[2 * MAXN];

int find(int u) { return par[u] == u ? u : par[u] = find(par[u
    ]); }

void unite(int u, int v) {
    u = find(u), v = find(v);
    if (u == v) return;
    par[u] = par[v] = par[id] = id;
    adj[id] = {u, v};
    id++;
}
```

DirectedMST.h

Description: Finds a minimum spanning tree/arborescence of a directed graph, given a root node. If no MST exists, returns -1.
Time: $\mathcal{O}(E \log V)$

"../data-structures/UnionFindRollback.h" 39e620, 69 lines

```
struct Edge {
    int a, b;
    ll w;
};
struct Node {
    Edge key;
    Node *l, *r;
    ll delta;
    void prop() {
        key.w += delta;
        if (l) l->delta += delta;
        if (r) r->delta += delta;
        delta = 0;
    }
    Edge top() {
        prop();
        return key;
    }
};
Node* merge(Node* a, Node* b) {
    if (!a || !b) return a ?: b;
    a->prop(), b->prop();
    if (a->key.w > b->key.w) swap(a, b);
    swap(a->l, (a->r = merge(b, a->r)));
    return a;
}
void pop(Node*& a) {
    a->prop();
    a = merge(a->l, a->r);
}

pair<ll, vi> dmst(int n, int r, vector<Edge>& g) {
    RollbackUF uf(n);
    vector<Node*> heap(n);
    for (Edge e : g) heap[e.b] = merge(heap[e.b], new Node{e});
    ll res = 0;
    vi seen(n, -1), path(n), par(n);
    seen[r] = r;
    vector<Edge> Q(n), in(n, {-1, -1}), comp;
    deque<tuple<int, int, vector<Edge>>> cycs;
    rep(s, 0, n) {
        int u = s, qi = 0, w;
        while (seen[u] < 0) {
            if (!heap[u]) return {-1, {}};
            Edge e = heap[u]->top();
            heap[u]->delta -= e.w, pop(heap[u]);
            Q[qi] = e, path[qi++] = u, seen[u] = s;
            res += e.w, u = uf.find(e.a);
            if (seen[u] == s) {
                Node* cyc = 0;
                int end = qi, time = uf.time();
                do cyc = merge(cyc, heap[w = path[--qi]]);
                while (uf.join(u, w));
                u = uf.find(u), heap[u] = cyc, seen[u] = -1;
                cycs.push_front({u, time, {&Q[qi], &Q[end]}});
            }
        }
        rep(i, 0, qi) in[uf.find(Q[i].b)] = Q[i];
    }

    for (auto& [u, t, comp] : cycs) {  // restore sol (optional
        )
        uf.rollback(t);
        Edge inEdge = in[u];
        for (auto& e : comp) in[uf.find(e.b)] = e;
        in[uf.find(inEdge.b)] = inEdge;
    }
    rep(i, 0, n) par[i] = in[i].a;
    return {res, par};
}
```

Strings (5)

AhoCorasick.h

689087, 78 lines

```
// at step i, longest suffix of s ending at i
// that is prefix of word in dict
const int S = 26;
struct AhoCorasick {
    vector<vvector<int>> AL, end;
    vector<array<int, S>> trie;
    vector<int> fail, vis, ans;
    int cnt = 1;

    AhoCorasick() : trie(2), end(2) {}
    void add(int i, string& s) {
        int u = 1;
        for (char cc : s) {
            int c = cc - 'a';
            if (!trie[u][c]) {
                trie[u][c] = ++cnt;
                trie.push_back({});
                end.push_back({});
            }
            u = trie[u][c];
        }
        end[u].push_back(i);
        ans.push_back(0);
    }
    // build after adding all patterns
    void build() {
        fail.resize(cnt + 1), AL.resize(cnt + 1);
        queue<int> q;

        for (int c = 0; c < S; c++) {
            if (trie[1][c]) {
                fail[trie[1][c]] = 1;
                q.push(trie[1][c]);
            } else {
                trie[1][c] = 1;
            }
        }

        while (!q.empty()) {
            int u = q.front();
            q.pop();
            for (int c = 0; c < 26; c++) {
                if (trie[u][c]) {
                    fail[trie[u][c]] = trie[fail[u]][c];
                    q.push(trie[u][c]);
                } else {
                    // failure built-in trie
                    trie[u][c] = trie[fail[u]][c];
                }
            }
        }

        // exit links reversed, proper suffix -> string
        for (int u = 2; u <= cnt; u++) {
            AL[fail[u]].push_back(u);
        }
    }

    void search(string& s) {
        vis.assign(cnt + 1, 0);
        int u = 1;
        for (char cc : s) {
            int c = cc - 'a';
            u = trie[u][c];
            vis[u] = 1;
        }
    }

    // handle exit links
    int dfs(int u) {
        int res = vis[u];
        for (int v : AL[u]) {
            res += dfs(v);
        }
        for (int i : end[u]) ans[i] = res;
        return res;
    }
};
```

Hashing.h

141266, 28 lines

```
const int N = 3e5 + 10;
mt19937 rng((uint32_t)chrono::steady_clock::now().
    time_since_epoch().count());
struct StringHash {
    const int p, M;
    int P[N], iP[N];

    StringHash(int _M) : p(uniform_int_distribution<int>(0, _M
        - 1)(rng)), M(_M) {
        P[0] = 1;
```

```
        for (int i = 1; i < N; i++) P[i] = (ll)P[i - 1] * p % M
            ;
        iP[N - 1] = minv(P[N - 1], M);
        for (int i = N - 2; i >= 0; i--) iP[i] = (ll)iP[i + 1]
            * p % M;
    }

    vi hash(const string& s) {
        int n = s.size();
        vi h(n);
        for (int i = 0; i < n; i++) {
            h[i] = (ll)P[i] * s[i] % M;
            if (i) h[i] += h[i - 1], h[i] %= M;
        }
        return h;
    }

    int get(vi& h, int l, int r) {
        if (l == 0) return h[r];
        return (ll)(h[r] - h[l - 1] + M) % M * iP[l] % M;
    }
} sh(1e9 + 7);
```

KMP.h

c8a2f6, 29 lines

```
vi get_pi(const string& s) {
    int n = s.size();
    vi pi(n);

    for (int i = 1; i < n; i++) {
        int j = pi[i - 1];
        while (j > 0 && s[i] != s[j]) {
            j = pi[j - 1];
        }
        if (s[i] == s[j]) j++;
        pi[i] = j;
    }
    return pi;
}
int count_occurences(const string& T, const string& P) {
    vi pi = get_pi(P);
    int i = 0, j = 0, freq = 0;
    while (i < T.size()) {
        while (j > 0 && T[i] != P[j]) j = pi[j - 1];
        if (T[i] == P[j])
            j++;
        i++;
        if (j == P.size()) {
            freq++;
            j = pi[j - 1];
        }
    }
    return freq;
}
```

Manacher.h

e8c8e4, 15 lines

```
// O(n) | p[0][i]: even pal right center
//      | p[1][i]: odd pal center
// pal[x][l + len / 2] >= len / 2 -> [l, r] pal
vector<vi> manacher(string &s) {
    int n = s.size();
    vector<vi> p(2, vi(n, 0));
    for (int z = 0, l = 0, r = 0; z < 2; z++, l = 0, r = 0)
        for (int i = 0; i < n; i++) {
            if (i < r) p[z][i] = min(r - i + !z, p[z][l + r - i
                + !z]);
            int L = i - p[z][i], R = i + p[z][i] - !z;
            while (L - 1 >= 0 && R + 1 < n && s[L - 1] == s[R +
                1]) p[z][i]++, L--, R++;
            if (R > r) l = L, r = R;
        }
    return p;
}
```

SuffixArray.h

8bd956, 125 lines

```
struct SuffixArray {
    vi sa, lcp;
    string s;
    // vector<vi> C;

    void build(string& s_) {
        s = s_;
        // C.clear();
        sa = suffix_array_construction(s);
        lcp = lcp_construction(s);
    }

    vi suffix_array_construction(string s) {
        s += "$";
        vi sorted_shifts = sort_cyclic_shifts(s);
        sorted_shifts.erase(sorted_shifts.begin());
        return sorted_shifts;
    }

    vi sort_cyclic_shifts(string& s) {
        int n = s.size();
        vi p(n), c(n), cnt(max(256, n), 0);
        for (int i = 0; i < n; i++)
            cnt[s[i]]++;
        for (int i = 1; i < 256; i++)
            cnt[i] += cnt[i - 1];
        for (int i = 0; i < n; i++)
            p[--cnt[s[i]]] = i;
        c[p[0]] = 0;
        int classes = 1;
        for (int i = 1; i < n; i++) {
            if (s[p[i]] != s[p[i - 1]])
                classes++;
            c[p[i]] = classes - 1;
        }
        vi pn(n), cn(n);
        // C.push_back(c);
        for (int h = 0; (1 << h) < n; ++h) {
            for (int i = 0; i < n; i++) {
                pn[i] = p[i] - (1 << h);
                if (pn[i] < 0)
                    pn[i] += n;
            }
            fill(cnt.begin(), cnt.begin() + classes, 0);
            for (int i = 0; i < n; i++)
                cnt[c[pn[i]]]++;
            for (int i = 1; i < classes; i++)
                cnt[i] += cnt[i - 1];
            for (int i = n - 1; i >= 0; i--)
                p[--cnt[c[pn[i]]]] = pn[i];
            cn[p[0]] = 0;
            classes = 1;
            for (int i = 1; i < n; i++) {
                ii cur = {c[p[i]], c[(p[i] + (1 << h)) % n]};
                ii prev = {c[p[i - 1]], c[(p[i - 1] + (1 << h))
                    % n]};
                if (cur != prev)
                    ++classes;
                cn[p[i]] = classes - 1;
            }
            swap(c, cn);
            // C.push_back(c);
        }

        return p;
    }

    vi lcp_construction(string& s) {
        int n = s.size();
        vi rank(n, 0);
        for (int i = 0; i < n; i++)
            rank[sa[i]] = i;

        int k = 0;
        vi lcp(n - 1, 0);
        for (int i = 0; i < n; i++) {
            if (rank[i] == n - 1) {
                k = 0;
                continue;
            }
            int j = sa[rank[i] + 1];
            while (i + k < n && j + k < n && s[i + k] == s[j +
                k])
                k++;
            lcp[rank[i]] = k;
            if (k)
                k--;
        }
        return lcp;
    }

    ii string_matching(string& p) {
        int n = s.size();
        int l = 0, r = n - 1;
        int lb = n, ub = n;
        while (l <= r) {
            int m = l + r >> 1;
            string sub = s.substr(sa[m], (int)p.size());
            if (sub >= p) {
                lb = m;
                r = m - 1;
            } else {
                l = m + 1;
            }
        }
        l = 0, r = n - 1;
        while (l <= r) {
            int m = l + r >> 1;
            string sub = s.substr(sa[m], (int)p.size());
            if (sub > p) {
                ub = m;
                r = m - 1;
            } else {
                l = m + 1;
            }
        }
        return {lb, ub};
    }

    /*int compare(int i, int j, int l) {
        int n = s.size(), k = lg2[l];
        pair<int, int> a = {C[k][i], C[k][(i + l - (1 << k)) %
            n]};
        pair<int, int> b = {C[k][j], C[k][(j + l - (1 << k)) %
            n]};
        return a == b ? 0 : a < b ? -1
                                  : 1;
    }*/
};
```

SuffixAuto.h

89d095, 169 lines

```
// DAWG; use dp.
struct SuffixAuto {
    struct state {
        int len, link;
        int next[28];
        state() : len(0), link(-1) {
            memset(next, -1, sizeof next);
        }
    };

    state* st;
    int sz, last;

    SuffixAuto() : st(0) {}

    SuffixAuto(const string& s) {
        build(s);
    }

    void build(const string& s) {
        delete[] st;
        st = new state[2 * s.size()];

        last = 0;
        sz = 1;

        for (char c : s) {
            extend(c);
        }
    }

    void extend(char _c) {
        int c = _c - 'A';

        int cur = sz++;
        st[cur].len = st[last].len + 1;
        int p = last;
        // cnt[cur] = 1;

        while (p != -1 && st[p].next[c] == -1) {
            st[p].next[c] = cur;
            p = st[p].link;
        }
        if (p == -1) {
            st[cur].link = 0;
        } else {
            int q = st[p].next[c];
            if (st[p].len + 1 == st[q].len) {
                st[cur].link = q;
            } else {
                int clone = sz++;
                // cnt[clone] = 0;
                st[clone].len = st[p].len + 1;
                copy(st[q].next, st[q].next + 28, st[clone].
                    next);
                st[clone].link = st[q].link;
                while (p != -1 && st[p].next[c] == q) {
                    st[p].next[c] = clone;
                    p = st[p].link;
                }
                st[q].link = st[cur].link = clone;
            }
        }
        last = cur;
    }
    ~SuffixAuto() { delete[] st; }
} sa;
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
    -- -- -- -- -- -- -- --int memo1[2 * N];
ll memo2[2 * N], memo3[2 * N];
// common substrings
int dp1(int u) {
    int& ans = memo1[u];
    if (~ans) return ans;
    ans = 0;
    for (int c = 0; c < 28; c++) {
        int v = sa.st[u].next[c];
        if (v == -1) continue;
        if (c == 26) {
            ans |= 1;
        } else if (c == 27)
            ans |= 2;
        else
            ans |= dp1(v);
    }
    return ans;
}
// number of distinct substrings (paths) starting at state u
ll dp2(int u) {
    if (dp1(u) != 3 || u == sa.last) return 0;
    ll& ans = memo2[u];
    if (~ans) return ans;
    ans = 1;
    for (int c = 0; c < 28; c++) {
        int v = sa.st[u].next[c];
        if (v == -1) continue;
        ans += dp2(v);
    }
    return ans;
}
ll dfs(int u) {
    if (dp1(u) != 3) return 0;
    ll& ans = memo3[u];
    if (~ans) return ans;
    if (u) {
        ll concat = (dp2(u) - 1 + M) % M;
        ans = concat % M * concat % M;
    } else
        ans = 0;
    for (int c = 0; c < 26; c++) {
        int v = sa.st[u].next[c];
        if (v == -1) continue;
        ans = (ans + dfs(v)) % M;
    }
    return ans;
}

// cnt[u]: nb of occurences of state u substrings (endpos size)
    .
// uncomment: cnt[cur] = 1, cnt[clone] = 0
// case distinct: replace cnt by 1
ll dfs_dp(int u) {
    if (dp[u] != 0)
        return dp[u];
    for (int c = 0; c < 26; c++) {
        int v = sa.st[u].next[c];
        if (v != -1)
            dp[u] += dfs_dp(v);
    }
    return dp[u] += cnt[u];
}
// kth substring
string ans;
void dfs(int u, ll& k) {
    if (k < cnt[u])
        return;
    else
        k -= cnt[u];
    for (int c = 0; c < 26; c++) {
        int v = sa.st[u].next[c];
        if (v != -1) {
            if (k >= dp[v])
                k -= dp[v];
            else {
                ans += char(c + 'a');
                dfs(v, k);
                return;
            }
        }
    }
}
vector<ii> v(2 * T.size());
for (int i = 0; i < v.size(); i++)
    v[i] = {sa.st[i].len, i};
sort(rall(v));
for (auto [len, id] : v)
    if (sa.st[id].link != -1)
        cnt[sa.st[id].link] += cnt[id];
cnt[0] = 1;

ll get_tot_len_diff_substings() {
    ll tot = 0;
    for (int i = 1; i < sz; i++) {
        ll shortest = sa.st[st[i].link].len + 1;
        ll longest = sa.st[i].len;

        ll num_strings = longest - shortest + 1;
        ll cur = num_strings * (longest + shortest) / 2;
        tot += cur;
    }
    return tot;
}
```

SuffixAutomaton.h

<bits/stdc++.h> f3e896, 200 lines

```
using namespace std;
typedef long long ll;

const int K = 26;
const char C = 'a';
struct SuffixAuto {
    struct state {
        int len, link;
        vector<int> next;
        state() : len(0), link(-1), next(K, -1) {}
    };

    int n;
    const string& S;
    vector<state> st;
    int sz, last;
    // optional
    vector<int> firstpos, cnt;  // firstpos <-> min(endpos),
        cnt <-> endpos_size

    SuffixAuto(const string& S_) : S(S_) {
        n = S.size();
        st.resize(2 * n);

        st[0].len = 0;
        st[0].link = -1;
        last = 0;
        sz = 1;

        cnt.resize(2 * n);
```

```
    firstpos.resize(2 * n);
    for (char c : S) {
        extend(c);
    }

    vector<pair<int, int>> v(2 * S.size());
    for (int i = 0; i < v.size(); i++)
        v[i] = {st[i].len, i};
    sort(v.rbegin(), v.rend());
    for (auto [len, id] : v)
        if (st[id].link != -1)
            cnt[st[id].link] += cnt[id];
    cnt[0] = 1;
}

void extend(char c_) {
    int c = c_ - C;

    int cur = sz++;
    st[cur].len = st[last].len + 1;
    int p = last;
    cnt[cur] = 1;
    firstpos[cur] = st[cur].len - 1;

    while (p != -1 && st[p].next[c] == -1) {
        st[p].next[c] = cur;
        p = st[p].link;
    }
    if (p == -1) {
        st[cur].link = 0;
    } else {
        int q = st[p].next[c];
        if (st[q].len == st[p].len + 1) {
            st[cur].link = q;
        } else {
            int clone = sz++;
            cnt[clone] = 0;
            firstpos[clone] = firstpos[q];
            st[clone].len = st[p].len + 1;
            st[clone].link = st[q].link;
            st[clone].next = st[q].next;
            while (p != -1 && st[p].next[c] == q) {
                st[p].next[c] = clone;
                p = st[p].link;
            }
            st[q].link = st[cur].link = clone;
        }
    }
    last = cur;
}

// applications
ll get_diff_strings() {
    ll tot = 0;
    for (int i = 1; i < sz; i++) {
        tot += st[i].len - st[st[i].link].len;
    }
    return tot;
}

ll get_tot_len_diff_substings() {
    ll tot = 0;
    for (int i = 1; i < sz; i++) {
        ll shortest = st[st[i].link].len + 1;
        ll longest = st[i].len;

        ll num_strings = longest - shortest + 1;
        ll cur = num_strings * (longest + shortest) / 2;
        tot += cur;
    }
    return tot;
}

int first_match(const string& P) {
    int cur = 0;
    for (char c_ : P) {
        int c = c_ - C;
        if (st[cur].next[c] == -1) return -1;

        cur = st[cur].next[c];
    }
    return firstpos[cur] - P.size() + 1;
}

string lcs(const string& T) {
    int v = 0, l = 0, best = 0, bestpos = 0;
    for (int i = 0; i < T.size(); i++) {
        int c = T[i] - C;
        while (v && st[v].next[c] == -1) {
            v = st[v].link;
            l = st[v].len;
        }
        if (st[v].next[c] != -1) {
            v = st[v].next[c];
            l++;
        }
        if (l > best) {
            best = l;
            bestpos = i;
        }
    }
    return T.substr(bestpos - best + 1, best);
}

vector<ll> dp;
ll dfs_dp(int u) {
    if (dp[u] != 0)
        return dp[u];
    for (int c = 0; c < K; c++) {
        int v = st[u].next[c];
        if (v != -1)
            dp[u] += dfs_dp(v);
    }
    return ++dp[u];  // *if not-distinct: dp += cnt*
}
void dfs(int u, ll& k, string& s) {
    if (k == 0)  // empty string starting from u. *if not-
        distinct: k<cnt*
        return;
    k--;  // remove empty string. *if not-distinct: k -=
        cnt*
    for (int c = 0; c < K; c++) {
        int v = st[u].next[c];
        if (v == -1) continue;
        if (k >= dp[v])
            k -= dp[v];
        else {
            s += char(c + C);
            dfs(v, k, s);
            return;
        }
    }
}
string kth_distinct_substring(ll k) {
    dp.resize(2 * n);
    string s;
    dfs_dp(0);
    dfs(0, k, s);
    return s;
}

string longest_repeating() {
    int best_len = -1, bfp = -1;

    for (int u = 1; u < 2 * n; u++) {
        if (cnt[u] <= 1 || st[u].len == 0) continue;
        int len = st[u].len;
        if (len > best_len) {
            best_len = len;
            bfp = firstpos[u];
        }
    }
    if (best_len == -1) return "-1";
    return S.substr(bfp - best_len + 1, best_len);
}

string shortest_unique() {
    int best_len = 1e9, bfp = 1e9;

    for (int u = 1; u < 2 * n; u++) {
        if (cnt[u] > 1 || st[u].len == 0) continue;
        int mn_len = st[st[u].link].len + 1;
        if (mn_len < best_len) {
            best_len = mn_len;
            bfp = firstpos[u];
        } else if (mn_len == best_len && firstpos[u] < bfp)
            {
            bfp = firstpos[u];
        }
    }

    return S.substr(bfp - best_len + 1, best_len);
}
};
```

Trie.h

f1358e, 42 lines

```
struct node {
    char c;
    bool exist;
    vector<node*> child;
    node(char c) : c(c), exist(0), child(26, 0) {}
};

struct Trie {
    node* root;
    Trie() { root = new node('!'); }
    void insert(string& s) {
        node* cur = root;
        for (int i = 0; i < s.size(); i++) {
            int c = s[i] - 'a';
            if (cur->child[c] == NULL) {
                cur->child[c] = new node(s[i]);
            }
            cur = cur->child[c];
        }
        cur->exist = 1;
    }
    bool search(string& s) {
        node* cur = root;
        for (int i = 0; i < s.size(); ++i) {
            int c = s[i] - 'a';
            if (cur->child[c] == NULL)
                return false;
            cur = cur->child[c];
        }
        return cur->exist;
```

```
    }
    bool starts_with(string& s) {
        node* cur = root;
        for (int i = 0; i < s.size(); ++i) {
            int c = s[i] - 'a';
            if (cur->child[c] == NULL)
                return false;
            cur = cur->child[c];
        }
        return true;
    }
};
```

Z.h

42005c, 16 lines

```
vi Z(string& s) {
    int n = s.size();
    vi z(n);
    int l = 0, r = 0;
    for (int i = 1; i < n; i++) {
        if (r >= i)
            z[i] = min(r - i, z[i - l]);
        while (i + z[i] < n && s[z[i]] == s[i + z[i]]) {
            z[i]++;
        }
        if (i + z[i] > r) {
            l = i, r = i + z[i];
        }
    }
    return z;
}
```

Geometry (6)

Circle.h

13b223, 19 lines

```
int insideCircle(point_i p, point_i c, int r) {
    int dx = p.x - c.x, dy = p.y - c.y;
    int Euc = dx * dx + dy * dy, rSq = r * r;
    return Euc < rSq ? 1 : Euc == rSq ? 0
                                      : -1;
    // inside/border/outside
}

bool circle2PtsRad(point p1, point p2, double r, point& c) {
    // to get the other center, reverse p1 and p2
    double d2 = (p1.x - p2.x) * (p1.x - p2.x) +
                (p1.y - p2.y) * (p1.y - p2.y);
    double det = r * r / d2 - 0.25;
    if (det < 0.0) return false;
    double h = sqrt(det);
    c.x = (p1.x + p2.x) * 0.5 + (p1.y - p2.y) * h;
    c.y = (p1.y + p2.y) * 0.5 + (p2.x - p1.x) * h;
    return true;
}
```

ClosestPoints.h

022c9a, 30 lines

```
// change comparisons when dealing with real coordinates
struct point {
    ll x, y;
    point(ll x = 0, ll y = 0) : x(x), y(y) {}
    bool operator<(point p) const {
        if (x != p.x) return x < p.x;
        return y < p.y;
    }
};
sort(all(points));

auto cmp = [](const point& p, const point& q) {
    if (p.y != q.y) return p.y < q.y;
    return p.x < q.x;
};

set<point, decltype(cmp)> active(cmp);
int leftmost_i = 0;
ll min_dist = 1e18;
for (point curr : points) {
    while ((curr.x - points[leftmost_i].x) * (curr.x - points[
        leftmost_i].x) > min_dist) {
        active.erase(points[leftmost_i]);
        leftmost_i++;
    }
    auto it = active.lower_bound({curr.x, curr.y - (int)sqrt(
        min_dist)});
    for (; it != active.end() && it->y <= curr.y + (int)sqrt(
        min_dist);
        it++)
        min_dist = min(min_dist, dist2(*it, curr));
    active.insert(curr);
}
```

Geo3D.h

65ec37, 7 lines

```
double gcDist(double pLa, double pLo, double qLa, double qLo,
    double r) {
    pLa *= M_PI / 180;
    pLo *= M_PI / 180;
    qLa *= M_PI / 180;
    qLo *= M_PI / 180;
    return r * acos(cos(pLa) * cos(pLo) * cos(qLa) * cos(qLo) +
        cos(pLa) * sin(pLo) * cos(qLa) * sin(qLo) + sin(pLa)
        * sin(qLa));
}
```

InterSegs.h

d5392c, 103 lines

```
struct point {
    int x, y;
    point(int x, int y) : x(x), y(y) {}
};

struct segment {
    point p, q;
    int id;
    segment(point p, point q, int i) : p(p), q(q), id(i) {}
    double get_y(double x) const {
        if (abs(p.x - q.x) < EPS)
            return p.y;
        return p.y + (q.y - p.y) * (x - p.x) / (q.x - p.x);
    }
};

bool on_segment(point p, point q, point r) {
    return q.x <= max(p.x, r.x) && q.x >= min(p.x, r.x) &&
           q.y <= max(p.y, r.y) && q.y >= min(p.y, r.y);
}

int orientation(point p, point q, point r) {
    point pq(q.x - p.x, q.y - p.y), pr(r.x - p.x, r.y - p.y);
    int cross = pq.x * pr.y - pq.y * pr.x;

    if (cross == 0) return 0;
    return (cross > 0) ? 1 : -1;
}
bool seg_intersect(const segment& s1, const segment& s2) {
    point p1 = s1.p, q1 = s1.q, p2 = s2.p, q2 = s2.q;
    int o1 = orientation(p1, q1, p2);
    int o2 = orientation(p1, q1, q2);
    int o3 = orientation(p2, q2, p1);
    int o4 = orientation(p2, q2, q1);

    if (o1 != o2 && o3 != o4)
        return true;

    if (o1 == 0 && on_segment(p1, p2, q1)) return true;
    if (o2 == 0 && on_segment(p1, q2, q1)) return true;
    if (o3 == 0 && on_segment(p2, p1, q2)) return true;
    if (o4 == 0 && on_segment(p2, q1, q2)) return true;

    return false;
}
bool operator<(const segment& a, const segment& b) {
    double x = max(min(a.p.x, a.q.x), min(b.p.x, b.q.x));
    return a.get_y(x) < b.get_y(x) - EPS;
}

struct event {
    double x;
    int tp, id;

    event() {}
    event(double x, int tp, int id) : x(x), tp(tp), id(id) {}

    bool operator<(const event& e) const {
        if (abs(x - e.x) > EPS)
            return x < e.x;
        return tp > e.tp;
    }
};

set<segment> s;
vector<set<segment>::iterator> where;

set<segment>::iterator prev(set<segment>::iterator it) {
    return it == s.begin() ? s.end() : --it;
}
set<segment>::iterator next(set<segment>::iterator it) {
    return ++it;
}

ii find_inter(const vector<segment>& a) {
    int n = (int)a.size();
    vector<event> e;
    for (int i = 0; i < n; ++i) {
        e.push_back(event(min(a[i].p.x, a[i].q.x), +1, i));
        e.push_back(event(max(a[i].p.x, a[i].q.x), -1, i));
    }
    sort(e.begin(), e.end());

    s.clear();
    where.resize(a.size());
    for (size_t i = 0; i < e.size(); ++i) {
        int id = e[i].id;
        if (e[i].tp == +1) {
            set<segment>::iterator nxt = s.lower_bound(a[id]),
                prv = prev(nxt);
            if (nxt != s.end() && intersect(*nxt, a[id]))
                return {nxt->id, id};
            if (prv != s.end() && intersect(*prv, a[id]))
                return {prv->id, id};
            where[id] = s.insert(nxt, a[id]);
        } else {
            set<segment>::iterator nxt = next(where[id]), prv =
                prev(where[id]);
```

```
            if (nxt != s.end() && prv != s.end() && intersect(*
                 nxt, *prv))
                return {prv->id, nxt->id};
            s.erase(where[id]);
        }
    }
    return {-1, -1};
}
```

InterSegsHV.h

0af8ef, 78 lines

```
// change comp when dealing with real coordinates
struct point {
    int x, y;
    point(int x = 0, int y = 0) : x(x), y(y) {}
    bool operator<(point p) const {
        if (x != p.x) return x < p.x;
        return y < p.y;
    }
};
struct segment {
    point l, r;
    bool h;
    segment(point l, point r) : l(l), r(r) {
        if (l.x == r.x)
            h = 0;
        else
            h = 1;
    }
    bool operator<(segment s) const {
        if (l.y != s.l.y) return l.y < s.l.y;
        if (l.x != s.l.x) return l.x < s.l.x;
        return r.x < s.r.x;
    }
};
struct event {
    point p;
    segment s;
    int type;

    event(point p, segment s) : p(p), s(s) {
        if (s.h)
            if (s.l.x == p.x)
                type = 0;  // add
            else
                type = 2;  // remove
        else
            type = 1;  // vertical event
    }
    bool operator<(event e) const {
        if (p.x != e.p.x) return p.x < e.p.x;
        return type < e.type;
    }
};
void solve() {
    int n;
    cin >> n;
    vector<event> events;
    for (int i = 0; i < n; i++) {
        point p, q;
        cin >> p.x >> p.y >> q.x >> q.y;
        segment s(p, q);
        event e(p, s);
        events.push_back(e);
        if (s.h) {
            event e2(q, s);
            events.push_back(e2);
        }
    }
    sort(all(events));
    set<segment> active;
    set<point> inter;
    for (event e : events) {
        if (e.type == 0) {
            active.insert(e.s);
        } else if (e.type == 1) {
            auto it = active.lower_bound(
                segment({(int)-1e9, e.s.l.y}, {(int)-1e9, e.s.l
                    .y}));
            for (; it != active.end() && it->l.y <= e.s.r.y; it
                ++) {
                // avoid coincident end-points
                if (it->l.x == e.p.x && (it->l.y == e.s.l.y ||
                    it->l.y == e.s.r.y)) continue;
                if (it->r.x == e.p.x && (it->r.y == e.s.l.y ||
                    it->l.y == e.s.r.y)) continue;
                inter.insert(point(e.p.x, it->l.y));
            }
        } else if (e.type == 2) {
            active.erase(e.s);
        }
    }
}
```

Line.h

854080, 134 lines

```
struct line {
    double a, b, c;
};
void pointsToLine(point p1, point p2, line& l) {
    if (fabs(p1.x - p2.x) < EPS)
        l = {1.0, 0.0, -p1.x};
    else {
        double a = -(double)(p1.y - p2.y) / (p1.x - p2.x);
        l = {a, 1.0, -(double)(a * p1.x) - p1.y};
    }
}
struct line2 {
    double m, c;
};

int pointsToLine2(point p1, point p2, line2& l) {
    if (p1.x == p2.x) {
        l.m = INF;
        l.c = p1.x;
        return 0;
    } else {
        l.m = (double)(p1.y - p2.y) / (p1.x - p2.x);
        l.c = p1.y - l.m * p1.x;
        return 1;
    }
}

bool areParallel(line l1, line l2) {
    return (fabs(l1.a - l2.a) < EPS) && (fabs(l1.b - l2.b) <
        EPS);
}

bool areSame(line l1, line l2) {
    return areParallel(l1, l2) && (fabs(l1.c - l2.c) < EPS);
}

bool areIntersect(line l1, line l2, point& p) {
    if (areParallel(l1, l2)) return false;
    p.x = (l2.b * l1.c - l1.b * l2.c) / (l2.a * l1.b - l1.a *
        l2.b);
    if (fabs(l1.b) > EPS)
        p.y = -(l1.a * p.x + l1.c);
    else
        p.y = -(l2.a * p.x + l2.c);
    return true;
}

struct vec {
    double x, y;
    vec(double _x, double _y) : x(_x), y(_y) {}
};
vec toVec(const point& a, const point& b) {
    return vec(b.x - a.x, b.y - a.y);
}
vec scale(const vec& v, double s) {
    return vec(v.x * s, v.y * s);
}

point translate(const point& p, const vec& v) {
    return point(p.x + v.x, p.y + v.y);
}

// convert point and slope to line
void pointSlopeToLine(point p, double m, line& l) {
    l.a = -m;
    l.b = 1;
    l.c = -((l.a * p.x) + (l.b * p.y));
}

void closestPoint(line l, point p, point& ans) {
    line perpendicular;
    if (fabs(l.b) < EPS) {
        ans.x = -(l.c);
        ans.y = p.y;
        return;
    }
    if (fabs(l.a) < EPS) {
        ans.x = p.x;
        ans.y = -(l.c);
        return;
    }
    pointSlopeToLine(p, 1 / l.a, perpendicular);
    areIntersect(l, perpendicular, ans);
}

// returns the reflection of point on a line
void reflectionPoint(line l, point p, point& ans) {
    point b;
    closestPoint(l, p, b);
    vec v = toVec(p, b);
    ans = translate(translate(p, v), v);
}

double dot(vec a, vec b) { return (a.x * b.x + a.y * b.y); }

double norm_sq(vec v) { return v.x * v.x + v.y * v.y; }

double angle(const point& a, const point& o, const point& b) {
    vec oa = toVec(o, a), ob = toVec(o, b);  // a != o != b
    return acos(dot(oa, ob) / (sqrt(norm_sq(oa)) * sqrt(norm_sq
        (ob))));
}  // angle aob in rad, sqrt both to avoid overflow

// returns the distance from p to the line defined by
// two points a and b (a and b must be different)
// the closest point is stored in the 4th parameter
double distToLine(point p, point a, point b, point& c) {
    vec ap = toVec(a, p), ab = toVec(a, b);
    double u = dot(ap, ab) / norm_sq(ab);
    // formula: c = a + u*ab
    c = translate(a, scale(ab, u));
```

```
    return dist(p, c);
}

double distToLineSegment(point p, point a, point b, point& c) {
    vec ap = toVec(a, p), ab = toVec(a, b);
    double u = dot(ap, ab) / norm_sq(ab);
    if (u < 0.0) {
        c = point(a.x, a.y);
        return dist(p, a);
    }
    if (u > 1.0) {
        c = point(b.x, b.y);
        return dist(p, b);
    }
    return distToLine(p, a, b, c);
}

double cross(vec a, vec b) { return a.x * b.y - a.y * b.x; }

bool ccw(point p, point q, point r) {
    return cross(toVec(p, q), toVec(p, r)) > EPS;
}

bool collinear(point p, point q, point r) {
    return fabs(cross(toVec(p, q), toVec(p, r))) < EPS;
}
```

Point.h

1b30c5, 22 lines

```
#define M_PI 3.14159265358979323846
struct point {
    double x, y;
    point(double _x = 0.0, double _y = 0.0) : x(_x), y(_y) {}
    bool operator<(point other) const {
        if (fabs(x - other.x) > EPS)
            return x < other.x;
        return y < other.y;
    }
    bool operator==(point other) const {
        return (fabs(x - other.x) < EPS && (fabs(y - other.y) <
              EPS));
    }
};
double dist(point p1, point p2) {
    return hypot(p1.x - p2.x, p1.y - p2.y);
}
// rotate p by theta degrees CCW w.r.t origin (0, 0)
point rotate(point p, double theta) {
    double rad = DEG_to_RAD(theta);
    return point(p.x * cos(rad) - p.y * sin(rad),
                 p.x * sin(rad) + p.y * cos(rad));
}
```

Polygon.h

6caa01, 158 lines

```
double perimeter(const vector<point>& P) {
    double ans = 0.0;
    for (int i = 0; i < (int)P.size() - 1; ++i)
        // note: P[n-1] = P[0]
        ans += dist(P[i], P[i + 1]);
    return ans;
}
double area(const vector<point>& P) {
    double ans = 0.0;
    for (int i = 0; i < (int)P.size() - 1; ++i)
        ans += (P[i].x * P[i + 1].y - P[i + 1].x * P[i].y);
    return fabs(ans) / 2.0;
}
// returns true if we always make the same turn
bool isConvex(const vector<point>& P) {
    int n = (int)P.size();
    // a point/sz=2 or a line/sz=3 is not convex
    if (n <= 3) return false;
    bool firstTurn = ccw(P[0], P[1], P[2]);
    for (int i = 1; i < n - 1; ++i)
        if (ccw(P[i], P[i + 1], P[(i + 2) == n ? 1 : i + 2]) !=
              firstTurn)
            return false;  // different -> concave
    return true;           // otherwise -> convex
}

// returns 1/0/-1 if point p is inside/on (vertex/edge)/outside
     of
// either convex/concave polygon P
int insidePolygon(point pt, const vector<point>& P) {
    int n = (int)P.size();
    if (n <= 3) return -1;  // avoid point or line
    bool on_polygon = false;
    for (int i = 0; i < n - 1; ++i)
        if (fabs(dist(P[i], pt) + dist(pt, P[i + 1]) - dist(P[i
             ], P[i + 1])) < EPS)
            on_polygon = true;
    if (on_polygon) return 0;
    double sum = 0.0;
    for (int i = 0; i < n - 1; ++i) {
        if (ccw(pt, P[i], P[i + 1]))
            sum += angle(P[i], pt, P[i + 1]);
        else
            sum -= angle(P[i], pt, P[i + 1]);
    }
    return fabs(sum) > M_PI ? 1 : -1;  // 360d->in, 0d->out
}

// compute the intersection point between line segment p-q and
     line A-B
point lineIntersectSeg(point p, point q, point A, point B) {
    double a = B.y - A.y, b = A.x - B.x, c = B.x * A.y - A.x *
         B.y;
    double u = fabs(a * p.x + b * p.y + c);
    double v = fabs(a * q.x + b * q.y + c);
    return point((p.x * v + q.x * u) / (u + v), (p.y * v + q.y
         * u) / (u + v));
}

// cuts polygon Q along the line formed by point A->point B (
     order matters)
// (note: the last point must be the same as the first point)
vector<point> cutPolygon(point A, point B, const vector<point>&
     Q) {
    vector<point> P;
    for (int i = 0; i < (int)Q.size(); ++i) {
        double left1 = cross(toVec(A, B), toVec(A, Q[i])),
             left2 = 0;
        if (i != (int)Q.size() - 1) left2 = cross(toVec(A, B),
             toVec(A, Q[i + 1]));
        if (left1 > -EPS) P.push_back(Q[i]);
        if (left1 * left2 < -EPS)
            P.push_back(lineIntersectSeg(Q[i], Q[i + 1], A, B))
                ;
    }
    if (!P.empty() && !(P.back() == P.front()))
        P.push_back(P.front());
    return P;
}

vector<point> CH_Andrew(vector<point>& Pts) {
    int n = Pts.size(), k = 0;
    vector<point> H(2 * n);
    sort(Pts.begin(), Pts.end());
    for (int i = 0; i < n; ++i) {
        while ((k >= 2) && !ccw(H[k - 2], H[k - 1], Pts[i])) --
             k;
        H[k++] = Pts[i];
    }
    for (int i = n - 2, t = k + 1; i >= 0; --i) {
        while ((k >= t) && !ccw(H[k - 2], H[k - 1], Pts[i])) --
             k;
        H[k++] = Pts[i];
    }
    H.resize(k);
    return H;
}

// P has no repeated vertix at the back
vii antipodal_pairs(vector<point>& P) {
    int n = P.size();

    if (n == 1) return {};
    if (n == 2) return {{0, 1}};

    vii ans;
    int i0 = n - 1;
    int i = 0;
    int j = i + 1;
    int j0 = j;
    while (tri_area(P[i], P[i + 1], P[j + 1]) - tri_area(P[i],
         P[i + 1], P[j]) > EPS) {
        j = j + 1;
        j0 = j;
    }
    ans.push_back({i, j});
    while (j != i0) {
        i = i + 1;
        ans.push_back({i, j});

        while (tri_area(P[i], P[i + 1], P[j + 1]) - tri_area(P[
             i], P[i + 1], P[j]) > EPS) {
            j = j + 1;
            if ((i != j0 && j != i0))
                ans.push_back({i, j});
            else
                return ans;
        }
        if (fabs(tri_area(P[j], P[i + 1], P[j + 1]) - tri_area(
             P[i], P[i + 1], P[j])) < EPS) {
            if (i != j0 && j != i0)
                ans.push_back({i, j + 1});
            else
                ans.push_back({i + 1, j});
        }
    }
    return ans;
}
// Minkowski Sum
void reorder_polygon(vector<point>& P) {
    P.pop_back();
    size_t pos = 0;
    for (size_t i = 1; i < P.size(); i++) {
        if (P[i].y < P[pos].y || (P[i].y == P[pos].y && P[i].x
             < P[pos].x))
            pos = i;
    }
    rotate(P.begin(), P.begin() + pos, P.end());
    P.push_back(P[0]);
}
vector<point> minkowski(vector<point>& P, vector<point>& Q) {
    reorder_polygon(P);
```

```
    reorder_polygon(Q);

    vector<point> result;
    size_t i = 0, j = 0;
    int n = P.size(), m = Q.size();
    while (i < n - 1 || j < m - 1) {
        result.push_back(P[i] + Q[j]);
        auto crs = cross(P[i + 1] - P[i], Q[j + 1] - Q[j]);
        if (crs >= 0 && i < P.size() - 1)
            ++i;
        if (crs <= 0 && j < Q.size() - 1)
            ++j;
    }
    result.push_back(result[0]);
    return result;
}
// for (auto& p : poly1) {
//     p.x = -p.x;
//     p.y = -p.y;
// }
// vector<point> mink_sum = minkowski(poly1, poly2);
// min_dist b/w polygons is min_dist from (0, 0) to mink_sum
// insidePolygon(mink_sum, point()) >= 0 (polygons intersect)
```

Quadrilateral.h

775850, 8 lines

```
int insideRectangle(int x, int y, int w, int h, int a, int b) {
    if ((x < a) && (a < x + w) && (y < b) && (b < y + h))
        return 1;  // strictly inside
    else if ((x <= a) && (a <= x + w) && (y <= b) && (b <= y +
         h))
        return 0;  // at border
    else
        return -1;  // outside
}
```

Simpson.h

92bf15, 14 lines

```
double simpson(double a, double b) {
    int steps = 1e6;
    double dx = 1.0 * (b - a) / steps;
    double x = a;
    double ans = f(x);
    for (int i = 1; i < steps; i++) {
        x += dx;
        double curr = f(x);
        ans += ((i & 1) ? 4 * curr : 2 * curr);
    }
    x += dx;
    ans += f(x);
    return ans * dx / 3.0;
}
```

Triangle.h

a8b2ea, 60 lines

```
double area(double ab, double bc, double ca) {
    // Heron's formula, split sqrt(a*b) into sqrt(a)*sqrt(b)
    double s = 0.5 * perimeter(ab, bc, ca);
    return sqrt(s) * sqrt(s - ab) * sqrt(s - bc) * sqrt(s - ca)
        ;
}
double rInCircle(double ab, double bc, double ca) {
    return area(ab, bc, ca) / (0.5 * perimeter(ab, bc, ca));
}

// returns 1 if there is an inCircle center, returns 0
    otherwise
int inCircle(point p1, point p2, point p3, point& ctr, double&
    r) {
    r = rInCircle(p1, p2, p3);
    if (fabs(r) < EPS) return 0;

    line l1, l2;
    double ratio = dist(p1, p2) / dist(p1, p3);
    point p = translate(p2, scale(toVec(p2, p3), ratio / (1 +
        ratio)));
    pointsToLine(p1, p, l1);

    ratio = dist(p2, p1) / dist(p2, p3);
    p = translate(p1, scale(toVec(p1, p3), ratio / (1 + ratio))
        );
    pointsToLine(p2, p, l2);
    areIntersect(l1, l2, ctr);
    return 1;
}
double rCircumCircle(double ab, double bc, double ca) {
    return ab * bc * ca / (4.0 * area(ab, bc, ca));
}

// returns 1 if there is a circumCenter center, returns 0
    otherwise
int circumCircle(point p1, point p2, point p3, point& ctr,
    double& r) {
    double a = p2.x - p1.x, b = p2.y - p1.y;
    double c = p3.x - p1.x, d = p3.y - p1.y;
    double e = a * (p1.x + p2.x) + b * (p1.y + p2.y);
    double f = c * (p1.x + p3.x) + d * (p1.y + p3.y);
    double g = 2.0 * (a * (p3.y - p2.y) - b * (p3.x - p2.x));
    if (fabs(g) < EPS) return 0;

    ctr.x = (d * e - b * f) / g;
    ctr.y = (a * f - c * e) / g;
    r = dist(p1, ctr);  // r = distance from center to 1 of the
        3 points
    return 1;
}

// returns true if point d is inside the circumCircle defined
    by a,b,c
int inCircumCircle(point a, point b, point c, point d) {
    return (a.x - d.x) * (b.y - d.y) * ((c.x - d.x) * (c.x - d.
        x) + (c.y - d.y) * (c.y - d.y)) +
                        (a.y - d.y) * ((b.x - d.x) * (b.x - d.x)
                            + (b.y - d.y) * (b.y - d.y)) * (c.
                            x - d.x) +
                        ((a.x - d.x) * (a.x - d.x) + (a.y - d.y)
                            * (a.y - d.y)) * (b.x - d.x) * (c.
                            y - d.y) -
                        ((a.x - d.x) * (a.x - d.x) + (a.y - d.y)
                            * (a.y - d.y)) * (b.y - d.y) * (c.
                            x - d.x) -
                        (a.y - d.y) * (b.x - d.x) * ((c.x - d.x)
                            * (c.x - d.x) + (c.y - d.y) * (c.y
                            - d.y)) -
                        (a.x - d.x) * ((b.x - d.x) * (b.x - d.x)
                            + (b.y - d.y) * (b.y - d.y)) * (c.
                            y - d.y) >
                    0
                ? 1
                : 0;
}

bool canFormTriangle(double a, double b, double c) {
    return (a + b > c) && (a + c > b) && (b + c > a);
}
```

Miscellaneous (7)

Hash.h

396bae, 7 lines

```
template <>
struct std::hash<Key> {
    std::size_t operator()(const Key& k) const {
        using std::hash;
        return ((hash<string>()(k.first) ^ (hash<string>()(k.
            second) << 1)) >> 1) ^ (hash<int>()(k.third) << 1)
            ;
    }
};
```

MaxSuffixQuery.h

6fccc4, 14 lines

```
map<int, ll> m;
void ins(int a, ll b) {
    auto it = m.lower_bound(a);
    if (it != end(m) && it->second >= b) return;
    it = m.insert(it, {a, b});
    it->second = b;
    while (it != begin(m) && prev(it)->second <= b) {
        m.erase(prev(it));
    }
}
ll query(int x) {
    auto it = m.lower_bound(x);
    return it == end(m) ? 0 : it->second;
}
```

OrderedSet.h

<ext/pb_ds/assoc_container.hpp>, <ext/pb_ds/tree_policy.hpp> 0a779f, 3 lines

```
using namespace __gnu_pbds;
template <class T>
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
    tree_order_statistics_node_update>;
```

TernarySearch.h

1b0993, 17 lines

```
// min of a unimodal function f
for (int i = 0; i < 50; ++i) {
    double delta = (r - l) / 3.0;
    double m1 = l + delta;
    double m2 = r - delta;
    (f(m1) > f(m2)) ? l = m1 : r = m2;
}

// discrete, max
while (l + 3 < r) {
    int m1 = (2 * l + r) / 3;
    int m2 = (l + 2 * r) / 3;
    f(m1) <= f(m2) ? l = m1 : r = m2;
}
ll mx = f(l++);
while (l <= r) chmax(mx, f(l++));
return mx;
```

Techniques (A)

techniques.txt 159 lines

```
Recursion
Divide and conquer
  Finding interesting points in N log N
Algorithm analysis
  Master theorem
  Amortized time complexity
Greedy algorithm
  Scheduling
  Max contiguous subvector sum
  Invariants
  Huffman encoding
Graph theory
  Dynamic graphs (extra book-keeping)
  Breadth first search
  Depth first search
  * Normal trees / DFS trees
  Dijkstra's algorithm
  MST: Prim's algorithm
  Bellman-Ford
  Konig's theorem and vertex cover
  Min-cost max flow
  Lovasz toggle
  Matrix tree theorem
  Maximal matching, general graphs
  Hopcroft-Karp
  Hall's marriage theorem
  Graphical sequences
  Floyd-Warshall
  Euler cycles
  Flow networks
  * Augmenting paths
  * Edmonds-Karp
  Bipartite matching
  Min. path cover
  Topological sorting
  Strongly connected components
  2-SAT
  Cut vertices, cut-edges and biconnected components
  Edge coloring
  * Trees
  Vertex coloring
  * Bipartite graphs (=> trees)
  * 3^n (special case of set cover)
  Diameter and centroid
  K'th shortest path
  Shortest cycle
Dynamic programming
  Knapsack
  Coin change
  Longest common subsequence
  Longest increasing subsequence
  Number of paths in a dag
  Shortest path in a dag
  Dynprog over intervals
  Dynprog over subsets
  Dynprog over probabilities
  Dynprog over trees
  3^n set cover
  Divide and conquer
  Knuth optimization
  Convex hull optimizations
  RMQ (sparse table a.k.a 2^k-jumps)
  Bitonic cycle
  Log partitioning (loop over most restricted)
Combinatorics
  Computation of binomial coefficients
  Pigeon-hole principle
  Inclusion/exclusion
  Catalan number
  Pick's theorem
Number theory
  Integer parts
  Divisibility
  Euclidean algorithm
  Modular arithmetic
  * Modular multiplication
  * Modular inverses
  * Modular exponentiation by squaring
  Chinese remainder theorem
  Fermat's little theorem
  Euler's theorem
  Phi function
  Frobenius number
  Quadratic reciprocity
  Pollard-Rho
  Miller-Rabin
  Hensel lifting
  Vieta root jumping
Game theory
  Combinatorial games
  Game trees
  Mini-max
  Nim
  Games on graphs
  Games on graphs with loops
  Grundy numbers
  Bipartite games without repetition
  General games without repetition
  Alpha-beta pruning
Probability theory
Optimization
  Binary search
  Ternary search
  Unimodality and convex functions
  Binary search on derivative
Numerical methods
  Numeric integration
  Newton's method
  Root-finding with binary/ternary search
  Golden section search
Matrices
  Gaussian elimination
  Exponentiation by squaring
Sorting
  Radix sort
Geometry
  Coordinates and vectors
  * Cross product
  * Scalar product
  Convex hull
  Polygon cut
  Closest pair
  Coordinate-compression
  Quadtrees
  KD-trees
  All segment-segment intersection
Sweeping
  Discretization (convert to events and sweep)
  Angle sweeping
  Line sweeping
  Discrete second derivatives
Strings
  Longest common substring
  Palindrome subsequences
  Knuth-Morris-Pratt
  Tries
  Rolling polynomial hashes
  Suffix array
  Suffix tree
  Aho-Corasick
  Manacher's algorithm
  Letter position lists
Combinatorial search
  Meet in the middle
  Brute-force with pruning
  Best-first (A*)
  Bidirectional search
  Iterative deepening DFS / A*
Data structures
  LCA (2^k-jumps in trees in general)
  Pull/push-technique on trees
  Heavy-light decomposition
  Centroid decomposition
  Lazy propagation
  Self-balancing trees
  Convex hull trick (wcipeg.com/wiki/Convex_hull_trick)
  Monotone queues / monotone stacks / sliding queues
  Sliding queue using 2 stacks
  Persistent segment tree
```