

Contents

1 Basic	
1.1 bashrc	1
1.2 vimrc	1
1.3 Default	1
1.4 Debug code	1
1.5 Pragma	1
2 Data Structure	
2.1 Black Magic	1
2.2 Lazy Segment Tree	2
2.3 Persistent BIT	2
2.4 Treap	2
2.5 DSU Undo	3
2.6 Lichao Tree	3
2.7 Linear Basis	3
2.8 Heavy Light Decomposition	3
2.9 Link Cut Tree	3
3 Graph	
3.1 Bridge CC	4
3.2 Vertex BCC	4
3.3 Strongly Connected Component	5
3.4 Two SAT	5
3.5 Virtual Tree	5
3.6 Dominator Tree	5
3.7 Dinic	6
3.8 Min Cost Max Flow	6
3.9 Stoer Wagner Algorithm	7
3.10 General Matching	7
3.11 Hopcroft Karp Algorithm	7
3.12 Directed MST	7
3.13 Edge Coloring	8
4 Geometry	
4.1 Basic	8
4.2 2D Convex Hull	9
4.3 Farthest Pair	9
4.4 Minkowski Sum	9
4.5 Circle	10
4.6 Tangent Lines of Circle and Point	10
4.7 Tangent Lines of Circles	10
4.8 Delaunay Triangular	11
4.9 Half Plane Intersection	11
4.10 Point In Convex	12
4.11 Voronoi Diagram	12
5 String	
5.1 KMP	12
5.2 Z Value	12
5.3 Suffix Array	12
5.4 AC Automaton	13
5.5 Booth Algorithm	13
5.6 Manacher Algorithm	13
5.7 Palindromic Tree	13
6 Math	
6.1 Lemma And Theory	13
6.1.1 Pick's Theorem	13
6.1.2 Euler's Planar Graph Theorem	13
6.1.3 Modular inversion recurrence	13
6.2 Numbers	13
6.2.1 Catalan number	13
6.2.2 Primes	13
6.3 Extgcd	13
6.4 Chinese Remainder Theorem	14
6.5 Linear Sieve	14
6.6 Fast Walsh Transform	14
6.7 Floor Sum	14
6.8 Linear Programming	14
6.9 Miller Rabin	15
6.10 Pollard's Rho	15
6.11 Gauss Elimination	15
6.12 Determinant	15
6.13 Fast Fourier Transform	15
6.14 Primes NTT	16
6.15 Number Theory Transform	16
6.16 Berlekamp Massey	16
6.17 Fraction	16
7 Misc	
7.1 Josephus Problem	16

1 Basic

1.1 bashrc

```
| xmodmap -e 'clear Lock' -e 'keycode 0x42 = Escape'
```

1.2 vimrc

```
set nu rnu is ls=2 hls ts=4 sw=4 et sts=4 ai bs=2 sc
acd mouse=a encoding=utf-8
syn on
filetype plugin indent on
colo desert
inoremap {<CR> {<CR><Esc>O
inoremap jj <Esc>
nnoremap <F8> :w <bar> !g++ -std=c++17 % -o %:r -O2<CR>
nnoremap <F9> :w <bar> !g++ -std=c++17 % -o %:r -Wall -
Wextra -Wconversion -Wshadow -Wfatal-errors -
fsanitize=undefined,address -g -Dgenshin <CR>
nnoremap <F10> :!./%:r <CR>
ca Hash w !cpp -dD -P -fpreprocessed \ | tr -d '[:space
:]' \ | md5sum \ | cut -c-6
```

1.3 Default [5c3786]

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

using ll = long long;
using ull = unsigned long long;
using ld = long double;
using uint = unsigned int;
const double EPS = 1e-8;
const int INF = 0x3F3F3F3F;
const ll LINF = 46 * 11(1e17);
const int MOD = 1e9+7;
const int maxn = 1e5 + 25;

signed main() { ios::sync_with_stdio(0); cin.tie(0);
}
```

1.4 Debug code [c1cc07]

```
#ifdef genshin
#include <experimental/iterator>
#define safe cerr<<__PRETTY_FUNCTION__<<" Line "<<
__LINE__<<" safe\n"
#define debug(a...) debug_#a, a)
#define print(a...) print_#a, a)
void debug_(const char *s, auto ...a) {
cerr << "\e[1;31m(" << s << ") = (" ;
int f = 0;
(..., (cerr << (f++ ? ", " : "") << a));
cerr << ")\e[0m\n";
}
void print_(const char *s, auto L, auto R) {
cerr << "\e[1;33m[ " << s << " ] = [ " ;
using namespace experimental;
copy(L, R, make_ostream_joiner(cerr, ", "));
cerr << "]\e[0m\n";
}
#else
#define safe ((void)0)
#define debug(...) safe
#define print(...) safe
#endif
```

1.5 Pragma [d346d7]

```
#pragma GCC optimize("Ofast,no-stack-protector")
#pragma GCC optimize("no-math-errno,unroll-loops")
#pragma GCC target("sse,sse2,sse3,ssse3,sse4")
#pragma GCC target("popcnt,abm,mmx,avx,tune=native")
```

2 Data Structure

2.1 Black Magic [68c435]

```
template<typename T>
using pbds_tree = __gnu_pbds::tree<T, null_type, less<T>
>,
rb_tree_tag, tree_order_statistics_node_update>;
// find_by_order: like array accessing, order_of_key
// join: (one should smaller than the other)
// split(v, b): <= v are a, > v are b
template<typename T, typename T2>
using hash_table = __gnu_pbds::gp_hash_table<T, T2>;
// ht.find(a) ht[a] = v
template<typename T>
using rope = __gnu_cxx::rope<T>;
// array stands for string &s, char *s or int *a
```

```
// push_back, pop_back, insert(pos, x)
// insert(pos, array, len): from pos, insert len
// elements of array
// append(array, pos, len): append len elements from
// pos of array
// substr(pos, len), at(pos), erase(pos, len)
// copy(pos, len, array): from pos, replace len
// elements from array
// Use = and + to concat substrs, += to append element
// O(log n) or O(1). Use pointer and new for persistent
// use:
vector<rope<int>>* r(n);
r[0] = new rope<int>();
r[i] = new rope<int>(*r[i - 1]);
r[i]->push_back(i);
```

2.2 Lazy Segment Tree [a46c93]

```
// 0-based, [l, r)
// Remember to call init
struct tag {
    // Construct identity element
    tag() {}
    // apply tag
    tag& operator+=(const tag &b) {
        return *this;
    }
};
struct node {
    // Construct identity element
    node() {}
    // Merge two nodes
    node operator+(const node &b) const {
        node res = node();
        return res;
    }
    // Apply tag to this node
    void operator()(const tag &t) {}
};
template<typename N, typename T>
struct lazy_segtree {
    N arr[maxn << 1];
    T tag[maxn];
    int n;
    void init(const vector<N> &a) {
        n = a.size();
        for (int i = 0; i < n; i++)
            arr[i + n] = a[i], tag[i] = T();
        for (int i = n - 1; i; i--)
            arr[i] = arr[i << 1] + arr[i << 1 | 1];
    }
    void upd(int p, T v) {
        if (p < n)
            tag[p] += v;
        arr[p](v);
    }
    void pull(int p) {
        for (p >>= 1; p; p >>= 1) {
            arr[p] = arr[p << 1] + arr[p << 1 | 1];
            arr[p](tag[p]);
        }
    }
    void push(int p) {
        for (int h = __lg(p); h; h--) {
            int i = p >> h;
            upd(i << 1, tag[i]);
            upd(i << 1 | 1, tag[i]);
            tag[i] = T();
        }
    }
    void edt(int l, int r, T v) {
        int tl = l + n, tr = r + n - 1;
        push(tl); push(tr);
        for (l += n, r += n; l < r; l >>= 1, r >>= 1) {
            if (l & 1)
                upd(l++, v);
            if (r & 1)
                upd(--r, v);
        }
        pull(tl); pull(tr);
    }
    N que(int l, int r) {
```

```
N resl = N(), resr = N();
int tl = l + n, tr = r + n - 1;
push(tl); push(tr);
for (l += n, r += n; l < r; l >>= 1, r >>= 1) {
    if (l & 1)
        resl = resl + arr[l++];
    if (r & 1)
        resr = arr[--r] + resr;
}
return resl + resr;
};
```

2.3 Persistent BIT [3226a9]

```
// Remember to call init
struct fenwicktree {
    vector<int> arr[maxn], sum[maxn];
    void init() {
        for (int i = 0; i < maxn; i++) {
            arr[i].push_back(0);
            sum[i].push_back(0);
        }
    }
    // edit must be called with non-increasing t
    void edt(int p, int v, int t) {
        for (; p < maxn; p += p & -p) {
            arr[p].push_back(t);
            sum[p].push_back(sum[p].back() + v);
        }
    }
    inline int get(int i, int t) {
        return sum[i][(upper_bound(arr[i].begin(), arr[i].
            end(), t) - arr[i].begin()) - 1];
    }
    int que(int l, int r, int t) {
        if (r < 1 || l == 0)
            return 0;
        int res = 0;
        for (; r; r -= r & -r)
            res += get(r, t);
        for (l--; l; l -= l & -l)
            res -= get(l, t);
        return res;
    }
    // return the last p s.t. bit[dw..up][1..p] < v
    // maxn - 1 if v > sum and 0 if v = 0
    int bin(int v, int dw, int up) {
        int res = 0, p = 0;
        for (int i = 1 << 20; i; i >= 1) {
            int g = get(p | i, up) - get(p | i, dw - 1);
            if ((p | i) < maxn && res + g < v) {
                res += g;
                p |= i;
            }
        }
        return p;
    }
} bit;
```

2.4 Treap [cbb731]

```
__gnu_cxx::sfmt19937 rnd(48763);
namespace Treap {
    struct node {
        int size, pri;
        node *lc, *rc, *pa;
        node() : size(1), pri(rnd()), lc(0), rc(0), pa(0) {}
        void pull() {
            size = 1; pa = 0;
            if (lc) { size += lc->size; lc->pa = this; }
            if (rc) { size += rc->size; rc->pa = this; }
        }
    };
    int SZ(node *x) { return x ? x->size : 0; }
    node *merge(node *L, node *R) {
        if (!L || !R) return L ? L : R;
        if (L->pri > R->pri)
            return L->rc = merge(L->rc, R), L->pull(), L;
        else
            return R->lc = merge(L, R->lc), R->pull(), R;
    }
    void splitBySize(node *o, int k, node *&L, node *&R) {
        if (!o) { L = R = 0; }
```

```

else if (int s = SZ(o->lc) + 1; s <= k) {
    L = o, splitBySize(o->rc, k-s, L->rc, R);
    L->pull();
}
else {
    R = o, splitBySize(o->lc, k, L, R->lc);
    R->pull();
}
} // SZ(L) == k
int getRank(node *o) { // 1-base
    int r = SZ(o->lc) + 1;
    for (; o->pa; o = o->pa)
        if (o->pa->rc == o) r += SZ(o->pa->lc) + 1;
    return r;
}
} // namespace Treap, not tested

```

2.5 DSU Undo [d41d8c]

```

// If undo is not needed, remove st, time() and
// rollback()
// e stands for size (roots) and parent
// int t = dsu.tim(); ...; uf.rollback(t);
struct dsu_undo {
    vector<int> e;
    vector<pair<int, int>> st;
    dsu_undo(int n) : e(n, -1) {}
    int size(int x) { return -e[find(x)]; }
    int find(int x) { return e[x] < 0 ? x : find(e[x]); }
    int time() { return st.size(); }
    void rollback(int t) {
        for (int i = time(); i-- > t;)
            e[st[i].first] = st[i].second;
        st.resize(t);
    }
    bool join(int a, int b) {
        a = find(a), b = find(b);
        if (a == b) return false;
        if (e[a] > e[b]) swap(a, b);
        st.push_back({a, e[a]});
        st.push_back({b, e[b]});
        e[a] += e[b]; e[b] = a;
        return true;
    }
};

```

2.6 Lichao Tree [ef9ec7]

```

struct lichao { // maxn: range
    struct line {
        ll a, b;
        line(): a(0), b(0) {} // or b(LINF) if min
        line(ll a, ll b): a(a), b(b) {}
        ll operator()(ll x) { return a * x + b; } // v[x]
        after Li san hua
    } arr[maxn << 2];
    void insert(int l, int r, int id, line x) {
        int m = (l + r) >> 1;
        if (arr[id](m) < x(m))
            swap(arr[id], x);
        if (l == r - 1)
            return;
        if (arr[id].a < x.a)
            insert(m, r, id << 1 | 1, x);
        else
            insert(l, m, id << 1, x);
    } // change to > if query min
    // maxn -> v.size() after Li san hua
    void insert(ll a, ll b) { insert(0, maxn, 1, line(a, b)); }
    ll que(int l, int r, int id, int p) {
        if (l == r - 1)
            return arr[id](p);
        int m = (l + r) >> 1;
        if (p < m)
            return max(arr[id](p), que(l, m, id << 1, p));
        return max(arr[id](p), que(m, r, id << 1 | 1, p));
    } // chnage to min if query min
    // maxn -> v.size() after Li san hua
    ll que(int p) { return que(0, maxn, 1, p); }
} tree;

```

2.7 Linear Basis [3a3430]

```

template<int BITS>
struct linear_basis {
    array<uint64_t, BITS> basis;
    linear_basis() { basis.fill(0); }
    void insert(uint64_t x) {
        for (int i = BITS - 1; i >= 0; i--) if ((x >> i) & 1) {
            if (basis[i] == 0) {
                basis[i] = x;
                return;
            }
            x ^= basis[i];
        }
    }
    bool valid(uint64_t x) {
        for (int i = BITS - 1; i >= 0; i--)
            if ((x >> i) & 1) x ^= basis[i];
        return x == 0;
    }
    uint64_t operator()(int i) { return basis[i]; }
}; // max xor sum: greedy from high bit
// min xor sum: zero(if possible) or min_element

```

2.8 Heavy Light Decomposition [2c4577]

```

/* Requirements:
 * N := the count of nodes
 * edge[N] := the edges of the graph
 * Can be modified:
 * tree := Segment Tree or other data structure
 */
struct heavy_light_decomposition {
    int dep[N], pa[N], hea[N], hev[N], pos[N], t;
    int dfs(int u) {
        int mx = 0, sz = 1;
        hev[u] = -1;
        for(int v : edge[u]) {
            if(v == pa[u])
                continue;
            pa[v] = u;
            dep[v] = dep[u] + 1;
            int c = dfs(v);
            if(c > mx)
                mx = c, hev[u] = v;
            sz += c;
        }
        return sz;
    }
    void find_head(int u, int h) {
        hea[u] = h;
        pos[u] = t++; // 0-indexed !!!
        if(~hev[u])
            find_head(hev[u], h);
        for(int v : edge[u])
            if(v != pa[u] && v != hev[u])
                find_head(v, v);
    }
    void init(int rt) {
        dfs(rt, rt);
        find_head(rt, rt);
    }
    /* It is necessary to edit below for every use */
    void edt(int a, int b, int v) {
    }
    int query(int a, int b) { // query path sum
        int res = 0;
        for(; hea[a] != hea[b]; a = pa[hea[a]]) {
            if(dep[hea[a]] < dep[hea[b]])
                swap(a, b);
            res += tree.que(pos[hea[a]], pos[a] + 1);
        }
        if(dep[a] > dep[b])
            swap(a, b);
        return res + tree.que(pos[a], pos[b] + 1);
    }
} hld;

```

2.9 Link Cut Tree [d41d8c]

```

namespace LCT {
    const int N = 1e5 + 25;

```

```

int pa[N], ch[N][2];
ll dis[N], prv[N], tag[N];
vector<pair<int, int>> edge[N];
vector<pair<ll, ll>> eve;
inline bool dir(int x) { return ch[pa[x]][1] == x; }
inline bool is_root(int x) { return ch[pa[x]][0] != x
    && ch[pa[x]][1] != x; }
inline void rotate(int x) {
    int y = pa[x], z = pa[y], d = dir(x);
    if(!is_root(y))
        ch[z][dir(y)] = x;
    pa[x] = z;
    ch[y][d] = ch[x][!d];
    if(ch[x][!d])
        pa[ch[x][!d]] = y;
    ch[x][!d] = y;
    pa[y] = x;
}
inline void push_tag(int x) {
    if(!tag[x])
        return;
    prv[x] = tag[x];
    if(ch[x][0])
        tag[ch[x][0]] = tag[x];
    if(ch[x][1])
        tag[ch[x][1]] = tag[x];
    tag[x] = 0;
}
void push(int x) {
    if(!is_root(x))
        push(pa[x]);
    push_tag(x);
}
inline void splay(int x) {
    push(x);
    while(!is_root(x)) {
        if(int y = pa[x]; !is_root(y))
            rotate(dir(y) == dir(x) ? y : x);
        rotate(x);
    }
}
inline void access(ll t, int x) {
    int lst = 0, tx = x;
    while(x) {
        splay(x);
        if(lst) {
            ch[x][1] = lst;
            eve.push_back({prv[x] + dis[x], t + dis[x]});
        }
        lst = x;
        x = pa[x];
    }
    splay(tx);
    if(ch[tx][0])
        tag[ch[tx][0]] = t;
}
void dfs(int u) {
    prv[u] = -LINF;
    for(const auto &[v, c] : edge[u]) {
        if(v == pa[u])
            continue;
        pa[v] = u;
        ch[u][1] = v;
        dis[v] = dis[u] + c;
        dfs(v);
    }
}
};

```

3 Graph

3.1 Bridge CC [25b45e]

```

namespace bridge_cc {
    vector<int> tim, low;
    stack<int, vector<int>> st;
    int t, bcc_id;
    void dfs(int u, int p, const vector<vector<pair<int,
        int>>> &edge, vector<int> &pa) {
        tim[u] = low[u] = t++;
        st.push(u);
        for(const auto &[v, id] : edge[u]) {
            if(id == p)

```

```

                continue;
            if(tim[v])
                low[u] = min(low[u], tim[v]);
            else {
                dfs(v, id, edge, pa);
                if(low[v] > tim[u]) {
                    int x;
                    do {
                        pa[x = st.top()] = bcc_id;
                        st.pop();
                    } while(x != v);
                    bcc_id++;
                }
                else
                    low[u] = min(low[u], low[v]);
            }
        }
    }
    vector<int> solve(const vector<vector<pair<int, int>>> &edge) { // (to, id)
        int n = edge.size();
        tim.resize(n);
        low.resize(n);
        t = bcc_id = 1;
        vector<int> pa(n);

        for(int i = 0; i < n; i++) {
            if(!tim[i]) {
                dfs(i, -1, edge, pa);
                while(!st.empty()) {
                    pa[st.top()] = bcc_id;
                    st.pop();
                }
                bcc_id++;
            }
        }
        return pa;
    } // return bcc id(start from 1)
};

```

3.2 Vertex BCC [6be52c]

```

class bicon_cc {
private:
    int n, ecnt;
    vector<vector<pair<int, int>>> G;
    vector<int> bcc, dfn, low, st;
    vector<bool> ap, ins;
    void dfs(int u, int f) {
        dfn[u] = low[u] = dfn[f] + 1;
        int ch = 0;
        for(auto [v, t]: G[u]) if(v != f) {
            if(!ins[t]) {
                st.push_back(t);
                ins[t] = true;
            }
            if(dfn[v]) {
                low[u] = min(low[u], dfn[v]);
                continue;
            }
            ++ch;
            dfs(v, u);
            low[u] = min(low[u], low[v]);
            if(low[v] >= dfn[u]) {
                ap[u] = true;
                while(true) {
                    int eid = st.back();
                    st.pop_back();
                    bcc[eid] = ecnt;
                    if(eid == t) break;
                }
                ecnt++;
            }
        }
        if(ch == 1 && u == f) ap[u] = false;
    }
public:
    void init(int n_) {
        G.clear(); G.resize(n = n_);
        ecnt = 0; ap.assign(n, false);
        low.assign(n, 0); dfn.assign(n, 0);
    }
    void add_edge(int u, int v) {

```

```

    G[u].emplace_back(v, ecnt);
    G[v].emplace_back(u, ecnt++);
}
void solve() {
    ins.assign(ecnt, false);
    bcc.resize(ecnt); ecnt = 0;
    for (int i = 0; i < n; ++i)
        if (!dfn[i]) dfs(i, i);
}
// The id of bcc of the x-th edge (0-indexed)
int get_id(int x) { return bcc[x]; }
// Number of bcc
int count() { return ecnt; }
bool is_ap(int x) { return ap[x]; }
}; // 0-indexed

```

3.3 Strongly Connected Component [ac4987]

```

namespace scc {
    vector<int> edge[maxn], redge[maxn];
    stack<int, vector<int>> st;
    bool vis[maxn];
    void dfs(int u) {
        vis[u] = true;
        for(int v : edge[u])
            if(!vis[v])
                dfs(v);
        st.push(u);
    }
    void dfs2(int u, vector<int> &pa) {
        for(int v : redge[u])
            if(!pa[v])
                pa[v] = pa[u], dfs2(v, pa);
    }
    void add_edge(int u, int v) {
        edge[u].push_back(v);
        redge[v].push_back(u);
    }
    // pa[i]: scc id of all nodes in topo order
    vector<int> solve(int n) {
        vector<int> pa(n + 1);
        for(int i = 1; i <= n; i++)
            if(!vis[i])
                dfs(i);
        int id = 1; // start from 1
        while(!st.empty()) {
            int u = st.top();
            st.pop();
            if(!pa[u])
                pa[u] = id++, dfs2(u, pa);
        }
        return pa;
    } // 1-based
};

```

3.4 Two SAT [0b5797]

```

// maxn >= 2 * n (n: number of variables)
// clauses: (x, y) = x V y, -x if neg, var are 1-based
// return empty is no solution
vector<bool> solve(int n, const vector<pair<int, int>>
    &clauses) {
    auto id = [&](int x) { return abs(x) + n * (x < 0); };
    for(const auto &a, b : clauses) {
        scc::add_edge(id(-a), id(b));
        scc::add_edge(id(-b), id(a));
    }
    auto pa = scc::solve(n * 2);
    vector<bool> ans(n + 1);
    for(int i = 1; i <= n; i++) {
        if(pa[i] == pa[i + n])
            return vector<bool>();
        ans[i] = pa[i] > pa[i + n];
    }
    return ans;
}

```

3.5 Virtual Tree [ad5cf5]

```

// dfn: the dfs order, vs: important points, r: root
vector<pair<int, int>> build(vector<int> vs, int r) {
    vector<pair<int, int>> res;

```

```

    sort(vs.begin(), vs.end(), [](int i, int j) {
        return dfn[i] < dfn[j]; });
    vector<int> s = {r};
    for (int v : vs) if (v != r) {
        if (int o = lca(v, s.back()); o != s.back()) {
            while (s.size() >= 2) {
                if (dfn[s[s.size() - 2]] < dfn[o]) break;
                res.emplace_back(s[s.size() - 2], s.back());
                s.pop_back();
            }
            if (s.back() != o) {
                res.emplace_back(o, s.back());
                s.back() = o;
            }
        }
        s.push_back(v);
    }
    for (size_t i = 1; i < s.size(); ++i)
        res.emplace_back(s[i - 1], s[i]);
    return res; // (x, y): x->y
} // The returned virtual tree contains r (root).

```

3.6 Dominator Tree [af4aab]

```

/* Find dominator tree with root s in O(n)
 * Return the father of each node, **-2 for unreachable
 ** */
struct dominator_tree { // 0-based
    int tk;
    vector<vector<int>> g, r, rdom;
    vector<int> dfn, rev, fa, sdom, dom, val, rp;
    dominator_tree(int n): tk(0), g(n), r(n), rdom(n),
        dfn(n, -1), rev(n, -1), fa(n, -1), sdom(n, -1),
        dom(n, -1), val(n, -1), rp(n, -1) {}
    void add_edge(int x, int y) { g[x].push_back(y); }
    void dfs(int x) {
        rev[dfn[x] = tk] = x;
        fa[tk] = sdom[tk] = val[tk] = tk;
        tk++;
        for (int u : g[x]) {
            if (dfn[u] == -1) dfs(u), rp[dfn[u]] = dfn[x];
            r[dfn[u]].push_back(dfn[x]);
        }
    }
    void merge(int x, int y) { fa[x] = y; }
    int find(int x, int c = 0) {
        if (fa[x] == x) return c ? -1 : x;
        if (int p = find(fa[x], 1); p != -1) {
            if (sdom[val[x]] > sdom[val[fa[x]]])
                val[x] = val[fa[x]];
            fa[x] = p;
            return c ? p : val[x];
        } else {
            return c ? fa[x] : val[x];
        }
    }
}
vector<int> build(int s, int n) {
    dfs(s);
    for (int i = tk - 1; i >= 0; --i) {
        for (int u : r[i])
            sdom[i] = min(sdom[i], sdom[find(u)]);
        if (i) rdom[sdom[i]].push_back(i);
        for (int u : rdom[i]) {
            int p = find(u);
            dom[u] = (sdom[p] == i ? i : p);
        }
        if (i) merge(i, rp[i]);
    }
    vector<int> p(n, -2);
    p[s] = -1;
    for (int i = 1; i < tk; ++i)
        if (sdom[i] != dom[i]) dom[i] = dom[dom[i]];
    for (int i = 1; i < tk; ++i)
        p[rev[i]] = rev[dom[i]];
    return p;
}
};

```

3.7 Dinic [a007ae]

```

// Return max flor from s to t. INF, LINF and maxn
// required
template<typename T> // maxn: edge/node counts
struct dinic { // T: int or LL, up to range of flow

```

```

const T IN_INF = (is_same_v<T, int>) ? INF : LINF;
struct E {
    int v; T c; int r;
    E(int v, T c, int r):
        v(v), c(c), r(r){}
};
vector<E> adj[maxn];
pair<int, int> is[maxn]; // counts of edges
void add_edge(int u, int v, T c, int i = 0) {
    is[i] = {u, adj[u].size()};
    adj[u].push_back(E(v, c, (int) adj[v].size()));
    adj[v].push_back(E(u, 0, (int) adj[u].size() - 1));
}
int n, s, t;
void init(int nn, int ss, int tt) {
    n = nn, s = ss, t = tt;
    for (int i = 0; i <= n; ++i)
        adj[i].clear();
}
int le[maxn], it[maxn];
int bfs() {
    fill(le, le + maxn, -1); le[s] = 0;
    queue<int> q; q.push(s);
    while (!q.empty()) {
        int u = q.front(); q.pop();
        for (auto [v, c, r] : adj[u]) {
            if (c > 0 && le[v] == -1)
                le[v] = le[u] + 1, q.push(v);
        }
    }
    return ~le[t];
}
T dfs(int u, T f) {
    if (u == t) return f;
    for (int &i = it[u]; i < (int) adj[u].size(); ++i)
    {
        auto &[v, c, r] = adj[u][i];
        if (c > 0 && le[v] == le[u] + 1) {
            T d = dfs(v, min(c, f));
            if (d > 0) {
                c -= d;
                adj[v][r].c += d;
                return d;
            }
        }
    }
    return 0;
}
T flow() {
    T ans = 0, d;
    while (bfs()) {
        fill(it, it + maxn, 0);
        while ((d = dfs(s, IN_INF)) > 0) ans += d;
    }
    return ans;
}
T rest(int i) {
    return adj[is[i].first][is[i].second].c;
}
};

```

3.8 Min Cost Max Flow [56c624]

```

struct cost_flow { // maxn: node count
    static const int64_t INF = 102938475610293847LL;
    struct Edge {
        int v, r;
        int64_t f, c;
        Edge(int a, int b, int _c, int d): v(a), r(b), f(_c), c(d)
        {}
    };
    int n, s, t, prv[maxn], prvL[maxn], inq[maxn];
    int64_t dis[maxn], fl, cost;
    vector<Edge> E[maxn];
    void init(int _n, int _s, int _t) {
        n = _n; s = _s; t = _t;
        for (int i = 0; i < n; i++) E[i].clear();
        fl = cost = 0;
    }
    void add_edge(int u, int v, int64_t f, int64_t c) {
        E[u].push_back(Edge(v, E[v].size(), f, c));
        E[v].push_back(Edge(u, E[u].size()-1, 0, -c));
    }
};

```

```

pair<int64_t, int64_t> flow() {
    while (true) {
        for (int i = 0; i < n; i++) {
            dis[i] = INF;
            inq[i] = 0;
        }
        dis[s] = 0;
        queue<int> que;
        que.push(s);
        while (!que.empty()) {
            int u = que.front(); que.pop();
            inq[u] = 0;
            for (int i = 0; i < E[u].size(); i++) {
                int v = E[u][i].v;
                int64_t w = E[u][i].c;
                if (E[u][i].f > 0 && dis[v] > dis[u] + w) {
                    prv[v] = u; prvL[v] = i;
                    dis[v] = dis[u] + w;
                    if (!inq[v]) {
                        inq[v] = 1;
                        que.push(v);
                    }
                }
            }
        }
        if (dis[t] == INF) break;
        int64_t tf = INF;
        for (int v = t, u, l; v != s; v = u) {
            u = prv[v]; l = prvL[v];
            tf = min(tf, E[u][l].f);
        }
        for (int v = t, u, l; v != s; v = u) {
            u = prv[v]; l = prvL[v];
            E[u][l].f -= tf;
            E[v][E[u][l].r].f += tf;
        }
        cost += tf * dis[t];
        fl += tf;
    }
    return {fl, cost};
};

```

3.9 Stoer Wagner Algorithm [85ecf3]

```

// return global min cut in  $O(n^3)$ 
struct SW { // 1-based
    int edge[maxn][maxn], wei[maxn], n;
    bool vis[maxn], del[maxn];
    void init(int _n) {
        n = _n; MEM(edge, 0); MEM(del, 0);
    }
    void add_edge(int u, int v, int w) {
        edge[u][v] += w; edge[v][u] += w;
    }
    void search(int &s, int &t) {
        MEM(wei, 0); MEM(vis, 0);
        s = t = -1;
        while (true) {
            int mx = -1;
            for (int i = 1; i <= n; i++) {
                if (del[i] || vis[i]) continue;
                if (mx == -1 || wei[mx] < wei[i])
                    mx = i;
            }
            if (mx == -1) break;
            vis[mx] = true;
            s = t; t = mx;
            for (int i = 1; i <= n; i++)
                if (!vis[i] && !del[i])
                    wei[i] += edge[mx][i];
        }
    }
    int solve() {
        int ret = INF;
        for (int i = 1; i < n; i++) {
            int x, y;
            search(x, y);
            ret = min(ret, wei[y]);
            del[y] = true;
            for (int j = 1; j <= n; j++) {
                edge[x][j] += edge[y][j];
                edge[j][x] += edge[y][j];
            }
        }
    }
};

```



```

    }
}
return ret;
}
} sw;

```

3.10 General Matching [5c9691]

```

// Find max matching on general graph in  $O(|V|^3)$ 
vector<int> max_matching(vector<vector<int>> g) {
    int n = g.size();
    vector<int> match(n + 1, n), pre(n + 1, n), que;
    vector<int> s(n + 1), mark(n + 1), pa(n + 1);
    function<int(int)> fnd = [&](int x) {
        if(x == pa[x]) return x;
        return pa[x] = fnd(pa[x]);
    };
    auto lca = [&](int x, int y) {
        static int tk = 0;
        tk++;
        x = fnd(x);
        y = fnd(y);
        for(;; swap(x, y))
            if(x != n) {
                if(mark[x] == tk)
                    return x;
                mark[x] = tk;
                x = fnd(pre[match[x]]);
            }
    };
    auto blossom = [&](int x, int y, int l) {
        while(fnd(x) != l) {
            pre[x] = y;
            y = match[x];
            if(s[y] == 1)
                que.push_back(y), s[y] = 0;
            if(pa[x] == x) pa[x] = l;
            if(pa[y] == y) pa[y] = l;
            x = pre[y];
        }
    };
    auto bfs = [&](int r) {
        fill(s.begin(), s.end(), -1);
        iota(pa.begin(), pa.end(), 0);
        que = {r}; s[r] = 0;
        for(int it = 0; it < que.size(); it++) {
            int x = que[it];
            for(int u : g[x]) {
                if(s[u] == -1) {
                    pre[u] = x;
                    s[u] = 1;
                    if(match[u] == n) {
                        for(int a = u, b = x, lst;
                            b != n; a = lst, b = pre[a]) {
                            lst = match[b];
                            match[b] = a;
                            match[a] = b;
                        }
                        return;
                    }
                    que.push_back(match[u]);
                    s[match[u]] = 0;
                }
                else if(s[u] == 0 && fnd(u) != fnd(x)) {
                    int l = lca(u, x);
                    blossom(x, u, l);
                    blossom(u, x, l);
                }
            }
        }
    };
    for(int i = 0; i < n; i++)
        if(match[i] == n) bfs(i);
    match.resize(n);
    for(int i = 0; i < n; i++)
        if(match[i] == n) match[i] = -1;
    return match;
} // 0-based

```

3.11 Hopcroft Karp Algorithm [01aa79]

```

// Find maximum bipartite matching in  $O(E\sqrt{V})$ 
// g: edges for all nodes at left side

```

```

vector<int> hopcroft_karp(vector<vector<int>> g, int l,
    int r) {
    vector<int> match_l(l, -1), match_r(r, -1);
    vector<int> dis(l);
    vector<bool> vis(l);
    while(true) {
        queue<int> que;
        for(int i = 0; i < l; i++) {
            if(match_l[i] == -1)
                dis[i] = 0, que.push(i);
            else
                dis[i] = -1;
            vis[i] = false;
        }
        while(!que.empty()) {
            int x = que.front();
            que.pop();
            for(int y : g[x])
                if(match_r[y] != -1 && dis[match_r[y]] == -1) {
                    dis[match_r[y]] = dis[x] + 1;
                    que.push(match_r[y]);
                }
        }
        auto dfs = [&](auto dfs, int x) {
            vis[x] = true;
            for(int y : g[x]) {
                if(match_r[y] == -1) {
                    match_l[x] = y;
                    match_r[y] = x;
                    return true;
                }
                else if(dis[match_r[y]] == dis[x] + 1
                    && !vis[match_r[y]]
                    && dfs(dfs, match_r[y])) {
                    match_l[x] = y;
                    match_r[y] = x;
                    return true;
                }
            }
            return false;
        };
        bool ok = true;
        for(int i = 0; i < l; i++)
            if(match_l[i] == -1 && !dfs(dfs, i))
                ok = false;
        if(ok)
            break;
    }
    return match_l;
} // 0-based

```

3.12 Directed MST [f61898]

```

// Find minimum directed minimum spanning tree in  $O(E \log V)$ 
// DSU rollback is required
// Return parent of all nodes, -1 for unreachable ones and root
struct dmst_edge { int a, b; ll w; };
struct dmst_node { // Lazy skew heap node
    dmst_edge key;
    dmst_node *l, *r;
    ll delta;
    void prop() {
        key.w += delta;
        if (l) l->delta += delta;
        if (r) r->delta += delta;
        delta = 0;
    }
    dmst_edge top() { prop(); return key; };
};
dmst_node *dmst_merge(dmst_node *a, dmst_node *b) {
    if (!a || !b) return a ? b;
    a->prop();
    b->prop();
    if (a->key.w > b->key.w) swap(a, b);
    swap(a->l, (a->r = dmst_merge(b, a->r)));
    return a;
}
void dmst_pop(dmst_node*& a) {
    a->prop();
    a = dmst_merge(a->l, a->r);
}

```

```

pair<ll, vector<int>> dmst(int n, int r, const vector<
    dmst_edge>& g) {
    dsu_uf uf(n);
    vector<dmst_node*> heap(n);
    vector<dmst_node*> tmp;
    for (dmst_edge e : g) {
        tmp.push_back(new dmst_node {e});
        heap[e.b] = dmst_merge(heap[e.b], tmp.back());
    }
    ll res = 0;
    vector<int> seen(n, -1), path(n), par(n);
    seen[r] = r;
    vector<dmst_edge> Q(n), in(n, {-1, -1}), comp;
    deque<tuple<int, int, vector<dmst_edge>>> cys;
    for (int s = 0; s < n; s++) {
        int u = s, qi = 0, w;
        while (seen[u] < 0) {
            if (!heap[u]) return {-1, {}};
            dmst_edge e = heap[u]->top();
            heap[u]->delta -= e.w;
            dmst_pop(heap[u]);
            Q[qi] = e;
            path[qi++] = u;
            seen[u] = s;
            res += e.w;
            u = uf.find(e.a);
            if (seen[u] == s) { // found cycle, contract
                dmst_node* cyc = 0;
                int end = qi, time = uf.time();
                do {
                    cyc = dmst_merge(cyc, heap[w = path[--qi]]);
                } while (uf.join(u, w));
                u = uf.find(u);
                heap[u] = cyc;
                seen[u] = -1;
                cys.push_front({u, time, {&Q[qi], &Q[end]}});
            }
        }
        for (int i = 0; i < qi; i++)
            in[uf.find(Q[i].b)] = Q[i];
    }

    for (auto& [u, t, comp] : cys) { // restore sol (
        optional)
        uf.rollback(t);
        dmst_edge indmst_edge = in[u];
        for (auto& e : comp) in[uf.find(e.b)] = e;
        in[uf.find(indmst_edge.b)] = indmst_edge;
    }
    for (int i = 0; i < n; i++)
        par[i] = in[i].a;
    for (auto &a : tmp)
        delete a;
    return {res, par};
}

```

3.13 Edge Coloring [6d0f61]

/ Find a edge coloring using at most d+1 colors, where d is the max deg, in $O(V^3)$*
** mat[i][j] is the color between i, j in 1-based (0 for no edge)*
** use recolor() to add edge. Calculation is done in every recolor */*

```

struct edge_coloring { // 0-based
    int n;
    int mat[maxn][maxn];
    bool vis[maxn], col[maxn];
    void init(int _n) { n = _n; } // remember to init
    int check_conflict(int x, int loc) {
        for (int i = 0; i < n; i++)
            if (mat[x][i] == loc)
                return i;
        return n;
    }
    int get_block(int x) {
        memset(col, 0, sizeof col);
        for (int i = 0; i < n; i++) col[mat[x][i]] = 1;
        for (int i = 1; i < n; i++) if (!col[i]) return i;
        return n;
    }
    void recolor(int x, int y) {
        int pre_mat = get_block(y);

```

```

        int conflict = check_conflict(x, pre_mat);
        memset(vis, 0, sizeof vis);
        vis[y] = 1;
        vector<pair<int, int>> mat_line;
        mat_line.push_back({y, pre_mat});
        while (conflict != n && !vis[conflict]) {
            vis[conflict] = 1;
            y = conflict;
            pre_mat = get_block(y);
            mat_line.push_back({y, pre_mat});
            conflict = check_conflict(x, pre_mat);
        }
        if (conflict == n) {
            for (auto t : mat_line) {
                mat[x][t.first] = t.second;
                mat[t.first][x] = t.second;
            }
        }
        else {
            int pre_mat_x = get_block(x);
            int conflict_x = check_conflict(conflict, pre_mat_x);
            mat[x][conflict] = pre_mat_x;
            mat[conflict][x] = pre_mat_x;
            while (conflict_x != n) {
                int tmp = check_conflict(conflict_x, pre_mat);
                mat[conflict][conflict_x] = pre_mat;
                mat[conflict_x][conflict] = pre_mat;
                conflict = conflict_x;
                conflict_x = tmp;
                swap(pre_mat_x, pre_mat);
            }
            recolor(x, mat_line[0].first);
        }
    }
} mg;

```

4 Geometry

4.1 Basic [e546a6]

```

template<typename T>
struct point {
    T x, y;
    point(): x(0), y(0) { }
    point(T a, T b): x(a), y(b) { }
    template<typename V>
    explicit point(point<V> p): x(p.x), y(p.y) { }
    point operator-(const point &b) const {
        return point(x - b.x, y - b.y);
    }
    point operator+(const point &b) const {
        return point(x + b.x, y + b.y);
    }
    point<ld> operator*(ld r) const {
        return point<ld>(x * r, y * r);
    }
    point<ld> operator/(ld r) const {
        return point<ld>(x / r, y / r);
    }
    point operator-() const { return point(-x, -y); }
    bool operator<(const point &b) const {
        return x == b.x ? y < b.y : x < b.x;
    }
    T dis2() const { return x * x + y * y; }
    ld dis() const { return sqrt(dis2()); }
    point perp() const { return point(-y, x); }
    point norm() const {
        ld d = dis();
        return *this / d;
    }
    point rot(double o) const {
        double c = cos(o), s = sin(o);
        return point(c * a.x - s * a.y, s * a.x + c * a.y);
    }
};
using ptld = point<ld>;
using ptll = point<ll>;
template<typename T>
T cross(const point<T> &a, const point<T> &b, const
    point<T> &c) {
    auto x = b - a, y = c - a;
    return x.x * y.y - y.x * x.y;
}

```



```

template<typename T>
T cross2(const point<T> &x, const point<T> &y) {
    return x.x * y.y - y.x * x.y;
}
template<typename T>
T dot(const point<T> &a, const point<T> &b, const point<T> &c) {
    auto x = b - a, y = c - a;
    return x.x * y.x + x.y * y.y;
}
template<typename T>
ld area(const point<T> &a, const point<T> &b, const point<T> &c) {
    return ld(cross(a, b, c)) / 2;
}
int sgn(ld v) {
    if (abs(v) < EPS)
        return 0;
    return v > 0 ? 1 : -1;
}
int sgn(ll v) { return (v > 0 ? 1 : (v < 0 ? -1 : 0)); }
template<typename T>
int ori(point<T> a, point<T> b, point<T> c) {
    return sgn(cross(a, b, c));
}
template<typename T>
bool collinearity(point<T> a, point<T> b, point<T> c) {
    return ori(a, b, c) == 0;
}
template<typename T>
bool btw(point<T> p, point<T> a, point<T> b) {
    return collinearity(p, a, b) && sgn(dot(p, a, b)) <= 0;
}
template<typename T>
point<ld> projection(point<T> p1, point<T> p2, point<T> p3) {
    return (p2 - p1) * dot(p1, p2, p3) / (p2 - p1).dis2();
}
template<typename T>
int quad(point<T> a) {
    if (a.x == 0 && a.y == 0) // change this for ld
        return -1;
    if (a.x > 0)
        return a.y > 0 || a.y == 0 ? 0 : 3;
    if (a.x < 0)
        return a.y > 0 ? 1 : 2;
    return a.y > 0 ? 1 : 3;
}
template<typename T>
bool cmp_by_polar(const point<T> &a, const point<T> &b) {
    // start from positive x-axis
    // Undefined if a or b is the origin
    if (quad(a) != quad(b))
        return quad(a) < quad(b);
    if (ori(point<T>(), a, b) == 0)
        return a.dis2() < b.dis2();
    return ori(point<T>(), a, b) > 0;
}
int arg_quad(ptll p) {
    return (p.y == 0) // use sgn for ptld
        ? (p.x < 0 ? 3 : 1) : (p.y < 0 ? 0 : 2);
}
template<typename T>
int arg_cmp(point<T> a, point<T> b) {
    // returns 0/+1, starts from theta = -PI
    int qa = arg_quad(a), qb = arg_quad(b);
    if (qa != qb) return sgn(ll(qa - qb));
    return sgn(cross2(b, a));
}
using Line = pair<ptll, ptll>;
bool seg_intersect(Line a, Line b) {
    auto [p1, p2] = a;
    auto [p3, p4] = b;
    tie(p1, p2) = a;
    tie(p3, p4) = b;
    if (btw(p1, p3, p4) || btw(p2, p3, p4) || btw(p3, p1, p2) || btw(p4, p1, p2))
        return true;
}

```

```

return ori(p1, p2, p3) * ori(p1, p2, p4) < 0 &&
    ori(p3, p4, p1) * ori(p3, p4, p2) < 0;
}
ptld intersect(Line a, Line b) {
    ptll p1, p2, p3, p4;
    tie(p1, p2) = a;
    tie(p3, p4) = b;
    ld a123 = cross(p1, p2, p3);
    ld a124 = cross(p1, p2, p4);
    return (p4 * a123 - p3 * a124) / (a123 - a124);
}

```

4.2 2D Convex Hull [5346ef]

```

// returns a convex hull in counterclockwise order
// for a non-strict one, change cross >= to >
// Be careful of n <= 2
vector<point> convex_hull(vector<point> p) {
    sort(p.begin(), p.end());
    if (p[0] == p.back()) return { p[0] };
    int s = 1, t = 0;
    vector<point> h(p.size() + 1);
    for (int _ = 2; _--; s = t--, reverse(p.begin(), p.end()))
        for (point i : p) {
            while (t > s && ori(i, h[t - 1], h[t - 2]) >= 0)
                t--;
            h[t++] = i;
        }
    return h.resize(t), h;
}

```

4.3 Farthest Pair [9117eb]

```

// p is CCW convex hull w/o colinear points
void farthest_pair(vector<point> p) {
    int n = p.size(), pos = 1; ll ans = 0;
    for (int i = 0; i < n; i++) {
        P e = p[(i + 1) % n] - p[i];
        while (cross(e, p[(pos + 1) % n] - p[i]) >
            cross(e, p[pos] - p[i]))
            pos = (pos + 1) % n;
        for (int j: {i, (i + 1) % n})
            ans = max(ans, norm(p[pos] - p[j]));
    } // tested @ AOJ CGL_4_B
}

```

4.4 Minkowski Sum [d97584]

```

// If we want to calculate the minkowski sum of vectors
// sort <v_i, -v_i, v_{i+1}, -v_{i+1}, ...> by polar angle order
// The prefix sum of vectors is a convex polygon and is the minkowski sum
// To get the new origin, compare the max (x, y) of the convex and the sum of positive (x, y) of the vectors

```

```

// A, B are convex hull rotated to min by (X, Y)
// i.e. rotate(A.begin(), min_element(all(A)), A.end())
vector<point> Minkowski(vector<point> A, vector<point> B) {
    vector<point> C(1, A[0] + B[0]), s1, s2;
    const int N = (int) A.size(), M = (int) B.size();
    for (int i = 0; i < N; ++i)
        s1.push_back(A[(i + 1) % N] - A[i]);
    for (int i = 0; i < M; ++i)
        s2.push_back(B[(i + 1) % M] - B[i]);
    for (int i = 0, j = 0; i < N || j < M; )
        if (j >= N || (i < M && cross(s1[i], s2[j]) >= 0))
            C.push_back(C.back() + s1[i++]);
        else
            C.push_back(C.back() + s2[j++]);
    return convex_hull(C);
}

```

4.5 Circle [11ff47]

```

struct Circle {
    point c;
    double r;
};
// Calculate intersection between given circle and Line

```

```

vector<point> inter_circle_line(Circle cir, Line l) {
    const auto &[c, r] = cir;
    const auto &[a, b] = l;
    point p = a + (b - a) * dot(a, b, c) / (b - a).dis2();
    ;
    double s = cross(a, b, c), h2 = r * r - s * s / (b - a).dis2();
    if (h2 < 0) return {};
    if (h2 == 0) return {p};
    point h = (b - a) / (b - a).dis() * sqrt(h2);
    return {p - h, p + h};
} // no tested

// return p4 is strictly in circumcircle of tri(p1,p2,p3)
inline ll sqr(ll x) { return x * x; }
bool in_cc(const point& p1, const point& p2, const point& p3, const point& p4) {
    ll u11 = p1.x - p4.x; ll u12 = p1.y - p4.y;
    ll u21 = p2.x - p4.x; ll u22 = p2.y - p4.y;
    ll u31 = p3.x - p4.x; ll u32 = p3.y - p4.y;
    ll u13 = sqr(p1.x) - sqr(p4.x) + sqr(p1.y) - sqr(p4.y);
    ll u23 = sqr(p2.x) - sqr(p4.x) + sqr(p2.y) - sqr(p4.y);
    ll u33 = sqr(p3.x) - sqr(p4.x) + sqr(p3.y) - sqr(p4.y);
    __int128 det = (__int128)-u13 * u22 * u31 + (__int128)u12 * u23 * u31 + (__int128)u13 * u21 * u32 - (__int128)u11 * u23 * u32 - (__int128)u12 * u21 * u33 + (__int128)u11 * u22 * u33;
    return det > EPS;
} // not tested

// Return the area of intersection of poly and circle
double _area(point pa, point pb, double r) {
    if (pa.dis2() < pb.dis2()) swap(pa, pb);
    if (pb.dis() < EPS) return 0;
    double S, h, theta;
    double a = pb.dis(), b = pa.dis(), c = (pb - pa).dis();
    double cosB = dot2(pb, pb - pa) / a / c, B = acos(cosB);
    double cosC = dot2(pa, pb) / a / b, C = acos(cosC);
    if (a > r) {
        S = (C / 2) * r * r;
        h = a * b * sin(C) / c;
        if (h < r && B < PI / 2) S -= (acos(h / r) * r * r - h * sqrt(r * r - h * h));
    }
    else if (b > r) {
        theta = PI - B - asin(sin(B) / r * a);
        S = 0.5 * a * r * sin(theta) + (C - theta) / 2 * r * r;
    }
    else S = 0.5 * sin(C) * a * b;
    return S;
}
double area_poly_circle(const vector<point> poly, const Circle c) {
    const auto &[O, r] = c;
    double S = 0;
    for (int i = 0; i < poly.size(); ++i)
        S += _area(poly[i] - O, poly[(i + 1) % poly.size() - O, r] * ori(O, poly[i], poly[(i + 1) % poly.size()]);
    return abs(S);
} // not tested

// Return intersection of two circles in p1 and p2
bool CCinter(Circle &a, Circle &b, point &p1, point &p2) {
    point o1 = a.O, o2 = b.O;
    double r1 = a.r, r2 = b.r, d2 = (o1 - o2).dis2(), d = sqrt(d2);
    if (d < max(r1, r2) - min(r1, r2) || d > r1 + r2) return 0;
    point u = (o1 + o2) * 0.5 + (o1 - o2) * ((r2 * r2 - r1 * r1) / (2 * d2));

```

```

    double A = sqrt((r1 + r2 + d) * (r1 - r2 + d) * (r1 + r2 - d) * (-r1 + r2 + d));
    point v = point(o1.Y - o2.Y, -o1.X + o2.X) * A / (2 * d2);
    p1 = u + v, p2 = u - v;
    return 1;
} // not tested

```

4.6 Tangent Lines of Circle and Point [003418]

```

vector<Line> tangent(Circle c, point p) {
    vector<Line> z;
    double d = (p - c.c).dis();
    if (sign(d - c.r) == 0) {
        point i = (p - c.c).rot(PI / 2);
        z.push_back({p, p + i});
    } else if (d > c.r) {
        double o = acos(c.r / d);
        point i = (p - c.c).norm(), j = i.rot(o) * c.r, k = i.rot(-o) * c.r;
        z.push_back({c.c + j, p});
        z.push_back({c.c + k, p});
    }
    return z;
} // not tested

```

4.7 Tangent Lines of Cricles [d4f0a6]

```

vector <Line> tangent(Circle c1, Circle c2, int sign1) {
    // sign1 = 1 for outer tang, -1 for inter tang
    vector <Line> ret;
    double d_sq = abs2(c1.c - c2.c);
    if (sgn(d_sq) == 0) return ret;
    double d = sqrt(d_sq);
    point v = (c2.c - c1.c) / d;
    double c = (c1.r - sign1 * c2.r) / d;
    if (c * c > 1) return ret;
    double h = sqrt(max(0.0, 1.0 - c * c));
    for (int sign2 = 1; sign2 >= -1; sign2 -= 2) {
        point n = point(v.x * c - sign2 * h * v.y, v.y * c + sign2 * h * v.x);
        point p1 = c1.c + n * c1.r;
        point p2 = c2.c + n * (c2.r * sign1);
        if (sign(p1.x - p2.x) == 0 && sign(p1.y - p2.y) == 0)
            p2 = p1 + (c2.c - c1.c).perp();
        ret.push_back({p1, p2});
    }
    return ret;
} // not tested

```

4.8 Delaunay Triangular [f3d422]

```

/* please ensure input points are unique */
/* A triangulation such that no points will strictly inside circumcircle of any triangle.
find(root, p) : return a triangle contain given point
add_point : add a point into triangulation
Region of triangle u: iterate each u.e[i].tri, each points are u.p[(i+1)%3], u.p[(i+2)%3]
Voronoi diagram: for each triangle in `res`, the bisector of all its edges will split the region. */
#define L(i) ((i) == 0 ? 2 : (i) - 1)
#define R(i) ((i) == 2 ? 0 : (i) + 1)
#define F3 for (int i = 0; i < 3; i++)
bool in_cc(const array<ptll, 3> &p, ptll q) {
    __int128 det = 0;
    F3 det += __int128(p[i].dis2() - q.dis2()) * cross2(p[L(i)] - q, p[R(i)] - q);
    return det > 0;
}
struct Tri;
struct E {
    Tri *t; int side; E() : t(0), side(0) {}
    E(Tri *t_, int side_) : t(t_), side(side_) {}
};
struct Tri {
    bool vis;
    array<ptll, 3> p;
    array<Tri*, 3> ch;
    array<E, 3> e;

```

```

Tri(ptll a = ptll(), ptll b = ptll(), ptll c = ptll()
) : vis(0), p{a,b,c}, ch{} {}
bool has_chd() const { return ch[0] != nullptr; }
bool contains(ptll q) const {
    F3 if (ori(p[i], p[R(i)], q) < 0) return false;
    return true;
}
} pool[maxn * 10], *it;
void link(E a, E b) {
    if (a.t) a.t->e[a.side] = b;
    if (b.t) b.t->e[b.side] = a;
}
const int C = 100 * 1007 * 1007;
struct Trigs {
    Tri *root;
    Trigs() { // should at least contain all points
        root = // C = 100*MAXC^2 or just MAXC?
            new(it++) Tri(ptll(-C, -C), ptll(C * 2, -C), ptll
                (-C, C * 2));
    }
    void add_point(ptll p) { add_point(find(p, root), p); }
    static Tri* find(ptll p, Tri *r) {
        while (r->has_chd()) for (Tri *c: r->ch)
            if (c && c->contains(p)) { r = c; break; }
        return r;
    }
    void add_point(Tri *r, ptll p) {
        array<Tri*, 3> t; /* split into 3 triangles */
        F3 t[i] = new (it++) Tri(r->p[i], r->p[R(i)], p);
        F3 link(E(t[i], 0), E(t[R(i)], 1));
        F3 link(E(t[i], 2), r->e[L(i)]);
        r->ch = t;
        F3 flip(t[i], 2);
    }
    void flip(Tri* A, int a) {
        auto [B, b] = A->e[a]; /* flip edge between A,B */
        if (!B || !in_cc(A->p, B->p[b])) return;
        Tri *X = new(it++) Tri(A->p[R(a)], B->p[b], A->p[a
        ]);
        Tri *Y = new(it++) Tri(B->p[R(b)], A->p[a], B->p[b
        ]);
        link(E(X, 0), E(Y, 0));
        link(E(X, 1), A->e[L(a)]);
        link(E(X, 2), B->e[R(b)]);
        link(E(Y, 1), B->e[L(b)]);
        link(E(Y, 2), A->e[R(a)]);
        A->ch = B->ch = {X, Y, nullptr};
        flip(X, 1); flip(X, 2); flip(Y, 1); flip(Y, 2);
    }
};
vector<Tri*> res;
void go(Tri *now) { // store all tri into res
    if (now->vis) return;
    now->vis = true;
    if (!now->has_chd()) res.push_back(now);
    for (Tri *c : now->ch) if (c) go(c);
}
vector<Directed_Line> frame;
vector<vector<ptld>> build_voronoi_cells(const vector<
    ptll> &p, const vector<Tri*> &res); // Only need
    for voronoi
// !!! The order is shuffled !!!
vector<vector<ptld>> build(vector<ptll> &ps) {
    it = pool; res.clear();
    shuffle(ps.begin(), ps.end(), mt19937(487638763));
    Trigs tr; for (point p : ps) tr.add_point(p);
    go(tr.root); // use `res` afterwards
    return build_voronoi_cells(ps, res); // Only needed
        for voronoi
// res is the result otherwise
}

```

4.9 Half Plane Intersection [ced799]

```

// O(NlogN), undefined if the result has area INF (not
    enclosed)
struct Directed_Line {
    ptll st, ed, dir;
    Directed_Line(ptll s, ptll e) : st(s), ed(e), dir(e -
        s) {}
};
using LN = const Directed_Line &;

```

```

ptld intersect(LN A, LN B) {
    ld t = cross2(B.st - A.st, B.dir) / ld(cross2(A.dir,
        B.dir));
    return ptld(A.st) + A.dir * t; // C^3 / C^2
}
int sgn(__int128 v) { return (v > 0 ? 1 : (v < 0 ? -1 :
    0)); }
bool cov(LN l, LN A, LN B) {
    __int128 u = cross2(B.st - A.st, B.dir);
    __int128 v = cross2(A.dir, B.dir);
    // ori(L.st, L.ed, A.st + A.dir*(u/v)) <= 0?
    __int128 x = (A.dir).x * u + (A.st - l.st).x * v;
    __int128 y = (A.dir).y * u + (A.st - l.st).y * v;
    return sgn(x * (l.dir).y - y * (l.dir).x) * sgn(v) >=
        0;
} // x, y are C^3
bool operator<(LN a, LN b) {
    if (int c = arg_cmp(a.dir, b.dir)) return c == -1;
    return ori(a.st, a.ed, b.st) < 0;
}
// cross(pt-Line.st, line.dir)<=0 <-> pt in half plane
// the half plane is the LHS when going from st to ed
vector<ptld> HPI(vector<Directed_Line> &q) {
    sort(q.begin(), q.end());
    int n = (int)q.size(), l = 0, r = -1;
    for (int i = 0; i < n; i++) {
        if (i && !arg_cmp(q[i].dir, q[i - 1].dir)) continue
        ;
        while (1 < r && cov(q[i], q[r-1], q[r])) --r;
        while (1 < r && cov(q[i], q[l], q[l + 1])) ++l;
        q[++r] = q[i];
    }
    while (1 < r && cov(q[l], q[r-1], q[r])) --r;
    while (1 < r && cov(q[r], q[l], q[l + 1])) ++l;
    n = r - l + 1; // q[l .. r] are the lines
    if (n <= 1 || !arg_cmp(q[l].dir, q[r].dir)) return {
        };
    vector<ptld> pt(n);
    for (int i = 0; i < n; i++)
        pt[i] = intersect(q[i + 1], q[(i + 1) % n + 1]);
    return pt;
}

```

4.10 Point In Convex [324eed]

```

bool in_convex(const vector<point> &convex, point p,
    bool strict = true) {
    if (convex.empty())
        return false;
    int a = 1, b = convex.size() - 1, r = !strict;
    if (b < 2)
        return r && btw(p, convex[0], convex.back());
    if (ori(convex[0], convex[a], convex[b]) > 0) swap(a,
        b);
    if (ori(convex[0], convex[a], p) >= r || ori(convex
        [0], convex[b], p) <= -r)
        return false;
    while (abs(a - b) > 1) {
        int c = (a + b) / 2;
        (ori(convex[0], convex[c], p) > 0 ? b : a) = c;
    }
    return ori(convex[a], convex[b], p) < r;
} // no tested

```

4.11 Voronoi Diagram [519adb]

```

vector<Directed_Line> frame;
vector<vector<ptld>> build_voronoi_cells(const vector<
    ptll> &p, const vector<Tri*> &res) {
    // O(nlogn)
    vector<vector<int>> adj(p.size());
    map<ptll, int> mp;
    for (size_t i = 0; i < p.size(); ++i)
        mp[p[i]] = i;
    const auto Get = [&](ptll z) {
        auto it = mp.find(z);
        return it == mp.end() ? -1 : it->second;
    };
    for (Tri *t : res) F3 {
        ptll A = t->p[i], B = t->p[R(i)];
        int a = Get(A), b = Get(B);
        if (a == -1 || b == -1) continue;
        adj[a].emplace_back(b);
    }
}

```

```
// use `adj` and `p` and HPI to build cells
vector<vector<ptld>> owo;
for (size_t i = 0; i < p.size(); i++) {
    assert(!frame.empty());
    vector<Directed_Line> ls = frame; // the frame, a
    // rectangle closing all points
    // coordinates of frame should be doubled
    for (int j : adj[i]) {
        point m = p[i] + p[j], d = (p[j] - p[i]).perp();
        assert(d.dis2() != 0);
        ls.emplace_back(m, m + d); // doubled coordinate
    }
    // use HPI(ls) to get the convex hull closing point
    i
    owo.push_back(HPI(ls));
}
return owo;
}
```

5 String

5.1 KMP [647790]

```
vector<int> kmp(const string &s) {
    int n = s.size();
    vector<int> dp(n);
    for (int i = 1, j = 0; i < n; i++) {
        while (j && s[i] != s[j])
            j = dp[j - 1];
        if (s[i] == s[j])
            j++;
        dp[i] = j;
    }
    return dp;
}
```

5.2 Z Value [f762e6]

```
// Return Z value of string s in O(|S|)
// Note that z[0] = |S|
vector<int> Zalgo(const string &s) {
    vector<int> z(s.size(), (int) s.size());
    for (int i = 1, l = 0, r = 0; i < z[0]; ++i) {
        int j = clamp(r - i, 0, z[i - 1]);
        while (i + j < z[0] && s[i + j] == s[j])
            j++;
        if (i + (z[i] = j) > r)
            r = i + z[i];
    }
    return z;
}
```

5.3 Suffix Array [fb97cc]

```
int sa[maxn], tmp[2][maxn], c[maxn];
void get_sa(const string &s) { // m: char set
    int *x = tmp[0], *y = tmp[1], m = 256, n = s.size();
    for (int i = 0; i < m; i++) c[i] = 0;
    for (int i = 0; i < n; i++) c[x[i]] = s[i]++;
    for (int i = 1; i < m; i++) c[i] += c[i - 1];
    for (int i = n - 1; i >= 0; --i) sa[--c[x[i]]] = i;
    for (int k = 1; k < n; k <= 1) {
        for (int i = 0; i < m; i++) c[i] = 0;
        for (int i = 0; i < n; i++) c[x[i]]++;
        for (int i = 1; i < m; i++) c[i] += c[i - 1];
        int p = 0;
        for (int i = n - k; i < n; i++) y[p++] = i;
        for (int i = 0; i < n; i++)
            if (sa[i] >= k) y[p++] = sa[i] - k;
        for (int i = n - 1; i >= 0; --i) sa[--c[x[y[i]]]] = y[i];
        y[sa[0]] = p = 0;
        for (int i = 1; i < n; i++) {
            int a = sa[i], b = sa[i - 1];
            if (x[a] == x[b] && a + k < n && b + k < n && x[a + k] == x[b + k]) { }
            else p++;
            y[sa[i]] = p;
        }
        if (n == p + 1)
            break;
        swap(x, y);
        m = p + 1;
    }
}
```

```
} // sa[i]: index which ranks i
int rk[maxn], lcp[maxn]; // lcp[i] : lcp with i-1
void get_lcp(const string &s) {
    int n = s.size(), val = 0;
    for (int i = 0; i < n; i++) rk[sa[i]] = i;
    for (int i = 0; i < n; i++) {
        if (rk[i] == 0) lcp[rk[i]] = 0;
        else {
            if (val) val--;
            int p = sa[rk[i] - 1];
            while (val + i < n && val + p < n && s[val + i] == s[val + p])
                val++;
            lcp[rk[i]] = val;
        }
    }
}
```

5.4 AC Automaton [971e58]

// Remember to call init then compile

```
class AhoCorasick {
private:
    static constexpr int Z = 26;
    struct node {
        node *nxt[Z], *fail;
        vector<int> data;
        node(): fail(nullptr) {
            memset(nxt, 0, sizeof(nxt));
            data.clear();
        }
    } *rt;
    inline int Idx(char c) { return c - 'a'; }
public:
    void init() { rt = new node(); }
    void add(const string &s, int d) { // d is index, etc
        node* cur = rt;
        for (auto c : s) {
            if (!cur->nxt[Idx(c)])
                cur->nxt[Idx(c)] = new node();
            cur = cur->nxt[Idx(c)];
        }
        cur->data.push_back(d);
    }
    void compile() {
        vector<node*> bfs;
        size_t ptr = 0;
        for (int i = 0; i < Z; i++) {
            if (!rt->nxt[i]) {
                // uncomment 2 lines to make it DFA
                // rt->nxt[i] = rt;
                continue;
            }
            rt->nxt[i]->fail = rt;
            bfs.push_back(rt->nxt[i]);
        }
        while (ptr < bfs.size()) {
            node* u = bfs[ptr++];
            // More code here to record information...
            // rt is NOT in bfs
            for (int i = 0; i < Z; i++) {
                if (!u->nxt[i]) {
                    // u->nxt[i] = u->fail->nxt[i];
                    continue;
                }
                node* u_f = u->fail;
                while (u_f) {
                    if (!u_f->nxt[i]) {
                        u_f = u_f->fail;
                        continue;
                    }
                    u->nxt[i]->fail = u_f->nxt[i];
                    break;
                }
            }
            if (!u_f) u->nxt[i]->fail = rt;
            bfs.push_back(u->nxt[i]);
        }
    }
    void match(const string &s, vector<int> &ret) {
        node* u = rt;
        for (auto c : s) {
```

```

    while (u != rt && !u->nxt[Idx(c)])
        u = u->fail;
    u = u->nxt[Idx(c)];
    if (!u) u = rt;
    node* tmp = u;
    while (tmp != rt) {
        for (auto d : tmp->data)
            ret.push_back(d);
        tmp = tmp->fail;
    }
}
}
} ac;

```

5.5 Booth Algorithm [e7cb5d]

```

// return start index of minimum rotation in O(|s|)
int min_rotation(string s) {
    s += s;
    int k = 0;
    vector<int> f(s.size(), -1);
    for(int j = 1; j < s.size(); j++) {
        int i = f[j - k - 1];
        for(i = f[j - k - 1];
            i != -1 && s[j] != s[i + k + 1]; i = f[i])
            if(s[k + i + 1] > s[j])
                k = j - i - 1;
        if(i == -1 && s[j] != s[k + i + 1]) {
            if(s[j] < s[k + i + 1])
                k = j;
            f[j - k] = -1;
        }
        else
            f[j - k] = i + 1;
    }
    return k;
}

```

5.6 Manacher Algorithm [5cc8bc]

```

vector<int> manacher_algorithm(string s) {
    int n = 2 * s.size() + 1;
    string t(n, 0);
    vector<int> len(n); // len[i]: max length when mid at i
    for(int i = 0; i < n; i++) {
        if(i & 1)
            t[i] = s[i / 2];
    }
    for(int i = 0, l = 0, r = -1; i < n; i++) {
        len[i] = (i <= r ? min(len[2 * l - i], r - i) : 0);
        while(i - len[i] >= 0 && i + len[i] < n && t[i - len[i]] == t[i + len[i]])
            len[i]++;
        len[i]--;
        if(i + len[i] > r)
            l = i, r = i + len[i];
    }
    return len;
}

```

5.7 Palindromic Tree [0673ee]

```

struct PalindromicTree {
    struct node {
        // len: length of max palindromic at i
        // next[c]: next id if add c at front & back
        int nxt[26], f, len; // num = depth of fail link
        int cnt, num; // = #pal_suffix of this node
        node(int l = 0) : nxt{}, f(0), len(l), cnt(0), num(0) {}
    };
    vector<node> st; vector<char> s; int last, n;
    void init() {
        st.clear(); s.clear();
        last = 1; n = 0;
        st.push_back(0); st.push_back(-1);
        st[0].f = 1; s.push_back(-1);
    }
    int getFail(int x) {
        while (s[n - st[x].len - 1] != s[n]) x = st[x].f;
        return x;
    }
    void add(int c) {
        s.push_back(c - 'a'); ++n;
    }
}

```

```

int cur = getFail(last);
if (!st[cur].nxt[c]) {
    int now = st.size();
    st.push_back(st[cur].len + 2);
    st[now].f = st[getFail(st[cur].f)].nxt[c];
    st[cur].nxt[c] = now;
    st[now].num = st[st[now].f].num + 1;
}
last = st[cur].nxt[c]; ++st[last].cnt;
}
}
void dpcnt() { // cnt = #occurrence in whole str
    for (int i = st.size() - 1; i >= 0; i--)
        st[st[i].f].cnt += st[i].cnt;
}
int size() { return st.size() - 2; }
} pt; // not tested
/* usage
string s; cin >> s; pt.init();
for (int i = 0; i < SZ(s); i++) {
    int prvsz = pt.size(); pt.add(s[i]);
    if (prvsz != pt.size()) {
        int r = i, l = r - pt.st[pt.Last].len + 1;
        // pal @ [l,r]: s.substr(l, r-l+1)
    }
} */

```

6 Math

6.1 Lemma And Theory

6.1.1 Pick's Theorem

For a simple polygon, its area A can be written as $A = i + \frac{b}{2} - 1$ in which i is the number of points that are strictly interior to the polygon and b is the number of points that are on the polygon's boundary.

6.1.2 Euler's Planar Graph Theorem

F : number of regions bounded by edges.
 $V - E + F = C + 1, E \leq 3V - 6$

6.1.3 Modular inversion recurrence

For some prime p ,

$$inv_i = \begin{cases} 1 & i = 1 \\ p - \lfloor \frac{p}{i} \rfloor \times inv_{(p \bmod i)} & 1 < i < p \end{cases}$$

6.2 Numbers

6.2.1 Catalan number

Start from $n = 0 : 1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786, \dots$

$$C_n = \frac{1}{n+1} \binom{2n}{n} = \frac{(2n)!}{(n+1)!n!} = \prod_{k=2}^n \frac{n+k}{k}$$

$$C_n = \binom{2n}{n} - \binom{2n}{n+1}$$

Recurrence

$$C_0 = 1$$

$$C_{n+1} = \sum_{i=0}^n C_i C_{n-i}$$

$$C_{n+1} = \frac{2(2n+1)}{n+2} C_n$$

6.2.2 Primes

12721, 13331, 14341, 75577

999997771, 999991231, 1000000007, 1000000009, 1000696969

$$10^{12} + 39, 10^{15} + 37$$

6.3 Extgcd [d8844c]

```

// return (d, x, y) s.t. ax+by=d=gcd(a,b)
template<typename T>
tuple<T, T, T> extgcd(T a, T b) {
    if(!b) return make_tuple(a, 1, 0);
    auto [d, x, y] = extgcd(b, a % b);
    return make_tuple(d, y, x - (a / b) * y);
}

```


6.4 Chinese Remainder Theorem [69f820]

```
// x % m1 = x1, x % m2 = x2
ll chre(ll x1, ll m1, ll x2, ll m2){
    ll g = __gcd(m1, m2);
    if ((x2 - x1) % g) return -1; // no solution
    m1 /= g; m2 /= g;
    ll p = get<1>(extgcd(m1, m2));
    ll lcm = m1 * m2 * g;
    ll res = p * (x2 - x1) * m1 + x1;
    // might overflow for above two lines, be cautious
    return (res % lcm + lcm) % lcm;
}
```

6.5 Linear Sieve [59dc40]

```
int least_prime_divisor[maxn];
vector<int> pr;
void linear_sieve() {
    for(int i = 2; i < maxn; i++) {
        if(!least_prime_divisor[i]) {
            pr.push_back(i);
            least_prime_divisor[i] = i;
        }
        for(int p : pr) {
            if(1LL * i * p >= maxn) break;
            least_prime_divisor[i * p] = p;
            if(i % p == 0) break;
        }
    }
}
```

6.6 Fast Walsh Transform [239248]

```
/* do not move ta,tb, default for xor
 * remove last 2 lines for non-xor
 * or convolution:
 * x[i]=ta,x[j]=ta+tb; x[i]=ta,x[j]=tb-ta for inv
 * and convolution:
 * x[i]=ta+tb,x[j]=tb; x[i]=ta-tb,x[j]=tb for inv */
void fwt(int x[], int N, bool inv = false) {
    for(int d = 1; d < N; d <= 1) {
        for(int s = 0, d2 = d * 2; s < N; s += d2)
            for(int i = s, j = s + d; i < s + d; i++, j++) {
                int ta = x[i], tb = x[j];
                x[i] = modadd(ta, tb);
                x[j] = modsub(ta, tb);
            }
    }
    if(inv) for(int i = 0, invn = modinv(N); i < N; i++)
        x[i] = modmul(x[i], invn);
} // N: array len
```

6.7 Floor Sum [5e4ea5]

```
// @param n `n < 2^32`
// @param m `1 <= m < 2^32`
// @return sum_{i=0}^{n-1} floor((ai + b)/m) mod 2^64
ull floor_sum_unsigned(ull n, ull m, ull a, ull b) {
    ull ans = 0;
    while (true) {
        if (a >= m) {
            ans += n * (n - 1) / 2 * (a / m); a %= m;
        }
        if (b >= m) {
            ans += n * (b / m); b %= m;
        }
        ull y_max = a * n + b;
        if (y_max < m) break;
        // y_max < m * (n + 1)
        // floor(y_max / m) <= n
        n = (ull)(y_max / m), b = (ull)(y_max % m);
        swap(m, a);
    }
    return ans;
}
ll floor_sum(ll n, ll m, ll a, ll b) {
    ull ans = 0;
    if (a < 0) {
        ull a2 = (a % m + m) % m;
        ans -= 1ULL * n * (n - 1) / 2 * ((a2 - a) / m);
        a = a2;
    }
    if (b < 0) {
        ull b2 = (b % m + m) % m;

```

```
ans -= 1ULL * n * ((b2 - b) / m);
b = b2;
}
return ans + floor_sum_unsigned(n, m, a, b);
}
```

6.8 Linear Programming [e03738]

```
/* M constraints, i-th constraint is:
 \sum_{j=0}^{n-1} A[i][j] * x_j <= B[i]
 Let v = \sum_{j=0}^{n-1} C[j] * x_j
 maximize v satisfying constraints
 sol[i] = x_i
 remind the precision error */
struct Simplex { // 0-based
    using T = long double;
    static const int N = 410, M = 30010;
    const T eps = 1e-7;
    int n, m;
    int Left[M], Down[N];
    T a[M][N], b[M], c[N], v, sol[N];
    bool eq (T a, T b) { return fabs(a - b) < eps; }
    bool ls (T a, T b) { return a < b && !eq(a, b); }
    void init(int _n, int _m) {
        n = _n, m = _m, v = 0;
        for (int i = 0; i < m; ++i) for (int j = 0; j < n; ++j) {
            a[i][j] = 0;
        }
        for (int i = 0; i < m; ++i) b[i] = 0;
        for (int i = 0; i < n; ++i) c[i] = sol[i] = 0;
    }
    void pivot (int x, int y) {
        swap(Left[x], Down[y]);
        T k = a[x][y]; a[x][y] = 1;
        vector<int> nz;
        for (int i = 0; i < n; ++i) {
            a[x][i] /= k;
            if(!eq(a[x][i], 0)) nz.push_back(i);
        }
        b[x] /= k;
        for (int i = 0; i < m; ++i) {
            if (i == x || eq(a[i][y], 0)) continue;
            k = a[i][y], a[i][y] = 0;
            b[i] -= k * b[x];
            for (int j : nz) a[i][j] -= k * a[x][j];
        }
        if (eq(c[y], 0)) return;
        k = c[y], c[y] = 0, v += k * b[x];
        for (int i : nz) c[i] -= k * a[x][i];
    }
    // 0: found solution, 1: no feasible solution, 2:
    // unbounded
    int solve() {
        for (int i = 0; i < n; ++i) Down[i] = i;
        for (int i = 0; i < m; ++i) Left[i] = n + i;
        while (1) {
            int x = -1, y = -1;
            for (int i = 0; i < m; ++i) if (ls(b[i], 0) && (x == -1 || b[i] < b[x])) x = i;
            if (x == -1) break;
            for (int i = 0; i < n; ++i) if (ls(a[x][i], 0) && (y == -1 || a[x][i] < a[x][y])) y = i;
            if (y == -1) return 1;
            pivot(x, y);
        }
        while (1) {
            int x = -1, y = -1;
            for (int i = 0; i < n; ++i) if (ls(0, c[i]) && (y == -1 || c[i] > c[y])) y = i;
            if (y == -1) break;
            for (int i = 0; i < m; ++i) if (ls(0, a[i][y]) && (x == -1 || b[i] / a[i][y] < b[x] / a[x][y])) x = i;
            if (x == -1) return 2;
            pivot(x, y);
        }
        for (int i = 0; i < m; ++i) if(Left[i] < n) sol[Left[i]] = b[i];
        return 0;
    }
} LP;
```


6.9 Miller Rabin [f458b0]

```
ull mpow(__uint128_t a, ull b, ull m);
bool is_prime(ull x) {
    static auto witn = [](ull a, ull n, int t) {
        if (!a) return false;
        while (t--) {
            ull a2 = __uint128_t(a) * a % n;
            if (a2 == 1 && a != 1 && a != n - 1) return true;
            a = a2;
        }
        return a != 1;
    };
    if (x < 2) return false;
    if (!(x & 1)) return x == 2;
    int t = __builtin_ctzll(x - 1);
    ull odd = (x - 1) >> t;
    for (ull m:
        {2, 325, 9375, 28178, 450775, 9780504,
         1795265022})
        if (witn(mpow(m % x, odd, x), x, t))
            return false;
    return true;
}
```

6.10 Pollard's Rho [5f80a7]

```
ull f(ull x, ull k, ull m) {
    return (__uint128_t(x) * x + k) % m;
}
// does not work when n is prime
// return any non-trivial factor (NOT necessary be a
// prime)
ull pollard_rho(ull n) {
    if (!(n & 1)) return 2;
    mt19937_64 rnd(120821011);
    while (true) {
        ull y = 2, yy = y, x = rnd() % n, t = 1;
        for (ull sz = 2; t == 1; sz <= 1, y = yy) {
            for (ull i = 0; t == 1 && i < sz; ++i) {
                yy = f(yy, x, n);
                t = __gcd(yy > y ? yy - y : y - yy, n);
            }
        }
        if (t != 1 && t != n) return t;
    }
}
```

6.11 Gauss Elimination [93d9bd]

```
// Returns n - rank
int gauss_elimination(vector<vector<double>> &d) {
    int n = d.size(), m = d[0].size();
    for (int i = 0, r = 0; i < m; ++i) {
        int p = -1;
        for (int j = r; j < n; ++j) {
            if (fabs(d[j][i]) < eps) continue;
            if (p == -1 || fabs(d[j][i]) > fabs(d[p][i])) p = j;
        }
        if (p == -1) continue;
        swap(d[p], d[r]);
        for (int j = 0; j < n; ++j) {
            if (r == j) continue;
            double z = d[j][i] / d[r][i];
            for (int k = 0; k < m; ++k) d[j][k] -= z * d[r][k];
        }
        ++r;
    }
    return r;
}
```

6.12 Determinant [bb39ae]

```
ll Det(vector<vector<ll>> a) {
    int n = a.size();
    ll det = 1;
    for (int i = 0; i < n; ++i) {
        if (!a[i][i]) {
            det = -det;
            if (det < 0) det += MOD;
            for (int j = i + 1; j < n; ++j) if (a[j][i]) {
                swap(a[j], a[i]);
                break;
            }
        }
    }
}
```

```
    }
    if (!a[i][i]) return 0;
}
det = det * a[i][i] % MOD;
ll mul = mpow(a[i][i], MOD - 2);
for (int j = 0; j < n; ++j)
    a[i][j] = a[i][j] * mul % MOD;
for (int j = 0; j < n; ++j) if (i ^ j) {
    ll mul = a[j][i];
    for (int k = 0; k < n; ++k) {
        a[j][k] -= a[i][k] * mul % MOD;
        if (a[j][k] < 0) a[j][k] += MOD;
    }
}
}
return det;
} // not tested
```

6.13 Fast Fourier Transform [8389ae]

```
using cplx = complex<double>;
const double pi = acos(-1);
cplx omega[maxn * 4];
void prefft(int n) {
    for (int i = 0; i <= n; i++)
        omega[i] = cplx(cos(2 * pi * i / n),
                        sin(2 * pi * i / n));
}
void fft(vector<cplx> &v, int n) {
    int z = __builtin_ctz(n) - 1;
    for (int i = 0; i < n; i++) {
        int x = 0, j = 0;
        for (; (1 << j) < n; j++) x ^= (i >> j & 1) << (z - j);
        if (x > i) swap(v[x], v[i]);
    }
    for (int s = 2; s <= n; s <= 1) {
        int z = s >> 1;
        for (int i = 0; i < n; i += s) {
            for (int k = 0; k < z; k++) {
                cplx x = v[i + z + k] * omega[n / s * k];
                v[i + z + k] = v[i + k] - x;
                v[i + k] = v[i + k] + x;
            }
        }
    }
}
void ifft(vector<cplx> &v, int n) {
    fft(v, n); reverse(v.begin() + 1, v.end());
    for (int i = 0; i < n; i++) v[i] = v[i] * cplx(1.0 / n, 0);
}
v1 convolution(const v1 &a, const v1 &b) {
    // Should be able to handle N <= 10^5, C <= 10^4
    int sz = 1, tot = a.size() + b.size() - 1;
    while (sz < tot) sz <= 1;
    prefft(sz);
    vector<cplx> v(sz);
    for (int i = 0; i < sz; i++) {
        double re = i < a.size() ? a[i] : 0;
        double im = i < b.size() ? b[i] : 0;
        v[i] = cplx(re, im);
    }
    fft(v, sz);
    for (int i = 0; i <= sz / 2; i++) {
        int j = (sz - i) & (sz - 1);
        cplx x = (v[i] + conj(v[j])) * (v[i] - conj(v[j]))
                * cplx(0, -0.25);
        if (j != i) v[j] = (v[j] + conj(v[i])) * (v[j] - conj(v[i]))
                * cplx(0, -0.25);
        v[i] = x;
    }
    ifft(v, sz);
    v1 c(sz);
    for (int i = 0; i < sz; i++) c[i] = round(v[i].real());
    c.resize(tot);
    return c;
}
```

6.14 3 Primes NTT [8f0997]

```
// MOD: arbitrary prime
const int M1 = 998244353;
const int M2 = 1004535809;
```

```
const int M3 = 2013265921;
int super_big_crt(int64_t A, int64_t B, int64_t C) {
    static_assert(M1 <= M2 && M2 <= M3);
    ll r12 = mpow(M1, M2 - 2, M2);
    ll r13 = mpow(M1, M3 - 2, M3);
    ll r23 = mpow(M2, M3 - 2, M3);
    ll M1M2 = 1LL * M1 * M2 % MOD;
    B = (B - A + M2) * r12 % M2;
    C = (C - A + M3) * r13 % M3;
    C = (C - B + M3) * r23 % M3;
    return (A + B * M1 + C * M1M2) % MOD;
} // return ans % MOD
```

6.15 Number Theory Transform [c79a7e]

```
/* mod | g | maxn possible values:
998244353 | 3 | 8388608
1004535809 | 3 | 2097152
2013265921 | 31 | 134217728 */
template <int mod, int G, int maxn>
struct NTT {
    ll mpow(ll a, ll b) {
        ll res = 1;
        for (; b >= 1, a = a * a % mod)
            if (b & 1)
                res = res * a % mod;
        return res;
    }
    static_assert(maxn == (maxn & -maxn));
    int roots[maxn];
    NTT() {
        ll r = mpow(G, (mod - 1) / maxn);
        for (int i = maxn >> 1; i; i >>= 1) {
            roots[i] = 1;
            for (int j = 1; j < i; j++)
                roots[i + j] = roots[i + j - 1] * r % mod;
            r = r * r % mod;
        }
        // n = f.size() must be 2^k, and 0 <= f[i] < mod
        // n >= the size after convolution
        // practical:
        // int sz = 1;
        // while(sz < n + m - 1) sz <<= 1;
        void operator()(vector<ll> &f, int n, bool inv =
            false) {
            for (int i = 0, j = 0; i < n; i++) {
                if (i < j) swap(f[i], f[j]);
                for (int k = n >> 1; (j ^= k) < k; k >>= 1) {
                    for (int s = 1; s < n; s *= 2) {
                        for (int i = 0; i < n; i += s * 2) {
                            for (int j = 0; j < s; j++) {
                                ll a = f[i + j];
                                ll b = f[i + j + s] * roots[s + j] % mod;
                                f[i + j] = (a + b) % mod;
                                f[i + j + s] = (a - b + mod) % mod;
                            }
                        }
                    }
                }
            }
            if (inv) {
                int invn = mpow(n, mod - 2);
                for (int i = 0; i < n; i++)
                    f[i] = f[i] * invn % mod;
                reverse(f.begin() + 1, f.end());
            }
        }
    };
};
```

6.16 Berlekamp Massey [73b3cc]

```
// need add, sub, mul
vector<ll> BerlekampMassey(vector<ll> a) {
    // find min |c| such that a_n = sum c_j * a_{n - j - 1}, 0-based
    // O(N^2), if |c| = k, |a| >= 2k sure correct
    auto f = [&](vector<ll> v, ll c) {
        for (ll &x : v) x = mul(x, c);
        return v;
    };
    vector<ll> c, best;
    int pos = 0, n = a.size();
    for (int i = 0; i < n; ++i) {
        ll error = a[i];
```

```
        for (int j = 0; j < c.size(); ++j)
            error = sub(error, mul(c[j], a[i - 1 - j]));
        if (error == 0) continue;
        ll inv = mpow(error, mod - 2);
        if (c.empty()) {
            c.resize(i + 1), pos = i, best.pb(inv);
        } else {
            vector<ll> fix = f(best, error);
            fix.insert(fix.begin(), i - pos - 1, 0);
            if (fix.size() >= c.size()) {
                best = f(c, sub(0, inv));
                best.insert(best.begin(), inv);
                pos = i, c.resize(fix.size());
            }
            for (int j = 0; j < fix.size(); ++j)
                c[j] = add(c[j], fix[j]);
        }
    }
    return c;
}
```

6.17 Fraction [5c3898]

```
struct fraction {
    ll n, d;
    fraction(const ll _n=0, const ll _d=1): n(_n), d(_d)
    {
        ll t = gcd(n, d);
        n /= t, d /= t;
        if (d < 0) n = -n, d = -d;
    }
    fraction operator-() const
    { return fraction(-n, d); }
    fraction operator+(const fraction &b) const
    { return fraction(n * b.d + b.n * d, d * b.d); }
    fraction operator-(const fraction &b) const
    { return fraction(n * b.d - b.n * d, d * b.d); }
    fraction operator*(const fraction &b) const
    { return fraction(n * b.n, d * b.d); }
    fraction operator/(const fraction &b) const
    { return fraction(n * b.d, d * b.n); }
    void print() {
        cout << n;
        if (d != 1) cout << "/" << d;
    }
}; // not tested
```

7 Misc

7.1 Josephus Problem [f4494f]

```
// n people kill m for each turn
int f(int n, int m) {
    int s = 0;
    for (int i = 2; i <= n; i++)
        s = (s + m) % i;
    return s;
}
// died at kth
int kth(int n, int m, int k){
    if (m == 1) return n-1;
    for (k = k*m+m-1; k >= n; k = k-n+(k-n)/(m-1));
    return k;
} // both not tested
```