



University of Copenhagen

3 little 3 late

Asbjørn Lind, Elias Rasmussen Lolck, Thor Vejen Eriksen

NWERC 2024

November 22, 2024

Setup

INFO.tex

How to submit/debug

Remember:

- Fast input.
- Unsure of time limit? Generate simple max cases!
- Check memory limits.
- Check overflow!
- Turbo mega check the right file gets submitted.
- Compile (and run test cases) at least once with `dc`; strongly consider resolving warnings.
- Overflow?
- Make sure you are reading e.g. n and m in the right order.
- Do not have uninitialized variables!
- If WA/RE: print code. Take a quick walk. Maybe even rewrite everything. RE can mean MLE. Invalidated pointers/iterators?

During test session

- `setxkbmap dk/us`.
- That `bashrc`/`vimrc` works.
- Printing.
- Sending clarification.
- `cppreference`.
- CLI submission if it exists.
- Whitespace sensitivity in submissions.
- Return non-zero from main.
- Printing to `stderr` during otherwise correct submission.
- Source code size limit (if not stated by jury).
- Get MLE and check if it shows as RE.
- Check compile time limit.
- `__int128`.
- Check available binaries (yoinked from `kactl`): `echo $PATH | tr ':' ' ' | xargs ls | grep -v / | sort | uniq | tr ' ' '\n' |`

bashrc.sh

```
-----4b42a4
64a45f setxkbmap -option caps:escape
64a45f # fast:
92e6e6 Xset r rate 200 120
92e6e6 # normal:
efd146 Xset r rate 500 35
efd146 # debug compile (C++):
3b0f04 dc() {
53a367     bsnm=$(basename "$1" .cpp)
53a367     # EUC uses -std=gnu++20
```

```
8f434d command="g++ ${bsnm}.cpp -o $bsnm -Wshadow -Wall -g -
fsanitize=address,undefined -D_GLIBCXX_DEBUG -std=gnu
++20 -Wfatal-errors"
98f084 echo $command
26aae8 $command
1b7e88}
4b42a4 set -o vi
```

hash.sh

```
-----5246ca
d41d8c # hashes a file, ignoring whitespaces and comments
d41d8c # use for verifying that code is copied correctly
5246ca cpp -dD -P -fpreprocessed | tr -d '[:space:]' | md5sum |
cut -c-6
```

init.lua

```
-----1d7b4f
d8df16 local options = {
8cc3ed     cmdheight = 1,
a35030     ignorecase = true,
527981     mouse = "a",
b432b9     expandtab = true,
9720f0     shiftwidth = 4,
8129cf     tabstop = 4,
0276a2     cursorline = true,
02041e     number = true,
8e1996     relativenumber = true,
651c51     numberwidth = 1,
337bd1     signcolumn = "yes",
8ba69f     wrap = false,
58e240     scrolloff = 6,
a42cac     sidescrolloff = 6,
477b34     foldmethod="indent",
1bb339     foldlevel=99,
f8d18f     colorcolumn='80',
fd3a04}
fd3a04
fee201 local keymap = vim.api.nvim_set_keymap
cdad30 local ops = { noremap = true, silent = true }
cdad30
053dbf keymap("v", "<A-Down>", ":m '>+1<CR>gv=gv", ops)
ed4d76a keymap("v", "<A-Up>", ":m '<-2<CR>gv=gv", ops)
401f0e keymap("v", "p", "\"_dP", ops)
401f0e
b8fa45 for k, v in pairs(options) do
92cae6     vim.opt[k] = v
459d9e end
459d9e
a65a02 local function hashCurrentBuffer()
49b46b     local buffer_content = table.concat(vim.api.
nvim_buf_get_lines(0, 0, -1, false), "\n")
384df1     local command = "echo '".buffer_content..' ' | cpp -
dD -P -fpreprocessed | tr -d '[:space:]' | md5sum |
cut -c-6"
a60269     local hash = vim.fn.system(command)
6a349e     hash = hash:gsub("%s+", " ")
57f324     print("Buffer Hash: " .. hash)
37ba3e end
1d7b4f vim.api.nvim_create_user_command('Hash',
hashCurrentBuffer, {})
```

KACTL template

```
-----bd2055
d41d8c // in addition to template.h, kactl uses:
7ab427 #define rep(i,a,b) for(int i = a; i < (b); ++i)
793c8d #define sz(x) (int)(x).size()
c1f7c4 typedef pair<int,int> pii;
bd2055 typedef vector<int> vi;
```

template

```
-----d74cfd
d41d8c // #include <bits/stdc++.h>
ca417d using namespace std;
fbb4e1 typedef long long ll;
22323a #define all(x) (x).begin(), (x).end()
22323a
251bca int main() {
04007e     ios::sync_with_stdio(0); cin.tie(0);
d74cfd}
```

vimrc

```
-----88d5be
f112b5 se ch=1 ic mouse=a sw=4 ts=4 nu rnu nuw=4 nowrap so=6
siso=8 fdm=indent fdl=99 tm=100
2f1e84 ca Hash w !cpp -dD -P -fpreprocessed \ | tr -d '[:space
:]' \ | md5sum \ | cut -c-6
5c5f32 vnoremap <silent> <A-Down> :m '>+1<CR>gv=gv
7c854e vnoremap <silent> <A-Up> :m '<-2<CR>gv=gv
88d5be vnoremap <silent> p "_dP
```

Data_structures

Disjoint Set Union

Description: Classic DSU using path compression and union by rank. `unite` returns true iff `u` and `v` were disjoint.
Usage: `Dsu d(n); d.unite(a, b); d.find(a);`
Complexity: `find()`, `unite()` are amortized $\mathcal{O}(\alpha(n))$, where $\alpha(n)$ is the inverse Ackermann function. Basically $\mathcal{O}(1)$.

```
-----5168ab
e9a6d7 struct Dsu {
20b72f     vector<int> p, rank;
b849ca     Dsu(int n) {
743598         p.resize(n); rank.resize(n, 0);
53d18c         iota(p.begin(), p.end(), 0);
b27a44     }
aef80c     int find(int x) {
f5ffdc         return p[x] == x ? x : p[x] = find(p[x]);
a1cfa1     }
fecab9     bool unite(int u, int v) {
675018         if ((u = find(u)) == (v = find(v))) return false;
de3de6         if (rank[u] < rank[v]) swap(u, v);
859abb         p[v] = u;
0e6393         rank[u] += rank[v] == rank[v];
49561c         return true;
052f6f     }
5168ab};
```

Li-Chao tree

Description: Contianer of lines, online insertion/querying. Retrieve the line f with minimum $f(x)$ for a given x .
Usage: `LCT lct(n); lct.insert(line, 0, n); lct.query(x, 0, n);`
Complexity: $\mathcal{O}(\log n)$ per insertion/query

```
-----ba9fe6
4bbcd8 struct Line { ll a, b; ll f(ll x) { return a * x + b; }
};
7988a9 constexpr const Line LINF { 0, 1LL << 60 };
ffb13a struct LCT {
358a49     vector<Line> v; // coord-compression: modify v[x] ->
v[conert(x)]
f584d6     LCT(int size) { v.resize(size + 1, LINF); }
d9141d     void insert(Line line, int l, int r) {
```

```
eaba83      if (l > r) return;
fb3606      int mid = (l + r) >> 1;
fa90d9      if (line.f(mid) < v[mid].f(mid)) swap(line, v[mid]);
9e933f      if (line.f(l) < v[mid].f(l)) insert(line, l, mid -
1);
17cdcd      else insert(line, mid + 1, r);
3dae75      }
995afc      Line query(int x, int l, int r) {
d3ae28      if (l > r) return LINF;
3e2cdb      int mid = (l + r) >> 1;
5b4a90      if (x == mid) return v[mid]; // faster on avg. - not
necessary
9c3466      if (x < mid) return best_of(v[mid], query(x, l, mid
- 1), x);
0065c4      return best_of(v[mid], query(x, mid + 1, r), x);
9d9761      }
c445dc      Line best_of(Line a, Line b, ll x) { return a.f(x) < b
.f(x) ? a : b; }

ba9fe6};
```

Rollback Union Find

Description: Yoinked from kactl. Disjoint-set data structure with undo. If undo is not needed, skip st, time() and rollback().
Usage: int t = uf.time(); ...; uf.rollback(t);
Complexity: $\mathcal{O}(\log n)$.

```
-----de4ad0
47a5e9struct RollbackUF {
32cc46    vi e; vector<pii> st;
66f6eb    RollbackUF(int n) : e(n, -1) {}
df49a1    int size(int x) { return -e[find(x)]; }
f73c5d    int find(int x) { return e[x] < 0 ? x : find(e[x]); }
821d77    int time() { return sz(st); }
154abb    void rollback(int t) {
d4a702        for (int i = time(); i --> t;)
96b4b9            e[st[i].first] = st[i].second;
93333b        st.resize(t);
e7fe82    }
3f4ca5    bool join(int a, int b) {
9dd20b        a = find(a), b = find(b);
081d43        if (a == b) return false;
40458e        if (e[a] > e[b]) swap(a, b);
3aaa7c        st.push_back({a, e[a]});
5f71eb        st.push_back({b, e[b]});
fa6967        e[a] += e[b]; e[b] = a;
21e12e        return true;
f0724e    }
de4ad0};
```

Fenwick tree

Description: Computes prefix sums and single element updates. Uses 0-indexing.
Usage: Fen f(n); f.update(ind, val); f.query(ind); f.lower_bound(sum);
Complexity: $\mathcal{O}(\log n)$ per update/query

```
-----1743e1
92fe3cstruct Fen {
04c831    vector<ll> v;
15fd8d    Fen(int s) : v(s, 0) {}
f76ea5    void update(int ind, ll val) {
4238a4        for (; ind < (int) v.size(); ind |= ind + 1) v[ind]
+= val;
222f2c    }
7b09a2    ll query(int ind) { // [0, ind), ind < 0 returns 0
37f317        ll res = 0;
cc7a2a        for (; ind > 0; ind &= ind - 1) res += v[ind - 1];
552720        // operation can be modified
1c3977        return res;
}
```

```
348a7a      int lower_bound(ll sum) { // returns first i with
query(i + 1) >= sum, n if not found
1f0b41          int ind = 0;
fe1e46          for (int p = 1 << 25; p; p >= 1) // 1 << 25 can be
lowered to ceil(log2(v.size()))
a63f8c              if (ind + p <= (int) v.size() && v[ind + p - 1] <
sum)
a9fd291                  sum -= v[(ind += p) - 1];
15c383              return ind;
ac78de          }
1743e1};
```

Fast hash map

Description: 3x faster hash map, 1.5x more memory usage, similar API to std::unordered_map. Initial capacity, if provided, must be power of 2.
Usage: hash_map<key_t, val_t> mp; mp[key] = val; mp.find(key); mp.begin(); mp.end(); mp.erase(key); mp.size();
Complexity: $\mathcal{O}(1)$ per operation on average.

```
-----c7be5a
d41d8c// #include <bits/extc++.h>
d41d8c
75f3c2struct chash {
0d8969    const uint64_t C = 1l(4e18 * acos(0)) | 71;
16eb60    ll operator () (ll x) const { return __builtin_bswap64
(x * C); }
cdd37e};
cdd37e
c7be5atemplate <typename KEY_T, typename VAL_T> using hash_map
= __gnu_pbds::gp_hash_table<KEY_T, VAL_T, chash>;
```

Implicit 2D segment tree

Description: Classic implicit 2D segment tree taken from my solution to IOI game 2013. It is in rough shape, but it works. Designed to be [inclusive, exclusive). It is old and looks shady, only rely slightly on it, maybe even just make a new one if you need one.
Usage: See usage example at the bottom.
Complexity: $\mathcal{O}(\log^2 n)$ per operation *I think*.

```
-----ae92aa
299b05constexpr const int MX_RC = 1 << 30;
299b05
a3032estruct Inner {
493223    long long val;
140419    int lv, rv;
4cb72f    Inner* lc,* rc;
f24ef1    Inner(long long _val, int _l, int _r) :
3cdb99        val(_val), lv(_l), rv(_r), lc(nullptr), rc(nullptr)
{ }
ab764d    ~Inner() {
60af3c        delete(lc);
2e9793        delete(rc);
02da3e    }
b8d074    void update(int ind, long long nev, int l = 0, int r =
MX_RC) {
00ea11        if (!(r - l - 1)) {
ca7a61            assert(lv == l && rv == r);
226ff1            assert(ind == l);
bac672            val = nev;
a41337            return;
b0b081        }
78f219        int mid = (l + r) >> 1;
23eba4        if (ind < mid) {
286913            if (lc) {
246e24                if (lc->lv != l || lc->rv != mid) {
3a66b2                    Inner* tmp = lc;
926f8a                    lc = new Inner(0, l, mid);
c8fd20                    (tmp->lv < ((l + mid) >> 1) ? lc->lc : lc->rc)
= tmp;
653efd                }
94bb73                lc->update(ind, nev, l, mid);
e88f6e            }
}
```

```
a69813        } else lc = new Inner(nev, ind, ind + 1);
1d67a7    } else {
a18480        if (rc) {
849ead            if (rc->lv != mid || rc->rv != r) {
3d82c2                Inner* tmp = rc;
08e48b                rc = new Inner(0, mid, r);
3cf492                (tmp->lv < ((mid + r) >> 1) ? rc->lc : rc->rc)
= tmp;
18683f            }
1ddbfc            rc->update(ind, nev, mid, r);
637a18        } else rc = new Inner(nev, ind, ind + 1);
1ea254    }
97c33a    val = std::gcd(lc ? lc->val : 0, rc ? rc->val : 0);
45be42    }
66c546    long long query(int tl, int tr, int l = 0, int r =
MX_RC) {
a00435        if (l >= tr || r <= tl) return 0;
edccb1        if (!(rv - lv - 1)) {
81886f            if (lv >= tr || rv <= tl) return 0;
6228a6            return val;
c4aa5d        }
0dbaae        assert(l == lv && r == rv);
791073        if (l >= tl && r <= tr) return val;
88a336        int mid = (l + r) >> 1;
b766e2        return std::gcd(lc ? lc->query(tl, tr, l, mid) : 0,
rc ? rc->query(tl, tr, mid, r) : 0);
3c130a    }
f0c650    void fill(Inner* source) {
a568f5        val = source->val;
13392a        if (!(lv - rv - 1)) return;
221c61        if (source->lc) {
e7c4fa            lc = new Inner(source->lc->val, source->lc->lv,
source->lc->rv);
74f1f6            lc->fill(source->lc);
071c5b        }
ad50a0        if (source->rc) {
9adebe            rc = new Inner(source->rc->val, source->rc->lv,
source->rc->rv);
946ac9            rc->fill(source->rc);
b8bed9        }
c66f9e    }
ca99e3};
ca99e3
fc64b2struct Outer {
5d6d11    Inner* inner;
int lv, rv;
999186    Outer* lc,* rc;
9777b6    Outer(Inner* _inner, int _l, int _r) :
0d648e        inner(_inner), lv(_l), rv(_r), lc(nullptr), rc(nullptr)
{ }
b56d7c    }
6940a1    void update(int ind_outer, int ind_inner, long long
nev, int l = 0, int r = MX_RC) {
262130        if (!(r - l - 1)) {
a44e79            assert(lv == l && rv == r);
42e19d            assert(ind_outer == l);
5de54d            assert(inner);
084529            inner->update(ind_inner, nev);
01581a            return;
66ce83        }
922ab4        int mid = (l + r) >> 1;
4a146c        if (ind_outer < mid) {
ad897f            if (lc) {
033f38                if (lc->lv != l || lc->rv != mid) {
8382cb                    Outer* tmp = lc;
90c8a1                    lc = new Outer(new Inner(0, 0, MX_RC), l, mid)
;
68043c                    lc->inner->fill(tmp->inner);
bb30e9                    (tmp->lv < ((l + mid) >> 1) ? lc->lc : lc->rc)
= tmp;
dd2110                }
238e44                lc->update(ind_outer, ind_inner, nev, l, mid);
1b68a4            }
}
```

```

    } else {
        lc = new Outer(new Inner(0, 0, MX_RC), ind_outer
, ind_outer + 1);
        lc->inner->update(ind_inner, nev);
    }
} else {
    if (rc) {
        if (rc->lv != mid || rc->rv != r) {
            Outer* tmp = rc;
            rc = new Outer(new Inner(0, 0, MX_RC), mid, r)
;
            rc->inner->fill(tmp->inner);
            (tmp->lv < ((mid + r) >> 1) ? rc->lc : rc->rc)
= tmp;
        }
        rc->update(ind_outer, ind_inner, nev, mid, r);
    } else {
        rc = new Outer(new Inner(nev, 0, MX_RC),
ind_outer, ind_outer + 1);
        rc->inner->update(ind_inner, nev);
    }
}
inner->update(ind_inner, std::gcd(
lc ? lc->inner->query(ind_inner, ind_inner + 1) : 0,
rc ? rc->inner->query(ind_inner, ind_inner + 1) : 0)
);
}
}
long long query(int tl_outer, int tr_outer, int
tl_inner, int tr_inner, int l = 0, int r = MX_RC) {
    if (l >= tr_outer || r <= tl_outer) return 0;
    if (!(rv - lv - 1)) {
        if (lv >= tr_outer || rv <= tl_outer) return 0;
        return inner->query(tl_inner, tr_inner);
    }
    assert(l == lv && r == rv);
    if (l >= tl_outer && r <= tr_outer)
        return inner->query(tl_inner, tr_inner);
    int mid = (l + r) >> 1;
    return std::gcd(
        lc ? lc->query(tl_outer, tr_outer, tl_inner,
tr_inner, l, mid) : 0,
        rc ? rc->query(tl_outer, tr_outer, tl_inner,
tr_inner, mid, r) : 0);
}
}
aae6cb
82e377};
82e377
82e377// this is how it has been used in the solution to IOI
game 2013
319c4cOuter root(new Inner(0, 0, MX_RC), 0, MX_RC);
1b943bvoid update(int r, int c, long long k) {
b78d10    root.update(r, c, k);
a445e8}
07e107long long calculate(int r_l, int c_l, int r_r, int c_r)
{
2f2876    return root.query(r_l, r_r + 1, c_l, c_r + 1);
ae92aa}

```

Lazy segment tree

Description: Zero-indexed, bounds are [l, r), operations can be mod-ified. $\mathcal{O}(\log n)$ find_first and the like can be implemented by checking bounds, then checking left tree, then right tree, recursively.
Usage: Lazy_segtree seg(n); seg.update(l, r, val); seg.query(l, r);
Complexity: $\mathcal{O}(\log n)$ per update/query

```

-----39cb07
186ef1struct Lazy_segtree {
142517    typedef ll T; // change type here
7bd302    typedef ll LAZY_T; // change type here
b372d3    static constexpr T unit = 0; // change unit here

```

```

962f5f    static constexpr LAZY_T lazy_unit = 0; // change lazy
unit here
7f83a4    T f(T l, T r) { return l + r; } // change operation
here
c9312a    void push(int now, int l, int r) {
b8a6a8        if (w[now] == lazy_unit) return;
2624b4        v[now] += w[now] * (r - l); // operation can be
modified
baf9bc        if (r - l - 1)
a7681c            w[now * 2 + 1].first += w[now],
693215            w[now * 2 + 2].first += w[now];
bcea3b            w[now] = lazy_unit;
6dfa7c    }
b801ca    int size;
c396cf    vector<T> v;
ce40fb    vector<LAZY_T> w;
37a544    Lazy_segtree(int s = 0) : size(s ? 1 << (32 -
__builtin_clz(s)) : 0), v(size << 1, unit), w(size <<
1, lazy_unit) {}
de9785    template<typename U> void update(int l, int r, U val)
{ update(l, r, val, 0, 0, size); }
677cbe    T query(int l, int r) { return query(l, r, 0, 0, size)
; }
9f1fbb    template<typename U> void update(int tl, int tr, U
val, int now, int l, int r) {
a8f911        push(now, l, r);
0733ac        if (l >= tr || r <= tl) return;
7fba5d        if (l >= tl && r <= tr) {
7fba5d            // this does not *have* to accumulate, push is
called before this:
95311e            w[now] += val; // operation can be modified
0745ca            push(now, l, r);
a906ff            return;
92ee41        }
d9ba9d        int mid = (l + r) >> 1;
ef2051        update(tl, tr, val, now * 2 + 1, l, mid);
e67970        update(tl, tr, val, now * 2 + 2, mid, r);
d8795a        v[now] = f(v[now * 2 + 1], v[now * 2 + 2]);
162f13    }
T query(int tl, int tr, int now, int l, int r) {
99a91a        push(now, l, r);
eeefb9        if (l >= tr || r <= tl) return unit;
15115d        if (l >= tl && r <= tr) return v[now];
a2b812        int mid = (l + r) >> 1;
a2df56        return f(query(tl, tr, now * 2 + 1, l, mid), query(
2c75e7        tl, tr, now * 2 + 2, mid, r));
293acc    }
546d89    template<typename U> void build(const vector<U>& a)
{
111070        for (int i = 0; i < (int) a.size(); i++) v[size - 1
+ i] = a[i]; // operation can be modified
24e497        for (int i = size - 2; i >= 0; i--) v[i] = f(v[i * 2
+ 1], v[i * 2 + 2]);
db93fb    }
69cb07};

```

Matrix

Description: Yinked from kactl. Basic operations on square matrices.
Usage: Matrix<int, 3> A; A.d = {{{{1,2,3}}, {{4,5,6}}, {{7,8,9}}}}; vector<int> vec = {1,2,3}; vec = (A^N) * vec;
Complexity: $\mathcal{O}(n^3)$ per multiplication, $\mathcal{O}(n^3 \log p)$ per exponentiation.

```

c43c7d
-----3237d5
a9546ftemplate<class T, int N> struct Matrix {
3ef34d    typedef Matrix M;
e08add    array<array<T, N>, N> d{};
c6c78f    M operator*(const M& m) const {
1aac3d        M a;
5aa3ab        rep(i,0,N) rep(j,0,N)
b57a4b            rep(k,0,N) a.d[i][j] += d[i][k]*m.d[k][j];

```

```

5a335a    return a;
ae016f    }
db76e7    vector<T> operator*(const vector<T>& vec) const {
8c6915        vector<T> ret(N);
c690c4        rep(i,0,N) rep(j,0,N) ret[i] += d[i][j] * vec[j];
85dfd1        return ret;
2f2bd4    }
d7ecae    M operator^(ll p) const {
8c6dad        assert(p >= 0);
75b035        M a, b(*this);
31b02f        rep(i,0,N) a.d[i][i] = 1;
6e9149        while (p) {
56e68d            if (p&1) a = a*b;
74866d            b = b*b;
12d234            p >>= 1;
1c0b4f        }
3182fe        return a;
c85f9b    }
c43c7d};

```

Ordered Map

Description: extc++.h order statistics tree. find_by_order returns an iterator to the k th element (0-indexed), order_of_key returns the index of the element (0-indexed), i.e. the number of elements less than the argument.
Usage: ordered_set<int> s; s.insert(1); s.insert(2); *s.find_by_order(0) = 3; s.erase(3); s.order_of_key(2);
Complexity: Everything is $\mathcal{O}(\log n)$.

```

-----93d061
d41d8c// #include <bits/extc++.h>
d41d8c// if judge does not have extc++.h, use:
d41d8c// #include <ext/pb_ds/assoc_container.hpp>
d41d8c// #include <ext/pb_ds/tree_policy.hpp>
0d73c5using namespace __gnu_pbds;
647225template<typename T> using ordered_set = tree<T,
null_type, less<T>, rb_tree_tag,
tree_order_statistics_node_update>;
d8888dtemplate<typename T, typename U> using ordered_map =
tree<T, U, less<T>, rb_tree_tag,
tree_order_statistics_node_update>;
d8888d
d8888d// yeet from kactl:
b3a888void example() {
ce24bb    ordered_set<int> t, t2; t.insert(8);
26f820    auto it = t.insert(10).first;
4f4555    assert(it == t.lower_bound(9));
aa0d3a    assert(t.order_of_key(10) == 1);
de8f5b    assert(t.order_of_key(11) == 2);
080225    assert(*t.find_by_order(0) == 8);
86be9f    t.join(t2); // assuming T < T2 or T > T2, merge t2
into t
93d061}

```

Persistent segment tree

Description: Zero-indexed, bounds are [l, r), operations can be mod-ified. update(...) returns a pointer to a new tree with the applied update, all other trees remain unchanged. $\mathcal{O}(\log n)$ find_first and the like can be implemented by checking bounds, then checking left tree, then right tree, recursively.
Usage: Node* root = build(arr, 0, n); Node* another_root = update(root, ind, val, 0, n); query(some_root, l, r, 0, n).val; Node* empty_root = nullptr; Node* another_version = update(empty_root, ind, val, 0, n);
Complexity: $\mathcal{O}(\log n)$ per update/query, $\mathcal{O}(n)$ per build

```

-----3237d5
bf28eastruct Node {
24f2c2    Node* l,* r;
1eddfe    int val; // i.e. data
9f97da    Node(int _v) : l(nullptr), r(nullptr), val(_v) {}

```

```
ad01ea Node(Node* _l, Node* _r) : l(_l), r(_r), val(0) {
ad01ea     // i.e. merge two nodes:
6cb990     if (l) val += l->val;
bdea62     if (r) val += r->val;
97b9e8 }
089802};
089802
089802// slightly more memory, much faster:
3e798e template <typename... ARGS> Node* new_node(ARGS&&...
    args) {
196c33     static deque <Node> pool;
17bd12     pool.emplace_back(forward <ARGS> (args)...);
cc621a     return &pool.back();
b16dc2}
b16dc2// slightly less memory, much slower:
b16dc2// #define new_node(...) new Node(__VA_ARGS__)
b16dc2
b16dc2// optional:
a8e5c9 Node* build(const vector <int>& a, int l, int r) {
085265     if (!(r - l - 1)) return new_node(a[l]);
c5e761     int mid = (l + r) >> 1;
80c83f     return new_node(build(a, l, mid), build(a, mid, r));
7b790d}
7b790d
7b790d// can be called with node == nullptr
9954ai Node* update(Node* node, int ind, int val, int l, int r)
    {
f8778c     if (!(r - l - 1)) return new_node(val); // i.e. point
        update
2b5823     int mid = (l + r) >> 1;
7c550e     Node* lf = node ? node->l : nullptr;
28db3c     Node* rg = node ? node->r : nullptr;
d13bbf     return new_node
49ef9f         (ind < mid ? update(lf, ind, val, l, mid) : lf,
8e334d         ind >= mid ? update(rg, ind, val, mid, r) : rg);
7d1cf8}
7d1cf8
ea439d Node query(Node* node, int tl, int tr, int l, int r) {
d3c68e     if (l >= tr || r <= tl || !node) return Node(0); // i.
        e. empty node
24ae6b     if (l >= tl && r <= tr) return *node;
27c8e9     int mid = (l + r) >> 1;
162e7e     Node lf = query(node->l, tl, tr, l, mid);
9e1e8a     Node rg = query(node->r, tl, tr, mid, r);
39468c     return Node(&lf, &rg);
323745}
```

Segment tree

Description: Zero-indexed, bounds are [l, r), operations can be modified. $\mathcal{O}(\log n)$ find_first and the like can be implemented by checking bounds, then checking left tree, then right tree, recursively.
Usage: Segtree seg(n); seg.update(ind, val); seg.query(l, r);
Complexity: $\mathcal{O}(\log n)$ per update/query.

```
-----f40463
1a258c struct Segtree {
134fc2     typedef ll T; // change type here
47b331     static constexpr T unit = 0; // change unit here
07bf9f     T f(T l, T r) { return l + r; } // change operation
        here
d1107d     int size;
fded6e     vector <T> v;
031350     Segtree(int s = 0) : size(s ? 1 << (32 - __builtin_clz
        (s)) : 0), v(size << 1, unit) {}
65274c     void update(int ind, T val) { update(ind, val, 0, 0,
        size); }
d6eb05     T query(int l, int r) { return query(l, r, 0, 0, size)
        ; }
0e0910     void update(int ind, T val, int now, int l, int r) {
3cbe11         if (!(r - l - 1)) { v[now] = val; return; } //
            operation can be modified
e3fe0c         int mid = (l + r) >> 1;
```

```
84ed9a     if (ind < mid) update(ind, val, now * 2 + 1, l, mid)
        ;
aedcc0     else update(ind, val, now * 2 + 2, mid, r);
0c1e51     v[now] = f(v[now * 2 + 1], v[now * 2 + 2]);
d6c412 }
b77366 T query(int tl, int tr, int now, int l, int r) {
0ad629     if (l >= tr || r <= tl) return unit;
c6cec7     if (l >= tl && r <= tr) return v[now];
0651cf     int mid = (l + r) >> 1;
77a729     return f(query(tl, tr, now * 2 + 1, l, mid), query(
        tl, tr, now * 2 + 2, mid, r));
fb82ee }
e06a94 template <typename U> void build(const vector <U>& a)
    {
005858     for (int i = 0; i < (int) a.size(); i++) v[size - 1
        + i] = a[i]; // operation can be modified
96a3b6     for (int i = size - 2; i >= 0; i--) v[i] = f(v[i * 2
        + 1], v[i * 2 + 2]);
0e4fbc }
f40463};
```

Sparse table

Description: Yoinked from kactl. Classic sparse table, implemented with range minimum queries, can be modified.
Usage: Sparse s(vec); s.query(a, b);
Complexity: $\mathcal{O}(|V| \log |V| + Q)$.

```
-----5b1135
e15847 template<class T> struct Sparse {
f1bb87     vector<vector<T>> jmp;
7c0bd0     Sparse(const vector<T>& V) : jmp(1, V) {
9d924a         for (int pw = 1, k = 1; pw * 2 <= sz(V); pw *= 2, ++
            k) {
4d9c0c             jmp.emplace_back(sz(V) - pw * 2 + 1);
a5cbfb             rep(j, 0, sz(jmp[k]))
80e3af                 jmp[k][j] = min(jmp[k - 1][j], jmp[k - 1][j + pw
                ]);
                }
2e7366     }
0be414 }
09c287 T query(int a, int b) { // interval [a, b)
68a824     assert(a < b); // or return inf if a == b
ac3fda     int dep = 31 - __builtin_clz(b - a);
d9d00b     return min(jmp[dep][a], jmp[dep][b - (1 << dep)]);
4f0efb }
5b1135};
```

Treap

Description: Yoinked from kactl. A short self-balancing tree. It acts as a sequential container with log-time splits/joins, and is easy to augment with additional data.
Complexity: $\mathcal{O}(\log n)$ operations.

```
-----9556fc
bf28ea struct Node {
09cf42     Node *l = 0, *r = 0;
6098a7     int val, y, c = 1;
1e3bd6     Node(int val) : val(val), y(rand()) {}
829930     void recalc();
daabb7};
daabb7
6c5593 int cnt(Node* n) { return n ? n->c : 0; }
371cf9 void Node::recalc() { c = cnt(l) + cnt(r) + 1; }
371cf9
6b5795 template<class F> void each(Node* n, F f) {
19c27d     if (n) { each(n->l, f); f(n->val); each(n->r, f); }
cfbf7f }
cfbf7f
0452f8 pair<Node*, Node*> split(Node* n, int k) {
818a92     if (!n) return {};
38e9ec     if (cnt(n->l) >= k) { // "n->val >= k" for lower_bound
        (k)
a9e21e         auto pa = split(n->l, k);
```

```
b3ae32     n->l = pa.second;
e89f16     n->recalc();
91a1e4     return {pa.first, n};
94ec96 } else {
b44179     auto pa = split(n->r, k - cnt(n->l) - 1); // and
        just "k"
80e6ae     n->r = pa.first;
db3d30     n->recalc();
56aeda     return {n, pa.second};
52e0e9 }
ead756}
ead756
c22ce2 Node* merge(Node* l, Node* r) {
9eb9c7     if (!l) return r;
060405     if (!r) return l;
51ba0c     if (l->y > r->y) {
4fbb1d         l->r = merge(l->r, r);
dbdd77         l->recalc();
154058         return l;
954f5c     } else {
046fe4         r->l = merge(l, r->l);
813afc         r->recalc();
cc9743         return r;
bbfab8     }
473a83 }
473a83
bd3837 Node* ins(Node* t, Node* n, int pos) {
e7b5ee     auto pa = split(t, pos);
4be5c5     return merge(merge(pa.first, n), pa.second);
148cb1}
148cb1
148cb1// Example application: move the range [l, r) to index k
5a0384 void move(Node*& t, int l, int r, int k) {
919f55     Node *a, *b, *c;
8e6808     tie(a, b) = split(t, l); tie(b, c) = split(b, r - l);
ae6bc0     if (k <= l) t = merge(ins(a, b, k), c);
55d03b     else t = merge(a, ins(c, b, k - r));
956fzc }
```

Wavelet tree

Description: Taken from https://ideone.com/Tkters. k -th smallest element in a range. Count number of elements less than or equal to k in a range. Count number of elements equal to k in a range.
Usage: wavelet_tree wt(arr, arr+n, 1, 1000000000); wt.kth(l, r, k); wt.LTE(l, r, k); wt.count(l, r, k);
Complexity: $\mathcal{O}(\log n)$ per query

```
-----364273
137ebf struct wavelet_tree{
2f784e     #define vi vector<int>
6a3389     #define pb push_back
bd5515     int lo, hi;
441687     wavelet_tree *l, *r;
d7a498     vi b;
d7a498
d7a498     //nos are in range [x,y]
d7a498     //array indices are [from, to)
4907d3     wavelet_tree(int *from, int *to, int x, int y){
50c38b         lo = x, hi = y;
15e543         if(lo == hi or from >= to) return;
034eb1         int mid = (lo+hi)/2;
276c4a         auto f = [mid](int x){
4d4ca8             return x <= mid;
dc9b96         };
290aa3         b.reserve(to-from+1);
80c53a         b.pb(0);
55caf2         for(auto it = from; it != to; it++)
9e0a5f             b.pb(b.back() + f(*it));
9e0a5f         //see how lambda function is used here
f87134         auto pivot = stable_partition(from, to, f);
834105         l = new wavelet_tree(from, pivot, lo, mid);
765e4a         r = new wavelet_tree(pivot, to, mid+1, hi);
```



```

    }
    //kth smallest element in [l, r]
    int kth(int l, int r, int k){
        if(l > r) return 0;
        if(lo == hi) return lo;
        int inLeft = b[r] - b[l-1];
        int lb = b[l-1]; //amt of nos in first (l-1) nos
        that go in left
        int rb = b[r]; //amt of nos in first (r) nos that go
        in left
        if(k <= inLeft) return this->l->kth(lb+1, rb , k);
        return this->r->kth(l-lb, r-rb, k-inLeft);
    }
    //count of nos in [l, r] Less than or equal to k
    int LTE(int l, int r, int k) {
        if(l > r or k < lo) return 0;
        if(hi <= k) return r - l + 1;
        int lb = b[l-1], rb = b[r];
        return this->l->LTE(lb+1, rb, k) + this->r->LTE(l-lb
        , r-rb, k);
    }
    //count of nos in [l, r] equal to k
    int count(int l, int r, int k) {
        if(l > r or k < lo or k > hi) return 0;
        if(lo == hi) return r - l + 1;
        int lb = b[l-1], rb = b[r], mid = (lo+hi)/2;
        if(k <= mid) return this->l->count(lb+1, rb, k);
        return this->r->count(l-lb, r-rb, k);
    }
    ~wavelet_tree(){
        delete l;
        delete r;
    }
};
64273};

```

Geometry

3D convex hull

Description: Yoinked from kactl. 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.
Complexity: $\mathcal{O}(n^2)$.

```

-----5b45fc
d41d8c // #include "Point_3D.h"
d41d8c
b8e08b typedef Point3D<double> P3;
b8e08b
6aa2ed struct PR {
cc2473 void ins(int x) { (a == -1 ? a : b) = x; }
e28e42 void rem(int x) { (a == x ? a : b) = -1; }
531490 int cnt() { return (a != -1) + (b != -1); }
5178b5 int a, b;
9a9457};
9a9457
538b68 struct F { P3 q; int a, b, c; };
538b68
7d6924 vector<F> hull3d(const vector<P3>& A) {
1d7f45 assert(sz(A) >= 4);
39c3b5 vector<vector<PR>> E(sz(A), vector<PR>(sz(A), {-1,
-1}));
39ded9#define E(x,y) E[f.x][f.y]
6eec88 vector<F> FS;
9469d2 auto mf = [&](int i, int j, int k, int l) {
47e4ee P3 q = (A[j] - A[i]).cross((A[k] - A[i]));
60a935 if (q.dot(A[l]) > q.dot(A[i]))
d6434b q = q * -1;

```

```

ed7472 F f{q, i, j, k};
dd2b5a E(a,b).ins(k); E(a,c).ins(j); E(b,c).ins(i);
d2c39f FS.push_back(f);
f13ccf };
411dfe rep(i,0,4) rep(j,i+1,4) rep(k,j+1,4)
489c42 mf(i, j, k, 6 - i - j - k);
489c42
42c30d rep(i,4,sz(A)) {
b33224 rep(j,0,sz(FS)) {
77d954 F f = FS[j];
c1b7a2 if(f.q.dot(A[i]) > f.q.dot(A[f.a])) {
d54d8c E(a,b).rem(f.c);
6ed4b4 E(a,c).rem(f.b);
5384c9 E(b,c).rem(f.a);
2eb5b4 swap(FS[j--], FS.back());
3244b8 FS.pop_back();
40e2cb }
66122d }
47a0d8 int nw = sz(FS);
930bd5 rep(j,0,nw) {
5d88f4 F f = FS[j];
460e4f#define C(a, b, c) if (E(a,b).cnt() != 2) mf(f.a, f.b, i
, f.c);
cccf10 C(a, b, c); C(a, c, b); C(b, c, a);
9bd3f7 }
c8c803 }
29960f for (F& it : FS) if ((A[it.b] - A[it.a]).cross(
3622d0 A[it.c] - A[it.a]).dot(it.q) <= 0) swap(it.c, it.b);
77f1dc return FS;
5b45fc};

```

Angle

Description: Yoinked from kactl. A class for ordering angles (as
represented by int points and a number of rotations around the origin).
Useful for rotational sweeping. Sometimes also represents points or
vectors.
Usage: vector<Angle> v = w[0], w[0].t360() ...; // sorted
int j = 0; rep(i,0,n) { while (v[j] < v[i].t180()) ++j; } //
sweeps j such that (j-i) represents the number of positively
oriented triangles with vertices at 0 and i

```

-----0f0602
755634 struct Angle {
022c62 int x, y;
76ee53 int t;
d184d3 Angle(int x, int y, int t=0) : x(x), y(y), t(t) {}
6c948b Angle operator-(Angle b) const { return {x-b.x, y-b.y,
t}; }
020235 int half() const {
b0dc15 assert(x || y);
9d5c24 return y < 0 || (y == 0 && x < 0);
39c79d }
12afc7 Angle t90() const { return {-y, x, t + (half() && x >=
0)}; }
05c9a0 Angle t180() const { return {-x, -y, t + half()}; }
3ad266 Angle t360() const { return {x, y, t + 1}; }
e258c0};
c1efa9 bool operator<(Angle a, Angle b) {
c1efa9 // add a.dist2() and b.dist2() to also compare
distances
a1f0ad return make_tuple(a.t, a.half(), a.y * (1l)b.x <
7d3b54 make_tuple(b.t, b.half(), a.x * (1l)b.y);
e78926}
e78926}
e78926// Given two points, this calculates the smallest angle
between
e78926// them, i.e., the angle that covers the defined line
segment.
ccb19a pair<Angle, Angle> segmentAngles(Angle a, Angle b) {
48d2ad if (b < a) swap(a, b);
c0377f return (b < a.t180() ?

```

```

4b88b6 make_pair(a, b) : make_pair(b, a.t360()));
eecd19}
c1d8eAngle operator+(Angle a, Angle b) { // point a + vector
b
c7f4a3 Angle r(a.x + b.x, a.y + b.y, a.t);
7cc5c9 if (a.t180() < r) r.t--;
e12799 return r.t180() < a ? r.t360() : r;
3fb429}
89aa95Angle angleDiff(Angle a, Angle b) { // angle b - angle a
99d8df int tu = b.t - a.t; a.t = b.t;
33f708 return {a.x*b.x + a.y*b.y, a.x*b.y - a.y*b.x, tu - (b
< a)};
0f0602}

```

Circle circle intersection

Description: Yoinked from kactl. Computes the pair of points at
which two circles intersect. Returns false in case of no intersection.
Complexity: $\mathcal{O}(1)$.

```

-----84d6d3
d41d8c // #include "Point.h"
d41d8c
6269ec typedef Point<double> P;
888549 bool circleInter(P a,P b,double r1,double r2,pair<P, P>*
out) {
7e53c0 if (a == b) { assert(r1 != r2); return false; }
2e6973 P vec = b - a;
deb755 double d2 = vec.dist2(), sum = r1+r2, dif = r1-r2,
7b252e p = (d2 + r1*r1 - r2*r2)/(d2*2), h2 = r1*r1 - p
*p*d2;
6ad02a if (sum*sum < d2 || dif*dif > d2) return false;
70d886 P mid = a + vec*p, per = vec.perp() * sqrt(fmax(0, h2)
/ d2);
3dd318 *out = {mid + per, mid - per};
212ced return true;
84d6d3}

```

Circle line intersection

Description: Yoinked from kactl. 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>.

```

-----e0cfba
d41d8c // #include "Point.h"
d41d8c
7dc51e template<class P>
0406ad vector<P> circleLine(P c, double r, P a, P b) {
cdd4b5 P ab = b - a, p = a + ab * (c-a).dot(ab) / ab.dist2();
e51742 double s = a.cross(b, c), h2 = r*r - s*s / ab.dist2();
64a27f if (h2 < 0) return {};
3d9ab3 if (h2 == 0) return {p};
1be847 P h = ab.unit() * sqrt(h2);
3b1a3f return {p - h, p + h};
e0cfba}

```

Circle polygon intersection

Description: Yoinked from kactl. Returns the area of the intersection
of a circle with a ccw polygon.
Complexity: $\mathcal{O}(n)$.

```

-----a1ee63
d41d8c // #include "Point.h"
d41d8c
6269ec typedef Point<double> P;
cf6463#define atan2 arg(p, q) atan2(p.cross(q), p.dot(q))
cf0422 double circlePoly(P c, double r, vector<P> ps) {
419913 auto tri = [&](P p, P q) {
a6cf13 auto r2 = r * r / 2;
c0445a P d = q - p;
702f07 auto a = d.dot(p)/d.dist2(), b = (p.dist2()-r*r)/d.
dist2();
4c3d03 auto det = a * a - b;

```

```
3710c6      if (det <= 0) return arg(p, q) * r2;
15e178      auto s = max(0., -a-sqrt(det)), t = min(1., -a+sqrt(
det));
1b08d3      if (t < 0 || 1 <= s) return arg(p, q) * r2;
a53ae4      P u = p + d * s, v = p + d * t;
f0b5ed      return arg(p,u) * r2 + u.cross(v)/2 + arg(v,q) * r2;
6470ed    };
dabb77    auto sum = 0.0;
48e7de    rep(i,0,sz(ps))
          sum += tri(ps[i] - c, ps[(i + 1) % sz(ps)] - c);
67d763    return sum;
a1ee63}
```

Circle tangents

Description: Yoinked from kactl. 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.

```
-----b0153d
d41d8c// #include "Point.h"
d41d8c
7dc51etemplate<class P>
e80549vector<pair<P, P>> tangents(P c1, double r1, P c2,
double r2) {
c7e310    P d = c2 - c1;
45b12a    double dr = r1 - r2, d2 = d.dist2(), h2 = d2 - dr * dr
;
c18727    if (d2 == 0 || h2 < 0) return {};
f9fd85    vector<pair<P, P>> out;
0072fe    for (double sign : {-1, 1}) {
48be0b        P v = (d * dr + d.perp() * sqrt(h2) * sign) / d2;
729d07        out.push_back({c1 + v * r1, c2 + v * r2});
41b560    }
2313ea    if (h2 == 0) out.pop_back();
054e70    return out;
b0153d}
```

Circumcircle

Description: Yoinked from kactl. The circumcirle 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.

```
-----1caa3a
d41d8c// #include "Point.h"
d41d8c
6289ectypedef Point<double> P;
5995a9double ccRadius(const P& A, const P& B, const P& C) {
2d2b60    return (B-A).dist()*(C-B).dist()*(A-C).dist()/
d37107        abs((B-A).cross(C-A))/2;
032e3d}
990f04P ccCenter(const P& A, const P& B, const P& C) {
d94b4d    P b = C-A, c = B-A;
fc3ed0    return A + (b*c.dist2()-c*b.dist2()).perp()/b.cross(c)
/2;
1caa3a}
```

Closest pair of points

Description: Yoinked from kactl. Finds the closest pair of points. **Complexity:** $\mathcal{O}(n \log n)$.

```
-----ac41a6
d41d8c// #include "Point.h"
d41d8c
2c0584typedef Point<ll> P;
7549f9pair<P, P> closest(vector<P> v) {
b02c53    assert(sz(v) > 1);
```

```
8f0c0e    set<P> S;
9e7fd4    sort(all(v), [](P a, P b) { return a.y < b.y; });
db620d    pair<ll, pair<P, P>> ret{LLONG_MAX, {P(), P()}};
2ac587    int j = 0;
14a5ea    for (P p : v) {
484ee7        P d{1 + (ll)sqrt(ret.first), 0};
0a3d44        while (v[j].y <= p.y - d.x) S.erase(v[j++]);
270154        auto lo = S.lower_bound(p - d), hi = S.upper_bound(p
+ d);
e75de8        for (; lo != hi; ++lo)
4128f5            ret = min(ret, {(lo - p).dist2(), {lo, p}});
af9942        S.insert(p);
a4382b    }
65a931    return ret.second;
ac41a6}
```

Convex hull

Description: Yoinked from kactl. 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. **Complexity:** $\mathcal{O}(n \log n)$.

```
-----310954
d41d8c// #include "Point.h"
d41d8c
2c0584typedef Point<ll> P;
af1648vector<P> convexHull(vector<P> pts) {
bf096e    if (sz(pts) <= 1) return pts;
086de3    sort(all(pts));
3e3497    vector<P> h(sz(pts)+1);
cc9643    int s = 0, t = 0;
8b7a3b    for (int it = 2; it--; s = --t, reverse(all(pts)))
2fd8c4        for (P p : pts) {
ef7eb7c            while (t >= s + 2 && h[t-2].cross(h[t-1], p) <= 0)
t--;
f4a7b9            h[t++] = p;
56ac78        }
b08f4b    return {h.begin(), h.begin() + t - (t == 2 && h[0] ==
h[1])};
310954}
```

Delaunay triangulation

Description: Yoinked from kactl. Computes the Delaunay triangulation of a set of points. Each circumcircle contains none of the input points. If any three points are collinear or any four are on the same circle, behavior is undefined. **Complexity:** $\mathcal{O}(n^2)$.

```
-----c0e7bc
d41d8c// #include "Point.h"
d41d8c// #include "3d_hull.h"
d41d8c
6abbcctemplate<class P, class F>
b5fdcavoid delaunay(vector<P>& ps, F trifun) {
6b1956    if (sz(ps) == 3) { int d = (ps[0].cross(ps[1], ps[2])
< 0);
0c9f52        trifun(0,1+d,2-d); }
d1e435    vector<P3> p3;
3ffe22    for (P p : ps) p3.emplace_back(p.x, p.y, p.dist2());
263f28    if (sz(ps) > 3) for(auto t:hull3d(p3)) if ((p3[t.b]-p3
[t.a]).
cf39a1        cross(p3[t.c]-p3[t.a]).dot(P3(0,0,1)) < 0)
c20439        trifun(t.a, t.c, t.b);
c0e7bc}
```

Dynamic Convex Hull

Description: Supports building a convex hull one point at a time. Viewing the convex hull along the way.

```
-----431bba
be520bstruct point {
0196fa    ll x, y;
```

```
f2e821    point(ll x=0, ll y=0): x(x), y(y) {}
0293d7    point operator-(const point &p) const { return point
(x-p.x, y-p.y); }
5daee5    point operator*(const ll k) const { return point(k*x
, k*y); }
f50d29    ll cross(const point &p) const { return x*p.y - p.x*
y; }
9d44db    bool operator<(const point &p) const { return x < p.
x || x == p.x && y < p.y; }
77f7cb};
77f7cb
2ce416bool above(set<point> &hull, point p, ll scale = 1) {
b5ac08    auto it = hull.lower_bound(point((p.x+scale-1)/scale
, 0));
75d58b    if (it == hull.end()) return true;
b7dcd8    if (p.y <= it->y*scale) return false;
fb2eae    if (it == hull.begin()) return true;
8a5eb9    auto jt = it--;
a7a017    return (p-*it*scale).cross(*jt-*it) < 0;
ecae32}
ecae32
2b34b3void add(set<point> &hull, point p) {
de0486    if (!above(hull, p)) return;
0a152b    auto pit = hull.insert(p).first;
3ba588    while (pit != hull.begin()) {
2b6ffc        auto it = prev(pit);
9de99b        if (it->y <= p.y || (it != hull.begin() && (*it
-*prev(it)).cross(*pit-*it) >= 0))
65eae8            hull.erase(it);
d03c84        else
87aeef            break;
f787d7    }
2f06a3    auto it = next(pit);
78b06b    while (it != hull.end()) {
d7d62c        if (next(it) != hull.end() && (*it-p).cross(*
next(it)-*it) >= 0)
b4dd19            hull.erase(it++);
6f504f        else
ae162a            break;
7a0510    }
431bba}
```

Hull diameter

Description: Yoinked from kactl. Returns the two points with max distance on a convex hull (ccw, no duplicate/collinear points). **Complexity:** $\mathcal{O}(n)$.

```
-----c571b8
d41d8c// #include "Point.h"
d41d8c
2c0584typedef Point<ll> P;
28b700array<P, 2> hullDiameter(vector<P> S) {
9bdd0c    int n = sz(S), j = n < 2 ? 0 : 1;
12ea1a    pair<ll, array<P, 2>> res({0, {S[0], S[0]}});
5c70ae    rep(i,0,j)
e5ff70        for (; j = (j + 1) % n) {
26329e            res = max(res, {(S[i] - S[j]).dist2(), {S[i], S[j
]}});
e7f091            if ((S[(j + 1) % n] - S[j]).cross(S[i + 1] - S[i])
>= 0)
49f898                break;
cf85e0        }
d9bfbba    return res.second;
c571b8}
```

Inside polygon

Description: Yoinked from kactl. Returns true if p lies within the polygon. If strict is true, it returns false for points on the boundary. The algorithm uses products in intermediate steps so watch out for overflow.

Usage: vector<P> v = {P{4,4}, P{1,2}, P{2,1}};
bool in = inPolygon(v, P{3, 3}, false);
Complexity: $\mathcal{O}(n)$.

```
-----2bf504
d41d8c// #include "Point.h"
d41d8c// #include "On_segment.h"
d41d8c// #include "Segment_distance.h"
d41d8c
7dc51etemplate<class P>
8cfa07bool inPolygon(vector<P> &p, P a, bool strict = true) {
68a46b     int cnt = 0, n = sz(p);
49a14b     rep(i,0,n) {
1c161f         P q = p[(i + 1) % n];
ca77bc         if (onSegment(p[i], q, a)) return !strict;
ca77bc         //or: if (segDist(p[i], q, a) <= eps) return !strict
;
8d185a         cnt ^= ((a.y<p[i].y) - (a.y<q.y)) * a.cross(p[i], q)
> 0;
ae1a12     }
3f2423     return cnt;
2bf504}
```

KD-tree

Description: Yoinked from kactl. 2D, can be extended to 3D. See comments for details.

```
-----bac5b0
d41d8c// #include "Point.h"
d41d8c
9ae6170typedef long long T;
d34771typedef Point<T> P;
3b6fa3const T INF = numeric_limits<T>::max();
3b6fa3
632da2bool on_x(const P& a, const P& b) { return a.x < b.x; }
624f7sbool on_y(const P& a, const P& b) { return a.y < b.y; }
624f7s
319cdastruct Node {
7cd9b0     P pt; // if this is a leaf, the single point in it
1149c5     T x0 = INF, x1 = -INF, y0 = INF, y1 = -INF; // bounds
3f2a9e     Node *first = 0, *second = 0;
3f2a9e
edbce8     T distance(const P& p) { // min squared distance to a point
71ed74         T x = (p.x < x0 ? x0 : p.x > x1 ? x1 : p.x);
6963e4         T y = (p.y < y0 ? y0 : p.y > y1 ? y1 : p.y);
4a1b67         return (P(x,y) - p).dist2();
1460d4     }
1460d4
3f46ab     Node(vector<P>&& vp) : pt(vp[0]) {
ae3536         for (P p : vp) {
516c49             x0 = min(x0, p.x); x1 = max(x1, p.x);
28bf16             y0 = min(y0, p.y); y1 = max(y1, p.y);
2be9c2         }
a1b63f         if (vp.size() > 1) {
a1b63f             // split on x if width >= height (not ideal...)
172b91             sort(all(vp), x1 - x0 >= y1 - y0 ? on_x : on_y);
172b91             // divide by taking half the array for each child
(not
172b91             // best performance with many duplicates in the middle)
21b567             int half = sz(vp)/2;
24742c             first = new Node({vp.begin(), vp.begin() + half});
ae6d3b             second = new Node({vp.begin() + half, vp.end()});
470fcd         }
0265cf     }
6fda19};
6fda19
ce4e50struct KDTree {
eee062     Node* root;
677e4a     KDTree(const vector<P>& vp) : root(new Node({all(vp)})
) {}
677e4a
7daf7f     pair<T, P> search(Node *node, const P& p) {
```

```

23e6bd         if (!node->first) {
23e6bd             // uncomment if we should not find the point
itself:
23e6bd             // if (p == node->pt) return {INF, P()};
df1914             return make_pair((p - node->pt).dist2(), node->pt)
;
19dc67         }
19dc67
Node *f = node->first, *s = node->second;
f3c18d     T bfirst = f->distance(p), bsec = s->distance(p);
c51266     if (bfirst > bsec) swap(bsec, bfirst), swap(f, s);
5cf03e
5cf03e     // search closest side first, other side if needed
fa9faa     auto best = search(f, p);
b7e192     if (bsec < best.first)
18c5d3         best = min(best, search(s, p));
891524     return best;
3771f7 }
3771f7
3771f7 // find nearest point to a point, and its squared
distance
3771f7 // (requires an arbitrary operator< for Point)
5c5074     pair<T, P> nearest(const P& p) {
961132         return search(root, p);
60e74e     }
bac5b0};
```

Line hull intersection

Description: Yoinked from kactl. 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 polyon:

- $(-1, -1)$ 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.

Complexity: $\mathcal{O}(\log n)$.

```
-----7cf45b
d41d8c// #include "Point.h"
d41d8c
53058e#define cmp(i,j) sgn(dir.perp().cross(poly[(i)%n]-poly[(j)%n]))
d4b890#define extr(i) cmp(i + 1, i) >= 0 && cmp(i, i - 1 + n) < 0
8387c5template <class P> int extrVertex(vector<P>& poly, P dir) {
6c658c     int n = sz(poly), lo = 0, hi = n;
b9df6a     if (extr(0)) return 0;
b3e410     while (lo + 1 < hi) {
407848         int m = (lo + hi) / 2;
1b27ac         if (extr(m)) return m;
604289         int ls = cmp(lo + 1, lo), ms = cmp(m + 1, m);
c739cd         (ls < ms || (ls == ms && ls == cmp(lo, m)) ? hi : lo
) = m;
efdc09     }
743d4a     return lo;
ba41ca}
ba41ca
911b88#define cmpL(i) sgn(a.cross(poly[i], b))
26a22btemplate <class P>
d01376array<int, 2> lineHull(P a, P b, vector<P>&& poly) {
d0d8a9     int endA = extrVertex(poly, (a - b).perp());
bc546b     int endB = extrVertex(poly, (b - a).perp());
f177a0     if (cmpL(endA) < 0 || cmpL(endB) > 0)
07bb09         return {-1, -1};
```

```

a8a9c2     array<int, 2> res;
aa612e     rep(i,0,2) {
090d37         int lo = endB, hi = endA, n = sz(poly);
0ef38e         while ((lo + 1) % n != hi) {
71097d             int m = ((lo + hi + (lo < hi ? 0 : n)) / 2) % n;
d0c0d9             (cmpL(m) == cmpL(endB) ? lo : hi) = m;
72e441         }
c0e123         res[i] = (lo + !cmpL(hi)) % n;
541f6a         swap(endA, endB);
d56a85     }
d847be     if (res[0] == res[1]) return {res[0], -1};
e14e7a     if (!cmpL(res[0]) && !cmpL(res[1]))
5b4ca0         switch ((res[0] - res[1] + sz(poly) + 1) % sz(poly))
{
         case 0: return {res[0], res[0]};
         case 2: return {res[1], res[1]};
     }
     return res;
7cf45b}
```

Line line intersection

Description: Yoinked from kactl. If a unique intersection point of the lines going through $s1,e1$ and $s2,e2$ exists $\{1, \text{point}\}$ 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|| l_i 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;

```
-----a01f81
d41d8c// #include "Point.h"
d41d8c
7dc51etemplate<class P>
ebe700pair<int, P> lineInter(P s1, P e1, P s2, P e2) {
662a43     auto d = (e1 - s1).cross(e2 - s2);
a6ba9e     if (d == 0) // if parallel
47e53e         return {-(s1.cross(e1, s2) == 0), P(0, 0)};
dfc20b     auto p = s2.cross(e1, e2), q = s2.cross(e2, s1);
c4c8fb     return {1, (s1 * p + e1 * q) / d};
a01f81}
```

Line projection and reflection

Description: Yoinked from kactl. Projects point p onto line ab. Set refl=true to get reflection of point p across line ab instead. The wrong point will be returned if P is an integer point and the desired point doesn't have integer coordinates. Products of three coordinates are used in intermediate steps so watch out for overflow.

```
-----b5562d
d41d8c// #include "Point.h"
d41d8c
7dc51etemplate<class P>
31ae53P lineProj(P a, P b, P p, bool refl=false) {
3c6965     P v = b - a;
3d9bc7     return p - v.perp()*(1+refl)*v.cross(p-a)/v.dist2();
b5562d}
```

Linear transformation

Description: Yoinked from kactl. Apply the linear transformation (translation, rotation and scaling) which takes line p0-p1 to line q0-q1 to point r.

```
-----03a306
d41d8c// #include "Point.h"
d41d8c
62699ectypedef Point<double> P;
a0133aP linearTransformation(const P& p0, const P& p1,
f9bd62     const P& q0, const P& q1, const P& r) {
```



```
16967b P dp = p1-p0, dq = q1-q0, num(dp.cross(dq), dp.dot(dq)
);
d52dff return q0 + P((r-p0).cross(num), (r-p0).dot(num))/dp.
dist2();
03a306}
```

Manhattan MST

Description: Yoinked from kactl. Given N points, returns up to $4N$ edges, which are guaranteed to contain a minimum spanning tree for the graph with edge weights $w(p,q) = |p.x - q.x| + |p.y - q.y|$. Edges are in the form $(distance, src, dst)$. Use a standard MST algorithm on the result to find the final MST.

Complexity: $\mathcal{O}(n \log n)$.

```
d41d8c// #include "Point.h"
d41d8c
b5e58ctypedef Point<int> P;
10762cvector<array<int, 3>> manhattanMST(vector<P> ps) {
82bb37 vi id(sz(ps));
129d92 iota(all(id), 0);
bded47 vector<array<int, 3>> edges;
463af8 rep(k,0,4) {
55be09 sort(all(id), [&](int i, int j) {
return (ps[i]-ps[j]).x < (ps[j]-ps[i]).y});
f00400 map<int, int> sweep;
0a2d30 for (int i : id) {
6ada5f for (auto it = sweep.lower_bound(-ps[i].y);
2327aa it != sweep.end(); sweep.erase(it++)) {
7348ca int j = it->second;
931774 P d = ps[i] - ps[j];
5297c6 if (d.y > d.x) break;
874f9c edges.push_back({d.y + d.x, i, j});
547f14 }
28e949 sweep[-ps[i].y] = i;
5f0d0f }
9ea743 for (P& p : ps) if (k & 1) p.x = -p.x; else swap(p.x
9c2fdc , p.y);
}
666542 return edges;
af3f66
dfe659}
```

Minimum enclosing circle

Description: Yoinked from kactl. Computes the minimum circle that encloses a set of points.

Complexity: $\mathcal{O}(n)$.

```
d41d8c// #include "circumcircle.h"
d41d8c
a287afpair<P, double> mec(vector<P> ps) {
31fcb8 shuffle(all(ps), mt19937(time(0)));
76de0f P o = ps[0];
56a5f0 double r = 0, EPS = 1 + 1e-8;
b5031b rep(i,0,sz(ps)) if ((o - ps[i]).dist() > r * EPS) {
5e7038 o = ps[i], r = 0;
af79ee rep(j,0,i) if ((o - ps[j]).dist() > r * EPS) {
57476d o = (ps[i] + ps[j]) / 2;
da034d r = (o - ps[i]).dist();
14cf15 rep(k,0,j) if ((o - ps[k]).dist() > r * EPS) {
931d7a o = ccCenter(ps[i], ps[j], ps[k]);
b9c1f4 r = (o - ps[i]).dist();
7cd516 }
03da47 }
bfac59 }
5ebee7 return {o, r};
09dd0a}
```

Is on segment

Description: Yoinked from kactl. Returns true iff p lies on the line segment from s to e . Use $(segDist(s,e,p) \leq \epsilon)$ instead when using `Point <double>`.

```
d41d8c// #include "Point.h"
d41d8c
5145abtemplate<class P> bool onSegment(P s, P e, P p) {
b95df6 return p.cross(s, e) == 0 && (s - p).dot(e - p) <= 0;
c597e8}
```

2D Point

Description: Yoinked from kactl. Class to handle points in the plane. T can be e.g. double or long long. (Avoid int.).

```
48b588template<class T> int sgn(T x) { return (x > 0) - (x <
0); }
fcf845template<class T>
74299cstruct Point {
f773fb typedef Point P;
fa79fb T x, y;
551774 explicit Point(T x=0, T y=0) : x(x), y(y) {}
1a0130 bool operator<(P p) const { return tie(x,y) < tie(p.x,
p.y); }
3a27ca bool operator==(P p) const { return tie(x,y)==tie(p.x,
p.y); }
1dc17e P operator+(P p) const { return P(x+p.x, y+p.y); }
189cbc P operator-(P p) const { return P(x-p.x, y-p.y); }
268af3 P operator*(T d) const { return P(x*d, y*d); }
8cb755 P operator/(T d) const { return P(x/d, y/d); }
716d84 T dot(P p) const { return x*p.x + y*p.y; }
76cf42 T cross(P p) const { return x*p.y - y*p.x; }
520e7b T cross(P a, P b) const { return (a-*this).cross(b-*
this); }
e7b843 T dist2() const { return x*x + y*y; }
039a77 double dist() const { return sqrt((double)dist2()); }
039a77 // angle to x-axis in interval [-pi, pi]
c670a2 double angle() const { return atan2(y, x); }
b02e9c P unit() const { return *this/dist(); } // makes dist
()=1
e05505 P perp() const { return P(-y, x); } // rotates +90
degrees
c0e5d2 P normal() const { return perp().unit(); }
c0e5d2 // returns point rotated 'a' radians ccw around the
origin
91d8d5 P rotate(double a) const {
4e5d5d return P(x*cos(a)-y*sin(a),x*sin(a)+y*cos(a)); }
70601a friend ostream& operator<<(ostream& os, P p) {
0e491f return os << "(" << p.x << ", " << p.y << ")"; }
47ec0a};
```

3D Point

Description: Yoinked from kactl. Class to handle points in 3D space. T can be e.g. double or long long. (Avoid int.).

```
f10732template<class T> struct Point3D {
144fa4 typedef Point3D P;
cac5b9 typedef const P& R;
521bb2 T x, y, z;
c7b740 explicit Point3D(T x=0, T y=0, T z=0) : x(x), y(y), z(
z) {}
9e2218 bool operator<(R p) const {
af5a46 return tie(x, y, z) < tie(p.x, p.y, p.z); }
16e4b3 bool operator==(R p) const {
fa5b42 return tie(x, y, z) == tie(p.x, p.y, p.z); }
141e02 P operator+(R p) const { return P(x+p.x, y+p.y, z+p.z)
; }
825225 P operator-(R p) const { return P(x-p.x, y-p.y, z-p.z)
; }
```

```
1ee29d P operator*(T d) const { return P(x*d, y*d, z*d); }
660687 P operator/(T d) const { return P(x/d, y/d, z/d); }
d7cc17 T dot(R p) const { return x*p.x + y*p.y + z*p.z; }
a9fb7d P cross(R p) const {
b90dcd return P(y*p.z - z*p.y, z*p.x - x*p.z, x*p.y - y*p.x
);
f914db }
574fd0 T dist2() const { return x*x + y*y + z*z; }
f12431 double dist() const { return sqrt((double)dist2()); }
f12431 //Azimuthal angle (longitude) to x-axis in interval [-
pi, pi]
c5f1d1 double phi() const { return atan2(y, x); }
c5f1d1 //Zenith angle (latitude) to the z-axis in interval
[0, pi]
c1e43f double theta() const { return atan2(sqrt(x*x+y*y),z);
}
339ecd P unit() const { return *this/(T)dist(); } //makes
dist()=1
339ecd //returns unit vector normal to *this and p
89ad86 P normal(P p) const { return cross(p).unit(); }
89ad86 //returns point rotated 'angle' radians ccw around
axis
cfb921 P rotate(double angle, P axis) const {
6e0acf double s = sin(angle), c = cos(angle); P u = axis.
unit();
8303ee return u*dot(u)*(1-c) + (*this)*c - cross(u)*s;
6cb0dd }
8058ae};
```

Is point in convex polygon

Description: Yoinked from kactl. 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.

Complexity: $\mathcal{O}(\log n)$.

```
d41d8c// #include "Point.h"
d41d8c// #include "Side_of.h"
d41d8c// #include "On_segment.h"
d41d8c
2c0584typedef Point<ll> P;
2c0584
912e4abool inHull(const vector<P>& l, P p, bool strict = true)
{
3f3fc6 int a = 1, b = sz(l) - 1, r = !strict;
7a3fc8 if (sz(l) < 3) return r && onSegment(l[0], l.back(), p
);
b8cb94 if (sideOf(l[0], l[a], l[b]) > 0) swap(a, b);
3c3a3b if (sideOf(l[0], l[a], p) >= r || sideOf(l[0], l[b], p
)<= -r)
bc80dd return false;
709831 while (abs(a - b) > 1) {
e79ab6 int c = (a + b) / 2;
2a9b80 (sideOf(l[0], l[c], p) > 0 ? b : a) = c;
e4f356 }
0b5229 return sgn(l[a].cross(l[b], p)) < r;
71446b}
```

Polygon area

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

```
d41d8c// #include "Point.h"
d41d8c
4fce64template<class T>
df7c3fT polygonArea2(vector<Point<T>>& v) {
ab8862 T a = v.back().cross(v[0]);
0711d6 rep(i,0,sz(v)-1) a += v[i].cross(v[i+1]);
b195d0 return a;
```

| | |
|-----------------------------------------------------------------------------------|------------------------------------------------------|
| f12300} | |
| | |
| Polygon center of mass | |
| Description: Yoinked from kactl. Returns the center of mass for a polygon. | |
| Complexity: $\mathcal{O}(n)$. | |
| ----- | 9706dc |
| d41d8c// #include "Point.h" | |
| d41d8c | |
| 6269ec | typedef Point<double> P; |
| fa2dc3P | polygonCenter(const vector<P>& v) { |
| a6f845 | P res(0, 0); double A = 0; |
| 1dc006 | for (int i = 0, j = sz(v) - 1; i < sz(v); j = i++) { |
| 082251 | res = res + (v[i] + v[j]) * v[j].cross(v[i]); |
| c6e9e9 | A += v[j].cross(v[i]); |
| 01751d | } |
| 9d5722 | return res / A / 3; |
| 9706dc} | |
| | |

Polygon cut

Description: Yoinked from kactl. 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));

| | |
|-----------------------------------------|------------------------------------------------------|
| f2b7d4 | |
| ----- | |
| d41d8c// #include "Point.h" | |
| d41d8c// #include "Line_intersection.h" | |
| d41d8c | |
| 6269ec | typedef Point<double> P; |
| b4b253vector<P> | polygonCut(const vector<P>& poly, P s, P e) { |
| b83885 | vector<P> res; |
| f6354c | rep(i,0,sz(poly)) { |
| 3664ba | P cur = poly[i], prev = i ? poly[i-1] : poly.back(); |
| 41eaab | bool side = s.cross(e, cur) < 0; |
| f87882 | if (side != (s.cross(e, prev) < 0)) |
| f7bea5 | res.push_back(lineInter(s, e, cur, prev).second); |
| f5439d | if (side) |
| cf4e26 | res.push_back(cur); |
| 567ae4 | } |
| f25262c | return res; |
| f2b7d4} | |
| | |

Polygon union

Description: Yoinked from kactl. 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.)

Complexity: $\mathcal{O}(n^2)$ where n is the total number of points.

| | |
|-------------------------------|-----------------------------------------------------|
| ----- | 3931c6 |
| d41d8c// #include "Point.h" | |
| d41d8c// #include "Side_of.h" | |
| d41d8c | |
| 6269ec | typedef Point<double> P; |
| 940b75double | rat(P a, P b) { return sgn(b.x) ? a.x/b.x : a.y/b |
| | .y; } |
| 51eb9cdouble | polyUnion(vector<vector<P>>& poly) { |
| 9680ea | double ret = 0; |
| 49c6ab | rep(i,0,sz(poly)) rep(v,0,sz(poly[i])) { |
| 1ea114 | P A = poly[i][v], B = poly[i][(v + 1) % sz(poly[i]) |
| |]; |
| e9da64 | vector<pair<double, int>> segs = {{0, 0}, {1, 0}}; |
| aea249 | rep(j,0,sz(poly)) if (i != j) { |
| 03624d | rep(u,0,sz(poly[j])) { |
| 0826f1 | P C = poly[j][u], D = poly[j][(u + 1) % sz(poly[|
| | j])]; |
| c62a46 | int sc = sideOf(A, B, C), sd = sideOf(A, B, D); |
| ac826b | if (sc != sd) { |
| | |

| | |
|---------|------------------------------------------------------|
| a48d6d | double sa = C.cross(D, A), sb = C.cross(D, B); |
| aeaa76 | if (min(sc, sd) < 0) |
| 13f2a7 | segs.emplace_back(sa / (sa - sb), sgn(sc - |
| | sd)); |
| ce5e1a | } else if (!sc && !sd && j<i && sgn((B-A).dot(D- |
| | C))>0){ |
| a4636e | segs.emplace_back(rat(C - A, B - A), 1); |
| d44814 | segs.emplace_back(rat(D - A, B - A), -1); |
| 67520d | } |
| c4b419 | } |
| a1900f | } |
| 97ae86 | sort(all(segs)); |
| 4e8cac | for (auto& s : segs) s.first = min(max(s.first, 0.0) |
| | , 1.0); |
| 00b8ae | double sum = 0; |
| 40a9a7 | int cnt = segs[0].second; |
| 317ef1 | rep(j,1,sz(segs)) { |
| 84ade9 | if (!cnt) sum += segs[j].first - segs[j - 1].first |
| | ; |
| 625398 | cnt += segs[j].second; |
| d3398f | } |
| 0e34c6 | ret += A.cross(B) * sum; |
| 6f2b4e | } |
| 52ed80 | return ret / 2; |
| 3931c6} | |
| | |

Polyhedron volume

Description: Yoinked from kactl. Magic formula for the volume of a polyhedron. Faces should point outwards.

| | |
|----------------------------------|-------------------------------------------------------|
| ----- | 3058c3 |
| f9cf71template<class V, class L> | |
| 8b5f1fdouble | signedPolyVolume(const V& p, const L& trilst) { |
| 75c331 | double v = 0; |
| 828881 | for (auto i : trilst) v += p[i.a].cross(p[i.b]).dot(p |
| | [i.c]); |
| 27c3d1 | return v / 6; |
| 3058c3} | |
| | |

Points line-segments distance

Description: Yoinked from kactl. Returns the shortest distance between point p and the line segment from point s to e.

Usage: Point <double> a, b(2,2), p(1,1);

bool onSegment = segDist(a,b,p) < 1e-10;

| | |
|-----------------------------|--------------------------------------------------------|
| ----- | 5c88f4 |
| d41d8c// #include "Point.h" | |
| d41d8c | |
| 6269ec | typedef Point<double> P; |
| 789af4double | segDist(P& s, P& e, P& p) { |
| 3139df | if (s==e) return (p-s).dist(); |
| 2506d7 | auto d = (e-s).dist2(), t = min(d,max(.0,(p-s).dot(e-s |
| |)); |
| b95d89 | return ((p-s)*d-(e-s)*t).dist()/d; |
| 5c88f4} | |
| | |

Line segment line segment intersection

Description: Yoinked from kactl. 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 PointIjL 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;

| | |
|-----------------------------|--------|
| ----- | 9d57f2 |
| d41d8c// #include "Point.h" | |

| | |
|---------------------------------|--------------------------------------------------------|
| d41d8c// #include "OnSegment.h" | |
| d41d8c | |
| dae11dtemplate<class P> | vector<P> segInter(P a, P b, P c, P d) |
| | { |
| f4c95c | auto oa = c.cross(d, a), ob = c.cross(d, b), |
| 5041fa | oc = a.cross(b, c), od = a.cross(b, d); |
| 5041fa | // Checks if intersection is single non-endpoint point |
| | . |
| dec360 | if (sgn(oa) * sgn(ob) < 0 && sgn(oc) * sgn(od) < 0) |
| ab16eb | return {(a * ob - b * oa) / (ob - oa)}; |
| 43185b | set<P> s; |
| d73b7a | if (onSegment(c, d, a)) s.insert(a); |
| 9f9c48 | if (onSegment(c, d, b)) s.insert(b); |
| 64d2c1 | if (onSegment(a, b, c)) s.insert(c); |
| 1dcb4f | if (onSegment(a, b, d)) s.insert(d); |
| c505dc | return {all(s)}; |
| 9d57f2} | |
| | |

Side of

Description: Yoinked from kactl. 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 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;

| | |
|-----------------------------|------------------------------------------------------|
| ----- | 3af81c |
| d41d8c// #include "Point.h" | |
| d41d8c | |
| 7dc51etemplate<class P> | |
| fa9d99int | sideOf(P s, P e, P p) { return sgn(s.cross(e, p)); } |
| fad9c9 | |
| bb2891template<class P> | |
| 059ae5int | sideOf(const P& s, const P& e, const P& p, double |
| | eps) { |
| 37dc17 | auto a = (e-s).cross(p-s); |
| ea3543 | double l = (e-s).dist()*eps; |
| 765665 | return (a > 1) - (a < -1); |
| 3af81c} | |
| | |

Spherical distance

Description: Yoinked from kactl. Returns the shortest distance on the sphere with radius radius between the points with azimuthal angles (longitude) f1 (ϕ_1) and f2 (ϕ_2) from x axis and zenith angles (latitude) t1 (θ_1) and t2 (θ_2) from z axis (0 = north pole). All angles measured in radians. The algorithm starts by converting the spherical coordinates to cartesian coordinates so if that is what you have you can use only the two last rows. $dx \cdot radius$ is then the difference between the two points in the x direction and d*radius is the total distance between the points.

| | |
|--------------|------------------------------------------------|
| 611f07 | |
| ----- | |
| c5faf9double | sphericalDistance(double f1, double t1, |
| 86b44b | double f2, double t2, double radius) { |
| 2b5463 | double dx = sin(t2)*cos(f2) - sin(t1)*cos(f1); |
| aa0db3 | double dy = sin(t2)*sin(f2) - sin(t1)*sin(f1); |
| 6da400 | double dz = cos(t2) - cos(t1); |
| 819384 | double d = sqrt(dx*dx + dy*dy + dz*dz); |
| 5b1067 | return radius*2*asin(d/2); |
| 611f07} | |
| | |

Line distance

Description: Yoinked from kactl. 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.
-----f6bf6b
d41d8c// #include "Point.h"
d41d8c
7dc51atemplate<class P>
869862double lineDist(const P& a, const P& b, const P& p) {
0aca9c     return (double)(b-a).cross(p-a)/(b-a).dist();
f6bf6b}
```

Graphs

Articulation points finding

Description: Yoinked from CP-algorithms. Standard articulation points finding algorithm.
Complexity: $\mathcal{O}(V + E)$.

```
-----23f413
1a88faint n; // number of nodes
2e71a7vector<vector<int>>> adj; // adjacency list of graph
2e71a7
553d0avector<bool> visited;
00663bvector<int> tin, low;
901990int timer;
901990
d987ecvoid dfs(int v, int p = -1) {
1a0c45     visited[v] = true;
e286e0     tin[v] = low[v] = timer++;
9157ae     int children=0;
b2fdrc     for (int to : adj[v]) {
ac6371         if (to == p) continue;
efefc6         if (visited[to]) {
bb0d96             low[v] = min(low[v], tin[to]);
2cd5f         } else {
51b8d7             dfs(to, v);
bce7e             low[v] = min(low[v], low[to]);
bcdf3d             if (low[to] >= tin[v] && p!=-1)
fc8020                 IS_CUTPOINT(v);
9909e4             ++children;
beb204         }
59a553     }
37508f     if(p == -1 && children > 1)
939ca9         IS_CUTPOINT(v);
a16b47}
a16b47
4d5f0evoid find_cutpoints() {
5828db     timer = 0;
7c3d3d     visited.assign(n, false);
7e557a     tin.assign(n, -1);
3c48fc     low.assign(n, -1);
3c9834     for (int i = 0; i < n; ++i) {
44cd93         if (!visited[i])
006f9c             dfs(i);
aa8a61     }
23f413}
```

Bellman-Ford

Description: Yoinked from kactl. Calculates shortest paths from s in a graph that might have negative edge weights. Unreachable nodes get $\text{dist} = \text{inf}$; nodes reachable through negative-weight cycles get $\text{dist} = -\text{inf}$. Assumes $V^2 \max |w_i| < \sim 2^{63}$.
Usage: `bellmanFord(nodes, edges, s)`.
Complexity: $\mathcal{O}(VE)$.

```
-----830a8f
f5a3e7const ll inf = LLONG_MAX;
5567e9struct Ed { int a, b, w, s() { return a < b ? a : -a;
    }};
2045f7struct Node { ll dist = inf; int prev = -1; };
2045f7
```

```
019c78void bellmanFord(vector<Node>& nodes, vector<Ed>& eds,
    int s) {
ec0b61     nodes[s].dist = 0;
15a23e     sort(all(eds), [](Ed a, Ed b) { return a.s() < b.s();
    });
96d3f0     int lim = sz(nodes) / 2 + 2; // /3+100 with shuffled
    vertices
9004e9     rep(i,0,lim) for (Ed ed : eds) {
c5c796         Node cur = nodes[ed.a], &dest = nodes[ed.b];
ed7594         if (abs(cur.dist) == inf) continue;
4a6344         ll d = cur.dist + ed.w;
167727         if (d < dest.dist) {
729010             dest.prev = ed.a;
68e296             dest.dist = (i < lim-1 ? d : -inf);
2e4a08         }
cab225     }
824bac     rep(i,0,lim) for (Ed e : eds) {
5e8ff4         if (nodes[e.a].dist == -inf)
6495a8             nodes[e.b].dist = -inf;
8d0b1c     }
830a8f}
```

Biconnected components

Description: Yoinked from kactl. 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(b, eid); ed[b].emplace(a, eid++); } bicomps([&] (const vi& edgelist) { ... });`
Complexity: $\mathcal{O}(E + V)$.

```
-----2965e5
16a1edvi num, st;
5c7bd5vector<vector<pii>> ed;
5c17a1int Time;
bf2641template<class F>
3e8edaint dfs(int at, int par, F& f) {
d1b332     int me = num[at] = ++Time, e, y, top = me;
95a358     for (auto pa : ed[at]) if (pa.second != par) {
e55cf3         tie(y, e) = pa;
4e5b73         if (num[y]) {
fe0f3e             top = min(top, num[y]);
145ca4             if (num[y] < me)
01b6d5                 st.push_back(e);
51d5dc         } else {
8aee96             int si = sz(st);
e478b0             int up = dfs(y, e, f);
4c0c04             top = min(top, up);
fb91dd             if (up == me) {
0aa7e5                 st.push_back(e);
10c0ea                 f(vi(st.begin() + si, st.end()));
7a2eb7                 st.resize(si);
4c59fd             }
e01a87             else if (up < me) st.push_back(e);
47e7b7             else { /* e is a bridge */ }
7a2ccf         }
55adf3     }
58e3ce     return top;
0b5c9f}
0b5c9f
2617cctemplate<class F> void bicomps(F f) {
b5c03f     num.assign(sz(ed), 0);
14c211     rep(i,0,sz(ed)) if (!num[i]) dfs(i, -1, f);
2965e5}
```

Binary lifting with LCA

Description: Yoinked from kactl. Finds power of two jumps in a tree - and standard LCA. Assumes the root node points to itself!

Usage: `vector<vi> jmps = treeJump(parents); int l = lca(jmps, depth, a, b);`
Complexity: $\mathcal{O}(N \log N)$ construction. $\mathcal{O}(\log N)$ per query.

```
-----bfce85
750796vector<vi> treeJump(vi& P){
d7f747     int on = 1, d = 1;
4e1485     while(on < sz(P)) on *= 2, d++;
40155b     vector<vi> jmp(d, P);
bcb753     rep(i,1,d) rep(j,0,sz(P))
35de77         jmp[i][j] = jmp[i-1][jmp[i-1][j]];
9cd4c2     return jmp;
6d3434}
6d3434
d0c552int jmp(vector<vi>& tbl, int nod, int steps){
68ef34     rep(i,0,sz(tbl))
fa7843         if(steps&(1<<i)) nod = tbl[i][nod];
5f4dea     return nod;
7ce14c}
7ce14c
48e3efint lca(vector<vi>& tbl, vi& depth, int a, int b) {
dae62d     if (depth[a] < depth[b]) swap(a, b);
afb472     a = jmp(tbl, a, depth[a] - depth[b]);
74edff     if (a == b) return a;
ea1a60     for (int i = sz(tbl); i--;) {
67fff4         int c = tbl[i][a], d = tbl[i][b];
6533fb         if (c != d) a = c, b = d;
863967     }
b796a3     return tbl[0][a];
bfce85}
```

Bridge finding

Description: Yoinked from CP-algorithms. Standard bridge finding algorithm.
Complexity: $\mathcal{O}(V + E)$.

```
-----a44485
1a88fdint n; // number of nodes
2e71a7vector<vector<int>>> adj; // adjacency list of graph
2e71a7
553d0avector<bool> visited;
00663bvector<int> tin, low;
901990int timer;
901990
d987ecvoid dfs(int v, int p = -1) {
1a0c45     visited[v] = true;
e286e0     tin[v] = low[v] = timer++;
e82afa     for (int to : adj[v]) {
4b3a29         if (to == p) continue;
57a119         if (visited[to]) {
440a56             low[v] = min(low[v], tin[to]);
f87d63         } else {
c6c172             dfs(to, v);
00a71d             low[v] = min(low[v], low[to]);
f54aa9             if (low[to] > tin[v])
08b207                 IS_BRIDGE(v, to);
030e02         }
276ef5     }
8768b3}
8768b3
190845void find_bridges() {
12b6f3     timer = 0;
bbc736     visited.assign(n, false);
87eddd     tin.assign(n, -1);
864765     low.assign(n, -1);
bc4c5c     for (int i = 0; i < n; ++i) {
00e55b         if (!visited[i])
307b77             dfs(i);
6a780d     }
a44485}
```

DFS Bipartite Matching

Description: Yoinked from kactl. 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); dfsMatching(g, btoa);
Complexity: $\mathcal{O}(VE)$.

```
-----522b98
a47cc3bool find(int j, vector<vi>& g, vi& btoa, vi& vis) {
98d83d if (btoa[j] == -1) return 1;
6aa9ef vis[j] = 1; int di = btoa[j];
d093f9 for (int e : g[di])
400b9b if (!vis[e] && find(e, g, btoa, vis)) {
b1c950 btoa[e] = di;
107fe8 return 1;
cc0de1 }
bf43f0 return 0;
d13a81}
1578f8int dfsMatching(vector<vi>& g, vi& btoa) {
ae152c vi vis;
49e964 rep(i,0,sz(g)) {
62eadd vis.assign(sz(btoa), 0);
0eda2c for (int j : g[i])
c468b2 if (find(j, g, btoa, vis)) {
407765 btoa[j] = i;
5b1f88 break;
5609e1 }
61061f }
c95a04 return sz(btoa) - (int)count(all(btoa), -1);
522b98}
```

Dinic's Algorithm

Description: Yoinked from kactl. Finds the maximum flow from s to t in a directed graph. To obtain the actual flow values, look at all edges with capacity > 0 (zero capacity edges are residual edges).
Usage: Dinic dinic(n); dinic.addEdge(a, b, c); dinic.maxFlow(s, t);
Complexity: $\mathcal{O}(VE \log U)$ where $U = \max | \text{capacity} |$. $\mathcal{O}(\min(\sqrt{E}, V^{2/3})E)$ if $U = 1$; so $\mathcal{O}(\sqrt{VE})$ for bipartite matching.

```
-----d7f0f1
14df72struct Dinic {
9230ca struct Edge {
ca825e int to, rev;
eeace ll c, oc;
299dbe ll flow() { return max(oc - c, 0LL); } // if you
need flows
};
9d5927 vi lvl, ptr, q;
0aa82d vector<vector<Edge>> adj;
31ed82 Dinic(int n) : lvl(n), ptr(n), q(n), adj(n) {}
fdd5b9 void addEdge(int a, int b, ll c, ll rcap = 0) {
4f21de adj[a].push_back({b, sz(adj[b]), c, c});
3c87cb adj[b].push_back({a, sz(adj[a]) - 1, rcap, rcap});
95d74a }
a45d7e ll dfs(int v, int t, ll f) {
0e705d if (v == t || !f) return f;
836e00 for (int& i = ptr[v]; i < sz(adj[v]); i++) {
2410c4 Edge& e = adj[v][i];
d7080f if (lvl[e.to] == lvl[v] + 1)
591b8b if (ll p = dfs(e.to, t, min(f, e.c))) {
fad0d4 e.c -= p, adj[e.to][e.rev].c += p;
aedea1 return p;
02fe28 }
d3bb27 }
f4fbae }
7da8aa return 0;
72048c }
e7b939 ll calc(int s, int t) {
2195f9 ll flow = 0; q[0] = s;
```

```
b15633 rep(L,0,31) do { // 'int L=30' maybe faster for
random data
d70f60 lvl = ptr = vi(sz(q));
5d9371 int qi = 0, qe = lvl[s] = 1;
5702d8 while (qi < qe && !lvl[t]) {
a7da4e int v = q[qi++];
8c4b36 for (Edge e : adj[v])
3c4dab if (!lvl[e.to] && e.c >> (30 - L))
0d5640 q[qe++] = e.to, lvl[e.to] = lvl[v] + 1;
16dd6b }
12fc53 while (ll p = dfs(s, t, LLONG_MAX)) flow += p;
f27334 } while (lvl[t]);
14d62b return flow;
2b90e4 }
761cc4 bool leftOfMinCut(int a) { return lvl[a] != 0; }
a7f0f1};
```

MST in directed graphs

Description: Yoinked from kactl. Finds a minimum spanning tree/arborescence of a directed graph, given a root node. If no MST exists, returns -1.
Usage: pair<ll, vi> res = DMST(n, edges, root);
Complexity: $\mathcal{O}(E \log V)$.

```
-----39e620
d41d8c// #include "../Data_structures/dsu_rollback.h"
d41d8c
030131struct Edge { int a, b; ll w; };
7519f2struct Node { /// lazy skew heap node
45a840 Edge key;
348382 Node *l, *r;
59f245 ll delta;
void prop() {
958c51 key.w += delta;
c4174f if (l) l->delta += delta;
9353bd if (r) r->delta += delta;
69a899 delta = 0;
cfc93b }
31f792 }
61e0cf Edge top() { prop(); return key; }
67708e};
d59b55Node *merge(Node *a, Node *b) {
6b68b8 if (!a || !b) return a ?: b;
839210 a->prop(), b->prop();
7c5d9a if (a->key.w > b->key.w) swap(a, b);
c76878 swap(a->l, (a->r = merge(b, a->r)));
046c62 return a;
5e360c}
821d19void pop(Node*& a) { a->prop(); a = merge(a->l, a->r); }
821d19
ef4c12pair<ll, vi> dmst(int n, int r, vector<Edge>& g) {
4a59c3 RollbackUF uf(n);
a7352a vector<Node*> heap(n);
a4c794 for (Edge e : g) heap[e.b] = merge(heap[e.b], new Node {e});
5ec2e1 ll res = 0;
9ed102 vi seen(n, -1), path(n), par(n);
92e6e3 seen[r] = r;
c7b0b9 vector<Edge> Q(n), in(n, {-1,-1}), comp;
fc7b25 deque<tuple<int, int, vector<Edge>>> cys;
360529 rep(s,0,n) {
96b18b int u = s, qi = 0, w;
fae505 while (seen[u] < 0) {
2158f1 if (!heap[u]) return {-1,{};};
bcb3d2 Edge e = heap[u]->top();
cc1e56 heap[u]->delta -= e.w, pop(heap[u]);
d9d5a2 Q[qi] = e, path[qi++] = u, seen[u] = s;
fcb967 res += e.w, u = uf.find(e.a);
7ed0a if (seen[u] == s) { /// found cycle, contract
2e137f Node* cyc = 0;
5167d4 int end = qi, time = uf.time();
618ecf do cyc = merge(cyc, heap[w = path[--qi]]);
7cef71 while (uf.join(u, w));
```

```
3eb5cd u = uf.find(u), heap[u] = cys, seen[u] = -1;
3a9488 cys.push_front({u, time, {&Q[qi], &Q[end]}});
ea74cd }
db364e }
93005f rep(i,0,qi) in[uf.find(Q[i].b)] = Q[i];
f2bc30 }
f2bc30
186c68 for (auto& [u,t,comp] : cys) { // restore sol (optional)
55dced uf.rollback(t);
6dda7b Edge inEdge = in[u];
a32e6d for (auto& e : comp) in[uf.find(e.b)] = e;
b09240 in[uf.find(inEdge.b)] = inEdge;
c54d7 }
4f8a9a rep(i,0,n) par[i] = in[i].a;
d28015 return {res, par};
39e620}
```

(D + 1)-edge coloring

Description: Yoinked from kactl. 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.)
Usage: vi res = edgeColoring(N, eds);
Complexity: $\mathcal{O}(NM)$.

```
-----e210e2
f41922vi edgeColoring(int N, vector<pii> eds) {
aa3ad0 vi cc(N + 1), ret(sz(eds)), fan(N), free(N), loc;
04f5f4 for (pii e : eds) ++cc[e.first], ++cc[e.second];
a8572e int u, v, ncols = *max_element(all(cc)) + 1;
d26648 vector<vi> adj(N, vi(ncols, -1));
fc7443 for (pii e : eds) {
e8084f tie(u, v) = e;
1235a9 fan[0] = v;
2ddcc8 loc.assign(ncols, 0);
716e30 int at = u, end = u, d, c = free[u], ind = 0, i = 0;
96c76c while (d = free[v], !loc[d] && (v = adj[u][d]) != -1)
loc[d] = ++ind, cc[ind] = d, fan[ind] = v;
9115a5 cc[loc[d]] = c;
5a2c0f for (int cd = d; at != -1; cd ^= c ^ d, at = adj[at][cd])
swap(adj[at][cd], adj[end = at][cd ^ c ^ d]);
2827eb while (adj[fan[i]][d] != -1) {
f3efaf int left = fan[i], right = fan[++i], e = cc[i];
e98916 adj[u][e] = left;
90bb57 adj[left][e] = u;
4e1b6a adj[right][e] = -1;
e7082c free[right] = e;
657a28 }
a781ab adj[u][d] = fan[i];
2eeb98 adj[fan[i]][d] = u;
783efe for (int y : {fan[0], u, end})
2b3648 for (int& z = free[y] = 0; adj[y][z] != -1; z++);
e9f8dc }
967649 rep(i,0,sz(eds))
0c6ff6 for (tie(u, v) = eds[i]; adj[u][ret[i]] != v;) ++ret[i];
ce6fa1 return ret;
e210e2}
```

Edmonds-Karp

Description: Yoinked from kactl. Flow algorithm with guaranteed complexity $\mathcal{O}(VE^2)$. To get edge flow values, compare capacities before and after, and take the positive values only.
Usage: edmondsKarp(graph, source, sink);
Complexity: $\mathcal{O}(EV^2)$.


```
676711template<class T> T edmondsKarp(vector<unordered_map<int
    , T>>& graph, int source, int sink) {
dc891c    assert(source != sink);
16aa01    T flow = 0;
324dc1    vi par(sz(graph)), q = par;
324dc1
6b3baa    for (;;) {
b6886e        fill(all(par), -1);
8ed190        par[source] = 0;
f85f7e        int ptr = 1;
968ffa        q[0] = source;
968ffa
481db7        rep(i,0,ptr) {
4dfc15            int x = q[i];
0b66e7            for (auto e : graph[x]) {
47c24f                if (par[e.first] == -1 && e.second > 0) {
edc6f5                    par[e.first] = x;
7bf6c2                    q[ptr++] = e.first;
3c94b0                    if (e.first == sink) goto out;
013016                }
e083c2            }
b22780        }
b14b2c        return flow;
a8b66f    out:
f9f5c6        T inc = numeric_limits<T>::max();
ff74aa        for (int y = sink; y != source; y = par[y])
59bbb1            inc = min(inc, graph[par[y]][y]);
59bbb1
b7fad4        flow += inc;
874b49        for (int y = sink; y != source; y = par[y]) {
7342d3            int p = par[y];
39f2f7            if ((graph[p][y] -= inc) <= 0) graph[p].erase(y);
63483a            graph[y][p] += inc;
868f7e        }
98a343    }
482fe0}
```

Floyd-Warshall

Description: Yoinked from kactl. Calculates all-pairs shortest path in a directed graph that might have negative edge weights. Input is an distance matrix m , where $m[i][j]$ = inf if i and j are not adjacent. As output, $m[i][j]$ is set to the shortest distance between i and j , inf if no path, or -inf if the path goes through a negative-weight cycle.

Usage: floydWarshall(m);

Complexity: $\mathcal{O}(n^3)$.

```
964414const ll inf = 1LL << 62;
433b02void floydWarshall(vector<vector<ll>>& m) {
b0c2bb    int n = sz(m);
2b4646    rep(i,0,n) m[i][i] = min(m[i][i], 0LL);
2794c2    rep(k,0,n) rep(i,0,n) rep(j,0,n)
7be85c        if (m[i][k] != inf && m[k][j] != inf) {
46581f            auto newDist = max(m[i][k] + m[k][j], -inf);
9a15b1            m[i][j] = min(m[i][j], newDist);
2682ca        }
747b97    rep(k,0,n) if (m[k][k] < 0) rep(i,0,n) rep(j,0,n)
54f5ea        if (m[i][k] != inf && m[k][j] != inf) m[i][j] = -inf
;
531245}
```

General matching

Description: Yoinked from kactl. Matching for general graphs. Finds a maximum subset of edges such that each vertex is incident to at most one edge. Fails with probability $\frac{N}{\text{mod}}$.

Usage: generalMatching(N, ed)

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

```
-----cb1912
d41d8c// #include "../Maths/Matrix_inverse_mod.h"
d41d8c
```

```
d41d8c
376046vector<pii> generalMatching(int N, vector<pii>& ed) {
8a1892    vector<vector<ll>> mat(N, vector<ll>(N)), A;
ae1d83    for (pii pa : ed) {
d77802        int a = pa.first, b = pa.second, r = rand() % mod;
19e55d        mat[a][b] = r, mat[b][a] = (mod - r) % mod;
614800    }
614800
5763d0    int r = matInv(A = mat), M = 2*N - r, fi, fj;
acff617    assert(r % 2 == 0);
acff617
9bc254    if (M != N) do {
480c1c        mat.resize(M, vector<ll>(M));
dfai34        rep(i,0,M) {
1593eb            mat[i].resize(M);
ea98c1            rep(j,N,M) {
d8fdfd                int r = rand() % mod;
60f83e                mat[i][j] = r, mat[j][i] = (mod - r) % mod;
36a855            }
be41a1        }
c9966b    } while (matInv(A = mat) != M);
c9966b
50a07b    vi has(M, 1); vector<pii> ret;
17b324    rep(it,0,M/2) {
348eac        rep(i,0,M) if (has[i])
2b3a54            rep(j,i+1,M) if (A[i][j] && mat[i][j]) {
4934d8                fi = i; fj = j; goto done;
be61bb            } assert(0); done:
1a9b3a            if (fj < N) ret.emplace_back(fi, fj);
fcefbe            has[fi] = has[fj] = 0;
341959            rep(sw,0,2) {
b0634f                ll a = modpow(A[fi][fj], mod-2);
e24316                rep(i,0,M) if (has[i] && A[i][fj]) {
600e4d                    ll b = A[i][fj] * a % mod;
d7826c                    rep(j,0,M) A[i][j] = (A[i][j] - A[fi][j] * b) %
mod;
9debef                }
07f84a                swap(fi,fj);
6c623e            }
f16c12        }
cd02c8        return ret;
cb1912}
```

Global minimum cut

Description: Yoinked from kactl. Finds a global minimum cut in an undirected graph, as represented by an adjacency matrix.

Usage: pair<int, vi> res = globalMinCut(mat);

Complexity: $\mathcal{O}(V^3)$.

```
-----8b0e19
192f1dpair<int, vi> globalMinCut(vector<vi> mat) {
81f955    pair<int, vi> best = {INT_MAX, {}};
a4b19e    int n = sz(mat);
165100    vector<vi> co(n);
f640ab    rep(i,0,n) co[i] = {i};
a62b4e    rep(ph,1,n) {
bfa30c        vi w = mat[0];
6e33f2        size_t s = 0, t = 0;
76cb1b        rep(it,0,n-ph) { //  $\mathcal{O}(V^2)$  ->  $\mathcal{O}(E \log V)$  with prio.
queue
c98135            w[t] = INT_MIN;
2c2cfb            s = t, t = max_element(all(w)) - w.begin();
9d976b            rep(i,0,n) w[i] += mat[t][i];
8c07c9        }
727626        best = min(best, {w[t] - mat[t][t], co[t]});
c26622        co[s].insert(co[s].end(), all(co[t]));
d24f0e        rep(i,0,n) mat[s][i] += mat[t][i];
c7b746        rep(i,0,n) mat[i][s] = mat[s][i];
e2544b        mat[0][t] = INT_MIN;
076888    }
6a68fc    return best;
8b0e19}
```

Heavy-light decomposition

Description: Yoinked from kactl. 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. NOTE: below implementation uses kactl lazy segtree, this detail must be modified!

Usage: HLD <false> hld(adj); hld.query_path(u, v); ...

Complexity: $\mathcal{O}(\log n)$ segtree operations per operation.

```
-----6f34db
d41d8c// #include "...kactl_segtree..."
d41d8c
303396template <bool VALS_EDGES> struct HLD {
931c5a    int N, tim = 0;
878e29    vector<vi> adj;
644a8a    vi par, siz, depth, rt, pos;
Node *tree;
6b55a4    HLD(vector<vi> adj_)
d266b7        : N(sz(adj_)), adj(adj_), par(N, -1), siz(N, 1),
9b9a5f        depth(N),
ef2f12        rt(N),pos(N),tree(new Node(0, N)){ dfsSz(0);
dfsHld(0); }
f1f501    void dfsSz(int v) {
6937fc        if (par[v] != -1) adj[v].erase(find(all(adj[v]), par
[v]));
c2274a        for (int& u : adj[v]) {
20e816            par[u] = v, depth[u] = depth[v] + 1;
f64490            dfsSz(u);
7b9912            siz[v] += siz[u];
ef818f            if (siz[u] > siz[adj[v][0]]) swap(u, adj[v][0]);
b0fa49        }
9ba8db    }
b0240c    void dfsHld(int v) {
925ec3        pos[v] = tim++;
6a30a7        for (int u : adj[v]) {
039f8a            rt[u] = (u == adj[v][0] ? rt[v] : u);
2698ee            dfsHld(u);
39b629        }
39d559    }
9dbc9b    template <class B> void process(int u, int v, B op) {
ddfb6b        for (; rt[u] != rt[v]; v = par[rt[v]]) {
f6dd0a            if (depth[rt[u]] > depth[rt[v]]) swap(u, v);
87a197            op(pos[rt[v]], pos[v] + 1);
fa17fe        }
f837ff        if (depth[u] > depth[v]) swap(u, v);
3bc5a1        op(pos[u] + VALS_EDGES, pos[v] + 1);
0d5603    }
178671    void modifyPath(int u, int v, int val) {
99a5b1        process(u, v, [&](int l, int r) { tree->add(l, r,
val); });
79ce98    }
fb7383    int queryPath(int u, int v) { // Modify depending on
problem
c2ff5e0        int res = -1e9;
0e4f0a        process(u, v, [&](int l, int r) {
26c08a            res = max(res, tree->query(l, r));
29a64c        });
e9dec3        return res;
f00cd2    }
4e9b11    int querySubtree(int v) { // modifySubtree is similar
7db27d        return tree->query(pos[v] + VALS_EDGES, pos[v] + siz
[v]);
8aad63    }
6f34db};
```

Hopcroft-Karp Bipartite Matching

Description: Yoinked from kactl. Fast bipartite matching algorithm. Graph g should be a list of neighbors of the left partition, and $btoa$ should be a vector full of -1 's of the same size as the right partition. Returns the size of the matching. $btoa[i]$ will be the match for vertex i on the right side, or -1 if it's not matched.

Usage: `vi btoa(m, -1); hopcroftKarp(g, btoa);`

Complexity: $\mathcal{O}(\sqrt{VE})$.

```
-----f612e4-----
b0e7c1 bool dfs(int a, int L, vector<vi>& g, vi& btoa, vi& A,
vi& B) {
59b291 if (A[a] != L) return 0;
86baa8 A[a] = -1;
77ef66 for (int b : g[a]) if (B[b] == L + 1) {
d9e76d B[b] = 0;
1a816f if (btoa[b] == -1 || dfs(btoa[b], L + 1, g, btoa, A,
B))
return btoa[b] = a, 1;
}
84f762 }
4cc63e return 0;
9e7938 }
9e7938
9e641c int hopcroftKarp(vector<vi>& g, vi& btoa) {
7f282c int res = 0;
252756 vi A(g.size()), B(btoa.size()), cur, next;
a02d20 for (;;) {
d1f7e80 fill(all(A), 0);
591ffa fill(all(B), 0);
591ffa // Find the starting nodes for BFS (i.e. layer 0).
3bf28f cur.clear();
69a5d0 for (int a : btoa) if (a != -1) A[a] = -1;
0fe82b rep(a, 0, sz(g)) if (A[a] == 0) cur.push_back(a);
0fe82b // Find all layers using bfs.
cefa37 for (int lay = 1; lay++) {
93e008 bool islast = 0;
786fae next.clear();
342697 for (int a : cur) for (int b : g[a]) {
96ecca if (btoa[b] == -1) {
17e7a8 B[b] = lay;
87b4fe islast = 1;
}
else if (btoa[b] != a && !B[b]) {
a0fd80 B[b] = lay;
a3408c next.push_back(btoa[b]);
6e6ba7 }
81e09f }
ebc136 if (islast) break;
3b7f1a if (next.empty()) return res;
f1b696 for (int a : next) A[a] = lay;
fc4842 cur.swap(next);
a29db8 }
e487ce }
e487ce // Use DFS to scan for augmenting paths.
b03a1c rep(a, 0, sz(g))
ae47e7 res += dfs(a, 0, g, btoa, A, B);
f385af }
f612e4 }
```

Link-cut tree

Description: Yoinked from kactl. Represents a forest of unrooted trees. You can add and remove edges (as long as the result is still a forest), and check whether two nodes are in the same tree.

Usage: See comments in code.

Complexity: Amortized $\mathcal{O}(\log n)$ per (any) operation.

```
-----5909e2-----
bf28ea struct Node { // Splay tree. Root's pp contains tree's
parent.
0dc895 Node *p = 0, *pp = 0, *c[2];
038f31 bool flip = 0;
210611 Node() { c[0] = c[1] = 0; fix(); }
```

```
a4e156 void fix() {
5b7890 if (c[0]) c[0]->p = this;
577fff if (c[1]) c[1]->p = this;
577fff // (+ update sum of subtree elements etc. if wanted)
4268f1 }
34cb58 void pushFlip() {
1b908c if (!flip) return;
a0ef26 flip = 0; swap(c[0], c[1]);
da653a if (c[0]) c[0]->flip ^= 1;
168072 if (c[1]) c[1]->flip ^= 1;
d94cfc }
829eb8 int up() { return p ? p->c[1] == this : -1; }
b374bb void rot(int i, int b) {
f8bc45 int h = i ^ b;
042831 Node *x = c[i], *y = b == 2 ? x : x->c[h], *z = b ?
y : x;
679f6a if ((y->p = p)) p->c[up()] = y;
59c9a7 c[i] = z->c[i ^ 1];
9fc417 if (b < 2) {
0ef3d2 x->c[h] = y->c[h ^ 1];
345fac z->c[h ^ 1] = b ? x : this;
}
y->c[i ^ 1] = b ? this : x;
fix(); x->fix(); y->fix();
if (p) p->fix();
swap(pp, y->pp);
}
void splay() { // Splay this up to the root. Always
finishes without flip set.
for (pushFlip(); p; ) {
if (p->p) p->p->pushFlip();
p->pushFlip(); pushFlip();
int c1 = up(), c2 = p->up();
if (c2 == -1) p->rot(c1, 2);
else p->p->rot(c2, c1 != c2);
}
}
Node* first() { // Return the min element of the
subtree rooted at this, splayed to the top.
pushFlip();
return c[0] ? c[0]->first() : (splay(), this);
}
52e7b8 struct LinkCut {
582edb vector<Node> node;
4db5d7 LinkCut(int N) : node(N) {}
ed6206 void link(int u, int v) { // add an edge (u, v)
bc1570 assert(!connected(u, v));
f4f3ff makeRoot(&node[u]);
7de638 node[u].pp = &node[v];
8486b3 }
68dfcd void cut(int u, int v) { // remove an edge (u, v)
b178fe Node *x = &node[u], *top = &node[v];
d040ce makeRoot(top); x->splay();
fca899 assert(top == (x->pp ? x->c[0]));
20e52d if (x->pp) x->pp = 0;
341913 else {
ab704a x->c[0] = top->p = 0;
df6ee9 x->fix();
2353b3 }
6bfe4b }
22b84b bool connected(int u, int v) { // are u, v in the same
tree?
Node* nu = access(&node[u])->first();
return nu == access(&node[v])->first();
}
void makeRoot(Node* u) { // Move u to root of
represented tree.
access(u);
u->splay();
}
ccc76d
76a0ba }
```

```
7d90ba if (u->c[0]) {
a8c58c u->c[0]->p = 0;
d3dad8 u->c[0]->flip ^= 1;
870182 u->c[0]->pp = u;
40f699 u->c[0] = 0;
d40ef6 u->fix();
3199b1 }
4c36b5 }
76328e Node* access(Node* u) { // Move u to root aux tree.
Return the root of the root aux tree.
u->splay();
while (Node* pp = u->pp) {
pp->splay(); u->pp = 0;
if (pp->c[1]) {
pp->c[1]->p = 0; pp->c[1]->pp = pp; }
pp->c[1] = u; pp->fix(); u = pp;
}
return u;
}
e0364b }
5909e2 ;
```

Minimum cost maximum flow (faster)

Description: Yoinked from kactl. Does not support negative cost cycles. call `setpi` before `maxflow` if costs can be negative. To obtain the actual flow, look at positive values only.

Complexity: $\mathcal{O}(FE \log(V))$ where F is max flow. $\mathcal{O}(VE)$ for `setpi`.

```
135b73 -----
d41d8c // #include <bits/extc++.h>
d41d8c
9f43ac const ll INF = numeric_limits<ll>::max() / 4;
9f43ac
49eea0 struct MCMF {
1681cd struct edge {
d4edf5 int from, to, rev;
00467c ll cap, cost, flow;
2b1b2e };
3ecc0d int N;
1d58ff vector<vector<edge>> ed;
90fe37 vi seen;
8389f8 vector<ll> dist, pi;
560ffe vector<edge*> par;
560ffe
d4418d MCMF(int N) : N(N), ed(N), seen(N), dist(N), pi(N),
par(N) {}
d4418d
113bdc void addEdge(int from, int to, ll cap, ll cost) {
884902 if (from == to) return;
a2884d ed[from].push_back(edge{ from, to, sz(ed[to]), cap, cost
, 0 });
eaf8bb ed[to].push_back(edge{ to, from, sz(ed[from])-1, 0, -
cost, 0 });
}
578c8c
578c8c void path(int s) {
e1cfe9 fill(all(seen), 0);
58d504 fill(all(dist), INF);
0bc4f9 dist[s] = 0; ll di;
1a59a2
1a59a2
675e44 __gnu_pbds::priority_queue<pair<ll, int>> q;
0f8f1f vector<decltype(q)::point_iterator> its(N);
40b20b q.push({ 0, s });
40b20b
99e6e8 while (!q.empty()) {
50f450 s = q.top().second; q.pop();
bb19ce seen[s] = 1; di = dist[s] + pi[s];
7b482b for (edge& e : ed[s]) if (!seen[e.to]) {
3e16c7 ll val = di - pi[e.to] + e.cost;
000394 if (e.cap - e.flow > 0 && val < dist[e.to]) {
75bb3e dist[e.to] = val;
be7851 par[e.to] = &e;
8d8cfd if (its[e.to] == q.end())
```

```

        its[e.to] = q.push({ -dist[e.to], e.to });
    else
        q.modify(its[e.to], { -dist[e.to], e.to });
    }
}
rep(i,0,N) pi[i] = min(pi[i] + dist[i], INF);
}
pair<ll, ll> maxflow(int s, int t) {
    ll totflow = 0, totcost = 0;
    while (path(s), seen[t]) {
        ll fl = INF;
        for (edge* x = par[t]; x; x = par[x->from])
            fl = min(fl, x->cap - x->flow);
        totflow += fl;
        for (edge* x = par[t]; x; x = par[x->from]) {
            x->flow += fl;
            ed[x->to][x->rev].flow -= fl;
        }
        rep(i,0,N) for(edge& e : ed[i]) totcost += e.cost * e.flow;
        return {totflow, totcost/2};
    }
}
// If some costs can be negative, call this before
maxflow:
void setpi(int s) { // (otherwise, leave this out)
    fill(all(pi), INF); pi[s] = 0;
    int it = N, ch = 1; ll v;
    while (ch-- && it--)
        rep(i,0,N) if (pi[i] != INF)
            for (edge& e : ed[i]) if (e.cap)
                if ((v = pi[i] + e.cost) < pi[e.to])
                    pi[e.to] = v, ch = 1;
    assert(it >= 0); // negative cost cycle
}
15b773};

```

Maximum clique callbacks

Description: Yoinked from kactl. 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.

Usage: cliques(eds, callback, ...);

Complexity: $\mathcal{O}(3^{n/3})$ - much faster for sparse graphs.

```

-----b0d5b1
753236typedef bitset<128> B;
6454cctemplate<class F>
05d32cvoid cliques(vector<B>& eds, F f, B P = ~B(), B X={}, B
    R={}) {
    d462aa if (!P.any()) { if (!X.any()) f(R); return; }
    abbe26 auto q = (P | X)._Find_first();
    01a6f3 auto cands = P & ~eds[q];
    876203 rep(i,0,sz(eds)) if (cands[i]) {
    074813     R[i] = 1;
    cf4187     cliques(eds, f, P & eds[i], X & eds[i], R);
    c889e0 R[i] = P[i] = 0; X[i] = 1;
    2b8ca5 }
    b0d5b1}

```

Maximum clique

Description: Yoinked from kactl. Finds a maximum clique of a graph given as a symmetric bitset matrix. Can be used to find a maximum independent set by finding a clique of the complement graph.

Complexity: About 1 second for $n = 155$, worst case random graphs ($p = .90$). Runs faster for sparse graphs.

```

54ea03typedef vector<bitset<200>> vb;
913d3dstruct Maxclique {
2b09f0 double limit=0.025, pk=0;
93b51d struct Vertex { int i, d=0; };
b929e8 typedef vector<Vertex> vv;
8ec016 vb e;
071744 vv V;
ccd5a0 vector<vi> C;
b548bf vi qmax, q, S, old;
f625cf void init(vv& r) {
4a81cc     for (auto& v : r) v.d = 0;
993a80     for (auto& v : r) for (auto j : r) v.d += e[v.i][j.i]
    };
06b9b4 sort(all(r), [](auto a, auto b) { return a.d > b.d;
    });
16d40c     int mxD = r[0].d;
964a7f     rep(i,0,sz(r)) r[i].d = min(i, mxD) + 1;
d5dc84 }
e66dc void expand(vv& R, int lev = 1) {
ac13ae     S[lev] += S[lev - 1] - old[lev];
8602ba     old[lev] = S[lev - 1];
67e58a     while (sz(R)) {
09eb24         if (sz(q) + R.back().d <= sz(qmax)) return;
20ce0c         q.push_back(R.back().i);
0b52a4         vv T;
b0e686         for(auto v:R) if (e[R.back().i][v.i]) T.push_back
    ({v.i});
e23129         if (sz(T)) {
c706bf             if (S[lev]++ / ++pk < limit) init(T);
86a266             int j = 0, mxk = 1, mnk = max(sz(qmax) - sz(q) +
    1, 1);
fb8d45             C[1].clear(), C[2].clear();
abd788             for (auto v : T) {
d6bf0a                 int k = 1;
3e1b9e                 auto f = [&](int i) { return e[v.i][i]; };
6fcc14                 while (any_of(all(C[k]), f)) k++;
30a122                 if (k > mxk) mxk = k, C[mxk + 1].clear();
f8575a                 if (k < mnk) T[j++] .i = v.i;
8dee8a                 C[k].push_back(v.i);
5ebe7a             }
df11ee             if (j > 0) T[j - 1].d = 0;
bfc7c             rep(k,mnk,mxk + 1) for (int i : C[k])
b4de6c                 T[j].i = i, T[j++].d = k;
e72ba9             expand(T, lev + 1);
86a1f3             } else if (sz(q) > sz(qmax)) qmax = q;
ad6614             q.pop_back(), R.pop_back();
c01dd9         }
901020     }
12c3d2     vi maxClique() { init(V), expand(V); return qmax; }
6c200c     Maxclique(vb conn) : e(conn), C(sz(e)+1), S(sz(C)),
    old(S) {
64b603         rep(i,0,sz(e)) V.push_back({i});
21f145     }
f7c0bc};

```

Minimum cost maximum flow (old version)

Description: Yoinked from kactl. $\text{cap}[i][j] \neq \text{cap}[j][i]$ is allowed; double edges are not. If costs can be negative, call `setpi` before `maxflow`, but note that negative cost cycles are not supported. To obtain the actual flow, look at positive values only. Note: duplicate edges and anti-parallel edges are not allowed.

Complexity: $\mathcal{O}(E^2)$ o.o.

```

-----f0549f
d41d8c// #include <bits/extc++.h>
d41d8c
9f43acconst ll INF = numeric_limits<ll>::max() / 4;
e7aa4dtypedef vector<ll> VL;
e7aa4d
7600c3struct MCMF {
70940d     int N;

```

```

17badf vector<vi> ed, red;
180f43 vector<VL> cap, flow, cost;
2da736 vi seen;
0aaea7 VL dist, pi;
8a6a40 vector<pii> par;
8a6a40
c625a3 MCMF(int N) :
40dc47     N(N), ed(N), red(N), cap(N, VL(N)), flow(cap), cost(
    cap),
    seen(N), dist(N), pi(N), par(N) {}
0ff3f1
0ff3f1
0d5aaf void addEdge(int from, int to, ll cap, ll cost) {
dfe9fd     this->cap[from][to] = cap;
2746cf     this->cost[from][to] = cost;
0aeef0     ed[from].push_back(to);
2e301a     red[to].push_back(from);
0d7444 }
0d7444
d51da6 void path(int s) {
1bbf83     fill(all(seen), 0);
a47d93     fill(all(dist), INF);
940b6d     dist[s] = 0; ll di;
940b6d
253bf8 __gnu_pbds::priority_queue<pair<ll, int>> q;
7cbcf8 vector<decltype(q)::point_iterator> its(N);
ceeedd q.push({0, s});
ceeedd
131c47
3e7e39 auto relax = [&](int i, ll cap, ll cost, int dir) {
e73b04     ll val = di - pi[i] + cost;
995e7c     if (cap && val < dist[i]) {
9ffb7f         dist[i] = val;
6f572c         par[i] = {s, dir};
        if (its[i] == q.end()) its[i] = q.push({-dist[i]
    }, i);
        else q.modify(its[i], {-dist[i], i});
    }
    }
    };
    while (!q.empty()) {
        s = q.top().second; q.pop();
        seen[s] = 1; di = dist[s] + pi[s];
        for (int i : ed[s]) if (!seen[i])
            relax(i, cap[s][i] - flow[s][i], cost[s][i], 1);
        for (int i : red[s]) if (!seen[i])
            relax(i, flow[i][s], -cost[i][s], 0);
    }
    rep(i,0,N) pi[i] = min(pi[i] + dist[i], INF);
}
pair<ll, ll> maxflow(int s, int t) {
    ll totflow = 0, totcost = 0;
    while (path(s), seen[t]) {
        ll fl = INF;
        for (int p,r,x = t; tie(p,r) = par[x], x != s; x =
            p)
            fl = min(fl, r ? cap[p][x] - flow[p][x] : flow[x]
    ] [p]);
        totflow += fl;
        for (int p,r,x = t; tie(p,r) = par[x], x != s; x =
            p)
            if (r) flow[p][x] += fl;
            else flow[x][p] -= fl;
        }
        rep(i,0,N) rep(j,0,N) totcost += cost[i][j] * flow[i]
    ] [j];
        return {totflow, totcost};
    }
}
// If some costs can be negative, call this before
maxflow:
ad4fa8 void setpi(int s) { // (otherwise, leave this out)
da8610     fill(all(pi), INF); pi[s] = 0;
f5fcf8     int it = N, ch = 1; ll v;

```

```

ebc38e    while (ch-- && it--)
544ef3      rep(i,0,N) if (pi[i] != INF)
ab9631        for (int to : ed[i]) if (cap[i][to])
9b7ba1          if ((v = pi[i] + cost[i][to]) < pi[to])
10724d            pi[to] = v, ch = 1;
a95142      assert(it >= 0); // negative cost cycle
e98539    }
f0549f};

```

Minimum vertex cover

Description: Yoinked from kactl. 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.
Complexity: Idk, look code.

```

d41d8c// #include "DFS_matching.h"
d41d8c
0ba9d3vi cover(vector<vi>& g, int n, int m) {
cb5948    vi match(m, -1);
372cb7    int res = dfsMatching(g, match);
b8f9d0    vector<bool> lfound(n, true), seen(m);
60f20a    for (int it : match) if (it != -1) lfound[it] = false;
d5d915    vi q, cover;
3be03d    rep(i,0,n) if (lfound[i]) q.push_back(i);
813047    while (!q.empty()) {
e11082        int i = q.back(); q.pop_back();
19fc1a        lfound[i] = 1;
113bda        for (int e : g[i]) if (!seen[e] && match[e] != -1) {
1aca58            seen[e] = true;
3b97a6            q.push_back(match[e]);
b97b04        }
b947f3    }
570cd5    rep(i,0,n) if (!lfound[i]) cover.push_back(i);
a12f34    rep(i,0,m) if (seen[i]) cover.push_back(n+i);
8dba7    assert(sz(cover) == res);
6300f6    return cover;
da4196}

```

Strongly connected components

Description: Yoinked from kactl. Finds strongly connected components in a directed graph. If vertices u, v belong to the same component, we can reach u from v and vice versa.
Usage: `scc(graph, [&] (vi& v) { ... })` visits all components in reverse topological order. `comp[i]` holds the component index of a node (a component only has edges to components with lower index). `ncomps` will contain the number of components.
Complexity: $\mathcal{O}(E + V)$.

```

508bd7vi val, comp, z, cont;
c218d3int Time, ncomps;
d31820template<class G, class F> int dfs(int j, G& g, F& f) {
9b6eaf    int low = val[j] = ++Time, x; z.push_back(j);
ed28ae    for (auto e : g[j]) if (comp[e] < 0)
3cf550        low = min(low, val[e] ?: dfs(e,g,f));
3cf550
903808    if (low == val[j]) {
12fcbe        do {
b81dc2            x = z.back(); z.pop_back();
c3db61            comp[x] = ncomps;
6ddcbd            cont.push_back(x);
cf1bb0        } while (x != j);
122be2        f(cont); cont.clear();
d65942        ncomps++;
6e1ce2    }
862574    return val[j] = low;
ab59d0}
c745fa
8d248c    template<class G, class F> void scc(G& g, F f) {
46ec08        int n = sz(g);
        val.assign(n, 0); comp.assign(n, -1);

```

```

011e3c    Time = ncomps = 0;
383892    rep(i,0,n) if (comp[i] < 0) dfs(i, g, f);
76b5c9}

```

Topological sort

Description: Yoinked from kactl. Given is an oriented graph. Output is an ordering of vertices, such that there are edges only from left to right. If there are cycles, the returned list will have size smaller than n – nodes reachable from cycles will not be returned.
Usage: `vi res = topoSort(gr);`
Complexity: $\mathcal{O}(V + E)$.

```

01eaf1vi topoSort(const vector<vi>& gr) {
cd1a35    vi indeg(sz(gr)), ret;
611d40    for (auto& li : gr) for (int x : li) indeg[x]++;
942024    queue<int> q; // use priority_queue for lexic. largest ans.
3ae360    rep(i,0,sz(gr)) if (indeg[i] == 0) q.push(i);
779e30    while (!q.empty()) {
1d3439        int i = q.front(); // top() for priority queue
2a2af3        ret.push_back(i);
9447f3        q.pop();
d83bd2        for (int x : gr[i])
94c522            if (--indeg[x] == 0) q.push(x);
9b0e88    }
435aa0    return ret;
66a137}

```

Weighted bipartite matching

Description: Yoinked from kactl. 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 $N \leq M$.
Complexity: $\mathcal{O}(N^2M)$.

```

325ee8pair<int, vi> hungarian(const vector<vi> &a) {
497519    if (a.empty()) return {0, {}};
ec9978    int n = sz(a) + 1, m = sz(a[0]) + 1;
0c9f93    vi u(n), v(m), p(m), ans(n - 1);
64fc2f    rep(i,1,n) {
9a06cd        p[0] = i;
c3251b        int j0 = 0; // add "dummy" worker 0
3b3e45        vi dist(m, INT_MAX), pre(m, -1);
ced645        vector<bool> done(m + 1);
564738        do { // dijkstra
2c1b77            done[j0] = true;
6773fe            int i0 = p[j0], j1, delta = INT_MAX;
0023e6            rep(j,1,m) if (!done[j]) {
264023                auto cur = a[i0 - 1][j - 1] - u[i0] - v[j];
41fd29                if (cur < dist[j]) dist[j] = cur, pre[j] = j0;
f7e9b7                if (dist[j] < delta) delta = dist[j], j1 = j;
31ae76            }
14a4d0            rep(j,0,m) {
7ceba5                if (done[j]) u[p[j]] += delta, v[j] -= delta;
84199f                else dist[j] -= delta;
6cc461            }
b39843            j0 = j1;
45ce9c        } while (p[j0]);
6cc3ef        while (j0) { // update alternating path
971e56            int j1 = pre[j0];
632eb8            p[j0] = p[j1], j0 = j1;
26ae9e        }
9ae72cf    }
78ec8c    rep(j,1,m) if (p[j]) ans[p[j] - 1] = j - 1;
ae202a    return {-v[0], ans}; // min cost
1e0fe9}

```

Two SAT

Description: Yoinked from kactl. Solves 2-SAT.
Usage: `TwoSat ts(n)` where n is the number of variables. `ts.either(i,j)` means that either i or j must be true. `ts.setValue(i)` means that i must be true. `ts.atMostOne(1)` means that at most one of the variables in l can be true. `ts.solve()` returns true iff it is solvable. `ts.values` will contain one possible solution. Negated variables are represented by bit-inversions ($\sim x$).
Complexity: $\mathcal{O}(N + E)$ where N is the number of variables and E is the number of clauses.

```

d9d94estruct TwoSat {
257c73    int N;
a0af70    vector<vi> gr;
7c0806    vi values; // 0 = false, 1 = true
c1fbac    TwoSat(int n = 0) : N(n), gr(2*n) {}
e10f30    int addVar() { // (optional)
b4b080        gr.emplace_back();
ca34a5        gr.emplace_back();
0f7e62        return N++;
8e7f67    }
1446f5    void either(int f, int j) {
5e1028        f = max(2*f, -1-2*f);
bc62d9        j = max(2*j, -1-2*j);
7f876f        gr[f].push_back(j^1);
511183        gr[j].push_back(f^1);
f602cc    }
cbc333    void setValue(int x) { either(x, x); }
69157f    void atMostOne(const vi& li) { // (optional)
7d932b        if (sz(li) <= 1) return;
7721ca        int cur = ~li[0];
66f796        rep(i,2,sz(li)) {
28a590            int next = addVar();
3de60b            either(cur, ~li[i]);
8bda68            either(cur, next);
001557            either(~li[i], next);
f470ff            cur = ~next;
7cdc2a        }
f21674        either(cur, ~li[1]);
06911d    }
594dbb    vi val, comp, z; int time = 0;
92303b    int dfs(int i) {
fa1d30        int low = val[i] = ++time, x; z.push_back(i);
c93f40        for(int e : gr[i]) if (!comp[e])
c634a9            low = min(low, val[e] ?: dfs(e));
a0ccd1        if (low == val[i]) do {
cf7006            x = z.back(); z.pop_back();
2c346c            comp[x] = low;
a8f0bd            if (values[x>>1] == -1)
fb7b0d                values[x>>1] = x&1;
5a0145        } while (x != i);
3fe09e        return val[i] = low;
088d97    }
12670e    bool solve() {
07c73a        values.assign(N, -1);
a75f85        val.assign(2*N, 0); comp = val;
27da39        rep(i,0,2*N) if (!comp[i]) dfs(i);
a77564        rep(i,0,N) if (comp[2*i] == comp[2*i+1]) return 0;
95beae        return 1;
4fdfc4    }
5f9706};

```


Maths

Chinese remainder theorem

Description: Yoinked from kactl. `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 $mn < 2^{62}$.
Complexity: $\mathcal{O}(\log n)$.

```
-----bd5cec
441d8c// #include "Euclid.h"
441d8c
24a21811 crt(ll a, ll m, ll b, ll n) {
6cb862     if (n > m) swap(a, b), swap(m, n);
8f59af     ll x, y, g = euclid(m, n, x, y);
7424cf     assert((a - b) % g == 0); // else no solution
eae82a     x = (b - a) % n * x % n / g * m + a;
000521     return x < 0 ? x + m*n/g : x;
04d93a}
```

Continued fractions

Description: Yoinked from kactl. 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 ∞ ; if x is the root of a degree 2 polynomial then a 's eventually become cyclic.
Complexity: $\mathcal{O}(\log n)$.

```
-----dd6c5e
0705catypedef double d; // for N ~ 1e7; long double for N ~ 1
e9
d72231pair<ll, ll> approximate(d x, ll N) {
709975     ll LP = 0, LQ = 1, P = 1, Q = 0, inf = LLONG_MAX; d y
= x;
63f648     for (;;) {
32bc0f         ll lim = min(P ? (N-LP) / P : inf, Q ? (N-LQ) / Q :
inf),
82cd25             a = (ll)floor(y), b = min(a, lim),
12b990             NP = b*P + LP, NQ = b*Q + LQ;
426849             if (a > b) {
426849                 // If b > a/2, we have a semi-convergent that
gives us a
426849                 // better approximation; if b = a/2, we *may* have
one.
426849                 // Return {P, Q} here for a more canonical
approximation.
f5e16c                 return (abs(x - (d)NP / (d)NQ) < abs(x - (d)P / (d)
Q)) ?
3c2b26                     make_pair(NP, NQ) : make_pair(P, Q);
451a2f             }
e56f08             if (abs(y = 1/(y - (d)a)) > 3*N) {
32957f                 return {NP, NQ};
ec2d82             }
db887b             LP = P; P = NP;
ed0e32             LQ = Q; Q = NQ;
a15756     }
dd6c5e}
```

Determinant

Description: Yoinked from kactl. Calculates determinant of a matrix. Destroys the matrix.
Complexity: $\mathcal{O}(N^3)$.

```
-----bd5cec
e36c74double det(vector<vector<double>>& a) {
590c12     int n = sz(a); double res = 1;
d90a91     rep(i,0,n) {
4bd724         int b = i;
309239         rep(j,i+1,n) if (fabs(a[j][i]) > fabs(a[b][i])) b =
j;
c6c8fd         if (i != b) swap(a[i], a[b]), res *= -1;
```

```
658965     res *= a[i][i];
390833     if (res == 0) return 0;
15fcb2     rep(j,i+1,n) {
356eb5         double v = a[j][i] / a[i][i];
979baa         if (v != 0) rep(k,i+1,n) a[j][k] -= v * a[i][k];
ebf330     }
aa3042     }
7feeff     return res;
bd5cec}
```

Divisor Count

Description: Counts number of divisors

```
-----2ff470
b6b22011 divisor_cnt(ll n) {
2967be     ll cnt = 1;
6baf3f     map<ll, ll> factors = factorize(n);
b96605     for (auto p : factors) cnt *= p.second+1;
301d7a     return cnt;
2ff470}
```

Sieve of Eratosthenes

Description: Yoinked from kactl. Prime sieve for generating all primes up to a certain limit. `isprime[i]` is true iff i is a prime.
Complexity: $\text{lim} = 100'000'000 \approx 0.8$ s. Runs 30% faster if only odd indices are stored.

```
-----7c144c
129374const int MAX_PR = 5'000'000;
4c8273bitset<MAX_PR> isprime;
e30526vi eratosthenesSieve(int lim) {
b80135     isprime.set(); isprime[0] = isprime[1] = 0;
b716b2     for (int i = 4; i < lim; i += 2) isprime[i] = 0;
6c665e     for (int i = 3; i*i < lim; i += 2) if (isprime[i])
4c1ab1         for (int j = i*i; j < lim; j += i*2) isprime[j] = 0;
081019     vi pr;
98b2cc     rep(i,2,lim) if (isprime[i]) pr.push_back(i);
379a9c     return pr;
7c144c}
```

Euclid

Description: Yoinked from kactl. Finds two integers x and y , such that $ax + by = \text{gcd}(a, b)$. If a and b are coprime, then x is the inverse of $a \pmod b$.
Complexity: $\mathcal{O}(\log n)$.

```
-----33ba8f
c2276e11 euclid(ll a, ll b, ll &x, ll &y) {
fda33f     if (!b) return x = 1, y = 0, a;
d3dcdb     ll d = euclid(b, a % b, y, x);
05ab91     return y -= a/b * x, d;
33ba8f}
```

Fast fourier transform

Description: Yoinked from kactl. `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: `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.
Complexity: $\mathcal{O}(n \log n)$ with $N = |A| + |B|$. (~ 1 s for $N = 2^{22}$)

```
-----3dd197
bccabctypedef complex<double> C;
b05ddbtypedef vector<double> vd;
760a36void fft(vector<C>& a) {
547c8a     int n = sz(a), L = 31 - __builtin_clz(n);
1ec777     static vector<complex<long double>> R(2, 1);
```

```
1e9f4b     static vector<C> rt(2, 1); // (^ 10% faster if double
)
beb684     for (static int k = 2; k < n; k *= 2) {
af116f         R.resize(n); rt.resize(n);
69a3c0         auto x = polar(1.0L, acos(-1.0L) / k);
148d3c         rep(i,k,2*k) rt[i] = R[i] = i&1 ? R[i/2] * x : R[i
/2];
42ea68     }
d8b6b6     vi rev(n);
394b0e     rep(i,0,n) rev[i] = (rev[i / 2] | (i & 1) << L) / 2;
8afd17     rep(i,0,n) if (i < rev[i]) swap(a[i], a[rev[i]]);
14a253     for (int k = 1; k < n; k *= 2)
9f2153         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) /// include-line
71bb8d             auto x = (double *)&rt[j+k], y = (double *)&a[i+j+
k];
f0fec3             C z(x[0]*y[0] - x[1]*y[1], x[0]*y[1] + x[1]*y[0]);
ab793c             // exclude-line
939962             a[i + j + k] = a[i + j] - z;
a3c605             a[i + j] += z;
de1acd         }
br0709vd     conv(const vd& a, const vd& b) {
368356         if (a.empty() || b.empty()) return {};
cc42f4         vd res(sz(a) + sz(b) - 1);
819e9e         int L = 32 - __builtin_clz(sz(res)), n = 1 << L;
95ab64         vector<C> in(n), out(n);
1f7947         copy(all(a), begin(in));
6e8e10         rep(i,0,sz(b)) in[i].imag(b[i]);
dc6bfc         fft(in);
0ff507         for (C& x : in) x *= x;
a1ed40         rep(i,0,n) out[i] = in[-i & (n - 1)] - conj(in[i]);
d6e709         fft(out);
399c53         rep(i,0,sz(res)) res[i] = imag(out[i]) / (4 * n);
0ac860         return res;
3dd197}
```

Fast fourier transform under arbitrary MOD

Description: Yoinked from kactl. 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 $[0, \text{mod})$.
Complexity: $\mathcal{O}(n \log n)$, where $N = |A| + |B|$ (twice as slow as NTT or FFT).

```
-----b82773
d41d8c// #include "FFT.h"
d41d8c
192b04typedef vector<ll> vl;
1dbf8btemplate<int M> vl convMod(const vl &a, const vl &b) {
ffec4         if (a.empty() || b.empty()) return {};
9094f2         vl res(sz(a) + sz(b) - 1);
2c46a2         int B=32-__builtin_clz(sz(res)), n=1<<B, cut=int(sqrt(M));
21440b         vector<C> L(n), R(n), outs(n), outl(n);
ff2f33         rep(i,0,sz(a)) L[i] = C((int)a[i] / cut, (int)a[i] %
cut);
f13a07         rep(i,0,sz(b)) R[i] = C((int)b[i] / cut, (int)b[i] %
cut);
f8a1f3         fft(L), fft(R);
747bd0         rep(i,0,n) {
153b79             int j = -i & (n - 1);
a19b88             outl[j] = (L[i] + conj(L[j])) * R[i] / (2.0 * n);
1a97e3             outs[j] = (L[i] - conj(L[j])) * R[i] / (2.0 * n) / 1
i;
455f55         }
67d701         fft(outl), fft(outs);
08642a         rep(i,0,sz(res)) {
```

```
8bdaab    ll av = ll(real(outl[i])+.5), cv = ll(imag(outs[i])
+.5);
9ac06e    ll bv = ll(imag(outl[i])+.5) + ll(real(outs[i])+.5);
0af53f    res[i] = ((av % M * cut + bv) % M * cut + cv) % M;
26b37c    }
94c360    return res;
b82773}
```

Fast sieve of Eratosthenes

Description: Yoinked from kactl. Prime sieve for generating all primes smaller than LIM.

Complexity: LIM= 1e9 ≈ 1.5s. Utalizes cache locality.

```
-----4756fc
2409caconst int LIM = 1e6;
04d672bitset<LIM> isPrime;
7fd17evi eratosthenes() {
a11a60    const int S = (int)round(sqrt(LIM)), R = LIM / 2;
058587    vi pr = {2}, sieve(S+1); pr.reserve(int(LIM/log(LIM)
*1.1));
81984e    vector<pii> cp;
d3b762    for (int i = 3; i <= S; i += 2) if (!sieve[i]) {
97f6a7        cp.push_back({i, i * i / 2});
579cfb        for (int j = i * i; j <= S; j += 2 * i) sieve[j] =
1;
e31824    }
91c71c    for (int L = 1; L <= R; L += S) {
8834d0        array<bool, S> block{};
7bcfd5        for (auto &[p, idx] : cp)
1df3ce            for (int i=idx; i < S+L; idx = (i+=p)) block[i-L]
= 1;
ac0862        rep(i,0,min(S, R - L))
3db15e            if (!block[i]) pr.push_back((L + i) * 2 + 1);
4de4a4    }
d77909    for (int i : pr) isPrime[i] = 1;
71024d    return pr;
6b2912}
```

Gauss-Jordan elimination

Description: Yoinked from CP-algorithms. The description is taken from CP-algorithms as well: Following is an implementation of Gauss-Jordan. Choosing the pivot row is done with heuristic: choosing maximum value in the current column. The input to the function gauss is the system matrix *a*. The last column of this matrix is vector *b*. The function returns the number of solutions of the system (0, 1, or ∞). If at least one solution exists, then it is returned in the vector *ans*. Implementation notes:

- The function uses two pointers - the current column *col* and the current row *row*.
- For each variable *x_i*, the value *where(i)* is the line where this column is not zero. This vector is needed because some variables can be independent.
- In this implementation, the current *i* th line is not divided by *a_{ii}* as described above, so in the end the matrix is not identity matrix (though apparently dividing the *i* th line can help reducing errors).
- After finding a solution, it is inserted back into the matrix - to check whether the system has at least one solution or not. If the test solution is successful, then the function returns 1 or inf, depending on whether there is at least one independent variable.

kactl also has code for solving linear systems somewhere in the document, if needed.

Complexity: $O(\min(n,m) \cdot nm)$ – I.e. cubic.

```
-----d847fe
bf69a1const double EPS = 1e-9;
7028f6const int INF = 2; // it doesn't actually have to be
infinity or a big number
7028f6
```

```
5e5247int gauss (vector < vector<double> > a, vector<double> &
ans) {
ce0964    int n = (int) a.size();
6871ca    int m = (int) a[0].size() - 1;
6871ca
d0bcc8    vector<int> where (m, -1);
d32c0b    for (int col=0, row=0; col<m && row<n; ++col) {
a8bbf4        int sel = row;
fd9a8b        for (int i=row; i<n; ++i)
890f89            if (abs (a[i][col]) > abs (a[sel][col]))
sel = i;
36d21e        if (abs (a[sel][col]) < EPS)
8ae3bc            continue;
98b44f        for (int i=col; i<=m; ++i)
2adf03            swap (a[sel][i], a[row][i]);
25bd85        where[col] = row;
25bd85
61999a        for (int i=0; i<n; ++i)
663ab4            if (i != row) {
dae7da                double c = a[i][col] / a[row][col];
dae014                for (int j=col; j<=m; ++j)
05001f                    a[i][j] -= a[row][j] * c;
fcb170            }
d17c22            ++row;
0efca2        }
0efca2
7e1e24    ans.assign (m, 0);
2791e8    for (int i=0; i<m; ++i)
125ed2        if (where[i] != -1)
d66745            ans[i] = a[where[i]][m] / a[where[i]][i];
534241    for (int i=0; i<n; ++i) {
ef58b9        double sum = 0;
a81964        for (int j=0; j<m; ++j)
90e125            sum += ans[j] * a[i][j];
fa6b1c        if (abs (sum - a[i][m]) > EPS)
a57e89            return 0;
df2afc    }
df2afc
f0f556    for (int i=0; i<m; ++i)
208713        if (where[i] == -1)
8f5b03            return INF;
893c4e    return 1;
d847fe}
```

Integer determinant

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

Complexity: $O(n^3)$.

```
-----3313dc
0311ccconst ll mod = 12345;
ea0b38ll det(vector<vector<ll>>& a) {
aeac6f    int n = sz(a); ll ans = 1;
c9d9cd    rep(i,0,n) {
cab51f        rep(j,i+1,n) {
4f621e            while (a[j][i] != 0) { // gcd step
155e04                ll t = a[i][i] / a[j][i];
f94a75                if (t) rep(k,i,n)
618162                    a[i][k] = (a[i][k] - a[j][k] * t) % mod;
4d6748                swap(a[i], a[j]);
cbbac3                ans *= -1;
3e9488            }
7effce        }
7173b1        ans = ans * a[i][i] % mod;
c4c228        if (!ans) return 0;
666fb0    }
cd2f86    return (ans + mod) % mod;
3313dc}
```

Integration

Description: Yoinked from kactl. Simple integration of a function over an interval using Simpson’s rule. The error should be proportional to h^4 , although in practice you will want to verify that the result is stable to desired precision when epsilon changes.

Complexity: $O(n)$ evaluations of *f*.

```
-----4756fc
751e63template<class F> double quad(double a, double b, F f,
const int n = 1000) {
840c14    double h = (b - a) / 2 / n, v = f(a) + f(b);
b84885    rep(i,1,n*2)
e9333e        v += f(a + i*h) * (i&1 ? 4 : 2);
df3a8f    return v * h / 3;
4756fc}
```

Linear Diophantine Equation

Description: See below

```
-----002852
d41d8c/*
d41d8cA Linear Diophantine Equation (in two variables) is an
equation of the general form:
d41d8c
d41d8c$$ax + by = c$$
d41d8c
d41d8cwhere $a$, $b$, $c$ are given integers, and $x$, $y$ are
unknown integers.
d41d8c
d41d8c## The degenerate case
d41d8cA degenerate case that need to be taken care of is when
$a = b = 0$. It is easy to see that we either have no
solutions or infinitely many solutions, depending on
whether $c = 0$ or not. In the rest of this article,
we will ignore this case.
d41d8c
d41d8c## Analytic solution
d41d8c
d41d8cWhen $a \neq 0$ and $b \neq 0$, the equation $ax+by=c$
can be equivalently treated as either of the following
:
d41d8c
d41d8c\begin{gather}
d41d8cax \equiv c \pmod b, \text{\\newline}
d41d8cby \equiv c \pmod a.
d41d8c\end{gather}
d41d8c
d41d8cWithout loss of generality, assume that $b \neq 0$ and
consider the first equation. When $a$ and $b$ are co-
prime, the solution to it is given as
d41d8c
d41d8c$$x \equiv ca^{-1} \pmod b, $$
d41d8c
d41d8cwhere $a^{-1}$ is the [modular inverse](module-inverse.
md) of $a$ modulo $b$.
d41d8c
d41d8cWhen $a$ and $b$ are not co-prime, values of $ax$ modulo
$b$ for all integer $x$ are divisible by $g=\gcd(a, b)$,
so the solution only exists when $c$ is divisible
by $g$. In this case, one of solutions can be found by
reducing the equation by $g$:
d41d8c
d41d8c$$\frac{a}{g} x \equiv \frac{c}{g} \pmod{\frac{b}{g}}. $$
d41d8c
d41d8cBy the definition of $g$, the numbers $a/g$ and $b/g$
are co-prime, so the solution is given explicitly as
d41d8c
d41d8c\begin{cases}
d41d8cx \equiv \frac{c}{g} \frac{a}{g}^{-1} \pmod{\frac{b}{g}}, \\
d41d8cy = \frac{c-ax}{b}.
d41d8c\end{cases}
d41d8c
d41d8c## Algorithmic solution
d41d8c
```

To find one solution of the Diophantine equation with 2 unknowns, you can use the [Extended Euclidean algorithm](extended-euclid-algorithm.md). First, assume that a and b are non-negative. When we apply Extended Euclidean algorithm for a and b , we can find their greatest common divisor g and 2 numbers x_g and y_g such that:

$$ax_g + by_g = g$$

If c is divisible by $g = \gcd(a, b)$, then the given Diophantine equation has a solution, otherwise it does not have any solution. The proof is straight-forward: a linear combination of two numbers is divisible by their common divisor.

Now supposed that c is divisible by g , then we have:

$$a \cdot x_g \cdot \frac{c}{g} + b \cdot y_g \cdot \frac{c}{g} = c$$

Therefore one of the solutions of the Diophantine equation is:

$$x_0 = x_g \cdot \frac{c}{g}, y_0 = y_g \cdot \frac{c}{g}$$

The above idea still works when a or b or both of them are negative. We only need to change the sign of x_0 and y_0 when necessary.

Finally, we can implement this idea as follows (note that this code does not consider the case $a = b = 0$):

```

// gcd(int a, int b, int& x, int& y) {
//     if (b == 0) {
//         x = 1;
//         y = 0;
//         return a;
//     }
//     int x1, y1;
//     int d = gcd(b, a % b, x1, y1);
//     x = y1;
//     y = x1 - y1 * (a / b);
//     return d;
// }
bool find_any_solution(int a, int b, int c, int &x0, int &y0, int &g) {
    g = gcd(abs(a), abs(b), x0, y0);
    if (c % g) {
        return false;
    }
    x0 *= c / g;
    y0 *= c / g;
    if (a < 0) x0 = -x0;
    if (b < 0) y0 = -y0;
    return true;
}

// Getting all solutions
void find_all_solutions(int a, int b, int c, int minx, int maxx, int miny, int maxy) {
    int x, y, g;
    if (!find_any_solution(a, b, c, x, y, g))
        return;
    a /= g;

```

From one solution (x_0, y_0) , we can obtain all the solutions of the given equation.

Let $g = \gcd(a, b)$ and let x_0, y_0 be integers which satisfy the following:

$$ax_0 + by_0 = c$$

Now, we should see that adding b / g to x_0 , and, at the same time subtracting a / g from y_0 will not

break the equality:

$$a(x_0 + \frac{b}{g}) + b(y_0 - \frac{a}{g}) = a x_0 + b y_0 + a \cdot \frac{b}{g} - b \cdot \frac{a}{g} = c$$

Obviously, this process can be repeated again, so all the numbers of the form:

$$x = x_0 + k \cdot \frac{b}{g}, y = y_0 - k \cdot \frac{a}{g}$$

are solutions of the given Diophantine equation.

Moreover, this is the set of all possible solutions of the given Diophantine equation.

Finding the number of solutions and the solutions in a given interval

From previous section, it should be clear that if we don't impose any restrictions on the solutions, there would be infinite number of them. So in this section, we add some restrictions on the interval of x and y , and we will try to count and enumerate all the solutions.

Let there be two intervals: $[min_x; max_x]$ and $[min_y; max_y]$ and let's say we only want to find the solutions in these two intervals.

Note that if a or b is 0 , then the problem only has one solution. We don't consider this case here.

First, we can find a solution which have minimum value of x , such that $x \geq min_x$. To do this, we first find any solution of the Diophantine equation. Then, we shift this solution to get $x \geq min_x$ (using what we know about the set of all solutions in previous section). This can be done in $O(1)$.

Denote this minimum value of x by l_{x1} .

Similarly, we can find the maximum value of x which satisfy $x \leq max_x$. Denote this maximum value of x by r_{x1} .

Similarly, we can find the minimum value of y $(y \geq min_y)$ and maximum values of y $(y \leq max_y)$. Denote the corresponding values of x by l_{x2} and r_{x2} .

The final solution is all solutions with x in intersection of $[l_{x1}, r_{x1}]$ and $[l_{x2}, r_{x2}]$, $r_{x2} \geq l_{x1}$. Let denote this intersection by $[l_x, r_x]$.

Following is the code implementing this idea.

Notice that we divide a and b at the beginning by g .

Since the equation $ax + by = c$ is equivalent to the equation $\frac{a}{g}x + \frac{b}{g}y = \frac{c}{g}$, we can use this one instead and have $\gcd(\frac{a}{g}, \frac{b}{g}) = 1$, which simplifies the formulas.

```

// void shift_solution(int &x, int &y, int a, int b, int cnt) {
//     x += cnt * b;
//     y -= cnt * a;
// }
int find_all_solutions(int a, int b, int c, int minx, int maxx, int miny, int maxy) {
    int x, y, g;
    if (!find_any_solution(a, b, c, x, y, g))
        return 0;
    a /= g;

```

```

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

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

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

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

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

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

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

// Once we have  $l_x$  and  $r_x$ , it is also simple to enumerate through all the solutions. Just need to iterate through  $x = l_x + k \cdot \frac{b}{g}$  for all  $k \geq 0$  until  $x = r_x$ , and find the corresponding  $y$  values using the equation  $ax + by = c$ .

// Find the solution with minimum value of  $x + y$ 
// data-toc-label='Find the solution with minimum value of  $x + y$ '
// Find the solution with minimum value of  $x + y$ 

// Here,  $x$  and  $y$  also need to be given some restriction, otherwise, the answer may become negative infinity.

// The idea is similar to previous section: We find any solution of the Diophantine equation, and then shift the solution to satisfy some conditions.

// Finally, use the knowledge of the set of all solutions to find the minimum:

//  $x' = x + k \cdot \frac{b}{g}, y' = y - k \cdot \frac{a}{g}$ 

// Note that  $x + y$  change as follows:

//  $x' + y' = x + y + k \cdot \frac{b}{g} - k \cdot \frac{a}{g}$ 

// If  $a < b$ , we need to select smallest possible value of  $k$ . If  $a > b$ , we need to select the largest possible value of  $k$ . If  $a = b$ , all solution will have the same sum  $x + y$ .

```

Linear Recurrences

Description: Having a linear recurrence of the form $f(n) = a_1 \cdot f(n-1) + a_2 \cdot f(n-2) \cdots$ can be solved in log time with matrix exponentiation.

```
-----5a1314
eb96bf#define Matrix vector<vector<ll>>
8bc5b1const ll m = 1000000007;
33276bMatrix operator*(const Matrix& a, const Matrix& b) {
7f1348    Matrix c = Matrix(len(a), vector<ll>(len(b[0])));
5f7e41    for (int i = 0; i < len(a); i++) {
2fb27e        for (int j = 0; j < len(b[0]); j++) {
6a6fbc            for (int k = 0; k < len(b); k++) {
ce0135                c[i][j] += a[i][k]*b[k][j]%m;
24ce23                c[i][j] %= m;
fbb356            }
6f6d02        }
4230ff    }
221621    return c;
c21a0e}
c21a0e// DOES THIS WORK? Why dp needed?
f1af06Matrix fast_exp(const Matrix& a, ll b, map<ll, Matrix>&
dp) {
615c66    if (dp.count(b)) return dp[b];
0c86d8    if (b == 1) return a;
c7dcfb    if (b%2) return dp[b] = fast_exp(a, b/2, dp)*
fast_exp(a, b/2, dp)*a;
f1ec45    return dp[b] = fast_exp(a, b/2, dp)*fast_exp(a, b/2,
dp);
283d18}
c56361Matrix operator^(const Matrix& a, ll b) {
7cdd13    map<ll, Matrix> dp;
f2b51c    return fast_exp(a, b, dp);
d8af57}
ecd36avoid linear_recurrence() {
ecd36a    /*
ecd36a        dp[j] += dp[i] * X[i][j] <-- genral case
ecd36a    */
5a1314}
```

Matrix inverse

Description: Yoinked from kactl. 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.

Complexity: $\mathcal{O}(n^3)$.

```
-----ebfff6
4b565bint matInv(vector<vector<double>>& A) {
e91afd    int n = sz(A); vi col(n);
2e69f1    vector<vector<double>> tmp(n, vector<double>(n));
9a9a66    rep(i,0,n) tmp[i][i] = 1, col[i] = i;
9a9a66
8ece41    rep(i,0,n) {
a71041        int r = i, c = i;
3ff7a0        rep(j,i,n) rep(k,i,n)
c8b6a2            if (fabs(A[j][k]) > fabs(A[r][c]))
6b4e10                r = j, c = k;
baa3bb        if (fabs(A[r][c]) < 1e-12) return i;
7482dd        A[i].swap(A[r]); tmp[i].swap(tmp[r]);
c4816d        rep(j,0,n)
6e2f7f            swap(A[j][i], A[j][c]), swap(tmp[j][i], tmp[j][c])
;
6ce940        swap(col[i], col[c]);
59c017        double v = A[i][i];
e17078        rep(j,i+1,n) {
1c2a5d            double f = A[j][i] / v;
3cc4a2            A[j][i] = 0;
9da1ac            rep(k,i+1,n) A[j][k] -= f*A[i][k];
293c3d            rep(k,0,n) tmp[j][k] -= f*tmp[i][k];
4b5802        }
```

```

f7a458    rep(j,i+1,n) A[i][j] /= v;
678f7a    rep(j,0,n) tmp[i][j] /= v;
bbee47    A[i][i] = 1;
}
}
cd352a
cd352a
cd352a    /// forget A at this point, just eliminate tmp
backward
28ee96    for (int i = n-1; i > 0; --i) rep(j,0,i) {
973479        double v = A[j][i];
b3722c        rep(k,0,n) tmp[j][k] -= v*tmp[i][k];
fd4d51    }
fd4d51
09764f    rep(i,0,n) rep(j,0,n) A[col[i]][col[j]] = tmp[i][j];
898124    return n;
ebfff6}
```

Matrix inverse mod prime

Description: Yoinked from kactl. 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.

Complexity: $\mathcal{O}(n^3)$.

```
-----a6f68f
d41d8c// #include "Mod_pow.h"
d41d8c
7025f3int matInv(vector<vector<ll>>& A) {
8d1bdf    int n = sz(A); vi col(n);
ff2cbf    vector<vector<ll>> tmp(n, vector<ll>(n));
ebd124    rep(i,0,n) tmp[i][i] = 1, col[i] = i;
ebd124
4c70b5    rep(i,0,n) {
196537        int r = i, c = i;
163a60        rep(j,i,n) rep(k,i,n) if (A[j][k]) {
843bfc            r = j; c = k; goto found;
670a88        }
43b703        return i;
79369efound:
6e7747        A[i].swap(A[r]); tmp[i].swap(tmp[r]);
994d92        rep(j,0,n) swap(A[j][i], A[j][c]), swap(tmp[j][i],
tmp[j][c]);
f483b9        swap(col[i], col[c]);
a33b6a        ll v = modpow(A[i][i], mod - 2);
221dbc        rep(j,i+1,n) {
4dc1d6            ll f = A[j][i] * v % mod;
820a75            A[j][i] = 0;
191b80            rep(k,i+1,n) A[j][k] = (A[j][k] - f*A[i][k]) % mod
;
2034cf            rep(k,0,n) tmp[j][k] = (tmp[j][k] - f*tmp[i][k]) %
mod;
3af408        }
402ef6        rep(j,i+1,n) A[i][j] = A[i][j] * v % mod;
6e1d6e        rep(j,0,n) tmp[i][j] = tmp[i][j] * v % mod;
7099c7        A[i][i] = 1;
b5fe9f    }
b5fe9f
9c015a    for (int i = n-1; i > 0; --i) rep(j,0,i) {
8a334f        ll v = A[j][i];
f92823        rep(k,0,n) tmp[j][k] = (tmp[j][k] - v*tmp[i][k]) %
mod;
597d8e    }
597d8e
765b04    rep(i,0,n) rep(j,0,n)
43e42e        A[col[i]][col[j]] = tmp[i][j] % mod + (tmp[i][j] < 0
? mod : 0);
d429b2    return n;
a6f68f}
```

Millar-Rabin primality test

Description: Yoinked from kactl. Deterministic Miller-Rabin primality test. Guaranteed to work for numbers up to $7 \cdot 10^{18}$.

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

```
-----60dcd1
d41d8c// #include "Mod_mul_LL.h"
d41d8c
da49edbool isPrime(ull n) {
6e0366    if (n < 2 || n % 6 % 4 != 1) return (n | 1) == 3;
ad415b    ull A[] = {2, 325, 9375, 28178, 450775, 9780504,
1795265022},
13b9b1        s = __builtin_ctzll(n-1), d = n >> s;
60a421    for (ull a : A) { // ^ count trailing zeroes
29e314        ull p = modpow(a%n, d, n), i = s;
4ab836        while (p != 1 && p != n - 1 && a % n && i--)
f7944d            p = modmul(p, p, n);
56ff8c            if (p != n-1 && i != s) return 0;
1fad05        }
3c0060        return 1;
60dcd1}
```

Modular inverses

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

```
-----b4a981
d41d8c// const ll mod = 1000000007, LIM = 200000; ///include-
line
664058ll* inv = new ll[LIM] - 1; inv[1] = 1;
b4a981rep(i,2,LIM) inv[i] = mod - (mod / i) * inv[mod % i] %
mod;
```

Modulo multiplication for 64-bit integers

Description: Yoinked from kactl. Calculate $a \cdot b \bmod c$ (or $a^b \bmod c$) for $0 \leq a, b \leq c \leq 7.2 \cdot 10^{18}$. This runs 2x faster than the naive $(_int128.t)a * b \% M$.

Complexity: $\mathcal{O}(1)$ for modmul, $\mathcal{O}(\log b)$ for modpow.

```
-----bbbd8f
f4cf5btypedef unsigned long long ull;
92e1d3ull modmul(ull a, ull b, ull M) {
00ac89    ll ret = a * b - M * ull(1.L / M * a * b);
21b1bc    return ret + M * (ret < 0) - M * (ret >= (1l)M);
a9c350}
438153ull modpow(ull b, ull e, ull mod) {
c04010    ull ans = 1;
aea873    for (; e; b = modmul(b, b, mod), e /= 2)
f5aa70        if (e & 1) ans = modmul(ans, b, mod);
6d3d5f    return ans;
bbbd8f}
```

Mod pow

Description: Yoinked from kactl. What u think mans. (this interface is used by a few other things, hence included in the document)

Complexity: $\mathcal{O}(\log e)$.

```
-----b83e45
e2e0e3const ll mod = 1000000007; // faster if const
e2e0e3
cceb35ll modpow(ll b, ll e) {
cd37e8    ll ans = 1;
8bc5f9    for (; e; b = b * b % mod, e /= 2)
c96cd7        if (e & 1) ans = ans * b % mod;
a23ec3    return ans;
b83e45}
```

Modular arithmetic

Description: Yoinked from kactl. Simple operators for modular arithmetic. You need to set mod to some number first and then you can use the structure.

```
-----35bf6a
d41d8c// #include "Euclid.h"
d41d8c
4eb587const ll mod = 17; // change to something else
```



```
6655aastruct Mod {
e58fcd    ll x;
316e8ef    Mod(ll xx) : x(xx) {}
9af9c9    Mod operator+(Mod b) { return Mod((x + b.x) % mod); }
884537    Mod operator-(Mod b) { return Mod((x - b.x + mod) %
mod); }
622079    Mod operator*(Mod b) { return Mod((x * b.x) % mod); }
bac0da    Mod operator/(Mod b) { return *this * invert(b); }
727966    Mod invert(Mod a) {
ef23f5        ll x, y, g = euclid(a.x, mod, x, y);
8ac2fc        assert(g == 1); return Mod((x + mod) % mod);
f65d71    }
f9c260    Mod operator^(ll e) {
cc1619        if (!e) return Mod(1);
9708c3        Mod r = *this ^ (e / 2); r = r * r;
1f093c        return e&1 ? *this * r : r;
f16184    }
35bf6ea};
```

Number theoretic transform

Description: Yoinked from kactl. $\text{ntt}(a)$ 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^ab+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 $[0, mod)$. **Complexity:** $\mathcal{O}(n \log n)$.

```
-----ced03d
d41d8c// #include "Mod_pow.h"
d41d8c
b5e822const ll mod = (119 << 23) + 1, root = 62; // =
998244353
b5e822// For p < 2^30 there is also e.g. 5 << 25, 7 << 26, 479
<< 21
b5e822// and 483 << 21 (same root). The last two are > 10^9.
7458catypedef vector<ll> vl;
0ca385void ntt(vl &a) {
c96375    int n = sz(a), L = 31 - __builtin_clz(n);
7bd0b3    static vl rt(2, 1);
668758    for (static int k = 2, s = 2; k < n; k *= 2, s++) {
4c5a31        rt.resize(n);
1759b1        ll z[] = {1, modpow(root, mod >> s)};
2921d8        rep(i,k,2*k) rt[i] = rt[i / 2] * z[i & 1] % mod;
5faa22    }
3ee1db    vi rev(n);
78dccb    rep(i,0,n) rev[i] = (rev[i / 2] | (i & 1) << L) / 2;
158770    rep(i,0,n) if (i < rev[i]) swap(a[i], a[rev[i]]);
225017    for (int k = 1; k < n; k *= 2)
61bd17        for (int i = 0; i < n; i += 2 * k) rep(j,0,k) {
64cbc8            ll z = rt[j + k] * a[i + j + k] % mod, &ai = a[i +
j];
cba978            a[i + j + k] = ai - z + (z > ai ? mod : 0);
4b5040            ai += (ai + z >= mod ? z - mod : z);
35d5bf        }
29a029}
bbaf00Vl conv(const vl &a, const vl &b) {
4001b0    if (a.empty() || b.empty()) return {};
cb0e4e    int s = sz(a) + sz(b) - 1, B = 32 - __builtin_clz(s),
n = 1 << B;
10d0fe    int inv = modpow(n, mod - 2);
5e3527    vl L(a), R(b), out(n);
8e31ec    L.resize(n), R.resize(n);
6415db    ntt(L), ntt(R);
1c4346    rep(i,0,n) out[-i & (n - 1)] = (ll)L[i] * R[i] % mod *
inv % mod;
4af30c    ntt(out);
70c6bc    return {out.begin(), out.begin() + s};
ced03d}
```

Polynomial root finding

Description: Yoinked from kactl. Finds the real roots to a polynomial. **Usage:** polyRoots({{2,-3,1}}, -1e9, 1e9); // solve $x^2-3x+2 = 0$ **Complexity:** $\mathcal{O}(n^2 \log(\frac{1}{\epsilon}))$.

```
-----b00bfe
d41d8c// #include "Polynomial.h"
d41d8c
64af29vector<double> polyRoots(Poly p, double xmin, double
xmax) {
a63eaa    if (sz(p.a) == 2) { return {-p.a[0]/p.a[1]}; }
343f7f    vector<double> ret;
2acf4e    Poly der = p;
8409d9    der.diff();
105e2f    auto dr = polyRoots(der, xmin, xmax);
31d1fe    dr.push_back(xmin-1);
324645    dr.push_back(xmax+1);
5604f0    sort(all(dr));
50119c    rep(i,0,sz(dr)-1) {
d045cc        double l = dr[i], h = dr[i+1];
2748c8        bool sign = p(l) > 0;
ea5d57        if (sign ^ (p(h) > 0)) {
cc4926            rep(it,0,60) { // while (h - l > 1e-8)
40bdf6                double m = (l + h) / 2, f = p(m);
145fe6                if ((f <= 0) ^ sign) l = m;
8da3ef                else h = m;
4f1379            }
f5991f            ret.push_back((l + h) / 2);
1c9b1d        }
d5f24e    }
a514b7    return ret;
b00bfe}
```

Polynomial thing

Description: Yoinked from kactl. Some poly things I guess.

```
-----c9b7b0
213314struct Poly {
640a33    vector<double> a;
aea975    double operator()(double x) const {
4b0030        double val = 0;
1b799c        for (int i = sz(a); i--;) (val *= x) += a[i];
3743d7        return val;
f7a37b    }
187735    void diff() {
462d92        rep(i,1,sz(a)) a[i-1] = i*a[i];
1e1024        a.pop_back();
d447a3    }
cd4862    void divroot(double x0) {
3236c3        double b = a.back(), c; a.back() = 0;
06b4f8        for(int i=sz(a)-1; i--;) c = a[i], a[i] = a[i+1]*x0+
b, b=c;
a.pop_back();
071796    }
43bc43    }
c9b7b0};
```

SOS DP

Description: Some solution from some problem Elias solved. For each of n elements x : The number of elements y such that $x \mid y = x$. The number of elements y such that $x \& y = x$. The number of elements y such that $x \& y \neq 0$. NOTE: if TLE issues, try loop unrolling or C style arrays. **Complexity:** $\mathcal{O}(V \log V + n)$ where V is the maximum value.

```
-----29cc3d
ac9985constexpr const int lgmxV = 20;
e1fff58constexpr const int mxV = 1 << lgmxV;
e1fff58
28d892int main(){
d90c3e    ios_base::sync_with_stdio(0); cin.tie(0); cout.tie
(0);
```

```
c1cd68    int n; cin >> n;
5d1c88    vector<int> v(n);
b9f5bb    for(auto &x : v)cin >> x;
924113    vector<vector<int>> sos1(mxV, vector<int> (lgmxV +
1, 0));
657ed7    vector<vector<int>> sos2(mxV, vector<int> (lgmxV +
1, 0));
2d7593    for(int i = 0; i < n; ++i){
22d226        sos1[v[i]][0]++;
a24c4a        sos2[v[i] ^ (mxV - 1)][0]++;
932fe0    }
5de047    for(int i = 0; i < mxV; ++i){
b88e0d        for(int j = 0; j < lgmxV; ++j){
6965f4            sos1[i][j + 1] = sos1[i][j];
5b556f            sos2[i][j + 1] = sos2[i][j];
73e5b1            if(i & (1 << j)) { sos1[i][j + 1] += sos1[i
- (1 << j)][j]; };
cf7af2            if(i & (1 << j)) { sos2[i][j + 1] += sos2[i
- (1 << j)][j]; };
54565b        }
2735ac        for(int i = 0; i < n; ++i){
61582a            cout << sos1[v[i]][lgmxV] << ' ' << sos2[v[i] ^
(mxV - 1)][lgmxV] << ' ' << n - sos1[v[i] ^ (mxV - 1)
][lgmxV] << '\n';
b94f88        }
f1eea6
29cc3d}
```

Simplex

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

```
-----aa8530
943c93typedef double T; // long double, Rational, double + mod
<P>...
4a7fa3typedef vector<T> vd;
19471c    typedef vector<vd> vvd;
19471c
6296c1const T eps = 1e-8, inf = 1/.0;
20f308#define MP make_pair
80a946#define ltj(X) if(s == -1 || MP(X[j],N[j]) < MP(X[s],N[s
])) s=j
80a946
004b50struct LPSolver {
34f6a6    int m, n;
a8b98c    vi N, B;
a50829    vvd D;
a50829
e8814c    LPSolver(const vvd& A, const vd& b, const vd& c) :
09ecbe        m(sz(b)), n(sz(c)), N(n+1), B(m), D(m+2, vd(n+2)) {
a00ca8        rep(i,0,m) rep(j,0,n) D[i][j] = A[i][j];
eab15d        rep(i,0,m) { B[i] = n+i; D[i][n] = -1; D[i][n+1] =
b[i];}
03bb56        rep(j,0,n) { N[j] = j; D[m][j] = -c[j]; }
4c20cd        N[n] = -1; D[m+1][n] = 1;
dcadf8    }
dcadf8
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;
rep(j,0,n+2) b[j] -= a[j] * inv2;
```

```

ee22d8      b[s] = a[s] * inv2;
df792b    }
d3cb55    rep(j,0,n+2) if (j != s) D[r][j] *= inv;
9e2376    rep(i,0,m+2) if (i != r) D[i][s] *= -inv;
6bf9c5    D[r][s] = inv;
b3404b    swap(B[r], N[s]);
193de8  }
ede257
f695c2    bool simplex(int phase) {
0aa9db      int x = m + phase - 1;
8b65cd      for (;;) {
96f50e        int s = -1;
e72781        rep(j,0,n+1) if (N[j] != -phase) ltj(D[x]);
fcd18c        if (D[x][s] >= -eps) return true;
a7d0e5        int r = -1;
f65882        rep(i,0,m) {
01fd61          if (D[i][s] <= eps) continue;
8af3f7          if (r == -1 || MP(D[i][n+1] / D[i][s], B[i])
= i;          < MP(D[r][n+1] / D[r][s], B[r])) r
170720        }
23b7a6        if (r == -1) return false;
100fe3        pivot(r, s);
d81c2f      }
62b7d3    }
62b7d3
T solve(vd &x) {
48ae53      int r = 0;
b0718e      rep(i,1,m) if (D[i][n+1] < D[r][n+1]) r = i;
cc8cd8      if (D[r][n+1] < -eps) {
dc34d7        pivot(r, n);
fbrfb80        if (!simplex(2) || D[m+1][n+1] < -eps) return -inf
09ceea      ;
6b2bed      rep(i,0,m) if (B[i] == -1) {
9aa881        int s = 0;
db9144        rep(j,1,n+1) ltj(D[i]);
d11ba5        pivot(i, s);
213eb8      }
36d5c1    }
e286bf    bool ok = simplex(1); x = vd(n);
002972    rep(i,0,m) if (B[i] < n) x[B[i]] = D[i][n+1];
8dddea    return ok ? D[m][n+1] : inf;
bc3870  }
aa8530};
```

Solve linear equations

Description: Yoinked from kactl. Solves $A * x = b$. If there are multiple solutions, an arbitrary one is returned. Returns rank, or -1 if no solutions. Data in A and b is lost.

Complexity: $\mathcal{O}(n^2m)$.

```

-----44c9ab
ae03ae    typedef vector<double> vd;
1784ea    const double eps = 1e-12;
1784ea
dbdd92    int solveLinear(vector<vd>& A, vd& b, vd& x) {
2cfc7      int n = sz(A), m = sz(x), rank = 0, br, bc;
61ac86      if (n) assert(sz(A[0]) == m);
274909      vi col(m); iota(all(col), 0);
274909
27c9a7      rep(i,0,n) {
cdb1df        double v, bv = 0;
9bbd0f        rep(r,i,n) rep(c,i,m)
889ccc          if ((v = fabs(A[r][c])) > bv)
4cafdf            br = r, bc = c, bv = v;
236408        if (bv <= eps) {
008896          rep(j,i,n) if (fabs(b[j]) > eps) return -1;
b9eea0          break;
e8dea5        }
e256ad        swap(A[i], A[br]);
f84bc6        swap(b[i], b[br]);
b1eb75        swap(col[i], col[bc]);
```

```

0bea42      rep(j,0,n) swap(A[j][i], A[j][bc]);
bc2598      bv = 1/A[i][i];
292cf7      rep(j,i+1,n) {
416953        double fac = A[j][i] * bv;
f8d04b        b[j] -= fac * b[i];
fe2cdd        rep(k,i+1,m) A[j][k] -= fac*A[i][k];
34df26      }
cc5189      rank++;
66cd8f    }
66cd8f
5f0090      x.assign(m, 0);
21a20d      for (int i = rank; i--;) {
5fa421        b[i] /= A[i][i];
9d7b80        x[col[i]] = b[i];
a0bd4f        rep(j,0,i) b[j] -= A[j][i] * b[i];
55ec26      }
ec3430      return rank; // (multiple solutions if rank < m)
44c9ab}
```

Solve linear equations extended

Description: Yoinked from kactl. To get all uniquely determined values of x back from SolveLinear, make the following changes:

```

-----08e495
d41d8c// #include "Solve_linear.h"
d41d8c
f9498c    rep(j,0,n) if (j != i) // instead of rep(j,i+1,n)
f9498c// ... then at the end:
3b9d44    X.assign(m, undefined);
45bf44    rep(i,0,rank) {
22b426      rep(j,rank,m) if (fabs(A[i][j]) > eps) goto fail;
46800e      x[col[i]] = b[i] / A[i][i];
08e495    fail::; }
```

Phi function

Description: Yoinked from kactl. $Euler's \phi$ 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
fd7760    const int LIM = 5000000;
b4bbf9    int phi[LIM];
b4bbf9
e30f2f    void calculatePhi() {
70ba16      rep(i,0,LIM) phi[i] = i&1 ? i : i/2;
9fb18b      for (int i = 3; i < LIM; i += 2) if(phi[i] == i)
4678aa        for (int j = i; j < LIM; j += i) phi[j] -= phi[j] /
i;
cf7d6d}
```

Strings

Aho-Corasick automaton

Description: Yoinked from kactl. 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.

Complexity: $26 \cdot \mathcal{O}(N)$ to construct, where N = sum of length of patterns. find(x) is $\mathcal{O}(N)$, where N = length of x . findAll is $\mathcal{O}(NM)$.

```

f35677-----
51f3fc    struct AhoCorasick {
ba2f89      enum {alpha = 26, first = 'A'}; // change this!
b513ea      struct Node {
b513ea        // (nmatches is optional)
724017        int back, next[alpha], start = -1, end = -1,
nmatches = 0;
c328c4        Node(int v) { memset(next, v, sizeof(next)); }
9e9dd8      };
99b2f6      vector<Node> N;
b8ee95      vi backp;
abd02c      void insert(string& s, int j) {
77af37        assert(!s.empty());
77757e        int n = 0;
503315        for (char c : s) {
77af36          int& m = N[n].next[c - first];
eebd80          if (m == -1) { n = m = sz(N); N.emplace_back(-1);
}
else n = m;
}
if (N[n].end == -1) N[n].start = j;
backp.push_back(N[n].end);
N[n].end = j;
N[n].nmatches++;
}
AhoCorasick(vector<string>& pat) : N(1, -1) {
rep(i,0,sz(pat)) insert(pat[i], i);
N[0].back = sz(N);
N.emplace_back(0);

queue<int> q;
for (q.push(0); !q.empty(); q.pop()) {
int n = q.front(), prev = N[n].back;
rep(i,0,alpha) {
int &ed = N[n].next[i], y = N[prev].next[i];
if (ed == -1) ed = y;
else {
N[ed].back = y;
(N[ed].end == -1 ? N[ed].end : backp[N[ed].
start])
= N[y].end;
N[ed].nmatches += N[y].nmatches;
q.push(ed);
}
}
}
vi find(string word) {
int n = 0;
vi res; // ll count = 0;
for (char c : word) {
n = N[n].next[c - first];
res.push_back(N[n].end);
// count += N[n].nmatches;
}
return res;
}
vector<vi> findAll(vector<string>& pat, string word) {
vi r = find(word);
vector<vi> res(sz(word));
rep(i,0,sz(word)) {
int ind = r[i];
while (ind != -1) {
res[i - sz(pat[ind]) + 1].push_back(ind);
ind = backp[ind];
}
}
return res;
}
```

```
e9b14e    }
f35677};
```

String hashing

Description: Yoinked from kactl. Self-explanatory methods for string hashing.

```
-----2d2a67
d41d8c// Arithmetic mod 2^64-1. 2x slower than mod 2^64 and
more
d41d8c// code, but works on evil test data (e.g. Thue-Morse,
where
d41d8c// ABBA... and BAAB... of length 2^10 hash the same mod
2^64).
d41d8c// "typedef ull H;" instead if you think test data is
random,
d41d8c// or work mod 10^9+7 if the Birthday paradox is not a
problem.
41d24atypedef uint64_t ull;
95307estruct H {
80cf70 ull x; H(ull x=0) : x(x) {}
1f94d8 H operator+(H o) { return x + o.x + (x + o.x < x); }
98ccfa H operator-(H o) { return *this + ~o.x; }
df4ab4 H operator*(H o) { auto m = (__uint128_t)x * o.x;
4efff44 return H((ull)m + (ull)(m >> 64)); }
f17b1d ull get() const { return x + !~x; }
f8f361 bool operator==(H o) const { return get() == o.get(); }
}
442de3 bool operator<(H o) const { return get() < o.get(); }
40d284};
27469dstatic const H C = (1l)1e11+3; // (order ~ 3e9; random
also ok)
27469d
ff1702struct HashInterval {
5588fd vector<H> ha, pw;
76de09 HashInterval(string& str) : ha(sz(str)+1), pw(ha) {
48388c pw[0] = 1;
77ec24 rep(i,0,sz(str))
aadaa5 ha[i+1] = ha[i] * C + str[i],
7ccac8 pw[i+1] = pw[i] * C;
47478c }
90ccf6 H hashInterval(int a, int b) { // hash [a, b)
143f12 return ha[b] - ha[a] * pw[b - a];
c82f9f }
91d0ec};
91d0ec
690e88vector<H> getHashes(string& str, int length) {
6b6485 if (sz(str) < length) return {};
355ef9 H h = 0, pw = 1;
22cbb3 rep(i,0,length)
a4ed43 h = h * C + str[i], pw = pw * C;
707e0f vector<H> ret = {h};
e1bef5 rep(i,length,sz(str)) {
54d254 ret.push_back(h = h * C + str[i] - pw * str[i-length
]);
}
0f079f }
4311cc return ret;
2f26bc}
2f26bc
2d2a67H hashString(string& s){H h{}; for(char c:s) h=h*C+c;
return h;}
```

Knuth-Morris-Pratt algorithm

Description: Yoinked from kactl. Finds all occurrences of a pattern in a string. `p[x]` computes the length of the longest prefix of `s` that ends at `x`, other than `s[0...x]` itself (abacaba -> 0010123).

Complexity: $\mathcal{O}(n)$.

```
-----d4375c
132da4vi pi(const string& s) {
adaa84 vi p(sz(s));
049209 rep(i,1,sz(s)) {
```

```
c043e7 int g = p[i-1];
f6d6b9 while (g && s[i] != s[g]) g = p[g-1];
21a657 p[i] = g + (s[i] == s[g]);
}
6c1f11 return p;
63e9df
9cb7fc}
9cb7fc
7c0957vi match(const string& s, const string& pat) {
58166d vi p = pi(pat + '\0' + s), res;
608c16 rep(i,sz(p)-sz(s),sz(p))
68390b if (p[i] == sz(pat)) res.push_back(i - 2 * sz(pat));
c66a2a return res;
d4375c}
```

Manacher

Description: Yoinked from kactl. 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).

Complexity: $\mathcal{O}(n)$.

```
-----e7ad79
fc122barray<vi, 2> manacher(const string& s) {
f03a96 int n = sz(s);
daf4bc array<vi,2> p = {vi(n+1), vi(n)};
4d112b rep(z,0,2) for (int i=0,l=0,r=0; i < n; i++) {
510161 int t = r-i+!z;
2a504d if (i<r) p[z][i] = min(t, p[z][l+t]);
3613b8 int L = i-p[z][i], R = i+p[z][i]-!z;
508df3 while (L>=1 && R+1<n && s[L-1] == s[R+1])
c79056 p[z][i]++, L--, R++;
168507 if (R>r) l=L, r=R;
21a1fb }
4ae824 return p;
e7ad79}
```

Min rotation

Description: Yoinked from kactl. Finds the lexicographically smallest rotation of a string.

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

Complexity: $\mathcal{O}(n)$.

```
-----d07a42
5fa8d6int minRotation(string s) {
e7cf68 int a=0, N=sz(s); s += s;
5ca080 rep(b,0,N) rep(k,0,N) {
f656d5 if (a+k == b || s[a+k] < s[b+k]) {b += max(0, k-1);
break;}
20f912 if (s[a+k] > s[b+k]) { a = b; break; }
b2e25e }
5fafdc return a;
d07a42}
```

Rolling Hash

Description: RH prepare string `s`, and hash gives the hash of the substring `[l, r]` inclusive. `ib` is `pow(b, -1, MD)`, `MD` should be prime

Complexity: $\mathcal{O}(n)$ preprocessing, $\mathcal{O}(1)$ hash.

```
-----2e25f9
c5aa9estruct RH {
64eb2a int MD, n, b, ib; // b is base, ib inverse base mod MD
3b195e vector<int> p, ip, hs;
011265 RH(string s, int _b = 69, int _ib = 579710149, int _MD
= 1e9 + 7) : MD(_MD), n((int)s.size()), b(_b), ib(_ib
), p(n), ip(n), hs(n) { // _b = 63, _ib = 698412843,
MD = 1e9 + 207
74c3ce p[0] = ip[0] = 1;
d28127 hs[0] = s[0];
5bb806 for(int i = 1; i < n; ++i){
3f448a p[i] = (1l) p[i - 1] * b % MD;
4870cc ip[i] = (1l) ip[i - 1] * ib % MD;
66aa32 hs[i] = ((1l) s[i] * p[i] + hs[i - 1]) % MD; // s[
i] can be changed to some hash function
```

```
adef78 }
1e7e6b }
16c258 int hash(int l, int r){
d9aae2 return (1l) (hs[r] - (l ? hs[l - 1] : 0) + MD) * ip[
1] % MD;
1379de }
2e25f9};
```

Suffix array

Description: Yoinked from kactl. 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.

Complexity: $\mathcal{O}(n \log n)$ per update/query

```
-----38db9f
3f48c2struct SuffixArray {
1ff472 vi sa, lcp;
e88c75 SuffixArray(string& s, int lim=256) { // or
basic_string<int>
91bda5 int n = sz(s) + 1, k = 0, a, b;
58cf39 vi x(all(s)+1), y(n), ws(max(n, lim)), rank(n);
74da6a sa = lcp = y, iota(all(sa), 0);
c642ff for (int j = 0, p = 0; p < n; j = max(1, j * 2), lim
= p) {
323b5c p = j, iota(all(y), n - j);
70faae rep(i,0,n) if (sa[i] >= j) y[p++] = sa[i] - j;
066cf5 fill(all(ws), 0);
499169 rep(i,0,n) ws[x[i]]++;
3549fa rep(i,1,lim) ws[i] += ws[i - 1];
fa1cc1 for (int i = n; i--;) sa[--ws[x[y[i]]]] = y[i];
f9337c swap(x, y), p = 1, x[sa[0]] = 0;
3a641e rep(i,1,n) a = sa[i - 1], b = sa[i], x[b] =
248123 (y[a] == y[b] && y[a + j] == y[b + j]) ? p - 1 :
p++;
f30252 }
0b3927 rep(i,1,n) rank[sa[i]] = i;
271bfc for (int i = 0, j; i < n - 1; lcp[rank[i++]] = k)
f2d8ac for (k && k--, j = sa[rank[i] - 1];
2b582e s[i + k] == s[j + k]; k++);
22a139 }
38db9f};
```

Suffix automaton

Description: Standard suffix automaton. Does what you'd expect.

Usage: See example main function below. This was thrown in last minute from a working cses solution.

Complexity: $\mathcal{O}(\log n)$ per update/query

```
-----3d234e
10747dstruct SA {
31fdad struct State {
fad143 int length;
7e049f int link;
ec43e2 int next[26];
209696 int cnt;
0a95ea bool is_clone;
dafc14 int first_pos;
0fbc43 State(int _length, int _link) :
578718 length(_length),
8f88e0 link(_link),
05402c cnt(0),
c214c3 is_clone(false),
c445b2 first_pos(-1)
df1390 {
24aab0 memset(next, -1, sizeof(next));
c13476 }
575a7c };
5c435a std::vector <State> states;
```

```

0c2d55 int size;
dadfd1 int last;
26a9fe bool did_init_count;
7c701c int str_len;
339b92 bool did_init_css;
edd2c0 SA() :
247d2e states(1, State(0, -1)),
27dd74 size(1),
f6f1cc last(0),
b25e35 did_init_count(false),
5b001a str_len(0),
1d383e did_init_css(false)
18e6a6 { }
1ca6810 void push(char c) {
525d03 str_len++;
8f2dae did_init_count = false;
4a4bd8 did_init_css = false;
26359b int cur = size;
d5aba5 states.resize(++size, State(states[last].length + 1,
-1));
01ccfe states[cur].first_pos = states[cur].length - 1;
68f4ae int p = last;
5f2312 while (p != -1 && states[p].next[c - 'a'] == -1) {
67b05d states[p].next[c - 'a'] = cur;
73ba4b p = states[p].link;
0db291 }
a55669 if (p == -1) {
0cd45a states[cur].link = 0;
577086 } else {
c98ad9 int q = states[p].next[c - 'a'];
6024e1 if (states[p].length + 1 == states[q].length) {
1de958 states[cur].link = q;
930e14 } else {
aed05d int clone = size;
afb223 states.resize(++size, State(states[p].length +
1, states[q].link));
4443c2 states[clone].is_clone = true;
afb2e1 memcpy(states[clone].next, states[q].next,
sizeof(State::next));
61ac3d states[clone].first_pos = states[q].first_pos;
13bea7 while (p != -1 && states[p].next[c - 'a'] == q)
{
627f1c states[p].next[c - 'a'] = clone;
411652 p = states[p].link;
20432b }
34a7da states[q].link = states[cur].link = clone;
98914e }
0461f9 }
591347 last = cur;
301567 }
d0cce2 bool exists(const std::string& pattern) {
0ffabb int node = 0;
13a5cf int index = 0;
192e18 while (index < (int) pattern.length() && states[node]
].next[pattern[index] - 'a'] != -1) {
efffe7 node = states[node].next[pattern[index] - 'a'];
cbf0e9 index++;
709389 }
356eef return index == (int) pattern.size();
4db848 }
0ff9b8 int count(const std::string& pattern) {
66e217 if (!did_init_count) {
13d2c1 did_init_count = true;
702df7 for (int i = 1; i < size; i++) {
57b2d4 states[i].cnt = !states[i].is_clone;
24878a }
9cd77 std::vector<std::vector<int>> of_length(str_len
+ 1);
d9c5db for (int i = 0; i < size; i++) {
c408de of_length[states[i].length].push_back(i);
9d793e }
e08272 for (int l = str_len; l >= 0; l--) {

```

```

e9fd3e for (int node : of_length[l]) {
ff7da1 if (states[node].link != -1) {
fa5d99 states[states[node].link].cnt += states[node]
].cnt;
c92599 }
9f0d9a }
418535 }
ce47a0 }
c62dc8 int node = 0;
1a6274 int index = 0;
d32f26 while (index < (int) pattern.length() && states[node]
].next[pattern[index] - 'a'] != -1) {
6d8dce node = states[node].next[pattern[index] - 'a'];
1ad0b3 index++;
edf68d }
72ab54 return index == (int) pattern.size() ? states[node].
cnt : 0;
f7682f }
f397ab int first_occ(const std::string& pattern) {
53dadcd int node = 0;
6bbd47 int index = 0;
442e13 while (index < (int) pattern.length() && states[node]
].next[pattern[index] - 'a'] != -1) {
652cc2 node = states[node].next[pattern[index] - 'a'];
8a968d index++;
ef6d88 }
a59113 return index == (int) pattern.size() ? states[node].
first_pos - (int) pattern.size() + 1 : -1;
a65c30 }
9afeb2 size_t count_substrings() {
a7f74b static std::vector<size_t> dp;
9e504d if (!did_init_css) {
9a3afa did_init_css = true;
fce801 dp = std::vector<size_t>(size, 0);
75426a auto dfs = [&](auto&& self, int node) -> size_t {
673f0b if (node == -1) {
0b0f06 return 0;
9fa531 }
99b459 if (dp[node]) {
ac9ba2 return dp[node];
519c50 }
983e54 dp[node] = 1;
14020f for (int i = 0; i < 26; i++) {
2a5625 dp[node] += self(self, states[node].next[i]);
515699 }
0260ef return dp[node];
b1fb1b };
a3a17c dfs(dfs, 0);
d8b4f0 }
8b5414 return dp[0] - 1;
e1c0a8 }
4b005c };
4b005c };
4b005c // usage example: Repeating Substring submission on cses
2f5768 .fi
109b3e int main() {
c0bcd4 std::ios::sync_with_stdio(0); std::cin.tie(0);
c9c93c std::string s; std::cin >> s;
0c8f98 int n; std::cin >> n;
3b67c6 SA sa;
5bd287 for (char c : s) {
27d539 sa.push(c);
c64da9 }
66d2ad sa.count("");
bb09b1 int len = -1;
af0b43 int ind = -1;
f4d141 for (int i = 1; i < sa.size; i++) {
e65645 if (sa.states[i].cnt > 1) {
9b1e2f if (len < sa.states[i].length) {
becb1e len = sa.states[i].length;
5af6dc ind = sa.states[i].first_pos - len + 1;
}
}
}

```

```

3b9795 }
0f2256 }
f0ebc0 if (len == -1) {
de5034 std::cout << "-1\n";
c8c5ae return 0;
a99b6e }
f38c31 for (int i = 0; i < len; i++) {
0d86ab std::cout << s[i + ind];
42f1ff }
228fb9 std::cout << "\n";
3d234e }

```

Suffix tree

Description: Yoinked from kactl. Ukkonen's algorithm for online suffix tree construction. Each node contains indices [l, r] into the string, and a list of child nodes. Suffixes are given by traversals of this tree, joining [l, r] substrings. The root is 0 (has l = -1, r = 0), non-existent children are -1. To get a complete tree, append a dummy symbol – otherwise it may contain an incomplete path (still useful for substring matching, though).

Complexity: $26 \cdot \mathcal{O}(n)$.

```

3a1cf8 struct SuffixTree {
749ba4 enum { N = 200010, ALPHA = 26 }; // N ~ 2*maxlen+10
e2aa04 int toi(char c) { return c - 'a'; }
c011c3 string a; // v = cur node, q = cur position
b1fb1b int t[N][ALPHA], l[N], r[N], p[N], s[N], v=0, q=0, m=2;
b1fb1b
557107 void ukkadd(int i, int c) { suff:
a822f9 if (r[v]<=q) {
f7dbb8 if (t[v][c]==-1) { t[v][c]=m; l[m]=i;
810ece p[m++]=v; v=s[v]; q=r[v]; goto suff; }
54a4b2 v=t[v][c]; q=l[v];
6b58ee }
8cb728 if (q==-1 || c==toi(a[q])) q++; else {
141302 l[m+1]=i; p[m+1]=m; l[m]=l[v]; r[m]=q;
694340 p[m]=p[v]; t[m][c]=m+1; t[m][toi(a[q])]=v;
d13a03 l[v]=q; p[v]=m; t[p[m]][toi(a[l[m]])]=m;
d6dde8 v=s[p[m]]; q=l[m];
afa4f2 while (q<r[m]) { v=t[v][toi(a[q])]; q+=r[v]-l[v];
}
8b395d if (q==r[m]) s[m]=v; else s[m]=m+2;
3ea06d q=r[v]-(q-r[m]); m+=2; goto suff;
451104 }
0b7995 }
0b7995
d444af SuffixTree(string a) : a(a) {
88dda6 fill(r,r+N,sz(a));
cb23ac memset(s, 0, sizeof s);
ab059b memset(t, -1, sizeof t);
0a991e fill(t[1],t[1]+ALPHA,0);
3b133d s[0] = 1; l[0] = l[1] = -1; r[0] = r[1] = p[0] = p
[1] = 0;
372f3e rep(i,0,sz(a)) ukkadd(i, toi(a[i]));
e6a350 }
e6a350 // example: find longest common substring (uses ALPHA
e6a350 = 28)
66b1ec pii best;
66dfb7 int lcs(int node, int i1, int i2, int olen) {
dc2e91 if (l[node] <= i1 && i1 < r[node]) return 1;
bb99fd if (l[node] <= i2 && i2 < r[node]) return 2;
088265 int mask = 0, len = node ? olen + (r[node] - l[node]
]) : 0;
3619dd rep(c,0,ALPHA) if (t[node][c] != -1)
7c5642 mask |= lcs(t[node][c], i1, i2, len);
f72e9f if (mask == 3)
f77c80 best = max(best, {len, r[node] - len});
2dd106 return mask;
526a4c }

```



```
e2ea9b static pii LCS(string s, string t) {
a8444c     SuffixTree st(s + (char)('z' + 1) + t + (char)('z' +
                2));
fb62d1     st.lcs(0, sz(s), sz(s) + 1 + sz(t), 0);
ccb226     return st.best;
9dc48b }
aae0b8};
```

Z function

Description: Yoinked from kactl. `z[x]` computes the length of the longest common prefix of `s[i:]` and `s`, except `z[0] = 0`. (abacaba -> 0010301)

Complexity: $\mathcal{O}(n)$.

```
444be3 vi Z(const string& S) {
cf9608     vi z(sz(S));
661ccc     int l = -1, r = -1;
4fa4a3     rep(i, 1, sz(S)) {
f3a3fa         z[i] = i >= r ? 0 : min(r - i, z[i - 1]);
145eb7         while (i + z[i] < sz(S) && S[i + z[i]] == S[z[i]])
d97d13             z[i]++;
18e9a9         if (i + z[i] > r)
0d1f1b             l = i, r = i + z[i];
b90033     }
cc04e8     return z;
ee09e2 }
```

Various

Bump allocator

Description: Yoinked from kactl. When you need to dynamically allocate many objects and don't care about freeing them. "new X" otherwise has an overhead of something like 0.05us + 16 bytes per allocation.

```
411d8c // Either globally or in a single class:
2b9528 static char buf[450 << 20];
73a19f void* operator new(size_t s) {
3d5bc2     static size_t i = sizeof buf;
c17d54     assert(s < i);
e69924     return (void*)&buf[i - s];
0c4c77 }
745db2 void operator delete(void*) {}
```

Fast integer input

Description: Yoinked from kactl. USE THIS IF TRYING TO CONSTANT TIME OPTIMIZE SOLUTION READING IN LOTS OF INTEGERS!!! Read an integer from stdin. Usage requires your program to pipe in input from file.

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

Complexity: About 5x as fast as `cin/scanf`.

```
c304cb inline char gc() { // like getchar()
b5396f     static char buf[1 << 16];
0c057f     static size_t bc, be;
62a7c2     if (bc >= be) {
c5125f         buf[0] = 0, bc = 0;
bba013         be = fread(buf, 1, sizeof(buf), stdin);
e9a035     }
973215     return buf[bc++]; // returns 0 on EOF
0261eb }
b36081 int readInt() {
b8176b     int a, c;
d5554c     while ((a = gc()) < 40);
```

```
bc51ee if (a == '-') return -readInt();
e7b4e7 while ((c = gc()) >= 48) a = a * 10 + c - 480;
5eb5ba return a - 48;
7b3c70 }
```

Fast knapsack

Description: Yoinked from kactl. 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.

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

```
4d398e int knapsack(vi w, int t) {
cca251     int a = 0, b = 0, x;
ba551c     while (b < sz(w) && a + w[b] <= t) a += w[b++];
3f688f     if (b == sz(w)) return a;
e2b1c9     int m = *max_element(all(w));
11fd10     vi u, v(2*m, -1);
7d8c93     v[a+m-t] = b;
682d61     rep(i, b, sz(w)) {
c83dfe         u = v;
e898de         rep(x, 0, m) v[x+w[i]] = max(v[x+w[i]], u[x]);
51a6b1         for (x = 2*m; --x > m;) rep(j, max(0, u[x]), v[x])
5e6b65             v[x-w[j]] = max(v[x-w[j]], j);
d2b439     }
700a1e     for (a = t; v[a+m-t] < 0; a--);
e4db33     return a;
b20ccc }
```

Fast mod reduction

Description: Yoinked from kactl. Compute $a \% b$ about 5 times faster than usual, where b is constant but not known at compile time. Returns a value congruent to $a \pmod b$ in the range $[0, 2b)$. (proven correct)

```
f4cf5b typedef unsigned long long ull;
a7666a struct FastMod {
a51f1f     ull b, m;
551bab     FastMod(ull b) : b(b), m((-1ULL / b) {}
010304     ull reduce(ull a) { // a % b + (0 or b)
c7e7c1         return a - (ull)((_uint128_t(m) * a) >> 64) * b;
03d237     }
751a02 };
```

Interval container

Description: Yoinked from kactl. 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).

Complexity: $\mathcal{O}(\log n)$ per update/query

```
d91403 set<pii>::iterator addInterval(set<pii>& is, int L, int
                R) {
905a62     if (L == R) return is.end();
117079     auto it = is.lower_bound({L, R}), before = it;
b0184b     while (it != is.end() && it->first <= R) {
ba1bd0         R = max(R, it->second);
a98b04         before = it = is.erase(it);
381108     }
6d817b     if (it != is.begin() && (--it)->second >= L) {
7d7c26         L = min(L, it->first);
d2faed         R = max(R, it->second);
8ea38c         is.erase(it);
5783d8     }
72f28b     return is.insert(before, {L, R});
d574d7 }
d574d7 void removeInterval(set<pii>& is, int L, int R) {
154403     if (L == R) return;
969cd4     auto it = addInterval(is, L, R);
f20f53     auto r2 = it->second;
51cff5 }
```

```
d09c40 if (it->first == L) is.erase(it);
c1de31 else (int&)it->second = L;
b4d977 if (R != r2) is.emplace(R, r2);
edce47 }
```

Interval cover

Description: Yoinked from kactl. 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).

Complexity: $\mathcal{O}(n \log n)$.

```
4fce64 template<class T>
68ecb6 vi cover(pair<T, T> G, vector<pair<T, T>> I) {
1a8df4     vi S(sz(I)), R;
fa5016     iota(all(S), 0);
0e2216     sort(all(S), [&](int a, int b) { return I[a] < I[b];
                });
a166e4     T cur = G.first;
4d4739     int at = 0;
7cae10     while (cur < G.second) { // (A)
3ef5e6         pair<T, int> mx = make_pair(cur, -1);
bfcc53         while (at < sz(I) && I[S[at]].first <= cur) {
201b40             mx = max(mx, make_pair(I[S[at]].second, S[at]));
9b4b3e             at++;
470978         }
f1e40b         if (mx.second == -1) return {};
206822         cur = mx.first;
93267c         R.push_back(mx.second);
cd0c49     }
c6ad6f     return R;
9e9d8d }
```

Knuth DP optimization

Description: Yoinked from kactl. 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 $a \leq b \leq c \leq d$.

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

```
d41d8c // generic implementation frmo cp-algorithms:
f5cbfa int solve() {
90d984     int N;
c75671     ... // read N and input
f99d43     int dp[N][N], opt[N][N];
7ed932     auto C = [&](int i, int j) {
163e32         ... // Implement cost function C.
a996a2     };
ce7594     for (int i = 0; i < N; i++) {
ea7c0e         opt[i][i] = i;
cd2bbe         ... // Initialize dp[i][i] according to the
problem
eb6571     }
82a94b     for (int i = N-2; i >= 0; i--) {
6c2b32         for (int j = i+1; j < N; j++) {
1bdeb6             int mn = INT_MAX;
b4c515             int cost = C(i, j);
0b2d26             for (int k = opt[i][j-1]; k <= min(j-1, opt[
i+1][j]); k++) {
27debe                 if (mn >= dp[i][k] + dp[k+1][j] + cost)
{
4b284e                     opt[i][j] = k;
e7e3b1                     mn = dp[i][k] + dp[k+1][j] + cost;
3a469f                 }
96a5d4             }
```

```
03175a      dp[i][j] = mn;
1659d3    }
ce8b93  }
008b9c    cout << dp[0][N-1] << endl;
210610}
```

Longest increasing subsequence

Description: Yoinked from kactl. Computes the longest increasing subsequence of a sequence.
Complexity: $\mathcal{O}(n \log n)$.

```
-----2932a0
8d31fctemplate<class I> vi lis(const vector<I>& S) {
be1376  if (S.empty()) return {};
67f1da  vi prev(sz(S));
c1cccf  typedef pair<I, int> p;
47f7ae  vector<p> res;
5dc126  rep(i,0,sz(S)) {
5dc126    // change 0 -> i for longest non-decreasing
          subsequence
f6ef94    auto it = lower_bound(all(res), p{S[i], 0});
b92110    if (it == res.end()) res.emplace_back(), it = res.
          end()-1;
26a0a3    *it = {S[i], i};
e3bc5b    prev[i] = it == res.begin() ? 0 : (it-1)->second;
f2ee22  }
```

```
b33eaf  int L = sz(res), cur = res.back().second;
3f07f2  vi ans(L);
ffa1d5  while (L--) ans[L] = cur, cur = prev[cur];
4593b0  return ans;
2932a0}
```

Small ptr

Description: Yoinked from kactl. A 32-bit pointer that points into BumpAllocator memory.

```
-----2dd6c9
d41d8c// #include "Bump_allocator.h"
d41d8c
0cae25template<class T> struct ptr {
dfb41f  unsigned ind;
953a17  ptr(T* p = 0) : ind(p ? unsigned((char*)p - buf) : 0)
          {
347c02    assert(ind < sizeof buf);
bda3ee  }
36a046  T& operator*() const { return *(T*)(buf + ind); }
c82e36  T* operator->() const { return &*this; }
dd2aa9  T& operator[](int a) const { return (&*this)[a]; }
881391  explicit operator bool() const { return ind; }
2dd6c9};
```

Xor Basis

Description: basis of vectors in Z_2^d

```
-----e54101
fc63fbint basis[d]; // basis[i] keeps the mask of the vector
          whose f value is i
89edd0int sz; // Current size of the basis
c03addvoid insertVector(int mask) {
29c5ff  for (int i = 0; i < d; i++) {
812690    if ((mask & 1 << i) == 0) continue; // continue if i
          != f(mask)
812690
          if (!basis[i]) { // If there is no basis vector with
dc6b5a    the i'th bit set, then insert this vector into the
          basis
          basis[i] = mask;
          ++sz;
          return;
          }
          mask ^= basis[i]; // Otherwise subtract the basis
          vector from this vector
3547b9  }
e54101}
```