

# Contents

<b>1 Basic</b>	<b>1</b>
1.1 Default Code	1
1.2 .vimrc	1
1.3 Fast IO	1
1.4 Random	2
1.5 PBDS Tree	2
1.6 Pragma	2
1.7 SVG Writer	2
<b>2 Data Structure</b>	<b>2</b>
2.1 Heavy-Light Decomposition	2
2.2 Link Cut Tree	2
2.3 Treap	3
2.4 KD Tree	3
2.5 Leftist Tree	4
2.6 Convex 1D/1D	4
<b>3 Flow &amp; Matching</b>	<b>4</b>
3.1 Dinic	4
3.2 Bounded Flow	4
3.3 MCMF	5
3.4 Min Cost Circulation	5
3.5 Gomory Hu	5
3.6 Stoer Wagner Algorithm	5
3.7 Bipartite Matching	6
3.8 Kuhn Munkres Algorithm	6
3.9 Max Simple Graph Matching	6
3.10 Stable Marriage	7
3.11 Flow Model	7
<b>4 Geometry</b>	<b>7</b>
4.1 Geometry Template	7
4.2 Polar Angle Comparator	8
4.3 Minkowski Sum	8
4.4 Intersection of Circle and Convex Polygon	8
4.5 Intersection of Circles	8
4.6 Tangent Line of Circles	8
4.7 Intersection of Line and Convex Polygon	8
4.8 Intersection of Line and Circle	9
4.9 Point in Circle	9
4.10 Point in Convex	9
4.11 Half Plane Intersection	9
4.12 Minimum Enclosing Circle	9
4.13 3D Point	9
4.14 ConvexHull3D	10
4.15 Delaunay Triangulation	10
4.16 Voronoi Diagram	11
4.17 Polygon Union	11
4.18 Tangent Point to Convex Hull	11
4.19 Heart	11
4.20 Rotating Sweep Line	11
4.21 Vector In Poly	11
4.22 Convex Hull DP	11
4.23 Calculate Points in Triangle	12
<b>5 Graph</b>	<b>12</b>
5.1 BCC	12
5.2 SCC	12
5.3 2-SAT	12
5.4 Dominator Tree	13
5.5 Virtual Tree	13
5.6 Fast DMST	13
5.7 Vizing	13
5.8 Maximum Clique	14
5.9 Number of Maximal Clique	14
5.10 Minimum Mean Cycle	14

5.11 Minimum Steiner Tree	15
5.12 Count Cycles	15
<b>6 Math</b>	<b>15</b>
6.1 Extended Euclidean Algorithm	15
6.2 Floor & Ceil	15
6.3 Legendre	15
6.4 Simplex	15
6.5 Simplex Construction	16
6.6 DiscreteLog	16
6.7 Miller Rabin & Pollard Rho	16
6.8 XOR Basis	16
6.9 Linear Equation	17
6.10 Chinese Remainder Theorem	17
6.11 Sqrt Decomposition	17
6.12 Floor Sum	17
<b>7 Polynomial</b>	<b>17</b>
7.1 FWHT	17
7.2 FFT	18
7.3 NTT	18
7.4 Polynomial Operation	18
7.5 Generating Function	20
Ordinary Generating Function	20
Exponential Generating Function	20
7.6 Bostan Mori	20
<b>8 String</b>	<b>21</b>
8.1 KMP Algorithm	21
8.2 Manacher Algorithm	21
8.3 Lyndon Factorization	21
8.4 Suffix Array	21
8.5 Suffix Automaton	21
8.6 Z-value Algorithm	22
8.7 Main Lorentz	22
8.8 AC Automaton	22
8.9 Palindrome Automaton	23
<b>9 Misc</b>	<b>23</b>
9.1 Cyclic Ternary Search	23
9.2 Matroid	23
9.3 Simulate Annealing	23
9.4 Binary Search On Fraction	23
9.5 Min Plus Convolution	23
<b>10 Notes</b>	<b>23</b>
10.1 Geometry	23
Rotation Matrix	23
Triangles	24
Quadrilaterals	24
Spherical coordinates	24
Green's Theorem	24
Point-Line Duality	24
10.2 Trigonometry	24
10.3 Calculus	24
10.4 Sum & Series	24
10.5 Misc	25
10.6 Number	25

```

void debug(T a, U ... b){cerr << a << " ", debug(b...);}
template<class T> void pary(T l, T r){
    while (l != r) cerr << *l << " ", l++;
    cerr << "\n";
}
#else
#define debug(...) void()
#define pary(...) void()
#endif

template<class A, class B>
ostream& operator<<(ostream& o, pair<A,B> p)
{ return o << '(' << p.ff << ',' << p.ss << ')'; }

int main(){
    ios_base::sync_with_stdio(0); cin.tie(0);
}

```

## 1.2 .vimrc [476d29]

```

sy on
se nu rnu bs=2 sw=4 ts=4 hls ls=2 si acd bo=all mouse=a
map <F9> :w<bar>!g++ "% " -o %:r -std=c++17 -Wall -
Wextra -Wshadow -O2 -Dzisk -g -fsanitize=undefined,
address<CR>
map <F8> :!./%:r<CR>
inoremap {<CR> {<CR>}<ESC>ko
ca Hash w !cpp -dD -P -fpreprocessed \ | tr -d '[:space
:]' \ | md5sum \ | cut -c-6
inoremap fj <ESC>
vnoremap fj <ESC>
" -D_GLIBCXX_ASSERTIONS, -D_GLIBCXX_DEBUG

```

## 1.3 Fast IO [c7ddfd]

```

// from JAW
inline int my_getchar() {
    const int N = 1<<20;
    static char buf[N];
    static char *p = buf, *end = buf;
    if(p == end) {
        if((end = buf + fread(buf, 1, N, stdin)) == buf)
            return EOF;
        p = buf;
    }
    return *p++;
}

inline int readint(int &x) {
    static char c, neg;
    while((c = my_getchar()) < '-') {
        if(c == EOF) return 0;
    }
    neg = (c == '-') ? -1 : 1;
    x = (neg == 1) ? c - '0' : 0;
    while((c = my_getchar()) >= '0') x = (x << 3) + (x <<
        1) + (c - '0');
    x *= neg;
    return 1;
}

```

```

const int kBufSize = 524288;
char inbuf[kBufSize];
char buf_[kBufSize]; size_t size_;
inline void Flush_() { write(1, buf_, size_); size_ =
    0; }
inline void CheckFlush_(size_t sz) { if (sz + size_ >
    kBufSize) Flush_(); }

inline void PutInt(int a) {
    static char tmp[22] = "01234567890123456789\n";
    CheckFlush_(10);
    if(a < 0){
        *(buf_ + size_) = '-';
        a = ~a + 1;
        size_++;
    }
    int tail = 20;
    if (!a) {
        tmp[--tail] = '0';
    } else {
        for (; a; a /= 10) tmp[--tail] = (a % 10) ^ '0';
    }
}

```

## 1 Basic

### 1.1 Default Code [37e06b]

```

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

#define iter(v) v.begin(),v.end()
#define SZ(v) int(v.size())
#define pb emplace_back
#define ff first
#define ss second

using ll = long long;
using pii = pair<int, int>;
using pll = pair<ll, ll>;

#ifdef zisk
void debug(){cerr << "\n";}
template<class T, class ... U>

```

```

    }
    memcpy(buf_ + size_, tmp + tail, 21 - tail);
    size_ += 21 - tail;
}

int main(){
    Flush_();
    return 0;
}

```

## 1.4 Random [4cf9ed]

```

mt19937 rng(chrono::system_clock::now().
    time_since_epoch().count());

```

## 1.5 PBDS Tree [7c702a]

```

#include <bits/extc++.h>
using namespace __gnu_pbds;
using Tree = tree<int, null_type, less<, rb_tree_tag,
    tree_order_node_statistics_update>;
// .find_by_order(x)
// .order_of_key(x)

```

## 1.6 Pragma [6006f6]

```

#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,arch=skylake")
__builtin_ia32_ldmxcsr(__builtin_ia32_stmxcsr())|0x8040)

```

## 1.7 SVG Writer [7adcc8]

```

class SVG {
    void p(string_view s) { o << s; }
    void p(string_view s, auto v, auto... vs) {
        auto i = s.find('$');
        o << s.substr(0, i) << v, p(s.substr(i + 1), vs...);
    }
    ofstream o; string c = "red";
public: // SVG svg("test.svg", 0, 0, 100, 100)
    SVG(auto f, auto x1, auto y1, auto x2, auto y2) : o(f)
    {
        p("<svg xmlns='http://www.w3.org/2000/svg' "
            "viewBox='$ $ $ $'>\n"
            "<style>{*stroke-width:0.5%;}</style>\n",
            x1, -y2, x2 - x1, y2 - y1); }
    ~SVG() { p("</svg>\n"); }
    void color(string nc) { c = nc; }
    void line(auto x1, auto y1, auto x2, auto y2) {
        p("<line x1='$' y1='$' x2='$' y2='$' stroke='$'>\n",
            x1, -y1, x2, -y2, c); }
    void circle(auto x, auto y, auto r) {
        p("<circle cx='$' cy='$' r='$' stroke='$' "
            "fill='none'/>\n", x, -y, r, c); }
    void text(auto x, auto y, string s, int w = 12) {
        p("<text x='$' y='$' font-size='$px'></text>\n",
            x, -y, w, s); }
};

```

# 2 Data Structure

## 2.1 Heavy-Light Decomposition [f2dbca]

```

struct HLD{ // 1-based
    int n, ts = 0; // ord is 1-based
    vector<vector<int>> g;
    vector<int> par, top, down, ord, dpt, sub;
    explicit HLD(int _n): n(_n), g(n + 1),
        par(n + 1), top(n + 1), down(n + 1),
        ord(n + 1), dpt(n + 1), sub(n + 1) {}
    void add_edge(int u, int v){ g[u].pb(v); g[v].pb(u); }
    void dfs(int now, int p){
        par[now] = p; sub[now] = 1;
        for(int i : g[now]){
            if(i == p) continue;
            dpt[i] = dpt[now] + 1;
            dfs(i, now);
            sub[now] += sub[i];
            if(sub[i] > sub[down[now]]) down[now] = i;
        }
    }
};

```

```

    }
}

void cut(int now, int t){
    top[now] = t; ord[now] = ++ts;
    if(!down[now]) return;
    cut(down[now], t);
    for(int i : g[now]){
        if(i != par[now] && i != down[now])
            cut(i, i);
    }
}

void build(){ dfs(1, 1), cut(1, 1); }
int query(int a, int b){
    int ta = top[a], tb = top[b];
    while(ta != tb){
        if(dpt[ta] > dpt[tb]) swap(ta, tb), swap(a, b);
        // ord[ta], ord[tb]
        tb = top[b = par[tb]];
    }
    if(ord[a] > ord[b]) swap(a, b);
    // ord[a], ord[b]
    return a; // lca
}

};

2.2 Link Cut Tree [cf4f34]
// 1-based
template <typename Val, typename SVal> struct LCT {
    struct node {
        int pa, ch[2]; bool rev; int size;
        Val v, sum, rsum; SVal sv, sub, vir;
        node() : pa{0}, ch{0, 0}, rev{false}, size{1}, v{},
            sum{}, rsum{}, sv{}, sub{}, vir{} {}
    };
    #define cur o[u]
    #define lc cur.ch[0]
    #define rc cur.ch[1]
    vector<node> o;
    bool is_root(int u) const {
        return o[cur.pa].ch[0] != u && o[cur.pa].ch[1] != u; }
    bool is_rch(int u) const {
        return o[cur.pa].ch[1] == u && !is_root(u); }
    void down(int u) {
        for (int c : {lc, rc}) if (c) {
            if (cur.rev) set_rev(c);
        }
        cur.rev = false;
    }
    void up(int u) {
        cur.sum = o[lc].sum + cur.v + o[rc].sum;
        cur.rsum = o[rc].rsum + cur.v + o[lc].rsum;
        cur.sub = cur.vir + o[lc].sub + o[rc].sub + cur.sv;
        cur.size = o[lc].size + o[rc].size + 1;
    }
    void set_rev(int u) {
        swap(lc, rc), swap(cur.sum, cur.rsum);
        cur.rev ^= 1;
    }
    /* --- */
    void rotate(int u) {
        int f = cur.pa, g = o[f].pa, l = is_rch(u);
        if (cur.ch[l ^ 1]) o[cur.ch[l ^ 1]].pa = f;
        if (not is_root(f)) o[g].ch[is_rch(f)] = u;
        o[f].ch[l] = cur.ch[l ^ 1], cur.ch[l ^ 1] = f;
        cur.pa = g, o[f].pa = u; up(f);
    }
    void splay(int u) {
        vector<int> stk = {u};
        while (not is_root(stk.back()))
            stk.push_back(o[stk.back()].pa);
        while (not stk.empty())
            down(stk.back()), stk.pop_back();
        for (int f = cur.pa; not is_root(u); f = cur.pa) {
            if (!is_root(f))
                rotate(is_rch(u) == is_rch(f) ? f : u);
            rotate(u);
        }
        up(u);
    }
    void access(int x) {
        for (int u = x, last = 0; u; u = cur.pa) {
            splay(u);
        }
    }
};

```

```

    cur.vir = cur.vir + o[rc].sub - o[last].sub;
    rc = last; up(last = u);
}
splay(x);
}
int find_root(int u) {
    int la = 0;
    for (access(u); u; u = lc) down(la = u);
    return la;
}
void split(int x, int y) { chroot(x); access(y); }
void chroot(int u) { access(u); set_rev(u); }
/* --- */
LCT(int n = 0) : o(n + 1) { o[0].size = 0; }
void set_val(int u, const Val &v) {
    splay(u); cur.v = v; up(u); }
void set_sval(int u, const SVal &v) {
    access(u); cur.sv = v; up(u); }
Val query(int x, int y) {
    split(x, y); return o[y].sum; }
SVal subtree(int p, int u) {
    chroot(p); access(u); return cur.vir + cur.sv; }
bool connected(int u, int v) {
    return find_root(u) == find_root(v); }
void link(int x, int y) {
    chroot(x); access(y);
    o[y].vir = o[y].vir + o[x].sub;
    up(o[x].pa = y);
}
void cut(int x, int y) {
    split(x, y); o[y].ch[0] = o[x].pa = 0; up(y); }
#undef cur
#undef lc
#undef rc
};

```

### 2.3 Treap [2ac37e]

```

mt19937 rng(880301);
struct node {
    ll data; int sz;
    node *l, *r;
    node(ll k = 0) : data(k), sz(1), l(0), r(0) {}
    void up() {
        sz = 1;
        if (l) sz += l->sz;
        if (r) sz += r->sz;
    }
    void down() {}
};
node pool[1000010]; int pool_cnt = 0;
node *newnode(ll k) { return &(pool[pool_cnt++] = node(k)); }
int sz(node *a) { return a ? a->sz : 0; }
node *merge(node *a, node *b) {
    if (!a || !b) return a ? a : b;
    if (int(rng() % (sz(a) + sz(b))) < sz(a))
        return a->down(), a->r = merge(a->r, b), a->up(),
            a;
    return b->down(), b->l = merge(a, b->l), b->up(), b;
}
// a: key <= k, b: key > k
void split(node *o, node *&a, node *&b, ll k) {
    if (!o) return a = b = 0, void();
    o->down();
    if (o->data <= k)
        a = o, split(o->r, a->r, b, k), a->up();
    else b = o, split(o->l, a, b->l, k), b->up();
}
// a: size k, b: size n - k
void split2(node *o, node *&a, node *&b, int k) {
    if (sz(o) <= k) return a = o, b = 0, void();
    o->down();
    if (sz(o->l) + 1 <= k)
        a = o, split2(o->r, a->r, b, k - sz(o->l) - 1);
    else b = o, split2(o->l, a, b->l, k);
    o->up();
}
node *kth(node *o, ll k) { // 1-based
    if (k <= sz(o->l)) return kth(o->l, k);
    if (k == sz(o->l) + 1) return o;
    return kth(o->r, k - sz(o->l) - 1);
}

```

```

int Rank(node *o, ll key) { // num of key < key
    if (!o) return 0;
    if (o->data < key)
        return sz(o->l) + 1 + Rank(o->r, key);
    else return Rank(o->l, key);
}
bool erase(node *&o, ll k) {
    if (!o) return 0;
    if (o->data == k) {
        node *t = 0;
        o->down(), o = merge(o->l, o->r);
        return 1;
    }
    node *&t = k < o->data ? o->l : o->r;
    return erase(t, k) ? o->up(), 1 : 0;
}
void insert(node *&o, ll k) {
    node *a, *b;
    split(o, a, b, k),
        o = merge(a, merge(new node(k), b));
}
tuple<node*, node*, node*> interval(node *&o, int l,
    int r) { // 1-based
    node *a, *b, *c; // b: [l, r]
    split2(o, a, b, l - 1), split2(b, b, c, r - l + 1);
    return make_tuple(a, b, c);
}

```

### 2.4 KD Tree [375ca2]

```

namespace kdt {
    int root, lc[maxn], rc[maxn], xl[maxn], xr[maxn],
        yl[maxn], yr[maxn];
    point p[maxn];
    int build(int l, int r, int dep = 0) {
        if (l == r) return -1;
        function<bool(const point &, const point &)> f =
            [dep](const point &a, const point &b) {
                if (dep & 1) return a.x < b.x;
                else return a.y < b.y;
            };
        int m = (l + r) >> 1;
        nth_element(p + l, p + m, p + r, f);
        xl[m] = xr[m] = p[m].x;
        yl[m] = yr[m] = p[m].y;
        lc[m] = build(l, m, dep + 1);
        if (~lc[m]) {
            xl[m] = min(xl[m], xl[lc[m]]);
            xr[m] = max(xr[m], xr[lc[m]]);
            yl[m] = min(yl[m], yl[lc[m]]);
            yr[m] = max(yr[m], yr[lc[m]]);
        }
        rc[m] = build(m + 1, r, dep + 1);
        if (~rc[m]) {
            xl[m] = min(xl[m], xl[rc[m]]);
            xr[m] = max(xr[m], xr[rc[m]]);
            yl[m] = min(yl[m], yl[rc[m]]);
            yr[m] = max(yr[m], yr[rc[m]]);
        }
        return m;
    }
    bool bound(const point &q, int o, long long d) {
        double ds = sqrt(d + 1.0);
        if (q.x < xl[o] - ds || q.x > xr[o] + ds ||
            q.y < yl[o] - ds || q.y > yr[o] + ds)
            return false;
        return true;
    }
    long long dist(const point &a, const point &b) {
        return (a.x - b.x) * 1ll * (a.x - b.x) +
            (a.y - b.y) * 1ll * (a.y - b.y);
    }
    void dfs(
        const point &q, long long &d, int o, int dep = 0)
        {
            if (!bound(q, o, d)) return;
            long long cd = dist(p[o], q);
            if (cd != 0) d = min(d, cd);
            if ((dep & 1) && q.x < p[o].x ||
                !(dep & 1) && q.y < p[o].y) {
                if (~lc[o]) dfs(q, d, lc[o], dep + 1);
                if (~rc[o]) dfs(q, d, rc[o], dep + 1);
            } else {

```

```

    if (~rc[o]) dfs(q, d, rc[o], dep + 1);
    if (~lc[o]) dfs(q, d, lc[o], dep + 1);
}
}
void init(const vector<point> &v) {
    for (int i = 0; i < v.size(); ++i) p[i] = v[i];
    root = build(0, v.size());
}
long long nearest(const point &q) {
    long long res = 1e18;
    dfs(q, res, root);
    return res;
}
} // namespace kdt

```

## 2.5 Leftist Tree [e91538]

```

struct node {
    ll v, data, sz, sum;
    node *l, *r;
    node(ll k)
        : v(0), data(k), sz(1), l(0), r(0), sum(k) {}
};
ll sz(node *p) { return p ? p->sz : 0; }
ll V(node *p) { return p ? p->v : -1; }
ll sum(node *p) { return p ? p->sum : 0; }
node *merge(node *a, node *b) {
    if (!a || !b) return a ? a : b;
    if (a->data < b->data) swap(a, b);
    a->r = merge(a->r, b);
    if (V(a->r) > V(a->l)) swap(a->r, a->l);
    a->v = V(a->r) + 1, a->sz = sz(a->l) + sz(a->r) + 1;
    a->sum = sum(a->l) + sum(a->r) + a->data;
    return a;
}
void pop(node *&o) {
    node *tmp = o;
    o = merge(o->l, o->r);
    delete tmp;
}

```

## 2.6 Convex 1D/1D [79e448]

```

template<class T>
struct DynamicHull {
    struct seg { int x, l, r; };
    T f; int C; deque<seg> dq; // range: 1~C
    explicit DynamicHull(T _f, int _C): f(_f), C(_C) {}
    // max t s.t. f(x, t) >= f(y, t), x < y, maintain max
    int intersect(int x, int y) {
        int l = 0, r = C + 1;
        while (l + 1 < r) {
            int mid = (l + r) / 2;
            if (f(x, mid) >= f(y, mid)) l = mid;
            else r = mid;
        }
        return l;
    }
    void push_back(int x) {
        for (int i; !dq.empty() &&
            (i = dq.back().l, f(dq.back().x, i) < f(x, i));
            )
            dq.pop_back();
        if (dq.empty()) return dq.pb(seg({x, 1, C})), void(
            );
        dq.back().r = intersect(dq.back().x, x);
        dq.pb(seg({x, dq.back().l + 1, C}));
    }
    int query(int x) {
        while (dq.front().r < x) dq.pop_front();
        return dq.front().x;
    }
};

```

# 3 Flow & Matching

## 3.1 Dinic [801a71]

```

struct Dinic { // 0-based,  $O(V^2E)$ , unit flow:  $O(\min(V^{2/3}E, E^{3/2}))$ , bipartite matching:  $O(\sqrt{V}E)$ 
    struct edge {
        ll to, cap, flow, rev;
    };
};

```

```

int n, s, t;
vector<vector<edge>> g;
vector<int> dis, ind;

void init(int _n) {
    n = _n;
    g.assign(n, vector<edge>());
}
void reset() {
    for (int i = 0; i < n; ++i)
        for (auto &j : g[i]) j.flow = 0;
}
void add_edge(int u, int v, ll cap) {
    g[u].pb(edge{v, cap, 0, SZ(g[v]))};
    g[v].pb(edge{u, 0, 0, SZ(g[u]) - 1});
    //change g[v] to cap for undirected graphs
}
bool bfs() {
    dis.assign(n, -1);
    queue<int> q;
    q.push(s), dis[s] = 0;
    while (!q.empty()) {
        int cur = q.front(); q.pop();
        for (auto &e : g[cur]) {
            if (dis[e.to] == -1 && e.flow != e.cap) {
                q.push(e.to);
                dis[e.to] = dis[cur] + 1;
            }
        }
    }
    return dis[t] != -1;
}
ll dfs(int u, ll cap) {
    if (u == t || !cap) return cap;
    for (int &i = ind[u]; i < SZ(g[u]); ++i) {
        edge &e = g[u][i];
        if (dis[e.to] == dis[u] + 1 && e.flow != e.cap) {
            ll df = dfs(e.to, min(e.cap - e.flow, cap));
            if (df) {
                e.flow += df;
                g[e.to][e.rev].flow -= df;
                return df;
            }
        }
    }
    dis[u] = -1;
    return 0;
}
ll maxflow(int _s, int _t) {
    s = _s; t = _t;
    ll flow = 0, df;
    while (bfs()) {
        ind.assign(n, 0);
        while ((df = dfs(s, INF))) flow += df;
    }
    return flow;
}
};

```

## 3.2 Bounded Flow [758826]

```

struct BoundedFlow : Dinic {
    vector<ll> tot;
    void init(int _n) {
        Dinic::init(_n + 2);
        tot.assign(n, 0);
    }
    void add_edge(int u, int v, ll lcap, ll rcap) {
        tot[u] -= lcap, tot[v] += lcap;
        g[u].pb(edge{v, rcap, lcap, SZ(g[v]))};
        g[v].pb(edge{u, 0, 0, SZ(g[u]) - 1});
    }
    bool feasible() {
        ll sum = 0;
        int vs = n - 2, vt = n - 1;
        for (int i = 0; i < n - 2; ++i)
            if (tot[i] > 0)
                add_edge(vs, i, 0, tot[i]), sum += tot[i];
            else if (tot[i] < 0) add_edge(i, vt, 0, -tot[i]);
        if (sum != maxflow(vs, vt)) sum = -1;
        for (int i = 0; i < n - 2; ++i)
            if (tot[i] > 0)

```

```

        g[vs].pop_back(), g[i].pop_back();
    else if(tot[i] < 0)
        g[i].pop_back(), g[vt].pop_back();
    return sum != -1;
}
ll boundedflow(int _s, int _t) {
    add_edge(_t, _s, 0, INF);
    if(!feasible()) return -1;
    ll x = g[_t].back().flow;
    g[_t].pop_back(), g[_s].pop_back();
    return x - maxflow(_t, _s); // min
    //return x + maxflow(_s, _t); // max
}
};

```

### 3.3 MCMF [8d3644]

```

struct MCMF { //  $\theta$ -based,  $O(SPFA * |f|)$ 
    struct edge {
        ll from, to, cap, flow, cost, rev;
    };
    int n;
    int s, t; ll mx;
    //mx: maximum amount of flow
    vector<vector<edge>> g;
    vector<ll> dis, up;
    bool BellmanFord(ll &flow, ll &cost) {
        vector<edge*> past(n);
        vector<int> inq(n);
        dis.assign(n, INF); up.assign(n, 0);
        queue<int> q;
        q.push(s), inq[s] = 1;
        up[s] = mx - flow, past[s] = 0, dis[s] = 0;
        while (!q.empty()) {
            int u = q.front();
            q.pop(), inq[u] = 0;
            if (!up[u]) continue;
            for (auto &e : g[u])
                if (e.flow != e.cap &&
                    dis[e.to] > dis[u] + e.cost) {
                    dis[e.to] = dis[u] + e.cost, past[e.to] = &e;
                    up[e.to] = min(up[u], e.cap - e.flow);
                    if (!inq[e.to]) inq[e.to] = 1, q.push(e.to);
                }
            if (dis[t] == INF) return 0;
            flow += up[t], cost += up[t] * dis[t];
            for (ll i = t; past[i]; i = past[i]->from) {
                auto &e = *past[i];
                e.flow += up[t], g[e.to][e.rev].flow -= up[t];
            }
            return 1;
        }
    }
    pll MinCostMaxFlow(int _s, int _t) {
        s = _s, t = _t;
        ll flow = 0, cost = 0;
        while (BellmanFord(flow, cost));
        return pll(flow, cost);
    }
    void init(int _n, ll _mx) {
        n = _n, mx = _mx;
        g.assign(n, vector<edge>());
    }
    void add_edge(int a, int b, ll cap, ll cost) {
        g[a].pb(edge{a, b, cap, 0, cost, SZ(g[b])});
        g[b].pb(edge{b, a, 0, 0, -cost, SZ(g[a]) - 1});
    }
};

```

### 3.4 Min Cost Circulation [47cf18]

```

struct MinCostCirculation { //  $\theta$ -based,  $O(VE * E \log C)$ 
    struct edge {
        ll from, to, cap, fcap, flow, cost, rev;
    };
    int n;
    vector<edge*> past;
    vector<vector<edge>> g;
    vector<ll> dis;
    void BellmanFord(int s) {
        vector<int> inq(n);
        dis.assign(n, INF);
        queue<int> q;
    }
};

```

```

auto relax = [&](int u, ll d, edge *e) {
    if (dis[u] > d) {
        dis[u] = d, past[u] = e;
        if (!inq[u]) inq[u] = 1, q.push(u);
    }
};
relax(s, 0, 0);
while (!q.empty()) {
    int u = q.front();
    q.pop(), inq[u] = 0;
    for (auto &e : g[u])
        if (e.cap > e.flow)
            relax(e.to, dis[u] + e.cost, &e);
}
}
void try_edge(edge &cur) {
    if (cur.cap > cur.flow) return ++cur.cap, void();
    BellmanFord(cur.to);
    if (dis[cur.from] + cur.cost < 0) {
        ++cur.flow, --g[cur.to][cur.rev].flow;
        for (int i = cur.from; past[i]; i = past[i]->from) {
            auto &e = *past[i];
            ++e.flow, --g[e.to][e.rev].flow;
        }
    }
    ++cur.cap;
}
void solve(int mxlg) { // mxlg >= Log(max cap)
    for (int b = mxlg; b >= 0; --b) {
        for (int i = 0; i < n; ++i)
            for (auto &e : g[i])
                e.cap *= 2, e.flow *= 2;
        for (int i = 0; i < n; ++i)
            for (auto &e : g[i])
                if (e.fcap >> b & 1)
                    try_edge(e);
    }
}
void init(int _n) {
    n = _n;
    past.assign(n, nullptr);
    g.assign(n, vector<edge>());
}
void add_edge(ll a, ll b, ll cap, ll cost) {
    g[a].pb(edge{a, b, 0, cap, 0, cost, SZ(g[b]) + (a == b)});
    g[b].pb(edge{b, a, 0, 0, 0, -cost, SZ(g[a]) - 1});
}
};

```

### 3.5 Gomory Hu [82d968]

```

void GomoryHu(Dinic &flow) { //  $\theta$ -based
    int n = flow.n;
    vector<int> par(n);
    for (int i = 1; i < n; ++i) {
        flow.reset();
        add_edge(i, par[i], flow.maxflow(i, par[i]));
        for (int j = i + 1; j < n; ++j)
            if (par[j] == par[i] && ~flow.dis[j])
                par[j] = i;
    }
}

```

### 3.6 Stoer Wagner Algorithm [a9917b]

```

struct StoerWagner { //  $\theta$ -based,  $O(V^3)$ 
    int n;
    vector<int> vis, del;
    vector<ll> wei;
    vector<vector<ll>> edge;
    void init(int _n) {
        n = _n;
        del.assign(n, 0);
        edge.assign(n, vector<ll>(n));
    }
    void add_edge(int u, int v, ll w) {
        edge[u][v] += w, edge[v][u] += w;
    }
    void search(int &s, int &t) {
        vis.assign(n, 0); wei.assign(n, 0);
        s = t = -1;
    }
};

```



```

while (1) {
    ll mx = -1, cur = 0;
    for (int i = 0; i < n; ++i)
        if (!del[i] && !vis[i] && mx < wei[i])
            cur = i, mx = wei[i];
    if (mx == -1) break;
    vis[cur] = 1, s = t, t = cur;
    for (int i = 0; i < n; ++i)
        if (!vis[i] && !del[i]) wei[i] += edge[cur][i];
}
}
ll solve() {
    ll ret = INF;
    for (int i = 0, x=0, y=0; i < n-1; ++i) {
        search(x, y), ret = min(ret, wei[y]), del[y] = 1;
        for (int j = 0; j < n; ++j)
            edge[x][j] = (edge[j][x] += edge[y][j]);
    }
    return ret;
}
};

```

### 3.7 Bipartite Matching [013c49]

*//min vertex cover: take all unmatched vertices in L  
and find alternating tree,  
//ans is not reached in L + reached in R*

*// O(VE)*  
int n; *// 1-based, max matching*  
int mx[maxn], my[maxn];  
bool adj[maxn][maxn], vis[maxn];  
bool dfs(int u) {  
 if (vis[u]) return 0;  
 vis[u] = 1;  
 for (int v = 1; v <= n; v++) {  
 if (!adj[u][v]) continue;  
 if (!my[v] || (my[v] && dfs(my[v]))) {  
 mx[u] = v, my[v] = u;  
 return 1;  
 }  
 }  
 return 0;  
}  
*// O(E sqrt(V)), O(E log V) for random sparse graphs*  
struct BipartiteMatching { *// 0-based*  
 int nl, nr;  
 vector<int> mx, my, dis, cur;  
 vector<vector<int>> g;  
 bool dfs(int u) {  
 for (int &i = cur[u]; i < SZ(g[u]); ++i) {  
 int e = g[u][i];  
 if (!my[e] || (dis[my[e]] == dis[u] + 1 && dfs(my[e])))  
 return mx[my[e] = u] = e, 1;  
 }  
 dis[u] = -1;  
 return 0;  
 }  
 bool bfs() {  
 int ret = 0;  
 queue<int> q;  
 dis.assign(nl, -1);  
 for (int i = 0; i < nl; ++i)  
 if (!mx[i]) q.push(i), dis[i] = 0;  
 while (!q.empty()) {  
 int u = q.front();  
 q.pop();  
 for (int e : g[u])  
 if (!my[e]) ret = 1;  
 else if (!dis[my[e]]) {  
 q.push(my[e]);  
 dis[my[e]] = dis[u] + 1;  
 }  
 }  
 return ret;  
 }  
 int matching() {  
 int ret = 0;  
 mx.assign(nl, -1); my.assign(nr, -1);  
 while (bfs()) {  
 cur.assign(nl, 0);  
 for (int i = 0; i < nl; ++i)  
 if (!mx[i] && dfs(i)) ++ret;  
 }  
 }  
};

```

}
return ret;
}
void add_edge(int s, int t) { g[s].pb(t); }
void init(int _nl, int _nr) {
    nl = _nl, nr = _nr;
    g.assign(nl, vector<int>());
}
};

```

### 3.8 Kuhn Munkres Algorithm [683e0a]

*struct KM { // 0-based, maximum matching, O(V^3)*  
 int n, ql, qr;  
 vector<vector<ll>> w;  
 vector<ll> hl, hr, slk;  
 vector<int> fl, fr, pre, qu, vl, vr;  
 void init(int \_n) {  
 n = \_n;  
*// -INF for perfect matching*  
 w.assign(n, vector<ll>(n, 0));  
 pre.assign(n, 0);  
 qu.assign(n, 0);  
 }  
 void add\_edge(int a, int b, ll wei) {  
 w[a][b] = wei;  
 }  
 bool check(int x) {  
 if (vl[x] = 1, ~fl[x])  
 return (vr[qu[qr++]] = fl[x]) = 1;  
 while (~x) swap(x, fr[fl[x] = pre[x]]);  
 return 0;  
 }  
 void bfs(int s) {  
 slk.assign(n, INF); vl.assign(n, 0); vr.assign(n, 0);  
 ql = qr = 0, qu[qr++] = s, vr[s] = 1;  
 for (ll d;;) {  
 while (ql < qr)  
 for (int x = 0, y = qu[ql++]; x < n; ++x)  
 if (!vl[x] && slk[x] >= (d = hl[x] + hr[y] - w[x][y])) {  
 if (pre[x] = y, d) slk[x] = d;  
 else if (!check(x)) return;  
 }  
 d = INF;  
 for (int x = 0; x < n; ++x)  
 if (!vl[x] && d > slk[x]) d = slk[x];  
 for (int x = 0; x < n; ++x) {  
 if (vl[x]) hl[x] += d;  
 else slk[x] -= d;  
 if (vr[x]) hr[x] -= d;  
 }  
 for (int x = 0; x < n; ++x)  
 if (!vl[x] && !slk[x] && !check(x)) return;  
 }  
 }  
 ll solve() {  
 fl.assign(n, -1); fr.assign(n, -1); hl.assign(n, 0);  
 hr.assign(n, 0);  
 for (int i = 0; i < n; ++i)  
 hl[i] = \*max\_element(iter(w[i]));  
 for (int i = 0; i < n; ++i) bfs(i);  
 ll res = 0;  
 for (int i = 0; i < n; ++i) res += w[i][fl[i]];  
 return res;  
 }  
};

### 3.9 Max Simple Graph Matching [907d7c]

*struct Matching { // 0-based, O(V^3)*  
 queue<int> q; int n;  
 vector<int> fa, s, vis, pre, match;  
 vector<vector<int>> g;  
 int Find(int u)  
 { return u == fa[u] ? u : fa[u] = Find(fa[u]); }  
 int LCA(int x, int y) {  
 static int tk = 0; tk++; x = Find(x); y = Find(y);  
 for (; swap(x, y)) if (x != n) {  
 if (vis[x] == tk) return x;  
 vis[x] = tk;  
 x = Find(pre[match[x]]);  
 }  
 }  
};

```

    }
}
void Blossom(int x, int y, int l) {
    for (; Find(x) != l; x = pre[y]) {
        pre[x] = y, y = match[x];
        if (s[y] == 1) q.push(y), s[y] = 0;
        for (int z: {x, y}) if (fa[z] == z) fa[z] = 1;
    }
}
bool Bfs(int r) {
    iota(iter(fa), 0); fill(iter(s), -1);
    q = queue<int>(); q.push(r); s[r] = 0;
    for (; !q.empty(); q.pop()) {
        for (int x = q.front(); int u : g[x])
            if (s[u] == -1) {
                if (pre[u] = x, s[u] = 1, match[u] == n) {
                    for (int a = u, b = x, last;
                        b != n; a = last, b = pre[a])
                        last = match[b], match[b] = a, match[a] = b;
                    return true;
                }
                q.push(match[u]); s[match[u]] = 0;
            }
        else if (!s[u] && Find(u) != Find(x)) {
            int l = LCA(u, x);
            Blossom(x, u, l); Blossom(u, x, l);
        }
    }
    return false;
}
Matching(int _n) : n(_n), fa(n + 1), s(n + 1), vis(n + 1), pre(n + 1, n), match(n + 1, n), g(n) {}
void add_edge(int u, int v) {
    g[u].pb(v), g[v].pb(u);
}
int solve() {
    int ans = 0;
    for (int x = 0; x < n; ++x)
        if (match[x] == n) ans += Bfs(x);
    return ans;
} // match[x] == n means not matched
};

```

### 3.10 Stable Marriage

- 1: Initialize  $m \in M$  and  $w \in W$  to free
- 2: **while**  $\exists$  free man  $m$  who has a woman  $w$  to propose to **do**
- 3:      $w \leftarrow$  first woman on  $m$ 's list to whom  $m$  has not yet proposed
- 4:     **if**  $\exists$  some pair  $(m', w)$  **then**
- 5:         **if**  $w$  prefers  $m$  to  $m'$  **then**
- 6:              $m' \leftarrow$  free
- 7:              $(m, w) \leftarrow$  engaged
- 8:         **end if**
- 9:     **else**
- 10:          $(m, w) \leftarrow$  engaged
- 11:     **end if**
- 12: **end while**

### 3.11 Flow Model

- Maximum/Minimum flow with lower bound / Circulation problem
  1. Construct super source  $S$  and sink  $T$ .
  2. For each edge  $(x, y, l, u)$ , connect  $x \rightarrow y$  with capacity  $u - l$ .
  3. For each vertex  $v$ , denote by  $in(v)$  the difference between the sum of incoming lower bounds and the sum of outgoing lower bounds.
  4. If  $in(v) > 0$ , connect  $S \rightarrow v$  with capacity  $in(v)$ , otherwise, connect  $v \rightarrow T$  with capacity  $-in(v)$ .
    - To maximize, connect  $t \rightarrow s$  with capacity  $\infty$  (skip this in circulation problem), and let  $f$  be the maximum flow from  $S$  to  $T$ . If  $f \neq \sum_{v \in V, in(v) > 0} in(v)$ , there's no solution. Otherwise, the maximum flow from  $s$  to  $t$  is the answer.
    - To minimize, let  $f$  be the maximum flow from  $S$  to  $T$ . Connect  $t \rightarrow s$  with capacity  $\infty$  and let the flow from  $S$  to  $T$  be  $f'$ . If  $f + f' \neq \sum_{v \in V, in(v) > 0} in(v)$ , there's no solution. Otherwise,  $f'$  is the answer.
  5. The solution of each edge  $e$  is  $l_e + f_e$ , where  $f_e$  corresponds to the flow of edge  $e$  on the graph.
- Construct minimum vertex cover from maximum matching  $M$  on bipartite graph  $(X, Y)$ 
  1. Redirect every edge:  $y \rightarrow x$  if  $(x, y) \in M$ ,  $x \rightarrow y$  otherwise.
  2. DFS from unmatched vertices in  $X$ .
  3.  $x \in X$  is chosen iff  $x$  is unvisited.
  4.  $y \in Y$  is chosen iff  $y$  is visited.
- Minimum cost cyclic flow
  1. Construct super source  $S$  and sink  $T$

2. For each edge  $(x, y, c)$ , connect  $x \rightarrow y$  with  $(cost, cap) = (c, 1)$  if  $c > 0$ , otherwise connect  $y \rightarrow x$  with  $(cost, cap) = (-c, 1)$
3. For each edge with  $c < 0$ , sum these cost as  $K$ , then increase  $d(y)$  by 1, decrease  $d(x)$  by 1
4. For each vertex  $v$  with  $d(v) > 0$ , connect  $S \rightarrow v$  with  $(cost, cap) = (0, d(v))$
5. For each vertex  $v$  with  $d(v) < 0$ , connect  $v \rightarrow T$  with  $(cost, cap) = (0, -d(v))$
6. Flow from  $S$  to  $T$ , the answer is the cost of the flow  $C + K$
- Maximum density induced subgraph
  1. Binary search on answer, suppose we're checking answer  $T$
  2. Construct a max flow model, let  $K$  be the sum of all weights
  3. Connect source  $s \rightarrow v$ ,  $v \in G$  with capacity  $K$
  4. For each edge  $(u, v, w)$  in  $G$ , connect  $u \rightarrow v$  and  $v \rightarrow u$  with capacity  $w$
  5. For  $v \in G$ , connect it with sink  $v \rightarrow t$  with capacity  $K + 2T - (\sum_{e \in E(v)} w(e)) - 2w(v)$
  6.  $T$  is a valid answer if the maximum flow  $f < K|V|$
- Minimum weight edge cover
  1. Let  $w'(u, v) = w(u, v) - \mu(u) - \mu(v)$ , where  $\mu(v)$  is the cost of the cheapest edge incident to  $v$ .
  2. Find the minimum weight matching  $M$  with  $w'$ . The answer is  $\sum \mu(v) + w'(M)$ .
- Project selection problem
  1. If  $p_v > 0$ , create edge  $(s, v)$  with capacity  $p_v$ ; otherwise, create edge  $(v, t)$  with capacity  $-p_v$ .
  2. Create edge  $(u, v)$  with capacity  $w$  with  $w$  being the cost of choosing  $u$  without choosing  $v$ .
  3. The mincut is equivalent to the maximum profit of a subset of projects.
- Dual of minimum cost maximum flow
  1. Capacity  $c_{uv}$ , Flow  $f_{uv}$ , Cost  $w_{uv}$ , Required Flow difference for vertex  $b_u$ .
  2. If all  $w_{uv}$  are integers, then optimal solution can happen when all  $p_u$  are integers.

$$\min \sum_{uv} w_{uv} f_{uv} \quad \min \sum_u b_u p_u + \sum_{uv} c_{uv} \max(0, p_v - p_u - w_{uv})$$

$$-f_{uv} \geq -c_{uv} \Leftrightarrow \sum_v f_{vu} - \sum_v f_{uv} = -b_u \quad p_u \geq 0$$

## 4 Geometry

### 4.1 Geometry Template [86f0f1]

```

using ld = ll;
using pdd = pair<ld, ld>;
#define X first
#define Y second
// ld eps = 1e-7;

pdd operator+(pdd a, pdd b)
{ return {a.X + b.X, a.Y + b.Y}; }
pdd operator-(pdd a, pdd b)
{ return {a.X - b.X, a.Y - b.Y}; }
pdd operator*(ld i, pdd v)
{ return {i * v.X, i * v.Y}; }
pdd operator*(pdd v, ld i)
{ return {i * v.X, i * v.Y}; }
pdd operator/(pdd v, ld i)
{ return {v.X / i, v.Y / i}; }
ld dot(pdd a, pdd b)
{ return a.X * b.X + a.Y * b.Y; }
ld cross(pdd a, pdd b)
{ return a.X * b.Y - a.Y * b.X; }
ld abs2(pdd v)
{ return v.X * v.X + v.Y * v.Y; }
ld abs(pdd v)
{ return sqrt(abs2(v)); }
int sgn(ld v)
{ return v > 0 ? 1 : (v < 0 ? -1 : 0); }
// int sgn(ld v){ return v > eps ? 1 : (v < -eps ? -1 : 0); }
int ori(pdd a, pdd b, pdd c)
{ return sgn(cross(b - a, c - a)); }
bool collinearity(pdd a, pdd b, pdd c)
{ return ori(a, b, c) == 0; }
bool btw(pdd p, pdd a, pdd b)
{ return collinearity(p, a, b) && sgn(dot(a - p, b - p))
    <= 0; }

bool seg_intersect(pdd p1, pdd p2, pdd p3, pdd p4){
    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;
}
pdd intersect(pdd p1, pdd p2, pdd p3, pdd p4){
    ld a123 = cross(p2 - p1, p3 - p1);
    ld a124 = cross(p2 - p1, p4 - p1);
    return (p4 * a123 - p3 * a124) / (a123 - a124);
}
pdd perp(pdd p1)
{ return pdd(-p1.Y, p1.X); }
pdd projection(pdd p1, pdd p2, pdd p3)
{ return p1 + (p2 - p1) * dot(p3 - p1, p2 - p1) / abs2(
    p2 - p1); }
pdd reflection(pdd p1, pdd p2, pdd p3)
{ return p3 + perp(p2 - p1) * cross(p3 - p1, p2 - p1) /
    abs2(p2 - p1) * 2; }
pdd linearTransformation(pdd p0, pdd p1, pdd q0, pdd q1
    , pdd r) {
    pdd dp = p1 - p0, dq = q1 - q0, num(cross(dp, dq),
        dot(dp, dq));
    return q0 + pdd(cross(r - p0, num), dot(r - p0, num))
        / abs2(dp);
} // from line p0--p1 to q0--q1, apply to r

```

## 4.2 Polar Angle Comparator [808e89]

```

// -1: a // b (if same), 0/1: a < b
int cmp(p1 a, p1 b, bool same = true){
#define is_neg(k) (sgn(k.Y) < 0 || (sgn(k.Y) == 0 &&
    sgn(k.X) < 0))
    int A = is_neg(a), B = is_neg(b);
    if(A != B)
        return A < B;
    if(sgn(cross(a, b)) == 0)
        return same ? abs2(a) < abs2(b) : -1;
    return sgn(cross(a, b)) > 0;
}

```

## 4.3 Minkowski Sum [98abff]

```

void reorder_poly(vector<pdd>& pnts){
    int mn = 0;
    for(int i = 1; i < (int)pnts.size(); i++){
        if(pnts[i].Y < pnts[mn].Y || (pnts[i].Y == pnts[mn].Y &&
            pnts[i].X < pnts[mn].X))
            mn = i;
    }
    rotate(pnts.begin(), pnts.begin() + mn, pnts.end());
}

vector<pdd> minkowski(vector<pdd> P, vector<pdd> Q){
    reorder_poly(P);
    reorder_poly(Q);
    int psz = P.size();
    int qsz = Q.size();
    P.pb(P[0]); P.pb(P[1]); Q.pb(Q[0]); Q.pb(Q[1]);
    vector<pdd> ans;
    int i = 0, j = 0;
    while(i < psz || j < qsz){
        ans.pb(P[i] + Q[j]);
        int t = sgn(cross(P[i + 1] - P[i], Q[j + 1] - Q[j]));
        if(t >= 0) i++;
        if(t <= 0) j++;
    }
    return ans;
}

```

## 4.4 Intersection of Circle and Convex Polygon [63653d]

```

double _area(pdd pa, pdd pb, double r){
    if(abs(pa)<abs(pb)) swap(pa, pb);
    if(abs(pb)<eps) return 0;
    double S, h, theta;
    double a=abs(pb), b=abs(pa), c=abs(pb-pa);
    double cosB = dot(pb, pb-pa) / a / c, B = acos(cosB);
    double cosC = dot(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 = .5*a*r*sin(theta) + (C-theta)/2*r*r;
}
else S = .5*sin(C)*a*b;
return S;
}
double areaPolyCircle(const vector<pdd> poly, const pdd
    &O, const double r){
    double S=0;
    for(int i=0; i<SZ(poly); ++i)
        S+=_area(poly[i]-O, poly[(i+1)%SZ(poly)]-O, r)*ori(0,
            poly[i], poly[(i+1)%SZ(poly)]);
    return fabs(S);
}

```

## 4.5 Intersection of Circles [f7a2fe]

```

bool CCinter(Cir &a, Cir &b, pdd &p1, pdd &p2) {
    pdd o1 = a.O, o2 = b.O;
    double r1 = a.R, r2 = b.R, d2 = abs2(o1 - o2), d =
        sqrt(d2);
    if(d < max(r1, r2) - min(r1, r2) || d > r1 + r2)
        return 0;
    pdd 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));
    pdd v = pdd(o1.Y - o2.Y, -o1.X + o2.X) * A / (2 * d2);
    p1 = u + v, p2 = u - v;
    return 1;
}

```

## 4.6 Tangent Line of Circles [c51d90]

```

vector<Line> CCTang(const Cir& c1, const Cir& c2,
    int sign1){
    vector<Line> ret;
    double d_sq = abs2(c1.O - c2.O);
    if (sgn(d_sq) == 0) return ret;
    double d = sqrt(d_sq);
    pdd v = (c2.O - c1.O) / d;
    double c = (c1.R - sign1 * c2.R) / d; // cos t
    if (c * c > 1) return ret;
    double h = sqrt(max(0.0, 1.0 - c * c)); // sin t
    for (int sign2 = 1; sign2 >= -1; sign2 -= 2) {
        pdd n = pdd(v.X * c - sign2 * h * v.Y,
            v.Y * c + sign2 * h * v.X);
        pdd p1 = c1.O + n * c1.R;
        pdd p2 = c2.O + n * (c2.R * sign1);
        if (sgn(p1.X - p2.X) == 0 and
            sgn(p1.Y - p2.Y) == 0)
            p2 = p1 + perp(c2.O - c1.O);
        ret.pb(Line(p1, p2));
    }
    return ret;
}

```

## 4.7 Intersection of Line and Convex Polygon [157258]

```

int TangentDir(vector<p1> &C, p1 dir) {
    return cyc_tsearch(SZ(C), [&](int a, int b) {
        return cross(dir, C[a]) > cross(dir, C[b]);
    });
}
#define cml(i) sign(cross(C[i] - a, b - a))
pii lineHull(p1 a, p1 b, vector<p1> &C) {
    int A = TangentDir(C, a - b);
    int B = TangentDir(C, b - a);
    int n = SZ(C);
    if (cml(A) < 0 || cml(B) > 0)
        return pii(-1, -1); // no collision
    auto gao = [&](int l, int r) {
        for (int t = l; (l + 1) % n != r; ) {
            int m = ((l + r + (l < r ? 0 : n)) / 2) % n;
            (cml(m) == cml(t) ? l : r) = m;
        }
        return (l + !cml(r)) % n;
    };
    pii res = pii(gao(B, A), gao(A, B)); // (i, j)
}

```



```

if (res.X == res.Y) // touching the corner i
    return pii(res.X, -1);
if (!cmpL(res.X) && !cmpL(res.Y)) // along side i, i
+1
    switch ((res.X - res.Y + n + 1) % n) {
        case 0: return pii(res.X, res.X);
        case 2: return pii(res.Y, res.Y);
    }
/* crossing sides (i, i+1) and (j, j+1)
crossing corner i is treated as side (i, i+1)
returned in the same order as the line hits the
convex */
return res;
} // convex cut: (r, l]

```

## 4.8 Intersection of Line and Circle [9183db]

```

vector<pdd> circleLineIntersection(pdd c, double r, pdd
a, pdd b) {
    pdd p = a + (b - a) * dot(c - a, b - a) / abs2(b - a)
    ;
    double s = cross(b - a, c - a), h2 = r * r - s * s /
    abs2(b - a);
    if (sgn(h2) < 0) return {};
    if (sgn(h2) == 0) return {p};
    pdd h = (b - a) / abs(b - a) * sqrt(h2);
    return {p - h, p + h};
}

```

## 4.9 Point in Circle [ecf954]

```

// return q's relation with circumcircle of tri(p[0],p
[1],p[2])
bool in_cc(const array<p11, 3> &p, p11 q) {
    __int128 det = 0;
    for (int i = 0; i < 3; ++i)
        det += __int128(abs2(p[i]) - abs2(q)) * cross(p[(i
+ 1) % 3] - q, p[(i + 2) % 3] - q);
    return det > 0; // in: >0, on: =0, out: <0
}

```

## 4.10 Point in Convex [f86640]

```

bool PointInConvex(const vector<p11> &C, p11 p, bool
strict = true) {
    int a = 1, b = SZ(C) - 1, r = !strict;
    if (SZ(C) == 0) return false;
    if (SZ(C) < 3) return r && btw(C[0], C.back(), p);
    if (ori(C[0], C[a], C[b]) > 0) swap(a, b);
    if (ori(C[0], C[a], p) >= r || ori(C[0], C[b], p) <=
    -r)
        return false;
    while (abs(a - b) > 1) {
        int c = (a + b) / 2;
        (ori(C[0], C[c], p) > 0 ? b : a) = c;
    }
    return ori(C[a], C[b], p) < r;
}

```

## 4.11 Half Plane Intersection [dfb833]

```

// from 8BQube
p11 area_pair(Line a, Line b)
{ return p11(cross(a.Y - a.X, b.X - a.X), cross(a.Y - a
.X, b.Y - a.X)); }
bool isin(Line l0, Line l1, Line l2) {
    // Check inter(l1, l2) strictly in l0
    auto [a02X, a02Y] = area_pair(l0, l2);
    auto [a12X, a12Y] = area_pair(l1, l2);
    if (a12X - a12Y < 0) a12X *= -1, a12Y *= -1;
    return (__int128) a02Y * a12X - (__int128) a02X *
    a12Y > 0; // C^4
}
/* Having solution, check size > 2 */
/* --- Line.X --- Line.Y --- */
vector<Line> halfPlaneInter(vector<Line> arr) {
    sort(iter(arr), [&](Line a, Line b) -> int {
        if (cmp(a.Y - a.X, b.Y - b.X, 0) != -1)
            return cmp(a.Y - a.X, b.Y - b.X, 0);
        return ori(a.X, a.Y, b.Y) < 0;
    });
    deque<Line> dq(1, arr[0]);
    for (auto p : arr) {

```

```

        if (cmp(dq.back().Y - dq.back().X, p.Y - p.X, 0) ==
        -1)
            continue;
        while (SZ(dq) >= 2 && !isin(p, dq[SZ(dq) - 2], dq.
        back()))
            dq.pop_back();
        while (SZ(dq) >= 2 && !isin(p, dq[0], dq[1]))
            dq.pop_front();
        dq.pb(p);
    }
    while (SZ(dq) >= 3 && !isin(dq[0], dq[SZ(dq) - 2], dq
    .back()))
        dq.pop_back();
    while (SZ(dq) >= 3 && !isin(dq.back(), dq[0], dq[1]))
        dq.pop_front();
    return vector<Line>(iter(dq));
}

```

## 4.12 Minimum Enclosing Circle [5af6d5]

```

using ld = long double;
pair<pdd, ld> circumcenter(pdd a, pdd b, pdd c);
pair<pdd, ld> MinimumEnclosingCircle(vector<pdd> &pts){
    random_shuffle(iter(pts));
    pdd c = pts[0];
    ld r = 0;
    for(int i = 1; i < SZ(pts); i++){
        if(abs(pts[i] - c) <= r) continue;
        c = pts[i]; r = 0;
        for(int j = 0; j < i; j++){
            if(abs(pts[j] - c) <= r) continue;
            c = (pts[i] + pts[j]) / 2;
            r = abs(pts[i] - c);
            for(int k = 0; k < j; k++){
                if(abs(pts[k] - c) > r)
                    tie(c, r) = circumcenter(pts[i], pts[j], pts[
                    k]);
            }
        }
    }
    return {c, r};
}

```

## 4.13 3D Point [badbbd]

```

// Copy from 8BQube
struct Point {
    double x, y, z;
    Point(double _x = 0, double _y = 0, double _z = 0): x
    (_x), y(_y), z(_z){}
    Point(pdd p) { x = p.X, y = p.Y, z = abs2(p); }
};
Point operator-(Point p1, Point p2)
{ return Point(p1.x - p2.x, p1.y - p2.y, p1.z - p2.z);
}
Point operator+(Point p1, Point p2)
{ return Point(p1.x + p2.x, p1.y + p2.y, p1.z + p2.z);
}
Point operator*(Point p1, double v)
{ return Point(p1.x * v, p1.y * v, p1.z * v); }
Point operator/(Point p1, double v)
{ return Point(p1.x / v, p1.y / v, p1.z / v); }
Point cross(Point p1, Point p2)
{ return Point(p1.y * p2.z - p1.z * p2.y, p1.z * p2.x -
    p1.x * p2.z, p1.x * p2.y - p1.y * p2.x); }
double dot(Point p1, Point p2)
{ return p1.x * p2.x + p1.y * p2.y + p1.z * p2.z; }
double abs(Point a)
{ return sqrt(dot(a, a)); }
Point cross3(Point a, Point b, Point c)
{ return cross(b - a, c - a); }
double area(Point a, Point b, Point c)
{ return abs(cross3(a, b, c)); }
double volume(Point a, Point b, Point c, Point d)
{ return dot(cross3(a, b, c), d - a); }
//Azimuthal angle (Longitude) to x-axis in interval [-
pi, pi]
double phi(Point p) { return atan2(p.y, p.x); }
//Zenith angle (Latitude) to the z-axis in interval [0,
pi]
double theta(Point p) { return atan2(sqrt(p.x * p.x + p
.y * p.y), p.z); }
Point masscenter(Point a, Point b, Point c, Point d)

```

```

{ return (a + b + c + d) / 4; }
pdd proj(Point a, Point b, Point c, Point u) {
// proj. u to the plane of a, b, and c
Point e1 = b - a;
Point e2 = c - a;
e1 = e1 / abs(e1);
e2 = e2 - e1 * dot(e2, e1);
e2 = e2 / abs(e2);
Point p = u - a;
return pdd(dot(p, e1), dot(p, e2));
}
Point rotate_around(Point p, double angle, Point axis)
{
double s = sin(angle), c = cos(angle);
Point u = axis / abs(axis);
return u * dot(u, p) * (1 - c) + p * c + cross(u, p)
    * s;
}

```

#### 4.14 ConvexHull3D [156311]

```

struct convex_hull_3D {
struct Face {
int a, b, c;
Face(int ta, int tb, int tc): a(ta), b(tb), c(tc) {}
}; // return the faces with pt indexes
vector<Face> res;
vector<Point> P;
convex_hull_3D(const vector<Point> &P): res(), P(_P) {
// all points coplanar case will WA, O(n^2)
int n = SZ(P);
if (n <= 2) return; // be careful about edge case
// ensure first 4 points are not coplanar
swap(P[1], *find_if(iter(P), [&](auto p) { return sgn
    (abs2(P[0] - p)) != 0; }));
swap(P[2], *find_if(iter(P), [&](auto p) { return sgn
    (abs2(cross3(p, P[0], P[1]))) != 0; }));
swap(P[3], *find_if(iter(P), [&](auto p) { return sgn
    (volume(P[0], P[1], P[2], p)) != 0; }));
vector<vector<int>> flag(n, vector<int>(n));
res.emplace_back(0, 1, 2); res.emplace_back(2, 1, 0);
for (int i = 3; i < n; ++i) {
vector<Face> next;
for (auto f : res) {
int d = sgn(volume(P[f.a], P[f.b], P[f.c], P[i]));
;
if (d <= 0) next.pb(f);
int ff = (d > 0) - (d < 0);
flag[f.a][f.b] = flag[f.b][f.c] = flag[f.c][f.a]
    = ff;
}
for (auto f : res) {
auto F = [&](int x, int y) {
if (flag[x][y] > 0 && flag[y][x] <= 0)
next.emplace_back(x, y, i);
};
F(f.a, f.b); F(f.b, f.c); F(f.c, f.a);
}
res = next;
}
}
bool same(Face s, Face t) {
if (sgn(volume(P[s.a], P[s.b], P[s.c], P[t.a])) != 0)
return 0;
if (sgn(volume(P[s.a], P[s.b], P[s.c], P[t.b])) != 0)
return 0;
if (sgn(volume(P[s.a], P[s.b], P[s.c], P[t.c])) != 0)
return 0;
return 1;
}
int polygon_face_num() {
int ans = 0;
for (int i = 0; i < SZ(res); ++i)
ans += none_of(res.begin(), res.begin() + i, [&](
    Face g) { return same(res[i], g); });
return ans;
}
double get_volume() {
double ans = 0;
for (auto f : res)
ans += volume(Point(0, 0, 0), P[f.a], P[f.b], P[f.c]);
return fabs(ans / 6);
}

```

```

}
double get_dis(Point p, Face f) {
Point p1 = P[f.a], p2 = P[f.b], p3 = P[f.c];
double a = (p2.y - p1.y) * (p3.z - p1.z) - (p2.z - p1
    .z) * (p3.y - p1.y);
double b = (p2.z - p1.z) * (p3.x - p1.x) - (p2.x - p1
    .x) * (p3.z - p1.z);
double c = (p2.x - p1.x) * (p3.y - p1.y) - (p2.y - p1
    .y) * (p3.x - p1.x);
double d = 0 - (a * p1.x + b * p1.y + c * p1.z);
return fabs(a * p.x + b * p.y + c * p.z + d) / sqrt(a
    * a + b * b + c * c);
}
};
// n^2 delaunay: facets with negative z normal of
// convexhull of (x, y, x^2 + y^2), use a pseudo-point
// (0, 0, inf) to avoid degenerate case

```

#### 4.15 Delaunay Triangulation [982e64]

```

/* Delaunay Triangulation:
Given a sets of points on 2D plane, find a
triangulation such that no points will strictly
inside circumcircle of any triangle. */
struct Edge {
int id; // oidx[id]
list<Edge>::iterator twin;
Edge(int _id = 0): id(_id) {}
};
struct Delaunay { // 0-base
int n, oidx[N];
list<Edge> head[N]; // result udir. graph
p11 p[N];
void init(int _n, p11 _p[]) {
n = _n, iota(oidx, oidx + n, 0);
for (int i = 0; i < n; ++i) head[i].clear();
sort(oidx, oidx + n, [&](int a, int b)
{ return _p[a] < _p[b]; });
for (int i = 0; i < n; ++i) p[i] = _p[oidx[i]];
divide(0, n - 1);
}
void addEdge(int u, int v) {
head[u].push_front(Edge(v));
head[v].push_front(Edge(u));
head[u].begin()->twin = head[v].begin();
head[v].begin()->twin = head[u].begin();
}
void divide(int l, int r) {
if (l == r) return;
if (l + 1 == r) return addEdge(l, l + 1);
int mid = (l + r) >> 1, nw[2] = {l, r};
divide(l, mid), divide(mid + 1, r);
auto gao = [&](int t) {
p11 pt[2] = {p[nw[0]], p[nw[1]]};
for (auto it : head[nw[t]]) {
int v = ori(pt[1], pt[0], p[it.id]);
if (v > 0 || (v == 0 && abs2(pt[1] - pt[0]) - p[it.
    id]) < abs2(pt[1] - pt[0]))
return nw[t] = it.id, true;
}
return false;
};
while (gao(0) || gao(1));
addEdge(nw[0], nw[1]); // add tangent
while (true) {
p11 pt[2] = {p[nw[0]], p[nw[1]]};
int ch = -1, sd = 0;
for (int t = 0; t < 2; ++t)
for (auto it : head[nw[t]])
if (ori(pt[0], pt[1], p[it.id]) > 0 && (
    ch == -1 || in_cc({pt[0], pt[1], p[ch]
    }, p[it.id])))
ch = it.id, sd = t;
if (ch == -1) break; // upper common tangent
for (auto it = head[nw[sd]].begin(); it != head[
    nw[sd]].end(); )
if (seg_strict_intersect(pt[sd], p[it->id], pt[
    sd ^ 1], p[ch]))
head[it->id].erase(it->twin), head[nw[sd]].
    erase(it++);
else ++it;
nw[sd] = ch, addEdge(nw[0], nw[1]);
}
}

```

```

}
} tool;

```

#### 4.16 Voronoi Diagram [da0c5e]

```

// all coord. is even, you may want to call
// halfPlaneInter after then
vector<vector<Line>> vec;
void build_voronoi_line(int n, pll *arr) {
    tool.init(n, arr); // Delaunay
    vec.clear(), vec.resize(n);
    for (int i = 0; i < n; ++i)
        for (auto e : tool.head[i]) {
            int u = tool.oidx[i], v = tool.oidx[e.id];
            pll m = (arr[v] + arr[u]) / 2LL, d = perp(arr[v]
                - arr[u]);
            vec[u].pb(Line(m, m + d));
        }
}

```

#### 4.17 Polygon Union [9fbf66]

```

// from 8BQube
ld rat(pll a, pll b) {
    return sgn(b.X) ? (ld)a.X / b.X : (ld)a.Y / b.Y;
} // all poly. should be ccw
ld polyUnion(vector<vector<pll>> &poly) {
    ld res = 0;
    for (auto &p : poly)
        for (int a = 0; a < SZ(p); ++a) {
            pll A = p[a], B = p[(a + 1) % SZ(p)];
            vector<pair<ld, int>> segs = {{0, 0}, {1, 0}};
            for (auto &q : poly) {
                if (&p == &q) continue;
                for (int b = 0; b < SZ(q); ++b) {
                    pll C = q[b], D = q[(b + 1) % SZ(q)];
                    int sc = ori(A, B, C), sd = ori(A, B, D);
                    if (sc != sd && min(sc, sd) < 0) {
                        ld sa = cross(D - C, A - C), sb = cross(D -
                            C, B - C);
                        segs.pb(sa / (sa - sb), sgn(sc - sd));
                    }
                    if (!sc && !sd && &q < &p && sgn(dot(B - A, D
                        - C)) > 0) {
                        segs.pb(rat(C - A, B - A), 1);
                        segs.pb(rat(D - A, B - A), -1);
                    }
                }
            }
            sort(iter(segs));
            for (auto &s : segs) s.X = clamp(s.X, 0.0, 1.0);
            ld sum = 0;
            int cnt = segs[0].second;
            for (int j = 1; j < SZ(segs); ++j) {
                if (!cnt) sum += segs[j].X - segs[j - 1].X;
                cnt += segs[j].Y;
            }
            res += cross(A, B) * sum;
        }
    return res / 2;
}

```

#### 4.18 Tangent Point to Convex Hull [523bc1]

```

// from 8BQube
/* The point should be strictly out of hull
   return arbitrary point on the tangent line */
pii get_tangent(vector<pll> &C, pll p) {
    auto gao = [&](int s) {
        return cyc_tsearch(SZ(C), [&](int x, int y)
            { return ori(p, C[x], C[y]) == s; });
    };
    return pii(gao(1), gao(-1));
} // return (a, b), ori(p, C[a], C[b]) >= 0

```

#### 4.19 Heart [082d19]

```

pdd circenter(pdd p0, pdd p1, pdd p2) { // radius = abs
    (center)
    p1 = p1 - p0, p2 = p2 - p0;
    double x1 = p1.X, y1 = p1.Y, x2 = p2.X, y2 = p2.Y;
    double m = 2. * (x1 * y2 - y1 * x2);
    pdd center;

```

```

    center.X = (x1 * x1 * y2 - x2 * x2 * y1 + y1 * y2 * (
        y1 - y2)) / m;
    center.Y = (x1 * x2 * (x2 - x1) - y1 * y1 * x2 + x1 *
        y2 * y2) / m;
    return center + p0;
}
pdd incenter(pdd p1, pdd p2, pdd p3) { // radius = area
    / s * 2
    double a = abs(p2 - p3), b = abs(p1 - p3), c = abs(p1
        - p2);
    double s = a + b + c;
    return (a * p1 + b * p2 + c * p3) / s;
}
pdd masscenter(pdd p1, pdd p2, pdd p3)
{ return (p1 + p2 + p3) / 3; }
pdd orthcenter(pdd p1, pdd p2, pdd p3)
{ return masscenter(p1, p2, p3) * 3 - circenter(p1, p2,
    p3) * 2; }

```

#### 4.20 Rotating Sweep Line [f5f689]

```

struct Event {
    pll d; int u, v;
    bool operator<(const Event &b) const {
        int ret = cmp(d, b.d, false);
        return ret == -1 ? false : ret; } // no tie-break
};
void rotatingSweepLine(const vector<pll> &p) {
    const int n = SZ(p);
    vector<Event> e; e.reserve(n * (n - 1));
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++) // pos[i] < pos[j] when
            the event occurs
            if (i != j) e.pb(p[j] - p[i], i, j);
    sort(iter(e));
    vector<int> ord(n), pos(n);
    iota(iter(ord), 0);
    sort(iter(ord), [&](int i, int j) { // initial order
        return p[i].Y != p[j].Y ? p[i].Y < p[j].Y : p[i].
            X < p[j].X; });
    for (int i = 0; i < n; i++) pos[ord[i]] = i;
    // initialize
    for (int i = 0, j = 0; i < SZ(e); i = j) {
        // do something
        vector<pii> tmp;
        for (; j < SZ(e) && !(e[i] < e[j]); j++)
            tmp.pb(pii(e[j].u, e[j].v));
        sort(iter(tmp), [&](pii x, pii y) {
            return pii(pos[x.ff], pos[x.ss]) < pii(pos[y.ff]
                , pos[y.ss]); });
        for (auto [x, y] : tmp) // pos[x] + 1 == pos[y]
            tie(ord[pos[x]], ord[pos[y]], pos[x], pos[y]) =
                make_tuple(ord[pos[y]], ord[pos[x]], pos[y],
                    pos[x]);
    }
}

```

#### 4.21 Vector In Poly [c6d0fa]

```

// ori(a, b, c) >= 0, valid: "strict" angle from a-b to
// a-c
bool btwangle(pll a, pll b, pll c, pll p, int strict) {
    return ori(a, b, p) >= strict && ori(a, p, c) >=
        strict;
}
// whether vector{cur, p} in counter-clockwise order
// prv, cur, nxt
bool inside(pll prv, pll cur, pll nxt, pll p, int
    strict) {
    if (ori(cur, nxt, prv) >= 0)
        return btwangle(cur, nxt, prv, p, strict);
    return !btwangle(cur, prv, nxt, p, !strict);
}

```

#### 4.22 Convex Hull DP [92fd4b]

```

sort(iter(pts), [&](pll x, pll y) {
    return x.Y != y.Y ? x.Y < y.Y : x.X < y.X;
});
auto getvec = [&](pii x) { return pts[x.ss] - pts[x.ff]
    };
// DP for convex hull vertices (no points on edges)
auto solve = [&](int bottom) { // O(n^3)

```

```

pll 0 = pts[bottom];
vector<pii> trans;
for (int j = bottom + 1; j < n; j++)
    for (int k = bottom + 1; k < n; k++) {
        if (ori(0, pts[j], pts[k]) <= 0) continue;
        // check whether j->k is legal
        trans.pb(pii(j, k));
    }
sort(iter(trans), [&](pii x, pii y) -> bool{
    int tmp = cmp(getvec(x), getvec(y), false);
    if (tmp != -1) return tmp;
    pll v = getvec(x);
    return dot(v, pts[x.ff]) > dot(v, pts[y.ff]);
});
// vector<LL> dp(n);
for (int j = bottom + 1; j < n; j++) {
    // check whether bottom -> j is legal
    // init trans -> j
}
for (auto [i, j] : trans) {
    // normal trans i -> j
}
for (int j = bottom + 1; j < n; j++) {
    // check whether j -> bottom is legal
    // end trans j ->
}
};
for (int i = 0; i < n; i++) solve(i);

```

## 4.23 Calculate Points in Triangle [bf746f]

```

// all points are distinct
// cnt[i][j] = # of point k s.t. strictly above ij, and
//           i < k < j
// cnt2[i][j] = # of points k s.t. strictly in ij
// preprocess space: O(n^2), time: O(n^3), query time:
//                 O(1)
vector cnt(n, vector<int>(n)), cnt2(n, vector<int>(n));
for (int i = 0; i < n; i++)
    for (int j = 0; j < n; j++) {
        if (pts[i] >= pts[j]) continue;
        for (int k = 0; k < n; k++) {
            if (pts[i] < pts[k] && pts[k] < pts[j]) {
                int tmp = ori(pts[i], pts[j], pts[k]);
                if (tmp > 0) cnt[i][j]++; // only for i < j
                else if (tmp == 0) cnt2[i][j]++, cnt2[j][i]++;
            }
        }
    }
auto calc_tri = [&](array<int, 3> arr) { // strictly
    inside
    sort(iter(arr), [&](int x, int y){ return pts[x] <
        pts[y]; });
    auto [x, y, z] = arr;
    int tmp = ori(pts[x], pts[y], pts[z]);
    if (tmp == 0) return 0;
    else if (tmp < 0)
        return cnt[x][z] - cnt[x][y] - cnt[y][z] - cnt2[x][
        y] - cnt2[y][z] - 1;
    else return cnt[x][y] + cnt[y][z] - cnt[x][z] - cnt2[
        x][z];
};

```

## 5 Graph

### 5.1 BCC [d04ebe]

```

struct BCC{ // 0-based, allow multi edges but not allow
    loops
    int n, m, cnt = 0;
    // n:|V|, m:|E|, cnt:#bcc
    // bcc i : vertices bcc_v[i] and edges bcc_e[i]
    vector<vector<int>> bcc_v, bcc_e;
    vector<vector<pii>> g; // original graph
    vector<pii> edges; // 0-based
    BCC(int _n, vector<pii> _edges):
        n(_n), m(SZ(_edges)), g(_n), edges(_edges){
        for (int i = 0; i < m; i++){
            auto [u, v] = edges[i];
            g[u].pb(pii(v, i)); g[v].pb(pii(u, i));
        }
    }
    void make_bcc(){ bcc_v.pb(); bcc_e.pb(); cnt++; }

```

```

// modify these if you need more information
void add_v(int v){ bcc_v.back().pb(v); }
void add_e(int e){ bcc_e.back().pb(e); }
void build(){
    vector<int> in(n, -1), low(n, -1), stk;
    vector<vector<int>> up(n);
    int ts = 0;
    auto _dfs = [&](auto dfs, int now, int par, int pe)
        -> void{
        if (pe != -1) up[now].pb(pe);
        in[now] = low[now] = ts++;
        stk.pb(now);
        for (auto [v, e] : g[now]){
            if (e == pe) continue;
            if (in[v] != -1){
                if (in[v] < in[now]) up[now].pb(e);
                low[now] = min(low[now], in[v]);
                continue;
            }
            dfs(dfs, v, now, e);
            low[now] = min(low[now], low[v]);
        }
        if ((now != par && low[now] >= in[par]) || (now ==
            par && SZ(g[now]) == 0)){
            make_bcc();
            for (int v = stk.back(); v = stk.back()){
                stk.pop_back(), add_v(v);
                for (int e : up[v]) add_e(e);
                if (v == now) break;
            }
            if (now != par) add_v(par);
        }
    };
    for (int i = 0; i < n; i++)
        if (in[i] == -1) _dfs(_dfs, i, i, -1);
}

```

### 5.2 SCC [2c9a01]

```

struct SCC{ // 0-based, output reversed topo order
    int n, cnt = 0;
    vector<vector<int>> g;
    vector<int> sccid;
    explicit SCC(int _n): n(_n), g(_n), sccid(_n, -1) {}
    void add_edge(int u, int v){
        g[u].pb(v);
    }
    void build(){
        vector<int> in(n, -1), low(n, -1), stk;
        vector<bool> instk(n);
        int ts = 0;
        auto dfs1 = [&](auto dfs, int now) -> void{
            stk.pb(now); instk[now] = true;
            in[now] = low[now] = ts++;
            for (int i : g[now]){
                if (in[i] == -1)
                    dfs(dfs, i), low[now] = min(low[now], low[i]);
                else if (instk[i] && in[i] < in[now])
                    low[now] = min(low[now], in[i]);
            }
            if (low[now] == in[now]){
                for (; stk.back() != now; stk.pop_back())
                    sccid[stk.back()] = cnt, instk[stk.back()] =
                    false;
                sccid[now] = cnt++, instk[now] = false, stk.
                pop_back();
            }
        };
        for (int i = 0; i < n; i++)
            if (in[i] == -1) dfs1(dfs1, i);
    }
};

```

### 5.3 2-SAT [0686a5]

```

struct SAT { // 0-based
    int n;
    vector<bool> istrue;
    SCC scc;
    SAT(int _n): n(_n), istrue(n + n), scc(n + n) {}
    int neg(int a) {

```

```

    return a >= n ? a - n : a + n;
}
void add_clause(int a, int b) {
    scc.add_edge(neg(a), b), scc.add_edge(neg(b), a);
}
bool solve() {
    scc.build();
    for (int i = 0; i < n; ++i) {
        if (scc.sccid[i] == scc.sccid[i + n]) return false;
        istrue[i] = scc.sccid[i] < scc.sccid[i + n];
        istrue[i + n] = !istrue[i];
    }
    return true;
}
};

```

## 5.4 Dominator Tree [2da9bb]

```

struct Dominator {
    int n;
    vector<vector<int>> g, r, rdom; int tk;
    vector<int> dfn, rev, fa, sdom, dom, val, rp;
    Dominator(int _n) : n(_n), g(n), r(n), rdom(n), tk(0)
    {
        dfn = rev = fa = sdom = dom =
            val = rp = vector<int>(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) {
        // return the father of each node in dominator tree
        dfs(s); // p[i] = -2 if i is unreachable, par[s] = -1
        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;
    }
};

```

## 5.5 Virtual Tree [f977f8]

```

// copy from 8BQube
vector<int> vG[N];
int top, st[N];
int vrt = -1;

void insert(int u) {
    if (top == -1) return st[++top] = vrt = u, void();
    int p = LCA(st[top], u);
    if (dep[vrt] > dep[p]) vrt = p;
    if (p == st[top]) return st[++top] = u, void();
    while (top >= 1 && dep[st[top - 1]] >= dep[p])
        vG[st[top - 1]].pb(st[top]), --top;
}

```

```

    if (st[top] != p)
        vG[p].pb(st[top]), --top, st[++top] = p;
    st[++top] = u;
}

void reset(int u) {
    for (int i : vG[u]) reset(i);
    vG[u].clear();
}

void solve(vector<int> &v) {
    top = -1;
    sort(ALL(v),
        [&](int a, int b) { return dfn[a] < dfn[b]; });
    for (int i : v) insert(i);
    while (top > 0) vG[st[top - 1]].pb(st[top]), --top;
    // do something
    reset(vrt);
}

```

## 5.6 Fast DMST [7b274d]

```

struct E { int s, t; ll w; }; // 0-base
struct PQ {
    struct P {
        ll v; int i;
        bool operator>(const P &b) const { return v > b.v; }
    };
    priority_queue<P, vector<P>, greater<>> pq; ll tag;
    // min heap
    void push(P p) { p.v -= tag; pq.emplace(p); }
    P top() { P p = pq.top(); p.v += tag; return p; }
    void join(PQ &b) {
        if (pq.size() < b.pq.size())
            swap(pq, b.pq), swap(tag, b.tag);
        while (!b.pq.empty()) push(b.top()), b.pq.pop();
    }
}; // O(E log^2 V), use Leftist tree for O(E log V)
vector<int> dmst(const vector<E> &e, int n, int root) {
    vector<PQ> h(n * 2);
    for (int i = 0; i < int(e.size()); ++i)
        h[e[i].t].push({e[i].w, i});
    vector<int> a(n * 2); iota(iter(a), 0);
    vector<int> v(n * 2, -1), pa(n * 2, -1), r(n * 2);
    auto o = [&](auto Y, int x) -> int {
        return x == a[x] ? x : a[x] = Y(Y, a[x]);
    };
    auto S = [&](int i) { return o(o, e[i].s); };
    int pc = v[root] = n;
    for (int i = 0; i < n; ++i) if (v[i] == -1)
        for (int p = i; v[p] < 0 || v[p] == i; p = S(r[p])) {
            if (v[p] == i)
                for (int q = pc++; p != q; p = S(r[p])) {
                    h[p].tag -= h[p].top().v; h[q].join(h[p]);
                    pa[p] = a[p] = q;
                }
            while (S(h[p].top().i) == p) h[p].pq.pop();
            v[p] = i; r[p] = h[p].top().i;
        }
    vector<int> ans;
    for (int i = pc - 1; i >= 0; i--) if (v[i] != n) {
        for (int f = e[r[i]].t; f != -1 && v[f] != n; f = pa[f])
            v[f] = n;
        ans.push_back(r[i]);
    }
    return ans; // default minimize, returns edgeid array
}

```

## 5.7 Vizing [f4ebad]

```

// find D+1 edge coloring of a graph with max deg D
struct vizing { // returns edge coloring in adjacent
    matrix G, 1 - based
    const int N = 105;
    int C[N][N], G[N][N], X[N], vst[N], n; // ans: G[i][j]
}

void init(int _n) { n = _n; // n = |V|+1
    for (int i = 0; i <= n; ++i)
        for (int j = 0; j <= n; ++j)
            C[i][j] = G[i][j] = 0;
}

void solve(vector<pii> &E) {
}

```



```

auto update = [&](int u)
{ for (X[u] = 1; C[u][X[u]]; ++X[u]); };
auto color = [&](int u, int v, int c) {
    int p = G[u][v];
    G[u][v] = G[v][u] = c;
    C[u][c] = v, C[v][c] = u;
    C[u][p] = C[v][p] = 0;
    if (p) X[u] = X[v] = p;
    else update(u), update(v);
    return p;
};
auto flip = [&](int u, int c1, int c2) {
    int p = C[u][c1];
    swap(C[u][c1], C[u][c2]);
    if (p) G[u][p] = G[p][u] = c2;
    if (!C[u][c1]) X[u] = c1;
    if (!C[u][c2]) X[u] = c2;
    return p;
};
fill_n(X + 1, n, 1);
for (int t = 0; t < SZ(E); ++t) {
    int u = E[t].X, v0 = E[t].Y, v = v0, c0 = X[u], c
        = c0, d;
    vector<pii> L;
    fill_n(vst + 1, n, 0);
    while (!G[u][v0]) {
        L.emplace_back(v, d = X[v]);
        if (!C[v][c]) for (int a = SZ(L) - 1; a >= 0;
            --a) c = color(u, L[a].X, c);
        else if (!C[u][d]) for (int a = SZ(L) - 1; a >=
            0; --a) color(u, L[a].X, L[a].Y);
        else if (vst[d]) break;
        else vst[d] = 1, v = C[u][d];
    }
    if (!G[u][v0]) {
        for (; v; v = flip(v, c, d), swap(c, d));
        if (int a; C[u][c0]) {
            for (a = SZ(L) - 2; a >= 0 && L[a].Y != c; --
                a);
            for (; a >= 0; --a) color(u, L[a].X, L[a].Y);
        }
        else --t;
    }
}
};

```

## 5.8 Maximum Clique [d50aa9]

```

struct MaxClique { // fast when N <= 100
    bitset<N> G[N], cs[N];
    int ans, sol[N], q, cur[N], d[N], n;
    void init(int _n) {
        n = _n;
        for (int i = 0; i < n; ++i) G[i].reset();
    }
    void add_edge(int u, int v) {
        G[u][v] = G[v][u] = 1;
    }
    void pre_dfs(vector<int> &r, int l, bitset<N> mask) {
        if (l < 4) {
            for (int i : r) d[i] = (G[i] & mask).count();
            sort(ALL(r), [&](int x, int y) { return d[x] > d[
                y]; });
        }
        vector<int> c(SZ(r));
        int lft = max(ans - q + 1, 1), rgt = 1, tp = 0;
        cs[1].reset(), cs[2].reset();
        for (int p : r) {
            int k = 1;
            while ((cs[k] & G[p]).any()) ++k;
            if (k > rgt) cs[++rgt + 1].reset();
            cs[k][p] = 1;
            if (k < lft) r[tp++] = p;
        }
        for (int k = lft; k <= rgt; ++k)
            for (int p = cs[k]._Find_first(); p < N; p = cs[k]
                ._Find_next(p))
                r[tp] = p, c[tp] = k, ++tp;
        dfs(r, c, l + 1, mask);
    }
    void dfs(vector<int> &r, vector<int> &c, int l,
        bitset<N> mask) {

```

```

        while (!r.empty()) {
            int p = r.back();
            r.pop_back(), mask[p] = 0;
            if (q + c.back() <= ans) return;
            cur[q++] = p;
            vector<int> nr;
            for (int i : r) if (G[p][i]) nr.pb(i);
            if (!nr.empty()) pre_dfs(nr, l, mask & G[p]);
            else if (q > ans) ans = q, copy_n(cur, q, sol);
            c.pop_back(), --q;
        }
    }
    int solve() {
        vector<int> r(n);
        ans = q = 0, iota(ALL(r), 0);
        pre_dfs(r, 0, bitset<N>(string(n, '1')));
        return ans;
    }
};

```

## 5.9 Number of Maximal Clique [11fa26]

```

struct BronKerbosch { // 1-base
    int n, a[N], g[N][N];
    int S, all[N][N], some[N][N], none[N][N];
    void init(int _n) {
        n = _n;
        for (int i = 1; i <= n; ++i)
            for (int j = 1; j <= n; ++j) g[i][j] = 0;
    }
    void add_edge(int u, int v) {
        g[u][v] = g[v][u] = 1;
    }
    void dfs(int d, int an, int sn, int nn) {
        if (S > 1000) return; // pruning
        if (sn == 0 && nn == 0) ++S;
        int u = some[d][0];
        for (int i = 0; i < sn; ++i) {
            int v = some[d][i];
            if (g[u][v]) continue;
            int tsu = 0, tnn = 0;
            copy_n(all[d], an, all[d + 1]);
            all[d + 1][an] = v;
            for (int j = 0; j < sn; ++j)
                if (g[v][some[d][j]])
                    some[d + 1][tsu++] = some[d][j];
            for (int j = 0; j < nn; ++j)
                if (g[v][none[d][j]])
                    none[d + 1][tnn++] = none[d][j];
            dfs(d + 1, an + 1, tsu, tnn);
            some[d][i] = 0, none[d][nn++] = v;
        }
    }
    int solve() {
        iota(some[0], some[0] + n, 1);
        S = 0, dfs(0, 0, n, 0);
        return S;
    }
};

```

## 5.10 Minimum Mean Cycle [3e5d2b]

```

// from 8BQube
ll road[N][N]; // input here
struct MinimumMeanCycle {
    ll dp[N + 5][N], n;
    pll solve() {
        ll a = -1, b = -1, L = n + 1;
        for (int i = 2; i <= L; ++i)
            for (int k = 0; k < n; ++k)
                for (int j = 0; j < n; ++j)
                    dp[i][j] =
                        min(dp[i - 1][k] + road[k][j], dp[i][j]);
        for (int i = 0; i < n; ++i) {
            if (dp[L][i] >= INF) continue;
            ll ta = 0, tb = 1;
            for (int j = 1; j < n; ++j)
                if (dp[j][i] < INF &&
                    ta * (L - j) < (dp[L][i] - dp[j][i]) * tb)
                    ta = dp[L][i] - dp[j][i], tb = L - j;
            if (ta == 0) continue;
            if (a == -1 || a * tb > ta * b) a = ta, b = tb;
        }
    }
};

```

```

if (a != -1) {
    ll g = __gcd(a, b);
    return pll(a / g, b / g);
}
return pll(-1LL, -1LL);
}
void init(int _n) {
    n = _n;
    for (int i = 0; i < n; ++i)
        for (int j = 0; j < n; ++j) dp[i + 2][j] = INF;
}
};

```

## 5.11 Minimum Steiner Tree [21acea]

```

// from 8BQube
// O(V 3^T + V^2 2^T)
struct SteinerTree { // 0-base
    static const int T = 10, N = 105, INF = 1e9;
    int n, dst[N][N], dp[1 << T][N], tdst[N];
    int vcost[N]; // the cost of vertexs
    void init(int _n) {
        n = _n;
        for (int i = 0; i < n; ++i) {
            for (int j = 0; j < n; ++j) dst[i][j] = INF;
            dst[i][i] = vcost[i] = 0;
        }
    }
    void add_edge(int ui, int vi, int wi) {
        dst[ui][vi] = min(dst[ui][vi], wi);
    }
    void shortest_path() {
        for (int k = 0; k < n; ++k)
            for (int i = 0; i < n; ++i)
                for (int j = 0; j < n; ++j)
                    dst[i][j] =
                        min(dst[i][j], dst[i][k] + dst[k][j]);
    }
    int solve(const vector<int> &ter) {
        shortest_path();
        int t = SZ(ter);
        for (int i = 0; i < (1 << t); ++i)
            for (int j = 0; j < n; ++j) dp[i][j] = INF;
        for (int i = 0; i < n; ++i) dp[0][i] = vcost[i];
        for (int msk = 1; msk < (1 << t); ++msk) {
            if (!(msk & (msk - 1))) {
                int who = __lg(msk);
                for (int i = 0; i < n; ++i)
                    dp[msk][i] =
                        vcost[ter[who]] + dst[ter[who]][i];
            }
            for (int i = 0; i < n; ++i)
                for (int submsk = (msk - 1) & msk; submsk;
                    submsk = (submsk - 1) & msk)
                    dp[msk][i] = min(dp[msk][i],
                        dp[submsk][i] + dp[msk ^ submsk][i] -
                        vcost[i]);
            for (int i = 0; i < n; ++i) {
                tdst[i] = INF;
                for (int j = 0; j < n; ++j)
                    tdst[i] =
                        min(tdst[i], dp[msk][j] + dst[j][i]);
            }
            for (int i = 0; i < n; ++i) dp[msk][i] = tdst[i];
        }
        int ans = INF;
        for (int i = 0; i < n; ++i)
            ans = min(ans, dp[(1 << t) - 1][i]);
        return ans;
    }
};

```

## 5.12 Count Cycles [c7e8f2]

```

// ord = sort by deg decreasing, rk[ord[i]] = i
// D[i] = edge point from rk small to rk big
for (int x : ord) { // c3
    for (int y : D[x]) vis[y] = 1;
    for (int y : D[x]) for (int z : D[y]) c3 += vis[z];
    for (int y : D[x]) vis[y] = 0;
}
for (int x : ord) { // c4
    for (int y : D[x]) for (int z : adj[y])

```

```

if (rk[z] > rk[x]) c4 += vis[z]++;
for (int y : D[x]) for (int z : adj[y])
    if (rk[z] > rk[x]) --vis[z];
} // both are O(M*sqrt(M))

```

## 6 Math

### 6.1 Extended Euclidean Algorithm [c51ae9]

```

// ax+ny = 1, ax+ny == ax == 1 (mod n)
void extgcd(ll x, ll y, ll &g, ll &a, ll &b) {
    if (y == 0) g = x, a = 1, b = 0;
    else extgcd(y, x % y, g, b, a), b -= (x / y) * a;
}

```

### 6.2 Floor & Ceil [134881]

```

ll ifloor(ll a, ll b) {
    return a / b - (a % b && (a < 0) ^ (b < 0));
}
ll iceil(ll a, ll b) {
    return a / b + (a % b && (a < 0) ^ (b > 0));
}

```

### 6.3 Legendre [4e4b23]

*// the Jacobi symbol is a generalization of the Legendre symbol,  
// such that the bottom doesn't need to be prime.  
// (n|p) -> same as Legendre  
// (n|ab) = (n|a)(n|b)  
// work with Long Long*

```

int Jacobi(int a, int m) {
    int s = 1;
    for (; m > 1; ) {
        a %= m;
        if (a == 0) return 0;
        const int r = __builtin_ctz(a);
        if ((r & 1) && ((m + 2) & 4)) s = -s;
        a >>= r;
        if (a & m & 2) s = -s;
        swap(a, m);
    }
    return s;
}

```

```

// 0: a == 0
// -1: a isn't a quad res of p
// else: return X with X^2 % p == a
// doesn't work with Long Long
int QuadraticResidue(int a, int p) {
    if (p == 2) return a & 1;
    if (int jc = Jacobi(a, p); jc <= 0) return jc;
    int b, d;
    for (; ; ) {
        b = rand() % p;
        d = (1LL * b * b + p - a) % p;
        if (Jacobi(d, p) == -1) break;
    }
    int f0 = b, f1 = 1, g0 = 1, g1 = 0, tmp;
    for (int e = (1LL + p) >> 1; e; e >>= 1) {
        if (e & 1) {
            tmp = (1LL * g0 * f0 + 1LL * d * (1LL * g1 * f1 % p)) % p;
            g1 = (1LL * g0 * f1 + 1LL * g1 * f0) % p;
            g0 = tmp;
        }
        tmp = (1LL * f0 * f0 + 1LL * d * (1LL * f1 * f1 % p)) % p;
        f1 = (2LL * f0 * f1) % p;
        f0 = tmp;
    }
    return g0;
}

```

### 6.4 Simplex [aa7741]

```

// maximize c^T x
// subject to Ax <= b, x >= 0
// and stores the solution;
typedef long double T; // Long double, Rational, double
                        + mod<P>...
typedef vector<T> vd;
typedef vector<vd> vvd;

```

```

const T eps = 1e-9, inf = 1/.0;
#define ltj(X) if(s == -1 || mp(X[j],N[j]) < mp(X[s],N[s])) s=j
#define rep(i, 1, n) for(int i = 1; i < n; i++)

struct LPSolver {
    int m, n;
    vector<int> N, B;
    vvd D;

    LPSolver(const vvd& A, const vd& b, const vd& c) :
        m(SZ(b)), n(SZ(c)), N(n+1), B(m), D(m+2, vd(n+2)) {
        rep(i,0,m) rep(j,0,n) D[i][j] = A[i][j];
        rep(i,0,m) { B[i] = n+i; D[i][n] = -1; D[i][n+1] = b[i]; }
        rep(j,0,n) { N[j] = j; D[m][j] = -c[j]; }
        N[n] = -1; D[m+1][n] = 1;
    }

    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;
            b[s] = a[s] * inv2;
        }
        rep(j,0,n+2) if (j != s) D[r][j] *= inv;
        rep(i,0,m+2) if (i != r) D[i][s] *= -inv;
        D[r][s] = inv;
        swap(B[r], N[s]);
    }

    bool simplex(int phase) {
        int x = m + phase - 1;
        for (;;) {
            int s = -1;
            rep(j,0,n+1) if (N[j] != -phase) ltj(D[x]);
            if (D[x][s] >= -eps) return true;
            int r = -1;
            rep(i,0,m) {
                if (D[i][s] <= eps) continue;
                if (r == -1 || mp(D[i][n+1] / D[i][s], B[i]) < mp(D[r][n+1] / D[r][s], B[r])) r = i;
            }
            if (r == -1) return false;
            pivot(r, s);
        }
    }

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

```

## 6.5 Simplex Construction

Standard form: maximize  $\sum_{1 \leq i \leq n} c_i x_i$  such that  $\sum_{1 \leq i \leq n} A_{ji} x_i \leq b_j$  for all  $1 \leq j \leq m$  and  $x_i \geq 0$  for all  $1 \leq i \leq n$ .

1. In case of minimization, let  $c'_i = -c_i$
2.  $\sum_{1 \leq i \leq n} A_{ji} x_i \geq b_j \rightarrow \sum_{1 \leq i \leq n} -A_{ji} x_i \leq -b_j$
3.  $\sum_{1 \leq i \leq n} A_{ji} x_i = b_j \rightarrow \text{add } \leq \text{ and } \geq$ .
4. If  $x_i$  has no lower bound, replace  $x_i$  with  $x_i - x'_i$

## 6.6 DiscreteLog [da27bf]

```

int DiscreteLog(int s, int x, int y, int m) {
    constexpr int kStep = 32000;
    unordered_map<int, int> p;

```

```

    int b = 1;
    for (int i = 0; i < kStep; ++i) {
        p[y] = i;
        y = 1LL * y * x % m;
        b = 1LL * b * x % m;
    }
    for (int i = 0; i < m + 10; i += kStep) {
        s = 1LL * s * b % m;
        if (p.find(s) != p.end()) return i + kStep - p[s];
    }
    return -1;
}

int DiscreteLog(int x, int y, int m) {
    if (m == 1) return 0;
    int s = 1;
    for (int i = 0; i < 100; ++i) {
        if (s == y) return i;
        s = 1LL * s * x % m;
    }
    if (s == y) return 100;
    int p = 100 + DiscreteLog(s, x, y, m);
    if (fpow(x, p, m) != y) return -1;
    return p; //returns: x^p = y (mod m)
}

```

## 6.7 Miller Rabin & Pollard Rho [d3ecd2]

```

// n < 4,759,123,141      3 : 2, 7, 61
// n < 1,122,004,669,633  4 : 2, 13, 23, 1662803
// n < 3,474,749,660,383  6 : primes <= 13
// n < 2^64              7 :
// 2, 325, 9375, 28178, 450775, 9780504, 1795265022
ll mul(ll a, ll b, ll n){
    return (__int128)a * b % n;
}

bool Miller_Rabin(ll a, ll n) {
    if ((a = a % n) == 0) return 1;
    if (n % 2 == 0) return n == 2;
    ll tmp = (n - 1) / ((n - 1) & (1 - n));
    ll t = __lg(((n - 1) & (1 - n))), x = 1;
    for (; tmp; tmp >>= 1, a = mul(a, a, n))
        if (tmp & 1) x = mul(x, a, n);
    if (x == 1 || x == n - 1) return 1;
    while (--t)
        if ((x = mul(x, x, n)) == n - 1) return 1;
    return 0;
}

bool prime(ll n){
    vector<ll> tmp = {2, 325, 9375, 28178, 450775,
                    9780504, 1795265022};
    for(ll i : tmp)
        if(!Miller_Rabin(i, n)) return false;
    return true;
}

map<ll, int> cnt;
void PollardRho(ll n) {
    if (n == 1) return;
    if (prime(n)) return ++cnt[n], void();
    if (n % 2 == 0) return PollardRho(n / 2), ++cnt[2], void();
    ll x = 2, y = 2, d = 1, p = 1;
#define f(x, n, p) ((mul(x, x, n) + p) % n)
    while (true) {
        if (d != n && d != 1) {
            PollardRho(n / d);
            PollardRho(d);
            return;
        }
        if (d == n) ++p;
        x = f(x, n, p), y = f(f(y, n, p), n, p);
        d = gcd(abs(x - y), n);
    }
}

```

## 6.8 XOR Basis [006505]

```

const int digit = 60; // [0, 2^digit)
struct Basis{
    int total = 0, rank = 0;
    vector<ll> b;
    Basis(): b(digit) {}
    bool add(ll v){ // Gauss Jordan Elimination
        total++;

```

```

for(int i = digit - 1; i >= 0; i--){
    if(!(1LL << i & v)) continue;
    if(b[i] != 0){
        v ^= b[i];
        continue;
    }
    for(int j = 0; j < i; j++){
        if(1LL << j & v) v ^= b[j];
    }
    for(int j = i + 1; j < digit; j++){
        if(1LL << i & b[j]) b[j] ^= v;
    }
    b[i] = v;
    rank++;
    return true;
}
return false;
}
ll getMax(ll x = 0){
    for(ll i : b) x = max(x, x ^ i);
    return x;
}
ll getMin(ll x = 0){
    for(ll i : b) x = min(x, x ^ i);
    return x;
}
bool can(ll x){
    return getMin(x) == 0;
}
ll kth(ll k){ // kth smallest, 0-indexed
    vector<ll> tmp;
    for(ll i : b) if(i) tmp.pb(i);
    ll ans = 0;
    for(int i = 0; i < SZ(tmp); i++){
        if(1LL << i & k) ans ^= tmp[i];
    }
    return ans;
}
};

```

## 6.9 Linear Equation [ab487b]

```

vector<int> RREF(vector<vector<ll>> &mat){
    int N = mat.size(), M = mat[0].size();
    int rk = 0;
    vector<int> cols;
    for (int i = 0; i < M; i++) {
        int cnt = -1;
        for (int j = N-1; j >= rk; j--){
            if(mat[j][i] != 0) cnt = j;
        }
        if(cnt == -1) continue;
        swap(mat[rk], mat[cnt]);
        ll lead = mat[rk][i];
        for (int j = 0; j < M; j++) mat[rk][j] = mat[rk][j] *
            modinv(lead) % mod;
        for (int j = 0; j < N; j++) {
            if(j == rk) continue;
            ll tmp = mat[j][i];
            for (int k = 0; k < M; k++)
                mat[j][k] = (mat[j][k] - mat[rk][k] * tmp % mod
                    + mod) % mod;
        }
        cols.pb(i);
        rk++;
    }
    return cols;
}
struct LinearEquation{
    bool ok;
    vector<ll> par; //particular solution (Ax = b)
    vector<vector<ll>> homo; //homogenous (Ax = 0)
    vector<vector<ll>> rref;
    //first M columns are matrix A
    //last column of eq is vector b
    void solve(const vector<vector<ll>> &eq){
        int M = (int)eq[0].size() - 1;
        rref = eq;
        auto piv = RREF(rref);
        int rk = piv.size();
        if(piv.size() && piv.back() == M){
            ok = 0; return;
        }
        ok = 1;
        par.resize(M);
        vector<bool> ispiv(M);
        for (int i = 0; i < rk; i++) {

```

```

            par[piv[i]] = rref[i][M];
            ispiv[piv[i]] = 1;
        }
        for (int i = 0; i < M; i++) {
            if (ispiv[i]) continue;
            vector<ll> h(M);
            h[i] = 1;
            for (int j = 0; j < rk; j++) h[piv[j]] = rref[j][i]
                ? mod-rref[j][i] : 0;
            homo.pb(h);
        }
    }
};

```

## 6.10 Chinese Remainder Theorem [6ef4a3]

```

ll solve_crt(ll x1, ll m1, ll x2, ll m2){
    ll g = gcd(m1, m2);
    if ((x2 - x1) % g) return {0, 0}; // no sol
    m1 /= g; m2 /= g;
    ll _, p, q;
    extgcd(m1, m2, _, p, q); // p <= C
    ll lcm = m1 * m2 * g;
    ll res = ((__int128)p * (x2 - x1) % lcm * m1 % lcm +
        x1) % lcm;
    // be careful with overflow, C^3
    return {(res + lcm) % lcm, lcm}; // (x, m)
}

```

## 6.11 Sqrt Decomposition [8d7bc0]

```

// for all i in [l, r], floor(n / i) = x
for(int l = 1, r; l <= n; l = r + 1){
    int x = ifloor(n, l);
    r = ifloor(n, x);
}
// for all i in [l, r], ceil(n / i) = x
for(int l = 1, r = n; r >= 1; r = l - 1){
    int x = iceil(n, r);
    l = iceil(n, x);
}

```

## 6.12 Floor Sum

- $m = \lfloor \frac{an+b}{c} \rfloor$
- Time complexity:  $O(\log n)$

$$f(a, b, c, n) = \sum_{i=0}^n \left\lfloor \frac{ai+b}{c} \right\rfloor$$

$$= \begin{cases} \left\lfloor \frac{a}{c} \right\rfloor \cdot \frac{n(n+1)}{2} + \left\lfloor \frac{b}{c} \right\rfloor \cdot (n+1) \\ + f(a \bmod c, b \bmod c, c, n), & a \geq c \vee b \geq c \\ 0, & n < 0 \vee a = 0 \\ nm - f(c, c-b-1, a, m-1), & \text{otherwise} \end{cases}$$

$$g(a, b, c, n) = \sum_{i=0}^n i \left\lfloor \frac{ai+b}{c} \right\rfloor$$

$$= \begin{cases} \left\lfloor \frac{a}{c} \right\rfloor \cdot \frac{n(n+1)(2n+1)}{6} + \left\lfloor \frac{b}{c} \right\rfloor \cdot \frac{n(n+1)}{2} \\ + g(a \bmod c, b \bmod c, c, n), & a \geq c \vee b \geq c \\ 0, & n < 0 \vee a = 0 \\ \frac{1}{2} \cdot (n(n+1)m - f(c, c-b-1, a, m-1)) \\ - h(c, c-b-1, a, m-1), & \text{otherwise} \end{cases}$$

$$h(a, b, c, n) = \sum_{i=0}^n \left\lfloor \frac{ai+b}{c} \right\rfloor^2$$

$$= \begin{cases} \left\lfloor \frac{a}{c} \right\rfloor^2 \cdot \frac{n(n+1)(2n+1)}{6} + \left\lfloor \frac{b}{c} \right\rfloor^2 \cdot (n+1) \\ + \left\lfloor \frac{a}{c} \right\rfloor \cdot \left\lfloor \frac{b}{c} \right\rfloor \cdot n(n+1) \\ + h(a \bmod c, b \bmod c, c, n) \\ + 2 \left\lfloor \frac{a}{c} \right\rfloor \cdot g(a \bmod c, b \bmod c, c, n) \\ + 2 \left\lfloor \frac{b}{c} \right\rfloor \cdot f(a \bmod c, b \bmod c, c, n), & a \geq c \vee b \geq c \\ 0, & n < 0 \vee a = 0 \\ nm(m+1) - 2g(c, c-b-1, a, m-1) \\ - 2f(c, c-b-1, a, m-1) - f(a, b, c, n), & \text{otherwise} \end{cases}$$

# 7 Polynomial

## 7.1 FWHT [c9cdb6]

```

/* x: a[j], y: a[j + (L >> 1)]
or: (y += x * op), and: (x += y * op)
xor: (x, y = (x + y) * op, (x - y) * op)
invop: or, and, xor = -1, -1, 1/2 */
void fwt(int *a, int n, int op) { //or
    for (int L = 2; L <= n; L <= 1)
        for (int i = 0; i < n; i += L)
            for (int j = i; j < i + (L >> 1); ++j)
                a[j + (L >> 1)] += a[j] * op;
}
const int N = 21;
int f[N][1 << N], g[N][1 << N], h[N][1 << N], ct[1 << N];
void subset_convolution(int *a, int *b, int *c, int L)
{
    // c_k = \sum_{i+j=k, i&j=0} a_i * b_j
    int n = 1 << L;
    for (int i = 1; i < n; ++i)
        ct[i] = ct[i & (i - 1)] + 1;
    for (int i = 0; i < n; ++i)
        f[ct[i]][i] = a[i], g[ct[i]][i] = b[i];
    for (int i = 0; i <= L; ++i)
        fwt(f[i], n, 1), fwt(g[i], n, 1);
    for (int i = 0; i <= L; ++i)
        for (int j = 0; j <= i; ++j)
            for (int x = 0; x < n; ++x)
                h[i][x] += f[j][x] * g[i - j][x];
    for (int i = 0; i <= L; ++i) fwt(h[i], n, -1);
    for (int i = 0; i < n; ++i) c[i] = h[ct[i]][i];
}

```

## 7.2 FFT [13ec2f]

// Errichto: FFT for double works when the result < 1e15, and < 1e18 with long double

```

using val_t = complex<double>;
template<int MAXN>
struct FFT {
    const double PI = acos(-1);
    val_t w[MAXN];
    FFT() {
        for (int i = 0; i < MAXN; ++i) {
            double arg = 2 * PI * i / MAXN;
            w[i] = val_t(cos(arg), sin(arg));
        }
    }
    void bitrev(vector<val_t> &a, int n) //same as NTT
    void trans(vector<val_t> &a, int n, bool inv = false)
    {
        bitrev(a, n);
        for (int L = 2; L <= n; L <= 1) {
            int dx = MAXN / L, dl = L >> 1;
            for (int i = 0; i < n; i += L) {
                for (int j = i, x = 0; j < i + dl; ++j, x += dx)
                {
                    val_t tmp = a[j + dl] * (inv ? conj(w[x]) : w[x]);
                    a[j + dl] = a[j] - tmp;
                    a[j] += tmp;
                }
            }
        }
        if (inv) {
            for (int i = 0; i < n; ++i) a[i] /= n;
        }
    }
    //multiplying two polynomials A * B:
    //fft.trans(A, siz, 0), fft.trans(B, siz, 0);
    //A[i] *= B[i], fft.trans(A, siz, 1);
};

```

## 7.3 NTT [bf683f]

```

//(2^16)+1, 65537, 3
//7*17*(2^23)+1, 998244353, 3
//1255*(2^20)+1, 1315962881, 3
//51*(2^25)+1, 1711276033, 29
// only works when sz(A) + sz(B) - 1 <= MAXN
template<int MAXN, ll P, ll RT> //MAXN must be 2^k
struct NTT {
    ll w[MAXN];
    ll mpow(ll a, ll n);

```

```

ll minv(ll a) { return mpow(a, P - 2); }
NTT() {
    ll dw = mpow(RT, (P - 1) / MAXN);
    w[0] = 1;
    for (int i = 1; i < MAXN; ++i) w[i] = w[i - 1] * dw % P;
}
void bitrev(vector<ll> &a, int n) {
    int i = 0;
    for (int j = 1; j < n - 1; ++j) {
        for (int k = n >> 1; (i ^ k) < k; k >>= 1);
        if (j < i) swap(a[i], a[j]);
    }
}
void operator()(vector<ll> &a, int n, bool inv = false) { //0 <= a[i] < P
    bitrev(a, n);
    for (int L = 2; L <= n; L <= 1) {
        int dx = MAXN / L, dl = L >> 1;
        for (int i = 0; i < n; i += L) {
            for (int j = i, x = 0; j < i + dl; ++j, x += dx)
            {
                ll tmp = a[j + dl] * w[x] % P;
                if ((a[j + dl] = a[j] - tmp) < 0) a[j + dl] += P;
                if ((a[j] += tmp) >= P) a[j] -= P;
            }
        }
    }
    if (inv) {
        reverse(a.begin() + 1, a.begin() + n);
        ll invn = minv(n);
        for (int i = 0; i < n; ++i) a[i] = a[i] * invn % P;
    }
}
};

```

## 7.4 Polynomial Operation [77a8a8]

```

// Copy from 8BQube
#define fi(s, n) for (int i = (int)(s); i < (int)(n); ++i)
#define neg(x) (x ? P - x : 0)
#define V (*this)
template<int MAXN, ll P, ll RT> // MAXN = 2^k
struct Poly : vector<ll> { // coefficients in [0, P)
    using vector<ll>::vector;
    static inline NTT<MAXN, P, RT> ntt;
    int n() const { return (int)size(); } // n() >= 1
    Poly(const Poly &p, int m) : vector<ll>(m) { copy_n(p.data(), min(p.n(), m), data()); }
    Poly &irev() { return reverse(data(), data() + n()), V; }
    Poly &isz(int m) { return resize(m), V; }
    static ll minv(ll x) { return ntt.minv(x); }
    Poly &iadd(const Poly &rhs) { // n() == rhs.n()
        fi(0, n()) if ((V[i] += rhs[i]) >= P) V[i] -= P;
        return V;
    }
    Poly &imul(ll k) {
        fi(0, n()) V[i] = V[i] * k % P;
        return V;
    }
    Poly Mul(const Poly &rhs) const {
        int m = 1;
        while (m < n() + rhs.n() - 1) m <= 1;
        assert(m <= MAXN);
        Poly X(V, m), Y(rhs, m);
        ntt(X, m), ntt(Y, m);
        fi(0, m) X[i] = X[i] * Y[i] % P;
        ntt(X, m, true);
        return X.isz(n() + rhs.n() - 1);
    }
    Poly Inv() const { // V[0] != 0, 2*sz<=MAXN
        if (n() == 1) return {minv(V[0])};
        int m = 1;
        while (m < n() * 2) m <= 1;
        assert(m <= MAXN);
        Poly Xi = Poly(V, (n() + 1) / 2).Inv().isz(m);
        Poly Y(V, m);
        ntt(Xi, m), ntt(Y, m);

```



```

fi(0, m) {
    Xi[i] *= (2 - Xi[i] * Y[i]) % P;
    if ((Xi[i] % P) < 0) Xi[i] += P;
}
ntt(Xi, m, true);
return Xi.isz(n());
}
Poly &shift_inplace(const ll &c) { // 2 * sz <= MAXN
    int n = V.n();
    vector<ll> fc(n), ifc(n);
    fc[0] = ifc[0] = 1;
    for (int i = 1; i < n; i++) {
        fc[i] = fc[i - 1] * i % P;
        ifc[i] = minv(fc[i]);
    }
    for (int i = 0; i < n; i++) V[i] = V[i] * fc[i] % P;
    Poly g(n);
    ll cp = 1;
    for (int i = 0; i < n; i++) g[i] = cp * ifc[i] % P,
        cp = cp * c % P;
    V = V.irev().Mul(g).isz(n).irev();
    for (int i = 0; i < n; i++) V[i] = V[i] * ifc[i] % P;
    return V;
}
Poly shift(const ll &c) const { return Poly(V).
    shift_inplace(c); }
Poly _Sqrt() const { // Jacobi(V[0], P) = 1
    if (n() == 1) return {QuadraticResidue(V[0], P)};
    Poly X = Poly(V, (n() + 1) / 2)._Sqrt().isz(n());
    return X.iadd(Mul(X.Inv()).isz(n()).imul(P / 2 + 1));
}
Poly Sqrt() const { // 2 * sz <= MAXN
    Poly a;
    bool has = 0;
    for (int i = 0; i < n(); i++) {
        if (V[i]) has = 1;
        if (has) a.push_back(V[i]);
    }
    if (!has) return V;
    if ((n() + a.n()) % 2 || Jacobi(a[0], P) != 1) {
        return Poly();
    }
    a = a.isz((n() + a.n()) / 2)._Sqrt();
    int sz = a.n();
    a.isz(n());
    rotate(a.begin(), a.begin() + sz, a.end());
    return a;
}
pair<Poly, Poly> DivMod(const Poly &rhs) const { //
    (rhs.)back() != 0
    if (n() < rhs.n()) return {{0}, V};
    const int m = n() - rhs.n() + 1;
    Poly X(rhs);
    X.irev().isz(m);
    Poly Y(V);
    Y.irev().isz(m);
    Poly Q = Y.Mul(X.Inv()).isz(m).irev();
    X = rhs.Mul(Q), Y = V;
    fi(0, n()) if ((Y[i] - X[i]) < 0) Y[i] += P;
    return {Q, Y.isz(max(1, rhs.n() - 1))};
}
Poly Dx() const {
    Poly ret(n() - 1);
    fi(0, ret.n()) ret[i] = (i + 1) * V[i + 1] % P;
    return ret.isz(max(1, ret.n()));
}
Poly Sx() const {
    Poly ret(n() + 1);
    fi(0, n()) ret[i + 1] = minv(i + 1) * V[i] % P;
    return ret;
}
Poly _tmul(int nn, const Poly &rhs) const {
    Poly Y = Mul(rhs).isz(n()) + nn - 1;
    return Poly(Y.data() + n() - 1, Y.data() + Y.n());
}
vector<ll> _eval(const vector<ll> &x, const vector<
    Poly> &up) const {
    const int m = (int)x.size();
    if (!m) return {};

```

```

vector<Poly> down(m * 2);
// down[1] = DivMod(up[1]).second;
// fi(2, m * 2) down[i] = down[i / 2].DivMod(up[i])
    .second;
down[1] = Poly(up[1]).irev().isz(n()).Inv().irev().
    _tmul(m, V);
fi(2, m * 2) down[i] = up[i ^ 1]._tmul(up[i].n() - 1, down[i / 2]);
vector<ll> y(m);
fi(0, m) y[i] = down[m + i][0];
return y;
}
static vector<Poly> _tree1(const vector<ll> &x) {
    const int m = (int)x.size();
    vector<Poly> up(m * 2);
    fi(0, m) up[m + i] = {neg(x[i]), 1};
    for (int i = m - 1; i > 0; --i) up[i] = up[i * 2].
        Mul(up[i * 2 + 1]);
    return up;
}
vector<ll> Eval(const vector<ll> &x) const { // 1e5,
    1s
    auto up = _tree1(x);
    return _eval(x, up);
}
static Poly Interpolate(const vector<ll> &x, const
    vector<ll> &y) { // 1e5, 1.4s
    const int m = (int)x.size();
    vector<Poly> up = _tree1(x), down(m * 2);
    vector<ll> z = up[1].Dx()._eval(x, up);
    fi(0, m) z[i] = y[i] * minv(z[i]) % P;
    fi(0, m) down[m + i] = {z[i]};
    for (int i = m - 1; i > 0; --i)
        down[i] = down[i * 2].Mul(up[i * 2 + 1]).iadd(
            down[i * 2 + 1].Mul(up[i * 2]));
    return down[1];
}
Poly Ln() const { // V[0] == 1, 2*sz<=MAXN
    return Dx().Mul(Inv()).Sx().isz(n());
}
Poly Exp() const { // V[0] == 0, 2*sz<=MAXN
    if (n() == 1) return {1};
    Poly X = Poly(V, (n() + 1) / 2).Exp().isz(n());
    Poly Y = X.Ln();
    Y[0] = P - 1;
    fi(0, n()) if ((Y[i] = V[i] - Y[i]) < 0) Y[i] += P;
    return X.Mul(Y).isz(n());
}
// M := P(P - 1). If k >= M, k := k % M + M.
Poly Pow(ll k) const { // 2*sz<=MAXN
    int nz = 0;
    while (nz < n() && !V[nz]) ++nz;
    if (nz * min(k, (ll)n()) >= n()) return Poly(n());
    if (!k) return Poly(Poly{1}, n());
    Poly X(data() + nz, data() + nz + n() - nz * k);
    const ll c = ntt.mpow(X[0], k % (P - 1));
    return X.Ln().imul(k % P).Exp().imul(c).irev().isz(
        n()).irev();
}
// sum_j w_j [x^j] f(x^i) for i in [0, m]
Poly power_projection(Poly wt, int m) { // 4*sz <=
    MAXN!
    assert(n() == wt.n());
    if (!n()) {
        return Poly(m + 1, 0);
    }
    if (V[0] != 0) {
        ll c = V[0];
        V[0] = 0;
        Poly A = V.power_projection(wt, m);
        fi(0, m + 1) A[i] = A[i] * fac[i] % P; //
            factorial
        Poly B(m + 1);
        ll pow = 1;
        fi(0, m + 1) B[i] = pow * ifac[i] % P, pow = pow
            * c % P; // inv. of fac
        A = A.Mul(B).isz(m + 1);
        fi(0, m + 1) A[i] = A[i] * fac[i] % P;
        return A;
    }
    int n = 1;

```

```

while (n < V.n()) n *= 2;
isz(n), wt.isz(n).irev();
int k = 1;
Poly p(wt, 2 * n), q(V, 2 * n);
q.imul(P - 1);

while (n > 1) {
    Poly r(2 * n * k);
    fi(0, 2 * n * k) r[i] = (i % 2 == 0 ? q[i] : neg(
        q[i]));
    Poly pq = p.Mul(r).isz(4 * n * k);
    Poly qq = q.Mul(r).isz(4 * n * k);
    fi(0, 2 * n * k) {
        pq[2 * n * k + i] += p[i];
        qq[2 * n * k + i] += q[i] + r[i];
        pq[2 * n * k + i] %= P;
        qq[2 * n * k + i] %= P;
    }
    fill(p.begin(), p.end(), 0);
    fill(q.begin(), q.end(), 0);
    for(int j = 0; j < 2 * k; j++) fi(0, n / 2) {
        p[n * j + i] = pq[(2 * n) * j + (2 * i + 1)];
        q[n * j + i] = qq[(2 * n) * j + (2 * i + 0)];
    }
    n /= 2, k *= 2;
}
Poly ans(k);
fi(0, k) ans[i] = p[2 * i];
return ans.irev().isz(m + 1);
}
Poly FPSinv() {
    const int n = V.n() - 1;
    if (n == -1) return {};
    assert(V[0] == 0);
    if (n == 0) return V;
    assert(V[1] != 0);
    ll c = V[1], ic = minv(c);
    imul(ic);
    Poly wt(n + 1);
    wt[n] = 1;

    Poly A = V.power_projection(wt, n);
    Poly g(n);
    fi(1, n + 1) g[n - i] = n * A[i] % P * minv(i) % P;
    g = g.Pow(neg(minv(n)));
    g.insert(g.begin(), 0);

    ll pow = 1;
    fi(0, g.n()) g[i] = g[i] * pow % P, pow = pow * ic
        % P;
    return g;
}
Poly TMul(const Poly &rhs) const { // this[i] - rhs[j]
    j = k;
    return Poly(*this).irev().Mul(rhs).isz(n()).irev();
}
Poly FPScomp(Poly g) { // solves V(g(x))
    auto rec = [&](auto &rec, int n, int k, Poly Q) ->
        Poly {
        if (n == 1) {
            Poly p(2 * k);
            irev();
            fi(0, k) p[2 * i] = V[i];
            return p;
        }
        Poly R(2 * n * k);
        fi(0, 2 * n * k) R[i] = (i % 2 == 0 ? Q[i] : neg(
            Q[i]));
        Poly QQ = Q.Mul(R).isz(4 * n * k);
        fi(0, 2 * n * k) {
            QQ[2 * n * k + i] += Q[i] + R[i];
            QQ[2 * n * k + i] %= P;
        }
        Poly nxt_Q(2 * n * k);
        for(int j = 0; j < 2 * k; j++) fi(0, n / 2) {
            nxt_Q[n * j + i] = QQ[(2 * n) * j + (2 * i + 0)];
        }
        Poly nxt_p = rec(rec, n / 2, k * 2, nxt_Q);
        Poly pq(4 * n * k);
        for(int j = 0; j < 2 * k; j++) fi(0, n / 2) {

```

```

        pq[(2 * n) * j + (2 * i + 1)] += nxt_p[n * j +
            i];
        pq[(2 * n) * j + (2 * i + 1)] %= P;
    }
    Poly p(2 * n * k);
    fi(0, 2 * n * k) p[i] = (p[i] + pq[2 * n * k + i
        ]) % P;
    pq.pop_back();
    Poly x = pq.TMul(R);
    fi(0, 2 * n * k) p[i] = (p[i] + x[i]) % P;
    return p;
};
int sz = 1;
while(sz < n() || sz < g.n()) sz <= 1;
return isz(sz), rec(rec, sz, 1, g.imul(P-1).isz(2 *
    sz)).isz(sz).irev();
}
};
#undef fi
#undef V
#undef neg
using Poly_t = Poly<1 << 19, 998244353, 3>;

```

## 7.5 Generating Function

### Ordinary Generating Function

- $C(x) = A(rx)$ :  $c_n = r^n a_n$  的一般生成函數。
- $C(x) = A(x) + B(x)$ :  $c_n = a_n + b_n$  的一般生成函數。
- $C(x) = A(x)B(x)$ :  $c_n = \sum_{i=0}^n a_i b_{n-i}$  的一般生成函數。
- $C(x) = A(x)^k$ :  $c_n = \sum_{i_1+i_2+\dots+i_k=n} a_{i_1} a_{i_2} \dots a_{i_k}$  的一般生成函數。
- $C(x) = xA(x)'$ :  $c_n = na_n$  的一般生成函數。
- $C(x) = \frac{A(x)}{1-x}$ :  $c_n = \sum_{i=0}^n a_i$  的一般生成函數。
- $C(x) = A(1) + x \frac{A(1)-A(x)}{1-x}$ :  $c_n = \sum_{i=n}^{\infty} a_i$  的一般生成函數。

常用展開式

- $\frac{1}{1-x} = 1 + x + x^2 + \dots + x^n + \dots$
- $(1+x)^a = \sum_{n=0}^{\infty} \binom{a}{n} x^n$ ,  $\binom{a}{n} = \frac{a(a-1)(a-2)\dots(a-n+1)}{n!}$ .

常見生函

- 卡特蘭數:  $f(x) = \frac{1-\sqrt{1-4x}}{2x}$

### Exponential Generating Function

$a_0, a_1, \dots$  的指數生成函數:

$$\hat{A}(x) = \sum_{i=0}^{\infty} \frac{a_i}{i!} = a_0 + a_1 x + \frac{a_2}{2!} x^2 + \frac{a_3}{3!} x^3 + \dots$$

- $\hat{C}(x) = \hat{A}(x) + \hat{B}(x)$ :  $c_n = a_n + b_n$  的指數生成函數
- $\hat{C}(x) = \hat{A}^{(k)}(x)$ :  $c_n = a_{n+k}$  的指數生成函數
- $\hat{C}(x) = x\hat{A}(x)$ :  $c_n = na_n$  的指數生成函數
- $\hat{C}(x) = \hat{A}(x)\hat{B}(x)$ :  $c_n = \sum_{k=0}^n \binom{n}{k} a_k b_{n-k}$  的指數生成函數
- $\hat{C}(x) = \hat{A}(x)^k$ :  $c_n = \sum_{i_1+i_2+\dots+i_k=n} \binom{n}{i_1, i_2, \dots, i_k} a_{i_1} a_{i_2} \dots a_{i_k}$  的指數生成函數
- $\hat{C}(x) = \exp(A(x))$ : 假設  $A(x)$  是一個分量 (component) 的生成函數, 那  $\hat{C}(x)$  是將  $n$  個有編號的東西分成若干個分量的指數生成函數

Lagrange's Inversion Formula

如果  $F$  跟  $G$  互反, 則有  $F(0), G(0) = 0, F'(0), G'(0) \neq 0$ 。若  $H$  為任意 FPS, 則

$$n[x^n]G(x) = [x^{n-1}] \frac{1}{(F(x)/x)^n}$$

$$n[x^n]H(G(x)) = [x^{n-1}]H'(x) \frac{1}{(F(x)/x)^n}$$

## 7.6 Bostan Mori [41c3bc]

```

const ll mod = 998244353;
NTT<262144, mod, 3> ntt;
// Finds the k-th coefficient of P / Q in O(d log d log
    k)
// size of NTT has to > 2 * d
ll BostanMori(vector<ll> P, vector<ll> Q, long long k)
{
    int d = max((int)P.size(), (int)Q.size() - 1);
    vector M = {P, Q};
    M[0].resize(d, 0);
    M[1].resize(d + 1, 0);
    int sz = (2 * d + 1 == 1 ? 2 : (1 << (__lg(2 * d) +
        1)));

```

```

vector<ll> Qn(sz);
vector N(2, vector<ll>(sz));
while(k) {
    fill(iter(Qn), 0);
    for(int i = 0; i < d + 1; i++){
        Qn[i] = M[1][i] * ((i & 1) ? -1 : 1);
        if(Qn[i] < 0) Qn[i] += mod;
    }
    ntt(Qn, sz, false);

    ll t[2] = {k & 1, 0};
    for(int i = 0; i < 2; i++){
        fill(iter(N[i]), 0);
        copy(iter(M[i]), N[i].begin());
        ntt(N[i], sz, false);
        for(int j = 0; j < sz; j++){
            N[i][j] = N[i][j] * Qn[j] % mod;
        }
        ntt(N[i], sz, true);
        for(int j = t[i]; j < 2 * siz(M[i]); j += 2){
            M[i][j >> 1] = N[i][j];
        }
    }
    k >>= 1;
}
return M[0][0] * ntt.minv(M[1][0]) % mod;
}

ll LinearRecursion(vector<ll> a, vector<ll> c, ll k) {
    // a_n = \sum_{j=1}^n a_{n-j} c_j
    int d = siz(a);
    int sz = (2 * d + 1 == 1 ? 2 : (1 << (lg(2 * d) + 1)));

    c[0] = mod - 1;
    for(ll &i : c) i = i % mod - i : 0;

    auto A = a; A.resize(sz);
    auto C = c; C.resize(sz);
    ntt(A, sz, false), ntt(C, sz, false);
    for(int i = 0; i < sz; i++) A[i] = A[i] * C[i] % mod;
    ntt(A, sz, true);
    A.resize(d);

    return BostanMori(A, c, k);
}

```

## 8 String

### 8.1 KMP Algorithm [c8b75f]

```

// 0-based
// fail[i] = max k<i s.t. s[0..k] = s[i-k..i]
vector<int> kmp_build_fail(const string &s){
    int n = SZ(s);
    vector<int> fail(n, -1);
    int cur = -1;
    for(int i = 1; i < n; i++){
        while(cur != -1 && s[cur + 1] != s[i])
            cur = fail[cur];
        if(s[cur + 1] == s[i])
            cur++;
        fail[i] = cur;
    }
    return fail;
}

void kmp_match(const string &s, const vector<int> &fail, const string &t){
    int cur = -1;
    int n = SZ(s), m = SZ(t);
    for(int i = 0; i < m; i++){
        while(cur != -1 && (cur + 1 == n || s[cur + 1] != t[i]))
            cur = fail[cur];
        if(cur + 1 < n && s[cur + 1] == t[i])
            cur++;
        // cur = max k s.t. s[0..k] = t[i-k..i]
    }
}

```

### 8.2 Manacher Algorithm [caf0f4]

```

/* center i: radius z[i * 2 + 1] / 2
   center i, i + 1: radius z[i * 2 + 2] / 2

```

```

both aba, abba have radius 2 */
vector<int> manacher(const string &tmp){ // 0-based
    string s = "%";
    int l = 0, r = 0;
    for(char c : tmp) s += c, s += '%';
    vector<int> z(SZ(s));
    for(int i = 0; i < SZ(s); i++){
        z[i] = r > i ? min(z[2 * l - i], r - i) : 1;
        while(i - z[i] >= 0 && i + z[i] < SZ(s)
            && s[i + z[i]] == s[i - z[i]])
            ++z[i];
        if(z[i] + i > r) r = z[i] + i, l = i;
    }
    return z;
}

```

### 8.3 Lyndon Factorization [7c612b]

```

// partition s = w[0] + w[1] + ... + w[k-1],
// w[0] >= w[1] >= ... >= w[k-1]
// each w[i] strictly smaller than all its suffix
void duval(const string &s, vector<pii> &w) {
    for (int n = (int)s.size(), i = 0, j, k; i < n; ) {
        for (j = i + 1, k = i; j < n && s[k] <= s[j]; j++)
            k = (s[k] < s[j] ? i : k + 1);
        // if (i < n / 2 && j >= n / 2) {
        // for min cyclic shift, call duval(s + s)
        // then here s.substr(i, n / 2) is min cyclic shift
        // }
        for (; i <= k; i += j - k)
            w.pb(pii(i, j - k)); // s.substr(l, len)
    }
}

```

### 8.4 Suffix Array [cd67ea]

```

struct SuffixArray {
    vector<int> sa, lcp, rank; // lcp[i] is lcp of sa[i]
                                // and sa[i-1]
                                // sa[0] = s.size()
                                // character should be 1-
                                // based
    SuffixArray(string &s, int lim=256) { // or
        basic_string<int>
        int n = s.size() + 1, k = 0, a, b;
        vector<int> x(n, 0), y(n), ws(max(n, lim));
        rank.assign(n, 0);
        for (int i = 0; i < n - 1; i++) x[i] = s[i];
        sa = lcp = y, iota(sa.begin(), sa.end(), 0);
        for (int j = 0, p = 0; p < n; j = max(1, j * 2),
            lim = p) {
            p = j, iota(y.begin(), y.end(), n - j);
            for (int i = 0; i < n; i++)
                if (sa[i] >= j) y[p++] = sa[i] - j;
            for (int &i : ws) i = 0;
            for (int i = 0; i < n; i++) ws[x[i]]++;
            for (int i = 1; i < lim; i++) ws[i] += ws[i - 1];
            for (int i = n; i--;) sa[--ws[x[y[i]]]] = y[i];
            swap(x, y), p = 1, x[sa[0]] = 0;
            for (int i = 1; i < n; i++){
                a = sa[i - 1], b = sa[i];
                x[b] = (y[a] == y[b] && y[a + j] == y[b + j]) ?
                    p - 1 : p++;
            }
        }
        for (int i = 1; i < n; i++) rank[sa[i]] = i;
        for (int i = 0, j; i < n - 1; lcp[rank[i+1]] = k)
            for (k && k--, j = sa[rank[i] - 1];
                s[i + k] == s[j + k]; k++);
    }
};

```

### 8.5 Suffix Automaton [016373]

```

struct exSAM {
    const int CNUM = 26;
    // Len: maxlength, Link: fail link
    // LenSorted: topo order, cnt: occur
    vector<int> len, link, lenSorted, cnt;
    vector<vector<int>> next;
    int total = 0;
    int newnode() {
        return total++;
    }
};

```

```

}
void init(int n) { // total number of characters
    len.assign(2 * n, 0); link.assign(2 * n, 0);
    lenSorted.assign(2 * n, 0); cnt.assign(2 * n, 0);
    next.assign(2 * n, vector<int>(CNUM));
    newnode(), link[0] = -1;
}
int insertSAM(int last, int c) {
    // not exSAM: cur = newnode(), p = last
    int cur = next[last][c];
    len[cur] = len[last] + 1;
    int p = link[last];
    while (p != -1 && !next[p][c])
        next[p][c] = cur, p = link[p];
    if (p == -1) return link[cur] = 0, cur;
    int q = next[p][c];
    if (len[p] + 1 == len[q]) return link[cur] = q, cur;
    ;
    int clone = newnode();
    for (int i = 0; i < CNUM; ++i)
        next[clone][i] = len[next[q][i]] ? next[q][i] : 0;
    len[clone] = len[p] + 1;
    while (p != -1 && next[p][c] == q)
        next[p][c] = clone, p = link[p];
    link[link[cur] = clone] = link[q];
    link[q] = clone;
    return cur;
}
void insert(const string &s) {
    int cur = 0;
    for (auto ch : s) {
        int &nxt = next[cur][int(ch - 'a')];
        if (!nxt) nxt = newnode();
        cnt[cur = nxt] += 1;
    }
}
void build() {
    queue<int> q;
    q.push(0);
    while (!q.empty()) {
        int cur = q.front();
        q.pop();
        for (int i = 0; i < CNUM; ++i)
            if (next[cur][i])
                q.push(insertSAM(cur, i));
    }
    vector<int> lc(total);
    for (int i = 1; i < total; ++i) ++lc[len[i]];
    partial_sum(iter(lc), lc.begin());
    for (int i = 1; i < total; ++i) lenSorted[--lc[len[i]]] = i;
}
void solve() {
    for (int i = total - 2; i >= 0; --i)
        cnt[link[lenSorted[i]]] += cnt[lenSorted[i]];
}
};

```

## 8.6 Z-value Algorithm [488d87]

```

// z[i] = max k s.t. s[0..k-1] = s[i..i+k-1]
// i.e. Length of longest common prefix
// z[0] = 0
vector<int> z_function(const string &s){
    int n = s.size();
    vector<int> z(n);
    for(int i = 1, l = 0, r = 0; i < n; i++){
        if(i <= r) z[i] = min(r - i + 1, z[i - l]);
        while(i + z[i] < n && s[z[i]] == s[i + z[i]])
            z[i]++;
        if(i + z[i] - 1 > r)
            l = i, r = i + z[i] - 1;
    }
    return z;
}

```

## 8.7 Main Lorentz [fcfb8f]

```

struct Rep{ int minl, maxl, len; };
vector<Rep> rep; // 0-base
// p \in [minl, maxl] => s[p, p + i) = s[p + i, p + 2i)
void main_lorentz(const string &s, int sft = 0) {

```

```

    const int n = s.size();
    if (n == 1) return;
    const int nu = n / 2, nv = n - nu;
    const string u = s.substr(0, nu), v = s.substr(nu),
        ru(u.rbegin(), u.rend()), rv(v.rbegin(), v.rend());
    main_lorentz(u, sft), main_lorentz(v, sft + nu);
    const auto z1 = z_function(ru), z2 = z_function(v + '#' + u),
        z3 = z_function(ru + '#' + rv), z4 = z_function(v);
    auto get_z = [](const vector<int> &z, int i) {
        return (0 <= i && i < (int)z.size()) ? z[i] : 0;
    };
    auto add_rep = [&](bool left, int c, int l, int k1, int k2) {
        const int L = max(1, l - k2), R = min(l - left, k1);
        ;
        if (L > R) return;
        if (left) rep.emplace_back(Rep({sft + c - R, sft + c - L, l}));
        else rep.emplace_back(Rep({sft + c - R - l + 1, sft + c - L - l + 1, l}));
    };
    for (int cntr = 0; cntr < n; cntr++) {
        int l, k1, k2;
        if (cntr < nu) {
            l = nu - cntr;
            k1 = get_z(z1, nu - cntr);
            k2 = get_z(z2, nv + 1 + cntr);
        } else {
            l = cntr - nu + 1;
            k1 = get_z(z3, nu + 1 + nv - 1 - (cntr - nu));
            k2 = get_z(z4, (cntr - nu) + 1);
        }
        if (k1 + k2 >= 1)
            add_rep(cntr < nu, cntr, l, k1, k2);
    }
}

```

## 8.8 AC Automaton [e925e5]

```

const int SIGMA = 26;
struct AC_Automaton {
    // child: trie, next: automaton
    vector<vector<int>> child, next;
    vector<int> fail, cnt, ord;
    int total = 0;
    int newnode() {
        return total++;
    }
    void init(int len) { // len >= 1 + total len
        child.assign(len, vector<int>(26, -1));
        next.assign(len, vector<int>(26, -1));
        fail.assign(len, -1); cnt.assign(len, 0);
        ord.clear();
        newnode();
    }
    int input(string &s) {
        int cur = 0;
        for (char c : s) {
            if (child[cur][c - 'A'] == -1)
                child[cur][c - 'A'] = newnode();
            cur = child[cur][c - 'A'];
        }
        return cur; // return the end node of string
    }
    void make_fl() {
        queue<int> q;
        q.push(0), fail[0] = -1;
        while(!q.empty()) {
            int R = q.front();
            q.pop(); ord.pb(R);
            for (int i = 0; i < SIGMA; ++i)
                if (child[R][i] != -1) {
                    int X = next[R][i] = child[R][i], Z = fail[R];
                    ;
                    while (Z != -1 && child[Z][i] == -1)
                        Z = fail[Z];
                    fail[X] = Z != -1 ? child[Z][i] : 0;
                    q.push(X);
                }
            else next[R][i] = R ? next[fail[R]][i] : 0;
        }
    }
}

```

```

    }
}
void solve() {
    for (int i : ord | views::reverse)
        cnt[fail[i]] += cnt[i];
}
};

```

## 8.9 Palindrome Automaton [8a071b]

```

struct PalindromicTree {
    struct node {
        int nxt[26], fail, len; // num = depth of fail link
        int cnt, num; // cnt = occur, num = #pal_suffix of
            this node
        node(int l = 0) : nxt{}, fail(0), len(l), cnt(0), num
            (0) {}
    };
    vector<node> st; vector<int> s; int last, n;
    void init() {
        st.clear(); s.clear(); last = 1; n = 0;
        st.pb(0); st.pb(-1);
        st[0].fail = 1; s.pb(-1);
    }
    int getFail(int x) {
        while (s[n - st[x].len - 1] != s[n]) x = st[x].fail;
        return x;
    }
    void add(int c) {
        s.pb(c -= 'a'); ++n;
        int cur = getFail(last);
        if (!st[cur].nxt[c]) {
            int now = SZ(st);
            st.pb(st[cur].len + 2);
            st[now].fail = st[getFail(st[cur].fail)].nxt[c];
            st[cur].nxt[c] = now;
            st[now].num = st[st[now].fail].num + 1;
        }
        last = st[cur].nxt[c]; ++st[last].cnt;
    }
    void dpCnt() {
        for (int i = SZ(st) - 1; i >= 0; i--) {
            auto nd = st[i];
            st[nd.fail].cnt += nd.cnt;
        }
    }
    int size() { return (int)st.size() - 2; }
};

```

## 9 Misc

### 9.1 Cyclic Ternary Search [9017cc]

```

/* bool pred(int a, int b);
f(0) ~ f(n - 1) is a cyclic-shift U-function
return idx s.t. pred(x, idx) is false forall x*/
int cyc_tsearch(int n, auto pred) {
    if (n == 1) return 0;
    int l = 0, r = n; bool rv = pred(1, 0);
    while (r - l > 1) {
        int m = (l + r) / 2;
        if (pred(0, m) ? rv : pred(m, (m + 1) % n)) r = m;
        else l = m;
    }
    return pred(1, r % n) ? l : r % n;
}

```

### 9.2 Matroid

$M = (E, \mathcal{I})$ , where  $\mathcal{I} \subseteq 2^E$  is nonempty, is a matroid if:

- If  $S \in \mathcal{I}$  and  $S' \subsetneq S$ , then  $S' \in \mathcal{I}$ .
- For  $S_1, S_2 \in \mathcal{I}$  s.t.  $|S_1| < |S_2|$ , there exists  $e \in S_2 \setminus S_1$  s.t.  $S_1 \cup \{e\} \in \mathcal{I}$ .

Matroid intersection:

Start from  $S = \emptyset$ . In each iteration, let

- $Y_1 = \{x \notin S \mid S \cup \{x\} \in \mathcal{I}_1\}$
- $Y_2 = \{x \notin S \mid S \cup \{x\} \in \mathcal{I}_2\}$

If there exists  $x \in Y_1 \cap Y_2$ , insert  $x$  into  $S$ . Otherwise for each  $x \in S, y \notin S$ , create edges

- $x \rightarrow y$  if  $S - \{x\} \cup \{y\} \in \mathcal{I}_1$ .
- $y \rightarrow x$  if  $S - \{x\} \cup \{y\} \in \mathcal{I}_2$ .

Find a *shortest* path (with BFS) starting from a vertex in  $Y_1$  and ending at a vertex in  $Y_2$  which doesn't pass through any other vertices in  $Y_2$ , and alternate the path. The size of  $S$  will be incremented by 1 in each iteration.

For the weighted case, assign weight  $w(x)$  to vertex  $x$  if  $x \in S$  and  $-w(x)$  if  $x \notin S$ . Find the path with the minimum number of edges among all minimum length paths and alternate it.

### 9.3 Simulate Annealing [ff826c]

```

ld anneal() {
    mt19937 rnd_engine(seed);
    uniform_real_distribution<ld> rnd(0, 1);
    const ld dT = 0.001;
    // Argument p
    ld S_cur = calc(p), S_best = S_cur;
    for (ld T = 2000; T > eps; T -= dT) {
        // Modify p to p_prime
        const ld S_prime = calc(p_prime);
        const ld delta_c = S_prime - S_cur;
        ld prob = min((ld)1, exp(-delta_c / T));
        if (rnd(rnd_engine) <= prob)
            S_cur = S_prime, p = p_prime;
        if (S_prime < S_best) // find min
            S_best = S_prime, p_best = p_prime;
    }
    return S_best;
}

```

### 9.4 Binary Search On Fraction [f6b9ec]

```

struct Q {
    ll p, q;
    Q go(Q b, ll d) { return {p + b.p * d, q + b.q * d};
    };
    // returns smallest p/q in [lo, hi] such that
    // pred(p/q) is true, and 0 <= p, q <= N
    Q frac_bs(ll N, auto &&pred) {
        Q lo{0, 1}, hi{1, 0};
        if (pred(lo)) return lo;
        assert(pred(hi));
        bool dir = 1, L = 1, H = 1;
        for (; L || H; dir = !dir) {
            ll len = 0, step = 1;
            for (int t = 0; t < 2 && (t ? step /= 2 : step *= 2);)
                if (Q mid = hi.go(lo, len + step);
                    mid.p > N || mid.q > N || dir ^ pred(mid))
                    t++;
                else len += step;
            swap(lo, hi = hi.go(lo, len));
            (dir ? L : H) = !len;
        }
        return dir ? hi : lo;
    }
};

```

### 9.5 Min Plus Convolution [09b5c3]

```

// a is convex a[i+1]-a[i] <= a[i+2]-a[i+1]
vector<int> min_plus_convolution(vector<int> &a, vector
    <int> &b) {
    int n = SZ(a), m = SZ(b);
    vector<int> c(n + m - 1, INF);
    auto dc = [&](auto Y, int l, int r, int jl, int jr) {
        if (l > r) return;
        int mid = (l + r) / 2, from = -1, &best = c[mid];
        for (int j = jl; j <= jr; ++j)
            if (int i = mid - j; i >= 0 && i < n)
                if (best > a[i] + b[j])
                    best = a[i] + b[j], from = j;
        Y(Y, l, mid - 1, jl, from), Y(Y, mid + 1, r, from, jr);
    };
    return dc(dc, 0, n - 1 + m - 1, 0, m - 1), c;
}

```

## 10 Notes

### 10.1 Geometry

#### Rotation Matrix

$$\begin{pmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{pmatrix}$$

- rotate  $90^\circ$ :  $(x, y) \rightarrow (-y, x)$
- rotate  $-90^\circ$ :  $(x, y) \rightarrow (y, -x)$



## Triangles

Side lengths:  $a, b, c$

$$\text{Semiperimeter: } p = \frac{a+b+c}{2}$$

$$\text{Area: } A = \sqrt{p(p-a)(p-b)(p-c)}$$

$$\text{Circumradius: } R = \frac{abc}{4A}$$

$$\text{Inradius: } r = \frac{A}{p}$$

Length of median (divides triangle into two equal-area triangles):  $m_a = \frac{1}{2}\sqrt{2b^2 + 2c^2 - a^2}$

$$\text{Length of bisector (divides angles in two): } s_a = \sqrt{bc \left( 1 - \left( \frac{a}{b+c} \right)^2 \right)}$$

$$\text{Law of sines: } \frac{\sin \alpha}{a} = \frac{\sin \beta}{b} = \frac{\sin \gamma}{c} = \frac{1}{2R}$$

$$\text{Law of cosines: } a^2 = b^2 + c^2 - 2bc \cos \alpha$$

$$\text{Law of tangents: } \frac{a+b}{a-b} = \frac{\tan \frac{\alpha+\beta}{2}}{\tan \frac{\alpha-\beta}{2}}$$

## Quadrilaterals

With side lengths  $a, b, c, d$ , diagonals  $e, f$ , diagonals angle  $\theta$ , area  $A$  and magic flux  $F = b^2 + d^2 - a^2 - c^2$ :

$$4A = 2ef \cdot \sin \theta = F \tan \theta = \sqrt{4e^2 f^2 - F^2}$$

For cyclic quadrilaterals the sum of opposite angles is  $180^\circ$ ,  $ef = ac + bd$ , and  $A = \sqrt{(p-a)(p-b)(p-c)(p-d)}$ .

## Spherical coordinates

$$\begin{aligned} x &= r \sin \theta \cos \phi & r &= \sqrt{x^2 + y^2 + z^2} \\ y &= r \sin \theta \sin \phi & \theta &= \arccos(z/\sqrt{x^2 + y^2 + z^2}) \\ z &= r \cos \theta & \phi &= \operatorname{atan2}(y, x) \end{aligned}$$

## Green's Theorem

$$\iint_D \left( \frac{\partial Q}{\partial x} - \frac{\partial P}{\partial y} \right) dx dy = \oint_{L^+} (P dx + Q dy)$$

$$\text{Area} = \frac{1}{2} \oint_L x dy - y dx$$

Circular sector:

$$x = x_0 + r \cos \theta$$

$$y = y_0 + r \sin \theta$$

$$\begin{aligned} A &= r \int_{\alpha}^{\beta} (x_0 + \cos \theta) \cos \theta + (y_0 + \sin \theta) \sin \theta d\theta \\ &= r(r\theta + x_0 \sin \theta - y_0 \cos \theta)|_{\alpha}^{\beta} \end{aligned}$$

## Point-Line Duality

$$p = (a, b) \leftrightarrow p^* : y = ax - b$$

- $p \in l \iff l^* \in p^*$
- $p_1, p_2, p_3$  are collinear  $\iff p_1^*, p_2^*, p_3^*$  intersect at a point
- $p$  lies above  $l \iff l^*$  lies above  $p^*$
- lower convex hull  $\leftrightarrow$  upper envelope

## 10.2 Trigonometry

$$\begin{aligned} \sinh x &= \frac{1}{2}(e^x - e^{-x}) & \cosh x &= \frac{1}{2}(e^x + e^{-x}) \\ \sin n\pi &= 0 & \cos n\pi &= (-1)^n \end{aligned}$$

$$\sin(\alpha + \beta) = \sin \alpha \cos \beta + \cos \alpha \sin \beta$$

$$\cos(\alpha + \beta) = \cos \alpha \cos \beta - \sin \alpha \sin \beta$$

$$\sin(2\alpha) = 2 \cos \alpha \sin \alpha$$

$$\cos(2\alpha) = \cos^2 \alpha - \sin^2 \alpha$$

$$= 2 \cos^2 \alpha - 1$$

$$= 1 - 2 \sin^2 \alpha$$

$$\tan(\alpha + \beta) = \frac{\tan \alpha + \tan \beta}{1 - \tan \alpha \tan \beta}$$

$$\sin \alpha + \sin \beta = 2 \sin \frac{\alpha + \beta}{2} \cos \frac{\alpha - \beta}{2}$$

$$\cos \alpha + \cos \beta = 2 \cos \frac{\alpha + \beta}{2} \cos \frac{\alpha - \beta}{2}$$

$$\sin \alpha \sin \beta = \frac{1}{2}(\cos(\alpha - \beta) - \cos(\alpha + \beta))$$

$$\sin \alpha \cos \beta = \frac{1}{2}(\sin(\alpha + \beta) + \sin(\alpha - \beta))$$

$$\cos \alpha \sin \beta = \frac{1}{2}(\sin(\alpha + \beta) - \sin(\alpha - \beta))$$

$$\cos \alpha \cos \beta = \frac{1}{2}(\cos(\alpha - \beta) + \cos(\alpha + \beta))$$

$$(V + W) \tan(\alpha - \beta)/2 = (V - W) \tan(\alpha + \beta)/2$$

where  $V, W$  are lengths of sides opposite angles  $\alpha, \beta$ .

$$a \cos x + b \sin x = r \cos(x - \phi)$$

$$a \sin x + b \cos x = r \sin(x + \phi)$$

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

## 10.3 Calculus

Integration by parts:

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

$$\frac{d}{dx} \arcsin x = \frac{1}{\sqrt{1-x^2}}$$

$$\frac{d}{dx} \arccos x = -\frac{1}{\sqrt{1-x^2}}$$

$$\frac{d}{dx} \tan x = 1 + \tan^2 x$$

$$\frac{d}{dx} \arctan x = \frac{1}{1+x^2}$$

$$\int \tan ax = -\frac{\ln |\cos ax|}{a}$$

$$\int x \sin ax = \frac{\sin ax - ax \cos ax}{a^2}$$

$$\int e^{-x^2} = \frac{\sqrt{\pi}}{2} \operatorname{erf}(x)$$

$$\int x e^{ax} = \frac{e^{ax}}{a^2} (ax - 1)$$

$$\int \sin^2(x) = \frac{x}{2} - \frac{1}{4} \sin 2x$$

$$\int \sin^3 x = \frac{1}{12} \cos 3x - \frac{3}{4} \cos x$$

$$\int \cos^2(x) = \frac{x}{2} + \frac{1}{4} \sin 2x$$

$$\int \cos^3 x = \frac{1}{12} \sin 3x + \frac{3}{4} \sin x$$

$$\int x \sin x = \sin x - x \cos x$$

$$\int x \cos x = \cos x + x \sin x$$

$$\int x e^x = e^x (x - 1)$$

$$\int x^2 e^x = e^x (x^2 - 2x + 2)$$

$$\int x^2 \sin x = 2x \sin x - (x^2 - 2) \cos x$$

$$\int x^2 \cos x = 2x \cos x + (x^2 - 2) \sin x$$

$$\int e^x \sin x = \frac{1}{2} e^x (\sin x - \cos x)$$

$$\int e^x \cos x = \frac{1}{2} e^x (\sin x + \cos x)$$

$$\int x e^x \sin x = \frac{1}{2} e^x (x \sin x - x \cos x + \cos x)$$

$$\int x e^x \cos x = \frac{1}{2} e^x (x \sin x + x \cos x - \sin x)$$

## 10.4 Sum & Series

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

$$1 + 2 + 3 + \dots + n = \frac{n(n+1)}{2}$$

$$1^2 + 2^2 + 3^2 + \dots + n^2 = \frac{n(2n+1)(n+1)}{6}$$

$$1^3 + 2^3 + 3^3 + \dots + n^3 = \frac{n^2(n+1)^2}{4}$$

$$1^4 + 2^4 + 3^4 + \dots + n^4 = \frac{n(n+1)(2n+1)(3n^2+3n-1)}{30}$$

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \dots, (-\infty < x < \infty)$$



## TOPC Addition

### Trapezoidalization [4d3bca]

```
template<class T>
struct SweepLine {
    struct cmp {
        cmp(const SweepLine &swp): swp(_swp) {}
        bool operator()(int a, int b) const {
            if (abs(swp.get_y(a) - swp.get_y(b)) <= swp.eps)
                return swp.slope_cmp(a, b);
            return swp.get_y(a) + swp.eps < swp.get_y(b);
        }
        const SweepLine &swp;
    } _cmp;
    T curTime, eps, curQ;
    vector<Line> base;
    multiset<int, cmp> sweep;
    multiset<pair<T, int>> event;
    vector<typename multiset<int, cmp>::iterator> its;
    vector<typename multiset<pair<T, int>>::iterator> eits;
    bool slope_cmp(int a, int b) const {
        assert(a != -1);
        if (b == -1) return 0;
        return sign(cross(base[a].Y - base[a].X, base[b].Y - base[b].X)) < 0;
    }
    T get_y(int idx) const {
        if (idx == -1) return curQ;
        Line l = base[idx];
        if (l.X.X == l.Y.X) return l.Y.Y;
        return ((curTime - l.X.X) * l.Y.Y + (l.Y.X - curTime) * l.X.Y) / (l.Y.X - l.X.X);
    }
    void insert(int idx) {
        its[idx] = sweep.insert(idx);
        if (its[idx] != sweep.begin())
            update_event(*prev(its[idx]));
        update_event(idx);
        event.emplace(base[idx].Y.X, idx + 2 * SZ(base));
    }
    void erase(int idx) {
        assert(eits[idx] == event.end());
        auto p = sweep.erase(its[idx]);
        its[idx] = sweep.end();
        if (p != sweep.begin())
            update_event(*prev(p));
    }
    void update_event(int idx) {
        if (eits[idx] != event.end())
            event.erase(eits[idx]);
        eits[idx] = event.end();
        auto nxt = next(its[idx]);
        if (nxt == sweep.end() || !slope_cmp(idx, *nxt))
            return;
        auto t = intersect(base[idx].X, base[idx].Y, base[*nxt].X, base[*nxt].Y.X);
        if (t + eps < curTime || t >= min(base[idx].Y.X, base[*nxt].Y.X)) return;
        eits[idx] = event.emplace(t, idx + SZ(base));
    }
    void swp(int idx) {
        assert(eits[idx] != event.end());
        eits[idx] = event.end();
        int nxt = *next(its[idx]);
        swap((int&)*its[idx], (int&)*its[nxt]);
        swap(its[idx], its[nxt]);
        if (its[nxt] != sweep.begin())
            update_event(*prev(its[nxt]));
        update_event(idx);
    }
    // only expected to call the functions below
    SweepLine(T t, T e, vector<Line> vec): _cmp(*this),
        curTime(t), eps(e), curQ(), base(vec), sweep(_cmp), event(), its(SZ(vec)), eits(end()) {
        for (int i = 0; i < SZ(base); ++i) {
            auto &p, q = base[i];
            if (p > q) swap(p, q);
            if (p.X <= curTime && curTime <= q.X)
                insert(i);
            else if (curTime < p.X)
```

```
                event.emplace(p.X, i);
        }
    }
    void setTime(T t, bool ers = false) {
        assert(t >= curTime);
        while (!event.empty() && event.begin()->X <= t) {
            auto [et, idx] = *event.begin();
            int s = idx / SZ(base);
            idx %= SZ(base);
            if (abs(et - t) <= eps && s == 2 && !ers) break;
            curTime = et;
            event.erase(event.begin());
            if (s == 2) erase(idx);
            else if (s == 1) swp(idx);
            else insert(idx);
        }
        curTime = t;
    }
    T nextEvent() {
        if (event.empty()) return INF;
        return event.begin()->X;
    }
    int lower_bound(T y) {
        curQ = y;
        auto p = sweep.lower_bound(-1);
        if (p == sweep.end()) return -1;
        return *p;
    }
};
```

### Weighted General Matching [900530]

```
#define pb emplace_back
#define REP(i, l, r) for (int i=(l); i<=(r); ++i)
struct WeightGraph { // 1-based
    static const int inf = INT_MAX;
    struct edge { int u, v, w; }; int n, nx;
    vector<int> lab; vector<vector<edge>> g;
    vector<int> slack, match, st, pa, S, vis;
    vector<vector<int>> flo, flo_from; queue<int> q;
    WeightGraph(int n_) : n(n_), nx(n * 2), lab(nx + 1),
        g(nx + 1, vector<edge>(nx + 1)), slack(nx + 1),
        flo(nx + 1), flo_from(nx + 1, vector(n + 1, 0)) {
        match = st = pa = S = vis = slack;
        REP(u, 1, n) REP(v, 1, n) g[u][v] = {u, v, 0};
    }
    int ED(edge e) {
        return lab[e.u] + lab[e.v] - g[e.u][e.v].w * 2;
    }
    void update_slack(int u, int x, int &s) {
        if (!s || ED(g[u][x]) < ED(g[s][x])) s = u;
    }
    void set_slack(int x) {
        slack[x] = 0;
        REP(u, 1, n)
            if (g[u][x].w > 0 && st[u] != x && S[st[u]] == 0)
                update_slack(u, x, slack[x]);
    }
    void q_push(int x) {
        if (x <= n) q.push(x);
        else for (int y : flo[x]) q.push(y);
    }
    void set_st(int x, int b) {
        st[x] = b;
        if (x > n) for (int y : flo[x]) set_st(y, b);
    }
    vector<int> split_flo(auto &f, int xr) {
        auto it = find(all(f), xr);
        if (auto pr = it - f.begin(); pr % 2 == 1)
            reverse(1 + all(f), it = f.end() - pr);
        auto res = vector(f.begin(), it);
        return f.erase(f.begin(), it), res;
    }
    void set_match(int u, int v) {
        match[u] = g[u][v].v;
        if (u <= n) return;
        int xr = flo_from[u][g[u][v].u];
        auto &f = flo[u], z = split_flo(f, xr);
        REP(i, 0, int(z.size())-1) set_match(z[i], z[i ^ 1]);
        set_match(xr, v); f.insert(f.end(), all(z));
    }
    void augment(int u, int v) {
        for (;;) {
            int xnv = st[match[u]]; set_match(u, v);
```

```

    if (!xnv) return;
    set_match(v = xnv, u = st[pa[xnv]]);
}
}
/* SPLIT_HASH_HERE */
int lca(int u, int v) {
    static int t = 0; ++t;
    for (++t; u || v; swap(u, v)) if (u) {
        if (vis[u] == t) return u;
        vis[u] = t; u = st[match[u]];
        if (u) u = st[pa[u]];
    }
    return 0;
}
void add_blossom(int u, int o, int v) {
    int b = int(find(n + 1 + all(st), 0) - begin(st));
    lab[b] = 0, S[b] = 0; match[b] = match[o];
    vector<int> f = {o};
    for (int x : {u, v}) {
        for (int y; x != o; x = st[pa[y]])
            f.pb(x), f.pb(y = st[match[x]]), q_push(y);
        reverse(1 + all(f));
    }
    flo[b] = f; set_st(b, b);
    REP(x, 1, nx) g[b][x].w = g[x][b].w = 0;
    REP(x, 1, n) flo_from[b][x] = 0;
    for (int xs : flo[b]) {
        REP(x, 1, nx)
            if (g[b][x].w == 0 || ED(g[xs][x]) < ED(g[b][x]))
                g[b][x] = g[xs][x], g[x][b] = g[x][xs];
        REP(x, 1, n)
            if (flo_from[xs][x]) flo_from[b][x] = xs;
    }
    set_slack(b);
}
void expand_blossom(int b) {
    for (int x : flo[b]) set_st(x, x);
    int xr = flo_from[b][g[b][pa[b]].u], xs = -1;
    for (int x : split_flo(flo[b], xr)) {
        if (xs == -1) { xs = x; continue; }
        pa[xs] = g[x][xs].u; S[xs] = 1, S[x] = 0;
        slack[xs] = 0; set_slack(x); q_push(x); xs = -1;
    }
    for (int x : flo[b])
        if (x == xr) S[x] = 1, pa[x] = pa[b];
        else S[x] = -1, set_slack(x);
    st[b] = 0;
}
bool on_found_edge(const edge &e) {
    if (int u = st[e.u], v = st[e.v]; S[v] == -1) {
        int nu = st[match[v]]; pa[v] = e.u; S[v] = 1;
        slack[v] = slack[nu] = 0; S[nu] = 0; q_push(nu);
    } else if (S[v] == 0) {
        if (int o = lca(u, v)) add_blossom(u, o, v);
        else return augment(u, v), augment(v, u), true;
    }
    return false;
}
}
/* SPLIT_HASH_HERE */
bool matching() {
    ranges::fill(S, -1); ranges::fill(slack, 0);
    q = queue<int>();
    REP(x, 1, nx) if (st[x] == x && !match[x])
        pa[x] = 0, S[x] = 0, q_push(x);
    if (q.empty()) return false;
    for (;;) {
        while (q.size()) {
            int u = q.front(); q.pop();
            if (S[st[u]] == 1) continue;
            REP(v, 1, n)
                if (g[u][v].w > 0 && st[u] != st[v]) {
                    if (ED(g[u][v]) != 0)
                        update_slack(u, st[v], slack[st[v]]);
                    else if (on_found_edge(g[u][v])) return true;
                }
        }
        int d = inf;
        REP(b, n + 1, nx) if (st[b] == b && S[b] == 1)
            d = min(d, lab[b] / 2);
        REP(x, 1, nx)

```

```

        if (int s = slack[x]; st[x] == x && s && S[x]
            <= 0)
            d = min(d, ED(g[s][x]) / (S[x] + 2));
        REP(u, 1, n)
            if (S[st[u]] == 1) lab[u] += d;
            else if (S[st[u]] == 0) {
                if (lab[u] <= d) return false;
                lab[u] -= d;
            }
        REP(b, n + 1, nx) if (st[b] == b && S[b] >= 0)
            lab[b] += d * (2 - 4 * S[b]);
        REP(x, 1, nx)
            if (int s = slack[x]; st[x] == x &&
                s && st[s] != x && ED(g[s][x]) == 0)
                if (on_found_edge(g[s][x])) return true;
        REP(b, n + 1, nx)
            if (st[b] == b && S[b] == 1 && lab[b] == 0)
                expand_blossom(b);
    }
    return false;
}
pair<lld, int> solve() {
    ranges::fill(match, 0);
    REP(u, 0, n) st[u] = u, flo[u].clear();
    int w_max = 0;
    REP(u, 1, n) REP(v, 1, n) {
        flo_from[u][v] = (u == v ? u : 0);
        w_max = max(w_max, g[u][v].w);
    }
    REP(u, 1, n) lab[u] = w_max;
    int n_matches = 0; lld tot_weight = 0;
    while (matching()) ++n_matches;
    REP(u, 1, n) if (match[u] && match[u] < u)
        tot_weight += g[u][match[u]].w;
    return make_pair(tot_weight, n_matches);
}
void set_edge(int u, int v, int w) {
    g[u][v].w = g[v][u].w = w; }
};

```