

Contents

1 Basic	1	6 Math	13
1.1 Template	1	6.1 Berlekamp Massey	13
1.2 Fast IO	1	6.2 Characteristic Polynomial	13
1.3 vimrc	1	6.3 Discrete Logarithm	13
2 Graph	1	6.4 Extgcd	13
2.1 2SAT (SCC)	1	6.5 Floor Sum	13
2.2 VertexBCC	2	6.6 Factorial Mod P^k	13
2.3 EdgeBCC	2	6.7 Gaussian Elimination	14
2.4 Centroid Decomposition	2	6.8 Linear Function Mod Min	14
2.5 Count Cycles	3	6.9 MillerRabin PollardRho	14
2.6 DirectedMST	3	6.10 Quadratic Residue	14
2.7 Dominator Tree	3	6.11 Simplex	15
2.8 Heavy Light Decomposition	3	6.12 Linear Programming	15
2.9 Virtual Tree	4	Construction	15
2.10 Vizing	4	6.13 Estimation	15
2.11 Maximum Clique Dynamic	4	6.14 Theorem	15
2.12 Theory	5	6.15 General Purpose Numbers	16
3 Data Structure	5	6.16 Integral	16
3.1 LiChao Tree	5	7 Polynomial	16
3.2 Dynamic Line Hull	5	7.1 FFT	16
3.3 Leftist Tree	5	7.2 NTT	16
3.4 Link Cut Tree	5	7.3 FWT	17
3.5 Splay Tree	6	7.4 Polynomial	17
3.6 Treap	6	7.5 Generating Functions	19
4 Flow/Matching	7	8 Geometry	19
4.1 Hopcroft Karp	7	8.1 Basic	19
4.2 Dinic	7	8.2 Convex Hull	19
4.3 Min Cost Max Flow	7	8.3 Dynamic Convex Hull	19
4.4 Min Cost Circulation	8	8.4 Point In Convex Hull	20
4.5 Kuhn Munkres	8	8.5 Point In Circle	20
4.6 Stoer Wagner (Min-cut)	9	8.6 Half Plane Intersection	20
4.7 GomoryHu Tree	9	8.7 Minkowski Sum	20
4.8 General Graph Matching	9	8.8 Polar Angle	20
4.9 Flow notes	10	8.9 Rotating Sweep Line	20
5 String	10	8.10 Segment Intersect	20
5.1 AC Automaton	10	8.11 Circle Intersect With	21
5.2 Lyndon Factorization	10	Any	21
5.3 KMP	10	8.12 Tangents	21
5.4 Manacher	11	8.13 Tangent to Convex Hull	21
5.5 Minimum Rotate	11	8.14 Minimum Enclosing Circle	21
5.6 Palindrome Tree	11	8.15 Union of Stuff	21
5.7 Palindrome Partition	11	8.16 Delaunay Triangulation	22
5.8 Repetition	11	8.17 Voronoi Diagram	22
5.9 Suffix Array	11	8.18 3D Basic	22
5.10 SAIS (C++20)	12	8.19 3D Convex Hull	23
5.11 Suffix Automaton	12	9 Misc	23
5.12 Z Value	12	9.1 Binary Search On Fraction	23
		9.2 Cyclic Ternary Search	23
		9.3 Matroid Intersection	23
		9.4 Min Plus Convolution	24
		9.5 Mo's Algorithm	24
		9.6 Mo's Algorithm On Tree	24
		9.7 PBDS	24
		9.8 Simulated Annealing	24
		9.9 SOS dp	24
		9.10 SMAWK	24
		9.11 Tree Hash	25
		9.12 Python	25

1 Basic

1.1 Template [30a783]

```
template<typename T> void _do(T x){cerr<<x<<"\n";}
template<typename T,typename ...U> void _do(T x,U ...y)
{cerr<<x<<" ";_do(y...);}
#define dbg(...) cerr<<#__VA_ARGS__<<" = ";do(
    __VA_ARGS__);
#define uni(c) c.resize(distance(c.begin(),unique(c.
    begin(),c.end())))
#define unisort(c) sort(c.begin(),c.end()),uni(c)
auto SEED = chrono::steady_clock::now().
    time_since_epoch().count();
mt19937 rng(SEED);
// cpp $1 -dD -P -fpreprocessed | tr -d '[:space:]' |
    md5sum | cut -c-6
```

1.2 Fast IO [b7f4fb]

```
#pragma GCC optimize("Ofast,inline,unroll-loops")
#pragma GCC target("bmi,bmi2,lzcnt,popcnt,avx2")
#pragma GCC target("sse,sse2,sse3,sse4,ssse3,sse4,popcnt,abm
    ,mmx,avx,tune=native")
```

```
#include<unistd.h>
char OB[65536]; int OP;
inline char RC() {
    static char buf[65536], *p = buf, *q = buf;
    return p == q && (q = (p = buf) + read(0, buf, 65536)
        ) == buf ? -1 : *p++;
}
inline int R() {
    static char c;
    while((c = RC()) < '0'); int a = c ^ '0';
    while((c = RC()) >= '0') a *= 10, a += c ^ '0';
    return a;
}
inline void W(int n) {
    static char buf[12], p;
    if (n == 0) OB[OP++] = '0'; p = 0;
    while (n) buf[p++] = '0' + (n % 10), n /= 10;
    for (--p; p >= 0; --p) OB[OP++] = buf[p];
    if (OP > 65520) write(1, OB, OP), OP = 0;
}
```

1.3 vimrc [994cc6]

```
sy on
set ru nu rnu cul cin et bs=2 ls=2 so=8 sw=4 sts=4
mouse=a
inoremap {<CR> {<CR><Esc>O
noremap <F9> <Esc>:w<CR>:!g++ "%:p" -o "%:p:r".out -std
    =c++14 -O2 -Wall -Wextra -Wshadow -Wconversion -
    fsanitize=address,undefined<CR>
noremap <F10> <Esc>:!"%:p:r".out<CR>
map <F11> <F9><F10>
```

2 Graph

2.1 2SAT (SCC) [1aebd0]

```
struct TwoSAT {
    // 0-indexed
    // idx i * 2 -> +i, i * 2 + 1 -> -i
    vector<vector<int>> adj, radj;
    vector<int> dfs_ord, idx, solution;
    vector<bool> vis;
    int n, nsc;
    TwoSAT() = default;
    TwoSAT(int _n) : n(_n), nsc(0) {
        adj.resize(n * 2), radj.resize(n * 2);
    }
    void add_clause(int x, int y) {
        // (x or y) = true
        int nx = x ^ 1, ny = y ^ 1;
        adj[nx].push_back(y), radj[y].push_back(nx);
        adj[ny].push_back(x), radj[x].push_back(ny);
    }
    void add_ifthen(int x, int y) {
        // if x = true then y = true
        add_clause(x ^ 1, y);
    }
    void add_must(int x) {
        // x = true
        int nx = x ^ 1;
        adj[nx].pb(x), radj[x].pb(nx);
    }
    void dfs(int v) {
        vis[v] = true;
        for (int u : adj[v]) if (!vis[u])
            dfs(u);
        dfs_ord.push_back(v);
    }
    void rdfs(int v) {
        idx[v] = nsc;
        for (int u : radj[v]) if (idx[u] == -1)
            rdfs(u);
    }
    bool find_sol() {
        vis.assign(n * 2, false), idx.assign(n * 2, -1),
            solution.assign(n, -1);
        for (int i = 0; i < n * 2; ++i) if (!vis[i])
            dfs(i);
        reverse(dfs_ord.begin(), dfs_ord.end());
        for (int i : dfs_ord) if (idx[i] == -1)
```

```

    rdfs(i), nsc++);
    for (int i = 0; i < n; i++) {
        if (idx[i < 1] == idx[i < 1 | 1])
            return false;
        if (idx[i < 1] < idx[i < 1 | 1])
            solution[i] = 0;
        else
            solution[i] = 1;
    }
    return true;
}
};

```

2.2 VertexBCC [d04ebe]

```

struct BCC{ // 0-based, allow multi edges but not allow
    loops
    int n, m, cnt = 0;
    // n:|V|, m:|E|, cnt:#bcc
    // bcc i : vertices bcc_v[i] and edges bcc_e[i]
    vector<vector<int>> bcc_v, bcc_e;
    vector<vector<pii>> g; // original graph
    vector<pii> edges; // 0-based
    BCC(int _n, vector<pii> _edges):
        n(_n), m(SZ(_edges)), g(_n), edges(_edges){
        for(int i = 0; i < m; i++){
            auto [u, v] = edges[i];
            g[u].pb(pii(v, i)); g[v].pb(pii(u, i));
        }
    }
    void make_bcc(){ bcc_v.pb(); bcc_e.pb(); cnt++; }
    // modify these if you need more information
    void add_v(int v){ bcc_v.back().pb(v); }
    void add_e(int e){ bcc_e.back().pb(e); }
    void build(){
        vector<int> in(n, -1), low(n, -1), stk;
        vector<vector<int>> up(n);
        int ts = 0;
        auto _dfs = [&](auto dfs, int now, int par, int pe)
            -> void{
            if(pe != -1) up[now].pb(pe);
            in[now] = low[now] = ts++;
            stk.pb(now);
            for(auto [v, e] : g[now]){
                if(e == pe) continue;
                if(in[v] != -1){
                    if(in[v] < in[now]) up[now].pb(e);
                    low[now] = min(low[now], in[v]);
                    continue;
                }
                dfs(dfs, v, now, e);
                low[now] = min(low[now], low[v]);
            }
            if((now != par && low[now] >= in[par]) || (now ==
                par && SZ(g[now]) == 0)){
                make_bcc();
                for(int v = stk.back(); v = stk.back()){
                    stk.pop_back(), add_v(v);
                    for(int e : up[v]) add_e(e);
                    if(v == now) break;
                }
                if(now != par) add_v(par);
            }
        };
        for(int i = 0; i < n; i++){
            if(in[i] == -1) _dfs(_dfs, i, i, -1);
        }
    }
};

```

2.3 EdgeBCC [8a9523]

```

vector<int> adj[N];
struct EdgeBCC {
    // 0-indexed
    vector<int> newadj[N];
    vector<int> low, dep, idx, stk, par;
    vector<bool> bridge; // edge i -> pa[i] is bridge ?
    int n, nbcc;
    EdgeBCC () = default;
    EdgeBCC (int _n) : n(_n), nbcc(0) {
        low.assign(n, -1), dep.assign(n, -1), idx.assign(n,
            -1);
    }
};

```

```

    par.assign(n, -1), bridge.assign(n, false);
    for (int i = 0; i < n; ++i) if (dep[i] == -1) {
        dfs(i, -1);
    }
    for (int i = 1; i < n; ++i) if (bridge[i]) {
        newadj[idx[i]].pb(idx[par[i]]);
        newadj[idx[par[i]]].pb(idx[i]);
    }
}
void dfs(int v, int pa) {
    low[v] = dep[v] = ~pa ? dep[pa] + 1 : 0;
    par[v] = pa;
    stk.push_back(v);
    bool visp = false;
    for (int u : adj[v]) {
        if (!visp && u == pa) {
            visp = true;
        } else if (dep[u] == -1) {
            dfs(u, v);
            low[v] = min(low[v], low[u]);
        } else {
            low[v] = min(low[v], low[u]);
        }
    }
    if (low[v] == dep[v]) {
        if (~pa) bridge[v] = true;
        int x;
        do {
            x = stk.back(), stk.pop_back();
            idx[x] = nbcc;
        } while (x != v);
        nbcc++;
    }
}
};

```

2.4 Centroid Decomposition [2356be]

```

vector<int> adj[N];
struct CentroidDecomposition {
    // 0-index
    vector<int> sz, cd_pa;
    int n;
    CentroidDecomposition () = default;
    CentroidDecomposition (int _n) : n(_n) {
        sz.assign(n, 0), cd_pa.assign(n, -2);
        dfs_cd(0, -1);
    }
    void dfs_sz(int v, int pa) {
        sz[v] = 1;
        for (int u : adj[v]) if (u != pa && cd_pa[u] == -2)
            dfs_sz(u, v), sz[v] += sz[u];
    }
    int dfs_cen(int v, int pa, int s) {
        for (int u : adj[v]) if (u != pa && cd_pa[u] == -2)
            if (sz[u] * 2 > s)
                return dfs_cen(u, v, s);
        return v;
    }
    vector<int> block;
    void dfs_cd(int v, int pa) {
        dfs_sz(v, pa);
        int c = dfs_cen(v, pa, sz[v]);
        cd_pa[c] = pa;
        // centroid D&C
        for (int u : adj[c]) if (cd_pa[u] == -2) {
            dfs_ans(u, c);
            // do something
        }
        for (int u : adj[c]) if (cd_pa[u] == -2) {
            dfs_cd(u, c);
        }
    }
    void dfs_ans(int v, int pa) {
        // calculate path through centroid
        // do something
        // remember delete path from the same size
        for (int u : adj[v]) if (u != pa && cd_pa[u] == -2)
            dfs_ans(u, v);
    }
};

```

```
// Centroid Tree Property:
// let k = lca(u, v) in Centroid Tree, then dis(u, v)
// = dis(u, k) + dis(k, v)
};
```

2.5 Count Cycles [c7e8f2]

```
// ord = sort by deg decreasing, rk[ord[i]] = i
// D[i] = edge point from rk small to rk big
for (int x : ord) { // c3
    for (int y : D[x]) vis[y] = 1;
    for (int y : D[x]) for (int z : D[y]) c3 += vis[z];
    for (int y : D[x]) vis[y] = 0;
}
for (int x : ord) { // c4
    for (int y : D[x]) for (int z : adj[y])
        if (rk[z] > rk[x]) c4 += vis[z]++;
    for (int y : D[x]) for (int z : adj[y])
        if (rk[z] > rk[x]) --vis[z];
} // both are O(M*sqrt(M)), test @ 2022 CCPC guangzhou
```

2.6 DirectedMST [d3eb5f]

```
using D = int;
struct edge {
    int u, v; D w;
};
// 0-based, return index of edges
vector<int> dmst(vector<edge> &e, int n, int root) {
    using T = pair<D, int>;
    using PQ = pair<priority_queue<T, vector<T>,
        greater<T>>, D>;
    auto push = [](PQ &pq, T v) {
        pq.first.emplace(v.first - pq.second, v.second);
    };
    auto top = [](const PQ &pq) -> T {
        auto r = pq.first.top();
        return {r.first + pq.second, r.second};
    };
    auto join = [&push, &top](PQ &a, PQ &b) {
        if (a.first.size() < b.first.size()) swap(a, b);
        while (!b.first.empty())
            push(a, top(b)), b.first.pop();
    };
    vector<PQ> h(n * 2);
    for (int i = 0; i < e.size(); ++i)
        push(h[e[i].v], {e[i].w, i});
    vector<int> a(n * 2), v(n * 2, -1), pa(n * 2, -1), r(
        n * 2);
    iota(a.begin(), a.end(), 0);
    auto o = [&](int x) { int y;
        for (y = x; a[y] != y; y = a[y]);
        for (int ox = x; x != y; ox = x)
            x = a[x], a[ox] = y;
        return y;
    };
    v[root] = n + 1;
    int pc = n;
    for (int i = 0; i < n; ++i) if (v[i] == -1) {
        for (int p = i; v[p] == -1 || v[p] == i; p = o(e[r[
            p]].u)) {
            if (v[p] == i) {
                int q = p; p = pc++;
                do {
                    h[q].second = -h[q].first.top().first;
                    join(h[pa[q]] = a[q] = p, h[q]);
                } while ((q = o(e[r[q]].u)) != p);
            }
            v[p] = i;
            while (!h[p].first.empty() && o(e[top(h[p]).
                second].u) == p)
                h[p].first.pop();
            r[p] = top(h[p]).second;
        }
    }
    vector<int> ans;
    for (int i = pc - 1; i >= 0; i--)
        if (i != root && v[i] != n) {
            for (int f = e[r[i]].v; f != -1 && v[f] != n; f =
                pa[f]) v[f] = n;
            ans.pb(r[i]);
        }
}
```

```
return ans;
}
```

2.7 Dominator Tree [6378d5]

```
struct Dominator_tree {
    int n, id;
    vector<vector<int>> adj, radj, bucket;
    vector<int> sdom, dom, vis, rev, par, rt, mn;
    Dominator_tree(int _n) : n(_n), id(0) {
        adj.resize(n), radj.resize(n), bucket.resize(n);
        sdom.resize(n), dom.resize(n, -1), vis.resize(n,
            -1);
        rev.resize(n), rt.resize(n), mn.resize(n), par.
            resize(n);
    }
    void add_edge(int u, int v) {adj[u].pb(v);}
    int query(int v, bool x) {
        if (rt[v] == v) return x ? -1 : v;
        int p = query(rt[v], true);
        if (p == -1) return x ? rt[v] : mn[v];
        if (sdom[mn[v]] > sdom[mn[rt[v]]]) mn[v] = mn[rt[v]
            ];
        rt[v] = p;
        return x ? p : mn[v];
    }
    void dfs(int v) {
        vis[v] = id, rev[id] = v;
        rt[id] = mn[id] = sdom[id] = id, id++;
        for (int u : adj[v]) {
            if (vis[u] == -1) dfs(u, par[vis[u]] = vis[v];
            radj[vis[u]].pb(vis[v]);
        }
    }
    void build(int s) {
        dfs(s);
        for (int i = id - 1; ~i; --i) {
            for (int u : radj[i]) {
                sdom[i] = min(sdom[i], sdom[query(u, false)]);
            }
            if (i) bucket[sdom[i]].pb(i);
            for (int u : bucket[i]) {
                int p = query(u, false);
                dom[u] = sdom[p] == i ? i : p;
            }
            if (i) rt[i] = par[i];
        }
        vector<int> res(n, -1);
        for (int i = 1; i < id; ++i) {
            if (dom[i] != sdom[i]) dom[i] = dom[dom[i]];
        }
        for (int i = 1; i < id; ++i) res[rev[i]] = rev[dom[
            i]];
        res[s] = s;
        dom = res;
    } // dom[]: parent on dom. tree, -1 if not reachable
};
```

2.8 Heavy Light Decomposition [372958]

```
vector<int> adj[N];
struct HLD {
    // 0-index
    vector<int> dep, pt, hd, idx, sz, par, vis;
    int n, _t;
    HLD() = default;
    HLD(int _n) : n(_n) {
        pt.assign(n, -1), hd.assign(n, -1), par.assign(n, -1);
        idx.assign(n, 0), sz.assign(n, 0), dep.assign(n, 0),
            vis.assign(n, 0);
        _t = 0;
        for (int i = 0; i < n; ++i) if (!vis[i]) {
            dfs1(i, -1);
            dfs2(i, -1, 0);
        }
    }
    void dfs1(int v, int pa) {
        par[v] = pa;
        dep[v] = ~pa ? dep[pa] + 1 : 0;
        sz[v] = vis[v] = 1;
        for (int u : adj[v]) if (u != pa) {
            dfs1(u, v);
        }
    }
}
```

```

    if (pt[v] == -1 || sz[pt[v]] < sz[u])
        pt[v] = u;
    sz[v] += sz[u];
}
}
void dfs2(int v, int pa, int h) {
    if (v == -1)
        return;
    idx[v] = _t++, hd[v] = h;
    dfs2(pt[v], v, h);
    for (int u : adj[v]) if (u != pa && u != pt[v]) {
        dfs2(u, v, u);
    }
}
void modify(int u, int v) {
    while (hd[u] != hd[v]) {
        if (dep[hd[u]] < dep[hd[v]])
            swap(u, v);
        // range [idx[hd[u]], idx[u] + 1)
        u = par[hd[u]];
    }
    if (dep[u] < dep[v])
        swap(u, v);
    // range [idx[v], idx[u] + 1)
}
int query(int u, int v) {
    int ans = 0;
    while (hd[u] != hd[v]) {
        if (dep[hd[u]] < dep[hd[v]])
            swap(u, v);
        // range [idx[hd[u]], idx[u] + 1)
        u = par[hd[u]];
    }
    if (dep[u] < dep[v])
        swap(u, v);
    // range [idx[v], idx[u] + 1)
    return ans;
}
};

```

2.9 Virtual Tree [dcbe4f]

```

// need lca
vector<int> _g[N], stk;
int st[N], ed[N];
void solve(vector<int> v) {
    auto cmp = [&](int x, int y) {return st[x] < st[y];};
    sort(all(v), cmp);
    int sz = v.size();
    for (int i = 0; i < sz - 1; ++i)
        v.pb(lca(v[i], v[i + 1]));
    sort(all(v), cmp);
    v.resize(unique(all(v)) - v.begin());
    stk.clear(), stk.pb(v[0]);
    for (int i = 1; i < v.size(); ++i) {
        int x = v[i];
        while (ed[stk.back()] < ed[x]) stk.pop_back();
        _g[stk.back()].pb(x), stk.pb(x);
    }
    // do something
    for (int i : v) _g[i].clear();
}

```

2.10 Vizing [fa4b32]

```

namespace 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;
void init(int _n) { n = _n;
    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 < E.size(); ++t) {
    int u = E[t].F, v0 = E[t].S, 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 = (int)L.size() - 1; a
            >= 0; --a) c = color(u, L[a].F, c);
        else if (!C[u][d]) for (int a = (int)L.size() -
            1; a >= 0; --a) color(u, L[a].F, L[a].S);
        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 = (int)L.size() - 2; a >= 0 && L[a].S !=
                c; --a);
            for (; a >= 0; --a) color(u, L[a].F, L[a].S);
        }
        else --t;
    }
}
} // namespace vizing

```

2.11 Maximum Clique Dynamic [6dde09]

```

const int N = 150;
struct MaxClique { // Maximum Clique
    bitset<N> a[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++) a[i].reset();
    }
    void addEdge(int u, int v) { a[u][v] = a[v][u] = 1; }
    void csort(vector<int> &r, vector<int> &c) {
        int mx = 1, km = max(ans - q + 1, 1), t = 0,
            m = r.size();
        cs[1].reset(), cs[2].reset();
        for (int i = 0; i < m; i++) {
            int p = r[i], k = 1;
            while ((cs[k] & a[p]).count()) k++;
            if (k > mx) mx++, cs[mx + 1].reset();
            cs[k][p] = 1;
            if (k < km) r[t++] = p;
        }
        c.resize(m);
        if (t) c[t - 1] = 0;
        for (int k = km; k <= mx; k++)
            for (int p = cs[k]._Find_first(); p < N;
                p = cs[k]._Find_next(p))
                r[t] = p, c[t] = k, t++;
    }
    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, nc;
            bitset<N> nmask = mask & a[p];
            for (int i : r)
                if (a[p][i]) nr.push_back(i);
            if (!nr.empty()) {
                if (1 < 4) {

```

```

    for (int i : nr)
        d[i] = (a[i] & nmask).count();
    sort(nr.begin(), nr.end(),
        [&](int x, int y) { return d[x] > d[y]; });
    }
    csort(nr, nc), dfs(nr, nc, l + 1, nmask);
} else if (q > ans) ans = q, copy_n(cur, q, sol);
c.pop_back(), q--;
}
}
int solve(bitset<N> mask = bitset<N>(
    string(N, '1')) { // vertex mask
    vector<int> r, c;
    ans = q = 0;
    for (int i = 0; i < n; i++)
        if (mask[i]) r.push_back(i);
    for (int i = 0; i < n; i++)
        d[i] = (a[i] & mask).count();
    sort(r.begin(), r.end(),
        [&](int i, int j) { return d[i] > d[j]; });
    csort(r, c), dfs(r, c, 1, mask);
    return ans; // sol[0 ~ ans-1]
}
} graph;

```

2.12 Theory

$| \text{Maximum independent edge set} | = |V| - | \text{Minimum edge cover} |$
 $| \text{Maximum independent set} | = |V| - | \text{Minimum vertex cover} |$

3 Data Structure

3.1 LiChao Tree [90f481]

```

//C is range of x
//INF is big enough integer
struct Line {
    ll m, k;
    Line(ll _m=0, ll _k=0): m(_m), k(_k){}
    ll val(ll x){return m*x+k;}
};
struct LiChaoTree { //max y value
    Line st[C<<2];
    void init(int l, int r, int id) {
        st[id] = Line(0, 0);
        if (l == r) return;
        int mid = (l+r)/2;
        init(l, mid, id<<1);
        init(mid+1, r, id<<1|1);
    }
    void upd(int l, int r, Line seg, int id) {
        if (l == r) {
            if (seg.val(l) > st[id].val(l)) st[id] = seg;
            return;
        }
        int mid = (l+r)/2;
        if (st[id].m > seg.m) swap(st[id], seg);
        if (st[id].val(mid) < seg.val(mid)) {
            swap(st[id], seg);
            upd(l, mid, seg, id<<1);
        } else upd(mid+1, r, seg, id<<1|1);
    }
    ll qry(int l, int r, ll x, int id) {
        if (l == r) return st[id].val(x);
        int mid = (l+r)/2;
        if (x <= mid) return max(qry(l, mid, x, id<<1), st[id].val(x));
        else return max(qry(mid+1, r, x, id<<1|1), st[id].val(x));
    }
}

```

3.2 Dynamic Line Hull [8ec1c7]

```

struct Line {
    mutable ll k, m, p;
    bool operator<(const Line& o) const { return k < o.k; }
    }
    bool operator<(ll x) const { return p < x; }
};

```

```

struct LineContainer : multiset<Line, less<>> {
    static const ll inf = LLONG_MAX;
    ll div(ll a, ll b) {
        return a / b - ((a ^ b) < 0 && a % b);
    }
    bool isect(iterator x, iterator y) {
        if (y == end()) return x->p = inf, 0;
        if (x->k == y->k) x->p = x->m > y->m ? inf : -inf;
        else x->p = div(y->m - x->m, x->k - y->k);
        return x->p >= y->p;
    }
    void add(ll k, ll m) {
        auto z = insert({k, m, 0}), y = z++, x = y;
        while (isect(y, z)) z = erase(z);
        if (x != begin() && isect(--x, y)) isect(x, y = erase(y));
        while ((y = x) != begin() && (--x)->p >= y->p)
            isect(x, erase(y));
    }
    ll query(ll x) {
        assert(!empty());
        auto l = *lower_bound(x);
        return l.k * x + l.m;
    }
};

```

3.3 Leftist Tree [473c12]

```

struct node {
    ll rk, data, sz, sum;
    node *l, *r;
    node(ll k) : rk(0), data(k), sz(1), l(0), r(0), sum(k) {}
};
ll sz(node *p) { return p ? p->sz : 0; }
ll rk(node *p) { return p ? p->rk : -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 (rk(a->r) > rk(a->l)) swap(a->r, a->l);
    a->rk = rk(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;
}

```

3.4 Link Cut Tree [87ade4]

```

// weighted subtree size, weighted path max
struct LCT {
    int ch[N][2], pa[N], v[N], sz[N], sz2[N], w[N], mx[N], _id;
    // sz := sum of v in splay, sz2 := sum of v in virtual subtree
    // mx := max w in splay
    bool rev[N];
    LCT() : _id(1) {}
    int newnode(int _v, int _w) {
        int x = _id++;
        ch[x][0] = ch[x][1] = pa[x] = 0;
        v[x] = sz[x] = _v;
        sz2[x] = 0;
        w[x] = mx[x] = _w;
        rev[x] = false;
        return x;
    }
    void pull(int i) {
        sz[i] = v[i] + sz2[i];
        mx[i] = w[i];
        if (ch[i][0])
            sz[i] += sz[ch[i][0]], mx[i] = max(mx[i], mx[ch[i][0]]);
        if (ch[i][1])
            sz[i] += sz[ch[i][1]], mx[i] = max(mx[i], mx[ch[i][1]]);
    }
};

```

```

}
void push(int i) {
    if (rev[i]) reverse(ch[i][0]), reverse(ch[i][1]),
        rev[i] = false;
}
void reverse(int i) {
    if (!i) return;
    swap(ch[i][0], ch[i][1]);
    rev[i] ^= true;
}
bool isrt(int i) { // rt of splay
    if (!pa[i]) return true;
    return ch[pa[i]][0] != i && ch[pa[i]][1] != i;
}
void rotate(int i) {
    int p = pa[i], x = ch[p][1] == i, c = ch[i][!x], gp =
        pa[p];
    if (ch[gp][0] == p) ch[gp][0] = i;
    else if (ch[gp][1] == p) ch[gp][1] = i;
    pa[i] = gp, ch[i][!x] = p, pa[p] = i;
    ch[p][x] = c, pa[c] = p;
    pull(p), pull(i);
}
void splay(int i) {
    vector<int> anc;
    anc.push_back(i);
    while (!isrt(anc.back())) anc.push_back(pa[anc.back()]);
    while (!anc.empty()) push(anc.back()), anc.pop_back();
    while (!isrt(i)) {
        int p = pa[i];
        if (!isrt(p)) rotate(ch[p][1] == i ^ ch[pa[p]][1]
            == p ? i : p);
        rotate(i);
    }
}
void access(int i) {
    int last = 0;
    while (i) {
        splay(i);
        if (ch[i][1])
            sz2[i] += sz[ch[i][1]];
        sz2[i] -= sz[last];
        ch[i][1] = last;
        pull(i), last = i, i = pa[i];
    }
}
void makert(int i) {
    access(i), splay(i), reverse(i);
}
void link(int i, int j) {
    // assert(findrt(i) != findrt(j));
    makert(i);
    makert(j);
    pa[i] = j;
    sz2[j] += sz[i];
    pull(j);
}
void cut(int i, int j) {
    makert(i), access(j), splay(i);
    // assert(sz[i] == 2 && ch[i][1] == j);
    ch[i][1] = pa[j] = 0, pull(i);
}
int findrt(int i) {
    access(i), splay(i);
    while (ch[i][0]) push(i), i = ch[i][0];
    splay(i);
    return i;
}
};

```

3.5 Splay Tree [e9029a]

```

struct Splay {
    int pa[N], ch[N][2], sz[N], rt, _id;
    ll v[N];
    Splay() {}
    void init() {
        rt = 0, pa[0] = ch[0][0] = ch[0][1] = -1;
        sz[0] = 1, v[0] = inf;
    }
}

```

```

int newnode(int p, int x) {
    int id = _id++;
    v[id] = x, pa[id] = p;
    ch[id][0] = ch[id][1] = -1, sz[id] = 1;
    return id;
}
void rotate(int i) {
    int p = pa[i], x = ch[p][1] == i, gp = pa[p], c =
        ch[i][!x];
    sz[p] -= sz[i], sz[i] += sz[p];
    if (~c) sz[p] += sz[c], pa[c] = p;
    ch[p][x] = c, pa[p] = i;
    pa[i] = gp, ch[i][!x] = p;
    if (~gp) ch[gp][ch[gp][1] == p] = i;
}
void splay(int i) {
    while (~pa[i]) {
        int p = pa[i];
        if (~pa[p]) rotate(ch[pa[p]][1] == p ^ ch[p][1]
            == i ? i : p);
        rotate(i);
    }
    rt = i;
}
int lower_bound(int x) {
    int i = rt, last = -1;
    while (true) {
        if (v[i] == x) return splay(i), i;
        if (v[i] > x) {
            last = i;
            if (ch[i][0] == -1) break;
            i = ch[i][0];
        }
        else {
            if (ch[i][1] == -1) break;
            i = ch[i][1];
        }
    }
    splay(i);
    return last; // -1 if not found
}
void insert(int x) {
    int i = lower_bound(x);
    if (i == -1) {
        // assert(ch[rt][1] == -1);
        int id = newnode(rt, x);
        ch[rt][1] = id, ++sz[rt];
        splay(id);
    }
    else if (v[i] != x) {
        splay(i);
        int id = newnode(rt, x), c = ch[rt][0];
        ch[rt][0] = id;
        ch[id][0] = c;
        if (~c) pa[c] = id, sz[id] += sz[c];
        ++sz[rt];
        splay(id);
    }
}
};

```

3.6 Treap [9dc91b]

```

struct Treap {
    int pri, sz, val;
    Treap *tl, *tr;
    Treap(int x) : val(x), sz(1), pri(rand()), tl(NULL),
        tr(NULL) {}
    void pull() {
        sz = (tl ? tl->sz : 0) + 1 + (tr ? tr->sz : 0);
    }
    void out() {
        if (tl) tl->out();
        cout << val << ' ';
        if (tr) tr->out();
    }
};
void print(Treap *t) {
    t->out();
    cout << endl;
}
Treap* merge(Treap *a, Treap *b) {

```



```

if (!a || !b) return a ? a : b;
if (a->pri < b->pri) {
    a->tr = merge(a->tr, b);
    a->pull();
    return a;
} else {
    b->t1 = merge(a, b->t1);
    b->pull();
    return b;
}
}
void split(Treap* t, int k, Treap* &a, Treap* &b) {
    if (!t) a = b = NULL;
    else if ((t->t1 ? t->t1->sz : 0) + 1 <= k) {
        a = t;
        split(t->tr, k - (t->t1 ? t->t1->sz : 0) - 1, a->tr, b);
        a->pull();
    } else {
        b = t;
        split(t->t1, k, a, b->t1);
        b->pull();
    }
}
}

```

4 Flow/Matching

4.1 Hopcroft Karp [4b930f]

```

struct HopcroftKarp {
    const int INF = 1 << 30;
    vector<int> adj[N];
    int match[N], dis[N], v, n, m;
    bool matched[N], vis[N];
    bool dfs(int x) {
        vis[x] = true;
        for (int y : adj[x])
            if (match[y] == -1 || (dis[match[y]] == dis[x] + 1 && !vis[match[y]] && dfs(match[y]))) {
                match[y] = x, matched[x] = true;
                return true;
            }
        return false;
    }
    bool bfs() {
        memset(dis, -1, sizeof(int) * n);
        queue<int> q;
        for (int x = 0; x < n; ++x) if (!matched[x])
            dis[x] = 0, q.push(x);
        int mx = INF;
        while (!q.empty()) {
            int x = q.front(); q.pop();
            for (int y : adj[x]) {
                if (match[y] == -1) {
                    mx = dis[x];
                    break;
                } else if (dis[match[y]] == -1)
                    dis[match[y]] = dis[x] + 1, q.push(match[y]);
            }
        }
        return mx < INF;
    }
    int solve() {
        int res = 0;
        memset(match, -1, sizeof(int) * m);
        memset(matched, 0, sizeof(bool) * n);
        while (bfs()) {
            memset(vis, 0, sizeof(bool) * n);
            for (int x = 0; x < n; ++x) if (!matched[x])
                res += dfs(x);
        }
        return res;
    }
    void init(int _n, int _m) {
        n = _n, m = _m;
        for (int i = 0; i < n; ++i) adj[i].clear();
    }
    void add_edge(int x, int y) {
        adj[x].pb(y);
    }
}

```

4.2 Dinic [8898fb]

```

template <typename T>
struct Dinic { // 0-based
    const T INF = numeric_limits<T>::max() / 2;
    struct edge { int to, rev; T cap, flow; };
    int n, s, t;
    vector<vector<edge>> g;
    vector<int> dis, cur;
    T dfs(int u, T cap) {
        if (u == t || !cap) return cap;
        for (int &i = cur[u]; i < (int)g[u].size(); ++i) {
            edge &e = g[u][i];
            if (dis[e.to] == dis[u] + 1 && e.flow != e.cap) {
                T 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;
    }
    bool bfs() {
        fill(all(dis), -1);
        queue<int> q;
        q.push(s), dis[s] = 0;
        while (!q.empty()) {
            int v = q.front(); q.pop();
            for (auto &u : g[v])
                if (!dis[u.to] && u.flow != u.cap) {
                    q.push(u.to);
                    dis[u.to] = dis[v] + 1;
                }
        }
        return dis[t] != -1;
    }
    T solve(int _s, int _t) {
        s = _s, t = _t;
        T flow = 0, df;
        while (bfs()) {
            fill(all(cur), 0);
            while ((df = dfs(s, INF))) flow += df;
        }
        return flow;
    }
    void reset() {
        for (int i = 0; i < n; ++i)
            for (auto &j : g[i]) j.flow = 0;
    }
    void add_edge(int u, int v, T cap) {
        g[u].pb(edge{v, (int)g[v].size(), cap, 0});
        g[v].pb(edge{u, (int)g[u].size() - 1, 0, 0});
    }
    Dinic(int _n) : n(_n), g(n), dis(n), cur(n) {}
};

```

4.3 Min Cost Max Flow [e18ab8]

```

struct MCMF {
    const int INF = 1 << 30;
    struct edge {
        int v, f, c;
        edge(int _v, int _f, int _c) : v(_v), f(_f), c(_c) {}
    };
    vector<edge> E;
    vector<vector<int>> adj;
    vector<int> dis, pot, rt;
    int n, s, t;
    MCMF(int _n, int _s, int _t) : n(_n), s(_s), t(_t) {
        adj.resize(n);
    }
    void add_edge(int u, int v, int f, int c) {
        adj[u].pb(E.size(), E.pb(edge(v, f, c)));
        adj[v].pb(E.size(), E.pb(edge(u, 0, -c)));
    }
    bool SPFA() {
        rt.assign(n, -1), dis.assign(n, INF);
        vector<bool> vis(n, false);
    }
};

```

```

queue <int> q;
q.push(s), dis[s] = 0, vis[s] = true;
while (!q.empty()) {
    int v = q.front(); q.pop();
    vis[v] = false;
    for (int id : adj[v]) if (E[id].f > 0 && dis[E[id].v] > dis[v] + E[id].c + pot[v] - pot[E[id].v]) {
        dis[E[id].v] = dis[v] + E[id].c + pot[v] - pot[E[id].v];
        rt[E[id].v] = id;
        if (!vis[E[id].v]) vis[E[id].v] = true, q.push(E[id].v);
    }
}
return dis[t] != INF;
}
bool dijkstra() {
    rt.assign(n, -1), dis.assign(n, INF);
    priority_queue<pair<int, int>, vector<pair<int, int>>, greater<pair<int, int>>> pq;
    dis[s] = 0, pq.emplace(dis[s], s);
    while (!pq.empty()) {
        int d, v; tie(d, v) = pq.top(); pq.pop();
        if (dis[v] < d) continue;
        for (int id : adj[v]) if (E[id].f > 0 && dis[E[id].v] > dis[v] + E[id].c + pot[v] - pot[E[id].v]) {
            dis[E[id].v] = dis[v] + E[id].c + pot[v] - pot[E[id].v];
            rt[E[id].v] = id;
            pq.emplace(dis[E[id].v], E[id].v);
        }
    }
    return dis[t] != INF;
}
pair<int, int> runFlow() {
    pot.assign(n, 0);
    int cost = 0, flow = 0;
    bool fr = true;
    while ((fr ? SPFA() : dijkstra())) {
        for (int i = 0; i < n; i++) {
            dis[i] += pot[i] - pot[s];
        }
        int add = INF;
        for (int i = t; i != s; i = E[rt[i] ^ 1].v) {
            add = min(add, E[rt[i]].f);
        }
        for (int i = t; i != s; i = E[rt[i] ^ 1].v) {
            E[rt[i]].f -= add, E[rt[i] ^ 1].f += add;
        }
        flow += add, cost += add * dis[t];
        fr = false;
        swap(dis, pot);
    }
    return make_pair(flow, cost);
}
};

```

4.4 Min Cost Circulation [ea0477]

```

template <typename F, typename C>
struct MinCostCirculation {
    struct ep { int to; F flow; C cost; };
    int n; vector<int> vis; int visc;
    vector<int> fa, fae; vector<vector<int>> g;
    vector<ep> e; vector<C> pi;
    MinCostCirculation(int n_) : n(n_), vis(n), visc(0), g(n), pi(n) {}
    void add_edge(int u, int v, F fl, C cs) {
        g[u].emplace_back((int)e.size());
        e.emplace_back(v, fl, cs);
        g[v].emplace_back((int)e.size());
        e.emplace_back(u, 0, -cs);
    }
    C phi(int x) {
        if (fa[x] == -1) return 0;
        if (vis[x] == visc) return pi[x];
        vis[x] = visc;
        return pi[x] = phi(fa[x]) - e[fae[x]].cost;
    }
    int lca(int u, int v) {
        for (; u != -1 || v != -1; swap(u, v)) if (u != -1)
            {

```

```

                if (vis[u] == visc) return u;
                vis[u] = visc; u = fa[u];
            }
        }
        return -1;
    }
    void pushflow(int x, C &cost) {
        int v = e[x ^ 1].to, u = e[x].to; ++visc;
        if (int w = lca(u, v); w == -1) {
            while (v != -1)
                swap(x ^ 1, fae[v]), swap(u, fa[v]), swap(u, v);
        }
        else {
            int z = u, dir = 0; F f = e[x].flow;
            vector<int> cyc = {x};
            for (int d : {0, 1})
                for (int i = (d ? u : v); i != w; i = fa[i]) {
                    cyc.push_back(fae[i] ^ d);
                    if (chmin(f, e[fae[i] ^ d].flow)) z = i, dir = d;
                }
            for (int i : cyc) {
                e[i].flow -= f; e[i ^ 1].flow += f;
                cost += f * e[i].cost;
            }
            if (dir) x ^ 1, swap(u, v);
            while (u != z)
                swap(x ^ 1, fae[v]), swap(u, fa[v]), swap(u, v);
        }
    }
    void dfs(int u) {
        vis[u] = visc;
        for (int i : g[u])
            if (int v = e[i].to; vis[v] != visc and e[i].flow)
                fa[v] = u, fae[v] = i, dfs(v);
    }
    C simplex() {
        fa.assign(g.size(), -1); fae.assign(g.size(), -1);
        C cost = 0; ++visc; dfs(0);
        for (int fail = 0; fail < ssize(e); )
            for (int i = 0; i < ssize(e); i++)
                if (e[i].flow and e[i].cost < phi(e[i ^ 1].to) - phi(e[i].to))
                    fail = 0, pushflow(i, cost), ++visc;
                else ++fail;
        return cost;
    }
};

```

4.5 Kuhn Munkres [a138de]

```

template <typename T>
struct KM { // 0-based
    const T INF = 1 << 30;
    T w[N][N], hl[N], hr[N], slk[N];
    int fl[N], fr[N], pre[N], n;
    bool vl[N], vr[N];
    queue<int> q;
    KM() {}
    void init(int _n) {
        n = _n;
        for (int i = 0; i < n; ++i)
            for (int j = 0; j < n; ++j) w[i][j] = -INF;
    }
    void add_edge(int a, int b, T wei) { w[a][b] = wei; }
    bool check(int x) {
        if (vl[x] = 1, ~fl[x])
            return q.push(fl[x]), vr[fl[x]] = 1;
        while (~x) swap(x, fr[fl[x] = pre[x]]);
        return 0;
    }
    void bfs(int s) {
        fill(slk, slk + n, INF), fill(vl, vl + n, 0);
        fill(vr, vr + n, 0);
        while (!q.empty()) q.pop();
        q.push(s), vr[s] = 1;
        while (true) {
            T d;
            while (!q.empty()) {
                int y = q.front(); q.pop();
                for (int x = 0; 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;
}
}
T solve() {
    fill(fl, fl + n, -1), fill(fr, fr + n, -1);
    fill(hr, hr + n, 0);
    for (int i = 0; i < n; ++i)
        hl[i] = *max_element(w[i], w[i] + n);
    for (int i = 0; i < n; ++i) bfs(i);
    T res = 0;
    for (int i = 0; i < n; ++i) res += w[i][fl[i]];
    return res;
}
};

```

4.6 Stoer Wagner (Min-cut) [ac255a]

```

struct SW {
    int g[N][N], sum[N], n;
    bool vis[N], dead[N];
    void init(int _n) {
        n = _n;
        for (int i = 0; i < n; ++i) fill(g[i], g[i] + n, 0);
        fill(dead, dead + n, false);
    }
    void add_edge(int u, int v, int w) {
        g[u][v] += w, g[v][u] += w;
    }
    int run() {
        int ans = 1 << 30;
        for (int round = 0; round + 1 < n; ++round) {
            fill(vis, vis + n, false), fill(sum, sum + n, 0);
            int num = 0, s = -1, t = -1;
            while (num < n - round) {
                int now = -1;
                for (int i = 0; i < n; ++i) if (!vis[i] && !
                    dead[i]) {
                    if (now == -1 || sum[now] < sum[i]) now = i;
                }
                s = t, t = now;
                vis[now] = true, num++;
                for (int i = 0; i < n; ++i) if (!vis[i] && !
                    dead[i]) {
                    sum[i] += g[now][i];
                }
            }
            ans = min(ans, sum[t]);
            for (int i = 0; i < n; ++i) {
                g[i][s] += g[i][t];
                g[s][i] += g[t][i];
            }
            dead[t] = true;
        }
        return ans;
    }
};

```

4.7 GomoryHu Tree [90ead2]

```

vector<array<int, 3>> GomoryHu(Dinic<int> flow) {
    // Tree edge min = mincut (0-based)
    int n = flow.n;
    vector<array<int, 3>> ans;
    vector<int> rt(n);
    for (int i = 1; i < n; ++i) {
        int t = rt[i];
        flow.reset();

```

```

        ans.pb({i, t, flow.solve(i, t)});
        flow.bfs();
        for (int j = i + 1; j < n; ++j)
            if (rt[j] == t && flow.dis[j] != -1) rt[j] = i;
    }
    return ans;
}

```

4.8 General Graph Matching [2b7f20]

```

struct Matching { // 0-based
    int fa[N], pre[N], match[N], s[N], v[N], n, tk;
    vector<int> g[N];
    queue<int> q;
    int Find(int u) {
        return u == fa[u] ? u : fa[u] = Find(fa[u]);
    }
    int lca(int x, int y) {
        tk++;
        x = Find(x), y = Find(y);
        for (; ; swap(x, y)) {
            if (x != n) {
                if (v[x] == tk) return x;
                v[x] = tk;
                x = Find(pre[match[x]]);
            }
        }
    }
    void blossom(int x, int y, int l) {
        while (Find(x) != l) {
            pre[x] = y, y = match[x];
            if (s[y] == 1) q.push(y), s[y] = 0;
            if (fa[x] == x) fa[x] = l;
            if (fa[y] == y) fa[y] = l;
            x = pre[y];
        }
    }
    bool bfs(int r) {
        for (int i = 0; i <= n; ++i) fa[i] = i, s[i] = -1;
        while (!q.empty()) q.pop();
        q.push(r);
        s[r] = 0;
        while (!q.empty()) {
            int x = q.front(); q.pop();
            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, 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;
    }
    int solve() {
        int res = 0;
        for (int x = 0; x < n; ++x) {
            if (match[x] == n) res += bfs(x);
        }
        return res;
    }
    void init(int _n) {
        n = _n, tk = 0;
        for (int i = 0; i <= n; ++i) match[i] = pre[i] = n;
        for (int i = 0; i < n; ++i) g[i].clear(), v[i] = 0;
    }
    void add_edge(int u, int v) {
        g[u].push_back(v), g[v].push_back(u);
    }
};

```

4.9 Flow notes

- Bipartite Matching Restore Answer


```
runBfs(); Answer is {!vis[x]|x ∈ L} ∪ {vis[x]|x ∈ R}
```
- Bipartite Minimum Weight Vertex Covering

$S \rightarrow \{x|x \in L\}$, cap = weight of vertex x
 $\{x|x \in L\} \rightarrow \{y|y \in R\}$, cap = ∞
 $\{y|y \in R\} \rightarrow T$, cap = weight of vertex y

For general version, change Dinic to MCMF and:

$S \rightarrow \{x|x \in L\}$, cap = weight of vertex x , cost = 0
 $\{x|x \in L\} \rightarrow \{y|y \in R\}$, cap = ∞ , cost = $-w$
 $\{y|y \in R\} \rightarrow T$, cap = weight of vertex y , cost = 0
- Useful Lemma

(Bipartite Maximum Weight Independent Set) + (Bipartite Minimum Weight Vertex Covering) = weight sum
- Min Cut Model

choose A but not choose B cost x : $A \rightarrow B$, cap = x
 choose A cost x : $A \rightarrow T$, cap = x
 not choose A cost x : $S \rightarrow A$, cap = x
 choose A gain $x \implies$ not choose A cost x , $tot += x$
 choose A and choose B cost x : NO!!!
 Bipartite: can flip one side
- Min Cut Restore Answer


```
runBfs(); Answer is {vis[x]|x ∈ V}
```
- Maximum/Minimum flow with lower bound / Circulation problem
 - Construct super source S and sink T .
 - For each edge (x, y, l, u) , connect $x \rightarrow y$ with capacity $u - l$.
 - For each vertex v , denote by $in(v)$ the difference between the sum of incoming lower bounds and the sum of outgoing lower bounds.
 - If $in(v) > 0$, connect $S \rightarrow v$ with capacity $in(v)$, otherwise, connect $v \rightarrow T$ with capacity $-in(v)$.
 - To maximize, connect $t \rightarrow s$ with capacity ∞ (skip this in circulation problem), and let f be the maximum flow from S to T . If $f \neq \sum_{v \in V, in(v) > 0} in(v)$, there's no solution. Otherwise, the maximum flow from s to t is the answer.
 - To minimize, let f be the maximum flow from S to T . Connect $t \rightarrow s$ with capacity ∞ and let the flow from S to T be f' . If $f + f' \neq \sum_{v \in V, in(v) > 0} in(v)$, there's no solution. Otherwise, f' is the answer.
 - The solution of each edge e is $l_e + f_e$, where f_e corresponds to the flow of edge e on the graph.
- Construct minimum vertex cover from maximum matching M on bipartite graph (X, Y)
 - Redirect every edge: $y \rightarrow x$ if $(x, y) \in M$, $x \rightarrow y$ otherwise.
 - DFS from unmatched vertices in X .
 - $x \in X$ is chosen iff x is unvisited.
 - $y \in Y$ is chosen iff y is visited.
- Minimum cost cyclic flow
 - Construct super source S and sink T
 - For each edge (x, y, c) , connect $x \rightarrow y$ with $(cost, cap) = (c, 1)$ if $c > 0$, otherwise connect $y \rightarrow x$ with $(cost, cap) = (-c, 1)$
 - For each edge with $c < 0$, sum these cost as K , then increase $d(y)$ by 1, decrease $d(x)$ by 1
 - For each vertex v with $d(v) > 0$, connect $S \rightarrow v$ with $(cost, cap) = (0, d(v))$
 - For each vertex v with $d(v) < 0$, connect $v \rightarrow T$ with $(cost, cap) = (0, -d(v))$
 - Flow from S to T , the answer is the cost of the flow $C + K$
- Maximum density induced subgraph
 - Binary search on answer, suppose we're checking answer T
 - Construct a max flow model, let K be the sum of all weights
 - Connect source $s \rightarrow v$, $v \in G$ with capacity K
 - For each edge (u, v, w) in G , connect $u \rightarrow v$ and $v \rightarrow u$ with capacity w
 - For $v \in G$, connect it with sink $v \rightarrow t$ with capacity $K + 2T - (\sum_{e \in E(v)} w(e)) - 2w(v)$
 - T is a valid answer if the maximum flow $f < K|V|$
- Minimum weight edge cover
 - For each $v \in V$ create a copy v' , and connect $u' \rightarrow v'$ with weight $w(u, v)$.
 - Connect $v \rightarrow v'$ with weight $2\mu(v)$, where $\mu(v)$ is the cost of the cheapest edge incident to v .
 - Find the minimum weight perfect matching on G' .
- Project selection problem
 - If $p_v > 0$, create edge (s, v) with capacity p_v ; otherwise, create edge (v, t) with capacity $-p_v$.
 - Create edge (u, v) with capacity w with w being the cost of choosing u without choosing v .
 - The mincut is equivalent to the maximum profit of a subset of projects.

5 String

5.1 AC Automaton [b9fe7c]

```
struct AC {
    int ch[N][26], to[N][26], fail[N], sz;
    vector<int> g[N];
    int cnt[N];
    AC() {sz = 0, extend();}
    void extend() {fill(ch[sz], ch[sz] + 26, 0), sz++;}
    int nxt(int u, int v) {
        if (!ch[u][v]) ch[u][v] = sz, extend();
        return ch[u][v];
    }
    int insert(string s) {
        int now = 0;
        for (char c : s) now = nxt(now, c - 'a');
        cnt[now]++;
        return now;
    }
    void build_fail() {
        queue<int> q;
        for (int i = 0; i < 26; ++i) if (ch[0][i]) {
            to[0][i] = ch[0][i];
            q.push(ch[0][i]);
            g[0].push_back(ch[0][i]);
        }
        while (!q.empty()) {
            int v = q.front(); q.pop();
            for (int j = 0; j < 26; ++j) {
                to[v][j] = ch[v][j] ? ch[v][j] : to[fail[v]][j];
            }
            for (int i = 0; i < 26; ++i) if (ch[v][i]) {
                int u = ch[v][i], k = fail[v];
                while (k && !ch[k][i]) k = fail[k];
                if (ch[k][i]) k = ch[k][i];
                fail[u] = k;
                cnt[u] += cnt[k], g[k].push_back(u);
                q.push(u);
            }
        }
    }
    int match(string &s) {
        int now = 0, ans = 0;
        for (char c : s) {
            now = to[now][c - 'a'];
            ans += cnt[now];
        }
        return ans;
    }
};
```

5.2 Lyndon Factorization [a9eeb0]

```
// partition s = w[0] + w[1] + ... + w[k-1],
// w[0] >= w[1] >= ... >= w[k-1]
// each w[i] strictly smaller than all its suffix
vector<string> duval(const string &s) {
    vector<string> ans;
    for (int n = (int)s.size(), i = 0, j, k; i < n; ) {
        for (j = i + 1, k = i; j < n && s[k] <= s[j]; j++)
            k = (s[k] < s[j] ? i : k + 1);
        for (; i <= k; i += j - k)
            ans.pb(s.substr(i, j - k)); // s.substr(1, len)
    }
    return ans;
}
```

5.3 KMP [5ac553]

```
vector<int> build_fail(string &s) {
    vector<int> f(s.length() + 1, 0);
    int k = 0;
    for (int i = 1; i < s.length(); ++i) {
        while (k && s[k] != s[i])
            k = f[k];
        if (s[k] == s[i])
            k++;
        f[i + 1] = k;
    }
}
```

```

    return f;
}
int match(string &s, string &t) {
    vector<int> f = build_fail(t);
    int k = 0, ans = 0;
    for (int i = 0; i < s.length(); ++i) {
        while (k && s[i] != t[k])
            k = f[k];
        if (s[i] == t[k])
            k++;
        if (k == t.length())
            ans++, k = f[k];
    }
    return ans;
}

```

5.4 Manacher [643e55]

```

int z[MAXN]; // 0-base
/* center i: radius z[i * 2 + 1] / 2
   center i, i + 1: radius z[i * 2 + 2] / 2
   both aba, abba have radius 2 */
void Manacher(string tmp) {
    string s = "%";
    int l = 0, r = 0;
    for (char c : tmp) s.pb(c), s.pb('%');
    for (int i = 0; i < s.size(); ++i) {
        z[i] = r > i ? min(z[2 * l - i], r - i) : 1;
        while (i - z[i] >= 0 && i + z[i] < s.size()
            && s[i + z[i]] == s[i - z[i]]) ++z[i];
        if (z[i] + i > r) r = z[i] + i, l = i;
    }
}

```

5.5 Minimum Rotate [acab8e]

```

string mcp(string s) {
    int n = s.size(), i = 0, j = 1;
    s += s;
    while (i < n && j < n) {
        int k = 0;
        while (k < n && s[i + k] == s[j + k]) k++;
        if (s[i + k] <= s[j + k]) j += k + 1;
        else i += k + 1;
        if (i == j) j++;
    }
    int ans = (i < n ? i : j);
    return s.substr(ans, n);
}

```

5.6 Palindrome Tree [8a071b]

```

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

```

```

}
void dpcnt() {
    for (int i = SZ(st) - 1; i >= 0; i--) {
        auto nd = st[i];
        st[nd.fail].cnt += nd.cnt;
    }
}
int size() { return (int)st.size() - 2; }
};

```

5.7 Palindrome Partition [c85c05]

```

// in PAM
/* node */ int dif = 0, slink = 0, g = 0;
vector<int> dp = {0};
// add
if (!st[cur].nxt[c]) {
    // ...
    st[now].dif = st[now].len - st[st[now].fail].len;
    if (st[now].dif == st[st[now].fail].dif)
        st[now].slink = st[st[now].fail].slink;
    else st[now].slink = st[now].fail;
}
dp.pb(0);
for (int x = last; x > 1; x = st[x].slink) {
    st[x].g = dp[n - st[st[x].slink].len - st[x].dif];
    if (st[x].dif == st[st[x].fail].dif)
        st[x].g = min(st[x].g, st[st[x].fail].g);
    dp[n] = min(dp[n], st[x].g + 1);
}

```

5.8 Repetition [f3da14]

```

int to_left[N], to_right[N];
vector<array<int, 3>> rep; // l, r, len.
// substr( [l, r], len * 2 ) are tandem
void findRep(string &s, int l, int r) {
    if (r - l == 1) return;
    int m = l + r >> 1;
    findRep(s, l, m), findRep(s, m, r);
    string sl = s.substr(l, m - l);
    string sr = s.substr(m, r - m);
    vector<int> Z = buildZ(sr + "#" + sl);
    for (int i = 1; i < m; ++i)
        to_right[i] = Z[r - m + 1 + i - 1];
    reverse(all(sl));
    Z = buildZ(sl);
    for (int i = 1; i < m; ++i)
        to_left[i] = Z[m - i - 1];
    reverse(all(sl));
    for (int i = 1; i + 1 < m; ++i) {
        int k1 = to_left[i], k2 = to_right[i + 1];
        int len = m - i - 1;
        if (k1 < 1 || k2 < 1 || len < 2) continue;
        int tl = max(1, len - k2), tr = min(len - 1, k1);
        if (tl <= tr) rep.pb({i + 1 - tr, i + 1 - tl, len});
    }
    Z = buildZ(sr);
    for (int i = m; i < r; ++i) to_right[i] = Z[i - m];
    reverse(all(sl)), reverse(all(sr));
    Z = buildZ(sl + "#" + sr);
    for (int i = m; i < r; ++i)
        to_left[i] = Z[m - 1 + 1 + r - i - 1];
    reverse(all(sl)), reverse(all(sr));
    for (int i = m; i + 1 < r; ++i) {
        int k1 = to_left[i], k2 = to_right[i + 1];
        int len = i - m + 1;
        if (k1 < 1 || k2 < 1 || len < 2) continue;
        int tl = max(len - k2, 1), tr = min(len - 1, k1);
        if (tl <= tr)
            rep.pb({i + 1 - len - tr, i + 1 - len - tl, len});
    }
    Z = buildZ(sr + "#" + sl);
    for (int i = 1; i < m; ++i)
        if (Z[r - m + 1 + i - 1] >= m - i)
            rep.pb({i, i, m - i});
}

```

5.9 Suffix Array [60547e]

```

#define FOR(i, a, b) for(int i = a; i <= b; i++)
#define ROF(i, a, b) for(int i = a; i >= b; i--)

```

```

int sa[N], tmp[2][N], c[N], rk[N], lcp[N];
void buildSA(string s) {
    int *x = tmp[0], *y = tmp[1], m = 256, n = (int)s.size();
    FOR(i, 0, m - 1) c[i] = 0;
    FOR(i, 0, n - 1) c[x[i]] = s[i]++;
    FOR(i, 1, m - 1) c[i] += c[i - 1];
    ROF(i, n - 1, 0) sa[--c[x[i]]] = i;
    for (int k = 1; k < n; k <= 1) {
        FOR(i, 0, m - 1) c[i] = 0;
        FOR(i, 0, n - 1) c[x[i]]++;
        FOR(i, 1, m - 1) c[i] += c[i - 1];
        int p = 0;
        FOR(i, n - k, n - 1) y[p++] = i;
        FOR(i, 0, n - 1) if (sa[i] >= k) y[p++] = sa[i] - k;
        ROF(i, n - 1, 0) sa[--c[x[y[i]]]] = y[i];
        y[sa[0]] = p = 0;
        FOR(i, 1, n - 1) {
            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])) p++;
            y[sa[i]] = p;
        }
        if (n == p + 1) break;
        swap(x, y), m = p + 1;
    }
}
void buildLCP(string s) {
    // lcp[i] = LCP(sa[i - 1], sa[i])
    // lcp(i, j) = min(lcp[rk[i] + 1], lcp[rk[i] + 2], ..., lcp[rk[j]])
    int n = (int)s.size(), val = 0;
    FOR(i, 0, n - 1) rk[sa[i]] = i;
    FOR(i, 0, n - 1) {
        if (!rk[i]) lcp[rk[i]] = 0;
        else {
            if (val) val--;
            int p = sa[rk[i] - 1];
            while (val + i < n && val + p < n && s[val + i] == s[val + p]) val++;
            lcp[rk[i]] = val;
        }
    }
}

```

5.10 SAIS (C++20) [6f26bc]

```

auto sais(const auto &s) {
    const int n = SZ(s), z = ranges::max(s) + 1;
    if (n == 1) return vector{0};
    vector<int> c(z); for (int x : s) ++c[x];
    partial_sum(ALL(c), begin(c));
    vector<int> sa(n); auto I = views::iota(0, n);
    vector<bool> t(n, true);
    for (int i = n - 2; i >= 0; --i)
        t[i] = (s[i] == s[i + 1] ? t[i + 1] : s[i] < s[i + 1]);
    auto is_lms = views::filter([&t](int x) {
        return x && t[x] && !t[x - 1];
    });
    auto induce = [&] {
        for (auto x = c; int y : sa)
            if (y-- if (!t[y]) sa[x[s[y] - 1]++] = y;
        for (auto x = c; int y : sa | views::reverse)
            if (y-- if (t[y]) sa[--x[s[y]]] = y;
    };
    vector<int> lms, q(n); lms.reserve(n);
    for (auto x = c; int i : I | is_lms)
        q[i] = SZ(lms), lms.pb(sa[--x[s[i]]] = i);
    induce(); vector<int> ns(SZ(lms));
    for (int j = -1, nz = 0; int i : sa | is_lms) {
        if (j >= 0) {
            int len = min({n - i, n - j, lms[q[i] + 1] - i});
            ns[q[i]] = nz += lexicographical_compare(
                begin(s) + j, begin(s) + j + len,
                begin(s) + i, begin(s) + i + len);
        }
        j = i;
    }
    fill(ALL(sa), 0); auto nsa = sais(ns);
    for (auto x = c; int y : nsa | views::reverse)
        y = lms[y], sa[--x[s[y]]] = y;
}

```

```

return induce(), sa;
}
// sa[i]: sa[i]-th suffix is the i-th lexicographically
// smallest suffix.
// hi[i]: LCP of suffix sa[i] and suffix sa[i - 1].
struct Suffix {
    int n; vector<int> sa, hi, ra;
    Suffix(const auto &s, int _n) : n(_n), hi(n), ra(n) {
        vector<int> s(n + 1); // s[n] = 0;
        copy_n(s, n, begin(s)); // s shouldn't contain 0
        sa = sais(s); sa.erase(sa.begin());
        for (int i = 0; i < n; ++i) ra[sa[i]] = i;
        for (int i = 0, h = 0; i < n; ++i) {
            if (!ra[i]) { h = 0; continue; }
            for (int j = sa[ra[i] - 1]; max(i, j) + h < n &&
                s[i + h] == s[j + h];) ++h;
            hi[ra[i]] = h ? h - 1 : 0;
        }
    }
};

```

5.11 Suffix Automaton [277d1d]

```

struct SAM {
    int ch[N][26], len[N], link[N], pos[N], cnt[N], sz;
    // node -> strings with the same endpos set
    // length in range [len(link) + 1, len]
    // node's endpos set -> pos in the subtree of node
    // link -> longest suffix with different endpos set
    // len -> longest suffix
    // pos -> end position
    // cnt -> size of endpos set
    SAM() { len[0] = 0, link[0] = -1, pos[0] = 0, cnt[0] = 0, sz = 1; }
    void build(string s) {
        int last = 0;
        for (int i = 0; i < s.length(); ++i) {
            char c = s[i];
            int cur = sz++;
            len[cur] = len[last] + 1, pos[cur] = i + 1;
            int p = last;
            while (~p && !ch[p][c - 'a'])
                ch[p][c - 'a'] = cur, p = link[p];
            if (p == -1) link[cur] = 0;
            else {
                int q = ch[p][c - 'a'];
                if (len[p] + 1 == len[q]) {
                    link[cur] = q;
                } else {
                    int nxt = sz++;
                    len[nxt] = len[p] + 1, link[nxt] = link[q];
                    pos[nxt] = pos[q];
                    for (int j = 0; j < 26; ++j)
                        ch[nxt][j] = ch[q][j];
                    while (~p && ch[p][c - 'a'] == q)
                        ch[p][c - 'a'] = nxt, p = link[p];
                    link[q] = link[cur] = nxt;
                }
            }
            cnt[cur]++;
            last = cur;
        }
        vector<int> p(sz);
        iota(all(p), 0);
        sort(all(p),
            [&](int i, int j) { return len[i] > len[j]; });
        for (int i = 0; i < sz; ++i)
            cnt[link[p[i]]] += cnt[p[i]];
    }
} sam;

```

5.12 Z Value [a8c33c]

```

vector<int> build(string s) {
    int n = s.length();
    vector<int> Z(n);
    int l = 0, r = 0;
    for (int i = 0; i < n; ++i) {
        Z[i] = max(min(Z[i - 1], r - i), 0);
        while (i + Z[i] < s.size() && s[Z[i]] == s[i + Z[i]]) {
            ++Z[i];
        }
    }
}

```

```

    l = i, r = i + Z[i], Z[i]++;
}
}
return Z;
}

```

6 Math

6.1 Berlekamp Massey [c34682]

```

const int MOD=998244353;
vector<ll> BerlekampMassey(vector<ll> a) {
    // find min |c| such that a_n = sum c_j * a_{n-j-1}, 0-based
    // O(N^2), if |c| = k, |a| >= 2k sure correct
    auto f = [&](vector<ll> v, ll c) {
        for (ll &x : v) x = x * c % MOD;
        return v;
    };
    vector<ll> c, best;
    int pos = 0, n = a.size();
    for (int i = 0; i < n; ++i) {
        ll error = a[i];
        for (int j = 0; j < c.size(); ++j) error = ((error
            - c[j] * a[i-1-j]) % MOD + MOD) % MOD;
        if (error == 0) continue;
        ll inve = inv(error, MOD);
        if (c.empty()) {
            c.resize(i+1);
            pos = i;
            best.pb(inve);
        } else {
            vector<ll> fix = f(best, error);
            fix.insert(fix.begin(), i-pos-1, 0);
            if (fix.size() >= c.size()) {
                best = f(c, inve > 0 ? MOD-inve : 0);
                best.insert(best.begin(), inve);
                pos = i;
                c.resize(fix.size());
            }
            for (int j = 0; j < fix.size(); ++j) c[j] = (c[j]
                + fix[j]) % MOD;
        }
    }
    return c;
}

```

6.2 Characteristic Polynomial [cd559d]

```

#define rep(x, y, z) for (int x=y; x<z; x++)
using VI = vector<int>; using VVI = vector<VI>;
void Hessenberg(VVI &H, int N) {
    for (int i = 0; i < N-2; ++i) {
        for (int j = i+1; j < N; ++j) if (H[j][i]) {
            rep(k, i, N) swap(H[i+1][k], H[j][k]);
            rep(k, 0, N) swap(H[k][i+1], H[k][j]);
            break;
        }
        if (!H[i+1][i]) continue;
        for (int j = i+2; j < N; ++j) {
            int co = mul(modinv(H[i+1][i]), H[j][i]);
            rep(k, i, N) subeq(H[j][k], mul(H[i+1][k], co));
            rep(k, 0, N) addeq(H[k][i+1], mul(H[k][j], co));
        }
    }
}
VI CharacteristicPoly(VVI A) {
    int N = (int)A.size(); Hessenberg(A, N);
    VVI P(N+1, VI(N+1)); P[0][0] = 1;
    for (int i = 1; i <= N; ++i) {
        rep(j, 0, i+1) P[i][j] = j ? P[i-1][j-1] : 0;
        for (int j = i-1; val = 1; j >= 0; --j) {
            int co = mul(val, A[j][i-1]);
            rep(k, 0, j+1) subeq(P[i][k], mul(P[j][k], co));
            if (j) val = mul(val, A[j][j-1]);
        }
    }
    if (N & 1) for (int &x: P[N]) x = sub(0, x);
    return P[N]; // test: 2021 PTZ Korea K
}

```

6.3 Discrete Logarithm [da27bf]

```

int DiscreteLog(int s, int x, int y, int m) {
    constexpr int kStep = 32000;
    unordered_map<int, int> p;
    int b = 1;
    for (int i = 0; i < kStep; ++i) {
        p[y] = i;
        y = 1LL * y * x % m;
        b = 1LL * b * x % m;
    }
    for (int i = 0; i < m+10; i += kStep) {
        s = 1LL * s * b % m;
        if (p.find(s) != p.end()) return i + kStep - p[s];
    }
    return -1;
}
int DiscreteLog(int x, int y, int m) {
    if (m == 1) return 0;
    int s = 1;
    for (int i = 0; i < 100; ++i) {
        if (s == y) return i;
        s = 1LL * s * x % m;
    }
    if (s == y) return 100;
    int p = 100 + DiscreteLog(s, x, y, m);
    if (fpow(x, p, m) != y) return -1;
    return p;
}

```

6.4 Extgcd [d8bbd5]

```

// a * p.first + b * p.second = gcd(a, b)
pair<ll, ll> extgcd(ll a, ll b) {
    pair<ll, ll> res;
    if (a < 0) {
        res = extgcd(-a, b);
        res.first *= -1;
        return res;
    }
    if (b < 0) {
        res = extgcd(a, -b);
        res.second *= -1;
        return res;
    }
    if (b == 0) return {1, 0};
    res = extgcd(b, a % b);
    return {res.second, res.first - res.second * (a / b)};
}

```

6.5 Floor Sum [49de67]

```

// sum_{i=0}^{n-1} floor((a * i + b) / m) in log(n + m + a + b)
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;
}

```

6.6 Factorial Mod P^k [c324f3]

```

// O(p^k + log^2 n), pk = p^k
ll prod[MAXP];
ll fac_no_p(ll n, ll p, ll pk) {
    prod[0] = 1;
    for (int i = 1; i <= pk; ++i)
        if (i % p) prod[i] = prod[i-1] * i % pk;
        else prod[i] = prod[i-1];
    ll rt = 1;
    for (; n /= p) {
        rt = rt * mpow(prod[pk], n / pk, pk) % pk;
        rt = rt * prod[n % pk] % pk;
    }
    return rt;
}
// (n! without factor p) % p^k

```

6.7 Gaussian Elimination [fa0977]

```
using VI = vector<int>; // be careful if A.empty()
using VVI = vector<VI>; // ensure that 0 <= x < mod
pair<VI, VVI> gauss(VVI A, VI b) { // solve Ax=b
    const int N = (int)A.size(), M = (int)A[0].size();
    vector<int> depv, free(M, true); int rk = 0;
    for (int i = 0; i < M; i++) {
        int p = -1;
        for (int j = rk; j < N; j++)
            if (p == -1 || abs(A[j][i]) > abs(A[p][i]))
                p = j;
        if (p == -1 || A[p][i] == 0) continue;
        swap(A[p], A[rk]); swap(b[p], b[rk]);
        const int inv = modinv(A[rk][i]);
        for (int &x : A[rk]) x = mul(x, inv);
        b[rk] = mul(b[rk], inv);
        for (int j = 0; j < N; j++) if (j != rk) {
            int z = A[j][i];
            for (int k = 0; k < M; k++)
                A[j][k] = sub(A[j][k], mul(z, A[rk][k]));
            b[j] = sub(b[j], mul(z, b[rk]));
        }
        depv.push_back(i); free[i] = false; ++rk;
    }
    for (int i = rk; i < N; i++)
        if (b[i] != 0) return {{}, {}}; // not consistent
    VI x(M); VVI h;
    for (int i = 0; i < rk; i++) x[depv[i]] = b[i];
    for (int i = 0; i < M; i++) if (free[i]) {
        h.emplace_back(M); h.back()[i] = 1;
        for (int j = 0; j < rk; j++)
            h.back()[depv[j]] = sub(0, A[j][i]);
    }
    return {x, h}; // solution = x + span(h[i])
}
```

6.8 Linear Function Mod Min [5552e3]

```
11 topos(11 x, 11 m) {x %= m; if (x < 0) x += m; return x;}
//min value of ax + b (mod m) for x \in [0, n - 1]. O(log m)
11 min_rem(11 n, 11 m, 11 a, 11 b) {
    a = topos(a, m), b = topos(b, m);
    for (11 g = __gcd(a, m); g > 1; ) return g * min_rem(n / g, m / g, a / g, b / g) + (b % g);
    for (11 nn, nm, na, nb; a; n = nn, m = nm, a = na, b = nb) {
        if (a <= m - a) {
            nn = (a * (n - 1) + b) / m;
            if (!nn) break;
            nn += (b < a);
            nm = a, na = topos(-m, a);
            nb = b < a ? b : topos(b - m, a);
        } else {
            11 lst = b - (n - 1) * (m - a);
            if (lst >= 0) {b = lst; break;}
            nn = -(lst / m) + (lst % m < -a) + 1;
            nm = m - a, na = m % (m - a), nb = b % (m - a);
        }
    }
    return b;
}
//min value of ax + b (mod m) for x \in [0, n - 1],
//also return min x to get the value. O(log m)
//{value, x}
pair<11, 11> min_rem_pos(11 n, 11 m, 11 a, 11 b) {
    a = topos(a, m), b = topos(b, m);
    11 mn = min_rem(n, m, a, b), g = __gcd(a, m);
    //ax = (mn - b) (mod m)
    11 x = (extgcd(a, m).first + m) * ((mn - b + m) / g) % (m / g);
    return {mn, x};
}
```

6.9 MillerRabin PollardRho [07ddf2]

```
11 mul(11 x, 11 y, 11 p) {return (x * y - (11)((long double)x / p * y) * p + p) % p;}
vector<11> chk = {2, 325, 9375, 28178, 450775, 9780504, 1795265022};
```

```
11 Pow(11 a, 11 b, 11 n) {11 res = 1; for (; b; b >>= 1, a = mul(a, a, n)) if (b & 1) res = mul(res, a, n); return res;}
bool check(11 a, 11 d, int s, 11 n) {
    a = Pow(a, d, n);
    if (a <= 1) return 1;
    for (int i = 0; i < s; ++i, a = mul(a, a, n)) {
        if (a == 1) return 0;
        if (a == n - 1) return 1;
    }
    return 0;
}
bool IsPrime(11 n) {
    if (n < 2) return 0;
    if (n % 2 == 0) return n == 2;
    11 d = n - 1, s = 0;
    while (d % 2 == 0) d >>= 1, ++s;
    for (11 i : chk) if (!check(i, d, s, n)) return 0;
    return 1;
}
const vector<11> small = {2, 3, 5, 7, 11, 13, 17, 19};
11 FindFactor(11 n) {
    if (IsPrime(n)) return 1;
    for (11 p : small) if (n % p == 0) return p;
    11 x, y = 2, d, t = 1;
    auto f = [&](11 a) {return (mul(a, a, n) + t) % n;};
    for (int l = 2; l <= 1) {
        x = y;
        int m = min(1, 32);
        for (int i = 0; i < l; i += m) {
            d = 1;
            for (int j = 0; j < m; ++j) {
                y = f(y), d = mul(d, abs(x - y), n);
            }
            11 g = __gcd(d, n);
            if (g == n) {
                l = 1, y = 2, ++t;
                break;
            }
            if (g != 1) return g;
        }
    }
}
map<11, int> PollardRho(11 n) {
    map<11, int> res;
    if (n == 1) return res;
    if (IsPrime(n)) return ++res[n], res;
    11 d = FindFactor(n);
    res = PollardRho(n / d);
    auto res2 = PollardRho(d);
    for (auto [x, y] : res2) res[x] += y;
    return res;
}
```

6.10 Quadratic Residue [e0bf30]

```
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;
}
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;
}

```

6.11 Simplex [ece29e]

```

struct Simplex { // 0-based
    using T = long double;
    static const int N = 410, M = 30010;
    const T eps = 1e-7;
    int n, m;
    int Left[M], Down[N];
    // Ax <= b, max c^T x
    // result : v, xi = sol[i]. 1 based
    T a[M][N], b[M], c[N], v, sol[N];
    bool eq(T a, T b) {return fabs(a - b) < eps;}
    bool ls(T a, T b) {return a < b && !eq(a, b);}
    void init(int _n, int _m) {
        n = _n, m = _m, v = 0;
        for (int i = 0; i < m; ++i) for (int j = 0; j < n;
            ++j) a[i][j] = 0;
        for (int i = 0; i < m; ++i) b[i] = 0;
        for (int i = 0; i < n; ++i) c[i] = sol[i] = 0;
    }
    void pivot(int x, int y) {
        swap(Left[x], Down[y]);
        T k = a[x][y]; a[x][y] = 1;
        vector<int> nz;
        for (int i = 0; i < n; ++i) {
            a[x][i] /= k;
            if (!eq(a[x][i], 0)) nz.push_back(i);
        }
        b[x] /= k;
        for (int i = 0; i < m; ++i) {
            if (i == x || eq(a[i][y], 0)) continue;
            k = a[i][y], a[i][y] = 0;
            b[i] -= k * b[x];
            for (int j : nz) a[i][j] -= k * a[x][j];
        }
        if (eq(c[y], 0)) return;
        k = c[y], c[y] = 0, v += k * b[x];
        for (int i : nz) c[i] -= k * a[x][i];
    }
    // 0: found solution, 1: no feasible solution, 2:
    // unbounded
    int solve() {
        for (int i = 0; i < n; ++i) Down[i] = i;
        for (int i = 0; i < m; ++i) Left[i] = n + i;
        while (1) {
            int x = -1, y = -1;
            for (int i = 0; i < m; ++i) if (ls(b[i], 0) && (x
                == -1 || b[i] < b[x])) x = i;
            if (x == -1) break;
            for (int i = 0; i < n; ++i) if (ls(a[x][i], 0) &&
                (y == -1 || a[x][i] < a[x][y])) y = i;
            if (y == -1) return 1;
            pivot(x, y);
        }
        while (1) {
            int x = -1, y = -1;
            for (int i = 0; i < n; ++i) if (ls(0, c[i]) && (y
                == -1 || c[i] > c[y])) y = i;
            if (y == -1) break;
            for (int i = 0; i < m; ++i) if (ls(0, a[i][y]) &&
                (x == -1 || b[i] / a[i][y] < b[x] / a[x][y]))
                x = i;
            if (x == -1) return 2;
            pivot(x, y);
        }
        for (int i = 0; i < m; ++i) if (Left[i] < n) sol[
            Left[i]] = b[i];
    }
}

```

```

return 0;
}
};

```

6.12 Linear Programming Construction

Standard form: maximize $c^T x$ subject to $Ax \leq b$ and $x \geq 0$.
 Dual LP: minimize $b^T y$ subject to $A^T y \geq c$ and $y \geq 0$.
 x and y are optimal if and only if for all $i \in [1, n]$, either $\bar{x}_i = 0$ or $\sum_{j=1}^m A_{ij} \bar{y}_j = c_i$ holds and for all $i \in [1, m]$ either $\bar{y}_i = 0$ or $\sum_{j=1}^n A_{ij} \bar{x}_j = b_j$ holds.

1. In case of minimization, let $c'_i = -c_i$
2. $\sum_{1 \leq i \leq n} A_{ji} x_i \geq b_j \rightarrow \sum_{1 \leq i \leq n} -A_{ji} x_i \leq -b_j$
3. $\sum_{1 \leq i \leq n} A_{ji} x_i = b_j$
 - $\sum_{1 \leq i \leq n} A_{ji} x_i \leq b_j$
 - $\sum_{1 \leq i \leq n} A_{ji} x_i \geq b_j$

4. If x_i has no lower bound, replace x_i with $x_i - x'_i$

6.13 Estimation

n	2	3	4	5	6	7	8	9	20	30	40	50	100		
$p(n)$	2	3	5	7	11	15	22	30	627	5604	4e4	2e5	2e8		
n	100	1e3	1e6			1e9			1e12		1e15		1e18		
$d(i)$	12	32	240			1344			6720		26880		103680		
arg	60	840	720720	735134400	963761198400	866421317361600	897612484786617600								
n	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\binom{2n}{n}$	2	6	20	70	252	924	3432	12870	48620	184756	7e5	2e6	1e7	4e7	1.5e8
n	2	3	4	5	6	7	8	9	10	11	12	13			
B_n	2	5	15	52	203	877	4140	21147	115975	7e5	4e6	3e7			

6.14 Theorem

- Kirchhoff's Theorem

Denote L be a $n \times n$ matrix as the Laplacian matrix of graph G , where $L_{ii} = d(i)$, $L_{ij} = -c$ where c is the number of edge (i, j) in G .

- The number of undirected spanning in G is $|\det(\tilde{L}_{11})|$.
- The number of directed spanning tree rooted at r in G is $|\det(\tilde{L}_{rr})|$.

- Tutte's Matrix

Let D be a $n \times n$ matrix, where $d_{ij} = x_{ij}$ (x_{ij} is chosen uniformly at random) if $i < j$ and $(i, j) \in E$, otherwise $d_{ij} = -d_{ji}$. $\frac{\text{rank}(D)}{2}$ is the maximum matching on G .

- Cayley's Formula

- Given a degree sequence d_1, d_2, \dots, d_n for each labeled vertices, there are

$$\frac{(n-2)!}{(d_1-1)!(d_2-1)! \cdots (d_n-1)!}$$

spanning trees.

- Let $T_{n,k}$ be the number of labeled forests on n vertices with k components, such that vertex $1, 2, \dots, k$ belong to different components. Then $T_{n,k} = kn^{n-k-1}$.

- Erdős-Gallai Theorem

A sequence of non-negative integers $d_1 \geq d_2 \geq \dots \geq d_n$ can be represented as the degree sequence of a finite simple graph on n vertices if and only if $d_1 + d_2 + \dots + d_n$ is even and

$$\sum_{i=1}^k d_i \leq k(k-1) + \sum_{i=k+1}^n \min(d_i, k)$$

holds for all $1 \leq k \leq n$.

- Burnside's Lemma

Let X be a set and G be a group that acts on X . For $g \in G$, denote by X^g the elements fixed by g :

$$X^g = \{x \in X \mid gx = x\}$$

Then

$$|X/G| = \frac{1}{|G|} \sum_{g \in G} |X^g|.$$

- Gale-Ryser theorem

A pair of sequences of nonnegative integers $a_1 \geq \dots \geq a_n$ and b_1, \dots, b_n is bigraphic if and only if $\sum_{i=1}^n a_i = \sum_{i=1}^n b_i$ and $\sum_{i=1}^k a_i \leq \sum_{i=1}^n \min(b_i, k)$ holds for every $1 \leq k \leq n$.

- Fulkerson-Chen-Anstee theorem

A sequence $(a_1, b_1), \dots, (a_n, b_n)$ of nonnegative integer pairs with $a_1 \geq \dots \geq a_n$ is digraphic if and only if $\sum_{i=1}^n a_i = \sum_{i=1}^n b_i$ and $\sum_{i=1}^k a_i \leq \sum_{i=1}^k \min(b_i, k-1) + \sum_{i=k+1}^n \min(b_i, k)$ holds for every $1 \leq k \leq n$.

- Möbius inversion formula

$$\begin{aligned} - f(n) &= \sum_{d|n} g(d) \Leftrightarrow g(n) = \sum_{d|n} \mu(d) f\left(\frac{n}{d}\right) \\ - f(n) &= \sum_{n|d} g(d) \Leftrightarrow g(n) = \sum_{n|d} \mu\left(\frac{d}{n}\right) f(d) \end{aligned}$$

- Spherical cap

- A portion of a sphere cut off by a plane.
- r : sphere radius, a : radius of the base of the cap, h : height of the cap, θ : $\arcsin(a/r)$.
- Volume $= \pi h^2(3r - h)/3 = \pi h(3a^2 + h^2)/6 = \pi r^3(2 + \cos\theta)(1 - \cos\theta)^2/3$.
- Area $= 2\pi r h = \pi(a^2 + h^2) = 2\pi r^2(1 - \cos\theta)$.

- Chinese Remainder Theorem

- $x \equiv a_i \pmod{m_i}$
- $M = \prod m_i, M_i = M/m_i$
- $t_i M_i \equiv 1 \pmod{m_i}$
- $x = \sum a_i t_i M_i \pmod{M}$

6.15 General Purpose Numbers

- Bernoulli numbers

$$B_0 = 1, B_1^\pm = \pm \frac{1}{2}, B_2 = \frac{1}{6}, B_3 = 0$$

$$\sum_{j=0}^m \binom{m+1}{j} B_j = 0, \text{ EGF is } B(x) = \frac{x}{e^x - 1} = \sum_{n=0}^{\infty} B_n \frac{x^n}{n!}.$$

$$S_m(n) = \sum_{k=1}^n k^m = \frac{1}{m+1} \sum_{k=0}^m \binom{m+1}{k} B_k^+ n^{m+1-k}$$

- Stirling numbers of the second kind Partitions of n distinct elements into exactly k groups.

$$S(n, k) = S(n-1, k-1) + k S(n-1, k), S(n, 1) = S(n, n) = 1$$

$$S(n, k) = \frac{1}{k!} \sum_{i=0}^k (-1)^{k-i} \binom{k}{i} i^n$$

$$x^n = \sum_{i=0}^n S(n, i) (x)_i$$

- Pentagonal number theorem

$$\prod_{n=1}^{\infty} (1 - x^n) = 1 + \sum_{k=1}^{\infty} (-1)^k \left(x^{k(3k+1)/2} + x^{k(3k-1)/2} \right)$$

- Catalan numbers

$$C_n^{(k)} = \frac{1}{(k-1)n+1} \binom{kn}{n}$$

$$C^{(k)}(x) = 1 + x[C^{(k)}(x)]^k$$

- Eulerian numbers

Number of permutations $\pi \in S_n$ in which exactly k elements are greater than the previous element. k j :s s.t. $\pi(j) > \pi(j+1)$, $k+1$ j :s s.t. $\pi(j) \geq j$, k j :s s.t. $\pi(j) > j$.

$$E(n, k) = (n-k)E(n-1, k-1) + (k+1)E(n-1, k)$$

$$E(n, 0) = E(n, n-1) = 1$$

$$E(n, k) = \sum_{j=0}^k (-1)^j \binom{n+1}{j} (k+1-j)^n$$

6.16 Integral

$$\frac{d}{dx} \frac{\sec^2 x}{\ln|\sec x|} = \frac{-\csc^2 x}{-\ln|\csc x|} = \frac{\tan x \sec x}{\ln|\tan x + \sec x|} = \frac{-\cot x \csc x}{-\ln|\cot x + \csc x|}$$

$$\frac{d}{dx} \frac{1}{\sqrt{1-x^2}} = \frac{-1}{\sqrt{1-x^2}} = \frac{1}{x^2+1} = \frac{\sec^3 x}{\frac{1}{2}(\sec x \tan x + \ln|\sec x + \tan x|)}$$

$$\bullet \int_0^{\frac{\pi}{2}} \sin^{2k} x \, dx = \frac{\pi}{2} \cdot \frac{1 \cdot 3 \cdots (2k-1)}{2 \cdot 4 \cdots (2k)} \quad \int_0^{\frac{\pi}{2}} \sin^{2k+1} x \, dx = \frac{2 \cdot 4 \cdots (2k)}{3 \cdot 5 \cdots (2k+1)}$$

$$\bullet \iint f(x, y) \, dx \, dy = \iint f(g(u, v), h(u, v)) \left| \frac{\partial x}{\partial u} \frac{\partial y}{\partial v} - \frac{\partial x}{\partial v} \frac{\partial y}{\partial u} \right| \, du \, dv$$

$$\bullet \text{ polar: } \iint r \cdot f(r \cos \theta, r \sin \theta) \, dr \, d\theta$$

$$\text{spherical: } \iiint \rho^2 \sin \phi \cdot f(\rho \sin \phi \cos \theta, \rho \sin \phi \sin \theta, \rho \cos \theta) \, d\phi \, d\theta \, d\rho$$

7 Polynomial

7.1 FFT [43cc28]

```
const double pi=acos(-1);
typedef complex<double> cp;
const int N=(1<<17);
struct FFT
{
    // n has to be same as a.size()
    int n, rev[N];
    cp omega[N], iomega[N];
    void init(int _n) {
        n=_n;
        for(int i=0; i<n; i++) {
            omega[i]=cp(cos(2*pi/n*i), sin(2*pi/n*i));
            iomega[i]=conj(omega[i]);
        }
        int k=log2(n);
        for(int i=0; i<n; i++) {
            rev[i]=0;
            for(int j=0; j<k; j++) if(i&(1<<j))
                rev[i]|=(1<<(k-j-1));
        }
    }
    void tran(vector<cp> &a, cp* xomega)
    {
        for(int i=0; i<n; i++) if(i<rev[i])
            swap(a[i], a[rev[i]]);
        for(int len=2; len<=n; len<=1) {
            int mid=len>>1, r=n/len;
            for(int j=0; j<n; j+=len) {
                for(int i=0; i<mid; i++) {
                    cp t=xomega[r*i]*a[j+mid+i];
                    a[j+mid+i]=a[j+i]-t;
                    a[j+i]+=t;
                }
            }
        }
    }
    void fft(vector<cp> &a) {tran(a, omega);}
    void ifft(vector<cp> &a) {
        tran(a, iomega);
        for(int i=0; i<n; i++) a[i]/=n;
    }
};
```

7.2 NTT [f68103]

```
//needs fpow
//needs inv

//(2^16)+1, 65537, 3
//7*17*(2^23)+1, 998244353, 3
//1255*(2^20)+1, 1315962881, 3
//51*(2^25)+1, 1711276033, 29
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(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()(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 + 1, a + n);
    ll invn = minv(n);
    for (int i = 0; i < n; ++i) a[i] = a[i] * invn %
        P;
}
};

```

7.3 FWT [af59af]

```

void fwt(vector<int> &a) {
    // and : x += y * (1, -1)
    // or  : y += x * (1, -1)
    // xor : x = (x + y) * (1, 1/2)
    //      y = (x - y) * (1, 1/2)
    int n = __lg(a.size());
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < 1 << n; ++j) if (j >> i & 1) {
            int x = a[j ^ (1 << i)], y = a[j];
            // do something
        }
    }
}
vector<int> subs_conv(vector<int> a, vector<int> b) {
    // c_i = sum_{j & k = 0, j | k = i} a_j * b_k
    int n = __lg(a.size());
    vector<vector<int>> ha(n + 1, vector<int>(1 << n));
    vector<vector<int>> hb(n + 1, vector<int>(1 << n));
    vector<vector<int>> c(n + 1, vector<int>(1 << n));
    for (int i = 0; i < 1 << n; ++i) {
        ha[__builtin_popcount(i)][i] = a[i];
        hb[__builtin_popcount(i)][i] = b[i];
    }
    for (int i = 0; i <= n; ++i)
        or_fwt(ha[i]), or_fwt(hb[i]);
    for (int i = 0; i <= n; ++i)
        for (int j = 0; i + j <= n; ++j)
            for (int k = 0; k < 1 << n; ++k)
                // mind overflow
                c[i + j][k] += ha[i][k] * hb[j][k];
    for (int i = 0; i <= n; ++i) or_fwt(c[i], true);
    vector<int> ans(1 << n);
    for (int i = 0; i < 1 << n; ++i)
        ans[i] = c[__builtin_popcount(i)][i];
    return ans;
}

```

7.4 Polynomial [69f2b5]

```

NTT<131072 * 2, 998244353, 3> ntt;
#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;
    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;
    }
}

```

```

Poly X(*this, m), Y(rhs, m);
ntt(X.data(), m), ntt(Y.data(), m);
fi(0, m) X[i] = X[i] * Y[i] % P;
ntt(X.data(), m, true);
return X.isz(n() + rhs.n() - 1);
}
Poly Inv() const { // (*this)[0] != 0, 1e5/95ms
    if (n() == 1) return {ntt.minv((*this)[0])};
    int m = 1;
    while (m < n() * 2) m <= 1;
    Poly Xi = Poly(*this, (n() + 1) / 2).Inv().isz(m);
    Poly Y(*this, m);
    ntt(Xi.data(), m), ntt(Y.data(), m);
    fi(0, m) {
        Xi[i] *= (2 - Xi[i] * Y[i]) % P;
        if ((Xi[i] %= P) < 0) Xi[i] += P;
    }
    ntt(Xi.data(), m, true);
    return Xi.isz(n());
}
Poly Sqrt() const { // Jacobi((*this)[0], P) = 1, 1e5
    //235ms
    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);
}
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, 1e5/170ms
    return Dx().Mul(Inv()).Sz().isz(n());
}
Poly Exp() const { // (*this)[0] == 0, 1e5/360ms
    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 {
    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;
}
vector<ll> chirp_z(ll c, int m) { // P(c^i) for i=0..m
    -1
    Poly B(*this);
    int sz = max(n(), m);
    vector<ll> res(m);
    Poly C(sz * 2), iC(sz);
    ll ic = ntt.minv(c);
    fi(0, sz * 2) C[i] = ntt.mpow(c, 1LL * i * (i - 1)
        / 2 % (P - 1));
    fi(0, sz) iC[i] = ntt.mpow(ic, 1LL * i * (i - 1) /
        2 % (P - 1));
    fi(0, n()) B[i] = B[i] * iC[i] % P;
    B = B.irev().Mul(C);
    fi(0, m) res[i] = B[n() - 1 + i] * iC[i] % P;
    return res;
}
Poly shift_c(ll c) { // P(x+c)
    ll tmp = 1;
    Poly A(n()), B(n() + 1);
    fi(0, n()) {
        A[i] = (*this)[i] * fac[i] % P; // fac[i]=i!
        B[i] = tmp * in[i] % P; // in[i]=inv(i!)
        tmp = tmp * c % P;
    }
    B.irev();
    Poly C = A.Mul(B);
    A.isz(n());
    fi(0, n()) A[i] = C[n() + i] * in[i] % P;
    return A;
}

```

```

// sum_j w_j[x^j]f^i for i=0,1,...,m
vector<ll> power_proj(Poly wt, int m) { // 1e5 2s,
    MAXN >= 4 * n()
    // wt.size() = n(), (*this[0]) == 0
    int sz = 1;
    while (sz < n()) sz <= 1;
    Poly f(*this, sz);
    wt.isz(sz).irev();
    int k = 1, ksz2 = 2 * sz * k, ksz4 = 4 * sz * k;
    Poly _P(2 * sz, 0), _Q(2 * sz, 0);
    rep(i, sz) _P[i] = wt[i], _Q[i] = (P - f[i]) % P;
    while (sz > 1) {
        Poly R(ksz2);
        rep(i, ksz2) R[i] = (i % 2 == 0 ? _Q[i] : (P - _Q
            [i]) % P);
        Poly PQ = _P.Mul(R), QQ = _Q.Mul(R);
        PQ.isz(ksz4), QQ.isz(ksz4);
        rep(i, ksz2) {
            if ((PQ[ksz2 + i] += _P[i]) >= P) PQ[ksz2 + i]
                -= P;
            if ((QQ[ksz2 + i] += _Q[i] + R[i]) >= P) QQ[ksz2
                + i] -= P;
            if (QQ[ksz2 + i] >= P) QQ[ksz2 + i] -= P;
        }
        fill(ALL(_P), 0), fill(ALL(_Q), 0);
        rep(j, 2 * k) rep(i, sz / 2) {
            _P[sz * j + i] = PQ[(2 * sz) * j + (2 * i + 1)
                ];
            _Q[sz * j + i] = QQ[(2 * sz) * j + (2 * i + 0)
                ];
        }
        sz /= 2, k *= 2;
    }
    vector<ll> p(k);
    rep(i, k) p[i] = _P[2 * i];
    reverse(ALL(p));
    p.resize(m + 1);
    return p;
}
Poly comp_inv() { // (*this)[0] == 0, (*this)[1] != 0
    Poly X(*this, n()), wt(n(), 0);
    ll ic = ntt.minv((*this)[1]);
    for (auto &x: X) x = x * ic % P;
    wt[n() - 1] = 1;
    vector<ll> A = X.power_proj(wt, n() - 1);
    Poly g(n() - 1);
    rep1(i, n() - 1) g[n() - 1 - i] = (n() - 1) * A[i]
        % P * ntt.minv(i) % P;
    g = g.Pow(ntt.minv(P - n() + 1));
    g.insert(g.begin(), 0);
    ll p = 1;
    rep(i, g.n()) g[i] = g[i] * p % P, p = p * ic % P;
    return g;
}
Poly TMul(const Poly &rhs) const { // this[i] - rhs[j]
    ] = k
    return Poly(*this).irev().Mul(rhs).isz(n()).irev();
}
Poly composition(Poly g) { // f(g(x)), 1e5 3s, MAXN
    >= 8n
    auto rec = [&](auto &rec, int n, int k, Poly Q) ->
        Poly {
        if (n == 1) {
            Poly p(2 * k);
            irev();
            fi(0, k) p[2 * i] = (*this)[i];
            return p;
        }
        Poly R(2 * n * k);
        fi(0, 2 * n * k) R[i] = (i % 2 == 0 ? Q[i] : (P -
            Q[i]) % P);
        Poly QQ = Q.Mul(R).isz(4 * n * k);
        fi(0, 2 * n * k) {
            QQ[2 * n * k + i] += Q[i] + R[i];
            QQ[2 * n * k + i] %= P;
        }
        Poly nxt_Q(2 * n * k);
        for(int j = 0; j < 2 * k; j++) fi(0, n / 2) {
            nxt_Q[n * j + i] = QQ[(2 * n) * j + (2 * i + 0)
                ];
        }
        Poly nxt_p = rec(rec, n / 2, k * 2, nxt_Q);
    }

```

```

Poly pq(4 * n * k);
for(int j = 0; j < 2 * k; j++) fi(0, n / 2) {
    pq[(2 * n) * j + (2 * i + 1)] += n * p[n * j + i];
    pq[(2 * n) * j + (2 * i + 1)] %= P;
}
Poly p(2 * n * k);
fi(0, 2 * n * k) p[i] = (p[i] + pq[2 * n * k + i]) % P;
pq.pop_back();
Poly x = pq.TMul(R);
fi(0, 2 * n * k) p[i] = (p[i] + x[i]) % P;
return p;
};
int sz = 1;
while(sz < n() || sz < g.n()) sz <<= 1;
return isz(sz), rec(rec, sz, 1, g.imul(P-1).isz(2 * sz)).isz(sz).irev();
}
};
#undef fi
using Poly_t = Poly<131072 * 2, 998244353, 3>;
//template<> decltype(Poly_t::ntt) Poly_t::ntt = {};

```

7.5 Generating Functions

- Ordinary Generating Function $A(x) = \sum_{i \geq 0} a_i x^i$

- $A(rx) \Rightarrow r^n a_n$
- $A(x) + B(x) \Rightarrow a_n + b_n$
- $A(x)B(x) \Rightarrow \sum_{i=0}^n a_i b_{n-i}$
- $A(x)^k \Rightarrow \sum_{i_1+i_2+\dots+i_k=n} a_{i_1} a_{i_2} \dots a_{i_k}$
- $xA(x)' \Rightarrow na_n$
- $\frac{A(x)}{1-x} \Rightarrow \sum_{i=0}^n a_i$

- Exponential Generating Function $A(x) = \sum_{i \geq 0} \frac{a_i}{i!} x^i$

- $A(x) + B(x) \Rightarrow a_n + b_n$
- $A^{(k)}(x) \Rightarrow a_{n+k}$
- $A(x)B(x) \Rightarrow \sum_{i=0}^n \binom{n}{i} a_i b_{n-i}$
- $A(x)^k \Rightarrow \sum_{i_1+i_2+\dots+i_k=n} \binom{n}{i_1, i_2, \dots, i_k} a_{i_1} a_{i_2} \dots a_{i_k}$
- $xA(x) \Rightarrow na_n$

- Special Generating Function

- $(1+x)^n = \sum_{i \geq 0} \binom{n}{i} x^i$
- $\frac{1}{(1-x)^n} = \sum_{i \geq 0} \binom{n+i-1}{n-1} x^i$

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

- Suppose $A(x)$ is the EGF of a labelled component. Then $B(x) = e^{A(x)}$ is the EGF of the same thing but with any number of components.

- Lagrange's Inversion Formula

$F(G(x)) = G(F(x)) = x$, then $F(0) = G(0) = 0$ and $F'(0), G'(0) \neq 0$. Let H be any FPS, then

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

- Newton's Method

Suppose FPS P satisfies $F(x, P(x)) = 0$ where $F(x, y)$ is some polynomial, then

$$P_{2n}(x) = P_n(x) - F(x, P_n(x)) / \frac{\partial F}{\partial y}(x, P_n(x)).$$

8 Geometry

8.1 Basic [54bb7b]

```

using T = long double; // int
const double eps=1e-9; // 1
T operator*(PT a,PT b){return a.x*b.x+a.y*b.y;}
T operator^(PT a,PT b){return a.x*b.y-a.y*b.x;}
T abs2(PT a){return a*a;}
double abs(PT a){return sqrt(a*a);}
int sign(T a){return abs(a)<eps?0:a>0?1:-1;}

```

```

int ori(PT a,PT b,PT c){return sign((b-a)^(c-a));}
bool btw(PT a,PT b,PT c){ //is C between AB
    return ori(a,b,c)?0:sign((a-c)*(b-c))<=0;}
PT proj(PT a,PT b,PT c){ //ac projection on ab
    return (b-a)*((c-a)*(b-a)/abs2(b-a));}
double dist(PT a,PT b,PT c){ //distance from C to AB
    return abs((c-a)^(b-a))/abs(b-a);}
PT ccw90(PT p){return PT(-p.y, p.x);}
struct Line{PT a,b;}
struct Cir{PT o;double r;}

```

8.2 Convex Hull [da4142]

```

#define sz(x) ((int)x.size())
vector<PT> ConvexHull(vector<PT> pt) {
    int n = sz(pt);
    sort(pt.begin(), pt.end(), [&](PT a, PT b) { return
        make_pair(a.x, a.y) < make_pair(b.x, b.y); });
    vector<PT> ans = {pt[0]};
    rep(t, 2) {
        int m = sz(ans);
        rep1(i, n - 1) {
            while (sz(ans) > m && ori(ans[sz(ans) - 2], ans.
                back(), pt[i]) <= 0) ans.pop_back();
            ans.pb(pt[i]);
        }
        reverse(pt.begin(), pt.end());
    }
    if (sz(ans) > 1) ans.pop_back();
    return ans;
}

```

8.3 Dynamic Convex Hull [24359e]

```

struct DynamicConvexHull {
    struct UpCmp {
        bool operator()(const PT a,const PT b) const {
            if(a.x==b.x) return a.y<b.y;
            return a.x<b.x;
        }
    };
    struct DownCmp {
        bool operator()(const PT a,const PT b) const {
            if (a.x==b.x) return a.y>b.y;
            return a.x>b.x;
        }
    };
    template <typename T>
    struct Hull {
        set<PT,T> hull;
        bool chk(PT i,PT j,PT k){return ((k-i)^(j-i))>0;}
        void insert(PT x) {
            if(inside(x)) return;
            hull.insert(x);
            auto it=hull.lower_bound(x);
            if(next(it)!=hull.end()) {
                for(auto it2=next(it);next(it2)!=hull.end();++
                    it2) {
                    if(chk(x,*it2,*next(it2))) break;
                    hull.erase(it2);
                    it2=hull.lower_bound(x);
                }
            }
            it=hull.lower_bound(x);
            if(it!=hull.begin()) {
                for(auto it2=prev(it);it2!=hull.begin();--it2)
                    {
                    if(chk(*prev(it2),*it2,x)) break;
                    hull.erase(it2);
                    it2=hull.lower_bound(x);
                    if(it2==hull.begin()) break;
                    }
            }
        }
        bool inside(PT x) {
            if(hull.lower_bound(x)!=hull.end()&&hull.
                lower_bound(x)==x)
                return true;
            auto it=hull.lower_bound(x);
            bool ans=false;
            if(it!=hull.begin()&&it!=hull.end()) {
                ans=!chk(*prev(it),x,*it);
            }
        }
    };
}

```



```

    }
    return ans;
}
};
Hull<UpCmp> up;
Hull<DownCmp> down;
void insert(PT x){up.insert(x),down.insert(x);}
bool inside(PT x){return up.inside(x)&&down.inside(x);}
};
};

```

8.4 Point In Convex Hull [9a1f2c]

```

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

```

8.5 Point In Circle [db1576]

```

//is p4 in circumscribed circle of p1p2p3
using i128 = __int128;
ll sq(ll x) { return x * x; }
bool in_cc(const PT &p1, const PT &p2, const PT &p3,
    const PT &p4) {
    ll u11 = p1.x - p4.x; ll u12 = p1.y - p4.y;
    ll u21 = p2.x - p4.x; ll u22 = p2.y - p4.y;
    ll u31 = p3.x - p4.x; ll u32 = p3.y - p4.y;
    ll u13 = sq(p1.x) - sq(p4.x) + sq(p1.y) - sq(p4.y);
    ll u23 = sq(p2.x) - sq(p4.x) + sq(p2.y) - sq(p4.y);
    ll u33 = sq(p3.x) - sq(p4.x) + sq(p3.y) - sq(p4.y);
    i128 det =
        - (i128)u13 * u22 * u31 + (i128)u12 * u23 * u31
        + (i128)u13 * u21 * u32 - (i128)u11 * u23 * u32
        - (i128)u12 * u21 * u33 + (i128)u11 * u22 * u33;
    return det > 0;
}

```

8.6 Half Plane Intersection [b9b059]

```

// depends on polar angle
auto area_pair(Line a, Line b) {
    return make_pair((a.b - a.a) ^ (b.a - a.a),
        (a.b - a.a) ^ (b.b - a.a));
}
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) a12X *= -1, a12Y *= -1;
    return (__int128) a02Y * a12X > (__int128) a02X *
        a12Y;
}
/* [solution exists] <=> [result.size() > 2] */
/* --- Line.a --- Line.b --- */
#define sz(x) ((int)x.size())
vector<Line> HalfPlaneInter(vector<Line> arr) {
    sort(arr.begin(), arr.end(), [&](Line a, Line b) {
        PT p1 = a.b - a.a, p2 = b.b - b.a;
        if ((p1 <= p2) != 0) return p1 < p2;
        return ori(a.a, a.b, b.b) < 0;
    });
    deque<Line> dq(1, arr[0]);
    auto pop_back = [&](int t, Line p) {
        while (sz(dq) >= t && !isin(p, dq[sz(dq) - 2], dq.
            back()))
            dq.pop_back();
    };
    auto pop_front = [&](int t, Line p) {
        while (sz(dq) >= t && !isin(p, dq[0], dq[1]))
            dq.pop_front();
    };
}

```

```

for (auto p : arr)
    if (((dq.back().b - dq.back().a) <= (p.b - p.a))
        != 0)
        pop_back(2, p), pop_front(2, p), dq.pb(p);
pop_back(3, dq[0]), pop_front(3, dq.back());
return vector<Line>(dq.begin(), dq.end());
}

```

8.7 Minkowski Sum [65bf29]

```

vector<Pt> Minkowski(vector<Pt> a, vector<Pt> b) {
    a = ConvexHull(a), b = ConvexHull(b);
    int n = a.size(), m = b.size();
    vector<Pt> c = {a[0] + b[0]}, s1, s2;
    rep(i, n) s1.pb(a[(i + 1) % n] - a[i]);
    rep(i, m) s2.pb(b[(i + 1) % m] - b[i]);
    for(int p1 = 0, p2 = 0; p1 < n || p2 < m;)
        if (p2 == m || (p1 < n && sign(s1[p1] ^ s2[p2]) >=
            0)) c.pb(c.back() + s1[p1++]);
        else c.pb(c.back() + s2[p2++]);
    return ConvexHull(c);
}

```

8.8 Polar Angle [9d4843]

```

// CCW starting from (1, 0) inclusive, w/o tie-breaking
int halfplane(PT p) {
    if (sign(p * p) == 0) return 0;
    return 1 - 2 * (sign(p.y) > 0 || (sign(p.y) == 0 &&
        sign(p.x) > 0));
} // upper(-1) -> origin(0) -> lower(1)
auto operator<=>(PT a, PT b) {
    int ha = halfplane(a), hb = halfplane(b);
    if (ha != hb) return ha <= hb;
    return 0 <= sign(a ^ b);
} // before c++20: replace <=> with <

```

8.9 Rotating Sweep Line [bdd83b]

```

// pts: 0-indexed Pt array
void RotSwpLine(int n, PT* pts) {
    using E = pair<PT, pii>;
    vector<E> ev; // dir, i, j: (i, j)=>(j, i)
    rep(i, n) rep(j, i) {
        PT dir = pts[j] - pts[i];
        halfplane(dir) < 0 ? ev.pb({dir, {i, j}})
            : ev.pb({PT(0, 0) - dir, {j, i}});
    }
    sort(ev.begin(), ev.end(), [&](E e1, E e2) {
        auto pol = (e1.F <= e2.F);
        return pol < 0 || (pol == 0 &&
            p11(e1.F * pts[e1.S.F], e1.F * pts[e1.S.S])
            < p11(e1.F * pts[e2.S.F], e1.F * pts[e2.S.S]));
    });
    vector<int> ord(n), rk(n);
    iota(ord.begin(), ord.end(), 0);
    sort(ord.begin(), ord.end(), [&](int i, int j) {
        return make_pair(pts[i].y, pts[i].x)
            < make_pair(pts[j].y, pts[j].x);
    });
    rep(i, n) rk[ord[i]] = i;
    init_order(ord); // ord[*]: point indices
    int ne = (int)ev.size();
    rep(ie, ne) {
        int i, j; tie(i, j) = ev[ie].S;
        update_swap(i, j); // i, j: point indices
        rk[i]++; rk[j]--; // rk[point idx]: rank in ord
        tie(ord[rk[i]], ord[rk[i] - 1]) = tie(i, j);
        if(ie == ne - 1 || (ev[ie + 1].F <= ev[ie].F) !=
            0) update_ans();
    }
}

```

8.10 Segment Intersect [3c8feb]

```

bool seg_sect(PT p1, PT p2, PT p3, PT p4) {
    int a123 = ori(p1, p2, p3);
    int a124 = ori(p1, p2, p4);
    int a341 = ori(p3, p4, p1);
    int a342 = ori(p3, p4, p2);
    if(!a123 && !a124) return btw(p1, p2, p3) || btw(p1,
        p2, p4) || btw(p3, p4, p1) || btw(p3, p4, p2);
}

```



```

return a123 * a124 <= 0 && a341 * a342 <= 0;
} // does p1p2 intersect p3p4
PT intersect(PT p1, PT p2, PT p3, PT p4) {
double a123 = (p2 - p1) ^ (p3 - p1);
double a124 = (p2 - p1) ^ (p4 - p1);
return (p4 * a123 - p3 * a124) / (a123 - a124);
} // C^3 / C^2

```

8.11 Circle Intersect With Any [c326f7]

```

vector<Pt> CircleLineInter(Cir c, Line l) { //cir-line
Pt p = 1.a + (1.b - 1.a) * ((c.o - 1.a) * (1.b - 1.a)
) / abs2(1.b - 1.a);
double s = (1.b - 1.a) ^ (c.o - 1.a), h2 = c.r * c.r
- s * s / abs2(1.b - 1.a);
if (sign(h2) == -1) return {};
if (sign(h2) == 0) return {p};
Pt h = (1.b - 1.a) / abs(1.b - 1.a) * sqrt(h2);
return {p - h, p + h};
}
vector<Pt> CirclesInter(Cir c1, Cir c2) { //cir-cir
double d2 = abs2(c1.o - c2.o), d = sqrt(d2);
if (d < max(c1.r, c2.r) - min(c1.r, c2.r) || d > c1.r
+ c2.r) return {};
Pt u = (c1.o + c2.o) / 2 + (c1.o - c2.o) * ((c2.r *
c2.r - c1.r * c1.r) / (2 * d2));
double A = sqrt((c1.r + c2.r + d) * (c1.r - c2.r + d)
* (c1.r + c2.r - d) * (-c1.r + c2.r + d));
Pt v = Pt(c1.o.Y - c2.o.Y, -c1.o.X + c2.o.X) * A / (2
* d2);
if (sign(v.X) == 0 && sign(v.Y) == 0) return {u};
return {u + v, u - v};
}
double _area(Pt pa, Pt pb, double r) { //for poly-cir
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 = pb * (pb - pa) / a / c, B = acos(cosB);
double cosC = (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 area_poly_circle(vector<Pt> poly, Pt O, double r
) { //poly-cir
double S = 0; int n = poly.size();
for (int i = 0; i < n; ++i)
S += _area(poly[i] - O, poly[(i + 1) % n] - O, r) *
ori(O, poly[i], poly[(i + 1) % n]);
return fabs(S);
}

```

8.12 Tangents [44761a]

```

PT unit(PT p) { return p / abs(p); }
PT ccw(PT p, double t) {
return PT(
p.x * cos(t) - p.y * sin(t),
p.x * sin(t) + p.y * cos(t)
);
}
vector<Line> tangent(Cir c, PT p) {
vector<Line> z;
double d = abs(p - c.o);
if (sign(d - c.r) == 0) {
z.push_back({p, p + ccw90(p - c.o)});
} else if (d > c.r) {
double o = acos(c.r / d);
PT i = unit(p - c.o), j = ccw(i, o) * c.r, k = ccw(
i, -o) * c.r;
z.push_back({c.o + j, p});
z.push_back({c.o + k, p});
}
}

```

```

return z;
}
vector<Line> tangent(Cir c1, Cir c2, int sign1) {
// sign1 = 1 for outer tang, -1 for inter tang
vector<Line> ret;
double d_sq = abs2(c1.o - c2.o);
if (sign(d_sq) == 0) return ret;
double d = sqrt(d_sq);
PT v = (c2.o - c1.o) / d;
double c = (c1.r - sign1 * c2.r) / d;
if (c * c > 1) return ret;
double h = sqrt(max(0.0, 1.0 - c * c));
for (int sign2 = 1; sign2 >= -1; sign2 -= 2) {
PT n = PT(v.X * c - sign2 * h * v.Y, v.Y * c +
sign2 * h * v.X);
PT p1 = c1.o + n * c1.r;
PT p2 = c2.o + n * (c2.r * sign1);
if (sign(p1.X - p2.X) == 0 && sign(p1.Y - p2.Y) ==
0)
p2 = p1 + ccw90(c2.o - c1.o);
ret.pb({p1, p2});
}
return ret;
}

```

8.13 Tangent to Convex Hull [e374ab]

```

/* The point should be strictly out of hull
return arbitrary point on the tangent line */
pii get_tangent(vector<PT> &C, PT p) {
auto gao = [&](int s) {
return cyc_tsearch(C.size(), [&](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

```

8.14 Minimum Enclosing Circle [4a08af]

```

PT circenter(PT a, PT b, PT c) {
PT ab = (a + b) / 2, ac = (a + c) / 2;
return intersect(
ab, ab + ccw90(b - a),
ac, ac + ccw90(c - a)
);
}
Cir min_enclosing(vector<PT> &p) {
shuffle(p.begin(), p.end(), mt19937(clock()));
double r = 0;
PT cent = p[0];
rep(i, p.size()) if (abs2(cent - p[i]) > r) {
cent = p[i]; r = 0;
rep(j, i) if (abs2(cent - p[j]) > r) {
cent = (p[i] + p[j]) / 2;
r = abs2(p[j] - cent);
rep(k, j) if (abs2(cent - p[k]) > r) {
cent = circenter(p[i], p[j], p[k]);
r = abs2(p[k] - cent);
}
}
}
return {cent, sqrt(r)};
}

```

8.15 Union of Stuff [234a81]

```

//Union of Circles
vector<pair<double, double>> CoverSegment(Cir a, Cir b)
{
double d = abs(a.o - b.o);
vector<pair<double, double>> res;
if (sign(a.r + b.r - d) == 0);
else if (d <= abs(a.r - b.r) + eps) {
if (a.r < b.r) res.emplace_back(0, 2 * pi);
} else if (d < abs(a.r + b.r) - eps) {
double o = acos((a.r * a.r + d * d - b.r * b.r) /
(2 * a.r * d)), z = atan2((b.o - a.o).Y, (b.o -
a.o).X);
if (z < 0) z += 2 * pi;
double l = z - o, r = z + o;
if (l < 0) l += 2 * pi;
if (r > 2 * pi) r -= 2 * pi;
}
}

```

```

    if (1 > r) res.emplace_back(1, 2 * pi), res.
        emplace_back(0, r);
    else res.emplace_back(1, r);
}
return res;
}
double CircleUnionArea(vector<Cir> c) { // circle
    should be identical
    int n = c.size();
    double a = 0, w;
    for (int i = 0; w = 0, i < n; ++i) {
        vector<pair<double, double>> s = {{2 * pi, 9}}, z;
        for (int j = 0; j < n; ++j) if (i != j) {
            z = CoverSegment(c[i], c[j]);
            for (auto &e : z) s.push_back(e);
        }
        sort(s.begin(), s.end());
        auto F = [&](double t) { return c[i].r * (c[i].r *
            t + c[i].o.X * sin(t) - c[i].o.Y * cos(t)); };
        for (auto &e : s) {
            if (e.first > w) a += F(e.first) - F(w);
            w = max(w, e.second);
        }
    }
    return a * 0.5;
}
// Union of Polygons
double polyUnion(vector<vector<Pt>> poly) {
    int n = poly.size();
    double ans = 0;
    auto solve = [&](Pt a, Pt b, int cid) {
        vector<pair<Pt, int>> event;
        for (int i = 0; i < n; ++i) {
            int st = 0, sz = poly[i].size();
            while (st < sz && ori(poly[i][st], a, b) != 1) st
                ++;
            if (st == sz) continue;
            for (int j = 0; j < sz; ++j) {
                Pt c = poly[i][(j + st) % sz], d = poly[i][(j +
                    st + 1) % sz];
                if (sign((a - b) ^ (c - d)) != 0) {
                    int ok1 = ori(c, a, b) == 1, ok2 = ori(d, a,
                        b) == 1;
                    if (ok1 ^ ok2) event.emplace_back(intersect(a
                        , b, c, d), ok1 ? 1 : -1);
                } else if (ori(c, a, b) == 0 && sign((a - b) *
                    (c - d)) > 0 && i <= cid) {
                    event.emplace_back(c, -1);
                    event.emplace_back(d, 1);
                }
            }
        }
        sort(all(event), [&](pair<Pt, int> i, pair<Pt,
            int> j) {
            return ((a - i.first) * (a - b)) < ((a - j.first)
                * (a - b));
        });
        int now = 0;
        Pt lst = a;
        for (auto [x, y] : event) {
            if (btw(a, b, lst) && btw(a, b, x) && !now) ans
                += lst ^ x;
            now += y, lst = x;
        }
    };
    for (int i = 0; i < n; ++i) for (int j = 0; j < poly[
        i].size(); ++j) {
        solve(poly[i][j], poly[i][(j + 1) % int(poly[i].
            size())], i);
    }
    return ans / 2;
}

```

8.16 Delaunay Triangulation [982e64]

```

/* Delaunay Triangulation:
Given a sets of points on 2D plane, find a
triangulation such that no points will strictly
inside circumcircle of any triangle. */
struct Edge {
    int id; // oidx[id]
    list<Edge>::iterator twin;

```

```

    Edge(int _id = 0):id(_id) {}
};
struct Delaunay { // 0-base
    int n, oidx[N];
    list<Edge> head[N]; // result udir. graph
    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;

```

8.17 Voronoi Diagram [da0c5e]

```

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

```

8.18 3D Basic [440428]

```

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-(const Point &p1, const Point &p2)
{return Point(p1.x - p2.x, p1.y - p2.y, p1.z - p2.z);}
Point operator+(const Point &p1, const Point &p2)
{return Point(p1.x + p2.x, p1.y + p2.y, p1.z + p2.z);}
Point operator/(const Point &p1, const double &v)
{ return Point(p1.x / v, p1.y / v, p1.z / v); }
Point cross(const Point &p1, const 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(const Point &p1, const Point &p2)
{ return p1.x * p2.x + p1.y * p2.y + p1.z * p2.z; }
double abs(const Point &a)
{ return sqrt(dot(a, a)); }
Point cross3(const Point &a, const Point &b, const
  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); }
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));
}

```

8.19 3D Convex Hull [875f37]

```

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(ALL(P), [&](auto p) { return sign
    (abs2(P[0] - p)) != 0; }));
  swap(P[2], *find_if(ALL(P), [&](auto p) { return sign
    (abs2(cross3(p, P[0], P[1])) != 0; }));
  swap(P[3], *find_if(ALL(P), [&](auto p) { return sign
    (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 = sign(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 (sign(volume(P[s.a], P[s.b], P[s.c], P[t.a])) !=
    0) return 0;
  if (sign(volume(P[s.a], P[s.b], P[s.c], P[t.b])) !=
    0) return 0;

```

```

  if (sign(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);
}
};

```

9 Misc

9.1 Binary Search On Fraction [765c5a]

```

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

```

9.2 Cyclic Ternary Search [9017cc]

```

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

```

9.3 Matroid Intersection

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

- If $S \in \mathcal{I}$ and $S' \subsetneq S$, then $S' \in \mathcal{I}$.

- For $S_1, S_2 \in \mathcal{I}$ s.t. $|S_1| < |S_2|$, there exists $e \in S_2 \setminus S_1$ s.t. $S_1 \cup \{e\} \in \mathcal{I}$.

Matroid intersection:

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

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

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

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

Find a shortest path (with BFS) starting from a vertex in Y_1 and ending at a vertex in Y_2 which doesn't pass through any other vertices in Y_2 , and alternate the path. The size of S will be incremented by 1 in each iteration. For the weighted case, assign weight $w(x)$ to vertex x if $x \in S$ and $-w(x)$ if $x \notin S$. Find the path with the minimum number of edges among all minimum length paths and alternate it.

9.4 Min Plus Convolution [09b5c3]

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

9.5 Mo's Algorithm [ea5261]

```
struct MoAlgorithm {
    struct query {
        int l, r, id;
        bool operator < (const query &o) {
            if (l / C == o.l / C)
                return (l / C) & 1 ? r > o.r : r < o.r;
            return l / C < o.l / C;
        }
    };
    int cur_ans;
    vector<int> ans;
    void add(int x) {} // do something
    void sub(int x) {} // do something
    vector<query> Q;
    void add_query(int l, int r, int id) { // [l, r)
        Q.push_back({l, r, id});
        ans.push_back(0);
    }
    void run() {
        sort(Q.begin(), Q.end());
        int pl = 0, pr = 0;
        cur_ans = 0;
        for (query &i : Q) {
            while (pl > i.l) add(a[--pl]);
            while (pr < i.r) add(a[pr++]);
            while (pl < i.l) sub(a[pl++]);
            while (pr > i.r) sub(a[pr--]);
            ans[i.id] = cur_ans;
        }
    }
};
```

9.6 Mo's Algorithm On Tree [8331c2]

```
/* Mo's Algorithm On Tree
Preprocess:
1) LCA
2) dfs with in[u] = dft++, out[u] = dft++
3) ord[in[u]] = ord[out[u]] = u
4) bitset<MAXN> inset
*/
```

```
struct Query {
    int L, R, LBid, lca;
    Query(int u, int v) {
        int c = LCA(u, v);
        if (c == u || c == v)
            q.lca = -1, q.L = out[c ^ u ^ v], q.R = out[c];
        else if (out[u] < in[v])
            q.lca = c, q.L = out[u], q.R = in[v];
        else
            q.lca = c, q.L = out[v], q.R = in[u];
        q.Lid = q.L / blk;
    }
    bool operator<(const Query &q) const {
        if (LBid != q.LBid) return LBid < q.LBid;
        return R < q.R;
    }
};
void flip(int x) {
    if (inset[x]) sub(arr[x]); // TODO
    else add(arr[x]); // TODO
    inset[x] = ~inset[x];
}
void solve(vector<Query> query) {
    sort(ALL(query));
    int L = 0, R = 0;
    for (auto q : query) {
        while (R < q.R) flip(ord[++R]);
        while (L > q.L) flip(ord[--L]);
        while (R > q.R) flip(ord[R--]);
        while (L < q.L) flip(ord[L++]);
        if (~q.lca) add(arr[q.lca]);
        // answer query
        if (~q.lca) sub(arr[q.lca]);
    }
}
```

9.7 PBDS [d65996]

```
#include <ext/pb_ds/tree_policy.hpp>
#include <ext/pb_ds/assoc_container.hpp>
using namespace __gnu_pbds;
tree<int, null_type, less<int>, rb_tree_tag,
    tree_order_statistics_node_update> oset;
// order_of_key, find_by_order
cc_hash_table<int, int> m1;
gp_hash_table<int, int> m2;
// like map, but much faster
```

9.8 Simulated Annealing [de78c6]

```
double factor = 100000;
const int base = 1e9; // remember to run ~ 10 times
for (int it = 1; it <= 100000; ++it) {
    // ans: answer, nw: current value, rnd(): mt19937
    if (exp(-(nw - ans) / factor) >= (double)rnd() %
        base) / base) ans = nw;
    factor *= 0.99995;
}
```

9.9 SOS dp [6aad1]

```
//memory optimized, super easy to code.
rep(i, (1 << N)) F[i] = A[i];
rep(i, N) rep(mask, (1 << N)) {
    if (mask & (1<<i)) F[mask] += F[mask^(1<<i)];
}
}
```

9.10 SMAWK [a2a4ce]

```
// For all 2x2 submatrix:
// If M[1][0] < M[1][1], M[0][0] < M[0][1]
// If M[1][0] == M[1][1], M[0][0] <= M[0][1]
// M[i][ans_i] is the best value in the i-th row
// select(int r, int u, int v) return true if f(r, v)
// is better than f(r, u)
vector<int> smawk(int N, int M, auto &&select) {
    auto dc = [&](auto self, const vector<int> &r, const
        vector<int> &c) {
        if (r.empty()) return vector<int>{};
        const int n = SZ(r); vector<int> ans(n), nr, nc;
        for (int i : c) {
            while (!nc.empty() &&
```

```

        select(r[nc.size() - 1], nc.back(), i))
    nc.pop_back();
    if (int(nc.size()) < n) nc.push_back(i);
}
for (int i = 1; i < n; i += 2) nr.push_back(r[i]);
const auto na = self(self, nr, nc);
for (int i = 1; i < n; i += 2) ans[i] = na[i >> 1];
for (int i = 0, j = 0; i < n; i += 2) {
    ans[i] = nc[j];
    const int end = i + 1 == n ? nc.back() : ans[i +
        1];
    while (nc[j] != end)
        if (select(r[i], ans[i], nc[++j])) ans[i] = nc[
            j];
}
return ans;
};
vector<int> R(N), C(M); iota(iter(R), 0), iota(iter(C
    ), 0);
return dc(dc, R, C);
}

```

9.11 Tree Hash [34aae5]

```

ull seed;
ull shift(ull x) {
    x ^= x << 13;
    x ^= x >> 7;
    x ^= x << 17;
    return x;
}
ull dfs(int u, int f) {
    ull sum = seed;
    for (int i : G[u]) if (i != f)
        sum += shift(dfs(i, u));
    return sum;
}

```

9.12 Python [ebfb5e]

```

from [decimal, fractions, math, random] import *
setcontext(Context(prec=10, Emax=MAX_EMAX, rounding=
    ROUND_FLOOR))
Decimal('1.1') / Decimal('0.2')
Fraction(3, 7)
Fraction(Decimal('1.14'))
Fraction('1.2').limit_denominator(4).numerator
Fraction(cos(pi / 3)).limit_denominator()
print(*[randint(1, C) for i in range(0, N)], sep=' ')

```