

Contents

1	Basic	1	6.23	Triangle Center	23
1.1	vimrc	1	7	Stringology	23
1.2	default	1	7.1	KMP	23
1.3	optimize	1	7.2	Z-algorithm	23
1.4	judge	2	7.3	Manacher	23
1.5	Random	2	7.4	SuffixArray Simple	23
1.6	Increase stack size	2	7.5	SuffixArray SAIS C++20	23
2	Matching and Flow	2	7.6	Aho-Corasick	24
2.1	Dinic	2	7.7	Palindromic Tree	24
2.2	MCMF	2	7.8	Suffix Automaton	24
2.3	HopcroftKarp	2	7.9	Lyndon Factorization	25
2.4	KM	3	7.10	SmallestRotation	25
2.5	SW	3	8	Misc	25
2.6	GeneralMatching	3	8.1	Fraction Binary Search	25
3	Graph	4	8.2	de Bruijn sequence	25
3.1	Strongly Connected Component	4	8.3	HilbertCurve	25
3.2	2-SAT	4	8.4	Grid Intersection	26
3.3	Tree	4	8.5	NextPerm	26
3.4	Functional Graph	5	8.6	Python FastIO	26
3.5	Manhattan MST	5	8.7	HeapSize	26
3.6	Count Cycles	5	1	Basic	
3.7	Maximum Clique	6	1.1	vimrc	
3.8	Min Mean Weight Cycle	6		set ts=4 sw=4 nu rnu et hls mouse=a	
3.9	Block Cut Tree	6		filetype indent on	
3.10	Dominator Tree	6		sy on	
3.11	Matroid Intersection	7		imap jk <Esc>	
3.12	Generalized Series-Parallel Graph	7		imap {<CR> {<CR>}<C-o>0	
4	Data Structure	7		nmap J 5j	
4.1	Lazy Segtree	7		nmap K 5k	
4.2	Fenwick Tree	8		nmap <F1> :w<bar>!g++ '%' -o run -std=c++20 -DLOCAL -	
4.3	Interval Segtree	8		Wfatal-errors -fsanitize=address,undefined -g &&	
4.4	PrefixMax Sum Segtree	9		echo done. && time ./run<CR>	
4.5	Disjoint Set Union-undo	9	1.2	default	
4.6	PBDS	9		#include <bits/stdc++.h>	
4.7	Centroid Decomposition	9		using namespace std;	
4.8	2D BIT	10		template<class F, class S>	
4.9	Big Binary	10		ostream &operator<<(ostream &s, const pair<F, S> &v) {	
4.10	Big Integer	10		return s << "(" << v.first << ", " << v.second << ")"	
4.11	Splay Tree	11		};	
4.12	Link Cut Tree	12		template<ranges::range T> requires (!is_convertible_v<T	
4.13	Static Top Tree	12		, string_view>)	
5	Math	13		istream &operator>>(istream &s, T &&v) {	
5.1	Theorem	13		for (auto &&x : v) s >> x;	
5.2	Linear Sieve	14		return s;	
5.3	Exgcd	14		template<ranges::range T> requires (!is_convertible_v<T	
5.4	Chinese Remainder Theorem	14		, string_view>)	
5.5	Factorize	14		istream &operator>>(istream &s, T &&v) {	
5.6	FloorBlock	15		for (auto &&x : v) s >> x;	
5.7	FloorCeil	15		return s;	
5.8	NTT Prime List	15		template<ranges::range T> requires (!is_convertible_v<T	
5.9	NTT	15		, string_view>)	
5.10	FWT	16		ostream &operator<<(ostream &s, T &&v) {	
5.11	FWT	16		for (auto &&x : v) s << x << ' ';	
5.12	Xor Basis	16		return s;	
5.13	Lucas	16		template<class... T> void dbg(T... x) {	
5.14	Min25 Sieve	16		char e{};	
5.15	Berlekamp Massey	17		((cerr << e << x, e = ' '), ...);	
5.16	Gauss Elimination	17		};	
5.17	Linear Equation	17		#define debug(x...) dbg(#x, '=', x, '\n')	
5.18	LinearRec	17		#else	
5.19	SubsetConv	17		#define debug(...) ((void)0)	
5.20	SqrtMod	18		#endif	
5.21	DiscreteLog	18		#define all(v) (v).begin(), (v).end()	
5.22	FloorSum	18		#define rall(v) (v).rbegin(), (v).rend()	
5.23	Linear Programming Simplex	18		#define ff first	
5.24	Lagrange Interpolation	19		#define ss second	
6	Geometry	19		template<class T> inline constexpr T inf =	
6.1	Point	19		numeric_limits<T>::max() / 2;	
6.2	Line	19		bool chmin(auto &a, auto b) { return (b < a) and (a = b	
6.3	Circle	19		, true); }	
6.4	Point to Segment Distance	19		bool chmax(auto &a, auto b) { return (a < b) and (a = b	
6.5	Point in Polygon	19		, true); }	
6.6	Intersection of Lines	19		using u32 = unsigned int;	
6.7	Intersection of Circle and Line	19		using i64 = long long;	
6.8	Intersection of Circles	19		using u64 = unsigned long long;	
6.9	Area of Circle and Polygon	20		using i128 = __int128;	
6.10	Area of Sector	20		1.3	optimize
6.11	Union of Polygons	20			#pragma GCC optimize("O3,unroll-loops")
6.12	Union of Circles	20			#pragma GCC target("avx2,bmi,bmi2,lzcnt,popcnt")
6.13	TangentLines of Circle and Point	21			
6.14	TangentLines of Circles	21			
6.15	Convex Hull	21			
6.16	Convex Hull trick	21			
6.17	Dynamic Convex Hull	21			
6.18	Half Plane Intersection	22			
6.19	Minkowski	22			
6.20	Minimal Enclosing Circle	22			
6.21	Point In Circumcircle	22			
6.22	Delaunay Triangulation	22			

1.4 judge

```
set -e
# g++ -O3 -DLOCAL -fsanitize=address,undefined -std=c
++20 A.cpp -o a
g++ -O3 -DLOCAL -std=c++20 A.cpp -o a
g++ -O3 -DLOCAL -std=c++20 ac.cpp -o c

for ((i = 0; ; i++)); do
    echo "case $i"
    python3 gen.py > inp
    time ./a < inp > wa.out
    time ./c < inp > ac.out
    diff ac.out wa.out || break
done
```

1.5 Random

```
mt19937 rng(random_device{}());
i64 rand(i64 l = -lim, i64 r = lim) {
    return uniform_int_distribution<i64>(l, r)(rng);
}
double randr(double l, double r) {
    return uniform_real_distribution<double>(l, r)(rng);
}
```

1.6 Increase stack size

```
ulimit -s
```

2 Matching and Flow

2.1 Dinic

```
template<class Cap>
struct Flow {
    struct Edge { int v; Cap w; int rev; };
    vector<vector<Edge>> G;
    int n;
    Flow(int n) : n(n), G(n) {}
    void addEdge(int u, int v, Cap w) {
        G[u].push_back({v, w, (int)G[v].size()});
        G[v].push_back({u, 0, (int)G[u].size() - 1});
    }
    vector<int> dep;
    bool bfs(int s, int t) {
        dep.assign(n, 0);
        dep[s] = 1;
        queue<int> que;
        que.push(s);
        while (!que.empty()) {
            int u = que.front(); que.pop();
            for (auto [v, w, _] : G[u])
                if (!dep[v] and w) {
                    dep[v] = dep[u] + 1;
                    que.push(v);
                }
        }
        return dep[t] != 0;
    }
    Cap dfs(int u, Cap in, int t) {
        if (u == t) return in;
        Cap out = 0;
        for (auto &[v, w, rev] : G[u]) {
            if (w and dep[v] == dep[u] + 1) {
                Cap f = dfs(v, min(w, in), t);
                w -= f;
                G[v][rev].w += f;
                in -= f;
                out += f;
                if (!in) break;
            }
        }
        if (in) dep[u] = 0;
        return out;
    }
    Cap maxFlow(int s, int t) {
        Cap ret = 0;
        while (bfs(s, t)) {
            ret += dfs(s, inf<Cap>, t);
        }
        return ret;
    }
};
```

2.2 MCMF

```
template<class T>
struct MCMF {
    struct Edge { int v; T f, w; int rev; };
    vector<vector<Edge>> G;
    const int n;
    MCMF(int n) : n(n), G(n) {}
    void addEdge(int u, int v, T f, T c) {
        G[u].push_back({v, f, c, ssize(G[v])});
        G[v].push_back({u, 0, -c, ssize(G[u]) - 1});
    }
    vector<T> dis;
    vector<bool> vis;
    bool spfa(int s, int t) {
        queue<int> que;
        dis.assign(n, inf<T>);
        vis.assign(n, false);
        que.push(s);
        vis[s] = 1;
        dis[s] = 0;
        while (!que.empty()) {
            int u = que.front(); que.pop();
            vis[u] = 0;
            for (auto [v, f, w, _] : G[u])
                if (f and chmin(dis[v], dis[u] + w))
                    if (!vis[v]) {
                        que.push(v);
                        vis[v] = 1;
                    }
        }
        return dis[t] != inf<T>;
    }
    T dfs(int u, T in, int t) {
        if (u == t) return in;
        vis[u] = 1;
        T out = 0;
        for (auto &[v, f, w, rev] : G[u])
            if (f and !vis[v] and dis[v] == dis[u] + w) {
                T x = dfs(v, min(in, f), t);
                in -= x;
                out += x;
                f -= x;
                G[v][rev].f += x;
                if (!in) break;
            }
        if (in) dis[u] = inf<T>;
        vis[u] = 0;
        return out;
    }
    pair<T, T> maxFlow(int s, int t) {
        T a = 0, b = 0;
        while (spfa(s, t)) {
            T x = dfs(s, inf<T>, t);
            a += x;
            b += x * dis[t];
        }
        return {a, b};
    }
};
```

2.3 HopcroftKarp

```
// Complexity:  $O(m \sqrt{n})$ 
// edge (u \in A) -> (v \in B) : G[u].push_back(v);
struct HK {
    const int n, m;
    vector<int> l, r, a, p;
    int ans;
    HK(int n, int m) : n(n), m(m), l(n, -1), r(m, -1),
        ans{} {}
    void work(const auto &G) {
        for (bool match = true; match; ) {
            match = false;
            queue<int> q;
            a.assign(n, -1), p.assign(n, -1);
            for (int i = 0; i < n; i++)
                if (l[i] == -1) q.push(a[i] = p[i] = i);
            while (!q.empty()) {
                int z, x = q.front(); q.pop();
                if (l[a[x]] != -1) continue;
                for (int y : G[x]) {
                    if (r[y] == -1) {
```

```

        for (z = y; z != -1; ) {
            r[z] = x;
            swap(l[x], z);
            x = p[x];
        }
        match = true;
        ans++;
        break;
    } else if (p[r[y]] == -1) {
        q.push(z = r[y]);
        p[z] = x;
        a[z] = a[x];
    }
}
}
}
};

```

2.4 KM

```

// max weight, for min negate the weights
template<class T>
T KM(const vector<vector<T>> &w) {
    const int n = w.size();
    vector<T> lx(n), ly(n);
    vector<int> mx(n, -1), my(n, -1), pa(n);
    auto augment = [&](int y) {
        for (int x, z; y != -1; y = z) {
            x = pa[y];
            z = mx[x];
            my[y] = x;
            mx[x] = y;
        }
    };
    auto bfs = [&](int s) {
        vector<T> sy(n, inf<T>);
        vector<bool> vx(n), vy(n);
        queue<int> q;
        q.push(s);
        while (true) {
            while (q.size()) {
                int x = q.front();
                q.pop();
                vx[x] = 1;
                for (int y = 0; y < n; y++) {
                    if (vy[y]) continue;
                    T d = lx[x] + ly[y] - w[x][y];
                    if (d == 0) {
                        pa[y] = x;
                        if (my[y] == -1) {
                            augment(y);
                            return;
                        }
                    }
                    vy[y] = 1;
                    q.push(my[y]);
                }
            } else if (chmin(sy[y], d)) {
                pa[y] = x;
            }
        }
    };
    T cut = inf<T>;
    for (int y = 0; y < n; y++)
        if (!vy[y])
            chmin(cut, sy[y]);
    for (int j = 0; j < n; j++) {
        if (vx[j]) lx[j] -= cut;
        if (vy[j]) ly[j] += cut;
        else sy[j] -= cut;
    }
    for (int y = 0; y < n; y++)
        if (!vy[y] and sy[y] == 0) {
            if (my[y] == -1) {
                augment(y);
                return;
            }
            vy[y] = 1;
            q.push(my[y]);
        }
    };
    for (int x = 0; x < n; x++)
        lx[x] = ranges::max(w[x]);
}

```

```

for (int x = 0; x < n; x++)
    bfs(x);
T ans = 0;
for (int x = 0; x < n; x++)
    ans += w[x][mx[x]];
return ans;
}

```

2.5 SW

```

int w[kN][kN], g[kN], del[kN], v[kN];
void AddEdge(int x, int y, int c) {
    w[x][y] += c;
    w[y][x] += c;
}
pair<int, int> Phase(int n) {
    fill(v, v + n, 0), fill(g, g + n, 0);
    int s = -1, t = -1;
    while (true) {
        int c = -1;
        for (int i = 0; i < n; ++i) {
            if (del[i] || v[i]) continue;
            if (c == -1 || g[i] > g[c]) c = i;
        }
        if (c == -1) break;
        v[c] = 1, s = t, t = c;
        for (int i = 0; i < n; ++i) {
            if (del[i] || v[i]) continue;
            g[i] += w[c][i];
        }
    }
    return make_pair(s, t);
}
int GlobalMinCut(int n) {
    int cut = kInf;
    fill(del, 0, sizeof(del));
    for (int i = 0; i < n - 1; ++i) {
        int s, t; tie(s, t) = Phase(n);
        del[t] = 1, cut = min(cut, g[t]);
        for (int j = 0; j < n; ++j) {
            w[s][j] += w[t][j];
            w[j][s] += w[j][t];
        }
    }
    return cut;
}

```

2.6 GeneralMatching

```

struct GeneralMatching { // n <= 500
    const int BLOCK = 10;
    int n;
    vector<vector<int>> > g;
    vector<int> hit, mat;
    std::priority_queue<pair<i64, int>, vector<pair<i64, int>>, greater<pair<i64, int>>> unmat;
    GeneralMatching(int _n) : n(_n), g(_n), mat(n, -1), hit(n) {}
    void add_edge(int a, int b) { // 0 <= a != b < n
        g[a].push_back(b);
        g[b].push_back(a);
    }
    int get_match() {
        for (int i = 0; i < n; i++) if (!g[i].empty()) {
            unmat.emplace(0, i);
        }
        // If WA, increase this
        // there are some cases that need >= 1.3 * n^2 steps
        for BLOCK=1
        // no idea what the actual bound needed here is.
        const int MAX_STEPS = 10 + 2 * n + n * n / BLOCK / 2;
        mt19937 rng(random_device{}());
        for (int i = 0; i < MAX_STEPS; ++i) {
            if (unmat.empty()) break;
            int u = unmat.top().second;
            unmat.pop();
            if (mat[u] != -1) continue;
            for (int j = 0; j < BLOCK; ++j) {
                ++hit[u];
                auto &e = g[u];
                const int v = e[rng() % e.size()];
                mat[u] = v;
                swap(u, mat[v]);
            }
        }
    }
}

```

```

        if (u == -1) break;
    }
    if (u != -1) {
        mat[u] = -1;
        unmat.emplace(hit[u] * 100ULL / (g[u].size() +
1), u);
    }
}
int siz = 0;
for (auto e : mat) siz += (e != -1);
return siz / 2;
}
};

```

3 Graph

3.1 2-SAT

```

struct TwoSat {
    int n;
    vector<vector<int>> G;
    vector<bool> ans;
    vector<int> id, dfn, low, stk;
    TwoSat(int n) : n(n), G(2 * n), ans(n),
        id(2 * n, -1), dfn(2 * n, -1), low(2 * n, -1) {}
    void addClause(int u, bool f, int v, bool g) { // (u
        = f) or (v = g)
        G[2 * u + !f].push_back(2 * v + g);
        G[2 * v + !g].push_back(2 * u + f);
    }
    void addImPLY(int u, bool f, int v, bool g) { // (u =
        f) -> (v = g)
        G[2 * u + f].push_back(2 * v + g);
        G[2 * v + !g].push_back(2 * u + !f);
    }
    int cur = 0, scc = 0;
    void dfs(int u) {
        stk.push_back(u);
        dfn[u] = low[u] = cur++;
        for (int v : G[u]) {
            if (dfn[v] == -1) {
                dfs(v);
                chmin(low[u], low[v]);
            } else if (id[v] == -1) {
                chmin(low[u], dfn[v]);
            }
        }
        if (dfn[u] == low[u]) {
            int x;
            do {
                x = stk.back();
                stk.pop_back();
                id[x] = scc;
            } while (x != u);
            scc++;
        }
    }
    bool satisfiable() {
        for (int i = 0; i < n * 2; i++)
            if (dfn[i] == -1) {
                dfs(i);
            }
        for (int i = 0; i < n; ++i) {
            if (id[2 * i] == id[2 * i + 1]) {
                return false;
            }
        }
        ans[i] = id[2 * i] > id[2 * i + 1];
        return true;
    }
};

```

3.2 Tree

```

struct Tree {
    int n, lgN;
    vector<vector<int>> G;
    vector<vector<int>> st;
    vector<int> in, out, dep, pa, seq;
    Tree(int n) : n(n), G(n), in(n), out(n), dep(n), pa(n
        , -1) {}
    int cmp(int a, int b) {
        return dep[a] < dep[b] ? a : b;
    }
};

```

```

}
void dfs(int u) {
    erase(G[u], pa[u]);
    in[u] = seq.size();
    seq.push_back(u);
    for (int v : G[u]) {
        dep[v] = dep[u] + 1;
        pa[v] = u;
        dfs(v);
    }
    out[u] = seq.size();
}
void build() {
    seq.reserve(n);
    dfs(0);
    lgN = __lg(n);
    st.assign(lgN + 1, vector<int>(n));
    st[0] = seq;
    for (int i = 0; i < lgN; i++)
        for (int j = 0; j + (2 << i) <= n; j++)
            st[i + 1][j] = cmp(st[i][j], st[i][j + (1 << i)
        ]]);
}
int inside(int x, int y) {
    return in[x] <= in[y] and in[y] < out[x];
}
int lca(int x, int y) {
    if (x == y) return x;
    if ((x = in[x] + 1) > (y = in[y] + 1))
        swap(x, y);
    int h = __lg(y - x);
    return pa[cmp(st[h][x], st[h][y - (1 << h)])];
}
int dist(int x, int y) {
    return dep[x] + dep[y] - 2 * dep[lca(x, y)];
}
int rootPar(int r, int x) {
    if (r == x) return -1;
    if (!inside(x, r)) return pa[x];
    return *--upper_bound(all(G[x]), r,
        [&](int a, int b) -> bool {
            return in[a] < in[b];
        });
}
int size(int x) { return out[x] - in[x]; }
int rootSiz(int r, int x) {
    if (r == x) return n;
    if (!inside(x, r)) return size(x);
    return n - size(rootPar(r, x));
}
int rootLca(int a, int b, int c) {
    return lca(a, b) ^ lca(b, c) ^ lca(c, a);
}
vector<int> virTree(vector<int> ver) {
    sort(all(ver), [&](int a, int b) {
        return in[a] < in[b];
    });
    for (int i = ver.size() - 1; i > 0; i--)
        ver.push_back(lca(ver[i], ver[i - 1]));
    sort(all(ver), [&](int a, int b) {
        return in[a] < in[b];
    });
    ver.erase(unique(all(ver)), ver.end());
    return ver;
}
void inplace_virTree(vector<int> &ver) { // O(n),
    need sort before
    vector<int> ex;
    for (int i = 0; i + 1 < ver.size(); i++)
        if (!inside(ver[i], ver[i + 1]))
            ex.push_back(lca(ver[i], ver[i + 1]));
    vector<int> stk, pa(ex.size(), -1);
    for (int i = 0; i < ex.size(); i++) {
        int lst = -1;
        while (stk.size() and in[ex[stk.back()]] >= in[ex
            [i]]) {
            lst = stk.back();
            stk.pop_back();
        }
        if (lst != -1) pa[lst] = i;
        if (stk.size()) pa[i] = stk.back();
        stk.push_back(i);
    }
}

```

```

}
vector<bool> vis(ex.size());
auto dfs = [&](auto self, int u) -> void {
    vis[u] = 1;
    if (pa[u] != -1 and !vis[pa[u]])
        self(self, pa[u]);
    if (ex[u] != ver.back())
        ver.push_back(ex[u]);
};
const int s = ver.size();
for (int i = 0; i < ex.size(); i++)
    if (!vis[i]) dfs(dfs, i);
inplace_merge(ver.begin(), ver.begin() + s, ver.end(),
    [&](int a, int b) { return in[a] < in[b]; });
ver.erase(unique(all(ver)), ver.end());
}
};

```

3.3 Functional Graph

```

// bel[x]: x is belong bel[x]-th jellyfish
// len[x]: cycle length of x-th jellyfish
// ord[x]: order of x in cycle (x == root[x])
struct FunctionalGraph {
    int n, _t = 0;
    vector<vector<int>> G;
    vector<int> f, bel, dep, ord, root, in, out, len;
    FunctionalGraph(int n) : n(n), G(n), root(n),
        bel(n, -1), dep(n), ord(n), in(n), out(n) {}
    void dfs(int u) {
        in[u] = _t++;
        for (int v : G[u]) if (bel[v] == -1) {
            dep[v] = dep[u] + 1;
            root[v] = root[u];
            bel[v] = bel[u];
            dfs(v);
        }
        out[u] = _t;
    };
    void build(const auto &_f) {
        f = _f;
        for (int i = 0; i < n; i++) {
            G[f[i]].push_back(i);
        }
        vector<int> vis(n, -1);
        for (int i = 0; i < n; i++) if (vis[i] == -1) {
            int x = i;
            while (vis[x] == -1) {
                vis[x] = i;
                x = f[x];
            }
            if (vis[x] != i) continue;
            int s = x, l = 0;
            do {
                bel[x] = len.size();
                ord[x] = l++;
                root[x] = x;
                x = f[x];
            } while (x != s);
            len.push_back(l);
        }
        for (int i = 0; i < n; i++)
            if (root[i] == i) {
                dfs(i);
            }
    }
    int dist(int x, int y) { // x -> y
        if (bel[x] != bel[y]) {
            return -1;
        } else if (dep[x] < dep[y]) {
            return -1;
        } else if (dep[y] != 0) {
            if (in[y] <= in[x] and in[x] < out[y]) {
                return dep[x] - dep[y];
            }
            return -1;
        } else {
            return dep[x] + (ord[y] - ord[root[x]]) + len[bel[x]];
        }
    }
};

```

3.4 Manhattan MST

```

// {w, u, v}
vector<tuple<int, int, int>> ManhattanMST(vector<Pt> P)
{
    vector<int> id(P.size());
    iota(all(id), 0);
    vector<tuple<int, int, int>> edg;
    for (int k = 0; k < 4; k++) {
        sort(all(id), [&](int i, int j) {
            return (P[i] - P[j]).ff < (P[j] - P[i]).ss;
        });
        map<int, int> sweep;
        for (int i : id) {
            auto it = sweep.lower_bound(-P[i].ss);
            while (it != sweep.end()) {
                int j = it->ss;
                Pt d = P[i] - P[j];
                if (d.ss > d.ff) {
                    break;
                }
                edg.emplace_back(d.ff + d.ss, i, j);
                it = sweep.erase(it);
            }
            sweep[-P[i].ss] = i;
        }
        for (Pt &p : P) {
            if (k % 2) {
                p.ff = -p.ff;
            } else {
                swap(p.ff, p.ss);
            }
        }
    }
    return edg;
}

```

3.5 Count Cycles

```

// 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

```

3.6 Maximum Clique

```

constexpr size_t kN = 150;
using bits = bitset<kN>;
struct MaxClique {
    bits G[kN], cs[kN];
    int ans, sol[kN], q, cur[kN], d[kN], n;
    void init(int _n) {
        n = _n;
        for (int i = 0; i < n; ++i) G[i].reset();
    }
    void addEdge(int u, int v) {
        G[u][v] = G[v][u] = 1;
    }
    void preDfs(vector<int> &v, int i, bits mask) {
        if (i < 4) {
            for (int x : v) d[x] = (G[x] & mask).count();
            sort(all(v), [&](int x, int y) {
                return d[x] > d[y];
            });
        }
        vector<int> c(v.size());
        cs[1].reset(), cs[2].reset();
        int l = max(ans - q + 1, 1), r = 2, tp = 0, k;
        for (int p : v) {
            for (k = 1;
                (cs[k] & G[p]).any(); ++k);
            if (k >= r) cs[++r].reset();
            cs[k][p] = 1;
            if (k < l) v[tp++] = p;
        }
    }
};

```

```

    for (k = l; k < r; ++k)
        for (auto p = cs[k]._Find_first(); p < kN; p = cs[k]._Find_next(p))
            v[tp] = p, c[tp] = k, ++tp;
    dfs(v, c, i + 1, mask);
}
void dfs(vector<int> &v, vector<int> &c, int i, bits mask) {
    while (!v.empty()) {
        int p = v.back();
        v.pop_back();
        mask[p] = 0;
        if (q + c.back() <= ans) return;
        cur[q++] = p;
        vector<int> nr;
        for (int x : v)
            if (G[p][x]) nr.push_back(x);
        if (!nr.empty()) preDfs(nr, i, mask & G[p]);
        else if (q > ans) ans = q, copy_n(cur, q, sol);
        c.pop_back();
        --q;
    }
}
int solve() {
    vector<int> v(n);
    iota(all(v), 0);
    ans = q = 0;
    preDfs(v, 0, bits(string(n, '1')));
    return ans;
}
} cliq;

```

3.7 Min Mean Weight Cycle

```

// d[i][j] == 0 if {i,j} !in E
long long d[1003][1003], dp[1003][1003];

pair<long long, long long> MMWC() {
    memset(dp, 0x3f, sizeof(dp));
    for (int i = 1; i <= n; ++i) dp[0][i] = 0;
    for (int i = 1; i <= n; ++i) {
        for (int j = 1; j <= n; ++j) {
            for (int k = 1; k <= n; ++k) {
                dp[i][k] = min(dp[i - 1][j] + d[j][k], dp[i][k]);
            }
        }
    }
    long long au = 1ll << 31, ad = 1;
    for (int i = 1; i <= n; ++i) {
        if (dp[n][i] == 0x3f3f3f3f3f3f3f3f) continue;
        long long u = 0, d = 1;
        for (int j = n - 1; j >= 0; --j) {
            if ((dp[n][i] - dp[j][i]) * d > u * (n - j)) {
                u = dp[n][i] - dp[j][i];
                d = n - j;
            }
        }
        if (u * ad < au * d) au = u, ad = d;
    }
    long long g = __gcd(au, ad);
    return make_pair(au / g, ad / g);
}

```

3.8 Block Cut Tree

```

struct BlockCutTree {
    int n;
    vector<vector<int>> adj;
    BlockCutTree(int _n) : n(_n), adj(_n) {}
    void addEdge(int u, int v) {
        adj[u].push_back(v);
        adj[v].push_back(u);
    }
    pair<int, vector<pair<int, int>>> work() {
        vector<int> dfn(n, -1), low(n), stk;
        vector<pair<int, int>> edg;
        int cnt = 0, cur = 0;
        function<void(int)> dfs = [&](int x) {
            stk.push_back(x);
            dfn[x] = low[x] = cur++;
            for (auto y : adj[x]) {
                if (dfn[y] == -1) {
                    dfs(y);
                    low[x] = min(low[x], low[y]);
                }
            }
        };
    }
};

```

```

    if (low[y] == dfn[x]) {
        int v;
        do {
            v = stk.back();
            stk.pop_back();
            edg.emplace_back(n + cnt, v);
        } while (v != y);
        edg.emplace_back(x, n + cnt);
        cnt++;
    }
    else {
        low[x] = min(low[x], dfn[y]);
    }
}
};
for (int i = 0; i < n; i++) {
    if (dfn[i] == -1) {
        stk.clear();
        dfs(i);
    }
}
return {cnt, edg};
}
};

```

3.9 Dominator Tree

```

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

```

3.10 Matroid Intersection

```

template<class Matroid1, class Matroid2>
vector<bool> MatroidIntersection(Matroid1 &m1, Matroid2
    &m2) {
    const int N = m1.size();
    vector<bool> I(N);
}

```



```

while (true) {
    m1.set(I);
    m2.set(I);
    vector<vector<int>> E(N + 2);
    const int s = N, t = N + 1;
    for (int i = 0; i < N; i++) {
        if (I[i]) { continue; }
        auto c1 = m1.circuit(i);
        auto c2 = m2.circuit(i);
        if (c1.empty()) {
            E[s].push_back(i);
        } else {
            for (int y : c1) if (y != i) {
                E[y].push_back(i);
            }
        }
        if (c2.empty()) {
            E[i].push_back(t);
        } else {
            for (int y : c2) if (y != i) {
                E[i].push_back(y);
            }
        }
    }
    vector<int> pre(N + 2, -1);
    queue<int> que;
    que.push(s);
    while (que.size() and pre[t] == -1) {
        int u = que.front();
        que.pop();
        for (int v : E[u]) {
            if (pre[v] == -1) {
                pre[v] = u;
                que.push(v);
            }
        }
    }
    if (pre[t] == -1) { break; }
    for (int p = pre[t]; p != s; p = pre[p]) {
        I[p] = !I[p];
    }
}
return I;
}

```

3.11 Generalized Series-Parallel Graph

```

/* Vertex: {u, -1}
 * Edge: {u, v}; u < v
 * Series: (e1, v1, e2) => e3; e1 < e2
 * Parallel: (e1, e2) => e3; e1 = e2
 * Dangling: (v1, e1, v2) => v3; e1 = {v1, v2}
 */
struct GSPGraph {
    int N;
    vector<pair<int, int>> S;
    vector<vector<int>> tree;
    vector<bool> isrt;
    int getv(int e, int u) { return S[e].ff ^ S[e].ss ^ u; }
    int newNode(pair<int, int> s, vector<int> sub) {
        S[N] = s, tree[N] = sub;
        for (int x : sub) isrt[x] = false;
        return N++;
    }
    GSPGraph(int n, const vector<pair<int, int>> &edge) {
        N = edge.size();
        S = edge;
        S.resize(N * 2 + n, {-1, -1});
        tree.resize(N * 2 + n);
        isrt.assign(N * 2 + n, true);
        vector<vector<int>> G(n);
        vector<int> vid(n), deg(n);
        unordered_map<pair<int, int>, int> eid;
        queue<int> que;
        auto add = [&](int e) {
            auto [u, v] = S[e];
            if (auto it = eid.find(S[e]); it != eid.end()) {
                it->ss = e = newNode(S[e], {e, it->ss});
                if (--deg[u] == 2) que.push(u);
                if (--deg[v] == 2) que.push(v);
            } else eid[S[e]] = e;
            G[u].push_back(e);

```

```

            G[v].push_back(e);
        };
        for (int i = N - 1; i >= 0; i--) {
            S[i] = minmax({S[i].ff, S[i].ss});
            add(i);
        }
        for (int i = 0; i < n; i++) {
            S[vid[i] = N++] = {i, -1};
            deg[i] += ssize(G[i]);
            if (deg[i] <= 2) que.push(i);
        }
        auto pop = [&](int x) {
            while (!isrt[G[x].back()]) G[x].pop_back();
            int e = G[x].back();
            isrt[e] = false;
            return e;
        };
        while (que.size()) {
            int u = que.front(); que.pop();
            if (deg[u] == 1) {
                int e = pop(u), v = getv(e, u);
                vid[v] = newNode(
                    {v, -1}, {vid[S[e].ff], e, vid[S[e].ss]}
                );
                if (--deg[v] == 2) que.push(v);
            } else if (deg[u] == 2) {
                int e1 = pop(u), e2 = pop(u);
                if (S[e1] > S[e2]) swap(e1, e2);
                add(newNode(
                    minmax(getv(e1, u), getv(e2, u)),
                    {e1, vid[u], e2}
                ));
            }
        }
        S.resize(N);
        tree.resize(N);
        isrt.resize(N);
    }
};

```

4 Data Structure

4.1 Lazy Segtree

```

template<class S, class T>
struct Seg {
    Seg *ls{}, *rs{};
    S sum{};
    T tag{};
    int l, r;
    Seg(int _l, int _r) : l(_l), r(_r) {
        if (r - l == 1) {
            return;
        }
        int m = (l + r) / 2;
        ls = new Seg(l, m);
        rs = new Seg(m, r);
        pull();
    }
    void pull() {
        sum = ls->sum + rs->sum;
    }
    void push() {
        ls->apply(tag);
        rs->apply(tag);
        tag = T{};
    }
    void apply(const T &f) {
        f(tag);
        f(sum);
    }
    S query(int x, int y) {
        if (y <= l or r <= x) {
            return sum;
        }
        if (x <= l and r <= y) {
            return sum;
        }
        push();
        return ls->query(x, y) + rs->query(x, y);
    }
    void apply(int x, int y, const T &f) {
        if (y <= l or r <= x) {

```

```

    return;
}
if (x <= l and r <= y) {
    apply(f);
    return;
}
push();
ls->apply(x, y, f);
rs->apply(x, y, f);
pull();
}
void set(int p, const S &e) {
    if (p < l or p >= r) {
        return;
    }
    if (r - l == 1) {
        sum = e;
        return;
    }
    push();
    ls->set(p, e);
    rs->set(p, e);
    pull();
}
pair<int, S> findFirst(int x, int y, auto &&pred, S
    cur = {}) {
    if (y <= l or r <= x) {
        return {-1, cur};
    }
    if (x <= l and r <= y and !pred(cur + sum)) {
        return {-1, cur + sum};
    }
    if (r - l == 1) {
        return {l, cur + sum};
    }
    push();
    auto L = ls->findFirst(x, y, pred, cur);
    if (L.ff != -1) {
        return L;
    }
    return rs->findFirst(x, y, pred, L.ss);
}
pair<int, S> findLast(int x, int y, auto &&pred, S
    cur = {}) {
    if (y <= l or r <= x) {
        return {-1, cur};
    }
    if (x <= l and r <= y and !pred(sum + cur)) {
        return {-1, sum + cur};
    }
    if (r - l == 1) {
        return {l, sum + cur};
    }
    push();
    auto R = rs->findLast(x, y, pred, cur);
    if (R.ff != -1) {
        return R;
    }
    return ls->findLast(x, y, pred, R.ss);
}
};

```

4.2 Fenwick Tree

```

template<class T>
struct Fenwick {
    int n;
    vector<T> a;
    Fenwick(int _n) : n(_n), a(_n) {}
    int lob(int x) { return x & -x; }
    void add(int p, T x) {
        assert(p < n);
        for (int i = p + 1; i <= n; i += lob(i)) {
            a[i - 1] = a[i - 1] + x;
        }
    }
    T sum(int p) { // sum [0, p]
        T s{};
        for (int i = min(p, n) + 1; i > 0; i -= lob(i)) {
            s = s + a[i - 1];
        }
        return s;
    }
};

```

```

int findFirst(auto &&pred) { // min{ k | pred(k) }
    T s{};
    int p = 0;
    for (int i = 1 << __lg(n); i; i >>= 1) {
        if (p + i <= n and !pred(s + a[p + i - 1])) {
            p += i;
            s = s + a[p - 1];
        }
    }
    return p == n ? -1 : p;
};

```

4.3 Interval Segtree

```

struct Seg {
    Seg *ls, *rs;
    int l, r;
    vector<int> f, g;
    // f : intervals where covering [l, r]
    // g : intervals where interset with [l, r]
    Seg(int _l, int _r) : l{_l}, r{_r} {
        int mid = (l + r) >> 1;
        if (r - l == 1) return;
        ls = new Seg(l, mid);
        rs = new Seg(mid, r);
    }
    void insert(int x, int y, int id) {
        if (y <= l or r <= x) return;
        g.push_back(id);
        if (x <= l and r <= y) {
            f.push_back(id);
            return;
        }
        ls->insert(x, y, id);
        rs->insert(x, y, id);
    }
    void fix() {
        while (!f.empty() and use[f.back()]) f.pop_back();
        while (!g.empty() and use[g.back()]) g.pop_back();
    }
    int query(int x, int y) {
        if (y <= l or r <= x) return -1;
        fix();
        if (x <= l and r <= y) {
            return g.empty() ? -1 : g.back();
        }
        return max({f.empty() ? -1 : f.back(), ls->query(x,
            y), rs->query(x, y)});
    }
};

```

4.4 PrefixMax Sum Segtree

```

// O(Nlog^2N)!
const int kC = 1E6;
struct Seg {
    static Seg pool[kC], *top;
    Seg *ls, *rs;
    int l, r;
    i64 sum = 0, rsum = 0, mx = 0;
    Seg() {}
    Seg(int _l, int _r, const vector<i64> &v) : l(_l), r(
        _r) {
        if (r - l == 1) {
            sum = mx = v[l];
            return;
        }
        int m = (l + r) / 2;
        ls = new (top++) Seg(l, m, v);
        rs = new (top++) Seg(m, r, v);
        pull();
    }
    i64 cal(i64 h) { // sigma i in [l, r) max(h, v[i])
        if (r - l == 1) {
            return max(mx, h);
        }
        if (mx <= h) {
            return h * (r - l);
        }
        if (ls->mx >= h) {
            return ls->cal(h) + rsum;
        }
        return h * (ls->r - ls->l) + rs->cal(h);
    }
};

```



```

}
void pull() {
    rsum = rs->cal(ls->mx);
    sum = ls->sum + rsum;
    mx = max(ls->mx, rs->mx);
}
void set(int p, i64 h) {
    if (r - l == 1) {
        sum = mx = h;
        return;
    }
    int m = (l + r) / 2;
    if (p < m) {
        ls->set(p, h);
    } else {
        rs->set(p, h);
    }
    pull();
}
i64 query(int p, i64 h) { // sigma i in [0, p) max(h, v[i])
    if (p <= l) {
        return 0;
    }
    if (p >= r) {
        return cal(h);
    }
    return ls->query(p, h) + rs->query(p, max(h, ls->mx));
}
} Seg::pool[kC], *Seg::top = Seg::pool;

```

4.5 Disjoint Set Union-undo

```

template<class T>
struct DSU {
    vector<T> tag;
    vector<int> f, siz, stk;
    int cc;
    DSU(int n) : f(n, -1), siz(n, 1), tag(n), cc(n) {}
    int find(int x) { return f[x] < 0 ? x : find(f[x]); }
    bool merge(int x, int y) {
        x = find(x);
        y = find(y);
        if (x == y) return false;
        if (siz[x] > siz[y]) swap(x, y);
        f[x] = y;
        siz[y] += siz[x];
        tag[x] = tag[x] - tag[y];
        stk.push_back(x);
        cc--;
        return true;
    }
    void apply(int x, T s) {
        x = find(x);
        tag[x] = tag[x] + s;
    }
    void undo() {
        int x = stk.back();
        int y = f[x];
        stk.pop_back();
        tag[x] = tag[x] + tag[y];
        siz[y] -= siz[x];
        f[x] = -1;
        cc++;
    }
    bool same(int x, int y) { return find(x) == find(y); }
    int size(int x) { return siz[find(x)]; }
};

```

4.6 PBDS

```

#include <bits/extc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
#include <ext/pb_ds/hash_policy.hpp>
#include <ext/pb_ds/priority_queue.hpp>
using namespace __gnu_pbds;
template<class T>
using BST = tree<T, null_type, less<T>, rb_tree_tag,
tree_order_statistics_node_update>;
// __gnu_pbds::priority_queue<node, decltype(cmp),
pairing_heap_tag> pq(cmp);

```

```

// gp_hash_table<int, gnu_pbds::priority_queue<node>::
point_iterator> pqPos;
// bst.insert((x << 20) + i);
// bst.erase(bst.lower_bound(x << 20));
// bst.order_of_key(x << 20) + 1;
// *bst.find_by_order(x - 1) >> 20;
// *--bst.lower_bound(x << 20) >> 20;
// *bst.upper_bound((x + 1) << 20) >> 20;

```

4.7 Centroid Decomposition

```

struct CenDec {
    vector<vector<pair<int, i64>>> G;
    vector<vector<i64>> pdis;
    vector<int> pa, ord, siz;
    vector<bool> vis;
    int getsiz(int u, int f) {
        siz[u] = 1;
        for (auto [v, w] : G[u]) if (v != f and !vis[v])
            siz[u] += getsiz(v, u);
        return siz[u];
    }
    int find(int u, int f, int s) {
        for (auto [v, w] : G[u]) if (v != f and !vis[v])
            if (siz[v] * 2 >= s) return find(v, u, s);
        return u;
    };
    void caldis(int u, int f, i64 dis) {
        pdis[u].push_back(dis);
        for (auto [v, w] : G[u]) if (v != f and !vis[v]) {
            caldis(v, u, dis + w);
        }
    }
    int build(int u = 0) {
        u = find(u, u, getsiz(u, u));
        ord.push_back(u);
        vis[u] = 1;
        for (auto [v, w] : G[u]) if (!vis[v]) {
            pa[build(v)] = u;
        }
        caldis(u, -1, 0); // if need
        vis[u] = 0;
        return u;
    };
    CenDec(int n) : G(n), pa(n, -1), vis(n), siz(n), pdis(n) {}
};

```

4.8 2D BIT

```

template<class T>
struct BIT2D {
    vector<vector<T>> val;
    vector<vector<int>> Y;
    vector<int> X;
    int lowbit(int x) { return x & -x; }
    int getp(const vector<int> &v, int x) {
        return upper_bound(all(v), x) - v.begin();
    }
    BIT2D(vector<pair<int, int>> pos) {
        for (auto &[x, y] : pos) {
            X.push_back(x);
            swap(x, y);
        }
        sort(all(pos));
        sort(all(X));
        X.erase(unique(all(X)), X.end());
        Y.resize(X.size() + 1);
        val.resize(X.size() + 1);
        for (auto [y, x] : pos) {
            for (int i = getp(X, x); i <= X.size(); i += lowbit(i))
                if (Y[i].empty() or Y[i].back() != y)
                    Y[i].push_back(y);
        }
        for (int i = 1; i <= X.size(); i++) {
            val[i].assign(Y[i].size() + 1, T{});
        }
    }
    void add(int x, int y, T v) {
        for (int i = getp(X, x); i <= X.size(); i += lowbit(i))
            for (int j = getp(Y[i], y); j <= Y[i].size(); j += lowbit(j))

```

```

        val[i][j] += v;
    }
    T qry(int x, int y) {
        T r{};
        for (int i = getp(X, x); i > 0; i -= lowbit(i))
            for (int j = getp(Y[i], y); j > 0; j -= lowbit(j))
                r += val[i][j];
        return r;
    }
};

```

4.9 Big Binary

```

struct BigBinary : map<int, int> {
    void split(int x) {
        auto it = lower_bound(x);
        if (it != begin()) {
            it--;
            if (it->ss > x) {
                (*this)[x] = it->ss;
                it->ss = x;
            }
        }
    }
    void add(int x) {
        split(x);
        auto it = find(x);
        while (it != end() and it->ff == x) {
            x = it->ss;
            it = erase(it);
        }
        (*this)[x] = x + 1;
    }
    void sub(int x) {
        split(x);
        auto it = lower_bound(x);
        // assert(it != end());
        auto [l, r] = *it;
        erase(it);
        if (l + 1 < r) {
            (*this)[l + 1] = r;
        }
        if (x < l) {
            (*this)[x] = l;
        }
    }
};

```

4.10 Big Integer

// 暴力乘法，只能做到 10^5 位數
 // 只加減不做乘法 Base 可到 $1E18$

```

struct uBig {
    static const i64 Base = 1E15;
    static const i64 Log = 15;
    vector<i64> d;
    uBig() : d{0} {}
    uBig(i64 x) {
        d = {x % Base};
        if (x >= Base) {
            d.push_back(x / Base);
        }
        fix();
    }
    uBig(string_view s) {
        i64 c = 0, pw = 1;
        for (int i = s.size() - 1; i >= 0; i--) {
            c += pw * (s[i] - '0');
            pw *= 10;
            if (pw == Base or i == 0) {
                d.push_back(c);
                c = 0;
                pw = 1;
            }
        }
    }
    void fix() {
        i64 c = 0;
        for (int i = 0; i < d.size(); i++) {
            d[i] += c;
            c = (d[i] < 0 ? (d[i] - 1 - Base) / Base : d[i] / Base);
        }
    }
};

```

```

        d[i] -= c * Base;
    }
    while (c) {
        d.push_back(c % Base);
        c /= Base;
    }
    while (d.size() >= 2 and d.back() == 0) {
        d.pop_back();
    }
}
bool isZero() const {
    return d.size() == 1 and d[0] == 0;
}
uBig &operator+=(const uBig &rhs) {
    if (d.size() < rhs.d.size()) {
        d.resize(rhs.d.size());
    }
    for (int i = 0; i < rhs.d.size(); i++) {
        d[i] += rhs.d[i];
    }
    fix();
    return *this;
}
uBig &operator-=(const uBig &rhs) {
    if (d.size() < rhs.d.size()) {
        d.resize(rhs.d.size());
    }
    for (int i = 0; i < rhs.d.size(); i++) {
        d[i] -= rhs.d[i];
    }
    fix();
    return *this;
}
friend uBig operator*(const uBig &lhs, const uBig &rhs) {
    const int a = lhs.d.size(), b = rhs.d.size();
    uBig res(0);
    res.d.resize(a + b);
    for (int i = 0; i < a; i++) {
        for (int j = 0; j < b; j++) {
            i128 x = (i128)lhs.d[i] * rhs.d[j];
            res.d[i + j] += x % Base;
            res.d[i + j + 1] += x / Base;
        }
    }
    res.fix();
    return res;
};
friend uBig &operator+(uBig lhs, const uBig &rhs) {
    return lhs += rhs;
}
friend uBig &operator-(uBig lhs, const uBig &rhs) {
    return lhs -= rhs;
}
uBig &operator*=(const uBig &rhs) {
    return *this = *this * rhs;
}
friend int cmp(const uBig &lhs, const uBig &rhs) {
    if (lhs.d.size() != rhs.d.size()) {
        return lhs.d.size() < rhs.d.size() ? -1 : 1;
    }
    for (int i = lhs.d.size() - 1; i >= 0; i--) {
        if (lhs.d[i] != rhs.d[i]) {
            return lhs.d[i] < rhs.d[i] ? -1 : 1;
        }
    }
    return 0;
}
friend ostream &operator<<(ostream &os, const uBig &rhs) {
    os << rhs.d.back();
    for (int i = ssize(rhs.d) - 2; i >= 0; i--) {
        os << setfill('0') << setw(Log) << rhs.d[i];
    }
    return os;
}
friend istream &operator>>(istream &is, uBig &rhs) {
    string s;
    is >> s;
    rhs = uBig(s);
    return is;
}

```

```

};

struct sBig : uBig {
    bool neg{false};
    sBig() : uBig() {}
    sBig(int64 x) : uBig(abs(x)), neg(x < 0) {}
    sBig(string_view s) : uBig(s[0] == '-' ? s.substr(1)
        : s), neg(s[0] == '-') {}
    sBig(const uBig &x) : uBig(x) {}
    sBig operator-(const {
        if (isZero()) {
            return *this;
        }
        sBig res = *this;
        res.neg ^= 1;
        return res;
    }
    sBig &operator+=(const sBig &rhs) {
        if (rhs.isZero()) {
            return *this;
        }
        if (neg == rhs.neg) {
            uBig::operator+=(rhs);
        } else {
            int s = cmp(*this, rhs);
            if (s == 0) {
                *this = {};
            } else if (s == 1) {
                uBig::operator-=(rhs);
            } else {
                uBig tmp = rhs;
                tmp -= static_cast<uBig>(*this);
                *this = tmp;
                neg = rhs.neg;
            }
        }
        return *this;
    }
    sBig &operator-=(const sBig &rhs) {
        neg ^= 1;
        *this += rhs;
        neg ^= 1;
        if (isZero()) {
            neg = false;
        }
        return *this;
    }
    sBig &operator*=(const sBig &rhs) {
        if (isZero() or rhs.isZero()) {
            return *this = {};
        }
        neg ^= rhs.neg;
        uBig::operator*=(rhs);
        return *this;
    }
    friend sBig operator+(sBig lhs, const sBig &rhs) {
        return lhs += rhs;
    }
    friend sBig &operator-(sBig lhs, const sBig &rhs) {
        return lhs -= rhs;
    }
    friend sBig operator*(sBig lhs, const sBig &rhs) {
        return lhs *= rhs;
    }
    friend ostream &operator<<(ostream &os, const sBig &
        rhs) {
        if (rhs.neg) {
            os << '-';
        }
        return os << static_cast<uBig>(rhs);
    }
    friend istream &operator>>(istream &is, sBig &rhs) {
        string s;
        is >> s;
        rhs = sBig(s);
        return is;
    }
};

```

4.11 Splay Tree

```

struct Node {
    Node *ch[2]{}, *p{};

```

```

    Info info{}, sum{};
    Tag tag{};
    int size{};
    bool rev{};
} pool[int(1E5 + 10)], *top = pool;
Node *newNode(Info a) {
    Node *t = top++;
    t->info = t->sum = a;
    t->size = 1;
    return t;
}
int size(const Node *x) { return x ? x->size : 0; }
Info get(const Node *x) { return x ? x->sum : Info{}; }
int dir(const Node *x) { return x->p->ch[1] == x; }
bool nroot(const Node *x) { return x->p and x->p->ch[
    dir(x)] == x; }
void reverse(Node *x) { if (x) x->rev = !x->rev; }
void update(Node *x, const Tag &f) {
    if (!x) return;
    f(x->tag);
    f(x->info);
    f(x->sum);
}
void push(Node *x) {
    if (x->rev) {
        swap(x->ch[0], x->ch[1]);
        reverse(x->ch[0]);
        reverse(x->ch[1]);
        x->rev = false;
    }
    update(x->ch[0], x->tag);
    update(x->ch[1], x->tag);
    x->tag = Tag{};
}
void pull(Node *x) {
    x->size = size(x->ch[0]) + 1 + size(x->ch[1]);
    x->sum = get(x->ch[0]) + x->info + get(x->ch[1]);
}
void rotate(Node *x) {
    Node *y = x->p, *z = y->p;
    push(y);
    int d = dir(x);
    push(x);
    Node *w = x->ch[d ^ 1];
    if (nroot(y)) {
        z->ch[dir(y)] = x;
    }
    if (w) {
        w->p = y;
    }
    (x->ch[d ^ 1] = y)->ch[d] = w;
    (y->p = x)->p = z;
    pull(y);
    pull(x);
}
void splay(Node *x) {
    while (nroot(x)) {
        Node *y = x->p;
        if (nroot(y)) {
            rotate(dir(x) == dir(y) ? y : x);
        }
        rotate(x);
    }
}
Node *nth(Node *x, int k) {
    assert(size(x) > k);
    while (true) {
        push(x);
        int left = size(x->ch[0]);
        if (left > k) {
            x = x->ch[0];
        } else if (left < k) {
            k -= left + 1;
            x = x->ch[1];
        } else {
            break;
        }
    }
    splay(x);
    return x;
}
Node *split(Node *x) {

```

```

assert(x);
push(x);
Node *l = x->ch[0];
if (l) l->p = x->ch[0] = nullptr;
pull(x);
return l;
}
Node *join(Node *x, Node *y) {
    if (!x or !y) return x ? x : y;
    y = nth(y, 0);
    push(y);
    y->ch[0] = x;
    if (x) x->p = y;
    pull(y);
    return y;
}
Node *find_first(Node *x, auto &&pred) {
    Info pre{};
    while (true) {
        push(x);
        if (pred(pre + get(x->ch[0]))) {
            x = x->ch[0];
        } else if (pred(pre + get(x->ch[0]) + x->info) or !
            x->ch[1]) {
            break;
        } else {
            pre = pre + get(x->ch[0]) + x->info;
            x = x->ch[1];
        }
    }
    splay(x);
    return x;
}

```

4.12 Link Cut Tree

```

namespace lct {
Node *access(Node *x) {
    Node *last = {};
    while (x) {
        splay(x);
        push(x);
        x->ch[0] = last;
        pull(x);
        last = x;
        x = x->p;
    }
    return last;
}
void make_root(Node *x) {
    access(x);
    splay(x);
    reverse(x);
}
Node *find_root(Node *x) {
    push(x = access(x));
    while (x->ch[1]) {
        push(x = x->ch[1]);
    }
    splay(x);
    return x;
}
bool link(Node *x, Node *y) {
    if (find_root(x) == find_root(y)) {
        return false;
    }
    make_root(x);
    x->p = y;
    return true;
}
bool cut(Node *a, Node *b) {
    make_root(a);
    access(b);
    splay(a);
    if (a->ch[0] == b) {
        split(a);
        return true;
    }
    return false;
}
Info query(Node *a, Node *b) {
    make_root(b);
    return get(access(a));
}
}

```

```

}
void set(Node *x, Info v) {
    splay(x);
    push(x);
    x->info = v;
    pull(x);
} }

```

4.13 Static Top Tree

```

template<class Vertex, class Path>
struct StaticTopTree {
    enum Type { Rake, Compress, Combine, Convert };
    int stt_root;
    vector<vector<int>> &G;
    vector<int> P, L, R, S;
    vector<Type> T;
    vector<Vertex> f;
    vector<Path> g;
    int buf;
    int dfs(int u) {
        int s = 1, big = 0;
        for (int &v : G[u]) {
            erase(G[v], u);
            int t = dfs(v);
            s += t;
            if (chmax(big, t)) swap(G[u][0], v);
        }
        return s;
    }
    int add(int l, int r, Type t) {
        int x = buf++;
        P[x] = -1, L[x] = l, R[x] = r, T[x] = t;
        if (l != -1) P[l] = x, S[x] += S[l];
        if (r != -1) P[r] = x, S[x] += S[r];
        return x;
    }
    int merge(auto l, auto r, Type t) {
        if (r - l == 1) return *l;
        int s = 0;
        for (auto i = l; i != r; i++) s += S[*i];
        auto m = l;
        while (s > S[*m]) s -= 2 * S[*m++];
        return add(merge(l, m, t), merge(m, r, t), t);
    }
    int pathCluster(int u) {
        vector<int> chs{pointCluster(u)};
        while (!G[u].empty()) chs.push_back(pointCluster(u
            = G[u][0]));
        return merge(all(chs), Type::Compress);
    }
    int pointCluster(int u) {
        vector<int> chs;
        for (int v : G[u] | views::drop(1))
            chs.push_back(add(pathCluster(v), -1, Type::
                Convert));
        if (chs.empty()) return add(u, -1, Type::Convert);
        return add(u, merge(all(chs), Type::Rake), Type::
            Combine);
    }
    StaticTopTree(vector<vector<int>> &G, int root = 0)
        : G(G) {
        const int n = G.size();
        P.assign(4 * n, -1);
        L.assign(4 * n, -1);
        R.assign(4 * n, -1);
        S.assign(4 * n, 1);
        T.assign(4 * n, Type::Rake);
        buf = n;
        dfs(root);
        stt_root = pathCluster(root);
        f.resize(buf);
        g.resize(buf);
    }
    void update(int x) {
        if (T[x] == Rake) f[x] = f[L[x]] * f[R[x]];
        else if (T[x] == Compress) g[x] = g[L[x]] + g[R[x]
            ];
        else if (T[x] == Combine) g[x] = f[L[x]] + f[R[x]];
        else if (T[L[x]] == Rake) g[x] = Path(f[L[x]]);
        else f[x] = Vertex(g[L[x]]);
    }
    void set(int x, const Vertex &v) {

```

```

    f[x] = v;
    for (x = P[x]; x != -1; x = P[x])
        update(x);
}
Vertex get() { return g[stt_root]; }
};
struct Path;
struct Vertex {
    Vertex() {}
    Vertex(const Path&);
};
struct Path {
    Path() {}
    Path(const Vertex&);
};
Vertex operator*(const Vertex &a, const Vertex &b) {
    return {};
}
Path operator+(const Vertex &a, const Vertex &b) {
    return {};
}
Path operator+(const Path &a, const Path &b) {
    return {};
}
Vertex::Vertex(const Path &x) {}
Path::Path(const Vertex &x) {}
/*
 * (root) 1 - 2 (heavy)
 *      / \
 *     3 4 5
 * type V: subtree DP info (Commutative Semigroup)
 * type P: path DP info (Semigroup)
 * V(2) + V(5) -> P(2)
 * V(1) + (V(3) * V(4)) -> P(1)
 * ans: V(P(1) + P(2))
 */

```

5 Math

5.1 Theorem

- Pick's Theorem**
 $A = i + \frac{b}{2} - 1$
 A : Area, i : grid number in the inner, b : grid number on the side
- Matrix-Tree theorem**
 undirected graph
 $D_{ii}(G) = \deg(i), D_{ij} = 0, i \neq j$
 $A_{ij}(G) = A_{ji}(G) = \#e(i, j), i \neq j$
 $L(G) = D(G) - A(G)$
 $t(G) = \det L(G)_{(1,2,\dots,i-1,i+1,\dots,n)}$
 leaf to root
 $D_{ii}^{out}(G) = \deg^{out}(i), D_{ij}^{out} = 0, i \neq j$
 $A_{ij}(G) = \#e(i, j), i \neq j$
 $L^{out}(G) = D^{out}(G) - A(G)$
 $t^{root}(G, k) = \det L^{out}(G)_{(1,2,\dots,k-1,k+1,\dots,n)}$
 root to leaf
 $L^{in}(G) = D^{in}(G) - A(G)$
 $t^{leaf}(G, k) = \det L^{in}(G)_{(1,2,\dots,k-1,k+1,\dots,n)}$
- Derangement**
 $D_n = (n-1)(D_{n-1} + D_{n-2}) = nD_{n-1} + (-1)^n$
- Möbius Inversion**
 $f(n) = \sum_{d|n} g(d) \Leftrightarrow g(n) = \sum_{d|n} \mu\left(\frac{n}{d}\right) f(d)$
- Euler Inversion**
 $\sum_{i|n} \varphi(i) = n$
- Binomial Inversion**
 $f(n) = \sum_{i=0}^n \binom{n}{i} g(i) \Leftrightarrow g(n) = \sum_{i=0}^n (-1)^{n-i} \binom{n}{i} f(i)$
- Subset Inversion**
 $f(S) = \sum_{T \subseteq S} g(T) \Leftrightarrow g(S) = \sum_{T \subseteq S} (-1)^{|S|-|T|} f(T)$
- Min-Max Inversion**
 $\max_{i \in S} x_i = \sum_{T \subseteq S} (-1)^{|T|-1} \min_{j \in T} x_j$
- Ex Min-Max Inversion**
 $kthmax_{i \in S} x_i = \sum_{T \subseteq S} (-1)^{|T|-k} \binom{|T|-1}{k-1} \min_{j \in T} x_j$
- Lcm-Gcd Inversion**
 $lcm_{i \in S} x_i = \prod_{T \subseteq S} (\gcd_{j \in T} x_j)^{(-1)^{|T|-1}}$
- Sum of powers**
 $\sum_{k=1}^n k^m = \frac{1}{m+1} \sum_{k=0}^m \binom{m+1}{k} B_k^+ n^{m+1-k}$
- $\sum_{j=0}^m \binom{m+1}{j} B_j^- = 0$
 note: $B_1^+ = -B_1^-, B_i^+ = B_i^-$
- Cayley's formula**
 number of trees on n labeled vertices: n^{n-2}
 Let $T_{n,k}$ be the number of labelled forests on n vertices with k connected components, such that vertices $1, 2, \dots, k$ all belong to different connected components. Then $T_{n,k} = kn^{n-k-1}$.
- High order residue**
 $[d^{\frac{p-1}{n, p-1}}] \equiv 1$
- Packing and Covering**
 $|\text{maximum independent set}| + |\text{minimum vertex cover}| = |V|$
- König's theorem**
 $|\text{maximum matching}| = |\text{minimum vertex cover}|$
- Dilworth's theorem**
 $\text{width} = |\text{largest antichain}| = |\text{smallest chain decomposition}|$
- Mirsky's theorem**
 $\text{height} = |\text{longest chain}| = |\text{smallest antichain decomposition}| = |\text{minimum antichain partition}|$
- Lucas' Theorem**
 For $n, m \in \mathbb{Z}^+$ and prime P , $\binom{m}{n} \mod P = \prod \binom{m_i}{n_i}$ where m_i is the i -th digit of m in base P .
- Stirling approximation**
 $n! \approx \sqrt{2\pi n} \left(\frac{n}{e}\right)^n e^{\frac{1}{12n}}$
- 1st Stirling Numbers(permutation $|P| = n$ with k cycles)**
 $S(n, k) = \text{coefficient of } x^k \text{ in } \prod_{i=0}^{n-1} (x+i)$
 $S(n+1, k) = nS(n, k) + S(n, k-1)$
- 2nd Stirling Numbers(Partition n elements into k non-empty set)**
 $S(n, k) = \frac{1}{k!} \sum_{j=0}^k (-1)^{k-j} \binom{k}{j} j^n$
 $S(n+1, k) = kS(n, k) + S(n, k-1)$
- Catalan number**
 $C_n = \frac{1}{n+1} \binom{2n}{n} = \binom{2n}{n} - \binom{2n}{n-1}$
 $\binom{n+m}{n} - \binom{n+m}{n+1} = (m+n)! \frac{n-m+1}{n+1}$ for $n \geq m$
 $C_n = \frac{1}{n+1} \binom{2n}{n} = \frac{(2n)!}{(n+1)n!}$
 $C_0 = 1$ and $C_{n+1} = 2 \binom{2n+1}{n+2} C_n$
 $C_0 = 1$ and $C_{n+1} = \sum_{i=0}^n C_i C_{n-i}$ for $n \geq 0$
- Extended Catalan number**
 $\frac{1}{(k-1)n+1} \binom{kn}{n}$
- Calculate $c[i-j] + = a[i] \times b[j]$ for $a[n], b[m]$**
 1. $a = \text{reverse}(a); c = \text{mul}(a, b); c = \text{reverse}(c[n]);$
 2. $b = \text{reverse}(b); c = \text{mul}(a, b); c = \text{rshift}(c, m-1);$
- Eulerian number (permutation $1 \sim n$ with m $a[i] > a[i-1]$)**
 $A(n, m) = \sum_{i=0}^m (-1)^i \binom{n+1}{i} (m+1-i)^n$
 $A(n, m) = (n-m)A(n-1, m-1) + (m+1)A(n-1, m)$
- Hall's theorem**
 Let $G = (X+Y, E)$ be a bipartite graph. For $W \subseteq X$, let $N(W) \subseteq Y$ denotes the adjacent vertices set of W . Then, G has a X' -perfect matching (matching contains $X' \subseteq X$) iff $\forall W \subseteq X', |W| \leq |N(W)|$.
- Tutte Matrix:**
 For a graph $G = (V, E)$, its maximum matching = $\frac{\text{rank}(A)}{2}$ where $A_{ij} = ((i, j) \in E ? (x_{ij} : -x_{ji}) : 0)$ and x_{ij} are random numbers.
- Erdős-Gallai theorem**
 There exists a simple graph with degree sequence $d_1 \geq \dots \geq d_n$ iff $\sum_{i=1}^n d_i$ is even and $\sum_{i=1}^k d_i \leq k(k-1) + \sum_{i=k+1}^n \min(d_i, k), \forall 1 \leq k \leq n$
- Euler Characteristic**
 planar graph: $V - E + F - C = 1$
 convex polyhedron: $V - E + F = 2$
 V, E, F, C : number of vertices, edges, faces(regions), and components
- Burnside Lemma**
 $|X/G| = \frac{1}{|G|} \sum_{g \in G} |X^g|$
- Polya theorem**
 $|Y^x/G| = \frac{1}{|G|} \sum_{g \in G} m^{c(g)}$
 $m = |Y|$: num of colors, $c(g)$: num of cycle
- Cayley's Formula**
 Given a degree sequence d_1, \dots, d_n of a labeled tree, there are $\frac{(n-2)!}{(d_1-1)! \dots (d_n-1)!}$ spanning trees.
- Find a Primitive Root of n :**
 n has primitive roots iff $n = 2, 4, p^k, 2p^k$ where p is an odd prime.
 1. Find $\phi(n)$ and all prime factors of $\phi(n)$, says $P = \{p_1, \dots, p_m\}$
 2. $\forall g \in [2, n]$, if $g^{\frac{\phi(n)}{p_i}} \neq 1, \forall p_i \in P$, then g is a primitive root.
 3. Since the smallest one isn't too big, the algorithm runs fast.
 4. n has exactly $\phi(\phi(n))$ primitive roots.

- Taylor series

$$f(x) = f(c) + f'(c)(x - c) + \frac{f''(c)}{2!}(x - c)^2 + \frac{f'''(c)}{3!}(x - c)^3 + \dots$$

- Lagrange Multiplier

$\min f(x, y)$, subject to $g(x, y) = 0$

$$\frac{\partial f}{\partial x} + \lambda \frac{\partial g}{\partial x} = 0$$

$$\frac{\partial f}{\partial y} + \lambda \frac{\partial g}{\partial y} = 0$$

$$g(x, y) = 0$$

- Calculate $f(x + n)$ where $f(x) = \sum_{i=0}^{n-1} a_i x^i$

$$f(x + n) = \sum_{i=0}^{n-1} a_i (x + n)^i = \sum_{i=0}^{n-1} x^i \cdot \frac{1}{i!} \sum_{j=i}^{n-1} \frac{a_j}{j!} \cdot \frac{n^{j-i}}{(j-i)!}$$

- Bell 數 (有 n 個人, 把他們拆組的方法總數)

$$B_0 = 1$$

$$B_n = \sum_{k=0}^n s(n, k) \quad (\text{second - stirling})$$

$$B_{n+1} = \sum_{k=0}^n \binom{n}{k} B_k$$

- Wilson's theorem

$$(p-1)! \equiv -1 \pmod{p}$$

$$(p^q!)_p \equiv \begin{cases} 1, & (p=2) \wedge (q \geq 3), \\ -1, & \text{otherwise.} \end{cases} \pmod{p^q}$$

- Fermat's little theorem

$$a^p \equiv a \pmod{p}$$

- Euler's theorem

$$a^b \equiv \begin{cases} a^{b \bmod \varphi(m)}, & \gcd(a, m) = 1, \\ a^b, & \gcd(a, m) \neq 1, b < \varphi(m), \\ a^{(b \bmod \varphi(m)) + \varphi(m)}, & \gcd(a, m) \neq 1, b \geq \varphi(m). \end{cases} \pmod{m}$$

- 環狀著色 (相鄰塗異色)

$$(k-1)(-1)^n + (k-1)^n$$

5.2 Linear Sieve

```
vector<int> primes, minp;
vector<int> mu, phi;
vector<bool> isp;
void Sieve(int n) {
    minp.assign(n + 1, 0);
    primes.clear();
    isp.assign(n + 1, 0);
    mu.resize(n + 1);
    phi.resize(n + 1);
    mu[1] = 1;
    phi[1] = 1;
    for (int i = 2; i <= n; i++) {
        if (minp[i] == 0) {
            minp[i] = i;
            isp[i] = 1;
            primes.push_back(i);
            mu[i] = -1;
            phi[i] = i - 1;
        }
        for (i64 p : primes) {
            if (p * i > n) {
                break;
            }
            minp[i * p] = p;
            if (p == minp[i]) {
                phi[p * i] = phi[i] * p;
                break;
            }
            phi[p * i] = phi[i] * (p - 1);
            mu[p * i] = mu[p] * mu[i];
        }
    }
}
```

5.3 Exgcd

```
// ax + by = gcd(a, b)
i64 exgcd(i64 a, i64 b, i64 &x, i64 &y) {
    if (b == 0) {
        x = 1, y = 0;
        return a;
    }
    i64 g = exgcd(b, a % b, y, x);
    y -= a / b * x;
    return g;
}
```

5.4 Chinese Remainder Theorem

```
// O(NlogC)
// E = {(m, r), ...}: x mod m_i = r_i
// return {M, R} x mod M = R
// return {-1, -1} if no solution
pair<i64, i64> CRT(vector<pair<i64, i64>> E) {
    i128 R = 0, M = 1;
    for (auto [m, r] : E) {
        i64 g, x, y, d;
        g = exgcd(M, m, x, y);
        d = r - R;
        if (d % g != 0) {
            return {-1, -1};
        }
        R += d / g * M * x;
        M = M * m / g;
        R = (R % M + M) % M;
    }
    return {M, R};
}
```

5.5 Factorize

```
u64 mul(u64 a, u64 b, u64 M) {
    i64 r = a * b - M * u64(1.L / M * a * b);
    return r + M * ((r < 0) - (r >= (i64)M));
}
u64 power(u64 a, u64 b, u64 M) {
    u64 r = 1;
    for (; b; b /= 2, a = mul(a, a, M))
        if (b & 1) r = mul(r, a, M);
    return r;
}
bool isPrime(u64 n) {
    if (n < 2 or n % 6 % 4 != 1) return (n | 1) == 3;
    auto magic = {2, 325, 9375, 28178, 450775, 9780504,
        1795265022};
    u64 s = __builtin_ctzll(n - 1), d = n >> s;
    for (u64 x : magic) {
        u64 p = power(x % n, d, n), i = s;
        while (p != 1 and p != n - 1 and x % n && i--)
            p = mul(p, p, n);
        if (p != n - 1 and i != s) return 0;
    }
    return 1;
}
u64 pollard(u64 n) {
    u64 c = 1;
    auto f = [&](u64 x) { return mul(x, x, n) + c; };
    u64 x = 0, y = 0, p = 2, q, t = 0;
    while (t++ % 128 or gcd(p, n) == 1) {
        if (x == y) c++, y = f(x = 2);
        if (q = mul(p, x > y ? x - y : y - x, n)) p = q;
        x = f(x); y = f(f(y));
    }
    return gcd(p, n);
}
u64 primeFactor(u64 n) {
    return isPrime(n) ? n : primeFactor(pollard(n));
}
```

5.6 FloorBlock

```
vector<i64> floorBlock(i64 x) { // x >= 0
    vector<i64> itv;
    for (i64 l = 1, r; l <= x; l = r) {
        r = x / (x / l) + 1;
        itv.push_back(l);
    }
    itv.push_back(x + 1);
    return itv;
}
```

5.7 FloorCeil

```
i64 ifloor(i64 a, i64 b) {
    if (b < 0) a = -a, b = -b;
    if (a < 0) return (a - b + 1) / b;
    return a / b;
}
i64 iceil(i64 a, i64 b) {
    if (b < 0) a = -a, b = -b;
    if (a > 0) return (a + b - 1) / b;
}
```



```
    return a / b;
}
```

5.8 NTT Prime List

Prime	Root	Prime	Root
7681	17	167772161	3
12289	11	104857601	3
40961	3	985661441	3
65537	3	998244353	3
786433	10	1107296257	10
5767169	3	2013265921	31
7340033	3	2810183681	11
23068673	3	2885681153	3
469762049	3	605028353	3
2748779069441	3	6597069766657	5
39582418599937	5	79164837199873	5
1231453023109121	3	1337006139375617	3
4179340454199820289	3	1945555039024054273	3
9223372036737335297	3		

5.9 NTT

```
template<i64 M, i64 root>
struct NTT {
    static const int Log = 21;
    array<i64, Log + 1> e{}, ie{};
    NTT() {
        static_assert(__builtin_ctz(M - 1) >= Log);
        e[Log] = power(root, (M - 1) >> Log, M);
        ie[Log] = power(e[Log], M - 2, M);
        for (int i = Log - 1; i >= 0; i--) {
            e[i] = e[i + 1] * e[i + 1] % M;
            ie[i] = ie[i + 1] * ie[i + 1] % M;
        }
    }
    void operator()(vector<i64> &v, bool inv) {
        int n = v.size();
        for (int i = 0, j = 0; i < n; i++) {
            if (i < j) swap(v[i], v[j]);
            for (int k = n / 2; (j ^= k) < k; k /= 2);
        }
        for (int m = 1; m < n; m *= 2) {
            i64 w = (inv ? ie : e)[__lg(m) + 1];
            for (int i = 0; i < n; i += m * 2) {
                i64 cur = 1;
                for (int j = i; j < i + m; j++) {
                    i64 g = v[j], t = cur * v[j + m] % M;
                    v[j] = (g + t) % M;
                    v[j + m] = (g - t + M) % M;
                    cur = cur * w % M;
                }
            }
        }
        if (inv) {
            i64 in = power(n, M - 2, M);
            for (int i = 0; i < n; i++) v[i] = v[i] * in % M;
        }
    }
};

template<int M, int G>
vector<i64> convolution(vector<i64> f, vector<i64> g) {
    static NTT<M, G> ntt;
    int n = ssize(f) + ssize(g) - 1;
    int len = bit_ceil(1ull * n);
    f.resize(len);
    g.resize(len);
    ntt(f, 0), ntt(g, 0);
    for (int i = 0; i < len; i++) {
        (f[i] *= g[i]) %= M;
    }
    ntt(f, 1);
    f.resize(n);
    return f;
}

vector<i64> inv(vector<i64> f) {
    const int n = f.size();
    int k = 1;
    vector<i64> g{inv(f[0])}, t;
    for (i64 &x : f) {
        x = (mod - x) % mod;
    }
    t.reserve(n);
    while (k < n) {
        k = min(k * 2, n);
        g.resize(k);
        t.assign(f.begin(), f.begin() + k);
    }
}
```

```
    auto h = g * t;
    h.resize(k);
    (h[0] += 2) %= mod;
    g = g * h;
    g.resize(k);
}
g.resize(n);
return g;
}

// CRT
vector<i64> convolution_ll(const vector<i64> &f, const
    vector<i64> &g) {
    constexpr i64 M1 = 998244353, G1 = 3;
    constexpr i64 M2 = 985661441, G2 = 3;
    constexpr i64 M1M2 = M1 * M2;
    constexpr i64 M1m1 = M2 * power(M2, M1 - 2, M1);
    constexpr i64 M2m2 = M1 * power(M1, M2 - 2, M2);
    auto c1 = convolution<M1, G1>(f, g);
    auto c2 = convolution<M2, G2>(f, g);
    for (int i = 0; i < c1.size(); i++) {
        c1[i] = ((i128)c1[i] * M1m1 + (i128)c2[i] * M2m2) %
            M1M2;
    }
    return c1;
}

// 2D convolution
vector<vector<i64>> operator*(vector<vector<i64>> f,
    vector<vector<i64>> g) {
    const int n = f.size() + g.size() - 1;
    const int m = f[0].size() + g[0].size() - 1;
    int len = bit_ceil(1ull * max(n, m));
    f.resize(len);
    g.resize(len);
    for (auto &v : f) {
        v.resize(len);
        ntt(v, 0);
    }
    for (auto &v : g) {
        v.resize(len);
        ntt(v, 0);
    }
    for (int i = 0; i < len; i++)
        for (int j = 0; j < i; j++) {
            swap(f[i][j], f[j][i]);
            swap(g[i][j], g[j][i]);
        }
    for (int i = 0; i < len; i++) {
        ntt(f[i], 0);
        ntt(g[i], 0);
    }
    for (int i = 0; i < len; i++)
        for (int j = 0; j < len; j++) {
            f[i][j] = mul(f[i][j], g[i][j]);
        }
    for (int i = 0; i < len; i++) {
        ntt(f[i], 1);
    }
    for (int i = 0; i < len; i++)
        for (int j = 0; j < i; j++) {
            swap(f[i][j], f[j][i]);
        }
    for (auto &v : f) {
        ntt(v, 1);
        v.resize(m);
    }
    f.resize(n);
    return f;
}
```

5.10 FWT

1. XOR Convolution

- $f(A) = (f(A_0) + f(A_1), f(A_0) - f(A_1))$
- $f^{-1}(A) = (f^{-1}(\frac{A_0 + A_1}{2}), f^{-1}(\frac{A_0 - A_1}{2}))$

2. OR Convolution

- $f(A) = (f(A_0), f(A_0) + f(A_1))$
- $f^{-1}(A) = (f^{-1}(A_0), f^{-1}(A_1) - f^{-1}(A_0))$

3. AND Convolution

- $f(A) = (f(A_0) + f(A_1), f(A_1))$
- $f^{-1}(A) = (f^{-1}(A_0) - f^{-1}(A_1), f^{-1}(A_1))$

5.11 FWT

```
void ORop(i64 &x, i64 &y) { y = (y + x) % mod; }
void ORinv(i64 &x, i64 &y) { y = (y - x + mod) % mod; }

void ANDop(i64 &x, i64 &y) { x = (x + y) % mod; }
void ANDinv(i64 &x, i64 &y) { x = (x - y + mod) % mod; }

void XORop(i64 &x, i64 &y) { tie(x, y) = pair{(x + y) % mod, (x - y + mod) % mod}; }
void XORinv(i64 &x, i64 &y) { tie(x, y) = pair{(x + y) * inv2 % mod, (x - y + mod) * inv2 % mod}; }

void FWT(vector<i64> &f, auto &op) {
    const int s = f.size();
    for (int i = 1; i < s; i *= 2)
        for (int j = 0; j < s; j += i * 2)
            for (int k = 0; k < i; k++)
                op(f[j + k], f[i + j + k]);
}
// FWT(f, XORop), FWT(g, XORop)
// f[i] *= g[i]
// FWT(f, XORinv)
```

5.12 Xor Basis

```
struct Basis {
    array<int, kD> bas{}, tim{};
    void insert(int x, int t) {
        for (int i = kD - 1; i >= 0; i--)
            if (x >> i & 1) {
                if (!bas[i]) {
                    bas[i] = x;
                    tim[i] = t;
                    return;
                }
                if (t > tim[i]) {
                    swap(x, bas[i]);
                    swap(t, tim[i]);
                }
                x ^= bas[i];
            }
    }
    bool query(int x) {
        for (int i = kD - 1; i >= 0; i--)
            chmin(x, x ^ bas[i]);
        return x == 0;
    }
};
```

5.13 Lucas

```
// comb(n, m) % M, M = p^k
// O(M) - O(log(n))
struct Lucas {
    const i64 p, M;
    vector<i64> f;
    Lucas(int p, int M) : p(p), M(M), f(M + 1) {
        f[0] = 1;
        for (int i = 1; i <= M; i++) {
            f[i] = f[i - 1] * (i % p == 0 ? 1 : i) % M;
        }
    }
    i64 CountFact(i64 n) {
        i64 c = 0;
        while (n) c += (n /= p);
        return c;
    }
    // (n! without factor p) % p^k
    i64 ModFact(i64 n) {
        i64 r = 1;
        while (n) {
            r = r * power(f[M], n / M % 2, M) % M * f[n % M] % M;
            n /= p;
        }
        return r;
    }
    i64 ModComb(i64 n, i64 m) {
        if (m < 0 or n < m) return 0;
        i64 c = CountFact(n) - CountFact(m) - CountFact(n - m);
        i64 r = ModFact(n) * power(ModFact(m), M / p * (p - 1) - 1, M) % M
```

```
        * power(ModFact(n - m), M / p * (p - 1) - 1, M) % M;
        return r * power(p, c, M) % M;
    }
};
```

5.14 Min25 Sieve

```
// Prefix Sums of Multiplicative Functions
// O(N^0.75 / log N)
// calc f(1) + ... + f(N)
// where f is multiplicative function
// construct completely multiplicative functions
// g_i s.t. for all prime x, f(x) = sigma c_i * g_i(x)
// def gsum(x) = g(1) + ... + g(x)
// call apply(g_i, gsum_i, c_i) and call work(f)
struct Min25 {
    const i64 N, sqrtN;
    vector<i64> Q;
    vector<i64> Fp, S;
    int id(i64 x) { return x <= sqrtN ? Q.size() - x : N / x - 1; }
    Min25(i64 N) : N(N), sqrtN(isqrt(N)) {
        // sieve(sqrtN);
        for (i64 l = 1, r; l <= N; l = r + 1) {
            Q.push_back(N / l);
            r = N / (N / l);
        }
        Fp.assign(Q.size(), 0);
        S.assign(Q.size(), 0);
    }
    void apply(const auto &f, const auto &fsum, i64 coef) {
        vector<i64> F(Q.size());
        for (int i = 0; i < Q.size(); i++) {
            F[i] = fsum(Q[i]) - 1;
        }
        for (i64 p : primes) {
            auto t = F[id(p - 1)];
            for (int i = 0; i < Q.size(); i++) {
                if (Q[i] < p * p) {
                    break;
                }
                F[i] -= (F[id(Q[i] / p)] - t) * f(p);
            }
        }
        for (int i = 0; i < Q.size(); i++) {
            Fp[i] += F[i] * coef;
        }
    }
    i64 work(const auto &f) {
        S = Fp;
        for (i64 p : primes | views::reverse) {
            i64 t = Fp[id(p)];
            for (int i = 0; i < Q.size(); i++) {
                if (Q[i] < p * p) {
                    break;
                }
                for (i64 pw = p; pw * p <= Q[i]; pw *= p) {
                    S[i] += (S[id(Q[i] / pw)] - t) * f(p, pw);
                    S[i] += f(p, pw * p);
                }
            }
        }
        for (int i = 0; i < Q.size(); i++) {
            S[i]++;
        }
        return S[0];
    }
};
```

5.15 Berlekamp Massey

```
template<int P>
vector<int> BerlekampMassey(vector<int> x) {
    vector<int> cur, ls;
    int lf = 0, ld = 0;
    for (int i = 0; i < (int)x.size(); ++i) {
        int t = 0;
        for (int j = 0; j < (int)cur.size(); ++j)
            (t += 1LL * cur[j] * x[i - j - 1] % P) %= P;
        if (t == x[i]) continue;
        if (cur.empty()) {
            cur.resize(i + 1);
```

```

    lf = i, ld = (t + P - x[i]) % P;
    continue;
}
int k = 1LL * fpow(ld, P - 2, P) * (t + P - x[i]) % P;
;
vector<int> c(i - lf - 1);
c.push_back(k);
for (int j = 0; j < (int)ls.size(); ++j)
    c.push_back(1LL * k * (P - ls[j]) % P);
if (c.size() < cur.size()) c.resize(cur.size());
for (int j = 0; j < (int)cur.size(); ++j)
    c[j] = (c[j] + cur[j]) % P;
if (i - lf + (int)ls.size() >= (int)cur.size()) {
    ls = cur, lf = i;
    ld = (t + P - x[i]) % P;
}
cur = c;
return cur;
}

```

5.16 Gauss Elimination

```

double Gauss(vector<vector<double>> &d) {
    int n = d.size(), m = d[0].size();
    double det = 1;
    for (int i = 0; i < m; ++i) {
        int p = -1;
        for (int j = i; j < n; ++j) {
            if (fabs(d[j][i]) < kEps) continue;
            if (p == -1 || fabs(d[j][i]) > fabs(d[p][i])) p = j;
        }
        if (p == -1) continue;
        if (p != i) det *= -1;
        for (int j = 0; j < m; ++j) swap(d[p][j], d[i][j]);
        for (int j = 0; j < n; ++j) {
            if (i == j) continue;
            double z = d[j][i] / d[i][i];
            for (int k = 0; k < m; ++k) d[j][k] -= z * d[i][k];
        }
    }
    for (int i = 0; i < n; ++i) det *= d[i][i];
    return det;
}

```

5.17 Linear Equation

```

void linear_equation(vector<vector<double>> &d, vector<
    double> &aug, vector<double> &sol) {
    int n = d.size(), m = d[0].size();
    vector<int> r(n), c(m);
    iota(r.begin(), r.end(), 0);
    iota(c.begin(), c.end(), 0);
    for (int i = 0; i < m; ++i) {
        int p = -1, z = -1;
        for (int j = i; j < n; ++j) {
            for (int k = i; k < m; ++k) {
                if (fabs(d[r[j]][c[k]] < eps) continue;
                if (p == -1 || fabs(d[r[j]][c[k]] > fabs(d[r[p]
                    ][c[z]])) p = j, z = k;
            }
        }
        if (p == -1) continue;
        swap(r[p], r[i]), swap(c[z], c[i]);
        for (int j = 0; j < n; ++j) {
            if (i == j) continue;
            double z = d[r[j]][c[i]] / d[r[i]][c[i]];
            for (int k = 0; k < m; ++k) d[r[j]][c[k]] -= z *
                d[r[i]][c[k]];
            aug[r[j]] -= z * aug[r[i]];
        }
    }
    vector<vector<double>> fd(n, vector<double>(m));
    vector<double> faug(n), x(n);
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < m; ++j) fd[i][j] = d[r[i]][c[j]
            ];
        faug[i] = aug[r[i]];
    }
    d = fd, aug = faug;
    for (int i = n - 1; i >= 0; --i) {
        double p = 0.0;
        for (int j = i + 1; j < n; ++j) p += d[i][j] * x[j]
            ];
    }
}

```

```

    x[i] = (aug[i] - p) / d[i][i];
}
for (int i = 0; i < n; ++i) sol[c[i]] = x[i];
}

```

5.18 LinearRec

```

template <int P>
int LinearRec(const vector<int> &s, const vector<int> &
    coeff, int k) {
    int n = s.size();
    auto Combine = [&](const auto &a, const auto &b) {
        vector<int> res(n * 2 + 1);
        for (int i = 0; i <= n; ++i) {
            for (int j = 0; j <= n; ++j)
                (res[i + j] += 1LL * a[i] * b[j] % P) %= P;
        }
        for (int i = 2 * n; i > n; --i) {
            for (int j = 0; j < n; ++j)
                (res[i - 1 - j] += 1LL * res[i] * coeff[j] % P)
                    %= P;
        }
        res.resize(n + 1);
        return res;
    };
    vector<int> p(n + 1), e(n + 1);
    p[0] = e[1] = 1;
    for (; k > 0; k >= 1) {
        if (k & 1) p = Combine(p, e);
        e = Combine(e, e);
    }
    int res = 0;
    for (int i = 0; i < n; ++i) (res += 1LL * p[i + 1] *
        s[i] % P) %= P;
    return res;
}

```

5.19 SubsetConv

```

vector<i64> SubsetConv(vector<i64> f, vector<i64> g) {
    const int n = f.size();
    const int U = __lg(n) + 1;
    vector F(U, vector<i64>(n));
    auto G = F, H = F;
    for (int i = 0; i < n; i++) {
        F[popcount<u64>(i)][i] = f[i];
        G[popcount<u64>(i)][i] = g[i];
    }
    for (int i = 0; i < U; i++) {
        FWT(F[i], ORop);
        FWT(G[i], ORop);
    }
    for (int i = 0; i < U; i++)
        for (int j = 0; j <= i; j++)
            for (int k = 0; k < n; k++)
                H[i][k] = (H[i][k] + F[i - j][k] * G[j][k]) %
                    mod;
    for (int i = 0; i < U; i++) FWT(H[i], ORinv);
    for (int i = 0; i < n; i++) f[i] = H[popcount<u64>(i)
        ][i];
    return f;
}

```

5.20 SqrtMod

```

// 0 <= x < p, s.t. x^2 mod p = n
int SqrtMod(int n, int P) {
    if (P == 2 || n == 0) return n;
    if (power(n, (P - 1) / 2, P) != 1) return -1;
    mt19937 rng(12312);
    i64 z = 0, w;
    while (power(w = (z * z - n + P) % P, (P - 1) / 2, P)
        != P - 1)
        z = rng() % P;
    const auto M = [P, w](auto &u, auto &v) {
        return pair{
            (u.ff * v.ff + u.ss * v.ss % P * w) % P,
            (u.ff * v.ss + u.ss * v.ff) % P
        };
    };
    pair<i64, i64> r{1, 0}, e{z, 1};
    for (int w = (P + 1) / 2; w; w >= 1, e = M(e, e))
        if (w & 1) r = M(r, e);
    return r.ff;
}

```

5.21 DiscreteLog

```
template<class T>
T BSGS(T x, T y, T M) {
    //  $x^? \equiv y \pmod M$ 
    T t = 1, c = 0, g = 1;
    for (T M_ = M; M_ > 0; M_ >>= 1) g = g * x % M;
    for (g = gcd(g, M); t % g != 0; ++c) {
        if (t == y) return c;
        t = t * x % M;
    }
    if (y % g != 0) return -1;
    t /= g, y /= g, M /= g;
    T h = 0, gs = 1;
    for (; h * h < M; ++h) gs = gs * x % M;
    unordered_map<T, T> bs;
    for (T s = 0; s < h; bs[y] = ++s) y = y * x % M;
    for (T s = 0; s < M; s += h) {
        t = t * gs % M;
        if (bs.count(t)) return c + s + h - bs[t];
    }
    return -1;
}
```

5.22 FloorSum

```
//  $\sigma_{0 \sim n-1} (a * i + b) / m$ 
i64 floorSum(i64 n, i64 m, i64 a, i64 b) {
    u64 ans = 0;
    if (a < 0) {
        u64 a2 = (a % m + m) % m;
        ans -= 1ULL * n * (n - 1) / 2 * ((a2 - a) / m);
        a = a2;
    }
    if (b < 0) {
        u64 b2 = (b % m + m) % m;
        ans -= 1ULL * n * ((b2 - b) / m);
        b = b2;
    }
    while (true) {
        if (a >= m) {
            ans += n * (n - 1) / 2 * (a / m);
            a %= m;
        }
        if (b >= m) {
            ans += n * (b / m);
            b %= m;
        }
        u64 y_max = a * n + b;
        if (y_max < m) break;
        n = y_max / m;
        b = y_max % m;
        swap(m, a);
    }
    return ans;
}
```

5.23 Linear Programming Simplex

```
// max{cx} subject to {Ax<=b, x>=0}
// n: constraints, m: vars !!!
// x[] is the optimal solution vector
// usage :
// x = simplex(A, b, c); (A <= 100 x 100)
vector<double> simplex(
    const vector<vector<double>>> &a,
    const vector<double> &b,
    const vector<double> &c) {

    int n = (int)a.size(), m = (int)a[0].size() + 1;
    vector val(n + 2, vector<double>(m + 1));
    vector<int> idx(n + m);
    iota(all(idx), 0);
    int r = n, s = m - 1;
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < m - 1; ++j)
            val[i][j] = -a[i][j];
        val[i][m - 1] = 1;
        val[i][m] = b[i];
        if (val[r][m] > val[i][m])
            r = i;
    }
    copy(all(c), val[n].begin());
    val[n + 1][m - 1] = -1;
```

```
for (double num; ; ) {
    if (r < n) {
        swap(idx[s], idx[r + m]);
        val[r][s] = 1 / val[r][s];
        for (int j = 0; j <= m; ++j) if (j != s)
            val[r][j] *= -val[r][s];
        for (int i = 0; i <= n + 1; ++i) if (i != r) {
            for (int j = 0; j <= m; ++j) if (j != s)
                val[i][j] += val[r][j] * val[i][s];
            val[i][s] *= val[r][s];
        }
    }
    r = s = -1;
    for (int j = 0; j < m; ++j)
        if (s < 0 || idx[s] > idx[j])
            if (val[n + 1][j] > eps || val[n + 1][j] > -eps
                && val[n][j] > eps)
                s = j;
    if (s < 0) break;
    for (int i = 0; i < n; ++i) if (val[i][s] < -eps) {
        if (r < 0
            || (num = val[r][m] / val[r][s] - val[i][m] /
                val[i][s]) < -eps
            || num < eps && idx[r + m] > idx[i + m])
            r = i;
    }
    if (r < 0) {
        // Solution is unbounded.
        return vector<double>{};
    }
}
if (val[n + 1][m] < -eps) {
    // No solution.
    return vector<double>{};
}
vector<double> x(m - 1);
for (int i = m; i < n + m; ++i)
    if (idx[i] < m - 1)
        x[idx[i]] = val[i - m][m];
return x;
}
```

5.24 Lagrange Interpolation

```
struct Lagrange {
    int deg{};
    vector<i64> C;
    Lagrange(const vector<i64> &P) {
        deg = P.size() - 1;
        C.assign(deg + 1, 0);
        for (int i = 0; i <= deg; i++) {
            i64 q = comb(-i) * comb(i - deg) % mod;
            if ((deg - i) % 2 == 1) {
                q = mod - q;
            }
            C[i] = P[i] * q % mod;
        }
    }
}
i64 operator()(i64 x) { //  $0 \leq x < \text{mod}$ 
    if (0 <= x and x <= deg) {
        i64 ans = comb(x) * comb(deg - x) % mod;
        if ((deg - x) % 2 == 1) {
            ans = (mod - ans);
        }
        return ans * C[x] % mod;
    }
    vector<i64> pre(deg + 1), suf(deg + 1);
    for (int i = 0; i <= deg; i++) {
        pre[i] = (x - i);
        if (i)
            pre[i] = pre[i] * pre[i - 1] % mod;
    }
    for (int i = deg; i >= 0; i--) {
        suf[i] = (x - i);
        if (i < deg)
            suf[i] = suf[i] * suf[i + 1] % mod;
    }
    i64 ans = 0;
    for (int i = 0; i <= deg; i++) {
        ans += (i == 0 ? 1 : pre[i - 1]) * (i == deg ? 1
            : suf[i + 1]) % mod * C[i];
    }
}
```

```

    ans %= mod;
}
if (ans < 0) ans += mod;
return ans;
}
};

```

6 Geometry

6.1 Point

```

using numbers::pi;
template<class T> inline constexpr T eps =
    numeric_limits<T>::epsilon() * 1E6;
using Real = long double;
struct Pt {
    Real x{}, y{};
    Pt operator+(Pt a) const { return {x + a.x, y + a.y}; }
    Pt operator-(Pt a) const { return {x - a.x, y - a.y}; }
    Pt operator*(Real k) const { return {x * k, y * k}; }
    Pt operator/(Real k) const { return {x / k, y / k}; }
    Real operator*(Pt a) const { return x * a.x + y * a.y; }
    Real operator^(Pt a) const { return x * a.y - y * a.x; }
    auto operator<=>(const Pt&) const = default;
    bool operator==(const Pt&) const = default;
};
int sgn(Real x) { return (x > -eps<Real>) - (x < eps<Real>); }
Real ori(Pt a, Pt b, Pt c) { return (b - a) ^ (c - a); }
bool argcmp(const Pt &a, const Pt &b) { // arg(a) < arg(b)
    int f = (Pt{a.y, -a.x} > Pt{} ? 1 : -1) * (a != Pt{});
    int g = (Pt{b.y, -b.x} > Pt{} ? 1 : -1) * (b != Pt{});
    return f == g ? (a ^ b) > 0 : f < g;
}
Pt rotate(Pt u) { return {-u.y, u.x}; }
Real abs2(Pt a) { return a * a; }
// floating point only
Pt rotate(Pt u, Real a) {
    Pt v{sinl(a), cosl(a)};
    return {u ^ v, u * v};
}
Real abs(Pt a) { return sqrtl(a * a); }
Real arg(Pt x) { return atan2l(x.y, x.x); }
Pt unit(Pt x) { return x / abs(x); }

```

6.2 Line

```

struct Line {
    Pt a, b;
    Pt dir() const { return b - a; }
};
int PtSide(Pt p, Line L) {
    return sgn(ori(L.a, L.b, p)); // for int
    return sgn(ori(L.a, L.b, p) / abs(L.a - L.b));
}
bool PtOnSeg(Pt p, Line L) {
    return PtSide(p, L) == 0 and sgn((p - L.a) * (p - L.b)) <= 0;
}
Pt proj(Pt p, Line l) {
    Pt dir = unit(l.b - l.a);
    return l.a + dir * (dir * (p - l.a));
}

```

6.3 Circle

```

struct Cir {
    Pt o;
    double r;
};
bool disjunct(const Cir &a, const Cir &b) {
    return sgn(abs(a.o - b.o) - a.r - b.r) >= 0;
}
bool contain(const Cir &a, const Cir &b) {
    return sgn(a.r - b.r - abs(a.o - b.o)) >= 0;
}

```

6.4 Point to Segment Distance

```

double PtSegDist(Pt p, Line l) {
    double ans = min(abs(p - l.a), abs(p - l.b));
    if (sgn(abs(l.a - l.b)) == 0) return ans;
    if (sgn((l.a - l.b) * (p - l.b)) < 0) return ans;
    if (sgn((l.b - l.a) * (p - l.a)) < 0) return ans;
    return min(ans, abs(ori(p, l.a, l.b)) / abs(l.a - l.b));
}
double SegDist(Line l, Line m) {
    return PtSegDist({0, 0}, {l.a - m.a, l.b - m.b});
}

```

6.5 Point in Polygon

```

int inPoly(Pt p, const vector<Pt> &P) {
    const int n = P.size();
    int cnt = 0;
    for (int i = 0; i < n; i++) {
        Pt a = P[i], b = P[(i + 1) % n];
        if (PtOnSeg(p, {a, b})) return 1; // on edge
        if ((sgn(a.y - p.y) == 1) ^ (sgn(b.y - p.y) == 1))
            cnt += sgn(ori(a, b, p));
    }
    return cnt == 0 ? 0 : 2; // out, in
}

```

6.6 Intersection of Lines

```

bool isInter(Line l, Line m) {
    if (PtOnSeg(m.a, l) or PtOnSeg(m.b, l) or
        PtOnSeg(l.a, m) or PtOnSeg(l.b, m))
        return true;
    return PtSide(m.a, l) * PtSide(m.b, l) < 0 and
        PtSide(l.a, m) * PtSide(l.b, m) < 0;
}
Pt LineInter(Line l, Line m) {
    double s = ori(m.a, m.b, l.a), t = ori(m.a, m.b, l.b);
    return (l.b * s - l.a * t) / (s - t);
}
bool strictInter(Line l, Line m) {
    int la = PtSide(m.a, l);
    int lb = PtSide(m.b, l);
    int ma = PtSide(l.a, m);
    int mb = PtSide(l.b, m);
    if (la == 0 and lb == 0) return false;
    return la * lb < 0 and ma * mb < 0;
}

```

6.7 Intersection of Circle and Line

```

vector<Pt> CircleLineInter(Cir c, Line l) {
    Pt H = proj(c.o, l);
    Pt dir = unit(l.b - l.a);
    double h = abs(H - c.o);
    if (sgn(h - c.r) > 0) return {};
    double d = sqrt(max((double)0., c.r * c.r - h * h));
    if (sgn(d) == 0) return {H};
    return {H - dir * d, H + dir * d};
    // Counterclockwise
}

```

6.8 Intersection of Circles

```

vector<Pt> CircleInter(Cir a, Cir b) {
    double d2 = abs2(a.o - b.o), d = sqrt(d2);
    if (d < max(a.r, b.r) - min(a.r, b.r) || d > a.r + b.r)
        return {};
    Pt u = (a.o + b.o) / 2 + (a.o - b.o) * ((b.r * b.r - a.r * a.r) / (2 * d2));
    double A = sqrt((a.r + b.r + d) * (a.r - b.r + d) * (a.r + b.r - d) * (-a.r + b.r + d));
    Pt v = rotate(b.o - a.o) * A / (2 * d2);
    if (sgn(v.x) == 0 and sgn(v.y) == 0) return {u};
    return {u - v, u + v}; // counter clockwise of a
}

```

6.9 Area of Circle and Polygon

```

double CirclePoly(Cir C, const vector<Pt> &P) {
    auto arg = [&](Pt p, Pt q) { return atan2(p ^ q, p * q); };
    double r2 = C.r * C.r / 2;
    auto tri = [&](Pt p, Pt q) {
        Pt d = q - p;
    };
}

```



```

    auto a = (d * p) / abs2(d), b = (abs2(p) - C.r * C.r) / abs2(d);
    auto det = a * a - b;
    if (det <= 0) return arg(p, q) * r2;
    auto s = max(0., -a - sqrt(det)), t = min(1., -a + sqrt(det));
    if (t < 0 or 1 <= s) return arg(p, q) * r2;
    Pt u = p + d * s, v = p + d * t;
    return arg(p, u) * r2 + (u ^ v) / 2 + arg(v, q) * r2;
};
double sum = 0.0;
for (int i = 0; i < P.size(); i++)
    sum += tri(P[i] - C.o, P[(i + 1) % P.size()] - C.o);
return sum;
}

```

6.10 Area of Sector

```

//  $\square AOB * r^2 / 2$ 
double Sector(Pt a, Pt b, double r) {
    double theta = atan2(a.y, a.x) - atan2(b.y, b.x);
    while (theta <= 0) theta += 2 * pi;
    while (theta >= 2 * pi) theta -= 2 * pi;
    theta = min(theta, 2 * pi - theta);
    return r * r * theta / 2;
}

```

6.11 Union of Polygons

```

// Area[i] : area covered by at least i polygon
vector<double> PolyUnion(const vector<vector<Pt>> &P) {
    const int n = P.size();
    vector<double> Area(n + 1);
    vector<Line> Ls;
    for (int i = 0; i < n; i++)
        for (int j = 0; j < P[i].size(); j++)
            Ls.push_back({P[i][j], P[i][(j + 1) % P[i].size()]});
    auto cmp = [&](Line &l, Line &r) {
        Pt u = l.b - l.a, v = r.b - r.a;
        if (argcmp(u, v)) return true;
        if (argcmp(v, u)) return false;
        return PtSide(l.a, r) < 0;
    };
    sort(all(Ls), cmp);
    for (int l = 0, r = 0; l < Ls.size(); l = r) {
        while (r < Ls.size() and !cmp(Ls[l], Ls[r])) r++;
        Line L = Ls[l];
        vector<pair<Pt, int>> event;
        for (auto [c, d] : Ls) {
            if (sgn((L.a - L.b) ^ (c - d)) != 0) {
                int s1 = PtSide(c, L) == 1;
                int s2 = PtSide(d, L) == 1;
                if (s1 ^ s2) event.emplace_back(LineInter(L, {c, d}), s1 ? 1 : -1);
            } else if (PtSide(c, L) == 0 and sgn((L.a - L.b) ^ (c - d)) > 0) {
                event.emplace_back(c, 2);
                event.emplace_back(d, -2);
            }
        }
        sort(all(event), [&](auto i, auto j) {
            return (L.a - i.ff) * (L.a - L.b) < (L.a - j.ff) * (L.a - L.b);
        });
        int cov = 0, tag = 0;
        Pt lst{0, 0};
        for (auto [p, s] : event) {
            if (cov >= tag) {
                Area[cov] += lst ^ p;
                Area[cov - tag] -= lst ^ p;
            }
            if (abs(s) == 1) cov += s;
            else tag += s / 2;
            lst = p;
        }
    }
    for (int i = n - 1; i >= 0; i--) Area[i] += Area[i + 1];
    for (int i = 1; i <= n; i++) Area[i] /= 2;
    return Area;
}

```

6.12 Union of Circles

```

// Area[i] : area covered by at least i circle
vector<double> CircleUnion(const vector<Cir> &C) {
    const int n = C.size();
    vector<double> Area(n + 1);
    auto check = [&](int i, int j) {
        if (!contain(C[i], C[j]))
            return false;
        return sgn(C[i].r - C[j].r) > 0 or (sgn(C[i].r - C[j].r) == 0 and i < j);
    };
    struct Teve {
        double ang; int add; Pt p;
        bool operator<(const Teve &b) { return ang < b.ang; }
    };
    auto ang = [&](Pt p) { return atan2(p.y, p.x); };
    for (int i = 0; i < n; i++) {
        int cov = 1;
        vector<Teve> event;
        for (int j = 0; j < n; j++) if (i != j) {
            if (check(j, i)) cov++;
            else if (!check(i, j) and !disjunct(C[i], C[j])) {
                auto I = CircleInter(C[i], C[j]);
                assert(I.size() == 2);
                double a1 = ang(I[0] - C[i].o), a2 = ang(I[1] - C[i].o);
                event.push_back({a1, 1, I[0]});
                event.push_back({a2, -1, I[1]});
                if (a1 > a2) cov++;
            }
        }
        if (event.empty()) {
            Area[cov] += pi * C[i].r * C[i].r;
            continue;
        }
        sort(all(event));
        event.push_back(event[0]);
        for (int j = 0; j + 1 < event.size(); j++) {
            cov += event[j].add;
            Area[cov] += (event[j].p ^ event[j + 1].p) / 2.;
            double theta = event[j + 1].ang - event[j].ang;
            if (theta < 0) theta += 2 * pi;
            Area[cov] += (theta - sin(theta)) * C[i].r * C[i].r / 2.;
        }
    }
    return Area;
}

```

6.13 TangentLines of Circle and Point

```

vector<Line> CircleTangent(Cir c, Pt p) {
    vector<Line> z;
    double d = abs(p - c.o);
    if (sgn(d - c.r) == 0) {
        Pt i = rotate(p - c.o);
        z.push_back({p, p + i});
    } else if (d > c.r) {
        double o = acos(c.r / d);
        Pt i = unit(p - c.o);
        Pt j = rotate(i, o) * c.r;
        Pt k = rotate(i, -o) * c.r;
        z.push_back({c.o + j, p});
        z.push_back({c.o + k, p});
    }
    return z;
}

```

6.14 TangentLines of Circles

```

vector<Line> CircleTangent(Cir c1, Cir c2, int sign1) {
    // sign1 = 1 for outer tang, -1 for inner tang
    vector<Line> ret;
    double d_sq = abs2(c1.o - c2.o);
    if (sgn(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 (sgn(p1.x - p2.x) == 0 && sgn(p1.y - p2.y) == 0)
        p2 = p1 + rotate(c2.o - c1.o);
    ret.push_back({p1, p2});
}
return ret;
}

```

6.15 Convex Hull

```

vector<Pt> Hull(vector<Pt> P) {
    sort(all(P));
    P.erase(unique(all(P)), P.end());
    if (P.size() <= 1) return P;
    P.insert(P.end(), P.rbegin() + 1, P.rend());
    vector<Pt> stk;
    for (auto p : P) {
        auto it = stk.rbegin();
        while (stk.rend() - it >= 2 and \
            ori(*next(it), *it, p) <= 0 and \
            (*next(it) < *it) == (*it < p)) {
            it++;
        }
        stk.resize(stk.rend() - it);
        stk.push_back(p);
    }
    stk.pop_back();
    return stk;
}

```

6.16 Convex Hull trick

```

struct Convex {
    int n;
    vector<Pt> A, V, L, U;
    Convex(const vector<Pt> &A) : A(A), n(A.size()) {
        // n >= 3
        auto it = max_element(all(A));
        L.assign(A.begin(), it + 1);
        U.assign(it, A.end()), U.push_back(A[0]);
        for (int i = 0; i < n; i++) {
            V.push_back(A[(i + 1) % n] - A[i]);
        }
    }
    int inside(Pt p, const vector<Pt> &h, auto f) {
        auto it = lower_bound(all(h), p, f);
        if (it == h.end()) return 0;
        if (it == h.begin()) return p == *it;
        return 1 - sgn(ori(*prev(it), p, *it));
    }
    // 0: out, 1: on, 2: in
    int inside(Pt p) {
        return min(inside(p, L, less{}), inside(p, U,
            greater{}));
    }
    static bool cmp(Pt a, Pt b) { return sgn(a ^ b) > 0; }
    // A[i] is a far/closer tangent point
    int tangent(Pt v, bool close = true) {
        assert(v != Pt{});
        auto l = V.begin(), r = V.begin() + L.size() - 1;
        if (v < Pt{}) l = r, r = V.end();
        if (close) return (lower_bound(l, r, v, cmp) - V.
            begin()) % n;
        return (upper_bound(l, r, v, cmp) - V.begin()) % n;
    }
    // closer tangent point
    array<int, 2> tangent2(Pt p) {
        array<int, 2> t{-1, -1};
        if (inside(p) == 2) return t;
        if (auto it = lower_bound(all(L), p); it != L.end()
            and p == *it) {
            int s = it - L.begin();
            return {(s + 1) % n, (s - 1 + n) % n};
        }
        if (auto it = lower_bound(all(U), p, greater{}); it
            != U.end() and p == *it) {
            int s = it - U.begin() + L.size() - 1;
            return {(s + 1) % n, (s - 1 + n) % n};
        }
    }
}

```

```

for (int i = 0; i != t[0]; i = tangent((A[t[0] = i]
    - p), 0));
for (int i = 0; i != t[1]; i = tangent((p - A[t[1]
    = i]), 1));
return t;
}
int find(int l, int r, Line L) {
    if (r < l) r += n;
    int s = PtSide(A[l % n], L);
    return *ranges::partition_point(views::iota(l, r),
        [&](int m) {
            return PtSide(A[m % n], L) == s;
        }) - 1;
};
// Line A_x A_x+1 interset with L
vector<int> intersect(Line L) {
    int l = tangent(L.a - L.b), r = tangent(L.b - L.a);
    if (PtSide(A[l], L) * PtSide(A[r], L) >= 0) return
        {};
    return {find(l, r, L) % n, find(r, l, L) % n};
}
};

```

6.17 Dynamic Convex Hull

```

template<class T, class Comp = less<T>>
struct DynamicHull {
    set<T, Comp> H;
    void insert(T p) {
        if (inside(p)) return;
        auto it = H.insert(p).ff;
        while (it != H.begin() and prev(it) != H.begin() \
            and ori(*prev(it), 2), *prev(it), *it) <= 0) {
            it = H.erase(--it);
        }
        while (it != --H.end() and next(it) != --H.end() \
            and ori(*it, *next(it), *next(it), 2)) <= 0) {
            it = --H.erase(++it);
        }
    }
    int inside(T p) { // 0: out, 1: on, 2: in
        auto it = H.lower_bound(p);
        if (it == H.end()) return 0;
        if (it == H.begin()) return p == *it;
        return 1 - sgn(ori(*prev(it), p, *it));
    }
};
// DynamicHull<Pt> D;
// DynamicHull<Pt, greater<>> U;
// D.inside(p) and U.inside(p)

```

6.18 Half Plane Intersection

```

bool cover(Line L, Line P, Line Q) {
    // return PtSide(LineInter(P, Q), L) <= 0; for double
    i128 u = (Q.a - P.a) ^ Q.dir();
    i128 v = P.dir() ^ Q.dir();
    i128 x = P.dir().x * u + (P.a - L.a).x * v;
    i128 y = P.dir().y * u + (P.a - L.a).y * v;
    return sgn(x * L.dir().y - y * L.dir().x) * sgn(v) >=
        0;
}
vector<Line> HPI(vector<Line> P) {
    sort(all(P), [&](Line l, Line m) {
        if (argcmp(l.dir(), m.dir()) return true;
        if (argcmp(m.dir(), l.dir()) return false;
        return ori(m.a, m.b, l.a) > 0;
    });
    int n = P.size(), l = 0, r = -1;
    for (int i = 0; i < n; i++) {
        if (i and !argcmp(P[i - 1].dir(), P[i].dir()))
            continue;
        while (l < r and cover(P[i], P[r - 1], P[r])) r--;
        while (l < r and cover(P[i], P[l], P[l + 1])) l++;
        P[++r] = P[i];
    }
    while (l < r and cover(P[l], P[r - 1], P[r])) r--;
    while (l < r and cover(P[r], P[l], P[l + 1])) l++;
    if (r - l <= 1 or !argcmp(P[l].dir(), P[r].dir()))
        return {}; // empty
    if (cover(P[l + 1], P[l], P[r]))
        return {}; // infinity
    return vector(P.begin() + l, P.begin() + r + 1);
}

```

6.19 Minkowski

```
// P, Q, R(return) are counterclockwise order convex
// polygon
vector<Pt> Minkowski(vector<Pt> P, vector<Pt> Q) {
    assert(P.size() >= 2 and Q.size() >= 2);
    auto cmp = [&](Pt a, Pt b) {
        return Pt{a.y, a.x} < Pt{b.y, b.x};
    };
    auto reorder = [&](auto &R) {
        rotate(R.begin(), min_element(all(R), cmp), R.end());
        R.push_back(R[0]), R.push_back(R[1]);
    };
    const int n = P.size(), m = Q.size();
    reorder(P), reorder(Q);
    vector<Pt> R;
    for (int i = 0, j = 0, s; i < n or j < m; ) {
        R.push_back(P[i] + Q[j]);
        s = sgn((P[i + 1] - P[i]) ^ (Q[j + 1] - Q[j]));
        if (s >= 0) i++;
        if (s <= 0) j++;
    }
    return R;
}
```

6.20 Minimal Enclosing Circle

```
Pt Center(Pt a, Pt b, Pt c) {
    Pt x = (a + b) / 2;
    Pt y = (b + c) / 2;
    return LineInter({x, x + rotate(b - a)}, {y, y +
        rotate(c - b)});
}
Cir MEC(vector<Pt> P) {
    mt19937 rng(time(0));
    shuffle(all(P), rng);
    Cir C{};
    for (int i = 0; i < P.size(); i++) {
        if (C.inside(P[i])) continue;
        C = {P[i], 0};
        for (int j = 0; j < i; j++) {
            if (C.inside(P[j])) continue;
            C = {(P[i] + P[j]) / 2, abs(P[i] - P[j]) / 2};
            for (int k = 0; k < j; k++) {
                if (C.inside(P[k])) continue;
                C.o = Center(P[i], P[j], P[k]);
                C.r = abs(C.o - P[i]);
            }
        }
    }
    return C;
}
```

6.21 Point In Circumcircle

```
// p[0], p[1], p[2] should be counterclockwise order
int inCC(const array<Pt, 3> &p, Pt a) {
    i128 det = 0;
    for (int i = 0; i < 3; i++)
        det += i128(abs2(p[i]) - abs2(a)) * ori(a, p[(i + 1) % 3], p[(i + 2) % 3]);
    return (det > 0) - (det < 0); // in:1, on:0, out:-1
}
```

6.22 Delaunay Triangulation

```
bool inCC(const array<Pt, 3> &p, Pt a) {
    i128 det = 0;
    for (int i = 0; i < 3; i++)
        det += i128(abs2(p[i]) - abs2(a)) * ori(a, p[(i + 1) % 3], p[(i + 2) % 3]);
    return det > 0;
}
struct Edge {
    int id;
    list<Edge>::iterator rit;
};
vector<list<Edge>> Delaunay(const vector<Pt> &P) {
    assert(is_sorted(all(P))); // need sorted before!
    const int n = P.size();
    vector<list<Edge>> E(n);
    auto addEdge = [&](int u, int v, auto a, auto b) {
        a = E[u].insert(a, {v});
        b = E[v].insert(b, {u});
    };
}
```

```
return array{b->rit = a, a->rit = b};
};
auto divide = [&](auto &&self, int l, int r) -> int {
    if (r - l <= 1) return l;
    int m = (l + r) / 2;
    array<int, 2> t{self(self, l, m), self(self, m, r)};
    int w = t[P[t[1]].y < P[t[0]].y];
    auto low = [&](int s) {
        for (Edge e : E[t[s]]) {
            if (ori(P[t[1]], P[t[0]], P[e.id]) > 0 or
                PtOnSeg(P[e.id], {P[t[0]], P[t[1]]})) {
                t[s] = e.id;
                return true;
            }
        }
        return false;
    };
    while (low(0) or low(1));
    array its = addEdge(t[0], t[1], E[t[0]].begin(), E[t[1]].end());
    while (true) {
        Line L{P[t[0]], P[t[1]]};
        auto cand = [&](int s) -> optional<list<Edge>::
            iterator> {
            auto nxt = [&](auto it) {
                if (s == 0) return (++it == E[t[0]].end() ? E[t[0]].begin() : it);
                return --(it == E[t[1]].begin() ? E[t[1]].end() : it);
            };
            if (E[t[s]].empty()) return {};
            auto lst = nxt(its[s]), it = nxt(lst);
            while (PtSide(P[it->id], L) > 0 and inCC({L.a, L.b, P[lst->id]}, P[it->id])) {
                E[t[s ^ 1]].erase(lst->rit);
                E[t[s]].erase(lst);
                it = nxt(lst = it);
            }
            return PtSide(P[lst->id], L) > 0 ? optional{lst} : nullopt;
        };
        auto lc = cand(0), rc = cand(1);
        if (!lc and !rc) break;
        int sd = !lc or (rc and inCC({L.a, L.b, P[(lc->id)], P[(rc->id)]}));
        auto lst = *(sd ? rc : lc);
        t[sd] = lst->id;
        its[sd] = lst->rit;
        its = addEdge(t[0], t[1], ++its[0], its[1]);
    }
    return w;
};
divide(divide, 0, n);
return E;
};
```

6.23 Triangle Center

```
Pt TriangleCircumCenter(Pt a, Pt b, Pt c) {
    Pt res;
    double a1 = atan2(b.y - a.y, b.x - a.x) + pi / 2;
    double a2 = atan2(c.y - b.y, c.x - b.x) + pi / 2;
    double ax = (a.x + b.x) / 2;
    double ay = (a.y + b.y) / 2;
    double bx = (c.x + b.x) / 2;
    double by = (c.y + b.y) / 2;
    double r1 = (sin(a2) * (ax - bx) + cos(a2) * (by - ay)) / (sin(a1) * cos(a2) - sin(a2) * cos(a1));
    return Pt(ax + r1 * cos(a1), ay + r1 * sin(a1));
}
Pt TriangleMassCenter(Pt a, Pt b, Pt c) {
    return (a + b + c) / 3.0;
}
Pt TriangleOrthoCenter(Pt a, Pt b, Pt c) {
    return TriangleMassCenter(a, b, c) * 3.0 - TriangleCircumCenter(a, b, c) * 2.0;
}
Pt TriangleInnerCenter(Pt a, Pt b, Pt c) {
    Pt res;
    double la = abs(b - c);
    double lb = abs(a - c);
    double lc = abs(a - b);
}
```

```

res.x = (la * a.x + lb * b.x + lc * c.x) / (la + lb +
lc);
res.y = (la * a.y + lb * b.y + lc * c.y) / (la + lb +
lc);
return res;
}

```

7 Stringology

7.1 KMP

```

vector<int> buildFail(string s) {
    const int len = s.size();
    vector<int> f(len, -1);
    for (int i = 1, p = -1; i < len; i++) {
        while (~p and s[p + 1] != s[i]) p = f[p];
        if (s[p + 1] == s[i]) p++;
        f[i] = p;
    }
    return f;
}

```

7.2 Z-algorithm

```

vector<int> zalgo(string s) {
    if (s.empty()) return {};
    int len = s.size();
    vector<int> z(len);
    z[0] = len;
    for (int i = 1, l = 1, r = 1; i < len; i++) {
        z[i] = i < r ? min(z[i - l], r - i) : 0;
        while (i + z[i] < len and s[i + z[i]] == s[z[i]]) z[i]++;
        if (i + z[i] > r) l = i, r = i + z[i];
    }
    return z;
}

```

7.3 Manacher

```

vector<int> manacher(string_view s) {
    string p = "@#";
    for (char c : s) {
        p += c;
        p += '#';
    }
    p += '$';
    vector<int> dp(p.size());
    int mid = 0, r = 1;
    for (int i = 1; i < p.size() - 1; i++) {
        auto &k = dp[i];
        k = i < mid + r ? min(dp[mid * 2 - i], mid + r - i) : 0;
        while (p[i + k + 1] == p[i - k - 1]) k++;
        if (i + k > mid + r) mid = i, r = k;
    }
    return vector<int>(dp.begin() + 2, dp.end() - 2);
}

```

7.4 SuffixArray Simple

```

struct SuffixArray {
    int n;
    vector<int> suf, rk, S;
    SuffixArray(vector<int> _S) : S(_S) {
        n = S.size();
        suf.assign(n, 0);
        rk.assign(n * 2, -1);
        iota(all(suf), 0);
        for (int i = 0; i < n; i++) rk[i] = S[i];
        for (int k = 2; k < n + n; k *= 2) {
            auto cmp = [&](int a, int b) -> bool {
                return rk[a] == rk[b] ? (rk[a + k / 2] < rk[b + k / 2]) : (rk[a] < rk[b]);
            };
            sort(all(suf), cmp);
            auto tmp = rk;
            tmp[suf[0]] = 0;
            for (int i = 1; i < n; i++) {
                tmp[suf[i]] = tmp[suf[i - 1]] + cmp(suf[i - 1], suf[i]);
            }
            rk.swap(tmp);
        }
    }
};

```

7.5 SuffixArray SAIS C++20

```

auto sais(const auto &s) {
    const int n = (int)s.size(), 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); t[n - 1] = 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-- and !t[y]) sa[x[s[y] - 1]++] = y;
        for (auto x = c; int y : sa | views::reverse)
            if (y-- and 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] = int(lms.size());
        lms.push_back(sa[--x[s[i]]] = i);
    }
    induce(); vector<int> ns(lms.size());
    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(
                s.begin() + j, s.begin() + j + len,
                s.begin() + i, s.begin() + i + len
            );
        }
        j = i;
    }
    ranges::fill(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.
// lcp[i]: LCP of suffix sa[i] and suffix sa[i + 1].
struct Suffix {
    int n;
    vector<int> sa, rk, lcp;
    Suffix(const auto &s) : n(s.size()),
        lcp(n - 1), rk(n) {
        vector<int> t(n + 1); // t[n] = 0
        copy(all(s), t.begin()); // s shouldn't contain 0
        sa = sais(t); sa.erase(sa.begin());
        for (int i = 0; i < n; i++) rk[sa[i]] = i;
        for (int i = 0, h = 0; i < n; i++) {
            if (!rk[i]) { h = 0; continue; }
            for (int j = sa[rk[i] - 1];
                i + h < n and j + h < n
                and s[i + h] == s[j + h];) ++h;
            lcp[rk[i] - 1] = h ? h-- : 0;
        }
    }
};

```

7.6 Aho-Corasick

```

const int sigma = ;

struct Node {
    Node *ch[sigma]{};
    Node *fail{};
    bool end{};
} pool[i64(1E6)]{};

struct ACauto {
    int top;
    Node *root;
    ACauto() {
        top = 0;
        root = new (pool + top++) Node();
    }
    int add(string_view s) {
        auto p = root;

```

```

    for (char c : s) {
        c -= ;
        if (!p->ch[c]) {
            p->ch[c] = new (pool + top++) Node();
        }
        p = p->ch[c];
    }
    p->end = true;
    return p - pool;
}
vector<Node*> ord;
void build() {
    queue<Node*> que;
    root->fail = root;
    for (auto &p : root->ch) {
        if (p) {
            p->fail = root;
            que.push(p);
        } else {
            p = root;
        }
    }
    while (!que.empty()) {
        auto p = que.front();
        que.pop();
        ord.push_back(p);
        p->next = (p->fail->end ? p->fail : p->fail->next);
        for (int i = 0; i < sigma; i++) {
            if (p->ch[i]) {
                p->ch[i]->fail = p->fail->ch[i];
                que.push(p->ch[i]);
            } else {
                p->ch[i] = p->fail->ch[i];
            }
        }
    }
}
};

```

7.7 Palindromic Tree

// 迴文樹的每個節點代表一個迴文串
 // len[i] 表示第 i 個節點的長度
 // fail[i] 表示第 i 個節點的失配指針
 // fail[i] 是 i 的次長迴文後綴
 // dep[i] 表示第 i 個節點有幾個迴文後綴
 // nxt[i][c] 表示在節點 i 兩邊加上字元 c 得到的點
 // nxt 邊構成了兩顆分別以 odd 和 even 為根的向下的樹
 // len[odd] = -1, len[even] = 0
 // fail 邊構成了一顆以 odd 為根的向上的樹
 // fail[even] = odd
 // 0 ~ node size 是一個好的 dp 順序
 // walk 是構建迴文樹時 lst 經過的節點

```

struct PAM {
    vector<array<int, 26>> nxt;
    vector<int> fail, len, dep, walk;
    int odd, even, lst;
    string S;
    int newNode(int l) {
        fail.push_back(0);
        nxt.push_back({});
        len.push_back(l);
        dep.push_back(0);
        return fail.size() - 1;
    }
    PAM() : odd(newNode(-1)), even(newNode(0)) {
        lst = fail[even] = odd;
    }
    void reserve(int l) {
        fail.reserve(l + 2);
        len.reserve(l + 2);
        nxt.reserve(l + 2);
        dep.reserve(l + 2);
        walk.reserve(l);
    }
    void build(string_view s) {
        reserve(s.size());
        for (char c : s) {
            walk.push_back(add(c));
        }
    }
    int up(int p){

```

```

        while (S.rbegin()[len[p] + 1] != S.back()) {
            p = fail[p];
        }
        return p;
    }
    int add(char c) {
        S += c;
        lst = up(lst);
        c -= 'a';
        if (!nxt[lst][c]) {
            nxt[lst][c] = newNode(len[lst] + 2);
        }
        int p = nxt[lst][c];
        fail[p] = (lst == odd ? even : nxt[up(fail[lst])][c]);
        lst = p;
        dep[lst] = dep[fail[lst]] + 1;
        return lst;
    }
};

```

7.8 Suffix Automaton

```

struct SAM {
    vector<array<int, 26>> nxt;
    vector<int> fail, len;
    int lst = 0;
    int newNode() {
        fail.push_back(0);
        len.push_back(0);
        nxt.push_back({});
        return fail.size() - 1;
    }
    SAM() : lst(newNode()) {}
    void reset() {
        lst = 0;
    }
    int add(int c) {
        if (nxt[lst][c] and len[nxt[lst][c]] == len[lst] + 1) {
            return lst = nxt[lst][c];
        }
        int cur = newNode();
        len[cur] = len[lst] + 1;
        while (lst and nxt[lst][c] == 0) {
            nxt[lst][c] = cur;
            lst = fail[lst];
        }
        int p = nxt[lst][c];
        if (p == 0) {
            fail[cur] = 0;
            nxt[0][c] = cur;
        } else if (len[p] == len[lst] + 1) {
            fail[cur] = p;
        } else {
            int t = newNode();
            nxt[t] = nxt[p];
            fail[t] = fail[p];
            len[t] = len[lst] + 1;
            while (nxt[lst][c] == p) {
                nxt[lst][c] = t;
                lst = fail[lst];
            }
            fail[p] = fail[cur] = t;
        }
        return lst = cur;
    }
    vector<int> order() { // 長度遞減
        vector<int> cnt(len.size());
        for (int i = 0; i < len.size(); i++)
            cnt[len[i]]++;
        partial_sum(rall(cnt), cnt.rbegin());
        vector<int> ord(cnt[0]);
        for (int i = len.size() - 1; i >= 0; i--)
            ord[--cnt[len[i]]] = i;
        return ord;
    }
};

```

7.9 Lyndon Factorization

// 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

```
// min rotate: last < n of duval_min(s + s)
// max rotate: last < n of duval_max(s + s)
// min suffix: last of duval_min(s)
// max suffix: last of duval_max(s + -1)
vector<int> duval(const auto &s) {
    int n = s.size(), i = 0;
    vector<int> pos;
    while (i < n) {
        int j = i + 1, k = i;
        while (j < n and s[k] <= s[j]) { // >=
            if (s[k] < s[j]) k = j; // >
            else k++;
            j++;
        }
        while (i <= k) {
            pos.push_back(i);
            i += j - k;
        }
    }
    pos.push_back(n);
    return pos;
}
```

7.10 SmallestRotation

```
string Rotate(const string &s) {
    int n = s.length();
    string t = s + s;
    int i = 0, j = 1;
    while (i < n && j < n) {
        int k = 0;
        while (k < n && t[i + k] == t[j + k]) ++k;
        if (t[i + k] <= t[j + k]) j += k + 1;
        else i += k + 1;
        if (i == j) ++j;
    }
    int pos = (i < n ? i : j);
    return t.substr(pos, n);
}
```

8 Misc

8.1 Fraction Binary Search

```
// Binary search on Stern-Brocot Tree
// Parameters: n, pred
// n: Q_n is the set of all rational numbers whose
//     denominator does not exceed n
// pred: pair<i64, i64> -> bool, pred({0, 1}) must be
//     true
// Return value: {{a, b}, {x, y}}
// a/b is bigger value in Q_n that satisfy pred()
// x/y is smaller value in Q_n that not satisfy pred()
// Complexity: O(log^2 n)
using Pt = pair<i64, i64>;
Pt operator+(Pt a, Pt b) { return {a.ff + b.ff, a.ss + b.ss}; }
Pt operator*(i64 a, Pt b) { return {a * b.ff, a * b.ss}; }
pair<pair<i64, i64>, pair<i64, i64>> FractionSearch(i64
n, const auto &pred) {
    pair<i64, i64> low{0, 1}, hei{1, 0};
    while (low.ss + hei.ss <= n) {
        bool cur = pred(low + hei);
        auto &fr{cur ? low : hei}, &to{cur ? hei : low};
        u64 L = 1, R = 2;
        while ((fr + R * to).ss <= n and pred(fr + R * to)
== cur) {
            L *= 2;
            R *= 2;
        }
        while (L + 1 < R) {
            u64 M = (L + R) / 2;
            ((fr + M * to).ss <= n and pred(fr + M * to) ==
cur ? L : R) = M;
        }
        fr = fr + L * to;
    }
    return {low, hei};
}
```

8.2 de Bruijn sequence

```
constexpr int MAXC = 10, MAXN = 1e5 + 10;
struct DBSeq {
    int C, N, K, L;
    int buf[MAXC * MAXN];
    void dfs(int *out, int t, int p, int &ptr) {
        if (ptr >= L) return;
        if (t > N) {
            if (N % p) return;
            for (int i = 1; i <= p && ptr < L; ++i)
                out[ptr++] = buf[i];
        } else {
            buf[t] = buf[t - p], dfs(out, t + 1, p, ptr);
            for (int j = buf[t - p] + 1; j < C; ++j)
                buf[t] = j, dfs(out, t + 1, t, ptr);
        }
    }
    void solve(int _c, int _n, int _k, int *out) { //
        alphabet, len, k
        int p = 0;
        C = _c, N = _n, K = _k, L = N + K - 1;
        dfs(out, 1, 1, p);
        if (p < L) fill(out + p, out + L, 0);
    }
} dbs;
```

8.3 HilbertCurve

```
i64 hilbert(int n, int x, int y) {
    i64 pos = 0;
    for (int s = (1 << n) / 2; s; s /= 2) {
        int rx = (x & s) > 0;
        int ry = (y & s) > 0;
        pos += 1LL * s * s * ((3 * rx) ^ ry);
        if (ry == 0) {
            if (rx == 1) x = s - 1 - x, y = s - 1 - y;
            swap(x, y);
        }
    }
    return pos;
}
```

8.4 Grid Intersection

```
int det(Pt a, Pt b) { return a.ff * b.ss - a.ss * b.ff; }
// find p s.t (d1 * p, d2 * p) = x
Pt gridInter(Pt d1, Pt d2, Pt x) {
    swap(d1.ss, d2.ff);
    int s = det(d1, d2);
    int a = det(x, d2);
    int b = det(d1, x);
    assert(s != 0);
    if (a % s != 0 or b % s != 0) {
        return {-1, -1};
    }
    return {a / s, b / s};
}
```

8.5 NextPerm

```
i64 next_perm(i64 x) {
    i64 y = x | (x - 1);
    return (y + 1) | (((~y & ~y) - 1) >> (__builtin_ctz(
x) + 1));
}
```

8.6 Python FastIO

```
import sys
sys.stdin.readline()
sys.stdout.write()
```

8.7 HeapSize

```
pair<i64, i64> Split(i64 x) {
    if (x == 1) return {0, 0};
    i64 h = __lg(x);
    i64 fill = (1LL << (h + 1)) - 1;
    i64 l = (1LL << h) - 1 - max(0LL, fill - x - (1LL <<
(h - 1)));
    i64 r = x - 1 - l;
    return {l, r};
}
```