

Contents

1	Basic	
1.1	Pragma	
2	Data Structure	
2.1	Black Magic	
2.2	Lichao Tree	
2.3	Linear Basis	
3	Graph	
3.1	Bridge CC	
3.2	Dinic	
3.3	Min Cost Max Flow	
3.4	Stoer Wagner Algorithm	
3.5	Hopcroft Karp Algorithm	
3.6	General Matching	
4	Geometry	
4.1	Basic	
4.2	2D Convex Hull	
5	String	
5.1	KMP	
5.2	Suffix Array	
5.3	Booth Algorithm	
6	Math	
6.1	Extgcd	
6.2	Linear Sieve	
6.3	Miller Rabin	
6.4	Pollard's Rho	
6.5	Fast Fourier Transform	
6.6	3 Primes NTT	
6.7	Number Theory Transform	

1 Basic

1.1 Pragma

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

2 Data Structure

2.1 Black Magic

```
template<typename T>
using pbds_tree = tree<T, null_type, less<T>,
    rb_tree_tag, tree_order_statistics_node_update>;
// find_by_order: like array accessing, order_of_key
```

2.2 Lichao Tree

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

2.3 Linear Basis

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

3 Graph

3.1 Bridge CC

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

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

3.2 Dinic

```
template<typename T> // maxn: edge/node counts
struct dinic { // T: int or LL, up to range of flow
    const T IN_INF = (is_same_v<T, int>) ? INF : LINF;
```

```

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

```

3.3 Min Cost Max Flow

```

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

```

```

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

```

3.4 Stoer Wagner Algorithm

```

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

```

```

    return ret;
}
} sw;

```

3.5 Hopcroft Karp Algorithm

```

// Find maximum bipartite matching in  $O(E\sqrt{V})$ 
// g: edges for all nodes at left side
vector<int> hopcroft_karp(vector<vector<int>> g, int l,
    int r) {
    vector<int> match_l(l, -1), match_r(r, -1);
    vector<int> dis(l);
    vector<bool> vis(l);
    while(true) {
        queue<int> que;
        for(int i = 0; i < l; i++) {
            if(match_l[i] == -1)
                dis[i] = 0, que.push(i);
            else
                dis[i] = -1;
            vis[i] = false;
        }
        while(!que.empty()) {
            int x = que.front();
            que.pop();
            for(int y : g[x])
                if(match_r[y] != -1 && dis[match_r[y]] == -1) {
                    dis[match_r[y]] = dis[x] + 1;
                    que.push(match_r[y]);
                }
        }
        auto dfs = [&](auto dfs, int x) {
            vis[x] = true;
            for(int y : g[x]) {
                if(match_r[y] == -1) {
                    match_l[x] = y;
                    match_r[y] = x;
                    return true;
                }
                else if(dis[match_r[y]] == dis[x] + 1
                    && !vis[match_r[y]]
                    && dfs(dfs, match_r[y])) {
                    match_l[x] = y;
                    match_r[y] = x;
                    return true;
                }
            }
            return false;
        };
        bool ok = true;
        for(int i = 0; i < l; i++)
            if(match_l[i] == -1 && !dfs(dfs, i))
                ok = false;
        if(ok)
            break;
    }
    return match_l;
} // 0-based

```

3.6 General Matching

```

// Find max matching on general graph in  $O(|V|^3)$ 
vector<int> max_matching(vector<vector<int>> g) {
    int n = g.size();
    vector<int> match(n + 1, n), pre(n + 1, n), que;
    vector<int> s(n + 1), mark(n + 1), pa(n + 1);
    function<int(int)> fnd = [&](int x) {
        if(x == pa[x]) return x;
        return pa[x] = fnd(pa[x]);
    };
    auto lca = [&](int x, int y) {
        static int tk = 0;
        tk++;
        x = fnd(x);
        y = fnd(y);
        for(;; swap(x, y))
            if(x != n) {
                if(mark[x] == tk)
                    return x;
                mark[x] = tk;
                x = fnd(pre[match[x]]);
            }
    };
    auto blossom = [&](int x, int y, int l) {

```

```

        while(fnd(x) != l) {
            pre[x] = y;
            y = match[x];
            if(s[y] == 1)
                que.push_back(y), s[y] = 0;
            if(pa[x] == x) pa[x] = l;
            if(pa[y] == y) pa[y] = l;
            x = pre[y];
        }
    };
    auto bfs = [&](int r) {
        fill(s.begin(), s.end(), -1);
        iota(pa.begin(), pa.end(), 0);
        que = {r}; s[r] = 0;
        for(int it = 0; it < que.size(); it++) {
            int x = que[it];
            for(int u : g[x]) {
                if(s[u] == -1) {
                    pre[u] = x;
                    s[u] = 1;
                    if(match[u] == n) {
                        for(int a = u, b = x, lst;
                            b != n; a = lst, b = pre[a]) {
                            lst = match[b];
                            match[b] = a;
                            match[a] = b;
                        }
                        return;
                    }
                }
                que.push_back(match[u]);
                s[match[u]] = 0;
            }
        }
        else if(s[u] == 0 && fnd(u) != fnd(x)) {
            int l = lca(u, x);
            blossom(x, u, l);
            blossom(u, x, l);
        }
    }
};
for(int i = 0; i < n; i++)
    if(match[i] == n) bfs(i);
match.resize(n);
for(int i = 0; i < n; i++)
    if(match[i] == n) match[i] = -1;
return match;
} // 0-based

```

4 Geometry

4.1 Basic

```

using pt = pair<ll, ll>;
using ptf = pair<ld, ld>;
pt operator+(pt a, pt b)
{ return pt {a.F + b.F, a.S + b.S}; }
pt operator-(pt a, pt b)
{ return pt {a.F - b.F, a.S - b.S}; }
ptf to_ptf(pt p) { return ptf {p.F, p.S}; }
int sign(ll x) { return (x > 0) - (x < 0); }
ll dot(pt a, pt b) { return a.F * b.F + a.S * b.S; }
ll cross(pt a, pt b) { return a.F * b.S - a.S * b.F; }
ld abs2(ptf a) { return dot(a, a); }
ld abs(ptf a) { return sqrt(dot(a, a)); }
int ori(pt a, pt b, pt c)
{ return sign(cross(b - a, c - a)); }
bool operator<(pt a, pt b)
{ return a.F != b.F ? a.F < b.F : a.S < b.S; }

```

4.2 2D Convex Hull

```

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

```

```

    }
    return h.resize(t), h;
} // not tested, but trust ckiseki!

```

5 String

5.1 KMP

```

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

```

5.2 Suffix Array

```

int sa[maxn], tmp[2][maxn], c[256];
void get_sa(const string &s) {
    int *x = tmp[0], *y = tmp[1], m = 256, n = s.size();
    for(int i = 0; i < m; i++) c[i] = 0;
    for(int i = 0; i < n; i++) c[x[i]]++;
    for(int i = 1; i < m; i++) c[i] += c[i - 1];
    for(int i = n - 1; i >= 0; --i) sa[--c[x[i]]] = i;
    for(int k = 1; k < n; k <= 1) {
        for(int i = 0; i < m; i++) c[i] = 0;
        for(int i = 0; i < n; i++) c[x[i]]++;
        for(int i = 1; i < m; i++) c[i] += c[i - 1];
        int p = 0;
        for(int i = n - k; i < n; i++) y[p++] = i;
        for(int i = 0; i < n; i++)
            if(sa[i] >= k) y[p++] = sa[i] - k;
        for(int i = n - 1; i >= 0; --i) sa[--c[x[y[i]]]] = y[i];
        y[sa[0]] = p = 0;
        for(int i = 1; i < n; i++) {
            int a = sa[i], b = sa[i - 1];
            if(x[a] == x[b] && a + k < n && b + k < n && x[a + k] == x[b + k])
                else p++;
            y[sa[i]] = p;
        }
        if(n == p + 1)
            break;
        swap(x, y);
        m = p + 1;
    }
} // sa[i]: index which ranks i
int rk[maxn], lcp[maxn];
void get_cp(const string &s) {
    int n = s.size(), val = 0;
    for(int i = 0; i < n; i++) rk[sa[i]] = i;
    for(int i = 0; i < n; i++) {
        if(rk[i] == 0) lcp[rk[i]] = 0;
        else {
            if(val) val--;
            int p = sa[rk[i] - 1];
            while(val + i < n && val + p < n && s[val + i] == s[val + p])
                val++;
            lcp[rk[i]] = val;
        }
    }
} // get_sa and get_lcp are not tested

```

5.3 Booth Algorithm

```

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

```

```

if(i == -1 && s[j] != s[k + i + 1]) {
    if(s[j] < s[k + i + 1])
        k = j;
    f[j - k] = -1;
}
else
    f[j - k] = i + 1;
}
return k;
}

```

6 Math

6.1 Extgcd

```

// return (d, x, y) s.t. ax+by=d=gcd(a,b)
template<typename T>
tuple<T, T, T> extgcd(T a, T b) {
    if(!b) return make_tuple(a, 1, 0);
    auto [d, x, y] = extgcd(b, a % b);
    return make_tuple(d, y, x - (a / b) * y);
} // not tested

```

6.2 Linear Sieve

```

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

```

6.3 Miller Rabin

```

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

```

6.4 Pollard's Rho

```

ull f(ull x, ull k, ull m) {
    return (__uint128_t(x) * x + k) % m;
}
// does not work when n is prime
// return any non-trivial factor
ull pollard_rho(ull n) {
    if(!(n & 1)) return 2;
    mt19937 rnd(120821011);
    while(true) {
        ull y = 2, yy = y, x = rnd() % n, t = 1;
        for(ull sz = 2; t == 1; sz <= 1, y = yy) {
            for(ull i = 0; t == 1 && i < sz; ++i) {
                yy = f(yy, x, n);
                t = __gcd(yy > y ? yy - y : y - yy, n);
            }
            if(t != 1 && t != n) return t;
        }
    }
}

```

6.5 Fast Fourier Transform

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

6.6 3 Primes NTT

```
// MOD: arbitrary prime
const int M1 = 998244353;
const int M2 = 1004535809;
const int M3 = 2013265921;
int super_big_crt(int64_t A, int64_t B, int64_t C) {
    static_assert(M1 <= M2 && M2 <= M3);
    ll r12 = mpow(M1, M2 - 2, M2);
    ll r13 = mpow(M1, M3 - 2, M3);
    ll r23 = mpow(M2, M3 - 2, M3);
    ll M1M2 = 1LL * M1 * M2 % MOD;
    B = (B - A + M2) * r12 % M2;
    C = (C - A + M3) * r13 % M3;
    C = (C - B + M3) * r23 % M3;
    return (A + B * M1 + C * M1M2) % MOD;
} // return ans % MOD
```

6.7 Number Theory Transform

```
/* mod | g | maxn possible values:
998244353 | 3 | 8388608
1004535809 | 3 | 2097152
```

```
2013265921 | 31 | 134217728 */
template <int mod, int G, int maxn>
struct NTT {
    ll mpow(ll a, ll b) {
        ll res = 1;
        for(; b >>= 1; a = a * a % mod)
            if(b & 1)
                res = res * a % mod;
        return res;
    }
    static_assert(maxn == (maxn & -maxn));
    int roots[maxn];
    NTT() {
        ll r = mpow(G, (mod - 1) / maxn);
        for(int i = maxn >> 1; i; i >>= 1) {
            roots[i] = 1;
            for(int j = 1; j < i; j++)
                roots[i + j] = roots[i + j - 1] * r % mod;
            r = r * r % mod;
        }
    }
    // n must be 2^k, and 0 <= f[i] < mod
    void operator()(vector<ll> &f, int n, bool inv = false) {
        for(int i = 0, j = 0; i < n; i++) {
            if(i < j) swap(f[i], f[j]);
            for(int k = n >> 1; (j ^= k) < k; k >>= 1);
        }
        for(int s = 1; s < n; s *= 2) {
            for(int i = 0; i < n; i += s * 2) {
                for(int j = 0; j < s; j++) {
                    ll a = f[i + j];
                    ll b = f[i + j + s] * roots[s + j] % mod;
                    f[i + j] = (a + b) % mod;
                    f[i + j + s] = (a - b + mod) % mod;
                }
            }
        }
        if(inv) {
            int invn = mpow(n, mod - 2);
            for(int i = 0; i < n; i++)
                f[i] = f[i] * invn % mod;
            reverse(f.begin() + 1, f.end());
        }
    }
};
```