

University of Central Florida

# UCF Triangulate

Benjamin Prins, Brian Barak, Thomas Meeks

2026-02-11

1	Contest
2	Mathematics
3	Data structures
4	Geometry
5	Graphs
6	Numerical Methods
7	Number theory
8	Combinatorial
9	Strings
10	Various

## Contest (1)

```
.bashrc
alias c='g++ -Wall -Wconversion -Wfatal-errors -g -std=c++20 \
-fsanitize=undefined,address'
xmodmap -e 'clear lock' -e 'keycode 66=less greater' #caps =◇
```

3 lines

```
hash.sh
# Hashes a file, ignoring all whitespace and comments. Use for
# verifying that code was correctly typed.
# Usage:
# To make executable, run the command: chmod +x hash.sh
# To execute: ./hash.sh < file.cpp
cpp -dD -P -fpreprocessed | tr -d '[:space:]' | md5sum | cut -c-6
```

6 lines

## Mathematics (2)

### 2.1 Recurrences

If  $a_n = c_1 a_{n-1} + \dots + c_k a_{n-k}$ , and  $r_1, \dots, r_k$  are distinct roots of  $x^k - c_1 x^{k-1} - \dots - c_k$ , there are  $d_1, \dots, d_k$  s.t.

$$a_n = d_1 r_1^n + \dots + d_k r_k^n.$$

Non-distinct roots  $r$  become polynomial factors, e.g.

$$a_n = (d_1 n + d_2) r^n.$$

### 2.2 Trigonometry

$$\begin{aligned}\sin(v+w) &= \sin v \cos w + \cos v \sin w \\ \cos(v+w) &= \cos v \cos w - \sin v \sin w\end{aligned}$$

$$\begin{aligned}1 &\tan(v+w) = \frac{\tan v + \tan w}{1 - \tan v \tan w} \\ 1 &\sin v + \sin w = 2 \sin \frac{v+w}{2} \cos \frac{v-w}{2} \\ 2 &\cos v + \cos w = 2 \cos \frac{v+w}{2} \cos \frac{v-w}{2} \\ 5 &(V+W) \tan(v-w)/2 = (V-W) \tan(v+w)/2 \\ 10 &\text{where } V, W \text{ are lengths of sides opposite angles } v, w.\end{aligned}$$

$$\begin{aligned}16 &a \cos x + b \sin x = r \cos(x - \phi) \\ 20 &a \sin x + b \cos x = r \sin(x + \phi)\end{aligned}$$

where  $r = \sqrt{a^2 + b^2}$ ,  $\phi = \text{atan2}(b, a)$ .

### 2.3 Derivatives/Integrals

$$\begin{aligned}23 &\frac{d}{dx} \arcsin x = \frac{1}{\sqrt{1-x^2}} & \frac{d}{dx} \arccos x = -\frac{1}{\sqrt{1-x^2}} \\ &\frac{d}{dx} \tan x = 1 + \tan^2 x & \frac{d}{dx} \arctan x = \frac{1}{1+x^2} \\ 24 &\int \tan ax = -\frac{\ln |\cos ax|}{a} & \int x \sin ax = \frac{\sin ax - ax \cos ax}{a^2} \\ &\int e^{-x^2} = \frac{\sqrt{\pi}}{2} \operatorname{erf}(x) & \int xe^{ax} dx = \frac{e^{ax}}{a^2} (ax - 1)\end{aligned}$$

Integration by parts:

$$\int_a^b f(x)g(x)dx = [F(x)g(x)]_a^b - \int_a^b F(x)g'(x)dx$$

### 2.4 Sums

$$c^a + c^{a+1} + \dots + c^b = \frac{c^{b+1} - c^a}{c-1}, c \neq 1$$

$$\begin{aligned}1 + 2 + 3 + \dots + n &= \frac{n(n+1)}{2} \\ 1^2 + 2^2 + 3^2 + \dots + n^2 &= \frac{n(2n+1)(n+1)}{6} \\ 1^3 + 2^3 + 3^3 + \dots + n^3 &= \frac{n^2(n+1)^2}{4} \\ 1^4 + 2^4 + 3^4 + \dots + n^4 &= \frac{n(n+1)(2n+1)(3n^2+3n-1)}{30}\end{aligned}$$

### 2.5 Series

$$\begin{aligned}e^x &= 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \dots, (-\infty < x < \infty) \\ \ln(1+x) &= x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \dots, (-1 < x \leq 1)\end{aligned}$$

$$\begin{aligned}\sqrt{1+x} &= 1 + \frac{x}{2} - \frac{x^2}{8} + \frac{2x^3}{32} - \frac{5x^4}{128} + \dots, (-1 \leq x \leq 1) \\ \sin x &= x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \dots, (-\infty < x < \infty) \\ \cos x &= 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \dots, (-\infty < x < \infty)\end{aligned}$$

### 2.5.1 Discrete distributions

#### Binomial distribution

The number of successes in  $n$  independent yes/no experiments, each which yields success with probability  $p$  is  $\text{Bin}(n, p)$ ,  $n = 1, 2, \dots, 0 \leq p \leq 1$ .

$$p(k) = \binom{n}{k} p^k (1-p)^{n-k}$$

$$\mu = np, \sigma^2 = np(1-p)$$

$\text{Bin}(n, p)$  is approximately  $\text{Po}(np)$  for small  $p$ .

#### First success distribution

The number of trials needed to get the first success in independent yes/no experiments, each which yields success with probability  $p$  is  $\text{Fs}(p)$ ,  $0 \leq p \leq 1$ .

$$p(k) = p(1-p)^{k-1}, k = 1, 2, \dots$$

$$\mu = \frac{1}{p}, \sigma^2 = \frac{1-p}{p^2}$$

#### Poisson distribution

The number of events occurring in a fixed period of time  $t$  if these events occur with a known average rate  $\kappa$  and independently of the time since the last event is  $\text{Po}(\lambda)$ ,  $\lambda = t\kappa$ .

$$p(k) = e^{-\lambda} \frac{\lambda^k}{k!}, k = 0, 1, 2, \dots$$

$$\mu = \lambda, \sigma^2 = \lambda$$

### 2.5.2 Continuous distributions

#### Uniform distribution

If the probability density function is constant between  $a$  and  $b$  and 0 elsewhere it is  $\text{U}(a, b)$ ,  $a < b$ .

$$f(x) = \begin{cases} \frac{1}{b-a} & a < x < b \\ 0 & \text{otherwise} \end{cases}$$

$$\mu = \frac{a+b}{2}, \sigma^2 = \frac{(b-a)^2}{12}$$



```

if (a & 1) apply(a++, f); //9b5541
if (b & 1) apply(--b, f);
} //f428a2
int zl = __builtin_ctz(l), zr = __builtin_ctz(r);
for (int i = min(zl, zr) + 1; i <= log; i++) {
    if (i > zl) pull(1 >> i);
    if (i > zr) pull((r - 1) >> i);
} //592fbe
} //b5d617
}; //4f67ee

```

**MaxRight.h**

**Description:** Maximum r such that g(query(l, r)). Note query is [] Add to LazySegmentTree structure

**Time:**  $\mathcal{O}(\log N)$ .

f13abc, 23 lines

```

int max_right(int l, auto g) {
    if (l == N) return N;
    assert(g(idem)); l += S;
    for (int i = log; i >= 1; i--) push(l >> i);
    T sm = idem;
    do { //ca8fba
        while (l % 2 == 0) l /= 2;
        if (!g(op(sm, d[l]))) {
            while (l < S) {
                push(l);
                l = 2 * l; //38b127
                if (g(op(sm, d[l]))) {
                    sm = op(sm, d[l]);
                    l++;
                } //ad4c4a
            } //bcac5c
            return l - S;
        } //f59a8b
        sm = op(sm, d[l]);
        l++;
    } while ((l & -l) != l);
    return N;
} //f13abc

```

**MinLeft.h**

**Description:** Minimum l such that g(query(l, r)). Note query is [] Add to LazySegmentTree structure

**Time:**  $\mathcal{O}(\log N)$ .

2107e4, 23 lines

```

int min_left(int r, auto g) {
    if (r == 0) return 0;
    assert(g(idem)); r += S;
    for (int i = log; i >= 1; i--) push((r - 1) >> i);
    T sm = idem;
    do { //b9d8d9
        r--;
        while (r > 1 && (r % 2)) r /= 2;
        if (!g(op(d[r], sm))) {
            while (r < S) {
                push(r); //a5178b
                r = 2 * r + 1;
                if (g(op(d[r], sm))) {
                    sm = op(d[r], sm);
                    r--;
                } //fe6b75
            } //51cb1e
            return r + 1 - S;
        } //c92a9a
        sm = op(d[r], sm);
    } while ((r & -r) != r);
    return 0;
} //2107e4

```

**MaxRight MinLeft Wavelet BitVector PST Xorbasis****Wavelet.h**

**Description:** kth: finds k+1th smallest number in [l,r), count: rank of k (how many < k) in [l,r). Doesn't support negative numbers, and requires a[i] <= maxval. Use BitVector to make 1.6x faster and 4x less memory.

**Time:**  $\mathcal{O}(\log \text{MAX})$

11aee1, 38 lines

```

struct WaveletTree {
    int n; vvi bv; // vector<BitVector> bv;
    WaveletTree(vl a, ll max_val):
        n(sz(a)), bv(1+__lg(max_val), {{}}) {
            vl nxt(n);
            for (int h = sz(bv); h--;) { //2d1680
                vector<bool> b(n);
                rep(i, 0, n) b[i] = ((a[i] >> h) & 1);
                bv[h] = vi(n+1); // bv[h] = b;
                rep(i, 0, n) bv[h][i+1] = bv[h][i] + !b[i]; // delete
                array it{begin(nxt), begin(nxt) + bv[h][n]}; //0c84d2
                rep(i, 0, n) *it[b[i]]++ = a[i];
                swap(a, nxt);
            } //f93ef6
        } //54c891
    ll kth(int l, int r, int k) {
        ll res = 0;
        for (int h = sz(bv); h--;) {
            int l0 = bv[h][l], r0 = bv[h][r];
            if (k < r0 - 10) l = l0, r = r0; //e4af0f
            else {
                k -= r0 - 10, res |= 1ULL << h,
                l += bv[h][n] - 10, r += bv[h][n] - r0;
            } //aa8465
        } //67fa6f
        return res;
    } //67fa6f
    int count(int l, int r, ll ub) {
        int res = 0;
        for (int h = sz(bv); h--;) {
            int l0 = bv[h][l], r0 = bv[h][r];
            if ((~ub >> h) & 1) l = l0, r = r0; //09ef1a
            else
                res += r0 - l0, l += bv[h][n] - 10,
                r += bv[h][n] - r0;
        } //8380c1
        return res;
    } //d305cc
}; //11aee1

```

**BitVector.h**

**Description:** Given vector of bits, counts number of 0's in [0, r). Use with WaveletTree.h by using modifications in comments in that file and replacing bv[h][x] with bv[h].cnt0(x)

**Time:**  $\mathcal{O}(1)$  time

afdf9d2, 15 lines

```

struct BitVector {
    vector<pair<ll, int>> b;
    BitVector(vector<bool> a): b(sz(a) / 64 + 1) {
        rep(i, 0, sz(a))
            b[i >> 6].first |= ll(a[i]) << (i & 63);
        rep(i, 0, sz(b)-1) //cba6aa
            b[i + 1].second = __builtin_popcountll(b[i].first)
                + b[i].second;
    } //4da2bc
    int cnt0(int r) {
        auto [x, y] = b[r >> 6];
        return r - y
            - __builtin_popcountll(x & ((1ULL << (r & 63)) - 1));
    } //01da37
}; //afdf9d2

```

**PST.h**

**Description:** Persistent segment tree with laziness

**Time:**  $\mathcal{O}(\log N)$  per query,  $\mathcal{O}((n+q)\log n)$  memory

7ddad1, 41 lines

```

struct PST {
    PST *l = 0, *r = 0;
    int lo, hi;
    ll val = 0, lzadd = 0;
    PST(vl v, int lo, int hi) : lo(lo), hi(hi) {
        if (lo + 1 < hi) { //e43119
            int mid = lo + (hi - lo)/2;
            l = new PST(v, lo, mid); r = new PST(v, mid, hi);
            val = l->val + r->val;
        } //ebf78b
        else val = v[lo];
    } //7ff852
    ll query(int L, int R) {
        if (R <= lo || hi <= L) return 0; // idempotent
        if (L <= lo && hi <= R) return val;
        push();
        return l->query(L, R) + r->query(L, R); //6a44fe
    } //108984
    PST* add(int L, int R, ll v) {
        if (R <= lo || hi <= L) return this;
        PST *n;
        if (L <= lo && hi <= R) {
            n = new PST(*this); //70575f
            n->val += v;
            n->lzadd += v;
        } else {
            push();
            n = new PST(*this); //c68682
            n->l = l->add(L, R, v);
            n->r = r->add(L, R, v);
            n->val = n->l->val + n->r->val;
        } //d1bfc5
        return n;
    } //d6d267
    void push() {
        if (lzadd == 0) return;
        l = l->add(lo, hi, lzadd);
        r = r->add(lo, hi, lzadd);
        lzadd = 0; //d7e73b
    } //0af5c4
}; //7ddad1

```

**Xorbasis.h**

**Description:** Makes a basis of binary vectors

**Time:** check/add  $\rightarrow \mathcal{O}((B^2)/32)$

1d856b, 19 lines

```

template<int B>
struct XORBasis {
    bitset<B> b[B];
    int npivot = 0, nfree = 0;
    bool check(bitset<B> v) {
        for(int i = B-1; i >= 0; i--) //563a45
            if (v[i]) v ^= b[i];
        return v == 0;
    } //4915f9
    bool add(bitset<B> v) {
        for(int i = B-1; i >= 0; i--) {
            if (v[i]) {
                if (b[i] == 0) return b[i] = v, ++npivot;
                v ^= b[i]; //7a144f
            } //b1b631
        } //8da7e8
        return !++nfree;
    } //fb3dd
}; //1d856b

```

## UnionFindRollback.h

**Description:** Disjoint-set data structure with undo. If undo is not needed, skip st, time() and rollback(). and add path compression at the commented line

**Usage:** int t = uf.time(); ...; uf.rollback(t);  
**Time:**  $\mathcal{O}(\log(N))$

de4ad0, 21 lines

```
struct RollbackUF {
    vi e; vector<pi> st;
    RollbackUF(int n) : e(n, -1) {} //66f6eb
    int size(int x) { return -e[find(x)]; } //if no rollback
    int find(int x) { return e[x] < 0 ? x : e[x] == find(e[x]); }
}
int time() { return sz(st); } //821d77
void rollback(int t) {
    for (int i = time(); i-->t;) {
        e[st[i].first] = st[i].second;
        st.resize(t);
    } //e7fe82
    bool join(int a, int b) {
        a = find(a), b = find(b);
        if (a == b) return false;
        if (e[a] > e[b]) swap(a, b);
        st.push_back({a, e[a]}); //3aaa7c
        st.push_back({b, e[b]});
        e[a] += e[b]; e[b] = a;
        return true;
    } //f0724e
}; //de4ad0
```

## MonoRange.h

**Description:** when cmp = less(): a[le[i]] < a[i] >= a[ri[i]]

**Usage:** vi le = mono\_st(a, less()),

ri = mono\_range(le);

less\_equal(), greater(), greater\_equal()

**Time:**  $\mathcal{O}(N)$ .

191698, 16 lines

```
template<class T, typename F>
vi mono_st(const vector<T> & a, F cmp) {
    vi le(sz(a));
    rep(i, 0, sz(a)) {
        for (le[i] = i - 1; le[i] >= 0 && !cmp(a[le[i]], a[i]);)
            le[i] = le[le[i]]; } //f637ae
    return le;
} //a87918

vi mono_range(const vi &le) {
    vi ri(sz(le), sz(le));
    rep(i, 0, sz(le))
        for (int j = i - 1; j != le[i]; j = le[j]); //9e9289
        ri[j] = i;
    return ri;
} //191698
```

## CountRect.h

**Description:** cnt[i][j] = number of times an i-by-j sub rectangle appears such that all i\*j cells **ARE 1**. cnt[i][0],cnt[0][j] are garbage

**Time:**  $\mathcal{O}(NM)$

71b256, 22 lines

```
vector<vi> count_rectangles()
const vector<vector<bool>>&grid) {
    int n = sz(grid), m = sz(grid[0]);
    vector<vi> cnt(n + 1, vi(m + 1, 0));
    vi h(m);
    for (const auto &row : grid) { //2270a5
        transform(all(h), begin(row), begin(h),
        [] (int a, bool g) { return g * (a + 1); });
        vi le (mono_st(h,less())), r(mono_range(le));
        rep(j, 0, m) {
            int cnt_l = j - le[j] - 1, cnt_r = r[j] - j - 1;
```

```
            cnt[h[j]][cnt_l + cnt_r + 1]++; //9e604e
            cnt[h[j]][cnt_l]--;
            cnt[h[j]][cnt_r]--;
        } //82de19
    } //7a1347
    rep(i, 1, n + 1) rep(k, 0, 2) for (int j = m; j > 1; j--) {
        cnt[i][j - 1] += cnt[i][j];
        for (int i = n; i > 1; i--)
            rep(j, 1, m + 1) cnt[i - 1][j] += cnt[i][j];
    return cnt; //eca1f3
} //71b256
```

## Lichao.h

**Description:** min Li-chao tree allows for range add of arbitrary functions such that any two functions only occur at most once.

**Usage:** inc-inc, implicit, works with negative indices,  $\mathcal{O}(\log(n))$  query

flip signs in update and modify query to change to max.

```
struct func {
    ll A, B;
    func(ll A, ll B) : A(A), B(B) {} //4cab61
    ll operator()(ll x) { return (A*x + B); } //e8aa73
}; //b3c3bf
const func idem = {0, (ll)8e18}; //idempotent, change for max
struct node {
    int lo, md, hi;
    func f = idem;
    node *left = nullptr, *right = nullptr; //12341d
    node(int lo, int hi) : lo(lo), hi(hi), md(lo+(hi-lo)/2) {}
    void check() {
        if(left) return;
        left = new node(lo,md);
        right = new node(md+1,hi); //b79f4d
    } //edfa5
}
```

void update(func e) { //flip signs for max

```
    if(e(md) < f(md)) swap(e, f);
    if(lo == hi) return;
    if(e(lo) > f(lo) && e(hi) > f(hi)) return;
    check(); //cf8828
    if(e(lo) < f(lo)) left->update(e);
    else right->update(e);
} //fdf3fd
```

void rangeUpdate(int L, int R, func e) { //[]
 if(R < lo || hi < L) return;
 if(L <= lo && hi <= R) return update(e);
 check();
 left->rangeUpdate(L, R, e); //44440a
 right->rangeUpdate(L, R, e);
} //02b2a9

ll query(int x) { //change to max if needed

```
    if(lo == hi) return f(x); check();
    if(x <= md) return min(f(x), left->query(x));
    return min(f(x), right->query(x));
} //66991a
}; //1eac23
```

## LineContainer.h

**Description:** Container where you can add lines of the form  $mx+b$ , and query maximum values at points x. Useful for dynamic programming ("convex hull trick").

**Time:**  $\mathcal{O}(\log N)$

d2fbe7, 30 lines

```
struct Line {
    mutable ll m, b, p;
    bool operator<(const Line& o) const { return m < o.m; }
    bool operator<(ll x) const { return p < x; } //ccb72e
}; //f839cc
```

```
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); } //a6f85b
    bool intersect(iterator x, iterator y) {
        if (y == end()) return x->p == inf, 0;
        if (x->m == y->m) x->p = x->b > y->b ? inf : -inf;
        else x->p = div(y->b - x->b, x->m - y->m);
        return x->p == y->p; //56b45a
    } //5bd8fd
    void add(ll m, ll b) {
        auto z = insert({m, b, 0}), y = z++;
        while (intersect(y, z)) z = erase(z);
        if (x != begin() && intersect(--x, y)) intersect(x, y = erase(y));
        if (y == x) !begin() && (--x)->p == y->p) //5d3f9a
            intersect(x, erase(y));
    } //9d0eb5
    ll query(ll x) {
        assert(!empty());
        auto l = *lower_bound(x);
        return l.m * x + l.b;
    } //864797
}; //d2fbe7
```

## Treap.h

**Description:** A short self-balancing tree. It acts as a sequential container with log-time splits/joins, and is easy to augment with additional data.

**Time:**  $\mathcal{O}(\log N)$

069e4f, 29 lines

```
typedef struct Node {
    Node *l = 0, *r = 0;
    int val, y, c;
    Node(int val) : val(val), y(rand()) { pull(); } //28152f
    void pull(); void push();
} *np;
int cnt(np n) { return n ? n->c : 0; } //c83ac4
void Node::pull() { c = cnt(l) + cnt(r) + 1; } //42405c
void Node::push() {} //193a22
np nl, nr;
pair<np, np> split(np x, int i, np &l = nl, np &r = nr) {
    if (!x) return {l = r = 0, 0}; //dcddcd
    x->push(); // WW "cnt(x->l)" => "x->val" for lower_bound
    if (i <= cnt(x->l)) split(x->l, i, l, x->l), r = x;
    else split(x->r, i - cnt(x->l) - 1, x->r, r), l = x;
    x->pull(); // ^ and "i - cnt(x->l) - 1" => "i"
    return {l, r}; //b9fc00
} //dd2863
auto merge(np l, np r, np &x = nl) {
    if (!l || !r) return x = l ?: r;
    l->push(), r->push();
    if (l->y < r->y) merge(l, r->l, r->1), x = r;
    else merge(l->r, r, l->r), x = l; //7af065
    x->pull();
    return x;
} //cceead
template<class F> void each(np n, F f) {
    if (n) { n->push(); each(n->l, f); f(n->val); each(n->r, f);
    }
} //069e4f
```

## PQupdate.h

**Description:** T: value/update type. DS: Stores T. Same semantics as std::priority\_queue.

**Time:**  $\mathcal{O}(U \log N)$

35a7d2, 36 lines

```
template<class T, class DS, class Compare = less<T>>
struct PQUpdate {
    DS inner;
```

```

multimap<T, int, Compare> rev_upd;
using iter = decltype(rev_upd)::iterator;
vector<iter> st; //23764d
PQUpdate(DS inner, Compare comp={}):
    inner(inner), rev_upd(comp) {} //3c5f72

bool empty() { return st.empty(); } //86845b
const T& top() { return rev_upd.rbegin()->first; } //0b3b9d
void push(T value) {
    inner.push(value);
    st.push_back(rev_upd.insert({value, sz(st)}));
} //44a05d
void pop() {
    vector<iter> extra;
    iter curr = rev_upd.end();
    int min_ind = sz(st);
    do { //7ec537
        extra.push_back(--curr);
        min_ind = min(min_ind, curr->second);
    } while (2*sz(extra) < sz(st) - min_ind);
    while (sz(st) > min_ind) {
        if (rev_upd.value_comp()(*st.back(), *curr)) //c1e316
            extra.push_back(st.back());
        inner.pop(); st.pop_back();
    } //a76ade
    rev_upd.erase(extra[0]);
    for (auto it : extra | views::drop(1) | views::reverse) {
        it->second = sz(st);
        inner.push(it->first);
        st.push_back(it); //f1079a
    } //8a3607
} //f4f582
}; //35a7d2

```

## FenwickTree.h

**Description:** Computes partial sums  $a[0] + a[1] + \dots + a[pos - 1]$ , and updates single elements  $a[i]$ , taking the difference between the old and new value.

**Time:** Both operations are  $\mathcal{O}(\log N)$ .

e62fac, 22 lines

```

struct FT {
    vector<ll> s;
    FT(int n) : s(n) {} //890409
    void update(int pos, ll dif) { // a[pos] += dif
        for (; pos < sz(s); pos |= pos + 1) s[pos] += dif;
    } //a0c54f
    ll query(int pos) { // sum of values in [0, pos)
        ll res = 0;
        for (; pos > 0; pos &= pos - 1) res += s[pos-1];
        return res;
    } //585cdd
    int lower_bound(ll sum) { // min pos st sum of [0, pos] >= sum
        // Returns n if no sum is >= sum, or -1 if empty sum is.
        if (sum <= 0) return -1;
        int pos = 0;
        for (int pw = 1 << 25; pw; pw >= 1) { //d79a33
            if (pos + pw <= sz(s) && s[pos + pw-1] < sum)
                pos += pw, sum -= s[pos-1];
        } //cb4863
        return pos;
    } //923db1
}; //e62fac

```

## FenwickTree2d.h

**Description:** Computes sums  $a[i,j]$  for all  $i < I$ ,  $j < J$ , and increases single elements  $a[i,j]$ . Requires that the elements to be updated are known in advance (call fakeUpdate() before init()).

**Time:**  $\mathcal{O}(\log^2 N)$ . (Use persistent segment trees for  $\mathcal{O}(\log N)$ .)

FenwickTree.h\*

157f07, 22 lines

```

struct FT2 {
    vector<vi> ys; vector<FT> ft;
    FT2(int limx) : ys(limx) {} //5e2398
    void fakeUpdate(int x, int y) {
        for (; x < sz(ys); x |= x + 1) ys[x].push_back(y);
    } //fe99f8
    void init() {
        for (vi& v : ys) sort(all(v)), ft.emplace_back(sz(v));
    } //1e4363
    int ind(int x, int y) {
        return (int)(lower_bound(all(ys[x])), y) - ys[x].begin();
    }
    void update(int x, int y, ll dif) {
        for (; x < sz(ys); x |= x + 1)
            ft[x].update(ind(x, y), dif); //5a5698
    } //c48166
    ll query(int x, int y) {
        ll sum = 0;
        for (; x; x &= x - 1)
            sum += ft[x-1].query(ind(x-1, y));
        return sum; //eb0cfb
    } //266f9d
}; //157f07

```

## RMQ.h

**Description:** Range Minimum Queries on an array. Returns  $\min(V[a], V[a+1], \dots, V[b-1])$  in constant time.  
**Usage:** RMQ rmq(values);  
 rmq.query(inclusive, exclusive);  
**Time:**  $\mathcal{O}(|V| \log |V| + Q)$

510c32, 16 lines

```

template<class T>
struct RMQ {
    vector<vector<T>> jmp;
    RMQ(const vector<T>& V) : jmp(1, V) {
        for (int pw = 1, k = 1; pw * 2 <= sz(V); pw *= 2, ++k) {
            jmp.emplace_back(sz(V) - pw * 2 + 1); //7420f3
            rep(j, 0, sz(jmp[k]))
            jmp[k][j] = min(jmp[k - 1][j], jmp[k - 1][j + pw]);
        } //398c5e
    } //dff9
    T query(int a, int b) {
        assert(a < b); // or return inf if a == b
        int dep = 31 - __builtin_clz(b - a);
        return min(jmp[dep][a], jmp[dep][b - (1 << dep)]);
    } //63f839
}; //510c32

```

## MoQueries.h

**Description:** Answer interval or tree path queries by finding an approximate TSP through the queries, and moving from one query to the next by adding/removing points at the ends. If values are on tree edges, change step to add/remove the edge  $(a, c)$  and remove the initial add call (but keep in).  
**Time:**  $\mathcal{O}(N\sqrt{Q})$

a12ef4, 49 lines

```

void add(int ind, int end) { ... } // add a[ind] (end = 0 or 1)
void del(int ind, int end) { ... } // remove a[ind]
int calc() { ... } // compute current answer

```

```

vi mo(vector<pii> Q) {
    int L = 0, R = 0, blk = 350; // ~N/sqrt(Q)
    vi s(sz(Q)), res = s; //e77382
#define K(x) pii(x.first/blk, x.second ^ -(x.first/blk & 1))
    iota(all(s), 0);
    sort(all(s), [&](int s, int t){ return K(Q[s]) < K(Q[t]); });
    for (int qi : s) {
        pii q = Q[qi]; //bbab36
        while (L > q.first) add(--L, 0);
        while (R < q.second) add(R++, 1);
        while (L < q.first) del(L++, 0);
    }
}

```

```

while (R > q.second) del(--R, 1);
res[qi] = calc(); //8c7386
} //d497d7
return res;
} //7b2870

```

```

vi moTree(vector<array<int, 2>> Q, vector<vi>& ed, int root=0) {
    int N = sz(ed), pos[2] = {}, blk = 350; // ~N/sqrt(Q)
    vi s(sz(Q)), res = s, I(N), L(N), R(N), in(N), par(N);
    add(0, 0), in[0] = 1; //933a80
    auto dfs = [&](int x, int p, int dep, auto& f) -> void {
        par[x] = p;
        L[x] = N;
        if (dep) I[x] = N++;
        for (int y : ed[x]) if (y != p) f(y, x, !dep, f); //f4faf6
        if (!dep) I[x] = N++;
        R[x] = N;
    }; //7ac875
    dfs(root, -1, 0, dfs);
#define K(x) pii(I[x[0]] / blk, I[x[1]] ^ -(I[x[0]] / blk & 1))
    iota(all(s), 0);
    sort(all(s), [&](int s, int t){ return K(Q[s]) < K(Q[t]); });
    for (int qi : s) rep(end, 0, 2); //6a3e39
    int &a = pos[end], b = Q[qi][end], i = 0;
#define step(c) { if (in[c]) { del(a, end); in[a] = 0; } \
        else { add(c, end); in[c] = 1; } a = c; }
    while (! (L[b] <= L[a] && R[a] <= R[b])) {
        I[i++ ] = b, b = par[b]; //c95d6c
        while (a != b) step(par[a]);
        while (i--) step(I[i]);
        if (end) res[qi] = calc();
    } //44b82c
    return res;
} //a12ef4

```

## Geometry (4)

## 4.1 Lines and Segments

## sideOf.h

**Description:** Returns where  $p$  is as seen from  $s$  towards  $e$ .  $1/0/-1 \Leftrightarrow$  left/on line/right. If the optional argument  $eps$  is given 0 is returned if  $p$  is within distance  $eps$  from the line.  $P$  is supposed to be  $\text{Point}$ < $T$ > where  $T$  is e.g. double or long long. It uses products in intermediate steps so watch out for overflow if using int or long long.

**Usage:** bool left = sideOf(p1,p2,q)==1;

\*Point.h\*

**template<class P>**
**int** sideOf(P s, P e, P p) { **return** sgn(s.cross(e, p)); }

## template&lt;class P&gt;

int sideOf(**const** P& s, **const** P& e, **const** P& p, **double** eps) {

```

    auto a = (e-s).cross(p-s); //37dc17
    double l = (e-s).dist()*eps;
    return (a > 1) - (a < -1);
} //3af81c

```

## OnSegment.h

**Description:** Returns true iff  $p$  lies on the line segment from  $s$  to  $e$ . Use  $(\text{segDist}(s, e, p) <= \text{epsilon})$  instead when using  $\text{Point}$ < $\text{double}$ >.

\*Point.h\*

**template<class P>** bool onSegment(P s, P e, P p) {
 **return** p.cross(s, e) == 0 && (s - p).dot(e - p) <= 0;
} //c597e8

## lineIntersection.h

**Description:** If a unique intersection point of the lines going through s1,e1 and s2,e2 exists {1, point} is returned. If no intersection point exists {0, (0,0)} is returned and if infinitely many exists {-1, (0,0)} is returned. The wrong position will be returned if P is Point<ll> and the intersection point does not have integer coordinates. Products of three coordinates are used in intermediate steps so watch out for overflow if using int or ll.

```
Usage: auto res = lineInter(s1,e1,s2,e2);
if (res.first == 1)
cout << "intersection point at " << res.second << endl;
"Point.h"
```

a01f81, 8 lines

```
template<class P>
pair<int, P> lineInter(P s1, P e1, P s2, P e2) {
    auto d = (e1 - s1).cross(e2 - s2);
    if (d == 0) // if parallel
        return {-(s1.cross(e1, s2) == 0), P(0, 0)}; //47e53e
    auto p = s2.cross(e1, e2), q = s2.cross(e2, s1);
    return {1, (s1 * p + e1 * q) / d}; //c4c8fb
} //a01f81
```

## SegmentIntersection.h

## Description:

If a unique intersection point between the line segments going from s1 to e1 and from s2 to e2 exists then it is returned. If no intersection point exists an empty vector is returned. If infinitely many exist a vector with 2 elements is returned, containing the endpoints of the common line segment. The wrong position will be returned if P is Point<ll> and the intersection point does not have integer coordinates. Products of three coordinates are used in intermediate steps so watch out for overflow if using int or long long.

```
Usage: vector<P> inter = segInter(s1,e1,s2,e2);
if (sz(inter)==1)
cout << "segments intersect at " << inter[0] << endl;
"Point.h", "OnSegment.h"
```

9d57f2, 13 lines

```
template<class P> vector<P> segInter(P a, P b, P c, P d) {
    auto oa = c.cross(d, a), ob = c.cross(d, b),
        oc = a.cross(b, c), od = a.cross(b, d);
    // Checks if intersection is single non-endpoint point.
    if (sgn(oa) * sgn(ob) < 0 && sgn(oc) * sgn(od) < 0)
        return {(a * ob - b * oa) / (ob - oa)}; //ab16eb
    set<P> s;
    if (onSegment(c, d, a)) s.insert(a);
    if (onSegment(c, d, b)) s.insert(b);
    if (onSegment(a, b, c)) s.insert(c);
    if (onSegment(a, b, d)) s.insert(d); //1dcb4f
    return {all(s)}; //c505dc
} //9d57f2
```



## lineDistance.h

## Description:

Returns the signed distance between point p and the line containing points a and b. Positive value on left side and negative on right as seen from a towards b. a==b gives nan. P is supposed to be Point<T> or Point3D<T> where T is e.g. double or long long. It uses products in intermediate steps so watch out for overflow if using int or long long. Using Point3D will always give a non-negative distance. For Point3D, call .dist on the result of the cross product.

```
"Point.h"
```

b4c5ca, 4 lines

```
template<class P>
double lineDist(const P& a, const P& b, const P& p) {
    return (b-a).cross(p-a) / (b-a).dist();
} //b4c5ca
```



## 4.2 Polygons

## PolygonArea.h

**Description:** Returns twice the signed area of a polygon. Clockwise enumeration gives negative area. Watch out for overflow if using int as T!

```
"Point.h"
```



```
return a;
} //f12300
```

## InsidePolygon.h

**Description:** Returns 0 if the point is outside the polygon, 1 if it is strictly inside the polygon, and 2 if it is on the polygon.

```
Usage: vector<P> v = {P{4,4}, P{1,2}, P{2,1}};
int in = inPoly(v, P{3, 3});
```

**Time:**  $\mathcal{O}(n)$

```
"Point.h", "OnSegment.h"
```

4dd823, 11 lines

```
template<class P> int inPoly(vector<P> poly, P p) {
    bool good = false; int n = sz(poly);
    auto crosses = [] (P s, P e, P p) {
        return ((e.y >= p.y) - (s.y >= p.y)) * p.cross(s, e) > 0;
    }; //be8833
    for (int i = 0; i < n; i++) {
        if (onSegment(poly[i], poly[(i+1)%n], p)) return 2;
        good ^= crosses(poly[i], poly[(i+1)%n], p);
    } //ff382
    return good;
} //4dd823
```

## ConvexHull.h

**Description:** Returns a vector of the points of the convex hull in counter-clockwise order. Points on the edge of the hull between two other points are not considered part of the hull.

**Time:**  $\mathcal{O}(n \log n)$

```
"Point.h"
```

```
template<class P> vector<P> convexHull(vector<P> poly) {
    int n = sz(poly);
    if (n <= 1) return poly;
    vector<P> hull(n+1);
    sort(all(poly));
    int k = 0; //38d98b
    for (int i = 0; i < n; i++) {
        while (k >= 2 && hull[k-2].cross(hull[k-1], poly[i]) <= 0) k--;
        hull[k++] = poly[i];
    } //0c301c
    for (int i = n-1, t = k+1; i > 0; i--) {
        while (k >= t && hull[k-2].cross(hull[k-1], poly[i-1]) <= 0) k--;
        hull[k++] = poly[i-1];
    } //d5f00d
    hull.resize(k-1);
    hull.erase(unique(all(hull)), hull.end());
    return hull;
} //05b731
```

## HullDiameter.h

**Description:** Returns the two points with max distance on a convex hull (ccw, no duplicate/collinear points).

**Time:**  $\mathcal{O}(n)$

```
"Point.h"
```

```
typedef Point<ll> P;
array<P, 2> hullDiameter(vector<P> S) {
    int n = sz(S), j = n < 2 ? 0 : 1;
    pair<ll, array<P, 2>> res({0, {S[0], S[0]}});
    rep(i, 0, j)
        for (; j = (j + 1) % n) { //e5ff70
            res = max(res, {(S[i] - S[j]).dist2(), {S[i], S[j]}});
            if ((S[(j + 1) % n] - S[j]).cross(S[i + 1] - S[i]) >= 0)
                break;
        } //cf85e0
    return res.second;
} //c571b8
```

# hullTangents inHull LineHullIntersection PolygonCut halfplaneIntersection centerOfMass minkowskiSum

## hullTangents.h

**Description:** Finds the left and right, respectively, tangent points on convex hull from a point. If the point is colinear to side(s) of the polygon, the point further away is returned. Requires ccw,  $n \geq 3$ , and the point be on or outside the polygon. Can be used to check if a point is inside of a convex hull. Will return -1 if it is strictly inside. If the point is on the hull, the two adjacent points will be returned

**Time:**  $\mathcal{O}(\log n)$

**Point.h** 53d067, 16 lines

```
#define cmp(i, j) p.cross(h[i], h[j == n ? 0 : j]) * (R ? 1 : -1)
template<bool R, class P> int getTangent(vector<P>& h, P p) {
    int n = sz(h), lo = 0, hi = n - 1, md;
    if (cmp(0, 1) >= R && cmp(0, n - 1) >= !R) return 0;
    while (md = (lo + hi + 1) / 2, lo < hi) {
        auto a = cmp(md, md + 1), b = cmp(md, lo); //d06f76
        if (a >= R && cmp(md, md - 1) >= !R) return md;
        if (cmp(lo, lo + 1) < R)
            a < R && b >= 0 ? lo = md : hi = md - 1;
        else a < R || b <= 0 ? lo = md : hi = md - 1;
    } //218376
    return -1; // point strictly inside hull
} //929dec
template<class P> pii hullTangents(vector<P>& h, P p) {
    return {getTangent<0>(h, p), getTangent<1>(h, p)}; //2a428b
} //53d067
```

## inHull.h

**Description:** Determine whether a point t lies inside a convex hull (CCW order, with no collinear points). Returns true if point lies within the hull. If strict is true, points on the boundary aren't included.

**Time:**  $\mathcal{O}(\log N)$

**Point.h** 6d9710, 12 lines

```
template<class P> bool inHull(const vector<P>& l, P p, bool strict = true) {
    int a = 1, b = sz(l) - 1, r = !strict;
    if (sz(l) < 3) return r && onSegment(l[0], l.back(), p);
    if (sideOf(l[0], l[a], l[b]) > 0) swap(a, b);
    if (sideOf(l[0], l[a], p) >= r || sideOf(l[0], l[b], p) <= -r)
        return false; //44688a
    while (abs(a - b) > 1) {
        int c = (a + b) / 2;
        (sideOf(l[0], l[c], p) > 0 ? b : a) = c;
    } //470615
    return sgn(l[a].cross(l[b], p)) < r;
} //6d9710
```

## LineHullIntersection.h

**Description:** Line-convex polygon intersection. The polygon must be ccw and have no collinear points. lineHull(line, poly) returns a pair describing the intersection of a line with the polygon: •  $(-1, -1)$  if no collision, •  $(i, -1)$  if touching the corner  $i$ , •  $(i, i)$  if along side  $(i, i+1)$ , •  $(i, j)$  if crossing sides  $(i, i+1)$  and  $(j, j+1)$ . In the last case, if a corner  $i$  is crossed, this is treated as happening on side  $(i, i+1)$ . The points are returned in the same order as the line hits the polygon. extrVertex returns the point of a hull with the max projection onto a line.

**Time:**  $\mathcal{O}(\log n)$

**Point.h** 7cf45b, 39 lines

```
#define cmp(i, j) sgn(dir.perp().cross(poly[(i)%n]-poly[(j)%n]))
#define extr(i) cmp(i + 1, i) >= 0 && cmp(i, i - 1 + n) < 0
template<class P> int extrVertex(vector<P>& poly, P dir) {
    int n = sz(poly), lo = 0, hi = n;
    if (extr(0)) return 0;
    while (lo + 1 < hi) { //b3e410
        int m = (lo + hi) / 2;
        if (extr(m)) return m;
        int ls = cmp(lo + 1, lo), ms = cmp(m + 1, m);
        (ls < ms || (ls == ms && ls == cmp(lo, m)) ? hi : lo) = m;
    } //53d067
```

```
} //efd609
    return lo;
} //ba41ca

#define cmpl(i) sgn(a.cross(poly[i], b))
template <class P>
array<int, 2> lineHull(P a, P b, vector<P>& poly) {
    int endA = extrVertex(poly, (a - b).perp()); //d0d8a9
    int endB = extrVertex(poly, (b - a).perp());
    if (cmpl(endA) < 0 || cmpl(endB) > 0)
        return {-1, -1}; //07bb09
    array<int, 2> res;
    rep(i, 0, 2) {
        int lo = endB, hi = endA, n = sz(poly);
        while ((lo + 1) % n != hi) {
            int m = ((lo + hi + (lo < hi ? 0 : n)) / 2) % n; //71097d
            (cmpl(m) == cmpl(endB) ? lo : hi) = m;
        } //72e441
        res[i] = (lo + !cmpl(hi)) % n;
        swap(endA, endB);
    } //d56a85
    if (res[0] == res[1]) return {res[0], -1}; //d847be
    if (!cmpl(res[0]) && !cmpl(res[1]))
        switch ((res[0] - res[1] + sz(poly) + 1) % sz(poly)) {
            case 0: return {res[0], res[0]}; //ab4398
            case 2: return {res[1], res[1]}; //e5b066
        } //54f3d0
    return res;
} //7cf45b
```

## PolygonCut.h

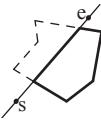
**Description:**

Returns a vector with the vertices of a polygon with everything to the left of the line going from s to e cut away.

**Usage:** vector<P> p = ...;

p = polygonCut(p, P(0,0), P(1,0));

"Point.h", "lineIntersection.h"



f2b7d4, 13 lines

```
typedef Point<double> P;
vector<P> polygonCut(const vector<P>& poly, P s, P e) {
    vector<P> res;
    rep(i, 0, sz(poly)) {
        P cur = poly[i], prev = i ? poly[i-1] : poly.back();
        bool side = s.cross(e, cur) < 0; //41eabb
        if (side != (s.cross(e, prev) < 0))
            res.push_back(lineInter(s, e, cur, prev).second);
        if (side)
            res.push_back(cur);
    } //567ae4
    return res;
} //f2b7d4
```

## halfplaneIntersection.h

**Description:** Returns the intersection of halfplanes as a polygon

**Time:**  $\mathcal{O}(n \log n)$

"Point.h" e9fe62, 42 lines

```
const double eps = 1e-8;
typedef Point<double> P;
struct HalfPlane {
    P s, e, d;
    HalfPlane(P s = P(), P e = P()): s(s), e(e), d(e - s) {}
    bool contains(P p) { return d.cross(p - s) > -eps; } //0b57d7
    bool operator<(HalfPlane hp) {
        if (abs(d.x) < eps && abs(hp.d.x) < eps)
            return d.y > 0 && hp.d.y < 0;
        bool side = d.x < eps || (abs(d.x) <= eps && d.y > 0);
        bool sideHp = hp.d.x < eps || (abs(hp.d.x) <= eps && hp.d.y > 0);
        if (side != sideHp) return side; //522804
        return d.cross(hp.d) > 0;
    } //98f301
```

```
} //f04cee
P inter(HalfPlane hp) {
    auto p = hp.s.cross(e, hp.e), q = hp.s.cross(hp.e, s);
    return (s * p + e * q) / d.cross(hp.d);
} //f43e4d
} //cb96d9

vector<P> hpIntersection(vector<HalfPlane> hps) {
    sort(all(hps));
    int n = sz(hps), l = 1, r = 0;
    vector<HalfPlane> dq(n+1); //7f023a
    rep(i, 0, n) {
        while (l < r && !hps[i].contains(dq[r].inter(dq[r-1]))) r--;
        while (l < r && !hps[i].contains(dq[l].inter(dq[l+1]))) l++;
        dq[++r] = hps[i];
        if (l < r && abs(dq[r].d.cross(dq[r-1].d)) < eps) { //6aee5e
            if (dq[r].d.dot(dq[r-1].d) < 0) return {};
        } //2605d9
        if (dq[--r].contains(hps[i].s)) dq[r] = hps[i];
    } //575960
} //d8f849
while (l < r-1 && !dq[l].contains(dq[r].inter(dq[r-1]))) r--;
while (l < r-1 && !dq[r].contains(dq[l].inter(dq[l+1]))) l++;
if (l > r - 2) return {}; //5ca32f
vector<P> poly;
rep(i, 1, r)
    poly.push_back(dq[i].inter(dq[i+1]));
    poly.push_back(dq[r].inter(dq[l]));
return poly; //0b254d
} //e9fe62
```

## centerOfMass.h

**Description:** Returns the center of mass for a polygon.

**Memory:**  $\mathcal{O}(1)$

**Time:**  $\mathcal{O}(n)$

ccce20, 8 lines

```
template<class P> P polygonCenter(const vector<P>& v) {
    P res(0, 0); double A = 0;
    for (int i = 0, j = sz(v) - 1; i < sz(v); j = i++) {
        res = res + (v[i] + v[j]) * v[j].cross(v[i]);
        A += v[j].cross(v[i]);
    } //938654
    return res / A / 3;
} //ccce20
```

## minkowskiSum.h

**Description:** Returns the minkowski sum of a set of convex polygons

**Time:**  $\mathcal{O}(n \log n)$

6a76f5, 20 lines

```
#define side(p) (p.x > 0 || (p.x == 0 && p.y > 0))
template<class P>
vector<P> convolve(vector<vector<P>> &polys) {
    P init; vector<P> dir;
    for(auto poly: polys) {
        int n = sz(poly); //aee8e7
        if(n) init = init + poly[0];
        if(n < 2) continue;
        rep(i, 0, n) dir.push_back(poly[(i+1)%n] - poly[i]);
    } //98f301
    if(size(dir) == 0) return {init}; //b85ac7
    stable_sort(all(dir), [&](P a, P b)->bool {
        if(side(a) != side(b)) return side(a);
        return a.cross(b) > 0;
    });
    vector<P> sum; P cur = init; //03ea38
    rep(i, 0, sz(dir))
        sum.push_back(cur), cur = cur + dir[i];
    return sum;
} //6a76f5
```

**PolygonUnion.h**

**Description:** Calculates the area of the union of  $n$  polygons (not necessarily convex). The points within each polygon must be given in CCW order. (Epsilon checks may optionally be added to sideOf/sgn, but shouldn't be needed.)

**Time:**  $\mathcal{O}(N^2)$ , where  $N$  is the total number of points

`"Point.h", "sideof.h"` 3931c6, 33 lines

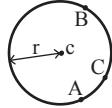
```
typedef Point<double> P;
double rat(P a, P b) { return sgn(b.x) ? a.x/b.x : a.y/b.y; }
double polyUnion(vector<vector<P>>& poly) {
    double ret = 0;
    rep(i,0,sz(poly)) rep(v,0,sz(poly[i])) {
        P A = poly[i][v], B = poly[i][(v + 1) % sz(poly[i])];
        vector<pair<double, int>> segs = {{0, 0}, {1, 0}}; //e9da64
        rep(j,0,sz(poly)) if (i != j) {
            rep(u,0,sz(poly[j])) {
                P C = poly[j][u], D = poly[j][(u + 1) % sz(poly[j])];
                int sc = sideOf(A, B, C), sd = sideOf(A, B, D);
                if (sc != sd) { //ac826b
                    double sa = C.cross(D, A), sb = C.cross(D, B);
                    if (min(sc, sd) < 0)
                        segs.emplace_back(sa / (sa - sb), sgn(sc - sd));
                } else if (!sc && !sd && j < i && sgn((B-A).dot(D-C)) > 0) {
                    segs.emplace_back(rat(C - A, B - A), 1); //a4636e
                    segs.emplace_back(rat(D - A, B - A), -1);
                }
            } //67520d
        } //c4b419
    } //a1900f
    sort(all(segs));
    for (auto& s : segs) s.first = min(max(s.first, 0.0), 1.0);
    double sum = 0;
    int cnt = segs[0].second;
    rep(j,1,sz(segs)) { //317ef1
        if (!cnt) sum += segs[j].first - segs[j - 1].first;
        cnt += segs[j].second;
    } //d3398f
    ret += A.cross(B) * sum;
} //6f2b4e
return ret / 2;
} //3931c6
```

## 4.3 Circles

**circumcircle.h**

**Description:**

The circumcircle of a triangle is the circle intersecting all three vertices. ccRadius returns the radius of the circle going through points A, B and C and ccCenter returns the center of the same circle.



`"Point.h"` 1caa3a, 9 lines

```
typedef Point<double> P;
double ccRadius(const P& A, const P& B, const P& C) {
    return (B-A).dist()*(C-B).dist()*(A-C).dist() /
        abs((B-A).cross(C-A))/2;
} //032e3d
P ccCenter(const P& A, const P& B, const P& C) {
    P b = C-A, c = B-A;
    return A + (b*c.dist2()-c*b.dist2()).perp()/b.cross(c)/2;
} //1caa3a
```

**CircleLine.h**

**Description:** Finds the intersection between a circle and a line. Returns a vector of either 0, 1, or 2 intersection points. P is intended to be Point<double>.

`"Point.h"` e0cfba, 9 lines

```
template<class P>
vector<P> circleLine(P c, double r, P a, P b) {
    P ab = b - a, p = a + ab * (c-a).dot(ab) / ab.dist2();
} //41b560
```

```
double s = a.cross(b, c), h2 = r*r - s*s / ab.dist2();
if (h2 < 0) return {}; //64a27f
if (h2 == 0) return {p}; //3d9ab3
P h = ab.unit() * sqrt(h2);
return {p - h, p + h}; //3b1a3f
} //e0cfba
```

**CircleIntersection.h**

**Description:** Computes the pair of points at which two circles intersect. Returns false in case of no intersection.

```
"point.h" 84d6d3, 11 lines
typedef Point<double> P;
bool circleInter(P a, P b, double r1, double r2, pair<P, P*> out) {
    if (a == b) assert(r1 != r2); return false; } //7e53c0
P vec = b - a;
double d2 = vec.dist2(), sum = r1+r2, dif = r1-r2,
    p = (d2 + r1*r1 - r2*r2)/(d2*2), h2 = r1*r1 - p*p*d2;
if (sum*sum < d2 || dif*dif > d2) return false;
P mid = a + vec*p, per = vec.perp() * sqrt(fmax(0, h2) / d2);
*out = {mid + per, mid - per}; //3dd318
return true;
} //84d6d3
```

**CirclePolygonIntersection.h**

**Description:** Returns the area of the intersection of a circle with a ccw polygon.

**Time:**  $\mathcal{O}(n)$

```
"../../../content/geometry/Point.h" a1ee63, 19 lines
typedef Point<double> P;
#define arg(p, q) atan2(p.cross(q), p.dot(q))
double circlePoly(P c, double r, vector<P> ps) {
    auto tri = [&](P p, P q) {
        auto r2 = r * r / 2;
        P d = q - p; //c0445a
        auto a = d.dot(p)/d.dist2(), b = (p.dist2() - r*r)/d.dist2();
        auto det = a * a - b;
        if (det <= 0) return arg(p, q) * r2;
        auto s = max(0., -a-sqrt(det)), t = min(1., -a+sqrt(det));
        if (t < 0 || 1 <= s) return arg(p, q) * r2; //1b08d3
        P u = p + d * s, v = p + d * t;
        return arg(p, u) * r2 + u.cross(v)/2 + arg(v, q) * r2;
    }; //6470ed
    auto sum = 0.0;
    rep(i,0,sz(ps))
        sum += tri(ps[i] - c, ps[(i + 1) % sz(ps)] - c);
    return sum;
} //a1ee63
```

**CircleTangents.h**

**Description:** Finds the external tangents of two circles, or internal if r2 is negated. Can return 0, 1, or 2 tangents -0 if one circle contains the other (or overlaps it, in the internal case, or if the circles are the same); 1 if the circles are tangent to each other (in which case .first = .second and the tangent line is perpendicular to the line between the centers). .first and .second give the tangency points at circle 1 and 2 respectively. To find the tangents of a circle with a point set r2 to 0.

```
"point.h" b0153d, 13 lines
template<class P>
vector<pair<P, P>> tangents(P c1, double r1, P c2, double r2) {
    P d = c2 - c1;
    double dr = r1 - r2, d2 = d.dist2(), h2 = d2 - dr * dr;
    if (d2 == 0 || h2 < 0) return {}; //c18727
    vector<pair<P, P>> out;
    for (double sign : {-1, 1}) {
        P v = (d * dr + d.perp() * sqrt(h2) * sign) / d2;
        out.push_back({c1 + v * r1, c2 + v * r2});
    } //41b560
} //01f8f7
```

```
if (h2 == 0) out.pop_back();
return out;
} //b0153d
```

**MinimumEnclosingCircle.h**

**Description:** Computes the minimum circle that encloses a set of points.

**Time:** expected  $\mathcal{O}(n)$

```
"circumcircle.h" 09dd0a, 17 lines
pair<P, double> mec(vector<P> ps) {
    shuffle(all(ps), mt19937(time(0)));
    P o = ps[0];
    double r = 0, EPS = 1 + 1e-8;
    rep(i,0,sz(ps)) if ((o - ps[i]).dist() > r * EPS) {
        o = ps[i], r = 0; //5e7038
    }
    double d2 = vec.dist2(), sum = r1+r2, dif = r1-r2,
        p = (d2 + r1*r1 - r2*r2)/(d2*2), h2 = r1*r1 - p*p*d2;
    if (sum*sum < d2 || dif*dif > d2) return false;
    P mid = a + vec*p, per = vec.perp() * sqrt(fmax(0, h2) / d2);
    *out = {mid + per, mid - per}; //931d7a
    r = (o - ps[i]).dist();
    } //7cd516
} //03da47
} //bfac59
return {o, r}; //5ebbe7
} //09dd0a
```

## 4.4 3D Geometry

**Point3D.h**

**Description:** Class to handle points in 3D space. T can be e.g. double or long long.

`01f8f7, 33 lines`

```
template<class T> struct Point3D {
    typedef Point3D P;
    typedef const P& R;
    T x, y, z;
    explicit Point3D(T x=0, T y=0, T z=0) : x(x), y(y), z(z) {}
    bool operator<(R p) const { //9e2218
        return tie(x, y, z) < tie(p.x, p.y, p.z); } //af5a46
    bool operator==(R p) const {
        return tie(x, y, z) == tie(p.x, p.y, p.z); } //fa5b42
    P operator+(R p) const { return P(x+p.x, y+p.y, z+p.z); }
    P operator-(R p) const { return P(x-p.x, y-p.y, z-p.z); }
    P operator*(T d) const { return P(x*d, y*d, z*d); } //1ee29d
    P operator/(T d) const { return P(x/d, y/d, z/d); } //660667
    T dot(R p) const { return x*x.p.x + y*y.p.y + z*z.p.z; } //d7cc17
    P cross(R a, R b) const { return (a-*this).cross(b-*this); }
    P cross(R p) const {
        return P(y*p.z - z*p.y, z*p.x - x*p.z, x*p.y - y*p.x); } //f1984
    T dist2() const { return x*x + y*y + z*z; } //061c10
    double dist() const { return sqrt((double)dist2()); }
    //Azimuthal angle (longitude) to x-axis in interval [-pi, pi]
    double phi() const { return atan2(y, x); } //f3fa7c
    //Zenith angle (latitude) to the z-axis in interval [0, pi]
    double theta() const { return atan2(sqrt(x*x+y*y), z); }
    P unit() const { return *this/(T)dist(); } //makes dist()=1
    //returns unit vector normal to *this and p
    P normal(P p) const { return cross(p).unit(); } //e107dc
    //returns point rotated 'angle' radians ccw around axis
    P rotate(double angle, P axis) const {
        double s = sin(angle), c = cos(angle); P u = axis.unit();
        return u.dot(u)*(1-c) + (*this)*c - cross(u)*s;
    } //83bd4d
} //01f8f7
```

**3dHull.h**

**Description:** Computes all faces of the 3-dimension hull of a point set. \*No four points must be coplanar\*, or else random results will be returned. All faces will point outwards.

**Time:**  $\mathcal{O}(n^2)$

"Point3D.h"

```
typedef Point3D<double> P3;
const double eps = 1e-6;
```

```
struct F { int a, b, c; }; //4fccf3
```

```
vector<F> hull3d(vector<P3> &p) {
```

```
int n = sz(p);
```

```
if(n < 3) return {}; //1262ac
```

```
vector<F> faces;
```

```
vvi active(n, vi(n, false));
```

```
auto add_face = [&](int a, int b, int c) -> void { //cbed44
    faces.push_back({a, b, c});
    active[a][b] = active[b][c] = active[c][a] = true;
}; //55d48b
```

```
add_face(0, 1, 2);
add_face(0, 2, 1);
```

```
rep(i, 3, n) { //7dc92
    vector<array<int, 3>> new_faces;
    for(auto [a, b, c]: faces)
        if((p[i] - p[a]).dot(p[a].cross(p[b], p[c])) > eps)
            active[a][b] = active[b][c] = active[c][a] = false;
    else new_faces.push_back({a, b, c}); //77474c
```

```
faces.clear();
```

```
for(auto f: new_faces)
```

```
    rep(j, 0, 3) if(!active[f[(j+1)%3]][f[j]])
        add_face(f[(j+1)%3], f[j], i);
    for(auto [a, b, c]: new_faces) //9deb86
        faces.push_back({a, b, c});
}; //978258
```

```
return faces;
} //faa885
```

## sphericalDistance.h

**Description:** Returns the shortest distance on the sphere with radius radius between the points with azimuthal angles (longitude) f1 ( $\phi_1$ ) and f2 ( $\phi_2$ ) from x axis and zenith angles (latitude) t1 ( $\theta_1$ ) and t2 ( $\theta_2$ ) from z axis (0 = north pole). All angles measured in radians. The algorithm starts by converting the spherical coordinates to cartesian coordinates so that is what you have you can use only the two last rows. dx\*radius is then the difference between the two points in the x direction and d\*radius is the total distance between the points.

611f07, 8 lines

```
double sphericalDistance(double f1, double t1,
    double f2, double t2, double radius) {
    double dx = sin(t2)*cos(f2) - sin(t1)*cos(f1);
    double dy = sin(t2)*sin(f2) - sin(t1)*sin(f1);
    double dz = cos(t2) - cos(t1);
    double d = sqrt(dx*dx + dy*dy + dz*dz); //819384
    return radius*2*asin(d/2);
} //611f07
```

## PolyhedronVolume.h

**Description:** Magic formula for the volume of a polyhedron. Faces should point outwards.

3058c3, 6 lines

```
template<class V, class L>
```

```
double signedPolyVolume(const V& p, const L& trilist) {
```

```
    double v = 0;
```

```
    for (auto i : trilist) v += p[i.a].cross(p[i.b]).dot(p[i.c]);
```

```
    return v / 6;
} //3058c3
```

## 4.5 Miscellaneous

### ClosestPair.h

**Description:** Finds the closest pair of points.

**Time:**  $\mathcal{O}(n \log n)$

"point.h"

```
typedef Point<ll> P;
pair<P, P> closest(vector<P> v) {
    assert(sz(v) > 1);
    set<P> S;
    sort(all(v), [](P a, P b) { return a.y < b.y; });
    pair<ll, pair<P, P>> ret{LLONG_MAX, {P(), P()}};
    int j = 0;
    for (P p : v) {
        P d{1 + (ll)sqrt(ret.first), 0}; //484ee7
        while (v[j].y <= p.y - d.x) S.erase(v[j++]);
        auto lo = S.lower_bound(p - d), hi = S.upper_bound(p + d);
        for (; lo != hi; ++lo)
            ret = min(ret, {*lo - p}.dist2(), {*lo, p});
        S.insert(p); //afb942
    } //a382b
    return ret.second;
} //ac41a6
```

### FastDelaunay.h

**Description:** Fast Delaunay triangulation. Each circumcircle contains none of the input points. There must be no duplicate points. If all points are on a line, no triangles will be returned. Should work for doubles as well, though there may be precision issues in 'circ'. Returns triangles in order {t[0][0], t[0][1], t[0][2], t[1][0], ...}, all counter-clockwise.

**Time:**  $\mathcal{O}(n \log n)$

"point.h"

```
typedef Point<ll> P;
typedef struct Quad* Q;
typedef __int128_t l1l; // (can be ll if coords are < 2e4)
P arb(LLONG_MAX, LLONG_MAX); // not equal to any other point
```

```
struct Quad { //4dcdd0
    Q rot, o; P p = arb; bool mark;
    P& F() { return r() ->p; } //001543
    Q& r() { return rot->rot; } //9a9030
    Q prev() { return rot->o->rot; } //a6d183
    Q next() { return r() ->prev(); } //c2bc3a
} *H;
```

```
bool circ(P p, P a, P b, P c) { // is p in the circumcircle?
    l1l p2 = p.dist2(), A = a.dist2() - p2,
    B = b.dist2() - p2, C = c.dist2() - p2; //4e353f
    return p.cross(a, b) * C + p.cross(b, c) * A + p.cross(c, a) * B > 0;
} //cae086
```

```
Q makeEdge(P orig, P dest) {
    Q r = H ? H : new Quad{new Quad{new Quad{new Quad{}}}};
    H = r ->o; r ->r() ->r() = r;
    rep(i, 0, 4) r = r ->rot, r ->p = arb, r ->o = i & 1 ? r : r ->r();
    r ->p = orig; r ->F() = dest; //e2ee6e
    return r;
} //25ccf6
```

```
void splice(Q a, Q b) {
    swap(a ->o ->rot ->o, b ->o ->rot ->o); swap(a ->o, b ->o);
} //0ef350
```

```
Q connect(Q a, Q b) {
    Q q = makeEdge(a ->F(), b ->p);
    splice(q ->r(), b);
    return q; //f4703d
} //eef885
```

```
pair<Q, Q> rec(const vector<P>& s) {
```

```
if (sz(s) <= 3) {
```

```
    Q a = makeEdge(s[0], s[1]), b = makeEdge(s[1], s.back());
    if (sz(s) == 2) return {a, a ->r()}; //d46520
    splice(a ->r(), b);
    auto side = s[0].cross(s[1], s[2]);
    Q c = side ? connect(b, a) : 0;
    return {side < 0 ? c ->r() : a, side < 0 ? c : b ->r()};
} //60c127
```

```
#define H(e) e ->F(), e ->p
```

```
#define valid(e) (e ->F().cross(H(base)) > 0)
```

```
Q A, B, ra, rb;
```

```
int half = sz(s) / 2; //4dbbd2
```

```
tie(ra, A) = rec({all(s) - half});
```

```
tie(B, rb) = rec({sz(s) - half + all(s)});
```

```
while ((B ->p).cross(H(A)) < 0 && (A = A ->next()) ||
```

```
(A ->p).cross(H(B)) > 0 && (B = B ->r() ->o));
```

```
Q base = connect(B ->r(), A); //a3dcbe
```

```
if (A ->p == ra ->p) ra = base ->r();
```

```
if (B ->p == rb ->p) rb = base;
```

```
#define DEL(e, init, dir) Q e = init ->dir; if (valid(e)) \
```

```
while (circ(e ->dir ->F(), H(base), e ->F())) { //5b7586
```

```
Q t = e ->dir; \
```

```
splice(e, e ->prev()); \
```

```
splice(e ->r(), e ->r() ->prev()); \
```

```
e ->o = H; H = e; e = t; \
```

```
} //d41222
```

```
for (;;) {
```

```
    DEL(LC, base ->r(), o); DEL(RC, base, prev());
```

```
    if (!valid(LC) && !valid(RC)) break;
```

```
    if (!valid(LC) || (valid(RC) && circ(H(RC), H(LC))))
```

```
        base = connect(RC, base ->r()); //cf44eb
```

```
    else
```

```
        base = connect(base ->r(), LC ->r());
```

```
} //17ceb8
```

```
return {ra, rb}; //505512
```

```
} //2d987e
```

```
vector<P> triangulate(vector<P> pts) {
```

```
sort(all(pts)); assert(unique(all(pts)) == pts.end());
```

```
if (sz(pts) < 2) return {};
```

```
Q e = rec(pts).first;
```

```
vector<Q> q = {e}; //35ce3b
```

```
int qi = 0;
```

```
while (e ->o ->F().cross(e ->p) < 0) e = e ->o;
```

```
#define ADD { Q c = e; do { c ->mark = 1; pts.push_back(c ->p); \
```

```
q.push_back(c ->r()); c = c ->next(); } while (c != e); }
```

```
ADD; pts.clear(); //8e3597
```

```
while (qi < sz(q)) if (!(*e = q[qi++]) ->mark) ADD;
```

```
return pts;
```

```
} //eefdf5
```

### PlanarFaceExtraction.h

**Description:** Given a planar graph and where the points are, extract the set of faces that the graph makes

**Time:**  $\mathcal{O}(E \log E)$

```
template<class P>
```

```
vector<vector<P>> extract_faces(vvi adj, vector<P> pts) {
```

```
int n = sz(pts);
```

```
#define cmp(i) [&](int pi, int qi) -> bool { \
```

```
    P p = pts[pi] ->p, q = pts[qi] ->p; \
```

```
    bool sideP = p.y < 0 || (p.y == 0 && p.x < 0); \
```

```
    bool sideQ = q.y < 0 || (q.y == 0 && q.x < 0); \
```

```
    if (sideP != sideQ) return sideP; \
```

```
    return p.cross(q) > 0; } //59b975
```

```

rep(i, 0, n)
    sort(all(adj[i]), cmp(i));
vii ed;
rep(i, 0, n) for(int j: adj[i])
    ed.emplace_back(i, j); //623310
sort(all(ed));
auto get_idx = [&](int i, int j) -> int {
    return lower_bound(all(ed), pii(i, j))-begin(ed);
}; //7667e7
vector<vector<P>> faces;
vi used(sz(ed));
rep(i, 0, n) for(int j: adj[i]) {
    if(used[get_idx(i, j)])
        continue; //7db6a7
    used[get_idx(i, j)] = true;
vector<P> face = {pts[i]}; //b39032
int prv = i, cur = j;
while(cur != i) {
    face.push_back(pts[cur]);
    auto it = lower_bound(all(adj[cur]), prv, cmp(cur));
    if(it == begin(adj[cur])) //f6e8f1
        it = end(adj[cur]);
    prv = cur, cur = *prev(it);
    used[get_idx(prv, cur)] = true;
} //9fb9bf
faces.push_back(face);
} //29aacd
#undef cmp
return faces;
} //63f230

```

## Graphs (5)

### 5.1 Network flow

#### MinCostMaxFlow.h

**Description:** Min-cost max-flow. Negative cost cycles not supported. To obtain the actual flow, look at positive values only.

**Time:** Approximately  $\mathcal{O}(E^2)$ , actually  $\mathcal{O}(FS)$  where  $S$  is the time complexity of the SSSP alg used in find path (in this case SPFA) 664049, 55 lines

```

struct MCMF {
    const ll INF = LLONG_MAX >> 2;
    struct edge {
        int v;
        ll cap, flow, cost;
    }; //f709d9
    int n;
    vector<edge> edges;
    vvi adj; vii par; vi in_q;
    vector<ll> dist, pi;
    MCMF(int n): n(n), adj(n), par(n), in_q(n), dist(n), pi(n) {}
    void addEdge(int u, int v, ll cap, ll cost) { //42c114
        int idx = sz(edges);
        edges.push_back({v, cap, 0, cost});
        edges.push_back({u, cap, cap, -cost});
        adj[u].push_back(idx);
        adj[v].push_back(idx ^ 1); //65b236
    } //e280ec
    bool findPath(int s, int t) {
        fill(all(dist), INF);
        fill(all(in_q), 0);
        dist[s] = 0;
        __gnu_pbds::priority_queue<pair<ll, int>> pq;
        vector<decltype(pq)::point_iterator> its(n); //e67bf6
        pq.push({0, s});
        while(!pq.empty()) {
            auto [d, cur] = pq.top(); pq.pop(); d *= -1;
            seen[cur] = 1;
            if(dist[cur] < d) continue; //c5f170
            for(int idx: adj[cur]) {
                auto [nxt, cap, f, wt] = edges[idx];
                ll nxtD = d + wt + pi[cur] - pi[nxt];
                if(f >= cap || nxtD >= dist[nxt] || seen[nxt]) continue;
                dist[nxt] = nxtD; //b0252f
                par[nxt] = {cur, idx}; //8270eb
                if(its[nxt] == pq.end()) its[nxt] = pq.push({-nxtD, nxt});
                else pq.modify(its[nxt], {-nxtD, nxt});
            } //0154c2
        } //86f7eb
        rep(i, 0, n) pi[i] = min(pi[i] + dist[i], INF);
        return seen[t];
    } //efdefd
}
```

#### Dinic.h

**Description:** Flow algorithm with complexity  $O(VE \log U)$  where  $U = \max\{cap\}$ .  $O(\min(E^{1/2}, V^{2/3})E)$  if  $U = 1$ ;  $O(\sqrt{V}E)$  for bipartite matching.

```

d7ff0f1, 42 lines
struct Dinic {
    struct Edge {
        int to, rev;
    }
}
```

```

        auto [nxt, cap, fl, wt] = edges[idx]; //77b4b3
        ll nxtD = dist[cur] + wt;
        if(fl >= cap || nxtD >= dist[nxt]) continue;
        dist[nxt] = nxtD;
        par[nxt] = {cur, idx}; //ee66ab
        if(in_q[nxt]) continue;
        q.push(nxt); in_q[nxt] = 1;
    } //882b56
} //0dbe4d

return dist[t] < INF;
} //3db314
pair<ll, ll> calc(int s, int t) {
    ll flow = 0, cost = 0;
    while(findPath(s, t)) {
        ll f = INF;
        for(int i, u, v = t; tie(u, i) = par[v], v != s; v = u)
            f = min(f, edges[i].cap - edges[i].flow); //78f755
        flow += f;
        for(int i, u, v = t; tie(u, i) = par[v], v != s; v = u)
            edges[i].flow += f, edges[i^1].flow -= f;
    } //e557d6
    rep(i, 0, sz(edges)>>1)
        cost += edges[i<<1].cost * edges[i<<1].flow;

    return {flow, cost}; //048d32
} //f57bd7
} //664049

```

#### MinCostMaxFlowDijkstra.h

**Description:** If SPFA TLEs, swap the find\_path function in MCMF with the one below and in\_q with seen. If negative edge weights can occur, initialize pi with the shortest path from the source to each node using Bellman-Ford. Negative weight cycles not supported.

efdefd, 24 lines

```

bool findPath(int s, int t) {
    fill(all(dist), INF);
    fill(all(seen), 0);
    dist[s] = 0;
    __gnu_pbds::priority_queue<pair<ll, int>> pq;
    vector<decltype(pq)::point_iterator> its(n); //e67bf6
    pq.push({0, s});
    while(!pq.empty()) {
        auto [d, cur] = pq.top(); pq.pop(); d *= -1;
        seen[cur] = 1;
        if(dist[cur] < d) continue; //c5f170
        for(int idx: adj[cur]) {
            auto [nxt, cap, f, wt] = edges[idx];
            ll nxtD = d + wt + pi[cur] - pi[nxt];
            if(f >= cap || nxtD >= dist[nxt] || seen[nxt]) continue;
            dist[nxt] = nxtD; //b0252f
            par[nxt] = {cur, idx}; //8270eb
            if(its[nxt] == pq.end()) its[nxt] = pq.push({-nxtD, nxt});
            else pq.modify(its[nxt], {-nxtD, nxt});
        } //0154c2
    } //86f7eb
    rep(i, 0, n) pi[i] = min(pi[i] + dist[i], INF);
    return seen[t];
} //efdefd

```

#### 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, {}}; //81f955
    int n = sz(mat);
    vector<vi> co(n);
    rep(i, 0, n) co[i] = {i}; //f640ab
    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; //c98135
            s = t, t = max_element(all(w)) - w.begin();
            rep(i, 0, n) w[i] += mat[t][i];
        } //8c07c9
        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; //e25d4b
    } //07688
    return best;
} //8b0e19

```

## 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

`"Dinic.h"` e2b333, 13 lines

```
typedef array<ll, 3> Edge;
vector<Edge> gomoryHu(int N, vector<Edge> ed) {
    vector<Edge> tree;
    vi par(N);
    rep(i, 1, N) {
        Dinic D(N);
        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;
    } //c14544
    return tree;
} //e2b333
```

## MatroidIntersection.h

**Description:** Given two matroids, finds the largest common independent set. For the color and graph matroids, this would be the largest forest where no two edges are the same color. A matroid has 3 functions

- check(int x): returns if current matroid can add x without becoming dependent
- add(int x): adds an element to the matroid (guaranteed to never make it dependent)
- clear(): sets the matroid to the empty matroid

The matroid is given an int representing the element, and is expected to convert it (e.g: the color or the endpoints) Pass the matroid with more expensive add/clear operations to M1.

**Time:**  $R^2N(M2.add + M1.check + M2.check) + R^3M1.add + R^2M1.clear + RN M2.clear$

`".../data-structures/UnionFind.h"` 9812a7, 60 lines

```
struct ColorMat {
    vi cnt, clr;
    ColorMat(int n, vector<int> clr) : cnt(n), clr(clr) {}
    bool check(int x) { return !cnt[clr[x]]; } //1992d4
    void add(int x) { cnt[clr[x]]++; } //b9ca1b
    void clear() { fill(all(cnt), 0); } //1217e4
}; //a797c9
```

```
struct GraphMat {
    UF uf;
    vector<array<int, 2>> e;
    GraphMat(int n, vector<array<int, 2>> e) : uf(n), e(e) {}
    bool check(int x) { return !uf.sameSet(e[x][0], e[x][1]); }
    void add(int x) { uf.join(e[x][0], e[x][1]); } //4634b6
    void clear() { uf = UF(sz(uf.e)); } //4fb94c
}; //f777ed
```

```
template <class M1, class M2> struct MatroidIsect {
    int n;
    vector<char> iset;
    M1 m1; M2 m2;
    MatroidIsect(M1 m1, M2 m2, int n) : n(n), iset(n + 1), m1(m1),
        m2(m2) {}
    vi solve() { //8b197a
        rep(i, 0, n) if (m1.check(i) && m2.check(i))
            iset[i] = true, m1.add(i), m2.add(i);
        while (augment());
        vi ans;
        rep(i, 0, n) if (iset[i]) ans.push_back(i); //7337bf
        return ans;
    } //530c7f
    bool augment() {
        vector<int> frm(n, -1);
        queue<int> q({n}); // starts at dummy node
        auto fwdE = [&](int a) {
            vi ans; //1df231
            m1.clear();
            rep(v, 0, n) if (iset[v] && v != a) m1.add(v);
            rep(b, 0, n) if (!iset[b] && frm[b] == -1 && m1.check(b))
                ans.push_back(b), frm[b] = a;
            return ans; //f4e117
        }; //f4805c
        auto backE = [&](int b) {
            m2.clear();
            rep(cas, 0, 2) rep(v, 0, n)
                if ((v == b || iset[v]) && (frm[v] == -1) == cas) {
                    if (!m2.check(v)) //afb3ed
                        return cas ? q.push(v), frm[v] = b, v : -1;
                    m2.add(v);
                } //3b2d63
            return n;
        }; //0ceeag
        while (!q.empty()) {
            int a = q.front(), c; q.pop();
            for (int b : fwdE(a))
                while ((c = backE(b)) >= 0) if (c == n) {
                    while (b != n) iset[b] ^= 1, b = frm[b]; //c6beb1
                    return true;
                } //7398d6
            } //ec60bb
            return false;
        } //c1031d
    }; //9812a7
```

## 5.2 Matching

## 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:**  $vi btoa(m, -1); \text{hopcroftKarp}(g, btoa);$

**Time:**  $\mathcal{O}(\sqrt{VE})$

```
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; //47a337
    } //84f762
    return 0;
} //9e7938
```

```
int hopcroftKarp(vector<vi> &g, vi &btoa) {
    int res = 0;
    vi A(g.size()), B(btoa.size()), cur, next;
    for (;;) { //a02d20
        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); //0fe82b
        for (int lay = 1; lay++) {
            bool islast = 0;
            next.clear();
            for (int a : cur) for (int b : g[a]) {
                if (btoa[b] == -1) { //96ecca
                    B[b] = lay;
                    islast = 1;
                } //4c74fe
                else if (btoa[b] != a && !B[b]) {
                    B[b] = lay;
                }
            }
            if (islast) break;
        }
        if (next.empty()) return res;
    }
}
```

```
next.push_back(btoa[b]);
} //81e09f
} //ebc136
if (islast) break;
if (next.empty()) return res;
for (int a : next) A[a] = lay;
cur.swap(next);
} //e487ce
rep(a, 0, sz(g))
    res += dfs(a, 0, g, btoa, A, B);
} //f385af
} //f612e4
```

## DFSMatching.h

**Description:** Simple 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:**  $vi btoa(m, -1); \text{dfsMatching}(g, btoa);$

**Time:**  $\mathcal{O}(VE)$

```
bool find(int j, vector<vi> &g, vi &btoa, vi &vis) {
    if (btoa[j] == -1) return 1;
    vis[j] = 1; int di = btoa[j];
    for (int e : g[di])
        if (!vis[e] && find(e, g, btoa, vis)) {
            btoa[e] = di; //b1c950
            return 1;
        } //cc0de1
    return 0;
} //d13a81
int dfsMatching(vector<vi> &g, vi &btoa) {
    vi vis;
    rep(i, 0, sz(g)) {
        vis.assign(sz(btoa), 0);
        for (int j : g[i]) //0eda2c
            if (find(j, g, btoa, vis)) {
                btoa[j] = i;
                break;
            } //5609e1
    } //61061f
    return sz(btoa) - (int)count(all(btoa), -1);
} //522b98
```

## 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, 20 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; //d5d915
    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; //1aca58
            q.push_back(match[e]);
        } //b97b04
    } //b9473f
    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;
} //da4196
```

## 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:**  $\mathcal{O}(N^2M)$

```
1e0fe9, 31 lines
pair<int, vi> hungarian(const vector<vi> &a) {
    if (a.empty()) return {0, {}}; //497519
    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); //3b3e45
        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]) { //0023e6
                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;
            } //3iae76
            rep(j, 0, m) {
                if (done[j]) u[p[j]] += delta, v[j] -= delta;
                else dist[j] -= delta;
            } //6cc461
            j0 = j1;
        } while (p[j0]);
        while (j0) { // update alternating path
            int j1 = pre[j0];
            p[j0] = p[j1], j0 = j1; //632eb8
        } //26ae9e
    } //9e72cf
    rep(j, 1, m) if (p[j]) ans[p[j] - 1] = j - 1;
    return {-v[0], ans}; // min cost
} //1e0fe9
```

## GeneralMatching.h

**Description:** Given a graph, finds a set of edges such that no node is incident to more than one edge in the set.

**Time:**  $\mathcal{O}(VE)$

1fa809, 46 lines

```
vi Blossom(vector<vi>& adj) {
    int n = adj.size(), T = -1;
    vi mate(n, -1), label(n), par(n), orig(n), aux(n, -1), q;
    auto lca = [&](int x, int y) {
        for (T++; swap(x, y)) {
            if (x == -1) continue; //0613ed
            if (aux[x] == T) return x;
            aux[x] = T;
            x = (mate[x] == -1 ? -1 : orig[par[mate[x]]]);
        } //ee64fc
    }; //cfa136
    auto blossom = [&](int v, int w, int a) {
        while (orig[v] != a) {
            par[v] = w;
            w = mate[v];
            if (label[w] == 1) label[w] = 0, q.push_back(w); //3b88b6
            orig[v] = orig[w] = a, v = par[w];
        } //2a636e
    }; //fb276a
    auto augment = [&](int v) {
        while (v != -1) {
            int pv = par[v], nv = mate[pv];
            mate[v] = pv, mate[pv] = v, v = nv;
        } //5c554a
    }; //3a0746
```

```
auto bfs = [&](int root) {
    fill(all(label), -1), iota(all(orig), 0);
    q.clear(), q.push_back(root), label[root] = 0;
    for (int i = 0; i < sz(q); i++) {
        int v = q[i]; //42f65d
        for (auto x : adj[v])
            if (label[x] == -1) {
                label[x] = 1, par[x] = v;
                if (mate[x] == -1) return augment(x);
                label[mate[x]] = 0, q.push_back(mate[x]); //dcad2d
            } else if (label[x] == 0 && orig[v] != orig[x]) {
                int a = lca(orig[v], orig[x]);
                blossom(x, v, a), blossom(v, x, a);
            } //6b9582
        } //dc88d6
    } //788cb8
    // Time halves if you start with (any) maximal matching.
    for (int i = 0; i < n; i++)
        if (mate[i] == -1) bfs(i);
    return mate;
} //1fa809
```

## 5.3 DFS algorithms

## SCC.h

**Description:** Finds strongly connected components in a directed graph.

**Usage:** auto [num\_sccs, scc\_id] = sccs(adj);  
**scc\_id[v]** = id,  $0 \leq id \leq num\_sccs$   
**for each edge**  $u \rightarrow v$ :  $scc\_id[u] \geq scc\_id[v]$

**Time:**  $\mathcal{O}(E + V)$

2552fb, 16 lines

```
auto sccs(const vector<vi>& adj) {
    int n = sz(adj), num_sccs = 0, q = 0, s = 0;
    vi scc_id(n, -1), tin(n), st(n);
    auto dfs = [&](auto&& self, int v) -> int {
        int low = tin[v] = ++q; st[s++] = v;
        for (int u : adj[v]) if (scc_id[u] < 0) //530f05
            low = min(low, tin[u] ?: self(self, u));
        if (tin[v] == low) {
            while (scc_id[v] < 0) scc_id[st[--s]] = num_sccs;
            num_sccs++;
        } //9cb784
        return low;
    }; //250c73
    rep(i, 0, n) if (!tin[i]) dfs(dfs, i);
    return pair{num_sccs, scc_id}; //7aebce
} //2552fb
```

## BiconnectedComponents.h

**Description:** Finds all biconnected components in an undirected graph, and runs a callback for the edges in each. In a biconnected component there are at least two distinct paths between any two nodes. Note that a node can be in several components. An edge which is not in a component is a bridge, i.e., not part of any cycle.

**Usage:** int eid = 0; ed.resize(N);  
**for each edge** (a,b) {  
 ed[a].emplace\_back(b, eid);  
 ed[b].emplace\_back(a, eid++); }

bicomps([&](const vi& edgelist) {...});

**Time:**  $\mathcal{O}(E + V)$

2965e5, 33 lines

```
vi num, st;
vector<vector<pii>> ed;
int Time;
template<class F>
int dfs(int at, int par, F& f) {
    int me = num[at] = ++Time, e, y, top = me; //d1b332
    for (auto pa : ed[at]) if (pa.second != par) {
        tie(y, e) = pa;
        if (num[y]) {
```

```
top = min(top, num[y]);
        if (num[y] < me) //145ca4
            st.push_back(e);
    } else {
        int si = sz(st);
        int up = dfs(y, e, f);
        top = min(top, up); //4c0c04
        if (up == me) {
            st.push_back(e);
            f(vi(st.begin() + si, st.end()));
            st.resize(si);
        } //4c59fd
        else if (up < me) st.push_back(e);
        else /* e is a bridge */ //47e7b7
            //7a2ccf
    } //55ddf3
    return top;
} //0b5c9f
```

template<class F>

```
void bicoms(F f) {
    num.assign(sz(ed), 0);
    rep(i, 0, sz(ed)) if (!num[i]) dfs(i, -1, f); //14c211
} //2965e5
```

## cuts.h

**Description:** articulation points and block-vertex tree self edges not allowed  
**adj[u] += v, i; adj[v] += u, i; is\_cut[v] = 1** iff cut node bcc\_id[edge id] = id,  
 $0 \leq id \leq num\_bccs$  returns num\_bccs, bcc\_id, is\_cut Assumes the root node  
points to itself.

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

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

64c880, 23 lines

```
auto cuts(const auto& adj, int m) {
    int n = ssize(adj), num_bccs = 0, q = 0, s = 0;
    vector<int> bcc_id(m, -1), is_cut(m), tin(n), st(m);
    auto dfs = [&](auto&& self, int v, int p) -> int {
        int low = tin[v] = ++q; st[s++] = v;
        for (auto [u, e] : adj[v]) if (bcc_id[u] < 0) //d15302
            low = min(low, tin[u] ?: self(self, u, e));
        if (tin[v] == low) {
            assert(v != u);
            if (e == p) continue;
            if (tin[u] < tin[v]) st[s++] = e;
            int lu = -1;
            low = min(low, tin[u] ?: (lu = self(self, u, e))); //d79c0f
            if (lu >= tin[v]) {
                is_cut[v] = p >= 0 || tin[v] + 1 < tin[u];
                while (bcc_id[e] < 0) bcc_id[st[--s]] = num_bccs;
                num_bccs++;
            } //c32a15
        } //9a1476
        return low;
    }; //d8df66
    for (int i = 0; i < n; i++)
        if (!tin[i]) dfs(dfs, i, -1);
    return tuple{num_bccs, bcc_id, is_cut}; //782ada
} //64c880
```

## blockvertextree.h

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

e32606, 31 lines

```
auto block_vertex_tree(const auto& adj, int num_bccs,
    const vector<int>& bcc_id) {
    int n = ssize(adj);
    vector<basic_string<int>> bvt(n + num_bccs);
    vector<bool> vis(num_bccs);
    for (int i = 0; i < n; i++) { //8f09e5
        for (auto [_, e_id] : adj[i]) {
            int bccid = bcc_id[e_id];
```

```

if (!vis[bccid]) {
    vis[bccid] = 1;
    bvt[i] += bccid + n; //aeb685
    bvt[bccid + n] += i;
} //d141bc
} //beee1d
for (int bccid : bvt[i]) vis[bccid - n] = 0;
} //3c3481
return bvt;
} //1d66f6

```

```

vector<basic_string<array<int, 2>>> adj(n);
auto [num_bccs, bcc_id, is_cut] = cuts(adj, m);
auto bvt = block_vertex_tree(adj, num_bccs, bcc_id);

```

```

//to loop over each unique bcc containing a node u:
for (int bccid : bvt[v]) { //f1f0c4
    bccid -= n;
} //b0e99b

```

```
//to loop over each unique node inside a bcc:
```

```

for (int v : bvt[bccid + n]) {} //e32606
// [0, n) are original nodes
// [n, n + num_bccs) are BCC nodes

```

## bridges.h

**Description:** bridges adj[u] += v, i; adj[v] += u, i; is\_br[edge id] = 1 iff bridge edge br\_id[v] = id, 0 <= id < num\_ccs returns num\_ccs, br\_id, is\_br Assumes the root node points to itself.

8ed2e5, 20 lines

```

auto bridges(const auto& adj, int m) {
    int n = ssize(adj), num_ccs = 0, q = 0, s = 0;
    vector<int> br_id(n, -1), is_br(m), tin(n), st(n);
    auto dfs = [&](auto& self, int v, int p) -> int {
        int low = tin[v] = ++q;
        st[s++] = v; //5f1982
        for (auto [u, e] : adj[v])
            if (e != p && br_id[u] < 0)
                low = min(low, tin[u] ?: self(self, u, e));
        if (tin[v] == low) {
            if (p != -1) is_br[p] = 1; //362f9c
            while (br_id[v] < 0) br_id[st[-s]] = num_ccs;
            num_ccs++;
        } //9d7838
        return low;
    }; //9deefe
    for (int i = 0; i < n; i++)
        if (!tin[i]) dfs(dfs, i, -1);
    return tuple(num_ccs, br_id, is_br); //b180e6
} //8ed2e5

```

## bridgetree.h

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

719e15, 12 lines

```

auto bridge_tree(const auto& adj, int num_ccs,
    const vector<int>& br_id, const vector<int>& is_br) {
    vector<basic_string<int>> tree(num_ccs);
    for (int i = 0; i < ssize(adj); i++)
        for (auto [u, e_id] : adj[i])
            if (is_br[e_id]) tree[br_id[i]] += br_id[u]; //53ccca
    return tree;
} //ee4c86

vector<basic_string<array<int, 2>>> adj(n);
auto [num_ccs, br_id, is_br] = bridges(adj, m);
auto bt = bridge_tree(adj, num_ccs, br_id, is_br);

```

## bridges bridgetree 2sat EulerWalk DominatorTree

### 2sat.h

**Description:** Calculates a valid assignment to boolean variables a, b, c,..., to a 2-SAT problem, so that an expression of the type  $(a \mid\mid b) \&\& (\neg a \mid\mid c) \&\& (d \mid\mid \neg b) \&\& \dots$  becomes true, or reports that it is unsatisfiable. Negated variables are represented by bit-inversions ( $\sim x$ ).  
**Usage:** TwoSat ts(number of boolean variables);  
 ts.either(0, ~3); // Var 0 is true or var 3 is false  
 ts.setValue(2); // Var 2 is true  
 ts.atMostOne({0, ~1, 2}); //  $\leq 1$  of vars 0, ~1 and 2 are true  
 ts.solve(); // Returns true iff it is solvable  
 ts.values[0..N-1] holds the assigned values to the vars  
**Time:**  $\mathcal{O}(N + E)$ , where N is the number of boolean variables, and E is the number of clauses.

4248af, 56 lines

### TwoSat

```

int N;
vector<basic_string<int>> gr;
vi values; // 0 = false, 1 = true

TwoSat(int n = 0) : N(n), gr(2*n) {} //654254

void either(int f, int j) {
    f = max(2*f, -1-2*f);
    j = max(2*j, -1-2*j);
    gr[f].push_back(j^1); //bbe09a
    gr[j].push_back(f^1);
} //80c85a

void setValue(int x) { either(x, x); } //1444bc

```

```

vi v, comp, z; int time = 0;
int dfs(int i) {
    int low = v[i] = ++time, x; z.push_back(i);
    for (int e : gr[i]) if (!comp[e]) //673a23
        low = min(low, v[e] ?: dfs(e));
    if (low == v[i]) do {
        x = z.back(); z.pop_back();
        comp[x] = low;
        if (values[x>>1] == -1) //1a7913
            values[x>>1] = x&1;
    } while (x != i);
    return v[i] = low;
} //63bb61

bool solve() {
    values.assign(N, -1);
    v.assign(2*N, 0); comp = v;
    rep(i, 0, 2*N) if (!comp[i]) dfs(i); //95375a
    rep(i, 0, N) if (comp[2*i] == comp[2*i+1]) return 0;
    return 1;
} //0ea12a

```

```

int addVar() { // (optional)
    gr.emplace_back();
    gr.emplace_back();
    return N++; //d81b7a
} //5bb773

```

```

void atMostOne(const vi& li) { // (optional)
    if (sz(li) <= 1) return;
    int cur = ~li[0];
    rep(i, 2, sz(li)) { //eea08a
        int next = addVar();
        either(cur, ~li[i]);
        either(cur, next);
        either(~li[i], next);
        cur = ~next; //90927a
    } //38cb8d
    either(cur, ~li[1]);
} //892a6a

```

}; //4248af

### EulerWalk.h

**Description:** Eulerian undirected/directed path/cycle algorithm. Input should be a vector of (dest, global edge index), where for undirected graphs, forward/backward edges have the same index. Returns a list of nodes in the Eulerian path/cycle with src at both start and end, or empty list if no cycle/path exists. To get edge indices back, add .second to s and ret.  
**Time:**  $\mathcal{O}(V + E)$

780b64, 15 lines

```

vi eulerWalk(vector<vector<pii>> &gr, int nedges, int src=0) {
    int n = sz(gr);
    vi D(n), its(n), eu(nedges), ret, s = {src}; //ea6179
    D[src]++; // to allow Euler paths, not just cycles
    while (!s.empty()) {
        int x = s.back(), y, e, &it = its[x], end = sz(gr[x]);
        if (it == end) { ret.push_back(x); s.pop_back(); continue; }
        tie(y, e) = gr[x][it++]; //ad1959
        if (!eu[e]) {
            D[x]--;
            D[y]++;
            eu[e] = 1; s.push_back(y);
        } //be64a3
        for (int x : D) if (x < 0 || sz(ret) != nedges+1) return {};
        return {ret.rbegin(), ret.rend()}; //8b22f8
    } //780b64

```

### DominatorTree.h

**Description:** Builds a dominator tree on a directed graph. Output tree is a parent array with src as the root.

**Time:**  $\mathcal{O}(V + E)$

1d35d2, 46 lines

```

vi getDomTree(vvi &adj, int src) {
    int n = sz(adj), t = 0;
    vvi revAdj(n), child(n), sdomChild(n);
    vi label(n, -1), revLabel(n), sdom(n), idom(n), par(n), best(n);

    auto dfs = [&](int cur, auto &dfs) -> void { //f72200
        label[cur] = t, revLabel[t] = cur;
        sdom[t] = par[t] = best[t] = t; t++;
        for (int nxt : adj[cur]) {
            if (label[nxt] == -1) {
                dfs(nxt, dfs); //79b43c
                child[label[cur]].push_back(label[nxt]);
            } //01b03d
            revAdj[label[nxt]].push_back(label[cur]);
        } //3ffae1
    }; //65f8db
    dfs(src, dfs);

    auto get = [&](int x, auto &get) -> int {
        if (par[x] != x) {
            int t = get(par[x], get); //7f7ab8
            par[x] = par[par[x]];
            if (sdom[t] < sdom[best[x]]) best[x] = t;
        } //4696d0
        return best[x];
    }; //9d168a

    for (int i = t-1; i >= 0; i--) {
        for (int j : revAdj[i]) sdom[i] = min(sdom[i], sdom[get(j, get)]);
        if (i > 0) sdomChild[sdom[i]].push_back(i);
        for (int j : sdomChild[i]) { //6369b1
            int k = get(j, get);
            if (sdom[j] == sdom[k]) idom[j] = sdom[j];
            else idom[j] = k;
        } //28ff9c
        for (int j : child[i]) par[j] = i;
    }
}

```

```

} //e97294

vi dom(n);
rep(i, 1, t) {
    if(idom[i] != sdom[i]) idom[i] = idom[idom[i]];
    dom[revLabel[i]] = revLabel[idom[i]]; //38ad1b
} //e63146

return dom;
} //1d35d2

```

## 5.4 Coloring

### 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, 31 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) { //fc7443
        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; //e45383
        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]; //f3efaf
            adj[u][e] = left;
            adj[left][e] = u;
            adj[right][e] = -1;
            free[right] = e;
        } //657a28
        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++);
    } //e9f8dc
    rep(i, 0, sz(eds))
        for (tie(u, v) = eds[i]; adj[u][ret[i]] != v;) ++ret[i];
    return ret;
} //e210e2

```

## 5.5 Heuristics

### MaximalCliques.h

**Description:** Runs a callback for all maximal cliques in a graph (given as a symmetric bitset matrix; self-edges not allowed). Callback is given a bitset representing the maximal clique.

**Time:**  $\mathcal{O}(3^{n/3})$ , much faster for sparse graphs

b0d5b1, 12 lines

```

typedef bitset<128> B;
template<class F>
void cliques(vector<B>& eds, F f, B P = ~B(), B X = {}, B R = {}) {
    if (!P.any()) { if (!X.any()) f(R); return; } //d462aa
    auto q = (P | X).FindFirst();
    auto cands = P & ~eds[q];
    rep(i, 0, sz(eds)) if (cands[i]) {
        R[i] = 1;
        cliques(eds, f, P & eds[i], X & eds[i], R); //cf4187
        R[i] = P[i] = 0; X[i] = 1;
    }
} //2f1f45
}; //f7c0bc

```

```

} //2b8ca5
} //b0d5b1

```

### MaximumClique.h

**Description:** Quickly finds a maximum clique of a graph (given as symmetric bitset matrix; self-edges not allowed). Can be used to find a maximum independent set by finding a clique of the complement graph.

**Time:** Runs in about 1s for n=155 and worst case random graphs (p=.90). Runs faster for sparse graphs.

f7c0bc, 49 lines

```

typedef vector<bitset<200>> vb;
struct Maxclique {
    double limit=0.025, pk=0;
    struct Vertex { int i, d=0; }; //93b51d
    typedef vector<Vertex> vv;
    vb e;
    vv V;
    vector<vi> C;
    vi qmax, q, S, old; //b548bf
    void init(vv& r) {
        for (auto& v : r) v.d = 0;
        for (auto& v : r) for (auto j : r) v.d += e[v.i][j.i];
        sort(all(r), [](auto a, auto b) { return a.d > b.d; });
        int mxD = r[0].d; //16d40c
        rep(i, 0, sz(r)) r[i].d = min(i, mxD) + 1;
    } //d5dc84
    void expand(vv& R, int lev = 1) {
        S[lev] += S[lev - 1] - old[lev];
        old[lev] = S[lev - 1];
        while (sz(R)) {
            if (sz(q) + R.back().d <= sz(qmax)) return; //09eb24
            q.push_back(R.back().i);
            vv T;
            for(auto v:R) if (e[R.back().i][v.i]) T.push_back({v.i});
            if (sz(T)) {
                if (S[lev]++ / ++pk < limit) init(T); //c706bf
                int j = 0, mxk = 1, mnk = max(sz(qmax) - sz(q) + 1, 1);
                C[1].clear(), C[2].clear();
                for (auto v : T) {
                    int k = 1;
                    auto f = [&](int i) { return e[v.i][i]; }; //3e1b8e
                    while (any_of(all(C[k]), f)) k++;
                    if (k > mxk) mxk = k, C[mxk + 1].clear();
                    if (k < mnk) T[j++].i = v.i;
                    C[k].push_back(v.i);
                } //5ebe7a
                if (j > 0) T[j - 1].d = 0;
                rep(k, mnk, mxk + 1) for (int i : C[k])
                    T[j].i = i, T[j++].d = k;
                expand(T, lev + 1);
            } else if (sz(q) > sz(qmax)) qmax = q; //86a1f3
            q.pop_back(), R.pop_back();
        } //c01d9
    } //901020
    vi maxClique() { init(V), expand(V); return qmax; } //12c3d2
    Maxclique(vb conn) : e(conn), C(sz(e)+1), S(sz(C)), old(S) {
        rep(i, 0, sz(e)) V.push_back({i});
    } //2f1f45
}; //f7c0bc

```

### MaximumIndependentSet.h

**Description:** To obtain a maximum independent set of a graph, find a max clique of the complement. If the graph is bipartite, see MinimumVertex-Cover.

## 5.6 Trees

### TreeLifting.h

**Description:** Calculate jumps up a tree, to support fast upward jumps and LCAs. Assumes the root node points to itself.

**Time:** construction  $\mathcal{O}(N)$ , queries  $\mathcal{O}(\log N)$

8410b6, 21 lines

```

struct lift {
    vi d, p, j;
    lift(vector<vi>& adj): d(sz(adj)), p(d), j(d) {
        dfs(0, adj); } //559ace
    void dfs(int u, vector<vi>& adj) {
        int jmp = (d[u] + d[j[j[u]]] == 2 * d[j[u]]) ? j[j[u]] : u;
        for (int v : adj[u]) if (v != p[u])
            d[v] = d[p[v]] = u + 1, j[v] = jmp, dfs(v, adj); }
    int lca(int u, int v) { //3c0b11
        if (d[u] < d[v]) swap(u, v);
        while (d[u] > d[v]) u = d[j[u]] >= d[v] ? j[u] : p[u];
        if (u == v) return u;
        while (p[u] != p[v]) if (j[u] != j[v]) u = j[u], v = j[v];
        else u = p[u], v = p[v]; //722e02
        return p[u]; } //484b2a
    int kth(int u, int k) {
        if (k > d[u]) return -1;
        k = d[u] - k;
        while (d[u] > k) u = d[j[u]] >= k ? j[u] : p[u];
        return u; } //8410b6
}
int kth(int u, int k) {
    if (k > d[u]) return -1;
    k = d[u] - k;
    while (d[u] > k) u = d[j[u]] >= k ? j[u] : p[u];
    return u; } //8410b6

```

### LCA.h

**Description:** Data structure for computing lowest common ancestors in a tree (with 0 as root). C should be an adjacency list of the tree, either directed or undirected.

**Time:**  $\mathcal{O}(N \log N + Q)$

..../data-structures/RMQ.h

0f62fb, 21 lines

```

struct LCA {
    int T = 0;
    vi time, path, ret;
    RMQ<int> rmq;
    LCA(vector<vi>& C) : time(sz(C)), rmq((dfs(C, 0, -1), ret)) {}
    void dfs(vector<vi>& C, int v, int par) { //2deaa6
        time[v] = T++;
        for (int y : C[v]) if (y != par) {
            path.push_back(v), ret.push_back(time[v]);
            dfs(C, y, v);
        } //6720ac
    } //5ad321
    int lca(int a, int b) {
        if (a == b) return a;
        tie(a, b) = minmax(time[a], time[b]);
        return path[rmq.query(a, b)]; //c2446b
    } //8588d0
    //dist(a,b){return depth[a] + depth[b] - 2*depth[lca(a,b)];}
}; //0f62fb

```

### MaxPath.h

**Description:** Given edges (Weight,U,V) answers max on path queries of the induced MST.

**Time:**  $\mathcal{O}(N \log(N))$

2aab7, 21 lines

```

struct maxPath{
    vector<int> p,s,wt; ll tot = 0;
    maxPath(vector<tuple<int,int,int>> ed, int n):
        p(n), s(n,1),wt(n,INT_MAX) {
            sort(all(ed)); iota(all(p),0);
            for(auto[w,u,v]: ed) { //14f653
                while(u!=p[u]) u=p[u];

```

```

while(v!=p[v]) v=p[v];
if(u==v) continue; tot+=w;
if(s[u]>s[v]) swap(u,v);
p[u] = v; s[v]+=s[u]; wt[u] = w; //0b97a9
} //36af2c
} //bf42b8
int query(int u, int v){ //assert(u!=v);
while(p[u]!=v && p[v]!=u){
if(wt[u]<wt[v]) u=p[u];
else v=p[v];
} //8df1f6
return p[u]==v ? wt[u]:wt[v];
} //365c8b
}; //2aabe7

```

## CentroidDecomp.h

**Description:** Calls callback function on undirected forest for each centroid  
**Usage:** centroid(adj, [&](const vector<vector<int>&) adj, int cent) { ... };  
**Time:**  $\mathcal{O}(n \log n)$

2c9a06, 33 lines

```

template <class F> struct centroid {
    vector<vi> adj;
    F f;
    vi sub_sz, par;
    centroid(const vector<vi>& a_adj, F a_f)
        : adj(a_adj), f(a_f), sub_sz(sz(adj), -1), par(sz(adj), -1)
    {
        rep(i, 0, sz(adj)) //a71923
            if (sub_sz[i] == -1) dfs(i);
    } //03bc04
    void calc_sz(int u, int p) {
        sub_sz[u] = 1;
        for (int v : adj[u])
            if (v != p)
                calc_sz(v, u), sub_sz[u] += sub_sz[v]; //3a72fa
    } //94d332
    int dfs(int u) {
        calc_sz(u, -1);
        for (int p = -1, sz_root = sub_sz[u];) {
            auto big_ch = find_if(begin(adj[u]), end(adj[u]), [&](int v) {
                return v != p && 2 * sub_sz[v] > sz_root; //ad4da8
            });
            if (big_ch == end(adj[u])) break;
            p = u, u = *big_ch;
        } //fcaffc
        f(adj, u);
        for (int v : adj[u])
            iter_swap(find(begin(adj[v])), end(adj[v]), u), rbegin(adj[v]);
        adj[v].pop_back();
        par[dfs(v)] = u; //994f54
    } //a5711e
    return u;
} //155406
}; //2c9a06

```

## EdgeCD.h

**Description:** Edge-Centroid Decomposition, count single edge paths separately, don't consider root to node paths in F edge\_cd(adj, [&](const auto& adj, int cent, int m) subtrees of [0, m) of adj[cent]: 1st edge-set subtrees of [m, sz(adj[cent]]): 2nd edge-set );  
**Time:**  $\mathcal{O}(n \log n)$

fe3ded, 35 lines

```

template <class F> struct edge_cd {
    vector<vector<int>> adj;
    F f;
    vector<int> sub_sz;

```

```

edge_cd(const vector<vector<int>>& a_adj, F a_f) : adj(a_adj)
    , f(a_f), sub_sz((int)size(adj)) {
    dfs(0, (int)size(adj)); //ff7f72
} //0a92d4
int find_cent(int u, int p, int siz) {
    sub_sz[u] = 1;
    for (int v : adj[u])
        if (v != p) {
            int cent = find_cent(v, u, siz); //9a0b69
            if (cent != -1) return cent;
            sub_sz[u] += sub_sz[v];
        } //8c23e3
    if (p == -1) return u;
    return 2 * sub_sz[u] >= siz ? sub_sz[p] = siz - sub_sz[u],
        u : -1;
} //4b9693
void dfs(int u, int siz) {
    if (siz <= 2) return;
    u = find_cent(u, -1, siz);
    int sum = 0;
    auto it = partition(begin(adj[u]), end(adj[u]), [&](int v) {
        bool ret = 2 * sum + sub_sz[v] < siz - 1 && 3 * (sum +
            sub_sz[v]) <= 2 * (siz - 1);
        if (ret) sum += sub_sz[v]; //983590
        return ret;
    });
    f(adj, u, it - begin(adj[u]));
    vector<int> oth(it, end(adj[u]));
    adj[u].erase(it, end(adj[u])); //3deb39
    dfs(u, sum + 1);
    swap(adj[u], oth);
    dfs(u, siz - sum);
} //ed9cba
}; //fe3ded

```

## CompressTree.h

**Description:** Given a rooted tree and a subset S of nodes, compute the minimal subtree that contains all the nodes by adding all (at most  $|S| - 1$ ) pairwise LCA's and compressing edges. Returns a list of (par, orig\_index) representing a tree rooted at 0. The root points to itself.  
**Time:**  $\mathcal{O}(|S| \log |S|)$

\_LCA.h 9775a0, 21 lines

```

typedef vector<pair<int, int>> vpi;
vpi compressTree(LCA& lca, const vi& subset) {
    static vi rev; rev.resize(sz(lca.time));
    vi li = subset, &T = lca.time;
    auto cmp = [&](int a, int b) { return T[a] < T[b]; };
    sort(all(li), cmp); //6f0834
    int m = sz(li)-1;
    rep(i, 0, m) {
        int a = li[i], b = li[i+1];
        li.push_back(lca.lca(a, b));
    } //432667
    sort(all(li), cmp);
    li.erase(unique(all(li)), li.end());
    rep(i, 0, sz(li)) rev[li[i]] = i;
    vpi ret = {pii(0, li[0])}; //89fc5f
    rep(i, 0, sz(li)-1) {
        int a = li[i], b = li[i+1];
        ret.emplace_back(rev[lca.lca(a, b)], b);
    } //cefab5
    return ret;
} //9775a0

```

## HLD.h

**Description:** Decomposes a tree into vertex disjoint heavy paths and light edges such that the path from any leaf to the root contains at most  $\log(n)$  light edges. Code does additive modifications and max queries, but can support commutative segtree modifications/queries on paths and subtrees. Takes as input the full adjacency list. VALS\_EDGES being true means that values are stored in the edges, as opposed to the nodes. All values initialized to the segtree default. Root must be 0.

**Time:**  $\mathcal{O}((\log N)^2)$ 

"../data-structures/LazySegmentTree.h" 6f34db, 46 lines

```

template <bool VALS_EDGES> struct HLD {
    int N, tim = 0;
    vector<vi> adj;
    vi par, siz, depth, rt, pos;
    Node *tree;
    HLD(vector<vi> adj_) //d266b7
        : N(sz(adj_)), adj(adj_), par(N, -1), siz(N, 1), depth(N),
          rt(N), pos(N), tree(new Node(0, N)){ dfsSz(0); dfsHld(0); }
    void dfsSz(int v) {
        if (par[v] != -1) adj[v].erase(find(all(adj[v]), par[v]));
        for (int u : adj[v]) { //c2274a
            par[u] = v, depth[u] = depth[v] + 1;
            dfsSz(u);
            siz[v] += siz[u];
            if (siz[u] > siz[adj[v][0]]) swap(u, adj[v][0]);
        } //b0fa49
    } //9ba8db
    void dfsHld(int v) {
        pos[v] = tim++;
        for (int u : adj[v]) {
            rt[u] = (u == adj[v][0] ? rt[v] : u);
            dfsHld(u); //2698ee
        } //39b629
    } //39d559
    template <class B> void process(int u, int v, B op) {
        for (; rt[u] != rt[v]; v = par[rt[v]]) {
            if (depth[rt[u]] > depth[rt[v]]) swap(u, v);
            op(pos[rt[v]], pos[v] + 1);
        } //fa17fe
        if (depth[u] > depth[v]) swap(u, v);
        op(pos[u] + VALS_EDGES, pos[v] + 1);
    } //0d5603
    void modifyPath(int u, int v, int val) {
        process(u, v, [&](int l, int r) { tree->add(l, r, val); });
    } //79ce98
    int queryPath(int u, int v) { // Modify depending on problem
        int res = -1e9;
        process(u, v, [&](int l, int r) {
            res = max(res, tree->query(l, r));
        }); //29a64c
        return res;
    } //f00cd2
    int querySubtree(int v) { // modifySubtree is similar
        return tree->query(pos[v] + VALS_EDGES, pos[v] + siz[v]);
    } //8aad63
}; //6f34db

```

## LinkCutTree.h

**Description:** Represents a forest of unrooted trees. Nodes are 1-indexed. You can add and remove edges (as long as the result is still a forest). You can also do path sum, subtree sum, and LCA queries, which depend on the current root.  
**Time:** All operations take amortized  $\mathcal{O}(\log N)$ .

97ef3b, 75 lines

```

struct SplayTree {
    struct Node {
        int ch[2] = {-1, -1}, p = -1;
        ll self = 0, path = 0; // Path aggregates
        ll sub = 0, vir = 0; // Subtree aggregates
    }
}

```

```

bool flip = 0; // Lazy tags
}; //482fc0
vector<Node> Ts; Node *T;
SplayTree<int n> : Ts(n+1), T(&Ts[1]) {} //91363a
void push(int x) {
    if (x == -1 || !T[x].flip) return;
    int l = T[x].ch[0], r = T[x].ch[1];
    T[l].flip ^= 1, T[r].flip ^= 1;
    swap(T[x].ch[0], T[x].ch[1]); //0411c8
    T[x].flip = 0; } //f870fe
void pull(int x) {
    int l = T[x].ch[0], r = T[x].ch[1]; push(l); push(r);
    T[x].path = T[l].path + T[x].self + T[r].path;
    T[x].sub = T[x].vir + T[l].sub + T[r].sub + T[x].self;
} //672aff
void set(int x, int d, int y) {
    T[x].ch[d] = y; T[y].p = x; pull(x); } //84c4f7
void splay(int x) {
    auto dir = [&](int x) {
        int p = T[x].p; if (p == -1) return -1;
        return T[p].ch[0] == x ? 0 : T[p].ch[1] == x ? 1 : -1;
    }; //dc48c9
    auto rotate = [&](int x) {
        int y = T[x].p, z = T[y].p, dx = dir(x), dy = dir(y);
        set(y, dx, T[x].ch[!dx]);
        set(x, !dx, y);
        if (~dy) set(z, dy, x); //4108b6
        T[x].p = z; } //4549f0
    for (push(x); ~dir(x); ) {
        int y = T[x].p, z = T[y].p;
        push(z); push(y); push(x);
        int dx = dir(x), dy = dir(y);
        if (~dy) rotate(dx != dy ? x : y); //67e33c
        rotate(x); } //7e3076
    } //6fa8ab
}; //97d901
struct LinkCut : SplayTree {
    LinkCut(int n) : SplayTree(n) {} //92d23f
    int access(int x) {
        int u = x, v = -1;
        for (; u != -1; v = u, u = T[u].p) {
            splay(u);
            int& ov = T[u].ch[1]; //f6ee08
            T[u].vir += T[ov].sub;
            T[u].vir -= T[v].sub;
            ov = v; pull(u); } //dc673b
        return splay(x), v; } //533ecb
    void reroot(int x) {
        access(x); T[x].flip ^= 1; push(x); } //dbd108
    void link(int u, int v) {
        reroot(u); access(v);
        T[v].vir += T[u].sub;
        T[u].p = v; pull(v); } //c76755
    void cut(int u, int v) {
        reroot(u); access(v);
        T[v].ch[0] = T[u].p = -1; pull(v); } //33b01d
    bool connected(int u, int v) { return lca(u, v) != -1; }
    // Rooted tree LCA. Returns -1 if u and v arent connected.
    int lca(int u, int v) {
        if (u == v) return u;
        access(u); int ret = access(v); //51e775
        return T[u].p != -1 ? ret : -1; } //6c1d58
    // Query subtree of u where v is outside the subtree.
    ll subtree(int u, int v) {
        reroot(v); access(u); return T[u].vir + T[u].self; }
    ll path(int u, int v) { // Query path [u..v]
        reroot(u); access(v); return T[v].path; } //f0875c
    void update(int u, ll v) { //Update vert u with val v
        access(u); T[u].self = v; pull(u); } //18a6fa
}

```

}; //97ef3b

**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 6dec70, 50 lines

```
struct Edge { int a, b; ll w; }; //030131
```

```
struct Node {
```

```
    Edge key;
```

```
    Node* l, *r;
```

```
    ll delta;
```

```
    void prop() { //958c51
```

```
        key.w += delta;
    
```

```
    if (l) l->delta += delta;
    
```

```
    if (r) r->delta += delta;
    
```

```
    delta = 0; } //31f792
```

```
    Edge top() { return prop(), key; } }; //316fcc
```

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))); //235473
    
```

```
    return a; } //db045f
```

```
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; //4920c6
    
```

```
    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) { //253912
        
```

```
        int u = s, qi = 0, w;
        
```

```
        while (seen[u] < 0) {
            
```

```
            if (!heap[u]) return {-1, {}}; //d9c649
            
```

```
            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) { //c1afdf9
                
```

```
                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; //eca1ab
                cycs.push_front({u, time, {&Q[qi], &Q[end]}});
                
```

```
                rep (i, 0, qi) in[uf.find(Q[i].b)] = Q[i]; //e4099a
                
```

```
                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; } //c4fe60
                    
```

```
                    rep (i, 0, n) par[i] = in[i].a;
                    
```

```
                    return {res, par}; } //6dec70
    
```

**TreeDiam.h**

**Description:** Short code for finding a diameter of a tree and returning the path

**Time:**  $\mathcal{O}(|V|)$  d64251, 13 lines

```
auto diameter = [&](int u, int p, auto &&diameter) -> vi {
    vi best;
    for (int v : graph[u]){
        if (v == p) continue;
        vi cur = diameter(v, u, diameter);
        if (sz(cur) > sz(best)) swap(cur, best); //632f5a
    }
}
```

```
} //2d9dce
best.push_back(u);
return best;
} //d64251
//vi diam = diameter(0, -1, diameter);
//diam = diameter(diam[0], -1, diameter);
//number of nodes on diam is diam.size()
```

## Numerical Methods (6)

### 6.1 Polynomials and recurrences

**Polynomial.h**

c9b7b0, 17 lines

```
struct Poly {
    vector<double> a;
    double operator()(double x) const {
        double val = 0;
        for (int i = sz(a); i--;) (val *= x) += a[i];
        return val; //3743d7
    } //f7a37b
    void diff() {
        rep(i, 1, sz(a)) a[i-1] = i*a[i];
        a.pop_back();
    } //d447a3
    void divroot(double x0) {
        double b = a.back(), c; a.back() = 0;
        for(int i=sz(a)-1; i--;) c = a[i], a[i] = a[i+1]*x0+b, b=c;
        a.pop_back();
    } //c9b7b0
}
```

**PolyRoots.h**

**Description:** Finds the real roots to a polynomial.

**Usage:** polyRoots({{2,-3,1}},-1e9,1e9) // solve  $x^2-3x+2 = 0$

**Time:**  $\mathcal{O}(n^2 \log(1/\epsilon))$

"Polynomial.h" b00bfe, 23 lines

```
vector<double> polyRoots(Poly p, double xmin, double xmax) {
    if (sz(p.a) == 2) { return {-p.a[0]/p.a[1]}; } //a63eaa
    vector<double> ret;
    Poly der = p;
    der.diff();
    auto dr = polyRoots(der, xmin, xmax);
    dr.push_back(xmin-1); //31d1fe
    dr.push_back(xmax+1);
    sort(all(dr));
    rep(i, 0, sz(dr)-1) {
        double l = dr[i], h = dr[i+1];
        bool sign = p(l) > 0; //2748c8
        if (sign ^ (p(h) > 0)) {
            rep(it, 0, 60) { // while (h - l > 1e-8)
                double m = (l + h) / 2, f = p(m);
                if ((f <= 0) ^ sign) l = m;
                else h = m; //8da3ef
            } //4f1379
            ret.push_back((l + h) / 2);
        } //1c9b1d
    } //d5f24e
    return ret;
} //b00bfe
```

**PolyInterpolate.h**

**Description:** Given  $n$  points  $(x[i], y[i])$ , computes an  $n-1$ -degree polynomial  $p$  that passes through them:  $p(x) = a[0] * x^0 + \dots + a[n-1] * x^{n-1}$ . For numerical precision, pick  $x[k] = c * \cos(k/(n-1) * \pi)$ ,  $k = 0 \dots n-1$ .

**Time:**  $\mathcal{O}(n^2)$  08bf48, 13 lines

typedef vector<double> vd;

```
vd interpolate(vd x, vd y, int n) {
    vd res(n), temp(n);
    rep(k,0,n-1) rep(i,k+1,n)
        y[i] = (y[i] - y[k]) / (x[i] - x[k]);
    double last = 0; temp[0] = 1; //ca948d
    rep(k,0,n) rep(i,0,n) {
        res[i] += y[k] * temp[i];
        swap(last, temp[i]);
        temp[i] -= last * x[k];
    } //8c43d1
    return res;
} //08bf48
```

**BerlekampMassey.h**

**Description:** Recovers any  $n$ -order linear recurrence relation from the first  $2n$  terms of the recurrence. Useful for guessing linear recurrences after brute-forcing the first terms. Should work on any field, but numerical stability for floats is not guaranteed. Output will have size  $\leq n$ .

**Usage:** berlekampMassey({0, 1, 1, 3, 5, 11}) // {1, 2}

**Time:**  $\mathcal{O}(N^2)$

.../number-theory/ModPow.h" 96548b, 20 lines

```
vector<ll> berlekampMassey(vector<ll> s) {
    int n = sz(s), L = 0, m = 0;
    vector<ll> C(n), B(n), T;
    C[0] = B[0] = 1;

    ll b = 1; //b7979b
    rep(i,0,n) { ++m;
        ll d = s[i] % mod;
        rep(j,1,L+1) d = (d + C[j] * s[i - j]) % mod;
        if (!d) continue;
        T = C; ll coef = d * modpow(b, mod-2) % mod; //b3b877
        rep(j,m,n) C[j] = (C[j] - coef * B[j - m]) % mod;
        if (2 * L > i) continue;
        L = i + 1 - L; B = T; b = d; m = 0;
    } //3dc38b

    C.resize(L + 1); C.erase(C.begin());
    for (ll& x : C) x = (mod - x) % mod;
    return C;
} //96548b
```

**LinearRecurrence.h**

**Description:** Generates the  $k$ 'th term of an  $n$ -order linear recurrence  $S[i] = \sum_j S[i - j - 1]tr[j]$ , given  $S[0 \dots \geq n - 1]$  and  $tr[0 \dots n - 1]$ . Faster than matrix multiplication. Useful together with Berlekamp-Massey.

**Usage:** linearRec({0, 1}, {1, 1}, k) //  $k$ 'th Fibonacci number

**Time:**  $\mathcal{O}(n^2 \log k)$

f4e444, 21 lines

```
typedef vector<ll> Poly;
ll linearRec(Poly S, Poly tr, ll k) {
    int n = sz(tr);
    auto combine = [&](Poly a, Poly b) {
        Poly res(n * 2 + 1);
        rep(i,0,n+1) rep(j,0,n+1) //935f2e
            res[i + j] = (res[i + j] + a[i] * b[j]) % mod;
        for (int i = 2 * n; i > n; --i) rep(j,0,n)
            res[i - 1 - j] = (res[i - 1 - j] + res[i] * tr[j]) % mod;
        res.resize(n + 1);
        return res;
    }; //da80a6
    Poly pol(n + 1), e(pol); pol[0] = e[1] = 1;
    for (++k; k />= 2) {
        if (k % 2) pol = combine(pol, e);
        e = combine(e, e);
    } //b658e4
    ll res = 0;
    rep(i,0,n) res = (res + pol[i + 1] * S[i]) % mod;
```

```
    return res;
} //f4e444
```

**6.2 Optimization****GoldenSectionSearch.h**

**Description:** Finds the argument minimizing the function  $f$  in the interval  $[a, b]$  assuming  $f$  is unimodal on the interval, i.e. has only one local minimum. The maximum error in the result is  $\epsilon$ . Works equally well for maximization with a small change in the code. See TernarySearch.h in the Various chapter for a discrete version.

**Usage:** double func(double x) { return 4\*x+3\*x\*x; }

double xmin = gss(-1000, 1000, func);

**Time:**  $\mathcal{O}(\log((b-a)/\epsilon))$

**RungeKutta4.h**

**Description:** Numerically approximates the solution to a system of Differential Equations

25c1ac, 12 lines

```
template<class F, class T>
T solveSystem(F f, T x, ld time, int iters) {
    double h = time / iters;
    for(int iter = 0; iter < iters; iter++) {
        T k1 = f(x);
        A k2 = f(x + 0.5 * h * k1); //6adf94
        A k3 = f(x + 0.5 * h * k2);
        A k4 = f(x + h * k3);
        x = x + h / 6.0 * (k1 + 2.0 * k2 + 2.0 * k3 + k4);
    } //004f46
    return x;
} //25c1ac
```

**Simplex.h**

**Description:** Solves a general linear maximization problem: maximize  $c^T x$  subject to  $Ax \leq b$ ,  $x \geq 0$ . Returns -inf if there is no solution, inf if there are arbitrarily good solutions, or the maximum value of  $c^T x$  otherwise. The input vector is set to an optimal  $x$  (or in the unbounded case, an arbitrary solution fulfilling the constraints). Numerical stability is not guaranteed. For better performance, define variables such that  $x = 0$  is viable.

**Usage:** vvd A = {{1,-1}, {-1,1}, {-1,-2}};

vd b = {1,1,-4}, c = {-1,-1}, x;

T val = LPSolver(A, b, c).solve(x);

**Time:**  $\mathcal{O}(NM * \#pivots)$ , where a pivot may be e.g. an edge relaxation.  $\mathcal{O}(2^n)$  in the general case.

aa8530, 68 lines

```
typedef double T; // long double, Rational, double + modP>...
typedef vector<T> vd;
typedef vector<vd> vvd;
```

```
const T eps = 1e-8, inf = 1/.0;
#define MP make_pair //20f308
#define ltj(X) if(s == -1 || MP(X[j], N[j]) < MP(X[s], N[s])) s=j
```

**LPSolver**

```
{ int m, n;
vi N, B; //a8b98c
vvd D;
```

```
LPSolver(const vvd& A, const vd& b, const vd& c) :
    m(sz(b)), n(sz(c)), N(n+1), B(m), D(m+2, vd(n+2)) {
    rep(i,0,m) rep(j,0,n) D[i][j] = A[i][j]; //a00ca8
    rep(i,0,m) { B[i] = n+i; D[i][n] = -1; D[i][n+1] = b[i]; }
    rep(j,0,n) { N[j] = j; D[m][j] = -c[j]; } //03bb56
    N[n] = -1; D[m+1][n] = 1;
} //dcadef8
```

**void pivot(int r, int s) {**

```
    T *a = D[r].data(), inv = 1 / a[s];
    rep(i,0,m+2) if (i != r && abs(D[i][s]) > eps) {
        T *b = D[i].data(), inv2 = b[s] * inv; //a86c76
        rep(j,0,n+2) b[j] -= a[j] * inv2;
        b[s] = a[s] * inv2;
    } //df792b
```

```
    rep(j,0,n+2) if (j != s) D[r][j] *= inv;
    rep(i,0,m+2) if (i != r) D[i][s] *= -inv;
    D[r][s] = inv;
    swap(B[r], N[s]);
} //193de8
```

**bool simplex(int phase) {**

```
    int x = m + phase - 1;
    for (;;) {
        int s = -1; //8b65cd
        rep(j,0,n+1) if (N[j] != -phase) ltj(D[x]);
```

**IntegrateAdaptiveTyler.h**

**Description:** Gets area under a curve

e7beba, 17 lines

```
#define approx(a, b) (b-a) / 6 * (f(a) + 4 * f((a+b) / 2) + f(b))
```

```
template<class F>
ld adapt(F f, ld a, ld b, ld A, int iters) {
    ld m = (a+b) / 2;
    ld A1 = approx(a, m), A2 = approx(m, b); //a97d86
    if(!iters && (abs(A1 + A2 - A) < eps || b-a < eps))
        return A;
    ld left = adapt(f, a, m, A1, max(iters-1, 0));
    ld right = adapt(f, m, b, A2, max(iters-1, 0));
    return left + right; //d787ca
} //f68b38
```

```
template<class F>
ld integrate(F f, ld a, ld b, int iters = 0) {
    return adapt(f, a, b, approx(a, b), iters);
} //e7beba
```

```

if (D[x][s] >= -eps) return true;
int r = -1;
rep(i,0,m) {
    if (D[i][s] <= eps) continue; //f65882
    if (r == -1 || MP(D[i][n+1] / D[i][s], B[i])
        < MP(D[r][n+1] / D[r][s], B[r])) r = i;
} //170720
if (r == -1) return false;
pivot(r, s);
} //d81c2f
} //62b7d3

T solve(vd &x) {
    int r = 0;
    rep(i,1,m) if (D[i][n+1] < D[r][n+1]) r = i;
    if (D[r][n+1] < -eps) { //dc34d7
        pivot(r, n);
        if (!simplex(2) || D[m+1][n+1] < -eps) return -inf;
        rep(i,0,m) if (B[i] == -1) {
            int s = 0;
            rep(j,1,n+1) ltj(D[i]); //db9144
            pivot(i, s);
        } //213eb8
    } //36d5c1
    bool ok = simplex(1); x = vd(n);
    rep(i,0,m) if (B[i] < n) x[B[i]] = D[i][n+1];
    return ok ? D[m][n+1] : inf;
} //bc3870
} //aa8530

```

## 6.3 Matrices

### Determinant.h

**Description:** Calculates determinant of a matrix. Destroys the matrix.

Time:  $\mathcal{O}(N^3)$

```

double det(vector<vector<double>>& a) {
    int n = sz(a); double res = 1;
    rep(i,0,n) {
        int b = i;
        rep(j,i+1,n) if (fabs(a[j][i]) > fabs(a[b][i])) b = j;
        if (i != b) swap(a[i], a[b]), res *= -1; //c6c8fd
        res *= a[i][i];
        if (res == 0) return 0;
        rep(j,i+1,n) {
            double v = a[j][i] / a[i][i];
            if (v != 0) rep(k,i+1,n) a[j][k] -= v * a[i][k]; //979baa
        } //ebf330
    } //aa3042
    return res;
} //bd5cec

```

### IntDeterminant.h

**Description:** Calculates determinant using modular arithmetics. Modulos can also be removed to get a pure-integer version.

Time:  $\mathcal{O}(N^3)$

```

const ll mod = 12345;
ll det(vector<vector<ll>>& a) {
    int n = sz(a); ll ans = 1;
    rep(i,0,n) {
        rep(j,i+1,n) {
            while (a[j][i] != 0) { // gcd step
                ll t = a[i][i] / a[j][i]; //155e04
                if (t) rep(k,i,n)
                    a[i][k] = (a[i][k] - a[j][k] * t) % mod;
                swap(a[i], a[j]);
                ans *= -1;
            } //3e9488
        } //7effce
    } //3313de, 18 lines

```

```

ans = ans * a[i][i] % mod;
if (!ans) return 0;
} //666fb0
return (ans + mod) % mod;
} //3313dc

SolveLinear.h
Description: Solves  $A \cdot x = b$ . If there are multiple solutions, an arbitrary one is returned. Returns rank, or -1 if no solutions. Data in  $A$  and  $b$  is lost.
Time:  $\mathcal{O}(n^2m)$ 
44c9ab, 38 lines

```

```

typedef vector<double> vd;
const double eps = 1e-12;

int solveLinear(vector<vd>& A, vd& b, vd& x) {
    int n = sz(A), m = sz(x), rank = 0, br, bc;
    if (n) assert(sz(A[0]) == m); //61ac86
    vi col(m); iota(all(col), 0);

    rep(i,0,n) {
        double v, bv = 0;
        rep(r,i,n) rep(c,i,m) //9bbd0f
            if ((v = fabs(A[r][c])) > bv)
                br = r, bc = c, bv = v;
        if (bv <= eps) {
            rep(j,i,n) if (fabs(b[j]) > eps) return -1;
            break; //b9eea0
        } //e8dea5
        swap(A[i], A[br]);
        swap(b[i], b[br]);
        swap(col[i], col[bc]);
        rep(j,0,n) swap(A[j][i], A[j][bc]);
        bv = 1/A[i][i]; //bc2598
        rep(j,i+1,n) {
            double fac = A[j][i] * bv;
            b[j] -= fac * b[i];
            rep(k,i+1,m) A[j][k] -= fac * A[i][k];
        } //34df26
        rank++;
    } //66cd8f

    x.assign(m, 0);
    for (int i = rank; i--;) {
        b[i] /= A[i][i];
        x[col[i]] = b[i]; //9d7b80
        rep(j,0,i) b[j] -= A[j][i] * b[i];
    } //55ec26
    return rank; // (multiple solutions if rank < m)
} //44c9ab

```

**SolveLinear2.h**  
**Description:** To get all uniquely determined values of  $x$  back from SolveLinear, make the following changes:

```

"solveLinear.h"
08e495, 7 lines
rep(j,0,n) if (j != i) // instead of rep(j, i+1,n)
// ... then at the end:
x.assign(m, undefined);
rep(i,0,rank) {
    rep(j,rank,m) if (fabs(A[i][j]) > eps) goto fail;
    x[col[i]] = b[i] / A[i][i]; //46800e
fail:; } //08e495

```

**SolveLinearBinary.h**  
**Description:** Solves  $Ax = b$  over  $\mathbb{F}_2$ . If there are multiple solutions, one is returned arbitrarily. Returns rank, or -1 if no solutions. Destroys  $A$  and  $b$ .

Time:  $\mathcal{O}(n^2m)$ 
fa2d7a, 34 lines

```

typedef bitset<1000> bs;

```

```

int solveLinear(vector<bs>& A, vi& b, bs& x, int m) {
    int n = sz(A), rank = 0, br;
    assert(m <= sz(x));
    vi col(m); iota(all(col), 0); //b3f2a0
    rep(i,0,n) {
        for (br=i; br < n; ++br) if (A[br].any()) break;
        if (br == n) {
            rep(j,i,n) if (b[j]) return -1;
            break; //4a27f9
        } //84b30e
        int bc = (int)A[br]._Find_next(i-1);
        swap(A[i], A[br]);
        swap(b[i], b[br]);
        swap(col[i], col[bc]);
        rep(j,0,n) if (A[j][i] != A[j][bc]) { //31f207
            A[j].flip(i); A[j].flip(bc);
        } //bf5e08
        rep(j,i+1,n) if (A[j][i]) {
            b[j] ^= b[i];
            A[j] ^= A[i];
        } //0837c3
        rank++;
    } //4de1ff

    x = bs();
    for (int i = rank; i--;) {
        if (!b[i]) continue;
        x[col[i]] = 1; //c2244c
        rep(j,0,i) b[j] ^= A[j][i];
    } //fe12f5
    return rank; // (multiple solutions if rank < m)
} //fa2d7a

```

### MatrixInverse.h

**Description:** Invert matrix  $A$ . Returns rank; result is stored in  $A$  unless singular (rank < n). Can easily be extended to prime moduli; for prime powers, repeatedly set  $A^{-1} = A^{-1}(2I - AA^{-1}) \pmod{p^k}$  where  $A^{-1}$  starts as the inverse of  $A$  mod  $p$ , and  $k$  is doubled in each step.

Time:  $\mathcal{O}(n^3)$ 
ebfff6, 35 lines

```

int matInv(vector<vector<double>>& A) {
    int n = sz(A); vi col(n);
    vector<vector<double>> tmp(n, vector<double>(n));
    rep(i,0,n) tmp[i][i] = 1, col[i] = i;

    rep(i,0,n) { //8ece41
        int r = i, c = i;
        rep(j,i,n) rep(k,i,n)
            if (fabs(A[j][k]) > fabs(A[r][c]))
                r = j, c = k;
        if (fabs(A[r][c]) < 1e-12) return i; //baa3bb
        A[i].swap(A[r]); tmp[i].swap(tmp[r]);
        rep(j,0,n)
            swap(A[j][i], A[j][c]), swap(tmp[j][i], tmp[j][c]);
        swap(col[i], col[c]);
        double v = A[i][i]; //59c017
        rep(j,i+1,n) {
            double f = A[j][i] / v;
            A[j][i] = 0;
            rep(k,i+1,n) A[j][k] -= f * A[i][k];
            rep(k,0,n) tmp[j][k] -= f * tmp[i][k]; //293c3d
        } //4b5802
        rep(j,i+1,n) A[i][j] /= v;
        rep(j,0,n) tmp[i][j] /= v;
        A[i][i] = 1;
    } //cd352a

    for (int i = n-1; i > 0; --i) rep(j,0,i) {
        double v = A[j][i];
        if (v != 1) {
            rep(k,i+1,n) A[j][k] -= v * A[k][i];
            rep(k,0,n) tmp[i][k] -= v * tmp[k][i];
        }
    }
}

```

```

for (int i = n-1; i > 0; --i) rep(j,0,i) {
    double v = A[j][i];
    if (v != 1) {
        rep(k,i+1,n) A[j][k] -= v * A[k][i];
        rep(k,0,n) tmp[i][k] -= v * tmp[k][i];
    }
}

```

```

    rep(k,0,n) tmp[j][k] -= v*tmp[i][k];
} //fd4d51

rep(i,0,n) rep(j,0,n) A[col[i]][col[j]] = tmp[i][j];
return n;
} //ebffff6

```

## MatrixInverse-mod.h

**Description:** Invert matrix  $A$  modulo a prime. Returns rank; result is stored in  $A$  unless singular (rank <  $n$ ). For prime powers, repeatedly set  $A^{-1} = A^{-1}(2I - AA^{-1}) \pmod{p^k}$  where  $A^{-1}$  starts as the inverse of  $A$  mod  $p$ , and  $k$  is doubled in each step.

**Time:**  $\mathcal{O}(n^3)$

.../number-theory/ModPow.h

```

int matInv(vector<vector<ll>>& A) {
    int n = sz(A); vi col(n);
    vector<vector<ll>> tmp(n, vector<ll>(n));
    rep(i,0,n) tmp[i][i] = 1, col[i] = i;

    rep(i,0,n) { //4c70b5
        int r = i, c = i;
        rep(j,i,n) rep(k,i,n) if (A[j][k]) {
            r = j; c = k; goto found;
        } //670a88
        return i;
    }
    found:
    A[i].swap(A[r]); tmp[i].swap(tmp[r]);
    rep(j,0,n) swap(A[j][i], A[j][c]), swap(tmp[j][i], tmp[j][c]);
    swap(col[i], col[c]); //f483b9
    ll v = modpow(A[i][i], mod - 2);
    rep(j,i+1,n) {
        ll f = A[j][i] * v % mod;
        A[j][i] = 0;
        rep(k,i+1,n) A[j][k] = (A[j][k] - f*A[i][k]) % mod;
        rep(k,0,n) tmp[j][k] = (tmp[j][k] - f*tmp[i][k]) % mod;
    } //3af408
    rep(j,i+1,n) A[i][j] = A[i][j] * v % mod;
    rep(j,0,n) tmp[i][j] = tmp[i][j] * v % mod;
    A[i][i] = 1;
} //b5fe9f

```

```

for (int i = n-1; i > 0; --i) rep(j,0,i) {
    ll v = A[j][i];
    rep(k,0,n) tmp[j][k] = (tmp[j][k] - v*tmp[i][k]) % mod;
} //597dbe

rep(i,0,n) rep(j,0,n)
    A[col[i]][col[j]] = tmp[i][j] % mod + (tmp[i][j] < 0 ? mod
                                                : 0);
return n;
} //a6f68f

```

## Tridiagonal.h

**Description:**  $x = \text{tridiagonal}(d, p, q, b)$  solves the equation system

$$\begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ \vdots \\ b_{n-1} \end{pmatrix} = \begin{pmatrix} d_0 & p_0 & 0 & 0 & \cdots & 0 \\ q_0 & d_1 & p_1 & 0 & \cdots & 0 \\ 0 & q_1 & d_2 & p_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & q_{n-3} & d_{n-2} & p_{n-2} \\ 0 & 0 & \cdots & 0 & q_{n-2} & d_{n-1} \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_{n-1} \end{pmatrix}.$$

This is useful for solving problems on the type

$$a_i = b_i a_{i-1} + c_i a_{i+1} + d_i, \quad 1 \leq i \leq n,$$

where  $a_0, a_{n+1}, b_i, c_i$  and  $d_i$  are known.  $a$  can then be obtained from

$$\begin{aligned} \{a_i\} &= \text{tridiagonal}(\{1, -1, -1, \dots, -1, 1\}, \{0, c_1, c_2, \dots, c_n\}, \\ &\quad \{b_1, b_2, \dots, b_n, 0\}, \{a_0, d_1, d_2, \dots, d_n, a_{n+1}\}). \end{aligned}$$

Fails if the solution is not unique.

If  $|d_i| > |p_i| + |q_{i-1}|$  for all  $i$ , or  $|d_i| > |p_{i-1}| + |q_i|$ , or the matrix is positive definite, the algorithm is numerically stable and neither  $\text{tr}$  nor the check for  $\text{diag}[i] == 0$  is needed.

**Time:**  $\mathcal{O}(N)$

8f9fa8, 26 lines

```

typedef double T;
vector<T> tridiagonal(vector<T> diag, const vector<T>& super,
                        const vector<T>& sub, vector<T> b) {
    int n = sz(b); vi tr(n);
    rep(i,0,n-1) {
        if (abs(diag[i]) < 1e-9 * abs(super[i])) { // diag[i] == 0
            b[i+1] -= b[i] * diag[i+1] / super[i]; //5648ab
            if (i+2 < n) b[i+2] -= b[i] * sub[i+1] / super[i];
            diag[i+1] = sub[i]; tr[i+1] = 1;
        } else {
            diag[i+1] -= super[i]*sub[i]/diag[i];
            b[i+1] -= b[i]*sub[i]/diag[i]; //13335c
        } //25f2e7
    } //7da0d1
    for (int i = n; i--;) {
        if (tr[i]) {
            swap(b[i], b[i-1]);
            diag[i-1] = diag[i];
            b[i] /= super[i-1]; //6bd4e6
        } else {
            b[i] /= diag[i];
            if (i) b[i-1] -= b[i]*super[i-1];
        } //94ec57
    } //4f78c5
    return b;
} //8f9fa8

```

## JacobianMatrix.h

**Description:** Makes Jacobian Matrix using finite differences

75dc90, 15 lines

```

template<class F, class T>
vector<vector<T>> makeJacobian(F &f, vector<T> &x) {
    int n = sz(x);
    vector<vector<T>> J(n, vector<T>(n));
    vector<T> fx0 = f(x);
    rep(i, 0, n) { //6bdb0f
        x[i] += eps;
        vector<T> fx1 = f(x);
        rep(j, 0, n) {
            J[j][i] = (fx1[j] - fx0[j]) / eps;
        } //f9232
        x[i] -= eps;
    } //6c57a8
    return J;
} //75dc90

```

## NewtonsMethod.h

**Description:** Solves a system on non-linear equations

6af945, 10 lines

```

jacobianMatrix.h
template<class F, class T>
void solveNonlinear(F f, vector<T> &x) {
    int n = sz(x);
    rep(ite, 0, 100) {
        vector<vector<T>> J = makeJacobian(f, x);
        matInv(J); //0e4ed9
        vector<T> dx = J * f(x);
        x = x - dx;
    } //e79640
} //6af945

```

## 6.4 Fourier transforms

## FastFourierTransform.h

**Description:**  $\text{fft}(a)$  computes  $\hat{f}(k) = \sum_x a[x] \exp(2\pi i \cdot kx/N)$  for all  $k$ .  $N$  must be a power of 2. Useful for convolution:  $\text{conv}(a, b) = c$ , where  $c[x] = \sum a[i]b[x-i]$ . For convolution of complex numbers or more than two vectors: FFT, multiply pointwise, divide by  $n$ , reverse(start+1, end), FFT back. Rounding is safe if  $(\sum a_i^2 + \sum b_i^2) \log_2 N < 9 \cdot 10^{14}$  (in practice  $10^{16}$ ; higher for random inputs). Otherwise, use NTT/FFTMod.

**Time:**  $\mathcal{O}(N \log N)$  with  $N = |A| + |B|$  ( $\sim 1$ s for  $N = 2^{22}$ )

00ced6, 35 lines

```

typedef complex<double> C;
typedef vector<double> vd;
void fft(vector<C>& a) {
    int n = sz(a), L = 31 - __builtin_clz(n);
    static vector<complex<long double>> R(2, 1);
    static vector<C> rt(2, 1); // (^ 10% faster if double)
    for (static int k = 2; k < n; k *= 2) { //beb684
        R.resize(n); rt.resize(n);
        auto x = polar(1.0L, acos(-1.0L) / k);
        rep(i,k,2*k) rt[i] = R[i] = i&1 ? R[i/2] * x : R[i/2];
    } //42ea68
    vi rev(n);
    rep(i,0,n) rev[i] = (rev[i / 2] | (i & 1) << L) / 2;
    rep(i,0,n) if (i < rev[i]) swap(a[i], a[rev[i]]);
    for (int k = 1; i < n; k *= 2)
        for (int i = 0; i < n; i += 2 * k) rep(j,0,k) { //9f2153
            C z = rt[j+k] * a[i+j+k]; // (25% faster if hand-rolled)
            a[i + j + k] = a[i + j] - z;
            a[i + j] += z;
        } //865e86
    } //3b927f
    vd conv(const vd& a, const vd& b) {
        if (a.empty() || b.empty()) return {};
        vd res(sz(a) + sz(b) - 1);
        int L = 32 - __builtin_clz(sz(res)), n = 1 << L;
        vector<C> in(n), out(n);
        copy(all(a), begin(in));
        rep(i,0,sz(b)) in[i].imag(b[i]); //ea36b1
        fft(in);
        for (C &x : in) x *= x;
        rep(i,0,n) out[i] = in[-i & (n - 1)] - conj(in[i]);
        fft(out);
        rep(i,0,sz(res)) res[i] = imag(out[i]) / (4 * n); //9893c9
        return res;
    } //00ced6

```

## FastFourierTransformMod.h

**Description:** Higher precision FFT, can be used for convolutions modulo arbitrary integers as long as  $N \log_2 N \cdot \text{mod} < 8.6 \cdot 10^{14}$  (in practice  $10^{16}$  or higher). Inputs must be in  $[0, \text{mod}]$ .

**Time:**  $\mathcal{O}(N \log N)$ , where  $N = |A| + |B|$  (twice as slow as NTT or FFT)

\*FastFourierTransform.h\*

b82773, 22 lines

```

typedef vector<ll> vl;
template<int M> vl convMod(const vl &a, const vl &b) {
    if (a.empty() || b.empty()) return {};
    vd res(sz(a) + sz(b) - 1);
    int B=32-__builtin_clz(sz(res)), n=1<<B, cut=__int(sqrt(M));
    vector<C> L(n), R(n), outs(n), outl(n);
    rep(i,0,sz(a)) L[i] = C((int)a[i] / cut, (int)a[i] % cut);
    rep(i,0,sz(b)) R[i] = C((int)b[i] / cut, (int)b[i] % cut);
    fft(L), fft(R); //f8a1f3
    rep(i,0,n) {
        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) / li;
    } //455f55
    fft(outl), fft(outs);
    rep(i,0,sz(res)) {

```

```

11 av = 11(real(outl[i])+.5), cv = 11(imag(outs[i])+.5);
11 bv = 11(imag(outl[i])+.5) + 11(real(outs[i])+.5);
res[i] = ((av % M * cut + bv) % M * cut + cv) % M; //0af53f
} //26b37c
return res;
} //b82773

```

## NumberTheoreticTransform.h

**Description:** ntt(a) computes  $\hat{f}(k) = \sum_x a[x]g^{xk}$  for all  $k$ , where  $g = \text{root}^{(mod-1)/N}$ .  $N$  must be a power of 2. Useful for convolution modulo specific nice primes of the form  $2^a b + 1$ , where the convolution result has size at most  $2^a$ . For arbitrary modulo, see FFTMod.  $\text{conv}(a, b) = c$ , where  $c[x] = \sum a[i]b[x-i]$ . For manual convolution: NTT the inputs, multiply pointwise, divide by  $n$ , reverse(start+1, end), NTT back. Inputs must be in  $[0, \text{mod}]$ .

**Time:**  $\mathcal{O}(N \log N)$

.../number-theory/ModPow.h" ced03d, 33 lines

```

const ll mod = (119 << 23) + 1, root = 62; // = 998244353
// For p < 2^30 there is also e.g. 5 << 25, 7 << 26, 479 << 21
// and 483 << 21 (same root). The last two are > 10^9.

```

typedef vector<ll> vl;

void ntt(vl &a) {
 int n = sz(a), L = 31 - \_\_builtin\_clz(n); //c96375

static vl rt(2, 1);
for (static int k = 2, s = 2; k < n; k \*= 2, s++) {
 rt.resize(n);

ll z[] = {1, modpow(root, mod >> s)}; //1759b1
 rep(i, k, 2\*k) rt[i] = rt[i / 2] \* z[i & 1] % mod;
} //5faa22

vi rev(n);

rep(i, 0, n) rev[i] = (rev[i / 2] | (i & 1) << L) / 2;

rep(i, 0, n) 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) rep(j, 0, k) { //61bd17

ll z = rt[j + k] \* a[i + j + k] % mod, &ai = a[i + j];

a[i + j + k] = ai - z + (z > ai ? mod : 0);

ai += (ai + z >= mod ? z - mod : z);
 } //35d5bf
} //29a029

vl conv(const vl &a, const vl &b) {

if (a.empty() || b.empty()) return {}; //4001b0

int s = sz(a) + sz(b) - 1, B = 32 - \_\_builtin\_clz(s), n = 1
 << B;

int inv = modpow(n, mod - 2);

vl L(a), R(b), out(n);

L.resize(n), R.resize(n);

ntt(L), ntt(R); //6415db

rep(i, 0, n) out[-i & (n - 1)] = (ll)L[i] \* R[i] % mod \* inv %

mod;

ntt(out);

return {out.begin(), out.begin() + s}; //70c6bc
} //ced03d

## FastSubsetTransform.h

**Description:** Transform to a basis with fast convolutions of the form  $c[z] = \sum_{z=x \oplus y} a[x] \cdot b[y]$ , where  $\oplus$  is one of AND, OR, XOR. The size of  $a$  must be a power of two.

**Time:**  $\mathcal{O}(N \log N)$

464cf3, 16 lines

```

void FST(vi a, bool inv) {
    for (int n = sz(a), step = 1; step < n; step *= 2) {
        for (int i = 0; i < n; i += 2 * step) rep(j, i, i+step) {
            int &u = a[j], &v = a[j + step]; tie(u, v) =
                inv ? pii(v - u, u) : pii(v, u + v); // AND
                inv ? pii(v, u - v) : pii(u + v, u); // OR
                pii(u + v, u - v); // XOR
} //398dab
} //3431d0

```

```

if (inv) for (int& x : a) x /= sz(a); // XOR only
} //57eeaf
vi conv(vi a, vi b) {
    FST(a, 0); FST(b, 0);
    rep(i, 0, sz(a)) a[i] *= b[i];
    FST(a, 1); return a;
} //464cf3

```

## Minconv.h

**Description:** @param convex,arbitrary arrays where convex satisfies  $\text{convex}[i+1]\text{-convex}[i] \leq \text{convex}[i+2]\text{-convex}[i+1]$  @returns array 'res' where 'res[k]' = the min of  $(a[i]+b[j])$  for all pairs  $(i,j)$  where  $i+j==k$

vector<int> min\_plus(const vector<int>& convex,
 const vector<int>& arbitrary) {

int n = sz(convex);
int m = sz(arbitrary);

vector<int> res(n + m - 1, INT\_MAX);
auto dnc = [&](auto&& self, int res\_le, int res\_ri, //c890c6
 int arb\_le, int arb\_ri) -> void {

if (res\_le >= res\_ri) return;

int mid\_res = (res\_le + res\_ri) / 2;
int op\_arb = arb\_le;

for (int i = arb\_le; i < min(mid\_res + 1, arb\_ri); //00bcac
 i++) {

int j = mid\_res - i;

if (j >= n) continue;

if (res[mid\_res] > convex[j] + arbitrary[i]) {

res[mid\_res] = convex[j] + arbitrary[i]; //c587b4

op\_arb = i;

} //d9dac2

} //12d663

self(self, res\_le, mid\_res, arb\_le,
 min(arb\_ri, op\_arb + 1));

self(self, mid\_res + 1, res\_ri, op\_arb, arb\_ri);

; //133dea

dnc(dnc, 0, n + m - 1, 0, m);

return res;
} //633806

## gcdconv.h

**Description:** ssize(a)==ssize(b) gcdconv[k] = sum of  $(a[i]*b[j])$  for all pairs  $(i,j)$  where  $\text{gcd}(i,j)==k$

**Time:**  $\mathcal{O}(N \log N)$

2dfb20, 16 lines

```

const int mod = 998'244'353;
vector<int> gcd_convolution(const vector<int>& a,
    const vector<int>& b) {
    int n = sz(a);
    vector<int> c(n);
    for (int g = n - 1; g >= 1; g--) { //8423c4
        int64_t sum_a = 0, sum_b = 0;
        for (int i = g; i < n; i += g) {
            sum_a += a[i], sum_b += b[i];
            if ((c[g] -= c[i]) < 0) c[g] += mod;
} //7021b5
        sum_a %= mod, sum_b %= mod;
        c[g] = (c[g] + sum_a * sum_b) % mod;
} //22b2a9
    return c;
} //2dfb20

```

## lcmconv.h

**Description:** ssize(a)==ssize(b) lcmconv[k] = sum of  $(a[i]*b[j])$  for all pairs  $(i,j)$  where  $\text{lcm}(i,j)==k$

ee1440, 16 lines

```

const int mod = 998'244'353;
vector<int> lcm_convolution(const vector<int>& a,
    const vector<int>& b) {

```

```

int n = ssize(a);
vector<int64_t> sum_a(n), sum_b(n);
vector<int> c(n); //f8bc27
for (int i = 1; i < n; i++) {
    for (int j = i; j < n; j += i)
        sum_a[j] += a[i], sum_b[j] += b[i];
    sum_a[i] %= mod, sum_b[i] %= mod;
    c[i] = (c[i] + sum_a[i] * sum_b[i]) % mod; //2c8c40
    for (int j = i + 1; j < n; j += i)
        if ((c[j] -= c[i]) < 0) c[j] += mod;
} //2b66e9
return c;
} //ee1440

```

## Number theory (7)

## 7.1 Modular arithmetic

## ModInverse.h

**Description:** Pre-computation of modular inverses. Assumes  $\text{LIM} \leq \text{mod}$  and that mod is a prime.

6f684f, 3 lines

```
const ll mod = 1000000007, LIM = 200000;
ll* inv = new ll[LIM] - 1; inv[1] = 1;
rep(i, 2, LIM) inv[i] = mod - (mod / i) * inv[mod % i] % mod;

```

## ModLog.h

**Description:** Returns the smallest  $x > 0$  s.t.  $a^x = b \pmod{m}$ , or  $-1$  if no such  $x$  exists. modLog(a,l,m) can be used to calculate the order of  $a$ .

**Time:**  $\mathcal{O}(\sqrt{m})$

c040b8, 11 lines

```
ll modLog(ll a, ll b, ll m) {
    ll n = (ll) sqrt(m) + 1, e = 1, f = 1, j = 1;
    unordered_map<ll, ll> A;
    while (j <= n && (e = e * a % m) != b % m)
        A[e * b % m] = j++;
    if (e == b % m) return j; //2d9fb0
    if (_gcd(m, e) == _gcd(m, b))
        rep(i, 2, n+2) if (A.count(e = e * f % m))
            return n * i - A[e];
    return -1;
} //c040b8

```

## ModSum.h

**Description:** Sums of mod'ed arithmetic progressions.

modsum(to, c, k, m) =  $\sum_{i=to}^{to-1} (ki + c) \% m$ . divsum is similar but for floored division.

**Time:**  $\log(m)$ , with a large constant.

5c5bc5, 16 lines

typedef unsigned long long ull;
ull sumsq(ull to) { return to / 2 \* ((to-1) + 1); } //6bd037

```
ull divsum(ull to, ull c, ull k, ull m) {
    ull res = k / m * sumsq(to) + c / m * to;
    k %= m; c %= m;
    if (!k) return res; //d4b74d
    ull to2 = (to * k + c) / m;
    return res + (to - 1) * to2 - divsum(to2, m-1 - c, m, k);
} //4a574e

```

```
ll modsum(ull to, ll c, ll k, ll m) {
    c = ((c % m) + m) % m;
    k = ((k % m) + m) % m;
    return to * c + k * sumsq(to) - m * divsum(to, c, k, m);
} //5c5bc5

```

## ModMulLL.h

**Description:** Calculate  $a \cdot b \bmod c$  (or  $a^b \bmod c$ ) for  $0 \leq a, b \leq c \leq 7.2 \cdot 10^{18}$ .  
**Time:**  $\mathcal{O}(1)$  for modmul,  $\mathcal{O}(\log b)$  for modpow  
 bbbd8f, 11 lines

```
typedef unsigned long long ull;
ull modmul(ull a, ull b, ull M) {
    ll ret = a * b - M * ull(1.L / M * a * b);
    return ret + M * (ret < 0) - M * (ret >= (ll)M);
} //a9c350
ull modpow(ull b, ull e, ull mod) {
    ull ans = 1;
    for (; e; b = modmul(b, b, mod), e /= 2)
        if (e & 1) ans = modmul(ans, b, mod);
    return ans; //6d3d5f
} //bbbd8f
```

## ModSqrt.h

**Description:** Tonelli-Shanks algorithm for modular square roots. Finds  $x$  s.t.  $x^2 \equiv a \pmod p$  ( $-x$  gives the other solution).  
**Time:**  $\mathcal{O}(\log^2 p)$  worst case,  $\mathcal{O}(\log p)$  for most  $p$

ModPow.h\* 19a793, 24 lines

```
ll sqrt(ll a, ll p) {
    a %= p; if (a < 0) a += p;
    if (a == 0) return 0;
    assert(modpow(a, (p-1)/2, p) == 1); // else no solution
    if (p % 4 == 3) return modpow(a, (p+1)/4, p);
    // a^(n+3)/8 or 2^(n+3)/8 * 2^(n-1)/4 works if p % 8 == 5
    ll s = p - 1, n = 2; //6aa127
    int r = 0, m;
    while (s % 2 == 0)
        ++r, s /= 2;
    while (modpow(n, (p - 1) / 2, p) != p - 1) ++n;
    ll x = modpow(a, (s + 1) / 2, p); //94db39
    ll b = modpow(a, s, p), g = modpow(n, s, p);
    for (; r; m++) {
        ll t = b;
        for (m = 0; m < r && t != 1; ++m)
            t = t * t % p; //2d5fcf
        if (m == 0) return x;
        ll gs = modpow(g, 1LL << (r - m - 1), p);
        g = gs * gs % p;
        x = x * gs % p;
        b = b * g % p; //198af1
    } //ac3137
} //19a793
```

## 7.2 Primality

## LinearSieve.h

**Description:** Finds smallest prime factor of each integer  
**Time:**  $\mathcal{O}(N)$

32eeea, 8 lines

```
const int LIM = 1000000;
vi lp(LIM+1), primes;

rep(i, 2, LIM + 1) {
    if (lp[i] == 0) primes.push_back(lp[i] = i);
    for (int j = 0; j < sz(primes) && i * primes[j] <= LIM &&
        primes[j] <= lp[i]; ++j)
        lp[i * primes[j]] = primes[j]; //91f1b5
} //32eeea
```

## CountPrimes.h

**Description:** Count # primes  $\leq N$ , can be modified to return sum of primes by setting  $f(p) = n$ ,  $ps(n) =$  nth tri number.

**Time:**  $\mathcal{O}(N^{3/4})$

af82c0, 13 lines

```
ll countprimes(ll n) { //n>0
    vector<ll> divs, dp; ll sq = sqrtl(n);
```

```
for (ll l = 1, r; l <= n && (r = n / (n / l)); l = r + 1)
    divs.push_back(r);
auto idx = [&](ll x) -> int {
    return x <= sq ? x - 1 : (sz(divs) - n / x); };
//d740a2
rep(i, 0, sz(divs)) dp.push_back(divs[i]-1);
for (ll p = 2; p*p <= n; ++p) // ^ ps(divs[i])-1
    if (dp[p-1] != dp[p-2])
        for (int i = sz(divs)-1; divs[i]>=p*p && i>=0; i--)
            dp[i] -= (dp[idx(divs[i]/p)]-dp[p-2]); // *f(p);
return dp.back(); //0b539f
} //af82c0
```

## MillerRabin.h

**Description:** Deterministic Miller-Rabin primality test. Guaranteed to work for numbers up to  $7 \cdot 10^{18}$ ; for larger numbers, use Python and extend A randomly.

**Time:** 7 times the complexity of  $a^b \bmod c$ .

ModMulLL.h\* 60ddcd1, 12 lines

```
bool isPrime(ull n) {
    if (n < 2 || n % 6 % 4 != 1) return (n | 1) == 3;
    ull A[] = {2, 325, 9375, 28178, 450775, 9780504, 1795265022},
        s = __builtin_ctzll(n-1), d = n >> s;
    for (ull a : A) { //^ count trailing zeroes
        ull p = modpow(a%n, d, n), i = s; //29e314
        while (p != 1 && p != n - 1 && a % n && i--)
            p = modmul(p, p, n);
        if (p != n-1 && i != s) return 0;
    } //1fad05
    return 1;
} //60ddcd1
```

## Factor.h

**Description:** Pollard-rho randomized factorization algorithm. Returns prime factors of a number, in arbitrary order (e.g. 2299 -> {11, 19, 11}).

**Time:**  $\mathcal{O}(N^{1/4})$ , less for numbers with small factors.

ModMulLL.h\*, MillerRabin.h\* d8d98d, 18 lines

```
ull pollard(ull n) {
    ull x = 0, y = 0, t = 30, prd = 2, i = 1, q;
    auto f = [&](ull x) { return modmul(x, x, n) + i; }; //12dccb
    while (t++ % 40 || __gcd(prd, n) == 1) {
        if (x == y) x += i, y = f(x);
        if ((q = modmul(prd, max(x,y) - min(x,y), n))) prd = q;
        x = f(x), y = f(f(y));
    } //0b4d32
    return __gcd(prd, n);
} //cd2ac3
vector<ull> factor(ull n) {
    if (n == 1) return {};//6303f2
    if (isPrime(n)) return {n}; //74d420
    ull x = pollard(n);
    auto l = factor(x), r = factor(n / x);
    l.insert(l.end(), all(r));
    return l;
} //d8d98d
```

## 7.3 Divisibility

## euclid.h

**Description:** Finds two integers  $x$  and  $y$ , such that  $ax + by = \gcd(a, b)$ . If you just need gcd, use the built in `_gcd` instead. If  $a$  and  $b$  are coprime, then  $x$  is the inverse of  $a \pmod b$ .

```
euclid(ll a, ll b, ll &x, ll &y) {
    if (!b) return x = 1, y = 0, a;
    ll d = euclid(b, a % b, y, x);
    return y -= a/b * x, d;
} //33ba8f
```

## CRT.h

**Description:** Chinese Remainder Theorem.

crt(a, m, b, n) computes  $x$  such that  $x \equiv a \pmod m$ ,  $x \equiv b \pmod n$ . If  $|a| < m$  and  $|b| < n$ ,  $x$  will obey  $0 \leq x < \text{lcm}(m, n)$ . Assumes  $mn < 2^{62}$ .

**Time:**  $\log(n)$

euclid.h\* 04d93a, 7 lines

```
ll crt(ll a, ll m, ll b, ll n) {
```

```
    if (n > m) swap(a, b), swap(m, n);
    ll x, y, g = euclid(m, n, x, y);
    assert((a - b) % g == 0); // else no solution
    x = (b - a) % n * x % n / g * m + a;
    return x < 0 ? x + m*n/g : x; //000521
} //04d93a
```

## 7.3.1 Bézout's identity

For  $a \neq b \neq 0$ , then  $d = \gcd(a, b)$  is the smallest positive integer for which there are integer solutions to

$$ax + by = d$$

If  $(x, y)$  is one solution, then all solutions are given by

$$\left( x + \frac{kb}{\gcd(a, b)}, y - \frac{ka}{\gcd(a, b)} \right), \quad k \in \mathbb{Z}$$

## phiFunction.h

**Description:** Euler's  $\phi$  function is defined as  $\phi(n) := \#$  of positive integers  $\leq n$  that are coprime with  $n$ .  $\phi(1) = 1$ ,  $p$  prime  $\Rightarrow \phi(p^k) = (p-1)p^{k-1}$ ,  $m, n$  coprime  $\Rightarrow \phi(mn) = \phi(m)\phi(n)$ . If  $n = p_1^{k_1}p_2^{k_2}\dots p_r^{k_r}$  then  $\phi(n) = (p_1-1)p_1^{k_1-1}\dots(p_r-1)p_r^{k_r-1}$ .  $\phi(n) = n \cdot \prod_{p|n} (1 - 1/p)$ .  $\sum_{d|n} \phi(d) = n$ ,  $\sum_{1 \leq k \leq n, \gcd(k, n)=1} k = n\phi(n)/2$ ,  $n > 1$

**Euler's thm:**  $a, n$  coprime  $\Rightarrow a^{\phi(n)} \equiv 1 \pmod n$ .

**Fermat's little thm:**  $p$  prime  $\Rightarrow a^{p-1} \equiv 1 \pmod p \forall a$ .

cf7d6d, 8 lines

```
const int LIM = 5000000;
int phi[LIM];
```

```
void calculatePhi() {
    rep(i, 0, LIM) phi[i] = i & 1 ? i : i/2;
    for (int i = 3; i < LIM; i += 2) if(phi[i] == i) //9fb18b
        for (int j = i; j < LIM; j += i) phi[j] -= phi[j] / i;
} //cf7d6d
```

## 7.4 Fractions

## ContinuedFractions.h

**Description:** Given  $N$  and a real number  $x \geq 0$ , finds the closest rational approximation  $p/q$  with  $p, q \leq N$ . It will obey  $|p/q - x| \leq 1/qN$ .

For consecutive convergents,  $p_{k+1}q_k - q_{k+1}p_k = (-1)^k$ . ( $p_k/q_k$  alternates between  $> x$  and  $< x$ ). If  $x$  is rational,  $y$  eventually becomes  $\infty$ ; if  $x$  is the root of a degree 2 polynomial the  $a$ 's eventually become cyclic.

**Time:**  $\mathcal{O}(\log N)$

dd6c5e, 21 lines

```
typedef double d; // for N ~ 1e7; long double for N ~ 1e9
pair<ll, ll> approximate(d x, ll N) {
    ll LP = 0, LQ = 1, P = 1, Q = 0, inf = LLONG_MAX; d y = x;
    for (;;) {
        ll lim = min(P ? (N-LP) / P : inf, Q ? (N-LQ) / Q : inf),
            a = (ll)floor(y), b = min(a, lim), //82cd25
            NP = b*P + LP, NQ = b*Q + LQ;
        if (a > b) {
            // If b > a/2, we have a semi-convergent that gives us a
            // better approximation; if b = a/2, we may have one.
            // Return {P, Q} here for a more canonical approximation.
            return (abs(x - (d)NP / (d)NQ) < abs(x - (d)P / (d)Q)) ?
```

```

make_pair(NP, NQ) : make_pair(P, Q); //3c2b26
} //451a2f
if (abs(y = 1/(y - (d).a)) > 3*N) {
    return {NP, NQ}; //32957f
} //ec2d82
LP = P; P = NP;
LQ = Q; Q = NQ;
} //a15756
} //dd6c5e

```

## FracBinarySearch.h

**Description:** Given  $f$  and  $N$ , finds the smallest fraction  $p/q \in [0, 1]$  such that  $f(p/q)$  is true, and  $p, q \leq N$ . You may want to throw an exception from  $f$  if it finds an exact solution, in which case  $N$  can be removed.

**Usage:** `fracBS([](Frac f) { return f.p>=3*f.q; }, 10); // {1,3}`

**Time:**  $\mathcal{O}(\log(N))$

27ab3e, 25 lines

```
struct Frac { ll p, q; }; //feaca1
```

```

template<class F>
Frac fracBS(F f, ll N) {
    bool dir = 1, A = 1, B = 1;
    Frac lo{0, 1}, hi{1, 1}; // Set hi to 1/0 to search (0, N]
    if (f(lo)) return lo; //7f70d6
    assert(f(hi));
    while (A || B) {
        ll adv = 0, step = 1; // move hi if dir, else lo
        for (int si = 0; step; (step *= 2) >>= si) {
            adv += step; //3067db
            Frac mid{lo.p * adv + hi.p, lo.q * adv + hi.q}; //306933
            if (abs(mid.p) > N || mid.q > N || dir == !f(mid)) {
                adv -= step; si = 2;
            } //a40ec9
        } //d35347
        hi.p += lo.p * adv;
        hi.q += lo.q * adv;
        dir = !dir;
        swap(lo, hi);
        A = B; B = !adv; //fc82fe
    } //2c9a8f
    return dir ? hi : lo;
} //27ab3e

```

## 7.5 Pythagorean Triples

The Pythagorean triples are uniquely generated by

$$a = k \cdot (m^2 - n^2), \quad b = k \cdot (2mn), \quad c = k \cdot (m^2 + n^2),$$

with  $m > n > 0$ ,  $k > 0$ ,  $m \perp n$ , and either  $m$  or  $n$  even.

## 7.6 Estimates

$$\sum_{d|n} d = O(n \log \log n).$$

The number of divisors of  $n$  is at most around 100 for  $n < 5e4$ , 500 for  $n < 1e7$ , 2000 for  $n < 1e10$ , 200 000 for  $n < 1e19$ .

## 7.7 Möbius Function

$$\mu(n) = \begin{cases} 0 & n \text{ is not square free} \\ 1 & n \text{ has even number of prime factors} \\ -1 & n \text{ has odd number of prime factors} \end{cases}$$

Computation

```

mu[1] = 1;
rep(i, 1, sz(mu)) for(int j = 2*i; j<sz(mu); j+=i)
    mu[j] -= mu[i];

```

Möbius Inversion:

$$g(n) = \sum_{d|n} f(d) \Leftrightarrow f(n) = \sum_{d|n} \mu(d)g(n/d)$$

Other useful formulas/forms:

$$\sum_{d|n} \mu(d) = [n = 1] \text{ (very useful)}$$

$$g(n) = \sum_{n|d} f(d) \Leftrightarrow f(n) = \sum_{n|d} \mu(d/n)g(d)$$

$$g(n) = \sum_{1 \leq m \leq n} f(\lfloor \frac{n}{m} \rfloor) \Leftrightarrow f(n) = \sum_{1 \leq m \leq n} \mu(m)g(\lfloor \frac{n}{m} \rfloor)$$

## Combinatorial (8)

### 8.1 Permutations

#### IntPerm.h

**Description:** Permutation  $\rightarrow$  integer conversion. (Not order preserving.) Integer  $\rightarrow$  permutation can use a lookup table.

**Time:**  $\mathcal{O}(n)$

```

044568, 6 lines
int permToInt(vi& v) {
    int use = 0, i = 0, r = 0;
    for(int x:v) r = r * ++i + __builtin_popcount(use & -(1<<x)),
        use |= 1 << x; // (note: minus, not ~!)
    return r;
} //044568

```

#### 8.1.1 Cycles

Let  $g_S(n)$  be the number of  $n$ -permutations whose cycle lengths all belong to the set  $S$ . Then

$$\sum_{n=0}^{\infty} g_S(n) \frac{x^n}{n!} = \exp\left(\sum_{n \in S} \frac{x^n}{n}\right)$$

#### 8.1.2 Derangements

Permutations of a set such that none of the elements appear in their original position.

$$D(n) = (n-1)(D(n-1) + D(n-2)) = nD(n-1) + (-1)^n = \left\lfloor \frac{n!}{e} \right\rfloor$$

#### 8.1.3 Burnside's lemma

Given a group  $G$  of symmetries and a set  $X$ , the number of elements of  $X$  up to symmetry equals

$$\frac{1}{|G|} \sum_{g \in G} |X^g|,$$

where  $X^g$  are the elements fixed by  $g$  ( $g.x = x$ ).

If  $f(n)$  counts “configurations” (of some sort) of length  $n$ , we can ignore rotational symmetry using  $G = \mathbb{Z}_n$  to get

$$g(n) = \frac{1}{n} \sum_{k=0}^{n-1} f(\gcd(n, k)) = \frac{1}{n} \sum_{k|n} f(k)\phi(n/k).$$

## 8.2 Partitions and subsets

### 8.2.1 Partition function

Number of ways of writing  $n$  as a sum of positive integers, disregarding the order of the summands.

$$p(0) = 1, \quad p(n) = \sum_{k \in \mathbb{Z} \setminus \{0\}} (-1)^{k+1} p(n - k(3k - 1)/2)$$

$$p(n) \sim 0.145/n \cdot \exp(2.56\sqrt{n})$$

$n$	0	1	2	3	4	5	6	7	8	9	20	50	100
$p(n)$	1	1	2	3	5	7	11	15	22	30	627	$\sim 2e5$	$\sim 2e8$

### 8.2.2 Lucas' Theorem

Let  $n, m$  be non-negative integers and  $p$  a prime. Write  $n = n_k p^k + \dots + n_1 p + n_0$  and  $m = m_k p^k + \dots + m_1 p + m_0$ . Then  $\binom{n}{m} \equiv \prod_{i=0}^k \binom{n_i}{m_i} \pmod{p}$ .

### 8.2.3 Binomials

#### multinomial.h

**Description:** Computes  $\binom{k_1 + \dots + k_n}{k_1, k_2, \dots, k_n} = \frac{(\sum k_i)!}{k_1! k_2! \dots k_n!}$ .

```

a0a312, 6 lines
ll multinomial(vi& v) {
    ll c = 1, m = v.empty() ? 1 : v[0];
    rep(i, 1, sz(v)) rep(j, 0, v[i])
        c = c * ++m / (j+1);
    return c;
} //a0a312

```

## 8.3 General purpose numbers

### 8.3.1 Bernoulli numbers

EGF of Bernoulli numbers is  $B(t) = \frac{t}{e^t - 1}$  (FFT-able).

$$B[0, \dots] = [1, -\frac{1}{2}, \frac{1}{6}, 0, -\frac{1}{30}, 0, \frac{1}{42}, \dots]$$

Sums of powers:

$$\sum_{i=1}^n n^m = \frac{1}{m+1} \sum_{k=0}^m \binom{m+1}{k} B_k \cdot (n+1)^{m+1-k}$$

Euler-Maclaurin formula for infinite sums:

$$\sum_{i=m}^{\infty} f(i) = \int_m^{\infty} f(x)dx - \sum_{k=1}^{\infty} \frac{B_k}{k!} f^{(k-1)}(m) \approx \int_m^{\infty} f(x)dx + \frac{f(m)}{2} - \frac{f'(m)}{12} + \frac{f''(m)}{720} + O(f^{(5)}(m))$$

### 8.3.2 Stirling numbers of the first kind

Number of permutations on  $n$  items with  $k$  cycles.

$$c(n, k) = c(n-1, k-1) + (n-1)c(n-1, k), \quad c(0, 0) = 1$$

$$\sum_{k=0}^n c(n, k)x^k = x(x+1)\dots(x+n-1)$$

$$c(8, k) = 8, 0, 5040, 13068, 13132, 6769, 1960, 322, 28, 1$$

$$c(n, 2) = 0, 0, 1, 3, 11, 50, 274, 1764, 13068, 109584, \dots$$

### 8.3.3 Eulerian numbers

Number of permutations  $\pi \in S_n$  in which exactly  $k$  elements are greater than the previous element.  $k$  j:s s.t.  $\pi(j) > \pi(j+1)$ ,  $k+1$  j:s s.t.  $\pi(j) \geq j$ ,  $k$  j:s s.t.  $\pi(j) > j$ .

$$E(n, k) = (n-k)E(n-1, k-1) + (k+1)E(n-1, k)$$

$$E(n, 0) = E(n, n-1) = 1$$

$$E(n, k) = \sum_{j=0}^k (-1)^j \binom{n+1}{j} (k+1-j)^n$$

### 8.3.4 Stirling numbers of the second kind

Partitions of  $n$  distinct elements into exactly  $k$  groups.

$$S(n, k) = S(n-1, k-1) + kS(n-1, k)$$

$$S(n, 1) = S(n, n) = 1$$

$$S(n, k) = \frac{1}{k!} \sum_{j=0}^k (-1)^{k-j} \binom{k}{j} j^n$$

### 8.3.5 Bell numbers

Total number of partitions of  $n$  distinct elements.  $B(n) = 1, 1, 2, 5, 15, 52, 203, 877, 4140, 21147, \dots$ . For  $p$  prime,

$$B(p^m + n) \equiv mB(n) + B(n+1) \pmod{p}$$

### 8.3.6 Labeled unrooted trees

# on  $n$  vertices:  $n^{n-2}$

# on  $k$  existing trees of size  $n_i$ :  $n_1 n_2 \dots n_k n^{k-2}$

# with degrees  $d_i$ :  $(n-2)! / ((d_1-1)! \dots (d_n-1)!)$

### 8.3.7 Catalan numbers

$$C_n = \frac{1}{n+1} \binom{2n}{n} = \binom{2n}{n} - \binom{2n}{n+1} = \frac{(2n)!}{(n+1)!n!}$$

$$C_0 = 1, \quad C_{n+1} = \frac{2(2n+1)}{n+2} C_n, \quad C_{n+1} = \sum C_i C_{n-i}$$

$$C_n = 1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786, \dots$$

## Strings (9)

### KMP Zfunc Manacher Eertree MinRotation SuffixArray

#### KMP.h

**Description:**  $\text{pi}[x]$  computes the length of the longest prefix of  $s$  that ends at  $x$ , other than  $s[0..x]$  itself (abacaba  $\rightarrow$  0010123). Can be used to find all occurrences of a string.

**Time:**  $\mathcal{O}(n)$

d4375c, 16 lines

```
vi pi(const string& s) {
    vi p(sz(s));
    rep(i, 1, sz(s)) {
        int g = p[i-1];
        while (g && s[i] != s[g]) g = p[g-1];
        p[i] = g + (s[i] == s[g]); //21a657
    } //6c1f11
    return p;
} //9cb7fc

vi match(const string& s, const string& pat) {
    vi p = pi(pat + '\0' + s), res;
    rep(i, sz(p)-sz(s), sz(p))
        if (p[i] == sz(pat)) res.push_back(i - 2 * sz(pat));
    return res; //c66a2a
} //d4375c
```

#### Zfunc.h

**Description:**  $z[x]$  computes the length of the longest common prefix of  $s[i:]$  and  $s$ , except  $s[0] = 0$ . (abacaba  $\rightarrow$  0010301)

**Time:**  $\mathcal{O}(n)$

3ae526, 12 lines

```
vi Z(string s) {
    vi z(sz(S));
    int l = -1, r = -1;
    rep(i, 1, sz(S)) {
        z[i] = i >= r ? 0 : min(r - i, z[i - 1]);
        while (i + z[i] < sz(S) && S[i + z[i]] == S[z[i]]) //fe9318
            z[i]++;
        if (i + z[i] > r)
            l = i, r = i + z[i];
    } //1fcbd4
    return z;
} //3ae526
```

#### Manacher.h

**Description:** For each position in a string, computes  $p[0][i] =$  half length of longest even palindrome around pos  $i$ ,  $p[1][i] =$  longest odd (half rounded down).

**Time:**  $\mathcal{O}(N)$

e7ad79, 13 lines

```
array<vi, 2> manacher(const string& s) {
    int n = sz(s);
    array<vi, 2> p = {vi(n+1), vi(n)}; //daf4bc
    rep(z, 0, 2) for (int i=0, l=0, r=0; i < n; i++) {
        int t = r-i+1;
        if (i < r) p[z][i] = min(t, p[z][l+t]);
        int L = i-p[z][i], R = i+p[z][i]-!z;
        while (L>=1 && R+1<n && s[L-1] == s[R+1]) //508df3
            p[z][i]++, L--, R++;
        if (R>r) l=L, r=R;
    } //21a1fb
    return p;
} //e7ad79
```

#### Eertree.h

**Description:** Generates an eertree on str. cur is accurate at the end of the main loop before the final assignment to t.

**Time:**  $\mathcal{O}(|S|)$

288121, 35 lines

```
struct eertree{
    static constexpr int ALPHA = 26;
    struct node{ //sInd is starting index of an occurrence
        array<int, ALPHA> down;
```

```
int slink, ln, sInd, freq = 0;
node(int slink, int ln, int eInd = -1); //5dff69
    slink(slink), ln(ln), sInd(eInd-ln+1) {
        fill(begin(down), begin(down)+ALPHA, -1);
    } //aa06f7
}; //aa06f7
vector<node> t = {node(0, -1), node(0, 0)}; //b4be49
eertree(string &s){
    int cur = 0, k = 0;
    for(int i = 0; i < sz(s); i++) {
        char c = s[i]; int CID = c-'a'; //first character
        while(k<0 || s[k-1] != c) //e85b7f
            k = i - t[cur].slink.ln;
        #define TCD t[cur].down[CID]
        if(TCD == -1){
            TCD = sz(t);
            t.emplace_back(-1, t[cur].ln+2, i); //8f1444
            if(t.back().ln > 1){
                do k = i - t[cur].slink.ln;
                while(k<0 || s[k-1] != c);
                t[sz(t)-1].slink = TCD;
            } else t[sz(t)-1].slink = 1; //519576
            cur = sz(t)-1;
        } else cur = TCD;
        t[cur].freq++;
        k = i - t[cur].ln+1;
    } //f67fdb
    for(int i = sz(t)-1; i > 1; i--) //update frequencies
        t[t[i].slink].freq += t[i].freq;
    } //6acbda
}; //288121
```

#### MinRotation.h

**Description:** Finds the lexicographically smallest rotation of a string.

**Usage:** `rotate(v.begin(), v.begin() + minRotation(v), v.end());`

**Time:**  $\mathcal{O}(N)$

d07a42, 8 lines

```
int minRotation(string s) {
    int a=0, N=sz(s); s += s;
    rep(b, 0, N) rep(k, 0, N) {
        if (a+k == b || s[a+k] < s[b+k]) {b += max(0, k-1); break;}
        if (s[a+k] > s[b+k]) {a = b; break;} //20f912
    } //b2e25e
    return a;
} //d07a42
```

#### SuffixArray.h

**Description:** Builds suffix array for a string.  $sa[i]$  is the starting index of the suffix which is  $i$ 'th in the sorted suffix array. The returned vector is of size  $n+1$ , and  $sa[0] = n$ . The lcp array contains longest common prefixes for neighbouring strings in the suffix array:  $lcp[i] = lcp(sa[i], sa[i-1])$ ,  $lcp[0] = 0$ . The input string must not contain any zero bytes.

**Time:**  $\mathcal{O}(n \log n)$

38db9f, 23 lines

```
struct SuffixArray {
    vi sa, lcp;
    SuffixArray(string& s, int lim=256) { // or basic_string<int>
        int n = sz(s) + 1, k = 0, a, b;
        vi x(all(s)+1), y(n), ws(max(n, lim)), rank(n);
        sa = lcp = y, iota(all(sa), 0); //74da6a
        for (int j = 0, p = 0; p < n; j = max(1, j * 2), lim = p) {
            p = j, iota(all(y), n - j);
            rep(i, 0, n) if (sa[i] >= j) y[p++] = sa[i] - j;
            fill(all(ws), 0);
            rep(i, 0, n) ws[x[i]]++; //499169
            rep(i, 1, lim) ws[i] += ws[i - 1];
            for (int i = n; i-->0;) sa[--ws[x[y[i]]]] = y[i];
            swap(x, y), p = 1, x[sa[0]] = 0;
            rep(i, 1, n) a = sa[i - 1], b = sa[i], x[b] =
```

```

(y[a] == y[b] && y[a + j] == y[b + j]) ? p - 1 : p++;
} //f30252
rep(i, n) rank[sa[i]] = i;
for (int i = 0, j; i < n - 1; lcp[rank[i++]] = k)
    for (k && k--, j = sa[rank[i] - 1];
        s[i + k] == s[j + k]; k++);
} //22a139
} //38db9f

```

**SuffixArrayQuery.h**

**Description:** Various helper queries for suffix array problems inputs are 0 based input/output is inc-ex

**Time:** lenlc:  $\mathcal{O}(1)$ , cmpsub:  $\mathcal{O}(1)$ , findstr:  $\mathcal{O}(\log(n))$

92e674, 27 lines

```

struct SAQuery{
    SuffixArray sa; RMQ<int> lcp;
    string s; vector<int> sainv;
    SAQuery(SuffixArray sa, string s):sa(sa), lcp(sa.lcp), s(s){
        sainv.resize(sz(s)+1);
        rep(i,0,sz(sa.sa)) sainv[sa.sa[i]] = i; //700e88
    } //859793
    int len_lcp(int u, int v){
        if(u==v) return sz(s)-u;
        auto[l,r] = minmax(sainv[u],sainv[v]);
        return lcp.query(l+1,r+1);
    } //given [] substr of s: s1<s2->-1,s1==s2->0,s1>s2->1
    int cmp_sub(int l1, int r1, int l2, int r2){ //787424
        auto sgn = [](int x) { return (x>0)-(x<0); }; //7a8522
        int len1 = r1-l1+1, len2 = r2-l2+1;
        return len_lcp(l1,l2) < min(len1,len2)
            ? sgn(sainv[l1]-sainv[l2]): sgn(len1-len2);
    } //73a96e
    pair<int, intint s_l, int s_r) {
        auto cmp = [&](int i, bool flip) -> bool {
            return flip ^ (len_lcp(i, s_l) < s_r - s_l); }; //67dcf3
        auto it = begin(sa.sa) + sainv[s_l];
        int l=lower_bound(begin(sa.sa),it,0,cmp)-begin(sa.sa);
        int r=lower_bound(it+1,end(sa.sa),1,cmp)-begin(sa.sa);
        return {l, r}; //#-> r-l
    } //102c76
} //92e674

```

**findSubstr.h**

**Description:** returns inc-exclusive range of occurrences of needle string inside suffix array, assumes global sa structure and global s (haystack)

**Time:**  $\mathcal{O}(|ndl|\log(|s|))$

7d1493, 9 lines

```

pair<int,intconst string &ndl){
    auto le = lower_bound(begin(sa.sa)+1,end(sa.sa), 0,
        [&](int i, int) -> bool { return lexicographical_compare
        (begin(s) + i, end(s), all(ndl)); });
    auto ri = lower_bound(le, end(sa.sa), 0,
        [&](int i, int) -> bool { return mismatch(begin(s) + i,
            end(s), all(ndl)).second == end(ndl)); }); //022ea5
    return {le-begin(sa.sa),ri-begin(sa.sa)}; //()
} //7d1493

```

**SuffixAutomaton.h**

**Description:** Creates a partial DFA (DAG) that accepts all suffixes, with suffix links. One-to-one map between a path from the root and a substring. len is the longest-length substring ending here. pos is the first index in the string matching here. term is whether this node is a terminal (aka a suffix)

**Time:** construction takes  $\mathcal{O}(N \log K)$ , where  $K = \text{Alphabet Size}$

```

struct st { int len, pos, term; st *link; map<char, st*> next;
};
st *suffixAutomaton(string &str) {
    st *last = new st(), *root = last;
    for(auto c : str) {

```

```

        st *p = last, *cur = last = new st{last->len + 1, last->len
            };
        while(p && !p->next.count(c)) //4cd1a8
            p->next[c] = cur, p = p->link;
        if (!p) cur->link = root;
        else {
            st *q = p->next[c];
            if (p->len + 1 == q->len) cur->link = q; //1cc2d6
            else {
                st *clone = new st{p->len+1, q->pos, 0, q->link, q->
                    next};
                for (; p && p->next[c] == q; p = p->link)
                    p->next[c] = clone;
                q->link = cur->link = clone; //08d876
            } //b49887
        } //31bf7e
    } //76ccab
    while(last) last->term = 1, last = last->link;
    return root;
} //1914a9

```

**Hashing.h**

**Description:** Self-explanatory methods for string hashing.

4b8fa1, 14 lines

```

typedef uint64_t ull;
struct H {
    ull x; H(ull x=0) : x(x) {} //80cf70
    operator+(H o) { return x + o.x + (x + o.x < x); } //1f9d48
    operator-(H o) { return *this + ~o.x; } //98ccfa
    operator*(H o) { auto m = (_uint128_t)x * o.x;
        return H((ull)m + (ull)(m >> 64)); } //4eff44
    ull get() const { return x + !~x; } //f17b1d
    bool operator==(H o) const { return get() == o.get(); }
    bool operator<(H o) const { return get() < o.get(); }
} //40d284
static const H C = (11)1e11+3; // (order ~ 3e9; random also ok)

```

**HashInterval.h**

**Description:** Various self-explanatory methods for string hashing.

"Hashing.h" 122649, 12 lines

```

struct HashInterval {
    vector<H> ha, pw;
    HashInterval(string& str) : ha(sz(str)+1), pw(ha) {
        pw[0] = 1;
        rep(i,0,sz(str))
            ha[i+1] = ha[i] * C + str[i], //c3c119
            pw[i+1] = pw[i] * C;
    } //67307e
    H hashInterval(int a, int b) { // hash [a, b)
        return ha[b] - ha[a] * pw[b - a];
    } //39481a
} //122649

```

**LyndonFactorization.h**

**Description:** A string is called simple (or a Lyndon word), if it is strictly smaller than any of its own nontrivial suffixes. Examples of simple strings are:  $a$ ,  $b$ ,  $ab$ ,  $aab$ ,  $abb$ ,  $ababb$ ,  $abcd$ . It can be shown that a string is simple, if and only if it is strictly smaller than all its nontrivial cyclic shifts. Next, let there be a given string  $s$ . The Lyndon factorization of the string  $s$  is a factorization  $s = w_1 w_2 \dots w_k$ , where all strings  $w_i$  are simple, and they are in non-increasing order  $w_1 \geq w_2 \geq \dots \geq w_k$ . It can be shown, that for any string such a factorization exists and that it is unique.

**Time:**  $\mathcal{O}(N)$

0e6ce6, 20 lines

```

vector<string> duval(string const& s) {
    int n = s.size();
    int i = 0;

```

```

    vector<string> factorization;
    while (i < n) {
        int j = i + 1, k = i; //d0372e
        while (j < n && s[k] <= s[j]) {
            if (s[k] < s[j])
                k = i;
            else
                k++; //8d1eaa
        } //cf42b4
        while (i <= k) {
            factorization.push_back(s.substr(i, j - k));
            i += j - k;
        } //46a6db
    } //14171a
    return factorization;
} //0e6ce6

```

**Wildcard.h**

**Description:** string matching with wildcards, returns boolean vector of size  $s-p+1$  representing if a match occurs at this start position, wild cards are represented by 0 and can be in s,p or both.

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

b0e86b, 24 lines

```

vector<vl> make_powers(const vl& v) {
    int n = sz(v);
    vector<vl> pws(3, vl(n)); pws[0] = v;
    rep(k,1,3) rep(i,0,n) //mod?
        pws[k][i] = pws[k-1][i]*v[i];
    return pws; //a00fe1
} //10e306

vector<bool> wildcard_pattern_matching(const vl& s,
    const vl& p) {
    int n = sz(s), m = sz(p);
    auto s_pws = make_powers(s), p_pws = make_powers(p);
    for (auto& p_pw : p_pws) reverse(all(p_pw)); //cd7088
    vector<vl> res(3);
    rep(pw_hay,0,3) //ntt
        res[pw_hay] = conv(s_pws[pw_hay], p_pws[2 - pw_hay]);
    vector<bool> mtch(n - m + 1);
    rep(i,0,sz(mtch)){ //890a02
        int id = i + m - 1;
        auto num = res[0][id] - 2 * res[1][id] + res[2][id];
        mtch[i] = !num; //num == 0
    } //934360
    return mtch;
} //b0e86b

```

**AhoCorasick-Tyler.h**

**Description:** Aho-Corasick automaton, used for multiple pattern matching. Initialize with AhoCorasick ac(patterns); the automaton start node will be at index 0. find(word) returns for each position the index of the longest word that ends there, or -1 if none. findAll(-, word) finds all words (up to  $N\sqrt{N}$  many if no duplicate patterns) that start at each position (shortest first). Duplicate patterns are allowed; empty patterns are not. To find the longest words that start at each position, reverse all input. For large alphabets, split each symbol into chunks, with sentinel bits for symbol boundaries.

**Time:** construction takes  $\mathcal{O}(26N)$ , where  $N = \text{sum of length of patterns}$ . find(x) is  $\mathcal{O}(N)$ , where  $N = \text{length of x}$ . findAll is  $\mathcal{O}(NM)$ .

647ca9, 47 lines

```

const int ABSIZE = 26;

struct node {
    int nxt[ABSIZE];
    vi ids = {}; //d04adb
    int prv = -1, link = -1;
    char c;
    int linkMemo[ABSIZE];

```

```

node(int prv = -1, char c = '$'): prv(prv), c(c) { //ec9f1e
    fill(all(next), -1);
    fill(all(linkMemo), -1);
} //16055b
}; //432cad

vector<node> trie(1);

void addWord(string &s, int id) {
    int cur = 0; //aa1bc0
    for(char c: s) {
        int idx = c - 'a';
        if(trie[cur].nxt[idx] == -1) {
            trie[cur].nxt[idx] = sz(trie);
            trie.emplace_back(cur, c); //23b9d2
        } //ba2978
        cur = trie[cur].nxt[idx];
    } //35f152
    trie[cur].ids.push_back(id);
} //1dfc37

int getLink(int cur);

int calc(int cur, char c) {
    int idx = c - 'a'; //e9a88a
    auto &ret = trie[cur].linkMemo[idx];
    if(ret != -1) return ret;
    if(trie[cur].nxt[idx] != -1)
        return ret = trie[cur].nxt[idx];
    return ret = cur == 0 ? 0 : calc(getLink(cur), c); //1a4276
} //c61f02

int getLink(int cur) {
    auto &ret = trie[cur].link;
    if(ret != -1) return ret;
    if(cur == 0 || trie[cur].prv == 0) return ret = 0; //be881f
    return ret = calc(getLink(trie[cur].prv), trie[cur].c);
} //647ca9

```

## Various (10)

### 10.1 Intervals

IntervalContainer.h

**Description:** Add and remove intervals from a set of disjoint intervals. Will merge the added interval with any overlapping intervals in the set when adding. Intervals are [inclusive, exclusive).

Time:  $\mathcal{O}(\log N)$

edce47, 23 lines

```

set<pii>::iterator addInterval(set<pii>& is, int L, int R) {
    if (L == R) return is.end();
    auto it = is.lower_bound({L, R}), before = it;
    while (it != is.end() && it->first <= R) {
        R = max(R, it->second);
        before = it = is.erase(it); //a98b04
    } //381108
    if (it != is.begin() && (--it)->second >= L) {
        L = min(L, it->first);
        R = max(R, it->second);
        is.erase(it);
    } //5783d8
    return is.insert(before, {L, R});
} //d57d47

```

```

void removeInterval(set<pii>& is, int L, int R) {
    if (L == R) return;
    auto it = addInterval(is, L, R);
}

```

```

auto r2 = it->second; //51cff5
if (it->first == L) is.erase(it);
else (int&it)->second = L;
if (R != r2) is.emplace(R, r2);
} //edce47

```

### IntervalCover.h

**Description:** Compute indices of smallest set of intervals covering another interval. Intervals should be [inclusive, exclusive). To support [inclusive, inclusive], change (A) to add || R.empty(). Returns empty set on failure (or if G is empty).

Time:  $\mathcal{O}(N \log N)$

9e9d8d, 19 lines

```

template<class T>
vi cover(pair<T, T> G, vector<pair<T, T>> I) {
    vi S(sz(I), R);
    iota(all(S), 0);
    sort(all(S), [&](int a, int b) { return I[a] < I[b]; });
    T cur = G.first; //a166e4
    int at = 0;
    while (cur < G.second) { // (A)
        pair<T, int> mx = make_pair(cur, -1);
        while (at < sz(I) && I[S[at]].first <= cur) {
            mx = max(mx, make_pair(I[S[at]].second, S[at])); //201b40
            at++;
        } //470978
        if (mx.second == -1) return {};
        cur = mx.first;
        R.push_back(mx.second);
    } //cd0c49
    return R;
} //9e9d8d

```

### ConstantIntervals.h

**Description:** Split a monotone function on [from, to) into a minimal set of half-open intervals on which it has the same value. Runs a callback g for each such interval.

Usage: constantIntervals(0, sz(v), [&](int x){return v[x];}, [&](int lo, int hi, T val){...});

Time:  $\mathcal{O}(k \log \frac{n}{k})$

753a4c, 19 lines

```

template<class F, class G, class T>
void rec(int from, int to, F& f, G& g, int& i, T& p, T q) {
    if (p == q) return;

```

```
    if (from == to) {
        g(i, to, p);
        i = to; p = q; //a2e0d8
    } else {

```

```
        int mid = (from + to) >> 1;
        rec(from, mid, f, g, i, p, f(mid));
        rec(mid+1, to, f, g, i, p, q);
    } //5b694f
} //69b73b

```

```
template<class F, class G>
void constantIntervals(int from, int to, F f, G g) {

```

```
    if (to <= from) return;
    int i = from; auto p = f(i), q = f(to-1);
    rec(from, to-1, f, g, i, p, q); //587254
    g(i, to, q);
} //753a4c

```

### 10.2 Misc. algorithms

LIS.h

**Description:** Compute indices for the longest increasing subsequence.

Time:  $\mathcal{O}(N \log N)$

2932a0, 17 lines

```

template<class I> vi lis(const vector<I>& S) {
    if (S.empty()) return {};
    vi prev(sz(S));
    for (int i = 1; i < sz(S); i++) {
        if (prev[i] < S[i]) prev[i] = S[i];
        else {
            int j = lower_bound(all(prev), S[i]) - prev;
            if (j < sz(prev)) prev[j] = S[i];
        }
    }
    return prev;
}

```

```

typedef pair<I, int> p;
vector<p> res;
rep(i, 0, sz(S)) {
    // change 0 -> i for longest non-decreasing subsequence
    auto it = lower_bound(all(res), p{S[i], 0}); //f6ef94
    if (it == res.end()) res.emplace_back(), it = res.end()-1;
    *it = {S[i], i}; //26a0a3
    prev[i] = it == res.begin() ? 0 : (it-1)->second;
} //f2ee22
int L = sz(res), cur = res.back().second;
vi ans(L);
while (L--) ans[L] = cur, cur = prev[cur];
return ans;
} //2932a0

```

### FastKnapsack.h

**Description:** Given N non-negative integer weights w and a non-negative target t, computes the maximum  $S \leq t$  such that S is the sum of some subset of the weights.

Time:  $\mathcal{O}(N \max(w_i))$

b20ccc, 16 lines

```

int knapsack(vi w, int t) {
    int a = 0, b = 0, x;
    while (b < sz(w) && a + w[b] <= t) a += w[b++];
    if (b == sz(w)) return a;
    int m = *max_element(all(w));
    vi u, v(2*m, -1); //11fd10
    v[a+m-t] = b;
    rep(i, b, sz(w)) {
        u = v;
        rep(x, 0, m) v[x+w[i]] = max(v[x+w[i]], u[x]);
        for (x = 2*m; --x > m; ) rep(j, max(0, u[x]), v[x]) //51a6b1
            v[x-w[j]] = max(v[x-w[j]], j);
    } //d2bd39
    for (a = t; v[a+m-t] < 0; a--);
    return a;
} //b20ccc

```

### 10.3 Dynamic programming

KnuthDP.h

**Description:** When doing DP on intervals:  $a[i][j] = \min_{i < k < j} (a[i][k] + a[k][j]) + f(i, j)$ , where the (minimal) optimal  $k$  increases with both  $i$  and  $j$ , one can solve intervals in increasing order of length, and search  $k = p[i][j]$  for  $a[i][j]$  only between  $p[i][j-1]$  and  $p[i+1][j]$ . This is known as Knuth DP. Sufficient criteria for this are if  $f(b, c) \leq f(a, d)$  and  $f(a, c) + f(b, d) \leq f(a, d) + f(b, c)$  for all  $a \leq b \leq c \leq d$ . Consider also: LineContainer (ch. Data structures), monotone queues, ternary search.

Time:  $\mathcal{O}(N^2)$

### DivideAndConquerDP.h

**Description:** Given  $a[i] = \min_{lo(i) \leq k < hi(i)} (f(i, k))$  where the (minimal) optimal  $k$  increases with  $i$ , computes  $a[i]$  for  $i = L..R-1$ .

Time:  $\mathcal{O}((N + (hi - lo)) \log N)$

d38d2b, 18 lines

```

struct DP { // Modify at will:
    int lo(int ind) { return 0; } //ce585d
    int hi(int ind) { return ind; } //f742b2
    ll f(int ind, int k) { return dp[ind][k]; } //29ea0c
    void store(int ind, int k, ll v) { res[ind] = pii(k, v); }

    void rec(int L, int R, int LO, int HI) {
        if (L >= R) return;
        int mid = (L + R) >> 1; //13ddb0
        pair<ll, int> best(LLONG_MAX, LO);
        rep(k, max(LO, lo(mid)), min(HI, hi(mid)))
            best = min(best, make_pair(f(mid, k), k));
        store(mid, best.second, best.first);
        rec(L, mid, LO, best.second+1); //4993b6
    }
}

```

```

    rec(mid+1, R, best.second, HI);
} //116ea5
void solve(int L, int R) { rec(L, R, INT_MIN, INT_MAX); }
}; //d38d2b

```

## 10.4 Optimization tricks

`__builtin_ia32_ldmxcsr(40896);` disables denormals (which make floats 20x slower near their minimum value).

### 10.4.1 Bit hacks

- `x & -x` is the least bit in `x`.
- `for (int x = m; x; ) { --x &= m; ... }` loops over all subset masks of `m` (except `m` itself).
- `c = x&-x, r = x+c; (((r^x) >> 2)/c) | r` is the next number after `x` with the same number of bits set.
- `rep(b,0,K) rep(i,0,(1 << K))  
if (i & 1 << b) D[i] += D[i^(1 << b)];`  
computes all sums of subsets.

### 10.4.2 Pragmas

- `#pragma GCC optimize ("Ofast")` will make GCC auto-vectorize loops and optimizes floating points better.
- `#pragma GCC target ("avx2")` can double performance of vectorized code, but causes crashes on old machines.
- `#pragma GCC optimize ("trapv")` kills the program on integer overflows (but is really slow).

## FastInput.h

**Description:** Read an integer from stdin. Usage requires your program to pipe in input from file.

**Usage:** `./a.out < input.txt`

**Time:** About 5x as fast as `cin/scanf`.

7b3c70, 17 lines

```

inline char gc() { // like getchar()
    static char buf[1 << 16];
    static size_t bc, be;
    if (bc >= be) {
        buf[0] = 0, bc = 0;
        be = fread(buf, 1, sizeof(buf), stdin); //bba013
    } //e9a035
    return buf[bc++]; // returns 0 on EOF
} //0261eb

int readInt() {
    int a, c;
    while ((a = gc()) < 40);
    if (a == '-') return -readInt(); //bc51ee
    while ((c = gc()) >= 48) a = a * 10 + c - 48;
    return a - 48;
} //7b3c70

```