

# Contents

<b>1 Basic</b>	<b>1</b>	<b>8 Polynomial</b>	<b>19</b>
1.1 Default Code	1	8.1 FWHT	19
1.2 .vimrc	2	8.2 FFT	19
1.3 Fast IO	2	8.3 NTT	19
1.4 Random	2	8.4 Polynomial Operation	20
1.5 Checker	2	8.5 Generating Function	21
1.6 PBDS Tree	2	8.5.1 Ordinary Generating Function	21
		8.5.2 Exponential Generating Function	21
		8.6 Bostan Mori	21
<b>2 Data Structure</b>	<b>2</b>	<b>9 String</b>	<b>22</b>
2.1 Heavy-Light Decomposition	2	9.1 KMP Algorithm	22
2.2 Link Cut Tree	2	9.2 Manacher Algorithm	22
2.3 Treap	3	9.3 MCP	22
2.4 KD Tree	4	9.4 Suffix Array	22
2.5 Leftist Tree	4	9.5 Suffix Automaton	22
		9.6 Z-value Algorithm	23
<b>3 Flow &amp; Matching</b>	<b>4</b>	9.7 Main Lorentz	23
3.1 Dinic	4	9.8 AC Automaton	23
3.2 Bounded Flow	5		
3.3 MCMF	5	<b>10 Formula</b>	<b>24</b>
3.4 Min Cost Circulation	5	10.1 Recurrences	24
3.5 Gomory Hu	6	10.2 Geometry	24
3.6 ISAP Algorithm	6	10.2.1 Rotation Matrix	24
3.7 Stoer Wagner Algorithm	6	10.2.2 Triangles	24
3.8 Bipartite Matching	6	10.2.3 Quadrilaterals	24
3.9 Kuhn Munkres Algorithm	7	10.2.4 Spherical coordinates	24
3.10 Max Simple Graph Matching	7	10.2.5 Green's Theorem	24
3.11 Stable Marriage	8	10.2.6 Point-Line Duality	24
		10.3 Trigonometry	24
<b>4 Geometry</b>	<b>8</b>	10.4 Derivatives/Integrals	25
4.1 Geometry Template	8	10.5 Sums	25
4.2 Convex Hull	8	10.6 Series	25
4.3 Minimum Enclosing Circle	8		
4.4 Minkowski Sum	9		
4.5 Polar Angle Comparator	9		
4.6 Half Plane Intersection	9		
4.7 Dynamic Convex Hull	9		
4.8 3D Point	10		
4.9 ConvexHull3D	10		
4.10 Circle Operations	10		
4.11 Delaunay Triangulation	11		
4.12 Voronoi Diagram	11		
4.13 Polygon Union	12		
4.14 Tangent Point to Convex Hull	12		
<b>5 Graph</b>	<b>12</b>		
5.1 Block Cut Tree	12		
5.2 2-SAT	12		
5.3 Dominator Tree	13		
5.4 Virtual Tree	13		
5.5 Directed Minimum Spanning Tree	14		
5.6 Fast DMST	14		
5.7 Vizing	14		
5.8 Maximum Clique	15		
5.9 Number of Maximal Clique	15		
5.10 Minimum Mean Cycle	15		
5.11 Minimum Steiner Tree	16		
<b>6 Math</b>	<b>16</b>		
6.1 Extended Euclidean Algorithm	16		
6.2 Floor & Ceil	16		
6.3 Legendre	16		
6.4 Simplex	16		
6.5 Floor Sum	17		
6.6 DiscreteLog	17		
6.7 Miller Rabin & Pollard Rho	17		
6.8 XOR Basis	18		
6.9 Linear Equation	18		
6.10 Chinese Remainder Theorem	18		
6.11 Sqrt Decomposition	18		
<b>7 Misc</b>	<b>18</b>		
7.1 Cyclic Ternary Search	18		
7.2 Matroid	19		

```
//Challenge: Accepted
//#pragma GCC optimize("Ofast")
#include <bits/stdc++.h>
using namespace std;

#define io ios_base::sync_with_stdio(0);cin.tie(0);cerr.tie(0)
#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>
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(){
    io;
}
```

## 1.2 .vimrc

```
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
# -D_GLIBCXX_ASSERTIONS, -D_GLIBCXX_DEBUG
```

## 1.3 Fast IO

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

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

## 1.4 Random

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

## 1.5 Checker

```
#!/usr/bin/env bash
set -e
while ;; do
    python3 gen.py > test.txt
    diff <./a.exe < test.txt) <./b.exe < test.txt)
done
```

## 1.6 PBDS Tree

```
#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)
```

# 2 Data Structure

## 2.1 Heavy-Light Decomposition

```
struct Heavy_light_Decomposition { // 1-base
    int n, up[maxn], dep[maxn], to[maxn], siz[maxn], pa[maxn]
        ];
    int C, ti[maxn], ord[maxn], wdown[maxn], edge[maxn], et =
        0;
    vector<pii> G[maxn];
    void init(int _n) {
        n = _n, C = 0, et = 1;
        for (int i = 1; i <= n; i++)
            G[i].clear(), to[i] = 0;
    }
    void add_edge(int a, int b, int w) {
        G[a].push_back(pii(b, et)), G[b].push_back(pii(a, et));
        edge[et++] = w;
    }
    void dfs(int u, int f, int d) {
        siz[u] = 1, pa[u] = f, dep[u] = d;
        for (auto &v: G[u])
            if (v.ff != f) {
                dfs(v.ff, u, d+1), siz[u] += siz[v];
                if (siz[to[u]] < siz[v]) to[u] = v;
            }
    }
    void cut(int u, int link) {
        ti[u] = C;
        ord[C++] = u, up[u] = link;
        if (!to[u]) return;
        cut(to[u], link);
        for (auto v: G[u]) {
            if (v.ff != pa[u] && v.ff != to[u]) cut(v.ff, v.ff);
        }
    }
    void build() { dfs(1, 1, 1), cut(1, 1); }
    int query(int a, int b) {
        int ta = up[a], tb = up[b], re = 0;
        while (ta != tb)
            if (dep[ta] < dep[tb])
                /*query*/, tb = up[b = pa[tb]];
            else /*query*/, ta = up[a = pa[ta]];
        if (a == b) return re;
        if (ti[a] > ti[b]) swap(a, b);
        /*query*/
        return re;
    }
};
```

## 2.2 Link Cut Tree

```
struct Splay { // LCT + PATH add
    static Splay nil;
```

```

Splay *ch[2], *f;
int rev;
int sz;
ll val, sum, tag;
Splay() : rev(0), sz(1), val(1), sum(1), tag(0) {
    f = ch[0] = ch[1] = &nil;
}
bool isr() { return f->ch[0] != this && f->ch[1] != this;
}
int dir() { return f->ch[0] == this ? 0 : 1; }
void setCh(Splay *c, int d) {
    ch[d] = c;
    if (c != &nil) c->f = this;
    pull();
}
void push() {
    for(int i = 0; i < 2; i++){
        if(ch[i] == &nil) continue;
        if(rev) swap(ch[i]->ch[0], ch[i]->ch[1]), ch[i]->rev
            ^= 1;
        if(tag != 0){
            ch[i]->tag += tag;
            ch[i]->val += tag;
            ch[i]->sum += tag * ch[i]->sz;
        }
    }
    tag = 0;
    rev = 0;
}
void pull() {
    // take care of the nil!
    sz = 1;
    sum = val;
    for(int i = 0; i < 2; i++){
        if(ch[i] == &nil) continue;
        ch[i]->f = this;
        sz += ch[i]->sz;
        sum += ch[i]->sum;
    }
}
void rotate(){
    Splay *p = f;
    int d = dir();
    if (!p->isr()) p->f->setCh(this, p->dir());
    else f = p->f;
    p->setCh(ch[!d], d);
    setCh(p, !d);
    p->pull(), pull();
}
void update(){
    if(f != &nil) f->update();
    push();
}
void splay(){
    update();
    for(Splay* fa; fa = f, !isr(); rotate())
        if(!fa->isr()) (fa->dir() == dir() ? fa : this)->
            rotate();
}
Splay *access(Splay* q = &nil){
    splay();
    setCh(q, 1);
    pull();
    if (f != &nil) return f->access(this);
    else return q;
}
void root_path(){access(), splay();}
void chroot() {root_path(), swap(ch[0], ch[1]), rev = 1,
    push(), pull();}
void split(Splay* y){chroot(), y->root_path();}
void link(Splay* y){root_path(), y->chroot(), setCh(y, 1)
    ;}
void cut(Splay* y) {split(y), y->push(), y->ch[0] = y->ch
    [0]->f = &nil;}
Splay *get_root(){
    root_path();

```

```

    auto q = this;
    for(; q->ch[0] != &nil; q = q->ch[0]) q->push();
    return q;
}
Splay *lca(Splay* y){
    access(), y->root_path();
    return y->f == &nil ? &nil : y->f;
}
bool conn(Splay* y){return get_root() == y->get_root();}
} Splay::nil;

```

## 2.3 Treap

```

struct node {
    int data, sz;
    node *l, *r;
    node(int k) : data(k), sz(1), l(0), r(0) {}
    void up() {
        sz = 1;
        if (l) sz += l->sz;
        if (r) sz += r->sz;
    }
    void down() {}
};
int sz(node *a) { return a ? a->sz : 0; }
node *merge(node *a, node *b) {
    if (!a || !b) return a ? a : b;
    if (rand() % (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;
}
void split(node *o, node *&a, node *&b, int 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();
}
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, int k) {
    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, int 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, int k) {
    if (!o) return 0;
    if (o->data == k) {
        node *t = o;
        o->down(), o = merge(o->l, o->r);
        delete t;
        return 1;
    }
    node *&t = k < o->data ? o->l : o->r;
    return erase(t, k) ? o->up(), 1 : 0;
}
void insert(node *&o, int k) {
    node *a, *b;
    split(o, a, b, k),
        o = merge(a, merge(new node(k), b));
}
void interval(node *&o, int l, int r) {
    node *a, *b, *c;

```

```

split2(o, a, b, l - 1), split2(b, b, c, r);
// operate
o = merge(a, merge(b, c));
}

```

## 2.4 KD Tree

```

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

```

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;
}

```

## 3 Flow & Matching

### 3.1 Dinic

```

struct Dinic { //  $\theta$ -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

```

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

```

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

```

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

```

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 ISAP Algorithm

```

struct Maxflow { //to be modified
    static const int MAXV = 20010;
    static const int INF = 1000000;
    struct Edge {
        int v, c, r;
        Edge(int _v, int _c, int _r)
            : v(_v), c(_c), r(_r) {}
    };
    int s, t;
    vector<Edge> G[MAXV * 2];
    int iter[MAXV * 2], d[MAXV * 2], gap[MAXV * 2], tot;
    void init(int x) {
        tot = x + 2;
        s = x + 1, t = x + 2;
        for (int i = 0; i <= tot; i++) {
            G[i].clear();
            iter[i] = d[i] = gap[i] = 0;
        }
    }
    void addEdge(int u, int v, int c) {
        G[u].push_back(Edge(v, c, SZ(G[v])));
        G[v].push_back(Edge(u, 0, SZ(G[u]) - 1));
    }
    int dfs(int p, int flow) {
        if (p == t) return flow;
        for (int &i = iter[p]; i < SZ(G[p]); i++) {
            Edge &e = G[p][i];
            if (e.c > 0 && d[p] == d[e.v] + 1) {
                int f = dfs(e.v, min(flow, e.c));
                if (f) {
                    e.c -= f;
                    G[e.v][e.r].c += f;
                    return f;
                }
            }
        }
        return 0;
    }
};

```

```

    if ((--gap[d[p]]) == 0) d[s] = tot;
    else {
        d[p]++;
        iter[p] = 0;
        ++gap[d[p]];
    }
    return 0;
}

int solve() {
    int res = 0;
    gap[0] = tot;
    for (res = 0; d[s] < tot; res += dfs(s, INF))
        ;
    return res;
}
} flow;

```

### 3.7 Stoer Wagner Algorithm

```

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;
    }
};

```

### 3.8 Bipartite Matching

```

//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>());
    }
};

```

### 3.9 Kuhn Munkres Algorithm

```

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;
}
};

```

### 3.10 Max Simple Graph Matching

```

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] = l;
        }
    }
    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
};

```

### 3.11 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

```

## 4 Geometry

### 4.1 Geometry Template

```

using ld = ll;
using pdd = pair<ld, ld>;
using Line = pair<pdd, pdd>;
#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(Line a, Line b){
    pdd p1, p2, p3, p4;
    tie(p1, p2) = a; tie(p3, p4) = b;
    if(btw(p1, p3, p4) || btw(p2, p3, p4) || btw(p3, p1, p2)
    || btw(p4, p1, p2))

```

```

    return true;
    return ori(p1, p2, p3) * ori(p1, p2, p4) < 0 &&
    ori(p3, p4, p1) * ori(p3, p4, p2) < 0;
}
pdd intersect(Line a, Line b){
    pdd p1, p2, p3, p4;
    tie(p1, p2) = a; tie(p3, p4) = b;
    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 Convex Hull

```

vector<int> getConvexHull(vector<pdd>& pts){
    vector<int> id(SZ(pts));
    iota(iter(id), 0);
    sort(iter(id), [&](int x, int y){ return pts[x] < pts[y]; });
    vector<int> hull;
    for(int tt = 0; tt < 2; tt++){
        int sz = SZ(hull);
        for(int j : id){
            pdd p = pts[j];
            while(SZ(hull) - sz >= 2 &&
                cross(pts[hull.back()] - pts[hull[SZ(hull) - 2]],
                    p - pts[hull[SZ(hull) - 2]]) <= 0)
                hull.pop_back();
            hull.pb(j);
        }
        hull.pop_back();
        reverse(iter(id));
    }
    return hull;
}

```

### 4.3 Minimum Enclosing Circle

```

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};
}

```



## 4.4 Minkowski Sum

```
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;
}
```

## 4.5 Polar Angle Comparator

```
// -1: a // b (if same), 0/1: a < b
int cmp(p11 a, p11 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;
}
```

## 4.6 Half Plane Intersection

```
// 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));
}
```

## 4.7 Dynamic Convex Hull

```
struct Line{
    ll a, b, l = MIN, r = MAX;
    Line(ll a, ll b): a(a), b(b) {}
    ll operator()(ll x) const{
        return a * x + b;
    }
    bool operator<(Line b) const{
        return a < b.a;
    }
    bool operator<(ll b) const{
        return r < b;
    }
};

ll iceil(ll a, ll b){
    if(b < 0) a *= -1, b *= -1;
    if(a > 0) return (a + b - 1) / b;
    else return a / b;
}

ll intersect(Line a, Line b){
    return iceil(a.b - b.b, b.a - a.a);
}

struct DynamicConvexHull{
    multiset<Line, less<>> ch;

    void add(Line ln){
        auto it = ch.lower_bound(ln);
        while(it != ch.end()){
            Line tl = *it;
            if(tl(tl.r) <= ln(tl.r)){
                it = ch.erase(it);
            }
            else break;
        }
        auto it2 = ch.lower_bound(ln);
        while(it2 != ch.begin()){
            Line tl = *prev(it2);
            if(tl(tl.l) <= ln(tl.l)){
                it2 = ch.erase(prev(it2));
            }
            else break;
        }
        it = ch.lower_bound(ln);
        if(it != ch.end()){
            Line tl = *it;
            if(tl(tl.l) >= ln(tl.l)) ln.r = tl.l - 1;
            else{
                ll pos = intersect(ln, tl);
                tl.l = pos;
                ln.r = pos - 1;
                ch.erase(it);
                ch.insert(tl);
            }
        }
        it2 = ch.lower_bound(ln);
        if(it2 != ch.begin()){
            Line tl = *prev(it2);
            if(tl(tl.r) >= ln(tl.r)) ln.l = tl.r + 1;
            else{
                ll pos = intersect(tl, ln);
                tl.r = pos;
                ln.l = pos;
                ch.erase(prev(it2));
            }
        }
    }
}
```

```

        ch.insert(tl);
    }
}
if(ln.l <= ln.r) ch.insert(ln);
}

ll query(ll pos){
    auto it = ch.lower_bound(pos);
    if(it == ch.end()) return 0;
    return (*it)(pos);
}
};

```

## 4.8 3D Point

```

// 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;
}

```

## 4.9 ConvexHull3D

```

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

```

## 4.10 Circle Operations

```

// from 8BQube
const double PI=acos(-1);

```

```

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};
}

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(O,poly
            [i],poly[(i+1)%SZ(poly)]);
    return fabs(S);
}

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;
}

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;
}

```

## 4.11 Delaunay Triangulation

*/\* 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
    pll p[N];
    void init(int _n, pll _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) {
            pll 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[t ^ 1] - 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) {
            pll 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;

```

## 4.12 Voronoi Diagram

*// 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));
    }
}

```

### 4.13 Polygon Union

```

// 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;
}

```

### 4.14 Tangent Point to Convex Hull

```

// 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

```

## 5 Graph

### 5.1 Block Cut Tree

```

struct BCC{
    vector<int> v, e, cut;
};
struct BlockCutTree{ // 0-based, allow multi edges but not
    allow loops

```

```

    int n, m, cnt = 0;
    // n:|V|, m:|E|, cnt:|bcc|
    vector<BCC> bcc;
    vector<vector<pii>> g; // original graph
    vector<pii> edges; // 0-based
    vector<vector<int>> vbcc;
    // vbcc[i] = BCCs containing vertex i, vbcc[i].size()>1
    // iff i is an articulation
    vector<int> ebcc;
    // edge i is a bridge iff bcc[ebcc[i]].e.size() == 1
    // block cut tree:
    // adj[bcc i]: bcc[i].cut
    // adj[cut i]: vbcc[i]

    BlockCutTree(int _n, vector<pii> _edges):
        n(_n), m(SZ(_edges)), g(_n), edges(_edges), vbcc(_n),
        ebcc(SZ(_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 build(){
        vector<int> in(n, -1), low(n, -1);
        vector<vector<int>> up(n);
        vector<int> stk;
        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)){
                bcc.pb();
                while(true){
                    int v = stk.back();
                    stk.pop_back();
                    vbcc[v].pb(cnt);
                    bcc[cnt].v.pb(v);
                    for(int e : up[v]){
                        ebcc[e] = cnt;
                        bcc[cnt].e.pb(e);
                    }
                    if(v == now) break;
                }
                if(now != par){
                    vbcc[par].pb(cnt);
                    bcc[cnt].v.pb(par);
                }
                cnt++;
            }
        };
        for(int i = 0; i < n; i++){
            if(in[i] == -1) _dfs(_dfs, i, i, -1);
        }
        for(int i = 0; i < cnt; i++){
            for(int j : bcc[i].v)
                if(SZ(vbcc[j]) > 1) bcc[i].cut.pb(j);
        }
    }
};

```

### 5.2 2-SAT

```

struct SAT{ // 0-based, [n, 2n) is neg of [0, n)

```

```

int n;
vector<vector<int>> g, rg;
bool ok = true;
vector<bool> ans;

void init(int _n){
    n = _n;
    g.resize(2 * n);
    rg.resize(2 * n);
    ans.resize(n);
}

int neg(int v){
    return v < n ? v + n : v - n;
}

void addEdge(int u, int v){
    g[u].pb(v);
    rg[v].pb(u);
}

void addClause(int a, int b){
    addEdge(neg(a), b);
    addEdge(neg(b), a);
}

void build(){
    vector<bool> vst(2 * n, false);
    vector<int> tmp, scc(2 * n, -1);
    int cnt = 1;
    function<void(int)> dfs = [&](int now){
        vst[now] = true;
        for(int i : rg[now]){
            if(vst[i]) continue;
            dfs(i);
        }
        tmp.pb(now);
    };
    for(int i = 0; i < 2 * n; i++){
        if(!vst[i]) dfs(i);
    }
    reverse(all(tmp));
    function<void(int, int)> dfs2 = [&](int now, int id){
        scc[now] = id;
        for(int i : g[now]){
            if(scc[i] != -1) continue;
            dfs2(i, id);
        }
    };
    for(int i : tmp){
        if(scc[i] == -1) dfs2(i, cnt++);
    }
    debug(scc);
    for(int i = 0; i < n; i++){
        if(scc[i] == scc[neg(i)]){
            ok = false;
            return;
        }
        if(scc[i] < scc[neg(i)]) ans[i] = true;
        else ans[i] = false;
    }
}

};

```

### 5.3 Dominator Tree

```

// copy from 8BQube
struct dominator_tree { // 1-base
    vector<int> G[N], rG[N];
    int n, pa[N], dfn[N], id[N], Time;
    int semi[N], idom[N], best[N];
    vector<int> tree[N]; // dominator_tree
    void init(int _n) {
        n = _n;
        for (int i = 1; i <= n; ++i)
            G[i].clear(), rG[i].clear();
    }
    void add_edge(int u, int v) {
        G[u].pb(v), rG[v].pb(u);
    }

```

```

}
void dfs(int u) {
    id[dfn[u] = ++Time] = u;
    for (auto v : G[u])
        if (!dfn[v]) dfs(v), pa[dfn[v]] = dfn[u];
}
int find(int y, int x) {
    if (y <= x) return y;
    int tmp = find(pa[y], x);
    if (semi[best[y]] > semi[best[pa[y]]])
        best[y] = best[pa[y]];
    return pa[y] = tmp;
}
void tarjan(int root) {
    Time = 0;
    for (int i = 1; i <= n; ++i) {
        dfn[i] = idom[i] = 0;
        tree[i].clear();
        best[i] = semi[i] = i;
    }
    dfs(root);
    for (int i = Time; i > 1; --i) {
        int u = id[i];
        for (auto v : rG[u])
            if (v = dfn[v]) {
                find(v, i);
                semi[i] = min(semi[i], semi[best[v]]);
            }
        tree[semi[i]].pb(i);
        for (auto v : tree[pa[i]]) {
            find(v, pa[i]);
            idom[v] =
                semi[best[v]] == pa[i] ? pa[i] : best[v];
        }
        tree[pa[i]].clear();
    }
    for (int i = 2; i <= Time; ++i) {
        if (idom[i] != semi[i]) idom[i] = idom[idom[i]];
        tree[id[idom[i]]].pb(id[i]);
    }
}
};

```

### 5.4 Virtual Tree

```

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

void insert(int u) {
    if (top == -1) return st[++top] = u, void();
    int p = LCA(st[top], u);
    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(v[0]);
}

```

## 5.5 Directed Minimum Spanning Tree

```

const ll INF = LLONG_MAX;
struct edge{
    int u = -1, v = -1;
    ll w = INF;
    int id = -1;
};

// 0-based, E[i].id = i
bool DMST(int n, vector<edge> &E, int root, vector<edge> &
sol){
    vector<int> id(n), vis(n);
    vector<edge> in(n);
    for(edge e : E)
        if(e.u != e.v && e.w < in[e.v].w && e.v != root)
            in[e.v] = e;
    for(int i = 0; i < n; i++)
        if(i != root && in[i].u == -1) return false; // no sol
    int cnt = 0;
    fill(iter(id), -1); fill(iter(vis), -1);
    for(int u = 0; u < n; u++){
        int v = u;
        while(vis[v] != u && id[v] == -1 && in[v].u != -1)
            vis[v] = u, v = in[v].u;
        if(v != root && id[v] == -1){
            for(int x = in[v].u; x != v; x = in[x].u)
                id[x] = cnt;
            id[v] = cnt++;
        }
    }
    if(!cnt) return sol = in, true; // no cycle
    for(int u = 0; u < n; u++)
        if(id[u] == -1) id[u] = cnt++;
    vector<edge> nE;
    for(int i = 0; i < SZ(E); i++){
        edge tmp = E[i];
        tmp.u = id[tmp.u], tmp.v = id[tmp.v];
        if(in[E[i].v].w != INF) tmp.w -= in[E[i].v].w;
        nE.pb(tmp);
    }
    vector<edge> tsol;
    if(!DMST(cnt, nE, id[root], tsol)) return false;
    sol.resize(n);
    for(int i = 0; i < cnt; i++){
        if(i == id[root]) continue;
        int t = tsol[i].id;
        sol[E[t].v] = E[t];
    }
    for(int i = 0; i < n; i++)
        if(sol[i].id == -1) sol[i] = in[i];
    return true;
}

```

## 5.6 Fast DMST

```

struct Edge { int a, b; ll w; };
struct Node { /// lazy skew heap node
    Edge key;
    Node *l, *r;
    ll delta;
    void prop() {
        key.w += delta;
        if (l) l->delta += delta;
        if (r) r->delta += delta;
        delta = 0;
    }
    Edge top() { prop(); return key; }
};

Node *merge(Node *a, Node *b) {
    if (!a || !b) return a ? b : a;
    a->prop(), b->prop();
    if (a->key.w > b->key.w) swap(a, b);
    swap(a->l, (a->r = merge(b, a->r)));
    return a;
}

```

```

void pop(Node*& a) { a->prop(); a = merge(a->l, a->r); }

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

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

```

## 5.7 Vizing

```

// 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

```

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

```

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

```

// 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

```

// 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;
    }
};

```

## 6 Math

### 6.1 Extended Euclidean Algorithm

```

//  $ax+ny = 1, ax+ny == ax == 1 \pmod n$ 
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

```

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

```

### 6.3 Legendre

```

// the Jacobi symbol is a generalization of the Legendre
// symbol,
// such that the bottom doesn't need to be prime.
//  $(n/p) \rightarrow$  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;
    const int jc = Jacobi(a, p);
    if (jc == 0) return 0;
    if (jc == -1) return -1;
    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

```

#pragma once

typedef double T; // Long double, Rational, double + mod<P>
>...
typedef vector<T> vd;
typedef vector<vd> vvd;

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

```



```

struct LPSolver {
    int m, n;
    vi 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 Floor Sum

```

// from 8BQube
ll floor_sum(ll n, ll m, ll a, ll b) {
    ll ans = 0;
    if (a >= m)
        ans += (n - 1) * n * (a / m) / 2, a %= m;
    if (b >= m)
        ans += n * (b / m), b %= m;
    ll y_max = (a * n + b) / m, x_max = (y_max * m - b);
    if (y_max == 0) return ans;
    ans += (n - (x_max + a - 1) / a) * y_max;
    ans += floor_sum(y_max, a, m, (a - x_max % a) % a);
}

```

```

    return ans;
} // sum^{n-1}_0 floor((a * i + b) / m) in log(n + m + a + b)
)

```

## 6.6 DiscreteLog

```

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

```

// 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

```

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

```

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] /= lead;
        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[rk][k] * tmp;
        }
        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];
            homo.pb(h);
        }
    }
}

```

## 6.10 Chinese Remainder Theorem

```

p11 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

```

// 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, r = n; r >= 1; r = l - 1){
    int x = iceil(n, r);
    l = iceil(n, x);
}

```

## 7 Misc

### 7.1 Cyclic Ternary Search

```

/* 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(l, r % n) ? l : r % n;
}

```

## 7.2 Matroid

我們稱一個二元組  $M = (E, \mathcal{I})$  為一個擬陣，其中  $\mathcal{I} \subseteq 2^E$  為  $E$  的子集所形成的非空集合，若：

- 若  $S \in \mathcal{I}$  以及  $S' \subseteq S$ ，則  $S' \in \mathcal{I}$
- 對於  $S_1, S_2 \in \mathcal{I}$  滿足  $|S_1| < |S_2|$ ，存在  $e \in S_2 \setminus S_1$  使得  $S_1 \cup \{e\} \in \mathcal{I}$

除此之外，我們有以下的定義：

- 位於  $\mathcal{I}$  中的集合我們稱之為獨立集 (independent set)，反之不在  $\mathcal{I}$  中的我們稱為相依集 (dependent set)
- 極大的獨立集為基底 (base)、極小的相依集為迴路 (circuit)
- 一個集合  $Y$  的秩 (rank)  $r(Y)$  為該集合中最大的獨立子集，也就是  $r(Y) = \max\{|X| \mid X \subseteq Y \text{ 且 } X \in \mathcal{I}\}$

性質：

1.  $X \subseteq Y \wedge Y \in \mathcal{I} \implies X \in \mathcal{I}$
2.  $X \subseteq Y \wedge X \notin \mathcal{I} \implies Y \notin \mathcal{I}$
3. 若  $B$  與  $B'$  皆是基底且  $B \subseteq B'$ ，則  $B = B'$   
若  $C$  與  $C'$  皆是迴路且  $C \subseteq C'$ ，則  $C = C'$
4.  $e \in E \wedge X \subseteq E \implies r(X) \leq r(X \cup \{e\}) \leq r(X) + 1$  i.e. 加入一個元素後秩不會降底，最多增加 1
5.  $\forall Y \subseteq E, \exists X \subseteq Y, r(X) = |X| = r(Y)$

一些等價的性質：

1. 對於所有  $X \subseteq E$ ， $X$  的極大獨立子集都有相同的大小
2. 對於  $B_1, B_2 \in \mathcal{B} \wedge B_1 \neq B_2$ ，對於所有  $e_1 \in B_1 \setminus B_2$ ，存在  $e_2 \in B_2 \setminus B_1$  使得  $(B_1 \setminus \{e_1\}) \cup \{e_2\} \in \mathcal{B}$
3. 對於  $X, Y \in \mathcal{I}$  且  $|X| < |Y|$ ，存在  $e \in Y \setminus X$  使得  $X \cup \{e\} \in \mathcal{B}$
4. 如果  $r(X \cup \{e_1\}) = r(X \cup \{e_2\}) = r(X)$ ，則  $r(X \cup \{e_1, e_2\}) = r(X)$ 。  
如果  $r(X \cup \{e\}) = r(X)$  對於所有  $e \in E'$  都成立，則  $r(X \cup E') = r(X)$ 。

擬陣交

Data: 兩個擬陣  $M_1 = (E, \mathcal{I}_1)$  以及  $M_2 = (E, \mathcal{I}_2)$   
Result:  $I$  為最大的位於  $\mathcal{I}_1 \cap \mathcal{I}_2$  中的獨立集  
 $I \leftarrow \emptyset$   
 $X_1 \leftarrow \{e \in E \setminus I \mid I \cup \{e\} \in \mathcal{I}_1\}$   
 $X_2 \leftarrow \{e \in E \setminus I \mid I \cup \{e\} \in \mathcal{I}_2\}$   
**while**  $X_1 \neq \emptyset$  且  $X_2 \neq \emptyset$  **do**  
    **if**  $e \in X_1 \cap X_2$  **then**  
         $I \leftarrow I \cup \{e\}$   
    **else**  
        構造交換圖  $\mathcal{D}_{M_1, M_2}(I)$   
        在交換圖上找到一條  $X_1$  到  $X_2$  且沒有捷徑的路徑  $P$   
         $I \leftarrow I \Delta P$   
    **end if**  
     $X_1 \leftarrow \{e \in E \setminus I \mid I \cup \{e\} \in \mathcal{I}_1\}$   
     $X_2 \leftarrow \{e \in E \setminus I \mid I \cup \{e\} \in \mathcal{I}_2\}$   
**end while**

## 8 Polynomial

### 8.1 FWHT

```

/* 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];
}

```

### 8.2 FFT

// 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);
};

```

### 8.3 NTT

```

//(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;
    }
}
};

```

## 8.4 Polynomial Operation

```

// Copy from 8BQube
#define fi(s, n) for (int i = (int)(s); i < (int)(n); ++i)
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()), *this; }
    Poly& isz(int m) { return resize(m), *this; }
    Poly& iadd(const Poly &rhs) { // n() == rhs.n()
        fi(0, n()) if (((*this)[i] += rhs[i]) >= P) (*this)[i] -= P;
        return *this;
    }
    Poly& imul(ll k) {
        fi(0, n()) (*this)[i] = (*this)[i] * k % P;
        return *this;
    }
    Poly Mul(const Poly &rhs) const {
        int m = 1;
        while (m < n() + rhs.n() - 1) m <= 1;
        assert(m <= MAXN);
        Poly X(*this, 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 { // (*this)[0] != 0, 1e5/95ms, 2*sz<=MAXN
        if (n() == 1) return {ntt.minv((*this)[0])};
        int m = 1;
        while (m < n() * 2) m <= 1;
        assert(m <= MAXN);
        Poly Xi = Poly(*this, (n() + 1) / 2).Inv().isz(m);
        Poly Y(*this, 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 = this->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] = ntt.minv(fc[i]);
        }
        for (int i = 0; i < n; i++) (*this)[i] = (*this)[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;
        *this = (*this).irev().Mul(g).isz(n).irev();
        for (int i = 0; i < n; i++) (*this)[i] = (*this)[i] * ifc[i] % P;
        return *this;
    }
    Poly shift(const ll &c) const { return Poly(*this).shift_inplace(c); }
    Poly _Sqrt() const { // Jacobi((*this)[0], P) = 1
        if (n() == 1) return {QuadraticResidue((*this)[0], P)};
        Poly X = Poly(*this, (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 ((*this)[i]) has = 1;
            if (has) a.push_back((*this)[i]);
        }
        if (!has) return *this;
        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}, *this};
        const int m = n() - rhs.n() + 1;
        Poly X(rhs); X.irev().isz(m);
        Poly Y(*this); Y.irev().isz(m);
        Poly Q = Y.Mul(X.Inv()).isz(m).irev();
        X = rhs.Mul(Q), Y = *this;
        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) * (*this)[i + 1] % P;
        return ret.isz(max(1, ret.n()));
    }
    Poly Sx() const {
        Poly ret(n() + 1);
        fi(0, n()) ret[i + 1] = ntt.minv(i + 1) * (*this)[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, *this);
    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] = {(x[i] ? P - x[i] : 0), 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] * ntt.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 { // (*this)[0] == 1, 2*sz<=MAXN
    return Dx().Mul(Inv()).Sx().isz(n());
}
Poly Exp() const { // (*this)[0] == 0, 2*sz<=MAXN
    if (n() == 1) return {1};
    Poly X = Poly(*this, (n() + 1) / 2).Exp().isz(n());
    Poly Y = X.Ln(); Y[0] = P - 1;
    fi(0, n()) if ((Y[i] = (*this)[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() && !(*this)[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();
}
static ll LinearRecursion(const vector<ll> &a, const
vector<ll> &coef, ll n) { // a_n = \sum c_j a_{n-j}
    const int k = (int)a.size();
    assert((int)coef.size() == k + 1);
    Poly C(k + 1), W(Poly {1}, k), M = {0, 1};
    fi(1, k + 1) C[k - i] = coef[i] ? P - coef[i] : 0;
    C[k] = 1;
    while (n) {
        if (n % 2) W = W.Mul(M).DivMod(C).second;
        n /= 2, M = M.Mul(M).DivMod(C).second;
    }
    ll ret = 0;
    fi(0, k) ret = (ret + W[i] * a[i]) % P;
    return ret;
}
};
#undef fi
using Poly_t = Poly<1 << 20, 998244353, 3>;
// template<> decltype(Poly_t::ntt) Poly_t::ntt = {};

```

## 8.5 Generating Function

### 8.5.1 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:  $f(x) = \frac{1-\sqrt{1-4x}}{2x}$

### 8.5.2 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
- $\hat{C}(x) = \hat{A}(x)^k$ :  $\sum_{i_1+i_2+\dots+i_k=n} \binom{n}{i_1, i_2, \dots, i_k} a_{i_1} a_{i_2} \dots a_{i_k}$  的指數生成函數
- $\hat{C}(x) = \exp(\hat{A}(x))$ : 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

## 8.6 Bostan Mori

```

NTT<262144, 998244353, 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);
    P.resize(d, 0);
    Q.resize(d + 1, 0);
    int sz = (2 * d + 1 == 1 ? 2 : (1 << (__lg(2 * d) + 1)));
    while(k) {
        vector<ll> Qneg(sz);
        for(int i = 0; i < (int)Q.size(); i++){
            Qneg[i] = Q[i] * ((i & 1) ? -1 : 1);
            if(Qneg[i] < 0) Qneg[i] += mod;
        }
        ntt(Qneg, sz, false);

        vector<ll> U(sz), V(sz);
        for(int i = 0; i < (int)P.size(); i++)
            U[i] = P[i];
        for(int i = 0; i < (int)Q.size(); i++)
            V[i] = Q[i];

        ntt(U, sz, false);
        ntt(V, sz, false);
        for(int i = 0; i < sz; i++)
            U[i] = U[i] * Qneg[i] % mod;
        for(int i = 0; i < sz; i++)
            V[i] = V[i] * Qneg[i] % mod;
        ntt(U, sz, true);
        ntt(V, sz, true);
    }
}

```

```

    for(int i = k & 1; i <= 2 * d - 1; i += 2)
        P[i >> 1] = U[i];
    for(int i = 0; i <= 2 * d; i += 2)
        Q[i >> 1] = V[i];
    k >>= 1;
}
return P[0] * ntt.minv(Q[0]) % mod;
}

```

## 9 String

### 9.1 KMP Algorithm

```

void kmp(string s){
    int siz = s.size();
    vector<int> f(siz, 0);
    f[0] = 0;
    for (int i = 1; i < siz; i++) {
        f[i] = f[i-1];
        bool zero = 0;
        while (s[f[i]] != s[i]) {
            if (f[i] == 0) {
                zero = 1;
                break;
            }
            f[i] = f[f[i]-1];
        }
        if (!zero) f[i]++;
    }
}

```

### 9.2 Manacher Algorithm

```

vector<int> manacher(string s) {
    int n = s.size();
    vector<int> v(n);
    int pnt = -1, len = 1;
    for (int i = 0; i < n; i++) {
        int cor = 2 * pnt - i;
        if (cor >= 0) v[i] = min(v[cor], cor - pnt + len);
        while (i+v[i] < n && i-v[i] >= 0 && s[i+v[i]] == s[i-v[i]]) v[i]++;
        if (i + v[i] >= pnt + len) pnt = i, len = v[i];
    }
    for (int i = 0; i < n; i++) v[i] = 2 * v[i] - 1;
    return v;
}

```

### 9.3 MCP

```

string mcp(string s) { //Duval algorithm for Lyndon
    factorization
    s += s;
    int n = s.size(), i = 0, ans = 0;
    while (i < n/2) {
        ans = i;
        int j = i+1, k=i;
        while (j < n && s[k] <= s[j]) {
            if (s[k] < s[j]) k = i;
            else k++;
            j++;
        }
        while (i <= k) i += j - k;
    }
    return s.substr(ans, n/2);
}

```

### 9.4 Suffix Array

```

struct SuffixArray { //tested
    vector<int> sa, lcp, rank; //lcp[i] is lcp of sa[i] and
    sa[i-1]
    //sa[0] = s.size();
    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.resize(n);
        for (int i = 0; i < n-1; i++) x[i] = (int)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++);
    }
};

```

### 9.5 Suffix Automaton

```

// from 8BQube
// at most 2n-1 states, 3n-4 edges

// to find longest common substring for multiple strings
    S_1, ..., S_k
// assign a special (distinct) character D_i to each string
// let T = S_1 D_1 ... S_k D_k, then build SAM of T
// answer is state with max length reachable to all D_i
const int maxn = 1000010;
struct SAM { //1 base
    vector<int> adj[maxn];
    int tot, root, lst, par[maxn], mx[maxn], fi[maxn], iter;
    //mx: maxlen of node, mx[par[i]]+1: minlen of node
    //fi: first endpos
    //corresponding substring of node can be found by fi and
        mx
    int nxt[maxn][33];
    int newNode() {
        int res = ++tot;
        fill(nxt[res], nxt[res] + 33, 0);
        par[res] = mx[res] = 0;
        fi[res] = iter;
        return res;
    }
    void init() {
        tot = 0;
        iter = 0;
        root = newNode();
        par[root] = 0, mx[root] = 0;
        lst = root;
    }
    void push(int c) {
        int p = lst;
        int np = newNode();
        mx[np] = mx[p] + 1;
        for (; p && nxt[p][c] == 0; p = par[p])
            nxt[p][c] = np;
        if (p == 0) par[np] = root;
        else {
            int q = nxt[p][c];
            if (mx[p] + 1 == mx[q]) par[np] = q;

```

```

    else {
        int nq = newNode();
        mx[nq] = mx[p] + 1;
        for (int i = 0; i < 33; i++)
            nxt[nq][i] = nxt[q][i];
        par[nq] = par[q];
        fi[nq] = fi[q];
        par[q] = nq;
        par[np] = nq;
        for (; p && nxt[p][c] == q; p = par[p])
            nxt[p][c] = nq;
    }
}
lst = np;
}

void push(string str) {
    for (int i = 0; str[i]; i++) {
        iter++;
        push(str[i] - 'a' + 1);
    }
}

ll get_diff_strings(){
    ll tot = 0;
    for(int i = 1; i <= tot; i++) tot += mx[i] - mx[par[i]
    ];
    return tot;
}

bool in[maxn];
int cnt[maxn]; //cnt is number of occurences of node
void count() {
    for (int i = 1; i <= tot; ++i)
        ++in[par[i]];
    queue<int> q;
    for (int i = 1; i <= tot; ++i)
        if (!in[i]) q.push(i);
    while (!q.empty()) {
        int u = q.front();
        q.pop();
        cnt[par[u]] += cnt[u];
        if (!--in[par[u]])
            q.push(par[u]);
    }
}
} sam;

```

## 9.6 Z-value Algorithm

```

vector<int> z_function(string const& 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;
}

```

## 9.7 Main Lorentz

```

vector<int> z_function(string const& s);
int get_z(vector<int> const& z, int i) {
    return (0 <= i && i < SZ(z)) ? z[i] : 0;
}

vector<pair<int, int>> rep;

void convert_to_rep(int shift, bool left, int cntr, int l,
    int k1, int k2) {
    int lef = max(1, l-k2), rig = min(1, k1);
    int minl, maxl;
    if (left) {
        rig = min(rig, l-1);

```

```

        minl = shift + cntr - rig, maxl = shift+cntr-lef;
    } else {
        minl = shift + cntr - 1 - rig + 1, maxl = shift + cntr
        - 1 - lef + 1;
    }
    //left endpoint: [minl, maxl], length: 2*L
}

void find_rep(string s, int shift = 0) {
    int n = s.size();
    if (n == 1) return;

    int nu = n / 2;
    int nv = n - nu;
    string u = s.substr(0, nu);
    string v = s.substr(nu);
    string ru(u.rbegin(), u.rend());
    string rv(v.rbegin(), v.rend());

    find_rep(u, shift);
    find_rep(v, shift + nu);

    vector<int> z1 = z_function(ru);
    vector<int> z2 = z_function(v + '#' + u);
    vector<int> z3 = z_function(ru + '#' + rv);
    vector<int> z4 = z_function(v);

    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)
            convert_to_rep(shift, cntr < nu, cntr, l, k1, k2);
    }
}

```

## 9.8 AC Automaton

```

const int maxn = 300005, maxc = 26;
struct AC_Automaton { //1-base
    int nx[maxn][maxc], fl[maxn], cnt[maxn], pri[maxn], tot;
    //pri: bfs order of trie (0-base)
    int newnode() {
        tot++;
        fill(nx[tot], nx[tot] + maxc, -1);
        return tot;
    }
    void init() { tot = 0, newnode(); }
    int input(string &s) { // return the end_node of string
        int X = 1;
        for (char c : s) {
            if (!nx[X][c - 'a']) nx[X][c - 'a'] = newnode();
            X = nx[X][c - 'a'];
        }
        return X;
    }
    void make_fl() { //fail link
        queue<int> q;
        q.push(1), fl[1] = 0;
        for (int t = 0; !q.empty(); t) {
            int R = q.front();
            q.pop(), pri[t++] = R;
            for (int i = 0; i < maxc; ++i)
                if (~nx[R][i]) {
                    int X = nx[R][i], Z = fl[R];
                    for (; Z && !~nx[Z][i];) Z = fl[Z];
                    fl[X] = Z ? nx[Z][i] : 1, q.push(X);
                }
        }
    }
}

```

```

}
void get_v(string &s) {
    //number of times prefix appears in strings
    int X = 1;
    fill(cnt, cnt + tot+1, 0);
    for (char c : s) {
        while (X && !~nx[X][c - 'a']) X = fl[X];
        X = X ? nx[X][c - 'a'] : 1, ++cnt[X];
    }
    for (int i = tot-1; i > 0; --i)
        cnt[fl[pri[i]]] += cnt[pri[i]];
    }
} ac;

```

## 10 Formula

### 10.1 Recurrences

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

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

Non-distinct roots  $r$  become polynomial factors, e.g.  $a_n = (d_1 n + d_2) r^n$ .

## 10.2 Geometry

### 10.2.1 Rotation Matrix

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

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

### 10.2.2 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):  $m_a = \frac{1}{2} \sqrt{2b^2 + 2c^2 - a^2}$

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

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

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

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

Incenter:

$$P_1 = (x_1, y_1), P_2 = (x_2, y_2), P_3 = (x_3, y_3)$$

$$s_1 = P_2 P_3, s_2 = P_1 P_3, s_3 = P_1 P_2$$

$$\frac{s_1 P_1 + s_2 P_2 + s_3 P_3}{s_1 + s_2 + s_3}$$

Circumcenter:

$$P_0 = (0, 0), P_1 = (x_1, y_1), P_2 = (x_2, y_2)$$

$$x_c = \frac{1}{2} \times \frac{y_2(x_1^2 + y_1^2) - y_1(x_2^2 + y_2^2)}{-x_2 y_1 + x_1 y_2}$$

$$y_c = \frac{1}{2} \times \frac{x_2(x_1^2 + y_1^2) - x_1(x_2^2 + y_2^2)}{-x_1 y_2 + x_2 y_1}$$

Check if  $(x_0, y_0)$  is in the circumcircle:

$$\begin{vmatrix} x_1 - x_0 & y_1 - y_0 & (x_1^2 + y_1^2) - (x_0^2 + y_0^2) \\ x_2 - x_0 & y_2 - y_0 & (x_2^2 + y_2^2) - (x_0^2 + y_0^2) \\ x_3 - x_0 & y_3 - y_0 & (x_3^2 + y_3^2) - (x_0^2 + y_0^2) \end{vmatrix}$$

0: on edge,  $> 0$ : inside,  $< 0$ : outside

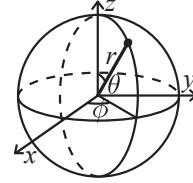
### 10.2.3 Quadrilaterals

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  $A = \sqrt{(p-a)(p-b)(p-c)(p-d)}$ .

### 10.2.4 Spherical coordinates



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

### 10.2.5 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:

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

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

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

### 10.2.6 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

## 10.3 Trigonometry

$$\sinh x = \frac{1}{2}(e^x - e^{-x}) \quad \cosh x = \frac{1}{2}(e^x + e^{-x})$$

$$\sin n\pi = 0 \quad \cos n\pi = (-1)^n$$

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

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

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

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

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

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

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

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

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

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

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

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

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



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

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

## 10.4 Derivatives/Integrals

Integration by parts:

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

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

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

$$\int \tan ax = -\frac{\ln |\cos ax|}{a} \quad \int x \sin ax = \frac{\sin ax - ax \cos ax}{a^2}$$

$$\int e^{-x^2} = \frac{\sqrt{\pi}}{2} \text{erf}(x) \quad \int x e^{ax} = \frac{e^{ax}}{a^2} (ax - 1)$$

$$\int \sin^2(x) = \frac{x}{2} - \frac{1}{4} \sin 2x \quad \int \sin^3 x = \frac{1}{12} \cos 3x - \frac{3}{4} \cos x$$

$$\int \cos^2(x) = \frac{x}{2} + \frac{1}{4} \sin 2x \quad \int \cos^3 x = \frac{1}{12} \sin 3x + \frac{3}{4} \sin x$$

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

$$\int x e^x = e^x (x - 1) \quad \int x^2 e^x = e^x (x^2 - 2x + 2)$$

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

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

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

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

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

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

## 10.5 Sums

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

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

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

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

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

## 10.6 Series

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

$$\ln(1+x) = x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \dots, (-1 < x \leq 1)$$

$$\sqrt{1+x} = 1 + \frac{x}{2} - \frac{x^2}{8} + \frac{2x^3}{32} - \frac{5x^4}{128} + \dots, (-1 \leq x \leq 1)$$

$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \dots, (-\infty < x < \infty)$$

$$\cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \dots, (-\infty < x < \infty)$$